А. В. Микушин, А. М. Сажнев, В. И. Сединин

Цифровые устройства и микропроцессоры

bhv

УДК 681.3.06(075.8)
ББК 32.973.26-04я73
М59

Микушин, А. В.

М59 Цифровые устройства и микропроцессоры: учеб. пособие / А. В. Микушин, А. М. Сажнев, В. И. Сединин. — СПб.: БХВ-Петербург, 2010. — 832 с.: ил. — (Учебная литература для вузов)

ISBN 978-5-9775-0417-1

Книга представляет собой учебник по курсу с одноименным названием, который читается авторами в течение многих лет студентам радиотехнических специальностей в Сибирском государственном университете телекоммуникаций и информатики и Новосибирском государственном техническом университете. Даны основы проектирования цифровых устройств с упором на создание принципиальных схем устройств связи. Рассмотрены вопросы аналого-цифрового преобразования и обработки сигналов, в частности, случаи изменения частоты дискретизации цифрового сигнала и узлы, позволяющие изменять эту частоту. В качестве примеров цифровых устройств рассмотрены такие современные устройства, как схемы прямого цифрового синтеза DDS, цифрового преобразования частоты вверх DUC, цифрового понижения частоты приёма DDC. Изложены основы микропроцессорной техники и особенности работы микроконтроллеров на примере семейства MCS-51. Даны основы программирования для микроконтроллеров на языках С и ассемблер. Две основные части курса иллюстрируются примером разработки одного и того же устройства — часов, на цифровых микросхемах и на микроконтроллере.

Для студентов, инженеров и специалистов радиотехнических специальностей

УДК 681.3.06(075.8)
ББК 32.973.26-04я73

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Евгений Рыбаков*
Зав. редакцией	*Григорий Добин*
Редактор	*Алиса Тяжбина*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Елены Беляевой*
Фото	*Кирилла Сергеева*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 29.01.10.
Формат 70×100$^{1}/_{16}$. Печать офсетная. Усл. печ. л. 67,08.
Тираж 1500 экз. Заказ № 3034
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.60.953.Д.005770.05.09 от 26.05.2009 г. выдано Федеральной службой по надзору в сфере защиты прав потребителей и благополучия человека

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0417-1

Оглавление

Введение

Цифровые микросхемы первоначально разрабатывались для построения электронно-вычислительных машин, получивших в дальнейшем название компьютеры. То есть первое их предназначение было заменить человека при выполнении рутинной вычислительной работы. Сейчас, наверное, никто и не вспомнит, что слово "калькулятор" еще каких-нибудь шестьдесят лет назад обозначало не маленький карманный прибор, а профессию большого числа людей, которые занимались расчетами по заданным математическим формулам.

Однако вскоре после начала массового производства цифровых микросхем выяснилось, что они очень удобны для управления какими-либо объектами. При этом управляемая схема обычно может находиться только в двух состояниях. Например, схема может быть включена или выключена, светодиод может светиться или не светиться, соединение в телефонной станции может присутствовать или отсутствовать, радиостанция может находиться в режиме передачи или приема. Это означает, что большинство технических устройств прекрасно описываются (и управляются) двоичными сигналами. При выполнении задачи управления для описания состояния объекта достаточно двух значений: напряжение высокое или низкое (положительное или отрицательное), ток протекает или не протекает.

Это свойство цифровых сигналов позволило избавиться от многих неприятных моментов аналоговых схем. Например, ошибка при прохождении через цифровую схему не увеличивается (в отличие от шумов аналоговых схем), а в ряде случаев даже может быть исправлена. Сами цифровые схемы при правильном их применении не вносят ошибок. Эти свойства цифровых микросхем привели к бурному развитию цифровой техники.

Кроме перечисленных достоинств цифровые микросхемы при массовом производстве оказались чрезвычайно дешевы, а вскоре превзошли другие техни-

ческие решения и по габаритам, и по массе. В результате цифровые микросхемы практически полностью вытеснили применявшиеся еще с XIX века для управления приборами электромагнитные реле и перфокарты. Использование цифровых микросхем резко повысило надежность устройств управления объектами.

Приведенные выше преимущества цифровых микросхем привели к тому, что в дальнейшем цифровая техника стала использоваться для решения и других задач. Например, для формирования высокостабильных колебаний в радиотехнических изделиях или в качестве эталонных интервалов времени в электронных и электромеханических часах. В этих устройствах, как и в устройствах управления, не стоит задача формирования сигнала строго определенной формы. Единственным условием является стабильность частоты генерируемого колебания. В результате в современном мире полностью изменилась технология изготовления генераторного оборудования и часовая промышленность.

С течением времени стали разрабатываться методы и теория применения цифровых микросхем для формирования аналоговых сигналов. И здесь тоже основным фактором была возможность заранее рассчитывать уровень шумов устройства. При этом уровень шума зависит только от сложности схемы и не зависит (ну, или почти не зависит) от количества схем, через которые проходит сигнал. Эта особенность приводит к возможности передавать сигнал на любое расстояние (или производить любое количество копий записанного сигнала).

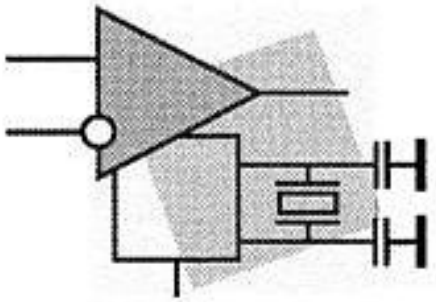
Постоянный прогресс в технологии производства цифровых микросхем позволяет снижать потребление энергии этими микросхемами и увеличивать сложность алгоритмов обработки сигналов. В результате область применения цифровых методов обработки аналоговых сигналов постоянно расширяется, как в область все более высоких частот, так и в области, ранее не охватываемые радиотехникой (например, цифровая фотография).

Отметим, что уровни логических сигналов не уменьшаются при распространении сигнала по цифровой схеме. Это означает, что цифровые микросхемы принципиально должны обладать усилением. В то же самое время логические уровни на выходе цифрового устройства точно такие же, как и на входе, т. е. они не возрастают при прохождении через логический элемент. Это обеспечивается тем, что на выходе цифровой микросхемы происходит ограничение сигнала, т. е. цифровые микросхемы работают в ключевом режиме: транзистор цифровой микросхемы может быть только открыт или закрыт. В результате на идеальном транзисторе рассеивания энергии не происходит ни в том, ни в другом состоянии. Это означает, что в цифровых микросхемах можно достичь коэффициента полезного действия близкого к 100%.

Изучение цифровой техники начнем с самых элементарных вопросов: из каких элементов состоят цифровые схемы и как они устроены? Затем научимся реализовывать на основе этих простейших элементов цифровые устройства любой сложности. Для этого нам потребуется изучить основы алгебры логики и методы запоминания цифровых сигналов. Мы научимся отображать цифровую информацию и вводить ее в цифровые микросхемы.

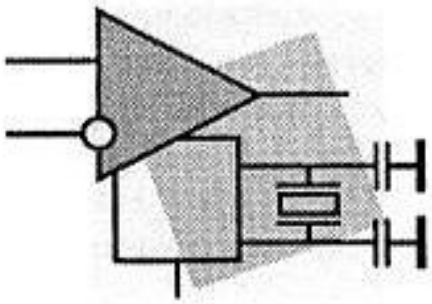
Для самостоятельного изучения (при наличии доступа в Интернет) можно посмотреть в качестве примера разработки цифрового устройства материалы сайта **http://digital.sibsutis.ru/MCU/strsxustr.htm** или **http://digital.sibsutis.ru /PrimerKP.zip**. При изучении первой части курса студенты СибГУТИ выполняют лабораторные работы, методические указания к которым присутствуют на сайте **http://www.labfor.ru/?act=metod**. Лабораторные работы можно выполнить дистанционно на странице **http://www.labfor.ru/?act= labs&target=lab_mk**. Эти работы могут помочь читателю освоить основы цифровой техники и познакомиться с проектированием цифровых устройств на ПЛИС.

В дальнейшем мы научимся создавать цифровые устройства с применением микропроцессорной техники, которую будем изучать на основе самого распространенного семейства однокристальных ЭВМ — MCS-51. В процессе работы с микропроцессорными устройствами нам потребуется среда программирования, которую можно скачать с сайта фирмы keil **http://www.keil.com /demo/eval/c51.htm** или с сайта автора книги **http://digital.sibsutis.ru /MP/KP_cu_mp.htm**. Пример проектирования микропроцессорного устройства (электронных часов) приведен на странице **http://digital.sibsutis.ru /MCU/strsxustr.htm**.



ЧАСТЬ I

Основы цифровой техники



ГЛАВА 1

Параметры цифровых микросхем

Цифровые микросхемы или микросборки, их элементы или компоненты обозначаются на принципиальных схемах условно-графическим обозначением в соответствии с ГОСТ 2.743-91. Условно-графическое обозначение микросхемы имеет форму прямоугольника, к которому подводят линии выводов. Оно может содержать три поля: основное и два дополнительных, которые располагают слева и справа от основного (рис. 1.1). В первой строке основного поля помещают обозначение функции, выполняемой элементом. В последующих строках основного поля располагают информацию по ГОСТ 2.743-91.

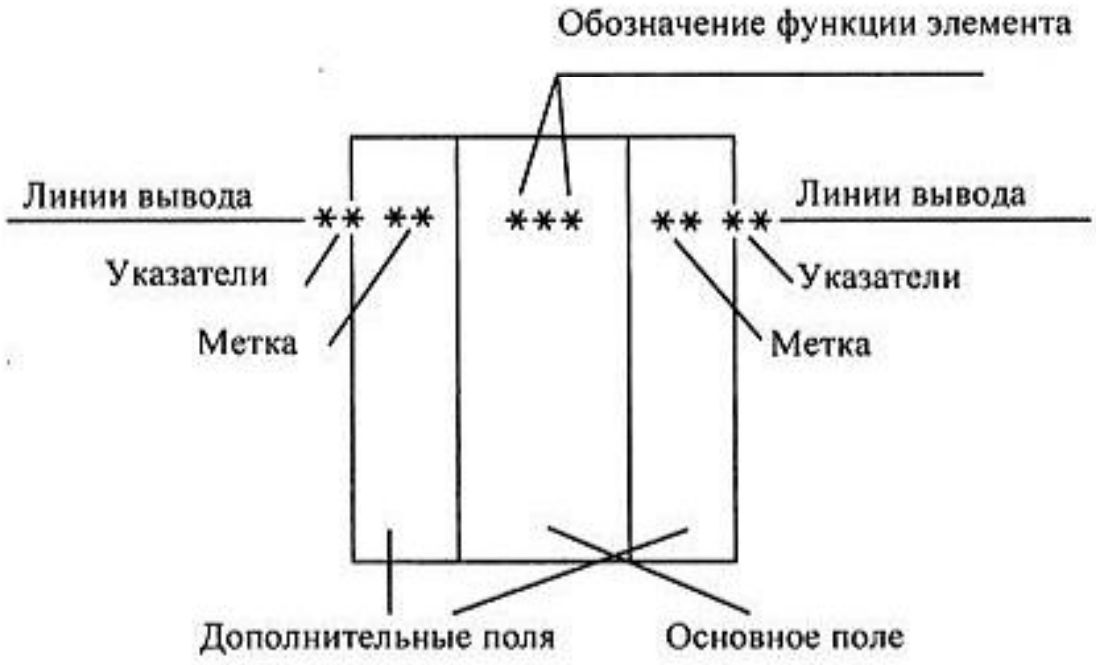


Рис. 1.1. Условно-графическое изображение цифровых микросхем

В дополнительных полях помещают информацию о назначениях выводов (метки выводов, указатели). Дополнительные поля на условно-графическом изображении цифровых микросхем могут отсутствовать. Входы на условно-графическом изображении цифровых микросхем располагают слева, а выходы — справа. Номера выводов микросхем помещают над линией вывода ближе к изображению микросхемы.

Точно так же, как и аналоговые схемы, цифровые схемы должны описываться некоторыми параметрами. Аналоговые схемы характеризуются напряжением питания, при котором они могут работать. Цифровые микросхемы тоже обладают этим параметром. В настоящее время наиболее распространены цифровые микросхемы с напряжением питания +5 В и +3,3 В, однако существуют микросхемы, способные работать в диапазоне напряжений от 2 до 6 В. Минимальное напряжение питания, при котором работают современные цифровые микросхемы составляет 0,7 В.

Уровни логического нуля и единицы

Как уже упоминалось ранее, цифровые микросхемы характеризуются тем, что могут находиться только в двух состояниях. Состояния цифровых микросхем могут быть описаны двумя цифрами: "0" и "1". При этом состояние микросхемы можно характеризовать различными параметрами. Например, током или напряжением в цепях микросхемы, открыты или закрыты транзисторы на выходе, светится или нет светодиод (если он входит в состав микросхемы).

В качестве логических состояний цифровых микросхем условились воспринимать напряжение на их входе и выходе. При этом высокое напряжение считается единицей, а низкое напряжение — нулем. В идеальном случае напряжение на выходе микросхем должно быть равным напряжению питания или нулевым потенциалом общего провода схемы.

В реальных схемах так не бывает. Даже на полностью открытом реальном транзисторе есть определенное падение напряжения. В результате на выходе цифровой микросхемы напряжение всегда будет меньше напряжения питания или больше потенциала общего провода. Поэтому в реальных схемах напряжение, меньшее заданного уровня (уровень логического нуля), считается нулем, а напряжение, большее заданного уровня (уровень логической единицы), считается единицей. Если же напряжение на выходе микросхемы будет больше уровня логического нуля, но меньше уровня логической единицы, то такое состояние микросхемы называется неопределенным.

На рис. 1.2 приведены уровни выходных логических сигналов, допустимые для цифровых ТТЛ-микросхем. ✧ *Обратите внимание*, что чем ближе вы-

ходное напряжение к напряжению питания или к напряжению общего провода схемы, тем выше КПД цифровой микросхемы.

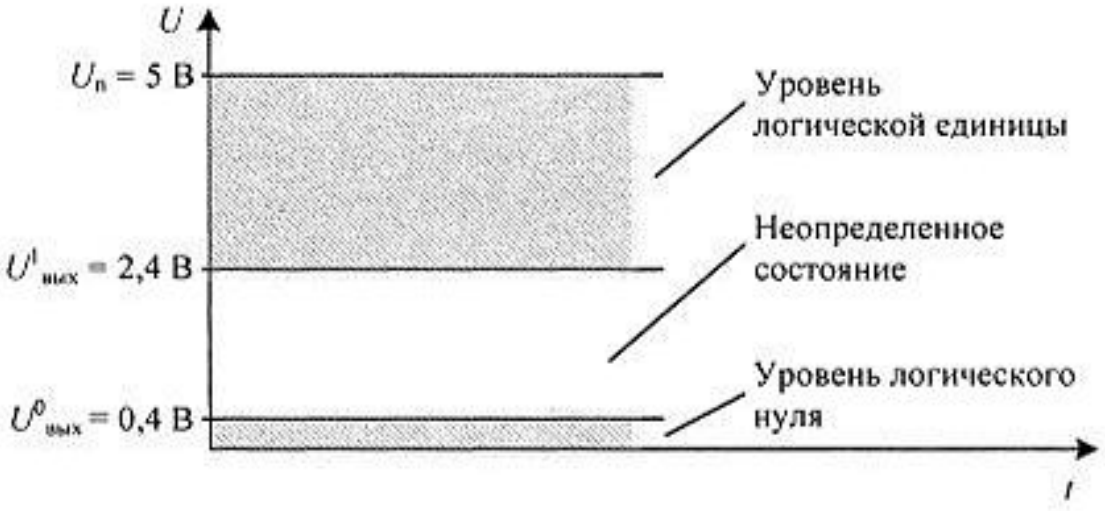


Рис. 1.2. Уровни логических сигналов на выходе цифровых ТТЛ-микросхем

В реальных схемах напряжение с выхода одной микросхемы передается на вход другой микросхемы по проводникам. В процессе передачи на эти проводники может наводиться напряжение или ток от каких-либо генераторов помех (осветительная сеть, радиопередатчики, генераторы импульсных сигналов и т. п.). В этом случае помехоустойчивость цифровых микросхем определяется максимальным напряжением помех, которое не приводит к превращению логической единицы в логический ноль, и она зависит от разности логического уровня цифровой микросхемы-источника и логического уровня микросхемы-приемника. Это напряжение можно определить из следующих выражений:

$$U^-_{пом} = \left| U_{Вых1min} - U_{Вх1min} \right|$$

То же самое относится и к помехам, превращающим логический ноль в логическую единицу. Обозначим такую помеху $U^+_{пом}$. Эта помеха определяется из следующего выражения:

$$U^+_{пом} = \left| U_{Вых0min} - U_{Вх0min} \right|$$

Чем меньше разница между $U_{Вх1min}$ и $U_{Вх0max}$, тем большим усилением обладает цифровая микросхема. Типовое усиление ТТЛ-микросхем по напряжению K_u составляет 40 раз. Это приводит к тому, что, подав на вход этой микросхемы напряжение, на 15 мВ меньшее уровня $U_{пор}$, мы воспримем его как логический ноль, и на выходе этой микросхемы получим нормальный логический уровень. При подаче на вход ТТЛ-микросхемы напряжения, на 15 мВ

большего уровня $U_{пор}$, это напряжение будет восприниматься как логическая единица. Данное рассуждение иллюстрируется следующей формулой:

$$U_{вых} = K_u \times U_{вх} = 15 \times 40 \text{ мВ} = 0{,}6 \text{ В}$$

Если теперь из порогового напряжения вычесть 0,6 В, то мы получим 0,8 В — уровень нормального логического нуля. А если к нему прибавить 0,6 В, то получим 2 В — минимальный уровень логической единицы. ✧ *Обратите внимание, что описанная ситуация предполагает отсутствие помех.*

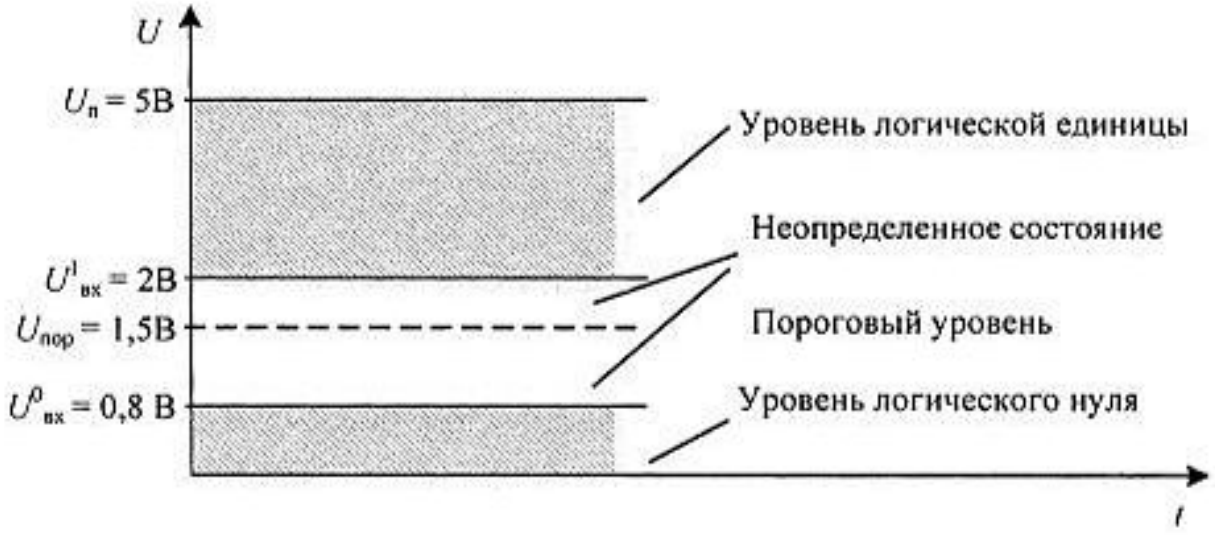


Рис. 1.3. Уровни логических сигналов на входе цифровых ТТЛ-микросхем

На рис. 1.3 приведены входные уровни логического нуля и логической единицы ТТЛ-микросхем. На этом же рисунке представлен пороговый уровень входного сигнала (граница уровней логического нуля и единицы). Этот уровень немного смещен от середины защитного интервала, т. к. переходная характеристика биполярных транзисторов нелинейна.

✧ *Обратите внимание*, что входные уровни логических сигналов отличаются от выходных. Это связано с тем, что микросхемы могут работать при воздействии неблагоприятных факторов, таких как пониженная температура, старение микросхем, воздействие радиации. На цепи питания и общего провода могут наводиться помехи. Поэтому производители микросхем гарантируют срабатывание микросхем с некоторым запасом. Например, фирма Texas Instruments объявляет для своих микросхем входной уровень единицы — 2 В, а уровень нуля — 0,8 В.

Теперь можно определить уровень помехи, который не будет восприниматься цифровой микросхемой. В качестве примера выберем ТТЛ-микросхемы. Мы уже знаем, что на выходе цифровой ТТЛ-микросхемы уровень логической единицы не может быть меньше 2,4 В, а уровень логического нуля не может быть больше 0,4 В. То есть напряжение помех, гарантированное производи-

телем, будет $U_{пом} = 2{,}4 - 2 = 0{,}8 - 0{,}4 = 0{,}4$ В. В реальности, даже при наведении на вход ТТЛ-микросхемы помехи, напряжением $U_{пом} = U^1 - U_{пор} = 2{,}4 - (1.5 + 0{,}04) = 0{,}86$ вольт, искажения цифровой информации не произойдет.

А что же произойдет, если напряжение на входе цифровой микросхемы будет близко к порогу, разделяющему уровень логического нуля и логической единицы? В этом случае микросхема перейдет в активный (усилительный) режим работы, и оба выходных транзистора могут оказаться открытыми. В результате микросхема может выйти из строя из-за перегрева. Поэтому *входы цифровых (особенно КМОП) микросхем ни в коем случае не должны быть оставлены неподключенными!* Если часть элементов цифровой микросхемы не используется, то их входы должны быть подключены к источнику питания или общему проводу схемы. Конкретная точка подключения неиспользуемого входа микросхемы будет подробно обсуждаться позднее. Соотношение уровней логических сигналов на выходе и входе логических микросхем приведено на рис. 1.4.

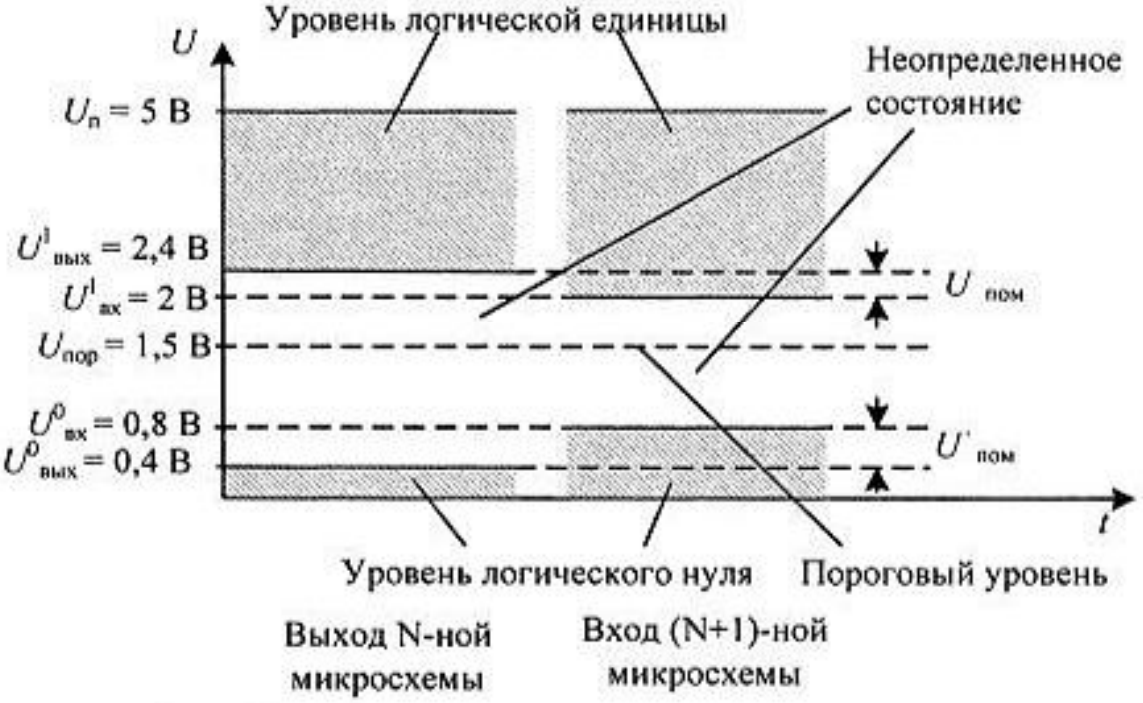


Рис. 1.4. Соотношение уровней логических сигналов на выходе и входе цифровых ТТЛ-микросхем

В заключение следует обратить внимание на то, что конкретное значение порога переключения для различных экземпляров и серий микросхем может изменяться в некоторых пределах. Это еще одна причина, по которой нельзя подавать на вход логических микросхем напряжение в пределах неопределенного состояния или оставлять входы микросхем неподключенными.

Входные и выходные токи цифровых микросхем

Еще один важный параметр любой микросхемы — это предельно допустимый выходной ток. Для цифровых микросхем есть два различных значения выходного тока: ток единицы (высокого потенциала) и ток нуля (низкого потенциала). В цифровых микросхемах эти значения могут быть различными. Путь протекания тока единицы в цифровых микросхемах показан на рис. 1.5.

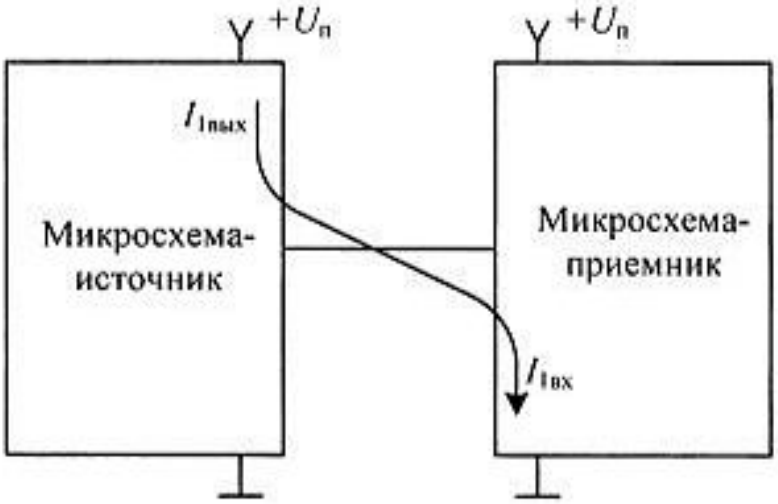


Рис. 1.5. Путь протекания выходного тока единицы в цифровых микросхемах

На этом рисунке видно, что в простейшем случае выходной ток цифровой микросхемы $I_{1вых}$ (вытекающий ток) совпадает с входным током единицы $I_{1вх}$ нагрузочной цифровой микросхемы (микросхемы-приемника). Часто требуется подавать сигнал с выхода одной микросхемы на несколько других микросхем. В этом случае выходной ток микросхемы будет определяться как сумма входных токов микросхем-приемников. Количество однотипных микросхем, которые могут быть одновременно подключены к выходу микросхемы, определяет *предельную нагрузочную способность микросхемы*.

Путь протекания выходного тока нуля $I_{0вых}$ (втекающий ток) показан на рис. 1.6. В этом случае выходной ток нуля микросхемы-источника $I_{0вых}$ тоже определяется суммой входных токов нуля $I_{0вх}$ микросхем-приемников, подключенных к ее выходу.

Для того чтобы выход цифровой микросхемы мог нагружаться на входы нескольких микросхем-приемников, их входной ток должен быть меньше выходного тока микросхемы-источника. Для ТТЛ-микросхем нагрузочная способность составляет обычно 10. Для КМОП-микросхем она может достигать 100, т. е. на выход одной КМОП-микросхемы можно нагружать до сотни входов других КМОП-микросхем.

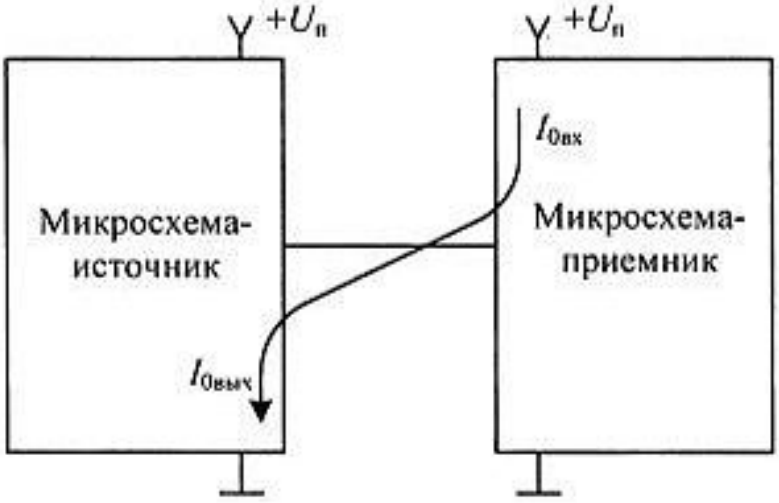


Рис. 1.6. Путь протекания выходного тока нуля в цифровых микросхемах

Параметры, определяющие быстродействие цифровых микросхем

Быстродействие цифровых микросхем определяется скоростями их перехода из одного состояния в другое. При этом быстродействие цифровой микросхемы определяется временем задержки выходного сигнала относительно входного. Не следует путать это время с длительностью фронта выходного импульса цифровой микросхемы.

В общем случае длительность переднего (rising — нарастающего) и заднего (falling — спадающего) фронтов цифрового сигнала не совпадает. Длительность фронта определяется как время нарастания выходного сигнала от напряжения $0{,}1\times U$ до напряжения $0{,}9\times U$, где U — это разность напряжений между уровнем логической единицы и уровнем логического нуля. На рис. 1.7 длительность переднего (rising — нарастающий) фронта обозначена как $t_{ф01}$, а длительность заднего (falling — спадающий) фронта — как $t_{ф10}$.

Время задержки выходного сигнала относительно входного обычно больше длительности фронта выходного сигнала, и именно этот параметр приводится в качестве характеристики цифровой микросхемы, определяющей ее быстродействие. Это время может быть измерено по точке пересечения входным и выходным сигналами порогового уровня. В цифровых микросхемах время задержки переднего фронта и время задержки заднего фронта обычно не совпадают. Времена задержки t^{01} и t^{10} показаны на временной диаграмме входного и выходного сигнала цифровой микросхемы, приведенной на рис. 1.8.

Тем не менее, для того, чтобы можно было сравнивать цифровые микросхемы между собой, часто пользуются усредненным временем задержки сигнала

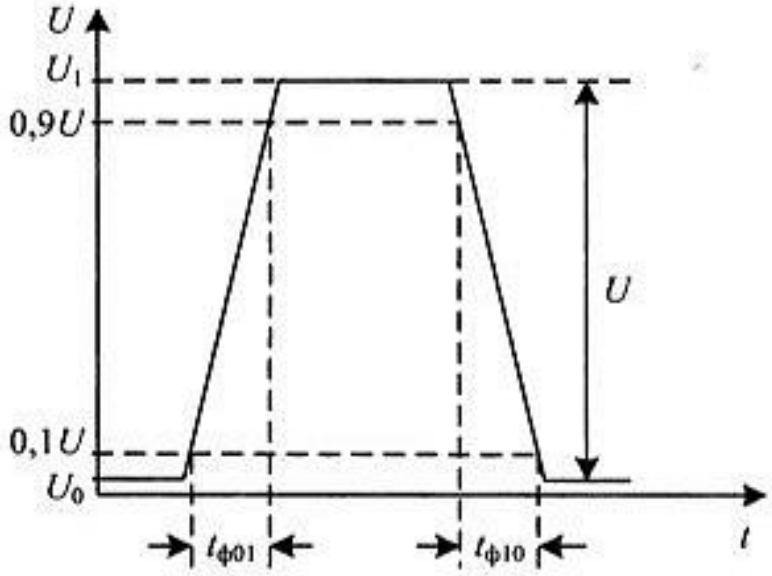


Рис. 1.7. Определение длительности переднего и заднего фронта выходного импульса

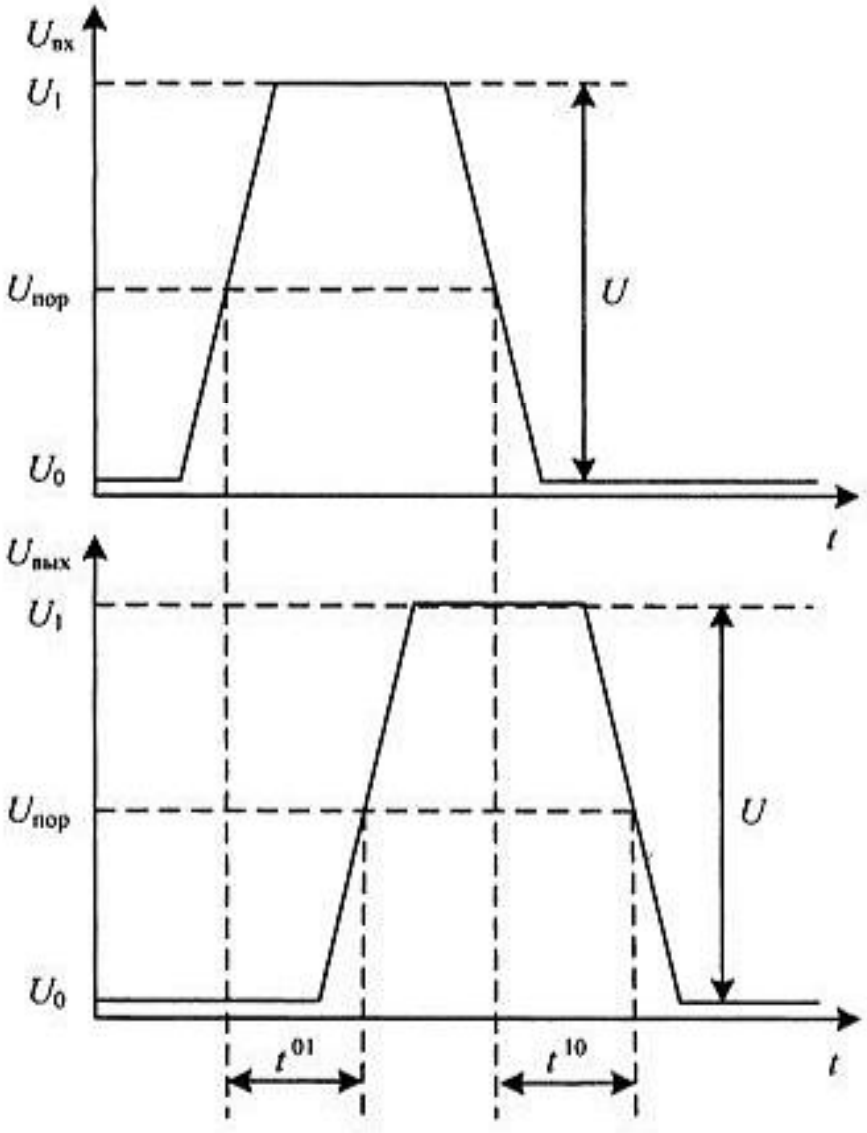


Рис. 1.8. Определение времени задержки цифровой микросхемы

цифровой микросхемой. Этот параметр определяется как половина суммы времен задержек t^{01} и t^{10}:

$$t_{зад} = \frac{t^{01} + t^{10}}{2}.$$

Описание логической функции цифровых схем

Как уже упоминалось ранее, особенностью использования цифровой техники является то, что при разработке схем можно отвлечься от особенностей практической реализации цифровых микросхем, а также от влияния конкретных значений выходного напряжения и токов нагрузки. При этом все внимание можно уделить разработке целевой логической функции, реализуемой цифровой схемой. Для этого входные и выходные сигналы цифровой схемы можно задать цифрами "0" и "1". При этом неважно, с использованием какой схемотехники (ТТЛ, КМОП или BiCMOS) будут реализованы сами логические элементы. Выбор конкретной схемотехники влияет только на эксплуатационные характеристики реализуемой аппаратуры, но совершенно не отразится на виде логической функции. Именно поэтому логические элементы, выполненные с использованием различных серий интегральных цифровых микросхем, изображаются на принципиальных схемах совершенно одинаково.

Выходные сигналы цифровых микросхем необходимо рассматривать в статическом режиме, когда все переходные процессы уже закончились, благодаря этому исключается влияние задержек распространения сигналов.

В простейших цифровых схемах выходные сигналы зависят только от входных сигналов, и не зависят от их значений в предыдущие моменты времени. Такие цифровые устройства получили название комбинационных цифровых устройств. Обычно такие устройства описываются при помощи таблицы истинности.

Процесс разработки цифрового устройства достаточно сильно алгоритмизирован. Это позволяет значительно сократить время разработки устройства и снизить вероятность возникновения ошибки на этапе проектирования устройства.

Разработка цифрового устройства разбивается на пять этапов:

1. Создание таблицы истинности.
2. Запись логического выражения.
3. Минимизация полученного логического выражения.

4. Создание схемы по минимизированному логическому выражению.
5. Реализация полученной схемы в заданном наборе микросхем.

Таблица истинности — это совокупность всех возможных комбинаций логических сигналов на входе цифрового устройства и значений выходных сигналов для каждой комбинации. Для того чтобы не пропустить ни одной комбинации входных сигналов, их обычно записывают в виде двоичного кода. Пример таблицы истинности приведен в табл. 1.1.

Таблица 1.1. *Таблица истинности цифровой микросхемы*

№ комбинации	вх1	вх2	вх3	вых1	вых2
0	0	0	0	0	0
1	0	0	1	0	1
2	0	1	0	0	1
3	0	1	1	1	0
4	1	0	0	0	1
5	1	0	1	1	0
6	1	1	0	1	0
7	1	1	1	1	1

Для описания принципов работы комбинационной цифровой схемы достаточно таблицы истинности. Этой же таблицы достаточно для создания ее принципиальной схемы.

Рассмотрим несколько примеров описания цифровой схемы с помощью таблицы истинности. Пусть требуется преобразовать двоично-десятичный код в десятичный. Поставим в соответствие каждой комбинации двоично-десятичного кода соответствующее десятичное значение (для отображения десятичных цифр необходимо использовать десять выходов схемы). Получившаяся таблица истинности содержится в табл. 1.2.

Составим еще одну таблицу истинности. На этот раз в качестве исходной постановки задачи используем временные диаграммы. В микропроцессорной технике достаточно часто приходится сталкиваться с задачей преобразования временных диаграмм системной шины (например, в процессорах фирмы Freescale, бывшая Motorola) во временные диаграммы системной шины, использующиеся в процессорах фирмы Intel. Эти временные диаграммы приведены на рис. 1.9. Сигналы DS и R/W соответствуют временным диаграммам системной шины процессоров фирмы Freescale, а инверсные сигналы RD

Таблица 1.2. *Таблица истинности десятичного дешифратора*

№ комбинации	Входы				Выходы									
	8	4	2	1	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	1	0	0	0	0	0	0	0	0
2	0	0	1	0	0	0	1	0	0	0	0	0	0	0
3	0	0	1	1	0	0	0	1	0	0	0	0	0	0
4	0	1	0	0	0	0	0	0	1	0	0	0	0	0
5	0	1	0	1	0	0	0	0	0	1	0	0	0	0
6	0	1	1	0	0	0	0	0	0	0	1	0	0	0
7	0	1	1	1	0	0	0	0	0	0	0	1	0	0
8	1	0	0	0	0	0	0	0	0	0	0	0	1	0
9	1	0	0	1	0	0	0	0	0	0	0	0	0	1

и WR — временным диаграммам системной шины процессоров, совместимых с процессорами фирмы Intel.

Преобразуем приведенные временные диаграммы в таблицу истинности. Для этого проведем вертикальное сечение этих диаграмм. Видим, что в начале временных диаграмм сигналы обеих системных шин соответствуют единичным потенциалам (DS = 1, R/W = 1, WR = 1, RD = 1). Записываем это состояние в таблицу истинности. Оно соответствует отсутствию каких-либо операций в системной шине. Следующая комбинация соответствует подготовке к операции записи в системной шине процессоров фирмы Freescale (DS = 1, R/W = 0). В системной шине процессоров фирмы Intel нет аналога, поэтому заменим комбинацией отсутствия операций WR = 1, RD = 1 (то же самое мы видим и по сигналам, приведенным на временных диаграммах). Аналогичным образом запишем в таблицу истинности потенциалы сигналов, соответствующие операциям записи и чтения системной памяти. В результате описанных действий мы получили таблицу истинности для схемы преобразования сигналов системных шин двух разных микропроцессоров (табл. 1.3).

Теперь эта таблица истинности может быть использована для записи логического выражения и создания принципиальной схемы перекодировки сигналов записи/чтения, использующихся в системных шинах разных производителей микросхем.

Таблица 1.3. *Таблица истинности схемы перекодировки*

№ комбинации	Входы		Выходы		Операция
	DS	R/W	WR	RD	
0	0	0	0	1	запись
1	0	1	1	0	чтение
2	1	0	1	1	—
3	1	1	1	1	—

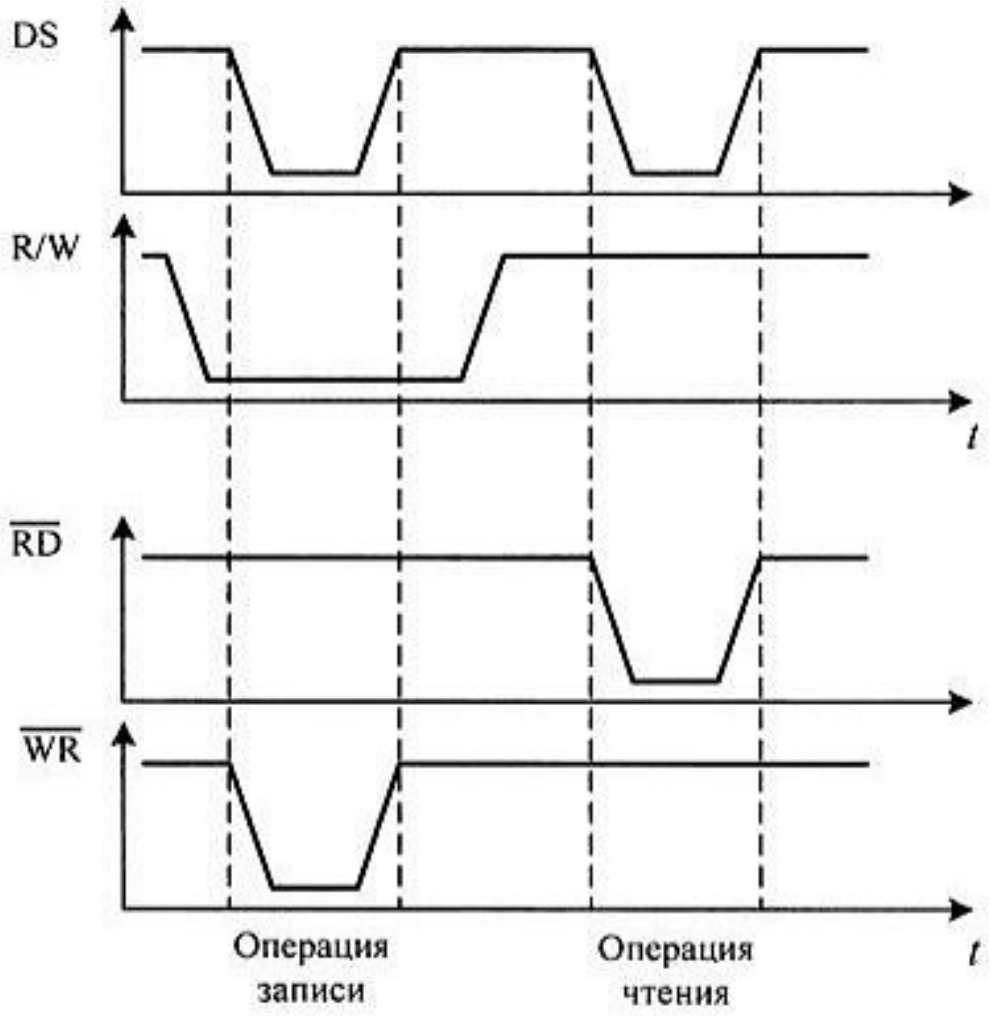
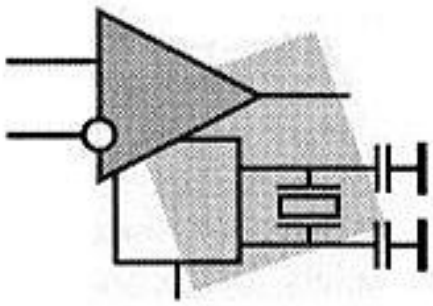


Рис. 1.9. Пример временных диаграмм цифрового устройства

Итоги

На этом можно закончить рассмотрение основных параметров цифровых микросхем. Полученных данных достаточно для того, чтобы начать работу с цифровыми микросхемами. Хотелось бы обратить внимание, что в последующих главах мы не будем касаться рассмотренных в данной главе вопросов. Однако при реальном проектировании цифровых устройств необходимо всегда учитывать рассмотренные параметры микросхем.

ГЛАВА 2



Основные логические функции и элементы

Практически все цифровые устройства без памяти (комбинаторные устройства) могут быть построены на основе трех простейших логических элементов. Все более сложные схемы реализуются из этих простейших логических элементов как из кубиков. Методы синтеза цифровых схем мы рассмотрим позднее, а пока остановимся подробнее на принципах работы и методах описания простейших логических элементов.

Функция "НЕ", инвертор

Простейшим логическим элементом является инвертор, который просто изменяет значение входного сигнала на прямо противоположное значение. Его функция записывается в следующем виде:

$$F(x) = \overline{x},$$

где черта над входным значением обозначает изменение его на противоположное. То же самое действие можно записать при помощи таблицы истинности, приведенной в табл. 2.1. Так как вход у этого логического элемента лишь один, его таблица истинности состоит только из двух строк.

Таблица 2.1. *Таблица истинности логического инвертора*

x	F
0	1
1	0

В качестве инвертора в простейшем случае можно использовать обычный усилитель с транзистором, включенным по схеме с общим эмиттером или

истоком. Схема усилителя, выполненная на биполярном n-p-n транзисторе и позволяющая реализовать функцию логического инвертирования, приведена на рис. 2.1.

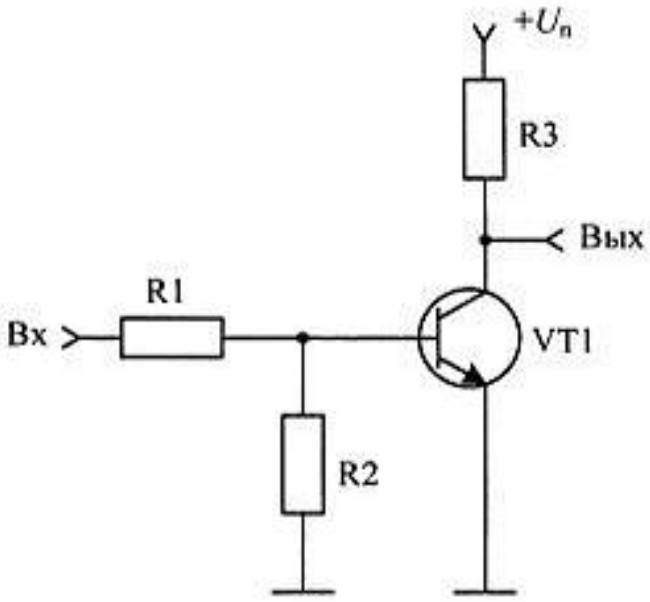


Рис. 2.1. Схема, позволяющая реализовать функцию логического инвертирования

Схемы инверторов могут обладать различным временем распространения сигнала и могут работать на различные виды нагрузки. Они могут быть выполнены на одном или на нескольких транзисторах, но независимо от схемы и ее параметров они осуществляют одну и ту же логическую функцию.

Для того чтобы особенности включения транзисторов не затеняли выполняемую функцию, для цифровых микросхем введены специальные условно-графические обозначения. Условно-графическое изображение инвертора приведено на рис. 2.2.

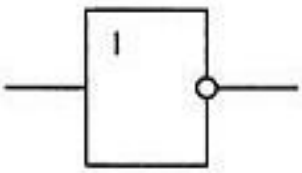


Рис. 2.2. Условно-графическое изображение логического инвертора

Функция "И", логическое умножение

Следующим простейшим логическим элементом является схема, реализующая операцию логического умножения "И":

$$F(x_1, x_2) = x_1 \wedge x_2,$$

где символ ^ обозначает функцию логического умножения (конъюнкцию). Иногда эта же функция записывается в другом виде:

$$F(x_1, x_2) = x_1 \wedge x_2 = x_1 \cdot x_2 = x_1 \,\&\, x_2 .$$

То же самое действие можно записать при помощи таблицы истинности, приведенной в табл. 2.2. В формуле, приведенной выше, использовано два аргумента. Поэтому элемент, выполняющий эту функцию, имеет два входа. Такой элемент обозначается "2И". Для элемента "2И" таблица истинности будет состоять из четырех строк. Количество строк таблицы истинности можно определить по формуле $N = 2^n$, где N — это количество строк в таблице истинности, а n — количество входов логического элемента. В нашем случае $N = 2^2 = 4$.

Таблица 2.2. *Таблица истинности схемы, выполняющей логическую функцию "2И"*

x_1	x_2	F
0	0	0
0	1	0
1	0	0
1	1	1

Как видно из приведенной таблицы истинности, активный сигнал на выходе этого логического элемента появляется только тогда, когда и на входе x_1 и на входе x_2 будут присутствовать логические единицы. То есть этот логический элемент действительно реализует операцию "И".

Условно-графическое изображение схемы, выполняющей логическую функцию "2И", на принципиальных схемах приведено на рис. 2.3, и с этого момента схемы, выполняющие функцию "И", будут приводиться именно в таком виде. Это изображение не зависит от конкретной принципиальной схемы устройства, реализующей функцию логического умножения.

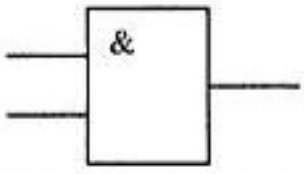


Рис. 2.3. Условно-графическое изображение схемы, выполняющей логическую функцию "2И"

Проще всего понять, как работает такой элемент при помощи схемы, построенной на идеализированных ключах с электронным управлением, как это показано на рис. 2.4. В приведенной схеме ток будет протекать только тогда, когда оба ключа будут замкнуты, а значит, единичный уровень на выходе схемы появится только при подаче на ее вход двух логических единиц.

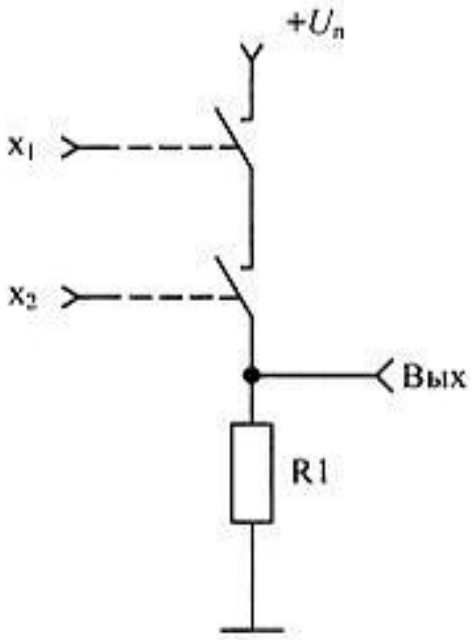


Рис. 2.4. Эквивалентная схема, реализующая логическую функцию "2И"

Аналогично описывается и функция логического умножения трех переменных:

$$F(x_1, x_2, x_3) = x_1 \wedge x_2 \wedge x_3.$$

Ее таблица истинности будет содержать уже восемь строк ($2^3 = 8$). Таблица истинности трехвходовой схемы логического умножения "3И" приведена в табл. 2.3, а условно-графическое изображение этого логического элемента на рис. 2.5. При этом в эквивалентной схеме, реализующей таблицу истинности и построенной по принципу схемы, приведенной на рис. 2.4, добавляется третий ключ.

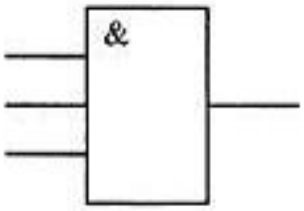


Рис. 2.5. Условно-графическое изображение схемы, выполняющей логическую функцию "3И"

Таблица 2.3. *Таблица истинности схемы, выполняющей логическую функцию "3И"*

x_1	x_2	x_3	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Функция "ИЛИ", логическое сложение

Следующим простейшим элементом является схема, реализующая операцию логического сложения "ИЛИ":

$$F(x_1, x_2) = x_1 \vee x_2,$$

где символ $\vee$ обозначает функцию логического сложения (дизъюнкцию). Иногда эта же функция записывается в другом виде:

$$F(x_1, x_2) = x_1 \vee x_2 = x_1 + x_2 = x_1 \mid x_2 .$$

То же самое действие можно записать при помощи таблицы истинности, приведенной в табл. 2.4. В формуле, показанной выше, использовано два аргумента. Поэтому элемент, выполняющий функцию логического сложения, имеет два входа. Такой элемент обозначается "2ИЛИ". Для элемента "2ИЛИ" таблица истинности будет состоять из четырех строк ($2^2 = 4$).

Таблица 2.4. *Таблица истинности схемы, выполняющей логическую функцию "2ИЛИ"*

x_1	x_2	F
0	0	0
0	1	1
1	0	1
1	1	1

Как и в случае, рассмотренном для схемы логического умножения, воспользуемся для реализации схемы логического элемента "2ИЛИ" идеализированными ключами с электронным управлением. На этот раз соединим ключи параллельно. Эквивалентная схема, реализующая таблицу истинности 2.4, приведена на рис. 2.6. Как видно из приведенной схемы, уровень логической единицы появится на ее выходе, как только будет замкнут любой из ключей.

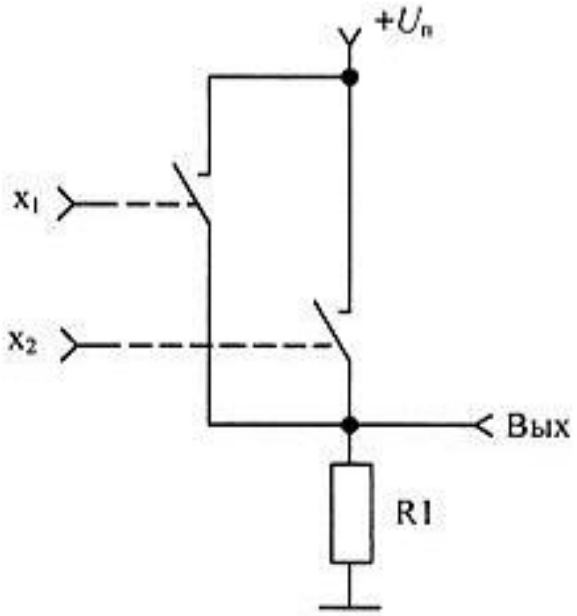


Рис. 2.6. Эквивалентная схема, реализующая логическую функцию "2ИЛИ"

Функция логического суммирования может быть реализована устройствами, собранными по разным принципиальным схемам. Тем не менее, независимо от конкретной схемы, реализуется одна и та же логическая функция, поэтому для обозначения устройства, реализующего логическую функцию "ИЛИ", используется свое условно-графическое обозначение. На условно-графическом изображении логического элемента "ИЛИ" используется специальный символ "1", как это приведено на рис. 2.7.

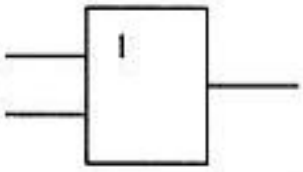


Рис. 2.7. Условно-графическое изображение схемы, выполняющей логическую функцию "2ИЛИ"

Подобным образом описывается и функция логического сложения трех переменных:

$$F(x_1, x_2, x_3) = x_1 \vee x_2 \vee x_3 .$$

Ее таблица истинности будет содержать уже восемь строк ($2^3 = 8$). Таблица истинности трехвходовой схемы логического сложения "ЗИЛИ" приведена в табл. 2.5, а условно-графическое изображение на рис. 2.8. В схеме, построенной по принципу схемы, приведенной на рис. 2.6, придется добавить третий ключ.

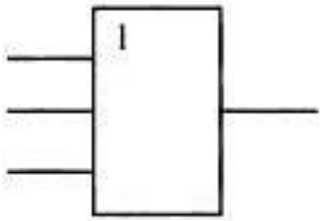


Рис. 2.8. Условно-графическое изображение схемы, выполняющей логическую функцию "ЗИЛИ"

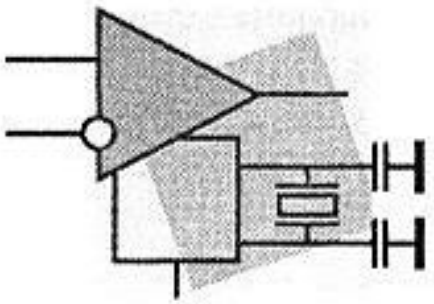
***Таблица 2.5.** Таблица истинности схемы, выполняющей логическую функцию "ЗИЛИ"*

x_1	x_2	x_3	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Теперь, после того как мы рассмотрели принципы построения логических элементов, можно подробно остановиться на особенностях их реализации в различных схемотехнических решениях.

Итоги

В данной главе были рассмотрены основные логические элементы цифровой схемотехники. Все цифровые устройства, как бы сложны они не были, строятся на основе этих элементов. Конкретная реализация этих элементов может отличаться от приведенной в данной главе, но принцип работы не изменяется.



ГЛАВА 3

Основные схемотехнические решения цифровых микросхем

В настоящее время используется несколько технологий схемотехнического построения логических элементов:

1. Диодно-транзисторная логика (ДТЛ).
2. Транзисторно-транзисторная логика (ТТЛ, TTL).
3. Логика на основе комплементарных МОП-транзисторов (КМОП, CMOS).
4. Логика на основе сочетания комплементарных МОП и биполярных транзисторов (BiCMOS).

Первоначально получили распространение цифровые микросхемы, построенные на основе ДТЛ- и ТТЛ-технологий. Поэтому до сих пор существует огромное количество микросхем, либо построенных по этой технологии, либо совместимых с такими микросхемами по напряжению питания, логическим уровням и конструктивному исполнению.

Диодно-транзисторная логика (ДТЛ)

Наиболее простая реализация логического элемента "И" получается при помощи диодов. Схема такого элемента приведена на рис. 3.1.

В этой схеме, при подаче нулевого потенциала на любой из входов (или на оба сразу) через резистор будет протекать ток и на его сопротивлении возникнет падение напряжения. Напряжение на выходе схемы в этой ситуации будет близко к потенциалу общего провода. В результате единичный потенциал на выходе схемы появится, только если подать единичный потенциал сразу на оба входа логического элемента. То есть анализируемая схема реализует логическую функцию "2И".

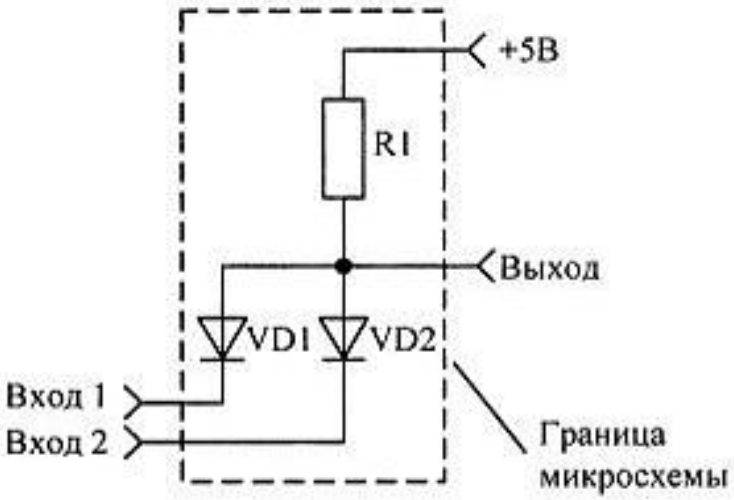


Рис. 3.1. Принципиальная схема логического элемента "2И", выполненного на диодах

Количество входов логического элемента "И" в данной схеме зависит от количества диодов. Если использовать два диода, то получится элемент "2И", если три диода —"3И", если четыре диода, то "4И", и т. д.

Приведенная схема обладает таким недостатком, как смещение логических уровней на выходе микросхемы. Напряжение нуля и напряжение единицы на выходе схемы выше входных уровней на 0,7 В. Это вызвано падением напряжения на диодах. Скомпенсировать данное смещение уровней можно диодом, включенным на выходе схемы, как это показано на рис. 3.2.

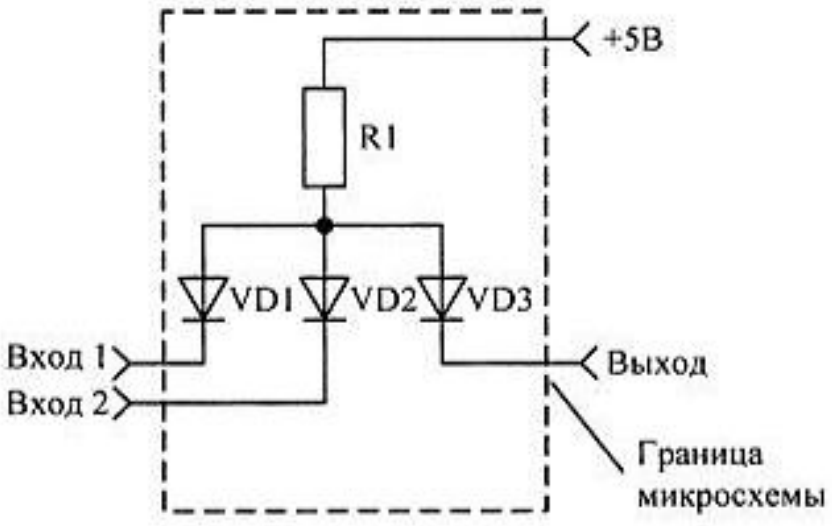


Рис. 3.2. Принципиальная схема усовершенствованного логического элемента "2И", выполненного на диодах

В этой схеме логические уровни на входе и выходе схемы одинаковы. Более того, схема, приведенная на рис. 3.2, будет нечувствительна не только к входным напряжениям, превышающим напряжение питания, но и к отрицательным входным напряжениям. Диоды обычно выдерживают обратное на-

пряжение до сотен вольт. Поэтому *такая схема дополнительно используется для защиты цифровых устройств от перегрузок по напряжению*, возникающих, например, в цепях, выходящих за пределы цифрового устройства. Если не требуется реализация логической функции, а необходимо просто защитить вход микросхемы от перегрузки по напряжению, то используется схема подключения диодов, приведенная на рис. 3.3.

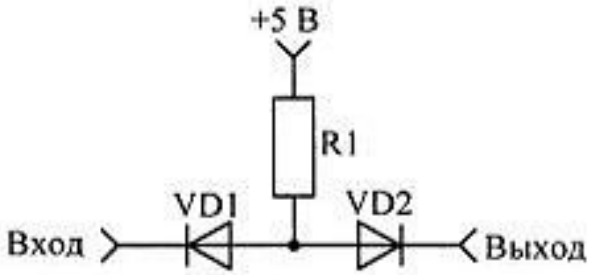


Рис. 3.3. Принципиальная схема диодной защиты входов цифрового устройства

Подобные схемы применяются на входах цифровых блоков, которые подключены к внешним разъемам этих блоков. Так как к этим разъемам есть доступ, к ним может прикоснуться человек, а на нем при передвижении (например, по линолеуму) могут наводиться статические потенциалы до нескольких тысяч вольт. Ни одна микросхема не способна выдержать воздействие такого потенциала. Защита на дискретных диодах выдержит.

Еще один источник опасности для микросхем заключается в том, что к разъемам обычно подключаются длинные линии, поэтому на них может наводиться высокое напряжение от сетевой проводки. Более того! Возможно прямое падение проводов с напряжением 220 или 380 В на линию передачи цифрового сигнала. Если защитные диоды выдерживают такое напряжение, то и в этом случае схема, приведенная на рис. 3.3, предотвратит выход цифрового блока из строя.

Приведенная на рис. 3.2 схема логического элемента "И", к сожалению, не может соединяться последовательно, т. к. вырабатывает только вытекающий ток, а для следующего каскада требуется втекающий выходной ток схемы. Кроме того, эта схема не обладает усилением. В результате логические сигналы при распространении в таких схемах будут затухать.

Для того чтобы избежать описанной ситуации, к схеме диодного логического элемента "И" обычно подключается двухтактный усилитель, выполненный на биполярных транзисторах. Схема полученного диодно-транзисторного логического (ДТЛ) элемента приведена на рис. 3.4.

Логический элемент "2И" в приведенной на рис. 3.4 схеме образуется элементами VD1, VD3 и R1. Использование двухтактного усилителя позволяет получать от схемы как втекающий, так и вытекающий ток, однако следует

помнить, что подобный усилитель является источником напряжения, и если не ограничить выходной ток микросхемы во внешних цепях, то можно вывести ее из строя.

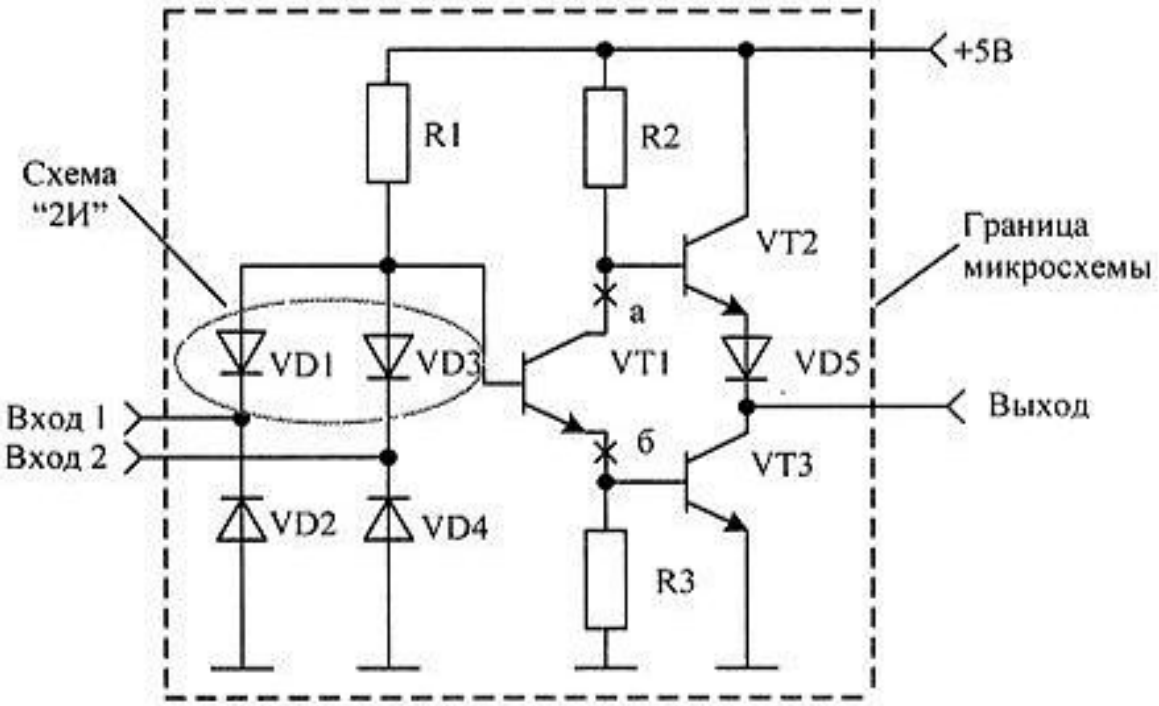


Рис. 3.4. Принципиальная схема базового элемента микросхемы ДТЛ

✧ *Обратите внимание*, что транзистор VT1 инвертирует сигнал на выходе элемента "И". То есть вместо логической единицы на выходе этой схемы присутствует логический ноль. И, наоборот, вместо логического нуля на выходе схемы присутствует логическая единица. В результате базовый элемент ДТЛ реализует не просто функцию "2И", а более сложную логическую функцию "2И-НЕ":

$$F = \overline{x1 \wedge x2}$$

Диоды VD2 и VD4 на входе схемы служат для защиты от отрицательного напряжения, которое может поступать на ее вход. Условно-графическое изображение такого логического элемента показано на рис. 3.5, а его таблица истинности приведена в табл. 3.1.

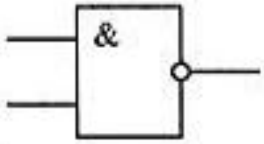


Рис. 3.5. Условно-графическое изображение элемента "2И-НЕ"

Таблица 3.1. *Таблица истинности схемы, выполняющей логическую функцию "2И-НЕ"*

x_1	x_2	F
0	0	1
0	1	1
1	0	1
1	1	0

Приведенная на рис. 3.4 схема логического элемента используется в таких современных сериях микросхем, как 555, 533, 1531, 1533. В этих сериях микросхем для повышения быстродействия применяются транзисторы и диоды Шоттки.

На основе базового элемента, приведенного на рис. 3.4, строится и ДТЛ инвертор. В этом случае на входе схемы используется только один диод. Схема инвертора ДТЛ приведена на рис. 3.6.

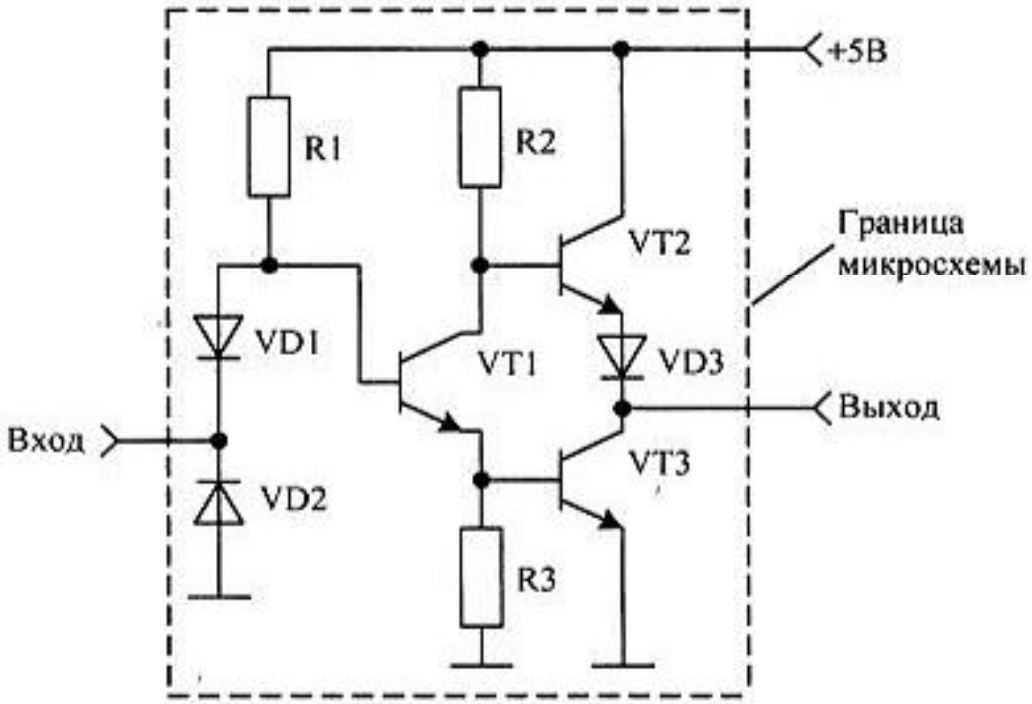


Рис. 3.6. Принципиальная схема инвертора ДТЛ-микросхемы

В состав современных серий цифровых микросхем, кроме логических элементов "И", входят логические элементы "ИЛИ". Для реализации логического элемента "ИЛИ", точно так же, как и в эквивалентной схеме, в схеме, приведенной на рис. 3.4, в точках "а" и "б" параллельно соединяются транзисторы VT1 и VT2. Выходной каскад при этом используется один. В качестве

примера на рис. 3.7 приведена принципиальная схема логического элемента "2ИЛИ-НЕ".

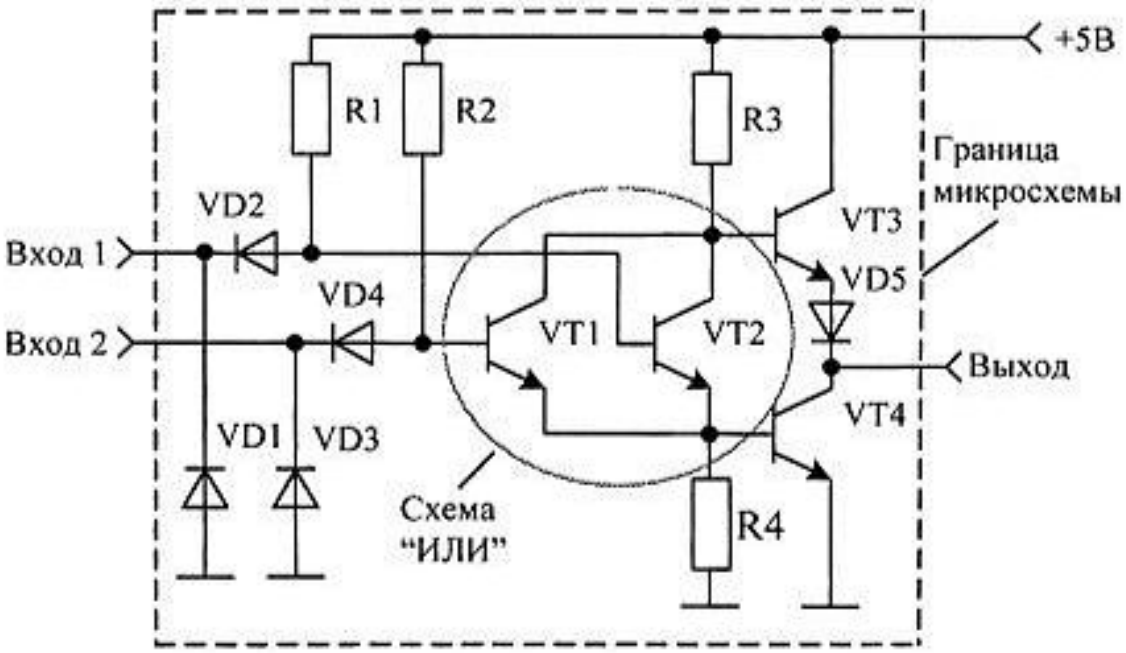


Рис. 3.7. Принципиальная схема логического элемента "2ИЛИ-НЕ" ДТЛ-микросхемы

Логические элементы "ИЛИ-НЕ" в отечественных сериях микросхем имеют обозначение ЛЕ. Например, схема К555ЛЕ1 содержит в одном корпусе четыре элемента "2ИЛИ-НЕ". Таблица истинности, реализуемая схемой, приведенной на рис. 3.7, сведена в табл. 3.2, а условно-графическое изображение рассматриваемого логического элемента показано на рис. 3.8.

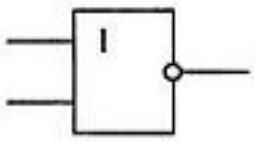


Рис. 3.8. Условно-графическое изображение элемента "2ИЛИ-НЕ"

***Таблица 3.2.** Таблица истинности схемы, выполняющей логическую функцию "2ИЛИ-НЕ"*

x_1	x_2	F
0	0	1
0	1	0
1	0	0
1	1	0

Транзисторно-транзисторная логика (ТТЛ)

В ТТЛ-схемах для реализации логического элемента "И" вместо параллельного соединения диодов используется многоэмиттерный транзистор. Физика работы этого элемента не отличается от работы диодного элемента "2И".

Высокий потенциал на коллекторе многоэмиттерного транзистора получается за счет отсутствия коллекторного тока только в том случае, когда на обоих входах логического элемента (эмиттерах транзистора) присутствует высокий потенциал. Принципиальная схема типового элемента ТТЛ-микросхемы приведена на рис. 3.9.

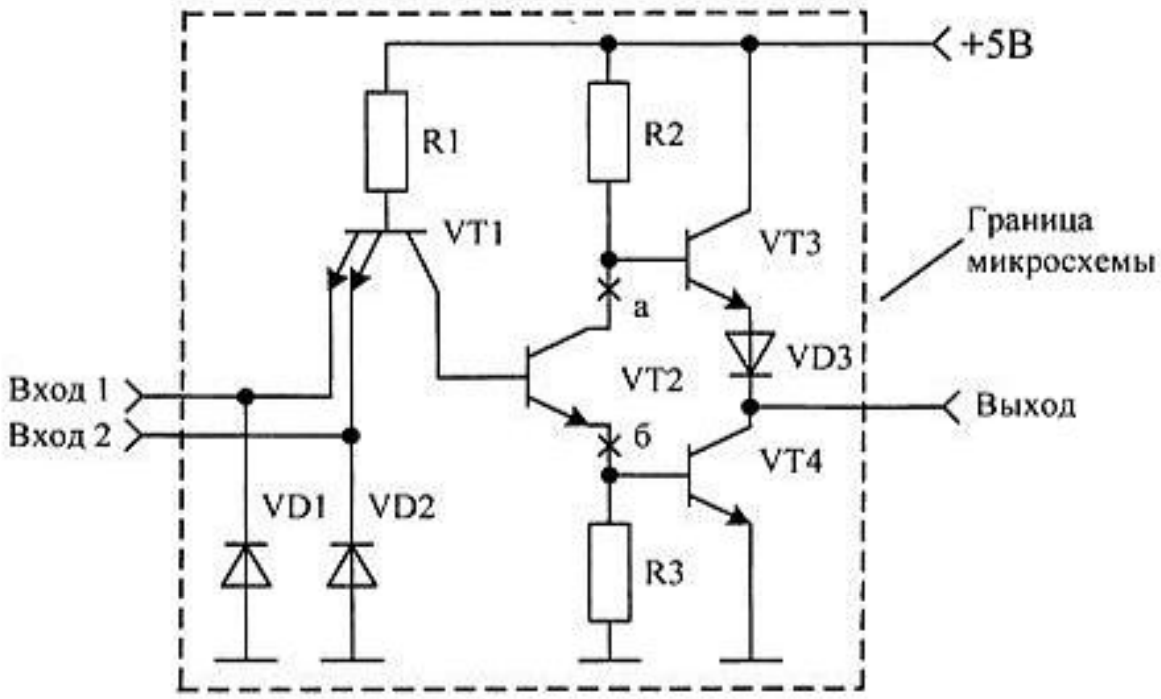


Рис. 3.9. Принципиальная схема типового элемента ТТЛ-микросхемы

Умощняющий усилитель, как и в диодно-транзисторном элементе, инвертирует сигнал на выходе схемы. По такой схеме выполнены базовые элементы микросхем серий 155, 131, 134 и 531. Схемы "И-НЕ" в этих сериях микросхем обычно имеют обозначение ЛА. Например, микросхема К531ЛА3 содержит в одном корпусе четыре элемента "2И-НЕ".

На основе базового элемента строятся и инверторы. В этом случае на входе логического элемента используется обычный одноэмиттерный транзистор. Схема инвертора, выполненного по ТТЛ-технологии, приведена на рис. 3.10.

При необходимости объединения нескольких логических элементов "И" по схеме "ИЛИ" (или при реализации логических элементов "ИЛИ") транзисторы VT2 соединяются параллельно в точках "а" и "б", показанных на рис. 3.9.

Выходной каскад при этом, как и в ДТЛ-элементах, используется один. В результате быстродействие такого элемента, выполняющего достаточно сложную логическую функцию, получается точно таким же, как и у одиночного инвертора.

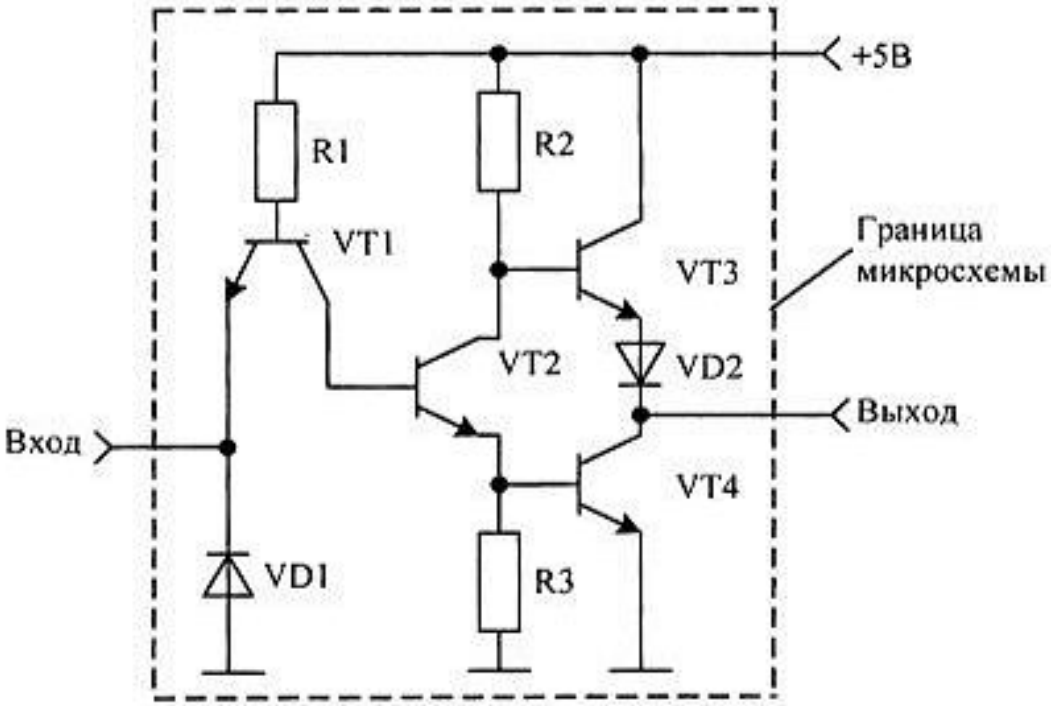


Рис. 3.10. Принципиальная схема инвертора ТТЛ-микросхемы

Как будет показано в следующих главах, это очень удобный и распространенный логический элемент. Принципиальная схема логического элемента "2И-2ИЛИ-НЕ" приведена на рис. 3.11.

Практическое применение таких логических элементов будет показано позднее, а условно-графическое изображение элемента "2И-2ИЛИ-НЕ" приведено на рис. 3.12. Подобные логические элементы содержатся в цифровых микросхемах, в обозначении которых содержатся буквы ЛР.

Схемы "ИЛИ-НЕ" в ТТЛ-сериях микросхем, как и в ДТЛ-сериях, обозначаются буквами ЛЕ. Например, микросхема К1531ЛЕ5 содержит в одном корпусе четыре элемента "2ИЛИ-НЕ".

Так как и в схемах ТТЛ, и в схемах ДТЛ используется одинаковый выходной усилитель, то и уровни логических сигналов в этих микросхемах одинаковы. Поэтому часто говорят, что это ТТЛ-микросхемы, не уточняя по какой схеме выполнен входной каскад рассматриваемых микросхем.

Сейчас уже существуют КМОП-микросхемы, совместимые с ТТЛ-микросхемами по логическим уровням, например серии микросхем К1564 (зарубежный аналог SN74HCT) или К1594 (зарубежный аналог SN74ACT).

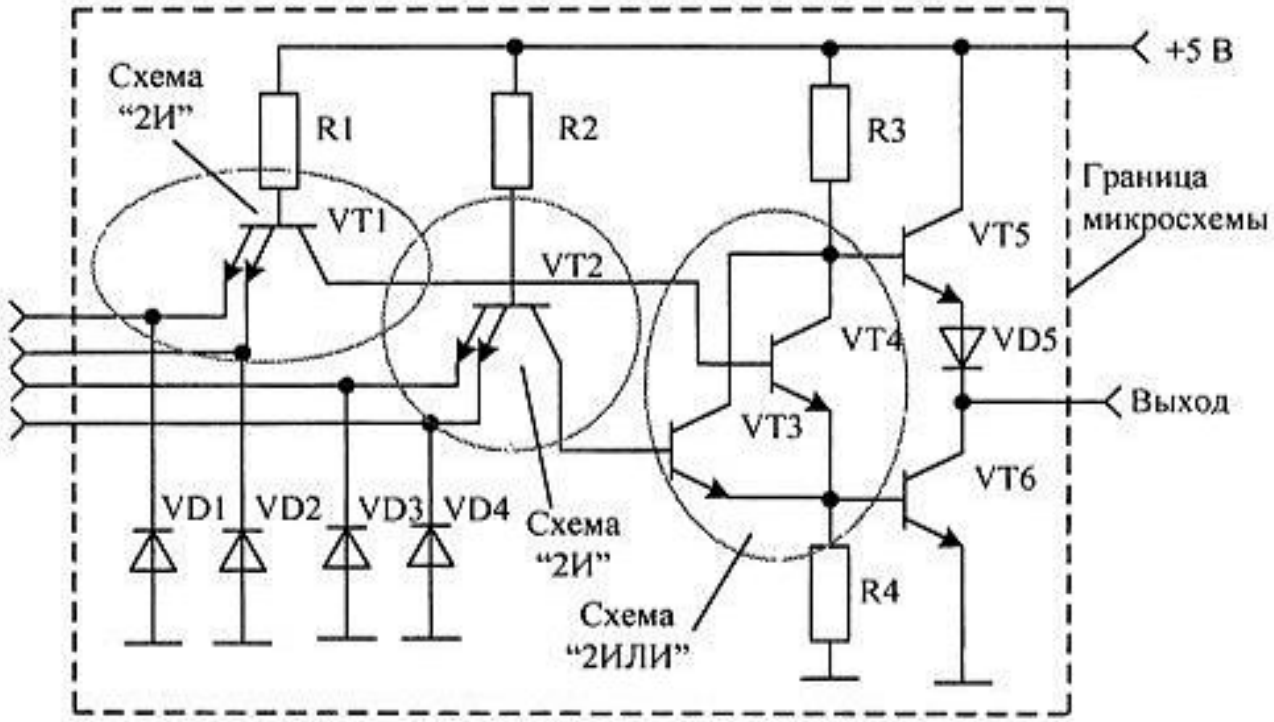


Рис. 3.11. Принципиальная схема ТТЛ-микросхемы "2И-2ИЛИ-НЕ"

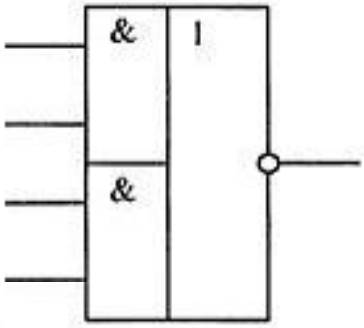


Рис. 3.12. Условно-графическое обозначение элемента "2И-2ИЛИ-НЕ" ТТЛ-микросхем

Логические уровни ТТЛ-микросхем

В настоящее время применяются два вида ТТЛ-микросхем — с пятивольтовым и с трехвольтовым питанием, но, независимо от напряжения питания микросхем, логические уровни нуля и единицы на выходе у них совпадают. Поэтому дополнительного согласования между такими ТТЛ-микросхемами обычно не требуется.

Мы помним, что логические уровни на выходе цифровых микросхем не обязательно равны напряжению питания или потенциалу общего провода. Конкретное значение напряжения зависит от многих факторов. Допустимые уровни напряжения на выходе цифровой ТТЛ-микросхемы показаны на рис. 3.13.

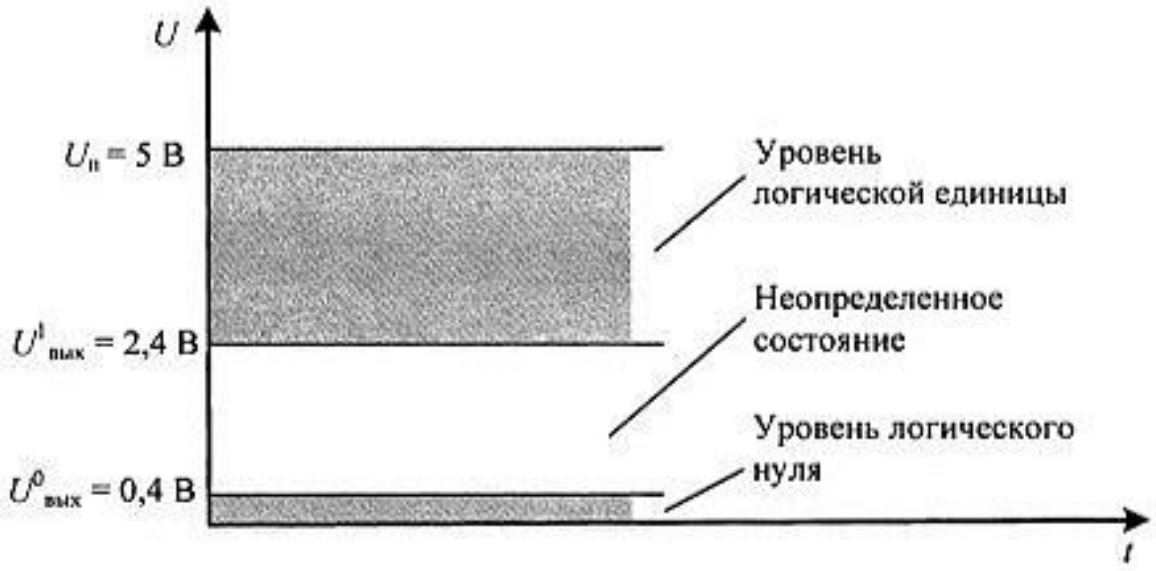


Рис. 3.13. Уровни логических сигналов на выходе цифровых ТТЛ-микросхем

Как уже говорилось ранее, разброс напряжений на входе цифровой микросхемы по сравнению с разбросом напряжений на ее выходе обычно допускается в больших пределах. Границы уровней напряжения логического нуля и единицы для ТТЛ-микросхем приведены на рис. 3.14.

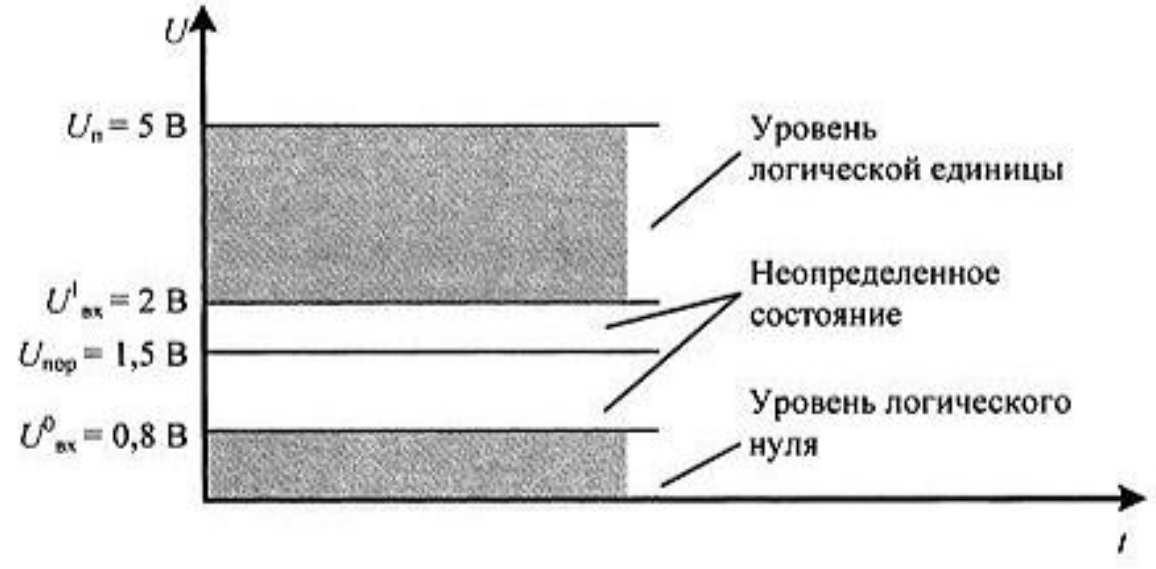


Рис. 3.14. Уровни логических сигналов на входе цифровых ТТЛ-микросхем

Пороговое напряжение для ТТЛ-микросхем, заявляемое рядом производителей микросхем, составляет 1,5 В.

Семейства ТТЛ-микросхем

Первые ТТЛ-микросхемы оказались на редкость удачным решением, поэтому их можно встретить в аппаратуре, работающей до сих пор. В качестве примера можно назвать семейство отечественных микросхем серии К155.

Стандартные ТТЛ-микросхемы — это микросхемы, питающиеся от источника напряжения +5 В. Зарубежные ТТЛ-микросхемы получили название SN74. Конкретные микросхемы этой серии обозначаются цифровым номером микросхемы, следующим за названием серии. Например, в микросхеме SN74S00 содержится четыре логических элемента "2И-НЕ" (отечественный аналог К555ЛА3).

Аналогичные микросхемы с расширенным температурным диапазоном получили название SN54 (отечественный вариант подобных микросхем — это серия микросхем К133).

Отечественные микросхемы, совместимые с семейством микросхем SN74, выпускались в составе серий К134 (низкое быстродействие и низкое энергопотребление — SN74L), К155 (среднее быстродействие и среднее энергопотребление — SN74) и К131 (высокое быстродействие и большое энергопотребление).

Затем были разработаны микросхемы повышенного быстродействия с диодами Шоттки. В названии зарубежных микросхем в обозначении серии появилась буква S. В отечественных сериях микросхем цифра 1 была заменена на цифру 5. В настоящее время выпускаются микросхемы серий К555 (SN74LS) — среднее быстродействие и низкое энергопотребление и серии микросхем и К531 (SN74S) — высокое быстродействие и большое энергопотребление.

В настоящее время промышленность выпускает цифровые микросхемы с улучшенными параметрами, серии: К1533 (низкое быстродействие, низкое потребление — зарубежный аналог SN74ALS) и К1531 (высокое быстродействие и большое потребление — зарубежный аналог SN74F).

Кроме того, производится трехвольтовый вариант ТТЛ совместимых микросхем — серия 1554 (зарубежный аналог — SN74ALB), выполненных по совмещенной биполярно-КМОП (Bi-CMOS) технологии, сочетающей преимущества этих технологий.

Логика на комплементарных МОП-транзисторах (КМДП)

Микросхемы на комплементарных парах транзисторов строятся на основе МОП-транзисторов с n- и p-каналами. В этих схемах один и тот же потенциал открывает транзистор с n-каналом и закрывает транзистор с p-каналом.

Простейший логический КМОП-элемент — это инвертор. Его схема приведена на рис. 3.15. При формировании логической единицы открыт верхний

транзистор, а нижний закрыт. В результате ток через микросхему не протекает. При формировании логического нуля открыт нижний транзистор, а верхний закрыт. И в этом случае ток через микросхему не протекает.

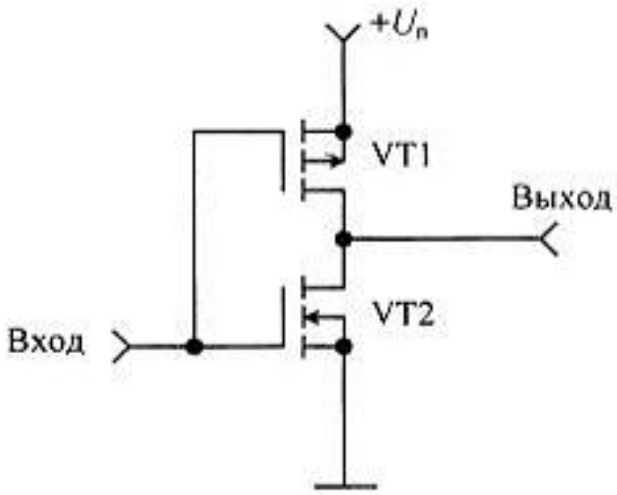


Рис. 3.15. Принципиальная схема инвертора, выполненного на комплементарных МОП-транзисторах

На схеме, приведенной на рис. 3.15, для упрощения понимания принципов работы микросхемы, не показаны защитные и "паразитные" диоды. *Особенностью микросхем на комплементарных МОП-транзисторах является то, что эти микросхемы в статическом режиме практически не потребляют ток.*

Потребление тока происходит только в момент переключения микросхемы из единичного состояния в нулевое и наоборот. Этим током перезаряжается паразитная емкость нагрузки цифровой микросхемы.

Схема логического элемента "И-НЕ" на КМОП-микросхемах практически совпадает с эквивалентной схемой логического элемента "И" на ключах с электронным управлением, которую мы рассматривали ранее. Отличие заключается в том, что нагрузка подключается не к общему проводу схемы, а к источнику питания. Это приводит к инверсии сигнала на выходе логического элемента. Принципиальная схема элемента "2И-НЕ", выполненного на комплементарных МОП-транзисторах, приведена на рис. 3.16.

В этой схеме можно было бы применить в верхнем плече обыкновенный резистор, однако при формировании напряжения низкого уровня схема постоянно потребляла бы ток. Вместо этого, в приведенной на рис. 3.16 схеме, в качестве нагрузки используются транзисторы р-МОП. Эти транзисторы образуют активную нагрузку. Если на выходе логического элемента требуется сформировать высокий потенциал, то нагрузочные транзисторы р-МОП открываются, а если низкий — закрываются.

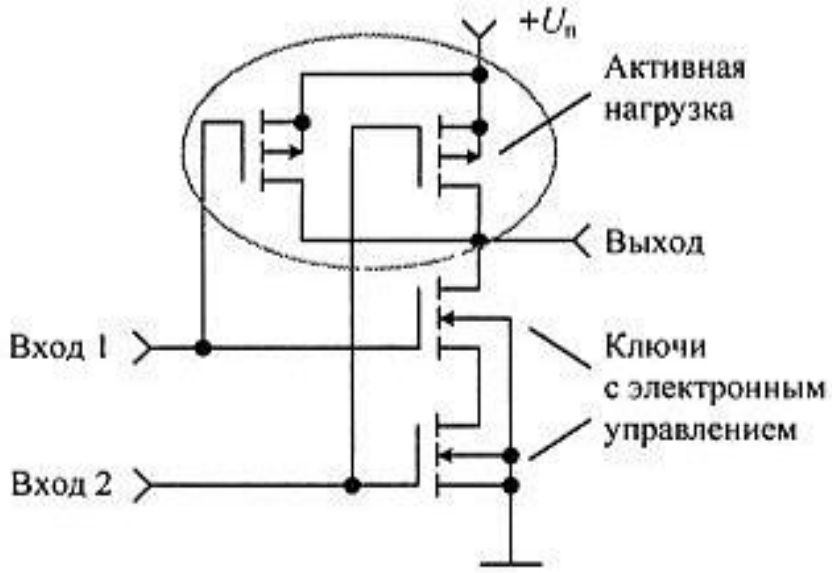


Рис. 3.16. Принципиальная схема элемента "2И-НЕ", выполненного на комплементарных МОП-транзисторах

В приведенной на рис. 3.16 схеме ток от источника питания на выход микросхемы будет поступать через один из транзисторов нагрузки, если хотя бы на одном из входов (или на обоих сразу) будет присутствовать низкий потенциал (уровень логического нуля). Если же на обоих входах будет присутствовать уровень логической единицы, то оба транзистора р-МОП будут закрыты, ток от источника питания поступать не будет, и на выходе микросхемы сформируется низкий потенциал. В этой схеме, так же как и в схеме инвертора, приведенной на рис. 3.15, если хотя бы один из транзисторов верхнего плеча будет открыт, то соответствующий транзистор нижнего плеча будет закрыт. Поэтому в статическом состоянии ток от источника питания логическим элементом потребляться не будет.

Условно-графическое изображение такого логического элемента показано на рис. 3.5, а его таблица истинности приведена в табл. 3.1. В табл. 3.1 входы обозначены как x_1 и x_2, а выход — F.

Логический элемент "ИЛИ-НЕ", выполненный на КМОП-транзисторах, представляет собой параллельное соединение ключей с электронным управлением. Отличие от схемы "2ИЛИ", рассмотренной ранее, заключается в том, что нагрузка подключается не к общему проводу схемы, а к источнику питания. Это приводит к инверсии выходного сигнала. Вместо резистора в качестве нагрузки используются транзисторы р-МОП. Принципиальная схема элемента "2ИЛИ-НЕ", выполненного на комплементарных МОП-транзисторах, приведена на рис. 3.17.

В схеме логического элемента "2ИЛИ-НЕ" в качестве нагрузки используются последовательно включенные транзисторы р-МОП. В ней ток от источника питания будет поступать на выход микросхемы, только если все транзисторы

в верхнем плече будут открыты, т. е. если сразу на всех входах будет присутствовать низкий потенциал (уровень логического нуля). Если же хотя бы на одном из входов будет присутствовать уровень логической единицы, то верхнее плечо будет закрыто, и ток от источника питания поступать на выход микросхемы не будет.

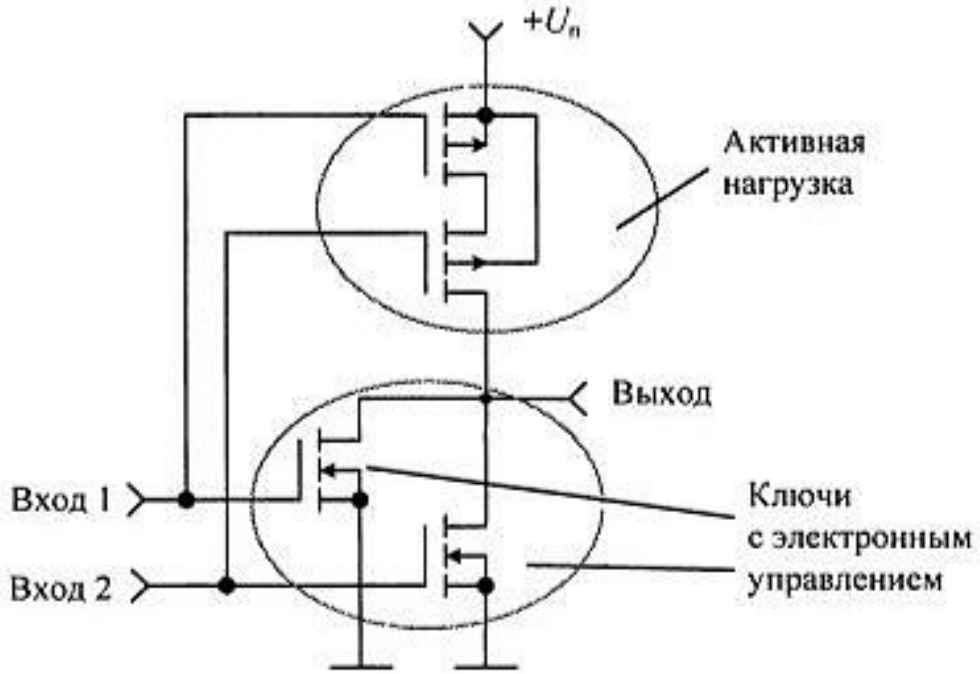


Рис. 3.17. Принципиальная схема элемента "2ИЛИ-НЕ", выполненного на комплементарных МОП-транзисторах

Таблица истинности, реализуемая схемой, приведенной на рис. 3.17, соответствует табл. 3.2, а условно-графическое обозначение этого элемента совпадает с условно-графическим обозначением, приведенным на рис. 3.8.

В настоящее время получили наибольшее развитие именно КМОП-микросхемы. Причем наблюдается постоянная тенденция к снижению напряжения питания этих микросхем. Первые серии КМОП-микросхем, такие как К1561 (зарубежный аналог С4000В), обладали достаточно широким диапазоном изменения напряжения питания (3 ... 18 В). При понижении напряжения питания у конкретной КМОП-микросхемы понижается ее предельная частота работы, и наоборот (на этом явлении основан так называемый разгон современных центральных процессоров персональных компьютеров). В дальнейшем, по мере совершенствования технологии производства, появились улучшенные КМОП-микросхемы с лучшими частотными свойствами. Однако для создания подобных микросхем потребовались транзисторы с меньшими линейными размерами, что привело к меньшим допустимым напряжениям питания этих микросхем.

Особенности применения КМОП-микросхем

Первой и основной особенностью КМОП-микросхем является их большое входное сопротивление. В результате на вход цифровой микросхемы, выполненной по КМОП-технологии, может наводиться любое паразитное напряжение, в том числе и равное половине напряжения питания. Это напряжение может сохраняться на нем достаточно долго.

При подаче на вход КМОП-микросхемы половины напряжения питания открываются транзисторы, как в верхнем, так и в нижнем плече выходного каскада микросхемы, в результате она начинает потреблять недопустимо большой ток и, как следствие, может выйти из строя. Поэтому *входы цифровых КМОП-микросхем ни в коем случае нельзя оставлять неподключенными.* Конструкция КМОП-инвертора приведена на рис. 3.18, а полная схема КМОП-инвертора с учетом защитных и паразитных диодов — на рис. 3.19.

К сожалению, преимущества КМОП-микросхем при определенных условиях могут оказаться недостатками. Так как сопротивление изоляции затвора МОП-транзистора очень велико, на затворе может наводиться очень высокий статический потенциал. Этим потенциалом изоляция затвора транзистора может быть пробита. Поэтому для защиты входного каскада от пробоя статическим электричеством в схему базового элемента были введены диоды VD1 и VD2.

Эта конструктивная особенность приводит к тому, что при подаче на вход микросхемы высокого потенциала он через диод VD1 попадет на шину питания микросхемы, и т. к. КМОП-микросхема потребляет достаточно малый ток, она начнет работать. Однако, в ряде случаев, тока для надежной работы схемы может не хватить. В результате микросхема может работать непра-

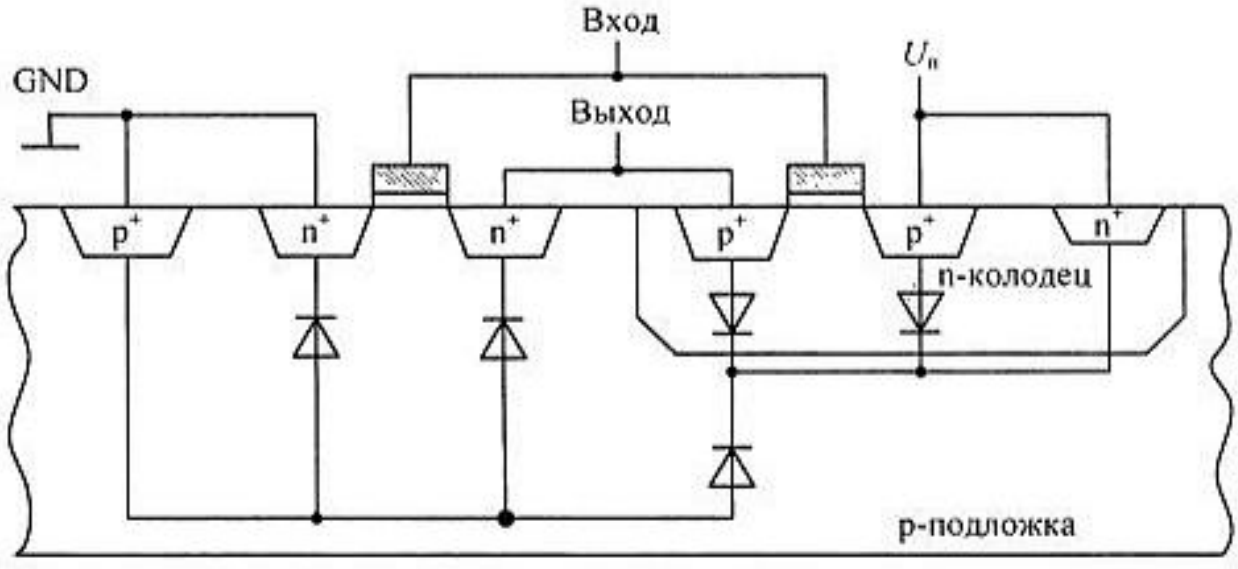


Рис. 3.18. Конструкция КМОП-инвертора

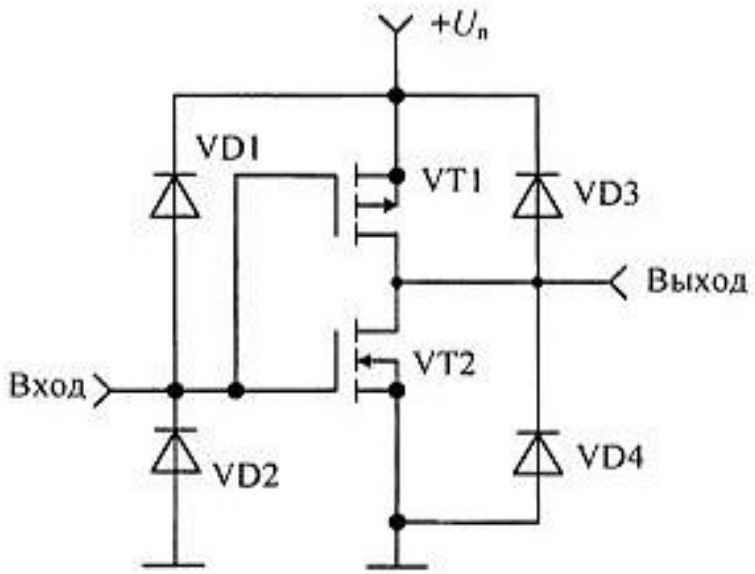


Рис. 3.19. Полная схема КМОП-инвертора

вильно. Вывод: *при неправильной работе микросхемы тщательно проверьте питание микросхемы*, особенно выводы корпуса. При плохо пропаянном выводе корпуса микросхемы его потенциал будет отличаться от потенциала общего провода схемы.

Третья особенность работы КМОП-микросхем связана с существованием паразитных диодов VD3 и VD4 (см. рис. 3.19). Эти диоды, при неправильно подключенном источнике питания, могут быть пробиты (микросхемы ТТЛ выдерживают кратковременную переполюсовку питания). *Для защиты микросхем от переполюсовки питания следует в цепи питания предусмотреть защитный диод.* Это относится и к ТТЛ-микросхемам, но в этой ситуации есть вероятность, что микросхема выйдет из строя только через некоторое время.

Эта же особенность может приводить к тому, что незапитанные КМОП-микросхемы могут через паразитные выходные диоды закорачивать на корпус линии передачи сигналов.

Четвертая особенность КМОП-микросхем — это протекание импульсного тока по цепи питания при переключении микросхемы из нулевого состояния в единичное и наоборот. В результате, при замене ТТЛ-микросхем на аналогичные КМОП-микросхемы резко увеличивается уровень помех. В ряде случаев это важно, и приходится отказываться от применения КМОП-микросхем в пользу микросхем, выполненных по технологии ТТЛ или BICMOS.

Еще одно неприятное явление в КМОП-микросхемах связано с паразитными структурами, образующимися в полупроводнике. Если внимательно посмотреть на рис. 3.18, то можно увидеть, что между источником питания и корпусом образуется паразитная p-n-p^+-n-структура. При превышении напряжения

на входе микросхемы над напряжением питания в n-карман может инжектироваться достаточное для того, чтобы этот n-карман исчез, количество дырок. В результате через микросхему будет протекать ничем не ограниченный ток. Теперь уже этот ток будет инжектировать в n-карман дополнительные положительные заряды. Процесс становится неуправляемым и может быть прекращен только снятием напряжения питания. *Это явление получило название защелкивания.* К подобному эффекту может привести подача на вход микросхемы и отрицательного напряжения. Поэтому при проектировании схемы цифрового блока следует принимать меры для того, чтобы напряжение на входе микросхемы находилось в пределах от потенциала общего провода до напряжения питания.

Логические уровни КМОП-микросхем

Логические уровни КМОП-микросхем существенно отличаются от логических уровней ТТЛ-микросхем. При отсутствии тока нагрузки напряжение на выходе КМОП-микросхемы совпадает с напряжением питания (логический уровень единицы) или с потенциалом общего провода (логический уровень нуля). При увеличении тока нагрузки напряжение логической единицы может уменьшаться до двух третьих от напряжения питания.

Для современных КМОП-микросхем требования по логическим уровням более жесткие. Допустимый уровень напряжения на выходе цифровой КМОП-микросхемы серии SN74HC при пятивольтовом питании показан на рис. 3.20.

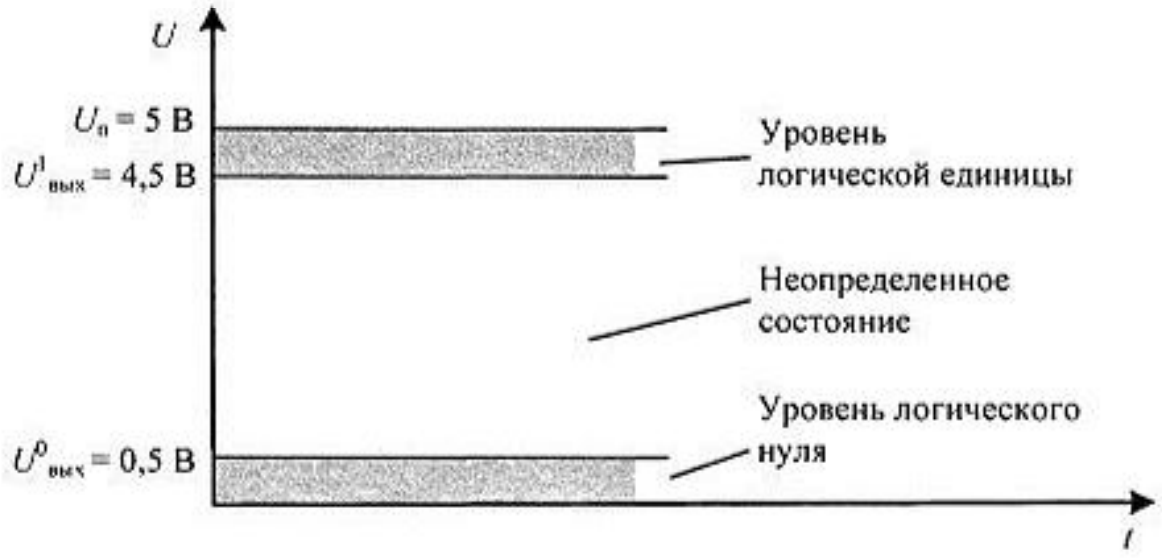


Рис. 3.20. Уровни логических сигналов на выходе цифровых КМОП-микросхем

Как уже отмечалось ранее, изменение напряжения на входе цифровой микросхемы по сравнению с выходом обычно допускается в больших пределах. Для КМОП-микросхем определен 30%-й запас по напряжению. Границы

уровней логического нуля и единицы для КМОП-микросхем при пятивольтовом питании приведены на рис. 3.21.

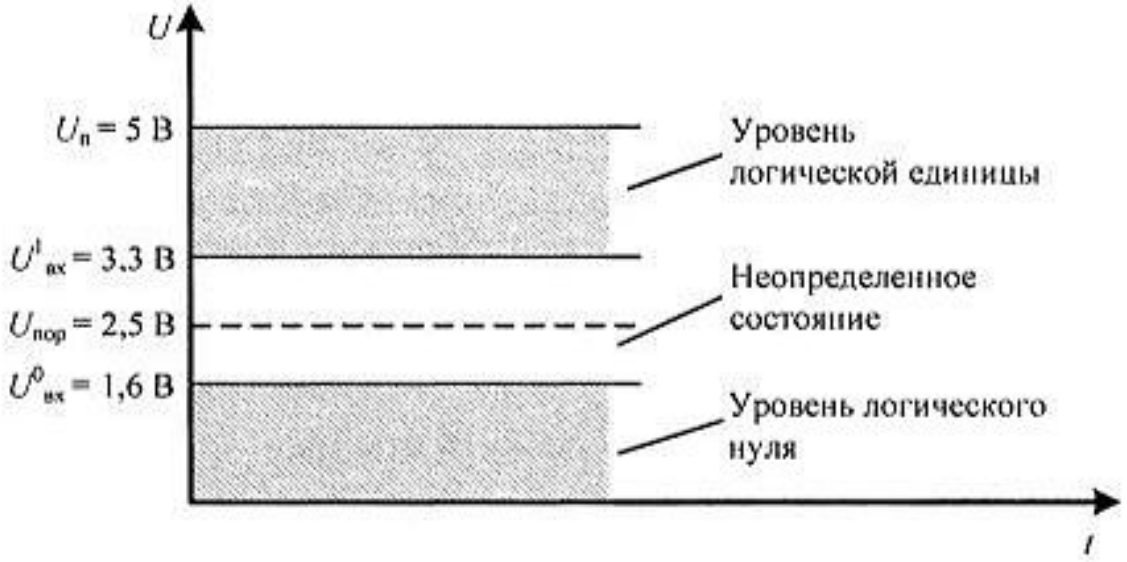


Рис. 3.21. Уровни логических сигналов на входе цифровых КМОП-микросхем

При уменьшении напряжения питания границы логического нуля и логической единицы уменьшаются в той же пропорции, как и на рис. 3.21. (Для того чтобы узнать уровень логического нуля, необходимо разделить напряжение питания на 3.)

Семейства КМОП-микросхем

Первые КМОП-микросхемы не имели защитных диодов на входе, поэтому их монтаж представлял значительные трудности. Затворы МОП-транзисторов при монтаже часто пробивались статическим электричеством. Это семейство цифровых микросхем серии К172. В следующем, улучшенном семействе микросхем серии К176 были включены защитные диоды. Эти микросхемы достаточно распространены и в настоящее время. Серия К1561 завершает развитие первого поколения КМОП-микросхем. В этом семействе было достигнуто быстродействие микросхем на уровне 90 нс и диапазон изменения напряжения питания 3 ... 15 В. Зарубежный аналог этих микросхем — С4000В.

Дальнейшим развитием КМОП-микросхем стала серия SN74HC. Эти микросхемы отечественного аналога не имеют. Они обладают быстродействием 27 нс и могут работать в диапазоне напряжений 2 ... 6 В. Микросхемы серии SN74HC совпадают по конструктивному исполнению и функциональному ряду с ТТЛ-микросхемами, но не совместимы с ними по логическим уровням, поэтому одновременно были разработаны микросхемы серии SN74HCT (оте-

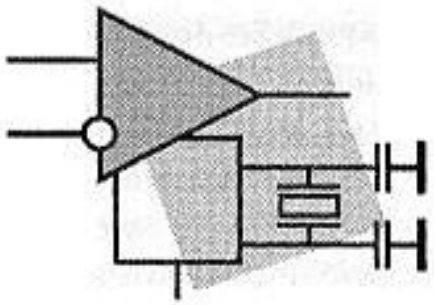
чественный аналог — К1564), совместимые с ТТЛ-микросхемами и по логическим уровням.

На данный момент в радиоэлектронной аппаратуре наметился переход на трехвольтовое питание. Для него были разработаны микросхемы SN74ALVC с диапазоном питания 1,65 ... 3,6 В и временем задержки сигнала 5,5 нс. Такое время задержки реализуется при напряжении питания 3,3 В. Время задержки сигнала при напряжении питания 2,5 В увеличивается до 9 нс.

В настоящее время наиболее перспективным семейством КМОП-микросхем считается семейство SN74AUC с временем задержки сигнала 1,9 нс и диапазоном напряжения питания 0,8 ... 2,7 В.

Итоги

В данной главе мы познакомились со схемотехнической реализацией наиболее распространенных решений цифровых микросхем. При работе с цифровыми устройствами всегда следует учитывать эти особенности. Это позволит избежать неприятных неожиданностей при первом включении цифрового блока (или при совместном применении нескольких цифровых устройств). Кроме того, не следует забывать и об остальных параметрах цифровых микросхем, рассмотренных в первой главе.



ГЛАВА 4

Согласование цифровых микросхем между собой

При проектировании цифровых схем, как правило, используют микросхемы одной серии. Однако это не всегда удается. Применять микросхемы других серий приходится:

1. Когда требуются микросхемы, отсутствующие в данной серии микросхем.
2. Когда отдельные узлы схемы должны работать на повышенной частоте.
3. При работе на внешние устройства могут потребоваться микросхемы с повышенной нагрузочной способностью.

Первый пункт не требует комментариев. Обычно малым набором микросхем характеризуются серии с повышенным быстродействием или с повышенной нагрузочной способностью. Серии микросхем с малым набором микросхем также обычно характеризуются высокой стоимостью. Так что первый пункт жестко связан с оставшимися двумя.

Что касается второго пункта, то выбор микросхем из различных серий может быть обусловлен двумя причинами.

Первая причина — это стоимость цифрового устройства в целом. Микросхемы с повышенным быстродействием стоят дороже микросхем со средним быстродействием. Микросхемы в одном и том же цифровом устройстве обычно работают на разных частотах. При этом на повышенной частоте работает не более одного процента от общего количества микросхем. В результате применение микросхем с различным быстродействием может существенно снизить стоимость всего цифрового устройства.

Вторая причина — это ток потребления микросхем. В сериях микросхем ТТЛ, р-МОП и n-МОП ток их потребления определяется быстродействием. Чем ниже быстродействие микросхемы (в пределах одной технологии), тем меньше ее ток потребления.

Данное обстоятельство не относится к КМОП-сериям микросхем. В микросхемах, выполненных по КМОП-технологии, ток потребления зависит от частоты, на которой работает в данный момент микросхема. Чем выше частота переключения логических элементов КМОП-микросхемы, тем выше ток потребления этой микросхемы. Иными словами, ток потребления в этих микросхемах регулируется автоматически, и основной причиной выбора конкретной серии микросхем остается только их стоимость и конструктивное исполнение.

Микросхемы с повышенной нагрузочной способностью обычно входят в состав любой серии микросхем, однако иногда требуются еще большие токи. В этом случае можно использовать микросхемы из серий с повышенным быстродействием, например К1531. При необходимости формирования на выходе цифровой микросхемы потенциалов, превышающих напряжение питания, можно применить микросхемы с открытым коллектором. В крайнем случае, для согласования микросхемы по току или напряжению можно применить транзисторный ключ.

Согласование цифровых микросхем из различных серий между собой

Рассмотрим сначала микросхемы, совместимые по логическим уровням с ТТЛ-микросхемами. Выбор ТТЛ-микросхем связан с тем, что ТТЛ логические уровни стали стандартом для современной цифровой техники. Даже если микросхемы внутри выполнены по КМОП-технологии, они обычно формируют на выходе логические уровни, совместимые по напряжению с уровнями ТТЛ.

Стандартные ТТЛ-микросхемы — это микросхемы, питающиеся от источника напряжения +5 В. В настоящее время наибольшее распространение получили ТТЛ-микросхемы с 3-вольтовым питанием. Кроме того, на рынке имеется большой выбор микросхем с 2,5-вольтовым и даже 0,7-вольтовым питанием. Как и любые другие электронные устройства, цифровые микросхемы обладают выходным и входным током. Поэтому, несмотря на то, что логические функции могут быть выполнены на микросхемах различных серий абсолютно одинаковым образом, вопросы согласования цифровых микросхем по напряжению и току являются очень важными.

Согласование микросхем по току

Согласование микросхем приведенных выше серий ТТЛ между собой сводится к согласованию по току, т. к. напряжения логических уровней этих

микросхем совпадают. Рассмотрим эквивалентную схему протекания выходного тока нуля I_0 ТТЛ-микросхемы, приведенную на рис. 4.1.

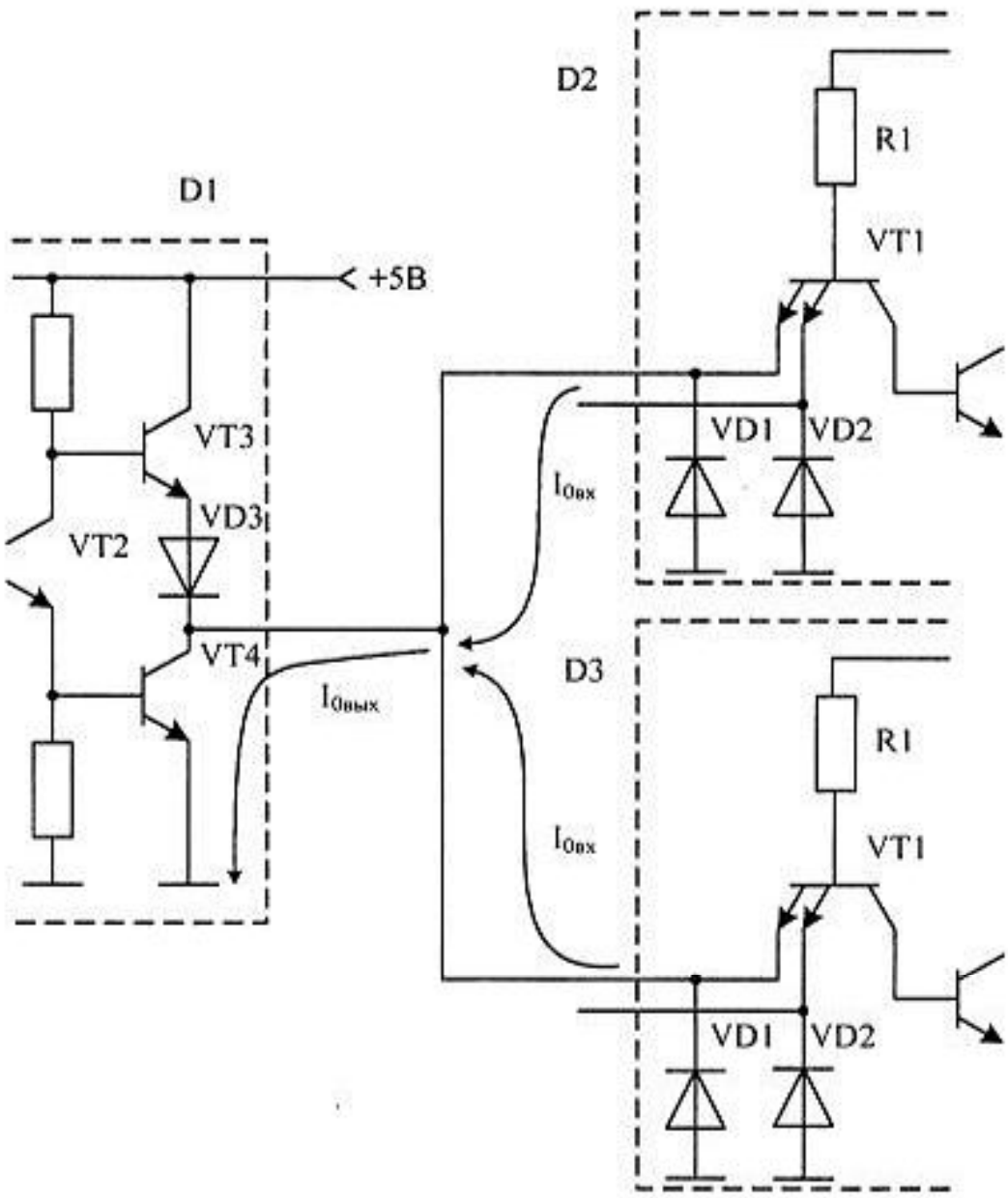


Рис. 4.1. Эквивалентная схема протекания выходного тока ТТЛ-микросхемы

Как видно из приведенной на рис. 4.1 схемы, выходной ток ТТЛ-микросхемы формируется из входных токов микросхем, подключенных к ее выходу. Это означает, что суммарный входной ток микросхем-нагрузок не должен превышать максимального выходного тока микросхемы — источника логического сигнала.

Рассмотрим возможность использования в качестве нагрузки для микросхемы К134ЛБ1 микросхемы К531ТМ2. Максимальный допустимый ток нуля микросхем серии К134 составляет 1,8 мА. Входной ток нуля микросхем серии К531 равен 2 мА. То есть входной ток микросхемы нагрузки превышает максимальный ток микросхемы источника сигнала.

Это означает, что между микросхемой серии К134 и микросхемой серии К531 должна находиться промежуточная микросхема серии, у которой входной ток будет меньшей величины, например, К555ЛН1. У этой микросхемы входной ток нуля не превышает значения 0,4 мА, т. е. к микросхеме К134ЛБ1 можно подключать до четырех входов микросхем серии К555:

$$I_{вых} = N \times I_{вх555} = 4 \times 0,4 \text{ мА} = 1,6 \text{ мА} < 1,8 \text{ мА}$$

У микросхем серии К555 допустимый выходной ток составляет 4 мА. Поэтому к выходу микросхемы этого семейства можно подключать до двух входов микросхем серии К531. Более современными являются микросхемы серии К1531. Они обладают быстродействием микросхем серии К531, но при этом их входной ток не превышает значения 0,6 мА, поэтому микросхемы этих серий могут быть подключены непосредственно к выходу микросхем серии К134. Максимальное допустимое количество входов микросхем серии К1531, которые могут быть подключены к выходу микросхемы серии К134 (коэффициент разветвления) можно рассчитать из формулы:

$$K_{раз} = \frac{I_{вых134}}{I_{вх1531}} = \frac{1,8 \text{ мА}}{0,6 \text{ мА}} = 3$$

Точно так же можно определить *коэффициент разветвления* и для других сочетаний микросхем. Даже в пределах одной серии микросхем можно воспользоваться этой формулой. Возьмем в качестве примера микросхемы серии К1533. Их входной ток равен 0,2 мА, выходной ток — 8 мА. В результате получаем коэффициент разветвления, равный 40:

$$K_{раз} = \frac{I_{вых1533}}{I_{вх1533}} = \frac{8 \text{ мА}}{0,2 \text{ мА}} = 40$$

В настоящее время происходит активный переход к микросхемам с пониженным напряжением питания, таким как 3,3 В; 2,5 В или 1,8 В, поэтому кроме вопроса согласования микросхем по току встает вопрос согласования микросхем по напряжению логических уровней. Точно такие же сложности возникают и при согласовании микросхем КМОП и ТТЛ.

Согласование микросхем с различным напряжением питания

Снижение напряжения питания цифровых микросхем обусловлено двумя причинами. Первая — это снижение потребляемой микросхемами мощности. Снижение напряжения питания с 5 до 3,3 В только по закону Ома приводит к снижению потребляемой мощности в 2—3 раза. Вторая причина — это

уменьшение линейных размеров транзисторов. При снижении линейных размеров транзисторов, входящих в состав цифровых микросхем, уменьшается их пробивное напряжение. В настоящее время наиболее распространенным напряжением питания цифровых микросхем стало напряжение питания 3,3 В.

Согласование 3- и 5-вольтовых ТТЛ-микросхем

Если в цифровом устройстве одновременно используются микросхемы с 5- и 3-вольтовым питанием, то, кроме согласования микросхем по току, требуется согласовать их по логическим уровням (рис. 4.2). Выходное напряжение современных 3-вольтовых микросхем, таких как SN74LVT, совпадает с ТТЛ-уровнями нуля и единицы, поэтому они могут быть непосредственно нагружены на ТТЛ-микросхемы с 5-вольтовым питанием.

Более того! Входные каскады большинства 3-вольтовых микросхем (например, серии SN74ALVT или SN74ALVC производства фирмы Texas Instruments) спроектированы так, что они выдерживают 5-вольтовое напряжение на входе. Вывод: микросхемы с 3- и 5-вольтовым питанием в ряде случаев можно соединять непосредственно (DATASHEETS фирмы Texas Instruments).

Будьте внимательны! Если в технических условиях (DATASHEETS) 3-вольтовой микросхемы не оговорено, что она выдерживает на входе 5-вольтовый потенциал, то вы можете вызвать "защелкивание" микросхемы и, как следствие, выход ее из строя.

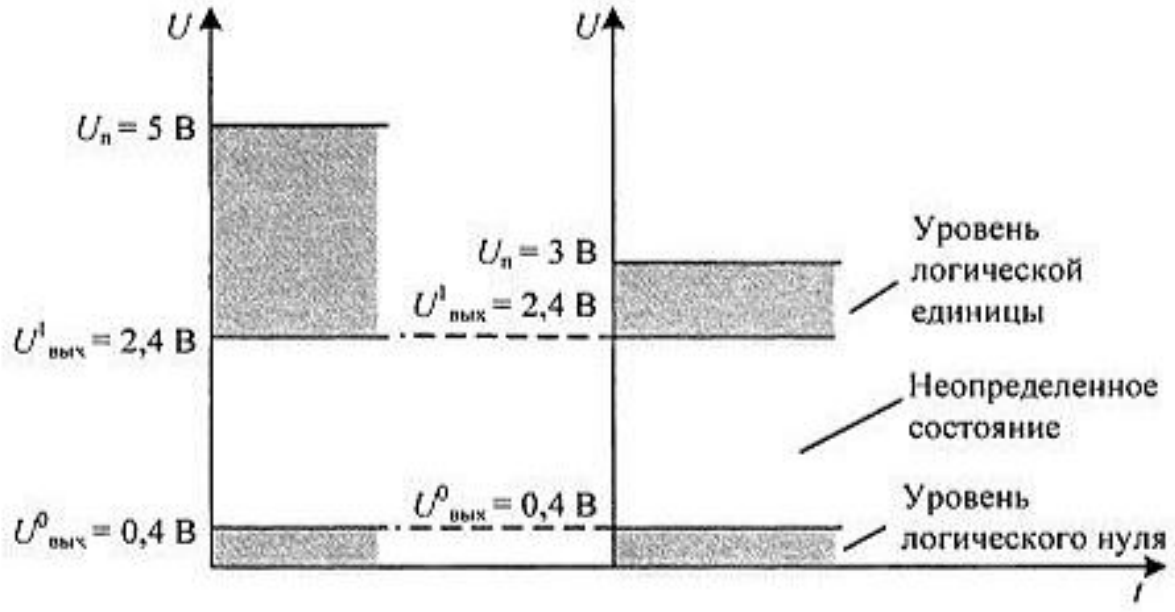


Рис. 4.2. Логические уровни микросхем с 5- и 3-вольтовым питанием

Согласование 3-вольтовых ТТЛ-микросхем и 2,5-вольтовых КМОП-микросхем

В настоящее время ТТЛ-микросхемы уже не развиваются. Практически все современные микросхемы выполнены по КМОП-технологии. Это же относится и к 2,5-вольтовым микросхемам. Порог срабатывания данных микросхем равен приблизительно 1,2 В. На рис. 4.3 приведены выходные уровни 3-вольтовых и входные уровни 2,5-вольтовых микросхем.

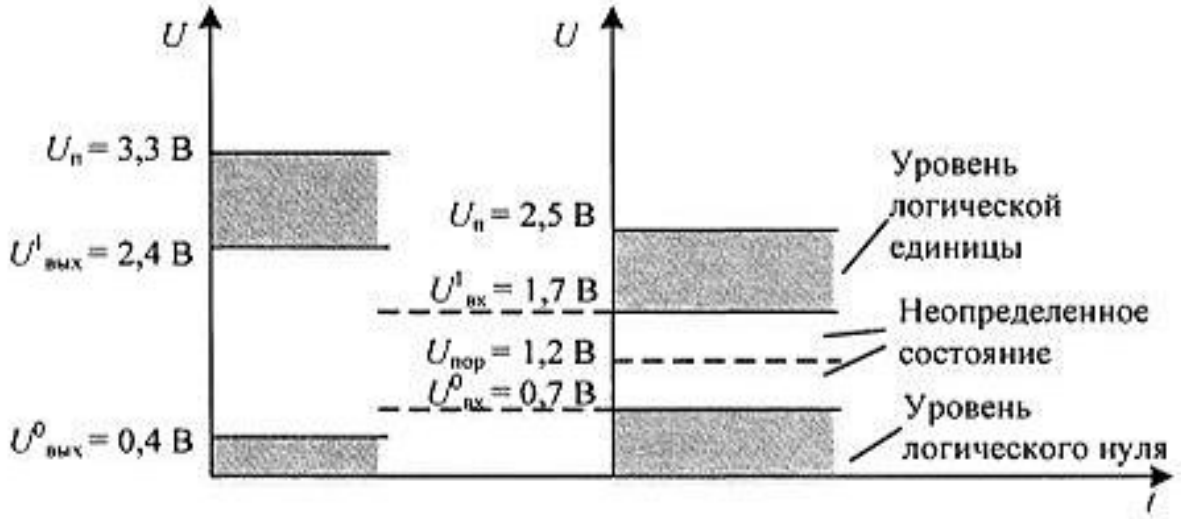


Рис. 4.3. Выходные логические уровни 3-вольтовых и входные уровни 2,5-вольтовых микросхем

Как видно из этого рисунка, 2,5-вольтовые микросхемы будут воспринимать логические уровни 3-вольтовых микросхем безошибочно. В то же самое время, по техническим условиям на часть 2,5-вольтовых микросхем, таких как SN74ALVC или SN74ALVT, входное напряжение может достигать 3,6 вольта. Для других микросхем следует свериться с техническим описанием (DATASHEET). Его можно скачать с сайта фирмы-производителя микросхем в виде PDF-файла. Это очень важно, т. к. превышение напряжения на входе микросхемы может вызвать эффект "защелкивания" и, как следствие, выход цифровой микросхемы из строя.

Похожая ситуация наблюдается и при обратном направлении прохождения сигнала (от 2,5-вольтовых микросхем к 3-вольтовым). На рис. 4.4 приведены выходные уровни 2,5-вольтовых КМОП и входные уровни 3-вольтовых ТТЛ-микросхем.

При анализе рисунков видно, что выходные уровни 2,5-вольтовой микросхемы находятся внутри границ входного напряжения 3,3-вольтовой микросхемы. Вывод: микросхемы с 3- и 2,5-вольтовым питанием можно соединять непосредственно.

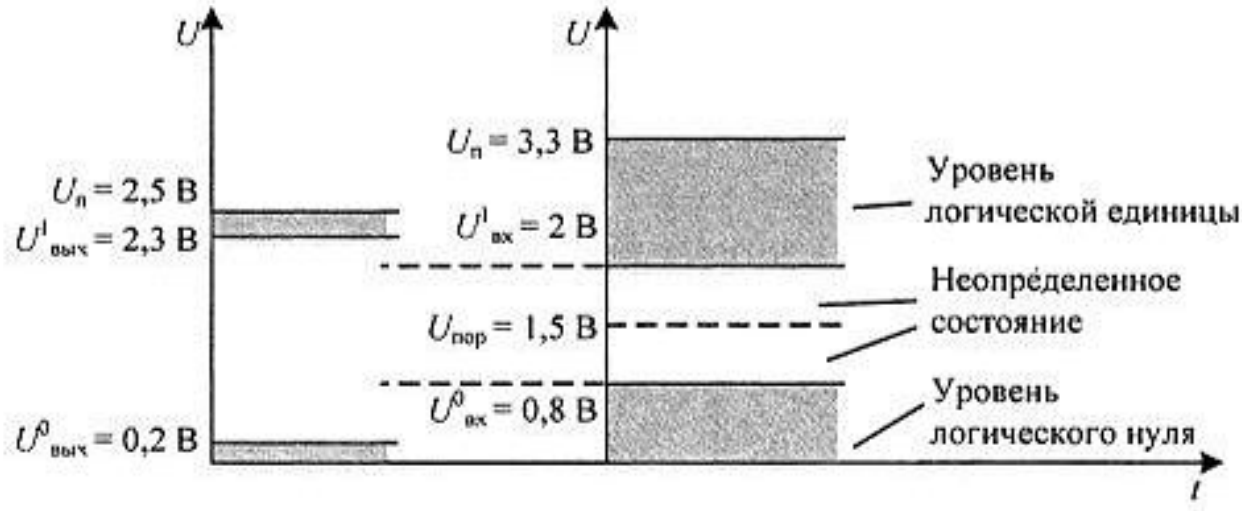


Рис. 4.4. Выходные логические уровни 2,5-вольтовых и входные уровни 3-вольтовых микросхем

Регенерация цифрового сигнала

Микросхемы соединяются между собой печатными проводниками или плоскими кабелями. При прохождении цифрового сигнала по длинным проводникам он неизбежно искажается. В основном это выражается в затягивании фронтов или возникновении в момент изменения логического уровня затухающих колебаний. Поэтому на приемном конце цифровой сигнал приходится восстанавливать. Для определения минимальной длины проводника, при которой следует учитывать переходные процессы, существует эмпирическая формула $l = 5\text{ см} \times t_{фр}$, где $t_{фр}$ выражается в наносекундах.

Примерная форма сигнала на входе цифровой микросхемы приведена на рис. 4.5. Кроме того, в ряде случаев приходится подавать на вход цифрового устройства обычные аналоговые сигналы (например, с выхода приемника).

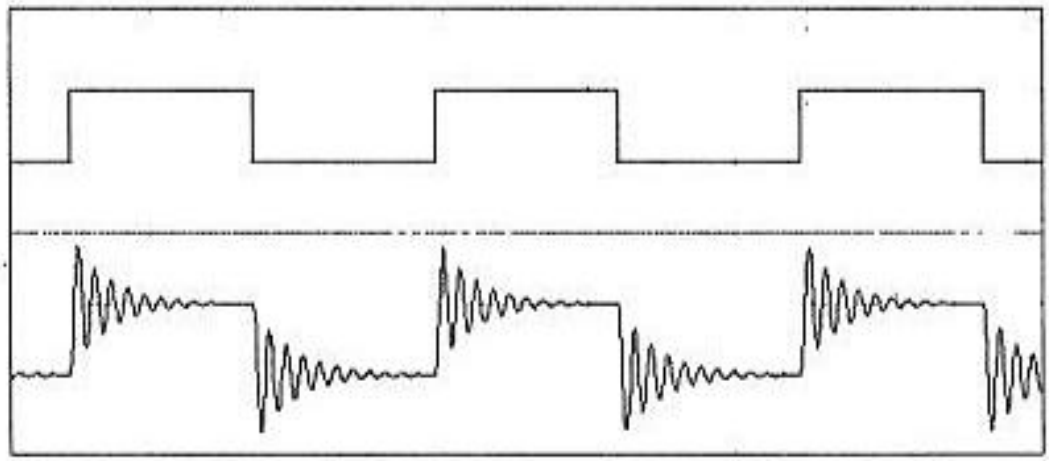


Рис. 4.5. Пример сигнала на входе и выходе печатного проводника (входе цифровой микросхемы)

Как видно из приведенного рисунка, сигнал на входе микросхемы может принимать любые значения, в том числе и запрещенные для цифровых микросхем. Как уже обсуждалось ранее, это может привести к выходу цифровых микросхем из строя. Кроме того, наличие глубоких провалов входного сигнала, обусловленных переходным процессом, может привести (и часто приводит) к возникновению импульсных помех на выходе приемной микросхемы.

Для того чтобы можно было обрабатывать такие сигналы, применяются специальные схемы, восстанавливающие исходные логические уровни сигнала, такие как триггеры Шмитта. Триггер Шмитта представляет собой устройство, охваченное положительной обратной связью. Наличие положительной обратной связи приводит к практически мгновенному изменению напряжения на выходе схемы при превышении входным сигналом порогового напряжения. Принципиальная схема триггера Шмитта, построенная на двух инверторах, приведена на рис. 4.6.

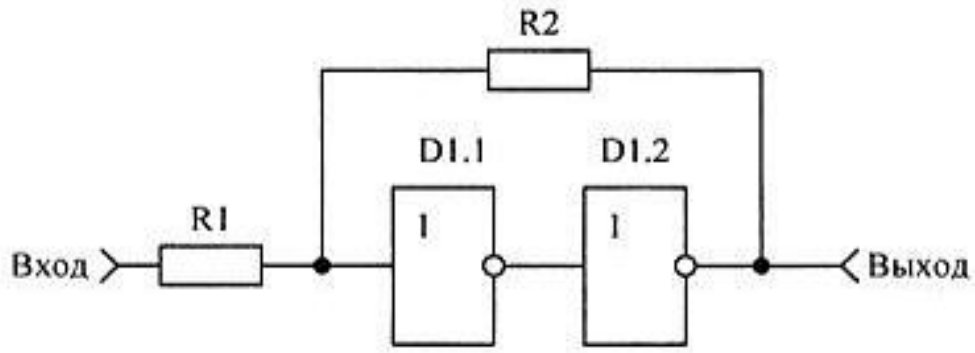


Рис. 4.6. Принципиальная схема триггера Шмитта

В этой схеме положительная обратная связь образуется двумя резисторами. Глубина обратной связи определяется соотношением между этими резисторами. То, что часть сигнала с выхода схемы триггера Шмитта подается на ее вход, приводит к тому, что вместо одного порога у нее образуется два. Один порог называется порогом срабатывания схемы (когда на выходе триггера Шмитта формируется единичный уровень). Второй порог называется порогом отпускания (когда на выходе триггера Шмитта формируется нулевой уровень). Из-за наличия двух порогов триггер Шмитта имеет еще одно название — схема с гистерезисом.

Наличие двух порогов отчетливо видно на рис. 4.7, где на вход триггера Шмитта подано синусоидальное напряжение. Входной и выходной сигналы исследуемой схемы на этом рисунке совмещены. В результате такого совмещения сигналов пороги срабатывания триггера Шмитта можно определить по точкам пересечения синусоиды и выходного сигнала.

Благодаря двум порогам схема нечувствительна к шумам на ее входе. Ведь если триггер Шмитта перешел в другое состояние, то для того, чтобы вернуть

его в прежнее состояние, нужно приложить напряжение, превышающее разность его порогов. Такое полезное свойство триггера Шмитта привело к его широкому использованию в схемах, подверженных влиянию шумов, таких как, например, шумоподавители ЧМ-приемников.

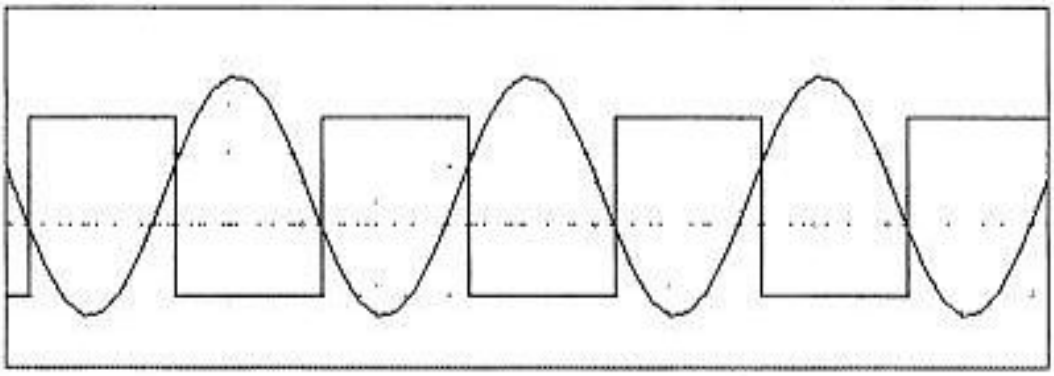


Рис. 4.7. Преобразование синусоидального сигнала в логический при помощи триггера Шмитта

В качестве примера можно привести сигнал на выходе компаратора (устройство с одним порогом срабатывания) при воздействии точно такого же синусоидального сигнала, как и на рис. 4.7. Эти сигналы приведены на рис. 4.8. Как видно из этого рисунка, в момент пересечения синусоидальным сигналом порогового уровня компаратора на его выходе появляются усиленные шумы входного сигнала. Это приводит к формированию лишних импульсов на выходе схемы, что не всегда приемлемо для правильной работы цифрового устройства в целом.

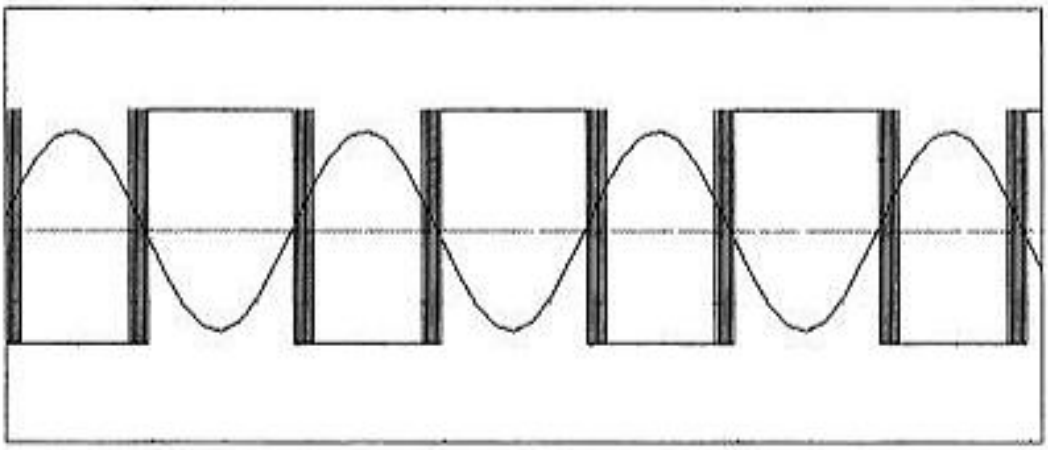


Рис. 4.8. Преобразование синусоидального сигнала в логический при помощи компаратора

Следует отметить, что наличие двух порогов не приводит к изменению логики работы цифровых устройств. Посмотрите внимательно на сигналы, приведенные на рис. 4.7. Если сдвинуть выходной сигнал относительно входного,

то точки их пересечения совместятся на одном уровне, т. е. выходной сигнал триггера Шмитта можно рассматривать просто как задержанный относительно входа усиленный и ограниченный сигнал.

Еще одно применение триггеры Шмитта нашли в качестве входных каскадов в системных шинах микропроцессоров. Мы помним, что входы цифровых микросхем нельзя "бросать" в воздухе, однако при работе на шину обязательным условием является возможность отключения источников цифровых сигналов от шины. Для того чтобы при этом входы цифровых микросхем не оставались в воздухе, все проводники в шине подключают к источнику питания или к корпусу при помощи внешних резисторов. Применение в качестве входных каскадов, подключенных к системной шине, триггеров Шмитта позволяет избавиться от этих внешних резисторов.

Перечисленные выше причины привели к широкому распространению триггеров Шмитта. Условно-графическое изображение триггера Шмитта приведено на рис. 4.9. Символ, изображенный на рабочем поле этого логического элемента, говорит о наличии гистерезиса (разности порогов срабатывания).

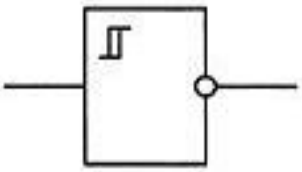


Рис. 4.9. Условно-графическое обозначение триггера Шмитта

В настоящее время промышленностью производится достаточно много микросхем, в которых содержится сразу несколько триггеров Шмитта. Пороги срабатывания в этих схемах установлены заранее. Например, в микросхеме 555ТЛ2 содержится сразу шесть триггеров Шмитта с разносом порогов 800 мВ.

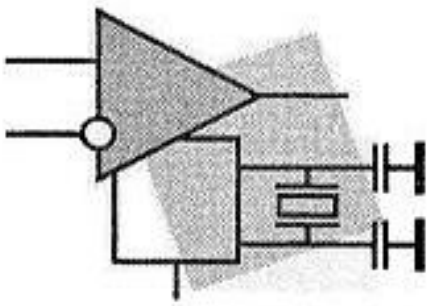
В КМОП-микросхемах пороги срабатывания и отпускания устанавливаются на трети напряжения питания. Примером подобной микросхемы может служить КМОП-микросхема К1561ТЛ1, в которой содержится четыре логических элемента "2И", каждый вход которого обладает гистерезисом.

Кроме того, в настоящее время широко распространены микросхемы малой логики, где в одном очень маленьком корпусе, обычно с пятью выводами, размещается одиночный логический элемент. В качестве примера одиночного триггера Шмитта можно назвать микросхемы SN74AHC1G14 (триггер Шмитта с инверсией) или SN74LVC1G17 (триггер Шмитта без инверсии).

Итоги

В данной главе мы рассмотрели основные параметры цифровых микросхем. Проектируя цифровое устройство, следует всегда помнить об этих параметрах. Несмотря на то, что разработку цифровых устройств можно в значительной степени формализовать, что и будет показано в последующих главах, пренебрежение нагрузочной способностью цифровых микросхем или несовпадение логических уровней микросхем, входящих в состав устройства, может привести к сбоям в его работе, а в особо неблагоприятных случаях и к выходу из строя элементов его схемы. Кроме того, при расчете цифровых схем следует учитывать особенности входных и выходных каскадов соединяемых микросхем. Именно поэтому во всех справочниках по цифровым микросхемам обязательно приводятся схемы этих каскадов.

ГЛАВА 5



Арифметические основы цифровой техники

В цифровых устройствах приходится иметь дело с обработкой различных видов информации. Это может быть в чистом виде двоичная информация, такая как включен прибор или выключен, исправно устройство или нет. Полезная информация может быть представлена в виде текстов, и тогда приходится буквы алфавита, цифры и вспомогательные символы кодировать при помощи двоичного представления. Достаточно часто полезная информация может представлять собой числа. Числа могут быть представлены в различных системах счисления. Форма записи чисел в них существенно различается между собой, поэтому, прежде чем перейти к особенностям представления чисел в цифровой технике, рассмотрим их запись в различных системах счисления.

Системы счисления

Начнем с определения системы счисления. Система счисления — это совокупность правил записи чисел цифровыми знаками. Системы счисления бывают позиционные и непозиционные. В настоящее время и в технике, и в быту широко используются как позиционные, так и непозиционные системы счисления. Сначала рассмотрим примеры непозиционных систем счисления.

В качестве классического примера непозиционной системы счисления обычно приводят римскую форму записи чисел. Тем не менее, это не единственная непозиционная система счисления, используемая в настоящее время. Сейчас, как и в глубокой древности, для записи числа используются так называемые "палочки". Эта форма записи чисел наиболее понятна и требует для записи числа всего один символ. Записанное число образуется суммой этих "палочек". Однако при записи больших чисел при помощи такой системы счисле-

ния возникают неудобства. Число получается громоздким и его трудно читать.

В следующем варианте непозиционной системы счисления для записи числа используется несколько символов (цифр). Каждая цифра обозначает определенное (различное) количество единиц. Результирующее число точно так же, как и в предыдущем варианте, образуется суммой отдельных цифр. Наиболее яркий вариант использования такой системы счисления — это денежные отношения. Мы с ними сталкиваемся каждый день. В процессе товарно-денежных отношений никому не приходит в голову, что сумма, которую мы выкладываем за товар, может зависеть от того, в каком порядке мы расположим монеты на столе! Номинал монеты или банкноты не зависит от того, в каком порядке она была вынута из кошелька. Это классический пример непозиционной системы счисления.

Однако чем большее число требуется представить в такой системе счисления, тем большее количество цифр необходимо для этого. Позиционные системы счисления были придуманы относительно недавно для того, чтобы сэкономить количество цифр, используемое для записи больших чисел.

Значение цифры в позиционной системе счисления зависит от ее позиции в записываемом числе. В позиционной системе счисления выделяются два очень важных понятия — основание системы счисления и вес цифры. Дело в том, что в позиционной системе счисления число представляется в виде формулы разложения:

$$A_p = a_n p^n + a_{n-1} p^{n-1} + \ldots + a_2 p^2 + a_1 p^1 + a_0 p^0 + a_{-1} p^{-1} + a_{-2} p^{-2} + \ldots + a_{-k} p^{-k},$$

где p — основание системы счисления;

p^i — вес единицы данного разряда;

a_i — цифры, разрешенные в данной системе счисления.

При этом количество цифр в системе счисления зависит от ее основания — p. Количество цифр равно основанию системы счисления. В двоичной системе счисления используются две цифры, в десятичной — десять, а в шестнадцатеричной — шестнадцать цифр. Число в любой позиционной системе счисления записывается в виде последовательности цифр, без уточнения веса разрядов:

$$A = a_n a_{n-1} \ldots a_2 a_1 a_0, a_{-1} a_{-2} \ldots a_{-k},$$

где a_i — цифры данной системы счисления, а цифра, соответствующая единицам, определяется по положению десятичной запятой (или десятичной точки в англоязычных странах). Каждая цифра, использованная в записи числа, называется разрядом.

Какие же системы счисления применяются в настоящее время? Наиболее ожидаемый ответ — это десятичная система счисления. Однако мы широко используем и другие системы счисления. Достаточно посмотреть на часы. Сколько минут помещается в часе? Шестьдесят! Сколько секунд помещается в минуте — тоже шестьдесят! Налицо признаки шестидесятеричной системы счисления. Это наследование древней вавилонской системы счисления, которую вместе с компасом и часами европейцы заимствовали от арабов.

Также существует множество других примеров. Картушка компаса делится на восемь румбов. Чем не восьмеричная система счисления? Относительно недавно в России отказались от полушек (четверть копейки) и грошей (половина копейки), а следующее значение монеты — две копейки. Чем не двоичная система счисления?

Ну а теперь подробнее остановимся на системах счисления, наиболее часто используемых в цифровой технике.

Десятичная система счисления

Основание этой системы счисления (p) равно десяти. В этой системе счисления используется десять цифр. В настоящее время для обозначения этих цифр используются символы 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Число в десятичной системе счисления записывается как сумма единиц, десятков, сотен, тысяч и т. д. То есть веса соседних разрядов различаются в десять раз. Точно так же записываются и числа, меньшие единицы. В этом случае разряды числа будут называться как десятые, сотые или тысячные доли единицы.

Рассмотрим пример записи десятичного числа. Для того чтобы показать, что в примере используется именно десятичная система счисления, применяем индекс 10. Если же кроме десятичной формы записи чисел не предполагается использования никакой другой, то индекс обычно не применяется:

$$A_{10} = 247{,}56_{10} = 2\times10^{2} + 4\times10^{1} + 7\times10^{0} + 5\times10^{-1} + 6\times10^{-2} =$$
$$= 200_{10} + 40_{10} + 7_{10} + 0{,}5_{10} + 0{,}06_{10}$$

В приведенной записи самый старший разряд числа называется сотнями. В данном примере сотням соответствует цифра 2. Следующий разряд называется десятками. В рассмотренном примере десяткам соответствует цифра 4. Следующий разряд называется единицами, и в рассматриваемом примере единицам соответствует цифра 7, десятым долям единицы соответствует цифра 5, а сотым — 6.

Арифметические операции в различных системах счисления выполняются на основании таблиц сложения и умножения. Таблицы сложения и умножения

для десятичной системы счисления прекрасно известны читателю, поэтому здесь рассматриваться не будут.

Двоичная система счисления

Основание этой системы счисления (p) равно двум. В этой системе счисления используется всего две цифры. Чтобы не выдумывать новых символов для обозначения цифр, в двоичной системе счисления были использованы символы десятичных цифр 0 и 1. Для того чтобы не спутать систему счисления, в записи двоичного числа используется индекс 2. Если же кроме двоичной формы записи чисел не предполагается использования никакой другой, то этот индекс можно опустить.

Число в двоичной системе счисления записывается как сумма единиц, двоек, четверок, восьмерок и т. д. Это означает, что веса соседних разрядов различаются ровно в два раза. Точно так же записываются и числа, меньшие единицы. В этом случае разряды числа будут называться как половины, четверти или восьмые доли единицы.

Рассмотрим пример записи двоичного числа:

$$A_2 = 101110{,}101_2 =$$

$$= 1\times 2^5 + 0\times 2^4 + 1\times 2^3 + 1\times 2^2 + 1\times 2^1 + 0\times 2^0 + 1\times 2^{-1} + 0\times 2^{-2} + 1\times 2^{-3} =$$

$$= 32_{10} + 8_{10} + 4_{10} + 2_{10} + 0{,}5_{10} + 0{,}125_{10} = 46{,}625_{10}$$

При записи во второй строке примера десятичных эквивалентов двоичных разрядов мы не стали записывать степени двойки, которые умножаются на ноль, т. к. это привело бы только к загромождению формулы и, как следствие, затруднению понимания материала.

Недостатком двоичной системы счисления можно считать большое количество разрядов, требующихся для записи чисел. В качестве преимущества этой системы счисления можно назвать простоту выполнения арифметических действий. Приведем правила операций сложения и умножения в двоичной системе счисления:

$0+0=0$	$0\times 0=0$
$0+1=1$	$0\times 1=0$
$1+0=1$	$1\times 0=0$
$1+1=10$	$1\times 1=1$

А теперь попробуем выполнить несколько операций в двоичной системе счисления. В качестве примера возьмем десятичные числа 5_{10} и $3{,}5_{10}$. Сначала преобразуем их в двоичную форму:

$$A = 5_{10} = 101_2 = 1\times 2^2 + 0\times 2^1 + 1\times 2^0 = 4_{10} + 1_{10}$$

$$B = 3{,}5_{10} = 11{,}1_2 = 1\times 2^1 + 1\times 2^0 + 1\times 2^{-1} = 2_{10} + 1_{10} + 0{,}5_{10}$$

Затем выполним суммирование этих чисел в двоичной системе счисления. Суммирование будем выполнять в "столбик". Так будет легче понять производимые арифметические действия:

$$\begin{array}{r} 101{,}0 \\ + \\ 011{,}1 \\ \hline 1000{,}1 \end{array}$$

При суммировании единичных разрядов возникает перенос в разряд двоек. В свою очередь суммирование двоек и прибавление к ним переноса из предыдущего разряда создает перенос в разряд четверок. Точно такая же ситуация и в разряде четверок. А теперь для проверки переведем получившийся результат в десятичную форму:

$$1000{,}1_2 = 1\times 2^3 + 0\times 2^2 + 0\times 2^1 + 0\times 2^0 + 1\times 2^{-1} = 8_{10} + 0{,}5_{10} = 8{,}5_{10}$$

Как видно из получившегося числа, результаты сложения совершенно одинаковы. То есть при выполнении суммирования в двоичной системе счисления ошибки не произошло. Теперь выполним операцию двоичного вычитания. Вычтем из числа 5_{10} число $3{,}5_{10}$. В результате выполнения операции мы ожидаем получить десятичное число 1,5.

$$\begin{array}{r} 101{,}0 \\ - \\ 011{,}1 \\ \hline 001{,}1 \end{array}$$

При вычитании разряда половинок сразу же возникает необходимость заема из старшего разряда. Если из единицы вычесть число 0,5, то в качестве результата получим тоже половину единичного разряда. Записываем на место половинок единицу. Аналогичная ситуация после заема единицы возникает и в разряде единиц. Придется занять двойку. В результате операции вычитания на месте единиц тоже записываем единицу. В разряде двоек вычитаем из единицы единицу. В результирующее число на место двоек записываем ноль. В разряде четверок после заема остался ноль. В результате вычитания нуля из нуля получаем ноль. Его и запишем на место четверок. Ну а теперь проверим, что же мы получили. Для этого преобразуем число из двоичной формы в десятичную:

$$001{,}1_2 = 1\times 2^0 + 1\times 2^{-1} = 1_{10} + 0{,}5_{10} = 1{,}5_{10}$$

Следующей рассмотрим операцию двоичного умножения. Умножать будем те же самые числа (5_{10} и $3{,}5_{10}$). В результате операции умножения ожидаем получить число $17{,}5_{10}$. Умножение произведем в "столбик". Умножение в "столбик" в двоичной системе выполняется точно так же, как и в десятичной системе счисления.

$$\begin{array}{r} 11{,}1 \\ \times\ 101 \\ \hline 111 \\ 000 \\ 111 \\ \hline 10001{,}1 \end{array}$$

Первое частичное произведение образуется при умножении младшего разряда множителя на множимое. В этом разряде записана единица. То есть в качестве частного произведения просто записываем само множимое. Во втором разряде множителя записан ноль. То есть в результате умножения получаем нулевое значение. Его сдвигаем на один двоичный разряд (как в десятичной системе счисления). Третье частичное произведение, как и первое, будет равно множимому. Его сдвигаем на два двоичных разряда. В конце операции умножения, как и в десятичной системе счисления, суммируем все частичные произведения.

Снова проверяем полученный результат. Для этого, как и в предыдущем случае, преобразуем число из двоичной формы в десятичную:

$$10001{,}1_2 = 1\times 2^4 + 0\times 2^3 + 0\times 2^2 + 0\times 2^1 + 1\times 2^0 + 1\times 2^{-1} =$$
$$= 16_{10} + 1_{10} + 0{,}5_{10} = 17{,}5_{10}$$

Восьмеричная система счисления

Основание этой системы счисления (p) равно восьми. Восьмеричную систему счисления можно рассматривать как более короткий вариант записи двоичных чисел, т. к. число восемь является степенью числа два. В восьмеричной системе счисления используется восемь цифр. Чтобы не вводить новых символов для обозначения цифр, в восьмеричной системе счисления используются символы десятичных цифр 0, 1, 2, 3, 4, 5, 6 и 7. Для того чтобы не спутать систему счисления, в записи восьмеричного числа используется индекс 8. Если же кроме восьмеричной формы записи чисел не предполагается использования никакой другой, то этот индекс обычно опускается.

Число в этой системе счисления записывается как сумма единиц, восьмерок, шестьдесятчетверок и т. д. То есть веса соседних разрядов различаются в восемь раз. Точно так же записываются и числа, меньшие единицы. В этом слу-

чае разряды числа будут называться как восьмые, шестьдесят четвертые (и т. д.) доли единицы.

Рассмотрим пример записи восьмеричного числа:

$$A_8 = 125,46_8 = 1\times 8^2 + 2\times 8^1 + 5\times 8^0 + 4\times 8^{-1} + 6\times 8^{-2} =$$

$$= 64_{10} + 16_{10} + 5_{10} + \frac{4_{10}}{8_{10}} + \frac{6_{10}}{64_{10}} = 85,59375_{10}$$

Во второй строке приведенного примера фактически осуществлен перевод числа, записанного в восьмеричной форме в десятичное представление того же самого числа. На приведенных примерах мы фактически рассмотрели один из способов преобразования чисел из одной формы представления в другую.

Так как в формуле используются простые дроби, то возможен вариант такой ситуации, при которой точный перевод из одной формы представления в другую становится невозможным. В этом случае ограничиваются заданным количеством дробных разрядов.

Приведем таблицы операций сложения (табл. 5.1) и умножения (табл. 5.2) в восьмеричной системе счисления.

***Таблица 5.1.** Таблица сложения восьмеричных чисел*

+	1	2	3	4	5	6	7
1	2	3	4	5	6	7	10
2	3	4	5	6	7	10	11
3	4	5	6	7	10	11	12
4	5	6	7	10	11	12	13
5	6	7	10	11	12	13	14
6	7	10	11	12	13	14	15
7	10	11	12	13	14	15	16

А теперь для иллюстрации действий в восьмеричной системе счисления выполним несколько операций. В качестве примера возьмем те же самые числа, что и в предыдущем случае — 5_{10} и $3,5_{10}$. Сначала преобразуем их в восьмеричную форму:

$$A = 5_{10} + 5_8$$

$$B = 3,5_{10} = 3,4_8 = 3\times 8^0 + 4\times 8^{-1} = 3_{10} + \frac{4_{10}}{8_{10}}$$

Таблица 5.2. *Таблица умножения восьмеричных чисел*

+	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7
2	2	4	6	10	12	14	16
3	3	6	11	14	17	22	25
4	4	10	14	20	24	30	34
5	5	12	17	24	31	36	43
6	6	14	22	30	36	44	52
7	7	16	25	34	43	52	61

Затем выполним суммирование этих чисел в восьмеричной системе счисления. В результате суммирования ожидаем получить число $8,5_{10}$. Суммирование будем выполнять в "столбик". Так будет легче понять арифметические действия:

$$\begin{array}{r} 5,0 \\ +\ 3,4 \\ \hline 10,4 \end{array}$$

При суммировании единичных разрядов возникает перенос в разряд восьмерок. Для проверки переведем получившийся результат в десятичную форму:

$$10,4_8 = 1\times 8^1 + 0\times 8^0 + 4\times 8^{-1} = 8_{10} + 0,5_{10} = 8,5_{10}$$

Как видно из полученного десятичного числа, результат совпадает с ожидаемым. То есть при выполнении суммирования в восьмеричной системе счисления ошибки не произошло.

Теперь выполним операцию восьмеричного вычитания. Вычтем из числа 5_{10} число $3,5_{10}$. В результате выполнения этой операции мы ожидаем получить десятичное число 1,5:

$$\begin{array}{r} 5,0 \\ -\ 3,4 \\ \hline 1,4 \end{array}$$

При вычитании разряда восьмых частей сразу же возникает необходимость "заема" из старшего разряда. Если из единицы вычесть число $0,4_8$, то в качестве результата получим тоже число $0,4_8$. Записываем на место восьмых частей четверку. В разряде единиц из оставшейся после "заема" цифры 4 вычтем цифру 3.

Ну а теперь проверим, что же мы получили. Для этого преобразуем число из восьмеричной формы в десятичную:

$$1,4_8 = 1\times 8^0 + 4\times 8^{-1} = 1_{10} + 0,5_{10} = 1,5_{10}$$

Выполним операцию восьмеричного умножения. Умножать будем те же самые числа (5_{10} и $3,5_{10}$). В результате операции умножения мы ожидаем получить число $17,5_{10}$. Умножение произведем в "столбик". Умножение в столбик в восьмеричной системе счисления выполняются точно так же, как и в десятичной системе:

$$\begin{array}{r} 5,0 \\ \times\ 3,4 \\ \hline 240 \\ 170 \\ \hline 21,40 \end{array}$$

Первое частичное произведение образуется при умножении младшего разряда множителя на множимое. При умножении цифры 4 на цифру 0 получаем 0. Записываем младший разряд частичного произведения. Результат умножения цифры 4 на цифру 5 определяем по табл. 5.2.

Точно так же получаем и второе частичное произведение. Его мы сдвигаем на один восьмеричный разряд (как в десятичной системе счисления) и точно так же, как и в десятичной системе счисления, в конце операции умножения мы суммируем все частичные произведения.

Снова проверяем, что же мы получили. Для этого преобразуем число из восьмеричной формы в десятичную:

$$21,4_8 = 2\times 8^1 + 1\times 8^0 + 4\times 8^{-1} = 16_{10} + 1_{10} + 0,5_{10} = 17,5_{10}$$

Все в порядке, результаты вычислений в различных системах счисления совпадают.

Шестнадцатеричная система счисления

Основание системы счисления (p) равно шестнадцати. В ней используется шестнадцать цифр. Эту систему счисления можно считать еще одним вариантом записи двоичного числа. В ней уже не хватает десяти цифр, поэтому дополнительно были введены шесть символов. Для обозначения этих цифр используются шесть первых букв латинского алфавита. При записи шестнадцатеричного числа неважно, буквы верхнего или нижнего регистра будут использоваться в качестве цифр. Таким образом, в качестве цифр в шест-

надцатеричной системе используются следующие символы 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Так как в шестнадцатеричной системе счисления появляются новые цифры, то приведем таблицу соответствия цифр этой системы их десятичным эквивалентам. В табл. 5.3 в левом столбце приведены шестнадцатеричные символы, а справа — их десятичные эквиваленты.

Таблица 5.3. *Таблица соответствия шестнадцатеричных цифр десятичным значениям*

Шестнадцатеричная цифра	Десятичный эквивалент
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
A	10
B	11
C	12
D	13
E	14
F	15

Число в этой системе счисления записывается как сумма единиц, чисел шестнадцать, двести пятьдесят шесть и т. д. То есть веса соседних разрядов различаются в шестнадцать раз. Точно так же записываются и числа, меньшие единицы. В этом случае разряды числа будут называться как шестнадцатые, двести пятьдесят шестые и т. д. доли единицы.

Рассмотрим пример записи шестнадцатеричного числа:

$$A_{16} = 2\text{AF},\text{C4}_{16} = 2\times16^2 + 10\times16^1 + 15\times16^0 + 12\times16^{-1} + 4\times16^{-2} =$$

$$= 512_{10} + 160_{10} + 15_{10} + \frac{12_{10}}{16_{10}} + \frac{4_{10}}{256_{10}} = 687,765625_{10}$$

Приведем таблицы операций сложения (табл. 5.4) и умножения в шестнадцатеричной системе счисления.

Для иллюстрации действий в шестнадцатеричной системе счисления выполним несколько операций. В качестве примера возьмем те же самые числа, что и в предыдущем случае, а именно 5_{10} и $3,5_{10}$. Сначала преобразуем их в шестнадцатеричную форму:

$$A = 5_{10} = 5_{16}$$

$$B = 3,5_{10} = 3,8_{16} = 3\times16^0 + 8\times16^{-1} = 3_{10} + \frac{8_{10}}{16_{10}}$$

Таблица 5.4. *Таблица сложения шестнадцатеричных чисел*

+	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
1	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	10
2	2	3	4	5	6	7	8	9	a	b	c	d	e	f	10	11
3	3	4	5	6	7	8	9	a	b	c	d	e	f	10	11	12
4	4	5	6	7	8	9	a	b	c	d	e	f	10	11	12	13
5	5	6	7	8	9	a	b	c	d	e	f	10	11	12	13	14
6	6	7	8	9	a	b	c	d	e	f	10	11	12	13	14	15
7	7	8	9	a	b	c	d	e	f	10	11	12	13	14	15	16
8	8	9	a	b	c	d	e	f	10	11	12	13	14	15	16	17
9	9	a	b	c	d	e	f	10	11	12	13	14	15	16	17	18
a	a	b	c	d	e	f	10	11	12	13	14	15	16	17	18	19
b	b	c	d	e	f	10	11	12	13	14	15	16	17	18	19	1a
c	c	d	e	f	10	11	12	13	14	15	16	17	18	19	1a	1b
d	d	e	f	10	11	12	13	14	15	16	17	18	19	1a	1b	1c
e	e	f	10	11	12	13	14	15	16	17	18	19	1a	1b	1c	1d
f	f	10	11	12	13	14	15	16	17	18	19	1a	1b	1c	1d	1e

Таблица 5.5. *Таблица умножения шестнадцатеричных чисел*

*	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
2	0	2	4	6	8	a	c	e	10	12	14	16	18	1a	1c	1e
3	0	3	6	9	c	f	12	15	18	1b	1e	21	24	27	2a	2d
4	0	4	8	c	10	14	18	1c	20	24	28	2c	30	34	38	3c
5	0	5	a	f	14	19	1e	23	28	2d	32	37	3c	41	46	4b
6	0	6	c	12	18	1e	24	2a	30	36	3c	42	48	4e	54	5a
7	0	7	e	15	1c	23	2a	31	38	3f	46	4d	54	5b	62	69
8	0	8	10	18	20	28	30	38	40	48	50	58	60	68	70	78
9	0	9	12	1b	24	2d	36	3f	48	51	5a	63	6c	75	7e	87
a	0	a	14	1e	28	32	3c	46	50	5a	64	6e	78	82	8c	96
b	0	b	16	21	2c	37	42	4d	58	63	6e	79	84	8f	9a	a5
c	0	c	18	24	30	3c	48	54	60	6c	78	84	90	9c	a8	b4
d	0	d	1a	27	34	41	4e	5b	68	75	82	8f	9c	a9	b6	c3
e	0	e	1c	2a	38	46	54	62	70	7e	8c	9a	a8	b6	c4	d2
f	0	f	1e	2d	3c	4b	5a	69	78	87	96	a5	b4	c3	d2	e1

Затем выполним суммирование этих чисел в шестнадцатеричной системе счисления. В результате суммирования ожидаем получить число $8,5_{10}$. Суммирование будем выполнять в "столбик". Так будет легче понять арифметические действия.

$$\begin{array}{r} 5,0 \\ + \\ 3,8 \\ \hline 8,8 \end{array}$$

Для проверки переведем получившийся при выполнении операции суммирования результат в десятичную форму:

$$8,8_{16} = 8 \times 16^{0} + 8 \times 16^{-1} = 8_{10} + 0,5_{10} = 8,5_{10}$$

Как видно из полученного десятичного числа, результаты совпадают. То есть при выполнении суммирования в шестнадцатеричной системе счисления ошибки не произошло.

Теперь выполним операцию шестнадцатеричного вычитания. Вычтем из числа 5_{10} число $3{,}5_{10}$. В результате выполнения этой операции мы ожидаем получить десятичное число 1,5.

$$\begin{array}{r} 5,0 \\ -\ 3,8 \\ \hline 1,8 \end{array}$$

При вычитании разряда шестнадцатых частей сразу же возникает необходимость "заема" из старшего разряда. Если из единицы вычесть число $0{,}8_{16}$, то в качестве результата получим тоже число $0{,}8_{16}$. Записываем на место шестнадцатых частей цифру восемь. В разряде единиц из оставшейся после "заема" цифры 4 вычтем цифру 3. Проверим полученный результат. Для этого преобразуем число из шестнадцатеричной формы в десятичную:

$$1{,}8_{16} = 1\times16^{0} + 8\times16^{-1} = 1_{10} + 0{,}5_{10} = 1{,}5_{10}$$

Выполним операцию шестнадцатеричного умножения. Умножать будем те же самые числа (5_{10} и $3{,}5_{10}$). В результате операции умножения мы ожидаем получить число $17{,}5_{10}$. Умножение произведем в "столбик". Умножение в столбик в шестнадцатеричной системе счисления выполняются точно так же, как и в десятичной системе.

$$\begin{array}{r} 5,0 \\ \times\ 3,8 \\ \hline 280 \\ F0 \\ \hline 11,80 \end{array}$$

Первое частичное произведение образуется при умножении младшего разряда множителя на множимое. При умножении цифры 8 на цифру 0 получаем 0. Записываем младший разряд частичного произведения. Результат умножения цифры 8 на цифру 5 определяем по табл. 5.5.

Точно так же получаем и второе частичное произведение, сдвигаем его на один шестнадцатеричный разряд (как в десятичной системе счисления) и точно так же, как и в десятичной системе счисления, в конце операции умножения суммируем все частичные произведения.

Снова проверяем полученный результат. Для этого преобразуем число из шестнадцатеричной формы в десятичную:

$$11{,}8_{16} = 1\times16^{1} + 1\times16^{0} + 8\times16^{-1} = 16_{10} + 1_{10} + 0{,}5_{10} = 17{,}5_{10}$$

Из приведенных примеров записи чисел в различных системах счисления вполне очевидно, что для записи одного и того же числа с одинаковой точ-

ностью в разных системах счисления требуется различное количество разрядов. Чем больше основание системы счисления, тем меньшее количество разрядов требуется для записи одного и того же числа.

Преобразование чисел из одной системы счисления в другую

При проектировании цифровых устройств достаточно часто требуется уметь переводить число из одной системы счисления в другую. Давайте научимся выполнять такие действия. Преобразование целых чисел и правильных дробей выполняется по разным правилам. В действительном числе преобразование целой и дробной части производят по отдельности.

Преобразование целой части числа

Для преобразования целочисленного значения необходимо исходное число разделить на основание новой системы счисления до получения целого остатка, который является младшим разрядом числа в новой системе счисления (единицы). Полученное частное снова делим на основание системы, и так до тех пор, пока частное не станет меньше основания новой системы счисления. *Все операции выполняются в исходной системе счисления.*

Рассмотрим для примера перевод числа из десятичной системы счисления в двоичную. Возьмем десятичное число $A_{10} = 124$ и поделим его на основание двоичной системы, т. е. на число 2. Деление будем производить "уголком", как показано на рис. 5.1. В результате первого деления получим разряд единиц (самый младший разряд). В результате второго деления получим разряд двоек. Деление продолжаем, пока результат деления больше двух. В конце операции преобразования получили двоичное число 1111100_2:

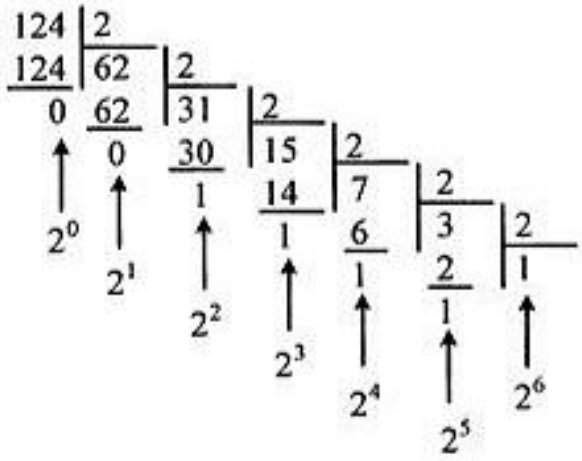


Рис. 5.1. Перевод числа из десятичной системы счисления в двоичную

Для того чтобы проверить, нет ли ошибки, преобразуем получившееся двоичное число в десятичную систему по обычной формуле разложения:

$$1\times 2^6 + 1\times 2^5 + 1\times 2^4 + 1\times 2^3 + 1\times 2^2 + 0\times 2^1 + 0\times 2^0 =$$
$$= 64_{10} + 32_{10} + 16_{10} + 8_{10} + 4_{10} = 124$$

Теперь то же самое число переведем в восьмеричную систему счисления. Для этого число 124_{10} будем делить на число 8, как показано на рис. 5.2.

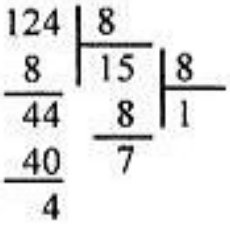


Рис. 5.2. Перевод числа из десятичной системы счисления в восьмеричную

Как видим, остаток от первого деления равен 4. То есть младший разряд восьмеричного числа содержит цифру 4 (единицы). Остаток от второго деления равен 7, т. е. второй разряд восьмеричного числа — это цифра 7. Старший разряд получился равным 1. То есть в результате многократного деления мы получили восьмеричное число 174_8.

Проверим, не ошиблись ли мы в процессе преобразования? Для этого преобразуем получившееся двоичное число в десятичную систему по обычной формуле разложения:

$$1\times 8^2 + 7\times 8^1 + 4\times 8^0 = 64_{10} + 56_{10} + 4_{10} = 124$$

А можно ли осуществить перевод из восьмеричной системы счисления в двоичную делением? Можно! Но деление нужно произвести по правилам восьмеричной арифметики. Правила работы в восьмеричной системе счисления мы рассмотрели в предыдущем разделе. Рассмотрим пример перевода в двоичную форму полученного ранее восьмеричного числа 174_8. Разделим его на основание новой системы счисления 2, как показано на рис. 5.3.

Как мы убедились, выполнять деление в восьмеричной системе очень неудобно, ведь подсознательно мы делим в десятичной системе счисления. Если обратить внимание на то, что число 8 является степенью числа 2, то можно считать восьмеричную систему счисления просто более короткой записью двоичного числа. Это означает, что для представления восьмеричной цифры можно использовать три двоичных бита ($8=2^3$). Составим таблицу соответствия для такого преобразования (табл. 5.6).

$A=174_8=1111100_2$

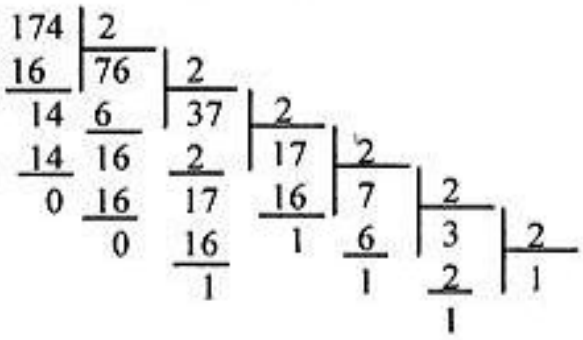


Рис. 5.3. Перевод в двоичную форму восьмеричного числа

Таблица 5.6. *Таблица соответствия восьмеричных цифр и двоичного кода*

Двоичный код	Восьмеричная цифра	Десятичный эквивалент
000	0	0
001	1	1
010	2	2
011	3	3
100	4	4
101	5	5
110	6	6
111	7	7

Используя эту таблицу, можно просто заменить каждую восьмеричную цифру тремя двоичными битами. Три двоичных бита обычно называют триадой или трибитом. Теперь переведем восьмеричное число 174_8 в двоичную форму при помощи табл. 5.6, как показано на рис. 5.4.

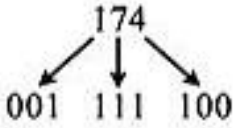


Рис. 5.4. Перевод восьмеричного числа в двоичную форму

Аналогично можно выполнить перевод числа из двоичной системы в восьмеричную. Для этого двоичное число разбивают на триады относительно крайнего правого разряда (или двоичной запятой) и, используя табл. 5.6, каждой триаде ставят в соответствие восьмеричную цифру.

Аналогичным образом можно выполнить перевод числа из шестнадцатеричной формы в двоичную и обратно. В этом случае для представления шестнадцатеричной цифры потребуется четыре двоичных разряда. Четыре двоичных разряда обычно называют тетрадой. Иногда при переводе иностранной литературы используется термин "нибл".

Составим таблицу соответствия двоичных тетрад и шестнадцатеричных цифр. Для этого будем просто прибавлять единицу к значению предыдущей строки в каждом столбце таблицы, в соответствии с используемой в этом столбце системой счисления. Результат приведен в табл. 5.7.

В качестве примера использования табл. 5.7 переведем шестнадцатеричное число $7C_{16}$ в двоичную форму представления, как показано на рис. 5.5.

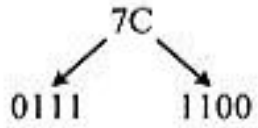


Рис. 5.5. Перевод шеетнадцатеричного числа в двоичную форму

Таблица 5.7. *Таблица соответствия шестнадцатеричных цифр и двоичного кода*

Двоичный код	Шестнадцатеричная цифра	Десятичный эквивалент
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Пример преобразования двоичного числа в восьмеричную и шестнадцатеричную форму приведен на рис. 5.6.

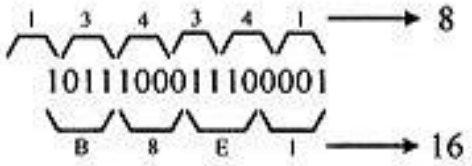


Рис. 5.6. Пример преобразования двоичного числа в восьмеричную и шестнадцатеричную форму

На этом рисунке внизу выделены двоичные тетрады и соответствующие им шестнадцатеричные цифры. Их соответствие можно проверить при помощи табл. 5.7. Сверху выделены триады и соответствующие им восьмеричные цифры. Старшая триада восьмеричного числа получилась неполной. Для того чтобы можно было бы воспользоваться табл. 5.6, ее необходимо дополнить старшими незначащими нулями.

Преобразование дробной части числа

Так как дробная часть числа меньше единицы, то ее преобразование выполняется умножением исходного числа на основание новой системы счисления. Целая часть результата умножения будет старшим разрядом числа в новой системе счисления. Дробную часть произведения снова умножают на основание системы счисления. Операция умножения выполняется до достижения требуемой точности результата. Все операции выполняют по правилам исходной системы счисления.

Для примера рассмотрим перевод дробного числа из десятичной системы счисления в двоичную. Пусть исходное число А будет равно 0,35. Выполним операцию последовательного умножения, как это показано на рис. 5.7.

В результате описанных действий получим двоичное представление числа А:

$$A = 0{,}35_{10} \approx 0{,}01011 = 0{,}34375$$

В общем случае перевод правильных дробей является приблизительным. Число разрядов в новой системе можно найти исходя из одинаковой точности представления чисел в разных системах счисления. Одинаковая точность числа, записанного в различных системах счисления, достигается при одинаковых весах младших разрядов соответствующей системы счисления. Определить вес младшего разряда числа можно по следующей формуле:

$$M = q^{-n},$$

где q — основание системы счисления.

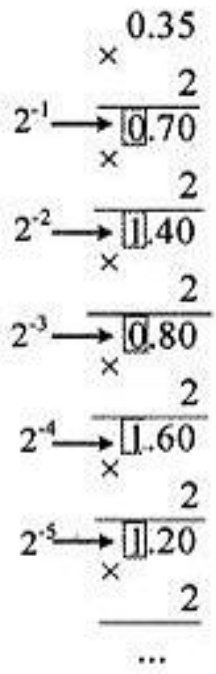


Рис. 5.7. Пример операции последовательного умножения

В предыдущем примере для десятичного числа 0.35 вес младшего разряда $M = \frac{1}{100} = 0.01$. В полученном двоичном числе вес младшего разряда будет равен $1/32 \approx 0{,}03$. То есть при операции преобразования числа в двоичную форму мы ухудшили точность представления числа в три раза. Для одинаковой точности представления чисел в различных системах счисления необходимо выполнить равенство:

$$p^{-n_p} = q^{-n_q},$$

где p и q — основания старой и новой систем счисления соответственно.

Для определения необходимого количества разрядов в новой системе счисления возьмем логарифм по основанию (p) от этого равенства:

$$-n_p = -n_q \log_p q,$$

откуда найдем требуемое количество разрядов:

$$n_q = \frac{n_p}{\log_p q}.$$

Определим необходимое число разрядов в двоичной системе счисления для рассмотренного ранее примера:

$$n_2 = \frac{n_{10}}{\lg 2} \approx \frac{n_{10}}{0{,}3}$$

В использованном нами примере количество разрядов десятичного числа после запятой $n_{10} = 2$, т. е. в двоичной системе счисления для той же точности числа количество разрядов должно быть равно:

$$n_2 = \frac{2}{0,3} \approx 7$$

Если же требуется обеспечить точность представления числа до трех разрядов после запятой $n_{10} = 3$, то количество двоичных разрядов должно быть не менее:

$$n_2 = \frac{3}{0,3} = 10$$

Определим формулы пересчета количества разрядов, требующихся для записи числа в восьмеричной и шестнадцатеричной системе счисления:

$$n_8 = \frac{n_{10}}{\lg 8} \approx \frac{n_{10}}{0,9},$$

$$n_{16} = \frac{n_{10}}{\lg 16} \approx \frac{n_{10}}{1,2}.$$

В рассмотренном выше примере для записи десятичного числа $0,35_{10}$ в восьмеричной форме с той же точностью по этой формуле потребуется три разряда. Проверим это утверждение:

$$A = 0,35_{10} \approx 0,263_8 = 2 \times 8^{-1} + 6^{-2} \times 8^{-2} + 3 \times 8^{-3} = \frac{2}{8} + \frac{6}{64} + \frac{3}{512} \approx 0,3496 \approx 0,35$$

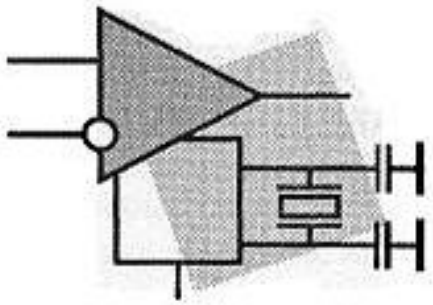
Аналогичным образом можно осуществлять перевод в любую систему счисления. Исключением является перевод из двоичной системы счисления в восьмеричную или шестнадцатеричную. Напомним, что эти системы счисления являются краткой записью двоичного числа, поэтому в этом случае можно переводить числа из одной системы счисления в другую при помощи таблиц перехода 5.6 и 5.7. Двоичное число разбивается на триады для перевода в восьмеричную систему счисления или на тетрады для перевода в шестнадцатеричную систему счисления. Разбиение начинается от двоичной запятой.

Итоги

Современные цифровые устройства проектируются на основе двоичной системы счисления. Эта система счисления отличается от десятичной системы

счисления, которую мы применяем в обыденной жизни. В данной главе мы рассмотрели арифметические основы систем счисления и способы перевода чисел из одной системы счисления в другую. Эти знания будут полезны при разработке цифровых устройств, предназначенных для обработки данных и отображения информации для человека. Однако следует помнить, что числа, записанные в двоичной, восьмеричной, десятичной и шестнадцатеричной системах счисления, не совпадают с кодами, имеющими аналогичное название, поэтому виды двоичных и десятичных кодов мы рассмотрим позднее, когда начнем изучение микропроцессорных устройств.

ГЛАВА 6



Комбинационные цифровые схемы

Комбинационными цифровыми схемами называют схемы, выходные сигналы которых зависят только от сигналов, поданных на их входы в данный момент, и не зависят от состояния схемы в предыдущий момент времени. К комбинационным схемам обычно относят такие микросхемы, как дешифраторы, шифраторы, мультиплексоры и демультиплексоры.

Создание цифровых комбинационных устройств в настоящий момент в значительной степени формализовано, при этом для создания их принципиальных схем могут оказаться полезны некоторые законы алгебры логики. Несмотря на то, что все эти законы получены из таблиц истинности простейших логических элементов, применение этих законов оказывается более удобным при создании принципиальной схемы цифрового устройства.

Законы алгебры логики

Законы алгебры логики базируются на аксиомах и позволяют преобразовывать логические функции. Логические функции преобразуются с целью их упрощения, что приводит к упрощению схемы цифрового устройства.

Аксиомы алгебры логики описывают действие логических функций "И" и "ИЛИ" и записываются следующими выражениями:

функция "И"	функция "ИЛИ"
$0 \times 0 = 0$	$0 + 0 = 0$
$0 \times 1 = 0$	$0 + 1 = 1$
$1 \times 0 = 0$	$1 + 0 = 1$
$1 \times 1 = 1$	$1 + 1 = 1$

В данных выражениях знак '×' используется для обозначения операции логического умножения, а знак '+' — для обозначения операции логического суммирования. Эти же функции могут быть описаны при помощи таблиц истинности (табл. 6.1 и 6.2).

Таблица 6.1. *Таблица истинности схемы, выполняющей логическую функцию "2И"*

x_1	x_2	F
0	0	0
0	1	0
1	0	0
1	1	1

Таблица 6.2. *Таблица истинности схемы, выполняющей логическую функцию "2ИЛИ"*

x_1	x_2	F
0	0	0
0	1	1
1	0	1
1	1	1

Всего при разработке цифровых комбинационных устройств используется пять законов алгебры логики.

Закон одинарных элементов

$1 \times X = X$

$0 \times X = 0$

$1 + X = 1$

$0 + X = X$

Этот закон непосредственно следует из приведенных выше выражений аксиом алгебры логики (таблицы истинности логических элементов). Верхние два выражения могут быть полезны при построении коммутаторов, ведь подавая на один из входов элемента "2И" логический ноль или единицу можно либо

пропускать сигнал на выход, либо формировать на выходе нулевой потенциал.

Второй вариант использования этих выражений заключается в возможности избирательного обнуления определенных разрядов многоразрядного числа. При поразрядном применении операции "И" можно либо оставлять прежнее значение разряда, либо обнулять его, подавая на соответствующие разряды единичный или нулевой потенциал. Например, в восьмиразрядном двоичном числе требуется обнулить 6, 3 и 1 разряды. Тогда, при выполнении операции поразрядного логического умножения, получим:

$$\begin{array}{r} 11010111 \\ \times\ 10110101 \\ \hline 10010101 \end{array}$$

В приведенном примере отчетливо видно, что для обнуления необходимых разрядов в маске (нижнее число) на месте соответствующих разрядов записаны нули (не забываем, что счет начинается с нулевого разряда), в остальных разрядах записаны единицы. В исходном числе (верхнее число) на месте 6 и 1 разрядов находятся единицы. После выполнения операции "И" на этих местах появляются нули. На месте третьего разряда в исходном числе находится ноль. В результирующем числе на этом месте тоже присутствует ноль. Остальные разряды исходного числа, как и требовалось по условию задачи, не изменены.

Записывать логические единицы в нужные нам разряды многоразрядного двоичного числа можно точно таким же образом. В этом случае необходимо воспользоваться нижними двумя выражениями закона одинарных элементов. При поразрядном применении операции "ИЛИ" можно либо оставлять прежнее значение разряда, либо заносить в него единичное значение, подавая на соответствующие разряды нулевой или единичный потенциал. Пусть требуется записать единицы в 7 и 6 биты восьмиразрядного числа.

Тогда, при выполнении операции поразрядного логического суммирования исходного числа с маской устанавливаемых бит, получим:

$$\begin{array}{r} 01010111 \\ +\ 11000000 \\ \hline 11010111 \end{array}$$

Здесь в маску (нижнее число) мы записали единицы в седьмой и шестой биты. Остальные биты содержат нули и, следовательно, не могут изменить первоначальное состояние исходного числа, что мы и видим в соответствующих разрядах результирующего числа под чертой.

Первое и последнее выражения рассматриваемого закона позволяют использовать логические элементы с большим количеством входов в качестве элементов с меньшим количеством входов, если в схеме уже есть такие свободные элементы и не хочется вводить в состав принципиальной схемы дополнительные микросхемы. Для этого неиспользуемые входы логического элемента "И" должны быть подключены к источнику питания, как это показано на рис. 6.1.

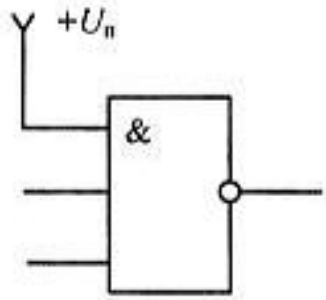


Рис. 6.1. Схема "2И-НЕ", реализованная на элементе "3И-НЕ"

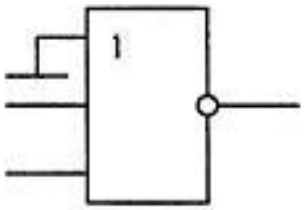


Рис. 6.2. Схема "2ИЛИ-НЕ", реализованная на элементе "3ИЛИ-НЕ"

В схеме логического элемента "ИЛИ", согласно закону одинарных элементов, для уменьшения количества входов логического элемента, неиспользуемые входы должны быть подключены к общему проводу схемы, как это показано на рис. 6.2.

Законы отрицания

Существуют следующие законы отрицания:

- закон дополнительных элементов:

$$x + \overline{x} = 1$$
$$x \times \overline{x} = 0$$

- двойное отрицание:

$$\overline{1} = 0 \quad \overline{0} = 1$$
$$\overline{\overline{1}} = 1 \quad \overline{\overline{x}} = x$$

Выражения, примененные в этих законах, широко используются для минимизации логических схем. Если удается выделить из общего выражения логической функции цифрового комбинационного устройства такие подвыражения, то можно сократить необходимое количество входов логических элементов в составе цифровой схемы, а иногда и вообще свести все выражение к логической константе.

- закон отрицательной логики:

$$\overline{\bar{a}+\bar{b}+\bar{c}} = a \cdot b \cdot c$$

$$\overline{\bar{a} \cdot \bar{b} \cdot \bar{c}} = a + b + c$$

Закон отрицательной логики справедлив для любого числа переменных. Этот закон позволяет реализовывать логическую функцию "И" при помощи логических элементов "ИЛИ" и наоборот: реализовывать логическую функцию "ИЛИ" при помощи логических элементов "И". Это особенно полезно в ТТЛ схемотехнике, т. к. там легко реализовать логические элементы "И", но при этом достаточно сложно реализуются логические элементы "ИЛИ". Благодаря закону отрицательной логики можно достаточно просто реализовывать элементы "ИЛИ" на логических элементах "И". На рис. 6.3 показана реализация логического элемента "2ИЛИ" на элементе "2И-НЕ" и двух инверторах.

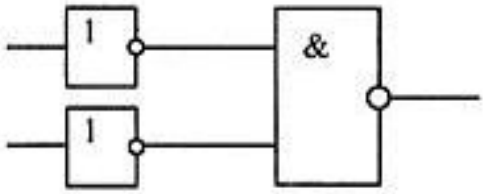


Рис. 6.3. Логический элемент "2ИЛИ", реализованный на элементе "2И-НЕ"

То же самое можно сказать и о схеме монтажного "ИЛИ". В случае необходимости его можно превратить в монтажное "И", применив инверторы на входе и выходе этой схемы.

Комбинационные законы

При применении комбинационных законов можно значительно упростить логическое выражение, описывающее цифровую схему и, тем самым, упростить ее принципиальную схему. Это позволяет сократить занимаемую цифровой схемой площадь на кристалле и потребляемый ею ток. Комбинационные законы алгебры логики во многом соответствуют комбинационным законам обычной алгебры, но есть и отличия.

- Закон тавтологии (многократное повторение):

$$x \times x \times x \times x = x$$

$$x + x + x + x = x$$

Применение этого закона зависит от цели. Его можно использовать для минимизации схемы, если такое выражение получается в результате преобразований исходного логического выражения.

Этот же закон позволяет использовать логические элементы с большим количеством входов в качестве элементов с меньшим количеством входов. Например, можно реализовать двухвходовую схему "2И" на элементе "3И", как это показано на рис. 6.4, или использовать схему "2И-НЕ" в качестве обычного инвертора, как это показано на рис. 6.5.

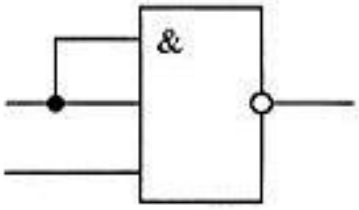


Рис. 6.4. Схема "2И-НЕ", реализованная на элементе "3И-НЕ"

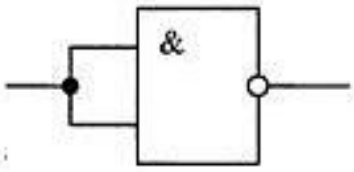


Рис. 6.5. Схема "НЕ", реализованная на элементе "2И-НЕ"

Однако следует предупредить, что объединение нескольких входов увеличивает входные токи логического элемента и его входную емкость, что увеличивает ток потребления предыдущих элементов и отрицательно сказывается на быстродействии цифровой схемы в целом. Для уменьшения числа входов в логическом элементе лучше воспользоваться законом одинарных элементов, как это было показано выше.

- Закон переместительности.

$$a+b+c+d=a+c+b+d$$

В случае применения этого закона можно сократить площадь печатной платы за счет того, что в ряде случаев одни выводы логического элемента можно заменить на другие. В результате при разработке конструкции цифрового устройства можно избежать переходов проводника на другой слой печатной платы или сократить общую длину проводников.

- Закон сочетательности.

$$a+b+c+d=a+(b+c)+d=a+b+(c+d)$$

Этот закон позволяет составлять многовходовые логические элементы из логических элементов с меньшим количеством входов. Причем комбинации элементов могут быть самыми разнообразными.

- Закон распределительности.

$$x_1+(x_2+x_3)=x_1\times x_2+x_1\times x_3$$

$$x_1+x_2\times x_3=(x_1+x_2)\times(x_1+x_3)$$

Докажем это выражение путем раскрытия скобок в правой части равенства:

$$(x_1+x_2)\times(x_1+x_3)=x_1\times x_1+x_1\times x_3+x_1\times x_2+x_2\times x_3=$$

$$=x_1(1+x_3+x_2)+x_2\times x_3=x_1+x_2\times x_3$$

- Правило поглощения.

$$x_1 + x_1 \times x_2 \times x_3 = x_1 \times (1 + x_2 \times x_3) = x_1$$

Одна переменная (в данном случае логическая переменная x_1) поглощает другие.

- Правило склеивания.

$$x_1 \cdot \overline{x}_2 \cdot x_3 + x_1 \cdot x_2 \cdot x_3 = x_1 \cdot x_3 \cdot \underbrace{(\overline{x}_2 + x_2)}_{1} = x_1 \cdot x_3$$

✧ *Обратите внимание*, что это правило выполняется только по одной переменной.

Также как в обычной математике, в алгебре логики имеется старшинство операций. При выполнении логических выражений первым выполняется:

1. Действие в скобках.
2. Операция с одним операндом (одноместная операция) — "НЕ".
3. Конъюнкция — "И".
4. Дизъюнкция — "ИЛИ".
5. Сумма по модулю два.

Операции одного ранга выполняются слева направо в порядке написания логического выражения. Алгебра логики линейна и для нее справедлив принцип суперпозиции.

Построение цифровой схемы по произвольной таблице истинности

Любая логическая схема без памяти полностью описывается таблицей истинности. При этом не обязательно, чтобы все комбинации входных сигналов были полезными. Возможна ситуация, когда только часть комбинаций входных сигналов является полезной. В этом случае выходные сигналы для оставшихся комбинаций входных сигналов могут быть доопределены произвольно. Обычно при этом стараются выбирать выходные сигналы таким образом, чтобы принципиальная схема цифрового устройства получилась простейшей.

Для реализации логических схем с произвольной таблицей истинности используется сочетание простейших логических элементов "И", "ИЛИ", "НЕ". Существует два способа синтеза цифровых схем, реализующих произвольную таблицу истинности. Это СКНФ (Совершенная Конъюнктивная Нормальная Форма — логическое произведение сумм входных сигналов) и

СДНФ (Совершенная Дизъюнктивная Нормальная Форма — сумма логических произведений входных сигналов).

При построении схемы, реализующей произвольную таблицу истинности, каждый выход анализируется (и строится схема) отдельно.

В настоящее время наиболее распространены микросхемы, совместимые с ТТЛ-технологией, а в этой технологии проще всего получить логические элементы "И". Поэтому первым рассмотрим способ реализации произвольной таблицы истинности, основанный на СДНФ.

Для реализации таблицы истинности при помощи логических элементов "И" достаточно рассмотреть только те строки таблицы истинности, которые содержат логические "1" в выходном сигнале. Строки, содержащие в выходном сигнале логический ноль, в построении схемы не участвуют. Каждая строка, содержащая в выходном сигнале логическую единицу, реализуется схемой логического "И" с количеством входов, совпадающим с количеством входных сигналов в таблице истинности.

Входные сигналы, описанные в таблице истинности логической единицей, подаются на вход этой схемы непосредственно, а входные сигналы, описанные в таблице истинности логическим нулем, подаются на вход этой же схемы "И" через инверторы. Объединение сигналов с выходов схем "И", реализующих отдельные строки таблицы истинности, производится при помощи схемы логического "ИЛИ". Количество входов в схеме "ИЛИ" определяется количеством строк в таблице истинности, в которых в выходном сигнале присутствует логическая единица.

Рассмотрим конкретный пример. Пусть необходимо реализовать таблицу истинности, приведенную в табл. 6.3.

Для построения схемы, реализующей сигнал F0, достаточно рассмотреть строки, выделенные жирным шрифтом. В рассматриваемой таблице истинности имеются всего три строки, содержащие единицу в выходном сигнале F0, поэтому в формуле СДНФ будет содержаться три произведения входных сигналов:

$$F0 = \overline{x0} \cdot \overline{x1} \cdot \overline{x2} \cdot x3 + \overline{x0} \cdot x1 \cdot x2 \cdot \overline{x3} + x0 \cdot x1 \cdot \overline{x2} \cdot \overline{x3}$$

Полученная формула в схеме на рис. 6.6 реализуется микросхемой D2. Как и в формуле, каждая строка таблицы истинности реализуется своей схемой "И", затем выходы этих схем объединяются схемой "ИЛИ". Количество входов элемента "И" однозначно определяется количеством входных сигналов в таблице истинности. Количество этих элементов, а значит и количество входов в логическом элементе "ИЛИ", определяется количеством строк с единичным сигналом на реализуемом выходе схемы.

Таблица 6.3. *Пример таблицы истинности*

Входы				Выходы	
x0	x1	x2	x3	F0	F1
0	0	0	0	0	0
0	**0**	**0**	**1**	**1**	0
0	0	1	0	0	0
0	0	1	1	0	0
0	1	0	0	0	0
0	*1*	*0*	*1*	*0*	*1*
0	**1**	**1**	**0**	**1**	0
0	1	1	1	0	0
1	0	0	0	0	0
1	*0*	*0*	*1*	*0*	*1*
1	0	1	0	0	0
1	0	1	1	0	0
1	**1**	**0**	**0**	**1**	0
1	1	0	1	0	0
1	1	1	0	0	0
1	*1*	*1*	*1*	*0*	*1*

Для построения схемы, реализующей сигнал F1, достаточно рассмотреть строки, выделенные курсивом. Эти строки в схеме на рис. 6.6 реализуются микросхемой D3. Принцип построения этой схемы не отличается от примера, рассмотренного выше. В таблице истинности присутствуют всего три строки, содержащие единицу в выходном сигнале F1, поэтому в формуле СДНФ выхода F1 будет содержаться три произведения входных сигналов:

$$F1 = \overline{x0} \cdot x1 \cdot \overline{x2} \cdot x3 + x0 \cdot \overline{x1} \cdot \overline{x2} \cdot x3 + x0 \cdot x1 \cdot x2 \cdot x3$$

Обычно при построении цифровых схем после реализации таблицы истинности производится минимизация схемы, но в данном случае для упрощения понимания материала минимизация производиться не будет. Это оправдано еще и с той точки зрения, что схемы, построенные по ДНФ, обычно обладают максимальным быстродействием. При реализации схемы на ТТЛ-микро-

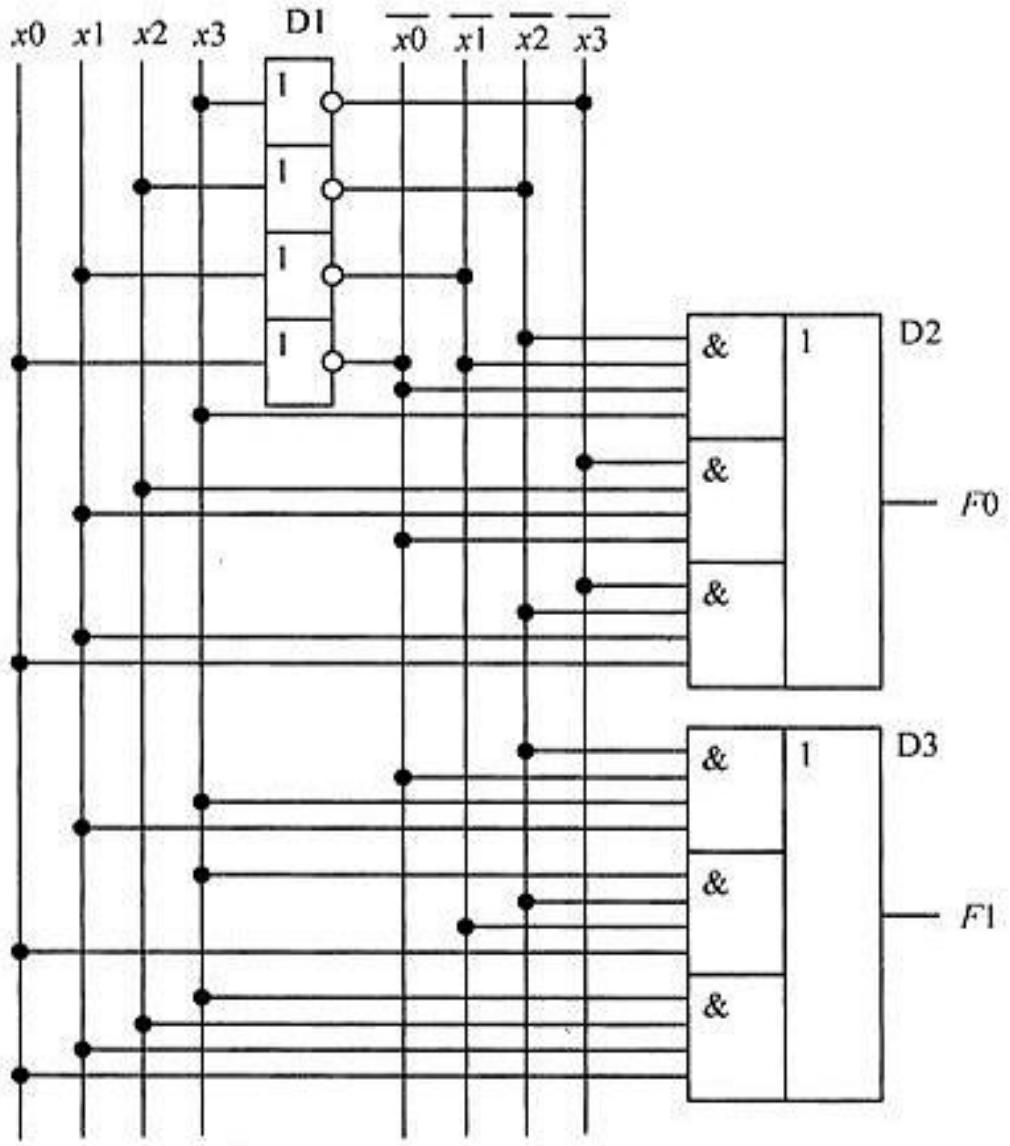


Рис. 6.6. Принципиальная схема, реализующая таблицу истинности, приведенную в табл. 6.3

схемах быстродействие такого узла будет равно быстродействию одиночного инвертора.

Для иллюстрации этого утверждения рассмотрим схему логического элемента "2И-2ИЛИ-НЕ", выполненного по ТТЛ-технологии. Она приведена на рис. 6.7. Основные компоненты, определяющие быстродействие этой схемы, — это транзисторы, образующие схему "ИЛИ", т. к. только они обладают усилением по напряжению, а, значит, их входная емкость увеличивается за счет эффекта Миллера.

Вспомним, что транзисторы соединены параллельно, а это означает, что время задержки сигнала по каждой цепи не суммируется. Быстродействие схемы в целом будет определяться наибольшей из рассматриваемых задержек сигнала, а быстродействие всей схемы в целом будет равно быстродействию одиночного (наихудшего с точки зрения быстродействия) инвертора.

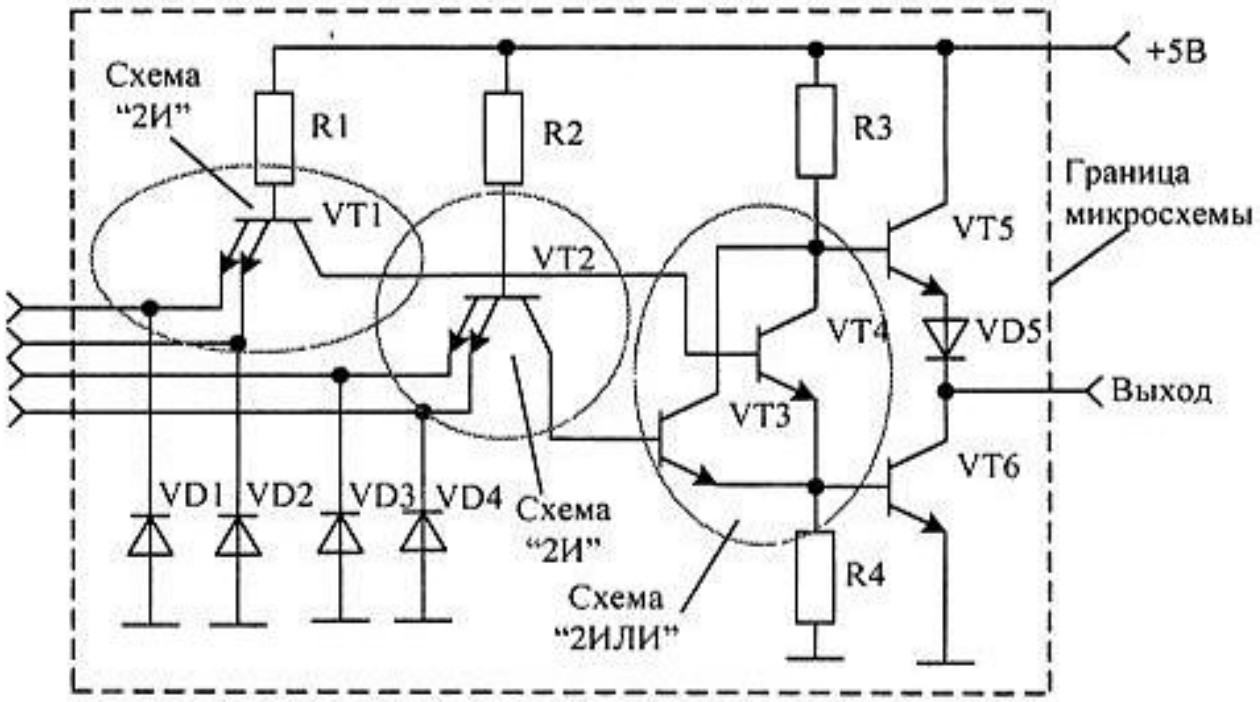


Рис. 6.7. Принципиальная схема ТТЛ-микросхемы "2И-2ИЛИ-НЕ"

Применение СКНФ для разработки цифровой схемы оправдано при большом количестве единиц в таблице истинности, описывающей выходной сигнал, как, например, в таблице истинности, приведенной в табл. 6.4.

Таблица 6.4. *Пример таблицы истинности*

№ комбинации	Входы				Выходы	
	x0	x1	x2	x3	F0	F1
0	0	0	0	0	1	1
1	*0*	*0*	*0*	*1*	*0*	1
2	0	0	1	0	1	1
3	0	0	1	1	1	1
4	*0*	*1*	*0*	*0*	*0*	1
5	**0**	**1**	**0**	**1**	1	**0**
6	**0**	**1**	**1**	**0**	1	**0**
7	0	1	1	1	1	1
8	1	0	0	0	1	1
9	1	0	0	1	1	1

Для реализации таблицы истинности при помощи логических элементов "ИЛИ" достаточно рассмотреть только те строки таблицы истинности, кото-

рые в выходном сигнале содержат логические "0". Строки, содержащие в выходном сигнале логическую единицу, в построении схемы не участвуют. Каждая строка, содержащая в выходном сигнале логический ноль, реализуется логическим элементом "ИЛИ" с количеством входов, совпадающим с количеством входных сигналов в таблице истинности. Входные сигналы, описанные в таблице истинности логическим нулем, подаются на вход этой схемы непосредственно, а входные сигналы, описанные в таблице истинности логической единицей, подаются на вход этой же схемы "ИЛИ" через инверторы.

Объединение сигналов с выходов логических элементов "ИЛИ", реализующих отдельные строки таблицы истинности, производится при помощи логического элемента "И". Количество входов в логическом элементе "И" определяется количеством строк в таблице истинности, в которых в выходном сигнале присутствует логический ноль.

Для построения схемы, реализующей сигнал F0, достаточно рассмотреть строки, выделенные курсивом. В рассматриваемой таблице истинности име-

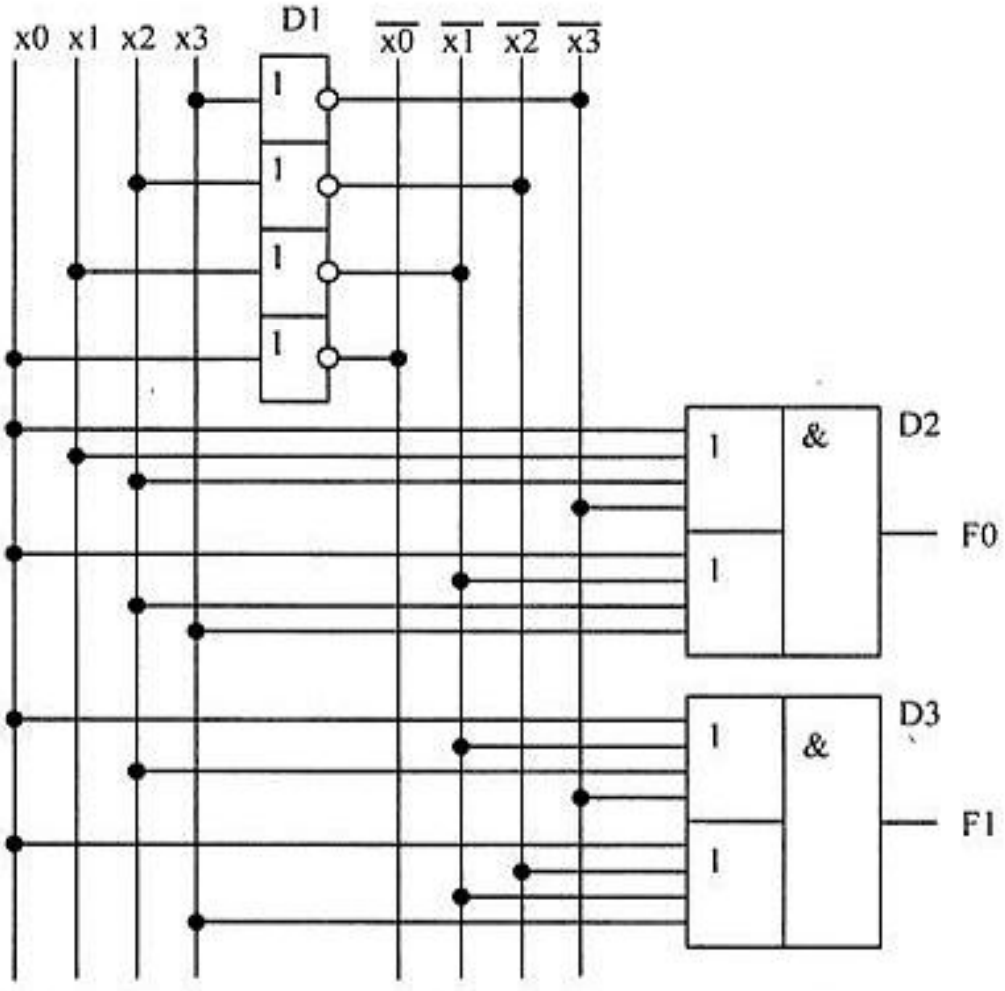


Рис. 6.8. Принципиальная схема цифрового устройства с таблицей истинности, приведенной в табл. 6.4

ются всего две строки, содержащие логический ноль в выходном сигнале F0, поэтому в формуле СКНФ будет содержаться две суммы входных сигналов:

$$F0 = (x0 + x1 + x2 + \overline{x3}) \cdot (x0 + \overline{x1} + x2 + x3)$$

Полученная формула в схеме, приведенной на рис. 6.8, реализуется логическим элементом D2.

Для построения схемы, формирующей выходной сигнал F1, достаточно рассмотреть строки, выделенные в табл. 6.2 жирным шрифтом. Эти строки в схеме на рис. 6.8 реализуются микросхемой D3. Принцип построения этой схемы не отличается от примера, рассмотренного выше. В таблице истинности присутствуют всего две строки, содержащие ноль в выходном сигнале F1, поэтому в формуле СКНФ, записанной для выхода F1, будет содержаться две суммы входных сигналов:

$$F1 = (x0 + \overline{x1} + x2 + \overline{x3}) \cdot (x0 + \overline{x1} + \overline{x2} + x3)$$

Как видно из приведенных примеров, построение цифровой схемы во многом формализовано, и не вызывает каких-либо затруднений. В следующих разделах данной главы будут рассмотрены дополнительные примеры реализации комбинационных цифровых устройств. В качестве примеров послужат наиболее распространенные виды цифровых комбинационных микросхем.

Декодеры

Декодеры (дешифраторы) позволяют преобразовывать одни виды бинарных кодов в другие. Например, преобразовывать позиционный двоичный код в линейный восьмеричный или шестнадцатеричный код. Преобразование бинарных кодов производится по правилам, описываемым в таблицах истинности, поэтому построение принципиальных схем дешифраторов не представляет трудностей. Для построения схемы дешифратора можно воспользоваться правилами построения схемы для произвольной таблицы истинности, которые были рассмотрены ранее.

Десятичный дешифратор

Рассмотрим пример построения декодера из двоичного кода в десятичный. Десятичный код обычно отображается одним битом на одну десятичную цифру. В десятичном коде применяется десять цифр, поэтому для отображения одного десятичного разряда требуется десять выходов дешифратора. Сигнал с этих выводов можно подать на десятичный индикатор. В простейшем случае для реализации десятичного дешифратора можно просто подписать индицируемую цифру над соответствующим светодиодом. Таблица истинности десятичного декодера приведена в табл. 6.5.

Таблица 6.5. *Таблица истинности десятичного декодера*

№ комбинации	Входы				Выходы									
	8	4	2	1	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	1	0	0	0	0	0	0	0	0
2	0	0	1	0	0	0	1	0	0	0	0	0	0	0
3	0	0	1	1	0	0	0	1	0	0	0	0	0	0
4	0	1	0	0	0	0	0	0	1	0	0	0	0	0
5	0	1	0	1	0	0	0	0	0	1	0	0	0	0
6	0	1	1	0	0	0	0	0	0	0	1	0	0	0
7	0	1	1	1	0	0	0	0	0	0	0	1	0	0
8	1	0	0	0	0	0	0	0	0	0	0	0	1	0
9	1	0	0	1	0	0	0	0	0	0	0	0	0	1

Десятичный код, записанный в таблице истинности 6.5, иногда называют линейным десятичным кодом. Точно таким же образом можно составить таблицу истинности и для восьмеричного декодера. В этом случае на его восьми выходах будет присутствовать линейный восьмеричный код.

Принципы построения цифровой схемы по произвольной таблице истинности были рассмотрены в предыдущих разделах. В соответствии с ними получим схему десятичного декодера, таблица истинности которого записана в табл. 6.5. Полученная в результате синтеза принципиальная схема десятичного дешифратора приведена на рис. 6.9.

Как видно на этой схеме, для реализации каждой строки таблицы истинности потребовалась схема "4И". Схема "ИЛИ" не понадобилась, т. к. в таблице истинности на каждом выходе присутствует только одна единица, а значит, объединять выходы логических элементов "И" не требуется.

Дешифраторы выпускаются в виде отдельных микросхем или используются в составе других микросхем. В настоящее время десятичные или восьмеричные дешифраторы используются в основном как составная часть других микросхем, таких как мультиплексоры, демультиплексоры, ПЗУ или ОЗУ.

На рис. 6.10 приведено условно-графическое обозначение двоично-десятичного дешифратора, полная принципиальная схема которого изображена на рис. 6.9.

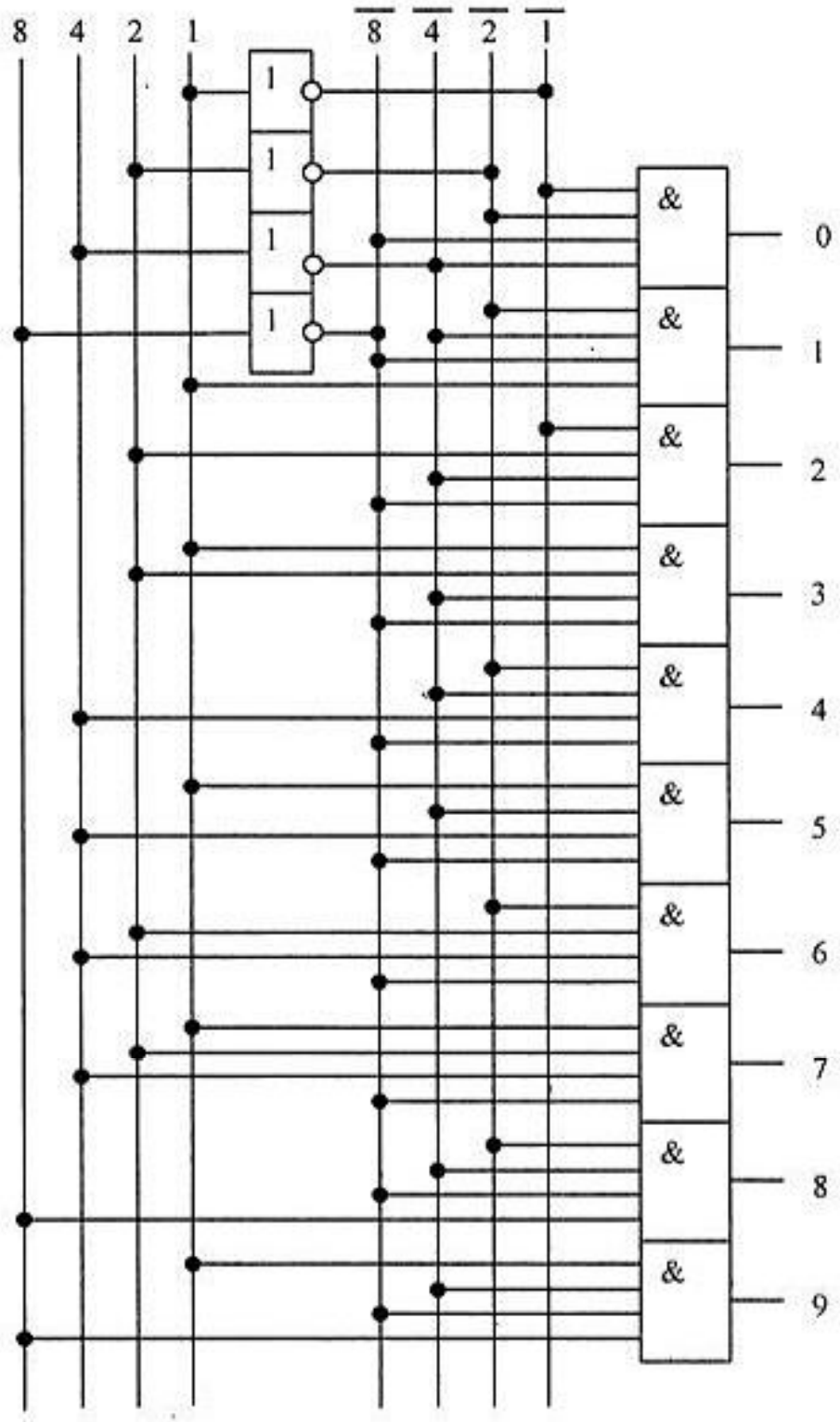


Рис. 6.9. Принципиальная схема двоично-десятичного декодера

Точно таким же образом можно получить принципиальную схему и для любого другого декодера (дешифратора). Наиболее распространены схемы восьмеричных и шестнадцатеричных дешифраторов. Для индикации такие дешифраторы в настоящее время практически не применяются. В основном они используются как составная часть более сложных цифровых модулей.

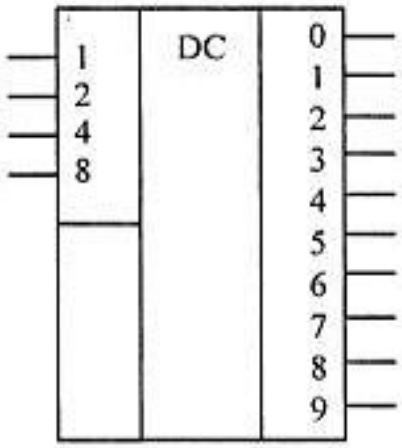


Рис. 6.10. Условно-графическое обозначение двоично-десятичного дешифратора

Семисегментный дешифратор

Для отображения десятичных и шестнадцатеричных цифр часто используется семисегментный индикатор. Внешний вид семисегментного индикатора и обозначение его сегментов приведено на рис. 6.11.

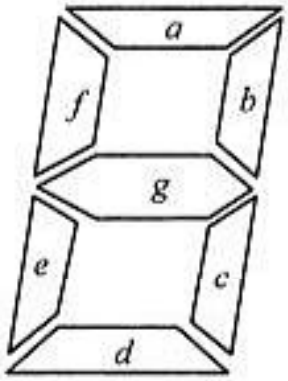


Рис. 6.11. Внешний вид семисегментного индикатора и название его сегментов

Для отображения на таком индикаторе цифры 0 достаточно зажечь сегменты "a", "b", "c", "d", "e", "f". Для изображения цифры 1 зажигают сегменты "b" и "c". Точно таким же образом можно получить изображения всех остальных десятичных или шестнадцатеричных цифр. Все комбинации двоичных кодов, позволяющих сформировать изображения цифр (и некоторых букв), получили название семисегментного кода.

Составим таблицу истинности дешифратора, который позволит преобразовывать двоичный (а точнее двоично-десятичный) код в семисегментный. Пусть сегменты индикатора зажигаются нулевым потенциалом. Тогда таблица истинности семисегментного дешифратора примет вид, приведенный в табл. 6.6. Конкретное значение сигналов на выходе дешифратора зависит от схемы подключения сегментов индикатора к выходу микросхемы. Эти схемы

будут рассмотрены позднее, в главе, посвященной отображению различных видов информации.

Таблица 6.6. *Таблица истинности семисегментного декодера*

№ комбинации	Входы				Выходы						
	8	4	2	1	a	b	c	d	e	f	g
0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	1	1	0	0	1	1	1	1
2	0	0	1	0	0	0	1	0	0	1	0
3	0	0	1	1	0	0	0	0	1	1	0
4	0	1	0	0	1	0	0	1	1	0	0
5	0	1	0	1	0	1	0	0	1	0	0
6	0	1	1	0	0	1	0	0	0	0	0
7	0	1	1	1	0	0	0	1	1	1	1
8	1	0	0	0	0	0	0	0	0	0	0
9	1	0	0	1	0	0	0	0	1	0	0

В соответствии с принципами построения схемы по произвольной таблице истинности реализуем принципиальную схему семисегментного декодера, работающую в соответствии с таблицей истинности, записанной в табл. 6.6. На этот раз не будем подробно расписывать процесс разработки схемы. Для проверки того, как вы разобрались в алгоритме разработки цифровых схем, попробуйте самостоятельно ее получить. Разработанная в результате такого синтеза принципиальная схема семисегментного дешифратора приведена на рис. 6.12.

Для облегчения понимания принципов работы приведенной на рис. 6.12 схемы на выходе логических элементов "И" показаны номера строк таблицы истинности, реализуемые ими. Например, на выходе сегмента ‘a’ логическая единица появится только при подаче на вход комбинации двоичных сигналов 0001 (1) и 0100 (4). Это осуществляется объединением соответствующих цепей элементом "2ИЛИ". На выходе сегмента ‘b’ логическая единица появится только при подаче на вход комбинации двоичных сигналов 0101 (5) и 0110 (6) и т. д.

В настоящее время семисегментные дешифраторы выпускаются в виде отдельных микросхем или используются в виде готовых блоков в составе дру-

гих микросхем. Условно-графическое обозначение микросхемы семисегментного дешифратора приведено на рис. 6.13.

В качестве примера семисегментных дешифраторов можно назвать такие микросхемы отечественного производства, как К176ИД3. Они предназначены для подключения газоразрядных индикаторов. В современных цифровых схемах семисегментные дешифраторы обычно входят в состав больших интегральных схем.

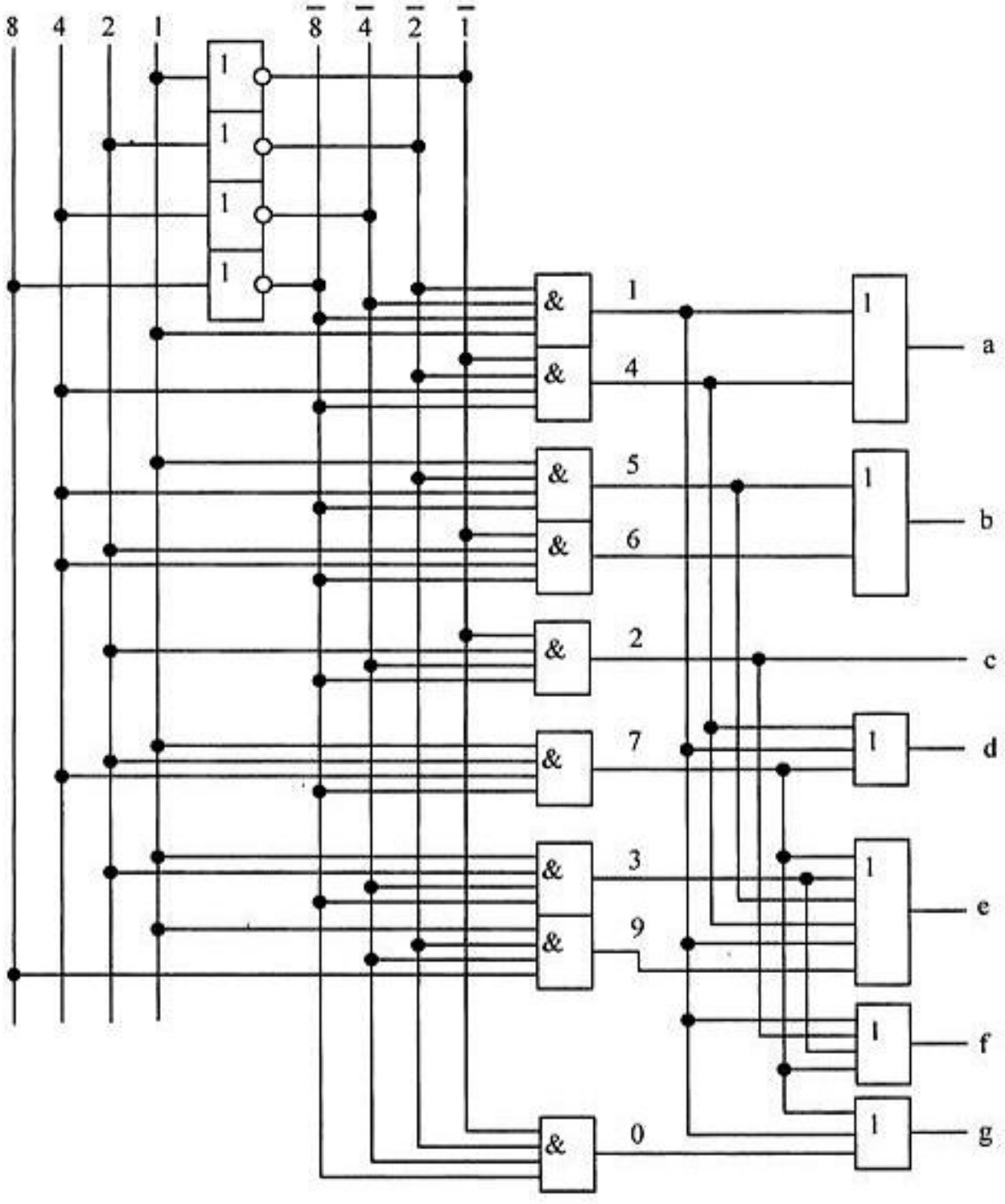


Рис. 6.12. Принципиальная схема семисегментного дешифратора

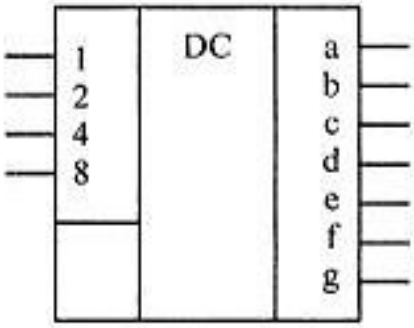


Рис. 6.13. Условно-графическое обозначение семисегментного дешифратора

Шифраторы

Достаточно часто перед разработчиками цифровой аппаратуры встает задача, обратная той, что решается с помощью дешифраторов. Например, требуется преобразовать восьмеричный или десятичный линейный код в двоичный. Линейный восьмеричный код может поступать, например, с выхода механического переключателя. Составим таблицу истинности подобного устройства.

Таблица 6.7. *Таблица истинности восьмеричного кодера*

№ комбинации	Входы							Выходы		
	1	2	3	4	5	6	7	4	2	1
0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	1
2	0	1	0	0	0	0	0	0	1	0
3	0	0	1	0	0	0	0	0	1	1
4	0	0	0	1	0	0	0	1	0	0
5	0	0	0	0	1	0	0	1	0	1
6	0	0	0	0	0	1	0	1	1	0
7	0	0	0	0	0	0	1	1	1	1

Еще одним источником линейного восьмеричного кода могут стать аналоговые компараторы с различными порогами срабатывания. Такая линейка компараторов используется в составе параллельного аналого-цифрового преобразователя для преобразования аналогового сигнала в цифровой код. Двоичный код более компактен и удобен для последующей обработки. Поэтому требуется преобразователь линейного восьмеричного или шестнадцатеричного кода в двоичный. Таблица истинности такого устройства несколько отличается

от таблицы, приведенной в табл. 6.7. Таблица истинности кодера параллельного аналого-цифрового преобразователя приведена в табл. 6.8.

Таблица 6.8. *Таблица истинности кодера параллельного аналого-цифрового преобразователя*

№ комбинации	Входы							Выходы		
	1	2	3	4	5	6	7	4	2	1
0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	1
2	1	1	0	0	0	0	0	0	1	0
3	1	1	1	0	0	0	0	0	1	1
4	1	1	1	1	0	0	0	1	0	0
5	1	1	1	1	1	0	0	1	0	1
6	1	1	1	1	1	1	0	1	1	0
7	1	1	1	1	1	1	1	1	1	1

Таблицы истинности двух рассмотренных устройств можно объединить. В этом случае ячейки таблицы истинности, для которых неважно, будет ли в них записан логический ноль или единица, помечены символом "X". Объединенная таблица истинности приведена под номером 6.9.

Таблица 6.9. *Таблица истинности восьмеричного универсального кодера*

№ комбинации	Входы							Выходы		
	1	2	3	4	5	6	7	4	2	1
1	1	0	0	0	0	0	0	0	0	1
2	X	1	0	0	0	0	0	0	1	0
3	X	X	1	0	0	0	0	0	1	1
4	X	X	X	1	0	0	0	1	0	0
5	X	X	X	X	1	0	0	1	0	1
6	X	X	X	X	X	1	0	1	1	0
7	X	X	X	X	X	X	1	1	1	1

Теперь по разработанной таблице истинности (табл. 6.9) можно составить принципиальную схему устройства. То, что практически во всех строках есть

неопределенные значения, позволяет значительно упростить схему восьмеричного кодера.

Наиболее простое решение получается для старшего разряда. Здесь можно обойтись схемой логического элемента "4ИЛИ". Для получения единичного сигнала в выходном сигнале '2' в 6-й и 7-й строках таблицы истинности достаточно объединить входные сигналы '7' и '6'. Точно так же добавляются строки 2 и 3, однако здесь уже потребуется дешифрация входных сигналов 2, 3, 4 и 5. Результирующая принципиальная схема восьмеричного кодера приведена на рис. 6.14.

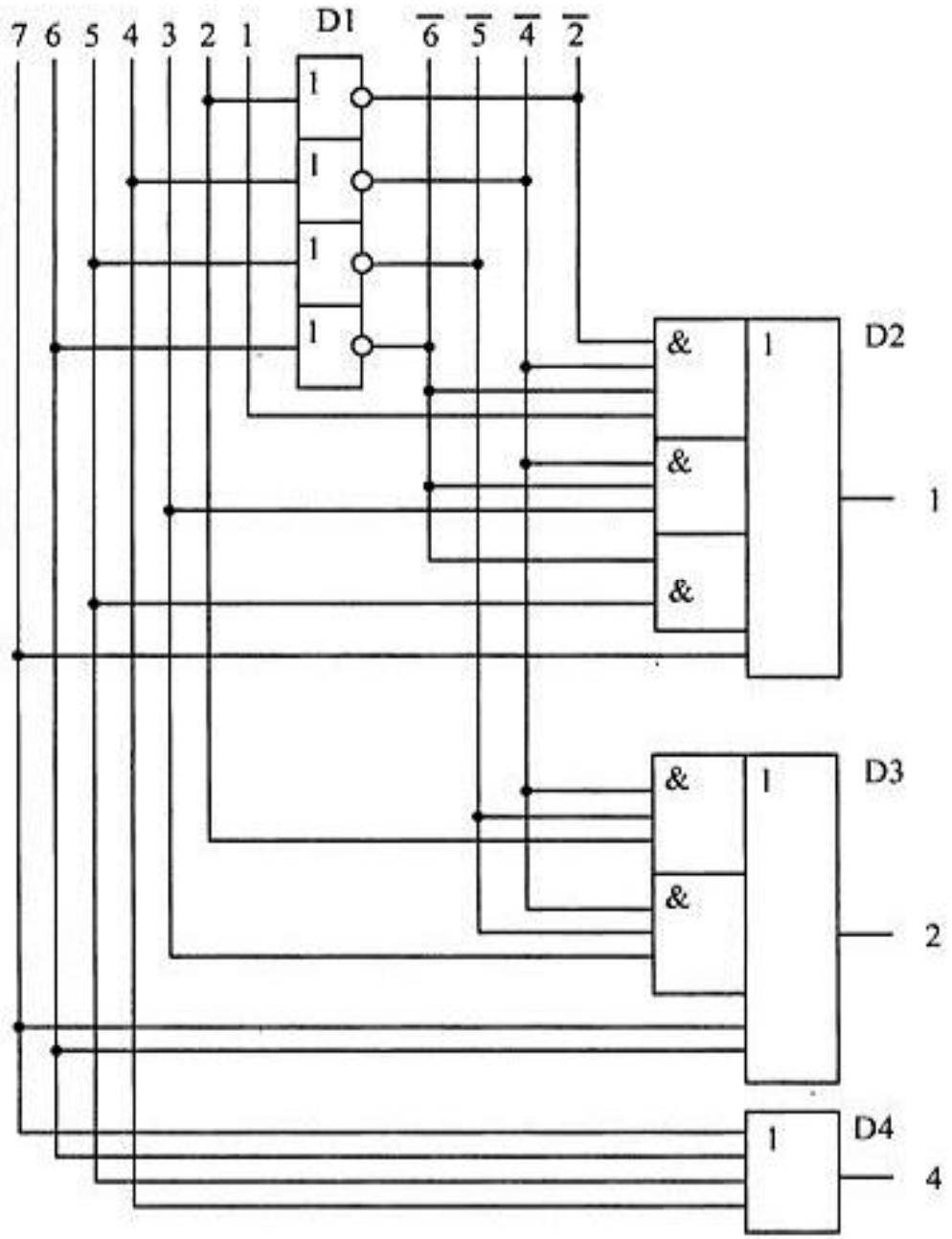


Рис. 6.14. Принципиальная схема восьмеричного кодера

В настоящее время шифраторы (кодеры) выпускаются в виде отдельных микросхем или используются в виде готовых блоков в составе других микросхем, таких как параллельные АЦП. Условно-графическое обозначение восьмеричного шифратора приведено на рис. 6.15.

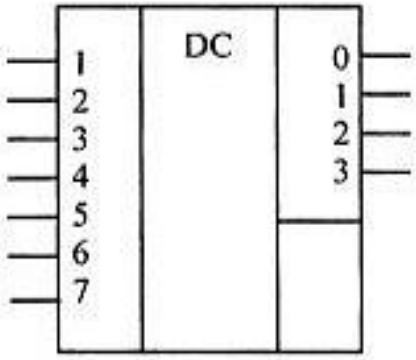


Рис. 6.15. Условно-графическое обозначение восьмеричного шифратора

В качестве примера интегрального исполнения шифраторов можно назвать такие микросхемы отечественного производства, как К555ИВ1 (восьмеричный шифратор) и К555ИВ3 (десятичный шифратор).

Мультиплексоры

Мультиплексорами называются устройства, которые позволяют подключать один из нескольких входов к одному выходу. Иными словами, мультиплексор — это коммутатор, у которого есть несколько входов и один выход. В простейшем случае такую коммутацию можно осуществить при помощи ключей с электронным управлением (рис. 6.16).

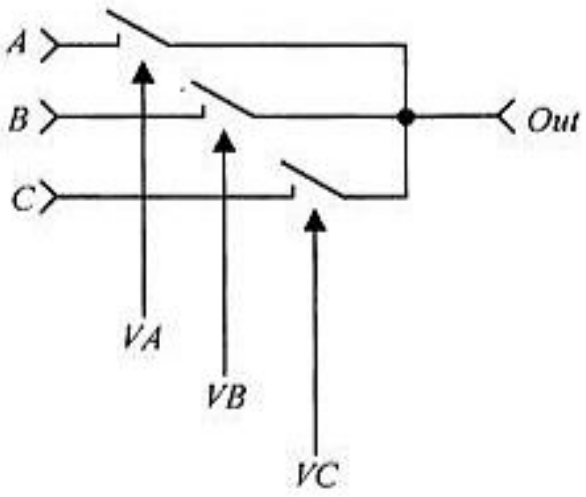


Рис. 6.16. Коммутатор (мультиплексор), собранный на ключах

Такой коммутатор одинаково хорошо будет работать как с аналоговыми, так и с цифровыми сигналами. Однако скорость работы механических ключей оставляет желать лучшего, да и управлять ключами часто приходится автоматически при помощи какой-либо схемы.

В цифровых схемах требуется управлять ключами при помощи логических уровней, поэтому желательно подобрать устройство, которое могло бы выполнять функции электронного ключа с электронным управлением цифровым сигналом.

Особенности построения мультиплексоров на ТТЛ-элементах

Попробуем использовать в качестве электронного ключа уже знакомые нам логические элементы, для этого рассмотрим таблицу истинности логического элемента "2И". При работе с таблицей один из входов логического элемента "2И" будем рассматривать как информационный вход электронного ключа, а другой вход — как управляющий. Так как оба входа логического элемента "2И" эквивалентны, то неважно, какой из них будет управляющим входом.

Пусть вход X будет управляющим, а Y — информационным. Для простоты рассуждений, разделим таблицу истинности логического элемента "2И" на две части в зависимости от уровня логического сигнала на управляющем входе X, как это показано на рис. 6.17.

X	Y	F_0
0	0	0
0	1	0
1	0	0
1	1	1

Рис. 6.17. Таблица истинности логического элемента "2И"

По таблице истинности отчетливо видно, что пока на управляющий вход X подан нулевой логический уровень, сигнал, поданный на вход Y, на выход F_0 не проходит. При подаче на управляющий вход X логической единицы сигнал, поступающий на вход Y, появляется на выходе F_0.

Это означает, что логический элемент "2И" можно использовать в качестве электронного ключа. При этом неважно, какой из входов элемента "2И" будет использоваться в качестве управляющего входа, а какой — в качестве ин-

формационного. Остается только объединить выходы элементов "2И" на один выход. Это делается при помощи логического элемента "ИЛИ" точно так же, как и при построении схемы по произвольной таблице истинности. Получившийся вариант схемы коммутатора с управлением логическими уровнями приведен на рис. 6.18.

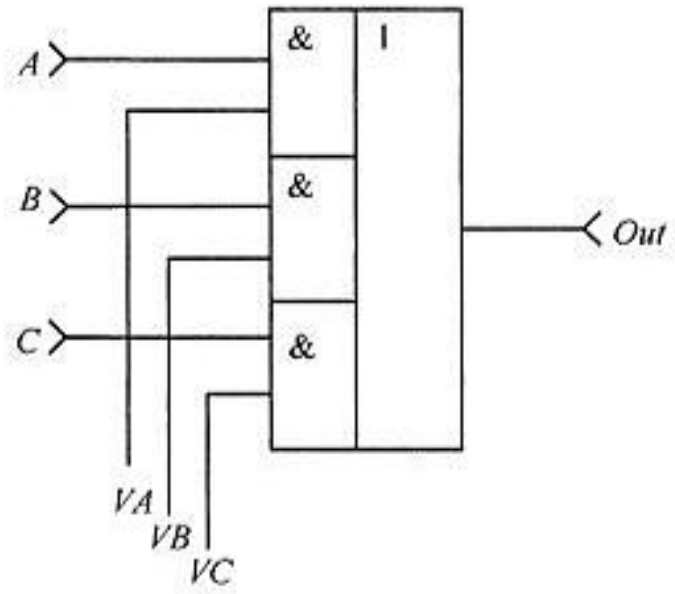


Рис. 6.18. Принципиальная схема мультиплексора, выполненного на логических элементах

В схемах, приведенных на рис. 6.16 и 6.18, можно одновременно подключать несколько входов на один выход. Однако обычно это приводит к непредсказуемым последствиям. Кроме того, для управления таким коммутатором требуется много входов, поэтому в состав мультиплексора обычно включают двоичный дешифратор, как показано на рис. 6.19. Такая схема позволяет управлять переключением информационных входов мультиплексора при помощи двоичных кодов, подаваемых на его управляющие входы. Количество информационных входов в схемах мультиплексоров обычно выбирают кратным степени числа два.

Условно-графическое обозначение четырехвходового мультиплексора с управлением двоичным кодом приведено на рис. 6.20. Входы A0 и A1 являются управляющими входами мультиплексора, определяющими адрес информационного входного сигнала, который будет соединен с выходным выводом Y мультиплексора MUX. Информационные входные сигналы обозначены как X0, X1, X2 и X3.

В условно-графическом обозначении названия информационных входов A, B, C и D заменены названиями X0, X1, X2 и X3, а название выхода Out заменено на название Y. Такие наименования входов и выходов более распространены в отечественной литературе. Адресные входы обозначены как A0 и A1.

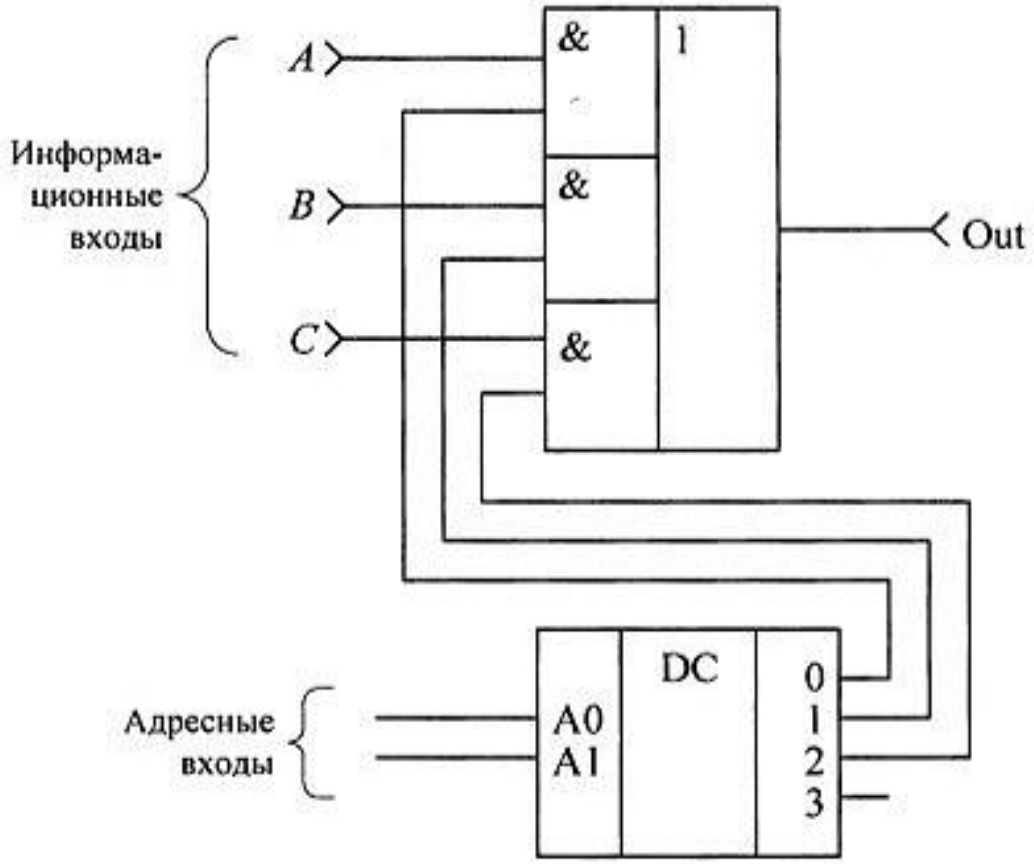


Рис. 6.19. Принципиальная схема мультиплексора, управляемого двоичным кодом

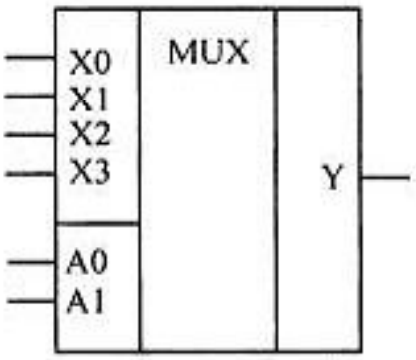


Рис. 6.20. Условно-графическое обозначение мультиплексора

Особенности построения мультиплексоров на КМОП-элементах

При работе с КМОП логическими элементами электронный ключ очень легко получить на одном или двух МОП-транзисторах, поэтому в КМОП-схемах логический элемент "И" в качестве электронного ключа не используется. Схема электронного ключа, выполненного на комплементарных МОП-транзисторах, приведена на рис. 6.21.

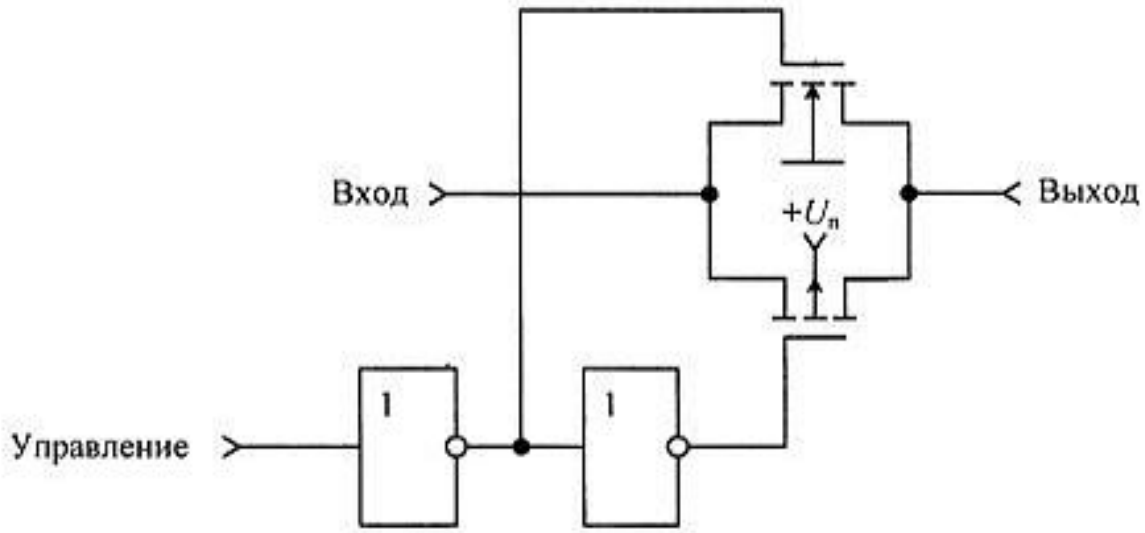


Рис. 6.21. Схема электронного ключа, выполненного на МОП-транзисторах

Такой ключ может коммутировать как цифровые, так и аналоговые сигналы. Сопротивление открытых транзисторов составляет десятки Ом, а сопротивление закрытых транзисторов превышает десятки мегаом. В этом есть как преимущества, так и недостатки. То, что ключ, собранный на МОП-транзисторе, не является обычным логическим элементом, позволяет объединять выходы электронных ключей в точном соответствии со схемой, приведенной на рис. 6.16. Это существенно упрощает схему устройства.

