МИРОВАЯ ЭЛЕКТРОНИКА

СЕРИЯ

ATMEL

В.Н. Баранов

Применение микроконтроллеров AVR: схемы, алгоритмы, программы

AT90S8515
8IC
0036

ISBN 5-94120-128-1

МИРОВАЯ ЭЛЕКТРОНИКА
СЕРИЯ

В. Н. Баранов

Применение микроконтроллеров AVR: схемы, алгоритмы, программы

2-е издание

Москва
Издательский дом «Додэка-XXI»
2006

УДК 621.316.544.1 (035.5)
ББК 32.844.1-04я2
Б24

Баранов В.Н.

Б24 Применение микроконтроллеров AVR: схемы, алгоритмы, программы (+CD), 2-е изд. испр. — М.: Издательский дом «Додэка-XXI», 2006. — 288 с.: ил. (серия «Мировая электроника»)

ISBN 5-94120-128-1

Какой микроконтроллер выбрать? Где найти его описание? Где взять программу, обеспечивающую написание, отладку и редактирование программ для микроконтроллера? Где взять программатор и програмное обеспечение для него? Как приступить к работе, когда все это уже есть? Как все сделать с минимальными затратами средств и времени?

Автор делится опытом работы с 8-разрядными микроконтроллерами AVR корпорации Atmel. Книга знакомит с действиями, необходимыми для начала применения микроконтроллеров. Показаны все этапы разработки устройств на микроконтроллерах. Особое внимание уделено связи предлагаемых схемных решений с программным обеспечением разрабатываемых устройств. В каждой главе предлагаются электрические схемы устройств на базе микроконтроллеров AVR, а также несколько программ, определяющих их функционирование. Все устройства с приведенными программами вполне работоспособны и могут быть повторены. Функциональные узлы микроконтроллеров описаны в объеме, достаточном для понимания программ. Приведенные в книге программы отлаживались в среде AVR Studio версии 4.08, работе с которой посвящена отдельная глава книги.

Второе издание комплектуется компакт-диском, на котором читатели найдут листинги программ, описанных в книге, дистрибутивы среды AVR Studio версий 4.08 и 4.12, спецификации на 8-разрядные RISC-микроконтроллеры семейства AVR и множество полезной информации по микроконтроллерам с сайта компании ATMEL.

Материал основан на рассмотрении реально работающих устройств и излагается по принципу «от простого к сложному». Поэтому книга представляет интерес как для начинающих «электронщиков», так и для специалистов.

УДК 621.316.544.1 (035.5)
ББК 32.844.1-04я2

Издательский дом «Додэка-XXI».
ОКП 95 3000
105318 Москва, а/я 70 Тел./факс: (095) 366-24-29, 366-81-45
E-mail: books@dodeca.ru; red@dodeca.ru
Отдел рекламы ra@dodeca.ru

Подписано в печать 17.02.2006. Формат 60x90/16. Бумага типограф. № 2.
Гарнитура «NewtonC». Печать офсетная. Объем 18.0 п. л. Усл. печ. л. 18.0.
Тираж 2000 экз. Изд. № 135. Заказ № 412.

Отпечатано с готовых диапозитивов в ОАО «Типография Новости».
105005 Москва, ул. Ф. Энгельса. 46

ISBN 5-94120-128-1

© Баранов В. Н., 2004
© Издательский Дом «Додэка-XXI», 2004
® Серия «Мировая электроника»

Оглавление

Предисловие 6

Глава 1. **Что нужно для работы с микроконтроллером** 8
1.1. Где найти и как загрузить минимальный набор программного обеспечения и документации для микроконтроллеров AVR 8
1.2. О программаторах 9
1.3. Источник питания 10
1.4. Дополнительные сведения 11

Глава 2. **Первый проект: контроллер сигнализации** 12
2.1. Постановка задачи 12
2.1.1. Устройства, подключаемые к контроллеру, и параметры входных и выходных сигналов 12
2.1.2. Логика работы контроллера 13
2.1.3. Схема сигнализации 14
2.1.4. Словесное описание алгоритма работы контроллера 16
2.2. Начинаем работу с AVR Studio 17
2.2.1. Создание первой программы на Ассемблере 17
2.2.2. Программа для контроллера сигнализации с использованием прерываний 48
2.3. Советы 66

Глава 3. **Работа с внешним статическим ОЗУ** 68
3.1. Интерфейс микроконтроллера ATmega8515 для подключения внешней памяти 68
3.2. Пример подключения внешнего ОЗУ к микроконтроллеру ATmega8515 70
3.2.1. Схема 70
3.2.2. Установка адреса 72
3.2.3. О выборе микросхемы регистра 72
3.2.4. Считывание данных из внешней памяти 73
3.2.5. Запись данных во внешнюю память 73

3.3. Программный доступ к оперативной памяти ... 73
3.3.1. Простая программа обращения к оперативной памяти ... 73
3.3.2. Отладка программы ... 75
3.3.3. Сохранение содержимого ОЗУ на диске ... 77
3.3.4. Запись данных в начальную область внешней памяти ... 77
3.4. Обращение к буферам как к ячейкам памяти микроконтроллера ATmega8515 ... 79
3.4.1. Электрическая схема подключения буферов ... 79
3.4.2. Программа обслуживания буферов ... 82
3.4.3. Отладка программы обслуживания буферов ... 84
3.5. Подключение внешней памяти 512 Кбайт к микроконтроллеру ATmega8535 ... 87
3.6. Схема подключения ОЗУ к микроконтроллеру ATmega8535 ... 87
3.6.1. Описание схемы ... 89
3.6.2. Запись в ячейку ... 90
3.6.3. Считывание из ячейки ... 90
3.7. Программа записи данных в ОЗУ 512 Кбайт ... 91
3.7.1. Отладка программы ... 95
3.7.2. Подпрограмма установки адреса SetAddr: ... 97
3.7.3. Подпрограмма копирования байта из внутреннего ОЗУ DataSt... 98
3.7.4. Подпрограмма копирования данных из внешней памяти во внутреннее ОЗУ DataLd ... 99

Глава 4. **Устройство динамической индикации на 7-сегментных индикаторах** .. 100
4.1. Принцип динамической индикации ... 100
4.2. Восьмиразрядное устройство отображения цифровой информации ... 102
4.2.1. Схема управления восьмиразрядным индикатором ... 102
4.2.2. Программа организации бегущей строки ... 105
4.2.3. Описание программы ... 108
4.3. Устройство управления двумя печами ... 114
4.3.1. Работа устройства ... 114
4.3.2. Программа управления двумя печами ... 116
4.3.3. Работа с устройством управления двумя печами ... 152
4.3.4. Особенности работы EEPROM микроконтроллера ... 154

Глава 5. **Связь микроконтроллера с компьютером** ... 157
5.1. Схема контроллера, обеспечивающая связь с COM-портом компьютера ... 157
5.2. Программное обеспечение связи по каналу RS-232 ... 160
5.2.1. Протокол обмена ... 161
5.2.2. Общие положения ... 161
5.2.3. Структура сообщения ... 162
5.2.4. Передаваемые сообщения (команды компьютера и ответы контроллера) ... 162
5.2.5. Программа для микроконтроллера ... 163
5.2.6. Отладка работы UART в AVR Studio ... 185
5.3. Канал RS-232: программное обеспечение для компьютера ... 187
5.3.1. Минимальные сведения о Delphi ... 188
5.3.2. Программа обмена данными с микроконтроллером ... 189
5.3.3. Описание работы программы ... 197
5.3.4. Сохранение, запуск, использование программы ... 201
5.4. Программа-монитор связи через COM-порты ... 201
5.5. Использование функций Windows API для обращения к COM-порту ... 206

Глава 6. **Организация аналоговых выходов для микроконтроллера** ... 209
6.1. Преобразование кода в ширину импульса ... 210
6.1.1. ЦАП и генератор пилообразного напряжения с PWM ... 210
6.1.2. Таймер T1 микроконтроллера в режиме PWM ... 210
6.1.3. Программа для генератора PWM ... 211
6.2. Преобразование кода в амплитуду импульса ... 214
6.2.1. Генератор пилообразного напряжения ... 214
6.2.2. Программа для генератора пилообразного напряжения ... 217
6.2.3. Генератор синусоидального сигнала ... 221
6.2.4. Программа для генератора синусоидального сигнала ... 223
6.3. Определение пространственного модуля сигнала ... 228
6.3.1. Алгоритм программы ... 228
6.3.2. Листинг программы вычисления модуля ... 229
6.4. Цифровой фильтр ... 243
6.4.1. Листинг С-программы цифрового фильтра ... 245

Приложение 1. **Как получить необходимые материалы через сеть Internet** ... 253

Приложение 2. **Устройства, облегчающие отладку контроллера в составе системы** ... 255

Приложение 3. **Программатор** ... 261

Приложение 4. **8-разрядные RISC-микроконтроллеры фирмы Atmel** ... 284

Материалы, размещенные на компакт-диске ... 288

Предисловие

Уважаемые читатели!

Книга, предлагаемая вашему вниманию, написана на основе опыта работы с 8-разрядными микроконтроллерами AVR серии ATmega корпорации Atmel, а также с учетом вопросов, поступающих на сайт *http://bvn123.boom.ru/*.

> **Контроллер** — законченное электронное устройство, обычно выполненное в виде платы (плата контроллера) и предназначенное для приема и обработки сигналов от датчиков, а также для управления внешними устройствами на основании результатов обработки принятых сигналов.
> **Микроконтроллер** — программно управляемая интегральная микросхема, применяемая для построения различных контроллеров.

Трудности, возникающие у разработчика при проектировании пятого или десятого контроллера, меркнут на фоне проблем, с которыми сталкивается новичок. Обычно возникают следующие вопросы:

- какой микроконтроллер выбрать;
- где найти его описание;
- где взять программу, обеспечивающую написание, отладку и редактирование программ для микроконтроллера;
- где взять программатор и программное обеспечение для него;
- как приступить к работе, когда все это уже есть;
- как все это сделать с минимальными затратами средств и времени.

Вопросам получения через сеть Internet минимального набора программ и документации, достаточного для работы с микроконтроллерами AVR, посвящена первая глава книги.

Так как поиск информации представляет определенные трудности для тех, кто имеет небольшой опыт работы в сети Internet, в Приложении 1 даны рекомендации по поиску необходимой информации.

Основная цель второй главы — обучение навыкам эффективной работы в среде разработки и отладки программ AVR Studio 4.08 для микроконтроллеров. В этой главе возможности AVR Studio рассматриваются очень подробно, так, чтобы разобраться в них смог читатель, не обладающий достаточным опытом работы на компьютере. Освоение AVR Studio проходит на конкретном примере полного цикла разработки устройства сигнализации — от описания требований к устройству до отладки программы. Здесь же даны рекомендации по обнаружению и исправлению ошибок в программе.

Примерно одинаково построение остальных глав книги. В каждой из них предлагаются электрические схемы контроллеров на базе микроконтроллеров AVR, а также несколько различных программ, определяющих функционирование контроллеров. Функциональные узлы микроконтроллеров описаны в объеме, достаточном для понимания программ, полное их описание можно найти в техническом описании микроконтроллеров.

В Приложении 2 вы найдете описание устройств, облегчающих тестирование контроллеров, а также цифровых схем.

В Приложении 3 приведены сведения, которые помогут вам при разработке собственного программатора.

Все описываемые в книге программы для микроконтроллеров отлаживались в AVR Studio версии 4.08. Программное обеспечение для компьютера написано в Delphi.

Описываемые устройства являются фрагментами больших разработок. Эти фрагменты адаптированы специально для книги так, чтобы наиболее лаконично проиллюстрировать работу изучаемой функции микроконтроллера либо предлагаемого аппаратного или программного решения. При этом все устройства с приведенными программами вполне работоспособны и могут быть повторены.

Наиболее продуктивным представляется просмотр книги с одновременной отработкой изучаемого материала на компьютере. Особенно это касается второй главы — ознакомление с нею без работы с компьютером вряд ли принесет пользу. В связи с этим перед изучением какой-либо программы, приводимой в книге, лучше предварительно набрать ее в AVR Studio и иметь ее перед глазами на мониторе. Кроме того, материал каждой последующей главы дается в расчете на то, что читатель ознакомился с предыдущими главами, поэтому описание уже разбиравшихся ранее фрагментов программ или схем опускается.

Ваши замечания и пожелания направляйте по адресу *bvn123@bk.ru*

Надеюсь, что книга окажется вам полезной.

Автор.

Глава 1. Что нужно для работы с микроконтроллером

Для работы с 8-разрядными микроконтроллерами AVR корпорации Atmel (далее — микроконтроллер) вам понадобится программное обеспечение для разработки и отладки проектов, документация на микроконтроллеры, программатор и источник питания.

Вероятно, компакт-диск, содержащий программное обеспечение, и документацию на различные программаторы можно приобрести в любом крупном городе.

Однако все необходимое, включая схему программатора, а также компьютерную программу для его обслуживания, можно загрузить из сети Internet.

В целях экономии вашего времени в следующем разделе приводятся ссылки непосредственно на материалы, которые вам понадобятся. Ссылки были проверены в марте 2004 года, если же они перестали работать, вам придется разыскать их самостоятельно. Возможно, вам будут полезны рекомендации по поиску и загрузке материалов из сети, приведенные в Приложении 1.

1.1. Где найти и как загрузить минимальный набор программного обеспечения и документации для микроконтроллеров AVR

Для работы вам необходимо загрузить с сайта корпорации Atmel следующие материалы:

- последнюю версию интегрированной среды разработки программного обеспечения для микроконтроллеров AVR;
- файлы документации на микроконтроллеры;
- архив avr000.zip с файлами *.inc наименований регистров и констант микроконтроллеров.

Вы можете найти эти материалы на прилагаемом компакт-диске или разыскать самостоятельно на сайте компании Atmel (*http://www.atmel.ru/*).

1.2. О программаторах

Ссылки на программаторы не приводятся. Попробуйте разыскать их самостоятельно. Воспользуйтесь для этого поисковыми сайтами, обеспечивающими поиск в сети по ключевым словам. В качестве ключевой можно использовать фразу «программатор микроконтроллера avr». Более подробно о поиске читайте в Приложении 1.

Предлагаемые для микроконтроллеров AVR программаторы можно классифицировать по способу загрузки программ в память микроконтроллеров и по способу подключения к компьютеру.

Программаторы с параллельной (побайтной) загрузкой программ практически не предлагаются. Их преимущества — более высокая скорость программирования и некоторые дополнительные возможности (в частности, они позволяют установить в микроконтроллере защиту от работы с последовательным программатором). К недостаткам можно отнести необходимость извлечения микроконтроллера из системы для перепрограммирования (то есть из платы контроллера, в составе которой работает микроконтроллер). Вероятно, основное назначение таких программаторов — массовое программирование микроконтроллеров перед их установкой в систему. Однако, используя программатор с последовательной загрузкой, например, совместно с микроконтроллером ATMega8, можно перепрограммировать его линию RESET (системный сброс микроконтроллера) так, что в дальнейшем она будет функционировать как обычная линия ввода/вывода. После этого работа программатора с последовательной загрузкой становится невозможной. Объясняется это тем, что для его работы требуется функционирование упомянутой линии именно в режиме RESET. Восстановить же работу линии в режиме RESET можно только с помощью параллельного программатора.

Для разработчика более удобен программатор с последовательной (побитной) загрузкой программ в микроконтроллер. При соблюдении некоторых мер на этапе проектирования схемы контроллера, о которых будет упомянуто позже, с помощью последовательного программатора микроконтроллер можно запрограммировать, не извлекая его из системы.

Программаторы подключаются либо к LPT-порту, либо к COM-порту компьютера. Последний дороже и неудобен, так как содержит значительно больше элементов, в том числе и недорогой микроконтроллер, который должен быть запрограммирован каким-то другим программатором. Зато кабель для его подключения к компьютеру может быть длинным.

Для этого можно воспользоваться простейшим программатором, представляющим собой несколько проводников, подключаемых с одной стороны к LPT-порту, а с другой — к программируемому микроконтроллеру.

Но такой программатор работает неустойчиво, а проводники должны быть как можно короче.

Вполне приемлемы программаторы, подключаемые к LPT-порту разъемом DB25. В корпусе такого разъема размещается печатная плата программатора. При длине кабеля до 1 м сбои в работе программатора не обнаруживаются. Основой программатора является микросхема КР1533АП5 или ее аналог.

Очень удобно, если после программирования микроконтроллера компьютерная программа, обслуживающая программатор, переводит все его выходные линии, связанные с микроконтроллером, в высокоимпедансное состояние — в этом случае нет необходимости каждый раз после программирования микроконтроллера отключать программатор.

Несомненным достоинством программатора, подключаемого к LPT-порту компьютера, является возможность отлаживать связь микроконтроллера с компьютером через COM-порт без отключения программатора и микроконтроллера от компьютера. Если мышка подключается к компьютеру через один из COM-портов, то в распоряжении разработчика обычно остается лишь один свободный COM-порт. При использовании программатора, подключаемого к COM-порту компьютера, вы будете вынуждены постоянно подсоединять то программатор, то микроконтроллер к COM-порту, разъем которого к тому же находится на задней панели компьютера.

При выборе программатора стоит также учесть его способность программировать как серию AT90, так и серию ATMega. Поэтому желательно, чтобы программное обеспечение программатора поддерживало протокол программирования обеих серий. Кроме того, необходимо убедиться, что программное обеспечение поддерживает программирование битов защиты памяти программ от считывания (Lock bits) именно с вашим программатором.

1.3. Источник питания

Обычно микроконтроллер и программатор питаются от одного источника напряжения (+5 В). Источник должен быть гальванически развязан от сети переменного тока 220 В 50 Гц, так как общий провод источника соединяется через программатор с компьютером.

Желательно использовать источник с регулируемым током срабатывания защиты, что позволит обезопасить как микроконтроллер, так и программатор при ошибках в монтаже контроллера, при неверном подключении питающих проводников, а также при коротких замыканиях.

Следует предварительно рассчитывать ток потребления системы и устанавливать соответствующий ток срабатывания защиты перед включением.

Указанным условиям соответствуют источники питания постоянного тока серии Б5-хх (например, Б5-44). Впрочем, вы можете сделать источник питания самостоятельно или даже питать микроконтроллер и программатор от батареек.

1.4. Дополнительные сведения

Чтобы решить, какой именно микроконтроллер AVR наилучшим образом подойдет для вашей задачи, необходимо сравнить возможности всех выпускаемых микроконтроллеров AVR. Таблица параметров выпускаемых микроконтроллеров AVR находится в Приложении 4.

Статьи о микроконтроллерах AVR, в том числе и обзоры, на русском языке можно найти на сайте *http://www.atmel.ru/*.

При разработке возникает необходимость в профессиональной консультации. Попробуйте получить ответ на ваш вопрос на русском сайте *http://www.atmel.ru/*. Получить специфическую информацию, которой не располагают специалисты русского сайта, можно в корпорации Atmel. Информация на страничке *http://www.atmel.com/dyn/general/contact.asp* поможет вам разыскать адрес электронной почты службы технической поддержки.

Полезно просматривать материалы телеконференций по микроконтроллерам, их адреса можно найти с помощью поисковых машин, введя в строку запроса фразу «конференция по микроконтроллерам».

Глава 2. **Первый проект: контроллер сигнализации**

Основные цели этой главы:
- ознакомить читателя с основными этапами разработки контроллера и максимально быстро обучить приемам работы в среде разработки программ для микроконтроллеров AVR Studio 4.08;
- разобраться с организацией и работой портов ввода/вывода;
- понять принцип работы аппаратных прерываний микроконтроллера;
- пройти этапы разработки контроллера от схемы до отладки программы.

Разработка простого устройства сигнализации, подходящего для защиты помещений от вторжения, — удобный пример, не перенасыщенный техническими деталями.

2.1. Постановка задачи

Пусть требуется изготовить простой контроллер сигнализации, работающей в следующих режимах:
- ожидание;
- вторжение;
- штатное отпирание двери;
- отпирание изнутри.

2.1.1. Устройства, подключаемые к контроллеру, и параметры входных и выходных сигналов

Питание сигнализации производится от автомобильного аккумулятора (+12 В). Аккумулятор подзаряжается от сети переменного тока через зарядное устройство, которое здесь не рассматривается. Исполнительные устройства сигнализации питаются непосредственно от аккумулятора.

На дверь устанавливается кнопка. При запертой двери кнопка нажата, а ее контакты разомкнуты, при отпирании двери или взломе контакты замкнуты.

Контроллер должен управлять электрическим замком. Открывание производится подачей на соленоид замка напряжения +12 В. При отсутствии напряжения на соленоиде замок запирается автоматически под действием пружинного механизма.

Пара контактов кодового устройства замыкается на короткое время после того, как кнопки устройства нажаты в определенной последовательности.

При нажатии кнопки открывания замка внутри помещения замок открывается без ввода кода.

Для сигнализации используется светодиод на удаленном пульте, а также сирена. Для включения сирены на нее должна быть подана импульсная последовательность, частота повторения импульсов 1 кГц, амплитуда импульсов 12 В.

При отказе или разрядке аккумулятора замок открывается изнутри вручную, сирена и светодиод не включатся, так как питания на них не будет.

2.1.2. Логика работы контроллера

Режим ожидания

В этом режиме контакты кнопки, установленной на двери, замкнуты, контакты кодового устройства разомкнуты, сирена и светодиод выключены, ток через соленоид замка протекать не должен.

Режим вторжения

При взломе двери контакты кнопки, установленной на двери, замкнуты. Если предшествующего замыкания контактов кодового устройства не было, это должно вызвать включение сирены и светодиода на удаленном пульте. Сирена должна работать в прерывистом режиме: продолжительность звука и продолжительность пауз равны и должны составлять 0.5 с. Выключение сирены и светодиода должно производиться внутри помещения отдельной кнопкой, расположенной на плате контроллера.

Режим штатного отпирания двери

После ввода кода в правильной последовательности кратковременно замыкается пара контактов кодового устройства, к соленоиду замка должно быть приложено напряжение 12 В в течение 2 с, замок отопрется. Если дверь после этого будет открыта, замыкание контактов кнопки, установленной на двери, не приведет к срабатыванию сирены и светодиода.

Режим отпирания двери изнутри

При замыкании контактов кнопки отпирания изнутри к соленоиду замка должно быть приложено в течение 2 с напряжение 12 В, замок отопрется. Если дверь после этого будет открыта, замыкание контактов кнопки, установленной на двери, не приведет к срабатыванию сирены и светодиода.

2.1.3. Схема сигнализации

На **Рис. 1** приведена схема, удовлетворяющая перечисленным условиям.

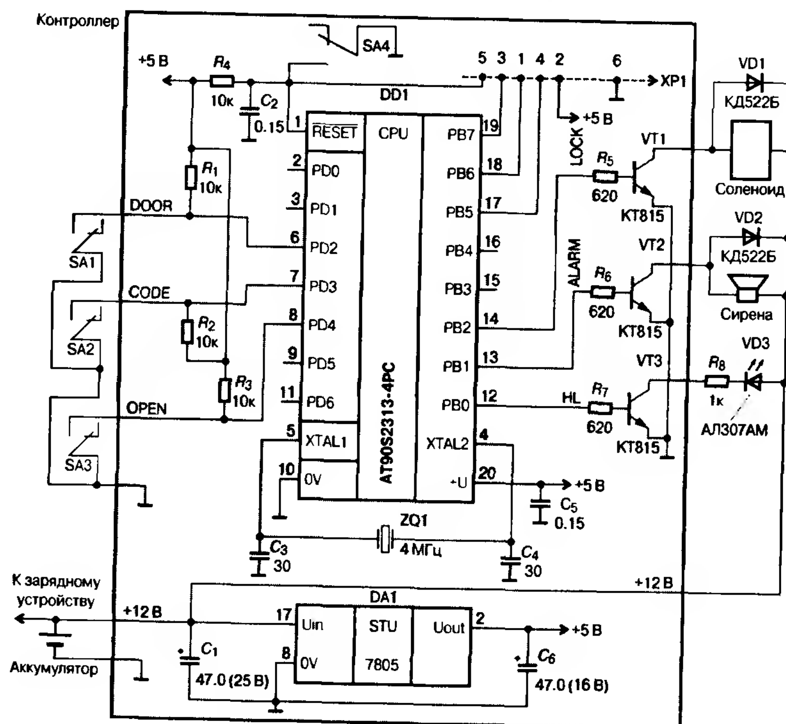


Рис. 1. Схема контроллера сигнализации

Кнопка SA1 установлена на двери, кнопка SA2 представляет пару контактов в кодовом устройстве, кнопка SA3 установлена внутри помещения. Положение контактов всех трех кнопок на схеме соответствует режиму ожидания (дверь заперта).

Параллельно соленоиду и сирене (индуктивные нагрузки) подключены диоды, они защищают транзисторы VT1, VT2 и VT3 контроллера от токов, возникающих в индуктивных нагрузках после снятия с них напряжения.

Питание схемы осуществляется от автомобильного аккумулятора.

Штриховой линией окружены компоненты, размещаемые на плате контроллера. 10-контактный разъем XP1 предназначен для подключения к микроконтроллеру программатора.

В устройстве используется микроконтроллер AT90S2313, для питания которого необходимо напряжение, не превышающее 6 В. Для получения стабилизированного напряжения питания +5 В применена микросхема 7805 (DA1).

Если цепь DOOR замкнута на общий провод кнопкой SA1, то в цепи устанавливается НИЗКИЙ логический уровень; при отсутствии такого замыкания в цепи поддерживается ВЫСОКИЙ логический уровень через резистор R_1, соединенный с источником напряжения +5 В. То же справедливо для кнопок SA2, SA3, резисторов R_2, R_3 и цепей CODE и OPEN.

НИЗКИЙ логический уровень близок к 0 В, ВЫСОКИЙ логический уровень близок к напряжению питания микроконтроллера, составляющему для этой схемы +5 В.

Резистор R_4 с конденсатором C_2 обеспечивают аппаратный сброс микроконтроллера при включении питания. При отсутствии напряжения или его пропадании конденсатор C_2 оказывается разряженным. После появления напряжения питания на контакте $\overline{\text{RESET}}$ микроконтроллера удерживается НИЗКИЙ логический уровень до тех пор, пока конденсатор C_2 не зарядится через резистор R_4.

Для сброса программы (выключения сирены и светодиода после вторжения) в схему введена кнопка SA4, при замыкании контактов которой на линии $\overline{\text{RESET}}$ микроконтроллера устанавливается НИЗКИЙ уровень, что вызывает аппаратный сброс микроконтроллера.

Для управления мощной нагрузкой, питающейся большим напряжением, служат транзисторы VT1, VT2 и VT3.

Если на выходе микроконтроллера, соединенного с базой транзистора через резистор, сформируется ВЫСОКИЙ уровень, транзистор откроется. Тогда подключенная к его коллектору цепь нагрузки соединится с общим проводом через малое сопротивление транзисторного перехода коллектор—эмиттер, почти все напряжение аккумулятора будет приложено к нагрузке.

Когда на выходе микроконтроллера НИЗКИЙ уровень, транзистор закрыт, сопротивление его перехода коллектор—эмиттер очень велико, напряжение на нагрузке оказывается близким к нулю.

> Таким образом, для срабатывания одной из нагрузок, подключенных к схеме, на соответствующем выходе микроконтроллера надо сформировать ВЫСОКИЙ уровень, для прекращения работы нагрузки — НИЗКИЙ уровень.

2.1.4. Словесное описание алгоритма работы контроллера

После включения питания по аппаратному сбросу

- линии порта D микроконтроллера DD1, соединенные с контактами кнопок SA1, SA2 и SA3, должны быть сконфигурированы как входные;
- линии порта B, управляющие нагрузками, необходимо сконфигурировать как выходные;
- на линиях порта B установить НИЗКИЕ уровни (нагрузки — соленоид замка, сирена и светодиод — выключены).

Режим ожидания

Контроллер находится в состоянии ожидания, необходимо организовать циклическую проверку состояния линий порта D.

Режим штатного отпирания двери

При появлении НИЗКОГО уровня в цепи CODE (введен правильный код, переход в режим штатного открывания двери) сформировать ВЫСОКИЙ уровень на линии LOCK, управляющей соленоидом замка; удерживать уровень в течение 2 с для отпирания замка.

Через пять секунд после запирания замка проверить уровень в цепи DOOR. Если в цепи ВЫСОКИЙ уровень (дверь не открывали или успели закрыть), контроллер должен вернуться в режим ожидания. Если уровень НИЗКИЙ (дверь еще открыта) — ожидать появления ВЫСОКОГО уровня (запирания двери), после чего перейти в режим ожидания.

Режим отпирания двери изнутри

При появлении НИЗКОГО уровня в цепи OPEN сформировать ВЫСОКИЙ уровень на линии LOCK, управляющей соленоидом замка; удерживать уровень в течение 2 с для отпирания замка.

Через пять секунд после запирания замка проверить уровень в цепи DOOR. Если в цепи ВЫСОКИЙ уровень (дверь не открывали или успели закрыть), контроллер должен вернуться в режим ожидания. Если уровень НИЗКИЙ (дверь еще открыта) — ожидать появления ВЫСОКОГО уровня (запирания двери), после чего перейти в режим ожидания.

Режим вторжения

При появлении НИЗКОГО уровня в цепи DOOR (дверь открыта) сформировать ВЫСОКИЙ уровень на линии HL для включения светодиода, на линии ALARM сформировать импульсную последовательность, вызывающую прерывистый звуковой сигнал сирены. Для этого чередовать импульсные последовательности с частотой повторения 1 кГц в течение 0.5 с (паузы также 0.5 с).

Возвращение в режим ожидания должно происходить при временном появлении НИЗКОГО уровня на контакте RESET микроконтроллера, что обеспечивается нажатием кнопки SA4 или выключением-включением питания.

2.2. Начинаем работу с AVR Studio

Установка AVR Studio обычно проходит гладко. Для начала установку стоит сделать, просто соглашаясь со всеми появляющимися предложениями программы-установщика.

2.2.1. Создание первой программы на Ассемблере

Подготовка к работе

Для начала необходимо распаковать архив avr000.zip.

Распакуйте файл avr000.zip (его скачали вместе с AVR Studio, см. подраздел 1.1) в папку C:\AVR\def.

В результате произведенной распаковки папка C:\AVR\def будет содержать файлы *def.inc, в которых хранятся предопределенные имена регистров и констант микроконтроллеров AVR. Именно это имя папки для файлов *def.inc (C:\AVR\Def) будет использоваться в примерах программ, приведенных в книге.

В папке C:\AVR для каждого нового проекта будем создавать новую папку для хранения всех файлов, относящихся к проекту.

Запустите AVR Studio. Появившееся окно (Рис.2) предоставляет возможность открыть существующий или создать новый проект. Если флажок «Show this dialog on open», расположенный в левом нижнем углу, оставить установленным, окно будет появляться при каждом запуске программы. Так как в течение одного сеанса может понадобиться работать с разными проектами, закроем это окно, щелкнув по кнопке Cancel, а затем создадим новый проект, пользуясь меню AVR Studio. В оставшемся на экране окне AVR Studio в первой строке находятся заголовки меню File, Project, View, Tools, Debug и Help.

Для ознакомления с возможностями AVR Studio откройте меню Help/AVR Studio User Guide. Для этого надо установить указатель мышки на слове Help, щелкнуть мышкой (нажать левую кнопку мышки), в от-

крывшемся окне меню переместить указатель мышки на строку AVR Studio User Guide и щелкнуть мышкой. Альтернативный вариант — воспользоваться клавиатурой.

Не пренебрегайте меню Help, обращайтесь к нему при любой возможности. Многие разработчики осваивают лишь некий набор возможностей используемой ими программы, позволяющий выполнять текущие задачи. Это касается использования любого программного пакета. Но даже при большом опыте работы в AVR Studio периодический просмотр тем из меню Help, которым раньше не уделялось внимание, поможет существенно повысить уровень знаний, а в итоге облегчить и ускорить работу.

Создание проекта

Наш проект назовем Alarm, программу на ассемблере — alarm.asm. Откройте меню Project/New Project. На экране появится окно Create new project (**Рис. 2**).

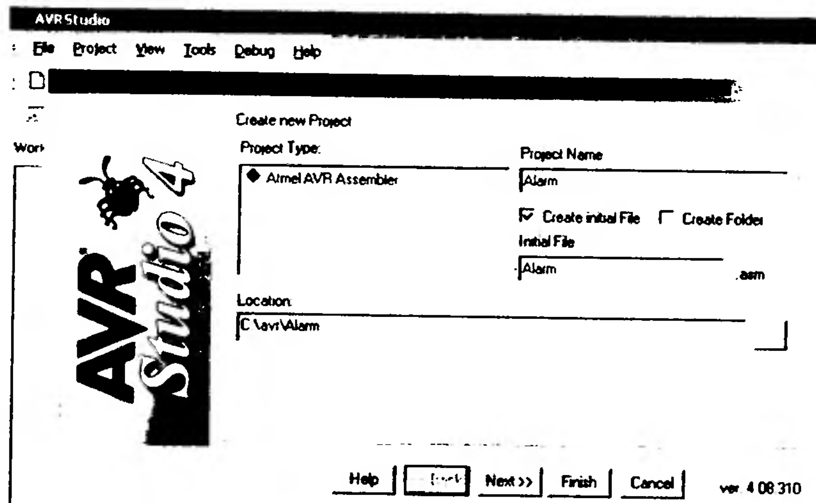


Рис. 2. Создание нового проекта

Введите имя проекта Alarm в строку ввода Project name, это же имя автоматически копируется в окошко Initial file — по умолчанию имя будет присвоено и файлу, который будет содержать программу на ассемблере (Alarm.asm).

Для определения папки, в которой будет находиться проект, щелкните по кнопке, находящейся справа от окошка Location. В открывшемся окне Select folder (**Рис. 3**) перейдите в папку c:\AVR\. В строке Current Folder допишите имя папки Alarm. После щелчка по кнопке Select подтвердите создание новой папки, нажав кнопку Да в появившемся окне. Затем, произойдет возврат в окно Create new project, а в окошке Location появится имя папки, в которой будут храниться файлы нашего проекта.

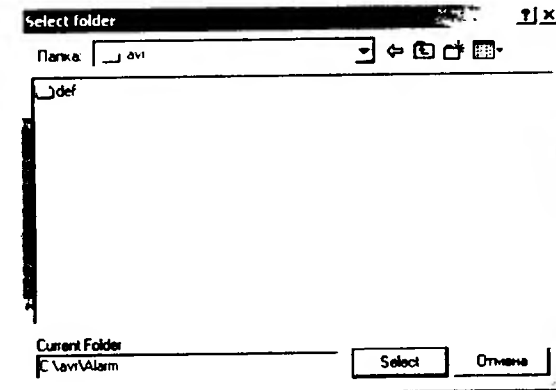


Рис. 3. Определение папки проекта

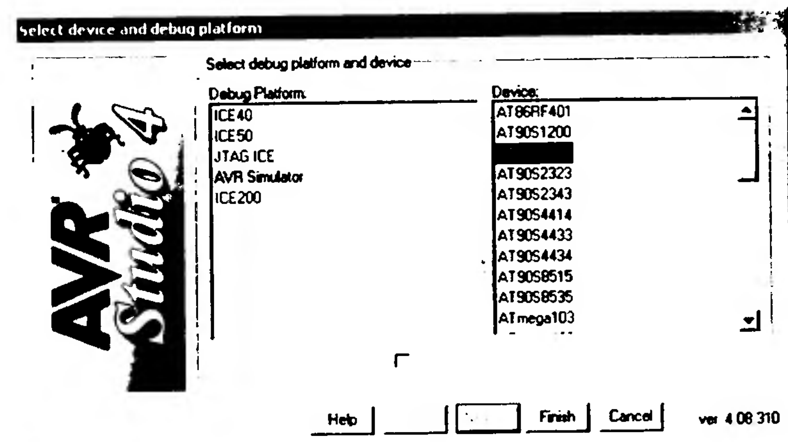


Рис. 4. Определение типа микроконтроллера

В окне Project type выберите тип проекта Atmel AVR Assembler и щелкните мышкой по кнопке Next. В открывшемся окне Select debug platform and device (**Рис. 4**) выберите в окошке Debug Platform строку AVR Simulator,

затем в окошке Device строку AT90S2313 — микроконтроллер, на котором выполнено наше устройство.

Теперь на экране активно окно C:\AVR\Alarm\Alarm.asm. В нем набирается и редактируется текст программы на ассемблере.

Ввод программы

Ввод программы следует производить в окне Alarm.asm.

Не опускайте символ точки и символ точки с запятой в начале строк: символом точки начинаются директивы ассемблера, символом точки с запятой начинаются комментарии.

Если комментарий не помещается на одной строке, его продолжение на следующей строке надо начинать символом точки с запятой.

Для работы с программой комментарии (строки с символом точки с запятой в начале) набирать не обязательно.

На **Рис. 5** показано, как выглядит окно программы при отладке.

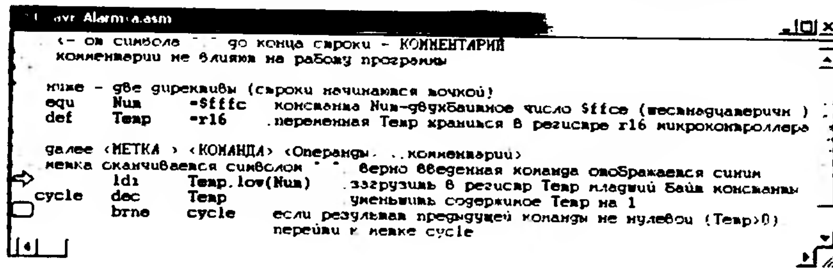


Рис. 5. Окно программы

Желтая стрелка слева от текста — это указатель отладчика, синий квадрат — маркер, он устанавливается и убирается при нажатии Ctrl + F2. Для поиска маркированных команд следует нажимать клавишу F2.

Наберите представленную ниже программу. Для удобства дальнейшего редактирования программы рекомендуется форматировать текст программы, пользуясь клавишей табуляции для отделения меток, команд с операндами и комментариев (в книге комментарии следуют непосредственно за командой с операндами).

Описание процесса отладки предполагает вашу работу за компьютером в AVR Studio с этой программой и с книгой.

Листинг программы

```
;Программа контроллера сигнализации
;=================================
.include "c:\avr\def\2313def.inc"
.def   tmp = r16
```



```
.equ   door = PD2
.equ   code = PD3
.equ   open = PD4
.equ   lock = PB2
.equ   alarm = PB1
.equ   hl = PB0
.cseg
.org 0
        rjmp   RESET              ;Reset Handler
        nop    ;rjmp EXT_INT0     ;IRQ0 Handler
        nop    ;rjmp EXT_INT1     ;IRQ1 Handler
        nop    ;rjmp TIM_CAPT1    ;Timer1 Capture Handler
        nop    ;rjmp TIM_COMP1    ;Timer1 Compare Handler
        nop    ;rjmp TIM_OVF1     ;Timer1 Overflow Handler
        nop    ;rjmp TIM_OVF0     ;Timer0 Overflow Handler
        nop    ;rjmp UART_RXC     ;UART RX Complete
                                  ;Handler
        nop    ;rjmp UART_DRE     ;UDR Empty Handler
        nop    ;rjmp UART_TXC     ;UART TX Complete Handler
        nop    ;rjmp ANA_COMP     ;Analog Comparator Handler
RESET:  ldi    tmp,low(RAMEND)    ;Определение начала
        out    SPL,tmp            ;стека в ОЗУ
        clr    tmp                ;Очистить регистр tmp (= 0)
        out    DDRD,tmp           ;Линии порта D - входы
        out    PORTB,tmp          ;На всех линиях порта В
                                  ;установить НИЗКИЕ уровни
        ser    tmp                ;Установить tmp (= $ff)
        out    PORTD,tmp          ;Установить на всех линиях
                                  ;порта D ВЫСОКИЕ уровни
        out    DDRB,tmp           ;Линии порта В - выходы
        out    PIND,tmp           ;Эта операция только для отладчика AVR
                                  ;Studio. На входных контактах порта D
                                  ;установить ВЫСОКИЕ уровни. Если не
                                  ;вставить эту команду, то при каждой новой
                                  ;отладке придется вручную устанавливать
                                  ;PIND в окне New IO View. Если PIND не
                                  ;установить, отладчик будет воспринимать
                                  ;состояние линий порта D таким, как при
                                  ;одновременном нажатии трех кнопок:
                                  ;«ДВЕРЬ», «КОДОВОЕ УСТРОЙСТВО» и
                                  ;«ОТПИРАНИЕ ИЗНУТРИ»
        ldi    tmp,15             ;Инициализация сторожевого
        out    WDTCR,tmp          ;таймера: при отсутствии команды
                                  ;сброса (wdr) в течение 2 с
                                  ;произойдет переход на прерывание
                                  ;RESET
        clt                       ;Очистка Т-флага регистра флагов
main:   wdr                       ;Сброс сторожевого таймера
        sbis   PIND,code          ;Пропуск следующей команды,
                                  ;если на линии code (PD3)
                                  ;ВЫСОКИЙ уровень
```

```
        rcall  OpenLock             ;Вызов подпрограммы OpenLock
        sbis   PIND,open            ;Пропуск следующей команды,
                                    ;если на линии open (PD4)
                                    ;ВЫСОКИЙ уровень
        rcall  OpenLock             ;Вызов подпрограммы OpenLock
        sbis   PIND,door            ;Пропуск следующей команды,
                                    ;если на линии door (PD2)
                                    ;ВЫСОКИЙ уровень
        rcall  DoorIsOpen           ;Вызов подпрограммы DoorIsOpen
        clt                         ;Очитска Т-флага регистра флагов
        rjmp   main                 ;Возврат к команде с меткой main:
;Подпрограмма управления замком
;=============================
OpenLock:
        sbi    PORTB,lock           ;Установка ВЫСОКОГО уровня
                                    ;на линии lock порта В
        rcall  d500ms               ;Четыре вызова подпрограммы
        rcall  d500ms               ;задержки на 500 мс каждая
        rcall  d500ms
        rcall  d500ms
        cbi    PORTB,lock           ;Установка НИЗКОГО уровня
                                    ;на линии lock (PB2)
        ldi    tmp,10               ;Загрузка в tmp константы 10
more:   rcall  d500ms               ;Вызов подпрограммы
                                    ;задержки на 500 мс
        dec    tmp                  ;Уменьшение содержимого tmp на 1
        brne   more                 ;Если флаг Z в регистре
                                    ;флагов не равен 1 (содержимое tmp не
                                    ;стало равным нулю),
                                    ;перейти на метку more:
        set                         ;Установить флаг Т в регистре флагов
        ret                         ;Возврат из подпрограммы
;Подпрограмма управления сиреной и светодиодом
;===========================================
DoorIsOpen:
        brts   NoClose              ;Если в регистре флагов флаг Т
                                    ;установлен, то перейти
                                    ;на метку NoClose:
        sbi    PORTB,hl             ;Установить ВЫСОКИЙ уровень
                                    ;на линии hl (PB0)
        ldi    XL,low(500)          ;Загрузить в пару регистров
        ldi    XH,high(500)         ;XH:XL константу 500
                                    ;(в XL загрузить младший
                                    ;байт числа 500,
                                    ;в XH - старший байт числа 500)
NewPls:wdr                          ;Сбросить сторожевой таймер
        sbi    PORTB,alarm          ;Установка ВЫСОКОГО уровня
                                    ;на линии alarm (PB1)
        rcall  d05ms                ;Вызов подпрограммы задержки
                                    ;на 0.5 мс
        cbi    PORTB,alarm          ;Установка НИЗКОГО уровня
```



```
                                    ;на линии alarm (PB1)
        rcall  d05ms                ;Вызов подпрограммы
                                    ;задержки на 0.5 мс
        sbiw   XL,1                 ;Вычитание единицы из
                                    ;двухбайтного слова,
                                    ;хранящегося в регистрах XH:XL
        brne   NewPls               ;Если флаг Z регистра флагов не равен
                                    ;нулю (содержимое XH:XL
                                    ;не равно нулю), перейти
                                    ;на метку NewPls:
        rcall  d500ms               ;Вызов подпрограммы задержки
                                    ;на 0.5 мс
        rjmp   DoorIsOpen           ;Возврат на метку DoorIsOpen
NoClose:
        sbis   PIND,door            ;Пропустить следующую команду,
                                    ;если на линии door (PD2)
                                    ;ВЫСОКИЙ уровень
        rjmp   NoClose              ;Переход на метку NoClose
        ret                         ;Возврат из подпрограммы
;Подпрограмма задержки на 0.5 мс
;=============================
d05ms: wdr                          ;Сброс сторожевого таймера
        ldi    YL,low(497)          ;Загрузка в YH:YL константы 497
        ldi    YH,high(497)
d05_1: sbiw    YL,1                 ;Вычитание из содержимого YH:YL
                                    ;единицы
        brne   d05_1                ;Если флаг Z<>0 (результат
                                    ;выполнения предыдущей команды
                                    ;не равен нулю), перейти
                                    ;на метку d05_1:
        ret                         ;Возврат из подпрограммы
;Подпрограмма задержки на 500 мс
;=============================
d500ms:ldi     XL,low(1000)         ;Загрузка в XH:XL
        ldi    XH,high(1000)        ;константы 1000
d500_1:rcall   d05ms                ;Вызов подпрограммы задержки
                                    ;на 0.5 мс
        sbiw   XL,1                 ;Вычитание единицы
                                    ;из содержимого XH:XL
        brne   d500_1               ;Если результат не равен нулю,
                                    ;перейти на метку d500_1:
        ret                         ;Возврат из подпрограммы
```

Рассмотрим набранную программу.

Программа начинается с директивы *.include* «имя файла», которую AVR Studio воспринимает так, как если бы вместо нее было вставлено содержимое файла. После начала отладки появится возможность просмотреть содержимое этого файла.

Ассемблирование программ

Результат ассемблирования программ может выводиться в различных форматах. По умолчанию результат сохраняется в файл *.hex, именно этот формат файла (Intel Intellec 8/MDS) может использоваться программаторами для загрузки программ в память микроконтроллеров. Если формат файла был изменен и вы не можете найти файл *.hex, следует восстановить формат.

> Для изменения формата файла, получаемого при ассемблировании, откройте меню Project/AVR Assembler Setup, в открывшемся окне AVR Assembler в окошке Additional Output file format: выберите формат Intel Intellec 8/MDS.

Для ассемблирования нашей программы можно воспользоваться меню Project/Build, кнопкой F7, кнопкой Build панели инструментов Project или, расположив курсор в окне Alarm.asm, нажать правую клавишу мышки и выбрать строкуRebuild project. Далее я буду приводить только тот вариант, которым пользуюсь сам. По мере освоения AVR Studio вы сможете выбрать варианты, наиболее подходящие вам.

Итак, для ассемблирования нажмем клавишу F7. В результате в окно Output (**Рис. 6**) выводится информация о результатах ассемблирования.

```
Creating 'C:\avr\Alarm\alarm.hex'
Creating 'C:\avr\Alarm\alarm.obj'
Creating 'C:\avr\Alarm\alarm.map'
Assembling 'C:\avr\Alarm\Alarm.asm'
Including 'c:\avr\def\2313def.inc'
Program memory usage:
Code           :  72 words
Constants (dw/db):  0 words
Unused         :  0 words
Total          :  72 words
Assembly complete with no errors.
Deleting 'C:\avr\Alarm\alarm.eep'
Build / Messages / Find in Files
```

Рис. 6. Окно Output

Интерес в этом окне представляет следующая информация:

- имя ассемблируемого файла программы;
- файлы, включенные в файл программы директивой *.include*;
- использование памяти программ микроконтроллера;
- сообщение о том, что ошибки не обнаружены.

Обнаружение ошибок при ассемблировании

Если в программе обнаружены ошибки, содержимое окна Output поможет локализовать их.

Введем ошибку в директиву *.def tmp = r16*. Имя *tmp* назначим регистру r36, которого не существует в рассматриваемых микроконтроллерах: *.def tmp = r36*. Снова ассемблируем программу (клавиша F7). В окне Output появились сведения об обнаруженных ошибках (**Рис. 7**).

Рис. 7. Окно Output со сведениями об ошибках, обнаруженных в программе

Каждой ошибке (error) соответствует сообщение, представленное одной строкой в окне Output. Например:

C:\avr\Alarm\Alarm.asm(4): error: Register r0-r31 expected.

Это значит, что в файле *C:\avr\Alarm\Alarm.asm* в строке 4 обнаружена ошибка: могут быть использованы регистры r0...r31, а мы попытались использовать r36.

Поскольку в программу директивами *.include* может быть включено несколько файлов, в каждом из которых может обнаружиться ошибка, указание полного имени файла в сообщении не является лишним.

Как быстро найти в программе строку с ошибкой? На приведенном выше рисунке представлен фрагмент окна AVR Studio с окном Output и окном программы Alarm.asm. Курсор находится в окне программы в строке с введенной нами ошибкой. Его местоположение указано в нижнем правом углу AVR Studio: Ln 4, Col 18, то есть, строка 4, колонка 18.

Нужную строку можно быстро найти, перемещая курсор в окне программы клавишами управления курсором или мышкой (переместить мышку на строку, номер которой требуется определить, и щелкнуть левой клавишей).

Почему обнаружилось так много ошибок, ведь мы ввели только одну? Теперь ошибка обнаруживается в каждой строке, содержащей имя *tmp*, которое мы попытались присвоить несуществующему регистру. Перед дальнейшей работой устраните ошибку, восстановив программу.

Определение опций симулятора

Вновь ассемблируйте программу (F7), не содержащую ошибок.

Для выполнения пошаговой отладки откроем меню Debug/Start Debugging, в окне программы слева от первой команды появится желтая стрелка, индицирующая ход выполнения отладки.

Теперь можно определить опции симулятора AVR Studio. Для этого откроем окно Simulator Options (меню Debug/AVR Simulator Options, **Рис. 8**). Заметим, что до начала отладки данная строка отсутствовала в меню Debug. Выполним первый шаг отладки, нажав клавишу F11 (меню Debug/Trace into F11).

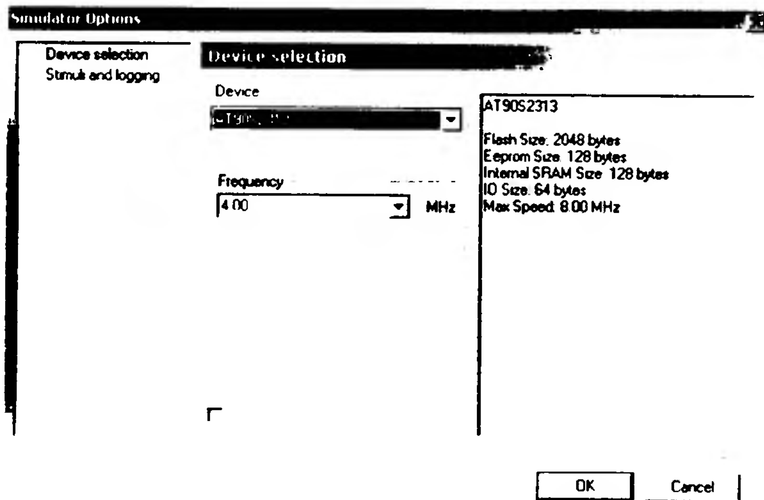


Рис. 8. Окно Simulator Options

Здесь можно определить тип микроконтроллера, для которого написана программа. Для этого щелкните мышкой по стрелке, расположенной справа в конце окошка Device, в открывшемся списке выберите микро-

контроллер AT90S2313. Тип выбранного микроконтроллера должен соответствовать файлу *def.inc, включаемому директивой *.include* в программу. Поэтому в нашу программу включен файл 2313def.inc.

В окошке Frequency таким же образом выберите частоту тактового генератора микроконтроллера из списка или введите ее вручную. Для нашего устройства эта частота равна 4 МГц и соответствует резонансной частоте кварцевого резонатора, подключенного к микроконтроллеру.

Значение частоты может быть дробным, например, 7.37 МГц, причем поддерживается лишь два знака после десятичной точки. Введенное вручную значение частоты тактового генератора автоматически ограничивается снизу частотой 0.01 МГц. Если введенная частота окажется выше допустимой для выбранного устройства, сообщение об этом появится в окне Output после нажатия кнопки OK.

*О файлах *def.inc*

Мы начали отладку, в окне программы желтая стрелка указателя отладчика установлена против первой команды: rjmp RESET. Теперь можно просмотреть содержимое файла 2313def.inc. На **Рис. 9** демонстрируется, как открыть файл *def.inc для просмотра.

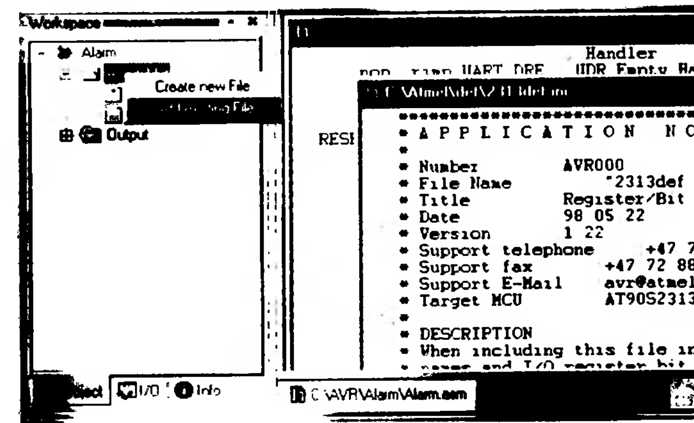


Рис. 9. Открытие файла *def.inc

Для того, чтобы открыть файл *def.inc для просмотра, добавим его в список файлов в окне Workspace, для этого щелкнем правой клавишей мышки по списку Assembler, в открывшемся окне щелкнем по строке Add Existing File, в появившемся окне Открытие файла найдем и откроем файл c:\AVR\def\2313def.inc (для поиска в окошке Тип файла надо заменить

«*.asm» на «Все файлы»). Двойным щелчком откройте добавившийся к списку Assembler файл 2313def.inc?\, на экране поверх окна программы появится окно с содержимым интересующего нас файла. Большую часть файла составляют директивы *.equ*, позволяющие присваивать имена константам, например *.equ PORTB = $18*.

Конечно, можно использовать в программе вместо имени *PORTB* константу *$18*, но удачно подобранные имена значительно облегчают чтение и понимание программ. Это же касается и директивы *.def*. Отличие состоит в том, что она позволяет назначить имена регистрам.

Замечу, что одним и тем же константам или регистрам можно присвоить несколько имен в зависимости от их назначения в том или ином блоке программы. Например, в файле 2313def.inc константе 2 назначено несколько имен: PD2, PB2, CSO2. В нашей программе директива *.equ door = PD2* назначает константе 2 имя *door*, но константа представлена не явно, а одним из своих имен.

В связи с ограниченным количеством регистров в микроконтроллере одни регистры в разных блоках программы могут использоваться для разных целей под разными именами. Так, в одном месте регистр r18 может использоваться как счетчик циклов (counter), в другом — как регистр временного хранения (store), поэтому удобно для этого регистра назначить соответствующие имена: *.def store = r18* или *.def cntr = r18*. В окне Program Output после ассемблирования программы появится предупреждение (warning) о том, что регистру уже назначено имя.

Поскольку обращение как к *store*, так и к *cntr* — это обращение к регистру r18, надо избегать конфликтов при их совместном использовании, ведь изменение содержимого *store* автоматически изменяет содержимое *cntr*.

Блок ссылок на обработчики прерываний

Обработчики прерываний входят в состав программы микроконтроллера и могут размещаться в любой ее части. Поэтому для обнаружения начала обработчика конкретного прерывания в программе его адрес заносится в фиксированную ячейку памяти программ.

В предложенном варианте программы только одно прерывание — *RESET*. Адрес подпрограммы, обрабатывающей прерывание *RESET*, всегда хранится в ячейке памяти программ с адресом *0x0000*.

Обработчик этого прерывания выполняется после изменения уровня с НИЗКОГО на ВЫСОКИЙ на контакте RESET микроконтроллера. В нашей схеме это происходит после подачи питания на микроконтроллер или после отпускания кнопки SA4 с некоторой задержкой (см. описание схемы).

Чтобы обозначить начало обработчика, в программе ставится метка (в нашей программе это метка *RESET:*), а ее адрес надо записать в нулевую

ячейку памяти программ. В программе это записывается в нулевой ячейке памяти так: *rjmp RESET*. Директива *.cseg* указывает компилятору, что код программы должен быть помещен в кодовый сегмент памяти (память программ); директива *.org 0* сообщает, что код следует размещать, начиная с нулевого адреса указанного сегмента памяти.

Каждый тип микроконтроллеров AVR имеет индивидуальный набор прерываний (для микроконтроллера AT90S2313 их одиннадцать).

Итак, в нулевой ячейке хранится адрес начала обработчика *RESET:*, в следующих десяти ячейках в строгой последовательности должны храниться адреса оставшихся десяти прерываний микроконтроллера AT90S2313. Так как мы не используем эти прерывания, вместо адресов помещаются пустые команды nop. Однако в тексте приведенной программы за каждой командой nop следует закомментированная ссылка (перед ней ставится точка с запятой) на подпрограмму обработки прерывания. Так, если понадобится использовать прерывание от аналогового компаратора, следует убрать последний оператор *nop* и точку с запятой за ним, тогда в этой строке окажется команда *rjmp ANA_COMP*. В программу понадобится добавить метку *ANA_COMP:*, которая обозначит начало подпрограммы обработки прерывания от аналогового компаратора. Завершить подпрограмму обработки прерывания надо командой *reti*.

Как узнать, в какой последовательности и сколько прерываний должно быть в блоке?

Такой блок можно найти в полном техническом описании соответствующего микроконтроллера в разделе /Interrupts/Interrupt Vectors/ или /Architectural Overview/Reset and Interrupt Handling либо в конце файла *def.inc.

Можно скопировать блок из раздела /Interrupts/Interrupt Vectors/ или /Architectural Overview/Reset and Interrupt Handling полного технического описания микроконтроллера (файл pdf) и вставить его в программу, убрав затем адреса, стоящие в начале каждой строки ($xxx). Вместо не используемых в программе прерываний надо вставить команды *nop*.

```
$000   rjmp RESET        ;Reset Handler
$001   rjmp EXT_INT0     ;IRQ0 Handler
$002   rjmp EXT_INT1     ;IRQ1 Handler
$003   rjmp TIM_CAPT1    ;Timer1 Capture Handler
$004   rjmp TIM_COMP1    ;Timer1 Compare Handler
$005   rjmp TIM_OVF1     ;Timer1 Overflow Handler
$006   rjmp TIM_OVF0     ;Timer0 Overflow Handler
$007   rjmp UART_RXC     ;UART RX Complete Handler
$008   rjmp UART_DRE     ;UDR Empty Handler
$009   rjmp UART_TXC     ;UART TX Complete Handler
$00a   rjmp ANA_COMP     ;Analog Comparator Handler
```

Для нашего микроконтроллера AT90S2313 в конце файла 2313def.inc можно найти такой блок:

```
.equ   INT0addr = $001   ;External Interrupt0 Vector Address
.equ   INT1addr = $002   ;External Interrupt1 Vector Address
.equ   ICP1addr = $003   ;Input Capture1 Interrupt Vector Address
.equ   OC1addr  = $004   ;Output Compare1 Interrupt Vector Address
.equ   OVF1addr = $005   ;Overflow1 Interrupt Vector Address
.equ   OVF0addr = $006   ;Overflow0 Interrupt Vector Address
.equ   URXCaddr = $007;UART Receive Complete Interrupt Vector
                                       ;Address
.equ   UDREaddr = $008;UART Data Register Empty Interrupt Vector
                                       ;Address
.equ   UTXCaddr = $009;UART Transmit Complete Interrupt Vector
                                       ;Address
.equ   ACIaddr = $00a;Analog Comparator Interrupt Vector
                                       ;Address
```

Здесь отсутствует лишь самое первое прерывание RESET, размещаемое по нулевому адресу. Но такой блок надо приводить к описанному выше виду.

Основная программа

Основная программа начинается командой с меткой RESET:.

Первая пара операторов определяет адрес начала стека в оперативной памяти микроконтроллера. Работу стека мы еще рассмотрим, поэтому сейчас заметим, что стек — это область оперативной памяти или оперативного запоминающего устройства микроконтроллера (далее — ОЗУ). Стек предназначен в основном для обслуживания подпрограмм; при занесении в стек нескольких величин адрес каждой последующей величины уменьшается (у первой — самый большой адрес, у последней — самый маленький). Поэтому стек обычно располагают в конце ОЗУ, начиная с последней ячейки ОЗУ.

Мы прервались после первого шага отладки: желтая стрелка указателя отладчика установлена против первой команды *rjmp RESET.*

Откройте окно просмотра состояния процессора (в окне Workspace двойной щелчок по строке Processor). Далее слова «просмотр состояния» будут опускаться.

В окне процессора указана выбранная нами тактовая частота 4 МГц; в окошках счетчика тактов микроконтроллера (Cycle Counter), указателя стека (Stack Pointer), счетчика команд (Program Counter), времени выполнения (Stop Watch) — нули.

На **Рис. 10** показано окно Workspace, в котором отображается состояние процессора во время отладки программы.

Откройте окно памяти (меню View/Memory, **Рис. 11**).

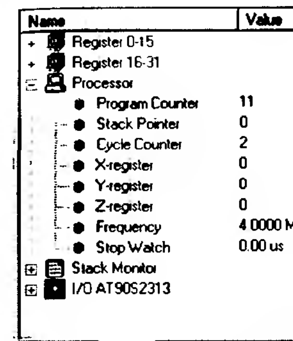


Рис. 10. Состояние процессора (окно Workspace)

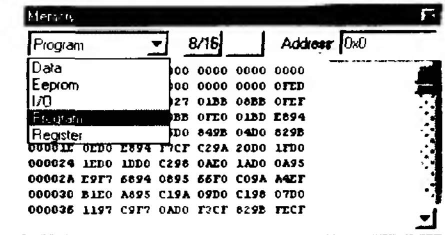


Рис. 11. Окно состояния памяти с открытым меню выбора типа памяти

Кнопкой 8/16 выберите отображение в виде двухбайтных слов, как на рисунке.

Заметьте, что в нулевой ячейке памяти программ находится число 0AC0 (правильнее было бы написать $0AC0 или 0x0AC0, что указывает на шестнадцатеричное представление величины, однако, вид записи соответствует виду, представленному в описываемом окне).Откроем окно Disassembler (**Рис. 12**), выбрав меню View/Disassembler.

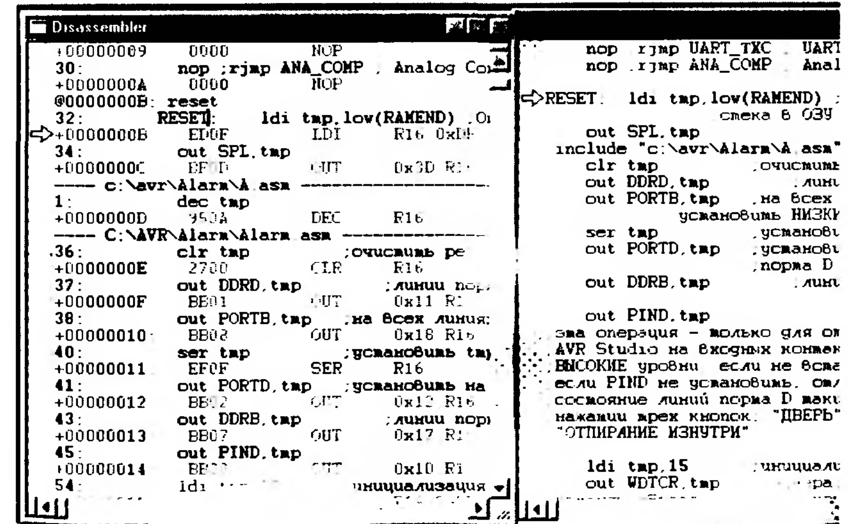


Рис. 12. Окно дизассемблированного кода и окно программы

Открывшееся окно Alarm закрывает собой окно программы Alarm.asm, поэтому его лучше сместить, как это показано на **Рис. 12**.

В первой колонке окна Alarm находится адрес двухбайтной ячейки памяти программ, во второй колонке — код команды, хранящейся в этой ячейке, в третьей — наименование команды, такое же, как и в нашей программе, в четвертой — операнды, приведенные к приемлемому для микроконтроллера виду. Например, вместо имени *tmp* здесь будет имя регистра *R16*, вместо метки *RESET:* — смещение адреса и так далее. В пятой колонке находятся комментарии. Здесь могут быть представлены, например, результаты адреса переходов или константы в шестнадцатеричном, двоичном и десятичном видах.

По существу, в этом окне представлен код программы в дизассемблированном виде.

Во второй колонке строки с адресом *+00000000:* — тот же код *C00A*, только с измененной последовательностью байтов, который находится в нулевой ячейке памяти программ. Это двухбайтный код, соответствующий команде *rjmp RESET. C0* — код команды *rjmp*, *0A* — смещение адреса, которое надо добавить к содержимому счетчика команд для перехода к обработчику прерывания, начинающемуся меткой *RESET:*.

В результате выполнения этой команды в счетчике окажется адрес 0A, для перехода к следующей команде содержимое счетчика увеличится на единицу, в результате следующей выполнится команда, расположенная по адресу 0B (смотрите пятую колонку строки).

С адреса *00000001* по адрес *0000000A* следуют коды *0000*, соответствующие нашим пустым командам nop, и так далее.

Вероятно, читатель уже уловил связь между строкой в программе и кодом, исполняемым микроконтроллером.

Новый шаг отладки вызвал бы перемещение указателя отладчика к следующей команде как в окне программы, так и в окне Disassembler.

Не многовато ли для первого шага отладки?

Если материал для вас пока не очень понятен — не расстраивайтесь, просто запомните, что вопрос о такой-то возможности отладчика рассматривался. Тогда, при необходимости, вы сможете вернуться к этой части книги. Поэтому дать максимум материала по отладке целесообразно именно при рассмотрении простейшей программы, чтобы не возвращаться к нему при рассмотрении более сложных программ.

Вернемся к программе, закроем окна Memory и Disassembler.

Выполним второй шаг отладки, нажав клавишу F11. Далее подобный текст будет заменен фразами «выполним F11:», «нажмите F11:» или просто «F11:».

В окне Processor состояние счетчика стало равным *$B*, что соответствует адресу следующей команды, в счетчике тактов — 2, то есть команда *rjmp*

выполнена за два такта микроконтроллера, в счетчике времени — 50 мкс, что соответствует двум тактам микроконтроллера при частоте 4 МГц.

Константа *RAMEND* в следующей команде — это адрес последней ячейки ОЗУ. Значение этой константы ($DF), а также других констант, которые не определены в самой программе директивой *.equ*, можно найти в файле 2313def.inc.

Функция *low(RAMEND)* выбирает младший байт из константы *RAMEND*. Для определения адреса последней ячейки ОЗУ микроконтроллера AT90S2313 достаточно одного байта, поэтому указатель стека представлен только одним регистром *SPL* и используется только младший байт адреса, а старший байт равен нулю.

Описание работы самих команд ищите в разделе Instruction Set Summary полного технического описания микроконтроллера.

Откройте меню View/Watch (**Рис. 13**), щелкните правой кнопкой мышки в появившемся окне, выберите строку Add Item. В окне появится строка ввода в колонке Watch. Введите в нее имя переменной *tmp*. Откройте меню View/Registers и посмотрите, что находится в регистре R16 — его содержимое соответствует значению переменной *tmp*, так как переменная *tmp* хранится в регистре R16.

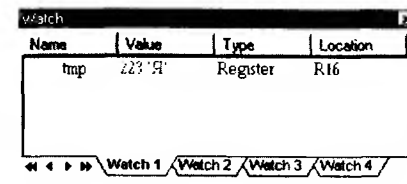


Рис. 13. Окно Watch

Удобно одновременно просматривать изменения сразу в нескольких окнах, для этого надо тщательно устанавливать их размер и выбирать область просмотра. Например, окно Watch можно значительно сузить, перемещая влево вертикальные линии, разделяющие колонки. Для этого установите указатель мышки на вертикальную линию между заголовками колонок, например между Name и Value, при этом должна появиться черная вертикальная полоска со стрелками влево/вправо, нажмите левую клавишу и, не отпуская ее, переместите линию влево.

В окне IO убраны строки Data Direction для портов D и B, для этого надо щелкнуть мышью по строкам, которые вы хотите убрать, нажать правую клавишу мышки и выбрать строку Hide. Для восстановления скрытой строки надо нажать правую клавишу мышки и выбрать строку Show, а из появившегося списка выбрать наименование строки, которую следует восстановить.

На **Рис. 14** представлено окно AVR Studio с окнами Workspace, Watch, Register, Memory, наиболее часто использующимися при отладке нашей программы. Окна могут быть размещены по усмотрению пользователя или стыковаться друг с другом, установить/сбросить соответствующее свойство окна можно щелкнув по нему правой клавишей мышки и выбрав строку Docking View в открывшемся окне (на рисунке это серое окно посередине снизу). Здесь же можно выбрать шрифт текста (строка Font) или выбрать представление чисел (десятичное/шестнадцатеричное). Разные окна обладают различными свойствами, проверьте работу этих свойств самостоятельно.

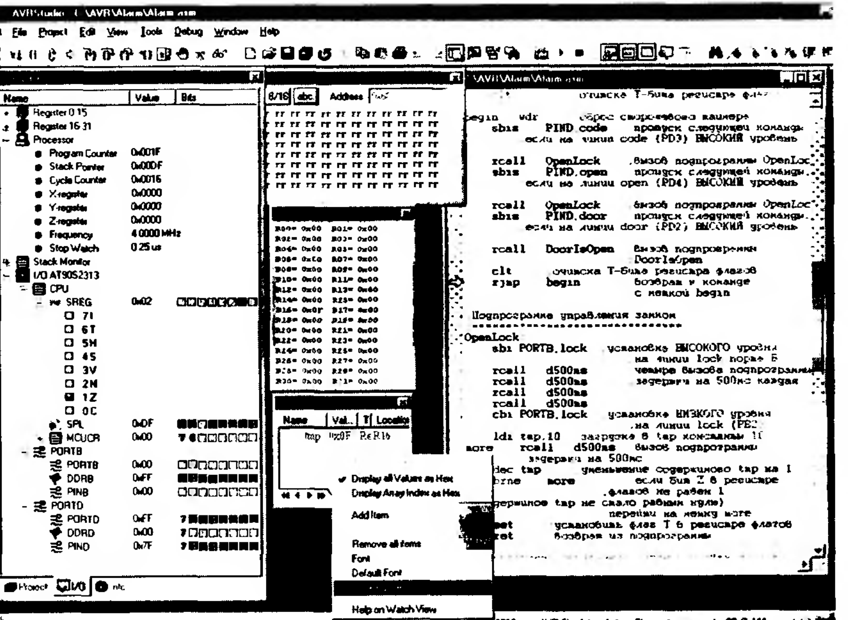


Рис. 14. Размещение окон в AVR Studio

Нажмите F11, в регистре R16 и в переменной *tmp*, как и ожидалось, одинаковые значения $DF. Окно Register можно закрыть.

В окне Proceessor есть строка StopWatch, двойным щелчком можно обнулить ее содержимое, что позволяет использовать это свойство по принципу секундомера для определения времени, необходимого для выполнения частей программы.

Нажимаем F11: в окне Processor указатель стека приобретает значение $DF.

Еще F11: переменная *tmp* очистилась (стала равной нулю). При нулевом результате команды (что бы это ни было — вычитание, сложение, очистка, сдвиг и так далее) установился флаг Z (смотри панель Flags в окне Processor).

На панели StopWatch обнаруживаем, что команды *out* и *clr*, перед которыми была нажата кнопка Clear, выполнены за 0.5 мкс.

Организация портов ввода/вывода

Прежде чем продолжать рассмотрение программы, думаю, уместно рассмотреть, как работает порт ввода/вывода на примере функционирования линии PB2 микроконтроллера AT90S2313; так как эта линия не обременена альтернативными возможностями, ее схема наиболее проста.

В состав одного порта, в зависимости от типа микроконтроллера, может входить до восьми линий ввода/вывода. Так, порт D микроконтроллера AT90S2313 содержит 7 линий (PD0...PD6), а порт B — 8 линий (PB0...PB7).

На **Рис. 15** представлена функциональная схема, обеспечивающая работу линии порта PB2.

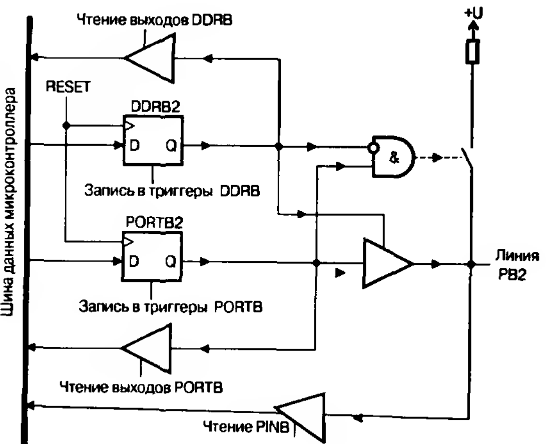


Рис. 15. Схема работы линии PB2 порта ввода/вывода

8-разрядная шина данных микроконтроллера — это восемь линий, по которым данные могут передаваться между различными узлами микроконтроллера.

Рассматриваемая линия PB2 связана лишь с одной из линий шины.

Под действием положительного импульса «ЗАПИСЬ В ТРИГГЕРЫ PORTB» информация, установленная на шине данных, записывается в восемь триггеров PORTB (в порт PORTB), в том числе и в триггер PORTB2.

Команда *out PORTB,Rr* вызывает запись в восемь триггеров порта B содержимого регистра Rr (Rr — любой регистр общего назначения микроконтроллера).

Для записи единицы в триггер PORTB2 хранящееся в регистре число должно иметь вид 0bxxxxx1xx, для записи нуля — 0bxxxxx0xx. Интересующий нас второй разряд находится не на второй, а на третьей позиции, так как нумерация разрядов начинается с нуля. Здесь x — любое значение (оно не повлияет на запись в триггер PORTB2), 0b — признак двоичного представления числа, так же, как 0x или $ — признаки шестнадцатеричного представления, а отсутствие символов перед числом — признак десятичного представления. Вспомните окно Alarm, вызванное по нажатию Ctrl + F11: в пятой колонке с комментариями можно было встретить соответствие между шестнадцатеричным, двоичным и десятичным представлением одного числа.

Под действием положительного импульса «ЗАПИСЬ В ТРИГГЕРЫ DDRB», информация, установленная на шине данных, одновременно записывается в восемь триггеров DDRB (в порт DDRB), в том числе и в триггер DDRB2.

Записи в порт DDRB соответствует команда *out DDRB,Rr.*

Данные, хранящиеся в триггерах PORTB, могут быть считаны на шину данных через соответствующие буферы под действием положительного импульса «ЧТЕНИЕ ВЫХОДОВ PORTB». Этой операции соответствует команда *in Rr,PORTB.* Результат будет занесен в регистр Rr, состояние второго разряда регистра будет соответствовать информации, записанной в триггер PORTB2.

Буферы на функциональной схеме изображены в виде треугольников. Можно рассматривать треугольник как стрелку, указывающую направление передачи. Передача происходит только при ВЫСОКОМ уровне на управляющей линии (линия, подведенная к треугольнику сверху или снизу).

Данные, хранящиеся в триггерах DDRB, могут быть считаны на шину данных через соответствующие буферы под действием положительного импульса «ЧТЕНИЕ ВЫХОДОВ DDRB». Этой операции соответствует команда *in Rr,DDRB.* То есть не обязательно помнить, какие данные записывались в PORTB или в DDRB, их всегда можно считать из этих портов.

Уровни, присутствующие на линиях PB, в том числе и PB2, могут быть считаны на шину данных через соответствующие буферы под действием положительного импульса «ЧТЕНИЕ PINB». Этой операции соответствует команда *in Rr,PINB.*

Итак, для работы с портом B программа должна обращаться к трем портам PORTB, DDRB и PINB.

Существенный момент: порт PINB не снабжен триггерами, поэтому запись в этот порт не приводит к изменениям физических состояний схемы, однако такую виртуальную запись удобно использовать при отладке программы.

Посмотрим, что будет происходить при различных уровнях на выходах Q триггеров DDRB2 и PORTB2. Для этого составим таблицу, отражающую состояние DDRB2, not DDRB2 (для учета инверсии на входе схемы &), PORTB2, & (выход схемы &), резистора и PB2 (**Табл. 1**).

Таблица 1. **Влияние выходных уровней триггеров DDRB2 и PORTB2 на состояние линии PB2**

DDRB2	not DDRB2	PORTB2	&	Резистор	PB2
0	1	0	0	отключен	вход, большое входное сопротивление
0	1	1	1	подключен	вход, при подаче НИЗКОГО уровня через резистор течет ток
1	0	0	0	отключен	выход, НИЗКИЙ уровень
1	0	1	0	отключен	выход, ВЫСОКИЙ уровень

Из таблицы видно, что при ВЫСОКИХ уровнях на выходах триггеров DDRB2 и PORTB2 на линии PB2 также присутствует ВЫСОКИЙ уровень. Чтение как порта PORTB, так и порта PINB даст единицу во втором разряде. Но если на контакт микроконтроллера PB2 подать НИЗКИЙ уровень (попросту соединить его с общим проводом), то чтение PORTB даст единицу в том же разряде, а чтение PINB даст ноль во втором разряде.

Любую линию порта можно организовать как вход или как выход:

1. Линия порта — вход, если в DDRBn (n — номер линии) установить НИЗКИЙ уровень (при НИЗКОМ уровне в PORTBn входное сопротивление велико; при ВЫСОКОМ уровне в PORTBn вход соединяется через внутренний резистор с цепью питания микроконтроллера, при отсутствии внешнего сигнала на контакте микроконтроллера PBn присутствует ВЫСОКИЙ уровень).
2. Линия порта — выход, если в DDRBn установить ВЫСОКИЙ уровень (при НИЗКОМ уровне в PORTBn на контакте микроконтроллера PBn присутствует НИЗКИЙ выходной уровень; при ВЫСОКОМ уровне в PORTBn возможен ввод внешних данных через контакт PBn при подсоединении к нему мощного внешнего источника сигнала).

Заметим, что все сказанное о порте В относится к любому другому порту ввода/вывода. Для порта D, например, наименования портов в программе изменятся на PORTD, DDRD и PIND. Аналогичное изменение наименований надо делать для портов А и С при использовании микроконтроллеров, имеющих эти порты.

Надо заметить, что аппаратное прерывание RESET автоматически обнуляет триггеры портов ввода/вывода.

Продолжение отладки программы

Переменная *tmp* обнулена. В окне Workspace раскрываем PORTB и PORTD, щелкнув по знаку плюс слева от них.

Дважды нажимаем F11: состояние PORTB и PORTD в окне Workspace не изменилось, так как порты, которые инициализировались командами *clr tmp*, *out DDRD,tmp* и *out PORTB,tmp*, уже были обнулены отладчиком, а в реальной схеме были обнулены аппаратным прерыванием RESET. Поэтому перечисленные команды можно опустить, но их присутствие позволит сделать некоторые комментарии.

Трижды нажимаем F11: в переменной *tmp* появилась величина 0xFF или 0b11111111.

В Workspace/DDRB и PORTD установлены все флажки. Далее такой записи будет соответствовать «все разряды DDRB и PORTD установлены».

Нажать F11: все разряды PIND установлены. Это физически бессмысленная операция, об этом говорилось выше. Однако не установленные PIND в окне Workspace эквивалентны одновременному нажатию кнопок SA1, SA2 и SA3 схемы контроллера сигнализации (НИЗКИЕ уровни на входных линиях DOOR, CODE и OPEN). Этим способом установить линии проще, чем вручную при каждой новой отладке, щелкая мышкой в окошках для установки галочек.

Нажимаем дважды F11: произошла инициализация сторожевого таймера (Watchdog Timer).

Сторожевой таймер защищает микроконтроллер от зависания. Его работа не зависит от тактового генератора микроконтроллера, она определяется отдельным внутренним генератором микроконтроллера. Если программно не определить режим работы сторожевого таймера, он будет вызывать прерывание RESET через каждые 16 мс при напряжении питания +5 В (частота генератора сторожевого таймера сильно зависит от напряжения питания). Периодичность прерывания может быть задана программно, мы определили период повторения прерываний примерно в 2 с при напряжении питания +5 В. Сторожевой таймер можно и отключить, но это вряд ли понадобится.

Команды сброса сторожевого таймера *wdr* расставляются в программе с таким расчетом, чтобы сторожевой таймер не успевал сбросить микроконтроллер.

Если программа зависает (останавливается, зацикливается), очередная команда сброса остается невыполненной, происходит сброс микроконтроллера, программа запускается сначала.

Нажимаем F11:, командой *clt* очищается флаг Т регистра флагов. Во флаг Т будет заноситься информация о том, разрешено или запрещено отпирание двери.

Далее в бесконечном цикле, начинающемся меткой *main:* и заканчивающемся командой *rjmp main* происходят следующие события:

1. В начале каждого цикла сбрасывается сторожевой таймер командой *wdr.*
2. Проверяется состояние линии CODE командой *sbis PIND,code.*
3. Когда на линии CODE контроллера ВЫСОКИЙ уровень, пропускается одна команда, следующая за командой sbis (обращайте внимание на наименование команд — они связаны с определяемыми командой действиями. Так, sbis — это Skip if Bit in IO register Set, то есть пропустить <следующую команду>, когда разряд в регистре ввода/вывода установлен).
4. То же происходит при проверке состояний линий OPEN и DOOR.
5. Если на какой-то из перечисленных линий НИЗКИЙ уровень (замкнута соответствующая кнопка), следующая команда вызова подпрограммы *rcall* выполняется, происходит вызов подпрограммы.
6. Предпоследняя команда *clt* очистит флаг Т, если он был установлен в какой-либо подпрограмме.

Нажмите F11 восемь—десять раз: понятно, что цикл повторяется. Обратите внимание, за какое число тактов команда *sbis* выполняется при выполнении условия (проверяемый разряд порта установлен, следующая команда будет пропущена). Для этого наблюдайте изменение состояния Cycle Counter в окне Processor.

Переместитесь с помощью вертикального скроллинга в начало программы и посмотрите, какому разряду или контакту порта PIND соответствует линия code (она соответствует контакту PD3). Сбросьте флажок в третьем разряде PIND (в окне Workspace убрать флажок с цифры 3).

Установите курсор на команде *sbis PIND,code* и нажмите Ctrl + F10 (меню Debug/Run to cursor): вы сразу перешли к команде, на которой был установлен курсор. Сбросив 3-й разряд PIND, вы имитировали нажатие кнопки SA2. При нажатии F11 проследите, за какое число тактов выполнится команда *sbis* при невыполнении условия (проверяемый разряд в порте не установлен, следующая команда не пропускается).

> По мере освоения отладчика рассмотрим эффекты от использования горячих клавиш (F10, F11, Ctrl + F10 и так далее), функции большинства из них отражены в меню Debug. Открывая это меню, можно использовать мышь вместо клавиатуры, а также ознакомиться с командами отладчика и соответствующими им горячими клавишами.

Часто бывает необходимо определить точное число тактов, требующееся для выполнения последовательности команд. В списке команд микроконтроллера указывается число тактов, за которое выполняется каждая из них, однако для части команд — команд ветвления — существуют варианты, зависящие от того, выполняется условие, проверяемое командой, или нет.

В предыдущем примере с помощью отладчика было проверено число тактов, требующееся для выполнения команды *sbis* при выполнении условия (2 такта) и при его невыполнении (1 такт). Рекомендую сделать соответствующие пометки в списке команд, а при отладке самостоятельно проверить некоторые из команд ветвления на предмет числа тактов, требующихся для их выполнения.

Указатель отладчика переходит на команду, вызывающую подпрограмму отпирания замка: *rcall OpenLock*.

Снова откроем окно Memory. Если вы не закрывали его, перейдите в режим просмотра данных Data (если выбран просмотр памяти Program, надо щелкнуть по окощку со словом Program, в открывшемся списке выбрать Data (Рис.11)). В соседнем окошке — адрес начала памяти данных 0x0060 в адресном пространстве микроконтроллера. Адрес последней ячейки памяти 0x00DF, это наша константа *RAMEND*, а также адрес начала стека.

Заметьте величины, хранящиеся в Stack Pointer и Cycle Counter окна Processor, а также в последней ячейке окна Memory, затем нажмите F11.

Посмотрите, как изменились названные величины: для перехода в подпрограмму потребовалось 3 такта микроконтроллера, в указателе стека (Stack Pointer) хранится адрес 0x000000DD (или 0x00DD, или просто 0xDD). Этот адрес на 2 меньше предыдущего адреса 0xDF, то есть адрес возврата из подпрограммы занимает 2 ячейки памяти, их адреса 0xDE и 0xDF, в них хранится адрес 001A команды, которая будет выполнена сразу после возврата из подпрограммы.

Следите за изменением состояния порта B в окне Workspace.

Нажмите F11: установился второй разряд PORTB (PORTB2), в следующем такте микроконтроллера (при выполнении следующей команды, уже не связанной с портом B) этот же разряд установится и на линии микроконтроллера PINB2, следовательно, и на контакте PB2. При этом транзистор VT1 откроется, через соленоид замка потечет ток, замок откроется.

Нам надо, чтобы замок был открыт в течение 2 с. Для тонового сигнала сирены мы организуем подпрограмму с задержкой 0.5 мс (это половина периода сигнала частотой 1 кГц, необходимой для сирены). Для организации пауз в звучании сирены организуем подпрограммы задержки на 500 мс, вызывая 1000 раз подпрограмму задержки на 0.5 мс, для 2-секундной и 5-секундной задержек вызовем задержку на 500 мс соответственно 4 и 10 раз.

Нажимаем F11: произошел переход в подпрограмму задержки на 500 мс. Продолжаем следить за работой стека. Содержимое Stack Pointer (окно Processor) уменьшилось на 2 и стало равно 0xDB, в ячейки памяти 0xDC и 0xDD помещен адрес возврата из подпрограммы 0022 (окно Memory).

Откройте окно Watches (меню View/Watch) и добавьте две переменные XH и XL (так определены регистры r27 и r26 в файле *def.inc).

Откроем View/Disassembler: в окне Alarm за указателем отладчика следуют две команды: *ldi R26,0xE8* и *ldi R27,0x03*.

Десятичная запись числа 1000 соответствует шестнадцатеричному представлению 0x03E8; старший байт числа 0x03 загружается в регистр *R27*, младший — в *R26*. Эта пара регистров именуется соответственно XH:XL или регистр X. В микроконтроллере есть еще две пары YH:YL (R28:R29), или регистр Y, и ZH:ZL (R30:R31), или регистр Z (не путать с флагом Z в регистре флагов).

Приятное отличие парных регистров состоит в том, что командой *adiw* к двухбайтному слову, хранящемуся в паре регистров, можно добавить константу, а командой *sbiw* — вычесть константу. Команды обращения к памяти данных (ОЗУ) также не обходятся без них, а регистр Z обслуживает и обращение к памяти программ.

Вернемся к окну программы Alarm.asm.

Дважды нажимаем F11, загружая число циклов 1000 в регистр X. Снова нажимаем F11, наблюдая работу стека (Stack Pointer = 0xB9, адрес возврата 0045). Теперь мы вошли в подпрограмму задержки на 0.5 мс. Трижды нажимаем F11: сторожевой таймер сброшен, в пару регистров YH:YL загружено число циклов 497.

Как рассчитывается число циклов? Для начала определяем, что будет в цикле. В нашем цикле всего две команды: *d05_1: sbiw YL,1* и *brne d05_1*. Цикл будет повторяться, пока содержимое Y не станет равным нулю.

Первая команда вычитания единицы из содержимого регистра Y выполняется за два такта. Вторая команда проверяет состояние флага Z в регистре флагов. Пока результат предыдущей операции не равен нулю (для этого фрагмента программы — пока содержимое Y не достигло нуля, то есть YH = 0 и YL = 0), флаг Z не установится, произойдет переход на метку *d05_1:*, команда выполнится за 2 такта. После установки флага Z выполнение цикла завершится, команда будет выполнена за 1 такт.

Итак, для получения общего числа тактов, обеспечивающих задержку на 0.5 мс, надо просуммировать следующее число тактов:

- 3 такта команды *rcall* (вход в подпрограмму);
- 3 такта команд *wdr, ldi, ldi* в начале подпрограммы;
- 4 такта команды *ret* возврата из подпрограммы;
- 4 такта·(число циклов) – 1 (в последнем цикле команда *brne* выполняется за 1 такт).

С другой стороны, общее число тактов, обеспечивающих задержку на 0.5 мс при частоте тактового генератора 4 МГц, составляет 4000000 [Гц] × 0.0005 [с] = 2000 тактов.

Составим уравнение: 3 + 3 + 4 + (4·N) – 1 = 2000, где N — искомое число циклов. Отсюда N = 1991/4, результат получится с остатком. Возьмем ближайшее меньшее делимое 1988, тогда число циклов N = 497, а недостающие до 1991-го три такта (1991...1988) мы могли бы добавить, вставив три пустые команды перед циклом.

Аналогичным образом можно было бы точно рассчитать число циклов для обеспечения задержек в 500 мс (подпрограмма d500ms). Но тогда пришлось бы сократить время выполнения подпрограммы d05ms (задержка на 0.5 мс) с тем, чтобы учесть время на ее вызов из подпрограммы d500ms. Там же, где нужно точное время выполнения подпрограммы d05ms, компенсировать сокращенное время вводом пустых команд перед вызовом этой подпрограммы.

В окно Watches добавьте переменные YH и YL.

Нажмите 10...15 раз F11, следя за изменением этих переменных. В окне Processor установите мышкой флаг Z регистра флагов, нажмите F11 пару раз: флаг Z сбросился после команды *sbiw*, поскольку результат оказался ненулевым.

В окне Watches измените значение, хранящееся в YL, для этого выполните двойной щелчок мышкой напротив переменной YL в колонке Value, измените находящееся там значение на 0x01 (достаточно нажать клавишу «1», затем клавишу «ENTER»). Теперь сделайте одно или два нажатия F11 для однократного выполнения команды *sbiw:* YL = 0, но установки флага Z (окно Processor) не произошло. Хотя команда записывается как *sbiw YL,1* (именно YL!), оценка результата производится по общему содержимому YH:YL (01:00).

Следя за изменением YH и YL, дважды нажмите F11.

Теперь замените содержимое YL на единицу (0xFF на 0x01). Сделайте одно или два нажатия F11 для однократного выполнения команды *sbiw:* флаг Z установился, так как Y = 0. Нажав F11, покидаем цикл.

Вы, конечно, заметили, что значение переменной можно изменять по своему усмотрению в окне Watches в процессе отладки.

Следующая команда, на которой остановился указатель, *ret* — возврат из подпрограммы на команду, адрес которой 0045 загружался в стек последним (окно Memory, ячейки памяти данных 0x00DA и 0x00DB).

Перейдем в окно Disassembler, нажав F11, увидим переход на команду с названным адресом. Содержимое Stack Pointer возросло на 2 и теперь в стеке только два адреса возврата из подпрограмм, вызванных ранее: 0022 и 001A. Адрес 0045 не изменился, но он уже не имеет значения, так как находится вне пределов стека.

Вернемся к программе. Нажимая клавишу F10 (не F11!), снова доберемся до команды *rcall d05ms*. В окне Workspace/Processor двойным щелчком очистим содержимое Stop Watch, заметим содержимое Cycle Counter, теперь нажмем F10, чтобы выполнить подпрограмму *d05ms*, не заходя в нее.

На выполнение подпрограммы ушло 499.25 мкс вместо 500 мкс (0.5 мс) или 1997 тактов микроконтроллера вместо 2000, что ровно на три такта меньше, как и получилось при расчете числа циклов, проведенном выше (напомню, что три команды nop не добавлялись). Проверьте указанное время и число тактов в соответствующих окошках окна Processor.

Далее подпрограмма *d05ms* будет вызываться из подпрограммы *d500ms* множество раз, интереса это не представляет, поэтому установим курсор на команду *ret*, последнюю в подпрограмме *d500ms*.

Теперь нажмем Ctrl + F10 и дождемся перемещения указателя отладчика к команде, на которую установлен курсор. На компьютере с невысокой производительностью это может занять несколько минут. Если вы обладатель такого компьютера и ожидание затянулось — нажмите Ctrl + F5 для остановки, замените в окне Watches переменные YH и XH на нули, а YL и XL на единицы, тогда вы сможете быстро покинуть подпрограмму, снова установив курсор на команде *ret* подпрограммы *d500ms* и нажав Ctrl + F10.

> В подобных случаях на время отладки удобно сократить число циклов или временно вставить перед первой командой подпрограммы команду возврата ret. Важно не забыть восстановить программу перед ассемблированием ее окончательного варианта.

Нажав F11, покидаем подпрограмму *d500ms*, проверяем, как изменился указатель стека и в каких ячейках памяти хранился адрес возврата из подпрограммы.

Чтобы не затягивать ожидание, добавьте команду *ret* сразу после метки *d500ms:*, после ввода команды обязательно нажмите клавишу ENTER.

Пометим маркером команду *main: wdr*. Для этого установим курсор в строке с этой командой и нажмем Ctrl + F2, слева от строки появится синий квадрат маркера.

> Установка и удаление маркеров производятся нажатием Ctrl + F2. Переходы от маркера к маркеру вниз по программе производится по нажатию F2. Переходы вверх по программе производятся по нажатию Shift + F2.

Снова ассемблируем программу (F7), начнем отладку (меню Debug/Start Debugging или соответствующая кнопка на панели инструментов), нажимаем F2 для быстрого перехода к строке, помеченной маркером; курсор автоматически установится в этой строке. Нажмем Ctrl + F10 (Run to cursor).

Далее все, как в предыдущем сеансе отладки: сбрасываем PIND3; дважды нажмем F11: указатель отладчика установился на первую команду подпрограммы *OpenLock:*.

Следующее нажатие F11: установился PORTB2; далее нажать F10: замок открыт (PINB2 установлен), выполнена первая задержка на 500 мс; еще трижды нажать F10: выполнены еще три задержки на 500 мс.

Четыре задержки по 500 мс составили общую задержку в 2 с, в течение которой замок был открыт. Так как мы сразу же возвращались из подпрограммы по команде *ret*, введенной для ускорения отладки, в окне Processor время выполнения задержки составляет всего несколько микросекунд вместо 2 с.

Нажать F11: PORTB2 сбросился; еще раз нажать F11: замок закрылся (PINB2 сброшен), в переменную *tmp* загрузилось число циклов (10) обращения к задержке в 500 мс.

Поскольку работу цикла мы рассматривали, установим курсор на команде *ret*, завершающей подпрограмму отпирания замка *OpenLock:* и нажмем Ctrl + F10, затем нажмем F11. Обратите внимание, что в регистре флагов (окно Workspace/IO AT90S2313/CPU/SREG) по команде *set* (она расположена перед командой *ret*) установился флаг Т; указатель стека снова указывает на последнюю ячейку памяти данных, так как мы вышли из подпрограммы самого верхнего уровня, и стек пуст.

Итак, мы проверили замыкание кнопки кодового устройства, открыли на 2 с замок, подождали 5 с и установили флаг Т — признак того, что дверь может находиться в открытом состоянии.

Теперь мы снова в основной программе. Прежде чем продолжить отладку, отпустите кнопку кодового устройства (установите PIND3 в окне Workspace/IO/PortD)! Иначе мы повторим отпирание замка.

Выполним основную программу в пошаговом режиме до команды *rjmp main*.

Команда *clr* очистит флаг Т. Это значит, что теперь открывание двери будет соответствовать режиму взлома. Для проверки работы программы в этом режиме сбросим разряд PIND2, что соответствует замыканию датчика отпирания двери. Теперь войдем в подпрограмму *DoorIsOpen:*, нажав F11 нужное для этого число раз.

Первая команда проверяет состояние флага Т, так как он сброшен, выполняется вторая и следующие за ней команды.

Далее используйте уже полюбившиеся вам горячие клавиши для выполнения пошаговой отладки с заходом в подпрограммы (F11), пошаговой отладки без захода в подпрограмму (F10), перехода к команде, на которой установлен курсор (Ctrl + F10). Сюда можно добавить еще и Shift + F11: используйте это сочетание клавиш, если вы по инерции зашли в подпрограмму, нажав F11, но не хотите проходить подпрограмму в пошаговом режиме. Проверьте работу Shift + F11 после захода в подпрограмму *d05ms:*.

Посмотрим, что происходит в этой подпрограмме.

Команда *sbi PORTB,hl* устанавливает ВЫСОКИЙ уровень на контакте PB0 (сначала устанавливается PORTB0, в следующем такте микроконтроллера устанавливается PINB0), вызывая отпирание транзистора VT3 (смотрите схему сигнализации) и включение светодиода VD3 на удаленном пульте.

В пару регистров XH:XL загружается число повторений цикла, начинающегося меткой *NewPls: NewPls: wdr ... brne NewPls.*

В каждом из пятисот циклов командой *sbi PORTB,alarm* устанавливается ВЫСОКИЙ уровень на контакте PB1 микроконтроллера, удерживающийся в течение 0.5 мс при выполнении подпрограммы *d05ms*, затем командой *cbi PORTB,alarm* устанавливается НИЗКИЙ уровень на этом контакте и также удерживается 0.5 мс при выполнении подпрограммы *d05ms*. В итоге на контакте PB1 микроконтроллера генерируется импульсная последовательность с длительностью импульса и длительностью паузы по 0.5 мс. Следовательно, период повторения импульсов составляет 1 мс, что соответствует частоте повторения 1 кГц. Таких импульсов насчитывается 500, то есть длительность импульсной последовательности — 500 мс.

По окончании цикла вызывается подпрограмма *d500ms:*, вызывающая паузу в течение 500 мс, во время которой на контакте микроконтроллера PB1 продолжает удерживаться НИЗКИЙ уровень.

После паузы происходит безусловный переход в начало подпрограммы и все повторяется, пока не будет произведен сброс.

Описанная импульсная последовательность управляет транзистором VT2, к которому подключена сирена. Когда на сирену в течение 500 мс поступают импульсы с частотой следования 1 кГц, она включается (издает звук), во время паузы в течение 500 мс она выключается. Так продолжается до тех пор, пока не произойдет сброс микроконтроллера либо при выключении питания с его последующим включением, либо при нажатии кнопки сброса SA4.

Имитируем сброс микроконтроллера, нажав Shift + F5 (Reset).

Нажмем F2, затем Ctrl + F10 для быстрого перехода к циклу проверки состояния входных контактов микроконтроллера, начинающемуся меткой *main:*. Сбросим PIND4, что соответствует замыканию кнопки SA3 (открывание замка изнутри помещения).

По ходу отладки снова попадаем в подпрограмму *OpenLock:*, которая уже вызывалась при замыкании кнопки кодового устройства. Следовательно, как правильный набор кода, так и нажатие кнопки открывания замка внутри помещения вызывают одинаковые действия программы. В схеме можно было соединить кнопки SA3 и SA2 с одним контактом микроконтроллера. Однако при наличии свободных контактов микроконтроллера делать этого не стоит, чтобы сохранить возможность изменять программу без изменения схемы для использования в других задачах.

Теперь будем считать, что при открывании замка дверь также открыли, для этого сбросим PIND2; одновременно установим PIND4, так как условие отпирания замка (сброшенное состояние PIND4) уже проверено.

Нажимаем Shift + F11 для выхода из подпрограммы открывания замка на 2 с, последующим ожиданием в течение 5 с и установкой флага Т, состояние которого свидетельствует о том, что дверь может оставаться открытой.

При продолжении отладки происходит проверка состояния контакта микроконтроллера PD4 (проверяется соответствующее ему состояние PIND2). Поскольку PIND2 сброшен, выполнится подпрограмма *DoorIsOpen:*.

Первая команда подпрограммы проверяет состояние флага Т, а так как на этот раз он установлен, происходит переход к метке *NoClose:*. В этой части подпрограммы, состоящей всего из двух команд (*NoClose: sbis PIND,door* и *rjmp NoClose*), возврат на метку *NoClose:* будет происходить до тех пор, пока мы не установим разряд PIND2, что соответствует закрытию двери.

После установки этого разряда произойдет возврат в основную программу, в которой флаг Т будет сброшен. Последующее открывание двери без замыкания кнопки кодового устройства или кнопки отпирания замка изнутри будет воспринято программой как взлом.

Мы отладили программу во всех возможных режимах работы. Посмотрим, все ли учтено?

Все должно работать прилично, если использовать идеальные кнопки, на самом деле при замыкании и размыкании контактов возникает эффект дребезга, заключающийся в том, что, как при замыкании, так и при размыкании, контакт многократно замыкается-размыкается в течение нескольких десятков миллисекунд, пока не примет требуемого положения.

Опасен ли дребезг в нашей схеме?

Для кнопок кодового устройства и открывания изнутри дребезг не представляет опасности, так как время дребезга значительно меньше времени выполнения подпрограммы открывания замка, вызываемой при замыкании этих кнопок.