МОП-мультиплексор может быть использован для коммутации аналоговых сигналов. Только при этом не следует забывать, что такая схема не выдерживает отрицательных напряжений. Это означает, что для коммутации аналоговых сигналов необходимо использовать дополнительную схему смещения, так, чтобы значения аналогового сигнала находились в диапазоне от потенциала общего провода схемы до напряжения питания мультиплексора. При работе с мультиплексором, собранным на таких ключах, необходимо дополнительно включать на его входе и выходе логические элементы. Только в этом случае цифровая схема в целом будет функционировать правильно (не вызывать затухания сигналов). Следует отметить, что в большинстве схем цифровых устройств это условие выполняется автоматически.

В мультиплексоре по логике его работы требуется подключать к выходу только один из входных сигналов, поэтому точно так же, как и в ТТЛ-микросхемах, для управления электронными ключами двоичным кодом в состав мультиплексора вводится дешифратор. Схема такого мультиплексора приведена на рис. 6.22.

Условно-графическое обозначение мультиплексоров не зависит от технологии изготовления микросхем, т. е. КМОП-мультиплексор обозначается точно так же, как это показано на рис. 6.20.

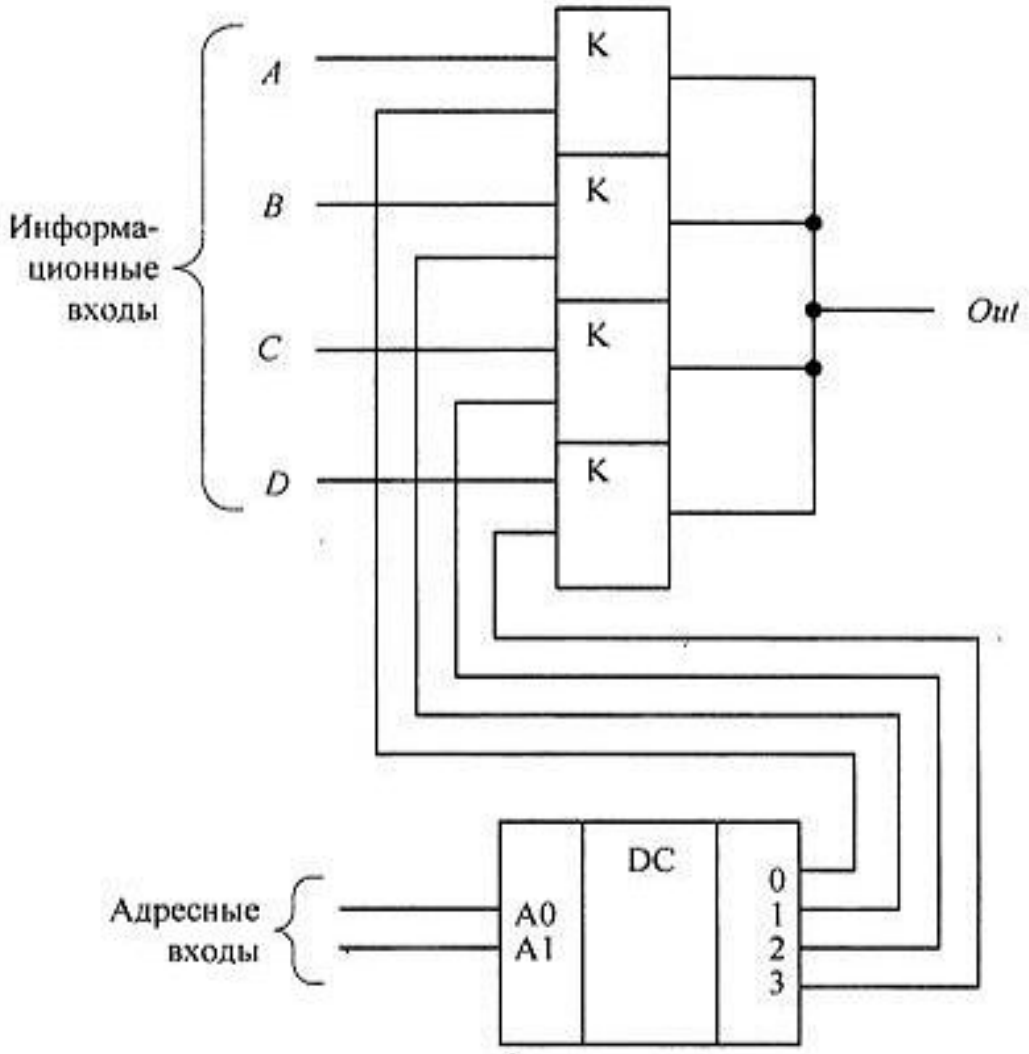


Рис. 6.22. Схема КМОП-мультиплексора

В отечественных микросхемах мультиплексоры обозначаются буквами КП, следующими непосредственно за номером серии микросхем. Например, микросхема К1533КП2 является сдвоенным четырехканальным мультиплексором, выполненным по ТТЛ-технологии, а микросхема К1561КП1 является сдвоенным четырехканальным мультиплексором, выполненным по КМОП-технологии.

Демультиплексоры

Демультиплексорами называются устройства, которые позволяют подключать один вход к одному из нескольких выходов. Демультиплексор можно построить на основе таких же логических элементов "2И", как и при построении мультиплексора. Существенным отличием от мультиплексора является возможность объединения нескольких входов в один без дополнительных логических схем. Однако для увеличения нагрузочной способности микросхемы, на входе демультиплексора в качестве усилителя входного сигнала дополнительно включается инвертор.

Схема демультиплексора, иллюстрирующая принцип его работы, приведена на рис. 6.23. В этой схеме для выбора конкретного выхода демультиплексора, как и в мультиплексоре, используется двоичный дешифратор.

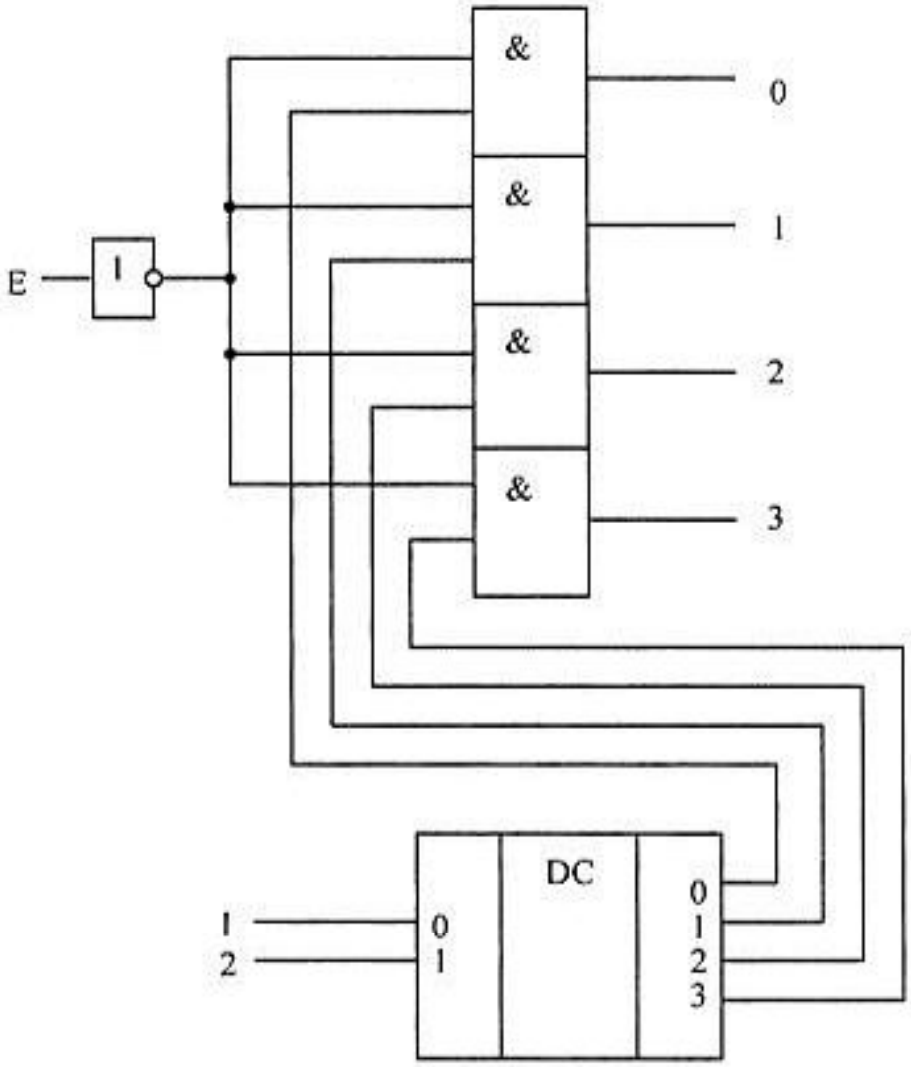


Рис. 6.23. Принципиальная схема демультиплексора

Тем не менее, если рассмотреть принципиальную схему самого дешифратора, то можно значительно упростить демультиплексор. Для реализации демультиплексора достаточно к каждому логическому элементу "И", входящему в состав принципиальной схемы дешифратора, добавить еще один вход — E. Такую схему часто называют дешифратором с входом разрешения работы. Условно-графическое изображение демультиплексора приведено на рис. 6.24.

В КМОП-микросхемах не существует отдельных микросхем демультиплексоров, т. к. МОП-мультиплексоры, описанные ранее, по информационным сигналам вход и выход не различают, т. е. направление распространения информационных сигналов, точно так же, как и в механических ключах, может быть произвольным. Если поменять входы и выход местами, то КМОП-

мультиплексоры будут работать в качестве демультиплексоров, поэтому их часто называют просто коммутаторами.

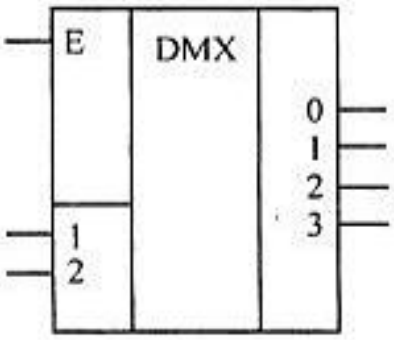
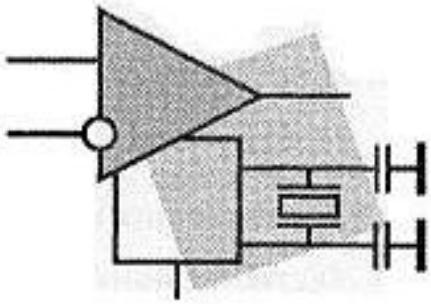


Рис. 6.24. Условно-графическое обозначение демультиплексора с четырьмя выходами

Итоги

В данной главе мы рассмотрели особенности проектирования цифровых комбинационных устройств. Мы убедились, что можно легко создать цифровое устройство любой сложности при помощи всего двух простых методов, используя в качестве исходного задания таблицу истинности цифрового комбинационного устройства. В качестве примеров данных методов мы рассмотрели наиболее распространенные цифровые устройства, которые в дальнейшем понадобятся нам для создания более сложных цифровых устройств. Это такие устройства, как дешифраторы, шифраторы и мультиплексоры. Теперь можно перейти к более сложным цифровым устройствам — цифровым устройствам с памятью (последовательностным устройствам). Однако любое из последовательностных устройств требует для своей работы импульсов синхронизации, поэтому сначала рассмотрим, как можно получить сигналы синхронизации.

ГЛАВА 7



Генераторы

При работе цифровых схем часто возникает задача синхронизации моментов изменения или записи сигналов. Для этого можно воспользоваться любым известным генератором периодических сигналов.

Генератор, в принципе, может быть построен на любом усилительном элементе, охваченном положительной обратной связью. Обобщенная схема генератора незатухающих колебаний приведена на рис. 7.1.

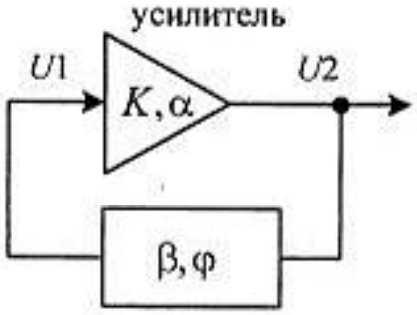


Рис. 7.1. Обобщенная схема генератора

Для самовозбуждения колебаний в такой схеме необходимо выполнить два условия:

1. Баланс амплитуд.
2. Баланс фаз.

Баланс амплитуд выполняется, если произведение коэффициента усиления усилителя K и коэффициента передачи цепи обратной связи β будет больше единицы:

$$|K| \cdot |\beta| \geq 1 \qquad (7.1)$$

Баланс фаз выполняется, если сумма фазового сдвига усилителя α и фазового сдвига цепи обратной связи φ будет равна нулю или кратна 360°:

$$(\alpha + \phi) = 2 \cdot \pi \cdot n,$$

где n — целое число.

В качестве усилительного элемента можно использовать любой активный элемент, обладающий усилением, в том числе транзистор или операционный усилитель. Однако в этом случае потребуется специальное устройство преобразования выходного сигнала генератора к цифровым логическим уровням, используемым в разрабатываемой схеме.

Намного проще было бы использовать для построения тактовых генераторов логические элементы. Так как любые логические элементы обладают усилением, то для построения генераторов можно использовать как инверторы, так и схемы логического "И" и "ИЛИ". В некоторых случаях для построения генераторов используют даже триггеры. Так как от параметров усилительного элемента в значительной степени зависят параметры генератора, то рассмотрим логический инвертор с точки зрения его усилительных параметров.

Усилительные параметры КМОП-инвертора

Основной характеристикой усилителя является его коэффициент усиления и зависимость коэффициента усиления от частоты. Для измерения коэффициента усиления инвертора может быть использована схема, подобная приведенной на рис. 7.2.

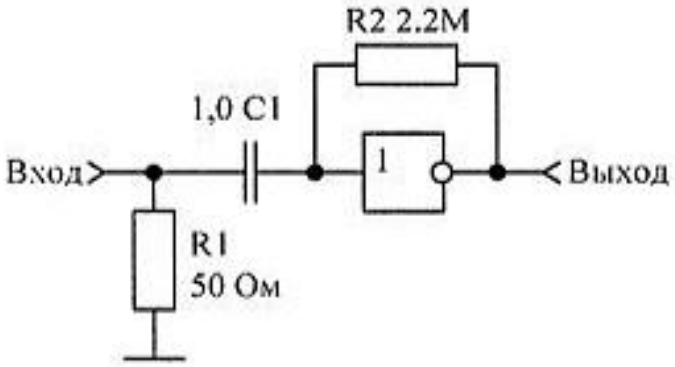


Рис. 7.2. Схема измерения усилительных свойств логического инвертора

В качестве примера усилительных свойств логического инвертора на рис. 7.3 приведена амплитудно-частотная характеристика инвертора 74LVC1GU04. Характеристика взята из материалов, размещенных на сайте фирмы Texas

Instruments. Приведенную амплитудно-частотную характеристику можно считать типовой для КМОП-микросхем.

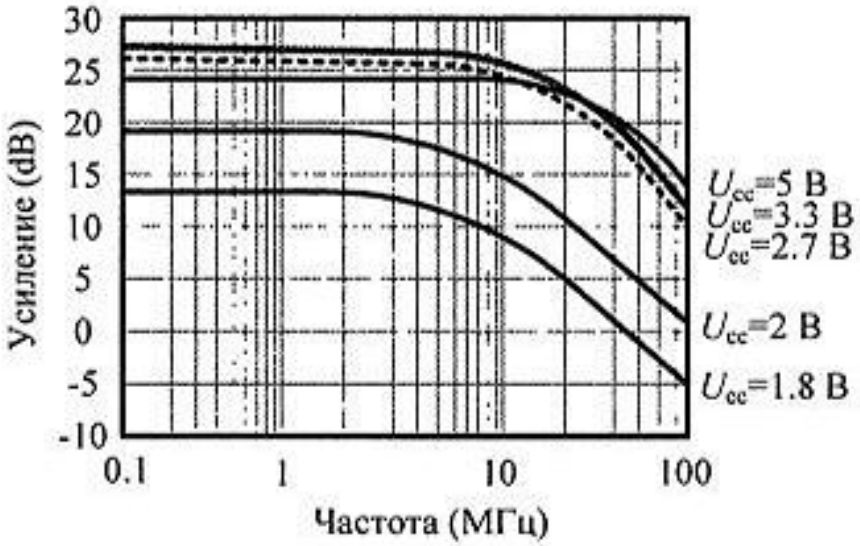


Рис. 7.3. Амплитудно-частотная характеристика инвертора 74LVC1GU04

Как видно из приведенных характеристик, коэффициент усиления инвертора зависит от напряжения питания. Чем меньше напряжение питания микросхемы, тем меньше результирующий коэффициент усиления инвертора.

Кроме того, на амплитудно-частотной характеристике явно наблюдается спад коэффициента усиления на частотах выше 5 МГц. Тем не менее, эта схема пригодна для построения генераторов, работающих на частотах вплоть до сотни мегагерц. Это обусловлено тем, что для возникновения колебаний в схеме генератора достаточно, чтобы коэффициент передачи активного элемента превышал единичное значение (7.1).

Осцилляторные схемы

Рассмотрим схему простейшего генератора. Для его самовозбуждения необходимо обеспечить баланс фаз на заданной частоте. Баланс амплитуд обеспечивается за счет усилительных свойств активного элемента. Генератор может быть выполнен по схеме индуктивной или емкостной трехточки. Такие схемы называются осцилляторными. В настоящее время обычно используется схема емкостной трехточки, как более дешевый вариант реализации генератора. На рис. 7.4 приведена подобная схема, выполненная на биполярном транзисторе.

В этой схеме усилительный элемент (транзистор VT1) включен в схему контура L1, C2, C3, резонансная частота которого и задает частоту генерации. Глубина обратной связи задается соотношением емкостей этого контура и коэффициентом усиления транзистора на заданной частоте.

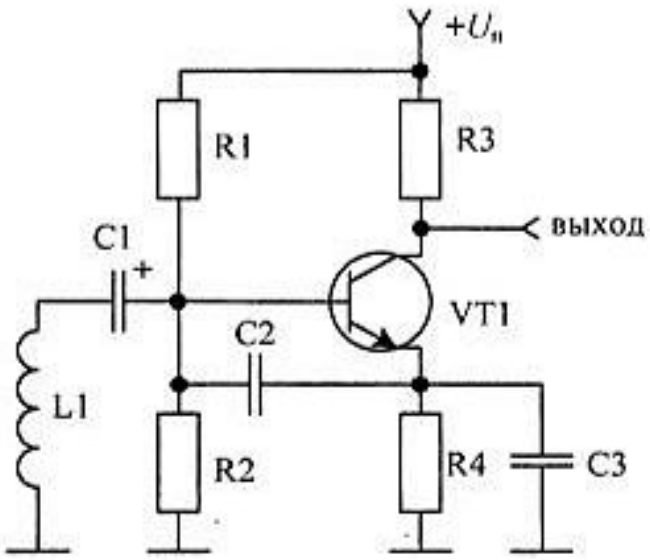


Рис. 7.4. Емкостная трехточка, выполненная на биполярном транзисторе

Приведенная схема генератора незатухающих колебаний достаточно сложна. Это определяется количеством элементов термостабилизации (резисторы R1, R2, R3 и R4) и задания режима по постоянному току (резистор R3 и конденсатор C1). Колебания, формируемые таким генератором, не совсем подходят для синхронизации цифровых микросхем, т. к. на выходе описанного генератора вырабатывается синусоидальное напряжение. Его необходимо преобразовать к прямоугольной форме с логическими уровнями, которые воспринимают цифровые микросхемы.

Генератор можно построить и на основе одиночного логического инвертора. Как уже говорилось в предыдущих главах, любой логический элемент обладает усилением. Этим будет обеспечен баланс амплитуд. Баланс фаз обеспечим точно так же, как и в предыдущей схеме генератора — при помощи резонансного контура.

Схема емкостной трехточки, построенной на основе логического инвертора, с индуктивностью в цепи обратной связи, приведена на рис. 7.5.

При реализации генераторов на логических элементах необходимо следить за тем, чтобы при запуске генератора логический элемент находился в активном режиме. При обычном включении логический инвертор находится в режиме ограничения. В случае применения режима ограничения осуществляется жесткий режим запуска генератора, при котором не возникает самопроизвольных колебаний в схеме, собранной на логическом элементе, и поэтому для их возникновения требуется подать мощный импульс на вход инвертора.

Для самопроизвольного возникновения колебаний в схеме генератора (мягкий режим запуска генератора) необходимо перевести логический элемент в усилительный режим. Для этого инвертор необходимо охватить отрицательной обратной связью по постоянному току. В приведенной на рис. 7.5 схеме

это осуществляется замыканием входа и выхода логического элемента через индуктивность L1. В результате транзисторы инвертора переводятся в активный режим работы.

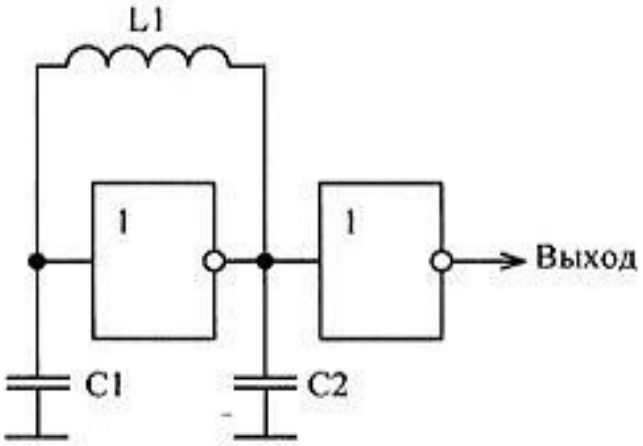


Рис. 7.5. Емкостная трехточка, выполненная на логическом инверторе

Форма сигнала на выходе первого инвертора благодаря фильтрующим свойствам контура будет близка к синусоидальной. Второй инвертор используется для преобразования ее к прямоугольной форме и приведения уровня генерируемого сигнала до цифровых логических уровней. Иными словами, он используется в качестве усилителя-ограничителя, кроме того, этот инвертор выполняет функции развязывающего (буферного) усилителя. Это означает, что изменение параметров нагрузки генератора не будет влиять на генерируемую им частоту.

Известно, что стабильность частоты LC-генератора невысока. Намного большей стабильностью обладают кварцевые генераторы. Схему на одном инверторе можно использовать и для построения генераторов с кварцевой стабилизацией. В этом случае, в емкостной трехточке вместо индуктивности следует включить кварцевый резонатор. Схема кварцевого генератора, выполненного на одном логическом инверторе, приведена на рис. 7.6.

Емкости C1 и C2 в частотозадающей цепочке обычно выбираются в пределах от 10 до 30 пФ. Номинал этих конденсаторов определяется значением емкости кварцедержателя, которая колеблется от 3 до 5 пФ. Соотношение емкостей задает глубину обратной связи, а значит, устойчивость запуска генератора в диапазоне температур. На высоких частотах значения емкостей конденсаторов обычно выбираются равными. В низкочастотных генераторах емкость C1 желательно выбирать меньше емкости конденсатора C2. Это обеспечит большее напряжение на входе инвертора, что, в свою очередь, приведет к меньшему потреблению тока. При необходимости подстройки частоты генератора в качестве емкости C2 может быть использован подстроечный конденсатор.

Кварцевый резонатор не пропускает постоянный ток, поэтому в кварцевом генераторе для обеспечения автоматического запуска генератора приходится использовать дополнительные резисторы. В схеме, приведенной на рис. 7.6, это резисторы R1 и R2. Резистор R1 переводит инвертор в активный режим. Соотношение резисторов R1/R2 определяет коэффициент усиления активного элемента генератора.

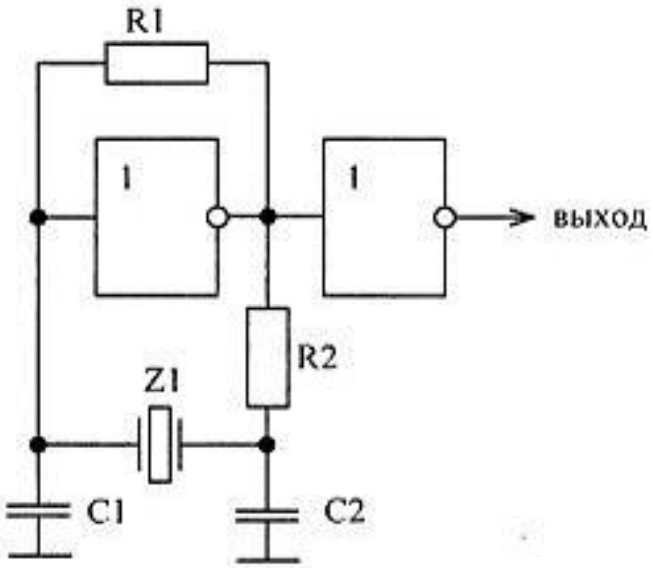


Рис. 7.6. Схема кварцевого генератора

При использовании очень высокочастотных кварцевых резонаторов резистор R2 для облегчения самовозбуждения генератора может отсутствовать. При работе с низкочастотными кварцевыми резонаторами резистор R2 и емкость C2 обеспечивают необходимый фазовый сдвиг и предотвращают самовозбуждение генератора на частоте емкости кварцедержателя. Кроме того, резистор R2 ограничивает мощность, рассеиваемую на кристалле кварца, что позволяет использовать в генераторе малогабаритные кристаллы.

Достаточно часто для уменьшения потребляемого тока возникает необходимость останавливать генератор. В этом случае вместо инвертора в схеме генератора можно использовать логический элемент "2И-НЕ". Подобная схема приведена на рис. 7.7. Именно такая схема обычно используется в современных микросхемах в качестве задающего тактового генератора.

В данной схеме, при подаче на вход логического элемента нулевого потенциала, на его выходе сформируется единичный потенциал, и генерация тактового сигнала прекратится. При подаче на этот же вход единичного потенциала, потенциал на выходе схемы будет определяться обратной связью, выполненной на резисторе R2, и в случае применения КМОП-микросхемы будет близок к половине напряжения питания. Это приведет к формированию условий самовозбуждения колебаний, и генератор возобновит свою работу.

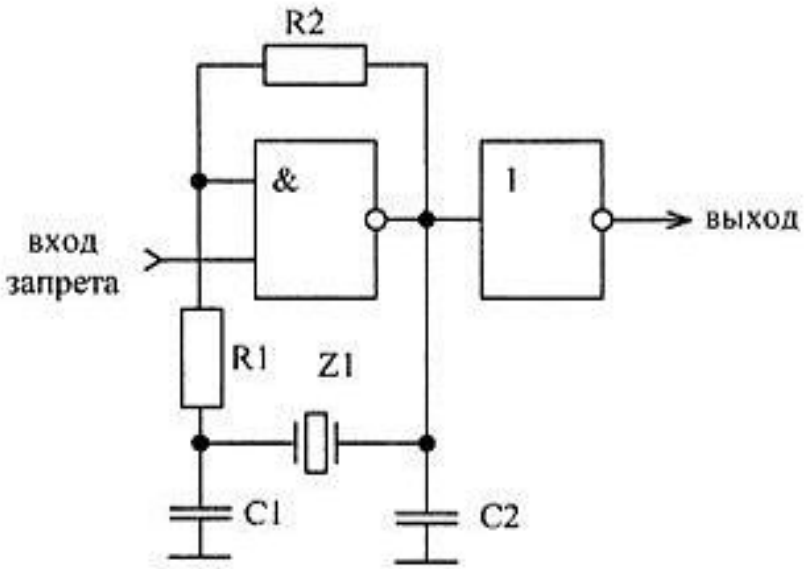


Рис. 7.7. Схема кварцевого генератора, выполненная на логическом элементе "2И-НЕ"

Мультивибраторы

Еще одной распространенной схемой генераторов, выполненных на логических элементах, является схема мультивибратора. В этой схеме для реализации положительной обратной связи используются два инвертора. Каждый из них осуществляет поворот фазы генерируемого сигнала на 180°. Схема мультивибратора приведена на рис. 7.8.

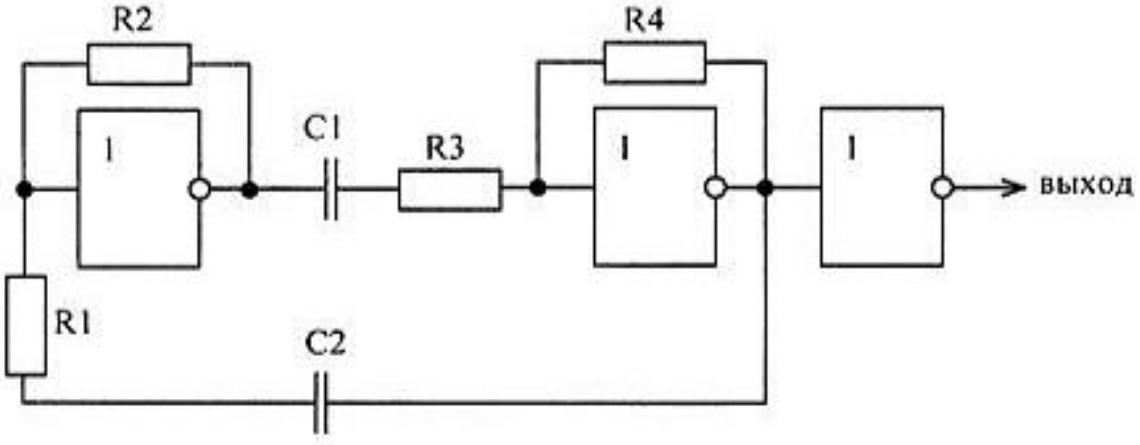


Рис. 7.8. Схема мультивибратора на двух логических инверторах

Коэффициент усиления каждого из усилителей определяется соотношением резисторов R2/R1 и R4/R3. В этой схеме возможна независимая регулировка частоты и скважности генерируемых колебаний. Длительность импульсов и длительность паузы между импульсами регулируется независимо при помо-

щи RC цепочек R2·C2 и R4·C1. Период следования импульсов T определяется как сумма двух времен заряда конденсаторов:

$$T = t_{зар1} + t_{зар2},$$

где $t_{зар1} = \tau 1 \cdot \ln\left(\frac{U^1}{U_{пор}}\right)$;

$$t_{зар2} = \tau 2 \cdot \ln\left(\frac{U^1}{U_{пор}}\right).$$

Если скважность генерируемых колебаний не важна, то можно упростить схему мультивибратора, применив второй инвертор по прямому назначению. Так как при реализации схемы генератора нас интересует максимальный петлевой коэффициент усиления (7.1), то последовательный резистор мы тоже можем исключить. Для обеспечения автоматического запуска генератора в схеме остается резистор, включенный с выхода на вход первого инвертора. В этом случае схема мультивибратора примет вид, показанный на рис. 7.9.

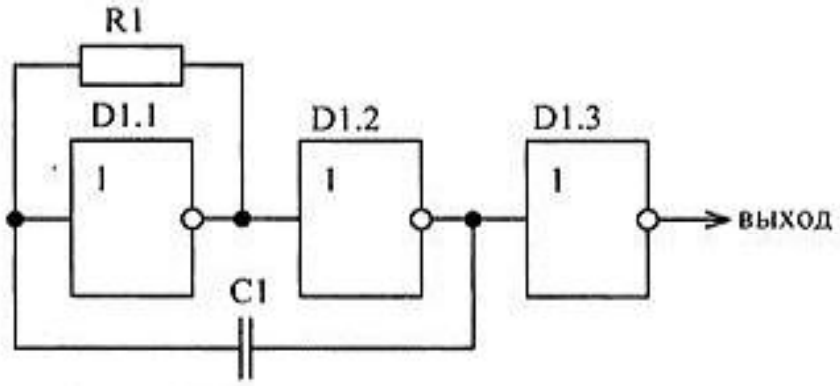


Рис. 7.9. Упрощенная схема мультивибратора

В схеме, приведенной на рис. 7.9, можно изменять только частоту генерируемых импульсов. Эта частота будет определяться постоянной времени $\tau = R1C1$. Скважность генерируемого данной схемой колебания будет зависеть только от соотношения токов нуля и единицы выбранного логического элемента. Период T импульсов, вырабатываемых мультивибратором, определяется в первом приближении постоянной времени $\tau = R1C1$ ($T = \alpha \cdot \tau$, где α обычно имеет значение 1...2). Частоту следования импульсов можно определить (с точностью до 10%) из выражения:

$$f = \frac{1}{2R1C1}$$

Довольно часто требуется возможность получить генератор, выходная частота которого могла бы изменяться в достаточно широких пределах. В этом случае в качестве частотозадающего элемента в генераторе может быть использован элемент с изменяемыми параметрами, например варикап в качестве емкости или полевой транзистор в качестве резистора. Схема такого генератора, управляемого напряжением, приведена на рис. 7.10.

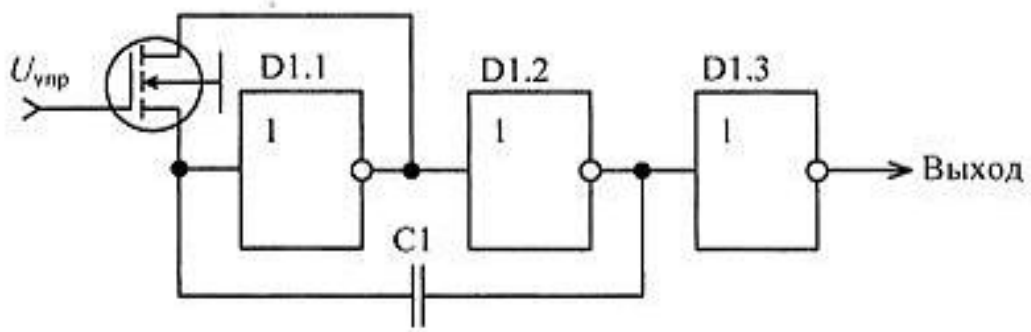


Рис. 7.10. Схема генератора, управляемого напряжением

Учитывая, что сопротивление полевого транзистора может изменяться в пределах от 10 Ом до 10 МОм, то генерируемая частота тоже может изменяться в десятки и сотни раз. Однако следует учесть, что такой генератор может быть использован только в цифровых схемах, не связанных с обработкой сигналов, т. к. его спектральные характеристики оставляют желать лучшего. Обычно такая схема используется в цепях умножения частоты внутри цифровых микросхем повышенной производительности. Примером специализированных микросхем-генераторов могут служить отечественные микросхемы 531ГГ1 и 564ГГ1.

В схеме на мультивибраторе можно использовать и кварцевую стабилизацию частоты. Для этого нужно кварцевый резонатор включить в цепь обратной

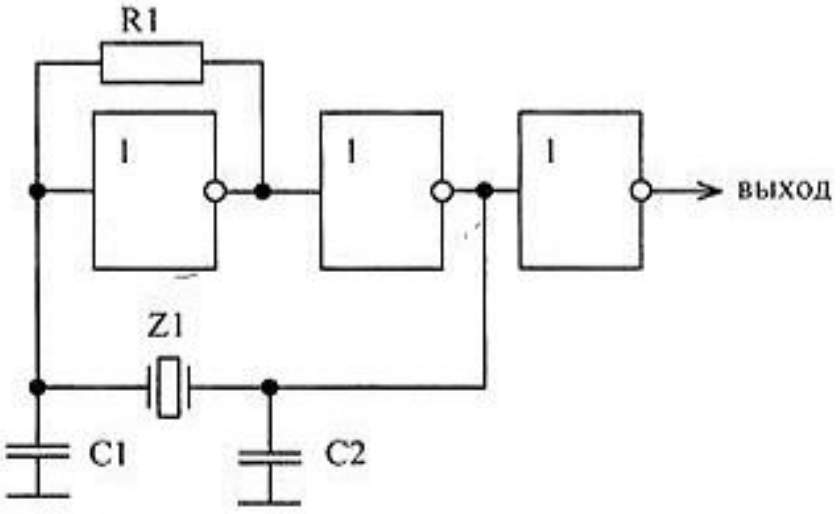


Рис. 7.11. Схема мультивибратора с кварцевой стабилизацией частоты

связи. Схема мультивибратора с кварцевой стабилизацией частоты приведена на рис. 7.11.

В этой схеме кварцевый резонатор используется в качестве фильтра, включенного в цепь обратной связи. В результате самовозбуждение схемы возможно только на частотах, которые пропускаются этим фильтром.

Особенности кварцевой стабилизации частоты генераторов

При разработке кварцевого генератора следует обращать внимание, что кварцевый генератор на основе мультивибратора работает несколько по другим принципам по сравнению со схемой емкостной трехточки. Если в емкостной трехточке кварцевый резонатор используется в качестве индуктивности, входящей в колебательный контур резонатора, то в схеме мультивибратора кварцевый резонатор используется в качестве узкополосного фильтра в цепи обратной связи. *Это приводит к тому, что один и тот же резонатор, включенный в схему мультивибратора или в схему емкостной трехточки, будет генерировать различные частоты!*

Для того чтобы разобраться с этим явлением, давайте рассмотрим эквивалентную схему кварцевого резонатора и характеристику зависимости сопротивления кварцевого резонатора от частоты. Эквивалентная схема кварцевого резонатора приведена на рис. 7.12.

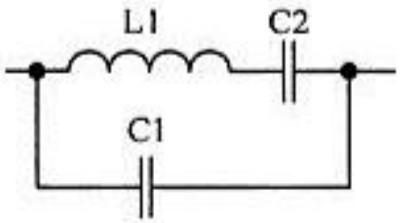


Рис. 7.12. Эквивалентная схема кварцевого резонатора

Элементы L1 и C2 определяются механическими параметрами кристалла кварцевого резонатора, а емкость C1 — это конструктивная емкость кварцедержателя и электродов. Емкость кварцедержателя много больше эквивалентной емкости последовательного контура, отображающего резонансные свойства кварцевого резонатора. Его комплексное сопротивление может быть записано в виде следующего выражения:

$$Z = \frac{j}{\omega} \cdot \frac{\omega^2 \cdot L1 \cdot C2 - 1}{(C1 + C2) - \omega^2 \cdot L \cdot C1 \cdot C2}$$

Напомним, что при последовательном резонансе сопротивление схемы становится нулевым. Частота последовательного резонанса приведенной на рис. 7.12 схемы может быть получена из следующего выражения:

$$f_{\text{посл}} = \frac{1}{2\pi\sqrt{L1 \cdot C2}}$$

При параллельном резонансе сопротивление резонатора стремится к бесконечности. Частота параллельного резонанса рассматриваемой схемы может быть определена из формулы:

$$f_{\text{пар}} = f_{\text{посл}}\sqrt{1 + \frac{C2}{C1}}$$

Так как в это выражение входят паразитные элементы (емкость C1), то стабильность частоты параллельного резонанса меньше стабильности последовательного резонанса, определяемого только механическими свойствами кварцевого кристалла.

Характеристика зависимости комплексного сопротивления схемы, приведенной на рис. 7.12 от частоты, приведена на рис. 7.13.

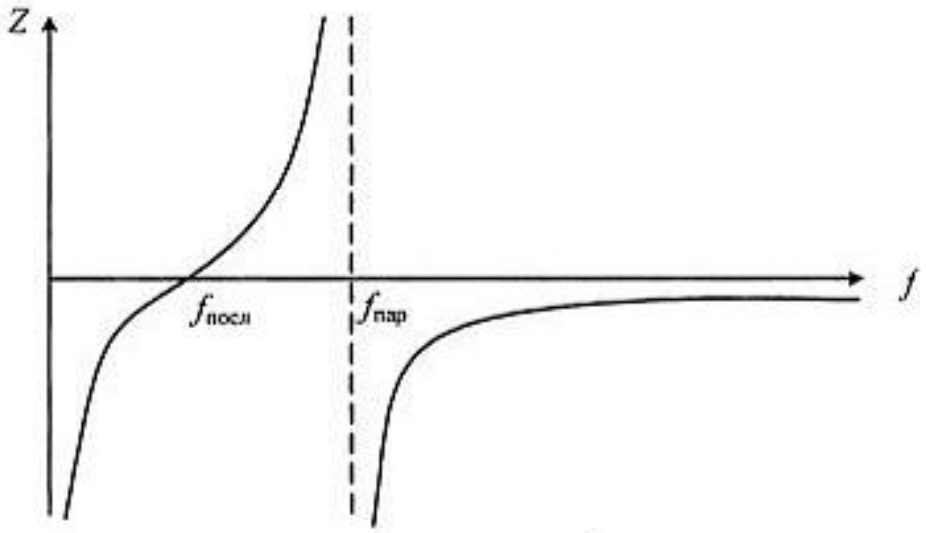


Рис. 7.13. Зависимость сопротивления кварцевого резонатора от частоты

В схеме мультивибратора используется последовательный резонанс кварцевого резонатора (собственные колебания кристалла), а в осцилляторной схеме генерация производится на частоте, близкой к параллельному резонансу контура, образуемого индуктивностью резонатора и емкостью кварцедержателя. Эти частоты близки, но не совпадают по определению. В результате этого, частоты генерируемых колебаний будут отличаться между собой. Обычно разность частот последовательного и параллельного резонансов составляет

около 1 кГц. Настолько же будут отличаться и частоты кварцевых генераторов, построенных по схеме мультивибратора и схеме емкостной трехточки.

Так как генератор, собранный по схеме мультивибратора, возбуждается на собственной частоте кварцевого кристалла, то стабильность кварцевого мультивибратора будет выше по сравнению с осцилляторной схемой, т. к. на частоту последовательного резонанса кварцевого резонатора не влияют внешние паразитные емкости. Однако в генераторе, собранном по схеме мультивибратора, возможно самовозбуждение генератора на частоте, далеко отстоящей от резонансной частоты кварцевого резонатора. Эта частота обуславливается емкостью кварцедержателя и входным сопротивлением активного элемента, поэтому в схеме мультивибратора необходимо предусматривать специальные меры для борьбы с этим явлением.

При построении схем генераторов следует отметить, что они являются мощными источниками помех, поэтому их обычно экранируют. Цепи питания микросхем, на которых реализуются генераторы, обязательно содержат фильтрующие высокочастотные конденсаторы. Часто для лучшей фильтрации по цепи питания кроме конденсаторов используются фильтрующие дроссели.

Для уменьшения помех используются и конструктивные меры. Например, проводники до кварцевого резонатора стараются делать как можно короче, рядом с цепью генерируемого сигнала прокладывают корпусные проводники. Таким образом, фактически образуется полосковая (или волноводная) линия передачи. Однако нельзя замыкать корпусной проводник вокруг потенциального проводника, иначе будет образована петлевая антенна, и мы, вместо подавления помех, улучшим их излучение.

Одновибраторы

При работе с цифровыми устройствами достаточно часто требуется формировать одиночные импульсы определенной длительности. Эту задачу выполняют специальные устройства — формирователи импульсов. Такие устройства формируют импульс заданной длительности из произвольного входного импульса. Простейшие формирователи импульсов могут быть реализованы на логических элементах.

Укорачивающие одновибраторы

В укорачивающих одновибраторах обязательным условием является то, что длительность входного импульса должна быть больше длительности формируемого импульса. Рассмотрим схему, приведенную на рис. 7.14.

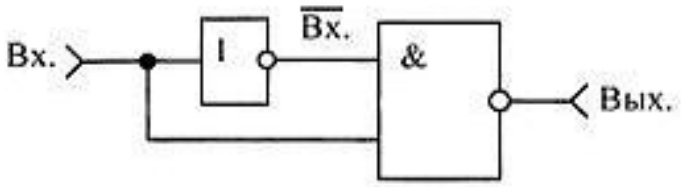


Рис. 7.14. Схема укорачивающего одновибратора

Если бы логические элементы не обладали задержкой, то на выходе такой схемы постоянно присутствовал единичный логический уровень. Однако это не так. Сигнал на выходе инвертора задержан по отношению к его входу. Временные диаграммы сигналов на входе и выходе инвертора, а также на выходе схемы "2И-НЕ" приведены на рис. 7.15.

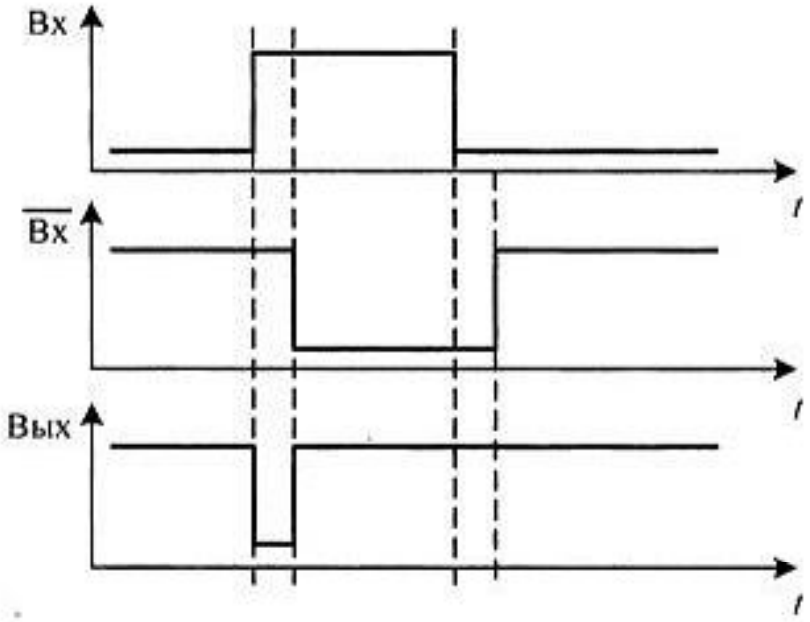


Рис. 7.15. Временные диаграммы укорачивающего одновибратора

Как видно из приведенных временных диаграмм, одновибратор, схема которого приведена на рис. 7.14, вырабатывает одиночный импульс по переднему фронту входного сигнала. Длительность импульса на выходе такой схемы будет равна времени задержки инвертора.

Если требуется длительность выходного импульса, бóльшая времени задержки одиночного инвертора, то можно применить дополнительные элементы задержки на пассивных RC-элементах. Пример подобной схемы приведен на рис. 7.16, а временные диаграммы работы этой схемы — на рис. 7.17.

Длительность выработанного формирователем импульса можно вычислить исходя из условия разряда конденсатора С. Действительно, пока конденсатор С разряжается до уровня порогового напряжения логического элемента $U_{пор}$, напряжение U_C воспринимается элементом "2И-НЕ" как уровень логической единицы, и на его выходе поддерживается уровень логического нуля.

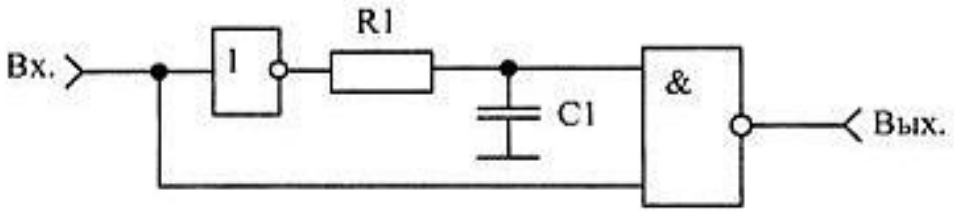


Рис. 7.16. Схема укорачивающего одновибратора с использованием RC элементов задержки

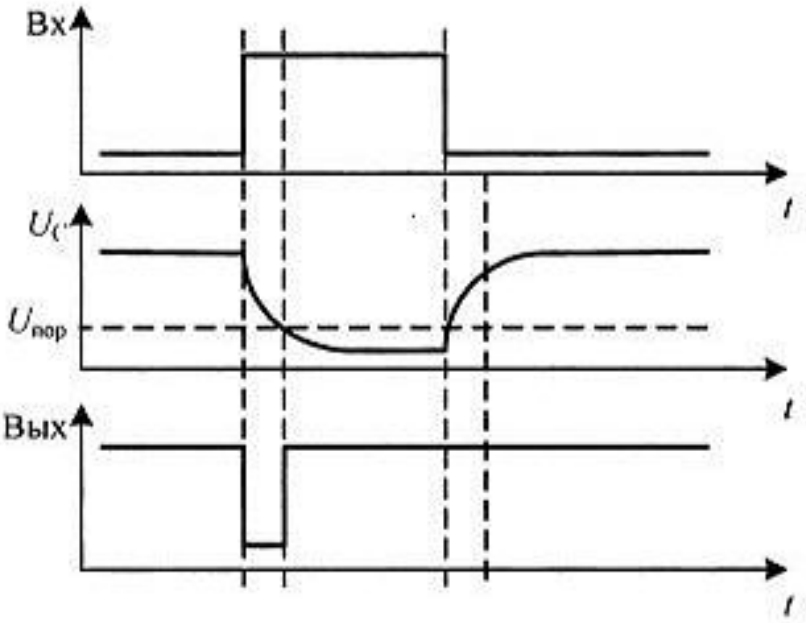


Рис. 7.17. Временные диаграммы укорачивающего одновибратора с использованием RC элементов задержки

С течением времени напряжение на конденсаторе С падает до значения порогового напряжения логического элемента $U_{пор}$. В этот момент на выходе элемента "2И-НЕ" появится уровень логической единицы. Если считать, что напряжение до начала разряда на конденсаторе было равно напряжению логической единицы U^1, то изменение напряжения U_C с течением времени можно представить как:

$$U_C = U^1 e^{-\frac{t}{\tau}},$$

где $\tau = RC$.

Следовательно

$$e^{\frac{t}{\tau}} = \frac{U^1}{U_C}.$$

Длительность импульса равна времени разряда конденсатора до порогового значения $U_{пор}$. Эту длительность можно определить, воспользовавшись следующей формулой:

$$t_и = RC \ln \frac{U^1}{U_{пор}}$$

Расширяющие одновибраторы

В расширяющих одновибраторах длительность входного (запускающего) импульса должна быть короче длительности формируемого импульса. Схема расширяющего одновибратора приведена на рис. 7.18, а временные диаграммы сигналов на его входе и выходе — на рис. 7.19.

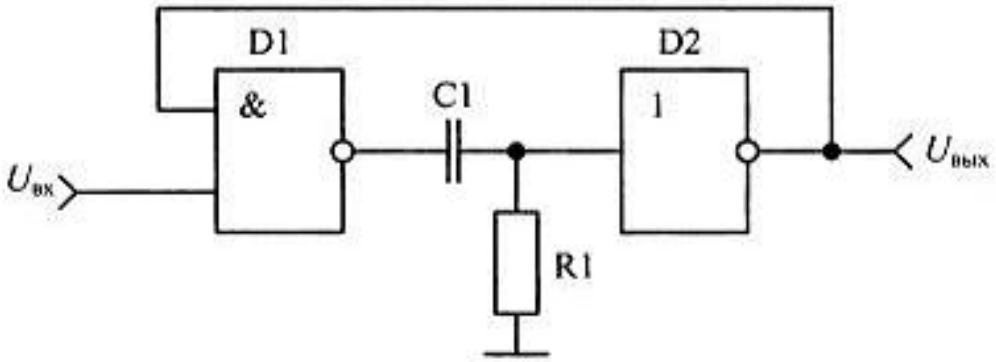


Рис. 7.18. Схема расширяющего одновибратора

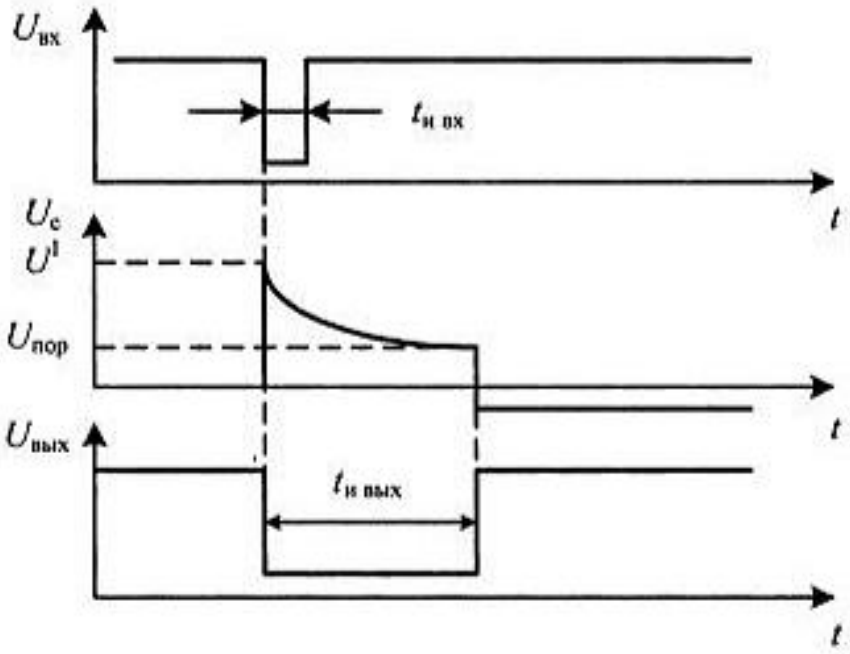


Рис. 7.19. Временные диаграммы расширяющего одновибратора

Расширяющий одновибратор выполнен на двух логических элементах "2И-НЕ". Схема охвачена положительной обратной связью, т. к. выход второго элемента соединен с входом первого. В исходном состоянии на выходе элемента D2 формируется уровень логической единицы, а на выходе элемента D1 — уровень логического нуля, т. к. на обоих входах логического элемента D1 присутствуют логические единицы. При поступлении на вход запускающего импульса с нулевым потенциалом на выходе элемента D1 появится уровень логической единицы, который через конденсатор C1 поступит на вход второго логического элемента. Элемент D2 инвертирует этот сигнал, и уровень "0" по цепи обратной связи подается на второй вход элемента D1. Теперь, даже если на входе снова появится уровень логической единицы, на выходе инвертора D1 будет сохраняться высокое напряжение.

На выходе элемента D2 уровень логического нуля будет присутствовать до тех пор, пока конденсатор C не зарядится до уровня $U_{C1} = U^1 - U_{пор}$, а напряжение на резисторе R не достигнет порогового уровня $U_{пор}$ (рис. 7.18).

Длительность выходного импульса одновибратора может быть определена с помощью выражения:

$$t_и = C1\left(R1 + R_{вых}\right)\ln\frac{U^1}{U_{пор}},$$

где $R_{вых}$ — выходное сопротивление первого элемента;

$U_{пор}$ — пороговое напряжение логического элемента.

Применение одновибраторов

Одновибраторы обычно используются для синхронизации цифровых устройств последовательностного типа, которые будут рассмотрены позднее. При помощи одновибраторов можно укорачивать и расширять импульсы синхронизирующего колебания, формируя тем самым заданную скважность колебания.

Укорочение длительности синхронизующего импульса обычно требуется для четкого определения момента записи или изменения информационного цифрового сигнала. Расширение входного импульса обычно применяется для задержки входного сигнала на заданный интервал времени.

С помощью укорачивающих одновибраторов можно формировать многофазные синхронизирующие сигналы. Рассмотрим пример такой схемы. Пусть требуется сформировать последовательность импульсов, приведенную на рис. 7.20.

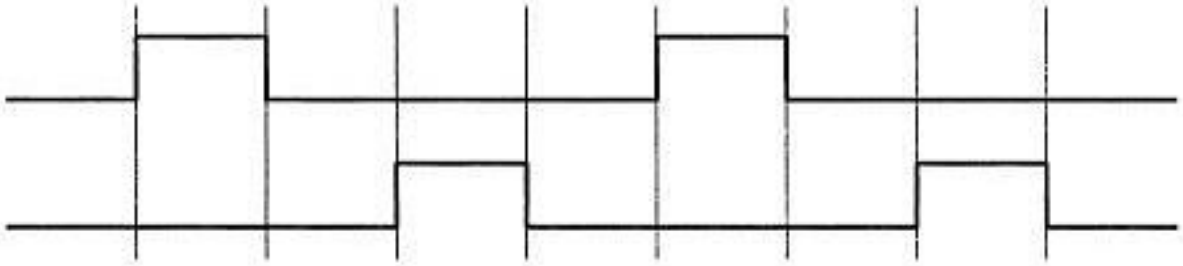


Рис. 7.20. Двухтактная синхронизирующая последовательность

Для формирования такой последовательности воспользуемся двумя укорачивающими одновибраторами, работающими по переднему и заднему фронту входного сигнала соответственно. В результате задержка выходных сигналов друг относительно друга будет определяться длительностью импульсов входного сигнала. Получившаяся схема тактовой синхронизации приведена на рис. 7.21.

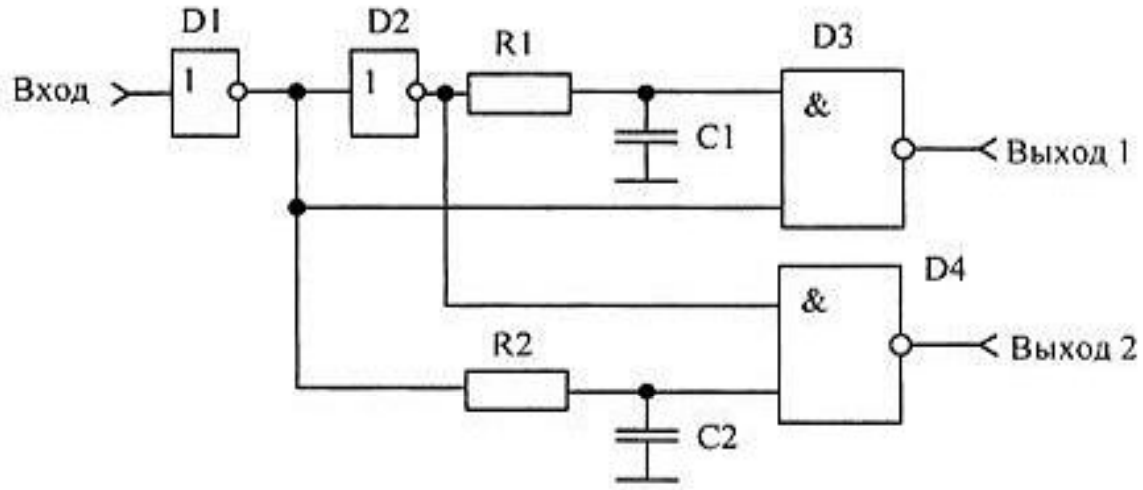


Рис. 7.21. Схема, формирующая двухтактную синхронизирующую последовательность

В данной схеме инвертор D1 на входе предназначен для устранения влияния параметров источника тактового сигнала на длительность формируемого импульса. Изменение фронта срабатывания нижнего укорачивающего одновибратора R2, C2, D4 достигнуто путем задержки сигнала не на выходе инвертора D2, а на его входе.

Итоги

В данной главе мы рассмотрели особенности реализации генераторов на логических элементах. Это позволит нам пользоваться последовательностями

прямоугольных импульсов для обеспечения правильной работы цифровых устройств, которые мы рассмотрим в последующих главах. Хотелось бы, прежде чем перейти к изучению дальнейшего материала, подчеркнуть, что здесь мы рассмотрели только основы схемотехники, реализующей прямоугольные колебания. Требования, предъявляемые к аналого-цифровым устройствам, заставляют применять несколько другие схемные и особенно конструктивные решения, однако обсуждение этих вопросов выходит за рамки данной книги.

ГЛАВА 8

Цифровые схемы последовательностного типа

Последовательностные устройства — это цифровые устройства с памятью. В них выходной сигнал определяется не только текущим состоянием входа, но и рядом предыдущих значений. Обычно в состав последовательностных устройств входят комбинационные устройства и запоминающие ячейки.

Последовательностные устройства позволяют изменять свое состояние в определенные, строго фиксированные моменты времени. Это позволяет учитывать времена задержки прохождения цифровых сигналов через комбинационные цифровые устройства. При этом обычно определяется наибольшее время распространения и изменение состояния цифрового устройства производится с периодом, большим или равным этому времени.

Простейшее последовательностное устройство — это триггер. Его особенностью является способность бесконечно долго находиться в одном из двух устойчивых состояний. Приняв одно состояние за ноль, другое за единицу, можно считать, что *триггер хранит один бит информации.*

Триггеры

Триггеры предназначены для запоминания двоичной информации. Использование триггеров позволяет реализовывать устройства оперативной памяти (т. е. памяти, информация в которой хранится только на время вычислений). Однако триггеры могут использоваться и для построения некоторых цифровых устройств с памятью, таких как счетчики, преобразователи последовательного кода в параллельный или цифровые линии задержки.

Простейшая схема, позволяющая запоминать двоичную информацию, может быть построена на двух инверторах, охваченных положительной обратной связью. Эта схема приведена на рис. 8.1.

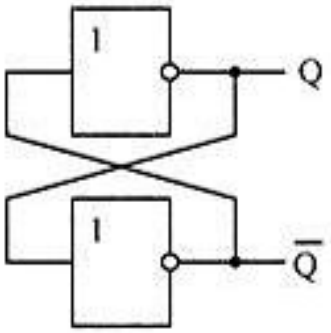


Рис. 8.1. Схема простейшего триггера, построенного на инверторах

Триггер может находиться только в двух устойчивых состояниях — на выходе Q присутствует логическая единица и на выходе Q присутствует логический ноль. Если логическая единица присутствует на выходе Q, то на инверсном выходе появится логический ноль, который после очередного инвертирования подтверждает уровень логической единицы на выходе Q. И наоборот, если на выходе Q присутствует логический ноль, то на инверсном выходе будет присутствовать логическая единица.

Может возникнуть вопрос: в каком же состоянии будет такой триггер при включении питания. Это зависит от многих факторов, таких как конструктивная емкость, подключенная ко входу инверторов, распределение напряжения по шине питания, влияния внутренних шумов и т. д. В результате воздействия всех этих факторов триггер при включении питания может оказаться как в нулевом, так и в единичном состоянии.

Состояние триггера будет сохраняться до тех пор, пока на схему подано напряжение питания. Но вот вопрос — а как записывать в такой триггер первоначальное значение? Для этого нам потребуются входы записи нуля и записи единицы.

RS-триггеры

RS-триггер получил название по имени своих входов. Вход S (от англ. *Set* — установить) позволяет устанавливать выход триггера Q в единичное состояние. Вход R (от англ. *Reset* — сбросить) позволяет сбрасывать выход триггера Q (от англ. *Quit* — выход) в нулевое состояние.

Для реализации RS-триггера воспользуемся логическими элементами "2И-НЕ". Его принципиальная схема приведена на рис. 8.2.

Рассмотрим работу изображенной на рис. 8.2 схемы подробнее. Пусть на входы R и S подаются единичные потенциалы. Если на выходе верхнего логического элемента "2И-НЕ" Q присутствует логический ноль, то на выходе нижнего логического элемента "2И-НЕ" появится логическая единица. Эта единица подтвердит логический ноль на выходе Q. Если на выходе верхнего логического элемента "2И-НЕ" Q первоначально присутствует логическая

единица, то на выходе нижнего логического элемента "2И-НЕ" появится логический ноль. Этот ноль подтвердит логическую единицу на выходе Q, т. е. при единичных входных уровнях схема RS-триггера работает точно так же, как и схема триггера, выполненная на инверторах (см. рис. 8.1).

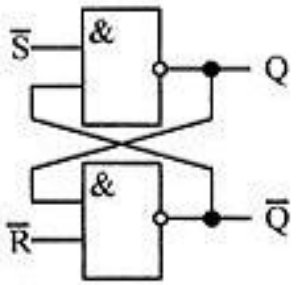


Рис. 8.2. Схема RS-триггера, построенного на логических элементах "2И-НЕ"

Подадим на вход S нулевой потенциал. Согласно таблице истинности логического элемента "2И-НЕ", на выходе Q появится единичный потенциал. Это приведет к появлению на инверсном выходе триггера нулевого потенциала. Теперь, даже если снять нулевой потенциал с входа S, на выходе триггера останется единичный потенциал, а значит, мы записали в триггер логическую единицу.

Точно так же можно записать в триггер и логический ноль. Для этого следует воспользоваться входом R. Так как активный уровень на рассмотренных входах триггера оказался нулевым, то входы R и S — инверсные. Составим таблицу истинности RS-триггера. Входы R и S в этой таблице будем рассматривать прямые, т. е. запись нуля, и запись единицы будут осуществляться единичными потенциалами (табл. 8.1).

***Таблица 8.1.** Таблица истинности RS-триггера*

R	S	Q(t)	Q(t+1)	Пояснения
0	0	0	0	Режим хранения информации R=S=0
0	0	1	1	
0	1	0	1	Режим установки единицы S=1
0	1	1	1	
1	0	0	0	Режим записи нуля R=1
1	0	1	0	
1	1	0	*	R=S=1 запрещенная комбинация
1	1	1	*	

Символ "*" в таблице означает неопределенное состояние.

RS-триггер можно построить и на логических элементах "2ИЛИ-НЕ". Схема RS-триггера, построенного на логических элементах "2ИЛИ-НЕ", приведена на рис. 8.3. Единственное отличие в работе этой схемы по сравнению со схемой, рассмотренной ранее, будет заключаться в том, что сброс и установка триггера производятся единичными логическими уровнями в полном соответствии с таблицей истинности RS-триггера, приведенной в табл. 8.1. Эти особенности связаны с принципами работы инверсной логики, которые рассматривались в предыдущих главах.

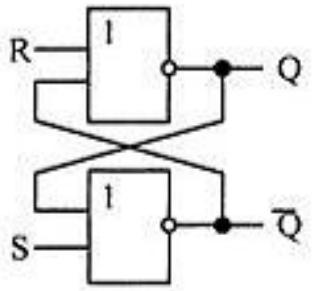


Рис. 8.3. Схема простейшего триггера на схемах "2ИЛИ-НЕ". Входы R и S прямые (активный уровень "1")

Так как RS-триггер при его реализации как на логических элементах "2И-НЕ", так и на элементах "2ИЛИ-НЕ" работает одинаково, то его условно-графическое изображение на принципиальных схемах тоже одинаково. Условно-графическое изображение RS-триггера на принципиальных схемах приведено на рис. 8.4.

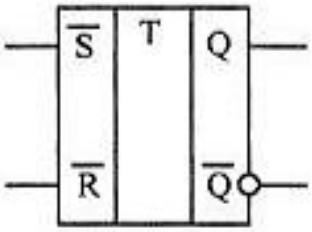


Рис. 8.4. Условно-графическое обозначение RS-триггера

Синхронные RS-триггеры

Схема RS-триггера позволяет запоминать состояние логической схемы, но т. к. при изменении входных сигналов может возникать переходный процесс (в цифровых схемах этот процесс называется "опасные гонки"), то запоминать состояния логической схемы нужно строго в определенные моменты времени, когда все переходные процессы закончены, и сигнал на выходе комбинационной схемы соответствует выполняемой ею функции. Это означает, что большинство цифровых схем требуют сигнала синхронизации (так-

тового сигнала). Все переходные процессы в комбинационной логической схеме должны закончиться за время периода синхросигнала, подаваемого на входы триггеров.

Триггеры, запоминающие входные сигналы только в момент времени, определяемый сигналом синхронизации, называются синхронными. Для того чтобы отличать от них рассмотренные ранее варианты: RS-триггер и триггер Шмитта получили название асинхронных.

Формирование синхронизирующих сигналов с различной частотой и скважностью при помощи генераторов и одновибраторов было рассмотрено в предыдущих главах. Теперь покажем, как управлять работой триггеров с помощью разрешающих (синхронизирующих) сигналов. Для этого нам потребуется схема, пропускающая входные сигналы только при наличии синхронизирующего сигнала. Такую схему мы уже использовали при построении схем мультиплексоров и демультиплексоров. Это логический элемент "2И". Принципиальная схема синхронного RS-триггера, построенного на элементах "2И-НЕ", приведена на рис. 8.5.

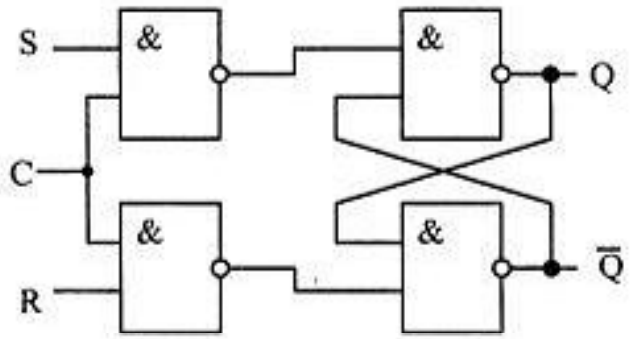


Рис. 8.5. Схема синхронного RS-триггера

В табл. 8.2 приведена таблица истинности синхронного RS-триггера. В этой таблице символ X означает, что значения логических уровней на данном входе не важны. Они не влияют на работу триггера.

Таблица 8.2. *Таблица истинности синхронного RS-триггера*

C	R	S	Q(t)	Q(t+1)	Пояснения
0	X	X	0	0	Режим хранения информации
0	X	X	1	1	
1	0	0	0	0	Режим хранения информации
1	0	0	1	1	
1	0	1	0	1	Режим установки единицы S=1
1	0	1	1	1	

Таблица 8.2 (окончание)

C	R	S	Q(t)	Q(t+1)	Пояснения
1	1	0	0	0	Режим записи нуля R=1
1	1	0	1	0	
1	1	1	0	*	R=S=1 запрещенная комбинация
1	1	1	1	*	

Символ "*" в таблице означает неопределенное состояние.

Как уже отмечалось в предыдущем разделе, RS-триггеры могут быть реализованы на различных видах логических элементов. При этом логика работы триггера не изменяется. RS-триггеры часто выпускаются в виде готовых микросхем (или реализуются внутри БИС в виде готовых модулей), поэтому на принципиальных схемах синхронные триггеры обычно изображаются в виде условно-графических обозначений. Условно-графическое обозначение синхронного RS-триггера приведено на рис. 8.6.

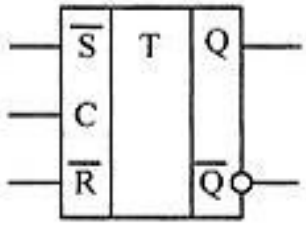


Рис. 8.6. Условно-графическое обозначение синхронного RS-триггера

Статические D-триггеры

В RS-триггерах для записи логического нуля и логической единицы требуются разные входы, что не всегда удобно. При записи и хранении данных один бит может принимать значение как нуля, так и единицы. Для передачи и приема бита достаточно одного проводника. Как мы уже видели ранее, сигналы установки и сброса триггера не должны появляться одновременно, поэтому можно объединить эти входы в один при помощи инвертора, как это показано на рис. 8.7.

Такой триггер получил название D-триггер (защелка). Название происходит от английского слова *delay* — задержка. Конкретное значение времени задержки цифрового сигнала 'D' определяется частотой следования импульсов синхронизации 'C'. Условно-графическое обозначение D-триггера на принципиальных схемах приведено на рис. 8.8.

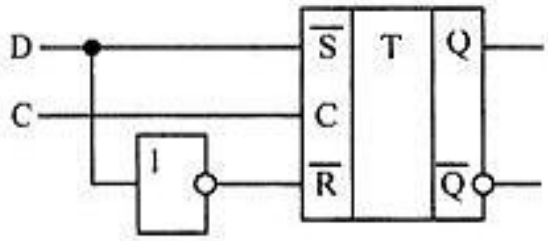


Рис. 8.7. Схема D-триггера

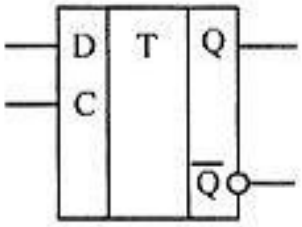


Рис. 8.8. Условно-графическое обозначение D-триггера

Таблица истинности D-триггера достаточно проста, она приведена в табл. 8.3. Как видно из этой таблицы, D-триггер способен запоминать по сигналу синхронизации и хранить один бит дискретной информации.

Таблица 8.3. *Таблица истинности D-триггера*

C	D	Q(t)	Q(t+1)	Пояснения
0	X	0	0	Режим хранения информации
0	X	1	1	
1	0	X	0	Режим записи информации
1	1	X	1	

Следует отметить, что отдельный инвертор при реализации триггера на логических ТТЛ элементах обычно не используется, т. к. самый распространенный элемент ТТЛ логики — это элемент "2И-НЕ". Принципиальная схема D-триггера на элементах "2И-НЕ" приведена на рис. 8.9.

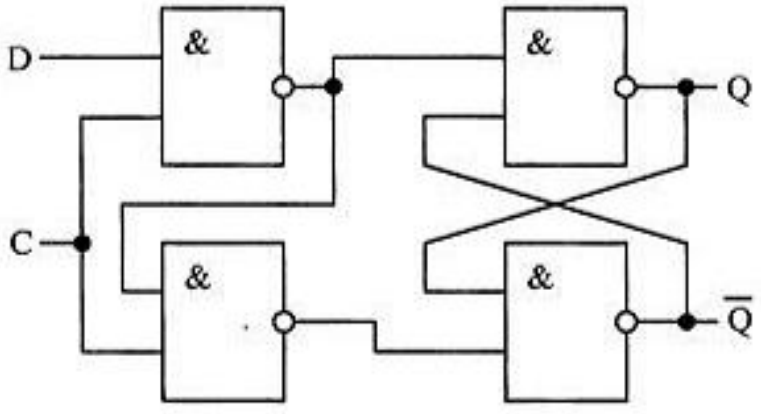


Рис. 8.9. Схема D-триггера, реализованная на ТТЛ элементах

Еще проще реализуется D-триггер на КМОП-логических элементах. В КМОП-микросхемах вместо логических элементов "И" используются обычные транзисторные ключи. Схема D-триггера, выполненная по КМОП-технологии приведена на рис. 8.10.

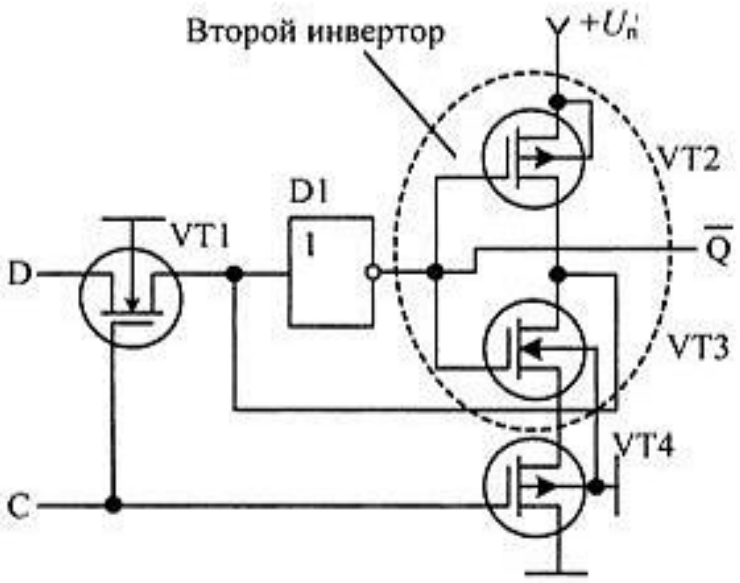


Рис. 8.10. Схема D-триггера, реализованная на КМОП-элементах

При подаче высокого уровня синхросигнала на вход С транзистор VT1 открывается и обеспечивает передачу сигнала с входа D на инверсный выход Q через инвертор D1. Транзистор VT2 при этом закрыт и отключает второй инвертор, собранный на транзисторах VT2 и VT3. При подаче низкого потенциала на вход С включается второй инвертор, который вместе с инвертором D1 и образует триггер.

Во всех рассмотренных схемах синхронных триггеров синхросигнал работает по уровню, поэтому рассмотренные триггеры называются триггерами, работающими по уровню. Еще одно название таких триггеров, пришедшее в отечественную литературу из зарубежной — триггеры-защелки. Легче всего объяснить происхождение этого названия по временным диаграммам входных и выходного сигналов D-триггера, приведенным на рис. 8.11.

По этим временным диаграммам видно, что триггер-защелка хранит данные на выходе только при нулевом потенциале на входе синхронизации. Если же на вход синхронизации подать высокий уровень, то напряжение на выходе триггера будет повторять напряжение, подаваемое на вход этого триггера. Входное напряжение запоминается в триггере только в момент изменения уровня напряжения на входе синхронизации С с высокого уровня на низкий. Входные данные как бы "защелкиваются" в этот момент, отсюда и название — триггер-защелка. Принципиально в этой схеме входной переходной процесс может беспрепятственно проходить на выход триггера. Поэтому там, где это важно, необходимо сокращать длительность импульса синхронизации до минимума при помощи одновибраторов, схемы которых мы рассматривали ранее. Чтобы преодолеть такое ограничение на длительность сигнала синхронизации, были разработаны триггеры, работающие по фронту.

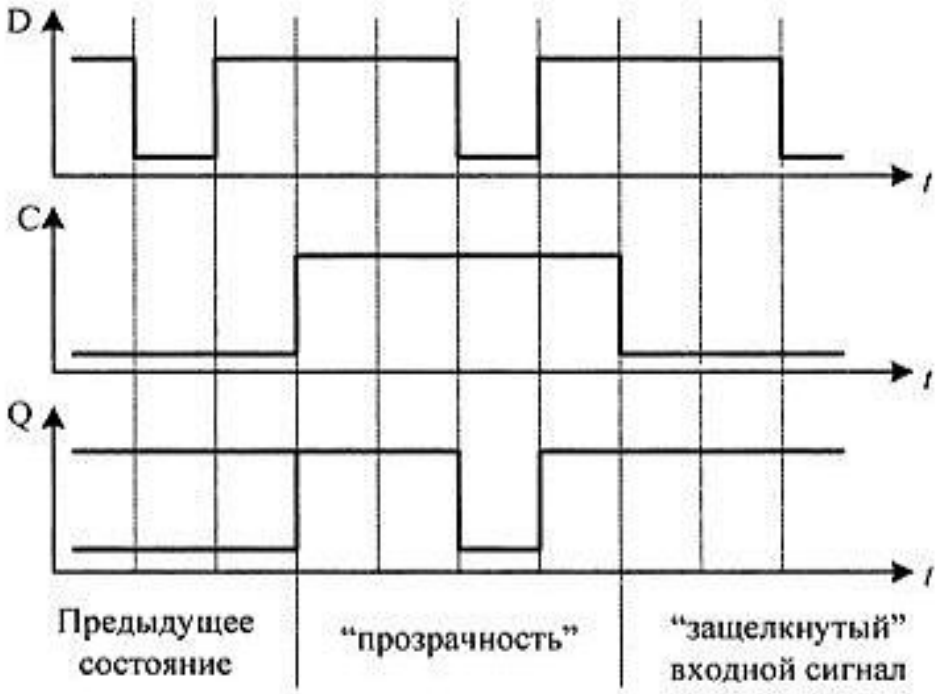


Рис. 8.11. Временные диаграммы сигналов D-триггера

Явление метастабильности

До сих пор предполагалось, что сигнал на входе триггера может принимать только два состояния: логический ноль и логическая единица. Однако синхроимпульс может прийти в любой момент времени, в том числе и в момент смены состояния сигнала на информационном входе триггера. Если синхросигнал попадет точно на момент перехода входным сигналом порогового уровня, то триггер на некоторое время может попасть в неустойчивое метастабильное состояние, при котором напряжение на его выходе будет находиться между уровнем логического нуля и логической единицы. Это может привести к нарушению правильной работы цифрового устройства.

Состояние метастабильности триггера подобно неустойчивому состоянию шарика, находящегося на вершине конического холма. Такая ситуация иллюстрируется рис. 8.12. Обычно триггер не может долго находиться в состоянии метастабильности и быстро возвращается в одно из стабильных состояний. Время нахождения в метастабильном состоянии зависит от уровня шумов схемы и использованной технологии изготовления микросхем.

Временные параметры триггера в момент возникновения состояния метастабильности и выхода из этого состояния приведены на рис. 8.13. Время t_{su} (register setup time — время предустановки регистра) на этом рисунке — это минимальное время перед синхроимпульсом, в течение которого логический уровень сигнала должен оставаться стабильным для того, чтобы избежать

метастабильности выхода триггера. Время t_H (register hold time — время удержания регистра) — это минимально необходимое время удержания стабильного сигнала на входе триггера для того, чтобы избежать метастабильности его выхода. Время состояния метастабильности случайно и зависит от многих параметров. На рис. 8.13 оно обозначено $t_{мет}$.

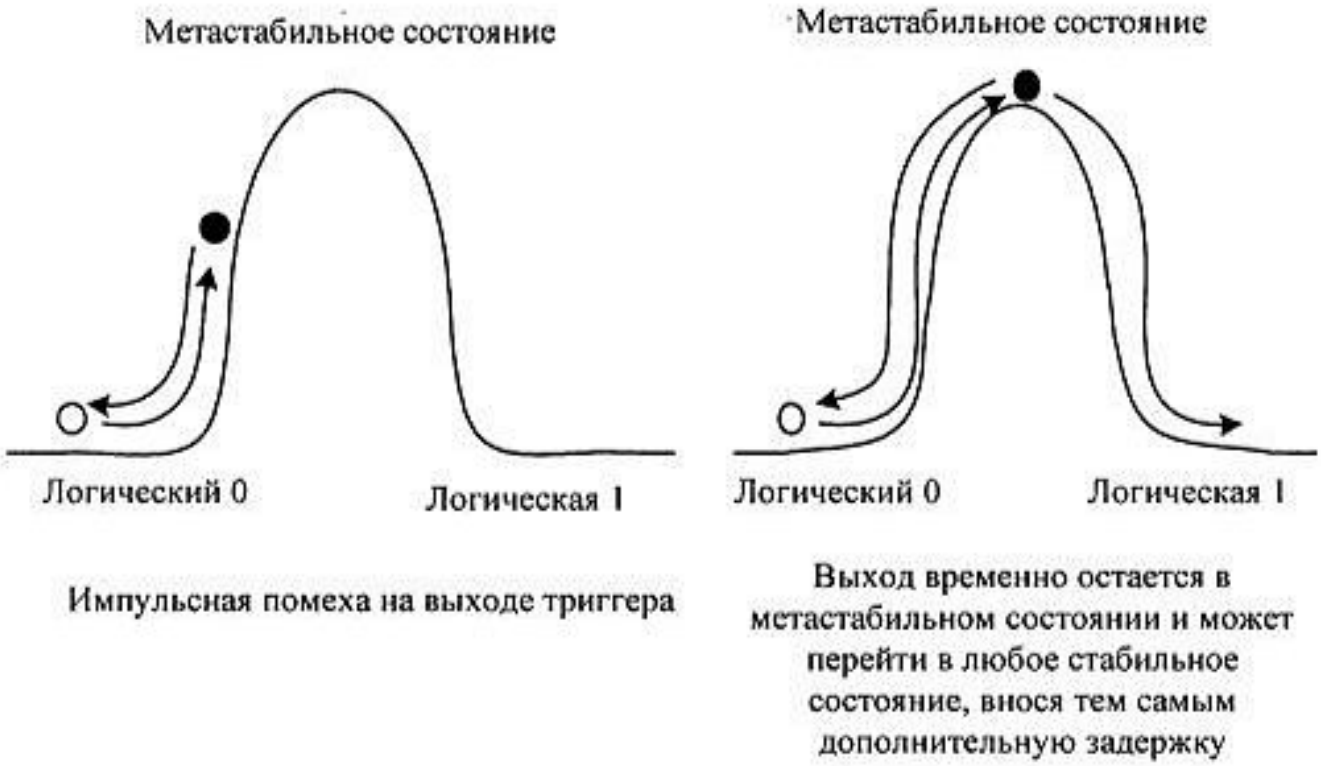


Рис. 8.12. Иллюстрация явления метастабильности

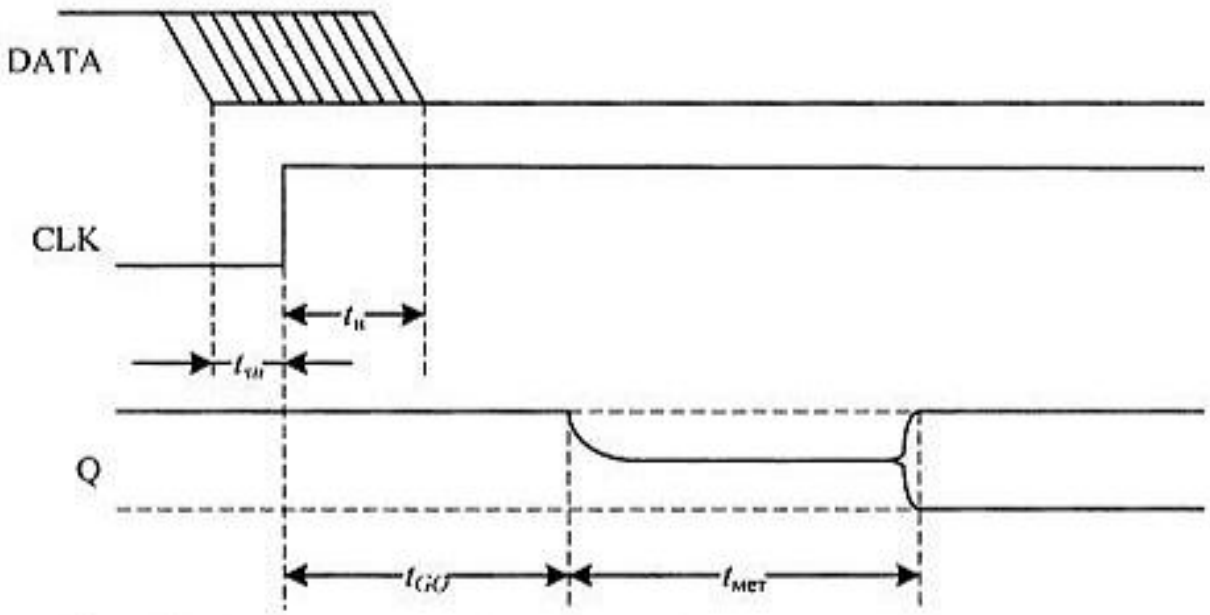


Рис. 8.13. Явление метастабильности. Временные параметры триггера

Вероятность того, что время метастабильности превысит заданную величину, экспоненциально уменьшается с ростом времени, в течение которого выход триггера находится в метастабильном состоянии:

$$P = e^{-\frac{t_{мет}}{\tau}},$$

где τ — это коэффициент, обратно пропорциональный коэффициенту усиления и полосе пропускания элементов, входящих в состав триггера.

Склонность триггеров к метастабильности обычно оценивается величиной, обратной скорости отказов. Это значение выражается как среднее время между отказами. Его можно определить по формуле:

$$MTBF = \frac{1}{скорость\ отказов} = \frac{e^{\frac{t_{мет}}{\tau}}}{t_0 \cdot f_c \cdot f_d},$$

где $t_0 = t_{SU} - t_H$;

f_c — тактовая частота;

f_d — частота, с которой меняются входные данные.

Для оценки этой величины приведем пример таблицы для двух микросхем (табл. 8.4). Последняя строчка этой таблицы эквивалентна времени метастабильности $t_{мет} = 5$ нс.

Таблица 8.4. *Сравнительные характеристики КМОП- и Bi-КМОП-триггеров*

Условия измерения	SN74ACT	SN74ABT
f_c = 33 МГц, f_d = 8 МГц	8400 лет	$8{,}1 \times 10^9$ лет
f_c = 40 МГц, f_d = 10 МГц	92 дня	1400 лет
f_c = 50 МГц, f_d = 12 МГц	—	2 часа

Метастабильное состояние не всегда приводит к неправильной работе цифрового устройства. Если время ожидания устройства после прихода импульса синхронизации достаточно велико, то триггер может успеть перейти в устойчивое состояние, и это будет незаметно. Это означает, что если заранее учитывать время метастабильности $t_{мет}$, то метастабильность никак не скажется на работе остальной цифровой схемы. Если же это время будет неприемлемым для работы схемы, то можно последовательно включить два триггера,

как это показано на рис. 8.14. Такое решение снизит вероятность возникновения метастабильного состояния.

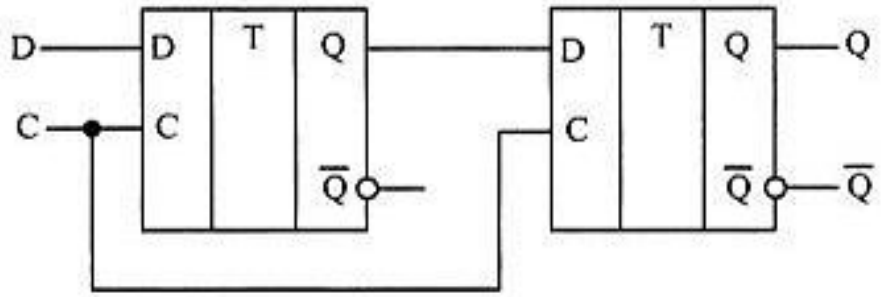


Рис. 8.14. Схема снижения вероятности возникновения метастабильного состояния

Приведем еще один пример. Проведем сравнение MTBF для тех же микросхем, что и в предыдущем примере (табл. 8.5). Время метастабильности $t_{мет}$ = 5 нс для 50 МГц, $t_{мет}$ = 5 нс для 67 МГц, $t_{мет}$ = 5 нс для 80 МГц.

Таблица 8.5. *Сравнительные характеристики КМОП- и Bi-КМОП-триггеров*

Условия измерения	SN74ACT	SN74ABT
f_c = 33 МГц, f_d = 8 МГц	$2{,}62\times10^{28}$ лет	$4{,}77\times10^{47}$ лет
f_c = 40 МГц, f_d = 10 МГц	$3{,}56\times10^{19}$ дня	$2{,}18\times10^{34}$ лет
f_c = 50 МГц, f_d = 12 МГц	$4{,}9\times10^{10}$	1×10^{21} лет
f_c = 67 МГц, f_d = 16 МГц	417 лет	$1{,}28\times10^{9}$ лет
f_c = 80 МГц, f_d = 20 МГц	—	2900 лет

Динамические D-триггеры

Фронт сигнала синхронизации, в отличие от высокого (или низкого) потенциала, не может длиться продолжительное время. В идеале длительность фронта равна нулю, поэтому в триггере, запоминающем входную информацию по фронту, не нужно предъявлять требования к длительности тактового сигнала.

Триггер, запоминающий входную информацию по фронту синхронизирующего сигнала, может быть построен из двух триггеров, работающих по потенциалу, сигнал синхронизации будем подавать на эти триггеры в противофазе. Для формирования такого синхронизирующего сигнала воспользуемся инвертором. Триггеры, соединенные по вышеуказанной схеме, называются двухтактными. Принципиальная схема двухтактного триггера приведена на рис. 8.15.

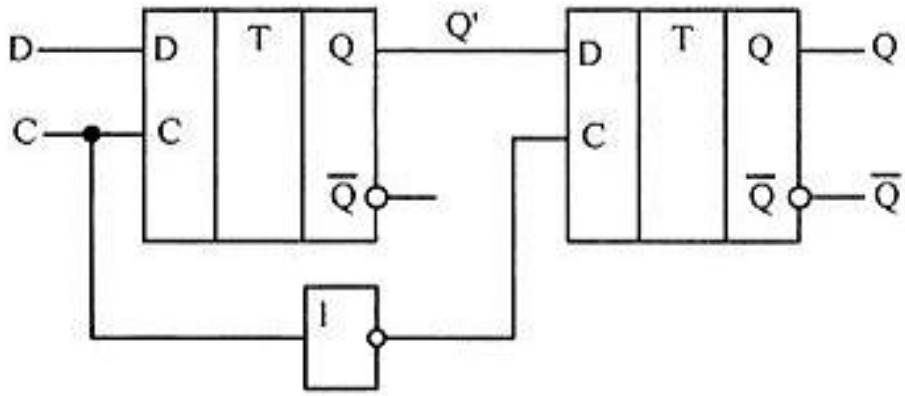


Рис. 8.15. Схема динамического D-триггера

Рассмотрим работу схемы динамического триггера подробнее. Для этого воспользуемся временными диаграммами, приведенными на рис. 8.17. На этих временных диаграммах обозначение Q' соответствует сигналу на выходе первого триггера. Так как на входы синхронизации триггеров тактовый сигнал поступает в противофазе, то когда первый триггер находится в режиме хранения, второй триггер пропускает сигнал на выход схемы. И наоборот, когда первый триггер пропускает сигнал с входа схемы на свой выход, второй триггер находится в режиме хранения.

✧ *Обратите внимание*, что сигнал на выходе всей схемы в целом в режиме хранения не зависит от сигнала на входе D. Если первый триггер пропускает сигнал данных со своего входа на выход Q', то второй триггер в это время находится в режиме хранения и поддерживает на выходе Q предыдущее значение сигнала, т. е. сигнал на выходе схемы тоже не может измениться.

Из приведенного выше анализа видно, что сигнал в схеме, показанной на рис. 8.15, запоминается в момент изменения сигнала на синхронизирующем входе C с единичного потенциала на нулевой.

Динамические D-триггеры выпускаются в виде готовых микросхем или входят в виде готовых блоков в составе больших интегральных схем, таких как базовый матричный кристалл (БМК), или программируемых логических интегральных схем (ПЛИС). Условно-графическое обозначение динамического D-триггера, запоминающего информацию по фронту тактового сигнала, приведено на рис. 8.16.

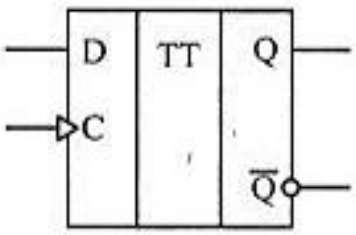


Рис. 8.16. Условно-графическое обозначение D-триггера

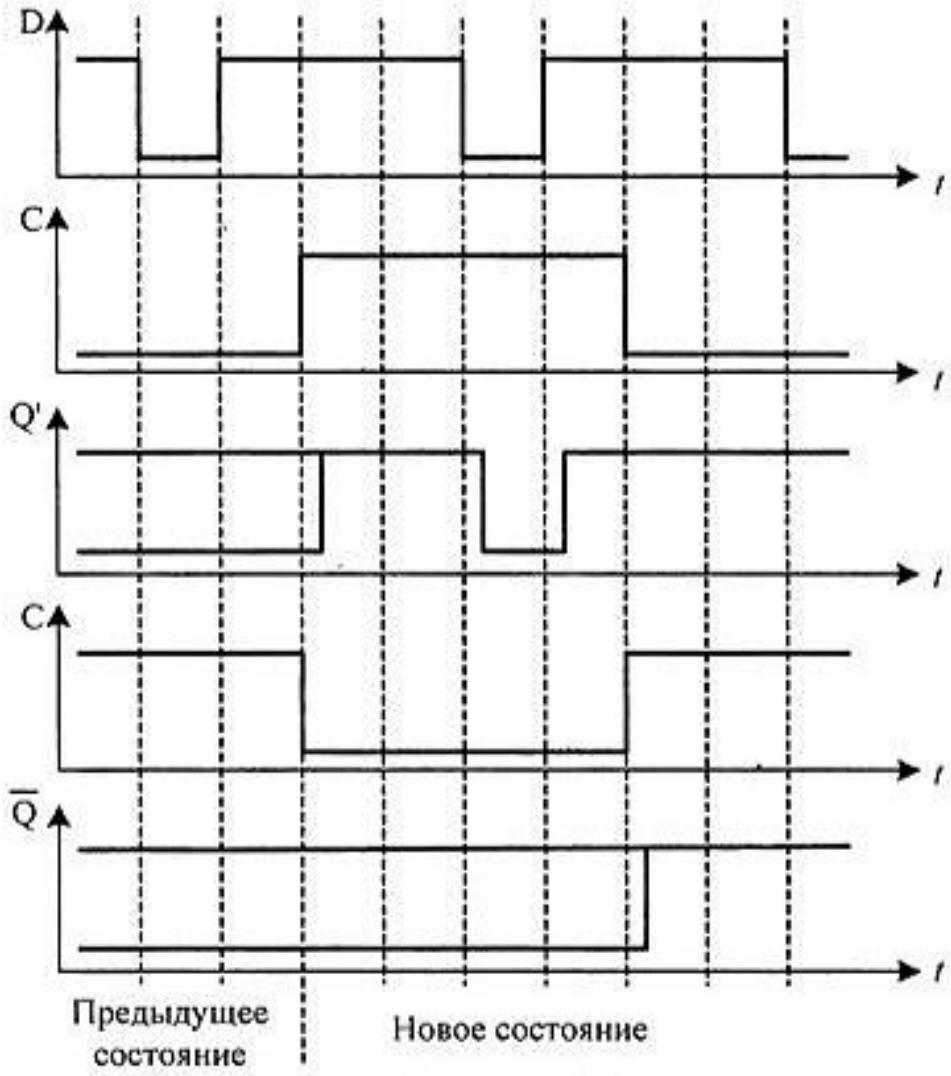


Рис. 8.17. Временные диаграммы D-триггера

То, что триггер запоминает входной сигнал по фронту, отображается на условно-графическом обозначении треугольником, изображенным на выводе входа синхронизации. То, что внутри этого триггера находится два триггера, отображается двойной буквой Т в среднем поле условно-графического изображения. Иногда при изображении динамического входа указывают, по какому фронту триггер изменяет свое состояние. В этом случае используется обозначение входа синхронизации, как это показано на рис. 8.18.

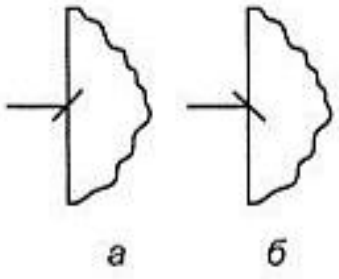


Рис. 8.18. Обозначение динамических входов

На рис. 8.18, *а* обозначен динамический вход, работающий по переднему (нарастающему) фронту сигнала. На рис. 8.18, *б* обозначен динамический вход, работающий по заднему (спадающему) фронту сигнала. Промышленностью выпускаются готовые микросхемы, содержащие динамические триггеры. В качестве примера можно назвать микросхему 1533ТМ2. В этой микросхеме содержится сразу два динамических триггера. Они *изменяют свое состояние по нарастающему фронту сигнала синхронизации.*

Т-триггер

Т-триггер — это счетный триггер, у которого имеется только один вход. После поступления на этот вход одиночного импульса состояние Т-триггера меняется на прямо противоположное. Счетным Т-триггер называется потому, что он как бы подсчитывает количество импульсов, поступивших на его вход. Жаль только, что считать этот триггер умеет только до двух. При поступлении на его вход второго импульса Т-триггер снова сбрасывается в исходное состояние.

Т-триггеры строятся только на базе двухступенчатых триггеров, подобных рассмотренному ранее D-триггеру. Использование двух триггеров позволяет избежать самовозбуждения схемы, т. к. счетные триггеры строятся при помощи схем с обратной связью.

Т-триггер можно синтезировать из любого двухступенчатого триггера. Рассмотрим пример синтеза Т-триггера из динамического D-триггера. Для того чтобы превратить D-триггер в счетный Т-триггер, необходимо ввести цепь обратной связи с инверсного выхода D-триггера на его вход данных D, как это показано на рис. 8.19.

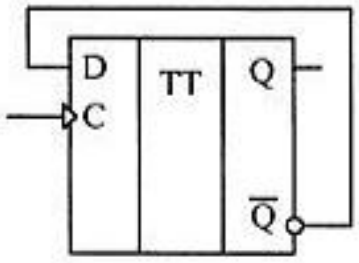


Рис. 8.19. Схема Т-триггера, построенная на основе D-триггера

Временная диаграмма Т-триггера приведена на рис. 8.20. При построении этой временной диаграммы был использован D-триггер, работающий по спадающему фронту синхронизирующего сигнала.

Т-триггеры используются при построении схем различных счетчиков, поэтому в составе программируемых БИС обычно есть готовые модули этих триг-

геров. Условно-графическое обозначение Т-триггера на принципиальных схемах приведено на рис. 8.21.

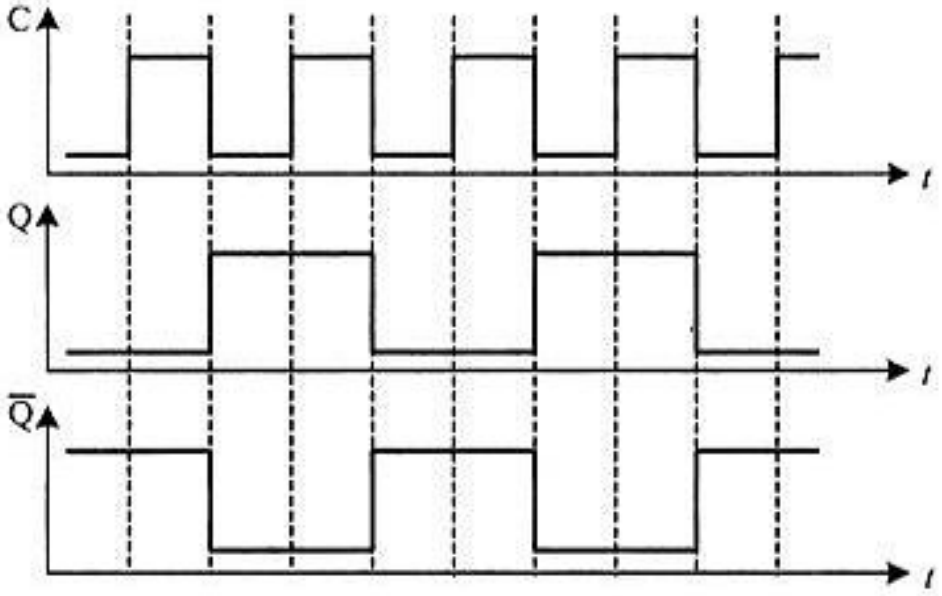


Рис. 8.20. Временные диаграммы Т-триггера

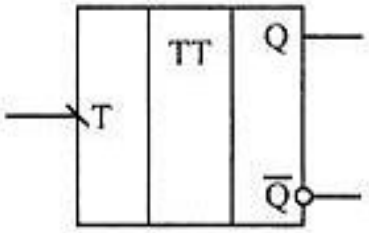


Рис. 8.21. Условно-графическое обозначение Т-триггера

Так как Т-триггеры легко получить из D-триггера или JK-триггера, который будет рассмотрен в следующем разделе, то отдельные микросхемы Т-триггеров промышленностью не выпускаются.

JK-триггер

Прежде чем начать изучение JK-триггера, вспомним принципы работы RS-триггера. Напомним, что в этом триггере есть запрещенные комбинации входных сигналов. Одновременная подача единичных сигналов на вход сброса R и вход установки единицы S RS-триггера запрещена. В JK-триггере этот недостаток устранен.

Таблица истинности JK-триггера практически совпадает с таблицей истинности синхронного RS-триггера. Для того чтобы исключить запрещенное состояние, схема триггера изменена таким образом, что при подаче двух единиц JK-триггер превращается в счетный триггер. Это означает, что при подаче на

тактовый вход C импульсов JK-триггер изменяет свое состояние на противоположное. Таблица истинности JK-триггера приведена в табл. 8.6.

Таблица 8.6. *Таблица истинности JK-триггера*

C	K	J	Q(t)	Q(t+1)	Пояснения
0	x	x	0	0	Режим хранения информации
0	x	x	1	1	
1	0	0	0	0	Режим хранения информации
1	0	0	1	1	
1	0	1	0	1	Режим установки единицы J=1
1	0	1	1	1	
1	1	0	0	0	Режим записи нуля K=1
1	1	0	1	0	
1	1	1	0	1	J=K=1 Счетный режим
1	1	1	1	0	

Один из вариантов внутренней схемы JK-триггера приведен на рис. 8.22.

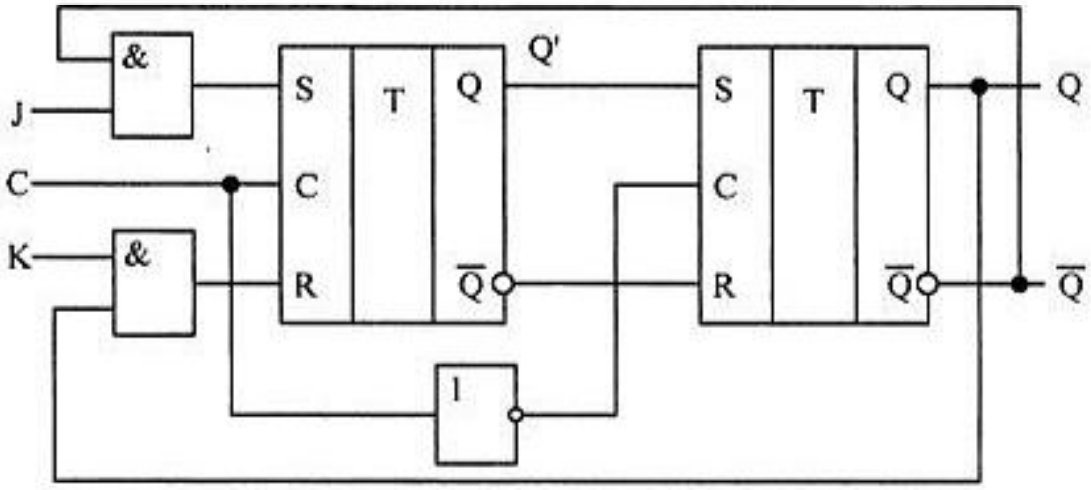


Рис. 8.22. Внутренняя схема JK-триггера

Для реализации счетного режима в схеме, приведенной на рис. 8.22, введена перекрестная обратная связь с выходов второго триггера на входы R и S первого триггера. Благодаря этой обратной связи на входах R и S никогда не может возникнуть запрещенная комбинация.

Приводить временные диаграммы работы JK-триггера не имеет смысла, т. к. они совпадают с приведенными ранее диаграммами RS- и T-триггера. Условно-графическое обозначение JK-триггера изображено на рис. 8.23.

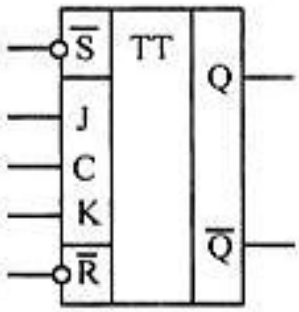


Рис. 8.23. Условно-графическое обозначение JK-триггера

На этом рисунке приведено обозначение типовой цифровой микросхемы JK-триггера, выполненной по ТТЛ-технологии. В промышленно выпускающихся микросхемах обычно, кроме входов JK-триггера, реализуются входы RS-триггера, которые позволяют устанавливать триггер в заранее определенное исходное состояние.

В названиях отечественных микросхем для обозначения JK-триггера присутствуют буквы ТВ. Например, микросхема К1554ТВ9 содержит в одном корпусе два JK-триггера. В качестве примеров иностранных микросхем, содержащих JK-триггеры, можно назвать такие микросхемы, как 74HCT73 или 74ACT109.

Так как JK-триггер является универсальной схемой, рассмотрим несколько примеров использования этого триггера. Начнем с примера использования JK-триггера в качестве обнаружителя коротких импульсов.

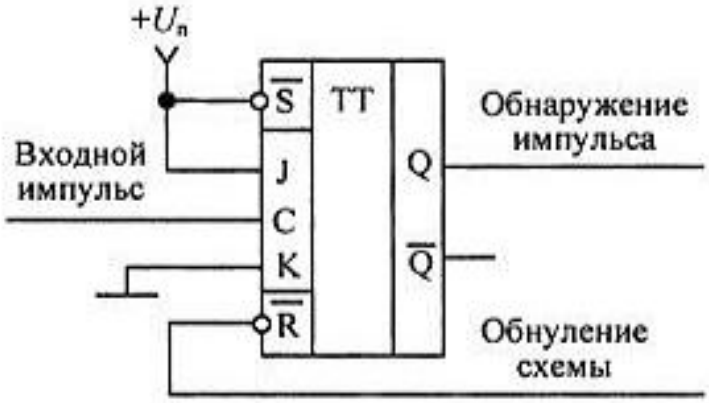


Рис. 8.24. Схема обнаружения короткого импульса

В данной схеме при поступлении на вход C импульса триггер переходит в единичное состояние, которое затем может быть обнаружено последующей схемой (например, микропроцессором). Для того чтобы привести схему в ис-

ходное состояние, необходимо подать на вход R уровень логического нуля. Схема, приведенная на рис. 8.24, очень полезна для формирования сигналов прерывания для микропроцессорных систем, которые не могут мгновенно отреагировать на кратковременные события. После выполнения всех необходимых действий схема приводится в исходное состояние сигналом обнуления схемы.

Теперь рассмотрим пример построения на JK-триггере ждущего мультивибратора. В предыдущей главе уже рассматривались схемы ждущих мультивибраторов, выполненных на логических элементах, однако на JK-триггере можно получить более универсальную схему, позволяющую реализовывать как укорачивающий, так и расширяющий мультивибратор. Один из вариантов подобной схемы мультивибратора приведен на рис. 8.25.

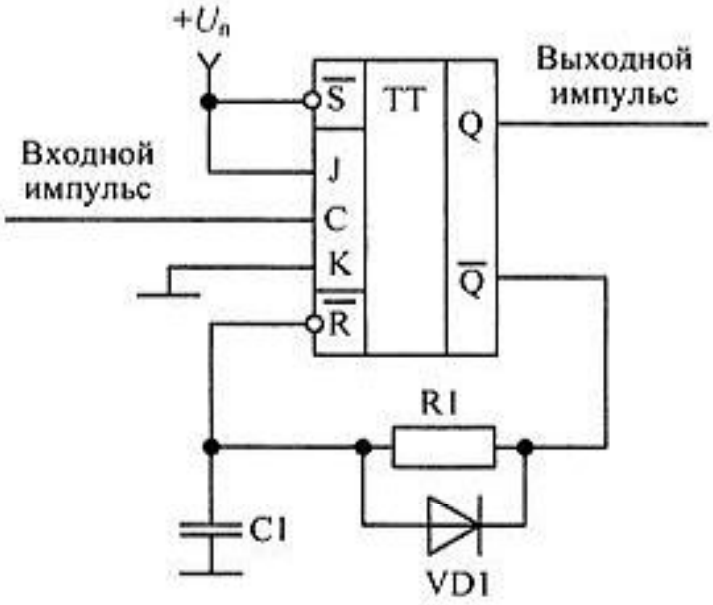


Рис. 8.25. Схема ждущего мультивибратора

Схема мультивибратора работает подобно схеме обнаружения короткого импульса. Длительность выходного импульса определяется постоянной времени RC-цепочки. Диод VD1 предназначен для быстрого восстановления исходного состояния схемы (разряда емкости C). Если быстрое восстановление схемы не требуется, например, когда длительность выходных импульсов гарантированно меньше половины периода следования входных импульсов, то диод VD1 можно исключить из приведенной схемы ждущего мультивибратора.

В качестве последнего примера использования универсального JK-триггера рассмотрим схему счетного T-триггера. Схема счетного триггера, выполненного на универсальном JK-триггере, приведена на рис. 8.26.

В схеме, приведенной на рис. 8.26, для реализации счетного режима работы триггера на входы J и K подаются уровни логической единицы. ✧ *Обратите*

внимание, что схема практически идентична схеме счетного триггера, выполнённого на D-триггере, просто обратная связь выполнена внутри микросхемы.

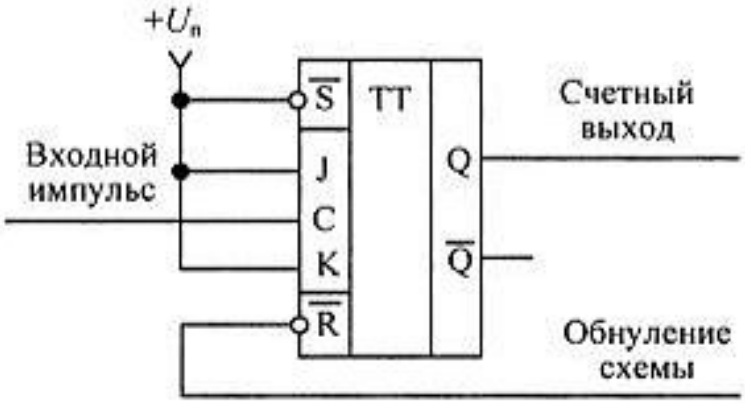


Рис. 8.26. Схема счетного триггера, построенного на JK-триггере

Регистры

Регистром в цифровой технике называется последовательное или параллельное соединение нескольких триггеров. Регистры обычно строятся на основе D-триггеров. При этом для построения регистров могут использоваться как динамические, так и статические D-триггеры (триггеры-защелки). Количество триггеров в составе регистра определяет его разрядность. В качестве отдельных микросхем обычно используются четырех- или восьмиразрядные триггеры.

Параллельные регистры

Название параллельного регистра связано с тем, что входы и выходы всех триггеров в этой схеме используются независимо. Входы синхронизации всех триггеров соединены параллельно. Это приводит к тому, что информация в них записывается одновременно. Параллельный регистр используется для одновременного запоминания многоразрядных двоичных (или недвоичных) чисел (ведь не будем же мы записывать отдельные разряды одного и того же числа в различные моменты времени).

Количество триггеров, входящих в состав параллельного регистра, определяет его разрядность. Принципиальная схема четырехразрядного параллельного регистра приведена на рис. 8.27, а его условно-графическое обозначение — на рис. 8.28. В условно-графическом обозначении параллельного регистра возле каждого входа D указывается степень двоичного разряда, который должен быть запомнен в этом триггере (разряде) регистра. Точно таким же

образом обозначаются и выходы регистра. То, что микросхема является регистром, указывается в центральном поле условно-графического обозначения символами RG.

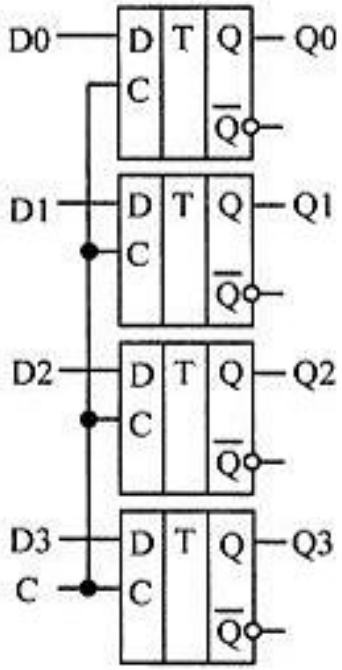


Рис. 8.27. Схема параллельного регистра

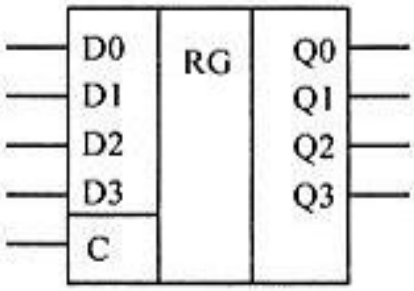


Рис. 8.28. Условно-графическое обозначение параллельного регистра

В приведенном на рис. 8.28 условно-графическом обозначении параллельного регистра инверсные выходы триггеров не показаны. В реально выпускающихся микросхемах параллельных регистров инверсные выходы триггеров обычно не выводятся наружу для экономии количества выводов корпуса.

При записи информации в параллельный регистр все биты (двоичные разряды) должны быть записаны одновременно. Поэтому все тактовые входы триггеров, входящих в состав регистра, объединяются параллельно. В схеме параллельного регистра для уменьшения входного тока вывода синхронизации C на этом входе применяется усилитель. В качестве подобного усилителя обычно используется логический инвертор.

Следует отметить, что назначение разрядов в параллельном регистре является условным. Если по каким-либо причинам (например, с точки зрения разводки печатной платы) удобно изменить нумерацию разрядов параллельного регистра, то это можно свободно осуществить. При перенумерации входов параллельного регистра нужно не забыть точно таким же образом изменить номера его выходов.

Для реализации параллельного регистра можно использовать как триггеры со статическим входом синхронизации, так и с динамическим. В переводной литературе при использовании в параллельном регистре триггеров-защелок

(триггеров со статическим входом синхронизации) этот регистр, в свою очередь, называют регистром-защелкой.

При использовании регистров со статическим входом тактирования удается достичь максимального быстродействия цифровой схемы, однако при этом следует соблюдать осторожность, т. к. при воздействии на вход синхронизации C единичного потенциала логические сигналы с входов такого регистра будут свободно проходить на его выходы. При использовании статических регистров в схеме цифрового устройства обычно применяется двухтактная синхронизация, подобная рассмотренной в главе, посвященной одновибраторам.

Промышленностью выпускаются четырехразрядные и восьмиразрядные микросхемы параллельных регистров. Для построения восьмиразрядных микросхем обычно используются регистры со статическим входом синхронизации. В качестве примера восьмиразрядных параллельных статических регистров можно назвать такие микросхемы, как К580ИР22 и 1533ИР33 (иностранный аналог 74ACT573).

При решении практических задач часто требуется разрядность параллельных регистров, бóльшая восьми. В таком случае можно увеличивать разрядность регистров параллельным соединением готовых микросхем. Принципиальная схема параллельного соединения четырех регистров приведена на рис. 8.29.

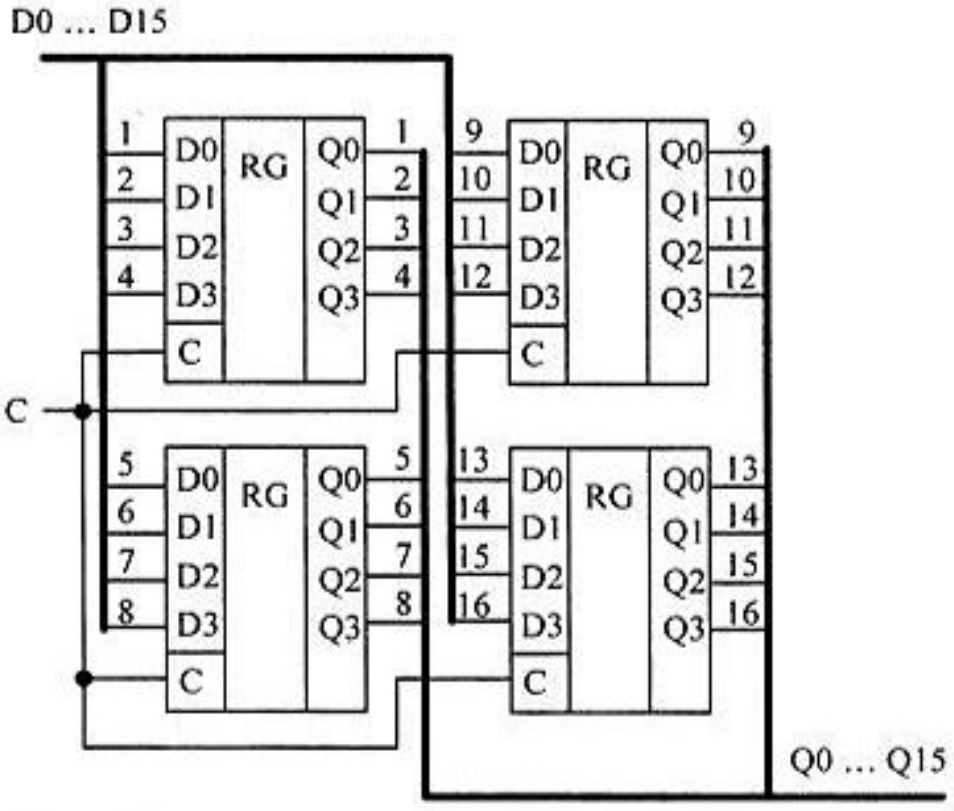


Рис. 8.29. Увеличение разрядности параллельного регистра

В схеме, приведенной на рис. 8.29, реализован 16-разрядный параллельный регистр. При необходимости, входной ток, потребляемый этой схемой по входу тактовой синхронизации, может быть уменьшен при помощи усилителя. В качестве усилителя сигнала тактовой синхронизации можно применить обыкновенный инвертор.

Последовательные регистры

Кроме параллельного соединения триггеров для построения регистров используется последовательное соединение этих элементов. Схемы, в которых триггеры соединены последовательно, называются последовательными регистрами.

Последовательный регистр (регистр сдвига) обычно служит для преобразования последовательного кода в параллельный, и наоборот. Применение последовательного кода связано с необходимостью передачи большого количества двоичной информации по ограниченному количеству соединительных линий. При параллельной передаче разрядов требуется большое количество соединительных проводников. Если двоичные разряды последовательно бит за битом передавать по одному проводнику, то можно значительно сократить размеры соединительных линий на плате (и размеры корпусов микросхем).

В настоящее время этот же принцип применяется для увеличения скорости передачи цифровых сигналов. Это связано с тем, что для малого количества соединительных проводников можно применить усложненную схему согласования и вместо обычного соединительного провода использовать микрополосковую линию передачи сигналов.

При обсуждении схемы преобразования параллельного кода в последовательный часто возникает вопрос — а почему бы не воспользоваться для этого преобразования мультиплексором? В качестве ответа можно привести то, что время переключения мультиплексора зависит от двоичного кода на адресном входе мультиплексора. Различные разряды этого кода, в свою очередь, могут поступать на вход мультиплексора тоже неодновременно.

Принципиальная схема последовательного регистра, собранного на основе двухступенчатых D-триггеров, и позволяющего осуществить преобразование последовательного кода в параллельный, приведена на рис. 8.30.

В этом регистре выход первого триггера соединен с входом второго, выход второго триггера — с входом третьего и т. д. Условно-графическое изображение рассмотренного на рис. 8.30 последовательного регистра приведено на рис. 8.31.

Входы синхронизации отдельных триггеров в схеме последовательного регистра, как и в схеме параллельного регистра, объединяются. Это обеспечивает

одновременность смены внутреннего состояния всех триггеров, входящих в состав последовательного регистра.

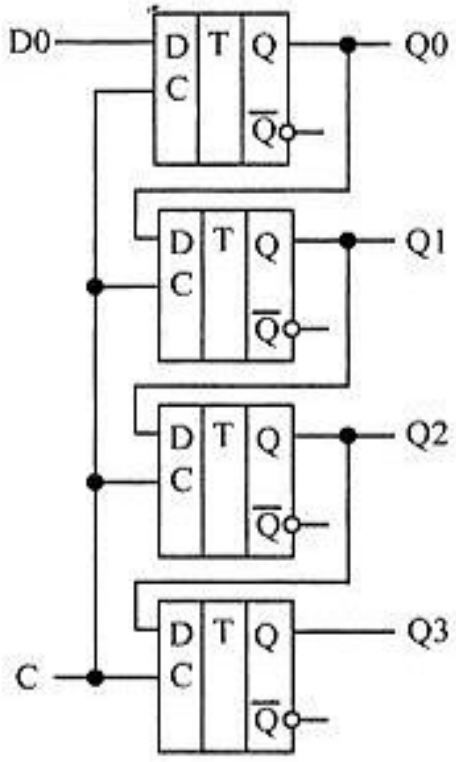


Рис. 8.30. Схема последовательного регистра

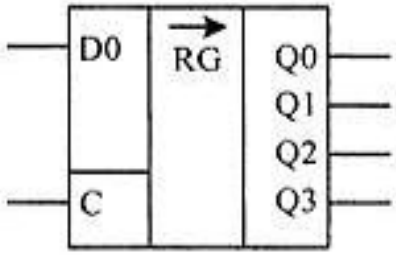


Рис. 8.31. Условно-графическое обозначение последовательного регистра на принципиальных схемах

Преобразование последовательного кода в параллельный производится следующим образом. Отдельные биты двоичной информации последовательно подаются на вход D0. Каждый бит сопровождается отдельным тактовым импульсом, который поступает на вход синхронизации C. После поступления первого тактового импульса логический уровень, присутствующий на входе D0, запоминается в первом триггере и поступает на его выход, а т. к. он соединен с входом второго триггера, то теперь этот потенциал будет присутствовать на входе второго триггера.

После поступления второго тактового импульса логический уровень, присутствующий на входе второго триггера, запоминается в нем и поступает на его выход, а т. к. он соединен с входом третьего триггера, то теперь этот потенциал подан на вход третьего триггера. Одновременно в первом триггере запоминается второй бит входного последовательного кода.

После поступления четвертого тактового импульса в триггерах регистра будут записаны уровни бит, которые последовательно присутствовали на входе D0 в моменты поступления предыдущих импульсов тактовой синхронизации. Теперь этими логическими уровнями можно воспользоваться, например, для отображения на индикаторах.

Пусть на вход последовательного регистра поступает сигнал, временная диаграмма которого изображена на рис. 8.32, тогда состояние выходов этого регистра будет последовательно принимать значения, записанные в табл. 8.7.

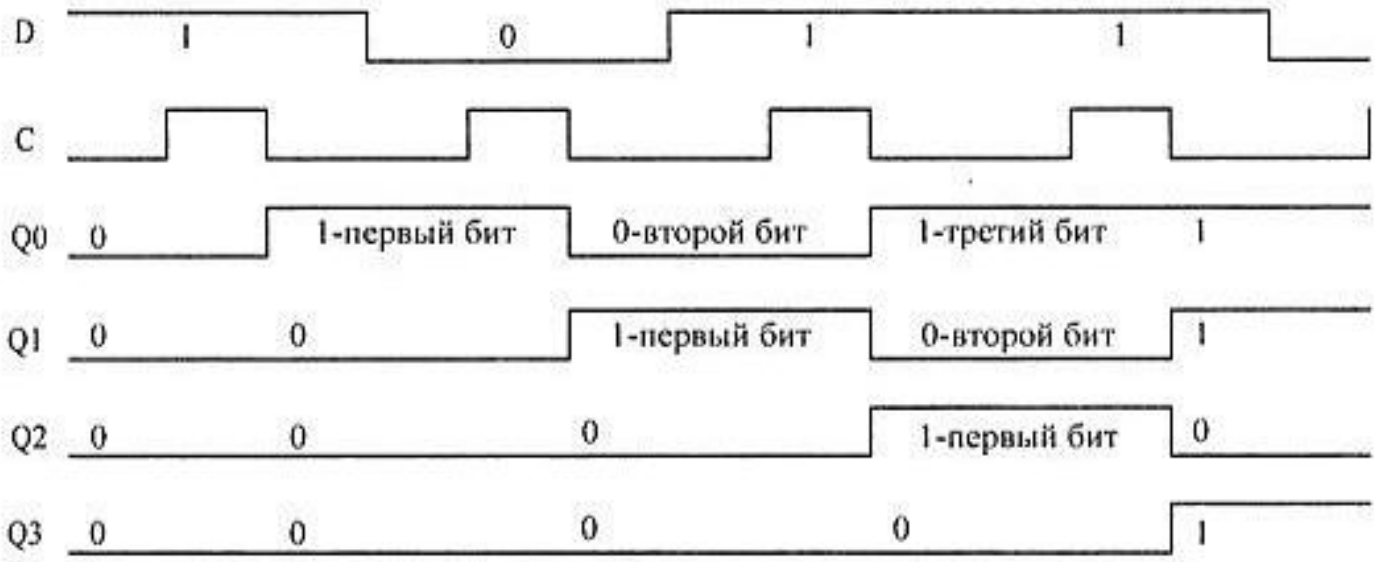


Рис. 8.32. Временная диаграмма работы сдвигового регистра

На рис. 8.32 вместе с логическими уровнями записываются значения бит, которые передаются по соединительной линии или присутствуют на выходах сдвигового регистра.

***Таблица 8.7.** Состояние выходов сдвигового регистра*

№ такта / Выход	1	2	3	1
Q0	1	0	1	1
Q1	X	1	0	1
Q2	X	X	1	0
Q3	X	X	X	1

В табл. 8.7 символом X обозначено неопределенное значение, возникающее в триггерах регистра при включении питания или оставшееся от предыдущего цикла работы схемы.

Универсальные регистры

Регистры сдвига выполняют обычно как универсальные последовательно-параллельные микросхемы. Это связано с необходимостью записи в регистр параллельного двоичного кода при преобразовании параллельного кода в последовательный.

Переключение регистра из параллельного режима работы в последовательный и наоборот осуществляется при помощи мультиплексора (коммутатора). Использование коммутатора позволяет подключать входы триггеров регистра либо к внешним выводам микросхемы, либо к выходу предыдущего триггера.

Напомним, что двухвходовый мультиплексор можно реализовать при помощи логических элементов "2И-2ИЛИ". Элементы "И" при этом работают в качестве электронных ключей, а элементы "ИЛИ" объединяют их выходы.

Схема универсального последовательно-параллельного регистра с использованием коммутаторов на логических элементах "2И-2ИЛИ" приведена на рис. 8.33. В этой схеме для переключения регистра из последовательного режима работы в параллельный используется вход V. Подача на этот вход единичного потенциала превращает схему в параллельный регистр. При этом на входы ключей, подключенных к информационным входам D, подается единичный потенциал. Это приводит к тому, что сигналы с входов параллельной записи данных поступают на входы логических элементов "ИЛИ", а на входы ключей, подключенных к выходам предыдущих триггеров, подаются нулевые

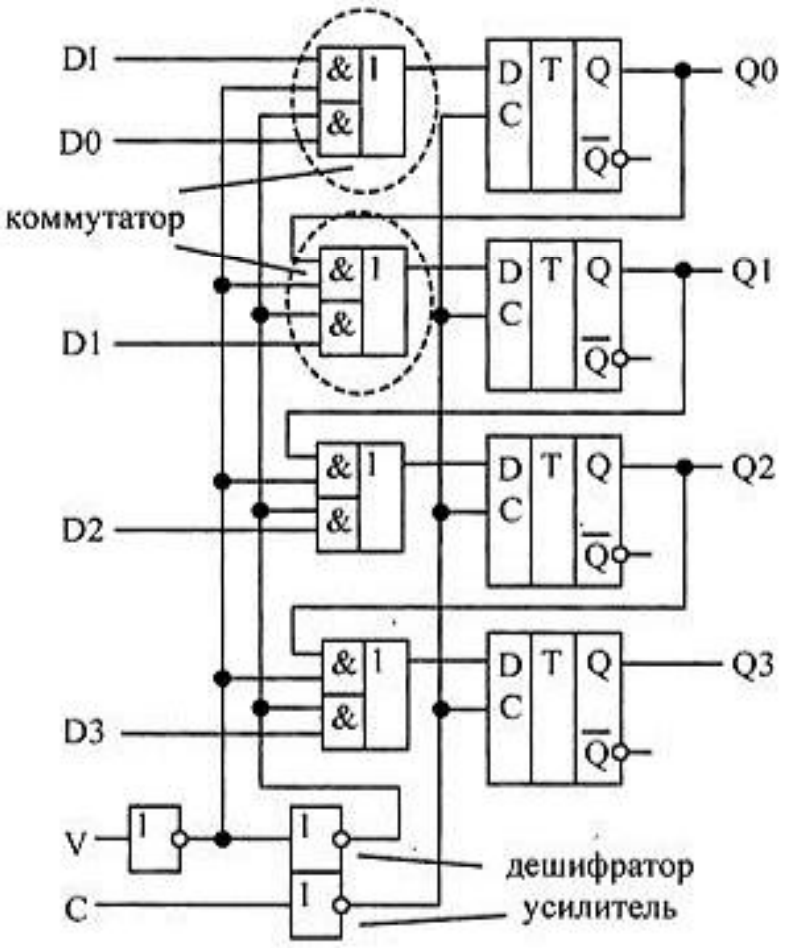


Рис. 8.33. Принципиальная схема универсального последовательно-параллельного регистра

потенциалы. То есть на выходах этих ключей будут присутствовать нулевые потенциалы, и они не будут мешать работе.

Подача на вход V нулевого потенциала приводит к отключению входов параллельных данных от входов триггеров. Сигналы с выхода предыдущего триггера свободно проходят через верхние логические элементы "И" на вход последующего триггера, т. к. на его второй вход подается единичный потенциал.

Инверторы на входах V и C использованы для усиления входного сигнала по току. В результате применения такого решения входной ток микросхемы будет равен не суммарному току четырех логических элементов "И", а входному току инвертора.

Условно-графическое изображение универсального регистра, принципиальная схема которого показана на рис. 8.33, приведено на рис. 8.34.

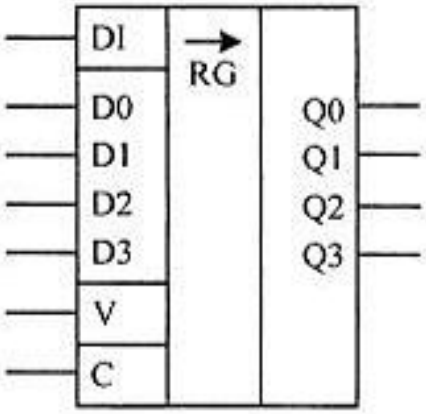


Рис. 8.34. Условно-графическое обозначение универсального регистра на принципиальных схемах

Вход последовательного ввода данных на этом рисунке обозначен как DI и отделен от других групп входов чертой. Точно так же выделены в отдельные группы и входы управления V и синхронизации C.

Счетчики

Счетчики используются для построения таймеров или для выборки инструкций из ПЗУ в микропроцессорах. Они могут использоваться как делители частоты в управляемых генераторах частоты (синтезаторах). При использовании в цепи ФАПЧ счетчики могут быть использованы для умножения частоты как в синтезаторах опорных частот, так и в микропроцессорах. С их помощью можно формировать импульсы строго определенной длительности. Особенности работы цепей фазовой автоподстройки частоты (ФАПЧ) будут рассмотрены подробно в *главе 12* данной книги.

Двоичные суммирующие асинхронные счетчики

Простейший вид счетчика — двоичный, может быть построен на основе Т-триггера. Для реализации Т-триггера в двоичном счетчике воспользуемся универсальным D-триггером с обратной связью, как это мы рассматривали в предыдущих разделах и как это показано на рис. 8.35.

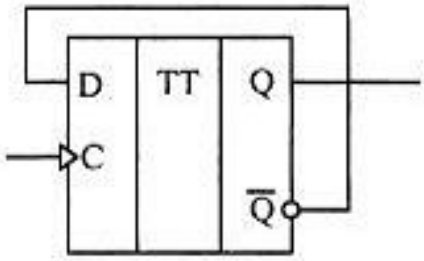


Рис. 8.35. Реализация счетного Т-триггера на универсальном D-триггере

Так как схема Т-триггера, как мы уже рассматривали ранее, при поступлении на вход импульсов меняет свое состояние на противоположное, то ее можно рассматривать как счетчик, считающий до двух.

Обычно требуется подсчитать большее количество импульсов. В этом случае можно использовать выходной сигнал первого счетного триггера как входной сигнал для следующего триггера, т. е. соединить простейшие двоичные счетчики последовательно друг за другом. Так можно построить любой счетчик, считающий до максимального числа, которое можно определить по следующей формуле.

$$M = 2^N,$$

где N — число триггеров, входящих в счетчик.

Схема счетчика, позволяющего посчитать любое количество импульсов, меньшее шестнадцати, приведена на рис. 8.36. Количество поступивших на вход импульсов можно узнать, подключившись к выходам счетчика Q0 ... Q3. Эти сигналы можно рассматривать как двоичное число, которое можно в дальнейшем обрабатывать по правилам двоичной арифметики.

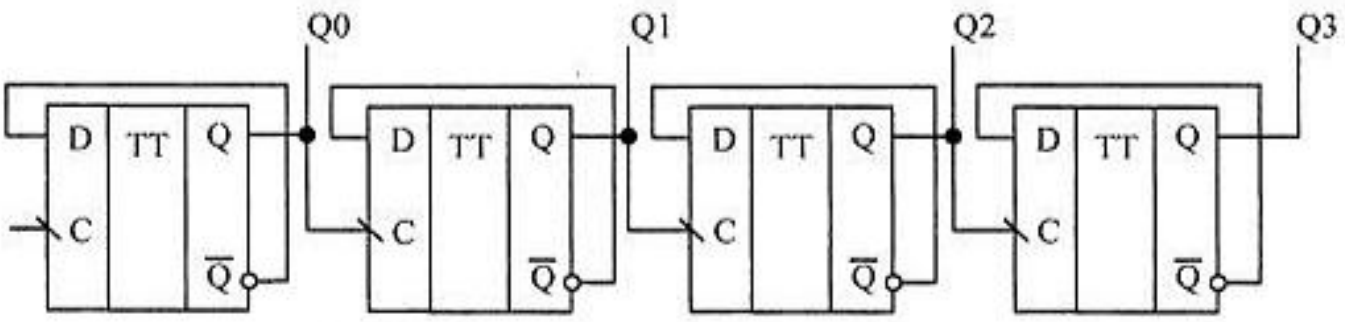


Рис. 8.36. Схема четырехразрядного счетчика, построенного на универсальных D-триггерах

Для иллюстрации работы двоичного счетчика воспользуемся временными диаграммами сигналов на входе и выходах этой схемы, приведенной на рис. 8.36. Эти временные диаграммы показаны на рис. 8.37.

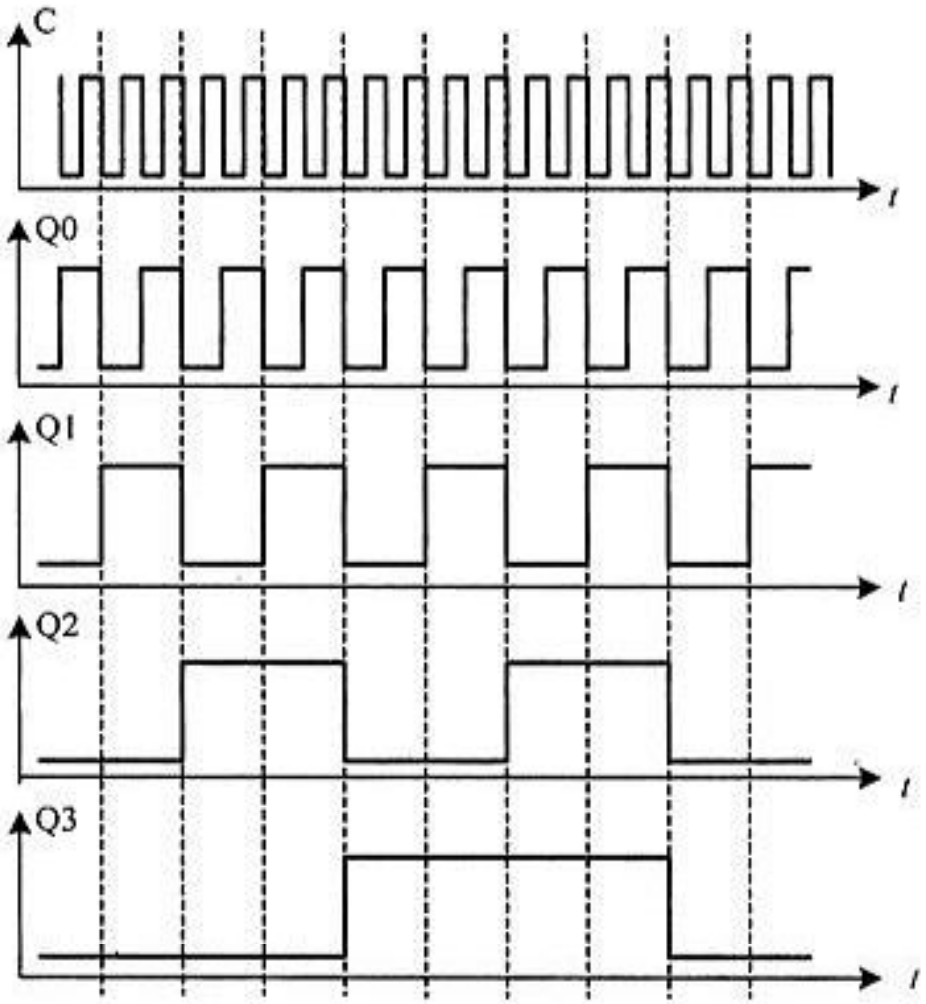


Рис. 8.37. Временная диаграмма четырехразрядного счетчика

Проанализируем приведенные временные диаграммы. Пусть первоначальное состояние всех триггеров анализируемого счетчика будет нулевым. Это состояние видно в начальной области временных диаграмм. Запишем его в нулевую строку табл. 8.8. После поступления на вход счетчика тактового импульса (который воспринимается по заднему фронту) первый триггер изменяет свое состояние на противоположное, т. е. теперь в этом триггере будет записана единица.

Запишем новое состояние выходов счетчика во вторую строку той же самой таблицы. Так как по приходу первого импульса изменилось состояние первого триггера, то этот триггер содержит младший разряд двоичного числа (единицы). В таблице поставим его значение на самом правом месте, как это принято в математике при записи любых многоразрядных чисел. Здесь мы впервые сталкиваемся с противоречием правил записи чисел в математике и правил распространения сигналов на принципиальных схемах.

Подадим на вход счетчика еще один тактовый импульс. Значение первого триггера снова изменится на прямо противоположное. На этот раз на выходе первого триггера, а значит, и на входе второго триггера сформируется спадающий фронт. Это означает, что второй триггер тоже изменит свое состояние на противоположное. Это отчетливо видно на временных диаграммах, приведенных на рис. 8.32. Запишем новое состояние выходов счетчика в третью строку табл. 8.8. В этой строке таблицы образовалось двоичное число 2. Оно совпадает с номером входного импульса.

Продолжая анализировать временную диаграмму, можно определить, что на выходах приведенной схемы счетчика последовательно появляются цифры от 0 до 15. Эти цифры записаны в двоичном виде. При поступлении на счетный вход счетчика очередного импульса содержимое его триггеров увеличивается на 1. Именно поэтому такие счетчики получили название суммирующих двоичных счетчиков.

Таблица 8.8. *Изменение уровней на выходе суммирующего двоичного счетчика при поступлении на его вход импульсов*

Номер входного импульса	Q3	Q2	Q1	Q0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

Условно-графическое обозначение суммирующего двоичного счетчика на принципиальных схемах приведено на рис. 8.38. В двоичных счетчиках для обнуления микросхемы обычно предусматривают вход R, который позволяет записать во все триггеры счетчика нулевое значение. Это состояние называют исходным состоянием счетчика.

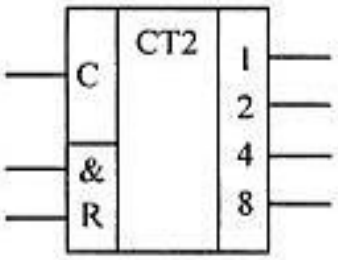


Рис. 8.38. Условно-графическое обозначение четырехразрядного двоичного счетчика

Промышленностью выпускаются микросхемы асинхронных двоичных счетчиков. Классическим примером такого счетчика является микросхема КР555ИЕ5. Подобные схемы существуют и внутри САПР программируемых логических интегральных схем в виде готовых макрофункций.

Двоичные вычитающие асинхронные счетчики

Счетчики при поступлении на счетный вход импульсов могут не только увеличивать свое внутреннее состояние на единицу, но и уменьшать его. Такие счетчики получили название вычитающих счетчиков. Для реализации вычитающего счетчика достаточно, чтобы Т-триггер изменял свое состояние по нарастающему фронту входного сигнала.

Изменить рабочий фронт входного сигнала можно инвертированием этого сигнала. В схеме, приведенной на рис. 8.39, для реализации вычитающего счетчика сигнал на входы последующих триггеров подается с инверсных выходов предыдущих триггеров. Для инвертирования сигнала на входе всей схемы применен отдельный инвертор.

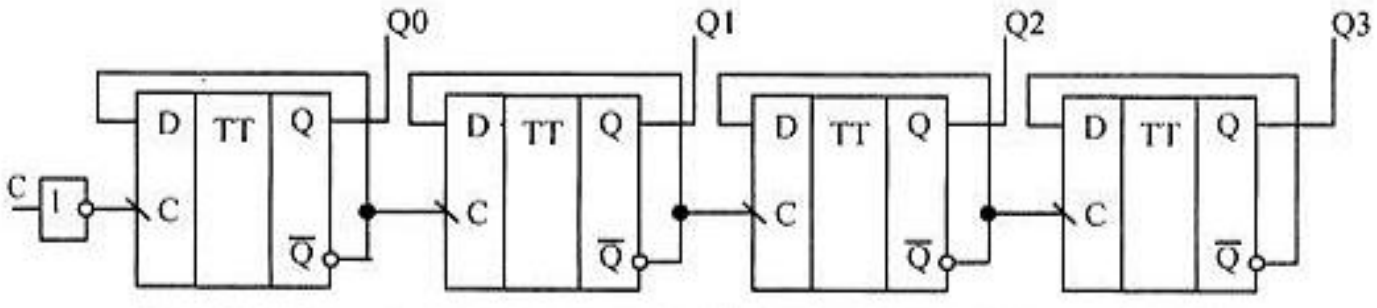


Рис. 8.39. Схема четырехразрядного двоичного вычитающего счетчика

Временные диаграммы сигналов на входе и выходах вычитающего счетчика приведены на рис. 8.40. Исходное состояние триггеров счетчика, как и в предыдущем случае, нулевое. По этим временным диаграммам видно, что при поступлении на вход счетчика первого же импульса на неинвертирующих выходах триггеров счетчика появляется максимально возможное для четырехразрядного счетчика число 15_{10}.

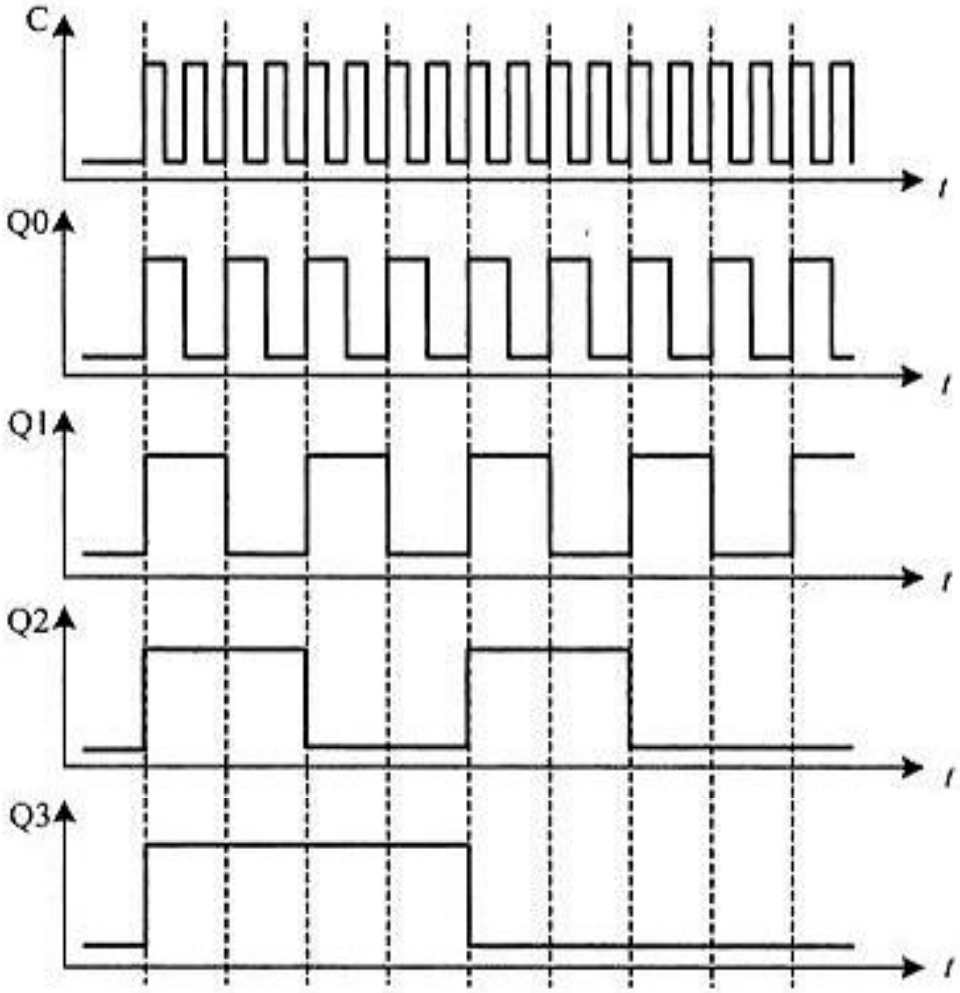


Рис. 8.40. Временные диаграммы четырехразрядного вычитающего счетчика

Это вызвано тем, что при поступлении нарастающего фронта тактового импульса первый триггер переходит в единичное состояние. В результате на его выходе Q0 тоже формируется нарастающий фронт. Он поступает на вход второго триггера, что приводит к записи единицы и в этот триггер. Точно такая же ситуация складывается со всеми триггерами счетчика, т. е. все триггеры перейдут в единичное состояние. Для четырехразрядного счетчика это и будет число 15_{10}. Запишем новое состояние вычитающего счетчика во вторую строку табл. 8.9.

Следующий тактовый импульс приведет к изменению состояния только первого триггера, т. к. при этом на его выходе Q0 сформируется задний фронт

сигнала. Запишем это состояние в третью строку табл. 8.9. ✧ *Обратите внимание*, что при поступлении каждого последующего импульса содержимое счетчика, построенного по анализируемой схеме, уменьшается на единицу. Этот процесс продолжается до тех пор, пока состояние счетчика не станет вновь равно 0. При поступлении следующих тактовых импульсов процесс повторяется снова.

Все возможные состояния логических сигналов на выходах вычитающего счетчика при поступлении на счетный вход схемы тактовых импульсов приведены в табл. 8.9. Эта таблица фактически повторяет временные диаграммы, приведенные на рис. 8.40, однако она более наглядно показывает физические основы работы счетчика, т. к. мы при работе с числами привыкли иметь дело с цифрами, а не с напряжениями, тем более в зависимости от времени.

Таблица 8.9. *Изменение уровней на выходе вычитающего счетчика при поступлении на его вход импульсов*

Номер входного импульса	Q3	Q2	Q1	Q0	Состояние счетчика
0	0	0	0	0	0
1	1	1	1	1	15
2	1	1	1	0	14
3	1	1	0	1	13
4	1	1	0	0	12
5	1	0	1	1	11
6	1	0	1	0	10
7	1	0	0	1	9
8	1	0	0	0	8
9	0	1	1	1	7
10	0	1	1	0	6
11	0	1	0	1	5
12	0	1	0	0	4
13	0	0	1	1	3
14	0	0	1	0	2
15	0	0	0	1	1

Для тех, кто привык работать с реально выпускаемыми микросхемами, следует обратить внимание, что в примере были использованы D-триггеры,

работающие по спадающему фронту. Микросхемы, выпускаемые промышленностью, например, 1533ТМ2 (два D-триггера в одном корпусе) работают по нарастающему фронту, поэтому схемы суммирующего и вычитающего счетчика, реализованные на этих микросхемах, поменяются местами.

Недвоичные счетчики с обратной связью

Если проанализировать временные диаграммы сигналов на выходах двоичного счетчика, приведенные на рис. 8.37 и 8.40, то можно увидеть, что частота сигналов на выходах двоичного счетчика будет уменьшаться в два раза по отношению к предыдущему выходу. Это позволяет использовать двоичные счетчики в качестве цифровых делителей частоты входного сигнала. Цифровые делители частоты используются в устройствах формирования высокостабильных генераторов частоты (синтезаторов частот или схем прямого цифрового синтеза).

Сформированные частоты могут быть использованы либо для синхронизации различных цифровых устройств (в том числе и микропроцессоров), либо в качестве высокостабильных генераторов опорных частот в радиоприемных и радиопередающих устройствах.

При применении цифровых счетчиков в качестве устройств формирования опорных частот часто требуется обеспечить коэффициент деления частоты, отличающийся от степени числа 2. В этом случае требуется цифровой счетчик с недвоичным коэффициентом счета.

Еще одна ситуация, когда могут потребоваться недвоичные счетчики, возникает при отображении информации, записанной в цифровом счетчике. Человек, который работает с электронной техникой, привык работать с десятичной системой счисления, поэтому возникает необходимость отображать хранящееся в счетчике число в десятичном виде. Это сделать намного проще, если и счет входных импульсов вести сразу в десятичном или двоично-десятичном коде, иначе для индикации потребуется перекодировать информацию из двоичного кода в двоично-десятичный. Такая ситуация встречается при построении измерителей длительности импульсов или частотомеров.

Построить недвоичный счетчик можно из двоичного за счет исключения лишних комбинаций нулей и единиц. Эта операция может быть осуществлена при помощи обратной связи, заведенной на вход обнуления состояния счетчика R. Для реализации недвоичного счетчика при помощи дешифратора определяется внутреннее состояние счетчика, соответствующее требуемому коэффициенту счета. Сигнал с выхода этого дешифратора обнуляет содержимое двоичного счетчика.

✧ *Обратите внимание*, что эти рассуждения справедливы для суммирующего двоичного счетчика. При использовании вычитающего счетчика необходимо декодировать число, равное отрицательному значению коэффициента счета. Такой счетчик обычно используется в качестве делителя частоты.

$$K_д = N_{max} - K_{сч},$$

где $K_д$ — число, на которое настраивается дешифратор;

N_{max} — максимальный коэффициент счета двоичного цифрового счетчика;

$K_{сч}$ — требуемый коэффициент счета недвоичного счетчика.

В качестве примера описанной идеи реализации недвоичного счетчика, построенного на суммирующем двоичном счетчике, рассмотрим схему двоично-десятичного счетчика, приведенную на рис. 8.41.

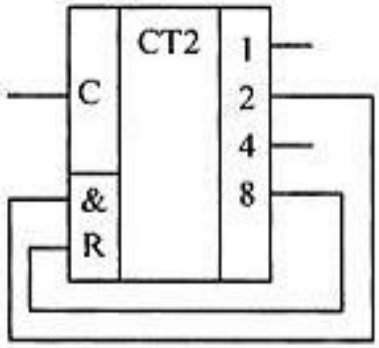


Рис. 8.41. Схема двоично-десятичного счетчика

В рассматриваемой схеме дешифратор построен на двухвходовом логическом элементе "2И", входящем в состав микросхемы суммирующего двоичного счетчика К155ИЕ5. Дешифратор декодирует число 10_{10}, соответствующее коду 1010_2 в двоичной системе счисления.

В соответствии с принципами построения схем по произвольной таблице истинности, для построения дешифратора требуется еще два инвертора, подключенных к выходам 1 и 4, однако после сброса счетчика числа, бóльшие 10_{10}, никогда не смогут появиться на выходах микросхемы счетчика. В результате учета этого фактора схема дешифратора числа 10_{10} упрощается, и вместо четырехвходового логического элемента "4И" можно обойтись двухвходовым. Инверторы в таком дешифраторе оказываются лишними.

Приведем в качестве еще одного примера реализации недвоичного счетчика схему делителя частоты на 1000. При разработке делителя частоты, прежде всего, определим, сколько потребуется микросхем двоичных счетчиков. Для этого определим степень числа 2, при которой число $M = 2^n$ будет больше требуемого числа 1000.

Получаем число десять. При возведении основания системы счисления 2 в 10 степень получится число 1024. Оно, естественно, больше числа 1000, а значит, при использовании для построения делителя частоты счетных триггеров достаточно будет десяти триггеров, однако обычно для построения делителей частоты используют готовые двоичные счетчики, поэтому определим необходимое количество микросхем двоичных счетчиков. При использовании четырехразрядных двоичных счетчиков достаточно будет трех микросхем, т. к. в трех микросхемах будет 3 × 4= 12 триггеров, что заведомо больше минимального числа триггеров.

Следующим этапом построения делителя частоты будет перевод коэффициента деления 1000 в двоичное представление. Десятичное число 1000_{10} в двоичном виде будет выглядеть как 0011 1110 1000_2. В этом числе шесть единиц, поэтому для построения дешифратора будет достаточно шестивходового логического элемента "6И", однако такие микросхемы не выпускаются, поэтому воспользуемся доступной микросхемой "8И-НЕ". Неиспользуемые входы этой микросхемы подключим к питанию. Теперь они мешать работе схемы делителя частоты не будут. Ненужную нам инверсию сигнала скомпенсируем дополнительным инвертором. Получившаяся схема делителя частоты с коэффициентом деления 1000 приведена на рис. 8.42.

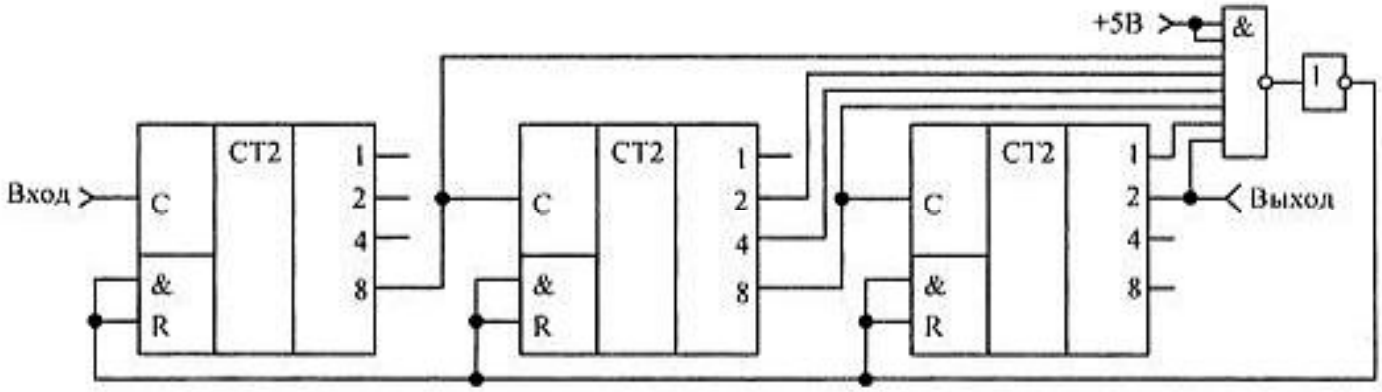


Рис. 8.42. Схема делителя на 1000, построенного на основе трех двоичных счетчиков

При использовании счетчиков в составе синтезаторов частот может потребоваться формирование определенного диапазона частот. В этом случае делитель, построенный на недвоичном счетчике, должен обладать возможностью изменения коэффициента деления.

Такие делители частоты получили название делителей с переменным коэффициентом деления (ДПКД). При использовании обратной связи для реализации ДПКД потребуется полный дешифратор всех возможных состояний двоичного счетчика и переключатели его выходов на вход сброса счетчика. Схема делителя частоты при этом получается сложной, а управление таким делителем неудобным.

Пример принципиальной схемы трехразрядного делителя с переменным коэффициентом деления (ДПКД), построенного на трех микросхемах десятичных счетчиков ($10^3 = 1000$), приведен на рис. 8.43. Этот делитель при совместном применении с кварцевым генератором тактовой частоты может использоваться в качестве синтезатора дискретной сетки частот.

✧ *Обратите внимание*, что для удобного управления таким синтезатором частоты использованы десятичные счетчики. Применение десятичных счетчиков позволяет выставлять необходимую частоту выходного сигнала непосредственно в десятичном виде. Входной сигнал для второго каскада делителя частоты D2 снимается со старшего разряда предыдущего счетчика. Точно так же соединены микросхемы D2 и D3. Таким образом формируется многоразрядный десятичный делитель частоты.

В приведенной на рис. 8.43 схеме для дешифрации внутреннего состояния счетчика применяется составной дешифратор. Он состоит из микросхем D4 ... D6 и объединяющего их логического элемента "3И-НЕ" D7. Инвертор D8 компенсирует инверсию логического элемента D7. Конкретное значение коэффициента деления выставляется десятичными переключателями S1 ... S3. Десятичный счетчик досчитывает до выставленного десятичными переключателями S1 ... S3 значения и сбрасывается сигналом с выхода микросхемы D7 в исходное нулевое состояние. Импульс сброса десятичного счетчика и является выходной частотой ДПКД.

Значение формируемой делителем ДПКД частоты можно нанести на корпусе прибора над клювиками переключателей или отображать набираемую частоту на десятичных индикаторах. Работу с индикаторами мы подробно рассмотрим в следующей главе.

В качестве определенного недостатка делителя частоты, построенного на счетчиках с обратной связью, можно отметить очень маленькую длительность выходных импульсов. Если требуется сформировать строго симметричное колебание, то на выходе подобного делителя необходимо дополнительно поставить одноразрядный двоичный делитель частоты, выполненный на Т-триггере. В этом случае на выходе делителя частоты будет формироваться "меандр" с очень высокой точностью равенства длительности нулевого и единичного потенциала. Такой сигнал вполне можно использовать в качестве опорного колебания для преобразователя частоты радиоприемного устройства, однако ДПКД обычно не применяется для формирования частоты непосредственно. Он обычно применяется в составе синтезаторов частот с цепью фазовой автоподстройки частоты, где нет необходимости строго выдерживать равенство длительности нулевого и единичного потенциала в выходном сигнале. Схему синтезатора частоты с применением фазовой автоподстройки частоты мы рассмотрим в последующих главах.

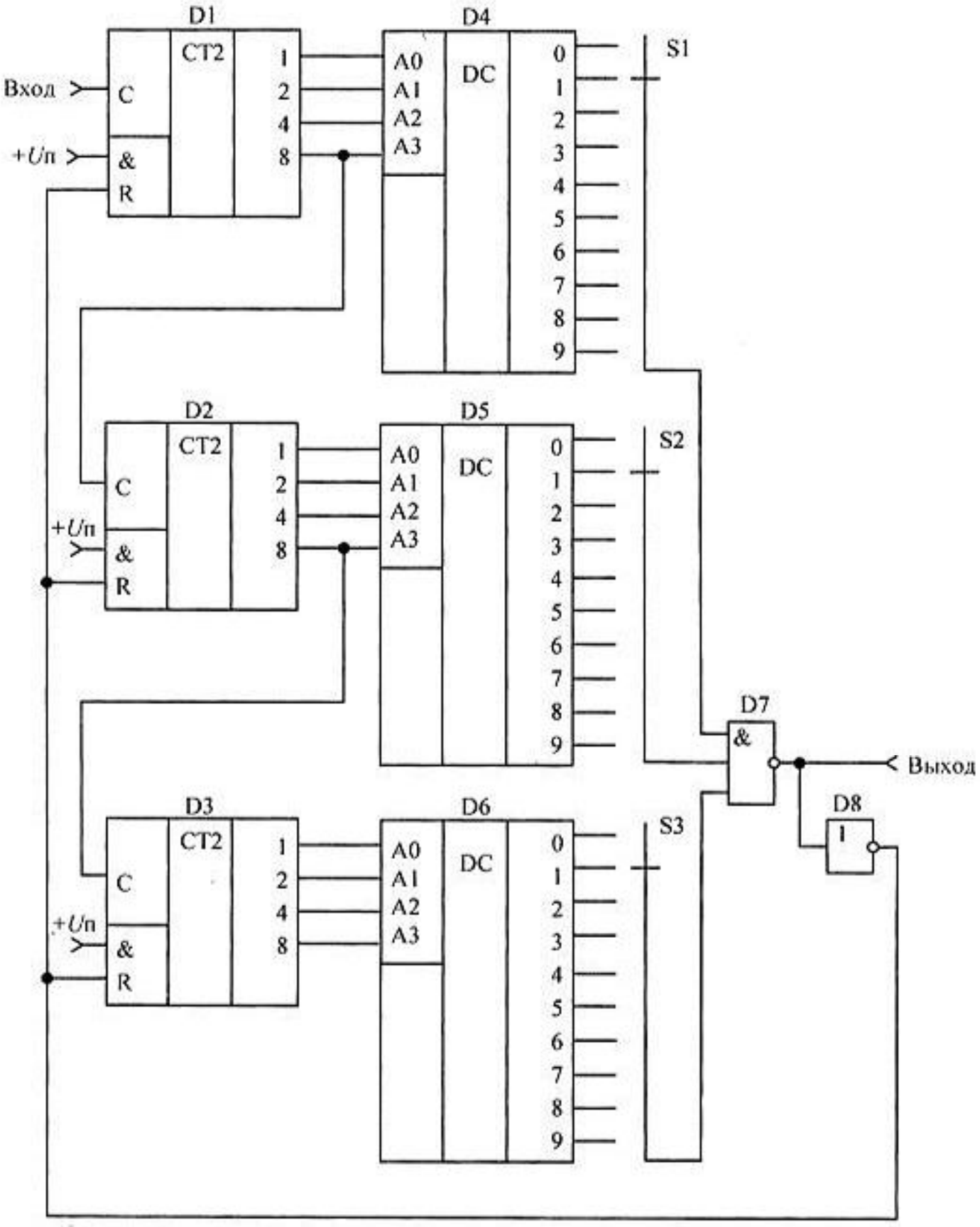


Рис. 8.43. Схема делителя частоты с переменным коэффициентом деления (с максимальным коэффициентом деления, равным 1000)

Недвоичные счетчики с предварительной связью

В счетчиках с обратной связью исключаются последние состояния двоичного счетчика. Можно поступить по-другому. Начать с последнего состояния счетчика и, воспользовавшись вычитающим счетчиком, определить нулевое состояние счетчика. Это состояние очень просто можно обнаружить при помощи логического элемента "И". В данной схеме начинать счет необходимо с числа, которое будет определять коэффициент деления делителя, построенного на таком счетчике.

При построении счетчика по таким принципам необходимо иметь возможность предварительной записи двоичного (или недвоичного) числа в цифровой счетчик. При предварительной записи внутреннего состояния цифровой счетчик должен вести себя как параллельный регистр. Для реализации этого принципа работы нам потребуется, как и при построении схемы универсального регистра, коммутатор логических сигналов.

Напомним, что в качестве коммутатора вполне успешно используется логический элемент "2И-2ИЛИ", главное обеспечить подачу на элементы "И" противофазных управляющих сигналов. Это условие нам обеспечивает логический инвертор.

Одна из схем цифрового счетчика, с возможностью параллельной записи двоичных кодов в его внутренние триггеры, приведена на рис. 8.44. В этой схеме вход С предназначен для подачи тактовых импульсов. Его часто обозначают "–1", т. к. при подаче на этот вход импульсов содержимое счетчика уменьшается на единицу. Входы счетчика D0 ... D3 предназначены для записи в него произвольного двоичного числа. Запись этого числа во внутренние триггеры счетчика производится по сигналу тактовой синхронизации, при условии присутствия разрешающего сигнала на входе параллельной записи PE.

На первый взгляд приведенная схема достаточно сложна. Однако если ее проанализировать, то можно увидеть, что схема состоит из совершенно одинаковых узлов. Информационные входы D-триггеров могут быть подключены либо к входу параллельной записи, либо к собственному инверсному выходу. При подключении входа триггера к собственному инверсному выходу будет реализован счетный режим работы. Так как в схеме применено четыре триггера, то для коммутации источников сигналов на входы этих триггеров требуется четыре мультиплексора (коммутатора). Коммутатор на входе первого триггера обведен пунктирной линией.

Источник сигнала на тактовых входах триггеров переключается при помощи точно такой же коммутирующей схемы. Входы триггеров в зависимости от

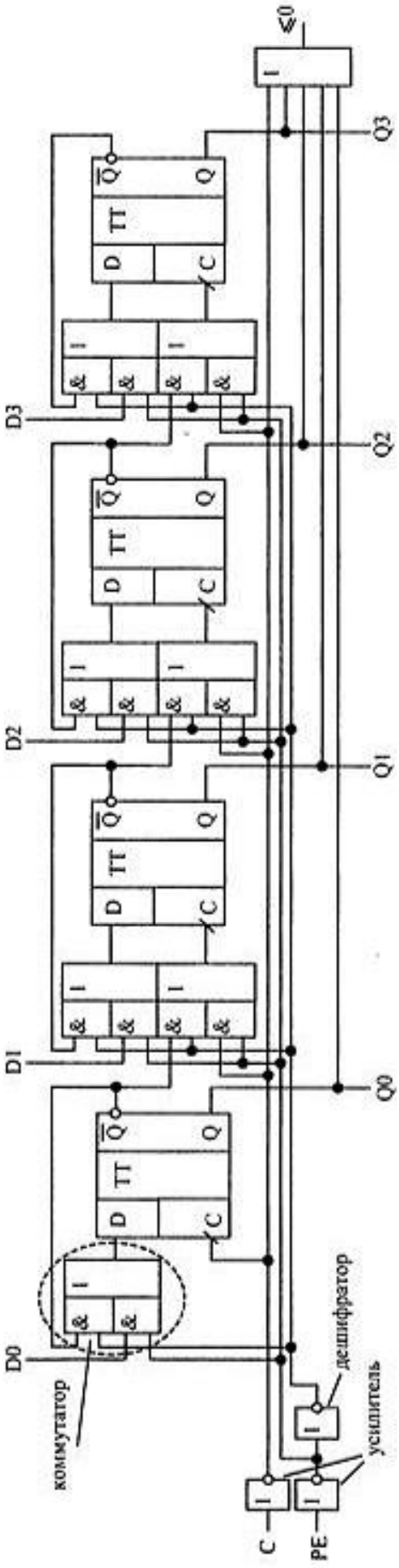


Рис. 8.44. Схема счетчика с возможностью параллельной записи

управляющего сигнала PE подключены либо к инверсному выходу предыдущего триггера (счетный режим работы), либо к цепи синхронизации (режим параллельной записи).

Особо следует остановиться на реализации возможности наращивания разрядности счетчиков. При работе недвоичного счетчика с предварительной записью, как это уже обсуждалось в начале раздела, требуется определять его нулевое состояние. Это легко можно реализовать при помощи четырехвходового логического элемента "4ИЛИ". Однако если необходимо учитывать состояние предыдущих счетчиков, то следует соединить счетный вход счетчика с пятым входом схемы обнаружения нулевого состояния счетчика, как это показано на рис. 8.44.

Условно-графическое обозначение двоичного счетчика с возможностью предварительной записи состояния счетчика приведено на рис. 8.45.

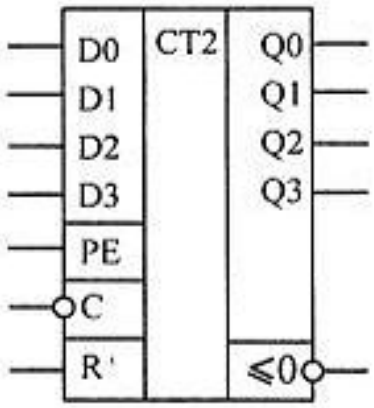


Рис. 8.45. Условно-графическое обозначение счетчика с возможностью параллельной записи

Ну а теперь, точно так же, как и в предыдущем примере, попробуем реализовать делитель частоты с коэффициентом деления 1000. Вспомним, что при разработке делителя частоты сначала определяется количество микросхем двоичных счетчиков. Для этого определим степень числа 2, при которой число $M = 2^n$ будет больше требуемого числа 1000.

Получаем число десять. При возведении основания системы счисления 2 в 10-ю степень получится число 1024. При использовании четырехразрядных двоичных счетчиков достаточно будет трех микросхем, т. к. в трех микросхемах будет $3 \times 4 = 12$ триггеров, что заведомо больше минимально необходимого числа триггеров.

Следующим этапом построения делителя частоты будет перевод коэффициента деления 1000 в двоичное представление. *Перевод чисел между системами счисления мы рассматривали в главе 5.* Десятичное число 1000_{10} в двоичном виде будет выглядеть как $0011\ 1110\ 1000_2$. Как мы уже говорили, с этого числа должен начинаться счет вычитающего счетчика.

Схема делителя частоты с коэффициентом деления, равным 1000, приведена на рис. 8.46. В этой схеме первая микросхема является младшей, поэтому в нее загружается младшая тетрада числа предварительной записи 1000_{10}, равная 1000_2. В следующую микросхему загружается число 1110_2, а в последнюю микросхему — 0011_2.

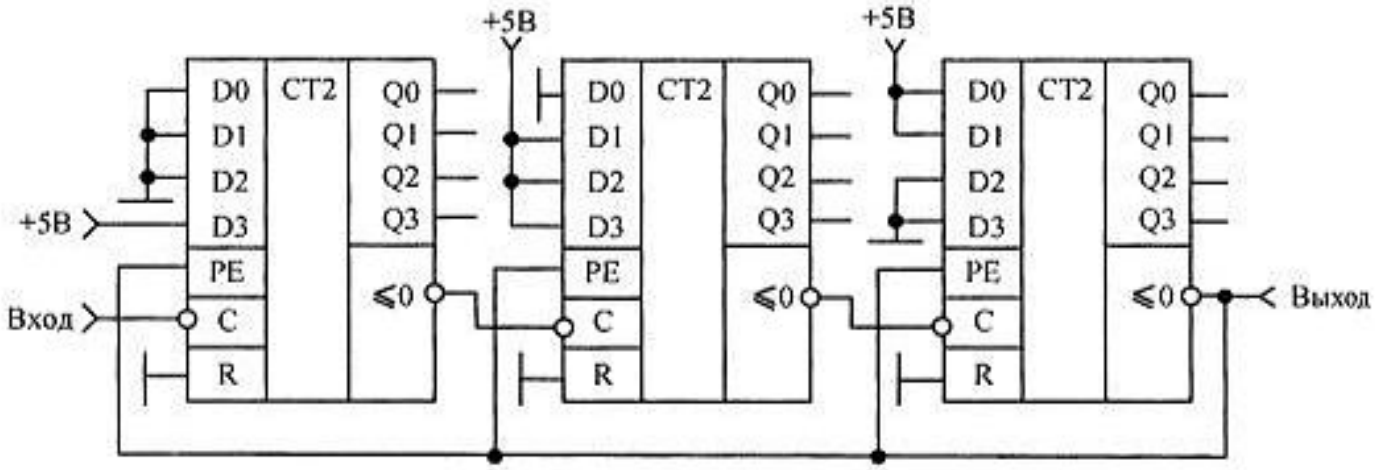


Рис. 8.46. Схема делителя на 1000, построенного на основе трех двоичных счетчиков с предварительной записью

Для определения нулевого состояния триггеров счетчика служит выход "≤0". Для этого внутри микросхемы расположен логический элемент "5ИЛИ". Чтобы определить, обнулились ли все три микросхемы, в схеме на рис. 8.46 счетные входы микросхем "С" соединяются с выходом переноса предыдущей микросхемы. Как только такое состояние обнаруживается, сигнал поступает на входы параллельной записи PE, и в счетчик снова записывается число 1000_{10}. В результате работы приведенной схемы на выходе делителя частоты импульс возникает один раз после подачи на его вход тысячи импульсов.

✧ *Обратите внимание*, что на этот раз коэффициент деления определяется не принципиальной схемой делителя, а задается кодом двоичного числа, подаваемого на вход параллельной записи счетчиков. В результате процесс изменения коэффициента деления счетчика значительно упрощается. Для изменения частоты на выходе делителя достаточно подать нужное число на входы управления. Схема самого делителя, в отличие от схемы недвоичного счетчика с обратной связью, при этом не меняется.

Для построения делителя с переменным коэффициентом деления мы использовали вычитающий счетчик. Можно такую же схему построить на суммирующем счетчике, однако для записи коэффициента деления в этом случае придется воспользоваться отрицательным числом, представленном в дополнительном коде. Для того чтобы получить отрицательное число в этом коде,

необходимо положительное двоичное число проинвертировать и прибавить единицу. Например, для реализации коэффициента деления 1000 возьмем его двоичный эквивалент $0011\ 1110\ 1000_2$. После инвертирования получим число $1100\ 0001\ 0111_2$. Окончательный результат будет равен $1100\ 0001\ 1000_2$.

Для десятиразрядного двоичного кода это число будет равно десятичному эквиваленту 24_{10}. Действительно, если в счетчике с коэффициентом $2^{10} = 1024$ начать считать от числа 24, то ровно через 1000_{10} импульсов счетчик переполнится и его состояние станет равным нулю. В рассмотренном случае был использован двенадцатиразрядный счетчик, поэтому в суммирующий счетчик следует занести число $4096 - 1000 = 3096_{10} = 1100\ 0001\ 1000$.

Синхронные счетчики

В рассмотренных схемах делителей частоты быстродействие всей схемы определяется временем распространения сигнала от входа счетчика до выхода его самого старшего разряда. При этом получается, что чем больше требуемый коэффициент деления, тем больше двоичных разрядов счетчика необходимо для реализации делителя частоты. И тем большее время требуется для распространения сигнала от входа синхронизации счетчика до его выхода, а затем до его входа сброса в исходное состояние или до входа предварительной записи кода. И тем меньше будет предельная частота сигнала, подаваемого на вход разработанного делителя.

Тем не менее, можно обойти эту неприятную особенность недвоичных счетчиков. Нужно, чтобы счетчик подготавливал свое новое состояние в промежутках между тактовыми импульсами и по приходу нового импульса только записывал его. Такие счетчики получили название синхронных счетчиков. Схемы цифровых счетчиков, рассмотренные в предыдущих разделах, по аналогии, получили название асинхронных счетчиков.

Синхронные счетчики на регистрах сдвига

Первая схема счетчика, построенного по данному принципу, которую мы рассмотрим, — это схема кольцевого счетчика. Такой счетчик можно реализовать на основе сдвигового регистра. Схема кольцевого синхронного счетчика приведена на рис. 8.47.

Рассмотрим работу этой схемы. Пусть первоначально в кольцевом счетчике записано число 00_2. После первого тактового импульса состояние счетчика станет равным 10_2, после второго — 11_2. Временные диаграммы работы этой схемы приведены на рис. 8.48.

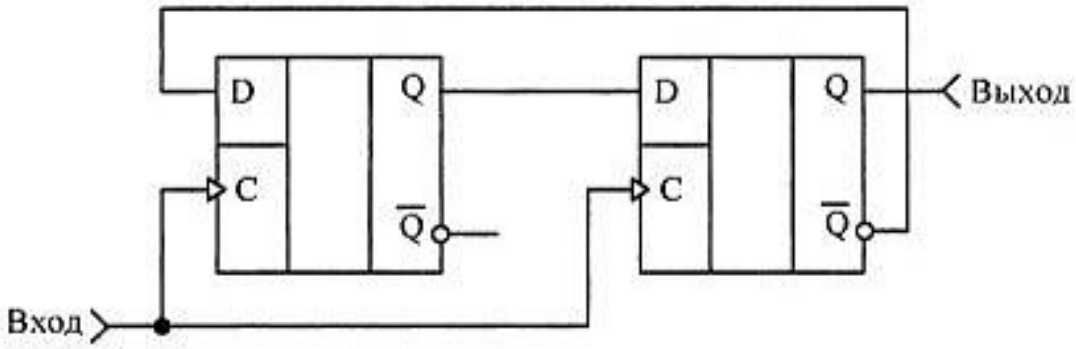


Рис. 8.47. Схема кольцевого счетчика

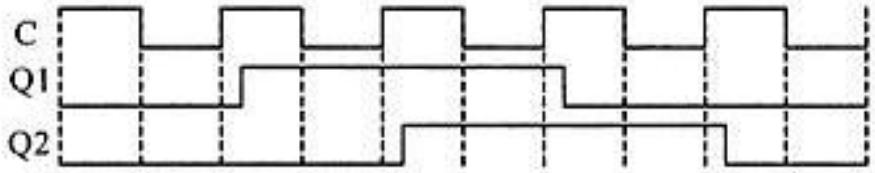


Рис. 8.48. Временные диаграммы кольцевого синхронного счетчика

В результате анализа временных диаграмм можно определить, что коэффициент деления схемы кольцевого счетчика будет равен:

$$K_д = 2 \times n$$

В качестве преимущества схемы кольцевого счетчика можно отметить то, что ее быстродействие зависит только от времени задержки одного триггера. Это означает, что на кольцевых счетчиках можно реализовывать самые быстродействующие делители частоты.

То, что коэффициент деления пропорционален не степени количества триггеров, а только их сумме, является недостатком данной схемы. Это означает, что при увеличении коэффициента деления частоты сложность схемы неоправданно возрастает по сравнению со схемой двоичного счетчика.

Еще одним недостатком схемы кольцевого счетчика является то, что при количестве триггеров большем трех, в результате воздействия помехи в регистр может быть записано число, содержащее несколько единиц. В результате коэффициент деления схемы изменится, а это является недопустимым. Временные диаграммы сигналов на входе и выходах 3-разрядного кольцевого счетчика при правильной и ошибочной работе приведены на рис. 8.49 и 8.50 соответственно.

Тем не менее счетчики, построенные на кольцевых регистрах, являются самыми быстродействующими цифровыми счетчиками.

Для того чтобы избежать неправильной работы счетчика, в него можно ввести схему контроля правильной работы. В простейшем случае это может быть обычный логический элемент "2И-НЕ". Этот элемент будет контроли-

ровать состояние счетчика, соответствующее единицам во всех его разрядах. Схема 2-разрядного счетчика со схемой проверки правильности его работы приведена на рис. 8.51. В этой схеме триггеры счетчика при поступлении импульсов на тактовый вход последовательно заполняются единицами. Как только все триггеры будут заполнены единицами, на выходе логического элемента "2И-НЕ" появится уровень логического нуля. При поступлении следующего тактового импульса этот ноль будет записан в первый триггер счетчика. В дальнейшем работа счетчика повторяется.

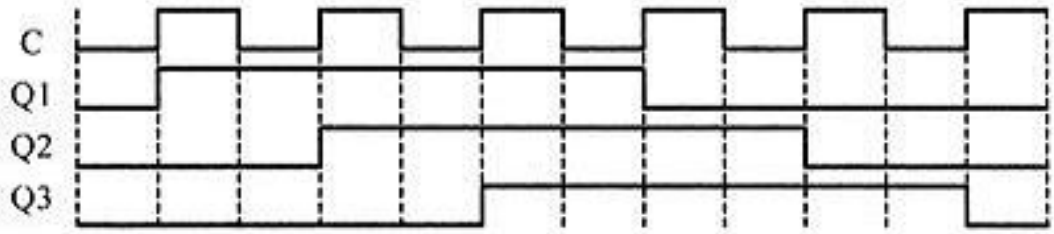


Рис. 8.49. Временные диаграммы сигналов
3-разрядного кольцевого синхронного счетчика при правильной работе

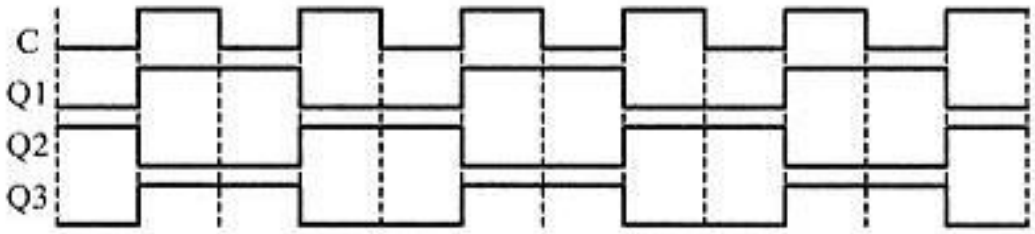


Рис. 8.50. Временные диаграммы сигналов
3-разрядного кольцевого синхронного счетчика при ошибочной работе

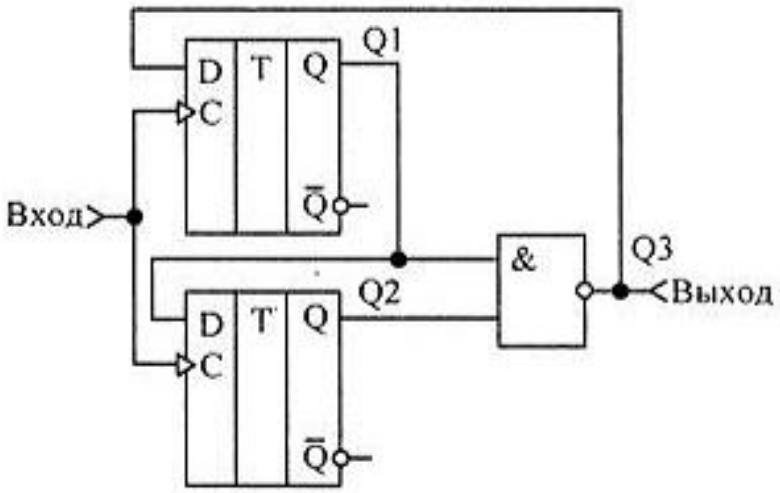


Рис. 8.51. Схема 2-разрядного счетчика с проверкой правильности его работы

Временные диаграммы сигналов на выходах этого счетчика приведены на рис. 8.52.

В результате анализа временных диаграмм сигналов, приведенных на рис. 8.52, можно определить, что коэффициент деления схемы частоты для кольцевого счетчика с проверкой правильности его работы будет равен:

$$K_д = n + 1,$$

где n — количество триггеров в кольцевом счетчике.

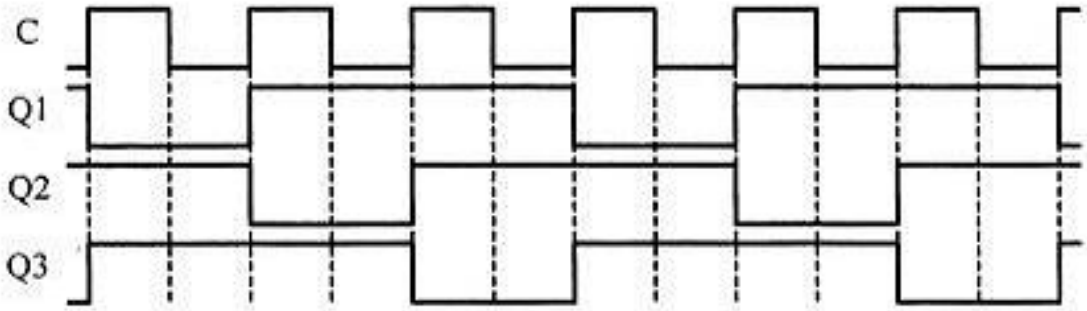


Рис. 8.52. Временные диаграммы кольцевого синхронного счетчика

Как видно из временных диаграмм, приведенных на рис. 8.51, в качестве выходного сигнала можно использовать сигнал с выхода любого триггера регистра или с выхода схемы "И-НЕ". Частота сигналов будет абсолютно идентична. Они отличаются только начальной фазой колебания. Это означает, что схему кольцевого счетчика с проверкой правильности его работы можно использовать в качестве многофазного генератора.

Еще одной особенностью рассмотренной схемы счетчика является то, что состояния счетчика описываются линейным кодом. Это означает, что при индикации состояний счетчика при помощи десятичного индикатора в схеме не потребуется дополнительный дешифратор. Правда, для десятичного счетчика потребуется целых девять триггеров.

Синхронные двоичные счетчики

Основным недостатком делителей, построенных на кольцевых счетчиках, является малый коэффициент деления. Двоичные счетчики в этом смысле более эффективны. Попробуем разработать синхронный счетчик, работающий по двоичному закону. Для этого обратим внимание, что переключение следующего разряда двоичного счетчика происходит только тогда, когда состояние всех предыдущих его разрядов равно единицам. Это состояние может быть легко определено при помощи логического элемента "И". Принципиальная схема одного из вариантов реализации четырехразрядного синхронного двоичного счетчика, построенного по этому принципу, приведена на рис. 8.53.

В этой схеме счетные триггеры реализованы на основе JK-триггеров. В ней все триггеры переключаются одновременно, т. к. входной тактовый сигнал

счетчика подается на вход синхронизации сразу всех триггеров. Разрешение переключения для каждого счетного триггера формируется схемами "И", включенными между этими триггерами.

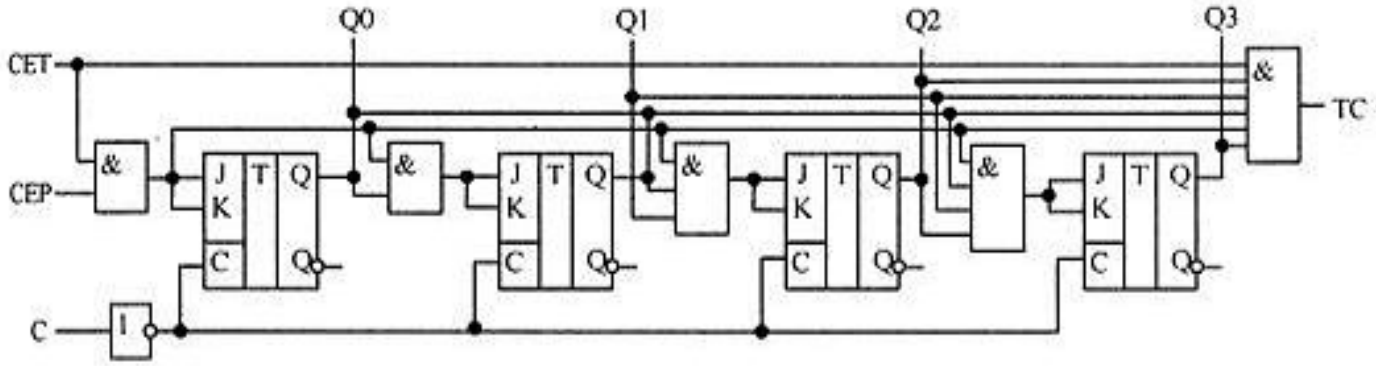


Рис. 8.53. Принципиальная схема четырехразрядного синхронного двоичного счетчика

При разработке синхронного двоичного счетчика, состоящего из нескольких микросхем, для формирования переноса, предназначенного для последующих разрядов двоичного счетчика, в приведенной на рис. 8.53 схеме синхронного счетчика формируется сигнал ТС. В следующих микросхемах этот сигнал подается на входы CEP или CET. Переключение триггеров в схеме возможно только при подаче на оба этих входа логической единицы.

В качестве примера условно-графического обозначения синхронного двоичного счетчика на рис. 8.54 приведено обозначение микросхемы К1533ИЕ10.

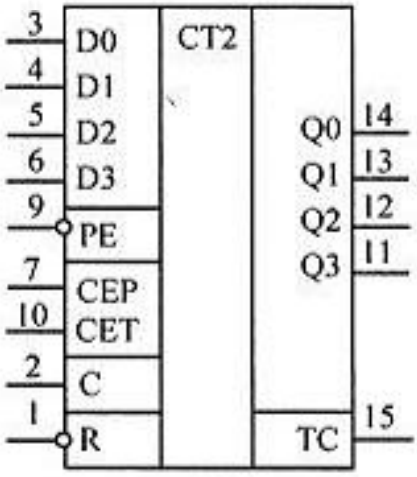


Рис. 8.54. Условно-графическое обозначение синхронного счетчика с возможностью параллельной записи

Рассмотрим в качестве примера реализацию 16-разрядного двоичного счетчика. Для этого используем четыре микросхемы К1533ИЕ10. Получившаяся принципиальная схема синхронного 16-разрядного двоичного счетчика приведена на рис. 8.55.

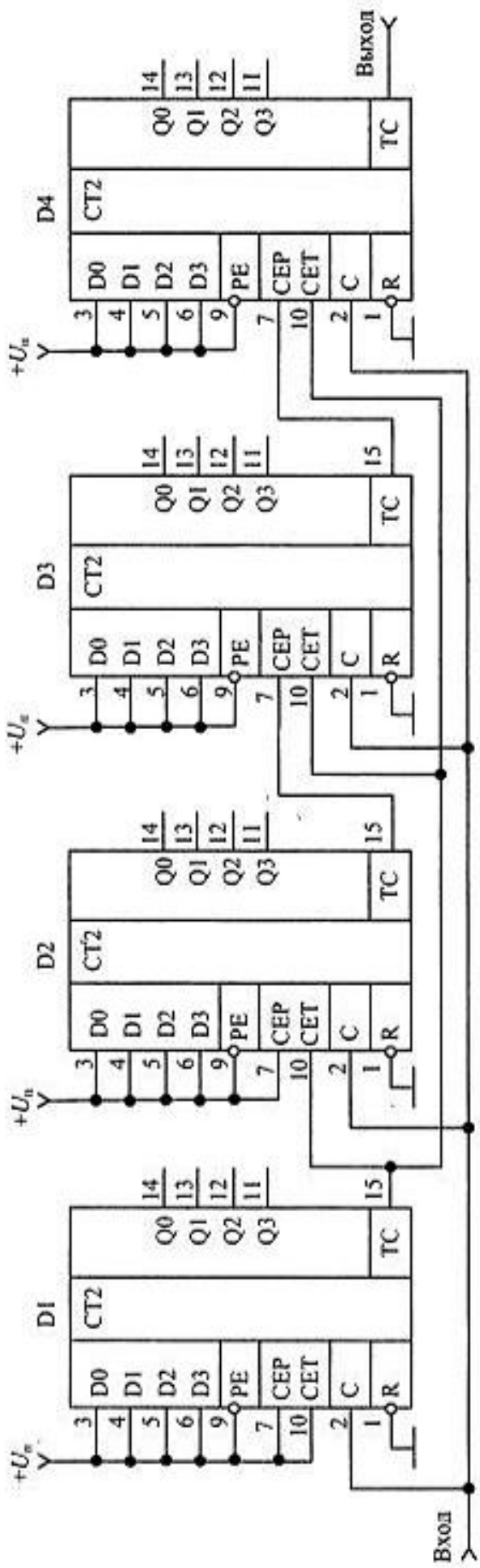


Рис. 8.55. Принципиальная схема 16-разрядного синхронного двоичного счетчика

Будет ли счетчик находиться в режиме счета или в режиме параллельной записи, определяется потенциалом на входах микросхем PE. При нулевом потенциале на входе PE производится запись информации с входов данных D во внутренние триггеры счетчиков. Именно поэтому на входы PE всех микросхем счетчика подан высокий потенциал (они подключены к источнику питания).

В схеме, приведенной на рис. 8.55, не используются входы параллельной записи, однако мы знаем, что входы цифровых микросхем нельзя "бросать" в воздухе, поэтому их следует присоединить либо к источнику питания, либо к общему проводу схемы. В данной схеме входы параллельной записи присоединены к источнику питания. Так как в принципиальной схеме, приведенной на рис. 8.54, применены микросхемы синхронных счетчиков, то все входы синхронизации должны быть соединены параллельно. Только в этом случае запись нового состояния счетчика во внутренние триггеры будет производиться одновременно.

Микросхема младших разрядов двоичного счетчика D1 должна работать всегда, пока на ее вход синхронизации поступают тактовые импульсы, поэтому входы разрешения счета CEP и CET в этой микросхеме присоединены к источнику питания.

Следующая микросхема D2 должна переключиться только тогда, когда во всех триггерах микросхемы D1 будет записана логическая единица. Для этого вход разрешения счета CEP соединен с выходом TC микросхемы младших разрядов D1. Второй вход разрешения счета остается подключенным к питанию схемы.

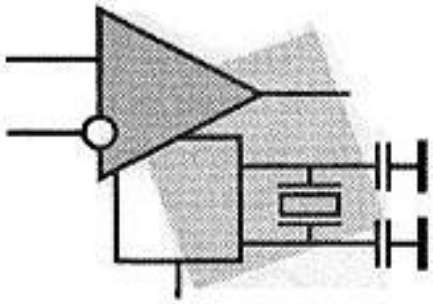
Следующая микросхема D3 подключается так же. Однако если не принять дополнительных мер, то время распространения сигнала разрешения счета при увеличении количества микросхем, использованных в счетчике, будет увеличиваться пропорционально количеству микросхем. Для того чтобы избежать этой ситуации, в схеме использован вспомогательный вход разрешения счета CEP. Сигнал с выхода TC микросхемы D1 подается на входы CEP всех последующих разрядов.

При необходимости рассмотренная схема счетчика может быть легко превращена в схему любого недвоичного счетчика, как при помощи обратных связей, так и при помощи предварительной записи исходного состояния счетчика. Принципы разработки недвоичных счетчиков для синхронных схем ничем не отличаются от принципов разработки недвоичных счетчиков для асинхронных схем. Разница будет заключаться только в возможности применения более высокой частоты на входе делителя частоты, реализованного на основе синхронного двоичного или десятичного счетчика.

Итоги

В данной главе мы рассмотрели особенности работы триггеров — основных элементов цифровых устройств. Теперь мы знаем, какие бывают разновидности цифровых триггеров и особенности применения каждого из этих видов. Кроме того, мы познакомились с еще одним видом последовательностных устройств, который широко применяется в цифровой и микропроцессорной технике, — регистрами. Именно эти устройства позволяют запоминать многоразрядные двоичные числа, поэтому они являются составными частями более сложных цифровых устройств. Не менее важными последовательностными устройствами являются цифровые счетчики различных видов. Без них невозможно построение устройств управления — контроллеров и устройств преобразования частоты, которые широко применяются в технике связи. Очень часто требуется знать содержимое регистров и счетчиков, поэтому в следующей главе мы познакомимся с устройствами отображения информации — индикаторами.

ГЛАВА 9



Индикаторы

Индикаторы предназначены для отображения различных видов информации для человека. Простейший вид информации — это двоичная информация. Например: исправен предохранитель или вышел из строя, включено питание или нет, задействован режим передачи или нет.

Особым видом двоичной информации можно считать пиктограммы, т. е. небольшие картинки. Примером таких картинок можно назвать батарейку или антенну, вертикальные линии, отображающие уровень заряда этой батарейки или уровень принимаемого сигнала, колокольчик, будильник или замочек. Пример изображения пиктограмм приведен на рис. 9.1.

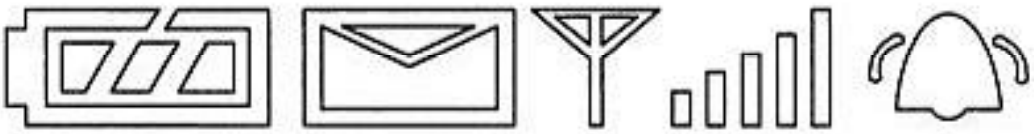


Рис. 9.1. Пример пиктограмм

Часто требуется отображать информацию в десятичном коде. В этом случае используется десятиразрядный бинарный код. Каждому разряду ставится в соответствие изображение символа десятичной цифры. В этом смысле десятичный код практически не отличается от пиктограммы. В отличие от нее изображения десятичных цифр располагаются друг под другом. Подобная конструкция реализована в газоразрядных индикаторах. Пример десятичного индикатора приведен на рис. 9.2. В каждый момент времени может отображаться только один символ.

С целью экономии количества выводов, необходимых для отображения одного цифрового разряда и упрощения конструкции индикаторов, были разработаны семисегментные индикаторы. В них цифровые данные отображаются

при помощи семи сегментов. Внешний вид сегментов такого индикатора и их имена приведены на рис. 9.3.

Рис. 9.2. Пример десятичного индикатора

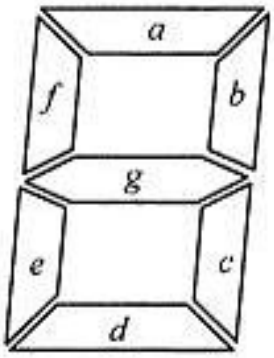


Рис. 9.3. Внешний вид сегментов семисегментного индикатора

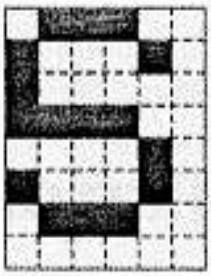


Рис. 9.4. Пример изображения буквы S на матричном индикаторе

Применение семисегментных индикаторов позволяет сформировать все десятичные цифры и часть букв алфавита, однако такие индикаторы отображают не все текстовые символы. Для отображения всех букв алфавита в настоящее время используются матричные индикаторы. В этих индикаторах изображение символа разбивается на отдельные точки точно так же, как это делается на экране телевизора. При этом, чем большее количество точек применяется для отображения внешнего вида символа, тем более качественной получается картинка. Однако увеличение количества точек в матрице приводит к удорожанию индикатора и усложнению управлением подобного индикатора.

Наиболее распространены матричные индикаторы 5×7. В этом индикаторе изображение символа представляется пятью столбцами и семью строками. Для отделения символов друг от друга используется еще один столбец. Точно так же для отделения одной строки символов от другой (если она имеется) применяется дополнительная строка. Пример отображения на таком индикаторе латинской буквы 'S' приведен на рис. 9.4.

Для отображения перечисленных видов информации можно воспользоваться индикаторами, выполненными с применением различных физических принципов. С этой целью могут быть использованы малогабаритные лампочки накаливания, газоразрядные индикаторные лампы, жидкокристаллические или светодиодные индикаторы. Для отображения могут быть использованы и другие виды индикаторов, но в настоящее время наиболее распространены перечисленные выше виды. Рассмотрим подробнее преимущества и недостатки каждого из них.

Малогабаритные лампочки накаливания

Наиболее простой схемой подключения к цифровым устройствам обладают лампочки накаливания. Пример схемы подключения малогабаритной индикаторной лампы накаливания к цифровой микросхеме, выполненной по ТТЛ-технологии, приведен на рис. 9.5.

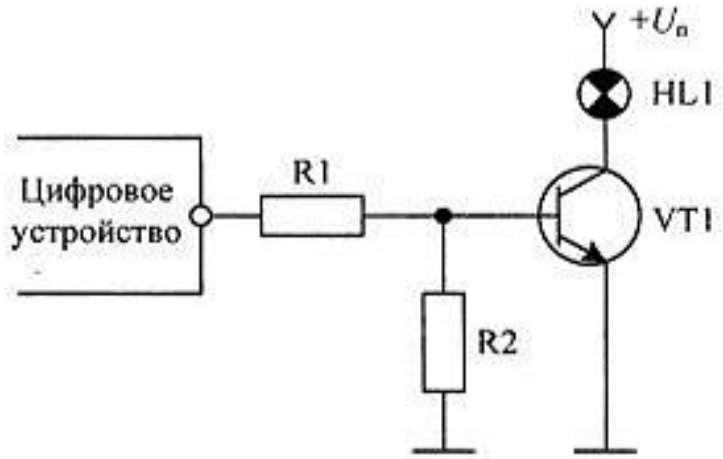


Рис. 9.5. Схема подключения индикаторной лампы накаливания к цифровой ТТЛ-микросхеме

В приведенной схеме потребовался транзистор, т. к. ток, протекающий через лампочку накаливания, достаточно велик. Его значение достигает нескольких сотен миллиампер. Кроме того, рассмотренная схема включения позволяет использовать лампочки накаливания с напряжением питания, отличающимся от напряжения питания цифровых микросхем. Иными словами, на транзисторе собран простейший усилитель цифрового сигнала, преобразующий напряжение ТТЛ-уровней цифрового устройства в дискретный сигнал наличия или отсутствия тока через индикаторную лампочку накаливания.

Расчет транзисторного ключа

Обычно студенты пугаются слова "усилитель". Однако в случае цифрового сигнала усилитель вырождается в схему электронного ключа. Это означает, что не нужно рассчитывать точное значение коэффициента усиления. При достаточно большом коэффициенте усиления транзистор переходит в режим ограничения тока, и выходной ток будет определяться только сопротивлением нагрузки и не зависеть от параметров транзистора. Поэтому достаточно определить только минимальный коэффициент усиления по току.

Рассчитаем этот коэффициент. Пусть для индикаторной лампы требуется ток 120 мА, а цифровая микросхема может выдать ток единицы около 4 мА (этот ток определяется по техническим условиям или DATASHEET на выбранную

микросхему). Тогда минимальный коэффициент усиления $h_{21э}$ можно определить по формуле:

$$h_{21э} = \frac{I_k}{I_б}$$

В нашем случае ток коллектора равен току, протекающему через индикаторную лампу, а ток базы — это максимальный допустимый выходной ток цифровой микросхемы ($I_{вых1}$). Делим 120 мА на 4 мА. Получаем минимальный коэффициент усиления по току, равный 30, т. е. в данном случае подойдет практически любой маломощный транзистор, например КТ3107.

Теперь следует обратить внимание на то, что транзистор управляется током, а цифровая микросхема является генератором напряжения. В простейшем случае для преобразования напряжения в ток можно использовать резистор. Эквивалентная схема подключения базовой цепи транзистора к цифровой ТТЛ-микросхеме приведена на рис. 9.6.

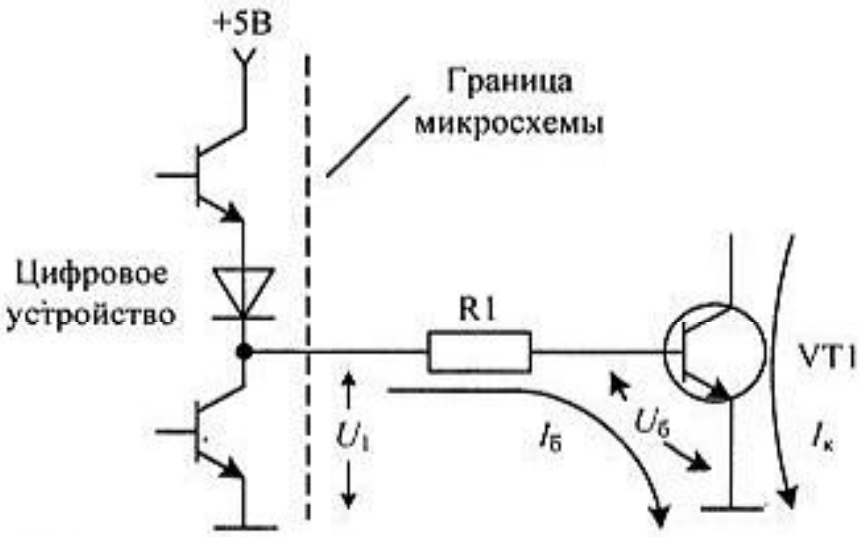


Рис. 9.6. Эквивалентная схема подключения транзисторного ключа к цифровой ТТЛ-микросхеме

В приведенной схеме ток базы транзистора задает резистор R1. Рассчитаем его сопротивление. Для этого необходимо определить падение напряжения на этом резисторе. Минимальное напряжение высокого уровня на выходе ТТЛ-микросхемы при максимальном допустимом токе единицы равно 2,4 В. Падение напряжения на базовом переходе транзистора можно считать постоянным и для кремниевых транзисторов равным 0,7 В. Тогда падение напряжения на сопротивлении R1 можно определить по формуле:

$$U_{R1} = U_1 - U_б = 2,4\text{В} - 0,7\text{В} = 1,7\text{ В}$$

Так как к цифровому выходу подключен только транзисторный ключ, то зададимся максимально возможным током цифровой микросхемы 4 мА. Тогда по закону Ома можно определить сопротивление резистора R1 как отношение падения напряжения на этом резисторе к току, протекающему через него:

$$R1 = \frac{1,7\ В}{4\ мА} = 425\ Ом$$

При выборе резистора из 10% шкалы можно взять резистор 510 Ом (больше чем рассчитали, чтобы не превысить допустимый ток цифровой микросхемы). При работе транзисторного ключа при комнатной температуре расчет на этом заканчивается. Если же предполагается работа транзисторного ключа при повышенных температурах, то транзистор может самопроизвольно открываться обратным током коллектора. Эквивалентная схема цепи протекания этого тока приведена на рис. 9.7.

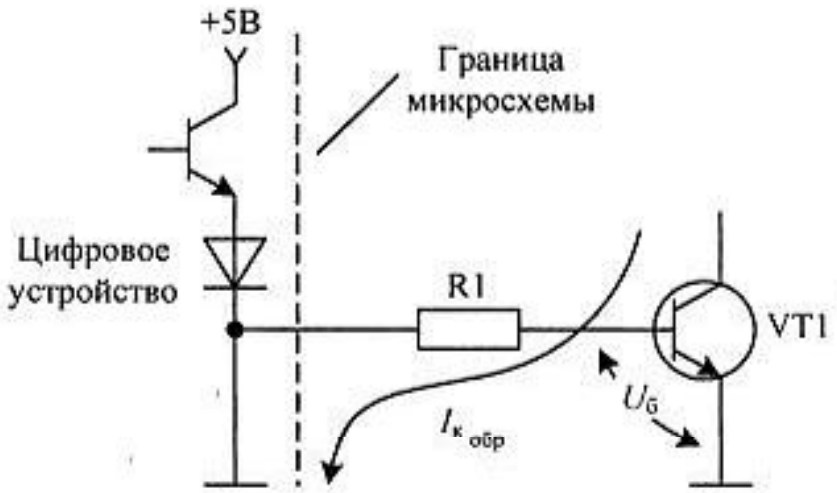


Рис. 9.7. Эквивалентная схема цепи протекания обратного коллекторного тока

В схеме, приведенной на рис. 9.7, видно, что на резисторе R1 обратный ток коллектора транзистора VT1 может создать падение напряжения 0,7 В и тем самым открыть транзистор. Для того чтобы уменьшить падение напряжения, можно параллельно этому резистору подключить еще один резистор (как показано на рис. 9.8) и, таким образом, уменьшить открывающее напряжение на базе транзистора.

В схеме, приведенной на рис. 9.8, можно задаться током, протекающим через резистор R2 в режиме выдачи цифровой микросхемой единичного уровня. Пусть этот ток будет в три раза меньше базового тока транзистора. Тогда ток через резистор R2 будет равен:

$$I_{R2} = \frac{4\ мА}{3} = 1,3\ мА$$

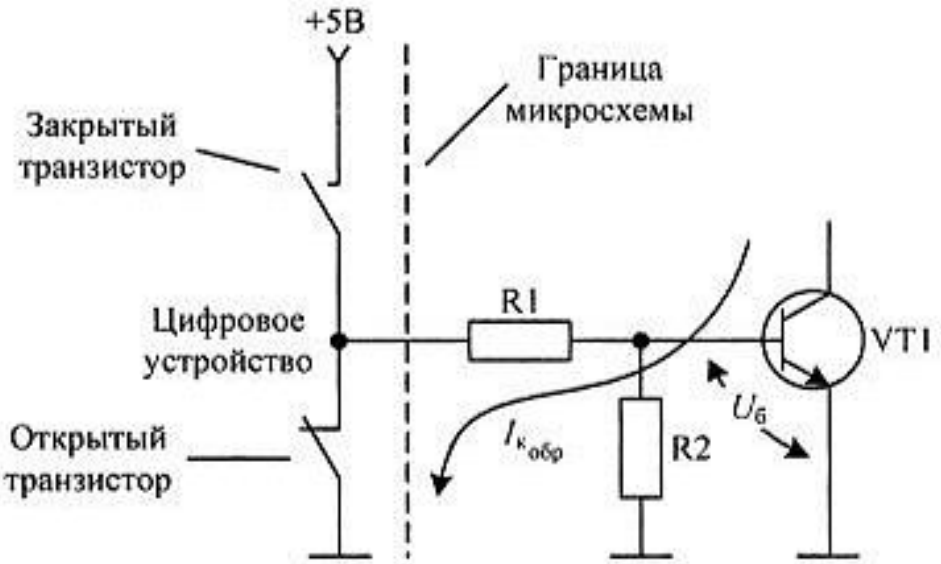


Рис. 9.8. Эквивалентная схема шунтирования цепи протекания обратного коллекторного тока $I_{к0}$ транзисторного ключа резистором

Определим сопротивление резистора R2. Для этого воспользуемся законом Ома. Учитывая, что падение напряжения на базовом переходе транзистора является константой и равно 0,7 В:

$$R2 = \frac{U_б}{I_{R2}} = \frac{0{,}7\ \text{В}}{1{,}3\ \text{мА}} = 510\ \text{Ом}$$

В режиме выдачи цифровой микросхемой логического нуля сопротивления R1 и R2 соединяются параллельно, и в рассчитанном случае падение напряжения уменьшается вдвое. ✧ *Обратите внимание, что схема на входе транзистора очень похожа на делитель напряжения, однако не является им.* Если бы это был делитель напряжения, то напряжение на базе транзистора уменьшалось бы в два раза, однако на самом деле оно уменьшается значительно больше!

Газоразрядные лампы

К сожалению, малогабаритные лампочки накаливания не отличаются надежностью, т. к. при включении питания через них протекает значительный ток, в результате воздействия которого на нить накаливания лампа может выйти из строя. Кроме того, они боятся механических ударов. Эти причины, а также большой потребляемый ток, привели к тому, что в настоящее время такие индикаторы практически не используются. Большее распространение получили газоразрядные индикаторы. Эти индикаторы, в отличие от ламп накаливания, управляются не напряжением, а током. Поэтому в принципиальную схему устройства приходится вводить токоограничивающий резистор. Схема включения газоразрядного индикатора приведена на рис. 9.9.

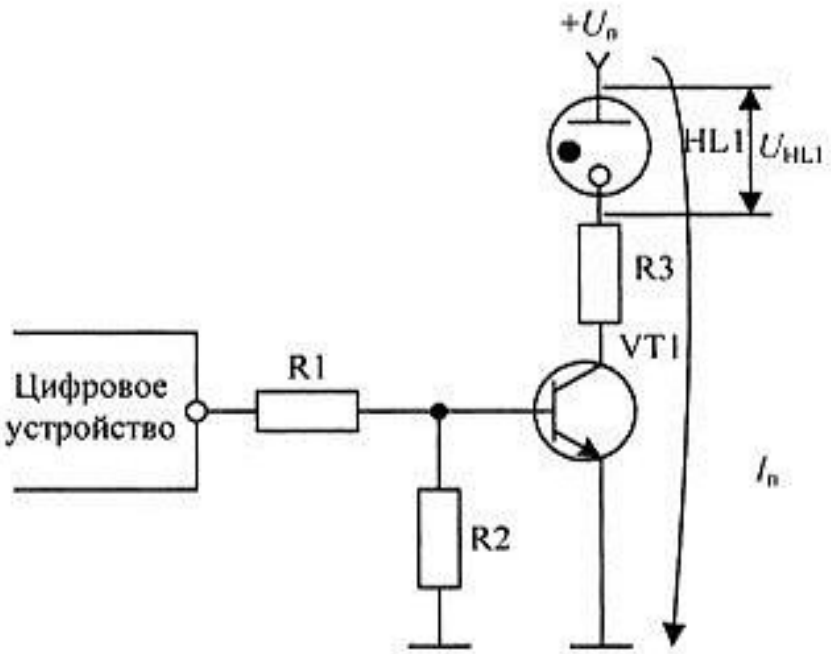


Рис. 9.9. Схема подключения индикаторной газоразрядной лампы к цифровой ТТЛ-микросхеме

В этой схеме транзистор необходим в основном для согласования по напряжению, т. к. газоразрядные индикаторы питаются от источника напряжением 180 … 300 В (напряжение зажигания газоразрядной лампы). Поэтому транзистор должен выдерживать напряжение до 300 В. Что касается сопротивления резистора R3, то оно рассчитывается по закону Ома. Необходимо от напряжения питания отнять падение напряжения на зажженной индикаторной лампе, которое можно взять из справочника по индикаторным лампам (обычно 80 В), и поделить на ток этой лампы. Падением напряжения на открытом транзисторе VT1 можно пренебречь. Например:

$$R3 = \frac{U_п - U_{HL1}}{I_п} = \frac{200\ В - 80\ В}{1\ мА} = 120\ кОм$$

Газоразрядные индикаторы используются как для индикации битовой информации, так и для отображения десятичной информации. При построении десятичных индикаторов катод газоразрядных индикаторов выполняется в виде десятичных цифр, как это показано на рис. 9.2. Пример индикаторной панели, выполненной на газоразрядных индикаторах, приведен на рис. 9.10.

Для уменьшения габаритов цифрового устройства и упрощения его принципиальной схемы были разработаны специальные микросхемы дешифраторов, выдерживающие напряжение до нескольких сотен вольт, например отечественная микросхема К155ИД1. Принципиальная схема подключения десятичного газоразрядного индикатора к микросхеме К155ИД1 приведена на рис. 9.11.

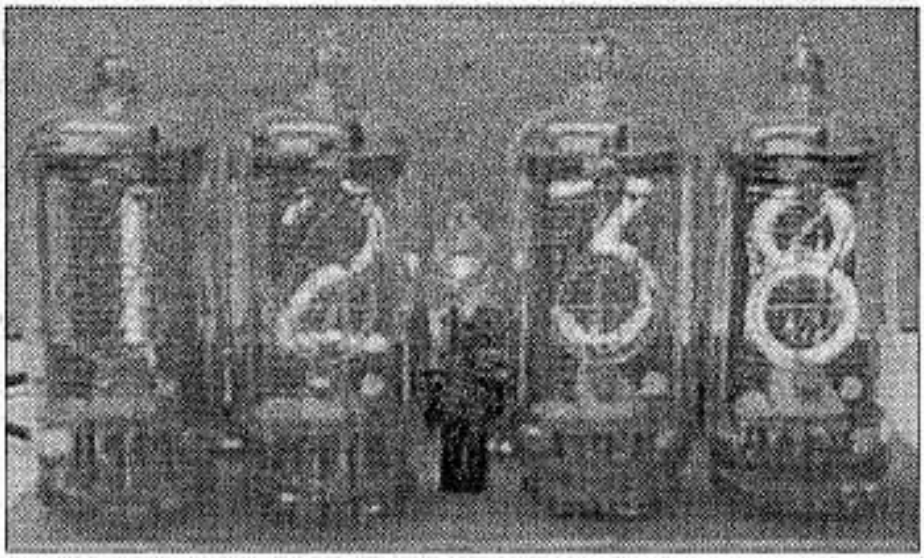


Рис. 9.10. Внешний вид индикаторной панели на газоразрядных лампах

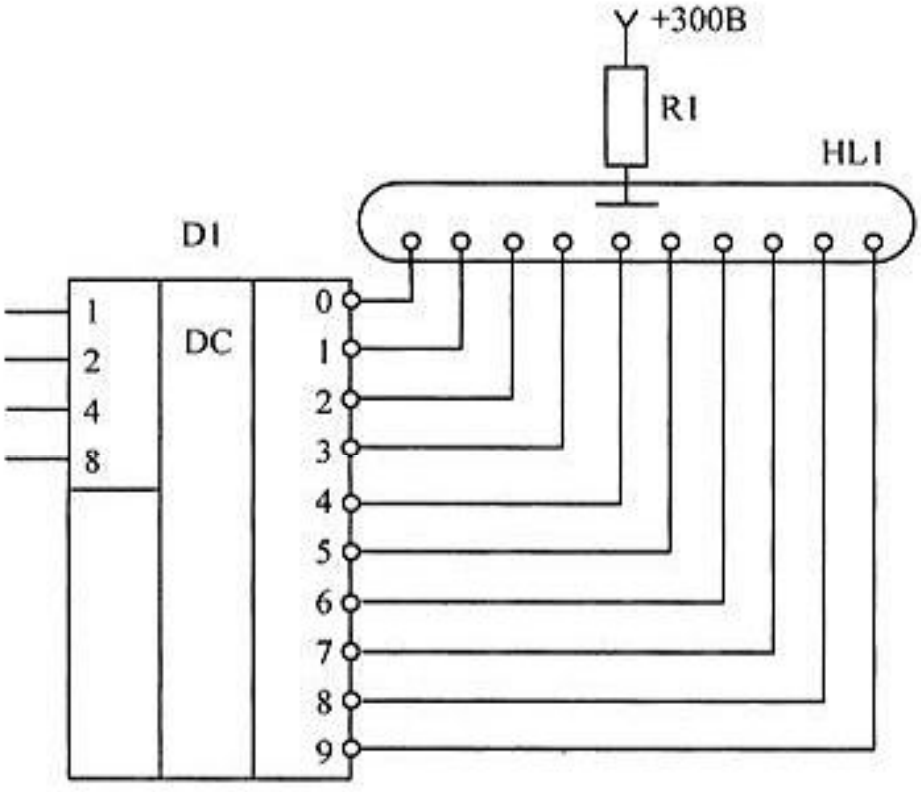


Рис. 9.11. Схема подключения индикаторной газоразрядной лампы к десятичному дешифратору

На вход этой схемы подается двоично-десятичный код. Он преобразуется микросхемой D1 в инверсный линейный десятичный код. Инверсия нужна для того, чтобы ток протекал только через тот вывод, двоично-десятичный код которого подан на вход схемы. В результате светится только тот катод, который подключен к этому выводу, а т. к. катод выполнен в форме десятичной цифры, то именно эта цифра и отображается на газоразрядном индикаторе.

Резистор R1 требуется для ограничения тока газоразрядного индикатора до допустимой величины. Одним резистором в схеме можно обойтись потому, что ток может протекать только через один из десяти катодов. Расчет ограничивающего ток резистора не отличается от расчета резистора R3 в схеме подключения одиночного газоразрядного индикатора, приведенной на рис. 9.9.

В настоящее время газоразрядные индикаторы с холодным катодом практически не используются. Обычно применяются более эффективные семисегментные газоразрядные индикаторы с подогревным катодом. Применение катода с подогревом позволяет снизить анодное напряжение подобного газоразрядного индикатора до 20 ... 27 В, а семисегментный анод позволяет увеличить угол обзора индикатора.

Внешний вид одного из газоразрядных индикаторов с подогревным катодом приведен на рис. 9.12.

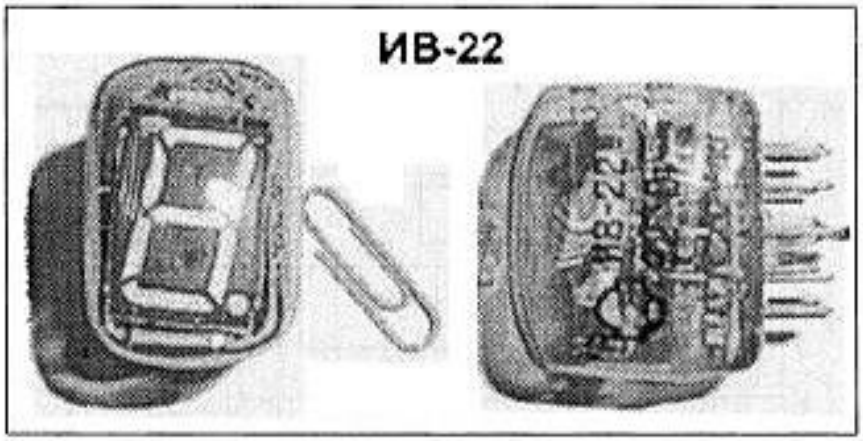


Рис. 9.12. Внешний вид газоразрядного индикатора с подогревным катодом

В описанных индикаторах газ светится не около катода, а в промежутке между управляющей сеткой и анодом. На рис. 9.12 аноды четко видны в виде белых сегментов. Управляющая сетка видна как фиолетовая поверхность, а катод выполнен в виде двух тонких проводников, которые почти незаметны на переднем плане индикатора. Если индикатор поместить за зеленым светофильтром, то ни нить накала, ни управляющая сетка видны не будут.

Если на нить накаливания подать постоянное напряжение, то на ней возникнет падение напряжения. Это напряжение будет суммироваться с анодным напряжением, в результате яркость свечения сегментов в индикаторе будет неравномерной. Конструктивно нить проложена так, чтобы этот эффект свести к минимуму, однако на нить накала подогревного катода желательно подавать переменное напряжение. Так как ток в этом случае будет протекать в различном направлении, то средняя яркость свечения сегментов будет равномерной.

Схема подключения газоразрядного индикатора с подогревным катодом к семисегментному дешифратору приведена на рис. 9.13.

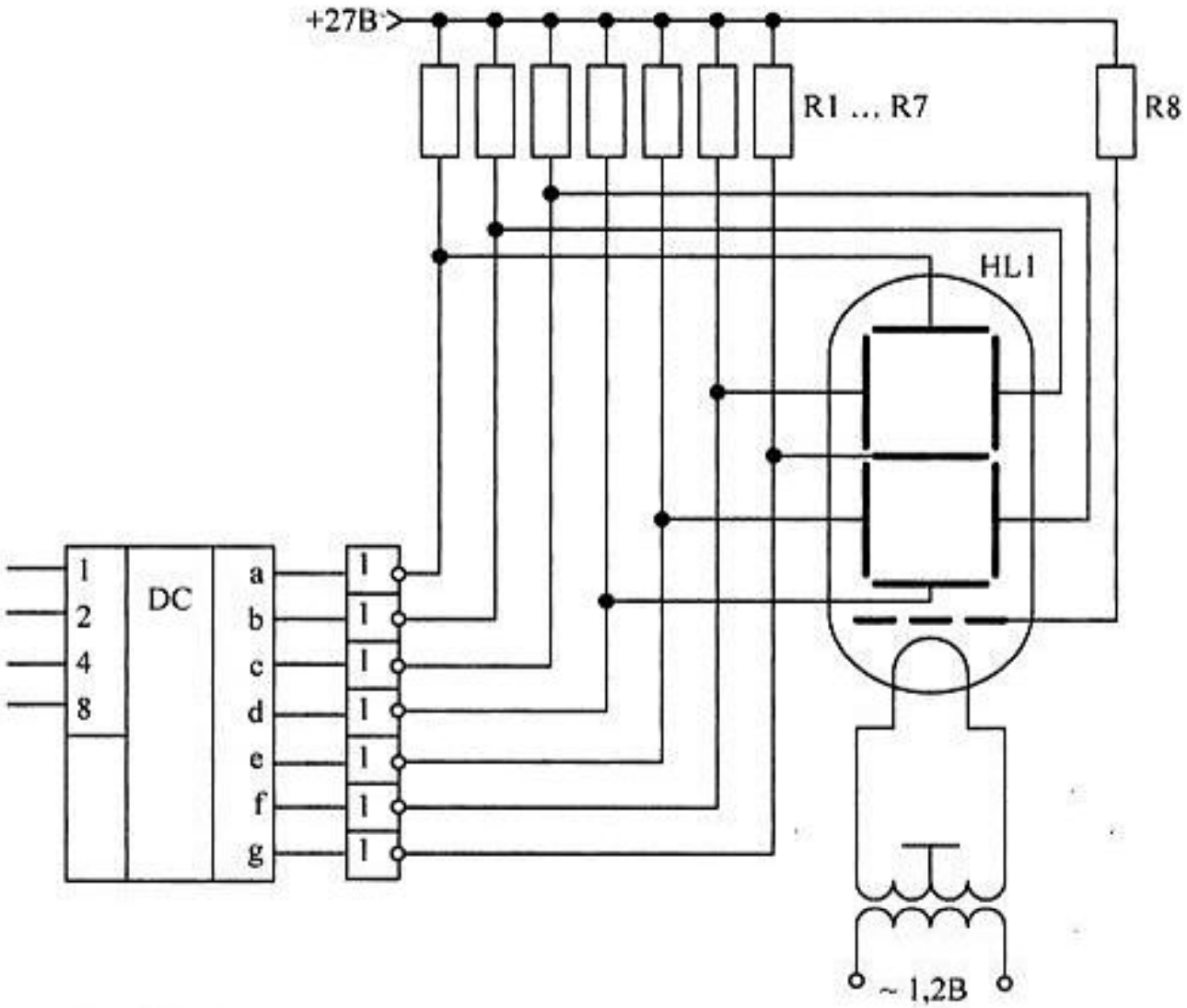


Рис. 9.13. Схема подключения семисегментного газоразрядного индикатора к дешифратору

На этой схеме в качестве ключей использована микросхема высоковольтных инверторов с открытым коллектором, выдерживающих напряжение на коллекторе до 30 В. ✧ *Обратите внимание*, что общий провод подводится к нити накала через среднюю точку трансформатора накала. Это обеспечивает равномерность свечения индикатора по всей поверхности.

В практических схемах чаще используется схема подключения газоразрядного индикатора с отрицательным напряжением питания. В этом случае дешифратор должен обеспечить вытекающий ток ключей. Подобная схема включения газоразрядного индикатора приведена на рис. 9.14.

В этой схеме транзистор VT1 и резистор R1 образуют генератор тока с большим входным и выходным сопротивлением. В результате яркость свечения

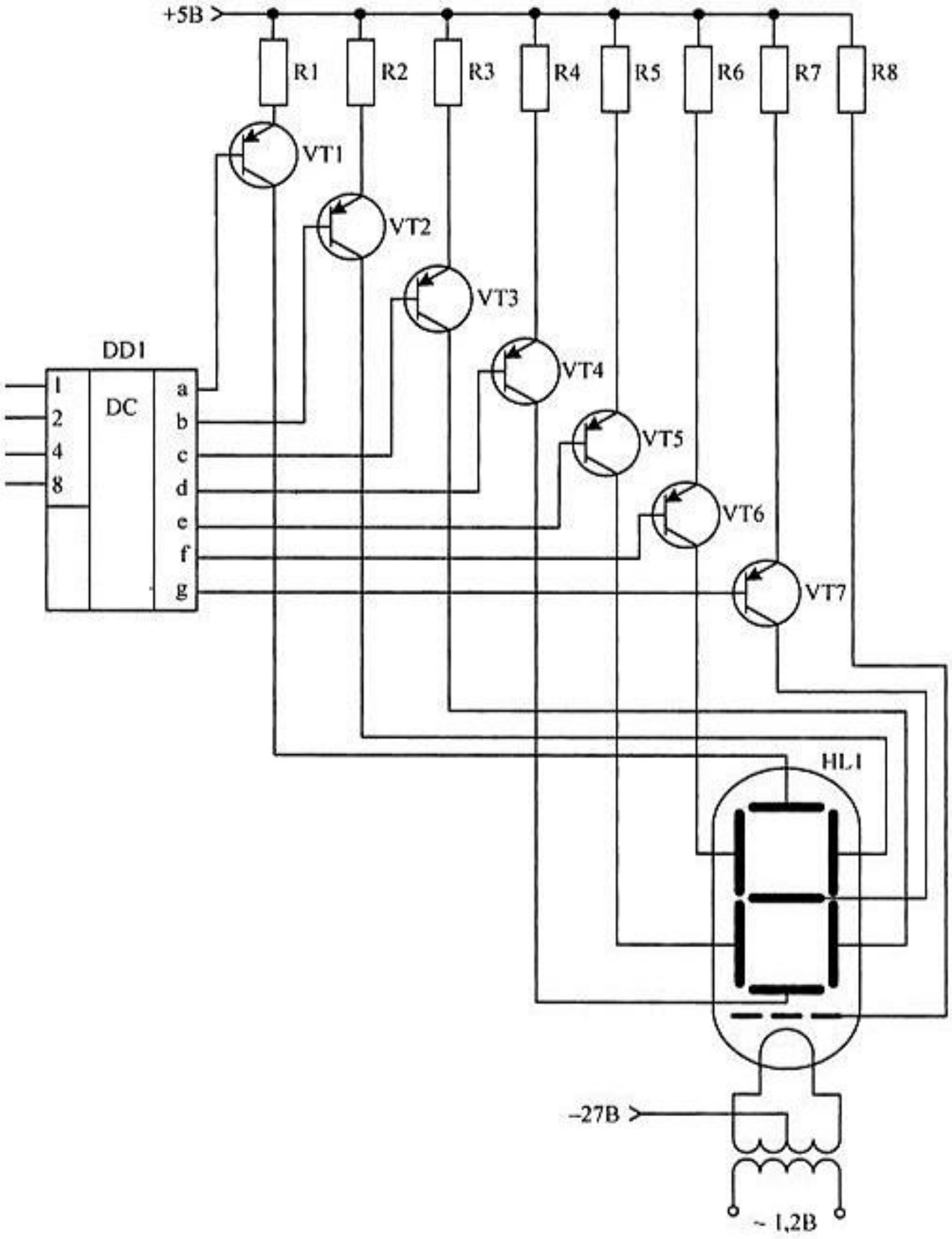


Рис. 9.14. Схема подключения семисегментного газоразрядного индикатора к дешифратору

индикатора будет слабо зависеть от напряжения питания 27 В. Зависимость тока, протекающего через сегмент индикатора, в схеме, приведенной на рис. 9.14, намного меньше по сравнению со схемой, изображенной на рис. 9.13.

Так как задача подключения газоразрядных индикаторов является распространенной, то промышленностью были разработаны и выпускаются до настоящего времени специализированные микросхемы К176ИД3, где показанные на рис. 9.14 генераторы тока входят в состав микросхемы. В результате данного схемотехнического решения выход дешифратора можно подключать к газоразрядному индикатору непосредственно.

В приведенных схемах подключения семисегментного газоразрядного индикатора управляющая сетка подключена непосредственно к питанию. Однако при создании схемы динамической индикации, которая будет рассмотрена несколько позднее, эта сетка используется для зажигания и гашения отдельных разрядов многоразрядного газоразрядного индикатора.

Светодиодные индикаторы

Газоразрядные индикаторы — это более экономичные индикаторы по сравнению с лампами накаливания, но требование высокого напряжения питания и низкая механическая прочность привели к тому, что эти индикаторы уже практически не используются при проектировании новых цифровых устройств.

В настоящее время практически в любых схемах для отображения двоичной информации используются светодиоды. Это обусловлено тем, что надежность светодиодов значительно превосходит надежность как индикаторных ламп накаливания, так и газоразрядных (неоновых) индикаторных ламп. Светодиоды труднее разбить, т. к. их корпус обычно выполнен из прозрачной пластмассы, а их вес значительно меньше веса индикаторных ламп.

Кроме того, при включении светодиодов не возникает импульсного тока значительной величины, который разрывает холодную нить накаливания индикаторных ламп своим магнитным полем. КПД светодиодов, особенно современных, значительно превосходит КПД индикаторных ламп. Основная причина повышенного КПД светодиодов — это принципиальное отсутствие теплового излучения. Электрический ток в светодиодных индикаторах непосредственно преобразуется в световое излучение.

Схемы подключения светодиодных индикаторов

Так как светодиод, так же как и газоразрядная лампа, управляется током, а не напряжением, то схема его подключения практически совпадает со схемой

подключения газоразрядной лампы. Отличается только напряжение зажигания светодиода. Схема подключения светодиодного индикатора к цифровой ТТЛ-микросхеме приведена на рис. 9.15.

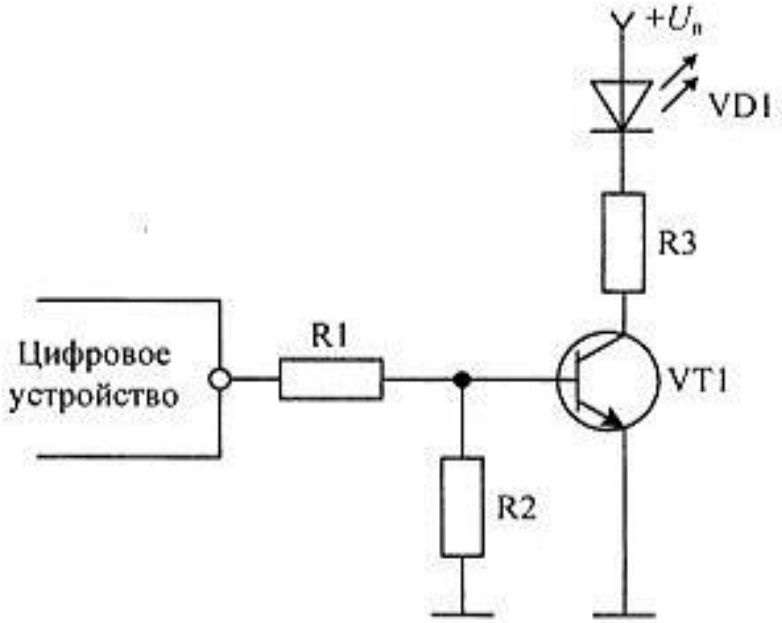


Рис. 9.15. Схема подключения светодиодного индикатора к цифровой ТТЛ-микросхеме

Расчет токоограничивающего резистора в этой схеме аналогичен расчету подобного резистора газоразрядного индикатора. Отличие заключается только в том, что падение напряжения на светодиодах лежит в пределах от 1,5 до 3 вольт (для светодиодов синего цвета свечения может достигать 4 вольт). Расчет резисторов R1 и R2 точно такой же, как и в остальных транзисторных ключах.

Теперь вспомним, что выходной ток современных цифровых микросхем превосходит минимальный ток зажигания светодиода (около 3 мА). Это означает, что в большинстве случаев можно обойтись без дополнительного транзисторного ключа для подключения светодиода. В результате принципиальная схема цифрового устройства значительно упрощается. Теперь достаточно просто ограничить ток через светодиод до допустимой величины. Такая схема подключения светодиодного индикатора к цифровой микросхеме приведена на рис. 9.16.

В схеме, приведенной на рис. 9.16, используется ток нуля цифровой микросхемы. Этот ток в большинстве цифровых схем больше тока единицы. В приведенной схеме мы не накладывали никаких ограничений на используемую цифровую микросхему, кроме того, что она должна обеспечивать необходимый выходной ток. Однако при использовании микросхемы с обычным двухтактным выходным каскадом необходимо, чтобы напряжение питания микросхемы было равно напряжению, подаваемому на светодиод.

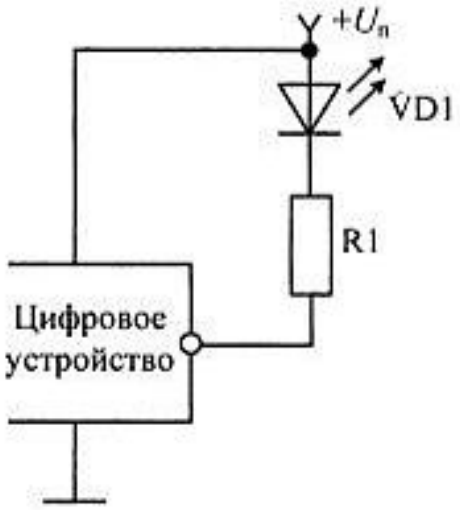


Рис. 9.16. Схема непосредственного подключения светодиодного индикатора к цифровой микросхеме

✧ *Обратите внимание*, что на цепь светодиода и резистора нужно подавать напряжение больше пяти вольт. Только в этом случае светодиод надежно откроется. Это означает, что приведенная на рис. 9.16 схема подходит только для микросхем с пятивольтовым (или более) питанием.

В большинстве современных микросхем ток единицы тоже превышает минимальный ток зажигания светодиода. В ряде случаев это может упростить принципиальную схему устройства. Схема с использованием единичного тока цифровой микросхемы приведена на рис. 9.17. Однако следует отметить, что если в схеме, приведенной на рис. 9.16, светодиод зажигается нулевым потенциалом, то в схеме, изображенной на рис. 9.17, для зажигания светодиода на выходе микросхемы следует сформировать единичный потенциал. В этой схеме напряжение питания цифровой микросхемы тоже должно превышать пять вольт.

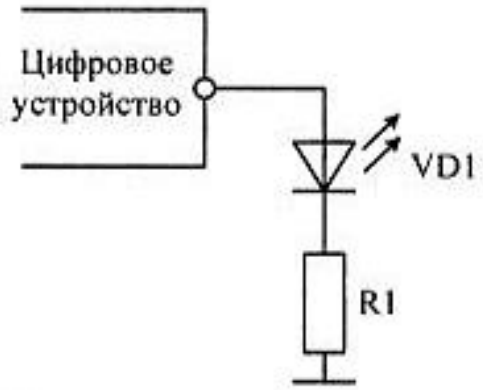


Рис. 9.17. Использование тока единицы микросхемы для зажигания светодиодного индикатора

Как уже говорилось ранее, в современных цифровых микросхемах часто используется напряжение питания 3.3, 2.5 или даже 0.7 В. Как же быть в таком

случае? Неужели использовать схему с дополнительным транзисторным ключом? Во всех цифровых сериях микросхем присутствуют логические элементы с открытым коллектором. Выходной транзистор этих микросхем способен выдерживать напряжение, превышающее напряжение питания самой микросхемы, поэтому микросхемы с открытым коллектором можно использовать для подключения светодиодных индикаторов. Подобная схема подключения светодиодного индикатора приведена на рис. 9.18.

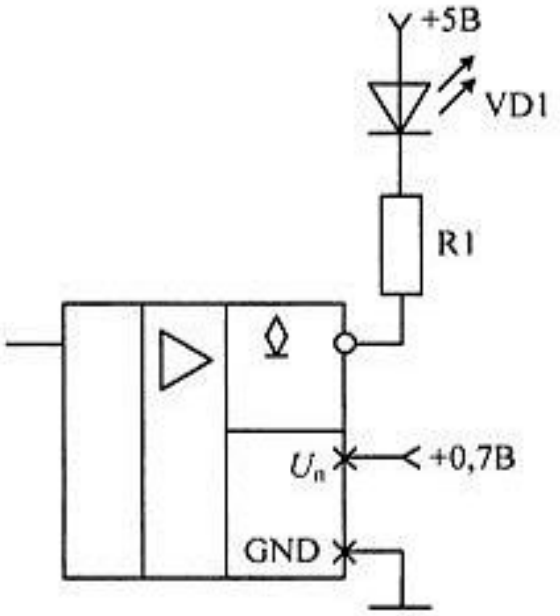


Рис. 9.18. Схема подключения светодиодного индикатора к цифровой микросхеме с открытым коллектором

Виды светодиодных индикаторов

Светодиодные индикаторы могут светиться различным цветом. Обычно используются светодиоды с зеленым, красным, синим или желтым цветом. Яркость светодиодов достигла такого значения, что их уже можно использовать в качестве источника подсветки и даже освещения. Конструктивное исполнение светодиодов тоже может различаться. Это могут быть круглые, прямоугольные или квадратные светодиоды. Есть светодиоды, предназначенные для поверхностного монтажа. Их габариты настолько малы, что эти светодиоды можно считать точечными источниками света.

Если в одном корпусе светодиода расположены несколько кристаллов с разным цветом свечения, то можно переключать цвет свечения светодиода, подавая ток на разные выводы этого электронного прибора. Благодаря конструкции светопровода таких индикаторов будет казаться, что изменяется цвет свечения одного и того же индикатора.

Подобным же образом устроены и семисегментные светодиоды. Однако цвет свечения светодиодов в них одинаковый. В этих индикаторах форма светодиодов подобрана таким образом, чтобы можно было с помощью них отображать различные цифры. В подобных индикаторах светодиоды называются сегментами. Внешний вид семисегментного индикатора рассматривался нами ранее и приведен на рис. 9.3.

В семисегментных светодиодных индикаторах для экономии количества выводов корпуса объединяются аноды или катоды светодиодов. В зависимости от этого обстоятельства будет использоваться схема включения отдельного сегмента индикатора, приведенная на рис. 9.17 или 9.18. В качестве примера схемы соединения светодиодов в семисегментном индикаторе на рис. 9.19 приведена схема индикатора BS-C506RII.

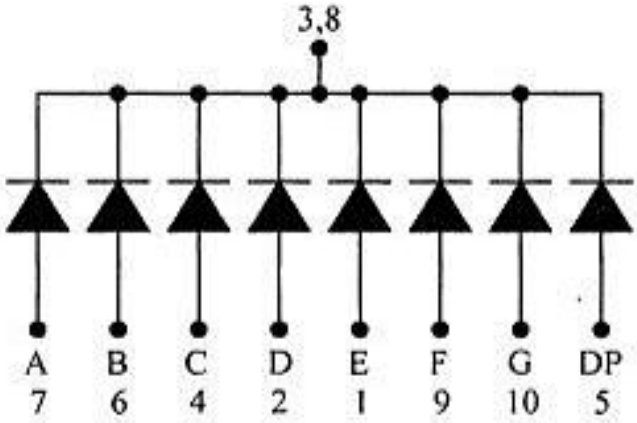


Рис. 9.19. Схема семисегментного светодиодного индикатора BS-C506RII

В матричных индикаторах светодиоды включаются в точках пересечения колонок и строк. При использовании матричных индикаторов обычно применяется динамическая индикация, принципы работы которой будут рассмотрены немного позже. В качестве примера подобного электронного прибора на рис. 9.20 приведена схема матричного светодиодного индикатора BM-20657MD.

Матричные индикаторы позволяют отображать любые символы. Для этого достаточно просто нарисовать требуемый символ, как это, например, показано на рис. 9.21.

На рис. 9.21 черным цветом показаны горящие светодиоды, а белым цветом — погашенные. Из этого рисунка совершенно ясно, как получить изображение любого требующегося символа, поэтому давайте перейдем к следующему разделу и рассмотрим подробнее, как осуществляется динамическая индикация.

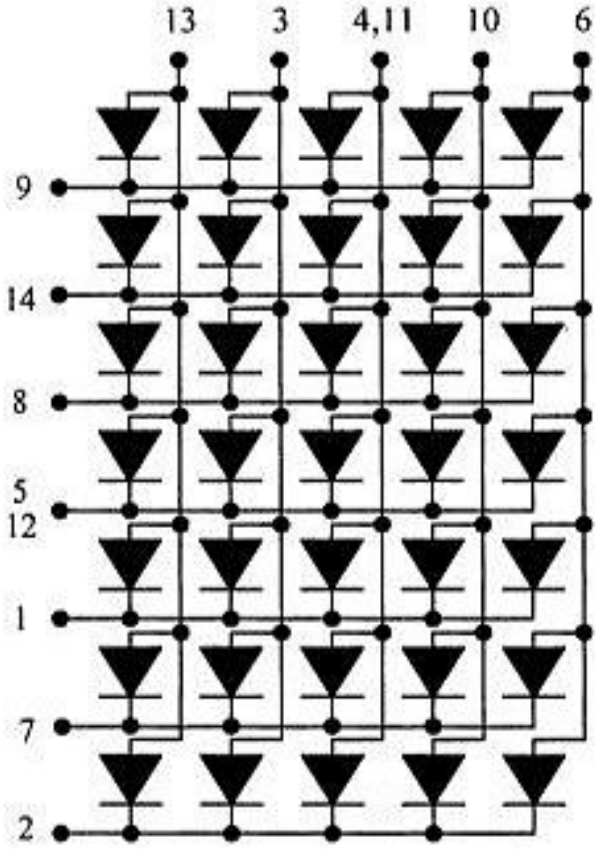


Рис. 9.20. Схема матричного светодиодного индикатора BM-20657MD

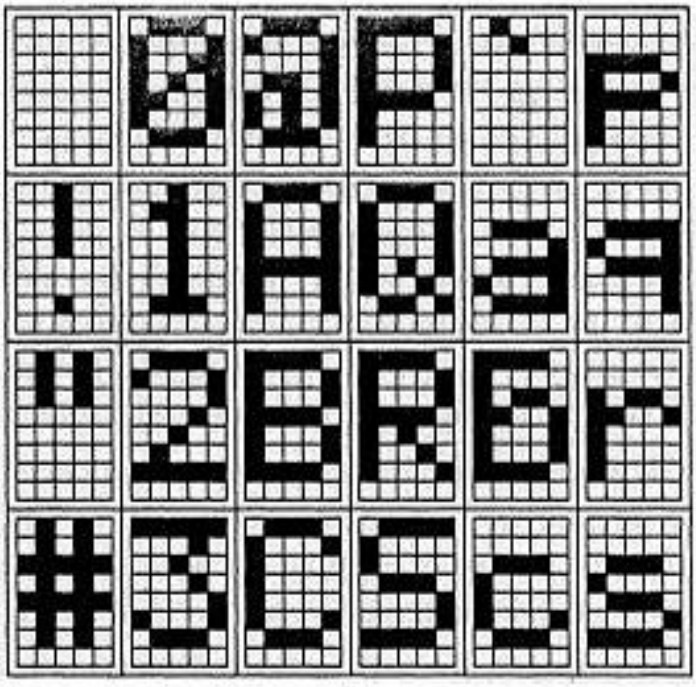


Рис. 9.21. Отображение символов на матричных индикаторах

Динамическая индикация

Индикаторы обычно располагают в местах, удобных для просмотра информации, отображаемой на них. Остальная цифровая схема может располагаться на других печатных платах. При увеличении количества индикаторов увеличивается количество проводников между платой индикаторов и остальной цифровой схемой. Это приводит к определенным неудобствам разработки конструкции и эксплуатации аппаратуры. Эта же причина приводит к увеличению ее стоимости.

Количество соединительных проводников можно уменьшить, если заставить индикаторы работать в импульсном режиме. Человеческий глаз обладает инерционностью, и если заставить индикаторы отображать информацию поочередно с достаточно большой скоростью, то человеку будет казаться, что все индикаторы отображают свою информацию непрерывно. В результате можно по одним и тем же проводникам поочередно передавать отображаемую информацию различных разрядов. Обычно для того, чтобы человек не воспринимал мерцание отображаемой буквенно-символьной информации, достаточно частоты обновления 50 Гц, но лучше увеличить эту частоту до 100 Гц.

Давайте рассмотрим структурную схему включения семисегментных светодиодных индикаторов, приведенную на рис. 9.22. Эта схема может обеспечить динамическую индикацию отображаемой цифровой информации.

В схеме, приведенной на рис. 9.22, отображаются четыре цифровых разряда. Каждый разряд кратковременно подключается к своему входу коммутатора. Генератор служит для задания скорости обновления информации на индикаторах. Двоичный счетчик последовательно формирует четыре состояния схемы, а дешифратор через ключи обеспечивает поочередную подачу питания на семисегментные индикаторы. В результате, когда коммутатор подает двоично-десятичный код с входа А на входы семисегментного дешифратора, то этот код отображается на индикаторе HL1. Когда коммутатор подает на входы семисегментного дешифратора двоично-десятичный код со входа В, то этот код отображается на индикаторе HL2, и т. д., по циклу.

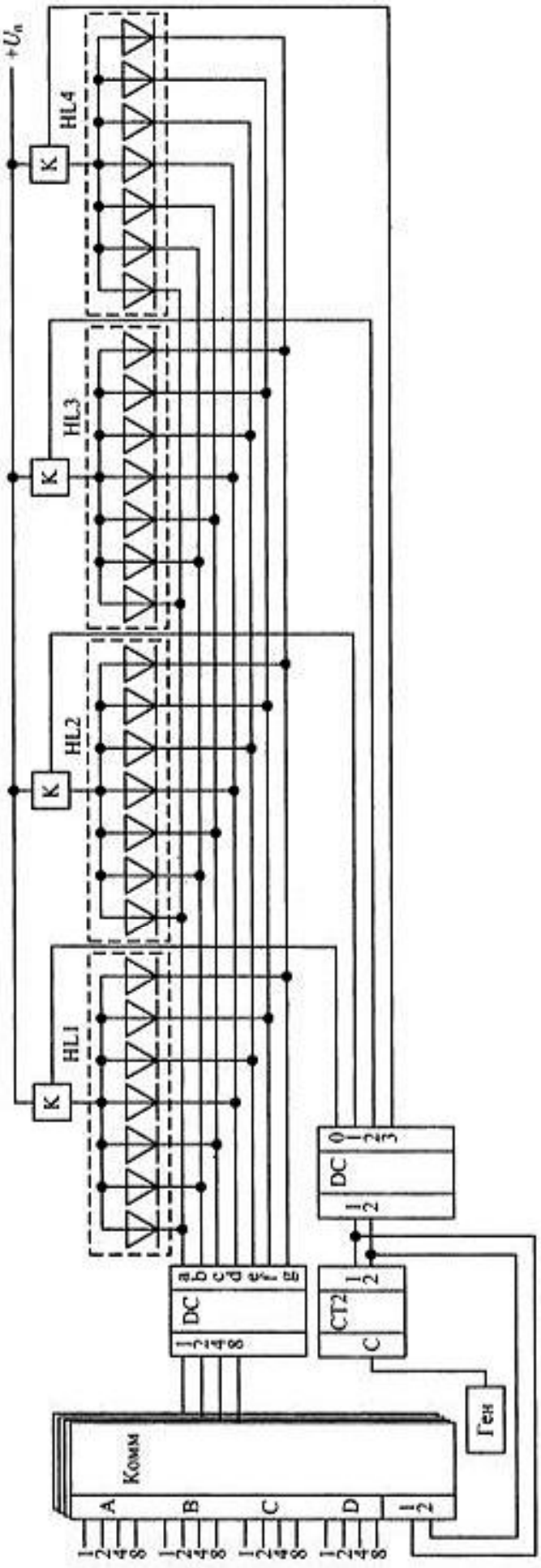
Скорость обновления информации в рассмотренной схеме будет в четыре раза меньше частоты тактового генератора "Ген". То есть для того, чтобы получить частоту мерцания индикаторов 100 Гц, требуется, чтобы генератор формировал сигнал с частотой 400 Гц.

Во сколько же раз мы в результате уменьшили количество соединительных проводников? Это зависит от того, где мы проведем сечение схемы. Если мы на плате индикации оставим только индикаторы, то для их работы потребуется семь информационных сигналов для сегментов и четыре коммутирующих сигнала. Всего одиннадцать проводников. В статической схеме индикации нам потребовалось бы $7 \times 4 = 28$ проводников. Как видим, выигрыш налицо. При реализации 8-разрядного блока индикации выигрыш в количестве соединительных проводников будет еще больше.

Еще больший выигрыш будет, если сечение схемы провести по входам дешифраторов. В этом случае для четырехразрядного блока индикации потребуется только шесть сигнальных проводников и два проводника питания схемы. Однако такая точка сечения схемы динамической индикации применяется редко.

Теперь давайте рассчитаем ток, протекающий через каждый сегмент светодиодного индикатора при его свечении. Для этого воспользуемся эквивалентной схемой протекания тока по одному из сегментов светодиодного индикатора. Данная схема приведена на рис. 9.23.

Как уже упоминалось ранее, для нормальной работы светодиода требуется ток от 3 до 10 мА. Зададимся минимальным током светодиода 3 мА, однако при импульсном режиме работы яркость свечения индикатора падает в N раз, где коэффициент N равен скважности импульсов тока, подаваемых на этот индикатор. Значение коэффициента N совпадает с количеством разрядов динамического индикатора.



Рис, 9.22. Структурная схема динамической индикации

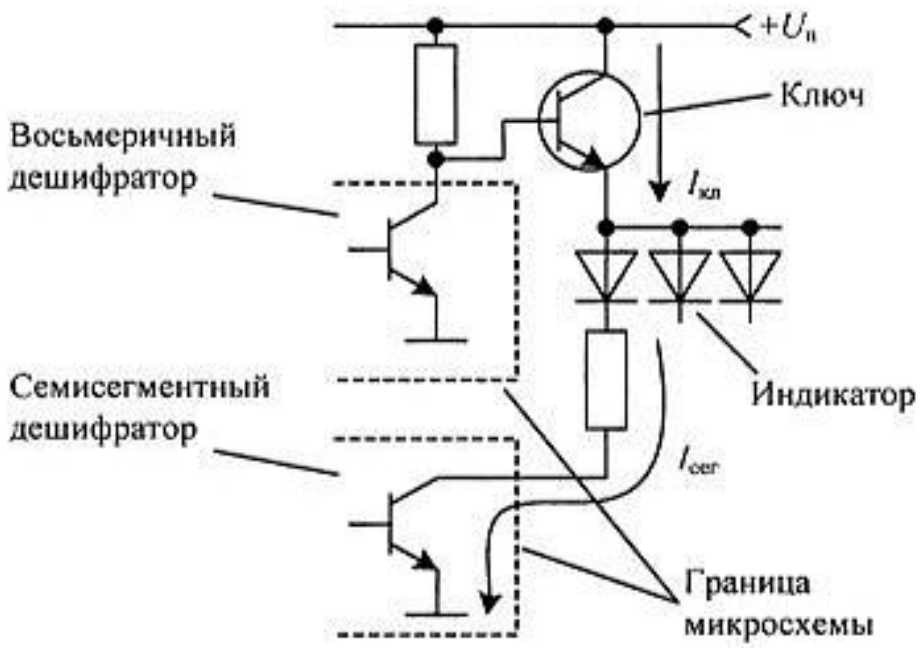


Рис. 9.23. Эквивалентная схема одного сегмента

Если мы собираемся сохранить ту же яркость свечения индикатора, то требуется увеличить величину импульсного тока, протекающего через сегмент, в N раз. Для восьмиразрядного индикатора коэффициент N равен восьми. Пусть первоначально мы выбрали статический ток через светодиод, равный 3 мА. Тогда для сохранения той же яркости свечения светодиода в восьмиразрядном индикаторе потребуется импульсный ток:

$$I_{сег\ дин} = I_{сег\ ст} \times N = 3\ мА \times 8 = 24\ мА$$

Такой ток с трудом смогут обеспечить только некоторые серии цифровых микросхем. Для большинства же серий микросхем потребуются усилители, выполненные на транзисторных ключах.

Теперь определим ток, который будет протекать через ключ, коммутирующий питание на отдельные разряды восьмиразрядного блока индикации. Как это видно из схемы, приведенной на рис. 9.22, через ключ может протекать ток любого сегмента индикатора. При отображении цифры "8" потребуется зажечь все семь сегментов индикатора, значит импульсный ток, протекающий в этот момент через ключ, можно определить следующим образом:

$$I_{кл} = I_{сег\ дин} \times N_{сег} = 24\ мА \times 7 = 168\ мА$$

В радиолюбительских схемах часто встречаются решения, где коммутирующий ток берется непосредственно с выхода дешифратора, который не может выдать ток больше 20 мА. Возникает вопрос — где смотреть такой индикатор? В полной темноте? Получается "прибор ночного видения", т. е. прибор, показания которого видны только в полной темноте.