Для кнопки-датчика открывания двери дребезг опасен: вспомните выход из последнего рассмотренного цикла, начинающегося меткой *NoClose:*. Дверь закрылась, после выхода из подпрограммы *DoorIsOpen:* флаг Т был сброшен, и все это произошло за несколько микросекунд. В это же время происходит дребезг контактов кнопки-датчика двери, следовательно, состояние контакта микроконтроллера PD2 изменяется; теперь кратковременное присутствие НИЗКОГО уровня на контакте означает взлом двери с включением сирены и светодиода на пульте.

Значит, при закрытии двери может включиться сирена и светодиод.

Что можно предпринять? Проще всего добавить вызов подпрограммы задержки на 500 мс, например, перед командой *clt*, очищающей флаг Т в основной программе. Тогда при закрытии двери происходит выход из подпрограммы *DoorIsOpen:*, а за время выполнения вставленной подпрограммы задержки дребезг датчика-контакта двери завершится.

Еще одна неприятность — наводки на провода, идущие от кнопок. Они могут возникнуть, например, при проведении сварочных работ вблизи нашего устройства.

Для устранения наводок можно ввести в схему фильтрующие конденсаторы, установив их вблизи контактов микроконтроллера параллельно кнопкам, а также снизить сопротивление резисторов R1, R2 и R3.

Другим вариантом является программная проверка состояния контактов с некоторыми интервалами времени. В нашей программе можно после обнаружения изменения состояния контактов проверить, что состояние остается неизменным, например, через 500 мс.

В целом устройство является работоспособным, а основная задача данной главы — знакомство со средой разработки AVR Studio, а также техникой разработки и отладки.

Перед окончательным ассемблированием восстановим программу, удалив команду *ret*, введенную нами в самое начало подпрограммы *d500ms:*.

Ассемблируйте программу, нажав F7. Теперь в файле c:\avr\Alarm\Alarm.hex находится исполняемый код нашей программы. Именно с этим файлом будет работать компьютерная программа, обслуживающая ваш программатор.

Теперь вы можете подключить программатор к разъему XP1 платы контроллера сигнализации, подсоединить программатор к соответствующему разъему компьютера, подать питание на плату контроллера, запустить компьютерную программу, обслуживающую ваш программатор, выполнить стирание Flash-памяти микроконтроллера, а затем записать туда исполняемый код нашей программы.

Надеюсь, что программирование прошло удачно.

2.2.2. Программа для контроллера сигнализации с использованием прерываний

Ввод программы

Создадим новый проект Alm_Int в новой папке c:\avr\Alm_Int, откроем новый файл программы Alm_Int.asm. О том, как это делается, смотрите в подразделе «Создание проекта».

Файл программы пуст, облегчим себе задачу, скопировав содержимое файла первого варианта программы Alarm.asm, для этого откроем меню File/Open File, в появившемся окне «Открыть» разыщем и откроем файл c:\avr\Alarm\Alarm.asm. Появившееся на экране окно с нашей прежней программой скрывает под собой окно нащей новой программы. Верхнее окно можно переместить в сторону.

Установите курсор в окне со старой программой, откройте контекстное меню для этого окна, шелкнув правой клавишей мышки, выберите строку Select all, скопируйте выделенный текст программы.

Для копирования можно выбрать один из вариантов:

- установить указатель мышки на выделенном фрагменте, нажать правую клавишу мышки и в открывшемся окне выбрать строку Copy;
- воспользоваться клавиатурой, нажав Ctrl + C;
- вызвать меню Edit/Copy;
- установить указатель мышки на выделенном фрагменте, нажав и удерживая клавишу Ctrl и левую клавишу мышки, переместить мышкой выделенный фрагмент в окно новой программы.

При наборе программ попробуйте все способы работы с фрагментами текста, знакомые вам по опыту общения с Windows 9x и Microsoft Office.

Установите указатель мышки в окно новой программы Alm_Int.asm и вставьте скопированный текст (снова есть множество вариантов, я пользуюсь правой кнопкой мыши, в открывшемся контекстном меню выбираю Past/Вставить).

Окно старой программы Alarm.asm закрываем, оно больше не понадобится.

Если при загрузке исполняемого кода программы в микроконтроллер вы не можете обнаружить файл hex, измените формат файла в меню Project/AVR Assembler Setup/Additional Output file format на формат Intel Intellec 8/MDS (Intel Hex) и снова ассемблируйте программу.

Ниже приведен листинг измененной программы. Сравните ее с предыдущей программой и внесите соответствующие коррективы.

Листинг программы

```
;Программа контроллера сигнализации с прерываниями
;==============================================
.include «c:\avr\def\2313def.inc»
.def   tmp = r16
.def   OpnEnbl = r17
.equ   door = PD2
.equ   code = PD3
.equ   open = PD4
.equ   lock = PB2
.equ   alarm = PB1
.equ   hl = PB0
.equ   t05mc = 249                 ;B TCCR0<-2 (Fck/8)
.equ   t500mc = 7812               ;B TCCR1B<-4 (Fck/256)
.equ   t7c = 27344                 ;B TCCR1B<-5 (Fck/1024)
.equ   t5c = 19531                 ;B TCCR1B<-5 (Fck/1024)
.cseg
.org 0
       rjmp   RESET                ;Reset Handler
       nop    ;rjmp EXT_INT0       ;IRQ0 Handler
       nop    ;rjmp EXT_INT1       ;IRQ1 Handler
       nop    ;rjmp TIM_CAPT1      ;Timer1 Capture Handler
       rjmp   TIM_COMP1            ;Timer1 Compare Handler
       rjmp   TIM_OVF1             ;Timer1 Overflow Handler
       rjmp   TIM_OVF0             ;Timer0 Overflow Handler
       nop    ;rjmp UART_RXC       ;UART RX Complete Handler
       nop    ;rjmp UART_DRE       ;UDR Empty Handler
       nop    ;rjmp UART_TXC       ;UART TX Complete Handler
       nop    ;rjmp ANA_COMP       ;Analog Comparator Handler
TIM_COMP1:
       cbi    PORTB,lock           ;Установка НИЗКОГО уровня
                                   ;на линии lock (PB2)
       reti
TIM_OVF1:
       clr    OpnEnbl              ;Очистка переменной
       cbi    PORTB,alarm          ;Установка НИЗКОГО уровня
                                   ;на линии alarm (PB1)
       ldi    tmp,high($10000 - t500mc);Загрузка
                                   ;в счетчик константы,
                                   ;соответствующей 500 мс
       out    TCNT1H,tmp
       ldi    tmp,low($10000 - t500mc)
       out    TCNT1L,tmp
       in     tmp,TIMSK
       sbrc   tmp,TOIE0
       rjmp   cTOIE0
       sbrs   tmp,TOIE0
       rjmp   sTOIE0
cTOIE0:
       cbr    tmp,1<<TOIE0         ;Запрет прерывания таймера T0
```

```
        rjmp   go_ret
sTOIE0:
        sbr    tmp,1<<TOIE0        ;Разрешение прерывания таймера T0
go_ret:
        out    TIMSK,tmp
        reti
TIM_OVF0:
        ldi    tmp,$100 - t05mc    ;Загрузка в счетчик
                                   ;константы,
                                   ;соответствующей 0.5 мс
        out    TCNT0,tmp
        sbis   PORTB,alarm
        rjmp   sAlarm
        sbic   PORTB,alarm
        rjmp   cAlarm
sAlarm:
        sbi    PORTB,alarm         ;Установка разряда alarm
        rjmp   next
cAlarm:
        cbi    PORTB,alarm         ;Сброс разряда alarm
next:   clr    tmp
        reti
RESET:  ldi    tmp,(1<<TOIE1) + (1<<OCIE1A) + (1<<TOIE0)
        out    TIMSK,tmp           ;Разрешение прерываний таймеров
        sei                        ;Запрет всех прерываний
        ldi    tmp,low(RAMEND)     ;Определение вершины стека
        out    SPL,tmp
        clr    tmp                 ;Очистить tmp
        out    DDRD,tmp            ;Линии порта D — входы
        out    PORTB,tmp           ;Порт B — НИЗКИЕ уровни
        ser    tmp                 ;Установить tmp
        out    PORTD,tmp           ;Порт D — ВЫСОКИЕ уровни
        out    DDRB,tmp            ;Линии порта B — выходы
        out    PIND,tmp            ;Операция только для отладчика
        ldi    tmp,15              ;Инициализация сторожевого
        out    WDTCR,tmp           ;таймера: без wdr сброс
                                   ;через 2 с
        clt                        ;Очистка T-флага регистра флагов
;ОСНОВНОЙ БЛОК ПРОГРАММЫ
;======================
main:   wdr                        ;Сброс сторожевого таймера
        sbis   PIND,code           ;Пропуск следующей команды,
                                   ;если на линии code (PD3)
                                   ;ВЫСОКИЙ уровень
        rcall  OpenLock            ;Вызов подпрограммы OpenLock
        sbis   PIND,open           ;Пропуск следующей команды,
                                   ;если на линии open (PD4)
                                   ;ВЫСОКИЙ уровень
        rcall  OpenLock            ;Вызов подпрограммы OpenLock
        sbis   PIND,door           ;Пропуск следующей команды,
                                   ;если на линии door (PD2)
```



```
                                   ;ВЫСОКИЙ уровень
        rcall  DoorIsOpen          ;Вызов подпрограммы
                                   ;DoorIsOpen
        clt                        ;Очистка T-флага регистра флагов
        rjmp   main                ;Возврат к команде
                                   ;с меткой main:
;КОНЕЦ ОСНОВНОГО БЛОКА ПРОГРАММЫ
;======================
;Подпрограмма управления замком
;=============================
OpenLock:
        sbi    PORTB,lock          ;Установка ВЫСОКОГО уровня
                                   ;на линии lock порта B
        ldi    tmp,high($10000 - t7c);Загрузка в счетчик
        out    TCNT1H,tmp          ;таймера константы для
        ldi    tmp,low($10000 - t7c);отсчета 7 с
        out    TCNT1L,tmp
        ldi    tmp,5               ;Запуск таймера T1 с
        out    TCCR1B,tmp          ;предварительным делением
                                   ;тактовой частоты на 1024
        ldi    tmp,high($10000 - t5c)
        out    OCR1AH,tmp          ;Ввод константы сравнения
        ldi    tmp,low($10000 - t5c);для прерывания
        out    OCR1AL,tmp          ;за 5 с до переполнения
        ldi    OpnEnbl,$ff         ;Загрузка в tmp константы
more:   wdr
        tst    OpnEnbl
        brne   more                ;Переходить на метку more:
                                   ;до возникновения прерывания
        clr    tmp
        out    TCCR1B,tmp          ;Остановить таймер T1
        set                        ;Установить флаг T в
                                   ;регистре флагов
        ret                        ;Возврат из подпрограммы
;Подпрограмма управления сиреной и светодиодом
;==========================================
DoorIsOpen:
        brts   NoClose             ;Если в регистре флагов флаг T
                                   ;установлен — перейти на
                                   ;метку NoClose:
        sbi    PORTB,hl            ;Установить ВЫСОКИЙ уровень
                                   ;на линии hl (PB0)
        in     tmp,TIMSK
        cbr    tmp,1<<OCIE1A       ;Запрет прерывания
        out    TIMSK,tmp           ;при совпадении содержимого
                                   ;счетчика таймера T1 и константы
        ldi    tmp,high($10000 - t500mc)
        out    TCNT1H,tmp          ;Загрузка константы для
                                   ;отсчета 500 мс в таймере T1
        ldi    tmp,low($10000 - t500mc)
        out    TCNT1L,tmp
```

```
        ldi     tmp,4                   ;Запуск таймера T1 с делением
        out     TCCR1B,tmp              ;тактовой частоты на 256
        ldi     tmp,$100 - t05mc        ;Загрузка константы для
        out     TCNT0,tmp               ;отсчета 0.5 мс таймером T0
        ldi     tmp,2                   ;Запуск таймера T0 с делением
        out     TCCR0,tmp               ;тактовой частоты на 8
        sbi     PORTB,alarm             ;Установка ВЫСОКОГО уровня
                                        ;на линии alarm порта B
NonStop:
        wdr
        rjmp    NonStop
NoClose:
        wdr
        sbis    PIND,door               ;Пропустить следующую команду,
                                        ;если на линии door
                                        ;(PD2) ВЫСОКИЙ уровень
        rjmp    NoClose                 ;Переход на метку NoClose
        ldi     tmp,$100 - 255
        out     TCNT0,tmp
        clr     tmp
        out     TIFR,tmp
        ldi     tmp,5
        out     TCCR0,tmp
        in      tmp,TIMSK
        sbr     tmp,1<<TOIE0            ;Разрешение прерывания таймера T0
        out     TIMSK,tmp
wait:   wdr
        tst     tmp
        brne    wait
        ret                             ;Возврат из подпрограммы
```

Что изменилось в программе?

Прежде всего, в блоке ссылок на обработчики прерываний появляются ссылки на обработчики прерываний переполнения таймера T0, переполнения таймера T1, а также прерывания сравнения таймера T1.

Подпрограммы задержек с подсчетом числа тактов микроконтроллера убираются из программы; а для реализации необходимых интервалов времени используются прерывания таймеров.

Отличие алгоритма программы для контроллера сигнализации с использованием таймеров

Внешние проявления работы контроллера не должны измениться. Мы сохраним прежние режимы и логику работы.

Нам необходимо отпереть замок (ВЫСОКИЙ уровень на линии lock), через две секунды запереть его (НИЗКИЙ уровень на линии lock). Через 5 с проверить состояние кнопки-датчика двери.

Воспользуемся таймером для организации интервалов времени в 2 с и в 5 с. Определим, какой максимальный интервал времени можно реализовать на более простом в программировании 8-разрядном таймере T0.

Как на таймер T1, так и на таймер T0 счетные импульсы поступают от предварительного делителя частоты, подключенного непосредственно к тактовому генератору микроконтроллера.

Для включения/выключения таймера T0 в три младших разряда регистра управления TCCR0, а для таймера T1 — в TCCR1B записываются значения от нуля до пяти.

В **Табл. 2** показано соответствие между числом N, записываемым в младшие разряды регистров управления таймеров, и частотой счетных импульсов $F_{счетн}$, поступающих на таймеры. $F_{тг}$ — частота тактового генератора микроконтроллера.

Таблица 2. **Частота счетных импульсов, соответствующая числу N**

N	0	1	2	3	4	5
$F_{счетн.}$	останов	$F_{тг}$	$F_{тг}/8$	$F_{тг}/64$	$F_{тг}/256$	$F_{тг}/1024$

Восьмиразрядный таймер T0 может отсчитать до 2^8 импульсов без переполнения, то есть он считает от нуля до 255, следующее значение — снова ноль и так далее. При переходах от состояния 255 к состоянию 0 в счетчике таймера наступает переполнение, что фиксируется микроконтроллером.

При выборе максимального коэффициента деления (1024) и частоте тактового генератора 4 МГц максимальное время, в течение которого таймер T0 может считать импульсы без переполнения, составит:

$1024 \cdot 2^8/4000000 = 0.065536$ с.

Значит, таймер T0 можно использовать только для организации интервалов времени в 0.5 мс, необходимых для работы сирены.

Определим максимальный интервал времени, который может обеспечить 16-разрядный таймер T1: $1024 \times 2^{16}/4000000 = 16.777216$ с.

Следовательно, в нашей программе можно организовать 7-секундный интервал времени, используя таймер T1.

При обнаружении замыкания кнопки отпирания замка изнутри или верного кода (кнопка кодового устройства замкнута) установим ВЫСОКИЙ уровень на линии управления замком для его отпирания, запустим таймер T1. В регистры сравнения таймера T1 занесем константу, которая будет соответствовать состоянию счетчика таймера через 2 с счета. Это через 2 с вызовет прерывание сравнения. В обработчике этого прерывания установим НИЗКИЙ уровень на линии управления замком для его запирания. Еще через 5 с окончится 7-секундный интервал времени, а да-

лее программа выполнит те же действия, которые предусматривались в предыдущей версии программы.

Для обеспечения режима взлома используем оба таймера. По прерыванию таймера T0 с периодичностью в 0.5 мс будет поочередно устанавливаться ВЫСОКИЙ и НИЗКИЙ уровни на линии управления сиреной — таймер T1 с интервалом в 500 мс будет поочередно разрешать и запрещать прерывания таймера T0. Таким образом, 500 мс на сирену будут подаваться импульсы частотой следования 1 кГц, в течение следующих 500 мс сирена будет выключена, далее процесс повторяется. В обработчике прерывания переполнения таймера T1 необходимо вставить команду установки НИЗКОГО уровня на линии управления сиреной, поскольку неизвестно, в каком состоянии останется эта линия при запрещении прерывания переполнения таймера T0. Указанная команда предотвратит подачу на сирену постоянного напряжения во время паузы.

Задержка после закрытия двери для устранения влияния дребезга контактов на работу сигнализации организуется на таймере T0.

Отладка программы

Для контроля отладки нам понадобятся окна Workspace и Memory.

Для удобства просмотра в окне Workspace можно убрать все, что нам не потребуется. Для этого надо развернуть интересующие вас строки, например, CPU, PORTB, PORTD, открыть контекстное меню (щелкнуть правой клавишей мышки в окне Workspace) и выбрать строку Hide non-expanded.

Конфигурация окон, представленных на **Рис. 16**, выбрана так, что все необходимое для просмотра доступно без использования прокруток и активизации то одного, то другого окна просмотра.

В окне Workspace шестнадцатеричное представление информации дополнено представлением в виде отдельных окошек для каждого разряда информации для регистров Timer/Counter 1 High, Timer/Counter 1 Low, Compare A High и Compare A Low. Для изменения представления данных следует воспользоваться контекстным меню окна, установив мышку на строке, в которой необходимо изменить представление информации, нажать правую клавишу мышки, а в открывшемся контекстном меню выбрать строку Show Bitnumber.

Выполняем три шага отладки (трижды нажимаем F11): какую константу загружают команды

ldi tmp,(1<<TOIE1) + (1<<OCIE1A) + (1<<TOIE0) и *out TIMSK,tmp*

в регистр TIMSK?

Значения констант TOIE1, OCIE1A и TOIE0 соответствуют 7-му, 6-му и 1-му разрядам регистра TIMSK (см. файл 2313def.inc).

Запись вида 1<<TOIE1 можно объяснить следующим образом: занесите единицу в младший разряд байта (получим двоичное число 00000001) и

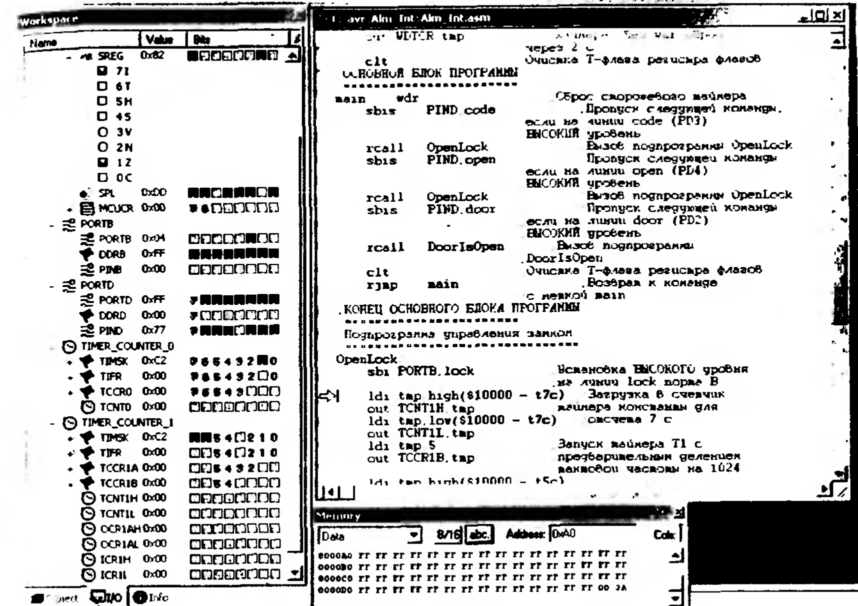


Рис. 16. Оптимизация конфигурации окон просмотра

сдвиньте эту единицу влево TOIE1 раз (TOIE1 = 7), то есть у нас получится двоичное число 10000000 (корректная запись двоичного числа 0b10000000). Тогда наша команда соответствует загрузке в регистр числа 0b10000000 + 0b01000000 + 0b00000010,

равного 0b11000010, или 0xC2 в шестнадцатеричном представлении, в чем вы можете убедиться, вызвав переменную tmp в окне Watches.

Эту запись можно интерпретировать и по-другому: в регистре TIMSK таким способом устанавливаются разряды TOIE1, OCIE1A и TOIE0, что соответствует разрешению прерывания при переполнении таймера T1, прерывания сравнения таймера T1 и прерывания при переполнении таймера T0.

Следующая команда *sei* разрешает все прерывания, при этом в регистре флагов устанавливается флаг I, далее основная программа до команды *rjmp main* нам знакома.

Выполните команды программы до метки *main:*.

Запись последовательности состояний порта

Симулятор AVR Studio предоставляет удобную возможность записи всех изменений состояния порта PORTx в файл, для нашего микроконтроллера — это PORTB или PORTD, а также использования похожего файла для автоматического изменения состояний порта PINx при отладке.

Откройте меню Debug/AVR Simulator Options/Stimuli and Logging, выберите PORTD, выберите имя файла, затем нажмите кнопку Add Entry, результат представлен на **Рис. 17**.

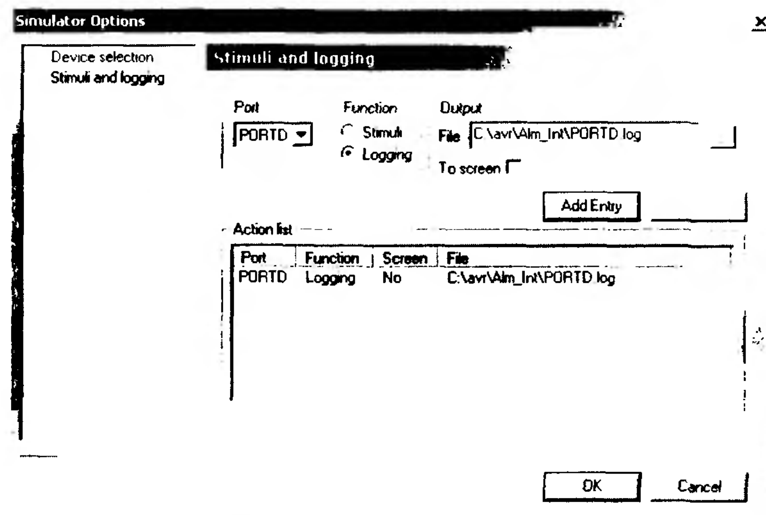


Рис. 17. Создание файла состояний портов

Нажмите кнопку OK, затем в окне Workspace сбросьте PORTD3 (именно изменение PORTD будет записано в файл), запомните содержимое Cycle Counter в окне Processor. Выполните несколько шагов отладки, а затем установите PORTD3. Теперь нажмите F7 — файл PortD.log создан и доступен для просмотра и изменения.

Найдите и откройте файл c:\avr\Alm_Int\PortD.log, пользуясь средствами Windows, например проводником. Файл будет содержать примерно такие строки: 000000017: F7, 000000042: FF.

Это значит, что на 17-м такте состояние порта изменилось на 0xF7 (третий разряд сброшен), на 42-м такте состояние изменилось на 0xFF (третий разряд снова установлен). Информации о том, с каким портом происходили изменения, в файле нет, поэтому лучше присваивать файлу значащее имя, как в этом случае.

Для чего нам такой файл? Можно, конечно, проверить, на каком такте происходили изменения состояния порта, но не это важно.

Средствами Windows скопируйте этот файл, измените его имя на PortD.sti, затем откройте этот файл и, добавив в конце строку 999999999: FF, сохраните файл. Без указанной строки использование этого файла в AVR Studio вызовет сообщение об ошибке.

> Для отключения записи состояния порта в файл надо снова вызвать окно Stimuli and Logging, выбрать то же имя файла, щелкнуть в окне Action List по имени порта (PORTD), при этом активизируется кнопка Delete Entry – щелкните по ней для прекращения записи.

Симулирование состояний порта

Попробуем использовать полученный файл sti для имитации изменения состояний входных контактов микроконтроллера.

Начнем программу сначала, сбросив ее нажатием Shft + F5. Выполним программу до метки *main:*, затем вызовем меню Debug/AVR Simulator Options/Stimuli and Logging, выберем PORTD, функцию Stimuli, в окно ввода введем имя нашего файла c:\avr\Alm_Int\PortD.sti или разыщем его в каталоге (кнопка с тремя точками справа от окна ввода). Теперь выполните несколько шагов, наблюдая за изменением PIND (именно PIND, а не PORTD!) и одновременно следя за Cycle Counter.

> Если часть программы, в которой надо изменять состояние входных контактов микроконтроллера, уже отлажена, можно воспользоваться указанным выше способом создания файла.sti, и, вызывая этот файл, при следующих сеансах отладки, проскакивать часть программы, не заботясь об изменении состояний порта вручную.
> Файл *.sti можно создать вручную, пользуясь любым текстовым редактором. Можно создать и набрать новый файл *.sti в окне Project нашего проекта точно так же, как создавался и редактировался файл *.asm. Важно завершать файл строкой с очень большим адресом и данными FF, например, 999999999:FF.

Установим курсор на первой команде подпрограммы *OpenLock:*, нажмем F9: появившаяся коричневая метка в правом поле против команды, на которой установлен курсор, — точка останова программы (BreakPoint). Если теперь запустить выполнение программы (меню Debug/Run F5), то программа будет выполняться без нашего вмешательства, пока не встретит точку останова.

Но при отладке в подпрограмму *OpenLock:* можно попасть из основного блока программы только когда разряд PIND3 порта PIND сброшен.

При запуске выполнения программы (F5) мы не сможем вручную сбросить разряд PIND3, при этом будет бесконечно повторяться выполнение основного блока программы.

Поможет решить эту задачу симулирование состояний порта с помощью нашего файла PortD.sti. Сбросьте программу и повторите вызов указанного файла.

> После повторного ассемблирования программы (F7) файл *.sti необходимо загружать снова.

Теперь нажмите F5, программа выполнилась без нашего вмешательства до точки останова в подпрограмме *OpenLock:*. Первая команда подпрограммы *sbi PORTB,lock* вызывает открывание замка.

Как и в предыдущей программе, в подпрограмме *OpenLock:* надо организовать 2-секундный интервал, после которого надо закрыть замок, а затем организовать 5-секундный интервал. Запустим таймер T1 для отсчета 7 с, после чего должно произойти прерывание переполнения таймера T1, а за 5 с до переполнения таймера должно произойти прерывание сравнения, обработчик которого закроет замок.

Определим величину константы N, которую надо загрузить в 16-разрядный счетчик таймера T1 для получения интервала времени в 7 с при коэффициенте предварительного деления 1024 и тактовой частоте 4 МГц: $N = 7 \times 4000000/1024 = 27343.75$

В нашей программе эта величина представлена константой *t7s*. С поступлением каждого импульса на счетчик, его содержимое увеличивается на единицу. Переполнение счетчика, а соответственно, и прерывание при переполнении наступят при изменении состояния 16-разрядного счетчика с 0xFFFF на 0x0000 (с 0b1111111111111111 на 0b0000000000000000). Если бы у счетчика был 17-й разряд, то счетчик перешел бы в состояние 0x10000. Поэтому для того, чтобы 16-разрядный счетчик до наступления переполнения отсчитал 27344 импульса, в него надо загрузить число 0x10000 – 27344.

Пусть вас не смущает представление чисел в разных форматах, такая возможность оказывается удобной при работе.

> В техническом описании микроконтроллеров подчеркивается, что для корректной работы 16-разрядного таймера следует сначала загружать старший байт, а затем младший.

Загрузив старший TCNT1H и младший TCNT1L байты счетчика TCNT1, запускаем его с коэффициентом деления 1024, для чего в регистр TCCR1B записываем значение, равное 5 (см. **Табл. 2**), сразу после этого счетчик начинает работать.

> **Запомните!** При записи данных в пары регистров таймера T1 и АЦП командами *out* первым загружается старший регистр, затем младший. Считывание данных выполняется в обратном порядке — сначала из младшего регистра затем из старшего.
> Это касается пар регистров TCNT1H:TCNT1L, OCR1AH:OCR1AL, OCR1BH:OCR1BL, ICR1H:ICR1L (регистры таймера T1) и ADCH:ADCL (регистры АЦП), часть из них может быть только считана (подробности смотрите в технических описаниях микроконтроллеров).

Следующий блок — загрузка 16-разрядного регистра сравнения OCR1A. При работе счетчика TCNT1 его содержимое будет сравниваться с содержимым OCR1A, когда состояние счетчика превысит состояние регистра сравнения, произойдет прерывание сравнения.

Счетчик таймера продолжает работать до тех пор, пока не будут обнулены три младших разряда регистра TCCR1B. Поэтому как прерывание переполнения, так и прерывание сравнения без остановки счетчика будут повторяться. Однако, если не записать новое значение в счетчик TCCR1B, следующее прерывание произойдет после подсчета 0xFFFF импульсов, так как теперь счет идет уже от нуля, а не от 23344. Прерывание сравнения наступит за пять секунд до прерывания переполнения, так как замок надо запереть за 5 с до истечения 7-секундного интервала времени.

Определим константу, которую необходимо загрузить в регистр сравнения OCR1A для того, чтобы прерывание сравнения произошло за 5 с до переполнения счетчика TCNT1:

$$M = 5 \cdot 4000000/1024 = 19531.25.$$

Полученную константу надо загрузить в регистр сравнения. Последовательность загрузки сначала старшего OCR1AH, затем младшего OCR1AL байта 16-разрядного регистра сравнения нарушать не следует.

Теперь мы должны ждать, пока не произойдут оба прерывания. Для этого организован цикл проверки переменной *OpnEnbl*, начинающийся меткой *more:*. Сброс этой переменной производится в обработчике прерывания переполнения, соответственно выход из цикла произойдет по истечении 7 с. Обязательной именно в этом цикле является команда сброса сторожевого таймера *wdr*, ведь максимально возможное время, через которое счетчик сторожевого таймера переполнится, вызывая сброс программы микроконтроллера, составляет лишь около 2 с при напряжении питания 5 В.

По выходе из цикла счетчик таймера T1 останавливается посылкой нуля в регистр TCCR1B, а команд *set* устанавливает флаг T в регистре флагов.

Как работает аппаратное прерывание программы?

Команды в программе выполняются друг за другом в порядке их записи, а при выполнении команд перехода или при вызове подпрограмм вместо выполнения следующей по порядку команды происходит переход к другой группе команд. Но куда именно и на каком этапе программы происходит переход, можно определить по листингу программы.

Аппаратное прерывание микроконтроллера может произойти при выполнении любой команды. Если возникшее прерывание программно разрешено, должна быть выполнена подпрограмма-обработчик именно этого прерывания, после чего выполнение программы должно быть продолжено.

Но как определяется адрес подпрограммы обработчика прерывания? Для каждого аппаратного прерывания в памяти программ микроконтроллера зарегистрирована ячейка, в которой надо поместить команду перехода на обработчик. Например, для обработчика прерывания переполнения таймера T0 микроконтроллера AT90S2313 это 6-я ячейка (их нумерация начинается с нуля), именно в этой ячейке (6-я строка программы, строки также считаем с нулевой) находится команда *rjmp TIM0_OVF*.

Пусть прерывание переполнения таймера T0 возникло при выполнении команды, находящейся, например, в строке № 40. Выполнение команды доводится до конца. Прерывание же инициализирует следующие действия:

- в стек из счетчика команд записывается адрес ячейки памяти, содержащей следующую команду, в программе она находится в строке № 41 — именно она должна быть выполнена после завершения подпрограммы обработки прерывания;
- в счетчик команд заносится адрес ячейки № 6 памяти программ, связанной с прерыванием переполнения таймера T0 и содержащей команду *rjmp TIM0_OVF*.

Выполнение этой команды вызывает переход на обработчик прерывания, который должен начинаться меткой *TIM0_OVF:*.

Обработчик прерывания должен завершаться командой *reti*, именно ее выполнение вызывает выталкивание из стека в счетчик команд адреса команды, которая должна выполняться по завершении обработчика, в нашем примере это команда, находящаяся в строке № 41.

> **Замечание.** Для каждого типа микроконтроллера набор прерываний и порядок размещения связанных с ними ячеек памяти программ индивидуален. Так, для микроконтроллера ATmega8515 это не 6-я, а 7-я ячейка. Группа таких ячеек — векторы прерываний микроконтроллера — размещается в начале памяти программ.

Продолжим рассмотрение программы. Наш цикл, начинающийся меткой *more:*, продолжает выполняться. Установим курсор на команде *rjmp TIM_COMP1*, находящейся в самом начале программы, и нажмем

Ctrl + F10 для перехода к этой команде. Наберитесь терпения: компьютеру с тактовой частотой процессора в 950 МГц на это требуется примерно 25 с.

Можно ускорить процесс: остановите отладку, нажав Ctrl + F5. В окне Workspace в регистрах TCNT1H и TCNT1L установите такие же значения, как в регистрах OCR1AH и OCR1AL соответственно. Если у вас информация для этих регистров выведена в виде Bitnumber, измените значения в регистрах TCCR1 установкой/сбросом флажков, если информация представлена в другом формате, например, в шестнадцатеричном, выполните двойной щелчок по величине, которую вы хотите исправить, в открывшемся окне измените ее значение.

Если снова нажать F5, программа быстро выполнится до точки останова.

Тем не менее для перехода к точке останова понадобится еще не менее 1024 тактов микроконтроллера, так как прерывание сравнения наступает после того, как состояние счетчика превысит состояние регистра хранения, а на счетчик поступает только каждый 1024-й импульс.

При работе с несколькими окнами не обязательно ждать перехода программы к точке останова, можно наблюдать за панелью Debug в окне AVR Studio. На **Рис. 18** на переднем плане находится активное окно MS Word, на заднем плане — окно AVR Studio, в котором выполняется отладка программы и ожидается переход к точке останова.

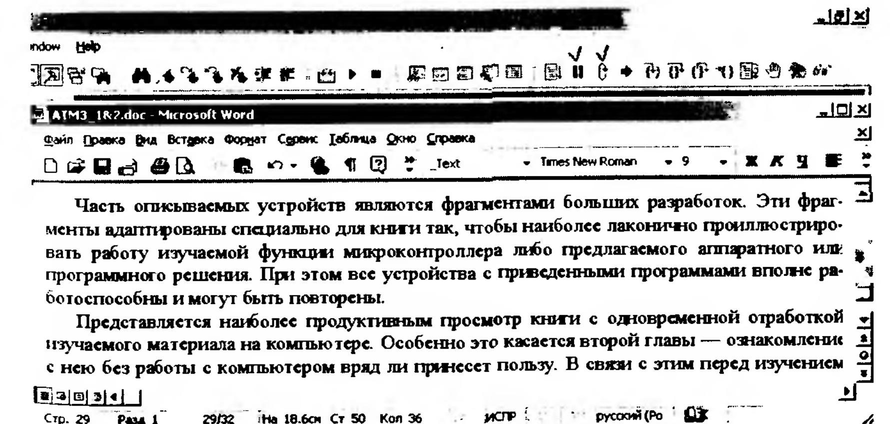


Рис. 18. Использование панели Debug для наблюдения за выполнением программы

Пока точка останова не достигнута, на панели Debug доступны кнопки Break (Ctrl + F5) и Reset (Shift + F5), пометим их галочками. Остальные кнопки окрашены в серый цвет — они недоступны. Как только программа

выполнится до точки останова, эти кнопки станут активными и их окраска изменится. Для расширения области активного окна MS Word панель Debug можно переместить в самый верх окна AVR Studio, для этого панель Debug надо потянуть мышкой за вертикальную линию левее кнопки Run (на ней изображен лист бумаги с вертикальной стрелкой справа, для определения кнопок используйте подсказки, появляющиеся при установке курсора на кнопку).

Наблюдая за работой стека в окнах Memory и Processor, нажимаем F11. Работа стека при вызове обработчика прерывания отличается от работы при вызове обычной подпрограммы тем, что записывается не адрес последней команды перед обработчиком *rjmp TIM_COMP1*, а адрес команды, которая должна была выполниться, когда возникло прерывание.

Переходим в подпрограмму обработки прерывания *TIM_COMP1:*.

Следующая команда вызывает запирание замка: *cbi PORTB,lock*.

По выходу из подпрограммы по команде *reti* мы снова попадаем на одну из команд цикла, начинающегося меткой *more:*.

Следующую точку останова можно разместить, например, в теле подпрограммы обработчика прерывания переполнения таймера T1, установив курсор на первой команде после метки *TIM_OVF1:* и нажав F9.

Нажимаем F5 и ожидаем выполнения программы до точки останова. Как и в случае с прерыванием сравнения, можно остановить программу (Ctrl + F5) и в окне Workspace установить в обоих регистрах TCCR1H и TCCR1L одинаковые значения 0xFF, предшествующие переполнению счетчика.

> Другой путь ускорения отладки при работе с таймерами — замена на время отладки большого предварительного коэффициента деления (1024) на единичный (устанавливается значение 001 в трех младших разрядах регистра TCCR1B для таймера T1 и регистра TCCR0 для таймера T0). Можно увеличить и константы, отправляемые в регистры TCNT1H и TCNT1L для таймера T1 или в TCNT0 для таймера T0, приблизив их к значениям 0xFFFF для таймера T1 и к 0xFF для таймера T0.
>
> Еще один способ — в окне IO вручную установить флажок прерывания переполнения, для таймера T0 это флажок TOV0, для таймера T1 — TOV1 (Timer Counter 1/TIFR/TOV1). Это же можно делать для имитации прерывания сравнения, установив флаг OCF1A, находящийся там же, где и флаг TOV1. Важно не переусердствовать, иначе может быть нарушена логика работы программы.

Первая команда очищает переменную *OpnEnbl: clr OpnEnbl*. Это значит, что теперь, по прошествии 7 секунд, дверь открывать нельзя, иначе это будет считаться взломом. Оставшиеся в обработчике прерывания команды относятся к режиму взлома (работа сирены) и будут рассмотрены ниже.

После возврата из обработчика прерывания переполнения таймера T1 переменная *OpnEnbl* равна нулю, происходит выход из цикла, начинающегося меткой *more:*. Следующая пара команд очисткой регистра TCCR1B останавливает счетчик таймера T1, а команда *set* устанавливает флаг T в регистре флагов.

Подпрограмма *OpenLock* выполнена, мы вернулись в основной блок программы.

Как и в предыдущей версии программы, набор верного кода и нажатие кнопки отпирания изнутри вызывают одинаковые действия контроллера, поэтому проверять работу программы при нажатии указанной кнопки не будем. Имитируем открывание двери, сбросив разряд PIND2 в окне IO.

После выполнения нескольких шагов мы попадаем в подпрограмму *DoorIsOpen:*. Так как флаг T установлен, первая команда *brts NoClose* вызывает переход к циклу, начинающемуся меткой *NoClose:*. Цикл будет продолжаться до тех пор, пока мы не установим разряд PIND2 в окне IO.

За циклом следует блок команд, инициализирующий таймер T0 таким образом, что его переполнение наступит через 500 мс, это позволяет устранить возможное влияние эффекта дребезга контактов при закрытии двери, о котором упоминалось при рассмотрении предыдущей программы.

Пока не произойдет переполнение таймера T0, выполняется цикл с меткой *wait:*. В обработчике прерывания таймера T0 переменная *tmp* очищается, после обработки прерывания происходит возврат в цикл с меткой *wait:*. Затем выход из цикла, так как проверка значения переменной *tmp* командой *tst tmp* теперь приводит к установке флага Z (переменная *tmp* равна нулю) и к выходу из цикла. Далее происходит возврат из подпрограммы *DoorIsOpen:* в основной блок программы, где команда *clt* очищает флаг T. Это значит, что при открывании двери программа перейдет в режим взлома.

Имитируем открывание двери сбросом разряда PIND2 в окне IO. Выполнив несколько шагов, попадаем в подпрограмму *DoorIsOpen:*. Поскольку флаг T теперь сброшен, команда *brts NoClose* не вызывает перехода на метку *NoClose:* и командой *sbi PORTB,hl* выполняется установка в регистре PORTB разряда hl, вызывающего включение светодиода на удаленном пульте.

Следующие три команды очищают разряд OCIE1A в регистре TIMSK, запрещая прерывание сравнения и не затрагивая разряды, отвечающие за другие прерывания таймеров.

Следующий блок из шести команд загружает регистры TCNT1H, TCNT1L и TCCR1B таким образом, что прерывание таймера T1 произойдет через 500 мс. Далее блок из пяти команд инициализирует регистры TCNT0 и TCCR0 так, чтобы прерывание таймера T0 произошло через 0.5 мс.

Расчет предварительных коэффициентов деления и констант, загру-жаемых в регистры счетчиков, аналогичен расчету, приведенному ранее.

Следуюшая команда *sbi PORTB,alarm* приводит к подаче высокого на-пряжения на сирену. Блок завершается бесконечным циклом с меткой *NonStop:*. Теперь до нажатия кнопки SA4, сбрасывающей программу, или до выключения питания будет продолжаться выполнение этого цикла, прерываемое таймером T0 один раз в 0.5 мс и таймером T1 один раз в 500 мс.

Посмотрим, как работает программа в режиме вторжения. В окне Workspace/Processor очищаем Stop Watch, устанавливаем курсор на метке *TIM_OVF0:*, которой начинается обработчик прерывания переполнения таймера T0, нажимаем Ctrl + F10. При этом время, потребовавшееся для переполнения счетчика таймера T0, составило 498.75мкс, что отражено в окне Processor/Stop Watch, флажок в разряде PB1 регистра PORTB уста-новлен, также установлен и флажок PINB1 (окно IO).

Следя за изменениями времени и состояний линий порта B, нажмем Ctrl + F10 несколько раз. При каждом нажатии приращение времени со-ставляет 500 мкс, а установка и сброс линий PORTB1 и PINB1 чередуются.

Пройдем обработчик прерывания переполнения таймера T0 в пошаго-вом режиме. Это прерывание должно повторяться каждые 0.5 мс, а при возникновении прерывания в регистре счетчика таймера T0 — нулевое значение. Поэтому в обработчике прерывания в счетчик таймера T0 нужно снова занести константу, соответствующую интервалу времени между пре-рываниями таймера T0 (0.5 мс). Уже известная нам константа загружается парой команд в регистр TCNT0. При этом остановки счетчика и измене-ния предварительного коэффициента деления не происходит, так как не было изменено состояние регистра TCCR1B.

Следующий блок команд до метки *cAlarm:* инвертирует сигнал, управ-ляющий сиреной: если в течение предыдущих 0.5 мс на сирену подавалось высокое напряжение, то в течение следующих 0.5 мс будет подаваться низ-кое напряжение, если подавалось низкое напряжение, то будет подаваться высокое напряжение.

Команду *sbis* мы рассматривали при отладке основного блока програм-мы; *sbis* — это аббревиатура наименования команды «Skip if Bit in I/O Reg-ister is Set», переводящегося как «перескочить через следующую команду, если разряд в регистре ввода/вывода установлен». Аналогично работает команда *sbic*, но следующая за ней команда игнорируется, когда разряд в регистре ввода/вывода сброшен.

Поэтому, когда разряд alarm в регистре PORTB установлен, команда *rjmp sAlarm* пропускается, выполняется команда *rjmp cAlarm* и разряд alarm в регистре PORTB сбрасывается командой *cbi*. При сброшенном разряде alarm выполняется команда *rjmp sAlarm*, а разряд alarm устанавливается ко-

мандой *sbi*. Затем следует команда *rjmp next*, препятствующая сбросу разря-да alarm следующей далее командой *cbi*.

Команда *clr tmp* на работу микроконтроллера в режиме взлома не влия-ет. Завершает обработчик команда возврата из прерывания *reti*.

Для того чтобы дождаться прерывания таймера T1, нам понадобилось бы повторить 1000 раз выполнение прерывания переполнения таймера T0.

Ускорим отладку. Установив курсор на метке *TIM_OVF1:*, которой на-чинается обработчик прерывания переполнения таймера T1, нажимаем Ctrl + F10.

О том, как сократить процесс ожидания перехода программы к этой метке, говорилось выше, но, если вы хотите проследить, какое время в действительности потребуется микроконтроллеру на выполнение (500 мс, окно Processor/Stop Watch), вам придется подождать.

Пройдите в пошаговом режиме обработчик прерывания переполнения таймера T1, начинающийся меткой *TIM_OVF1:*. Первая команда *clr OpnEnbl* рассматривалась ранее при выполнении задержки в 7 с, на ра-боту в режиме взлома не влияет.

Следующая команда *cbi PORTB,alarm* выключает сирену. Во время пау-зы на сирене (блоке) следует поддерживать низкое напряжение.

Далее без остановки счетчика таймера T1 в его регистры TCNT1H и TCNT1L загружаются константы, соответствующие интервалам времени между прерываниями таймера T1 (500 мс).

Следующая команда *in tmp,TIMSK* загружает в регистр tmp содержимое регистра маскирования прерываний TIMSK, а следующий далее блок ко-манд проверяет, установлен ли разряд TOIE0 в регистре TIMSK, то есть разрешено ли прерывание переполнения таймера T0. Если в течение пред-шествующих 500 мс прерывание было разрешено (разряд TOIE0 был уста-новлен), оно запрещается (разряд TOIE0 сбрасывается), если же прерыва-ние было запрещено, оно разрешается. Организация этого блока анало-гична организации блока инвертирования сигнала, управляющего сиреной в обработчике прерывания переполнения таймера T0, только здесь инвертируется состояние разряда, управляющего разрешением/за-претом прерывания таймера T0.

Получается, что в течение 500 мс таймер T0 генерирует прерывания с интервалом в 0.5 мс то включая, то выключая сирену, затем происходит прерывание таймера T1, выключающее сирену и запрещающее прерыва-ния таймера T0 на следующие 500 мс, следующее прерывание таймера T1 снова разрешает прерывание таймера T0 и весь описанный процесс повто-ряется.

Убедимся в том, что программа работает именно так. Установим курсор на какой-либо команде обработчика прерывания переполнения таймера T0, сбросим время в окне Processor/StopWatch, нажмем Ctrl + F10. Потре-

бовалось примерно 500 мс для того, чтобы наступило прерывание переполнения таймера T0, это значит, что сирена все это время молчала. Чтобы убедиться, что обработчик прерывания таймера T0 снова подает импульсы периодом 1 мс на сирену (0.5 мс импульс и 0.5 мс пауза), нажмем еще несколько раз Ctrl + F10, следя за изменением времени и состояния разряда PORTB1, управляющего работой сирены.

Все режимы работы программы проверены; прекратить работу отладчика можно, нажав клавишу F7 (ассемблирование программы).

> Ассемблирование программы приводит к автоматическому сохранению файлов проекта.

2.3. Советы

В заключение главы приведено несколько рекомендаций.

На первых порах кроме основного проекта удобно организовать еще один проект с пустым файлом программы. В этом проекте можно писать очень короткие программы, которые позволят проверить и понять работу команд, вызывающих затруднение или смысл которых не совсем понятен, а также разобраться с работой отладчика на более простых и коротких примерах.

Для получения более подробной информации по командам следует воспользоваться справкой, вызвав меню Help/AVR Tools User Guide, в открывшемся окне на вкладке Поиск введите наименование интересующей вас команды и нажмите кнопку Разделы, вы получите полную информацию о команде с примерами ее использования. То же можно найти на вкладке Содержание в разделе AVR Assembler/Instructions.

Справка по командам ассемблера будет полезна и для трактовки ошибок, обнаруженных при ассемблировании программы.

Старайтесь расшифровывать аббревиатуры команд — это позволит вам реже обращаться к списку команд и к справочной системе.

Прежде чем приступать к программированию и отладке, по техническому описанию микроконтроллера разберитесь с работой тех устройств микроконтроллера, которые вы собираетесь использовать.

При отладке старайтесь пользоваться полным набором горячих клавиш и всеми средствами просмотра. Например, состояние регистров можно проверять как в окне Register, так и в окне Workspace. Постепенно вы определите те средства и методы, которые удобны именно вам.

Сопоставьте сведения, полученные об организации стека в этой главе, с общей информацией об организации стека, приводящейся в большинстве книг по микропроцессорам (обычно в виде таблички с колонкой ячеек изображается область памяти, а рядом — указатель стека в виде стрелки).

Не забывайте о командах сброса сторожевого таймера, отладчик не симулирует его работу. Поэтому работающая в отладчике программа может постоянно сбрасываться, если вы забыли расставить в ней команды wdr.

Если вы делаете какие-то изменения в программе с тем, чтобы облегчить или ускорить процесс отладки, не ленитесь установить метки, по которым вы сможете найти команды с исправлениями. Для этого удобно использовать метки, устанавливаемые при нажатии Ctrl + F2. Найти в программе помеченные таким способом команды можно при нажатии F2. После восстановления команды метка убирается нажатием Ctrl + F2.

Еще один удобный способ — установка комментированных меток, например:

```
; + + +
        ldi tmp,1 ;5
```

Здесь фрагмент /; + + +/ отмечает исправленную на время отладки команду. В регистр tmp временно загружается единица, а значение 5, которое должно быть загружено в регистр в «боевой» программе, закомментировано, и вам не потребуется вспоминать, какой именно должна быть загружаемая величина.

Сочетание /; + + +/ вводится в окно поиска, вызываемое нажатием клавиш Ctrl + F2. После восстановления команды следующая метка /; + + +/ обнаруживается при нажатии клавиши F3.

Глава 3. Работа с внешним статическим ОЗУ

Одна из часто встречающихся задач — временное хранение данных, однако объема внутреннего ОЗУ зачастую оказывается недостаточно и приходится использовать внешнее ОЗУ.

Микроконтроллер ATmega8515 снабжен интерфейсом для подключения внешней памяти объемом до 64 Кбайт. Этот микроконтроллер полностью заменяет снятый с производства микроконтроллер AT90S8515, обладая дополнительными возможностями, а программы, написанные для AT90S8515, могут быть использованы без доработки в микроконтроллере ATmega8515. Такая возможность реализована также и в микроконтроллере ATmega8535, заменившем AT90S8535, для этого перед загрузкой программы в память микроконтроллеров ATmega8515 и ATmega8535 программируется Fuse-бит совместимости со старым микроконтроллером.

3.1. Интерфейс микроконтроллера ATmega8515 для подключения внешней памяти

Интерфейс включает в себя:

1. Порт A: шина младшего байта адреса/шина данных.
2. Порт C: шина старшего байта адреса.
3. Контакт ALE (Address Latch Enabled): разрешение фиксации адреса.
4. Контакты RD и WR: стробы записи и считывания.

Для инициализации работы интерфейса надо установить разряд SRE (Static RAM Enable — разрешение работы с внешней памятью) регистра MCUCR (MCU Control Register — регистр управления микроконтроллера). При установленном разряде SRE возможно использование стандартных команд работы с памятью (команды LD, LDD, LDS, ST, STD и STS) для обращения к внешнему ОЗУ (для записи или считывания), а микроконтроллер сможет обращаться к ячейкам памяти, адреса которых лежат в диапазоне 0x60...0xFFFF (в десятичном виде: 96...65535).

Свободно размещать данные в памяти микроконтроллера AT90S8515 можно начиная с адреса $0060. Адреса оперативной памяти

0x0000...0x001F заняты регистрами общего назначения R0...R31 (десятичное число 31 равно шестнадцатеричному числу 0x001F), адреса 0x0020...0x005F отведены для регистров ввода/вывода.

Для внутренней оперативной памяти микроконтроллера отведены адреса 0x0060...0x025F, во внешней памяти можно размещать данные в ячейках с адресами 0x0260...0xFFFF.

> Если до инициализации интерфейса внешнего ОЗУ линии порта A, порта C и контакты PD6 (WR) и PD7 (RD) порта D были запрограммированы как линии ввода или вывода, установка разряда SRE перепрограммирует эти линии для работы с внешней памятью.
> Если разряд SRE сбросить в ноль, работа с внешней памятью прекращается, устанавливается обычный режим работы портов A, C и контактов PD6 и PD7 порта D, а область оперативной памяти, к которой может обращаться микроконтроллер ATmega8515 с помощью команд обращения к памяти, ограничивается диапазоном 0x060...0x25F (всего 512 байт). При этом для обращения к внешней памяти потребуется группа команд, управляющих работой портов ввода/вывода.
> Обратите внимание на то, что количество адресуемых ячеек не 511 (0x25F – 0x060 = 0x1FF), а 512 (0x25F – 0x060 + 1 = 0x200), так как обращение происходит и к ячейке с адресом 0x060. Более наглядный пример можно продемонстрировать с двумя ячейками: 0x060 и 0x061 (0x061 – 0x060 + 1 = 2).

Порт A служит как для формирования младшего байта адреса, так и для передачи данных, в то время как порт C поддерживает на своих линиях старший байт адреса в течение всего цикла обращения к внешней памяти.

Для фиксации младшего байта адреса необходимо использовать дополнительный элемент хранения с записью положительным уровнем. Для этого подойдет 8-разрядный регистр 74HCT573N. Младший байт адреса, сформированный на восьми линиях порта A, записывается в регистр под действием импульса положительной полярности, появляющегося на контакте ALE микроконтроллера. Старший байт адреса устанавливается на выходах порта C.

Младший байт адреса записан и удерживается на выходах регистра, старший байт удерживается на выходах порта C.

В режиме записи на выходах порта A устанавливаются данные, предназначенные для записи во внешнее ОЗУ. На линии WR микроконтроллера формируется импульс отрицательной полярности, вызывающий запись данных.

При считывании из внешнего ОЗУ порт A переходит в режим приема данных, на линии RD микроконтроллера формируется импульс отрица-

тельной полярности и информация, присутствующая в это время на контактах порта А, считывается в микроконтроллер.

На линии WR большую часть времени присутствует ВЫСОКИЙ логический уровень. Для записи данных на этой линии на короткое время устанавливается НИЗКИЙ уровень, после чего на линии WR восстанавливается ВЫСОКИЙ уровень (формируется импульс отрицательной полярности). Если же упоминается импульс положительной полярности, то это означает, что основную часть времени на линии присутствует НИЗКИЙ уровень, а ВЫСОКИЙ уровень устанавливается лишь на короткое время.

Обращение к внешней памяти происходит за три такта микроконтроллера, однако можно удлинить время обращения на один дополнительный такт, установив разряд SRW (External SRAM Wait State — состояние ожидания внешней памяти) в регистре MCUCR.

3.2. Пример подключения внешнего ОЗУ к микроконтроллеру ATmega8515

3.2.1. Схема

Фрагмент схемы, в котором реализовано подключение микросхемы памяти 32K×8 (32 Кбайт) к микроконтроллеру, приведен на **Рис. 19**. В данной схеме используется только часть памяти (8 Кбайт).

Поскольку схема довольно большая, а формат книги вряд ли позволит напечатать ее целиком, я буду приводить отдельные фрагменты схемы. Связь между фрагментами схемы, представленными на разных рисунках, — по шине с общим наименованием (для этой схемы это шина BUS1). В связи с этим не удивляйтесь отсутствию сквозного позиционного обозначения схемных элементов. Например, на фрагменте, приведенном на **Рис. 19**, отсутствуют элементы DD1 и DD4 — в книге они могут быть представлены в другом фрагменте или совсем не представлены, если описание их работы не представляет интереса.

Для подключения с минимальным количеством дополнительных элементов пригодны микросхемы памяти, имеющие двунаправленную 8-разрядную шину данных (D0...D7), шину адреса и инверсные входы управления записью и считыванием (WE, OE).

На фрагменте схемы к микроконтроллеру ATmega8515 (DD2) подключена микросхема внешней памяти HM62256 (DD5) с организацией 32K×8,

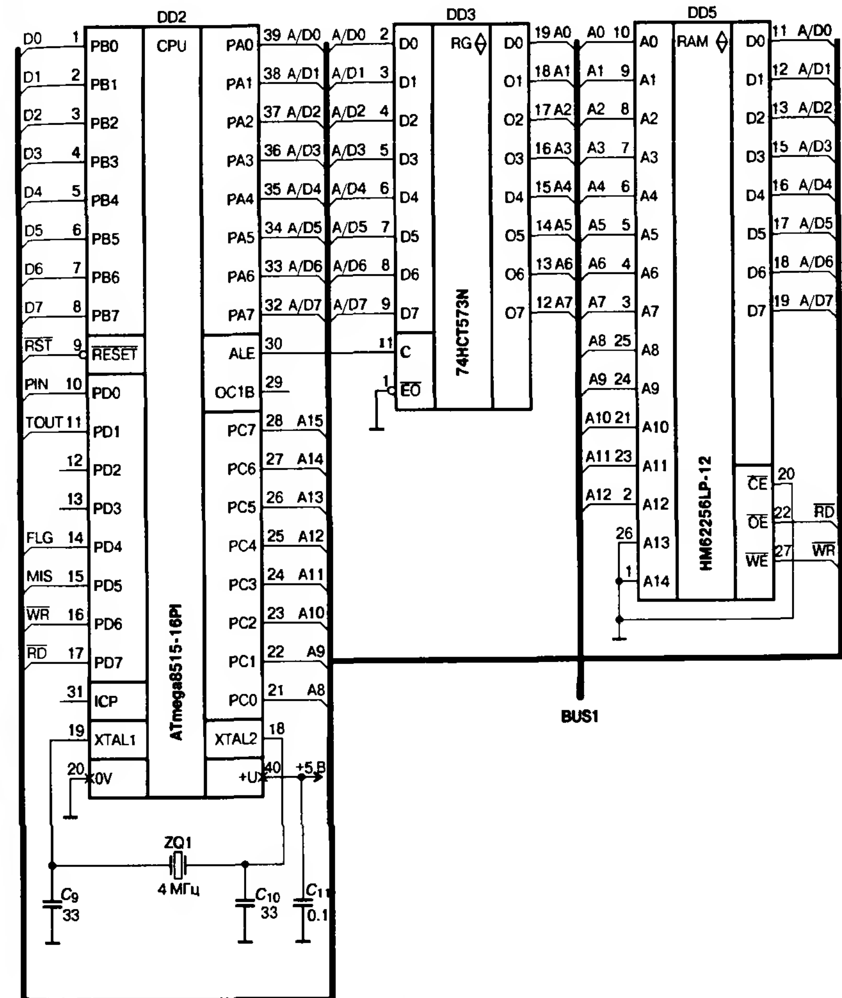


Рис. 19. Схема подключения внешнего статического ОЗУ к микроконтроллеру

из которых используется только 8K×8, что требует использования тринадцати линий адреса (A0...A12) из пятнадцати имеющихся. Свободные контакты A13 и A14 шины адреса микросхемы DD5 соединены с общим проводом. Эти контакты можно было соединить и с линией питания, что никак не сказалось бы ни на работе, ни на программе, просто для хранения данных в микросхеме использовался бы другой сегмент памяти.

Не использующиеся в приведенной схеме для работы с внешней памятью линии адреса микроконтроллера A13, A14 и A15 могут использоваться, например, для обращения к дополнительным регистрам, обеспечивающим передачу данных к внешним устройствам, о чем будет сказано ниже.

В режиме работы с внешним ОЗУ микроконтроллер ATmega8515 обеспечивает возможность использования старших незадействованных линий порта C в качестве обычных линий ввода/вывода, для этого используются разряды XMM2..XMM0 (eXternal Memory hig Mask) регистра специальных функций ввода/вывода SFIOR. По умолчанию значения этих разрядов равны нулю, и в режиме работы с внешним ОЗУ используется весь порт C. При значениях XMM2:XMM1:XMM0, равных 001, линия PC7 может использоваться как обычная линия ввода/вывода (далее – I/O), для 010 – линии PC7 и PC6, 011 – PC7, PC6 и PC5, ..., 111 – весь порт C – I/O.

3.2.2. Установка адреса

В регистр 74HCT573 (DD3) по ВЫСОКОМУ уровню сигнала на контакте ALE микроконтроллера производится запись младшего байта адреса, сформированного микроконтроллером на линиях A/D0...7. Записанный в регистр младший байт адреса появляется на выходах регистра, соединенных с адресными входами микросхемы памяти (линии A0...A7).

На контактах порта C микроконтроллера формируется старший байт адреса (линии A8...A15).

3.2.3. О выборе микросхемы регистра

Регистр 1533ИР33 является функциональным аналогом регистра 74HCT573, однако они выполнены по разной технологии. Попытка использования регистра 1533ИР33 для фиксации младшего байта при работе микроконтроллера с микросхемой памяти показала, что при записи-считывании массива /0, 1...255/ появляются сбои, число которых сокращается при подключении конденсатора емкостью в десятки пикофарад между контактом C регистра и общим проводом. Использование регистра 74HCT573 полностью решало проблему и без конденсатора. Поэтому к замене регистра следует относиться с пониманием.

Тестирование описанной схемы существенно облегчается, если есть возможность подключить ее к компьютеру, например, через COM-порт.

Использованию микроконтроллера UART (Universal asynchronous receiver-transmitter — универсальный асинхронный приемопередатчик) для обеспечения связи с COM-портом компьютера будет посвящена отдельная глава книги.

3.2.4. Считывание данных из внешней памяти

Линии порта A переводятся в режим ввода данных в микроконтроллер после установки адреса. На линии RD микроконтроллер формирует импульс отрицательной полярности. По сигналу RD шина данных D0...D7 микросхемы памяти DD5 переводится в режим вывода данных, и по линиям A/D0...A/D7 из адресуемой ячейки хранимые в ней данные передаются в порт A микроконтроллера.

3.2.5. Запись данных во внешнюю память

После того как адрес установлен, линии порта A переводятся в режим вывода данных из микроконтроллера, данные устанавливаются на контактах порта A, затем на линии WR микроконтроллер формирует импульс отрицательной полярности. По сигналу WR шина данных D0...D7 микросхемы памяти DD5 переводится в режим ввода данных, и по линиям A/D0...A/D7 из порта A микроконтроллера в адресуемую ячейку внешней памяти передаются данные для хранения.

3.3. Программный доступ к оперативной памяти

3.3.1. Простая программа обращения к оперативной памяти

Создайте проект SRAM.apr в папке SRAM с файлом программы SRAM.asm. Наберите следующую программу:

```
;Программа SRAM.asm
;Обращение к внешней памяти
;==========================
.include "c:\avr\def\m8515def.inc"
.def   tmp = r16
.def   cnt = r16
;Одному регистру в программе могут присваиваться разные имена,
;если в этом есть необходимость. В данном случае такой
;необходимости нет. Микроконтроллер имеет 32 регистра, но
;переменных в программе обычно больше
.equ   ArSize = 10
;Используем константу ArSize для определения размера массива,
;записываемого в память
.equ   aArBgn = $0270
```

```
;Используем константу aArBgn как адрес начальной ячейки для хранения
                                   массива
RESET: in      tmp,MCUCR           ;Ввод содержимого регистра MCUCR в
                                   ;регистр tmp. Физические адреса
                                   ;регистров MCUCR,ZL,ZH и константы
                                   ;SRE и SRW10, используемые в программе,
                                   ;определены в файле 8515def.inc.
                                   ;Если файл 8515def.inc не будет
                                   ;найден, команды с операндами
                                   ;MCUCR,ZL,ZH,SRE и SRW10 дадут ошибки
        ori    tmp,(1<<SRE) + (1<<SRW10)
;Операция ori (логическое ИЛИ, еще записывается как V)
;выполняется над содержимым регистра и константой.
;Для работы с внешней памятью надо установить
;разряды SRE и SRW10 в регистре MCUCR (установка SRW10 –
;когда нужно продлить состояние ожидания
;для «медленной» микросхемы памяти).
;Открыв файл 8515def.inc, обнаружим, что SRE = 7, SRW10 = 6.
;Значит величина, которую надо занести в регистр MCUCR в двоичном
;коде: 11bb bbbb, где b – это значения разрядов, которые мы не
;должны изменять. Запись (1<<SRE) + (1<<SRW10) равна сумме единицы,
;сдвинутой влево SRE (или 7) раз (= 1000 0000), и единицы,
;сдвинутой влево SRW10 (или 6) раз (0100 0000). Сумма 1000 0000 +
;0100 0000 = 1100 0000. Операция V (xxbb bbbb V 1100 0000 =
;= 11bb bbbb) дает нужный результат
        out    MCUCR,tmp           ;Вывод содержимого регистра
                                   ;tmp в регистр MCUCR
        ldi    ZL,low(aArBgn)
        ldi    ZH,high(aArBgn)
;ldi – операции загрузки в регистр однобайтной константы.
;aArBgn – двухбайтная константа – должна быть загружена
;в пару однобайтных регистров ZH:ZL, которые составляют
;двухбайтный регистр Z. Регистр Z и аналогичные ему
;двухбайтные регистры X и Y используются в операциях с
;памятью (st,ld,…). aArBgn = $0270, в ZL загрузится low($0270),
;т.е. младший байт, равный $70, а в ZH загрузится high($0270),
;т.е. старший байт $02. В Z (или в ZH:ZL) загрузится двухбайтное
;слово $02:$70, то есть, адрес, выбранный для начальной ячейки массива
        ldi    cnt,ArSize          ;Загрузить в cnt размер массива
NEXT:   st     Z +,cnt
;st – операция записи содержимого регистра cnt в ячейку памяти,
;адрес которой в регистре Z, знак « + » после Z – значит с
;последующим увеличением адреса в регистре Z на единицу. В первом
;цикле данные занесутся по адресу $0270, затем Z приобретет значение $0271.
;Во втором цикле – по адресу $0271, Z = $0272 и т.д. В память пишется
;состояние счетчика циклов

        dec    cnt                 ;Уменьшить содержимое регистра cnt на
                                   единицу
        nop                        ;Команда добавляет один пустой цикл
        nop                        ;То же
```



```
        brne   NEXT
;Если флаг Z в регистре состояния процессора SREG (не путать с
;регистром адреса Z) не установлен – перейти на команду с меткой
;NEXT:. Последняя команда, воздействующая на флаг Z – dec cnt.
;Команды nop введены для демонстрации отсутствия их влияния на
;флаг Z. Если вместо них вставить, например, команду inc tmp,
;влияющую на состояние флага Z, работа программы будет нарушена
;Далее – блок считывания данных из памяти
        ldi    ZL,low(aArBgn)
        ldi    ZH,high(aArBgn)
        ldi    cnt,ArSize
RD_BLK:
        ld     tmp, Z +
;Здесь производятся какие-то действия с tmp, например,
;передача в компьютер через UART микроконтроллера
        dec    cnt
        brne   RD_BLK
        rjmp   RESET               ;Перейти на команду с меткой RESET:
```

В этой программе нет обработчиков прерываний, кроме обработчика прерывания *RESET*, но этот обработчик и так начинается с нулевого адреса памяти программ, поэтому необходимости в блоке ссылок на обработчики прерываний в этой программе нет.

> Вы заметили, что имена регистров и константы, определенные во включаемом в программу файле m8515def.inc, довольно длинны? Не думайте, что, сокращая используемые имена до одной-двух букв, можно сэкономить время: через пару недель Вы не разберетесь в собственной программе! А вот имена MCUCR, SRE, являясь сокращениями английских наименований MCU control register и Static RAM Enable, быстро запоминаются. Те же рекомендации можно предложить и для определяемых вами имен: и в чужой программе можно понять, что ArSize — это Array Size (размер массива), aArBgn — адрес массива начальный, а cnt — счетчик.

3.3.2. Отладка программы

Ассемблируйте программу, выполните все уже знакомые вам действия по выбору устройства (ATmega8515 — с внешней памятью) и тактовой частоты (4 МГц).

Вероятно, программа покажется вам перегруженной комментариями, но они облегчают понимание программы.

Практически со всеми используемыми в ней командами мы уже работали.

Просмотреть содержимое регистра управления микроконтроллера MCUCR можно в окне Workspace/IO ATmega8515/CPU.

Здесь можно увидеть состояние интересующих нас разрядов SRE и SRW10, а при необходимости установить состояние какого-либо разряда прямо здесь, например, разряда SM1 (sleep mode), состояние которого не влияет на работу этой программы. Для этого установите галочку (щелкните) в квадратике, помеченном цифрой 4, находящемся правее слова SM1. Заметьте, состояние регистра (строка Control register) изменилось на 0x10.

Вызовем окно Watch, в нем будем проверять состояние переменных *tmp, cnt,* ZH и ZL. Нам понадобится также информация окон Workspace/Processor и Memory/Data.

> В окне Watch удобно группировать переменные в соответствии с размещением их в программе. Для одной части программы переменные можно разместить в группе, выбрав страницу окна Watch1 (смотрите нижнюю строку окна Watch), для другой части программы разместите переменные на странице Watch2 и т.д.
> Ввод имени константы вместо имени переменной в окно Watch (например, константа aArBgn из нашей программы) бессмысленно, оно воспринимается как имя не объявленной в программе переменной.

Выполните программу до цикла с меткой *Next:*.

Следующая часть программы — это цикл записи элементов массива в память. В окне Memory/Data в окошке справа установлен адрес 0x0060, он указывает адрес первой ячейки, которая видна в окне. Наша программа должна записывать массив, начиная с адреса 0x0270. Для удобства просмотра вместо адреса 0x0060 запишем в окошко адрес начальной ячейки массива 0x0270. Альтернативный вариант — переместиться к интересующей нас области памяти с помощью вертикального скроллера или колесика мышки, если оно есть.

Позаботьтесь об удобстве просмотра, выбрав размеры окон и их размещение на экране.

Все до сих пор выполнявшиеся команды программы занимали по одному такту микроконтроллера, мы выполнили 6 команд, что соответствует состоянию счетчика циклов (Cycle counter 00000006) в окне Processor.

Нажмите F11. В ячейку памяти 0x0170 записалось число $0A (окно Memory). Состояние счетчика циклов увеличилось лишь на два (окно Processor). Для команды *st* такое увеличение соответствует обращению лишь к внутренней памяти микроконтроллера. В нашем случае обращения к внешней памяти количество тактов должно было увеличиться на четыре, так как один дополнительный такт требуется для обращения к внешней памяти, и еще один такт, так как мы задали режим работы с дополнительным тактом ожидания, установив разряд SRW10 в регистре MCUCR. Не сделай мы этого, обращение происходило бы к внутренней оперативной памяти микроконтроллера, а число тактов отображалось бы верно.

> При отладке обращения к внешней памяти количество тактов, подсчитанное в окне Processor, не соответствует действительности.

Выполните несколько циклов, наблюдая за записью данных в ОЗУ, изменением переменных в окне Watch.

Замените вручную в окне Memory переменную в ячейке 0x0271, например, на CD.

Перейдите к блоку считывания данных из памяти, начинающемуся меткой *rd_blk:*, выполните несколько циклов, делая те же наблюдения. Обратите внимание на значение, считанное из ячейки 0x0271.

3.3.3. Сохранение содержимого ОЗУ на диске

Иногда бывает необходимо ввести в память множество данных вручную либо сохранить полученные результаты для того, чтобы использовать их, например, в другой программе. Для этого вызовем меню Debug/Up/Download Memories, и в окошке Memory Type выберем Extended SRAM, в окошке Hex File определим папку и имя сохраняемого файла, затем нажмем кнопку Save To File.

В следующем сеансе работы с AVR Studio вы можете загрузить в память данные из файла — для этого надо вызвать то же меню, выбрать загружаемый файл и нажать кнопку Load from File.

Все это можно выполнить и для электрически перепрограммируемой памяти микроконтроллера EEPROM.

3.3.4. Запись данных в начальную область внешней памяти

Внесем изменение в нашу программу: константе aArBgn присвоим значение 0x005E, теперь директива определения этой константы должна иметь вид: *.equaArBgn = $005E ;$0270 — прежнее значение закомментированно.*

Ассемблируем нашу программу и выполним ее до метки *NEXT:*. В регистре Z теперь хранится адрес 0x005E, по которому будет записано содержимое переменной *cnt*, равное 0x0A. Куда произойдет запись, ведь окно Memory позволяет просматривать содержимое ячеек, адрес которых не менее 0x0060?

Из технического описания микроконтроллера ATmega8515 следует, что 0x005E — это адрес старшего байта указателя стека (SPH), а 0x005F — адрес регистра флагов.

Откроем окно Processor для наблюдения за состоянием указателя стека и регистра флагов. После первой записи по адресу 0x005E содержимое указателя стека изменяется на 0x00000A00 (записано значение 0x0A), после

второй записи по адресу 0x005F в регистре флагов устанавливаются флаги V и C (записано значение 0x09). И только в третьем цикле запись производится в ячейку памяти 0x0060.

Запись данных в начальную область внешней памяти (0x0000... 0x005F) приводит к ошибкам при отладке программы в AVR Studio.

Поскольку разница во времени обращения к внешнему и внутреннему ОЗУ ощутима, при необходимости можно быстро записывать данные небольшими блоками во внутреннее ОЗУ микроконтроллера, а затем блоками переносить данные во внешнее ОЗУ. Для этого понадобится манипулировать разрядами SRE и SRWxx регистра MCUCR, чередуя режимы обращения к внутреннему и к внешнему ОЗУ.

Вы можете самостоятельно проверить, можно ли одновременно хранить разные данные, скажем, в ячейке 0x0060 внутреннего ОЗУ и в ячейке внешнего ОЗУ с тем же адресом. Следует отметить, что отладчик AVR Studio в этом вам не поможет, такую проверку можно сделать только на работающем контроллере с подключенной внешней памятью.

Как произвести такую проверку?

Для этого понадобится контроллер с подключенной внешней памятью и пара светодиодов с резисторами. Не забудьте о кварцевом резонаторе и о подсоединяемой к контакту RESET микроконтроллера *RC*-цепочке. Мы рассматривали ранее, как подключаются такие элементы, и вы знаете, как записать данные в ячейки внешней и внутренней памяти, как включить и выключить светодиод, подключенный к микроконтроллеру.

Алгоритм проверки следующий. Директивами *.def* зарезервировать регистры для четырех переменных: A, B, C, D. Переменным A и B присвоить разные значения, например $A5 для A и $7D для B. Переменные C и D очистить.

В регистр Z занести адрес ячейки памяти, существующей как во внешнем, так и во внутреннем ОЗУ, например 0x0070.

В режиме записи во внутреннюю память сохранить содержимое переменной A в ячейке, адрес которой хранится в регистре Z.

В режиме записи во внешнюю память сохранить содержимое переменной B в ячейке, адрес которой хранится в регистре Z.

В режиме считывания из внутренней памяти загрузить содержимое ячейки, адрес которой хранится в регистре Z, в переменную C.

В режиме считывания из внешней памяти загрузить содержимое ячейки, адрес которой хранится в регистре Z, в переменную D.

Сравнить значения переменных A и C: если они равны, включить светодиод № 1.

Сравнить значения переменных B и D: если они равны, включить светодиод № 2.

3.4. Обращение к буферам как к ячейкам памяти микроконтроллера ATmega8515

3.4.1. Электрическая схема подключения буферов

В предыдущем примере рассматривалось подключение внешней памяти к микроконтроллеру ATmega8515. Из четырех портов микроконтроллера порты A и C, а также два контакта порта D заняты обслуживанием внешней памяти.

Как быть, если в дополнение к подключению внешней памяти необходимо организовать передачу двух десятков управляющих сигналов от микроконтроллера к внешним устройствам? Возможный вариант — использование необходимого количества 8-разрядных регистров (буферов) с обращением к ним как к ячейкам памяти, расположенным в общем адресном пространстве внешней памяти. На **Рис. 20** изображен фрагмент схемы, в которой реализовано такое подключение. Данный фрагмент и фрагмент, представленный на **Рис. 19**, являются частями одной схемы, а шина BUS1 является общей для обоих рисунков.

При подключении внешней памяти объемом 8K×8 три адресные линии A13...A15 остались свободными, используем их для записи данных в три 8-разрядных регистра DD6...DD8.

Использование дополнительной микросхемы дешифратора 3-разрядного адреса A13...A15 позволило бы подключить до восьми (2^3) регистров.

Входные контакты D0...D7 регистров DD6...DD8 подключены к порту A микроконтроллера (линии A/D0...A/D7). К выходным контактам O0...O7 регистров можно подключить до 24 линий внешних устройств (линии B0...B23). Контакты EO регистров соединены с общим проводом, поэтому выходы регистров всегда активны и не переводятся в третье состояние.

Запись данных в регистры происходит при появлении ВЫСОКОГО уровня на их контактах C, микроконтроллер же формирует НИЗКИЙ уровень записи во внешнюю память (сигнал WR). Для формирования сигнала

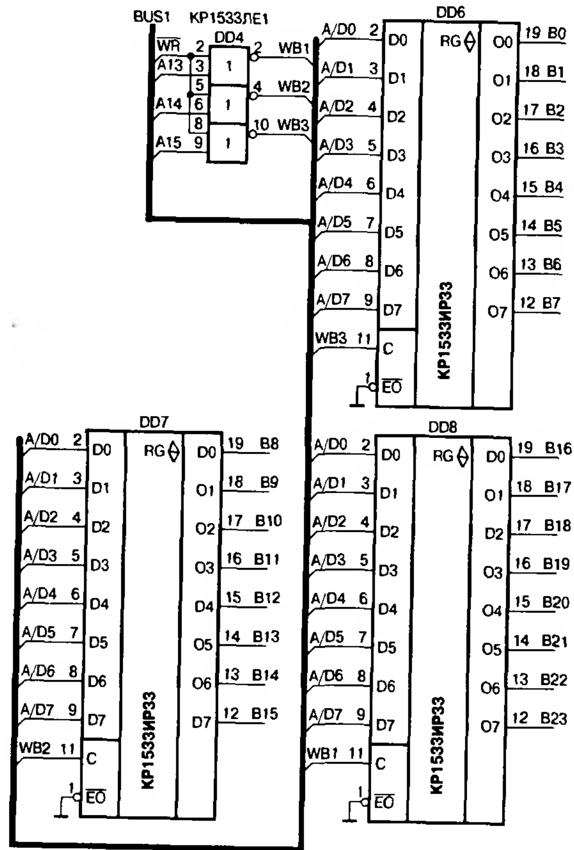


Рис. 20. Подключение буферов с обращением к ним как к ячейкам памяти

записи ВЫСОКОГО уровня в схеме применены элементы ИЛИ-НЕ (микросхема DD4).

Для записи данных в регистр DD6 на линии A15 необходимо установить НИЗКИЙ уровень. Тогда при появлении импульса отрицательной полярности на линии WR микроконтроллера данные, установленные на шине A/D0...A/D7, запишутся в регистр DD6 независимо от состояния других адресных линий.

Аналогично производится запись в регистр DD7 при НИЗКОМ уровне на линии A14 и в регистр DD8 при НИЗКОМ уровне на линии A13.

Для того чтобы запись в микросхему внешней памяти не вызывала записи в регистры, на линиях A13...A15 необходимо поддерживать ВЫСОКИЕ уровни.
Одновременное присутствие НИЗКИХ уровней на всех четырех линиях (WR, A13...A15) вызывает параллельную запись данных в три регистра.

Приведенная схема подключения регистров требует тщательного подхода к выбору адресов при написании программы.

Запись в регистр DD6 должна происходить при A15 = 0, A14 = 1 и A13 = 1, то есть в двоичном коде адреса регистров DD6, DD7 и DD8 будут /011x xxxx xxxx xxxx/, /101x xxxx xxxx xxxx/ и /110x xxxx xxxx xxxx/ соответственно.

Для того чтобы данные не записывались в регистры, адрес должен быть /111x xxxx xxxx xxxx/, где x — состояние разряда (0 или 1 не имеет значения).

Хотя состояния разрядов x не имеют значения, они должны быть определены. Это значит, что при записи в какой-либо регистр на контактах A0...A12 микросхемы памяти (схема на **Рис. 19**) будет сформирован адрес, а в ячейку с этим адресом будет произведена запись того же значения, которое посылается в регистр. Надо позаботиться, чтобы ячейка с таким адресом не использовалась для хранения данных.

Удобно определить все разряды x равными единице. Тогда адреса регистров примут вид:

адрес DD6: /0111 1111 1111 1111/ или $7FFF,

адрес DD7: /1011 1111 1111 1111/ или $BFFF,

адрес DD6: /1101 1111 1111 1111/ или $DFFF,

а в ячейке памяти с адресом /xxx1 1111 1111 1111/ нельзя хранить данные.

Линии адреса A13...A15 к микросхеме памяти не подключены. Поэтому, какими бы они ни были, запись производится в одну и ту же ячейку.

Но при записи в микросхему памяти xxx следует определить как 111 для того, чтобы не произошло параллельной записи в регистры. Значит, данные нельзя хранить в ячейке /1111 1111 1111 1111/ или $FFFF.

Для хранения данных доступны ячейки памяти с адресами /1110 0000 0000 0000 ... 1111 1111 1111 1110/ или $E000...$FFFE.

В разделе «памяти» была обозначена нижняя граница доступной для безопасного хранения данных области памяти: адрес $0060.

С учетом этого ограничения для безопасного хранения данных доступны ячейки памяти с адресами $E060...$FFFE. Именно этот диапазон адресов должен использоваться в программе для работы с данными, в микросхему памяти данные будут записываться по соответствующим физическим адресам в диапазоне $0060...$1FFE.

Есть более привлекательный вариант распределения памяти: при записи в регистр параллельно производить запись в соответствующую этому регистру ячейку памяти. При этом всегда можно проверить, что именно записывалось в регистр, считав данные из ячейки памяти.

Выберем для регистров следующие адреса:

адрес DD6: /0111 1111 1111 1101/ или $7FFD,

адрес DD7: /1011 1111 1111 1110/ или $BFFE,

адрес DD6: /1101 1111 1111 1111/ или $DFFF.

Теперь для безопасного хранения данных доступны ячейки памяти с адресами $E060...$FFFC, а из ячеек с адресами $FFFD, $FFFE и $FFFF можно считать данные, которые были записаны в регистры DD8, DD7 и DD6 соответственно.

> Считывая данные из ячеек, соответствующих регистрам, мы получаем информацию о том, что записывалось в эти регистры. Считывание данных правомерно только после первой записи в регистры. Поэтому следует записать исходные данные в регистры в начале программы.

Состояние выходов регистров может не соответствовать считанным из соответствующих ячеек памяти данным из-за неисправности регистров или из-за возможного соединения выхода регистра с общим проводом во внешнем устройстве. Иначе говоря, если на каком-либо выходе регистра установлен ВЫСОКИЙ уровень, то замыкание во внешнем устройстве на общий провод приводит к тому, что на этом выводе обнаруживается НИЗКИЙ уровень.

3.4.2. Программа обслуживания буферов

В AVR Studio создадим новый проект Buffers в папке C:\Avr\Buff\, в проекте создадим новый файл BUF_SRAM.asm, в котором напишем следующую программу:

```
;Программа BUF_SRAM.asm
;Обращение к внешней памяти, запись данных в буферы
;================================================
.include "C:\Avr\def\8515def.inc"
.def   tmp = r16
.def   cnt = r17                  ;Переменная cnt - счетчик цикла
RESET: in     tmp,MCUCR           ;Ввод содержимого регистра MCUCR в
                                  ;регистр tmp
       ori    tmp,(1<<SRE)        ;Установка разряда SRE (режим работы
                                  ;с внешней памятью). Разряды SRWxx не
                                  ;устанавливаем (работа без
                                  ;дополнительного цикла ожидания)
```



```
       out    MCUCR,tmp           ;Вывод содержимого регистра tmp в
                                  ;регистр MCUCR
;======== Загрузка данных в буферы
.equ   wrBuf1 = $7FFD             ;Адрес для записи данных в буфер 1
                                  ;(по схеме - элемент DD6)
.equ   wrBuf2 = $BFFE             ;Адрес для записи данных в буфер 2 (DD7)
.equ   wrBuf3 = $DFFF             ;Адрес для записи данных в буфер 3 (DD8)
.equ   rdBuf1 = $FFFD             ;Адрес для считывания данных из ячейки,
                                  ;соответствующей буферу 1
.equ   rdBuf2 = $FFFE             ;Адрес для считывания данных из ячейки,
                                  ;соответствующей буферу 2
.equ   rdBuf3 = $FFFF             ;Адрес для считывания данных из ячейки,
                                  ;соответствующей буферу 3
       ldi    tmp,$AA             ;Загрузка константы в tmp
       sts    wrBuf1, tmp         ;Сохранение содержимого
                                  ;регистра tmp в буфере 1
       ldi    tmp,$55             ;Загрузка константы в tmp
       sts    wrBuf2, tmp
       ldi    tmp,$71             ;Загрузка константы в tmp
       sts    wrBuf3, tmp
;======== Чтение/изменение данных в буферах
       lds    tmp, rdBuf1         ;Передача данных из ячейки с адресом
                                  ;rdBuf1 в регистр tmp
       ori    tmp, (1<<7) + (1<<6) + (1<<5) + (1<<4) ;Установка
                                  ;разрядов 4…7
       sts    wrBuf1, tmp         ;Сохранение содержимого регистра tmp
                                  ;в буфере 1
       lds    tmp, rdBuf2
       ori    tmp, 0b11110000     ;Установка тех же разрядов, что
                                  ;и для буфера 1. Только константа
                                  ;представлена в двоичной форме. Для
                                  ;обозначения двоичной константы
                                  ;1111 0000 перед ней ставят символы
                                  ;«ноль» и «b»
       sts    wrBuf2, tmp         ;Сохранение содержимого регистра tmp
                                  ;в буфере 2
       lds    tmp, rdBuf3
       andi   tmp, 0b11110000     ;Сброс тех же разрядов
       sts    wrBuf3, tmp         ;Сохранение содержимого регистра tmp
                                  ;в буфере 2
;======== Загрузка в память первого массива
.equ   ArSize = 10                ;Размеры массивов
.equ   aArBgn1 = $E060            ;Используем константу aArBgn как
                                  ;адрес начальной ячейки
                                  ;для хранения массива 1
       ldi    ZL,low(aArBgn1)
       ldi    ZH,high(aArBgn1)    ;Загрузка в регистр Z адреса
                                  ;начала массива 1
       ldi    cnt,ArSize          ;Загрузка в cnt размера массива
       ldi    tmp,$FF             ;Загрузка константы в tmp
[illegible]r1: st  Z+,tmp         ;Запись содержимого регистра tmp в
```

```
                                   ;ячейку памяти, адрес которой - в
                                   ;регистре Z, с последующим
                                   ;увеличением адреса в регистре Z
                                   ;на 1. В первом цикле содержимое tmp
                                   ;запишется по адресу aArBgn1 ($E060),
                                   ;во втором цикле - по адресу
                                   ;aArBgn1 + 1 ($E061) и т.д.
       inc    tmp                  ;Увеличить содержимое tmp на единицу
       dec    cnt                  ;Уменьшить содержимое счетчика циклов
                                   ;на единицу
       brne   arr1                 ;Если разряд (флаг) Z в регистре
                                   ;состояния процессора SREG не
                                   ;установлен - перейти на команду с
                                   ;меткой ARR1:
;======= Загрузка в память второго массива
.equ   aArBgn2 = $FFFC             ;Используем константу aArBgn2 как
                                   ;адрес последней ячейки для хранения
                                   ;массива 2
       ldi    ZL,low(aArBgn2 + 1)
       ldi    ZH,high(aArBgn2 + 1);Загрузка в двухбайтный
                                   ;регистр Z адреса конечного элемента
                                   ;массива 2, увеличенного на единицу
       ldi    cnt,ArSize           ;Загрузка в cnt размера массива
       ldi    tmp,$03              ;Загрузка константы в tmp
arr2:         st - Z,tmp           ;Сначала уменьшается на единицу
                                   ;адрес, хранящийся в регистре Z
                                   ;(поэтому в Z загружался адрес,
                                   ;увеличенный на единицу). Затем по
                                   ;новому адресу запишется содержимое
                                   ;регистра tmp. В первом цикле
                                   ;содержимое tmp запишется по адресу
                                   ;(aArBgn2 + 1) - 1, то есть по
                                   ;адресу aArBgn2, во втором цикле - по
                                   ;адресу aArBgn2 - 1 и т.д.
       dec    tmp                  ;Уменьшить содержимое tmp на единицу
       dec    cnt                  ;Уменьшить содержимое счетчика циклов
                                   ;на единицу
       brne   arr2                 ;Если разряд (флаг) Z в регистре
                                   ;состояния процессора SREG не
                                   ;установлен - перейти на команду с
                                   ;меткой ARR2:
STOP : rjmp   STOP                 ;Зацикливание программы (перейти на
                                   ;команду с меткой STOP:)
```

3.4.3. Отладка программы обслуживания буферов

Ассемблируем программу и, установив опции симулятора (тактовая частота 4 МГц), приступим к отладке.

Для контроля отладки вызовем окна переменных, процессора и памяти: View/Watch, Processor и Memory.

В окно Watch введем переменные *tmp* и *cnt*.

Первые три команды были отлажены в проекте Memory.apr, поэтому переместим курсор на первую команду «Загрузки данных в буфер» (*ldi tmp,$AA*) и нажмем Ctrl + F10.

Нажмем F11 — в переменную tmp загрузилась константа $AA (смотрите в окне Watch).

В окне просмотра памяти Memory перейдем на адрес $7FFD (адрес *wrBuf1*). Для этого в окошке ввода адреса окна Memory заменим 0x0060 на 0x7FFD. Нажимаем F11 — в ячейке с адресом 7FFD появились данные AA.

Таким же способом переходим к ячейке $BFFE (*wrBuf2*) (дважды нажимаем F11 и проверяем состояние ячейки — оно равно 55).

Повторяем действия для ячейки $DFFF — ее состояние станет равным 71.

Теперь указатель выполнения команд находится на первой команде «Чтения/изменения данных в буферах» (*lds tmp, rdBuf1*).

К сожалению, отладчик не позволит правильно отобразить процессы, происходящие в нашей схеме из-за того, что симулятор предполагает подключение к микроконтроллеру лишь внешней памяти размером 65536 Кбайт, поэтому параллельная запись данных в буфер и в соответствующую ячейку памяти никак не отражается симулятором.

Возможный вариант — до выполнения команды *lds* вручную загрузить в ячейки с адресами *rdBuf1, rdBuf2 и rdBuf3* данные, которые должны были в них загрузиться при записи в буферы 1...3 ($AA, $55 и $71 соответственно).

Итак, в окне Memory перейдем к ячейке с адресом $FFFD (*rdBuf1*). В этом случае удобнее воспользоваться скроллером, переместив его в самый низ, так как интересующие нас ячейки с адресами $FFFD, $FFFE, $FFFF (*rdBuf1, rdBuf2 и rdBuf3* соответственно) являются тремя последними ячейками, которые могут отображаться в окне Memory.

Щелкнем на третьей от конца ячейке — ее адрес в виде 0xFFFD отобразится в окошке адреса (справа в первой строке окна Memory). Вместо содержимого этой ячейки (скорее всего 00) введем AA, курсор автоматически перейдет на следующую ячейку с адресом 0xFFFE, введем в нее значение 55, в ячейку 0xFFFF — значение 71.

Еще один вариант (здесь не реализован, проверьте его отладку самостоятельно): после команд записи (*sts wrBuf1,tmp*) добавлять команды *sts rdBuf1,tmp* и аналогичные команды для буферов 2 и 3. Что при этом произойдет?

В реальной схеме при выполнении первой из команд запись произойдет как в буфер, так и в микросхему памяти (подробности смотрите в описании схемы), а вторая команда еще раз запишет те же данные в ту же ячейку микросхемы памяти.

При отладке одинаковые данные запишутся в два адреса: один — имитирующий ячейку микросхемы памяти, второй — имитирующий буфер.

При этом программа несколько удлиняется. Однако после отладки дополнительные команды можно удалить.

Продолжим отладку. Следя за ячейками памяти *wrBuf1*, *wrBuf1* и *wrBuf3*, а также за переменной *tmp*, выполним 9 команд (9 нажатий F11).

Теперь указатель выполнения команд находится на первой команде «Загрузки в память первого массива».

В окне Memory перейдем к ячейке с адресом 0xE060 (*aArBgn1*). Переведем курсор на команду *inc tmp* и нажмем Ctrl + F10. В ячейке с адресом 0xE060 появился байт данных FF, соответствующий значению переменной *tmp*.

Нажмем F11 — выполнение команды *inc* привело к тому, что переменная *tmp* стала равна нулю, в результате чего установился флаг Z (окно Processor). Поэтому если бы команда *inc* стояла перед командой *brne*, выполнился бы лишь один цикл записи элементов массива в память.

Замечание. $FF + 1 = $100, то есть результат увеличения содержимого *tmp* на единицу командой *inc* — двухбайтная величина, но поскольку регистр может хранить только один байт, в нем остается младшая часть, то есть ноль, а установка флага Z свидетельствует о переполнении регистра tmp в результате выполнения команды. Аналогичным образом $00 – 1 = $FF, при этом вычитание происходит как бы из двухбайтного числа $100.

Нажимаем F11 — выполнение команды DEC уменьшило до 9 значение переменной *cnt*, то есть результат не нулевой, поэтому сбросился флаг Z (Processor), установленный после выполнения команды *inc*.

Отладка аналогичного цикла уже комментировалась, поэтому можно установить курсор на команде с меткой *arr2:* и нажать Ctrl + F10. Заметьте, в регистр Z (окно Processor) записался адрес 0xFFFD, соответствующий константе *aArBgn2* + 1.

В окне Memory перейдем к ячейке 0xFFFD, нажмем F11: содержимое регистра tmp оказалось в ячейке с адресом 0xFFFC (*aArBgn2*).

Отладка данного цикла не требует комментариев.

Установим курсор на последней команде программы и нажмем Ctrl + F10. Проконтролируйте содержимое ячеек памяти, в которые записался второй массив.

Что будет, если убрать последнюю команду? Для проверки удобнее заменить ее командой *nop*. Ассемблируем программу, нажмем F11, установим курсор на последней команде *nop* и нажмем Ctrl + F10. В окне Processor/Cycle counter заметим число циклов процессора (153). Нажмем F11: выполнение программы продолжилось с первой команды программы, но посмотрите на число циклов процессора: их стало 4206! После того как все команды нашей программы были извлечены из памяти программ и выполнены, продолжался поиск команд в свободной части памяти программ, при переходе к новой свободной ячейке счетчик циклов процессора увеличивал свое состояние, а вместе с ним и счетчик команд (Processor/Program counter). Максимальное значение, которое может храниться в этом счетчике для микроконтроллера ATmega8515, равно 4095, следующее значение — снова ноль, поэтому и произошел переход в начало программы, а к содержимому счетчика циклов процессора добавились циклы загрузки отсутствующих команд.

Замечание. Обычно микроконтроллеры работают в бесконечном цикле. Если программа должна остановиться на последней команде в ожидании прерываний, установите в конце программы команду, обеспечивающую бесконечный цикл: *Stop: rjmp Stop*.

3.5. Подключение внешней памяти 512 Кбайт к микроконтроллеру ATmega8535

В серии ATmega есть микроконтроллеры со встроенным аналого-цифровым преобразователем (АЦП). Часто встречающаяся при разработке контроллеров задача — преобразование аналоговых сигналов в код в реальном времени с записью полученного кода в память. Однако внутренней памяти микроконтроллеров со встроенным АЦП для этого обычно не достаточно, интерфейса для подключения внешней памяти они не имеют.

Здесь представлен возможный вариант подключения внешней памяти большого объема к микроконтроллеру ATmega8535. Микроконтроллер снабжен встроенным 10-разрядным АЦП с восемью входами, коммутируемыми программно. В качестве аналоговых входов АЦП в микроконтроллере используются контакты порта A.

3.6. Схема подключения ОЗУ к микроконтроллеру ATmega8535

На **Рис. 21** представлен фрагмент схемы, реализующий подключение микросхемы статической памяти K6T4008C1B-GB55 (DD5) к микроконтроллеру ATmega8535-16PI (DD6).

Микросхема памяти K6T4008C1B-GB55 производства фирмы Samsung Electronics по своим функциям не отличается от аналогичной микросхемы HM62256, использованной в схеме подключения внешней памяти к микроконтроллеру ATmega8515 (смотрите предыдущий пример). Процедура записи и считывания данных для обеих микросхем одинакова.

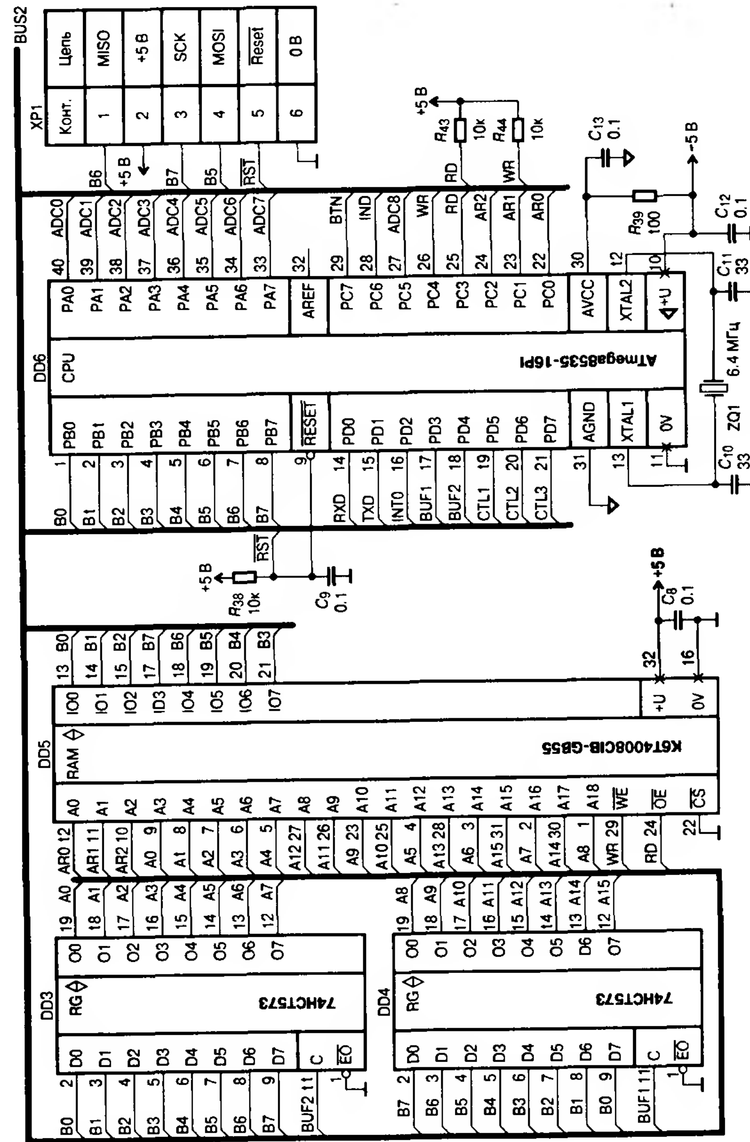


Рис. 21. Схема подключения внешней памяти 512 Кбайт к микроконтроллеру ATmega8535

Одинаково функционируют их двунаправленные 8-разрядные шины данных (контакты IO0...IO7), а также линии записи (контакт WR), управления шиной данных (контакт OE) и выбора микросхемы (контакт CS).

Отличие состоит в объеме памяти: 19-разрядная шина адреса (контакты A0...A18) микросхемы K6T4008C1B-GB55 обеспечивает обращение к 2^{19} ячейкам памяти, это значит, что в микросхеме можно хранить до 512 Кбайт данных (512K×8).

3.6.1. Описание схемы

В **Табл. 3** описаны линии, по которым микроконтроллер передает сигналы управления микросхемой памяти.

В исходном состоянии микроконтроллер устанавливает ВЫСОКИЕ уровни на линиях WR и RD, НИЗКИЕ уровни на линиях BUF1 и BUF2.

Таблица 3. **Линии связи микроконтроллера с микросхемой памяти**

Линия	Контакт микроконтроллера	Контакт микросхемы памяти	Назначение
WR	DD6/PC4	DD5/WE	Запись данных в DD5 при НИЗКОМ уровне
RD	DD6/PC3	DD5/OE	Считывание данных из DD5 при НИЗКОМ уровне
BUF1	DD6/PD3	DD4/C	Запись старшего байта адреса в регистр DD4 при ВЫСОКОМ уровне
BUF2	DD6/PD4	DD3/C	Запись младшего байта адреса в регистр DD3 при ВЫСОКОМ уровне
AR0...AR2	DD6/PC0...PC2	DD5/A0...A2	Адрес страницы

Программно организовано постраничное обращение к памяти, то есть все адресное пространство памяти как бы разбито на 8 страниц по 64 Кбайт в каждой. Три разряда адреса из 19 микроконтроллер формирует на линиях AR0...AR2, выбирая одну из восьми страниц памяти.

Адрес внутри страницы формируется двумя 8-разрядными регистрами 74HCT573 (DD3 и DD4).

С этой целью на шину данных (линии B0...B7) через порт B микроконтроллера (контакты PB0...PB7) выводится младший байт адреса, после чего микроконтроллер устанавливает ВЫСОКИЙ уровень на линии BUF2 для записи младшего байта адреса в регистр DD3. Затем на линии BUF2 устанавливается НИЗКИЙ уровень.

Далее на шину данных через порт B выводится старший байт адреса, запись которого в регистр DD4 происходит по установке микроконтролле-

ром ВЫСОКОГО уровня на линии BUF1. После записи на линии BUF1 устанавливается НИЗКИЙ уровень.

Теперь, когда на линиях A0...A18, следовательно, и на контактах A0...A18 микросхемы памяти адрес ячейки установлен, можно производить запись или считывание.