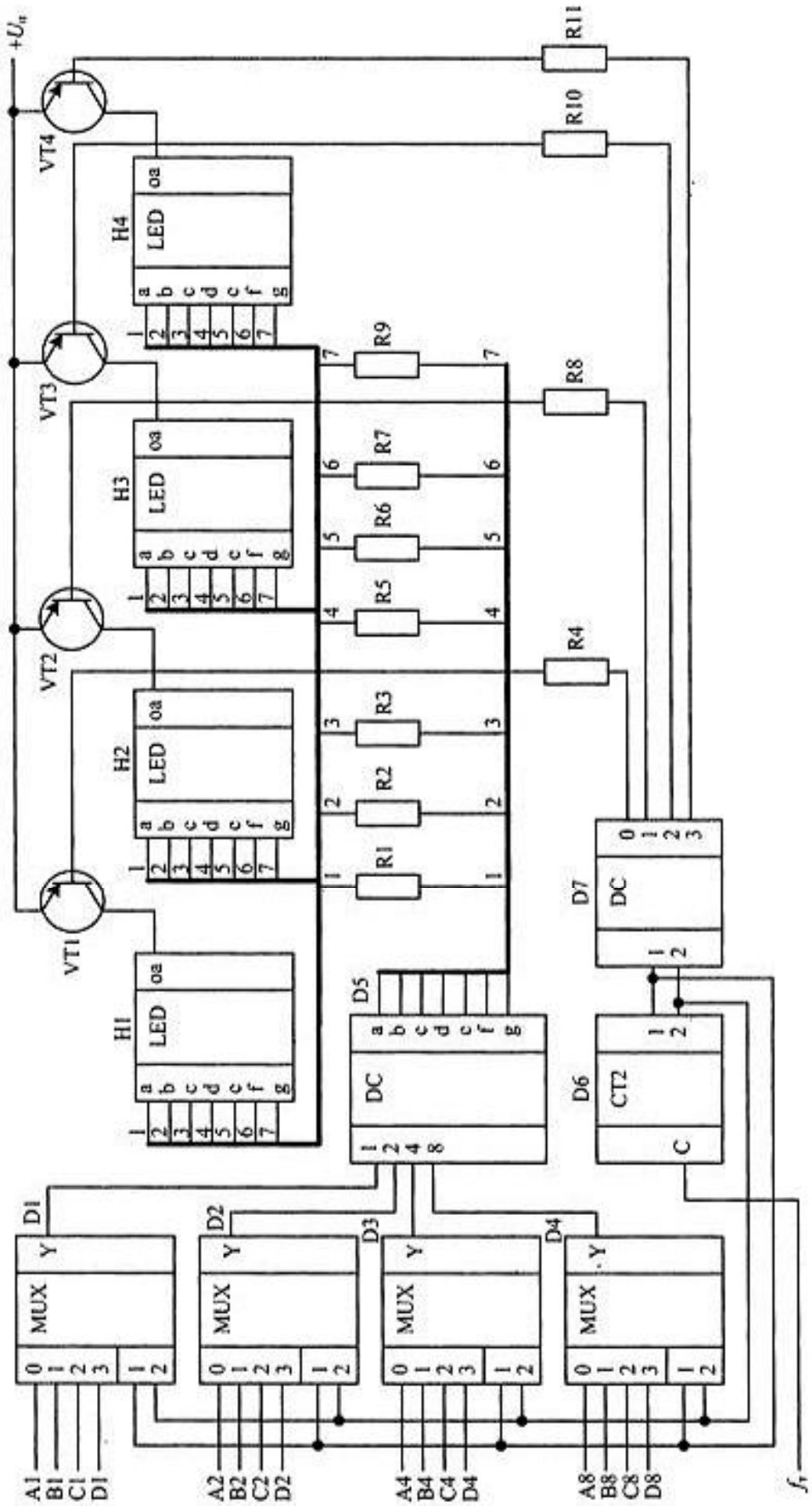


Рис. 9.24. Принципиальная схема блока динамической индикации

А теперь давайте рассмотрим принципиальную схему разработанного в результате расчета блока индикации. Она приведена на рис. 9.24.

Теперь, после того как мы разработали схему динамической индикации, можно обсудить ее достоинства и недостатки. Несомненным достоинством динамической индикации является малое количество соединительных проводов, что делает ее незаменимой в некоторых случаях, таких как работа с матричными индикаторами.

В качестве недостатка следует привести наличие больших импульсных токов, а т. к. любой проводник является антенной, то динамическая индикация служит мощным источником помех.

✧ *Обратим внимание*, что фронт у коммутирующих импульсов очень короткий, поэтому его гармонические составляющие перекрывают диапазон радиочастот вплоть до ультракоротких волн.

Итак, применение динамической индикации позволяет минимизировать количество соединительных проводов между цифровым устройством и индикатором, но является при этом мощным источником помех, поэтому ее применение в радиоприемных устройствах нежелательно.

Если по каким-либо причинам, например, необходимость применения матричных индикаторов, приходится использовать динамическую индикацию, то нужно принять все возможные конструктивные меры по подавлению возникающих в процессе ее работы помех.

В качестве мер по подавлению помех от динамической индикации можно назвать экранирование блока, соединительного кабеля и печатных плат. Кроме того, следует использовать соединительные провода с минимальной длиной, применять фильтры по питанию. При экранировании блока, возможно, потребуется экранировать и сами индикаторы. При этом для экранирования помех обычно используется металлическая сетка. Эта сетка одновременно может увеличить контрастность отображаемых символов.

Жидкокристаллические индикаторы

В настоящее время жидкокристаллические индикаторы являются наиболее распространенным видом индикаторов (если не требуется работа при отрицательных температурах). Хотя сами жидкие кристаллы (ЖК) были известны химикам еще с 1888 г., но только с 1960-х годов началось их практическое использование. В 1990 г. Де Жен получил Нобелевскую премию за теорию жидких молекулярных кристаллов.

Принципы работы жидкокристаллических индикаторов

Термином "жидкий кристалл" обозначается мезофаза между твердым состоянием и изотропным жидким состоянием вещества, при этом мезофаза сохраняет фундаментальные свойства, присущие двум состояниям материи. Жидкие кристаллы, с одной стороны, обладают текучестью как изотропная жидкость, с другой стороны, сохраняют определенный порядок в расположении молекул (как кристалл).

В отдельных случаях мезофаза оказывается стабильной в широком диапазоне температур, включая комнатную температуру. В этом случае говорят о жидких кристаллах. Большинство известных жидких кристаллов образуются стержневыми молекулами.

Обычно жидкокристаллический дисплей представляет собой стеклянную кювету толщиной менее 20 мкм, в которую помещен жидкий кристалл. Направление молекул жидкого кристалла может быть задано обработкой поверхностей кюветы таким образом, чтобы молекулы жидкого кристалла выстраивались в направлении, параллельном плоскости кюветы или перпендикулярно к ней. Один из способов обработки поверхности экрана заключается в нанесении на него тонкого слоя твердого полимера и последующего "натирания" его в одном направлении.

Используя различные ориентации направления молекул жидкого кристалла первоначально с помощью поверхностного упорядочения, а затем с помощью электрического поля, можно сконструировать простейший дисплей. Жидкокристаллический дисплей состоит из нескольких слоев, где ключевую роль играют две стеклянные панели, между которыми помещен жидкий кристалл.

На панели наносятся бороздки. Бороздки получаются в результате размещения на стеклянной поверхности тонких пленок из прозрачного пластика, который затем специальным образом обрабатывается. Бороздки расположены таким образом, что они параллельны на каждой панели, но перпендикулярны бороздкам соседней панели. Соприкасаясь с бороздками, молекулы в жидких кристаллах ориентируются одинаково по всей поверхности. В результате направление ориентации молекул жидкого кристалла поворачивается от верхней панели к нижней на 90°, вращая, таким образом, плоскость поляризации света, как это изображено на рис. 9.25. Изображение формируется при помощи поляризационных пленок, размещенных над и под жидкокристаллическим дисплеем. Если оси поляризации этих пленок перпендикулярны друг другу, то дисплей будет прозрачным.

На стеклянные панели наносится тонкий слой металла, образующий электроды. Если теперь к электродам подвести напряжение, то молекулы жидкого

кристалла развернутся вдоль электрического поля, вращение плоскости поляризации исчезнет, и свет не сможет пройти через поляризационные пленки, как это показано на рис. 9.26. Рис. 9.26 соответствует отсутствию электрического поля, а рис. 9.27 — приложенному к электродам напряжению.

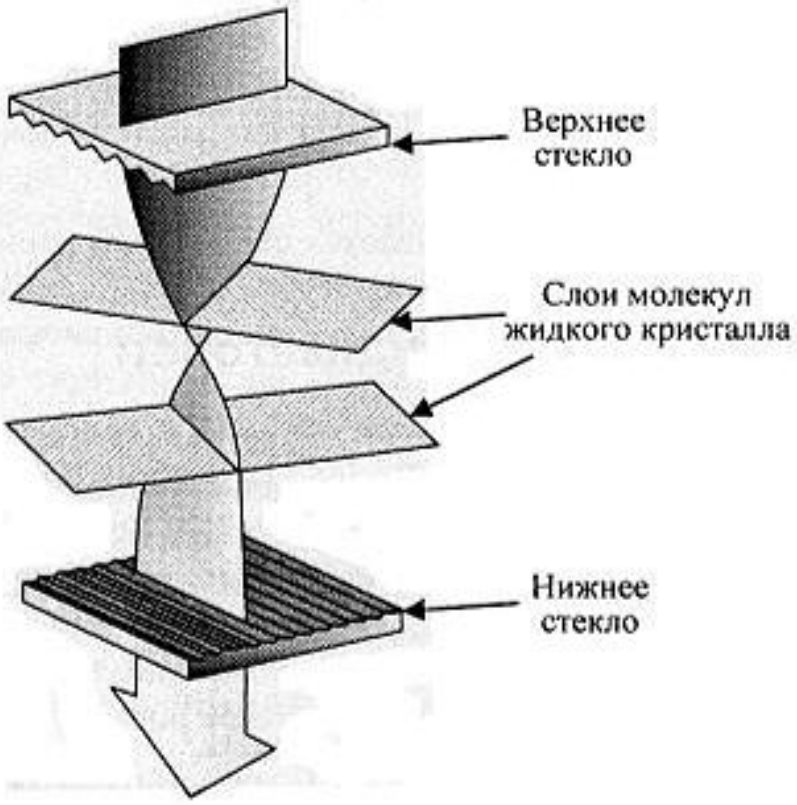


Рис. 9.25. Вращение поляризации света жидким кристаллом

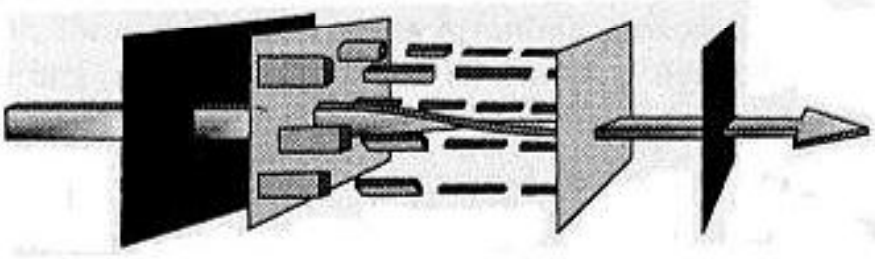


Рис. 9.26. Прохождение света через жидкий кристалл при отсутствии электрического поля

Напряжение, необходимое для поворота директора, составляет обычно от 2 до 5 В. Важно, что действие электрического поля не связано с дипольным моментом молекулы и поэтому не зависит от направления поля. Это позволяет использовать для управления индикатором переменное поле. Постоянное поле может приводить к электролизу жидкого кристалла и, в конечном итоге, выходу прибора из строя.

Электроды на жидкокристаллический индикатор наносятся в виде точек, пиктограмм или сегментов для отображения различных видов информации, как это уже обсуждалось ранее.

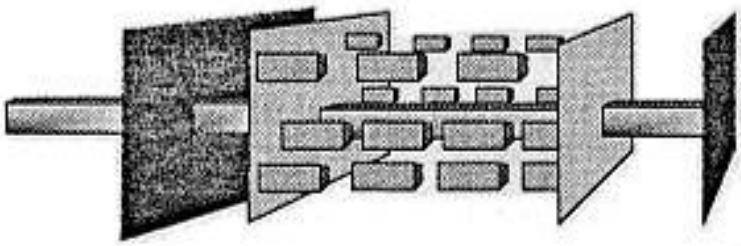


Рис. 9.27. Прохождение света через жидкий кристалл при воздействии на него электрического поля

Режимы работы жидкокристаллических индикаторов

Жидкокристаллические индикаторы используются в двух режимах работы: в режиме отражения света и в режиме просвечивания. Наиболее экономичный режим использования жидкокристаллического индикатора — это режим отражения. В этом режиме используются внешние источники света, такие как солнце или осветительные лампы помещения. Сами жидкокристаллические индикаторы в процессе отображения информации ток практически не потребляют.

При использовании режима отражения прозрачным оставляют весь дисплей. Информация же формируется непрозрачными участками жидкого кристалла, образующимися между электродами при подаче на них переменного напряжения.

В режиме просвечивания возможны два вида использования жидкокристаллического дисплея: формирование обычного изображения, как и в режиме отражения, и формирование негативного изображения. В режиме негативного изображения весь дисплей остается непрозрачным, а свет проходит только через участки изображения, которые в этом случае кажутся нарисованными краской. Негативный режим работы дисплея формируется поляризационными пленками с совпадающей поляризацией, т. е. для смены режима отображения информации достаточно повернуть одну из поляризационных пленок на 90°.

Для подсветки жидкокристаллического индикатора обычно используется газоразрядные лампы или светодиоды, т. к. эти источники света не выделяют тепла, способного вывести из строя жидкокристаллический индикатор. Для равномерного распределения света под жидкокристаллическими индикаторами располагаются специальные светопроводы, выполненные из рассеивающих свет материалов.

Параметры жидкокристаллических индикаторов

Важным параметром индикатора является время релаксации — время, необходимое для возвращения молекул жидкого кристалла в исходное состояние после отключения электрического поля. Оно определяется поворотом молекул и составляет 30 ... 50 мс. Такое время достаточно для работы различных индикаторов, но на несколько порядков превышает время, необходимое для работы компьютерного монитора.

Время релаксации сильно зависит от температуры жидкокристаллического индикатора. Именно временем релаксации определяется минимальная температура работы жидкокристаллических индикаторов. Время релаксации современных жидкокристаллических индикаторов при температуре –25 °С достигает нескольких секунд. Такое время смены информации неприемлемо для большинства практических приложений.

Не менее важным параметром жидкокристаллического индикатора является контрастность изображения. При нормальной температуре контрастность изображения достигает нескольких сотен. При повышении температуры контрастность изображения падает и при температуре порядка +50 °С изображение становится практически неразличимым.

Следующий параметр, характеризующий жидкокристаллический индикатор, — это угол обзора изображения. Угол обзора жидкокристаллического индикатора существенно зависит от скважности динамического режима индикации. Чем больше скважность — тем меньше получается угол обзора индикатора.

В современных жидкокристаллических компьютерных мониторах используется специальный метод формирования статического изображения при динамическом способе его подачи на дисплей. Это TFT-технология (тонкопленочная технология). При ее использовании около каждого элемента изображения, непосредственно на стекле, формируется запоминающий конденсатор и ключевой транзистор, который подключает этот конденсатор к цепям формирования изображения только в момент подачи информации именно для этого элемента изображения.

Так как толщина этих элементов делается незначительной, то они остаются прозрачными для видимого света, а, следовательно, остаются незаметными для конечного пользователя.

Формирование цветного изображения на жидкокристаллических индикаторах

Цветные жидкокристаллические индикаторы обычно выполняют в виде матрицы. При достаточно малых элементах этой матрицы из них можно сформи-

ровать как цифровые, так и буквенные символы. Более того, из них могут формироваться произвольные изображения.

Для реализации цветного изображения обычно используется режим просвечивания жидкокристаллического индикатора. Этот режим позволяет получать большую яркость изображения. Для формирования цветного изображения на жидкокристаллических индикаторах используются точно такие же принципы, как и в телевидении.

Каждый элемент изображения (пиксел) выполняется в виде трех близко расположенных элементов. Напротив каждого из этих элементов располагается свой светофильтр: синий, красный и зеленый. Этот фильтр наносится непосредственно на стекло индикатора методами фотолитографии. Так как размеры пикселов в жидкокристаллическом индикаторе выполняются очень маленькими, то человеческий глаз не может различить отдельные его элементы. В результате свет от каждого элемента пиксела суммируется, и формируется результирующий цвет всего пиксела в целом.

В простейшем случае при помощи трех бит, подаваемых на один элемент изображения, можно сформировать восемь различных цветов этого элемента. При затемнении всех трех элементов пиксела его цвет становится черным. В режиме прозрачности всех трех составляющих пиксела изображения его цвет становится белым.

Формирование управляющего напряжения для жидкокристаллического индикатора

Особенностью работы жидкокристаллического индикатора является то, что на него следует подавать переменное напряжение. Это связано с тем, что при подаче на жидкокристаллический индикатор постоянного напряжения происходит электролиз жидкого кристалла и индикатор выходит из строя.

Напряжение для работы жидкокристаллического индикатора формируется логическими элементами, поэтому для формирования изображения обычно используется прямоугольное колебание со скважностью равной двум. Его легко можно получить на выходе делителя частоты на два.

Теперь вспомним, что логические сигналы содержат постоянную составляющую. Ее можно убрать, подав сигнал на выводы жидкокристаллической ячейки в противофазе друг другу. Временная диаграмма такого напряжения приведена на рис. 9.28.

В каждый момент времени напряжение между электродами заставляет молекулы выстраиваться вдоль силовых линий электрического поля. То, что напряжение постоянно меняет знак, приводит к тому, что молекула движется то

к одному, то к другому электроду, оставаясь в среднем на одном и том же месте. В результате электролиза жидкого кристалла не происходит.

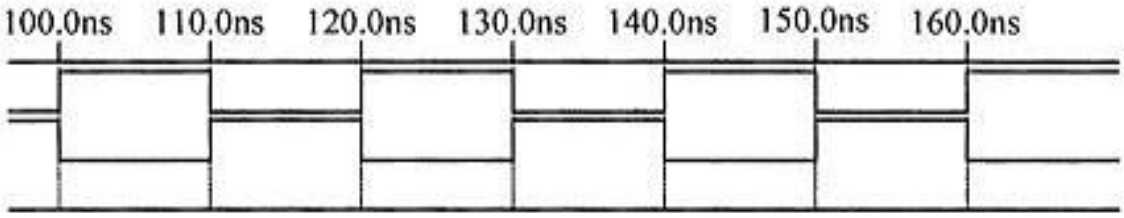


Рис. 9.28. Временная диаграмма напряжения на выводах жидкокристаллической ячейки

Если ячейку жидкокристаллического индикатора следует оставить прозрачной, то на ее выводы подаются синфазные напряжения. В этом случае разность потенциалов между электродами жидкокристаллического индикатора получается равной нулю. Молекулы жидкого кристалла остаются завернутыми в спираль. Свет при прохождении через него поворачивается на 90° и свободно проходит через обе поляризационные пленки.

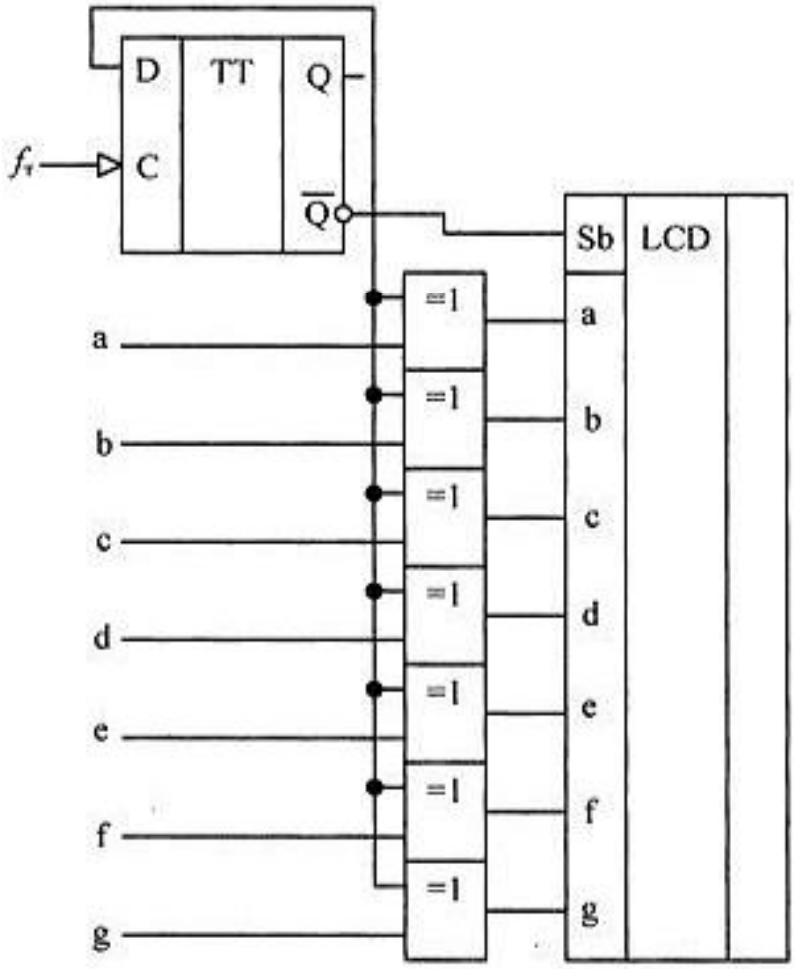


Рис. 9.29. Принципиальная схема контроллера семисегментного жидкокристаллического индикатора

При применении описанного принципа работы контроллер жидкокристаллического индикатора можно собрать с использованием логических элементов "исключающее ИЛИ". Подобная схема контроллера жидкокристаллического индикатора приведена на рис. 9.29.

В этой схеме скважность прямоугольного колебания, равную двум, обеспечивает делитель частоты, собранный на D-триггере. Если на управляющий вход сегмента контроллера подается единичный потенциал, то логический элемент "исключающее ИЛИ" инвертирует колебание, подаваемое на подложку жидкокристаллического индикатора Sb (см. таблицу истинности логического элемента "исключающее ИЛИ"). Напряжение между сегментом и подложкой становится противофазным и этот сегмент становится непрозрачным.

Если же на управляющий вход сегмента контроллера ЖК-индикатора поступает нулевой потенциал, то на выходе логического элемента "исключающее ИЛИ" колебание не инвертируется. Тем самым между соответствующим сегментом индикатора и подложкой сохраняется нулевая разность потенциалов. Этот сегмент остается прозрачным.

Особенности динамической индикации в жидкокристаллических индикаторах

Обычно динамическая индикация в жидкокристаллических индикаторах осуществляется по строкам. Пример схемы расположения электродов в индикаторах, предназначенных для динамической индикации, приведен на рис. 9.30.

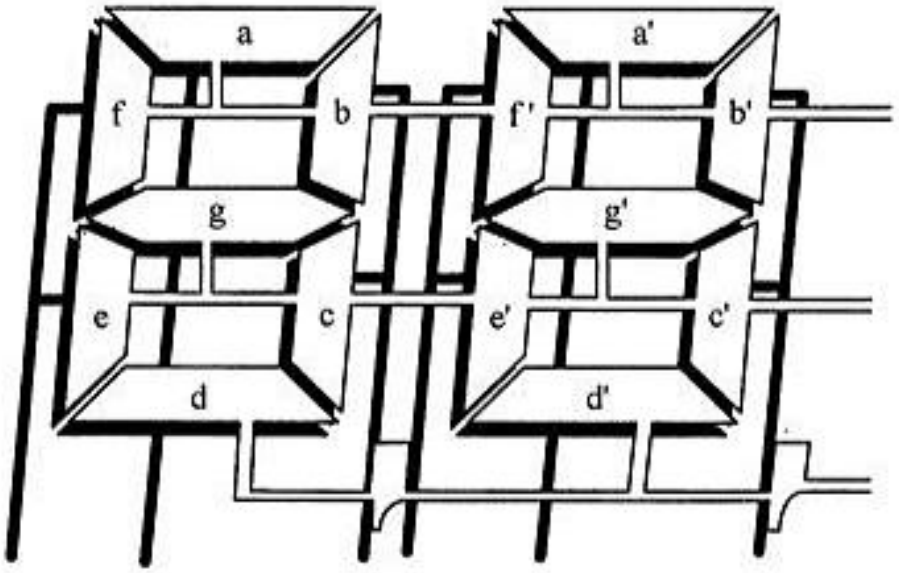


Рис. 9.30. Пример схемы расположения электродов в жидкокристаллическом индикаторе

В этой схеме использовано три строки. Именно по ним осуществляется динамическая индикация. Подложка тоже не выполнена в виде одной плоскости. Здесь на электроды подложки (показаны черным цветом) подаются информационные сигналы. В простейшем случае для формирования сигналов динамической индикации потребуются трехуровневые напряжения. Для формирования третьего уровня напряжения можно воспользоваться резистивным делителем. Этот делитель позволяет получить ровно половину питания цифровой схемы. Временные диаграммы сигналов, подаваемых на строки жидкокристаллического индикатора, приведены на рис. 9.31.

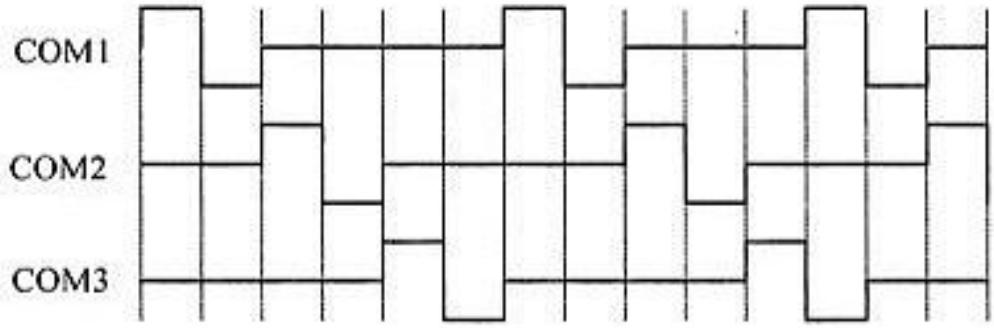


Рис. 9.31. Временные диаграммы напряжения на выводах строк жидкокристаллической ячейки

Теперь для того, чтобы выбрать или нет соответствующий сегмент, на его колонку следует подать сигнал в фазе или в противофазе к сигналу, действующему на соответствующей строке.

Например, если требуется отобразить на индикаторе сегменты "a" и "d", а сегмент "g" оставить прозрачным, то на вторую колонку приведенного на рис. 9.30 жидкокристаллического индикатора следует подать сигнал с формой, показанной на рис. 9.32.

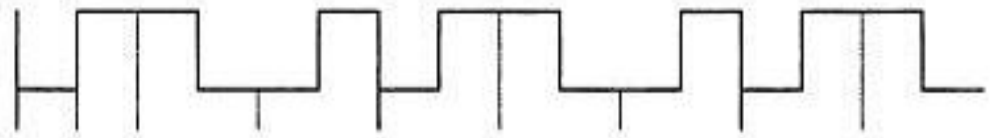


Рис. 9.32. Временная диаграмма напряжения на втором столбце жидкокристаллической ячейки

На приведенной в этом рисунке временной диаграмме в первой части кадра сигнал противоположен сигналу выборки первой строки. Во второй части кадра он синфазен сигналу выборки второй строки, а в последней части кадра он снова противоположен сигналу выборки третьей строки.

В результате взаимодействия сигналов COM1, COM2, COM3 и SEG0 на сегменты "a" и "d" подается возбуждающее напряжение, а на сегмент "g" напряжение не поступает. Схема, реализующая динамическую индикацию по описанному принципу, приведена на рис. 9.33.

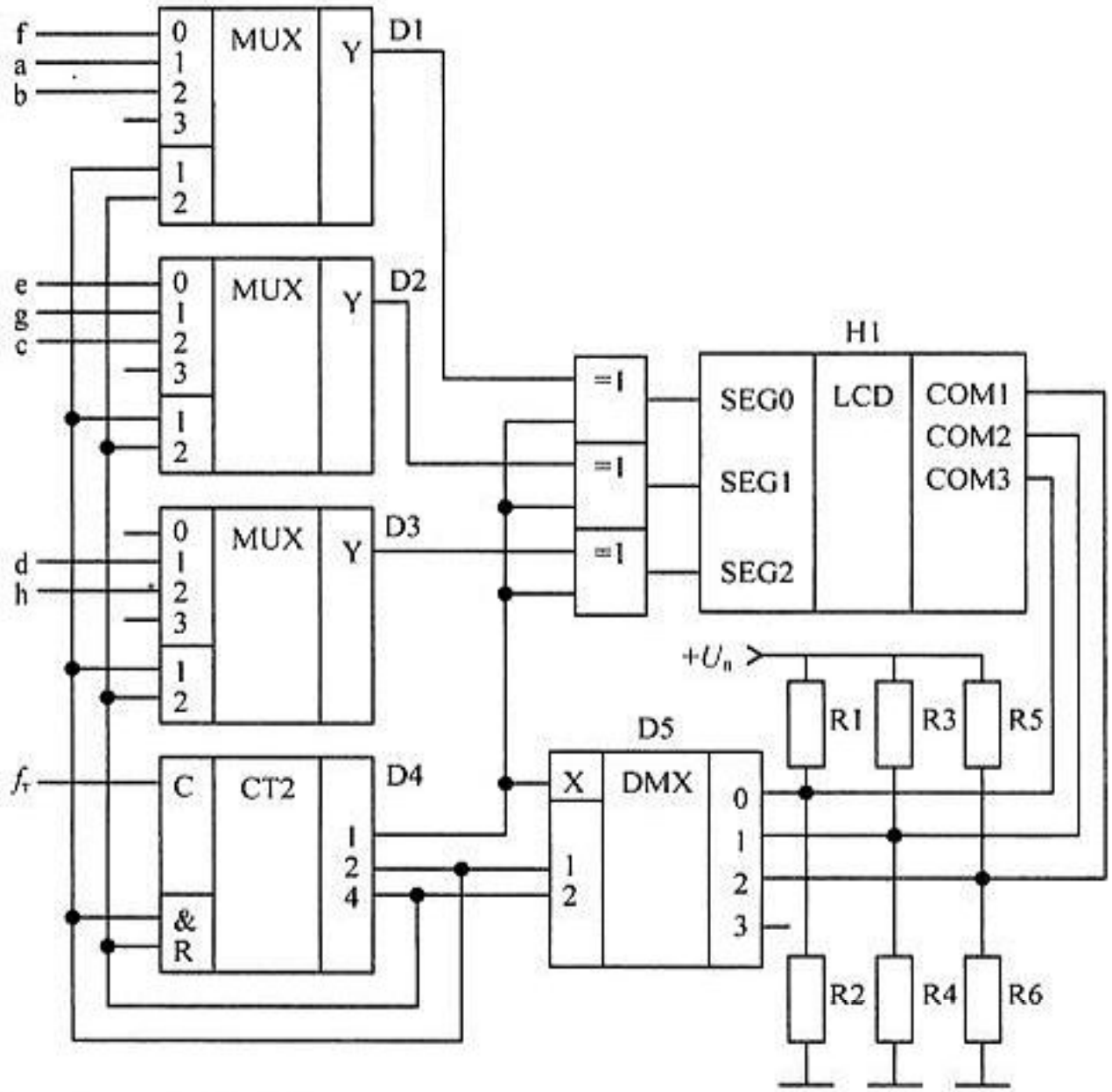


Рис. 9.33. Принципиальная схема контроллера жидкокристаллического индикатора с динамической индикацией

Казалось бы все замечательно. Однако если рассмотреть напряжение на отдельно взятом сегменте жидкокристаллического индикатора, то мы получим временные диаграммы, приведенные на рис. 9.34.

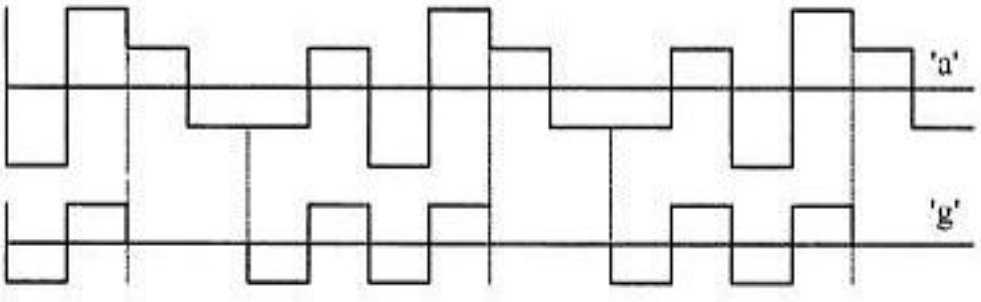


Рис. 9.34. Временная диаграмма напряжения на сегментах жидкокристаллической ячейки

Проанализировав эти временные диаграммы, можно определить, что напряжение возбужденной ячейки отличается от напряжения невозбужденной ячейки всего в два раза. В результате контраст изображения на жидкокристаллическом индикаторе при таком способе динамической индикации получается небольшим.

Для увеличения контрастности изображения в сигнал, подаваемый на строки жидкокристаллического индикатора, можно добавить дополнительный уровень, как это показано на рис. 9.35.

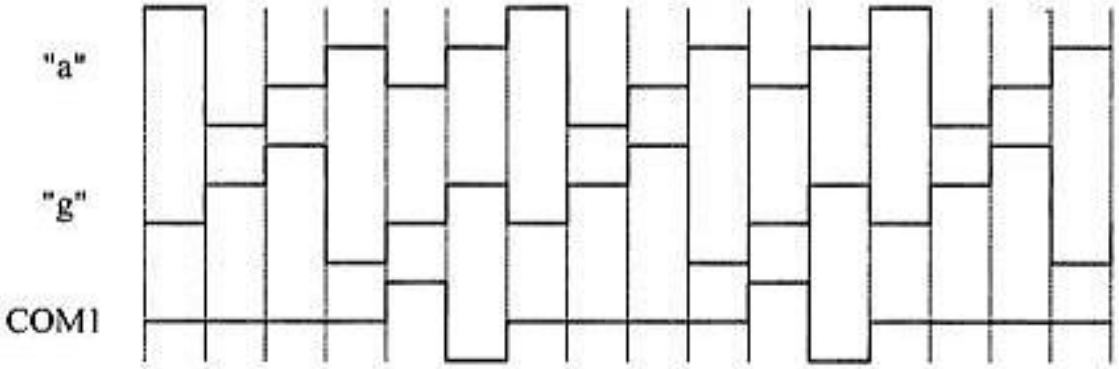
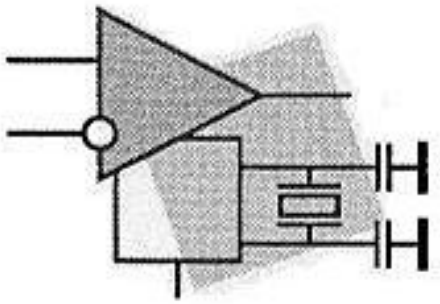


Рис. 9.35. Временные диаграммы напряжения на выводах строк жидкокристаллической ячейки

Использование дополнительных двух уровней в сигнале развертки позволяет частично компенсировать возбуждающее напряжение от соседних сегментов. В результате контрастность формируемого изображения возрастает.

Итоги

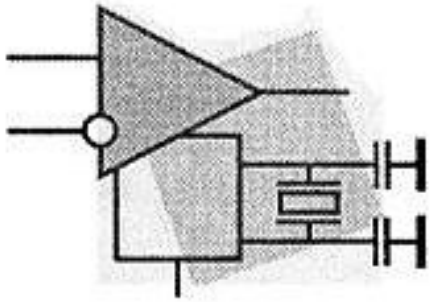
В данной главе мы познакомились с различными видами индикаторов. Теперь мы можем проектировать дисплеи отображения цифровой и символьной информации. Это очень важно при проектировании аппаратуры различного вида: от простейших приемопередающих устройств до сложнейших систем управления производственными процессами и системами наземной или мобильной связи, ведь именно при помощи индикаторов человек получает информацию от работающей аппаратуры. Эти знания потребуются как при разработке цифровых устройств, реализованных на базе микросхем ПЛИС, так и при разработке цифровых устройств на базе микропроцессоров и микроконтроллеров. Даже в тех блоках, где в качестве устройства отображения информации применяются компьютерные дисплеи, для оперативного контроля могут применяться светодиодные или жидкокристаллические индикаторы.



ЧАСТЬ II

Примеры реализации цифровых устройств

ГЛАВА 10



Разработка цифрового устройства на примере электронных часов

Проектирование любого устройства начинается с анализа технического задания. В зависимости от предъявляемых требований для реализации устройства может потребоваться различная элементная база. В ряде случаев подходят готовые микросхемы, ведь обычно решаются однотипные задачи с небольшими изменениями параметров. Иногда, особенно при решении новых уникальных задач, приходится использовать универсальные микросхемы.

В данном случае это действительно распространенная задача и поэтому имеется огромное количество готовых микросхем, выполняющих функции электронных часов. Однако мы собираемся использовать часы как один из примеров разработки цифрового устройства, поэтому не будем пользоваться готовыми микросхемами электронных часов.

Разработка часов, как и любого другого устройства, как аналогового, так и цифрового, начинается с разработки структурной схемы. Это позволяет сократить время разработки всего устройства в целом, за счет исключения системных ошибок на этапе проработки связей между блоками.

Разработка структурной схемы часов

Проанализируем, как должно работать разрабатываемое устройство. Часы обязательно должны содержать устройство измерения времени, которое в свою очередь всегда состоит из генератора эталонных интервалов времени и счетчика этих интервалов. Структурная схема устройства измерения времени приведена на рис. 10.1.

В качестве генератора эталонных импульсов в разное время использовали различные устройства. Это и вытекание воды или песка из какой-либо емкости, и движение тени от солнца по циферблату, и даже горение нити в огненных китайских часах.

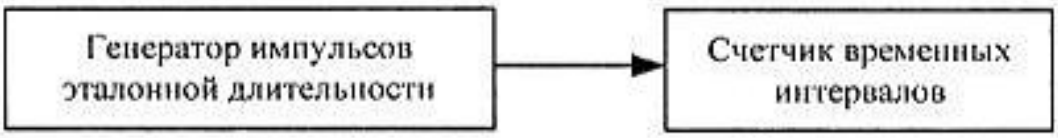


Рис. 10.1. Структурная схема устройства измерения времени

Для реализации часов в простейшем случае генератор импульсов эталонной длительности должен вырабатывать минутные импульсы, однако реализовать стабильный генератор такой длительности достаточно сложно. Даже в механических часах в качестве генератора импульсов эталонной длительности используется маятник с периодом колебаний от одной до нескольких секунд.

При использовании цифровых КМОП-микросхем можно реализовать такой генератор. Для этого потребуется RC-генератор, описанный в предыдущей части книги. Если использовать в качестве времязадающей цепочки резистор номиналом 1 МОм и конденсатор емкостью 30 мкФ, то мы реализуем период повторения импульсов 1 мин. Однако стабильность такого генератора будет невысока, т. к. на сопротивление резистора будет влиять сопротивление печатной платы, а оно в свою очередь будет зависеть от влажности воздуха.

Особенно нестабильными в этой схеме будут конденсаторы, т. к. номиналы большой емкости доступны только при применении электролитических конденсаторов, а производители допускают разброс их параметров до +500, –200%. Как вам понравятся часы, которые вместо одной минуты будут отсчитывать три?

В качестве генератора эталонных импульсов мог бы подойти кварцевый генератор, т. к. этот тип генераторов, как это мы уже обсуждали в предыдущих главах, обладает высокой стабильностью колебаний. Однако кварцевые генераторы вырабатывают колебания в диапазоне частот от 1 до 30 МГц. Это соответствует временным интервалам от 0,03 до 1 мкс.

Тем не менее, если к выходу кварцевого генератора подключить цифровой делитель частоты, то на его выходе можно получить импульсы с периодом повторения 1 минута. Так как цифровой делитель частоты достаточно легко реализовать, и он дешев, то именно это решение обычно используется в электронных часах в качестве генератора импульсов эталонной длительности.

Выберем рабочую частоту кварцевого генератора. В составе генератора можно применить кварцевый резонатор с частотой 32 768 Гц. Этот резонатор специально разрабатывался для применения в часах, поэтому частота его резонанса кратна степени двойки ($2^{15} = 32\,768$). Здесь мы не будем усложнять себе задачу, и используем именно этот резонатор. В результате выбора такого номинала тактовой частоты в составе генератора минутных импульсов можно применить простейшую схему двоичного делителя.

В этом месте хотелось бы отметить, какая грандиозная задача была решена разработчиками кварцевых кристаллов. Дело в том, что если посчитать длину акустической волны в кварце, то кварцевый резонатор с частотой 32 768 Гц получился бы впечатляющих размеров. Толщину кристалла кварца можно определить по общеизвестной формуле для длины волны. Как известно, скорость распространения звуковой волны в кристалле кварца равна 5570 м/с, тогда длина волны будет равна:

$$\lambda = \frac{v}{f} = \frac{5570}{32768} = 17 \text{ см},$$

где v — скорость звука в кристалле кварца;

f — резонансная частота.

То есть толщина кварцевого резонатора должна быть как минимум равна половине длины волны — 8,5 см. Длина кварцевого кристалла резонатора при этом должна быть, по крайней мере, в пять раз больше. Казалось бы, это неразрешимая проблема для малогабаритных и дешевых устройств, однако разработчики кварцевых резонаторов сумели решить ее.

Первым решением проблемы является то, что низкочастотные кварцевые резонаторы изготавливаются с использованием не объемных, а поверхностных волн. Точнее, крутильных колебаний. В результате в резонаторе используется не его толщина, а длина. Скорость распространения волны по поверхности кварца значительно ниже скорости распространения волны в его объеме и равна 3515 м/с. Однако даже в этом случае размеры кварцевого резонатора получаются значительными:

$$L = \frac{v}{f} = \frac{3515}{32768} = 10,7 \text{ см},$$

где v — скорость звука в кристалле кварца;

f — резонансная частота.

Решением проблемы оказалась разработка кварцевого резонатора, реализованного по принципу камертона. В таком резонаторе возбуждаются не объемные колебания, а колебания двух параллельно расположенных стержней, как это показано на рис. 10.2.

В такой конструкции частота резонанса уже не зависит от скорости распространения звуковой волны в кристалле. Она определяется упругостью кварца, длиной и толщиной зубьев получившейся вилки камертона. В то же самое время стабильность колебаний камертона при определенном угле среза относительно кристаллографических осей получается почти та же, что и у обычных кварцевых резонаторов. Угол среза кристалла для часовых резонаторов

отличается от угла среза кристалла для радиотехнических резонаторов. В результате температурная зависимость частоты часового резонатора отличается от температурной зависимости радиотехнического резонатора. Она является параболической. Ориентация камертона часового кварцевого резонатора относительно кристаллографических осей кварцевого кристалла приведена на рис. 10.3, а внешний вид и кристаллографические оси кварцевого кристалла — на рис. 10.4.

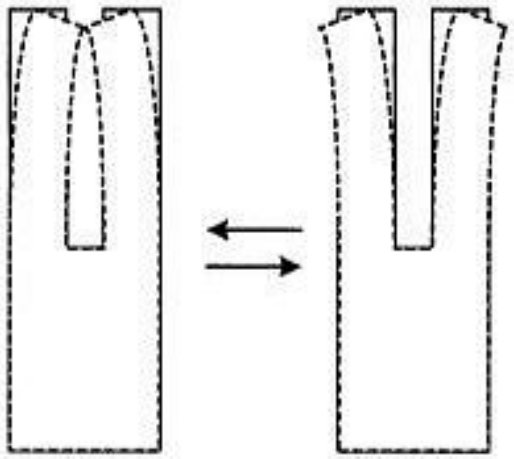


Рис. 10.2. Кристалл часового кварцевого резонатора в режиме колебаний камертона

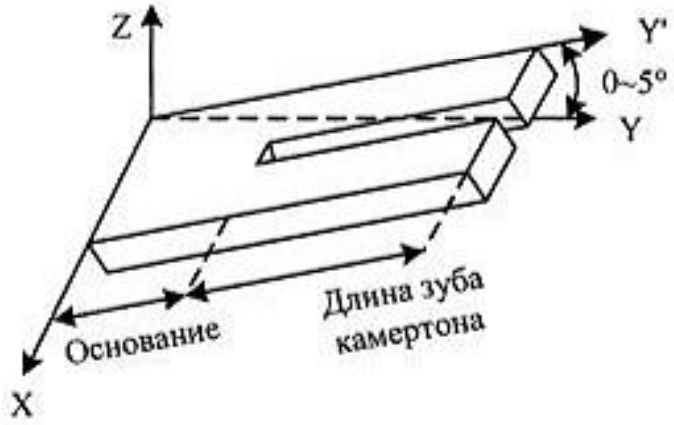


Рис. 10.3. Кристаллографическая ориентация камертона

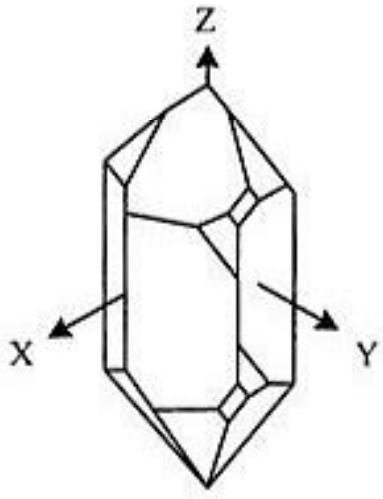


Рис. 10.4. Внешний вид и кристаллографические оси кварцевого кристалла

Естественно, что стоимость разработки часового кварцевого резонатора была огромнейшей. Однако и количество таких резонаторов, ежегодно потребляемых часовой промышленностью, тоже огромно. В результате стоимость часовых кварцевых резонаторов оказалась минимальной из всех представленных на рынке на данный момент кварцевых резонаторов. Благодаря своей

распространенности, малой цене, габаритам и малой рабочей частоте, часовые кварцевые резонаторы начинают применяться практически во всех цифровых устройствах, т. к. при помощи делителей и умножителей частоты мы можем получить любую нужную нам тактовую частоту.

Для решения поставленной перед нами задачи полезными свойствами часового кварцевого резонатора является малая цена, габариты, кратность частоты одному герцу и относительно низкая частота резонанса примененного в нем кристалла. Последнее свойство определяет частоту выходного колебания задающего генератора и, как следствие, малое значение тока потребления этим генератором от источника питания.

Итак, для формирования секундных импульсов (частота 1 Гц) потребуется делитель частоты с коэффициентом деления равным 32 768. Для формирования из секундных импульсов минутных потребуется еще один делитель частоты. Так как в минуте содержится 60 секунд, то его коэффициент деления тоже будет равен 60. Уточненная структурная схема разрабатываемого цифрового устройства приведена на рис. 10.5.

Рис. 10.5. Уточненная структурная схема устройства измерения времени

Теперь займемся структурной схемой второго блока часов — счетчика интервалов времени. Он будет состоять из счетчика минут и счетчика часов. Мы знаем, что счетчик минут должен работать по основанию 60. В то же самое время мы привыкли воспринимать числа в десятичной системе счисления. Поэтому будет удобно разбить счетчик минут на два счетчика: на десятичный счетчик и счетчик, считающий до шести.

Счетчик часов можно выполнить по основанию 12 и по основанию 24. И то и другое решение, по традиции, допустимо при отображении показаний часов. Пусть в нашем устройстве счетчик часов будет считать до 24-х. При этом, для упрощения схемы отображения информации, так же как и в счетчике минут, реализуем его на двух десятичных счетчиках.

Следующий блок, который обязательно должен входить в состав часов, — это устройство индикации. Ведь никого не устроят часы, которые будут точно отсчитывать время, но при этом мы не сможем увидеть результат!

Выберем в качестве устройства отображения времени светодиодные семисегментные индикаторы. В этом случае мы получим устройство, способное работать при отрицательной температуре окружающего воздуха и обладающее наиболее простой схемой.

Для преобразования кода, в котором работает счетчик минутных импульсов, в семисегментный код нам потребуется дешифратор. В результате блок индикации будет состоять из семисегментных дешифраторов и светодиодных индикаторов. Получившаяся структурная схема электронных часов приведена на рис. 10.6.

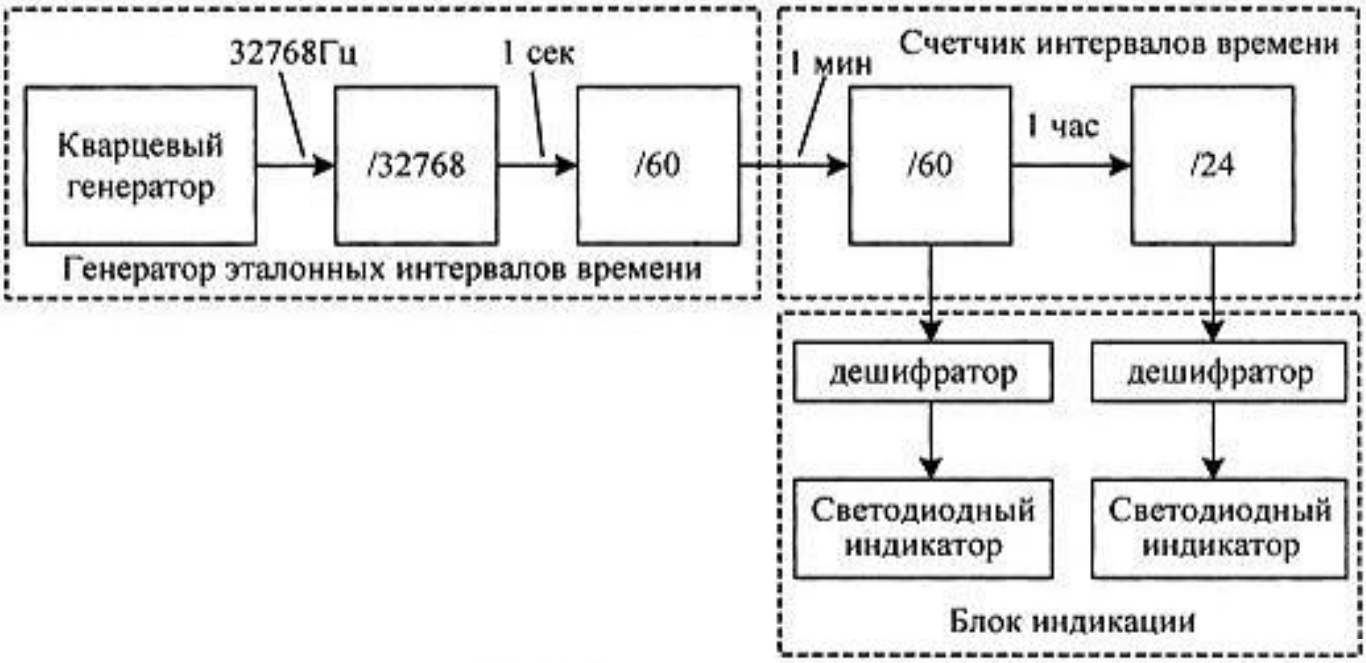


Рис. 10.6. Структурная схема часов

И, наконец, последнее замечание. Любые часы время от времени требуют коррекции своего внутреннего состояния с целью синхронизации своих показаний с всемирным временем. В нашей схеме это действие будет выполнять блок коррекции, который в свою очередь будет состоять из кнопок и схемы установки внутреннего состояния счетчика интервалов времени.

Пусть блок коррекции времени позволяет независимо устанавливать показания часов, минут и обнулять внутренние счетчики генератора эталонных интервалов времени. Для формирования команд, соответствующих этим трем задачам, нам будет достаточно трех кнопок.

При разработке нашего варианта электронных цифровых часов воспользуемся простейшим алгоритмом установки их показаний. При нажатии на кнопку установки минут или установки часов будем подавать на вход счетчика минут или часов повышенную частоту сигнала.

Человек вполне может воспринимать смену информации несколько раз в секунду, поэтому в режиме установки времени выберем частоту повторения импульсов четыре раза в секунду. В результате выбора такой частоты повторения импульсов время установки содержимого счетчика минут не будет превышать 15 сек, а максимальное время установки счетчика часов будет равно 6 сек.

На этом можно завершить разработку структурной схемы. Полная структурная схема часов с учетом блока индикации и блока коррекции времени приведена на рис. 10.7.

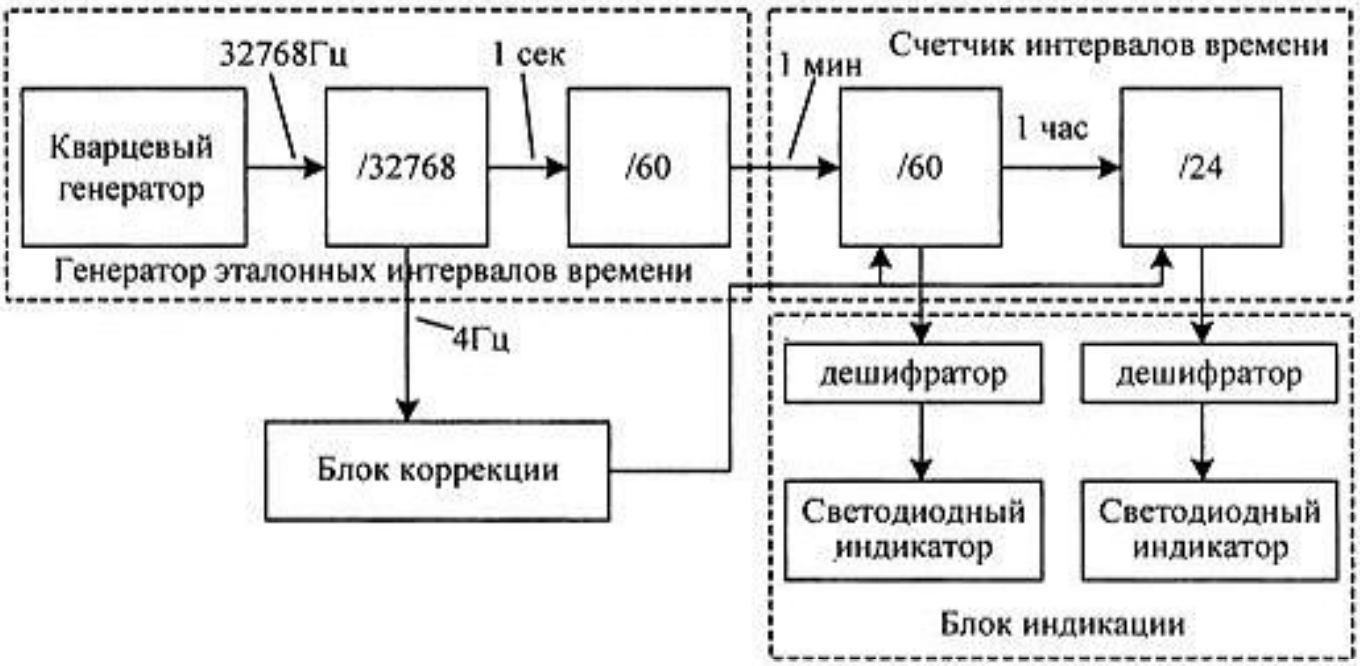


Рис. 10.7. Полная структурная схема часов

Теперь, после того как составлена структурная схема часов и рассчитаны параметры отдельных ее блоков, можно приступить к разработке принципиальной схемы этих блоков. Еще раз отмечу, что каждый отдельный блок структурной схемы мы можем разрабатывать независимо от других блоков, для чего, собственно говоря, нам и нужна была структурная схема.

Разработка принципиальной схемы часов

При разработке принципиальной схемы чрезвычайно важен выбор серии микросхем, на основе которой будет реализована эта схема. Для часов самым важным параметром является ток, потребляемый ими, т. к. в большинстве случаев или все часы, или часть схемы часов питается от элементов питания. Поэтому при разработке схемы будем выбирать микросхемы, реализованные по КМОП-технологии.

Разработка схемы генератора эталонных интервалов времени

Разработку схемы часов начнем с кварцевого генератора. Как уже обсуждалось при разработке структурной схемы, в составе генератора будет применен часовой кварцевый резонатор. Для уменьшения стоимости всего устройства в целом применим простейшую схему генератора — емкостную трехточку, а т. к. генератор предназначен для синхронизации цифрового устройства, то генератор выполним на логическом инверторе. Принципиальная схема такого кварцевого генератора приведена на рис. 10.8.

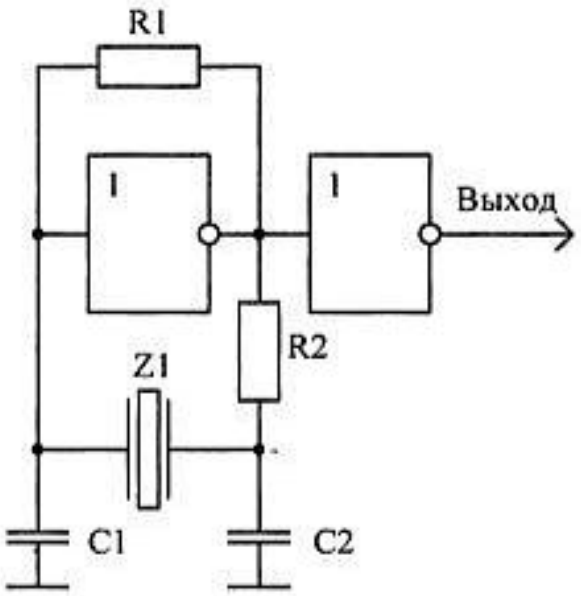


Рис. 10.8. Схема кварцевого генератора, выполненная на логическом инверторе

В качестве микросхемы, содержащей инверторы, выберем микросхему SN74LVC2G04DRLR. В этой микросхеме, построенной по КМОП-технологии, содержится два инвертора. О том, что в микросхеме содержится два элемента, говорит обозначение 2G. То, что это инверторы, обозначается цифрой 04, а то, что в микросхеме использован корпус с шагом выводов 0,5 мм — буквы DRL. Размеры корпуса этой микросхемы не превышают 1,6 × 1,6 мм (у корпуса всего шесть выводов). Микросхема способна работать в диапазоне напряжений от 1,5 до 5,5 В.

Напомним, что резистор R1 в схеме, приведенной на рис. 10.6, предназначен для автоматического запуска генератора при включении питания. С точки зрения запуска генератора при включении питания стоит выбирать этот резистор не слишком большого номинала.

Этот же элемент определяет коэффициент усиления инвертора, и чем больше будет этот коэффициент усиления, тем более прямоугольные колебания бу-

дут формироваться на его выходе, а это, в свою очередь, приведет к снижению тока, потребляемого кварцевым генератором. С этой точки зрения желательно увеличивать номинал резистора R1.

Рассчитаем мощность, потребляемую логическим элементом на частоте 32 768 Гц. Для этого воспользуемся формулой, приведенной в [15]. Номинал внутренней динамической емкости для микросхемы SN74LVC2G04DRLR равен $C_{pd} = 14$ пФ. Тогда:

$$P_T = C_{pd} \times V_{CC}^2 \times f \times N_{SW} = 14 \times 10^{-12} \times 3{,}3^2 \times 32768 \times 1 = 5 \text{ мкВт}$$

Ему соответствует ток потребления микросхемы, равный:

$$I_{CC} = \frac{P_T}{V_{CC}} = \frac{5}{3{,}3} = 1{,}5 \text{ мкА}$$

Выберем ток, потребляемый цепью смещения, в пять раз меньше. Тогда по закону Ома сопротивление резистора R1 получится равным:

$$I_L = \frac{I_{CC}}{5} = \frac{1{,}5 \text{ мкА}}{5} = 0{,}3 \text{ мкА}$$

$$R1 = \frac{V_{CC}}{I_L} = \frac{3{,}3 \text{ В}}{0{,}3 \text{ мкА}} = 10 \text{ МОм}$$

Это очень большое значение номинала резистора, поэтому следует позаботиться о конструктивном исполнении устройства. При неправильной конструкции или технологии изготовления устройства сопротивление поверхности печатной платы может оказаться меньше номинала резистора, и в результате схема генератора будет потреблять ток, намного превышающий расчетное значение. Это может привести к необходимости применения элементов питания слишком большой емкости, а значит, к увеличению габаритов и массы всего устройства в целом.

Номинал емкости конденсаторов C1 и C2 определяется величиной паразитных емкостей схемы. Выберем конденсаторы с емкостью 20 пФ. Такой номинал емкостей позволяет уменьшить влияние емкостей печатной платы и входной емкости инвертора на коэффициент обратной связи генератора. Кроме того, этот номинал соответствует рекомендуемому производителями кварцевых резонаторов значению емкости нагрузки.

Равное значение емкостей этих конденсаторов позволяет обеспечить коэффициент передачи колебательной системы, построенной на кварцевом резонаторе Z1 и конденсаторах C1 и C2, равный 0,5.

Резистор R2 предназначен для предотвращения самовозбуждения генератора на частоте, определяемой емкостью кварцедержателя. В [16] сопротивление

резистора R2 рекомендуется выбирать равным импедансу конденсатора C2. Рассчитаем это значение:

$$X_C = \frac{1}{j\omega C} = \frac{1}{2 \times 3{,}14 \times 32768 \times 20 \times 10^{-12}} = 243 \text{ кОм}$$

При таком номинале коэффициент обратной связи будет равным 0,5. Учитывая коэффициент передачи схемы на кварцевом резонаторе, общий коэффициент обратной связи будет равным 0,25. Для увеличения этого коэффициента передачи для более стабильного запуска генератора увеличим номинал резистора вдвое по сравнению с рассчитанным значением. Это, кроме того, приведет к более прямоугольной форме колебаний на выходе генератора, а значит, к меньшему току потребления схемы в целом. Выберем номинал резистора R2 равный 510 кОм.

Второй инвертор в схеме генератора предназначен для уменьшения длительности фронтов формируемого прямоугольного колебания. Это необходимо для уменьшения влияния последующей схемы на стабильность колебаний задающего генератора, а также для более надежной работы цифровых счетчиков делителя частоты.

Следующей реализуем схему делителя частоты опорного колебания до значения 1 Гц. Как это мы уже определили при разработке структурной схемы, его коэффициент деления должен быть равен 32 768. То есть для реализации делителя потребуется 15 счетных триггеров. Конечно, можно взять микросхему К176ИЕ12, специально разработанную для применения в часах, но мы не ищем простых путей, поэтому используем универсальную микросхему SN74HC393PW. В ней есть два независимых четырехразрядных двоичных счетчика. Это означает, что для реализации нашего делителя будет достаточно всего двух микросхем.

Размеры корпуса выбранной микросхемы не превышают 5 × 6,4 мм. У корпуса этой микросхемы имеется 14 выводов. По сравнению с микросхемой К176ИЕ12 по занимаемым габаритам мы получили значительный выигрыш. Если к габаритам часов нет особых требований, то можно использовать и отечественную микросхему К1564ИЕ19. Ее корпус больше корпуса выбранной нами микросхемы более чем в два раза. Тем не менее, при такой замене микросхем будут совпадать даже номера выводов корпусов. Полученная в результате приведенных выше рассуждений принципиальная схема генератора секундных импульсов представлена на рис. 10.9.

Теперь вспомним, что в генераторе временных интервалов необходим еще один делитель частоты. Период импульсов на его выходе будет равен 1 минуте. Делитель на шестьдесят можно реализовать на точно такой же микросхеме SN74HC393PW, что мы использовали и ранее для построения делителя на 32 768.

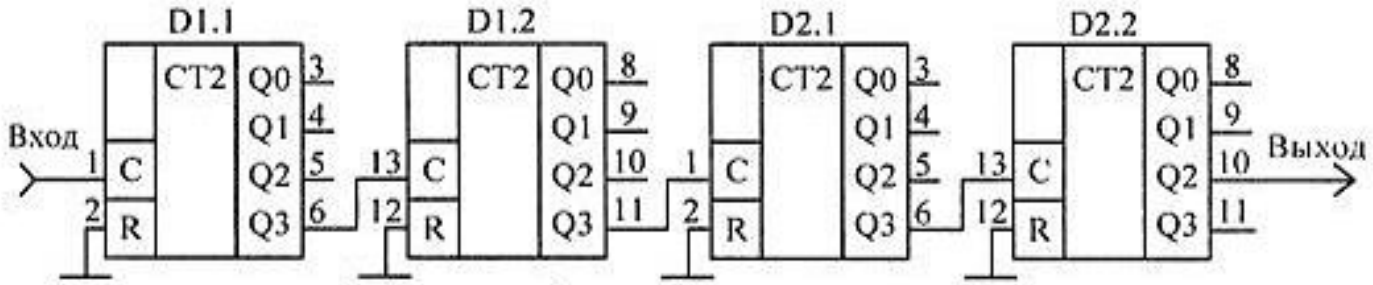


Рис. 10.9. Схема делителя на 32768

Коэффициент деления делителя на шестьдесят не кратен степени числа два, поэтому для его реализации потребуется обратная связь. Для упрощения схемы обратим внимание, что число 60 разбивается на числа 10 и 6. И то и другое число в двоичном представлении содержат только две единицы ($10_{10} = 1010_2$, $6_{10} = 0110_2$). Выводы 4-разрядных счетчиков выходят на разные стороны корпуса микросхемы. Поэтому будет удобно использовать два независимых логических элемента "2И" малой логики. Это позволит значительно упростить разводку печатной платы и сократить длину соединительных проводов, тем самым, уменьшив площадь печатной платы и возможные помехи от работающей схемы.

В качестве логических элементов "2И" используем две микросхемы SN74LVC1G08DRLR. То, что в микросхеме содержится только один логический элемент, мы определяем по символам 1G, а то, что это логический элемент "2И" — по цифрам 08. Размеры корпуса выбранной микросхемы не превышают 1,6 × 1,6 мм (при условии, что в названии микросхемы присутствуют буквы DRL).

Отечественные варианты подобной микросхемы, например К1554ЛИ1, содержат в одном корпусе сразу по четыре логических элемента, расстояние между выводами составляет минимум 1,25 мм. В результате схема, собранная на таких микросхемах, будет идентична по электрическим параметрам, но проиграет по размерам.

Полученная принципиальная схема делителя частоты на 60 приведена на рис. 10.10. Эта схема вырабатывает импульсы с периодом 1 мин и состоит из последовательно включенных делителей на 10 и на 6. Схема делителя на 60 реализована всего на трех микросхемах. Использование обратной связи с выводов Q1 и Q3 превращает двоичный счетчик D1.1 в десятичный, а применение обратной связи с выводов Q1 и Q2 микросхемы D1.2 реализует счетчик по модулю 6.

Итак, мы закончили разработку генератора минутных импульсов. Всего нам потребовалось шесть микросхем, при этом три из них относятся к микросхемам малой логики и занимают минимум места на печатной плате цифрового устройства.

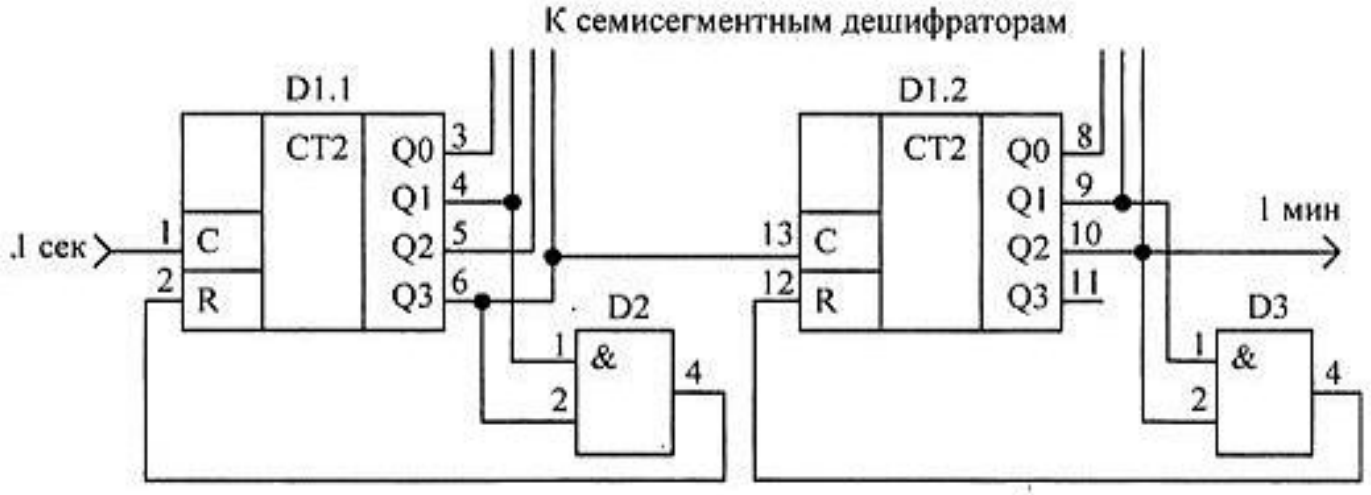


Рис. 10.10. Принципиальная схема делителя на 60 генератора минутных импульсов

Разработка схемы счетчика интервалов времени

Теперь можно приступить к разработке принципиальной схемы счетчика интервалов времени. Как мы уже выяснили при разработке структурной схемы часов, в состав этого счетчика входит точно такой же делитель на 60, как и в генераторе минутных импульсов, поэтому для счетчика минут можно воспользоваться уже разработанной принципиальной схемой, приведенной на рис. 10.10.

Отличие в применении схемы счетчика заключается только в том, что на этот раз нам потребуются все выходы счетчиков. Логические потенциалы с выходов счетчиков мы будем подавать на вход семисегментных дешифраторов блока индикации для отображения внутреннего состояния счетчика минут на светодиодных индикаторах.

Последний счетчик, который нам потребуется для реализации блока счетчика интервалов времени, — это счетчик часов с коэффициентом счета 24. Этот счетчик было бы удобно реализовать на микросхеме десятичного счетчика, однако сдвоенных микросхем асинхронных десятичных счетчиков не производится, поэтому реализуем счетчик часов на той же микросхеме, что и остальные блоки часов — микросхеме SN74HC393PW.

Сложность при разработке схемы счетчика часов заключается в том, что коэффициент его счета не кратен десяти, поэтому сигнал обратной связи необходимо заводить на обе части микросхемы одновременно. Можно было бы реализовать этот счетчик в двоичном виде, но тогда возникнут сложности с отображением содержимого этого счетчика на семисегментных индикаторах.

Для того чтобы реализовать на первом 4-разрядном счетчике десятичный счетчик и одновременно получить возможность сброса всего счетчика часов

в начале суток, используем в цепи обратной связи дополнительный логический элемент "2ИЛИ". Сигнал сброса на выходе этого элемента появится либо в случае достижения счетчиком D1.1 числа 10, либо при достижении всем счетчиком часов значения 24.

Полная принципиальная схема счетчика часовых импульсов, реализованная на микросхеме SN74HC393PW, приведена на рис. 10.11.

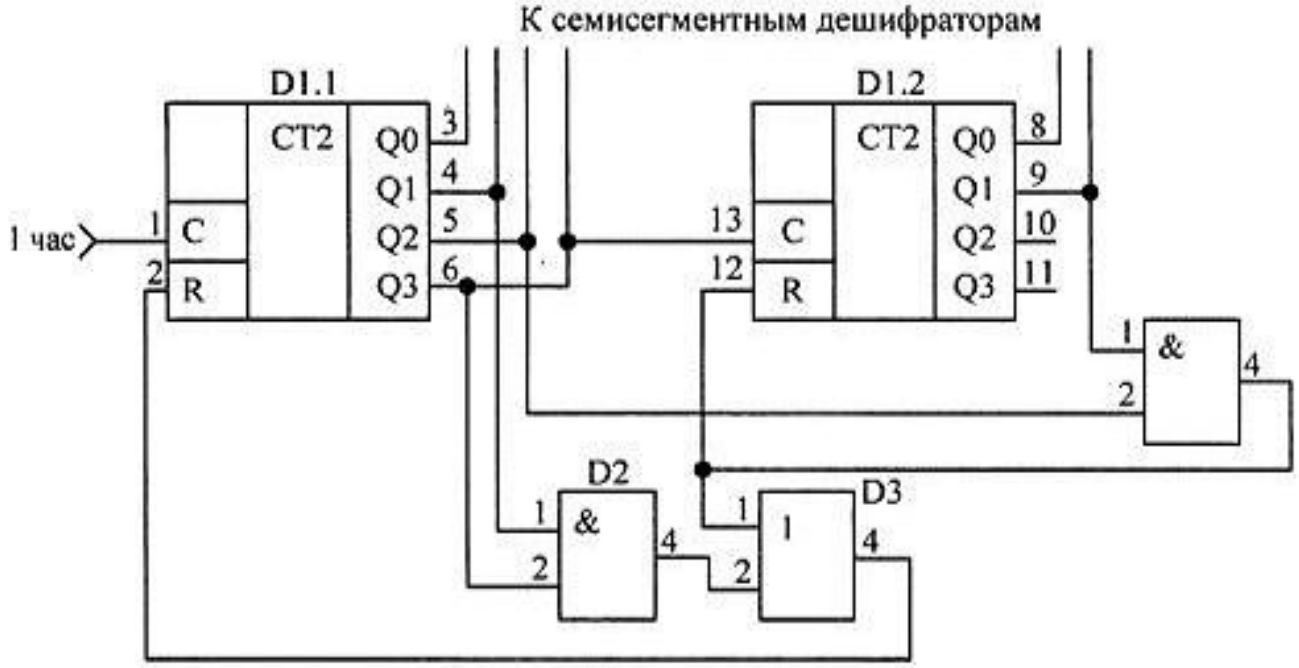


Рис. 10.11. Схема счетчика часовых импульсов

В качестве логического элемента "2ИЛИ" используем микросхему малой логики, подобную уже использованной микросхеме "2И". Это микросхема SN74LVC1G32DRLR. Цифра 32 в названии микросхемы и обозначает логический элемент "2ИЛИ". Размеры корпуса этой микросхемы не превышают 1,6 × 1,6 мм. В результате, несмотря на несколько более сложную по сравнению с применением двух десятичных счетчиков принципиальную электрическую схему, площадь, занимаемая счетчиком часов, значительно уменьшается.

Применение обратной связи с выводов Q1 и Q3 первой микросхемы превращает ее в десятичный счетчик. Для реализации счетчика по модулю 24 мы используем обратную связь с вывода Q1 старшего разряда счетчика (двойка) и вывода Q2 младшего разряда счетчика часов (четверка). Иначе говоря, микросхемы D1.1 и D2 образуют десятичный счетчик, а микросхемы D1.1, D1.2 и D4 реализуют счетчик по модулю 24. Микросхема D3 объединяет сигналы обратной связи десятичного счетчика и счетчика часов для первой микросхемы делителя часов D1.1.

Таким образом, мы реализовали основную часть схемы часов. Схема уже умеет измерять время, но, как уже обсуждалось при разработке структурной схемы, этого недостаточно. Требуется уметь отображать полученную информацию о текущем времени. Перейдем к разработке блока индикации часов.

Разработка принципиальной схемы блока индикации

При разработке блока счета интервалов времени мы построили схему так, чтобы блок индикации можно было бы выполнить из набора семисегментных дешифраторов и индикаторов. Конечно, как и в случае с часами, можно найти готовый блок индикации со встроенным контроллером, но как это уже говорилось ранее — мы не ищем легких путей. В разрабатываемых часах мы построим блок индикации на отдельных микросхемах.

Сначала давайте выберем микросхему семисегментного дешифратора. При разработке структурной схемы мы определили, что для индикации внутреннего состояния счетчиков будем использовать светодиодные индикаторы. Поэтому попробуем найти микросхему дешифратора, способную работать на светодиодный индикатор. В результате поиска находим достаточно старые, но выпускающиеся до настоящего времени микросхемы — SN74LS247D. Выбор иностранной микросхемы обусловлен исключительно размерами ее корпуса.

В Советском Союзе выпускались микросхемы К555ИД18 — полный аналог этой микросхемы за исключением корпуса. Корпус в два раза больше (как у микросхемы SN74LS247N). Эти микросхемы до сих пор можно найти в продаже. Однако с точки зрения корпуса, да и с точки зрения надежности лучше поставить другую отечественную микросхему — 514ИД2. Ее можно считать аналогом микросхемы SN54LS247W, выполняющей подобную функцию.

Проанализируем возможности микросхемы SN74LS247D. В результате этого анализа определяем, что на выходе этой микросхемы используется схема с открытым коллектором, допускающая подачу напряжения на выводы подключения светодиодных сегментов до 15 В. Втекающий ток этих выводов может достигать 24 мА. Приведенные параметры показывают, что микросхема SN74LS247D идеально подходит для использования совместно со светодиодным индикатором.

Так как схема подключения всех сегментов одинакова, то можно ограничиться расчетом элементов для одного сегмента. Номиналы элементов для остальных сегментов будут полностью идентичными. Рассчитаем схему подключения одного сегмента светодиодного индикатора к микросхеме семисегментного дешифратора SN74LS247D. Эквивалентная схема подключения

одного сегмента светодиодного индикатора к выходному каскаду микросхемы SN74LS247D приведена на рис. 10.12.

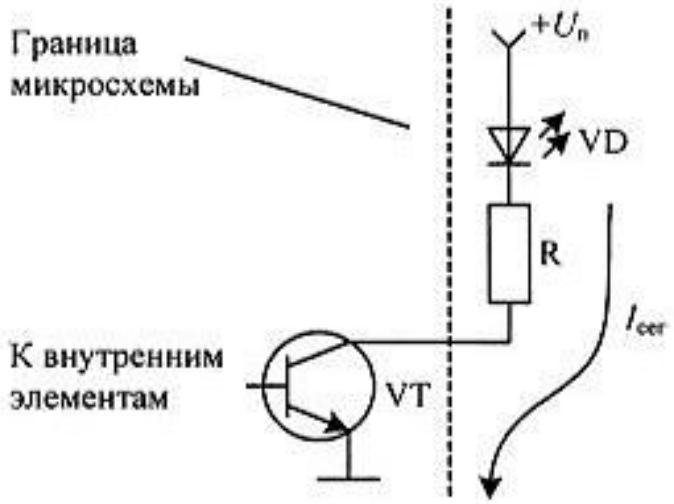


Рис. 10.12. Схема подключения одного сегмента светодиодного индикатора к выходному каскаду микросхемы SN74LS247D

Сначала определим напряжение питания блока индикации. Для этого необходимо знать параметры светодиодного индикатора, поэтому выберем светодиодный индикатор. Пусть в схеме будет использоваться суперъяркий светодиодный индикатор поверхностного монтажа с красным цветом свечения ACSA56-41SRWA-F01. Этот индикатор производится фирмой Kingbright.

Падение напряжения на этом индикаторе может составлять от 1,7 до 2,5 В. При пятивольтовом питании схемы, падение напряжения на балластном сопротивлении R можно определить по следующей формуле:

$$U_{R\max} = U_{П\max} - U_{VD\min} = 5{,}25\ \text{В} - 1{,}7\ \text{В} = 3{,}55\ \text{В}$$

$$U_{R\min} = U_{П\min} - U_{VD\max} = 4{,}75\ \text{В} - 2{,}5\ \text{В} = 2{,}25\ \text{В}$$

По закону Ома разброс падения напряжения на балластном сопротивлении будет определять разброс тока через светодиодные индикаторы. Это в конечном итоге даст разброс яркости свечения сегментов. В нашем случае разброс падения напряжения на балластном резисторе составил 1,6 раза, что вполне приемлемо для бытового прибора, поэтому оставим напряжение питания блока индикации равным пяти вольтам.

Зададимся током через сегмент светодиодного индикатора. Пусть этот ток будет равен 5 мА. Тогда сопротивление балластного резистора будет равно:

$$R = \frac{U_П - U_{Vd}}{I_{VD}} = \frac{5B - 1{,}7\ \text{В}}{5\ \text{мА}} = 650\ \text{Ом}$$

Выбираем ближайшее значение из десятипроцентного ряда резисторов — 670 Ом.

Теперь составим принципиальную схему подключения светодиодного индикатора ACSA56-41SRWA-F01 к микросхеме семисегментного дешифратора SN74LS247D. Она приведена на рис. 10.13.

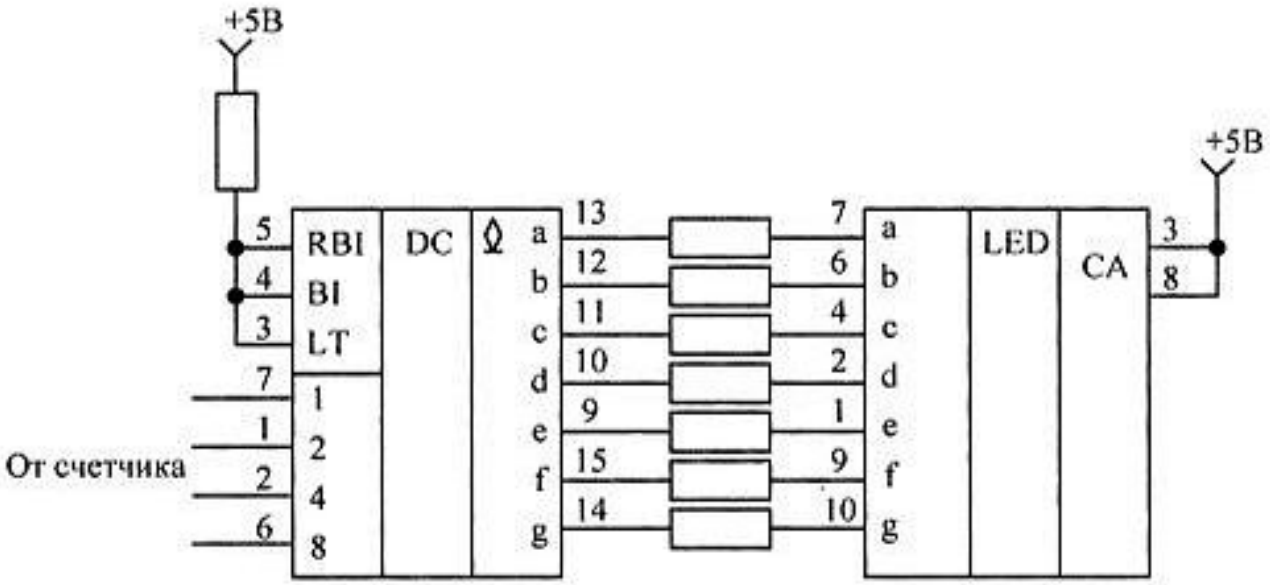


Рис. 10.13. Схема подключения светодиодного индикатора ACSA56-41SRWA-F01 к микросхеме семисегментного дешифратора SN74LS247D

В приведенной на рис. 10.13 схеме выход каждого сегмента дешифратора подключается к соответствующему сегменту индикатора через токоограничивающий резистор.

Кроме информационных входов "1" "2" "4" "8" в микросхеме дешифратора SN74LS247D имеются управляющие входы. Вход BI позволяет гасить индикатор. Вход LT позволяет проверять исправность сегментов индикатора, зажигая сразу все сегменты. Вход RBI позволяет гасить незначащие нули в отображаемом числе.

Мы не будем пользоваться всеми возможностями, предоставляемыми микросхемой, поэтому соединим перечисленные управляющие входы с высоким потенциалом. Так как вход BI может одновременно служить выходом, то высокий потенциал подадим через резистор R1.

Схема подключения индикатора для всех четырех отображаемых разрядов не отличается, поэтому полную схему блока индикации приводить не будем.

Разработка принципиальной схемы блока коррекции времени

Теперь последним неразработанным блоком часов остался блок коррекции времени. Приступим к его разработке.

Наиболее простой схемой будет обладать устройство установки содержимого счетчика минут и счетчика часов. Как уже было определено при разработке структурной схемы, для коррекции времени на вход счетчика минут или счетчика часов должна подаваться частота 4 Гц с выхода одного из двоичных счетчиков предварительного делителя тактовой частоты.

Блок коррекции времени можно выполнить на основе цифрового коммутатора, который будет подключать к счетному входу счетчиков либо импульсы с выхода предыдущего блока, либо частоту установки нового показания часов 4 Гц.

Выберем микросхему цифрового коммутатора. Как и в предыдущих случаях, основным критерием будут габариты этой микросхемы и потребляемый ею ток. По предъявленным параметрам наилучшим образом подходит микросхема SN74LVC1G3157DRLR. Эта микросхема может коммутировать как цифровые, так и аналоговые сигналы. Габариты микросхемы SN74LVC1G3157DRLR составляют 1,6×1,6 мм.

Для переключения входов коммутатора логический уровень на его управляющем входе будем изменять при помощи кнопки. Для формирования логического уровня единицы воспользуемся резистором, подключенным к источнику питания. При разомкнутых контактах кнопки этот уровень поступит на управляющий вход A коммутатора D2. При замыкании контактов кнопки S1 на этот вход будет подан потенциал корпуса. Таким образом, замыкание и размыкание контактов кнопки будет преобразовано в логические уровни нуля и единицы. Конденсатор C1 предотвращает возникновение серии импульсов из-за явления "дребезга" контактов. Такую схему мы уже неоднократно применяли ранее. Схема счетчика минут с устройством коррекции показаний приведена на рис. 10.14.

В нормальном режиме работы часов на управляющий вход A коммутатора D2 через резистор R1 от источника питания подается высокий потенциал. При этом на тактовый вход C десятичного счетчика D3.1 поступают импульсы с периодом повторения 1 мин.

При замыкании контактов кнопки установки минут S1 на управляющем входе коммутатора D2 формируется низкий потенциал. Выход коммутатора переключается к входу B2. В результате на выход коммутатора будет поступать сигнал установки показаний минут с частотой 4 Гц. Это обеспечит скорость изменения показаний часов в режиме установки времени 4 раза в секунду. Конденсатор C1 служит для того, чтобы не возникало ложных импульсов установки времени за счет "дребезга" контактов кнопки S1.

Схема установки внутреннего состояния счетчика часов строится подобным образом, поэтому приводить ее не имеет смысла.

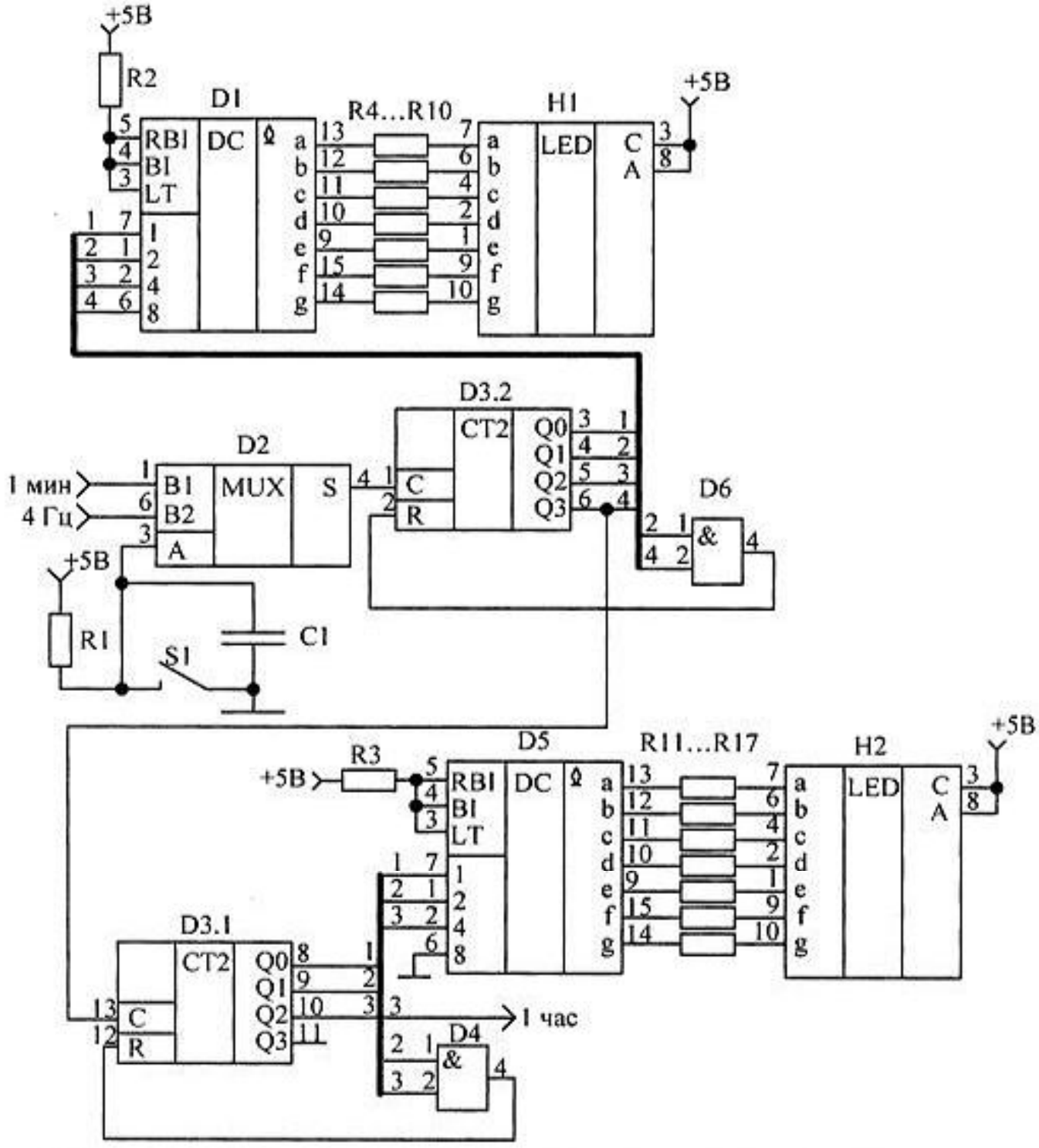


Рис. 10.14. Схема счетчика минут с устройством установки внутреннего состояния

Несколько отличается схема установки показаний счетчика секунд. При нажатии на кнопку установки секунд необходимо сбросить счетчик секунд (записать во все его разряды нулевое значение). Для этого достаточно подать потенциал на вход сброса R, однако мы уже использовали входы сброса для формирования модуля счета 60, поэтому в данной схеме тоже потребуются коммутаторы, которые позволят схеме либо работать в режиме счетчика секунд, либо оставаться в сброшенном состоянии.

Разработанная принципиальная схема счетчика секунд с блоком установки его в нулевое состояние приведена на рис. 10.15.

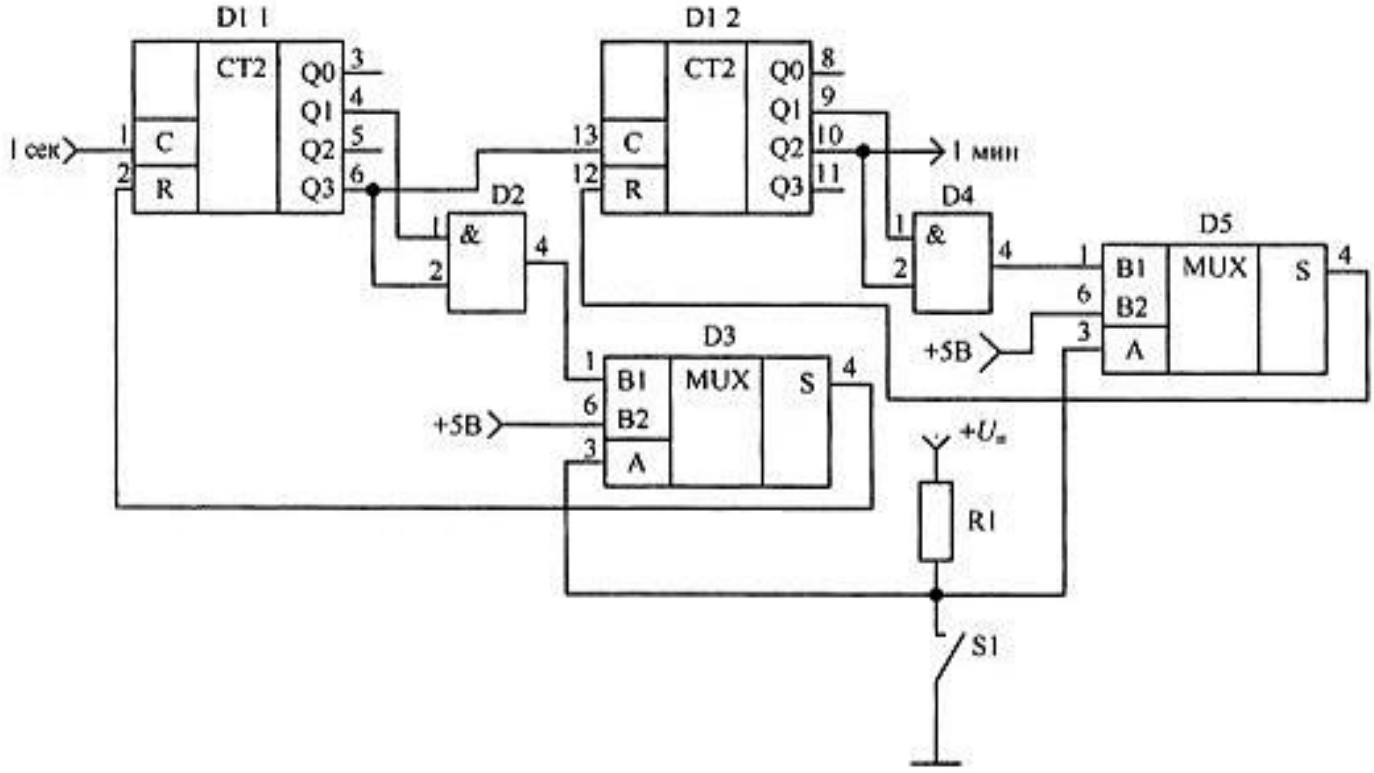


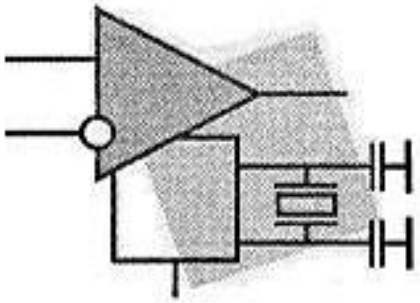
Рис. 10.15. Схема счетчика секунд с устройством сброса

В данной схеме кнопка сброса внутреннего состояния счетчика секунд S1 формирует сигнал сразу на оба коммутатора. Кроме того, в ней отсутствует схема устранения дребезга контактов. Это связано с тем, что при любом количестве нажатий на кнопку установки секунд счетчик устанавливается в одно и то же состояние.

Итоги

На этом разработка принципиальной схемы часов завершается. Часы можно считать классическим примером использования цифровых счетчиков. Здесь еще раз необходимо подчеркнуть, что приведенный пример является учебным. В практической реализации проще и дешевле взять готовую микросхему часов. Не менее эффективным решением является применение программируемой логической интегральной микросхемы (ПЛИС), однако проектирование цифровых устройств на ПЛИС имеет свои особенности и выходит за рамки данной книги. В учебном курсе "Цифровые устройства и микропроцессоры" проектирование цифровых устройств на ПЛИС изучается в процессе лабораторного практикума [24]. Далее рассмотрим примеры использования последовательного регистра.

ГЛАВА 11



Синхронные последовательные порты

При работе с современными цифровыми и аналого-цифровыми микросхемами достаточно важным вопросом является количество внешних выводов используемых микросхем. В большинстве случаев именно количество выводов определяет габариты микросхемы. Чем больше выводов — тем больше габариты корпуса микросхемы, а следовательно, и габариты всего проектируемого устройства в целом. Именно поэтому в современных микросхемах стараются минимизировать количество внешних выводов.

Максимально минимизировать количество выводов микросхемы с цифровым управлением можно при использовании последовательной передачи различных бит через один и тот же вывод микросхемы. Для того чтобы можно было снова разделить биты на приемном конце, используются специальные сигналы — сигналы синхронизации. Они могут передаваться через отдельные выводы микросхемы или совмещаться с передаваемыми информационными битами.

Цифровые устройства, формирующие совокупность информационных и синхронизирующих сигналов или способные принимать эту совокупность, называются последовательными портами. Последовательные порты встраиваются в состав микросхем однокристальных приемников, синтезаторов частот, аналого-цифровых и цифроаналоговых преобразователей, ну и естественно они входят в состав современных сигнальных процессоров и микроконтроллеров.

При разработке новых видов схем с использованием программируемых логических или программируемых аналоговых схем использование последовательных портов тоже может значительно уменьшить количество выводов микросхемы, требуемое для обмена информацией между различными блоками.

SSI-интерфейс (DSP-порт)

Структура синхронного последовательного интерфейса (synchronous serial interface — SSI) в основном определяется типом данных, передаваемых по этому порту. При обработке сигналов требуется передавать непрерывные потоки информации между микросхемами. Обработкой данных обычно занимаются цифровые сигнальные процессоры (DSP — Digital Signal Processor), поэтому последовательные порты, предназначенные для передачи цифровой информации с максимальной скоростью, часто называются DSP-портами.

Какие данные могут образовывать непрерывные потоки? Чаще всего это звуковые или видеосигналы, хотя в качестве сигнала, образующего непрерывный поток данных, могут выступать сигналы измерения биотоков живого организма или сигналы геомагнитных измерений. Из определения природы таких сигналов видно, что источником цифрового потока должен быть аналого-цифровой преобразователь (АЦП). Если микросхема (не сигнальный процессор) является приемником цифрового потока данных, то в ее состав обычно входит цифроаналоговый преобразователь (ЦАП).

При преобразовании аналоговой информации в цифровую форму и наоборот очень важно обеспечить стабильность тактового сигнала. Только в этом случае можно избежать искажений исходного или формируемого сигнала. Такой сигнал тактовой синхронизации обычно вырабатывается специальными термостабилизированными (или термокомпенсированными) высокостабильными кварцевыми генераторами, питаемыми от отдельного стабилизатора питания.

Скорость передачи информации в цифроаналоговый преобразователь или из аналого-цифрового преобразователя должна быть строго согласована по времени со скоростью передачи по каналу связи (последовательному порту). Именно поэтому сигналы синхронизации, необходимые для работы DSP-порта, вырабатываются из входного высокостабильного тактового сигнала самой микросхемой аналого-цифрового или цифроаналогового преобразователя и подаются на сигнальный процессор или программируемую логическую интегральную схему.

Проще всего преобразовать число, представленное в параллельном коде, в последовательный вид можно при помощи сдвигового регистра. При этом на приемном конце важно знать момент, когда производится запись в сдвиговый регистр. Знание этого момента времени позволит определить, какой из передаваемых бит является старшим значащим разрядом, а какой — младшим. Так как поток двоичной информации передается постоянно, то сигналы записи в регистр будут подаваться с одним и тем же периодом.

Выходной двоичный код бит за битом можно снимать с последнего выхода сдвигового регистра. Для того чтобы на приемном конце этот сигнал прини-

мался без ошибок, каждый бит должен сопровождаться синхронизирующим импульсом. При этом чтобы отличать синхросигналы друг от друга, импульсы, сопровождающие информационные биты, стали называть тактовой синхронизацией (CLK — clock), а сигнал, отмечающий момент записи в регистр, — кадровой синхронизацией (FS — frame synchronization).

Сигналы кадровой и тактовой синхронизации должны быть жестко связаны между собой, поэтому они обычно формируются из одного высокостабильного колебания при помощи цифрового счетчика.

Давайте рассмотрим пример передачи восьмиразрядного последовательного слова. В качестве источника цифрового потока используем аналого-цифровой преобразователь. Получившаяся в результате схема синхронного последовательного порта приведена на рис. 11.1.

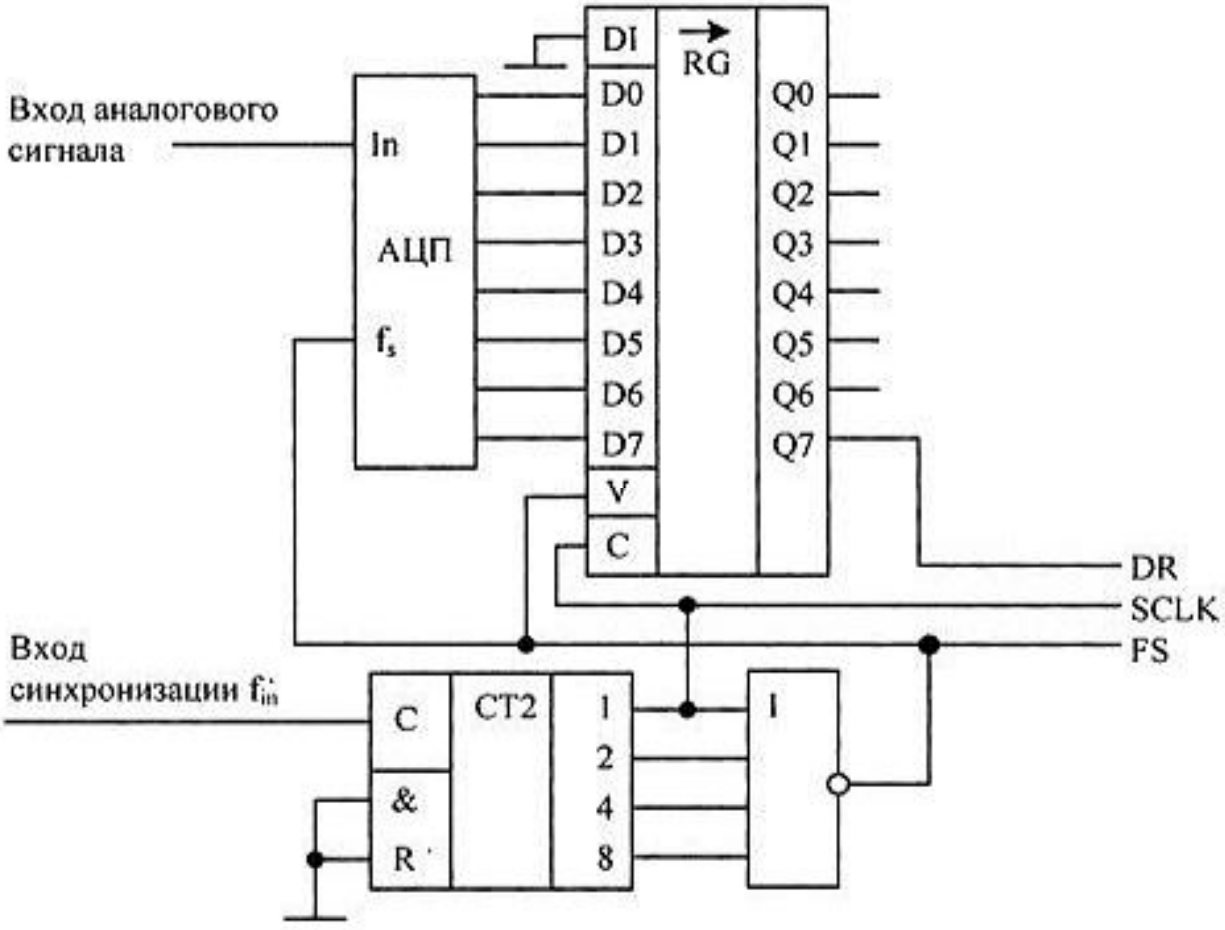


Рис. 11.1. Схема передающей части последовательного DSP-порта

Как видно из приведенной схемы, на вход двоичного счетчика подаются импульсы от генератора высокостабильных колебаний. Из этих импульсов вырабатывается сигнал тактовой синхронизации, который подается на выход схемы и одновременно поступает на вход синхронизации передающего сдвигового регистра.

Логический элемент "4ИЛИ-НЕ", подключенный к выходу четырехразрядного двоичного счетчика, формирует импульсы кадровой синхронизации FS с частотой в восемь раз меньшей частоты тактовой синхронизации SCLK — serial clock.

Такую частоту можно было бы получить с выхода "8" двоичного счетчика, но нам требуется длительность импульса, равная длительности импульса тактовой синхронизации SCLK. Логический элемент "4ИЛИ" позволяет декодировать нулевое состояние счетчика. В результате этого на его выходе длительность импульса равна длительности импульсов тактовой синхронизации, а сам импульс появляется в самом начале кадра передачи данных с выхода аналого-цифрового преобразователя (АЦП).

В свою очередь длительность импульсов на выводе SCLK последовательного порта равна периоду сигнала синхронизации всей схемы, т. к. этот сигнал снимается с выхода младшего разряда двоичного счетчика.

Кадр передачи данных начинается с параллельной записи результата преобразования АЦП в передающий сдвиговый регистр. Для этого импульс кадровой синхронизации подается на вход параллельной записи в регистр сдвига. В приведенной на рис. 11.1 схеме предполагается, что запись в регистр осуществляется по нарастающему фронту этого импульса.

Этот же импульс кадровой синхронизации подается на вход синхронизации аналого-цифрового преобразователя. Внутренняя схема АЦП выполнена так, чтобы аналого-цифровое преобразование начиналось по спадающему фронту импульса. Такой подбор схем АЦП и регистра сдвига позволяет сначала записать результат предыдущего преобразования в сдвиговый регистр, а затем начать новое преобразование аналогового сигнала в цифровую форму по одному и тому же импульсу синхронизации.

Пример временных диаграмм сигналов данных и сопровождающих их сигналов тактовой и кадровой синхронизации на выходе последовательного DSP-порта приведен на рис. 11.2.

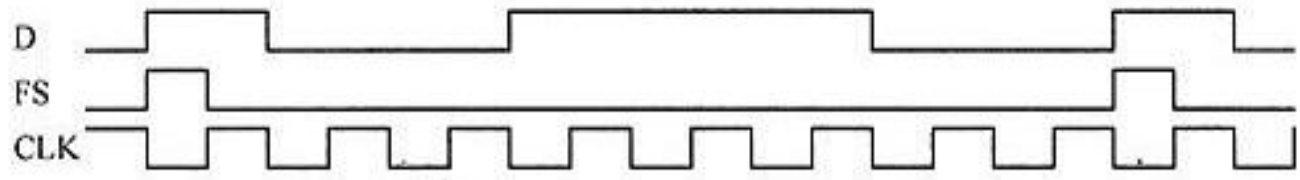


Рис. 11.2. Временные диаграммы сигналов на выходе синхронного последовательного DSP-порта

В приведенных на рис. 11.2 временных диаграммах осуществляется передача двоичного числа 10011100_2. При этом запись информации в последовательный регистр производится по сигналу кадровой синхронизации FS. В этот

момент на выходе Q7 регистра сдвига появляется старший бит результата преобразования. С этого момента начинают отсчитываться импульсы тактовой синхронизации CLK.

Приведенный на схеме рис. 11.1 регистр осуществляет сдвиг своего содержимого по спадающему фронту этих импульсов. В результате на его выходе Q7, а значит, и на выводе порта DR, последовательно один за другим будут появляться биты передаваемого двоичного числа, соответствующего результату работы АЦП.

Для безошибочной передачи информации последовательного порта по соединительным линиям, в приемный регистр эти данные должны записываться по нарастающему фронту импульсов тактовой синхронизации SCLK, т. к. именно этот фронт совпадает с серединой битового интервала сигнала на выводе DR. Это позволяет избежать влияния искажений, возникающих на фронтах передаваемой информации.

Как уже говорилось в начале главы, DSP-порты предназначены для передачи данных между аналого-цифровыми (или цифроаналоговыми) преобразователями и сигнальными процессорами. В качестве примера использования SSI-интерфейса рассмотрим схему соединения между собой аналого-цифрового преобразователя AD7890 и сигнального процессора TMS320C25. Упрощенная структурная схема соединения этих микросхем приведена на рис. 11.3.

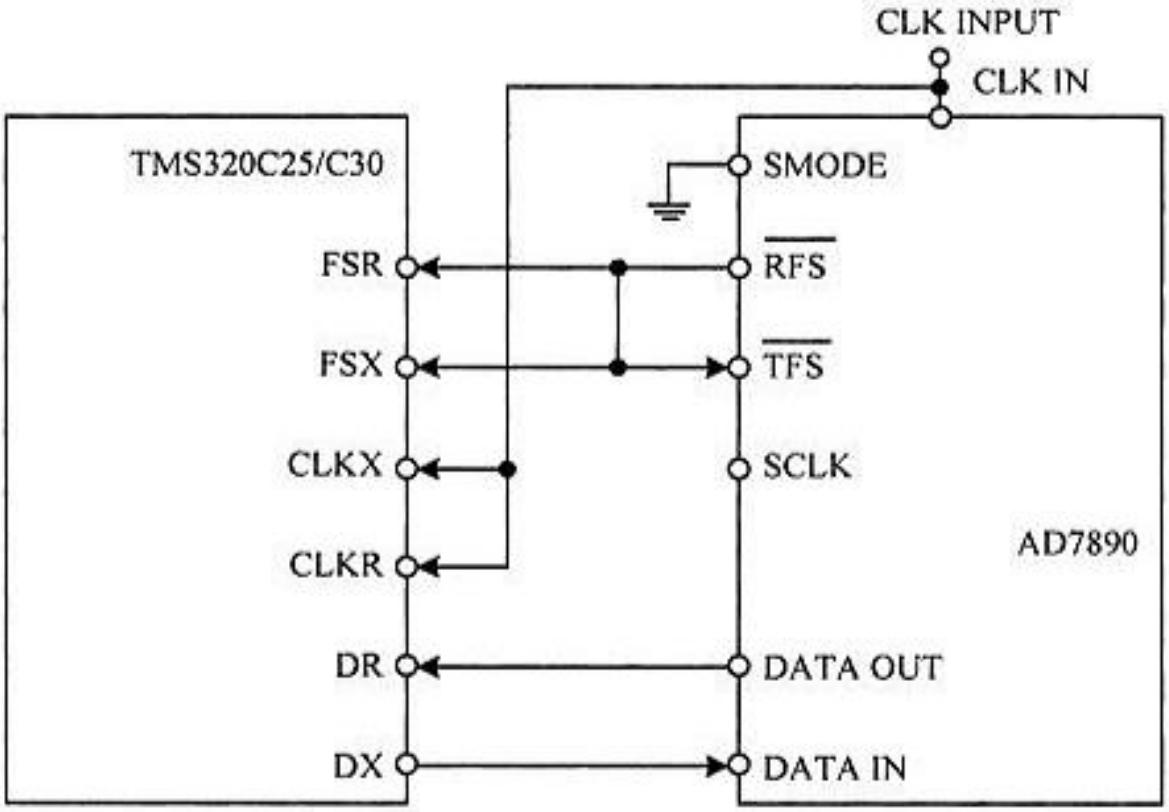


Рис. 11.3. Упрощенная схема соединения микросхемы АЦП с последовательным выходом и сигнального процессора

На этой схеме используется двунаправленный последовательный порт сигнального процессора. Синхронизация всей схемы осуществляется от внешнего высокостабильного генератора импульсов (вход CLK INPUT). Так как порт двунаправленный, то у него есть два сигнала кадровой синхронизации: RFS (receiver frame synchronization) — сигнал кадровой синхронизации приемника и TFS (transmitter frame synchronization) — сигнал кадровой синхронизации передатчика. У сигнального процессора TMS320C25 им соответствуют два вывода кадровой синхронизации: FSR (frame synchronization receiver) — сигнал кадровой синхронизации приемника и FSX — сигнал кадровой синхронизации передатчика синхронного последовательного порта.

В приведенной на рис. 11.3 схеме используется одна и та же скорость передачи данных, как в приемнике, так и в передатчике последовательного порта (интерфейса). Именно поэтому все выводы кадровой синхронизации соединены между собой и все синхронизируются от сигнала кадровой синхронизации, формируемого микросхемой аналого-цифрового преобразователя AD7890 на выводе RFS.

В качестве сигнала тактовой синхронизации последовательного порта используется сигнал внешнего генератора опорной частоты. Он подается на входы тактовой синхронизации сигнального процессора как приемника CLKR (clock receiver), так и передатчика CLKX.

Выводы передачи данных соединены соответственно своему назначению, вывод передатчика АЦП DATA OUT соединен с входом приемника сигнального процессора DR, а вывод передатчика сигнального процессора DT соединен с входом приемника АЦП DATA IN.

В качестве еще одного примера использования сдвиговых регистров в современной цифровой технике давайте рассмотрим один широко распространенный синхронный последовательный порт — SPI-интерфейс.

SPI-порт

Кроме задачи передачи непрерывного потока информации довольно часто требуется передавать отдельные цифровые пакеты данных или управляющие команды. Эти пакеты могут передаваться достаточно редко. Именно для передачи такого вида информации и предназначен синхронный последовательный интерфейс (SPI — serial peripheral interface).

В синхронном последовательном интерфейсе синхронизирующие импульсы не передаются постоянно. Это не нужно. Они присутствуют только в момент передачи управляющей команды или цифрового пакета данных. Соответственно новой задаче меняется и название сигналов тактовой и кадровой синхронизации. В SPI-интерфейсе сигнал кадровой синхронизации называется

выбор ведомого (slave select — SS). Сигнал тактовой синхронизации в этом интерфейсе получил название SCLOCK — serial clock (последовательная синхронизация). Временные диаграммы сигналов на выводах SPI-порта приведены на рис. 11.4.

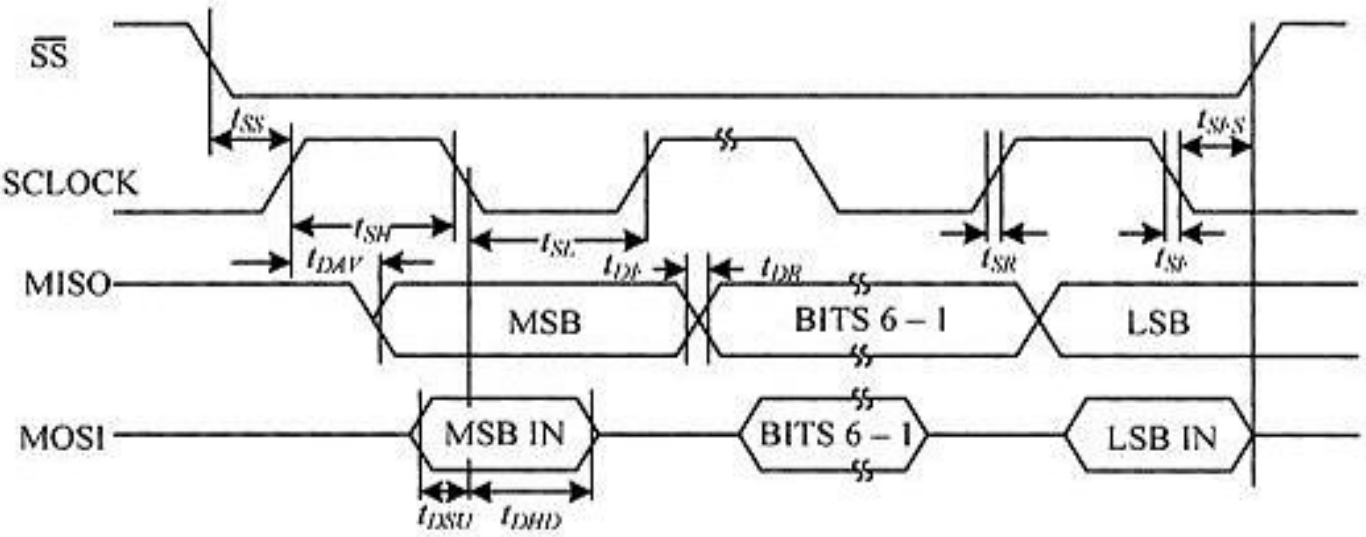


Рис. 11.4. Временные диаграммы сигналов на выводах синхронного последовательного интерфейса SPI

Как видно из приведенных на рис. 11.4 временных диаграмм, сигнал кадровой синхронизации присутствует во время всего промежутка времени, предназначенного для обмена информацией. Он разрешает этот обмен. Отсюда и название этого сигнала.

Сигнал тактовой синхронизации SCLOCK в SPI-интерфейсе используется один как для передатчика, так и для приемника. Это позволяет экономить внешние выводы микросхемы. Для того чтобы не запутаться, какой вывод передачи последовательных данных с каким выводом приемника соединять, соответствующая информация включена в название выводов. Название MISO обозначает вход главного устройства и выход подчиненного (master input — slave output), а название MOSI обозначает выход главного и вход подчиненного устройства (master output — slave input). В качестве главного устройства в этом интерфейсе обычно применяется микроконтроллер, реже сигнальный процессор. Обозначение MSB на временной диаграмме означает старший значащий бит, LSB — младший значащий бит, BITS — остальные передаваемые биты.

Рассмотрим принципиальную схему цифрового устройства, которая может реализовать обмен данными по интерфейсу SPI. Пример простейшего варианта схемы внутреннего устройства синхронного последовательного интерфейса приведен на рис. 11.5.

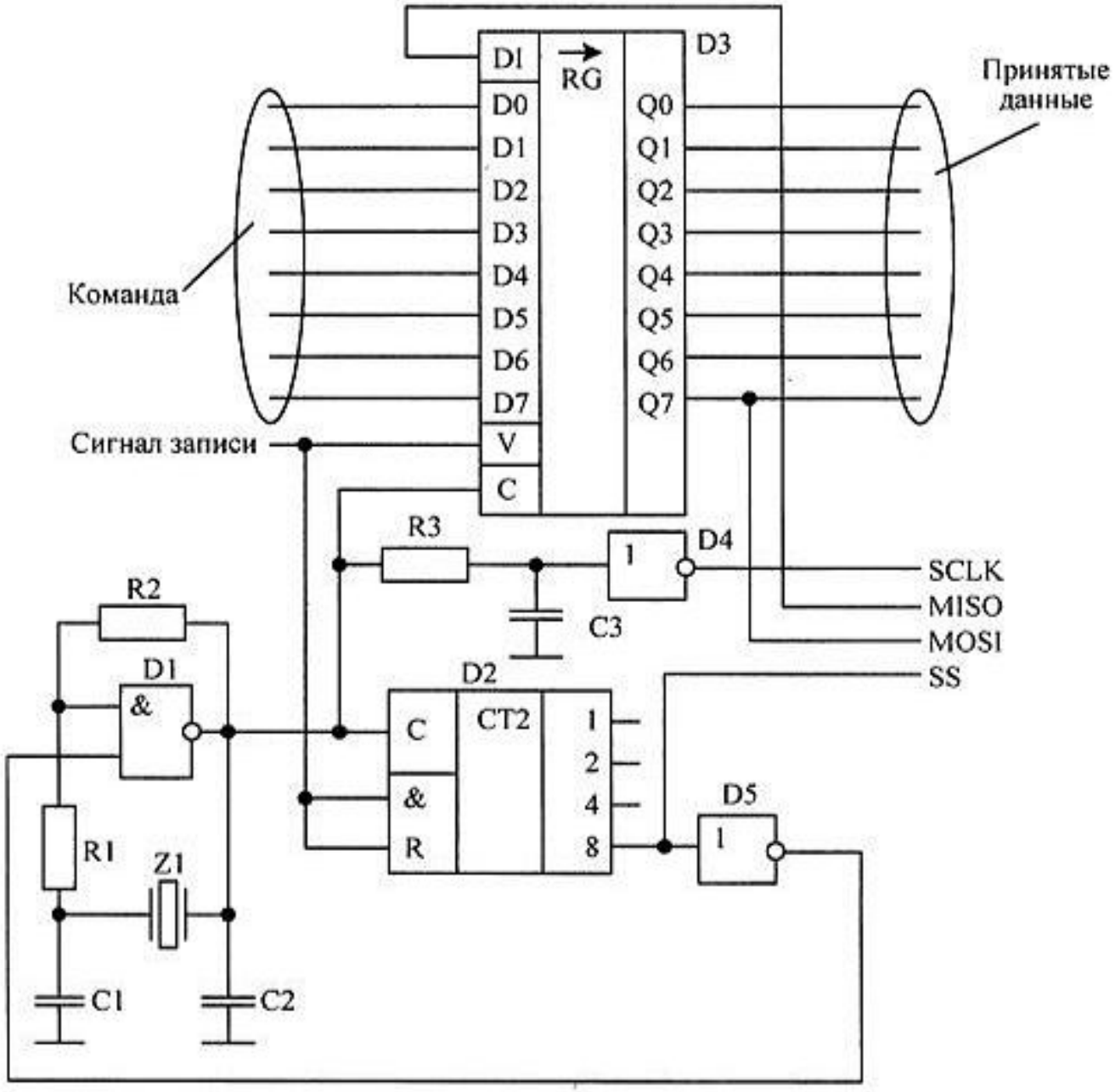


Рис. 11.5. Принципиальная схема master SPI-порта

В рассматриваемой схеме для передачи и приема последовательных данных используется сдвиговый регистр D3. В начале передачи данных по сигналу записи в регистр D3 записывается передаваемая по линии MOSI управляющая команда (при необходимости с сопутствующими данными). В момент передачи команды ее отдельные биты последовательно появляются на выходе этого сдвигового регистра (вывод Q7). Одновременно принимаемая по линии MISO информация записывается в первый триггер этого же регистра и постепенно замещает передаваемые данные. В момент окончания передачи в регистре D3 будут храниться данные, принятые по линии MISO.

Так как синхроимпульсы SPI-порта, в отличие от синхросигнала аналого-цифрового или цифроаналогового преобразователей, используются только

для синхронизации передачи данных, то требования к параметрам сигнала задающего генератора этого порта меньше. В результате для формирования синхронизирующего сигнала можно использовать встроенный кварцевый генератор (кварцевый резонатор, естественно, будет подключаться к внешним выводам микросхемы).

Кварцевый генератор в схеме, приведенной на рис. 11.5, построен на логическом элементе "2И-НЕ". Это позволяет останавливать и вновь запускать этот генератор. Для формирования сигнала выбора ведомого SS и для подсчета необходимого количества импульсов синхронизации (в нашем случае — восьми) служит двоичный счетчик D2.

Рассмотрим работу схемы формирования сигнала "выбор ведомого" подробнее. В исходном состоянии в двоичном счетчике записано число 1000_2. При этом на выводе последовательного интерфейса SS и на выходе логического элемента D1 присутствует высокий потенциал.

При параллельной записи данных в регистр передачи D3 импульс записи одновременно подается на вход обнуления счетчика D2. В результате обнуления содержимого счетчика D2, на выводе SS появляется низкий потенциал. Это означает, что SPI-интерфейс начинает передачу данных. Одновременно снимается запрещающий потенциал с логического элемента D1 "2И-НЕ". При этом на обоих входах этого элемента появляется единичный потенциал. В результате на выходе этого элемента появится нулевой потенциал и возникнут условия для самовозбуждения генератора.

Временные диаграммы на входе и выходах счетчика D2 приведены на рис. 11.6.

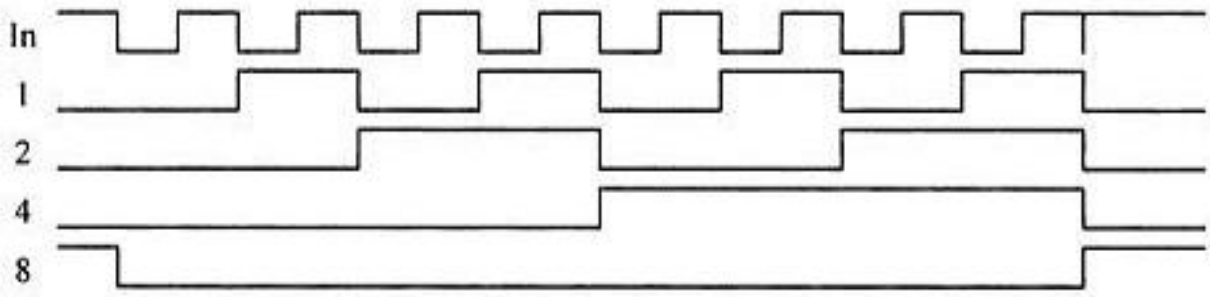


Рис. 11.6. Временные диаграммы
схемы формирования сигнала выбора подчиненного SS

Импульсы поступают на вход последовательной синхронизации сдвигового регистра D3 и вход счетчика D2. После поступления на вход схемы D3 восьмого импульса передача данных в последовательном виде завершается, а в счетчике D2 оказывается записанным число 8. В двоичном виде оно равно значению 1000_2, а значит, на выводе SS вновь появится высокий потенциал,

означающий завершение передачи данных по синхронному последовательному интерфейсу SPI.

Так как этот же сигнал через инвертор подается на вход логического элемента D1, то на его выходе появится единичный потенциал, и условия генерации сорвутся. Генератор больше не будет вырабатывать синхронизирующие импульсы, а значит, двоичный счетчик останется в состоянии 1000_2 до прихода следующего импульса записи очередной управляющей команды в SPI-порт.

Интегрирующая цепочка R3C3 в приведенной схеме используется для подавления короткого импульса в конце формирования сигнала SS, а инвертор D4 позволяет осуществлять запись в сдвиговый регистр D6 по спадающему фронту сигнала тактовой синхронизации. В результате этого подчиненное устройство успевает подготовить очередной бит передаваемых данных.

Мы рассмотрели схему SPI-интерфейса со стороны главного устройства. Схема подчиненного устройства намного проще. В ней не нужно формировать сигналы синхронизации. В результате она может быть построена на обыкновенном сдвиговом регистре. Пример выполнения схемы подчиненного SPI-порта приведен на рис. 11.7.

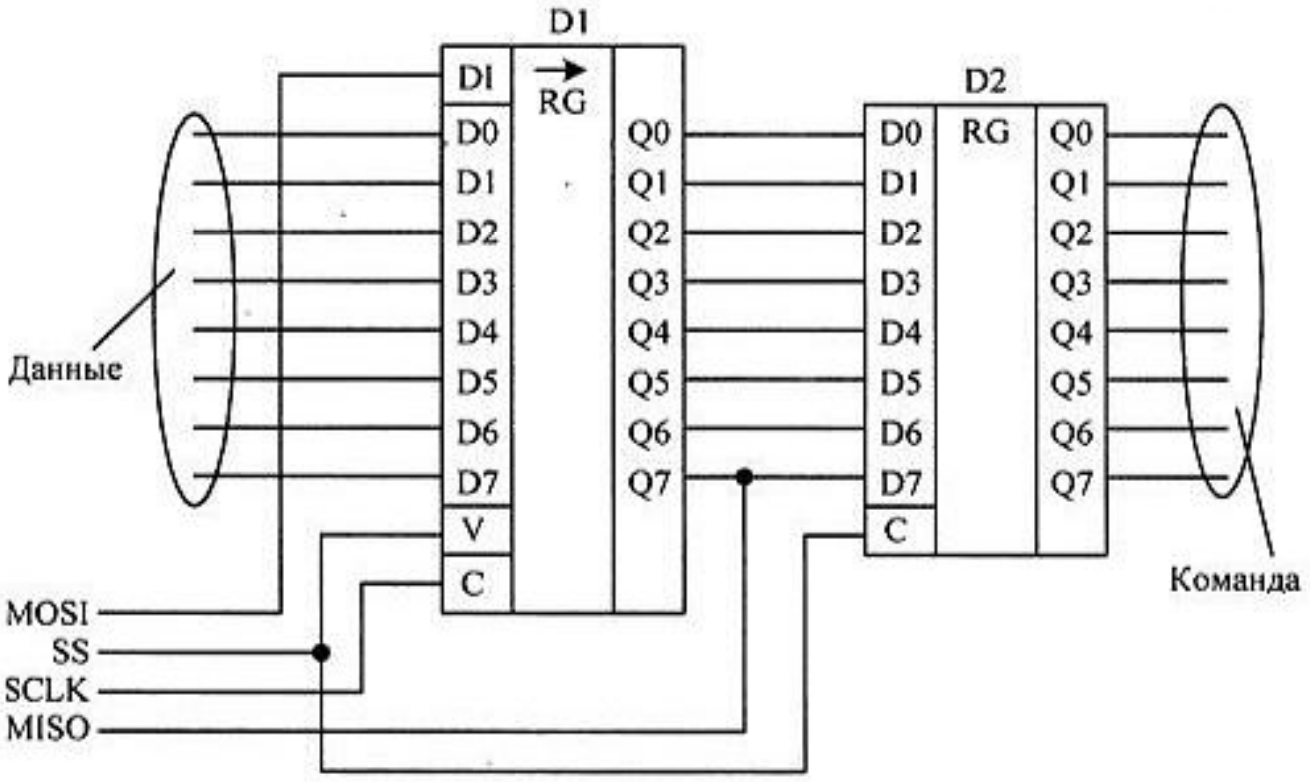


Рис. 11.7. Схема подчиненного SPI-порта

В этой схеме запись данных в универсальный регистр D1 производится по спадающему фронту сигнала SS. В результате на выходе Q7 появляется первый из передаваемых бит. Это значение по цепи MISO передается на вход

данных ведущего (главного) порта. При поступлении на вход "C" регистра D1 тактовых импульсов SCLK на выходе Q7 последовательно появляются все биты данных, присутствовавших на параллельных входах универсального регистра в момент записи.

Одновременно с этим процессом данные, передаваемые ведущим портом по цепи MISO, записываются в триггеры регистра D1. После поступления восьмитактовых импульсов SCLK на выходах сдвигового регистра D1 вместо данных подчиненного устройства уже записаны данные, переданные веду-

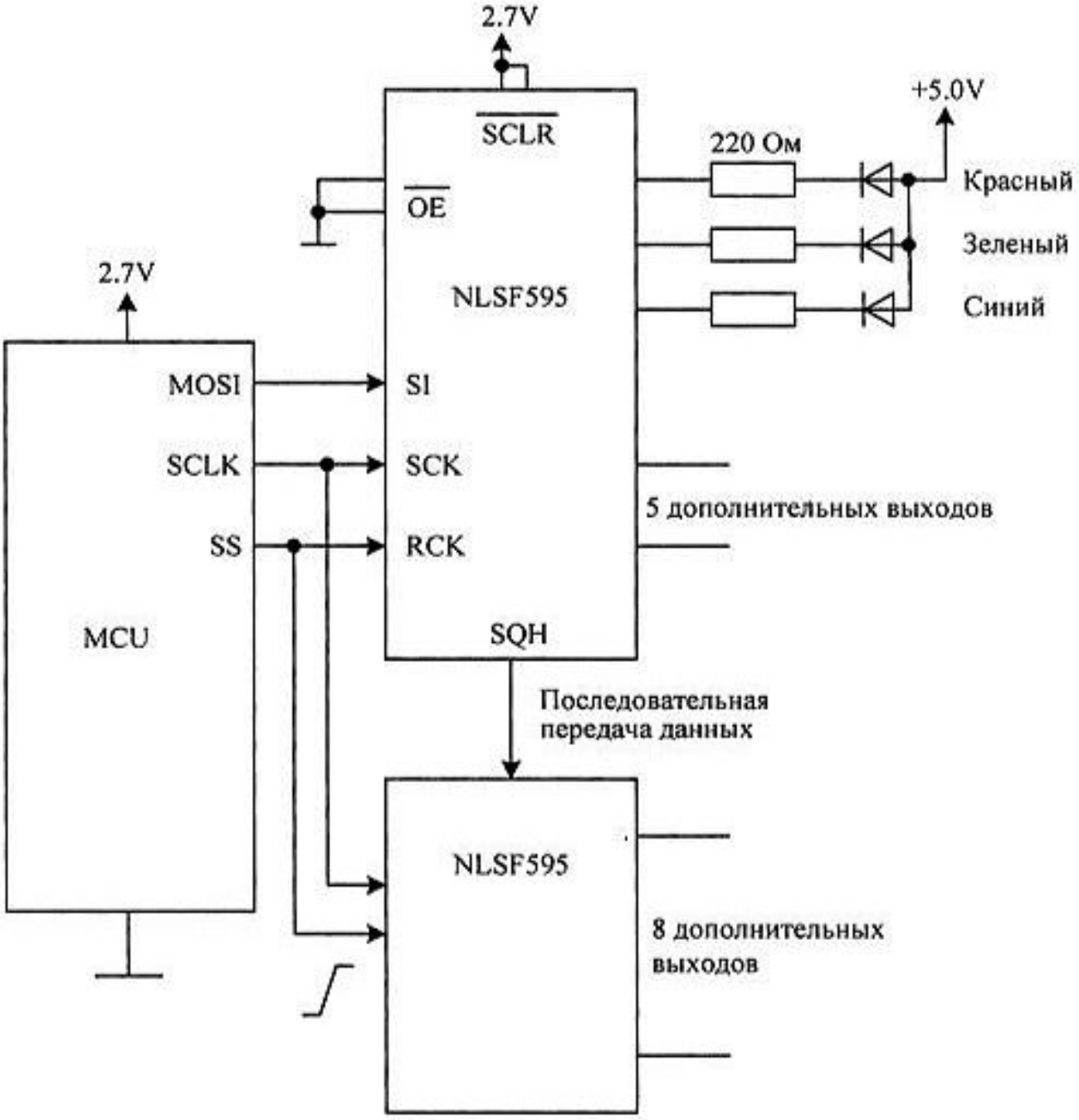


Рис. 11.8. Структурная схема соединения микроконтроллера и расширителя портов NLSF595

щим портом. Обмен данными завершается выдачей высокого потенциала по цепи SS. Тем самым формируется нарастающий фронт этого сигнала. По этому фронту данные с выходов последовательного регистра D1 переписываются в параллельный регистр D2.

В настоящее время SPI-порт является распространенным средством обмена информации между процессором и микросхемами обработки аналоговой и цифровой информации. Обычно этот порт уже содержится в составе этих микросхем в виде готового модуля.

В качестве примера использования SPI-порта на рис. 11.8 приведена структурная схема соединения микроконтроллера и микросхемы расширителя портов NLSF595 фирмы ON semiconductor, к которой на приведенной схеме подключены три светодиода. Эта микросхема способна только принимать информацию по SPI-порту и выдавать ее в параллельном виде. Получать дискретную информацию в параллельном виде и передавать ее микропроцессору она не может.

В качестве еще одного примера использования рассмотренного выше SPI-порта, на рис. 11.9 приведена структурная схема соединения сигнального процессора ADSP-2153x и микросхемы цифрового тракта промежуточной частоты AD9874 фирмы Analog devices, предназначенной для реализации радиоприемных устройств в цифровом виде.

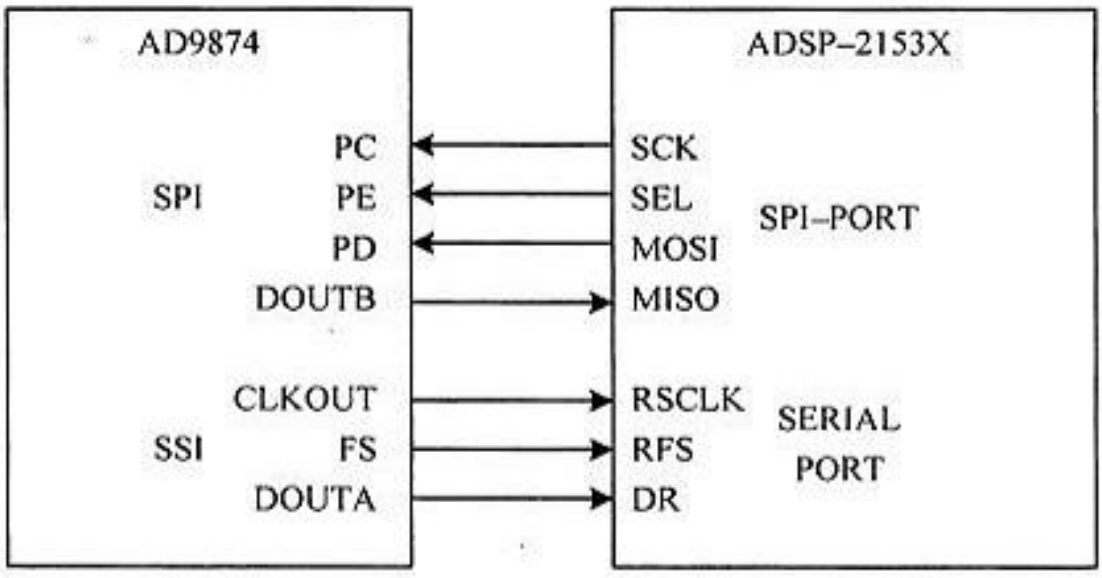


Рис. 11.9. Структурная схема соединения сигнального процессора ADSP-2153x и микросхемы цифрового тракта промежуточной частоты AD9874

В этой схеме, взятой из руководства по применению цифрового приемника AD9874, использовано два порта — DSP-порт для передачи потока данных и SPI-порт для настройки режимов работы микросхемы AD9874.

Синхронный последовательный интерфейс использует меньшее количество проводников по сравнению с DSP-портом, однако это количество все еще достаточно велико. В некоторых случаях этот параметр является критическим. Тогда применяется I^2C-порт.

I^2C-порт

Для обмена данными по I^2C — Inter-Integrated Circuit порту, который часто называется I^2C-шиной, используется всего два проводника (не считая корпуса).

В I^2C-шине прием и передача данных, а также передача адреса микросхемы и адреса регистра внутри микросхемы, к которому осуществляется обращение, производится по одному и тому же проводу. Для подключения к этому проводу используются микросхемы с открытым коллектором. Нагрузкой для всех микросхем, подключенных к линии SDA, служит внешний резистор. В результате общее количество проводников, требующихся для реализации данного последовательного порта, уменьшается до двух. Сокращение количества проводников позволяет уменьшить площадь печатной платы и размеры корпуса микросхемы (чем меньше количество выводов микросхемы, тем меньше ее габариты).

Естественно, что скорость передачи данных по такому порту будет ниже, по сравнению с SPI-портом, тем не менее, там где на первом месте стоит такой параметр, как габариты устройства, последовательный интерфейс I^2C незаменим. Тактовая синхронизация в I^2C-шине передается по линии SCL. Начало работы с микросхемой обозначается особой комбинацией сигналов SDA и SCL, которая называется условием старта. Эта же комбинация одновременно осуществляет кадровую синхронизацию. Завершение работы с микросхемой обозначается еще одной комбинацией сигналов SDA и SCL. Временная диаграмма последовательного порта I^2C приведена на рис. 11.10.

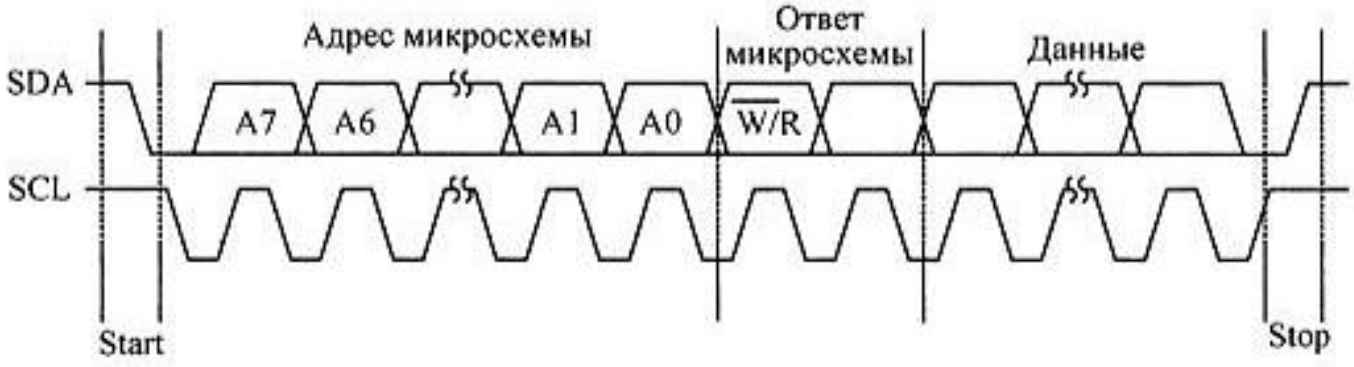


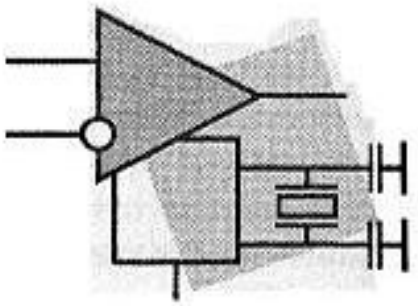
Рис. 11.10. Временная диаграмма I^2C-интерфейса

Внутренняя схема порта I^2C достаточно сложна. Раскрывать ее внутреннее устройство в качестве учебного примера использования цифровых схем не имеет смысла. Тем не менее знаний, полученных в процессе изучения первой части данной книги, вполне достаточно для реализации схемы данного последовательного порта. Этот порт в современной технике не менее распространен по сравнению с остальными видами синхронных последовательных портов. В качестве примера микросхем, использующих интерфейс I^2C, можно назвать микросхемы EEPROM, серии 24cXX.

Итоги

В данной главе мы рассмотрели несколько примеров проектирования цифровых блоков. В качестве учебных блоков были выбраны наиболее распространенные узлы современных интегральных схем. Естественно, проектировать последовательные порты обычно не имеет смысла (если только вы не проектируете цифровое устройство на ПЛИС), но понимать, как они работают в составе готовых микросхем, очень полезно. На этом можно завершить изучение синхронных последовательных портов. В качестве одного из наиболее распространенных цифровых устройств, использующего для настройки своего режима работы синхронный последовательный интерфейс SPI, рассмотрим синтезаторы частоты.

ГЛАВА 12



Синтезаторы частоты

В современных системах радиосвязи требуется формировать опорные колебания с различной частотой. При этом сталкиваются между собой требования обеспечения перестройки частоты формируемого колебания в достаточно широких пределах и стабильности его частоты. Стабильность частоты опорного колебания можно обеспечить при помощи генератора с кварцевой стабилизацией частоты, схема которого была рассмотрена ранее, однако при этом теряется возможность перестройки по заданному диапазону частот.

Можно реализовать перестраиваемый генератор, например, RC-генератор с настройкой частоты резистором с переменным сопротивлением, или LC-генератор с настройкой частоты колебательного контура варикапом, но при этом будет проблематично получить стабильность формируемых колебаний лучше 10^{-2} для RC-генератора и 10^{-3} для LC-генератора.

Из-за описанных проблем в случаях, когда требовалась высокая стабильность частоты, в течение определенного периода времени производилась аппаратура с генераторами кварц — частота. То есть в такой аппаратуре применялось столько кварцевых генераторов, сколько частотных каналов использовалось в этой аппаратуре.

В настоящее время удается объединить преимущества перестраиваемого генератора частоты и кварцевого генератора при помощи методов цифровой техники. Генераторы, способные осуществлять перестройку в широком диапазоне частот и при этом обладающие стабильностью кварцевых генераторов, получили название синтезаторов частот.

Синтезаторы частоты не способны осуществлять плавную перестройку частоты. Они могут формировать частоты только из определенного набора значений. Однако шаг перестройки частоты синтезатора может быть сделан сколь угодно малым.

Не следует ограничивать область применения синтезаторов частот только устройствами передачи или приема сигналов. Внутри достаточно большого числа цифровых микросхем используются частоты синхронизации, в несколько раз превышающие частоту опорного колебания, поступающего на их тактовый вход. Синтезаторы частоты внутри этих микросхем используются для увеличения частоты входного колебания.

При использовании синтезаторов частот для формирования тактовой частоты в ряде случаев полезным свойством является возможность программно изменять скорость работы микросхемы. Это может приводить к существенной экономии потребляемого микросхемой тока или снижать уровень помех, производимых данной микросхемой.

При таком применении синтезатора частот обычно достаточно всего нескольких ступеней регулировки внутренней тактовой частоты микросхемы. В качестве наиболее распространенного примера подобной микросхемы можно привести центральный процессор персонального компьютера, где при помощи программы BIOS можно изменять скорость его работы.

Первоначально синтезаторы частот выполнялись на основе генераторов гармоник. Современные цифровые синтезаторы частот обычно строятся с использованием схемы автоматической подстройки частоты.

Систему автоматической подстройки частоты в простейшем случае можно рассматривать как систему с отрицательной обратной связью, обладающую коэффициентом усиления в прямом направлении G(s) и коэффициентом передачи цепи обратной связи H(s). Структурная схема системы, охваченной отрицательной обратной связью по напряжению, приведена на рис. 12.1.

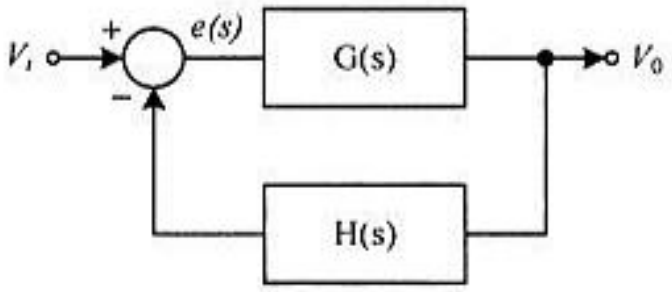


Рис. 12.1. Модель управляющей системы, охваченной отрицательной обратной связью

Передаточная функция системы, охваченной цепью отрицательной обратной связи, обычно записывается в виде следующих выражений:

$$G_{ОС} = \frac{G(s)}{1 + G(s) \times H(s)},$$

где $G_{ОС}$ — петлевой коэффициент передачи цепи обратной связи;

$G(s)$ — коэффициент усиления в прямом направлении;

$H(s)$ — коэффициент передачи цепи обратной связи.

$$S = j\omega = j2\pi f ,$$

где j — мнимая единица;

ω — круговая частота.

Сигнал ошибки формирования частоты *e(s)* в этой схеме может формироваться как частотным, так и фазовым детектором. Может возникнуть вопрос — при чем тут фазовый детектор, если мы собрались строить схему автоматической подстройки частоты?

Чтобы на него ответить, давайте вспомним, что мы называем частотой. По определению, частотой называется количество колебаний, вырабатываемое генератором в единицу времени (обычно в секунду). Теперь вспомним, как определить период колебания. Для этого необходимо определить в форме колебания одинаковое значение, которое называется фазой колебания.

Теперь частоту можно определить по формуле:

$$f = \frac{\Delta\phi}{\Delta t}$$

Это означает, что частота и фаза однозначно связаны между собой через операцию дифференцирования для частоты или операцию интегрирования для фазы. То есть для подстройки частоты генератора можно использовать как сигнал ошибки по частоте, так и сигнал ошибки по фазе.

Рассмотрим оба варианта формирования сигнала ошибки. Сначала рассмотрим вариант использования частотного детектора. При использовании частотного детектора сигнал ошибки будет пропорционален частоте, а это означает, что для настройки генератора управляемого напряжением на нужную нам частоту принципиально потребуется ошибка по частоте. Эту ошибку можно уменьшить за счет увеличения петлевого коэффициента передачи обратной связи, однако свести ее к нулю принципиально нельзя. Более того, эта ошибка будет зависеть от частоты настройки синтезатора, ведь только так можно обеспечить изменение напряжения на входе генератора управляемого напряжением при смене рабочей частоты.

Теперь рассмотрим вариант использования фазового детектора. В этом случае напряжение на выходе детектора будет зависеть от разности фаз опорного и формируемого колебаний. При различии частот опорного и формируемого колебаний разность фаз будет постоянно изменяться от 0 до 360°.

Вместе с разностью фаз с течением времени будет меняться и напряжение на входе генератора управляемого напряжением, а это означает, что рано или поздно частоты сравниваемых колебаний сравняются, и разность фаз больше не будет изменяться.

В схеме с подстройкой фазы выходного колебания напряжение на выходе фазового детектора будет изменяться при изменении частоты настройки синтезатора, т. е. будет меняться их разность фаз, однако ошибка по частоте при этом будет равна нулю. Именно поэтому современные синтезаторы частот строятся на базе цепей фазовой подстройки частоты. Другие схемы в настоящее время практически не используются. Поэтому остановимся на схеме фазовой подстройки частоты подробнее.

Схемы фазовой подстройки частоты

Рассмотрим основные блоки, входящие в структурную схему фазовой автоматической подстройки частоты (ФАПЧ). Структурная схема ФАПЧ приведена на рис. 12.2.

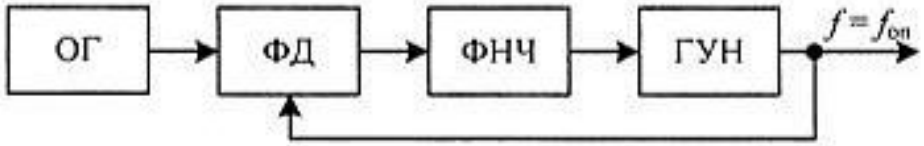


Рис. 12.2. Структурная схема цепи фазовой автоподстройки частоты

В состав этой структурной схемы входит фазовый детектор (ФД), формирующий сигнал ошибки формируемого колебания. Выходное колебание вырабатывается генератором, управляемым напряжением (ГУН). Образцовое колебание в этой схеме формирует опорный генератор (ОГ). Еще одним неотъемлемым звеном цепи фазовой автоподстройки частоты является фильтр нижних частот (ФНЧ), позволяющий избежать самовозбуждения всей схемы в целом.

В зависимости от элементов, использованных в схеме фазовой автоподстройки частоты, она может быть *аналоговой* (при использовании аналоговых схем фазового детектора), *цифровой* (при использовании в качестве фазового детектора логических цепей) и *полностью цифровой* (при реализации фильтра низкой частоты в цифровом виде).

В результате работы схемы, приведенной на рис. 12.2, мы в идеальном случае можем получить точно такое же колебание, что и колебание опорного генератора. Но тогда зачем нужна вся схема? Ведь можно было бы просто взять сигнал с выхода опорного генератора.

Первая задача, которую можно решить при использовании схемы фазовой автоматической подстройки частоты, — это реализация детектирования частотно-модулированного сигнала. Если снимать напряжение с выхода ФНЧ, входящего в состав схемы фазовой автоподстройки частоты, то его уровень будет пропорционален отклонению частоты опорного генератора от номинального значения.

Однако мы собирались использовать схему ФАПЧ для генерации заданного набора частот. То есть нам требуется научиться изменять частоту генератора управляемого напряжением. Для этого включим в цепь обратной связи делитель частоты, как это показано на рис. 12.3. Частота сигнала на выходе этого делителя уменьшится по сравнению с входным значением в коэффициент деления раз. Но ведь схема фазовой автоподстройки частоты будет поддерживать значения частот на входе фазового детектора равными друг другу. Это означает, что частота на выходе ГУН под действием цепи автоматической подстройки должна будет увеличиться в коэффициент деления раз относительно частоты опорного колебания. *Схема ФАПЧ работает как умножитель опорной частоты.*

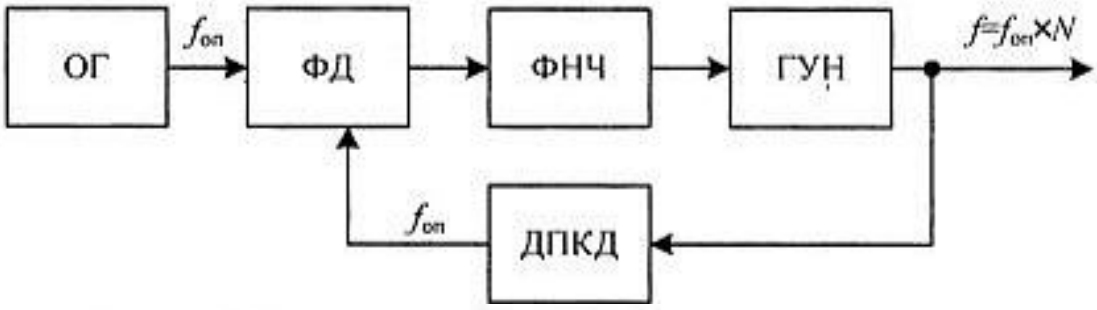


Рис. 12.3. Структурная схема цифрового синтезатора частот

В структурной схеме, приведенной на рис. 12.3, использован делитель с переменным коэффициентом деления (ДПКД). Изменяя коэффициент деления N делителя ДПКД, можно перестраивать выходную частоту генератора.

Как мы уже знаем из предыдущих глав, коэффициент деления цифрового делителя частоты может достигать значения в несколько тысяч. Выбрав достаточно низкую опорную частоту $f_{оп}$ можно получить шаг перестройки синтезатора, удовлетворяющий требованиям к перестраиваемому генератору частот. Шаг перестройки синтезатора в схеме ФАПЧ получается равным частоте опорного генератора.

Так как опорная частота в схеме, приведенной на рис. 12.3, используется для сравнения с частотой генератора, управляемого напряжением, то ее называют частотой сравнения. В результате действия цепи фазовой автоподстройки частоты частота ГУН в установившемся режиме в N раз больше частоты сравнения.

Обычно в радиотехнических схемах требуется малый шаг перестройки генератора. Величина этого шага составляет сотни герц или, в крайнем случае, единицы килогерц. В результате возникает новая проблема. Мы не можем использовать для формирования такой частоты кварцевый генератор, ведь приемлемые по габаритам и стоимости кварцевые резонаторы могут работать только в диапазоне частот от 1 до 30 МГц.

Тем не менее, для получения низкой частоты сравнения на входах фазового детектора, на выходе опорного генератора можно поставить еще один цифровой делитель частоты с постоянным коэффициентом деления, как это выполнено в схеме, приведенной на рис. 12.4. В этой схеме мы можем выбирать значения частот сравнения $f_{ср}$, опорной частоты $f_{оп}$ и выходного колебания f в достаточно широком диапазоне.

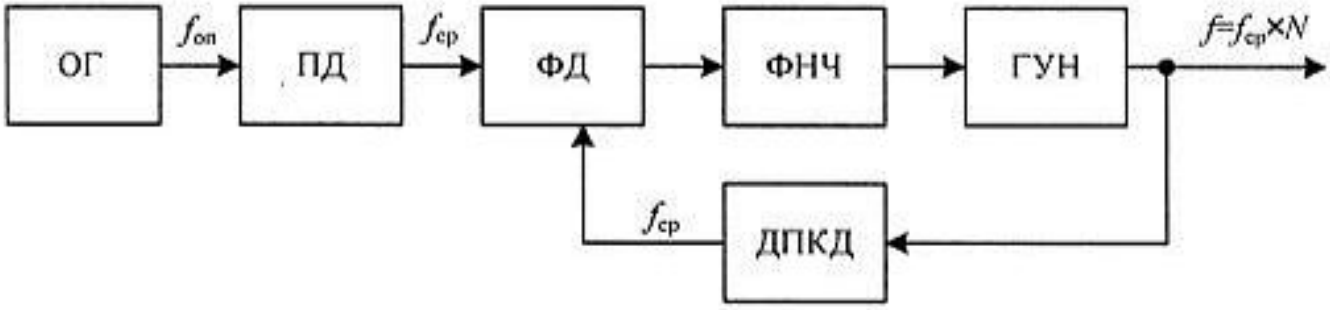


Рис. 12.4. Структурная схема цифрового синтезатора частот с малым шагом перестройки частоты

В качестве примера давайте определим требования к блокам, входящим в структурную схему синтезатора, вырабатывающего частоты в диапазоне от 146 до 174 МГц. Пусть в схеме будет использован генератор опорной частоты 6,4 МГц. Такие высокостабильные генераторы предлагаются многими фирмами в качестве готовых модулей, например модуль 6.4 MHz CFPT-9006-FC-1B фирмы C-MAC.

Шаг перестройки по частоте в заданном диапазоне частот определяется разносом радиоканалов по частоте. В настоящее время в этом диапазоне частот МККР (международный консультативный комитет по радио) рекомендует строить аппаратуру с шириной полосы радиоканала 12,5 кГц. Пусть наш синтезатор частот будет обладать именно таким шагом настройки частоты. Тогда частота сравнения на входе фазового детектора тоже должна соответствовать этому значению. Отсюда можно определить коэффициент деления постоянного делителя ПД:

$$K_{ПД} = \frac{f_{оп}}{f_{ср}} = \frac{6,4\times10^{6}}{12,5\times10^{3}} = 512$$

Теперь определим максимальное и минимальное значение коэффициентов деления ДПКД:

$$K_{min} = \frac{f_{min}}{f_{cp}} = \frac{146 \times 10^6}{12,5 \times 10^3} = 11680$$

$$K_{max} = \frac{f_{max}}{f_{cp}} = \frac{174 \times 10^6}{12,5 \times 10^3} = 13920$$

Все полученные коэффициенты деления легко реализуются одной из схем делителей частоты (цифровых счетчиков), рассмотренных нами в предыдущих главах. Теперь можно приступать к разработке принципиальной схемы синтезатора. Единственным блоком, не рассмотренным в предыдущих главах, остался блок определения ошибки по частоте. Остановимся на этом блоке подробнее.

Схемы определения ошибки по частоте

В качестве фазового детектора в цепи фазовой автоподстройки частоты могут быть использованы различные схемы. Схемы частотных и фазовых детекторов широко используются в аналоговой технике. Это могут быть хорошо известные схемы фазовых детекторов, построенные на диодах. В качестве фазового детектора прекрасно может работать аналоговый умножитель, хорошо известный в аналоговой схемотехнике.

Однако в большинстве случаев в таких схемах используются трансформаторы, что делает такие схемы неудобными для массового производства, поэтому имеет смысл попытаться найти решение при помощи цифровых микросхем.

Цифровой фазовый детектор

В аналоговых схемах наилучшими характеристиками обладают фазовые детекторы, построенные на основе умножителя. Составим таблицу истинности умножителя, сигнал на входе и выходе которого может принимать только два значения — единицу и минус единицу. Использование таких значений позволяет интерпретировать сигнал на входе умножителя как знак аналогового сигнала. Полученная таблица истинности умножителя знаков приведена в табл. 12.1.

Если теперь символ "–1" обозначить как "0", то мы увидим, что полученная таблица истинности совпадает с инвертированной таблицей истинности ло-

гического элемента "исключающее ИЛИ". Для сравнения в табл. 12.2 приведена таблица истинности логического элемента "исключающее ИЛИ".

Таблица 12.1. *Таблица истинности умножителя знаков*

X1	X2	Y
–1	–1	1
–1	1	–1
1	–1	–1
1	1	1

Таблица 12.2. *Таблица истинности элемента "исключающее ИЛИ"*

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

Инверсия выходного сигнала в нашем случае не имеет принципиального значения. В случае необходимости мы всегда можем использовать дополнительный инвертор или изменить схему включения регулировочного элемента в генераторе управляемом напряжением. Это означает, что логический элемент "исключающее ИЛИ" вполне может быть использован в качестве фазового детектора. Схема цифрового фазового детектора приведена на рис. 12.5.

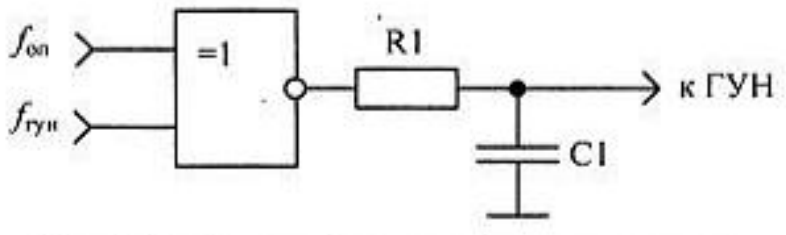


Рис. 12.5. Схема цифрового фазового детектора

Для проверки рассмотрим три варианта сигналов, поступающих на вход фазового детектора, построенного на основе схемы "исключающего ИЛИ". В первом варианте сигналы на входах фазового детектора полностью синфазны. Временные диаграммы сигналов на входе и выходе логического элемента "исключающее ИЛИ" приведены на рис. 12.6.

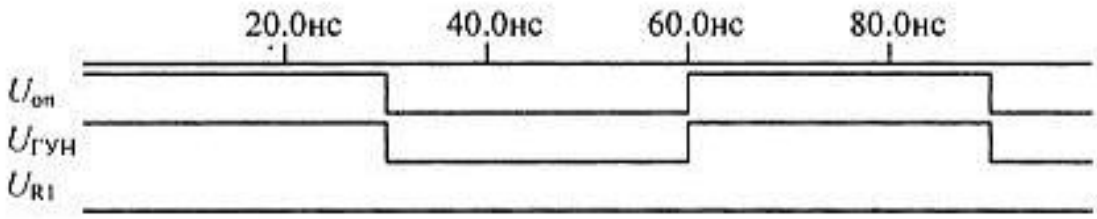


Рис. 12.6. Временные диаграммы синфазных сигналов

Анализируя эти сигналы можно сделать вывод, что при синфазных напряжениях на входах фазового детектора, построенного на логическом элементе "исключающее ИЛИ", на выходе присутствует нулевое напряжение.

Теперь подадим на входы фазового детектора сигналы, сдвинутые друг относительно друга на 15°. Временные диаграммы сигналов с таким сдвигом фазы на входе и выходе логического элемента "исключающее ИЛИ" приведены на рис. 12.7.

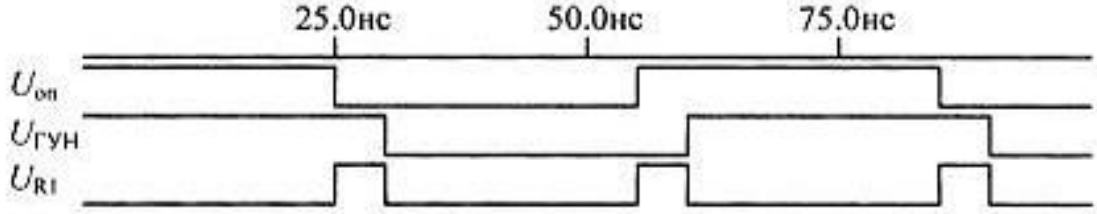


Рис. 12.7. Временные диаграммы сигналов, сдвинутых по фазе на 15°

В этом случае на выходе логического элемента "исключающее ИЛИ" появляются импульсы с частотой, равной частоте входных сигналов. Длительность формируемых импульсов пропорциональна сдвигу фаз входных сигналов. Если проинтегрировать этот сигнал, то можно получить напряжение, пропорциональное фазовому сдвигу между входными сигналами.

Подадим на входы фазового детектора сигналы, сдвинутые друг относительно друга на 165°. Временные диаграммы сигналов на входе и выходе логического элемента "исключающее ИЛИ" приведены на рис. 12.8.

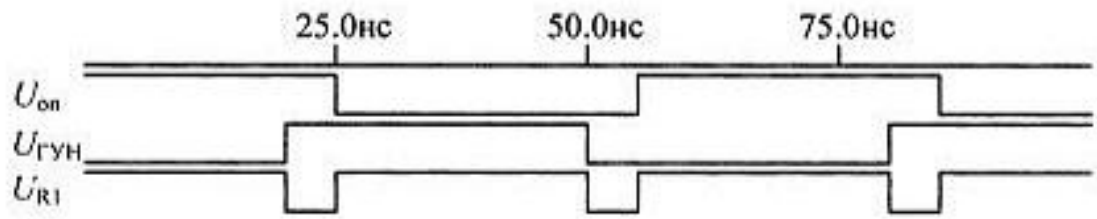


Рис. 12.8. Временные диаграммы сигналов, сдвинутых по фазе на 165°

Как и ожидалось, скважность сигнала на выходе фазового детектора изменилась. Теперь напряжение на выходе сглаживающей RC-цепочки близко к на-

пряжению питания. Можно построить зависимость напряжения на выходе схемы детектора от сдвига фаз на ее входе. Эта зависимость приведена на рис. 12.9.

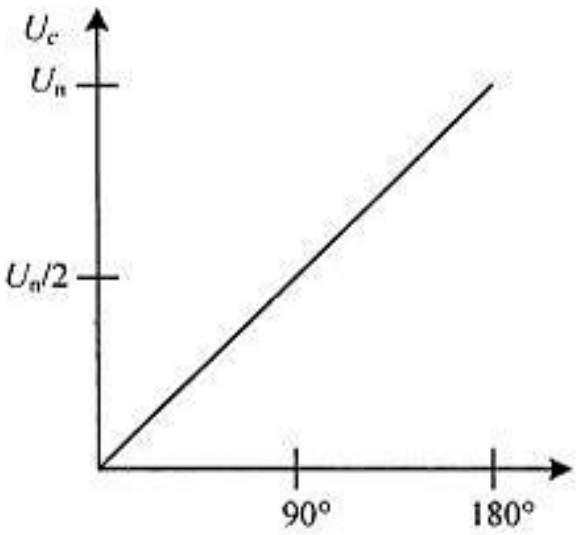


Рис. 12.9. Зависимость напряжения на выходе фазового детектора от сдвига фаз входных колебаний

Анализируя эту зависимость можно сделать вывод, что фазовый детектор, построенный на основе логического элемента "исключающее ИЛИ", обладает прекрасной линейностью преобразования "разность фаз — напряжение" и вполне может заменить аналоговый фазовый детектор.

Фазовый компаратор

При работе в цепи фазовой автоподстройки ошибка по фазе может приводить к неточной настройке синтезатора. Это связано с принципиальной работой фазового детектора — он вырабатывает напряжение, использующееся для настройки ГУН. Для удержания ГУН на заданной частоте необходим постоянный сдвиг фаз между опорным и подстраиваемым колебанием, вырабатываемым ГУН. Устранить этот недостаток можно при использовании схемы фазового компаратора.

Фазовый компаратор позволяет формировать напряжение не пропорциональное фазе, а соответствующее знаку фазы, т. е. на его выходе может быть три значения напряжения: отставание, опережение и точное совпадение по фазе входных колебаний.

Если проинтегрировать такое напряжение на конденсаторе, то при отставании фазы колебания с выхода ГУН (частота на выходе ГУН меньше требующегося значения), напряжение на конденсаторе будет уменьшаться. При опережении фазы колебания с выхода ГУН фазы опорного колебания (частота на

выходе ГУН больше требующегося значения), напряжение на конденсаторе будет увеличиваться. Один из вариантов реализации схемы фазового компаратора приведен на рис. 12.10.

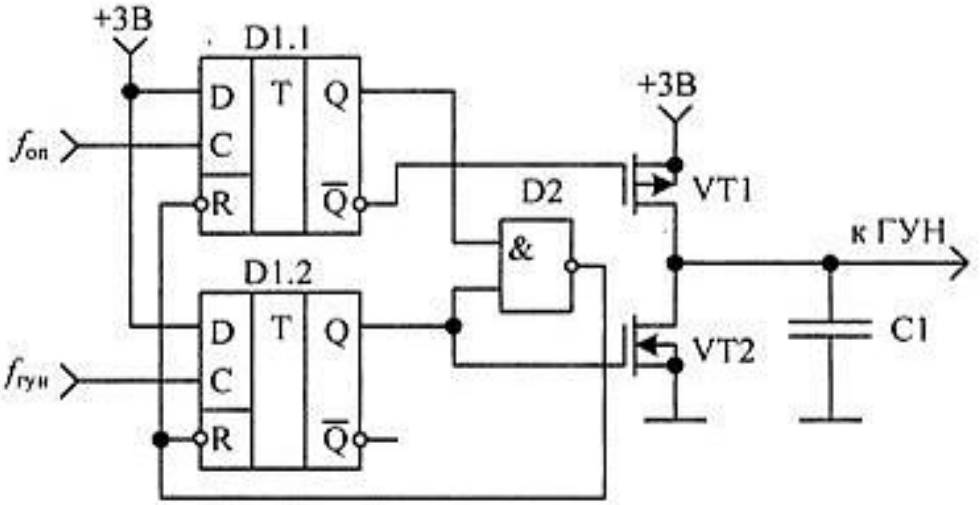


Рис. 12.10. Принципиальная схема фазового компаратора

Для построения фазового компаратора в этой схеме используются два D-триггера. На тактовые входы этих триггеров подаются опорное и подстраиваемое колебания. При поступлении на вход D-триггера нарастающего (положительного) фронта сигнала опорной частоты или частоты с выхода ГУН в триггер записывается единичное значение. При записи единичных значений в оба триггера схема обнуляются сигналом, формируемым логическим элементом D2.

В результате, в процессе работы схемы фазового компаратора при опережении фазы опорного колебания импульсы будут формироваться на выходе триггера D1.1, при опережении фазы подстраиваемого колебания импульсы появятся на выходе триггера D1.2. При точном совпадении фаз опорного и подстраиваемого колебания импульсы на выходе обоих триггеров будут настолько короткими, что ими можно пренебречь.

Импульсы с выходов триггера D1.1 подаются на ключ, заряжающий конденсатор C1, собранный на транзисторе VT1, а импульсы с выхода триггера D1.2 подаются на ключ, разряжающий конденсатор C1, собранный на транзисторе VT2. В результате, напряжение на конденсаторе C1 будет оставаться неизменным только при точном совпадении фазы опорного и подстраиваемого колебаний.

Временные диаграммы сигналов на входах и выходах триггеров, входящих в состав фазового компаратора в случае, когда опорное колебание опережает подстраиваемое по фазе, приведены на рис. 12.11.

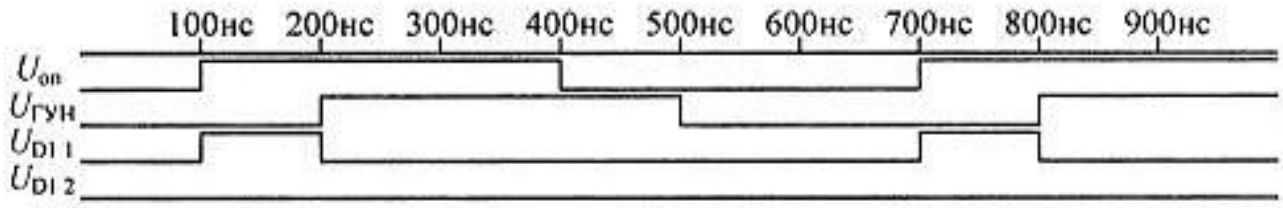


Рис. 12.11. Временные диаграммы сигналов на входах и выходах триггеров, входящих в состав фазового компаратора в случае, когда опорное колебание опережает подстраиваемое по фазе

Временные диаграммы сигналов на входах и выходах триггеров, входящих в состав фазового компаратора в случае, когда подстраиваемое колебание опережает опорное по фазе, приведены на рис. 12.12.

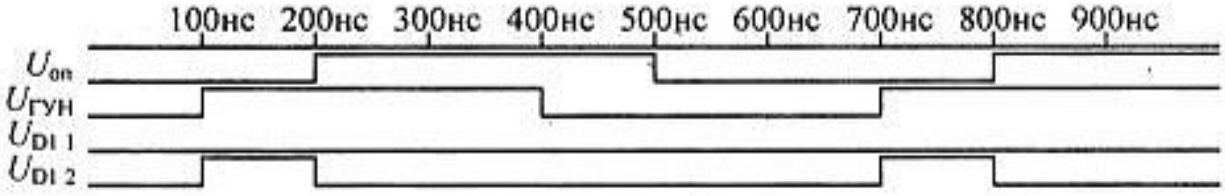


Рис. 12.12. Временные диаграммы сигналов на входах и выходах триггеров, входящих в состав фазового компаратора в случае, когда подстраиваемое колебание опережает опорное по фазе

Временные диаграммы сигналов на входах и выходах триггеров, входящих в состав фазового компаратора в случае, когда колебания совпадают по фазе, приведены на рис. 12.13.

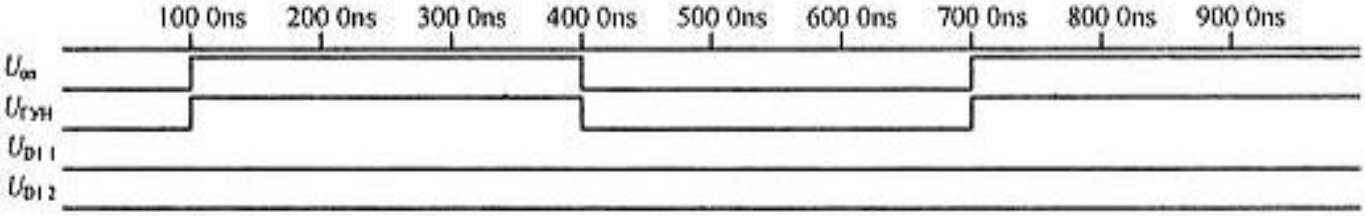


Рис. 12.13. Временные диаграммы сигналов на входах и выходах триггеров, входящих в состав фазового компаратора в случае, когда колебания совпадают по фазе

Обычно для управления варикапами, входящими в состав генератора управляемого напряжением (ГУН), требуются напряжения, большие напряжения питания цифровых микросхем (порядка 12 ... 15 В). В таких случаях на выходе фазового компаратора, схема которого приведена на рис. 12.10, требуется усилитель. В некоторых случаях можно обойтись без усилителя, если воспользоваться диодными ключами, как это показано на принципиальной схеме, приведенной на рис. 12.14.

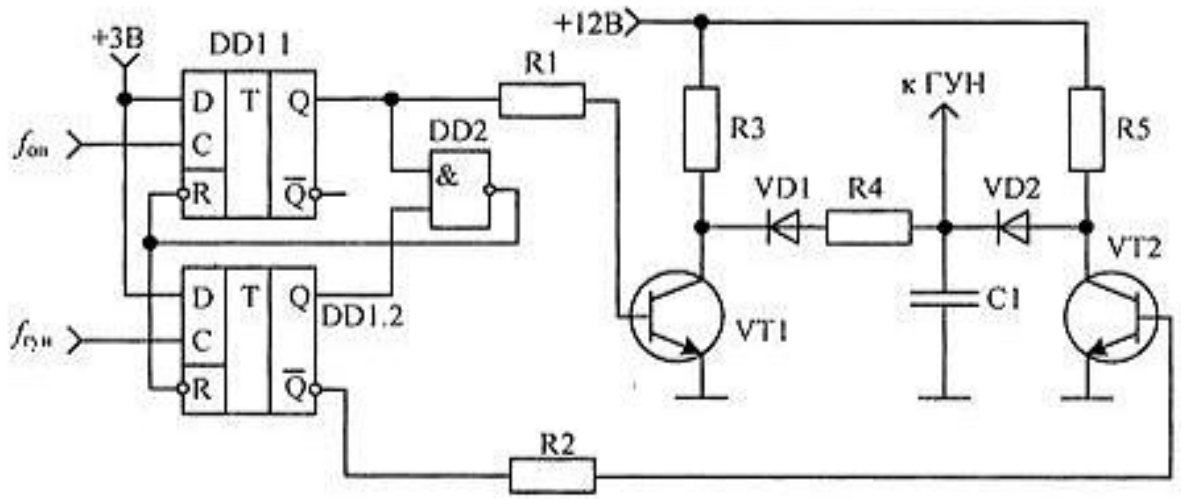


Рис. 12.14. Принципиальная схема фазового компаратора с диодными ключами

В приведенной на рис. 12.14 схеме фазового компаратора заряд емкости C1 производится через резистор R5, а разряд этой емкости — через резистор R4 и полностью открытый транзистор VT1. Использование различных резисторов в цепи заряда и разряда конденсаторов позволяет независимо регулировать время перестройки синтезатора вниз и вверх по частоте.

На этом можно завершить обзор элементов, входящих в состав схемы фазовой подстройки частоты, и перейти к рассмотрению конкретных примеров использования схем ФАПЧ.

Умножители частоты

Цепи фазовой подстройки частоты часто используются для умножения частоты. Раньше для этой цели использовались схемы генераторов гармоник с последующим выделением соответствующей гармоники узкополосным фильтром.

Намного лучше для этой цели подходит схема фазовой автоподстройки частоты. В этой схеме относительно просто можно изменять коэффициент умножения частоты изменением коэффициента деления в цепи обратной связи. Для умножения частоты внутри микросхем используется либо цифровая, либо полностью цифровая схема фазовой автоподстройки частоты.

Умножители частоты в настоящее время обычно используются для увеличения внутренней тактовой частоты больших интегральных микросхем. В этих микросхемах цифровая схема фазовой автоподстройки частоты получила название аналогового умножителя тактовой частоты, а полностью цифровая схема ФАПЧ получила название цифрового умножителя частоты.

Для увеличения тактовой частоты цифровых микросхем чаще используется полностью цифровая схема умножения частоты, а для смешанных схем или

схем, предназначенных для цифровой обработки сигналов, предпочтительнее использование аналогового умножителя частоты. Это связано со спектральной чистотой выходного сигнала. Аналоговая схема обеспечивает более стабильное колебание, но при этом медленнее выходит на рабочий режим.

Пример принципиальной схемы аналогового умножителя тактовой частоты приведен на рис. 12.15.

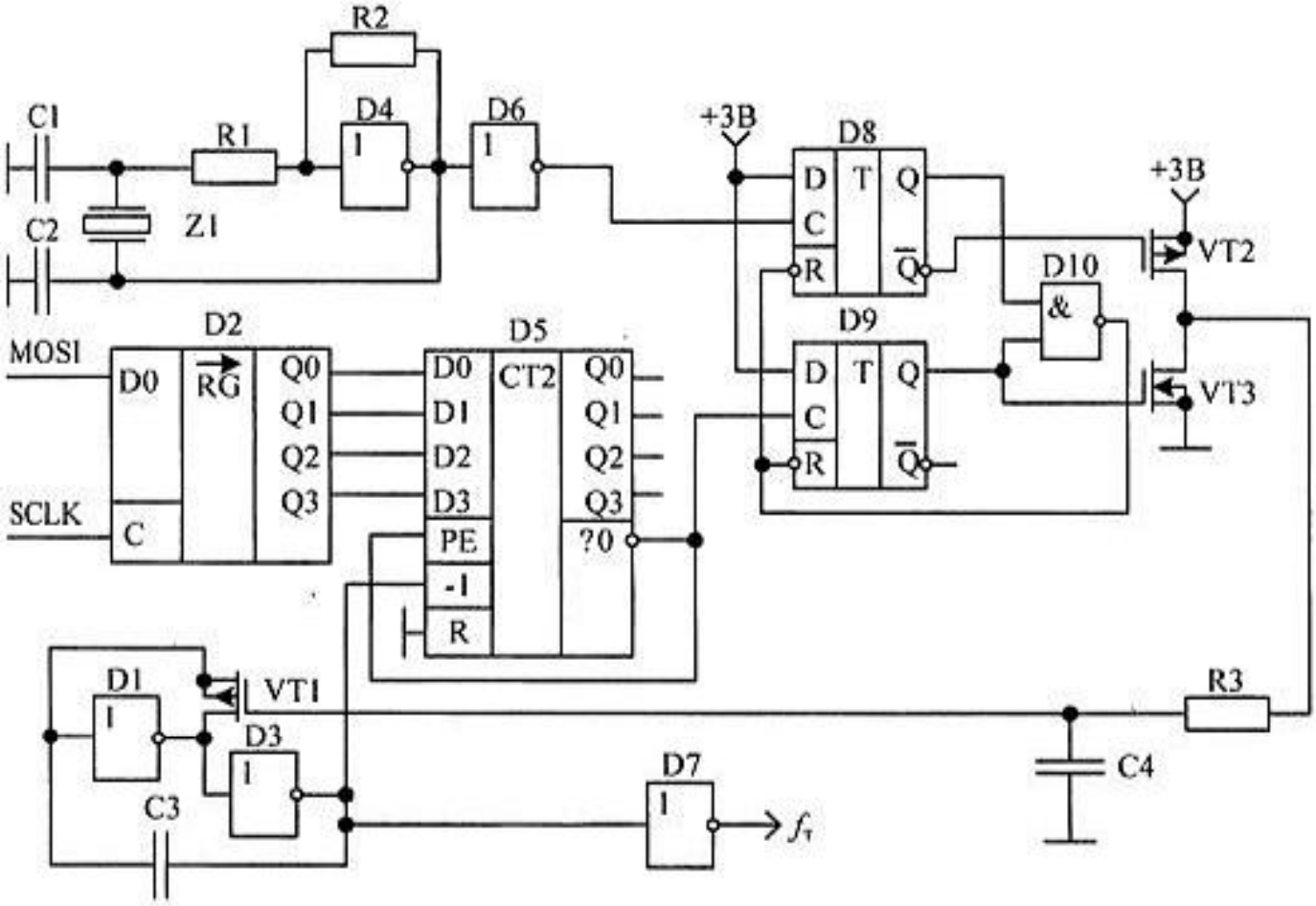


Рис. 12.15. Принципиальная схема аналогового умножителя частоты

В этой схеме опорный генератор с кварцевой стабилизацией частоты реализован на логических элементах D4 и D6. Генератор, управляемый напряжением, реализован на элементах D1 и D3. В качестве регулировочного элемента использован полевой транзистор VT1. Фазовый компаратор реализован на микросхемах D7, D8 и D10. Полосу захвата цепи фазовой автоподстройки определяет фильтр низкой частоты, реализованный на конденсаторе C4.

Данный умножитель частоты допускает только шестнадцать ступеней регулировки тактовой частоты, однако для целей задания тактовой частоты цифровой микросхемы такого набора частот вполне достаточно. Код, определяющий коэффициент умножения в схеме, изображенной на рис. 12.15, вво-

дится через SPI-порт, собранный на сдвиговом регистре D2. Особенности применения этого порта мы уже рассматривали в предыдущих главах.

В более сложных схемах умножителей частоты вводятся делители между опорным генератором и фазовым компаратором. Это позволяет реализовывать дробные коэффициенты умножения частоты.

Частотные детекторы, построенные на основе ФАПЧ

Частотные детекторы, построенные на основе элементов задержки, обычно реализуют широкую полосу детекторной характеристики. Однако используемые в реальных радиосистемах частотно-модулированные колебания обычно являются узкополосными. Поэтому для приема частотно-модулированных радиосигналов чаще используют частотные детекторы, построенные на основе схемы фазовой автоподстройки частоты.

В схеме частотного детектора не используется фазовый компаратор. Здесь лучше подходит схема фазового детектора, т. к. на ее выходе сигнал пропорционален фазе принимаемого колебания. Пример схемы частотного детектора, построенного на основе схемы фазовой автоподстройки частоты, приведен на рис. 12.16.

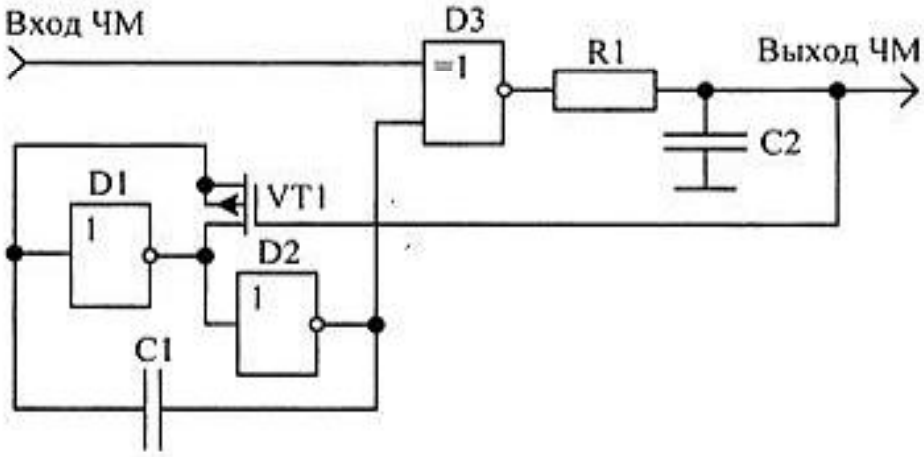


Рис. 12.16. Пример схемы частотного детектора, построенного на основе схемы фазовой автоподстройки частоты

В данной схеме частотный детектор реализован на основе фазового детектора. Как мы уже определили ранее, функции фазового детектора может выполнять логический элемент "исключающее ИЛИ". Генератор, управляемый напряжением, собран на инверторах D1 и D2, а подстройка его частоты осуществляется изменением сопротивления канала полевого транзистора VT1.

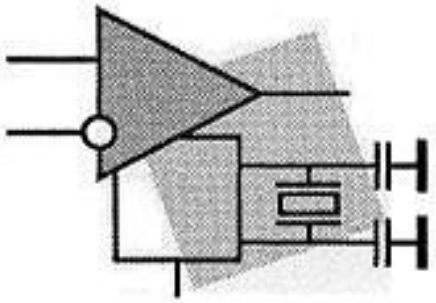
При изменении частоты входного сигнала схема фазовой автоподстройки вынуждена подстраивать ГУН на эту же частоту. При этом естественно изменяется напряжение на затворе транзистора VT1. То есть напряжение в этой точке будет соответствовать отклонению частоты входного сигнала от своего номинального значения, а значит, вся схема в целом будет осуществлять детектирование частотно-модулированного сигнала.

Полоса детектируемого сигнала в приведенной схеме будет зависеть от крутизны регулировочной характеристики генератора, управляемого напряжением и коэффициента усиления фазового детектора, выполненного на логическом элементе D3.

Итоги

В данной главе были рассмотрены схемотехнические особенности синтезаторов частоты. Эти блоки широко применяются в устройствах связи и внутри современных интегральных микросхем. Проектирование синтезаторов частоты позволяет наиболее полно изучить цифровые недвоичные счетчики. Знание внутреннего устройства синтезаторов частоты полезно не только разработчикам схем, но и прикладным программистам, которые пишут программы для готовых интегральных микросхем.

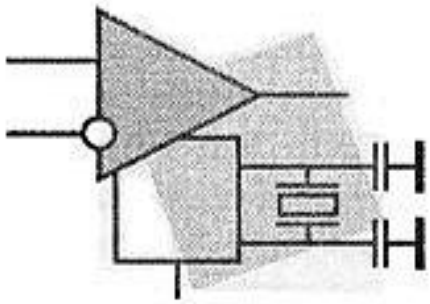
До сих пор рассматривались цифровые устройства, которые предназначены для управления устройствами связи. Не менее важны цифровые устройства, способные осуществлять обработку аналоговых сигналов непосредственно в цифровом виде. При обработке сигналов в цифровом виде появляется своя специфика и следует ее учитывать при разработке соответствующих устройств. Именно этим вопросам и будут посвящены последующие главы.



ЧАСТЬ III

Схемы цифровой обработки сигналов

ГЛАВА 13



Цифровая обработка сигналов

Известно, что интересующие потребителя сигналы, представляющие звуковую, телеметрическую или видеоинформацию не могут быть переданы по кабельному или радиоканалу, записаны на магнитный или оптический носители в непосредственном виде. Эти сигналы приходится преобразовывать к виду, наилучшим образом подходящему для передачи по одному из доступных каналов или для записи на какой-либо материальный носитель.

Задача обработки сигналов при их приеме или считывании заключается в необходимости извлечения содержащейся в них информации. Эта информация обычно заключена в амплитуде сигнала, его частоте или спектральном составе, в фазе или в относительных временных зависимостях нескольких сигналов.

При использовании цифровой обработки сигналов в связи аналоговая звуковая или видеоинформация сначала при помощи аналого-цифрового преобразователя преобразуется в цифровую форму, затем этот цифровой сигнал передается по цифровой линии связи.

Одной из задач обработки сигналов является сжатие полосы частот передаваемого сигнала без существенной потери информации. В высокоскоростных модемах и системах мобильной связи с этой целью широко используются алгоритмы устранения избыточности (алгоритмы сжатия) данных. Подобные методы используются в системах записи звука (MPEG), в телевидении высокой четкости (HDTV).

Промышленные системы управления используют информацию, полученную от датчиков системы сбора данных для выработки соответствующих сигналов, воздействующих на управляемый процесс.

В некоторых случаях в сигнале, содержащем информацию, присутствует шум, и основной целью обработки сигнала является его восстановление в

первоначальном виде. Для выполнения этой задачи используются такие методы, как фильтрация, автокорреляция, свертка и т. д. Причем эти функции могут быть реализованы как в аналоговой, так и в цифровой форме.

✧ *Обратите внимание*, что как системы обработки сигналов, так и системы управления (контроллеры) подразумевают использование аналого-цифровых преобразователей (АЦП), цифроаналоговых преобразователей (ЦАП), датчиков и устройств формирования сигнала. При этом в них в качестве обрабатывающих устройств могут применяться программируемые логические схемы, сигнальные процессоры или микроконтроллеры.

Структурная схема цифрового устройства обработки сигнала

В теореме Котельникова предполагается, что сигнал на входе аналого-цифрового преобразователя ограничен по частоте. Однако на самом деле исходные аналоговые сигналы обладают бесконечным спектром. Поэтому, прежде чем осуществить аналого-цифровое преобразование, непрерывный сигнал проходит через схему аналоговой обработки, которая выполняет такие функции, как усиление (или ослабление) сигнала, его фильтрацию, перенос спектра входного сигнала на заданную частоту.

Для подавления мешающих сигналов за пределами полосы частот полезного сигнала, а следовательно, для предотвращения наложения спектров при дискретизации исходного сигнала, на входе устройства цифровой обработки сигнала обязательно требуется поставить аналоговый фильтр низких частот или полосовой фильтр.

Типовая структурная схема устройства цифровой обработки сигналов приведена на рис. 13.1.

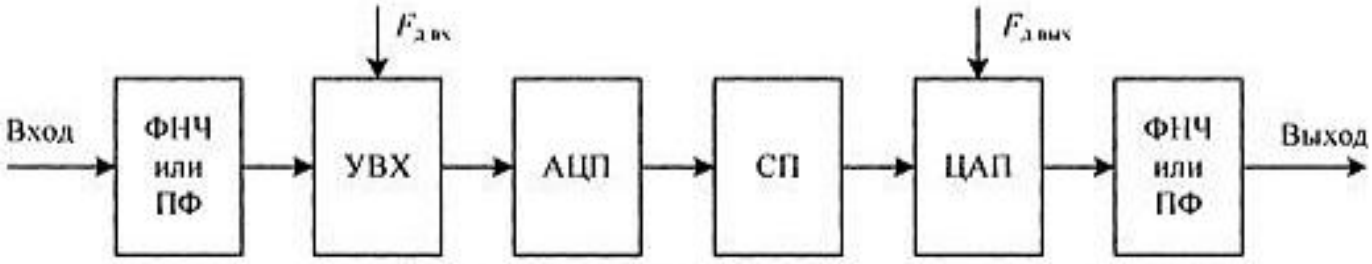


Рис. 13.1. Структурная схема цифрового устройства

Представленное на рис. 13.1 цифровое устройство должно работать в реальном масштабе времени. В составе рассматриваемого устройства устройство выборки и хранения (УВХ), которое также называют дискретизатором по времени, непрерывно стробирует аналоговый сигнал с частотой, равной $f_{д\,вх}$.

При этом аналого-цифровой преобразователь (АЦП), который также называют квантователем или дискретизатором по уровню, выдает новый цифровой отсчет сигнала процессору цифровой обработки сигнала. В качестве этого процессора может служить программируемая логическая схема, сигнальный процессор или даже микроконтроллер. Пример сигнала на выходе дискретизатора по времени приведен на рис. 13.2. На этом рисунке по оси ординат отложено значение двоичного знакового кода (в тысячах), по оси абсцисс отложен номер цифрового отсчета сигнала.

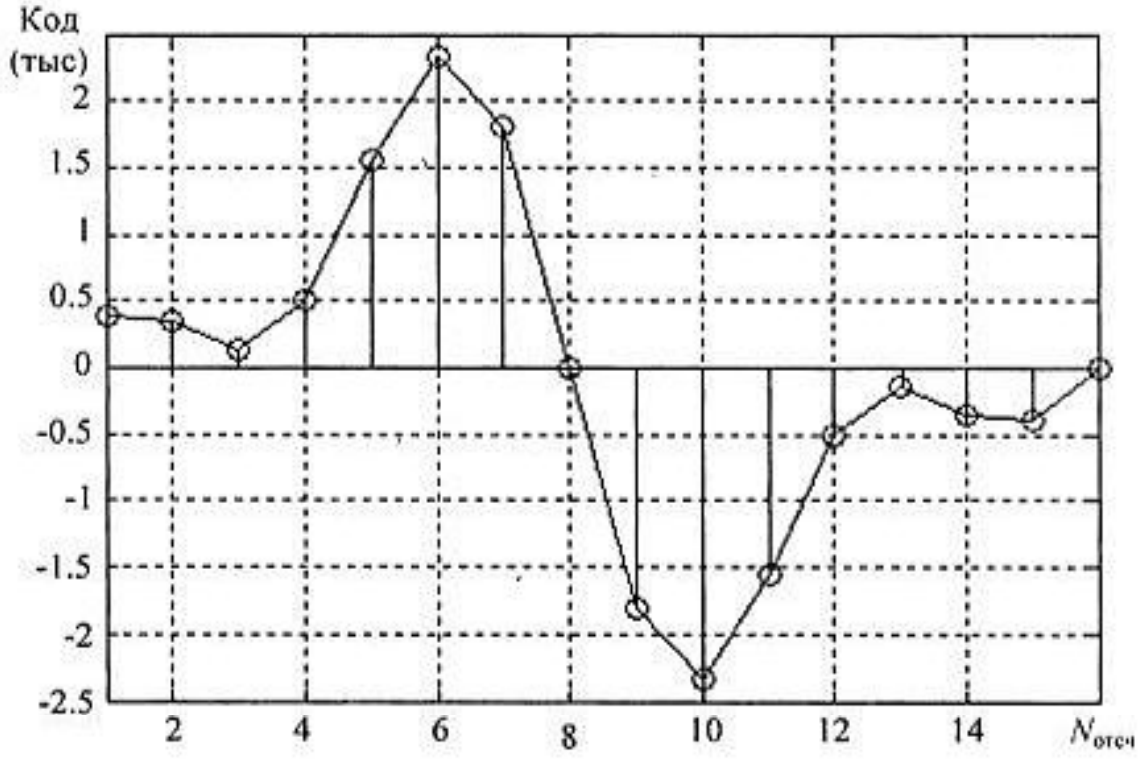


Рис. 13.2. График дискретизированного сигнала

Для обеспечения работы системы в реальном масштабе времени процессор цифровой обработки сигнала должен закончить все вычисления в пределах интервала дискретизации $1/f_{д\,вх}$ и передать сформированный выходной отсчет сигнала на цифроаналоговый преобразователь (ЦАП) до поступления следующего значения входного сигнала с выхода аналого-цифрового преобразователя. В качестве примера устройства цифровой обработки сигнала, работающего по такому принципу, может выступать цифровой фильтр.

Возможен другой принцип построения цифровых устройств реального времени. Например, при реализации на сигнальном процессоре алгоритма быстрого преобразования Фурье (БПФ), накопленный входной блок данных целиком загружается в память сигнального процессора.

В этом случае для обеспечения работы системы цифровой обработки сигнала в реальном масштабе времени необходимо, чтобы, пока сигнальный процес-

сор выполняет алгоритм БПФ над ранее полученным блоком данных, в его внутренней памяти накапливался новый блок данных. Сигнальный процессор должен успеть вычислить спектр входного сигнала в течение интервала накопления следующего блока данных, для того, чтобы быть готовым обработать его в следующий момент времени.

✧ *Обратите внимание*, что цифроаналоговый преобразователь в устройстве цифровой обработки сигнала требуется только в том случае, когда сформированные цифровые отсчеты сигнала необходимо преобразовывать в аналоговый сигнал (например, при обработке изображения, звука или при формировании радиосигнала).

Есть много устройств, где после аналого-цифрового преобразования сигнал остается в цифровом формате. Например, приемники цифровой или телеметрической информации, анализаторы спектра или осциллографы.

Точно так же существуют устройства, в которых сигнальный процессор предназначен исключительно для формирования цифрового сигнала, подаваемого на цифроаналоговый преобразователь. В качестве примера подобного устройства можно назвать передающую часть модема, возбудитель радиопередатчика или генератор сигналов различной формы.

Если в составе устройства цифровой обработки сигнала используется цифроаналоговый преобразователь, то для подавления побочных составляющих спектра полезного сигнала на его выходе необходимо применять формирующий аналоговый фильтр (anti-imaging filter).

В процессе аналого-цифрового и цифроаналогового преобразования сигнала можно выделить два основных этапа: дискретизация (квантование по времени) и квантование по амплитуде. Именно эти этапы в конечном итоге определяют разрешающую способность устройств аналого-цифрового и цифроаналогового преобразования, а значит, и точность или динамический диапазон всего устройства цифровой обработки сигналов в целом.

Особенности аналого-цифрового и цифроаналогового преобразования

Правильное понимание физических процессов, происходящих при дискретизации и квантовании аналоговых сигналов, является основополагающим моментом при разработке устройств цифровой обработки сигналов. Это понимание требуется, несмотря на то, что в большинстве современных микросхем на одном кристалле объединены не только устройства квантования и дискретизации аналогового сигнала, где в основном и проявляются эти эффекты, но и блоки цифровой обработки сигнала, такие как сигнальные процессоры и программируемые логические интегральные схемы.

Квантование аналогового сигнала по времени

Дискретизация непрерывных аналоговых данных должна осуществляться с интервалом времени $t_д = 1/f_д$. При разработке цифрового устройства этот период должен тщательно выбираться для реализации точного представления первоначального аналогового сигнала в цифровой форме. Ниже приведены основные требования к устройствам дискретизации аналогового сигнала — *критерии дискретизации по Котельникову*.

- Частота дискретизации $f_д$ сигнала с шириной полосы $f_в$ должна удовлетворять условию $f_д > 2f_в$.
- Эффект наложения спектров возникает, когда $f_д < 2f_в$.
- Эффект наложения спектров широко используется в таких задачах, как прямое преобразование ПЧ в цифровую форму.

Очевидно, что чем больше будет взято отсчетов аналогового сигнала на интервале времени (больше выбранная частота дискретизации), тем более точным будет представление этого сигнала в цифровом виде. При уменьшении количества отсчетов в единицу времени (уменьшении частоты дискретизации) можно достигнуть предела, после которого преобразованный в цифровую форму сигнал будет искажен до такой степени, что будет невозможно восстановить его в первоначальном виде.

Иными словами, в соответствии с теоремой Котельникова требуется, чтобы частота дискретизации аналогового сигнала была, по крайней мере, вдвое больше полосы полезного сигнала, иначе информация об исходном виде аналогового сигнала будет потеряна. Если выбрать частоту дискретизации меньше (а в большинстве практических устройства и равной) удвоенной полосы частот преобразуемого аналогового сигнала, то возникает эффект, известный как наложение (заворот) спектра (aliasing).

Обычно анализ аналоговых цепей производится при помощи синусоидального сигнала. На нем проще понять физический смысл явлений, возникающих в исследуемом блоке. Так как дискретизатор является аналоговым устройством, то воспользуемся этим методом и мы. Для понимания физического смысла наложения спектра рассмотрим эффекты, возникающие при дискретизации синусоидального сигнала. Эти эффекты мы проанализируем как во временном, так и в частотном представлении исследуемого сигнала.

В качестве примера, иллюстрирующего эффект наложения спектра (заворота спектра), на рис. 13.3 приведена временная диаграмма синусоидального сигнала, дискретизированного по времени идеальным дискретизатором. На этом рисунке по оси ординат отложено значение двоичного знакового кода (в тысячах), по оси абсцисс отложен номер цифрового отсчета сигнала.

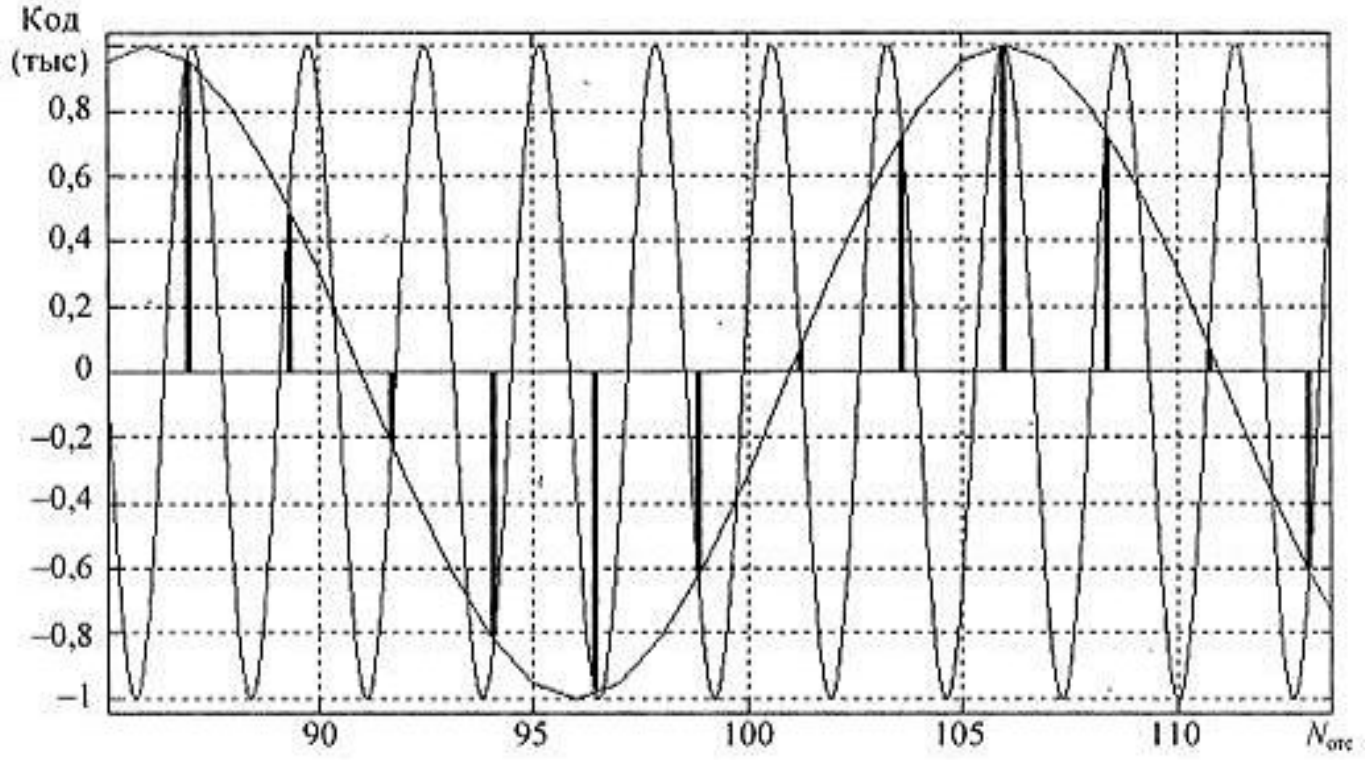


Рис. 13.3. Влияние стробоскопического эффекта во временной области, приводящее к наложению спектров входного сигнала

В приведенном на этом рисунке примере частота дискретизации $f_д$ выбрана лишь ненамного выше частоты входного аналогового сигнала $f_а$. То есть мы нарушили теорему Котельникова! ✧ *Обратите внимание*, что в результате дискретизации мы получили отсчеты сигнала, частота которого равна разности частот дискретизации и исходного сигнала $f_д - f_а$. То есть мы наблюдаем низкочастотный образ реального сигнала. Этот эффект известен в технике как стробоскопический эффект.

На рис. 13.4 приведено частотное представление той же самой ситуации. На этом рисунке четко видно, что на выходе идеального дискретизатора появляется не только низкочастотная составляющая с частотой $f_д - f_а$, но и $f_д + f_а$, $2f_д - f_а$, $2f_д + f_а$ и т. д.

Теперь рассмотрим дискретизацию одиночного синусоидального сигнала с частотой $f_а$ идеальным дискретизатором с частотой следования стробирующих импульсов $f_д$. Пусть, $f_д > 2f_а$. В частотном спектре на выходе дискретизатора появляются гармоники частоты дискретизации $f_д$, промодулированные исходным сигналом, в результате чего появляются образы входного сигнала на частотах, равных $\left|\pm Kf_д \pm f_а\right|$, где $K = 1,2,3,4,\ldots$ Эта ситуация отчетливо видна на спектре сигнала полученного с выхода идеального дис-

кретизатора, приведенном на рис. 13.4. На этом рисунке по оси ординат отложено значение плотности энергии (в децибелах), по оси абсцисс отложена частота сигнала (в килогерцах).

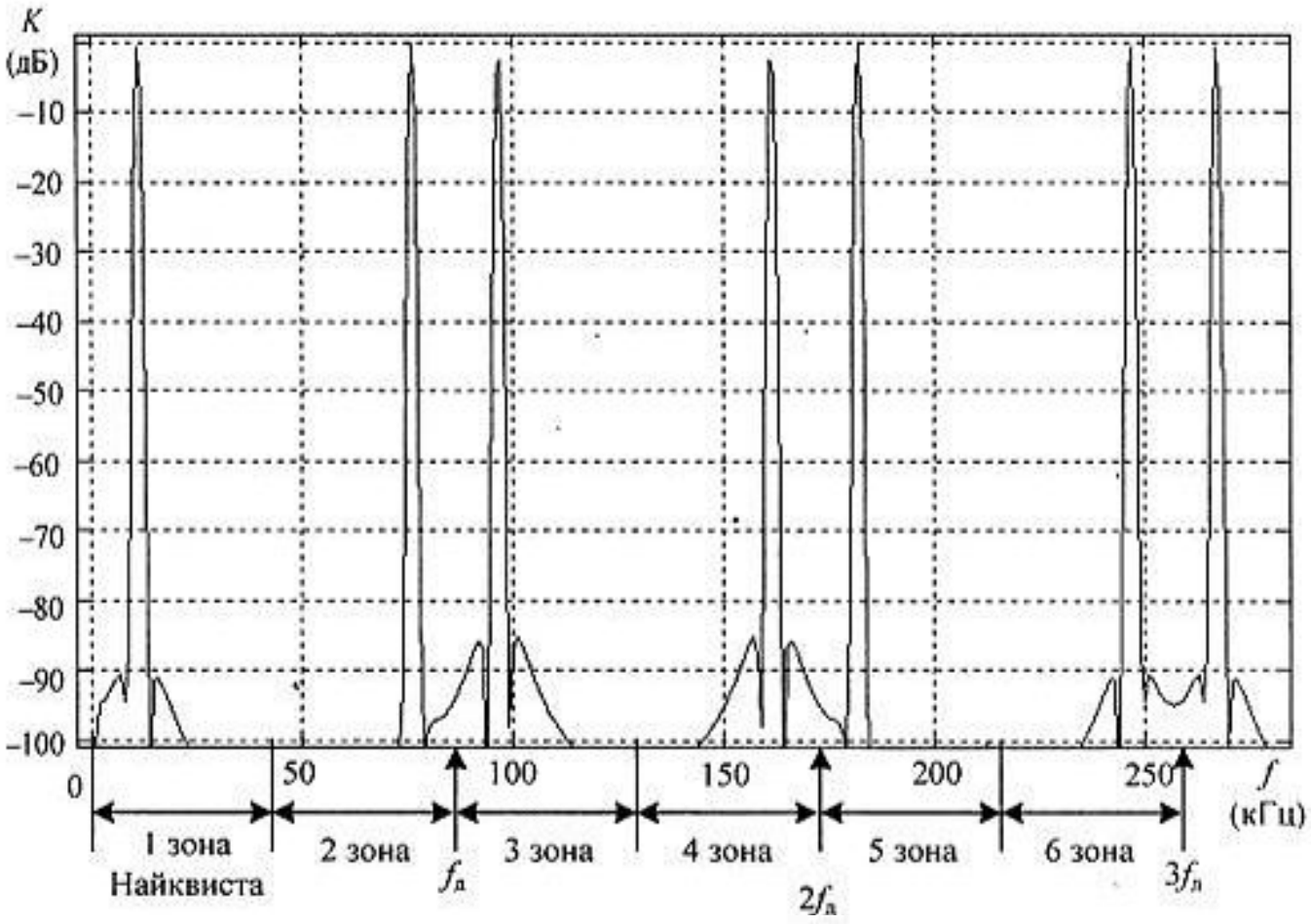


Рис. 13.4. Спектр дискретизированного аналогового сигнала

Полоса сигнала по Котельникову определяется как спектр от постоянного тока до $f_д / 2$. Частотный спектр на входе дискретизатора разделяется на бесконечное число зон. Полоса частот каждой зоны составляет $0{,}5 f_д$. На практике идеальный дискретизатор перемещает все высокочастотные образы сигнала в полосу частот от 0 до $f_д / 2$ и накладывает их на сигнал, присутствующий в первой зоне Найквиста.

Теперь рассмотрим случай, когда частота полезного сигнала выходит за пределы первой зоны Найквиста. Временная диаграмма ситуации, когда частота сигнала немного ниже частоты дискретизации, приведена на рис. 13.3. Этот случай тоже можно проиллюстрировать рис. 13.4, однако на этот раз в качестве входного сигнала следует рассматривать сигнал во второй зоне Найквиста, а компонент сигнала в первой зоне возникает после процесса дискретизации.

✧ *Обратите внимание*, что, несмотря на то, что сигнал находится вне первой зоны Найквиста, его продукт преобразования $f_д - f_в$ попадает внутрь этой зоны. Возвращаясь к рис. 13.4, становится ясно, что если мешающий сигнал появляется на любом из образов входной частоты $f_в$, то он тут же переносится на частоту $f_в$, приводя, таким образом, к появлению мешающего частотного компонента в первой зоне Найквиста.

Такой процесс подобен работе аналогового смесителя. Это означает, что перед устройством дискретизации сигнала обязательно требуется аналоговая фильтрация, подавляющая компоненты входного сигнала, частоты которых находятся вне полосы первой зоны Найквиста и после дискретизации попадают в ее пределы. Требования к амплитудно-частотной характеристике аналогового фильтра на входе дискретизатора будут зависеть от того, как близко частота внеполосного сигнала отстоит от $f_д / 2$, а также величиной требуемого подавления. Эти вопросы мы рассмотрим позднее в главе, посвященной фильтрам, предназначенным для устранения эффекта наложения спектров.

Погрешности дискретизатора

До сих пор предполагалось, что квантование по времени производится дельта-импульсами (импульсами с нулевой длительностью). Однако это математическая абстракция. Обычно сигнал на входе аналого-цифрового устройства запоминается на емкости на время, достаточное для преобразования этого сигнала в цифровое значение. Для реализации подобного устройства можно воспользоваться схемой устройства, приведенной на рис. 13.5.

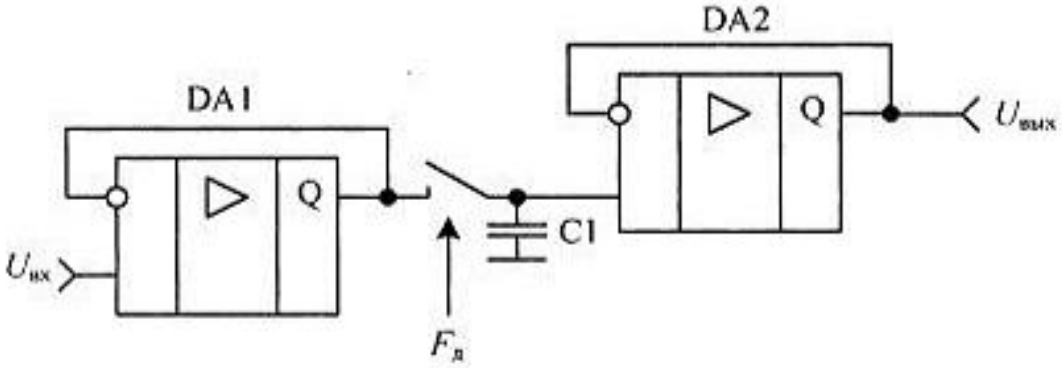


Рис. 13.5. Принципиальная схема устройства выборки и хранения

Такая схема обычно называется устройством выборки и хранения (УВХ). В приведенной на рис. 13.5 схеме малое время заряда запоминающей емкости обеспечивается низким выходным сопротивлением буферного усилителя DA1. Большое время хранения, необходимое для обеспечения заданной точности работы аналого-цифрового преобразователя, обеспечивается большим входным сопротивлением буферного усилителя DA2.

При выполнении перечисленных выше условий отношение времени хранения ко времени заряда запоминающей емкости C1 будет определяться в основном отношением сопротивлений закрытого и открытого ключа K1.

Погрешность хранения

Теперь давайте обратим внимание на то, что конденсатор, в отличие от цифровых устройств, не может хранить напряжение на своих обкладках без потерь. К моменту окончания аналого-цифрового преобразования напряжение на нем уменьшается (конденсатор разряжается). Для того чтобы преобразование аналогового сигнала в цифровую форму произошло без ошибок, необходимо, чтобы это уменьшение напряжения на обкладках конденсатора не превышало значения напряжения, соответствующего половине младшего разряда аналого-цифрового преобразователя.

Значение ошибки хранения УВХ определяется постоянной времени RC-цепочки. При этом ее сопротивление разряда R будет определяться параллельным включением входного сопротивления буферного усилителя DA2, сопротивления закрытого ключа и эквивалентного сопротивления паразитных токов утечки самого конденсатора и печатной платы.

Так как это сопротивление обычно стараются сделать как можно больше, то постоянная времени разряда конденсатора будет зависеть в основном от выбранного значения запоминающей емкости. К какому же значению постоянной времени разряда запоминающего конденсатора C1 следует стремиться?

Это зависит от характеристик аналого-цифрового преобразователя. Чем больше его разрядность, тем меньше должна быть ошибка устройства выборки и хранения. Обычно эту ошибку стараются свести к значению половины младшего разряда последующего аналого-цифрового преобразователя.

Ошибку, возникающую за счет разряда запоминающей емкости устройства выборки и хранения можно определить по следующей формуле:

$$\Delta = 1 - e^{-\frac{t}{RC}}$$

Тогда для трехразрядного АЦП ошибка не должна превышать значения $\Delta = 0,5 \times (1/2^3) = 0,0625$. Это значение может быть достигнуто при значении времени хранения $t_{хр} = (1/16) \times \tau$. То есть значение постоянной времени $\tau = RC$ должно быть в шестнадцать раз больше времени преобразования аналого-цифрового преобразователя $t_{АЦП}$!

Для восьмиразрядного АЦП требования к постоянной времени разряда запоминающей RC-цепочки еще жестче. Здесь ошибка хранения не должна пре-

вышать значения $\Delta = 0,5 \times (1/2^8) = 0,00195$. Для восьмиразрядного АЦП значение постоянной времени $\tau = R \cdot C1$ должно быть, по крайней мере, в 512 раз больше времени его преобразования $t_{АЦП}$.

Погрешность выборки

Итак, для уменьшения ошибки хранения требуется увеличивать значение емкости запоминающего конденсатора. Однако увеличение емкости конденсатора приводит к увеличению времени его заряда, а значит, к увеличению ошибки дискретизации.

Ключ устройства выборки и хранения должен открываться на время, достаточное для заряда запоминающего конденсатора входным сигналом. Этот процесс следует учитывать, даже если конденсатор на входе аналого-цифрового преобразователя — паразитный (в случае использования параллельных АЦП).

Для анализа погрешностей выборки аналогового сигнала воспользуемся принципиальной схемой устройства выборки-хранения, приведенной на рис. 13.6.

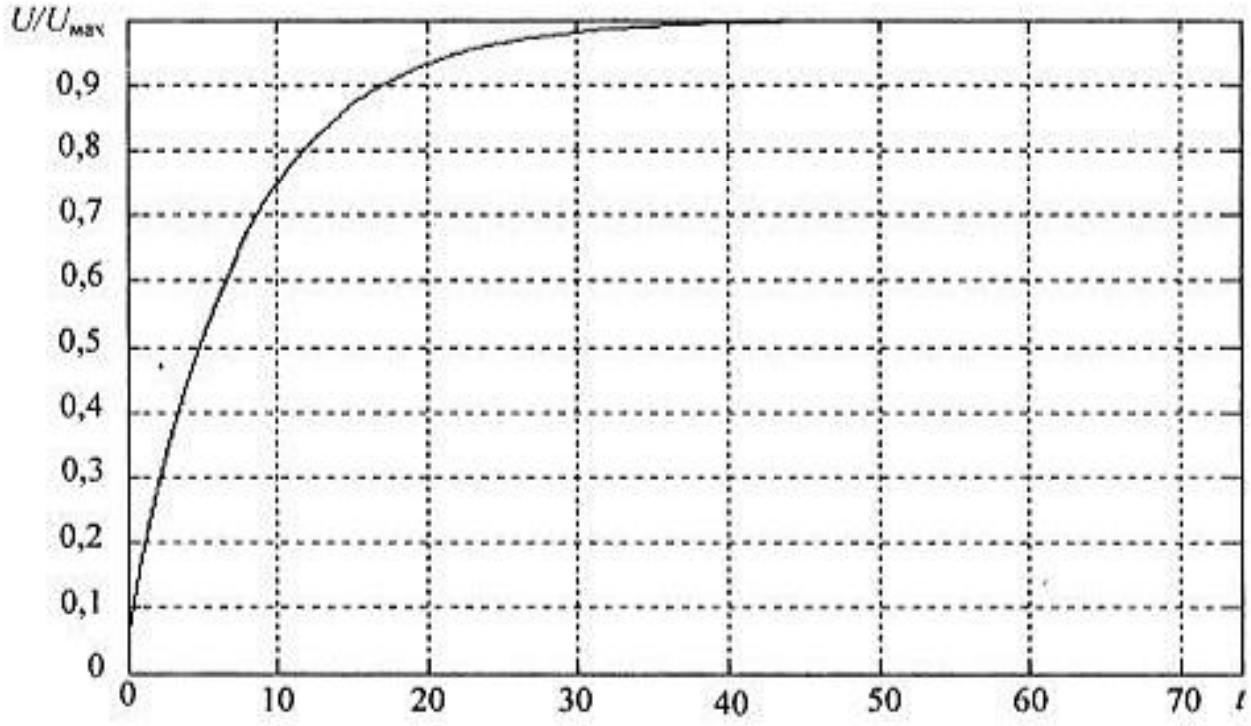


Рис. 13.6. Нарастание напряжения на выходе дискретизатора при открывании его ключа

В цепи протекания зарядного тока запоминающего конденсатора присутствуют, как минимум, два элемента — выходное сопротивление буферного усилителя плюс сопротивление открытого ключа K1 и емкость самого запо-

минающего конденсатора C1. Такая цепь обычно называется интегрирующей. Переходная характеристика подобной цепи приведена на рис. 13.6. На этом рисунке по оси ординат отложено относительное значение напряжения (в разах), по оси абсцисс отложено время.

Именно эта характеристика определяет время открывания ключа устройства выборки и хранения, достаточное для установления напряжения на входе АЦП с заданной точностью. При этом должна быть обеспечена погрешность, меньшая половины младшего разряда аналого-цифрового преобразователя. Выбранная точность установления напряжения на конденсаторе должна быть достаточна для того, чтобы амплитудные искажения дискретизатора были меньше погрешности аналого-цифрового преобразователя (квантователя по уровню). Такой режим работы устройства выборки и хранения называется режимом слежения.

Известно, что частотная характеристика линейного устройства может быть получена при помощи математической операции преобразования Фурье, выполненного над его импульсной характеристикой.

Полученная в результате амплитудно-частотная характеристика приведена на рис. 13.7. На этом рисунке по оси ординат отложен коэффициент передачи дискретизатора (в децибелах), по оси абсцисс отложена частота.

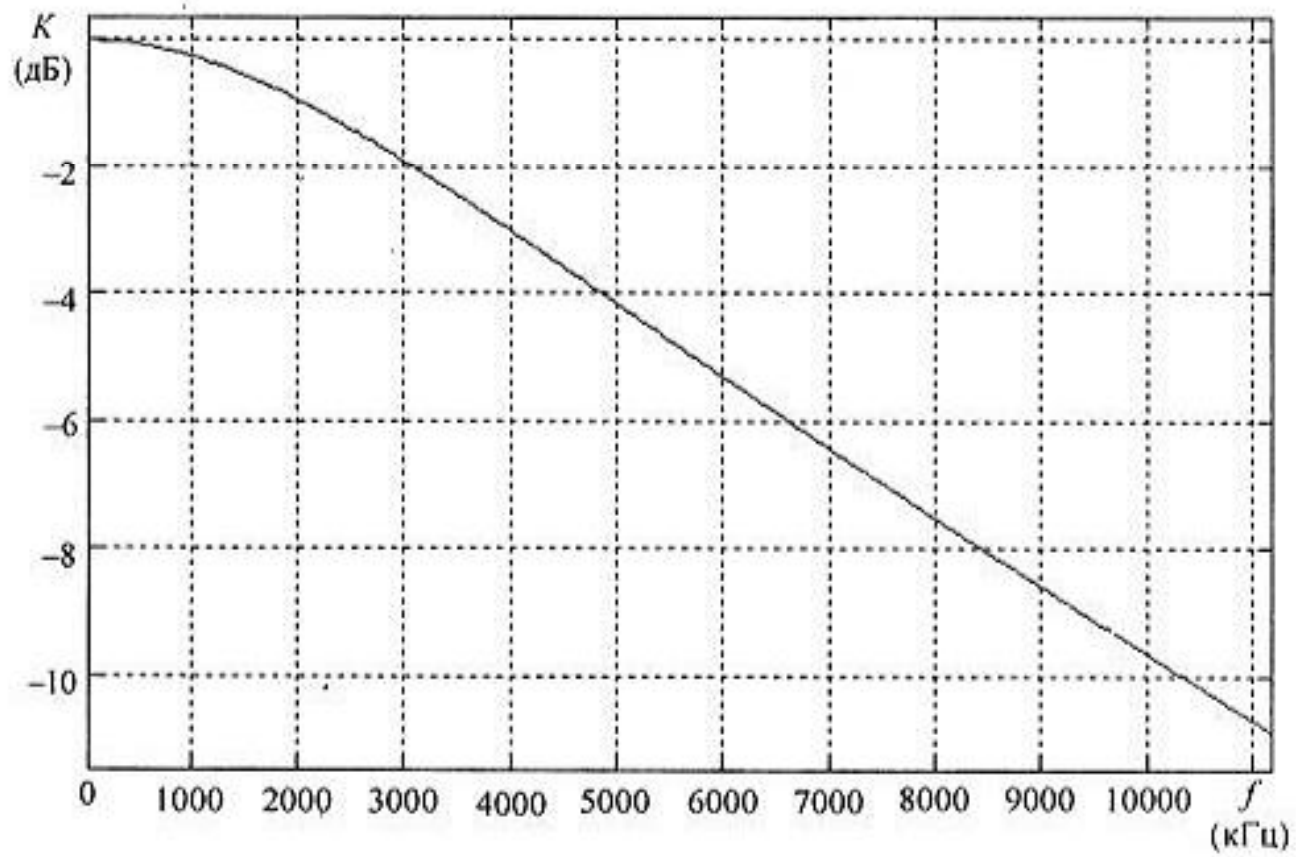


Рис. 13.7. Амплитудно-частотная характеристика дискретизатора

Получается, что на выходе рассмотренного устройства частотные образы дискретизированного сигнала уже нельзя считать распространяющимися до бесконечности. Более того! Устройство выборки и хранения начинает оказывать влияние на частотные свойства сигнала и в полосе частот первой зоны Найквиста.

Теперь, для того чтобы определить влияние высокочастотного образа сигнала, необходимо умножить его амплитуду на амплитудно-частотную характеристику устройства выборки и хранения. При неправильном выборе параметров устройства выборки-хранения это устройство может искажать сигнал и в полосе интересующих нас частот. Именно поэтому в современных микросхемах устройство выборки и хранения входит непосредственно в состав аналого-цифрового преобразователя. Характеристики аналого-цифрового преобразователя приводятся на все устройство в целом.

На высоких частотах на импульсную (а следовательно, и на амплитудно-частотную) характеристику устройства выборки и хранения начинают оказывать влияние элементы конструкции устройства цифровой обработки сигнала. В качестве примера таких элементов можно назвать индуктивность и емкость соединительных проводников, индуктивность заземляющих поверхностей печатной платы, влияние входных и выходных емкостей усилителей.

В результате влияния всех перечисленных элементов переходная характеристика устройства выборки и хранения становится более сложной по сравнению с рассмотренной ранее (рис. 13.7). В соответствии с ней изменится и частотная характеристика устройства дискретизации аналогового сигнала. Пример импульсного отклика подобной цепи на открывание ключа устройства выборки и хранения приведен на рис. 13.8. На этом рисунке по оси ординат отложено относительное значение напряжения (в разах), по оси абсцисс отложено время.

Соответствующая этому импульсному отклику амплитудно-частотная характеристика приведена на рис. 13.9. На этом рисунке по оси ординат отложен коэффициент передачи дискретизатора (в децибелах), по оси абсцисс отложена частота.

Следует заметить, что в приведенном примере паразитные элементы образовали фильтр низкой частоты, т. е. они помогают работе аналого-цифрового преобразователя.

Обычно это не так. Паразитные элементы вызывают резкие провалы амплитудно-частотной характеристики в рабочей полосе частот, могут вызвать резкое увеличение группового времени задержки на отдельных частотах рабочей полосы частот или искажение фазовых характеристик исходного сигнала.

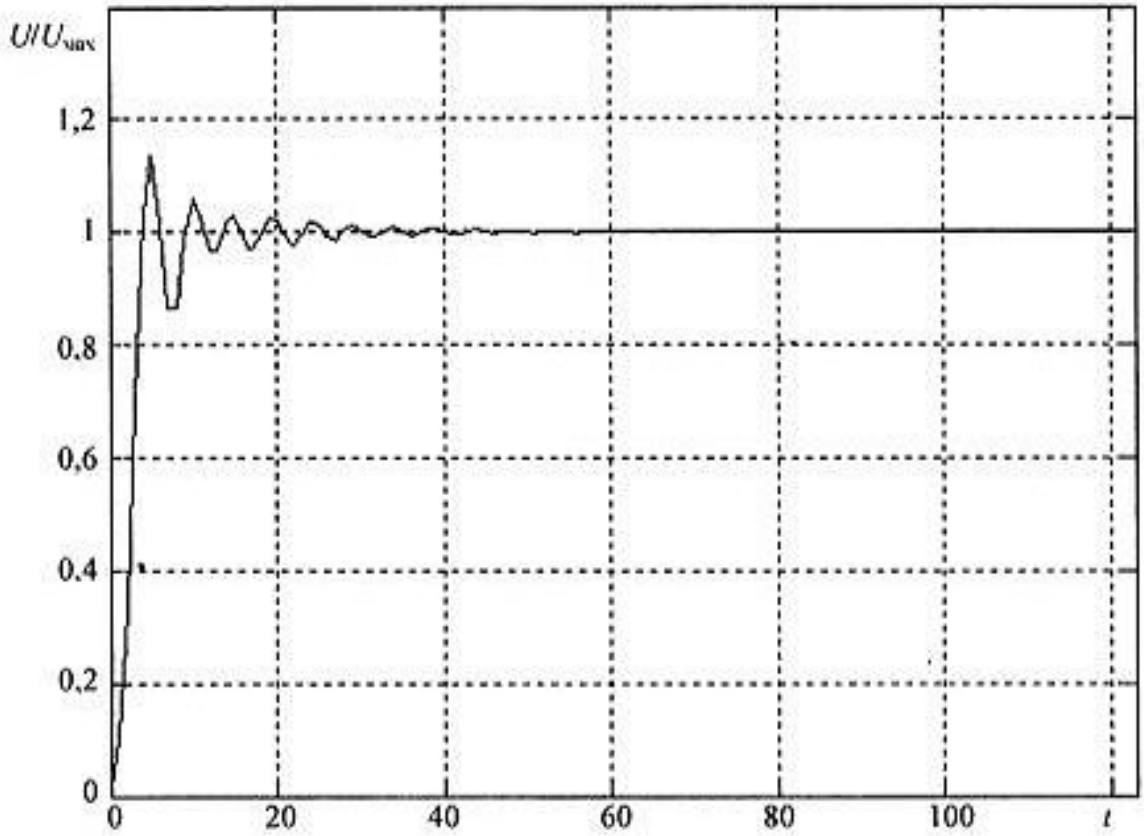


Рис. 13.8. Нарастание напряжения на выходе устройства выборки и хранения при открывании его ключа

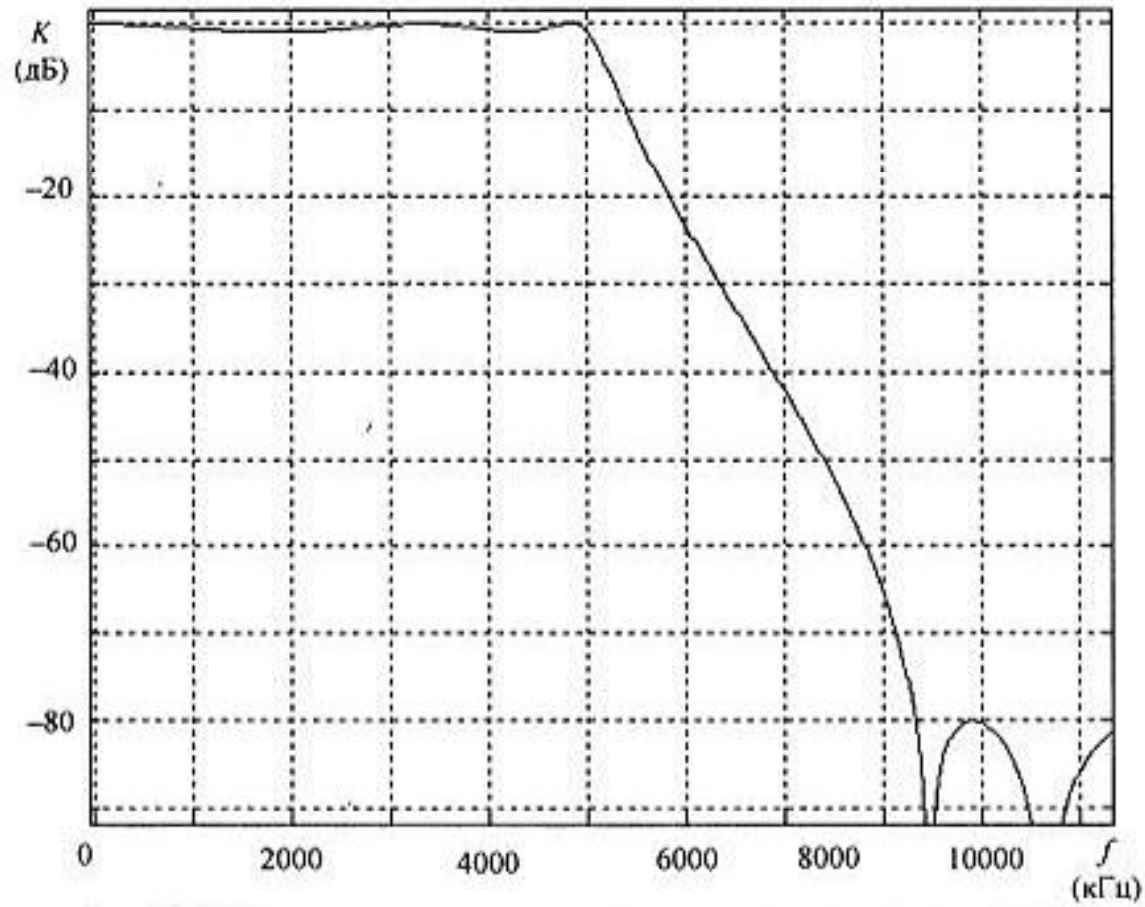


Рис. 13.9. Амплитудно-частотная характеристика дискретизатора

Для того чтобы этого не происходило, производители микросхем обычно вместе с DATASHEET на микросхему аналого-цифрового преобразователя приводят пример разводки печатной платы. При этом паразитные элементы платы часто включаются в состав аналогового фильтра.

Погрешность временного положения стробирующего импульса

Кроме частотных характеристик устройства выборки и хранения, на точность преобразования аналогового сигнала в цифровую форму существенно влияет точность временного положения импульса дискретизации.

В реальных схемах для дискретизации аналогового сигнала используются генераторы сигналов с конечной длительностью фронтов. Временное положение фронтов сигнала зависит от стабильности генераторов стробирующего сигнала и порога срабатывания логических схем. Кроме того, временное положение фронтов дискретизирующего импульса зависит от уровня помех на шинах питания цифровой схемы и шинах ее заземления.

В устройствах выборки и хранения, работающих в режиме слежения, временное положение, при котором считывается аналоговый сигнал, определяется задним фронтом стробирующего импульса. Время открывания ключа, как уже обсуждалось ранее, зависит от постоянной времени паразитных элементов схемы.

Однако мы знаем, что в зависимости от временного положения изменяется уровень сигнала на входе устройства выборки и хранения. В результате все перечисленные шумы добавляются к шумам квантования аналого-цифрового преобразователя. В ряде случаев уровень этих дополнительных шумов может значительно превосходить уровень шумов квантования.

Именно поэтому к генераторам дискретизирующего сигнала предъявляются точно такие же жесткие требования, как и к гетеродинам аналоговых приемников или возбудителям радиопередатчиков.

В качестве режима работы, альтернативного к режиму слежения, в устройствах выборки и хранения используется интегрирующий режим. В этом режиме работы используется начальный участок переходной характеристики схемы дискретизации. На этом участке, при подаче на вход постоянного напряжения, напряжение на выходе растет практически линейно, т. е. осуществляется интегрирование входного сигнала. При этом напряжение на запоминающем конденсаторе после окончания стробирующего импульса будет пропорционально энергии входного сигнала, а также длительности и форме стробирующего импульса.

Пример временной диаграммы синусоидального входного сигнала при работе устройства выборки и хранения в интегрирующем режиме приведен на рис. 13.10.

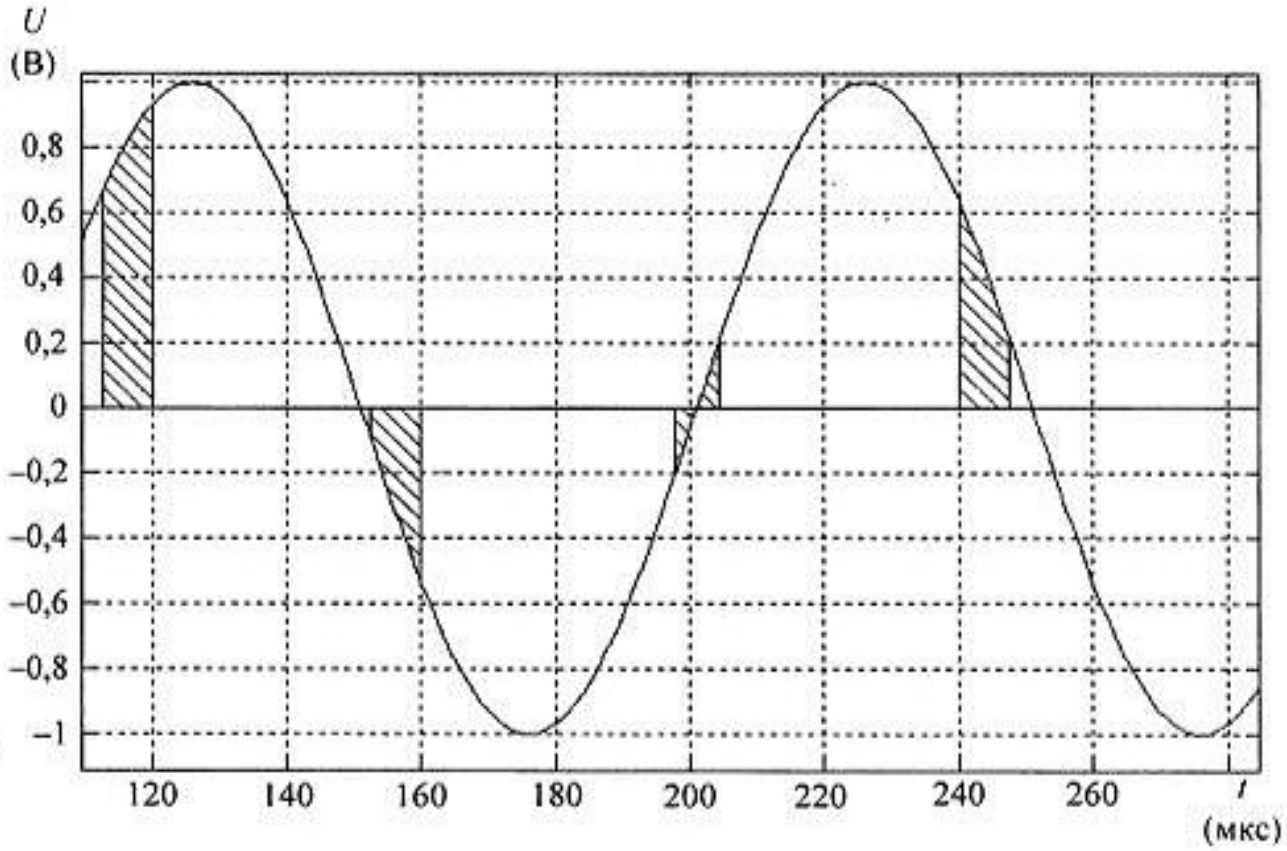


Рис. 13.10. Стробирование аналогового сигнала импульсами ненулевой длительности

На этом рисунке ширина стробирующего импульса показана заштрихованной областью. Для определения момента стробирования входного сигнала наиболее наглядным является импульс, совпадающий с 200 мкс отметкой времени. Если сравнить заштрихованные площади полезного сигнала выше и ниже нулевого уровня, то видно, что они равны. Отличаются эти площади только по знаку. В результате интегрирования заштрихованных областей анализируемого отсчета сигнала мы получим нулевое значение. Это означает, что момент стробирования входного сигнала в режиме интегрирования совпадает с серединой стробирующего импульса, т. к. именно в этот момент значение входного сигнала равно нулю.

В показанной на рис. 13.10 временной диаграмме использованы прямоугольные дискретизирующие импульсы, однако мы знаем, что в большинстве случаев получить такие импульсы на практике не представляется возможным.

Тем не менее, длительность и форма импульса стробирующего сигнала при работе УВХ в режиме интегрирования может быть учтена просто как константа. Это связано с тем, что импульсы дискретизации обладают постоянной

формой, амплитудой и длительностью, не зависящими от времени, следовательно, интеграл от данного импульса будет являться константой.

Так как в режиме интегрирования мы используем начальный участок переходной характеристики заряда запоминающего конденсатора, то напряжение на конденсаторе в конце интервала интегрирования будет меньше напряжения в режиме слежения. Уменьшение напряжения на конденсаторе может быть скомпенсировано дополнительным усилителем на входе АЦП.

Подобная схема устройства выборки и хранения приведена на рис. 13.11.

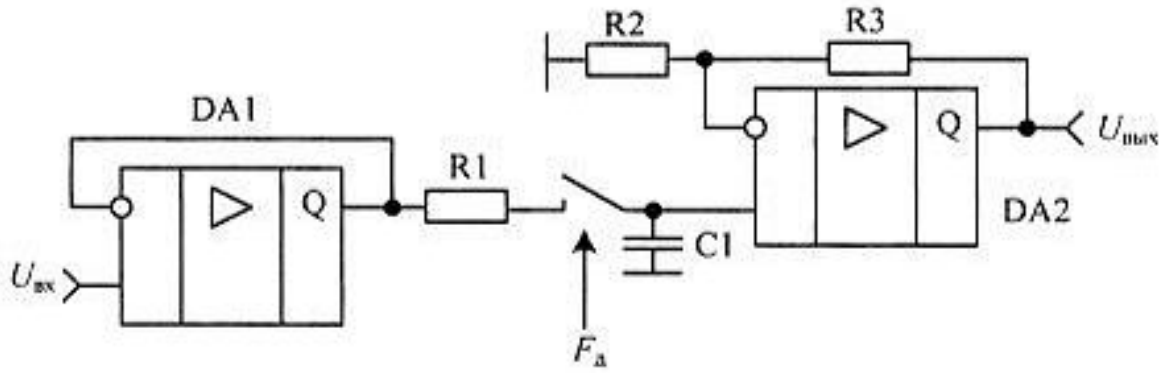


Рис. 13.11. Принципиальная схема устройства выборки и хранения, работающего в интегрирующем режиме

В данной схеме усиление буферного усилителя, компенсирующего уменьшение напряжения на запоминающей емкости, задается отношением резисторов R2 и R3. Постоянная времени цепи интегрирования определяется резистором R1. Использование этого резистора позволяет значительно уменьшить влияние параметров электронного ключа на точность дискретизации устройством выборки и хранения.

Основное преимущество интегрирующего режима работы перед следящим заключается в усреднении влияния переднего и заднего фронта стробирующего импульса, что приводит к большей точности преобразования исходного сигнала.

В качестве еще одного преимущества интегрирующего режима работы УВХ можно назвать тот факт, что в данном режиме работы при увеличении постоянной времени интегрирования уменьшается погрешность дискретизации. В результате, как для уменьшения погрешности дискретизации, так и для уменьшения погрешности хранения следует увеличивать значение емкости запоминающего конденсатора.

Максимальное значение запоминающей емкости будет ограничено только конструктивными особенностями конденсатора. Здесь имеется в виду то, что для запоминания можно применять конденсаторы только с очень маленьким

значением токов утечки, а такие конденсаторы изготавливаются с номиналом не более 10 нФ.

Теперь оценим частотные свойства устройства выборки и хранения, работающего в режиме интегрирования. Для этого, как и в предыдущем случае, воспользуемся импульсной характеристикой устройства. На этот раз импульсная характеристика устройства выборки и хранения будет определяться формой стробирующего импульса.

Если пренебречь влиянием паразитных элементов устройства (а это можно выполнить при малых значениях частоты дискретизации $f_д$), то эту форму можно считать прямоугольной. В результате преобразования Фурье мы получим амплитудно-частотную характеристику УВХ, определяемую функцией $\sin(x)/x$.

График амплитудно-частотной характеристики УВХ, работающего в режиме интегрирования, приведен на рис. 13.12.

По оси ординат характеристика, представленная на рис. 13.12, показана в логарифмическом масштабе и выражена в децибелах. Как видно из этого графика, данная амплитудно-частотная характеристика вносит частотные искажения в преобразуемый аналоговый сигнал, и для их компенсации в состав аналого-цифрового преобразователя желательно ввести цифровой фильтр с характеристикой, обратной амплитудно-частотной характеристике, приведенной на рис. 13.12.

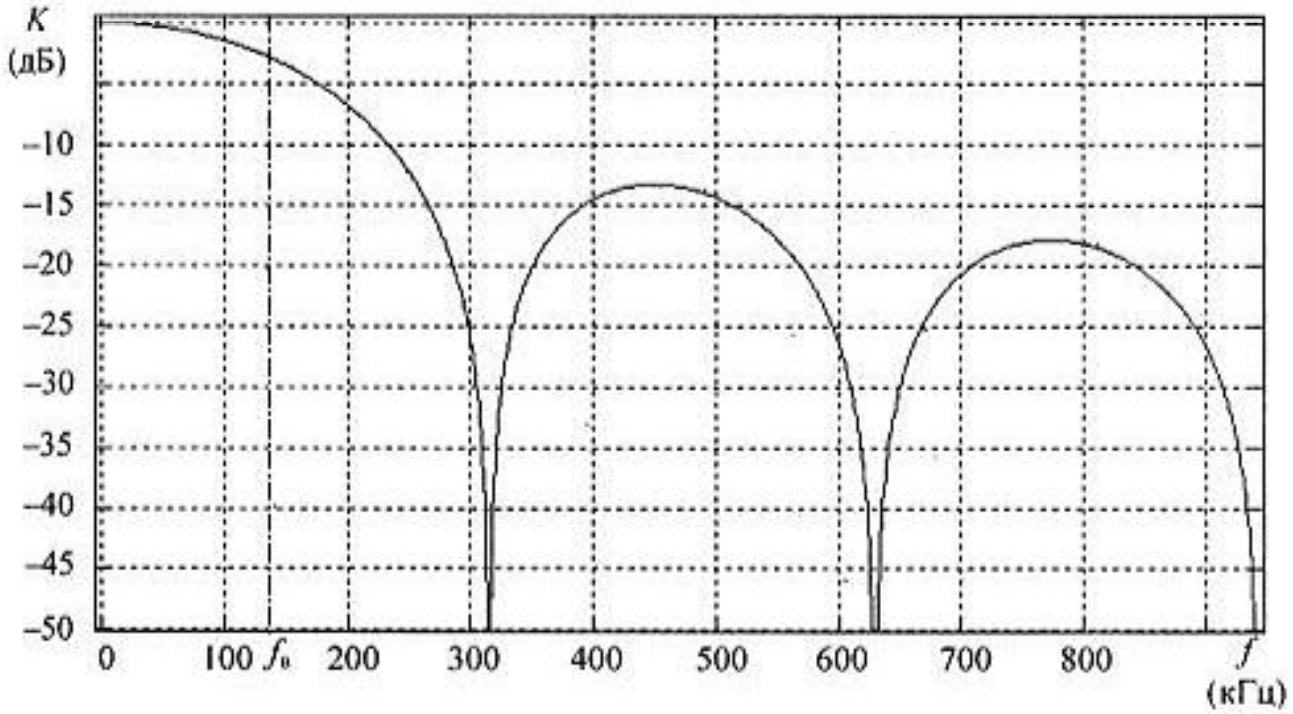


Рис. 13.12. Амплитудно-частотная характеристика УВХ, работающего в режиме интегрирования

Эта же характеристика накладывает ограничения на длительность импульса дискретизации, ведь, как известно, чем длиннее будет стробирующий импульс, тем ближе по оси частот будет находиться первый минимум приведенной на рис. 13.12 амплитудно-частотной характеристики, и, значит, тем больше будут частотные искажения в рабочей полосе входного сигнала.

Фильтры для устранения эффекта наложения спектров (Антиалиасинговые фильтры)

Говоря о дискретизации низкочастотного сигнала (например, звукового или видеосигнала) предполагается, что подлежащий дискретизации сигнал находится в первой зоне Найквиста. Важно обратить внимание на то, что без фильтрации на входе идеального дискретизатора любой частотный компонент (сигнал или шум), который находится выше верхней частоты Котельникова, будет отображаться в полосу частот полезного сигнала. Поэтому при дискретизации низкочастотного сигнала на входе аналого-цифрового преобразователя для подавления мешающих сигналов всегда используется фильтр нижних частот.

Очень важно правильно предъявить требования к характеристикам аналогового фильтра, ограничивающего спектр сигнала на входе АЦП. Для этого сначала определяются характеристики полезного сигнала, подлежащего дискретизации. Обозначим наивысшую из интересующих нас частот $f_в$. Аналоговый входной фильтр должен пропускать сигналы, лежащие в полосе частот полезного сигнала от 0 до $f_в$, и подавлять мешающие сигналы с частотой выше $f_д - f_в$.

Пусть верхняя частота полосы пропускания аналогового фильтра, ограничивающего спектр сигнала на входе АЦП, будет равна $f_в$. На рис. 13.13 показан эффект возникновения помехи, обусловленной отображением сигнала из второй зоны Найквиста в полосу частот полезного сигнала. Предположим, что уровень шумов квантования нашего устройства пренебрежимо мал, тогда уровень именно этой помехи будет определять динамический диапазон цифрового устройства. На рис. 13.14 динамический диапазон цифрового устройства обозначен как DR.

В приведенном примере составляющие спектра, которые попадают в диапазон частот от $f_в$ до $f_д / 2$, не представляют интереса, т. к. они будут в дальнейшем отфильтрованы цифровым фильтром. Поэтому они не ограничивают динамический диапазон разрабатываемой системы. Необходимо отметить,

что в ряде источников эффект отображения частот верхних зон Найквиста в первую зону называется эффектом "заворота спектра".

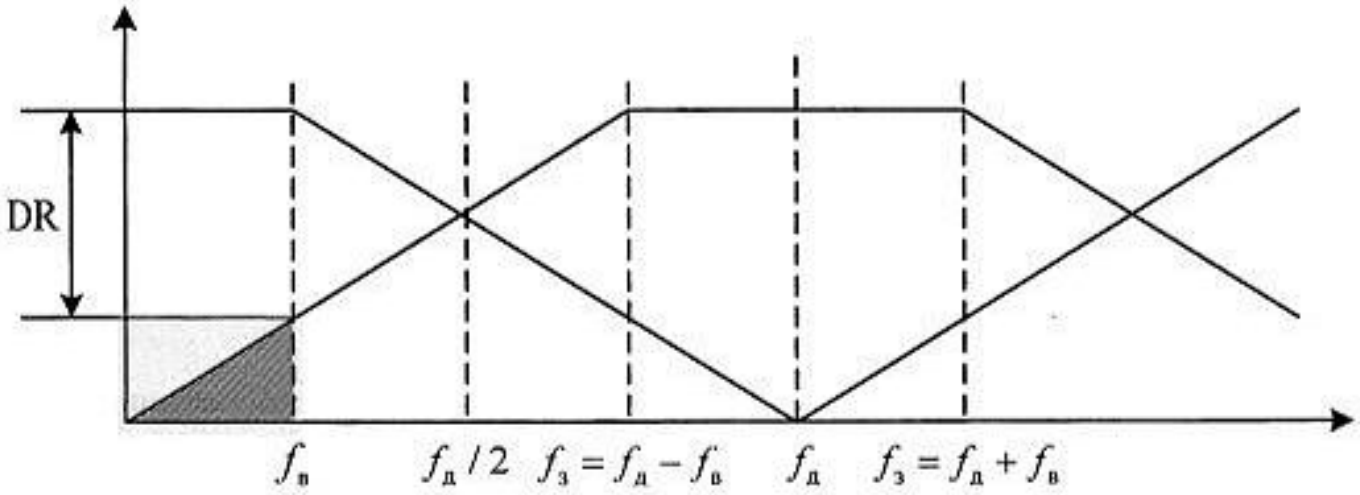


Рис. 13.13. Влияние частоты дискретизации на требования к характеристикам аналогового фильтра

Из рис. 13.13 видно, что требования к крутизне амплитудно-частотной характеристики входного аналогового фильтра определяется верхней частотой сигнала $f_в$, началом полосы задерживания $f_з = f_д - f_в$ и требуемым затуханием в полосе задерживания этого фильтра.

Как это уже было показано выше, требуемое затухание входного аналогового фильтра в полосе задерживания определяется динамическим диапазоном полезного сигнала DR. Динамический диапазон цифрового устройства выбирается исходя из заданной точности представления сигнала. При этом нижняя граница динамического диапазона DR этого устройства будет определяться уровнем всех помех, попадающих в полосу частот полезного сигнала.

При всех прочих равных условиях фильтры становятся более сложными при увеличении крутизны спада АЧХ. Известно, что фильтр Баттерворта первого порядка обладает крутизной спада АЧХ 6 дБ/октаву. При использовании фильтров более высокого порядка это значение просто умножается на порядок фильтра. Затухание фильтра Баттерворта в полосе задерживания можно определить по следующей формуле:

$$DR = N \times 6 \text{ дБ/октаву}$$

Применение фильтра Чебышева добавляет к этому значению еще 6 дБ, однако при этом резко ухудшаются фазовые характеристики фильтра в полосе пропускания. Затухание фильтра Чебышева в полосе задерживания можно определить по формуле (13.3):

$$DR = N \times 6 \text{ дБ/октаву} + 6 \text{ дБ}$$

Фильтр Золотарева-Кауэра позволяет добавить к затуханию фильтра Чебышева еще 6 дБ, как это определяется формулой:

$$DR = N \times 6 \text{ дБ/октаву} + 12 \text{ дБ}$$

Однако за это незначительное увеличение затухания придется заплатить увеличением сложности фильтра, сопоставимым с увеличением его порядка. Кроме того, в фильтре Золотарева-Кауэра затухание не увеличивается монотонно с увеличением отстройки частоты помехи от верхней частоты пропускания фильтра. В результате, при расчете динамического диапазона придется вместо оценки подавления второго образа полезного сигнала анализировать подавление сигнала на всем спектре входных частот.

Именно поэтому в качестве аналогового фильтра, ограничивающего спектр сигнала на входе аналого-цифрового преобразователя, обычно применяется фильтр Баттерворта.

Рассмотрим в качестве примера фильтр, требующийся для звуковой карты компьютера. В следующих рассуждениях предполагается, что будет использован фильтр Баттерворта нижних частот.

Зададимся верхней частотой звукового сигнала. Пусть она будет равна 20 кГц. Для обеспечения подавления мешающего сигнала на частоте 40 кГц (отстройка по частоте равна 1 октаве) на 60 дБ требуется как минимум фильтр 10-го порядка. Подобный фильтр весьма трудоемок при разработке и дорог в производстве.

Тем не менее, при таких условиях частота дискретизации $f_д$ входного звукового сигнала должна быть не менее 60 кГц, и при этом вне зависимости от вида используемого аналого-цифрового преобразователя мы сможем обеспечить точность представления сигнала в цифровом виде, эквивалентную 10-разрядной.

Кроме сложности разработки и производства подобных фильтров, фильтры высокого порядка обладают еще рядом недостатков, таких как нелинейная фазовая характеристика и связанное с ней увеличение групповой задержки полезного сигнала на краю полосы пропускания фильтра.

Увеличение групповой задержки на краю полосы пропускания фильтра может привести к тому, что даже при работе со звуковым сигналом эти искажения будут восприниматься человеком. Еще большее влияние фазовые искажения оказывают при приеме цифровых сигналов или при обработке сигналов изображения.

Все перечисленные выше факторы приводят к тому, что при преобразовании сигнала из аналоговой формы в цифровую нежелательно использовать для формирования спектра аналоговые фильтры высокого порядка, т. к. они вызывают значительные искажения формы исходного аналогового сигнала.

В качестве примера характеристик аналогового фильтра на рис. 13.14 приведена амплитудно-частотная характеристика фильтра Баттерворта 10-го порядка, а на рис. 13.15—13.17 — амплитудно-частотная, фазово-частотная характеристики этого же фильтра и зависимость группового времени запаздывания входного сигнала от частоты.

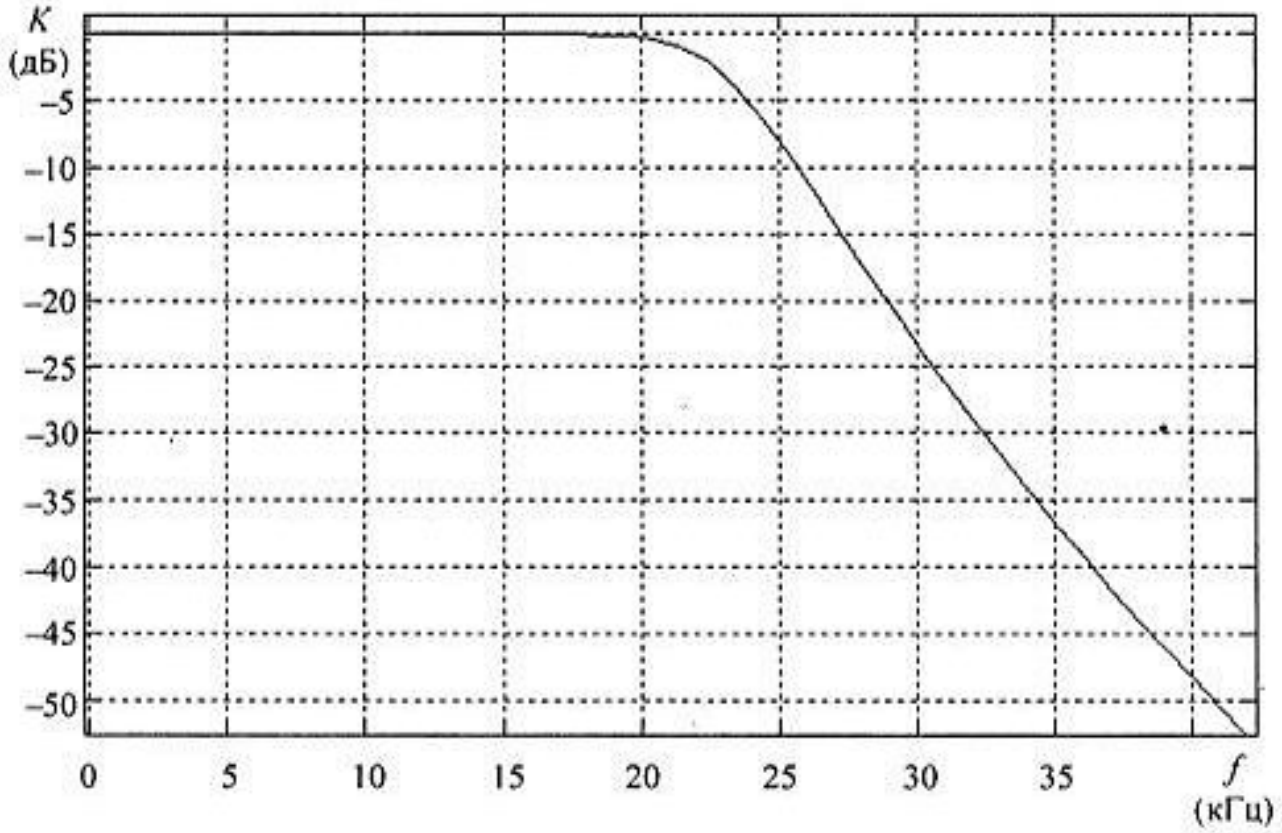


Рис. 13.14. АЧХ фильтра Баттерворта 10-го порядка

На рис. 13.16 по оси ординат отложено затухание фильтра, выраженное в децибелах, а по оси абсцисс — частота, выраженная в килогерцах. Два графика АЧХ позволяют оценить характеристики фильтра в полосе пропускания полезного сигнала и в полосе задерживания мешающих сигналов.

По характеристике фильтра в полосе пропускания видно, что на частоте 20 кГц неравномерность пропускания фильтра составляет 0,3 дБ, что соответствует 5-разрядной точности представления входного сигнала.

Из приведенного на рис. 13.16 графика видно, что фазовая характеристика обладает наибольшей крутизной на краю полосы пропускания фильтра — на частоте 23 кГц. Это обусловлено наибольшей задержкой высокочастотных составляющих входного сигнала. Задержка сигнала на частоте 23 кГц достигает значения 85 мс. Такое значение задержки высокочастотных составляющих звукового сигнала уже воспринимается человеком как искажение исходного сигнала.