Замечание. Некоторое смущение может вызвать то, что к контактам микросхемы памяти A0...A18 подсоединены линии адреса с другими именами (к контакту A17, например, линия A14). Это сделано для удобства трассировки печатной платы, на которой размещаются элементы схемы. Для микросхемы памяти безразлично, в какой последовательности выбираются ячейки.
Допустим, при записи на линиях A18...A0 был, например, установлен адрес /000 00000001 00000000/, то есть единица — только на линии A8, подходящей к контакту A18 микросхемы памяти.
Значит, физически байт данных будет записан в ячейку /100 00000000 00000000/. Если при считывании мы вновь установим тот же адрес на линиях A18...A0, то физически считывание произойдет из той же ячейки памяти, в которую производилась запись.
Если Вы занимались трассировкой, то представляете, как были перепутаны дорожки адреса до оптимизации соединений линий адреса с контактами адреса микросхемы памяти. Следует отметить, что при обмене данными между несколькими адресуемыми устройствами к такой оптимизации надо подходить крайне осторожно.

3.6.2. Запись в ячейку

После формирования адреса микроконтроллер выводит через порт B на шину данных B0...B7 байт информации. Затем микроконтроллер устанавливает НИЗКИЙ уровень на линии WR, что приводит к записи байт информации в ячейку памяти. После записи на линии WR устанавливается ВЫСОКИЙ уровень.

3.6.3. Считывание из ячейки

После формирования адреса микроконтроллер переводит порт B в режим ввода данных в микроконтроллер. Затем на линии RD микроконтроллер устанавливает НИЗКИЙ уровень, что приводит к выводу данных из адресуемой ячейки памяти на шину данных B0...B7. Микроконтроллер через порт B считывает байт данных, установленный на линиях B0...B7, после чего устанавливает ВЫСОКИЙ уровень на линии RD.

Замечание. Обратите внимание на резисторы R_{43} и R_{44}.
При программировании микроконтроллера через разъем XP1 линии портов микроконтроллера имеют высокий импеданс, а поскольку входы WE и OE микросхемы памяти также имеют высокий импеданс, наводки на подключенных к ним линиях могут привести как к записи в произвольную ячейку памяти, так и к выводу данных из произвольной ячейки на линии B0...B7.
Последняя ситуация опасна, так как запись программы в микроконтроллер производится с использованием линий B5...B7 (разъем для подключения программатора XP1 изображен справа на **Рис. 21**). Поэтому возможны сбои при программировании (они действительно наблюдались в отсутствие резистора R_{43}), а также выход из строя программатора (маловероятно из-за достаточной стойкости используемой в нем микросхемы) или микросхемы памяти (вполне возможно).
Установка резистора R_{43}, подключенного к источнику питания + 5 В, создает в описанной ситуации на линии RD ВЫСОКИЙ уровень и переводит выходы IO0...IO7 микросхемы памяти в высокоимпедансное состояние, исключая подобную ситуацию, резистор R_{44} устраняет возможность хаотичной записи при программировании. В обычном режиме работы эти резисторы не мешают микроконтроллеру управлять микросхемой памяти.

3.7. Программа записи данных в ОЗУ 512 Кбайт

В рабочей программе, взятой за основу для нашего примера, АЦП микроконтроллера производил группу преобразований, результаты которых записывались во внутреннюю оперативную память микроконтроллера. Результаты группы преобразований обрабатывались и также записывались во внутреннюю оперативную память в виде массива размером в 45 байт, начиная с ячейки с адресом *aPack* (смотрите программу ниже). Этот массив переносился во внешнюю память для хранения. АЦП выполнял новую группу преобразований, которые тем же способом обрабатывались, а очередной массив размером в 45 байт добавлялся во внешнюю память. Процесс продолжался до заполнения страницы внешней памяти (64 Кбайт), после чего вся информация со страницы внешней памяти через микроконтроллер передавалась по запросу в компьютер.

Представленная программа включает в себя подпрограммы, взятые из рабочей программы, оттуда же заимствовано распределение внутренней оперативной памяти микроконтроллера, а также имена констант и переменных.

Создадим новую директорию c:\Avr\Ram512, поместим в нее файл m8535def.inc, полученный при распаковке файла avr000.exe (ищите его там

же, где распаковался файл m8515def.inc). В директории c:\Avr\Ram512 создадим проект Ram512 с новым файлом программы *Ram512.asm*, в который следует перенести следующую программу:

```
;Подключение ОЗУ512К к AT90S8535
;===============================
.include "c:\Atmel\8535def.inc"
.equ   RamH = $16a                ;Адрес ячейки внутренней SRAM, хранящей
                                  ;старший байт адреса страницы внешней
                                  ;памяти
.equ   RamL = $16b                ;Адрес ячейки внутренней SRAM, хранящей
                                  ;младший байт адреса внешней памяти и три
                                  ;старших разряда адреса внешней памяти
                                  ;(номер страницы внешней памяти)
.equ   Ar0 = PC0
.equ   Ar1 = PC1
.equ   Ar2 = PC2
.equ   Wr = PC4                   ;Управление записью во внешнюю память
.equ   Rd = PC3                   ;Управление считыванием из внешней памяти
.equ   Buf1 = PD3                 ;Управление записью в буфер 1
.equ   Buf2 = PD4                 ;Управление записью в буфер 2
.equ   aADC = $60                 ;Адрес для переноса массива из внешней
                                  ;памяти во внутреннюю SRAM
.equ   aPack = $b0                ;Адрес 45-байтового массива во
                                  ;внутренней SRAM микроконтроллера
.def   tm = r16
.def   cnt = r19
RESET:
ldi    tm,(1<<Wr) + (1<<Rd)       ;Установить ВЫСОКИЕ уровни на линиях
       out    PORTC,tm            ;WR и RD
       ldi    tm,$ff              ;Контакты порта C в режиме выходов
       out    DDRC,tm
       clr    tm                  ;На линиях порта D НИЗКИЕ уровни,
       out    PORTD,tm            ;в том числе BUF1,BUF2 = 0
       ldi    tm,$ff              ;Контакты порта D в режиме выходов
       out    DDRD,tm
       ser    tm                  ;Установка регистра tm (tm = $ff)
       out    DDRB,tm             ;Вывод в порт DDRB содержимого tm
                                  ;Контакты порта B в режиме выходов
       ldi    tm,low(RAMEND)      ;Стек – начиная с конца внутренней
       out    SPL,tm              ;SRAM
       ldi    tm,high(RAMEND)
       out    SPH,tm
;Загрузка во внутреннее ОЗУ микроконтроллера, начиная с ячейки
;с адресом aPack, 45 байт массива: $20,$21,$22,…
       ldi    XL,low(aPack)
       ldi    XH,high(aPack)
       ldi    tm,$20
       ldi    cnt,45
StRAM:
       st     X +,tm
```



```
       inc    tm
       dec    cnt
       brne   StRAM
;Перепишем этот массив во внешнюю память на страницу № 2. Адрес
;страницы: AR2 AR1 AR0 = 010
       in     tm,PORTC
       andi   tm,$ff - ((1<<AR2) + (1<<AR1) + (1<<AR0))
;Очистка разрядов AR2,AR1,AR0.
;1111 1000 = $ff - ((1<<AR2) + (1<<AR1) + (1<<AR0))
       ori    tm,(1<<AR1)         ;Установка разряда AR1(адрес
                                  ;страницы 010 (страница № 2))
       out    PORTC,tm            ;Вывод в порт C содержимого tm
       clr    tm                  ;Очистка содержимого tm (tm = 0)
       sts    RamH,tm             ;Сохранение в ячейке RamH
                                  ;содержимого tm
       sts    RamL,tm
       rcall  St45bt              ;Вызов подпрограммы St45bt
;Скопируем массив из внешней памяти на странице № 2 (адрес страницы
;AR2 AR1 AR0 = 010) во внутреннее ОЗУ микроконтроллера
       clr    tm                  ;Очистка содержимого tm (tm = 0)
       sts    RamH,tm             ;Сохранение в ячейке RamH
                                  ;содержимого tm
       sts    RamL,tm
       rcall  DOutPrp

cycle: rjmp cycle
;Подпрограммы:
;=============
;Копирование массива из внутренней памяти во внешнюю память
St45bt:
       ldi    ZL,low(aPack)
       ldi    ZH,high(aPack)
       ldi    cnt,45
mSt45:
       rcall  SetAddr
       rcall  DataSt
       dec    cnt
       brne   mSt45
       ret
;==========
;Подпрограмма копирования 45 байт из внешней памяти во внутреннее ОЗУ
DOutPrp:
       ldi    ZL,low(aADC)
       ldi    ZH,high(aADC)
       ldi    cnt,45
DOut1:
       rcall  SetAddr
       rcall  DataLd
       dec    cnt
       brne   DOut1
       ret
```

```
;==========
;Подпрограмма установки адреса
SetAddr:
        lds     XL,RamL         ;Скопировать содержимое ячейки RamL в XL
        lds     XH,RamH
        out     PORTB,XL        ;Вывести в порт B содержимое XL
        nop                     ;Задержка 156 нс
        nop                     ;Задержка 156 нс
        sbi     PORTD,Buf1      ;Установить разряд Buf1 порта D
        nop
        nop
        cbi     PORTD,Buf1      ;Сбросить разряд Buf1 порта D
        nop
        nop
        out     PORTB,XH        ;Вывести в порт B содержимое XH
        nop
        nop
        sbi     PORTD,Buf2      ;Установить разряд Buf2 порта D
        nop
        nop
        cbi     PORTD,Buf2      ;Сбросить разряд Buf2 порта D
        adiw    XL,1            ;Увеличить содержимое пары XH:XL на 1
        sts     RamH,XH         ;Сохранить содержимое XH в ячейке RamH
        sts     RamL,XL
        ret                     ;Возврат из подпрограммы
;==========
;Подпрограмма копирования байта из внутреннего ОЗУ во внешнюю память
DataSt:
        ld      tm,Z +
        out     PORTB,tm
        nop
        nop
        cbi     PORTC,Wr
        nop
        nop
        sbi     PORTC,Wr
        ret
;===========
;Подпрограмма копирования байта из внешней памяти во внутреннее ОЗУ
DataLd:
        clr     tm              ;Очистка tm
        out     DDRB,tm         ;Контакты порта B в режиме входов
        cbi     PORTC,Rd        ;Сбросить разряд Rd порта C
        nop
        nop
        in      tm,PINB         ;Считать данные на линиях порта B в tm
        sbi     PORTC,Rd        ;Установить разряд Rd порта C
        st      Z +,tm          ;Сохранить содержимое tm в ячейке ОЗУ
        ser     tm              ;Установить tm
        out     DDRB,tm         ;Все контакты порта B в режиме выходов
        ret
```


3.7.1. Отладка программы

После ассемблирования (клавиша F7) при отсутствии ошибок приступим к отладке (клавиша F11). В появившемся окне Simulator options/Device выберем микроконтроллер ATmega8535.

Рассмотрим определение констант в программе. Ячейки внутренней памяти микроконтроллера с адресами *RamH* и *RamL* хранят старший и младший байты адреса внешней памяти. То есть в этих ячейках будет отражаться информация об адресе внешней памяти, которая выводится в буферы адреса (регистры DD3 и DD4 по схеме электрической).

Константа *AR0* равна константе *PC0*; *PC0* в свою очередь определена в файле, вставляемом в программу директивой *.include*.

Остальные константы, не определенные в тексте программы, также определяются в файле m8535def.inc.

Нажмем F11. Откроем Workspace/IO ATMega8535, в открывшемся окне выполним PortB/+, здесь же выполним PortC/+ и PortD/+ для контроля информации, выводимой на контакты портов B, C, D, а следовательно, и на шину данных нашей схемы (порт B), и на линии управления внешней памятью (соответствующие контакты портов C и D).

Замечание. Имена констант в программе соответствуют именам линий управления внешней памятью на электрической схеме.

Переносим курсор на команду блока загрузки внутренней SRAM массивом в 45 байт (*ldi XL,low(aPack)*) и нажимаем Ctrl + F10.

Произошла инициализация портов B, C и D, а также установлен начальный адрес стека.

Проверяем состояние линий портов в окне IO. Порт B инициализирован для вывода данных из микроконтроллера. Линии PC4 и PC3 порта C установлены (в электрической схеме на соответствующих линиях WR и RD будут ВЫСОКИЕ уровни). На линиях PD3 и PD4 порта D (соответствуют линиям BUF1, BUF2 в схеме) — НИЗКИЕ уровни. В этом же окне IO выполняем CPU/+ и проверяем состояние регистров указателя стека SPH и SPL.

Для имитации массива, получаемого в реальной программе в результате одной группы преобразований АЦП, в область памяти, начинающуюся адресом *aPack*, в программу введен цикл создания простого массива размером в 45 байт.

Вызовем окно Memory, выполнив View/New memory view, перенесем курсор на команду, следующую за циклом создания массива (*in tm,PINC*) и нажмем Ctrl + F10. Во внутренней памяти микроконтроллера создан мас-

сив, первый байт которого ($22) находится в ячейке с адресом *aPac*
(0x00B0).

Нажимаем F11 пять раз, проверяем состояние порта С в окне IO. Ли-
нии PC2, PC1, PC0 приобрели состояние 010, таким же будет состояни
трех старших линий адреса в схеме (AR2, AR1, AR0), определяющих номе
рабочей страницы внешней памяти.

Трижды нажимаем F11, очищаем старший и младший адреса внешне
памяти в ячейках RamH, RamL.

Указатель выполнения команд остановился на команде вызова под
программы переноса данных во внешнюю память. В окне Memory перей
дем в самый конец внутренней памяти, переместив скроллер окна вниз д
конца, выполним View/Processor, в открывшемся окне процессора заме
тим состояние счетчика команд (0x00001D — это номер команды *rcall*, ко
торая будет выполняться) и указателя стека (= 0x000025F).

Нажмем F11. Указатель выполнения команд перескочил на команду
меткой *St45bt:* (окно программы), в счетчике команд — номер команды
этой меткой. Во внутренней памяти по адресу 25F находится двухбайтны
номер команды, на которую надо вернуться после выполнения подпро
граммы (001E — именно эта команда следует за командой с номером 001D
и именно адрес 001E загрузится в счетчик команд после выполнения под
программы). Указатель стека сместился на две ячейки влево (25D), есл
внутри нашей подпрограммы будет подпрограмма следующего уровн
вложения. В ячейки 25D:25C запишется двухбайтный адрес возврата и
этой подпрограммы.

Устанавливаем курсор на команду с меткой *mSt45:*, нажимае
Ctrl + F10. Регистр Z содержит адрес начала массива во внутренней опе
ративной памяти микроконтроллера. Указатель выполнения команд на
ходится в начале цикла побайтного копирования данных из внутренне
оперативной памяти микроконтроллера во внешнюю память. Отличи
цикла от уже рассматривавшихся в этой главе циклов — в наличии дву
вызовов подпрограмм. Первый из них (*rcall SetAddr*) записывает младши
и старший байты адреса внешней памяти в два буфера (по электрическо
схеме — в регистры DD3 и DD4). Второй вызов подпрограммы
rcall DataSt извлекает из ячейки внутренней оперативной памяти микро
контроллера (внутреннего ОЗУ) байт данных и записывает его в ячейку
внешней памяти.

Наблюдая содержимое счетчика команд и указателя стека в окне Pro
cessor, а также изменения в окне Memory (последняя строка, начинающая
ся адресом 0x025F), нажимаем F11, указатель выполнения команд переме
щается на первую команду подпрограммы установки адреса внешне
памяти.

3.7.2. Подпрограмма установки адреса SetAddr:

1. Из ячеек RamH, RamL внутреннего ОЗУ в регистры XH, XL загружаются младший и старший байты адреса внешней памяти (две команды *lds*).
2. Младший байт адреса выводится на шину данных B0...B7 (смотрите схему) через порт B (команда *out PORTB,XL*).
3. После задержки, необходимой для завершения переходных процессов в схеме (две команды *nop*), на линии BUF1 устанавливается ВЫСОКИЙ уровень (команда *sbi PORTD,BUF1*), по которому в регистр DD4 записывается информация, установленная на шине данных.
4. После задержки (две команды *nop*) на линии BUF1 устанавливается НИЗКИЙ уровень (команда *cbi PORTD,BUF1*), запись в регистр завершена.
5. Старший байт адреса выводится на шину данных B0...B7 через порт B (команда *out PORTB,XH*).
6. После задержки на линии BUF2 устанавливается ВЫСОКИЙ уровень (команда *sbi PORTD,BUF2*), по которому в регистр DD3 записывается информация, установленная на шине данных.
7. После задержки (две команды *nop*) на линии BUF2 устанавливается НИЗКИЙ уровень (команда *cbi PORTD,BUF2*), завершая запись в регистр DD3.
8. Команда *adiw* увеличивает содержимое пары регистров XH:XL, теперь в них хранится адрес следующей ячейки внешней памяти, запись в нее будет произведена в следующем цикле установки адреса.

Проведите пошаговую отладку подпрограммы установки адреса, наблюдая за тем, какие данные выводятся в порты. Сопоставьте эти данные с состояниями, которые должны устанавливаться на соответствующих линиях электрической схемы. При выполнении команды выхода из подпрограммы (*ret*) наблюдайте за изменением состояний счетчика команд и указателя стека. Постарайтесь понять, где хранился адрес возврата из подпрограммы, на какой адрес возврата установлен указатель стека теперь.

> **Замечание.** В регистре ввода/вывода PORTC хранится то, что программа вывела в порт, в регистре ввода-вывода PINC — то, что присутствует на контактах порта (так, в PORTC можно вывести байт 11111111, контакты микроконтроллера PC0 и PC2 соединить с общим проводом, тогда состояние PORTC останется без изменений, состояние PINC станет равным 11111010). Для команд типа *out PORTC,tm* в окне IO информация появляется сначала в строке PORTC, а после выполнения следующей команды — в строке PINC.

Перед входом в цикл, начинающийся меткой *mSt45:*, в регистр Z был занесен адрес начала массива во внутреннем ОЗУ микроконтроллера.

3.7.3. Подпрограмма копирования байта из внутреннего ОЗУ DataSt

1. Первая команда подпрограммы (*ld tm,Z+*) копирует первый байт массива из ячейки, адрес которой хранится в регистре Z, в регистр tm, затем адрес, хранящийся в регистре Z, увеличивается на единицу. Теперь Z указывает на следующий элемент массива.
2. Элемент массива выводится на шину данных через порт В (команда *out PORTB,tm*).
3. После задержки (команды *nop*) на линии WR устанавливается НИЗКИЙ уровень командой *cbi PORTC,Wr*. Информация, установленная на шине данных, записывается в ячейку внешней памяти, адрес которой был установлен при выполнении подпрограммы *SetAddr*.
4. После задержки (команды *nop*) на линии WR устанавливается ВЫСОКИЙ уровень командой *sbi PORTC,Wr*. Копирование байта данных во внешнюю память завершено.
5. Выполняется возврат из подпрограммы.

Выполните один раз пошаговую отладку подпрограммы *DataSt*. Отладка цикла, в котором находится эта подпрограмма, не представляет интереса.

Отследить состояние внешней памяти с занесенным в нее массивом мы не сможем, так как наш вариант подключения внешней памяти к микроконтроллеру не предусмотрен стимулятором AVR Studio.

По этой же причине мы не сможем полностью провести отладку копирования массива из внешней памяти во внутреннее ОЗУ микроконтроллера. Однако можно воспользоваться приемами, которые предлагались в предшествующем примере: отразить массив в области внутреннего ОЗУ специально для отладки или после команды ввода в порт информации с шины данных вводить эти данные в порт вручную.

Воспользуемся вторым приемом при отладке подпрограммы копирования данных из внешней памяти во внутреннее ОЗУ микроконтроллера *DataLD*.

Переместите курсор на команду *rcall DoutPrp* и нажмите Ctrl + F10.

Первая пара команд подпрограммы загружает в регистр Z адрес новой области внутреннего ОЗУ микроконтроллера для массива, копируемого из внешней памяти. Далее выполняется команда загрузки счетчика циклов, затем цикл, начинающийся меткой *Dout1:*.

Цикл не отличается от ранее рассматривавшихся циклов. Подпрограмма *SetAddr* рассматривалась ранее. Поэтому, доходя до команды вызова этой подпрограммы, нажимайте клавишу F10, чтобы выполнить подпрограмму за один шаг.

В результате выполнения подпрограммы *SetAddr* адрес сформирован на линиях адреса внешней памяти.

3.7.4. Подпрограмма копирования данных из внешней памяти во внутреннее ОЗУ DataLd

1. В порт DDRB выводится нулевое значение, переводящее порт В в режим приема информации (все контакты порта В работают как входы).
2. На линии RD устанавливается НИЗКИЙ уровень (команда *cbi PORTC,Rd*). Поэтому из ячейки внешней памяти, адрес которой сформирован при выполнении подпрограммы *SetAddr*, на шину данных выводится информационный байт.
3. После задержки (команды *nop*) через порт В с шины данных в регистр микроконтроллера tm считывается байт информации (*in tm,PINB*).
4. На линии RD устанавливается ВЫСОКИЙ уровень (*sbi PORTC,Rd*), а контакты микросхемы памяти IO0...IO7 переводятся в высокоимпедансное состояние.
5. Данные из регистра tm переносятся в ячейку внутреннего ОЗУ микроконтроллера, адрес которой хранится в регистре Z. После этого содержимое Z увеличивается на единицу. Теперь регистр Z указывает на следующую ячейку ОЗУ, в которую будет произведена запись в следующем цикле (команда *st Z+,tm*).
6. Порт В переводится в режим вывода данных (все контакты порта — выходы).
7. Возврат из подпрограммы.

Удобно перевести порт В из режима передачи данных (выход) в режим приема (вход) только на время выполнения подпрограммы *DataLd*, так как в других подпрограммах порт В должен всегда находиться в режиме передачи.

При отладке подпрограммы *DataLd* перед командой *in tm,PINB* для имитации ввода данных установите какие-нибудь флажки в строке PINB для окна Workspace/IO/Port B. Тогда соответствующие данные будут занесены в ячейку внутреннего ОЗУ микроконтроллера.

Замечание. Приведенные программы подключения внешней памяти адаптированы для лучшего понимания работы устройства, программы и отладчика. Однако считать их завершенными, а файлы hex, полученные при их ассемблировании, загружать в микроконтроллер не следует. В реальной рабочей программе сторожевой таймер (Watch dog timer) должен периодически программно сбрасываться командами *wdr*, размещенными по всей программе, иначе программа будет регулярно сбрасываться этим таймером, возвращаясь на метку *RESET*. В приведенных примерах программ не размещены векторы прерываний, являющиеся неотъемлемой частью практически любой программы. Перечисленные темы в этой главе не рассматриваются.

Глава 4. Устройство динамической индикации на 7-сегментных индикаторах

4.1. Принцип динамической индикации

Светодиодные 7-сегментные индикаторы выпускаются как с общим анодом, так и с общим катодом. На **Рис. 22** представлена электрическая схема 7-сегментного (8-й сегмент — десятичная запятая, выполненная на светодиоде H) индикатора, слева от нее — физическое размещение светодиодов. Если на катод подать отрицательное напряжение, а на аноды B, C, E, F и G — положительное напряжение определенного уровня, через одноименные светодиоды будет протекать ток, образуя светящуюся букву «H».

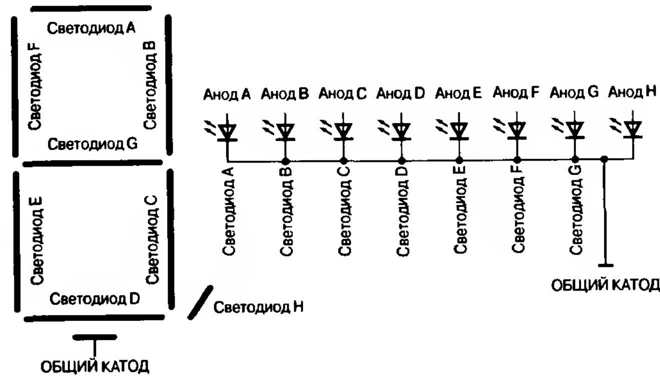


Рис. 22. Семисегментный светодиодный индикатор

Для отображения N символов надо использовать N таких индикатор[о]в. Можно соединить их катоды, а положительное напряжение подавать н[а] каждый анод отдельно (N индикаторов по 7 анодов в каждом). То есть индикаторам придется вести 7N линий управления и линию общего д[ля] всех индикаторов катода.

Более экономичный способ представлен на **Рис. 23**.

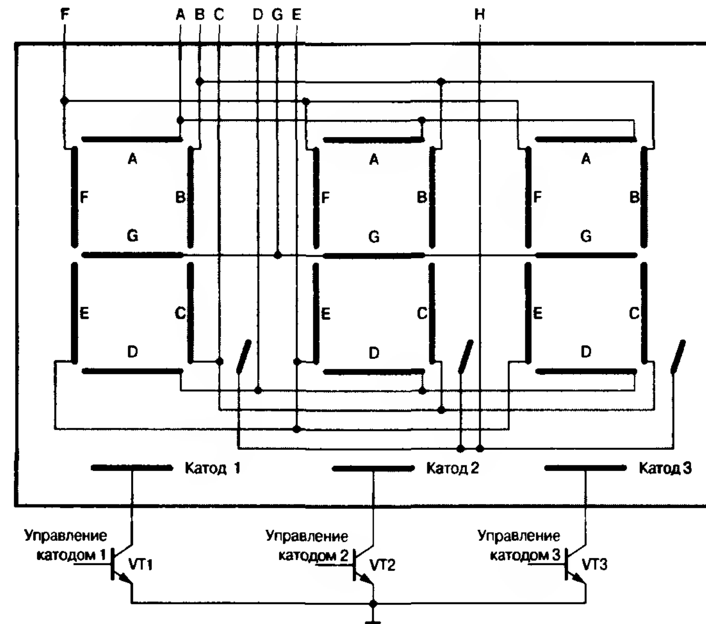


Рис. 23. Схема динамической индикации на 3 символа

Одноименные аноды трех индикаторов соединены между собой в общие аноды A, B, C, D, E, F и G, катоды коммутируются управляющими сигналами через транзисторные ключи на общий провод схемы.

Замечание. Многоразрядный динамический индикатор с раздельными катодами набирается из отдельных индикаторов с общими катодами.
На **Рис. 23** три индикатора обведены штриховой линией — такие трехразрядные индикаторы выпускаются в одном корпусе, их общие аноды объединены внутри корпуса (пример: TOT-3361AH-1N).

Опишем процедуры, которые надо выполнить для того, чтобы высветить цифры 3, 1 и 7 соответственно на 1-м, 2-м и 3-м индикаторах.

Для высвечивания цифры 3 подать положительное напряжение на общие аноды A, D, E, F и G, а катод 1 соединить с общим проводом, открыв ключ на транзисторе VT1. Аноды B и C отключаются ключами от источника напряжения (ключи на схеме не показаны). Ключи на транзисторах VT2 и VT3 надо запереть для отключения катодов 2 и 3 от общего провода, чтобы цифра 3 не светилась на втором и третьем индикаторах.

Для индикации цифры 1 на аноды E и F надо подать положительное напряжение. Остальные аноды следует отключить, ключ VT2 открыть, соединив катод 2 с общим проводом. Транзисторы VT1 и VT3 надо закрыть, чтобы цифра 1 не светилась на индикаторах 1 и 3.

Для индикации цифры 7 на аноды A, E и F следует подать положительное напряжение. Остальные аноды надо отключить, ключ VT3 открыть, соединив катод 3 с общим проводом. Транзисторы VT1 и VT2 надо закрыть, чтобы цифра 7 не светилась на индикаторах 1 и 2.

Если перечисленные действия циклически повторять, то будут светиться друг за другом цифра 3 на 1-м индикаторе, затем она погаснет и засветится цифра 1 на 2-м индикаторе, а вслед за ней цифра 7 на 3-м индикаторе.

Человек не замечает мелькания частотой выше 25 Гц. Поэтому, если в каждом цикле каждый индикатор будет светиться в течение времени, меньшего 1/25N секунды, где N = 3 (число индикаторов), мы будем воспринимать индикацию числа 317 так, как будто все три цифры светятся одновременно.

Замечание. Поскольку теперь каждый разряд светится лишь треть цикла, для сохранения яркости свечения надо увеличить токи, протекающие через сегменты. Однако не следует увеличивать их в три раза для 3-разрядного индикатора, так как в динамическом режиме зависимость яркости свечения светодиода от протекающего через него тока нелинейная.

4.2. Восьмиразрядное устройство отображения цифровой информации

4.2.1. Схема управления восьмиразрядным индикатором

На **Рис. 24** приведен фрагмент схемы контроллера станка по выпеканию печенья, обеспечивающего управление 8-разрядным индикатором.

Управление индикатором осуществляется через порты C и D микроконтроллера. Линия PB6 порта B используется для организации управления с помощью кнопочной клавиатуры. Порт A работает в режиме АЦП.

В качестве 8-разрядного индикатора использован узел, собранный из набора, предназначенного для использования в АОН (автоматический определитель номера для телефонов) и продающегося на радиорынках.

В набор входят три уже рассмотренных выше трехразрядных индикатора TOT-3361AH-1N и плата. Поскольку из такого набора получается 9-разрядная индикаторная линейка, разряд, находящийся посередине, в кон-

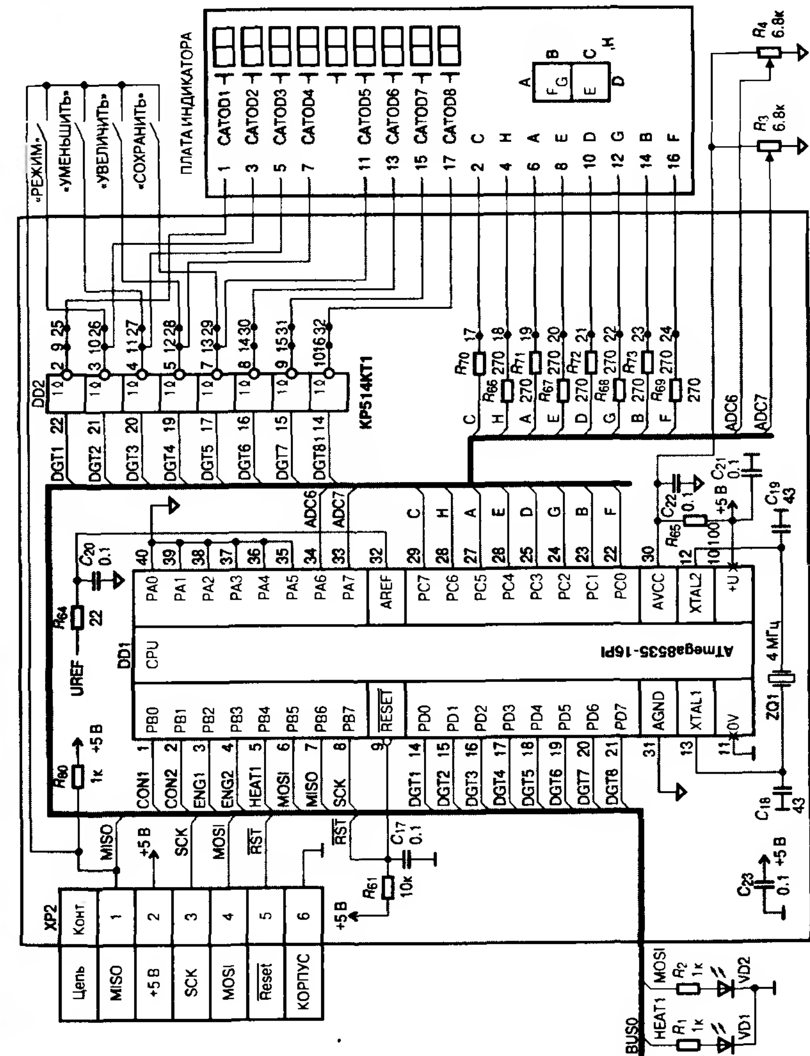


Рис. 24. Схема управления 8-разрядным светодиодным индикатором

троллере не используется, что позволяет отображать одновременно два 4-разрядных числа, разделенных неиспользуемым разрядом.

Справа на схеме изображена плата индикатора, внутри которой показано физическое расположение сегментов одного разряда индикатора, над ним — соответствие номеров катодов физическому расположению разрядов всей индикаторной линейки. На плате имеется маркировка первого контакта.

> **Замечание.** В данном устройстве сегменты Н индикаторов (запятые после цифр) не используются, но их общий анод подключается к микроконтроллеру. Для управления индикацией сегментов Н достаточно лишь изменить программу.
> Не стоит экономить на индикаторах и использовать лежащие в вашем столе АЛС333 и им подобные: их свечение в такой схеме будет крайне слабым.
> Индикаторы, использованные в схеме, светятся достаточно ярко и при непосредственном подключении их катодов к микроконтроллеру без элементов DD2.1...DD2.9. Последние обеспечивают дополнительную зашиту выходов микроконтроллера от перегрузки.

Резисторы R66...R73 ограничивают ток через сегменты (светодиоды) индикаторной линейки. Справа от резисторов и элементов DD2.1...DD2.9 расположены контакты платы контроллера для распайки проводов (нумерованные жирные точки).

Правее каждого из элементов DD2.1...DD2.9 находится по паре контактов, к одному из них подсоединяется соответствующий катод индикатора, к другому — один из пары контактов кнопки управления. Можно подключить до восьми кнопок управления; вторые контакты всех кнопок соединяются между собой и подключаются к контакту 33 платы контроллера (на **Рис. 24** он расположен рядом с разъемом программирования микроконтроллера XP2).

Резистор R80 обеспечивает ВЫСОКИЙ уровень на контакте PB6 микроконтроллера. Попеременно коммутируя катоды индикатора на общий провод, микроконтроллер проверяет состояние контакта PB6. Если, например, нажата кнопка, подключенная к элементу DD2.1, то через замкнутые контакты кнопки и через открытый ключ DD2.1 контакт микроконтроллера PB6 окажется соединенным с общим проводом и на контакте PB6 будет присутствовать НИЗКИЙ уровень. Обнаружение и обработка НИЗКОГО уровня на контакте PB6 определяется программой микроконтроллера.

> **Замечание.** Используемые кнопки с нормально разомкнутыми контактами не влияют на работу программатора микроконтроллера, если, конечно, в процессе загрузки программы не нажимать их.

Микросхема DD2 — КР514КТ1 включает в себя девять элементов с открытыми коллекторами. Функционально каждая ячейка представляет собой ключ, через который удобно коммутировать мощную нагрузку на общий провод схемы. Аналогичные функции выполняют транзисторы VT1...VT3 в схеме динамической индикации на три разряда, рассматривавшейся ранее.

Назначение остальных элементов схемы:

- R64, C20 и R65, C22 — фильтры опорного напряжения и напряжения питания для АЦП;
- R61, C17 — цепочка для обеспечения аппаратного сброса микроконтроллера при включении питания;
- C21, C23 — конденсаторы, служащие для устранения помех по линиям питания.

Для обеспечения лучшей помехозащищенности при аналого-цифровом преобразовании в схеме разделены аналоговый и цифровой общие провода. Аналоговый общий провод обозначен на схеме треугольником.

Граница схемы контроллера, так же как и граница платы индикатора, обозначена пунктирной линией.

Элементы, не входящие ни в контроллер, ни в плату индикатора, предназначены для управления работой схемы (кнопки), для имитации работы термодатчиков (резисторы R3, R4) и мощных оптотиристоров (R1, VD1, R2, VD2). Последние управляют работой печек в схеме контроллера, управляющего работой станка по выпеканию печенья.

4.2.2. Программа организации бегущей строки

Для работы программы с приведенной выше схемой подключения кнопок, резисторов $R_1...R_4$ и светодиодов VD_1, VD_2 не требуется.

Как создается проект в AVR Studio, вы уже знаете. Поэтому в дальнейшем приводится листинг программы с описанием тех моментов в программе, на которые стоит обратить внимание или которые не освещались ранее.

```
;Светодиодная матрица 7 сегментов, динамическая индикация
;справа налево по кольцу движутся символы: 7,6,5,4,3,2,1
;==================================================
.include "c:\avr\def\m8535def.inc"
.def   t1 = r16
.def   t2 = r17
.def   catod = r18
.def   cod = r19
.def   dy1 = r20
.def   dy2 = r21
.def   dy3 = r22
```

```
.def   cnt = r23
.def   tmp = r24
.equ   RAM = $060
.equ   cycles = 100
.CSEG
.org 0
       rjmp RESET                   ;Reset Handler
       nop                          ;rjmp EXT_INT0 ;IRQ0 Handler
       nop                          ;rjmp EXT_INT1 ;IRQ1 Handler
       nop                          ;rjmp TIM2_COMP ;Timer2 Compare Handler
       nop                          ;rjmp TIM2_OVF ;Timer2 Overflow Handler
       nop                          ;rjmp TIM1_CAPT ;Timer1 Capture Handler
       nop                          ;rjmp TIM1_COMPA ;Timer1 CompareA Handler
       nop                          ;rjmp TIM1_COMPB ;Timer1 CompareB Handler
       nop                          ;rjmp TIM1_OVF ;Timer1 Overflow Handler
       rjmp TIM0_OVF                ;Timer0 Overflow Handler
       nop                          ;rjmp SPI_STC ;SPI Transfer Complete
                                    ;Handler
       nop                          ;rjmp UART_RXC ;UART RX Complete Handler
       nop                          ;rjmp UART_DRE ;UDR Empty Handler
       nop                          ;rjmp UART_TXC ;UART TX Complete Handler
       nop                          ;ADC Conversion Complete Interrupt
                                    ;Handler
       nop                          ;rjmp EE_RDY ;EEPROM Ready Handler
       nop                          ;rjmp ANA_COMP ;Analog Comparator Handler
RESET:
       ldi    t1,15                 ;Сторожевой таймер:без wdr сброс
       out    WDTCR,t1              ;через 2 с
       wdr
       ldi    t1,HIGH(RAMEND)       ;Инициализация стека
       out    SPH,t1
       ldi    t1,LOW(RAMEND)
       out    SPL,t1
       ldi    t1,1<<TOV0
       out    TIFR,t1
       clr    t1
       out    TCCR0,t1
       ldi    t1,1<<TOIE0
       out    TIMSK,t1
       ldi    t1,1<<TOV0
       out    TIFR,t1
       ldi    t1,$ff - 77           ;Таймер T0 отсчитывает 77 циклов
       out    TCNT0,t1              ;с предварительным делением
       ldi    t1,3                  ;тактовой частоты микроконтроллера
       out    TCCR0,t1              ;на 64 (Ck/64 prescaling)
       ldi    t1,$ff
       out    PORTB,t1
       out    PORTD,t1
       out    DDRD,t1               ;Порты C и D — выходы
       out    DDRC,t1
       clr    t1                    ;Линии порта C сбрасываются, чтобы не
```



```
       out    PORTC,t1              ;вспыхивали индикаторы
                                    ;при программном переходе на RESET
       ldi    catod,1               ;catod хранит номер светящегося
                                    ;разряда индикатора
       sei
;Блок загрузки символов строки
       ldi    YL,low(RAM)
       ldi    YH,high(RAM)
       clr    cod
       st     Y +,cod
       st     Y +,cod
       st     Y +,cod
       st     Y +,cod
       st     Y +,cod
       st     Y +,cod
       st     Y +,cod
       st     Y +,cod
       ldi    cod,$a2               ;7
       st     Y +,cod
       ldi    cod,$bd               ;6
       st     Y +,cod
       ldi    cod,$ad               ;5
       st     Y +,cod
       ldi    cod,$87               ;4
       st     Y +,cod
       ldi    cod,$ae               ;3
       st     Y +,cod
       ldi    cod,$3e               ;2
       st     Y +,cod
       ldi    cod,$82               ;1
       st     Y +,cod
       ldi    cod,$00               ;Стирание разряда за
                                    ;последним символом
       st     Y +,cod
;Конец блока загрузки символов строки
       ldi    dy2,15                ;Число сдвигов символов, после
                                    ;которого информация в строке
                                    ;обновляется
       ldi    dy1,cycles
start:
       ldi    t1,(1<<SE) + (1<<SM2)+(0<<SM1) + (0<<SM0) ;Микроконтроллер
       out    MCUCR,t1              ;в Idle mode при переходе
       sleep                        ;в режим sleep
       dec    dy1                   ;Индикация строки без смещения
                                    ;символов в течение cycle переходов
       brne   start                 ;в режим sleep
       ldi    dy1,cycles
;Блок смещения строки на одну цифру влево
       ldi    YL,low(RAM)
       ldi    YH,high(RAM)
       ldi    ZL,low(RAM + 1)
```

```
        ldi     ZH,high(RAM + 1)
        ldi     cnt,15
more:
        ld      cod,Z +
        st      Y +,cod
        dec     cnt
        brne    more
;Конец блока смещения строки на одну цифру влево
        dec     dy2
        brne    m2                      ;Если вся строка прошла индикатор,
        rjmp    Reset                   ;новая загрузка символов в память
m2:     rjmp    start
;Обработка прерывания таймера T0
TIM0_OVF:
        wdr
        out     PORTD,catod             ;Индицировать выбранный разряд
        out     PORTC,cod               ;Через порт С подать на аноды
                                        ;напряжения для индикации разряда
                                        ;catod
        lsl     catod                   ;Следующий разряд, если его номер = 0
        tst     catod
        brne    m1
        ldi     catod,1                 ;Установить его равным 1
;Блок выбора кода в зависимости от номера разряда
m1:     ldi     YL,low(RAM)
        ldi     YH,high(RAM)
        sbrc    catod,1
        adiw    YL,1
        sbrc    catod,2                 ;Определение адреса (Y) кода,
        adiw    YL,2                    ;для следующего выбранного разряда
        sbrc    catod,3
        adiw    YL,3
        sbrc    catod,4
        adiw    YL,4
        sbrc    catod,5
        adiw    YL,5
        sbrc    catod,6
        adiw    YL,6
        sbrc    catod,7
        adiw    YL,7
        ld      cod,Y
;Конец блока выбора кода в зависимости от номера разряда ldi
        t1,255 - 77                     ;Перезапуск таймера
        out     TCNT0,t1
        reti
```

4.2.3. Описание программы

Кроме прерывания *RESET* в программе используется лишь одно прерывание таймера T0, обработчик которого, начинающийся меткой *TIM0_OVF:*, находится в конце программы.

После подачи питания или при переходе на метку *RESET:* уже знакомыми командами инициализируются сторожевой таймер, стек, таймер T0 и порты ввода/вывода. Здесь стоит обратить внимание только на выбор интервала времени, через который происходит прерывание таймера T0.

Как отмечалось ранее, для того чтобы человек воспринимал динамическое включение индикатора статичным, частота включения должна быть более 25 Гц.

Таймер T0 генерирует прерывания, частоту которых можно определить по формуле $F_{таймера} = F_{такт}/C_{таймера} \times Prescaling$, где $F_{такт}$ — тактовая частота микроконтроллера (4000 кГц) определяется частотой подключенного к нему кварцевого резонатора, Prescaling — коэффициент предварительного деления тактовой частоты (равен 64 путем записи числа три в TCCR0), $C_{таймера} = 77$ (это число тиков, отсчитываемое таймером до переполнения, заносится в TCNT0 как $256 - C_{таймера}$.

$F_{таймера} \approx 812$ Гц при условии перезагрузки в таймер T0 константы $C_{таймера}$ сразу после его переполнения. Временной интервал от начала счета до переполнения составляет 1/812 с, или 1.2 мс.

Так как индицируется 7 цифр, то вся строка будет повторяться с частотой около 116 Гц, то есть частота выбрана почти с 5-кратным запасом по отношению к минимальной частоте мелькания 25 Гц.

Далее в программе после инициализации переменных *catod* и *cnt* и разрешения прерываний находится блок загрузки символов строки.

Для того чтобы определить код, который необходимо выводить в порт С для индикации символа, снова обратимся к схеме управления 8-разрядным светодиодным индикатором (**Рис. 24**).

Справа от резисторов $R_{66}...R_{73}$ изображен один разряд индикатора, его сегменты, соответствующие анодам светодиодов, именуются латинскими буквами от А до Н. Этими же буквами именуются и цепи схемы, соединяющие аноды светодиодов через резисторы $R_{66}...R_{73}$ (ограничители тока) с линиями PC0...PC7 порта С. Например, анод С соединяется с линией PC7.

A, B, C — ВЫСОКИЕ уровни
D, E, F, G, H — НИЗКИЕ уровни

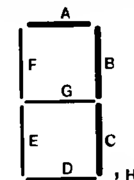


Рис. 25. Определение 7-сегментного кода для цифры 7

В первой строке **Табл. 4** представлены сегменты индикатора, ниже — линии порта С, к которым подключаются эти сегменты.

В ячейках на пересечении сегмента (линии порта С) и индицируемой цифры (символа) указаны логические уровни, которые должны присутствовать на данном сегменте для индикации выбранной цифры (1 — ВЫСОКИЙ уровень напряжения, 0 — НИЗКИЙ уровень).

Удобно нарисовать единичный индикатор и проставить буквенные наименования сегментов, как показано на **Рис. 25**. Для того чтобы отобра-

зить, например, цифру 7, надо вызвать свечение сегментов A, B и C, подав на них ВЫСОКИЕ уровни напряжения, на остальные сегменты подаются НИЗКИЕ уровни.

Таблица 4

Сегменты	C	H	A	E	D	G	B	F	Код
Линии порта С	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0	
Цифра 0	1	0	1	1	1	0	1	1	$bb
Цифра 1	1	0	0	0	0	0	1	0	$82
Цифра 2	0	0	1	1	1	1	1	0	$3e
Цифра 3	1	0	1	0	1	1	1	0	$ae
Цифра 4	1	0	0	0	0	1	1	1	$87
Цифра 5	1	0	1	0	1	1	0	1	$ad
Цифра 6	1	0	1	1	1	1	0	1	$bd
Цифра 7	1	0	1	0	0	0	1	0	$a2
Цифра 8	1	0	1	1	1	1	1	1	$bf
Цифра 9	1	0	1	0	1	1	1	1	$af
Символ «,»	0	1	0	0	0	0	0	0	$40
Нет свечения	0	0	0	0	0	0	0	0	$00

В таблице колонки сегментов упорядочены не по их наименованию, а по номеру подключаемой к ним линии порта С, что позволяет легко преобразовать комбинацию уровней для выбранной цифры в шестнадцатеричный код, отображаемый в последней колонке. Так, для цифры 7 комбинация 10100010 соответствует шестнадцатеричному коду $a2.

Замечание. Вы можете сами продолжить таблицу для других символов, которые понадобится отобразить на индикаторе.
Попробуйте сделать это для пяти символов, необходимых для отображения слова «ПАУЗА».
Попробуйте также определить коды шестнадцатеричных символов в диапазоне от A до F.

Вернемся к программе: в блоке загрузки символов строки в 15 ячеек оперативной памяти микроконтроллера заносятся сначала 8 значений $00, соответствующих выключению индикаторов, а затем коды цифр 7...1, составляющих нашу строку.

Далее в переменную *dy2* загружается сумма количества используемых индикаторов и числа символов в строке. Нам надо, чтобы первоначально индикатор не светился. Представьте, что перед входом в 8-разрядный индикатор справа стоит 7-разрядное число. Для того чтобы последний разряд числа попал в индикатор, число надо сдвинуть влево 7 раз, а для того чтобы он прошел индикатор, требуется еще 8 сдвигов. Только после этого вся строка «выйдет» из индикатора справа. После того как все символы строки пройдут по индикатору, в память могут быть загружены коды символов новой строки, но в нашей программе будет постоянно повторяться загрузка одной строки.

В переменную *dy1* загружается число циклов *cycle*, в течение которых строка будет стоять на месте. Например, цифра 7 индицируется в 5-м разряде, цифра 6 — в 6-м, цифра 5 — в 7-м, цифра 4 — в 8-м, цифры 3, 2 и 1 еще не «вошли» в индикатор. Разряды индикатора пронумерованы с 1-го (старшего) по 8-й (младший).

Число *cycle* определено равным 100. Значит, в течение 100 циклов будут включаться сегменты 5-го разряда индикатора, высвечивая цифру 7, затем цифра 7 погаснет и высветится цифра 6 в 6-м разряде, потом цифра 5 в 7-м разряде. После высвечивания цифры 4 в 8-м разряде снова будет повторяться весь цикл, начиная с цифры 7.

И только после ста таких циклов цифра 7 переместится в 4-й разряд, а за ней и вся цепочка цифр переместится на один разряд вправо, после чего снова в течение 100 циклов строка будет оставаться неподвижной.

Как было определено ранее, частота генерации прерываний таймером T0 ($F_{таймера}$) составляет 812 Гц, 100 циклов этой частоты составят 8 Гц, следовательно, время, в течение которого строка остается неподвижной, равно 1/8 секунды, поэтому глаз будет воспринимать движение строки как непрерывное.

Меткой *start:* начинается бесконечный цикл программы. Первые две команды цикла разрешают переход микроконтроллера в режим Idle при выполнении команды *sleep*. Установка разряда SE регистра MCUCR разрешает переход в режим Sleep (операция 1<<SE).

Сброс разрядов SM2, SM1, SM0 операцией (0<<SM2) + (0<<SM1) + (0<<SM0) определяет тип режима Sleep как Idle mode.

Во избежание сбоев желательно, чтобы команда *sleep* следовала сразу после записи информации, определяющей тип режима Sleep в регистр MCUCR.

Если переход в режим Sleep разрешен (разряд SE регистра MCUCR установлен), по команде *sleep* микроконтроллер переходит в режим ожидания, определяемый разрядами SM0, SM1, SM2 регистра MCUCR. То есть микроконтроллер останавливается и не переходит к выполнению следующей команды до тех пор, пока не произойдет прерывание, способное вывести его из этого режима.

Из режима Idle микроконтроллер AT90S8535 может быть выведен прерываниями SPI, UART, аналогового компаратора, АЦП, внешними

прерываниями INT0 и INT1, сторожевого таймера, а также таймеров 0, 1 и 2.

В нашей программе разрешено прерывание таймера T0, поэтому после выполнения команды *sleep* микроконтроллер останавливается, продолжает свою работу только таймер T0, который, как было показано выше, переполнится через 1.2 мс после запуска.

Итак, после разрешения счета таймера T0, инициализированного командами *ldi t1,3* и *out TCCR0,t1*, таймер T0 начал отсчет интервала времени в 1.2 мс (при выполнении команды *sleep* таймер T0 остановился в ожидании окончания отсчета этого интервала). По окончании отсчета произойдет переполнение таймера T0. Следовательно, произойдет прерывание переполнения таймера T0.

Значит, после команды *sleep* через 1.2 мс произойдет переход на вектор прерывания, содержащий команду *rjmp TIM_OVF0*. Этот переход вызовет выполнение подпрограммы обработчика прерывания, начинающейся меткой *TIM_OVF0:*. И только после этого будет выполнена команда, следующая в программе за командой *sleep*.

Замечание. Точное время перехода на подпрограмму обработчика прерывания будет несколько меньше, так как из 1.2 мс надо вычесть время, ушедшее на команды, выполнявшиеся от начала счета таймера до команды *sleep* включительно.
В данном случае это время гораздо меньше интервала счета таймера, однако, если интервал соизмерим со временем выполнения команд, приходится считать и время выполнения команд.
Программист должен убедиться, что время выполнения команд не может оказаться больше интервала счета таймера, иначе результат работы программы будет непредсказуемым.

В подпрограмме обработки прерывания TIM_OVF0 сбрасывается сторожевой таймер, в порт D выводится переменная *catod*. В **Табл. 5** показана связь между состоянием линий порта D и свечением разряда индикатора.

Разряд индикатора может светиться, когда на его катоде присутствует низкий уровень напряжения. Если на входе одного из элементов DD2 присутствует ВЫСОКИЙ уровень напряжения (**Рис. 24**), то подключенный к его выходу катод индикатора соединяется с общим проводом, разряд индикатора будет светиться при условии подачи ВЫСОКОГО уровня хотя бы на один общий анод индикатора.

Поэтому при установке, например, на линии PD0 ВЫСОКОГО уровня вызывается свечение первого (старшего) разряда индикатора.

Приведенные выше в таблице значения, которые должны выводиться в порт D для включения разрядов 1...8, соответствуют значениям переменной *catod* в программе.

Таблица 5

Светится:	PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0
Разряд 1	0	0	0	0	0	0	0	1
Разряд 2	0	0	0	0	0	0	1	0
Разряд 3	0	0	0	0	0	1	0	0
Разряд 4	0	0	0	0	1	0	0	0
Разряд 5	0	0	0	1	0	0	0	0
Разряд 6	0	0	1	0	0	0	0	0
Разряд 7	0	1	0	0	0	0	0	0
Разряд 8	1	0	0	0	0	0	0	0

В порт D выведена переменная *catod*, теперь в порт C выводится значение переменной *cod*, а значит происходит свечение соответствующей коду цифры в определенном переменной *catod* разряде.

Далее командой *lsl catod* единичка в переменной *catod* сдвигается влево, подготавливая свечение следующего разряда при следующей обработке прерывания переполнения таймера T0.

Следующие команды (*tst catod, brne m1, ldi catod,1, m1:*) выполняют проверку, не произошел ли в результате команды *lsl* сдвиг за пределы разрядной сетки (после состояния 10000000 команда *lsl* вызовет очистку регистра). Если регистр очищен, в него снова заносится единица (00000001).

В следующем далее блоке выбора кода в зависимости от номера разряда определяется код следующей цифры, который будет выводиться в порт C при следующей обработке прерывания переполнения таймера T0.

Например, если обработчик прерывания таймера T0 вызвал свечение первого разряда, то далее ведется подготовка к свечению второго разряда индикатора и в переменной *catod* для этого уже установлен второй разряд. Поэтому команда (*sbrc catod,2* — пропустить следующую команду при сброшенном разряде 2 в регистре *catod*) не выполняется, а значит, следующая за ней команда *adiw YL,2* не пропускается, и в регистре Y теперь находится адрес кода второй цифры строки.

Командой *ld cod,Y* в переменную *cod* загружается код цифры, которая будет индицироваться при следующем вызове обработчика прерывания таймера T0.

Следующая пара команд вновь заносит в счетчик таймера T0 константу, с этого момента до индикации следующей цифры вновь пройдет 1.2 мс.

Теперь произойдет возврат из обработчика прерывания к команде, следующей за командой *sleep*. Эта команда (*dec dy1*) уменьшает число циклов, оставшихся до сдвига строки.

Пока это число не уменьшилось до нуля, будет циклически повторяться индикация неподвижной строки (возврат к метке *start:*). В противном случае в переменную *dy1* вновь заносится число циклов и происходит выполнение блока смещения строки на одну цифру.

Фактически коды цифр перемешаются из одной ячейки памяти в другую: код 2-й цифры — в первую ячейку, код 3-й — во вторую и так далее, в последнюю из 15 ячеек заносится значение 0, чтобы после прохождения строки разряды оставались погашенными. Организация такого перемещения кодов цифр очевидна.

Далее выполняется уменьшение содержимого переменной *dy2* и проверка, не равна ли она нуля. Равенство переменной *dy2* нулю, свидетельствующее о том, что прохождение всей строки через индикатор завершилось, вызывает переход к метке *Reset:*, программа повторяется с самого начала.

> **Замечание.** Строка может занимать почти всю оперативную память, в том числе и внешнюю, если она подключена к микроконтроллеру, или периодически загружаться, например, через COM-порт.

Достигнув команды *sleep*, несколько раз нажмите F11, при этом следите за изменением состояния счетчика циклов, команда будет выполняться, пока не произойдет прерывание переполнения таймера T0. Установите курсор на команде *rjmp TIM0_OVF* (вектор прерывания переполнения таймера T0) и нажмите CTRL + F10, обратите внимание на изменение состояния счетчика циклов.

4.3. Устройство управления двумя печами

4.3.1. Работа устройства

Для работы программы со схемой, приведенной на **Рис. 24,** обязательно подключение резисторов R1...R4, светодиодов VD1, VD2 и кнопок.

Несколько слов о функциях устройства. Это часть контроллера станка по выпеканию печенья. В рабочем режиме дозированный объем сырого зерна засыпается в форму, к форме снизу и сверху подводятся два нагревателя (верхняя печь и нижняя печь — далее печь В и печь Н соответственно). Через определенный интервал времени печи отводятся от формы, готовое печенье выталкивается в контейнер.

Обе печи имеют независимые нагревательные спирали и термопары, регулировка их нагрева производится также независимо.

Из-за громоздкости программы приведена только та ее часть, которая управляет работой нагревателей печей.

Резисторы R_3 и R_4 имитируют работу термодатчиков, установленных на станке непосредственно рядом с нагревателями. Поворачивая вал одного из

резисторов, вы изменяете напряжение, поступающее на соответствующий вход АЦП (цепи ADC6 и ADC7), имитируя этим изменение температуры нагревателя.

Пропорциональные температуре нагревателя результаты аналого-цифрового преобразования накапливаются и усредняются на заданном интервале времени, а затем усредненные значения отображаются на индикаторе в виде двух 4-разрядных десятичных чисел, разделенных неиспользуемым разрядом индикатора. Накопление результатов с их усреднением обеспечивает устранение постоянных колебаний показаний в последних разрядах индицируемых чисел, мешающих их считыванию.

> **Замечание.** Употребляющиеся далее термины T°B и T°H соответствуют усредненным результатам 64 циклов аналого-цифровых преобразований сигналов, поступающих по линиям ADC6 и ADC7 соответственно. Для того чтобы T°B и T°H действительно соответствовали значениям температур, до которых нагрелись печь В и печь Н, производится настройка и калибровка не показанной на схеме аналоговой части, усиливающей и нормализующей сигналы термопар В и Н, установленных на печах.

Значение T°B постоянно сравнивается с заданным значением температуры нагрева B°C печи В. Пока печь В не нагрелась до температуры B°C + *cTAdd*, на ее спираль подается напряжение, после нагрева спираль отключается, включение произойдет только после того, как печь В остынет до температуры B°C – *cTSub*. Это же справедливо для печи Н, только температура ее нагрева сравнивается со значениями H°C + *cTAdd* и H°C – *cTSub*.

Константы *cTAdd* и *cTSub* определяются при пробной работе станка и записываются в рабочую программу.

Значения B°C и H°C оператор может корректировать, пользуясь кнопками.

Наименования параметров B°C и H°C обусловлены тем, что их можно отобразить на семисегментном индикаторе, отобразить же, например, символ T невозможно.

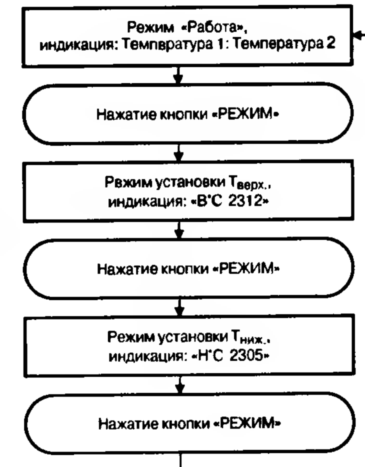


Рис. 26. Порядок изменения режимов при нажатии кнопки «РЕЖИМ»

Нажатие кнопки «РЕЖИМ» изменяет режим работы микроконтроллера и показания индикатора в порядке, представленном на **Рис. 26**:

При подаче питания микроконтроллер переходит в режим «Работа» с индикацией температуры нагрева обеих печей.

Для изменения значений В°С и Н°С следует вызвать соответствующий режим и кнопками «УВЕЛИЧИТЬ» или «УМЕНЬШИТЬ» добиться требуемых значений устанавливаемого параметра. Для записи измененного параметра в EEPROM микроконтроллера (электрически стираемое ПЗУ) надо нажать кнопку «СОХРАНИТЬ», если же после изменения параметра нажать кнопку «РЕЖИМ», внесенные изменения не сохранятся.

Замечание. EEPROM сохраняет данные и при отсутствии питания микроконтроллера.
Числа 2312 и 2305 в режимах установки приведены в качестве примера.

4.3.2. Программа управления двумя печами

Поскольку программа управления достаточно велика, она приведена блоками. Для удобства восстановления программы блоки пронумерованы.

Блок 1. Векторы прерываний. Определение переменных и констант

```
;Программа управления двумя электропечами
;========================================
.include "c:\avr\def\m8535def.inc"
.CSEG
.org 0
      rjmp RESET              ;Reset Handler
      nop                     ;rjmp EXT_INT0
      nop                     ;rjmp EXT_INT1
      nop                     ;rjmp TIM2_COMP
      nop                     ;rjmp TIM2_OVF
      nop                     ;rjmp TIM1_CAPT
      nop                     ;rjmp TIM1_COMPA
      nop                     ;rjmp TIM1_COMPB
      nop                     ;rjmp TIM1_OVF
      rjmp TIM0_OVF           ;Timer0 Overflow Handler
      nop                     ;rjmp SPI_STC
      nop                     ;rjmp UART_RXC
      nop                     ;rjmp UART_DRE
      nop                     ;rjmp UART_TXC
      rjmp ADC                ;ADC Conversion Complete Interrupt
                              ;Handler
      nop                     ;rjmp EE_RDY
      nop                     ;rjmp ANA_COMP
.def  cod = r1                ;Код - сегменты индикатора для
                              ;отображения цифры
.def  EEbt = r2               ;Данные EEPROM
```



```
.def  EEaH = r5               ;Для хранения HiAddr EEPROM
.def  tl = r6
.def  dgtnum = r8             ;Номер индицируемого разряда
.def  tm1 = r9
.def  tm2 = r12
.def  nmem = r13              ;Длительность (число циклов)
                              ;удержания кнопки «СОХРАНИТЬ»
.def  nADC = r14              ;Число преобразований АЦП для усреднения
.def  cntr = r15
.def  rgm = r16               ;Режим
.def  dgt = r17               ;Код для включения катода индикатора
.def  tmp = r18
.def  nmor = r19              ;Длительность (число циклов)
                              ;удержания кнопки «УВЕЛИЧИТЬ»
.def  nles = r20              ;Длительность (число циклов)
                              ;удержания кнопки «УМЕНЬШИТЬ»
.def  nrgm = r21              ;Длительность (число циклов)
                              ;удержания кнопки «РЕЖИМ»
.def  key = r24               ;Нажатая кнопка
.def  store = r25
.def  th = r25
.def  timh = r26
.def  EEaL = r26              ;Для хранения LoAddr EEPROM
;SRAM (распределение ОЗУ):
;RAM + [0..7]:коды индицируемых символов
;RAM + [$10..$13]:корректируемое десятичные число
.equ  RAM = $060              ;Начало SRAM
.equ  NUM = RAM + $14
.equ  C_E = RAM + $24
.equ  ADC6 = RAM + $35        ;Среднее значение АЦП 6
.equ  ADC7 = RAM + $3b        ;Среднее значение АЦП 7
.equ  AD6 = RAM + $40         ;Результат преобразования АЦП 6
.equ  AD7 = RAM + $44         ;Результат преобразования АЦП 7
.equ  sum6 = RAM + $48        ;Сумма значений АЦП 6
.equ  sum7 = RAM + $4c        ;Сумма значений АЦП 7
;Константы
.equ  SetPrm = 1              ;Режим установки параметров
.equ  OneCycl = 2
.equ  crgm = $32
.equ  cmor = $7
.equ  cles = $7
.equ  cmem = $32
.equ  cShft = 1000            ;Смещение для показаний температуры
.equ  cTAdd = 10
.equ  cTSub = 10
.equ  rWrk = 1
.equ  rTop = 2
.equ  rBot = 3
.equ  Eng_ = $f3
.equ  eeTop = $120 - 1
.equ  eeBot = $124 - 1
```

```
sTH:                                ;Константы для отображения «В°С»
        .db     $bf,$27
        .db     $39.0
        .db     0.0
        .db     0.0
sTL:                                ;Константы для отображения «Н°С»
        .db     $97,$27
        .db     $39.0
        .db     0.0
        .db     0.0
;БЛОК 1 конец
```

В блоке аппаратных прерываний остался обработчик прерывания переполнения таймера Т0: *rjmp TIM_OVF0* и добавился обработчик прерывания окончания преобразования АЦП: *rjmp ADC*.

Более подробное описание определенных директивами *.def* и *.equ* переменных и констант будет приведено по мере появления в программе использующих их команд.

Интерес представляют две группы из восьми констант, начинающиеся метками *sTH:* и *sTL:*. Вывод этих констант в порт С вызовет индикацию «В°С» и «Н°С» в режимах установки $T_{верх.}$ и $T_{ниж.}$ соответственно.

Замечание. Эти группы констант принципиально отличаются от других констант программы тем, что для них отведена область в той же памяти программ микроконтроллера, куда будет зашита и наша программа.

Как работает и где хранится константа, описанная так: *.equ cmem = $32?*

Если она не вызывается ни одной командой программы, то она вообще не записывается в память программ. Если же она вызывается несколькими командами (например, *ldi tmp,cmem*, *ldi YL,low(cmem)*, *ldi YH,high(cmem)*), то ее значение $32 при ассемблировании программы будет занесено непосредственно в код команды в качестве параметра. В трех представленных командах наша константа $32, а не ее имя cmem, будет присутствовать в кодах 1-й и 2-й команды, в коде 3-й команды ее не будет, поскольку *high*($32) = 0.

Во второй главе на **Рис. 12** (окно дизассемблированного кода и окно программы) в левом окне дизассемблированного кода Alarm строки

+0000000B: ED0F LDI r16,0xDF

+00000010: EF0F LDI r16,0xFF

являются хорошей иллюстрацией сказанного: во второй колонке представлены коды двух команд, отличающихся лишь константами, загружаемыми в регистр r16. Оба кода двухбайтные, причем константы даже не

представлены отдельными байтами. Очевидно, что Ex0x — это команда записи в регистр r16, а вместо xx вставляется константа.

Теперь рассмотрим группу, начинающуюся меткой *sTL:*. Константы группы будут записаны как данные в память программ микроконтроллера непосредственно в тело программы, что определяется директивами *.db*, предваряющими каждые два байта данных. Перед ними установлена метка для того, чтобы из самой программы, также записывающейся в память программ, можно было найти константы группы.

Важно! Программа не должна иметь возможности интерпретировать такие константы, как команды. Для этого в программе данные располагаются до метки *RESET:*, переход к которой происходит сразу после запуска по команде *rjmp RESET*, находящейся в нулевой ячейке памяти.

Каждая строка группы начинается директивой *.db*, предваряющей каждые два байта данных.

Первый байт группы $97 обеспечивает индикацию символа «Н», второй байт $27 — символа «°», третий байт $39 — символа «С». Определение кода для индикации нужного символа описывалось при рассмотрении программы организации бегущей строки.

Блок 2. Инициализация. Основной цикл программы

```
RESET:
        ldi     tmp,15              ;Сторожевой таймер = 2 с
        out     WDTCR,tmp
        wdr
        ldi     tmp,HIGH(RAMEND)    ;Начало стека
        out     SPH,tmp
        ldi     tmp,LOW(RAMEND)
        out     SPL,tmp
        cli                         ;Запретить все прерывания
;Инициализация таймера T0:
        ldi     tmp,1<<TOIE0        ;Разрешение прерывания
        out     TIMSK,tmp           ;Переполнения таймера T0
        ldi     tmp,0
        out     TCCR0,tmp
        ldi     tmp,1<<TOV0
        out     TIFR,tmp
        ldi     tmp,255 - 52        ;Переполнение таймера T0 через
                                    ;52 тика
        out     TCNT0,tmp
        ldi     tmp,3               ;Предварительный коэффициент
                                    ;деления 64
        out     TCCR0,tmp           ;Tovf = 1/Fтакт/(64 × 52) =
                                    ;= 64 × 52/4[МГц] = 832[мкс]
        sei                         ;Разрешить все прерывания
```

```
;Инициализация портов ввода/вывода:
        ldi     tmp,$ff             ;Линии портов C и D
        out     DDRD,tmp            ;Выходы
        out     DDRC,tmp
        ldi     tmp,$ff - $03       ;PB0 и PB1-входы
        out     DDRB,tmp            ;PB2…PB7-выходы
        ldi     tmp,$ff             ;На всех линиях портов C и D
        out     PORTD,tmp           ;ВЫСОКИЕ уровни
        out     PORTC,tmp
        ldi     tmp,Eng_            ;Eng_= $f3, на линиях PB2,PB3
                                    ;НИЗКИЕ уровни, на остальных PB
                                    ;ВЫСОКИЕ уровни
        out     PORTB,tmp
;Обнулить счетчики длительности нажатия кнопок:
        clr     nmor                ;Кнопки «УВЕЛИЧИТЬ»
        clr     nles                ;Кнопки «УМЕНЬШИТЬ»
        clr     nrgm                ;Кнопки «РЕЖИМ»
        clr     nmem                ;Кнопки «СОХРАНИТЬ»
        ldi     dgt,0b10000000
        clt                         ;Очистить флаг T
        ldi     rgm,rWrk            ;Режим «Работа»
        ldi     tmp,0
        out     EECR,tmp
;================================
;Основная часть (бесконечный цикл)
start:  wdr
        tst     dgt
        brne    m1
        ldi     dgt,0b10000000
m1:
        ldi     YL,low(RAM)
        ldi     YH,high(RAM)
        clr     key
        ldi     tmp,0
        out     PORTD,dgt
        out     PORTC,tmp
        sbrc    dgt,0
        rcall   mdgt0
        sbrc    dgt,1
        rcall   mdgt1
        sbrc    dgt,2
        rcall   mdgt2
        sbrc    dgt,3
        rcall   mdgt3
        sbrc    dgt,4
        rcall   mdgt4
        sbrc    dgt,5
        rcall   mdgt5
        sbrc    dgt,6
        rcall   mdgt6
        sbrc    dgt,7
```



```
        rcall   mdgt7
mdbg1:
        out     PORTC,cod
        rcall   mkey
        brts    m2
        rcall   mInd
m2:
        lsr     dgt                 ;Следующая цифра
        ldi     tmp,1<<SE
        out     MCUCR,tmp
        sleep
        rjmp    start
;БЛОК 2, конец
```

Блок 2 начинается меткой *RESET:*. Как обычно, выполняется инициализация сторожевого таймера, стека, портов ввода/вывода, таймера и различных служебных переменных.

Таймер T0 инициализируется так, что прерывание его переполнения наступает через 832 мкс.

В нашей программе используется только три линии порта B: Heat1 (управление Печью B), MOSI (управление Печью H) и MISO (проверка состояния кнопок). Тем не менее, инициализация порта B сохранена такой же, какой она была в исходной программе контроллера станка для совместимости с действующим макетом.

Счетчики длительности нажатия кнопок (*nmor, nles, nrgm, nmem*) фиксируют время, в течение которого нажата кнопка. Если это время больше заданного, счетчик сбрасывается и выполняется действие, соответствующее нажатой кнопке. Так, при нажатии кнопки «РЕЖИМ» произойдет переход к следующему режиму, если кнопка не отпущена, то после очередного заполнения счетчика длительности нажатия кнопки «РЕЖИМ» произойдет переход к очередному режиму, пока кнопка не будет отпущена.

Далее в переменную *dgt* заносится двоичное число 10000000. Назначение этой переменной такое же, как переменной *catod* в предыдущей программе: она выводится в порт D и управляет последовательным свечением разрядов индикатора.

В программе в качестве переменной используется флаг T регистра флагов. Его использование аналогично использованию булевой переменной в языках высокого уровня: флаг можно установить или сбросить, а также проанализировать его состояние — установлен он или сброшен. Когда в программе ощущается недостаток регистров для хранения переменных, флаг T может оказаться полезным. Команда *clt* сбрасывает флаг T.

Далее в переменную *rgm* заносится значение *rWrk*, соответствующее режиму «Работа», затем очищается регистр EECR, управляющий обращением к EEPROM микроконтроллера.

Далее следует основная часть программы, представляющая собой бесконечный цикл, начинающийся меткой *start:* и заканчивающийся командой *rjmp start*.

После сброса сторожевого таймера командой *wdr* проверяется состояние переменной *dgt*, управляющей катодами индикатора: если она стала равной нулю, в нее вновь заносится двоичное число 10000000.

В парный регистр Y заносится значение RAM — адрес начала ОЗУ микроконтроллера. Содержимое регистра Y будет использоваться в подпрограммах, вызываемых из рассматриваемой здесь основной части программы.

После очистки переменной *key* в порт С выводится нулевое значение, на короткое время полностью выключая индикатор. В порт D выводится значение переменной *dgt*.

Единица может присутствовать только в одном из восьми разрядов переменной *dgt*. Следующий блок определяет этот разряд и вызывает соответствующую подпрограмму *mdgt0...mdgt7*. Эти подпрограммы будут рассмотрены позже. Сейчас замечу, что в них определяется код символа (переменная *cod*), который должен засветиться на индикаторе, в разряде, определенном переменной *dgt*. Здесь же проверяется, не нажата ли одна из кнопок и если она нажата достаточно долго, в переменную *key* заносится ее код.

Далее следует команда с меткой *mdbg1:*. В порт С выводится код символа, вызывая индикацию этого символа.

В подпрограмме *mkey* проверяется, была ли нажата какая-то кнопка, если была, то вызывается подпрограмма обработки нажатия кнопки.

Командой *brts* проверяется состояние флага Т, которое зависит также от режима, в который перешла программа. Если флаг сброшен, вызывается подпрограмма *mInd*. Далее в переменной *dgt* производится сдвиг единицы влево для подготовки индикации очередного разряда, а следующие три команды переводят микроконтроллер в состояние Sleep, режим Idle, который уже рассматривался в программе организации бегущей строки (подраздел 0).

Очередное прерывание переполнения таймера Т0 «разбудит» микроконтроллер, «заснувший» по команде *sleep*, выполнится обработчик этого прерывания, программа вернется к последней команде, завершающей бесконечный цикл (*rjmp start*), произойдет переход к метке *start:* и весь цикл повторится.

Блок 3. Обработчики прерываний

```
;Обработчики прерываний
TIM0_OVF:                       ;Обработчик прерывания таймера T0
      wdr                       ;Сброс сторожевого таймера
;АЦП 6, инициализация
      ldi   tmp,6
      out   ADMUX,tmp
```



```
      sei
      ldi   tmp,(1<<ADEN) + (1<<ADSC) + (1<<ADIE) +
            + (1<<ADPS2) + (1<<ADPS0)
      out   ADCSRA,tmp
      ldi   tmp,1<<SE
      out   MCUCR,tmp
      sleep
      in    tmp,ADCL
      in    store,ADCH
      ldi   YL,low(AD6)
      ldi   YH,high(AD6)
      st    Y +,store
      st    Y,tmp
;АЦП 7, инициализация
      ldi   tmp,7
      out   ADMUX,tmp
      sei
      ldi   tmp,(1<<ADEN) + (1<<ADSC) + (1<<ADIE) +
            + (1<<ADPS2) + (1<<ADPS0)
      out   ADCSRA,tmp
      ldi   tmp,1<<SE
      out   MCUCR,tmp
      sleep
      in    tmp,ADCL
      in    store,ADCH
      ldi   YL,low(AD7)
      ldi   YH,high(AD7)
      st    Y +,store
      st    Y,tmp
      ldi   tmp,255 - 52
      out   TCNT0,tmp
      reti
;Обработчик TIM_OVF0, конец
;Обработчик прерывания АЦП
ADC:  wdr
      clr   tmp
      out   ADCSRA,tmp
      reti
;Конец обработчиков прерываний
```

В этот блок включены обработчик прерывания переполнения таймера Т0, начинающийся меткой *TIM_OVF0:*, и обработчик прерывания окончания преобразования АЦП (метка *ADC:*).

Обработчик таймера Т0 запускает преобразование АЦП 6, ждет его окончания, сохраняет результаты в ОЗУ микроконтроллера, выполняет те же действия для АЦП 7, после чего снова запускает таймер Т0 на счет.

Временной интервал между двумя вызовами обработчика прерывания таймера Т0 можно считать базовым для всей программы, он соответствует продолжительности индикации одного разряда. Далее будем ссылаться на

него как на цикл программы или просто цикл (не путать с циклом микроконтроллера!).

Надо сказать несколько слов о работе встроенного АЦП микроконтроллера. В микроконтроллере всего лишь один АЦП, но, в зависимости от программно определяемого состояния регистра ADMUX, вход АЦП соединяется с одной из восьми линий порта А. Поэтому употребление терминов АЦП 6 или АЦП 7 подразумевает, что ко входу АЦП подключены линии PA6 или PA7.

Это значит, что АЦП не может вести параллельное (одновременное) преобразование сигналов, поступающих по разным каналам. Сигнал, поступающий по одному каналу, запоминается имеющимся в микроконтроллере устройством выборки/хранения, а затем преобразовывается. Когда преобразование завершится, произойдет прерывание окончания преобразования АЦП (далее — прерывание АЦП). Только после этого может быть начато преобразование сигнала, поступающего по другому каналу.

Корпорация «Atmel» рекомендует выбирать тактовую частоту АЦП не выше 200 кГц, в противном случае резко увеличивается ошибка преобразования. Тактовая частота АЦП получается делением тактовой частоты микроконтроллера на коэффициент предварительного деления.

> **Замечание.** Аналогичный коэффициент предварительного деления частоты (Prescaling) мы программно определяем при работе с таймерами, в частности, в данной программе он равен 64 и определяется значением, посланным в регистр TCCR0 (мы посылали тройку).

Коэффициент деления для АЦП выбирается из ряда 2, 4, 8, 16, 32, 64 и 128. Надо, чтобы АЦП работал как можно быстрее, но при этом его тактовая частота не превышала 200 кГц. При тактовой частоте микроконтроллера 4 МГц коэффициент 16 обеспечит тактовую частоту АЦП 250 кГц (4000000/16 = 250000 Гц), что превышает максимально допустимую частоту, при коэффициенте 32 тактовая частота АЦП составит 125 кГц. Поэтому выбираем коэффициент 32. Длительность одного такта частоты 125 кГц составляет 8 мкс (1/125000 = 0.000008 с).

Встроенный в микроконтроллер 10-разрядный АЦП последовательного приближения требует 14 тактов для полного преобразования сигнала в режиме единичного преобразования. Поэтому общее время преобразования составит $8 \times 14 = 112$ мкс, а время двух преобразований (АЦП 6 и АЦП 7) — 224 мкс.

Чтобы определить продолжительность цикла программы, это время должно быть добавлено ко времени счета таймера. Сюда же надо добавить время выполнения команд обработчика прерывания таймера T0 (мы уже проводили подобный подсчет, в том числе с использованием отладчика), а

также удвоенное время выполнения обработчика прерывания АЦП (два АЦП — два раза вызывается обработчик АЦП).

Временная диаграмма, приведенная на **Рис. 27**, поможет разобраться с подсчетом длительности цикла.

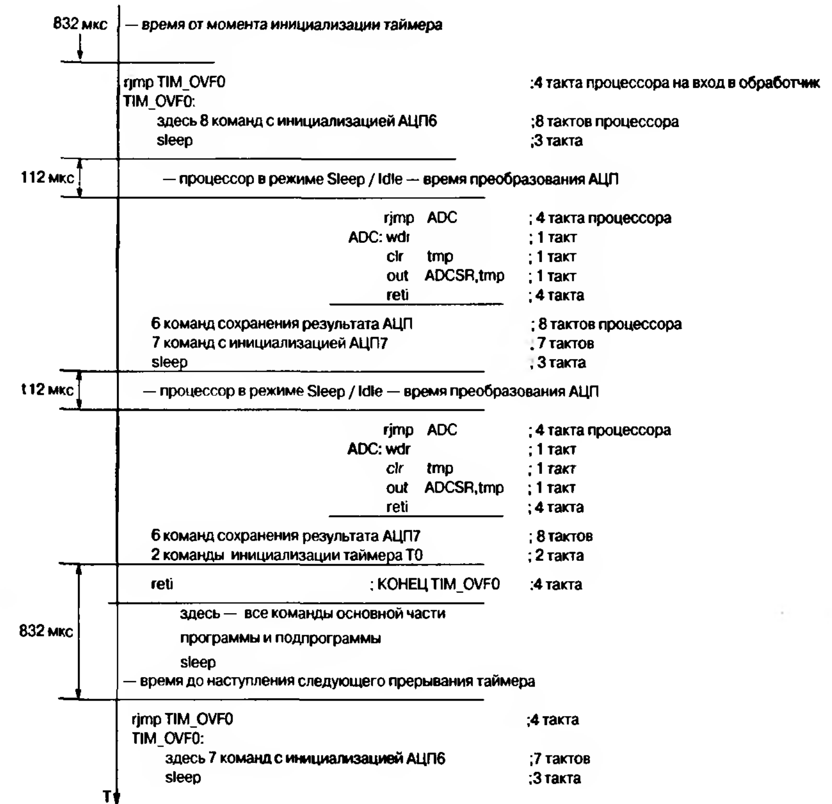


Рис. 27. Временная диаграмма цикла программы

Поскольку в обработчике прерывания таймера команды выполняются за 43 такта процессора (без команды *reti*, она не удлиняет цикл), команды прерывания АЦП выполняются за 11 тактов, а длительность одного такта процессора составляет 0.25 мкс (тактовая частота 4 МГц), общая продолжительность цикла программы составит:

$$832 + 224 + 43 \times 0.25 + 2 \times 11 \times 0.25 = 1083 \text{ мкс.}$$

Что из этого следует и зачем нужен такой расчет?

1. Это значит, что обновление индикации 8 символов будет происходить за 8.664 мс, а частота мелькания составит 1000/8.664, более 100 Гц, поэтому мелькание цифр не будет влиять на восприятие человеческим глазом.
2. Время преобразования АЦП составляет 112 мкс, сигнал может значительно измениться за это время, именно этим обусловлено наличие встроенного в микроконтроллер устройства выборки/хранения, которое запоминает сигнал (выполняет выборку сигнала) в момент начала преобразования и удерживает его неизменным в течение всего времени преобразования.
3. Можно ли запустить выполнение преобразования АЦП без выключения таймера с тем, чтобы интервал определялся лишь таймером? Это возможно, но таймер, работающий в непосредственной близости от АЦП, создает импульсные помехи, поэтому корпорация «Atmel» рекомендует выключать таймер на время преобразования для получения более точного результата. В этой программе остановки таймера нет, ее можно ввести, однако введенное в программу усреднение результатов по множеству циклов обеспечивает достаточное подавление помех.

В прикладных программах часто требуется достаточно точно приблизить интервал, обеспечиваемый таймером к заданному, поэтому приведенная методика определения интервала может оказаться полезной.

> **Замечание.** Для точного определения интервала в обработчик прерывания таймера (или АЦП) можно дополнительно вводить пустые команды *nop*. Однако частота, обозначенная на корпусе кварцевого резонатора, не точно соответствует тактовой частоте микроконтроллера, погрешность частоты определяется как допустимым разбросом характеристик кварцевого резонатора, так и схемой его включения. Возможны и ошибки в расчетах.
> Для «подстройки» длительности цикла программы можно вывести на какую-либо свободную линию одного из портов ввода/вывода чередующиеся от цикла к циклу значения 0 и 1, вставив необходимую последовательность команд в начале обработчика прерывания таймера. Сигнал на линии контролируется частотомером. Период сигнала, индицируемый частотомером, будет вдвое больше интервала работы таймера, так как период измеряется частотомером не от появления нуля до появления единицы, а между появлением двух единиц.

Вернемся к программе. В обработчике прерывания таймера T0 после сброса сторожевого таймера производится инициализация АЦП 6:

- в регистр выбора канала ADMUX выводится номер линии, подключаемой к АЦП;

- разрешаются все прерывания;
- в регистре управления ADCSRA устанавливаются следующие разряды:
  a) ADEN — работа АЦП разрешена,
  b) ADSC — режим однократного преобразования,
  c) ADIE — прерывание АЦП разрешено,
  d) ADPS2 и ADPS0 — определяют коэффициент предварительного деления, равный 32.

Теперь преобразование АЦП запущено, остается ждать его завершения, для этого микроконтроллер переводится в знакомый нам режим Idle командой *sleep*. Как уже было замечено, «разбудить» микроконтроллер, находящийся в режиме Idle, может не только прерывание таймера, но и другие прерывания, в том числе и прерывание АЦП.

Через 112 мкс после начала преобразования произойдет прерывание АЦП, после команды *sleep* выполнится команда *rjmp ADC*, находящаяся в блоке векторов прерываний (смотрите БЛОК 1 программы), далее будет выполнен обработчик прерывания АЦП, начинающийся меткой *ADC:*, после завершения которого выполнятся команды *in tmp,ADCL* и *in store,ADCH*, следующие за командой *sleep*.

> **Важно!** Последовательность доступа к результату преобразования нарушать нельзя: сначала считывается младший байт результата (ADCL), затем старший (ADCH). В противном случае результат может оказаться ошибочным.

Затем в регистр Y заносится адрес AD6 для хранения результата преобразования АЦП 6, и в пару соседних ячеек записывается результат.

Те же действия повторяются для АЦП 7, только к АЦП подключается линия PA7, а результаты преобразования записываются в пару соседних ячеек, адрес первой из них — AD7.

Далее вновь инициализируется таймер, что обеспечивает отсчет нового интервала времени до следующего прерывания таймера в 832мкс.

В обработчике прерывания АЦП в очередной раз сбрасывается сторожевой таймер, а также очищается содержимое регистра управления АЦП, это вызывает прекращение работы АЦП и запрет прерываний АЦП.

Блок 4. Подпрограммы определения кода символа mdgt0...mdgt7 и изменения режима tkey

```
;Определение 7-сегментного кода и проверка нажатия кнопок
mdgt0:
            ldd    cod,Y + 7
md0end:     ret
mdgt1:
            ldd    cod,Y + 6
```

```
            sbic  PINB,6
            clr   nrgm
            inc   nrgm
            cpi   nrgm,crgm
            brcs  md1end
            dec   nrgm
            ldi   key,1<<1
md1end:     ret
mdgt2:
            ldd   cod,Y + 5
            sbic  PINB,6
            clr   nmor
            inc   nmor
            cpi   nmor,cmor
            brcs  md2end
            dec   nmor
            ldi   key,1<<2
md2end:     ret
mdgt3:
            ldd   cod,Y + 4
            sbic  PINB,6
            clr   nles
            inc   nles
            cpi   nles,cles
            brcs  md3end
            dec   nles
            ldi   key,1<<3
md3end:     ret
mdgt4:
            ldd   cod,Y + 3
            sbic  PINB,6
            clr   nmem
            inc   nmem
            mov   tmp,nmem
            cpi   tmp,cmem
            brcs  md4end
            dec   nmem
            ldi   key,1<<4
md4end:     ret
mdgt5:
            ldd   cod,Y + 2
md5end:     ret
mdgt6:
            ldd   cod,Y + 1
md6end:     ret
mdgt7:
            ldd   cod,Y + 0
            rcall avrT
            rcall mheat
md7end:     ret
;===========
```



```
mkey:       ldi   YL,low(RAM + 8)
            ldi   YH,high(RAM + 8)
            sbrc  key,1
            rcall mRgm
            sbrc  key,2
            rcall mMor
            sbrc  key,3
            rcall mLes
            sbrc  key,4
            rcall mMem
            ret
;Конец определения кода и кнопки
```

БЛОК 4 состоит из 8 подпрограмм *mdgt0...mdgt7*, выполняемых в зависимости от того, какой из 8 разрядов индикатора подсвечивается, а также из подпрограммы *mkey*, выбирающей режим и вызывающая соответствующие подпрограммы в зависимости от нажатой кнопки.

В ОЗУ микроконтроллера в 8 ячейках хранятся коды 8 символов, отображаемых индикатором. Адрес первой из этих ячеек, хранящийся в константе RAM, уже занесен в регистр Y в основном цикле программы (смотри подраздел 4.3.2.2), оттуда же вызывается одна из подпрограмм *mdgt0...mdgt7*.

Если в переменной *dgt* установлен 0-й разряд, вызывается подпрограмма *mdgt0*, в переменную *cod* загружается код из ячейки с адресом (Y + 7).

Когда в переменной *dgt* установлен 1-й разряд, вызывается подпрограмма *mdgt1*, в переменную *cod* загружается код из ячейки с адресом (Y + 6). Команда *sbic PINB,6* проверяет, не нажата ли кнопка, соединенная с катодом, управляемым по линии PD1 (кнопка «РЕЖИМ»). Переменная *dgt*, 1-й разряд которой установлен, выводится в порт D. Если кнопка нажата, то появившееся низкое напряжение на катоде, определяемом переменной *dgt*, через замкнутые контакты кнопки передастся и на контакт PB6 (PINB6). Если кнопка не нажата, выполняется команда, очищающая счетчик нажатия *nrgm* кнопки «РЕЖИМ».

Далее содержимое счетчика увеличивается на единицу и сравнивается с заданным числом *crgm*. Если кнопка отпущена, то в переменной *nrgm* хранится единица, если нажата, то ее содержимое увеличивается на единицу каждые 8 циклов программы. Вызов этой подпрограммы повторится только после того, как индикатор последовательно отобразит все 8 цифр. Как было подсчитано, это время составляет 8.664 мс. Константа *crgm* = \$32 или 50 в десятичном формате, значит, нажатие кнопки будет воспринято только через $8.664 \times 50 = 433.2$ мс или примерно через полсекунды.

Итак, если кнопка была нажата в течение указанного времени, то содержимое счетчика *nrgm* уменьшается на единицу, а в переменную *key* вводится код кнопки «РЕЖИМ».

Когда в переменной *dgt* установлен 2-й разряд, вызывается подпрограмма *mdgt2*, в переменную *cod* загружается код из ячейки с адресом (Y + 5). Команда *sbic PINB,6* проверяет, не нажата ли кнопка, соединенная с катодом, управляемым по линии PD2. Это кнопка «УВЕЛИЧИТЬ». Если кнопка не нажата, выполняется команда, очищающая счетчик нажатия nmor кнопки «УВЕЛИЧИТЬ».

Далее содержимое счетчика увеличивается на единицу и сравнивается с заданным числом *cmor*. Если кнопка отпущена, то в переменной *nmor* хранится единица, если нажата, то ее содержимое увеличивается на единицу каждые 8 циклов программы. Константа *cmor* равна 7. Значит, нажатие кнопки будет воспринято через 60.7 мс (8.664 × 7).

Если кнопка была нажата в течение такого времени, то содержимое счетчика *nmor* уменьшается на единицу, а в переменную *key* вводится код кнопки «УВЕЛИЧИТЬ».

Когда в переменной *dgt* установлен 3-й разряд, вызывается подпрограмма *mdgt3*, в переменную *cod* загружается код из ячейки с адресом (Y + 4). Далее проверяется нажатие кнопки «УМЕНЬШИТЬ». Если кнопка не нажата, выполняется команда, очищающая счетчик нажатия *nles* кнопки «УМЕНЬШИТЬ».

Содержимое счетчика *nles* увеличивается на единицу и сравнивается с заданным числом *cles*. Если кнопка была нажата в течение 60.7 мс, содержимое счетчика *nles* уменьшается на единицу, а в переменную *key* вводится код кнопки «УМЕНЬШИТЬ».

Когда в переменной *dgt* установлен 4-й разряд, вызывается подпрограмма *mdgt4*, в переменную *cod* загружается код из ячейки с адресом (Y + 3). Проверка нажатия кнопки «СОХРАНИТЬ» аналогична проверке для кнопки «РЕЖИМ», счетчик нажатия кнопки — *nmem*, а в переменную *key* заносится код кнопки «СОХРАНИТЬ».

> **Замечание.** Реакция программы на нажатие кнопок «УВЕЛИЧИТЬ» и «УМЕНЬШИТЬ» на порядок выше, чем на нажатие кнопок «РЕЖИМ» и «СОХРАНИТЬ». При нажатии первых двух кнопок величина температуры, которую мы хотим изменить, корректируется лишь на 0.1°С за 60 мс, так как регулировка происходит в диапазоне 200...270°С, а для индикации используется 4 разряда, то есть младший разряд индицирует десятые доли градуса.

Если в переменной *dgt* установлен 5-й, 6-й или 7-й разряд, выполняется соответственно подпрограмма *mdgt5*, *mdgt6* или *mdgt7*, а в переменную *cod* заносится код из ячейки (Y + 2), (Y + 1) или Y. Лишь в подпрограмме *mdgt7* дополнительно вызывается подпрограмма *avrT* усреднения температурных данных ПечиВ и ПечиН и подпрограмма *mheat* управления нагревательными спиралями печей.

Подпрограмма *mkey* анализирует код нажатия кнопок. Если была нажата кнопка «РЕЖИМ», вызывается подпрограмма *mRgm*, при нажатии кнопок «УВЕЛИЧИТЬ», «УМЕНЬШИТЬ» или «СОХРАНИТЬ» вызываются подпрограммы *mMor*, *mLes* или *mMem* соответственно. Перед анализом состояния кнопок в регистр Y заносится адрес (RAM + 8).

Блок 5. Подпрограммы режимов mRgm, mMor, mLes, mMem

```
;Изменение режима при нажатии кнопки РЕЖИМ
;Работа->Ввод Тверх.->Ввод Тниж.
mRgm: clr    nrgm
      clt
mRg6: cpi    rgm,rBot
      brne   mRg7
      ldi    rgm,rWrk
      rjmp   mRg9
mRg7: cpi    rgm,rTop
      brne   mRg8
      ldi    rgm,rBot
mRg8: cpi    rgm,rWrk
      brne   mRg9
      ldi    rgm,rTop
mRg9: ret
;================
;Увеличение числа при нажатии кнопки УВЕЛИЧИТЬ
mMor: clr    nMor
      cpi    rgm,rWrk
      brne   mM1
      rjmp   mMor3
mM1:  set
      ldi    YL,low(NUM - 4)
      ldi    YH,high(NUM - 4)
      ld     store,Y
      inc    store              ;ПЕРВАЯ десятичная цифра
      st     Y,store
      cpi    store,$a
      brcs   mMor1
      ldi    store,0
      st     Y,store
      adiw   YL,1
      ld     store,Y
      inc    store              ;ВТОРАЯ десятичная цифра
      st     Y,store
      cpi    store,$a
      brcs   mMor1
      ldi    store,0
      st     Y,store
      adiw   YL,1
      ld     store,Y
      inc    store              ;ТРЕТЬЯ десятичная цифра
```

```
        st      Y,store
        cpi     store,$a
        brcs    mMor1
        ldi     store,0
        st      Y,store
        adiw    YL,1
        ld      store,Y
        inc     store                   ;ЧЕТВЕРТАЯ десятичная цифра
        st      Y,store
        cpi     store,$a
        brcs    mMor1
        ldi     store,0
        st      Y,store
mMor1:  ldi     YH,High(NUM)
        ldi     YL,Low(NUM)
        ldi     ZH,High(RAM + 4)
        ldi     ZL,Low(RAM + 4)
        rcall   mDecod
mMor3:  ret
;========
;Уменьшение числа при нажатии кнопки УМЕНЬШИТЬ
mLes:   clr     nLes
        cpi     rgm,rWrk
        brne    mL1
        rjmp    mLes3
mL1:    set
        ldi     YL,low(NUM - 4)
        ldi     YH,high(NUM - 4)
        ld      store,Y
        dec     store                   ;Первая десятичная цифра
        st      Y,store
        cpi     store,$ff
        brne    mLes1
        ldi     store,9
        st      Y,store
        adiw    YL,1
        ld      store,Y
        dec     store                   ;Вторая десятичная цифра
        st      Y,store
        cpi     store,$ff
        brne    mLes1
        ldi     store,9
        st      Y,store
        adiw    YL,1
        ld      store,Y
        dec     store                   ;Третья десятичная цифра
        st      Y,store
        cpi     store,$ff
        brne    mLes1
        ldi     store,9
        st      Y,store
```



```
        adiw    YL,1
        ld      store,Y
        dec     store                   ;Четвертая десятичная цифра
        st      Y,store
mLes1:  ldi     YH,High(NUM)
        ldi     YL,Low(NUM)
        ldi     ZH,High(RAM + 4)
        ldi     ZL,Low(RAM + 4)
        rcall   mDecod
mLes3:  ret
;====================
;Запись числа в EEPROM
mMem:   brtc    mMem2
        cli
        rcall   WriteEE
        sei
        clt
        ldi     rgm,rWrk
mMem2:  ret
;Конец обработки нажатия кнопок
```

Подпрограмма *mRgm* вызывается при нажатии кнопки «РЕЖИМ». В подпрограмме очищаются счетчик *nrgm* и флаг T регистра флагов (он же регистр статуса). Команда *cpi* сравнивает значение, хранящееся в переменной *rgm*, с константами *rBot*, *rTop* и *rWrk*, соответствующими режиму установки температуры ПечиН, режиму установки температуры ПечиВ и рабочему режиму.

Если значение переменной равно *rBot*, то в переменную заносится значение *rWrk*.

Когда значение *rgm* равно *rTop*, в переменную заносится значение *rBot*, при равенстве содержимого переменной величине *rWrk* в переменную *rgm* записывается значение *rTop*.

Это значит, что при нажатии кнопки «РЕЖИМ» режим индикации и работы будет изменяться, как было показано на **Рис. 26**. Если кнопку удерживать, то смена режимов будет происходить по кольцу через каждые полсекунды.

Подпрограмма *mMor* вызывается при нажатии кнопки «УВЕЛИЧИТЬ». В подпрограмме очищаются счетчик *nmor*, проверяется состояние переменной *rgm*, если оно соответствует режиму «Работа», вызывается подпрограмма *mDecod* и производится выход из подпрограммы.

В противном случае устройство работает в режиме увеличения температуры ПечиВ или ПечиН. Тогда устанавливается флаг T регистра флагов, в регистр Y заносится адрес первой десятичной цифры корректируемого числа, эта цифра считывается из ячейки памяти в переменную *store* и увеличивается на единицу. Если содержимое не стало равным 10, то перемен-

ная сохраняется в своей ячейке памяти, вызывается подпрограмма *mDecod*, а затем подпрограмма завершается.

Если *store* = 10, то содержимое регистра Y увеличивается на единицу, регистр указывает на ячейку, хранящую второй разряд десятичного числа, содержимое ячейки помещается в переменную, а далее происходит проверка на равенство числу десять, аналогичная проверке первого разряда.

Те же действия производятся с 3-м и 4-м разрядом корректируемого 4-разрядного десятичного числа. По существу, все это напоминает сложение десятичного числа с единицей в столбик на бумажке.

Упоминавшаяся подпрограмма *mDecod* преобразует скорректированное десятичное число в коды управления 7 сегментами индикатора и заносит эти коды в соответствующие ячейки памяти, опрашиваемые при индикации. В режимах коррекции температуры ПечиВ и ПечиН это ячейки, соответствующие четырем разрядам индикатора справа. Помните индикацию в этих режимах? Для ПечиВ она выглядит так: В°С 2301.

Для ПечиН вместо символа «В» — символ «Н». Здесь «2301» — это корректируемое число, именно коды для индикации цифр таких чисел определяет подпрограмма *mDecod*.

Подпрограмма *mLes* вызывается при нажатии кнопки «УМЕНЬШИТЬ» и отличается от подпрограммы *mMor* тем, что из содержимого ячеек вычитается единица, а результат сравнивается не с числом десять, а с числом $ff, так как в ячейке хранится байт данных и при вычитании из нуля байт приобретает значение $ff.

Здесь также уместна аналогия с вычитанием из десятичного числа единицы в столбик на бумажке. Полагаю, что после разбора подпрограммы *mMor* подпрограмма *mLes* будет проста для понимания.

Подпрограмма *mMem* вызывается при нажатии кнопки «СОХРАНИТЬ». Подпрограмма начинается с тестирования состояния флага T, если он равен нулю, то выполнение подпрограммы завершается, в противном случае запрещаются все прерывания, вызывается подпрограмма записи откорректированного параметра в EEPROM микроконтроллера, прерывания вновь разрешаются, флаг T сбрасывается, а в переменную *rgm* записывается значение *rWrk*, что вызовет переход устройства в режим «Работа».

Флаг T используется в уже рассмотренных блоках программы. Его сброс происходит при нажатии кнопок «РЕЖИМ» и «СОХРАНИТЬ», а установка — при нажатии кнопок «УВЕЛИЧИТЬ» и «УМЕНЬШИТЬ». Если при входе в подпрограмму *mMem* флаг оказывается сброшенным, это значит, что перед нажатием кнопки «СОХРАНИТЬ» была нажата кнопка «РЕЖИМ» или кнопка «СОХРАНИТЬ», то есть параметры не изменялись и сохранять нечего. Сохранение же результата в EEPROM происходит при

нажатии кнопки «СОХРАНИТЬ», если перед этим в режиме установки температуры ПечиВ или ПечиН нажимались кнопки «УВЕЛИЧИТЬ» или «УМЕНЬШИТЬ».

Если сократить наши рассуждения, то получится, что нажатие кнопки «СОХРАНИТЬ» работает только в режиме коррекции и только после нажатия кнопок «УВЕЛИЧИТЬ» или «УМЕНЬШИТЬ».