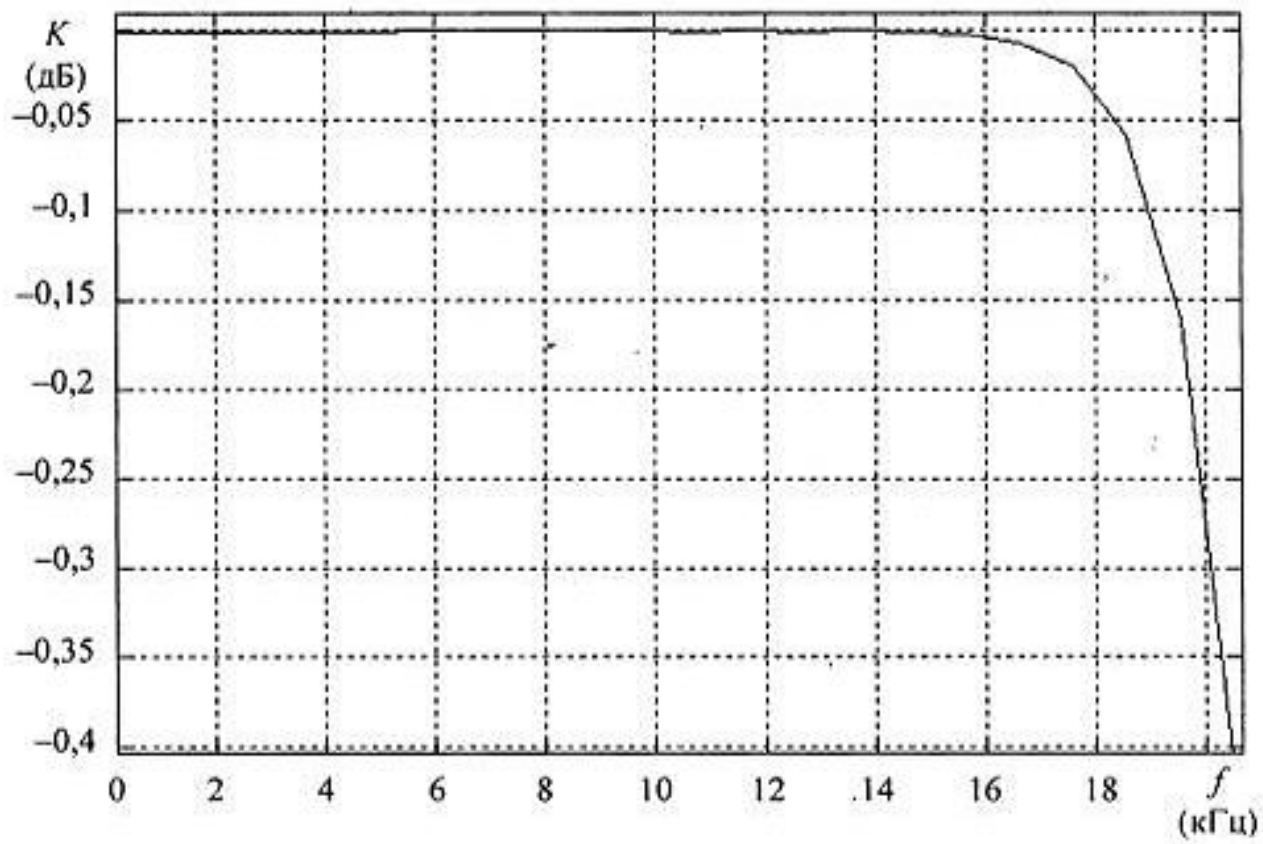


Рис. 13.15. АЧХ фильтра Баттерворта 10-го порядка в пределах полосы пропускания

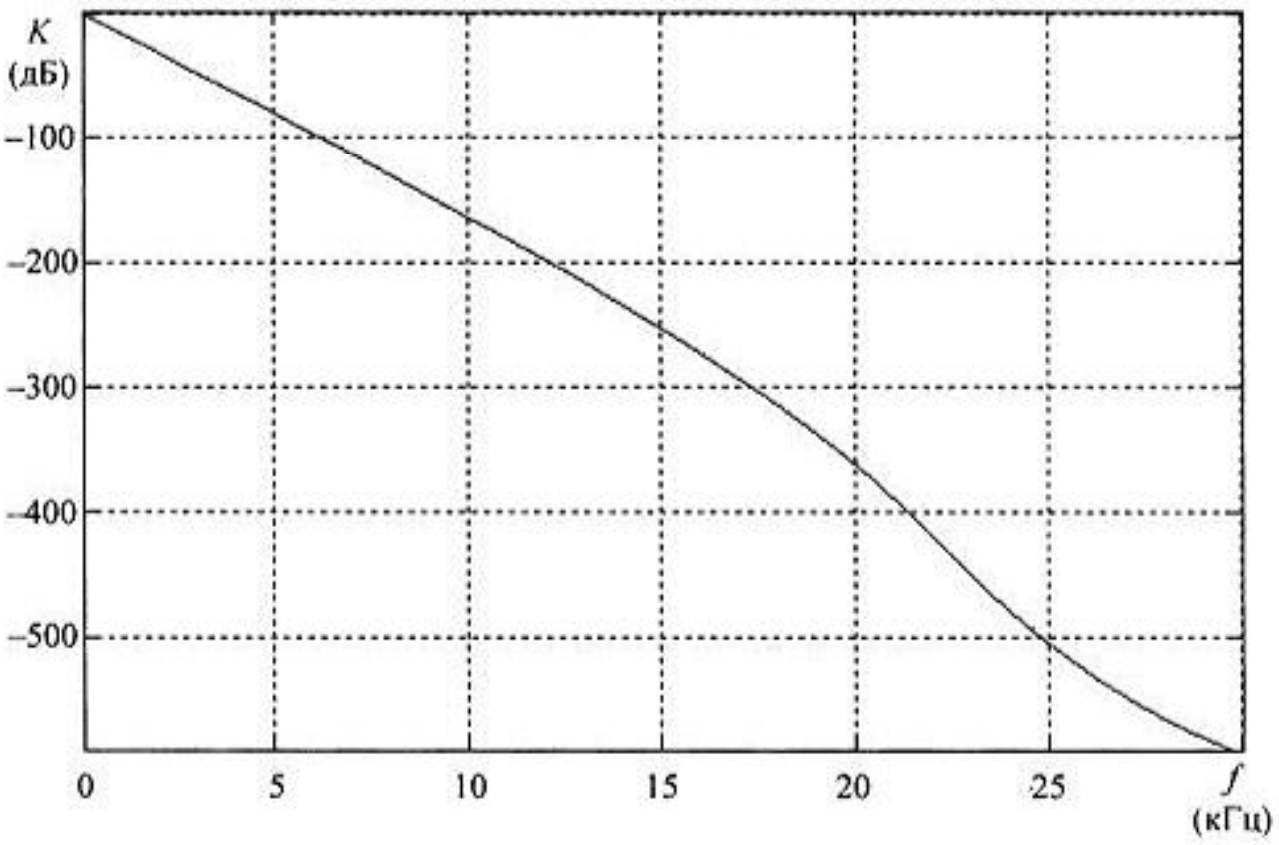


Рис. 13.16. ФЧХ фильтра Баттерворта 10-го порядка в пределах полосы пропускания

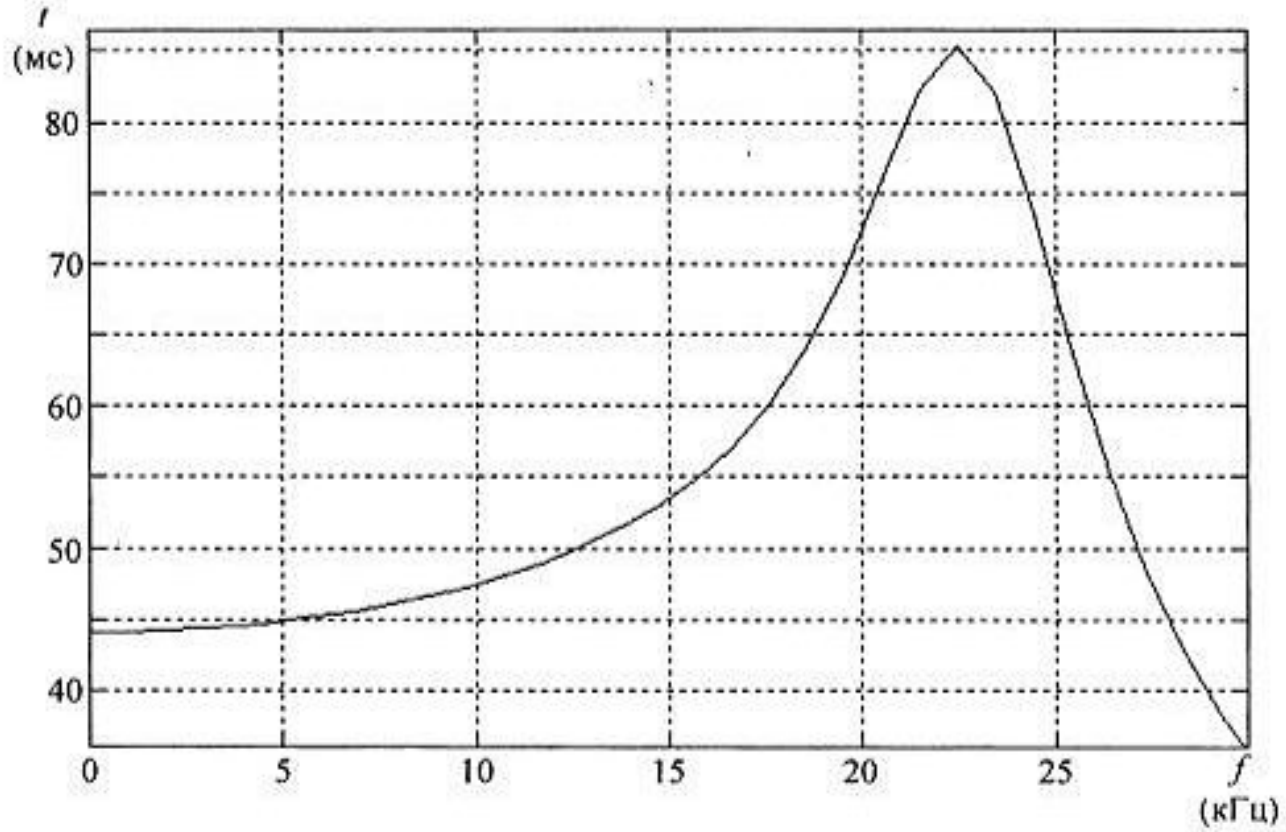


Рис. 13.17. Групповая задержка фильтра Баттерворта 10-го порядка в пределах полосы пропускания

Из приведенных выше рассуждений можно определить, что на входе аналого-цифрового преобразователя нежелательно использовать аналоговый фильтр высокого порядка. Тогда единственной возможностью увеличения динамического диапазона цифрового устройства остается увеличение разноса частот полезного и мешающего сигналов. Это может быть осуществлено только за счет увеличения частоты дискретизации входного сигнала.

Обычно частоту дискретизации увеличивают в целое число раз для того, чтобы в дальнейшем можно было ограничить полосу сигнала при помощи цифрового фильтра и затем в соответствующее число раз уменьшить частоту дискретизации сигнала на его выходе, иначе говоря, провести операцию децимации цифрового сигнала.

Подобная ситуация иллюстрируется рис. 13.18, где частота дискретизации аналогового сигнала увеличена в k раз, по сравнению со случаем, приведенным на рис. 13.13 при неизменных требованиях к частоте среза $f_в$ и к динамическому диапазону DR. Более пологий скат делает новый фильтр проще для проектирования, по сравнению со случаем, показанным на рис. 13.13.

Выбор более высокой скорости дискретизации приводит к необходимости использования более быстрого АЦП и более высокой скорости обработки данных. Тем не менее, *избыточная дискретизация уменьшает требования*

к крутизне спада амплитудно-частотной характеристики аналогового фильтра нижних частот.

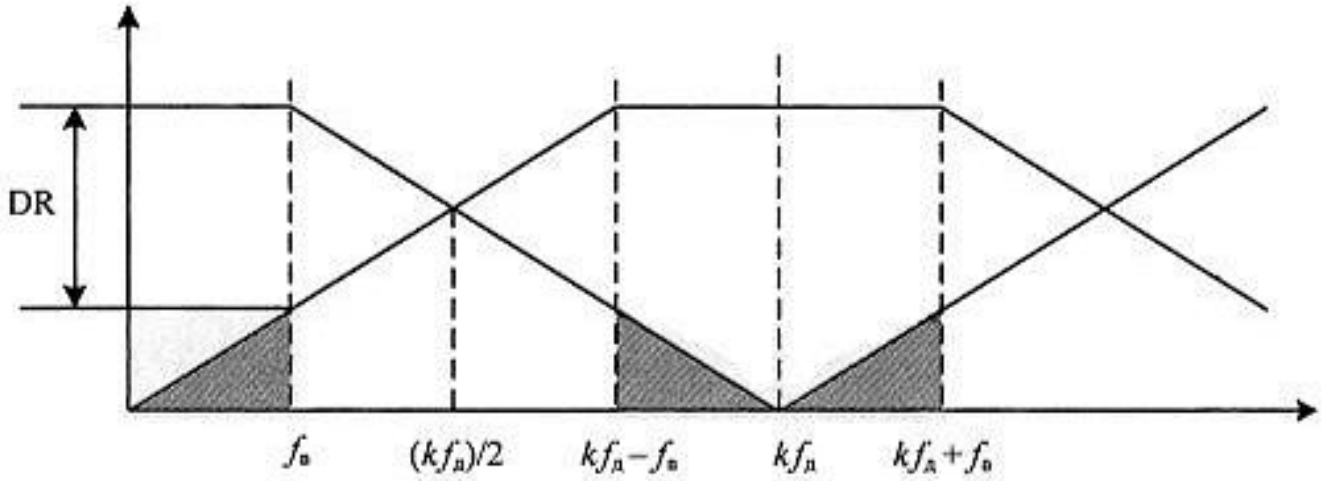


Рис. 13.18. Влияние частоты дискретизации на требования к характеристикам аналогового фильтра

Процесс проектирования аналогового фильтра, предназначенного для устранения эффекта наложения спектров, начинается с выбора начальной частоты дискретизации. Она обычно выбирается в диапазоне от $2{,}5f_в$ до $4f_в$. Затем, исходя из нужного динамического диапазона, определяются требования к амплитудно-частотной характеристике фильтра и определяется реализуемость такого фильтра с учетом ограничений по стоимости и габаритам разрабатываемой системы.

Если реализация входного аналогового фильтра окажется невозможной, то следует рассмотреть вариант с более высокой частотой дискретизации. При выборе такого варианта, возможно, потребуется более скоростной аналого-цифровой преобразователь. В ряде случаев разрядность скоростного АЦП можно взять ниже по сравнению с разрядностью низкоскоростного аналого-цифрового преобразователя, т. к. цифровые фильтры обладают свойством уменьшения шумов дискретизации.

Следует отметить, что сигма-дельта-АЦП изначально являются преобразователями с избыточной дискретизацией, и данное обстоятельство существенно ослабляет требования к аналоговому фильтру, предназначенному для устранения эффекта наложения спектров, что является дополнительным преимуществом при применении данного вида аналого-цифровых преобразователей.

Требования к аналоговому фильтру, предназначенному для устранения эффекта наложения спектров, могут быть несколько ослаблены, если вы уверены, что сигналы с частотами, лежащими в полосе задерживания $f_д - f_в$, никогда не превысят уровня полезного сигнала.

Во многих системах появление таких сигналов действительно маловероятно. Если известно, что максимальный уровень сигнала в полосе частот $f_д - f_в$ меньше амплитуды полезного сигнала на N дБ, то требования к затуханию в полосе задерживания входного фильтра могут быть уменьшены на ту же самую величину.

Новые требования к затуханию в полосе задерживания $f_д - f_в$ основаны на том факте, что в этом случае требующееся значение подавления мешающего сигнала составляет $DR - N$ дБ. В случае реализации этого варианта будьте внимательны. Убедитесь, что во входном сигнале нет никаких составляющих спектра с частотами выше частоты $f_в$ с уровнем, равным уровню полезного сигнала. Все эти составляющие спектра будут создавать низкочастотные мешающие образы в полосе частот полезного сигнала.

✧ *Обратите внимание*, что возможна обратная ситуация, когда уровень высокочастотных составляющих входного сигнала может превышать уровень полезного сигнала. В этом случае требования к входному фильтру низких частот ужесточаются на величину превышения уровня помех над уровнем полезного сигнала.

Дискретизация сигнала на промежуточной частоте (субдискретизация)

До сих пор мы рассматривали случай дискретизации низкочастотных сигналов (звуковых, видеосигналов или огибающих дискретного сигнала), когда все интересующие нас сигналы находятся в первой зоне частот Найквиста. На рис. 13.19 представлен именно этот случай, когда полоса частот полезного сигнала ограничена первой зоной Найквиста, а на выходе дискретизатора в остальных зонах Котельникова появляются образы полезного сигнала. На этом рисунке полоса частот полезного сигнала выделена черным цветом.

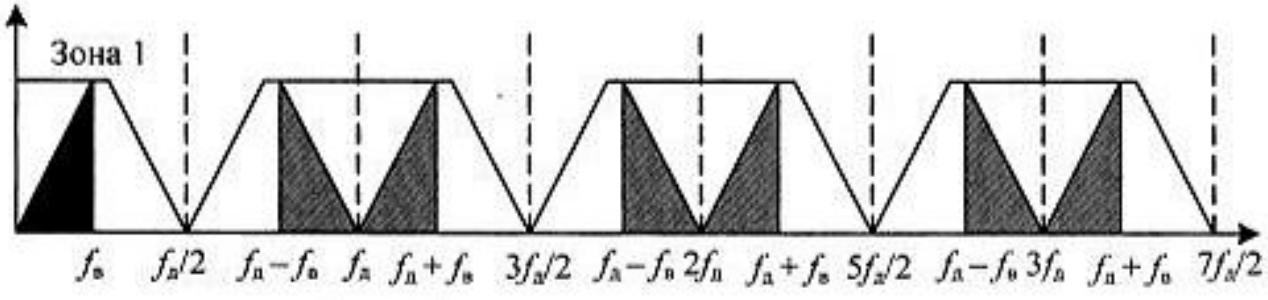


Рис. 13.19. Дискретизация низкочастотного сигнала

Теперь рассмотрим случай, показанный на рис. 13.20, где полоса полезного сигнала полностью находится во второй зоне Найквиста. Часто процесс дискретизации сигнала, находящегося вне первой зоны Найквиста, называется субдискретизацией или дискретизацией полосового сигнала.

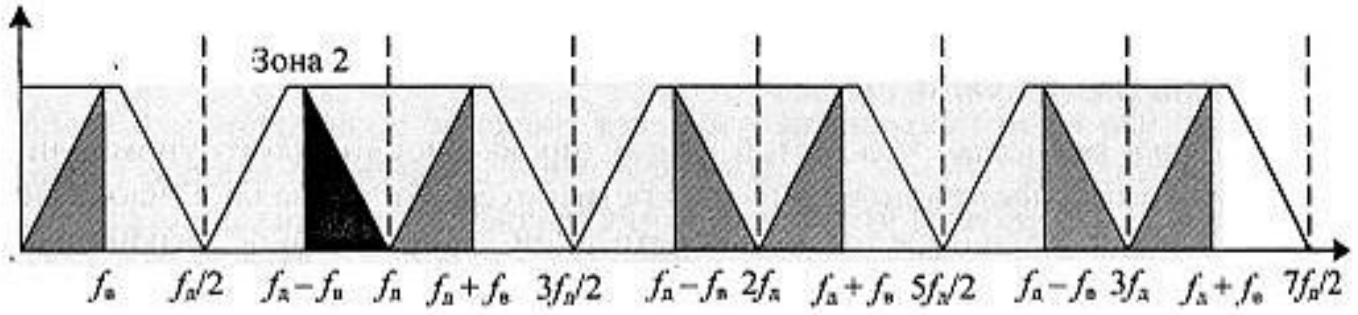


Рис. 13.20. Субдискретизация сигнала, находящегося во второй зоне Найквиста

Такая ситуация часто возникает при обработке сигнала на выходе радиоприемника. В радиоприемниках сигнал обычно переносится на промежуточную частоту. При этом гарантируется, что сигнал за пределами полосы пропускания фильтра промежуточной частоты отсутствует. Это требуется для работы аналогового приемника.

Отметим, что образ сигнала в первой зоне Найквиста, образующийся на выходе дискретизатора, содержит всю информацию об исходном сигнале, за исключением его первоначального местоположения на оси частот. Для четных зон Найквиста порядок частот в спектре образа сигнала обратный относительно первой зоны Найквиста, и это следует учитывать при дальнейшей обработке оцифрованного сигнала.

На рис. 13.21 показан вариант дискретизации сигнала, расположенного в третьей зоне Найквиста. Отметим, что в этом случае в сигнале, образующемся на выходе дискретизатора в первой зоне Найквиста, обращения частот не происходит.

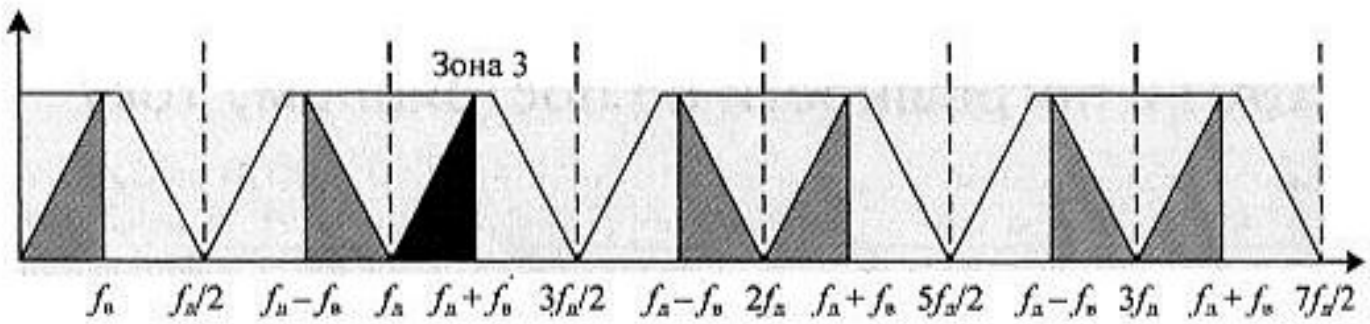


Рис. 13.21. Субдискретизация сигнала, находящегося в третьей зоне Найквиста

Итак, частоты подлежащих дискретизации сигналов могут лежать в любой зоне Найквиста, и сигнал в первой зоне является точным образом исходного

сигнала (за исключением обращения частот, которое происходит, когда сигналы расположены в четных зонах Найквиста).

Сейчас мы можем уточнить сформулированный ранее критерий преобразования сигнала в цифровую форму по Котельникову:

Сигнал должен быть дискретизирован со скоростью равной или большей удвоенной полосы частот полезного сигнала для того, чтобы сохранить всю информацию об исходном сигнале.

✧ *Обратите внимание*, что в этой формулировке нет никакого упоминания об абсолютном местоположении дискретизируемого сигнала в частотном спектре относительно частоты дискретизации. Единственное ограничение заключается в том, что *полоса подлежащих дискретизации сигналов ограничена одной зоной Найквиста*. Частотные компоненты дискретизируемых сигналов не должны пересекать частоту $f_д / 2$ с любым коэффициентом (это и является основной задачей аналогового фильтра, размещаемого на входе аналого-цифрового преобразователя).

Дискретизация сигналов, лежащих выше первой зоны Найквиста, стала популярной в аппаратуре связи, т. к. этот процесс эквивалентен аналоговой демодуляции. Обычным становится дискретизация сигналов промежуточной частоты радиоприемного устройства с последующим использованием цифровых методов для обработки сигнала. Таким способом исчезает необходимость использования демодулятора промежуточной частоты. Ясно, что с ростом промежуточной частоты растут и требования к производительности АЦП. Ширина полосы частот на входе АЦП и характеристики, связанные с допустимыми искажениями сигналов, должны быть адекватны скорее промежуточной частоте, чем основной полосе частот. Это является проблемой для большинства АЦП, предназначенных для обработки сигналов в первой зоне Найквиста, поэтому для субдискретизации требуется АЦП, который может обрабатывать сигналы в более высокочастотных зонах Котельникова.

Статическая передаточная функция АЦП и ЦАП и погрешности по постоянному току

Наиболее важным моментом, характеризующим и ЦАП, и АЦП является тот факт, что их входы или выходы являются цифровыми, а это означает, что аналоговый сигнал дискретизирован по уровню. N-разрядное слово представляется одним из 2^N возможных состояний, поэтому у N-разрядного ЦАП (с фиксированным источником опорного напряжения) может быть только 2^N значений аналогового сигнала, а АЦП может выдавать только 2^N различных значений двоичного кода. Аналоговые сигналы могут быть при этом представлены в виде напряжения или тока.

Разрешающая способность АЦП или ЦАП может быть выражена несколькими различными способами: весом младшего разряда (LSB), долей от полной шкалы размером в один миллион (ppm FS), милливольтами (мВ) и т. д. Различные устройства (даже у одного производителя микросхем) определяются по-разному, так что для правильного сравнения устройств пользователи АЦП и ЦАП должны уметь преобразовывать различные характеристики. Некоторые значения младшего значащего разряда (LSB) приведены в табл. 13.1.

Таблица 13.1. *Квантование: значение младшего значащего бита (LSB)*

Разрешающая способность N	2^N	Напряжение (10В FS)	ppm FS	% FS	dB FS
2 бит	4	2,5 В	250 000	25	-12
4 бит	16	625 мВ	62 500	6,25	-24
6 бит	64	156 мВ	15 625	1,56	-36
8 бит	256	39,1 мВ	3906	0,39	-48
10 бит	1024	9,77 мВ (10 мВ)	977	0,098	-60
12 бит	4096	2,44 мВ	244	0,024	-72
14 бит	16 384	610 мкВ	61	0,0061	-84
16 бит	65 536	153 мкВ	15	0,0015	-96
18 бит	262 144	38 мкВ	4	0,0004	-108
20 бит	1 048 576	9,54 мкВ (10 мкВ)	1	0,0001	-120
22 бит	4 194 304	2,38 мкВ	0,24	0,000024	-132
24 бит	16 777 216	596 нВ*	0,06	0,000006	-144

*600 нВ — это тепловой шум в полосе частот 10 кГц, возникающий на резисторе R=2,2 кОм при 25 °С.

Легко запомнить: 10-разрядное квантование при значении полной шкалы $FS = 10$ В соответствует $SLB = 10$ мВ, точность 1000 ppm или 0,1 %. Все остальные значения можно вычислить умножением на коэффициенты, равные степени числа 2.

Прежде чем рассматривать особенности внутреннего устройства АЦП и ЦАП, необходимо обсудить ожидаемые производительность и важнейшие параметры цифроаналоговых и аналого-цифровых преобразователей. Давайте рассмотрим определение погрешностей и технические требования, предъявляемые к аналого-цифровым и цифроаналоговым преобразователям. Это

очень важно для понимания сильных и слабых сторон АЦП и ЦАП, построенных по различным принципам.

Первые преобразователи данных были предназначены для использования в области измерений и управления, где точное задание момента преобразования входного сигнала обычно не имело значения. Скорость передачи данных в таких системах была невелика. В этих устройствах важны характеристики аналого-цифровых и цифроаналоговых преобразователей по постоянному току, а характеристики, связанные с кадровой синхронизацией, и характеристики по переменному току не имеют значения.

Сегодня многие, если не большинство АЦП и ЦАП, используются в системах дискретизации и восстановления звуковых, видео- и радиосигналов, где их характеристики по переменному току являются определяющими для работы всего устройства в целом, при этом характеристики преобразователей по постоянному току могут быть не важны.

На рис. 13.22 представлена идеальная функция передачи однополярного трехразрядного цифроаналогового преобразователя. В нем как входной, так и выходной сигналы квантованы, поэтому график передаточной функции содержит восемь отдельных точек. Независимо от способа аппроксимации этой функции, важно помнить, что реальной характеристикой передачи цифроаналогового преобразователя является не непрерывная линия, а множество дискретных точек.

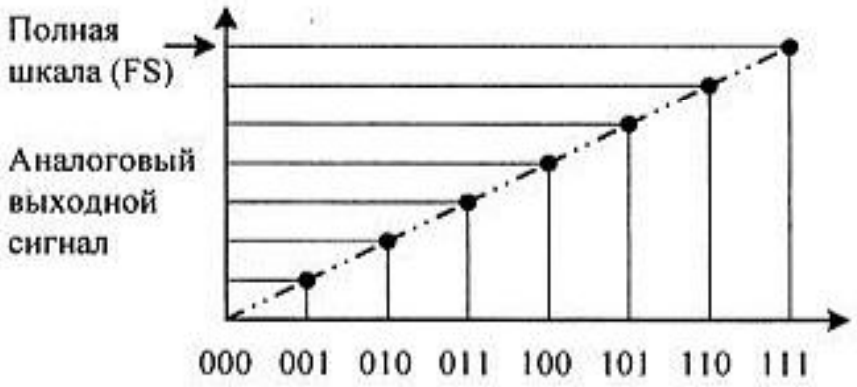


Рис. 13.22. Передаточная функция идеального трехразрядного цифроаналогового преобразователя

На рис. 13.23 приведена передаточная функция трехразрядного идеального беззнакового аналого-цифрового преобразователя. ✧ *Обратите внимание*, что аналоговый сигнал на входе АЦП не квантован, но его выходной сигнал является результатом квантования входного сигнала. Передаточная характеристика аналого-цифрового преобразователя состоит из восьми горизонтальных прямых, однако при анализе смещения, усиления и линейности аналого-цифровых преобразователей мы будем рассматривать линию, соединяющую средние точки этих отрезков.

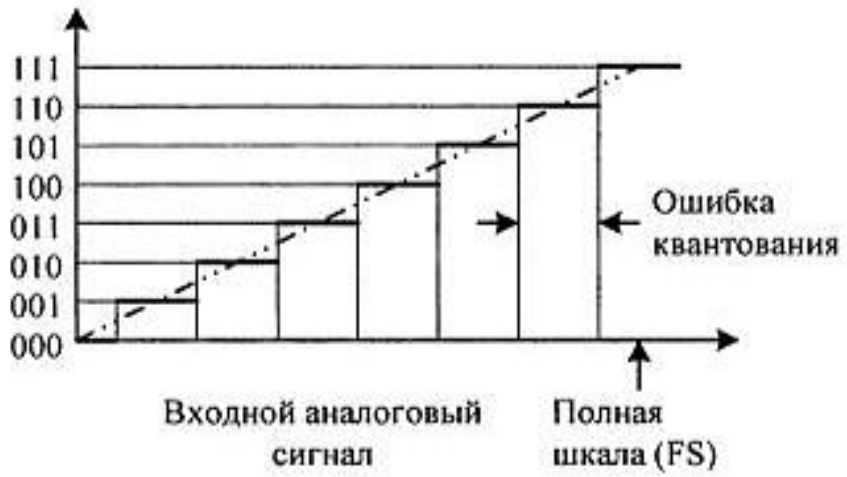


Рис. 13.23. Передаточная функция идеального 3-разрядного АЦП

В обоих рассмотренных случаях полная цифровая шкала (все "1") соответствует полной аналоговой шкале, совпадающей с опорным напряжением или напряжением, зависящим от него, поэтому цифровой код представляет собой нормированное отношение между аналоговым сигналом и опорным напряжением.

Переход идеального аналого-цифрового преобразователя к следующему цифровому коду происходит, начиная с напряжения, равного половине младшего разряда, до напряжения, меньшего напряжения полной шкалы на половину младшего разряда. Так как аналоговый сигнал на входе АЦП может принимать любое значение, а выходной цифровой сигнал является дискретным сигналом, то возникает ошибка между реальным входным аналоговым сигналом и соответствующим ему значением выходного цифрового сигнала. Эта ошибка может достигать половины младшего разряда. Этот эффект известен как ошибка квантования или неопределенность преобразования. В устройствах, использующих сигналы переменного тока, эта ошибка квантования приводит к шуму квантования.

В примерах, показанных на рис. 13.22 и 13.23, приведены переходные характеристики беззнаковых преобразователей, работающих с сигналом только одной полярности. Это самый простой тип преобразователей, но в реальных устройствах более полезны биполярные преобразователи.

В настоящее время используются два типа биполярных преобразователей. Более простой из них — это обычный униполярный преобразователь, на вход которого подается аналоговый сигнал с постоянной составляющей. Эта составляющая вводит смещение входного сигнала на величину, соответствующую единице старшего разряда (MSB). Во многих преобразователях можно переключать это напряжение или ток, для того чтобы использовать этот преобразователь как в режиме униполярного, так и в режиме биполярного преобразователя.

Другой, более сложный тип преобразователя, известен как знаковый АЦП, и в нем кроме N информационных разрядов имеется дополнительный разряд, который показывает знак аналогового сигнала. Знаковые аналого-цифровые преобразователи применяются довольно редко, в основном в составе цифровых вольтметров или термометров.

В аналого-цифровых и цифроаналоговых преобразователях различают четыре типа погрешностей по постоянному току: погрешность смещения, погрешность усиления и два типа погрешностей, связанных с линейностью.

Погрешности смещения и усиления АЦП и ЦАП аналогичны погрешностям смещения и усиления в обычных усилителях. На рис. 13.24 показано преобразование биполярных входных сигналов (погрешность смещения и погрешность нуля, идентичные в усилителях и униполярных АЦП и ЦАП, в биполярных преобразователях могут различаться, и это следует учитывать).

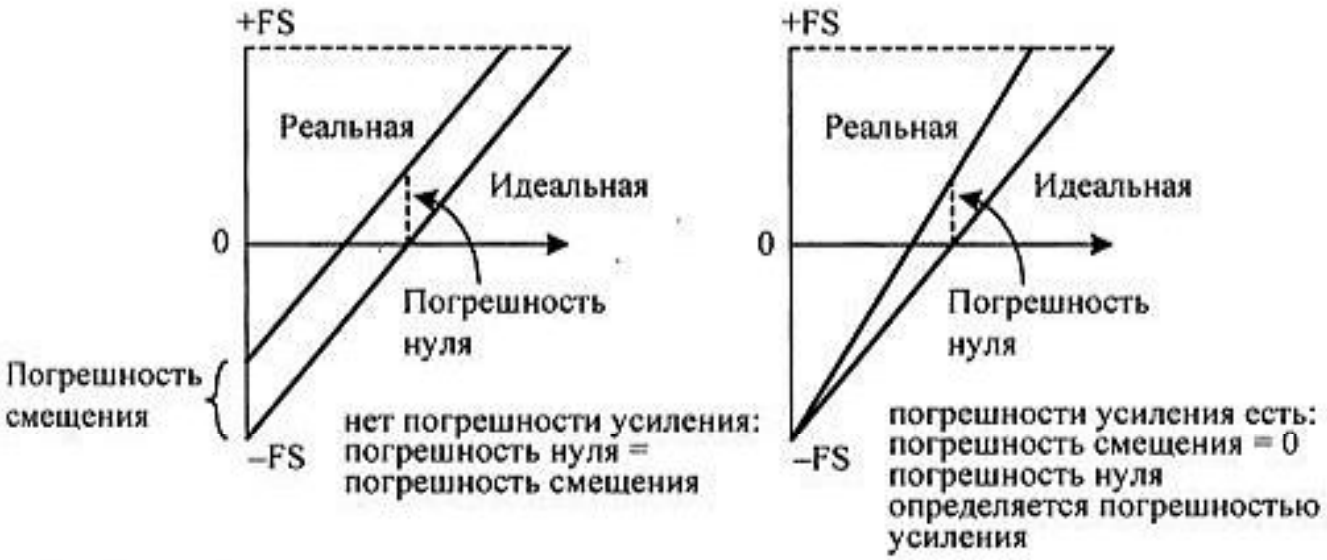


Рис. 13.24. Погрешность смещения нуля преобразователя и погрешность усиления

Передаточная характеристика и цифроаналогового, и аналого-цифрового преобразователя может быть выражена как $D = K + GA$, где D — цифровой код, A — аналоговый сигнал, K и G — константы. В униполярном преобразователе коэффициент K равен нулю, в биполярном преобразователе со смещением — равен единице старшего значащего разряда. Погрешность смещения преобразователя — это величина, на которую фактическое значение коэффициента передачи K отличается от идеального значения. Погрешность усиления преобразователя — это величина, на которую коэффициент усиления G отличается от идеального значения.

В общем случае, погрешность усиления может быть определена как разность двух коэффициентов, выраженная в процентах. Эту разность можно рассмат-

ривать как вклад погрешности усиления (в милливольтах или значениях младшего разряда LSB) в общую погрешность при максимальном значении входного сигнала. Обычно пользователю предоставляется возможность минимизации этих погрешностей. ✧ *Обратите внимание*, что в усилителе сначала регулируют смещение при нулевом входном сигнале, а затем настраивают коэффициент усиления при значении входного сигнала, близком к максимальному. Алгоритм настройки биполярных преобразователей более сложен.

Интегральная нелинейность ЦАП и АЦП аналогична нелинейности усилителя и определяется как максимальное отклонение фактической характеристики передачи преобразователя от прямой линии. В общем случае, она выражается в процентах от полной шкалы (но может представляться в значениях младших разрядов). Существует два общих метода аппроксимации характеристики передачи: метод конечных точек и метод наилучшего приближения (рис. 13.25).

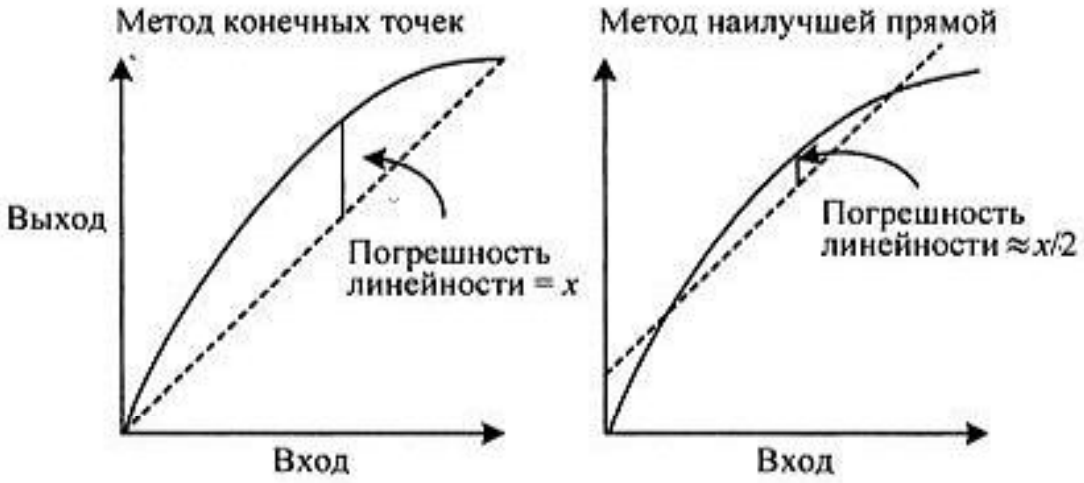


Рис. 13.25. Методы измерения суммарной погрешности линейности

При использовании метода≈ конечных точек измеряется отклонение произвольной точки характеристики (после коррекции усиления) от прямой, проведенной из начала координат. Таким образом измеряют значения интегральной нелинейности преобразователей, используемых в задачах измерения и управления (т. к. величина погрешности зависит от отклонения от идеальной характеристики, а не от произвольного "наилучшего приближения").

Метод наилучшего приближения дает более точный прогноз искажений в устройствах, работающих с сигналами переменного тока. Он обычно дает меньшее значение ошибки линейности в технических характеристиках, приводимых для микросхемы. При методе наилучшего приближения на передаточной характеристике устройства проводят прямую линию, используя стандартные методы минимизации ошибки. После этого определяется максимальное отклонение передаточной характеристики от этой линии.

Обычно полная нелинейность, измеренная таким образом, составляет только 50% от нелинейности, определенной по методу конечных точек. Это делает метод предпочтительным при указании впечатляющих технических характеристик в спецификации на микросхему, но менее полезным для анализа реальных значений погрешностей преобразователя. Для устройств, работающих с сигналами переменного тока, лучше использовать искажения, вместо нелинейности по постоянному току, так что при определении нелинейности преобразователя необходимость использования метода наилучшей прямой возникает довольно редко.

Еще один тип нелинейности аналого-цифровых и цифроаналоговых преобразователей — это дифференциальная нелинейность (DNL). Она связана с линейностью кодовых переходов преобразователя. В идеальном случае изменение цифрового кода на единицу младшего разряда точно соответствует изменению аналогового сигнала на значение единицы младшего разряда.

В аналого-цифровых преобразователях значение сигнала на аналоговом входе должно увеличиться точно на величину, соответствующую младшему разряду, для того, чтобы вызвать переход к следующему цифровому коду.

Если изменение цифрового кода на единицу младшего разряда соответствует изменению аналогового сигнала, большему или меньшему этой величины, то возникает дифференциальная ошибка DNL. DNL погрешность преобразователя обычно определяется как максимальное значение дифференциальной нелинейности из всех цифровых переходов.

В качестве примера на рис. 13.26 приведен случай, когда каждые два следующих значения напряжения на выходе цифроаналогового преобразователя смещены вверх или вниз от идеальной характеристики.

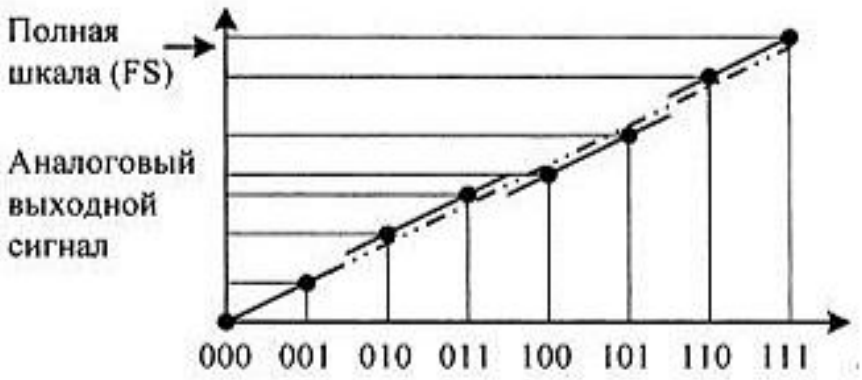


Рис. 13.26. Определение дифференциальной нелинейной погрешности

Подобная ситуация достаточно распространена и вызывается воздействием помехи от одного из двоичных разрядов на источник опорного напряжения.

Если дифференциальная нелинейность цифроаналогового преобразователя превышает значение –1 младшего разряда (LSB) на любом из цифровых пе-

реходов (рис. 13.27), то такой цифроаналоговый преобразователь называют немонотонным, и его характеристика передачи содержит один или несколько локальных максимумов или минимумов. Дифференциальная нелинейность, большая чем +1 младшего разряда (LSB), не вызывает нарушения монотонности, но также нежелательна. Во многих устройствах с применением цифроаналоговых преобразователей монотонность ЦАП очень важна. Особенно это важно в системах с обратной связью, где немонотонность цифроаналогового преобразователя может изменить отрицательную обратную связь на положительную.

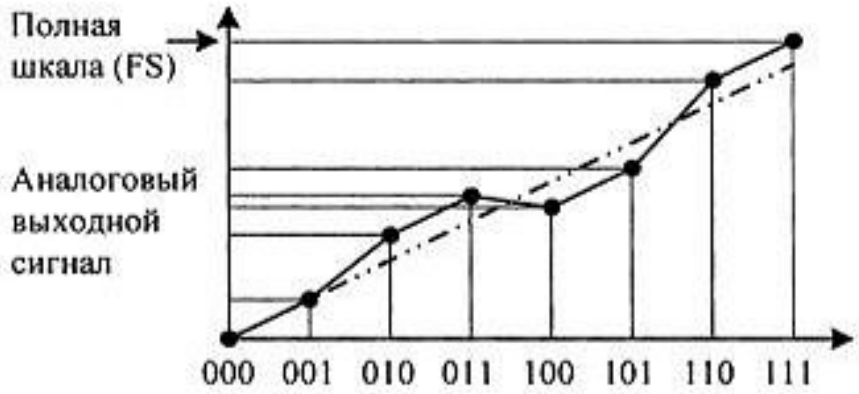


Рис. 13.27. Функция передачи неидеального 3-разрядного ЦАП

Обычно монотонность ЦАП явно оговаривается в техническом описании микросхемы. Однако если дифференциальная нелинейность гарантированно меньше единицы младшего разряда (т. е. $|DNL| \leq 1LSB$), то устройство будет обладать монотонностью, даже если в техническом описании это явно не указывается.

Характеристика аналого-цифрового преобразователя тоже может быть немонотонной, но избыточный DNL в аналого-цифровых преобразователях обычно проявляется в виде пропущенных кодов (рис. 13.28). Пропущенные коды (или немонотонность) в аналого-цифровых преобразователях столь же нежелательны, как и немонотонность в цифроаналоговых преобразователях. Напомню, что потерянные коды возникают при динамической нелинейности, большей значения младшего разряда ($DNL > 1LSB$).

Определение отсутствующих кодов является более сложной задачей по сравнению с определением немонотонности. Все АЦП характеризуются некоторым шумом перехода (transition noise), иллюстрируемым на рис. 13.29 (представьте себе этот шум как частые переключения последней цифры цифрового вольтметра между соседними значениями).

По мере роста разрешающей способности увеличивается диапазон входных сигналов, где уровень шумов перехода кода может достичь или даже превы-

сить значение единицы младшего разряда. В этом случае, особенно в сочетании с отрицательной DNL-погрешностью, могут появиться некоторые (или даже все) коды, где шум перехода будет присутствовать во всем диапазоне значений входных сигналов. В результате, возможно существование некоторых кодов, для которых не найдется такого значения входного сигнала, при котором этот код гарантированно бы появился на выходе, хотя и может существовать некоторый диапазон входного сигнала, при котором иногда будет появляться этот код.

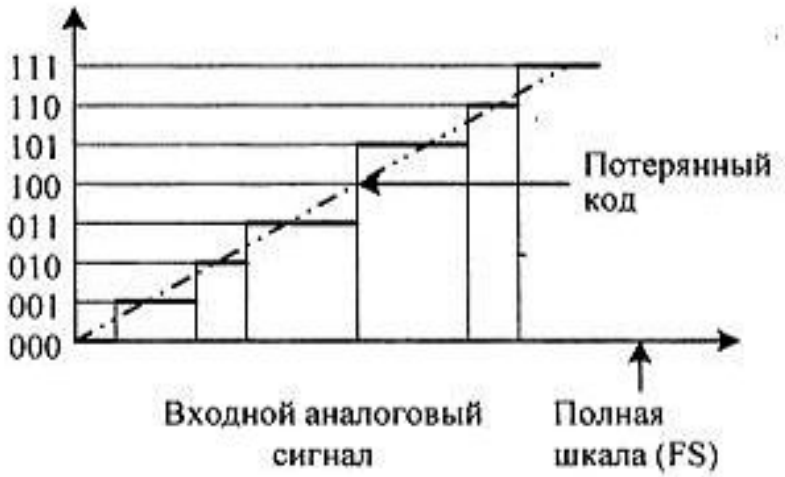


Рис. 13.28. Функция передачи неидеального 3-разрядного АЦП

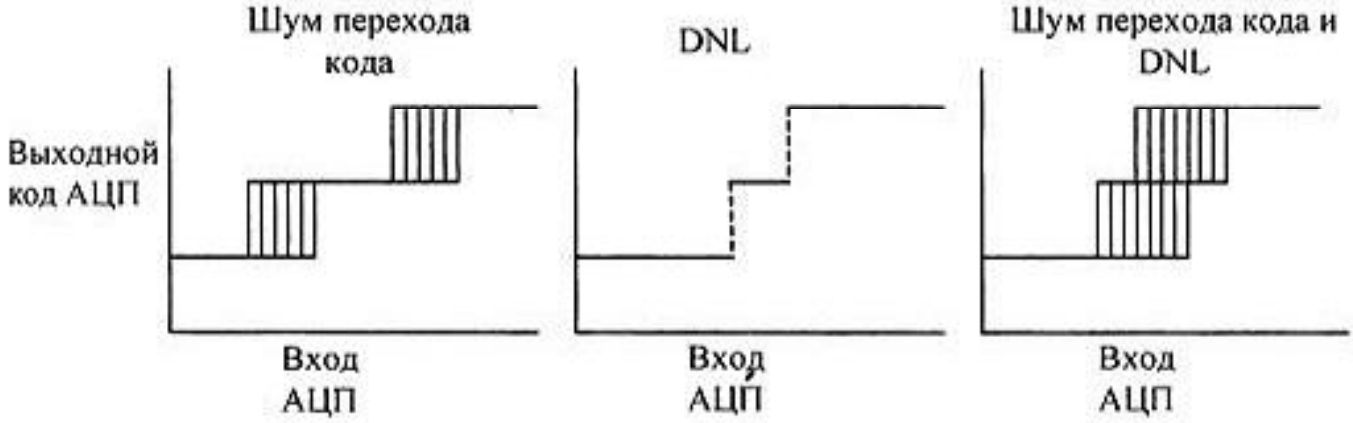


Рис. 13.29. Совместное действие шумов перехода кода и дифференциальной нелинейности (DNL)

Для АЦП с меньшей разрешающей способностью можно определить условие отсутствия пропущенных кодов как наличие гарантированной зоны напряжений или токов (скажем, 0,2 LSB), где отсутствовали бы помехи при любой комбинации шумов перехода и дифференциальной нелинейности.

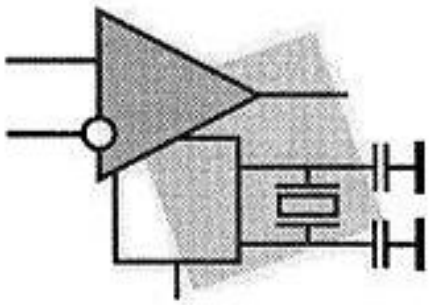
При таком подходе невозможно достичь столь высокой разрешающей способности, которую обеспечивают современные сигма-дельта-АЦП, или даже АЦП с меньшей разрешающей способностью в широкой полосе пропускания.

В этих случаях производитель должен определять уровни шумов и разрешающую способность каким-нибудь другим способом. Не так важно, какой метод используется, но спецификация на микросхему должна содержать четкое определение используемого метода и ожидаемые характеристики микросхемы.

Итоги

В данной главе мы рассмотрели особенности аналого-цифрового и цифроаналогового преобразования. Мы убедились, что, несмотря на то, что цифровая обработка сигналов может обеспечить любой, заранее задаваемый динамический диапазон сигнала, выполнить эту задачу в полной мере не удается. В настоящее время основной проблемой цифровых устройств обработки сигналов остаются узлы преобразования сигнала из аналоговой формы в цифровую и наоборот, при этом требования к аналоговым компонентам схемы и к их конструкции получаются очень жесткими. В настоящее время в основном именно эти узлы определяют конечные параметры цифровых устройств обработки сигналов и устройств управления (контроллеров). Кроме того, мы выяснили, что требования к аналого-цифровым и цифроаналоговым преобразователям существенно отличаются для измерительных устройств, устройств управления объектами различного назначения (в том числе устройствами связи) и для устройств обработки сигналов. Хороший инструментальный АЦП может плохо преобразовывать переменное напряжение. В то же самое время прекрасный преобразователь радиосигнала в цифровую форму может вносить недопустимые искажения при измерении медленно меняющихся напряжений. Соответственно различаются и схемы этих устройств преобразования. Основные схемотехнические решения аналого-цифровых преобразователей и получающиеся характеристики мы рассмотрим в следующей главе.

ГЛАВА 14



Виды аналого-цифровых преобразователей

Как уже упоминалось ранее, аналого-цифровые преобразователи сигналов используются в различных устройствах. Это означает, что к ним предъявляются требования, отличающиеся по быстродействию, количеству разрядов, потребляемой энергии, габаритам и т. д. В настоящее время не существует устройств, обладающих одинаково хорошими характеристиками по всем этим требованиям.

Одни преобразователи обладают прекрасным быстродействием, но большим потреблением энергии, другие обладают прекрасными характеристиками по разрядности, но их быстродействие оставляет желать лучшего.

Рассмотрим внутреннее устройство некоторых наиболее распространенных аналого-цифровых преобразователей.

Параллельные АЦП

Простейшим для понимания принципов работы (но отнюдь не по внутреннему устройству) является параллельный аналого-цифровой преобразователь.

Рассмотрим его работу на примере схемы трехразрядного параллельного АЦП, приведенной на рис. 14.1.

В этой схеме аналоговый сигнал $U_{вх}$ подается на соответствующий вход АЦП. Одновременно на другой его вход подается опорное напряжение U_{REF}. Это напряжение при помощи резистивного делителя, состоящего из резисторов с одинаковым сопротивлением, делится на семь одинаковых уровней.

Основой параллельного преобразователя являются семь аналоговых компараторов, которые сравнивают входной сигнал с опорным напряжением, подаваемым на их второй вход. Аналоговые компараторы представляют собой обычные усилители-ограничители с дифференциальным входом.

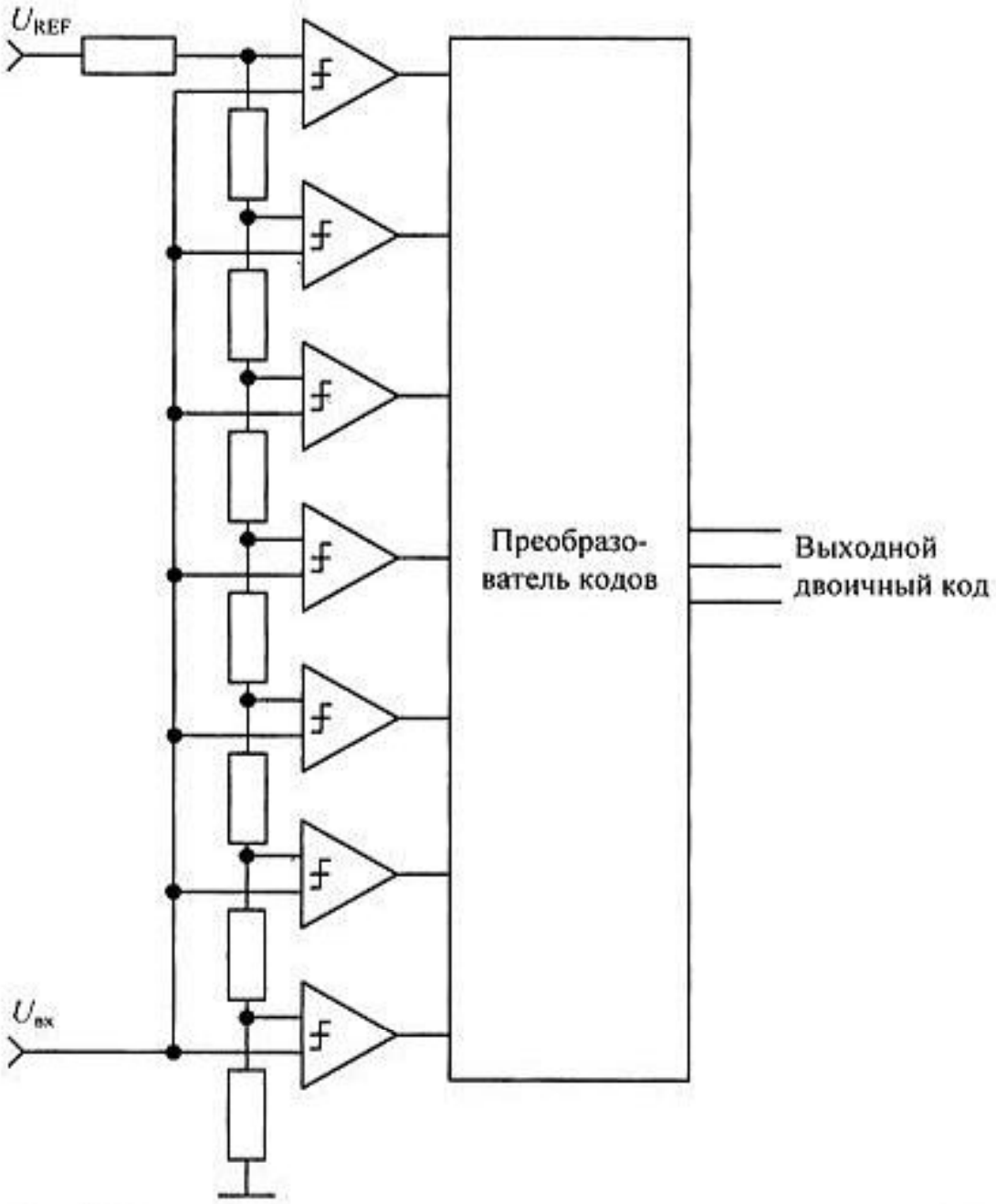


Рис. 14.1. Принципиальная схема трехразрядного параллельного АЦП

Если напряжение на входе преобразователя меньше всех напряжений, подаваемых на опорные входы компараторов, то на всех выходах формируются нулевые уровни сигналов. Код на выходе линейки компараторов будет равен 0000000.

Постепенно повышая уровень входного сигнала, можно превысить напряжение на опорном входе нижнего компаратора. В этом случае на его выходе сформируется уровень логической единицы. Код на выходе линейки компараторов примет значение 0000001. При дальнейшем увеличении уровня сигнала на входе АЦП код будет принимать значения 0000011, 0000111, и т. д. Максимальное значение кода 1111111 будет выдано на выходе аналого-

цифрового преобразователя при превышении входным сигналом значения сигнала на опорном входе самого верхнего компаратора.

Итак, мы достигли напряжения полной шкалы аналого-цифрового преобразователя. Однако, как вы заметили, код, получаемый на выходе линейки компараторов, не является двоичным, поэтому для его приведения к двоичному виду потребуется специальная цифровая схема — преобразователь кодов.

Такие схемы мы уже умеем разрабатывать. Этому мы научились в первой части книги. Если внимательно посмотреть на полученные нами на выходе линейки компараторов коды, то мы увидим, что с таким видом кодов мы уже встречались — это коды, которые мы использовали при построении восьмеричных шифраторов. А это, в свою очередь, означает, что в качестве преобразователя кодов мы можем использовать уже хорошо знакомую нам схему восьмеричного шифратора.

Как видите, у нас получилась достаточно простая и быстродействующая схема. Что может быть быстрее простого устройства сравнения — компаратора! Более того! Мы уже знаем, что высокое быстродействие аналого-цифрового преобразователя нам обычно требуется при оцифровке радио- и видеосигналов. При работе с подобными сигналами нас обычно не интересует абсолютная задержка этого сигнала (в пределах десятков миллисекунд). Нам важнее возможность непрерывно получать поток цифровых отсчетов.

В этом случае следует обратить внимание на то, что при изготовлении компараторов на одном кристалле разброс их параметров, в том числе и времени распространения сигнала с его входа на выход, будет значительно меньше абсолютного значения задержки. В результате частота дискретизации, подаваемая на тактовый вход подобного АЦП, может достигать нескольких гигагерц.

Итак, все хорошо и прекрасно? Но почему же в начале главы было отмечено, что у параллельного АЦП сложное внутреннее устройство? Мы рассмотрели трехразрядный АЦП и получили, что для его работы требуется семь компараторов. А сколько компараторов потребуется для реализации восьмиразрядного АЦП? Как мы уже знаем, количество разрядов должно быть на единицу меньше количества двоичных кодов. Для восьмиразрядного АЦП потребуется уже 256 компараторов, для десятиразрядного — 1023! Именно поэтому параллельные АЦП редко выполняются с разрядностью, большей восьми.

Последовательно-параллельные АЦП

Следующим видом аналого-цифровых преобразователей, занимающим промежуточное место между скоростными параллельными АЦП и наиболее распространенными АЦП последовательного приближения, являются последовательно-параллельные АЦП.

Рассмотрим работу последовательно-параллельного АЦП на примере восьмиразрядного АЦП. Структурная схема этого АЦП приведена на рис. 14.2.

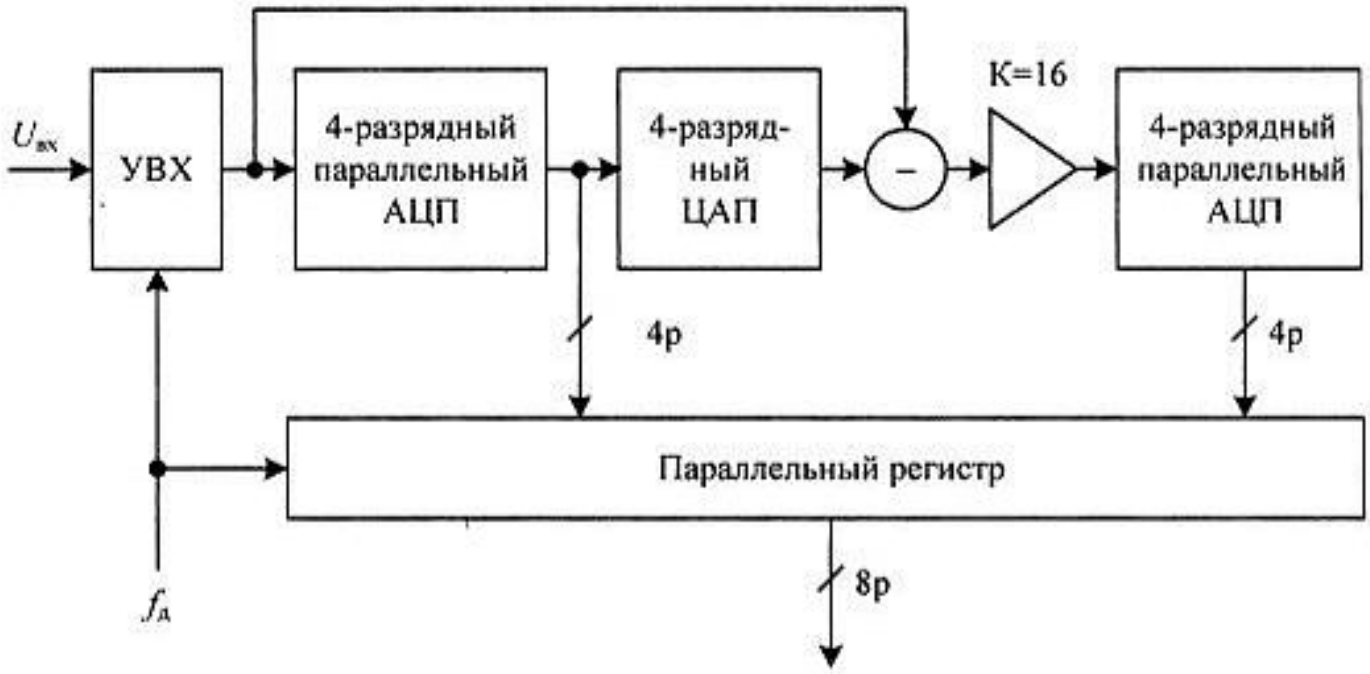


Рис. 14.2. Структурная схема восьмиразрядного последовательно-параллельного АЦП

В схеме восьмиразрядного последовательно-параллельного АЦП используются два параллельных четырехразрядных АЦП. Второй аналого-цифровой преобразователь оцифровывает ошибку квантования, выделяемую при помощи цифроаналогового преобразователя и аналогового вычитателя. Если бы мы преобразовывали входной сигнал в цифровую форму без погрешности, то на выходе АЦП (а затем и на выходе цифроаналогового преобразователя) мы бы получили точную копию входного сигнала, однако это не так. Поэтому на выходе аналогового вычитателя формируется сигнал ошибки преобразования.

Для того чтобы в схеме можно было бы использовать одинаковые АЦП, сигнал ошибки преобразования первого аналого-цифрового преобразователя усиливается в 16 раз. В результате уровень сигнала на входе второго АЦП равен уровню сигнала на входе первого АЦП, а значит, можно использовать схему, полностью идентичную первому аналого-цифровому преобразователю. Следует отметить, что вычитатели, как правило, выполняют с использованием операционных усилителей, поэтому обычно в составе последовательно-параллельного преобразователя используется усиливающий вычитатель.

Что же мы выиграли в результате усложнения схемы? Так как разрядность параллельных преобразователей снижена вдвое, то для их реализации в случае, приведенном на рис. 14.2, потребуются только $2 \times 15 = 30$ компараторов.

Для реализации восьмиразрядного АЦП, как это уже упоминалось в предыдущей главе, нам бы потребовалось 255 компараторов. То есть выигрыш по сложности реализации схемы составляет почти десять раз!

Теперь давайте оценим, во сколько же раз мы проиграли в быстродействии? Прежде чем мы сможем сформировать на выходе восьмиразрядный двоичный код, необходимо, чтобы сигнал был преобразован в цифровую форму первым АЦП, снова преобразован в аналоговую форму цифроаналоговым преобразователем. Затем должен быть сформирован и усилен сигнал ошибки, и этот сигнал должен быть снова оцифрован. В результате описанных действий время преобразования входного аналогового сигнала возрастает, по крайней мере, в четыре раза.

✧ *Обратите внимание* — время преобразования, а не тактовая частота! Как мы определили в предыдущей главе, время преобразования в параллельном АЦП в несколько раз больше периода тактовой частоты (частоты дискретизации аналогового сигнала). Все это время сигнал на входе преобразователя не должен меняться. Это означает, что в составе последовательно-параллельного АЦП должно находиться устройство выборки и хранения.

Тем не менее, как и в случае с параллельным аналого-цифровым преобразователем, быстродействие всей схемы в целом может быть увеличено за счет применения конвейерной обработки. Достаточно разбить алгоритм преобразования на несколько этапов, которые могут выполняться одновременно.

Подобная схема восьмиразрядного последовательно-параллельного преобразователя приведена на рис. 14.3.

В этой схеме в то время, когда осуществляется преобразование в цифровую форму сигнала ошибки, формируется сигнал ошибки следующего отсчета сигнала. Пока формируется сигнал ошибки следующего отсчета сигнала, осуществляется формирование старших четырех разрядов выходного кода. Единственная трудность заключается в том, что необходимо совместить сформированные старшие и младшие разряды во времени. Это осуществляется за счет задержки старших разрядов в цифровой линии задержки, собранной на параллельных регистрах.

В приведенной на рис. 14.3 схеме выходной отсчет сигнала появится только через три тактовых импульса. Все последующие отсчеты входного аналогового сигнала будут появляться с каждым очередным тактовым импульсом.

Итак, подведем итоги. Последовательно-параллельный АЦП способен осуществлять преобразование сигнала с большей разрядностью по сравнению с параллельным АЦП. Однако он обладает меньшим быстродействием, приблизительно равным времени задержки параллельного АЦП. Последовательно-параллельные АЦП способны формировать цифровой поток данных со скоростью несколько сотен миллионов отсчетов в секунду.

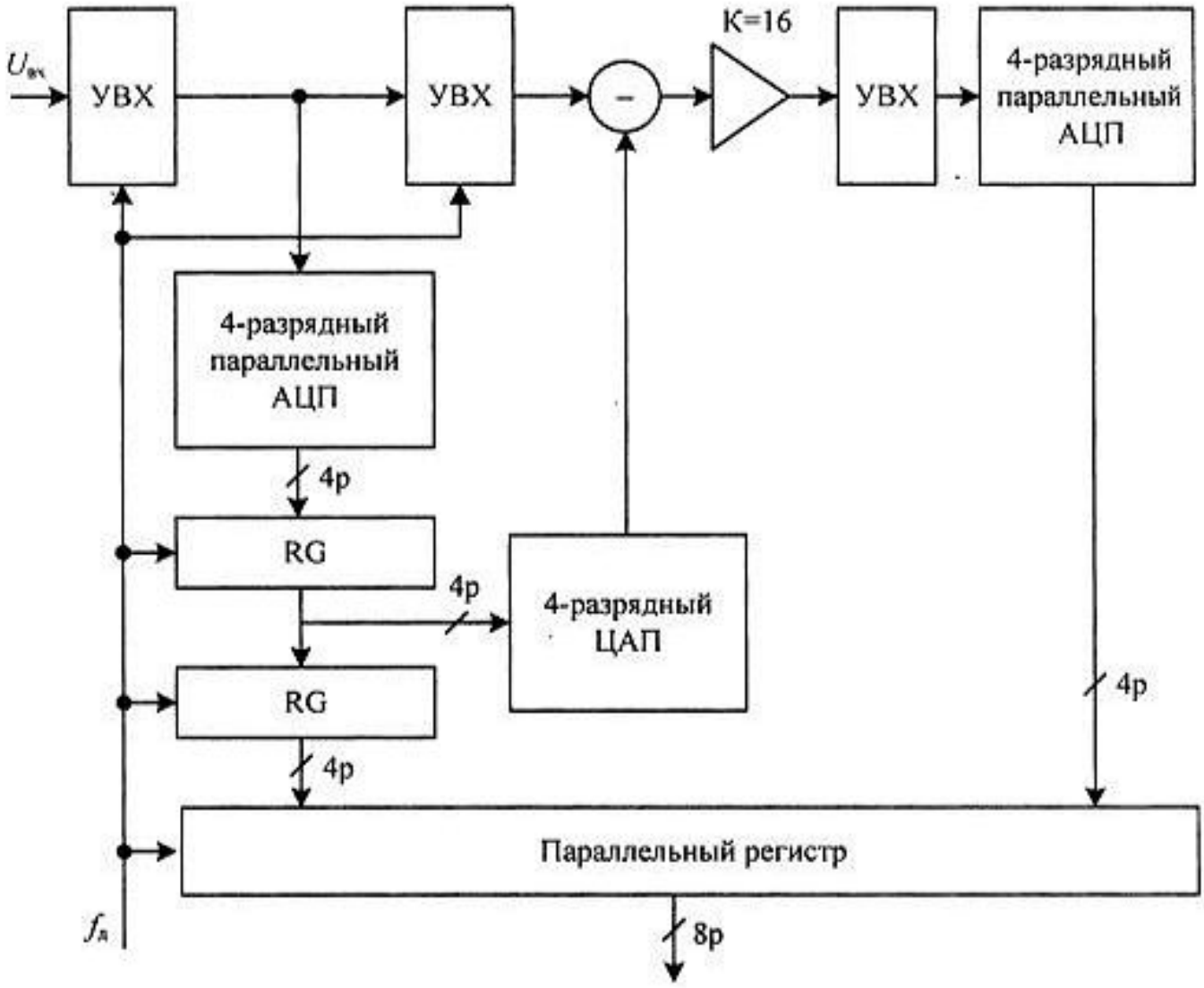


Рис. 14.3. Структурная схема конвейерного восьмиразрядного последовательно-параллельного АЦП

АЦП последовательного приближения

Наиболее распространенным видом аналого-цифровых преобразователей в настоящее время являются АЦП последовательного приближения. Эти преобразователи позволяют в течение одного периода тактового сигнала получить один двоичный разряд.

При измерении каких-либо объектов мы обычно последовательно увеличиваем точность измерения. Например, при измерении длины мы сначала определяем ее в метрах, затем добавляем к полученному значению остающиеся десятки сантиметров, потом остаток в сантиметрах и т. д. То есть при каждом последующем измерении точность увеличивается на один десятичный разряд. Подобным образом можно проводить измерения и в двоичной системе счисления. В этом случае каждый раз точность измерения будет возрастать

ровно в два раза. Подобный процесс измерения напряжения иллюстрируется рис. 14.4.

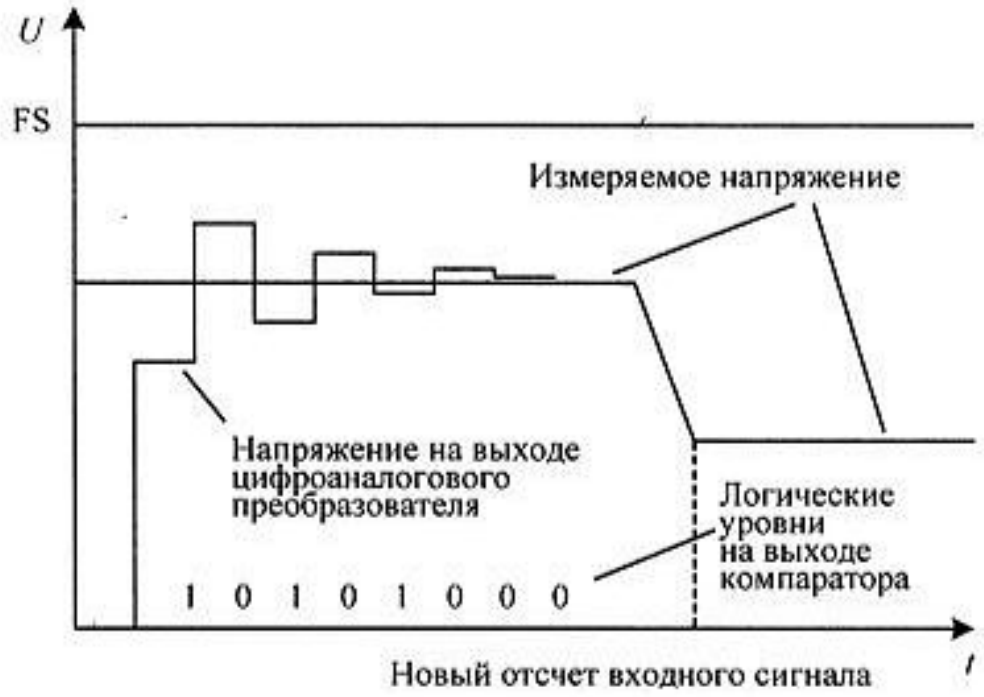


Рис. 14.4. Временные диаграммы напряжений на входах компаратора АЦП послёдовательного приближения

При измерении неизвестного расстояния оно сравнивается с эталоном длины — линейкой. Но где же взять эталонные напряжения? Для этого можно воспользоваться цифроаналоговым преобразователем. Если на его вход подавать цифровые коды, то на его выходе будут появляться напряжения, соответствующие этим цифровым кодам. Для формирования необходимых для измерения цифровых кодов служит специальная схема, называемая регистром последовательного приближения.

Для сравнения неизвестного напряжения, поступающего с выхода устройства выборки и хранения, с эталонными напряжениями, поступающими с выхода цифроаналогового преобразователя, воспользуемся уже известным нам аналоговым компаратором. Структурная схема аналого-цифрового преобразователя последовательного приближения приведена на рис. 14.5.

В первый момент времени после поступления первого тактового импульса на выходе регистра последовательного приближения формируется код половины полной шкалы преобразователя. Этот код соответствует двоичному числу 01111111. При подаче этого кода на входы цифроаналогового преобразователя на его выходе появится напряжение, соответствующее половине полной шкалы входных напряжений (или, что то же самое, половине опорного напряжения $U_{оп}$, подаваемого на соответствующий вход цифроаналогового преобразователя).

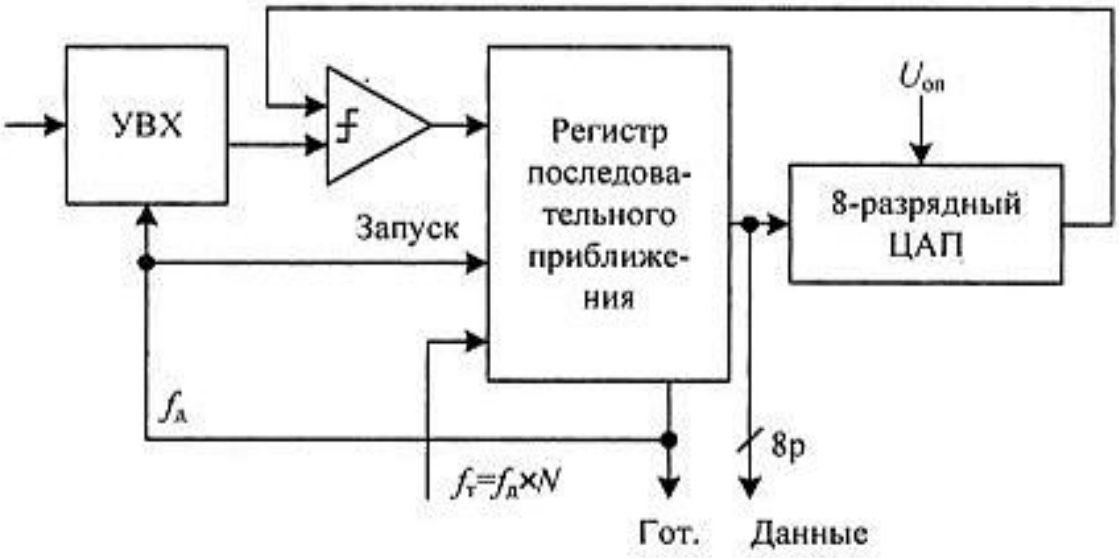


Рис. 14.5. Структурная схема АЦП последовательного приближения

При поступлении следующих тактовых импульсов этот код будет сдвигаться вправо, обеспечивая тем самым уменьшение веса разрядов ровно вдвое. Таким образом, если после первого тактового импульса на выходе цифроаналогового преобразователя присутствует половина полной шкалы, то после второго тактового импульса там будет присутствовать четверть, затем одна восьмая часть полной шкалы, и т. д.

В примере, приведенном на рис. 14.4, измеряемое напряжение превышает значение половины полной шкалы АЦП, а значит, на выходе аналогового компаратора появится уровень логической единицы. При поступлении второго тактового импульса этот сигнал запишется в старший разряд регистра последовательного приближения. В результате на выходе этого регистра появится код 10111111, а значит, напряжение на выходе ЦАП станет равным 3/4 от напряжения полной шкалы. Если бы напряжение на выходе УВХ оказалось меньше напряжения, поступающего с выхода ЦАП, то на выходе компаратора появился бы нулевой потенциал, и в регистр последовательного приближения был бы записан код 00111111, а значит, на выходе ЦАП сформировалось бы напряжение 1/2 от напряжения полной шкалы.

В примере, приведенном на рис. 14.4, напряжение на выходе ЦАП при втором измерении превысит напряжение с выхода УВХ, поэтому на выходе компаратора появится нулевой уровень. При поступлении третьего тактового импульса этот сигнал запишется во второй разряд регистра последовательного приближения, поэтому код на его выходе станет равным 10011111. На этот раз напряжение на выходе ЦАП уменьшится на 1/8 $U_{оп}$ от предыдущего значения.

Итак, на вход регистра последовательного приближения поступило три тактовых импульса, и мы получили два разряда цифрового кода. После поступ-

ления на вход регистра последовательного приближения девяти тактовых импульсов мы получим полный 8-разрядный двоичный код, соответствующий входному напряжению. В примере, приведенном на рис. 14.4, этот код равен 10101000.

После завершения преобразования на управляющем выходе регистра последовательного приближения "Гот." появляется нулевой потенциал, показывающий, что преобразование закончено.

Итак, для полного преобразования аналогового сигнала в цифровую форму АЦП последовательного приближения требуется, как минимум, $N+1$ тактовых импульсов (один такт на выдачу половинного напряжения и N тактов для получения N двоичных разрядов).

АЦП последовательного приближения могут работать как в режиме одиночного преобразования, так и в режиме создания непрерывного потока данных. На рис. 14.5 этот аналого-цифровой преобразователь включен в режиме непрерывного преобразования входного сигнала. В этом режиме тактовая частота должна подаваться от высокостабильного генератора.

Если требуется производить одиночное аналого-цифровое преобразование в определенные моменты времени, то обратная связь с выхода готовности на вход запуска регистра последовательного приближения разрывается и преобразование начинается сразу же после поступления импульса на вход запуска. В этом случае высокой стабильности от генератора тактовой частоты не требуется.

АЦП последовательного приближения используются на частотах преобразования от единиц килогерц до десятков мегагерц. При этом удается достигнуть точности преобразования до 18 двоичных разрядов.

Сигма-дельта-АЦП

В ряде случаев этой точности недостаточно. Самую высокую точность преобразования на настоящее время — 24 двоичных разряда достигают сигма-дельта-АЦП. В этих аналого-цифровых преобразователях для достижения такой высокой разрешающей способности совмещены достижения как аналоговой, так и цифровой техники.

Для того чтобы понять, как работает этот вид аналого-цифровых преобразователей, давайте сначала рассмотрим, каким образом может быть представлен сигнал для последующей передачи по цифровой линии связи. Первый вариант — это применение импульсно-кодовой модуляции. В этом случае каждый отсчет сигнала преобразуется в цифровой эквивалент, и это число передается по линии связи. Именно такой вариант мы и рассматривали до

настоящего времени. Структурная схема линии связи с импульсно-кодовой модуляцией приведена на рис. 14.6.

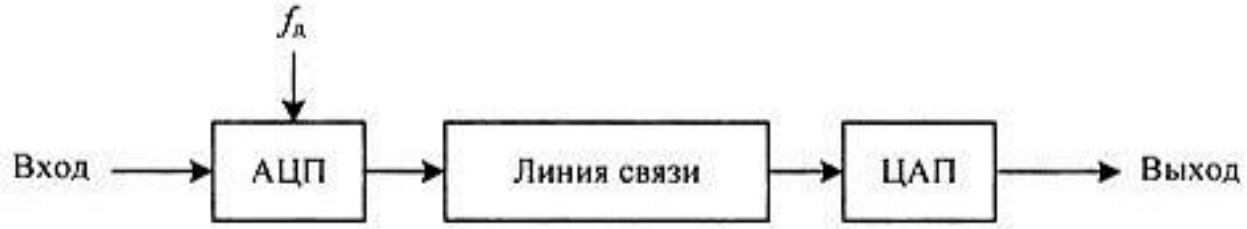


Рис. 14.6. Структурная схема линии связи с импульсно-кодовой модуляцией

Шумы квантования в такой системе равномерно распределены по частоте и не зависят от частоты входного сигнала. График уровня шума квантования на выходе аналого-цифрового преобразователя, в зависимости от частоты, приведен на рис. 14.7.

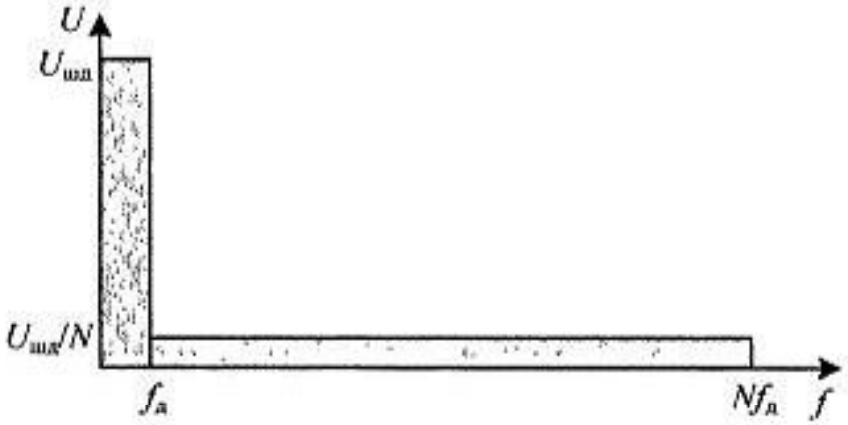


Рис. 14.7. Зависимость уровня шумов квантования дельта-модуляции от частоты

Существует альтернативный вариант передачи сигнала в цифровой форме. В этом варианте передаются (или запоминаются) не абсолютные значения сигнала, а только приращения сигнала на заданном отрезке времени. Такой вид представления сигнала в цифровой форме называется дельта-модуляцией. Структурная схема линии связи с дельта-модуляцией приведена на рис. 14.8.

Сигнал на выходе этой системы полностью соответствует сигналу на выходе схемы, приведенной на рис. 14.6. Теперь давайте обратим внимание, что в схеме используется два интегратора, но если переместить интегратор в цепи обратной связи на вход компаратора, то этот интегратор будет выполнять те же функции, что и интегратор на выходе схемы, а значит, он не потребуется — мы обойдемся одним интегратором. Новая схема дельта-модулятора приведена на рис. 14.9.

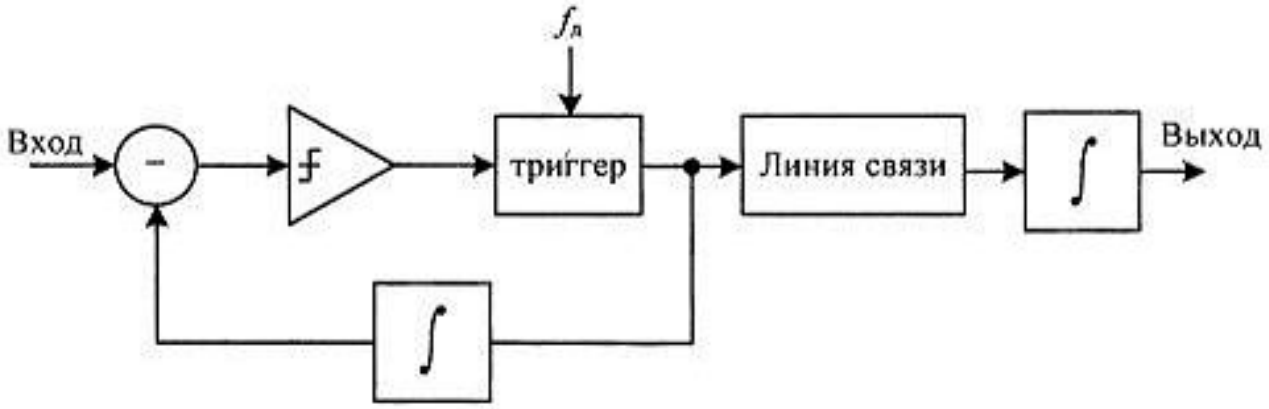


Рис. 14.8. Структурная схема линии связи с дельта-модуляцией

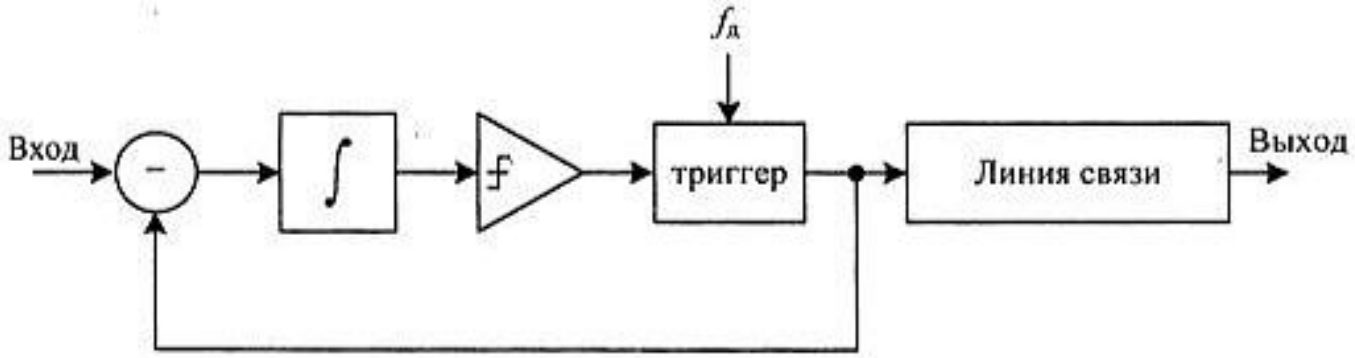


Рис. 14.9. Структурная схема дельта-модулятора первого порядка

Частота дискретизации в линии связи с дельта-модуляцией обычно выбирается выше частоты дискретизации импульсно-кодовой модуляции. Только в этом случае качество сигнала на выходе этой системы становится сравнимым с качеством сигнала на выходе ИКМ-системы. При этом чем выше частота дискретизации по отношению к частоте преобразуемого сигнала, тем меньше будет ошибка дискретизации этого сигнала.

Обратите на это внимание. В системе с дельта-модуляцией ошибка квантования не распределена равномерно по всем частотам, а растёт с ростом частоты преобразуемого сигнала. График уровня шума квантования на выходе дельта-модулятора, в зависимости от частоты, приведен на рис. 14.10.

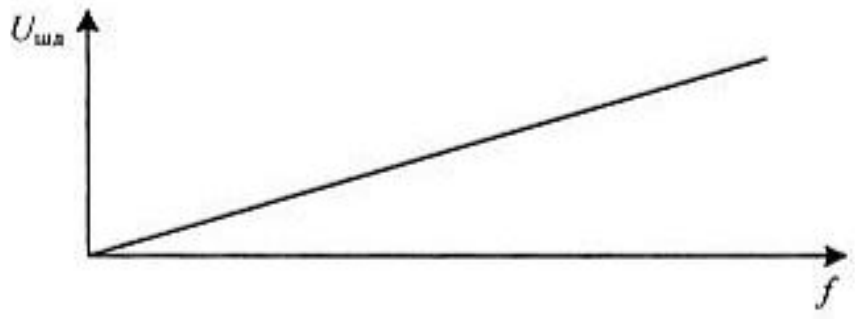


Рис. 14.10. Зависимость уровня шумов квантования дельта-модуляции от частоты

Именно этим свойством дельта-модуляции мы и воспользуемся для уменьшения шумов квантования в аналого-цифровом преобразователе. Нас обычно интересует область частот от постоянного тока до верхней частоты сигнала. Если при помощи цифрового фильтра отфильтровать высокочастотные составляющие шума квантования, то можно значительно увеличить отношение сигнал/шум сигнала, представленного в цифровой форме.

При этом отношение сигнал/шум в данной схеме будет значительно выше по отношению к сигналу на выходе схемы, приведенной на рис. 14.6. Теперь зададимся вопросом — если мы получили выигрыш по отношению сигнал/шум за счет неравномерного распределения шумов квантования, то нельзя ли сделать эту зависимость еще более неравномерной, т. е. как бы "вытеснить" шумы квантования в область верхних частот, где они будут подавлены цифровым фильтром нижних частот.

Для реализации этой идеи можно воспользоваться дельта-модулятором второго или третьего порядка. Зависимость шумов квантования от частоты на выходе дельта-модуляторов различного порядка приведена на рис. 14.11.

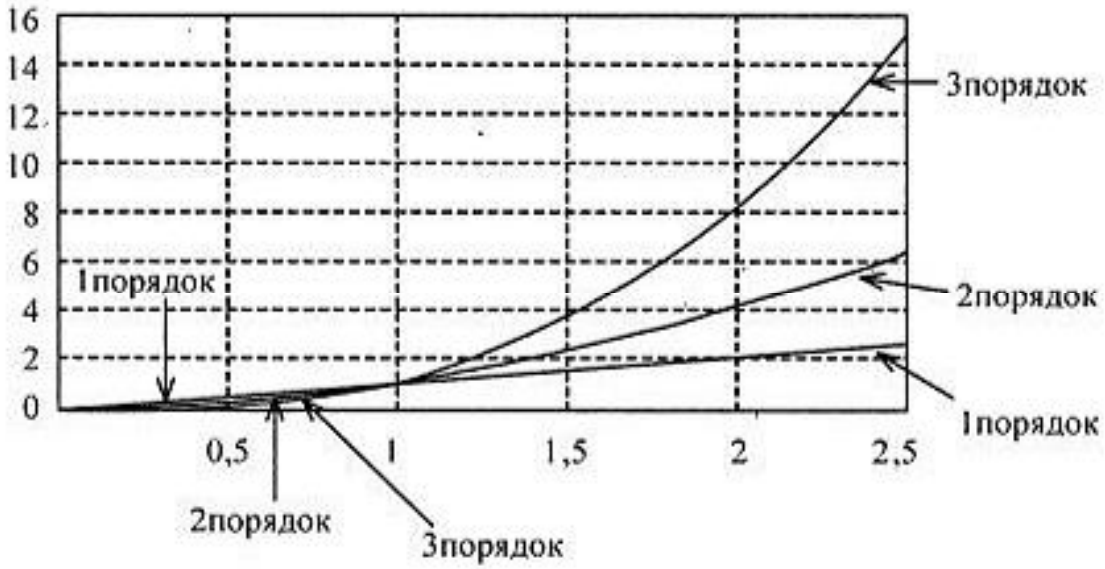


Рис. 14.11. Частотная зависимость шумов квантования на выходе дельта-модуляторов различного порядка

В настоящее время используются дельта-модуляторы третьего порядка. Модуляторы более высокого порядка не используются, т. к. их схемы являются потенциально неустойчивыми и могут самовозбуждаться. Структурная схема дельта-модулятора третьего порядка приведена на рис. 14.12.

Следующим блоком, определяющим высокие характеристики ΣΔ-АЦП, является цифровой фильтр. Именно в сумматорах цифрового фильтра множество одноразрядных цифровых отсчетов входного сигнала превращается в много-

разрядные числа, которые затем поступают на выход аналого-цифрового преобразователя.

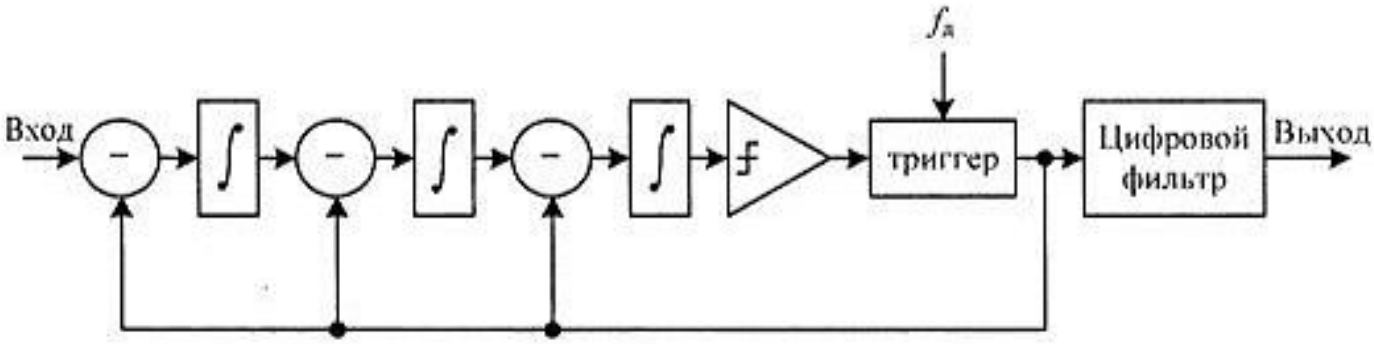


Рис. 14.12. Структурная схема дельта-модулятора третьего порядка

Основной задачей цифрового фильтра является уничтожение всех частотных составляющих выше верхней частоты полезного сигнала, поэтому на выходе этого цифрового фильтра можно значительно уменьшить частоту дискретизации, следовательно для синхронизации ΣΔ-АЦП требуется частота, в несколько сотен раз выше частоты его выходного потока данных.

Обычно полоса рабочих частот ΣΔ-АЦП, построенного по схеме, приведенной на рис. 14.12, не превышает нескольких десятков герц, поэтому для более широкополосных сигналов, таких как звуковой сигнал, применяется несколько измененная схема. В ней в качестве преобразователя аналог-цифра используется не одноразрядный АЦП (аналоговый компаратор), а параллельный АЦП. Структурная схема подобного ΣΔ-АЦП приведена на рис. 14.13.

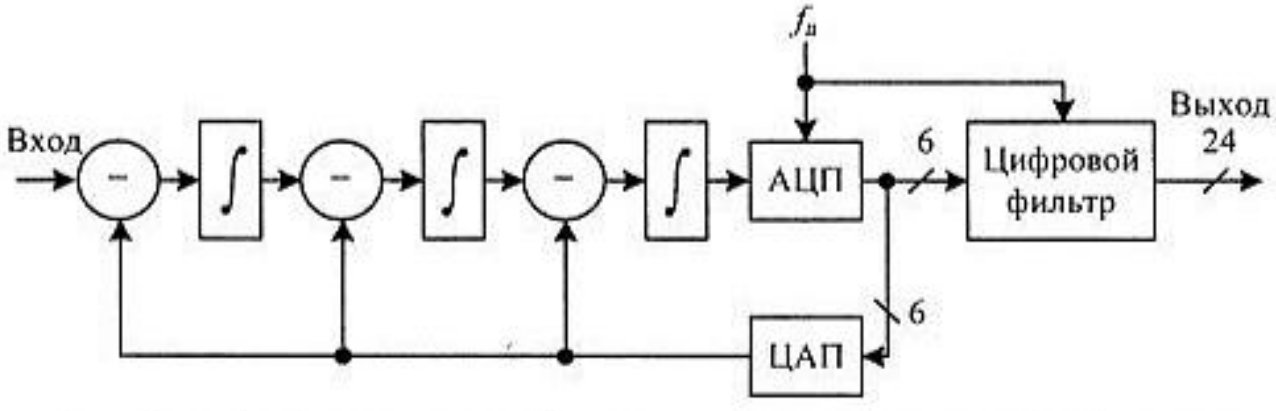


Рис. 14.13. Структурная схема ΣΔ-АЦП с применением параллельного АЦП

Еще одной распространенной задачей является преобразование в цифровую форму узкополосных радиосигналов промежуточной частоты. В этом случае необходимо очищать от шумов квантования не область частот около нулевой частоты, а некоторую область частот около промежуточной частоты. В этом случае в качестве интеграторов используются не фильтры низкой частоты

(RC-цепочки), а полосовые фильтры (обычно применяются LC-контура). Структурная схема ΣΔ-АЦП, осуществляющего оцифровку промежуточной частоты, приведена на рис. 14.14.

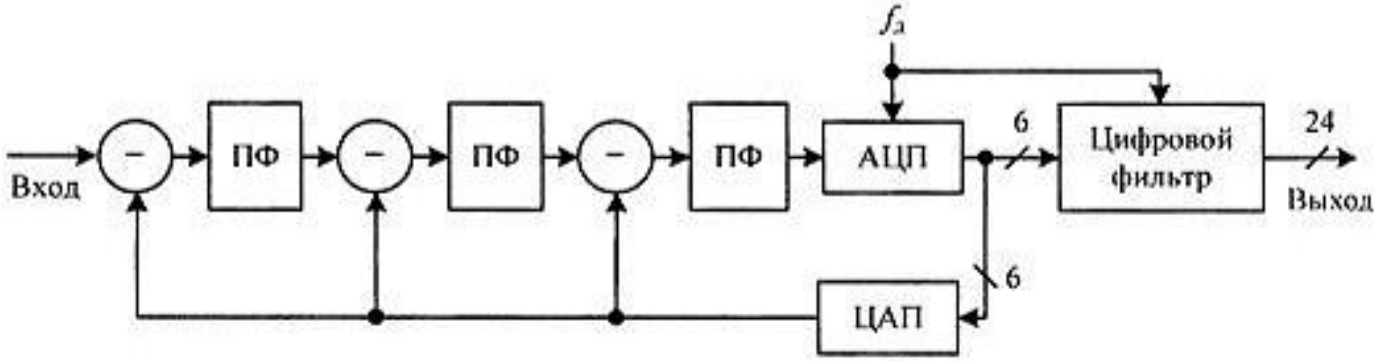
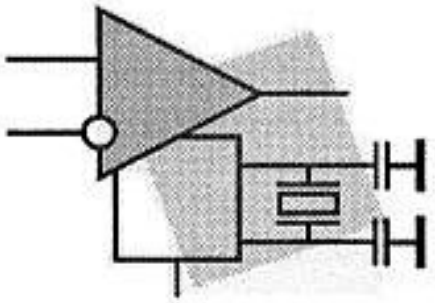


Рис. 14.14. Структурная схема ΣΔ-АЦП промежуточной частоты

Итоги

В данной главе мы рассмотрели особенности реализации основных типов аналого-цифровых преобразователей. Здесь хотелось бы особенно подчеркнуть, что выбор конкретного типа преобразователя зависит от вида решаемой задачи. Особое внимание следует обращать на синхронизацию аналого-цифровых преобразователей. Если в схемах управления (контроллерах) можно допустить синхронизацию АЦП от микропроцессора, то в схемах цифровой обработки сигналов подобный подход недопустим. В них синхронизация аналого-цифровых и цифроаналоговых преобразователей должна осуществляться от высокостабильного малошумящего генератора.

ГЛАВА 15



Основные блоки микросхем цифровой обработки сигналов

В настоящее время получили распространение схемы, в которых выходной сигнал формируется непосредственно в цифровой форме. Затем этот сигнал преобразуется в аналоговую форму при помощи цифроаналогового преобразователя. В составе этих микросхем широко используются сумматоры, умножители и цифровые фильтры. Изучение устройств прямого цифрового синтеза начнем с простейшего устройства обработки цифровых сигналов — двоичного сумматора.

Двоичные сумматоры

Важным элементом цифровых устройств, выполняющих арифметическую обработку цифровой информации, является сумматор. Построение двоичных многоразрядных сумматоров обычно начинается с одноразрядного сумматора по модулю 2. В табл. 15.1 приведена таблица истинности этого сумматора. Ее можно получить, исходя из правил суммирования одиночных бит в двоичной арифметике, которые мы рассматривали ранее.

Таблица 15.1. *Таблица истинности сумматора по модулю 2*

Вход X	Вход Y	Выход
0	0	0
0	1	1
1	0	1
1	1	0

В соответствии с принципами реализации принципиальной схемы по произвольной таблице истинности, рассмотренными в предыдущих главах, полу-

чим принципиальную схему сумматора по модулю 2. Формирование этой схемы ничем не отличается от примеров, рассмотренных ранее. Как и раньше выделяем строки, содержащие единицу в выходном сигнале. Они реализуются элементами "2И". Нулевые потенциалы входных сигналов в этих строках превращаются в единичные при помощи инверторов. Объединение выходов логических элементов в один производится логическим элементом "2ИЛИ". Полученная принципиальная схема сумматора по модулю 2 приведена на рис. 15.1.

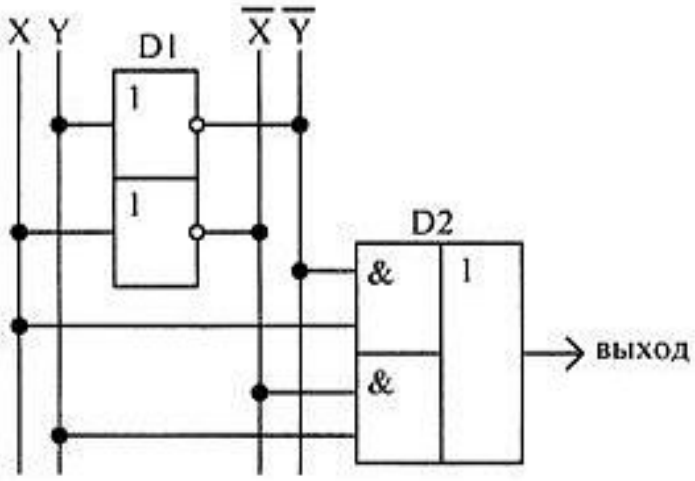


Рис. 15.1. Принципиальная схема, реализующая таблицу истинности сумматора по модулю 2

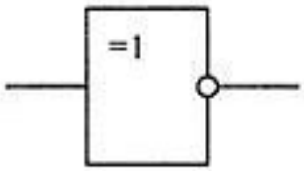


Рис. 15.2. Условно-графическое обозначение схемы, выполняющей логическую функцию "исключающего ИЛИ"

Сумматор по модулю 2 (для двух входов его схема полностью совпадает со схемой исключающего "ИЛИ") изображается на принципиальных схемах, как показано на рис. 15.2.

Сумматор по модулю 2 выполняет суммирование без учета переноса между двоичными разрядами. В полном двоичном сумматоре его необходимо учитывать, поэтому требуются элементы, позволяющие формировать перенос в следующий двоичный разряд. Таблица истинности такого устройства, называемого полусумматором, приведена в табл. 15.2.

✧ *Обратите внимание*, что сигналы в приведенной таблице истинности расположены в порядке, принятом для схем, т. е. в соответствии с тем, что сигнал распространяется слева направо. В результате перенос, который имеет двоичный вес больший, по сравнению с суммируемыми разрядами, записан правее. В математике принят другой порядок разрядов числа. Старший разряд на бумаге записывается самым левым, а младший разряд — самым правым. В результате может возникнуть путаница. Чтобы этого не произошло, приведу десятичный эквивалент каждой строки таблицы истинности полусумматора (табл. 15.2).

Первая строка этой таблицы истинности получена из арифметического выражения $0 + 0 = 0_{10}(00_2)$. Вторая строка получена из арифметического выражения $0 + 1 = 1_{10}(01_2)$. Третья строка получена из арифметического выражения $1 + 0 = 1_{10}(01_2)$. Четвертая строка получена из арифметического выражения $1 + 1 = 2_{10}(10_2)$.

Таблица 15.2. *Таблица истинности полусумматора*

Вход A	Вход B	Выход S	Выход PO	Математическое выражение
0	0	0	0	$0 + 0 = 0_{10}(00_2)$
0	1	1	0	$0 + 1 = 1_{10}(01_2)$
1	0	1	0	$1 + 0 = 1_{10}(01_2)$
1	1	0	1	$1 + 1 = 2_{10}(10_2)$

В соответствии с принципами построения произвольной таблицы истинности получим принципиальную схему полусумматора. Схема, соответствующая таблице истинности, содержащейся в табл. 15.2, приведена на рис. 15.3.

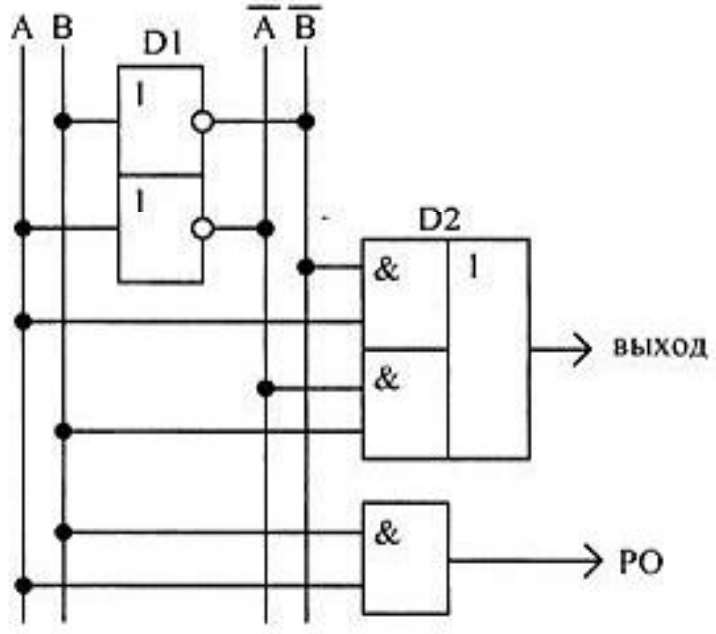


Рис. 15.3. Принципиальная схема цифрового устройства, реализующего таблицу истинности полусумматора

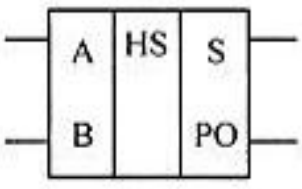


Рис. 15.4. Условно-графическое обозначение полусумматора

Полусумматоры выпускаются в виде отдельных микросхем и используются в качестве отдельных модулей в составе больших интегральных микросхем, поэтому ГОСТом предусмотрено условно-графическое обозначение полусумматора. Оно приведено на рис. 15.4.

Полусумматор формирует перенос в следующий разряд, но не может учитывать перенос из предыдущего разряда, поэтому он и называется полусумматором. В результате такой особенности полусумматор не может использоваться в качестве отдельного устройства. Практический интерес представляет полный сумматор.

Таблицу истинности полного одноразрядного двоичного сумматора (табл. 15.3), как и таблицу истинности полусумматора, можно получить из правил арифметического суммирования двоичных чисел. В обозначении входов и выходов полного сумматора использовано следующее правило: в качестве входов использованы одноразрядные двоичные числа A и B; сумма — это одноразрядное двоичное число S; перенос обозначен буквой P; для обозначения входа переноса используется сочетание букв PI (I — сокращение от английского слова *input*, вход); для обозначения выхода переноса используется сочетание букв PO (O — сокращение от английского слова *output*, выход).

Таблица 15.3. *Таблица истинности полного двоичного одноразрядного сумматора*

№	PI	A	B	S	PO	Математическое выражение
1	0	0	0	0	0	$0+0+0=0_{10}(00_2)$
2	0	0	1	1	0	$0+0+1=1_{10}(01_2)$
3	0	1	0	1	0	$0+1+0=1_{10}(01_2)$
4	0	1	1	0	1	$0+1+1=2_{10}(10_2)$
5	1	0	0	1	0	$1+0+0=1_{10}(01_2)$
6	1	0	1	0	1	$1+0+1=2_{10}(10_2)$
7	1	1	0	0	1	$1+1+0=2_{10}(10_2)$
8	1	1	1	1	1	$1+1+1=3_{10}(11_2)$

Теперь, точно так же как и в предыдущих случаях, в соответствии с правилами построения принципиальной схемы по произвольной таблице истинности получим схему полного двоичного одноразрядного сумматора. Схема, соответствующая таблице истинности, содержащейся в табл. 15.3, приведена на рис. 15.5. Эта схема построена с использованием СДНФ.

Схему полного одноразрядного сумматора можно минимизировать. Для этого достаточно посмотреть на две последние строки его таблицы истинности. Мы можем увидеть, что сигнал переноса в них не зависит от сигнала, присутст-

вующего на входе B. Поэтому этот вход можно не заводить на вход схемы "И".

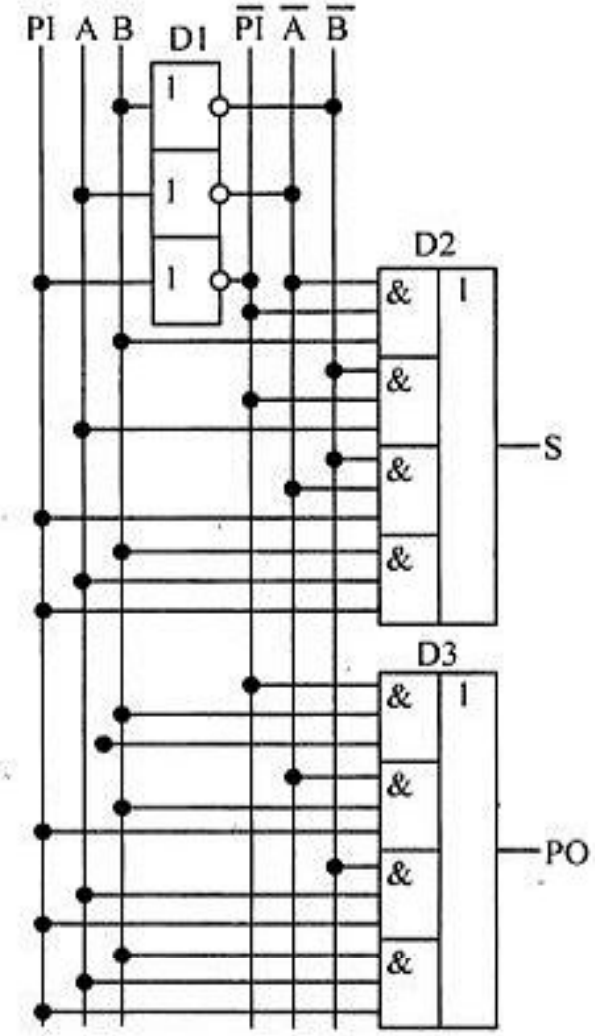


Рис. 15.5. Принципиальная схема, реализующая таблицу истинности полного двоичного одноразрядного сумматора

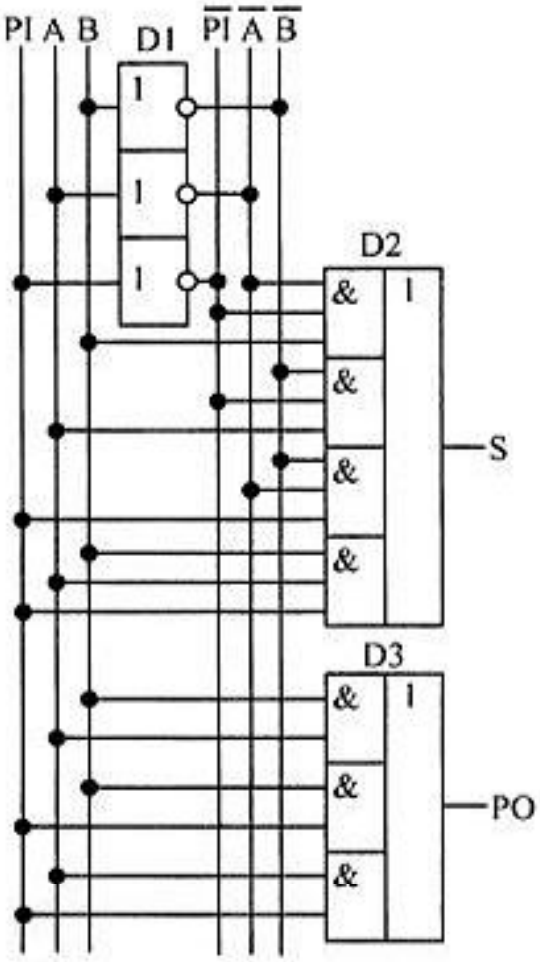


Рис. 15.6. Минимизированная принципиальная схема, реализующая таблицу истинности полного двоичного одноразрядного сумматора

В результате вместо двух нижних элементов "3И" можно воспользоваться одним двухвходовым элементом "2И". Точно такая же ситуация возникает, если рассмотреть строки 6 и 8. В этом случае лишним оказывается вход A. То есть и в этом случае можно обойтись одним двухвходовым логическим элементом "2И".

Вход PI оказывается лишним в выражениях, описывающих строки 4 и 8. В результате описанных действий принципиальная схема формирования переноса в следующий разряд сумматора упрощается и приобретает вид, приведенный на рис. 15.6.

Примером одноразрядного двоичного сумматора может служить микросхема средней интеграции К155ИМ1. Условно-графическое обозначение полного двоичного одноразрядного сумматора показано на рис. 15.7.

Обычно для выполнения вычислений в схемах цифровой обработки сигналов недостаточно точности одноразрядного сумматора. В них применяются 16- или даже 40-разрядные двоичные сумматоры.

Для того чтобы получить многоразрядный сумматор из полученного выше одноразрядного сумматора, достаточно соединить входы и выходы переносов соответствующих двоичных разрядов. Принципиальная схема четырехразрядного сумматора, реализованная на четырех одноразрядных сумматорах, приведена на рис. 15.8.

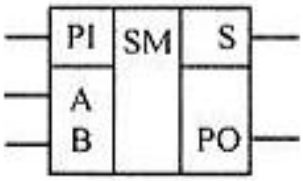


Рис. 15.7. Условно-графическое изображение полного двоичного одноразрядного сумматора

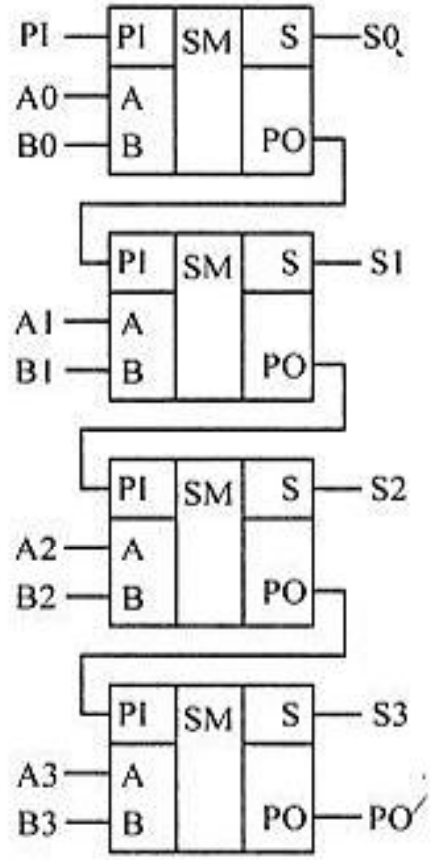


Рис. 15.8. Принципиальная схема четырехразрядного двоичного сумматора

На приведенной схеме двоичный вес разрядов суммируемых чисел A и B отображен непосредственно в названии цепи. Например, цепь A0 передает нулевой разряд числа A, а цепь B2 — второй разряд числа B. Названия входа PI и выхода переноса PO не изменены.

Полный двоичный четырехразрядный сумматор изображается на схемах с использованием условно-графического обозначения, показанного на рис. 15.9. ✧ *Обратите внимание*, что в этом обозначении входы двоичного

слова A объединены в отдельное поле. Точно так же объединены входы двоичного слова B. Вход и выход сигналов переноса на приведенном условно-графическом обозначении микросхемы тоже выделены в отдельные поля. Это не обязательно, и не требуется ГОСТом, однако изображенная таким образом микросхема намного более наглядно отображает свои функции.

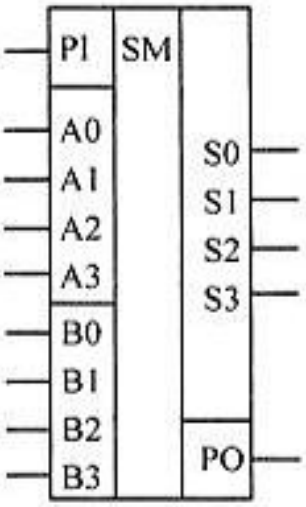


Рис. 15.9. Условно-графическое обозначение полного двоичного 4-разрядного сумматора на схемах

Приведенная на рис. 15.8 схема не оптимизирована по быстродействию, она служит лишь для пояснения принципа действия многоразрядного двоичного сумматора. В применяемых на практике схемах никогда не допускают последовательного распространения переноса через все разряды многоразрядного сумматора, т. к. это снижает его быстродействие.

Для увеличения скорости работы двоичного сумматора используется отдельная схема формирования переносов для каждого двоичного разряда. Таблицу истинности для такой схемы легко получить из алгоритма суммирования двоичных чисел, а затем применить хорошо известные нам принципы построения цифрового устройства по произвольной таблице истинности.

Следующим, широко используемым в схемах цифровой обработки сигналов устройством, является цифровой двоичный умножитель. Эти устройства используются как в схемах масштабирования (усилители или аттенюаторы) или гетеродинирования сигналов, так и в составе цифровых фильтров.

Цифровые умножители

Умножение чисел в двоичном виде производится подобно умножению в десятичной системе счисления. Как мы помним из школьного курса, легче всего осуществлять умножение в столбик. При реализации этого алгоритма по-

требуется перемножить каждый разряд множимого на соответствующий разряд множителя.

Рассмотрим в качестве примера умножение двух четырехразрядных двоичных чисел. Пусть требуется умножить число 1011_2 (11_{10}) на число 1101_2 (13_{10}). В результате умножения мы ожидаем получить число 10001111_2 (143_{10}). Выполним операцию умножения в столбик в двоичной системе, как это показано на рис. 15.10.

```
      1101
    × 1011
------------
      1101
+   1101
+  0000
+ 1101
------------
  10001111
```

Рис. 15.10. Выполнение операции умножения в столбик

Для формирования произведения требуется вычислить четыре частичных произведения. ✧ *Обратите внимание*, что в двоичной арифметике требуется выполнять умножение только на числа 0 и 1. Это означает, что нужно либо суммировать множимое к сумме остальных частичных произведений, либо нет. В результате для формирования частичного произведения можно воспользоваться логическими элементами "2И", подключенными к каждому двоичному разряду множимого.

Для формирования частичного произведения, кроме операции умножения на один разряд, требуется осуществлять его сдвиг влево на число разрядов, соответствующее весу разряда множителя. Сдвиг можно осуществить простым соединением соответствующих разрядов частичных произведений к необходимым разрядам двоичного сумматора.

Для того чтобы принципиальная схема умножителя была похожа на пример двоичного умножения, приведенный на рис. 15.10, используем условно-графические изображения микросхем, где входы расположены сверху, а выходы снизу. Это разрешено ГОСТом. В полном соответствии с алгоритмом умножения в столбик нам потребуются три четырехразрядных сумматора.

Принципиальная схема умножителя, реализующая алгоритм двоичного умножения в столбик, приведена на рис. 15.11.

Формирование частичных произведений в этой схеме осуществляют цифровые микросхемы D1, D3, D5 и D7. В этих микросхемах в одном корпусе содержится сразу четыре логических элемента "2И".

Сумматор, выполненный на микросхеме D6, суммирует первое и второе частные произведения. При этом младший разряд первого частного произведе-

ния не нуждается в суммировании (см. рис. 15.11). Поэтому он подается на выход умножителя непосредственно (разряд M0).

Второе частное произведение должно быть сдвинуто на один разряд. Это осуществляется тем, что младший разряд выходного числа сумматора D6 соединяется со вторым разрядом произведения (M1). Но тогда первое частное произведение необходимо сдвинуть на один разряд влево по отношению ко второму частному произведению!

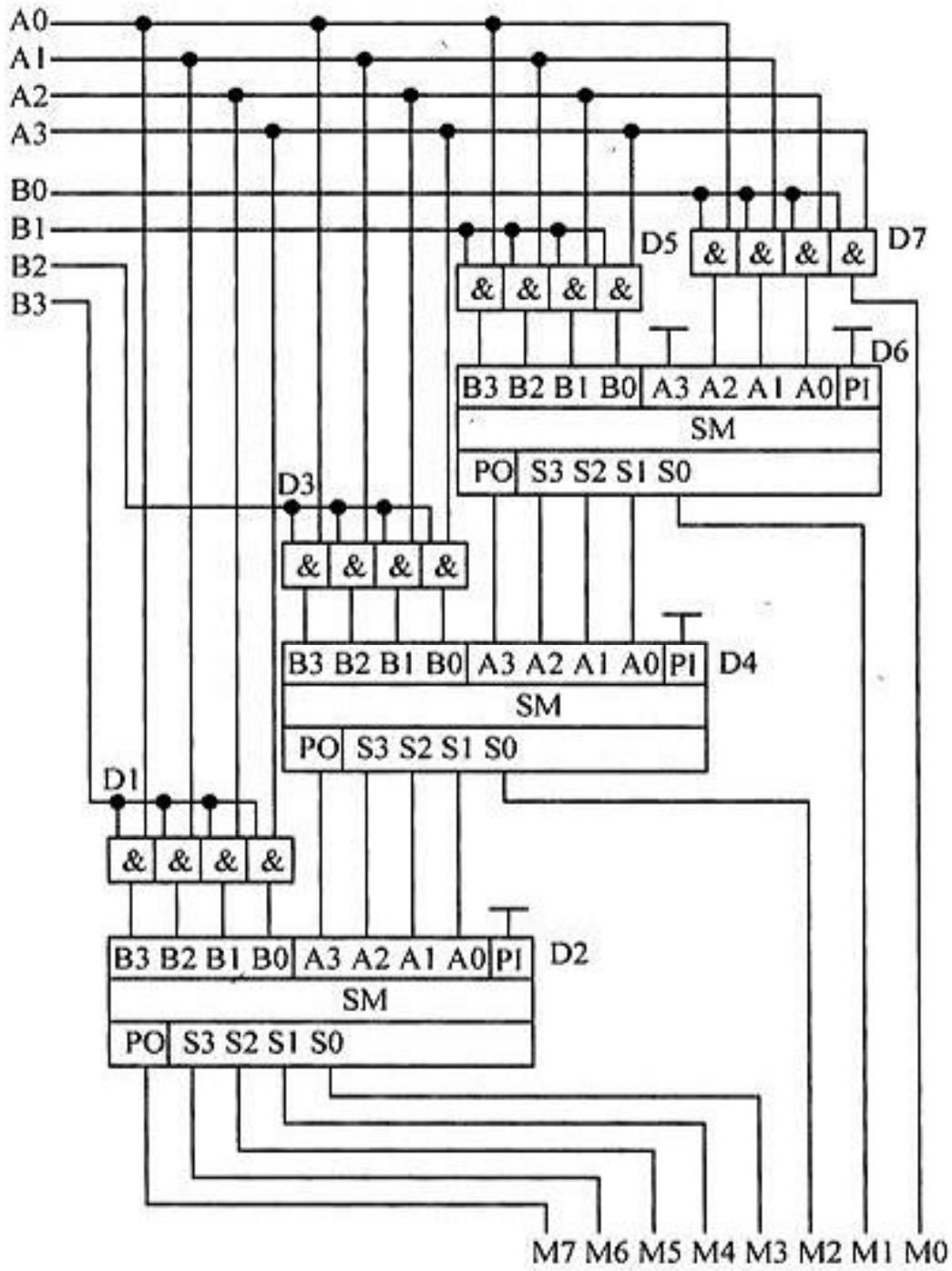


Рис. 15.11. Принципиальная схема матричного умножителя 4×4

Это арифметическое действие выполняется за счет того, что младший разряд группы входов A соединяется с первым разрядом частного произведения, первый разряд группы входов A соединяется со вторым разрядом частного произведения, и т. д. Однако старший разряд группы входов A не с чем соединять!

Для того чтобы разрешить это противоречие вспомним, что если записать слева от числа ноль, то значение исходного числа не изменится, поэтому мы должны этот разряд соединить с общим проводом схемы, добавляя тем самым ноль в старший разряд первого частного произведения.

Точно таким же образом осуществляется суммирование к результату третьего и четвертого частного произведения. Эту операцию выполняют микросхемы D4 и D2 соответственно. Отличие в построении схемы заключается только в том, что здесь не нужно задумываться о старшем разряде предыдущей суммы, ведь предыдущая микросхема сумматора формирует сигнал переноса для последующей микросхемы.

Если внимательно посмотреть на схему умножителя, приведенную на рис. 15.11, то можно увидеть, что она образует матрицу, сформированную цепями, по которым передаются разряды числа A и числа B. В точках пересечения этих цепей находятся логические элементы "2И". Именно по этой причине умножители, реализованные по данной схеме, получили название матричных умножителей.

Скорость работы схемы, приведенной на рис. 15.11, определяется максимальным временем распространения сигнала по самому длинному пути. Это путь, проходящий через микросхемы D7, D6, D4, D2. Время работы схемы можно сократить, если сумматоры располагать не последовательно друг за другом, как это предполагается алгоритмом двоичного умножения (пример умножения приведен на рис. 15.10), а суммировать частичные произведения попарно, затем суммировать пары частичных произведений и т. д. В этом случае время выполнения операции умножения значительно сократится.

Особенно заметен выигрыш в быстродействии при построении многоразрядных умножителей. Однако ничего не бывает бесплатно. В обмен на увеличение быстродействия придется заплатить увеличением разрядности сумматоров, а значит, сложностью схемы. Если сумматоры частных произведений останутся той же разрядности, что и ранее, то разрядность сумматоров пар частичных произведений должна быть увеличена на единицу.

Разрядность сумматоров четверок частичных произведений будет на два разряда больше разрядности сумматоров частичных произведений, т. к. при суммировании четырех чисел их значение в наиболее неблагоприятном случае может увеличиться в четыре раза, и т. д.

Цифровые матричные умножители широко применяются в схемах обработки сигналов для изменения коэффициента передачи устройства, для реализации преобразователей частоты, и как составляющая часть цифровых фильтров.

Теперь рассмотрим следующий блок, не менее часто используемый в схемах формирования цифровой обработки и сигналов. Это постоянные запоминающие устройства.

Постоянные запоминающие устройства

При выборе формы сигналов, передаваемых по линиям связи, очень важно, чтобы этот сигнал не искажался при передаче по линиям связи. В цифровых устройствах не искажаются прямоугольные сигналы. Однако как только мы попытаемся передать их через аналоговые цепи, такие как усилители мощности, фильтры или антенны, прямоугольные сигналы будут искажены.

Для передачи по аналоговым цепям идеально подходят синусоидальные сигналы. Именно они не искажаются при распространении по таким цепям. У этих сигналов может измениться только амплитуда и фаза. Поэтому если мы собираемся заняться обработкой сигналов, то мы должны уметь генерировать сигналы такой формы.

Самый простой способ получить синусоидальную функцию — это считать ее из таблицы. А это в свою очередь означает, что эту таблицу где-то надо хранить.

Еще одна распространенная ситуация, при которой требуется хранить цифровые значения при выключенном питании, — это весовые коэффициенты фильтров с конечной импульсной характеристикой.

Итак, для разработки устройств обработки сигналов нам потребовалось запоминающее устройство. Причем такое устройство, которое может сохранять информацию и при выключении питания. Такое устройство получило название постоянное запоминающее устройство — ПЗУ.

Масочное ПЗУ

В цифровых устройствах удобнее всего хранить информацию в двоичном коде. Но ведь в этом случае можно просто соединять соответствующий бит слова с шиной питания или общим проводом, т. е. в качестве запоминающей ячейки может работать металлизированная перемычка.

Теперь обратим внимание на то, что при формировании сигнала отсчеты амплитуды сигнала нам будут требоваться последовательно, т. е. все данные не требуются одновременно, поэтому простейшие устройства для запоминания постоянной информации можно построить на рассмотренных нами в преды-

дущих главах мультиплексорах. Схема постоянного запоминающего устройства, построенного на основе обычного мультиплексора, приведена на рис. 15.12.

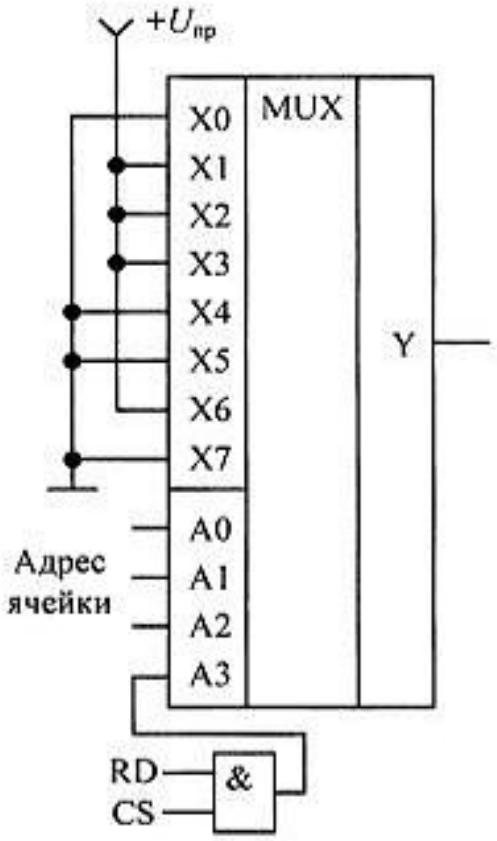


Рис. 15.12. Схема простейшего постоянного запоминающего устройства, построенная на мультиплексоре

В схеме, приведенной на рис. 15.12, построено постоянное запоминающее устройство на восемь одноразрядных ячеек. Запоминание конкретного бита в одноразрядную ячейку производится присоединением входа мультиплексора к источнику питания (запись единицы). Запись нуля осуществляется присоединением того же входа мультиплексора к общему проводу. Выбор конкретной ячейки памяти осуществляется при помощи адресных входов A0 ... A2. В приведенной на рис. 15.12 схеме это входы управления мультиплексора.

В ряде случаев постоянные запоминающие устройства выполняются в виде универсальных схем. Во многих радиоэлектронных устройствах информация на выходе ПЗУ не требуется постоянно. Она должна быть предоставлена только по специальному запросу. Этот запрос формирует сигнал RD. Название сигнала RD расшифровывается как read (чтение). Сигнал чтения можно завести на внутренний дешифратор мультиплексора, т. е. воспользоваться его управляющим входом, как это показано на рис. 15.12. В результате, содержимое ячейки памяти ПЗУ появится на его выходе только при активном

уровне сигнала чтения RD. При всех других условиях выход микросхемы будет оставаться в высокоомном состоянии.

Обычно при построении устройств памяти решается еще одна задача — это задача расширения объема памяти. Расширить объем памяти можно с помощью дополнительного дешифратора адреса. Но этот дешифратор нужно каким-либо способом подключать к микросхемам памяти для того, чтобы запрещать или разрешать работу выбираемых микросхем.

Для подключения дополнительного дешифратора адреса в микросхеме ПЗУ служит еще один вход — CS (chip select — выбор кристалла). У постоянного запоминающего устройства эта функция совпадает с функцией чтения содержимого памяти, поэтому сигнал выбора кристалла CS и сигнал чтения RD можно объединить при помощи логического элемента "2И".

На принципиальных схемах устройство запоминания постоянной информации (ПЗУ) обозначается так, как показано на рис. 15.13.

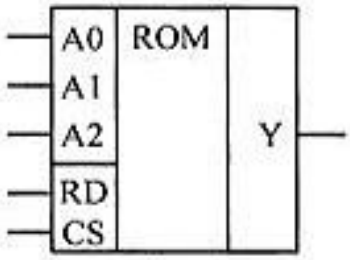


Рис. 15.13. Обозначение постоянного запоминающего устройства на принципиальных схемах

На этом рисунке приведено условно-графическое обозначение, соответствующее принципиальной схеме устройства, показанной на рис. 15.13. Надпись ROM в среднем поле обозначения микросхемы является сокращением от английских слов read only memory (память доступная только для чтения).

Для целей цифровой обработки сигналов при хранении отсчетов амплитуды сигнала недостаточно одного двоичного разряда. Обычно для запоминания конкретного значения напряжения используются многоразрядные двоичные числа.

Для того чтобы увеличить разрядность ячейки памяти ПЗУ, микросхемы, рассмотренные ранее, можно соединять параллельно. При этом информационные выходы и записанная в одноразрядных ячейках информация естественно остаются независимыми. Схема параллельного соединения одноразрядных ПЗУ для реализации многоразрядного запоминающего устройства приведена на рис. 15.14, а условно-графическое обозначение многоразрядного ПЗУ на принципиальных схемах — на рис. 15.15.

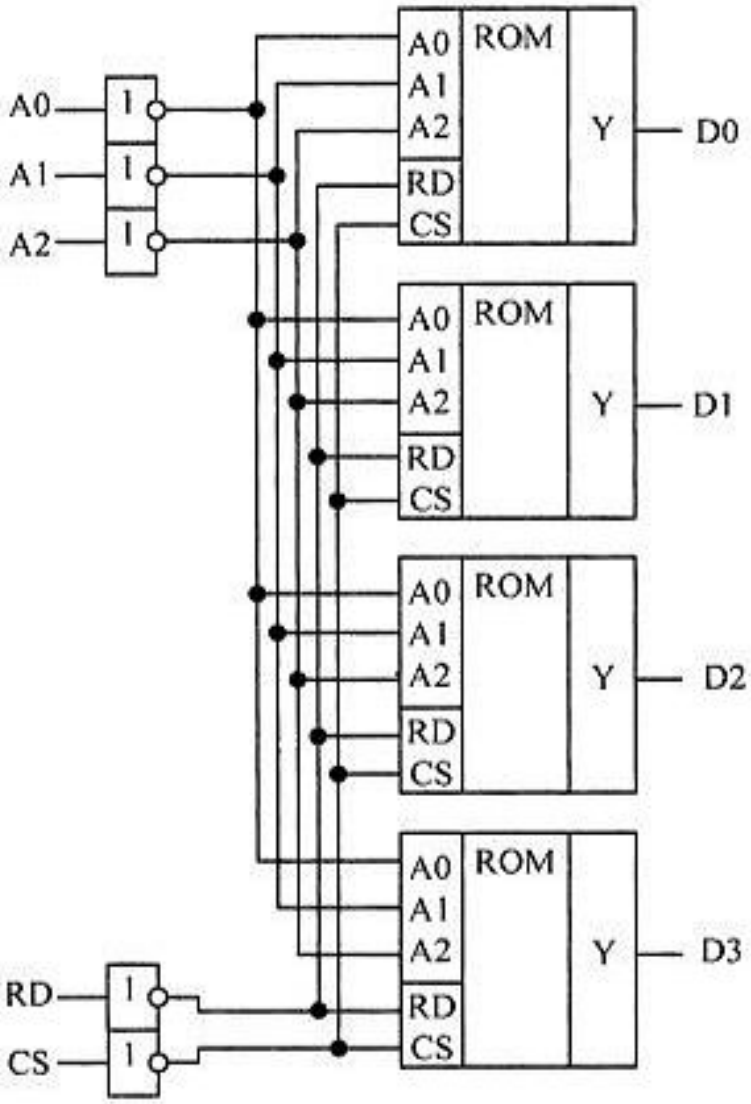


Рис. 15.14. Схема многоразрядного ПЗУ

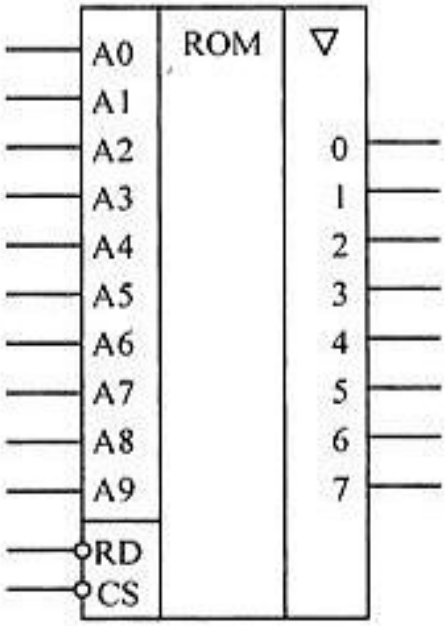


Рис. 15.15. Условно-графическое обозначение многоразрядного масочного постоянного запоминающего устройства

Как видно на рис 15.14, адресные входы схемы объединяются параллельно. При этом возрастает входной ток схемы по каждому из адресных входов. Для того чтобы этого не происходило, на адресных входах обычно применяются усилители сигнала. В качестве усилителей цифрового сигнала, как и в предыдущих примерах, можно использовать самые обыкновенные инверторы, как это показано на рис. 15.14. С точно такой же целью поставлены инверторы и на входах чтения и выбора кристалла.

В реальных схемах ПЗУ запись информации производится при помощи последней операции производства микросхемы — металлизации. Металлизация производится при помощи маски, поэтому такие ПЗУ получили название масочных ПЗУ.

Еще одно отличие реальных микросхем от упрощенной модели, приведенной выше, — это использование в качестве дешифратора адреса не только мультиплексора, но и демультиплексора. Такое схемное решение позволяет пре-

вратить одномерную запоминающую структуру в двухмерную и, тем самым, существенно сократить объем дешифратора адреса ПЗУ. Реализация двухмерного ПЗУ приведена на рис. 15.16.

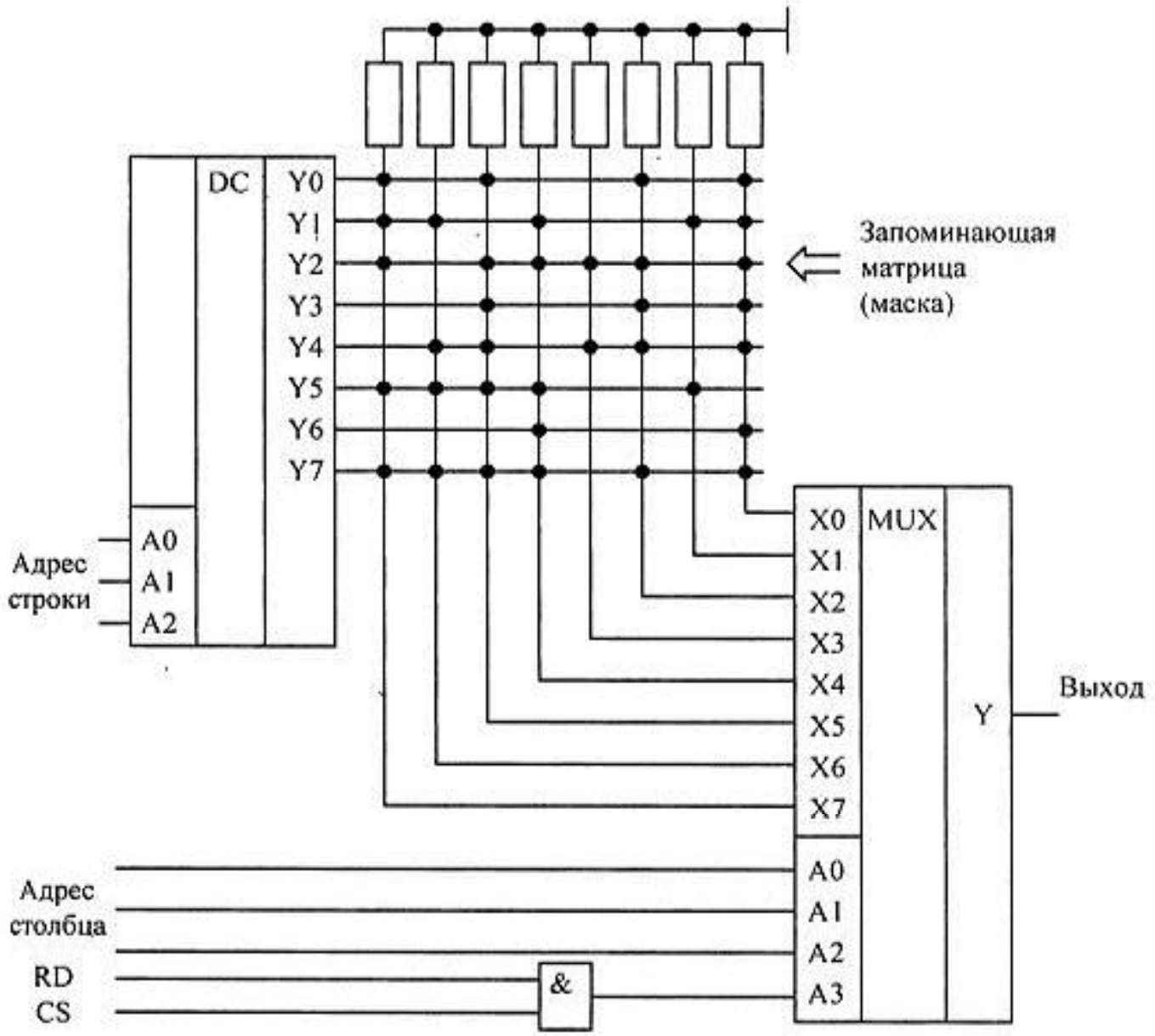


Рис. 15.16. Схема масочного постоянного запоминающего устройства

Условно-графическое обозначение микросхемы масочного ПЗУ приведено на рис. 15.15. Надпись ROM в центральной части микросхемы является сокращением от английских слов read only memory (память доступная только по чтению).

Программирование масочного ПЗУ производится на заводе-изготовителе, что очень неудобно для мелких и средних серий производства, не говоря уже о стадии разработки устройства. Естественно, что для крупносерийного производства масочные ПЗУ являются самым дешевым видом ПЗУ, и поэтому широко применяются до настоящего времени.

ПЗУ с записанной в нем таблицей синуса относится к таким схемам. Поэтому такие ПЗУ широко распространены в настоящее время. Еще чаще эти ПЗУ встречаются в составе микросхем, предназначенных для цифровой обработки сигналов — сигнальных процессорах и микросхемах прямого цифрового синтеза (DDS).

Программируемые постоянные запоминающие устройства

Для мелких и средних серий производства радиоаппаратуры были разработаны микросхемы, которые можно программировать в специальных устройствах — программаторах. В этих микросхемах постоянное соединение проводников в запоминающей матрице заменяется плавкими перемычками, изготовленными из поликристаллического кремния. Поликристаллический кремний наносится сверху на оксид кремния, используемый в кремниевых микросхемах как изолятор.

Так как внутренняя схема такого ПЗУ не отличается от масочного ПЗУ, то для иллюстрации внутреннего устройства этой схемы можно воспользоваться рис. 15.16, предполагая при этом, что все точки пересечения вертикальных и горизонтальных проводов являются плавкими поликремниевыми перемычками.

При производстве микросхемы ПЗУ изготавливаются все перемычки, что эквивалентно записи во все биты ячейки памяти логических единиц. Запись нуля в нужные биты ячейки памяти производится в специальном устройстве, называемом программатором. Эта операция выполняется уже на этапе разработки или производства аппаратуры, а это значит, что такой вид ПЗУ более удобен при мелком и среднесерийном производстве.

В процессе программирования на выводы питания и выходы микросхемы подается повышенное питание. При этом если на выход микросхемы подается напряжение питания (логическая единица), то на обоих концах перемычки будет присутствовать один и тот же потенциал, а значит, через перемычку ток протекать не будет и перемычка останется неповрежденной. Эта ситуация иллюстрируется рис. 15.17.

Если же на выход микросхемы подать низкий уровень напряжения (присоединить этот вывод ПЗУ к корпусу), то через перемычку будет протекать ток, как это показано на рис. 15.18. Этот ток испарит плавкую поликремниевую перемычку, и при последующем считывании информации из этой ячейки будет считываться логический ноль.

Микросхемы, работающие по такому принципу, называются программируемыми ПЗУ (ППЗУ) и изображаются на принципиальных схемах так, как это показано на рис. 15.19. Надпись PROM в центральной части микросхемы

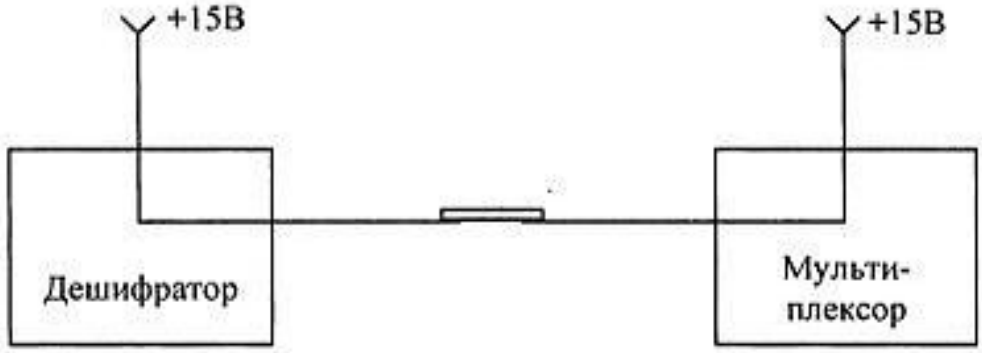


Рис. 15.17. Запись в ячейку программируемого постоянного запоминающего устройства логической единицы

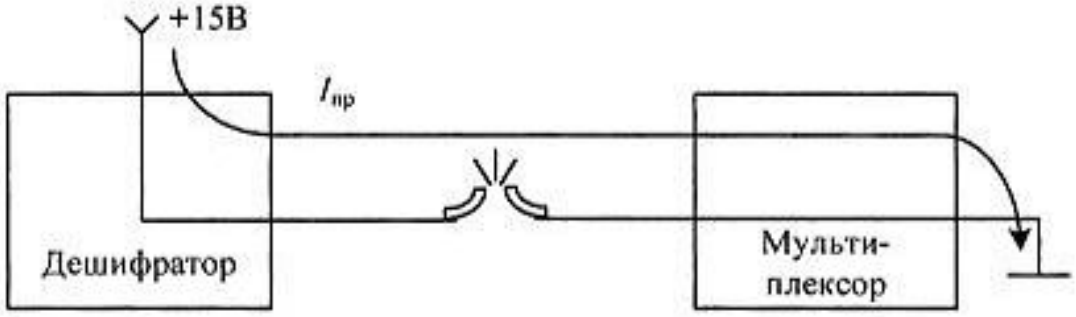


Рис. 15.18. Протекание тока через плавкую перемычку программируемого постоянного запоминающего устройства при записи нуля

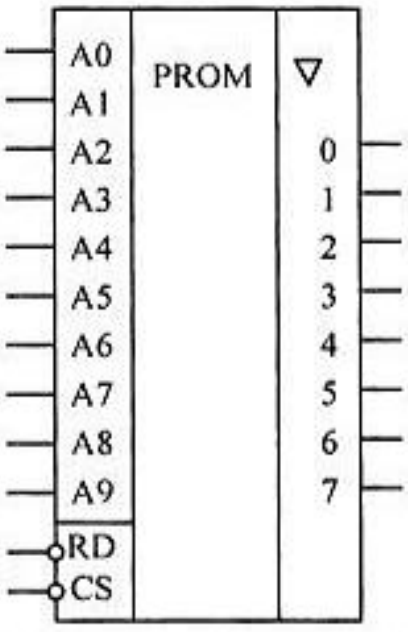


Рис. 15.19. Условно-графическое обозначение постоянного запоминающего устройства

является сокращением от английских слов programming read only memory (программируемая память, доступная только по чтению). В качестве примера таких ПЗУ можно назвать отечественные микросхемы 155РЕ3, 556РТ4, 556РТ8.

Программируемые ПЗУ оказались очень удобны при мелкосерийном и среднесерийном производстве. Однако при разработке радиоэлектронных устройств часто приходится менять записываемую в ПЗУ программу. ППЗУ при этом невозможно использовать повторно, поэтому раз записанное ППЗУ при ошибочной или промежуточной программе приходится выбрасывать, что естественно повышает стоимость разработки аппаратуры. Для устранения этого недостатка был разработан еще один вид ПЗУ, содержимое которого могло бы стираться и программироваться заново.

ПЗУ с ультрафиолетовым стиранием

ПЗУ с ультрафиолетовым стиранием строится на основе запоминающей матрицы, построенной на ячейках памяти, внутреннее устройство которой приведено на рис. 15.20.

Запоминающая ячейка представляет собой МОП-транзистор, в котором затвор выполняется из поликристаллического кремния. Затем в процессе изготовления микросхемы этот затвор окисляется и в результате он будет окружен оксидом кремния — диэлектриком с прекрасными изолирующими свойствами. Из-за того, что затвор со всех сторон окружен диэлектриком — он "плавает" внутри диэлектрика, этот затвор получил название плавающего затвора.

В описанной ячейке при полностью стертом ПЗУ заряда в плавающем затворе нет, и поэтому транзистор ток не проводит. При программировании микросхемы на программирующий электрод, находящийся над плавающим затвором, подается высокое напряжение и в последнем, за счет туннельного эффекта, индуцируются заряды. После снятия программирующего напряжения на плавающем затворе индуцированный заряд сохраняется и, следовательно, транзистор остается в проводящем состоянии. Заряд на плавающем затворе может храниться десятки лет.

Структурная схема постоянного запоминающего устройства с ультрафиолетовым стиранием не отличается от приведенной на рис. 15.16 схемы масочного ПЗУ. Единственным отличием является то, что вместо плавкой поликремниевой перемычки используется описанная выше ячейка. Такой вид ПЗУ в отечественной литературе получил название репрограммируемые ПЗУ.

Стирание ранее записанной информации в репрограммируемых ПЗУ осуществляется ультрафиолетовым излучением. Для того чтобы оно могло беспрепятственно воздействовать на полупроводниковый кристалл, в корпус микро-

схемы РПЗУ встраивается окошко из кварцевого стекла. При облучении микросхемы изолирующие свойства оксида кремния теряются, накопленный заряд из плавающего затвора стекает в объем полупроводника, и транзистор запоминающей ячейки переходит в закрытое состояние. Время стирания микросхемы колеблется в пределах 10—30 минут.

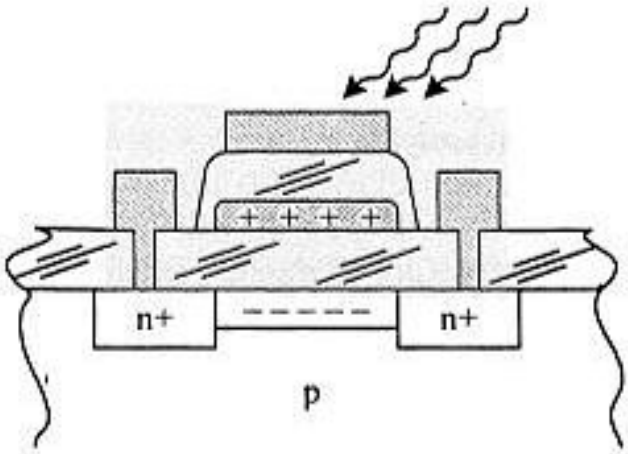


Рис. 15.20. Запоминающая ячейка ПЗУ с ультрафиолетовым и электрическим стиранием

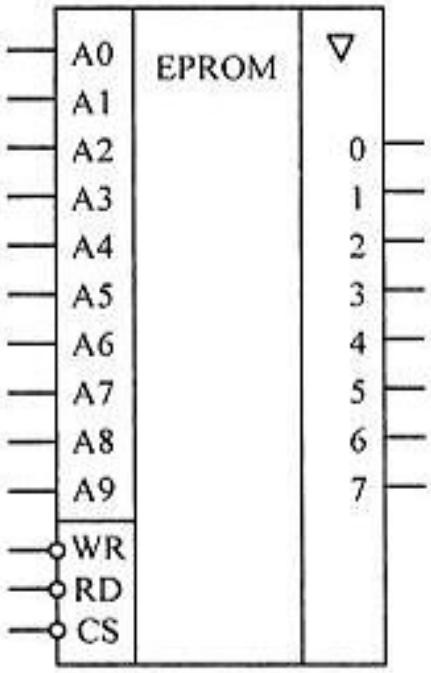


Рис. 15.21. Условно-графическое обозначение репрограммируемого постоянного запоминающего устройства

Количество циклов записи-стирания микросхем РПЗУ составляет от 10 до 100 раз, после чего вследствие разрушающего действия ультрафиолетового излучения микросхема выходит из строя. В качестве примера таких микросхем можно назвать микросхемы 573-й серии российского производства, микросхемы серий 27cXXX зарубежного производства. В этих микросхемах чаще всего хранятся программы для сигнальных процессоров или конфигурация соединений программируемых логических схем (ПЛИС). Репрограммируемые ПЗУ изображаются на схемах в виде условного графического обозначения, показанного на рис. 15.21. Надпись "EPROM" в центральной части микросхемы является сокращением от английских слов erasable programmable read-only memory (стираемая программируемая память, доступная только для чтения).

ПЗУ с электрическим стиранием информации

То, что корпуса микросхем с кварцевым окошком очень дороги, а также малое количество циклов записи-стирания привели к поиску способов стирания

информации из ППЗУ электрическим способом. На этом пути встретилось много трудностей, которые к настоящему времени практически решены.

В настоящее время достаточно широко распространены микросхемы с электрическим стиранием информации. В качестве запоминающей ячейки в них используются такие же ячейки, как и в РПЗУ, но они стираются электрическим потенциалом, поэтому количество циклов записи-стирания для этих микросхем достигает 1 000 000 раз. Время стирания ячейки памяти в таких микросхемах уменьшается до 10 мс.

Схема управления для микросхем с электрическим стиранием получилась сложная, поэтому наметилось два направления развития этих микросхем:

1. Электрически стираемые ПЗУ (ЭСППЗУ).
2. FLASH-ПЗУ.

Из-за сложности внутренней схемы управления запоминающими элементами электрически стираемые ПЗУ дороже и меньше по объему, но зато позволяют перезаписывать каждую ячейку памяти отдельно. В результате эти микросхемы обладают максимальным количеством циклов записи-стирания. Область применения электрически стираемых ПЗУ — хранение данных, которые не должны разрушаться при выключении питания. К таким микросхемам относятся отечественные микросхемы 573РР3 и зарубежные микросхемы серии 28cXX. Электрически стираемые ПЗУ изображаются на схемах при помощи условного графического обозначения, показанного на рис. 15.22. Надпись "EEPROM" в среднем поле расшифровывается как electrically erasable programmable read-only memory — электрически стираемая программируемая память, доступная только для чтения.

В последнее время наметилась тенденция уменьшения габаритов ЭСРПЗУ за счет сокращения количества внешних выводов микросхем. Для этого адрес и данные передаются в микросхему и из нее в виде последовательного кода. При этом используются два вида последовательных интерфейсов: SPI и I^2C (микросхемы серий 93cXX и 24cXX соответственно). Зарубежной серии 24cXX соответствует отечественная серия микросхем 558РРх.

FLASH-ПЗУ отличаются от ЭСППЗУ тем, что стирание производится не каждой ячейки отдельно, а всей микросхемы в целом или блока запоминающей матрицы этой микросхемы, как это делалось в РПЗУ. Изображение FLASH-ПЗУ на принципиальных схемах приведено на рис. 15.23. Использование блочного стирания микросхем позволяет уменьшить сложность их внутренней схемы, поэтому микросхемы FLASH предоставляют максимальный объем внутренней памяти. Кроме того, эти микросхемы обладают значительно меньшей стоимостью по сравнению с электрически стираемыми ПЗУ. В настоящее время FLASH-ПЗУ постепенно вытесняют все остальные виды ПЗУ за исключением электрически стираемых ПЗУ.

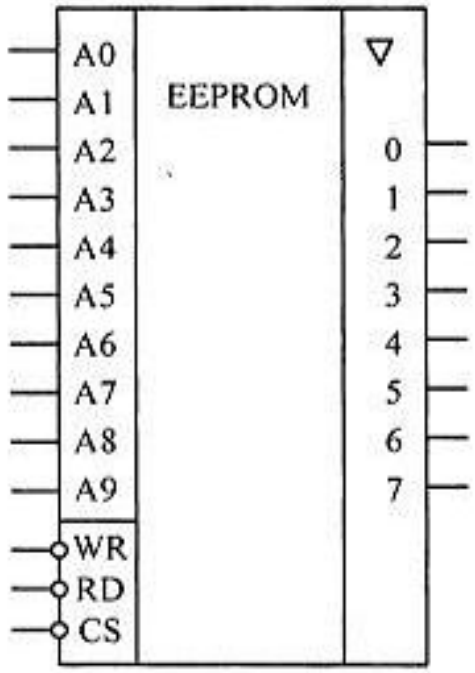


Рис. 15.22. Условно-графическое обозначение электрически стираемого постоянного запоминающего устройства

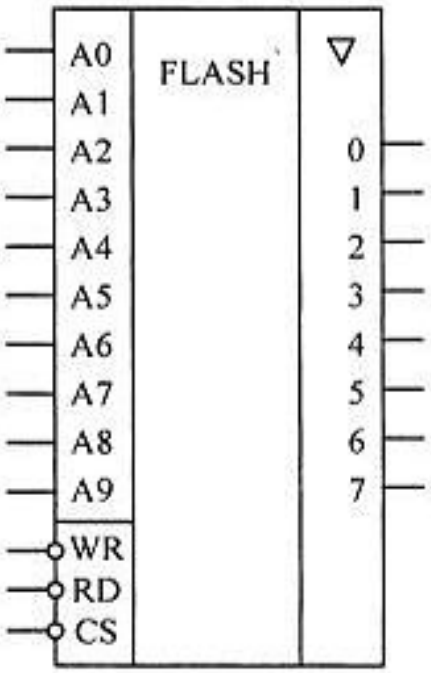


Рис. 15.23. Обозначение FLASH-памяти на принципиальных схемах

Статические оперативные запоминающие устройства (ОЗУ)

В устройствах цифровой обработки сигналов для хранения отсчетов входного или выходного сигналов широко используются параллельные регистры, однако в ряде случаев требуется осуществлять обработку информации в последовательном виде. В этом случае информация с выходов регистров не требуется одновременно, поэтому входные отсчеты можно хранить в устройствах памяти, подобных по структуре постоянным запоминающим устройствам.

Устройства памяти, в которых в качестве запоминающих ячеек используются параллельные регистры, называются статическими ОЗУ, т. к. информация в них сохраняется все время, пока к микросхеме подключено питание.

Так как в устройствах последовательной обработки отсчеты цифрового сигнала не нужны одновременно, то в ОЗУ можно воспользоваться механизмом адресации, который уже рассматривался ранее при объяснении принципов работы ПЗУ.

В микросхемах статических ОЗУ присутствуют две операции: запись и чтение. Для их выполнения можно использовать различные шины данных (как это делается в сигнальных процессорах), но чаще используется одна и та же шина. Это позволяет экономить выводы микросхем, подключаемых к этой

шине, и легко осуществлять коммутацию сигналов между различными устройствами.

Схема статического ОЗУ приведена на рис. 15.24. Вход и выход микросхемы в этой схеме объединены при помощи шинного формирователя. Естественно, что схемы реальных ОЗУ будут иными, чем приведенная на этом рисунке. Тем не менее она позволяет понять, как работает реальное ОЗУ статического типа. Условное графическое обозначение ОЗУ на схемах приведено на рис. 15.25.

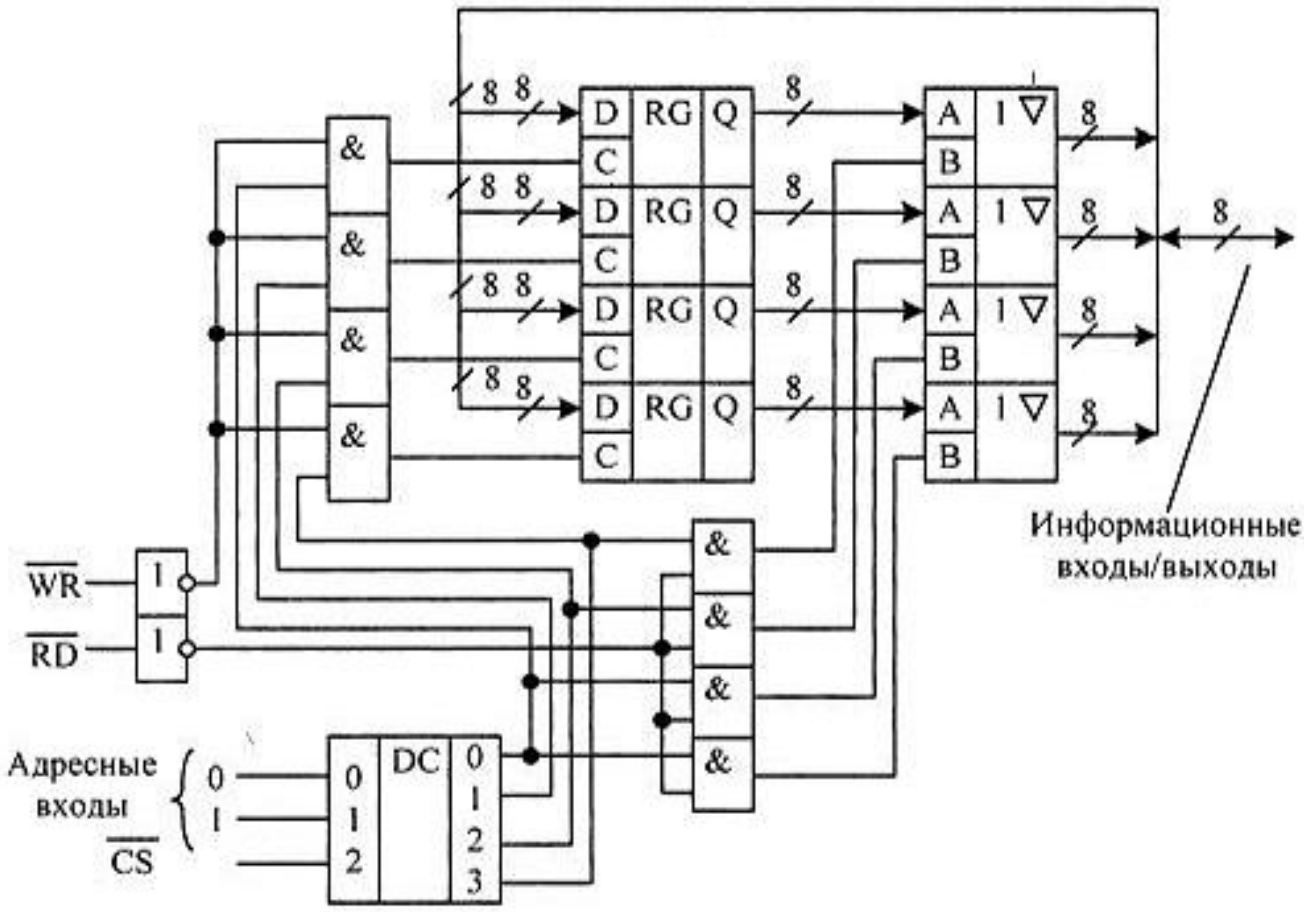


Рис. 15.24. Структурная схема ОЗУ

На схеме рис. 15.25 для обозначения того, что используется инвертированный сигнал или сигнал с активным низким уровнем, над именем цепи проставляется черта. К сожалению, в обычном тексте затруднительно использовать такую же черту. Поэтому для обозначения таких сигналов используется два способа: символ подчеркивания перед именем цепи (_WR) или символ '#' после имени (WR#).

Сигнал записи WR# позволяет записать логические уровни, присутствующие на информационных входах, во внутреннюю ячейку ОЗУ. Сигнал чтения RD# позволяет выдать содержимое внутренней ячейки памяти на информацион-

ные выходы микросхемы. В приведенной на рис. 15.25 схеме невозможно одновременно производить операцию записи и чтения, но это в большинстве случаев и не нужно.

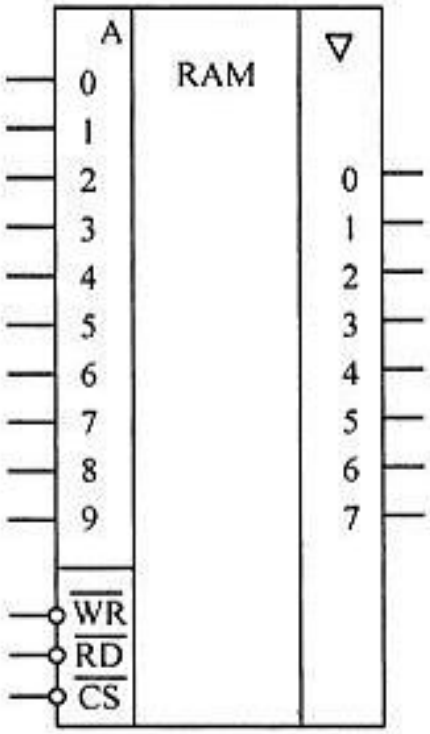


Рис. 15.25. Условное графическое обозначение ОЗУ

Конкретная ячейка микросхемы, в которую будет записываться информация, выбирается при помощи двоичного кода — адреса ячейки. Объем памяти микросхемы зависит от количества ячеек, содержащихся в ней. Количество адресных выводов микросхемы ОЗУ однозначно определяется количеством находящихся в ней ячеек памяти. Исходя из этого, количество ячеек памяти M в микросхеме можно определить по количеству адресных выводов N. Для этого необходимо возвести число 2 в степень, равную количеству адресных выводов микросхемы:

$$M = 2^N.$$

Вывод выбора кристалла CS позволяет объединять несколько микросхем для увеличения объема памяти ОЗУ. Пример объединения четырех микросхем ОЗУ с помощью дополнительного дешифратора адреса приведен на рис. 15.26. При этом общий объем памяти запоминающего устройства увеличивается в четыре раза.

Временные диаграммы чтения данных из статического ОЗУ совпадают с временными диаграммами для рассмотренного ранее ПЗУ. Временные диаграммы записи в статическое ОЗУ и чтения из него приведены на рис. 15.27.

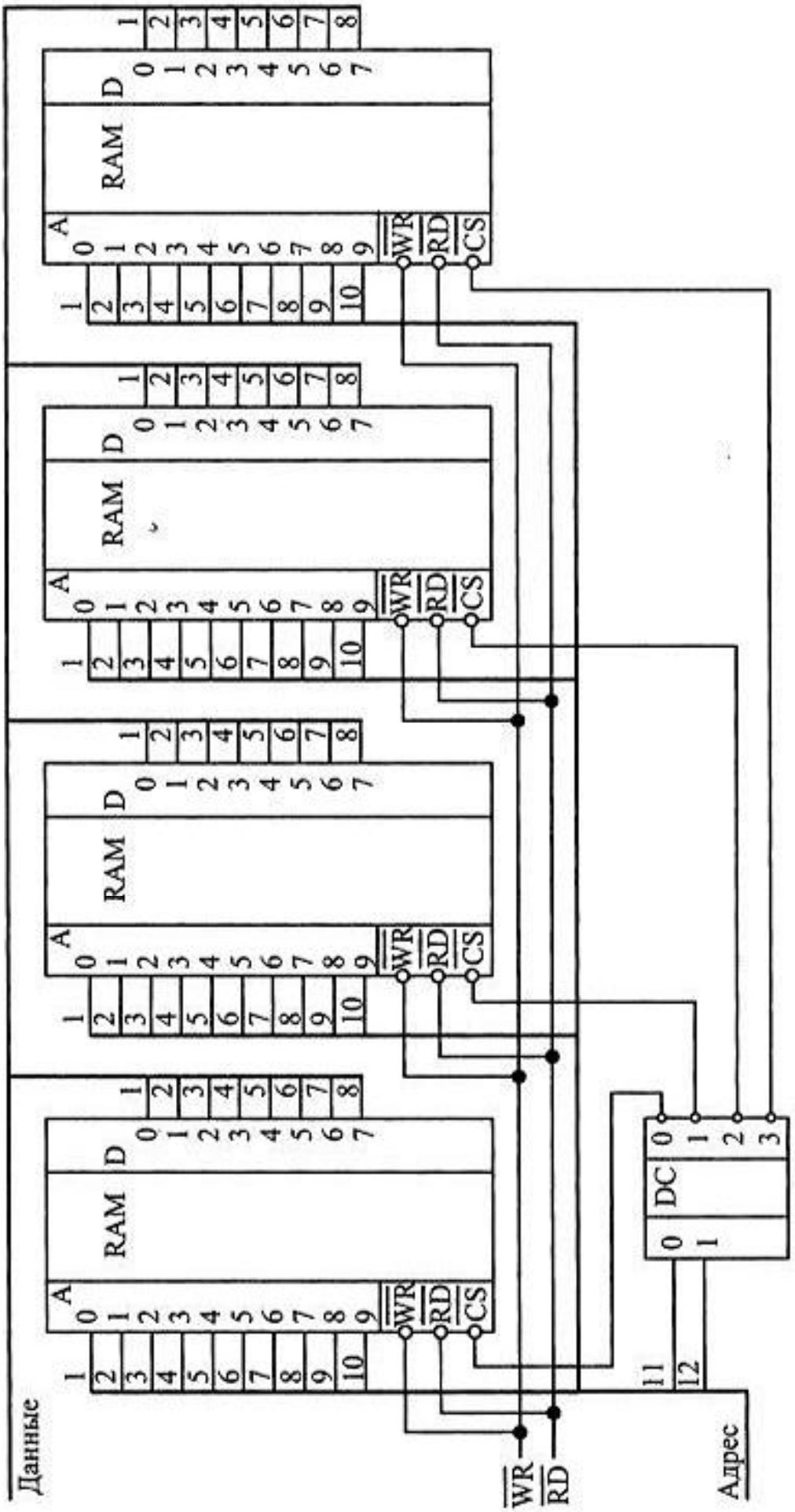


Рис. 15.26. Схема ОЗУ, построенного на нескольких микросхемах памяти

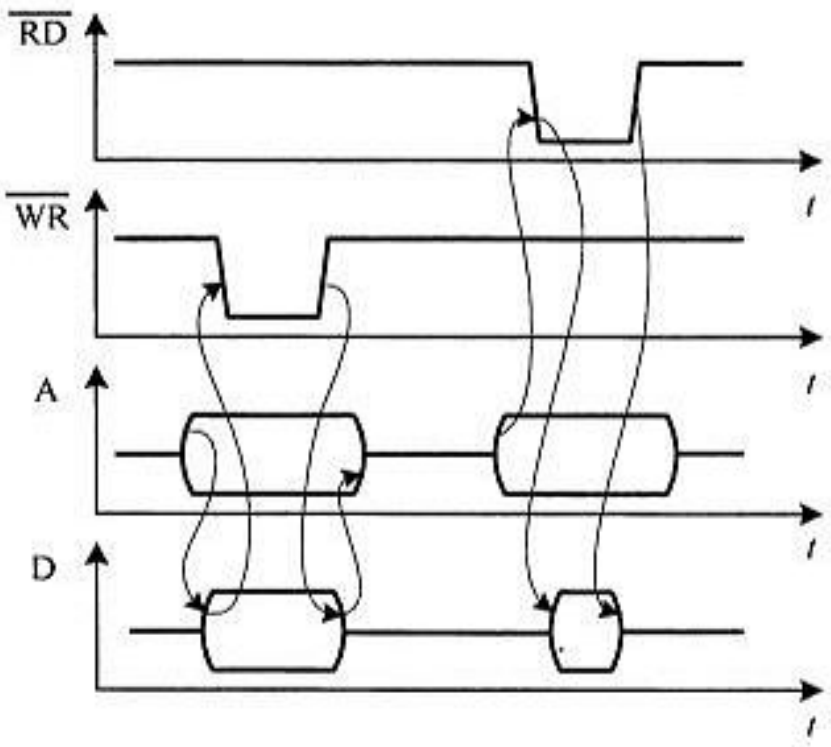


Рис. 15.27. Временная диаграмма обращения к ОЗУ

На рис. 15.27 стрелочками показана последовательность, в которой должны формироваться управляющие сигналы. На этом рисунке RD — это инвертированный сигнал чтения; WR — инвертированный сигнал записи; A — сигналы шины адреса (т. к. отдельные биты в шине адреса могут принимать разные значения, то показаны пути перехода двоичного сигнала как в единичное, так и в нулевое состояние); DI — входная информация, предназначенная для записи в ячейку ОЗУ, расположенную по адресу A1; DO — выходная информация, считанная из ячейки ОЗУ, расположенной по адресу A2.

Цифровые фильтры

Следующим, широко используемым блоком в схемах цифровой обработки сигналов являются цифровые фильтры. Прежде чем начать подробное обсуждение вопросов реализации цифровых фильтров, давайте вспомним — что же такое частотные фильтры? Частотные фильтры требуются для подавления мешающих сигналов, отличающихся по частоте от полезного. В частотной области зависимость коэффициента передачи фильтра можно изобразить так, как приведено на рис. 15.28.

На этом рисунке приведена частотная характеристика фильтра, выделяющего нужную нам полосу частот. Однако мы знаем, что операция фильтрации (выделения полезной части данных) в частотной области эквивалентна операции вычисления свертки во временной области и наоборот:

$$S_{вых}(f) = \int_{-\infty}^{\infty} G(f) \times S_{вх}(f) \times df \Leftrightarrow s_{вых}(t) = \int_{-L}^{0} s_{вх}(t-\tau) \times g(\tau) \times d\tau, \qquad (15.1)$$

где $S(f)$ — спектр сигнала;

$s(t)$ — временная реализация сигнала;

$G(f)$ — частотная характеристика полосового фильтра;

$g(\tau)$ — импульсная характеристика полосового фильтра;

L — длина импульсной характеристики полосового фильтра.

Таким образом, для реализации фильтра нам достаточно определить форму импульсной характеристики фильтра и вычислить операцию свертки. Импульсная характеристика связана с частотной характеристикой преобразованием Фурье. Поэтому в простейшем случае рассчитать цифровой фильтр можно, используя преобразование Фурье от требуемой частотной характеристики.

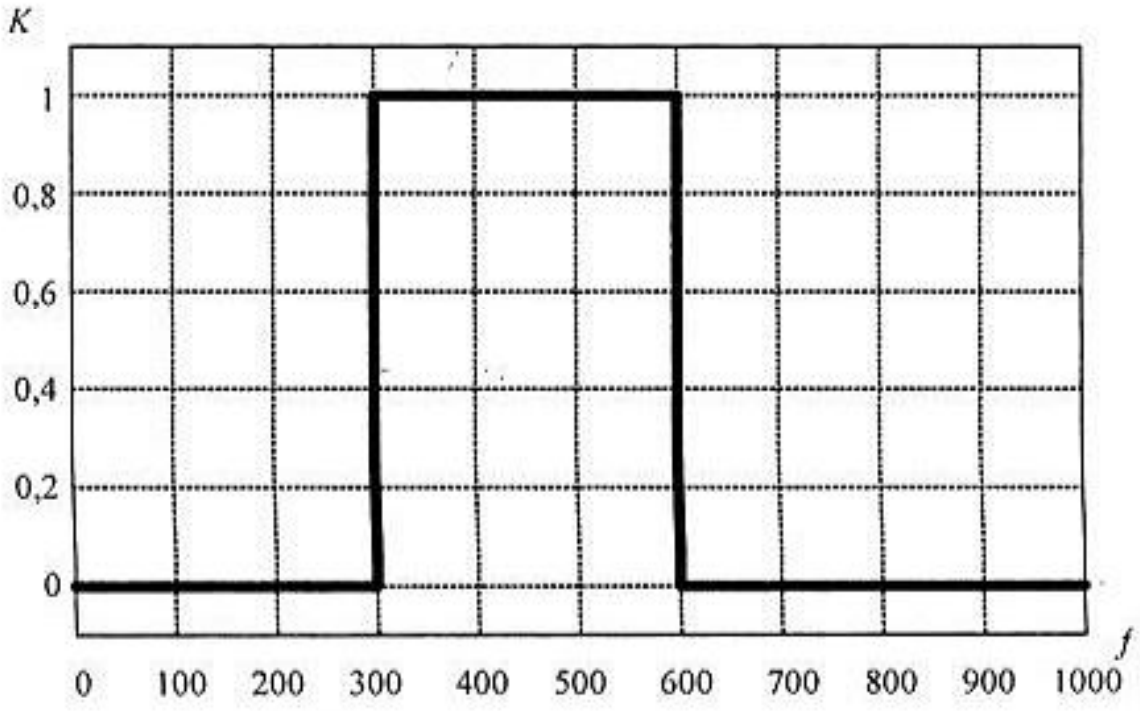


Рис. 15.28. Амплитудно-частотная характеристика коэффициента передачи фильтра

Кроме знания формы импульсной характеристики для создания цифрового фильтра нам требуется уметь запоминать значения входного сигнала в предыдущие моменты времени. Для этого могут быть использованы параллельные регистры, внутреннее устройство которых рассмотрено в предыдущих главах. В формуле вычисления свертки используется арифметическая опера-

ция умножения. Устройство, способное выполнять эту операцию, мы уже рассматривали в предыдущих главах.

Остается операция интегрирования. Однако при использовании целочисленных значений входного сигнала ее можно представить как сумму всех отсчетов этого сигнала, а внутреннее устройство арифметического сумматора мы уже знаем. При замене операции интегрирования суммой формула (15.1) примет следующий вид:

$$s_{\text{вых}}(n) = \sum_{i=1}^{N} s_{\text{вх}}(n-i) \times g(i) \qquad (15.2)$$

Рассмотрим структурную схему устройства, способного вычислять операцию свертки (цифровой фильтр). Структурная схема цифрового фильтра приведена на рис. 15.29.

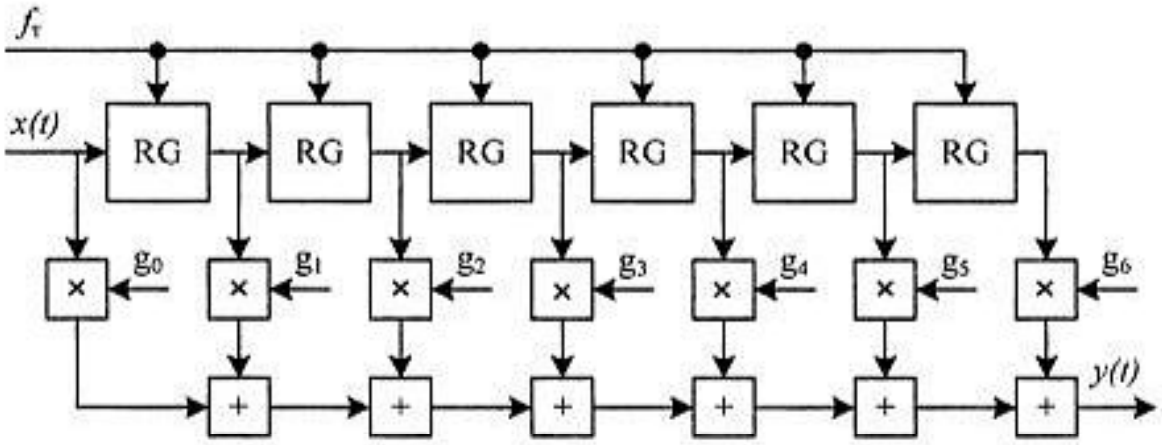


Рис. 15.29. Структурная схема устройства, способного вычислять операцию свертки

В качестве элементов задержки в этой схеме использованы схемы параллельных регистров. Разрядность этих регистров определяет динамический диапазон входного сигнала. Если мы собираемся работать с динамическим диапазоном сигнала равным 96 дБ, то разрядность параллельных регистров должна составлять $N = 96\text{дБ} = 16$ разрядов.

Точно такой же должна быть разрядность весовых коэффициентов фильтра. Это означает, что в составе схемы цифрового фильтра необходимо использовать умножители 16×16. Разрядность произведения при этом будет равна 32. Суммирование сигналов со всех отводов линии задержки, реализованной на параллельных регистрах, производится двухвходовыми сумматорами. Их разрядность должна быть, по крайней мере, на восемь разрядов больше разрядности произведения содержимого регистров на весовые коэффициенты. То есть разрядность сумматоров должна быть равной 40 разрядам.

Давайте проверим, как будет реагировать приведенное на рис. 15.29 устройство на одиночный импульс единичной амплитуды, поданный на его вход. Для наглядности рассуждений возьмем импульсную характеристику одиночного колебательного контура. Эта характеристика приведена на рис. 15.30.

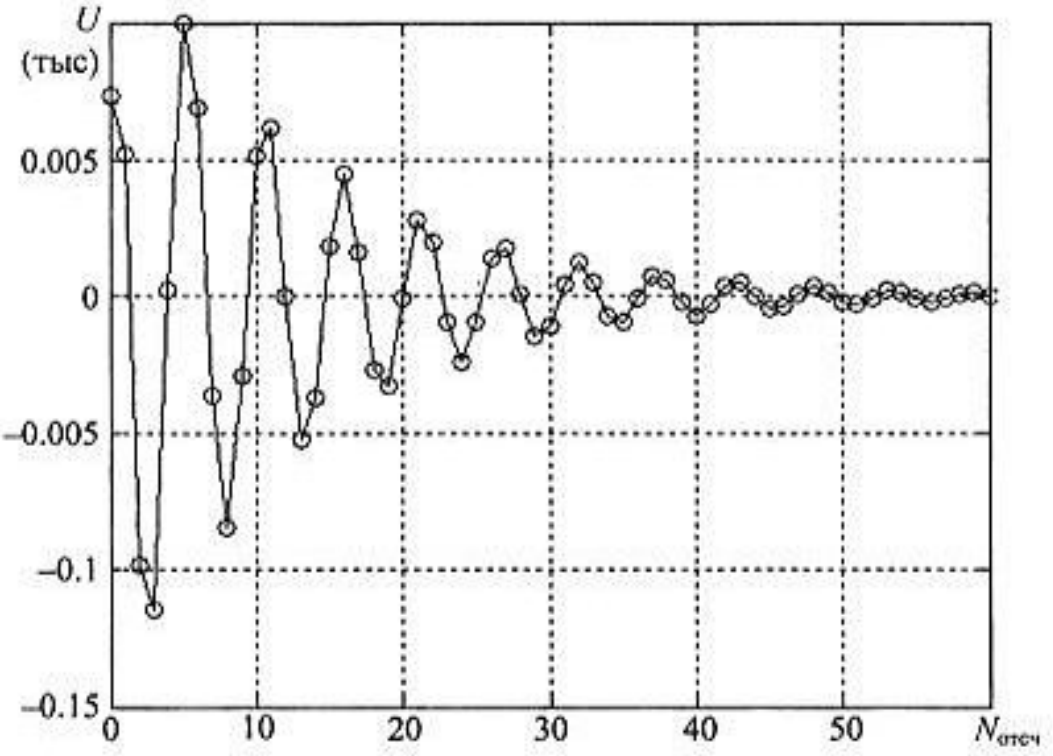


Рис. 15.30. Форма импульсной характеристики колебательного контура

На рис. 15.30 кружочками обозначены значения весовых коэффициентов импульсной характеристики цифрового фильтра. Именно эти коэффициенты подаются на входы умножителей в структурной схеме цифрового фильтра, приведенной на рис. 15.29. Для наглядности эти коэффициенты на рисунке соединены между собой прямыми линиями (так характеристика становится более похожей на импульсную характеристику аналогового колебательного контура).

Частотную характеристику фильтра, обладающего импульсной характеристикой, соответствующей рис. 15.30, можно получить, проведя преобразование Фурье над этой импульсной характеристикой. Полученная в результате этого преобразования амплитудно-частотная характеристика фильтра приведена на рис. 15.31. По оси абсцисс на этом рисунке отложена частота в килогерцах, а по оси ординат — коэффициент передачи цифрового фильтра в децибелах.

Теперь подадим на вход схемы, приведенной на рис. 15.29, цифровой код, соответствующий единичному уровню сигнала. В первый момент времени во всех внутренних регистрах фильтра содержатся нулевые значения. Это озна-

чает, что при умножении этих значений на весовые коэффициенты мы получим в результате нули. Отличаться будет только результат на выходе первого умножителя. При перемножении весового коэффициента $g0$ на единичное значение входного сигнала мы получим на выходе умножителя значение сигнала с амплитудой $g0$.

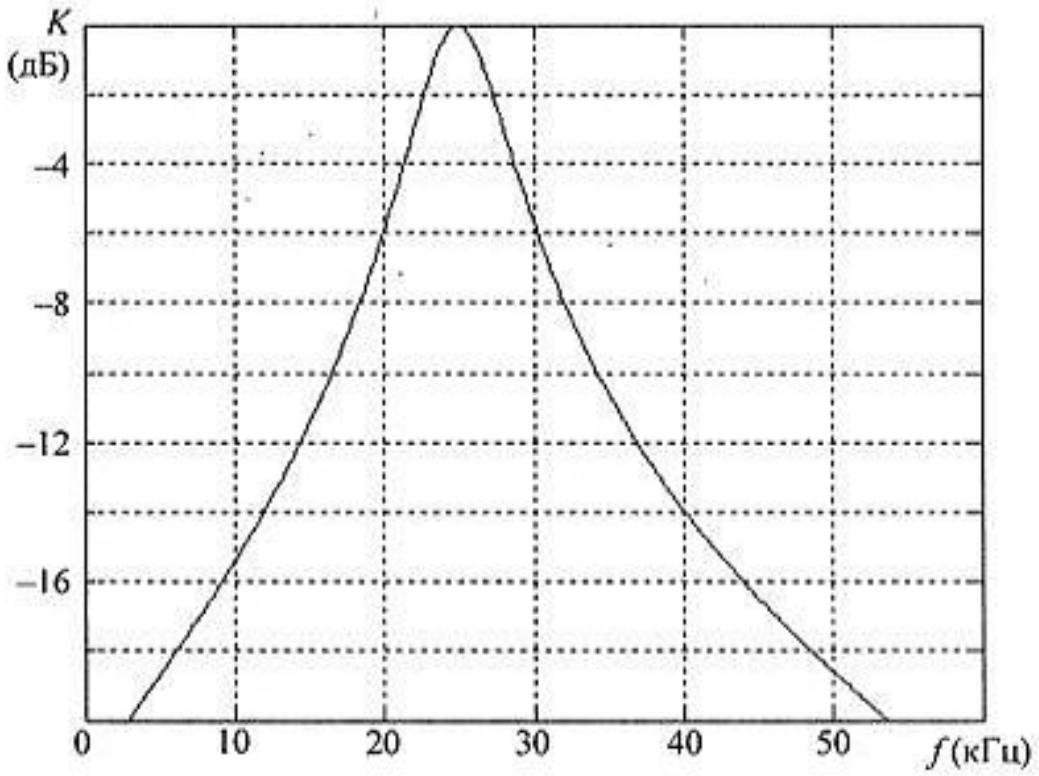


Рис. 15.31. Форма частотной характеристики фильтра

Как мы уже знаем из предыдущих глав, задержка в схеме определяется частотой тактового сигнала, подаваемого на входы синхронизации параллельных регистров. При поступлении первого тактового импульса код, присутствовавший на входе схемы, запишется в первый регистр (элемент задержки). По этому же тактовому сигналу содержимое первого регистра (нулевое значение) перепишется во второй регистр, содержимое второго регистра перепишется в третий регистр и т. д.

Как мы договорились, теперь на вход фильтра мы подадим код, соответствующий нулевому значению сигнала. В результате на выходе всех умножителей, кроме второго, снова будет присутствовать нулевой код. Так как в первом регистре на этот раз содержится единичное значение, то в результате умножения на коэффициент $g1$, на выходе второго умножителя мы получим значение сигнала с амплитудой $g1$.

При поступлении последующих тактовых импульсов процесс будет повторяться, и мы на выходе схемы последовательно будем получать значения сигналов, соответствующие весовым коэффициентам.

Итак, мы убедились, что схема ведет себя подобно обычному фильтру, и научились рассчитывать весовые коэффициенты этой схемы, требуемые для получения заданных характеристик фильтров. Собственно говоря, было бы удивительно не получить фильтр, ведь обычные аналоговые схемы фактически работают так же. Энергия на колебательном контуре постепенно накапливается за счет суммирования текущего входного напряжения и всех предыдущих значений, активный RC-фильтр с обратными связями ведет себя точно так же.

Преимуществом рассмотренной схемы является то, что в ней мы можем выбирать коэффициенты импульсной характеристики произвольным образом без ограничений, которые обычно существуют в других видах фильтров. В результате этой особенности мы можем получить исключительные свойства, нереализуемые в других схемах.

Например, мы можем получить строго симметричную импульсную характеристику фильтра, подав на умножители коэффициенты, соответствующие этой импульсной характеристике. Таким образом, как это известно из теории линейных цепей, можно реализовать фильтр со строго линейной фазовой характеристикой, или, что то же самое, одинаковое время задержки фильтра во всей полосе рабочих частот. Это свойство чрезвычайно полезно для аппаратуры передачи данных или обработки телевизионных сигналов.

В качестве еще одного примера использования рассмотренной ранее схемы можно назвать реализацию фильтров Найквиста. Как известно, в этих фильтрах импульсная характеристика должна принимать нулевые значения строго через определенные интервалы, равные длительности передаваемого символа. И такая возможность нам доступна — кто же может запретить нам записать в нужные ячейки памяти коэффициентов нулевой двоичный код?

Снова зададимся вопросом: как же рассчитывать цифровые фильтры? Простейший метод расчета этих фильтров мы рассмотрели выше по тексту. Это выполнение дискретного преобразования Фурье над заданной амплитудно-частотной характеристикой. Однако при решении инженерных задач осуществлять каждый раз преобразование Фурье достаточно трудоемкая задача.

Кроме того, для получения приемлемых результатов полученную таким образом импульсную характеристику фильтра нужно обрабатывать временными окнами. При этом в каждом конкретном случае оптимальное с точки зрения получения заданной амплитудно-частотной характеристики решение может различаться. Более того, кроме метода преобразования Фурье существуют другие методы расчета цифровых фильтров. Подчас эти методы приводят к более эффективным решениям.

Для решения инженерных задач имеет смысл воспользоваться специализированными программами расчета цифровых фильтров. В качестве примера по-

добных программ можно привести такие программы, как FGEN или FILTER DESIGN, входящую в состав программной среды MATHLAB 6.12. Внешний вид рабочего окна программы FILTER DESIGN приведен на рис. 15.32.

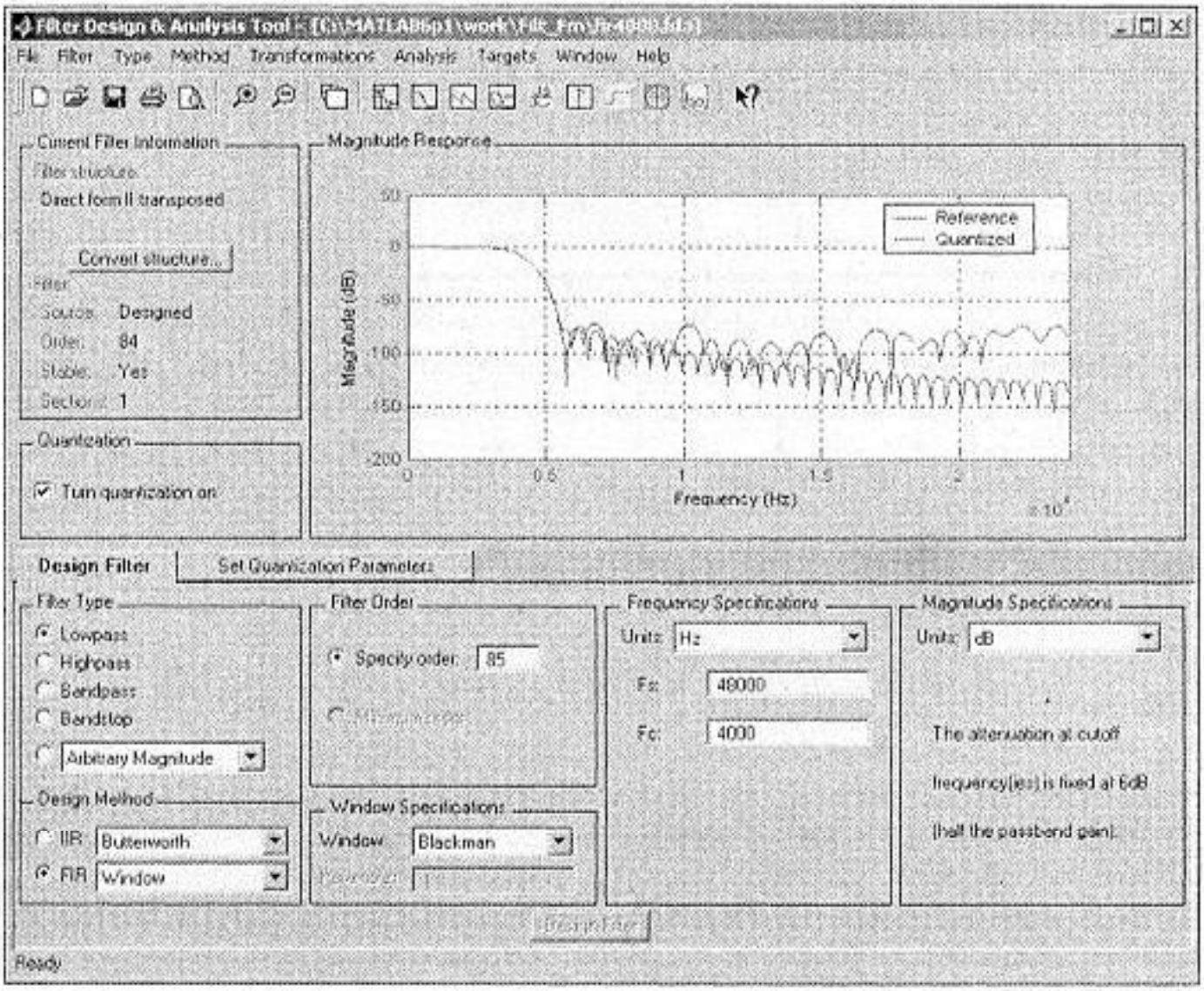


Рис. 15.32. Внешний вид программы FILTER DESIGN

Схемная реализация нерекурсивного фильтра

Теперь снова вернемся к схемотехнической реализации цифровых фильтров. В схеме, приведенной на рис. 15.25, применяется столько умножителей, сколько используется коэффициентов для реализации цифрового фильтра.

Обычно цифровые фильтры с произвольной импульсной характеристикой реализуются на частотах в несколько раз меньших предельной частоты работы цифровых микросхем. В этом случае вычисление результатов произведения всех весовых коэффициентов на задержанные значения входного сигнала

для формирования одного выходного отсчета цифрового фильтра можно выполнить на одиночном цифровом умножителе и сумматоре.

Чтобы понять, как реализовать предложенный принцип вычислений, давайте рассмотрим подробнее структурную схему цифрового фильтра с конечной импульсной характеристикой. На рис. 15.33 пунктиром показана повторяющаяся часть структурной схемы фильтра. А раз она повторяется, то эту часть схемы при вычислении выходного отсчета сигнала можно использовать многократно.

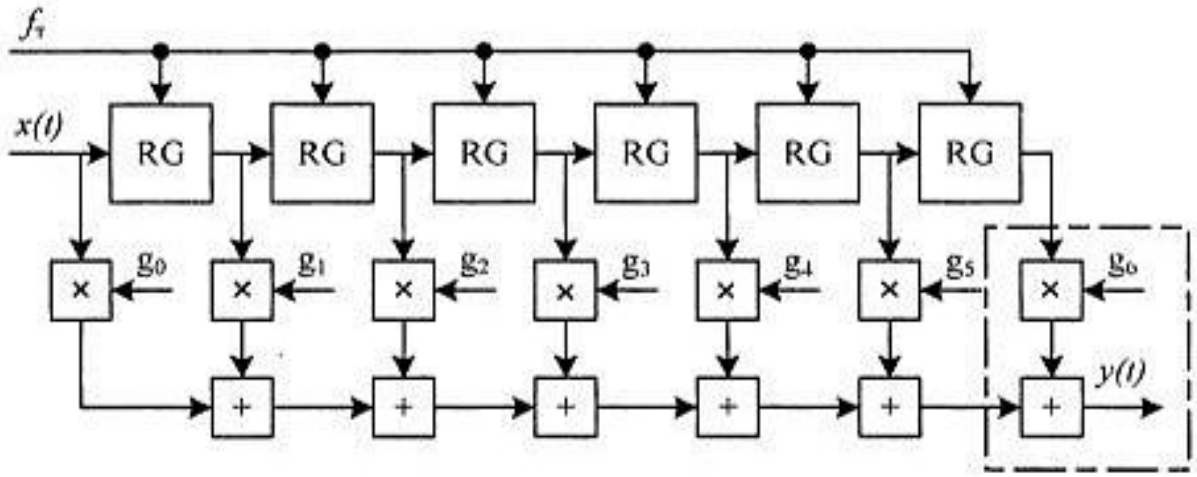


Рис. 15.33. Структурная схема цифрового фильтра седьмого порядка

Для запоминания промежуточных результатов суммирования нам потребуется дополнительный регистр. Так как в этом регистре результаты умножения отсчетов входного сигнала на весовые коэффициенты фильтра накапливаются, то этот регистр получил название аккумулятор (накопитель). Вся же схема, осуществляющая математическую обработку сигнала при реализации цифрового фильтра, получила название умножителя-аккумулятора (multiplier/accumulator — MAC).

Давайте перерисуем структурную схему цифрового фильтра с конечной импульсной характеристикой, приведенную на рис. 15.33, так, как это показано на рис. 15.34.

Так как в качестве примера мы выбрали фильтр с конечной импульсной характеристикой седьмого порядка, то для синхронизации этой схемы потребуется частота, в семь раз большая частоты дискретизации входного (и выходного) сигнала.

Данные на входы умножителя-аккумулятора поступают с выходов кольцевых многоразрядных регистров. После поступления на вход синхронизации схемы семи тактовых импульсов данные совершают полный круг и возвращаются на свое первоначальное место. В этот момент в аккумуляторе содержится вычисленное значение отсчета выходного сигнала фильтра.

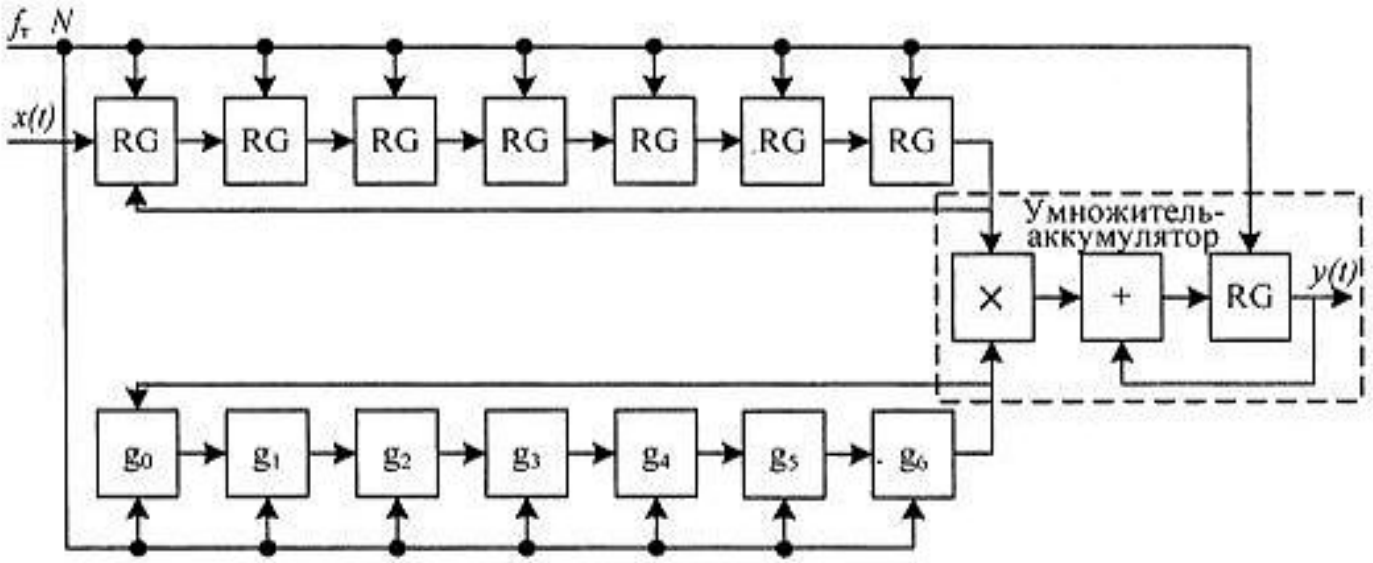


Рис. 15.34. Структурная схема цифрового фильтра седьмого порядка

В следующий момент времени следует обнулить значение аккумулятора, подключить вход сдвигового регистра к входу $x(t)$ и подать еще один тактовый импульс. Этот импульс позволит записать новое значение входного сигнала в первый параллельный регистр кольцевого многоразрядного регистра и осуществить перезапись содержимого всех остальных регистров в соседнюю ячейку (сдвинуть данные на одну позицию).

Цепи, осуществляющие эти действия в схеме, приведенной на рис. 15.34, не показаны для того, чтобы не затенять основную схему. Теперь обратим внимание, что на этот раз (в отличие от схемы, приведенной на рис. 15.33) нам не нужны все выходы параллельных регистров одновременно, поэтому для реализации кольцевого регистра можно воспользоваться оперативным запоминающим устройством (ОЗУ).

Весовые коэффициенты фильтра при этом будут находиться в ПЗУ. Схема цифрового нерекурсивного фильтра с использованием в качестве кольцевых регистров ОЗУ и ПЗУ приведена на рис. 15.35.

В данной схеме кольцевой регистр образуется ОЗУ и счетчиком. При поступлении на счетный вход счетчика тактовых импульсов на входы умножителя-накопителя поступают задержанные отсчеты данных и соответствующие им весовые коэффициенты фильтра.

Коэффициенты счета счетчиков Сч1 и Сч2 различаются. Коэффициент счета счетчика Сч2 равен N+1, т. е. равен количеству тактовых импульсов, поступающих за один период дискретизации обрабатываемого сигнала. Коэффициент счета счетчика Сч1 равен N. В результате адрес ОЗУ после вычисления очередного выходного значения фильтра сдвигается на одну позицию. Так обеспечивается сдвиг данных в линии задержки фильтра.

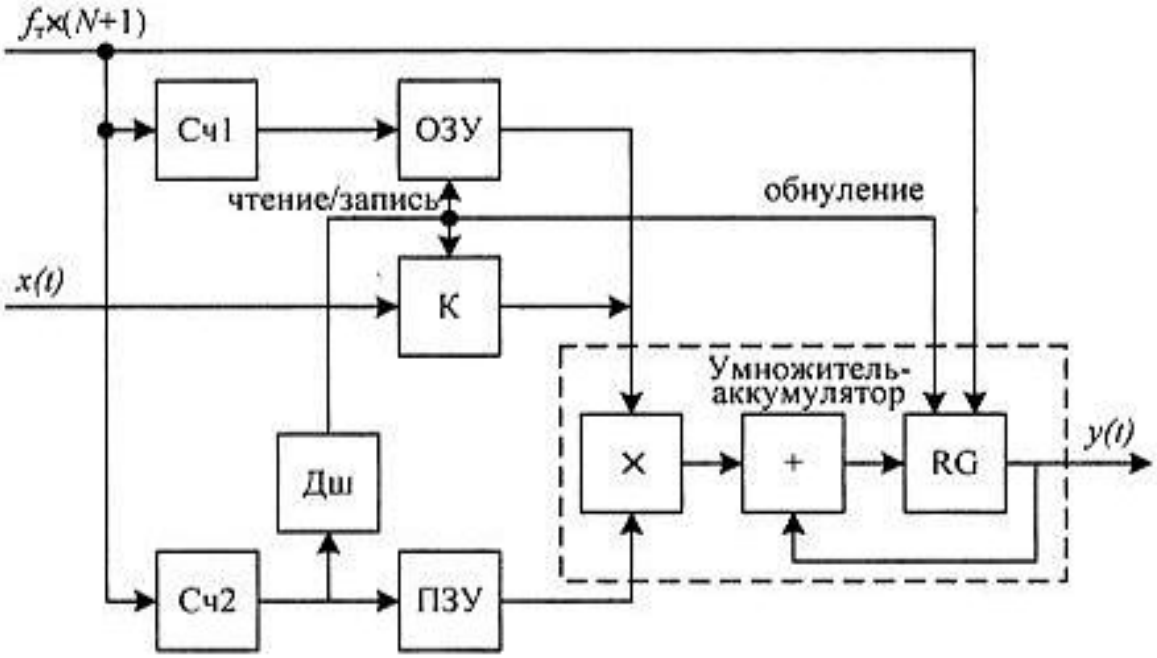


Рис. 15.35. Структурная схема цифрового фильтра

После поступления на тактовый вход фильтра $N+1$ импульсов счетчик Сч1 указывает на самое старое значение данных, записанное в ОЗУ. В этот момент ОЗУ переводится в режим записи, открывается ключ и обнуляется регистр-аккумулятор сигналом, вырабатываемым с выхода дешифратора, подключенного к счетчику Сч2. В результате старое значение данных заменяется на новый отсчет входного сигнала.

Итак, приведенная на рис. 15.35 схема выполняет те же самые действия, что и прямая реализация фильтра с конечной импульсной характеристикой, но при этом гораздо проще. Кроме того, эта схема очень легко перенастраивается на фильтры с различным числом коэффициентов. Для этого достаточно просто изменить коэффициент счета цифровых счетчиков Сч1 и Сч2.

Однородный цифровой фильтр

В предыдущей главе мы научились осуществлять фильтрацию в цифровом виде, однако часто цифровую фильтрацию требуется осуществить на очень высоких частотах дискретизации полезного сигнала. Для увеличения быстродействия желательно уменьшить время выполнения операций каждым из блоков, входящих в состав цифрового фильтра. Наиболее сложным внутренним устройством обладают цифровые умножители, т. е. быстродействие фильтра в целом в основном определяется именно этим блоком. Было бы неплохо отказаться от использования умножителей в составе цифрового фильтра.

На очень высоких частотах для подавления мешающих сигналов обычно используется так называемый однородный фильтр [2], [15]. Для реализации

этого фильтра не требуются умножители. Однородный фильтр представляет собой фильтр с конечной импульсной характеристикой, обладающий прямоугольной импульсной характеристикой (т. е. представляет собой обычный усреднитель).

Передаточная характеристика однородного фильтра описывается формулой:

$$H(z)=\sum_{n=0}^{N-1} z^{-1} \qquad (15.3)$$

Для однородного фильтра седьмого порядка эта формула выглядит следующим образом:

$$y(n)=x(n)+x(n-1)+x(n-2)+x(n-3)+x(n-4)+x(n-5)+x(n-6) \qquad (15.4)$$

Структурная схема фильтра, реализующего формулу 15.3, приведена на рис. 15.36.

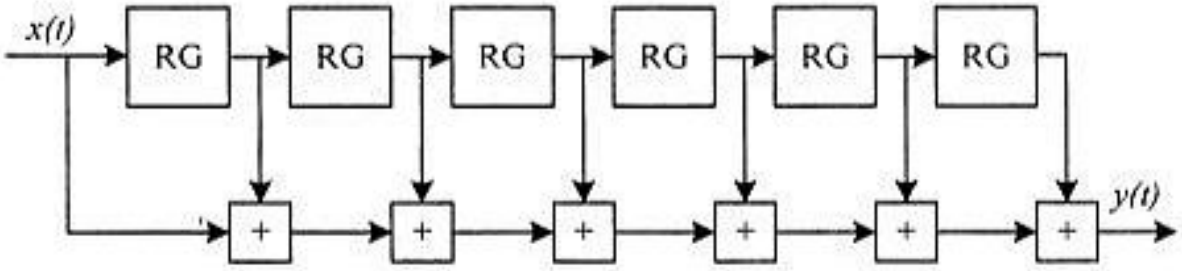


Рис. 15.36. Структурная схема однородного фильтра седьмого порядка

Этот фильтр реализует математическую операцию скользящего среднего. Применив к импульсной характеристике фильтра (15.4) дискретное преобразование Фурье, получим выражение, описывающее его амплитудно-частотную характеристику.

$$|H(f)|=\left|\frac{\sin(2\pi f)}{2\pi f}\right| \qquad (15.5)$$

Это хорошо известная функция $\sin(x)/x$. График амплитудно-частотной характеристики однородного фильтра приведен на рис. 15.37. На этом рисунке по оси ординат отложен коэффициент передачи фильтра в децибелах, по оси абсцисс — частота. Как и ожидалось, однородный фильтр является фильтром низкой частоты. Так как импульсная характеристика данного фильтра симметрична, то его фазовая характеристика будет строго линейна. Это означает, что однородный фильтр не вносит фазовых искажений в исходный цифровой сигнал. Однако глубина подавления частот вне полосы пропускания у рассмотренного фильтра для большинства приложений явно недостаточна. Известно, что при включении нескольких фильтров друг за другом глубина по-

давления мешающих сигналов возрастает, т. к. при последовательном включении четырехполюсников их коэффициенты передачи перемножаются.

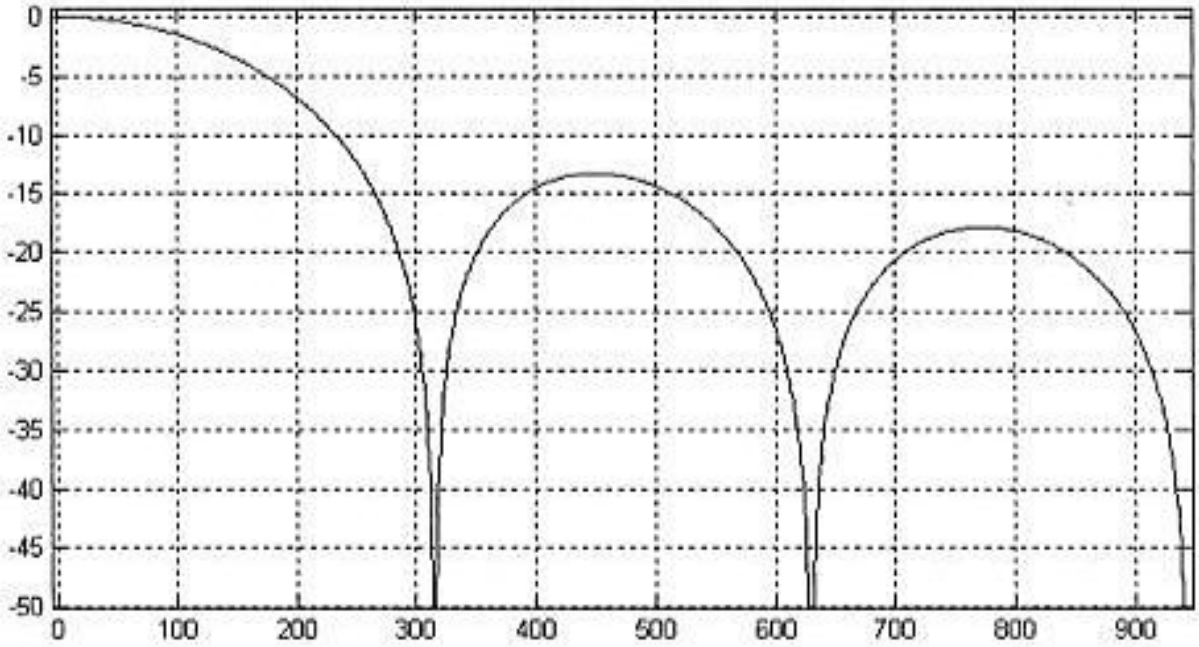


Рис. 15.37. Амплитудно-частотная характеристика однородного фильтра

Давайте включим четыре однородных фильтра последовательно друг за другом. На рис. 15.38 приведены амплитудно-частотные характеристики, полученные для одно- двух- трех- и четырехкаскадного однородного фильтра. На этом рисунке по оси ординат отложен коэффициент передачи фильтра в децибелах, по оси абсцисс — частота.

Как видно из этих амплитудно-частотных характеристик, применение многокаскадного однородного фильтра позволяет достичь требуемого подавления мешающих сигналов.

Обычно однородные фильтры используются в составе устройств повышения частоты дискретизации (интерполяции) сигнала или устройств понижения частоты дискретизации (децимации) сигнала. В этом случае можно дополнительно упростить схему однородного фильтра и тем самым увеличить быстродействие схемы, но мы рассмотрим это в последующих главах.

К сожалению, однородный фильтр вносит амплитудные искажения в полосе полезного сигнала, поэтому обычно он используется вместе с дополнительным нерекурсивным фильтром. Этот дополнительный фильтр кроме своей основной задачи должен компенсировать искажения, вносимые в сигнал однородным фильтром.

На этом можно завершить краткий обзор, посвященный реализации фильтров в цифровом виде. Ну а теперь рассмотрим конкретные примеры использова-

ния цифровых фильтров для реализации типовых задач, встречающихся в радиотехнических устройствах.

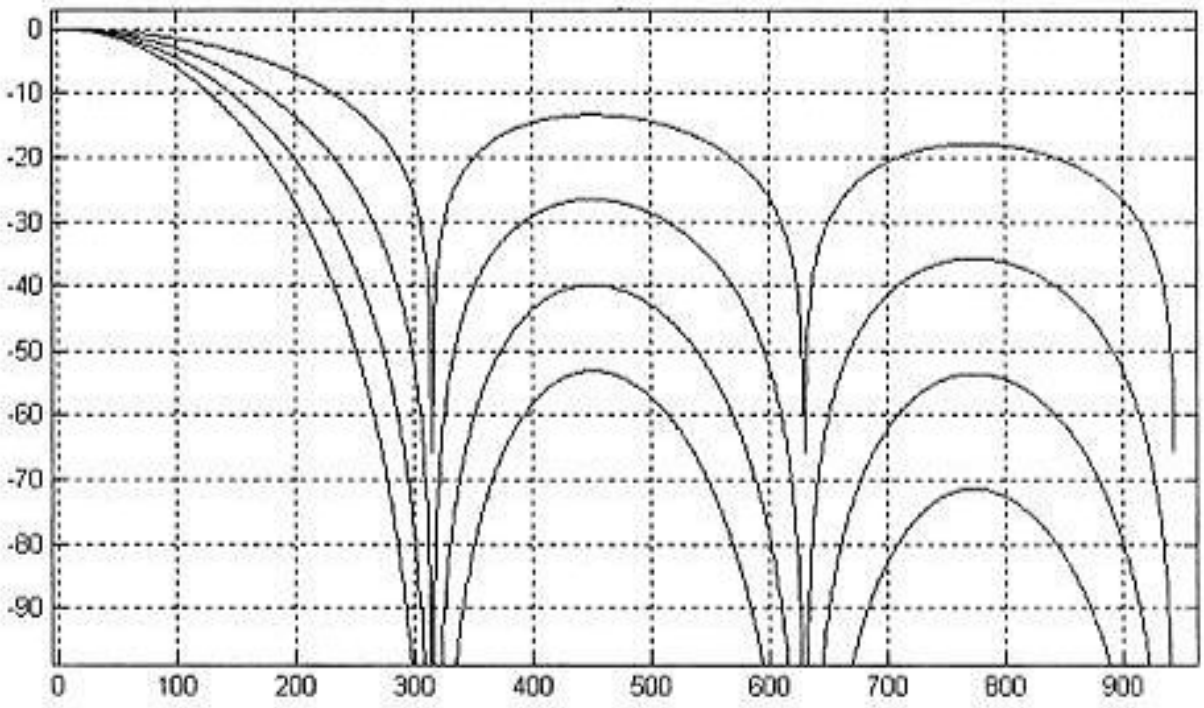
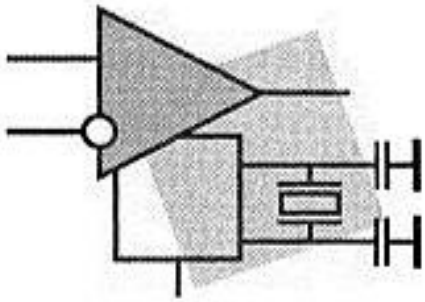


Рис. 15.38. Амплитудно-частотная характеристика многокаскадного однородного фильтра

Итоги

В данной главе мы рассмотрели основные блоки цифровых устройств, предназначенных для обработки данных. В этих блоках необходимо уметь выполнять операции сложения, умножения и хранения данных. В основном они требуются для реализации смесителей и фильтров. В данной главе были приведены особенности реализации цифровых фильтров. Схемы были рассмотрены с уровнем детализации до структурных схем. Применив методы, изученные в предыдущих главах, эти схемы легко превратить в принципиальные. Так как схемы цифровых фильтров требуют достаточно много элементов, то их обычно реализуют на микросхемах ПЛИС. Но даже если вы будете применять готовые микросхемы, содержащие цифровые фильтры, то материал данной главы может пригодиться для понимания особенностей их работы и позволит правильно их настраивать.



ГЛАВА 16

Реализация передатчиков радиосигналов в цифровом виде

Традиционно радиосигналы получали аналоговыми методами при помощи специализированных генераторов синусоидального сигнала и аналоговых смесителей. Однако в последнее время радиосигналы все чаще начинают формировать непосредственно в цифровом виде и только затем они преобразуются в аналоговую форму. Это обусловлено высокой повторяемостью характеристик цифровых устройств и возможностью избежать влияния паразитных связей между блоками, входящими в состав разрабатываемого устройства. Отсутствие паразитных связей между внутренними блоками цифрового устройства, в свою очередь, ведет к возможности получить более высокие характеристики формируемых радиосигналов.

В настоящее время применяются в основном два подхода к формированию радиосигналов в цифровой форме. Первый способ — это формирование квадратурных компонент сигнала I и Q в полосе частот от 0 до $f_в$ с последующим переносом этих сигналов на несущую или промежуточную частоту при помощи аналоговых или цифровых умножителей. Такие устройства получили название квадратурные модуляторы. В иностранной литературе такие устройства получили название "Up converter".

Второй способ формирования сигнала заключается в непосредственной генерации синусоидального сигнала цифровыми методами. При таком способе формирования сигнала предусматривается возможность изменения амплитуды, частоты или фазы радиосигнала. Такие устройства обычно называются полярными модуляторами. В иностранной литературе устройства непосредственной генерации сигнала получили название схем прямого цифрового синтеза — DDS (Direct Digital Synthesis).

При цифровом формировании радиосигнала по методу прямого цифрового синтеза аналоговый сигнал получается на выходе аналого-цифрового преобразователя. Форма генерируемого таким способом аналогового сигнала опре-

деляется цифровыми кодами, подаваемыми на вход аналого-цифрового преобразователя.

Цифровые коды отсчетов полезного сигнала можно хранить в постоянном запоминающем устройстве. Выдавать их на выход ПЗУ можно, последовательно перебирая все возможные комбинации на его адресных входах. Таким способом можно сформировать на выходе цифрового устройства любую форму выходного сигнала.

Упрощенная структурная схема устройства прямого цифрового синтеза (DDS) приведена на рис. 16.1.

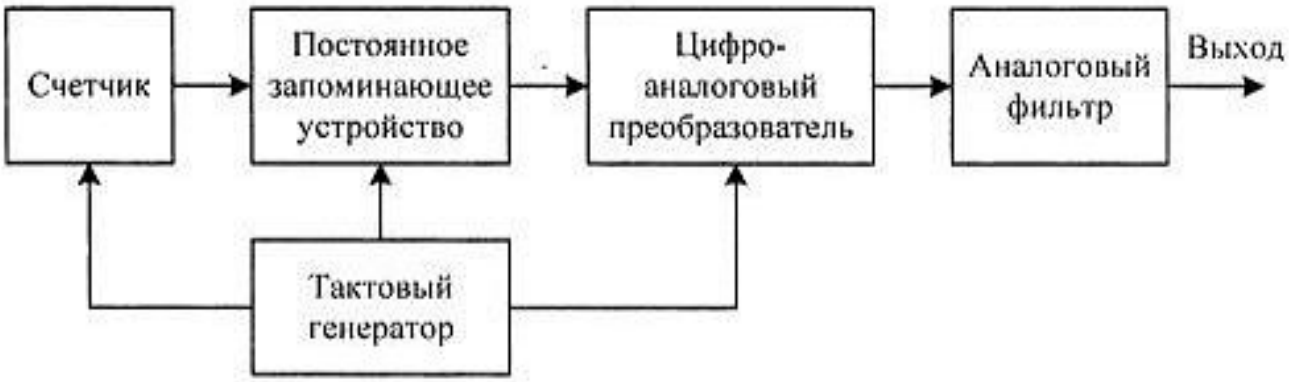


Рис. 16.1. Структурная схема устройства прямого цифрового синтеза

При распространении сигнала по электрическим цепям очень важно, чтобы он не подвергался искажениям. При распространении сигнала по цифровым схемам такому требованию удовлетворяет сигнал прямоугольной формы, однако если подобный сигнал подать на вход аналогового фильтра (а фильтрующими свойствами обладают все аналоговые схемы), то на его выходе прямоугольный сигнал будет искажен. При распространении сигнала по аналоговым схемам форма не изменяется только у синусоидальных сигналов. На выходе аналогового устройства будет изменена только амплитуда и фаза этого сигнала. Именно поэтому, несмотря на неоднократные попытки использовать для радиосвязи сигналы с другой формой, в радиоприемниках и передатчиках до сих пор применяются именно синусоидальные сигналы.

Устройства, позволяющие непосредственно в цифровом виде формировать отсчеты синусоидального сигнала с программируемой частотой, получили название генераторов с цифровым управлением — NCO (Numeral Controlling Oscillator). Рассмотрим работу этих устройств подробнее.

Генераторы с цифровым управлением (NCO)

Для формирования синусоидального радиосигнала в цифровом виде можно воспользоваться ПЗУ с записанными в него значениями функции синуса. При

считывании из него этих значений через равномерные промежутки времени на выходе цифроаналогового преобразователя можно наблюдать синусоидальный сигнал. Пример подобной формы сигнала приведен на рис. 16.2. На этом рисунке кружочками обозначены значения напряжения на выходе цифроаналогового преобразователя. По оси абсцисс отложен номер отсчета цифрового сигнала. Цифровое значение отсчета сигнала считывается из ячейки ПЗУ. Фильтр низкой частоты позволяет сгладить дискретность сигнала на выходе цифроаналогового преобразователя. На рис. 16.2 этот сигнал показан сплошной линией, соединяющей дискретные отсчеты сигнала.

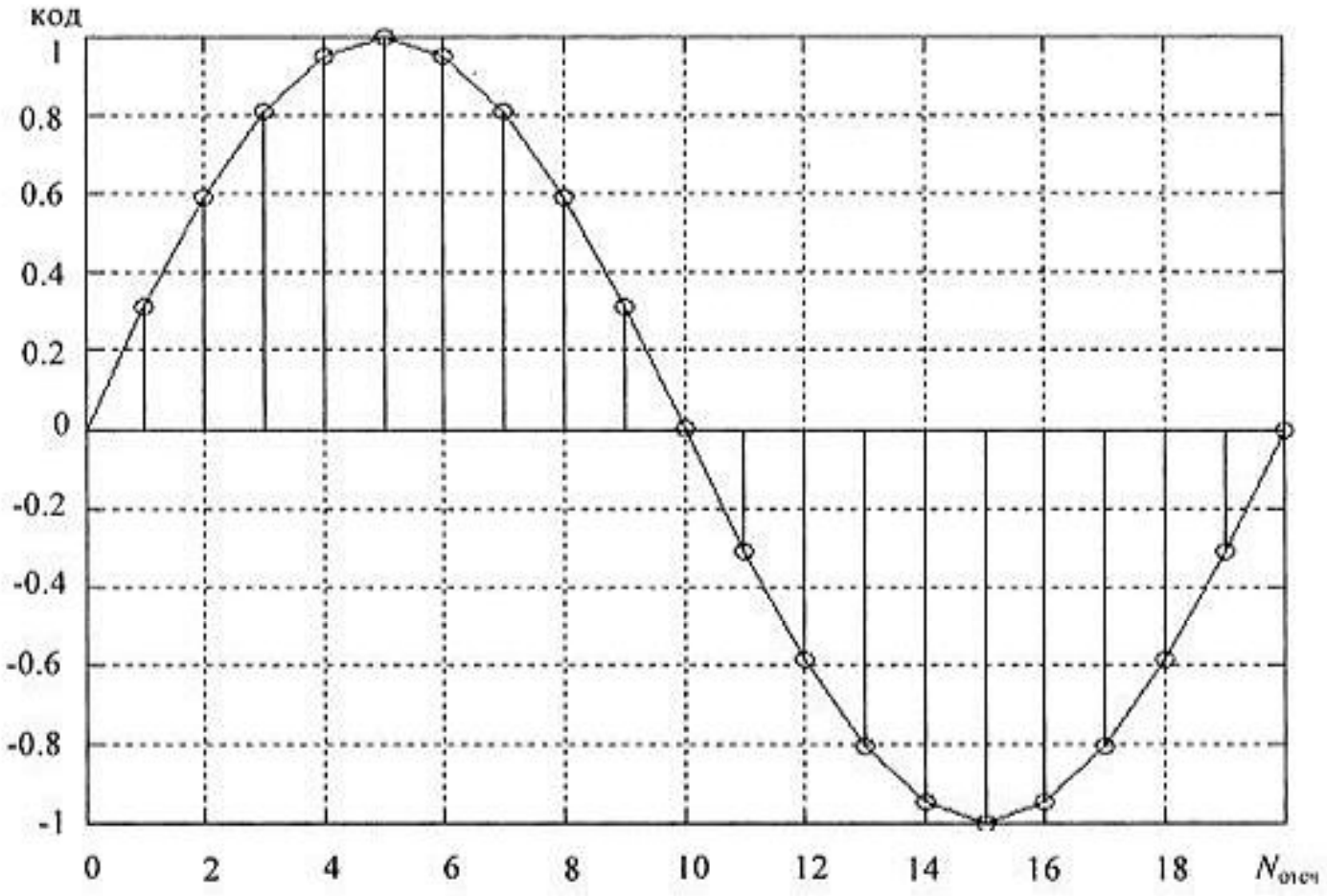


Рис. 16.2. Временная диаграмма сигнала на выходе фазового аккумулятора

Как видно из приведенного рисунка, значение сигнала на выходе цифрового генератора в каждый момент времени определяется номером отсчета сигнала. Частоту полученного синусоидального сигнала можно изменять несколькими способами.

Первый и наиболее очевидный способ заключается в изменении тактовой частоты устройства прямого цифрового синтеза. Однако такой способ изменения частоты выходного сигнала неудобен, т. к. приводит к необходимости применять в качестве тактового генератора синтезатор частот.

Известно, что стабильность частоты колебания, вырабатываемого синтезатором, зависит от диапазона его перестройки. А именно, не существует синтезаторов с большим диапазоном перестройки частоты выходного колебания, которые обладают хорошими спектральными характеристиками.

Еще одним очень существенным недостатком приведенного способа настройки частоты цифрового генератора является то, что синтезатор частот не может мгновенно изменить свою частоту. Некоторое время после изменения частоты настройки синтезатора его частота будет колебаться около нового значения.

Второй способ перестройки частоты заключается в том, что при поступлении очередного тактового импульса можно считывать значения синусоидального сигнала из постоянного запоминающего устройства не из соседних ячеек, а пропуская одно или несколько значений отсчетов синусоидального сигнала.

Если мы будем считывать значения синусоидального сигнала через одну ячейку памяти, то полностью период синусоидального сигнала на выходе аналого-цифрового преобразователя будет получен за время в два раза меньшее относительно первоначально рассмотренного случая. В результате частота формируемого синусоидального сигнала увеличится вдвое.

Если мы будем выдавать на выход цифрового генератора содержимое каждой третьей ячейки ПЗУ синусоидального сигнала, то для завершения одного периода этого сигнала нам потребуется втрое меньшее количество тактов. А это означает, что период такого сигнала будет в три раза короче периода сигнала, полученного при считывании всех ячеек ПЗУ.

Итак, получается, что мы можем регулировать частоту выходного синусоидального сигнала, просто изменяя коэффициент счета входных импульсов. При этом номер отсчета синусоидального сигнала можно считать его фазой, а т. к. в процессе работы схемы номер отсчета постоянно увеличивается, то устройство, осуществляющее это действие, можно назвать аккумулятором (накопителем) фазы.

Фазовый аккумулятор обычно выполняется на арифметическом двоичном сумматоре и регистре, запоминающем результат суммирования. На один из входов сумматора подадим содержимое накапливающего регистра, а на другой вход будем подавать шаг изменения фазы. Структурная схема фазового аккумулятора приведена на рис. 16.3.

Если на вход фазового аккумулятора будем подавать единицу, то эта схема будет работать как обычный двоичный счетчик. При подаче на вход этой схемы двойки числа на выходе накапливающего регистра будут изменяться через два. При подаче на вход числа пять, при поступлении очередного тактового импульса, содержимое аккумулятора будет изменяться на пять единиц.

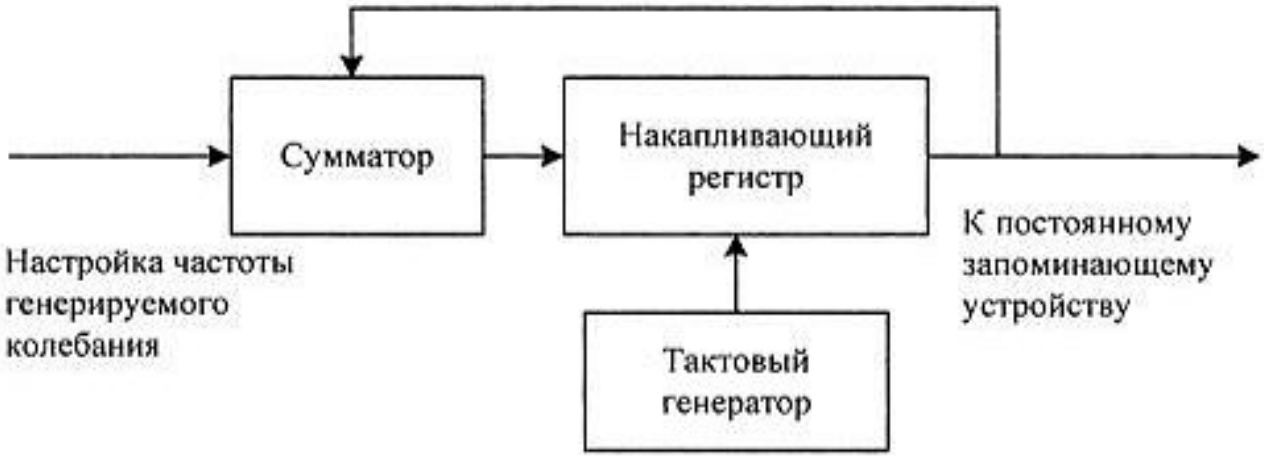


Рис. 16.3. Структурная схема фазового аккумулятора

Даже если на вход фазового аккумулятора подать число ноль, то мы получим нулевое приращение фазы, т. е. получим нулевую частоту (постоянный ток). Итак, изменением числа на входе фазового аккумулятора можно регулировать частоту синусоидального сигнала, формируемого на его выходе.

Теперь определим требования к параметрам блоков, входящих в состав фазового аккумулятора. Сначала определим количество ячеек постоянного запоминающего устройства. Количество ячеек ПЗУ будет определять минимальную частоту, которую мы сможем сформировать фазовым аккумулятором. Чем больше количество ячеек постоянного запоминающего устройства, тем ниже эта частота и, соответственно, меньше шаг перестройки синусоидального генератора прямого цифрового синтеза.

Рассмотрим конкретный пример. Пусть тактовая частота фазового аккумулятора будет 40 МГц. Тогда, если выбрать количество ячеек ПЗУ равным 4096, мы сможем получить минимальную частоту 10 кГц. Современные микросхемы ПЗУ при приемлемой стоимости могут достигать объема 65536 ячеек. При использовании таких ПЗУ для хранения отсчетов функции синуса точность настройки частоты цифрового генератора возрастет до 610 Гц.

Теперь оценим необходимую разрядность ячеек постоянного запоминающего устройства. Для этого определим разность между значениями синуса, хранящимися в соседних ячейках памяти. Разность определим в точке наибольшего изменения функции синуса:

$$\Delta = \sin\left(\frac{2\times\pi}{4069}\right) - \sin(0) = \sin\left(\frac{2\times\pi}{4069}\right) = 1{,}53\times10^{-3}$$

Эта разность соответствует точности одиннадцатиразрядного числа. Одиннадцатиразрядное число обеспечивает точность представления $0{,}98\times10^{-3}$. Это означает, что для хранения значений синусов в постоянном запоминающем устройстве с 4096 ячейками памяти достаточно одиннадцати-двенадцатираз-

рядной точности. Для хранения значений синуса в ПЗУ с 65536 ячейками памяти потребуются уже пятнадцатиразрядные ячейки.

При использовании для хранения синуса постоянного запоминающего устройства с шестнадцатиразрядными ячейками можно реализовать динамический диапазон устройства прямого цифрового синтеза до 96 дБ (приблизительно по 6 дБ на каждый разряд). Это значительно превышает динамический диапазон аналоговых устройств. Динамический диапазон устройства в целом будет ограничиваться аналоговыми цепями, поэтому увеличивать разрядность ячеек ПЗУ синуса выше шестнадцати разрядов не имеет смысла.

В качестве иллюстрации возможностей рассмотренной схемы, на рис. 16.4 приведен спектр сигнала, сформированного цифровым синусоидальным генератором.

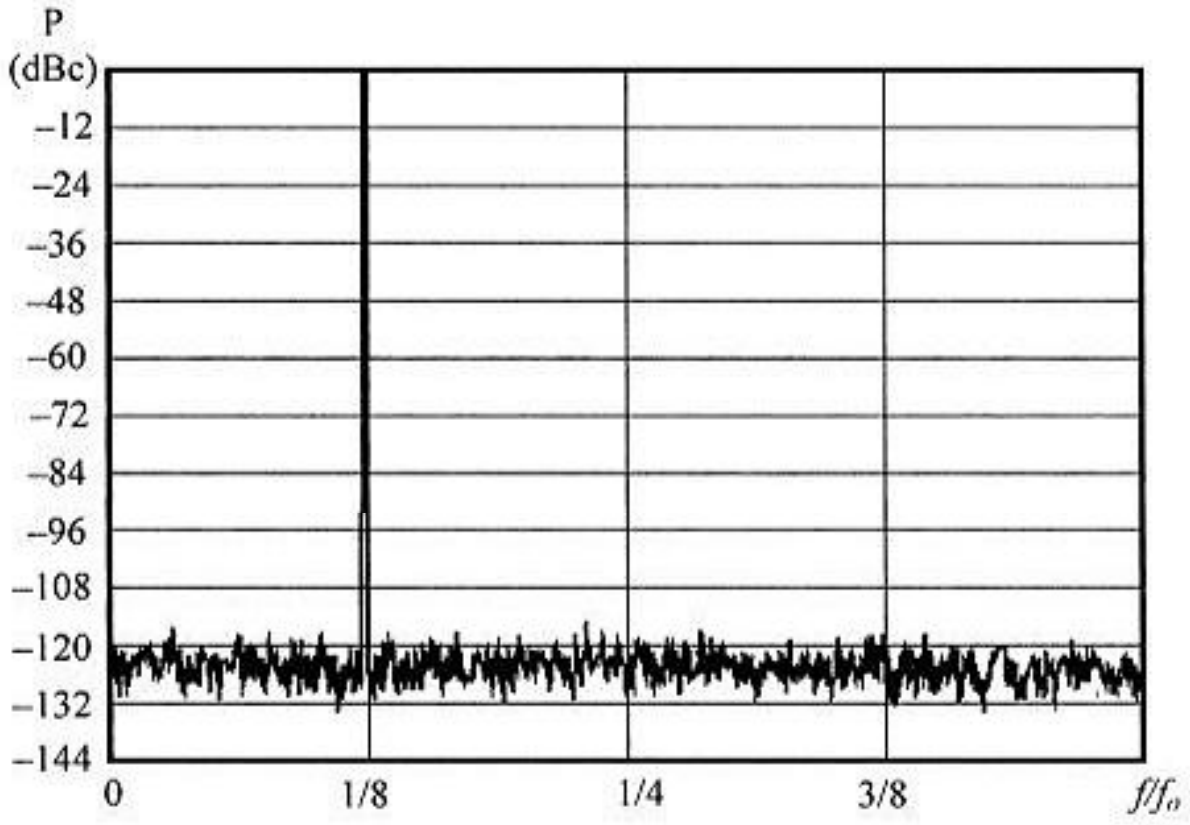


Рис. 16.4. Спектр цифрового синусоидального генератора

Точно так же не имеет смысла увеличивать количество ячеек в этом ПЗУ. Кто сомневается, может провести расчеты по приведенной выше методике. Какой смысл хранить в соседних ячейках одинаковые значения? Так что же, мы не можем получить шаг перестройки синусоидального генератора, реализованного на фазовом аккумуляторе, меньший рассчитанного выше значения? Да нет же, можем.

Теперь определим требования к разрядности накапливающего регистра и сумматора. На первый взгляд разрядность этих устройств должна совпадать

с разрядностью шины адреса постоянного запоминающего устройства, т. е. не превышать шестнадцати разрядов. Однако это не так.

Для этого достаточно увеличить разрядность сумматора и накапливающего регистра, входящих в состав фазового аккумулятора, а на адресные входы постоянного запоминающего устройства подавать старшие разряды результата суммирования, как это показано на рис. 16.5.

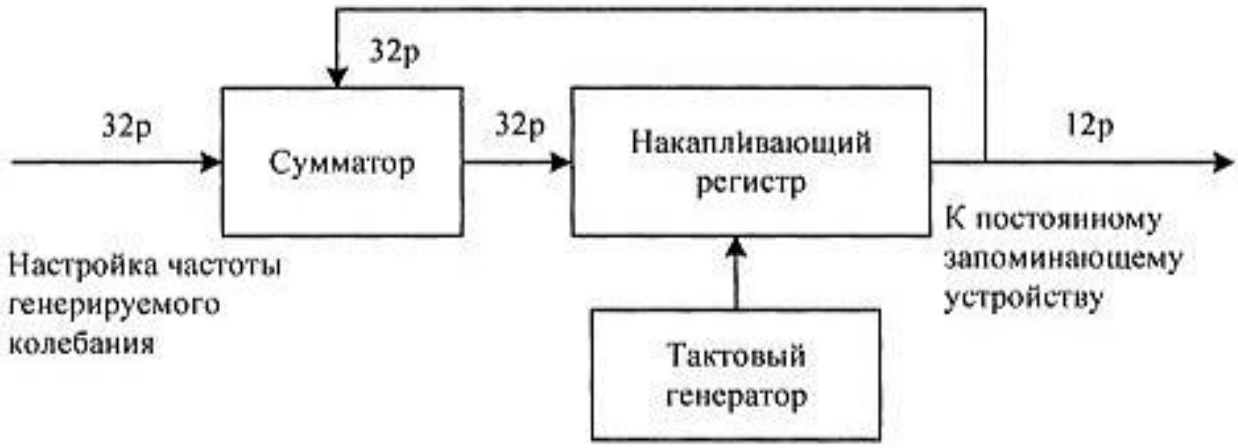


Рис. 16.5. Структурная схема фазового аккумулятора с уменьшенным шагом настройки частоты

В результате применения сумматора с разрядностью, большей разрядности адресной шины постоянного запоминающего устройства, в накапливающем регистре фаза может изменяться с любым сколь угодно малым шагом. При этом код напряжения на выходе ПЗУ будет изменяться только тогда, когда изменение значения синуса превысит шаг квантования цифроаналогового преобразователя.

При использовании шага изменения фазы, меньшего разрядности адресной шины ПЗУ, возможно дробное соотношение периода синуса и периода накопления фазы, равной 360° (переполнения фазового аккумулятора). В этом случае возможно формирование синусоидального сигнала с периодом, отличающимся в соседних интервалах времени. Однако средняя частота генерируемого синусоидального сигнала будет точно равна заданному значению.

Микросхемы прямого цифрового синтеза (DDS)

Итак, мы научились управлять фазой и частотой синусоидального сигнала. Это означает, что мы можем осуществить его частотную и фазовую модуляцию. Однако в ряде случаев требуется уметь управлять амплитудой синусоидального сигнала. Например, такое действие требуется при формировании радиосигнала с амплитудной модуляцией.

Рассмотрим формулу, описывающую синусоидальный сигнал:

$$s(t) = A(t) \times \sin(\varphi(t)\omega t + \phi(t))$$

Анализируя эту формулу, мы видим, что для изменения амплитуды синусоидального сигнала достаточно умножить его на функцию, зависящую от времени. Строить цифровые умножители мы уже научились в предыдущих главах, поэтому, для того чтобы получить амплитудный модулятор, достаточно в структурную схему устройства прямого цифрового синтеза просто добавить цифровой умножитель, как это показано на рис. 16.6.

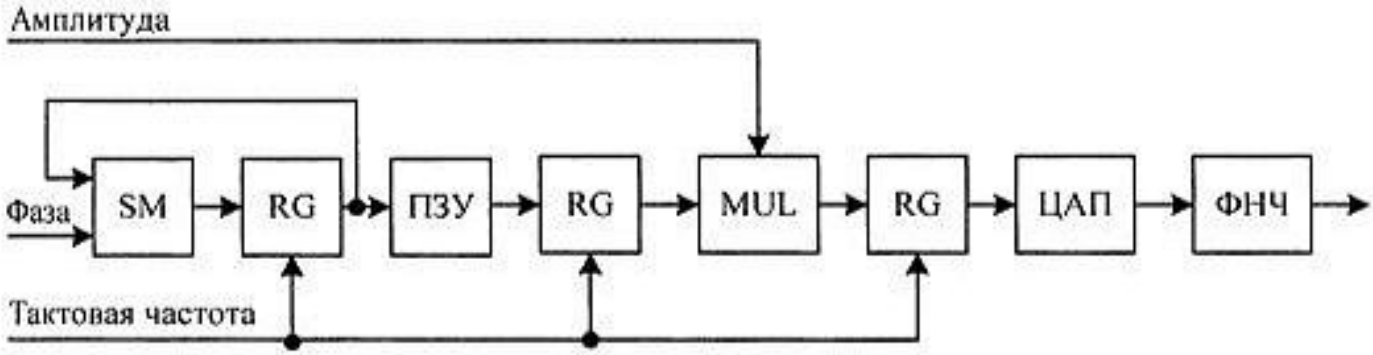


Рис. 16.6. Схема полярного модулятора

В схеме, приведенной на рис. 16.6, после постоянного запоминающего устройства и умножителя поставлены регистры. Они служат для увеличения быстродействия устройства в целом. Период тактовой частоты зависит от времени распространения сигнала по цифровым комбинационным схемам, таким как сумматор, умножитель или дешифраторы адреса постоянного запоминающего устройства.

Время распространения сигналов через каждое из перечисленных устройств меньше времени распространения через все эти устройства сразу, поэтому введение дополнительных регистров позволяет увеличить тактовую частоту схемы прямого цифрового синтеза.

Рассмотренная схема устройства прямого цифрового синтеза позволяет независимо изменять частоту и амплитуду генерируемого сигнала. Именно так представляется синусоидальный сигнал в полярной системе координат. Поэтому приведенная схема прямого цифрового синтеза сигнала получила название полярного модулятора.

Квадратурные модуляторы (Up converter)

Полярные модуляторы позволяют легко реализовать аналоговые виды модуляции, такие как частотная, фазовая или амплитудная модуляции. Из цифровых методов модуляции полярные модуляторы позволяют легко осуществить

такие виды частотной модуляции, как MSK (Minimal Shift keying) и GMSK (Gaussian Minimal Shift keying) модуляции. Реализация же таких видов модуляции как квадратурная амплитудная модуляция (КАМ) или фазовая модуляция с высокой скоростью передачи бит легче осуществляется при использовании квадратурного модулятора.

Для осуществления квадратурной амплитудной модуляции требуются два канала: синфазный канал I и квадратурный канал Q. В каждом из каналов производится умножение входного сигнала на частоту опорного сигнала, сдвинутую на 90°.

На выходе квадратурного модулятора сигналы, полученные с выходов умножителей, суммируются. Для осуществления сдвига фаз в цифровом виде в квадратурных каналах в ПЗУ цифрового генератора опорного сигнала можно записать сразу таблицу значений функций синуса и косинуса. Структурная схема описанного выше цифрового квадратурного модулятора приведена на рис. 16.7.

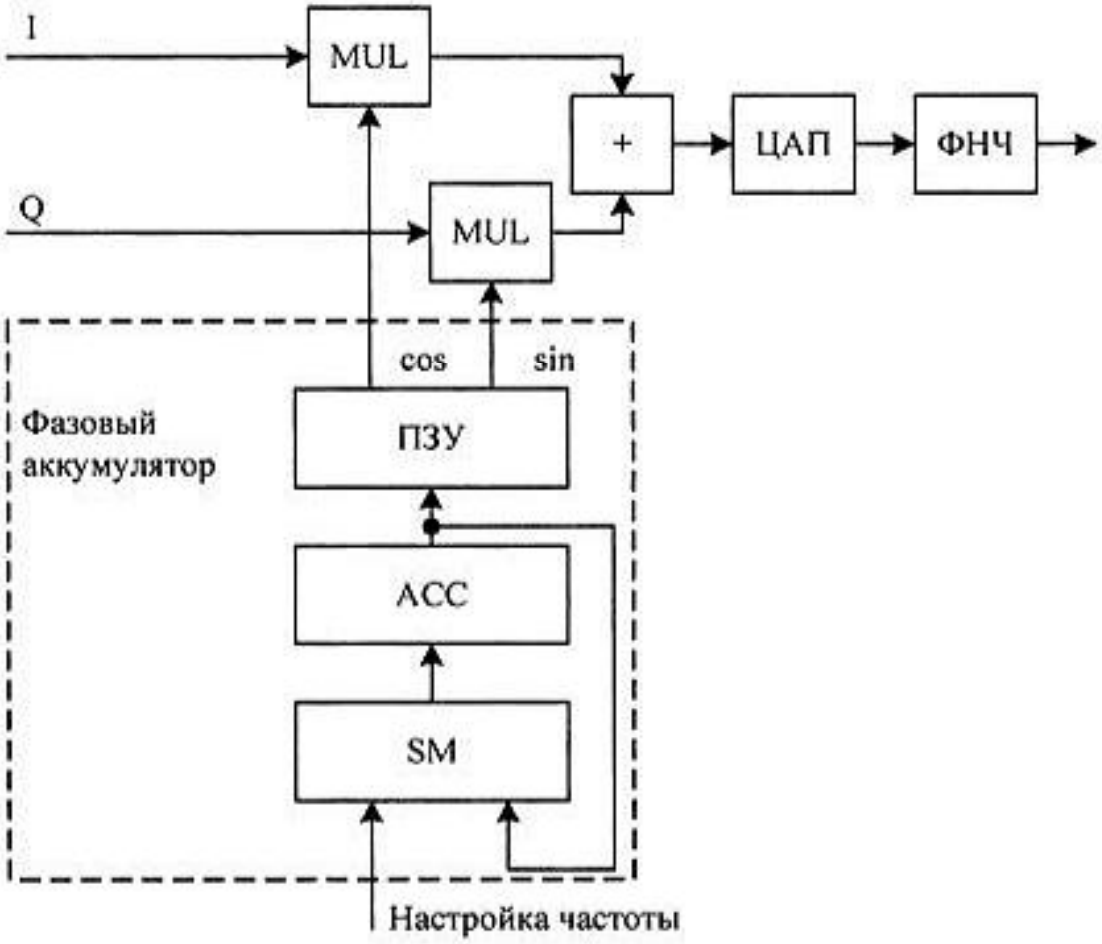


Рис. 16.7. Структурная схема квадратурного модулятора

В схеме, приведенной на рис. 16.7, квадратурные компоненты исходного сигнала I и Q, подаваемые на вход умножителей, должны быть сформированы в

полосе частот от 0 до $f_в$. В этой схеме, как и в схеме полярного модулятора, можно применить промежуточные регистры, увеличивающие ее быстродействие в целом, однако для наглядности рисунка эти регистры не показаны.

В большинстве случаев тактовая частота квадратурных компонент сигнала не совпадает с тактовой частотой, необходимой для формирования высокочастотного сигнала. Так как частота высокочастотного сигнала многократно превосходит верхнюю частоту исходного сигнала, то для ее представления требуется увеличить частоту дискретизации. Однако при осуществлении операции переноса спектра, выполняемой умножителями MUL, сигналы I и Q должны следовать с той же самой частотой дискретизации.

Это означает, что перед подачей на вход арифметических умножителей частота дискретизации исходных квадратурных сигналов должна быть доведена до частоты дискретизации формируемого высокочастотного радиосигнала.

Для увеличения частоты отсчетов квадратурных сигналов обычно применяются интерполирующие цифровые фильтры. Давайте рассмотрим принципы работы этих фильтров подробнее.

Интерполирующие цифровые фильтры

Интерполяцией называется увеличение частоты дискретизации сигнала. Отношение количества выходной частоты дискретизации сигнала к входной частоте дискретизации сигнала называется коэффициентом интерполяции. Обычно коэффициент интерполяции выбирается целым числом.

При увеличении частоты отсчетов цифрового сигнала, в соответствии с теоремой Котельникова, расширяется полоса частот, описываемых этими отсчетами. Это означает, что в новую полосу частот попадает несколько частотных образов первоначального варианта сигнала. При выполнении операции интерполяции необходимо выбрать нужный частотный образ. Обычно выбирается полоса частот от 0 до $f_в$ (низкочастотный образ сигнала).

Задача выбора необходимого частотного образа решается при помощи цифрового фильтра. Такой фильтр называется интерполирующим. Именно этот фильтр вычисляет значения сигнала в точках между первоначальными отсчетами.

Интерполирующий фильтр с конечной импульсной характеристикой

Рассмотрим пример первоначального представления сигнала во временной области. Пример временной реализации исходного сигнала приведен на рис. 16.8.

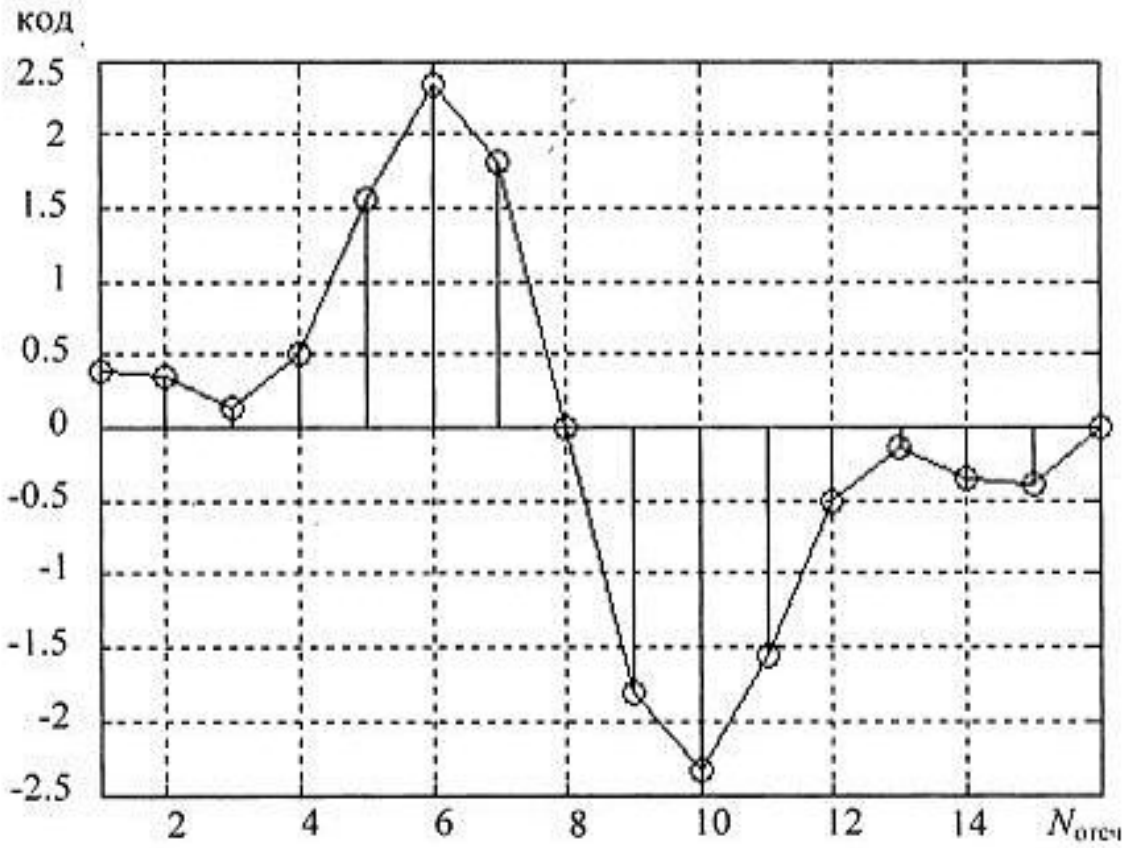


Рис. 16.8. Пример временной реализации сигнала

В данном рисунке отсчеты полезного сигнала обозначены кружочками, а для того чтобы легче было представить форму исходного сигнала, они соединены прямыми линиями. При интерполяции сигнала требуется увеличить количество его отсчетов в единицу времени. При интерполяции новые отсчеты сигнала заполняются нулевыми значениями, как это показано на рис. 16.9.

Здесь необходимо подчеркнуть, что *промежуточные отсчеты должны быть заполнены именно нулями*. Если их заполнить другими значениями, например повторениями предыдущего отсчета сигнала, то уровень высокочастотных составляющих спектра значительно уменьшится. Казалось бы, это облегчит работу интерполирующего фильтра, однако при этом будет искажен и сигнал в интересующей нас полосе частот.

Более того, станет невозможным применять в качестве фильтра-интерполятора фильтры Найквиста, т. к. будет невозможно получить нулевое значение межсимвольной интерференции между соседними символами в отсчетных точках сигнала.

Спектр цифрового сигнала, изображенного на рис. 16.9, приведен на рис. 16.10. На графике четко прослеживается повторяющийся характер его спектра. Теперь, для того чтобы осуществить интерполяцию этого сигнала, необходимо подавить его нежелательные спектральные компоненты, распо-

ложенные на частоте высокочастотных образов входной частоты $f_д$ исходного сигнала.

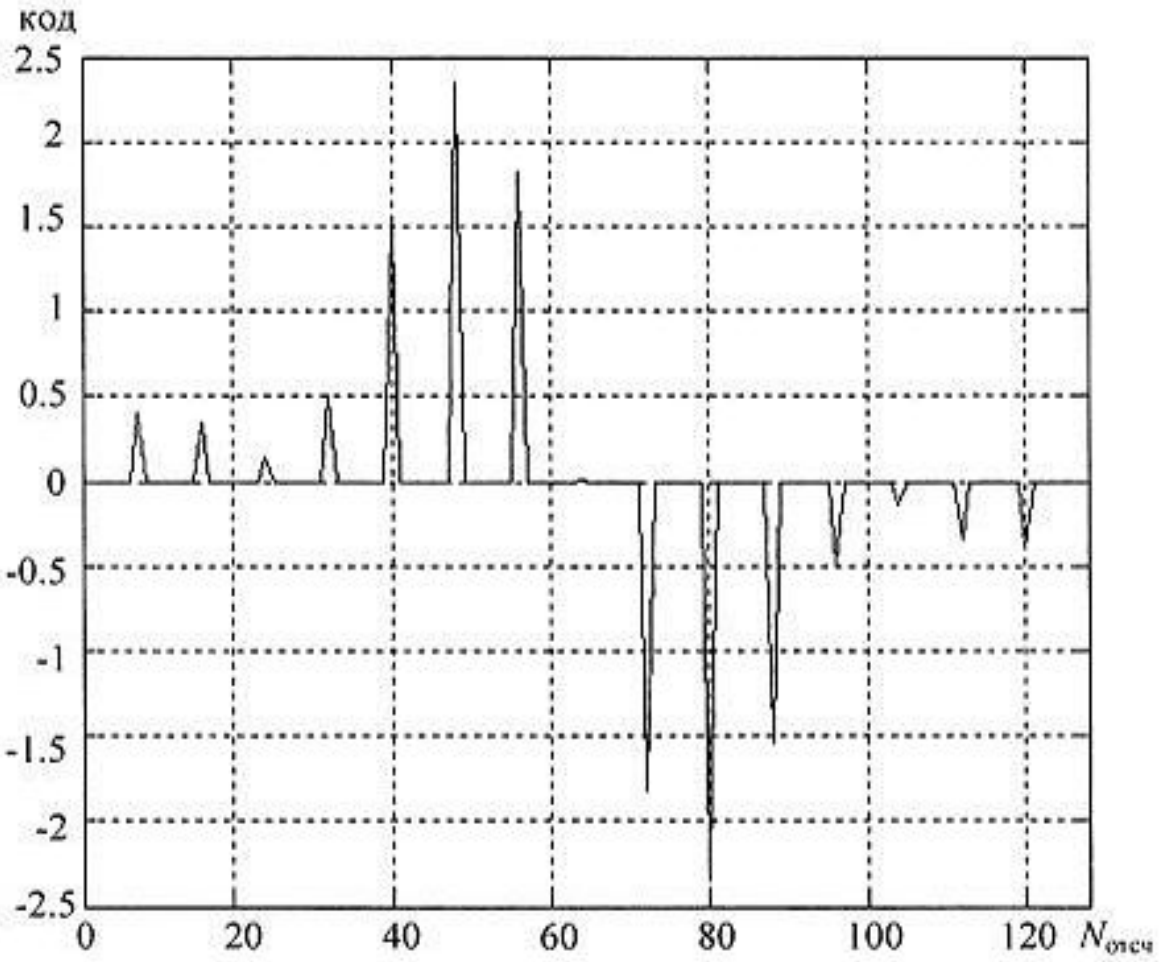


Рис. 16.9. Сигнал на входе интерполирующего фильтра

Подавим все высокочастотные составляющие спектра сигнала при помощи цифрового фильтра низких частот. Для этого зададимся уровнем подавления высокочастотных составляющих спектра равным –75 дБ. Это эквивалентно 12-разрядному представлению полезного сигнала. Такой уровень подавления высокочастотных составляющих спектра можно реализовать при помощи цифрового нерекурсивного фильтра со 128 отводами.