Блок 6. Подпрограммы mInd и hex_dc

```
;Индикация «В°С» и «Н°С» при вводе Tмакс и Tмин
mInd:  cpi    rgm,rTop
       brne   mI1
       ldi    store,low(eeTop)
       ldi    ZL,low(sTH x 2)
mI1:   cpi    rgm,rBot
       brne   mI2
       ldi    store,low(eeBot)
       ldi    ZL,low(sTL x 2)
mI2:   cpi    rgm,rWrk
       breq   mI9
       rjmp   mI10
mI9:   push   tl
       push   th
       push   XL
       push   XH
       ldi    YL,low(ADC6 - 2)
       ldi    YH,high(ADC6 - 2)
       rcall  hex_dc
       ldi    YL,low(ADC7 - 2)
       ldi    YH,high(ADC7 - 2)
       rcall  hex_dc
       pop    XH
       pop    XL
       pop    th
       pop    tl
       ldi    YH,High(ADC6 - 2)
       ldi    YL,Low(ADC6 - 2)
       ldi    ZH,High(RAM + 8)
       ldi    ZL,Low(RAM + 8)
       rcall  mDecod
       ldi    YH,High(ADC7 - 2)
       ldi    YL,Low(ADC7 - 2)
       ldi    ZH,High(RAM + 4)
       ldi    ZL,Low(RAM + 4)
       rcall  mDecod
       rjmp   mInd1
mI10:  ldi    YH,high(RAM + 8)
       ldi    YL,low(RAM + 8)
       ldi    tmp,8
copyStp:
```

```
        lpm
        st      - Y,r0
        adiw    ZL,1
        dec     tmp
        brne    copyStp
        ldi     tmp,high(eeBot)
        out     EEARH,tmp
        out     EEARL,store        ;EEAR<-$00 - 1 (начальный адрес минус 1)
        ldi     YH,High(NUM)
        ldi     YL,Low(NUM)
        rcall   ReadEE
        ldi     YH,High(NUM)
        ldi     YL,Low(NUM)
        ldi     ZH,High(RAM + 4)
        ldi     ZL,Low(RAM + 4)
        rcall   mDecod
mIndl:  ret
;Конец (Индикация «В°С» и «Н°С» при вводе Тмакс и Тмин)
;Преобразование шестнадцатеричного кода в десятичный код
hex_dc:ld       XH,Y +
        ld      XL,Y +
        sbiw    YL,2
        ldi     th,low(1000)
        mov     tl,th
        ldi     th,high(1000)
        rcall   count
        ldi     th,low(100)
        mov     tl,th
        ldi     th,high(100)
        rcall   count
        ldi     th,low(10)
        mov     tl,th
        ldi     th,high(10)
        rcall   count
        st      - Y,XL
        ret
count:
        ldi     tmp,$ff
cl:     inc     tmp
        sub     XL,tl
        sbc     XH,th
        brcc    cl
        st      - Y,tmp
        add     XL,tl
        adc     XH,th
        ret
;Конец (Преобразование шестнадцатеричного кода в десятичный код)
```

Подпрограмма *mInd* вызывается из основного цикла программы, рассматривавшегося в подразделе 4.3.2.2, при сброшенном флаге Т. Посредст-

вом проверки содержимого переменной *rgm* в подпрограмме определяется, в каком режиме находится устройство.

Если это режим установки $T_{верх.}$, то в переменную *store* заносится адрес ячеек EEPROM микроконтроллера, в которых хранится значение В°С (температура, которая должна поддерживаться ПечьюВ). В переменную ZL — адрес ячеек памяти программ микроконтроллера, в которых хранятся коды трех символов «В°С», — они будут отображаться на индикаторе слева при коррекции значения В°С, которое будет отображаться 4 цифрами справа.

Замечание. Термины «адрес ячеек» или «адрес области ОЗУ» подразумевают адрес начальной ячейки или адрес ячейки, следующей за областью ОЗУ. Адрес ячейки легко определить по тексту самой программы.

Для режима установки $T_{ниж.}$ в переменную *store* заносится адрес ячеек EEPROM микроконтроллера, в которых хранится значение Н°С (температура, которая должна поддерживаться ПечьюН). В переменную ZL записывается адрес ячеек памяти программ микроконтроллера, в которых хранятся коды символов «Н°С». Для этих двух режимов выполняется часть подпрограммы от метки *mI10:* до конца подпрограммы.

Если же устройство находится в режиме «Работа», выполняется часть подпрограммы от метки *mI9:* до метки *mI10:*, после чего выполнение подпрограммы завершается. В этом режиме командами *push* содержимое переменных *tl*, *th*, *XL*, *XH* помешается в стек.

В регистр Y заносится адрес ячеек ОЗУ, хранящих усредненное шестнадцатеричное значение результатов преобразований АЦП 6, затем вызывается подпрограмма *hex_dc*, преобразующая шестнадцатеричное число в 4-разрядное десятичное и записывающая его в 4 ячейки ОЗУ непосредственно перед ячейками, хранящими шестнадцатеричное значение — об этом подробнее при рассмотрении подпрограммы *hex_dc*. Это десятичное число должно быть отражено на индикаторе как показание текущей температуры ПечиВ.

Замечание. Не удивляйтесь тому, что усредненное значение результатов преобразований АЦП 6 хранится по адресу (ADC6 – 2) — это было удобно в исходной программе, а рассматриваемой программе досталось «по наследству». Это же касается и хранения усредненных данных для АЦП 7, а также некоторых других операций, которые в этой программе можно было бы опустить.

Далее действия повторяются для усредненных значений результатов преобразований АЦП 7, которые также преобразуются в 4-разрядное десятичное число и записывается в 4 ячейки ОЗУ перед ячейками, хранящими

шестнадцатеричное представление результатов. Десятичное число должно быть отражено на индикаторе как показание текущей температуры ПечиН.

Команды *pop* восстанавливают значения переменных *XH*, *XL*, *th*, *tl* из стека. Восстановление значений требовалось для правильной работы исходной программы.

Замечание. Обратите внимание на последовательность команд *push* и *pop* — значение, посланное в стек последним, извлекается первым. Вспомните, как работает указатель стека: он всегда указывает на последнее введенное значение. В данном фрагменте после выполнения команды *push XH* указатель стека зафиксирован на ячейке, хранящей значение *XH*.
Далее выполняется подпрограмма *hex_dc*, и в стек заносится два байта адреса возврата из подпрограммы. Из подпрограммы *hex_dc* в свою очередь вызывается подпрограмма *count*, адрес возврата из нее также заносится в стек, а указатель стека теперь находится на 4 ячейки левее ячейки со значением *XH*.
Затем происходит возврат из подпрограммы *count*, указатель перемещается на 2 ячейки вправо, затем возврат из подпрограммы *hex_dc* и указатель возвращается к ячейке со значением *XH*.
Далее снова выполняется еще одно обращение к подпрограмме *hex_dc*, после возврата из нее указатель снова на ячейке со значением *XH*.
После выполнения команды *pop XH* указатель находится на ячейке со значением *XL*.
Далее последовательность команд *pop* ясна.

Теперь в регистр Y заносится адрес 4-разрядного десятичного числа, пропорционального температуре ПечиВ, в регистр Z — адрес 4 ячеек памяти, в которые должен быть помещен соответствующий этому десятичному числу 7-сегментный код для отображения на индикаторе.

Здесь оказалось удобным помещать разряды чисел в ОЗУ не слева направо, а справа налево, поэтому в регистр на самом деле помещается адрес ячейки, следующей за последней ячейкой, содержащей старший разряд числа.

Затем вызывается подпрограмма *mDecod*, преобразующая 4 разряда десятичного числа в 4 разряда 7-сегментного кода и помещающая этот код в определяемую регистром Z область ОЗУ для непосредственного отображения на индикаторе.

Действия повторяются для десятичного результата АЦП 7, формируются еще 4 разряда 7-сегментного кода отображаемых на индикаторе данных.

В режимах установки $T_{ниж.}$ или $T_{верх.}$ выполняется другая часть подпрограммы, начинающаяся меткой *m110:*.

Предположим, устройство находится в режиме установки $T_{верх.}$, тогда в переменной *store* хранится адрес *eeTop*, а в *ZL* — адрес (sTH·2).

Замечание. Почему адрес sTH удваивается? Потому, что адрес метки воспринимается программой как адрес ячейки памяти программ размером в двухбайтное слово.

В регистр Y записывается адрес области памяти, в которую переносятся 7-сегментные коды символов для отображения на индикаторе. Здесь, как уже было раньше, оказалось удобным использовать адрес ячейки, следующей за этой областью.

В переменную *tmp*, используемую в качестве счетчика, заносится число 8, а цикл считывания байта данных из памяти программ, начинающийся меткой *copyStp:*, выполняется 8 раз.

Первая команда *lpm* этого цикла извлекает из ячейки памяти программ, адрес которой хранится в регистре Z, один байт данных и помещает его в регистр R0. R0,Z — это как бы скрытые параметры команды, которые невозможно изменить.

Следующей командой (st – Y,R0) предварительно уменьшается на единицу адрес ячейки, хранящийся в регистре Y, и только после этого в ячейку помещается содержимое регистра R0, содержимое регистра Z увеличивается на единицу, указывая на следующую ячейку в памяти программ.

Замечание. Когда мы вычисляли адрес метки, он удваивался, так как метка указывает на двухбайтную команду или два байта данных (константы, обозначенные метками *sTH:* и *sTL:* предварялись директивой *.db* и вводились по два байта). Сейчас же мы оперируем с однобайтными ячейками.

Декрементируется содержимое счетчика *tmp*, цикл повторяется, пока все 8 констант не будут считаны. Первые три константы представляют собой 7-сегментные коды для отображения трех символов «В°С» на индикаторе слева, еще 5 констант — это нули, они погасили бы оставшиеся 5 разрядов индикатора. Но 4 разряда заполнятся корректируемым числом, поэтому не светящимся окажется один разряд. Еще один разряд не подключен.

В переменную *tmp* заносится старший байт адреса *eeBot*. Мы загружаем параметр для ПечиВ, находящийся в EEPROM по адресу *eeTop*, однако адреса *eeTop* и *eeBot* выбраны так, что старшая их часть одинакова.

Теперь пара переменных *tmp:store* хранит адрес *eeTop*, далее парой команд *out* этот адрес передается в пару регистров EEARH:EEARL. В регистр Y записывается адрес *NUM*, затем вызывается подпрограмма *ReadEE*, считывающая из EEPROM четыре десятичные цифры, представляющие значение температуры, которая должна поддерживаться ПечьюВ. Извлеченные цифры записываются в ячейки ОЗУ, предшествующие ячейке с адресом *NUM*.

По окончании подпрограммы *ReadEE* в регистр Y снова записывается адрес *NUM*, а в регистр Z — адрес ячеек ОЗУ, в которые надо поместить 7-сегментные коды, соответствующие четырем извлеченным из EEPROM цифрам, для отображения на индикаторе, после этого вызывается подпрограмма *mDecod*, преобразующая десятичные цифры в 7-сегментные коды.

Подпрограмма *hex_dc* преобразовывает шестнадцатеричное число в 4-разрядное десятичное число. Перед вызовом подпрограммы в регистр Y заносится адрес шестнадцатеричного числа, двумя командами *ld* это число извлекается из ячеек ОЗУ в пару регистров XH:XL, в содержимое *th:tl* помещается число 1000 и вызывается подпрограмма *count*, которая вычитает из содержимого регистров XH:XL содержимое *th:tl*, подсчитывая число вычитаний до тех пор, пока результат не станет меньше нуля. Тогда к результату прибавляется содержимое *th:tl*.

Заметим, что команда *ldi* не может заносить константу в переменную *tl*, так как этой переменной назначен регистр R6, а команда работает только с регистрами R16...R31.

Поскольку мы вычитали число 1000, то число вычитаний — это количество тысяч в преобразуемом числе, оно записывается в ячейку ОЗУ. Результат вычитаний числа 1000 и последнего его прибавления сейчас лежит в диапазоне 0...999. Теперь, аналогичным образом загружая в содержимое *th:tl* число 100, подсчитывается и заносится в ОЗУ количество сотен в преобразуемом числе.

Так же подсчитываются десятки и единицы. В результате в 4 ячейках ОЗУ хранится 4 разряда десятичного числа, соответствующего преобразуемому шестнадцатеричному.

Замечание. Работа команд *ld* и *st* с предварительным уменьшением адреса (например, *st – Z, temp*) и с последующим увеличением адреса (например, *st Y +, temp*) подробно рассматривалась ранее. Поэтому, пользуясь листингом программы, легко проследить, из каких ячеек считывались данные, в какие ячейки записывался результат.

При работе с программой полезно изобразить карту распределения ОЗУ. На ней можно отобразить размещение переменных, имена констант, стек. Это поможет избежать перекрывания данных или, например, их уничтожения стеком. Такую карту удобно хранить вместе с блок-схемой программы, если вы ее делаете. Это значительно облегчает понимание программы, которую вы когда-то написали.

Посмотрим, какое количество ячеек в конце ОЗУ займет стек при обращении к подпрограмме *mInd*. При вызове подпрограммы *mInd* в стек заносится 2-байтный адрес возврата, 4 команды *push* занимают еще 4 ячейки, адреса возврата из подпрограммы *hex_dc* и *count* займут еще 4 ячейки, итого 10 ячеек в конце ОЗУ микроконтроллера.

При этом следует искать самое узкое место в программе с наибольшим числом вызовов вложенных подпрограмм и команд *push*, следует также учитывать, могут ли при этом происходить аппаратные прерывания, которые также заносят адрес возврата и могут также содержать вложенные подпрограммы или сами аппаратные прерывания могут быть вложенными, как в нашей программе: прерывание АЦП выполняется, когда выполняется обработчик прерывания таймера T0.

Блок 7. Подпрограммы WriteEE, ReadEE записи и чтения EEPROM

```
;Чтение EEPROM
ReadEE:
        ldi     tmp,4
loop2:  rcall   EE_Rd                   ;ПОЛУЧИТЬ данные из EEPROM
        st      - Y,EEbt                ;СОХРАНИТЬ в SRAM
        dec     tmp                     ;Конец?
        brne    loop2                   ;Нет - еще раз
        ldi     tmp,0
        out     EECR,tmp
        ret
EE_Rd:
        in      EEaH,EEARH
        in      EEaL,EEARL
        adiw    EEaL,1                  ;УВЕЛИЧИТЬ адрес
        out     EEARH,EEaH              ;ОТПРАВИТЬ его
        out     EEARL,EEaL
        sbi     EECR,EERE               ;EEPROM Read строб
                                        ;занимает 4 цикла микроконтроллера
        sbi     EECR,EERE               ;EEPROM Read строб еще раз
                                        ;занимает 4 цикла микроконтроллера
        in      EEbt,EEDR               ;ПРИНЯТЬ данные
        ret
;Конец (Чтение EEPROM)
;=====================
;Запись в EEPROM
WriteEE:
        ldi     YH,High(NUM)
        ldi     YL,Low(NUM)
        clr     tmp
        out     EECR,tmp
        in      EEaH,EEARH
        in      EEaL,EEARL
        sbiw    EEaL,4                  ;Установить адрес
        out     EEARH,EEaH
        out     EEARL,EEaL
        ldi     tmp,4
loop1:
        ld      EEbt, - Y
        rcall   EE_Wr                   ;EEPROM <- данные
        dec     tmp                     ;КОНЕЦ?
```

```
        brne   loop1              ;Нет-еще раз
        ldi    tmp,0
        out    EECR,tmp
        ret
EE_Wr:
        in     EEaH,EEARH
        in     EEaL,EEARL
        adiw   EEaL,1             ;УВЕЛИЧИТЬ адрес
EE_wr1:
        sbic   EECR,EEWE
        rjmp   EE_wr1
        out    EEARH,EEaH         ;ВЫВЕСТИ адрес
        out    EEARL,EEaL
        out    EEDR,EEbt
        sbi    EECR,EEMWE
        sbi    EECR,EEWE
        ret
;Конец (Запись в EEPROM)
;======================
```

За основу подпрограмм записи и считывания EEPROM взят пример avr100.asm, который можно разыскать на сайте корпорации «Atmel».

Подпрограмма *ReadEE* считывает из EEPROM в ОЗУ микроконтроллера 4 байта данных, соответствующих 4 разрядам десятичного значения температуры, которую должна поддерживать одна из печей. В зависимости от того, для какой из печей извлекаются данные, перед вызовом подпрограммы в пару регистров EEARH:EEARL записывается адрес EEPROM, по которому находится необходимый параметр, а в регистр Y — адрес ОЗУ, по которому этот параметр надо разместить.

Регистр *tmp* используется в качестве счетчика циклов, а сама подпрограмма, а также вызываемая из нее подпрограмма *EE_Rd* выполняют алгоритм считывания данных из EEPROM, предлагаемый в описании микроконтроллера и в примере avr100.asm.

Подпрограмма *WriteEE* и вызываемая из нее подпрограмма *EE_Wr* обеспечивают запись 4 байт данных в EEPROM, причем адрес записи для подпрограммы остается после выполнения подпрограммы *ReadEE*, поскольку делается запись того же параметра, вызванного подпрограммой *ReadEE*, только откорректированного при нажатии кнопок «УВЕЛИЧИТЬ» или «УМЕНЬШИТЬ».

Блок 8. Подпрограмма mDecod преобразования цифр в 7-сегментный код и подпрограмма mheat регулировки нагрева печей

```
;Преобразование 4 цифр в 7-сегментный код
mDecod:
        ldi    tmp,4
mDec_1:
        ld     store, - Y
```



```
        tst    store
        brne   mDc9
        ldi    store,$bb
mDc9:   cpi    store,9            ;Не изменять порядок!
        brne   mDc8
        ldi    store,$af
mDc8:   cpi    store,8
        brne   mDc7
        ldi    store,$bf
mDc7:   cpi    store,7
        brne   mDc6
        ldi    store,$a2
mDc6:   cpi    store,6
        brne   mDc5
        ldi    store,$bd
mDc5:   cpi    store,5
        brne   mDc4
        ldi    store,$ad
mDc4:   cpi    store,4
        brne   mDc3
        ldi    store,$87
mDc3:   cpi    store,3
        brne   mDc2
        ldi    store,$ae
mDc2:   cpi    store,2
        brne   mDc1
        ldi    store,$3e
mDc1:   cpi    store,1
        brne   mDc
        ldi    store,$82
mDc:    st     - Z,store
        dec    tmp
        brne   mDec_1
        ret
;Конец (Преобразование цифры в 7-сегментный код)
;Регулировка нагрева печей
mheat:  ldi    tmp,high(eeBot)
        out    EEARH,tmp
        in     tmp,EEARL
        push   tmp
        push   YL
        push   YH
        push   ZL
        push   ZH
        ldi    YL,low(C_E)
        ldi    YH,high(C_E)
        ldi    tmp,low(eeTop)
        out    EEARL,tmp
        rcall  ReadEE
        rcall  trcod              ;timh:tm2 - код для нагревателей
        mov    XH,timh            ;timh,XL = r26
```

```
        mov     XL,tm2
        ldi     YL,low(ADC6 - 2)
        ldi     YH,high(ADC6 - 2)
        ld      tm2,Y +
        ld      tm1,Y
        adiw    XL,cTAdd
        cp      XL,tm1
        cpc     XH,tm2
        brcc    mh1
        cbi     PORTB,4
mh1:    ldi     tmp,low(cTAdd + cTSub)
        ldi     store,high(cTAdd + cTSub)
        sub     XL,tmp
        sbc     XH,store
        cp      tm1,XL
        cpc     tm2,XH
        brcc    mh3
        sbi     PORTB,4
mh3:    ldi     YL,low(C_E)
        ldi     YH,high(C_E)
        ldi     tmp,low(eeBot)
        out     EEARL,tmp
        rcall   ReadEE
        rcall   trcod                    ;timh:tm2 – код для нагревателей
        mov     XH,timh                  ;timh,XL = r26
        mov     XL,tm2
        ldi     YL,low(ADC7 - 2)
        ldi     YH,high(ADC7 - 2)
        ld      tm2,Y +
        ld      tm1,Y
        adiw    XL,cTAdd
        cp      XL,tm1
        cpc     XH,tm2
        brcc    mh5
        cbi     PORTB,5
mh5:    ldi     tmp,low(cTAdd + cTSub)
        ldi     store,high(cTAdd + cTSub)
        sub     XL,tmp
        sbc     XH,store
        cp      tm1,XL
        cpc     tm2,XH
        brcc    mh7
        sbi     PORTB,5
mh7:    pop     ZH
        pop     ZL
        pop     YH
        pop     YL
pop     tmp
out     EEARL,tmp
        ret
;Конец (Регулировка нагрева печей)
```


Подпрограмма *mDecod* извлекает 4 цифры преобразуемого десятичного числа из ячеек ОЗУ, адрес которых определяется регистром Y, преобразует их в 7-сегментный код для отображения этих цифр на индикаторе и помещает полученный код в 4 ячейки ОЗУ, адрес которых определяется регистром Z. Поэтому перед вызовом подпрограммы *mDecod* инициализируются регистры Y и Z.

В подпрограмме переменная *tmp* используется в качестве счетчика, в нее загружается число 4, равное количеству разрядов преобразуемого десятичного числа. В переменную *store* помещается первая цифра, далее последовательно проверяется, равно ли значение переменной store числам 0, 9, 8, ..., 1.

Если значение переменной *store* равно, например, 7, то в саму переменную *store* записывается 7-сегментный код цифры 7, равный $a2.

Использование переменной *store* как для хранения десятичной цифры, так и для хранения кода возможно, поскольку ни один код не совпадает ни с одной десятичной цифрой. Следовательно, проверка содержимого переменной *store* после записи в нее кода пройдет успешно.

Определение 7-сегментного кода для десятичных цифр уже рассматривалось, а в **Табл. 4** (стр. 110) содержатся десятичные цифры и их 7-сегментные коды.

Полученный код для первой цифры помещается в ячейку ОЗУ, адрес которой извлекается из регистра Z.

Значение переменной *tmp* декрементируется и проверяется на равенство нулю. Пока это значение больше нуля, все действия повторяются, коды оставшихся трех десятичных цифр определяются аналогичным способом и помещаются в ОЗУ.

Подпрограмма *mheat* сравнивает текущие средние значения температур ПечиВ и ПечиН со значениями температур, которые должны поддерживаться этими печами (с заданными температурами). Если какая-либо печь остыла, на ее спираль подается напряжение, при перегреве печи спираль отключается.

Замечание. Если выключать печь, как только измеренная температура превысит заданную, а включать — когда она станет меньше заданной, то коммутация спирали будет происходить слишком часто. Обычно спирали печей выходили из строя при включении.
Опыт показывает, что удержание температуры в некотором диапазоне не сказывается на качестве выпекаемого печенья.
Поэтому в рассматриваемой подпрограмме выключение спирали производится, когда текущая температура печи превышает заданную температуру на 1°C, а включение — когда температура падает на 1°C ниже заданной температуры.

В подпрограмме *mheat* в регистр EEARH загружается старший байт адреса *eeBot* (старшие байты адресов *eeBot* и *eeTop* одинаковы).

В стек командами *push* помещаются значения, хранившиеся в регистрах EEARL, YL, YH, ZL и ZH. Содержимое этих регистров изменяется подпрограммой *mheat*, но значения, хранившиеся в них перед вызовом подпрограммы, используются в других частях программы, поэтому они сохраняются в стеке, а перед окончанием подпрограммы восстанавливаются.

В регистр Y записывается адрес *C_E*, а в регистр EEARL — младший байт адреса области EEPROM, хранящей заданную температуру ПечиB.

Подпрограмма *ReadEE* считывает из EEPROM микроконтроллера 4-разрядное десятичное значение заданной температуры ПечиB и помещает его в ячейки ОЗУ, адрес которых определяется содержимым регистра Y.

Следующая вызываемая подпрограмма *trcod* преобразует это 4-разрядное десятичное значение в шестнадцатеричное число и сохраняет его в регистрах *timh:tm2*.

Затем командами *mov* шестнадцатеричное значение температуры передается в пару регистров XH:XL. В регистр Y загружается адрес области ОЗУ, хранящей шестнадцатеричное текущее значение температуры, до которой нагрелась ПечьB. Это значение температуры, полученное в результате работы АЦП 6, загружается в регистры *tm2:tm1*.

К значению заданной температуры, хранящейся в регистре X, командой *adiw* добавляется константа *cTAdd* (она соответствует одному градусу, теперь в регистре X содержится увеличенное на 1°C значение заданной температуры).

Содержимое регистров XH:XL сравнивается с содержимым *tm2:tm1*. Если первое из них меньше, это значит, что текущая температура выше увеличенной на 1°C заданной температуры. Следовательно, ПечьB перегрета, на линии PB4 командой *cbi* устанавливается НИЗКИЙ уровень, это вызывает отключение спирали ПечиB. В нашей схеме к этой линии через резистор R1 подключен светодиод VD1, который в указанном случае перестанет светиться.

С метки *mh1:* по метку *mh3:* подпрограмма проверяет, не слишком ли остыла ПечьB.

Двумя командами *ldi* в регистры *store* и *tmp* загружается сумма констант *cTAdd* и *cTSub*. Эта сумма вычитается из содержимого регистров XH:XL. Если до этого в регистре X хранилось увеличенное на *cTAdd* (1°C) значение заданной температуры, то после вычитания там стало храниться уменьшенное на *cTSub* (также 1°C) значение.

Теперь значение текущей температуры, хранящееся в регистрах *tm2:tm1*, сравнивается с уменьшенным на градус значением заданной температуры ПечиB, если первое значение ниже второго — ПечьB остыла, на

линии PB4 устанавливается ВЫСОКИЙ уровень, вызывающий подачу напряжения на спираль ПечиB. В нашей схеме загорится светодиод VD1.

Далее с метки *mh3:* по метку *mh5:* выполняются точно такие же сравнения текущей температуры, полученной в результате работы АЦП 7 для ПечиH с заданной температурой для этой печи. Управление спиралью ПечиH производится по линии PB5 микроконтроллера. В нашей схеме к этой линии через резистор R2 подключен светодиод VD2, имитирующий работу спирали.

Перед завершением подпрограммы из стека извлекаются значения переменных ZH, ZL, YH, YL и EEARL, хранившиеся в них перед вызовом подпрограммы.

Блок 9. Подпрограмма trcod преобразования десятичного числа в шестнадцатеричное и подпрограмма avrT усреднения результатов преобразования АЦП 6 и АЦП 7

```
;Преобразование десятичного кода в шестнадцатеричный результат в timh:timl
trcod: ldi    YL,low(C_E)
       ldi    YH,high(C_E)
       ld     store, - Y
       ldi    tmp,4 - 1
       mov    dgtnum,tmp
       clr    timh
nxdgt: clr    tm2
       rcall  mul10
       ld     store, - Y
       add    store,tm2
       brcc   noinc
       inc    timh
noinc: dec    dgtnum
       tst    dgtnum
       brne   nxdgt
       mov    tm2,store
       ret
mul10: clr    tm1
       ldi    tmp,10
ad10:  add    tm2,store
       brcc   nxad
       inc    tm1
nxad:  add    tm1,timh
       dec    tmp
       brne   ad10
       mov    timh,tm1
       ret
;Конец (Преобразование десятичного кода в шестнадцатеричный)
;Накопление - усреднение данных АЦП
avrT:  tst    nADC
       brne   avr3
```

```
        ldi     tmp,5
        mov     cntr,tmp
        ldi     YL,low(sum6)
        ldi     YH,high(sum6)
        ld      store,Y +
        ld      tmp,Y
avr1:   lsr     store                 ;СУММА ADC6/64
        ror     tmp
        dec     cntr
        brne    avr1
        ldi     YL,low(cShft)
        ldi     YH,high(cShft)
        add     tmp,YL
        adc     store,YH
        ldi     YL,low(ADC6)
        ldi     YH,high(ADC6)
        st      - Y,tmp
        st      - Y,store
        ldi     tmp,5
        mov     cntr,tmp
        ldi     YL,low(sum7)
        ldi     YH,high(sum7)
        ld      store,Y +
        ld      tmp,Y
avr2:   lsr     store                 ;Сумма ADC7/64
        ror     tmp
        dec     cntr
        brne    avr2
        ldi     YL,low(cShft)
        ldi     YH,high(cShft)
        add     tmp,YL
        adc     store,YH
        ldi     YL,low(ADC7)
        ldi     YH,high(ADC7)
        st      - Y,tmp
        st      - Y,store
        clr     tmp
        ldi     YL,low(sum6)
        ldi     YH,high(sum6)
        st      Y +,tmp
        st      Y,tmp
        ldi     YL,low(sum7)
        ldi     YH,high(sum7)
        st      Y +,tmp
        st      Y,tmp
        ldi     tmp,64
        mov     nADC,tmp
avr3:   dec     nADC
        ldi     YL,low(AD6)
        ldi     YH,high(AD6)
        ld      store,Y +
```



```
        ld      tmp,Y
        ldi     YL,low(sum6)
        ldi     YH,high(sum6)
        ld      cntr,Y +
        ld      tm1,Y
        add     tmp,tm1
        adc     store,cntr
        sbiw    YL,1
        st      Y +,store
        st      Y,tmp
        ldi     YL,low(AD7)
        ldi     YH,high(AD7)
        ld      store,Y +
        ld      tmp,Y
        ldi     YL,low(sum7)
        ldi     YH,high(sum7)
        ld      cntr,Y +
        ld      tm1,Y
        add     tmp,tm1
        adc     store,cntr
        sbiw    YL,1
        st      Y +,store
        st      Y,tmp
        ret
;Конец (Накопление – усреднение данных АЦП)
```

Подпрограмма *trcod* преобразует 4-разрядное десятичное число в шестнадцатеричное число и помещает его в ОЗУ. Для преобразования будем вычислять результат каждого действия как двухбайтное шестнадцатеричное число, сами действия — операции над шестнадцатеричными числами.

Преобразование состоит в следующем:

- количество тысяч в десятичном числе умножим на десять;
- прибавим количество сотен;
- результат умножим на десять;
- прибавим количество десятков;
- снова умножим на десять;
- прибавим число единиц — получим шестнадцатеричный результат.

В подпрограмме две команды *ldi* загружают в регистр Y хранимый константой *C_E* адрес ячеек ОЗУ, в которых находится десятичное число.

Командой *ld store, – Y* находящийся в регистре адрес предварительно декрементируется, после чего регистр Y указывает на 4-й (старший) разряд десятичного числа. Поэтому в переменную *store* загружается количество тысяч преобразуемого числа. Переменная *dgtnum* выполняет функцию счетчика числа циклов, а поскольку команда *ldi* может загружать константы только в регистры r16...r31, а переменная *dgtnum* хранится в регистре r8, то в качестве посредника используется переменная *tmp*.

Переменные *timh* и *tm2*, в которых будет накапливаться шестнадцатеричный результат преобразования, очищаются, после чего вызывается подпрограмма *mul10*, производящая умножение на десять.

В подпрограмме *mul10* к содержимому *tm1:tm2* десять раз прибавляется содержимое *timh:store* (сейчас это количество тысяч в десятичном числе).

В подпрограмме *mul10* переменная *tmp* служит счетчиком, в нее загружается число циклов, равное 10.

Если при сложении содержимого переменных *tm2* и *store* происходит переполнение, к переменной *tm1* добавляется единица.

По окончании десяти циклов содержимое переменной *tm1* передается в переменную *timh*.

После возврата из подпрограммы *mul10* в подпрограмму *trcod* в переменную *store* загружается количество сотен в преобразуемом десятичном числе (3-й разряд числа), к этому количеству прибавляется результат умножения количества тысяч на десять и снова вызывается подпрограмма умножения на десять.

Действия повторяются для количества десятков в десятичном числе (2-й разряд). К результату прибавляются единицы преобразуемого числа, а сумма, старший байт которой находится в переменной *timh*, а младший — в переменной *tm2*, представляет собой шестнадцатеричный результат преобразования.

Подпрограмма *avrT* вызывается при свечении последнего разряда индикатора и усредняет результаты преобразования АЦП 6 и АЦП 7. В течение 64 обращений к подпрограмме происходит накопление результатов, то есть суммируется 64 результата преобразования АЦП 6, а затем делятся на 64. Те же действия производятся над результатами преобразования АЦП 7.

Поскольку на индикацию одной цифры в программе приходится одно прерывание таймера T0, вызывающее выполнение преобразования АЦП 6 и АЦП 7, то на индикацию всех восьми цифр — 8 преобразований АЦП. Но подпрограмма *avrT* вызывается только при индикации последней цифры, это значит, что в программе 7 результатов преобразований из восьми не используются. Как было показано в разделе 0, время индикации одной цифры составляет 1083 мкс, время индикации восьми цифр 8.664 мс, усреднение результатов преобразования АЦП составляет 554.496 мс (64×8.664).

Переменная *nADC* хранит число обращений к подпрограмме *avrT*, которое осталось до завершения накопления результатов преобразования АЦП. При каждом обращении к подпрограмме это число декрементируется, и, когда оно станет равным нулю, накопленная сумма будет разделена на 64, переменные, хранящие накопленный результат, будут очищены, а в переменную *nADC* снова будет записано число 64 для обеспечения нового накопления.

В подпрограмме значение переменной *nADC* тестируется, если оно равно нулю, значит, накопление завершено, а результат надо разделить на 64.

Число 64 равно 2^6, а деление двоичного числа на 2 выполняется посредством его сдвига вправо на один двоичный разряд. Значит, для деления на 64 надо шесть раз сдвинуть накопленную сумму вправо.

В связи с тем, что температура печей представляет интерес только в диапазоне 100...300°С, а результат преобразования АЦП может лежать только в диапазоне 0...1023, необходимо привязать показания АЦП к температуре печей. При индикации в нашем устройстве не используется десятичная запятая, поэтому при 100°С на индикаторе отображается число 1000, при 300°С — число 3000. В подпрограмме *avrT* выполняется дополнительное масштабирование сигнала: результат усреднения удваивается, поэтому результат делится не на 64, а на 32, соответственно необходимое количество сдвигов результата вправо равно не 6, а 5, а к полученному среднему значению добавляется константа *cShft*, равная тысяче.

Таким образом, на индикаторе средние значения могут отображаться в интервале от $0 \cdot 2 + 1000$ до $1023 \cdot 2 + 1000$ или от 1000 до 3046. При должной настройке предварительных усилителей сигналов термопар эти показатели будут соответствовать температурам от 100 до 304.6°С.

В переменную *cntr*, являющуюся счетчиком циклов, с помощью переменной *tmp* загружается число пять.

В регистр Y загружается адрес накопленной суммы 64 преобразований АЦП 6, в пару переменных *store* и *tmp* загружаются сами суммы. Командами *lsr* и *ror* содержимое переменных сдвигается вправо.

Команда *lsr* сдвигает старший байт суммы, старший разряд сдвигаемого байта очищается, а младший разряд перемещается во флаг C регистра флагов. Команда *ror* сдвигает младший байт суммы, при этом содержимое флага C помещается в освобождающийся старший разряд, а младший разряд также помещается во флаг C регистра флагов.

Замечание. Используя цепочку команд (*lsr — ror — ... — ror*), можно сдвигать вправо многобайтные числа. Командой *lsr* должен сдвигаться самый старший байт, остальные байты сдвигаются командами *ror*.

После сдвига двухбайтного числа содержимое счетчика декрементируется, также выполняются оставшиеся 4 цикла сдвигов, пока содержимое счетчика циклов *cntr* не станет равным нулю. В переменных *store:tmp* теперь хранится удвоенное среднее значение последних 64 циклов преобразования АЦП 6.

В регистр Y помещается константа *cShft*, равная тысяче. Содержимое регистра Y добавляется к удвоенному среднему значению АЦП 6. Результат помещается в ОЗУ по адресу *ADC6*.

Все перечисленные действия повторяются для суммы, накопленной в результате 64 циклов преобразования АЦП 7. Сдвинутое на тысячу удвоенное среднее значение АЦП 7 сохраняется в ОЗУ микроконтроллера по адресу *ADC7*.

Затем ячейки ОЗУ, в которых накапливаются суммы результатов преобразований АЦП 6 и АЦП 7, очищаются, а в переменную *nADC* записывается число 64.

Меткой *avr3:* начинается часть подпрограммы *avrT*, выполняющаяся при каждом обращении к подпрограмме. Содержимое переменной *nADC* декрементируется.

В переменные *store:tmp* из ячеек ОЗУ микроконтроллера с адресом *AD6* загружается результат последнего преобразования АЦП 6, в переменные *cntr:tm1* из ячеек ОЗУ с адресом *sum6* загружается накапливаемая сумма результатов преобразования АЦП 6. Содержимое переменных *store:tmp* складывается с содержимым переменных *cntr:tm1*, результат помещается в ячейки ОЗУ по адресу *sum6*.

Описанные в последнем абзаце действия повторяются с результатами последнего преобразования АЦП 7 и накапливаемой суммой результатов преобразования АЦП 7. После приращения сумма помещается в ячейки ОЗУ по адресу *sum7*.

4.3.3. Работа с устройством управления двумя печами

Текст описанной программы надо записать в окне ассемблируемого файла *.asm пакета AVRStudio блок за блоком. После ассемблирования в окне Project Output выводится размер кода, который должен составлять 620 слов, и размер констант, составляющий 8 слов.

После загрузки программы в микроконтроллер устройство вполне работоспособно, но при переходе в режимы изменения температур ПечиВ и ПечиН на индикаторе появляются строки с нечитаемым значением температуры.

Строка, индицируемая в режиме установки температуры ПечиВ (верхней печи): «В°С 8.8.8.8.».

Для ПечиН (нижней печи) строка имеет вид: «Н°С 8.8.8.8.».

В результате стирания, обязательно выполняющегося перед записью программы в память программ микроконтроллера, во всех ячейках как памяти программ, так и EEPROM оказываются значения $FF.

Значение корректируемой температуры извлекается из EEPROM микроконтроллера, а свечение восьмерок с точками в 4 разрядах вызвано тем, что на все сегменты индикатора при подсвечивании этих разрядов подаются ВЫСОКИЕ уровни, так как извлеченные из EEPROM значения $FF = 0b11111111 (во всех восьми разрядах извлеченного байта единицы).

Можно нажать кнопку «УВЕЛИЧИТЬ» и удерживать ее, пока во всех разрядах не появятся цифры от нуля до девяти. Более быстрый способ — попеременное нажатие кнопок «УВЕЛИЧИТЬ» и «УМЕНЬШИТЬ». Тогда после трех-четырех переходов младшего разряда через ноль во всех разрядах окажутся цифры от нуля до девяти. Самый правильный способ — записать необходимые данные в EEPROM с помощью программатора.

Для начала нужно создать файл данных *.hex. Если установленное на компьютере программное обеспечение программатора обеспечивает редактирование данных EEPROM, то данные можно внести прямо в соответствующем окне.

На **Рис. 28** представлен фрагмент окна программы «ATMEL» AVR ISP.

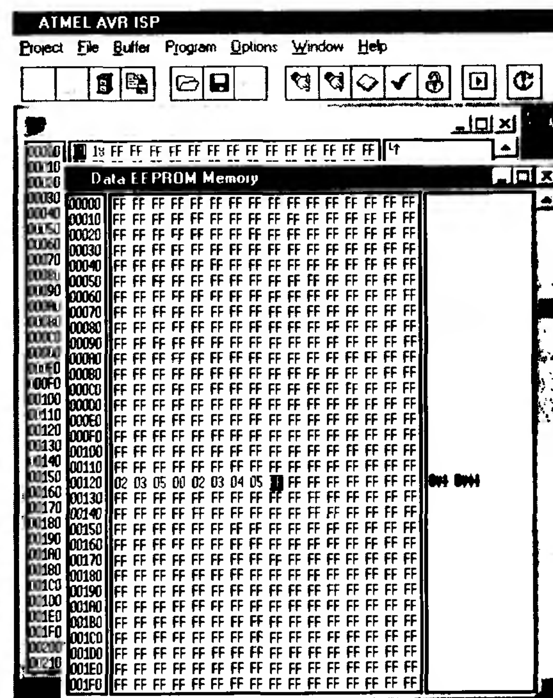


Рис. 28. Окно программы «ATMEL» AVR ISP

На переднем плане расположено окно Data EEPROM Memory, в колонке слева — адреса первых ячеек каждой строки. В нашей программе значение температуры, до которой должна нагреваться ПечьВ, хранится в EEPROM по адресу $120 (*eeTop*), температура, до которой должна нагре-

ваться ПечьН, — по адресу $124 (*eeBot*). Каждое значение температуры представлено 4 десятичными разрядами.

В строке, адрес ячеек которой начинается с $120, первые четыре байта 02 03 05 00, хранящиеся в ячейках $120...$123, представляют число 2350, соответствующее температуре 235°С, до которой должна нагреваться ПечьВ. Следующие четыре байта 02 03 04 05, хранящиеся в ячейках $124...$127, представляют число 2345, соответствующее температуре 234.5°С для ПечиН.

Ввод этих чисел в окно производится вручную, программа обеспечивает сохранение набранных данных EEPROM в виде файла.

Для записи данных непосредственно в EEPROM микроконтроллера в меню программы AVR ISP выбирается Program/Program EEPROM.

Если используется программное обеспечение программатора, не обеспечивающее создание и корректировку файла данных для EEPROM, можно воспользоваться окном Memory программы AVR Studio. Для вызова этого окна надо перейти в режим отладки любой программы. Для этого надо ассемблировать какую-нибудь программу (это может быть программа с единственной командой, например *wdr*) и перейти в режим пошаговой отладки (F11). В появившемся окне Simulator Options выберите тот микроконтроллер, в EEPROM которого необходимо записать данные, иначе размер EEPROM может оказаться не соответствующим микроконтроллеру и вы просто не найдете в окне Memory/EEPROM адреса $120.

Теперь откройте окно Memory и в появившемся на экране изображении в окошке «Выбор типа» вместо просмотра памяти Data (ОЗУ) выберите просмотр памяти EEPROM.

Ввод данных ничем не отличается от описанного выше ввода в окне программы «ATMEL» AVR ISP.

Откройте меню Debug/Up/Download Memories и для сохранения данных в файл выберите тип памяти EEPROM. Введите или выберите имя файла *.hex, под которым его надо сохранить и путь к этому файлу. Сохраните файл, нажав кнопку Save to File.

Когда будете программировать микроконтроллер, данные, хранящиеся именно в этом файле, надо будет загружать в EEPROM микроконтроллера.

4.3.4. Особенности работы EEPROM микроконтроллера

EEPROM микроконтроллера способна длительно хранить данные при отсутствии напряжения питания. Однако при включении/выключении питания иногда наблюдается частичная потеря данных, хранящихся в этой памяти.

Для борьбы с таким явлением рекомендуется использовать встроенный в микроконтроллер Brown-детектор, вызывающий аппаратный сброс микроконтроллера при уровне напряжения питания ниже допустимого.

В работе можно столкнуться с эффектом утраты данных в EEPROM микроконтроллера. Наиболее часто эффект наблюдается при работе с миниатюрным импульсным блоком питания. При использовании лабораторного блока питания эффект утраты данных почти не проявлялся. Увеличение емкости конденсатора *RC*-цепочки, подключаемой к контакту RESET микроконтроллера, не изменило ситуацию. Потери данных резко сокращались, когда сначала включался блок питания, а затем к нему подключался контроллер, а при выключении — сначала отключался контроллер, а затем выключался блок питания.

Несколько сократить потери данных удалось, разделив спаренный выключатель, имевшийся в блоке. Одна пара его контактов размыкалась раньше другой. После доработки через первую пару стало подаваться питание на контроллер, через вторую — напряжение сети 220 В. Теперь при выключении сначала отключалось питание контроллера, затем отключалось напряжение 220 В. Конечно, невозможно гарантировать, что через какое-то время за счет износа выключателя последовательность отключения не изменится.

Проверка показала, что при включении/выключении чаще искажались данные, корректировавшиеся последними. В контроллере станка для выпекания печенья кроме значений температур двух печей в EEPROM хранится еще 4 параметра времени. И еще: обычно изменялся один параметр, очень редко — два. Это позволило сделать предположение, что утрачиваются данные, адрес которых остался в регистрах EEARH:EEARL.

Следующим шагом было изменение адреса EEPROM после записи и считывания на адрес области, не содержащей данных.

Поскольку допустимая температура нагрева печей в нашем примере находится в диапазоне 200...300°С, после запуска программы считывались параметры из EEPROM. Для каждого параметра температуры производилась проверка, лежат ли величины всех четырех считанных байтов в диапазоне 0...9 и равно ли значение 1-го из них двум или трем. Если нет — в EEPROM записывались данные, соответствующие температуре 230°С без участия оператора. Аналогичные проверки производились для параметров времени.

Еще одна возможность — мажоритирование. Одни и те же параметры записываются в разные области EEPROM, при запуске программы извлекаются и сравниваются данные из разных областей, соответствующие одному параметру. Параметру присваивается наиболее часто встречающееся значение, а выбранное таким образом значение записывается во все облас-

ти, заменяя испорченные данные. При корректировке параметра и записи его оператором все хранящиеся экземпляры параметра также должны автоматически корректироваться.

Замечание. Считывать все экземпляры параметров и выполнять мажоритирование надо только после запуска программы. В процессе работы в этом нет необходимости, так как утрата данных наблюдается только при включении/выключении и, вероятно, при значительных перепадах напряжения питания. В приведенной программе перечисленные выше методы не отражены.
Параметры, не изменяемые оператором, старайтесь хранить в памяти программ. Организовать хранение и обращение к ним можно так же, как это сделано в приведенной программе (константы для отображения на индикаторе символов «B°C» и «H°C» начинаются метками *sTH:* и *sTL:*).

Глава 5. Связь микроконтроллера с компьютером

Еще одна задача программирования микроконтроллеров — обеспечение связи контроллера с компьютером. Такая связь позволяет решать вопросы управления контроллером, накопления данных контроллером в реальном времени с последующей передачей массивов данных в компьютер для хранения, обработки и отображения.

При решении таких задач у разработчиков, специализирующихся на применении микроконтроллеров, возникают трудности при проверке связи с компьютером.

В этой главе рассматриваются этапы разработки программ для микроконтроллера и компьютера, связь между которыми выполнена по каналу RS-232 в соответствии со стандартом RS-232.

Для поддержки связи по каналу RS-232 в используемом микроконтроллере предусмотрена аппаратная функция UART (Universal Asynchronous Receiver and Transmitter — универсальный асинхронный приемопередатчик). Компьютер для поддержки связи снабжен COM-портом. Поэтому для описания работы компьютера и микроконтроллера в режиме связи используется разная терминология (UART, RS-232, COM-порт).

5.1. Схема контроллера, обеспечивающая связь с COM-портом компьютера

Подробную информацию, связанную с передачей сигналов по каналу RS-232, легко найти как в литературе, так и в Internet, поэтому ограничимся минимальными сведениями.

Канал RS-232 предназначен для обмена информацией между двумя устройствами как в синхронном, так и в асинхронном режиме.

В компьютере для организации COM-порта используется только асинхронный способ передачи и лишь часть сигналов и функций, предусмотренных стандартом RS-232.

В микроконтроллерах AVR с UART аппаратно-программными средст вами поддерживаются только две линии: линия приема (RxD) и линия пе редачи данных (TxD).

Для связи компьютера с микроконтроллером мы используем тольк общий провод и две сигнальные линии. По одной из них данные переда ются от микроконтроллера к компьютеру, по другой — от компьютера микроконтроллеру.

Данные по каналу передаются последовательно, бит за битом. Инфор мация передается группами битов. Для выделения группы перед ее нача лом передается стартовый бит, по окончании передачи группы — один ил два стоповых бита, которым может предшествовать бит четности.

Описываемое здесь устройство передает стартовый бит, 8-битную груп пу (один байт), бит четности и один стоповый бит.

И передающее, и принимающее устройства должны работать на одина ковой скорости, которая может быть выбрана из предусмотренного стан- дартом ряда.

Логическая единица передается по каналу RS-232 уровнем напряже- ния −3...−12 В относительно общего провода, логический ноль — уровнем напряжения +3...+12 В. Указанные уровни напряжений приведены ко вхо- ду приемника сигнала.

Уровни сигналов в диапазоне −3...+3 В попадают в зону нечувстви- тельности и не воспринимаются приемником.

Компьютер снабжен необходимыми преобразователями уровней, и на контактах его COM-портов уровни сигналов соответствуют требованиям стандарта RS-232.

На линиях RxD и TxD микроконтроллера уровни сигналов соответст- вуют уровням TTL-логики, поэтому для подключения микроконтроллера к каналу RS-232 требуется преобразователь уровней.

На **Рис. 29** приведен фрагмент схемы контроллера, обеспечивающий подключение микроконтроллера к COM-порту компьютера.

Линия TxD микроконтроллера подключается к линии RxD компьюте- ра через преобразователь уровня, а линия TxD компьютера — к линии RxD микроконтроллера также через преобразователь уровня.

Подобные преобразователи уровней в интегральном исполнении вы- пускаются многими компаниями, например Analog Devices, Maxim.

Обычно в наименовании преобразователей присутствуют числа 232, 202, 242. Чаще встречаются микросхемы с двумя или четырьмя преобразо- вателями.

Микросхемы преобразователей могут работать с конденсаторами ем- костью 0.1 мкФ или с электролитическими конденсаторами емкостью 1 мкФ, обычно это определяется буквой, следующей за числом в наимено- вании микросхемы.

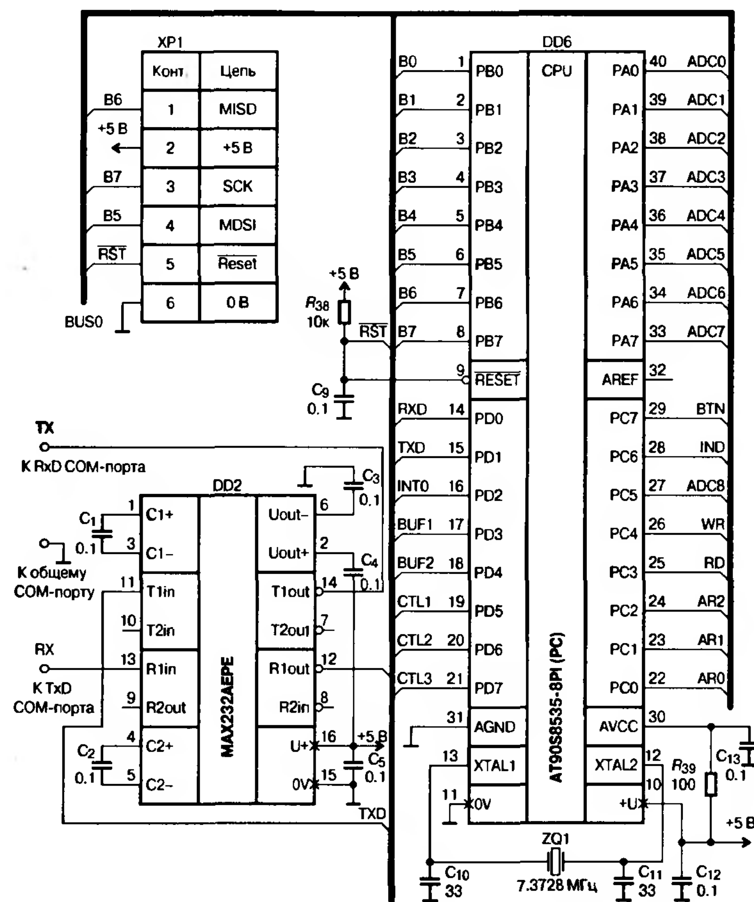


Рис. 29. Схема обеспечения связи микроконтроллера с компьютером

Выпускаются преобразователи, не требующие конденсаторов, их цена несколько выше.