Получившаяся в результате расчета амплитудно-частотная характеристика интерполирующего фильтра с конечной импульсной характеристикой при использовании шестнадцатиразрядных весовых коэффициентов приведена на рис. 16.11.

В полосе пропускания такой фильтр обеспечивает неравномерность коэффициента передачи на уровне 0,001 дБ. Таким образом, учитывая, что фильтр с симметричной импульсной характеристикой обладает линейной фазовой характеристикой, разработанный фильтр практически не вносит искажений в сходный сигнал.

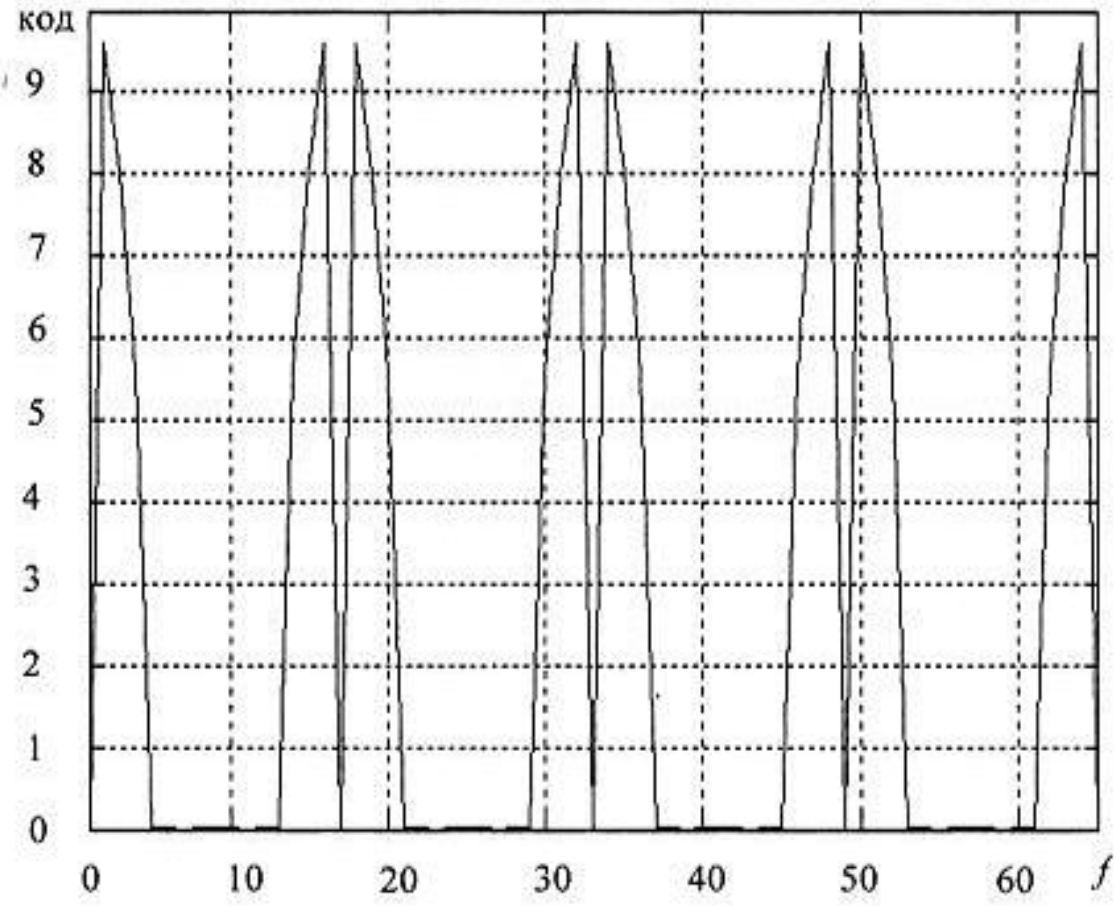


Рис. 16.10. Спектр сигнала, приведенного на рис. 16.9

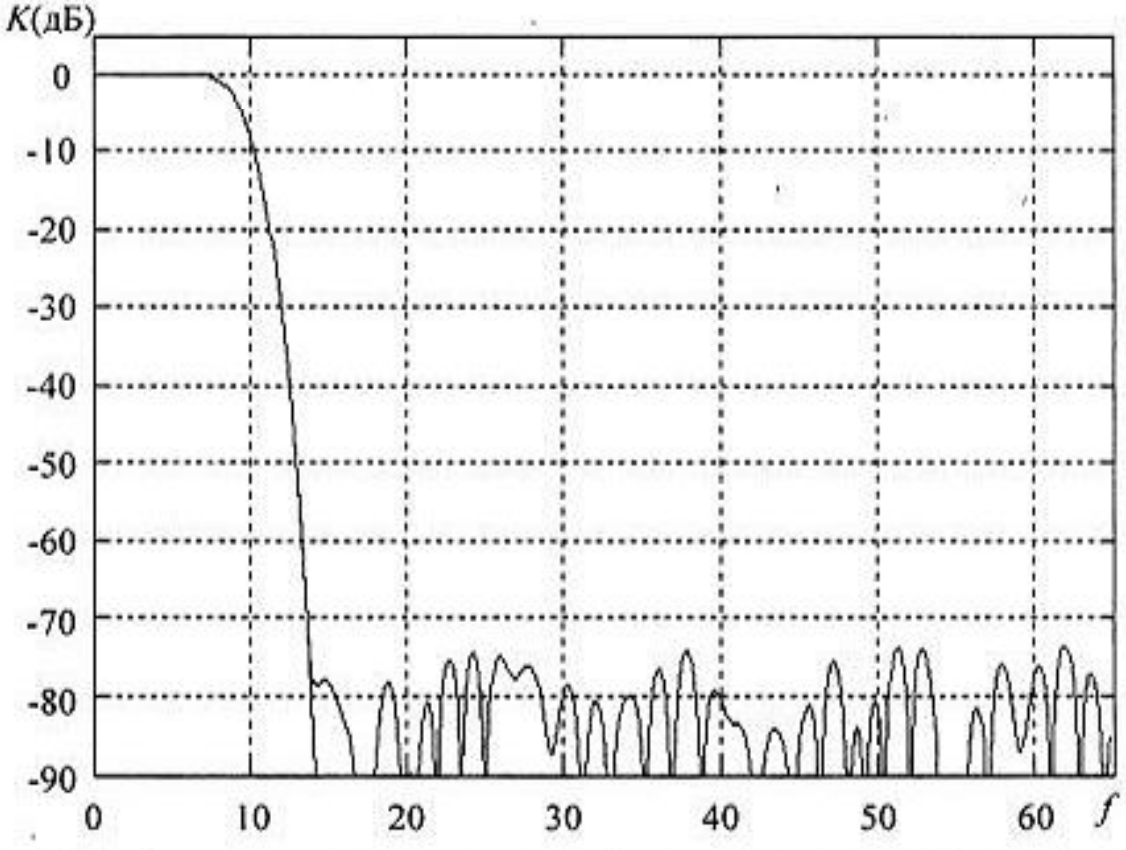


Рис. 16.11. Амплитудно-частотная характеристика интерполирующего фильтра

Сигнал, показанный на рис. 16.9, при прохождении через фильтр с амплитудно-частотной характеристикой, приведенной на рис. 16.11, принимает вид, изображенный на рис. 16.12. ✧ *Обратите внимание*, что сигнал на выходе фильтра будет задержан на время, равное групповой задержке фильтра. Для фильтра с конечной импульсной характеристикой это время равно частоте дискретизации выходного сигнала, умноженной на половину количества отводов фильтра.

На рис. 16.12 приведено 128 временных отсчетов сигнала на выходе фильтра. Они практически сливаются друг с другом, поэтому отдельные отсчеты не выделяются кружочками, как это было сделано на рис. 16.8. Как видно из приведенного на рисунке графика, сигнал на выходе фильтра практически не отличается от исходного (существовавшего до дискретизации) сигнала.

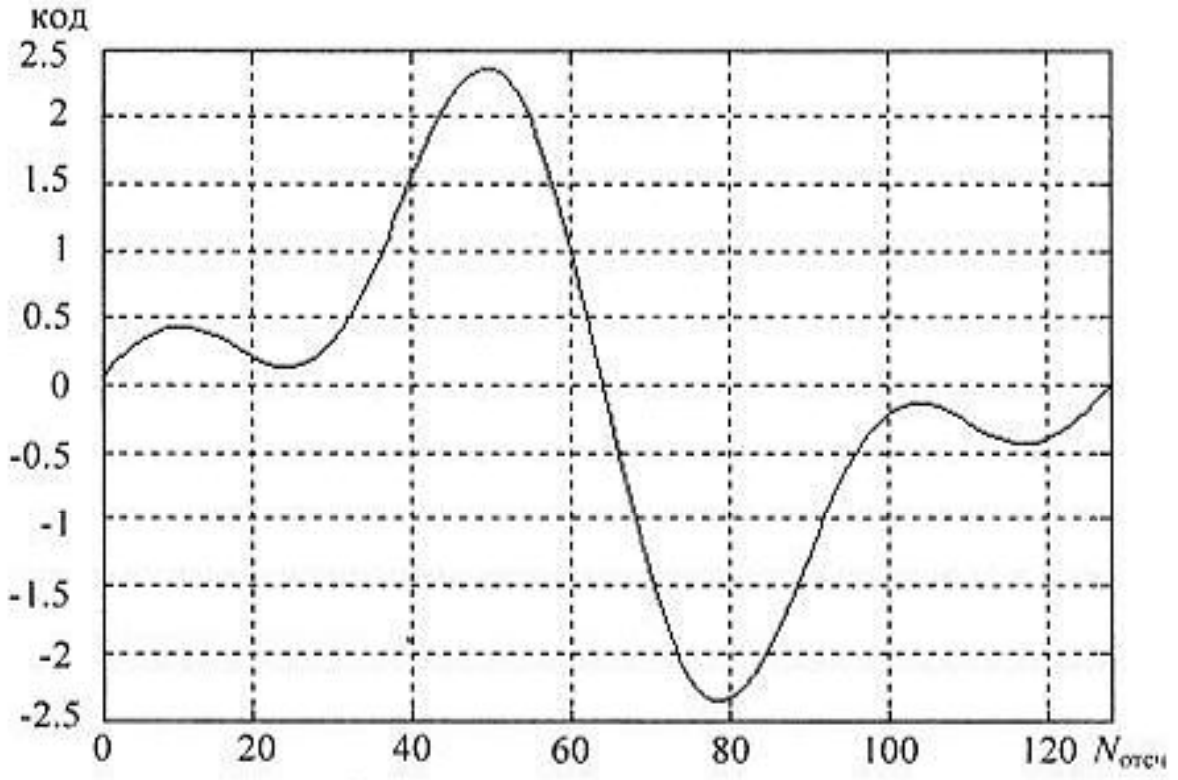


Рис. 16.12. Сигнал на выходе интерполирующего фильтра

При разработке интерполирующего фильтра следует обращать внимание, что, т. к. реальный фильтр всегда имеет конечную крутизну ската своей амплитудно-частотной характеристики, то полоса частот сигнала, подлежащего интерполяции, всегда должна быть меньше половины частоты дискретизации. Только в этом случае интерполяция исходного цифрового сигнала может быть выполнена без искажений.

Теперь рассмотрим, как будет выглядеть этот же сигнал, если из исходного сигнала выделить не нулевой, а первый частотный образ сигнала. Получившийся на выходе полосового фильтра сигнал приведен на рис. 16.13. Для

сравнения на этом же рисунке приведен низкочастотный (нулевой) образ сигнала.

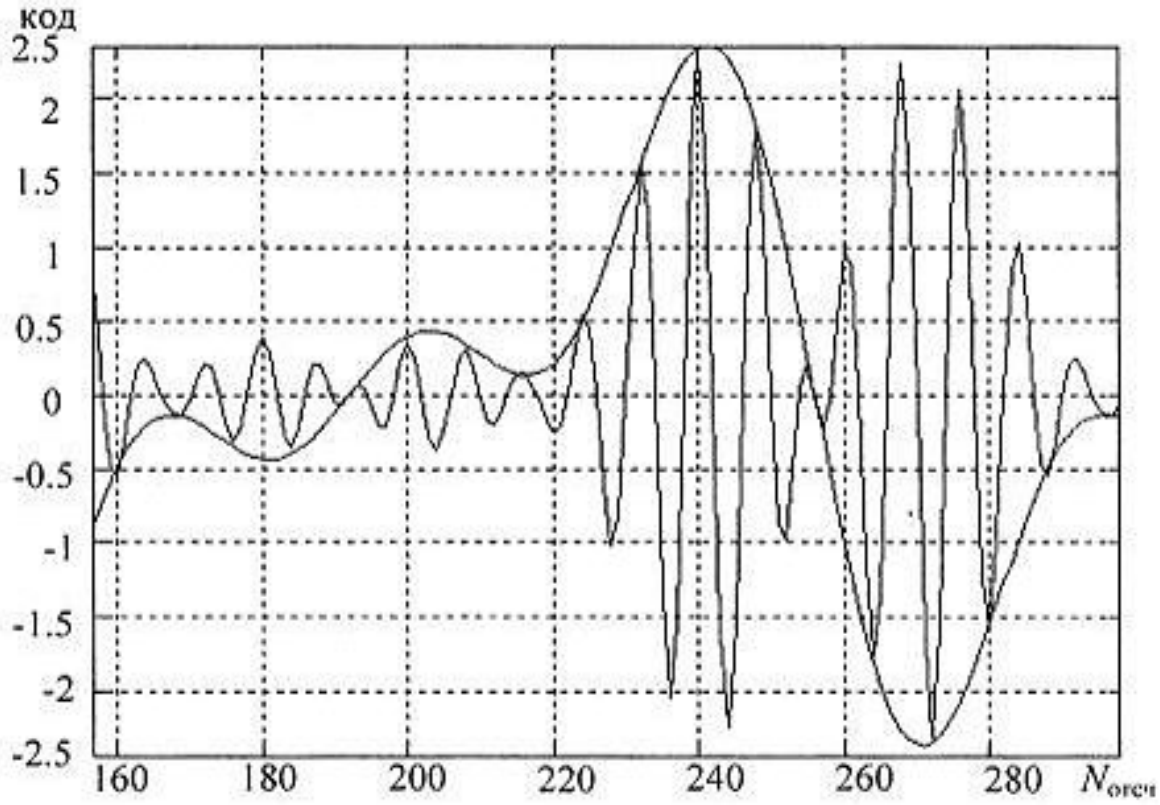


Рис. 16.13. Сигнал на выходе интерполирующего фильтра

На рисунке отчетливо видно, что исходные значения сигнала совпадают как в низкочастотном, так и в первом образе сигнала. В момент пересечения нулевого значения фаза несущей частоты первого образа меняет свой знак. Точно так же вели бы себя и второй и третий образ сигнала. Отличие заключается только в значении несущей частоты.

Применение для формирования несущей частоты высокочастотных образов первоначального сигнала неудобно, т. к. в этом случае можно реализовать всего несколько фиксированных частот. Намного удобнее для переноса спектра исходного сигнала на несущую частоту использовать схему квадратурного модулятора, приведенную на рис. 16.7. Эта схема позволяет переносить спектр исходного сигнала на любую частоту, не превышающую половину частоты дискретизации.

Интерполяция обычно производится в несколько этапов. Первые два этапа, как правило, обеспечивают увеличение скорости отсчетов сигнала в два раза каждый. Это связано с тем, что первоначально почти вся полоса частот от 0 до $f_д / 2$ занята полезным сигналом, т. е. полезный сигнал и его высокочастотные образы находятся близко друг от друга. В результате от интерполирующего фильтра требуется высокая крутизна ската амплитудно-частотной

характеристики и для его реализации требуется большое количество отводов и коэффициентов.

После выполнения этих первых двух этапов интерполяции полезный сигнал занимает только 25% полосы частот. В результате требования к избирательности фильтра уменьшаются, а значит, последующий интерполирующий фильтр может обеспечить больший коэффициент интерполяции.

Параллельная реализация интерполирующего фильтра с конечной импульсной характеристикой

Для реализации фильтра с конечной импульсной характеристикой обычно требуется несколько десятков отводов от цифровой линии задержки. При интерполяции это становится определяющим фактором, ведь при увеличении количества отводов уменьшается быстродействие цифрового фильтра! А в результате интерполяции частота дискретизации возрастает, и быстродействие фильтра становится определяющим значением.

Неужели в интерполирующих фильтрах придется использовать прямую реализацию цифрового фильтра с конечной импульсной характеристикой? Какой же ток тогда будет потреблять наша микросхема! Однако не так все страшно. Давайте вспомним, что в интерполирующем фильтре большинство входных отсчетов являются нулями.

Получается, что большую часть производительности нашей микросхемы мы будем тратить на умножение на ноль!

Итак, у нас появилась реальная возможность уменьшить количество операций, которые необходимо выполнить в период между импульсами выходной дискретизации. Получается, что нам следует умножать на весовые коэффициенты фильтра только те отсчеты входного сигнала, которые в данный момент не равны нулю.

Но как же это сделать? Ведь каждый отсчет последовательно проходит через все весовые коэффициенты. Рассмотрим пример интерполирующего фильтра на основе полуполосного фильтра.

Частоту дискретизации на выходе этого фильтра можно только удвоить по отношению к входной частоте, поэтому в данном фильтре каждый второй отсчет в линии задержки содержит ноль. Тогда все весовые коэффициенты фильтра можно разбить на две части — четную и нечетную.

Расположим эти части в постоянном запоминающем устройстве последовательно друг за другом. В результате у нас получились два фильтра с длиной, в два раза меньшей по сравнению с исходным фильтром. При этом фильтры

будут выдавать свои отсчеты на выход по очереди. Сначала один, а затем другой.

Что интересно, для обоих фильтров можно воспользоваться одной и той же линией задержки, только сдвиг в этой линии следует делать после того, как отработают все фильтры. Для реализации этого фильтра можно воспользоваться структурной схемой нерекурсивного фильтра, приведенной на рис. 15.35.

Повторим эту схему на рис. 16.14. Для реализации новых принципов работы потребуются небольшие изменения. Прежде всего, т. к. длина линии задержки уменьшается вдвое, то коэффициент счета счетчика Сч1 будет равен $N/2$. В результате за $N+1$ импульс тактовой частоты этот счетчик успеет совершить два оборота.

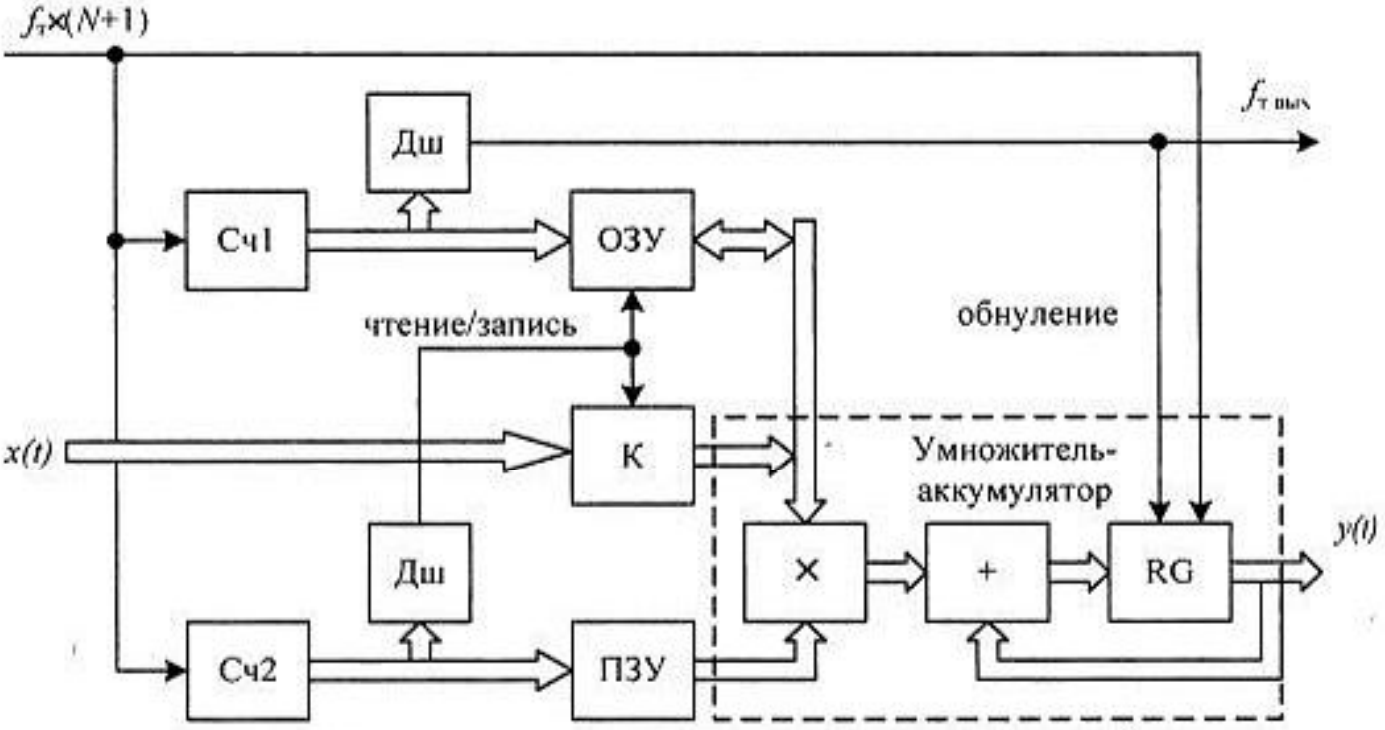


Рис. 16.14. Структурная схема цифрового интерполирующего фильтра

В качестве еще одного преимущества данной схемы следует отметить то, что нулевые значения в фильтр-интерполятор заносить уже не требуется.

Рассмотренные принципы позволяют реализовать фильтр-интерполятор с любым коэффициентом интерполяции, однако на последних стадиях интерполяции обычно используют обладающие наиболее простой структурой и максимальным быстродействием — однородные интерполирующие фильтры.

Давайте рассмотрим подробнее особенности реализации интерполирующих однородных фильтров.

Интерполирующий однородный фильтр

Наиболее просто в цифровом виде реализовать рассмотренный нами ранее однородный фильтр, т. к. для его реализации не требуются цифровые умножители. Для однородного фильтра седьмого порядка формула выглядит следующим образом:

$$y(n) = x(n) + x(n-1) + x(n-2) + x(n-3) + x(n-4) + x(n-5) + x(n-6)$$

Структурная схема фильтра, реализующего данную формулу, приведена на рис. 16.15.

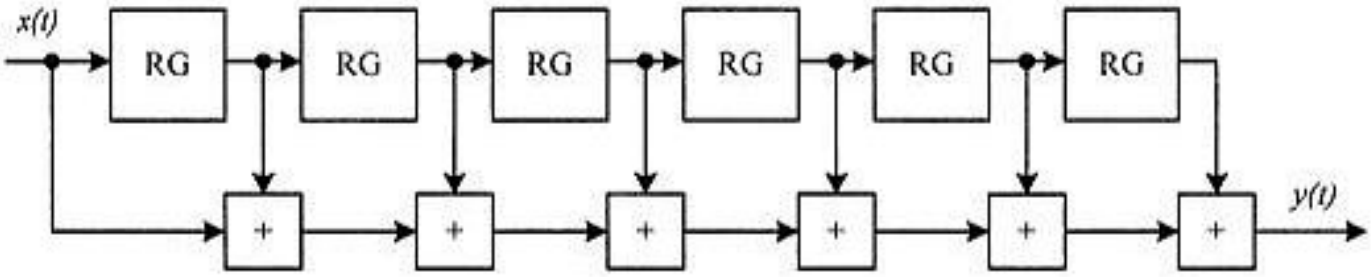


Рис. 16.15. Структурная схема однородного фильтра седьмого порядка

При реализации такого фильтра потребуется 6 сумматоров. Во столько же раз уменьшится быстродействие цифрового фильтра. Можно несколько видоизменить структуру данного фильтра. Для сокращения количества выполняемых операций формула может быть переписана в следующем виде:

$$y(n) = x(n) + y(n-1) - x(n-6)$$

Эта формула может быть реализована за два действия:

$$y(n) = x(n) + y(n-1)$$
$$y(n) = x(n) - x(n-6)$$

В таком случае для реализации однородного фильтра потребуется два каскада. Второй каскад будет выполнять интегрирование, а первый — это фильтр с конечной импульсной характеристикой всего с двумя ненулевыми коэффициентами, равными единице. Новая структурная схема однородного фильтра приведена на рис. 16.16.

В этой схеме максимальное время задержки сигнала определяется быстродействием сумматора и временем записи в регистр. Мы увеличили быстродействие схемы почти в семь раз.

Ну а теперь вспомним, что анализируемый фильтр работает при частоте дискретизации сигнала в N раз выше частоты дискретизации входного сигнала. В этом случае для формирования того же самого значения времени задержки

до увеличения частоты дискретизации нам потребуется всего только один регистр, т. к. на его вход тактовой синхронизации будет поступать частота, в шесть раз меньше, чем частота синхронизации на выходе интерполирующего фильтра.

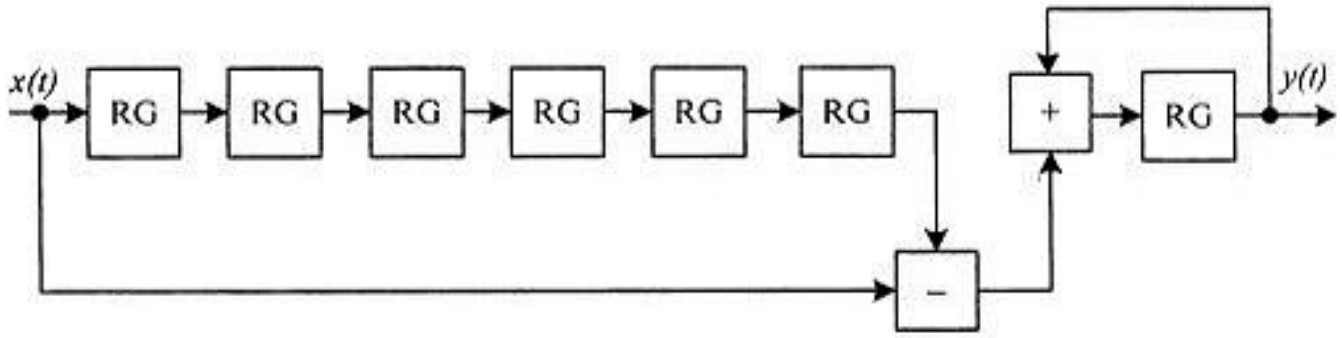


Рис. 16.16. Структурная схема двухкаскадного фильтра, эквивалентного фильтру, приведенному на рис. 16.15

Это означает, что имеет смысл тактировать первую часть фильтра входной частотой дискретизации. Получившаяся в результате всех описанных выше преобразований структурная схема однородного фильтра-интерполятора приведена на рис. 16.17.

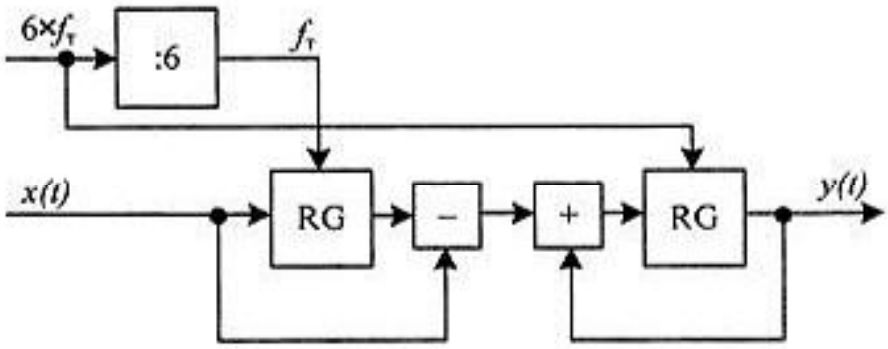


Рис. 16.17. Структурная схема фильтра-интерполятора, эквивалентного фильтру, приведенному на рис. 16.16

Новая схема содержит всего два регистра и два двоичных сумматора, т. е. количество элементов в этой схеме получилось в три раза меньше, чем в схеме однородного фильтра, приведенной на рис. 16.15.

Получившийся в результате преобразований фильтр трудно назвать однородным, однако для того, чтобы отобразить особенности его импульсной и амплитудно-частотной характеристик, сохраним название "однородный" и для этого фильтра. В иностранной литературе такой фильтр получил название CIC-фильтр (Cascaded integrator-comb filter).

Если по техническому заданию требуется еще больший коэффициент интерполяции по сравнению с рассмотренным выше случаем, то выигрыш при реа-

лизации однородного фильтра-интерполятора по схеме, приведенной на рис. 16.17, будет еще большим.

Хотелось бы напомнить, что при анализе характеристик однородного фильтра для получения приемлемого уровня подавления мешающего сигнала нам потребовалось включить друг за другом несколько каскадов.

Давайте включим последовательно друг за другом три фильтра-интерполятора, как это показано на структурной схеме фильтра, приведенной на рис. 16.18.

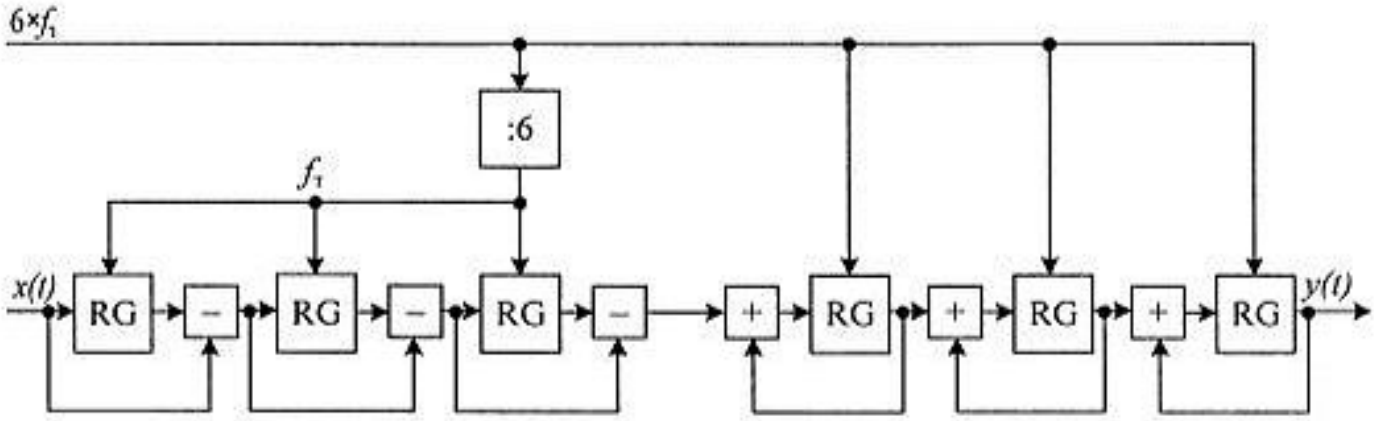


Рис. 16.18. Структурная схема трехкаскадного фильтра-интерполятора (CIC^3)

На рис. 16.19 приведена амплитудно-частотная характеристика четырехкаскадного однородного фильтра. ✧ *Обратите внимание*, что образ полезного сигнала сосредоточен около частоты дискретизации исходного сигнала $x(t)$.

Проанализировав амплитудно-частотную характеристику четырехкаскадного однородного фильтра можно определить, что этот фильтр в полосе частот высокочастотных образов полезного сигнала может обеспечить подавление мешающих сигналов до 90 дБ, что эквивалентно 16-разрядному представлению полезного сигнала. На приведенном рисунке черным цветом закрашена область частот, которая подавляется однородным интерполирующим фильтром. Остальные частоты (там, где находятся максимумы в полосе подавления однородного фильтра) были подавлены ранее полуполосными интерполирующими фильтрами.

На этом закончим обзор особенностей реализации интерполирующих цифровых фильтров.

В заключение необходимо привести результирующую структурную схему цифрового квадратурного модулятора, на вход которой можно подавать цифровой поток с частотой отсчетов, в несколько раз более низкой по отношению к требующейся для формирования выходного радиосигнала. В этой схеме на входе умножителей используются интерполирующие фильтры. Схема цифрового квадратурного модулятора приведена на рис. 16.20.

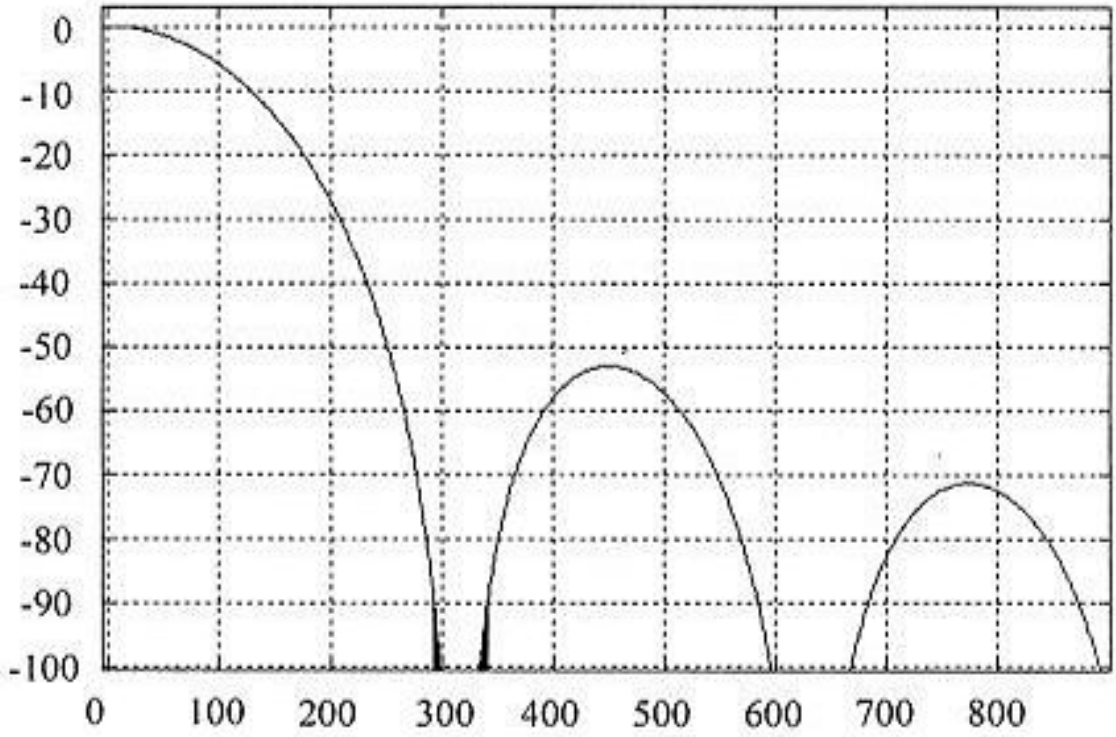


Рис. 16.19. Амплитудно-частотная характеристика четырехкаскадного однородного фильтра-интерполятора

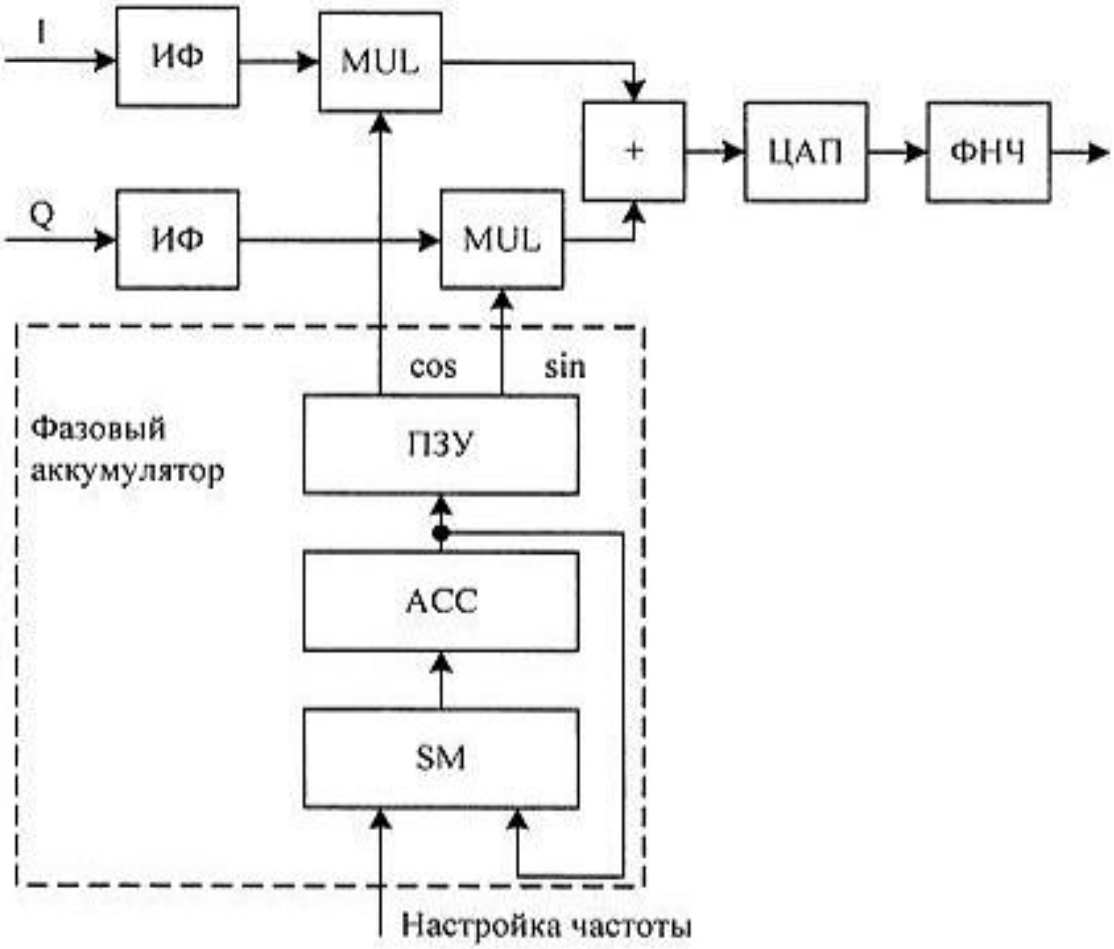


Рис. 16.20. Структурная схема квадратурного модулятора с низкоскоростным потоком квадратурных сигналов

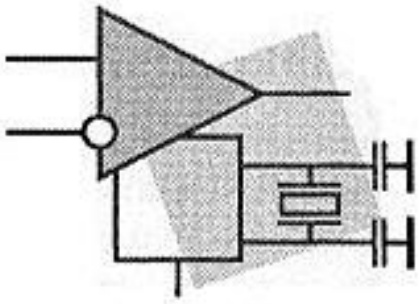
Итоги

В этой главе мы рассмотрели одно из современных направлений применения цифровой техники в системах радиосвязи — формирование радиосигналов непосредственно в цифровом виде. Это направление позволяет формировать радиосигналы с недостижимой ранее точностью, что позволяет уменьшить уровень внеполосных излучений, увеличить скорости передачи данных по радиоканалам и более эффективно использовать радиочастотный спектр.

С точки зрения радиоаппаратуры применение рассмотренных методов позволяет уменьшить габариты ее передающих узлов и со временем уменьшить потребление ею электрического тока.

При формировании радиосигналов используются два вида описания радиосигналов — непосредственное применение формулы узкополосного модулированного колебания и разложение радиосигнала на две квадратурные компоненты. Применение интерполирующих фильтров позволяет подавать на вход этих схем достаточно низкоскоростные потоки цифровых данных, что значительно снижает нагрузку на сигнальные процессоры, где обычно происходит формирование модулирующих сигналов.

ГЛАВА 17



Реализация радиоприемников в цифровом виде

При построении радиоприемных устройств действуют очень жесткие требования к избирательности по соседнему каналу. Вполне реальна ситуация, когда требуется принять сигнал от удаленной станции при условии, что на соседнем канале работает близкорасположенный мощный передатчик.

Пусть уровень полезного сигнала будет 0,2 мкВ, а мощность соседнего передатчика 100 Вт. Рассчитаем уровень напряжения, наводимый этим передатчиком на входе радиоприемного устройства:

$$U = \sqrt{P \times R_{вх}} = \sqrt{100 \times 50} = 70 \text{ В}$$

Теперь определим, во сколько раз требуется подавить этот сигнал для нормального приема полезного сигнала. Нормальный прием обычно осуществляется при отношении сигнал/помех равный 12 дБ (четыре раза). Следовательно, требуется подавить помеху до уровня:

$$U_{пом} = \frac{0,2}{4} = 0,05 \text{ мкВ}$$

А это означает, что мы должны подавить помеху в

$$K_{под} = \frac{U_{пом}}{U_{пом\ доп}} = \frac{70}{0,05 \times 10^{-6}} = 1,4 \times 10^{9} \text{ раз} = 183 \text{ дБ}$$

Ни одно из существующих в настоящее время устройств не способно выполнить эти требования. Аналоговые радиоприемные устройства позволяют подавить сигнал соседнего канала на 80 дБ. Это приводит к необходимости организационных мер, которые позволяют избежать описанной ситуации.

Разработка радиоприемных устройств с более высокими параметрами по избирательности по соседнему каналу могла бы существенно ослабить требо-

вания к взаимному размещению радиоэлектронных средств или, что то же самое, увеличить пропускную способность радиоэфира.

Цифровые фильтры достаточно легко позволяют обеспечить подавление нежелательных сигналов до 120 дБ. В то же самое время предельное значение подавления нежелательных каналов в аналоговых фильтрах ограничивается величиной 80 дБ. Этим объясняется интерес к разработке радиоприемных устройств, выполненных полностью с использованием цифровых технологий.

Структурные схемы приемников радиосигналов, реализованных в цифровом виде, не отличаются от классических схем, используемых в аналоговой схемотехнике. Наиболее распространена супергетеродинная схема с переносом принимаемой частоты на промежуточную частоту. Структурная схема высокочастотного тракта супергетеродинного приемника приведена на рис. 17.1.

Рис. 17.1. Структурная схема супергетеродинного приемника

В этой схеме после переноса смесителем частоты принимаемого сигнала на промежуточную частоту полезный сигнал выделяется при помощи фильтра основной избирательности (ФОИ). В аналоговых схемах требуются усилители для обеспечения оптимального режима работы демодулятора (ДМ).

При цифровой реализации супергетеродинного тракта следует принимать во внимание, что цифровые фильтры обычно обладают усилением, поэтому усилитель может не потребоваться.

Цифровые преобразователи частоты

Фильтр с хорошими избирательными свойствами можно получить только на определенной частоте. Реализация перестраиваемого фильтра с подобными характеристиками, а тем более с характеристиками одинаковыми при перестройке по диапазону частот, практически невозможна. Это было определено еще при разработке аналоговых приемников.

Именно поэтому обычно производится преобразование входного сигнала к определенной, заранее известной частоте. Эта частота называется промежуточной частотой.

Задача преобразователя частоты заключается в переносе спектра принимаемого сигнала на заданную частоту без искажений. Из теории построения радиоприемных устройств известно, что перенос частот заданного диапазона на промежуточную или на нулевую частоту наилучшим образом осуществляет умножитель, в каком бы виде он не был реализован.

Для этого входной сигнал необходимо умножить на синусоидальное напряжение гетеродина. На выходе умножителя напряжение (или цифровые отсчеты сигнала) может быть выражено при помощи следующей формулы:

$$U_{вых} = A_c \times \sin(\omega_c t) \times \sin(\omega_r t)$$

Как известно, при помощи тригонометрических преобразований это выражение может быть приведено к следующему виду:

$$U_{вых} = \frac{1}{2}(\cos((\omega_r - \omega_c)t) - \cos((\omega_r + \omega_c)t))$$

Это означает, что на выходе умножителя входной сигнал переносится на частоты, равные сумме и разности частот принимаемого сигнала и гетеродина. Обычно преобразование частоты осуществляется вниз. Значение промежуточной частоты при преобразовании вниз может быть определено по следующей формуле:

$$\omega_{пр} = \omega_r - \omega_c$$

Именно эта частота выделяется при помощи полосового фильтра. Проблема заключается в том, что аналоговые схемы могут выполнять операцию умножения только с некоторыми ограничениями. В результате на выходе преобразователя частоты появляется неподавленное напряжение гетеродина и сигнала, появляются продукты преобразования гармоник сигнала и гетеродина.

Кроме того, аналоговые преобразователи частоты способны выполнять операцию умножения только в определенном диапазоне частот и напряжений. Это связано с большим динамическим диапазоном входного и выходного сигнала, что приводит к значительному изменению тока преобразователя. Отношение минимального и максимального токов может достигать значения в несколько порядков. Очень трудно обеспечить желаемые характеристики электронного прибора при таком диапазоне изменения рабочего тока.

Цифровой умножитель выполняет операцию умножения непосредственно как математическую операцию. Поэтому мы можем заранее рассчитать допустимый уровень мешающих сигналов. При необходимости можно снизить этот уровень, для этого достаточно увеличить разрядность умножителя или увеличить частоту дискретизации входного сигнала.

Структурная схема цифрового приемника приведена на рис. 17.2.

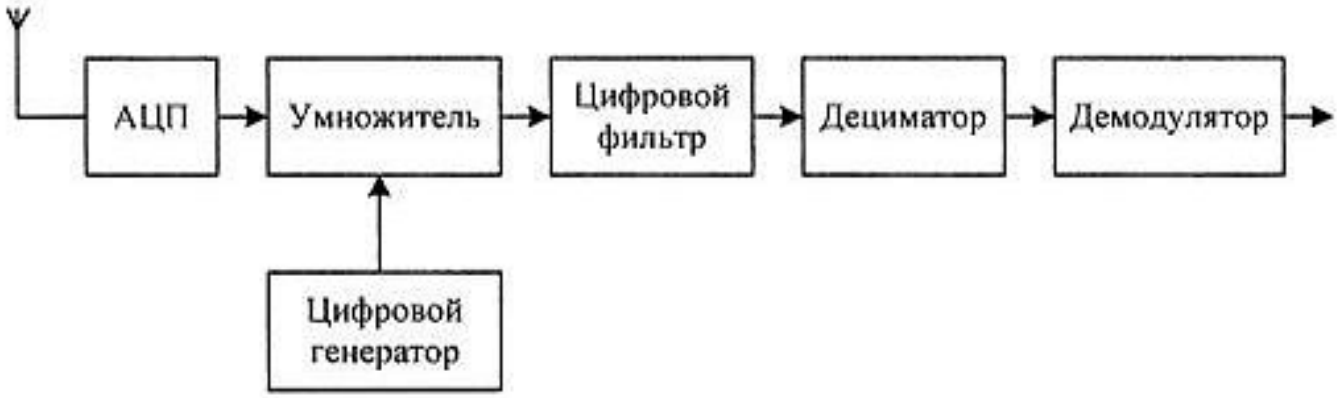


Рис. 17.2. Структурная схема цифрового приемника

Цифровой квадратурный демодулятор

В цифровых приемниках перенос частоты осуществляется сразу на нулевую частоту. При приеме сигналов со сложными видами модуляции важен точный прием не только амплитудной, но и фазовой составляющей сигнала.

Для того чтобы не потерять фазу принимаемого сигнала, из сигнала с выхода цифрового фильтра основной избирательности выделяется его синфазная I и квадратурная составляющие. Для этого сигнал умножается на тригонометрические функции $\sin(\omega_{пр}t)$ и $\cos(\omega_{пр}t)$. На выходе умножителя на синусоидальную функцию формируется сигнал, описываемый следующей формулой:

$$u_{вых}(t) = (I(t) \times \cos(\omega_0 t) + Q(t) \times \sin(\omega_0 t)) \times \sin(\omega_0 t) =$$

$$= Q(t) \times \sin^2(\omega_0 t) + \frac{1}{2} I(t) \times \sin(2\omega_0 t) =$$

$$= \frac{1}{2} Q(t) - \frac{1}{2} I(t) \times \cos(2\omega_0 t) + \frac{1}{2} I(t) \times \sin(2\omega_0 t)$$

После пропускания этого сигнала через цифровой фильтр низкой частоты на его выходе остается сигнал квадратурной составляющей входного сигнала.

На выходе умножителя на косинусоидальную функцию формируется сигнал, описываемый следующей формулой:

$$u_{вых}(t) = (I(t) \times \cos(\omega_0 t) + Q(t) \times \sin(\omega_0 t)) \times \cos(\omega_0 t) =$$

$$= I(t) \times \cos^2(\omega_0 t) + \frac{1}{2} Q(t) \times \sin(2\omega_0 t) =$$

$$= \frac{1}{2} I(t) + \frac{1}{2} I(t) \times \cos(2\omega_0 t) + \frac{1}{2} Q(t) \times \sin(2\omega_0 t)$$

Этот сигнал тоже пропускается через фильтр низких частот с точно такой же частотной характеристикой. На выходе этого фильтра остается сигнал синфазной составляющей входного сигнала.

Структурная схема квадратурного демодулятора, реализованного в цифровом виде, приведена на рис. 17.3.

Для формирования сигналов синуса и косинуса принимаемой частоты обычно используется цифровой генератор, описанный в *главе 16* этой книги.

После ограничения преобразованного по частоте сигнала по спектру появляется возможность уменьшить частоту его дискретизации. Поэтому на выходе фильтров низкой частоты ставятся дециматоры. Обычно операции децимации и фильтрации удобно выполнять в одном устройстве. Такие устройства получили название децимирующих фильтров.

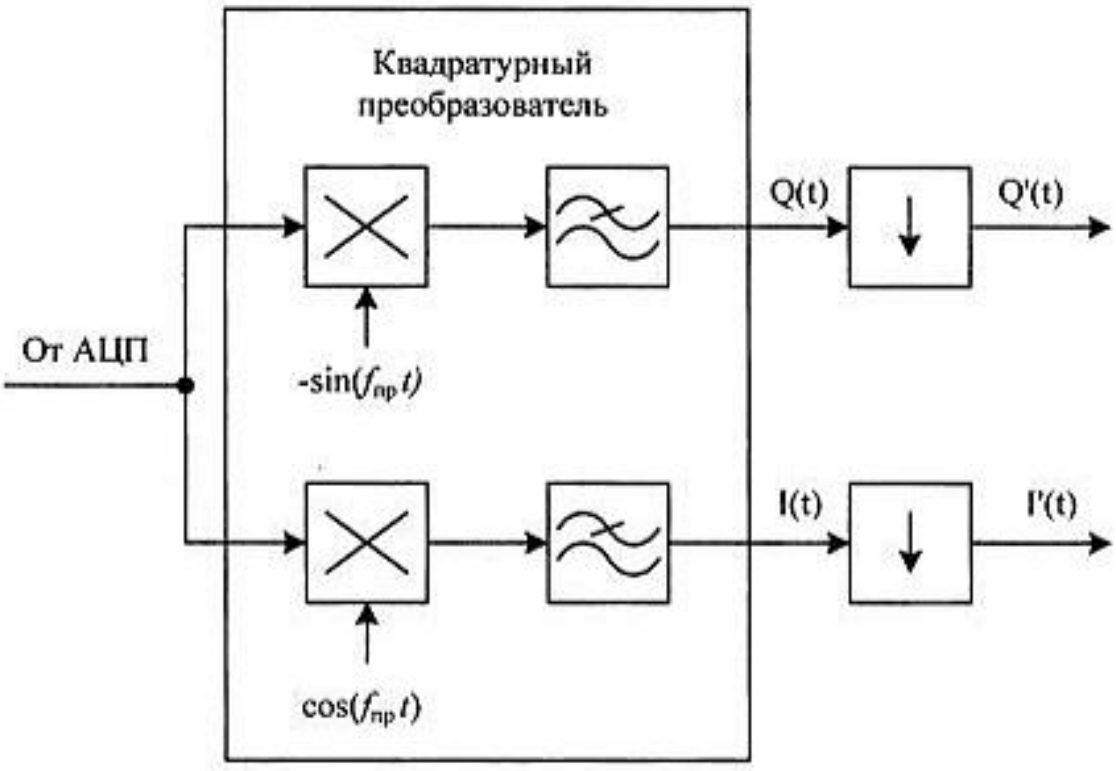


Рис. 17.3. Структурная схема квадратурного демодулятора

Децимирующие фильтры

Децимирующий фильтр предназначен для уменьшения частоты дискретизации обрабатываемого сигнала. Формально это можно было бы сделать, просто передавая на выход схемы каждый пятый или каждый второй отсчет входного сигнала. Устройство, выполняющее данную задачу, называется дециматором.

Задача усложняется тем, что сигнал на входе дециматора не должен содержать спектральных составляющих в полосе образов выходного полезного сигнала. Поэтому прежде чем выбрасывать лишние отсчеты входного сигнала, его следует ограничить по полосе.

Существует ряд факторов, которые приводят к тому, что задача реализации фильтра-дециматора является трудной. Первое это то, что входной поток данных поступает на вход этого фильтра с очень высокой скоростью. Фильтр должен выполнять вычисления в реальном времени. При этом скорость выполнения математических операций в цифровом фильтре должна быть в несколько раз выше скорости поступления отсчетов сигнала, поступающих на его вход.

Частоты в полосе за пределами рабочей полосы сигнала должны быть подавлены до заданного уровня, определяемого динамическим диапазоном полезного сигнала. При этом в полосе рабочего сигнала фильтр-дециматор не должен вносить амплитудных или частотных искажений. Кроме того, структура фильтра должна быть простой и он (фильтр-дециматор) должен легко реализовываться в интегральном исполнении.

Децимирующий фильтр с конечной импульсной характеристикой

На рис. 17.4 приведен пример амплитудно-частотной характеристики децимирующего фильтра. Этот фильтр способен подавить мешающие сигналы в полосе трех высокочастотных образов полезного сигнала.

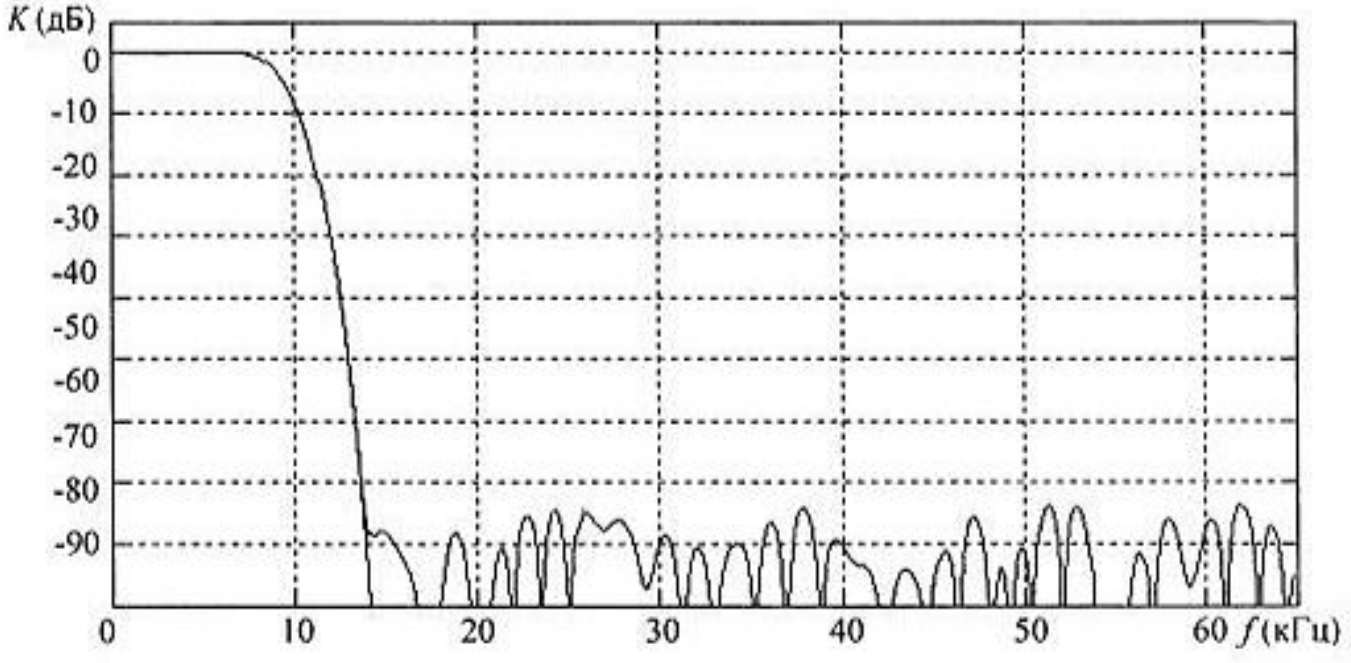


Рис. 17.4. Амплитудно-частотная характеристика децимирующего фильтра

То, что фильтр пропускает на выход только четвертую часть входного спектра, означает, что на выходе подобного фильтра можно снизить частоту дискретизации сигнала в четыре раза.

Однородный децимирующий фильтр

Наиболее просто в цифровом виде реализовать рассмотренный нами ранее однородный фильтр, т. к. для его реализации не требуются умножители. Как было показано в *главе 16*, в таком случае для реализации фильтра потребуется два каскада. Так как фильтр является линейным устройством, то операции интегрирования и фильтра с конечной импульсной характеристикой можно поменять местами. Первый каскад будет выполнять интегрирование, а второй — это фильтр с конечной импульсной характеристикой всего с двумя ненулевыми коэффициентами, равными единице. Такая перестановка позволит в дальнейшем дополнительно упростить схему однородного фильтра. Структурная схема нового фильтра приведена на рис. 17.5.

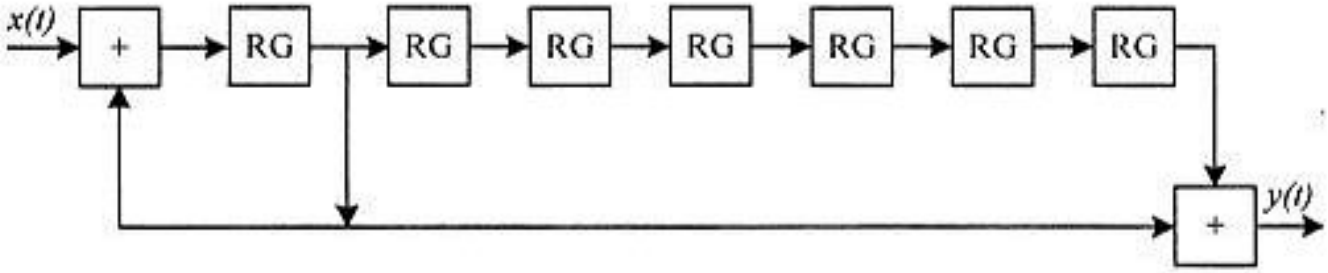


Рис. 17.5. Структурная схема двухкаскадного фильтра, эквивалентного фильтру, приведенному на рис. 16.15

В этой схеме максимальное время задержки сигнала определяется быстродействием сумматора и временем записи в регистр. Мы увеличили быстродействие почти в семь раз.

Ну а теперь вспомним, для чего нам потребовался фильтр. Правильно. Для уменьшения количества отсчетов в единицу времени. Так как операция децимации линейна, то вторую часть фильтра мы можем перенести на выход дециматора.

При таком схемном решении для формирования того же самого значения времени задержки нам потребуется только один регистр, т. к. на его вход тактовой синхронизации будет поступать частота в шесть раз меньше, чем частота синхронизации первого регистра. Получившаяся в результате всех преобразований структурная схема фильтра-дециматора приведена на рис. 17.6.

Новая схема содержит всего два регистра и два двоичных сумматора, т. е. данная схема получилась в три раза проще схемы однородного фильтра, при-

веденной на рис. 16.15. Получившийся в результате преобразований фильтр трудно назвать однородным, однако для того, чтобы отобразить особенности его импульсной и амплитудно-частотной характеристик, сохраним название "однородный" и для этого фильтра.

Если по техническому заданию требуется еще больший коэффициент децимации по сравнению с рассмотренным выше случаем, то выигрыш при реализации однородного фильтра-дециматора по схеме, приведенной на рис. 17.6, будет еще большим.

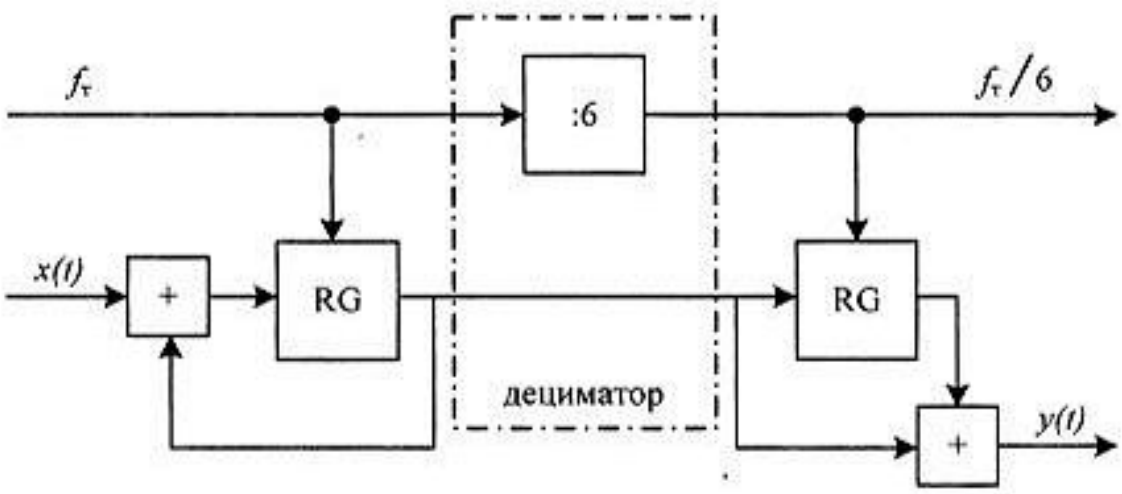


Рис. 17.6. Структурная схема фильтра-дециматора, эквивалентного фильтру, приведенному на рис. 17.5

Хотелось бы напомнить, что при анализе характеристик однородного фильтра для получения приемлемого уровня подавления мешающего сигнала нам потребовалось включить друг за другом несколько каскадов.

Давайте включим последовательно друг за другом три фильтра-дециматора, как это показано на структурной схеме однородного фильтра, приведенной на рис. 17.7.

На рис. 17.8 показана амплитудно-частотная характеристика четырехкаскадного однородного фильтра. ✧ *Обратите внимание*, что образ полезного сигнала сосредоточен около выходной частоты дискретизации.

Проанализировав амплитудно-частотную характеристику четырехкаскадного однородного фильтра можно определить, что этот фильтр обладает максимальным подавлением мешающих сигналов именно в полосе частот высокочастотных образов полезного сигнала. Четырехкаскадный однородный фильтр может обеспечить подавление мешающих сигналов, находящихся в зоне высокочастотных образов полезного сигнала до 90 дБ, что вполне достаточно для реализации 16-разрядного представления полезного сигнала.

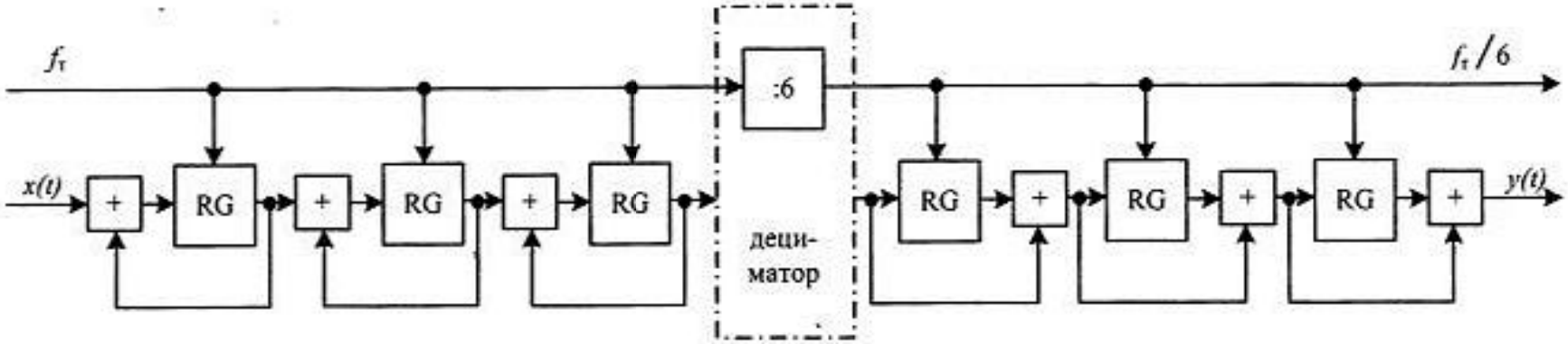


Рис. 17.7. Структурная схема фильтра-дециматора, эквивалентного фильтру, приведенному на рис. 17.6

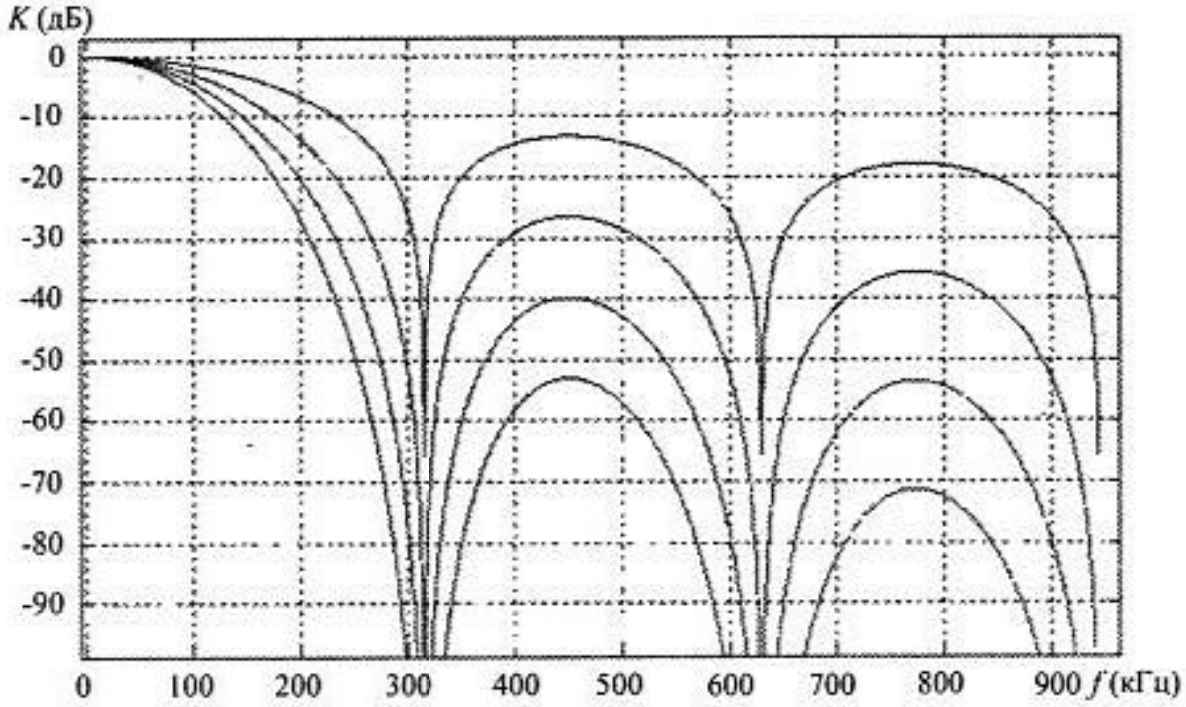


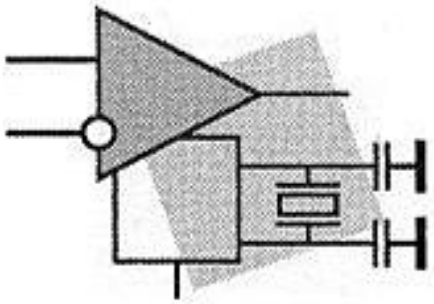
Рис. 17.8. Амплитудно-частотная характеристика четырехкаскадного однородного фильтра-дециматора

Итоги

В этой главе мы кратко познакомились с принципами реализации радиоприемников непосредственно в цифровом виде. Сейчас уже нет сомнений, что в ближайшее время радиоприемные устройства можно будет выполнить полностью в цифровом виде. При этом полученные уже сейчас характеристики цифровых радиоприемных устройств превосходят характеристики аналоговой реализации. Ток потребления получается пока что немного большим аналоговых эквивалентов, однако по мере развития элементной базы и этот параметр вскоре будет уменьшен.

В данной главе цифровые блоки были представлены в основном в виде структурных схем, однако они легко могут быть превращены в принципиальные схемы, и мы ознакомились, как это можно сделать, воспользовавшись знаниями, полученными в первых главах. Реализовать их можно, воспользовавшись ПЛИС, как это показывается, например, на сайтах **http://leso.sibsutis.ru** или **http://digital.sibsutis.ru**. Кроме того, в настоящее время ряд фирм предлагает готовые микросхемы, выполненные по рассмотренным в этой главе принципам.

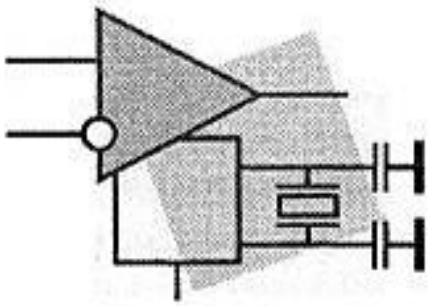
Особенности расчета радиоприемных устройств не рассматривались, т. к. данная книга не является учебным пособием по радиоприемным устройствам, а иллюстрирует возможности цифровой техники, тем не менее, рассчитав приемник по известным методикам, можно затем реализовать его в цифровом виде.



ЧАСТЬ IV

Микропроцессоры

ГЛАВА 18



Принципы работы микропроцессора

В предыдущей части книги были рассмотрены принципы работы цифровых микросхем. Основной особенностью построения цифровых устройств является то, что алгоритм работы устройства заложен в его принципиальной схеме. Кроме того, цифровые микросхемы обладают огромным быстродействием. Это позволяет производителям микросхем обеспечить их пригодность для выполнения различных задач. Для достижения такого быстродействия в ряде случаев приходится увеличивать потребляемый этими микросхемами ток. В то же самое время в большинстве случаев от микросхемы не требуется предельное быстродействие. Возникает противоречие.

Это противоречие было разрешено с появлением КМОП-микросхем. Как это уже было показано в предыдущих главах, если в этих схемах уменьшается скорость переключения внутренних вентилей микросхемы, то ток, потребляемый микросхемой, снижается. Обычно зависимость тока потребления КМОП-микросхемы от частоты носит линейный характер. Иными словами, ток, потребляемый ею от источника питания, увеличивается в два раза при удвоении тактовой частоты устройства.

Другой вопрос, который возникает при построении цифрового устройства, состоит в том, что стоимость микросхемы зависит от ее быстродействия, и чем более быстродействующая схема, тем выше ее стоимость. Как уже упоминалось ранее, при решении ряда задач предельного быстродействия от микросхемы не требуется, поэтому возникает закономерный вопрос — а нельзя ли одну и ту же микросхему использовать многократно для реализации нескольких узлов цифрового устройства? В результате такого подхода для построения всего цифрового устройства можно было бы воспользоваться одной-единственной микросхемой!

Именно такое решение используется в микропроцессорной технике. В составе цифрового устройства применяется одна универсальная микросхема —

микропроцессор, которая последовательно реализует все части алгоритма, по которому должно функционировать это цифровое устройство. Более того! В ряде случаев на одной микросхеме удается реализовать несколько независимых цифровых устройств, главное, чтобы хватило быстродействия микропроцессора и его внешних портов.

В микропроцессорных системах обычно в явном виде выделяются: устройство обработки данных (операционный блок), устройства хранения данных и устройства ввода-вывода информации. Операционный блок предназначен для выполнения команд, т. е. реализует операции обработки данных. В предыдущих главах для построения отдельных цифровых узлов нам потребовались устройства, позволяющие хранить достаточно большие объемы информации, мы рассмотрели внутреннее устройство последовательных и параллельных портов. Единственным неизвестным нам устройством, необходимым для построения микропроцессорной системы, является операционный блок.

Однако прежде чем рассмотреть этот блок, давайте научимся представлять данные в двоичном виде и немного поучимся считать. ✧ *Обратите внимание*, что все дальнейшие примеры будут приведены именно в двоичном виде. Это связано с тем, что в такой форме выполняет обработку данных цифровая аппаратура, а значит, так легче будет понять принципы ее работы. Здесь не будет использоваться шестнадцатеричная или восьмеричная форма записи двоичного кода. Эти формы записи двоичного числа удобны своею краткостью, но для лучшего понимания принципов работы микропроцессора удобней использовать именно двоичную запись.

Вы можете возразить: но ведь мы уже рассматривали системы счисления в предыдущих главах! Все верно. Однако в реальных цифровых устройствах обработки данных присутствуют свои особенности. Например, математические операции над двоичными числами всегда выполняются с фиксированным количеством двоичных разрядов. В математическом представлении систем счисления кроме цифр ‘0’ и ‘1’ присутствуют и другие символы, такие как ‘–’, ‘+’, двоичная запятая и основание системы ‘2’, а мы в памяти микропроцессора можем хранить только символы ‘0’ и ‘1’! Кроме того, при обработке данных в цифровом устройстве очень невыгодно использовать операцию вычитания, и это тоже накладывает свои особенности на вид данных, запоминаемых в устройстве хранения двоичной информации.

Виды двоичных кодов

В микропроцессорах двоичные коды используются для представления любых обрабатываемых данных: чисел, текста, команд и т. д. При этом разрядность двоичных чисел может превышать разрядность внутренних регистров самого

процессора и ячеек используемой памяти. В таком случае длинное двоичное число может занимать несколько ячеек памяти и обрабатываться несколькими командами процессора. Подчеркнем, что все ячейки памяти, выделенные под многобайтное число, рассматриваются в этом случае как одно число.

Для представления числовых данных в микропроцессоре могут использоваться знаковые и беззнаковые коды. Для определенности примем разрядность процессора (и разрядность запоминающих ячеек памяти) равной восьми битам, и в последующих примерах будем рассматривать именно такие числа.

Беззнаковые двоичные коды

Первый вид двоичных кодов, который мы рассмотрим, используется для представления целых беззнаковых чисел. Это простейший вид двоичного кода. В нем каждый двоичный разряд представляет собой степень цифры 2. Формат 8-разрядного беззнакового двоичного кода приведен на рис. 18.1.

Рис. 18.1. Формат 8-разрядного беззнакового двоичного кода

На этом рисунке над каждым разрядом беззнакового двоичного кода приведено значение его веса. При этом минимально возможное число, которое можно записать таким двоичным кодом, равно 0. Максимально возможное число, которое можно представить этим кодом, определяется следующей формулой:

$$M = 2^n - 1,$$

где n — разрядность двоичного числа. Разрядность числа обычно выбирают кратной разрядности микропроцессора.

Эти два числа полностью определяют диапазон значений чисел, которые можно представить двоичным кодом. В случае двоичного 8-разрядного беззнакового двоичного кода целые числа, которые можно записать с его помощью, находятся в диапазоне от 0 до 255. Восьмиразрядное двоичное число обычно называют байтом, иногда — октетом или полусловом.

Для беззнакового двоичного 16-разрядного кода диапазон представляемых значений будет от 0 до 65 535. В микропроцессорной системе, построенной на 8-разрядном процессоре, для хранения 16-разрядного числа используются две ячейки памяти, расположенные в соседних адресах.

В качестве примера записи 16-разрядного числа в двух соседних 8-разрядных ячейках памяти на рис. 18.2 приведена запись числа $56249_{10} = 1101101110111001_2$.

Рис. 18.2. Формат 16-разрядного беззнакового двоичного кода, записанного в двух соседних ячейках памяти

Подобным же образом можно записать 24- или 32-разрядное число. В результате, разрядность обрабатываемых чисел практически не зависит от разрядности операционного блока микропроцессора. Она ограничивается только объемом его оперативной памяти.

В качестве примера записи 24-разрядного числа в 8-разрядных ячейках памяти, на рис. 18.3 приведена запись числа $14414009_{10} = 110110111111000010111001_2$.

Рис. 18.3. Формат 16-разрядного беззнакового двоичного кода, записанного в двух соседних ячейках памяти

При работе с числами, занимающими несколько ячеек памяти, микропроцессор обрабатывает каждый байт отдельно. Для учета переноса между младшими, старшими или промежуточными байтами многоразрядного двоичного числа используются специальные команды микропроцессора, учитывающие перенос, формируемый предыдущей командой. Описанный алгоритм работы с многобайтовыми числами иллюстрируется рис. 18.4.

Следует отметить, что, в отличие от действий в математике, при записи числа в прямом двоичном коде не существует возможности не записывать старшие незначащие нули. Поэтому число 12_{10} не может быть записано как 1100. Оно обязательно должно быть записано во всех восьми разрядах кода: 00001100. Если число первоначально было определено как 16-разрядное, то в этом случае это же самое число 12_{10} должно быть записано как 0000000000001100. В качестве еще одного примера рассмотрим запись числа 31_{10}. Оно будет за-

писано в 8-разрядном прямом беззнаковом двоичном коде как 00011111, а в 16-разрядном — 0000000000011111.

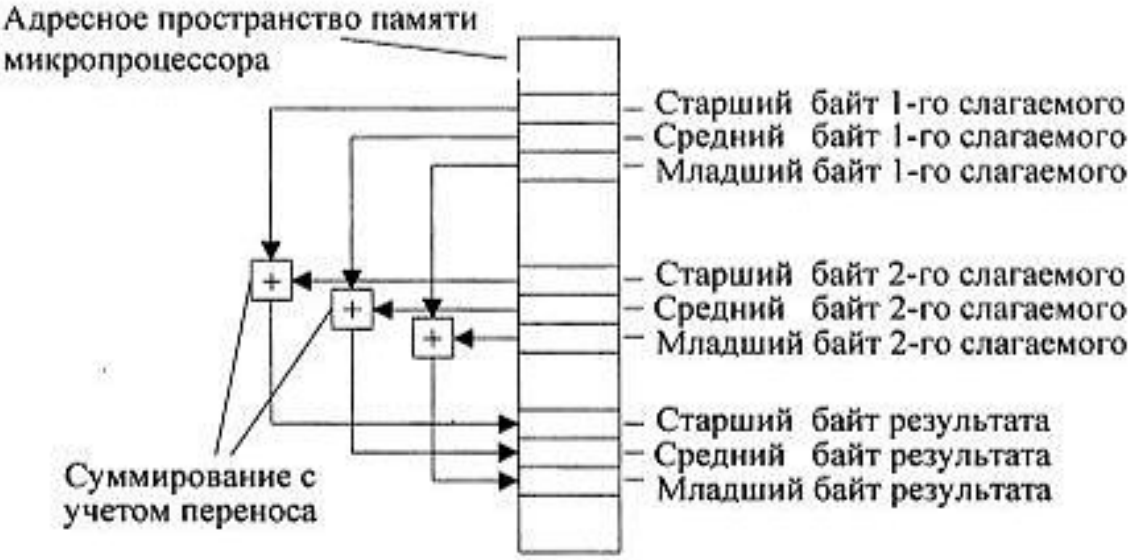


Рис. 18.4. Суммирование многоразрядного числа

Прямые знаковые двоичные коды

Второй вид двоичных кодов, который мы рассмотрим, — это прямые целые знаковые коды. В этих кодах старший разряд используется для представления знака числа. В прямом знаковом коде нулем обозначается знак '+', а единицей — знак '–'. В результате введения знакового разряда диапазон чисел, представляемых двоичным кодом, смещается в сторону отрицательных чисел. Формат 8-разрядного прямого знакового двоичного кода приведен на рис. 18.5. На рисунке приведено шесть различных чисел, записанных в этом коде. Слева указаны десятичные числа, соответствующие двоичному представлению обратного кода, а сверху, над каждым разрядом двоичного кода, указан его вес.

Диапазон 8-разрядных целых чисел, которые можно записать, пользуясь таким кодом, простирается от –127 до +127. Для 16-разрядного прямого знакового двоичного кода диапазон чисел составит от –32 767 до +32 767. В 8-разрядном процессоре для хранения такого числа используются две ячейки памяти, расположенные в соседних адресах.

✧ *Обратите внимание*, что при считывании содержимого памяти микропроцессора мы ни по каким внешним признакам не сможем отличить знаковый двоичный код от беззнакового! Это сможет сделать только программа, в которой заложена информация о том, в каких ячейках памяти какой код следует использовать для хранения чисел. В результате человек или программа, не обладающие подобной информацией, при попытке прочитать ячейки памяти

не смогут узнать — какое же число записано в данных конкретных ячейках памяти.

Рис. 18.5. Формат 8-разрядного прямого знакового двоичного кода

Недостатком прямого знакового кода является то, что знаковый разряд и цифровые разряды приходится обрабатывать раздельно. Алгоритм программ, работающих с такими двоичными кодами, получается сложным. Для выделения и изменения знакового разряда приходится применять механизм маскирования разрядов, что резко увеличивает размер программы и уменьшает ее быстродействие. Для того чтобы алгоритм обработки знакового и цифровых разрядов при выполнении операций суммирования и вычитания не различался, были введены обратные двоичные коды.

Знаковые обратные двоичные коды

Обратные двоичные коды отличаются от прямых только тем, что отрицательные числа в них получаются инвертированием всех разрядов положительного числа. При этом обработка знакового и цифровых разрядов не различаются. Алгоритм работы с такими кодами резко упрощается.

Формат 8-разрядного обратного знакового двоичного кода приведен на рис. 18.6. На рисунке приведено шесть различных чисел, записанных в этом коде. Слева указаны десятичные числа, соответствующие двоичному представлению обратного кода.

✧ *Обратите внимание*, что знак числа при инвертировании получается автоматически. При инвертировании положительного числа, в котором знак '+' обозначен логическим нулем, мы получаем логическую единицу, т. е. знак '–'.

В этом коде точно так же, как и в прямом знаковом двоичном коде, можно записывать как 8-разрядные, так и 16- или 32-разрядные двоичные числа. При этом потребуется использовать несколько ячеек оперативного запоми-

нающего устройства, как это иллюстрировалось на рис. 18.2 и 18.3 для прямого беззнакового двоичного кода.

Рис. 18.6. Формат 8-разрядного обратного знакового двоичного кода

Тем не менее, при работе с обратными кодами все еще требуется специальный алгоритм распознавания знака, вычисления абсолютного значения числа и восстановления знака результата числа. Кроме того, в прямом и обратном кодах для представления числа 0 используется два разных кода, в то время как из школьного курса математики известно, что число 0 является положительным числом и отрицательным не может быть никогда.

Знаковые дополнительные двоичные коды

От перечисленных недостатков свободны дополнительные коды. Они позволяют суммировать положительные и отрицательные числа, не анализируя знаковый разряд, и при этом получать правильный результат. Все это становится возможным благодаря тому, что дополнительные числа являются естественным кольцом чисел, а не искусственным образованием, каким являются прямые и обратные коды. Кроме того, немаловажным является то обстоятельство, что вычислять дополнение в двоичном коде чрезвычайно легко. Для этого достаточно к обратному коду добавить единицу. Формат 8-разрядного дополнительного знакового двоичного кода представлен на рис. 18.7. Здесь же приведено шесть различных чисел, записанных в этом коде.

Числа, которые можно представлять 8-разрядным дополнительным двоичным кодом, находятся в диапазоне от –128 до +127. Для 16-разрядного кода этот диапазон будет от –32 768 до +32 767. В 8-разрядном процессоре для хранения 16-разрядного числа используется две ячейки памяти, расположенные в соседних адресах.

В обратных и дополнительных кодах наблюдается интересная особенность, которая называется эффектом распространения знака: при преобразовании

однобайтного числа в двухбайтное достаточно всем битам старшего байта присвоить значение знакового бита исходного байта. То есть для хранения знака числа можно использовать сколько угодно старших битов. При этом значение кода совершенно не изменяется. Эффект распространения знака используется при подключении таких устройств, как АЦП или ЦАП к микропроцессору, если их разрядности не совпадают.

Рис. 18.7. Формат 8-разрядного дополнительного знакового двоичного кода

Использование для представления знака числа двух битов предоставляет интересную возможность контролировать возникновение переполнения при выполнении арифметических операций. В качестве второго знакового бита обычно используется флаг переноса C. Можно, конечно, использовать и большее количество знаковых битов, но это никаких дополнительных преимуществ не дает. Рассмотрим несколько примеров работы с дополнительными двоичными кодами.

1. Просуммируем числа +12 и +5. Суммирование этих чисел в двоичном и десятичном представлении приведено на рис. 18.8.

Рис. 18.8. Суммирование чисел +12 и +5

В этом примере видно, что в результате суммирования получается правильный результат. Это можно проконтролировать по флагу переноса C, который совпадает со знаком результата (эффект распространения знака действует).

2. Просуммируем два отрицательных числа –12 и –5. Суммирование этих чисел в двоичном и десятичном представлении приведено на рис. 18.9.

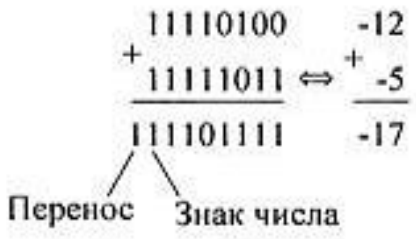


Рис. 18.9. Суммирование чисел –12 и –5

В этом примере флаг переноса C тоже совпадает со знаком результата, т. е. переполнения не произошло и в этом случае.

3. Просуммируем отрицательное и положительное числа –12 и +5. Суммирование этих чисел в двоичном и десятичном представлении приведено на рис. 18.10.

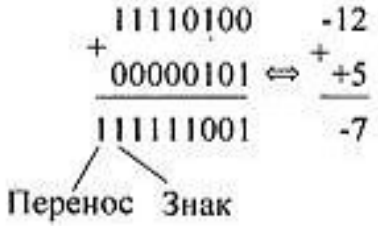


Рис. 18.10. Суммирование чисел –12 и +5

В этом примере при суммировании положительного и отрицательного числа автоматически получается правильный знак результата. В данном случае знак результата отрицательный. Флаг переноса совпадает со знаком результата, поэтому мы делаем вывод, что переполнения не было (мы можем убедиться в этом непосредственными вычислениями на бумаге или при помощи калькулятора).

4. Просуммируем положительное и отрицательное число +12 и –5. Суммирование этих чисел в двоичном и десятичном представлении приведено на рис. 18.11.

Рис. 18.11. Суммирование чисел +12 и –5

В данном примере знак результата положительный. Флаг переноса совпадает со знаком результата, поэтому переполнения не было и в этом случае.

5. Просуммируем числа 100 и 31. Суммирование этих чисел в двоичном и десятичном представлении приведено на рис. 18.12.

Рис. 18.12. Суммирование чисел +100 и +31

В этом примере видно, что в результате суммирования произошло переполнение 8-битовой переменной, т. к. в результате операции над положительными числами получился отрицательный результат. Если рассмотреть флаг переноса C, то он не совпадает со знаком результата. Эта ситуация является признаком переполнения результата и легко обнаруживается при помощи операции "исключающее ИЛИ" над старшим битом результата и флагом переноса C. Большинство процессоров осуществляют эту операцию аппаратно и помещают результат во флаг переполнения OV.

6. Просуммируем числа –100 и –31. Суммирование этих чисел в двоичном и десятичном представлении приведено на рис. 18.13.

Рис. 18.13. Суммирование чисел –100 и –31

В этом примере при выполнении операции над отрицательными числами в результате суммирования произошло переполнение 8-битовой переменной, т. к. получился положительный результат. И в этом случае если рассмотреть флаг переноса C, то он не совпадает со знаком результата. Отличие от предыдущего случая только в комбинации этих битов. В примере 5 говорят о переполнении результата (комбинация 01), а в примере 6 — об антипереполнении результата (комбинация 10).

Представление рациональных чисел в двоичном коде с фиксированной запятой

Кроме целых чисел, в процессе вычислений часто требуется работать с рациональными числами. Чаще всего это необходимо при цифровой обработке сигналов в сигнальных процессорах или в микроконтроллерах. Как и в случае целых чисел, рациональные числа могут быть беззнаковыми и знаковыми. Для двоичного представления знаковых рациональных чисел могут быть использованы прямые, обратные и дополнительные коды. Принцип их построения точно такой же, как и в случае целых чисел.

Рассмотрим, как можно записать рациональное число. Ранее, рассматривая целые числа, мы предполагали, что в двоичном числе запятая, разделяющая целую и дробную части, находится правее самого младшего разряда. Но кто сказал, что она должна всегда находиться именно в этом месте? Мы можем договориться, что запятая, разделяющая целую и дробную части двоичного числа, находится слева от самого старшего разряда, и тогда в подобной переменной можно будет записывать только дробные числа, меньшие $1,0_{10}$. Формат 8-разрядного дробного беззнакового двоичного кода представлен на рис. 18.14. Здесь же приведено два числа, записанных в этом коде.

Рис. 18.14. Формат 8-разрядного дробного беззнакового двоичного кода

Или договоримся, что двоичная запятая находится точно посередине числа, и тогда мы сможем записывать числа, содержащие как целую, так и дробную части. Формат такого восьмиразрядного беззнакового двоичного кода представлен на рис. 18.15. Здесь же приведено два числа, записанных в этом коде.

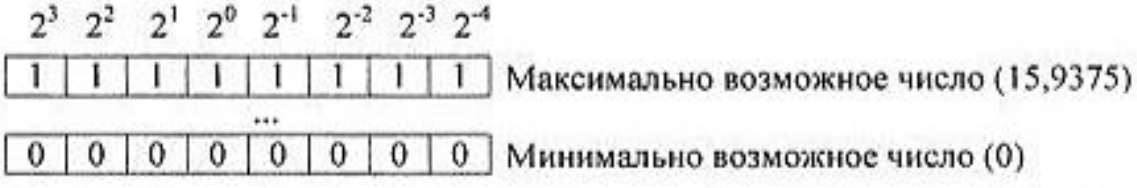


Рис. 18.15. Формат 8-разрядного смешанного беззнакового двоичного кода

Остальные виды двоичных кодов, используемых для представления чисел с фиксированной запятой, рассматривать не будем. Они строятся точно так же, как и для целых чисел.

Представление рациональных чисел в двоичном коде с плавающей запятой

Часто приходится обрабатывать очень большие числа (например, расстояние между звездами) или, наоборот, очень маленькие числа (например, размеры атомов или электронов). При таких вычислениях пришлось бы использовать числа с фиксированной запятой очень большой разрядности. В то же время нам не нужно знать расстояние между звездами с точностью до миллиметра. Для вычислений с такими величинами числа с фиксированной запятой неэффективны.

В десятичной арифметике в таких случаях число записывается в виде мантиссы, умноженной на 10 в степени, отображающей порядок числа, например:

2×10^{5}; $1{,}6\times10^{-38}$.

В алгебре такое представление рациональных чисел называют стандартным видом числа. В двоичной арифметике тоже используется похожая форма записи чисел — представление с плавающей запятой (часто также называемое представлением с плавающей точкой).

А теперь рассмотрим промышленные стандарты, используемые для представления чисел с плавающей запятой в компьютерах. Существует стандарт IEEE 754 для представления чисел с одинарной точностью (float) и с двойной точностью (double). Для записи числа в формате с плавающей запятой одинарной точности требуется 32-битовое слово. Для записи чисел с двойной точностью требуется 64-битовое слово. Чаще всего числа хранятся в нескольких соседних ячейках памяти процессора. Форматы одинарной точности и удвоенной точности числа с плавающей запятой приведены на рис. 18.16.

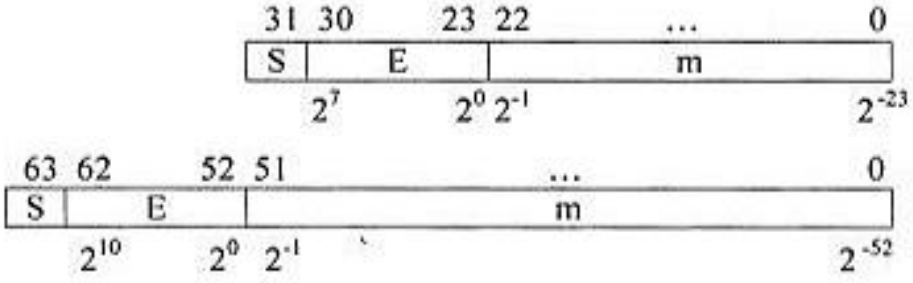


Рис. 18.16. Форматы чисел с плавающей запятой

На рис. 18.16 над полями числа с плавающей запятой показан номер двоичного разряда, а внизу двоичный вес каждого разряда. При этом буквой S обозначен знак числа, 0 — это положительное число, 1 — отрицательное число,

E обозначает смещенный порядок числа. Смещение требуется, чтобы не вводить в число еще один знак. Смещенный порядок — всегда положительное число. В формате одинарной точности для порядка выделено 8 битов. Для смещенного порядка двойной точности отводится 11 битов. Для формата одинарной точности принято смещение 127, а для формата двойной точности — 1023. В десятичной мантиссе числа стандартного вида старший разряд — это цифра от 1 до 9. Старший разряд двоичной мантиссы — всегда 1. Поэтому для хранения старшей единицы двоичной мантиссы не выделяется отдельный бит. Единица подразумевается, как и запятая, отделяющая дробную часть от целой. Кроме того, в формате чисел с плавающей точкой принято, что мантисса всегда больше 1. То есть значения мантиссы лежат в диапазоне от 1 до 2.

Рассмотрим несколько примеров:

1. Определить число с плавающей запятой, лежащее в четырех соседних байтах:

   11000001 01001000 00000000 00000000.

   - Знаковый бит, равный 1, показывает, что число отрицательное.
   - Экспонента 10000010 в десятичном виде соответствует числу 130. Вычтя число 127 (смещение) из 130, получим число 3.
   - Теперь запишем мантиссу с учетом неявной единицы:
     100 1000 0000 0000 0000 0000 соответствует 1,1001
   - И, наконец, определим десятичное число: $1100{,}1_2 = 12{,}5_{10}$

2. Определить число с плавающей запятой, лежащее в четырех соседних байтах:

   11000011 00110100 00000000 00000000.

   - Знаковый бит, равный 1, показывает, что число отрицательное.
   - Экспонента 10000110 в десятичном виде соответствует числу 134. Вычтя число 127 из 134, получим число 7.
   - Теперь запишем мантиссу: 011 0100 0000 0000 0000 0000 соответствует 1,01101
   - И, наконец, определим десятичное число: $10110100_2 = 180_{10}$

Для того чтобы записать ноль в коде с плавающей запятой, достаточно записать в смещенный порядок число 00000000_2. Значение мантиссы при этом не имеет значения. Число, в котором все байты равны 0, тоже попадает в этот диапазон значений. Именно его обычно и используют для записи нуля в коде с плавающей запятой.

Бесконечность соответствует смещенному порядку 11111111_2 и мантиссе, равной 1,0. При этом существует минус бесконечность и плюс бесконечность (переполнение и антипереполнение), которые часто отображаются на экране дисплея как +INF и –INF.

При таком значении порядка все остальные комбинации битов в мантиссе (в том числе и все единицы) воспринимаются как не числа и отображаются на экране как NaN.

Представление десятичных чисел

Иногда бывает удобно хранить числа в памяти процессора в десятичном виде (например, для вывода на экран дисплея или при финансовых расчетах). Для представления таких чисел используются двоично-десятичные коды. Цифра одного десятичного разряда представляется при помощи четырех двоичных битов, называемых тетрадой. Иногда встречается название, пришедшее из англоязычной литературы: нибл. При помощи четырех битов можно закодировать шестнадцать цифр. Лишние комбинации в двоично-десятичном коде являются запрещенными. Таблица соответствия двоично-десятичного кода и десятичных цифр приведена в табл. 18.1.

***Таблица 18.1.** Таблица соответствия двоично-десятичного кода и десятичных цифр*

Двоично-десятичный код				Десятичный код
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9

Остальные комбинации двоичного кода в тетраде являются запрещенными.

Запишем пример двоично-десятичного кода:

```
1258   =      0001 0010     0101 1000
589    =      0000 0101     1000 1001
```

Достаточно часто в памяти процессора для хранения одной десятичной цифры выделяется одна ячейка памяти (8-, 16- или 32-разрядная). Это делается для повышения скорости работы программы. Для того чтобы отличить такое представление двоично-десятичного числа от стандартного, последнее называют упакованной формой двоично-десятичного числа. Запишем те же числа, что и в предыдущем примере в неупакованном двоично-десятичном коде для восьмиразрядного процессора:

```
1258 = 00000001
       00000010
       00000101
       00001000

589  = 00000000
       00000101
       00001000
       00001001
```

Суммирование двоично-десятичных чисел

Суммирование двоично-десятичных чисел можно производить по правилам обычной двоичной арифметики, а затем производить двоично-десятичную коррекцию, заключающуюся в проверке каждой тетрады на допустимость ее кода. Если в какой-либо тетраде обнаруживается запрещенная комбинация или был перенос в старшую тетраду, то это говорит о переполнении. В этом случае необходимо произвести двоично-десятичную коррекцию. Она заключается в дополнительном суммировании числа шесть (число запрещенных комбинаций) с тетрадой, в которой произошло переполнение. Приведем два примера использования двоично-десятичной коррекции. Просуммируем десятичное число 18, записываемое в двоично-десятичном коде как 0001 1000 и десятичное число 13, двоично-десятичный код 0001 0011. Ожидаемый результат 31. Запишем наши действия в столбик, как это показано на рис. 18.17.

$$\begin{array}{r} 0001\ 1000 \\ +\ 0001\ 0011 \\ \hline 0010\ 1011 \end{array} \Rightarrow \begin{array}{r} 0010\ 1011 \\ +\ 0000\ 0110 \\ \hline 0011\ 0001 \end{array}$$

Рис. 18.17. Суммирование чисел 18 и 13 в двоично-десятичном коде

В результате выполнения двоичного суммирования получим число 0010 1011 ($2B_{16}$). То есть младшая тетрада содержит запрещенную комбинацию. Это означает, что необходимо выполнить десятичную коррекцию. Прибавим

к младшей тетраде код коррекции 6. Эта операция показана на рис. 18.17 в столбике, записанном справа. В результате второго двоичного суммирования получаем результат 31. То есть именно то, что и ожидалось!

Во втором примере просуммируем два десятичных числа 19, записываемых в двоично-десятичном коде как 0001 1001. Ожидаемый результат 38. Запишем наши действия в столбик, как это показано на рис. 18.18.

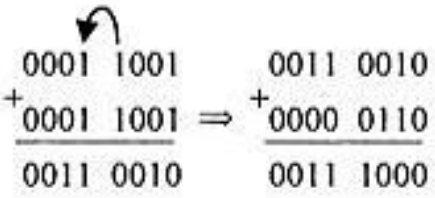


Рис. 18.18. Суммирование чисел 19 в двоично-десятичном коде

В результате выполнения двоичного суммирования получим число 0011 0010 (32). В этом случае запрещенных комбинаций нет. Но зато был перенос в старшую тетраду, а значит, и в этом случае необходимо выполнить десятичную коррекцию. Прибавим к младшей тетраде код коррекции 6. Эта операция показана на рис. 18.18 в столбике, записанном справа. В результате второго двоичного суммирования получаем результат 38. То есть именно то, что и ожидалось! Работа со старшей тетрадой ничем не отличается от работы с младшей тетрадой, рассмотренной в приведенных примерах.

Представление текстовых данных в памяти процессора

Кроме обработки числовых данных, при проектировании цифровых устройств на микропроцессорах часто приходится сталкиваться с текстовой информацией. Это связано с тем, что в конечном итоге получателем информации является человек, и ему легче воспринимать ее в обычном текстовом виде. Эта информация может отображаться как на небольших индикаторных панелях, так и на больших мониторах.

Но ведь, как нам известно, во внутренней памяти микропроцессора мы можем хранить только комбинации логических нулей и единиц! Мы уже убедились, что одна и та же комбинация нулей и единиц в различных двоичных кодах может обозначать совершенно разные числа. Для кодирования текстов тоже можно воспользоваться комбинациями нулей и единиц. Для этого нужно составить таблицу соответствия между комбинациями нулей и единиц и буквами алфавита.

В русском языке содержится 33 буквы, в английском — 28. Казалось бы, для кодирования текстов достаточно шести двоичных разрядов (количество раз-

личных комбинаций пяти двоичных разрядов равно $2^5 = 32$). Однако буквы бывают строчные и прописные (по отношению к клавиатуре компьютера говорят, что буквы бывают верхнего и нижнего регистра). В телеграфном коде для переключения между регистрами используется специальный двоичный код (один из оставшихся от 28 букв). Мы же можем к двоичному коду просто добавить еще один разряд.

Казалось бы, теперь все проблемы решены, но для записи текстов кроме букв требуются дополнительные символы, такие как скобки, звездочки, знаки препинания, кавычки различных видов. В результате необходимое количество двоичных разрядов возрастает до семи. Такое количество двоичных разрядов позволяет закодировать до 128 различных текстовых символов.

Наиболее распространенной является таблица кодирования текста ASCII. Однако в России, да и в большинстве стран мира, обычно используется сразу два языка — английский и национальный. В результате к основной таблице кодирования текста ASCII добавляется дополнительная таблица с национальным алфавитом. Такая таблица называется национальным расширением. Подобные таблицы ASCII, с национальными расширениями, широко применялись в операционной системе DOS и до сих пор используются для записи текстов в цифровых устройствах, построенных на основе микропроцессоров. Таблица ASCII символов приведена в приложении.

В настоящее время в компьютерах в основном применяется операционная система Windows, в которой используется другая таблица соответствия двоичного кода и текстовых символов. Эта таблица называется ANSI. В этих двух таблицах первые 128 строк соответствия символов совпадают. В этой части таблицы содержатся символы цифр, знаков препинания, латинские буквы верхнего и нижнего регистров и управляющие символы. Национальные расширения символьных таблиц и символы псевдографики содержатся в последних 128 кодах этих таблиц, поэтому кодировки русских текстов в DOS и Windows не совпадают. В результате текст, набранный в кодировке ASCII, в Windows выглядит как набор непонятных значков.

Кроме перечисленных двух таблиц кодировки текста, существует еще целый ряд таблиц. Это такие таблицы, как КОИ8-U, КОИ8-R, ДКОИ8, E-mail Cyrillic, и ряд других кодов, которые используются в других видах компьютеров.

В качестве примера в листинге 18.1 приведен фрагмент текстового файла и двоичных кодов, соответствующих ему.

Листинг 18.1. Пример соответствия двоичных кодов, записанных в шестнадцатеричном виде, и текста, соответствующего этим кодам

```
000002A0: CED5D4D8D3D120D5 | D4D2CFCD2C20CBCF   нуться утром, ко
000002B0: C7C4C120D3D7C5D4 | C9D420D3CFCCCED9   гда светит солны
```

```
000002C0: DBCBCF2E20F7CFC4 | C120D720D3CFD3C5   шко. Вода в сосе
000002D0: C4CEC9C820D2D5DE | C5CACBC1C820DAC1   дних ручейках за
000002E0: CDC5D2DACCC12E0D | 0AF3D4D2C1CECECF   мерзла.♪◙Странно
000002F0: 2C20C120D7DEC5D2 | C120D7C5DEC5D2CF   , а вчера вечеро
00000300: CD20DAC4C5D3D820 | CCC5C420D7D2CFC4   м здесь лед врод
```

Этот же участок текста будет отображаться на экране дисплея следующим образом:

```
нуться утром, когда светит солнышко. Вода в соседних ручейках замерзла.
Странно, а вчера вечером здесь лед врод
```

В этом листинге для записи текста использован код КОИ8-R. В столбце слева приведен адрес конкретного байта, затем идет шестнадцатеричная запись содержимого восьми соседних байтов, через вертикальную черту запись содержимого следующих восьми байтов, и в конце строки листинга приводится их соответствие тексту. Например, первый байт, расположенный по адресу 2A0, содержит код CE, означающий букву ‘н’. Следующий байт, расположенный по адресу 2A1, содержит код D5, означающий букву ‘у’.

Знаки ♪◙ в пятой строке служат для обозначения конца строки и перехода на следующую. Это служебные символы `0D` и `0A`.

Для записи текстов в микропроцессорных системах обычно используется та кодовая таблица записи текстов, что применяется в операционной системе, где ведется написание программы для этой микропроцессорной системы.

На этом можно завершить обзор возможных видов двоичных кодов. При необходимости вы сможете самостоятельно освоить новые виды этих кодов. Главное, что необходимо вынести из этого материала, это то, что одни и те же комбинации логических единиц и нулей могут представлять собой совершенно разную информацию. Микропроцессор же рассматривает их просто как двоичные числа. Например, он может просто суммировать между собой различные буквы, может просуммировать или вычесть букву и число и т. д.

Теперь, после того как мы научились работать с двоичными кодами, можно перейти к устройствам, которые могут выполнять различные операции над этими числами — суммировать, вычитать, увеличивать и уменьшать на единицу. При этом выбор выполняемой операции желательно выполнять также при помощи двоичного кода. Такое устройство получило название арифметического устройства. Если же оно, кроме арифметических операций, выполняет еще и логические, то его называют арифметико-логическим устройством (АЛУ).

Арифметико-логические устройства

Ранее были рассмотрены схемы, осуществляющие суммирование многоразрядных кодов. Однако часто требуется осуществлять не только суммирование, но и вычитание двоичных кодов. Двоичные коды, при помощи которых можно записывать отрицательные числа, уже рассматривались в предыдущих разделах. Там же было показано, что при использовании дополнительных кодов операцию вычитания двух положительных чисел можно заменить операцией суммирования положительного и отрицательного числа, при этом получение двоичного отрицательного числа из положительного является элементарной операцией. Для получения кода отрицательного числа необходимо проинвертировать его и прибавить к полученному результату число 1.

Принципиальная схема вычитателя числа A из числа B приведена на рис. 18.19, а схема вычитателя числа B из числа A — на рис. 18.20. В этих схемах прибавление единицы к проинвертированному числу осуществляется подачей уровня логической единицы на вход переноса PI схемы сумматора. Основным элементом этих двух схем является арифметический сумматор. Различаются они лишь местом включения инверторов.

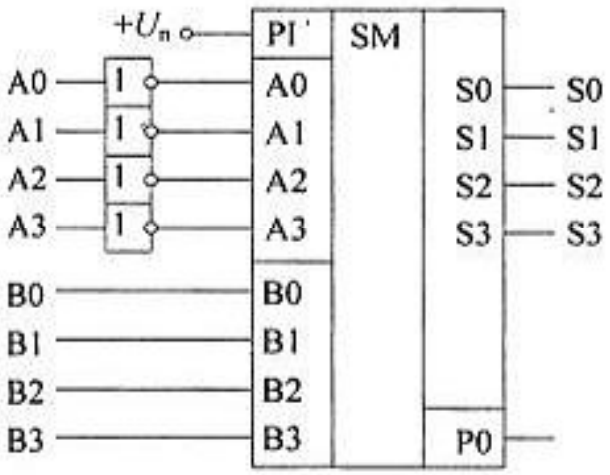


Рис. 18.19. Принципиальная схема вычитателя числа A из числа B

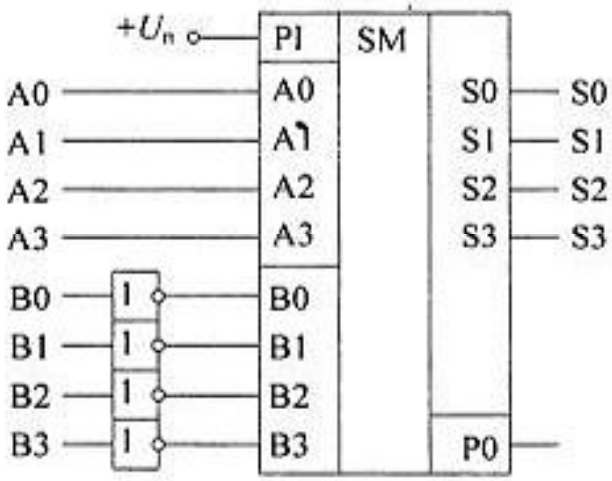


Рис. 18.20. Принципиальная схема вычитателя числа B из числа A

Если же в процессе вычислений потребуется изменять арифметическую операцию, то в схему следует ввести коммутатор, который будет изменять ее внутреннюю структуру в зависимости от выполняемой арифметической операции. Такое устройство получило название арифметического устройства.

Структурная схема арифметического устройства, способного суммировать, вычитать, превращать положительное число в отрицательное или просто передавать на выход число с одного или другого входа, показана на рис. 18.21. По приведенной на этом рисунке структурной схеме можно легко получить

полную принципиальную схему, поэтому, для упрощения анализа, все дальнейшие рассуждения будем производить именно по ней.

В приведенной на рис. 18.21 схеме используются четырехвходовые мультиплексоры, для управления каждым из которых достаточно двух битов. Это означает, что для управления всей схемой в целом достаточно четырех сигналов управления. Попытаемся построить таблицу операций, которые будет выполнять эта схема.

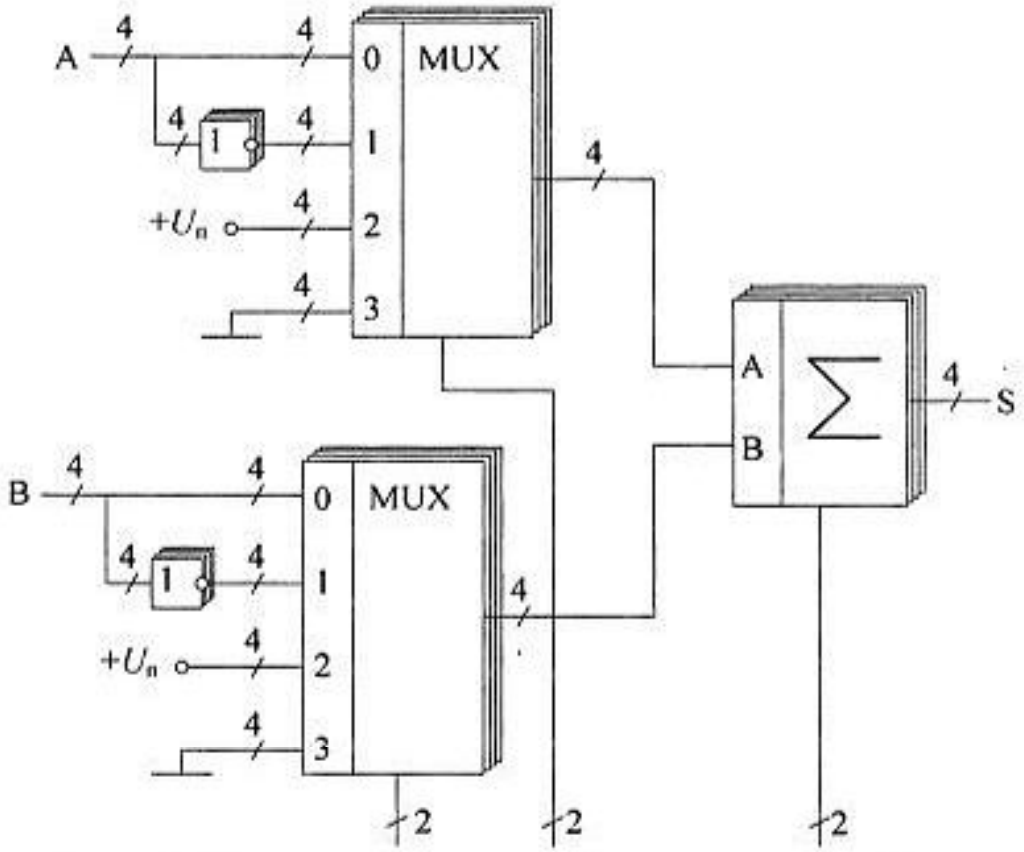


Рис. 18.21. Структурная схема арифметического устройства

Для определенности, пусть двоичный код 00 подключает к входу сумматора "0" мультиплексора MUX, код 01 подключаем к входу "1", код 10 подключаем к его входу "2", а код 11 — входу "3". На результат операции будет влиять входной сигнал переноса сумматора PI, поэтому его тоже включим в состав кода, управляющего схемой. В результате в таблице должно содержаться $2^5 = 32$ строки. Операции, которые будут выполняться данным арифметическим устройством в зависимости от кода, поданного на его управляющие входы, приведены в табл. 18.2.

Проанализируем эту таблицу. Если на все управляющие входы подать низкий потенциал, то к входу сумматора будут подключены коды A и B без инверсии. В этом случае будет производиться операция суммирования. Эта ситуация отображена первыми двумя строками (с номерами 0 и 1) таблицы выпол-

няемых операций. В зависимости от состояния входа переноса PI к этой сумме будет или не будет добавляться дополнительная единица.

Операция вычитания осуществляется строками 2, 3, 8 и 9. В этом случае один из операндов поступает на вход сумматора через блок инверторов. Единица, требуемая для получения дополнительного кода по одному из входов, подается на вход переноса сумматора PI.

Таблица 18.2. *Список команд арифметического устройства*

Номер строки	Управляющий код					Выполняемая операция
	K0	K1	K2	K3	PI	
0	0	0	0	0	0	S = A + B
1	0	0	0	0	1	S = A + B + 1
2	0	0	0	1	0	S = B – A – 1
3	0	0	0	1	1	S = B –A
4	0	0	1	0	0	S = B – 1
5	0	0	1	0	1	S = B
6	0	0	1	1	0	S = B
7	0	0	1	1	1	S = B + 1
8	0	1	0	0	0	S = A – B – 1
9	0	1	0	0	1	S = A – B
10	0	1	0	1	0	S = – A – B – 2
11	0	1	0	1	1	S = – B – A – 1
12	0	1	1	0	0	S = – B – 2
13	0	1	1	0	1	S = – B – 1
14	0	1	1	1	0	S = – B – 1
15	0	1	1	1	1	S = – B
16	1	0	0	0	0	S = A – 1
17	1	0	0	0	1	S = A
18	1	0	0	1	0	S = – A – 2
19	1	0	0	1	1	S = – A – 1
20	1	0	1	0	0	S = – 2
21	1	0	1	0	1	S = – 1
22	1	0	1	1	0	S = – 1

Таблица 18.2 (окончание)

Номер строки	Управляющий код					Выполняемая операция
	K0	K1	K2	K3	PI	
23	1	0	1	1	1	S = 0
24	1	1	0	0	0	S = A
25	1	1	0	0	1	S = A + 1
26	1	1	0	1	0	S = – A – 1
27	1	1	0	1	1	S = – A
28	1	1	1	0	0	S = – 1
29	1	1	1	0	1	S = 0
30	1	1	1	1	0	S = 0
31	1	1	1	1	1	S = +1

Часто используемой в микропроцессорах операцией является увеличение числа на единицу (инкрементирование) или уменьшение числа на единицу (декрементирование). Эти операции позволяют легко организовывать циклы в программе и переходить от предыдущего операнда к следующему. Они могут быть выполнены при помощи кодов, записанных в строках 4, 7, 16 и 25. И в этом случае единица, необходимая для увеличения значения числа, подается на вход переноса PI.

Код –1 приходится формировать в виде константы. Мы уже знаем, что отрицательное число можно получить из положительного числа проинвертировав его и добавив к полученному значению 1. В результате выполнения этих действий можно убедиться, что число –1 = 1111. Именно поэтому все входы "2" многоразрядных мультиплексоров MUX соединены с источником питания.

Кроме перечисленных действий, схема арифметического устройства может просто передавать на выход любой из входных кодов без изменения, что позволяет осуществлять копирование данных (суммирование с константой ноль) через арифметическое устройство без применения дополнительных схем коммутации.

При небольшом изменении принципиальной схемы полученное нами универсальное устройство сможет осуществлять не только арифметические, но и логические операции. Для этого нужно ввести дополнительный коммутатор, который будет разрывать цепи переноса между разрядами двоичного сумматора Σ. Эта управляющая цепь обычно называется M. В результате будет получена возможность осуществлять операцию суммирования по модулю два.

Подчеркнем основную особенность арифметико-логического устройства: выбор вида выполняемой операции производится при помощи кода, подаваемого на специальные выводы этого устройства. Это дает возможность использовать одно и то же устройство для выполнения различных функций. Разработка такого устройства позволила обменивать большую скорость выполнения отдельных операций на сложность реализуемого алгоритма, что, в конце концов, привело к разработке микропроцессорных систем. Развитие этих систем изменило окружающий нас мир.

Классификация микропроцессоров

Прежде чем приступить к изучению внутреннего устройства микропроцессоров, рассмотрим основные их типы, т. к. в зависимости от типа микропроцессора может значительно изменяться их внутреннее устройство.

По *внутреннему устройству* в настоящее время наметилось два направления развития микропроцессоров:

- RISC-процессоры (процессоры с сокращенным набором команд);
- CISC-процессоры (процессоры с полным набором команд).

В микропроцессорах с полным набором команд используется уровень микропрограммирования, обеспечивающий декодирование и выполнение команд микропроцессора. Команды, использующиеся для написания микропрограмм, называют микрокомандами. В этих микропроцессорах формат команды не зависит от аппаратуры процессора. Особенности построения аппаратуры микропроцессора отражаются только на микрокомандах. На одной и той же аппаратуре при смене микропрограммы могут быть реализованы различные микропроцессоры.

С другой стороны, при использовании микропрограммирования смена внутреннего устройства микропроцессора никак не влияет на программное обеспечение, предназначенное для него. При разработке новых микросхем микропроцессоров можно использовать аппаратурные решения никак не связанные с архитектурой предыдущей микросхемы. Главное, чтобы микропрограмма полностью эмулировала работу предыдущей микросхемы. В результате подобных действий пользователь воспринимает новую микросхему микропроцессора как полный аналог старой. С его точки зрения у микропроцессора только увеличивается производительность, снижается потребление энергии, уменьшаются габариты (если это необходимо).

Определение понятия микрокоманды и пример реализации микропрограммы будет подробно рассмотрен ниже по тексту данной главы. Поэтому сейчас эти понятия уточняться не будут.

Неявным недостатком CISC-процессоров является то, что производители микросхем стараются увеличить количество команд, которые может выполнять микропроцессор, тем самым увеличивая сложность микропрограммы и увеличивая время выполнения каждой команды.

В RISC-процессорах декодирование и исполнение команды производится аппаратно, поэтому количество команд ограниченно минимальным набором. В этих процессорах понятия "команда" и "микрокоманда" совпадают. Преимуществом RISC-процессоров является то, что команда может быть в принципе выполнена за один такт (не требуется выполнение микропрограммы), однако для выполнения тех же действий, которые выполняет одиночная команда CISC-процессора, обычно требуется выполнение некоторой последовательности команд RISC-процессоров, иногда эта последовательность команд довольно длинная. В результате выигрыш в быстродействии при применении RISC-микропроцессора может быть сведен к нулю.

В большинстве случаев быстродействие у RISC-процессоров выше, чем у CISC-процессоров. Тем не менее, при выборе процессора необходимо принимать в расчет все параметры в целом. Нужно учитывать, что тактовая частота RISC-процессора может оказаться значительно ниже, чем у CISC-процессора (особенно если в CISC-процессоре применяются специальные меры по повышению производительности). Это еще одна причина, по которой быстродействие RISC-процессора может оказаться ниже быстродействия подобного CISC-процессора.

Еще одним важным параметром, характеризующим микропроцессор, является объем исполняемой программы, требующийся для реализации конкретной задачи. Разрядность команды у RISC-процессора может оказаться выше, чем у CISC-процессора (что чаще всего и бывает). В результате общий объем исполняемой программы для RISC-процессора, как правило, превышает объем подобной программы для CISC-процессора.

Следующий признак классификации архитектур микропроцессоров — это система команд. По *системе команд* микропроцессоры отличаются огромным разнообразием, зависящим от фирмы-производителя и выполняемого класса задач. Тем не менее, можно определить два крайних варианта построения архитектуры микропроцессоров:

- аккумуляторные микропроцессоры;
- микропроцессоры с регистрами общего назначения.

В микропроцессорах с регистрами общего назначения операнды арифметических или логических команд могут находиться в любом внутреннем регистре, а в крайнем варианте реализации этой архитектуры и в любой ячейке памяти микропроцессорной системы. В зависимости от типа операции команда может быть одноадресной, двухадресной или трехадресной. Из-за такого разно-

образия возможностей система команд в микропроцессорах с регистрами общего назначения получается очень сложной, что является их явным недостатком.

Принципиальным отличием аккумуляторных микропроцессоров является то, что арифметические и логические операции могут производиться только над одной особой ячейкой памяти — аккумулятором. Для того чтобы произвести операцию над произвольной ячейкой памяти, ее содержимое необходимо скопировать в аккумулятор, выполнить требуемую операцию, а затем скопировать полученный результат обратно (или в другую произвольную ячейку памяти). В этом случае система команд получается наиболее простой, однако для выполнения того же самого действия, что выполняется одной командой микропроцессора с регистрами общего назначения, аккумуляторному микропроцессору потребуется несколько команд.

В настоящее время в чистом виде не существует ни та, ни другая архитектуры микропроцессоров. Все выпускаемые в настоящее время процессоры обладают системой команд с признаками как аккумуляторных процессоров, так и процессоров с регистрами общего назначения.

Следующий признак, по которому классифицируются микропроцессоры, — это способ работы с системной памятью. По *способу работы с системной памятью* существует два основных принципа построения микропроцессоров:

- гарвардская архитектура;
- архитектура фон Неймана.

В *гарвардской архитектуре* принципиально различаются два вида памяти:

- память программ;
- память данных.

В гарвардской архитектуре принципиально невозможно производить операцию записи в память программ, что исключает возможность случайного разрушения управляющей программы в случае неправильных действий над данными. Кроме того, в ряде случаев для памяти программ и памяти данных выделяются отдельные шины обмена данными. Эти особенности определили области применения гарвардской архитектуры микропроцессоров. Она применяется в микроконтроллерах, где требуется обеспечить высокую надежность работы аппаратуры. В сигнальных процессорах эта архитектура, кроме высокой надежности работы устройств, позволяет обеспечить высокую скорость выполнения программы за счет одновременного считывания управляющих команд и обрабатываемых данных.

Отличие архитектуры *фон Неймана* заключается в принципиальной возможности работы над управляющими программами точно так же, как и над данными. Это позволяет производить загрузку и выгрузку управляющих про-

грамм в произвольное место памяти процессора, которая в этой архитектуре не разделяется на память программ и память данных. Любой участок памяти может служить как памятью программ, так и памятью данных. ✧ *Обратите внимание*, что в разные моменты времени одна и та же область памяти может использоваться и как память программ, и как память данных. Для того чтобы программа могла работать в произвольной области памяти, ее необходимо модифицировать перед загрузкой, т. е. работать с нею как с обычными данными. Эта особенность архитектуры фон Неймана позволяет наиболее гибко управлять работой микропроцессорной системы, но создает принципиальную возможность искажения управляющей программы, что понижает надежность работы аппаратуры. Архитектура фон Неймана используется в универсальных компьютерах и в некоторых видах микроконтроллеров.

В качестве примера реализации микропроцессора рассмотрим устройство процессора с полным набором команд. Ядро CISC-микропроцессора состоит из двух основных частей:

- операционного блока;
- блока микропрограммного управления.

Операционный блок (ОБ) предназначен для считывания команд из системной памяти микропроцессорной системы и их выполнения. Эти действия он осуществляет под управлением блока микропрограммного управления (БМУ), который формирует последовательность микрокоманд, необходимую для выполнения операционным блоком одной машинной команды.

Типовые структуры операционного блока микропроцессора

Основным узлом операционного блока микропроцессора является арифметико-логическое устройство, внутреннее устройство которого мы рассмотрели выше по тексту. У этого узла присутствуют два входа и один выход данных. В результате логичным образом получается структурная схема операционного блока, приведенная на рис. 18.22.

На этом рисунке источники информации и результат выполнения операции хранятся в специальном сверхоперативном ОЗУ (СОЗУ), которое представляет собой небольшое количество регистров с возможностью одновременного считывания из двух регистров и записи в третий. Так как для передачи данных требуется три шины данных, то такая структура операционного блока называется трехшинной.

✧ *Обратите внимание*, что теперь в составе цифрового устройства присутствуют регистры, и, следовательно, для запоминания результатов работы арифметико-логического устройства на вход синхронизации этих регистров

необходимо подавать сигнал синхронизации CLK. Сигнал тактовой синхронизации обязательно требуется для правильной работы любого операционного блока.

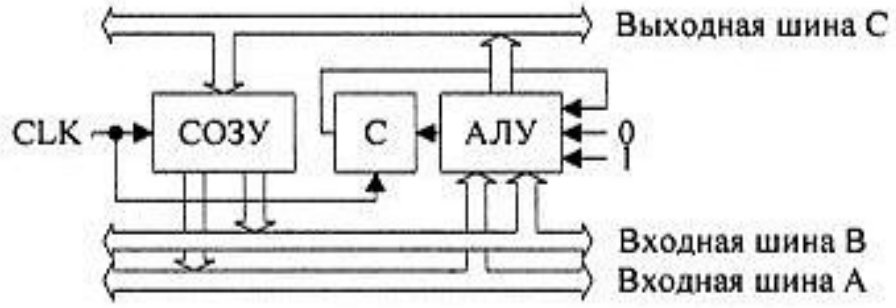


Рис. 18.22. Трехшинная структура операционного блока микропроцессора

Тактовые сигналы на операционный блок микропроцессора поступают с выхода тактового генератора, причем максимально возможная частота этого генератора, а следовательно, и время выполнения микропроцессором одной операции будет определяться временем прохождения цифровых сигналов через арифметико-логическое устройство и регистры сверхоперативного запоминающего устройства. Очень часто быстродействие микропроцессоров оценивают именно по значению максимально возможной частоты тактового генератора.

Обычно при построении операционного блока микропроцессора один из регистров-источников и регистр-приемник информации объединяют. Это позволяет сократить количество адресных шин в управляющем коде микропроцессора. В результате минимальное количество регистров в сверхоперативном ОЗУ составляет два регистра, однако количество регистров в операционном блоке обычно делается равным шестнадцати. Этого количества регистров вполне достаточно для реализации довольно сложных алгоритмов обработки данных и в то же время не приводит к необходимости значительного увеличения адресной части управляющей микрокоманды.

В микропроцессорной системе с применением трехшинного операционного блока возможно выполнение арифметических и логических операций в течение одного такта сигнала синхронизации. Это позволяет достигнуть максимального быстродействия микропроцессора. Именно поэтому подобная структура операционного блока широко используется внутри микросхем сигнальных процессоров.

Для обеспечения возможности работы с числами, число разрядов в которых превышает разрядность АЛУ, в состав операционного блока включен дополнительный триггер, в котором хранится флаг переноса в следующий разряд "С". При этом для того, чтобы операционный блок мог выполнять операции суммирования или вычитания с одиночной разрядностью чисел, к входу пе-

реноса АЛУ можно подключать не только выход триггера хранения флага переноса С, но и подавать уровень логического нуля или единицы.

Недостатком трехшинной структуры операционного блока является огромная площадь на кристалле микросхемы, которую занимают шины передачи данных, поэтому в более дешевых микропроцессорах используется другая структура операционного блока. Структурная схема подобного операционного блока микропроцессора приведена на рис. 18.23.

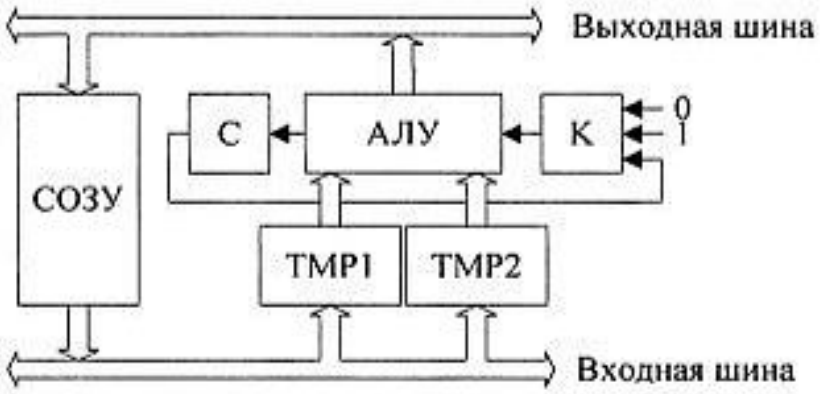


Рис. 18.23. Двухшинная структура операционного блока микропроцессора

На данном рисунке сигнал синхронизации не показан, однако этот сигнал подводится ко всем регистрам и триггеру хранения признака переноса С. В этой структуре операционного блока микропроцессора используется только две шины передачи данных, поэтому она получила название двухшинной. Для формирования двух источников данных для входов АЛУ в двухшинной схеме операционного блока микропроцессора используются два регистра временного хранения: TMP1 и TMP2.

В результате того, что входные данные к арифметико-логическому устройству передаются по одной шине данных, получается, что для выполнения одной операции требуется, как минимум, два такта сигнала синхронизации CLK. Это приводит к тому, что быстродействие данной структуры при той же частоте тактовой синхронизации микропроцессора будет ниже быстродействия трехшинной структуры операционного блока микропроцессора.

Наименьшую площадь на кристалле занимает одношинная структура операционного блока микропроцессора. Структурная схема подобного операционного блока микропроцессора приведена на рис. 18.24.

Теперь давайте рассмотрим подробнее особенности работы микропроцессора, построенного на основе подобного операционного блока. Прежде чем останавливаться на его внутреннем устройстве, определим, какие же задачи должен решать микропроцессор. Как уже упоминалось ранее, операционный блок микропроцессора предназначен для считывания команд из системной

памяти процессора и последующего их выполнения. При этом не важно, будет ли программа размещена в постоянном или оперативном запоминающем устройстве. Именно поэтому, прежде чем перейти к построению операционного блока микропроцессора, рассмотрим особенности команд, управляющих его работой.

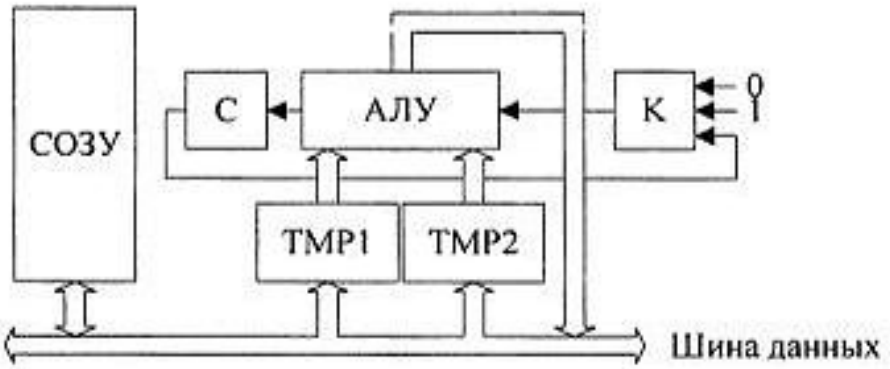


Рис. 18.24. Одношинная структура операционного блока микропроцессора

Наиболее простой структурой, как это тоже упоминалось ранее, обладают команды аккумуляторного микропроцессора. Давайте остановимся подробнее на принципах построения команд подобного процессора.

Команды микропроцессора

Команды микропроцессора в отличие от микрокоманд (которые жестко связаны с конкретной архитектурой операционного блока микропроцессора) разрабатываются независимо от внутреннего устройства микросхемы. В то же самое время они обычно хранятся в запоминающем устройстве, разрядность ячеек которого обычно кратна восьми разрядам (байту), поэтому разрядность команд тоже выбирается кратной восьми разрядам.

Команда микропроцессора должна содержать, как минимум, код исполняемой операции (КОП). В простейшем случае она может состоять только из кода операции, когда не требуется указывать адрес операнда (операнды — это данные, над которыми выполняется заданная операция). В случае применения восьмиразрядного кода операции микропроцессор может обладать 256 различными командами. Однобайтовые команды позволяют работать с внутренними программно доступными регистрами процессора (СОЗУ). Таких кодов операций назначается обычно меньше 256 значений.

Когда требуется работать с ячейками внутренней памяти, то однобайтового кода операции недостаточно для того, чтобы указать, с какой конкретной ячейкой памяти следует осуществить операцию. В этом случае в состав команды микропроцессора кроме кода операции включается адрес операнда,

и команда становится многобайтовой. Для многобайтовых команд назначаются оставшиеся значения кодов операции.

Многобайтовые команды могут содержать адреса операндов, размещенных в ОЗУ (или ПЗУ) системной памяти микропроцессора, или сами операнды (данные). Форматы команд микропроцессора очень сильно зависят от его архитектуры. Рассмотрим пример системы команд восьмиразрядного аккумуляторного процессора, построенного по архитектуре фон Неймана. Форматы команд подобного процессора приведены на рис. 18.25.

Рис. 18.25. Форматы различных команд микропроцессора

Если для кода операции используется восьмиразрядное число (байт), то при помощи этого числа можно закодировать 256 операций. В процессе разработки системы команд для операции может быть назначен любой код. Система команд является важным признаком, характеризующим конкретное семейство микропроцессоров.

При разработке системы команд проектировщик микропроцессора может назначить любой операции любое число. Например, для операции сложения можно назначить код 1, для операции вычитания код 12 и т. д. Для выполнения одной и той же операции над разными регистрами процессора назначаются разные коды команд. Поэтому для операции суммирования может потребоваться 8 чисел (команд). Например, 1 — просуммировать аккумулятор с регистром R0, 2 — просуммировать аккумулятор с регистром R1, 3 — просуммировать аккумулятор с регистром R2 и т. д.

В случае, когда надо выполнить операцию с использованием какой-либо константы, то восьмиразрядного кода команды становится недостаточно, ведь для 8-разрядного процессора требуется на один из входов арифметико-логического устройства подать восьмибитовое число. В результате в команду кроме кода операции включается само число, и команда микропроцессора становится 16-разрядной (двухбайтовой). В восьмиразрядном микропроцессоре для ее хранения потребуется две соседних ячейки памяти.

Следующая ситуация, которая приводит к увеличению длины команды, — это обращение к конкретной ячейке процессора. Обычно это команда копирования содержимого ячейки памяти в один из регистров сверхоперативного

запоминающего устройства. Для задания адреса в микропроцессоре с 64 килобайтами памяти требуется как минимум шестнадцатиразрядное число. Это число должно содержаться в составе команды микропроцессора. В результате длина команды с учетом кода операции увеличивается до трех байт, и она становится двадцатичетырехразрядной.

Все рассмотренные варианты форматов команд микропроцессора приведены на рис. 18.25. На этом рисунке все байты команд расположены друг под другом точно так же, как они располагаются в системной памяти микропроцессора.

Запоминать коды команд микропроцессора очень утомительно для человека. При программировании в машинных кодах легко совершить ошибку и очень трудно найти ее, особенно если коды команд различаются только одним битом. Конечно, для сокращения объема записи вместо двоичного кода можно воспользоваться шестнадцатеричным, однако это не увеличивает наглядности программы. Фрагмент шестнадцатеричного представления исполняемого кода программы для микропроцессора приведен в листинге 18.2.

Листинг 18.2. Фрагмент исполняемого кода микропроцессора

```
75D7001207E81200261216581216B9E4
FF121147BEFF05BFFF02800AE4FF1211
3079291201BA7F01121147BEFF05BFFF
02800A7F01121130792D1201BA1217CF
120005900003E493C0E0740193C0E074
01C0E0E4C0E07408C0E07B057A007936
12080074FB2581F581121074D2AFC298
```

В этом листинге в одной строке записано шестнадцать последовательных байт программного кода. Это сделано для экономии записи содержимого (дампа) памяти процессора. Каждый байт записан в шестнадцатеричном коде. Для записи байта при этом требуется две шестнадцатеричной цифры. Это тоже сделано для сокращения длины записи. Кроме того, непрерывная последовательность нулей и единиц значительно затрудняет восприятие информации человеком.

Ну, как? Очень легко разобраться в последовательности чисел, приведенной в листинге? Я думаю, не слишком. Чтобы уменьшить объем запоминаемой информации и увеличить наглядность исходного текста программы, для каждой операции процессора используют ее мнемоническое обозначение.

В качестве мнемонического обозначения операции микропроцессора обычно используют сокращения английских слов, образующих название этой операции. Например, для операции копирования содержимого одной ячейки памя-

ти в другую используется мнемоническое обозначение MOV; для операции суммирования — ADD; для операции вычитания — SUB; для операции умножения — MUL и т. д. Такие обозначения намного лучше запоминаются человеком. Более того! Количество запоминаемых команд значительно сокращается — ведь одному мнемоническому обозначению операции микропроцессора соответствует сразу несколько машинных команд. Обычно их количество соответствует количеству внутренних регистров микропроцессора.

Полная запись команды микропроцессора содержит мнемоническое обозначение операции и используемые этой операцией операнды, которые перечисляются через запятую. При этом обычно операнд-приемник результата записывается первым, а операнд-источник операнда — вторым. Например:

Листинг 18.3. Пример записи однобайтовых команд

```
адрес код   команда      комментарий
0000  F8    MOV R0, A    ;Скопировать содержимое регистра A в регистр R0
0001  2D    ADD A, R5    ;Просуммировать содержимое регистров R5 и A,
                         ;результат поместить в регистр A
```

В данном листинге в первой колонке приведен адрес ячейки памяти, в следующей колонке — значение команды, записанное в этой ячейке. Команда для краткости представлена в шестнадцатеричном коде. В следующей колонке показано мнемоническое обозначение этой команды. В последней колонке приведено пояснение действия команды на русском языке. Такое пояснение в языках программирования называется комментарием.

В первой строке листинга 18.3 приведена команда копирования содержимого регистра-аккумулятора в регистр R0. Команде соответствует исполняемый код F8, расположенный в нулевой ячейке памяти. Эта же команда соответствует мнемоническому обозначению `MOV R0, A`.

Во второй строке записана еще одна однобайтовая команда сложения содержимого регистров A и R5. Этой команде соответствует исполняемый код 2D, и т. к. эта команда следует за предыдущей командой, то она расположена в первой ячейке памяти. Команде соответствует мнемоническое обозначение `ADD A, R5`.

Приведенные в листинге 18.3 команды — однобайтовые, т. к. в них используются только внутренние регистры процессора. Если в команде используется константа в качестве операнда или указывается адрес операнда в памяти, то код команды, как это уже обсуждалось ранее, будет занимать в памяти два или три байта. Пример записи двухбайтовых команд приведен в листинге 18.4.

Листинг 18.4. Пример записи двухбайтовых команд

```
адрес код   команда        комментарий
0000  A819   MOV A, 1025   ;Скопировать содержимое ячейки памяти с
                           ;адресом1025 в регистр A
0002  246E   ADD A, #110   ;Просуммировать содержимое регистра A с числом
                           ;110
```

Несмотря на то что общий объем исходного текста программы увеличивается, скорость написания, и особенно отладки программ, при применении мнемонического обозначения команд микропроцессора возрастает. Кто в этом сомневается, может попробовать разобраться в участке программы, приведенной в листинге 18.2! Теперь вместо одного кода программы в памяти компьютера или на бумаге придется хранить два варианта ее представления: один для человека, в дальнейшем будем называть этот вариант исходным текстом программы; другой для микропроцессора, в дальнейшем будем называть этот вариант загрузочным модулем.

Следует отметить, что преобразование программы, записанной в мнемоническом виде, в машинные коды является рутинной работой, которую можно поручить компьютерной программе. Язык программирования, в котором для обозначения машинных команд используются мнемонические обозначения, получил название "ассемблер". Точно так же называют и программу (или пакет программ), которая осуществляет трансляцию (преобразование) исходного текста программы, написанной на языке программирования ассемблер (исходный модуль), в машинные коды (загрузочный модуль).

Теперь, после того как мы познакомились с понятием команды микропроцессора и тем как создаются наборы команд микропроцессоров, можно приступить к рассмотрению принципов построения внутреннего устройства микропроцессора, и начнем это с реализации его операционного блока.

Операционный блок микропроцессора

Попробуем в качестве примера внутреннего устройства микропроцессора реализовать аккумуляторный процессор с архитектурой фон Неймана. В этом случае потребуется наиболее простая система команд. В предыдущей главе мы выяснили, что микропроцессор считывает команды из оперативного (или постоянного) запоминающего устройства, а простейшая команда микропроцессора занимает в памяти микропроцессорной системы всего один байт. Это означает, что для ее считывания достаточно прочитать всего одну ячейку памяти. Поэтому в качестве первого примера выберем исполнение однобайтовой команды.

Прежде всего, давайте вспомним, как осуществляется чтение данных из оперативного запоминающего устройства, ведь команды микропроцессора ничем не отличаются от любого другого типа данных. Так как команда находится в запоминающем устройстве, то выполнение команды должно начаться с ее считывания из этого запоминающего устройства. Для этого операционный блок должен сформировать на внешних выводах микропроцессора сигналы, необходимые для считывания команды. Временные диаграммы сигналов, формируемых микропроцессором, при считывании однобайтных команд из памяти микропроцессорной системы приведены на рис. 18.26.

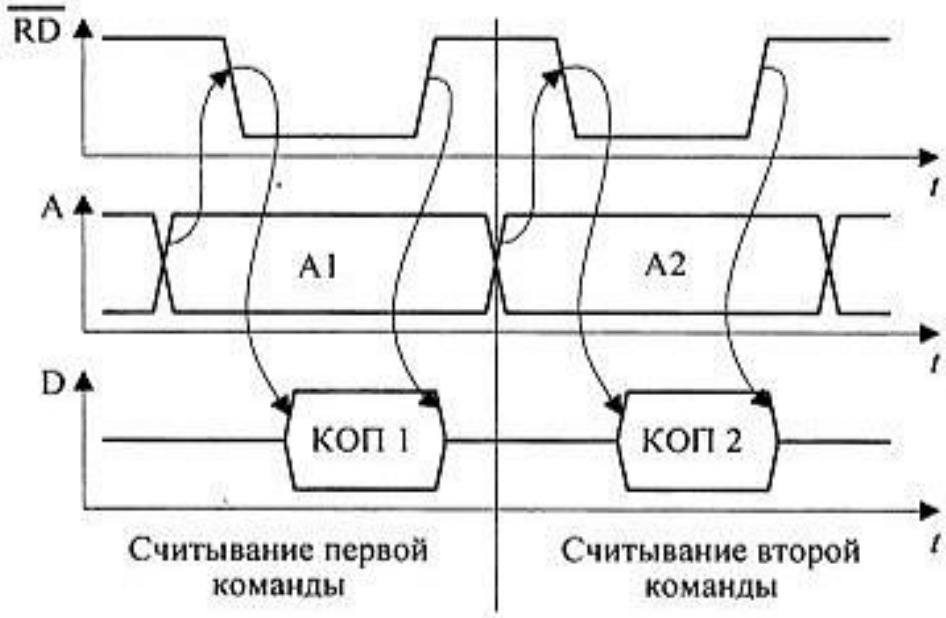


Рис. 18.26. Временные диаграммы сигналов считывания однобайтных команд из памяти

На этих временных диаграммах видно, что операция чтения команды из ОЗУ должна начинаться с выдачи на шину адреса ее текущего адреса, а это в свою очередь значит, что операционный блок микропроцессора должен уметь формировать сигналы на линиях адресной шины.

Определим необходимую разрядность шины адреса. Так как в качестве примера мы выбрали восьмиразрядный микропроцессор, то и все регистры в этом процессоре восьмиразрядные. Максимальное беззнаковое число, которое можно записать в такой регистр — 255, но для большинства программ такого объема памяти недостаточно. Именно поэтому для хранения адреса обычно выделяют два регистра, что позволяет формировать шестнадцатиразрядный адрес. Максимальное число, которое можно записать в этих двух регистрах, равно 65 535, что во многих случаях достаточно для адресации программ и обрабатываемых ими данных.

Для того чтобы можно было сформировать адрес ячейки запоминающего устройства по своему желанию, давайте выходы двух регистров, входящих в

состав сверхоперативного запоминающего устройства операционного блока, выведем за пределы микросхемы. В этом случае значения битов регистра адреса будут непосредственно определять уровни сигналов на линиях шины адреса. В результате такого действия мы потеряем возможность читать содержимое этих регистров, зато получим способность формировать на выводах микросхемы любое шестнадцатиразрядное число. Это число и будет адресом ячейки памяти микропроцессорного устройства. Для выдачи адреса на внешние выводы микросхемы достаточно записать его в эти два регистра.

Таким образом, в данной главе впервые рассказано о записи двоичного кода в регистр с целью формирования логических сигналов на внешних выводах микросхемы. Этот метод широко применяется в микропроцессорной технике, примеры его использования будут часто встречаться в последующих главах.

Назовем выбранную пару регистров — регистром адреса (RA). Два регистра требуются для того, чтобы получить шестнадцатиразрядный адрес, а регистры в сверхоперативном ОЗУ — восьмиразрядные. Применение двух восьмиразрядных регистров позволяет образовать шестнадцатиразрядную пару регистров. Для того чтобы различать регистры старшего и младшего байта регистра адреса, обозначим их как RAH — старший байт и RAL — младший байт.

Из временных диаграмм, приведенных на рис. 18.26, видно, что, кроме адреса и собственно данных, для взаимодействия с оперативным запоминающим устройством нам потребуется сигнал чтения. В дальнейшем мы собираемся научиться работать с данными, а значит, кроме сигнала чтения (RD) нам потребуется еще и сигнал записи (WR). Для формирования этих сигналов воспользуемся подобным решением и выведем за пределы микросхемы выходы еще одного регистра сверхоперативного запоминающего устройства. Так как сигналы, которые будет формировать этот регистр, являются управляющими, то и регистр назовем регистром управления — CNTR.

Для формирования необходимых для обмена данными сигналов достаточно записывать в определенный бит регистра управления(CNTR) логический 0 или 1. Определим его формат. Пусть бит 0 этого регистра будет сигналом записи, а бит 1 — сигналом чтения. На этапе первоначального проектирования это не важно. Однако после определения формата и внешних выводов микропроцессора это становится законом, и изменять его произвольным образом уже нельзя. Остальные биты регистра управления пока не важны. Если потребуются дополнительные сигналы управления системной шиной, то можно воспользоваться этими зарезервированными сейчас битами. Полученный в результате приведенных выше рассуждений формат регистра управления CNTR приведен на рис. 18.27.

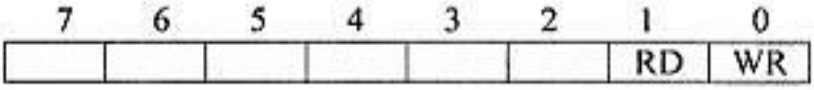


Рис. 18.27. Формат регистра управления (CNTR)

Уточненный вариант структурной схемы одношинного операционного блока, способный выполнять поставленные выше задачи, показан на рис. 18.28. Давайте остановимся на особенностях этого варианта схемы подробнее.

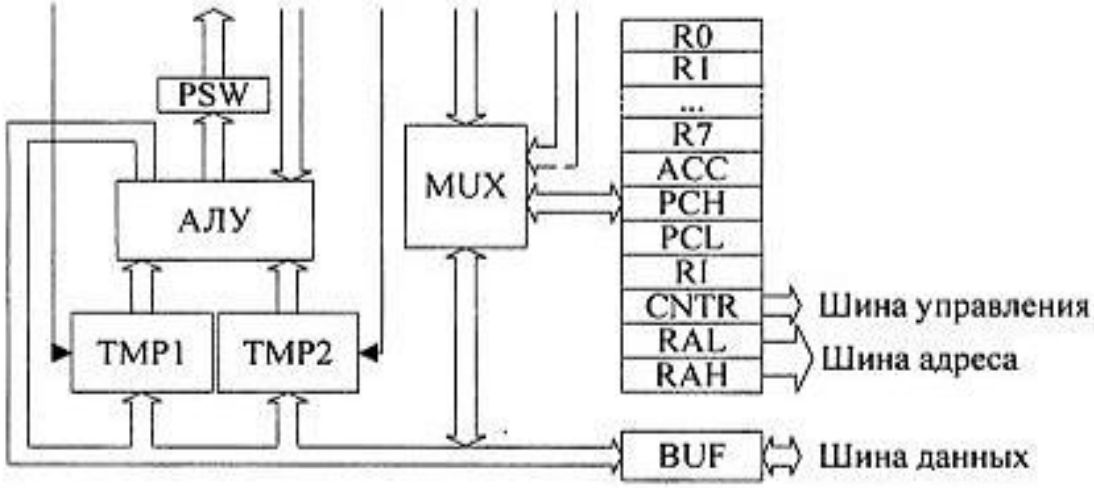


Рис. 18.28. Операционный блок микропроцессора

В схеме расписано назначение каждого регистра сверхоперативного запоминающего устройства. Может показаться, что у этих регистров есть какие-то особенности, однако это не так. В момент проектирования операционного блока все регистры абсолютно одинаковы. Особенности появляются под действием микропрограммы. Мы просто договариваемся, что первые восемь регистров будут использоваться микропрограммой как регистры общего назначения. Далее, для работы микропроцессора нам нужен аккумулятор? Назначим следующий по порядку регистр сверхоперативного запоминающего устройства аккумулятором (ACC).

Для работы с микросхемами ОЗУ (или ПЗУ), в которых хранится программа, требуется специальный счетчик, который будет определять начальный адрес команд микропроцессора. Назовем это устройство программным счетчиком (PC). Программный счетчик, как и регистр адреса, должен быть шестнадцатиразрядным. Для его реализации выделим следующие два регистра сверхоперативного запоминающего устройства. Для того чтобы превратить их в счетчик команд, в процессе выполнения микропрограммы к их содержимому будем добавлять длину текущей команды и тем самым вычислять адрес следующей. Для того чтобы различать регистры старшего и младшего байта программного счетчика, обозначим их как PCH — старший байт и PCL — младший байт. Это позволяет при помощи восьмиразрядного АЛУ формиро-

вать 16-разрядный адрес очередной команды при помощи последовательной работы с младшим и старшим байтом адреса.

Выход программного счетчика можно было бы подключить к внешним выводам микропроцессора, в состав которого входит рассматриваемый операционный блок. Однако кроме адресов команд при их выполнении часто требуются адреса данных, над которыми будут производиться операции. Именно поэтому программный счетчик выделен как отдельное устройство.

Для правильной работы микропрограммы нужно где-то запоминать текущую команду. Назначим следующий по порядку регистр — регистром команд (RI), ну а остальные регистры сверхоперативного ОЗУ, изображенные на структурной схеме рис. 18.28, мы уже рассматривали ранее.

Следующее отличие данной схемы от рассмотренной ранее схемы одношинного операционного блока — это отсутствие триггера хранения признака переноса С. Этот триггер вошел в состав специального регистра операционного блока — PSW. Кроме данного триггера в этот регистр объединены триггер, хранящий знак обработанного числа N, триггер, хранящий признак переполнения результата OV, а также триггер хранения признака нулевого результата Z. Значение признака переполнения вычисляется при помощи операции исключающего 'ИЛИ' между признаком переноса и знаком числа. Содержимое регистра PSW в схеме, приведенной на рис. 18.28, передается в блок микропрограммного управления. Так как признаки, содержащиеся в регистре PSW, характеризуют результат операции, то по каждому из них может быть организовано ветвление программы.

Формат регистра состояния микропроцессора PSW приведен на рис. 18.29. При необходимости значение признака переноса С может быть подано на вход АЛУ PI блоком микропрограммного управления.

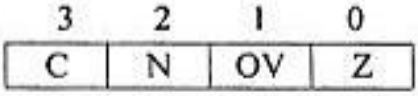


Рис. 18.29. Формат регистра состояния PSW

В схеме, приведенной на рис. 18.28, сверху к операционному блоку подведены линии, управляющие различными его блоками. Изменяя потенциалы на этих проводниках, можно выбирать операцию, которую выполнит операционный блок при поступлении очередного тактового импульса. Совокупность этих потенциалов называется микрокомандой. Если сразу же после поступления тактового импульса изменять микрокоманду, то операционный блок будет последовательно выполнять операции, заданные этой микрокомандой.

Теперь остановимся подробнее на формате микрокоманды. В схеме, приведенной на рис. 18.28, явно просматривается, что отдельные группы бит мик-

рокоманды управляют различными элементами операционного блока, поэтому их можно рассматривать независимо друг от друга. Такие группы битов называются полями микрокоманды. Формат микрокоманды операционного блока, показанного на рис. 18.28, приведен на рис. 18.30. Общая длина микрокоманды в данном случае получилась равной 22 битам.

6 бит	4 бит	1 бит	1 бит	8 бит	1 бит	1 бит
АЛУ	Адр. RG	tmp1	tmp2	Const	BUFin	BUFout

Рис. 18.30. Формат микрокоманды операционного блока, изображенного на рис. 4.27

Теперь нерешенной осталась еще одна проблема — откуда же будут поступать на входы управления операционного блока микропроцессора микрокоманды? Устройство, которое формирует последовательность микрокоманд, подаваемую на операционный блок микропроцессора, называется блоком микропрограммного управления. Давайте остановимся на устройстве этого блока подробнее.

Блок микропрограммного управления

В простейшем случае блок микропрограммного управления можно построить на двоичном счетчике и ПЗУ. В ПЗУ будут храниться микрокоманды, а двоичный счетчик определять, какая из микрокоманд будет подаваться в данный момент времени на операционный блок. Структурная схема подобного блока микропрограммного управления приведена на рис. 18.31.

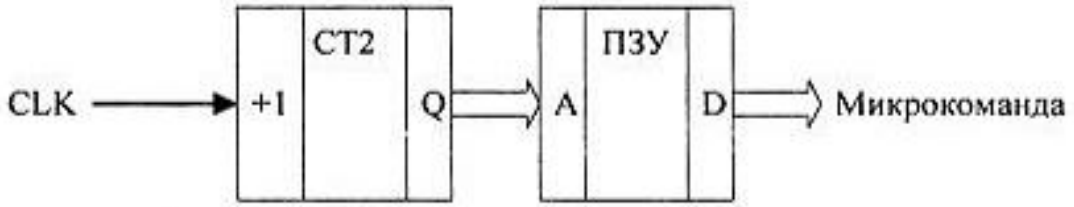


Рис. 18.31. Блок микропрограммного управления

В этой структурной схеме адрес очередной микрокоманды формирует двоичный счетчик. Свое состояние этот счетчик должен изменять с той же самой частотой, которая используется для синхронизации операционного блока. Разрядность счетчика определяется количеством микрокоманд, необходимых для реализации одной микропроцессорной команды, разрядность микрокоманды определяется операционным блоком. К сожалению, данная схема способна обеспечить последовательность микрокоманд, реализующую только одну команду микропроцессора.

Обычно после считывания из оперативного запоминающего устройства команды микропроцессора требуется декодировать ее, и, в зависимости от результата этого действия, выполнить свою, особенную последовательность действий, реализующую данную команду микропроцессора. После выполнения очередной команды блок микропрограммного управления снова переходит к считыванию из оперативного запоминающего устройства следующей команды. Количество веток в микропрограмме определяется количеством команд, выполняемых микропроцессором. При этом количество шагов (микрокоманд), требующееся для выполнения каждой из команд процессора, обычно невелико. Подобный алгоритм микропрограммы приведен на рис. 18.32.

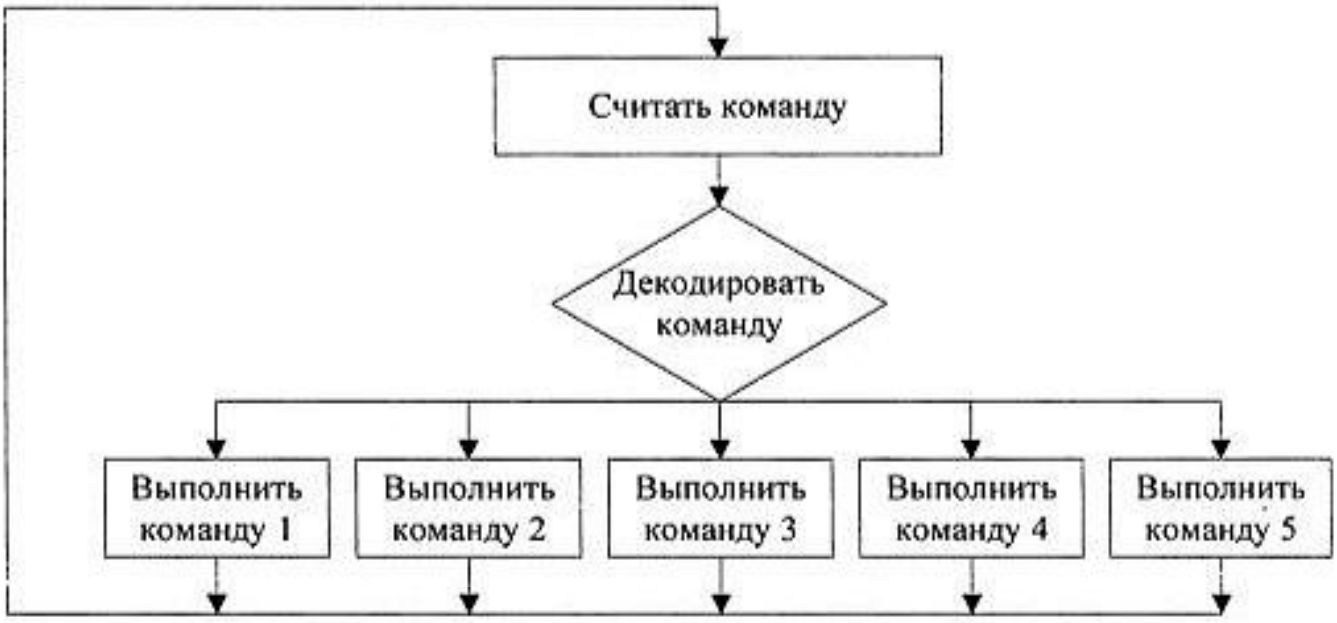


Рис. 18.32. Алгоритм микропрограммы операционного блока

Возникает вопрос — как же реализовать эти ветви алгоритма на одном ПЗУ? Для каждой из ветвей микропрограммы можно выделить отдельную область адресов ПЗУ, кратную двоичному числу, и тогда для перехода в эту область достаточно записать в счетчик старшие биты адреса. В младшие биты адреса при этом записываются нулевые значения. Обычно старшие разряды адреса микрокоманды совпадают с кодом команды микропроцессора, как это показано на рис. 18.33, и тогда для перехода на свою ветвь микропрограммы в счетчик можно записать непосредственно код команды микропроцессора из регистра команды RI.

Для изменения содержимого счетчика можно использовать дополнительный бит, записанный в ПЗУ. Этот сигнал соединяется с входом параллельной записи двоичного счетчика с предварительной записью. Для того чтобы вернуться в начало микропрограммы, достаточно подать сигнал на вход сброса счетчика в исходное состояние ‘R’. В результате описанных действий раз-

рядность микрокоманды увеличивается на два бита, и кроме битов, управляющих операционным блоком, появляются биты, управляющие самим блоком микропрограммного управления. Усовершенствованный блок микропрограммного управления приведен на рис. 18.34. В этой схеме биты, управляющие блоком микропрограммного управления, передаются по отдельным одноразрядным линиям.

Рис. 18.33. Размещение микропрограмм в ПЗУ

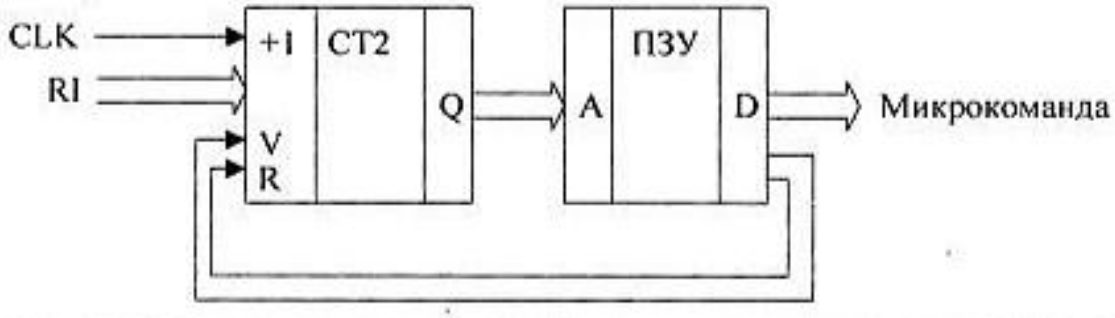


Рис. 18.34. Усовершенствованный блок микропрограммного управления

Существует ряд команд микропроцессора, которые должны выполняться только при каком-либо условии. Например, при установленном признаке переноса 'C'. Такие команды обычно реализуют условный переход на заданный адрес программы. Алгоритм выполнения команды условного перехода приведен на рис. 18.35.

Если в процессе выполнения команды требуется осуществить переход на новый адрес, то данное действие производится записью этого адреса в двоич-

ный счетчик команд. Переход к новому адресу микрокоманды произойдет по очередному фронту сигнала синхронизации микропроцессора CLK.

Рис. 18.35. Алгоритм условного перехода

Возникает вопрос — куда же мы будем писать новое значение адреса микрокоманды и где его взять? ✧ *Обратите внимание*, что немного ранее мы разбили весь диапазон значений адресов ПЗУ на ряд зон, отвечающих за каждую микрокоманду. Переход в процессе выполнения команды нужно осуществлять только в пределах блока адресов, выделенных для реализации данной команды. Это означает, что записывать новое значение адреса достаточно только в младшие разряды адреса, туда, куда ранее мы записывали нули.

Теперь остается открытым вопрос — где же взять значение нового адреса микрокоманды? Никто, кроме разработчика микропрограммы, этого не может знать, поэтому для хранения младших бит адреса перехода придется выделить еще несколько бит в постоянном запоминающем устройстве и, тем самым, дополнительно увеличить разрядность микрокоманды. Теперь формат микрокоманды нашего микропроцессора будет выглядеть так, как это приведено на рис. 18.36.

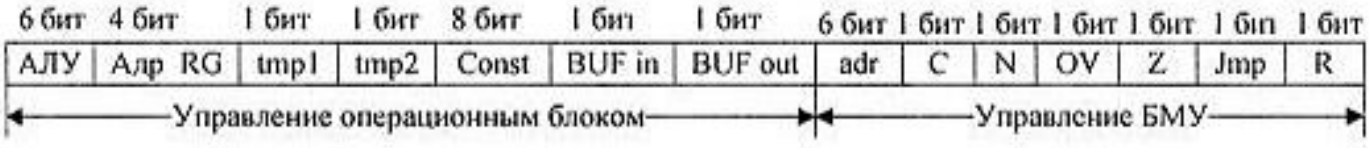


Рис. 18.36. Формат микрокоманды микропроцессора

Структурная схема блока микропрограммного управления тоже изменится, и теперь она будет выглядеть так, как это показано на рис. 18.37.

В приведенной схеме условный переход возможен по знаку результата операции, переносу, нулевому результату или переполнению. Следует заметить, что достаточно лишь флага N (знака числа) для реализации перехода по нескольким условиям: больше, меньше, больше или равно, меньше или равно.

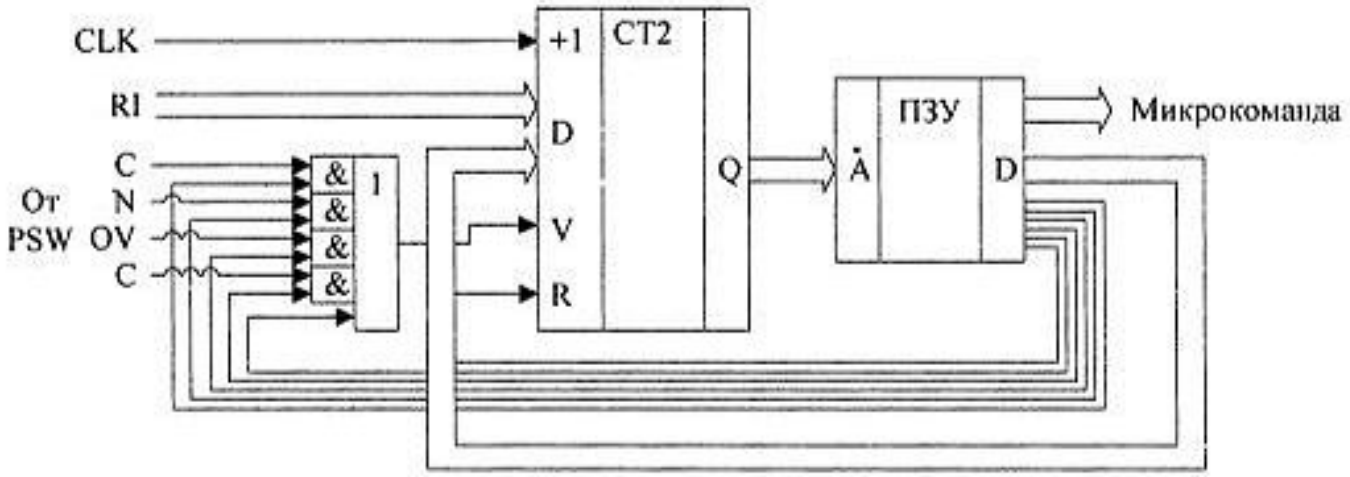


Рис. 18.37. Структурная схема блока микропрограммного управления

Содержимое ПЗУ блока микропрограммного управления называется микропрограммой. Именно эта микропрограмма и реализует конкретный микропроцессор. При смене микропрограммы, в принципе, можно реализовать на одном и том же кристалле другой микропроцессор, выполняющий другой набор команд и формирующий совершенно другие временные диаграммы на выводах системной шины.

Теперь, когда мы рассмотрели все составные части микропроцессора: операционный блок, блок микропрограммного управления, — а также систему команд, можно, наконец, приступить к реализации самого микропроцессора. Напомню, что архитектура микропроцессора в основном реализуется микропрограммой и очень мало зависит от внутреннего устройства микросхемы, для которой пишется эта микропрограмма. То есть, написав микропрограмму, мы тем самым построим микропроцессор с конкретной архитектурой. Микропрограмма состоит из сотен однотипных блоков, написанных для различных команд микропроцессора, поэтому для иллюстрации работы микропрограммы достаточно рассмотреть реализацию нескольких типовых команд.

Микропрограммирование

Все действия микропроцессора и сигналы на его выводах определяются последовательностью микрокоманд, подаваемых на управляющие входы его операционного блока. Эта *последовательность микрокоманд называется микропрограммой.*

При изучении принципов работы ОЗУ и ПЗУ приводились временные диаграммы, которые требуется сформировать для того, чтобы записать или прочитать необходимые данные. Любую временную диаграмму, как это уже обсуждалось в предыдущей главе, формирует микропроцессор. Устройство микросхемы, на примере которой мы будем рассматривать формирование

необходимых сигналов, было проанализировано при обсуждении операционного блока. По его структурной схеме (см. рис. 18.28) можно определить формат микрокоманды, управляющей этим блоком. Ее полный формат приведен на рис. 18.36. Для того чтобы разобраться в приводимых далее участках микропрограммы, желательно постоянно иметь перед собой временную диаграмму считывания команды микропроцессора, формат микрокоманды, схему операционного блока и список команд арифметического устройства.

Работа любого цифрового устройства начинается с заранее заданных начальных условий. Эти начальные условия в микропроцессоре устанавливаются специальным сигналом RESET (сброс), который формируется после подачи питания. Договоримся, что сигнал сброса микропроцессора будет записывать в регистр программного счетчика (PC) нулевое значение. Это условие справедливо не для всех микропроцессоров. Например, IBM-совместимые микропроцессоры при сбросе микросхемы записывают в свой программный счетчик значение F0000h, а процессоры 68HC11 фирмы freescale заносят в него содержимое ячейки памяти с адресом FFFFh.

Выполнение любой команды микропроцессора начинается со считывания ее кода из микросхем системной памяти (ОЗУ или ПЗУ). Необходимые для выполнения этого действия микрокоманды подаются на входы управления операционного блока из ПЗУ блока микропрограммного управления, как только с соответствующего вывода микропроцессора снимается сигнал сброса RESET.

В случае выполнения однобайтной команды, из системной памяти микропроцессора достаточно считать только код операции и выполнить задаваемые им действия. Временная диаграмма этого процесса приведена на рис. 18.38. Последовательность операций, которые необходимо выполнить микропрограмме, на этой временной диаграмме показана стрелочками. Для считывания следующей команды микропрограмма запускается заново (зацикливается), как это уже показывалось в алгоритме, приведенном на рис. 18.32.

Для того чтобы считать код операции из памяти, сначала необходимо адрес этой команды выставить на шине адреса системной шины микропроцессорной системы. Этот адрес хранится в счетчике команд (PC) микропроцессора. После сигнала сброса микропроцессора RESET в регистрах программного счетчика хранится нулевое значение. Скопируем его содержимое в регистр адреса (RA), выходы которого подключены к шине адреса. Так как его выходы подключены к выводам шины адреса микропроцессора, то на шине адреса появится именно этот адрес.

Листинг участка микропрограммы, который обеспечивает выдачу адреса из программного счетчика на шину адреса микропроцессорной системы, приведен в листинге 18.5.

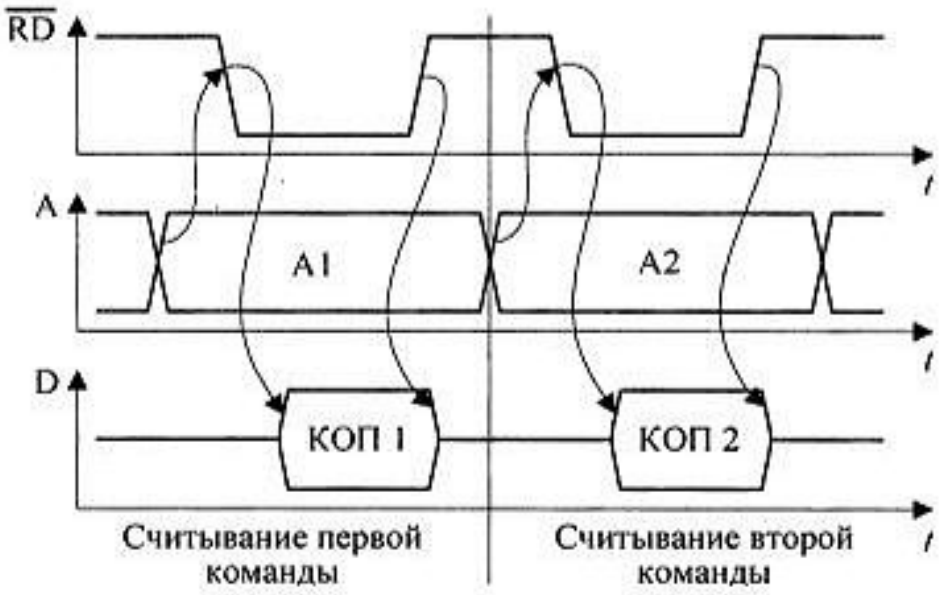


Рис. 18.38. Временные диаграммы сигналов считывания однобайтных команд из памяти

Листинг 18.5. Копирование содержимого программного счетчика в регистр адреса

		Поля микрокоманды операционного блока						
№	Описание	АЛУ	Адр. RG	TMP1	TMP2	Константа	BUF in	BUF out
1)	PCH -> TMP1	1111 1 1	1001	1	0	1111 1111	0	0
2)	TMP1-> RAH	1000 1 0	1110	0	0	1111 1111	0	0
3)	PCL -> TMP1	1111 1 1	1010	1	0	1111 1111	0	0
4)	TMP1-> RAL	1000 1 0	1101	0	0	1111 1111	0	0

В первой микрокоманде АЛУ операционного блока микропроцессора не участвует, поэтому в соответствующее поле можно занести любой код. В примере, для определенности, в это поле записаны все единицы. Следующее поле микрокоманды определяет адрес регистра сверхоперативного запоминающего устройства, откуда будет осуществляться копирование информации. Ранее мы договорились, что в качестве старшего байта программного счетчика PCH мы будем использовать его девятый регистр. Именно этот адрес и записан в этом поле микрокоманды. В двоичном коде число девять записывается как 1001. Единица в поле TMP1 означает, что запись кода с выхода сверхоперативного запоминающего устройства осуществляется именно в этот регистр. Регистр TMP2 в данной микрокоманде не используется. Для этого мы записываем в соответствующее поле микрокоманды значение логического нуля. Константа в первой микрокоманде тоже не нужна, и т. к. мультиплексор подключен к девятому регистру, а, следовательно, вход константы отключен, то

мы можем записать в поле константы что угодно. Для определенности заносим в это поле микрокоманды все единицы. Последние два бита микрокоманды закрывают шинный формирователь BUF.

В следующей микрокоманде осуществляется обратная операция. Теперь содержимое временного регистра TMP1 копируется в сверхоперативное запоминающее устройство, поэтому в данной операции участвует АЛУ. Для определения команд АЛУ используется табл. 18.2. В этой таблице передаче двоичной информации с входа A на выход АЛУ соответствует управляющий код 10001. Именно он и заносится в поле микрокоманды АЛУ. Старший байт регистра адреса является четырнадцатым регистром сверхоперативного запоминающего устройства, поэтому в поле Адр. RG занесено число четырнадцать — 1110. Операция записи в регистр сверхоперативного запоминающего устройства выбирается комбинацией бит TMP1 и TMP2, равной 00.

Третья и четвертая микрокоманды практически совпадают с первой и второй. Отличие заключается только в адресах регистров сверхоперативного запоминающего устройства. Здесь использованы младший регистр программного счетчика PCL (номер десять) — 1010 и младший регистр адреса RAL (номер тринадцать) — 1101.

Теперь на выводах шины адреса микропроцессора появился адрес текущей команды. Судя по временной диаграмме, приведенной на рис. 18.38, после выдачи адресной информации требуется сформировать нулевой потенциал на выводе сигнала чтения RD (листинг 18.6). Для этого в регистр управления CNTR запишем константу 1111 1101 (см. формат регистра управления CNTR, приведенный на рис. 18.27).

Листинг 18.6. Формирование нулевого потенциала на выводе микропроцессора RD

```
                                Поля микрокоманды операционного блока
№    Описание              АЛУ     Адр. RG  TMP1  TMP2  Константа  BUF in BUF out
5) 11111101 -> TMP1  1111 1 1     1111       1     0    1111 1101      0       0
6) TMP1-> CNTR       1000 1 0     1100       0     0    1111 1111      0       0
```

После выполнения пятой и шестой микрокоманд на временной диаграмме, приведенной на рис. 18.38, сигнал чтения системной памяти RD примет нулевое значение. В этом можно убедиться при помощи осциллографа. В пятой микрокоманде используется константа 1111 1101, которую необходимо записать в регистр управления. В качестве адреса регистра сверхоперативного запоминающего устройства в данной микрокоманде используется адрес константы (пятнадцать) — 1111.

Теперь можно считать число с шины данных, а т. к. постоянное запоминающее устройство в этот момент под воздействием сигнала чтения RD выдает на нее код операции микропроцессора, то именно этот код операции и будет считан микропроцессором. Как это уже обсуждалось ранее, для правильной работы блока микропрограммного управления код операции команды микропроцессора необходимо скопировать в регистр команд RI. Это осуществляется следующей микрокомандой (листинг 18.7):

Листинг 18.7. Чтение кода операции с шины данных микропроцессора

		Поля микрокоманды операционного блока						
№	Описание	АЛУ	Адр. RG	TMP1	TMP2	Константа	BUF in	BUF out
7)	data -> RI	1111 1 1	1011	0	0	1111 1111	1	0

В седьмой микрокоманде открывается буфер шинного формирователя битом микрокоманды BUF in. Учитывая, что для выполнения операции копирования АЛУ на этот раз не требуется, то можно сразу же выбрать адрес назначения RI — двенадцать (1100). Операция записи в регистр сверхоперативного запоминающего устройства выбирается комбинацией бит TMP1 и TMP2, равной 00. Поля константы и управления АЛУ в данной микрокоманде не требуются, поэтому запишем в них единичные значения (хотя если записать туда другие значения, то действие микрокоманды от этого не изменится).

После запоминания кода операции в регистре RI необходимо снять сигнал чтения с системной шины (выставить на этом выводе микропроцессора единичный потенциал). Для этого в регистр управления CNTR снова запишем константу 1111 1111 (листинг 18.8).

Листинг 18.8. Формирование единичного потенциала на выводе микропроцессора RD

		Поля микрокоманды операционного блока						
№	Описание	АЛУ	Адр. RG	TMP1	TMP2	Константа	BUF in	BUF out
8)	11111111 -> TMP1	1111 1 1	1111	1	0	1111 1111	0	0
9)	TMP1-> CNTR	1000 1 0	1100	0	0	1111 1111	0	0

Действие и формат этих микрокоманд практически не отличается от микрокоманд 5 и 6, поэтому назначение их полей приводиться не будет.

Прежде чем перейти к дальнейшему выполнению микропрограммы, увеличим содержимое счетчика команд на 1, подготовившись тем самым к выполнению следующей команды микропроцессора (листинг 18.9).

Листинг 18.9. Увеличение содержимого программного счетчика на 1

		Поля микрокоманды операционного блока						
№	Описание	АЛУ	Адр. RG	TMP1	TMP2	Константа	BUF in	BUF out
10)	PCL -> TMP1	1111 1 1	1010	1	0	1111 1111	0	0
11)	TMP1+1 -> PCL	1100 1 0	1010	0	0	1111 1111	0	0
12)	PCH -> TMP1	1111 1 1	1001	1	0	1111 1111	0	0
13)	TMP1+C-> PCH	1100 1 0	1001	0	0	1111 1111	0	0

До сих пор в листингах приводилось содержимое только той части микрокоманды, которая отвечает за работу операционного блока. Однако микрокоманды 11 и 13 не отличаются друг от друга. Отличие проявляется в части управления блоком микропрограммного управления. В результате по 11 команде на вход переноса PI арифметико-логического блока поступит уровень логической единицы, а в команде 13 к этому же входу будет подключен выход триггера хранения признака переноса, обеспечивая тем самым выполнение операции многоразрядного суммирования.

После считывания команды ее необходимо декодировать. Четырнадцатым тактом микропрограмма направляется на одну из 256 ветвей, отвечающую за выполнение считанной команды. Так как в этом случае работают поля управления блоком микропрограммного управления, то в листинге 18.10 приведены именно эти поля микрокоманды.

Листинг 18.10. Декодирование команды микропроцессора

		Поля микрокоманды блока микропрограммного управления						
№	Описание	Адр. ПЗУ	C	N	OV	Z	Jmp	R
14)	RI -> CT	000000	0	0	0	0	1	0

То, что в поле адреса ПЗУ записано нулевое значение означает, что переход осуществляется на адрес первой микрокоманды выполнения команды микропроцессора. Это поле, как это уже обсуждалось ранее, требуется при выполнении условных переходов внутри участка микропрограммы, отвечающего за выполнение команды микропроцессора. В рассматриваемой микрокоманде все условные переходы запрещены нулевыми значениями в соответствующих битах. Переход разрешен только битом Jmp. По этому сигналу в старшие биты счетчика микрокоманд заносится содержимое регистра инструкций RI, которое совпадает с начальным адресом микропрограммы выполнения соответствующей команды микропроцессора.

Теперь остается выполнить данную команду. Например, если это была команда копирования содержимого регистра `R0` в аккумулятор `A` (`MOV A,R0`), то для ее выполнения будет достаточно буквально двух микрокоманд. Этот

участок микропрограммы будет выглядеть так, как это приведено в листинге 18.11. В нем тоже приведена только часть микрокоманды, отвечающая за управление операционным блоком микропроцессора.

Листинг 18.11. Выполнение действия команды микропроцессора

		Поля микрокоманды операционного блока						
№	Описание	АЛУ	Адр. RG	TMP1	TMP2	Константа	BUF in	BUF out
15)	R0 -> TMP1	1111 1 1	0000	1	0	1111 1111	0	0
16)	TMP1-> ACC	1000 1 0	1000	0	0	1111 1111	0	0

И т. к. в этом случае команда микропроцессора полностью выполнена, то счетчик микрокоманд сбрасывается для считывания из системной памяти и выполнения следующей команды. Это действие осуществляется записью в бит R микрокоманды единичного значения (поля управления блоком микропрограммного управления). ✧ *Обратите внимание*, что в первой колонке описания микрокоманд приводится номер тактового импульса сигнала синхронизации, а не адрес ячейки памяти микропрограммы, который будет зависеть от выполняемой команды микропроцессора.

Для того чтобы убедиться, что мы правильно разобрались с принципами написания микропрограмм для микропроцессоров, давайте рассмотрим еще один пример выполнения его команды. На этот раз познакомимся с особенностями выполнения многобайтовой команды. Пусть из системной памяти микропроцессора считывается команда безусловного перехода JMP 1234. Временная диаграмма сигналов, формируемых микропроцессором для считывания данной команды, приведена на рис. 18.39.

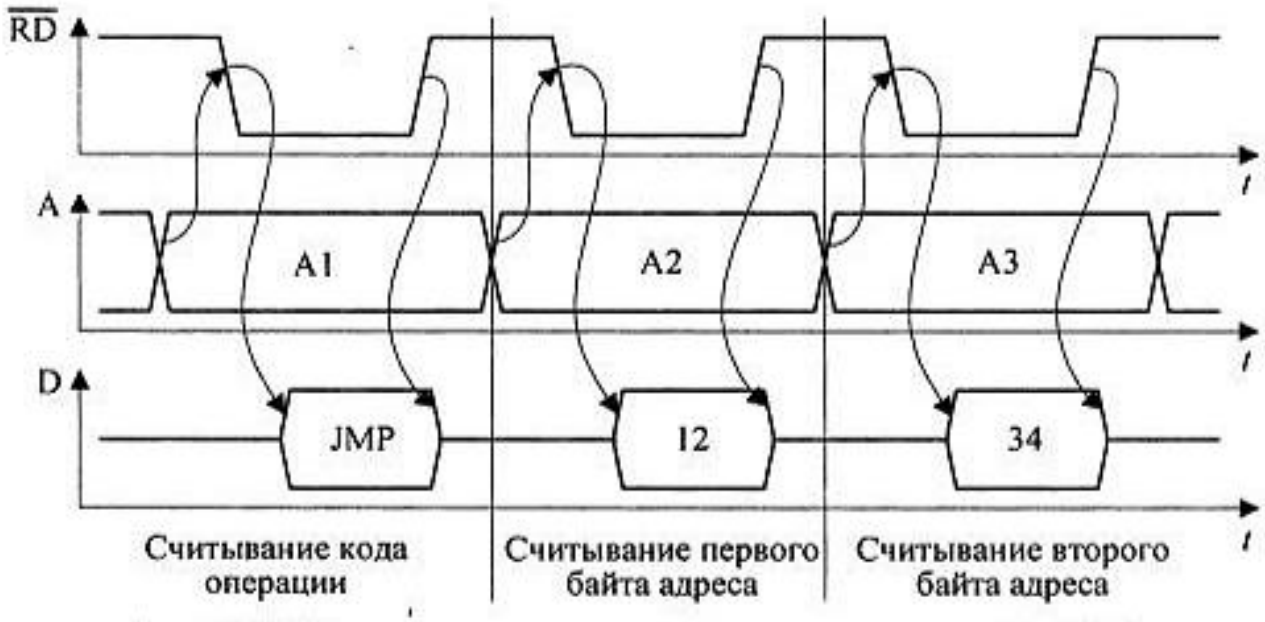


Рис. 18.39. Временная диаграмма выполнения команды JMP 1234

Первые четырнадцать микрокоманд совпадают для всех команд микропроцессора, т. к. код операции требуется считать для любой команды. Различие наступает, начиная с пятнадцатой микрокоманды, которая зависит от выполняемой инструкции. При выполнении команды безусловного перехода необходимо считать адрес новой команды, который записан в байтах, следующих за кодом операции. Этот процесс аналогичен считыванию кода операции и приведен в листинге 18.12.

Листинг 18.12. Считывание первого байта адреса перехода

```
                                Поля микрокоманды операционного блока
№    Описание              АЛУ     Адр. RG TMP1 TMP2 Константа  BUF in BUF out
Выдать адрес первого байта адреса перехода
15) PCH -> TMP1        1111 1 1   1001     1    0    1111 1111     0       0
16) TMP1-> RAH         1000 1 0   1110     0    0    1111 1111     0       0
17) PCL -> TMP1        1111 1 1   1010     1    0    1111 1111     0       0
18) TMP1-> RAL         1000 1 0   1101     0    0    1111 1111     0       0
Сформировать низкий уровень сигнала чтения RD
19) 11111101 -> TMP1   1111 1 1   1111     1    0    1111 1101     0       0
20) TMP1-> CNTR        1000 1 0   1100     0    0    1111 1111     0       0
Запомнить первый байт адреса перехода в регистре TMP2
21) data -> TMP2       1111 1 1   1100     0    1    1111 1111     1       0
Сформировать высокий уровень сигнала чтения RD
22) 11111111 -> TMP1   1111 1 1   1111     1    0    1111 1111     0       0
23) TMP1-> CNTR        1000 1 0   1100     0    0    1111 1111     0       0
Перейти к следующему байту команды
24) PCL -> TMP1        1111 1 1   1010     1    0    1111 1111     0       0
25) TMP1+1 -> PCL      1100 1 0   1010     0    0    1111 1111     0       0
26) PCH -> TMP1        1111 1 1   1001     1    0    1111 1111     0       0
27) TMP1+C-> PCH       1100 1 0   1001     0    0    1111 1111     0       0
```

В результате выполнения этого участка микропрограммы первый байт адреса перехода оказывается запомненным в регистре временного хранения TMP2. Теперь считаем второй байт адреса перехода. Эта операция, а также переход на новый адрес выполняемой команды приведен в листинге 18.13.

Листинг 18.13. Считывание второго байта адреса перехода

```
                                Поля микрокоманды операционного блока
№    Описание              АЛУ    Адр. RG  TMP1 TMP2 Константа  BUF in BUF out
Выдать адрес второго байта адреса перехода
28) PCH -> TMP1        1111 1 1   1001     1    0   1111 1111     0      0
29) TMP1-> RAH         1000 1 0   1110     0    0   1111 1111     0      0
30) PCL -> TMP1        1111 1 1   1010     1    0   1111 1111     0      0
31) TMP1-> RAL         1000 1 0   1101     0    0   1111 1111     0      0
Сформировать низкий уровень сигнала чтения RD
32) 11111101 -> TMP1   1111 1 1   1111     1    0   1111 1101     0      0
33) TMP1-> CNTR        1000 1 0   1100     0    0   1111 1111     0      0
Занести второй байт адреса перехода в старший регистр программного
счетчика PCH
34) data -> PCH        1111 1 1   1001     0    0   1111 1111     1      0
Сформировать высокий уровень сигнала чтения RD
35) 11111111 -> TMP1   1111 1 1   1111     1    0   1111 1111     0      0
36) TMP1-> CNTR        1000 1 0   1100     0    0   1111 1111     0      0
Занести первый байт адреса перехода в младший регистр программного
счетчика PCL
37) TMP2 -> PCL        0001 0 1   1010     0    0   1111 1111     0      0
```

В результате выполнения этой микропрограммы в программный счетчик будет загружено число, записанное во втором и третьем байтах команды безусловного перехода `JMP 1234`, а это означает, что следующей будет считана команда микропроцессора, лежащая по этому адресу его системной памяти.

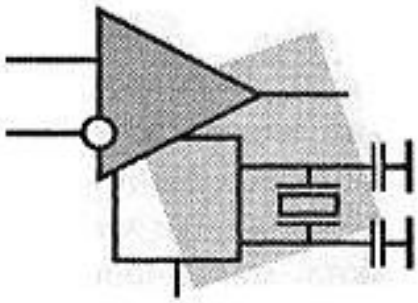
По аналогии с рассмотренными примерами выполнения команд микропроцессора можно разработать и другие ветки микропрограммы, которые могут понадобиться в дальнейшем.

Итоги

В данной главе мы изучили особенности отображения различных видов информации при помощи двоичных кодов. Познакомились с принципами обработки двоичных кодов, разрядность которых превышает разрядность микропроцессора. Рассмотрели принципы формирования основных признаков завершения операции (знак, перенос, переполнение, антипереполнение и ра-

венство нулю). Кроме того, мы рассмотрели внутреннее устройство основного узла микропроцессора — АЛУ и особенности реализации операционного блока микропроцессоров различного назначения. В качестве примера был рассмотрен один из способов реализации блока микропрограммного управления. На этом закончим рассмотрение внутреннего устройства и принципов работы микропроцессора. Полученных знаний вполне достаточно для того, чтобы разобраться, как работает микропроцессор, и приступить к рассмотрению принципов работы систем, построенных с его использованием.

ГЛАВА 19



Принципы работы микропроцессорной системы

В предыдущей главе были рассмотрены принципы работы микропроцессора — универсального цифрового устройства, позволяющего выполнять различные виды арифметических и логических операций. Как будет показано в последующих главах, с его помощью можно реализовывать цифровые устройства различного назначения. Однако мы пока не умеем работать с микропроцессором. Сам по себе микропроцессор выполнить эти задачи не может. Для обеспечения его работы к нему нужно подключить дополнительные устройства, такие как постоянные и оперативные запоминающие устройства, порты ввода-вывода, таймеры. Иными словами, требуется построить микропроцессорную систему. В данной главе будут рассмотрены структурные схемы подключения к микропроцессору этих устройств. Структурные схемы микропроцессорной системы приводятся с уровнем детализации, позволяющим легко превратить их в принципиальные схемы.

Кроме того, в данной главе будут рассмотрены основные методы расширения адресного пространства микропроцессорной системы, позволяющие продлить время жизни конкретного семейства микропроцессоров и облегчить построение многозадачных операционных систем, а также кратко будут рассмотрены некоторые решения, позволяющие повысить быстродействие системы в целом.

Не следует забывать, что микропроцессорная система как самостоятельное устройство никого не интересует. Это только инструмент решения задач управления какими-либо объектами или устройство, предназначенное для выполнения задач обработки сигналов. В данной главе мы рассмотрим узлы микропроцессорной системы, позволяющие управляющей программе получать информацию из окружающей аппаратной среды (в том числе и от человека) и, в свою очередь, воздействовать на нее.

При решении задач управления или обработки сигналов очень важно, чтобы решения процессора были согласованы во времени с окружающими событиями. Поэтому будут рассмотрены узлы микропроцессорной системы, позволяющие организовывать взаимодействие микропроцессорной системы с окружающей средой в реальном времени.

Подключение внешних устройств к микропроцессору

Микропроцессорные системы часто используются для управления устройствами, блоками или системами связи. При этом для больших и дорогих систем связи, таких как автоматические телефонные станции или коммутационные центры сотовых систем связи, в качестве микропроцессорного устройства может быть использован универсальный компьютер или группа компьютеров, объединенных локальной или глобальной сетью связи. Для дешевой и портативной аппаратуры применяются специализированные микропроцессорные устройства, в качестве которых чаще всего используется однокристальный микроконтроллер, представляющий собой полностью законченную управляющую микропроцессорную систему с ПЗУ, ОЗУ, последовательными и параллельными портами, АЦП и таймерами, размещенными внутри одной микросхемы. Именно такой вариант решения вопросов управления устройствами связи и рассматривается в данной книге.

Внешними устройствами называются любые устройства, которыми управляет, от которых получает или которым передает информацию управляющая программа микропроцессора. В качестве внешних устройств для микропроцессора может выступать принтер или дисплей, клавиатура или модем, но для устройств связи в качестве внешних устройств чаще всего выступают микросхемы кабельных, радио- или оптических приемников, кабельных, радио- или оптических передатчиков (в том числе и передатчики, построенные на базе сигнальных процессоров), микросхемы синтезаторов частоты, исполнительные устройства или постоянные запоминающие устройства с электрическим стиранием.

Управление подобными устройствами осуществляется обычно через параллельные или последовательные порты. В предыдущих главах в качестве примера использования цифровой техники подробно описывалось внутреннее устройство последовательных портов. Немного позднее мы рассмотрим, как эти порты подключаются к микропроцессорной системе.

Согласование сигналов цифровых микросхем между собой не представляет трудностей, т. к. практически все современные цифровые микросхемы по входным и выходным напряжениям согласованы с уровнями TTL. Если же

это не так, то для согласования нестандартных цифровых логических уровней с уровнями TTL выпускаются специальные микросхемы. Эти вопросы мы уже рассматривали подробно в самом начале данной книги. Несколько иначе обстоит дело с устройствами индикации и исполнительными устройствами.

Подключение различных типов индикаторов и расчет согласующих схем для этих устройств тоже был подробно рассмотрен при изучении основ цифровой техники. Несколько иначе выглядит схема подключения внешних исполнительных электромеханических устройств. Это связано с тем, что чаще всего такие исполнительные устройства являются индуктивной нагрузкой. В качестве примера можно назвать такие исполнительные устройства, как электромагнитное реле, шаговый двигатель или электромагнит, однако именно эти устройства и представляют максимальный интерес при разработке устройств управления.

Сложность при подключении таких устройств, как вы это помните из курса основ электротехники, заключается в том, что ток через индуктивность не может измениться мгновенно, поэтому при закрывании транзисторного ключа возникает напряжение самоиндукции, которое стремится сохранить предыдущее значение тока. Это приводит к тому, что чем больше отношение сопротивлений открытого и закрытого ключа, тем большее напряжение самоиндукции будет приложено к этому ключу. В результате если не принять дополнительных мер защиты, то транзистор электронного ключа будет выведен из строя.

В простейшем случае для этой цели достаточно поставить диод, через который будет замыкаться ток самоиндукции. В качестве примера, на рис. 19.1 приведена схема транзисторного ключа, позволяющая подключать к микропроцессорной системе электромагнитное исполнительное реле. Диод VD1 в этой схеме служит для ограничения напряжения импульсов ЭДС самоиндукции, которые могут вывести из строя силовой транзистор VT1.

При закрывании транзисторного ключа напряжение самоиндукции открывает диод VD1, и ток, протекавший ранее через открытый транзистор VT1, через этот диод замыкается на выводы индуктивной нагрузки. В результате напряжение на коллекторе транзистора VT1 увеличивается по сравнению с напряжением питания только на значение падения напряжения на открытом диоде. Расчет остальных элементов транзисторного ключа не отличается от расчета транзисторного ключа, приведенного в главе, посвященной схемам индикации.

Итак, в результате анализа устройств, которыми требуется управлять микропроцессорной системе, можно сделать вывод, что для управления всеми этими устройствами достаточно на выходе микропроцессорной системы создать логический уровень нуля или единицы. Это легко можно осуществить

записью в соответствующий триггер параллельного регистра нуля или единицы, а это означает, что управление исполнительными устройствами сводится к записи определенной комбинации нулей и единиц в особые ячейки памяти микропроцессора. Так как они осуществляют связь между микропроцессорным устройством и внешним миром, то эти ячейки памяти получили название — порты.

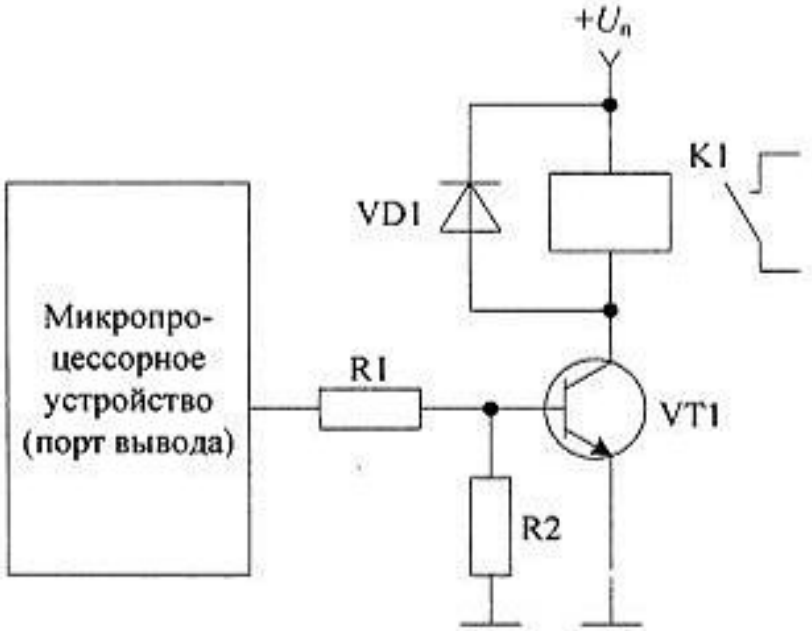


Рис. 19.1. Подключение внешнего устройства с индуктивной нагрузкой

Для успешного управления системами связи, кроме формирования управляющих воздействий, микропроцессорной системе требуется постоянно получать информацию об изменении внешней ситуации, режимов работы системы, а также информацию об исполнении управляющих команд.

При вводе информации из внешнего устройства возникают проблемы, подобные проблемам, возникающим при управлении исполнительными устройствами. Источники дискретной или аналоговой информации могут иметь различную физическую природу. Они могут находиться на значительном расстоянии от управляющей системы, иметь различное напряжение питания, но их данные должны быть безошибочно считаны управляющей программой микропроцессорной системы. Практически всегда при работе с внешними датчиками требуется гальваническая развязка между датчиками и управляющей микропроцессорной системой.

Для решения указанных проблем все датчики обычно выполняются так, что с точки зрения электрической схемы представляют собой контакты, работающие на замыкание и размыкание. При этом неважно, являются ли эти контакты механическими или представляют собой транзисторы с оптической или какой-либо другой развязкой.

В результате подобного подхода схема подключения практически любого датчика и механической кнопки не различаются. Со стороны микропроцессорного устройства требуется преобразовать замыкание/размыкание контактов в логические уровни, необходимые для правильной работы его управляющей программы. Эту функцию выполняет схема, приведенная на рис. 19.2.

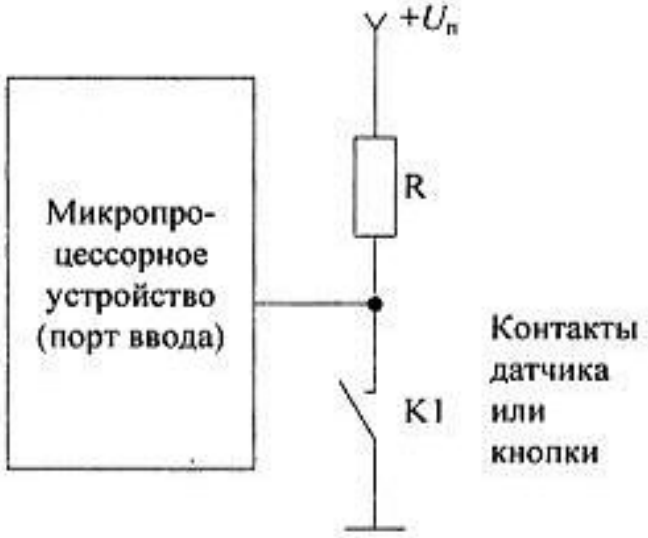


Рис. 19.2. Подключение источника дискретной информации с гальванической развязкой

В данной схеме при разомкнутых контактах датчика K1 напряжение от источника питания поступает непосредственно на вход цифрового устройства (в рассматриваемом случае — микропроцессорной системы). Так как входной ток этого устройства практически равен нулю, то и падения напряжения на резисторе R не происходит. В результате на вход микропроцессорного устройства поступает потенциал источника питания — уровень логической единицы.

При замыкании контактов датчика K1 через них будет протекать ток, ограниченный сопротивлением R1. На этом сопротивлении возникает падение напряжения $\Delta U = I_{конт} \times R1 = U_п / (R1 + R_{конт}) \times R1 \approx U_п$. В результате на входе микропроцессорного устройства напряжение уменьшится до потенциала корпуса и будет определяться падением напряжения на сопротивлении замкнутых контактов. При опросе этого вывода порта управляющей программой микропроцессора будет определен уровень логического нуля.

Итак, для ввода информации о состоянии системы в микропроцессорную систему управляющей программе достаточно просто прочитать соответствующую ячейку памяти, т. е. этот процесс с точки зрения программы практически не отличается от процесса управления внешними объектами.

Ну а теперь, после того как были рассмотрены вопросы, для чего нам требуется микропроцессорная система и как она взаимодействует с окружающей эту систему аппаратной средой, можно перейти к рассмотрению ее внутреннего устройства. Пожалуй, одним из самых значительных событий в развитии микропроцессорной техники было создание системной шины, позволяющей передавать информацию между различными блоками микропроцессорной системы.

Системная шина

Системная шина предназначена для обмена информацией между микропроцессором и любыми внутренними устройствами микропроцессорной системы (контроллера, сигнального процессора или компьютера). В качестве обязательных устройств, которые входят в состав любой микропроцессорной системы, можно назвать такие устройства, как ОЗУ, где хранятся обрабатываемые данные, ПЗУ, в котором обычно хранится управляющая программа и таблицы перекодировки, таймер, позволяющий микропроцессорной системе работать в реальном времени, и порты ввода-вывода, осуществляющие обмен данными с внешней средой. Структурная схема простейшей микропроцессорной системы, включающей перечисленные устройства, приведена на рис. 19.3.

В состав системной шины, в зависимости от конкретного типа микропроцессора, входит одна или несколько шин адреса, одна или несколько шин данных и шина управления. Несколько шин данных и адреса применяется для увеличения производительности системы и используется, как правило, в сигнальных процессорах. В универсальных процессорах и контроллерах обычно применяется одна шина адреса и одна шина данных даже при использовании микропроцессора, построенного по гарвардской структуре.

Иногда шину адреса и шину данных совмещают для экономии внешних выводов микропроцессора. При этом через одни и те же выводы поочередно передаются адреса и данные, однако в дальнейшем адреса и данные снова разделяют либо сразу же около микропроцессора, либо внутри микросхем ОЗУ и ПЗУ.

По шине данных информация передается либо к процессору, либо от процессора в зависимости от операции (записи или чтения), выполняемой микропроцессором в данный момент времени.

В любом случае все сигналы, необходимые для работы системной шины, формируются или опрашиваются микросхемой процессора, как это рассматривалось в предыдущей главе при изучении внутреннего устройства его операционного блока. Это означает, что без микропроцессора системная шина

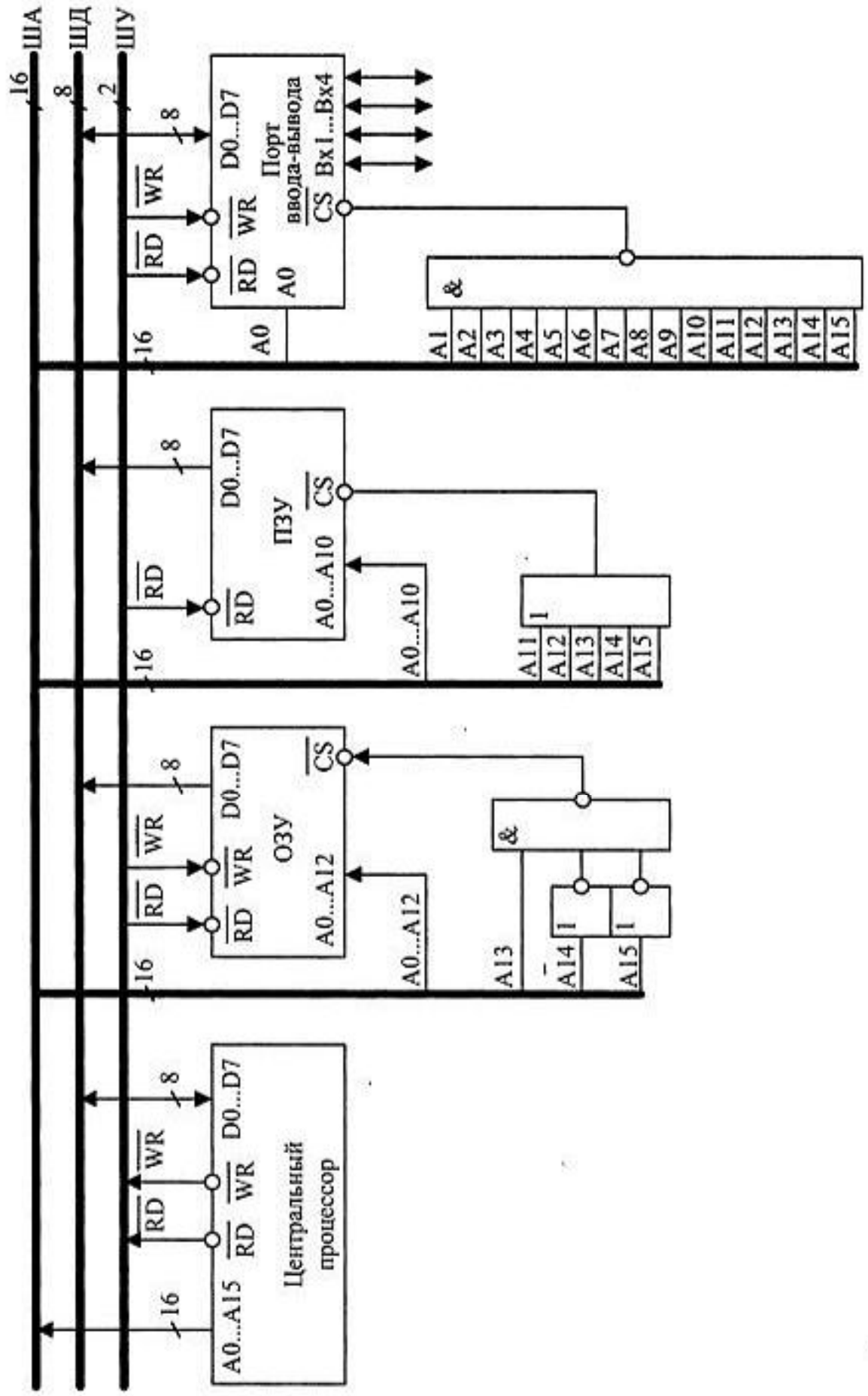


Рис. 19.3. Структурная схема простейшей микропроцессорной системы

функционировать не может. Более того, когда в микропроцессорной системе говорят об операции чтения, то предполагается, что это именно микропроцессор читает данные, если в этой системе говорят об операции записи, то запись данных осуществляет опять же микросхема микропроцессора.

Иногда, для увеличения скорости обработки информации, функции управления системной шиной берет на себя отдельная микросхема (например, контроллер прямого доступа к памяти или сопроцессор), и тогда операции записи или чтения будет осуществлять именно эта микросхема. В современных микроконтроллерах или сигнальных процессорах контроллер прямого доступа к памяти или сопроцессор могут находиться непосредственно в составе микросхемы.

Адресное пространство микропроцессорного устройства

При подключении различных устройств к системной шине возникает вопрос — как различать эти устройства между собой? С этой целью используют индивидуальный адрес для каждого устройства, подключенного к системной шине микропроцессора. Достаточно часто устройства, подключаемые к системной шине, занимают целый диапазон адресов. Так как обращение производится к каждой ячейке памяти устройства индивидуально, то возникает понятие диапазона адресов, занимаемого каждым устройством, и распределения адресного пространства микропроцессорного устройства в целом.

Адресное пространство микропроцессорного устройства изображается графически прямоугольником, одна из сторон которого соответствует разрядам адресуемой ячейки этого микропроцессора, а другая сторона — всему диапазону доступных адресов для этого же микропроцессора. Обычно в качестве минимально адресуемого элемента адресного пространства, доступного для микропроцессора, выбирается 8-разрядная ячейка (байт).

Диапазон доступных адресов микропроцессора определяется разрядностью шины адреса, которая обычно совпадает с разрядностью счетчика команд и разрядностью указателя данных этого микропроцессора. При этом минимальный номер ячейки памяти (адрес) будет равен нулю, а максимальный — определяется из формулы:

$$M = 2^N - 1,$$

где N — это количество разрядов шины адреса микропроцессора.

Для 16-разрядной шины адреса — это будет число 65 535 (64K). Адресное пространство микропроцессора с 16-разрядной шиной адреса приведено на рис. 19.4. Это наиболее распространенный размер адресного пространства

современных микроконтроллеров. Рис. 19.4 соответствует адресному пространству памяти микропроцессора, входящего в состав системы, структурная схема которой приведена на рис. 19.3.

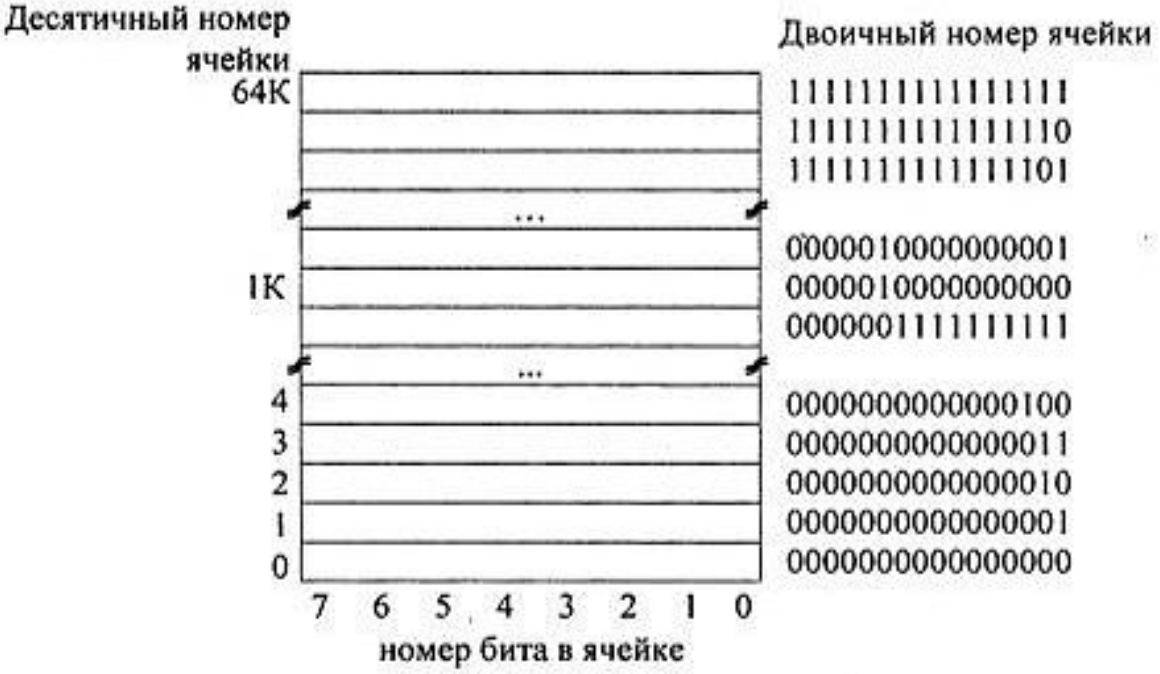


Рис. 19.4. Адресное пространство микропроцессора с 16-разрядной шиной адреса

На этом рисунке слева приведены адреса ячеек памяти в десятичном виде, справа — их двоичный эквивалент. Знание двоичного эквивалента адреса позволяет в ряде случаев упростить принципиальную схему микропроцессорного устройства. Для этой же цели в микропроцессорных системах часто пользуются числами, кратными степени числа 2. Например, широко используется число, ближайшее к числу тысяча — килобайт, равный 1024 байтам, которое является десятой степенью числа 2. Это число обозначается как 1К. Для микропроцессорных систем, более сложных по сравнению с системой, использованной нами в качестве примера микропроцессорной системы, применяются числа, близкие к миллиону — мегабайт, равный 1024 килобайтам, и к миллиарду — гигабайт, равный 1024 мегабайтам. Мегабайт обозначается как 1М, а гигабайт — как 1Г.

Различные адреса для ячеек памяти нам потребовались для того, чтобы отличать различные устройства, подключенные к микропроцессору, поэтому при построении микропроцессорной системы кроме адресного пространства нам требуется знать конкретные адреса (или группы адресов) устройств, подключенных к микропроцессору через системную шину микропроцессорной системы.

Распределением памяти микропроцессорной системы называют разбиение адресного пространства микропроцессора на несколько областей, каждая из

которых выделена для размещения ячеек какого-либо определенного элемента этой системы: ОЗУ, ПЗУ или внешних устройств. Часто его изображают в форме рисунка. Адресное пространство, соответствующее структурной схеме микропроцессорной системы, приведенной на рис. 19.3, изображено на рис. 19.5.

Рис. 19.5. Распределение памяти микропроцессора с 16-разрядной шиной адреса

Обычно адресное пространство распределяют одновременно с проектированием структурной схемы устройства и созданием дешифраторов адреса для каждого подключаемого к системной шине устройства. Это позволяет упростить его принципиальную схему. Давайте рассмотрим, как было получено распределение памяти, приведенное на рис. 19.5. Начнем распределение адресного пространства микропроцессорной системы с выделения диапазона адресов для микросхемы ПЗУ.

Микропроцессоры после включения питания и выполнения процедуры сброса всегда начинают выполнение программы с определенного адреса, чаще всего нулевого. Однако есть и исключения. Например, процессоры, на основе которых строятся универсальные компьютеры IBM PC или Macintosh, стартуют не с нулевого адреса. Исполняемая программа или ее загрузчик должны храниться в памяти микропроцессорной системы, которая не стирается при выключении питания, т. е. в ПЗУ. Таким образом, адрес, записываемый в счетчик команд процессора после выполнения его сброса, обязательно должен попадать в диапазон адресов, выделенных для размещения ПЗУ.

Выберем для построения микропроцессорной системы микросхему ПЗУ объемом 2 Кбайт, как это показано на рис. 19.3 и 19.5. При разработке операционного блока микропроцессора мы договорились, что микропроцессор после снятия сигнала сброса RESET начинает работу с нулевого адреса, поэтому для ПЗУ в адресном пространстве выделим номера ячеек, начиная с нулевого адреса.

Если мы просто подключим адресные входы ПЗУ к соответствующим проводникам адресной шины, то останется еще пять старших разрядов этой шины. Можно ли оставить эти разряды незадействованными? Нет! Ни в коем случае! Если не использовать эти разряды, то ячейки ПЗУ займут все адресное пространство микропроцессора. Для того, чтобы понять, почему это происходит, внимательно посмотрите на рис. 19.4 или 19.5. Обратите внимание на двоичное представление адреса ячеек памяти. Младшие одиннадцать разрядов адреса в адресном пространстве повторяются 32 раза! Поэтому для того, чтобы ПЗУ откликалось только на свои адреса, потребовалось ввести дополнительное устройство — дешифратор адреса.

Дешифратор адреса представляет собой комбинационное устройство, позволяющее выделить определенную комбинацию двоичных символов и подать разрешающий сигнал на вывод выбора кристалла ПЗУ. Для того чтобы нулевая ячейка ПЗУ оказалась расположенной по нулевому адресу адресного пространства микропроцессора, старшие разряды шины адреса (5 старших разрядов, начиная с разряда A11) должны быть равны 0. Именно на такую комбинацию следует построить дешифратор. Это выполняется при помощи внешнего дешифратора адреса, который в данном случае вырождается в 5-входовую схему "ИЛИ".

При использовании дешифратора адреса обращение микропроцессора за пределы нижней области 2 Кбайт не приведет к чтению ячеек ПЗУ, т. к. в этом случае на входе выбора кристалла CS уровень напряжения останется высоким (неактивным).

Теперь подключим к системной шине микропроцессорной системы оперативное запоминающее устройство. Для примера выберем микросхему объемом 8 Кбайт. Выбор любой из ячеек памяти этой микросхемы возможен при помощи 13-разрядного адреса, поэтому необходимо дополнительно декодировать сигналы трех старших линий 16-разрядной шины адреса. Так как начальные ячейки памяти адресного пространства уже заняты ПЗУ, то их использовать нельзя. Им соответствует значение старших разрядов адресной шины 000. Выберем для адресации ОЗУ комбинацию сигналов 001 и используем уже известные нам принципы построения схемы по произвольной таблице истинности. Дешифратор адреса выродится в данном случае в 3-входовую схему "И-НЕ" с двумя инверторами на входе. Принципиальная схема дешифратора адреса ОЗУ приведена на рис. 19.3. Этот дешифратор адреса обеспечивает нулевой уровень сигнала на входе CS микросхемы ОЗУ только при комбинации старших битов адреса A15 ... A13, равной 001.

✧ *Обратите внимание*: т. к. объем выбранной нами микросхемы ПЗУ меньше объема микросхемы ОЗУ, то между областями адресов микросхем ПЗУ и ОЗУ образовалось пустое пространство неиспользуемых адресов памяти микропроцессорной системы. Его можно было бы избежать при усложнении

схемы дешифратора адреса ОЗУ или применения дополнительного двоичного сумматора, однако стоит ли это делать? Это, кроме усложнения схемы устройства, приводит к снижению быстродействия микропроцессорной системы в целом. Если микропроцессорная система не будет усложняться, то зачем усложнять схему дешифратора адреса, а если будет, то, скорее всего, будет увеличен объем постоянного запоминающего устройства и тогда этот резерв адресов будет просто полезен.

Так как все микропроцессорные системы предназначены для обработки данных, поступающих извне, то в них должны присутствовать порты ввода-вывода. Как устроены порты ввода-вывода и как их применять при создании цифровых устройств, мы рассмотрим в последующих главах. Сейчас же для того, чтобы закончить распределение адресного пространства (и структурной схемы микропроцессорной системы), будем считать, что порт ввода-вывода отображается в адресное пространство микропроцессорного устройства как одиночная ячейка памяти, поэтому можно выбрать практически любой свободный адрес.

Ну а сейчас обратите внимание на то, что, т. к. порт представляет собой группу адресов, состоящую только из двух ячеек памяти, то приходится декодировать 15-разрядный адрес. При этом, чем больший номер будет иметь этот адрес, тем большее количество единиц будет присутствовать в двоичном представлении этого числа, и тем проще будет реализовываться дешифратор адреса соответствующего устройства.

Проще всего построить дешифратор самого старшего адреса — 65535. Это число соответствует шестнадцатеричному представлению 0FFFFh. В этом случае дешифратор адреса вырождается в обычную 15-входовую схему "15И-НЕ", поэтому и выберем данный адрес для размещения порта ввода-вывода. Именно такой дешифратор и изображен на структурной схеме, приведенной на рис. 19.3.

✧ Итак, *обратите внимание*, что выбор распределения ячеек памяти в адресном пространстве микропроцессора существенно влияет на структурную (а значит, и принципиальную) схему микропроцессорной системы. Ну а теперь рассмотрим подробнее внутреннее устройство параллельных портов микропроцессорных систем.

Принципы построения параллельного порта

Параллельные порты предназначены для обмена многоразрядными двоичными данными между микропроцессором и внешними устройствами. В качестве внешнего устройства, как это уже упоминалось ранее, может служить

любой объект управления или источник информации (различные кнопки, датчики, микросхемы приемников, синтезаторов частот, дополнительной памяти, исполнительные механизмы, двигатели, реле и т. д.). Иногда в качестве внешнего устройства может выступать другой компьютер или микропроцессорная система.

Параллельные порты позволяют согласовывать низкую скорость работы внешнего устройства и высокую скорость работы системной шины микропроцессора. С точки зрения внешнего устройства порт представляет собой обычный источник или приемник информации со стандартными цифровыми логическими уровнями (обычно ТТЛ), а с точки зрения микропроцессора — это ячейка памяти, куда можно записывать данные или где сама собой появляется информация. В зависимости от направления передачи данных параллельные порты называются портами ввода, портами вывода или портами ввода-вывода информации.

Параллельный порт вывода

Простейший порт вывода может быть построен на базе параллельного регистра, т. к. это устройство позволяет запоминать данные, передаваемые микропроцессором по системной шине, и хранить их до тех пор, пока на микропроцессорную систему подается питание. Все это время сигналы с выходов параллельного регистра подаются на внешнее устройство (или сразу на несколько внешних устройств). Принципиальная схема простейшего порта вывода с использованием параллельного регистра приведена на рис. 19.6.

Данные с системной шины микропроцессора записываются в параллельный регистр по сигналу записи "WR". Это становится возможным только при наличии нулевого уровня на выходе дешифратора адреса. Как мы уже рассматривали ранее, логическую функцию "2И" с инверсией сигналов на входе можно заменить на логическую функцию "2ИЛИ", что и показано на рис. 19.6.

Выходы "Q" регистра могут быть использованы как источники логических уровней для управления внешними устройствами. Этот регистр называется регистром данных порта вывода. Иногда, для упрощения схемы микропроцессорного устройства, в состав параллельного порта включаются силовые транзисторные ключи, подключенные к выходам регистра. Схема параллельного порта при этом несколько усложняется. Такой вариант построения параллельного порта будет рассмотрен в следующей главе.

В порт вывода, построенный по схеме, приведенной на рис. 19.6, возможна только запись двоичного многоразрядного числа. Так как в данной схеме нет цепей чтения содержимого регистра данных, то чтение его содержимого микропроцессором становится невозможным. Это не следует считать недостат-

ком, т. к. копия содержимого регистра данных порта вывода обычно хранится в ОЗУ.

Для отображения регистра данных параллельного порта вывода информации только в одну ячейку памяти адресного пространства микропроцессорного устройства, а не во все адресное пространство микропроцессорной системы, совместно с портом вывода всегда используется дешифратор адреса. Разработка дешифратора адреса и вопросы выбора конкретного адреса для параллельного порта подробно обсуждались ранее при рассмотрении распределения адресного пространства микропроцессорного устройства, поэтому здесь этот вопрос рассматриваться не будет.

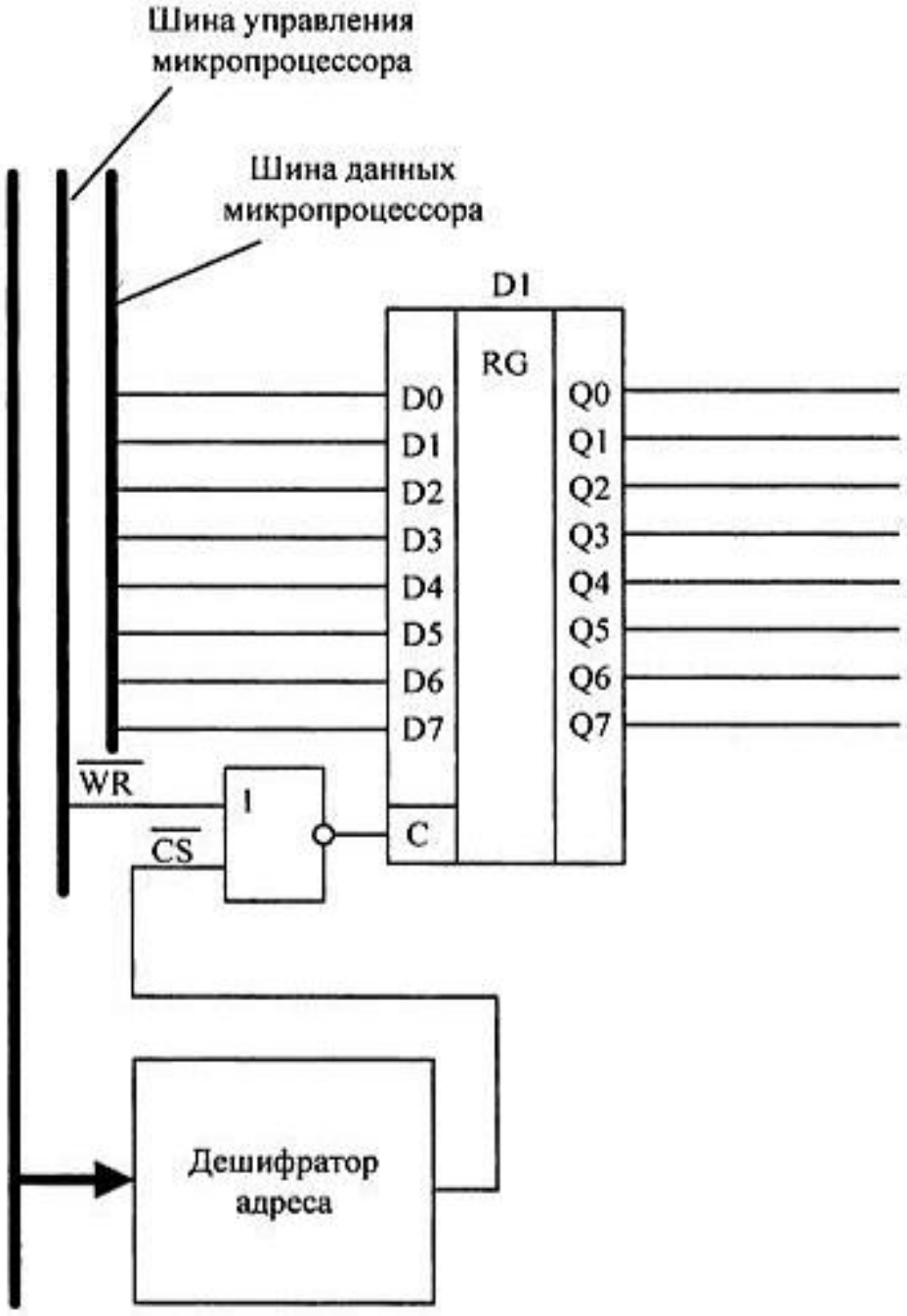


Рис. 19.6. Принципиальная схема порта вывода

Параллельный порт ввода

Задача порта ввода дискретной информации является обратной по отношению к задаче порта вывода. В данном случае логические сигналы, поступающие с внешнего устройства, необходимо подключать к шине данных системной шины по сигналу запроса RD от микропроцессора. Все остальное время сигналы, формируемые внешним устройством, не должны мешать нормальной работе микропроцессорной системы.

В качестве порта ввода обычно используются схемы с третьим (Z) состоянием. Микросхема, объединяющая несколько таких элементов, называется шинным формирователем. Из порта ввода возможно только чтение информации. Структурная схема простейшего *порта ввода* приведена на рис. 19.7. Для построения порта ввода, выход шинного формирователя подключается к внутренней шине данных, а к его входу подводятся сигналы, которые требуется контролировать микропроцессорной системой. Значение сигнала

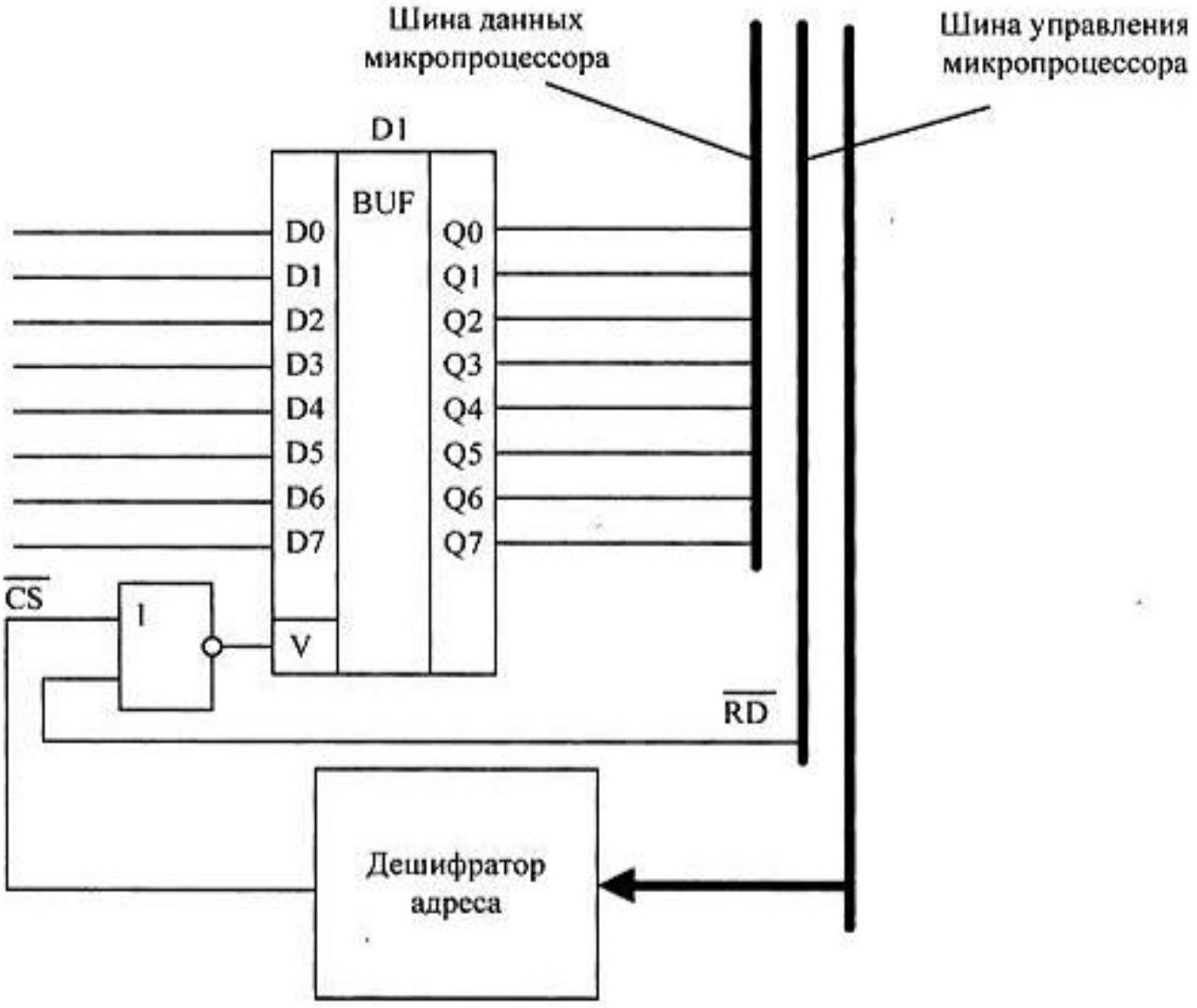


Рис. 19.7. Структурная схема порта ввода

с внешнего вывода порта передается на шину данных (считывается) по управляющему сигналу "RD".

Для отображения шинного формирователя порта ввода только в один адрес в пространстве адресов микропроцессорного устройства совместно с портом ввода используют дешифратор адреса. Так как с точки зрения программиста эта ячейка памяти ничем не отличается от регистра данных порта вывода, то по аналогии она называется регистром данных порта ввода.

Параллельный порт ввода-вывода

Параллельные порты выпускаются в составе универсальных микросхем, но на заводе, где производятся эти микросхемы, неизвестно, сколько конечному пользователю на самом деле потребуется линий ввода информации и сколько понадобится линий вывода двоичной информации. Количество же внешних выводов у микросхемы ограничено. Поэтому обычно в одной универсальной микросхеме размещаются и порт ввода, и порт вывода информации одновременно. Конечному пользователю при этом предоставляют возможность настройки линий порта на ввод или вывод информации. Такие порты называются *портами ввода-вывода* информации. Структурная схема параллельного порта ввода-вывода приведена на рис. 19.8.

Теперь обратите внимание, что из порта ввода возможно только чтение, а в порт вывода возможна только запись данных, поэтому в адресном пространстве микропроцессора для порта ввода и порта вывода можно назначать один и тот же адрес. В схеме на рис. 19.8 регистр данных и шинный формирователь тоже используют один и тот же адрес. Это обеспечивается подачей на вход логических элементов "3И-НЕ" одного и того же потенциала из линии A0. Какое из устройств в данный момент будет подключено к шине данных, определяется сигналами записи WR и чтения RD.

Для подключения портов ввода или портов вывода информации к внешним выводам микросхемы в схеме, приведенной на рис. 19.8, используется коммутатор. Именно он определяет направление передачи данных. Управляет этим коммутатором еще один (внутренний) параллельный порт вывода, регистр данных которого называется регистром управления параллельного порта ввода-вывода. Регистру управления и регистрам данных порта ввода-вывода обычно назначаются соседние адреса. На приведенной схеме это обеспечивается инверсией линии младшего разряда адреса A0.

В схеме можно было обеспечить чтение внутренней информации регистра управления, подключив еще один шинный формирователь (что обычно и делается), однако это бы уменьшило наглядность рисунка, поэтому эта возможность в схеме, приведенной на рис. 19.8, отсутствует.

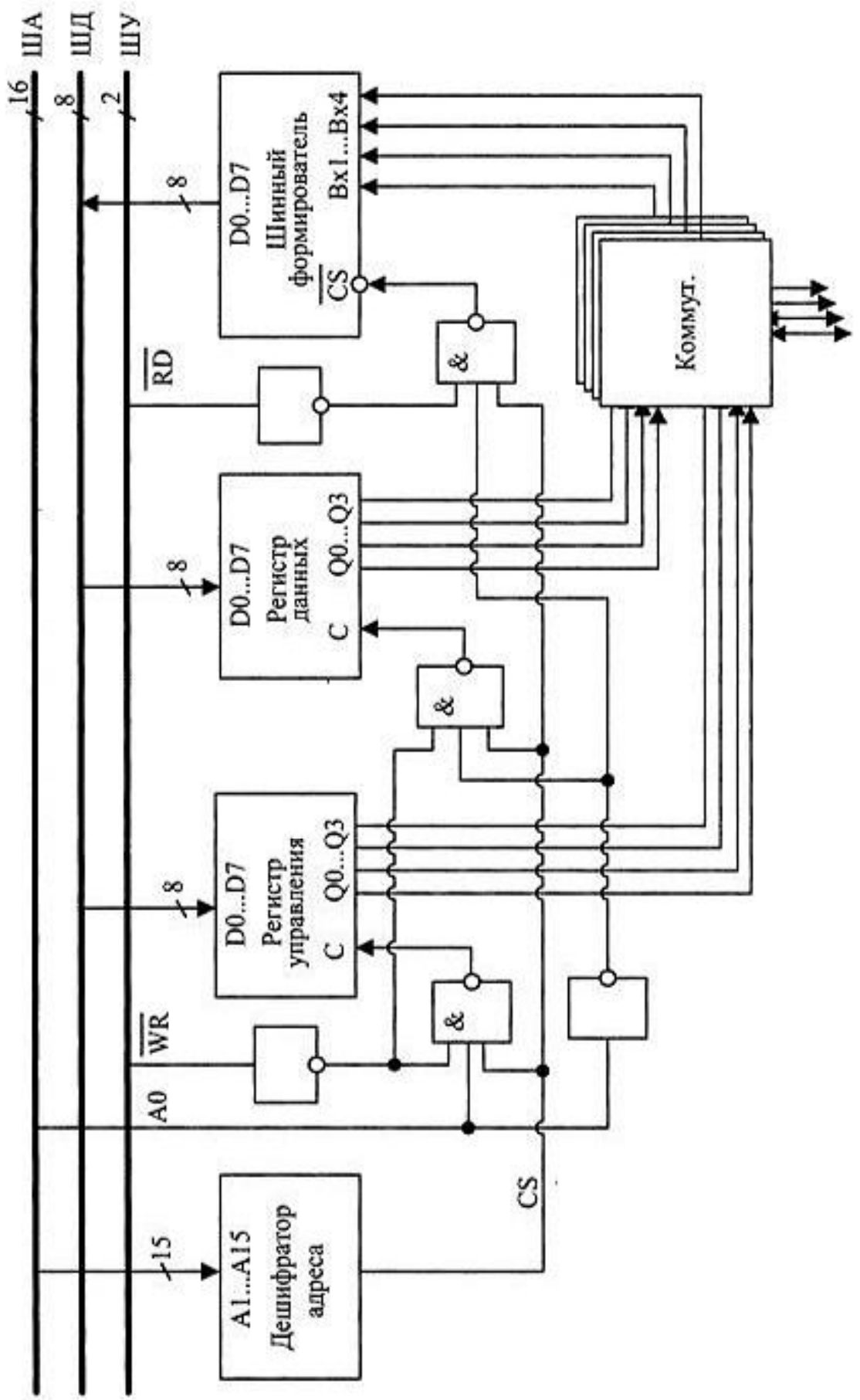


Рис. 19.8. Структурная схема параллельного порта ввода-вывода

Следует отметить, что обычно то, что регистр управления подключается через внутренний параллельный порт, не указывается, и вся схема в целом называется портом ввода-вывода.

В некоторых микропроцессорах для портов ввода-вывода выделяется отдельное адресное пространство. В этом случае для записи в порт и для чтения из порта используются отдельные сигналы чтения и записи. Чаще всего они называются "IOWR#" и "IORD#". В этом названии буквы IO означают ввод-вывод (input-output), а знак # — использование активного нулевого потенциала (как в рассмотренных ранее сигналах записи и чтения памяти микропроцессорной системы). Применение подобного символа позволяет избежать использования черты над названием цепи, что неприменимо для ряда современных редакторов схем.

Примеры использования параллельных портов

Одной из основных функций параллельного порта является ввод информации с кнопок или механических датчиков, однако эти источники информации не формируют логические потенциалы непосредственно. Для преобразования замыкания и размыкания их контактов в логические потенциалы обычно используют резистор. Подобная схема приведена на рис. 19.2.

Если в схеме, приведенной на рис. 19.2, поменять резистор и кнопку местами, то уровню логической единицы будет соответствовать замыкание контактов, а уровню логического нуля — размыкание контактов кнопки. В остальном работа схемы не изменится, тем не менее, обычно применяется схема, приведенная на рис. 19.2, т. к. она позволяет заземлить один из контактов кнопки или датчика.

На рис. 19.2 приведена схема подключения одиночной кнопки, однако к восьмиразрядному порту ввода можно подключить одновременно до восьми кнопок. При этом каждой кнопке будет соответствовать отдельный бит в восьмиразрядном слове, считанном из регистра данных параллельного порта. При необходимости ввода информации с большего количества кнопок или датчиков можно ввести в схему микропроцессорной системы еще один или несколько параллельных портов ввода.

Ввод информации с клавиатуры

Иногда требуется вводить информацию с очень большого количества кнопок. В этом случае для уменьшения количества линий ввода параллельного порта используется клавиатура, представляющая собой двухмерную матрицу кнопок, организованных в ряды и колонки. Для подключения клавиатуры обычно используется два порта: порт ввода и порт вывода (или порт ввода-вывода с соответствующей настройкой направления передачи данных). Схема под-

ключения шестнадцатикнопочной клавиатуры к портам ввода и вывода микропроцессорного устройства приведена на рис. 19.9.

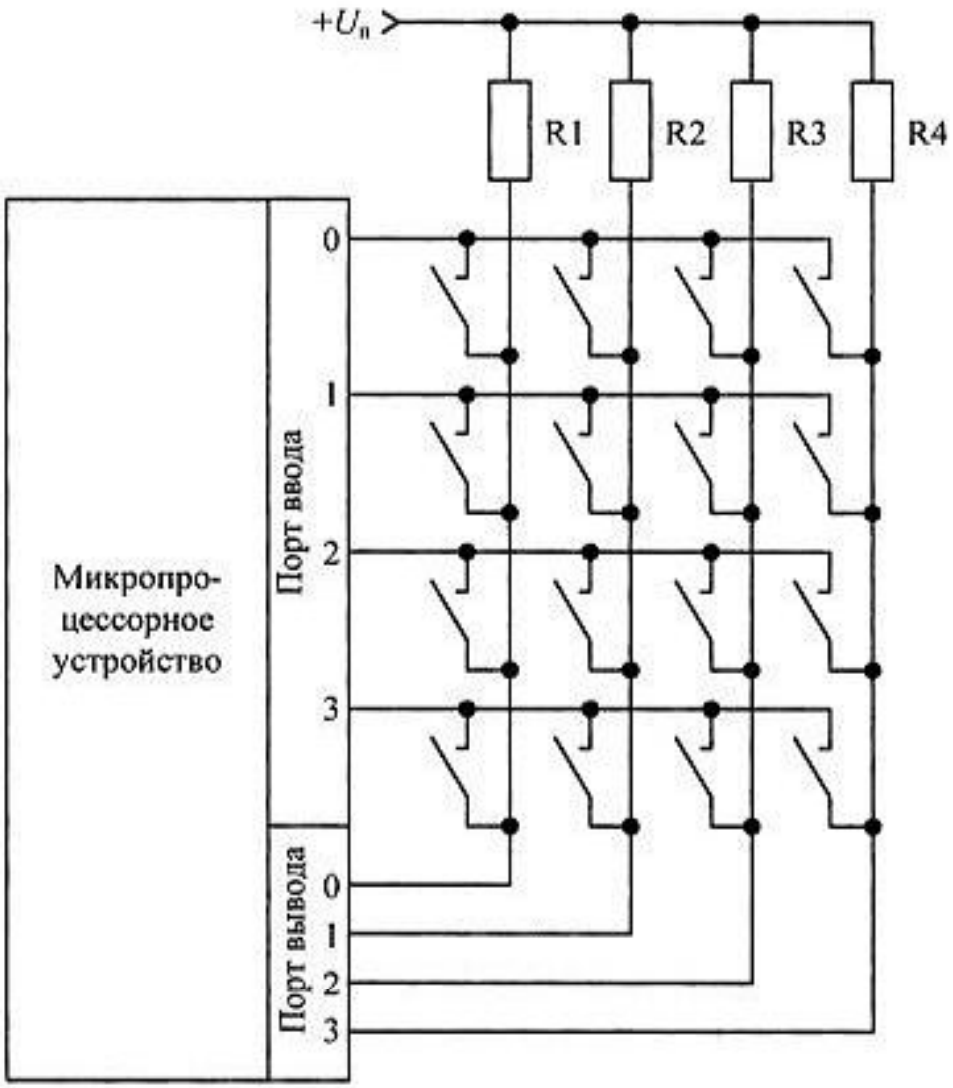


Рис. 19.9. Подключение клавиатуры к микропроцессорному устройству

Подключение клавиатуры к параллельному порту ввода отличается от схемы подключения одиночной кнопки тем, что потенциал общего провода на контакты опрашиваемых кнопок подается не непосредственно, а через нижний транзистор параллельного порта вывода. В каждый момент времени потенциал логического 0 подается на нижние контакты только одного столбца кнопок, а, значит, к корпусу оказывается подключенным именно этот столбец кнопок. Двоичные сигналы, образующиеся при этом на строках клавиатуры, считываются через порт ввода микропроцессорной системы.

Кнопки, не подключенные к корпусу через порт вывода, на формирование логических сигналов, поступающих на входы параллельного порта ввода, влияния не оказывают, т. к. ток через них не протекает ни при замыкании, ни при размыкании их контактов.

Для опроса состояния всех кнопок клавиатуры необходимо поочередно подать нулевой потенциал на все столбцы клавиатуры и считать логические сигналы с входов порта ввода. Временная диаграмма напряжений, присутствующих на порту вывода при выполнении программы опроса клавиатуры, приведена на рис. 19.10.

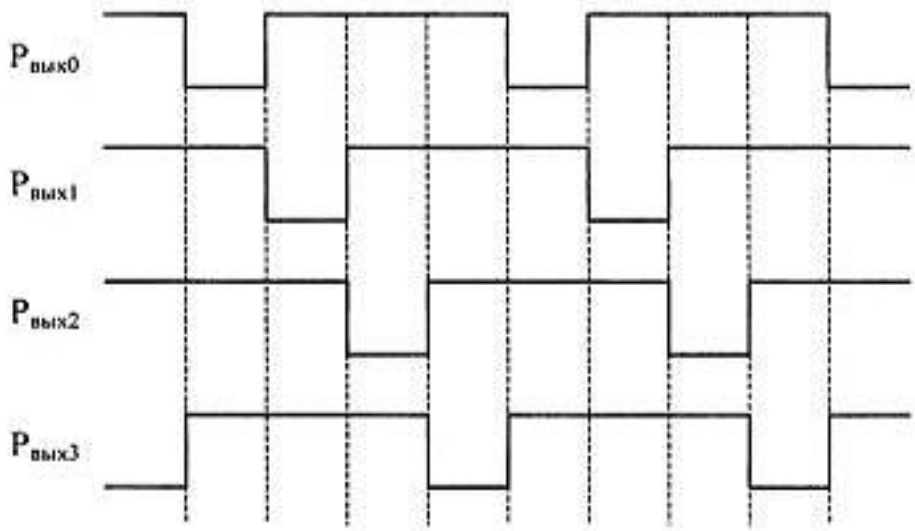


Рис. 19.10. Временные диаграммы напряжений на выводах порта вывода при опросе кнопок клавиатуры

Как видно из приведенных на рис. 19.10 временных диаграмм сигналов опроса колонок кнопок, управляющая программа микропроцессорной системы осуществляет непрерывный опрос состояния всех кнопок клавиатуры. Период опроса кнопок клавиатуры определяет время реакции системы на изменение их состояния. Работа с подобной программой ввода данных с клавиатуры будет более подробно рассмотрена в последующих главах.

В рассмотренном случае для подключения 16 кнопок потребовалось только 8 линий ввода-вывода. При большем количестве кнопок выигрыш по количеству выводов параллельных портов может оказаться еще большим. Например, при подключении к микропроцессорной системе по рассмотренной схеме 120 кнопок потребуется только 15 линий ввода-вывода. Именно поэтому подобные схемы подключения кнопок получили широкое распространение в цифровой технике.

Обмен данными между микропроцессорами при помощи параллельных портов

Параллельные порты, предназначенные для обмена данными между компьютерами, или компьютером и принтером, устроены несколько иначе по сравнению с уже рассмотренными простейшими видами параллельных портов

ввода и вывода. Основным отличием этого вида обмена информацией от уже рассмотренных примеров является большой объем передаваемых данных. При обмене данными между компьютерами в микропроцессорную систему поступает не один или даже несколько байтов двоичной информации, как это осуществляется при вводе информации с датчиков или клавиатуры, а длинные последовательности байтов, которые вводятся в нее через один и тот же параллельный порт.

Для того чтобы отличать один передаваемый (или принимаемый) байт от другого вводится специальный сигнал синхронизации CLK (сокращение от слова clock — синхронизация). (Иначе невозможно было бы передавать одинаковые байты, как, например, при передаче нескольких пробелов в текстовой информации.) Временная диаграмма обмена данными через параллельный порт приведена на рис. 19.11.

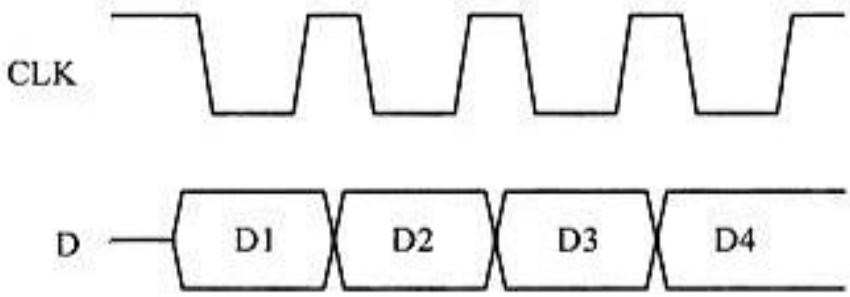


Рис. 19.11. Временная диаграмма сигналов обмена данными через параллельный порт

Для формирования сигнала синхронизации байта CLK можно воспользоваться еще одним параллельным портом, и получить его программным способом. Для этого в один из его битов потребуется последовательно записывать логический "0" и логическую "1". В результате этих действий на соответствующем выводе дополнительного параллельного порта появится импульс. Однако программная реализация импульса синхронизации CLK значительно уменьшит скорость обмена информацией через параллельный порт, т. к. в этом случае для передачи одного байта данных потребуется три команды записи в параллельный порт вместо одной. Поэтому обычно в параллельном порту, предназначенном для обмена большими объемами информации, сигнал синхронизации CLK формируется аппаратно из сигнала "WR#" при записи очередного байта в его регистр данных.

Для формирования сигнала синхронизации CLK в состав подобного параллельного порта вводится отдельная цифровая схема. Создание подобной схемы является тривиальной задачей, поэтому в данной главе рассматриваться не будет. Принципы создания подобных схем мы уже обсудили ранее при изучении схемотехники цифровых устройств.

В данной книге рассмотрены основы работы параллельного порта. Тот, кому интересно познакомиться более детально с особенностями работы параллельных портов, может обратиться к специализированной литературе, посвященной компьютерной и микроконтроллерной тематике [14 ... 17].

Кроме параллельных портов в микропроцессорной технике для ввода, вывода и обмена цифровой информацией широко используются последовательные порты, поэтому остановимся подробнее на принципах работы этого вида цифровых устройств.

Принципы построения последовательного порта

Последовательные порты предназначены для обмена информацией между микропроцессорами, а также между микропроцессорами и внешними устройствами, если критично количество соединительных проводов. Чаще всего в качестве внешних устройств для микропроцессора выступают отдельные микросхемы. В настоящее время широко используются две группы последовательных портов:

- синхронные последовательные порты;
- асинхронные последовательные порты.

В предыдущих главах мы уже рассматривали устройство синхронных последовательных портов в качестве примеров реализации цифровых устройств. В данной главе мы рассмотрим особенности подключения этих устройств к системной шине микропроцессора.

Синхронные последовательные порты

Начнем с простейшего вида синхронного последовательного порта — SSI-интерфейса. В этом случае для передачи последовательного потока данных достаточно только одного провода. Передаваемая и принимаемая информация обычно представляется в виде однобайтовых или многобайтовых слов (кадров). Вес каждого бита в кадре различен, поэтому кроме сигнала кадровой синхронизации FS (frame synchronization), аналогичной байтовой синхронизации для параллельного порта, требуется дополнительный сигнал битовой синхронизации SCLK (serial clock). Этот сигнал позволяет однозначно определять номер каждого бита в передаваемом кадре. Временные диаграммы сигналов для случая передачи восьмиразрядного кадра по синхронному последовательному порту SSI приведены на рис. 19.12.

На временной диаграмме, приведенной на рис. 19.12, показаны два синхросигнала: тактовой синхронизации CLK и кадровой синхронизации FS. По-

лярность сигналов синхронизации зависит от конкретного типа применяемых микросхем, поэтому в большинстве последовательных портов возможна настройка полярности сигналов синхронизации. Напомню, что эти сигналы должны быть жестко связаны с сигналом дискретизации аналогового сигнала, а его спектральные характеристики определяют параметры преобразованного в цифровую форму аналогового сигнала. Причины этого мы уже рассматривали подробно в предыдущих главах, посвященных вопросам аналого-цифрового преобразования. Именно поэтому сигналы синхронизации обязательно должны формироваться в аналого-цифровом или цифроаналоговом преобразователе из сигнала высокостабильного малошумящего генератора опорной частоты.

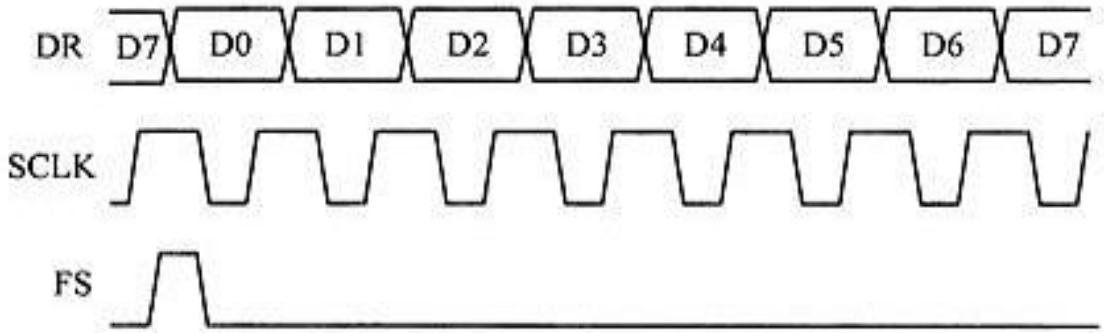


Рис. 19.12. Временные диаграммы сигналов при передаче одного кадра двоичной информации по последовательному интерфейсу SSI

Как мы уже выяснили ранее, любое устройство, подключаемое к системной шине, не должно нарушать ее работу. Это означает, что для доступа к системной шине в состав последовательного порта обязательно должен входить параллельный порт. То, что сигналы синхронизации формируются не микропроцессором, означает, что его последовательный порт SSI должен быть подчиненным устройством. Упрощенная принципиальная схема синхронного последовательного порта SSI приведена на рис. 19.13.

Как же микропроцессор "узнает", что последовательный порт завершил прием информации? Для этого можно выделить отдельный триггер, при опросе содержимого этого триггера микропроцессор и определит, закончился ли прием очередного кадра входного сигнала. Содержимое подобных триггеров, отображающее внутреннее состояние цифровых устройств, обычно называется флагами.

В состав последовательного порта, приведенного на рис. 19.13, входит универсальный последовательно-параллельный регистр D1. Он обеспечивает преобразование последовательного кода, поступающего из SSI-шины в параллельный вид. После записи в этот регистр последнего бита в триггер D7 заносится единичное значение (устанавливается флаг готовности). Его можно

было бы сформировать счетчиком импульсов тактовой синхронизации, но в данной схеме единичный потенциал заносится в младший бит сдвигового регистра D1 сигналом кадровой синхронизации.

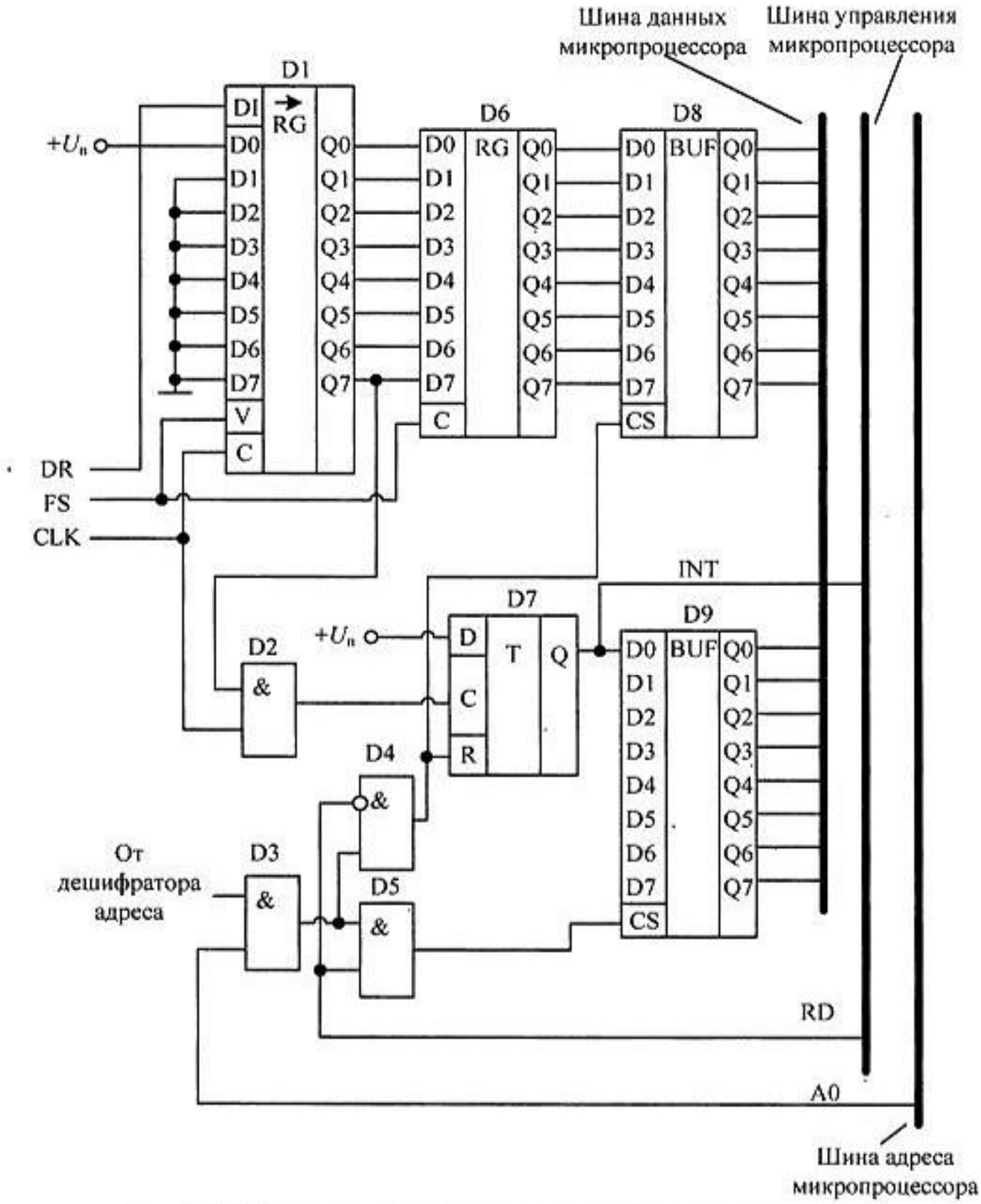


Рис. 19.13. Упрощенная схема синхронного последовательного порта

После заполнения всего последовательного регистра этот потенциал поступает на вход логического элемента "2И" D2. В результате в триггер D7 записывается единица (устанавливается флаг). Этот флаг может быть обнаружен либо при опросе содержимого триггера D7 микропроцессором, либо этот сигнал может непосредственно вызвать прерывание работы программы микропроцессора по входу запроса прерывания INT.

Одновременно с установкой флага готовности принятый кадр из последовательного регистра копируется в параллельный регистр (регистр данных D6). В последовательный регистр D1 при этом записывается комбинация 00000001, которая используется в качестве счетчика битов, и начинается новый цикл приема данных по интерфейсу SSI. Микропроцессор должен за время приема следующего кадра успеть обнаружить окончание приема предыдущего и считать его. В противном случае предыдущая информация будет замещена новой.

Как это уже упоминалось ранее, флаг готовности порта рассматривается как отдельный бит восьмиразрядного слова состояния порта и размещается обычно в соседнем по отношению к регистру данных адресу. Для этого в состав схемы введен дополнительный шинный формирователь D9 и заведен младший разряд шины адреса. Логические элементы D4 и D5 в зависимости от логического уровня на этом разряде подключают к шине данных либо регистр данных последовательного порта D6, либо триггер D7, содержащий его флаг готовности.

При обращении центрального процессора к регистру данных последовательного порта D6 вырабатывается сигнал чтения RD, который подается на вход CS шинного формирователя D8. Одновременно этот же сигнал обнуляет триггер D7, сбрасывая тем самым флаг готовности данных последовательного порта.

В качестве примера применения синхронного последовательного порта SSI в составе микропроцессорной системы, на рис. 19.14 приведена схема подключения аналого-цифрового преобразователя AD7853 фирмы Analog Devices к синхронному последовательному порту сигнального процессора ADSP-2189 той же фирмы.

Подключение других АЦП, ЦАП или кодеков производится подобным образом. Вариант подключения АЦП с входом синхронизации последовательного порта, работающим от внешнего генератора, рассматривался в предыдущих главах, посвященных примерам реализации цифровых устройств.

В синхронном последовательном порту информация SSI передается непрерывно, что, конечно, удобно для устройств с непрерывным потоком информации, как, например, в кодеках речи. Но существуют устройства, к которым необходимо обращаться только периодически, как, например, синтезаторы

частоты, микросхемы приемников, блоков цветности телевизоров, микросхем памяти данных и многие другие устройства. В этих случаях используются другие виды синхронных последовательных портов, такие как SPI и I^2C. Даже микросхемы АЦП и ЦАП, кроме порта передачи данных SSI, обычно оборудуются подобными последовательными портами, предназначенными для управления ими. Временные диаграммы сигналов SPI-интерфейса приведены на рис. 19.15.

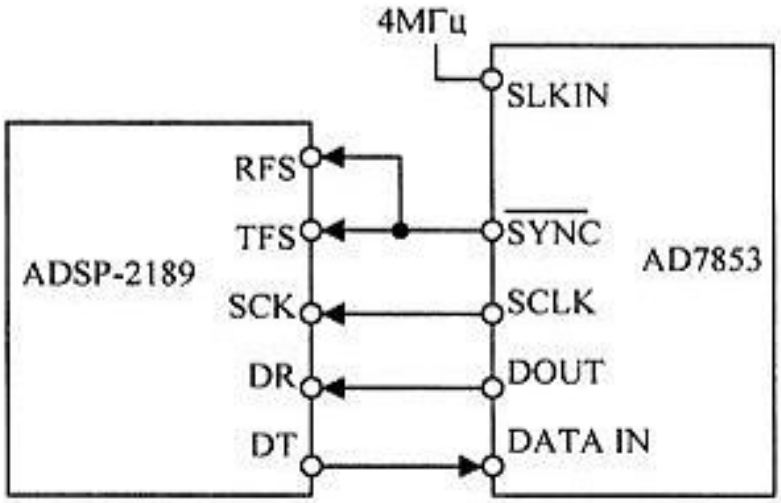


Рис. 19.14. Схема подключения кодека к синхронному последовательному порту

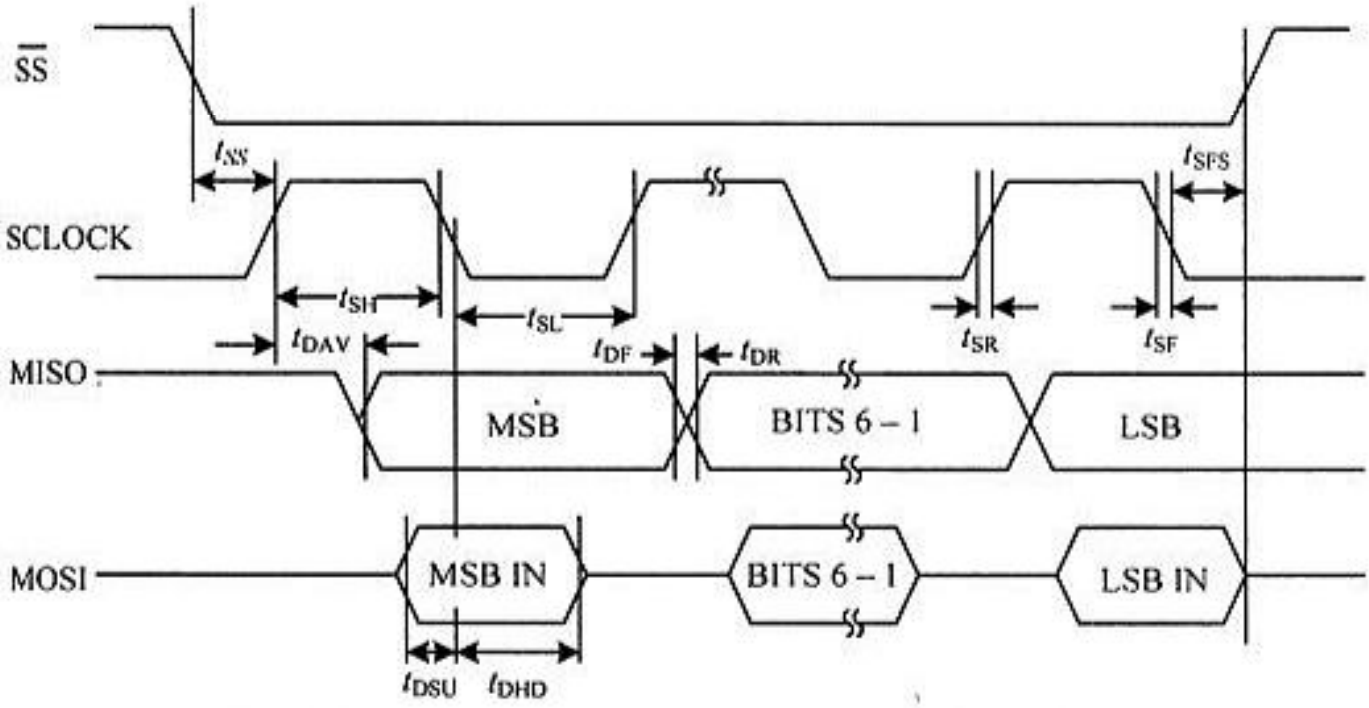


Рис. 19.15. Временные диаграммы сигналов SPI-интерфейса

Основное отличие этого синхронного последовательного интерфейса от SSI заключается в том, что сигнал тактовой синхронизации передается только в

момент действия импульса кадровой синхронизации. Активный уровень сигнала кадровой синхронизации длится до окончания передачи последнего бита в передаваемом кадре. По одним и тем же линиям передачи данных: MISO (вход для главного, выход для ведомого) и MOSI (выход для главного, вход для ведомого), может передаваться информация к совершенно различным микросхемам. Выбор среди микросхем, подключенных к одному и тому же порту, той, для которой предназначена информация, производится сигналом SS (выбор ведомого). В SPI-интерфейсе в приемнике не требуется счетчик тактовых импульсов. Запись принятой информации в параллельный регистр данных производится по окончании кадрового импульса.

В *главе 11* на рис. 11.5 и 11.7 были приведены схемы ведущего и подчиненного SPI-портов. Для подключения этих схем к системной шине микропроцессорной системы достаточно дополнить приведенные схемы схемой параллельного порта ввода-вывода (последовательный порт выступает в данном случае в качестве внешнего устройства), поэтому приводить полную схему подобного порта не имеет смысла. Общая схема собирается подобно детским кубикам. Более того! Подробная схема получится достаточно громоздкой и вместо иллюстрации материала может запутать читателя.

Кроме пакетного режима работы, еще одной особенностью SPI-интерфейса является то, что по этому интерфейсу можно подключать к микропроцессорной системе одновременно несколько различных устройств. При этом обращение к данным устройствам должно производиться в различные моменты времени. Подобная схема подключения двух устройств, управляемых по SPI-интерфейсу, приведена на рис. 19.16.

В схеме, приведенной на рис. 19.16, конкретно к какому из подключенных устройств обращается микропроцессорная система, определяется сигналами выбора ведомого SS1 и SS2. При обращении микропроцессорной системы к устройству 1 низкий потенциал появится на выводе SS1, а при обращении к устройству 2 — на выводе SS2. Таким образом по линиям обмена данными MOSI и SCLK могут передаваться команды управления, предназначенные для различных устройств.

Если в разрабатываемой системе используется много микросхем, требующих управления от микропроцессора, то при применении SPI-интерфейса количество линий выбора ведомого SS_n становится значительным, поэтому в таких случаях может быть использован еще один вид синхронного последовательного интерфейса: I^2C.

Временная диаграмма сигналов этого интерфейса приведена на рис. 19.17. В I^2C-интерфейсе прием и передача данных, а также передача адреса микросхемы и адреса регистра внутри микросхемы, к которому осуществляется обращение, производится по одной и той же линии данных SDA. Для подклю-

чения к этой линии используются микросхемы с открытым коллектором. Нагрузкой для всех микросхем, подключенных к линии SDA, служит внешний резистор. Естественно, что скорость передачи данных по такому интерфейсу будет ниже, чем в случае применения SPI-интерфейса.

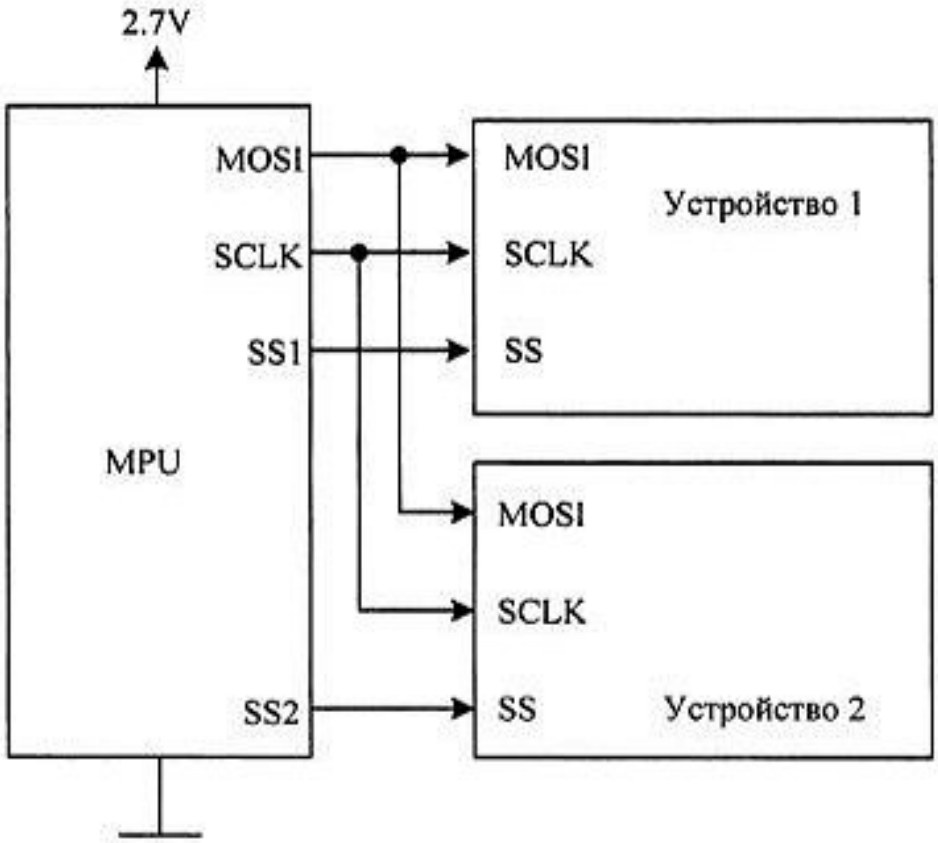


Рис. 19.16. Подключение двух SPI-устройств

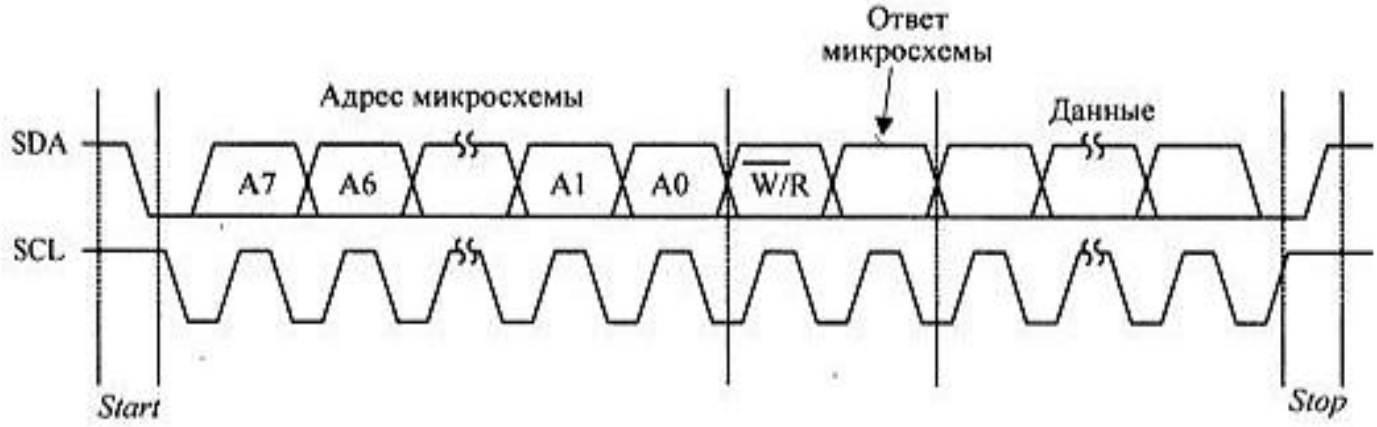


Рис. 19.17. Временная диаграмма I^2C-интерфейса

Сигнал тактовой синхронизации в I^2C-шине передается по линии SCL. Начало работы с микросхемой обозначается особой комбинацией сигналов SDA и SCL (переход 1-0 SDA при высоком уровне SCL), которая называется усло-

вием старта. Эта же комбинация одновременно осуществляет кадровую синхронизацию передачи данных. Завершение работы с микросхемой обозначается еще одной комбинацией сигналов SDA и SCL — переходом 0–1 SDA при высоком уровне SCL. В качестве примера микросхем, использующих интерфейс I^2C, можно назвать микросхемы EEPROM серии 24cXX.

Обмен данными по интерфейсу I^2C подчиняется достаточно сложным алгоритмам и требует отдельного рассмотрения. Эти вопросы выходят за рамки тем, рассматриваемых в данной книге. Описание сигналов, используемых в интерфейсе I^2C, приведены для полноты обзора синхронных последовательных портов, применяемых в микропроцессорных системах.

На этом можно завершить краткий обзор наиболее распространенных синхронных последовательных портов. Еще одним достаточно распространенным видом последовательных портов являются асинхронные последовательные порты.

Асинхронные последовательные порты

Рассмотренные в предыдущем разделе синхронные последовательные порты позволяют достигнуть больших скоростей передачи данных, однако линия, по которой ведется передача синхросигнала, практически не несет информации. Такой сигнал можно было бы сформировать и на приемном конце линии передачи, если заранее договориться о скорости передачи.

В настоящее время используются стандартные скорости передачи, кратные скорости 1200 бит/с. При этом масштабирование может проводиться как в сторону увеличения скорости обмена, так и в сторону уменьшения скорости обмена двоичной информацией. Например, стандартной скоростью передачи последовательного порта будет скорость 2400 и 4800 бит/с. Стандартными же скоростями обмена будут 600 и 300 бит/с.

Проблема, возникающая при асинхронном способе обмена данными, — это невозможность добиться от двух внутренних генераторов, осуществляющих синхронизацию передачи данных на приемном и передающем концах линии, одинаковой частоты и фазы генерируемых сигналов. Проблема решается принудительной синхронизацией тактового генератора на приемном конце при помощи особого условия начала асинхронной передачи — стартового бита.

Все время, пока не ведется передача информации, на линии присутствует стоп-сигнал единичного уровня. Перед началом передачи каждого байта передается старт-бит, сигнализирующий приемнику о начале посылки данных, за которым следуют информационные биты. Стартовый бит всегда передается нулевым уровнем с длительностью, как у информационных битов. Вре-

менная диаграмма передаваемых сигналов при асинхронной передаче данных приведена на рис. 19.18.

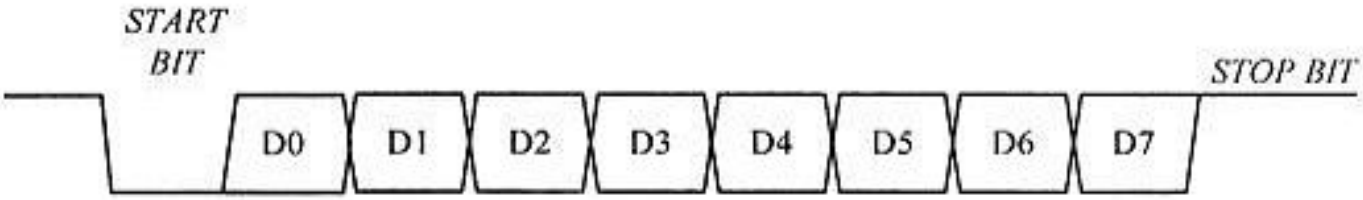


Рис. 19.18. Временная диаграмма передаваемых сигналов при асинхронной передаче

Внутренний генератор синхронизации приемника использует счетчик-делитель опорной частоты, обнуляемый в момент приема начала старт-бита. Этот счетчик генерирует внутренние стробы, по которым приемник фиксирует последующие принимаемые биты. В идеале стробы располагаются в середине битовых интервалов, что позволяет принимать данные и при незначительном рассогласовании скоростей приемника и передатчика. Очевидно, что при передаче 8 бит данных, одного контрольного и одного стоп-бита, предельно допустимое рассогласование скоростей, при котором данные будут распознаны верно, не может превышать 5%.

В некоторых случаях после передачи битов данных может передаваться бит паритета (четности). Завершается передача данных стоп-сигналом. Минимальная длительность стопового сигнала должна быть 1,5 длительности информационных битов, но обычно используют паузу между соседними пакетами данных, равную двум длительностям информационного бита.

Формат асинхронной посылки позволяет выявлять возможные ошибки передачи. Если принят перепад, сигнализирующий о начале посылки, а по стробу старт-бита зафиксирован уровень логической единицы, старт-бит считается ложным и приемник снова переходит в состояние ожидания. Об этой ошибке приемник может и не сообщать.

Если во время, отведенное под стоп-бит, обнаружен уровень логического нуля, фиксируется ошибка стоп-бита.

Если применяется контроль четности, то после посылки бит данных передается *контрольный бит.* Этот бит дополняет количество единичных бит данных до четного или нечетного числа в зависимости от принятого соглашения. Прием байта с неверным значением контрольного бита приводит к фиксации ошибки.

Наиболее распространенным в настоящее время является последовательный асинхронный порт, работающий по стандарту RS-232. Временная диаграмма сигнала на выводе RxD этого порта приведена на рис. 19.19.

Рис. 19.19. Временная диаграмма передаваемых сигналов интерфейса RS-232

Порт RS-232 использует уровни передачи ±10 В. Использование трехуровневых сигналов в этом порту позволяет контролировать обрыв линии передачи между передатчиком и приемником данного асинхронного последовательного порта.

Существует ряд международных стандартов на асинхронные последовательные интерфейсы: RS-232C, RS-423A, RS-422A и RS-485. На рис. 19.20 приве-

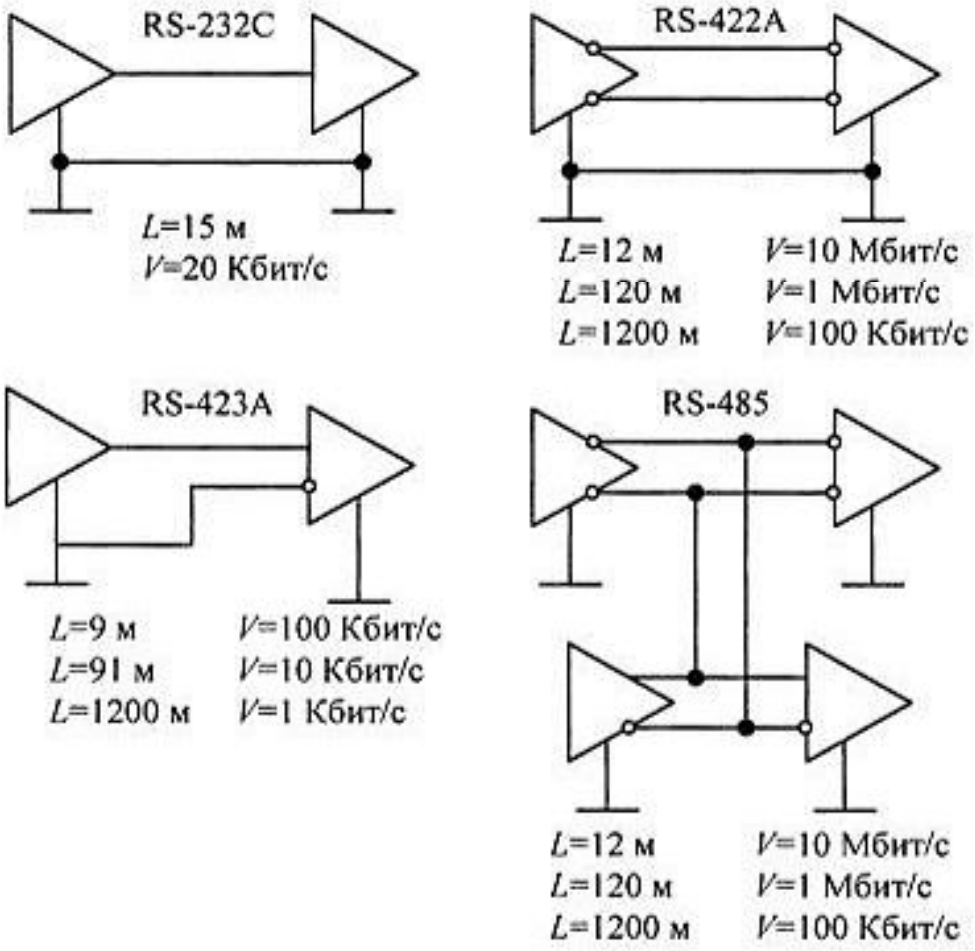


Рис. 19.20. Схемы соединения приемников и передатчиков в различных стандартах асинхронных последовательных интерфейсов

дены схемы соединения приемников и передатчиков, а также показаны ограничения на длину линии (L) и максимальную скорость передачи данных (V) по этим интерфейсам.

Несимметричные линии интерфейсов RS-232C и RS-423A имеют самую низкую защищенность от синфазной помехи, хотя дифференциальный вход приемника RS-423A несколько смягчает ситуацию. Лучшие параметры имеет двухточечный интерфейс RS-422A и его магистральный (шинный) аналог RS-485, работающие на симметричных линиях связи. В них для передачи каждого сигнала используются дифференциальные сигналы с отдельной (витой) парой проводов.

Последовательный асинхронный порт, работающий по стандартам RS-232, RS-423A и RS-422A, позволяет соединять между собой только два устройства. Это связано с тем, что при параллельном соединении двух передатчиков их выходные каскады могут выйти из строя. В ряде случаев требуется объединить несколько устройств. Для того чтобы выходные каскады передатчиков последовательных портов не выходили из строя, необходимо применять специальные меры, которые обсуждались в предыдущих главах. Эти меры реализованы в интерфейсе RS-485.

На этом можно завершить обзор применяемых в микропроцессорной технике асинхронных последовательных портов. Подробное рассмотрение вопросов применения этих портов выходит за рамки данной книги. Теперь можно перейти к обсуждению вопросов построения систем реального времени.

Принципы построения таймеров

Таймеры предназначены для формирования временных интервалов, позволяя микропроцессорной системе работать в режиме реального времени. Именно благодаря таймерам удается согласовать время реакции микропроцессорной системы с окружающей аппаратной средой. Кроме того, таймеры в ряде случаев позволяют формировать импульсы заданной длительности, периодические последовательности и другие радиотехнические сигналы.

Таймеры представляют собой обычные цифровые счетчики, которые подсчитывают импульсы, поступающие на их вход от высокостабильного генератора частоты (который и является эталоном времени). К системной шине микропроцессора таймеры, как и все рассмотренные ранее устройства, входящие в состав микропроцессорной системы, подключаются при помощи внутренних параллельных портов.

Генератор периодических импульсов, входящий в состав схемы таймера, определяет минимальный интервал времени, который может формировать таймер. Интервалы времени, задаваемые таймером, могут устанавливаться толь-

ко из дискретного набора допустимых интервалов времени. Их конкретные значения тоже определяются частотой задающего генератора. Разрядность цифрового счетчика, входящего в состав таймера, определяет максимальный интервал времени, который может формировать таймер.

Суммирующие и вычитающие таймеры

Обычно в составе микропроцессорной системы используются 16-разрядные таймеры, поэтому для их подключения к 8-разрядному процессору требуется два параллельных порта. Обычно требуется не только устанавливать значение таймера (производить запись числового значения в цифровой счетчик), но и считывать текущее значение этого счетчика. Поэтому для подключения счетчика таймера обычно используются порты ввода-вывода.

Кроме задания временных интервалов и считывания текущего значения таймера, таймером нужно управлять. В процессе работы микропроцессорной системы обычно требуется включать и выключать таймер. Часто необходимо определять, не возникало ли переполнение таймера. Факт переполнения легко запомнить в дополнительном триггере, подключенном к выходу переноса счетчика таймера. Выходной сигнал этого триггера называется флагом переполнения таймера.

Триггер включения и выключения таймера и триггер переполнения подобно любым внешним устройствам подключают к системной шине микропроцессора через дополнительный порт ввода-вывода. Содержимое этих триггеров обычно называют флагами и объединяют в восьмиразрядный регистр управления таймером. Этот регистр одновременно является регистром данных внутреннего порта ввода-вывода. Оставшиеся шесть разрядов этого регистра остаются неиспользуемыми. Запись в эти разряды нулевых или единичных значений ни к каким действиям не приводит. При чтении этих разрядов могут быть получены случайные комбинации нулей и единиц.

Формат подобного регистра управления таймером показан на рис. 19.21, а структурная схема таймера, реализующая описанные выше принципы, приведена на рис. 19.22.

Рис. 19.21. Формат регистра управления таймером

В данном регистре бит TF отображает содержимое флага переполнения таймера, а бит TR определяет, включен ли таймер.

В зависимости от типа использованного цифрового счетчика таймеры бывают *суммирующие* (с суммирующим счетчиком) или *вычитающие* (с вычи-

тающим счетчиком). Использование вычитающего счетчика позволяет проще задавать интервалы времени. В этом случае записываемый в таймер код $Code_{sub}$ будет соответствовать длительности временного интервала T_{timer}, вырабатываемого таймером:

$$T_{timer} = Code_{sub} \times T_{gen},$$

где T_{gen} — период импульсов внутреннего генератора.

Это объясняется тем, что счет начинается от заданного числа и заканчивается при достижении счетчиком таймера нулевого значения. Например, при записи в таймер числа 100 потребуется ровно 100 импульсов на входе вычитающего счетчика для того, чтобы его содержимое приняло нулевое значение. Если генератор вырабатывает колебание с частотой 1 МГц, то его период равен 1 мкс, а это означает, что переполнение вычитающего таймера произойдет ровно через 100 мкс.

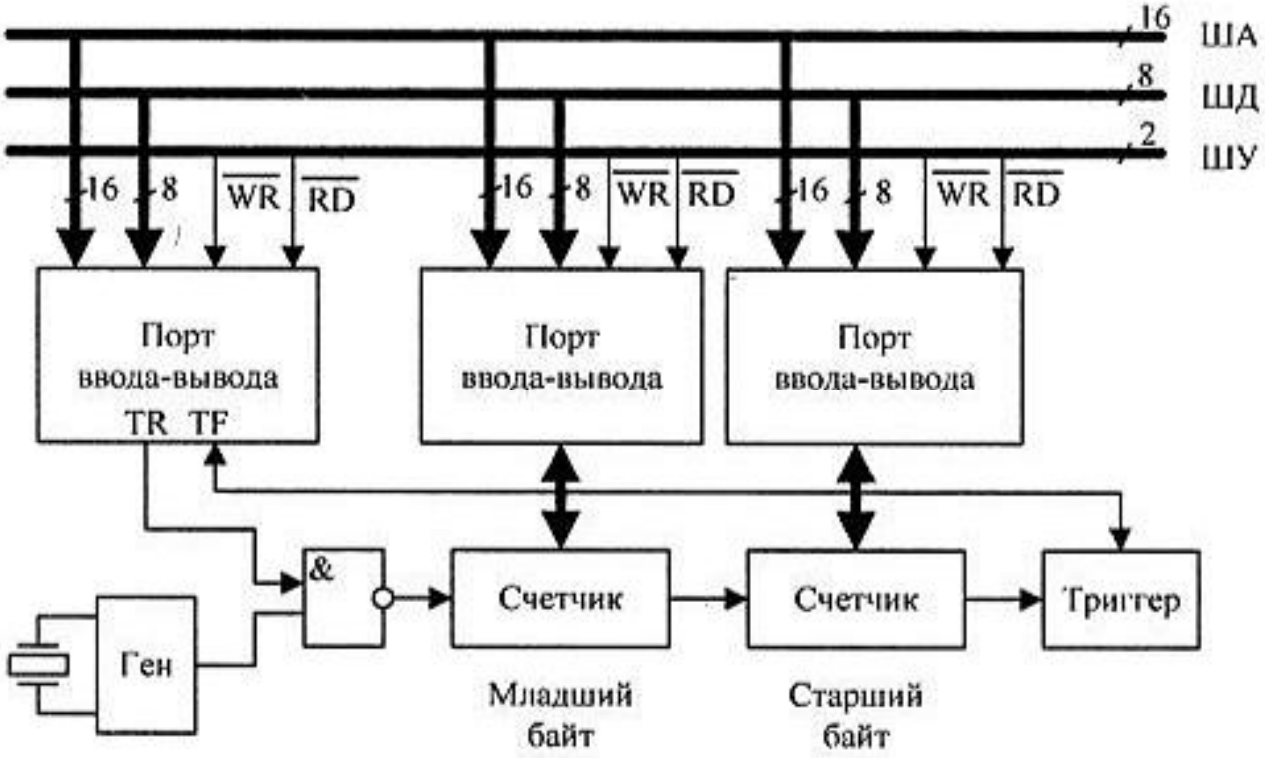


Рис. 19.22. Структурная схема таймера

В случае использования суммирующего таймера код, записываемый в таймер для задания интервала времени T_{timer}, определяется из другой формулы:

$$T_{timer} = (Code_{max} - Code_{sum}) \times T_{gen}$$

В этой формуле код $Code_{sum}$, который заносится в таймер, представляет собой дополнение кода интервала времени до максимального кода $Code_{max}$,

который можно записать в таймер. Максимальный код $Code_{max}$ определяется разрядностью таймера. В рассмотренном примере разрядность таймера равна 16. Это означает, что максимальный код равен $Code_{max} = 2^{16} = 65536$.

В данном случае для задания интервала времени 100 мкс в таймер потребуется записать число 65435. После поступления на вход счетчика таймера 100 импульсов частоты задающего генератора это число станет равным 65535 (все разряды счетчика содержат единичное значение), а это в свою очередь означает, что произошло переполнение таймера и в триггер флага переполнения запишется единичный потенциал. Результат работы будет тем же самым, что и при использовании вычитающего таймера — флаг переполнения установился ровно через 100 мкс.

Как видите, использование суммирующего таймера для задания интервалов времени несколько сложнее применения вычитающего таймера, однако т. к. все расчеты производятся на этапе написания программы, то это не вызывает значительных трудностей. Более того — ✧ *обратите внимание, что* $Code_{max}$ *равно модулю счета суммирующего счетчика, поэтому, как это уже обсуждалось в предыдущих главах, его в вычислениях можно не учитывать и заносить в счетчик таймера просто отрицательное значение временного интервала в двоичном дополнительном коде. Вычисление отрицательного числа в двоичном дополнительном коде умеет выполнять любой транслятор с любого языка программирования.*

✧ *Обратите внимание*, что как в суммирующем, так и в вычитающем таймере после задания нового временного интервала таймера необходимо программно обнулять его флаг переполнения TF. Иначе будет невозможно определить, произошло ли очередное переполнение таймера, или флаг установлен при формировании таймером предыдущего временного интервала.

Таймеры с автозагрузкой

Рассмотренные схемы таймеров позволяют сформировать одиночный интервал времени, однако при создании систем реального времени, или при генерации периодического сигнала, требуется формировать интервалы времени с одинаковым периодом. Эту задачу можно решить постоянной перезагрузкой таймера, но это требует постоянной работы от центрального процессора. При формировании сигнала с коротким периодом на процессор может ложиться значительная вычислительная нагрузка. Намного удобнее было бы решить эту задачу аппаратным образом. Подобная задача уже решалась нами при рассмотрении счетчиков с произвольным коэффициентом счета.

Если в составе таймера применить счетчики с возможностью параллельной загрузки, то подобный таймер будет переполняться с заранее заданным

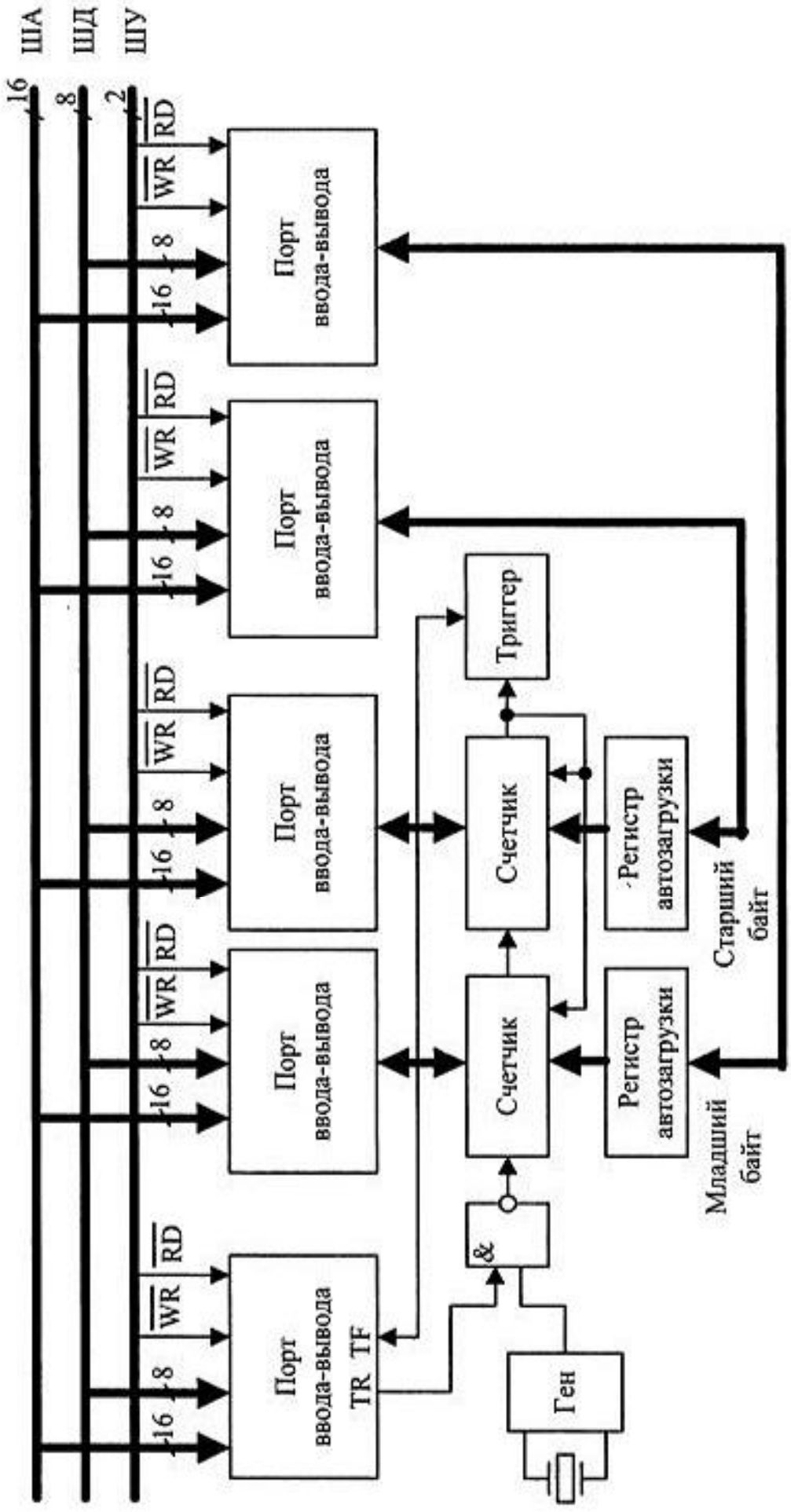


Рис. 19.23. Структурная схема таймера с автозагрузкой

периодом. Число, которое следует загружать в таймер, хранится в специальном 16-разрядном регистре автозагрузки. Доступ к этому регистру, точно так же как и к счетчику таймера, можно получить при помощи двух дополнительных параллельных портов ввода-вывода. Общее количество ячеек, которое будет занимать таймер в адресном пространстве микропроцессорной системы, при этом достигает пяти. Структурная схема таймера с автозагрузкой приведена на рис. 19.23.

Число, которое следует заносить в регистр автозагрузки, можно посчитать по тем же самым формулам, что приведены для суммирующего и вычитающего таймеров. Контроль переполнения таймера может быть осуществлен как программным опросом флага переполнения таймера, так и при помощи механизма аппаратного прерывания выполняемой программы.

Реверсивные таймеры

Иногда таймеры выполняются реверсивными. В этом случае в таймере применяется реверсивный счетчик и для задания направления счета в регистр управления таймером вводится дополнительный бит, значение которого будет определять, в каком из режимов будет применен таймер — в суммирующем или в вычитающем. Формат подобного регистра управления таймером показан на рис. 19.24.

-	-	-	-	TS	TR	TF

Рис. 19.24. Формат регистра управления реверсивным таймером

В данном регистре бит TF отображает содержимое флага переполнения таймера, бит TS задает направление счета внутреннего счетчика таймера, а бит TR определяет, включен ли таймер.

Структурная схема таймера, реализующая описанные выше принципы, приведена на рис. 19.25. В этой схеме для упрощения понимания принципа работы реверсивного таймера не применен механизм автоматической загрузки.

В схеме, приведенной на рис. 19.25, направление счета двоичного реверсивного счетчика определяется потенциалом на входе выбора режима счета '+/–1' этого счетчика.

Свободнобегущие таймеры

Достаточно часто в микропроцессорной системе требуется одновременно формировать несколько временных интервалов. В этом случае приходится применять несколько таймеров. (Структурная схема такого таймера показана на рис. 19.23.)

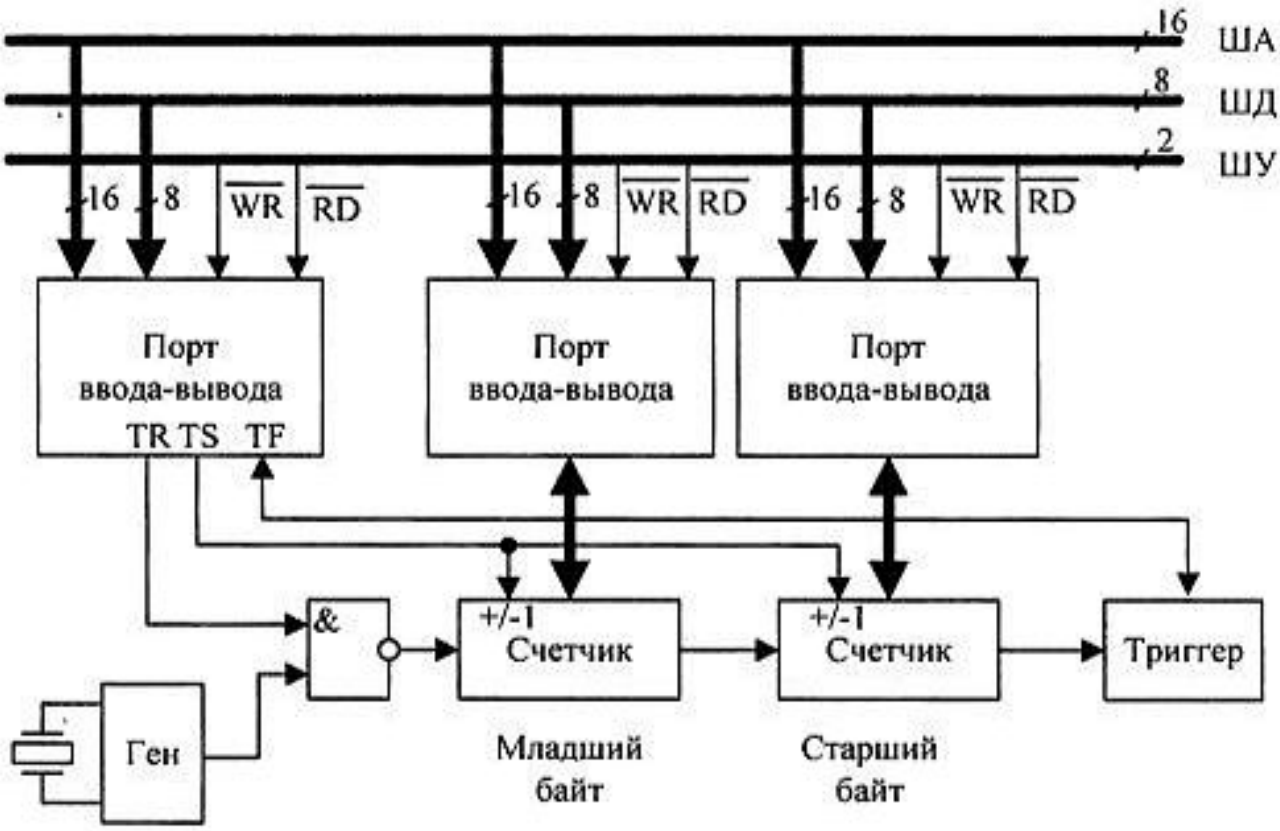


Рис. 19.25. Структурная схема таймера

В ряде случаев подобный механизм становится неудобным. Тогда для одновременного задания нескольких временных интервалов применяются свободнобегущие суммирующие таймеры. Свободнобегущие таймеры используются как системные часы, задающие время внутри микропроцессорной системы.

Для задания промежутков времени микропроцессор считывает значение текущего системного времени и суммирует с ним код задаваемого промежутка времени. Полученный результат записывается в регистр сравнения таймера. При совпадении значений таймера и регистра сравнения устанавливается флаг совпадения. Значение этого флага можно определить программным опросом или воспользоваться механизмом прерывания работы процессора, т. е. работа со свободнобегущим таймером похожа на работу с обычным будильником, к которому мы привыкли в обычной жизни.

Структурная схема свободнобегущего таймера с одиночным модулем сравнения приведена на рис. 19.26.

В данной схеме вычисленное время окончания временного интервала заносится в регистры модуля сравнения RGL и RGH. Как только содержимое этих регистров и содержимое таймера совпадет, кодовый компаратор выдаст единичный потенциал. Этот сигнал записывается в флаг совпадения кодов TC, который в данном случае работает точно так же, как и флаг переполнения обычного таймера TF. В качестве кодового компаратора можно применить

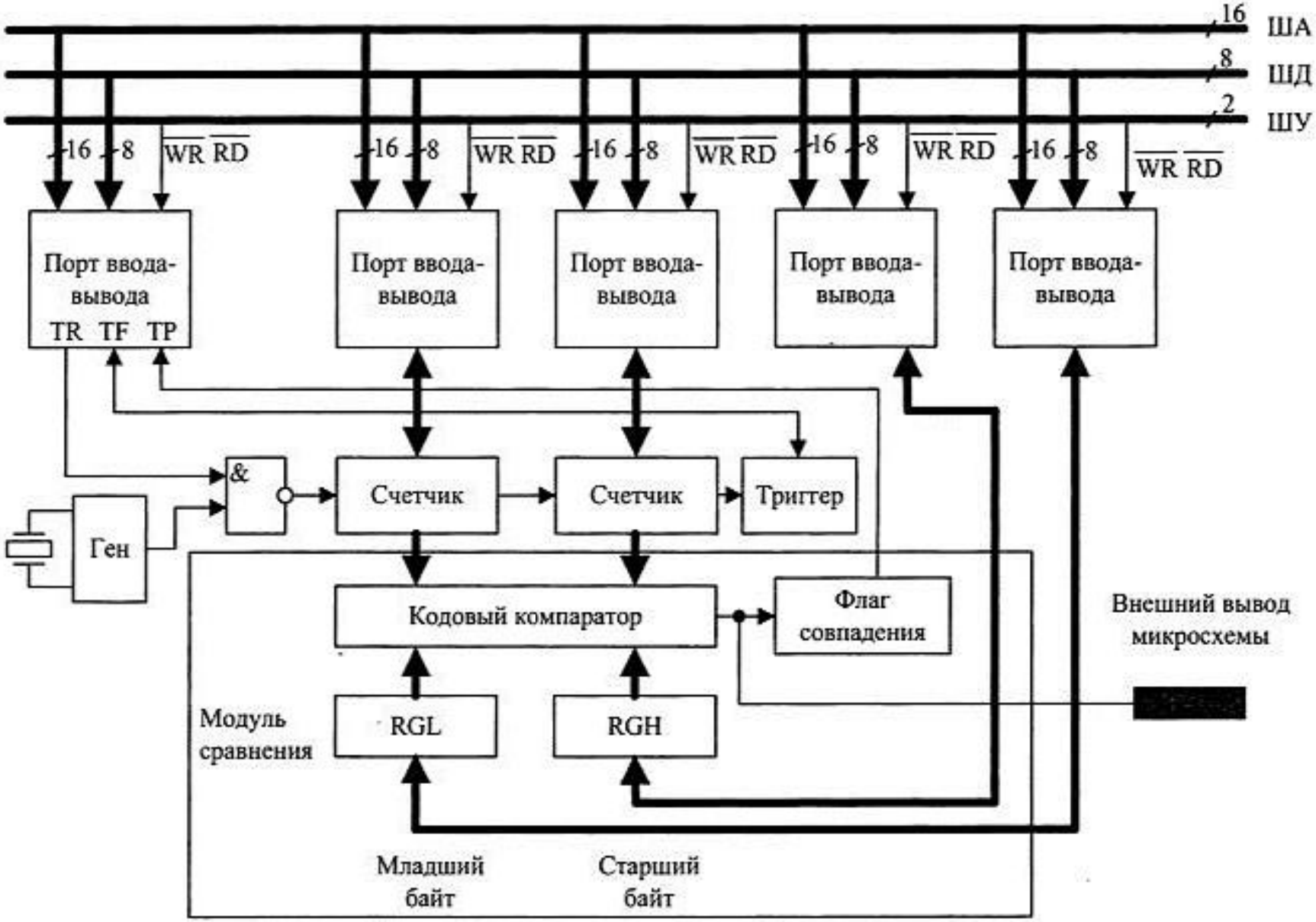


Рис. 19.26. Структурная схема свободнобегущего таймера с модулем сравнения

схему вычитателя, которая уже рассматривалась нами в предыдущей главе. В качестве выхода кодового компаратора используется выход переноса, на котором будет сформирован знак результата вычитания.

Часто с одним свободнобегущим таймером применяется несколько модулей сравнения. Это похоже на использование часов с несколькими будильниками. В результате один свободнобегущий таймер может обеспечить формирование сразу нескольких временны́х интервалов.

В ряде случаев модули сравнения используются для формирования сигналов с широтно-импульсной модуляцией. В этом режиме кодовый компаратор сравнивает два кода, поступающие на его входы, и вырабатывает единичный сигнал при превышении содержимого счетчика над содержимым регистров модуля сравнения. Если в регистры записать нулевое значение, то на этом выходе постоянно будет присутствовать единичный потенциал. Если в них записать число, равное максимальному значению счетчика, то на выходе кодового компаратора постоянно будет присутствовать нулевой потенциал. Записав в регистры модуля захвата половинное значение от максимального значения счетчика таймера, мы получим скважность выходного колебания, равную 2.

Таким образом, записывая в регистр модуля сравнения различные числа, можно регулировать отношение высокого и низкого потенциала на выходе кодового компаратора с высокой точностью. Период колебания, вырабатываемого данной схемой, будет соответствовать периоду переполнения счетчика таймера. Описанный принцип формирования широтно-импульсного сигнала иллюстрируется на рис. 19.27.

При довольно большой частоте тактового сигнала частота переполнения свободнобегущего таймера окажется достаточно высокой, и выход кодового компаратора можно использовать вместо цифроаналогового преобразователя.

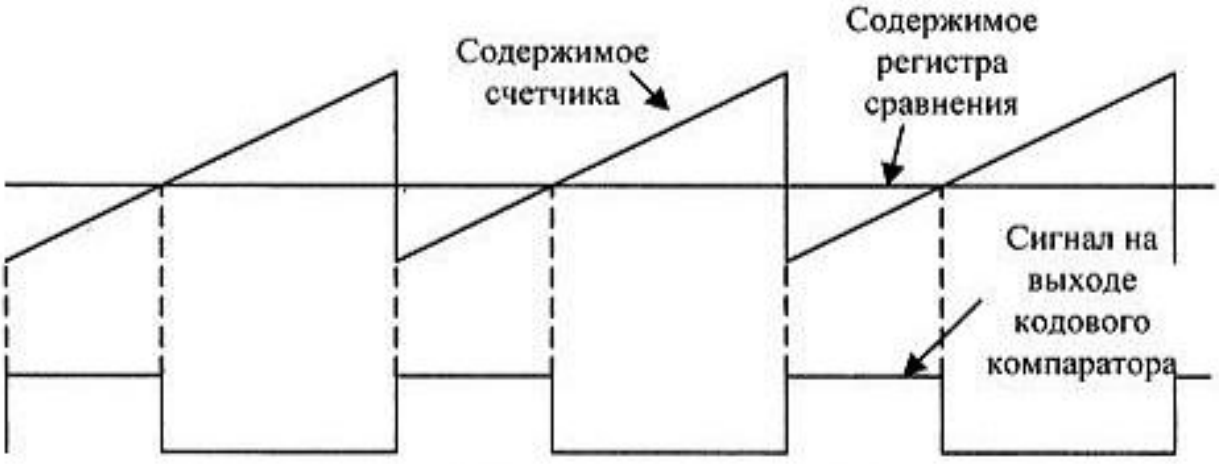


Рис. 19.27. Формирование широтно-импульсного сигнала на выходе кодового компаратора

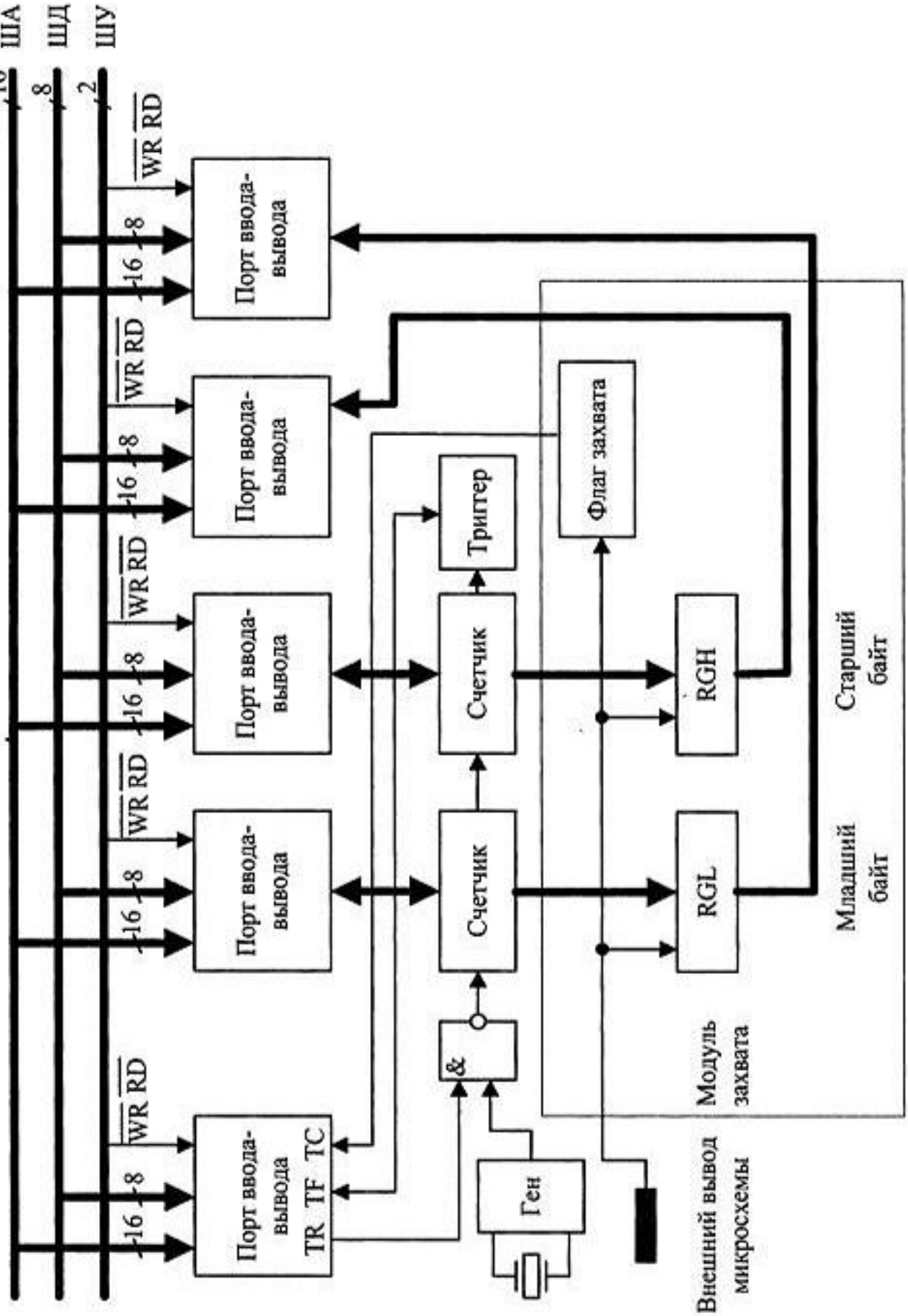


Рис. 19.28. Структурная схема свободнобегущего таймера с модулем захвата

Достаточно на его выходе поставить простейшую сглаживающую RC-цепочку и изменение скважности выходного сигнала будет приводить к изменению напряжения на емкости этой схемы.

Кроме модулей сравнения, со свободнобегущим таймером работают модули захвата, которые позволяют аппаратно запоминать состояния внутренних счетчиков в момент какого-либо внешнего события (как, например, фронта входного сигнала измерение при измерении его периода входного сигнала) без участия центрального процессора. При этом тактовая частота таймера может быть в несколько раз выше скорости выполнения команд микропроцессора. В результате точность определения времени события тоже будет в несколько раз выше по сравнению с использованием обычного таймера. Структурная схема свободнобегущего таймера с модулем захвата приведена на рис. 19.28.

Флаг захвата в этой схеме позволяет определить — произошло внешнее событие или нет. Этот флаг добавляется в регистр управления таймера. При необходимости этот флаг может вызывать аппаратное прерывание выполнения программы микропроцессорной системы.

Использование модулей захвата позволяет повысить точность измерения времени каких-либо событий, т. к. на нее перестает влиять такой нестабильный параметр, как время реакции программы, и она будет определяться только быстродействием цифровых микросхем свободнобегущего таймера.

Так как схема модуля захвата незначительно отличается от модуля сравнения, то обычно этот модуль делается настраиваемым, и в зависимости от отдельно выделяемого бита настройки работает либо в режиме сравнения, либо в режиме захвата. Применение модуля в режиме сравнения или в режиме захвата определяется конечным пользователем (программистом микропроцессорной системы).

На этом завершим обзор различных видов таймеров, применяемых в микропроцессорных системах. Теперь кратко остановимся на методах расширения возможностей микропроцессорной системы.

Способы расширения адресного пространства микропроцессора

В предыдущих главах мы определили, что размер адресного пространства микропроцессорной системы определяется разрядностью счетчика команд микропроцессора. Достаточно часто при развитии микропроцессорной системы возможности адресного пространства исчерпываются (суммарный объем доступных микросхем ОЗУ и ПЗУ превышает объем адресного простран-

ства). В таком случае приходится прибегать к методам расширения адресного пространства.

Как это уже обсуждалось ранее, размер адресного пространства микропроцессора определяется количеством адресных проводников в составе адресной шины. Для увеличения его адресного пространства необходимо увеличить количество этих проводников (добавить старшие разряды к шине адреса).

Метод страничного расширения адресного пространства

Для того чтобы понять, как можно увеличить адресное пространство в уже существующей микропроцессорной системе, не изменяя счетчик команд, а значит, не меняя микропроцессор, давайте обратим внимание на то, что сигналы на адресных проводниках ничем не отличаются от сигналов, используемых для управления внешними устройствами, поэтому для расширения адресного пространства микропроцессорной системы можно воспользоваться схемой, подобной параллельному порту. Выходы регистра в этом случае мы будем использовать в качестве старших разрядов адресной шины. При добавлении к адресной шине одного дополнительного разряда мы удваиваем адресное пространство микропроцессорной системы. Записав в этот бит нулевое значение, мы получим прежний вариант микропроцессорной системы, а записав единичное значение, мы будем обращаться к дополнительному адресному пространству. Добавив к адресной шине восемь дополнительных бит, мы уже получим 256 адресных пространств, подобных первоначальному адресному пространству. Записывая в дополнительные разряды адреса число, соответствующее номеру адресного пространства, мы будем переключаться между этими адресными пространствами.

Исходное адресное пространство эквивалентно странице книги, а переключение между адресными пространствами эквивалентно перелистыванию страниц в этой книге. Поэтому описанный метод расширения адресного пространства получил название *страничного метода* адресации. Параллельный порт, при его применении для расширения адресного пространства микропроцессорной системы, называется диспетчером памяти, а его регистр данных — переключателем страниц. Схема центрального процессора с диспетчером памяти, реализованным по схеме параллельного порта, приведена на рис. 19.29.

На данной схеме изображен микропроцессор с наиболее распространенным видом микропроцессорной шины. В нем для экономии внешних выводов шина данных и шина адреса объединены и для их разделения используется специальный сигнал — ALE. Разделение адресов и данных производит параллельный регистр D3, который запоминает младший байт адреса по сигналу

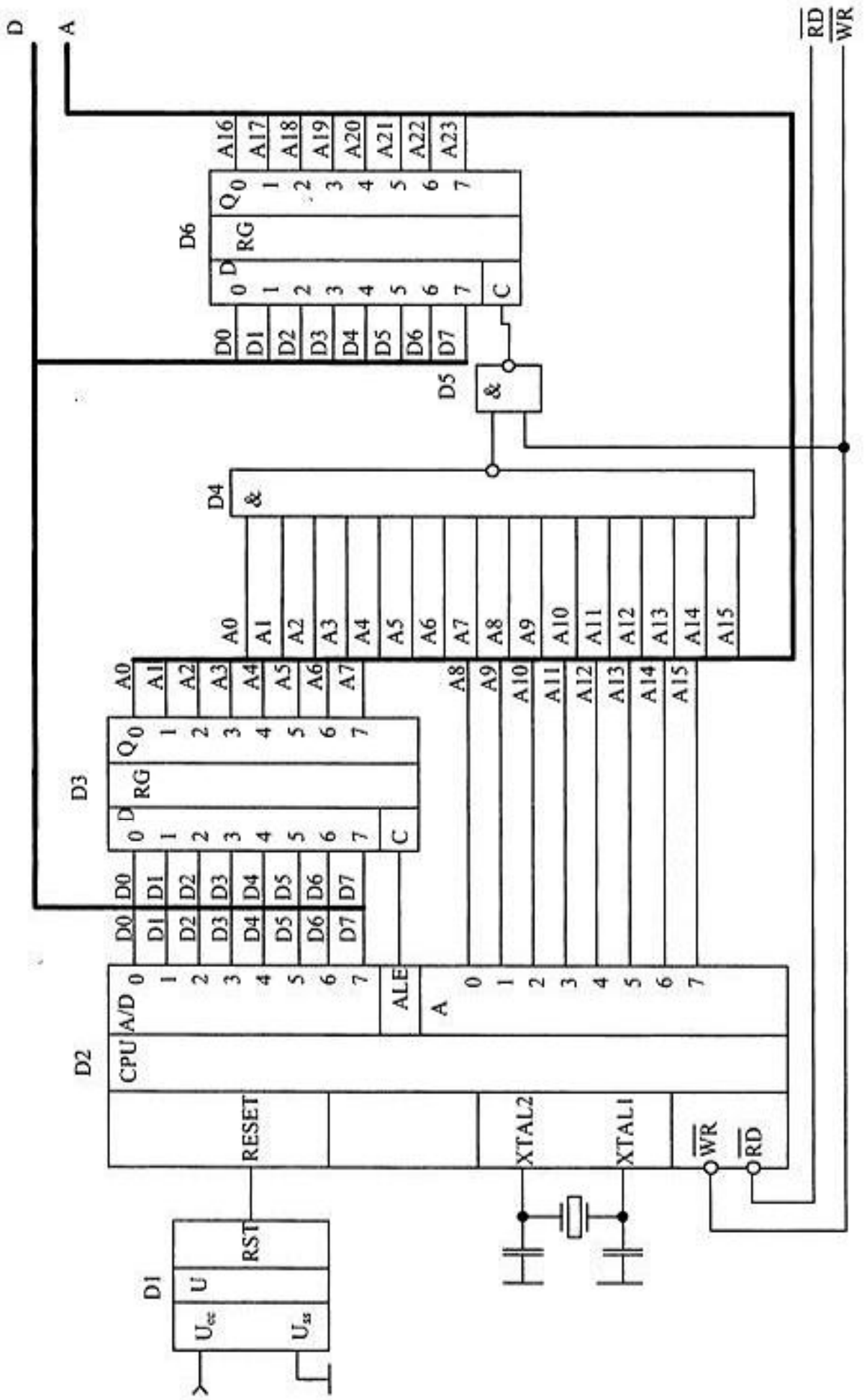


Рис. 19.29. Схема центрального процессора с переключателем страниц памяти, реализованным по схеме параллельного порта

ALE. Подобный принцип применяется в микропроцессорах К580, К1821, К1816, К1830 и ряде зарубежных микропроцессоров, например, семействе микроконтроллеров AVR. В результате применения параллельного регистра D3 формируется уже рассмотренная нами ранее 16-разрядная шина адреса.

В качестве регистра данных переключателя страниц в схеме на рис. 19.29 использован параллельный регистр D6. Для обращения к этому регистру используется дешифратор адреса, собранный на микросхеме D4. Дешифратор адреса настроен на число 0FFFFh, что позволяет обращаться к переключателю страниц на каждой странице как к самой старшей ячейке памяти страницы, а это значит, что мы потеряем только одну ячейку адресного пространства каждой страницы.

Как уже обсуждалось ранее, при применении восьмиразрядного переключателя страниц в микропроцессорной системе появились дополнительные восемь

Рис. 19.30. Структура страничного адресного пространства

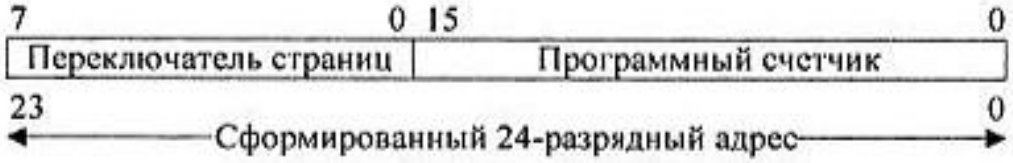


Рис. 19.31. Формирование адреса с использованием переключателя страниц

линий адреса. В результате адресное пространство микропроцессорной системы увеличилось до 2^{24} = 16 Мбайт. Структура нового адресного пространства приведена на рис. 19.30, а принцип формирования нового адреса с использованием переключателя страниц пояснен на рис. 19.31.

Метод сегментного расширения адресного пространства

Метод страничной адресации прост в реализации, и при формировании адреса физической памяти не приводит к дополнительным временным задержкам. Однако при использовании многозадачного режима работы процессора для каждой активной задачи выделяется целая страница в системной памяти микропроцессора. Если программный код задачи не занимает полностью страницу, то в системной памяти процессора остается много неиспользуемых областей. Решить возникшую проблему позволяет метод *сегментной организации памяти*.

При использовании этого метода для расширения адресного пространства используется базовый регистр, относительно которого производится адресация команд или данных в программе. Разрядность базового регистра обычно выбирают равной разрядности счетчика команд. Для обращения к базовому регистру диспетчера памяти, как и при страничной организации памяти, можно использовать схему, подобную схеме параллельного порта. Правда для отображения 16-разрядного регистра в адресное пространство 8-разрядного микропроцессора потребуется уже две ячейки памяти.

Пример распределения адресного пространства при использовании сегментного метода адресации и различном размере программных сегментов приведен на рис. 19.32.

Как видно из этого рисунка, размеры отдельных сегментов памяти в отличие от размеров страниц могут быть различными. Это связано с тем, что окна (а размер окна остается прежним — 64 Кбайт) при сегментной организации памяти могут перекрываться, и если часть памяти в окне, выделенном для предыдущей программы, не используется, то следующее окно можно разместить, начиная с первой свободной ячейки памяти. Такой метод расширения адресного пространства позволяет более экономно использовать доступные ресурсы памяти микропроцессора.

Для формирования физического адреса, при сегментном виде расширения адресного пространства микропроцессорной системы, используется параллельный двоичный сумматор. На входы этого сумматора подается содержимое базового регистра и счетчика команд. Суммирование производится со смещением кода базового регистра влево на несколько битов относительно

содержимого счетчика команд. В результате максимальный размер сегмента определяется разрядностью программного счетчика, а максимальная неиспользуемая область памяти — смещением базового регистра относительно программного счетчика. Учитывая, что базовый регистр смещен относительно программного счетчика на четыре разряда, минимальный шаг при размещении окон будет $2^4 = 16$ байт, т. е. в этом случае максимальная область неиспользуемой памяти между программными сегментами будет равна 15 байтам.

Рис. 19.32. Пример адресного пространства с использованием трех программных сегментов

Рассмотренные принципы формирования адреса при сегментной адресации иллюстрируются рис. 19.33. Вместо программного счетчика могут выступать регистры указателя данных и указателя стека. Особенности применения этих регистров будут рассмотрены в последующих главах.

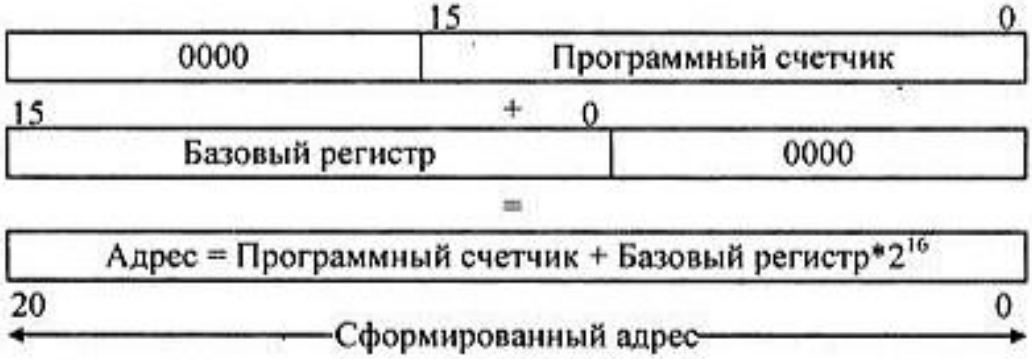


Рис. 19.33. Формирование адреса при сегментной адресации

Обычно в диспетчере памяти используется не один, а несколько базовых регистров. Это позволяет микропроцессору работать с несколькими сегментами одновременно. Количество базовых регистров обычно определяет максимальный объем данных, доступных для программы.

Количество одновременно используемых сегментов определяется количеством базовых регистров. Сегменты могут перекрываться в адресном пространстве, и тем самым можно регулировать размер памяти, который отводится под каждый конкретный сегмент памяти. В компьютерах семейства IBM PC имеются четыре базовых регистра, определяющих сегмент данных — DS, сегмент программы — PS, сегмент стека — SS и дополнительный сегмент — ES. Информацию в базовые регистры заносит операционная система при запуске программы и переключении между активными программами.

Метод расширения адресного пространства при помощи окон

Еще одним распространенным способом увеличения адресного пространства микропроцессорной системы является *применение окон*. При использовании окон производится расширение не всего адресного пространства, а только его части. Внутри адресного пространства выделяется некоторая область, которая называется окном. В это окно может отображаться часть другого адресного пространства.

При использовании окон может быть применен как страничный, так и сегментный метод отображения соседнего адресного пространства в окно. При этом размер страницы, отображаемой в окно, не может превышать размер самого окна.

При использовании страничного метода отображения, конкретная страница дополнительного адресного пространства, отображаемая в окно основной памяти, определяется переключателем страниц. Переключатель страниц строится по принципу, рассмотренному ранее (см. схему на рис. 19.29).

При использовании сегментного метода конкретная область адресного пространства, которая будет отображаться в окно, определяется содержимым базового регистра. Если разрядность адреса вспомогательного адресного пространства, отображаемого в окно основной памяти, совпадает с разрядностью базового регистра, то любая область вспомогательной памяти может быть отображена в основную память с точностью до байта.

Принцип использования оконной адресации при отображении страниц дополнительной памяти в основное адресное пространство можно легко понять по рис. 19.34.

Рис. 19.34. Применение окна для расширения адресного пространства

Оконная адресация часто используется при развитии микропроцессорных семейств, когда размера областей памяти, отведенных для конкретных задач в младших моделях семейства, не хватает для старших моделей, а при этом нужно поддерживать аппаратную совместимость с младшими моделями семейства. В качестве примера можно привести микросхемы I81c96 фирмы INTEL или TMS320c5410 фирмы Texas Instrument, где для расширения области регистров специальных функций используется оконная адресация.

Динамические оперативные запоминающие устройства (ОЗУ)

Статические ОЗУ позволяют обеспечивать хранение записанной информации до тех пор, пока на микросхему подается питание. Однако запоминающая ячейка статического ОЗУ занимает относительно большую площадь, поэтому для ОЗУ большого объема применяют более простую и потому компактную запоминающую ячейку — конденсатор. Естественно, что заряд на конденсаторе с течением времени уменьшается, поэтому его необходимо подзаряжать с периодом приблизительно 10 мс, называемым периодом регенерации. Подзарядка емкости производится при считывании ячейки памяти, поэтому для регенерации информации достаточно просто считать регенерируемую ячейку памяти.

Схема запоминающего элемента динамического ОЗУ, выполненного на одиночном коммутирующем транзисторе и запоминающей емкости, а также его конструкция приведены на рис. 19.35.

При считывании заряда емкости необходимо учитывать, что линия считывания имеет намного большую электрическую емкость, по сравнению с запо-

минающей ячейкой, поэтому при подключении к запоминающей ячейке линии считывания напряжение на ней изменяется. Графики изменения напряжения на линии считывания при выполнении операции чтения информации из запоминающей ячейки без использования схемы регенерации приведены на рис. 19.36.

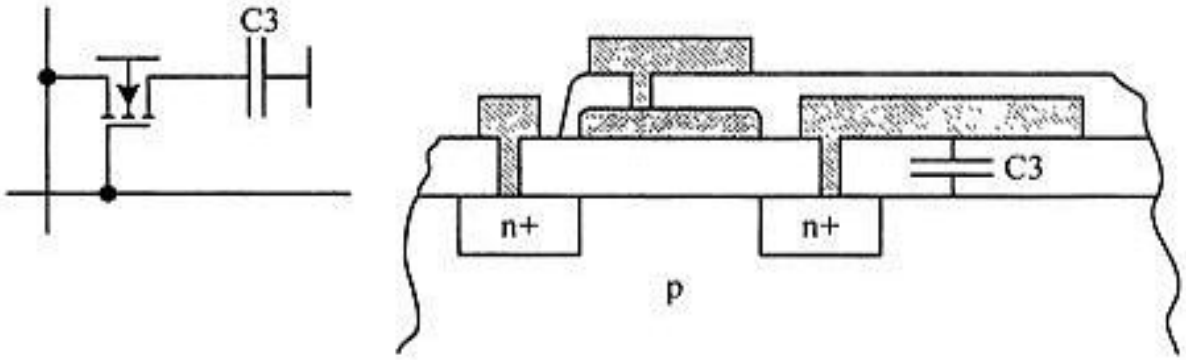


Рис. 19.35. Схема запоминающего элемента динамического ОЗУ и его конструкция

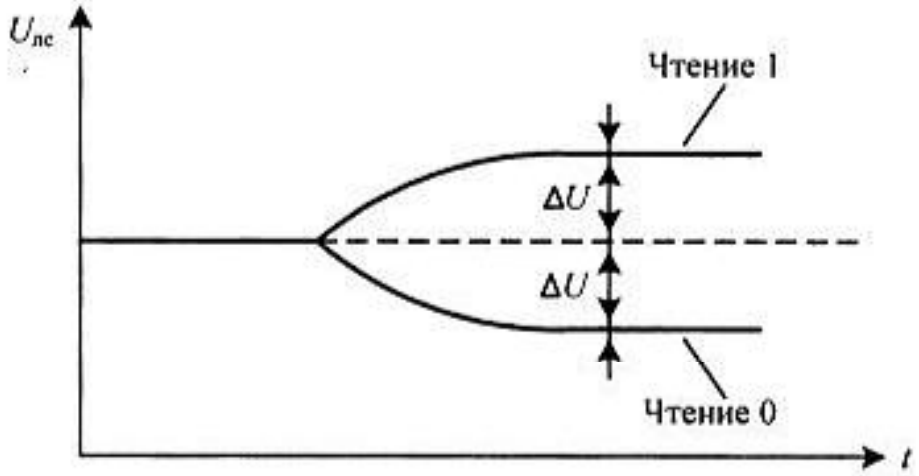


Рис. 19.36. Графики изменения напряжения на линии считывания при считывании информации с запоминающей ячейки

Первоначально на линии записи/считывания присутствует половина напряжения питания микросхемы. При подключении к линии записи/считывания запоминающей ячейки заряд, хранящийся в запоминающей ячейке, изменяет напряжение на линии на небольшую величину ΔU. Теперь это напряжение необходимо восстановить до первоначального логического уровня. Если приращение напряжения ΔU было положительным, то напряжение необходимо довести до напряжения питания микросхемы. Если приращение ΔU было отрицательным, то напряжение необходимо довести до потенциала общего провода.

Для регенерации первоначального заряда, хранившегося в запоминающей ячейке, в схеме применяется RS-триггер, включенный между двумя линиями

записи/считывания. Схема такого регенерирующего устройства приведена на рис. 19.37. Эта схема за счет положительной обратной связи восстанавливает первоначальное значение напряжения, хранившегося в запоминающей ячейке. При этом на соседней линии считывания формируется противоположный сигнал, но т. к. она в данный момент никуда не подключена, то это неважно. Подробно работа триггера рассматривалась в предыдущих главах книги. Результатом работы данного регенератора является то, что при считывании содержимого запоминающей ячейки динамического ОЗУ производится регенерация хранящегося в ней заряда. Для уменьшения времени регенерации микросхема устроена так, что при считывании одной ячейки памяти в строке запоминающей матрицы регенерируется вся строка.

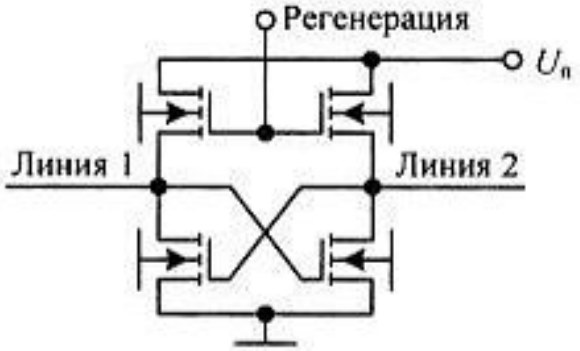


Рис. 19.37. Схема регенерирующего каскада

Особенностью использования динамических ОЗУ является мультиплексирование шины адреса. Адрес строки и адрес столбца передаются поочередно. Адрес строки синхронизируется стробирующим сигналом RAS# (Row Address Strobe), а адрес столбца — сигналом CAS# (Column Address Strobe). Мультиплексирование адресов позволяет уменьшить количество выводов микросхем ОЗУ, что очень важно для микросхем с большим объемом внутренней памяти, т. е. с большой разрядностью адресной шины. Условно-графическое обозначение микросхемы динамического ОЗУ на схемах приведено на рис. 19.38, а временные диаграммы сигналов обращения к такой микросхеме — на рис. 19.39.

Анализируя временные диаграммы сигналов, приведенные на рис. 19.39, можно прийти к выводу, что между интерфейсами микросхем динамического ОЗУ и системной шины процессора имеются существенные различия. Именно поэтому микросхемы динамического ОЗУ подключаются к системной шине через специализированную микросхему контроллера динамического ОЗУ, которая преобразует сигналы системной шины в сигналы обращения к микросхемам динамического ОЗУ. Кроме того, эта микросхема осуществляет регенерацию содержимого ОЗУ с периодом 10 мс. В современных микропроцессорных системах, таких как, например, сигнальный процессор, контроллер динамического ОЗУ встраивается в саму микросхему.

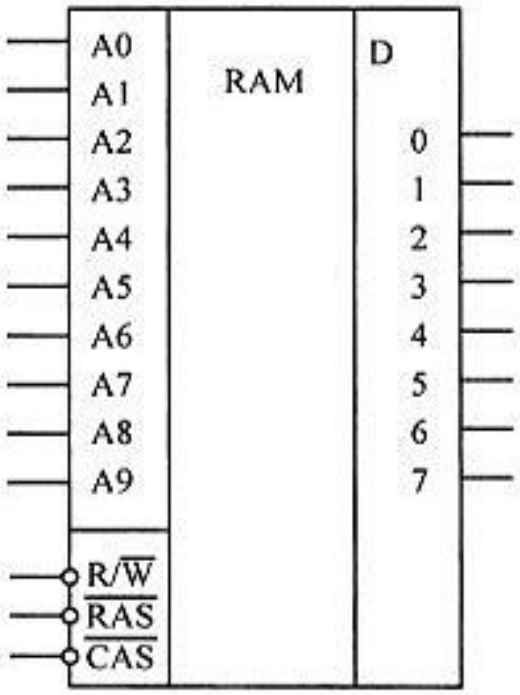


Рис. 19.38. Условно-графическое обозначение динамического ОЗУ

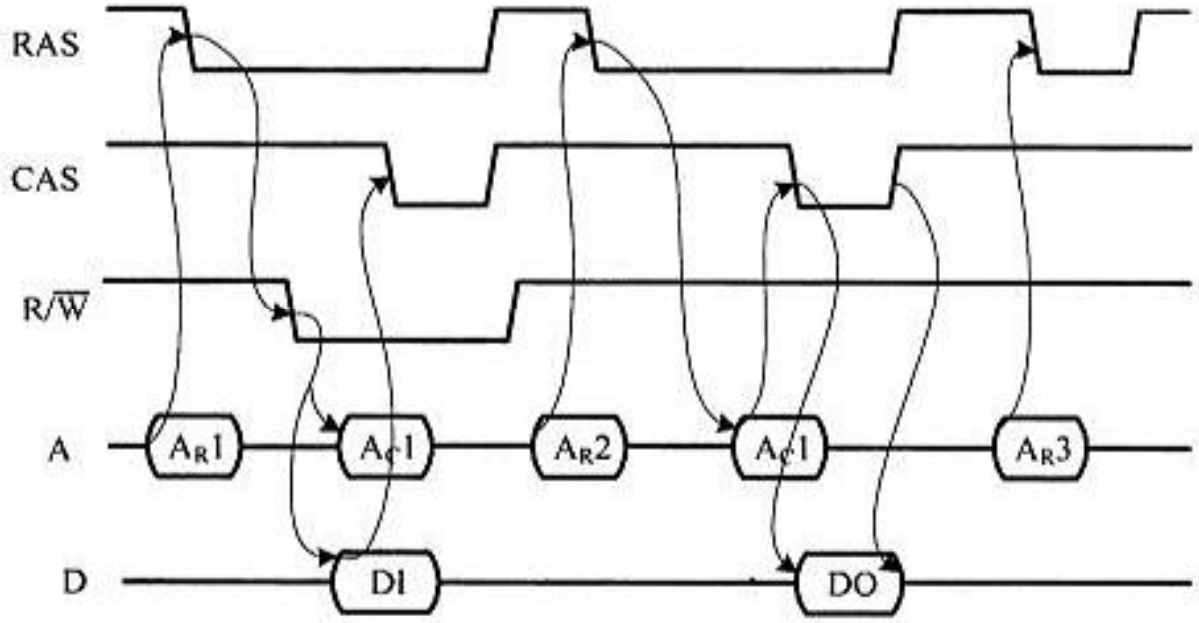


Рис. 19.39. Временная диаграмма обращения к динамическому ОЗУ

Долгое время работа с динамическими ОЗУ велась в строгом соответствии с временными диаграммами сигналов, приведенными на рис. 19.39. Затем было замечено, что обычно обращение ведется к данным, лежащим в соседних ячейках памяти, поэтому не обязательно при считывании или записи каждый раз передавать адрес строки. Данные стали записывать или считывать блоками и адрес строки передавать только в начале блока. При этом можно сократить общее время обращения к динамическому ОЗУ и, тем самым, увеличить быстродействие компьютера.

Такое обращение к динамическому ОЗУ называется быстрым страничным режимом доступа (FPM, Fast Page Mode). Длина считываемого блока данных обычно равна четырем словам. Время доступа к памяти принято оценивать в тактах системной шины процессора. В обычном режиме доступа к памяти оно одинаково для всех слов. Поэтому цикл обращения к динамической памяти можно записать как 5-5-5-5. При режиме быстрого страничного доступа цикл обращения к динамической памяти можно записать как 5-3-3-3, т. е. время обращения к первой ячейке не изменяется по сравнению с предыдущим случаем, а считывание последующих ячеек сокращается до трех тактов. При этом среднее время доступа к памяти сокращается почти в полтора раза. Временная диаграмма режима FPM приведена на рис. 19.40.

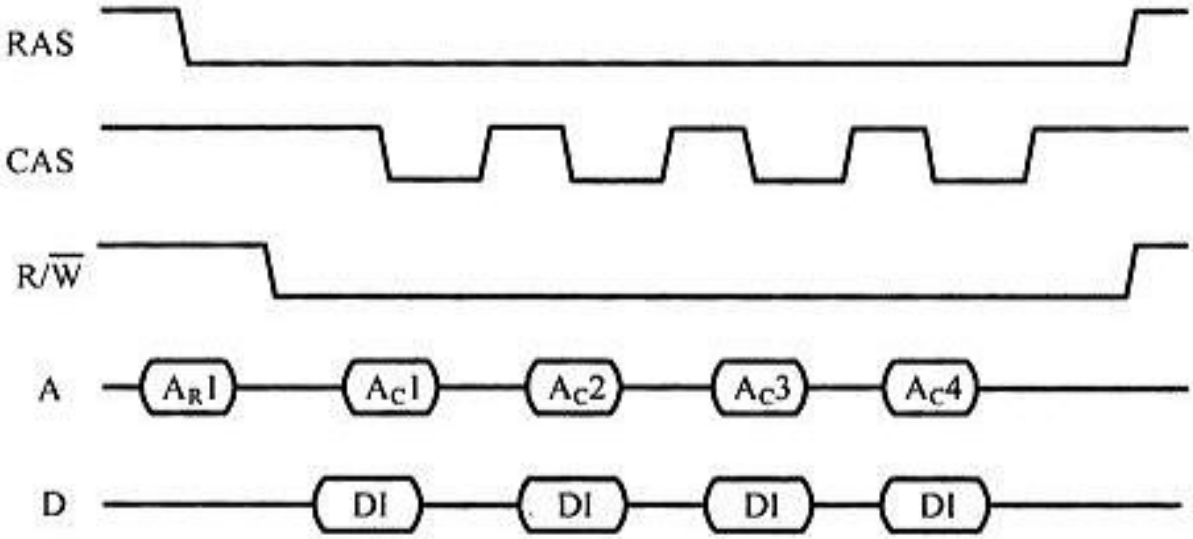


Рис. 19.40. Временная диаграмма записи в динамическое ОЗУ в режиме FPM

Еще одним способом увеличения быстродействия ОЗУ является применение микросхем EDO (Extended Data Out — ОЗУ с расширенным выходом данных). В EDO ОЗУ усилители-регенераторы не сбрасываются по окончании строба CAS#, поэтому времени для считывания данных в таком режиме больше. Теперь, для того чтобы сохранить время считывания на прежнем уровне, можно увеличить тактовую частоту системной шины и тем самым увеличить быстродействие компьютера. Для EDO ОЗУ цикл обращения к динамической памяти можно записать как 5-2-2-2.

Следующим шагом в развитии схем динамического ОЗУ было применение в составе ОЗУ счетчика столбцов. То есть при переходе адреса ячейки к следующему столбцу запоминающей матрицы адрес столбца инкрементируется (увеличивается) автоматически. Такое ОЗУ получило название BEDO (ОЗУ с пакетным доступом). В этом типе ОЗУ удалось достигнуть режима обращения к динамической памяти 5-1-1-1.

В синхронном динамическом ОЗУ (SDRAM) увеличение быстродействия получается за счет применения конвейерной обработки сигнала. Как известно, при использовании конвейера можно разделить операцию считывания или записи на отдельные подоперации, такие как выборка строк, выборка столбцов, считывание ячеек памяти, и производить эти операции одновременно. При этом, пока на выход передается считанная ранее информация, производится дешифрация столбца для текущей ячейки памяти и дешифрация строки для следующей ячейки памяти. Этот процесс иллюстрируется рис. 19.41.

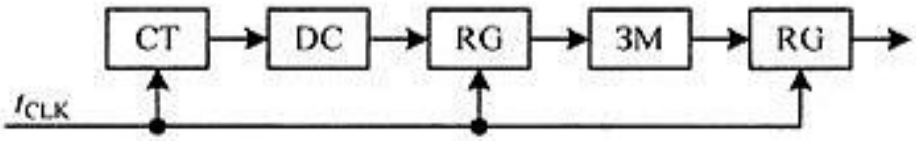


Рис. 19.41. Структурная схема конвейерной обработки данных

Из приведенного рисунка видно, что, несмотря на то что при считывании одной ячейки памяти время доступа к ОЗУ увеличивается, при считывании нескольких соседних ячеек памяти общее быстродействие микросхем синхронного динамического ОЗУ возрастает. Для сравнения, на рис. 19.41 приведена структурная схема обычного динамического ОЗУ. Время задержки распространения сигнала в этой схеме равно периоду тактового сигнала в шине обращения к ОЗУ, и определяется по формуле:

$$t_з = t_{CT} + t_{DC} + t_{ЗМ},$$

где t_{CT} — время срабатывания счетчика адреса динамического ОЗУ;

t_{DC} — время распространения сигнала дешифратора адреса;

$t_{ЗМ}$ — время появления сигнала на выходе запоминающей матрицы.

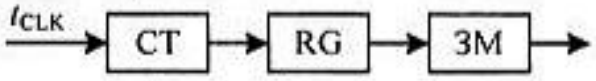


Рис. 19.42. Структурная схема обычного динамического ОЗУ

Время задержки распространения сигнала в схеме синхронного динамического ОЗУ (рис. 19.42) можно определить по формуле:

$$t_з = t_{CT} + t_{DC} + t_{RG} + t_{ЗМ} + t_{RG},$$

где t_{RG} — это записи в параллельный регистр.

То есть, время доступа к синхронному динамическому ОЗУ больше по сравнению с обычным динамическим ОЗУ. Однако период тактового сигнала

можно значительно уменьшить, т. к. он будет определяться максимальным значением из времен:

$$t_{обр} = \max\begin{cases} t_{CT} \\ t_{DC} + t_{RG} \\ t_{3M} + t_{RG} \end{cases}$$

Поэтому, несмотря на то что при обращении к одиночной ячейке памяти время доступа к SDRAM возрастает, при пакетном считывании последовательно расположенных байт общее время считывания оказывается значительно меньшим, т. к. все последующие данные на выходе ОЗУ будут появляться с периодом $t_{обр}$. Выигрыш при пакетной работе SDRAM может быть достаточно большим, т. к. при обращении к этому типу памяти допустимо устанавливать размер пакета данных, равный 256 слов.

Согласование быстродействия системной памяти и микропроцессора (кэш-память)

Универсальные микропроцессоры применяются в настольных или портативных компьютерах, а также во встраиваемых системах, и в настоящее время именно на них отрабатываются самые передовые решения по повышению быстродействия микросхем.

Паразитные емкости печатной платы компьютера или другого устройства, в котором используется микропроцессор, не позволяют достигнуть предельного быстродействия, с которым может работать кристалл микропроцессора. Кроме того, невозможно реализовать кварцевые резонаторы на частоты, на которых работают современные микропроцессоры. Поэтому внутренняя и внешняя тактовая частота микропроцессора различаются. Обычно внутренняя тактовая частота в несколько раз выше внешней.

Умножение внешней тактовой частоты внутри кристалла процессора производится при помощи схемы фазовой автоподстройки частоты, поэтому для установления стабильной внутренней частоты микропроцессора требуется некоторое время, определяемое обычно десятками микросекунд. Схему подобного умножителя частоты мы уже рассматривали в предыдущих главах.

Первым фактором, ограничивающим быстродействие микропроцессорной системы в целом, является то, что для увеличения доступной емкости системной памяти компьютера используют микросхемы ОЗУ динамического вида, однако они обладают относительно невысоким быстродействием. В результате возникает противоречие между высоким быстродействием микро-

процессора и недостаточным быстродействием системной памяти, что ограничивает производительность микропроцессорной системы в целом.

В качестве решения этой проблемы в современных компьютерах предлагается использование кэш-памяти. Эта память с точки зрения программиста никак не видна и общий объем системной памяти вследствие ее наличия не увеличивается.

Кэш-память выполняется в виде статической памяти небольшого размера и высокого быстродействия. Она ставится как буфер между основной памятью и микропроцессором. Кэш-память в компьютерах располагается на материнской плате. Естественно, что при первом обращении к системной памяти быстродействие снижается на задержку, вносимую копированием информации в кэш-память. Выигрыш в быстродействии достигается при повторном обращении к одному и тому же участку памяти. В этом случае обращение к основной памяти не требуется, т. к. в кэш-памяти уже хранится копия содержимого основной памяти. Учитывая, что выполнение программ обычно реализуется в виде циклов, когда один и тот же участок программного кода повторяется многократно, общее быстродействие системы в целом будет определяться быстродействием кэш-памяти. Всю логику работы с кэш-памятью выполняет контроллер памяти, входящий в набор микросхем (chip set) материнской платы компьютера.

Рассмотренный выше метод увеличивает общее быстродействие системной памяти, но только до значения тактовой частоты системной шины (внешняя тактовая частота микропроцессора). Согласовать внутреннее быстродействие микропроцессора и быстродействие системной шины позволяет применение внутренней кэш-памяти. Естественно, ее объем меньше, чем у кэш-памяти, расположенной на материнской плате компьютера.

При рассмотрении принципов работы цифровых микросхем мы узнали, что потребляемый микросхемой ток определяется быстродействием микросхемы, поэтому внутренняя кэш-память в свою очередь разделяется на два уровня: первый уровень малого объема, но высокого быстродействия, совпадающего с внутренним быстродействием микропроцессора, и второй уровень, с большим объемом памяти, но с меньшим быстродействием. Кэш-память, расположенную на материнской плате, называют, продолжая нумерацию, кэш-памятью третьего уровня.

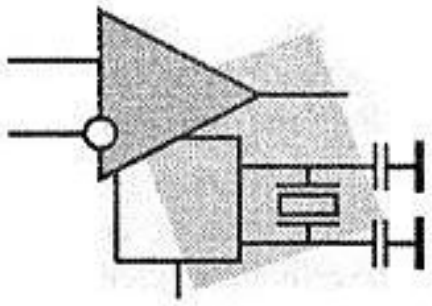
Итоги

В главе были рассмотрены схемы подключения к микропроцессору устройств хранения, ввода и вывода данных. Кроме того, были рассмотрены основные методы расширения адресного пространства микропроцессорной системы и

некоторые решения, позволяющие повысить ее быстродействие. Структурные схемы приведены с уровнем детализации, достаточным для превращения их в принципиальные схемы.

В настоящее время никто не разрабатывает схемы, подобные рассмотренным в данной главе, ведь это стандартные схемы. Поэтому сейчас на мировом рынке представлено огромное количество готовых микросхем, построенных по рассмотренным принципам. Теперь можно перейти к изучению этих микросхем, представляющих собой универсальные цифровые устройства.

ГЛАВА 20



Принципы работы микроконтроллеров

В предыдущей главе мы познакомились с принципами построения микропроцессорных систем. Однако цель нашей книги — научиться работать с микроконтроллерами, или как их раньше называли — однокристальными микроЭВМ. Внутри эти микросхемы устроены подобно микропроцессорной системе, пример которой мы рассматривали ранее. Тем не менее у каждого микроконтроллера есть индивидуальные отличия. Ряд дополнительных внутренних узлов микроконтроллера позволяет значительно упростить принципиальную схему разрабатываемого устройства. Рассмотрим эти дополнительные узлы и особенности их применения на примере самого распространенного на данный момент семейства микроконтроллеров.

Напомним, что при изучении (и разработке) микропроцессорной техники необходимо иметь в виду две разные модели устройства — схемотехническую и программистскую. Под схемотехнической моделью подразумевается перечень тех аппаратных средств (внутренних устройств), которые включены в состав микроконтроллера, и особенности их схемной реализации. В программистской модели указываются формат регистров микропроцессора, адреса и формат управляющих регистров, область адресов ОЗУ и ПЗУ. В этой главе в основном будут рассматриваться схемотехнические особенности микросхем, принадлежащих к выбранному семейству микроконтроллеров. Кроме того, будут рассмотрены особенности подключения наиболее распространенных узлов к данным микросхемам при разработке принципиальной схемы устройства.

Внутренние устройства микроконтроллеров нужно уметь использовать, ведь именно наличие таких устройств и позволяет упростить принципиальную схему разрабатываемого микропроцессорного устройства. Поэтому там, где это необходимо, будут приведены участки программ, позволяющие настроить внутренние блоки микроконтроллеров для работы с подключенными

к этой микросхеме внешними устройствами. Если чтение программ вызовет определенные трудности, то можно почитать описание языка программирования, приведенное в последующих главах, а затем вернуться к этой главе.

Очень важной характеристикой микропроцессорного устройства и микроконтроллеров, как одного из представителей микропроцессоров, является система команд. Именно особенности построения команд характеризуют микропроцессоры, определяют их алгоритмическое быстродействие и требования к объему внутренней памяти, поэтому в данной главе подробно рассматриваются команды и виды адресации, используемые в этих командах, для выбранного семейства микроконтроллеров.

Семейство микроконтроллеров MCS-51

В настоящее время среди всех 8-разрядных микроконтроллеров семейство MCS-51 является несомненным чемпионом по числу разновидностей и количеству компаний, выпускающих его модификации. Первый представитель этого семейства — микроконтроллер 8051, выпущенный в 1980 г. на базе технологии n-МОП.

Удачный набор периферийных устройств, возможность гибкого выбора внешней или внутренней программной памяти и приемлемая цена обеспечили этому микроконтроллеру успех на рынке. С точки зрения технологии микроконтроллер 8051 являлся для своего времени очень сложным изделием: в кристалле было использовано 128 тыс. транзисторов, что в 4 раза превышало количество транзисторов в 16-разрядном микропроцессоре 8086. Важную роль в достижении такой высокой популярности семейства 8051 сыграла открытая политика фирмы Intel, родоначальницы архитектуры, направленная на широкое распространение лицензий на ядро 8051 среди большого количества ведущих полупроводниковых компаний мира. В результате на сегодняшний день существует более 200 модификаций микроконтроллеров семейства 8051, выпускаемых почти двадцатью компаниями. Эти модификации включают в себя кристаллы с широчайшим спектром периферии: от простых 20-выводных устройств с одним таймером и 1К программной памяти до сложнейших 100-выводных кристаллов с 10-разрядными АЦП, массивами таймеров-счетчиков, аппаратными 16-разрядными умножителями и 64К программной памяти на кристалле. Каждый год появляются все новые варианты представителей этого семейства. Основными направлениями развития микроконтроллеров являются:

- увеличение быстродействия (повышение тактовой частоты и переработка архитектуры);

- снижение напряжения питания и потребления;
- увеличение объема ОЗУ и флэш-памяти на кристалле с возможностью внутрисхемного программирования;
- введение в состав периферии микроконтроллера сложных устройств типа системы управления приводами, CAN- и USB-интерфейсов и т. п.

Очень важным направлением развития микроконтроллеров является производство микросхем в маленьких корпусах. Это позволяет осуществлять проектирование и производство малогабаритной аппаратуры.

Микросхемы семейства MCS51 производятся рядом фирм различных стран мира, таких как Philips, Siemens, Intel, Atmel, Dallas, Temic, Oki, AMD, MHS, Gold Star, Winbond, Silicon Systems и ряд других. Микроконтроллеры семейства MCS-51 выпускают и заводы СНГ. Производство микроконтроллера 8051 осуществляется в Киеве, Воронеже (1816ВЕ31/51, 1830ВЕ31/51), Минске (1834ВЕ31) и Новосибирске (1850ВЕ31). В качестве примера, в табл. 20.1 приведены названия нескольких микросхем, производимых зарубежными фирмами, в табл. 20.2 — микросхемы российского производства.

Таблица 20.1. *Микросхемы семейства MCS-51, производимые зарубежными фирмами*

Микроконтроллер	ОЗУ	ПЗУ	EEPROM	UART	SPI	Таймеры	PCA	АЦП
AT89c1051	128 байт	1 Кбайт	—	1	—	2	—	комп.
AT89c2051	128 байт	2 Кбайт	—	1	—	2	—	комп.
80c51	128 байт	4 Кбайт	—	1	—	2	—	—
80c31	128 байт	—	—	1	—	2	—	—
8Xc52	256 байт	8 Кбайт	—	1	—	3	—	—
AT89c52	256 байт	8 Кбайт	—	1	—	3	—	—
AT89c8252	256 байт	8 Кбайт	2 Кбайт	1	1	3	—	—
AT89c55	256 байт	20 Кбайт	—	1	—	3	—	—
8Xc54	256 байт	16 Кбайт	—	1	—	3	—	—
8Xc58	256 байт	32 Кбайт	—	1	—	3	—	—
8Xc51FA	128 байт	8 Кбайт	—	1	—	4	1	—
8Xc51FB	256 байт	16 Кбайт	—	1	—	4	1	—
8Xc51FC	256 байт	32 Кбайт	—	1	—	4	1	—

Таблица 20.1 (окончание)

Микро-контроллер	ОЗУ	ПЗУ	EEPROM	UART	SPI	Тай-меры	PCA	АЦП
8Xc51GB	256 байт	8 Кбайт	—	1	1	5	2	10 р
AduC834	4 Кбайт	64 Кбайт	2 Кбайт	1	1	3	—	24 р

Примечание.

1. Вместо символа 'X' в названии микроконтроллера должны стоять символы:
   0 — в микросхемах n-МОП без ПЗУ;
   3 — в микросхемах n-МОП с ПЗУ;
   7 — в микросхемах n-МОП с РПЗУ;
   0с — в микросхемах КМОП без ПЗУ;
   3с — в микросхемах КМОП с ПЗУ;
   7с — в микросхемах КМОП с РПЗУ;
   9с — в микросхемах КМОП с FLASH-памятью.
2. комп. — аналоговый компаратор.
3. 10 р — количество разрядов во встроенном АЦП.

Таблица 20.2. Микросхемы семейства MCS -51 российского производства

Микроконтроллер	ОЗУ	ПЗУ	ППЗУ	Таймер 2	PCA	PCA1	АЦП
КР1816ВЕ51	128 байт	4 Кбайт	—	—	—	—	—
КР1816ВЕ751	128 байт	—	4 Кбайт	—	—	—	—
КР1816ВЕ31	128 байт	—	—	—	—	—	—
КР1830ВЕ51	128 байт	4 Кбайт	—	—	—	—	—
КР1830ВЕ751	128 байт	—	4 Кбайт	—	—	—	—
КР1830ВЕ31	128 байт	—	—	—	—	—	—

Примечание.

Серия микросхем 1816 выполнена по n-МОП-технологии.

Серия микросхем 1830 выполнена по КМОП-технологии.

Архитектура микроконтроллеров MCS-51

Архитектура семейства MCS-51 в значительной мере предопределяет ее назначение — это построение компактных и дешевых цифровых устройств. Все функции микроконтроллера реализуются с помощью единственной микро-

схемы. В состав семейства MCS-51 входит ряд микросхем от самых простых микроконтроллеров до достаточно сложных. Микроконтроллеры семейства MCS-51 позволяют выполнять как задачи управления различными устройствами, так и реализовывать простейшие алгоритмы цифровой обработки сигналов. Все микросхемы этого семейства работают с одной и той же системой команд. Большинство микросхем выполняется в одинаковых корпусах с совпадающей цоколевкой (схемой расположения выводов). Это позволяет использовать для разработанного устройства микросхемы разных фирм-производителей (таких как Intel, Dallas, Atmel, Philips и т. д.) без переделки принципиальной схемы устройства и программы.

Структурная схема микроконтроллера представлена на рис. 20.1 и состоит из следующих основных функциональных узлов:

- блока управления;
- арифметико-логического блока;
- блока таймеров/счетчиков;
- блока последовательного интерфейса и прерываний;
- программного счетчика, памяти данных и памяти программ.

Двусторонний обмен данными между элементами внутренней структуры микроконтроллера осуществляется с помощью внутренней 8-разрядной шины данных.

По такой схеме построены практически все представители семейства MCS-51. Различные микросхемы этого семейства различаются только регистрами специального назначения (в том числе и количеством портов). Система команд всех контроллеров семейства MCS-51 содержит 111 базовых команд длиной 1, 2 или 3 байта и не изменяется при переходе от одной микросхемы к другой. Это обеспечивает прекрасную переносимость программ с одной микросхемы на другую. Рассмотрим подробнее назначение каждого блока.

Блок управления и синхронизации предназначен для выработки синхронизирующих и управляющих сигналов, обеспечивающих координацию совместной работы блоков микроконтроллера во всех допустимых режимах его работы. В состав блока управления входят:

- устройство формирования временных интервалов;
- логика ввода-вывода;
- регистр команд;
- регистр управления потреблением электроэнергии;
- дешифратор команд, логика управления микроконтроллером.

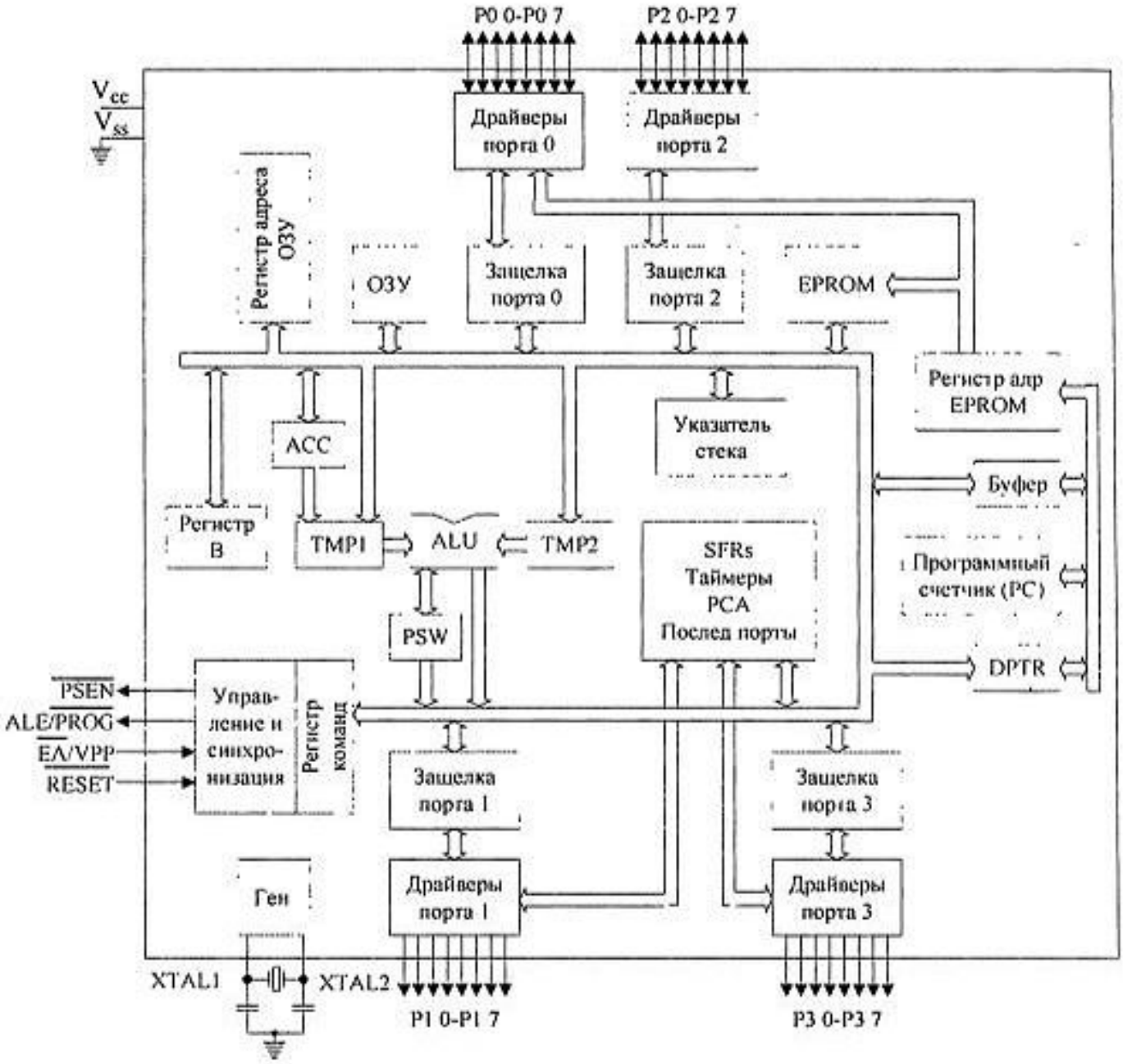


Рис. 20.1. Структурная схема микроконтроллера К1830ВЕ751

Устройство формирования временных интервалов предназначено для формирования и выдачи внутренних синхросигналов фаз, тактов и циклов. Количество машинных циклов определяет продолжительность выполнения команд. Практически все команды микроконтроллера выполняются за один или два машинных цикла, кроме команд умножения и деления, продолжительность выполнения которых составляет четыре машинных цикла. Обозначим частоту задающего генератора через F_z. Тогда длительность машинного цикла равна $12 / F_z$ или составляет 12 периодов сигнала задающего генератора. Логика ввода-вывода предназначена для приема и выдачи сигналов, обеспечивающих обмен информации с внешними устройствами через порты ввода-вывода P0–P3.

Регистр команд предназначен для записи и хранения 8-разрядного кода операции выполняемой команды. С помощью дешифратора команд и логики управления микроконтроллера он преобразуется в микропрограмму выполнения заданной команды.

Регистр управления потреблением (PCON) позволяет останавливать микроконтроллер для уменьшения уровня помех и потребления электроэнергии. Еще большего уменьшения помех и потребления электроэнергии можно добиться, остановив задающий генератор микроконтроллера при помощи переключения битов регистра управления потреблением PCON. В вариантах микросхемы, изготовленных по технологии n-МОП (серия 1816 или иностранных микросхем, в названии которых в середине отсутствует буква 'c' или 's'), регистр управления потреблением PCON содержит только один бит, управляющий скоростью передачи последовательного порта SMOD, а биты управления потреблением электроэнергии отсутствуют.

Арифметико-логический блок (АЛБ) представляет собой параллельное 8-разрядное устройство, обеспечивающее выполнение арифметических и логических операций. АЛБ состоит из:

- *регистров временного хранения TMP1 и TMP2* (это восьмиразрядные регистры, предназначенные для приема и хранения операндов на время выполнения операций над ними. Регистры временного хранения программно не доступны, ими управляет только микропрограмма выполнения команд);
- *ПЗУ констант* (обеспечивает выработку корректирующего кода при двоично-десятичном представлении данных или кода маски при битовых операциях и констант);
- *арифметико-логического устройства* (представляет собой схему комбинационного типа с последовательным переносом, предназначенную для выполнения арифметических операций сложения, вычитания и логических операций "И", "ИЛИ", суммирования по модулю два и инвертирования);
- *дополнительного регистра (регистра B)* (восьмиразрядный регистр, используемый во время операций умножения и деления. Для других инструкций он может рассматриваться как дополнительный регистр внутренней памяти микроконтроллера);
- *аккумулятора (ACC)* (восьмиразрядный регистр, предназначенный для приема и хранения результата, полученного при выполнении арифметико-логических операций или операций сдвига);
- регистра состояния программ (PSW).

Блок последовательного интерфейса и прерываний предназначен для организации последовательного ввода-вывода информации и организации прерываний выполнения программы.

В состав этого блока входят:

- логика управления;
- регистр управления;
- буфер передатчика;
- буфер приемника;
- приемопередатчик последовательного порта;
- регистр приоритетов прерываний;
- регистр разрешения прерываний;
- логика обработки флагов прерываний.

Счетчик команд предназначен для формирования текущего 16-разрядного адреса внутренней или внешней памяти программ. В состав счетчика команд входят 16-разрядные буфер счетчика команд, регистр счетчика команд и схема инкремента (увеличения содержимого на 1).

Память данных предназначена для временного хранения информации, используемой в процессе выполнения программы.

Порты P0, P1, P2, P3 являются квазидвунаправленными портами ввода-вывода и предназначены для обеспечения обмена информацией между микроконтроллером и внешними устройствами, образуя 32 линии ввода-вывода.

Регистр состояния программы (PSW) предназначен для хранения информации о состоянии АЛУ при выполнении программы.

Память программ предназначена для хранения программного кода и представляет собой постоянное запоминающее устройство (ПЗУ). В разных микросхемах применяются масочные, стираемые ультрафиолетовым излучением или FLASH ПЗУ.

Регистр указателя данных (DPTR) предназначен для формирования 16-разрядного адреса внешней памяти данных или памяти программ при считывании таблиц констант.

Указатель стека (SP) представляет собой 8-разрядный регистр, предназначенный для организации особой области памяти данных (стека), в которой хранятся адреса возврата из подпрограмм, переменные и содержимое внутренних регистров микроконтроллера (в том числе регистры PSW и аккумулятор).

Система команд микроконтроллеров MCS-51

Ни один из видов микропроцессоров не может рассматриваться отдельно от системы команд. Не является исключением и семейство микроконтроллеров MCS-51. Система команд микроконтроллеров этого семейства предоставляет большие возможности обработки данных, обеспечивает реализацию логических, арифметических операций, а также управление устройствами в режиме реального времени.

В этой системе команд реализованы побитная, потетрадная (4 бита), побайтовая (8 бит) и 16-разрядная обработка данных. Микросхемы семейства MCS-51 — это 8-разрядные микропроцессоры, а это означает, что ПЗУ, ОЗУ, регистры специального назначения, АЛУ и внешние шины имеют байтовую организацию. Двухбайтовые данные используются только регистром-указателем (DTPR) и счетчиком команд (PC).

В машинном коде команда занимает один, два или три байта в зависимости от типа адресации. Команды выполняются за один, два или четыре (умножение и деление) машинных цикла. Здесь следует отметить, что в ряде современных микросхем, принадлежащих к этому семейству, команды выполняются значительно быстрее. Вплоть до того, что в процессорах фирмы sygnal команды выполняются за два такта синхронизации (одна шестая машинного цикла).

Запись команд в машинных кодах для человека неудобна. Кроме того, машинные команды, отличающиеся младшими битами или вторым байтом, выполняют одинаковые действия над разными ячейками памяти. Поэтому для записи команд микропроцессоров была придумана система мнемонических обозначений. В записи команды микропроцессора сначала ставится мнемоническое обозначение кода операции, затем указывается ячейка памяти — приемник результата выполнения операции и, наконец, источник данных для выполнения операции. Например, в команде

```
E535    MOV A, 35h
```

символы MOV обозначают операцию копирования, второй операнд 35 определяет, что данные необходимо взять из ячейки памяти с шестнадцатеричным адресом 35, а первый операнд A определяет, что результат необходимо поместить в регистр-аккумулятор. При этом старое значение регистра-аккумулятора будет стерто. Слева приведена машинная команда микроконтроллера в шестнадцатеричной записи, соответствующая находящейся справа мнемонической записи команды.

Мнемоническое обозначение кода операции отделяется от операндов одним или несколькими символами пробела или табуляции, а операнды отделяются

друг от друга запятыми. При этом между операндами допустимо использование символов пробела или табуляции, которые могут потребоваться для более наглядной записи команды.

Если операция требует для выполнения двух источников и одного приемника результата операции (например, команда сложения ADD или вычитания SUBB), то первый операнд является одновременно и источником и приемником результата операции. Например, в команде

```
2535    ADD A, 35h
```

символы ADD обозначают операцию сложения двух чисел, данные будут взяты из ячейки памяти с шестнадцатеричным адресом 35 и аккумулятора, а результат будет помещен в аккумулятор вместо старого значения этого регистра.

В табл. 20.1 приведены мнемонические обозначения машинных команд, влияющих на значения флагов регистра слова состояния программы PSW, а в табл. 20.2 приведены обозначения и символы, используемые при описании команд микроконтроллеров семейства MCS-51.

Команды микроконтроллера условно можно разбить на пять групп:

- арифметические команды;
- логические команды с байтовыми переменными;
- команды передачи данных;
- команды битового процессора;
- команды ветвления программ и передачи управления ОЭВМ.

Арифметические команды

В наборе команд микроконтроллера имеются следующие арифметические операции:

- сложение — ADD;
- сложение с учетом флага переноса — ADDC;
- вычитание с заемом из флага переноса — SUBB;
- инкрементирование (увеличение на 1) — INC;
- декрементирование (уменьшение на 1) — DEC;
- десятичная коррекция — DA;
- умножение — MUL;
- деление — DIV.

Действия производятся над целыми беззнаковыми восьмиразрядными числами. Арифметические операции в рассматриваемой системе команд могут быть выполнены только над содержимым регистра-аккумулятора.

При операции умножения содержимое аккумулятора A умножается на содержимое регистра B, и результат размещается следующим образом: младший байт в регистре B, старший — в регистре A.

В случае выполнения операции деления целое от деления помещается в аккумулятор A, остаток — в регистр B.

Логические команды с байтовыми переменными

Система команд рассматриваемого микроконтроллера позволяет реализовать следующие логические операции:

- логическое "И" — `ANL`;
- логическое "ИЛИ" — `ORL`;
- исключающее "ИЛИ" — `XRL`.

Логические операции могут выполняться только над аккумулятором или над портами ввода-вывода.

Существуют логические операции, которые выполняются только на аккумуляторе:

- сброс всех восьми разрядов A — `CLR A`;
- инвертирование всех восьми разрядов A — `CPL A`;
- циклический сдвиг влево и вправо без учета флага переноса — `RL A` и `RR A`;
- циклический сдвиг влево и вправо с учетом флага переноса — `RLC A` и `RRC A`;
- обмен местами старшей и младшей тетрад внутри аккумулятора — `SWAP A`.

Команды пересылки данных

Как было рассмотрено ранее, арифметические и логические команды могут быть выполнены только над содержимым регистра-аккумулятора, поэтому исключительное значение в системе команд приобретают команды пересылки данных. С помощью этих команд можно скопировать содержимое любой ячейки памяти в регистр-аккумулятор или, наоборот, скопировать содержимое аккумулятора в любую ячейку памяти. Так как в микроконтроллере присутствует три независимых области памяти, то для обращения к ним введены различные команды:

- копирование данных во внутреннем ОЗУ — MOV;
- обмен данными аккумулятора с внутренним ОЗУ — XCH, XCHD;
- копирование из внешней памяти данных — MOVX;
- копирование данных из памяти программ — MOVC.

Рассмотрим несколько примеров использования команд пересылки данных.

Любая из 256 ячеек внутреннего ОЗУ данных может быть выбрана с использованием косвенно-регистровой адресации через регистры-указатели R0 и R1 (выбранного банка рабочих регистров):

Листинг 20.1. Пример команд копирования данных

```
MOV A, @R0      ;Скопировать число из ячейки памяти с адресом,
                ;хранящимся в R0, в аккумулятор
MOV @R1, A      ;Скопировать число из аккумулятора, в ячейку памяти с
                ;адресом, хранящимся в R1
```

Команды пересылки между ячейками памяти, использующие прямую адресацию, позволяют заносить содержимое порта в ячейку внутреннего ОЗУ или пересылать содержимое из одной ячейки внутреннего ОЗУ в другую без использования аккумулятора:

```
MOV 15, 25          ;Скопировать содержимое 25-й ячейки  в 15-ю ячейку
```

Таблицы символов (кодов), записанные в ПЗУ программы, могут быть скопированы в аккумулятор с помощью команд передачи данных с косвенной адресацией, например:

```
MOVC A, @A+DPTR     ;Скопировать символ в аккумулятор
```

Ячейка адресного пространства 64-килобайтного внешнего ОЗУ также может быть выбрана с использованием косвенно-регистровой адресации через регистр-указатель данных DPTR, например:

```
MOVX A, @DPTR ;Скопировать число из внешней ячейки памяти с адресом,
              ;хранящимся в DPTR, в аккумулятор
```

Содержимое аккумулятора может быть обменено с содержимым рабочих регистров выбранного банка, например:

```
XCH A, R0.
```

Битовые команды

Каждый бит из битового пространства внутренней памяти может быть установлен в 1, сброшен в 0, или инвертирован.

Эти операции можно выполнить при помощи следующих команд:

- установить бит (записать логическую единицу) — SETB;
- сбросить бит (записать логический ноль) — CLR;
- проинвертировать значение бита (изменить на противоположное) — CPL;
- записать бит во флаг переноса или считать из флага переноса — MOV.

По значениям бита могут быть реализованы переходы:

- если бит установлен (содержит логическую 1) — JB;
- если бит не установлен (содержит логический 0) — JNB;
- переход, если бит установлен со сбросом этого бита после выполнения команды (запись в этот бит 0) — JBC.

Между любым битом из битового пространства внутренней памяти и флагом переноса могут быть произведены логические операции "И" или "ИЛИ".

- "И" — ANL
- "ИЛИ" — ORL

Команды ветвления и передачи управления

Команды ветвления позволяют реализовывать условные операторы и операторы циклов. В микроконтроллерах семейства MCS-51 доступны следующие команды:

- безусловный переход: LJMP, AJMP, SJMP;
- вызов и возврат из подпрограммы: LCALL, ACALL, RET, RETI;
- переход в зависимости от результата проверки содержимого аккумулятора: JZ, JNZ, CJNE, JMP;
- переход в зависимости от значения флага переноса C: JC, JNC;
- переход в зависимости от значения любого бита в битовом пространстве: JB, JNB, JBC.

В системе команд присутствуют команды 16-разрядных безусловных переходов и вызовов подпрограмм. Они позволяют осуществить переход в любую точку адресного пространства памяти программ объемом до 64 Кбайт. Эти команды передачи управления в любую точку адресного пространства памяти программ приведены в листинге 20.2.

Листинг 20.2. Команды передачи управления в любую точку адресного пространства

```
LJMP Metka              ;Переход к команде, расположенной по адресу,
                        ;обозначенному меткой 'Metka'
```

```
LCALL Podprogramma      ;Запоминание адреса следующей команды и вызов
                        ;подпрограммы, расположенной по адресу,
                        ;обозначенному меткой 'Podprogramma'
```

Команды 11-разрядных переходов и вызовов подпрограмм позволяют сократить объем программы, но при этом обеспечивают переходы только внутри сегмента программной памяти размером 2 Кбайт. Эти команды принципиально могут приводить к необнаруживаемым транслятором ошибкам, когда программный модуль размещается на двух соседних 2-килобайтовых сегментах памяти. Примеры команд передачи управления внутри сегмента программной памяти размером 2 Кбайт приведены в листинге 20.3.

Листинг 20.3. Команды передачи управления в пределах 2 Кбайт

```
AJMP Metka              ;Переход к команде, расположенной по адресу,
                        ;обозначенному меткой 'Metka'
ACALL Podprogramma      ;Вызов подпрограммы по адресу,
                        ;обозначенному меткой 'Podprogramma', и запоминание
                        ;адреса следующей команды
```

Может возникнуть вопрос — а зачем же нужны такие команды? Подобные команды занимают в памяти программ только два байта, поэтому они могут сократить объем загрузочного кода программы. Часто при разработке программы требуется хранить ее готовые оттранслированные участки в виде отдельных файлов, а затем соединять их между собой. При этом абсолютные адреса команд, на которые следует передавать управление, обычно меняются. Это приводит к необходимости изменять машинные коды команд безусловного перехода, что в ряде случаев может оказаться неудобным, поэтому в состав команд введена команда безусловного перехода с адресацией относительно начального адреса следующей команды в пределах от (PC) – 127 до (PC) + 127. Такой вид адресации не приводит к изменению адресной части команды передачи управления. Это команда SJMP. Она, как и команды AJMP и ACALL, занимает в памяти программ два байта.

Может показаться, что пользоваться подобной командой неудобно, однако это не так. При передаче управления конкретной команде обычно пользуются меткой, а все необходимые вычисления производит программа-транслятор. Это означает, что, с точки зрения разработчика программы, ее исходный текст по сравнению с абсолютной адресацией не изменяется:

```
SJMP Metka              ;Переход к команде, расположенной по адресу,
                        ;обозначенному меткой 'Metka'
```

Кроме передачи управления в заранее известную точку программы часто требуется организовать выполнение того или иного участка программы в зави-

симости от определенного условия. Для этого в систему команд микроконтроллеров семейства MCS-51 введены команды условных и безусловных переходов. Они передают управление команде, находящейся относительно начального адреса следующей команды в пределах от (PC) – 127 до (PC) + 127 байт. Примеры команд условных переходов приведены в листинге 20.4.

Листинг 20.4. Примеры команд условных переходов

```
SJMP Metka              ;Переход к команде, расположенной по адресу,
                        ;обозначенному меткой 'Metka'
JB P3.5, TstNxtUsl      ;Если на выводе 6 порта P3 нулевой потенциал,
  ACALL Podprogramma      ;то вызвать подпрограмму, обозначенную меткой
                          ;'Podprogramma'
CJNE A, #5, TstNxtUsl   ;Если в аккумуляторе содержится число 5,
  ACALL Podprogramma      ;то вызвать подпрограмму, обозначенную меткой
                          ;'Podprogramma'
```

Команды условного перехода в зависимости от результата анализа содержимого аккумулятора или значения флага переноса C могут быть использованы для реализации проверки различных условий. При этом содержимое проверяемой ячейки памяти не изменяется. Это означает, что если требуется произвести несколько проверок одной и той же переменной, то повторно заносить значение этой переменной в аккумулятор не нужно. Примеры подобных конструкций приведены в листинге 20.5.

Листинг 20.5. Примеры проверки наиболее распространенных условий

```
  MOV A, 34             ;Если в переменной, хранящейся в ячейке внутренней
  JNB ACC_7, TstEQ5     ;памяти с адресом 34, число меньше нуля,
    CALL Podprogramma     ;то вызвать подпрограмму, обозначенную меткой
                          ;'Podprogramma'
TstEQ5 ;-------------------------------------------------------------
  CJNE A,#5,TstLT5      ;Если в переменной, хранящейся в ячейке внутренней
                        ;памяти с адресом 34, занесено число 5,
    CALL Podpr5           ;то вызвать подпрограмму, обозначенную меткой
                          ;'Podpr5'
TstLT5 ;-------------------------------------------------------------
  JNC TstGE5            ;Если в переменной, хранящейся в ячейке внутренней
                        ;памяти с адресом 34, занесено число, меньшее 5,
    CALL PodprLT5         ;то вызвать подпрограмму, обозначенную меткой
                          ;'PodprLT5'
```

```
TstGE5 ;-------------------------------------------------------------
  JC TstNxtUsl          ;Если в переменной, хранящейся в ячейке внутренней
                        ;памяти с адресом 34 занесено число, большее или
                        ;равное 5,
    CALL PodprGE5         ;то вызвать подпрограмму, обозначенную меткой
                          ;'PodprGE5'
TstGT5 ;-------------------------------------------------------------
  CJNE A,#6,$+3         ;Если в переменной, хранящейся в ячейке внутренней
  JC TstNxtUsl          ;памяти с адресом 34, занесено число, большее 5,
    CALL PodprGT5         ;то вызвать подпрограмму, обозначенную меткой
                          ;'PodprGT5'
;-------------------------------------------------------------------
  JNC TstNxtUsl         ;Если в переменной, хранящейся в ячейке внутренней
                        ;памяти 34, занесено число, меньшее или равное 5,
    CALL PodprLE5         ;то вызвать подпрограмму, обозначенную меткой
                          ;'PodprLE5'
```

Косвенный переход `JMP @A+DPTR` в системе команд микроконтроллеров семейства MCS-51 обеспечивает ветвление программы по содержимому аккумулятора A. Это позволяет реализовывать операцию перехода по заданному коду, эквивалентную оператору `case` в языке программирования Pascal, но намного быстрее (за два машинных цикла). Использование в этой команде указателя данных `DPTR` позволяет размещать таблицу переходов в любом месте памяти программ. Пример реализации команды выбора варианта:

Листинг 20.6. Пример реализации оператора выбора нескольких вариантов

```
       BeginOpCase:   ;НАЧАЛО ОПЕРАТОРА ВЫБОРА ВАРИАНТОВ------------
900008   MOV DPTR, #JMP_TBL ;Задать начальный адрес таблицы адресов
                            ;переходов

E521     MOV A, 33          ;В ячейке с адресом 33 хранится переменная,
                            ;по содержимому которой необходимо
                            ;осуществить переход на обслуживающую
                            ;программу

C3       CLR C              ;Осуществить арифметический сдвиг
33       RLC A              ;аккумулятора вправо (умножить на 2), т. к.
                            ;команды переходов занимают два байта

73       JMP @A+DPTR        ;Перейти к выполнению заданного в  ячейке 33
                            ;программного кода
```

```
;НАЧАЛО ТАБЛИЦЫ ПЕРЕХОДОВ ПО СОДЕРЖИМОМУ ПЕРЕМЕННОЙ В ЯЧЕЙКЕ ПАМЯТИ 33
      JMP_TBL:
020000   JMP Case0        ;Перейти к выполнению кода по числу 0
020000   JMP Case1        ;Перейти к выполнению кода по числу 1
020017   JMP EndCase      ;Это число в ячейке 33 в список разрешенных не
                          ;входит
020017   JMP EndCase      ;Это число в ячейке 33 в список разрешенных не
                          ;входит
020000   JMP Case4        ;Перейти к выполнению кода по числу 4
      EndCase:;-----------------------------------------------------
```

Способы адресации операндов

При определении способа адресации операндов в команде необходимо учитывать, что виды адресации для каждого операнда команды (источника или приемника) могут не совпадать.

Неявная адресация. При неявной адресации регистр-источник или регистр-приемник подразумевается в самом коде операции. Например:

Листинг 20.7. Примеры команд с неявной адресацией

```
03    RR A        ;Сдвинуть содержимое аккумулятора вправо
D4    DA A        ;Произвести десятичную коррекцию результата суммирования
E8    MOV A, R0   ;В первом операнде использована неявная адресация,
                  ;а во втором — регистровая
```

Регистровая адресация используется для обращения к восьми рабочим регистрам выбранного банка рабочих регистров, а также для обращения к регистрам A, B, AB (сдвоенному регистру), DPTR и к флагу переноса C. Номер регистра записывается в трех младших битах команды. Например:

Листинг 20.8. Примеры команд с регистровой адресацией

```
F8    MOV R0, A ;в первом операнде использована регистровая адресация,
                ;а во втором - неявная
```

Прямая байтовая адресация используется для обращения к ячейкам внутренней памяти (ОЗУ) данных (адреса 0 ... 127) и к регистрам специального назначения (адреса 128 ... 256). Адрес ячейки памяти помещается во второй байт команды. Например:

Листинг 20.9. Примеры команд с прямой байтовой адресацией

```
E520  MOV A, 20h ;во втором операнде использована прямая байтовая
                 ;адресация, а в первом - неявная
8D15  MOV 15h,R6 ;в первом операнде использована прямая байтовая
                 ;адресация, а во втором - регистровая
```

Прямая битовая адресация используется для обращения к отдельно адресуемым 128 битам, расположенным в ячейках с адресами 20H-2FH, и к отдельно адресуемым битам регистров специального назначения. Например:

Листинг 20.10. Примеры команд с прямой битовой адресацией

```
D220  SETB 20h  ;использована прямая битовая адресация
C215  CLR  15H  ;использована прямая битовая адресация
```

Косвенно-регистровая адресация используется для обращения к ячейкам внутреннего ОЗУ данных. В качестве регистров-указателей применяются регистры R0, R1 выбранного банка регистров. Пример использования косвенно-регистровой адресации для работы с ячейками внутреннего ОЗУ приведен в листинге 20.11.

Листинг 20.11. Примеры команд с косвенно-регистровой адресацией

```
E6    MOV A,@R0 ;В первом операнде использована неявная адресация, а во
                ;втором — косвенно-регистровая
F7    MOV @R1,A ;В первом операнде использована косвенно-регистровая
                ;адресация, а во втором — неявная
```

Косвенно-регистровая адресация используется также для обращения к внешней памяти данных. В этом случае с помощью регистров-указателей R0 и R1 (рабочего банка рабочих регистров) выбирается ячейка из блока 256 байт внешней памяти данных. Номер блока предварительно задается содержимым порта P2. Пример использования косвенно-регистровой адресации для работы с ячейками внешнего ОЗУ приведен в листинге 20.12.

Листинг 20.12. Примеры команд с косвенно-регистровой адресацией

```
E2    MOVX A,@R0 ;В первом операнде использована неявная адресация, а во
                 ;втором — косвенно-регистровая
F3    MOVX @R1,A ;В первом операнде использована косвенно-регистровая
                 ;адресация, а во втором — неявная
```

Если в качестве регистра-указателя используется 16-разрядный указатель данных (DPTR), то можно выбрать любую ячейку внешней памяти данных объемом до 64 Кбайт. (В некоторых моделях микроконтроллеров семейства MSC-51, таких как ADuC841, таким образом можно обращаться к внутренней памяти данных объемом более 256 байт.)

Листинг 20.13. Примеры команд с косвенно-регистровой адресацией

```
E0    MOVX A,DPTR;В первом операнде использована неявная адресация, а во
                 ;втором — косвенно-регистровая
F0    MOVX DPTR,A;В первом операнде использована косвенно-регистровая
                 ;адресация, а во втором — неявная
```

Косвенно-регистровая адресация по сумме базового и индексного регистра (содержимое аккумулятора A) упрощает просмотр таблиц, записанных в памяти программ. Любой байт из таблицы может быть выбран по адресу, определяемому суммой содержимого DPTR или PC и содержимого A, например:

Листинг 20.14. Примеры команд с косвенно-регистровой адресацией

```
83    MOVC A,@A+PC   ;В первом операнде использована неявная адресация, а
                     ;во втором — косвенно-регистровая
93    MOVC A,@A+DPTR;В первом операнде использована неявная адресация, а
                     ;во втором — косвенно-регистровая
```

Непосредственная адресация позволяет выбрать из адресного пространства памяти программ константы, явно указанные в команде, например:

Листинг 20.15. Примеры команд с непосредственной адресацией

```
7414     MOV A, #14h       ;В первом операнде использована неявная
                           ;адресация, а во втором — непосредственная
902048   MOV DPTR, #2048h ;В первом операнде использована неявная
                           ;адресация, а во втором — непосредственная
```

Полный список команд микроконтроллеров семейства MCS-51, упорядоченный по коду команды, приведен в *приложении 1*, а их подробное описание — в *приложении 2*. Теперь, после того как мы ознакомились с системой команд выбранного семейства микроконтроллеров, можно перейти к изучению особенностей построения его внутренних устройств.

Устройство параллельных портов микроконтроллеров MCS-51

Порты ввода-вывода P0, P1, P2, P3 являются квазидвунаправленными и обеспечивают обмен информацией между микроконтроллером и внешними устройствами, образуя 32 линии ввода-вывода. Каждый из портов содержит 8-разрядный регистр, имеющий байтовую и битовую адресацию для установки (запись логической 1) или сброса (запись логического 0) разрядов этого регистра с помощью программного обеспечения микроконтроллера. Выходы этих регистров соединены с внешними выводами микросхемы. Все разряды параллельных портов устроены одинаково. Упрощенная схема одного разряда параллельного порта микроконтроллера показана на рис. 20.2. Отличаются только выводы порта P0, у которых отсутствуют внутренние генераторы тока в верхней части схемы.

Один разряд регистра-защелки порта представляет собой D-триггер, запись входных данных в который происходит по высокому уровню синхросигнала. На рис. 20.2 этот сигнал назван "запись в защелку". Сигнал записанного бита с инвертирующего выхода триггера умощняется при помощи МОП-транзистора и поступает на внешний вывод микросхемы.

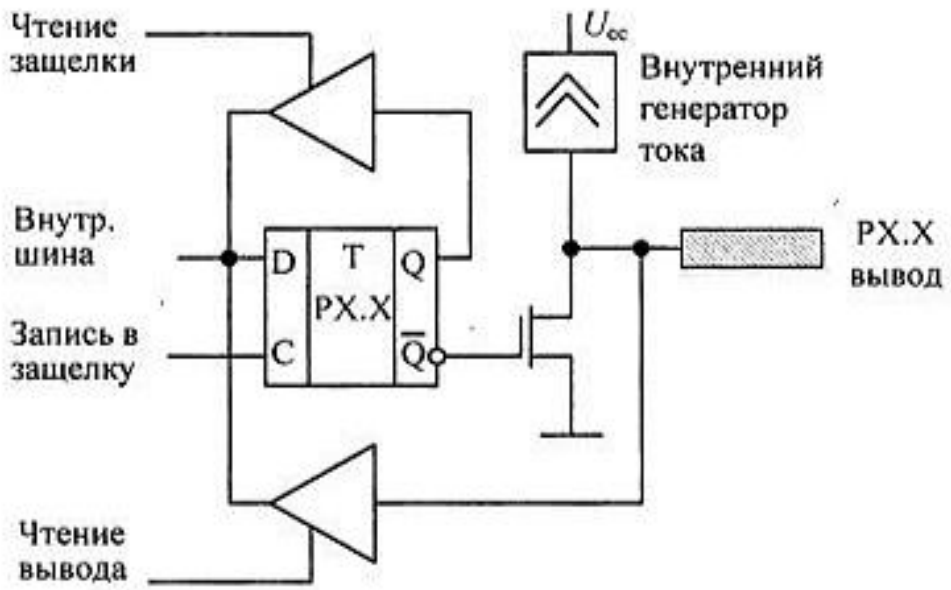


Рис. 20.2. Упрощенная схема одного бита порта

Естественно, что внутреннее устройство порта намного сложнее приведенного на рис. 20.2. Упрощение сделано для облегчения понимания работы параллельных портов микроконтроллера. Реально, параллельно генератору тока подключены цепи быстрого заряда паразитной емкости, схема альтернативной функции порта, а также триггер Шмитта, обеспечивающий надежное определение логического уровня сигнала, подаваемого на вывод параллельного

порта микроконтроллера. Желающие более подробно познакомиться с внутренним устройством параллельных портов микроконтроллера могут обратиться к техническому описанию конкретной микросхемы данного семейства микроконтроллеров.

Внутренняя схема порта построена таким образом, чтобы максимально упростить принципиальную схему подключения внешних устройств к микроконтроллеру. Например, внешний транзистор может быть подключен непосредственно базой к выводу микроконтроллера без дополнительного токоограничивающего резистора, как это показано на рис. 20.3. Это становится возможным благодаря внутреннему генератору тока, ограничивающему ток базы внешнего транзистора. В то же самое время, этот генератор тока не мешает непосредственно подключать к выводу микроконтроллера светодиод, т. к. значение его тока невелико. При этом, естественно, приходится для питания микроконтроллера использовать напряжение, равное напряжению, подаваемому на светодиод. Подобный вариант подключения тоже показан на рис. 20.3.

Значение сигнала непосредственно с внешнего вывода порта считывается по сигналу "чтение вывода". Однако при выполнении операций с отдельными битами требуется считывать содержимое внутреннего регистра-защелки порта. Если база биполярного транзистора непосредственно подключена к выводу порта, то считывание значения логической 1 с вывода порта становится невозможным. Дело в том, что напряжение на p-n-переходе база-эмиттер транзистора не может превысить ~0,7 В (для кремниевого транзистора). Такое напряжение однозначно будет воспринято цифровой схемой как логический ноль, поэтому при выполнении операций над отдельными битами параллельного порта считывание производится непосредственно с выхода регистра-защелки порта. При этом выход Q D-триггера подключается к внутренней шине (считывается) по сигналу "чтение защелки".

Чтение внешних выводов порта P3 осуществляется командами:

```
MOV A, P3       ;Скопировать состояние выводов порта P3 в аккумулятор
JB P3.4, Metka  ;Если на выводе 4 порта P3 логическая 1, то перейти на
                ;метку Metka
```

Чтение регистра-защелки осуществляется командами чтение-модификация-запись. Например:

```
CPL P3.1        ;Проинвертировать сигнал на выводе 1 порта P3
ORL P2, #56h    ;Установить единичный сигнал на выводах 1,2,4 и 6 порта P2
ANL P3, #03h    ;Установить нулевой сигнал на выводах 0 и 1 порта P3
SETB P3.1       ;Установить высокий потенциал на выводе 1 порта P3
```

Порты микросхемы служат для управления внешними устройствами, подключенными к микроконтроллеру. Схема подключения простейших внешних

устройств приведена на рис. 20.3. Он иллюстрирует особенности подключения светодиодных индикаторов к параллельным портам микроконтроллера MCS-51.

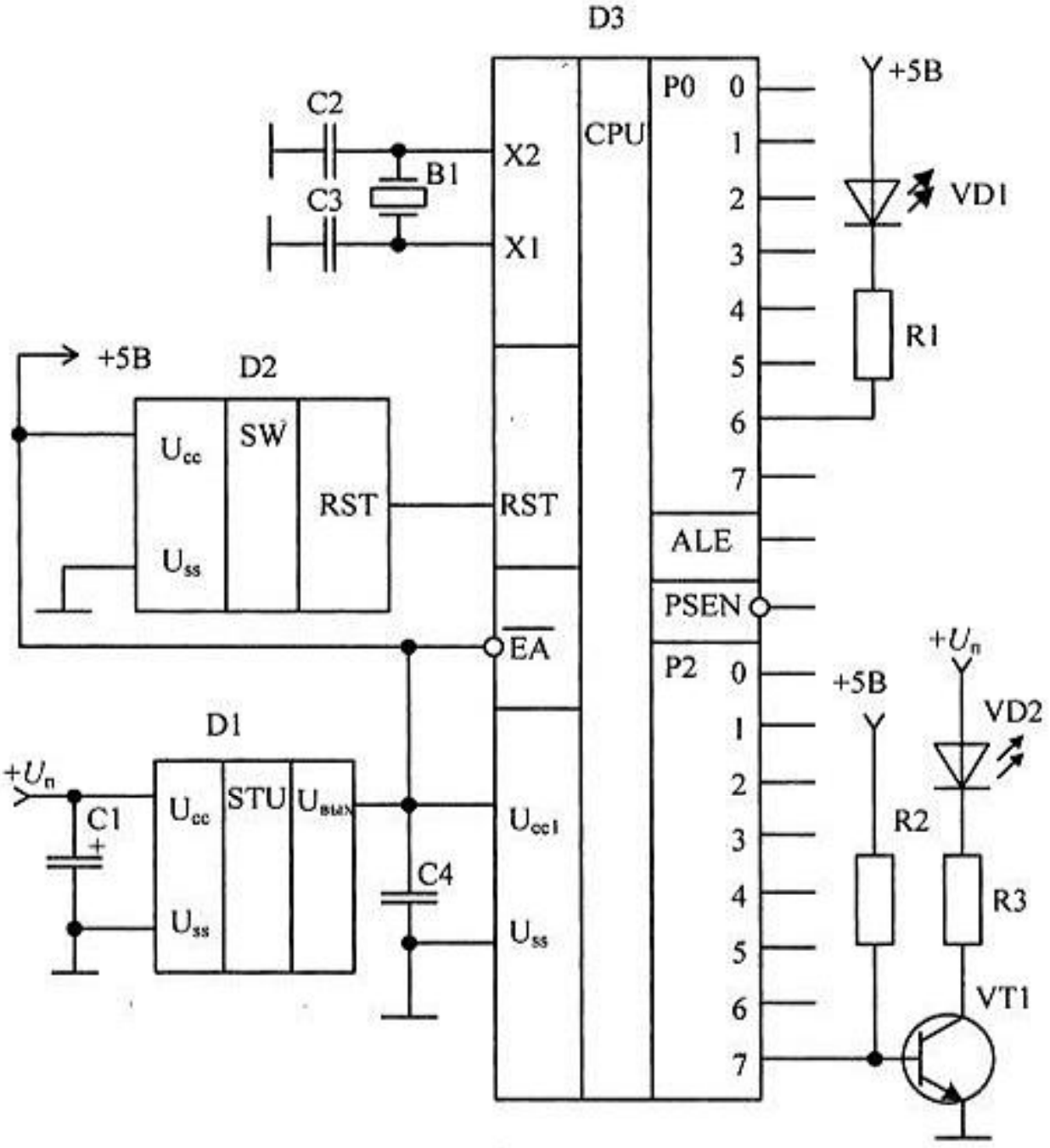


Рис. 20.3. Схема подключения светодиодных индикаторов к параллельному порту

Присутствие в схеме порта выходного мощного транзистора позволяет подключать к выводам порта светодиодные индикаторы непосредственно, без усилителя мощности, однако при этом необходимо следить за максимальной допустимой мощностью, рассеиваемой на микросхеме и отдельных выводах порта. Эквивалентная схема, на которой показан путь протекания выходного

тока порта, приведена на рис. 20.4. Как видно из приведенной схемы, именно выходной ток порта I_0 используется для зажигания светодиода.

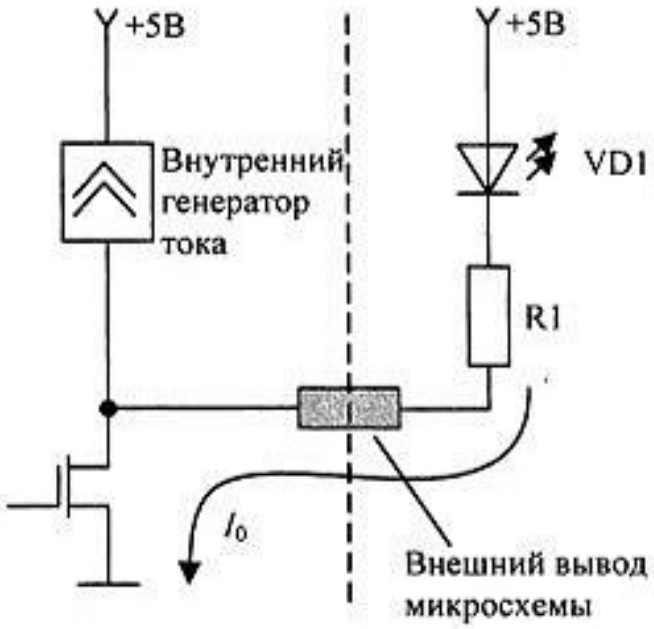


Рис. 20.4. Эквивалентная схема подключения светодиодного индикатора к параллельному порту

Для умощнения выводов порта можно применить транзисторный ключ, показанный также на рис. 20.3. Эта же схема используется при низковольтном питании микроконтроллера. Напряжения 3,3 В недостаточно для открывания p-n-перехода светодиода. ✧ *Обратите внимание*, что база транзистора подключена непосредственно к выводу порта. Это стало возможным только благодаря использованию в схеме порта генератора тока в верхнем плече выходного каскада.

Если выходного тока порта достаточно для открывания транзисторного ключа, то резистор R2 не используется. Этот резистор подключают для увеличения тока базы транзисторного ключа. На максимальное значение этого тока накладываются те же ограничения, что и для непосредственного подключения светодиодного индикатора к выводам порта. Резистор R3 рассчитывается исходя из допустимого тока через светодиод VD2. При возможности выбора напряжения питания для светодиодов лучше выбрать более высокое напряжение. Это позволит обеспечить более равномерное свечение светодиодов, т. к. разброс параметров светодиодов будет оказывать меньшее влияние на разброс токов, протекающих через них.

Микроконтроллеры предназначены для управления внешними устройствами. Однако изменять напряжения на выводах параллельного порта микроконтроллера можно только при помощи программы, записанной в память программ. Какие напряжения необходимо подавать на выходы микросхемы, за-

висит от схемы подключения индикатора. В приведенной на рис. 20.3 схеме для зажигания светодиода VD1, в разряд 6 порта P0 необходимо записать логический 0. Для зажигания светодиода VD2 необходимо в разряд 7 порта P2 записать логическую единицу, а для его гашения — логический ноль.

Чтобы изменять потенциалы на выводах микросхемы, можно воспользоваться следующими командами с байтовой адресацией:

1. MOV (пересылка), например:

```
MOV P1, #01110011b   ;Выдать на все восемь выводов P1 число 01110011₂
MOV P3, A            ;Выдать на все восемь выводов P3 содержимое ACC
```

2. ANL (логическое 'И'), например:

```
ANL P1, #11110011b   ;выдать низкий потенциал на выводах P1.2 и P1.3
```

3. XRL (исключающее 'ИЛИ'), например:

```
XRL P3, #01000100b   ;инвертировать состояние выводов P3.2 и P3.6
```

4. ORL (логическое 'ИЛИ'), например:

```
ORL P1, #00001100b   ;выдать высокий потенциал на выводах P1.2 и P1.3
```

Эти команды изменяют потенциал сразу на нескольких выводах порта. Для изменения потенциалов только на одном выводе микросхемы, можно воспользоваться следующими командами с битовой адресацией:

1. MOV (пересылка), например:

```
MOV P1.2, C   ;выдать содержимое бита переноса через вывод 2 порта P2
```

2. CPL (инверсия), например:

```
CPL P1.2      ;проинвертировать 2-й бит порта P2
```

3. SETB (установить бит), например:

```
SETB P2.3     ;выдать высокий потенциал на вывод 3 порта P2
```

4. CLR (сбросить бит), например:

```
CLR P2.3      ;выдать низкий потенциал на вывод 3 порта P2
```

При записи в разряд порта логического 0 выходной транзистор открывается и на выводе микросхемы появляется низкий потенциал, изменить который извне невозможно. Поэтому при опросе этого вывода порта микросхемы входная информация в этом случае всегда будет восприниматься как логический 0 независимо от состояния выходов внешних устройств. Если в указанный разряд записать логическую 1, то выходной транзистор закрывается и на выводе микросхемы за счет генератора тока появляется высокий потенциал. Он может быть изменен извне на нулевой потенциал замыканием соответствующего вывода микросхемы на общий провод. В этом случае считываемое микро-

контроллером значение бита будет соответствовать сигналу на выходе внешнего устройства. Поэтому, перед тем как осуществить ввод информации по какому-либо выводу порта, соответствующий разряд необходимо *настроить на ввод* — записать в него логическую единицу.

По той же причине *при настройке выводов порта на выполнение альтернативных функций* в соответствующие разряды параллельного порта должны быть записаны логические 1.

Кроме работы в качестве обычных портов ввода-вывода, внешние выводы портов P0–P3 могут выполнять ряд дополнительных (альтернативных) функций.

Порт P0 может быть использован для организации части шины адреса и шины данных при работе микроконтроллера с внешней памятью данных или программ. При этом через него из микроконтроллера выводится младший байт адреса A0-A7, а также принимается в микроконтроллер или выдается из него байт данных. Во время чтения содержимого внешней памяти во все триггеры-защелки порта P0 аппаратно записываются 1 (т. е. содержимое порта теряется). Кроме того, через порт P0 передаются данные при программировании внутреннего ППЗУ, и читается содержимое внутренней памяти программ при работе с программатором.

При сбросе микросхемы во все разряды порта P0 записываются 1. У схемы P0, в отличие от схем всех других портов, отсутствует внутренний генератор тока. Поэтому при работе с этим портом приходится подключать внешние резисторы к плюсу источника питания.

Формат и адрес порта P0 приведены на рис. 20.5. На этом же рисунке приведены адреса отдельных битов порта P0 в битовом пространстве. На рис. 20.6 приведена схема использования выводов портов P0 и P2 для подключения внешней памяти программ и внешней памяти данных.

Порт P1 может быть использован для чтения внутренней памяти программ или для передачи младшего байта адреса при программировании внутреннего РПЗУ. В младших моделях микроконтроллера семейства других альтернативных функций у порта P1 нет. *При сбросе микросхемы во все разряды порта записываются 1.*

Адрес порта	Ст. зн. разр.							Мл. зн. разр.	
080h	A/D7 P0.7	A/D6 P0.6	A/D5 P0.5	A/D4 P0.4	A/D3 P0.3	A/D2 P0.2	A/D1 P0.1	A/D0 P0.0	P0
	87h	86h	85h	84h	83h	82h	81h	80h	

Рис. 20.5. Формат параллельного порта P0

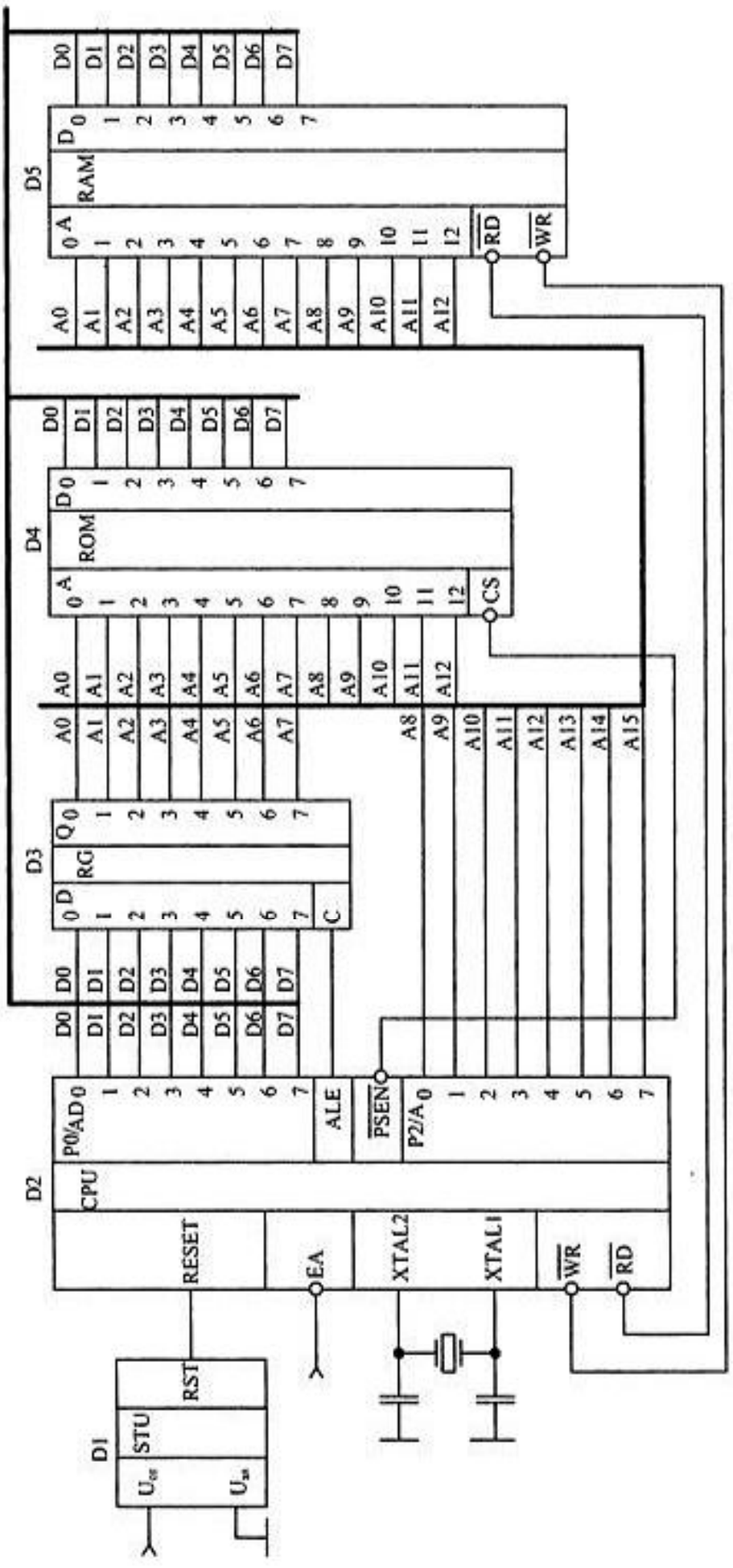
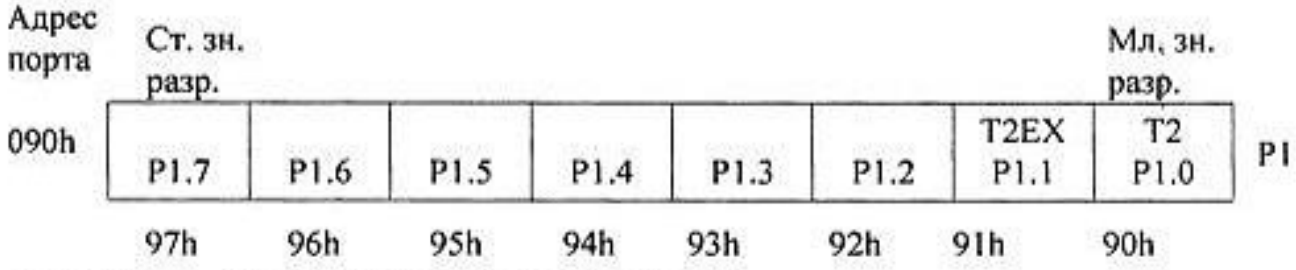


Рис. 20.6. Использование выводов портов P0 и P2 для подключения внешней памяти программ и внешней памяти данных

Формат и адрес порта P1 приведены на рис. 20.7. На этом же рисунке приведены адреса отдельных битов порта P1 в битовом пространстве. Заметим, что этот порт в самых младших моделях микроконтроллеров альтернативных функций не имеет, а приведенные альтернативные функции относятся к микроконтроллерам с ядром MCS-52. Линии порта P1 могут выполнять альтернативные функции только в том случае, если в соответствующие этим линиям разряды регистра записаны логические 1, иначе на линиях порта будет присутствовать логический 0 независимо от характера принимаемой или передаваемой информации.



P1.0 T2 - Внешний вход таймера/счетчика 2

P1.1 T2EX - Вход управления перезагрузки/захвата таймера

Рис. 20.7. Формат параллельного порта P1

Порт P2 может быть использован для передачи старшего байта адреса при программировании внутреннего ППЗУ и при чтении внутренней памяти программ. Через порт P2 выводится старший байт адреса A8—A15 при работе с внешней памятью программ и внешней памятью данных (с 16-разрядным адресом). Во время доступа к внешней памяти содержимое регистра-защелки порта P2 не изменяется. *При сбросе микросхемы во все разряды порта записываются 1.* Формат и адрес порта P2 приведены на рис. 20.8. На рис. 20.6 представлена схема использования порта P2 при подключении внешней памяти программ и внешней памяти данных.

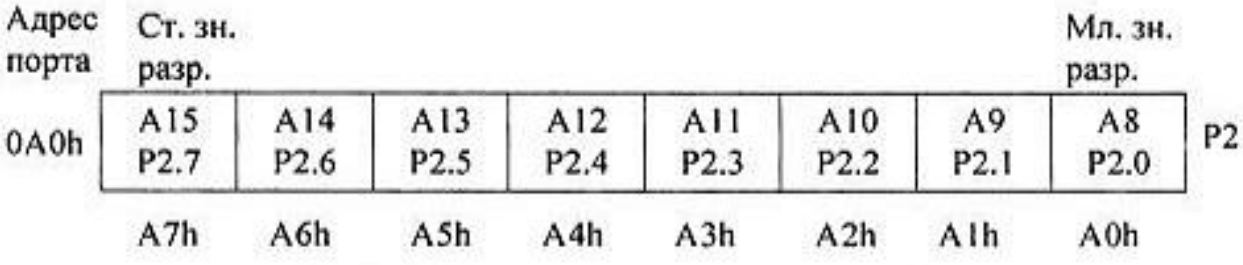


Рис. 20.8. Формат параллельного порта P2

Порт P3. Каждая линия порта P3 имеет индивидуальную альтернативную функцию, которая может быть задействована простым обращением к устройству, соединенному с выводом порта. Например, для того чтобы был вы-

работан строб WR, достаточно обратиться с внешней памяти командой `MOVX @DPTR, A` или `MOVX @R0, A`. Линии порта P3 могут выполнять альтернативные функции только в том случае, если в соответствующие этим линиям разряды регистра записаны логические 1, иначе на линиях порта будет присутствовать логический 0 независимо от характера принимаемой или передаваемой информации. *При сбросе микросхемы во все разряды порта записываются 1.* Формат и адрес порта P3 приведены на рис. 20.9.

Адрес порта	Ст. зн. разр.							Мл. зн. разр.	
0B0h	RD P3.7	WR P3.6	T1 P3.5	T0 P3.4	INT1 P3.3	INT0 P3.2	TxD P3.1	RxD P3.0	P1
	B7h	B6h	B5h	B4h	B3h	B2h	B1h	B0h	

P3.0 RxD – вход последовательного порта;
P3.1 TxD – выход последовательного порта;
P3.2 INT0 – вход 0 внешнего запроса прерываний;
P3.3 INT1 – вход 1 внешнего запроса прерываний;
P3.4 T0 – вход счетчика внешних событий 0;
P3.5 T1 – вход счетчика внешних событий 1;
P3.6 WR – строб записи во внешнюю память данных;
P3.7 RD – строб чтения из внешней памяти данных.

Рис. 20.9. Формат параллельного порта P3

Особенности построения памяти микроконтроллеров семейства MCS-51

Микроконтроллеры семейства MCS-51 построены по гарвардской архитектуре. Это означает, что память данных и память программ в этих микросхемах разделены и имеют отдельные адресные пространства. В этих микроконтроллерах имеется три адресных пространства: памяти программ, внешней памяти данных и внутренней памяти. Такое построение памяти позволяет удвоить доступное адресное пространство внешней памяти. Кроме того, подобное построение по гарвардской структуре позволяет в ряде случаев увеличить быстродействие микросхем. Для микроконтроллеров особенно важным является то обстоятельство, что не существует команд, способных осуществить запись данных в память программ и тем самым испортить управляющую программу.

Схема подключения внешних микросхем памяти к микроконтроллерам семейства MCS-51 приведена на рис. 20.6. Регистр адреса D3 на этой схеме предназначен для запоминания младших восьми битов адреса, передаваемых

через шину данных/адреса, совмещенную с портом P0. Старшие восемь битов адреса передаются через шину адреса, совмещенную с портом P2. Во время передачи адреса через порт P0 микроконтроллер вырабатывает синхроимпульс на выводе ALE. Именно этот импульс позволяет запомнить младший байт адреса в регистре D3 и тем самым организовать 16-разрядную шину адреса внешней памяти.

При обращении к памяти данных и к памяти программ используются одни и те же шина адреса и шина данных, но разные управляющие сигналы. Для чтения памяти программ вырабатывается сигнал PSEN, а для чтения памяти данных — сигнал RD. Для записи информации в память данных вырабатывается сигнал WR. Именно эти управляющие сигналы и разделяют память микроконтроллера на память программ и память данных. То есть память программ доступна только для чтения, а память данных доступна и для чтения, и для записи любой информации, записанной в двоичном коде.

Память программ микроконтроллеров MCS-51

Память программ предназначена для хранения программ и имеет отдельное от памяти данных адресное пространство объемом 64 Кбайт, причем у некоторых микросхем (например, КР1816ВЕ51, КМ1819ВЕ751, КР1830ВЕ51) для хранения управляющих программ на кристалле микроконтроллера расположено ПЗУ. Это ПЗУ отображается в область младших адресов памяти программ. Поскольку выполнение программы после сброса микроконтроллера всегда начинается с нулевого адреса памяти программ, при включении питания начнет выполняться программа, записанная во внутреннем ПЗУ микроконтроллера.

Микроконтроллеры, не имеющие внутреннего ПЗУ (например, КР1816ВЕ31 и КР1830ВЕ31), могут работать только с внешней микросхемой ПЗУ емкостью до 64 Кбайт (при использовании портов P1 и P3 в качестве расширителя адреса объем подключаемой ПЗУ может быть увеличен до 1 Гбайт).

Микроконтроллеры семейства MCS-51 имеют внешний вывод EA, с помощью которого можно запретить работу внутренней памяти, для чего необходимо подать на вывод EA логический 0 (соединить этот вывод с общим проводом). При этом внутренняя память программ отключается и, начиная с нулевого адреса, все обращения происходят к внешней памяти программ.

Доступ к внешней памяти программ осуществляется в двух случаях:

1. При действии сигнала EA=0 независимо от адреса обращения.
2. В любом случае, если программный счетчик (PC) содержит число большее, чем максимальный адрес внутренней памяти программ.

Распределение памяти программ микроконтроллера КР1830ВЕ51 представлено на рис. 20.10. На этом рисунке объем внутренней памяти приведен для микроконтроллеров 1816ВЕ751. У других микросхем этого семейства он может оказаться иным, и соответственно будет иначе проходить граница между внутренней и внешней памятью программ. Количество доступных векторов прерываний тоже зависит от конкретного типа микроконтроллера.

Рис. 20.10. Распределение памяти программ микроконтроллеров КР1830ВЕ751

Адреса векторов прерываний и соответствующие им аппаратные источники прерываний для микроконтроллеров AT89c52 приведены в табл. 20.3. Что такое векторы прерываний и принципы работы с этими особыми ячейками памяти программ будут рассмотрены позднее.

***Таблица 20.3.** Адреса векторов прерываний*

Адрес вектора прерывания	Флаги, вызывающие прерывание	Источник прерывания
0000H	—	Рестарт (сброс) контроллера RESET
0003H	IE0	Внешнее прерывание INT0
000BH	TF0	Таймер 0
0013H	IE1	Внешнее прерывание INT1
001BH	TF1	Таймер 1
0023H	RI, TI	Последовательный порт
002BH	TF2, EXF2	Таймер 2

Сейчас разработано огромное количество микросхем, принадлежащих к этому семейству, у многих из них есть дополнительные векторы прерываний. Для получения сведений об этих векторах прерываний необходимо обратиться к техническим описаниям соответствующих микроконтроллеров. Принципы работы с основными и с дополнительными векторами прерываний ничем не отличаются.

Так как для хранения программы используется ПЗУ, то в этом же запоминающем устройстве имеет смысл хранить и данные, которые не будут изменяться при выполнении программы. Это такие данные, как таблицы элементарных функций, например, синус, таблицы перекодировки данных, например, таблица семисегментных кодов. Для чтения подобных данных из памяти программ используются команды, обозначаемые мнемоническим обозначением MOVC. Примеры команд, осуществляющих чтение памяти программ, приведены в листинге 20.16.

Листинг 20.16. Примеры команд чтения памяти программ

```
MOVC A, A+@DPTR ;Считать байт из памяти программ по адресу, вычисляемому
                ;как сумма содержимого регистров аккумулятора и DPTR
MOVC A, A+@PC   ;Считать байт из области памяти программ, начинающейся за
                ;данной командой
```

Внешняя память данных микроконтроллеров MCS-51

Внешняя память данных предназначена для временного хранения информации, используемой в процессе выполнения программы. Эта память физически должна быть подключена к микроконтроллеру при помощи схемы, изображенной на рис. 20.6. Максимальный объем этой памяти определяется разрядностью регистра DPTR и составляет 64 Кбайт. Адресное пространство внешней памяти данных приведено на рис. 20.11. Точно так же, как и в случае внешней памяти программ, объем доступной внешней памяти данных может быть увеличен за счет использования портов P1 и P3 до 1 Гбайт.

Внешняя память данных для своей работы требует использование портов P0, P2 и P3. Это приводит к увеличению габаритов устройства, росту уровня помех и, в конечном итоге, удорожанию устройства в целом. Поэтому в современных устройствах внешняя память данных обычно не используется.

Однако в некоторых микроконтроллерах (таких как 87c550 фирмы Dallas или ADuC834 фирмы Analog Devices) команды обращения к внешней памяти используются для работы с дополнительной внутренней памятью данных (ОЗУ) большого объема.

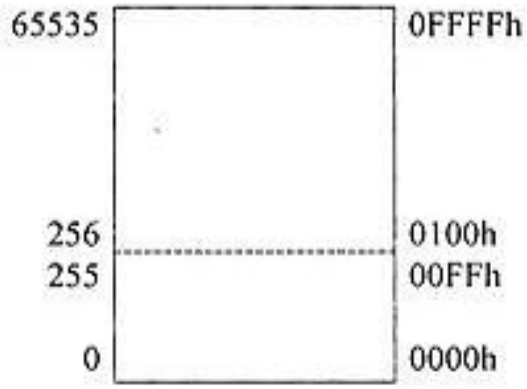


Рис. 20.11. Адресное пространство внешней памяти данных

Для обращения к внешней памяти данных, как записи, так и чтения, служат команды, использующие 16-разрядный регистр адреса DPTR:

```
MOVX A, @DPTR ;команда чтения
MOVX @DPTR, A ;команда записи
```

Иногда для того, чтобы сохранить P2 в качестве параллельного порта, для обращения к внешней памяти данных применяют команды, использующие 8-разрядные регистры адреса R0 или R1, как это приведено, например, в листинге 20.17.

Листинг 20.17. Примеры команд чтения внешней памяти данных

```
;команды чтения
MOVX A, @R0
MOVX A, @R1
;команды записи
MOVX @R0, A
MOVX @R1, A
```

В этом случае адресное пространство внешней памяти данных уменьшается до размеров страницы 256 байт. На рис. 20.11 показан вариант, когда эта страница расположена в области младших адресов.

Отметим, что в качестве внешней памяти данных могут быть использованы как микросхемы ОЗУ, так и микросхемы ПЗУ. В последнем случае соответствующая область памяти данных будет доступна только для чтения. Это необходимо учитывать при написании программы микроконтроллера.

Внутренняя память данных микроконтроллеров MCS-51

Несмотря на то, что внутренняя память данных имеет самое маленькое адресное пространство из рассматриваемых видов памяти, оно устроено наи-

более сложным образом. Распределение внутренней памяти данных микроконтроллеров серии MCS-51 приведено на рис. 20.12.

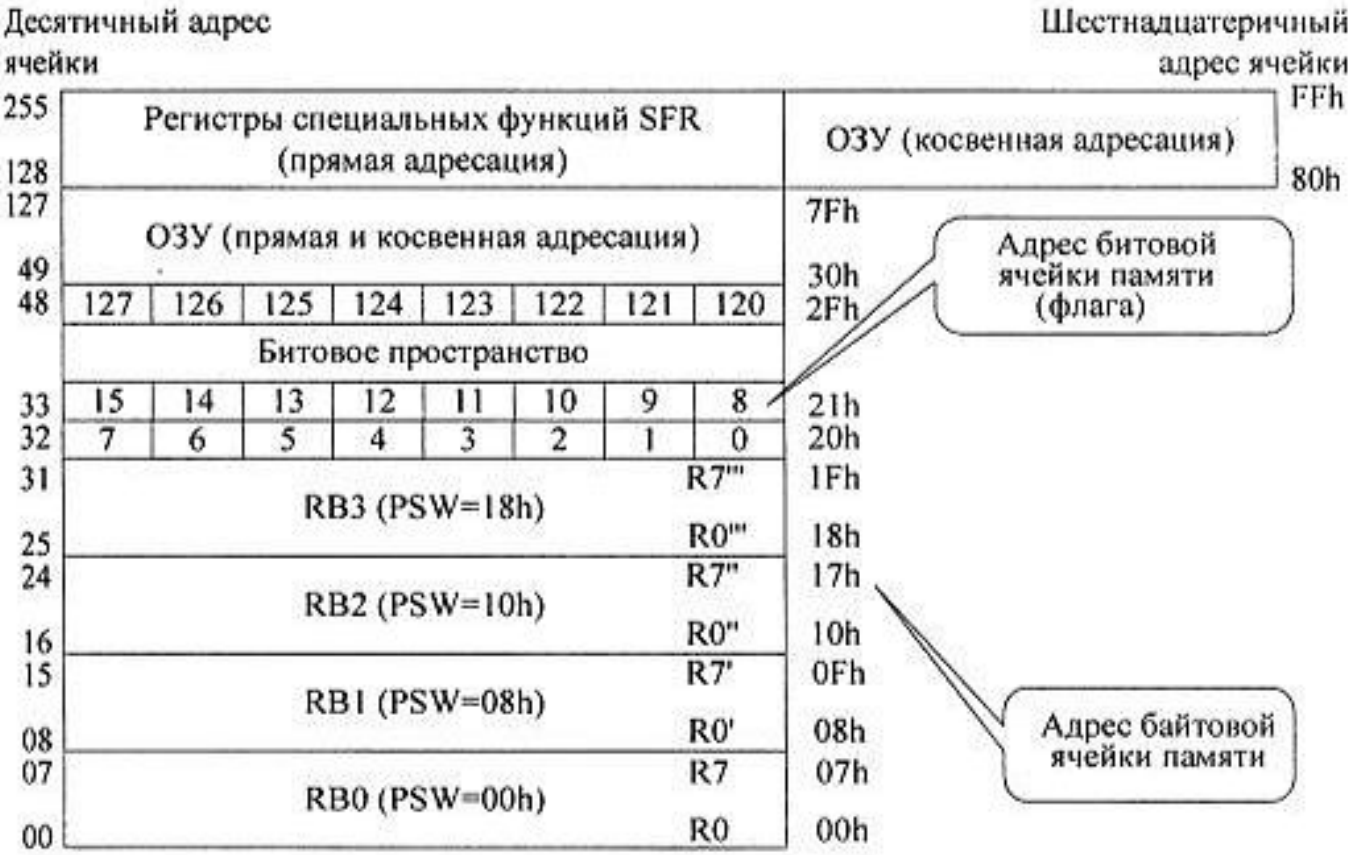


Рис. 20.12. Адресное пространство внутренней памяти данных

Внутреннее ОЗУ данных предназначено для временного хранения информации, используемой в процессе выполнения программы, и занимает 128 младших байтов, с адресами от 000h до 07Fh для микроконтроллеров 8051, 8031, КР1816ВЕ31, КР1816ВЕ51, КР1816ВЕ751 КР1830ВЕ31, КР1830ВЕ51, КР1830ВЕ751. Для всех остальных микроконтроллеров семейства MCS-51 внутреннее ОЗУ данных занимает 256 8-разрядных ячеек, с адресами от 000h до 0FFh.

Регистры специальных функций занимают адреса внутренней памяти данных с 080h по 0FFh. Так как адреса регистров специальных функций совпадают со старшими адресами внутреннего ОЗУ данных, то имеются особенности при обращении к ячейкам ОЗУ и к регистрам специальных функций.

Система команд микроконтроллера позволяет обращаться к ячейкам внутренней памяти данных при помощи прямой и косвенно-регистровой адресации. Обращение к ячейкам памяти с адресами 0 ... 127 (07Fh) может производиться с использованием любого из этих видов адресации. При этом команда будет производить выборку одной и той же ячейки памяти.

При обращении к ячейкам ОЗУ с адресами 128(080h)-256(0FFh) следует воспользоваться косвенно-регистровой адресацией. Учитывая, что работа со стеком ведется при помощи косвенной адресации, а сам стек растет вверх, то имеет смысл размещать в этой области внутренней памяти данных стек. Само понятие стека и особенности его использования будут рассмотрены позднее. Если же требуется обратиться к регистрам специальных функций, то нужно использовать прямую адресацию. Примеры команд обращения к ОЗУ и регистрам специального назначения приведены в листинге 20.18.

Листинг 20.18. Примеры команд обращения к ОЗУ и регистрам специального назначения

```
MOV A, 80h    ;Скопировать сигналы с внешних выводов порта P0 в
              ;аккумулятор

MOV R0, #80h ;Скопировать в аккумулятор содержимое
MOV A, @R0   ;ячейки внутреннего ОЗУ  с адресом 80h
```

Регистры общего назначения позволяют писать самые эффективные программы. У микроконтроллеров семейства MCS-51 для программиста доступны восемь регистров. Более того, в этом семействе микроконтроллеров есть четыре набора (банка) регистров с именами RB0 ... RB3. Для сравнения, такие мощные процессоры, как "электроника-79" и SPARC, обладают всего двумя наборами регистров. Банк регистров состоит из восьми 8-разрядных регистров с именами R0, R1, ..., R7. Несколько банков регистров служат для организации независимой работы нескольких параллельно выполняемых программных потоков. Переключение банков регистров производится при помощи двух особых битов, входящих в регистр слова состояния программы PSW (биты RS0 и RS1). Если организация нескольких параллельных потоков обработки данных не нужна, то можно пользоваться только нулевым банком регистров, включающимся автоматически после подачи питания и сброса микроконтроллера, а остальные ячейки памяти использовать как обычное ОЗУ.

Все четыре банка регистров объединены с 32 младшими байтами внутреннего ОЗУ данных (см. рис. 20.12). Так как физически регистры и ячейки внутреннего ОЗУ объединены, то команды программы могут обращаться к регистрам используя их имена R0-R7 (регистровая адресация):

```
MOV A, R0   ;Скопировать содержимое регистра R0 в аккумулятор
MOV R7, A   ;Скопировать содержимое регистра R7 в аккумулятор
```

или их адрес во внутренней памяти данных (прямая байтовая адресация):

```
MOV A, 0    ;Скопировать содержимое нулевой ячейки ОЗУ в аккумулятор
MOV 7, A    ;Скопировать содержимое аккумулятора в седьмую ячейку ОЗУ
```

Следующие после банков регистров 16 ячеек внутреннего ОЗУ данных (адреса 20H-2FH) образуют область памяти, к которой возможна как байтовая, так и битовая адресация. В этих ячейках располагаются 128 программных флагов (битовых ячеек памяти). Обращение к отдельным битам этих ячеек возможно по их битовым адресам. Например, команды:

```
SETB 15         ;Запомнить во флаге 15 логическую единицу
JB 15, Metka ;Если во флаге 15 записана логическая единица, то перейти
             ;по адресу Metka
```

обращаются к флагу 15, расположенному в старшем бите байтовой ячейки памяти 21h. Использование однобитовых ячеек памяти позволяет сократить необходимый для работы программы объем памяти данных, т. к. для хранения битовых переменных выделяется один бит в памяти данных, а не машинное слово, как это делается в универсальных микропроцессорах (компьютерах).

В битовой области внутренней памяти данных сосредоточено только 128 флагов. Битовая адресация возможна также в области регистров специальных функций (см. табл. 20.4). Наиболее яркий пример использования битовой адресации в данной области — это обращение к отдельным выводам параллельных портов:

```
CPL 92          ;Проинвертировать второй бит порта P1
```

Выше адреса 128 (080h) располагаются регистры специальных функций, которые будут рассмотрены позже. Некоторые из регистров специальных функций допускают битовую адресацию к каждому из восьми своих битов. Оставшаяся область внутренней памяти данных (ячейки памяти с адресами от 49 до 127) используется как обычное ОЗУ, без особенностей.

Следует отметить, что в современных микроконтроллерах данного семейства эту память необходимо рассматривать как встроенные 256 регистров сверхоперативного ОЗУ. Основной памятью постепенно становится внутренняя память микроконтроллера, доступная при помощи команд `MOVX @DPTR, A`.

Регистры специальных функций

Адреса внутренней памяти данных с 080h по 0FFh используются регистрами специальных функций. Они принадлежат дополнительным устройствам, расположенным на кристалле микроконтроллера, регистры которых отображаются в адресное пространство внутренней памяти данных. В различных микросхемах семейства MCS-51 состав дополнительных устройств различается. Микроконтроллеры рассматриваемого семейства различаются между собой количеством параллельных портов, последовательных портов, таймеров. Некоторые из регистров специальных функций с указанием их адресов в адресном пространстве SFR внутреннего ОЗУ приведены в табл. 20.4.

Таблица 20.4. *Адреса регистров специальных функций во внутренней памяти данных и их значения после сброса*

F8									FF
F0	**B** 00000000								F7
E8									EF
E0	**ACC** 00000000								E7
D8									DF
D0	**PSW** 00000000								D7
C8	**T2CON** 00000000	**T2MOD** 00000000	**RCAP2L** 00000000	**RCAP2H** 00000000	**TL2** 00000000	**TH2** 00000000			CF
C0									C7
B8	**IP** X0000000	**SADEN** 00000000							BF
B0	**P3** 11111111								B7
A8	**IE** 00000000	**SADDR** 00000000							AF
A0	**P2** 11111111								A8
98	**SCON** 00000000	**SBUF** XXXXXXXX							9F
90	**P1** 11111111								97
88	**TCON** 00000000	**TMOD** 00000000	**TL1** 00000000	**TL1** 00000000	**TH0** 00000000	**TH1** 00000000			8F
80	**P0** 11111111	**SP** 00000111	**DPL** 00000000	**DPH** 00000000				**PCON** 00X10000	87
	Битовая адресация								

Примечание.

1. Регистры, выделенные **жирным подчеркнутым текстом**, присутствуют во всех микросхемах семейства.
2. Регистры, выделенные **жирным текстом**, присутствуют в микросхемах с ядром 8052.
3. *X* — неопределенное состояние.

Внутренние таймеры микроконтроллера, особенности их применения

В базовых моделях семейства (ядро MCS-51) имеются два программируемых 16-битных таймера/счетчика (T/C0 и T/C1), которые могут быть использованы и как таймеры, и как счетчики внешних событий. Каждый из них состоит из двух 8-битных регистров: TH0 (старший байт) и TH0 (младший байт) для таймера 0 или TH1 (старший байт) и TH1 (младший байт) для таймера 1. При переполнении таймеров производится запись логической единицы в дополнительный триггер (флаг) TF0 для таймера 0 или TF1 для таймера 1.

В старших моделях рассматриваемого семейства микроконтроллеров появляется еще один, причем более удобный, таймер T2. Но рассмотрение принципов работы этого таймера не входит в задачу данной книги.

В режиме таймера содержимое соответствующего таймера/счетчика инкрементируется в каждом машинном цикле, т. е. через каждые 12 периодов колебаний кварцевого резонатора.

Таймер 0 и таймер 1 могут работать в четырех режимах:

- режим 0: 13-битный таймер;
- режим 1: 16-битный таймер;
- режим 2: 8-битный таймер с автоматической перезагрузкой;
- режим 3: Таймер 0 как 2 раздельных 8-битных таймера.

В режиме счетчика содержимое соответствующего таймера/счетчика инкрементируется (увеличивается на единицу) под воздействием перехода из 1 в 0 внешнего входного сигнала, подаваемого на вывод микроконтроллера T0 или T1. Так как на распознавание периода требуются два машинных цикла, то максимальная частота подсчета входных сигналов равна 1/24 частоты резонатора. На максимальную длительность периода входных сигналов ограничений нет. Для гарантированного обнаружения перехода уровень входного сигнала не должен изменяться как минимум в течение одного машинного цикла микроЭВМ.

Кроме того, таймер 1 можно использовать для задания скорости обмена последовательного порта, работающего в режиме с настраиваемой скоростью работы.

Для управления режимами работы таймеров используется регистр TMOD (Timer — таймер, MODe — режим). Его формат приведен на рис. 20.13. Каждая тетрада регистра TMOD управляет своим таймером. Рассмотрим режимы работы внутренних таймеров более подробно.

Адрес ст.зн. разр. Значение после сброса = 0000 0000B мл.зн. разр.

89H	GATE	C/$\overline{T}$	M1	M2	GATE	C/$\overline{T}$	M1	M0	TMOD
	Таймер 1				Таймер 0				

Бит	Назначение
GATE	Управление блокировкой. Когда установлен, таймер/счетчик 0 или 1 разрешен только при высоком уровне на выводе $\overline{INT0}$ или $\overline{INT0}$ высокий уровень и установленном бите управления TR0 или TR1 соответственно. Когда сброшен, таймер 0 или 1 разрешен только при установленном бите управления TR0 или TR1 соответственно
C/$\overline{T}$	Бит выбора режима таймера или счетчика. Режим таймера включается при C/$\overline{T}$=0 (счет внутренних импульсов синхронизации). Режим счетчика

M1	M0	Режим работы
0	0	8-битный таймер/счетчик (THx), TLx используется как 5-битный предварительный делитель
0	1	16-битный таймер/счетчик
1	0	8-битный таймер/счетчик с автоматической перезагрузкой. THx содержит значение, которое загружается в TLx при каждом переполнении
1	1	(Таймер 0). TL0 как 8-битный таймер/счетчик, управляемый битами управления таймера 0. TH0 только как 8-битный таймер, управляемый битами управления таймера 1.
1	1	(Таймер 1). Таймер/счетчик останавливается.

Рис. 20.13. Формат регистра выбора режимов таймеров (TMOD)

Режим 0

В режиме 0 таймер работает как 13-битный счетчик, состоящий из 8 битов регистра THx и младших 5 битов регистра TLx. Заметим, что x в обозначении регистра заменяется на 0 или 1 в зависимости от номера таймера, которым мы управляем в данный момент. Состояние старших 3 битов регистров TLx в режиме 0 не определены и они игнорируются. Установка запускающего таймер флага TR0 или TR1 не очищает эти регистры. Работе таймера 0 или таймера 1 в режиме 0 соответствует схема, приведенная для T0 на рис. 20.14. Флаг прерывания таймера TFx устанавливается (принимает значение логической 1) при изменении содержимого счетчика из состояния все 1 в состояние все 0, т. е. при переполнении.

Этот режим был введен для совместимости с устаревшим семейством микроконтроллеров MCS-48, чтобы облегчить перенос уже разработанных программ на новые процессоры, и поэтому в настоящее время не используется. Тем не менее, в этом режиме можно обеспечить формирование одиночного

интервала времени длительностью до 8096 мс при частоте задающего генератора 12 МГц.

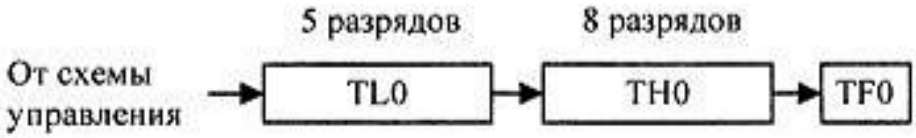


Рис. 20.14. Схема таймера T0, работающего в режиме 0

Обычно пользователя интересует не максимальный интервал времени, а некоторый конкретный временной промежуток. Для уменьшения интервала времени в регистры таймера можно предварительно занести число и тем самым сформировать промежуток времени нужной длительности. Рассмотрим пример подготовки таймера T0 для формирования временного интервала 5 мс, приведенный в листинге 20.19.

Листинг 20.19. Подготовка таймера T0 для формирования временного интервала 5 мс

```
;Настроить режим работы таймера-----------------------------------------
  MOV TMOD,#00000000b ;настроить таймеры T0 и T1 на нулевой режим работы
;            ||||||||
;            ||||||++-----Перевести таймер T0 в 13-разрядный режим
;            ||||||       работы
;            |||||+------Синхронизироваться от внутреннего генератора
;            ||||+-------Запретить управление таймером от ножки INT0
;            ||++--------Перевести таймер T1 в 13-разрядный режим
;            ||           работы
;            |+----------Синхронизироваться от внутреннего генератора
;            +-----------Запретить управление таймером от вывода INT1

;Настроить таймер на генерацию 5-миллисекундного интервала времени----
  MOV TH0, #HIGH(-5000) ;Загрузить старший байт таймера
  MOV TL0, #LOW(-5000)  ;Загрузить младший байт таймера
```

В рассмотренном исходном тексте программы используется двоичное представление управляющей константы. Это позволяет показать каждый отдельный бит константы, тем более что разные биты управляют различными узлами таймера. Для того чтобы программа была более понятной, в комментарии поясняется назначение отдельных битов константы. В исходном тексте программы, особенно при написании его под Windows, невозможно использовать символы псевдографики, поэтому для указания к какому же из битов константы относится комментарий используются символы '–' и '|'. Для перехода

от горизонтальной черты к вертикальной используется символ '+'. В случае, когда для управления блоком таймера требуется несколько битов константы, в одной строке может быть использовано несколько символов '+'.

При настройке таймера требуется загрузить 16-битную константу в счетчик таймера. Однако в системе команд микроконтроллера существуют только команды загрузки 8-битной константы. Для расщепления 16-битной константы на два отдельных байта в приведенном участке программы были использованы функции выделения старшего и младшего байта HIGH и LOW соответственно. Эти функции присутствуют в большинстве языков программирования ассемблер, предназначенных для микроконтроллеров MCS-51. Если же язык программирования не содержит в своем составе подобные функции, то можно для выделения байтов воспользоваться операцией деления на 256, как это показано в листинге 20.20.

Листинг 20.20. Использование операции деления для выделения старшего и младшего байта числа

```
;Настроить таймер на генерацию 5-миллисекундного интервала времени----
    MOV TH0, #-5000/256          ;Загрузить старший байт таймера
    MOV TL0, #-(5000-5000/256) ;Загрузить младший байт таймера
```

Режим 1

В режиме 1 таймер работает как шестнадцатиразрядный счетчик. Этот режим похож на режим 0, за исключением того, что в регистрах таймера использует все 16 битов. В этом режиме регистры THx и TLx также включены последовательно друг за другом. Работе таймера T0 или таймера T1 в режиме 1 соответствует схема, приведенная на рис. 20.15. На этой схеме изображен таймер T0.

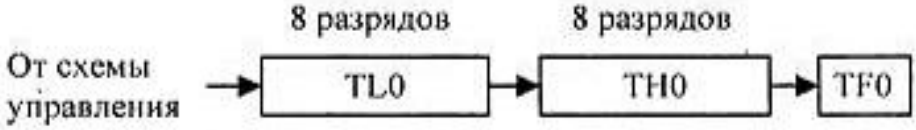


Рис. 20.15. Схема таймера T0, работающего в режиме 1

В этом режиме можно обеспечить формирование интервала времени длительностью до 65 536 мкс при частоте задающего генератора 12 МГц.

Рассмотрим пример использования таймера T0 для формирования временного интервала 15 мс, приведенный в листинге 20.21.

Листинг 20.21. Подготовка таймера T0 для формирования временного интервала 15 мс

```
;Настроить режим работы таймера-------------------------------------
  MOV TMOD,#00000001b ;перевести таймер T0 в режим 1, а T1 - в режим 0
;            ||||||||
;            ||||||++---Перевести таймер T0 в 16-разрядный режим
;            |||||+-----Синхронизироваться от внутреннего генератора
;            ||||+------Запретить управление таймером от вывода INT0
;            ||++-------Перевести таймер T1 в 13-разрядный режим
;            |+---------Синхронизироваться от внутреннего генератора
;            +----------Запретить управление таймером от вывода INT1

;Настроить таймер на генерацию 15-миллисекундного интервала времени------
  MOV TH0, #HIGH(-15000) ;Загрузить старший байт таймера
  MOV TL0, #LOW(-15000)  ;Загрузить младший байт таймера

OjidanTimer:
    JNB TF0, OjidanTimer ;Подождать, пока не переполнится таймер
```

В рассмотренном примере переполнение таймера произойдет через 15 000 циклов процессора, т. е. при частоте тактового генератора микроконтроллера, равной 12 МГц, ровно через 15 мс. Программа будет постоянно проверять состояние флага переполнения таймера и, как только он установится в единицу, перейдет к выполнению следующей команды. Это не самый лучший вариант использования таймера, но для иллюстрации его настройки для работы в режиме 1 вполне подходит.

Режимы 0 и 1 таймеров T0 и T1 предназначены для формирования одиночного интервала времени. Если возникает необходимость формировать периодическую последовательность интервалов времени, то следует обеспечить программную загрузку регистров TH0 и TL0 для задания нужного интервала времени, что для коротких интервалов времени может привести к значительным затратам процессорного времени.

Для формирования последовательности одинаковых интервалов времени используется режим работы таймера с перезагрузкой — режим 2.

Режим 2

В режимах 0 и 1 таймер позволяет сформировать одиночный временной интервал. В ряде случаев требуется формировать колебание с постоянным периодом. При этом потребуется постоянная загрузка в таймер первоначального значения. В режиме 2 регистр таймера TL0 работает как 8-битный суммирующий счетчик с автоматической перезагрузкой начального значения из

регистра TH0. Переполнение счетчика TL0 не только устанавливает флаг TF0, но и снова загружает регистр TL0 содержимым TH0. Перезагрузка таймера не изменяет содержимое регистра TH0, а это означает, что в этом режиме будут формироваться события с одинаковым интервалом времени.

Для задания периода временны́х интервалов в регистр TH0 записывается отрицательное число. Работе таймера 0 или таймера 1 в режиме 2 соответствует схема, приведенная на рис. 20.16. При этом первый период колебания будет произвольным, но в большинстве случаев это не важно. Если же первый период колебания важен, то кроме регистра TH0 необходимо программно устанавливать значение регистра TH1.

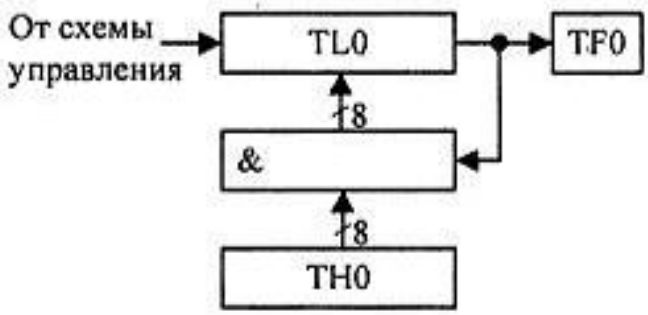


Рис. 20.16. Схема таймера T0, работающего в режиме 2

Подготовка таймеров к работе во втором режиме не отличается от рассмотренных ранее примеров, поэтому следующий пример инициализации таймера для генерации прямоугольного колебания с частотой 10 кГц (период 100 мкс), и скважностью 2, приведенный в листинге 20.22, приводится без дополнительных пояснений.

Листинг 20.22. Подготовка таймера T0 для формирования временного интервала 15 мс

```
;Настроить режим работы таймера-----------------------------------------
  MOV TMOD, #00000010b ;перевести таймер T0 в режим 2, а T1 в режим 0
;              ||||||||
;              ||||||++---Перевести таймер T0 в режим 8-разрядного
;              ||||||     таймера с автозагрузкой
;              |||||+-----Работа от внутреннего генератора
;              ||||+------Запретить управление таймером от вывода INT0
;              ||++-------Перевести таймер T1 в 13-разрядный режим
;              |+---------Работа от внутреннего генератора
;              +----------Запретить управление таймером от вывода INT1

;Настроить таймер на генерацию 50-микросекундного интервала времени------
  MOV TH0, #-50          ;Загрузить старший байт таймера
  MOV TL0, #-50          ;Загрузить младший байт таймера
```

```
OjidanTimer:
    JNB TF0, OjidanTimer  ;Подождать, пока не переполнится таймер
    CPL P2.6              ;Проинвертировать сигнал на выводе 6 порта P2

    SJMP OjidanTimer      ;Снова перейти к ожиданию окончания временного
                          ;интервала
```

Режим 3

Таймер 1 при работе в режиме 3 просто хранит свое значение. Эффект такой же, как при сбросе бита разрешения счета TR1. Таймер 0 в режиме 3 представляет собой два раздельных 8-битных счетчика (регистры TL0 и TH0). Регистр TL0 использует биты управления таймера 0: C/T0, GATE0, TR0 и TF0. Регистр TH0 работает тоже в режиме отдельного таймера и использует биты TR1 и TF1 таймера 1. Логика работы таймера 0 в режиме 3 показана на схеме, приведенной на рис. 20.17.

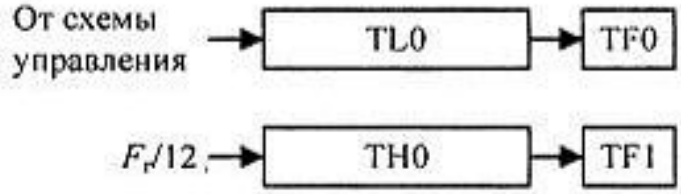


Рис. 20.17. Логика работы таймера 0 в режиме 3

Работа таймера TL0 разрешена, если бит TR0 = 1, а таймера TH0 — если бит TR1 = 1. Таймер 1 при работе таймера 0 в режиме 3 постоянно включен, т. к. он обычно при этом используется для синхронизации последовательного порта и работает в первом или втором режимах работы.

Этот режим работы позволяет реализовать два независимых таймера, если таймер 1 используется для работы последовательного порта, однако следует отметить, что на практике, особенно после появления таймера 2 в ядре MCS-52, режим 3 мало интересен.

Управление таймерами/счетчиками

Схема управления таймерами 0 и 1 идентична и для таймера T0 приведена на рис. 20.18. Для схемы управления таймером T1 изменятся только номера управляющих битов (в их наименованиях 0 будет заменен на 1). В приведенной схеме заштрихованным прямоугольником обозначены внешние выводы микросхемы микроконтроллера.

Из схемы видно, что таймер T0 может включаться и выключаться битом TR0 (для таймера T1 — битом TR1). Таким образом, выключая ненужный в дан-

ный момент таймер, можно уменьшать ток потребления микросхемы и уровень помех, создаваемый ею. Счетчики таймеров переключаются на высокой частоте, поэтому они могут потреблять до половины от общего тока потребления микроконтроллера. Следует отметить, что при включении и после сброса микроконтроллера работа таймеров запрещена.

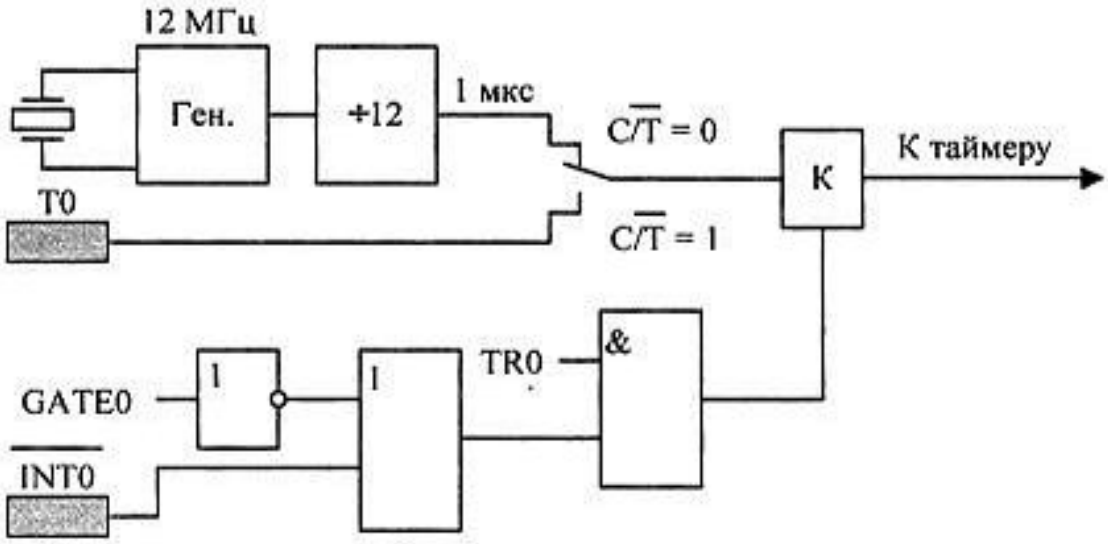


Рис. 20.18. Схема управления таймерами

Есть возможность управлять работой таймера извне при помощи внешнего вывода INT0 — для таймера T0 или INT1 — для таймера T1. Эти выводы совмещены с выводами параллельного порта P3. Чтобы разрешить передачу разрешающего сигнала с вывода INT0, необходимо записать в бит GATE0 логическую единицу (не забыв при этом разрешить работу таймера при помощи бита TR0). То же самое можно проделать и для таймера T1.

Кроме того, таймер может работать от внешнего генератора. Для этого в бит управления C/T нужно записать логическую единицу. При этом максимальная частота внешнего генератора не может превышать 1/24 частоты тактового генератора.

Биты включения таймеров TR0 и TR1 размещены в регистре управления таймерами TCON, а биты GATE и C/T — в регистре режима таймеров TMOD. Название регистра TCON образовалось при соединении двух слов: Timer — таймер и CONtrol — управлять. Формат регистра TCON рассматривался ранее, а формат регистра TCON и описание его отдельных битов приведен на рис. 20.19.

Кроме того, схема управления таймерами интересна тем, что позволяет использовать таймеры в качестве измерителей длительности импульсов и частотомеров. Рассмотрим эту возможность подробнее.

Адрес	ст.зн. разр.							мл.зн. разр.	
88H	TF1 TCON.7	TR1 TCON.6	TF0 TCON.5	TR0 TCON.4	IE1 TCON.3	IT1 TCON.2	IE0 TCON.1	IT0 TCON.0	TCON
	8Fh	8Eh	8Dh	8Ch	8Bh	8Ah	89h	88h	

Наименование	Назначение
TF1	Флаг переполнения таймера 1. Устанавливается аппаратно при переполнении таймера/счетчика 1. Сбрасывается аппаратно, когда процессор вызывает программу обслуживания прерывания
TR1	Бит управления запуском таймера 1. Устанавливается/сбрасывается программно для запуска/останова таймера/счетчика 1
TF0	Флаг переполнения таймера 0. Устанавливается аппаратно при переполнении таймера/счетчика 0. Сбрасывается аппаратно, когда процессор вызывает программу обслуживания прерывания
TR0	Бит управления запуском таймера 0. Устанавливается/сбрасывается программно для запуска/останова таймера/счетчика 0
IE1	Флаг прерывания 1. Устанавливается аппаратно, когда обнаруживается сигнал внешнего прерывания (перепад или активный уровень). Сбрасывается аппаратно в процессе прерывания только в том случае, если оно произошло по фронту
IT1	Бит управления типом прерывания 1. Устанавливается/сбрасывается программно для определения типа прерывания 1 (по срезу/низким уровнем)
IE0	Флаг прерывания 0. Устанавливается аппаратно, когда обнаруживается сигнал внешнего прерывания (перепад или активный уровень). Сбрасывается аппаратно в процессе прерывания только в том случае, если оно произошло по фронту
IT0	Бит управления типом прерывания 0. Устанавливается/сбрасывается программно для определения типа прерывания 0 (по срезу/низким уровнем)

Рис. 20.19. Формат регистра TCON

Использование таймера в качестве измерителя длительности импульсов

Известно, что измерение длительности импульса можно произвести, подсчитав количество импульсов эталонной частоты, поступающих за время длительности измеряемого импульса. Это эквивалентно использованию линейки. Мы прикладываем линейку с эталонными делениями к измеряемой поверхности и подсчитываем количество этих делений. Принцип измерения длительности импульсов иллюстрируется временной диаграммой, приведенной на рис. 6.20. В данном случае в качестве делений служит эталонный период

опорного сигнала. Например, если за время длительности импульса на вход счетчика таймера поступят 15-микросекундных импульсов, то длительность измеренного входного импульса равна 15 мкс.

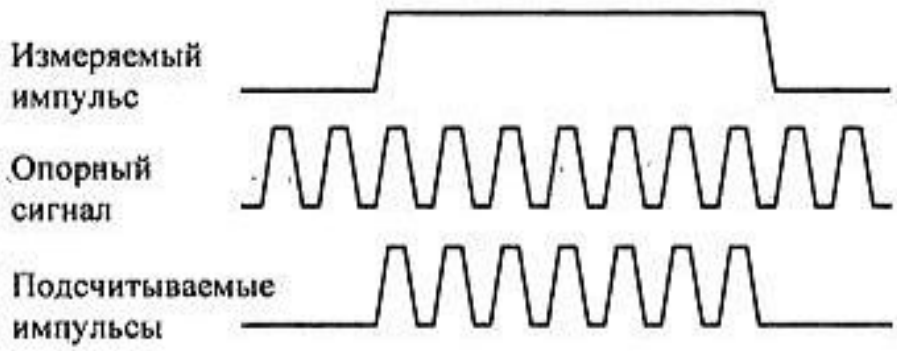


Рис. 20.20. Временная диаграмма измерения длительности импульсов

Для измерения длительности импульса измеряемый сигнал подается на вывод микроконтроллера INT0 или INT1, и в соответствующий бит управления GATE записывается разрешающий сигнал логической единицы. Таймер/счетчик настраивается в режим таймера записью в бит C/T логического нуля. Содержимое таймера перед измерением длительности импульса обнуляется. Исходный текст процедуры измерения длительности импульса приведен в листинге 20.23.

Листинг 20.23. Процедура измерения длительности импульса

```
    MOV TMOD,#00001001b
              ;||||||||
              ;||||||++---Перевести таймер T0 в 16-разрядный режим
              ;|||||+-----Работать от внутреннего генератора
              ;||||+------Включать таймер от вывода микроконтроллера INT0
              ;||++-------Перевести таймер T1 в 13-разрядный режим
              ;|+---------Работать от внутреннего генератора
              ;+----------Запретить управление таймером от вывода INT1

    MOV TH0, #0           ;Обнулить старший байт таймера
    MOV TL0, #0           ;Обнулить младший байт таймера

    SETB  TR0             ;Включить измеритель ширины импульса

TstLog0: JNB INT0, TstLog0 ;Подождать начало импульса
TstLog1: JNB INT0, TstLog1 ;Подождать конец импульса

    CLR  TR0              ;Отключить измеритель ширины импульса
```

Если теперь на вход микроконтроллера INT0 подать импульс с неизвестной длительностью, то в регистрах TH0 и TL0 будет записана его длительность в микросекундах. Следует отметить, что совмещение вывода измерения импульса с прерыванием позволяет одновременно с окончанием измерения сформировать прерывание, обеспечив тем самым минимальное время реакции микроконтроллера на обработку результатов измерения.

Использование таймера в качестве частотомера

Известно, что измерение частоты можно произвести, подсчитав количество периодов неизвестной частоты за единицу времени. Принцип измерения частоты иллюстрируется рис. 20.21. Например, если за 1 мс на вход счетчика таймера поступят 15 импульсов, то измеренная частота равна 15 кГц.

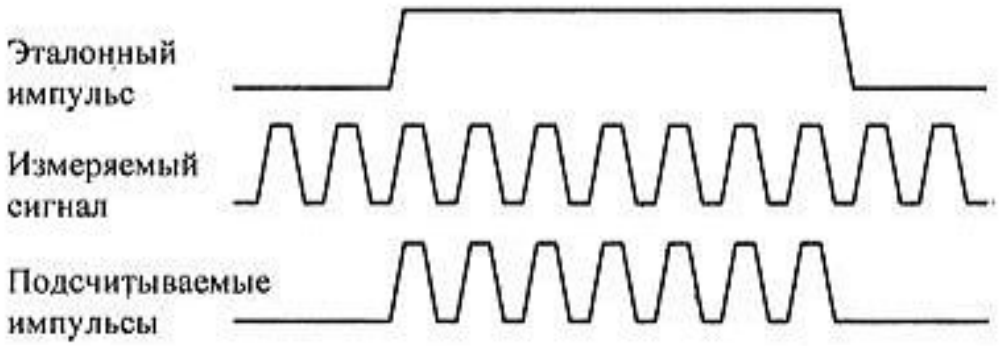


Рис. 20.21. Временная диаграмма измерения частоты

Измеряемый сигнал подается на вывод микроконтроллера T0 или T1. Таймер/счетчик настраивается в режим счетчика записью в бит C/T логической единицы. Содержимое таймера обнуляется. Таймер включается на строго определенный интервал времени. Этот интервал задается другим таймером.

Пример исходного текста программы измерения частоты сигнала на выводе микроконтроллера T0 приведен в листинге 20.24.

Если теперь на вход микроконтроллера T0 подать сигнал с неизвестной частотой, то в регистрах TH0 и TL0 будет записана его частота в килогерцах.

Листинг 20.24. Процедура измерения частоты сигнала

```
MOV TMOD,#00010101b
          ;||||||||
          ;||||||++---Перевести таймер T0 в 16-разрядный режим
          ;|||||+-----Работать от сигнала на выводе T0
          ;||||+------Запретить управление таймером от вывода INT0
          ;||++-------Перевести таймер T1 в 16-разрядный режим
          ;|+---------Работать от внутреннего генератора
          ;+----------Запретить управление таймером от вывода INT1
```

```
  MOV TH0, #0           ;Обнулить старший байт счетчика
  MOV TL0, #0           ;Обнулить младший байт счетчика
;---измерение вести в течение 1 мс-----------------------------------
  MOV TH1, #HIGH(-1000) ;Загрузить старший байт таймера
  MOV TL1, #LOW(-1000)  ;Загрузить младший байт таймера

  MOV TCON,#01010000b  ;Включить частотомер
           ;||||||||
           ;|||||||+---Прерывание от вывода INT1 возникает по фронту
           ;||||||+----Сбросить запрос прерывания от вывода INT1
           ;|||||+-----Прерывание от вывода INT1 возникает по фронту
           ;||||+------Сбросить запрос прерывания от вывода INT1
           ;|||+-------Включить таймер T0
           ;||+--------Обнулить флаг таймера T0
           ;|+---------Включить таймер T1
           ;+----------Обнулить флаг таймера T1

TstTimeOut:
  JNB TF1, TstTimeOut  ;Если 1 мс прошла
         MOV TCON, #00000000b ;то отключить частотомер
            ;||||||||
            ;|||||||+--Прерывание от вывода INT1 возникает по фронту
            ;||||||+---Сбросить запрос прерывания от вывода INT1
            ;|||||+----Прерывание от вывода INT1 возникает по фронту
            ;||||+-----Сбросить запрос прерывания от вывода INT1
            ;|||+------Отключить таймер T0
            ;||+-------Обнулить флаг таймера T0
            ;|+--------Отключить таймер T1
            ;+---------Обнулить флаг таймера T1
```

Последовательный порт микроконтроллеров семейства MCS-51

Через универсальный последовательный порт микроконтроллера осуществляются прием и передача информации, представленной в последовательном коде (младшими битами вперед). Наличие буферного регистра приемника позволяет совмещать операцию чтения ранее принятого байта с приемом очередного. Но если к моменту окончания приема байта предыдущий не был считан из SBUF, то он будет потерян. Работой последовательного порта управляют три регистра:

- регистр управления/статуса приемопередатчика SCON;
- регистр управления мощностью PCON, бит SMOD;
- буферный регистр приемопередатчика SBUF.

Адрес	ст.зн. разр.							мл.зн. разр.	
98H	SM0 SCON.7	SM1 SCON.6	SM2 SCON.5	REN SCON.4	TB8 SCON.3	RB8 SCON.2	TI SCON.1	RI SCON.0	SCON
	9Fh	9Eh	9Dh	9Ch	9Bh	9Ah	99h	98h	

Наименование	Позиция	Назначение
SM0 SM1	SCON.7 SCON.6	Биты управления режимом работы приемопередатчика. Устанавливаются/сбрасываются программно см. примечание 1

SM0	SM1	Режим работы приемопередатчика
0	0	Сдвигающий регистр расширения ввода/вывода
0	1	8-битовый приемопередатчик, изменяемая скорость приема/передачи
1	0	9-битовый приемопередатчик, фиксированная скорость приема/передачи
1	1	9-битовый приемопередатчик, изменяемая скорость приема/передачи

SM2	SCON.5	Бит управления режимом приемопередатчика. Устанавливается программно для запрета приема сообщения, в котором девятый бит имеет значение 0
REN	SCON.4	Бит разрешения приема. Устанавливается/сбрасывается программно для разрешения/запрета приема последовательных данных
TB8	SCON.3	Передача бита 8. Устанавливается/сбрасывается программно для задания девятого передаваемого бита в режиме 9-битового передатчика
RB8	SCON.2	Прием бита 8. Устанавливается/сбрасывается аппаратно для фиксации девятого принимаемого бита в режиме 9-битового приемника
TI	SCON.1	Флаг прерывания передатчика. Устанавливается аппаратно при окончании передачи байта. Сбрасывается программно после обслуживания прерывания
RI	SCON.1	Флаг прерывания приемника. Устанавливается аппаратно при приеме байта. Сбрасывается программно после обслуживания прерывания

Рис. 20.22. Формат регистра управления последовательным портом

Последовательный порт может работать в четырех различных режимах:

- режим 0, синхронный;
- режим 1, асинхронный 8-битовый обмен;
- режим 2, асинхронный 9-битовый обмен с фиксированной скоростью передачи;
- режим 3, асинхронный 9-битовый режим.

Управление режимами работы приемопередатчика осуществляется через специальный регистр с символическим именем SCON. Он содержит не только управляющие биты, определяющие режим работы последовательного порта, но и девятый бит принимаемых или передаваемых данных (RB8 и TB8) и биты прерывания приемника/передатчика (RI и TI). Формат регистра управления последовательным портом приведен на рис. 20.22.

Прикладная программа путем загрузки в старшие биты регистра SCON двухбитного кода задает режим работы приемопередатчика. Во всех четырех режимах передача инициализируется любой командой, в которой буферный регистр SBUF указан как получатель байта. Как уже отмечалось, прием в режиме 0 осуществляется при условии, что RI=0 и REN=1, в остальных режимах — при условии, что REN=1.

В бите TB8 программно устанавливается значение девятого бита данных, который будет передан в режиме 2 или 3 в составе 9-битового поля данных кадра передачи. В бите RB8 в этих режимах запоминается девятый принимаемый бит данных. В режиме 1 в бит RB8 заносится значение стоп-бита. В режиме 0 бит RB8 не используется.

Флаг прерывания передатчика TI устанавливается аппаратно после передачи стоп-бита во всех режимах. Подпрограмма, обслуживающая прерывания или опрашивающая флаг, должна сбрасывать бит TI.

Флаг прерывания приемника RI устанавливается аппаратно после приема восьмого бита данных в режиме 0 и в середине периода приема стоп-бита в режимах 1, 2 и 3. Подпрограмма обслуживания прерывания должна сбрасывать бит RI.

Скорость приема/передачи информации через последовательный порт

Скорость приема/передачи в различных режимах задается разными способами.

В режиме 0 частота передачи зависит только от резонансной частоты кварцевого резонатора $f_{рез}$:

$$f = \frac{f_{рез}}{12}.$$

При этом за машинный цикл последовательный порт передает/принимает один бит информации.

В режимах 1, 2 и 3 скорость приема/передачи зависит от значения управляющего бита SMOD в регистре специальных функций PCON. Формат регистра управления питанием, в котором расположен бит управления скорости передачи, приведен на рис. 20.23.

Адрес	ст.зн. разр.							мл.зн. разр.	
87H	SMOD PCON.7	— PCON.6	— PCON.5	— PCON.4	GF1 PCON.3	GF0 PCON.2	PD PCON.1	IDL PCON.0	PCON

Наименование	Позиция	Назначение
SMOD	PCON.7	Удвоенная скорость. При SMOD=1 скорость приема/передачи вдвое больше, чем при SMOD = 0. По сбросу SMOD = 0
	PCON.6	Не используется
	PCON.5	Не используется
	PCON.4	Не используется
GF1 GF0	PCON.3 PCON.2	Флаги, специфицируемые пользователем (флаги общего назначения)
PD	PCON.1	Бит пониженной мощности. При PD=1 микроЭВМ переходит в режим пониженной потребляемой мощности
IDL	PCON.0	Бит холостого хода. Если IDL=1, то микроЭВМ переходит в режим холостого хода

Примечание
При одновременной записи 1 в PD и IDL бит PD имеет преимущество. Сброс содержимого PCON выполняется путем загрузки в него кода 0XXX0000.

Рис. 20.23. Формат регистра управления питанием

В режиме 2 частота приема/передачи определяется выражением

$$f = \frac{2^{SMOD} \times f_{рез}}{64}.$$

Иными словами, при SMOD = 0 частота передачи равна 1/64 частоты кварцевого резонатора $f_{рез}$, а при SMOD = 1 частота передачи равна 1/32 частоты кварцевого резонатора $f_{рез}$.

В режимах 1 и 3 в формировании частоты приема/передачи, кроме управляющего бита SMOD, принимает участие таймер 1. При этом частота прие-

ма/передачи f зависит от частоты переполнения таймера 1 — f_{ovfT1} и определяется следующим образом:

$$f = \frac{2^{SMOD} \times f_{ovfT1}}{32}$$

При использовании таймера 1 для тактирования последовательного порта прерывания от этого таймера должны быть запрещены. Таймер может быть использован как в режиме 16-разрядного таймера, так и в режиме таймера с автозагрузкой. Обычно используется режим таймера с автозагрузкой (старшая тетрада регистра TMOD = 0010B). При этом скорость передачи последовательного порта определяется выражением:

$$f = \frac{2^{SMOD} \times f_{\text{рез}}}{(32 \times 12 \times (256\text{-TH1}))}$$

Предельно низких скоростей приема и передачи по последовательному порту можно достичь при использовании таймера в режиме 1 (старшая тетрада регистра TMOD = 0001B). Перезагрузка 16-битного таймера должна осуществляться программным путем. При этом для того, чтобы можно было, кроме приема/передачи, выполнять дополнительные задачи, необходимо использовать механизм обработки прерываний и для этого разрешить прерывания от таймера 1.

Отметим, что для старших моделей семейства MCS-51 при использовании для синхронизации последовательного порта таймеров 1 и 2 скорости приема и передачи информации по последовательному порту могут различаться.

Режим 0. Синхронный режим работы последовательного порта

В режиме 0 последовательный порт работает как обыкновенный сдвиговый регистр. Это позволяет использовать его для увеличения количества входов-выходов устройства. Применение сдвиговых регистров для увеличения количества цифровых выходов микропроцессорной системы показано на рис. 20.24. В этом случае регистры территориально могут быть расположены около исполняющего устройства или даже внутри управляемой микросхемы. Как видно из приведенной схемы, количество проводников, по которым передаются сигналы управления, при этом может быть сокращено до двух.

Передача данных через последовательный порт начинается с записи байта в регистр данных SBUF. После завершения передачи аппаратно устанавливается флаг прерывания передатчика TI. Временная диаграмма сигнала, вырабатываемого последовательным портом микроконтроллера, при передаче восьми бит данных приведена на рис. 20.25.

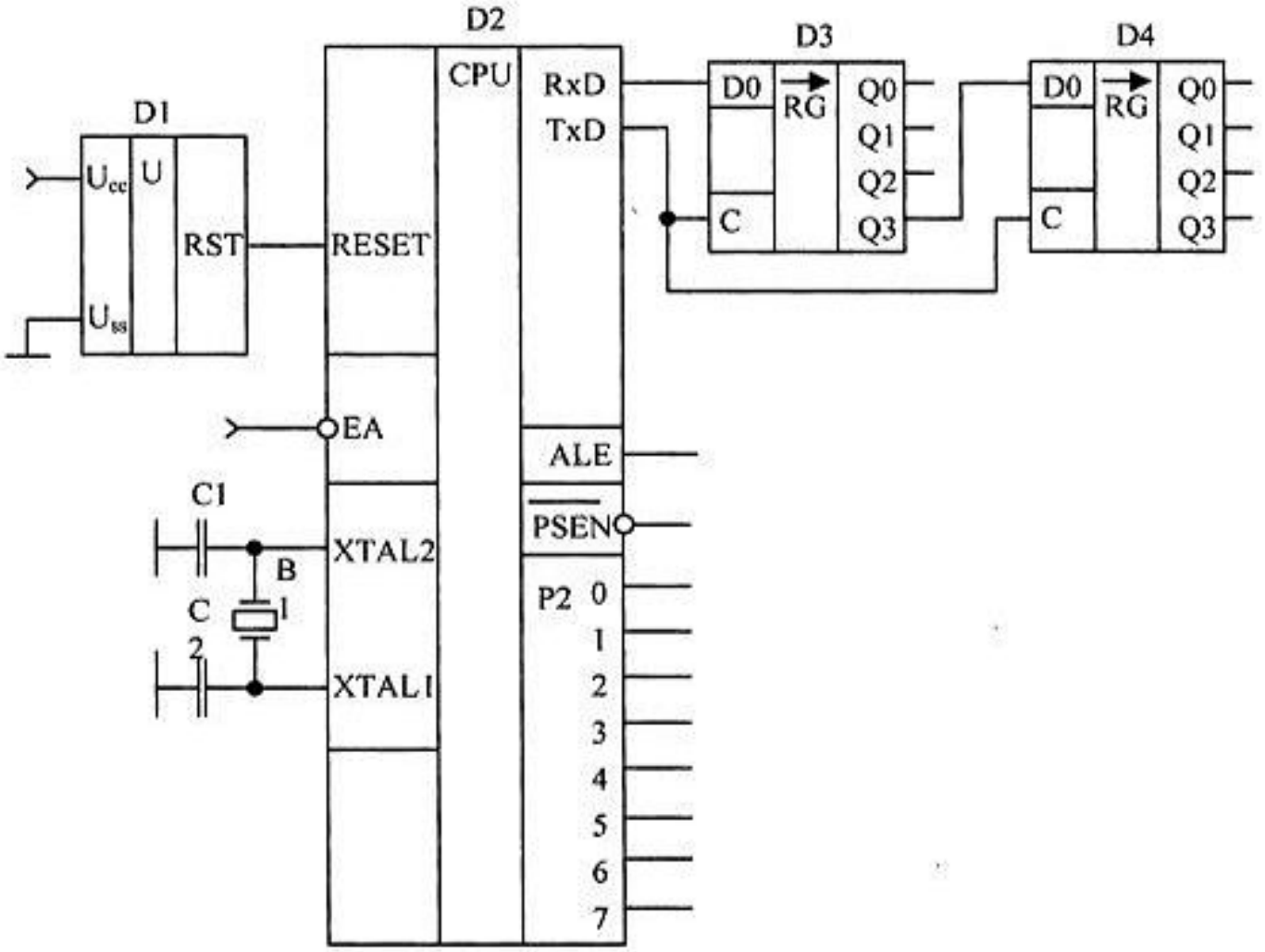


Рис. 20.24. Схема подключения сдвиговых регистров к последовательному порту микроконтроллера для увеличения количества выводов устройства, работающих на выход

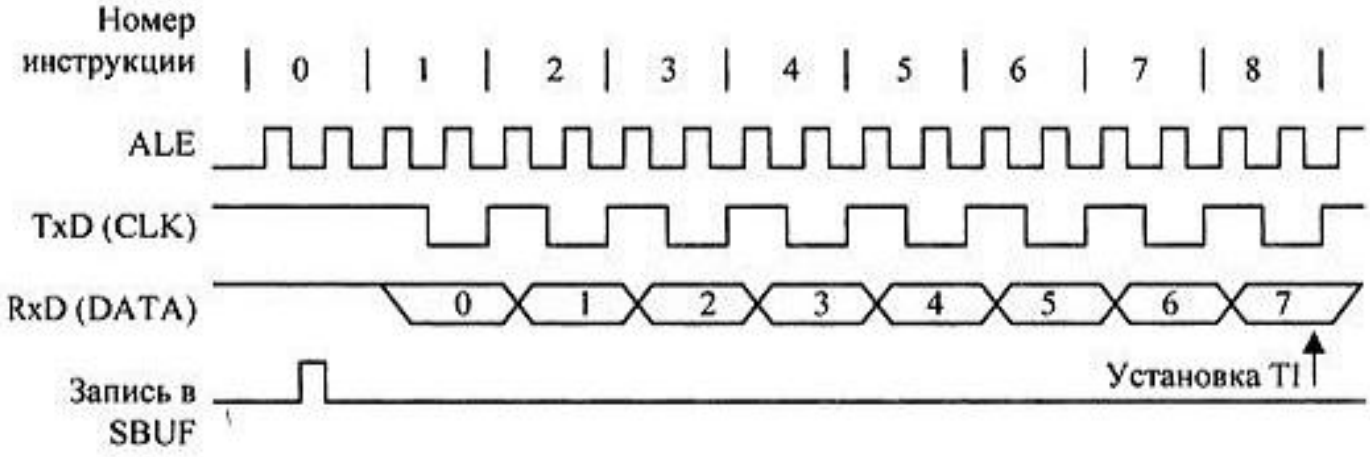


Рис. 20.25. Временная диаграмма работы последовательного порта в режиме 0 после записи передаваемого байта в регистр данных SBUF

Для осуществления передачи байта данных достаточно занести его в буфер данных SBUF, как это показано в листинге 20.25.

Листинг 20.25. Передача байта данных в режиме 0

```
MOV SCON, #0    ;Настроить последов. порт на передачу в режиме 0

MOV SBUF, A     ;Передать содержимое аккум. по последовательному порту и
JNB TI, $       ;подождать окончания передачи

MOV SBUF, #56H ;Передать по последовательному порту число 56h и
JNB TI, $       ;подождать окончания передачи
```

Применение сдвиговых регистров для увеличения количества цифровых входов микропроцессорной системы показано на рис. 20.24. В этом случае, как и в случае формирования управляющих сигналов, регистры территориально могут быть расположены около устройства сбора информации или даже

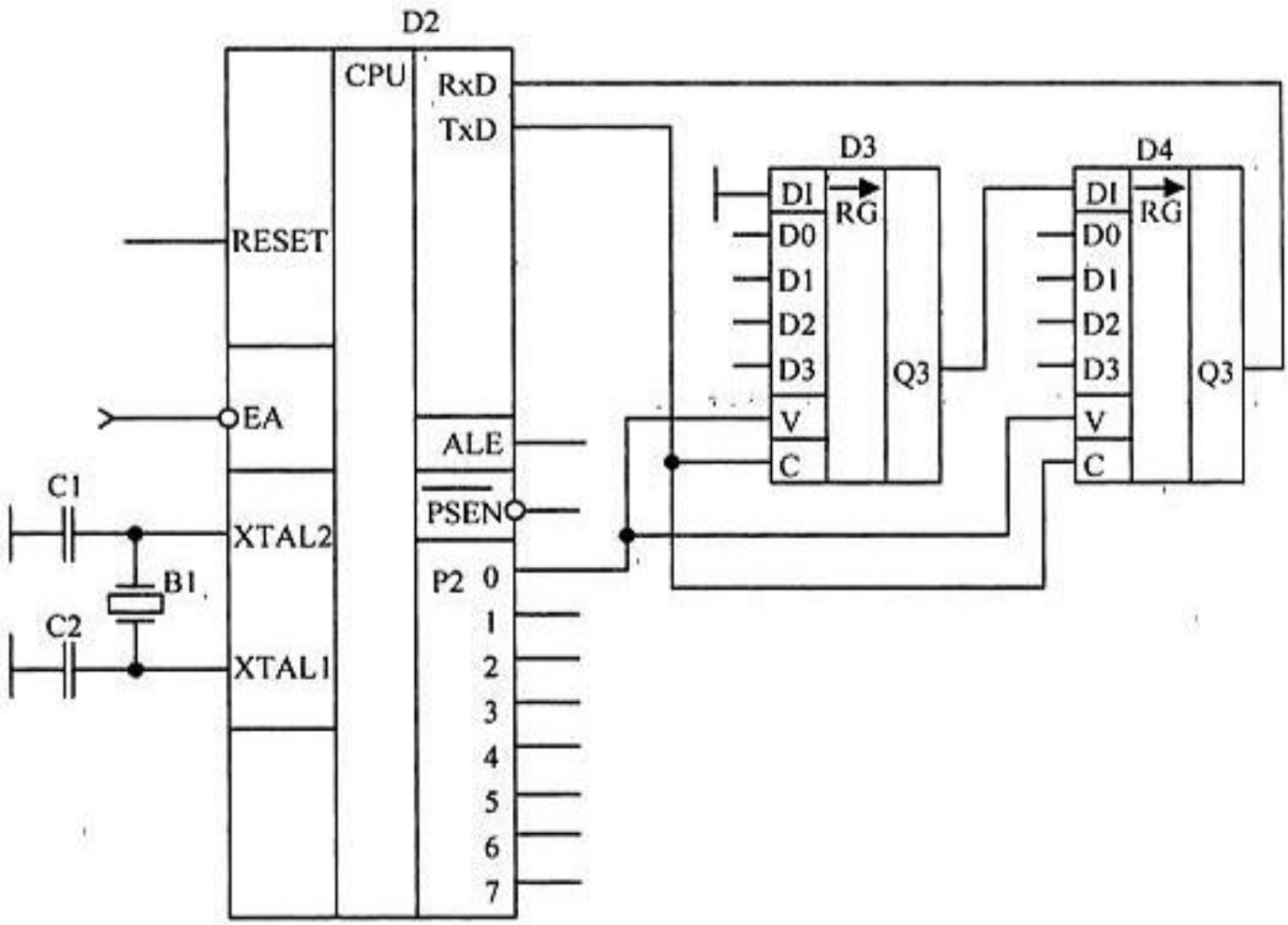


Рис. 20.26. Использование режима 0 работы последовательного порта для увеличения количества выводов устройства, работающих на ввод

внутри контролируемой микросхемы. Как видно из приведенной схемы, количество проводников, по которым передаются сигналы управления, при этом может быть сокращено до трех. Третий провод предназначен для передачи сигнала записи опрашиваемых данных в сдвиговые регистры. Этот сигнал придется сформировать программно.

Прием байта через последовательный порт в режиме 0 начинается после обнуления флага готовности приемника RI. Временная диаграмма приема входной информации последовательным портом в режиме 0 приведена на рис. 20.27.

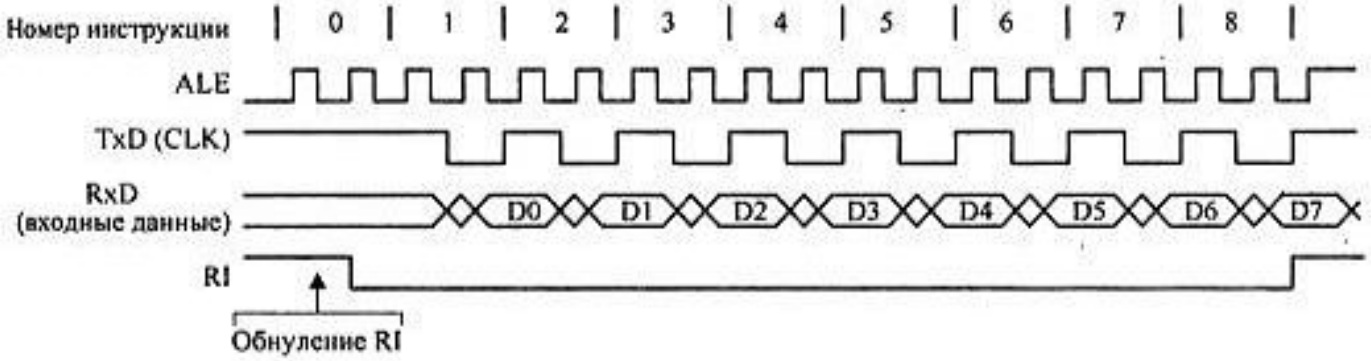


Рис. 20.27. Временная диаграмма приема входной информации последовательным портом в нулевом режиме после обнуления флага готовности приемника RI

Для осуществления приема байта данных достаточно настроить порт на прием в синхронном режиме (режим 0) и обнулить флаг приема RI, как это показано в примере программы, приведенной в листинге 20.26. Затем можно считывать принятый байт из регистра буфера данных SBUF.

Листинг 20.26. Прием байта в режиме 0

```
;Настроить режим работы последовательного порта-------------------------
 MOV SCON,#00010001b  ;настроить последов. порт на режим 0
          ;||||||||
          ;|||||||+-------Установить флаг приемника RI
          ;||||||+--------Обнулить флаг передатчика TI
          ;|||||+---------Обнулить девятый бит приемника RB8
          ;||||+----------Обнулить девятый бит передатчика TB8
          ;|||+-----------Разрешить работу приемника
          ;||+------------В синхронном режиме не имеет значения
          ;++-------------Включить синхронный режим работы
          ;                последовательного порта

  SETB P2.0             ;Сформировать импульс параллельной записи данных
  CLR  P2.0             ;в последовательные регистры
```

```
CLR  RI              ;С этого момента начинается прием байта

JNB RI, $            ;Подождать окончания приема байта по последоват.
MOV A, SBUF          ;порту и скопировать его в аккумулятор
```

Синхронный режим 0 задается записью комбинации 00 в биты SM0, SM1 регистра SCON. В этом режиме информация передается/принимается через вывод входа приемника RxD, т. е. в этом режиме работы последовательный порт работаем в симплексном режиме. Через вывод TxD выдаются импульсы синхронизации, которые сопровождают каждый информационный бит. Скорость передачи в этом режиме фиксирована и составляет $F_{ген}/12$. Это означает, что при частоте задающего генератора 24 МГц обмен данными осуществляется на скорости 2 Мбит/с.

В настоящее время разработано огромное количество микросхем, управление которыми осуществляется по синхронному последовательному порту. Это, например, синтезаторы частоты, микросхемы приемников, блоков цветности телевизоров, памяти. При этом микросхемы обычно реализуют синхронные протоколы обмена SPI или I^2C.

Последовательный порт микроконтроллеров семейства MCS-51, работающий в режиме 0, позволяет осуществлять обмен с такими микросхемами при минимальных программно-аппаратных затратах. Справедливости ради необходимо отметить, что в современных микросхемах этого семейства присутствуют отдельные последовательные порты, работающие по протоколу SPI или I^2C. В качестве примера такой микросхемы можно назвать микроконтроллер ADuC834 фирмы Analog Devices. В подобных микросхемах последовательный порт используется исключительно для связи с универсальным компьютером через стандартный последовательный интерфейс RS232 или с любым другим микроконтроллером, обладающим асинхронным последовательным портом.

Режим 1. Асинхронный 8-битовый режим

В режиме 1 последовательный порт работает в асинхронном режиме, принципы работы которого рассматривались ранее. Временная диаграмма передаваемых через последовательный порт сигналов в асинхронном последовательном режиме работы совпадает с временной диаграммой, приведенной на рис. 20.28.

Режим 1 задается записью комбинации 01 в биты SM0 и SM1 регистра SCON. В асинхронном режиме информация передается через вывод выхода передатчика последовательного порта микроконтроллера TxD, а принимается через вывод входа приемника RxD, т. е. в этом режиме работы последовательный порт работает в дуплексном режиме. Это означает, что передача и прием ин-

формации может вестись одновременно и независимо друг от друга. Скорость передачи в этом режиме задается при помощи таймера T1.

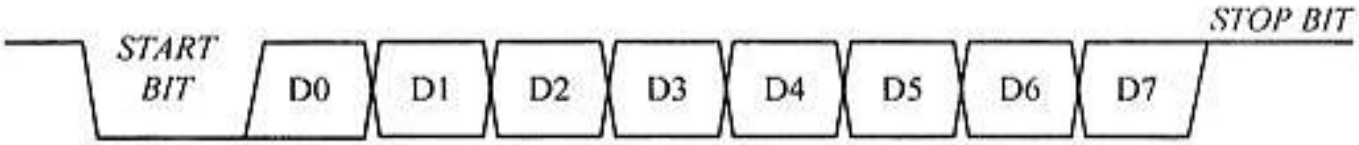


Рис. 20.28. Временная диаграмма приема или передачи информации последовательным портом в режиме 1

При работе в асинхронном режиме два микроконтроллера могут обмениваться информацией между собой, используя минимум соединительных проводов между блоками или даже отдельными устройствами. Единственное условие: скорости работы передающего и принимающего последовательных портов должны быть одинаковыми. При разработке функционально законченного устройства может быть использована любая скорость обмена данными, однако при разработке устройства, способного работать с другими устройствами, требуется договариваться об определенных скоростях передачи.

Обычно используются стандартные скорости передачи, такие как 1200 бит/с, 2400 бит/с и т. д. Любая стандартная скорость передачи может быть получена путем деления или умножения скорости 1200 бит/с на два. Для получения таких скоростей передачи в микроконтроллерах, принадлежащих к семейству MCS-51, обычно используется кварцевый резонатор с частотой 11,0592 МГц. Значения констант, загружаемых в таймер 1 для получения стандартных скоростей приема/передачи при использовании кварцевого резонатора с данной резонансной частотой, приведены в табл. 20.5.

Таблица 20.5. *Настройка таймера 1 для некоторых стандартных скоростей обмена через последовательный порт в режимах 1 и 3*

Частота приема/передачи	Частота резонатора МГц	Таймер/счетчик 1			
		SMOD	C/T	Режим (M1, M0)	Перезагружаемое число
62,2 Кбит/с	12	1	0	1, 0	0FFH
19,2 Кбит/с	11,059	1	0	1, 0	0FDH
9,6 Кбит/с	11,059	0	0	1, 0	0FDH
4,8 Кбит/с	11,059	0	0	1, 0	0FAH
2,4 Кбит/с	11,059	0	0	1, 0	0F4H
1,2 Кбит/с	11,059	0	0	1, 0	0E8H
110 бит/с	6	0	0	1, 0	072H

В отличие от синхронного режима 0 в асинхронном режиме 1 возможен обмен информацией между двумя микроконтроллерами, а не только между микроконтроллером и исполнительными микросхемами. Схема соединения двух микроконтроллеров между собой для обмена информацией приведена на рис. 20.29. Подобным образом может быть построена простейшая многопроцессорная система.

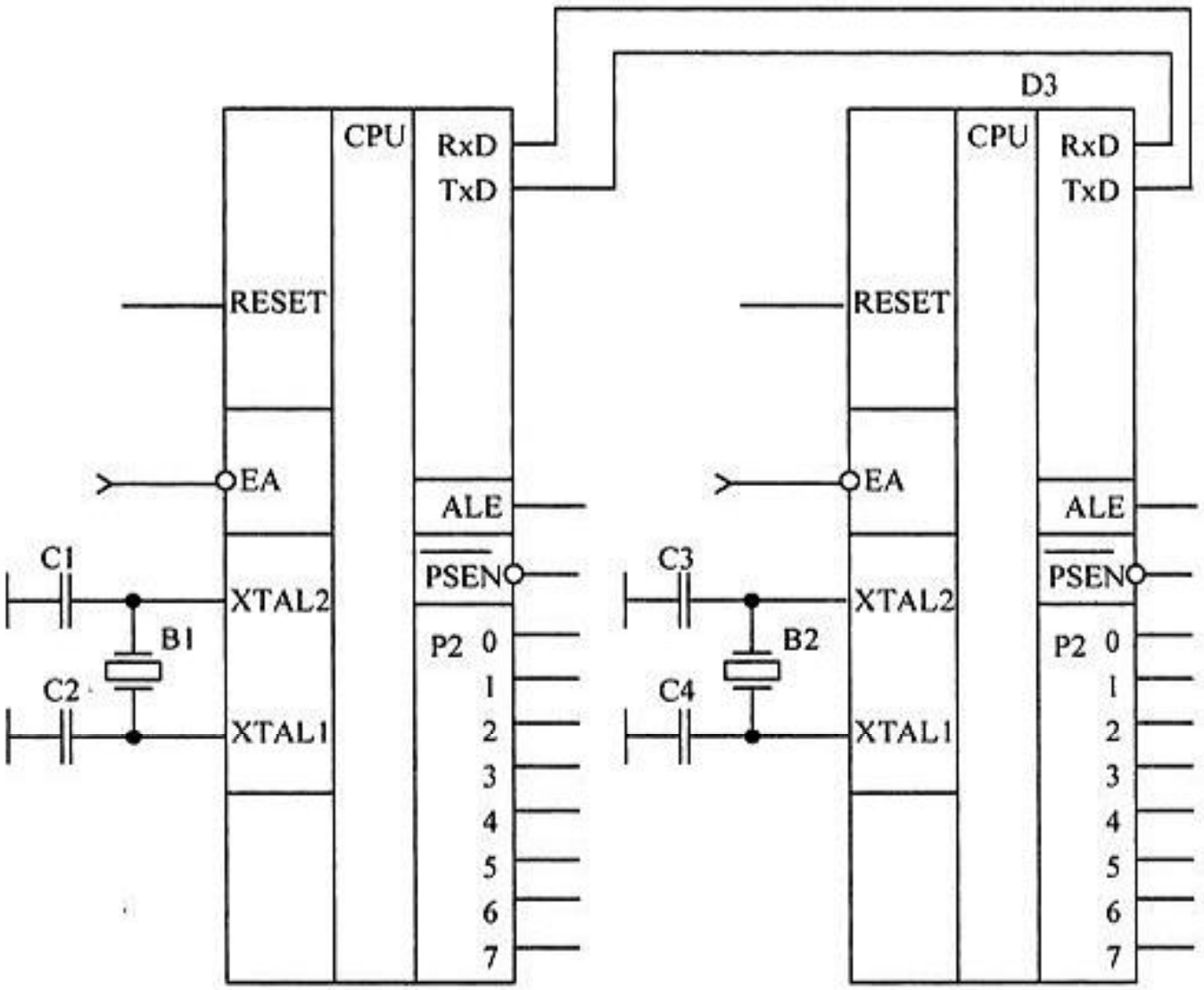


Рис. 20.29. Схема обмена информацией между двумя микроконтроллерами через последовательные порты

В режиме 1 для передачи байта через последовательный порт достаточно скопировать его в буфер данных SBUF. Единственное отличие от работы в синхронном режиме заключается в том, что, кроме настройки регистра SCON, необходимо настроить таймер для задания скорости передачи информации по последовательному порту. Прием начинается только после обнаружения стартового бита. Пример программы, позволяющей осуществить прием информации по последовательному порту в режиме асинхронного обмена, приведен в листинге 20.27.

Листинг 20.27. Прием информации через последовательный порт в режиме асинхронного обмена

```
;*****************************************************************
;НАСТРОЙКА ПОСЛЕДОВАТЕЛЬНОГО ПОРТА
;*****************************************************************
;Настроить режим работы последовательного порта-------------------------
  MOV SCON,#01110000b ;настроить последовательный порт на режим 1
           ;||||||||
           ;|||||||+---Обнулить флаг приемника RI
           ;||||||+----Обнулить флаг передатчика TI
           ;|||||+-----Обнулить девятый бит приемника RB8
           ;||||+------Обнулить  девятый бит передатчика TB8
           ;|||+-------Разрешить работу приемника
           ;||+--------Проверять ошибку кадра (прием нулевого бита на
           ;||          месте стоп-бита)
           ;++---------Включить асинхронный режим работы

;Настроить режим работы таймера T1 -------------------------
  ANL TMOD,#00001111b ;Подготовить таймер T1 к настройке
                      ;(таймер T0 не трогать!)
  ORL TMOD,#00100000b ;перевести таймер T1 в режим 2
           ;||||       ;(таймер T0 не трогать!)
           ;||++-------Перевести таймер T1 в режим автозагрузки
           ;|+--------Работа от внутреннего генератора
           ;+---------Запретить управление таймером от вывода INT1

;Настроить таймер на генерацию 3-микросекундного интервала времени-------
  MOV TH0, #0fdh          ;Загрузить старший байт таймера
  MOV TL0, #0fdh          ;Загрузить младший байт таймера
  SETB TR1                ;Включить таймер 1

;*****************************************************************
;РАБОТА С ПОСЛЕДОВАТЕЛЬНЫМ ПОРТОМ
;*****************************************************************
  JNB RI, $   ;Подождать окончания приема байта по последовательному
  MOV A, SBUF ;порту и скопировать его в аккумулятор
```

Возможность работы в асинхронном режиме позволяет использовать последовательный порт для связи с универсальным компьютером через его последовательный порт COM. К сожалению, уровни сигналов последовательного порта микроконтроллера не соответствуют спецификациям стандартного интерфейса RS232, используемого в COM-порту универсальных компьютеров,

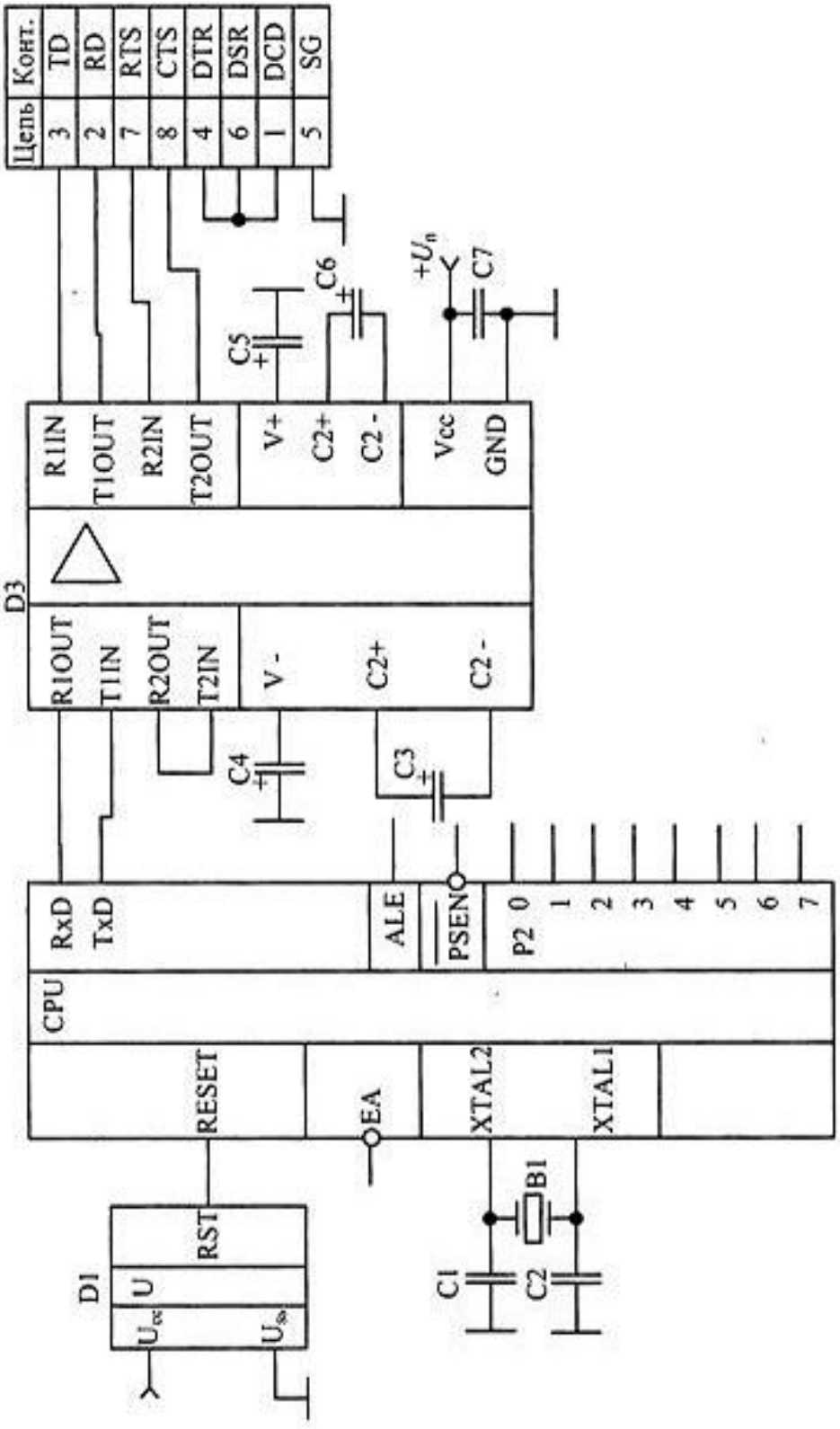


Рис. 20.30. Подключение последовательного порта микроконтроллеров семейства MCS-51 к последовательному COM-порту компьютера

поэтому для подключения приходится дополнительно использовать специализированные микросхемы согласования уровней. Эти же микросхемы обеспечивают защиту микроконтроллера от вывода из строя статическим потенциалом при подключении разъемов.

Обычно для обмена информацией используются только сигнальные цепи COM-порта компьютера. Тем не менее, оставшиеся буферы интерфейсной микросхемы могут быть использованы для контроля питания микроконтроллерной схемы. Для этого сигнал запроса готовности удаленного устройства от компьютера пропускается через интерфейсную микросхему. Затем этот же сигнал подается на вход подтверждения готовности удаленного устройства. Если на интерфейсную микросхему не будет подано питание, то сигнал подтверждения готовности не будет сформирован.

Типовая схема подключения компьютера к последовательному порту микроконтроллеров семейства MCS-51 с применением интерфейсной микросхемы ADM3202 приведена на рис. 20.30.

Использование последовательного порта компьютера позволяет не только управлять микроконтроллерным устройством, используя клавиатуру компьютера, но и отображать внутреннюю информацию этого устройства, используя дисплей компьютера. Это значительно расширяет возможности ввода и вывода информации в микроконтроллерных устройствах. В последнее время дополнительно появилась возможность заносить программу во внутреннюю память программ наиболее современных микроконтроллеров, например, ADuC841. Эта возможность позволяет модернизировать микропроцессорные устройства, не применяя специализированных программаторов и даже не разбирая устройство.

Режим 2. Асинхронный 9-битовый режим с фиксированной скоростью передачи

В этом режиме последовательный порт работает на фиксированной скорости передачи, так же как в режиме 0. Скорость обмена определяется значением бита SMOD и при частоте кварцевого резонатора 12 МГц составляет 375 Кбит/с. У современных микроконтроллеров, способных работать с более высокой тактовой частотой, скорость обмена может превышать 1 Мбит/с.

Основной особенностью работы последовательного порта в этом режиме является передача девятого информационного бита, который может быть использован для контроля достоверности передаваемой информации. Для вычисления четности передаваемого байта можно воспользоваться аппаратным вычислителем, подключенным к аккумулятору микроконтроллера. Результат вычисления четности байта сохраняется в бите четности P регистра PSW,

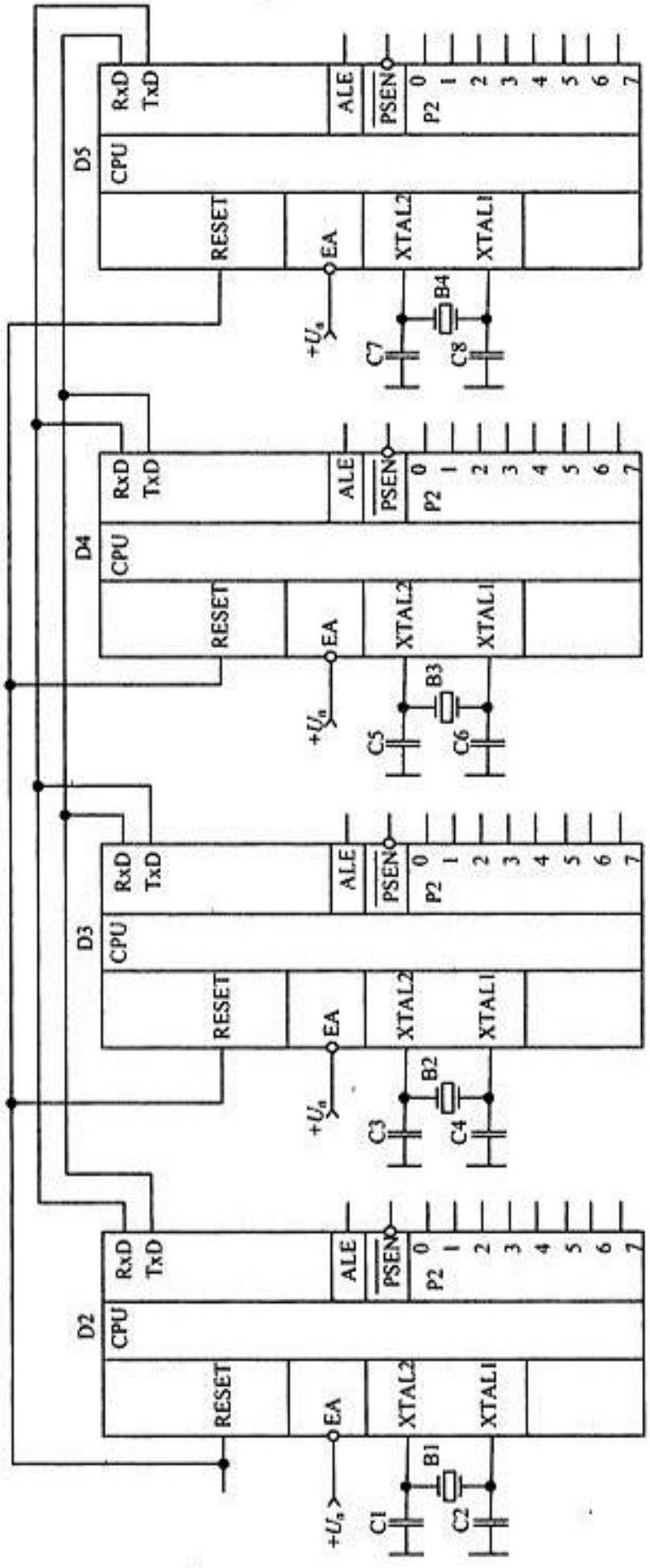


Рис. 20.31. Схема соединения нескольких микроконтроллеров между собой через последовательные порты, работающие в асинхронном режиме

откуда его можно скопировать в девятый информационный бит последовательного порта TB8, расположенный в регистре управления последовательным портом SCON.

Еще бóльшие возможности для построения устройств предоставляет 9-разрядный режим работы при реализации многопроцессорных систем. Параллельные порты микроконтроллеров семейства MCS-51 построены по схеме с открытым стоком. Это позволяет объединять несколько выходов передатчиков в одну шину. Такое выполнение выходных каскадов микросхем облегчает построение многопроцессорных систем. В многопроцессорной системе один процессор должен быть главным (master), остальные — подчиненными (slave). Естественно, что при таком построении системы, команды главного процессора должны восприниматься подчиненными, поэтому выход передатчика последовательного порта главного процессора соединяется с входами приемников подчиненных. Выходы передатчиков подчиненных процессоров при этом объединяются и подключаются к входу приемника последовательного порта главного процессора. Схема многопроцессорной системы, построенной по подобному принципу, приведена на рис. 20.31.

Команды главного процессора могут быть обращены к конкретному подчиненному процессору, поэтому в состав команд включается адрес подчиненного процессора. При работе в шине необходимо уметь отличать адресную информацию от данных. Это можно осуществить при помощи девятого бита. Обычно при передаче адреса в девятый бит записывают единицу, а при передаче данных и команд — ноль. Таким образом, микроконтроллер, даже подключившийся к шине позднее остальных, легко может осуществить синхронизацию с многопроцессорной шиной. Временная диаграмма работы многопроцессорной шины приведена на рис. 20.32.

Рис. 20.32. Временная диаграмма работы многопроцессорной последовательной шины

Настройка и работа с портом в режиме 2 никаких особенностей не имеет, поэтому можно воспользоваться примером настройки последовательного порта (см. листинг 20.27), приведенным для режима 1. Все особенности работы сосредоточены на протокольном уровне, а это не входит в задачу рассмотрения принципов работы последовательного порта.

Режим 3. Асинхронный 9-битовый режим

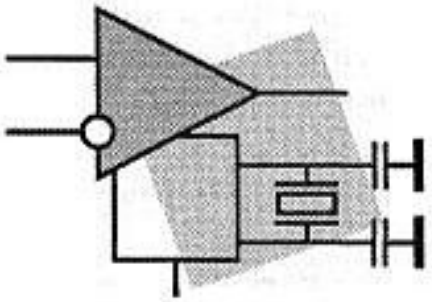
Работа последовательного порта в этом режиме не отличается от работы в режиме 2 за исключением скорости передачи. Скорость передачи по последовательному порту задается таймером 1, так же как и в режиме 1. Для построения программы можно воспользоваться примером, приведенным для режима 1 (см. листинг 20.27), с учетом того, что в регистре SCON необходимо задать вместо режима 1 режим 3.

Итоги

В главе было рассмотрено внутреннее устройство и система команд наиболее распространенного семейства микроконтроллеров. При описании внутренних узлов этого семейства были приведены типовые участки программ, иллюстрирующие работу с этими узлами. Там, где это необходимо, были приведены участки принципиальных схем устройств с использованием рассматриваемого микроконтроллера.

Естественно, что в одной книге невозможно охватить все особенности применения микроконтроллеров. Тем более, что у каждого вида микросхем, принадлежащего к этому семейству, имеются свои особенности. Тем не менее были рассмотрены узлы, присутствующие во всех микросхемах данного семейства. При необходимости полученные знания можно применить для того, чтобы разобраться и научиться управлять узлами микроконтроллеров, которые не были рассмотрены в данной книге. Это актуально даже для микросхем, которые только появятся на рынке через несколько лет.

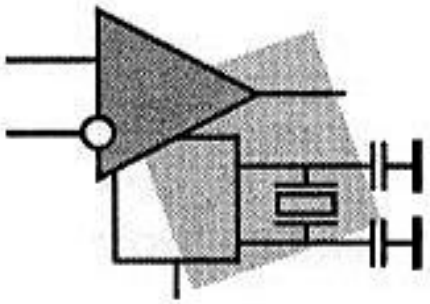
Однако уметь управлять отдельными узлами микроконтроллера еще не значит уметь писать программы. Точно так же, как недостаточно иметь в наличии кирпичи, доски, окна и двери для того, чтобы построить дом. При написании программ используются специфические приемы, позволяющие реализовывать законченные блоки аппаратуры и организовывать взаимодействие между ними. Тем не менее, теперь можно перейти к изучению принципов написания программ для микроконтроллеров, реализующих различные цифровые устройства. Именно этим мы и займемся в следующей главе.



ЧАСТЬ V

Программирование микроконтроллеров

ГЛАВА 21



Принципы создания программ для микроконтроллеров

В предыдущих главах были рассмотрены основы построения микропроцессорных систем, подробно описано внутреннее устройство самого распространенного семейства микроконтроллеров — MCS-51. Однако одновременно выяснилось, что для разработки устройства на основе микроконтроллера создания одной только принципиальной схемы недостаточно. Кроме разработки принципиальной схемы, требуется разработать и занести во внутреннюю память микроконтроллера программу. Именно она вместе с аппаратурой микроконтроллера и обеспечивает функционирование проектируемого устройства.

В предыдущей главе была рассмотрена система команд микроконтроллера, но знания только системы команд недостаточно для создания программного обеспечения микропроцессорной системы. Нужны специальные программные средства, позволяющие облегчить процесс разработки программы. В настоящее время предлагаются инструментальные интегрированные среды разработки программ.

Микроконтроллеры предназначены для миниатюризации устройств управления, поэтому они интересны, прежде всего, для разработчиков различной аппаратуры. Однако обычно у этих специалистов отсутствуют навыки программирования. В данной главе будут описаны основные принципы разработки программного обеспечения, причем основное внимание будет уделено особенностям построения программ для микроконтроллеров.

Основное внимание будет уделено методу структурного программирования, как наиболее подходящему для современного уровня развития микроконтроллеров. Кроме того, мы остановимся на особенностях построения программ, реагирующих на сигналы, поступающие извне, рассмотрим программную реализацию блоков, которые ранее выполнялись аппаратно, и взаимодействие программы с человеком.

Все современные программы пишутся на каких-либо языках программирования, поэтому начнем главу с обзора существующих языков программирования и программ-трансляторов этих языков.

Языки программирования для микроконтроллеров

Программирование для микроконтроллеров, как и программирование для универсальных компьютеров, прошло большой путь развития — от использования машинных кодов до применения современных интегрированных сред, предоставляющих встроенный текстовый редактор, обеспечивающих трансляцию программ, их отладку и загрузку во внутреннюю память микроконтроллеров. В настоящее время исходный текст программы пишется на каком-либо из языков программирования, относящемся к одной из двух групп:

- языки программирования "низкого" уровня;
- языки программирования "высокого" уровня.

Классификация языков программирования приведена на рис. 21.1. В языках "низкого" уровня каждому оператору соответствует не более одной машинной команды. В их состав обязательно входит набор машинных команд каждого конкретного процессора. Языки программирования низкого уровня в настоящее время называются ассемблерами (старое название "автокоды"). Для каждого процессора существует своя группа ассемблеров. Ассемблеры для одного и того же процессора различаются между собой дополнительными возможностями, облегчающими программирование.

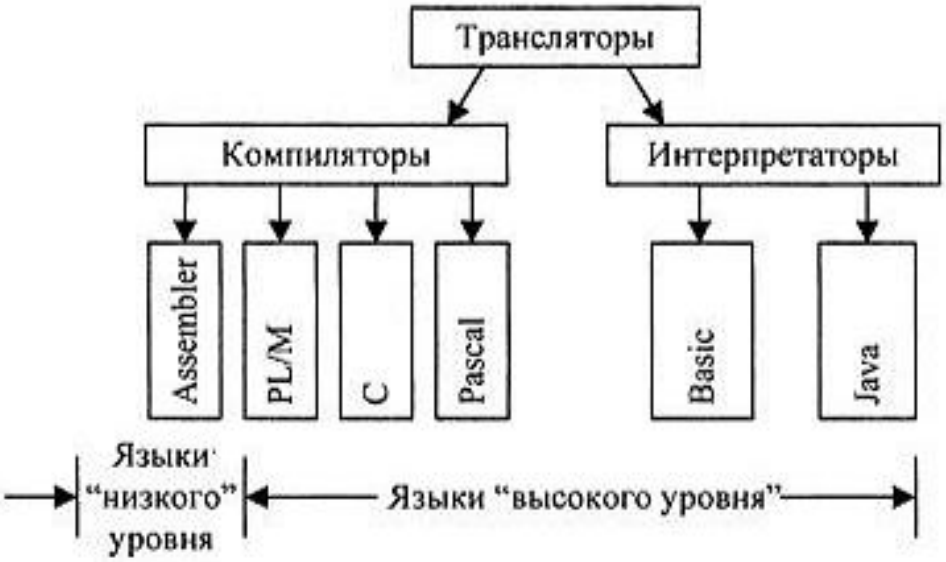


Рис. 21.1. Классификация языков программирования

Языки программирования "высокого" уровня позволяют заменять один оператор несколькими машинными командами. Это увеличивает производительность труда программистов. Кроме того, языки "высокого" уровня позволяют писать программы, которые могут выполняться на различных микропроцессорах. (Естественно, что при этом необходимо использовать программы-трансляторы для соответствующего процессора.) В настоящее время для микроконтроллеров широко используются такие языки программирования высокого уровня, как C и PLM.

О преимуществах и недостатках языков высокого и низкого уровней говорилось достаточно много. Выбор языка программирования зависит от состава аппаратуры, для которой пишется программа, наличия тех или иных средств разработки программного обеспечения, а также от требуемого быстродействия всего программно-аппаратного комплекса в целом.

В тех случаях, когда объем ОЗУ и ПЗУ мал (в районе нескольких килобайтов) альтернативы ассемблеру нет. Именно этот язык программирования позволяет получать самый короткий и самый быстродействующий код программы (при прочих равных условиях, т. к. испортить можно все!).

Языки программирования высокого уровня позволяют значительно сократить время создания программы, но при этом увеличивается ее размер, поэтому использование такого языка программирования для микропроцессорных систем требует достаточно большого объема памяти программ (несколько десятков килобайтов). Увеличение объема программы связано с несколькими факторами:

1. Язык программирования рассчитывается на все случаи жизни, поэтому в большинстве случаев человек мог бы написать программу короче, исключив ненужные в данном конкретном случае проверки или защиты.
2. Если программист не знает, к чему приводит использование тех или других операторов языка программирования, то он может выбирать операторы, не оптимальные как с точки зрения длины машинного кода программы, так и с точки зрения быстродействия программы.
3. Программист не использует подпрограммы там, где они могли бы сократить объем программы, т. к. на языке программирования высокого уровня это всего один или несколько операторов.

Первый из этих пунктов постепенно утрачивает свое значение с появлением все более совершенных трансляторов. Третий пункт тоже решается тем же путем при применении различных видов оптимизаторов, входящих в состав компилятора. Однако в большинстве случаев оптимизатор не может определить одинаковые действия, если они отличаются хотя бы одной командой. Кроме того, оптимизатор работает только в пределах одного программного модуля! Понятие программного модуля будет рассмотрено позднее.

Виды программ-трансляторов

Процесс преобразования операторов исходного языка программирования в машинные коды микропроцессора называется *трансляцией* исходного текста. В настоящее время ручная трансляция программ практически не используется. Трансляция производится специальными программами-трансляторами.

Существует два больших класса таких программ: компиляторы и интерпретаторы. При использовании компиляторов весь исходный текст программы преобразуется в машинные коды, и именно эти коды записываются в память микропроцессора. При использовании интерпретатора в память микропроцессора записывается исходный текст программы, а трансляция производится при считывании очередного оператора. Естественно, что быстродействие интерпретаторов намного ниже, чем у компиляторов, т. к., например, при использовании оператора в цикле он транслируется многократно.

Применение интерпретатора может обеспечить выигрыш только в случае его разработки для языка программирования "высокого" уровня. В этом случае может быть сэкономлена внутренняя память программ, а также облегчен процесс отладки (при применении языка программирования BASIC) или облегчен перенос программ с одного типа процессора на другой (в случае использования языка программирования Java).

При программировании на ассемблере применение интерпретатора приводит к проигрышу по всем параметрам, поэтому для языков программирования низкого уровня применяются только программы-компиляторы.

Для программирования микроконтроллеров как на языке программирования "низкого" уровня, так и на языке программирования "высокого" уровня чаще используются компиляторы, поэтому рассмотрим подробнее виды этих трансляторов.

Виды компиляторов

Программы-компиляторы бывают оценочные и профессиональные.

Оценочные компиляторы позволяют написать простейшие программы для конкретного процессора и определить, подходит ли процессор для тех задач, которые предстоит решать в процессе разработки устройства. Конечно, если программа очень проста, то можно весь программный продукт выполнить на оценочном компиляторе. Оценочные компиляторы позволяют транслировать одиночный файл исходного текста программы. Иногда такие компиляторы позволяют включать в процесс трансляции содержимое отдельных файлов специальной директивой. В результате работы оценочного компилятора сразу получается исполняемый или загрузочный модуль программы, поэтому такие компиляторы называются *компиляторами с единой трансляцией*.

Профессиональные трансляторы позволяют производить компиляцию исходного текста программы по частям. Это позволяет значительно сократить время трансляции, т. к. нужно компилировать не весь текст программы, а только ту ее часть, которая менялась после предыдущей трансляции.

Кроме того, каждый программный модуль может писать отдельный программист. Это позволяет сократить время написания программы. Даже в том случае, если программу пишет один человек, время написания программы сокращается за счет использования готовых отлаженных и оттранслированных программных модулей.

При многомодульном программировании процесс получения исполняемого кода программы разбивается на два этапа: компиляция исходного текста модулей (некоторых или всех) и связывание (компоновка) полученных объектных файлов модулей в единую программу. Такой процесс получил название *раздельная компиляция-компоновка*.

Оценочные компиляторы обычно предлагаются бесплатно фирмами-производителями микроконтроллеров.

Профессиональные компиляторы разрабатываются и продаются независимыми фирмами-разработчиками программного обеспечения. Для микроконтроллеров семейства MCS-51 получили известность продукты таких фирм, как FRANKLIN, IAR, KEIL и др.

В состав современных профессиональных средств написания и отладки программ для микроконтроллеров обычно входят эмуляторы процессоров или отладочные платы, текстовый редактор, компиляторы языка высокого уровня (чаще всего "С") и ассемблера, редактор связей (компоновщик) и загрузчик программы в отладочную плату. Все программы обычно объединены интегрированной средой разработки программного проекта, которая позволяет поддерживать один или несколько программных проектов.

Теперь, после того как были рассмотрены языки программирования для микроконтроллеров и программы-трансляторы, перейдем к обсуждению основных принципов написания программ.

Применение подпрограмм

В программах часто приходится повторять одни и те же операторы (например, при реализации алгоритма работы с параллельным или последовательным портом). Было бы неплохо использовать один и тот же участок кода, вместо того, чтобы повторять одни и те же операторы несколько раз.

Участок программы, к которому можно обращаться из различных мест программы для выполнения некоторых действий и последующего возврата в место вызова, называется *подпрограммой*.

Проблема, с которой приходится сталкиваться при многократном использовании участков кодов, — как определить, в какое место памяти программ возвращаться после завершения подпрограммы. Обращение к подпрограмме производится из нескольких мест основной программы. Описанную ситуацию иллюстрирует рис. 21.2. На нем изображено адресное пространство микроконтроллера. Младшие адреса адресного пространства находятся в нижней части рисунка.

Для того чтобы получить возможность возвращаться на команду, следующую за вызовом подпрограммы, требуется запомнить ее адрес. Адрес возврата хранится в особых ячейках памяти данных. После выполнения подпрограммы необходимо осуществить переход к адресу, который записан в этих ячейках.

Для обращения к подпрограмме и возврата из нее в систему команд микропроцессоров вводят специальные команды. В микроконтроллерах семейства MCS-51 это команды LCALL, ACALL для вызова подпрограммы и команда RET для возврата из подпрограммы. Команды вызова не только осуществляют передачу управления на указанный адрес, но и запоминают адрес команды, следующей за вызовом подпрограммы — адрес возврата. Команда возврата из подпрограммы передает управление по адресу возврата, которой был запомнен при вызове подпрограммы.

Рис. 21.2. Вызов подпрограммы и возврат к выполнению основной программы

Пример вызова подпрограммы и ее реализации на языке программирования ASM-51 приведен в листинге 21.1.

✧ *Внимание! Ни в коем случае нельзя попадать в подпрограмму любым способом, кроме команды вызова подпрограммы CALL!* В противном случае

команда возврата из подпрограммы передаст управление случайному адресу! По этому адресу могут быть расположены данные, которые в этом случае будут интерпретированы как программа, или обратиться к внешней памяти, откуда будут считываться случайные числа.

Листинг 21.1. Пример вызова подпрограммы

```
  ...
  MOV A,#56
  CALL PeredatByte
  ...
  MOV A,#37
  CALL PeredatByte
  ...

;*****************************************
;Подпрограмма передачи байта
;через последовательный порт
;*****************************************
PeredatByte:
  JB TI, $              ;Если предыдущий байт передан,
  MOV SBUF, G_Per       ;то передать очередной байт
  RET
```

В приведенном в листинге 21.1 примере перед подпрограммой *обязательно должен быть бесконечный цикл*. Иначе в подпрограмму можно попасть не через вызов подпрограммы, а при последовательном выполнении операторов. Тогда команда RET передаст управление на случайный адрес. Это может привести к непредсказуемым последствиям, при особенно неблагоприятных обстоятельствах даже к выходу микропроцессорной системы из строя. В программах, которые пишутся для микроконтроллеров, требуется обеспечить бесконечный цикл для того, чтобы программа никогда не завершала свою работу. Если основная программа микроконтроллера с бесконечным циклом будет располагаться до первой из подпрограмм, то приведенное условие будет выполняться автоматически.

Очень часто возникает необходимость из одной подпрограммы обращаться к другой подпрограмме. Такое обращение к подпрограмме называется вложенным вызовом подпрограммы. Количество вложенных подпрограмм называется уровнем вложенности подпрограмм. Максимально допустимый уровень вложенности подпрограмм определяется количеством ячеек памяти, предназначенных для хранения адресов возврата из подпрограмм.

Стек, его организация и структура

Адреса возврата из подпрограмм хранятся в области памяти, называемой *стеком*. Логически доступ к этим ячейкам памяти организован так, чтобы считывание последнего записанного адреса возврата производилось первым, а первого — последним. Логическая организация стека пояснена на рис. 21.3. Адресация ячеек стека осуществляется с использованием специального регистра, называемого *указателем стека*, SP. Ячейка памяти, в которую в данный момент может быть записан адрес возврата из подпрограммы, называется *вершиной стека*. Ее адрес всегда хранится в указателе стека. Количество ячеек памяти, выделенных для стека, называется *глубиной стека*. Последняя ячейка стека, в которую можно производить запись, называется *дном стека*.

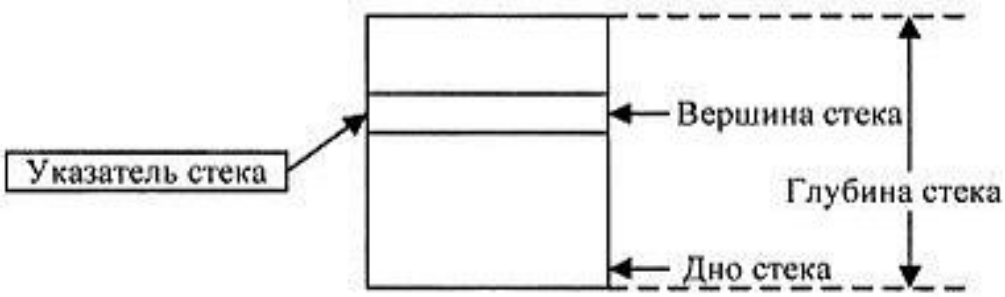


Рис. 21.3. Организация стека в памяти данных микропроцессора

Первоначально стек выполнялся аппаратно на отдельных ячейках памяти, затем его стали размещать в обычной памяти данных микропроцессоров. Это позволило в каждом конкретном случае устанавливать необходимую для программы глубину стека. Оставшуюся память можно использовать для размещения глобальных и локальных переменных программы. Глубина стека устанавливается при помощи записи начального адреса вершины стека в регистр SP. Обычно глубина стека устанавливается один раз после включения питания в процедуре инициализации контроллера.

Например, в микроконтроллерах семейства MCS-51 при занесении информации в стек содержимое указателя стека увеличивается (стек растет вверх), поэтому стек размещается в самой верхней части памяти данных. Для того чтобы установить глубину стека 28 байт, необходимо вычесть из адреса максимальной ячейки внутренней памяти микроконтроллера глубину стека и записать полученное значение в указатель стека SP:

```
DnoSteka EQU 127          ;для микроконтроллера AT89c51 размер
                          ;внутренней памяти равен 128 байтам

  MOV SP, #DnoSteka-28    ;Установить глубину стека 28 байт
```

Кроме значения программного счетчика, часто требуется запоминать содержимое внутренних регистров и флагов процессора, локальных переменных подпрограммы. Стек оказался удобным средством и для решения этих задач. Сохранение локальных переменных в стеке позволило осуществлять вызов подпрограммы самой из себя (реализовывать рекурсивные алгоритмы). Для работы со стеком в систему команд микропроцессоров введены специальные команды. У микроконтроллеров семейства MCS-51 это команды PUSH и POP. Их использование показано в листинге 21.2.

Листинг 21.2. Пример использования команд работы со стеком

```
Podprogramma:
  PUSH PSW        ;Сохранить содержимое
  PUSH ACC        ;используемых в
  PUSH R0         ;подпрограмме регистров

  ...             ;Текст подпрограммы

  POP R0          ;Восстановить содержимое
  POP ACC         ;используемых в
  POP PSW         ;подпрограмме регистров

  ret             ;Вернуться из подпрограммы
```

✧ *Обратите внимание*, что содержимое сохраненных в стеке регистров должно восстанавливаться в порядке, обратном порядку его запоминания.

Подпрограммы-процедуры и подпрограммы-функции

Подпрограммы предназначены для выполнения определенных действий над находящимися внутри микросхемы периферийными устройствами, внешними устройствами, подключенными к выводам микросхемы микроконтроллера, или числами, хранящимися в его внутренней памяти. В любом случае, с точки зрения программы, операции производятся над переменными. Переменные могут быть локальными (доступными только из подпрограммы) или глобальными (доступными из любого места программы).

Если подпрограмма осуществляет действия над глобальными переменными или выполняет определенный набор действий, то такая подпрограмма называется *процедурой*. Эта подпрограмма может осуществлять управление какими-то устройствами или осуществлять какие-либо вычисления. Если произ-

водятся вычисления, то результат помещается в глобальную переменную для того, чтобы этим результатом могла воспользоваться другая подпрограмма или основная программа. Пример фрагмента программы управления последовательным портом, написанного на языке высокого уровня С-51, приведен в листинге 21.3.

Листинг 21.3. Фрагмент программы управления последовательным портом

```
...
G_Per=56;          //Занести передаваемое число в глобальную переменную
PeredatByte();     //Передать это число
...
G_Per=37;          // Занести передаваемое число в глобальную переменную
PeredatByte();     //Передать это число
...

/*************************************************************
Подпрограмма передачи байта через последовательный порт
*************************************************************/
void PeredatByte(void)
{do;while(TI==0);      //Если предыдущий байт передан,
 SBUF=G_Per;           //то передать очередной байт
}
```

Часто подпрограмма должна выполнять действия над каким-либо числом, значение которого неизвестно в момент написания программы. Это число можно передать через глобальную переменную, как в приведенном выше примере подпрограммы. Однако намного удобнее использовать подпрограмму с параметрами.

Параметры подпрограммы — это локальные переменные подпрограммы, начальные значения которым присваиваются в вызывающей программе или подпрограмме. В алгоритмическом языке С-51 параметры подпрограммы записываются в скобках после ее имени. Пример вызова такой подпрограммы представлен в листинге 21.4.

Листинг 21.4. Пример вызова подпрограммы с параметром

```
...
PeredatByte(56);
...
PeredatByte(57);
...
```

```
/*********************************************************
Подпрограмма передачи байта через последовательный порт
*********************************************************/
void PeredatByte(char byte)
{do;while(TI==0);     //Если предыдущий байт передан,
 SBUF=byte;          //то передать очередной байт
}
```

Сравните с программой, использующей глобальные переменные (см. листинг 21.3). Как, по-вашему, какая из программ обладает большей наглядностью? В подпрограмму можно передавать и значительные объемы данных, например, строки или массивы данных:

```
PeredatStroky("Напечатать строку");
```

Естественно, что в этом случае сама подпрограмма `PeredatStroky` должна быть написана несколько иначе, чем в примерах листингов 21.3 или 21.4. Здесь потребуется применение переменных-указателей, которые будут рассмотрены позднее.

Часто требуется передавать результат вычислений из подпрограммы в основную программу. Для этого можно воспользоваться подпрограммой-функцией.

Подпрограмма-функция — это подпрограмма, которая возвращает вычисленное значение. Пример использования подпрограммы-функции:

$$Y = \sin(x)$$

Использование подпрограмм-функций позволяет приблизить текст программы к математической записи выражений, которые необходимо вычислить, а также увеличивает наглядность программ и, в результате, повышает скорость написания и отладки программного обеспечения.

Подпрограммы-функции обычно возвращают значения простых типов, таких как байт, слово или целое. Однако при помощи указателя можно возвращать и значения более сложных типов, таких как массивы переменных, структуры или строки.

Применение комментариев

В текст программы, наряду с операторами языка программирования, можно включать поясняющие комментарии, что служит улучшению наглядности программы и ее документированию. Комментарий может содержать любые печатные символы, а также пробелы и символы табуляции.

Комментарии применяются, например, для того, чтобы поставить в соответствие элементу блок-схемы программы или неформального описания алгоритма группу операторов языка программирования. Так как алгоритм может быть написан с различной степенью детализации, это должно быть отображено при помощи комментариев. Комментарии для частей описания алгоритма с наибольшей степенью детализации обычно пишут справа от операторов, которые реализуют эту часть алгоритма. Более крупные блоки алгоритма отражаются комментариями, занимающими в тексте программы отдельные строки. Эти комментарии, чтобы их легче было заметить, пишут буквами верхнего регистра. Один из таких блоков в листинге 21.5 выделен комментарием, названным комментарием алгоритма второго уровня. Еще более крупные блоки алгоритма выделяют специальными символами, которые сразу бросаются в глаза. Пример использования комментариев приведен в листинге 21.5.

Листинг 21.5. Пример использования комментариев

```
  ...
;ПЕРЕДАТЬ ДВА ЧИСЛА ЧЕРЕЗ ПОСЛЕДОВАТЕЛЬНЫЙ ПОРТ

  MOV G_Per, #56    ;Передать число 56 через
  CALL PeredatByte  ;последовательный порт

  MOV G_Per, #37    ;Передать число 37 через
  CALL PeredatByte  ;последовательный порт
;-----------------------------------------

  ...

;************************************************
;Подпрограмма передачи байта
;через последовательный порт
;************************************************
PeredatByte:
  JB TI,$           ;Если предыдущий байт выдан,
  MOV SBUF,G_Per    ;то передать следующий байт
  RET
```

КОММЕНТАРИЙ АЛГОРИТМА ВТОРОГО УРОВНЯ

КОММЕНТАРИЙ, ПОЯСНЯЮЩИЙ НЕСКОЛЬКО ОПЕРАТОРОВ

КОНЕЦ БЛОКА АЛГОРИТМА ВТОРОГО УРОВНЯ

ТАК ВЫДЕЛЯЮТСЯ ОСОБО ВАЖНЫЕ БЛОКИ АЛГОРИТМА

КОММЕНТАРИЙ, ПОЯСНЯЮЩИЙ ДЕЙСТВИЕ КОМАНДЫ В ДАННОМ КОНКРЕТНОМ СЛУЧАЕ

В листинге 21.5 показан отрывок программы на языке ассемблера ASM-51, для которого комментарии имеют наибольшее значение, но точно так же можно (и нужно) использовать комментарии и в исходных текстах программы на языке высокого уровня. При этом нужно помнить, что те фрагменты

текста программы, которые абсолютно ясны в момент написания, через месяц вызовут затруднения даже у программиста, написавшего этот текст, не говоря уже о человеке, который видит его впервые.

Программа читается, прежде всего, по комментариям и только потом, если она по каким-либо причинам не работает, проверяется на соответствие комментариям конкретных операторов языка программирования.

Такой стиль написания программ позволяет значительно экономить время, т. к. не приходится повторно разбираться с уже написанными участками кода при поиске ошибки программы.

Структурное программирование

Программирование для универсальных компьютеров начиналось с использования машинных кодов, затем появились языки высокого уровня. Позже были развиты сначала принципы структурного программирования, а потом — объектно-ориентированное программирование. В настоящее время активно развивается визуальное программирование.

Программирование для микроконтроллеров во многом повторяет тот же путь. Переход от этапа к этапу зависит от доступных внутренних ресурсов микроконтроллеров. Еще несколько лет назад использование языков высокого уровня при написании программ для микроконтроллеров было невозможно из-за малого объема внутренней памяти программ. (В дешевых моделях микроконтроллеров эта ситуация сохраняется до сих пор.) В настоящее время с появлением микроконтроллеров и сигнальных процессоров с объемом внутренней памяти в несколько десятков килобайт появляется возможность использования методов структурного, а в некоторых случаях и объектно-ориентированного проектирования.

Применение структурного программирования позволяет увеличить скорость написания программ и облегчить их отладку за счет сокращения количества доступных программных конструкций. Поэтому в начале семидесятых годов был разработан ряд языков высокого уровня, ориентированных на структурное программирование. Это такие языки, как C, PASCAL, ADA. Однако не надо думать, что структурное программирование возможно только на этих языках, для этого пригодны и такие языки, как ассемблер или FORTRAN, где не предусмотрено структурных операторов.

В настоящее время существует два способа написания программ: снизу вверх и сверху вниз. При написании программы снизу вверх приступить к ее отладке невозможно, не завершив полностью разработку текста всей программы. При этом ошибка в написании (или понимании работы) крупных блоков программы приводит к тому, что даже правильно написанные части программы

приходится переделывать или выбрасывать полностью! Но наиболее неприятный момент заключается в том, что при таком методе программирования в программе содержится огромное количество ошибок, каждая из которых может привести к неработоспособности разрабатываемого устройства (мы помним, что в микроконтроллерах именно программа во многом определяет работу устройства).

И еще одна особенность написания и отладки программы для микроконтроллеров и сигнальных процессоров. В отличие от универсальных компьютеров никто не гарантирует, что правильно работает аппаратура! Возможна ситуация, что устройство не работает не из-за ошибки в программе, а из-за ошибки в схеме или ошибки монтажа! В процессе написания и отладки программы для микроконтроллерного устройства производится поиск ошибок не только в программном обеспечении, но и в схеме.

При разработке программы сверху вниз на любом этапе она может быть оттранслирована и выполнена, при этом можно отследить выполнение всех фрагментов алгоритма, написанных к этому времени. Более того, разработку алгоритма можно объединить с его реализацией на языке программирования! Для этого можно воспользоваться подпрограммами. Выполняемое алгоритмическое действие обычно отображается в названии подпрограммы. Это значительно повышает наглядность программы и тем самым скорость ее написания и отладки. Пример такого подхода показан в листинге 21.6.

Листинг 21.6. Пример программы, в которой отдельным элементам алгоритма соответствуют подпрограммы

```
/*****************************************
Подпрограмма чтения порта
*****************************************/
void ProchitatPort(void)
{//Пока это только подпрограмма-заглушка!
}
/*****************************************
Подпрограмма включения индикатора
*****************************************/
void VklychitIndikator(void)
{//Пока это только подпрограмма-заглушка!
}

void main(void)
{ProchitatPort ();          //Прочитать порт
 VklychitIndikator ();      //Включить индикатор
}
```

В листинге 21.6 текст программы практически не отличается от описания алгоритма. В начале процесса разработки программного обеспечения подпрограммы обычно еще не написаны. Для того чтобы можно было оттранслировать программу, можно воспользоваться механизмом заглушек. Подпрограмма-заглушка ничего не делает, но уже объявлена и имеет имя, совпадающее с тем действием, которое она должна будет в дальнейшем реализовать.

Подпрограммы будут написана позднее, когда уже будет отлажен верхний уровень программы. При написании подпрограмм уже не нужно будет заботиться о верхнем уровне, т. к. он уже к этому времени будет отлажен. Не потребуется также учитывать остальные подпрограммы, т. к. подпрограммы одного уровня не зависят друг от друга.

Для улучшения восприятия текста программы, кроме "говорящих" названий подпрограмм, широко используются "говорящие" имена переменных. Особенно улучшается читаемость программы в том случае, если имена этих переменных связаны с аппаратурой микроконтроллерного устройства.

Например, для микроконтроллера at89c51 можно назначить флагу завершения приема по последовательному порту `RI` имя `BytePeredan` (байт передан). Тогда участок программы на языке программирования С-51, соответствующий ожиданию завершения приема байта последовательным портом, будет выглядеть следующим образом:

```
if (BytePeredan) SBUF = SledujushchiyByte;
```

При таком выборе имени переменной исходный текст программы практически совпадает с фрагментом алгоритма, который этот участок программы реализует! В этом случае можно даже отказаться от применения комментария для пояснения фрагмента программы (SledujushchiyByte — это записанная транслитом фраза "следующий байт").

Применение подпрограмм является не только средством структурирования программы, но и существенно сокращает размер машинного кода, если подпрограммы многократно используются. Однако необходимо отметить, что при использовании подпрограмм несколько уменьшается быстродействие устройства. В случае, если для разрабатываемого устройства это недопустимо, вместо подпрограмм можно использовать макросы, в названии которых точно так же будет отображаться выполняемое действие. Макрос, в отличие от подпрограммы, не передает управление в отдельную область памяти с последующим возвратом к следующему за вызовом подпрограммы оператору, а вызывает повторную подстановку одних и тех же команд столько раз, сколько в тексте программы встретилось имя макроса. Для образования макроса в программе на языке программирования С-51 достаточно объявить подпрограмму с префиксом `inline`.

Основная идея структурного программирования заключается в том, что существует только четыре управляющих конструкции. При использовании этих структурных управляющих конструкций можно построить сколь угодно сложную программу, не изобретая какие бы то ни было дополнительные программные конструкции.

Линейная цепочка операторов

Первая управляющая конструкция это *линейная цепочка операторов*. Она довольно часто используется, поскольку многие задачи могут быть разбиты на несколько более простых последовательно выполняемых подзадач. Такой подход достаточно очевиден и применялся с самого начала программирования. Недостаток прямого использования такого подхода — это то, что пока не написана вся программа целиком, ее нельзя отлаживать. В результате отлаживаемая программа получается сложной и в ней трудно найти ошибки.

При использовании линейной цепочки операторов выполнение подзадач может быть поручено подпрограмме, в названии которой можно (и нужно) отразить подзадачу, которую должна решать эта подпрограмма. При этом вместо настоящих программ имеет смысл использовать подпрограммы-заглушки, принцип работы с которыми был рассмотрен ранее.

Блок-схема стандартной структурной конструкции управления "линейная цепочка операторов" приведен на рис. 21.4. Поясним процесс преобразования этой блок-схемы в исходный текст программы.

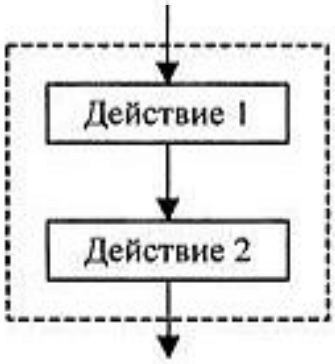


Рис. 21.4. Блок-схема конструкции управления "линейная цепочка операторов"

На момент написания алгоритма (и программы) верхнего уровня нам не известны детали их реализации, поэтому вместо настоящих подпрограмм, соответствующих элементам блок-схемы "Действие 1" и "Действие 2", поставим подпрограммы-заглушки, которые ничего не делают. Однако в именах подпрограмм отразим те алгоритмические действия, которые они должны осуществлять в дальнейшем. Для отладки программы (и алгоритма) того уровня,

который пишется в данный момент, неважно, что подпрограммы ничего не делают. Главное, что мы можем проверить, каков порядок вызовов подпрограмм и соответствует ли он ожидаемому. На этом же этапе можно отладить взаимодействие между программами.

Для этого мы можем произвести трансляцию программы и на отладчике проследить все действия программы. Возможность производить трансляцию и отладку программы на любом этапе ее написания позволяет находить ошибки сразу же после написания оператора! Ведь если мы написали только один оператор, то и совершить ошибку мы могли только в этом месте. То есть процесс поиска ошибки существенно упрощается. А ведь процесс написания программы на 90% состоит из поиска ошибок.

После завершения отладки программы (и алгоритма!) текущего уровня можно перейти к написанию и отладке любой из подпрограмм, т. е. к превращению подпрограммы-заглушки в нормальную отлаженную программу. При этом очень важно заметить, что подпрограммы отлаживаются независимо друг от друга, что значительно облегчает написание программы (да и чтение этой программы в дальнейшем).

Пример использования подпрограмм-заглушек для реализации конструкции управления "линейная цепочка операторов" на языке программирования С-51 приведен в листинге 21.7, а на языке программирования ASM-51 — в листинге 21.8.

Листинг 21.7. Конструкция управления "линейная цепочка операторов" на языке программирования С-51

```
/*********************************************
Определения подпрограмм
*********************************************/
void Deistviel(void)
{//Пока это только подпрограмма-заглушка!
}

void Deistvie2(void)
{//Пока это только подпрограмма-заглушка!
}

/*********************************************
Главная программа
*********************************************/
  Deistviel();//Подпрограмма 1
  Deistvie2();//Подпрограмма 2
```

Как видно из приведенного в листинге 21.7 исходного текста программы, для организации подпрограммы-заглушки на языке программирования C-51 достаточно определить подпрограмму, не помещая в ее тело ни одного оператора. При этом объявление подпрограммы-заглушки должно быть точно таким же, как у подпрограммы в готовой написанной и отлаженной программе.

Листинг 21.8. Конструкция управления "линейная цепочка операторов" на языке программирования ASM-51

```
;*****************************************************
;Главная программа (с обязательным бесконечным циклом)
;*****************************************************
  ...
  call Deistvie1  ;Подпрограмма 1
  call Deistvie2  ;Подпрограмма 2
  ...
;*****************************************************
;Определения подпрограмм
;*****************************************************
;--Пока это только подпрограмма-заглушка!------------
Deistvie1:
  ret

;--Пока это только подпрограмма-заглушка!------------
Deistvie2:
  ret
```

На языке программирования ASM-51 имя подпрограммы совпадает с меткой в начале подпрограммы. Однако, в отличие от языков программирования высокого уровня, для организации подпрограммы-заглушки один исполняемый оператор все-таки нужен. Это оператор возвращения из подпрограммы `ret`. И еще одно замечание. При программировании на языке ассемблера первый написанный оператор и выполняется первым. Для того чтобы исключить возможность случайного попадания в подпрограмму не по вызову подпрограммы, подпрограммы обычно располагаются в конце исходного текста программы, за обязательным бесконечным циклом основной программы. Иногда подпрограммы все же располагаются до основной программы, сразу за векторами прерывания, но в этом случае все подпрограммы обходятся при помощи команды безусловного перехода LJMP.

Условное выполнение операторов

Вторая конструкция управления называется "*условным выполнением операторов*". Достаточно часто одно или другое действие должно исполняться в

зависимости от определенного условия, которое зависит от результатов выполнения предыдущей части программы или от состояния (сигналов) внешних устройств. Блок-схема конструкции управления "условное выполнение операторов" приведена на рис. 21.5.

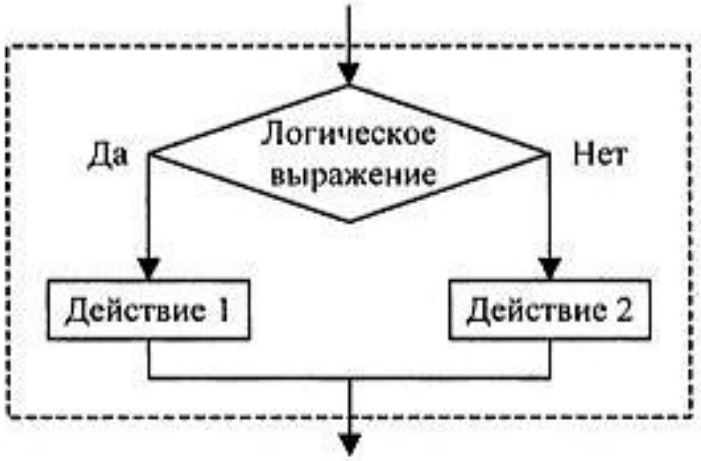


Рис. 21.5. Блок-схема конструкции управления "условное выполнение операторов"

Как видно из приведенной в листинге 21.9 программы на языке С-51, конструкция условного выполнения операторов реализуется благодаря встроенным средствам самого языка С-51, т. к. он относится к структурированным языкам программирования. В приведенном примере подпрограммы-заглушки использованы для отображения названий частей условного оператора.

В структурированных языках программирования второе плечо условного оператора записывается после зарезервированного слова "else". В случае необходимости использования в любом из плеч нескольких операторов, применяется структурный оператор "линейная цепочка операторов", реализация которого была описана ранее.

Листинг 21.9. Реализация конструкции управления "условное выполнение операторов" средствами языка С-51

```
/*********************************************
Определения подпрограмм
*********************************************/
void Deistviel(void)
{//Пока это только подпрограмма-заглушка!
}

void Deistvie2(void)
{//Пока это только подпрограмма-заглушка!
}
```

```
/*****************************************
Главная программа
*****************************************/
  ...
  if(PrinjatByte)//Выражение
     Deistviel();//Реализация ветви 1
    else
     Deistvie2();//Реализация ветви 2
  ...
```

Пример реализации конструкции управления "условное выполнение операторов" на языке программирования ASM-51 приведен в листинге 21.10. Язык программирования ASM-51 не является структурированным, поэтому для реализации конструкции условного выполнения операторов приходится использовать несколько команд микропроцессора (операторов языка программирования ассемблер).

Для реализации блока "логическое выражение", изображенного на рис. 21.5 в виде ромба, можно использовать любую команду условного перехода, входящую в систему команд микропроцессора. При этом в простейшем случае логическое выражение сводится к анализу сигналов на выводах микроконтроллера или его внутренних флагов. Но в отличие от двухмерной блок-схемы, память программ микропроцессора одномерна, т. е. все команды располагаются друг за другом. Для того чтобы выполнялся только один из операторов (Действие 1 или Действие 2), необходимо после выполнения одного из них обойти другой. Это можно осуществить при помощи команды безусловного перехода.

Размещение различных частей конструкции управления "условное выполнение операторов" в памяти программ микропроцессора приведено на рис. 21.6. Направления возможных переходов при ее выполнении показано на этом же рисунке. При использовании такой конструкции будет выполнена только один из операторов. Какой — зависит от результатов выполнения условного выражения.

Листинг 21.10. Реализация конструкции управления "условное выполнение операторов" средствами языка ASM-51

```
;*********************************************************
;Главная программа
;*********************************************************
  ...
;----------Условное выполнение операторов-----------------
  jb PrinjatByte,ElseUsl;Условное выражение
    Call Deistviel          ;Реализация
    jmp EndUsl             ;первой ветви
```

```
ElseUsl:
    Call Deistvie2          ;Реализация ветви 2
EndUsl:;-------------------------------------
  ...

;*****************************************************************
;Определения подпрограмм
;*****************************************************************
;--Пока это только подпрограмма-заглушка!----------------------
 Deistvie1:;Метка относится к следующему оператору
  ret

;--Пока это только подпрограмма-заглушка!----------------------
 Deistvie2:;Метка относится к следующему оператору
  ret
```

В листинге. 21.10 проявляется еще одно полезное свойство подпрограмм. Команды условного перехода, используемые для реализации условного выражения, могут передавать управление на участок программы, отстоящий от них не более чем на 127 байтов, однако для реализации алгоритма одной из ветвей может потребоваться большее количество команд. Выделение их в отдельную подпрограмму позволяет сократить необходимое расстояние условного перехода до трех байт (длины команды вызова подпрограммы LCALL).

Рис. 21.6. Размещение различных частей конструкции управления условным выполнением операторов в памяти программ микропроцессора

Подпрограмма может быть размещена в любом месте памяти программ микроконтроллера и содержать произвольное количество машинных команд.

Конструкция условного выполнения операторов может использоваться в неполном варианте, когда присутствует только один оператор. Для ее реализации не обязательно использовать команду безусловного перехода. Она легко реализуется одной командой условного перехода, входящей в состав системы команд любого микропроцессора (а значит, и языка программирования ассемблер для него). Однако если система команд микропроцессора неполная (например, есть команда проверки на равенство, но нет проверки на неравенство), то для реализации противоположного оператора тоже может потребоваться команда безусловного перехода.

Блок-схема и примеры реализации конструкции управления условным выполнением одного оператора на языках программирования C-51 и ASM-51 приведены на рис. 21.7 и в листингах 21.11 и 21.12. Каждый из исходных текстов содержит очень подробные комментарии, поэтому дополнительные пояснения не понадобятся.

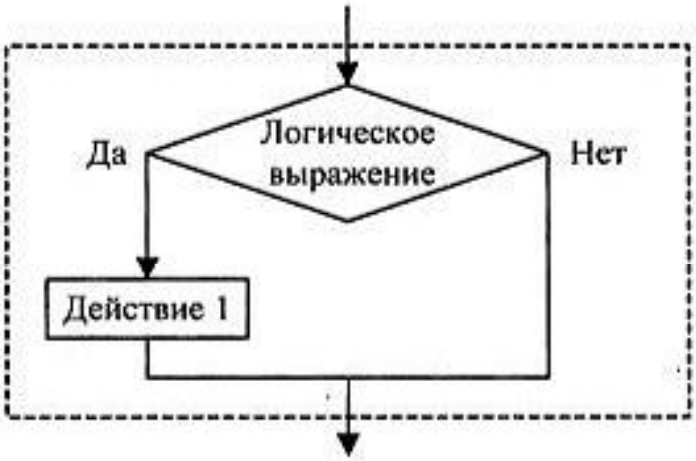


Рис. 21.7. Блок-схема конструкции управления условным выполнением одного оператора

Листинг 21.11. Реализация конструкции управления условным выполнением одного оператора средствами языка C-51

```
/*********************************************
Определения подпрограмм
*********************************************/
void Deistvie1(void)
{//Пока это только подпрограмма-заглушка!
}
```

```
/*******************************************
Главная программа
*******************************************/
  ...
  if(PrinjatByte)//Условное выражение
      Deistviel();     //Реализация ветви
  ...
```

Листинг 21.12. Реализация конструкции управления условным выполнением одного оператора средствами языка ASM-51

```
;*****************************************************************
;Главная программа
;*****************************************************************
...
;--конструкция управления условным выполнением одного оператора----
  jb PrinjatByte, EndUsl;Условное выражение
      call Operator1          ; Реализация ветви
EndUsl:;---------------------------------------------------------------
  ...
;*****************************************************************
; Определения подпрограмм
;*****************************************************************
;-- Пока это только подпрограмма-заглушка!-------------------------
Operator1:; Метка относится к следующему исполняемому оператору
  ret
```

Конструкция управления циклическим выполнением оператора с проверкой условия после тела цикла

Третья конструкция управления — это *циклическое выполнение оператора с проверкой условия после тела цикла*. Оператор (или операторы), который должен повторяться в процессе выполнения этой конструкции, называется телом цикла. В процессе выполнения этих операторов обычно модифицируется некоторая переменная, значение которой влияет на завершение цикла. Эта переменная получила название параметр цикла. Отметим, что параметр цикла может изменяться и аппаратурой, подключенной к микроконтроллеру или входящей в его состав.

Блок-схема конструкции управления циклическим выполнением оператора с проверкой условия после тела цикла приведена на рис. 21.8. Напомню, что в качестве тела цикла можно использовать любую структурную конструкцию, в том числе и еще один оператор цикла.

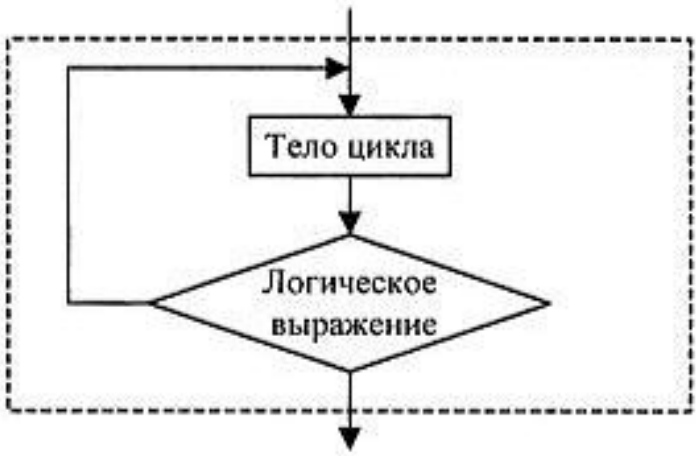


Рис. 21.8. Блок-схема конструкции управления циклическим выполнением оператора с проверкой условия после тела цикла

На языках программирования высокого уровня такая конструкция входит в состав языка (оператор `do ... while` в языке программирования C или оператор `repeat ... until` в языке программирования Pascal). Пример использования циклического выполнения оператора с проверкой условия после тела цикла на языке программирования C-51 представлен в листинге 21.13. Обратите внимание, что в приведенном примере в качестве логического выражения использована константа — единица. Это приводит к бесконечному циклу, который необходим для реализации управляющих программ микроконтроллера. Применение оператора `while(1);` эквивалентно безусловной передаче управления на начало тела цикла при помощи ассемблерной команды `jmp`.

Листинг 21.13. Реализация средствами языка программирования C циклического выполнения оператора с проверкой условия после тела цикла

```
/*********************************************
Определения подпрограмм
*********************************************/
void TeloCikla(void)
{//Пока это только подпрограмма-заглушка!
}

/*********************************************
Главная программа
*********************************************/
  ...
  do
    TeloCikla();//Реализация тела цикла
  while(1);
  ...
```

Не сложнее реализуется эта структурная конструкция управления и на языке программирования ассемблер при помощи команды условного или безусловного (для организации бесконечного цикла) перехода. Реализация конструкции цикла с проверкой условия после тела цикла очень похожа на реализацию конструкции условного выполнения оператора. Отличие заключается в том, что передача управления в команде условного перехода осуществляется не вперед, как при условном выполнении оператора, а назад, на начало цикла. В системе команд микроконтроллеров семейства MCS-51 для реализации цикла с проверкой условия после тела цикла введена специальная команда — `djnz`. Она позволяет реализовать сразу два алгоритмических действия: вычитания единицы из параметра цикла и проверку содержимого параметра цикла на равенство нулю. Для обозначения начала тела цикла в программе, написанной на языке программирования ассемблер, применяется метка.

Пример реализации оператора цикла с проверкой условия после тела цикла на языке программирования ASM-51 приведен в листинге 21.14. В этом примере предполагается опрос бита завершения приема байта последовательным портом RI, который объявлен где-то в тексте программы как переменная `PrinjatByte` (означает "принять байт").

Листинг 21.14. Пример реализации цикла с проверкой условия после тела цикла на языке программирования ASM-51

```
;*****************************************************************
;Главная программа
;*****************************************************************
  ...
Nachalo:;- конструкция управления циклическим выполнением оператора------
    call TeloCikla          ;Реализация тела цикла
  jb PrinjatByte,Nachalo;Условное логическое выражение
;----------------------------------------------------------------------
  ...

;*****************************************************************
;Определения подпрограмм
;*****************************************************************
;--Пока это только подпрограмма-заглушка! -----------------------------
TeloCikla:
  ret    ;Оператор помечен предыдущей меткой
```

Структурная конструкция циклического выполнения оператора с проверкой условия до тела цикла

Четвертая структурная конструкция управления — это *цикл с проверкой условия до тела цикла*. В отличие от предыдущей конструкции оператора, тело цикла в этом операторе может ни разу не выполниться, если условие завершения цикла сразу же выполнено. Блок-схема цикла с проверкой условия до тела цикла приведена на рис. 21.9.

Конструкция цикла с проверкой условия до тела цикла может быть реализована при помощи средств, входящих в состав языка программирования С-51, как впрочем и других языков, предназначенных для разработки структурированных программ.

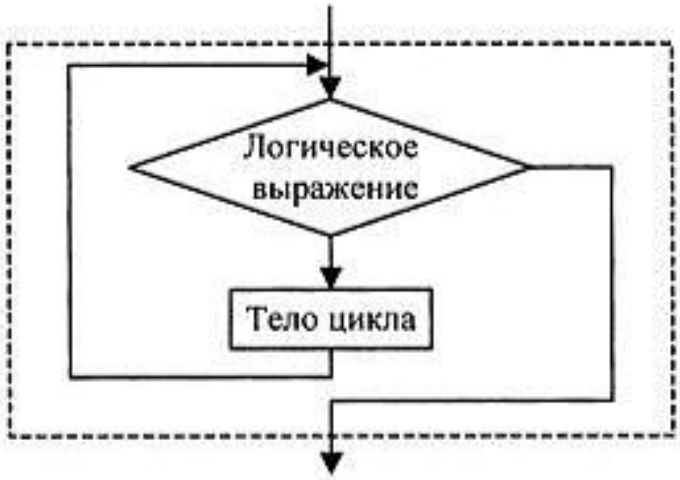


Рис. 21.9. Блок-схема структурной конструкции управления "циклическое выполнение оператора с проверкой условия до тела цикла"

Пример использования цикла с проверкой условия до тела цикла в программе, написанной на языке программирования С-51, приведен в листинге 21.15. Как и в предыдущих случаях, для иллюстрации возможности повысить наглядности текста программы за счет использования правильно выбранных названий подпрограмм функция, выполняемая в теле цикла, названа `TeloCikla`.

Листинг 21.15. Пример использования цикла с проверкой условия до тела цикла в программе на языке программирования С-51

```
/*********************************************
Определения подпрограмм
*********************************************/
void TeloCikla(void)
{//Пока это только подпрограмма-заглушка!
}
```

```
/*******************************************
Главная программа
*******************************************/
  ...
  while(KnNaj)
    TeloCikla();//Реализация тела цикла
  ...
```

Пример реализации цикла на языке программирования ASM-51 приведен в листинге 21.16. Эту структурную конструкцию, как и условное выполнение операторов, невозможно реализовать при помощи одной машинной команды, поэтому реализуем его при помощи команд условного и безусловного перехода. На этот раз команда безусловного перехода помещается в конец конструкции и осуществляет переход на команду проверки условия, т. е. на начало конструкции. Команда безусловного перехода `sjmp` передает управления на начало цикла после выполнения его тела. Расположенная за командой безусловного перехода метка `KonCykla` обозначает команду, на которую передается управление, когда прекращается выполнение цикла.

Листинг 21.16. Пример использования цикла с проверкой условия до тела цикла в программе на языке программирования ASM-51

```
;*****************************************************************
;Главная программа
;*****************************************************************
  ...
Nachalo:;-------- Оператор цикла ---------------------------------
  jb KnNaj,KonCikla     ;Проверка условия завершения цикла
    Call TeloCikla        ;Тело цикла
    sjmp Nachalo
KonCikla:;-------------------------------------------------------
  ...

;*****************************************************************
;Определения подпрограмм
;*****************************************************************
;--Пока это только подпрограмма-заглушка! ----------------------
TeloCikla:
  ret   ;Оператор помечен предыдущей меткой
```

Понятие многофайлового и многомодульного программирования

В процессе написания программ обычно накапливаются подпрограммы и фрагменты кода, которые можно использовать в нескольких программах. Фрагменты исходного кода можно копировать из программы в программу при помощи текстового редактора, в котором вы пишете программы, однако это может привести к некоторым неудобствам. Прежде всего, разрастается текст программы, и в нем становится трудно ориентироваться при написании и редактировании программы. Кроме того, при обнаружении ошибок в отлаженном ранее участке кода или при переходе к работе с другими устройствами (от светодиодного индикатора — к жидкокристаллическому, от микросхемы АЦП одного типа — к микросхеме другого типа) приходится искать включенные ранее участки кода и заменять их новыми.

Это трудоемкая работа, которая приводит к ошибкам и, в конечном счете, замедляет написание и отладку программ. Но как же решить эту проблему? Намного удобнее использовать и хранить исходный текст программы в нескольких файлах, предоставляя работу по соединению этих файлов в единую программу транслятору. Обычно в один файл помещается участок кода, работающий с каким-либо конкретным устройством (жидкокристаллическим индикатором, последовательной микросхемой ПЗУ и т. д.) или отвечающий за определенную задачу (вывод цифровой и текстовой информации, помехоустойчивое кодирование, прием управляющих сигналов).

Использование готовых участков кодов позволяет собирать новые программы из готовых кусков как в детском конструкторе. Правда, не стоит надеяться, что эти куски будут так же легко соединяться!

Многофайловые программы

Самым простым способом соединения нескольких файлов в одну программу является использование директивы включения текстового файла. В языке программирования ASM-51 это директива `include`. Точно так же, только буквами нижнего регистра, записывается эта директива и в языке программирования C.

При использовании директивы `include` в исходный текст программы добавляется содержимое включаемого файла, и только после этого производится трансляция исходного текста в исполняемый код программы. Иными словами, содержимое главного файла программы и включаемых в него файлов объединяются препроцессором транслятора во временном файле, и только после этого производится трансляция полученного временного файла в исполняемый код микроконтроллера. Пример использования директивы вклю-

чения текстового файла в программе на языке программирования C-51 приведен в листинге 21.17.

Листинг 21.17. Пример программы на языке C-51, использующей директивы include

```
#include <global.h> /* Добавление файла определений
                       переменных, связанных с выводами микросхемы*/
#include <reg51.h>  /* Добавление файла описаний
                       регистров специальных функций микроконтроллера */
#include <IO.h>     /*Подключить файл с подпрограммами, осуществляющими
                      ввод и вывод данных в микросхему микроконтроллера*/
... /* Остальная часть программы */
```

В отдельные файлы выделяются, как правило, описания внутренних регистров микроконтроллера и переменных, связанных с выводами микросхемы микроконтроллера. Рассмотрим для примера содержимое включаемых файлов, имена которых использованы в примере, приведенном в листинге 21.17. Фрагменты исходных текстов этих файлов представлены в листингах 21.18 и 21.19.

Содержимое файла `IO.h` является примером использования отдельного файла для хранения функций, осуществляющих ввод и вывод данных. Такое использование включаемых файлов позволяет разделить программу по функциям. В результате новые программы можно собирать из таких файлов, как из кирпичиков, используя готовые уже отлаженные подпрограммы. Таким образом, использование включаемых файлов резко упрощает программирование.

Листинг 21.18. Фрагмент исходного текста, содержащегося в файле `IO.h`

```
void UART (void) interrupt 4
{register char tmp;
 if(RI)
   {RI=0;
    tmp=SBUF;
    *ptrBufUART++=tmp;
   }
    if(tmp==10)
      {msgRsvd=1;
       *ptrBufUART=0;
       return;
      }
}
```

```
/*************************************************************************
Подпрограмма преобразования тетрады в символ
*************************************************************************/
char Hex(char Nible)
{Nible&=0x0f;
 if(Nible>=10)return(Nible+='7');
 return(Nible+='0');
}

/*************************************************************************
Подпрограмма преобразования слова управления устройством в строку,
завершенную нулем
*************************************************************************/
void LongToHex(char data *s,void data *Byte,unsigned char i)
{unsigned char tmp;
 do{tmp=*(char data*)Byte;
    Byte+=1;
    s++ =Hex(tmp>>4);
    *s++ =Hex(tmp);
 }while(--i!=0);
 *s =0;
}
```

В файле REG51.h объявляются переменные, связанные с регистрами специального назначения микроконтроллера 89c51. Они должны использоваться в любой программе, работающей с микроконтроллером 89c51. Не имеет смысла повторять эти объявления в каждой программе — их можно выделить в отдельный файл, что и сделано в приведенном в листинге 7.19 примере. Более того, файлы объявления регистров специальных функций для различных типов микроконтроллеров обычно поставляются разработчиками компиляторов для исключения ошибок.

Листинг 21.19. Фрагмент исходного кода, содержащегося в файле REG51.h

```
/*-----------------------------------------------------------------
REG51.H
Файл объявления внутренних регистров для микроконтроллеров 80C51 и 80C31.
-----------------------------------------------------------------*/

/*  BYTE Register  */
sfr P0   = 0x80;
sfr P1   = 0x90;
sfr P2   = 0xA0;
sfr P3   = 0xB0;
```

```
sfr PSW  = 0xD0;
sfr ACC  = 0xE0;
sfr B    = 0xF0;
sfr SP   = 0x81;
sfr DPL  = 0x82;
sfr DPH  = 0x83;
sfr PCON = 0x87;
sfr TCON = 0x88;
sfr TMOD = 0x89;
sfr TL0  = 0x8A;
sfr TL1  = 0x8B;
sfr TH0  = 0x8C;
sfr TH1  = 0x8D;
sfr IE   = 0xA8;
sfr IP   = 0xB8;
sfr SCON = 0x98;
sfr SBUF = 0x99;
```

Файл объектного кода программы, который будет преобразован в машинные коды микроконтроллера, формируется программой-транслятором в процессе обработки исходного текста программы. Как уже упоминалось ранее, прежде чем начать собственно преобразование исходного текста программы, транслятор производит предварительную обработку программы, в процессе которой выполняются директивы условной трансляции, включения содержимого дополнительных файлов и макроподстановок. Часто говорят, что перед трансляцией программы работает препроцессор.

В результате этой предварительной обработки программы транслятор формирует временный файл. Например, в результате выполнения участка программы, приведенного в листинге 21.17, будет сформирован временный файл, содержание которого представлено в листинге 21.20. Именно этот файл и будет преобразован в машинные коды микроконтроллера, которые будут сохранены на жестком диске компьютера в виде загрузочного файла.

Листинг 21.20. Временный файл, полученный в результате работы препроцессора

```
/*-------------------------------------------------------------------
IO.H
-------------------------------------------------------------------*/
void UART (void) interrupt 4
{register char tmp;
 if(RI)
   {RI=0;
```

```
    tmp=SBUF;
    *ptrBufUART++=tmp;
   }
    if(tmp==10)
      {msgRsvd=1;
       *ptrBufUART=0;
       return;
      }
}

/*************************************************************************
Подпрограмма преобразования тетрады в символ
*************************************************************************/
char Hex(char Nible)
{Nible&=0x0f;
 if(Nible>=10)return(Nible+='7');
 return(Nible+='0');
}

/*************************************************************************
Подпрограмма преобразования слова управления устройством в строку,
завершенную нулем
*************************************************************************/
void LongToHex(char data *s, void data *Byte, unsigned char i)
{unsigned char tmp;
 do{tmp=*(char data*)Byte;
    Byte+=1;
    *s++ =Hex(tmp>>4);
    *s++ =Hex(tmp);
 }while(--i!=0);
 *s =0;
}

/*-----------------------------------------------------------------------
REG51.H
Файл объявления внутренних регистров для микроконтроллеров 80C51 и 80C31.
-----------------------------------------------------------------------*/

/*  BYTE Register  */
sfr P0   = 0x80;
sfr P1   = 0x90;
sfr P2   = 0xA0;
sfr P3   = 0xB0;
sfr PSW  = 0xD0;
sfr ACC  = 0xE0;
```

```
sfr B    = 0xF0;
sfr SP   = 0x81;
sfr DPL  = 0x82;
sfr DPH  = 0x83;
sfr PCON = 0x87;
sfr TCON = 0x88;
sfr TMOD = 0x89;
sfr TL0  = 0x8A;
sfr TL1  = 0x8B;
sfr TH0  = 0x8C;
sfr TH1  = 0x8D;
sfr IE   = 0xA8;
sfr IP   = 0xB8;
sfr SCON = 0x98;
sfr SBUF = 0x99;
```

Подведем итоги. При многофайловом программировании:

- использование нескольких файлов позволяет разбить исходный текст программы на несколько частей, каждая из которых реализует свою независимую задачу;
- удобнее всего в отдельные файлы выносить подпрограммы, которые должны быть построены таким образом, чтобы их связь с основной программой была минимальной;
- разбираться с короткими файлами, реализующими одну или несколько связанных между собой функций, намного легче, чем работать с одним большим файлом;
- различные части программы могут быть написаны разными программистами, которым намного легче работать со своей программой, оформленной в виде отдельного файла.

Разбивать на несколько файлов можно не только программы, исходный код которых разработан на языках высокого уровня. Включение файлов можно использовать и в программах на языке ассемблера. В листинге 21.21 приведен пример использования включения файлов в программе на языке ASM-51.

Листинг 21.21. Включение файлов в программе на ASM-51

```
$include (stdio.asm)   ; Включение файла
                       ; с функциями стандартного ввода-вывода
$include (reg51.inc)   ; Включение файла с описаниями
                       ; регистров специальных функций микроконтроллера

  ...                  ;Остальная часть программы
```

В результате работы программы-транслятора, точно так же, как и в предыдущем примере, будет сформирован временный файл, в который будет помещена информация из обоих включаемых файлов. И только затем будет осуществлена трансляция этого временного файла.

С разбиением программы на несколько файлов связано понятие программного проекта. В состав программного проекта включают все файлы, содержащие исходный текст программы. Обычно программный проект размещают в отдельном каталоге.

Кроме исходных текстов программы, в состав проекта входит загрузочный файл, в который помещаются машинные коды микроконтроллера, полученные в результате трансляции программы. Очень часто они хранятся в файле не непосредственно в двоичном виде, а в специальном формате, который позволяет исключить ошибочную запись неправильного кода в микроконтроллер. Наибольшее распространение получил HEX-формат загрузочного файла.

Еще один файл, который обычно включается в состав программного проекта, — это файл листинга компилятора, в который помещается исходный текст программы, машинные коды в текстовом виде и сообщения об ошибках. Этот файл облегчает поиск и исправление синтаксических ошибок.

Многомодульные программы

Разбиение исходного текста программы на несколько файлов делает его более понятным для программистов, участвующих в создании программного продукта. Однако при использовании этих файлов при помощи директив включения остаются нерешенными еще несколько задач:

- программа-транслятор работает со всем исходным текстом целиком, ведь она соединяет все файлы перед трансляцией вместе. Поэтому время трансляции исходного текста программы получается значительным. В то же самое время программа никогда не переписывается целиком. Обычно изменяется только небольшой участок программы. В связи с этим значительные затраты времени на компиляцию представляются необоснованно большими;
- максимальное количество имен переменных и меток бывает ограничено программой-транслятором и может быть исчерпано при написании исходного текста программы, если он достаточно велик;
- различные программисты, участвующие в создании программного продукта, могут назначать одинаковые имена для своих переменных, и при попытке соединения таких файлов в единую программу обычно возникают проблемы.

Все эти проблемы могут быть решены при раздельной трансляции программы. То есть, было бы неплохо уметь транслировать каждый файл с исходным текстом части программы отдельно и соединять затем готовые оттранслированные участки в конечную программу.

Компиляторы, которые позволяют транслировать отдельные участки программы, называются *компиляторами с раздельной трансляцией*.

Часть программы, которая может быть отдельно оттранслирована, называется *программным модулем*. Оттранслированный программный модуль сохраняется в виде отдельного файла в объектном формате, где, кроме машинных команд, сохраняется информация об именах переменных, адресах команд, требующих модификации при объединении модулей в единую программу, и отладочная информация. Одновременно с объектным файлом может создаваться листинг этого модуля.

Создание загрузочного файла программы при раздельной трансляции производится с использованием двух программ: *транслятора* исходного текста и *редактора связей (компоновщика)* объектных файлов

Программа *редактор связей (компоновщик)* позволяет объединять несколько объектных файлов в один исполняемый файл. Для этого имена всех объектных файлов передаются в программу редактора связей в качестве параметров. Вот как выглядит вызов редактора связей из командной строки DOS для объединения трех модулей:

```
rl51.exe progr.obj, modul1.obj, modul2.obj
```

В результате работы редактора связей, вызванного при помощи приведенной командной строки, будет создан исполняемый файл программы с именем progr.

При использовании интегрированной среды программирования, входящей в состав большинства современных инструментальных средств разработки программного обеспечения микроконтроллеров, работа с редактором связей значительно упрощается. Для объединения нескольких модулей в загрузочный файл достаточно создать программный проект, в который включены необходимые файлы с исходным кодом.

На первый взгляд раздельная трансляция не должна вызывать каких-либо проблем, однако это не так. В процессе компиляции исходного текста программы транслятор, обрабатывая определения переменных, констант и операторы, составляет таблицу ссылок. Если при втором просмотре исходного текста программы, во время которого формируется объектный модуль, транслятор обнаружит в выражении или операторе имя переменной или метки, отсутствующее в таблице, то будет сформировано сообщение об ошибке и объектный файл будет стерт с диска компьютера.

Для того чтобы транслятор вместо формирования сообщения об ошибке записал в объектный файл информацию, необходимую для работы редактора связей, нужно использовать специальные директивы ссылок на внешние (определенные в других модулях) переменные или метки. В языке программирования ASM-51 эти директивы называются PUBLIC (общедоступные) и EXTRN (внешние).

Для ссылки на переменную или метку из другого программного модуля служит директива EXTRN. В ней перечисляются через запятую метки и переменные, точное значение которых редактор связей должен получить из другого модуля и модифицировать все команды, в которых эти метки или переменные используются. Пример использования директивы EXTRN в программе на языке программирования ASM-51:

```
EXTRN DATA (BufInd, ERR)
EXTRN CODE (Podprogr)
```

Для того чтобы редактор связей мог осуществить связывание модулей в единую программу, переменные и метки, объявленные, по крайней мере, в одном из модулей как EXTRN, в другом модуле должны быть объявлены как доступные для всех модулей при помощи директивы PUBLIC. Пример использования директивы PUBLIC на языке программирования ASM-51:

```
PUBLIC BufInd, Parametr
PUBLIC Podprogr, ?Podprogr?Byte
```

Использование нескольких модулей при разработке программы увеличивает скорость трансляции и, в конечном итоге, скорость написания программы. Однако объявления переменных и имен подпрограмм внешних модулей загромождают исходный текст модуля. Кроме того, при использовании чужих модулей трудно объявить переменные и подпрограммы без ошибок. Поэтому объявления переменных, констант и предварительные объявления подпрограмм хранят во включаемых файлах, которые называются файлами-заголовками. Правилом хорошего тона считается при разработке программного модуля сразу же написать для него файл-заголовок, который может быть использован программистами, работающими с вашим программным модулем.

Из всего, рассмотренного ранее понятно, что большую программу можно разделить на части. Остается открытым вопрос — как разделять единую программу на части, ведь это же цельный организм, который живет и развивается по мере разработки программы! Как найти часть программы, наименьшим образом связанную с остальными частями? А искать особенно и не нужно! Ведь у нас уже есть такие части, которые связаны с остальной программой

либо одним, либо несколькими заранее оговоренными и неизменными параметрами! Это подпрограммы.

Именно подпрограммы обычно выносят в отдельные модули. При этом стараются собрать в одном модуле подпрограммы, решающие подобные задачи. Например, в один модуль можно поместить подпрограммы, вычисляющие математические функции, такие как синус, косинус, тангенс или котангенс. В другой модуль можно поместить подпрограммы, отвечающие за ввод и вывод информации из вашего устройства. В третьем модуле собрать подпрограммы, осуществляющие кодирование и декодирование информации.

Такое разделение программы на отдельные модули позволяет либо привлечь для написания программных модулей специалистов, хорошо разбирающихся в данной области, либо решать поставленную задачу по частям. Если модули выделены правильно, то зависимости между отдельными частями программы получаются минимальными и изменения в одной из них минимально затрагивают другие или вовсе на них не влияют. Многомодульность дает значительный выигрыш в скорости разработки программы. Дополнительный выигрыш получится при написании следующих программ, т. к. в них вы сможете использовать готовые написанные и отлаженные модули.

Наиболее ярким примером использования модулей являются языки программирования высокого уровня, такие как С. Ведь никому не приходит в голову писать собственные программы вычисления синусов и косинусов или функции ввода-вывода для этого языка! Используются готовые подпрограммы.

Программа-монитор

Разработка программ для микропроцессоров происходит во многом иначе, чем для универсального компьютера. При выполнении программы на универсальном компьютере ее запуск, взаимодействие с внутренними и внешними устройствами или человеком берет на себя операционная система. Программа, написанная для микроконтроллера, должна решать все эти задачи самостоятельно. Программа универсального компьютера когда-нибудь запускается и завершается. Программа, управляющая микроконтроллером, запускается при включении устройства и не завершает свою работу, пока не будет выключено питание.

Рассмотрим программу для микроконтроллера, которая будет выполнять поставленные задачи. Назовем эту программу "монитор", поскольку она "наблюдает" за использованием ресурсов системы. Схема алгоритма программы-монитора приведена на рис. 21.10. После включения питания эта программа должна настроить микросхему микроконтроллера для выполняемой разраба-

тываемым устройством задачи. Для этого она должна запрограммировать определенные выводы микросхемы микроконтроллера на ввод или вывод информации, включить и настроить внутренние таймеры и т. д. Этот блок алгоритма программы-монитора называется инициализацией процессора. Инициализация микроконтроллера выполняется только один раз. Повторно она может потребоваться только при сбоях в работе микроконтроллера. Устранение таких сбоев производится аппаратным сбросом микроконтроллера.

Рис. 21.10. Схема алгоритма программы-монитора

Основная часть программы, реализующая алгоритм работы устройства, начинает выполняться после инициализации микроконтроллера. При этом необходимо понимать, что если в устройствах, не содержащих программно-управляемых компонентов, ввод, обработка и вывод информации производятся различными аппаратными блоками, то при выполнении программы эти же действия производятся последовательно одним и тем же устройством — микропроцессором. Для выполнения каждой задачи обычно пишется отдельная подпрограмма. То есть при программной реализации устройства подпрограмма выполняет те же функции, что и отдельный блок при схемотехнической реализации устройства.

Точно так же, как и при аппаратной реализации различных блоков устройства, необходимо, чтобы каждая подпрограмма решала свою конкретную задачу. При написании программы очень часто возникает соблазн решать проблемы в месте их возникновения. В результате появляется огромное количество участков кода, занимающихся вводом информации, еще столько же участков, занимающихся управлением одним и тем же устройством. Гораздо лучше, если вводом информации занимается одна подпрограмма, для управ-

ления устройством, подключенным к микроконтроллеру, служит другая подпрограмма, а общий алгоритм работы устройства формирует третья подпрограмма. При этом нельзя допускать ситуации, когда подпрограмма ввода информации тут же пытается обработать полученные данные, и тем более начать управление каким-либо устройством.

Для того чтобы работали все написанные подпрограммы, они включаются в один бесконечный цикл. Это эквивалентно периодическому запуску аппаратных блоков. Соединению между блоками соответствует взаимодействие частей программы, осуществляемое при помощи глобальных переменных. В этом же цикле обычно предусматривается блок обработки ошибок. Его предназначение сообщать оператору (пользователю) о непредвиденной ситуации, такой как неправильный ввод с клавиатуры или неправильные данные, полученные от подключенного к микроконтроллеру устройства.

Приведем пример реализации такого алгоритма работы программы. Для написания программы воспользуемся принципами структурного программирования, рассмотренными ранее. В этом случае для проверки правильности работы программы можно воспользоваться программами-заглушками. Исходный текст программы, реализующей алгоритм программы-монитора, схема которого показана на рис. 21.10, приведен в листинге 21.22.

Листинг 21.22. Пример реализации программы-монитора на языке С-51

```
/*-------------------------------------------------------
Подпрограмма инициализации микроконтроллера
-------------------------------------------------------*/
void Init(void)
{//По мере написания программы здесь будут добавляться операторы,
   //настраивающие элементы внутренней структуры микроконтроллера
 //на необходимый режим работы.)

/*-------------------------------------------------------
Подпрограмма опроса клавиатуры
-------------------------------------------------------*/
void SborInf(void)
{}

/*-------------------------------------------------------
Подпрограмма обработки информации
-------------------------------------------------------*/
void ObrabInf(void)
{//В этой подпрограмме обычно осуществляется переключение
 //режимов работы устройства}
```

```
/*-------------------------------------------------------
Подпрограмма обработки ошибок
-------------------------------------------------------*/
void ObrabOshib(void)
{//В этой подпрограмме обычно осуществляется индикация
 //ошибочного ввода информации с клавиатуры}

void main(void)
{Init();
 while(1)
 {SborInf();
  ObrabInf();
  ObrabOshib();}
}
```

Как видно из приведенного текста, программа еще ничего не делает. Все будущие действия обозначены в именах подпрограмм-заглушек, однако эта программа уже может быть оттранслирована и запущена. В отладчике можно проверить, что при включении устройства мы действительно попадаем в подпрограмму инициализации, а затем последовательно вызываются подпрограммы сбора и обработки информации, а также подпрограмма обработки ошибок.

При использовании нескольких подпрограмм встает проблема обмена информацией между ними. Как уже рассматривалось ранее, информация в подпрограмму может быть передана через параметры или через глобальные переменные. При создании программы-монитора может потребоваться передавать одну и ту же информацию нескольким подпрограммам (что аналогично параллельному соединению аппаратных блоков), поэтому в мониторе информация обычно передается через глобальные переменные, которые последовательно подвергаются обработке всеми подпрограммами, включенными в бесконечный цикл.

Рассмотрим подробнее, как реализуется чтение информации с выводов микроконтроллера. Очень важно, чтобы сигналы со всех выводов считывалась одновременно, иначе можно получить комбинации битов, не соответствующие действительности. Иллюстрация такой ситуации приведена на рис. 21.11. В результате опроса сигналов в контроллер будет введен код 000, который никогда не присутствовал на выводах микросхемы!

К сожалению, микроконтроллер может опрашивать порты только последовательно. Тем не менее, содержимое всех четырех портов может быть скопировано во внутреннюю память микроконтроллера четырьмя командами. Благодаря высокому быстродействию микроконтроллера можно обеспечить почти

одновременное считывание информации со всех выводов микросхемы. Это действие займет единицы микросекунд, поэтому при таком подходе к разработке управляющей программы вероятность возникновения ситуации, представленной на рис. 21.11, невысока.

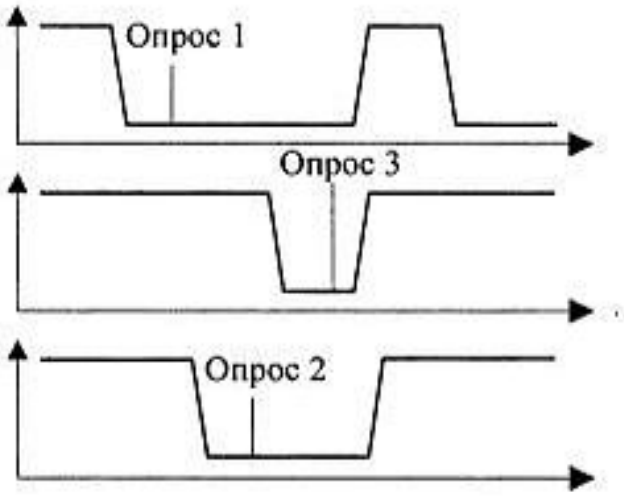


Рис. 21.11. Пример неодновременного опроса сигналов на выводах микроконтроллера

В качестве примера рассмотрим ввод информации с 16-кнопочной клавиатуры. Управляющая программа для микроконтроллера жестко зависит от принципиальной схемы разрабатываемого устройства. Невозможно написать программу для микроконтроллерного устройства, не имея перед глазами его принципиальной схемы. Схема подключения клавиатуры к микроконтроллеру приведена на рис. 21.12.

Рассмотрим только подпрограмму ввода информации, т. к. уже указывалось ранее — остальные части программы не должны зависеть от алгоритма опроса кнопок клавиатуры. Они лишь должны использовать информацию, сформированную подпрограммой ввода информации.

Объявим глобальную переменную `SkanCode`, в которой будем хранить значения электрических сигналов на выводах микроконтроллера, подключенных к контактам клавиатуры. Переменная используется для формирования управляющих сигналов и считывания состояний контактов. Код, хранящийся в этой переменной и отображающий значения электрических сигналов на выводах микроконтроллера, называется скан-кодом клавиатуры.

Теперь собственно о программе. Если бы речь шла не о клавиатуре, то ввод информации свелся бы к простому копированию сигналов с внешних выводов порта в переменную командой

```
MOV SkanCode, P1;  //Скопировать состояние сигналов на порту P1
                   // в переменную SkanCode.
```

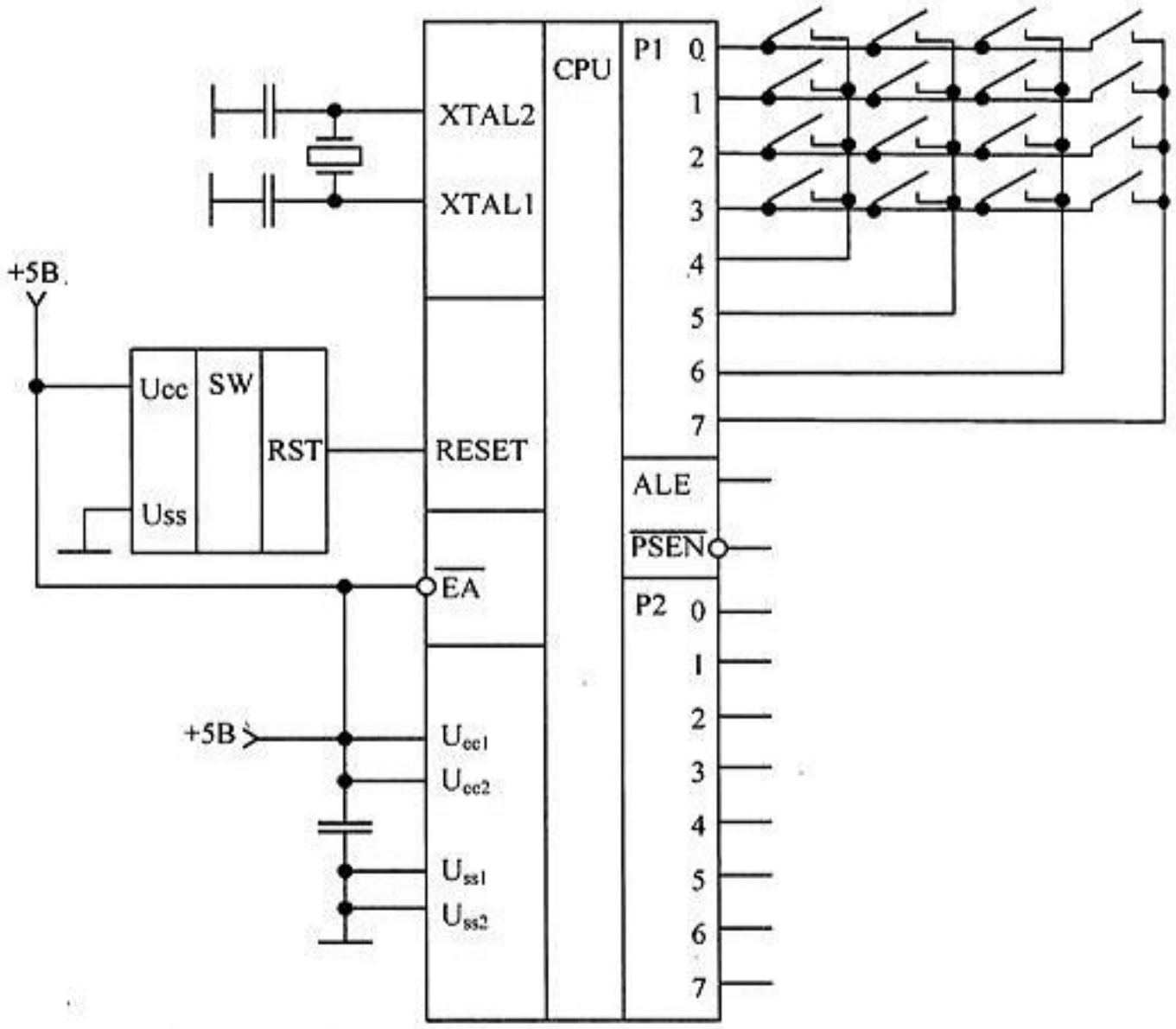


Рис. 21.12. Схема подключения клавиатуры к микроконтроллеру

Но при вводе информации с клавиатуры необходимо осуществить ее сканирование, т. е. последовательный опрос колонок или столбцов клавиатуры (по вашему желанию возможен любой из двух вариантов).

Сначала определим, каким сигналом будем осуществлять сканирование. Для этого нужно вспомнить внутреннее устройство порта. Обратите внимание, что в качестве примера выбран микроконтроллер семейства MCS-51! При использовании других микроконтроллеров принципиальная схема и сигналы опроса могут оказаться другими! Путь протекания тока через замкнутую кнопку клавиатуры и элементы порта для микроконтроллеров семейства MCS-51 показан на эквивалентной схеме двух разрядов порта P1 (рис. 21.13).

Из приведенной на рис. 21.13 эквивалентной схемы видно, что для опроса колонки кнопок необходимо открыть нижний транзистор порта, подключенного к выбранной колонке. При этом нужно обеспечить запирание транзисто-

ров, подключенных к остальным колонкам кнопок. Это позволит исключить неоднозначность определения номера нажатой кнопки. Для открывания транзистора достаточно записать в соответствующий бит порта логический 0, а для запирания — логическую 1. В результате коды опроса клавиатуры для схемы, приведенной на рис. 21.12, будут выглядеть следующим образом:

- код опроса первой колонки клавиатуры — `11110111b`.
- код опроса второй колонки клавиатуры — `11111011b`.
- код опроса третьей колонки клавиатуры — `11111101b`.
- код опроса четвертой колонки клавиатуры — `11111110b`.

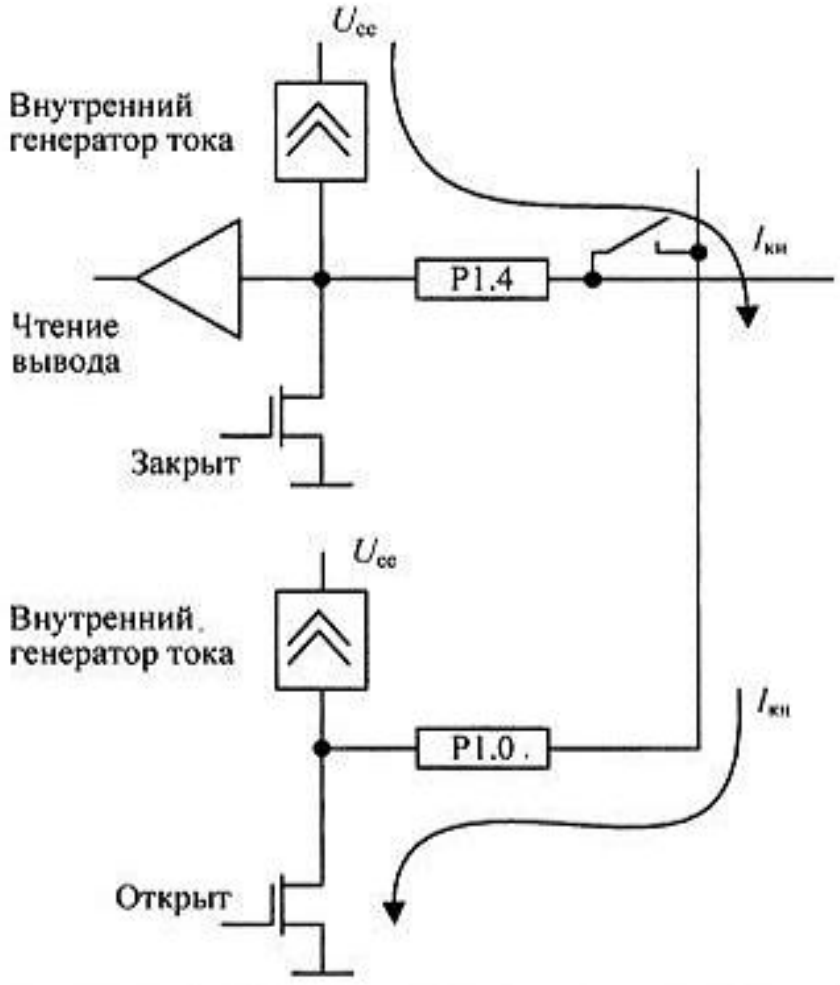


Рис. 21.13. Эквивалентная схема цепи протекания тока через замкнутую кнопку клавиатуры

То есть для опроса состояния кнопок клавиатуры потребуется выдача на порт P1, по крайней мере, четырех кодов. При этом следует отметить, что скан-код, считанный с клавиатуры, содержит намного больше информации, чем просто номер замкнутого контакта. Например, при помощи скан-кода можно определить нажатие нескольких кнопок одновременно, поэтому лучше хра-

нить во внутренней памяти контроллера непосредственно скан-код, а не готовый номер кнопки. Пример подпрограммы опроса клавиатуры приведен в листинге 21.23.

Листинг 21.23. Подпрограмма опроса клавиатуры на языке программирования С-51

```
/*-------------------------------------------------------
Подпрограмма опроса клавиатуры
-------------------------------------------------------*/
#define ScanCol1 0xf7 //11110111b
#define ScanCol2 0xfb //11111011b
#define ScanCol3 0xfd //11111101b
#define ScanCol4 0xfe //11111110b

void SborInf(void)
{register char tmp;
 SkanCode=0xff;       //Занести код отсутствия нажатой кнопки

 P1=ScanCol1;         //Выдать код опроса кнопок в колонке 1,
 tmp=P1;              //прочитать состояние кнопок,
 if(tmp != ScanCol1) //и если хоть одна кнопка в колонке нажата,
   SkanCode &= tmp;     //то запомнить скан-код
 P1=ScanCol2;         //Выдать код опроса кнопок в колонке 2,
 tmp=P1;              //прочитать состояние кнопок,
 if(tmp != ScanCol2) //и если хоть одна кнопка в колонке нажата,
   SkanCode &= tmp;     //то запомнить скан-код
 P1=ScanCol3;         //Выдать код опроса кнопок в колонке 3,
 tmp=P1;              //прочитать состояние кнопок,
 if(tmp != ScanCol2) //и если хоть одна кнопка в колонке нажата,
   SkanCode &= tmp;     //то запомнить скан-код
 P1=ScanCol4;         //Выдать код опроса кнопок в колонке 4,
 tmp=P1;              //прочитать состояние кнопок,
 if(tmp != ScanCol2) //и если хоть одна кнопка в колонке нажата,
   SkanCode &= tmp;     //то запомнить скан-код
}
```

В результате работы этой подпрограммы нули появятся в битах, соответствующих цепям колонок и строк клавиатуры, к которым подключены нажатые кнопки. Использование операции побитового умножения & позволяет определить не только нажатие одиночной кнопки клавиатуры, но и одновременное нажатие двух кнопок, а также некоторых комбинаций трех кнопок клавиатуры.

Эта же подпрограмма может быть реализована и на языке программирования ассемблер. При этом она практически не будет отличаться от подпрограммы, приведенной выше (см. листинг 21.23). Пример реализации опроса клавиатуры на языке программирования ASM-51 приведен в листинге 21.24.

Листинг 21.24. Подпрограмма опроса клавиатуры на языке ASM-51

```
  public OprosKlaviat

OprosCol_1 equ 11110111b
OprosCol_2 equ 11111011b
OprosCol_3 equ 11111101b
OprosCol_4 equ 11111110b

_code segment code

  rseg _code
;-------------------------------------------------------------------
;ПОДПРОГРАММА ОПРОСА КЛАВИАТУРЫ
;-------------------------------------------------------------------
;Подпрограмма опроса клавиатуры сканирует клавиатуру 4x4
;и возвращает в аккумуляторе полученный скан-код клавиатуры

OprosKlaviat:
  mov  SkanCode,#FFh          ;Занести код не нажатой кнопки

  mov  AdrKlav,#OprosCol_1    ;Выдать код опроса кнопок в колонке 1,
  mov  A,AdrKlav              ;прочитать состояние кнопок,
  cjne A,#OprosCol_1,TstCol2  ;и если хоть одна кнопка в колонке нажата
  sjmp TstCol2                ;(код опроса не совпадает со считанным),
SetCol1: anl A,SkanCode               ;то запомнить
         mov SkanCode,A               ;скан-код
TstCol2:
  mov  AdrKlav,#OprosCol_2    ;Выдать код опроса кнопок в колонке 2,
  mov  A,AdrKlav              ;прочитать состояние кнопок,
  cjne A,#OprosCol_2,SetCol2  ;и если хоть одна кнопка
  sjmp TstCol3                ;в колонке нажата,
SetCol2: anl A,SkanCode               ;то запомнить
         mov SkanCode,A               ;скан-код
TstCol3:
  mov  AdrKlav,#OprosCol_3    ;Выдать код опроса кнопок в колонке 3,
  mov  A,AdrKlav              ;прочитать состояние кнопок,
```

```
  cjne A,#OprosCol_3,SetCol3  ;и если хоть одна кнопка
  sjmp TstCol4                ;в колонке нажата,
SetCol3: anl A,SkanCode               ;то запомнить
         mov SkanCode,A               ;скан-код
TstCol4:
  mov  AdrKlav,#OprosCol_4    ;Выдать код опроса кнопок в колонке 4,
  mov  A,AdrKlav              ;прочитать состояние кнопок,
  cjne A,#OprosCol_4,SetCol4  ;и если хоть одна кнопка
  sjmp EndOpros               ;в колонке нажата,
SetCol4: anl A,SkanCode               ;то запомнить
         mov SkanCode,A               ;скан-код
EndOpros:
  ret
```

Интересной особенностью приведенной подпрограммы является использование команды `sjmp`, позволяющее превратить команду перехода по неравенству кодов `cjne` в команду перехода по равенству кодов. Остальные действия подробно пояснены комментариями и поэтому в дополнительных объяснениях не нуждаются.

Следующая подпрограмма обработки данных может из скан-кодов сформировать последовательности (строки) кодов нажатых кнопок, которые могут быть использованы как управляющие команды. Для хранения этих последовательностей можно использовать массив символов (в языке программирования ассемблер это просто соседние ячейки памяти).

При написании программы до сих пор считалось, что мы работаем с идеальными кнопками. Однако это не так. Обычно при нажатии и отпускании кнопки возникает переходный процесс, который называется дребезгом контактов. Начинающие разработчики аппаратуры применяют различные методы борьбы с этим явлением — от применения специальных схемотехнических решений до повторного опроса кнопок в течение некоторого времени.

Рассмотрим подробнее механизм возникновения явления дребезга контактов. Его причиной является упругость самих контактов. При нажатии на кнопку или при срабатывании какого-либо датчика контакты испытывают механическое импульсное воздействие. Обладая некоторой упругостью и массой, они начинают совершать колебательное движение: один контакт ударяется о другой (замыкание) и снова отходит (размыкание). Колебательный процесс продолжается некоторое время. При этом количество замыканий и размыканий контактов случайно и зависит от механических свойств контактов и механической силы, воздействующей на эти контакты. Обычно этот процесс длится от одной до восьми миллисекунд.

Если мы будем опрашивать состояние контактов с периодом, превышающим максимальную длительность дребезга, то даже не заметим, что они несколько

раз замыкались или размыкались в промежутке между опросами контактов. Для этого можно включить в основной цикл программу, которая будет обеспечивать выполнение цикла один раз за строго определенное время.

Пример временной диаграммы сигнала на контактах кнопки приведен на рис. 21.14. На временной оси этого рисунка моменты считывания сигналов показаны рисками. Внизу рисунка обозначены номера временных слотов. На приведенной временной диаграмме четко просматривается зона дребезга контактов.

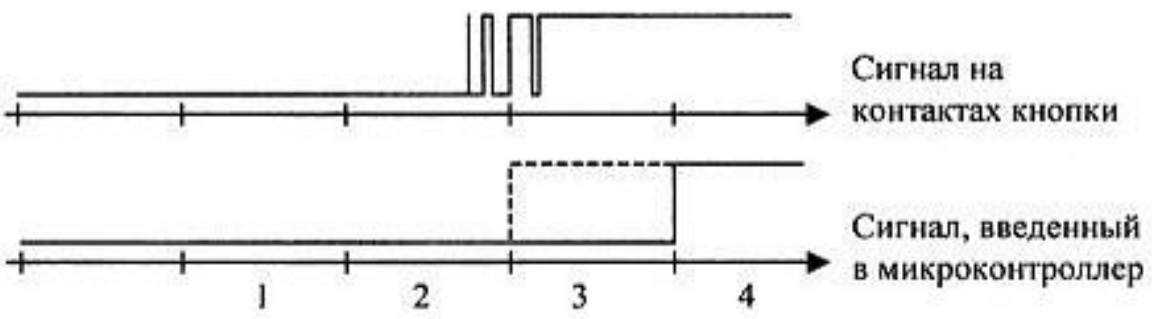


Рис. 21.14. Временные диаграммы напряжения на контактах кнопки и сигнала, введенного в микроконтроллер

Для иллюстрации эффекта подавления дребезга контактов за счет ввода информации в строго определенные моменты времени на этой временной диаграмме выбран наихудший случай момента взятия отсчета. Этот момент совпадает с зоной дребезга контактов. При этом на выводе порта микроконтроллера может быть считан сигнал логического нуля или логической единицы. Но даже в этом случае дополнительный импульс в считанном сигнале не возникает! В случае считывания в момент дребезга контактов логического нуля сигнал, введенный в микроконтроллер, будет выглядеть так, как показано на рис. 21.14 сплошной линией, а в случае считывания логической единицы — пунктиром. Однако вне зависимости от того, какое значение сигнала мы считаем — нулевое или единичное, переход будет только один.

Дребезг контактов приводит только к неопределенности определения времени нажатия кнопки, которое не превышает периода опроса клавиатуры. Выберем это время. Так как мы уже определили, что время дребезга контактов не превышает 8 мс, то можно производить опрос портов, к которым подключены механические контакты, с периодом, который несколько больше, например, 10 мс. Это время и будет временем реакции системы. Но как обеспечить периодический опрос клавиатуры? Ведь программа в процессе выполнения может проходить по различным путям в зависимости от состояния опрашиваемых контактов и содержимого внутренних переменных микроконтроллера! В результате время выполнения программы (тела рабочего цикла) будет случайным. Для того чтобы время прохождения программы по циклу было строго фиксированным, можно воспользоваться таймером.

Таймер, являясь аппаратным элементом микроконтроллера, работает независимо от выполняемой программы, поэтому будет отсчитывать строго определенные промежутки времени. Если в конце выполнения цикла дожидаться срабатывания таймера, то время одного прохождения по циклу будет строго фиксированным. Единственное условие — максимальное время прохождения рабочего участка тела цикла должно быть меньше 10 мс. Но ведь то же самое требуется и с точки зрения максимального времени реакции системы. Если мы успеем несколько раз нажать на кнопки, а устройство не среагирует на эти нажатия, то что полезного можно сделать с его помощью? Если рабочая часть тела цикла будет выполнена быстро, то ждать срабатывания таймера придется долго. Если же проход по циклу будет выполняться долго, то после завершения рабочей части цикла долго ждать срабатывания таймера не придется. Соотношение времени выполнения рабочей части цикла и ожидания срабатывания таймера приведено на рис. 21.15.

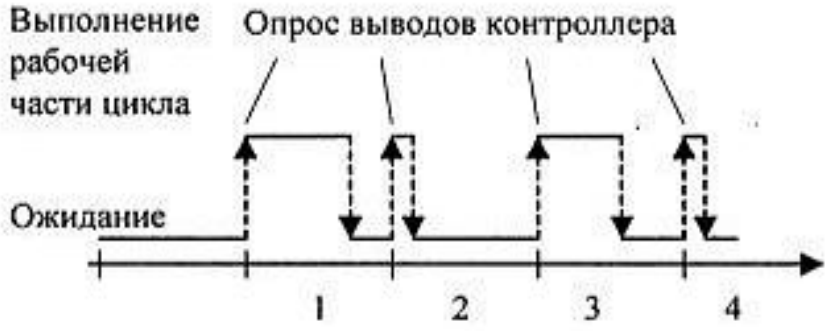


Рис. 21.15. Соотношение времени выполнения рабочей части цикла и ожидания срабатывания таймера

Для организации периодического опроса клавиатуры следует усложнить подпрограмму инициализации. Теперь в ней требуется настроить таймер на выбранный период времени, а значит, задать и его режим работы. Выберем для задания равных промежутков времени таймер T0. Он может обеспечить задание 10-мс промежутка времени только в режиме 16-разрядного таймера. Подпрограмма инициализации микроконтроллера, выполняющая настройку таймера, приведена в листинге. 21.25.

Листинг 21.25. Подпрограмма инициализации микроконтроллера, выполняющая настройку таймера T0 на выработку 10-мс промежутков времени

```
/*-------------------------------------------------------
Подпрограмма инициализации микроконтроллера
-------------------------------------------------------*/
void Init(void)
```

```
{TMOD=0x01;          //Настроить T0 на режим 16-разрядного таймера
 TH0=(-10000)>>8; //Занести старший байт числа -10000
 TL=-10000;          //Занести младший байт числа -10000
 TF=0;               //Обнулить флаг переполнения таймера
 TR0=1;              //Включить таймер
}
```

В приведенном примере предполагается, что микроконтроллер работает с частотой тактового генератора 12 МГц. В этом случае на вход таймера T0 будут поступать импульсы с периодом 1 мкс. Число, которое необходимо загружать в таймер, можно найти как отношение требуемого интервала времени, 10 мс, и периода импульсов на входе таймера, 1 мкс. Так как таймер T0 суммирующий, то в регистры таймера будем загружать отрицательное число с абсолютным значением, равным требуемому отношению. В приведенном участке программы (см. листинг 21.25) для выделения старшего байта из шестнадцатиразрядного числа использована функция сдвига этого числа на 8 разрядов вправо.

Теперь в тело основного цикла нужно включить участок программы, который будет ожидать окончания работы таймера, и только после этого приступать к выполнению следующего прохода по циклу. Это можно сделать при помощи команды, которая будет проверять флаг переполнения таймера TF0. Затем необходимо снова задать следующий интервал времени. Пример программы, в которой один проход по бесконечному циклу будет осуществляться один раз за 10 мс, приведен в листинге 21.26.

Иными словами, программа, приведенная в листинге 21.26, реализует схему, изображенную на рис. 21.16. В этой схеме есть три блока, синхронизированные от одного генератора, выполненного на внутреннем генераторе и таймере. Связям между блоками соответствует взаимодействие частей программы при помощи глобальных переменных, изображенных над линиями со стрелочками. Порядок выполнения подпрограмм в цикле неважен, т. к. они только подготавливают данные для выдачи на выводы микроконтроллера. Сигналы появятся на выводах микросхемы только в следующем временном слоте, т. е. после завершения реакции системы. Важны только связи между подпрограммами, а они осуществляются глобальными переменными.

Исключение влияния порядка выполнения подпрограмм — важнейший принцип рассматриваемой системы. Время между моментами ввода и вывода сигналов как бы останавливается. И когда бы ни начиналось выполнение подпрограмм, они завершают работу одновременно. Все это верно при условии, что цикл должен успеть завершиться до срабатывания таймера!

Использование глобальных переменных для связи между подпрограммами позволяет осуществлять не только последовательное, но и параллельное со-

единение аппаратных блоков, реализуемых программно. Пример такого соединения аппаратных блоков мы рассмотрим позднее.

Листинг 21.26. Программа, выполняющая тело рабочего цикла один раз в 10 мс

```
char SkanCode;      //Переменная, через которую будет передана информация
                    //подпрограмме ObrabInf
char Error; ;       //Переменная, через которую будет передана информация
                    //подпрограмме ObrabOshib

void main(void)
{Init();
 while(1)
 {SborInf();        //Опросить кнопки, подключенные к микроконтроллеру
  ObrabInf();
  ObrabOshib();

  do;while(TF==0); //Подождать, пока не истечет 10 мс
//------ подготовить таймер к следующему циклу --------------------
  TF=0;             //Обнулить флаг переполнения таймера
  TH0=(-10000)>>8; //Задать следующий интервал времени,
  TL=-10000;        //равный 10 мс
 }
}
```

В программе, приведенной в листинге 21.26, процессор все время потребляет максимальный ток, определяемый тактовой частотой микроконтроллера. Чем выше тактовая частота, тем больше ток. Максимальный ток потребляется даже при ожидании срабатывания таймера. В то же время в режиме ожидания микроконтроллер можно перевести в режим пониженного энергопотребления. Это делается записью соответствующего бита в регистр SCON. Выйти из этого режима и продолжить выполнение программы микроконтроллер сможет только по прерыванию от таймера.

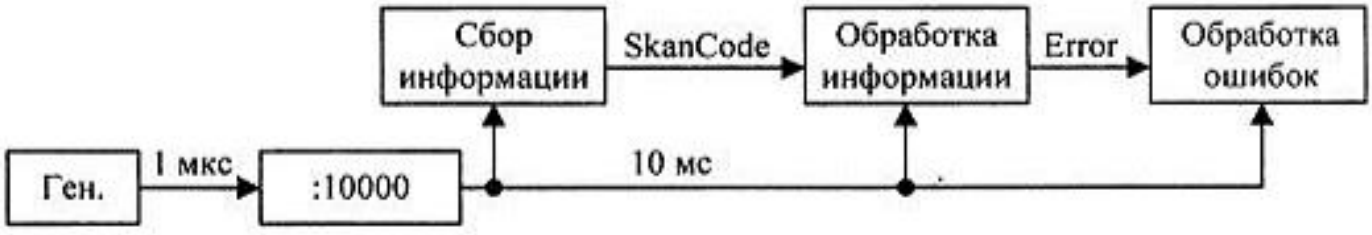


Рис. 21.16. Эквивалентная схема устройства, реализованного программой, приведенной в листинге 21.26

При включении питания все прерывания запрещены, поэтому следует разрешить прерывания от таймера. Это достаточно сделать только один раз после включения питания микроконтроллера, поэтому команды, разрешающие прерывания от таймера, нужно поместить в подпрограмму инициализации микроконтроллера. Ее новый вариант приведен в листинге 21.27.

Листинг 21.27. Подпрограмма инициализации микроконтроллера, настраивающая таймер T0 для задания 10-мс промежутков времени и разрешающая прерывания T0

```
/*-------------------------------------------------------
Подпрограмма инициализации микроконтроллера
-------------------------------------------------------*/
void Init(void)
{TMOD=0x01;          //Настроить таймер T0 на первый режим работы
 TH0=(-10000)>>8; //Занести старший байт числа -10000
 TL=-10000;          //Занести младший байт числа -10000
 TR0=1;              //Включить таймер
 IE=(1<<7)|          //Разрешить прерывания (седьмой бит регистра IE)
    (1<<1);          //Разрешить прерывания от таймера T0
                        //(первый бит регистра IE)
}
```

В этой подпрограмме флаг переполнения таймера T0 не сбрасывается, т. к. после выполнения сброса микроконтроллера этот флаг и так содержит нулевое значение. Разрешение прерываний осуществляется командой записи в регистр IE соответствующего числа. Для разрешения прерываний от таймера T0 достаточно записать единицу в первый бит этого регистра. Кроме того, для разрешения прерываний необходимо записать единицу в седьмой бит регистра IE, разрешающий или запрещающий все прерывания.

Для тех, кто еще не привык считать в двоичной арифметике и легко переводить двоичные числа в шестнадцатеричные и обратно, в этом примере приведен способ вычисления констант с помощью операции сдвига. Известно, что $2^0=1$. Тогда операция двоичного сдвига (1<<5) вычислит константу 2^5. Нам требуется записать единицу в два бита. Необходимую константу можно образовать при помощи операции логического сложения '|'. Не нужно пугаться довольно сложного выражения для вычисления константы, записываемой в IE. Оно вычисляется только один раз, на этапе трансляции программы в машинные коды микроконтроллера. В разрабатываемую программу будет помещен готовый результат вычислений. При выполнении программы значение константы уже известно.

Еще одно преимущество приведенного способа записи констант — это возможность снабдить комментарием каждый бит константы, что повышает на-

глядность программы, а значит — увеличивает скорость ее написания и отладки.

Теперь при переполнении таймера будут возникать прерывания, которые будут передавать управление программой на вектор прерывания. Нам пока ничего не нужно делать по прерыванию, но, тем не менее, подпрограмму обслуживания прерывания для того, чтобы вернуться из прерывания в основную программу, необходимо написать. Пример такой подпрограммы на языке программирования C-51 приведен в листинге 21.28.

Листинг 21.28. Подпрограмма обслуживания прерываний от таймера T0

```
/*-------------------------------------------------------
Подпрограмма обслуживания прерываний от таймера T0
-------------------------------------------------------*/
void PrerT0(void) interrupt 1
{
}
```

На то, что это подпрограмма обслуживания прерывания, указывает ключевое слово `interrupt`. Номеру прерывания `1` соответствует конкретный адрес вектора прерывания, где будет размещаться переход на подпрограмму обслуживания прерывания. В приведенном примере это вектор прерывания от таймера T0.

Теперь все подготовительные операции выполнены, и можно осуществить режим пониженного потребления тока микроконтроллера. У микроконтроллеров семейства MCS-51 есть два режима энергопотребления: остановка процессорного ядра и остановка задающего генератора. В нашем случае останавливать задающий генератор ни в коем случае нельзя, т. к. при этом остановится таймер, и микроконтроллер можно будет разбудить только при помощи аппаратного сброса. Остается только режим остановки процессорного ядра.

Режимы пониженного энергопотребления задаются при помощи двух младших битов регистра `PCON`. Остановить процессорное ядро можно, записав в нулевой разряд этого регистра единицу. Для того чтобы не изменить содержимое остальных битов этого регистра, воспользуемся операцией логического суммирования. Пример использования режима пониженного потребления тока для задания 10-мс интервалов времени между проходами по основному циклу программы показан в листинге 21.29.

Листинг 21.29. Основная функция программы, работающей один раз в 10 мс с пониженным потреблением тока

```
void main(void)
{Init();
```

```
while(1)
 {SborInf();          //Опросить кнопки, подключенные к микроконтроллеру
  ObrabInf();
  ObrabOshib();
  PCON|=1;            //Остановить процессорное ядро и
                       //подождать, пока не истекут 10 мс
//------ подготовить таймер к следующему циклу --------------------
  TF=0;               //Обнулить флаг переполнения таймера
  TH0=(-10000)>>8;    //Задать следующий интервал времени,
  TL=-10000;          //равный 10 мс
 }
}
```

Хотелось бы сразу подчеркнуть, что описанная программа вызывает импульсные помехи с периодом 10 мс. Они распространяются по цепям питания микроконтроллера. В ряде случаев этот фактор критичен и возможно придется отказаться от режима понижения тока потребления в пользу варианта программы, приведенного в листинге 21.26. Если при этом микроконтроллер будет большую часть времени простаивать, то имеет смысл рассмотреть возможность уменьшения потребляемого тока за счет снижения тактовой частоты микроконтроллера. (Следует отметить, что ряд современных микроконтроллеров, например AduC824, позволяют регулировать частоту работы процессорного ядра непосредственно в процессе работы.)

Достаточно часто программа, написанная для микроконтроллера, реализует несколько режимов работы. Так как в каждый отдельный момент времени требуется только один режим работы, то для каждого из них можно использовать отдельную программу-монитор, вызываемую как подпрограмма из основной программы-монитора. Именно он должен осуществлять переключение между режимами. Поэтому завершение работы любого из режимов должно осуществляться выходом из подпрограммы, реализующей этот режим.

Использование таймера для организации параллельных программных потоков

В рассмотренном примере все время процессора неограниченно принадлежит одной программе-монитору. Это естественно, если время реакции любого алгоритмического блока, входящего в программу-монитор, одинаково. Однако возможны задачи, когда на одном микроконтроллере реализуются алгоритмические блоки, время реакции которых на поступающие от входных

портов события должно быть различно. В таком случае используют разбиение времени микроконтроллера на временные слоты (интервалы). Это, как и в предыдущем случае, делается при помощи таймера. Однако для реализации устройства используется несколько подпрограмм-мониторов. Кроме них в состав программы вводится еще один алгоритмический блок — диспетчер. Собственно говоря, этот блок уже присутствовал в предыдущем примере. Это программа, осуществлявшая разбиение времени процессора на строго определенные интервалы.

Пример функциональной схемы устройства, в котором требуется различное время реакции на входное воздействие, приведен на рис. 21.17.

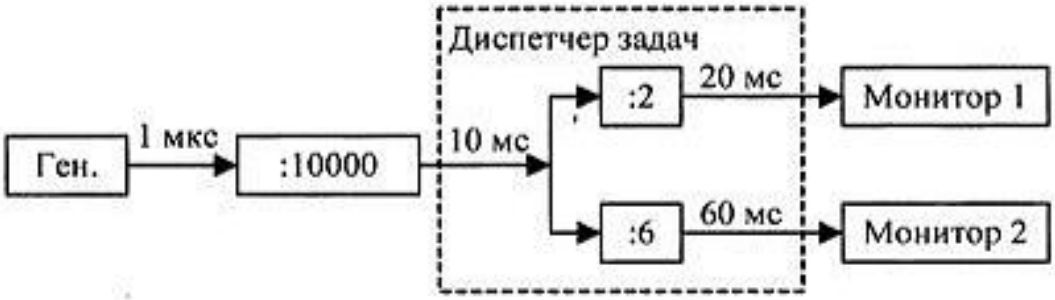


Рис. 21.17. Схема устройства, реализующая разное время реакции на входные воздействия

Можно было бы просто включить в основной монитор два различных счетчика и, анализируя их значения, вызывать соответствующие подпрограммы, но при этом возможна ситуация, когда общее время выполнения подпрограмм превысит значение временного слота монитора. Намного лучше заранее разбить время выполнения программы на временные слоты и выделить для каждой подпрограммы различные слоты. Это позволит отодвинуть выполнение некритичных по времени подпрограмм на более позднее время. Так реализуются параллельные программные потоки.

На временной диаграмме рис. 21.18 общее время выполнения программы (максимальное допустимое время реакции системы) разбито на шесть временных слотов. При этом первый, третий и пятый временные слоты выделены для второго монитора, что обеспечивает время реакции устройства t2, реализуемое этим монитором, не более 3,3 мс. Это будет один программный поток.

Для первой подпрограммы-монитора с временем реакции на входное воздействие t1 выделен четвертый временной слот. Если времени одного слота для выполнения монитора 1 недостаточно, то есть еще два свободных временных слота. Подпрограмма-монитор 1 может быть разбита на три части, каждая из

которых будет вызываться в своем временном слоте. При этом максимальное время реакции на входное событие у монитора 1 составит t1 = 10 мс.

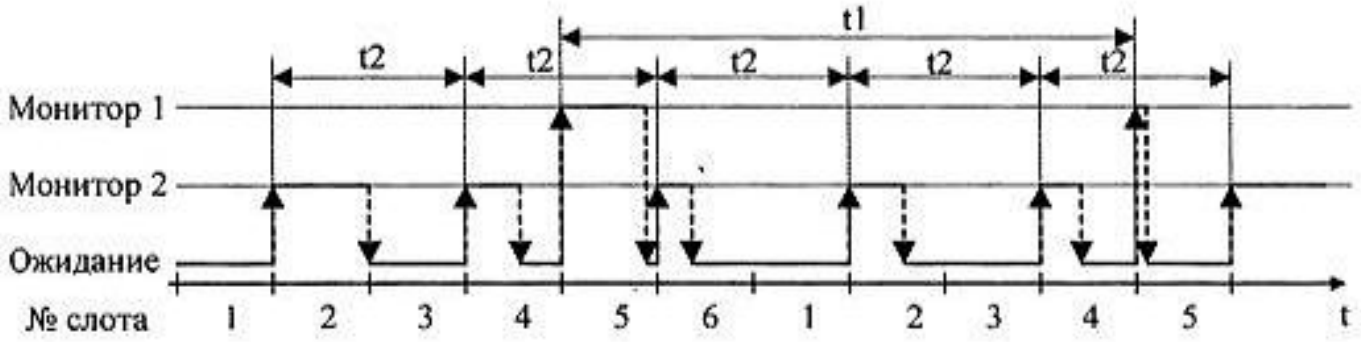


Рис. 21.18. Пример временной диаграммы работы микроконтроллера с двумя подпрограммами-мониторами

Таким образом, в одном процессоре реализовано два программных потока с различным временем реакции на изменение сигналов на выводах микроконтроллера. В принципе, вместо ожидания срабатывания таймера можно организовать еще один программный поток и разместить в этом потоке подпрограммы, время выполнения которых не является критическим для разрабатываемого устройства.

Листинг 21.30. Пример реализации диспетчера программ с двумя программными мониторами

```
char NomSlot=6;
void main(void)
{Init();
 while(1)
 {if(NomSlot==6)SborInf();  //Если шестой слот, то опросить выводы контр.
  else if(NomSlot==5)Monit2();//Если пятый слот, то вызвать монитор 2
  else if(NomSlot==4)Monit1();//Если четвертый слот, то вызвать монитор 1
  else if(NomSlot==3)Monit2();//Если третий слот, то вызвать монитор 2
  else if(NomSlot==1)Monit2();//Если первый слот, то вызвать монитор 2

  if(--NomSlot==0)NomSlot=6;//Уменьшить номер слота, и если первый
                            //слот уже выполнен, то перейти к шестому
  PCON|=1;                //Остановить ядро и подождать, пока не истечет 1,67 мс
//------ подготовить таймер к следующему циклу -------------------------
  TF=0;                   //Обнулить флаг переполнения таймера
  TH0=(-1666)>>8;         //Задать следующий интервал времени, равный 1,67 мс
  TL=-1666;               //(6 слотов = 10 мс)
 }
}
```

Теперь рассмотрим, как можно реализовать приведенные выше принципы организации программы на языке программирования С-51. Исходный текст программы приведен в листинге 21.30. Для нумерации слотов, на которые разбивается время, используется глобальная переменная `NomSlot`. Эта переменная используется как счетчик временных слотов. Именно по ее значению вызывается одна из подпрограмм-мониторов. Подпрограмма инициализации микроконтроллера такая же, как в примере, приведенном в листинге 21.27, и поэтому сейчас рассматриваться не будет.

Использование прерываний для ввода информации о кратковременных сигналах и событиях, наступающих в произвольный момент времени

Использование временных слотов позволяет реализовывать устройства с различными временами реакции. Однако размер временного слота не позволяет обрабатывать быстро текущие процессы. Часто сигнал на входе микроконтроллера длится в течение нескольких микросекунд. Для того чтобы не пропустить такой сигнал, время реакции системы должно быть в пределах нескольких команд микроконтроллера. Если сделать такой маленький временной слот, то микроконтроллер будет постоянно переключаться с задачи на задачу, и ему некогда будет заниматься реализацией основного алгоритма работы. В то же самое время временной интервал между приходом очередного сигнала на вход микроконтроллера может быть достаточно велик, т. е. время реакции системы даже для короткого сигнала может достигать единиц и даже десятков миллисекунд.

Для решения возникшей проблемы в микроконтроллерах предусмотрен механизм прерываний основной программы. Разработчики микроконтроллера предлагают аппаратный вызов подпрограмм при возникновении какого-либо события. Это может быть изменение потенциала на особых выводах микроконтроллера (входах запроса прерывания), переполнение таймеров, завершения передачи или приема байта через последовательный порт. В некоторых типах микроконтроллеров могут быть дополнительные источники прерываний.

В предыдущих главах мы реализовывали ввод информации в начале рабочего цикла программы-монитора. Тем самым мы обеспечивали ввод информации строго через равные интервалы времени. При этом предполагалось, что сигналы на выводах микроконтроллера меняются медленнее интервала опроса.

Использование прерываний позволяет обрабатывать короткие сигналы или пакеты сигналов, приходящие в случайные моменты времени. Основное ограничение при использовании прерываний это то, что мы должны успеть запомнить полученную информацию в глобальной переменной до поступления очередного запроса на прерывание.

Наиболее ярким примером источника событий, наступающих в произвольный момент времени, является последовательный порт. Обычно через него принимаются или передаются многобайтовые команды или пакеты данных. Рассмотрим пример обмена микроконтроллера с универсальным компьютером. Обычно для обмена используется последовательный порт компьютера (COM-порт). Схема согласования уровней сигналов последовательного порта микроконтроллера и COM-порта компьютера приведена на рис. 21.19.

Прежде чем начать программирование обмена через последовательный порт, необходимо определить формат команд обмена с компьютером. Пусть обмен будет производиться ANSI-символами (их легче всего сформировать в любой терминальной программе на персональном компьютере).

Первый переданный символ будет рассматриваться как команда. При использовании в качестве команды заглавных и строчных букв латинского алфавита будет доступно 56 команд. При желании можно добавить еще 64 команды, обозначаемыми буквами русского алфавита. Следующие несколько символов составят поле данных. Обычно здесь используются цифры. Пусть у нас поле данных содержит четыре символа. В качестве завершения команды используем символ возврата каретки (ASCII-код '13'). Этот символ вводится при нажатии на клавишу <Enter>.

Итак, подпрограмма ввода информации должна принять шесть восьмиразрядных символов и только после этого передать управление программе обработки информации. Естественно, что подпрограмма обработки информации, которая входит в один из мониторов, не знает, когда будет завершен прием команды, поэтому введем однобитовую переменную (флаг) завершения приема команды. Подпрограмма ввода информации будет записывать в эту переменную единицу, а подпрограмма обработки команд после выполнения команды будет записывать в эту переменную ноль.

Для того чтобы не пропустить ни одного байта, полученного через последовательный порт, оформим ввод информации как подпрограмму обработки прерывания. Для этого необходимо разрешить прерывания от последовательного порта при помощи подпрограммы инициализации микроконтроллера, приведенной в листинге 21.31.

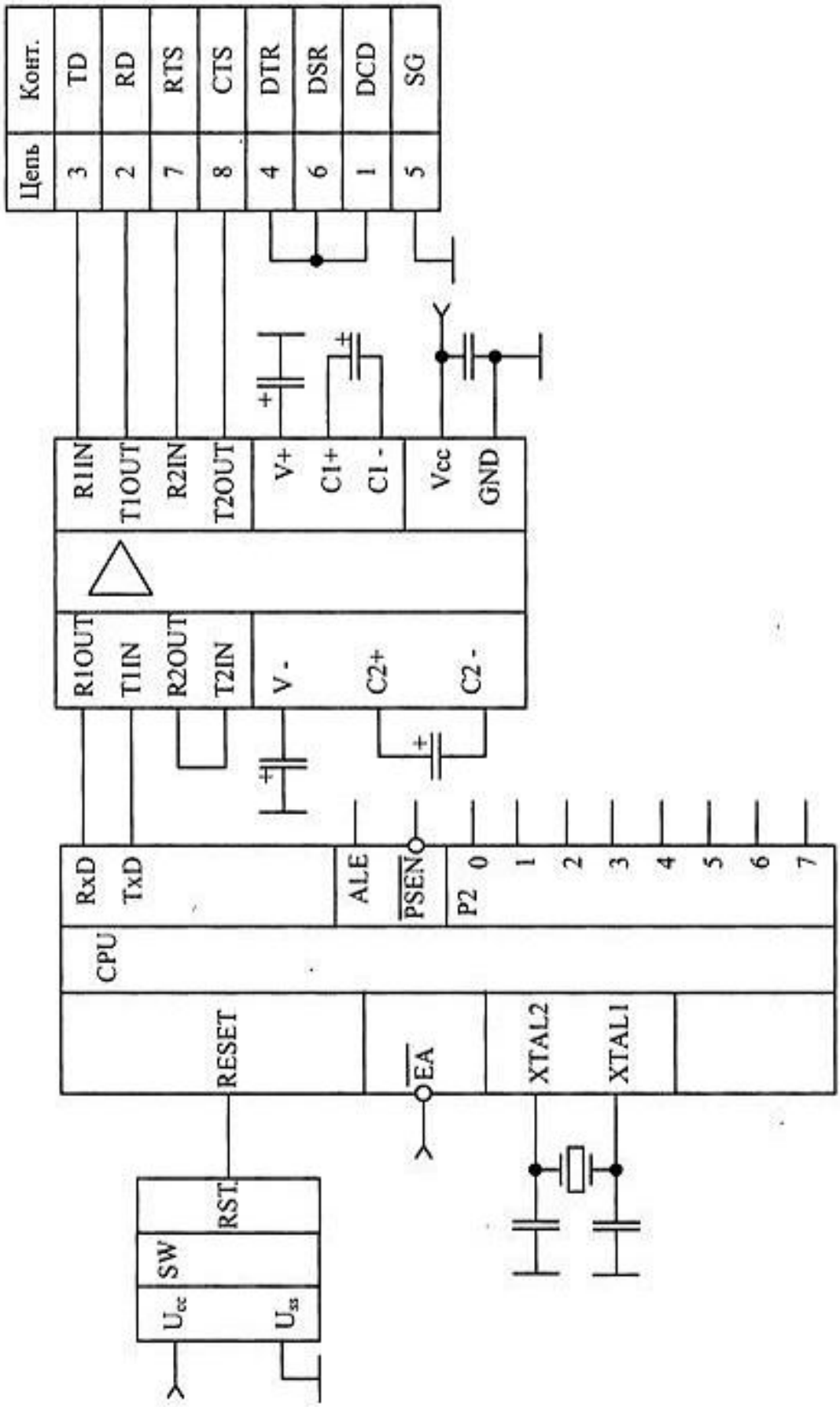


Рис. 21.19. Схема согласования уровней сигналов последовательного порта микроконтроллера и COM-порта компьютера

Листинг 21.31. Подпрограмма инициализации микроконтроллера, обеспечивающая настройку последовательного порта

```
/*--------------------------------------------------------
Подпрограмма инициализации микроконтроллера
--------------------------------------------------------*/
void Init(void)
{TMOD=0x20;        //Настроить таймер 1 на режим работы с автозагрузкой
 TH1=-3;           //Настроить последовательный порт на скорость 9600 бит/с
 TR1=1;            //Включить таймер 1
 SCON=(1<<6)|      //Выбрать восьмибитовый асинхронный режим работы
      (1<<4);      //Разрешить работу приемника
 IE=(1<<7)|         //Разрешить прерывания (установить седьмой бит
                    //регистра IE)
    (1<<4);         //Разрешить прерывания от последовательного порта
}
```

Если к микроконтроллеру не подключены кнопки или датчики, то процессорное время на временные интервалы можно не разделять, и микроконтроллер будет находиться в режиме ожидания, пока подпрограмма ввода не примет через последовательный порт команду от персонального компьютера. В этом случае основная программа будет выглядеть так, как показано в листинге 21.32.

Листинг 21.32. Главная функция программы, принимающей и выполняющей команды от персонального компьютера

```
bit cmPrinjata=0;
void main(void)
{Init();
 while(1)
 {PCON|=1;          //Остановить процессорное ядро и подождать,
                        //пока не будет принят очередной байт
  if(cmPrinjata) //Если прием команды завершен,
  {cmPrinjata=0;    //то сбросить флаг приема команды
   ObrabInf();      //и выполнить команду.
  }
}
```

Теперь рассмотрим, как будет выполняться прием команды от компьютера. В подпрограмме инициализации мы разрешили прерывания после приема байта через последовательный порт, поэтому прием команды будет осуществляться в подпрограмме обработки прерывания.

Листинг 21.33. Подпрограмма приема команд от персонального компьютера через последовательный порт

```
/*-------------------------------------------------------------
Подпрограмма обслуживания прерываний от последовательного порта
-------------------------------------------------------------*/
char cmd[6];
char data* ptr=cmd-1;
void PriemCmd (void) interrupt 4
{if(RI)        //Если последовательный порт принял байт,
 {RI=0;         //то обнулить флаг прерывания от приемника.
  ++ptr;        //Перейти к следующему байту буфера команд
  *ptr=SBUF; //и запомнить в нем принятый байт
  if(*ptr==13) //Если принятый байт это символ конца строки,
   {cmPrinjata=1; //то установить флаг завершения приема команды
    ptr=cmd-1;     //и подготовиться к приему очередной команды
   }
 }
}
```

Для обмена данными с основной программой используется глобальная переменная буфера команд `Cmd` и флаг завершения приема команды `cmPrinjata`. Выберем длину буфера команд равной максимальной длине команды — шесть байтов.

Первое действие этой подпрограммы — обнуление флага запроса прерывания для того, чтобы разрешить дальнейшие прерывания от приемника последовательного порта. Затем принятый байт сохраняется в буфере команд. Для работы с буфером команд используется указатель, в который при запуске программы занесен адрес первого байта буфера команд. При приеме очередного байта указатель переходит к следующему байту буфера команд. Использование указателя позволяет сделать подпрограмму обслуживания прерываниями максимально короткой, а это значит, что ее выполнение будет занимать минимальное время, что и требуется от любой подпрограммы обслуживания прерывания.

Флаг завершения приема команды `cmPrinjata` устанавливается при обнаружении символа возврата каретки. Теперь необходимо подготовиться к приему следующей команды. Для этого в указатель `ptr` записывается адрес начального байта буфера обмена.

Работа с другими источниками прерываний происходит так же, как в рассмотренном примере. Подпрограмма обслуживания прерывания должна осуществить ввод или вывод информации, и не более того! Вся основная обра-

ботка информации будет проводиться в главной программе. Это делается с целью не пропустить очередное прерывание.

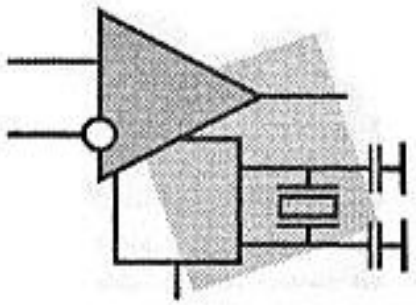
Итоги

В главе были рассмотрены языки программирования, применяющиеся при разработке программ для микроконтроллеров. При этом были рассмотрены преимущества и недостатки использования языков программирования высокого и низкого уровней. Для полноты картины было описано, что такое подпрограммы, виды подпрограмм и как можно ими пользоваться. При этом основной акцент сделан на использовании подпрограмм для структурирования программ. Особое внимание в главе было уделено комментариям к программе и различным способам их использования. Рассмотрено структурное программирование и примеры реализации структурных управляющих конструкций на различных языках программирования.

Особое внимание было уделено методу создания программ сверху вниз. Для этого был рассмотрен пример развития программы от построения программы-прототипа до детализации, достаточной для реализации устройства.

В главе были использованы некоторые интуитивно понятные конструкции языков программирования C и ASM-51. Однако для написания серьезных программ требуются более глубокие знания по этим языкам программирования, поэтому в последующих главах мы остановимся на них подробнее.

ГЛАВА 22



Язык программирования С-51

С — это язык программирования общего назначения, предназначенный для написания программ, эффективных по исполняемому коду, с элементами структурного программирования и богатым набором операторов. Язык программирования С практически не имеет ограничений, что позволяет использовать его для эффективного решения широкого круга задач. Однако при написании программ для микроконтроллеров, принадлежащих к семейству MCS-51, необходимо учитывать особенности построения аппаратуры этих микросхем, поэтому был создан диалект этого языка.

В состав языка программирования С-51 по сравнению со стандартным языком программирования С введен ряд изменений, отображающих особенности построения памяти микроконтроллеров семейства MCS-51. Кроме того, эти изменения позволяют непосредственно обращаться к встроенным портам, таймерам и другим устройствам микроконтроллеров указанного семейства. Особенности микроконтроллеров этого семейства в основном отображаются через описания переменных.

Язык программирования С-51 удовлетворяет стандарту ANSI-C и предназначен для получения компактных и быстродействующих программ, предназначенных для микроконтроллеров семейства MCS-51. Язык С-51 обеспечивает гибкость программирования на широко известном языке С, при скорости работы и компактности, сравнимой с программами, написанными на языке программирования ассемблер.

Так как язык программирования С не имеет собственных средств ввода и вывода, то он обращается к соответствующим функциям операционных систем. В языке программирования С-51 вместо этого имеется возможность изменять библиотечные функции и тем самым обращаться к конкретным ячейкам памяти микроконтроллера, для которого пишется программа.

Компилятор c51.exe — это программное средство, которое транслирует исходный текст, написанный на языке программирования C-51 в перемещаемые объектные модули. Эти модули затем могут объединяться с другими модулями, написанными на языках программирования C-51, PLM-51 или ASM-51. Компилятор выводит на экран дисплея или в файлы листингов сообщения об ошибках и вспомогательную информацию, которая может быть использована при отладке и разработке программ.

Компилятор c51.exe может быть установлен на компьютерах серии IBM или совместимых с ними в операционной системе DOS 3.X и выше и использоваться для генерации команд микроконтроллеров семейства MCS-51.

Применение C-51

Язык программирования C-51 и его библиотеки являются частью интегрированного набора средств разработки программного обеспечения для микроконтроллеров семейства MCS-51. Язык программирования C-51 поддерживает модульное написание программ. Процесс разработки программ на языке программирования C-51 показан на рис. 22.1.

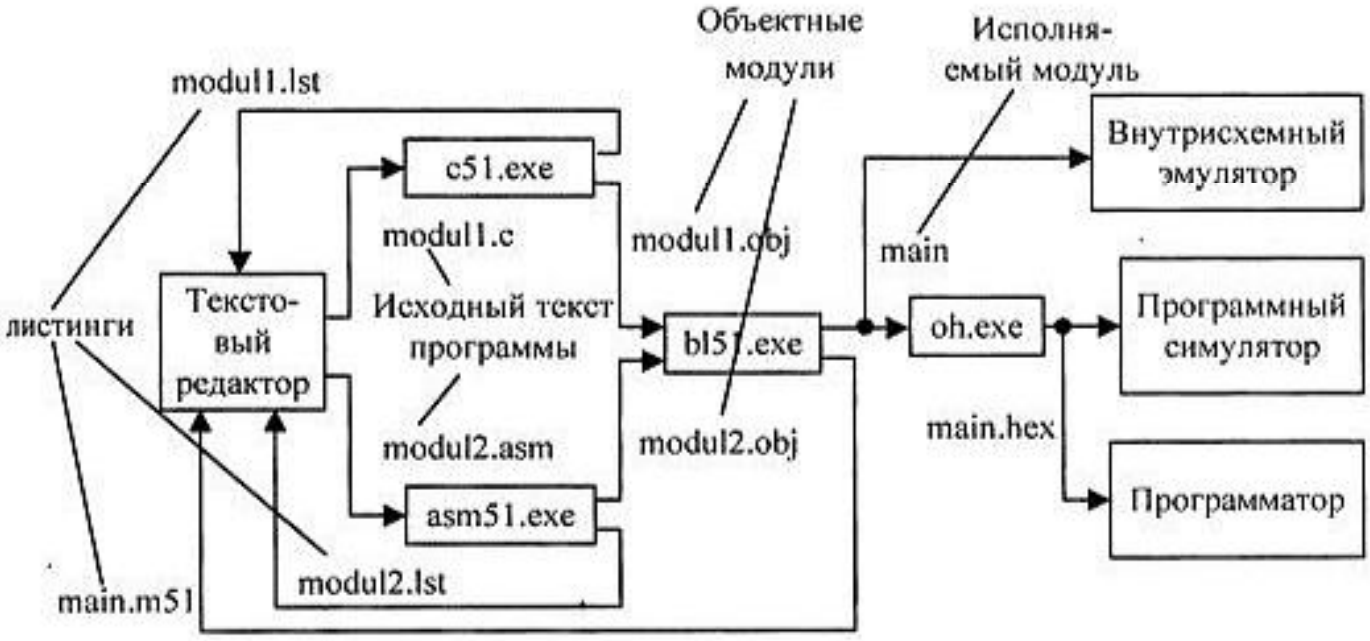


Рис. 22.1. Процесс написания программы на языке C-51

При разработке программного обеспечения выполняются следующие этапы:

- постановка задачи (полное определение решаемой проблемы);
- разработка принципиальной схемы и выбор необходимого программного обеспечения;

- разработка системного программного обеспечения. Этот важный шаг состоит из нескольких этапов, включающих: описание последовательности выполняемых каждым блоком задач, выбор языка программирования и используемых алгоритмов;
- написание текста программы и подготовка к трансляции при помощи любого текстового редактора;
- компиляция программы;
- исправление синтаксических ошибок, выявленных компилятором, в текстовом редакторе с последующей перетрансляцией;
- создание и сохранение библиотек часто используемых объектных модулей при помощи программы lib51.exe;
- связывание полученных перемещаемых объектных модулей в абсолютный модуль и размещение переменных в памяти микроконтроллера при помощи редактора связей bl51.exe;
- создание программы, записываемой в ПЗУ микроконтроллера (загружаемый модуль) в HEX-формате, при помощи программы oh.exe;
- проверка полученной программы при помощи символьного отладчика или других программных или аппаратных средств.

Файл, в котором хранится программа, написанная на языке С-51 (исходный текст программы), называется исходным модулем. Для исходного текста программы принято использовать расширения файла: '*.c'. Исходный текст программы можно написать, используя любой текстовый редактор, однако намного удобнее воспользоваться интегрированной средой программирования, подобной keil-C. В интегрированную среду программирования кроме текстового редактора обычно входят отладчик программ, менеджер проектов и средства запуска программ-трансляторов.

Готовый оттранслированный участок программы обычно хранится на диске в виде файла, записанного в объектном формате. Такой файл называется объектным модулем. Получить объектный модуль можно, указав имя исходного модуля программы в качестве параметра программы-транслятора в DOS-строке или строке командного файла, как это показано в следующем примере:

```
c51.exe modul.c
```

В этом примере в результате трансляции исходного текста программы, содержащегося в файле modul.c, будет получен объектный модуль, который будет записан в файл с именем modul.obj. Как показано на рис. 22.1, объектный модуль не может быть загружен в память программ микроконтроллера. В память микроконтроллера загружается исполняемый модуль.

В интегрированной среде программирования процесс трансляции исходного текста программы проходит намного проще. Для получения объектного модуля достаточно нажать кнопку трансляции файла, как это показано на рис. 22.2.

Рис. 22.2. Кнопка трансляции исходного текста файла

Программа, которая может быть выполнена микроконтроллером, получается после соединения объектных модулей в единый исполняемый модуль. Получить исполняемый модуль программы можно, указав все имена объектных модулей программы в качестве параметров программы редактора связей в DOS-строке или строке командного файла, как это показано в следующем примере:

```
bl51.exe main.obj, modul1.obj, modul2.obj
```

Имя исполняемого модуля программы по умолчанию совпадает с именем первого объектного файла в списке параметров строки запуска редактора связей. Исполняемый модуль программы записывается в файл без расширения. При выполнении приведенной выше в качестве примера командной строки будет получен исполняемый модуль, который будет записан в файл с именем `main`.

В интегрированной среде программирования процесс получения исполняемого модуля не сложнее предыдущего варианта. Для трансляции всего программного проекта достаточно нажать на соответствующую кнопку, как это показано на рис. 22.3.

Большинство программаторов, предназначенных для записи информации в память программ микроконтроллеров, не может работать с объектным форматом исполняемого модуля программы, поэтому для загрузки машинного кода в процессор необходимо преобразовать объектный формат исполняемого модуля в общепринятый для программаторов HEX-формат. При преобразовании форматов вся отладочная информация теряется. Машинный код процессора в HEX-формате называется загрузочным модулем.

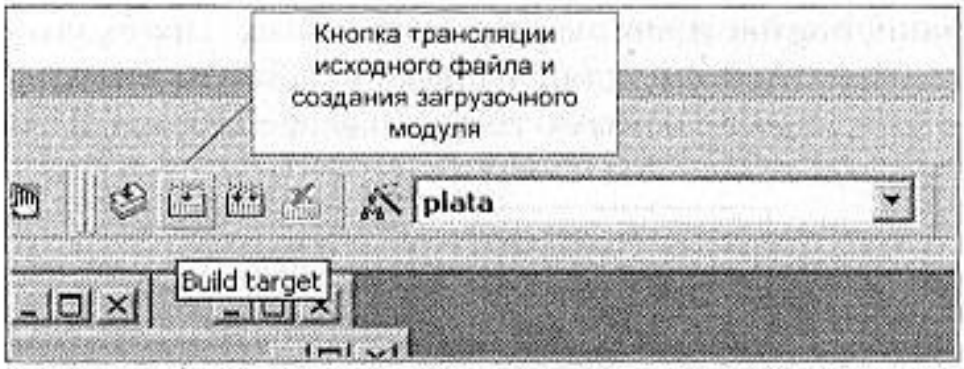


Рис. 22.3. Кнопка получения исполняемого и загрузочного модулей в интегрированной среде программирования keil-c

Загрузочный модуль программы можно получить при помощи программы — преобразователя программы oh.exe, передав ей в качестве параметра имя файла исполняемого модуля программы, например:

```
oh.exe main
```

В результате выполнения этой командной строки будет получен загрузочный модуль программы, который будет записан в файл с именем main.hex.

В интегрированной среде программирования загрузочный файл получается автоматически при выполнении трансляции программного проекта, т. к. интегрированная среда программирования сама выполняет перечисленные выше действия в соответствии с настройками программного проекта.

Отладка программ

После того, как программные модули были успешно оттранслированы, размещены по конкретным адресам и связаны между собой, для отладки программы можно воспользоваться любым из методов, показанных на рис. 22.1:

- внутрисхемным эмулятором;
- встроенным программным отладчиком;
- внешним программным отладчиком;
- отлаживаемым устройством с записанным в память программ двоичным кодом программы.

Внутрисхемный эмулятор с отображением переменных языка программирования на дисплее компьютера оказывает значительную помощь при отладке программ непосредственно на разрабатываемой аппаратуре. Этот метод отладки предоставляет наиболее удобную среду, когда можно непосредственно в отлаживаемом устройстве останавливать программу, контролировать выполнение программы непосредственно по исходному тексту программы, со-

стояние внешних портов и внутренних переменных, как входящих в состав микросхемы, так и объявленных при написании исходного текста программы. Необходимое для отладки программ оборудование показано на рис. 22.4.

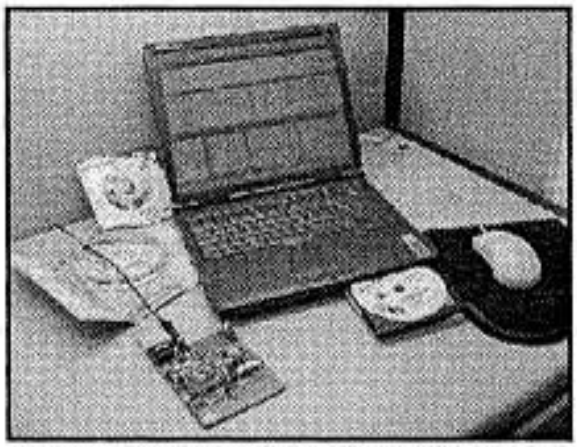


Рис. 22.4. Пример системы отладки программного обеспечения для микроконтроллеров

При отладке программы с использованием внутрисхемного эмулятора необходимо включать в объектные модули символьную информацию. Для этого используются директивы компилятора. (При использовании интегрированной среды программирования достаточно установить соответствующую галочку в свойствах проекта.) В компиляторе языка программирования С-51 возможны следующие действия:

- включение информации о типе переменных для проверки типов при связывании модулей. Эта же информация используется внутрисхемным эмулятором. Исключение информации о переменных пользователя может использоваться для создания прототипов или для уменьшения размера объектного модуля;
- включение или исключение таблиц символьной информации;
- конфигурация вызовов подпрограмм для обеспечения связывания с модулями, написанными на языке программирования ASM-51;
- определение желаемого содержания и формата выходного листинга программы. Распечатка промежуточных кодов на языке ассемблера после компилирования программ, написанных на языке программирования PLM-51. Включение или исключение листингов отдельных блоков исходного текста.

Ну а теперь, прежде чем начать знакомство со средой программирования, рассмотрим основные принципы построения языка программирования С-51 и его отличие от стандартного языка ANSI-C.

Структура программ С-51

Язык программирования С-51 является структурно-модульным языком. Каждый оператор в программе является частью, по крайней мере, одного модуля. Каждая программа, написанная на языке программирования С-51, состоит из одного или более модулей. Каждый модуль записывается в отдельном файле и компилируется отдельно.

В *модуле* помещаются операторы, составляющие программу. Эти операторы объявляют константы или переменные и выполняют необходимые действия. Операторы, выполняющие действия, обязательно должны быть помещены в подпрограммы. Исполнение программы всегда начинается с функции с именем main (т. е. в простейшем случае достаточно написать только эту подпрограмму).

Функция начинается с заголовка подпрограммы, в который входит тип возвращаемой переменной, имя подпрограммы и круглых скобок, внутри которых объявляются переменные-параметры подпрограммы. В подпрограмме main (а также во всех подпрограммах, где не нужно возвращать переменные) вместо типа переменной указывается слово void. Вот уже и начались отличия в программировании микроконтроллеров! В стандартной программе, написанной на языке С, подпрограмма main может быть типа int. Здесь это не так. Ведь программа микроконтроллера никогда не должна завершаться! Исполняемые операторы подпрограммы заключаются в фигурные скобки.

Все переменные и константы, которые будут использоваться в программе, обязательно должны быть объявлены до подпрограммы, где они будут использованы первый раз.

При написании программы для микроконтроллеров всегда необходимо видеть перед глазами принципиальную схему устройства, для которого пишется программа, т. к. схема и программа тесно связаны между собой и дополняют друг друга. Для иллюстрации простейшей программы, написанной на языке программирования С-51, воспользуемся схемой, приведенной на рис. 22.5.

Для примера заставим гореть светодиод VD1. Этот светодиод будет светиться только тогда, когда через него будет протекать ток. Для этого на шестом выводе порта P0 должен присутствовать нулевой потенциал. Запишем его первой же командой нашей программы:

Листинг 22.1. Программа зажигания светодиода VD1

```
#include<reg51.h>
void main(void)
{P0=0;       //Зажигание светодиода
 while(1);  //Бесконечный цикл
}
```

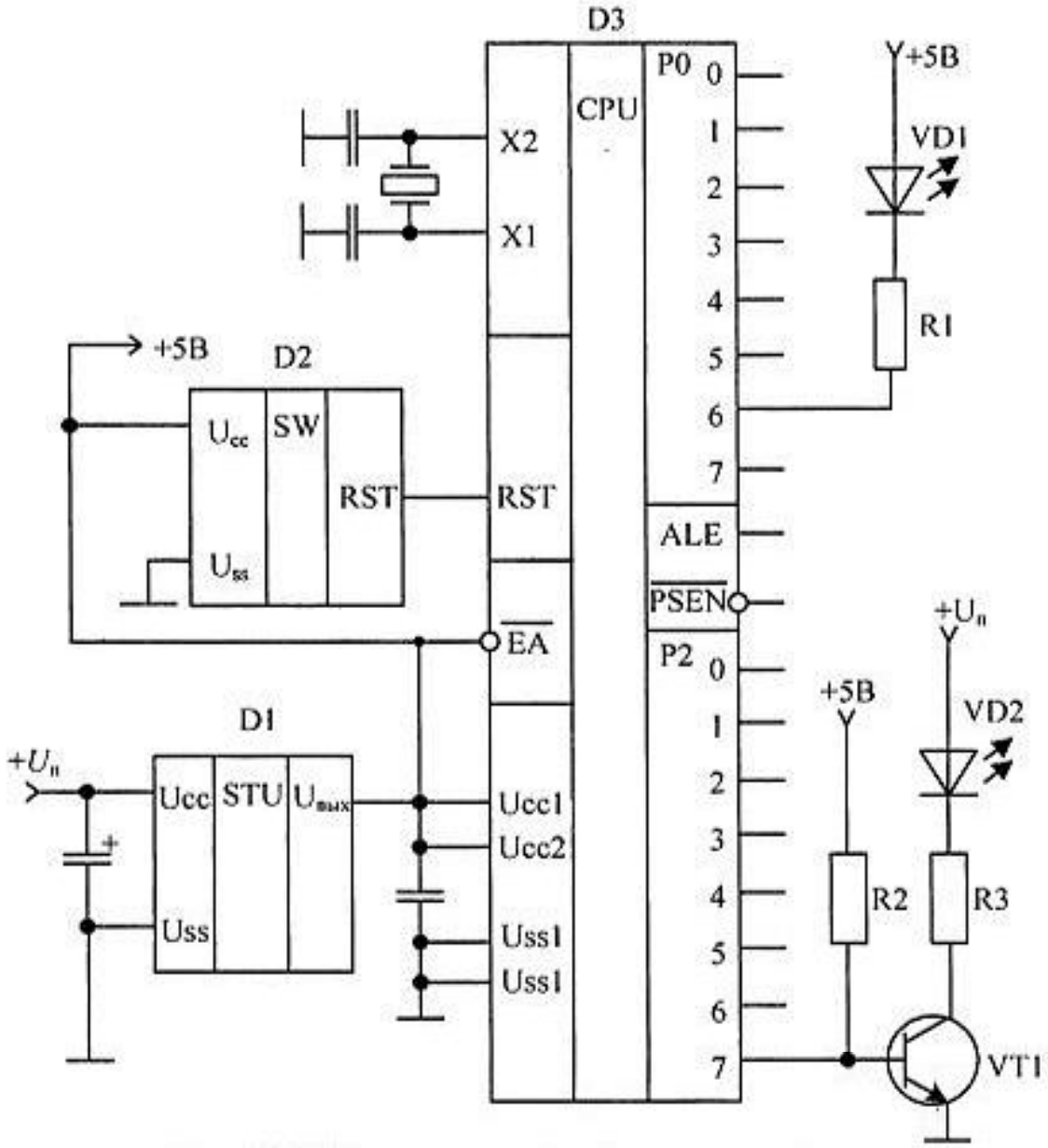


Рис. 22.5. Пример простейшей схемы устройства, построенной с использованием микроконтроллера

Эта программа содержит только один исполняемый оператор. Это оператор присваивания `P0=0;`. Следующий оператор `while(1);` обеспечивает зацикливание программы. Зацикливание программы сделано для того, чтобы микроконтроллер не выполнял больше никаких действий. В противном случае микроконтроллер перейдет к следующей ячейке памяти и будет выполнять команды, которые мы не записывали.

✧ *Обратите внимание*, что язык программирования "знает", по какому адресу памяти данных находится порт P0. Эта информация содержится в строке `#include<reg51.h>`.

Для того чтобы получить более полное представление о структуре программ, написанных на языке программирования С-51, приведем пример исходного текста программы с использованием подпрограмм:

Листинг 22.2. Программа с применением подпрограммы

```
#include<reg51.h>

void svGorit(void)
{P0=0;       //Зажигание светодиода
}

void main(void)
{svGorit(); //Вызов подпрограммы с именем svGorit
 while(1);  //Бесконечный цикл
}
```

В приведенном примере использование подпрограммы никаких преимуществ не дает, но в более сложных программах использование "говорящих" имен переменных может приблизить исходный текст программы к алгоритму и, тем самым, сделать программу более понятной. Это в свою очередь значительно уменьшит время отладки программы.

Символы языка программирования С-51

Как и в любом другом текстовом файле, в исходном тексте программы, написанной на языке программирования С-51, используются символы ASCII или ANSI. Множество символов, используемых в языке программирования С-51, можно разделить на пять групп.

1. *Символы, используемые для образования ключевых слов и идентификаторов* (табл. 22.1). В эту группу входят прописные и строчные буквы английского алфавита, а также символ подчеркивания. Следует отметить, что язык программирования С-51 различает прописные и строчные буквы. Например, идентификаторы start и Start будут считаться различными идентификаторами.

***Таблица 22.1.** Символы, используемые для образования ключевых слов и идентификаторов*

Прописные буквы латинского алфавита	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Строчные буквы латинского алфавита	a b c d e f g h i j k l m n o p q r s t u v w x y z
Символ подчеркивания	_
Арабские цифры	0 1 2 3 4 5 6 7 8 9

2. *Прописные и строчные буквы русского алфавита* (табл. 22.2).

Таблица 22.2. *Прописные и строчные буквы русского алфавита*

Прописные буквы русского алфавита	А Б В Г Д Е Ж З И К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ы Ь Э Ю Я
Строчные буквы русского алфавита	а б в г д е ж з и к л м н о п р с т у ф х ц ч ш щ ъ ы ь э ю я

3. *Знаки нумерации и специальные символы* (табл. 22.3). Эти символы используются для организации процесса вычислений, а также для передачи компилятору определенного набора инструкций.

Таблица 22.3. *Знаки нумерации и специальные символы*

Символ	Наименование	Символ	Наименование
,	Запятая	)	Круглая скобка правая
.	Точка	(	Круглая скобка левая
;	Точка с запятой	}	Фигурная скобка правая
:	Двоеточие	{	Фигурная скобка левая
?	Вопросительный знак	<	Меньше
'	Апостроф	>	Больше
!	Восклицательный знак	[	Квадратная скобка
|	Вертикальная черта	]	Квадратная скобка
/	Дробная черта	#	Номер
\	Обратная черта	%	Процент
~	Тильда	&	Амперсанд
*	Звездочка	^	Исключающее ИЛИ
+	Плюс	=	Равно
–	Минус	"	Кавычки

4. *Управляющие и разделительные символы*. К этой группе символов относятся: пробел, символы табуляции, перевода строки, возврата каретки, символы новой страницы и новой строки. Эти символы отделяют друг от друга лексические единицы языка, к которым относятся ключевые слова,

константы, идентификаторы и т. д. Последовательность разделительных символов (например, последовательность пробелов) рассматривается компилятором как один символ.

5. *Управляющие последовательности*, т. е. специальные символьные комбинации, используемые в функциях ввода и вывода информации. Управляющая последовательность начинается с использования обратной косой черты (\), за которой следует комбинация латинских букв и цифр. Список управляющих последовательностей приведен в табл. 22.4.

***Таблица 22.4.** Управляющие последовательности*

Управляющая последовательность	Наименование	Шестнадцатеричный код
\a	Звонок	007
\b	Возврат на шаг	008
\t	Горизонтальная табуляция	009
\n	Переход на новую строку	00A
\v	Вертикальная табуляция	00B
\r	Возврат каретки	00D
\f	Новая страница	00C
\"	Кавычки	022
\'	Апостроф	027
\0	Ноль-символ	000
\\	Обратная дробная черта	05C
\OOO	Восьмеричный код ASCII- или ANSI-символа	
\xHHH	Шестнадцатеричный код ASCII- или ANSI-символа	HHH

Управляющие последовательности `\ooo` и `\xHHH` (здесь `o` обозначает восьмеричную цифра; `H` — шестнадцатеричную цифру) позволяют представить символ из кодовой таблицы ASCII или ANSI как последовательность восьмеричных или шестнадцатеричных цифр соответственно. Например, символ возврата каретки может быть представлен следующими способами:

```
\r — управляющая последовательность,
\015 — восьмеричный код символа возврата каретки,
\x00D — шестнадцатеричный код символа возврата каретки.
```

Следует отметить, что в строковых константах всегда обязательно задавать все три цифры управляющей последовательности. Например, отдельную управляющую последовательность \n (переход на новую строку) можно представить как \010 или \xA, но в строковых константах необходимо задавать все три цифры, в противном случае символ или символы, следующие за управляющей последовательностью, будут рассматриваться как ее недостающая часть. Например:

```
"ABCDE\x009FGH"
```

Данная строковая команда будет напечатана с использованием определенных функций языка С, как два отдельных слова ABCDE и FGH, разделенные табуляцией, в этом случае если указать неполную управляющую строку "ABCDE\x09FGH", то при печати появится строка ABCDEЯGH, т. к. компилятор воспримет последовательность \x09F как символ "Я".

Отметим тот факт, что если обратная дробная черта предшествует символу, не являющемуся управляющей последовательностью (т. е. не включенному в табл. 22.4) и не являющемуся цифрой, то эта черта игнорируется, а сам символ представляется как литеральный. Например:

```
символ \h представляется символом h в строковой или символьной константе
```

Кроме определения управляющей последовательности, символ обратной дробной черты (\) используется так же, как символ продолжения. Если за (\) следует символ возврата каретки, то оба символа игнорируются, а следующая строка является продолжением предыдущей. Это свойство может быть использовано для записи длинных строк. Например:

```
printf("Это очень длинная \
строка")
```

Компилятор С-51 выдает сообщение об ошибке, если в тексте исходной программы встречается символ, отличающийся от символов, перечисленных выше.

Лексические единицы, разделители и использование пробелов

Наименьшей единицей операторов С-51 является лексическая единица. Каждая из лексических единиц относится к одному из классов:

- идентификаторы;
- ключевые слова;
- простые ограничители (все специальные символы, кроме _, являются простыми ограничителями);

- составные ограничители (они образуются посредством определенных комбинаций двух спецсимволов, а именно: `!=`, `+=`, `-=`, `*=`, `<<`, `>>`, `<=`, `>=`, `/*`, `*/`, `//`);
- числовые константы;
- текстовые строковые константы.

В большинстве случаев вполне очевидно, где заканчивается одна лексическая единица и начинается следующая. Например, в операторе присваивания:

```
X=AP*(FT-3)/A;
```

`X`, `AP`, `FT`, `A` — являются идентификаторами переменных;

3 — числовой константой;

все прочие символы — простыми ограничителями.

Ключевые слова, идентификаторы и числовые константы должны обязательно отделяться друг от друга. Если между двумя идентификаторами, числовыми константами или ключевыми словами не может быть указан простой или составной ограничитель, то в качестве разделителя между ними должен вставляться символ пробела. Для улучшения читабельности программы вместо одного символа пробел может использоваться несколько символов пробела.

Идентификаторы

Идентификаторы в языке программирования C-51 используются для определения имени переменной, подпрограммы, символической константы или метки оператора. Длина идентификатора может достигать 255 символов, но транслятор различает идентификаторы только по первым 31 символам.

Возникает вопрос — а зачем тогда нужен такой длинный идентификатор? Ответ: для создания "говорящего" имени подпрограммы или переменной, которое может состоять из нескольких слов. Например:

```
ProchitatPort();      //Прочитать порт
VklychitIndikator(); //Включить индикатор
```

В приведенном примере подпрограмма `ProchitatPort` выполняет действия, необходимые для чтения порта, а подпрограмма `VklychitIndikator` выполняет действия, необходимые для зажигания индикатора. Естественно, что намного легче прочитать действие подпрограммы непосредственно из имени подпрограммы, чем каждый раз анализировать алгоритм программы или искать ее исходный текст для того, чтобы в очередной раз разобраться — что же она

выполняет? Для этого при объявлении имени подпрограммы можно потратить количество символов и большее чем 31!

То же самое можно сказать и про имена переменных. Например:

```
sbit ReleVklPitanija = 0x80; /*К нулевому выводу порта P0 подключено реле
                               включения питания*/
sbit svDiod = 0x81;          /*К первому выводу порта P0 подключен
                               светодиод*/
sbit DatTemperat = 0x82;     /*Ко второму выводу порта P0 подключен
                               датчик температуры*/
```

В приведенном примере каждой ножке микроконтроллера назначается переменная с именем, отображающим устройство, подключенное к этой ножке. В результате при чтении программы не потребуется обращаться к принципиальной схеме устройства каждый раз, как только производится операция записи или чтения переменной, связанной с портами микроконтроллера. (Разбираться с принципиальной схемой занятие не менее "увлекательное" по сравнению с поиском неизвестной и неведомо что выполняющей подпрограммы.)

В качестве идентификатора может быть использована любая последовательность строчных или прописных букв латинского алфавита и цифр, а также символов подчеркивания '_'. Идентификатор может начинаться только с буквы или символа '_', но ни в коем случае с цифры. Это позволяет программе-транслятору различать идентификаторы и числовые константы. Строчные и прописные буквы в идентификаторе различаются. Например: идентификаторы `abc` и `ABC`, `A128B` и `a128b` воспринимаются как разные.

Идентификатор создается при объявлении переменной, функции, структуры и т. п., после этого его можно использовать в последующих операторах разрабатываемой программы. Необходимо обратить внимание на следующие важные особенности при определении идентификатора.

- Идентификатор не должен совпадать с ключевыми словами, с зарезервированными словами и именами функций из библиотеки компилятора языка С.
- Следует обратить особое внимание на использование символа подчеркивания (_) в качестве первого символа идентификатора, поскольку идентификаторы, построенные таким образом, могут совпадать с именами системных функций или переменных, в результате чего они станут недоступными.

Следует отметить, что никто не запрещает объявлять идентификатор, совпадающий с именами функций из библиотек компилятора языка С. Однако после объявления такого идентификатора вы не сможете обратиться к функции с таким же именем никаким образом.

Примеры правильных идентификаторов:

```
A
XYR_56
OpredKonfigPriem
Byte_Prinjat
SvdiodGorit
```

Ключевые слова

Ключевые слова — это зарезервированные слова, которые используются для построения операторов языка.

Список ключевых слов:

_at_	alien	auto	bdata	bit	break	case
char	code	compact	const	continue	data	default
do	double	else	enum	extern	far	float
for	goto	idata	if	int	interrupt	large
long	pdata	_priority_	reentrant	register	return	sbit
sfr	sfr16	short	signed	sizeof	small	static
struct	switch	typedef	_task_	union	unsigned	using
void	volatile	while	xdata			

✧ *Обратите внимание*, что ключевые слова не могут быть использованы в качестве идентификаторов.

Константы

Константы предназначены для введения чисел в состав выражений операторов языка программирования С. В отличие от идентификаторов, всегда начинающихся с буквы, константы всегда начинаются с цифры. В языке программирования С-51 разделяют четыре типа констант:

- целые знаковые и беззнаковые константы;
- константы с плавающей запятой;
- символьные константы;
- литеральные строки.

Целочисленные константы могут быть записаны как восьмеричные, десятичные или шестнадцатеричные числа в зависимости от того, какая система счисления удобнее для представления константы. Константа может быть представлена в десятичной, восьмеричной или шестнадцатеричной форме. При выполнении вычислений обычно пользуются десятичными константами.

При работе с ножками микроконтроллера или передаче двоичных данных удобнее пользоваться двоичными числами или их более короткой формой записи — восьмеричными или шестнадцатеричными числами.

- *Десятичная константа* состоит из одной или нескольких десятичных цифр, причем первая цифра не может быть нулем (иначе число будет воспринято как восьмеричное).
- *Восьмеричная константа* состоит из обязательного нуля и одной или нескольких восьмеричных цифр (среди цифр должны отсутствовать цифры восемь и девять, т. к. эти цифры не входят в восьмеричную систему счисления). Если константа содержит цифру, недопустимую в восьмеричной системе счисления, то константа считается ошибочной.
- *Шестнадцатеричная константа* начинается с обязательной последовательности символов 0x или 0X и содержит одну или несколько шестнадцатеричных цифр (`0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F`).

Примеры целых констант:

Десятичная константа	Восьмеричная константа	Шестнадцатеричная константа
16	020	0x10
127	0117	0x2B
240	0360	0XF0

Если требуется сформировать отрицательную целую константу, то используют знак '–' перед записью константы (который будет называться унарным минусом). Например: –0x2A, –088, –16.

Каждой целой константе присваивается тип, определяющий преобразования, которые должны быть выполнены, если константа используется в выражениях. Тип константы определяется следующим образом:

- десятичные константы рассматриваются как знаковые числа, и им присваивается тип `int` (целая) или `long` (длинная целая) в соответствии со значением константы. Если константа меньше 32 768, то ей присваивается тип `int`, в противном случае `long`;
- восьмеричным и шестнадцатеричным константам присваивается тип `int`, `unsigned int` (беззнаковая целая), `long` или `unsigned long` в зависимости от значения константы, согласно табл 22.5.

Иногда требуется с самого начала интерпретировать константу как длинное целое число. Для того чтобы любую целую константу определить типом `long`, достаточно в конце константы поставить букву "l" или "L". Пример:

```
5l, 6l, 128L, 0105L, 0X2A11L.
```

Таблица 22.5. Восьмеричные и шестнадцатеричные константы и их типы

Диапазон шестнадцатеричных констант	Диапазон восьмеричных констант	Тип
0x0 — 0x7FFF	0 — 077777	int
0X8000 — 0XFFFF	0100000 — 0177777	unsigned int
0X10000 — 0X7FFFFFFF	0200000 — 017777777777	long
0X80000000 — 0XFFFFFFFF	020000000000 — 037777777777	unsigned long

Примеры синтаксически недопустимых целочисленных констант:

12AF — шестнадцатеричная константа не имеет символов 0x в начале константы, поэтому по умолчанию для нее принимается десятичная система счисления, но тогда в ней присутствуют недопустимые символы.

0x2ADG — символ G недопустим при записи шестнадцатеричных чисел.

Константа с плавающей запятой — это десятичное число, представленное в виде действительного числа с десятичной запятой и порядком числа. Формат записи константы с плавающей запятой:

```
[ цифры ].[ цифры ] [ E|e [+|-] цифры ].
```

Число с плавающей запятой состоит из целой и дробной части и (или) порядка числа. Для определения отрицательного числа необходимо сформировать константное выражение, состоящее из знака минуса и положительной константы. Примеры записи констант с плавающей запятой:

```
115.75, 1.5E-2, -0.025, .075, -0.85E2
```

Символьная константа — представляется символом ASCII или ANSI, заключенном в апострофы. Управляющая последовательность тоже может быть использована в символьных константах. При этом она рассматривается как одиночный символ. Значением символьной константы является числовой код символа. Примеры символьных констант:

```
' '  — пробел ,
'Q'  — буква Q ,
'\n' — символ новой строки ,
'\\' — обратная дробная черта ,
'\v' — вертикальная табуляция.
```

Символьные константы имеют тип `int` и при преобразовании типов дополняются знаком. Символьные константы используются обычно при управлении микроконтроллерным устройством от клавиатуры.

Пример использования символьной константы на языке программирования С-51 приведен ниже:

```
if(NajKn=='p')VklUstr();//Если нажата кнопка p, то включить устройство
```

В этом примере если в переменной NajKn содержится код, соответствующий букве "p", то будет выполнена подпрограмма VklUstr.

Строковые константы. Если символьные константы используются обычно при вводе информации с клавиатуры, то при отображении информационных сообщений обычно используются целые строки символов. В строке допускается использование пробелов.

Строковая константа (литерал или литеральная строка) — это последовательность символов (включая строковые и прописные буквы русского и латинского алфавита, а также цифры), заключенных в кавычки ("). Например:

```
"Школа N 35", "город Тамбов", "YZPT КОД"
```

Отметим, что все управляющие символы, кавычка ("), обратная дробная черта (\) и символ новой строки в литеральной строке и в символьной константе представляются соответствующими управляющими последовательностями. Каждая управляющая последовательность представляет собой один символ. Например, при печати литеральной строки "Школа \n N 35" его часть "Школа" будет напечатана на одной строке, а вторая часть "N 35" — на следующей.

Символы литеральной строки обычно хранятся в памяти программ, но могут храниться и в памяти данных. В конец каждой литеральной строки компилятором добавляется нулевой символ, который можно записать как: "\0". Именно этот символ и является признаком конца строки.

Литеральная строка рассматривается как массив символов (char[ ]). Отметим важную особенность: число элементов массива равно числу символов в строке плюс 1, т. к. нулевой символ (символ конца строки) также является элементом массива. Все литеральные строки рассматриваются компилятором как различные объекты.

Одна литеральная строка может выводиться на дисплей как несколько строк. Такие строки разделяются при помощи обратной дробной черты и символа возврата каретки \n. На одной строке исходного текста программы можно записать только одну литеральную строку. Если необходимо продолжить написание одной и той же литеральной строки на следующей строке исходного текста программы, то в конце строки исходного текста можно поставить обратную строку. Например, исходный текст:

```
"строка неопределенной \
длины"
```

полностью идентичен литеральной строке:

```
"строка неопределенной длины".
```

Однако более удобно для объединения литеральных строк использовать символ (символы) пробела. Если в программе встречаются два или более литерала, разделенные только пробелами или символами табуляции, то они будут рассматриваться как одна литеральная строка. Этот принцип можно использовать для формирования литералов, занимающих более одной строки.

Выражения в операторах языка программирования С-51

Выражением называется комбинация знаков операций и операндов, результатом которой является определенное значение. Знаки операций определяют действия, которые должны быть выполнены над операндами. Каждый операнд в выражении в свою очередь может быть выражением. Значение выражения зависит от расположения знаков операций и круглых скобок в выражении, а также от приоритета выполнения операций. Примеры выражений:

```
A+B
A*(B+C)-(D-E)/F
```

Выражение в языке программирования С-51 состоит из операндов, которые комбинируются при помощи различных арифметических или логических операций, а также операций отношения. Над переменными-указателями возможно проведение адресных операций.

Операндом в выражении может быть переменная, числовая константа, подпрограмма-функция или указатель. Любой операнд, который имеет константное значение, называется константным выражением. Каждый операнд имеет тип.

Если в качестве операнда используется константа, то ему соответствует значение и тип представляющей его константы. Целая константа может быть типа `int, long, unsigned int, unsigned long`, в зависимости от ее значения и от формы записи. Символьная константа имеет тип `int`. Константа с плавающей точкой всегда имеет тип `float`.

При вычислении выражений тип каждого операнда может быть преобразован к другому типу. Преобразования типов могут быть неявными, при выполнении операций и вызовов функций, или явными, при выполнении операций приведения типов. Из-за того, что неявные преобразования типов могут различаться для трансляторов разных фирм, то при написании программы лучше использовать явное преобразование типов переменных.

Примеры явных преобразований типов операндов:

```
a=(int)b+(int)c;      /*Переменные a и b могут быть восьмиразрядными.
                        Преобразование типов нужно, чтобы избежать
                        переполнения*/
s=sin((float)a/15)); /*Если не преобразовать тип переменной a, то деление
                        будет целочисленным и результат деления будет
                        равен нулю.*/
```

В выражениях в качестве операндов могут использоваться подвыражения. Подвыражение — это обычное выражение, заключенное в скобки. Подвыражения могут использоваться для группировки частей выражения, точно так же, как и в обычной алгебраической записи. Использование подвыражений позволяет сократить количество операторов в программе, а значит, и объем исходного текста программы (но не объем исполняемой программы), но затрудняет отладку этой программы.

В языке программирования С-51 используются *арифметические операции*, результат которых зависит от типа операндов:

1. + — суммирование.
2. – — вычитание.
3. * — умножение.
4. / — деление.
5. % — вычисление остатка от целочисленного деления.

Примеры выражений, использующие арифметические операции:

```
A+B
A+B-C
A*T+F/D
A*(B+C)-(D-E)/F
```

В языке программирования С-51 также определено несколько одноместных арифметических операций:

1. '-' — изменение знака операнда на противоположное значение.
2. '+' — знак плюс не влияет на значение операнда.
3. '++' — увеличение значения операнда на единицу.
4. '--' — уменьшение значения операнда на единицу.

Для одноместной операции требуется один операнд, которому она предшествует. Например:

```
P3 = -5;    //Присвоить порту P3 значение числа -5
a = -b;     //Присвоить переменной a отрицательное значение переменной b
```

```
c=++a + 2; //Увеличить значение переменной a на 1 и прибавить 2
c=a++ + 2; /*Прибавить к переменной a 2, присвоить это значение к
             переменной c и только после этого.увеличить значение
             переменной a на 1.*/
```

Над операндами можно осуществлять *логические операции*:

1. `'&&'` — логическое "и".
2. `'&'` — побитовое логическое "и".
3. `'||'` — логическое "или".
4. `'|'` — побитовое логическое "или".
5. `'^'` — "исключающее или" (суммирование по модулю два).

Здесь следует объяснить различие между логическими и побитовыми логическими операциями. Дело в том, что в стандартном языке ANSI C не существует битовых переменных. Для хранения битовых значений истина '1' и ложь '0' используются стандартные целые типы переменных. В простейшем случае это байт. При этом все значения переменной, отличающиеся от 0, считаются 1. Например, пусть в переменной `a` хранится число 5, а в переменной `b` — число 6. Тогда:

```
a|b=7     //00000101 or 00000110 = 00000111
a||b=1    /*(00000101)->1 (00000110)->1; (1 or 1= 1),
              Результат равен 00000001*/
a&b=4     //00000101 and 00000110 = 00000100
a&&b=1    /*(00000101)->1 (00000110)->1; (1 and 1= 1),
              Результат равен 00000001*/
a^b=3     //00000101 xor 00000110 = 00000011
```

В языке программирования C-51 также определено несколько одноместных логических операций:

1. '!' — инверсия операнда.
2. '~' — побитовая инверсия операнда.

Например, пусть в переменной `a` хранится число 5. Тогда:

```
a=~a;  //~00000101 = 11111010 = 250
a=~a;  //~11111010 =  00000101 = 5
a=!a;  //(00000101)->1; ~1 = 0
a=!a;  // ~0 = 1
```

В условном операторе и операторах цикла используются *операции отношения*:

1. < — меньше.
2. > — больше.

3. <= — меньше или равно.

4. >= — больше или равно.

5. == — равно.

6. != — не равно.

Если указанное отношение между операндами верно, то результат равен 1, иначе 0. Например, если d = 7, то:

```
 (d > 5) результат будет 1 (истина)
(d = 4) результат будет 0 (ложь)
```

Над переменными-указателями возможно проведение *адресных операций.*

1. '*' — операция косвенной адресации.

2. '&' — вычисление адреса переменной.

Операция косвенной адресации '*' осуществляет доступ к переменной при помощи указателя. Результатом операции является значение переменной, на которую указывает операнд. Типом результата является тип переменной, адресуемой указателем. При работе с указателями необходимо быть предельно осторожными, т. к. если указатель содержит недопустимый адрес, то результат операции чтения или записи будет непредсказуем и может привести к выходу проектируемого устройства из строя.

Операция вычисления адреса переменной '&' дает адрес ячейки памяти своего операнда. Операндом может быть любой идентификатор. Имя функции или массива также может быть операндом операции вычисления адреса переменной, хотя в этом случае знак операции вычисления адреса переменной является лишним, т. к. имена массивов и функций и так являются адресами. Операция вычисления адреса переменной не может применяться к элементам структуры, являющимся полями битов, и к объектам с классом памяти register.

Пример использования адресных операций при работе с указателем:

```
int t, f=0,  *adress;
adress = &t /*переменной adress, объявленной как указатель,
              присваивается адрес переменной t */
*adress =f; /*переменной, находящейся по адресу, содержащемуся в
              переменной adress, присваивается значение переменной f,
              т. е. 0, что эквивалентно t=f; т. е. t=0;*/
```

Может возникнуть вопрос: а зачем тогда нужны указатели, если можно прекрасно обойтись обычным присваиванием переменной? Использование указателей очень удобно при написании подпрограмм. Ведь одни и те же действия подпрограмма должна выполнять над различными участками памяти!

Конкретный адрес переменной будет передан в подпрограмму при ее вызове. Рассмотрим пример вывода информации через последовательный порт:

Листинг 22.3. Вывод информации через последовательный порт

```
void PutNadp(char code *c)//Объявлен указатель с на символ в памяти
                          //программ
{do{while(!TI);           //Подождать готовности последовательного
                          //порта
    TI=0;
    SBUF=*c++;            //Передать очередной символ
 }while(*c!=0);           //Если передан последний символ строки,
}                         //то выйти из подпрограммы
...
PutNadp("привет!");       //Вывод одной строки
...
PutNadp("Вася!");         //Вывод второй строки
```

Приоритеты выполнения операций

В языке С-51 операции с высшими приоритетами вычисляются первыми. Наивысшим приоритетом является приоритет, равный 1. Приоритеты и порядок операций приведены в табл. 22.6. Порядок вычисления выражения следующий: сначала выполняются операторы в круглых скобках, в них от старшего приоритета к младшему, а среди равнозначных операторов — слева направо.

***Таблица 22.6.** Приоритеты и порядок операций*

Приоритет	Знак операции	Типы операции	Порядок выполнения
2	`() [] . ->`	Выражение	Слева направо
1	`- ~ ! * & ++ -- sizeof` операции приведения типов	Унарные	Справа налево
3	`* / %`	Мультипликативные	Слева направо
4	`+ -`	Аддитивные	
5	`<< >>`	Сдвиг	
6	`< > <= >=`	Отношение	
7	`== !=`	Отношение (равенство)	
8	`&`	Поразрядное И	

Таблица 22.6 (окончание)

Приоритет	Знак операции	Типы операции	Порядок выполнения
9	^	Поразрядное исключающее ИЛИ	Слева направо
10	|	Поразрядное ИЛИ	
11	&&	Логическое И	
12	||	Логическое ИЛИ	
13	? :	Условная	
14	= *= /= %= += -= &= |= >>= <<= ^=	Простое и составное присваивание	Справа налево
15		Последовательное вычисление	Слева направо

Операторы языка программирования С-51

В языке программирования С-51 используется два типа операторов: операторы объявления и выполняемые операторы. Все операторы С-51 заканчиваются точкой с запятой.

Операторы объявления

Объявление является неисполняемым оператором, который объявляет некоторый объект или набор объектов, связывает с ним один или несколько идентификаторов и, если это необходимо, распределяет память микроконтроллера.

В объявлении могут быть объявлены три типа объектов: переменные, метки и функции. В программе для каждого имени допустимо только одно объявление. Метка полностью объявлена, если она стоит перед выполняемым оператором и заканчивается двоеточием ':'. Например:

```
vfg: svGorit(); //Вызов подпрограммы с именем svGorit
```

Переменные, константы, литералы и подпрограммы должны быть объявлены раньше, чем они будут использоваться в исполняемом операторе. Объявление переменной заключается в задании типа этой переменной и ее имени. Тип переменной записывается перед именем переменной. Например:

```
char Rejim; //Переменная, хранящая номер режима размером в один байт
int Schet;  //Переменная, использующаяся как счетчик размером в два байта
```

В языке программирования С-51 нет необходимости знать конкретный адрес переменной, достаточно обратиться к ней по имени `Rejim` или `Schet`.

Функция объявляется точно так же, как и переменные. То есть перед именем подпрограммы-функции записывается тип переменной, которую возвращает подпрограмма после своего завершения. Так как в подпрограмму в свою очередь тоже могут передаваться переменные, которые называются параметрами подпрограммы, то эти переменные записываются в скобках после имени подпрограммы-функции. Перед каждой переменной обязательно указывается ее тип. Например:

```
char PrinjatByte(void);  //Подпрограмма, предназначенная для приема
одного байта.
```

Так как подпрограмме приема обычно не требуется никаких дополнительных переменных, то вместо переменной-параметра подпрограммы указано слово `void`, обозначающее, что у подпрограммы-функции нет параметров.

Исполняемые операторы

- Оператор присваивания
- Условный оператор
- Структурный оператор
- Оператор цикла `for`
- Оператор цикла с проверкой условия до тела цикла
- Оператор цикла с проверкой условия после тела цикла
- Оператор `break`
- Оператор `continue`
- Оператор выбора
- Оператор безусловного перехода
- Оператор выражения
- Оператор возвращения из подпрограммы
- Пустой оператор

При вычислении операторов используются выражения, в состав которых входят одноместные, двухместные и трехместные операции.

Оператор присваивания

Оператор присваивания записывается в виде:

```
Переменная=выражение;
```

Выражение вычисляется, и полученное значение присваивается переменной. Например:

Листинг 22.4. Примеры оператора присваивания

```
P0=2;          //Установить начальные потенциалы на ножках второго порта
               // микроконтроллера
a=cos(b*5); //Этот оператор присваивания осуществляет вызов
               //подпрограммы-функции.
```

Достаточно часто требуется изменять значение какой-либо переменной. То есть, и источником и приемником данных служит одна и та же переменная. В этом случае можно воспользоваться составным оператором присваивания. Использование составного оператора сокращает исходный текст программы. Например:

```
sum+=3;     //Оператор эквивалентен оператору sum=sum+3;
Umensh-=5; //Оператор эквивалентен оператору Umensh=Umensh-5;
a*=10;      //Оператор эквивалентен оператору a=a*5;
mask&=0x10;/* Оператор эквивалентен оператору mask=mask&5;
              Обычно используется для записи нулей в определенные биты
              переменной */
```

Условный оператор

Оператор `if` обеспечивает условное выполнение операторов. Он записывается в следующей форме:

```
if(<выражение>)
    <operator-1>;
  [else
    <operator-2>;]
```

При этом ключевое слово `else` со следующим за ним исполняемым оператором представляют собой необязательную часть условного оператора. Если результат вычисления выражения равен 1 (истина), то выполняется `operator-1`. Если результат вычисления выражения равен 0 (ложь), то выполняется `operator-2`. Если выражение ложно и отсутствует `оператор-2`, то выполняется оператор, следующий за условным. Пример записи условного оператора:

```
if(Wes<Min)              /*Условная операция*/
  Schetch=Schetch+1;     /*Плечо 1*/
 else
  Schetch=0;             /*Плечо 2*/
```

В качестве условия могут быть использованы не только операции отношения. Напомню, что операции отношения вычисляют битовую величину. Поэтому в условном операторе могут быть использованы любые битовые переменные, в том числе ножки портов и подпрограммы-функции, возвращающие битовое значение. Использование битовых переменных и подпрограмм позволяет увеличить читаемость исходного текста программы. Примеры использования битовых переменных и подпрограмм-функций в условном операторе:

```
if(P1.5)fnKn5Naj();          /* В этом примере предполагается, что к
                                пятому выводу порта P1 подключена кнопка
                                с надписью "5"*/
if(Kn5Naj)fnKn5Naj();        /* Этот пример эквивалентен предыдущему, но
                                выводу P1.5 поставлена в соответствие
                                переменная Kn5Naj*/
if(PrinjatByte())DecodCmd(); /* Предполагается, что функция PrinjatByte
                                возвращает значение '1', если байт
                                принят*/
```

В приведенном примере использована подпрограмма-функция, осуществляющая прием байта из последовательного порта. В названиях подпрограмм отображены действия, выполняемые этими подпрограммами.

Структурный оператор {}

Существует два основных способа использования структурного оператора:

1. Структурный оператор может рассматриваться в качестве отдельного оператора языка C-51 и использоваться в программе везде, где может встречаться отдельный исполняемый оператор. Это применяется в операторах `for`, `while`, `do while`, `switch of` и `if`.
2. Структурный оператор ограничивает область действия локальных переменных.

Каждый оператор внутри структурного оператора может являться любым оператором языка C-51, в том числе и объявлением, при условии, что все объявления внутри структурного оператора должны быть выполнены до первого исполняемого оператора.

Структурный оператор начинается с открывающей скобки '{' и записывается в следующем виде:

```
{<operator-1>;  //Здесь могут быть объявления переменных
 <operator-2>;
    ...
 <operator-n>;
}
```

Заметим, что в конце составного оператора точка с запятой не ставится. Пример использования структурного оператора:

Листинг 22.5. Пример структурного оператора

```
if(Wes<Min)       /*Условная операция*/
  {incr=incr*2;          /*Структурный оператор*/
   Schetch=Schetch+1;  /*Содержит два оператора*/
  }
```

Структурные операторы могут вкладываться друг в друга:

```
{<operator-1>;
 <operator-2>;
 {<operator-A>;
  <operator-B>
  <operator-C>;
  }
 <operator-3>;
 <operator-4>;
}
```

Пример использования объявлений внутри вложенного структурного оператора:

Листинг 22.6. Пример объявлений в структурном операторе

```
int main ()
 {int q,b;
  float t,d;
  ...
  if(...)
    {int e,g;
     float f,q;
     ...
    }
  ...
  return (0);
 }
```

Переменные `e`, `g`, `f`, `q` будут уничтожены после выполнения составного оператора. Отметим, что переменная `q` является локальной в составном операторе, т. е. она никоим образом не связана с переменной `q`, объявленной в начале функции `main` с типом `int`. Отметим так же, что выражение, стоящее после

return, может быть заключено в круглые скобки, хотя наличие последних необязательно.

Оператор цикла *for*

Оператор for — это наиболее общий способ организации цикла. Оператор цикла for записывается в следующей форме:

```
for ( выражение 1 ; выражение 2 ; выражение 3 ) тело цикла;
```

Выражение 1 обычно используется для установления начального значения переменных, управляющих циклом. Выражение 2 — это выражение, определяющее условие, при котором тело цикла будет выполняться. Выражение 3 определяет изменение переменных, управляющих циклом после каждого выполнения тела цикла. В качестве тела цикла может служить любой исполняемый оператор языка С-51, в том числе и составной оператор. Внутри составного оператора может быть заключено любое количество исполняемых операторов.

Схема выполнения оператора for:

1. Вычисляется выражение 1.
2. Вычисляется выражение 2.
3. Если значения выражения 2 отлично от нуля (истина), выполняется тело цикла, вычисляется выражение 3 и осуществляется переход к пункту 2, если выражение 2 равно нулю (ложь), то управление передается на оператор, следующий за оператором for.

✧ *Обратите внимание*, что проверка условия всегда выполняется в начале цикла. Это значит, что тело цикла может ни разу не выполниться, если условие выполнения сразу будет ложным.

Пример использования оператора for:

```
for(i=1;i<10;i++)  //от i равного 1 до 10 с шагом 1 выполнить
  b=i*i;
```

В этом примере вычисляются квадраты чисел от 1 до 9.

Некоторые варианты использования оператора for повышают его гибкость за счет возможности использования нескольких переменных, управляющих циклом.

Пример:

Листинг 22.7. Пример оператора цикла for

```
int main()
 {int top,bot;
```

```
 char string[100],temp;
 for(top=0,bot=100;top<bot;top++,bot--)
  {temp=string[top];
   string[bot]=temp;
  }
 return 0;
}
```

В этом примере, реализующем запись строки символов в обратном порядке, для управления циклом используются две переменные — top и bot. Отметим, что на месте выражение 1 и выражение 3 здесь используются несколько выражений, записанных через запятую и выполняемых последовательно.

Так как согласно синтаксису языка С-51 оператор может быть пустым, тело оператора for также может быть пустым. Такая форма оператора может быть использована для организации поиска.

Пример:

```
for(i=0;t[i]<10;i++);
```

В данном примере переменная цикла i принимает значение номера первого элемента массива t, значение которого больше 10.

При написании программ для микроконтроллеров достаточно часто требуется реализовать бесконечный цикл. В этом случае оператор цикла for принимает следующий вид:

Листинг 22.8. Пример оператора цикла for

```
for(;;)          //Постоянно
  {Knop=P2;        //опрашивать порт P2
   ObrabSobyt(); //и обрабатывать нажатие кнопок
  }
```

Оператор цикла for — это универсальный оператор, поэтому его реализация в машинных командах довольно сложна. Для написания программ часто достаточно более простых операторов цикла. Использование таких операторов обычно приводит к более компактным и быстродействующим программам, однако это не относится к языку программирования С-51. В компиляторе языка программирования С-51 именно этот оператор оптимизирован наилучшим образом.

Оператор цикла с проверкой условия до тела цикла *while*

Оператор цикла `while` называется циклом с предусловием и имеет следующий формат:

```
while (выражение) тело цикла;
```

Оператор `while` содержит условную операцию (такую же, как в операторе `if`) и вызывает исполнение операторов в этом блоке до тех пор, пока условие верно. Проверка условия производится до выполнения тела цикла, поэтому оператор тела цикла может быть не выполнен ни разу. В качестве выражения допускается использовать любое выражение языка С, а в качестве тела — любой оператор, в том числе пустой или составной. Схема выполнения оператора `while` следующая:

1. Вычисляется выражение.
2. Если выражение ложно, то выполнение оператора `while` заканчивается и выполняется следующий по порядку оператор. Если выражение истинно, то выполняется тело цикла (которое может быть составным оператором).
3. Процесс повторяется с пункта 1.

Оператор `while` может быть полезен для ожидания срабатывания какого-либо устройства микроконтроллера, например таймера:

Листинг 22.9. Пример оператора цикла `while`

```
...
while(!TF0);   //Программа ожидает переполнения таймера T0
TF0=0;
TL0=time;      //Настроить таймер T0
TH0=time>>8;   //на очередной интервал времени
...
```

В следующем примере оператор `while` используется для пошагового прохождения по элементам массива `Table` до тех пор, пока очередной элемент не превысит значение скалярной переменной с именем `Level` (от англ. *Level* — уровень):

```
i = 0;
while(Table(i)<=Level)i++;
```

В приведенном примере `Table` — это предварительно объявленный массив. Переменные `Level` и `i` тоже должны быть предварительно объявлены.

Переменной i первоначально присваивается значение 0, затем она используется как индекс для массива Table. Так как i увеличивается при каждом проходе цикла while, то каждый раз, когда выполняется оператор внутри блока while, с переменной Level сравнивается следующий элемент массива Table. Когда найден элемент, превышающий значение переменной Level, то условие в операторе while больше неверно, выполнение блока не повторяется и управление передается следующему за циклом while оператору. С этого момента переменная i является индексом первого элемента массива Table, который превышает значение переменной Level.

Оператор цикла с проверкой условия после тела цикла *do while*

Оператор цикла while называется циклом с постусловием и имеет следующий формат:

```
do тело цикла while (выражение);
```

Оператор do while содержит условную операцию (такую же, как в операторе if) и вызывает исполнение операторов в этом блоке до тех пор, пока условие верно. Проверка условия производится после выполнения тела цикла, поэтому оператор тела цикла будет выполнен хотя бы один раз. В качестве выражения допускается использовать любое выражение языка С, а в качестве тела — любой оператор, в том числе пустой или составной. Схема выполнения оператора do while следующая:

1. Выполняется тело цикла (которое может быть составным оператором).
2. Вычисляется выражение.
3. Если выражение ложно, то выполнение оператора do while заканчивается и выполняется следующий по порядку оператор. Если выражение истинно, то выполнение оператора продолжается с пункта 1.

Чтобы прервать выполнение цикла до того, как условие станет ложным, можно использовать оператор break.

Оператор do while в большинстве случаев соответствует одной машинной команде, поэтому может быть использован для написания эффективных по коду и быстродействию программ. Например:

```
i=10
do тело цикла; while(--i<0);
```

Однако, конкретно в реализации компилятора С-51, намного эффективнее следующая конструкция:

```
for(i=10;i>0;i--) тело цикла;
```

Операторы `while` и `do while` могут быть вложенными.

Пример использования вложенных операторов цикла:

Листинг 22.10. Пример использования операторов цикла `while` и `do while`

```
int i,j,k;
...
i=0;j=0;k=0;
do{i++;j--;
   while (a[k]<i)k++;
  }
while(i<30&&j<-30);
```

Оператор *break*

Оператор `break` обеспечивает прекращение выполнения самого внутреннего из охватывающих его операторов `switch`, `do`, `for`, `while`. После выполнения оператора `break` управление передается оператору, следующему за прерванным оператором цикла или выбора.

Оператор *continue*

Оператор `continue`, как и оператор `break`, используется только внутри операторов цикла, но в отличие от него выполнение цикла не прерывается, а начинается следующая итерация цикла. Формат записи оператора:

```
continue;
```

Листинг 22.11. Пример использования оператора `continue`

```
int main()
{int a,b;
 for(a=1,b=0;a<100;b+=a,a++)
  {if(b%2)continue;
   ... /* обработка четных сумм */
  }
 return 0;
}
```

Когда сумма чисел от 1 до `a` становится нечетной, оператор `continue` передает управление на очередную итерацию цикла `for`, не выполняя операторы обработки четных сумм.

Оператор continue, как и оператор break, прерывает самый внутренний из объемлющих его циклов.

Оператор выбора *switch*

Оператор switch предназначен для организации выбора из множества различных вариантов. Формат оператора следующий:

```
switch ( выражение )
 [[объявление]
      ...
  [ case  константное-выражение1]: [ список-операторов1]
  [ case  константное-выражение2]: [ список-операторов2]
      ...
      ...
  [ default: [ список операторов ]]
 }
```

Выражение, следующее за ключевым словом switch в круглых скобках, может быть любым выражением, допустимым в языке С, значение которого должно быть целым. Отметим, что можно использовать явное приведение к целому типу.

Значение этого выражения является ключевым для выбора из нескольких вариантов. Тело оператора switch состоит из нескольких операторов, помеченных ключевым словом case с последующим константным выражением.

Так как константное выражение вычисляется во время трансляции, оно не может содержать переменные или вызовы функций. Обычно в качестве константного выражения используются целые или символьные константы.

Все константные выражения в операторе switch должны быть уникальны. Кроме операторов, помеченных ключевым словом case, может быть, но обязательно один, фрагмент, помеченный ключевым словом default.

Список операторов может быть пустым либо содержать один или более операторов. Причем в операторе switch не требуется заключать последовательность операторов в фигурные скобки.

Отметим также, что в операторе switch можно использовать свои локальные переменные, объявления которых находятся перед первым ключевым словом case, однако в объявлениях не должна использоваться инициализация.

Схема выполнения оператора switch следующая:

1. Вычисляется выражение в круглых скобках.
2. Вычисленные значения последовательно сравниваются с константными выражениями, следующими за ключевыми словами case.

3. Если одно из константных выражений совпадает со значением выражения, то управление передается на оператор, помеченный соответствующим ключевым словом `case`.
4. Если ни одно из константных выражений не равно выражению, то управление передается на оператор, помеченный ключевым словом `default`, а в случае его отсутствия управление передается на следующий после `switch` оператор.

Отметим интересную особенность использования оператора `switch`: конструкция со словом `default` может быть не последней в теле оператора `switch`. Ключевые слова `case` и `default` в теле оператора `switch` существенны только при начальной проверке, когда определяется начальная точка выполнения тела оператора `switch`. Все операторы, между начальным оператором и концом тела, выполняются вне зависимости от ключевых слов, если только какой-то из операторов не передаст управления из тела оператора `switch`. Таким образом, программист должен сам позаботиться о выходе из оператора `case`, если это необходимо. Чаще всего для этого используется оператор `break`.

Для того чтобы выполнить одни и те же действия для различных значений выражения, можно пометить один и тот же оператор несколькими ключевыми словами `case`.

Если значение выражения равно 0, то выполняется только первый оператор присваивания, и 0 будет присвоен переменной `red` (красный — англ.). Если значение выражения равно 1, то будет выполнен только второй оператор присваивания и переменной `blue` (голубой — англ.) будет присвоен 0. Значения выражения 2 и 3 вызовет присваивание 0 переменным `green` (зеленый — англ.) и `gray` (серый — англ.) соответственно.

Следует отметить, что с точки зрения программиста этот оператор выглядит очень эффектно. Однако при рассмотрении машинного кода получается довольно страшная конструкция с использованием таблиц переходов. Использование нескольких условных операторов `if`, несмотря на то что они не очень красиво выглядят в исходном тексте программы, приводит к короткому и быстродействующему машинному коду программы.

Оператор безусловного перехода *goto*

Оператор `goto` изменяет порядок выполнения программы при помощи передачи управления на оператор, метка которого указана в операторе `goto`. Оператор `goto` записывается в следующем виде:

```
goto имя-метки;
          ...
имя-метки: оператор;
```

Оператор goto передает управление на оператор, помеченный меткой имя-метки. Помеченный оператор должен находиться в той же функции, что и оператор goto, а используемая метка должна быть уникальной, т. е. одно имя-метки не может быть использовано для разных операторов программы. Имя-метки — это идентификатор.

Любой оператор в составном операторе может иметь свою метку. Используя оператор goto, можно передавать управление внутрь составного оператора. Но нужно быть осторожным при входе в составной оператор, содержащий объявления переменных с инициализацией, т. к. объявления располагаются перед выполняемыми операторами и значения объявленных переменных при таком переходе будут не определены.

Использование операторов goto является неизбежным при некоторых ситуациях, однако в большинстве случаев там, где должна быть предусмотрена передача управления, более предпочтительным является использование итеративного оператора while, do while, switch, if или вызова функции. Неограниченное использование операторов goto в программе приводит к тому, что программу становится трудно понимать, модифицировать и сопровождать. Реальная практика использования языка С-51 показывает, что в большинстве случаев программные модули могут не содержать оператор goto без ухудшения их эффективности.

Оператор выражения

Любое выражение, которое заканчивается точкой с запятой, является оператором.

Выполнение оператора выражения заключается в вычислении выражения. Полученное значение выражения никак не используется, поэтому, как правило, такие выражения вызывают побочные эффекты. Заметим, что вызвать подпрограмму-процедуру можно только при помощи оператора выражения.

Примеры записи операторов-выражений:

```
++ i;      /*Этот оператор представляет выражение, которое увеличивает
             значение переменной i на единицу. */
a(x,y);    /*Этот оператор представляет выражение, состоящее из вызова
             подпрограммы-процедуры.*/
```

Оператор возвращения из подпрограммы *return*

Оператор return завершает выполнение функции, в которой он задан, и возвращает управление в вызывающую функцию, в точку, непосредственно следующую за вызовом. Формат оператора:

```
return [выражение] ;
```

Значение выражения, если оно задано, возвращается в вызывающую функцию в качестве значения вызываемой функции. Если выражение опущено, то возвращаемое значение не определено. Выражение может быть заключено в круглые скобки, хотя их наличие не обязательно.

Если в какой-либо функции отсутствует оператор `return`, то передача управления в вызывающую функцию происходит после выполнения последнего оператора вызываемой функции. При этом возвращаемое значение не определено. Если функция не должна иметь возвращаемого значения (подпрограмма-процедура), то ее нужно объявлять с типом `void`.

Таким образом, использование оператора `return` необходимо либо для немедленного выхода из функции, либо для передачи возвращаемого значения.

Пример:

```
int sum(int a,int b)
{return(a+b);
}
```

Функция `sum` имеет два формальных параметра `a` и `b` типа `int` и возвращает значение типа `int`, о чем говорит описатель, стоящий перед именем функции. Возвращаемое оператором `return` значение равно сумме фактических параметров.

Листинг 22.12. Пример использования оператора `continue`

```
void prov (int a, double b)
{float c;
 if(a<3)
     return;
   else if(b>10)
     return;
   else
     {c=a+b;
      if((2*c-b)==11)return;
     }
}
```

В этом примере оператор `return` используется для выхода из функции в случае выполнения одного из проверяемых условий.

Пустой оператор

Пустой оператор состоит только из точки с запятой. У пустого оператора может быть метка. Пустой оператор не содержит ничего, кроме пробелов и

комментариев. При выполнении этого оператора ничего не происходит. Он обычно используется в следующих случаях:

1. В операторах цикла `do while`, `for`, `while` при использовании их в качестве элементов задержки, в операторе `if` в строках, когда место оператора не требуется, но по синтаксису нужен хотя бы один оператор.
2. При необходимости пометить структурный оператор.

Синтаксис языка С-51 требует, чтобы после метки обязательно следовал оператор. Фигурная же скобка оператором не является. Поэтому, если надо передать управление на фигурную скобку, приходится использовать пустой оператор.

Листинг 22.13. Пример использования пустого оператора

```
int main ( )
{ ...
    {if (...) goto a;     /* переход на скобку */
          { ...
          }
 a:;}
 return 0;
}
```

Объявление переменных в языке программирования С-51

Описание переменных в языке программирования С имеет огромнейшее значение, т. к. именно оно в большинстве случаев определяет объем программы. Обычно большой объем загрузочного модуля программы вызван неправильным объявлением переменных в ее исходном тексте. Обращение к внутренним регистрам микроконтроллеров и внешним ресурсам разрабатываемого устройства тоже производится при помощи заранее объявленных переменных.

В языке программирования С-51 любая переменная должна быть объявлена до первого использования этой переменной. Как уже говорилось ранее, этот язык программирования предназначен для написания программ для микроконтроллеров семейства MCS-51, поэтому в составе языка должна отображаться внутренняя структура этого семейства микроконтроллеров. Эти особенности отражены во введении новых типов данных. В остальном язык программирования С-51 не отличается от стандарта ANSI.

Объявление переменной в языке программирования С-51 представляется в следующем виде:

```
[спецификатор класса памяти] спецификатор типа [спецификатор типа памяти] описатель [=инициатор] [,описатель [= инициатор] ]...
```

Описа́тель — идентификатор простой переменной либо более сложная конструкция с квадратными скобками, круглыми скобками или звездочкой (набором звездочек).

Спецификатор типа — одно или несколько ключевых слов, определяющих тип объявляемой переменной. В языке С-51 имеется стандартный набор типов данных, используя который можно сконструировать новые (уникальные) типы данных.

Инициатор задает начальное значение или список начальных значений, которые (которое) присваивается переменной при объявлении.

Спецификатор класса памяти определяется одним из ключевых слов языка С: `auto`, `bit`, `extern`, `register`, `sbit`, `sfr`, `sfr16`, `static`, и указывает, каким образом будет распределяться память под объявляемую переменную, с одной стороны, а с другой — область видимости этой переменной, т. е. из каких частей программы можно к ней обратиться.

Спецификатор типа памяти определяется одним из шести ключевых слов языка С-51: `code`, `data`, `idata`, `bdata`, `xdata`, `pdata`, и указывает, в какой области памяти микроконтроллера будет размещена переменная.

Категории типов данных

Ключевые слова для определения основных типов данных:

Целые типы данных:	Типы данных с плавающей запятой:
bit	float
sbit	
char	
int	
short	
long	
signed	
unsigned	
sfr	
sfr16	

Переменная любого типа может быть объявлена как неизменяемая. Это достигается добавлением ключевого слова `const` к спецификатору типа. Объекты с типом `const` представляют собой данные, используемые только для чтения,

т. е. этой переменной не может быть присвоено новое значение. Отметим, что если после слова const отсутствует спецификатор типа, то подразумевается спецификатор типа int. Если ключевое слово const стоит перед объявлением составных типов (массив, структура, смесь, перечисление), то это приводит к тому, что каждый элемент также должен являться немодифицируемым, т. е. значение ему может быть присвоено только один раз.

Примеры использования ключевого слова const:

```
const float A=2.128E-2;
const B=286;                    //подразумевается const int B=286
```

Отметим, что переменные со спецификатором класса памяти размещаются во внутреннем ОЗУ. Неизменяемость контролируется только на этапе трансляции. Для размещения переменной в ПЗУ лучше воспользоваться спецификатором типа памяти code.

Целые типы данных

Для определения данных целого типа используются различные ключевые слова, которые определяют диапазон значений и размер области памяти, выделяемой под переменные (табл. 22.7).

Таблица 22.7. Ключевые слова для определения данных целого типа

Тип	Размер памяти в битах	Размер памяти в байтах	Диапазон значений
bit	1		от 0 до 1
char	8	1	от –128 до 127
unsigned shar	8	1	от 0 до 255
int, short	16	2	от –32768 до 32767
long	32	4	от –2 147 483 648 до 2 147 483 647
unsigned int, unsigned short	16	2	от 0 до 65535
unsigned long	32	4	от 0 до 4 294 967 295
sbit	1		0 или 1
sfr	8	1	от 0 до 255
sfr16	16	2	от 0 до 65 535

Отметим, что ключевые слова signed и unsigned необязательны. Они указывают, как интерпретируется нулевой бит объявляемой переменной, т. е. если

указано ключевое слово unsigned, то нулевой бит интерпретируется как часть числа, в противном случае нулевой бит интерпретируется как знаковый.

В случае отсутствия ключевого слова unsigned целая переменная считается знаковой. В том случае, если спецификатор типа состоит из ключевого типа signed или unsigned и далее следует идентификатор переменной, то она будет рассматриваться как переменная типа int. Например:

```
unsigned int n;  //Беззнаковое шестнадцатиразрядное число n
unsigned int b;
int c;           //подразумевается  signed   int c
unsigned d;      //подразумевается  unsigned int d
signed f;        //подразумевается  signed   int f
```

Отметим, что модификатор типа char используется для представления одиночного символа или для объявления строковых литералов. Значением объекта типа char является код (размером 1 байт), соответствующий представляемому символу.

Отметим также, что восьмеричные и шестнадцатеричные константы тоже могут иметь модификатор unsigned. Это достигается указанием суффикса u или U после константы, константа без этого префикса считается знаковой.

Например:

```
0xA8C   //int  signed
017861  //long signed
0xF7u   //int  unsigned
```

Числа с плавающей запятой

Для переменных, представляющих число с плавающей запятой, используется модификатор типа float. Модификатор double тоже допустим в языке программирования С-51, но он не приводит к увеличению точности результата.

Величина с модификатором типа float занимает 4 байта. Из них 1 бит отводится для знака, 8 бит для смещенного порядка и 23 бита для мантиссы. Отметим, что старший бит мантиссы всегда равен 1, поэтому он не запоминается в памяти микроконтроллера, в связи с этим диапазон значений переменной с плавающей точкой равен от ±1.175494E-38 до ±3.402823E+38.

Пример объявления переменной с плавающей запятой:

```
float f, a, b;
```

Переменные перечислимого типа

Переменная, которая может принимать значение из некоторого списка значений, называется *переменной перечислимого типа* или *перечислением*.

Использование такого вида переменной эквивалентно применению целой знаковой переменной char или int. Это означает, что для переменной перечислимого вида будет выделен один или два байта в зависимости от максимального значения используемых этой переменной констант. В отличие от переменных целого типа, переменные перечислимого типа позволяют вместо безликих чисел использовать имена констант, которые более понятны и легче запоминаются человеком.

Например, вместо использования чисел 1, 2, 3, 4, 5, 6, 7 можно применять названия дней недели: Poned, Vtorn, Sreda, Chetv, Pjatn, Subb, Voskr. При этом каждой константе будет соответствовать свое конкретное число. Однако использование имен констант приведет к более понятной программе. Более того, транслятор сам позволяет отслеживать правильность употребления констант и при попытке использования константы, не входящей в объявленный заранее список, выдает сообщение об ошибке.

Переменные enum типа могут использоваться в индексных выражениях и как операнды в арифметических операциях и в операциях отношения. Например:

```
if(rab_ned == SUB) dejstvie = rabota [rab_ned];
```

При объявлении перечисления определяется тип переменной перечисления и список именованных констант, называемый списком перечисления. Значением каждого имени этого списка является целое число. Объявление перечислимой переменной начинается с ключевого слова enum и может быть представлено в двух форматах:

```
enum [имя типа перечисления] {список констант} имя1 [,имя2 ...];
enum имя перечисления описатель [,описатель..];
```

В первом формате имена и значения констант задаются в списке констант. Необязательное имя типа объявляемой переменной — это идентификатор, который представляет собой тип переменной, соответствующий списку констант. За списком констант записывается имя одной или нескольких переменных.

Список констант содержит одну или несколько конструкций вида:

```
идентификатор [= константное выражение]
```

Каждый идентификатор — это имя константы. Все идентификаторы в списке enum должны быть уникальными. В случае если константе явным образом не задается число, то первому идентификатору присваивается значение 0, следующему идентификатору — значение 1 и т. д.

Пример объявления переменной rab_ned и типа переменных week, совместимых с этой переменной, выглядит следующим образом:

Листинг 22.14. Пример объявления переменной перечислимого типа

```
enum week {SUB = 0, /* константе SUB присвоено значение 0 */
           VOS = 0, /* константе VOS присвоено значение 0 */
           POND,    /* константе POND присвоено значение 1 */
           VTOR,    /* константе VTOR присвоено значение 2 */
           SRED,    /* константе SRED присвоено значение 3 */
           HETV,    /* константе HETV присвоено значение 4 */
           PJAT     /* константе PJAT присвоено значение 5 */
          } rab_ned;
```

Идентификатор, связанный с константным выражением, принимает значение, задаваемое этим константным выражением. Результат вычисления константного выражения должен иметь тип `int` и может быть как положительным, так и отрицательным. Следующему идентификатору в списке, если этот идентификатор не имеет своего константного выражения, присваивается значение, равное константному выражению предыдущего идентификатора плюс 1. Использование констант должно подчиняться следующим правилам:

1. Объявляемая переменная может содержать повторяющиеся значения констант.
2. Идентификаторы в списке констант должны быть отличны от всех других идентификаторов в той же области видимости, включая имена обычных переменных и идентификаторы из других списков констант.
3. Имена типов перечислений должны быть отличны от других имен типов перечислений, структур и смесей в этой же области видимости.
4. Значение может следовать за последним элементом списка перечисления.

Во втором формате для объявления переменной перечислимого типа используется уже готовый тип переменной, объявленный ранее. Например:

```
enum week rab1;
```

К переменной перечислимого типа можно обращаться при помощи указателей. При этом необходимо заранее определить тип переменной, на которую будет ссылаться указатель. Это может быть сделано, как описывалось выше или при помощи оператора `typedef`.

Листинг 22.15. Пример объявления типа переменной перечислимого типа

```
Typedef enum {SUB = 0, /* константе SUB присвоено значение 0 */
              VOS = 0, /* константе VOS присвоено значение 0 */
              POND,    /* константе POND присвоено значение 1 */
              VTOR,    /* константе VTOR присвоено значение 2 */
```

```
    SRED,     /* константе SRED присвоено значение 3 */
    HETV,     /* константе HETV присвоено значение 4 */
    PJAT      /* константе PJAT присвоено значение 5 */
  } week;
```

Этот оператор не объявляет переменную, а только определяет тип переменной, отличающийся от стандартного. В дальнейшем этот тип может быть использован для объявления переменных и указателей на переменные.

Объявление массивов в языке программирования С-51

При обработке данных достаточно часто приходится работать с рядом переменных одинакового типа (и описывающих одинаковые объекты). В этом случае эти переменные имеет смысл объединить одним идентификатором. Это позволяют сделать массивы.

Массивы — это группа элементов одинакового типа (`float`, `char`, `int` и т. п.). Из объявления массива компилятор должен получить информацию о типе элементов массива и их количестве. Объявление массива имеет два формата:

```
спецификатор-типа описатель [константное-выражение];
спецификатор-типа описатель [ ];
Описатель — это идентификатор массива.
```

Спецификатор-типа задает тип элементов объявляемого массива. Элементами массива не могут быть функции и элементы типа `void`.

Константное-выражение в квадратных скобках задает количество элементов массива. Константное-выражение при объявлении массива может быть опущено в следующих случаях:

1. При объявлении массив инициализируется.
2. Массив объявлен как формальный параметр функции.
3. Массив объявлен как ссылка на массив, явно определенный в другом файле.

В языке С-51 определены только одномерные массивы, но поскольку элементом массива может быть массив, можно определить и многомерные массивы. Они формализуются списком константных-выражений, следующих за идентификатором массива, причем каждое константное-выражение заключается в свои квадратные скобки.

Каждое константное-выражение в квадратных скобках определяет число элементов по данному измерению массива, так что объявление двухмерного

массива содержит два константных-выражения, трехмерного — три и т. д. Отметим, что в языке С-51 первый элемент массива имеет индекс, равный 0.

Листинг 22.16. Пример объявления массивов

```
int a[2][3]; /* представлено в виде матрицы
                a[0][0]  a[0][1] a[0][2]
                a[1][0]  a[1][1] a[1][2] */
float b[10]; /* вектор из 10  элементов, имеющих тип double */
int w[3][3] = { { 2, 3, 4 },
                { 3, 4, 8 },
                { 1, 0, 9 } };
```

В последнем примере объявлен массив `w[3][3]`. Списки, выделенные в фигурные скобки, соответствуют строкам массива, в случае отсутствия скобок инициализация будет выполнена неправильно.

В языке С-51 можно использовать сечения массива, как и в других языках высокого уровня (PLM и т. п.), однако на использование сечений накладывается ряд ограничений. Сечения формируются вследствие опускания одной или нескольких пар квадратных скобок. Пары квадратных скобок можно отбрасывать только справа налево и строго последовательно. Сечения массивов используются при организации вычислительного процесса в функциях языка С-51, разрабатываемых пользователем.

Примеры использования сечений массивов:

Пусть объявлен массив `s`:

```
int s[2][3];
```

Если при обращении к некоторой функции написать `s[0]`, то в эту функцию будет передаваться нулевая строка массива `s`.

```
int b[2][3][4];
```

При обращении к массиву `b` можно написать, например, `b[1][2]` и будет передаваться вектор из четырех элементов, а обращение `b[1]` даст двухмерный массив размером 3 на 4. Нельзя написать `b[2][4]`, подразумевая, что передаваться будет вектор, потому что это не соответствует ограничению, наложенному на использование сечений массива.

Для работы с символьными строками в языке программирования С используются массивы символов, например:

```
char str[] = "объявление массива символов";
```

Следует учитывать, что в символьной строке находится на один элемент больше, т. к. последним элементом строки должен быть '\0'. В этом примере использовано неявное задание длины массива символов. Это стало возможным, т. к. массиву сразу присваивается конкретное значение.

При написании программ для микроконтроллеров семейства MCS-51 такое задание массива может привести к неоправданному расходу внутренней памяти данных, а эта память в микроконтроллере составляет всего 120 байт. Поэтому лучше воспользоваться размещением строки в памяти программ, как это показано в следующем примере:

```
char code str[] = "объявление массива символов";
```

Структуры

Работа с массивами облегчает понимание и написание программы, когда для обозначения похожих элементов используется один идентификатор. Однако в ряде случаев приходится обрабатывать разнородные элементы, описывающие один объект. В этом случае вместо массива используется структура.

Структура — это составной объект, в который входят элементы любых типов, за исключением функций. В отличие от массива, который является однородным объектом, структура может быть неоднородной. Тип структуры определяется записью вида:

```
struct { список описаний}
```

В структуре обязательно должен быть указан хотя бы один компонент. Компоненты структуры называются полями структуры. Объявление полей производится в следующем виде:

```
тип-данных описатель;
```

где `тип-данных` указывает тип структуры для объектов, определяемых в описателях. В простейшей форме описатели представляют собой идентификаторы переменных или массивов.

Листинг 22.17. Пример объявления структур

```
 struct//Описание типа структуры-----------------------
 {char tzvet,//Цвет точки
  int  x,     //Координата X
       y;     //Координата Y
 }//-----------------------------------------------------
tochka1, tochka2, //Переменные, обозначающие точки дисплея
simv[7][9];       //Переменная, содержащая рисунок символа
```

```
struct
 {int year;    //Поле структуры, в котором хранится год
  char  moth,  //Поле структуры, в котором хранится месяц
        day;   //Поле структуры, в котором хранится день
 }//-------------------------------------------------
 date1, date2;//Переменные, обозначающие две различных даты
```

Переменные `tochka1`, `tochka2` объявляются как структуры, каждая из которых состоит из трех полей: `tzvet`, `x` и `y`. Переменная `simv` объявляется как двухмерный массив, состоящий из 63 переменных, описывающих точку дисплея. Во втором объявлении каждая из двух переменных `date1`, `date2` состоит из трех компонентов: `year`, `moth`, `day`.

Существует и другой способ связывания имени переменной с типом структуры, он основан на использовании отдельного объявления типа структуры. Тип структуры описывается следующим образом:

```
struct тип-структуры { список описаний; };
```

где `тип-структуры` является идентификатором типа объявляемой структуры и может быть использован для последующего объявления структур данного вида (т. е. содержащих точно такие же поля) в форме:

```
struct тип-структуры список-идентификаторов;
```

В приведенном ниже примере идентификатор `student` описывается как тип структуры:

Листинг 22.18. Пример объявления структуры

```
struct student
{char name[25];//Имя и фамилия студента
 int  id,      //Номер в журнале
      age;     //Возраст
 char usp;     //Успеваемость
 };
```

Пример:

```
struct student st1[23];//объявление массива переменных типа студент
```

Доступ к полям структуры осуществляется с помощью указания имени структуры и следующего через точку имени поля, например:

Листинг 22.19. Пример обращения к полям структуры

```
st[1].name="Иванов";
st[1].id=2;
st[1].age=23;
```

Поля битов

Элементом структуры может быть битовое поле, обеспечивающее доступ к отдельным битам памяти. Вне структур битовые поля объявлять нельзя. Не следует также организовывать массивы битовых полей и применять к полям операцию определения адреса. В общем случае тип структуры с битовым полем задается в следующем виде:

```
struct { unsigned идентификатор 1 :  длина-поля  1;
         unsigned идентификатор 2 :  длина-поля  2;    }
```

где длина-поля задается целым выражением или константой. Эта константа определяет число битов, отведенное соответствующему полю. Поле нулевой длины обозначает выравнивание на границу следующего слова.

Листинг 22.20. Пример объявления полей битов

```
struct
{unsigned R:  1;    //Флаг приема байта
 unsigned T:  1;    //Флаг передачи байта
 unsigned Cmd:5;    //Поле команды
 unsigned St: 1;    //Поле статуса
} Cntr;
```

Структуры битовых полей могут содержать и знаковые компоненты. Такие компоненты автоматически размещаются на соответствующих границах слов, при этом некоторые биты слов могут оставаться неиспользованными.

Ссылки на поле битов выполняются точно так же, как и компоненты общих структур. Само же битовое поле рассматривается как целое число, максимальное значение которого определяется длиной поля. Например:

```
Cntr.Cmd=30;
```

Следует отметить, что, несмотря на то что язык программирования С-51, как наследник стандартного языка программирования С, позволяет использовать поля битов, лучше для битовых переменных использовать специализированный тип переменных `bit`, размещаемых в пространстве битовой адресации.

Объединения (смеси)

Главной особенностью объединения является то, что для каждого из объявленных элементов этого объединения выделяется одна и та же область памяти, т. е. они перекрываются. Хотя доступ к этой области памяти возможен

с использованием любого из элементов, элемент для этой цели должен выбираться так, чтобы полученный результат не был бессмысленным.

Объединение применяется для следующих целей:

1. Использования одной и той же области памяти для размещения переменных различного типа.
2. Интерпретации представления переменной одного типа, как несколько переменных другого типа.

Объединение по описанию подобно структуре. Тип объединения может задаваться в следующем виде:

```
union {   описание элемента 1;
          ...
          описание элемента n; };
```

Доступ к элементам объединения осуществляется тем же способом, что и к структурам.

Память, которая соответствует переменной типа объединения, определяется величиной, необходимой для размещения наиболее длинного элемента объединения. Когда используется элемент меньшей длины, то переменная типа объединения может содержать неиспользуемую память. Все элементы объединения хранятся в одной и той же области памяти, начиная с одного адреса.

Второй вариант можно проиллюстрировать следующим образом. Например, требуется передать число плавающего типа. Однако последовательный порт может передавать или принимать только однобайтовые числа. В этом случае можно воспользоваться объединением:

Листинг 22.21. Пример объединения

```
union {float Koeff; //Интерпретация объединения как переменной
                    //плавающего типа
       char byte[4];//Интерпретация объединения как массива
       }buffer;       //Объявление переменной buffer
```

Объединение `buffer` позволяет последовательному порту получить отдельный доступ ко всем байтам числа `buffer.Koeff`, начиная от младшего байта `buffer.byte[0]` и заканчивая старшим байтом `buffer.byte[3]`. В программе затем можно пользоваться загруженным числом, как числом с плавающей запятой.

Объявление указателей в языке программирования С-51

Указатель — это переменная, которая может содержать адрес другой переменной. Указатель может быть использован для работы с переменной, адрес которой он содержит. Использование указателей позволяет осуществить более эффективную обработку массивов, структур, а также реализовывать подпрограммы, которые будут работать над различными областями памяти микроконтроллера. Для этого в подпрограмму нужно только передать начальный адрес обрабатываемой области памяти.

Для инициализации указателя (записи начального адреса переменной) можно использовать идентификатор переменной, при этом в качестве идентификатора может выступать имя переменной, массива, структуры, литеральной строки.

При объявлении переменной-указателя необходимо определить тип объекта данных, адрес которых будет содержать переменная, и имя указателя с предшествующей звездочкой (или группой звездочек). Формат объявления указателя:

```
спецификатор-типа [ модификатор ] *описатель.
```

Спецификатор-типа задает тип объекта и может быть любого основного типа, структуры или смеси (об этих типах будет сказано ниже). Задавая вместо спецификатора-типа ключевое слово `void`, можно отсрочить определение типа, на который ссылается указатель. Переменная, объявляемая как указатель на тип `void`, может быть использована для ссылки на объект любого типа. Однако для того, чтобы можно было выполнить арифметические и логические операции над указателями или над объектами, на которые они указывают, необходимо при выполнении каждой операции явно определить тип объектов. Такие определения типов могут быть выполнены с помощью операции приведения типов.

Листинг 22.22. Пример объявления указателей на различные типы переменных

```
unsigned int * ptr; /* переменная ptr  представляет собой указатель на
                       целую беззнаковую) переменную*/
float * x;          /* переменная  x  указывает  на переменную с
                       плавающей  точкой*/
char *buffer ;      /* объявляется  указатель с именем buffer, который
                       указывает на символьную переменную*/
```

Теперь, для того чтобы начать работать с этими указателями, достаточно их инициализировать.

Листинг 22.23. Пример инициализации указателей

```
ptr=&A;              //Присвоить адрес переменной A
*ptr=2+2;            //Работаем с переменной A
a=&B;
*ptr=3*4;            //A теперь работаем с переменной B
```

В качестве модификаторов при объявлении указателя могут выступать ключевые слова `const`, `data`, `idata`, `xdata`, `code`. Ключевое слово `const` указывает, что указатель не может быть изменен в программе. Размер переменной, объявленной как указатель, зависит от модификатора и используемого вида памяти, для которой будет компилироваться программа. Указатели на различные типы данных не обязательно должны иметь одинаковую длину.

Для изменения размера указателя можно использовать ключевые слова `data`, `idata`, `xdata`, `code`.

Задавая вместо спецификатора типа ключевое слово `void`, можно отсрочить определение типа, на который ссылается указатель. Переменная, объявляемая как указатель на тип `void`, может быть использована для ссылки на объект любого типа. Однако для того, чтобы можно было выполнить арифметические и логические операции над указателями или над объектами, на которые они указывают, необходимо при выполнении каждой операции явно определить тип объектов. Такие определения типов могут быть выполнены с помощью операции приведения типов.

Листинг 22.24. Пример определения типов указателей с помощью операции приведения типов

```
float nomer;
void *addres;
addres = &nomer;
(float *)addres ++;
/* Переменная addres объявлена как указатель на объект любого типа.
   Поэтому ей можно присвоить адрес любого объекта
   (& — операция вычисления адреса).
   Однако, как было отмечено выше, ни одна арифметическая операция не
   может быть выполнена над указателем, пока не будет явно определен
   тип данных,  на которые он указывает. Это можно сделать, используя
   операцию приведения типа (float *) для преобразования типа указателя
   addres к типу float. Затем оператор ++ отдает приказ перейти
   к следующему адресу.*/
```

Вследствие уникальности архитектуры микроконтроллера 8051 и его последующих модификаций, компилятор C-51 поддерживает 2 вида указателей: память-зависимые и нетипизированные.

Нетипизированные указатели

Нетипизированные указатели объявляются точно так же, как указатели в стандартном языке программирования С. Для того чтобы не зависеть от типа памяти, в которой может быть размещена переменная, для нетипизированных указателей выделяется 3 байта. В первом байте указывается вид памяти переменной, во втором байте — старший байт адреса, в третьем — младший байт адреса переменной. Нетипизированные указатели могут быть использованы для обращения к любым переменным независимо от типа памяти микроконтроллера. Именно поэтому многие библиотечные функции языка программирования С-51 используют указатели этого типа, при этом им совершенно неважно, в какой именно области памяти размещаются переменные. Приведем листинг, в котором отображаются особенности трансляции нетипизированных указателей:

Листинг 22.25. Особенности трансляции нетипизированных указателей

```
stmt level source
   1       char *c_ptr; /* char ptr */
   2       int  *i_ptr; /* int  ptr */
   3       long *l_ptr; /* long ptr */
   4
   5       void main (void)
   6       {
   7     1 char data dj; /*переменные во внутренней памяти данных data */
   8     1 int data dk;
   9     1 long data dl;
  10     1
  11     1 char xdata xj; /*переменные во внешней памяти данных xdata */
  12     1 int xdata xk;
  13     1 long xdata xl;
  14     1
  15     1 char code cj = 9; /*переменные в памяти программ code */
  16     1 int code ck = 357;
  17     1 long code cl = 123456789;
  18     1
  19     1 /*настроим указатели на внутреннюю память данных data */
  20     1 c_ptr = &dj;
  21     1 i_ptr = &dk;
  22     1 l_ptr = &dl;
  23     1 /*настроим указатели на внешнюю память данных xdata */
  24     1 c_ptr = &xj;
  25     1 i_ptr = &xk;
  26     1 l_ptr = &xl;
```

```
  27     1 /*настроим указатели на память программ code */
  28     1 c_ptr = &cj;
  29     1 i_ptr = &ck;
  30     1 l_ptr = &cl;
  31     1 }

ASSEMBLY LISTING OF GENERATED OBJECT CODE
      ; FUNCTION main (BEGIN)
                           ; SOURCE LINE # 5
                           ; SOURCE LINE # 6
                           ; SOURCE LINE # 20
0000 750000 R        MOV   c_ptr,#00H
0003 750000 R        MOV   c_ptr+01H,#HIGH dj
0006 750000 R        MOV   c_ptr+02H,#LOW dj
                           ; SOURCE LINE # 21
0009 750000 R        MOV   i_ptr,#00H
000C 750000 R        MOV   i_ptr+01H,#HIGH dk
000F 750000 R        MOV   i_ptr+02H,#LOW dk
                           ; SOURCE LINE # 22
0012 750000 R        MOV   l_ptr,#00H
0015 750000 R        MOV   l_ptr+01H,#HIGH dl
0018 750000 R        MOV   l_ptr+02H,#LOW dl
                           ; SOURCE LINE # 24
001B 750001 R        MOV   c_ptr,#01H
001E 750000 R        MOV   c_ptr+01H,#HIGH xj
0021 750000 R        MOV   c_ptr+02H,#LOW xj
                           ; SOURCE LINE # 25
0024 750001 R        MOV   i_ptr,#01H
0027 750000 R        MOV   i_ptr+01H,#HIGH xk
002A 750000 R        MOV   i_ptr+02H,#LOW xk
                           ; SOURCE LINE # 26
002D 750001 R        MOV   l_ptr,#01H
0030 750000 R        MOV   l_ptr+01H,#HIGH xl
0033 750000 R        MOV   l_ptr+02H,#LOW xl
                           ; SOURCE LINE # 28
0036 7500FF R        MOV   c_ptr,#0FFH
0039 750000 R        MOV   c_ptr+01H,#HIGH cj
003C 750000 R        MOV   c_ptr+02H,#LOW cj
                           ; SOURCE LINE # 29
003F 7500FF R        MOV   i_ptr,#0FFH
0042 750000 R        MOV   i_ptr+01H,#HIGH ck
0045 750000 R        MOV   i_ptr+02H,#LOW ck
                           ; SOURCE LINE # 30
```

```
0048 7500FF R   MOV   l_ptr,#0FFH
004B 750000 R   MOV   l_ptr+01H,#HIGH cl
004E 750000 R   MOV   l_ptr+02H,#LOW cl
                      ; SOURCE LINE # 31
0051 22         RET
     ; FUNCTION main (END)
```

Память-зависимые указатели

В объявления память-зависимых указателей всегда включается модификатор памяти. Обращение всегда происходит к указанной области памяти, например:

```
char data *str;     //указатель на строку во внутренней памяти данных data
int xdata *numtab; //указатель на целую во внешней памяти данных xdata
long code *powtab; //указатель на длинную целую в памяти программ code
```

Поскольку модель памяти определяется во время компиляции, типизированным указателям не нужен байт, в котором указывается тип памяти микроконтроллера. Поэтому программа с использованием типизированных указателей будет короче и будет выполняться быстрее по сравнению с программой, использующей нетипизированные указатели. Типизированные указатели могут иметь размер в 1 байт (указатели на память `idata`, `data`, `bdata`, и `pdata`) или в 2 байта (указатели на память `code` и `xdata`).

Объявление новых типов переменных

В языке программирования C-51 имеется возможность заранее объявить тип переменной, а затем воспользоваться им при объявлении переменных. Использование заранее объявленного типа позволяет при объявлении переменной сократить его длину, избежать ошибок при объявлении переменных в разных местах программы и добиться полной идентичности объявляемых переменных.

Объявить новый тип переменной можно двумя способами. Первый способ — указать имя типа при объявлении структуры, объединения или перечисления, а затем использовать это имя в объявлении переменных и функций. Второй — использовать для объявления типа ключевое слово `typedef`.

При объявлении типа с ключевым словом `typedef` идентификатор, стоящий на месте описываемого объекта, является именем объявляемого типа данных, и далее этот тип может быть использован для объявления переменных.

Отметим, что любой тип может быть объявлен с использованием ключевого слова `typedef`, включая типы указателей, функций или массивов. Имя с клю-

чевым словом typedef для типов указателя, структуры, объединения может быть объявлено прежде, чем эти типы будут определены, но в пределах видимости объявителя.

Листинг 22.26. Примеры объявления и использования новых типов

```
typedef
 float (* MATH)( );       /* MATH – новое имя типа, представляющее
                             указатель на  функцию, возвращающую значения
                             типа float*/
typedef
 char FIO[40]             // FIO – массив из сорока символов

  MATH cos;               /* cos  указатель на функцию, возвращающую
                             значения типа  double*/

 // Можно провести эквивалентное объявление

  float (* cos)( );

 FIO person;              //Переменная person – массив из сорока символов

// Это эквивалентно объявлению

   char person[40];
```

При объявлении переменных и типов здесь были использованы имена типов MATH и FIO. Помимо объявления переменных, имена типов могут еще использоваться в трех случаях: в списке формальных параметров при объявлении функций, в операциях приведения типов и в операции sizeof.

Инициализация данных

В языке программирования C-51, как и в других версиях языка C, при объявлении переменной ей можно присвоить начальное значение, присоединяя инициатор к описателю. При этом во время запуска вашей программы в ячейки памяти, соответствующие этим переменным, будут записаны начальные значения. Только после этого выполнение программы будет передано подпрограмме main();.

Инициатор переменной начинается со знака "=" и может быть записан в следующих форматах:

```
Формат 1: = инициатор;
Формат 2: = { список – инициаторов };
```

Формат 1 используется при инициализации переменных основных типов и указателей, а формат 2 — при инициализации составных объектов.

Примеры присваивания первоначальных значений простым переменным:

```
char tol = 'N';                    //Переменная tol инициализируется
символом 'N'.
const long megabyte = (1024*1024);
```

Немодифицируемой переменной `megabyte` присваивается значение константного выражения, после чего эта переменная не может быть изменена. Отмечу, что для микроконтроллеров семейства MCS-51 внутренняя память является дефицитным ресурсом, поэтому использовать ее для хранения констант нерационально. Лучше объявить переменную со спецификатором типа памяти `code`.

```
static int b[2][2] = {1,2,3,4};
```

Инициализируется двухмерный массив `b` целых величин, элементам массива присваиваются значения из списка. Эта же инициализация может быть выполнена следующим образом:

```
static int b[2][2] = { { 1,2 }, { 3,4 } };
```

При инициализации массива можно опустить одну или несколько размерностей:

```
static int
 b[3] = { { 1,2 }, { 3,4 } };
```

Если при инициализации указано меньше значений для строк, то оставшиеся элементы инициализируются 0, т. е. при описании:

```
static int b[2][2] = { { 1,2 }, { 3 } };
```

элементы первой строки получат значения 1 и 2, а второй 3 и 0.

При инициализации составных объектов нужно внимательно следить за использованием скобок и списков инициализаторов.

Листинг 22.27. Примеры инициализации составных объектов

```
struct complex
 {float real;
  float imag;
 }comp[2][3]={{{1,1},{2,3},{ 4, 5}},
              {{6,7},{8,9},{10,11}}
              };
```

В данном примере инициализируется массив структур `comp` из двух строк и трех столбцов, где каждая структура состоит из двух элементов `real` и `imag`.

```
struct
 complex comp2 [2][3] = { {1,1},{2,3},{4,5},{6,7},{8,9},{10,11} };
```

В этом примере компилятор интерпретирует рассматриваемые фигурные скобки следующим образом:

1. Первая левая фигурная скобка — начало составного инициатора для массива `comp2`.
2. Вторая левая фигурная скобка — начало инициализации первой строки массива `comp2[0]`. Значения 1,1 присваиваются двум элементам первой структуры.
3. Первая правая скобка (после 1) указывает компилятору, что список инициаторов для строки массива окончен, и элементы оставшихся структур в строке `comp[0]` автоматически инициализируются нулем.
4. Аналогично список `{2,3}` инициализирует первую структуру в строке `comp[1]`, а оставшиеся структуры массива обращаются в нули.
5. На следующий список инициализаторов `{4,5}` компилятор будет сообщать о возможной ошибке, т. к. строка 3 в массиве `comp2` отсутствует.

При инициализации объединения задается значение первого элемента объединения в соответствии с его типом.

Листинг 22.28. Пример инициализации объединения

```
union tab
 {unsigned char name[10];
  int tab1;
  }pers={'A','H','T','O','H'};
```

Инициализируется переменная `pers.name`, и т. к. это массив, для его инициализации требуется список значений в фигурных скобках. Первые пять элементов массива инициализируются значениями из списка, остальные нулями.

Инициализацию массива символов можно выполнить при помощи литеральной строки.

```
char stroka[ ] = "привет";
```

Инициализируется массив символов из 7 элементов, последним элементом (седьмым) будет символ '\0', которым завершаются все литеральные строки.

В случае если задается размер массива, а литеральная строка длиннее, чем размер массива, то лишние символы отбрасываются. Следующее объявление инициализирует переменную `stroka` как массив, состоящий из семи элементов.

```
char stroka[5]="привет";
```

В переменную stroka попадают первые пять элементов литерала, а символы 'т' и '\0' отбрасываются. Если строка короче размерности массива, то оставшиеся элементы массива заполняются нулями. Отметим, что инициализация переменной типа tab может иметь следующий вид:

```
union tab pers1="Антон";
```

И, таким образом, в символьный массив попадут символы:

```
'А','Н','Т','О','Н','\0',
```

А в остальные элементы будут записаны нули.

Использование функций в языке программирования С-51

В отличие от других языков программирования высокого уровня, в языке С нет деления на основную программу, процедуры и функции. В этом языке вся программа строится только из функций. Мощность языка С во многом определяется легкостью и гибкостью объявления и реализации функций.

Функция — это совокупность объявлений и операторов, обычно предназначенная для решения определенной задачи. Термин "функция" в языке программирования С эквивалентен понятию подпрограммы. Действия, выполняемые основной программой в других языках программирования, такие как очистка внутреннего ОЗУ и присваивание начального значения переменным, выполняются автоматически при включении питания. После завершения этих действий вызывается подпрограмма с именем main. Каждая функция должна иметь имя, которое используется для ее объявления, определения и вызова.

В любой программе, написанной на языке программирования С-51, должна быть функция с именем main (главная функция), именно с этой функции, в каком бы месте программы она не находилась, начинается выполнение программы. ✧ *Обратите внимание*, что на языке программирования С-51 пишутся программы для микроконтроллеров. Поэтому эти программы не должны завершаться до тех пор, пока включено питание микроконтроллера. Это значит, что в функции main обязательно должен быть бесконечный цикл. В противном случае при выходе из этой функции управление будет передано случайному адресу памяти программ. Это может привести к непредсказуемым результатам, вплоть до выхода микроконтроллерной системы из строя.

При вызове функции ей при помощи аргументов (формальных параметров) могут быть переданы некоторые значения (фактические параметры), используемые во время выполнения функции. Функция может возвращать некоторое, но только одно, значение. Это возвращаемое значение и есть результат

выполнения функции, который при выполнении программы подставляется в выражение, из которого производился вызов функции.

Допускается также использовать функции, не имеющие аргументов, и функции, не возвращающие никаких значений. Действие таких функций может состоять, например, в изменении значений некоторых переменных, выводе на экран жидкокристаллического индикатора текстовых надписей и т. п. Иными словами можно сказать, что такая функция работает подобно подпрограмме-процедуре.

С использованием функций в языке С-51 связаны три понятия — определение функции (описание действий, выполняемых подпрограммой-функцией), объявление функции (задание формы обращения к функции) и вызов функции.

Определение функций

Определение функции состоит из заголовка и тела. Определение функции записывается в следующем виде:

```
[спецификатор класса памяти] [спецификатор типа] имя функции ([список
формальных параметров]) //Заголовок функции
{
//тело функции
}
```

Заголовок функции задает тип возвращаемого значения, имя функции, типы и число формальных параметров.

Тело функции — это составной оператор, содержащий операторы, определяющие действие функции. Тело функции начинается с фигурной скобки '{' и состоит из объявления переменных и исполняемых операторов. Именно эти операторы, входящие в тело функции, и определяют действие функции. Завершается тело функции закрывающей фигурной скобкой '}'.

Листинг 22.29. Пример определения функции

```
bit SostKnIzm(void)//Заголовок функции
{//--------------начало тела функции-------------
 bit tmp=0;
 if(P0!=0xff)tmp=1;
 return tmp;
}//--------------конец тела функции----------------

//================== Вызывающая подпрограмма ==========
  ...
```

```
if(SostKnIzm()) //Вызов подпрограммы SostKnIzm
  DecodSostKn();
```

В приведенном примере показано, как при помощи функции, возвращающей битовую переменную, можно повысить наглядность исходного текста программы. Оператор `if(SostKnIzm()) DecodSostKn();` практически не требует комментариев. Имя функции `SostKnIzm` показывает, что контролирует эта функция.

Необязательный спецификатор класса памяти задает класс памяти функции, который может быть `static` или `extern`.

При использовании спецификатора класса памяти `static` функция становится невидимой из других файлов программного проекта, т. е. информация об этой функции не помещается в объектный файл. Использование спецификатора класса памяти `static` может быть полезно для того, чтобы имя этой функции могло быть использовано в других файлах программного проекта для реализации совершенно других задач. Если функция, объявленная со спецификатором класса памяти `static`, ни разу не вызывалась в данном файле, то она вообще не транслируется компилятором и не занимает места в программной памяти микроконтроллера, а программа-компилятор языка программирования С-51 выдает предупреждение об этом. Это свойство может быть полезным при отладке программ и программных модулей.

Использование спецификатора класса памяти `extern` применяется для связывания подпрограмм, находящихся в других модулях (и, возможно, написанных на других языках программирования) с вызовом подпрограммы из данного программного модуля. В основном использование этого спецификатора класса памяти эквивалентно предварительному объявлению функции, которое будет рассмотрено далее. Если спецификатор класса памяти функции не указан, то подразумевается класс памяти `extern`, т. е. по умолчанию все подпрограммы считаются глобальными и доступными из всех файлов программного проекта.

Спецификатор типа функции задает тип возвращаемого значения и может задавать любой тип. Если спецификатор типа не задан, то по умолчанию предполагается, что функция возвращает числовое значение типа `int`. Функция не может возвращать массив или функцию, но это ограничение можно обойти, т. к. она может возвращать указатель на любой тип, в том числе и на массив и на функцию. Тип возвращаемого значения, задаваемый в определении функции, должен соответствовать типу в объявлении этой функции.

Функция возвращает значение, если ее выполнение заканчивается оператором `return`, содержащим некоторое выражение. Указанное выражение вычисляется, преобразуется, если необходимо, к типу возвращаемого значения и возвращается в точку вызова функции в качестве результата.

Листинг 22.30. Пример определения функции, возвращающей двоичный код числа, принятого по последовательному порту микроконтроллера

```
#include <reg51.h>/*Подключить описания внутренних регистров
                    микроконтроллера*/

char getkey ()     /*Заголовок функции, возвращающей байт, принятый по
                     последовательному порту*/
{while (!RI);      //Если последовательный порт принял байт,
   RI = 0;          //то подготовиться к приему следующего байта
   return (SBUF);   //и передать принятый байт в вызывающую подпрограмму.
}
```

В операторе `return` возвращаемое значение может записываться как в скобках, так и без них. Если функция определена как функция, возвращающая некоторое значение (подпрограмма-функция), а в операторе `return` при выходе из нее отсутствует выражение, то это может привести к непредсказуемым результатам.

Для функций, не использующих возвращаемое значение (подпрограмм-процедур), должен быть использован тип `void`, указывающий на отсутствие возвращаемого значения. Если оператор `return` не содержит выражения или выполнение функции завершается после выполнения последнего ее оператора (без выполнения оператора `return`), то возвращаемое значение не определено.

Листинг 22.31. Пример функции, не возвращающей значение

```
void putchar (char c)  /*Заголовок функции, передающей один байт через
                         последовательный порт                        */
{while (!TI);         /*Если передатчик последовательного порта готов
                        к передаче байта*/
 SBUF = c;             //то занести в буфер передатчика последовательного
 TI = 0;               //порта байт и начать передачу
}
```

Все переменные, объявленные в теле функции без указания класса памяти, имеют класс памяти `auto`, т. е. они являются локальными. Так как глубина стека в процессорах семейства MCS-51 ограничена 256 байтами, то при вызове функций аргументам назначаются конкретные адреса во внутренней памяти микроконтроллера и производится их инициализация. Управление передается первому оператору тела функции и начинается выполнение функции, которое продолжается до тех пор, пока не встретится оператор `return` или

последний оператор тела функции. Управление при этом возвращается в точку, следующую за точкой вызова, а локальные переменные становятся недоступными. При выходе из функции значения этих переменных теряются, т. к. при вызове других функций эти же ячейки памяти распределяются для их локальных переменных.

Если требуется, чтобы переменная, объявленная внутри подпрограммы, сохраняла свое значение при следующем вызове подпрограммы, то ее необходимо объявить с классом памяти `static`.

Параметры функций

Список формальных параметров — это последовательность объявлений формальных параметров, разделенная запятыми. Формальные параметры — это переменные, используемые внутри тела функции и получающие значение при вызове функции путем копирования в них значений соответствующих фактических параметров.

Листинг 22.32. Пример определения функции с одним параметром

```
int rus (unsigned char r)  //Заголовок функции
{//---------------начало тела функции-------------
 if (r>='A' && c<=' ')
      return 1;
   else
      return 0;
}//---------------конец тела функции----------------
```

В данном примере определена функция с именем `rus`, имеющая один параметр с именем `r` и типом `unsigned char`. Функция возвращает целое значение, равное 1, если параметр функции является буквой русского алфавита, или 0 в противном случае.

Если функция не использует параметров, то наличие круглых скобок обязательно, а вместо списка параметров рекомендуется указать слово `void`.

Листинг 22.33. Пример определения функции без параметров

```
void main(void)
{P0=0;       //Зажигание светодиода
 while(1);  //Бесконечный цикл
}
```

Порядок и типы формальных параметров должны быть одинаковыми в определении функции и во всех ее объявлениях. Поэтому желательно объявление

функции поместить в отдельный файл, который затем можно включить в исходные тексты программных модулей при помощи директивы `#include`. Типы фактических параметров при вызове функции должны быть совместимы с типами соответствующих формальных параметров. Тип формального параметра может быть любым основным типом, структурой, объединением, перечислением, указателем или массивом. Если тип формального параметра не указан, то этому параметру присваивается тип `int`.

Для формального параметра можно задавать класс памяти `register`, при этом для величин типа `int` спецификатор типа можно опустить. Однако все известные мне компиляторы языка программирования С-51 игнорируют этот спецификатор, т. к. расположение параметров в памяти микроконтроллера оптимизируется с точки зрения использования минимального количества необходимой внутренней памяти.

По умолчанию компиляторы стараются передавать параметры в функцию через регистры, тем самым максимально сохраняя внутреннюю память микроконтроллеров. Номера регистров, используемые для передачи параметров в функцию в зависимости от типа аргументов, приведены в табл. 22.8.

***Таблица 22.8.** Номера регистров, используемые для передачи параметров в функцию в зависимости от типа аргументов*

Номер аргумента	char, однобайтовый указатель	int, двухбайтовый указатель	long, float	Нетипизированные указатели
1	R7	R6,R7	R4 — R7	R1 — R3
2	R5	R4,R5	R4 — R7	R1 — R3
3	R3	R2,R3		R1 — R3

Поскольку при вызове функции значения фактических параметров копируются в локальные переменные, в теле функции нельзя изменить значения переменных в вызывающей функции. Например, нужно поменять местами значения переменных `x` и `y`:

```
/*    Неправильное использование параметров функции    */
void change (int x, int y)
{int k=x;
     x=y;
     y=k;
}
```

В данной функции значения локальных переменных `x` и `y`, являющихся формальными параметрами, меняются местами, но поскольку эти переменные

существуют только внутри функции change, значения фактических параметров, используемых при вызове функции, останутся неизменными.

Однако если в качестве параметра функции использовать указатель на переменную, то можно изменить значение переменной, адрес которой будет содержаться в указателе. Для того чтобы менялись местами значения фактических аргументов, можно использовать функцию, приведенную в следующем примере:

```
/*     Правильное использование параметров функции     */
void change (int *x, int *y)
{int k=*x;
     *x=*y;
     *y=k;
}
```

При вызове такой функции в качестве фактических параметров должны быть использованы не значения переменных, а их адреса:

```
change (&a,&b);
```

Предварительное объявление подпрограмм

В языке С-51 нет требования, чтобы определение функции обязательно предшествовало ее вызову. Определения используемых функций могут следовать за определением функции main, перед ним или находиться в другом файле. Однако для того, чтобы компилятор мог осуществить проверку соответствия типов передаваемых фактических параметров типам формальных параметров и, если необходимо, выполнить соответствующие преобразования, до вызова функции нужно поместить объявление (прототип) функции.

Если объявление функции не задано, то по умолчанию строится прототип функции на основе анализа первого вызова функции. Тип возвращаемого значения создаваемого прототипа int, а список типов и числа параметров функции формируется на основании типов и числа фактических параметров, используемых при данном вызове. Однако такой прототип не всегда согласуется с последующим определением функции. При размещении функции в другом файле или после оператора ее вызова рекомендуется задавать прототип функции. Это позволит компилятору либо выдавать диагностические сообщения, при неверном использовании функции, либо правильным образом преобразовывать типы аргументов при ее вызове.

Прототип — это явное объявление функции, которое предшествует определению функции. Тип возвращаемого значения при объявлении функции должен соответствовать типу возвращаемого значения в определении функции. Если прототип задан с классом памяти static, то и определение функции

должно иметь класс памяти static. Объявление (прототип) функции имеет следующий формат:

```
[спецификатор класса памяти] [спецификатор типа] имя функции ([список формальных параметров]);
```

В отличие от определения функции, в прототипе за заголовком сразу же следует точка с запятой, а тело функции отсутствует. Правила использования остальных элементов формата такие же, как и при определении функции.

Для функции, определенной в предыдущем разделе, прототип может быть записан в следующем виде:

```
int rus (unsigned char r);
```

При этом в объявлении функции имена формальных параметров могут быть опущены:

```
int rus (unsigned char);
```

Вызов функций

Вызов функции имеет следующий формат:

```
имя функции ([список выражений]);
```

Список выражений представляет собой список фактических параметров, передаваемых в функцию. Этот список может быть и пустым, если в определении функции отсутствуют параметры, но наличие круглых скобок обязательно. Примеры вызова функции (подпрограммы-процедуры):

```
DecodSostKn();
VypFunc();
```

Функция, если она возвращает какое-либо значение (подпрограмма-функция), может быть вызвана и в составе выражения, например:

```
y=sin(x);                  //sin - это имя подпрограммы-функции
if(rus(c))SvDiod=Gorit;    //rus - это имя подпрограммы-функции
```

Выполнение вызова функции происходит следующим образом:

1. Вычисляются выражения в списке выражений и подвергаются обычным арифметическим преобразованиям. Затем тип полученного фактического аргумента сравнивается с типом соответствующего формального параметра. Если они не совпадают, то либо производится преобразование типов, либо формируется сообщение об ошибке. Число выражений в списке выражений должно совпадать с числом формальных параметров. Если в прототипе функции указано, что ей не требуются параметры, а при вызове они указаны, формируется сообщение об ошибке.

2. Происходит присваивание значений фактических параметров соответствующим формальным параметрам.
3. Управление передается на первый оператор функции.

Поскольку имя функции является адресом начала тела функции, то в качестве обращения к функции может быть использовано не только имя функции, но и значение указателя на функцию. Это значит, что функция может быть вызвана через указатель на функцию. Пример объявления указателя на функцию:

```
int (*fun)(int x, int *y);
```

Здесь объявлена переменная `fun` как указатель на функцию с двумя параметрами: типа `int` и указателем на `int`. Сама функция должна возвращать значение типа `int`. Круглые скобки, содержащие имя указателя `fun` и признак указателя `*`, обязательны, иначе запись

```
int *fun (int x,int *y);
```

будет интерпретироваться как объявление функции `fun`, возвращающей указатель на `int`.

Вызов функции возможен только после инициализации значения указателя:

Листинг 22.34. Пример инициализации указателя на функцию

```
float (*funPtr)(int x, int y);
float fun2(int k, int l);
   ...
   funPtr=fun2;          /* инициализация указателя на функцию */
   (*funPtr)(2,7);       /* обращение к функции                */
```

В рассмотренном примере указатель на функцию `funPtr` описан как указатель на функцию с двумя параметрами, возвращающую значение типа `double`. и также описана функция `fun2`. В противном случае, т. е. когда указателю на функцию присваивается функция, описанная иначе, чем указатель, произойдет ошибка.

Рассмотрим пример использования указателя на функцию в качестве параметра функции, вычисляющей производную от функции `cos(x)`.

Листинг 22.35. Пример использования указателя на функцию в качестве параметра функции, вычисляющей производную от функции `cos(x)`

```
float proiz(float x,float dx,float(*f)(float x));
float fun(float z);
```

```
int main()
{float x;                  /*  точка вычисления производной */
 float dx;                 /*  приращение                   */
 float z;                  /*  значение производной         */

 scanf("%f,%f",&x,&dx);    /*  ввод значений x и dx         */
 z=proiz(x,dx,fun);        /*  вызов функции                */
 printf("%f",z);           /*  печать значения производной  */
}

/* функция, вычисляющая производную  */
float proiz(float x,float dx,float (*f)(float z))
{float xk,xk1;
 xk=fun(x);
 xk1=fun(x+dx);
 return (xk1/xk-1e0)*xk/dx;
}

float fun( float z)       /* функция, от которой вычисляется производная */
{return (cos(z));
}
```

Для вычисления производной от какой-либо другой функции можно изменить тело функции `fun` или использовать при вызове функции `proiz` имя другой функции. В частности, для вычисления производной от функции `cos(x)` можно вызвать функцию `proiz` в форме:

```
z=proiz(x,dx,cos);
```

а для вычисления производной от функции `sin(x)` в форме

```
z=proiz(x,dx,sin);
```

Рекурсивный вызов подпрограмм

В стандартном языке программирования С все функции могут быть вызваны сами из себя или использоваться различными программными потоками одновременно. Для этого все локальные переменные располагаются в стеке. В микроконтроллерах семейства MCS-51 ресурсы внутренней памяти данных ограничены, поэтому в языке программирования С-51 для функций локальные переменные по умолчанию располагаются не в стеке, а непосредственно во внутренней памяти микроконтроллера. Если же подпрограмма должна вызываться рекурсивно, то ее необходимо объявить как программу с повторным вызовом (`reentrant`):

```
return_type funcname ([args]) reentrant
```

Классический пример рекурсии — это математическое определение факториала n!:

```
n! = 1    при n=0;
n*(n-1)! при n>1 .
```

Функция, вычисляющая факториал, будет иметь следующий вид:

```
long fakt(int n) reentrant
{return ((n==1)?1:n*fakt(n-1));
}
```

Подпрограммы обработки прерываний

Атрибут `interrupt` позволяет объявить подпрограмму-процедуру обработки сигналов прерываний, поступающих от внешних устройств. Подпрограмма-процедура с этим атрибутом вызывается при получении микроконтроллером соответствующего сигнала прерывания. Подпрограмма обработки прерываний не может быть подпрограммой-функцией и не может иметь переменные-параметры. Формат использования атрибута:

```
interrupt N;
```

где `N` — любое десятичное число от 0 до 31.

Число `N` определяет номер обрабатываемого прерывания. При этом номер 0 соответствует внешнему прерыванию от вывода INT0, номер 1 — прерыванию от таймера 0, номер 2 — внешнему прерыванию от вывода INT1 и т. д. Пример подпрограммы-обработчика прерывания от таймера 0:

```
void IntTim0(void) interrupt 1
{TH0=25; TL0=32;   /*Задать новый интервал времени таймера T0          */
 TF=0;             /*Сбросить флаг таймера T0 для разрешения следующего
                     прерывания от данного таймера                     */
}
```

При работе с прерываниями определяющим фактором является время реакции на прерывание. Для того чтобы не сохранять содержимое используемых регистров микроконтроллера в стеке, в микроконтроллерах предусмотрено использование отдельных регистровых банков. В языке программирования С-51 для этого необходимо в объявлении подпрограммы указать используемый ею банк регистров. Для этого служит атрибут `using`:

```
void IntTim0(void) interrupt 2 using 1
{TH0=25; TL0=32;   /*Задать новый интервал времени таймера T0          */
 TF=0;             /*Сбросить флаг таймера T0 для разрешения следующего
                     прерывания от данного таймера                     */
}
```

Области действия переменных и подпрограмм

Любой объект, который объявляется в программе, написанной на языке программирования С-51, имеет область действия. В языке программирования С область действия объекта распространяется на всю программу, т. е. любой объект является глобальным.

К объектам запрещено обращение до того, как они будут объявлены, поэтому при обращении к объекту, объявленному в другом программном модуле, этот объект должен быть объявлен в данном программном модуле как `external`. Например:

```
extern char code ERROR [];              //Строка сообщения об ошибке
extern struct mrec current;             //Текущее измерение
extern struct interval setinterval; //Значения установленных интервалов
extern struct interval counter ;        //Счетчик интервалов
```

При использовании внутри модуля переменной, уже объявленной в другом модуле, могут возникнуть проблемы при связывании программы. Если использование переменной с тем же именем, что и в другом модуле делает программу более наглядной, то можно в одном из модулей (или во всех сразу) применить одно и то же имя с атрибутом `static`.

Например:

```
static int i;    /*Глобальная переменная, недоступная из других модулей
                   (а значит, и не мешающая им)*/
void tmp(void)
{i=5;
}
```

Некоторые блоки могут содержать структурные операторы, как показано в следующих примерах (рис. 22.6 и 22.7). Использование внутренних блоков позволяет объявлять локальные переменные. Использование локальных переменных позволяет применять одни и те же ячейки памяти для различных переменных и тем самым экономить ресурсы памяти данных.

Локальные переменные существуют только в пределах структурного оператора. Если внутри структурного оператора были присвоены локальным переменным какие-то значения, то не следует ожидать, что вернувшись снова в этот оператор, переменные сохранят присвоенные ранее или вычисленные значения.

В случае, если требуется сохранить значение переменных и при повторном вхождении в подпрограмму или структурный оператор, эту переменную необходимо объявить с атрибутом `static`.

Например:

```
void tmp(void)
{static int  i=5;
}
```

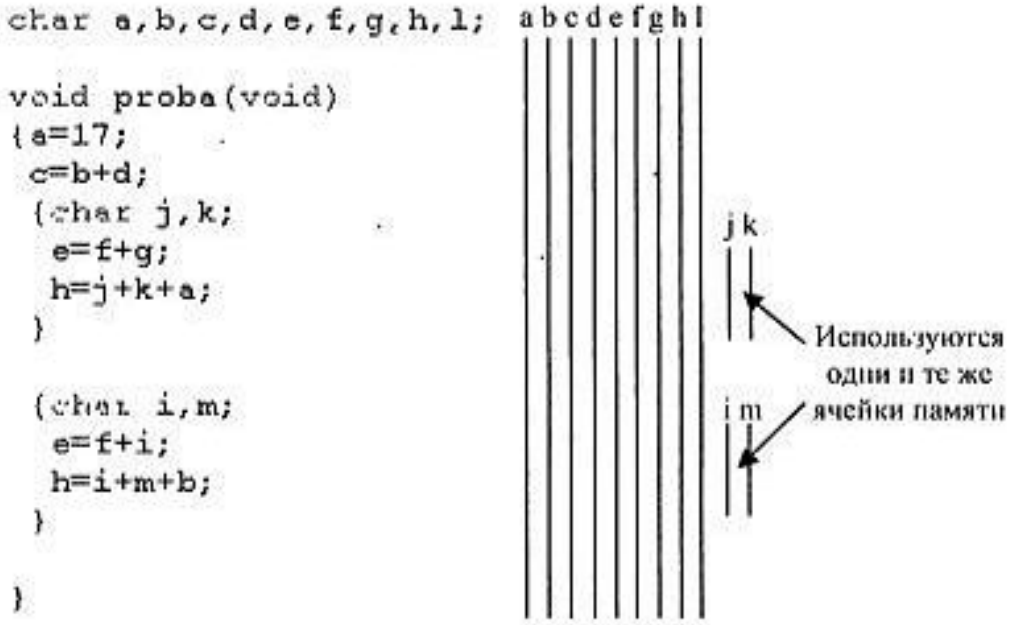


Рис. 22.6. Области действия глобальных и локальных переменных

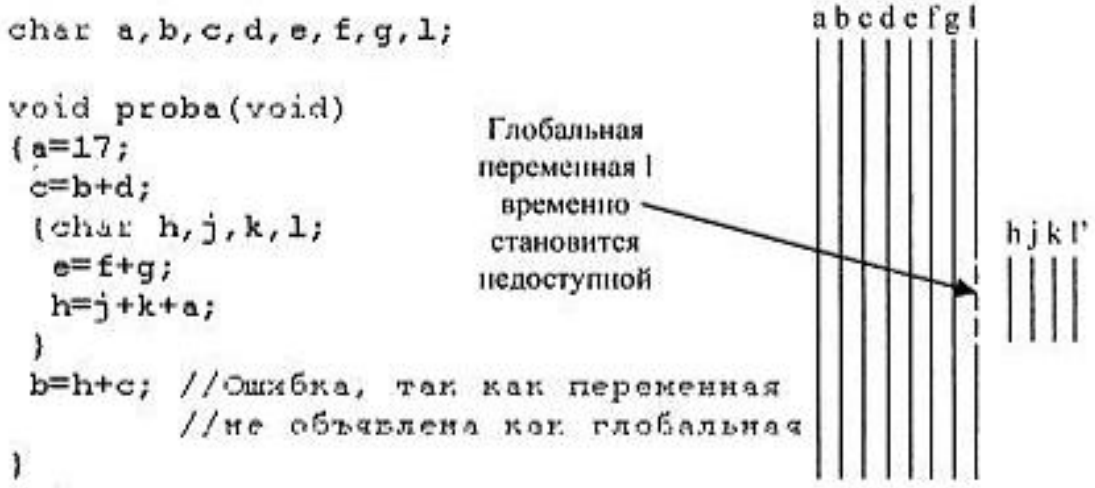


Рис. 22.7. Области действия глобальных и локальных переменных

То же самое относится и к подпрограммам. Если требуется объявить подпрограмму, которая не должна быть видна из других модулей, то ее необходимо объявить с атрибутом `static`. Это, кроме прочего, приведет к тому, что если эта подпрограмма не используется и в данном модуле, то она просто не будет транслироваться и занимать место в памяти программ. Пример объявления подпрограммы с атрибутом `static`:

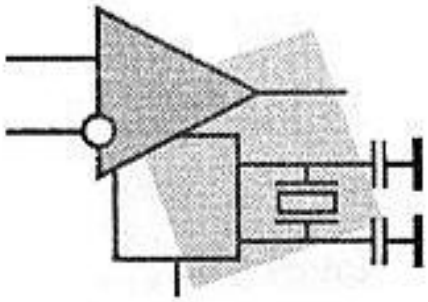
```
static void tmp(void)
{i=5;
}
```

Иногда необходимо локальные переменные хранить не в одних и тех же ячейках памяти, а в стеке. Это требуется, например, при вызове подпрограммы самой из себя. В таком случае подпрограмма должна быть объявлена как `reentrant`.

Итоги

В данной главе было приведено краткое описание языка программирования С-51. Использование этого языка позволяет сократить время разработки программ для микроконтроллеров. В большинстве случаев ресурсов выбранного микроконтроллера более чем достаточно для реализации требуемого алгоритма. Это позволяет использовать для создания программы язык С-51. В главе показаны примеры использования С-51 для управления микроконтроллером. Этот язык позволяет создавать достаточно сложные программы при минимальных затратах времени. Однако следует помнить, что ничего не бывает бесплатно. Избавляя от одних проблем, язык программирования С-51 приводит к другим. Как это неоднократно подчеркивалось в тексте главы, при программировании на языке С-51 необходимо чрезвычайно тщательно выбирать типы используемых переменных и следить за их правильным использованием. При неверном выборе типов можно значительно увеличить объем программы и снизить ее быстродействие по сравнению с программой, написанной на языке программирования ассемблер.

ГЛАВА 23



Язык программирования ASM-51

Несмотря на то, что в настоящее время программирование микроконтроллеров ведется в основном на языке С, применение ассемблера остается актуальной задачей. Это обусловлено несколькими причинами. Одна из них та, что наиболее дешевые модели микроконтроллеров обладают настолько ограниченными ресурсами, что программа, написанная на С, просто не умещается во внутреннюю память микроконтроллера. А ведь применение именно наиболее дешевых моделей позволяет выигрывать в конкурентной борьбе при производстве аппаратуры. Вторая причина — повышенные требования к быстродействию микроконтроллера. Эти требования возникают при выполнении микроконтроллером некоторых задач обработки сигналов. В этом случае возможно совместное применение для написания программы как языка С, так и языка программирования ассемблер. В настоящее время для создания программного обеспечения для микроконтроллеров наиболее распространены ассемблеры ASM-51 и A51, являющийся развитием языка ASM-51. Мы рассмотрим более простую версию — язык программирования ASM-51.

Язык программирования ASM-51 поддерживает модульное написание программ. Графическое изображение процесса разработки программы совместно на языках программирования С-51 и ASM-51 приведено на рис. 23.1.

Файл, в котором хранится программа, написанная на языке ASM-51 (исходный текст программы), называется *исходным модулем*. Его можно создать, используя любой текстовый редактор. Для файла исходного текста программы принято использовать следующие расширения: asm, a51, srs, s51.

Получить объектный модуль можно, указав имя исходного модуля программы в качестве параметра вызова программы-транслятора в DOS-строке или строке командного файла:

```
asm51.exe modul.asm
```

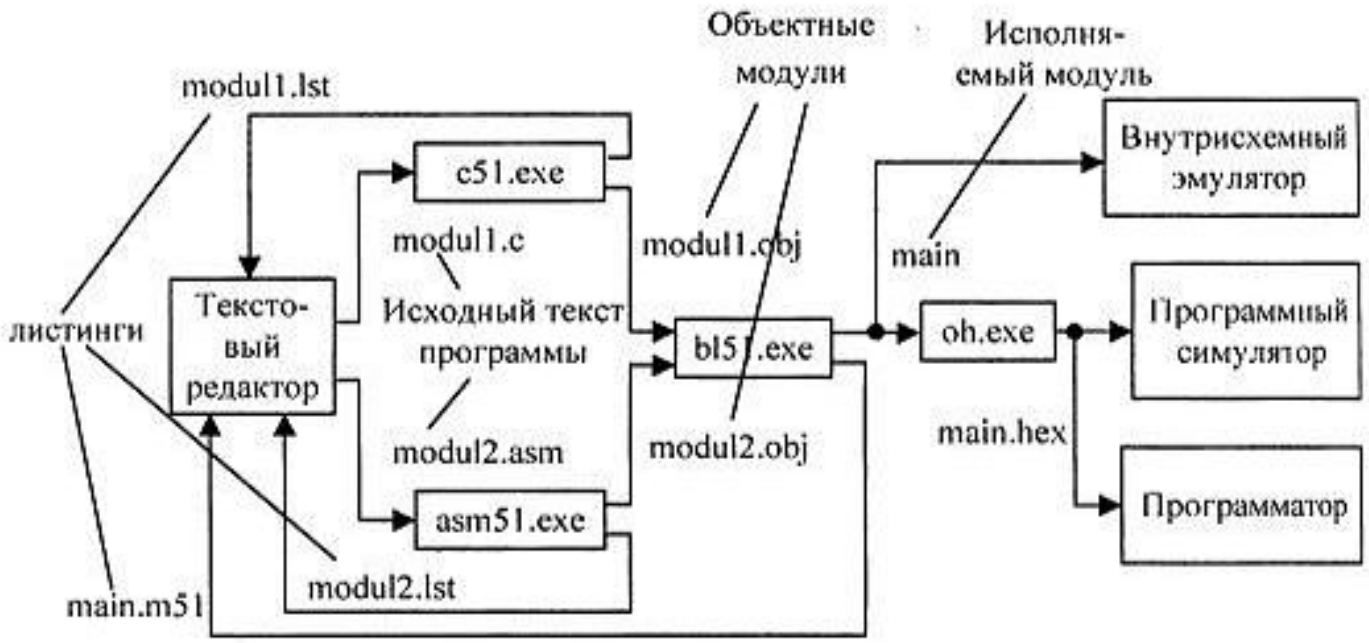


Рис. 23.1. Схема процесса написания программы на языке программирования ASM-51

При этом одновременно будет сформирован файл листинга, содержащий список синтаксических ошибок. До тех пор, пока не будут исправлены все синтаксические ошибки, выходной файл объектного модуля не формируется. Исправление ошибок производится в текстовом редакторе, в котором и был написан исходный текст программного модуля. Затем процесс трансляции этого модуля повторяется. Описанный процесс показан на рис. 23.1 стрелочками. После получения всех объектных модулей можно приступать к формированию исполняемого модуля. В простейшем случае программа состоит из одного-единственного модуля.

Получить исполняемый модуль программы можно, указав все имена объектных модулей, входящих в состав программы в качестве параметров вызова программы-редактора связей (компоновщика) в DOS-строке или строке командного файла:

```
bl51.exe main.obj, modul1.obj, modul2.obj
```

Имя исполняемого модуля программы по умолчанию совпадает с именем первого объектного файла в списке параметров строки запуска редактора связей. Исполняемый модуль программы записывается в файл с именем, но без расширения. Именно это имя и является именем программного проекта. Если имя программного проекта не совпадает с именем первого модуля (что чаще всего и бывает), то можно воспользоваться следующей командой:

```
bl51.exe main.obj, modul1.obj, modul2.obj to programma.abs
```

В этой команде `bl51.exe` — это имя редактора связей, `main.obj, modul1.obj, modul2.obj` — имена программных модулей, а `to programma.abs` — приказ создать исполняемый модуль программы с именем `programma.abs`.

Большинство программаторов не может работать с объектным форматом исполняемого модуля программы, поэтому для загрузки машинного кода в микроконтроллер необходимо преобразовать объектный формат исполняемого модуля в общепринятый для программаторов HEX-формат. При таком преобразовании вся отладочная информация, содержащаяся в исполняемом модуле, теряется. Машинный код процессора, записанный в HEX-формате, называется загрузочным модулем.

Загрузочный модуль программы можно получить при помощи программы-преобразователя `oh.exe`, передав ей при вызове в качестве параметра имя файла исполняемого модуля, например:

```
oh.exe main
```

После того, как программные модули успешно оттранслированы, размещены по конкретным адресам и связаны между собой, для отладки программы можно воспользоваться внутрисхемным эмулятором. Это инструментальное средство позволяет отображать переменные программы на дисплее персонального компьютера и оказывает значительную помощь при отладке программ непосредственно на разрабатываемой аппаратуре. Необходимое для отладки программ оборудование показано на рис. 22.4. Отдельные участки программы могут быть отлажены и при помощи специальных программ-отладчиков (симуляторов).

Как эмуляторы, так и симуляторы предназначены для обнаружения логических ошибок, содержащихся в исходном тексте программы, поэтому после обнаружения ошибки приходится исправлять исходный текст и заново транслировать его для получения исправленной версии исполняемого или загрузочного модуля.

Тем не менее, окончательная отладка программного обеспечения для разрабатываемого устройства производится только после записи загрузочного модуля в память микроконтроллерной системы. Именно на этом этапе находятся и устраняются последние ошибки, допущенные при написании программы.

Исходный текст программы на языке программирования ASM-51

Исходный текст программы представляет собой последовательность операторов языка программирования ASM-51, сгруппированных в сегменты и оформленных в виде файла.

Оператор — это базовая конструкция языка программирования, определяющая действия в программе. В языке программирования ASM-51 *в одной строке может быть записан только один оператор!* Максимальный допустимый размер строки — 255 символов. Признаком конца оператора является символ "возврат каретки", который вводится в текст программы при нажатии клавиши <Enter> в конце строки.

Оператор языка программирования ASM-51 состоит из трех полей:

```
<поле метки> <поле операции> <поле комментария>
```

Любое из полей, в том числе и все поля, могут отсутствовать. Оператор, в котором все поля отсутствуют, называется пустым оператором. Он используется для увеличения наглядности программы. Пример оператора, записанного на языке программирования ASM-51, приведен на рис. 23.2.

Поле метки используется для записи меток или имен директив. Метка представляет собой определяемое программистом имя, заканчивающееся двоеточием, и используется для организации условных и безусловных переходов, а также для объявления переменных и констант.

Имя директивы представляет собой определяемое программистом имя и используется для объявления переменных, констант и имен программных сегментов.

Рис. 23.2. Пример оператора, записанного на языке программирования ASM-51

Если в операторе присутствует только метка, то она помечает ближайший следующий оператор, в котором присутствует инструкция процессора или директива ассемблера. Использование оператора, содержащего только метку, может быть вызвано либо слишком большой длиной самой метки, либо необходимостью присвоить одному непустому оператору несколько меток. Наиболее яркий пример использования нескольких меток — это когда одна метка используется в качестве имени подпрограммы, а другая обозначает начало цикла. Пример использования оператора, содержащего только метку, приведен в листинге 23.1.

Листинг 23.1. Пример использования оператора, содержащего только метку

```
PodprogrammaPeredachiDannyh:  ;Помечается следующий оператор
CopyNxtByte: Mov R0, A
             Mov A, @R0
             DJNZ R0, CopyNxtByte
```

Поле операции используется для записи директивы языка или команды микроконтроллера и состоит из мнемонического обозначения команды и одного или нескольких операндов. При записи машинной команды первый операнд всегда является приемником результата операции. Второй операнд всегда является источником данных для выполняемой операции. Если в команде для выполнения операции требуется два источника данных (например, операция суммирования), то первый операнд используется и в качестве источника, и в качестве приемника данных. В качестве операндов могут использоваться адреса ячеек памяти, обозначения регистров или метки операторов. Операнды отделяются друг от друга запятыми. Вместе с запятыми для увеличения читаемости программы допускается использование символов интервала (пробел или табуляция). Поле операции подробно показано на рис. 23.2.

Поле комментария начинается с символа "точка с запятой" (;) и может содержать любые ASCII- или ANSI-символы. Это поле используется для записи пояснений к программе. Оператор, в котором присутствует только поле комментария, используется для увеличения наглядности программы. Примеры записи комментариев на языке программирования ASM-51 приведены в листинге. 23.2.

Листинг 23.2. Примеры записи комментариев на языке программирования ASM-51

```
;-------------------------------------------------------------------
; ПОДПРОГРАММА ВЫЧИСЛЕНИЯ ФУНКЦИИ
;-------------------------------------------------------------------
; X + Y * Z
```

Символы языка ASM-51

Символы исходной программы представляют собой подмножество таблиц символов ASCII для DOS и ANSI для WINDOWS. В исходном тексте программы, написанном на языке программирования ASM-51, могут быть использованы следующие символы:

- символы интервала;
- буквы;

- знаки;
- цифры.

Символы интервала определяют один или несколько пробелов в предложении исходного модуля. К этим символам относятся "пробел" и " табуляция".

В качестве *букв* воспринимаются латинские буквы верхнего и нижнего регистра:

```
A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X,
Y, Z, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v,
w, x, y, z.
```

Кроме того, в качестве букв могут быть использованы символы вопросительного знака (?) и подчеркивания (_).

Ниже приведен перечень *цифр*:

```
0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
```

Для записи шестнадцатеричных цифр дополнительно могут быть использованы следующие символы:

```
a, b, c, d, e, f, A, B, C, D, E, F.
```

Наименования *знаков* и их обозначение приведено в табл. 23.1.

***Таблица 23.1.** Наименования знаков и их обозначение*

Наименование	Обозначение	Наименование	Обозначение
Номер	#	Точка	.
Знак денежной единицы	$	Дробная черта	/
Апостроф	'	Двоеточие	:
Круглая скобка левая	(	Точка с запятой	;
Круглая скобка правая	)	Меньше	<
Звездочка	*	Равно	=
Плюс	+	Больше	>
Запятая	,	Вопросительный знак	?
Минус	-	Коммерческое эт	@

Знаки, комбинации знаков (<>, >=, <=), а также символы интервала являются разделителями конструкций языка. До и после знака-разделителя в любой конструкции языка могут быть вставлены символы интервала.

ASCII- или ANSI-символы, не входящие в перечень основных символов алфавита языка, считаются дополнительными. Они могут использоваться в

комментариях для пояснений в исходном тексте программы, а также для определения символьных констант.

Из символов формируются идентификаторы и числа.

Идентификаторы

Идентификатор — это символическое обозначение объекта программы. В качестве идентификатора может быть использована любая последовательность букв и цифр. При этом в качестве буквы может быть использована любая буква латинского алфавита, а также вопросительный знак (?) и знак "нижнее подчеркивание" (_). *Идентификатор может начинаться только с буквы!* Это позволяет отличать его от числа. В идентификаторах язык программирования ASM-51 различает буквы верхнего и нижнего регистров.

Количество символов в идентификаторе ограничено только длиной строки (255 символов), но при этом транслятор языка программирования ASM-51 различает идентификаторы по первым 31 символам.

Примеры записи идентификаторов:

```
ADD5, FFFFH, ?, ALFA_1.
```

В языке программирования ASM-51 имеются три категории идентификаторов:

1. Ключевые слова.
2. Встроенные имена:
3. Определяемые имена.

Ключевые слова

Ключевое слово является определяющей частью оператора языка программирования. Значения ключевых слов языка ASM-51 не могут быть изменены или переопределены в программном модуле каким-либо образом. Ключевому слову не может быть назначено имя-синоним. *Ключевые слова могут быть написаны буквами как верхнего, так и нижнего регистров.* То есть ключевое слово `MOV` и ключевое слово `mov` полностью эквивалентны.

В языке программирования ASM-51 имеются следующие категории ключевых слов:

- инструкции;
- директивы;
- вспомогательные слова;
- операции.

Инструкции по форме записи совпадают с мнемоническими обозначениями команд микроконтроллеров семейства MCS-51 и совместно с операндами составляют команды микроконтроллера. Список инструкций:

```
ACALL, ADD, ADDC, AJMP, ANL, CALL, CJNE, CLR, CPL, DA, DEC, DIV, DJNZ,
INC, JB, JBC, JC, JMP, JNB, JNC, JNZ, JZ, LCALL, LJMP, MOV, MOVC, MOVX,
MUL, NOP, ORL, POP, PUSH, RET, RETI, RL, RLC, RR, RRC, SETB, SJMP, SUBB,
SWAP, XCH, XCHD, XRL.
```

Директивы совместно с *вспомогательными словами* определяют действия, которые должны быть выполнены ассемблером в процессе преобразования исходного текста программы в объектный код. В языке программирования ASM-51 используются:

Директивы:

```
BIT, BSEG, CODE, CSEG, DATA, DB, DBIT, DS, DSEG, DW, END, EQU, EXTRN,
IDATA, ISEG, NAME, ORG, PUBLIC, RSEG, SEGMENT, SET, USING, XDATA, XSEG.
```

Вспомогательные слова:

```
AT, BIT, BITADDRESSABLE, CODE, DATA, IDATA, INBLOCK, INPAGE, NUMBER,
PAGE, UNIT, XDATA.
```

Операции выполняются ассемблером в процессе вычисления выражений на этапе трансляции исходного текста программы для определения конкретного числа, которое используется в команде. Перечень операций, использующихся языком программирования ASM-51:

```
AND, EQ, GE, GT, HIGH, LE, LOW, LT, MOD, NE, NOT, OR, SHL, SHR, XOR.
```

Встроенные имена

Встроенные имена присвоены адресам регистров специальных функций, адресам флагов специальных функций AR0-AR7, рабочим регистрам R0-R7 текущего банка регистров, а также аккумулятору A и флагу переноса C. Список встроенных имен приведен в табл. 23.2.

***Таблица 23.2.** Встроенные имена*

Имя	Регистр
A	Аккумулятор
R0-R7	8-разрядный рабочий регистр текущего банка рабочих регистров
AR0-AR7	Адреса 8-разрядных рабочих регистров текущего банка рабочих регистров
DPTR	16-разрядный регистр-указатель данных
PC	16-разрядный счетчик команд

Таблица 23.2 (окончание)

Имя	Регистр
C	Флаг переноса
AB	Регистровая пара, состоящая из аккумулятора A (старшая часть) и регистра B (младшая часть)

Определяемые имена

Определяемые имена объявляются пользователем. В языке программирования ASM-51 имеются следующие категории определяемых идентификаторов:

- метки;
- внутренние и внешние переменные адресного типа;
- внутренние и внешние переменные числового типа;
- имена сегментов;
- названия программных модулей.

Числа и литеральные строки

В языке программирования ASM-51 используются целые беззнаковые числа, представленные в двоичной, восьмеричной, десятичной и шестнадцатеричной формах записи. Для определения основания системы счисления используется суффикс (буква, следующая за числом):

- B определяет двоичное число (разрешенные цифры 0, 1);
- Q\O определяет восьмеричное число (разрешенные цифры 0, 1, 2, 3, 4, 5, 6, 7);
- D определяет десятичное число (разрешенные цифры 0, 1, 2, 3, 4, 5, 6, 7, 8, 9);
- H определяет шестнадцатеричное число (разрешенные цифры 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F).

Для десятичного числа суффикс может отсутствовать. Количество символов в числе ограничено размером строки, однако значение числа определяется по модулю 2^{16} (т. е. диапазон значений числа находится в пределах от 0 до 65 535).

Примеры записи чисел:

```
011101b, 1011100B, 735Q, 456o, 256, 0fah, 0CBH
```

Число всегда начинается с цифры. Это необходимо для того, чтобы отличать шестнадцатеричное число от идентификатора.

Например:

`ADCH` — идентификатор;

`0ADCH` — число.

Часто бывает удобно выполнить некоторые вычисления для того, чтобы получить число. При этом если поместить в текст программы предварительно вычисленное на калькуляторе значение, то может возникнуть вопрос: откуда взялось это значение. Лучше ввести формулу расчета и сами значения непосредственно в исходный текст программы. Язык программирования ASM-51 позволяет выполнять беззнаковые операции над числами. В таких выражениях допустимо использовать следующие арифметические операции:

- `+` — суммирование;
- `-` — вычитание;
- `*` — умножение;
- `/` — деление;
- `mod` — вычисление остатка от целочисленного деления.

В языке программирования ASM-51 также определена одноместная операция изменения знака — "минус" '–'. Для нее требуется только один операнд — тот, которому она предшествует.

Часто требуется выполнять операции в определенном порядке, отличающемся от принятого по умолчанию. Для изменения порядка выполнения операций можно воспользоваться скобками. Более того, использование скобок в ряде случаев повышает наглядность программы и тем самым уменьшает время ее отладки.

Кроме арифметических операций в выражениях допустимо использование следующих логических операций:

- `not` — побитовая инверсия операнда;
- `and` — логическое "И";
- `or` — логическое "ИЛИ";
- `xor` — "исключающее ИЛИ" (суммирование по модулю два);
- и функций выделения старшего '`HIGH`' и младшего '`LOW`' байта шестнадцатиразрядного числа.

Пример использования выражений языка программирования ASM-51 для определения числовой константы приведен в листинге 23.3.

Листинг 23.3. Пример использования выражений для определения числовой константы

```
Init:
;----------Настроить таймер T0-----------------------------------------
  mov TMOD,#00000001b
           ;     ||||
           ;     ||++-------Выбрать режим 16-разрядного таймера
           ;     |+---------Использовать внутреннюю синхронизацию
           ;     +----------Запретить управление таймером от ножки INT0

  mov TH0, #HIGH(-(F_ZQ/12)*10)  ;Выражение для определения
  mov TL0, #LOW(-(F_ZQ/12)*10)   ;константы

  setb TR0        ;Включить таймер 0
```

Часто операнд используется для представления символов на экране дисплея. В этом случае для определения его значения удобнее воспользоваться не числом, а литеральной константой. Литеральная константа заключается в апострофы:

```
  'a', 'W'
  mov SBUF, #'Б'  ;Передать по последовательному порту ANSI код буквы 'Б'
```

Часто на экране дисплея приходится отображать не одну букву, а целые фразы. Их удобно запоминать в памяти программ, а затем передавать на дисплей при помощи специальной подпрограммы. Для записи фраз в памяти программ можно воспользоваться литеральными строками, для ввода которых в память программ удобно воспользоваться директивой DB:

```
Nadp: DB 'Ошибка в блоке 5'
```

В этом случае каждый символ строки заменяется отдельным байтом и запоминается в ПЗУ памяти программ. Начало строки обязательно помечается при помощи метки. Для увеличения наглядности программы следует содержание надписи отобразить в имени использованной метки.

Директивы языка программирования ASM-51

Любой ассемблер всегда включает в себя команды микроконтроллера, но ими не ограничивается набор операторов этого языка программирования. Дело в том, что нужно уметь управлять самим процессом трансляции программы.

Первое, чем приходится по необходимости заниматься, если в программе используются только команды микроконтроллера — это вручную распределять

память данных микроконтроллера. При таком распределении приходится помнить, какой вид данных находится в каждой конкретной ячейке памяти, и указывать значение адреса памяти в качестве операнда команд. При необходимости изменить местоположение данных в ячейках памяти приходится просматривать всю программу и изменять соответствующие команды.

При чтении программы трудно отличить константы от переменных, ведь в командах они отличаются только видом адресации. Это приводит к увеличению времени написания программы, т. к. каждый раз приходится решать загадку: что же подразумевалось под конкретной цифрой — переменная, константа или маска?

Преодолеть эту трудность можно при помощи идентификаторов. Можно назначить какой-либо ячейке памяти идентификатор и работать с ним, как с переменной. Более того! При необходимости изменить адрес переменной в памяти данных можно просто изменить значение идентификатора, а не просматривать всю программу, каждый раз решая, является ли найденное число адресом переменной, маской или константой!

Директива equ позволяет назначать имена для констант и значения адресов для переменных. При использовании этой директивы можно назначить идентификатору переменной адрес в одном месте программы и пользоваться идентификатором этой переменной во всей программе.

Правда, за использование идентификатора именно в качестве переменной отвечает программист. Если он попытается использовать адрес переменной как константу, то транслятор не выдаст никакого сообщения об ошибке. Он спокойно оттранслирует оператор. При использовании назначения адресов переменных с помощью директивы equ изменение адреса при необходимости можно сделать в одном месте программы, а не просматривать всю программу, судорожно пытаясь вспомнить, где же еще была использована эта ячейка памяти. Все необходимые изменения в машинном коде программы сделает сам транслятор!

Пример назначения имен переменных с использованием директивы equ приведен в листинге 23.4.

Листинг 23.4. Назначение имен констант и адресов переменных при помощи директивы equ

```
FuncSet     equ 0x20
_8bit       equ 0x10

DispDat equ P0    ;Шина данных ЖКИ
RDS     equ P1_2 ;Чтение команды ЖКИ
```

```
RW      equ P1_1 ;Сигнал выбора записи/чтения
E       equ P1_0 ;Строб синхронизации ЖКИ

  mov DispDat,#(FuncSet or _8bit);Выставить на шине данных
  setb E                         ;команду установки функции
  clr E                          ;и выдать стробирующий сигнал
```

Как видно из приведенного примера, использование идентификаторов значительно повышает наглядность программы, т. к. в них отражается назначение соответствующих констант и переменных.

При помощи директивы `equ` можно назначать не только переменные, но и константы. Как уже говорилось ранее, будет ли использован идентификатор как переменная или как константа, зависит от команд и видов адресации, которые использует программист. И даже если адрес переменной и константа будут обладать одинаковым числовым значением, то лучше им назначить разные имена. (Никто не гарантирует, что в дальнейшем, в процессе отладки программы, не придется изменить конкретное значение константы.)

В приведенном в листинге 23.4 примере имена `FuncSet` и `_8bit` являются константами, но это становится понятным только по использованию непосредственной адресации в инструкции микроконтроллера `mov`.

Один раз назначенный при помощи директивы `equ` идентификатор уже не может быть изменен при дальнейшем написании программы, и при повторной попытке назначения точно такого же имени идентификатора транслятором будет выдано сообщение об ошибке.

Директива `set`. Если требуется в различных местах программы назначать одному и тому же идентификатору различные значения, то нужно пользоваться директивой `set`. Она используется точно так же, как директива `equ`, поэтому иллюстрировать это примером не будем.

Константы, назначаемые директивами `equ` или `set`, могут быть использованы только в качестве операндов команд микроконтроллера. В то же время достаточно часто требуется работа с таблицей констант, такой, как таблица перекодировки, таблицы элементарных функций или синдромы помехоустойчивых кодов. Такие константы используются не на этапе трансляции для формирования машинного кода инструкций, а при исполнении программы. Они заносятся в память программ микроконтроллера. Для помещения значений констант в память программ микроконтроллера используются директивы `db` и `dw`.

Директива `db` используется для занесения в память программ однобайтовых констант. Пример использования директивы `db` для формирования таблицы перекодировки двоично-десятичного кода в семисегментный код приведен в листинге 23.5.

Листинг 23.5. Создание таблицы констант при помощи директивы db

```
Decod:
  mov  DPTR,#TabDecod
  movc A,@A+DPTR
  ret

TabDecod:;-abcdefg
        DB 01111110b ;символ '0'
        DB 00110000b ;символ '1'
        DB 01101101b ;символ '2'
        DB 01111001b ;символ '3'
        DB 00110011b ;символ '4'
        DB 01011011b ;символ '5'
        DB 01011111b ;символ '6'
        DB 01110000b ;символ '7'
        DB 01111111b ;символ '8'
        DB 01111011b ;символ '9'
```

```
Расположение сегментов в
семисегментном индикаторе
          a
         ---
      f|   |b
         ---
      e|   |c
         ---
          d
```

В этом примере использована подпрограмма-функция, перекодирующая двоично-десятичное число в семисегментный код при помощи таблицы TabDecod. В эту функцию двоично-десятичный код передается через аккумулятор и через этот же регистр в вызывающую программу возвращается семисегментный код.

В директиве db можно задавать сразу несколько констант, разделенных запятыми. Можно одновременно использовать все системы счисления, но обычно имеет смысл снабдить каждую константу комментарием, как это сделано в предыдущем примере. Так программа становится более понятной и ее легче отлаживать.

Эта же директива позволяет легко размещать в памяти строки, которые в дальнейшем потребуется высвечивать на встроенном дисплее или экране дисплея универсального компьютера, подключенного к разрабатываемому устройству через какой-либо интерфейс. Пример использования директивы db для занесения строк в память программ микроконтроллера приведен в листинге 23.6.

Листинг 23.6. Применение директивы db для размещения строк в памяти программ микроконтроллера

```
  mov R7,#(EndNadp-NadpSvjazUst);Занести количество символов в строке
  mov DPTR,#NadpSvjazUst        ;Подготовиться к передаче первого символа
```

```
PrdSledSmv:
  clr A                          ;
  movc A,@A+DPTR                 ;Считать очередной символ надписи
  inc  DPTR                      ;

  call PrdSmv                    ;Передать очередной символ надписи, и
  djnz R7,PrdSledSmv             ;если это был последний символ надписи,
  ret                           ;то выйти из подпрограммы

NadpSvjazUst: db 'Связь установлена',10,13
EndNadp:
```

Директива `dw` позволяет заносить в память программ двухбайтовые числа. В ней, как и в директиве `db`, можно записывать несколько чисел, разделенных запятой. Пример фрагмента программы приведен в листинге 23.7.

Листинг 23.7. Применение директивы dw

```
0016 0001          264       dw 1,2,0abh,'a','QW'
0018 0002
001A 00AB
001C 0061
001E 5157
```

В листинге 23.7 фрагмент программы приведен для того, чтобы можно было проследить, какие байты заносятся в память программ микроконтроллера. В самой правой колонке листинга приведены адреса, в которые будут занесены числа, являющиеся операндами директивы `dw`. В следующей колонке приведены двухбайтовые числа, которые будут заноситься в память программ микроконтроллера. ✧ *Обратите внимание*: несмотря на то, что первые два операнда директивы `dw` состоят только из одной цифры, в память микроконтроллера заносятся четыре шестнадцатеричных цифры (двухбайтовое число).

При трансляции исходного текста программ по умолчанию предполагается, что первая команда расположена по нулевому адресу. Адрес последующих команд зависит от длины и количества предыдущих команд. Пример начального участка программы приведен в листинге 23.8.

Листинг. 23.8. Начальный участок программы

```
LOC  OBJ  LINE     SOURCE

0000 78FF    1       mov R0,#255      ;Обнулить 255 ячеек памяти
0002 E4      2       clr A
```

```
0003          3    ClrOZU:
0003 F6       4         mov @R0,A          ;Обнулить очередную ячейку памяти
0004 D8FD     5         djnz R0,ClrOZU     ;Если все ячейки обнулены,
              6                            ;то продолжить выполнение программы
```

Иногда нужно расположить команду по определенному адресу. Наиболее часто это требуется при использовании прерываний, когда первая команда подпрограммы-обработчика прерываний должна быть расположена точно в ячейке с определенным адресом (вектором), зависящем от источника прерывания. Векторы размещаются в начальных ячейках программной памяти и занимают 8 байтов каждый. Если эта область не используется целиком, то можно применять команду `nop` для заполнения промежутков между векторами прерывания. Но лучше для назначения нужных адресов подпрограммам прерываний воспользоваться директивой `org`.

Директива `org` предназначена для записи в счетчик адреса сегмента значения своего операнда. То есть при помощи этой директивы можно разместить команду (или данные) в памяти микроконтроллера по произвольному адресу. Пример использования директивы `org` для размещения подпрограмм обработки прерываний в нужных адресах показан в листинге 23.9.

Листинг 23.9. Пример использования директивы `org`

```
reset:
  ljmp main

;Перезагрузка таймера---------------------------------------
  ORG AT 0bh ;Вектор прерывания от таймера 0
IntT0:
  mov  TL0,  #LOW(-(F_ZQ/12)*10-2)     ;Настроить таймер
  mov  TH0,  #HIGH(-(F_ZQ/12)*10-2)    ;на период 10 мс
  reti

;Начало основной программы микроконтроллера ----------------------
  rseg _code
main:
  mov  SP,#VershSteka       ;Настроить указатель стека на вершину стека
  lcall init                ;Настроить микроконтроллер
;-----------------------------------------------------------------
```

Необходимо отметить, что при использовании этой директивы возможна ситуация, когда программист приказывает транслятору разместить новый код программы в уже занятом участке памяти, поэтому использование этой директивы допустимо только в крайних случаях. Обычно это размещение подпрограмм прерываний.

Директива `using`. При использовании прерываний критичным является время выполнения подпрограммы-обработчика прерываний. Его можно значительно сократить, выделив для обработки прерываний отдельный банк регистров при помощи директивы `using`. Номер используемого банка регистров указывается в директиве в качестве операнда. При этом реальное включение банка регистров производится записью необходимой константы в регистр PSW. Пример использования директивы `using` для подпрограммы обслуживания прерываний от таймера 0 приведен в листинге 23.10.

Листинг 23.10. Пример использования директивы `using`

```
  USING 2          ;Использовать второй банк регистров
IntT0:
  push PSW         ;Сохранить содержимое слова состояния микроконтроллера
  push ACC         ;Сохранить содержимое аккумулятора
  mov  PSW,#00010000B                    ;Включить второй банк регистров

  mov  TL0,  #LOW(-(F_ZQ/12)*10-2)       ;Настроить таймер
  mov  TH0,  #HIGH(-(F_ZQ/12)*10-2)      ;на период 10 мс
  pop ACC          ;Восстановить содержимое аккумулятора
  pop PSW          ;Восстановить содержимое слова состояния микроконтроллера
  reti
```

Остальные директивы ассемблера предназначены для управления сегментами памяти и будут рассмотрены позднее.

Управляющие команды

Кроме директив, для управления процессом трансляции используются команды языка программирования. С их помощью можно управлять работой компилятора ASM-51. Команды могут задаваться как параметры в DOS-строке вызова компилятора, или как управляющие строки в исходном тексте файла.

Если знак доллара (`$`) стоит в самом крайнем левом поле строки, то такая строка воспринимается компилятором как управляющая. Управляющие строки должны начинаться знаком доллара и могут содержать одну или более команд, разделенных пробелами.

Примеры управляющих строк:

```
$PRINT(A:\PROG.LST) OBJECT(PROG.OBJ)
$LIST DEBUG XREF
```

Так как целью книги является дать минимальные сведения, достаточные для того, чтобы начать работать с микроконтроллерами, то в данной книге будут рассмотрены только основные команды языка программирования ASM-51.

Команда `include`. Это, пожалуй, наиболее часто используемая команда языка программирования ASM-51. Она позволяет включать в состав программы участки текста из другого файла. Это удобно при многофайловом написании программы, например, для того, чтобы вынести описания внутренних регистров микроконтроллера в отдельный файл. Пример использования команды `include` для включения файла описания внутренних регистров микроконтроллера 89c51 выглядит следующим образом:

```
$INCLUDE (REG51.INC)
```

При использовании этой команды все содержимое включаемого файла помещается в выходной листинг программы, в результате чего становится трудно читать этот листинг. Поэтому в состав команд языка программирования ASM-51 включены команды `list/nolist`.

Команды `list/nolist` позволяют включать и выключать листинг исходного текста соответственно. При активной команде `nolist` в файл листинга будут помещаться только сообщения об ошибках. Пример запрета размещения содержимого включаемого файла в листинге программы будет выглядеть следующим образом:

```
$NOLIST                    //Запретить создание листинга включаемого файла
$INCLUDE (REG51.INC)
$LIST                      //Разрешить создание листинга дальнейшего текста
```

Команда `debug/nodebug` позволяет помещать в объектный модуль отладочную информацию (имена и местоположение переменных, меток и операторов) или запрещать размещение отладочной информации в объектном модуле.

Команда `pagelength (n)` определяет максимальное число строк на странице файла листинга. Это число включает заголовок страницы. Количество строк в странице может изменяться в пределах от 10 до 65 535.

Команда `pagewidth (n)` определяет максимальное число символов в строке листинга. Количество символов в строке листинга может изменяться от 72 до 132.

Реализация подпрограмм на языке ASM-51

Подпрограммы на языке программирования ASM-51 выносятся отдельно от основного текста программы. Обычно при программировании на языке ассемблера подпрограммы размещают после основного текста программы для того, чтобы случайно не передать управление подпрограмме не командой ее

вызова, а последовательным выполнением операторов основной программы. Такая ситуация может произойти из-за того, что ассемблер назначает адреса машинным командам в порядке их написания. Если в начале поместить исходный текст подпрограммы, то именно она будет размещена по нулевому адресу памяти программ и после сброса будет выполнена раньше, чем основная программа. При завершении подпрограммы команда `ret` передаст управление по неопределенному адресу памяти программ, что может привести к непредсказуемым последствиям. Если в начале поместить текст основной программы, то после сброса начнется ее выполнение. После инициализаций (включая установку начального значения указателя стека) основная программа всегда содержит бесконечный цикл. Это означает, что попасть в подпрограмму в результате последовательного выполнения операторов невозможно. Управление в нее может быть передано только с помощью команды вызова подпрограммы `lcall`.

Исходный текст подпрограммы начинается с метки, которая одновременно является именем подпрограммы. Именно это имя указывается в качестве операнда в команде вызова подпрограммы `lcall`. Возвращение из подпрограммы к команде, следующей за вызовом подпрограммы, осуществляется оператором `ret`. Все команды, которые должны быть выполнены в подпрограмме, располагаются между меткой, обозначающей имя подпрограммы, и оператором возврата из подпрограммы. В *главе 21* говорилось, что подпрограммы бывают двух видов: подпрограммы-функции и подпрограммы-процедуры. На языке программирования ассемблер проще реализуются подпрограммы-процедуры, поэтому сначала рассмотрим их.

Реализация подпрограмм-процедур на языке ASM-51

Подпрограмма-процедура вызывается командами процессора `lcall` и `acall`. В языке программирования ASM-51 допустимо использование директивы `call`. Выполняя ее, компилятор автоматически подбирает наиболее подходящую к данному случаю по размеру команду. В листинге 23.11 приведен пример использования подпрограммы-процедуры для управления последовательным портом.

Листинг 23.11. Пример подпрограммы-процедуры

```
...
MOV G_Per,#56      ;Передать число 56 через
CALL PeredatByte   ;последовательный порт
...
```

```
  MOV G_Per,#37      ;Передать число 37 через
  CALL PeredatByte   ;последовательный порт
  ...

;*******************************************
;Подпрограмма передачи байта
;через последовательный порт
;*******************************************
PeredatByte:
  JB TI, $           ;Если предыдущий байт передан,
  MOV SBUF, G_Per    ;то передать очередной байт
  RET
```

Передача переменных-параметров в подпрограмму

В приведенном в листинге 23.11 примере байт передается в подпрограмму через глобальную переменную G_Per. Однако программа будет эффективнее при использовании подпрограммы с параметрами. Мы уже знаем, что параметр подпрограммы — это локальная переменная, а при использовании локальных переменных могут значительно снизиться требования к памяти данных, т. к. локальные переменные разных подпрограмм располагаются в одних и тех же ячейках памяти микроконтроллера. Для размещения локальных переменных лучше всего использовать внутренние регистры процессора, т. к. кроме экономии памяти данных, использование регистровых переменных приводит к сокращению длины машинных команд, а значит, и длины всей программы в целом. Кроме того, использование подпрограмм с параметрами позволяет вызывать подпрограмму саму из себя, например, при реализации рекурсивных алгоритмов. На языке ASM-51 для передачи параметра размерностью один байт обычно используется аккумулятор, как показано в примере программы, приведенном в листинге 23.12.

Листинг 23.12. Пример подпрограммы-процедуры с передачей параметра через аккумулятор

```
  ...
  MOV A,#56          ;Передать число 56 через
  CALL PeredatByte   ;последовательный порт
  ...
  MOV A,#37          ;Передать число 37 через
  CALL PeredatByte   ;последовательный порт
  ...
```

```
;*******************************************
;Подпрограмма передачи байта
;через последовательный порт
;*******************************************
PeredatByte:
  JB TI, $            ;Если предыдущий байт передан,
  MOV SBUF, A         ;то передать очередной байт
  RET
```

Часто пример использования подпрограммы с параметрами более понятно выглядит на языке высокого уровня. Вызов подпрограммы с одним параметром, приведенный в листинге 23.12, на языке программирования С выглядел бы следующим образом:

```
PeredatByte(56);     //Передать число 56
PeredatByte(37);     //Передать число 37
```

Часто в подпрограмме требуется обрабатывать большие объемы данных, такие как массивы или структуры. При обращении к массивам или структурам, расположенным во внутренней памяти данных в качестве указателя адреса, обычно используются регистры R0 или R1. Пример передачи в подпрограмму массива в качестве параметра, написанный на языке программирования ASM-51, приведен в листинге 23.13.

Листинг 23.13. Пример подпрограммы-процедуры с передачей адреса массива через регистр R0

```
...
MOV R0,#Massiv          ;Задать адрес обрабатываемого массива
CALL ObrabotatMassiv ;Вызвать подпрограмму обработки массива
...
```

Если требуется, чтобы подпрограмма обработала такого же вида данные, но расположенные во внешней памяти данных или в памяти программ, то начальный адрес этих данных можно передать через двухбайтовый параметр-указатель. В качестве такого указателя обычно используется регистр-указатель данных DPTR. Пример передачи в подпрограмму адреса начального элемента строки в качестве параметра, написанный на языке программирования ASM-51, приведен в листинге 23.14. Строка расположена в памяти программ. Ее размещение в памяти программ показано в последней строке этого же листинга.

Листинг 23.14. Пример вызова подпрограммы-процедуры с передачей адреса строки через регистр DPTR

```
  ...
  MOV DPTR,#Stroka    ;Задать адрес первого байта строки
  CALL PeredatStroky ;Вызвать подпрограмму передачи строки
  ...
Stroka:  db 'Напечатать строку'
```

Если в подпрограмму нужно передать в качестве параметра двухбайтовое число, то для этого используется пара регистров (обычно это регистры R6 — старший байт и R7 — младший байт). Пример вызова подпрограммы с передачей в нее двухбайтового параметра, написанный на языке программирования ASM-51, приведен в листинге 23.15.

Листинг 23.15. Пример вызова подпрограммы-процедуры с передачей в нее двухбайтового числа (параметра) через регистры R7 и R6

```
  ...
;Пример передачи в подпрограмму двухбайтового числа
  MOV R7,#56      ;Передать младший байт
  MOV R6,#0       ;Передать старший байт
  CALL Podprogr   ;Вызвать подпрограмму
  ...
```

Если в подпрограмму нужно передать четырехбайтовое значение (параметр типа `long`, `unsigned long` или `float`), то обычно используются регистры R4–R7 (регистр R4 — старший байт). Пример вызова подпрограммы с передачей в нее четырехбайтового параметра, написанный на языке программирования ASM-51, приведен в листинге 23.16.

Листинг 23.16. Пример вызова подпрограммы-процедуры с передачей в нее четырехбайтового числа (параметра) через регистры R4–R6

```
  ...
;Пример передачи в подпрограмму четырехбайтового числа
  MOV R7,#56      ;Передать младший байт
  MOV R6,#0
  MOV R5,#0
  MOV R4,#0       ;Передать старший байт
  CALL Podprogr   ;Вызвать подпрограмму
  ...
```

В примере показана передача в подпрограмму константы, но точно так же можно передавать и переменную, расположенную во внутренней или внеш-

ней памяти данных. Для этого достаточно просто скопировать переменную в регистры R4–R7.

Реализация подпрограмм-функций на языке ASM-51

Часто требуется передавать результат вычислений из подпрограммы в основную программу. Для этого обычно используется подпрограмма-функция. Наиболее наглядным примером использования подпрограмм-функций является вычисление элементарных функций. Подпрограмма-функция для вычисления синуса в программе на языке высокого уровня вызывается следующим образом:

```
Y=sin(x);    //Вызов подпрограммы-функции
```

Как видно из приведенного примера, использование подпрограмм-функций значительно увеличивает наглядность программ и приближает запись на языке программирования к общепринятой математической форме. На языке программирования ASM-51 этот же вызов подпрограммы-функции выглядит следующим образом:

```
mov A,x    ;Передать в подпрограмму вычисления синуса параметр x
call sin   ;Вызвать подпрограмму вычисления синуса
mov Y,A    ;Запомнить значение, которое вернула подпрограмма в Y
```

В этом примере подпрограмма вычисления синуса перед выполнением оператора возврата в основную программу должна поместить результат вычисления синуса в аккумулятор. Переменные x и Y должны быть объявлены в начале программы при помощи директивы equ, как это показывалось ранее. Как видно из приведенного примера, вызов подпрограммы-функции на языке ассемблера менее нагляден по сравнению с языком программирования высокого уровня, но использование подпрограмм-функций позволяет значительно сокращать требования к внутренней памяти микроконтроллера благодаря максимальному использованию внутренних регистров микроконтроллера.

Элементарные операции на языке программирования ассемблер чаще всего вычисляются табличным способом, подобным тому, как проводилась перекодировка чисел, приведенная в примере листинга 23.5. Сами значения функций при этом рассчитываются заранее с использованием компьютера и вводятся во внутреннюю память микроконтроллера при помощи директивы db.

Подпрограмма-функция может возвращать и многобайтовые переменные. Для этого можно использовать два или четыре регистра. Обычно используются регистровая пара R6, R7 или регистры R4–R7. Можно также рассчитать и записать во внутренней памяти полученную в результате вычислений в под-

программе переменную-массив или структуру и передать в вызывающую программу адрес этой переменной при помощи регистра-указателя R0 или R1.

Кроме подпрограмм-процедур и подпрограмм-функций существует особый класс подпрограмм. Это подпрограммы обработки аппаратных прерываний.

Реализация подпрограмм обработки прерываний на языке ASM-51

Подпрограммы обработки прерываний вызываются аппаратурой в произвольный момент времени и не могут иметь параметров. Подпрограммы обработки прерываний не могут быть подпрограммами-функциями. При переходе на подпрограмму обслуживания прерывания автоматически запрещается возникновение последующих прерываний, поэтому при возвращении из подпрограммы обработки прерывания должны быть разрешены прерывания. Команда возвращения из подпрограммы ret не снимает запрет на обработку прерываний, поэтому возврат из подпрограммы обработки прерывания может быть осуществлен только специальной командой возврата из подпрограммы обслуживания прерывания reti. Пример подпрограммы обработки прерывания на языке программирования ASM-51 приведен в листинге 23.17.

Листинг 23.17. Пример подпрограммы обработки прерывания

```
;Старт программы----------------------------------------------
  ORG 0     ;Вектор рестарта процессора
Reset: jmp main

  ORG 0Bh   ;Вектор прерывания таймера 0
  jmp IntT0
  ORG 23h   ;Вектор прерывания последовательного порта
  jmp PoslPort

IntT0: ;Перезагрузка таймера---------------------------------
  mov TL0,#LOW(15000)   ;Настроить таймер
  mov TH0,#HIGH(15000)  ;на период 15 мс
  reti
;Начало основной программы------------------------------------
main:
  ...
```

Достаточно часто требуется обработка прерываний от нескольких источников. В результате подпрограмма обработки прерываний не может уместиться между векторами прерываний на участке памяти длиной 8 байтов, поэтому

подпрограммы выносятся из области векторов прерывания. Для перехода на эти подпрограммы используются команды безусловного перехода. В листинге 23.17 директива `ORG 0BH` использована для того, чтобы поместить команду перехода на подпрограмму обработки прерывания точно на вектор прерывания таймера 0.

Сигнал прерывания, а значит, и вызов подпрограммы обработки прерывания может произойти в произвольный момент времени, т. е. в любом месте выполнения основной программы. Чтобы не повлиять на выполнение основной программы, подпрограмма обработки прерываний не должна изменять содержимое регистров, ведь в них могут быть записаны данные, используемые в основной программе. Поэтому все регистры, которые используются подпрограммой обработки прерываний, должны быть сохранены в стеке, а затем восстановлены из него.

Если подпрограмма обработки прерывания использует несколько регистров, то на сохранение регистров в стеке и на восстановление их из стека тратится достаточно много времени. Микроконтроллеры семейства MCS-51 предлагают возможность использовать для подпрограмм прерываний отдельный банк регистров. Переключение банков регистров производится при помощи регистра PSW. В языке программирования ASM-51 то, что программа использует ненулевой банк регистров, указывается при помощи директивы `using`. Переключение банков регистров в подпрограмме обработки прерывания от таймера T0, а также резервирование первого банка регистров при помощи директивы `using` показано в листинге 23.18.

Листинг 23.18. Пример подпрограммы обработки прерывания с использованием первого банка регистров

```
;Старт программы-----------------------------------------------------
  ORG 0      ;Вектор рестарта процессора
Reset: jmp main

  ORG 0Bh    ;Вектор прерывания таймера 0
  jmp IntT0
  ORG 23h    ;Вектор прерывания последовательного порта
  jmp PoslPort

  USING 1
PoslPort: ;Подпрограмма обработки прерывания от посл. порта------------
  push PSW                ;Сохранить регистр слова состояния в стеке
  mov  PSW,#00001000b     ;Включить первый банк регистров

  mov  @R0,SBUF           ;Записать принятое значение в буфер приемника
  inc  R0                 ;Перейти к следующей ячейке буфера приемника
```

```
  cjne R0,#KonBuf,EndProv;Если достигнут конец буфера приемника,
    mov  R0,#NachBuf        ;то перейти к первой ячейке
EndProv:
  pop  PSW                ;Восстановить регистр слова состояния из стека
  reti
```

Структурное программирование на языке ASM-51

Применение структурного программирования позволяет увеличить скорость написания программ и облегчить их отладку. Языки программирования C, PASCAL, PL/M разрабатывались на основе принципов структурного программирования, поэтому в состав этих языков входят структурные операторы. Ассемблер не относится к структурированным языкам программирования. Тем не менее, структурное программирование возможно и на языках программирования низкого уровня, в том числе и на языке программирования ASM-51, где не предусмотрено структурных операторов.

При разработке программы с использованием методов структурного программирования она может быть оттранслирована и выполнена на любом этапе написания, при этом можно отследить все алгоритмические действия программы, реализованные к этому времени. При использовании методов структурного программирования процесс написания программы не отличается от процесса создания алгоритма. Более того! Эти этапы создания программы можно объединить!

Для реализации методов структурного программирования огромное значение имеет использование "говорящих" меток, обозначаемых не просто `M0`, `M1` и т. д., а в названии которых отображается действие, выполняемое программой. Для людей, не владеющих иностранным языком, ограничение в использовании для назначения меток букв только латинского алфавита создает определенные трудности. Однако, используя транслитерацию, и латинскими буквами можно писать русские слова. При этом для обозначения действия может потребоваться несколько слов, использование же пробелов внутри метки недопустимо! Выйти из такой ситуации можно двумя способами:

- применять специальные символы-разделители;
- начинать каждое новое слово внутри метки с буквы верхнего регистра.

В качестве разделителей внутри метки можно использовать символы подчеркивания '_' и вопроса '?'. Примеры назначения "говорящих" меток:

```
Priem_Comandy           ;Использование символов-разделителей
ProveritBitGotovnosti   ;Использование букв верхнего регистра
```

Надо сказать, что в языке программирования ассемблер роль метки исключительно важна. Метка используется для обозначения переменных и констант, а также имен подпрограмм и программных модулей.

Одна из основных идей структурного программирования заключаются в том, чтобы использовать только четыре структурных конструкции управления. При помощи этих структурных конструкций управления можно построить сколь угодно сложную программу.

Наиболее распространенная структурная конструкция управления — линейная цепочка операторов. Любая задача может быть разбита на несколько более простых подзадач. Выполнение подзадач лучше оформить как вызов подпрограмм, в названии которых можно (и нужно) отразить подзадачу, которую решает эта подпрограмма. Например:

```
ProchitatPort       ;Прочитать порт
VklychitIndikator   ;Включить индикатор
```

При этом с точки зрения структурного программирования использовать подпрограмму имеет смысл даже в том случае, когда действие будет выполняться только один раз! Выполняемое алгоритмическое действие отображается в названии подпрограммы, поэтому программу можно читать по названиям подпрограмм. Человеческий глаз может охватить большую часть алгоритма, а значит, программа будет более понятна, что приведет к более быстрому завершению ее отладки. Программы, понятные при чтении их исходного текста, часто называют самодокументирующимися.

На момент написания алгоритма (и программы) верхнего уровня нас не интересует, как будут решаться задачи, зато очень интересуют взаимосвязи между ними. Поэтому первоначально вместо законченных подпрограмм можно (и нужно) использовать подпрограммы-заглушки. Использование подпрограмм-заглушек позволяет отработать взаимодействие между задачами, убедиться, что все они выполняются в нужной последовательности и именно тогда, когда возникает необходимость в решении данной конкретной задачи.

Для взаимодействия между задачами обычно используются глобальные переменные. Эти переменные и вводятся на верхнем уровне программы. Если переменные отвечают за переключение между задачами, то изменение переменных производят вручную в отладчике программ. Затем при пошаговой отладке проверяют, вызывается ли подпрограмма, отвечающая за выполнение поставленной задачи, и не вызываются ли при этом лишние подпрограммы.

После завершения отладки верхнего уровня программы приступают к написанию и отладке каждой из подпрограмм-заглушек, т. е. к превращению подпрограмм-заглушек в законченные подпрограммы. При этом т. к. программа верхнего уровня уже отлажена, то решается только задача, выполняемая отлаживаемой в данный момент подпрограммой. Особенно тщательно отслеживается взаимодействие с программой верхнего уровня для того, чтобы не

нарушить логику ее работы. Одновременно отрабатывается взаимодействие с подпрограммами более нижнего, по сравнению с отлаживаемым, уровня.

Пример реализации линейной цепочки операторов на языке программирования ASM-51 приведен в листинге 23.19.

Листинг 23.19. Пример использования подпрограмм для структурирования программы, написанной на языке программирования ASM-51

```
;*****************************************************************
;Главная программа
;*****************************************************************
  ...
;----- Линейная цепочка операторов -------------------------------
  Call Deistvie1  ;Алгоритмическое действие 1
  Call Deistvie2  ;Алгоритмическое действие 2
;-----------------------------------------------------------------
  ...
;*****************************************************************
;Определения подпрограмм
;*****************************************************************
;--- Подпрограмма-заглушка 1 -------------------------------------
Deistvie1: ret
;--- Подпрограмма-заглушка 2 -------------------------------------
Deistvie2: ret
```

Вторая структурная конструкция управления — условное выполнение оператора. Как уже рассматривалось в предыдущей главе, эта конструкция может быть двух видов — с одной ветвью и с двумя.

Если реализуется конструкция условного выполнения оператора только с одной ветвью, то можно воспользоваться любой командой условного перехода, входящей в набор команд микроконтроллера. Соответствующий пример приведен в листинге 23.20. В приведенном примере переменная `SV1` и константа `NajKnZvezd` должны быть объявлены ранее (например, при помощи директивы `equ`).

Листинг 23.20. Пример использования команд условного перехода для реализации конструкции условного выполнения оператора с одной ветвью

```
;-----------------------------------------------------------------
;Оператор обработки нажатой кнопки
;-----------------------------------------------------------------
  cjne A,#NajKnZvezd,NeGasitSvDiod  ;Если нажата кнопка '*',
    clr SV1                         ;то погасить светодиод SV1
NeGasitSvDiod:
```

Полная конструкция условного выполнения операторов реализуется на языке ассемблера более сложным образом. Для этого потребуется уже две команды перехода. Для проверки результата логического выражения используется команда условного перехода. Чтобы исключить выполнение второй ветви, потребуется команда безусловного перехода. Пример реализации полной конструкции условного выполнения операторов приведен в листинге 23.21.

Листинг 23.21. Пример реализации конструкции условного выполнения операторов на языке программирования ASM-51

```
;*************************************************
;Главная программа
;*************************************************
  ...
;----------Условный оператор--------------------
  jb PrinjatByte,ElseUsl;Условная операция
    Call Plecho1          ;Реализация
    jmp EndUsl            ;ветви 1
ElseUsl:
    Call Plecho2          ;Реализация ветви 2
EndUsl:;-----------------------------------------
  ...

;*************************************************
;Определения подпрограмм
;*************************************************
;--Пока это только подпрограмма-заглушка!------
Plecho1:;Метка помечает следующий оператор
  ret

;--Пока это только подпрограмма-заглушка!------
Plecho2:;Метка помечает следующий оператор
  ret
```

Третья структурная конструкция — это *цикл с проверкой условия после тела цикла*. Такая конструкция легко реализуется на ассемблере при помощи одной команды условного или безусловного перехода. Отличие этой конструкции от условного выполнения операторов заключается в том, что передача управления осуществляется не вперед, а назад. Однако в системе команд микроконтроллера MCS-51 для реализации цикла предусмотрена специальная команда, выполняющая сразу два алгоритмических действия, необходимых для реализации цикла, — DJNZ. Пример использования этой команды для реализации цикла с проверкой условия после тела цикла приведен в листинге 23.22.

Листинг 23.22. Пример реализации цикла с проверкой условия после тела цикла на языке программирования ASM-51

```
;Программа обнуления внутренней памяти микроконтроллера
  mov R0,#128          ;Обнулить 128 ячеек
  mov A,#0             ;внутренней памяти
Obnulenie:
    mov @R0,A            ;Обнулить очередную ячейку,
    DJNZ R0, Obnulenie ;и если все ячейки обнулены,
                       ;то выйти из цикла
```

Четвертая структурная конструкция управления — это *цикл с проверкой условия до тела цикла*. В отличие от предыдущего варианта цикла, тело цикла в данном случае может ни разу не выполниться, если условие цикла сразу же выполнено. Этот цикл, как и конструкцию условного выполнения операторов, невозможно реализовать при помощи одной машинной команды. Дополнительно потребуется команда безусловного перехода. Пример реализации цикла с проверкой условия до тела цикла приведен в листинге 23.23.

Листинг 23.23. Пример оператора цикла с проверкой условия до тела цикла на языке программирования ASM-51

```
;****************************************
;Главная программа
;****************************************
  ...
;-------- Оператор цикла ----------------
Nachalo:
  jb KnNaj,KonCykla;Условная операция
    Call TeloCykla   ;Реализация обработки
    sjmp Nachalo     ;подпрограмм ожидания
KonCykla:
;----------------------------------------
  ...

;****************************************
;Определения подпрограмм
;****************************************
;--- Подпрограмма-заглушка --------------
TeloCykla:
  ret  ;Оператор помечен предыдущей меткой
```

Многомодульные программы

Как это обсуждалось в *главе 21*, разбиение исходного текста программы на отдельные модули делает его более понятным для программиста или нескольких программистов, участвующих в создании программного продукта. Язык программирования ASM-51 позволяет писать многомодульные программы. Однако каждый модуль программы должен быть оформлен соответствующим образом.

Обычно все переменные и константы (хранящиеся в памяти программ), использующиеся в программном проекте, оформляются в отдельном модуле. В отдельный модуль выносятся подпрограммы, отвечающие за работу с каким-либо внешним или внутренним устройством микроконтроллерной системы. Все эти элементы программы должны быть доступны из основной программы или других модулей. Для того чтобы транслятор записал в объектный модуль информацию, необходимую для объединения модулей в единую программу, нужно применять специальные директивы ссылок на метки, используемые для обозначения переменных и подпрограмм.

В языке программирования ASM-51 для этой цели используется директива `public` (общедоступные). Директива может быть использована в любом месте исходного текста модуля, однако обычно она размещается в его начале. Имена переменных в этой директиве перечисляются через запятую. Если в результате получается слишком длинная строка, то можно использовать несколько таких директив. Глобальное имя может быть объявлено только в одном модуле программы. Несколько глобальных переменных или подпрограмм с одним и тем же именем недопустимы. Пример использования директивы `public` на языке программирования ASM-51:

```
PUBLIC BufInd, Parametr
PUBLIC Podprogr, ?Podprogr?Byte
```

Для ссылки на переменную или метку, объявленную в другом модуле, используется директива `extrn`. Идентификатор в пределах одного модуля не может быть одновременно объявлен как `public` и как `extrn`. В директиве `extrn` перечисляются через запятую имена подпрограмм и переменных, числовое значение которых редактор связей должен получить из других модулей и модифицировать все команды, в которых эти метки или переменные используются. Кроме того, для правильного использования инструкций микроконтроллера в директиве `extrn` должен быть указан тип памяти (`data`, `bit`, `code`, `idata` или `xdata`), в которой расположена переменная или метка. Для передачи числовых констант между модулями можно воспользоваться вспомогательным словом `number`. Директива `extrn` может располагаться в любом месте исходного текста модуля. Идентификатор внешнего имени не может быть

переопределен в программном модуле каким-либо способом. Рекомендуется с целью оптимизации результирующего объектного кода (в частности, команд JMP и CALL) размещать директиву EXTRN до ссылки на соответствующее внешнее имя (как правило, в начале исходного текста программного модуля или программного сегмента). Пример использования директивы extrn на языке программирования ASM-51:

```
EXTRN DATA (BufInd, ERR), CODE (ASC_BIN, BIN_ASC), NUMBER (LIMIT)
EXTRN CODE (Podprogr)
```

Объявления переменных и имен подпрограмм внешних модулей загромождают исходный текст модуля. Кроме того, при использовании чужих модулей трудно объявить переменные и подпрограммы без ошибок. Поэтому, как это обсуждалось в предыдущей главе, объявления имен хранят во включаемых файлах, которые называются файлами-заголовками. Файл-заголовок на языке программирования asm-51 записывается на диск с расширением *.inc (сокращение от англ. слова *include* — включать) и включается в исходный текст программы при помощи директивы include. Например:

```
$INCLUDE (REG51.INC)
```

При работе с пакетом программ, поставляемым фирмой keil, программа-транслятор с языка программирования ASM-51 "понимает" и файлы-заголовки, написанные для языка программирования C-51. Поэтому можно воспользоваться и этими файлами. Например:

```
#include <at87f51rc.h>
$include (init.inc)
```

Теперь рассмотрим команды редактора связей, которые позволят объединить несколько объектных модулей в один. В простейшем случае для объединения модулей можно использовать имя программы-редактора связей с необходимыми ключами. Для объединения нескольких модулей в исполняемую программу имена всех модулей передаются в редактор связей r151.exe в качестве параметров при запуске этой программы. Приведем пример вызова редактора связей из командной строки DOS для объединения трех модулей:

```
r151.exe progr.obj, modul1.obj, modul2.obj
```

В результате работы редактора связей в этом примере будет создан исполняемый модуль с именем progr. Формат записи информации в этом файле остается прежним — объектным. Это позволяет объединять модули по частям, т. е. при желании можно из нескольких мелких модулей получить один более крупный.

В настоящее время часто пользуются интегрированными средами программирования, например, фирм Franclin или keil, в состав которых входят язык программирования ASM-51. В этих программах создание строки вызова ре-

дактора связей производится автоматически при настройке программного проекта, а вызов редактора связей при помощи этой строки производится в ходе построения программного проекта. Настройка программного проекта происходит при подключении к нему новых программных модулей и при изменении его свойств, таких как разрешение или запрет создания карты памяти программы, выбор папки для хранения выходных файлов, разрешение или запрет помещения в выходной файл отладочной информации, разрешение или запрет создания загрузочного HEX-файла.

Использование сегментов в языке программирования ассемблер

При трансляции программы по частям возникает вопрос, как с этими частями работать. Иначе говоря, встает вопрос соединения сегментов. Справедливости ради необходимо отметить, что даже когда мы не задумываемся о сегментах, в программе присутствует два сегмента: сегмент кода и сегмент данных. Если внимательно присмотреться к программе, то можно обнаружить, что кроме инструкций микроконтроллера в памяти программ хранятся константы, т. е. в памяти программ микроконтроллера располагаются, по крайней мере, два сегмента: код и данные. Чередование кода и данных может привести к нежелательным последствиям. Вследствие каких-либо причин данные могут быть случайно выполнены в качестве машинных команд или, наоборот, коды машинных команд могут быть восприняты и обработаны как данные.

Перечисленные выше причины приводят к тому, что в программе желательно явным образом выделить, по крайней мере, четыре сегмента:

- кода;
- переменных;
- стека;
- констант.

И лучше, если эти сегменты будут перемещаемыми. Тогда редактор связей сможет автоматически скомпоновать программу наилучшим способом.

Пример размещения сегментов в адресном пространстве памяти программ и внутренней памяти данных приведен на рис. 23.3. На этом рисунке видно, что при использовании нескольких сегментов переменных во внутренней памяти данных редактор связей может разместить меньший из них на месте неиспользованных банков регистров. Под сегмент стека обычно отводится вся область внутренней памяти, не занятая переменными. Это позволяет создавать программы с максимальным уровнем вложенности подпрограмм. Сег-

мент переменных, расположенный на рис. 23.3 во внешней памяти данных, при использовании современных микросхем, таких как AduC842, может находиться в ОЗУ, расположенном на кристалле микроконтроллера.

Рис. 23.3. Разбиение памяти программ и памяти данных на сегменты

Наиболее простой способ определения сегментов — это использование абсолютных сегментов памяти. При этом способе распределение памяти ведется вручную точно так же, как это делалось при использовании директивы `EQU`. В этом случае начальный адрес сегмента жестко задается программистом, и он же следит за тем, чтобы сегменты не перекрывались друг с другом в памяти микроконтроллера. Использование абсолютных сегментов позволяет более гибко работать с памятью данных, т. к. теперь байтовые переменные в памяти данных могут быть назначены при помощи директивы резервирования памяти `DS`, а битовые переменные при помощи директивы резервирования битов `DBIT`.

Для определения абсолютных сегментов памяти используются следующие директивы:

- `BSEG` — абсолютный сегмент в области битовой адресации;
- `CSEG` — абсолютный сегмент в области памяти программ;
- `DSEG` — абсолютный сегмент в области внутренней памяти данных;
- `ISEG` — абсолютный сегмент в области внутренней памяти данных с косвенной адресацией;
- `XSEG` — абсолютный сегмент в области внешней памяти данных.

Директива BSEG позволяет определить абсолютный сегмент во внутренней памяти данных с битовой адресацией по определенному адресу. Эта директива не назначает имени сегменту, т. е. объединение сегментов из различных программных модулей невозможно. Для определения конкретного начального адреса сегмента применяется атрибут AT. Если атрибут AT не используется, то начальный адрес сегмента предполагается равным нулю. Использование битовых переменных позволяет значительно экономить внутреннюю память программ микроконтроллера. Пример использования директивы BSEG для объявления битовых переменных приведен в листинге 23.24.

Листинг 23.24. Пример использования директивы BSEG перед определениями битовых переменных

```
  BSEG AT 8      ;Сегмент начинается с восьмого бита
RejInd   DBIT 1 ;Флаг режима индикации
RejPriem DBIT 1 ;Флаг режима приема
Flag     DBIT 1 ;Флаг общего назначения
```

Директива CSEG позволяет определить абсолютный сегмент в памяти программ по определенному адресу. Директива не назначает имени сегменту, т. е. объединение сегментов из различных программных модулей невозможно. Для определения конкретного начального адреса сегмента применяется атрибут AT. Если атрибут AT не используется, то начальный адрес сегмента предполагается равным нулю. Пример использования директивы CSEG для размещения подпрограммы обслуживания прерывания от таймера 0 приведен в листинге 23.25.

Листинг 23.25. Пример использования директивы CSEG для размещения подпрограммы обслуживания прерывания

```
;Перезагрузка таймера---------------------------------------
  CSEG AT 0bh ;Вектор прерывания от таймера 0
IntT0:
  mov  TL0,  #LOW(-(F_ZQ/12)*10-2)     ;Настроить таймер
  mov  TH0,  #HIGH(-(F_ZQ/12)*10-2)    ;на период 10 мс
reti
```

Директива DSEG позволяет определить абсолютный сегмент во внутренней памяти данных по определенному адресу. Предполагается, что к этому сегменту будут обращаться команды с прямой адресацией. Эта директива не назначает имени сегменту, т. е. объединение сегментов из различных программных модулей невозможно. Для определения конкретного начального адреса сегмента применяется атрибут AT. Если атрибут AT не используется, то

начальный адрес сегмента предполагается равным нулю. Пример использования директивы DSEG для объявления байтовых переменных приведен в листинге 23.26.

Листинг 23.26. Пример использования директивы DSEG перед определением байтовых переменных

```
  DSEG AT 20 ;Разместить сегмент в битовом пространстве микроконтроллера
             ;(для возможности одновременно битовой и байтовой адресации)
RejInd DS 1  ;Переменная, отображающая состояние программ обслуживания
               ;аппаратуры
Rejim  DS 1  ;Переменная, отображающая режимы работы
Massiv DS 10 ;Десятибайтовый массив
```

Последний пример связан с примером, приведенным в листинге 23.24. То есть команды, изменяющие битовые переменные RejInd, RejPriem или Flag, одновременно будут изменять содержимое переменной Rejim, и наоборот, команды, работающие с переменной Rejim, одновременно изменяют содержимое флагов RejInd, RejPriem или Flag. Такое объявление переменных позволяет написать наиболее эффективную программу управления контроллером и подключенными к нему устройствами.

Директива ISEG позволяет определить абсолютный сегмент во внутренней памяти данных с косвенной адресацией по определенному адресу. Напомню, что адресное пространство внутренней памяти с косвенной адресацией в два раза больше адресного пространства памяти с прямой адресацией. Именно в этой области памяти размещается стек. Директива ISEG не назначает имени сегменту, т. е. объединение сегментов из различных программных модулей невозможно. Для определения конкретного начального адреса сегмента применяется атрибут AT. Если атрибут AT не используется, то начальный адрес сегмента предполагается равным нулю. Пример использования директивы ISEG для объявления байтовых переменных приведен в листинге 23.27.

Листинг 23.27. Пример использования директивы ISEG для объявления байтовых переменных

```
  ISEG AT 80  ;Разместить сегмент в диапазоне адресов, совмещенных с SFR

Bufer  DS 10  ;Десятибайтовый массив
Stack  DS 245 ;Стек
```

Директива XSEG позволяет определить абсолютный сегмент во внешней памяти данных по определенному адресу. Эта директива не назначает имени сегменту, т. е. объединение сегментов из различных программных модулей

невозможно. Для определения конкретного начального адреса сегмента применяется атрибут AT. Если атрибут AT не используется, то начальный адрес сегмента предполагается равным нулю. До недавнего времени использование внешней памяти не имело смысла, т. к. это значительно увеличивало габариты и цену устройства. Однако в последнее время ряд фирм стал размещать на кристалле значительные объемы ОЗУ, доступ к которому осуществляется как к внешней памяти. Так как эта директива применяется так же, как DSEG, то отдельный пример приводиться не будет.

Использование абсолютных сегментов позволяет облегчить работу программиста по распределению памяти микроконтроллера для различных переменных. Однако в большинстве случаев абсолютный адрес переменной нас совершенно не интересует. Исключение составляют только регистры специальных функций. Так зачем же вручную задавать начальный адрес сегментов?

Одна из ситуаций, когда нас не интересует начальный адрес сегмента, — это программные модули. Как уже говорилось ранее, в программные модули обычно выносятся подпрограммы. Естественно, что конкретные адреса, по которым будут находиться эти подпрограммы в адресном пространстве микроконтроллера, нас тоже мало интересуют.

Если абсолютные адреса переменных или участков программ не интересны, то можно воспользоваться перемещаемыми сегментами. Имя перемещаемого сегмента задается директивой `segment`.

Директива `segment` позволяет определить имя сегмента и область памяти, где будет размещаться данный сегмент памяти. Для каждой области памяти определено ключевое слово:

- `data` — размещает сегмент во внутренней памяти данных с прямой адресацией;
- `idata` — размещает сегмент во внутренней памяти данных с косвенной адресацией;
- `bit` — размещает сегмент во внутренней памяти данных с битовой адресацией;
- `xdata` — размещает сегмент во внешней памяти данных;
- `code` — размещает сегмент в памяти программ.

После определения имени сегмента можно использовать этот сегмент при помощи директивы `rseg`.

Директива `rseg` позволяет поместить в конкретный перемещаемый сегмент переменные или фрагмент кода программы. Обращение к одному и тому же сегменту может осуществляться в разных местах исходного текста программы (даже в разных файлах). При этом все участки сегмента будут находиться

в соседних участках области памяти микроконтроллера, выделенного для этого сегмента.

Использование сегмента зависит от области памяти, для которой он предназначен. Если это память данных, то в сегменте объявляются байтовые или битовые переменные. Если это память программ, то в сегменте размещаются константы или участки кода программы. Пример использования директив `segment` и `rseg` для определения байтовых переменных во внутренней памяти данных с косвенной адресацией приведен в листинге 23.28.

Листинг 23.28. Пример использования директив `segment` и `rseg` для определения байтовых переменных

```
_data segment idata
  public VershSteka

;Определение переменных----------------------------
rseg _data
buferKlav:  ds 8
VershSteka:
  End
```

В этом примере объявлен массив `buferKlav`, состоящий из восьми байтовых переменных. Кроме того, в данном примере объявлена переменная `VershSteka`, соответствующая последней ячейке памяти, используемой для хранения переменных. Переменная `VershSteka` может быть использована для начальной инициализации указателя стека для того, чтобы отвести под стек максимально доступное количество ячеек внутренней памяти. Это необходимо для того, чтобы избежать переполнения стека при вложенных вызовах подпрограмм.

Объявление и использование сегментов данных в области внутренней или внешней памяти данных не отличается от приведенного в последнем примере, за исключением ключевого слова, определяющего область памяти данных.

Еще один пример использования директив `segment` и `rseg` приведен в листинге 23.29. В этом примере директива `segment` используется для определения сегмента битовых переменных.

Листинг 23.29. Пример использования директив `segment` и `rseg` для определения битовых переменных

```
_bits segment bit
  public knIzm,strVv
```

```
;Определение битовых переменных ----------------------------------
  rseg _bits
knIzm: dbit 1
strVv: dbit 1
  end
```

Наибольший эффект от применения сегментов можно получить при написании основного текста программы с использованием модулей. Обычно каждый программный модуль оформляется в виде отдельного перемещаемого сегмента. Это позволяет редактору связей скомпоновать программу оптимальным образом. При использовании абсолютных сегментов памяти программ пришлось бы это делать вручную, а т. к. в процессе написания программы размер программных модулей постоянно меняется, то пришлось бы вводить защитные области неиспользуемой памяти между программными модулями.

Пример использования перемещаемых сегментов в исходном тексте программы содержится в листинге 23.30. В этом примере приведен начальный участок основной программы микроконтроллера, на который производится переход с нулевой ячейки памяти программ. Использование такой структуры программы позволяет в любой момент времени при необходимости использовать любой из векторов прерывания, доступный в конкретном микроконтроллере, для которого пишется эта программа. Достаточно разместить определение этого вектора с использованием директивы `cseg`.

В приведенном примере использовано имя перемещаемого сегмента `_code`. Оно было объявлено в самой первой строке исходного текста программы. Конкретное имя перемещаемого сегмента может быть любым, но, как уже говорилось ранее, оно должно отображать ту задачу, которую решает данный конкретный модуль.

Листинг 23.30. Пример использования директив `segment` и `rseg` в программном модуле

```
_code segment code

;Старт программы-------------------------------------------
  CSEG AT 0    ;Вектор рестарта процессора
reset:
  jmp main

;Начало основной программы микроконтроллера ----------------------
  rseg _code
```

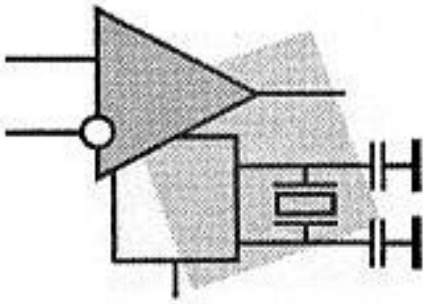
```
main:
  MOVX @DPTR,A
  mov  SP,#VershSteka       ;Настроить указатель стека на вершину стека
  call init                 ;Настроить микроконтроллер
;-------------------------------------------------------------------
```

Итоги

В данной главе рассмотрены основные средства языка программирования ASM-51, достаточные для написания довольно сложных программ, однако в процессе работы может потребоваться дополнительная информация, которую можно получить в описании языка программирования, поставляемом вместе с самой программой-транслятором.

Язык программирования ассемблер позволяет разрабатывать самые компактные и эффективные программы, но процесс их создания на этом языке трудоемкий, а это значит, что он занимает достаточно длительное время. Однако в настоящее время предлагаются, причем по приемлемой цене, микросхемы с внутренними ресурсами, достаточными для размещения и выполнения программ, написанных на языке высокого уровня. Поэтому программы сейчас все чаще создаются на языке С как наиболее распространенном для микроконтроллеров. В последующих главах мы рассмотрим пример создания программы на языке программирования С-51.

ГЛАВА 24



Работа с интегрированной средой программирования

При выборе системы программирования приходится решать целый ряд задач. Прежде всего, приходится решать, платный или бесплатный продукт следует применять при разработке своей программы. Если требуется создать простую программу, то можно применить бесплатную версию компилятора для данного семейства микроконтроллеров, однако опыт показывает, что простые программы имеют тенденцию к разрастанию и усложнению выполняемых задач. В результате может возникнуть ситуация, когда бесплатное программное обеспечение перестанет удовлетворять разработчика программы, но переход на новую среду программирования может оказаться слишком дорог (дороже первоначальной стоимости среды программирования). Поэтому при выборе конкретного компилятора и среды программирования следует исходить из тех соображений, что стоимость разработки в основном определяется рабочим временем, затрачиваемым на создание и сопровождение программного комплекса микроконтроллерного устройства.

В настоящее время программы пишутся на одном из языков программирования в виде текстовых файлов. Это означает, что для написания программы можно воспользоваться любым текстовым редактором. Для того чтобы программа-транслятор могла преобразовать исходный текст программы в машинные коды микропроцессора, этот текст должен быть записан с использованием символов ASCII или ANSI таблиц. К сожалению, некоторые текстовые редакторы для увеличения возможностей редактирования и отображения текста используют для его записи формат RTF или свои собственные форматы (например, текстовый редактор WORD). Такие текстовые файлы не распознаются программами-трансляторами и, следовательно, не могут быть использованы для записи исходного текста программы.

Для облегчения процесса разработки программы часто используются интегрированные среды программирования. В состав интегрированной среды про-

граммирования включается определенный набор программных средств: редактор исходного текста, трансляторы с выбранного языка программирования, редакторы связей, загрузчики, отладчики и т. д.

Редактор текстов обычно является первой программой, которую приходится использовать в процессе разработки программ: благодаря ему программист получает возможность набирать исходные тексты программ, написанных на ассемблере или на одном из языков высокого уровня, и сохранять их на жестком диске.

Лучшей системой программирования для микроконтроллеров семейства MCS-51, на мой взгляд, является продукт фирмы keil. Оценочную версию этого программного продукта можно скачать с сайта этой фирмы **http://www.keil.com**. Интегрированная среда программирования этого программного пакета — µVision2 в значительной степени напоминает широко известную среду программирования Visual C. Именно эту среду программирования мы и рассмотрим в данной главе.

Работа с текстовым редактором интегрированной среды программирования keil-C

Работа с текстовыми файлами начинается с создания нового файла. Создать текстовый файл можно несколькими способами. Первый способ — воспользоваться главным меню, как показано на рис. 24.1.

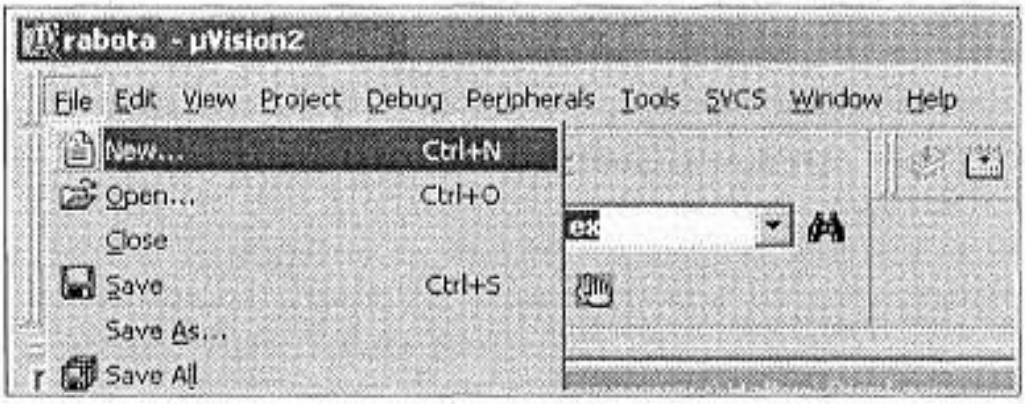


Рис. 24.1. Создание нового файла через главное меню

Второй способ — использовать быстрые клавиши <Ctrl>+<N>. И третий способ — это нажать на пиктограмму создания нового файла, как показано на рис. 24.2.

После выполнения этих действий открывается окно текстового редактора, в котором можно вводить исходный текст программы. Внешний вид про-

граммы с открытым окном текстового редактора показан на рис. 24.3. Ввод программы производится с клавиатуры. Стирание одиночных ошибочно введенных символов возможно при помощи клавиш <Delete> и <Backspace>.

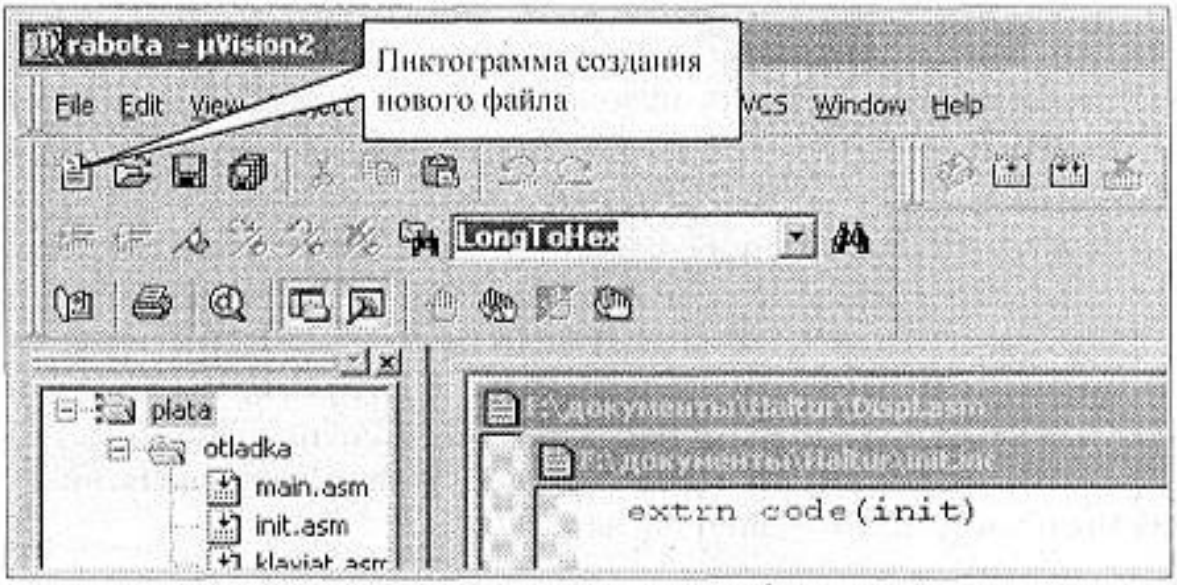


Рис. 24.2. Создание нового файла при помощи пиктограммы

Набрав исходный текст программы в окне текстового редактора, файл необходимо сохранить на диске компьютера несколькими способами. Первый способ — воспользоваться меню **File** (файл), как показано на рис. 24.3.

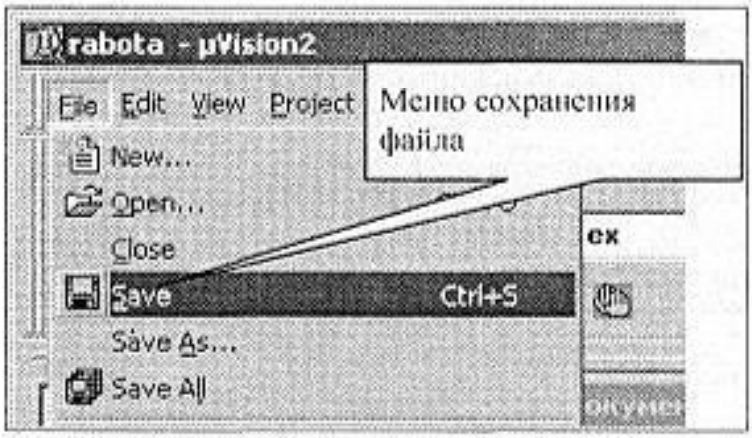


Рис. 24.3. Сохранение файла через главное меню

Второй способ — использовать быстрые клавиши <Ctrl>+<S>. И третий способ — это нажать на пиктограмму сохранения файла, как показано на рис. 24.4.

При наборе исходного текста программы часто требуется копировать участки программ из одного файла в другой. Для этого в текстовом редакторе открываются оба файла. Затем необходимый участок текста выделяется при помощи мыши или клавиатуры. Для выделения строк нажимается левая кнопка

мыши в начале выделяемого фрагмента и, не отпуская ее, курсор мыши перемещается в конец этого фрагмента. Для выделения столбцов производятся те же действия, но, кроме того, нажимается клавиша <Alt> на клавиатуре.

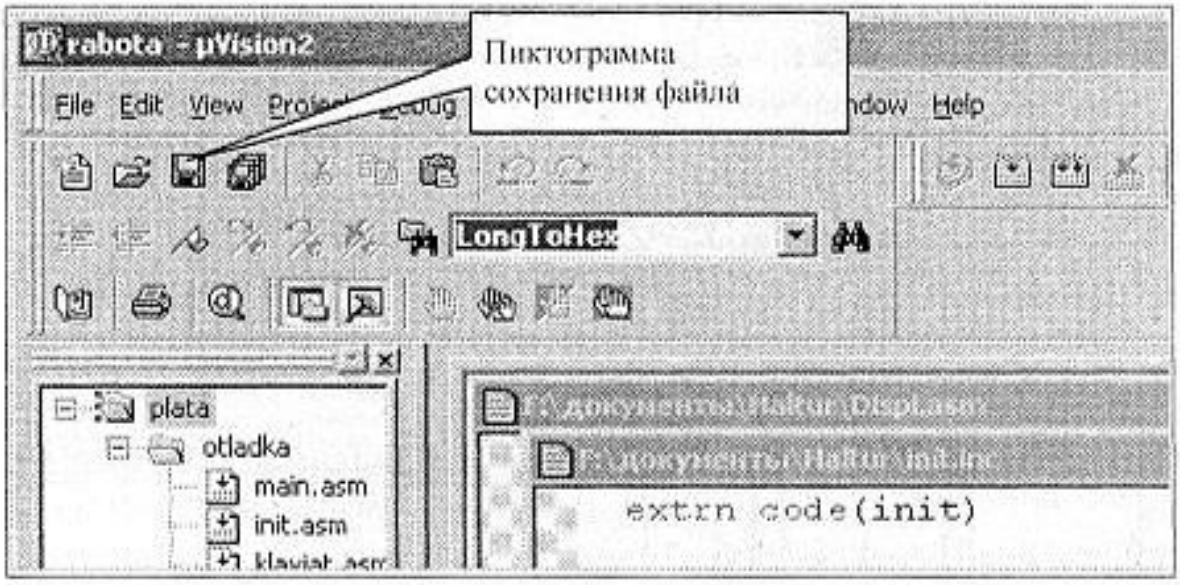


Рис. 24.4. Сохранение файла при помощи пиктограммы

После выделения необходимого фрагмента текста этот фрагмент копируется в буфер обмена. Скопировать можно, щелкнув мышью по пиктограмме копирования, как это показано на рис. 24.5.

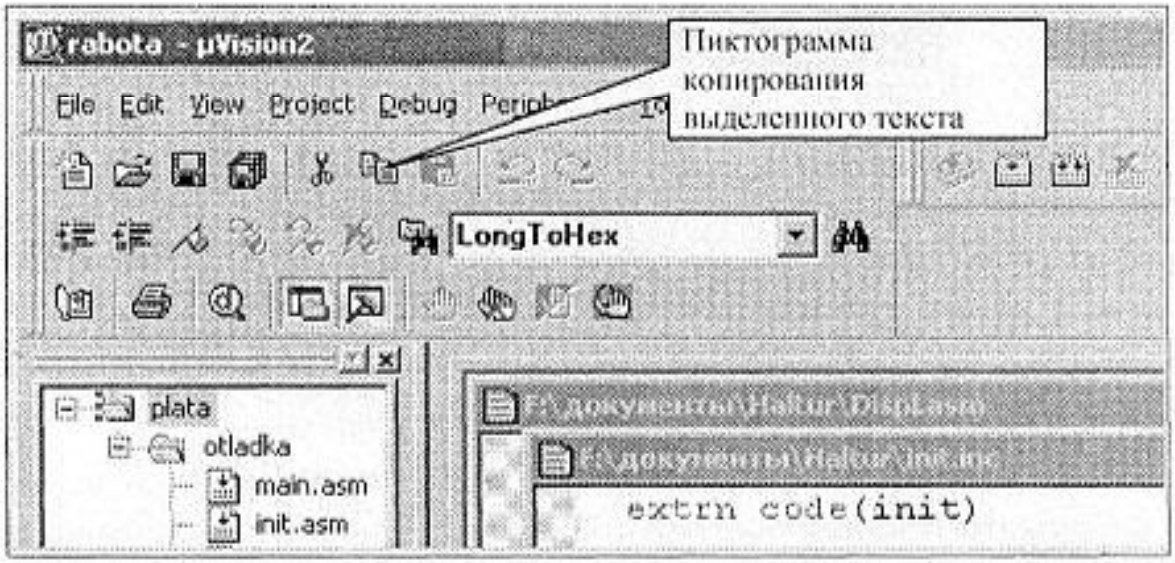


Рис. 24.5. Копирование выделенного фрагмента в буфер обмена при помощи пиктограммы

Теперь можно переключиться в окно редактирования файла, куда нужно поместить скопированный текстовый фрагмент при помощи главного меню, как это показано на рис. 24.6, и вставить этот фрагмент перед текстовым курсором.

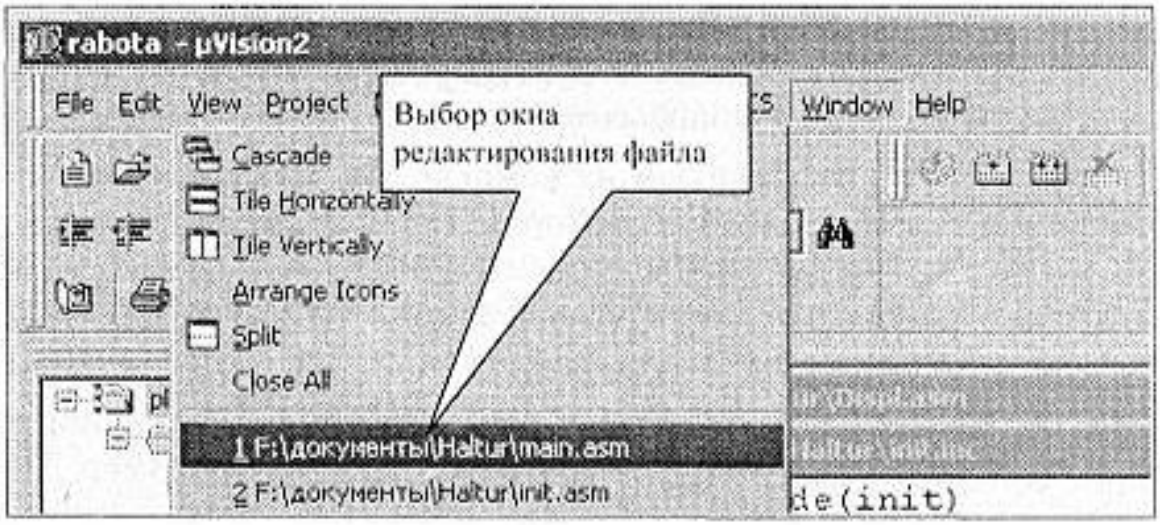


Рис. 24.6. Выбор окна редактирования файла

Вставка фрагмента из буфера обмена может быть произведена либо из меню **Edit** (редактировать), либо при помощи пиктограммы **Paste** (вставить), как показано на рис. 24.7. Фрагмент исходного текста будет вставлен в то место набираемого текста, где в настоящее время находится курсор.

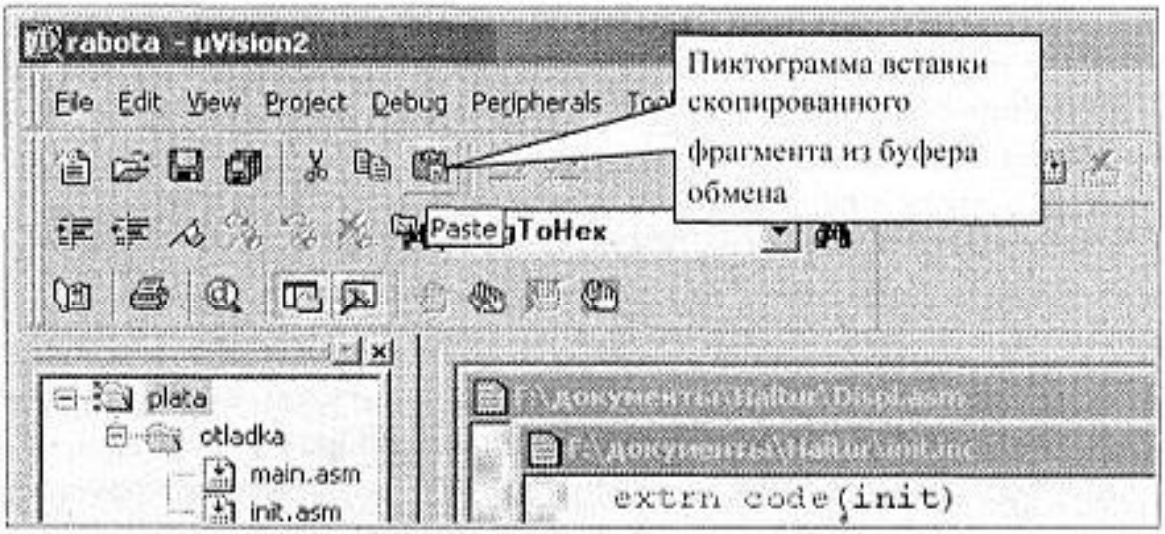


Рис. 24.7. Вставка скопированного фрагмента из буфера обмена при помощи пиктограммы

Напомним, что копирование участка исходного текста программы эквивалентно набору этого текста программы с клавиатуры и поэтому файл, в который производилось копирование текста, необходимо сохранить на диске любым из методов, описанных ранее.

Создание программных проектов

На первом шаге разработки программных средств формулируются технические требования к системе и составляется блок-схема процесса решения нуж-

ных задач, которая обеспечит реализацию заданных требований. Блок-схема должна быть надлежащим образом структурирована, чтобы гарантировалась высокая эффективность логики программ.

Для удобства навигации по файлам на компьютере, решение каждой задачи обычно помещают в отдельную директорию (папку). Написание программы тоже следует начинать в отдельной папке, тем более что (как будет показано далее) даже простейшая программа обычно состоит из нескольких файлов.

Для того чтобы не запутаться в этих файлах при разработке программы, их размещают в своей, отдельной директории. Имя этой директории обычно назначают по имени программно-аппаратного проекта. Например: "мобильная радиостанция", "стационарная радиостанция" или "контроллер базовой станции".

В большинстве случаев программа состоит из нескольких программных модулей. Использование в составе одной программы нескольких программных модулей позволяет увеличить скорость трансляции программ, поручать написание программных модулей различным программистам, делать программы более понятными.

Принцип разбиения единой программы на программные модули заключается в том, что для реализации работы с отдельными узлами аппаратуры пишутся отдельные подпрограммы, которые относительно слабо связаны с остальными частями программы. Эти подпрограммы можно выделить в отдельную программу, которую можно хранить в отдельном файле, и транслировать отдельно от остальной программы. Часто одни и те же модули могут входить в состав нескольких программ, выполняющих различные задачи, но использующие при этом одни и те же устройства.

В качестве примера можно назвать работу с клавиатурой, индикацию различных видов информации, работу с последовательными портами, с АЦП, с ЦАП. Каждое из этих устройств может обслуживаться отдельными программными модулями. Этот список можно продолжать и далее, но для составления представления о программных модулях этого достаточно.

Часто программа пишется и отлаживается на одной аппаратуре (например, на оценочных платах, предлагаемых фирмами-изготовителями микросхем), а используется на другой. При этом программа при отладке немного отличается от программы, которая будет использоваться в реальной аппаратуре. Для того чтобы учесть эти отличия, в составе программного проекта создаются назначения (target) проекта. В качестве примера назначений программного проекта можно назвать отладку и реализацию, а также различные версии программы.

Для облегчения процесса разработки программы часто используются системы поддержки разработок. В состав систем поддержки разработок включает-

ся определенный набор программных средств: редакторы текстов, компиляторы, компоновщики, загрузчики и т. д. Каждая из этих программ создает свой файл (или свои файлы) на диске компьютера.

Программные проекты можно создавать и поддерживать вручную, но в последнее время обычно используются различные системы поддержки разработок. Это создает ряд дополнительных преимуществ. При работе с программными проектами вручную требуется каждый раз при запуске компилятора прописывать полный путь транслируемого файла. Кроме того, приходится задавать параметры компиляции в виде ключей, передаваемых компилятору в командной строке. Это достаточно неудобно. Конечно, частично можно автоматизировать этот процесс, написав командный файл компиляции файла, но это не снимает всех проблем работы с программным проектом.

Второй этап работы с программой является поиск синтаксических ошибок. При ручной работе эта работа заключается в поиске по файлу листинга ошибок и исправление их в исходном тексте программного модуля. Этот процесс тоже достаточно трудоемок. Интегрированная среда программирования облегчает данный процесс тем, что в окне результата трансляции программного модуля выводится список ошибок, а при двойном щелчке мышью по строке ошибки вы сразу оказываетесь на соответствующей строке исходного текста программы.

Создание программного проекта в интегрированной среде программирования keil-C

Работа с программными проектами начинается с создания нового файла проекта. Для создания файла проекта в интегрированной среде разработки программ можно воспользоваться главным меню, как показано на рис. 24.8.

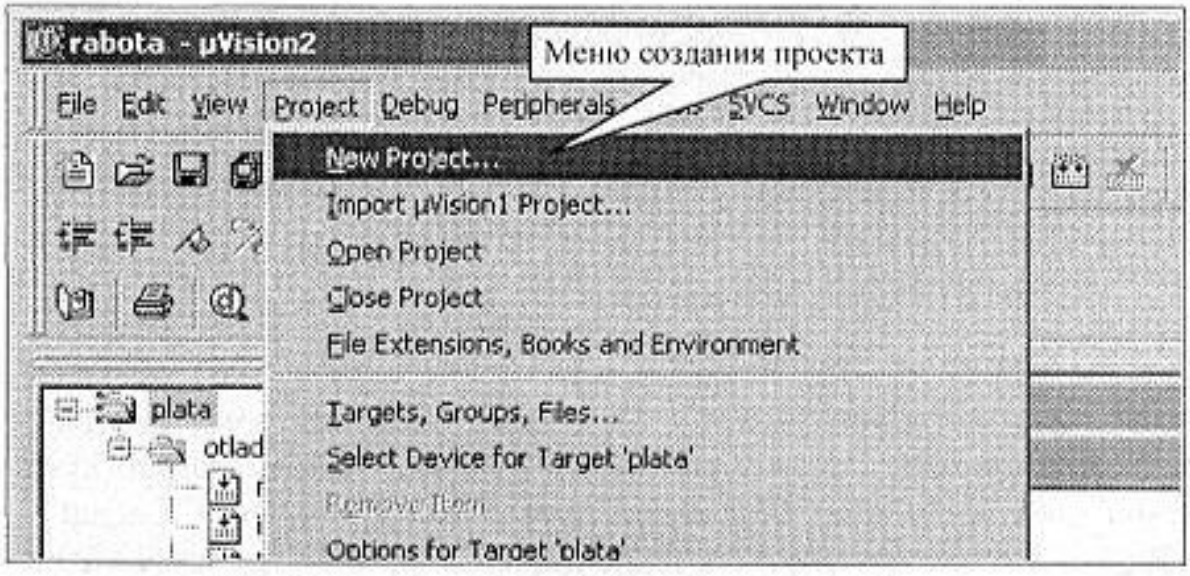


Рис. 24.8. Создание нового программного проекта

Интегрированная среда программирования предлагает создавать проекты в отдельной специализированной для своих проектов папке на диске компьютера. Рекомендуется не пользоваться этой папкой, а создать свою отдельно от папок, используемых интегрированной средой программирования. Точно так же не стоит пользоваться папкой "Мои документы", предлагаемой операционной системой Windows. Это связано с тем, что при переустановке либо компилятора, либо операционной системы исходные тексты ваших программ могут быть стерты.

После создания новой директории и нового файла программного проекта интегрированная среда программирования предлагает выбрать конкретную микросхему из семейства MCS-51, как показано на рис. 24.9.

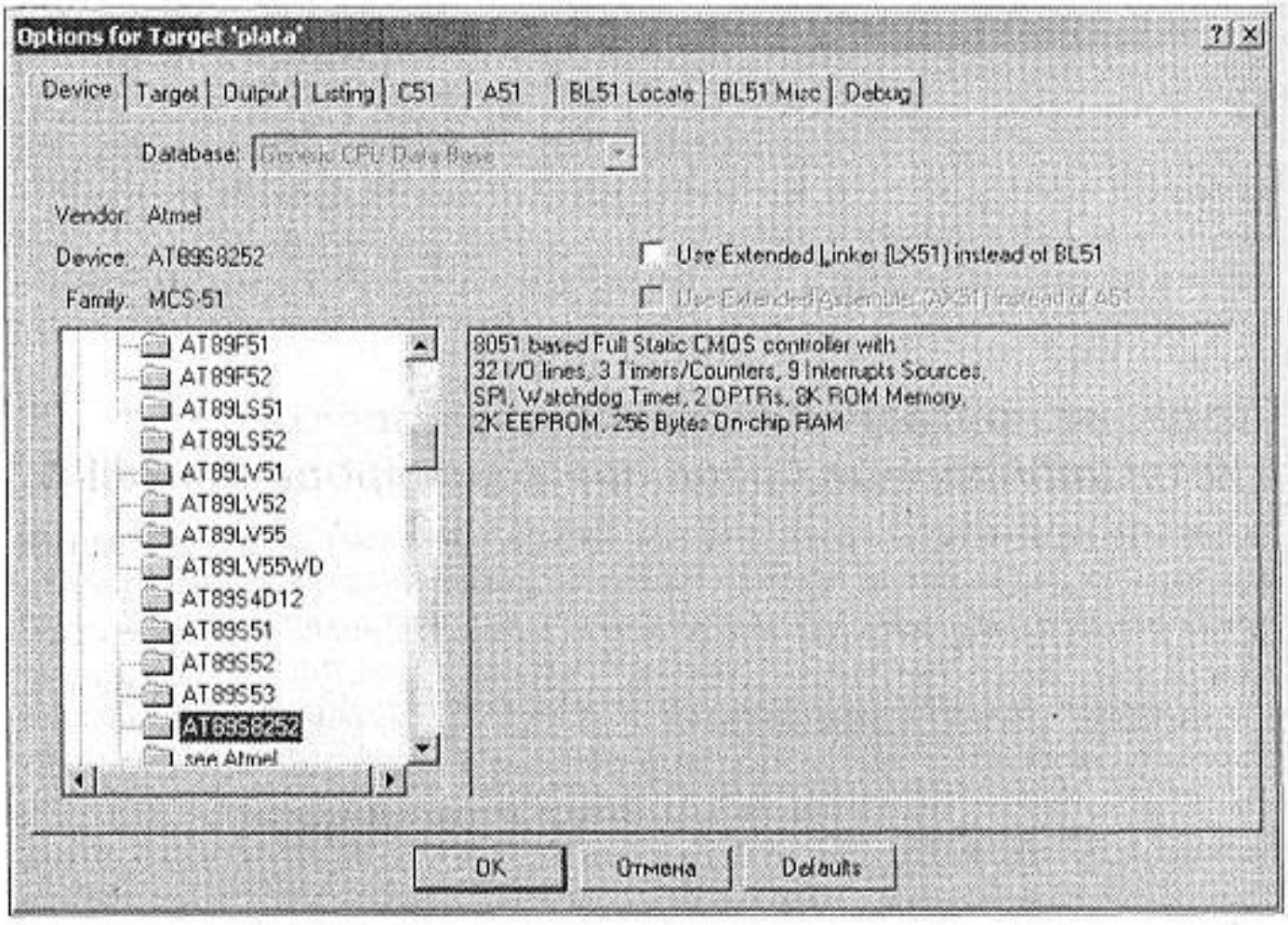


Рис. 24.9. Диалоговое окно выбора конкретной микросхемы для программного проекта

При этом в интегрированной среде программирования окно менеджера проекта приобретает вид, показанный на рис. 24.10. По умолчанию среда программирования предлагает структуру программного проекта **Target 1** — назначение 1, в состав которого входит Source Group 1 — группа файлов 1. Название назначения программного проекта можно изменить, щелкнув кнопкой

мыши по названию назначения программного проекта в окне менеджера проекта (например **Target 1** можно изменить на следующие названия: otladka — отладка, realizacija — реализация или soprovojdenie — сопровождение). Точно так же можно изменить название устройства в составе программного проекта (например **Source Group 1** можно заменить на следующие названия: nosimaja — носимая радиостанция, avtomobilnaja — автомобильная радиостанция, stacionarnaja — стационарная радиостанция или bazovaja — базовая радиостанция).

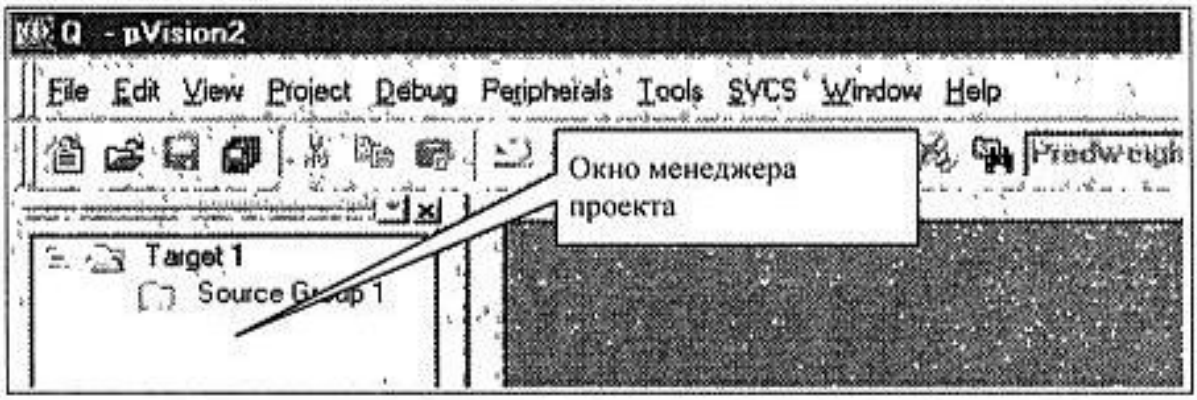


Рис. 24.10. Внешний вид окна менеджера проекта после создания программного проекта

Настройка свойств программного проекта в интегрированной среде программирования keil-C

После создания программного проекта в интегрированной среде программирования keil-C конечным файлом трансляции является абсолютный файл, совпадающий с названием программного проекта. На диске он хранится без расширения. Его назначение мы рассматривали в *главах 22* и *23*. Для загрузки в микросхему обычно используется HEX-файл. Для создания этого файла необходимо включить соответствующую опцию в свойствах программного проекта.

Изменить свойства программного проекта можно несколькими способами. Первый способ — воспользоваться главным меню, как показано на рис. 24.11. Второй способ — это нажать на кнопку изменения свойств программного проекта, как показано на рис. 24.12.

При этом на экране компьютера появляется диалоговое окно изменения свойств программного проекта, как показано на рис. 24.13. В этом окне необходимо ввести параметры внешней памяти программ и памяти данных.

При написании программ для устройств с внешней памятью программ нужно учитывать эту особенность разработки. Поэтому начальный адрес памяти

программ необходимо установить за пределами внутренней памяти программ, например, 0x2000. При использовании внешней памяти данных нужно установить ее начальный адрес, например, с адреса 0x8000.

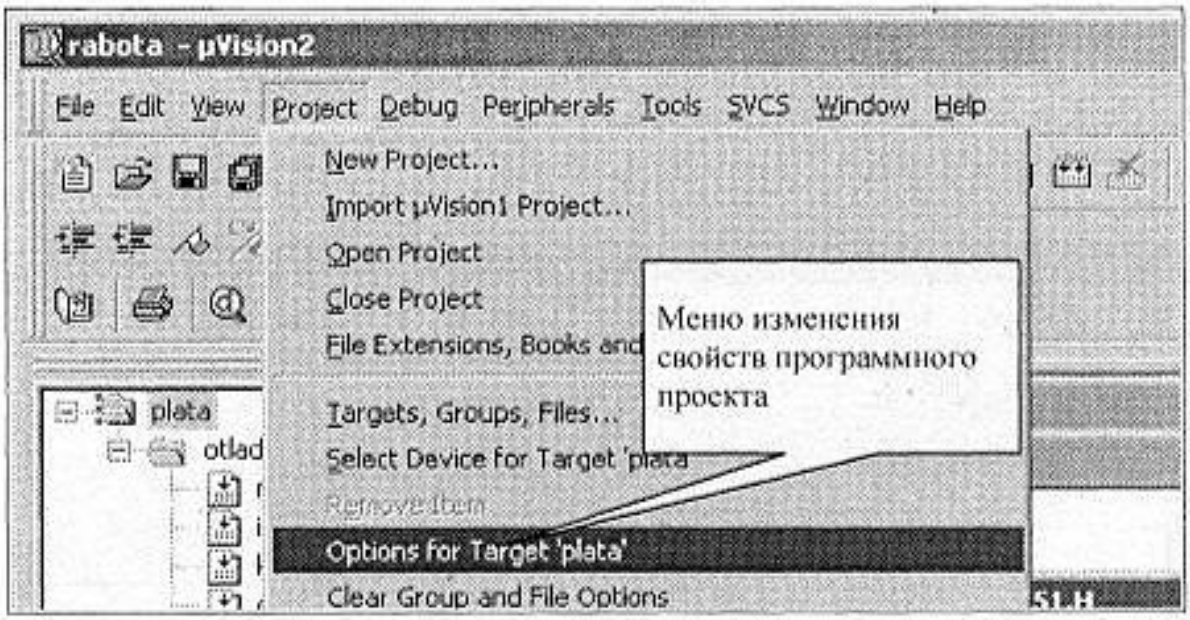


Рис. 24.11. Изменение свойств программного проекта через главное меню

Рис. 24.12. Изменение свойств программного проекта при помощи пиктограммы

Затем необходимо установить выходные параметры программного проекта. Для этого открываем вкладку **Output** (выход), как показано на рис. 24.14. В этой вкладке убеждаемся, что установлена галочка создания выходного загрузочного файла в HEX-формате. Именно этот файл будет загружаться во внутреннюю память микроконтроллера. Для того чтобы не загромождать директорию проекта файлами объектных кодов, можно (и нужно) создать отдельную директорию. Например, с названием OBJ. Новая директория может быть создана после нажатия на кнопку **Select Folder for Object** (выбрать папку для хранения объектных файлов).

Точно так же можно создать директорию (папку) для файлов листингов. Для этого необходимо выбрать вкладку **Listing** (выбрать папку для хранения файлов листингов). В файлах листингов помещается информация об ошибках, ассемблерный код и соответствующий ему машинный код программного

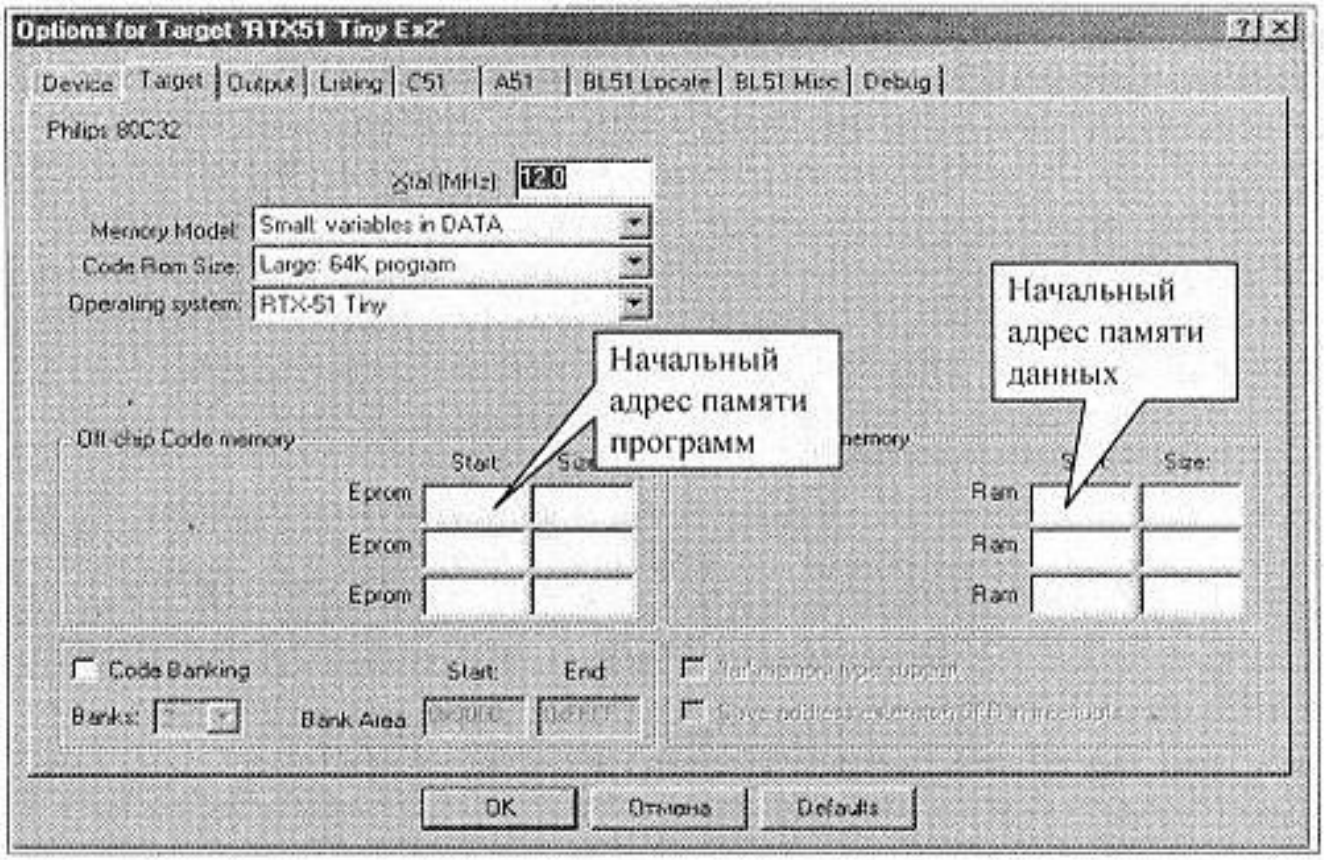


Рис. 24.13. Диалоговое окно настройки свойств программного проекта

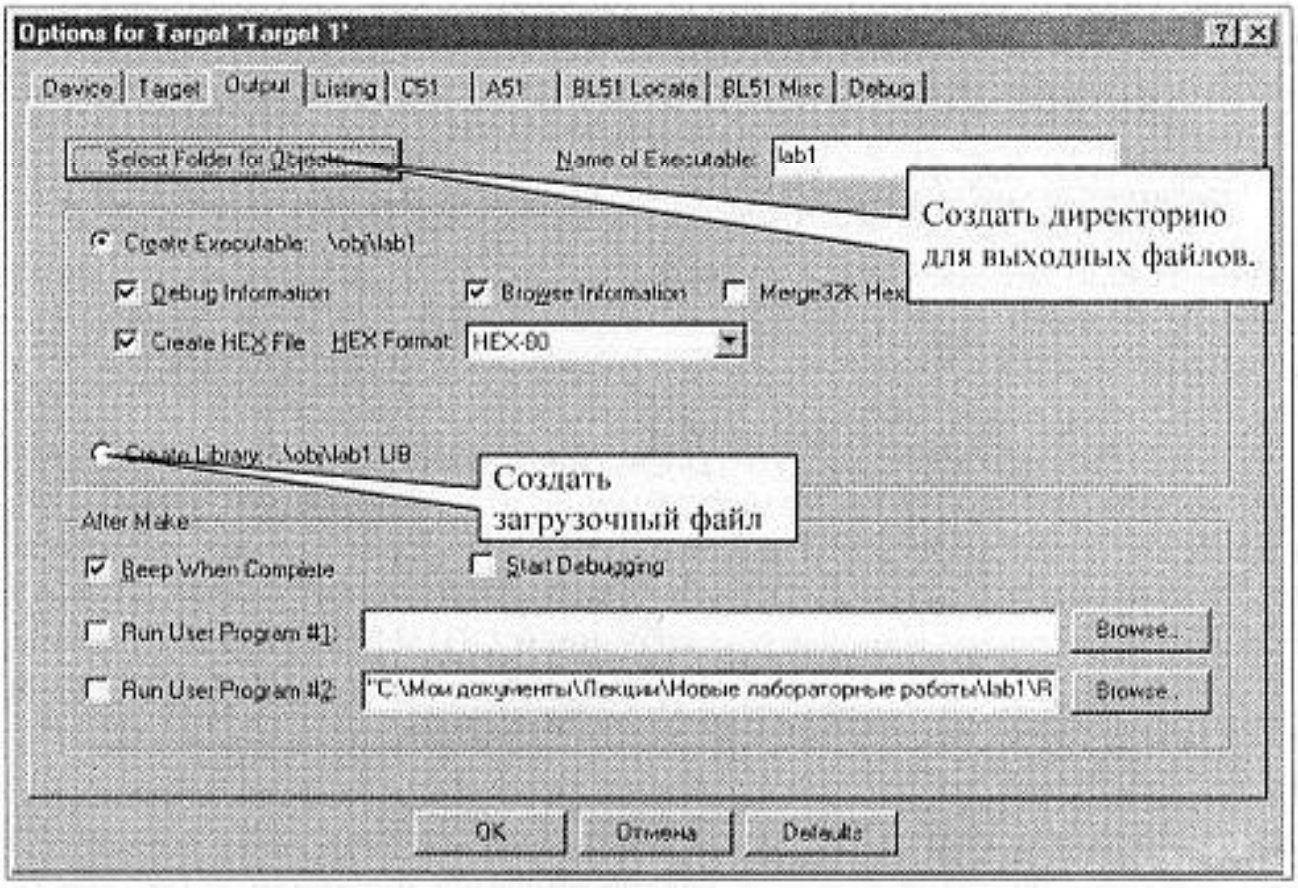


Рис. 24.14. Диалоговое окно настройки выходных параметров программного проекта

модуля. Использование листингов позволяет оптимизировать программу, а при работе без интегрированной среды программирования и находить синтаксические ошибки программы. Отметим, что создание файлов листингов замедляет процесс трансляции. Обычно имя для папки листингов выбирают LST.

Необходимость создания отдельных папок для листингов и объектных кодов возникает при большом количестве программных модулей, а значит, при большом количестве файлов в одной папке. Обычно директории для листингов и для объектных кодов располагают внутри папки программного проекта, где содержатся исходные тексты программы.

Для настройки параметров компиляции выбирается вкладка **C51**. В этой вкладке настраивается уровень оптимизации транслируемого программного модуля и цель оптимизации (по скорости работы программы или по размеру выходного файла). Кроме того, в этой вкладке заносится адрес векторов прерывания.

После настройки свойств программного проекта в диалоговом окне оно закрывается нажатием кнопки **OK**. Если нужно отменить все сделанные изменения программного проекта, то нажимается кнопка **Отмена**.

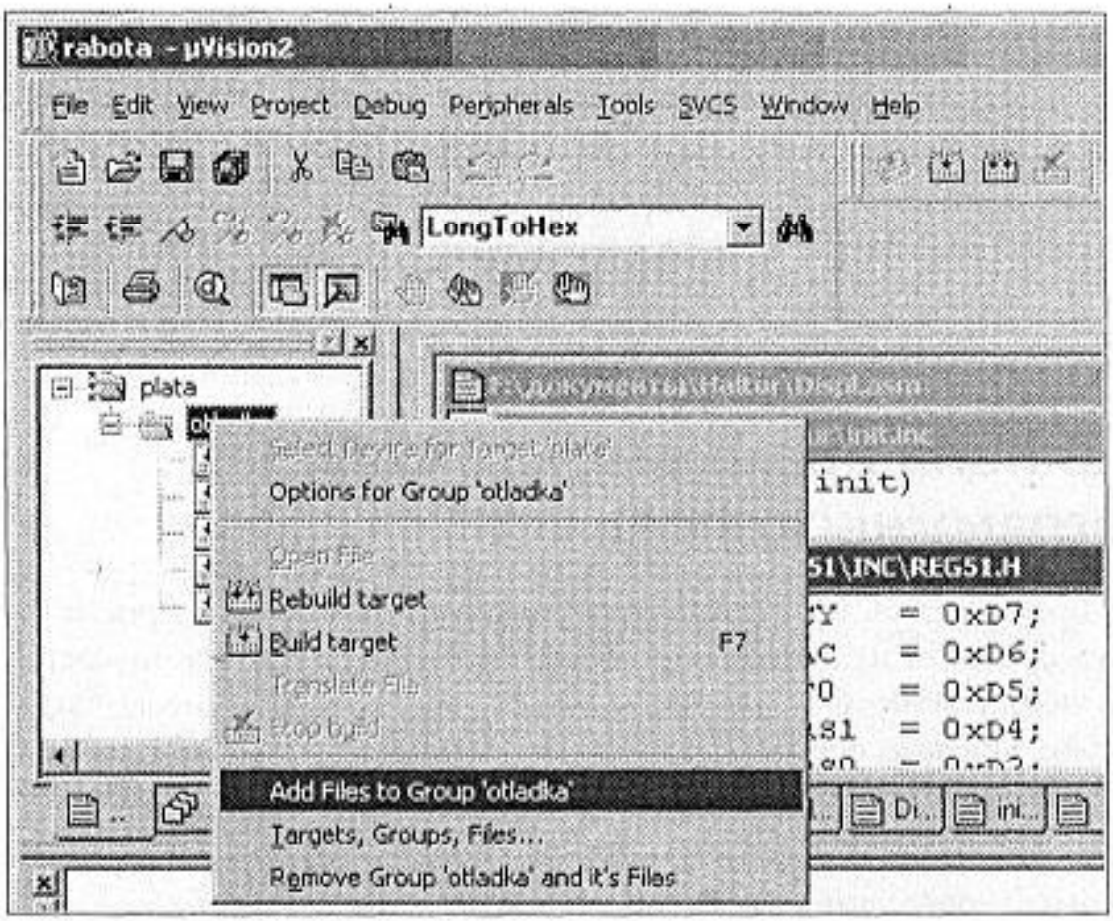


Рис. 24.15. Всплывающее меню менеджера проектов с выбранной опцией добавления файлов к программному проекту

Теперь следует подключить к программному проекту файлы с исходным текстом программных модулей. Для этого нужно щелкнуть правой кнопкой мыши по значку группы файлов в окне менеджера проектов, как это показано на рис. 24.15, и выбрать опцию добавления файлов к программному проекту. ✧ *Внимание! Если файлы с исходным текстом программы не будут подключены к проекту, то они не могут быть оттранслированы и связаны с другими модулями программы.*

Исходные тексты файлов программного проекта создаются точно так же, как это было описано ранее, при рассмотрении текстового редактора.

Работа с программным проектом в интегрированной среде программирования keil-C

После завершения создания программного проекта и включения в него файлов с написанной на языке С или ассемблере программой, можно открыть исходные тексты программных модулей, щелкнув левой кнопкой мыши на названии соответствующего файла в окне менеджера проектов.

Между программными модулями проекта можно переключаться, используя окно менеджера проектов. Для этого следует щелкнуть левой кнопкой мыши на названии соответствующего файла в окне менеджера проектов.

В случае необходимости можно выключать окно менеджера программных проектов, щелкнув левой кнопкой мыши по пиктограмме  , но обычно использование этого окна удобно при написании программы. Повторный щелчок по этой пиктограмме приводит к включению окна менеджера проектов. Если окно менеджера проектов отключено, то переключаться между исходными текстами программных модулей можно, используя меню **Window** (окна).

Трансляция программных модулей и программных проектов

Как уже обсуждалось в предыдущих главах, программный проект (программа) может состоять из нескольких модулей. В программном проекте, состоящем из нескольких файлов, появляется возможность нескольких видов трансляции. Прежде всего, можно оттранслировать только один файл программного модуля, не транслируя остальные модули программного проекта.

Только после того, как были оттранслированы все программные модули, производится трансляция всего программного проекта в целом. Для экономии времени трансляции обычно осуществляется сокращенная трансляция программного проекта. При этом производится одновременная трансляция

последнего измененного программного модуля и компоновка программного проекта.

Иногда для полного соответствия программы исходному тексту производится трансляция всех программных модулей, входящих в программный проект, и связывание их в конечную программу (абсолютный модуль программы).

Трансляция программных модулей

Независимая трансляция отдельных программных модулей позволяет обнаружить и исправить все синтаксические ошибки в отдельном программном модуле. При трансляции каждого программного модуля на жестком диске формируются перемещаемый файл в объектном формате, который затем будет использоваться для создания загрузочного файла программного проекта. Этот файл называется объектным модулем.

Одновременно с объектным модулем на диске формируется файл листинга программного модуля, в который помещается исходный текст программного модуля и сообщения о синтаксических ошибках. В листинге 24.1 приведен пример исходного текста программы, написанного на языке программирования С, а в листинге 24.2 приведено содержимое файла листинга, создаваемого при трансляции этого исходного текста.

Листинг 24.1. Исходный текст программы

```
/*======================== main.c ============================
Программа лабораторной работы № 5

Проект          ОБУЧАЮЩИЕ ПРОГРАММЫ
Программист     :::::::::..
Версия          1.00
Написана        ::::::.

Последнее
изменение       ::::.
программы
=============================================================*/

#include "ADuC812.h"     //Подключить определения всех внутренних
                         //регистров ADuC812

#define ParPort 0x8      //Определить константу переключения
                         // на 8-ю страницу памяти
```

```
#define SvDiod *(volatile unsigned char xdata *)7 /*Переменная косвенной
                    адресации 7-й ячейки  внешней памяти (светодиодов)*/

main()
{DPP=ParPort;    //Подключиться к параллельному порту
 SvDiod=1;      //Зажечь светодиод номер 0

 while(1)        //Обеспечить зацикливание микроконтроллера
  {              /*(чтобы не было программного сброса микросхемы      */
  }              /*языком программирования)                           */
}
```

Листинг 24.2. Содержимое файла листинга

```
C51 COMPILER V7.20   MAIN                  08/06/2006      23:14:21 PAGE 1

C51 COMPILER V7.20, COMPILATION OF MODULE MAIN
OBJECT MODULE PLACED IN .\obj\main.obj
COMPILER INVOKED BY: C:\Keil\C51\BIN\C51.EXE main.c BROWSE DEBUG
OBJECTEXTEND PRINT(.\lst\
                   -main.lst) PAGEWIDTH(90) OBJECT(.\obj\main.obj)

line level    source

   1          /*========================== main.c
============================
   2          Программа лабораторной работы № 5
   3
   4          Проект         ОБУЧАЮЩИЕ ПРОГРАММЫ
   5          Программист    :::::::::..
   6          Версия         1.00
   7          Написана       ::::::.
   8
   9          Последнее
  10          изменение      :::::.
  11          программы
  12
============================================================*/
  13
  14 #include "ADuC812.h"//Подключить определения всех внутренних
                         //регистров ADuC812
  15
```

```
 16  #define ParPort 0x8  //Определить константу переключения
                          //на 8-ю страницу памяти
 17
 18  #define SvDiod *(volatile unsigned char xdata *)7 /*Переменная
 19        косвенной адресации 7-й ячейки внешней памяти (светодиодов)*/
 20
 21          main()
 22          {DPP=ParPort;    //Подключиться к параллельному порту
 23    1      SvDiod=1;                //Зажечь светодиод номер 0
 24    1
 25    1      while(1)          //Обеспечить зацикливание микроконтроллера
 26    1      {               /*(чтобы не было программного сброса
                                 микросхемы      */
 27    2       }              /*языком программирования)   */
 28    1      }
 29
 30
```

```
MODULE INFORMATION:   STATIC OVERLAYABLE
   CODE SIZE          =     11     ----
   CONSTANT SIZE      =   ----     ----
   XDATA SIZE         =   ----     ----
   PDATA SIZE         =   ----     ----
   DATA SIZE          =   ----     ----
   IDATA SIZE         =   ----     ----
   BIT SIZE           =   ----     ----
END OF MODULE INFORMATION.

C51 COMPILATION COMPLETE.  0 WARNING(S),  0 ERROR(S)
```

При трансляции программного модуля, написанного на языке программирования ассемблер, в листинг помещаются машинные коды команд процессора, и их адрес относительно начала программного модуля. Пример содержимого такого файла приведен в листинге 24.3.

Листинг 24.3. Содержимое файла листинга программы, написанной на ассемблере

```
A51 MACRO ASSEMBLER  Модуль инициализации процессора
05/24/2004 14:00:11 PAGE     1

MACRO ASSEMBLER A51 V7.01
OBJECT MODULE PLACED IN .\OBJ\init.obj
```

```
ASSEMBLER INVOKED BY: F:\Keil\C51\BIN\A51.EXE init.asm NOMOD51 SET(SMALL)
DEBUG PRINT(.\LST\init.lst) XREF OBJECT(.\OBJ\init.obj) EP

LOC  OBJ             LINE     SOURCE

                        1     $title  (Модуль инициализации процессора)
                        2
                        3 ;========== init.asm ======================
                        4     ;Учебная программа
                        5     ;
                        6     ;Проект         Аттестационная работа
                        7     ;Программист    Борисенко С.    т. 66-02-86
                        8     ;Версия         1.00
                        9     ;Написана       9 апреля 2004.
                       10     ;
                       11     ;Последнее      12 апреля 2004.
                       12     ;изменение      v1.00
                       13     ;программы
                       14
;=============================================================
                      254     $LIST
                      255
                      256     _code segment code
                      257
                      258       public init
                      259
----                  260       rseg _code
                      261
                      262    ;----------------------------------------
                      263    ;ПОДПРОГРАММА ИНИЦИАЛИЗАЦИИ МИКРОКОНТРОЛЛЕРА
                      264    ;----------------------------------------
                      265  ;настраивает таймер 0 в качестве системных
                            часов на период 10 мс
                      266  ;(Этого периода достаточно для устранения
                            дребезга клавиатуры)
                      267
0000                  268     init:
                      269     ;--- Настроить Timer 0 --------------------
0000 758901           270       mov  TMOD, #00000001b
                      271     ;     ||||
                      272   ;     ||++--Выбрать режим 16-разрядного таймера
                      273   ;   |+----Использовать внутреннюю синхронизацию
                      274   ;   +-----Запретить управление таймером от INT0
                      275
```

```
0003 758A94          276       mov  TL0,  #LOW(-(F_ZQ/12)*10) ;Настроить
                                                               таймер
0006 758C94          277       mov  TH0,  #HIGH(-(F_ZQ/12)*10)
                                                    ;на период 10 мс
                     278
0009 D28C            279       setb TR0                ;Включить таймер 0
                     280     ;-------------------------------------------
                     281
                     282     ;--- Погасить светодиоды ------------------
000B C293            283       clr SV1
000D C294            284       clr SV2
                     285
000F 438E01          286       orl AUXR,#1   ;Запретить выдачу частоты
                                               ALE на выход
                     287                              ;микроконтроллера
0012 75A882          288       mov IE,#10000010b
                     289             ;|     |
                     290             ;|     +--Разрешить прерывания
                                              от таймера 0
                     291             ;+--------Разрешить прерывания
                     292     ;-------------------------------------------
                     293
0015 120000    F     294       call IndInit ;Инициализировать ЖКИ-дисплей
0018 22              295       ret
                     296
                     297       end
A51 MACRO ASSEMBLER  Модуль инициализации процессора
05/24/2004 14:00:11 PAGE     3

XREF SYMBOL TABLE LISTING
---- ------ ----- -------

N A M E              T Y P E  V A L U E   ATTRIBUTES / REFERENCES
AC . . . . . .       B ADDR   00D0H.6 A      204#
ACC. . . . . .       D ADDR   00E0H   A      60#
ADRKLAV. . . . .     D ADDR   00A0H   A      225#
EA . . . . . .       B ADDR   00A8H.7 A      162#
ES . . . . . .       B ADDR   00A8H.4 A      159#
IE . . . . . .       D ADDR   00A8H   A      56# 288
INDINIT. . . . .     C ADDR   -----        EXT   252# 294
INIT . . . . . .     C ADDR   0000H   R    SEG=_CODE   258 268#
INT0 . . . . . .     B ADDR   00B0H.2 A      178#
INT1 . . . . . .     B ADDR   00B0H.3 A      179#
IP . . . . . .       D ADDR   00B8H   A      58#
```

```
P0 . . . . . .  D ADDR   0080H   A      30#
P1 . . . . . .  D ADDR   0090H   A      52#
P2 . . . . . .  D ADDR   00A0H   A      55# 225
P3 . . . . . .  D ADDR   00B0H   A      57#
PCON . . . . . .  D ADDR   0087H   A      36#
PSW. . . . . .  D ADDR   00D0H   A      59#
PUTCHAR. . . . . .  C ADDR   -----       EXT   252#
SBUF . . . . . .  D ADDR   0099H   A      54#
SCON . . . . . .  D ADDR   0098H   A      53#
SP . . . . . .  D ADDR   0081H   A      31#
T0 . . . . . .  B ADDR   00B0H.4 A      180#
T1 . . . . . .  B ADDR   00B0H.5 A      181#
TCON . . . . . .  D ADDR   0088H   A      37#
TMOD . . . . . .  D ADDR   0089H   A      38# 270
_CODE. . . . . .  C SEG    0019H       REL=UNIT   256# 260

REGISTER BANK(S) USED: 0

ASSEMBLY COMPLETE.  0 WARNING(S), 0 ERROR(S)
```

При трансляции исходного текста программы, написанной на языке программирования высокого уровня, таком как C или PLM, программа-транслятор может быть настроена так, что она будет создавать файл с текстом исходного модуля, написанного на ассемблере, или помещать операторы языка программирования ассемблер в листинг программного модуля. Таким образом, появляется возможность писать и отлаживать программу на языке высокого уровня, а затем переводить ее на язык программирования ассемблер и окончательно оптимизировать ее вручную. В листинге 24.4 приведено содержимое файла листинга, создаваемого при трансляции исходного текста программы, написанного на языке C-51, с ассемблерным кодом.

Листинг 24.4. Содержимое файла листинга

```
C51 COMPILER V7.20   MAIN                    08/06/2006     23:28:20 PAGE 1

C51 COMPILER V7.20, COMPILATION OF MODULE MAIN
OBJECT MODULE PLACED IN .\obj\main.obj
COMPILER INVOKED BY: C:\Keil\C51\BIN\C51.EXE main.c BROWSE DEBUG
OBJECTEXTEND CODE PRINT(.-\lst\main.lst) PAGEWIDTH(90)
OBJECT(.\obj\main.obj)

line level    source
```

```
  1          /*========================= main.c ========================
  2          Программа лабораторной работы № 5
  3
  4          Проект          ОБУЧАЮЩИЕ ПРОГРАММЫ
  5          Программист     :::::::::..
  6          Версия          1.00
  7          Написана        ::::::.
  8
  9          Последнее
 10          изменение       ::::.
 11          программы
 12
=============================================================*/
 13
 14  #include "ADuC812.h"  //Подключить определения всех внутренних
 15                        //регистров ADuC812
 16  #define ParPort 0x8   //Определить константу переключения
 17                        //на 8-ю страницу памяти
 18  #define SvDiod *(volatile unsigned char xdata *)7 /*Переменная
 19      косвенной адресации 7-й ячейки внешней памяти (светодиодов)*/
 20
 21          main()
 22          {DPP=ParPort;    //Подключиться к параллельному порту
 23    1      SvDiod=1;                //Зажечь светодиод номер 0
 24    1
 25    1      while(1)          //Обеспечить зацикливание микроконтроллера
 26    1        {       /*(чтобы не было программного сброса микросхемы */
 27    2        }       /*языком программирования) */
 28    1      }
 29
 30
C51 COMPILER V7.20   MAIN                    08/06/2006   23:28:20 PAGE 2

ASSEMBLY LISTING OF GENERATED OBJECT CODE

            ; FUNCTION main (BEGIN)
                                             ; SOURCE LINE # 21
                                             ; SOURCE LINE # 22
0000 758408             MOV     DPP,#08H
                                             ; SOURCE LINE # 23
0003 900007             MOV     DPTR,#07H
0006 7401               MOV     A,#01H
0008 F0                 MOVX    @DPTR,A
```

```
0009          ?C0001:
                                                ; SOURCE LINE # 25
                                                ; SOURCE LINE # 26
                                                ; SOURCE LINE # 27
0009 80FE                  SJMP    ?C0001
             ; FUNCTION main (END)

MODULE INFORMATION:   STATIC OVERLAYABLE
   CODE SIZE          =     11     ----
   CONSTANT SIZE      =   ----     ----
   XDATA SIZE         =   ----     ----
   PDATA SIZE         =   ----     ----
   DATA SIZE          =   ----     ----
   IDATA SIZE         =   ----     ----
   BIT SIZE           =   ----     ----
END OF MODULE INFORMATION.

C51 COMPILATION COMPLETE.  0 WARNING(S),  0 ERROR(S)
```

Связывание объектных модулей и получение загрузочного файла

После того как оттранслированы без ошибок все программные модули и, тем самым, получены файлы объектных модулей, производится трансляция всего программного проекта (связывание объектных модулей). При этом на диске формируются абсолютный и загрузочный файлы программного проекта. Для контроля ошибок связывания формируется файл листинга редактора связей с расширением .m51.

Если обнаруживаются ошибки связывания, то абсолютный и загрузочный файлы программного проекта не формируются. В этом случае необходимо изменить исходный текст программного модуля, из-за которого возникла ошибка связывания, оттранслировать его и снова попытаться произвести трансляцию программного проекта (связывание объектных модулей). После получения загрузочного модуля можно начинать отладку программы.

Трансляция программных проектов

Иногда возникает необходимость произвести одновременно трансляцию всех программных модулей, входящих в проект, и произвести связывание полученных объектных модулей в абсолютный файл. Это можно производить вручную, вызывая программы компилятора c51.exe и asm51.exe, редактора

связей и преобразователя кодов oh.exe. Однако при создании программы трансляцию программного проекта приходится производить очень часто, поэтому такой способ получения загрузочного модуля достаточно трудоемок.

Этот процесс можно упростить при использовании специальных командных файлов, так называемых BAT-файлов. В этом файле можно описать последовательность действий, которую следует выполнять операционной системе при трансляции программного проекта. Однако и этот путь тоже остается трудоемким. При создании командного файла приходится устанавливать ключи вызываемых программ, описывать полные пути транслируемых, создаваемых и подключаемых файлов, что достаточно утомительно и требует высокой квалификации программиста. Намного удобнее при трансляции программных проектов пользоваться интегрированными средами программирования.

Применение интегрированной среды программирования keil-C для трансляции программного проекта

Интегрированная среда программирования позволяет максимально облегчить трансляцию программных проектов. Так как параметры программного проекта уже настроены, то для трансляции исходного текста программного модуля достаточно загрузить исходный текст этого программного модуля в окно текстового редактора. Это можно сделать одним из способов, рассмотренных ранее.

После загрузки исходного текста программного модуля достаточно нажать кнопку трансляции программного модуля, как это показано на рис. 24.16.

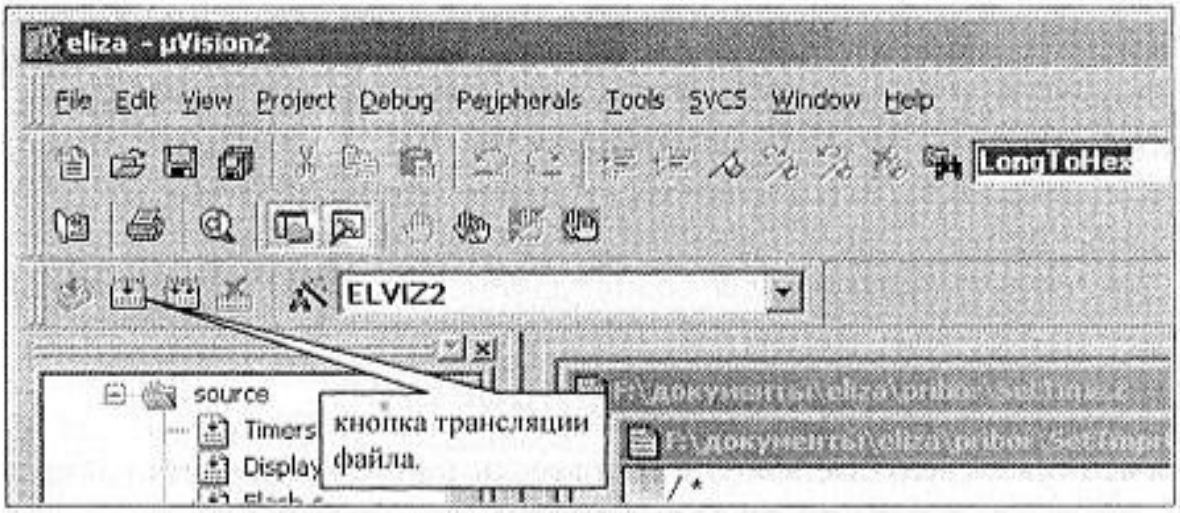


Рис. 24.16. Трансляция программного модуля при помощи кнопки трансляции файла

Еще один способ трансляции программного модуля — воспользоваться главным меню, как это показано на рис. 24.17.

Надо отметить, что в составе интегрированной среды программирования для поиска синтаксических ошибок удобнее пользоваться не файлом листинга, а окном **Build** (транслировать весь программный проект), где выводятся все сообщения об ошибках. При этом если дважды щелкнуть мышью по сообщению об ошибке в окне **Build**, то в окне текстового редактора будет выделена строка программы, где была обнаружена данная ошибка.

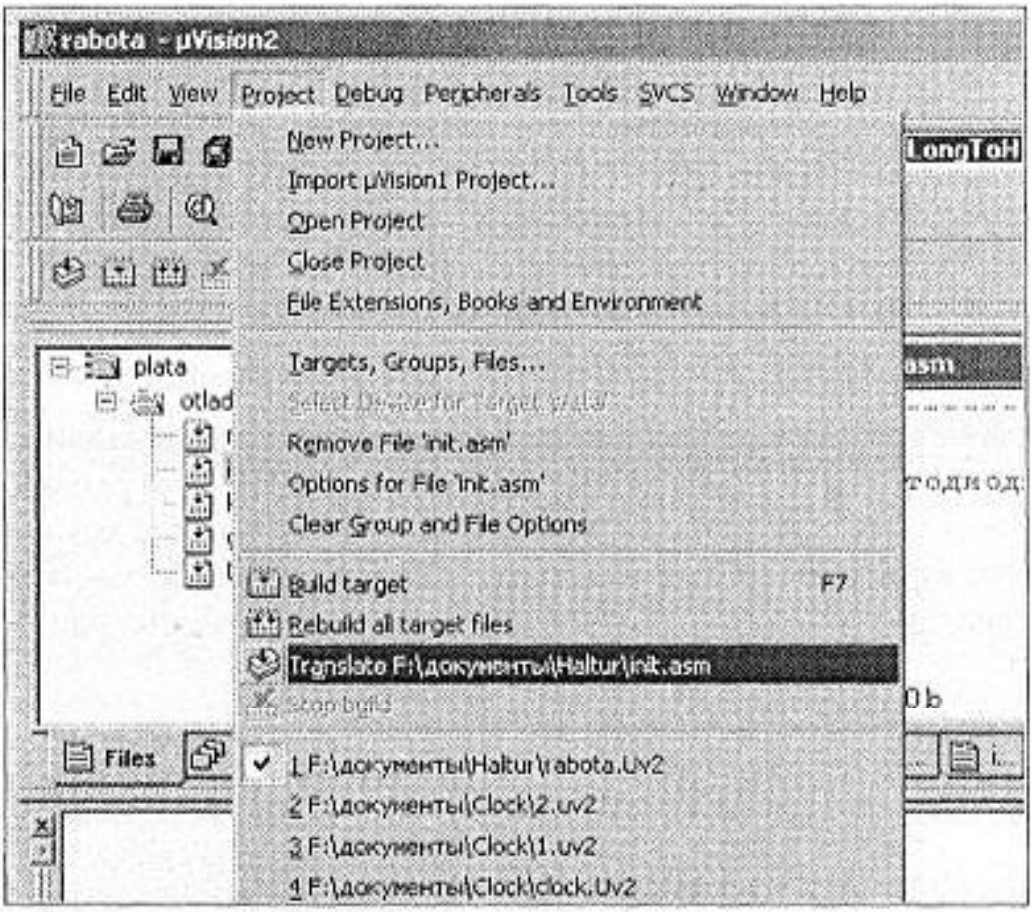


Рис. 24.17. Трансляция программного модуля при помощи главного меню

Трансляция программного модуля и получение загрузочного файла в интегрированной среде программирования производится нажатием кнопки **Build target**, как это показано на рис. 24.18. Еще один способ трансляции программного проекта в интегрированной среде программирования — воспользоваться главным меню, как это показано на рис. 24.19.

Если же необходимо оттранслировать все программные модули вне зависимости, имеются объектные модули или нет, и получить загрузочный файл, то нажимается кнопка **Rebuild all target files** (компилировать все модули и связать их в абсолютный модуль программы) или выбирается соответствующее меню. Такая необходимость возникает, если программа ведет себя, мягко го-

воря, странно. Это может быть связано с тем, что редактор связей использует устаревшие объектные модули, а интегрированная среда программирования считает их новыми.

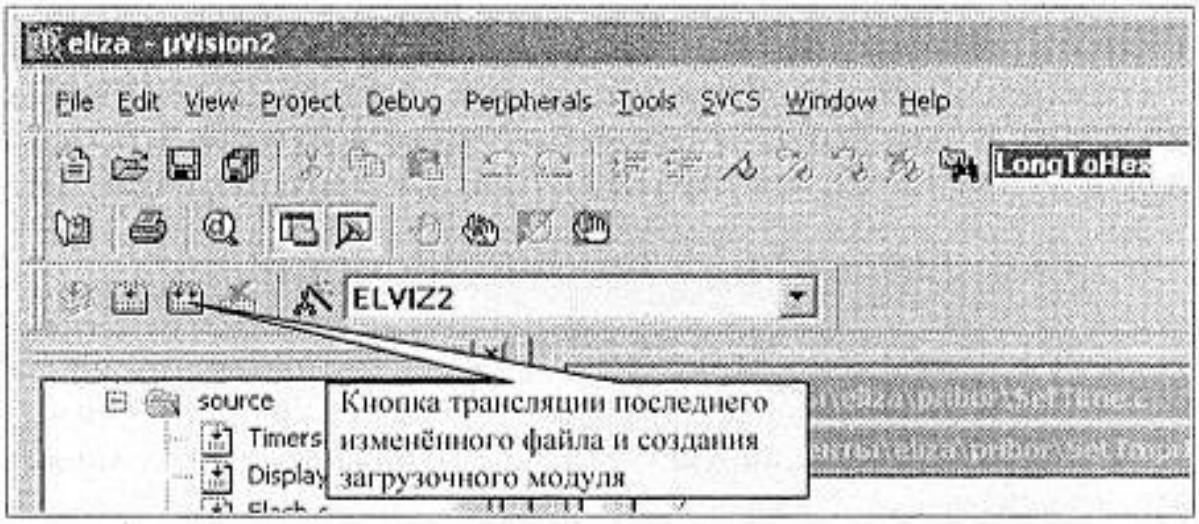


Рис. 24.18. Трансляция программного проекта при помощи кнопки **Build target**

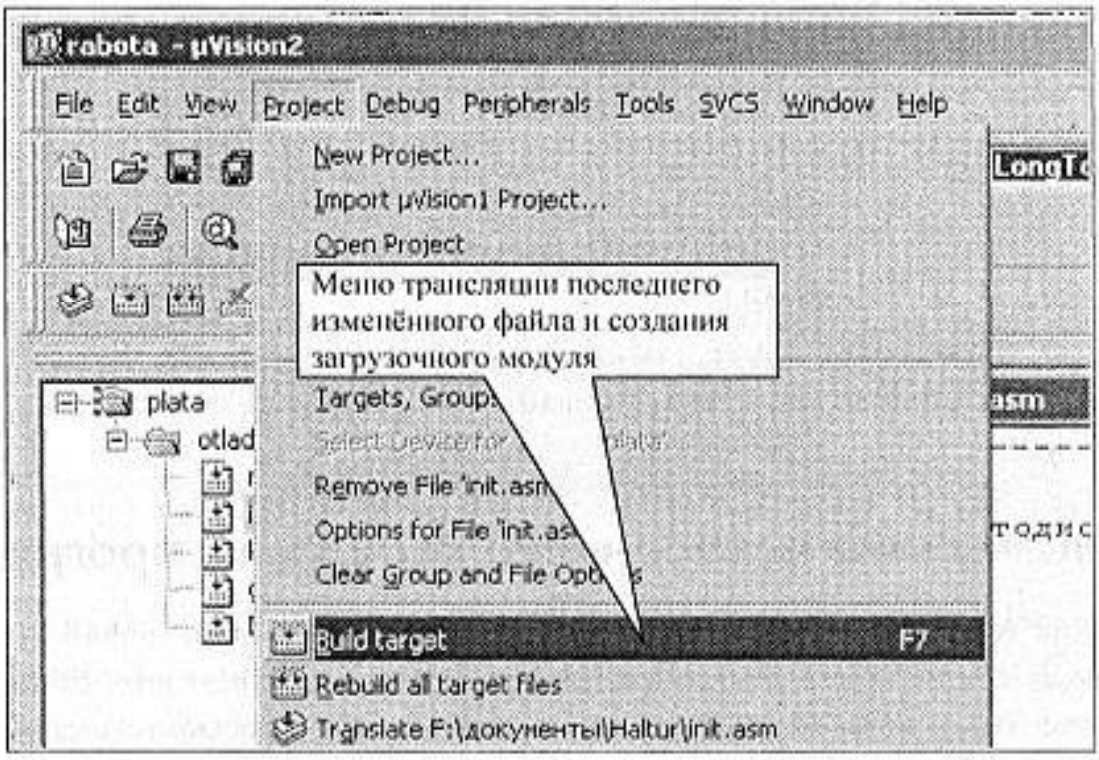


Рис. 24.19. Трансляция программного модуля при помощи главного меню

Перетранслировав все исходные файлы программного проекта можно быть уверенным, что полученный загрузочный модуль полностью соответствует исходному тексту программы, содержащемуся в написанных вами файлах.

Отладка программ во встроенном отладчике программ

Отладка программ заключается в проверки правильности работы программы и аппаратуры. Программа, не содержащая синтаксических ошибок, тем не менее, может содержать логические ошибки, не позволяющие программе выполнять заложенные в ней функции. Логические ошибки могут быть связаны с алгоритмом программы или неправильным пониманием работы аппаратуры, подключенной к портам микроконтроллера.

Способы отладки программ

Встроенный отладчик позволяет отладить те участки кода программы, которые не зависят от работы аппаратуры, не входящей в состав микросхемы микроконтроллера.

Для отладки программ обычно применяют три способа:

1. Пошаговая отладка программ с заходом в подпрограммы.
2. Пошаговая отладка программ с выполнением подпрограммы как одного оператора.
3. Выполнение программы до точки останова.

Пошаговая отладка программ заключается в том, что выполняется один оператор программы, затем контролируются те переменные, на которые должен был воздействовать данный оператор.

Если в программе имеются уже отлаженные подпрограммы, то подпрограмму можно рассматривать как один оператор программы и воспользоваться вторым способом отладки программ.

Использование встроенного отладчика программ

Вызов встроенного отладчика удобнее всего осуществить, нажав кнопку отладчика  на панели инструментов **File** (файлы), как показано на рис. 24.20.

Для вызова отладчика можно воспользоваться главным меню интегрированной среды программирования или воспользоваться комбинацией клавиш <Ctrl>+<F5>.

После этого внешний вид интегрированной среды программирования принимает вид, показанный на рис. 24.21. В верхней части программы появляется дополнительная панель инструментов отладчика программ. В нижней части программы появляется окно просмотра памяти контроллера и окно контроля переменных **Watch** (окно просмотра переменных).

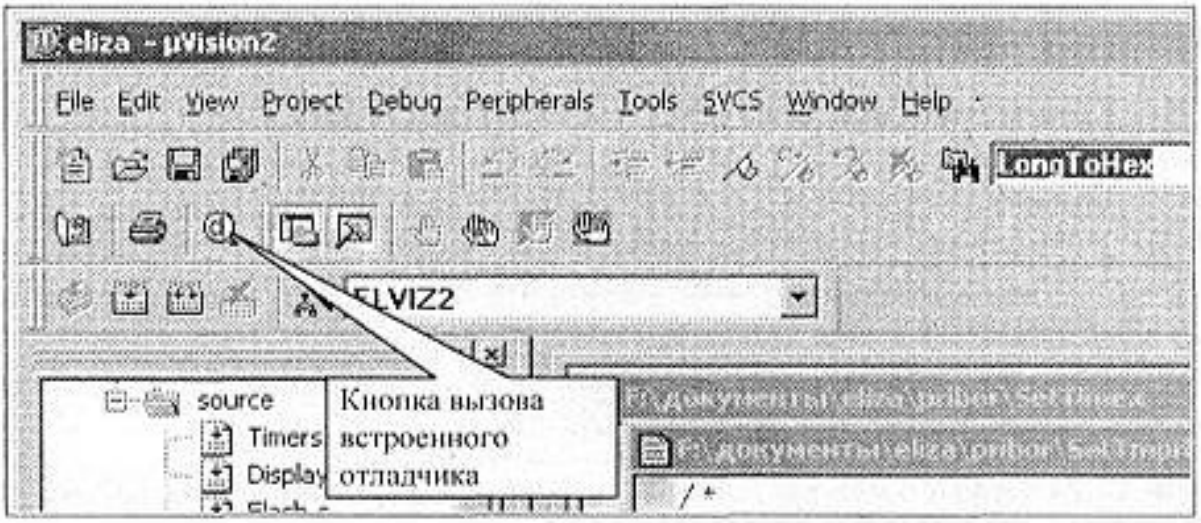


Рис. 24.20. Вызов встроенного отладчика с использованием кнопки на панели **File**

Окно просмотра памяти контроллера можно настроить на просмотр памяти программ или памяти данных, введя в диалоговое окно "адрес" ключ, двоеточие и адрес начальной ячейки памяти. Например:

- d:0 — просмотреть память данных начиная с нулевой ячейки;
- c:0 — просмотреть память программ начиная с нулевой ячейки;
- x:0 — просмотреть внешнюю память данных начиная с нулевой ячейки.

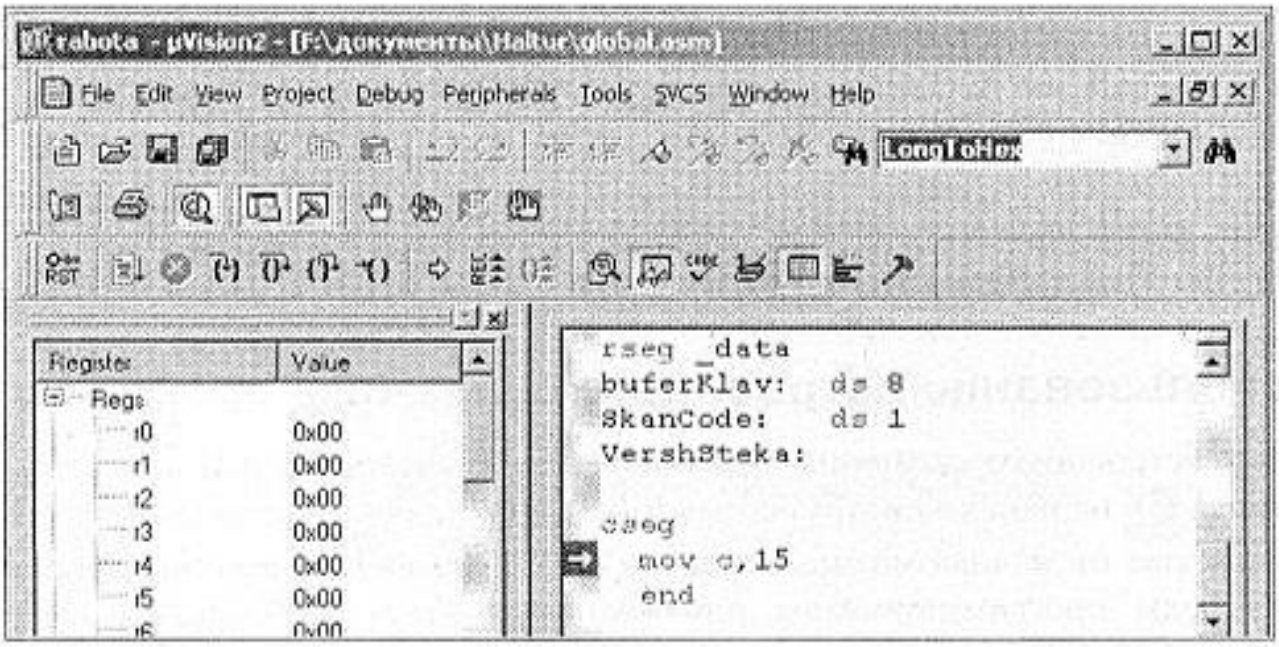


Рис. 24.21. Внешний вид интегрированной среды программирования в режиме отладки программ

При использовании встроенного отладчика программ для контроля переменных можно воспользоваться окном **Watch**. В большинстве случаев это намного выгоднее, чем использовать просмотр памяти данных. Переменные в

этом окне отображаются в том формате, в котором они были объявлены в программе. Для добавления переменной в окно **Watch** достаточно щелкнуть правой кнопкой мыши по переменной в окне отладчика программ, как это показано на рис. 24.22.

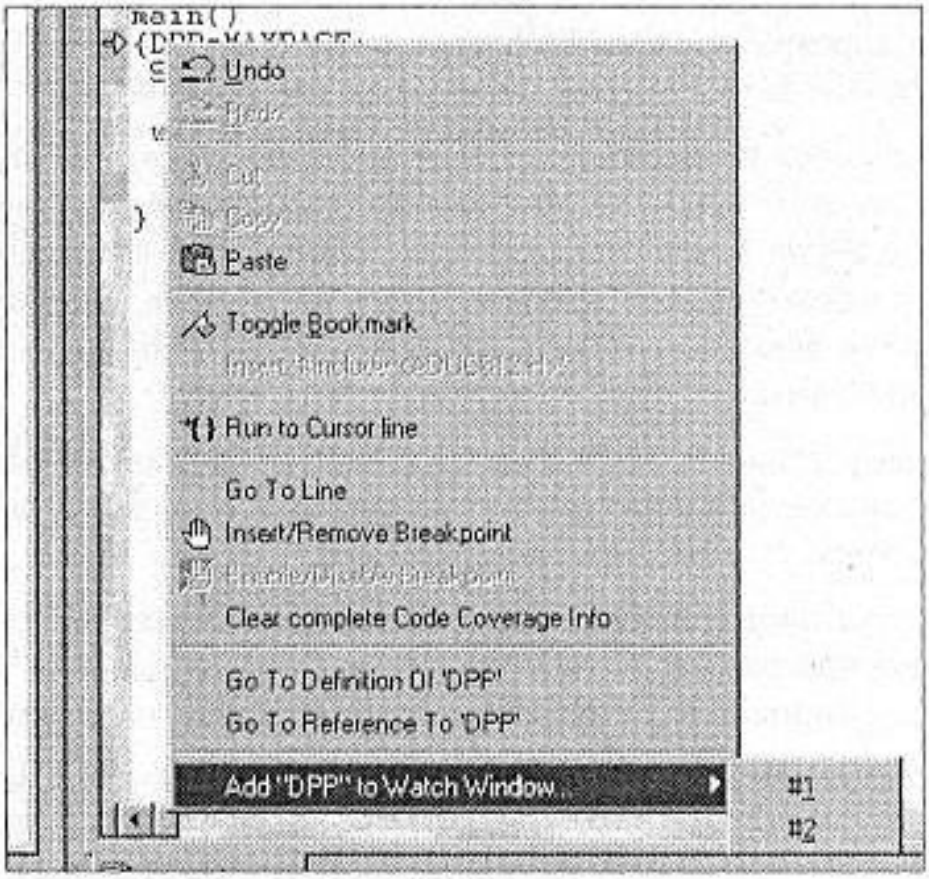


Рис. 24.22. Всплывающее меню для добавления переменной в окно просмотра **Watch**

Окно просмотра переменных содержит две закладки **Watch #1** и **Watch #2**. Это позволяет группировать переменные, по какому-либо признаку, например по отлаживаемым подпрограммам. При добавлении переменной вы выбираете номер окна просмотра переменных.

Кроме просмотра глобальных переменных, которые существуют на протяжении всей программы, окно просмотра переменных содержит закладку **Locals** (локальные переменные). Эта закладка позволяет отслеживать локальные переменные, которые существуют только внутри подпрограммы. Вводить имена локальных переменных в эту закладку не нужно. Они появляются в этой закладке автоматически, как только вы попадаете в подпрограмму, в которой используются локальные переменные.

При отладке программ на языке программирования ассемблер очень важно контролировать содержимое внутренних регистров микроконтроллера. Это позволяет сделать закладка **Regs** (регистры) в окне менеджера проектов, по-

казанная на рис. 24.21. В этом окне можно проконтролировать содержимое регистров текущего банка, указателя стека и программного счетчика, содержимое аккумуляторов A и B, а также состояние рабочих флагов микроконтроллера в регистре PSW.

Один оператор программы может быть выполнен нажатием клавиши <F11>. Если вызов подпрограммы рассматривается как один оператор, то пошаговая отладка программы осуществляется нажатием клавиши <F10>.

Использование точек останова позволяет пропускать уже отлаженную часть программы. Для того чтобы установить точку останова, можно воспользоваться кнопкой на панели файлов или главным или всплывающим меню. Перед тем как нажать кнопку установки точки останова, следует установить курсор на строку исходного текста программы, где необходимо остановить выполнение программы.

Точка останова устанавливается в местах, где необходимо проверить содержимое переменных или просто проконтролировать, передается ли управление данному оператору.

После того как установлены все необходимые точки останова, осуществляется выполнение программы в свободнобегущем режиме. Для этого можно воспользоваться кнопкой или нажать клавишу <F5> на клавиатуре.

Может возникнуть ситуация, что программа не передает управление ни одному из операторов, на которых установлены точки останова. В этом случае для прекращения выполнения программы следует воспользоваться кнопкой или нажать клавишу <Esc> на клавиатуре.

Точка останова может быть использована многократно. Иногда возникает необходимость однократно пропустить часть операторов. В этом случае можно воспользоваться кнопкой выполнения программы до курсора . При нажатии на эту кнопку программа будет выполняться до тех пор, пока управление не будет передано оператору, на котором находится курсор. Как только это произойдет, выполнение программы будет остановлено, и можно будет проконтролировать переменные и продолжить выполнение программы в пошаговом или свободнобегущем режиме.

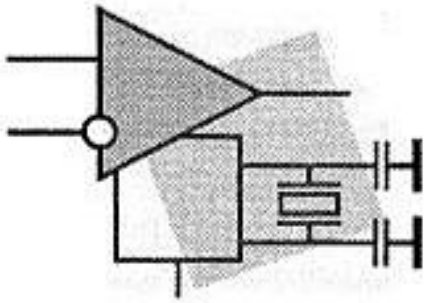
Итоги

В данной главе мы рассмотрели особенности работы с интегрированной средой программирования профессионального средства создания программного обеспечения фирмы keil. Как видно из этого обзора, работа по созданию программ для микроконтроллеров с применением интегрированной среды про-

граммирования значительно легче аналогичной работы бесплатными системами компиляции, а возможность одновременного применения двух языков программирования — языка C-51 и ассемблера, позволяет быстро создавать и, что особенно ценно, отлаживать эффективное программное обеспечение для микроконтроллеров.

Теперь, после того как мы обсудили особенности работы с интегрированной средой программирования keil-C, давайте рассмотрим, как можно при помощи языка программирования C управлять устройствами, подключенными к портам микроконтроллера.

ГЛАВА 25



Пример реализации микроконтроллерного устройства

Рассмотрим пример написания программы для микроконтроллера. Прежде всего, не нужно забывать, что программа не может существовать независимо от схемы устройства. Если при написании программы для универсального компьютера, такого как IBM PC, можно не задумываться о схеме, т. к. она стандартная, то перед написанием программы для микроконтроллера необходимо разработать принципиальную схему устройства, в состав которого будет входить этот микроконтроллер. Кроме того, обязательно следует разработать конструкцию устройства.

Выберем в качестве примера разработку часов на микроконтроллере. Пример разработки этого устройства приводится для того, чтобы можно было сравнить особенности реализации устройства методами схемотехники и программной реализации. Как и разработка любого устройства, разработка микропроцессорного устройства начинается с разработки структурной схемы.

Структурная схема часов

Так как мы уже разрабатывали структурную схему часов в *главе 10*, то можно воспользоваться уже готовой структурной схемой этого устройства. Для удобства дальнейшей работы, разработанная в *главе 10* структурная схема часов приведена на рис. 25.1.

Разработка принципиальной схемы

В соответствии с разработанной структурной схемой составим принципиальную схему часов. Анализируя структурную схему часов, можно выделить часть схемы, которая может быть выполнена с применением микроконтроллера. Это, несомненно, делители частоты, счетчик временных интервалов,

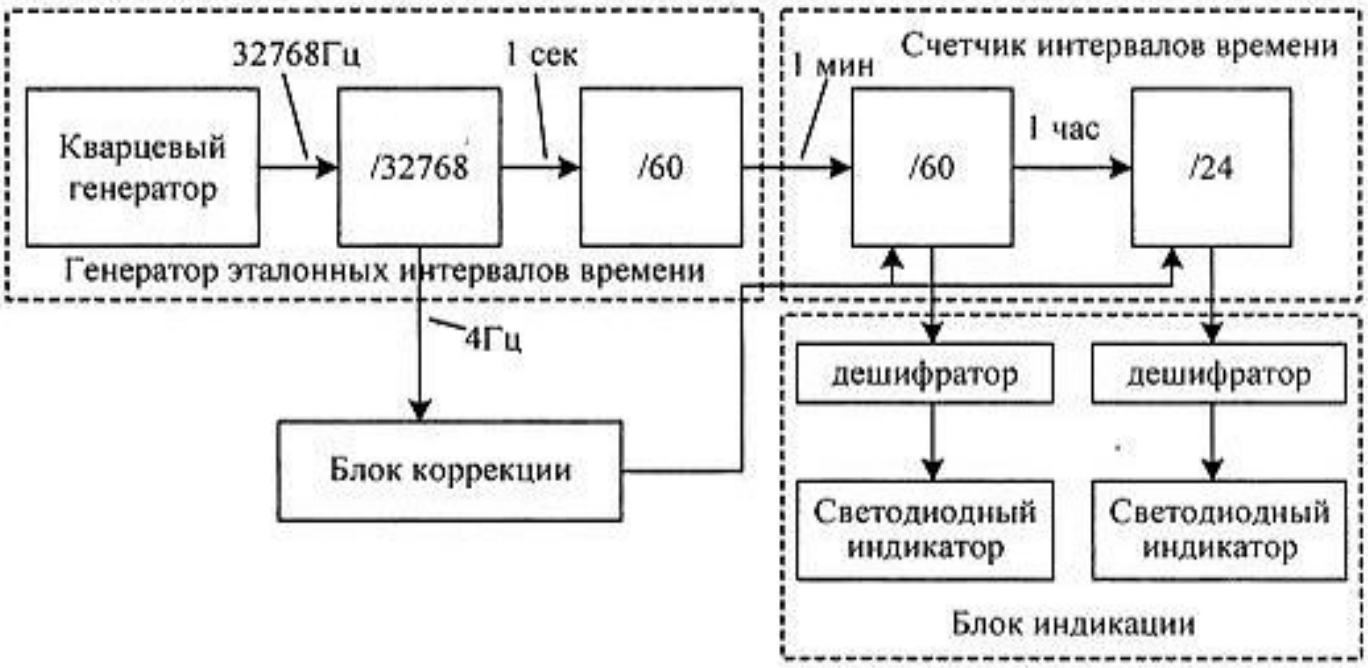


Рис. 25.1. Структурная схема часов

схема установки внутреннего состояния счетчика временных интервалов и дешифратор. Для реализации полной схемы часов микроконтроллер придется дополнить кварцевым резонатором, кнопками и светодиодными индикаторами.

Из блока индикации только дешифратор можно реализовать программно. Для индикации цифр воспользуемся семисегментными светодиодными индикаторами. Так как ток через светодиоды должен быть ограничен, то последовательно с сегментами необходимо поставить резисторы. Ток через сегменты светодиодных индикаторов ограничим значением 2 мА. Для современных светодиодных индикаторов с повышенной яркостью этого значения будет достаточно.

В качестве индикаторов применим те же самые индикаторы, что и в *главе 10* — суперъяркий светодиодный индикатор поверхностного монтажа с красным цветом свечения ACSA56-41SRWA-F01. Эти индикаторы производятся фирмой Kingbright. Схема подключения одного сегмента светодиодного индикатора к выходному каскаду микроконтроллера приведена на рис. 25.2.

Ток сегмента, величиной 2,14 мА, выбирается исходя из предельного тока, который способен обеспечить микроконтроллер. Выберем в качестве основного элемента часов микроконтроллер фирмы Atmel AT89S51-24AI, способный работать при напряжении питания +5 В. От вывода этой микросхемы можно получить ток до 15 мА, но суммарный ток через порт не должен превышать 15 мА. Поэтому:

$$I_c = \frac{I_p}{N_c} = \frac{15\ \text{мА}}{7} = 2{,}14\ \text{мА}$$

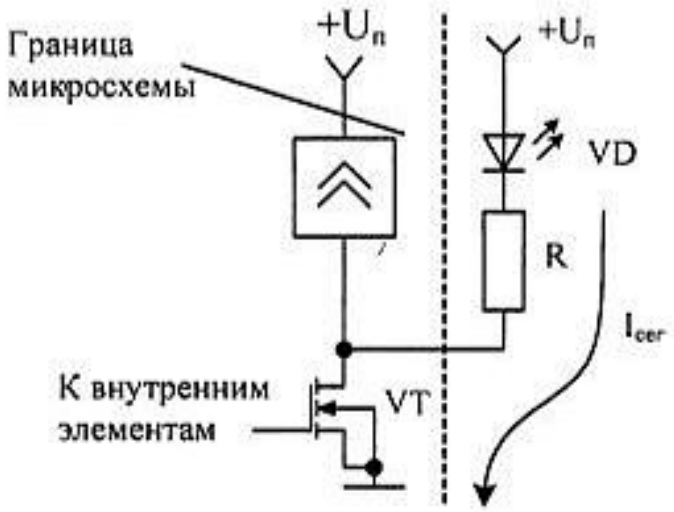


Рис. 25.2. Схема подключения одного сегмента светодиодного индикатора к выходному каскаду микроконтроллера

Резисторы рассчитываются по закону Ома. Напряжение на резисторах можно определить как напряжение питания схемы минус падение напряжения на светодиодах.

$$R = \frac{U_{\text{П}} - U_{\text{СВ}}}{I_{\text{С}}} = \frac{5\text{ В} - 2\text{ В}}{2{,}14\text{ мА}} = 1{,}4\text{ кОм}$$

Так как у микроконтроллера есть четыре восьмиразрядных параллельных порта, к ним можно подключить четыре семисегментных светодиодных индикатора. Четырех индикаторов достаточно для отображения информации о текущем времени. Это десятки и единицы минут и десятки и единицы секунд.

После подключения этих индикаторов остаются свободными четыре одиночных вывода параллельных портов. Эти выводы можно использовать для подключения трех кнопок блока коррекции. Как мы уже рассматривали ранее, кнопки необходимо включить между выводом порта и общим проводом устройства. Трех кнопок для управления часами будет вполне достаточно. Это кнопки коррекции часов, минут и установки счетчика секунд в нулевое состояние. Схема подключения кнопок коррекции времени к выводам микроконтроллера приведена на рис. 25.3.

Для правильной работы микроконтроллера при включении питания необходимо осуществить его сброс. Сброс микроконтроллера осуществляется путем подачи на вход RESET сигнала логической единицы. Так как микроконтроллер AT89S51-24AI работает в диапазоне напряжений от 4 до 5,5 В, то схема сброса должна удерживать вывод RESET в высоком состоянии до тех пор, пока источник питания формирует выходное напряжение ниже 4 В. Более того, по техническим условиям на выбранный микроконтроллер напряжение

питания должно оставаться выше 4 В до снятия сигнала RESET в течение, по крайней мере, 10 мс.

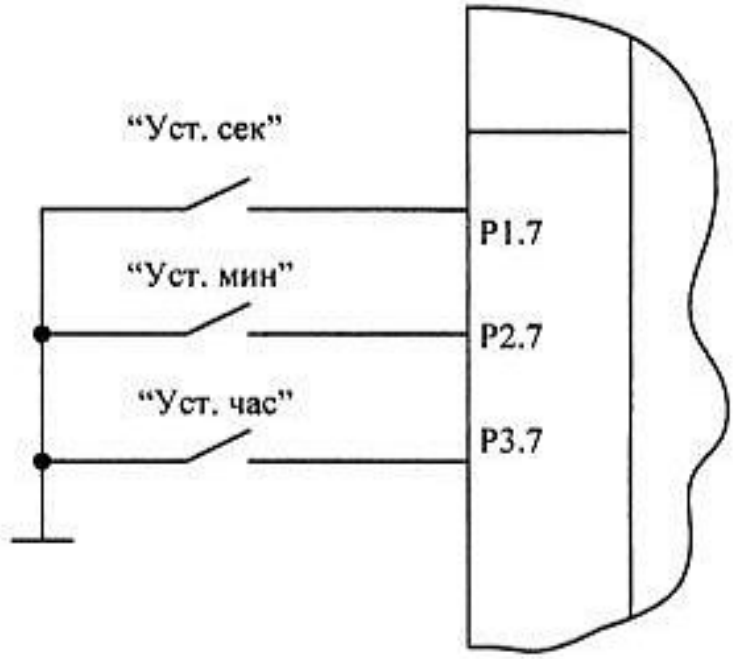


Рис. 25.3. Схема подключения кнопок коррекции времени к выводам микроконтроллера

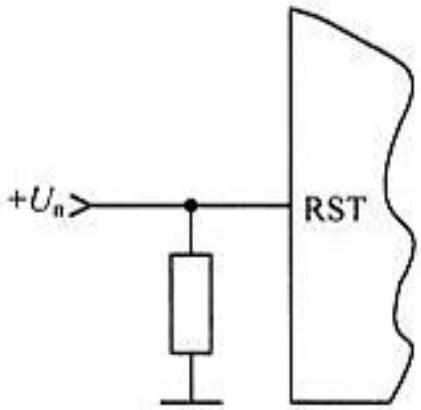


Рис. 25.4. Принципиальная схема устройства сброса, выполненного на конденсаторе

Удержать уровень логической единицы в течение 10 мс на входе RESET микроконтроллера при включении питания можно при помощи конденсатора, подключенного к микроконтроллеру в соответствии с рис. 25.4.

Но при работе от аккумулятора могут возникать спады напряжения (например, при понижении температуры), при которых появляются сбои в работе микросхемы микроконтроллера. Этих сбоев можно избежать, если удерживать микросхему в сброшенном состоянии при пониженном напряжении питания. Кроме того, сигнал сброса необходимо удерживать в течение 10 мс после восстановления рабочего уровня напряжения. Но это невозможно осуществить при помощи простейшей RC-цепи.

Для надежного запуска часов при включении питания дополним схему специальной микросхемой супервизора питания, которая обеспечит надежный сброс микроконтроллера при уменьшении напряжения питания ниже допустимого уровня. Воспользуемся микросхемой фирмы Analog devises ADM810. Напряжение 4 В контролирует микросхема ADM810M. Для уменьшения размеров выберем вариант микросхемы в корпусе SC70. Окончательно выбираем микросхему ADM810MAKSZ-REEL7.

На вход супервизора питания ADM810 подадим питающее напряжение микроконтроллера. При уменьшении напряжения на входе микросхемы ниже до-

пустимого уровня на выходе этой микросхемы появляется активный уровень, обеспечивая уверенный сброс микроконтроллера. Принципиальная схема подключения супервизора питания ADM810 к микроконтроллеру приведена на рис. 25.5

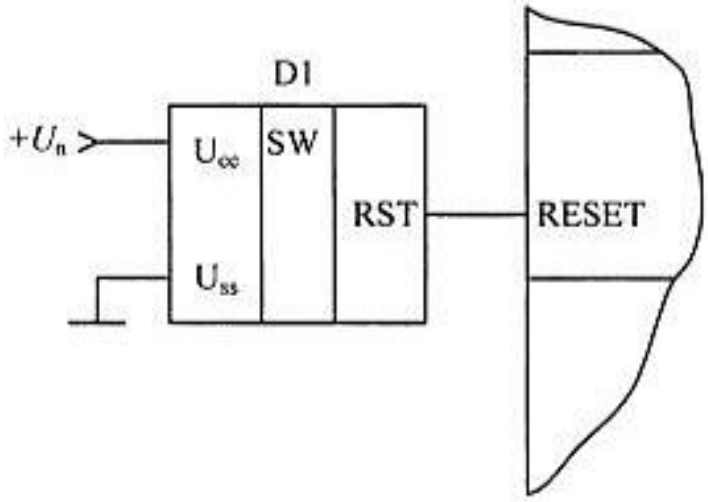


Рис. 25.5. Принципиальная схема устройства сброса, выполненного на супервизоре питания

Временные диаграммы напряжений, формируемых на выходе специализированного устройства сброса (супервизора питания) и простейшей RC-цепи, в зависимости от напряжения питания приведены на рис. 25.6.

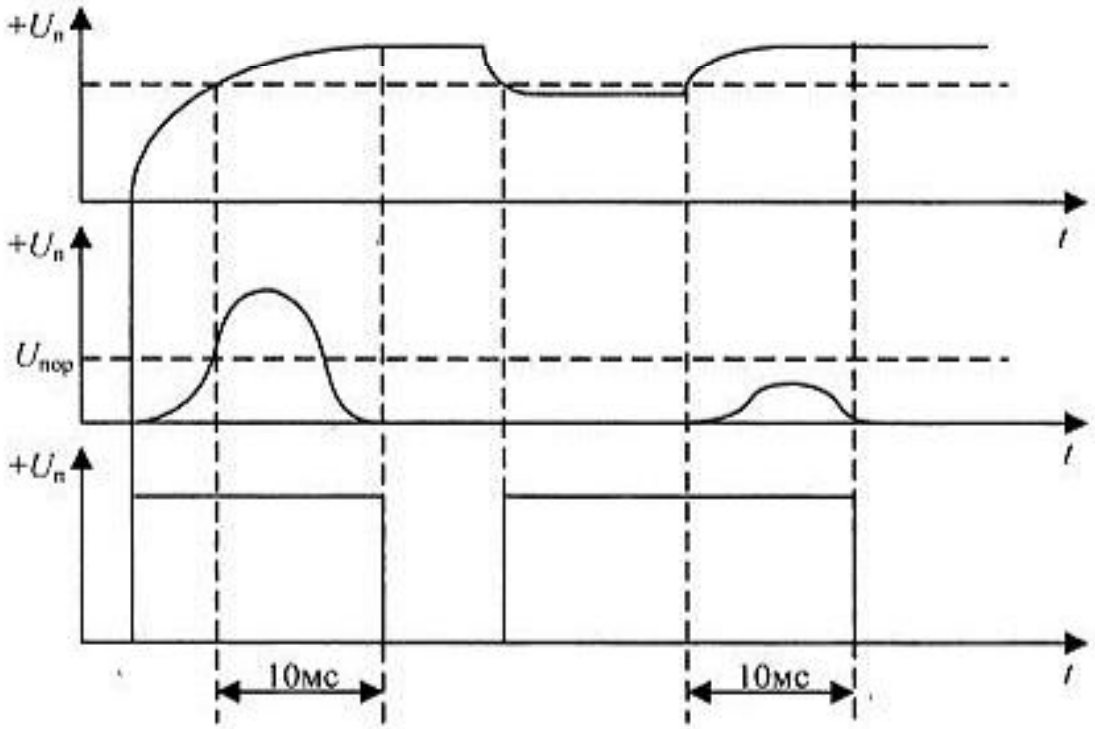


Рис. 25.6. Временные диаграммы напряжений, формируемых на выходе устройства сброса

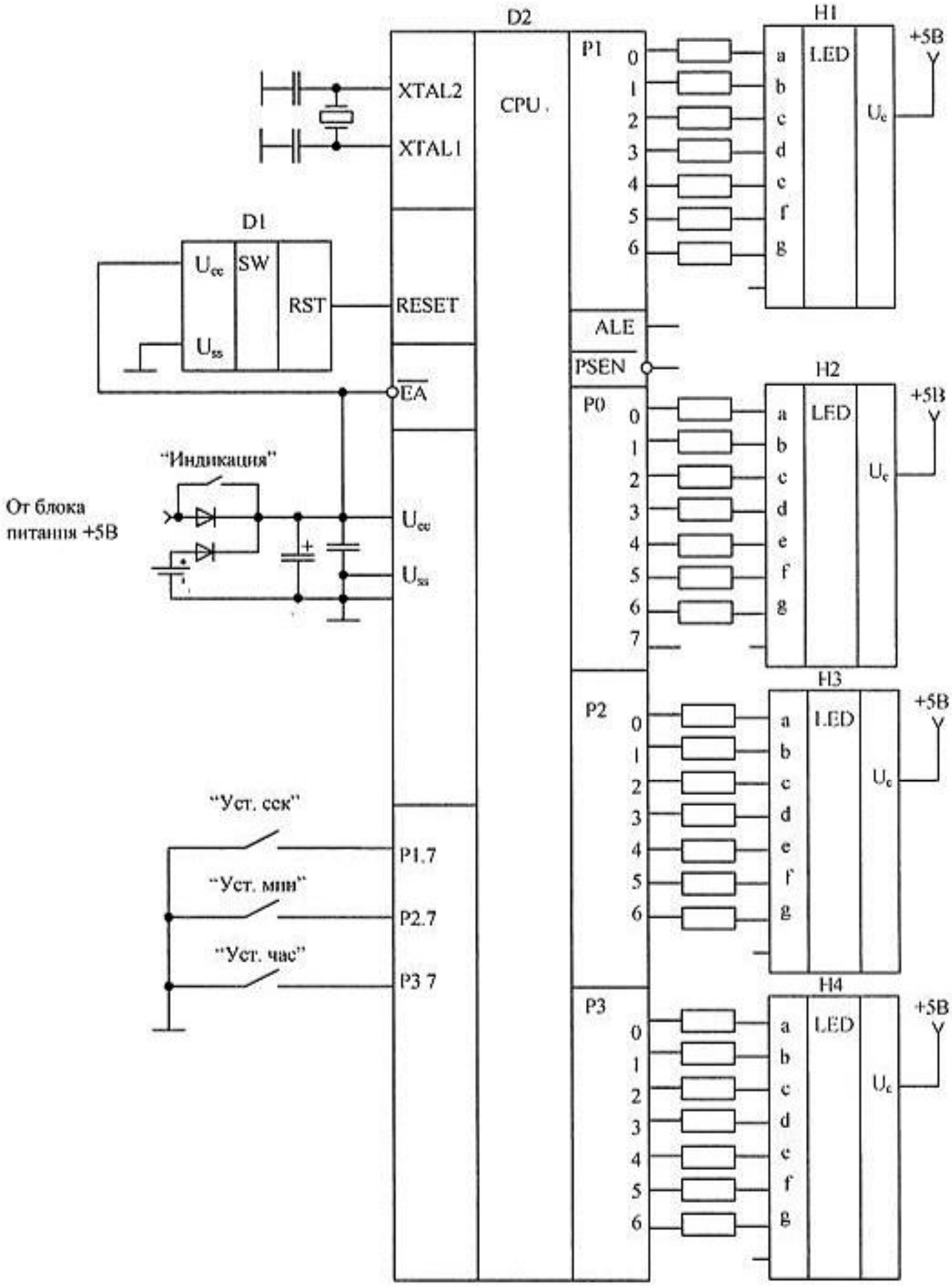


Рис. 25.7. Принципиальная схема часов

Рис. 25.6 иллюстрирует преимущества использования специализированного супервизора питания. По временным диаграммам видно, что если на вход конденсатора поступает полный перепад напряжения при включении питания, то на микросхему подается полноценный сигнал сброса. При восстановлении питания от некоторого пониженного значения приводит только к незначительному импульсу на входе RESET микроконтроллера, чего недостаточно для восстановления работоспособности микросхемы.

Кроме того, схема сброса, выполненная на конденсаторе, не обеспечивает сброс микроконтроллера при пониженном напряжении питания, что может привести к непредсказуемым последствиям.

Решающем же фактором выбора микросхемы супервизора питания в нашем случае является цена и габариты устройства. Стоимость электролитического конденсатора превышает стоимость микросхемы ADM810. Размеры корпуса SC70 тоже меньше размеров электролитического конденсатора.

Полная принципиальная схема часов приведена на рис. 25.7.

Схема часов с применением микроконтроллера получилась намного проще схемы, построенной на микросхемах средней интеграции.

Теперь определим ток, потребляемый схемой от источника питания. В технических данных на выбранную микросхему микроконтроллера, ток потребления микроконтроллера при работе с кварцевым резонатором 12 МГц составляет 6,5 мА. Так как микросхема является КМОП-устройством, то при уменьшении тактовой частоты ток потребления уменьшается пропорционально. При использовании часового кварцевого резонатора 32768 кГц потребляемый ток уменьшается до значения 17,7 мкА. При применении батареи емкостью 500 мА/час этого достаточно для работы в течение нескольких лет (при условии, если светодиодные индикаторы не будут использоваться).

Разработка программы устройства

Прежде всего, необходимо разделить аппаратную и программную часть схемы часов. В состав генератора эталонных интервалов времени входит делитель частоты, часть которого можно реализовать аппаратно при помощи делителя, входящего в состав одного из таймеров микроконтроллера.

Максимальный коэффициент деления на таймере можно получить равным 65 536. Прежде всего, необходимый делитель следует разделить на 12. Число 12 — это внутренний делитель таймера, который всегда присутствует в составе микросхемы и не может быть нами отключен.

$$32768/12 = 2730,7$$

Выберем коэффициент деления таймера равным 2731. Это число меньше максимально возможного числа, допустимого для шестнадцатиразрядного таймера ($2^{16} = 65536$). К сожалению, при этом за сутки уход часов составит 10,5 с, однако его можно скорректировать программным образом.

Теперь, после того как определилась принципиальная схема устройства, можно приступить к написанию программы для микроконтроллера, входящего в состав разработанной принципиальной схемы.

Как уже обсуждалось ранее, алгоритм и программу можно писать одновременно. Начнем написание программы с организации простейшей программы-монитора. Напомним, что в программе, написанной на языке программирования С-51, первой выполняется подпрограмма с именем main. Именно в этой подпрограмме и организуем этот монитор. Первоначальный вариант программы приведен в листинге 25.1.

Листинг 25.1. Исходный текст первоначального варианта программы

```
void main(void)
{while(1);   //Бесконечный цикл
}
```

Как видно из приведенного исходного текста программы, она пока ничего не делает. Однако в этой программе предусмотрен бесконечный цикл, который не позволит программе когда-либо завершиться до выключения питания устройства. Организация бесконечного цикла производится заданием условия цикла, которое никогда не может стать нулевым (ложным).

По циклу при подключенном к микроконтроллеру кварцевом резонаторе 32768 кГц программа будет проходить один раз за 732 мкс. В этом можно убедиться, посмотрев на дизассемблированный код программы, приведенный в листинге 25.2. Дизассемблированный листинг программы можно увидеть в окне отладчика среды программирования keil-C, нажав кнопку  .

Листинг 25.2. Дизассемблированный текст первоначального варианта программы

```
C:0x0000    020003   LJMP    C:0003
C:0x0003    787F     MOV     R0,#0x7F
C:0x0005    E4       CLR     A
C:0x0006    F6       MOV     @R0,A
C:0x0007    D8FD     DJNZ    R0,C:0006
C:0x0009    758107   MOV     SP(0x81),#0x07
C:0x000C    02000F   LJMP    main(C:000F)
     2: {while(1);   //Бесконечный цикл
C:0x000F    80FE     SJMP    main(C:000F)
```

Для того чтобы было легче ориентироваться в тексте программы, перед операторами ассемблерного кода в этом листинге приводится строка исходного текста программы. Например, в листинге 25.2 это предпоследняя строка.

Бесконечный цикл программы в этом листинге реализуется командой SJMP, расположенной по адресу 000F. Эта команда выполняется за два цикла микроконтроллера. Каждый же цикл микроконтроллера выполняется за двенадцать тактов, отсюда и была получена цифра 732 мкс. 2731 кГц/24=1,37 кГц. Период этой частоты и будет равен 732 микросекундам.

Кроме определения времени прохода программы по циклу, в листинге 25.2 можно убедиться, что программа обнуляет все содержимое памяти данных (команды, расположенные в ячейках памяти программ с адреса 0003 по 0008) и устанавливает первоначальное значение указателя стека (команда, расположенная по адресу 0009). То есть многие действия, которые приходится делать в программе, написанной на языке программирования ассемблер, в программе, написанной на языке программирования С-51, выполняются автоматически.

Разработка генератора секундных импульсов

Теперь реализуем проход по основному циклу программы один раз в секунду. Это будет эквивалентно реализации делителя частоты. При этом на выходе делителя будут присутствовать импульсы с периодом одна секунда. При дальнейшей реализации программы нам уже не придется беспокоиться о синхронизации времени работы программы с реальным временем.

Как уже упоминалось выше, это мы реализуем при помощи внутреннего таймера микроконтроллера. Использование аппаратного таймера позволит избавить процессор от скоростной работы делителя частоты и, таким образом, освободить его для реализации часов или других задач.

Выберем в качестве таймера, задающего необходимый интервал, таймер T0. Настройка таймера T0 нам потребуется только один раз (при включении часов). Поэтому настройку таймера поместим в блок инициализации микроконтроллера (часть программы, не входящая в бесконечный цикл).

Программа в основном режиме может ожидать переполнения таймера, опрашивая его флаг TF0, но при использовании прерываний от таймера можно значительно снизить потребление микроконтроллера. Экономия тока потребления реализуется при переведении микроконтроллера в спящий режим записью в нулевой разряд регистра PCON логической единицы. После вы-

полнения этой команды микроконтроллер может проснуться только после сброса микроконтроллера или возникновения прерывания.

Для реализации прохождения микроконтроллером по основному циклу один раз в секунду поместим команду `PCON=1` в конце тела бесконечного цикла, образованного оператором `while(1)`. В качестве источника прерываний будет служить таймер T0. Переход на начало бесконечного цикла будет осуществляться сразу после возврата из подпрограммы обслуживания прерывания.

Так как прерывания нам потребуются каждую секунду, то разрешение прерываний от таймера разместим в подпрограмме инициализации таймера. Разрешение прерываний от таймера производится записью единиц в биты регистра специального назначения IE. Разрешение прерываний от таймера производится битом ET0, а общее разрешение всех прерываний битом EA. Можно записать в эти биты единицу двумя командами присваивания `EA=1` и `ET0=1`, но вспомним, что эти биты находятся в одном регистре. Это означает, что мы можем обойтись одной командой.

Установим биты EA и ET0 в единичное состояние командой логического сложения `IE = IE | 0x82`, ведь шестнадцатеричное число 0x82 соответствует двоичному числу 10000010. Старший бит этого числа соответствует биту EA, а второй бит этого числа — биту ET0. Использование команды логического сложения вместо команды присваивания позволяет не изменять оставшиеся биты выбранного регистра вне зависимости от состояния этих бит. Язык программирования C позволяет записать эту команду короче: `IE |= 0x82`.

✧ *Обратите внимание!* Зачастую перевод чисел из одной системы счисления в другую достаточно затруднителен для человека, впервые столкнувшегося с программированием. Эту работу можно возложить на компилятор. Программа при этом получится длиннее, но зато можно описать комментариями каждый бит многоразрядного числа. Дело в том, что все операции над константами компилятор выполняет в процессе трансляции исходного текста программы, а в загрузочный модуль размещает результирующее число. Поэтому команда IE|=0x82 может выглядеть в следующем виде: IE|=(1<<7)|(1<<1). Здесь мы просто указываем нужные нам номера битов. Выражение в первых скобках просто означает выделение седьмого бита (единица, сдвинутая на семь позиций влево), а выражение во вторых скобках означает выделение первого бита (единица, сдвинутая на одну позицию влево). Именно такую форму команды мы и выберем.

Теперь микроконтроллер каждую секунду будет не только "просыпаться", но и передавать управление на вектор прерывания таймера T0. Этот вектор находится по адресу 0Bh. Подпрограмму повторной записи числа 50 мс в таймер нужно поместить точно на этот адрес. Размещение подпрограммы на век-

торе прерывания мы осуществим зарезервированным словом `interrupt`. Вектор прерывания конкретно таймера T0 выбирается числом 1.

Исходный текст варианта программы с секундным циклом приведен в листинге 25.3. Этот вариант программы можно тщательно проверить на работоспособность на программном эмуляторе или непосредственно на собранном устройстве. При измерении тока потребления микроконтроллера при помощи осциллографа, на экране этого осциллографа отчетливо будут видны импульсы повышенного тока с периодом одна секунда. Эти импульсы будут возникать в момент прохода программы по основному циклу.

Листинг 25.3. Исходный текст программы, обеспечивающей проход по циклу один раз за *50 мс*

```
#include<reg51.h>

void Timer0(void) interrupt 1  //ВЕКТОР ПРЕРЫВАНИЯ ТАЙМЕРА T0
{TH0=-2731/256;        //Загрузить старший байт таймера
 TL0=-2731;            //Загрузить младший байт таймера
}

/*********************************************************************
Подпрограмма настройки таймера T0 на 50 мс режим работы
*********************************************************************/
void Timer0_Init(void)
{TMOD=(0<<7)|    //Запретить управление таймером T1 от ножки INT1
      (0<<6)|    //Синхронизировать таймер T1 от внутреннего генератора
      (0<<4)|    //Перевести таймер T1 в тринадцатиразрядный режим работы
      (0<<3)|    //Запретить управление таймером T0 от ножки INT0
      (0<<2)|    //Синхронизировать таймер T0 от внутреннего генератора
       1;        //Перевести таймер T0 в первый режим работы

//Настройка таймера на генерацию 50-миллисекундного интервала времени
  TH0=-2731/256;     //Загрузить старший байт таймера
  TL0=-2731;         //Загрузить младший байт таймера

  IE |=(1<<7)|       //Разрешить прерывания в микроконтроллере
       (1<<1);       //Разрешить прерывания конкретно от таймера T0
  TR0=1;             //Включить таймер T0
}                    //и вернуться в основную программу

void main(void)
{//------------ИНИЦИАЛИЗАЦИЯ МИКРОКОНТРОЛЛЕРА-----------------------
 Timer0_Init();//Настроить таймер T0 на прерывания с периодом 50 мс
```

```
//-------------ОСНОВНАЯ ПРОГРАММА МИКРОКОНТРОЛЛЕРА-------------------
 while(1)        //Бесконечный цикл
 {PCON=1;        //Перевести микроконтроллер в пониженный режим
 }               //потребления тока и подождать переполнения таймера
}
```

Программа, приведенная в листинге 25.3, реализует генератор эталонных интервалов времени с периодом 1 с. Теперь в состав программы введем подпрограмму, реализующую часы. Новый вариант программы приведен в листинге 25.4.

Листинг 25.4. Исходный текст программы, обеспечивающей вызов подпрограммы часов один раз в секунду

```
#include<reg51.h>
#include "clock.h"

...

/*************************************************************************
Подпрограмма реализации часов
*************************************************************************/
void Clock(void)
{
}

/*************************************************************************
ВЫПОЛНЕНИЕ ПРОГРАММЫ НАЧИНАЕТСЯ ОТСЮДА
*************************************************************************/
char Delit=20;    //8-разрядная ячейка памяти для делителя частоты (1с)
void main(void)
{//-------------ИНИЦИАЛИЗАЦИЯ МИКРОКОНТРОЛЛЕРА-------------------------
 Timer0_Init();  //Настроить таймер T0 на прерывания с периодом 50 мс
//-------------ОСНОВНАЯ ПРОГРАММА МИКРОКОНТРОЛЛЕРА-------------------
 while(1)        //Бесконечный цикл
 {Clock();       //Вызвать подпрограмму счетчика секунд.
  PCON=1;        //Перевести микроконтроллер в пониженный режим
 }               //потребления тока и подождать переполнения таймера
}
```

В этом листинге для большей наглядности и экономии места не приведены подпрограммы инициализации и обслуживания прерывания таймера, но они не отличаются от приведенных в предыдущем варианте программы. Подпрограмма реализации часов `Clock` вызывается точно один раз в секунду.

Для наглядности имя этой подпрограммы было выбрано Clock, что на английском языке означает часы. В подпрограмме Clock пока никаких действий не выполняется, поэтому подпрограмму можно считать подпрограммой-заглушкой, необходимой только для трансляции исходного текста программы и проверки работы генератора секундных импульсов.

При отладке программы на реальной схеме в этой подпрограмме можно разместить оператор, инвертирующий состояние одного из светодиодов, и тем самым убедиться в том, что программа работает правильно.

Разработка подпрограммы часов

Теперь можно приступить к реализации подпрограммы часов. При написании этой подпрограммы мы уверены, что попадаем в нее один раз в секунду, поэтому ее можно рассматривать отдельно и независимо от остальной программы. Более того, эту подпрограмму можно разместить в отдельном файле и подключить ее к проекту. Так мы и сделаем.

В этой подпрограмме часов мы реализуем счетчик секунд, но для его работы, естественно, потребуется еще одна переменная. Содержимое файла с первым вариантом подпрограммы часов приведено в листинге 25.5.

Листинг 25.5. Исходный текст подпрограммы, реализующей счетчик секунд

```
unsigned char SEC=60;    //Переменная счетчика секунд

/*********************************************************************
Подпрограмма реализации часов
*********************************************************************/
void Clock(void)
{if(--SEC==0) //Если прошла одна минута, то
    SEC=60;       //настроить делитель на коэффициент деления 60
}
```

Переменная SEC объявляется до первого использования, в самом начале файла. Именно в этой переменной и будет осуществляться подсчет количества прошедших секунд. Как и в предыдущем случае, будет достаточно однобайтовой переменной. Первоначальное значение этой переменной присваивается сразу же при ее объявлении, а переключение счетчика секунд организуется подобно работе предварительного делителя, рассмотренного в предыдущем варианте программы.

Теперь усложним подпрограмму часов и введем в их состав счетчик минут. Для того чтобы при этом подпрограмма была более наглядной, немного из-

меним организацию счетчика секунд. При разработке счетчика минут обратим внимание, что его содержимое нужно отображать на светодиодном индикаторе. При этом показания минут должны увеличиваться. Поэтому счетчик минут выполним с помощью команды увеличения содержимого переменной на единицу. Новый вариант подпрограммы часов приведен в листинге 25.6.

Листинг 25.6. Исходный текст подпрограммы, реализующей счетчики секунд и минут

```
unsigned char SEC=60;    //Переменная счетчика секунд
unsigned char MIN=0;     //Переменная счетчика минут

/*********************************************************************
Подпрограмма реализации часов
*********************************************************************/
void Clock(void)
{if(--SEC!=0)  //Если одна минута еще не прошла, то
    return;       //вернуться из подпрограммы в основной цикл
 SEC=60;        //Настроить счетчик секунд на коэффициент деления 60
 if(++MIN!=60) //Если один час еще не прошел, то
    return;     //вернуться из подпрограммы в основной цикл
 MIN=0;         //Обнулить показания счетчика минут
}
```

В приведенном исходном тексте подпрограммы для увеличения показания счетчика минут использована команда суммирования. В результате количество машинных команд, требующихся для реализации условного оператора, резко увеличится. В системах реального времени очень важно применять команды, занимающие минимальное время. В листинге 25.6 для реализации счетчиков секунд и минут используются различные операторы языка С. Давайте проанализируем их реализацию в машинных кодах. Для этого воспользуемся дизассемблированным текстом подпрограммы, приведенным в листинге 25.7.

Листинг 25.7. Дизассемблированный текст подпрограммы, реализующей счетчики секунд и минут

```
     7: void Clock(void)
     8: {if(--SEC!=0) //Если одна минута еще не прошла, то
     9:     return;       //вернуться из подпрограммы в основной цикл
C:0x0099    D5080D   DJNZ     SEC(0x08),C:00A9
    10:  SEC=60;      //Настроить счетчик секунд на коэффициент деления 60
```

```
C:0x009C    75083C   MOV      SEC(0x08),#0x3C
    11:  if(++MIN!=60) //Если один час еще не прошел, то
    12:      return;       //вернуться из подпрограммы в основной цикл
C:0x009F    0509     INC      MIN(0x09)
C:0x00A1    E509     MOV      A,MIN(0x09)
C:0x00A3    B43C03   CJNE     A,#0x3C,C:00A9
    13:  MIN=0;        //Обнулить показания счетчика минут
C:0x00A6    E4       CLR      A
C:0x00A7    F509     MOV      MIN(0x09),A
    14: }
C:0x00A9    22       RET
    11: void Timer0_Init(void)
```

Мы видим, что одинаковые по внешнему виду команды условного оператора приводят к различному машинному коду, отличающемуся в два раза, как по объему, так и по быстродействию. Именно поэтому использовать команду увеличения стоит только тогда, когда это определяется какими-либо причинами, как, например, в нашем случае необходимостью отображать показания счетчика на индикаторе.

Теперь добавим в программу счетчик часов. Так как показания часов тоже необходимо отображать на индикаторе, то реализуем его подобно счетчику минут. Новый вариант подпрограммы часов приведен в листинге 25.8.

Листинг 25.8. Исходный текст подпрограммы, реализующей счетчики секунд, минут и часов

```
unsigned char SEC=60;    //Переменная счетчика секунд
unsigned char MIN=0;     //Переменная счетчика минут
unsigned char Chas=0;    //Переменная счетчика часов

/*****************************************************************
Подпрограмма реализации часов
*****************************************************************/
void Clock(void)
{if(--SEC!=0)  //Если одна минута еще не прошла, то
    return;       //вернуться из подпрограммы в основной цикл
 SEC=60;       //Настроить счетчик секунд на коэффициент деления 60
 if(++MIN!=60) //Если один час еще не прошел, то
    return;       //вернуться из подпрограммы в основной цикл
 MIN=0;        //Обнулить показания счетчика минут
 if(++Chas!=24)//Если сутки еще не прошли, то
    return;       //вернуться из подпрограммы в основной цикл
 Chas=0;       //Обнулить показания счетчика часов
}
```

Счетчик часов реализуется подобно счетчику минут. Единственное отличие заключается в том, что в сутках двадцать четыре часа, поэтому счетчик часов будет обнуляться по достижении этого значения.

Разработка подпрограммы индикации

Теперь можно заняться следующим блоком часов — блоком индикации. При разработке часов мы получили устройство, в разных частях которого действуют различные частоты. Поэтому первоначально необходимо определить частоту, с которой следует производить обновление информации на светодиодных индикаторах.

Учитывая, что изменение показаний часов при установке времени может происходить несколько раз в секунду, выберем точку подключения блока индикации 50 мс, т. е. подпрограмму индикации необходимо разместить внутри основного цикла программы.

Индикацию внутреннего состояния часов легче всего произвести в составе отдельной подпрограммы. Это, как уже говорилось ранее, позволит разделить задачи и решать их независимо. На данный момент уже реализованы часы. Блок индикации может непосредственно обращаться к переменным часов для считывания хранящейся в них информации. Исходный текст основной программы с циклом, в котором производится вызов подпрограммы индикации, приведен в листинге 25.9.

Листинг 25.9. Исходный текст основной программы

```
#include <reg51.h>
#include "clock.h"
#include "Indic.h"

...

/*************************************************************************
ВЫПОЛНЕНИЕ ПРОГРАММЫ НАЧИНАЕТСЯ ОТСЮДА
*************************************************************************/
char Delit=20;    //8-разрядная ячейка памяти для делителя частоты (1с)
void main(void)
{//-------------ИНИЦИАЛИЗАЦИЯ МИКРОКОНТРОЛЛЕРА-----------------------
 Timer0_Init();  //Настроить таймер Т0 на прерывания с периодом 50 мс
//-------------ОСНОВНАЯ ПРОГРАММА МИКРОКОНТРОЛЛЕРА-------------------
 while(1)        //Бесконечный цикл
 {Indic();       //Произвести индикацию состояния часов на светодиодных
  Clock();       //индикаторах и вызвать подпрограмму счетчика секунд.
```

```
    PCON=1;          //Перевести микроконтроллер в пониженный режим
  )                  //потребления тока и подождать переполнения таймера
}
```

В приведенном исходном тексте не отображены подпрограммы инициализации таймера и реализации часов (они не показаны для уменьшения размера листинга).

Теперь можно заняться подпрограммой блока индикации. На принципиальной схеме часов не указано, что и на каком индикаторе должно отображаться. Зададимся, что на индикаторе, подключенном к порту P1, будут отображаться единицы минут. На индикаторе, подключенном к порту P2, будут отображаться десятки минут. На индикаторе, подключенном к порту P0, будут отображаться единицы часов. На индикаторе, подключенном к порту P3, будут отображаться десятки часов.

Какой индикатор что отображает, определяется конструкцией часов. При необходимости изменить порядок подключения индикаторов можно просто заменить в программе имена параллельных портов, к которым подключены индикаторы, и перетранслировать программу.

Для декодирования двоично-десятичного кода предусмотрим подпрограмму-функцию decod. При этом декодируемое двоично-десятичное число будет передаваться в подпрограмму через ее параметр. Полученный в результате работы подпрограммы семисегментный код будет возвращаться самой функцией.

Точно так же, как и в случае с часами, для индикации создадим отдельный файл (модуль). Исходный текст модуля индикации приведен в листинге 25.10.

Листинг 25.10. Исходный текст модуля индикации

```
#include<reg51.h>
#include "clock.h"

/*************************************************************************
Подпрограмма семисегментного дешифратора
*************************************************************************/
char Decod(char inp)
{
}

/*************************************************************************
Подпрограмма блока индикации
*************************************************************************/
void Indic(void)
```

```
{char tmp;
 tmp=MIN%10;     //Считать содержимое счетчика минут и выделить младшую
                 //тетраду (единицы минут)
 P1=Decod(tmp); //Преобразовать ее в семисегментный код и передать через
                 //порт P1 на индикатор

 tmp=MIN/10;     //Считать содержимое счетчика минут и выделить старшую
                 //тетраду (десятки минут)
 P2=Decod(tmp); //Преобразовать ее в семисегментный код и передать через
                 //порт P2 на индикатор

 tmp=Chas%10;    //Считать содержимое счетчика часов и выделить младшую
                 //тетраду (единицы часов)
 P0=Decod(tmp); //Преобразовать ее в семисегментный код и передать через
                 //порт P0 на индикатор

 tmp=Chas/10;    //Считать содержимое счетчика минут и выделить старшую
                 //тетраду (десятки часов)
 P3=Decod(tmp); //Преобразовать ее в семисегментный код и передать через
                 //порт P3 на индикатор
}
```

Выделение единиц минут осуществляется при помощи операции взятия остатка от деления на константу 10. Эта операция выполняется при использовании символа '%'. При этом осуществляется целочисленное беззнаковое деление содержимого счетчика минут на число 10. В результате в остатке мы получим единицы минут. Это число не будет превышать числа 10, а значит, для его представления достаточно четырех бит (тетрады).

Десятки минут выделяются аналогичным образом. Отличие заключается в том, что вместо операции взятия остатка используется обычное целочисленное беззнаковое деление.

Для того чтобы убедиться в правильности написания программы, можно воспользоваться дизассемблированным текстом подпрограммы индикации, приведенным в листинге 25.11.

Листинг 25.11. Дизассемблированный текст подпрограммы индикации

```
    14: void Indic(void)
    15: {char tmp;
    16:  tmp=MIN%10;     //Считать содержимое счетчика минут и выделить
                         //младшую тетраду
C:0x0099    E50A     MOV      A,MIN(0x0A)
C:0x009B    75F00A   MOV      B(0xF0),#MIN(0x0A)
```

```
C:0x009E    84        DIV       AB
C:0x009F    AFF0      MOV       R7,B(0xF0)
    17:  P1=Decod(tmp); //Преобразовать ее в семисегментный код и
    18:                 // передать через порт P1 на индикатор
    19:                 }
C:0x00A1    110A      ACALL     Decod(C:000A)
C:0x00A3    8F90      MOV       P1(0x90),R7
    20:  tmp=MIN/10;    //Считать содержимое счетчика минут и выделить
                          старшую тетраду
C:0x00A5    E50A      MOV       A,MIN(0x0A)
C:0x00A7    75F00A    MOV       B(0xF0),#MIN(0x0A)
C:0x00AA    84        DIV       AB
C:0x00AB    FF        MOV       R7,A
    21:  P2=Decod(tmp); //Преобразовать ее в семисегментный код и
    22:                 // передать через порт P2 на индикатор
    23:
C:0x00AC    110A      ACALL     Decod(C:000A)
C:0x00AE    F5A0      MOV       P2(0xA0),A
```

Как видно из этого листинга, деление осуществляется встроенной командой целочисленного деления `DIV AB`. В качестве примера неправильного использования данных операций воспользуемся листингом 25.12, где переменная `MIN` объявлена не как `unsigned char`, а как просто `char`.

Листинг 25.12. Дизассемблированный текст подпрограммы индикации с переменной типа MIN char

```
    14: void Indic(void)
    15: (char tmp;
    16:  tmp=MIN%10;    //Считать содержимое счетчика минут и выделить
                          младшую тетраду
C:0x0099    E50A      MOV       A,MIN(0x0A)
C:0x009B    75F00A    MOV       B(0xF0),#MIN(0x0A)
C:0x009E    11CA      ACALL     C?SCDIV(C:00CA)
C:0x00A0    AFF0      MOV       R7,B(0xF0)
    17:  P1=Decod(tmp); //Преобразовать ее в семисегментный код и
    18:                 // передать через порт P1 на индикатор
    19:
C:0x00A2    110A      ACALL     Decod(C:000A)
C:0x00A4    8F90      MOV       P1(0x90),R7
    20:  tmp=MIN/10;    //Считать содержимое счетчика минут и выделить
                          старшую тетраду
C:0x00A6    E50A      MOV       A,MIN(0x0A)
C:0x00A8    75F00A    MOV       B(0xF0),#MIN(0x0A)
```

```
C:0x00AB    11CA      ACALL     C?SCDIV(C:00CA)
C:0x00AD    FF        MOV       R7,A
    21:  P2=Decod(tmp); //Преобразовать ее в семисегментный код и
    22:                 // передать через порт P2 на индикатор
    23:
C:0x00AE    110A      ACALL     Decod(C:000A)
C:0x00B0    F5A0      MOV       P2(0xA0),A
```

Как видно из приведенного листинга, в этой программе вместо машинной команды `DIV AB` используется подпрограмма знакового деления `C?SCDIV`, что вовсе не входило в наши планы, т. к. вряд ли эта подпрограмма состоит из одной машинной команды. Этот пример показывает насколько важен выбор типа переменной для каждого конкретного случая.

Теперь необходимо вспомнить об индикации, ведь именно для этого мы и начали писать подпрограмму. Собственно говоря, индикация осуществляется при копировании полученного из подпрограммы декодирования семисегментного кода в параллельный порт. В рассмотренном листинге подпрограммы индикации мы осуществили индикацию всех четырех цифр времени: десятков часов, часов, десятков минут и минут.

У нас осталась неразработанной подпрограмма семисегментного дешифратора, которая преобразует двоично-десятичное значение, содержащееся в локальной переменной `tmp`, в семисегментный код, пригодный для отображения на светодиодном индикаторе. Приступим к разработке этой подпрограммы.

Разработка подпрограммы семисегментного дешифратора

Рассмотрим подпрограмму преобразования двоично-десятичного кода в семисегментный. Декодирование осуществляется за счет таблицы перекодировки. Количество строк в таблице соответствует количеству комбинаций двоично-десятичного кода, т. е. равно десяти. Нули и единицы в получаемом коде определяются свечением сегментов светодиодного индикатора.

Для того чтобы определить, при каком логическом уровне на выводе порта будет светиться светодиодный сегмент, необходимо воспользоваться принципиальной схемой. Для нашего устройства принципиальная схема приведена на рис. 25.1. Для того чтобы в этой схеме засветился светодиодный сегмент, нужно пропустить через него ток. Для этого необходимо открыть транзистор. Транзистор открывается при протекании базового тока, а это произойдет, если на выводе порта будет присутствовать логическая единица.

Теперь необходимо поставить в соответствие выводы порта и название сегментов семисегментного индикатора. В приведенной схеме номер вывода порта соответствует сегменту индикатора. Для того чтобы не запутаться, в подпрограмме используется двоичная запись числа, а над каждым битом числа приводится имя сегмента, соответствующее этому биту. Исходный текст подпрограммы декодирования двоично-десятичного числа приведен в листинге 25.13.

Листинг 25.13. Исходный текст подпрограммы семисегментного дешифратора

```
char code TabSemiSeg[]= //  abcdefg
 {0xfe,             // 11111110b - символ '0'       a
  0xbd,             // 10110000b - символ '1'     -------
  0xed,             // 11101101b - символ '2'    |       |
  0xf9,             // 11111001b - символ '3'  f|       | b
  0xb3,             // 10110011b - символ '4'    |   g   |
  0xdb,             // 11011011b - символ '5'     -------
  0xdf,             // 11011111b - символ '6'    |       |
  0xf0,             // 11110000b - символ '7'  e|       |c
  0xff,             // 11111111b - символ '8'    |   d   |
  0xfb};            // 11111011b - символ '9'     -------

/*****************************************************************************
Подпрограмма семисегментного дешифратора
*****************************************************************************/
char Decod(unsigned char inp)
{return TabSemiSeg[inp];
}
```

Здесь тоже очень важно не перепутать типы данных. Если их сменить, по сравнению с приведенными в листинге, то вы сразу же получите намного больший программный код. Желающие могут поэкспериментировать, меняя типы переменных и просматривая в листинге или в дизассемблированном тексте получающийся после этого программный код.

Разработка блока коррекции часов

Теперь единственным блоком часов, который необходимо реализовать, остался блок коррекции показаний часов. В состав этого блока входят три кнопки. Обычно при нажатии или отпускании кнопки возникает переходный процесс, который называется дребезгом контактов. Начинающие разработчики аппаратуры используют различные методы борьбы с этим явлением:

от применения специальных схемотехнических решений до повторного опроса кнопок в течение некоторого времени.

В *главе 21* мы рассматривали механизм возникновения явления дребезга контактов и способы борьбы с этим явлением. Если мы будем опрашивать состояние контактов с периодом, превышающим максимальную длительность дребезга контактов, то мы даже не заметим, что контакты замыкались или размыкались в промежутке между опросами контактов. В нашем примере один проход по циклу программ соответствует 50 мс, что заведомо превышает время дребезга контактов. А это значит, что в разработанной нами программе мы можем не беспокоиться о подавлении дребезга контактов. Дребезг контактов приводит только к неопределенности определения времени нажатия кнопки. Эта неопределенность не превышает времени реакции системы, равной 50 мс.

Новый вариант программы с включенной в цикл подпрограммой опроса кнопок блока коррекции времени приведен в листинге 25.14.

Листинг 25.14. Исходный текст основной программы с подпрограммой опроса кнопок коррекции времени

```
/*************************************************************************
ВЫПОЛНЕНИЕ ПРОГРАММЫ НАЧИНАЕТСЯ ОТСЮДА
*************************************************************************/
char Delit=20;    //8-разрядная ячейка памяти для делителя частоты (1с)
void main(void)
{//------------ИНИЦИАЛИЗАЦИЯ МИКРОКОНТРОЛЛЕРА-----------------------
 Timer0_Init();  //Настроить таймер Т0 на прерывания с периодом 50 мс
//------------ОСНОВНАЯ ПРОГРАММА МИКРОКОНТРОЛЛЕРА-------------------
 while(1)        //Бесконечный цикл
 {
  OprosKnop();   //Опросить кнопки

  Indic();       //Произвести индикацию состояния часов на светодиодных
                 //индикаторах

  if(--Delit==0) //Если прошла одна секунда, то
    {Delit=20;      //настроить делитель на коэффициент деления 20
     Clock();       //и вызвать подпрограмму счетчика секунд.
    }
  PCON=1;        //Перевести микроконтроллер в пониженный режим
 }               //потребления тока и подождать переполнения таймера
}
```

Теперь можно заняться реализацией подпрограммы опроса кнопок. Для хранения состояния кнопок в этой программе пришлось ввести новую переменную. Ее задача состоит в копировании логических уровней с выводов порта во внутреннюю переменную микроконтроллера SostKn. *Так как это будет происходить быстро и с постоянной задержкой по времени, то время взятия отсчетов (а также время вывода на индикаторы) будет строго постоянным.* Исходный текст подпрограммы опроса кнопок приведен в листинге 25.15.

Листинг 25.15. Исходный текст подпрограммы опроса кнопок

```
sbit P1_7 = 0x97;
sbit P2_7 = 0xa7;
sbit P3_7 = 0xb7;

unsigned char SostKn=0;

/*************************************************************************
Подпрограмма опроса кнопок
*************************************************************************/
void OprosKnop(void)
{if(P1_7)
     SostKn|=1;
   else
     SostKn&=~1;
 if(P2_7)
     SostKn|=2;
   else
     SostKn&=~2;
 if(P3_7)
     SostKn|=4;
   else
     SostKn&=~4;
}
```

В этой подпрограмме кнопки, разнесенные по трем различным портам, сводятся в одну переменную SostKn. Для опроса состояния кнопок установки времени используется условный оператор if. При помощи этого оператора опрашивается состояние логического уровня на входах седьмого вывода портов P1, P2 и P3. Объединение сигналов производится при помощи команд логического "И" и "ИЛИ".

При обнаружении на выводе порта логической единицы в соответствующий бит переменной SostKn записывается логическая единица. Запись произво-

дится при помощи оператора логического сложения с константой, соответствующей логической единице в заданном бите.

При обнаружении на выводе порта логического нуля, в соответствующий бит переменной SostKn записывается логический ноль. Запись производится при помощи оператора логического умножения с константой, содержащей логический ноль в заданном бите. Константа вычисляется на этапе трансляции программы при помощи одноместной операции побитовой инверсии '~'.

Исполнение команд, вводимых при помощи кнопок, можно осуществить в любом месте основного цикла программы. Это обеспечит время исполнения команды в пределах 50 мс. Точно так же, как и в предыдущих случаях, выделим это действие в отдельную подпрограмму. Пусть эта подпрограмма называется CorrClock.

Первое действие, которое мы выполним в подпрограмме CorrClock, — это обработка команды, вводимой кнопкой установки секунд. По этой команде нужно просто обнулить счетчик секунд. Команду будем получать через глобальную переменную состояния кнопок SostKn. Исходный текст подпрограммы, выполняющей данное действие, приведен в листинге 25.16.

Листинг 25.16. Исходный текст первого варианта подпрограммы коррекции внутреннего состояния часов

```
/*********************************************************************
Подпрограмма блока коррекции часов
*********************************************************************/
void CorrClock(void)
{if((SostKn&1)==0) //Если нажата кнопка "уст сек",
   SEC=60;         //то обнулить счетчик секунд
}
```

Обработка нажатия на кнопки "уст мин" и "уст часов" производится несколько иначе. Дело в том, что по общепринятому алгоритму коррекции показаний часов при нажатии на эти кнопки необходимо непрерывно увеличивать показания счетчика часов или минут. Предварительный делитель начинает счет от числа 20, которое в двоичном представлении выглядит как 10100_2. То, что в младших разрядах записаны нулевые значения означает, что эти разряды будут меняться по двоичному закону, и единица в этих разрядах будет появляться через одинаковые промежутки времени.

Например, в младшем разряде единица будет появляться один раз за 100 мс, т. е. десять раз в секунду. В следующем разряде единица будет появляться один раз за 200 мс, т. е. пять раз в секунду. В третьем же разряде появление единицы будет неравномерным. Два раза единица в этом разряде появится

через 400 мс, а один раз через 200 мс. Если использовать сигнал с этого бита для коррекции времени, то эта непериодичность коррекции показаний часов будет раздражать пользователя, поэтому такой вариант неприемлем.

Пусть изменение показаний часов происходит пять раз в секунду. Эта скорость изменения показаний часов еще может восприниматься человеком. Подпрограмма коррекции показания часов будет выглядеть так, как это приведено в листинге 25.17.

Листинг 25.17. Исходный текст окончательного варианта подпрограммы коррекции внутреннего состояния часов

```
/*********************************************************************
Подпрограмма блока коррекции часов
*********************************************************************/
void CorrClock(void)
{if((SostKn&1)==0) //Если нажата кнопка "уст сек",
   SEC=60;           //то обнулить счетчик секунд
 if((Delit&3)==2)  //Если в предделителе число не кратно 2,
   return;           //то выйти из подпрограммы коррекции времени
 if((SostKn&2)==0) //Если нажата кнопка "уст мин",
      IncMin();      //то увеличить содержимое счетчика минут
 if((SostKn&4)==0) //Если нажата кнопка "уст час",
      IncChas();      //то увеличить содержимое счетчика минут
}
```

В приведенном варианте программы проверяется два младших бита делителя. Для выделения двух младших бит используется логическое умножение содержимого переменной `Delit` с маской `00011b`. Это число соответствует десятичному эквиваленту 3_{10}. Так как эти биты принимают значение 10_2 пять раз в секунду, то соответствующее количество раз в секунду будет вызываться подпрограмма увеличения значения содержимого счетчика минут (или часов).

Если скорость коррекции показаний часов пять раз в секунду покажется слишком высокой, то можно изменить соотношение коэффициентов деления таймера и делителя, например: 32×31250. Тогда можно будет выбрать скорость коррекции показания часов из набора вариантов два, четыре или восемь раз в секунду. Это обеспечивается двоичным характером коэффициента деления предварительного делителя, равного числу 32. В этом случае, т. к. изменился коэффициент деления таймера, один проход по основному циклу программы будет осуществляться за 31,25 мс. То есть время реакции часов увеличится. Правда, уменьшится и потребление тока от источника питания.

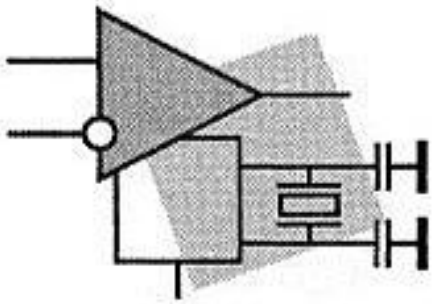
На этом можно завершить разработку часов. В заключение необходимо отметить, что ток потребления микроконтроллера на частоте 12 МГц в режиме пониженного энергопотребления составляет 6,5 мА. Это означает, что даже если отключить светодиодную индикацию, то батарейки с емкостью 300 мА/ч хватит на 46 часов работы.

Ток потребления можно снизить, применив вместо стандартного кварцевого резонатора часовой резонатор с частотой 32,768 кГц. При этом ток потребления микроконтроллера снизится в 366 раз и составит 18 мкА. При таком токе потребления той же самой батарейки хватит на 16 666 часов или на 694 суток (почти два года работы). Есть разница? Однако при этом программа предварительного делителя будет немного сложнее.

Придется строить делитель с дробным коэффициентом деления. Такие делители обычно строят, переключая два коэффициента деления с определенным периодом, но в данном учебном примере это не было сделано. Основная цель приведенного примера была не в разработке рабочего устройства, а в обучении создания и развития программного проекта на языке программирования С-51, а также иллюстрация его возможностей.

Итоги

На этом мы завершим изучение микропроцессорных устройств. В результате мы, начав с простейших логических схем, выполненных буквально на одном транзисторе, научились проектировать достаточно сложные узлы приемных и передающих устройств, использующихся в современной технике связи. Мы подробно рассмотрели особенности проектирования как устройств управления блоками аппаратуры связи, так и основные узлы цифровой обработки сигналов, такие как синтезаторы частот, микросхемы прямого цифрового синтеза, up- и down-конвертеры. Немного в стороне остались вопросы реализации цифровых устройств при помощи программируемых логических интегральных схем, но этот недостаток легко можно компенсировать при помощи специализированной литературы. Конечно, для полноты картины следовало бы рассмотреть особенности архитектуры и применения сигнальных процессоров, но это отдельный и достаточно объемный вопрос. Поэтому он остался за рамками данной книги. Надеемся, что эта книга помогла вам освоить основы цифровых устройств в достаточной степени, для того, чтобы получать дополнительные знания самостоятельно.



ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ 1

Система команд микроконтроллеров семейства MCS-51

Таблица П1.1. Машинные команды, влияющие на значения флагов регистра PSW

Мнемоника	Флаги			Мнемоника	Флаги		
	C	OV	AC		C	OV	AC
ADD	+	+	+	CLR C	0		
ADDC	+	+	+	CPL C	+		
SUBB	+	+	+	ANL C, bit	+		
MUL	0	+		ANL C, /bit			
DIV	0	+		ORL C, bit	+		
DA	+			ORL C, /bit	+		
RRC	+			MOV C, bit	+		
RLC	+			CJNE	+		
SETB C	I						

Таблица П1.2. Обозначения и символы, используемые при описании команд

Обозначение	Назначение
A	Аккумулятор
Rn	Регистры текущего выбранного банка регистров
R	Номер загружаемого регистра, указанного в команде
direct	Прямоадресуемая 8-битовая ячейка памяти данных
@Ri	Косвенно-адресуемая 8-битовая ячейка внутреннего ОЗУ данных
data 8	8-битная константа, входящая в код операции (КОП)
data 16	16-битная константа, входящая в код операции (КОП)
data H	Старшие биты (15-8) 16-битной константы
data L	Младшие биты (7-0) 16-битной константы
adr 11	Младшие 11 бит адреса
adr 16	Полный 16-битный адрес
adr L	Младшие биты адреса
rel	Байт смещения с учетом знака
bit	Адрес бита

Таблица П1.3. Перечень команд микроконтроллеров семейства MCS-51

Hex код	Число байт	Число циклов	Мнем. код	Операнды	Описание команды
00	1	1	NOP		Нет операции
01	2	2	AJMP	code addr (0 стр)	Безусловный переход в пределах 2 Кбайт
02	3	2	LJMP	code addr	Безусловный переход в пределах 64 Кбайт
03	1	1	RR	A	Циклический сдвиг вправо
04	1	1	INC	A	ACC ← ACC + 1
05	2	1	INC	data addr	data ← data + 1
06	1	1	INC	@R0	(R0) ← (R0) + 1
07	1	1	INC	@R1	(R1) ← (R1) + 1
08	1	1	INC	R0	R0 ← R0 + 1
09	1	1	INC	R1	R1 ← R1 + 1

Таблица П1.3 (продолжение)

Hex код	Число байт	Число циклов	Мнем. код	Операнды	Описание команды
0A	1	1	INC	R2	R2 ← R2 + 1
0B	1	1	INC	R3	R3 ← R3 + 1
0C	1	1	INC	R4	R4 ← R4 + 1
0D	1	1	INC	R5	R5 ← R5 + 1
0E	1	1	INC	R6	R6 ← R6 + 1
0F	1	1	INC	R7	R7 ← R7 + 1
10	3	2	JBC	bitaddr, codeaddr	Усл. переход, если бит = 1, и сброс бита
11	2	2	ACALL	codeaddr (0 стр)	Вызов подпрограммы в пределах 2 Кбайт
12	3	2	LCALL	Dataaddr	Вызов подпрограммы в пределах 64 Кбайт
13	1	1	RRC	A	Циклический сдвиг вправо через флаг C
14	1	1	DEC	A	ACC ← ACC - 1
15	2	1	DEC	dataaddr	data ← data - 1
16	1	1	DEC	@R0	(R0) ← (R0) - 1
17	1	1	DEC	@R1	(R1) ← (R1) - 1
18	1	1	DEC	R0	R0 ← R0-1
19	1	1	DEC	R1	R1 ← R1-1
1A	1	1	DEC	R2	R2 ← R2-1
1B	1	1	DEC	R3	R3 ← R3-1
1C	1	1	DEC	R4	R4 ← R4-1
1D	1	1	DEC	R5	R5 ← R5-1
1E	1	1	DEC	R6	R6 ← R6-1
1F	1	1	DEC	R7	R7 ← R7-1
20	3	2	JB	bit addr, data addr	Условный переход, если бит = 1
21	2	2	AJMP	code addr (1 стр)	Безусловный переход в пределах 2 Кбайт

Таблица П1.3 (продолжение)

Hex код	Число байт	Число циклов	Мнем. код	Операнды	Описание команды
22	1	2	RET		Возвращение из подпрограммы
23	1	1	RL	A	Циклический сдвиг влево
24	2	1	ADD	A, #data	ACC ← ACC + число
25	2	1	ADD	A, data addr	ACC ← ACC + data
26	1	1	ADD	A, @R0	ACC ← ACC + (R0)
27	1	1	ADD	A, @R1	ACC ← ACC + (R1)
28	1	1	ADD	A, R0	ACC ← ACC + R0
29	1	1	ADD	A, R1	ACC ← ACC + R1
2A	1	1	ADD	A, R2	ACC ← ACC + R2
2B	1	1	ADD	A, R3	ACC ← ACC + R3
2C	1	1	ADD	A, R4	ACC ← ACC + R4
2D	1	1	ADD	A, R5	ACC ← ACC + R5
2E	1	1	ADD	A, R6	ACC ← ACC + R6
2F	1	1	ADD	A, R7	ACC ← ACC + R7
30	3	2	JNB	bitaddr, codeaddr	Условный переход, если бит = 0
31	2	2	ACALL	codeaddr (1 стр)	Вызов подпрограммы в пределах 2 Кбайт
32	1	2	RETI		Возвр. из подпр. обслужив. прерывания
33	1	1	RLC	A	Циклический сдвиг влево через флаг C
34	2	1	ADDC	A, #data	ACC ← ACC + число + C
35	2	1	ADDC	A, dataaddr	ACC ← ACC + data + C
36	1	1	ADDC	A, @R0	ACC ← ACC + (R0) + C
37	1	1	ADDC	A, @R1	ACC ← ACC + (R1) + C
38	1	1	ADDC	A, R0	ACC ← ACC + R0+ C
39	1	1	ADDC	A, R1	ACC ← ACC + R1+ C
3A	1	1	ADDC	A, R2	ACC ← ACC + R2+ C

Таблица П1.3 (продолжение)

Hex код	Число байт	Число циклов	Мнем. код	Операнды	Описание команды
3B	1	1	ADDC	A, R3	ACC ← ACC + R3+ C
3C	1	1	ADDC	A, R4	ACC ← ACC + R4+ C
3D	1	1	ADDC	A, R5	ACC ← ACC + R5+ C
3E	1	1	ADDC	A, R6	ACC ← ACC + R6+ C
3F	1	1	ADDC	A, R7	ACC ← ACC + R7+ C
40	2	2	JC	codeaddr	Условный переход, если флаг C = 1
41	2	2	AJMP	codeaddr (2 стр)	Безусловный переход в пределах 2 Кбайт
42	2	1	ORL	dataaddr, A	data ← data | ACC
43	3	2	ORL	dataaddr, #data	data ← data | число
44	2	1	ORL	A, #data	ACC ← ACC | число
45	2	1	ORL	A, data addr	ACC ← ACC | data
46	1	1	ORL	A, @R0	ACC ← ACC | (R0)
47	1	1	ORL	A, @R1	ACC ← ACC | (R1)
48	1	1	ORL	A, R0	ACC ← ACC | R0
49	1	1	ORL	A, R1	ACC ← ACC | R1
4A	1	1	ORL	A, R2	ACC ← ACC | R2
4B	1	1	ORL	A, R3	ACC ← ACC | R3
4C	1	1	ORL	A, R4	ACC ← ACC | R4
4D	1	1	ORL	A, R5	ACC ← ACC | R5
4E	1	1	ORL	A, R6	ACC ← ACC | R6
4F	1	1	ORL	A, R7	ACC ← ACC | R7
50	2	2	JNC	code addr	Условный переход, если флаг C = 0
51	2	2	ACALL	code addr (2 стр)	Вызов подпрограммы в пределах 2 Кбайт
52	2	1	ANL	data addr, A	data ← data & ACC
53	3	2	ANL	data addr, #data	data ← data & число
54	2	1	ANL	A, #data	ACC ← ACC & число

Таблица П1.3 (продолжение)

Hex код	Число байт	Число циклов	Мнем. код	Операнды	Описание команды
55	2	1	ANL	A, data addr	ACC ← ACC & data
56	1	1	ANL	A, @R0	ACC ← ACC & (R0)
57	1	1	ANL	A, @R1	ACC ← ACC & (R1)
58	1	1	ANL	A, R0	ACC ← ACC & R0
59	1	1	ANL	A, R1	ACC ← ACC & R1
5A	1	1	ANL	A, R2	ACC ← ACC & R2
5B	1	1	ANL	A, R3	ACC ← ACC & R3
5C	1	1	ANL	A, R4	ACC ← ACC & R4
5D	1	1	ANL	A, R5	ACC ← ACC & R5
5E	1	1	ANL	A, R6	ACC ← ACC & R6
5F	1	1	ANL	A, R7	ACC ← ACC & R7
60	2	2	JZ	codeaddr	если ACC=0, то PC= codeaddr, иначе PC+2
61	2	2	AJMP	codeaddr (3 стр)	Безусловный переход в пределах 2 Кбайт
62	2	1	XRL	dataaddr, A	data ← data ⊕ ACC
63	3	2	XRL	dataaddr, #data	data ← data ⊕ число
64	2	1	XRL	A, #data	ACC ← ACC ⊕ число
65	2	1	XRL	A, data addr	ACC ← ACC ⊕ data
66	1	1	XRL	A,@R0	ACC ← ACC ⊕ (data)
67	1	1	XRL	A,@R1	ACC ← ACC ⊕ (data)
68	1	1	XRL	A, R0	ACC ← ACC ⊕ R0
69	1	1	XRL	A, R1	ACC ← ACC ⊕ R1
6A	1	1	XRL	A, R2	ACC ← ACC ⊕ R2
6B	1	1	XRL	A, R3	ACC ← ACC ⊕ R3
6C	1	1	XRL	A, R4	ACC ← ACC ⊕ R4
6D	1	1	XRL	A, R5	ACC ← ACC ⊕ R5
6E	1	1	XRL	A, R6	ACC ← ACC ⊕ R6
6F	1	1	XRL	A, R7	ACC ← ACC ⊕ R7

Таблица П1.3 (продолжение)

Hex код	Число байт	Число циклов	Мнем. код	Операнды	Описание команды
70	2	2	JNZ	codeaddr	если ACC≠0, то PC= codeaddr, иначе PC+2
71	2	2	ACALL	codeaddr (3 стр)	Вызов подпрограммы в пределах 2 Кбайт
72	2		ORL	C, bitaddr	C ← C | bit
73	1	2	JMP	@A+DPTR	Безусловный переход (PC = A+DPTR)
74	2	1	MOV	A, #data	ACC ← число
75	3	2	MOV	dataaddr, #data	data ← число
76	2	1	MOV	@R0, #data	(data) ← число
77	2	1	MOV	@R1, #data	(data) ← число
78	2	2	MOV	R0, #data	R0 ← число
79	2	2	MOV	R1, #data	R1 ← число
7A	2	2	MOV	R2, #data	R2 ← число
7B	2	2	MOV	R3, #data	R3 ← число
7C	2	2	MOV	R4, #data	R4 ← число
7D	2	2	MOV	R5, #data	R5 ← число
7E	2	2	MOV	R6, #data	R6 ← число
7F	2	2	MOV	R7, #data	R7 ← число
80	2	2	SJMP	codeaddr	Безусл. переход в пределах ±127 байт
81	2	2	AJMP	codeaddr (4 стр)	Безусловный переход в пределах 2 Кбайт
82	2	2	ANL	C, bitaddr	C ← C & bit
83	1	2	MOVC	A, @A+PC	ACC ← (A+PC)
84	1	4	DIV	AB	A ← A/B, B ← A%B
85	3	2	MOV	dataaddr1,dataaddr2	data1 ← data2
86	2	2	MOV	dataaddr, @R0	data ← (data)
87	2	2	MOV	dataaddr, @R1	data ← (data)
88	2	2	MOV	Dataaddr, R0	data ← R0

Таблица П1.3 (продолжение)

Hex код	Число байт	Число циклов	Мнем. код	Операнды	Описание команды
89	2	2	MOV	Dataaddr, R1	data ← R0
8A	2	2	MOV	Dataaddr, R2	data ← R0
8B	2	2	MOV	Dataaddr, R3	data ← R0
8C	2	2	MOV	Dataaddr, R4	data ← R0
8D	2	2	MOV	Dataaddr, R5	data ← R0
8E	2	2	MOV	Dataaddr, R6	data ← R0
8F	2	2	MOV	Dataaddr, R7	data ← R0
90	2	2	MOV	DPTR. #data	DPTR ← число
91	2	2	ACALL	codeaddr (4 стр)	Вызов подпрограммы в пределах 2 Кбайт
92	2		MOV	bitaddr, C	bit ← C
93	1	2	MOVC	A, @A+DPTR	ACC ← (A+DPTR)
94	2	1	SUBB	A, #data	ACC ← ACC - число
95	2	1	SUBB	A, dataaddr	ACC ← ACC - data
96	1	1	SUBB	A, @R0	ACC ← ACC - (data)
97	1	1	SUBB	A, @R1	ACC ← ACC - (data)
98	1	1	SUBB	A, R0	ACC ← ACC - R0
99	1	1	SUBB	A, R1	ACC ← ACC - R1
9A	1	1	SUBB	A, R2	ACC ← ACC - R2
9B	1	1	SUBB	A, R3	ACC ← ACC - R3
9C	1	1	SUBB	A, R4	ACC ← ACC - R4
9D	1	1	SUBB	A, R5	ACC ← ACC - R5
9E	1	1	SUBB	A, R6	ACC ← ACC - R6
9F	1	1	SUBB	A, R7	ACC ← ACC - R7
A0	2	2	ORL	C, /bitaddr	C ← c | not(bit)
A1	2	2	AJMP	codeaddr (5 стр)	Безусловный переход в пределах 2 Кбайт
A2	2	1	MOV	C, bitaddr	C ← bit
A3	1	2	INC	DPTR	DPTR ← DPTR+1

Таблица П1.3 (продолжение)

Hex код	Число байт	Число циклов	Мнем. код	Операнды	Описание команды
A4	1	4	MUL	AB	A ← A * B, B ← A * B
A5	Зарезервировано				
A6	2	2	MOV	@R0, dataaddr	(R0) ← data
A7	2	2	MOV	@R1, dataaddr	(R1) ← data
A8	2	2	MOV	R0, dataaddr	R0 ← data
A9	2	2	MOV	R1, dataaddr	R1 ← data
AA	2	2	MOV	R2, dataaddr	R2 ← data
AB	2	2	MOV	R3, dataaddr	R3 ← data
AC	2	2	MOV	R4, dataaddr	R4 ← data
AD	2	2	MOV	R5, dataaddr	R5 ← data
AE	2	2	MOV	R6, dataaddr	R6 ← data
AF	2	2	MOV	R7, dataaddr	R7 ← data
B0	2	2	ANL	C, /bitaddr	C ← c & not(bit)
B1	2	2	ACALL	codeaddr (5 стр)	Вызов подпрограммы в пределах 2 Кбайт
B2	2	1	CPL	bitaddr	bit ← not(bit)
B3	1	1	CPL	C	C ← not(C)
B4	3	2	CJNE	A, #data, codeaddr	Условный переход, если ACC ≠ числу
B5	3	2	CJNE	A, dataaddr, codeaddr	Условный переход, если ACC ≠ data
B6	3	2	CJNE	@R0, #data, codeaddr	Условный переход, если (R0) ≠ числу
B7	3	2	CJNE	@R1, #data, codeaddr	Условный переход, если (R1) ≠ числу
B8	3	2	CJNE	R0, #data, codeaddr	Условный переход, если R0 ≠ числу
B9	3	2	CJNE	R1, #data, codeaddr	Условный переход, если R1 ≠ числу
BA	3	2	CJNE	R2, #data, code addr	Условный переход, если R2 ≠ числу

Таблица П1.3 *(продолжение)*

Нех код	Число байт	Число циклов	Мнем. код	Операнды	Описание команды
BB	3	2	CJNE	R3, #data, code addr	Условный переход, если R3 ≠ числу
BC	3	2	CJNE	R4, #data, code addr	Условный переход, если R4 ≠ числу
BD	3	2	CJNE	R5, #data, code addr	Условный переход, если R5 ≠ числу
BE	3	2	CJNE	R6, #data, code addr	Условный переход, если R6 ≠ числу
BF	3	2	CJNE	R7, #data, code addr	Условный переход, если R7 ≠ числу
C0	2	2	PUSH	dataaddr	Сохранить ячейку памяти в стеке
C1	2	2	AJMP	code addr (6 стр)	Безусловный переход в пределах 2 Кбайт
C2	2	1	CLR	bitaddr	bit ← 0
C3	1	1	CLR	C	C ← 0
C4	1	1	SWAP	A	ACC.3.. ACC.0 ↔ ACC.7.. ACC.4
C5	2	1	XCH	A, dataaddr	ACC ↔ data
C6	1	1	XCH	A, @R0	ACC ↔ (R0)
C7	1	1	XCH	A, @R1	ACC ↔ (R1)
C8	1	1	XCH	A, R0	ACC ↔ R0
C9	1	1	XCH	A, R1	ACC ↔ R1
CA	1	1	XCH	A, R2	ACC ↔ R2
CB	1	1	XCH	A, R3	ACC ↔ R3
CC	1	1	XCH	A, R4	ACC ↔ R4
CD	1	1	XCH	A, R5	ACC ↔ R5
CE	1	1	XCH	A, R6	ACC ↔ R6
CF	1	1	XCH	A, R7	ACC ↔ R7
D0	2	2	POP	dataaddr	Извлечь содержим. стека в ячейку памяти
D1	2	2	ACALL	codeaddr (6 стр)	Вызов подпрограммы в пределах 2 Кбайт

Таблица П1.3 (продолжение)

Hex код	Число байт	Число циклов	Мнем. код	Операнды	Описание команды
D2	2	1	SETB	bitaddr	bit ← 1
D3	1	1	SETB	C	C ← 1
D4	1	1	DA	A	Десятичная коррекция аккумулятора
D5	3	2	DJNZ	dataaddr. codeaddr	data←data -1; если data≠0, то PC←codeaddr
D6	1	1	XCHD	A, @R0	ACC.3.. ACC.0 ↔ (R0.3.. R0.0)
D7	1	1	XCHD	A. @R1	ACC.3.. ACC.0 ↔ (R1.3.. R1.0)
D8	2	2	DJNZ	R0, codeaddr	R0 ← R0-1; если R0≠0. то PC = codeaddr
D9	2	2	DJNZ	R1, codeaddr	R1 ← R1-1; если R1≠0. то PC = codeaddr
DA	2	2	DJNZ	R2, codeaddr	R2 ← R2-1; если R2≠0, то PC = codeaddr
DB	2	2	DJNZ	R3, codeaddr	R3 ← R3-1; если R3≠0. то PC = codeaddr
DC	2	2	DJNZ	R4, codeaddr	R4 ← R4-1; если R4≠0. то PC = codeaddr
DD	2	2	DJNZ	R5, codeaddr	R5 ← R5-1; если R5≠0. то PC = codeaddr
DE	2	2	DJNZ	R6, codeaddr	R6 ← R6-1; если R6≠0. то PC = codeaddr
DF	2	2	DJNZ	R7, codeaddr	R7 ← R7-1; если R7≠0, то PC = codeaddr
E0	1	2	MOVX	A, @DPTR	ACC ← data из внешней памяти
E1	2	2	AJMP	codeaddr (7 стр)	Безусловный переход в пределах 2 Кбайт
E2	1	2	MOVX	A, @R0	ACC ← data из внешней памяти
E3	1	2	MOVX	A,@R1	ACC ← data из внешней памяти
E4	1	1	CLR	A	ACC ← 0

Таблица П1.3 (окончание)

Hex код	Число байт	Число циклов	Мнем. код	Операнды	Описание команды
E5	2	1	MOV	A, dataaddr	ACC ← data
E6	1	1	MOV	A, @R0	ACC ← (R0)
E7	1	1	MOV	A, @R1	ACC ← (R1)
E8	1	1	MOV	A, R0	ACC ← R0
E9	1	1	MOV	A, R1	ACC ← R1
EA	1	1	MOV	A, R2	ACC ← R2
EB	1	1	MOV	A, R3	ACC ← R3
EC	1	1	MOV	A, R4	ACC ← R4
ED	1	1	MOV	A, R5	ACC ← R5
EE	1	1	MOV	A, R6	ACC ← R6
EF	1	1	MOV	A, R7	ACC ← R7
F0	1	2	MOVX	@DPTR, A	data внешней памяти ← ACC
F1	2	2	ACALL	codeaddr (7 стр)	Вызов подпрограммы в пределах 2 Кбайт
F2	1	2	MOVX	@R0, A	(R0) ← ACC
F3	1	2	MOVX	@R1, A	(R1) ← ACC
F4	1	1	CPL	A	ACC ← not(ACC)
F5	2	1	MOV	dataaddr, A	Data ← ACC
F6	1	1	MOV	@R0, A	data внешней памяти ← ACC
F7	1	1	MOV	@R1, A	data внешней памяти ← ACC
F8	1	1	MOV	R0, A	R0 ← ACC
F9	1	1	MOV	R1, A	R1 ← ACC
FA	1	1	MOV	R2, A	R2 ← ACC
FB	1	1	MOV	R3, A	R3 ← ACC
FC	1	1	MOV	R4, A	R4 ← ACC
FD	1	1	MOV	R5, A	R5 ← ACC
FE	1	1	MOV	R6, A	R6 ← ACC
FF	1	1	MOV	R7, A	R7 ← ACC

ПРИЛОЖЕНИЕ 2

Таблица ASCII-кодов

Таблица П2.1. Таблица ASCII-кодов

Десятичный код	Шестнадцате-ричный код	Отображаемый символ	Значение
0	00		NUL
1	01	☺	SOH (слово управления дисплеем)
2	02	☻	STX (Первое передаваемое слово)
3	03	♥	ETX (Последнее слово передачи)
4	04	♦	EOT (конец передачи)
5	05	♣	ENQ (инициализация)
6	06	♠	ACK (подтверждение приема)
7	07	•	BEL (звуковой сигнал)
8	08	◘	BS
9	09	○	HT (горизонтальная табуляция)
10	0A	◙	LF (перевод строки)
11	0B	♂	VT (вертикальная табуляция)
12	0C	♀	FF (следующая страница)
13	0D	♪	CR (возврат каретки)
14	0E	♫	SO (двойная ширина)
15	0F	☼	SI (уплотненная печать)
16	10	►	DLE
17	11	◄	DC1
18	12	↕	DC2 (отмена уплотненной печати)

Таблица П2.1 (продолжение)

Десятичный код	Шестнадцатеричный код	Отображаемый символ	Значение
19	13	‼	DC3 (готовность)
20	14	¶	DC4 (отмена двойной ширины)
21	15	§	NAC (неподтверждение приема)
22	16	▬	SYN
23	17	↨	ETB
24	18	↑	CAN
25	19	↓	EM
26	1A	→	SUB
27	1B	←	ESC (начало управл. послед.)
28	1C	∟	FS
29	1D	↔	GS
30	1E	▲	RS
31	1F	▼	US
32	20		Пробел
33	21	!	Восклицательный знак
34	22	«	Угловая скобка
35	23	#	Знак номера
36	24	$	Знак денежной единицы (доллар)
37	25	%	Знак процента
38	26	&	Амперсанд
39	27	'	Апостроф
40	28	(	Открывающая скобка
41	29	)	Закрывающая скобка
42	2A	*	Звездочка
43	2B	+	Знак плюс
44	2C	,	Запятая
45	2D	-	Знак минус
46	2E	.	Точка
47	2F	/	Дробная черта
48	30	0	Цифра ноль

Таблица П2.1 *(продолжение)*

Десятичный код	Шестнадцатеричный код	Отображаемый символ	Значение
49	31	1	Цифра один
50	32	2	Цифра два
51	33	3	Цифра три
52	34	4	Цифра четыре
53	35	5	Цифра пять
54	36	6	Цифра шесть
55	37	7	Цифра семь
56	38	8	Цифра восемь
57	39	9	Цифра девять
58	3A	:	Двоеточие
59	3B	;	Точка с запятой
60	3C	<	Знак меньше
61	3D	=	Знак равно
62	3E	>	Знак больше
63	3F	?	Знак вопрос
64	40	@	Коммерческое эт
65	41	A	Прописная латинская буква A
66	42	B	Прописная латинская буква B
67	43	C	Прописная латинская буква C
68	44	D	Прописная латинская буква D
69	45	E	Прописная латинская буква E
70	46	F	Прописная латинская буква F
71	47	G	Прописная латинская буква G
72	48	H	Прописная латинская буква H
73	49	I	Прописная латинская буква I
74	4A	J	Прописная латинская буква J
75	4B	K	Прописная латинская буква K
76	4C	L	Прописная латинская буква L
77	4D	M	Прописная латинская буква M

Таблица П2.1 (продолжение)

Десятичный код	Шестнадцатеричный код	Отображаемый символ	Значение
78	4E	N	Прописная латинская буква N
79	4F	O	Прописная латинская буква O
80	50	P	Прописная латинская буква P
81	51	Q	Прописная латинская буква Q
82	52	R	Прописная латинская буква R
83	53	S	Прописная латинская буква S
84	54	T	Прописная латинская буква T
85	55	U	Прописная латинская буква U
86	56	V	Прописная латинская буква V
87	57	W	Прописная латинская буква W
88	58	X	Прописная латинская буква X
89	59	Y	Прописная латинская буква Y
90	5A	Z	Прописная латинская буква Z
91	5B	[	Открывающая квадратная скобка
92	5C	\	Обратная черта
93	5D	]	Закрывающая квадратная скобка
94	5E	^	"Крышечка"
95	5F	_	Символ подчеркивания
96	60	`	Апостроф
97	61	a	Строчная латинская буква a
98	62	b	Строчная латинская буква b
99	63	c	Строчная латинская буква c
100	64	d	Строчная латинская буква d
101	65	e	Строчная латинская буква e
102	66	f	Строчная латинская буква f
103	67	g	Строчная латинская буква g
104	68	h	Строчная латинская буква h
105	69	i	Строчная латинская буква i
106	6A	j	Строчная латинская буква j

Таблица П2.1 *(продолжение)*

Десятичный код	Шестнадцатеричный код	Отображаемый символ	Значение
107	6B	k	Строчная латинская буква k
108	6C	l	Строчная латинская буква l
109	6D	m	Строчная латинская буква m
110	6E	n	Строчная латинская буква n
111	6F	o	Строчная латинская буква o
112	70	p	Строчная латинская буква p
113	71	q	Строчная латинская буква q
114	72	r	Строчная латинская буква r
115	73	s	Строчная латинская буква s
116	74	t	Строчная латинская буква t
117	75	u	Строчная латинская буква u
118	76	v	Строчная латинская буква v
119	77	w	Строчная латинская буква w
120	78	x	Строчная латинская буква x
121	79	y	Строчная латинская буква y
122	7A	z	Строчная латинская буква z
123	7B	{	Открывающая фигурная скобка
124	7C	|	Вертикальная черта
125	7D	}	Закрывающая фигурная скобка
126	7E	~	Тильда
127	7F	⌂	Знак стирания
128	80	А	Прописная русская буква А
129	81	Б	Прописная русская буква Б
130	82	В	Прописная русская буква В
131	83	Г	Прописная русская буква Г
132	84	Д	Прописная русская буква Д
133	85	Е	Прописная русская буква Е
134	86	Ж	Прописная русская буква Ж
135	87	З	Прописная русская буква З

Таблица П2.1 *(продолжение)*

Десятичный код	Шестнадцатеричный код	Отображаемый символ	Значение
136	88	И	Прописная русская буква И
137	89	Й	Прописная русская буква Й
138	8A	К	Прописная русская буква К
139	8B	Л	Прописная русская буква Л
140	8C	М	Прописная русская буква М
141	8D	Н	Прописная русская буква Н
142	8E	О	Прописная русская буква О
143	8F	П	Прописная русская буква П
144	90	Р	Прописная русская буква Р
145	91	С	Прописная русская буква С
146	92	Т	Прописная русская буква Т
147	93	У	Прописная русская буква У
148	94	Ф	Прописная русская буква Ф
149	95	Х	Прописная русская буква Х
150	96	Ц	Прописная русская буква Ц
151	97	Ч	Прописная русская буква Ч
152	98	Ш	Прописная русская буква Ш
153	99	Щ	Прописная русская буква Щ
154	9A	Ъ	Прописная русская буква Ъ
155	9B	Ы	Прописная русская буква Ы
156	9C	Ь	Прописная русская буква Ь
157	9D	Э	Прописная русская буква Э
158	9E	Ю	Прописная русская буква Ю
159	9F	Я	Прописная русская буква Я
160	A0	а	Строчная русская буква а
161	A1	б	Строчная русская буква б
162	A2	в	Строчная русская буква в
163	A3	г	Строчная русская буква г
164	A4	д	Строчная русская буква д

Таблица П2.1 (продолжение)

Десятичный код	Шестнадцатеричный код	Отображаемый символ	Значение
165	A5	е	Строчная русская буква е
166	A6	ж	Строчная русская буква ж
167	A7	з	Строчная русская буква з
168	A8	и	Строчная русская буква и
169	A9	й	Строчная русская буква й
170	AA	к	Строчная русская буква к
171	AB	л	Строчная русская буква л
172	AC	м	Строчная русская буква м
173	AD	н	Строчная русская буква н
174	AE	о	Строчная русская буква о
175	AF	п	Строчная русская буква п
176	B0		Символ псевдографики
177	B1		Символ псевдографики
178	B2	▒	Символ псевдографики
179	B3	│	Символ псевдографики
180	B4	┤	Символ псевдографики
181	B5	╡	Символ псевдографики
182	B6	╢	Символ псевдографики
183	B7	╖	Символ псевдографики
184	B8	╕	Символ псевдографики
185	B9	╣	Символ псевдографики
186	BA	║	Символ псевдографики
187	BB	╗	Символ псевдографики
188	BC	╝	Символ псевдографики
189	BD	╜	Символ псевдографики
190	BE	╛	Символ псевдографики
191	BF	┐	Символ псевдографики
192	C0	└	Символ псевдографики
193	C1	┴	Символ псевдографики

Таблица П2.1 (продолжение)

Десятичный код	Шестнадцатеричный код	Отображаемый символ	Значение
194	C2	┬	Символ псевдографики
195	C3	├	Символ псевдографики
196	C4	─	Символ псевдографики
197	C5	┼	Символ псевдографики
198	C6	╞	Символ псевдографики
199	C7	╟	Символ псевдографики
200	C8	╚	Символ псевдографики
201	C9	╔	Символ псевдографики
202	CA	╩	Символ псевдографики
203	CB	╦	Символ псевдографики
204	CC	╠	Символ псевдографики
205	CD	═	Символ псевдографики
206	CE	╬	Символ псевдографики
207	CF	╧	Символ псевдографики
208	D0	╨	Символ псевдографики
209	D1	╤	Символ псевдографики
210	D2	╥	Символ псевдографики
211	D3	╙	Символ псевдографики
212	D4	╘	Символ псевдографики
213	D5	╒	Символ псевдографики
214	D6	╓	Символ псевдографики
215	D7	╫	Символ псевдографики
216	D8	╪	Символ псевдографики
217	D9	┘	Символ псевдографики
218	DA	┌	Символ псевдографики
219	DB		Символ псевдографики
220	DC		Символ псевдографики
221	DD		Символ псевдографики
222	DE		Символ псевдографики

Таблица П2.1 *(продолжение)*

Десятичный код	Шестнадцатеричный код	Отображаемый символ	Значение
223	DF	■	Символ псевдографики
224	E0	р	Строчная русская буква р
225	E1	с	Строчная русская буква с
226	E2	т	Строчная русская буква т
227	E3	у	Строчная русская буква у
228	E4	ф	Строчная русская буква ф
229	E5	х	Строчная русская буква х
230	E6	ц	Строчная русская буква ц
231	E7	ч	Строчная русская буква ч
232	E8	ш	Строчная русская буква ш
233	E9	щ	Строчная русская буква щ
234	EA	ъ	Строчная русская буква ъ
235	EB	ы	Строчная русская буква ы
236	EC	ь	Строчная русская буква ь
237	ED	э	Строчная русская буква э
238	EE	ю	Строчная русская буква ю
239	EF	я	Строчная русская буква я
240	F0	Ё	Прописная русская буква Ё
241	F1	ё	Строчная русская буква ё
242	F2	Є	Прописная русская буква Є
243	F3	є	Строчная русская буква є
244	F4	Ї	Прописная буква Ї
245	F5	ї	Строчная буква ї
246	F6	Ў	Прописная буква Ў
247	F7	ў	Строчная буква ў
248	F8	°	Знак градуса
249	F9	·	Булит (маркер абзаца)
250	FA	·	Знак умножения (точка в середине строки по высоте)
251	FB	√	Радикал (взятие корня)

***Таблица П2.1** (окончание)*

Десятичный код	Шестнадцатеричный код	Отображаемый символ	Значение
252	FC	№	Знак номера
253	FD	¤	Знак денежной единицы (рубль)
254	FE	■	Символ псевдографики
255	FF		Неразрывный пробел

Список литературы

1. Богданович М. И. Цифровые интегральные микросхемы. Мн.: Беларусь, 1996.
2. Бродин В. Б., Калинин А. В. Системы на микроконтроллерах и БИС программируемой логики. — М.: Эком, 2002.
3. Гольденберг Л. М. Цифровая обработка сигналов. — М.: Радио и связь, 1985.
4. Майоров С. А., Кириллов В. В., Приблуда А. А. Введение в микро-ЭВМ. —М.: Мир, 1988.
5. Микушин А. В., Сажнев А. М., Сединин В. И. Цифровые устройства и микропроцессоры. Учебное пособие. — Новосибирск: ВЕДИ, 2005.
6. Петровский И. И., Прибыльский А. В. и др. Логические ИС КР1533, КР1554. — М.: БИНОМ, 1993.
7. Рафикумазан М. Микропроцессоры и машинное проектирование микропроцессорных систем. — М.: Мир, 1988.
8. Ричард Лайонс. Цифровая обработка сигналов. — М.: Бином-Пресс, 2006.
9. Сажнев А. М., Сединин В. И. Цифровые устройства и микропроцессоры. Учебное пособие. — Новосибирск СибГУТИ, 2004.
10. Сергиенко А. Б. Цифровая обработка сигналов. — СПб.: Питер, 2002.
11. Титце У., Шенк К. Полупроводниковая схемотехника. — М.: Мир, 1983.
12. Угрюмов Е. Цифровая схемотехника. Учебное пособие. — СПб.: БХВ-Петербург, 2001.
13. Уэйкерли Дк. Ф. Проектирование цифровых устройств. — М.: Постмаркет, 2002.
14. Хоровиц П., У. Хилл. Искусство схемотехники, 3-е издание. — М.: Мир, 1986.

15. Шило В. Л. Популярные цифровые микросхемы: Справочник. — М.: Р и С, 1987.
16. SCAA035B Texas Instruments CMOS Power Consumption and Cpd Calculation. — June 1997.
17. SZZA043 Moshiul Haque and Ernest Cox Use of the CMOS Unbuffered Inverter in Oscillator Circuits. — January 2004. Application Report.
18. SCZA004A Chris Wellheuser Metastability Performance of Clocked FIFOs. — 1996.
19. SDYA010 Texas Instruments Input and Output Characteristics of Digital Integrated Circuits. — November 1996.
20. SCYD013 Texas Instruments Digital Logic Pocket. Data Book. — 2003.
21. XAPP077 Xilinx Metastability Considerations January, 1997 (Version 1.0) Application Note.
22. http://digital.sibsutis.ru/contCU.htm — конспект лекций по курсу "Цифровые устройства".
23. http://digital.sibsutis.ru/content.htm — конспект лекций по курсу "Микропроцессоры".
24. http://dfe3300.karelia.ru/koi/posob/log_basis/index.html — конспект лекций.
25. http://www.labfor.ru/
26. http://www.ti.com/digitalcontrol
27. http://www.bashedu.ru/wsap/posobie/Content.htm — конспект лекций.
28. http://www.inp.nsk.su/~kozak/ac/ach00.htm — справочные данные цифровых микросхем.

Предметный указатель

T

U

А

Б

В

Г

Д

Ж

З

И

К

Л

М

Н

О

П

P

C

Т

У

Ф

Ц

Ч

Ш

Э

Я