Возможно изготовление преобразователя на дискретных элементах. Пример такой схемы можно найти на сайте корпорации «Atmel» в доку- ментации на последовательный программатор Avr910, управляемый ком- пьютером через COM-порт. Для организации преобразования уровней в этой схеме используется 2 транзистора, 2 диода, 6 резисторов и один элек- тролитический конденсатор.

В отличие от интегральных преобразователей, где уровни напряжений, соответствующие требованиям стандарта RS-232, вырабатываются самой микросхемой, питающейся от источника +5 В, в схеме программатора Avr910 используются уровни напряжений на контактах COM-порта компьютера.

На момент написания этой страницы описание программатора Avr910 находилось по адресу:
http://www.atmel.com/dyn/resources/prod_documents/DOC0943.PDF.

Адрес был найден поисковой системой сайта «Atmel» при вводе ключевого слова «avr910».

В компьютере для организации COM-порта используется либо 9-контактный (DB-9M), либо 25-контактный разъем (DB-25M). Поэтому для изготовления кабеля надо подобрать подходящий к разъему COM-порта вашего компьютера разъем DB-9F или DB-25F.

В **Табл. 6** приведены наименования и номера контактов этих разъемов, использующихся для подключения нашей схемы к COM-порту компьютера.

Таблица 6

Наименование контакта	Разъем DB-9F	Разъем DB-25F
TxD	3	2
RxD	2	3
Общий	5	7

Обратите внимание на частоту кварцевого резонатора ZQ1. Она равна 7.3728 МГц. Использование кварцевого резонатора с такой частотой позволяет точно устанавливать скорости передачи, рекомендованные стандартом RS-232. Подробнее этот вопрос будет рассматриваться в описании программы для микроконтроллера.

5.2. Программное обеспечение связи по каналу RS-232

Для того чтобы получить или передать данные по каналу RS-232, необходимо создать программы для обоих связываемых устройств. Значит, нужна программа для микроконтроллера и программа для компьютера.

Ниже рассматривается одна общая программа для микроконтроллера и два примера программ для компьютера.

Программы для компьютера написаны в Borland Delphi, а для обращения к COM-порту используется дополнительно установленный в Delphi компонент.

Прежде всего, надо решить, что, как и когда передавать.

Воспользуемся программами для микроконтроллера и компьютера, которые были разработаны для управления программатором микросхем флеш-памяти и электрически программируемых ПЗУ. Сами программы велики, нам понадобятся лишь их фрагменты, которые позволят проверить наличие связи по каналу RS-232, передать массив из компьютера в ОЗУ микроконтроллера, очистить ячейки ОЗУ, хранящие массив, заполнить эти ячейки значением, указанным в команде компьютера, и считать массив из ОЗУ микроконтроллера в компьютер. Все запросы будет генерировать компьютер, на каждый запрос микроконтроллер должен дать ответ.

5.2.1. Протокол обмена

Обычно программное обеспечение для компьютера и для микроконтроллера разрабатывают разные специалисты или группы специалистов.

Протокол обмена должен содержать всю необходимую информацию для разработки обеих программ так, чтобы независимо созданные программы, удовлетворяющие всем требованиям протокола, могли быть в кратчайший срок стыкованы друг с другом.

Поэтому в протоколе должны быть учтены все тонкости совместной работы программ, а его положения должны быть четкими и не должны допускать возможности различного толкования. В то же время нет смысла перегружать протокол, например, переписывая из стандарта RS-232 требования к электрическим параметрам сигнала.

Протокол согласовывается всеми заинтересованными сторонами.

Даже если вы самостоятельно разрабатываете все программное обеспечение, написание и сохранение неформального протокола обмена, например, только его части, касающейся команд, позволит систематизировать ваши представления о том, что и когда будет передаваться. Очень полезен такой протокол наряду с блок-схемой программы и картой распределения памяти, когда вы пытаетесь вспомнить, как работает давно написанная вами программа, и тем более при разборе чужой программы.

Ниже приводится пример протокола обмена, он содержит достаточно информации, чтобы понять задачу, которую предстоит решить в этой главе.

5.2.2. Общие положения

Обмен информацией выполняется асинхронными сериями в соответствии с требованиями интерфейса RS-232. Электрические характеристики сигнала также должны соответствовать требованиям интерфейса RS-232-С.

Данные передаются сообщениями. Сообщения состоят из 5...134 байт и требуют передачи не чаще одного раза в секунду.

Каждый байт сообщения, состоящий из 8 информационных бит, передается в составе группы из одиннадцати бит. Группа начинается стартовым битом, затем передаются 8 информационных бит, за ними следуют бит четности и стоп-бит.

Параметры передаваемых групп:

- скорость передачи 115200 бит/с;
- 8 информационных бит;
- проверка на нечетность;
- стоп-бит.

5.2.3. Структура сообщения

Сообщения имеют следующий вид: 5A CMD <байты данных> CSH CSL, где 5A — стартовый байт, CMD — байт команды, <байты данных> — их количество зависит от байта команды, CSH и CSL — 2 байта контрольной суммы.

Контрольная сумма получается суммированием всех байтов сообщения в сумматоре длиной в 2 байта. В зависимости от команды старший байт контрольной суммы CSH может не передаваться.

5.2.4. Передаваемые сообщения (команды компьютера и ответы контроллера)

В тексте сообщений неизменяемые байты записаны в шестнадцатеричном формате, изменяемые байты записаны как подчеркнутые трехсимвольные переменные.

1. Команда ТЕСТ RS-232: 5A 20 00 00 00 00 7A.
2. Ответ на команду ТЕСТ RS-232: 5A 20 **CRH CRL CSL**, где **CRH** и **CRL** — два байта контрольной суммы содержимого памяти программ контроллера, а **CSL** — младший байт контрольной суммы команды.
3. Команда ПЕРЕДАТЬ БЛОК 128 байт в контроллер: 5A 21 00 00 **B00** ... **BFF CSH CSL**, где **B00** ... **BFF** — 128 байт данных, передаваемых из компьютера, а **CSH** и **CSL** — два байта контрольной суммы команды.
4. Ответ на команду ПЕРЕДАТЬ БЛОК: 5A 21 00 00 7B.
5. Команда ЗАПРОС БЛОКА 128 байт из контроллера: 5A 22 00 00 00 00 7C.
6. Ответ на команду ЗАПРОС БЛОКА: 5A 22 00 00 **B00** ... **BFF CSH CSL**, где **B00** ... **BFF** — 128 байт данных, передаваемых из ОЗУ контроллера, а **CSH** и **CSL** — два байта контрольной суммы команды.
7. Команда ОЧИСТИТЬ ОЗУ: 5A 23 00 00 00 00 7D.
8. Ответ на команду ОЧИСТИТЬ ОЗУ: 5A 23 00 00 7D.

9. Команда ЗАПОЛНИТЬ ОЗУ: 5A 24 **FIL** 00 00 **CSH CSL**, где **FIL** — байт, которым заполняется ОЗУ.
10. Ответ на команду ОЧИСТИТЬ ОЗУ: 5A 24 00 00 7E.
11. Сообщение контроллера об ошибках: 5A 0C **ERR** 00 **CSL**, где **ERR** — код ошибки (E0 — ошибка четности или первый байт команды не 5A, E1 — ошибка контрольной суммы команды, E2 — неизвестная команда), а **CSL** — младший байт контрольной суммы.

Это уже реализованный протокол обмена. Если протокол пишется для себя, то обычно готовится набросок протокола, который корректируется в процессе работы над программами.

И микроконтроллер, и компьютер проверяют поступающие сообщения, в том числе и на их длину. Проще проверять длину сообщений, когда для одного источника она одинакова. В нашем устройстве команды компьютера имеют длину в 7 байт, а ответы контроллера — 5 байт. Исключения составляют сообщения, содержащие блоки данных. Их длина составляет 134 байта.

> **Замечание.** «Лишние» нулевые байты в командах и ответах — это наследство исходных программ, в которых были сообщения с параметрами. Максимальная длина таких сообщений, не содержащих нулевые байты, достигала 7 байт для команд компьютера и 5 байт для ответов контроллера.

5.2.5. Программа для микроконтроллера

Как организовать прием команд, описанных в протоколе обмена? Предположим, все команды содержат одинаковое количество байтов, например семь. Тогда после приема первого байта надо принять еще шесть байт, а затем начать обработку принятой последовательности.

Что произойдет при такой организации, если прием первого байта был вызван помехой? Микроконтроллер при этом просто зависнет, ожидая следующих шести байт, а часть следующей команды будет воспринята как продолжение предыдущей. Если длина команд различна, то, приняв первый байт, надо продолжать прием в течение времени, достаточного для приема самой длинной команды, предусмотренной протоколом обмена, а по истечении этого времени проанализировать принятую последовательность.

Организуем программу для микроконтроллера следующим образом: после необходимой инициализации программа будет выполнять бесконечный цикл, ожидая прерывания после приема первого байта данных по каналу RS-232. Обработчик прерывания окончания приема символа UART (универсальным синхронным и асинхронным приемопередатчи-

ком) запустит таймер T0 и вызовет выполнение подпрограммы приема последовательности байтов. Завершение счета таймера T0 прекратит выполнение подпрограммы приема и инициализирует выполнение ряда подпрограмм анализа принятой последовательности, распознавания команды, ее обработки и передачи ответа микроконтроллером по каналу RS-232.

> **Замечание.** Прерывание окончания приема символа UART происходит после того, как принятый символ помещен в регистр UDR (UART Data Register — регистр данных UART). UART принимает символ длиной 9 бит, 8 из них представляют передаваемый байт, 9-й — это передаваемый бит четности.

Как и в предыдущей главе, листинг программы приводится блоками с комментариями после каждого блока.

> **Замечание.** Следующие далее программы связи с компьютером были написаны для микроконтроллера AT90S8535 и могут без доработок использоваться в микроконтроллере ATmega8535.
> Для этого перед программированием необходимо сбросить в 0 Fuse High Byte (дословно — старший байт перемычек) микроконтроллера ATmega8535 разряд, именуемый S8535 (7-й разряд), что обеспечит работу этого микроконтроллера с программами, написанными для микроконтроллера AT90S8535.
> Такой же разряд, но именуемый S8515, есть в микроконтроллере ATmega8515, обеспечивающий совместимость с программами, написанными для микроконтроллера AT90S8515.
> Используемый вами программатор должен обеспечивать возможность работы с Fuse-байтами используемых вами программируемых микроконтроллеров.

Блок 1. Векторы прерываний, определение констант и переменных, основная часть программы, обработчики прерываний таймера T0 и UART

```
;Программа связи с компьютером по RS-232
.include "c:\avr\def\8535def.inc"
.CSEG
.org 0
        rjmp RESET                  ;Reset Handler
        nop    ;rjmp EXT_INT0       ;IRQ0 Handler
        nop    ;rjmp EXT_INT1       ;IRQ1 Handler
        nop    ;rjmp TIM2_COMP      ;Timer T2 Compare Handler
        nop    ;rjmp TIM2_OVF       ;Timer T2 Overflow Handler
        nop    ;rjmp TIM1_CAPT      ;Timer T1 Capture Handler
        nop    ;rjmp TIM1_COMPA     ;Timer T1 CompareA Handler
        nop    ;rjmp TIM1_COMPB     ;Timer T1 CompareB Handler
```



```
        nop    ;rjmp TIM1_OVF       ;Timer T1 Overflow Handler
        rjmp TIM0_OVF               ;Timer T0 Overflow Handler
        nop    ;rjmp SPI_STC        ;SPI Transfer Handler
        rjmp UART_RXC               ;UART RX Complete Handler
        nop    ;rjmp UART_DRE       ;UDR Empty Handler
        nop    ;rjmp UART_TXC       ;UART TX Complete Handler
        nop                         ;ADC Conversion Complete
                                    ;Interrupt Handler
        nop    ;rjmp EE_RDY         ;EEPROM Ready Handler
        nop    ;rjmp ANA_COMP       ;Analog Comparator Handler
.def    RxTimOvf = r5               ;Устанавливается на время приема
.def    cntRx = r7
.def    tm3 = r8
.def    tm4 = r9
.def    tm1 = r16
.def    tm2 = r17
.def    cnt = r18
.def    cntrs = r20                 ;Число принятых байтов
.def    parrs = r21                 ;Вычисленный бит четности
.def    parinrs = r22               ;Принятый бит четности
.def    rrs = r23
.def    datrs = r24                 ;Переменная, хранящая данные для UART
.def    cmdrs = r25                 ;Переменная кода команды
.equ    r9b = (1<<RXEN) + (1<<CHR9);Прием 9-битных слов
.equ    t9b = (1<<TXEN) + (1<<CHR9);Передача 9-битных слов
;Адреса в ОЗУ:
.equ    aCMD = $60                  ;АДРЕС принятой последовательности
.equ    aNumBlk = $7e               ;АДРЕС номера блока
.equ    aBlk = $80                  ;АДРЕС блока
.equ    RXD = PD0                   ;ЛИНИЯ приема данных
.equ    TXD = PD1                   ;ЛИНИЯ передачи данных
.equ    erParity = $E0              ;ОШИБКА четности
.equ    erCR = $E1                  ;ОШИБКА контрольной суммы
.equ    erUnkCmd = $E2              ;ОШИБКА: команда неизвестна
.equ    FrstRx = $5a                ;Первый байт команды
.equ    cmUnk = $01                 ;неизвестная команда
.equ    cmRS = $20                  ;Команда Тест RS-232
.equ    cmBlPC_S = $21              ;128 байт из PC в ОЗУ МК
.equ    cmBlS_PC = $22              ;128 байт из ОЗУ МК в PC
.equ    cmZero = $23                ;ОЧИСТИТЬ ОЗУ
.equ    cmAA = $24                  ;ЗАПОЛНИТЬ память байтами AA
.equ    cmHi = $25                  ;Код последней команды + 1
RESET:
        ldi    tm1,15               ;WatchDog = 2 c
        out    WDTCR,tm1
        wdr
        ser    tm1                  ;tm1 <- $ff
        out    DDRA,tm1             ;Все линии портов — выходы
        out    DDRB,tm1
        out    DDRC,tm1
        out    DDRD,tm1
```

```
        out     PORTA,tm1
        out     PORTB,tm1
        out     PORTC,tm1
        out     PORTD,tm1
        ldi     tm1,low(RAMEND)     ;ИНИЦИАЛИЗАЦИЯ стека
        out     SPL,tm1
        ldi     tm1,high(RAMEND)
        out     SPH,tm1
;Инициализация UART:
        ldi     tm1,3               ;Для скорости 9600 бод: 47, для
                                    ;115200 бод: 3 (при F = 7.3728 МГц)
        out     UBRR,tm1
        ldi     datrs,r9b + (1<<RXCIE)
        out     UCR,datrs           ;РАЗРЕШЕНИЕ приема 9-битных слов
        clr     tm1
        out     TIMSK,tm1           ;ЗАПРЕТ прерываний таймеров
        clr     RxTimOvf
        clr     cmdrs
        sei
;БЕСКОНЕЧНЫЙ цикл программы:
cycle:  wdr
        tst     RxTimOvf            ;RxTimOvf<-$ff в UART_RXC:
        breq    tstCmd
        rcall   GetCmd
tstCmd:
        cpi     cmdrs,cmRS
        brlo    cycle
        rcall   ExeCmd
lastCmd:
        rjmp    cycle
;конец (БЕСКОНЕЧНЫЙ цикл программы)
;======================
;ОБРАБОТЧИКИ прерываний:
;Обработчик прерывания ОКОНЧАНИЯ приема символа UART
UART_RXC:
        in      rrs,SREG
        push    rrs                 ;СОСТОЯНИЯ регистра статуса – в стек
        clr     RxTimOvf            ;ДАЛЬНЕЙШИЙ прием пока не разрешен
        clr     cntRx               ;ОЧИСТИТЬ счетчик принятых байтов
        ldi     YL,LOW(aCMD)
        ldi     YH,HIGH(aCMD)
        ldi     cmdrs,cmUnk
        in      datrs,UDR
        in      parinrs,UCR
        andi    parinrs,1<<RXB8     ;ВЫДЕЛЕНИЕ бита четности
        lsr     parinrs             ;из принятой группы битов
        rcall   ParClc              ;ВЫЧИСЛЕНИЕ бита четности
        cp      parrs,parinrs       ;Бит четности верен?
        brne    RxRet
        cpi     datrs,FrstRx        ;ПЕРВЫЙ принятый байт верен?
        brne    RxRet
```



```
        clr     cmdrs               ;ЕСЛИ да, cmdrs = 0
        dec     RxTimOvf            ;RxTimOvf = 0 - 1 = $ff –
                                    ;разрешить прием
        clr     rrs
        out     TCCR0,rrs           ;ОСТАНОВИТЬ счет таймера T0
        ldi     rrs,256 - 250
        out     TCNT0,rrs
        in      rrs,TIMSK
        sbr     rrs,1<<TOIE0        ;РАЗРЕШИТЬ прерывание
        out     TIMSK,rrs           ;ПЕРЕПОЛНЕНИЯ таймера T0
        ldi     rrs,5               ;Старт таймера T0 на
                                    ;250 × 1024/7372800 = 0.035 с
        out     TCCR0,rrs           ;С prescaling = 1/1024
        cbi     UCR,RXCIE           ;ЗАПРЕТИТЬ прерывание UART
RxRet:  tst     cmdrs               ;Если ошибки уже есть, то
        breq    endIU               ;послать сообщение об ошибке
        ldi     cmdrs,erParity
        rcall   BadCMD
endIU:  pop     rrs                 ;ВОССТАНОВИТЬ содержимое
        out     SREG,rrs            ;регистра статуса
        reti
;КОНЕЦ (Обработчик окончания приема символа UART)
;Обработчик прерывания переполнения таймера T0
TIM0_OVF:
        clr     RxTimOvf            ;RxTimOvf = 0 – прием закончен
        out     TCCR0,RxTimOvf
        in      tm1,TIMSK
        cbr     tm1,1<<TOIE0        ;ЗАПРЕТ прерывания таймера T0
        out     TIMSK,tm1
        reti
;КОНЕЦ (ОБРАБОТЧИКИ прерываний)
;======================
```

В программе используется прерывание аппаратного сброса *Reset*, а также прерывание переполнения таймера T0 (его обработчик начинается меткой *TIM0_OVF:*) и прерывание завершения приема символа UART (обработчик которого начинается меткой *UART_RXC*).

Определяемые переменные и константы прокомментированы непосредственно в листинге, дополнительные комментарии будут даны по ходу рассмотрения программы.

В обработчике прерывания аппаратного сброса, начинающегося меткой *RESET:*, сторожевой таймер микроконтроллера инициализируется так, чтобы через 2 с работы программы происходил аппаратный сброс микроконтроллера, если не будет обнаружена команда *wdr*. Как вы помните, каждая команда *wdr* будет продлять время работы микроконтроллера без сброса сторожевым таймером еще на 2 с.

Команда *ser* вызывает установку всех разрядов переменной *tm1*, поэтому вывод содержимого этой переменной в порты DDRA, DDRB, DDRC и

DDRD приводит к тому, что все линии портов A, B, C и D работают как выходы. Вывод этой переменной в порты PORTA, PORTB, PORTC и PORTD вызывает установку высоких уровней на всех линиях портов A, B, C и D.

Следующие четыре команды инициализируют стек, размещая его в конце ОЗУ микроконтроллера.

В регистр UBRR (UART Baud Rate Register), определяющий скорость передачи и приема информации UART микроконтроллера, заносится коэффициент 3.

В техническом описании микроконтроллера, имеющего аппаратную поддержку UART, этот коэффициент называется UBRR и связан со скоростью обмена данными UART следующей формулой: $BAUD = F_{такт.}/(16\cdot(1 + UBRR))$, где $F_{такт.} = 7372800$ Гц — тактовая частота микроконтроллера, определяется частотой подключенного к нему кварцевого резонатора; BAUD — скорость обмена информацией UART.

Вычислить коэффициент UBRR, который надо занести в одноименный регистр микроконтроллера для получения скорости BAUD, можно по формуле

$UBRR = (F_{такт.}/16\cdot BAUD) - 1$. Скорость BAUD может быть выбрана из ряда, предусмотренного стандартом RS-232. В этот ряд входят скорости 9600 бод, 19200 бод, 38400 бод, 57600 бод, 115200 бод (весь ряд можно найти в стандарте). В техническом описании микроконтроллера ATmega8535 приводится таблица, по которой можно выбрать коэффициент UBRR в зависимости от частоты подключенного к микроконтроллеру кварцевого резонатора и желаемой скорости обмена информацией.

Поскольку в регистр UBRR мы можем записать только целое значение, для получения точного значения скорости нужно выбирать кварцевые резонатор с определенными частотами, в противном случае скорость будет установлена с некоторой погрешностью. Согласно рекомендации технического описания микроконтроллера максимальная относительная погрешность установки скорости не должна превышать 1%.

При рассмотрении контроллера, управляющего температурой нагрева двух печей, использовался кварцевый резонатор с частотой 6.4 МГц, что обеспечивало максимально допустимую тактовую частоту АЦП и наиболее быстрое преобразование. Определим, какова была бы погрешность установки скорости передачи 9600 бод при использовании этого кварцевого резонатора. По последней формуле рассчитываем точное значение коэффициента UBRR:

$$UBRR = F_{такт.}/(16\cdot BAUD) - 1 = 6400000/(16\cdot 9600) - 1 = 40.6(6).$$

Округляя полученный результат до ближайшего целого, получим UBRR = 41.

Теперь определим значение скорости $BAUD_{уст.}$, которая будет установлена при таком коэффициенте:

$$BAUD_{уст.} = F_{такт.}/(16\cdot(1 + UBRR)) = 6400000/(16(1 + 41)) = 9523.81.$$

Относительная погрешность установки скорости составит

$$|100(BAUD_{уст.} - BAUD)/BAUD| = |100(9523.81 - 9600)/9600| = 0.79\%.$$

Это значит, что кварцевый резонатор с частотой 6.4 МГц обеспечит относительную погрешность менее одного процента при установке скорости обмена информацией в 9600 бод. Вычисление относительной погрешности установки скорости 38400 бод приводит к неудовлетворительному результату (более 4 %).

Частота кварцевого резонатора 7.3728 МГц, выпускаемого, вероятно, специально для подобных целей, позволяет устанавливать любую из перечисленных скоростей ряда с нулевой погрешностью, а для обеспечения скорости в 115200 бод коэффициент UBRR должен быть равен 3. В этом легко убедиться, воспользовавшись представленными выше формулами.

Замечание. Все сказанное касается связи микроконтроллера с компьютером или другим устройством, скорость которого точно соответствует одной из скоростей ряда. Если же производится соединение, например, двух микроконтроллеров по каналу RS-232, то речь может идти об относительной погрешности рассогласования скоростей. При одинаковых тактовых частотах и одинаковом выборе коэффициентов UBRR обоих микроконтроллеров скорости связи, поддерживаемые ими, будут одинаковы (относительная погрешность рассогласования скоростей равна нулю), хотя и могут значительно отличаться от стандартных. Не надо забывать, что значительное отклонение от стандартной скорости является нарушением стандарта.

Продолжим рассмотрение программы. После загрузки регистра UBRR происходит инициализация регистра UCR (UART Control Register — регистр управления UART). Загрузкой константы *r9b* определяется установка разрядов RXEN (Recevier Enable — разрешить работу приемника) и CHR9 (9 Bit Characters — символы в 9 разрядов) регистра UCR. Установка первого из них разрешает прием данных, второго — обмен символами в девять информационных бит (стартовый и стоповый биты считаются служебными, они передаются независимо от разрядности символа), девятый информационный бит используем в качестве бита четности.

Для разрешения прерывания окончания приема символа в регистре UCR устанавливается также и разряд RXCIE.

Очисткой регистра TIMSK запрещаются все прерывания таймеров микроконтроллера.

После очистки переменных *RxTimOvf* и *cmdrs* выполняется команда *sei*, снимающая запрет аппаратных прерываний.

Меткой *cycle:* начинается бесконечный цикл программы, в котором производится очередной сброс сторожевого таймера, после чего проверяется состояние переменной *RxTimOvf*. Если оно не равно нулю, значит, первый байт уже принят, вызывается подпрограмма *GetCmd* приема остальных байтов команды. Манипулируют переменной *RxTimOvf* обработчики прерываний окончания приема символа UART и переполнения таймера T0.

Подпрограмма *GetCmd* в свою очередь вызывает подпрограмму распознавания команд, а при обнаружении ошибок — подпрограмму передачи сообщения об ошибке. После возврата из подпрограммы *GetCmd* переменная *cmdrs* либо содержит код принятой команды, либо ее значение равно нулю (команда обработана, микроконтроллер продолжает выполнение бесконечного цикла). Если содержимое переменной *cmdrs* не ниже самого маленького значения кода команд, которое принадлежит команде ТЕСТ RS-232, то вызывается подпрограмма *ExeCmd* обработки команды, в противном случае, или по завершении обработки команды, происходит возврат к началу бесконечного цикла (метка *cycle:*).

Обработчик прерывания окончания приема символа UART (UART RX Complete Handler) вызывается после того, как первый байт команды оказывается в регистре UDR.

При вызове обработчика прерывания командой *push* в стек помещается содержимое регистра статуса *SREG*.

Это содержимое может измениться при выполнении обработчика. Но для чего оно должно быть сохранено? Вернемся к бесконечному циклу программы. Прерывание может произойти при выполнении любой команды. Представьте, что была выполнена команда бесконечного цикла *tstCmd: cpi cmdrs,cmRS* и содержимое переменной *cmdrs* оказалось меньше константы *cmRS*. Это значит, что флаг C регистра статуса (он же регистр флагов) установлен, поэтому выполнение следующей команды *brlo* вызовет переход на метку *cycle:*.

Если перед выполнением команды *brlo* произойдет прерывание окончания приема символа UART, состояние флага C регистра статуса может измениться, а после возврата из обработчика прерывания вместо перехода на метку *cycle:* будет вызвана подпрограмма *ExeCmd*.

Командами *clr* очищается содержимое переменных *RxTimOvf* и *cntRx*. В регистр Y загружается адрес области ОЗУ микроконтроллера, в которую будут помещаться остальные байты команды по мере их приема.

В переменную *cmdrs* помещается значение константы *cmUnk* (неизвестная команда).

Командой *in* в переменную *datrs* помещается принятый и хранящийся в регистре UDR первый байт команды.

Так как выбран режим приема 9-разрядных символов UART, а регистр UDR, как и другие регистры микроконтроллера, 8-разрядный, 9-й бит принятого символа (это бит четности) помещается в разряд RXB8 регистра UCR.

В переменную *parinrs* помещается содержимое регистра UCR, а команда логического умножения (*andi parinrs,1<<RXB8*) сбрасывает все разряды переменной *parinrs*, кроме разряда RXB8.

Следующая команда *lsr* логического сдвига вправо перемещает содержимое разряда RXB8 (это 1-й разряд) в нулевой разряд переменной *parinrs*. Теперь в 1-м разряде переменной *parinrs* хранится принятый UART бит четности, в остальных разрядах находятся нули.

В вызываемой подпрограмме *ParClc* вычисляется и записывается в переменную *parrs* бит четности принятого первого байта команды.

Переменная *parrs*, хранящая рассчитанный бит четности, сравнивается с переменной *parinrs*, содержащей принятый UART бит четности. При равенстве содержимого обеих переменных считается, что четность принятого символа в норме. В этом случае проверяется значение принятого байта, оно должно быть равно $5A. Если одна из проверок окончилась неудачей, происходит переход на метку *RxRet:* — эта часть обработчика выполняется обязательно и будет рассмотрена позже. Сейчас отметим, что при таком переходе в переменной *cmdrs* хранится код неизвестной команды *cmUnk*.

При успешном выполнении обеих проверок содержимое переменной *cmdrs* очищается, а содержимое переменной *RxTimOvf* декрементируется. Поскольку в начале рассматриваемого обработчика прерывания в эту переменную было занесено нулевое значение, теперь в ней хранится значение $ff.

Следующая группа команд разрешает прерывание таймера T0 и запускает его так, что переполнение его счетчика, а значит, и прерывание переполнения таймера T0 произойдет через 0.035 с. Установка интервала времени счета таймера неоднократно рассматривалась. Остановимся на вопросе, для чего нужен такой интервал и как его определить.

Согласно протоколу обмена, в команде может насчитываться 7 или 134 байта. Поскольку первый принятый байт оказался первым байтом команды ($5A), следует ожидать приема остальных 6 или 133 байт. Время ожидания рассчитаем исходя из максимальной длины команды.

Передача группы начинается стартовым битом, за которым следуют 8 бит байта команды, завершают группу бит четности и стоповый бит. Это означает, что для передачи одного байта по каналу RS-232 в выбранном режиме необходимо переслать 11 бит. Для окончания приема оставшихся 133

байт команды должно быть выделено время, необходимое для передачи по каналу RS-232 1463 бита (11 × 133).

При скорости передачи в 115200 бод (115200 бит/с) для приема этого количества битов потребуется 0.013 с (1463/115200).

Значит, интервал времени счета таймера T0 в 0.035 с выбран почти с трехкратным запасом. Надо учитывать, что компьютер при передаче будет обрабатывать свои внутренние прерывания, увеличивая общее время передачи, поэтому не стоит сокращать запас времени до расчетного предела.

> **Замечание.** Если бы была выбрана скорость передачи в 9600 бод, то пришлось бы воспользоваться 16-разрядным таймером T1, поскольку максимального времени счета 8-разрядного таймера T0 оказалось бы недостаточно для приема команды длиной в 134 байта.

Поскольку первый байт команды принят, а для приема остальных байтов будет вызвана подпрограмма *GetCmd*, поступление следующих байтов не должно вызывать прерывания окончания приема символа UART.

Для того чтобы снова не вызывался обработчик этого прерывания при поступлении последующих байтов, командой *cbi* в регистре UCR сбрасывается разряд RXCIE, вызывая запрет прерывания окончания приема символа UART.

Командой *tst* с меткой *RxRet:* проверяется содержимое переменной *cmdrs*.

Напомню, что на эту метку производится переход при обнаружении ошибок в первом принятом символе, в этом случае в переменной хранится значение *cmUnk*. Если ошибка в первом символе была обнаружена, в переменную *cmdrs* помещается код ошибки *erParity* и вызывается подпрограмма *BadCmd*, посылающая в компьютер сообщение об ошибке, код которой *erParity.* Заметьте, отправляемый код ошибки в нашей программе одинаков для ошибки четности и для неверного первого байта команды.

Перед возвратом из обработчика прерывания восстанавливается состояние регистра статуса, сохраненное в стеке.

В обработчике прерывания переполнения таймера T0, начинающемся меткой *TIM0_OVF:*, очищается переменная *RxTimOvf*, состояние которой отличалось от нуля в течение времени счета таймера T0, посредством очистки регистра TCCR0 счет таймера T0 останавливается, а сбросом разряда TOIE0 регистра TIMSK запрещается прерывание переполнения таймера T0.

По окончании обработки принятой команды снова будет разрешено прерывание окончания приема символа UART, а микроконтроллер вернется в режим ожидания новой команды.

Блок 2. Подпрограммы GetCmd, RxCh, TxCh, TxEnd и ParClc

```
;====== Подпрограммы:
;Получение байтов после прерывания UART_RXC
GetCmd:
        rcall  RxCh
        st     Y +,datrs
        cpi    datrs,cmBlPC_S
        brne   GetMore
        ldi    YL,low(aNumBlk)
        ldi    YH,high(aNumBlk)
;Цикл накопления команды:
GetMore:
        rcall  RxCh
        st     Y +,datrs
        tst    RxTimOvf
        brne   GetMore
        rcall  RcgCmd
        ret
;конец GetCmd
;Прием символа
RxCh:   ldi    datrs,r9b            ;ПРИЕМ 9-битных символов
        out    UCR,datrs
mrx1:   wdr
        tst    RxTimOvf
        breq   mrx2
        sbis   USR,RXC              ;ОЖИДАНИЕ приема в UDR
        rjmp   mrx1
        in     datrs,UDR            ;8 бит слова
        in     parinrs,UCR          ;9-й бит (четность) в UCR[RXB8]
        inc    cntRx
        andi   parinrs,1<<RXB8
        lsr    parinrs
        rcall  ParClc               ;ПРОВЕРКА четности
        cp     parrs,parinrs
        breq   mrx2                 ;ЕСЛИ бит четность не верен
        ldi    cmdrs,1<<cmUnk       ;УСТАНОВИТЬ бит cmUnk в cmdrs
mrx2:   ret
;КОНЕЦ RxCh
;Передача символа
TxCh:   rcall  ParClc
        ldi    rrs,t9b
        sbrc   parrs,0
        inc    rrs
mtx:    sbis   USR,UDRE
        rjmp   mtx
        out    UCR,rrs
        out    UDR,datrs
        ret
;конец TxCh
;Передача последнего символа ответа микроконтроллера
```

```
TxEnd: ldi    tm1,1<<TXC
       out    USR,tm1
       rcall  TxCh
       sbi    PORTD,TXD
TxE:   sbis   USR,TXC
       rjmp   TxE
       ldi    rrs,r9b + (1<<RXCIE)
       out    UCR,rrs
       ret
;конец TxEnd
;Вычисление бита четности
ParClc:ldi    parrs,1              ;0 - для четности, 1 - для нечетности
       ldi    cntrs,8
       mov    rrs,datrs
parclc1:lsrrrs                     ;ОПРЕДЕЛЕНИЕ бита четности
       brcc   parclc2
       com    parrs
parclc2:deccntrs
       brne   parclc1
       andi   parrs,1
       brne   parclc3
       clr    parrs
parclc3:ret                        ;parrs — бит четности
;КОНЕЦ ParClc
```

Подпрограмма *GetCmd* обеспечивает прием команды, за исключением первого байта, выполнение подпрограммы продолжается, пока не будет вызвано прерывание переполнения таймера Т0.

Первой командой подпрограммы вызывается подпрограмма *RxCh*, обслуживающая прием одного байта и помещение его в переменную *datrs*.

После возврата из подпрограммы *RxCh* принятый байт сохраняется в области ОЗУ, отведенной для команды компьютера. Этот байт является вторым байтом команды и определяет код команды. Командой *cpi* выполняется проверка, не равно ли содержимое байта коду команды *cmBlPC_S*, в этом случае принимается команда передачи 128-байтного блока данных, которые должны быть помещены в отдельную область ОЗУ. Тогда в регистр Y помещается адрес *aNumBlk* области ОЗУ, в которую будут помещены последующие два байта номера блока и 128 байт самого блока при выполнении последовательности команд, начинающихся меткой *GetMore:*.

Номер блока — это наследие взятой за основу программы обслуживания программатора микросхем EEPROM и флеш-памяти, объем программируемой памяти достигал 128 килобайт, соответственно количество 128-байтных блоков, передаваемых по каналу RS-232, достигало 1024, а нумерация требовала двух байт.

Если принятый байт содержит любой другой код, отличный от *cmBlPC_S*, запись принимаемых байтов будет продолжена в область ОЗУ, отведенную для команды компьютера.

В цикле, начинающемся меткой *GetMore:*, вызывается подпрограмма *RxCh*, сохраняющая очередной принятый байт в переменной *datrs*. Этот байт записывается в ОЗУ микроконтроллера по адресу, определяемому содержимым регистра Y.

Командой *tst* проверяется содержимое переменной *RxTimOvf*: пока оно не равно нулю, происходит возврат к метке *GetMore:*, прием команды продолжается.

Как только содержимое переменной *RxTimOvf* станет равным нулю, а очищается оно обработчиком прерывания переполнения таймера Т0, время приема команды истекло, выполнение цикла приема завершится, будет вызвана подпрограмма распознавания команды *RcgCmd*.

В подпрограмме *RxCh* в регистр UCR передается значение константы *r9b*, устанавливая режим приема 9-битных символов UART, командой *wdr* (метка *mrx1:*) сбрасывается сторожевой таймер.

Так же, как и в подпрограмме *GetCmd*, выполняется проверка состояния переменной *RxTimOvf*, ведь окончание счета таймера Т0 вызывающее очистку этой переменной, может наступить как во время выполнения подпрограммы *GetCmd*, так и подпрограммы *RxCh*.

Получается, что при обнулении переменной *RxTimOvf* подпрограмма *RxCh* завершится (произойдет переход на метку *mrx2:*), а при возврате из этой подпрограммы закончится и выполнение цикла накопления команды в подпрограмме *GetCmd*.

Команда *sbis* вызывает проверку разряда RXC (Receive Complet — прием завершен) в регистре USR (UART Status Register — регистр статуса UART).

> **Замечание.** В UART для приема используется регистр сдвига, к которому нет программного доступа. Очередной принятый бит данных записывается в этот регистр, сдвигая уже записанные биты. По окончании приема всего символа байт данных передается в регистр UDR, если установлен режим приема 9-разрядных символов, то последний бит помещается в разряд RXB8 регистра UCR. Только после этого устанавливается разряд RXC в регистре USR, что свидетельствует о завершении приема символа.

Если разряд RXC регистра USR не установлен, ожидание поступления данных продолжается, происходит возврат на метку *mrx1:*.

Если символ принят, выполняются две команды *in*, передающие содержимое регистра UDR переменной *datrs*, а содержимое регистра UDR — переменной *parinrs*.

Содержимое счетчика поступивших байтов *cntRx* увеличивается на единицу.

Команды *andi* и *lsr* выделяют разряд RXB8 (9-й разряд принятого символа — бит четности), а затем вызывается подпрограмма вычисления бита четности и сравнение его с выделенным битом четности, так же, как это было в обработчике прерывания окончания приема символа UART.

Если обнаружена ошибка четности (расчетный и принятый биты четности не одинаковы), в переменную *cmdrs* записывается значение cmUnk, если четность в норме — в переменной остается нулевое значение, занесенное в нее в обработчике прерывания окончания приема символа UART.

Выполнение подпрограммы *RxCh* завершено, а по состоянию переменной *cmdrs* можно определить, были ли ошибки при приеме всех уже поступивших символов.

Замечание. Нельзя прекращать прием сразу после обнаружения ошибки четности, надо дождаться окончания времени приема. в противном случае микроконтроллер может успеть обработать ошибку и снова перейти в режим приема еще до окончания поступления команды с обнаруженной ошибкой четности. Это значит, что будет приниматься та же команда, но не с ее начала.

Перед вызовом следующей подпрограммы *TxCh*, выполняющей передачу байта данных из микроконтроллера, в переменную *datrs* должен быть записан подлежащий передаче байт.

Первая команда подпрограммы *TxCh* вызывает подпрограмму *ParClc* вычисления бита четности для байта, хранящегося в переменной *datrs*. Результат вычисления записывается в переменную *parrs*.

В переменную *rrs* записывается значение константы *t9b*. Позже содержимое переменной *rrs* будет записано в регистр UCR. Значит, после такой записи в регистре UCR окажутся установленными разряды TXEN (Transmitter Enable — разрешение работы передатчика) и CHR9 (работа UART с 9-разрядными символами). Если вычисленный бит четности не равен нулю, то содержимое переменной *rrs* увеличивается на единицу, значит, после записи ее содержимого в регистре UCR установится разряд TXB8 — последний разряд передаваемого 9-разрядного символа.

Для передачи данных в UART используется пара регистров. Один из них — регистр UDR (он использовался и при приеме данных). При передаче в UDR помещается байт данных, а при передаче 9-разрядных символов в разряд TXB8 регистра UCR помещается последний передаваемый бит символа. Второй регистр — это регистр сдвига. Как только в регистр UDR записываются данные, они передаются в регистр сдвига вместе с последним передаваемым битом из разряда TXB8 регистра UCR, после чего начинается передача.

Только после переноса данных в сдвиговый регистр передатчика в регистре статуса USR устанавливается разряд UDRE, а в разряд TXB8 регистра UCR и в регистр UDR можно загружать новый передаваемый символ.

Поэтому перед загрузкой передаваемого символа командой *sbis* выполняется проверка состояния разряда UDRE регистра USR. Пока передатчик не готов к загрузке следующего символа, происходит возврат на метку *mtx:*. По готовности передатчика в регистр UCR передается содержимое переменной *rrs*, а в регистр UDR — состояние переменной *datrs*, инициализируя передачу символа по каналу RS-232. Этим завершается подпрограмма *TxCh*.

Подпрограмма *TxEnd* обеспечивает передачу последнего символа ответа микроконтроллера, она сразу обращается к подпрограмме *TxCh*. Поэтому до вызова подпрограммы *TxEnd* передаваемый байт должен быть помещен в переменную *datrs*.

В регистре статуса USR очищается разряд TXC. Следующая установка этого разряда UART будет свидетельствовать об окончании передачи последнего символа ответа микроконтроллера.

После выполнения подпрограммы *TxCh* командой *sbi* на линии TXD (это имя, данное нами линии PD1 порта D) удерживается ВЫСОКИЙ уровень до тех пор, пока передача не завершится. О завершении передачи свидетельствует установка разряда TXC регистра USR: пока этот разряд не установится, будет происходить возврат на метку *TxE:*.

При замене подпрограммы *TxEnd* подпрограммой *TxCh* для передачи последнего байта на компьютере обнаруживались ошибки четности при приеме последнего байта.

По окончании передачи последнего символа ответа микроконтроллера регистр UCR вновь устанавливается так, чтобы обеспечить прерывание окончания приема символа UART после поступления первого символа очередной команды компьютера.

Подпрограмма *ParClc* вычисляет бит четности для байта данных, хранящегося в переменной *datrs*. Эта подпрограмма вызывается как после приема символа, так и перед передачей символа.

Согласно составленному протоколу обмена должен выполняться контроль нечетности пересылаемого байта. Это значит, что если в байте четное число единиц, то бит четности равен единице, в противном случае — ноль.

Если бы выполнялась проверка четности, то в случае четного числа единиц в байте бит четности был бы равен нулю.

Определение типа проверки в подпрограмме *ParClc* осуществляется выбором в первой команде *ldi* подпрограммы значения загружаемой в пе-

ременную *parrs* величины. Для проверки нечетности пересылаемого байта в переменную parrs должна быть загружена единица, для контроля четности — ноль.

В счетчик циклов *cntrs* загружается число 8, равное количеству разрядов в байте.

В переменную *rrs* копируется значение передаваемого байта, хранящееся в переменной *datrs*, это обеспечит сохранность последней переменной после выхода из подпрограммы.

Определять бит четности будем так: будем сдвигать байт данных вправо восемь раз, пока содержимое самого старшего разряда не переместится за разрядную сетку. Если при очередном сдвиге за разрядной сеткой оказалась единица, то все содержимое переменной parrs поразрядно инвертируется. При четном количестве единиц в анализируемом байте число инверсий будет четным, а в переменной parrs сохранится исходное значение (туда была записана единица). При нечетном количестве единиц в байте младший разряд переменной parrs будет содержать ноль.

Меткой *parclc1:* начинается цикл сдвига анализируемого байта, хранящегося в переменной *rrs*. После логического сдвига вправо, вызываемого командой *lsr*, команда *brcc* проверяет, установился ли флаг С в регистре флагов. Если флаг установлен, выполняется поразрядное инвертирование содержимого переменной *rrs*.

Установка флага С вызывается выходом единицы за разрядную сетку при сдвиге байта. Это значит, что перед сдвигом в младшем (нулевом) разряде сдвигаемого байта находилась единица, после сдвига вправо содержимое 1-го разряда было перенесено в 0-й разряд, 2-го разряда — в 1-й разряд и так далее, содержимое же нулевого разряда как бы перешло в разряд С (флаг С) регистра флагов.

Содержимое счетчика циклов *cntrs* декрементируется, если оно не достигло нуля, происходит возврат на метку *parclc1:* и цикл повторяется.

По завершении цикла командой *andi* обнуляются все разряды переменной *parrs* кроме младшего. Если в младшем разряде остался ноль, выполняется очистка всех разрядов переменной *parrs* командой *clr*.

Выполнение последних трех команд дает правильный результат как при контроле четности, так и при контроле нечетности.

Блок 3. Подпрограммы RcgCmd, BadCmd и ExeCmd

```
;Распознать команду, проверить контрольную сумму
RcgCmd:wdr
        sbrs    cmdrs,cmUnk
        rjmp    chkCR
;ОШИБКА четности в принятой команде:
        ldi     cmdrs,erParity
        rcall   BadCmd
```



```
        rjmp    rcgend
chkCR:  ldi     cmdrs,FrstRx
        mov     cntrs,cntRx
        subi    cntrs,2                 ;Без 2 последних байт CR
        clr     tm4
        clr     tm3
        ldi     YL,LOW(aCMD)
        ldi     YH,HIGH(aCMD)
        ld      datrs,Y +
        add     cmdrs,datrs
        adc     tm4,tm3
        dec     cntrs
        cpi     datrs,cmBlPC_S
        brne    nextbt
        ldi     YL,low(aNumBlk)
        ldi     YH,high(aNumBlk)
nextbt:ld       datrs,Y +
        add     cmdrs,datrs
        adc     tm4,tm3
        dec     cntrs
        brne    nextbt
        ld      datrs,Y +
        ld      tm3,Y +
        cp      tm3,cmdrs
        cpc     datrs,tm4
        breq    defCmd
        ldi     cmdrs,erCR
        rcall   BadCmd
        rjmp    rcgend
defCmd:ldi      YL,LOW(aCMD)
        ldi     YH,HIGH(aCMD)
        ld      cmdrs,Y +
        cpi     cmdrs,cmHi
        brlo    cmd_chl
        clr     cmdrs
cmd_chl:cpicmdrs,cmRS
        brge    rcgend
        clr     cmdrs
nocmd:  tst     cmdrs
        brne    rcgend
        ldi     cmdrs,erUnkCmd
        rcall   BadCmd
rcgend:ret
;КОНЕЦ RcgCmd
;Передача сообщения об ошибке команды компьютера
BadCmd:clr      cntRx
        ldi     datrs,FrstRx
        add     cntRx,datrs
        rcall   TxCh
        ldi     datrs,$0c
        add     cntRx,datrs
```

```
        rcall  TxCh
        mov    datrs,cmdrs
        add    cntRx,datrs
        rcall  TxCh
        ldi    datrs,0
        rcall  TxCh
        mov    datrs,cntRx
        rcall  TxEnd
        clr    cmdrs
        ldi    datrs,r9b + (1<<RXCIE);ПРИЕМ 9-битных слов
        out    UCR,datrs
        ret
;КОНЕЦ BadCmd
;Выполнить команду
ExeCmd:cbi     UCR,RXCIE
        wdr
        cpi    cmdrs,cmRS
        brne   RSc1
        rcall  xTstRS
RSc1:   cpi    cmdrs,cmBlPC_S
        brne   RSc2
        rcall  xBlPC_S
RSc2:   cpi    cmdrs,cmBlS_PC
        brne   RSc3
        rcall  xBlM_PC
RSc3:   cpi    cmdrs,cmZero
        brne   RSc4
        clr    tm1
        rcall  xLdZ
        rcall  xBlPC_S
RSc4:   cpi    cmdrs,cmAA
;Выполнить команды
        ldi    YL,low(aCMD + 1)
        ld     tm1,Y
        rcall  xLdZ
        rcall  xBlPC_S
RSc5:   clr    cmdrs
        ldi    datrs,r9b + (1<<RXCIE)
        out    UCR,datrs
        ret
;end ExeCmd
```

Подпрограмма *RcgCmd* выполняет распознавание принятой команды, вычисляет контрольную сумму, а при необходимости передает в компьютер сообщение об обнаружении ошибок.

После сброса сторожевого таймера выполняется проверка состояния переменной *cmdrs*, если оно оказывается равным *cmUnk*, значит, при приеме команды была ошибка.

В случае ошибки в переменную *cmdrs* записывается код ошибки *erParity* и вызывается подпрограмма *BadCmd*, передающая сообщение об ошибке компьютеру, после чего выполнение подпрограммы завершается.

При отсутствии ошибки происходит переход на метку *chkCR:*, которой начинается вычисление контрольной суммы принятой команды.

В переменную *cmdrs* записывается константа *FrstRx*, равная значению первого байта любой команды. В переменную *cntrs* помещается число принятых байтов, а поскольку саму контрольную сумму представляют два последних байта команды, содержимое переменной уменьшается на два.

Переменные *tm4* и *tm3* очищаются, а в регистр Y записывается адрес *aCMD* области ОЗУ, хранящей принятую команду. Напомню, что всегда одинаковый первый байт команды в ОЗУ не записывался, сейчас его значение — в переменной *cmdrs*.

В переменную *datrs* считывается из ОЗУ второй байт команды, его значение прибавляется к содержимому пары переменных *tm4:cmdrs*, в которых накапливается контрольная сумма.

Второй байт содержит код команды, командой *cpi* он сравнивается с кодом *cmBlPC_S*, если они равны, значит, принята команда длиной в 134 байта и оставшиеся байты размещены в другой области ОЗУ, тогда в регистр Y заносится адрес этой области.

Меткой *nextbt:* обозначено начало цикла, в котором накопление контрольной суммы команды продолжается до обнуления счетчика числа принятых байтов *cntrs*.

Вычисленная контрольная сумма остается в переменных *tm4:cmdrs*. Два последних байта команды копируются из ОЗУ в пару переменных *datrs:tm3* — они представляют принятую по каналу RS-232 контрольную сумму команды. Обе контрольные суммы сравниваются.

Если они не равны — произошла ошибка контрольной суммы, в переменную *cmdrs* помещается код ошибки контрольной суммы и выполняется подпрограмма *BadCmd* передачи сообщения об ошибке, затем подпрограмма *RcgCmd* завершается.

Если контрольная сумма команды верна, с метки *defCmd:* начинается проверка кода принятой команды. Если код команды больше кода *cmHi* или меньше кода *cmRS* — принята неизвестная команда, тогда вызывается подпрограмма *BadCmd*, передающая сообщение об ошибке с кодом *erUnkCmd* (неизвестная команда).

По завершении подпрограммы *RcgCmd* в переменной *cmdrs* хранится либо код команды, а он больше или равен коду *cmRS*, или код, меньший кода *cmRS* при обнаружении какой-либо ошибки в принятой команде.

Подпрограмма *BadCmd* обеспечивает передачу сообщения об ошибке. Контрольная сумма ответа микроконтроллера накапливается в переменной *cntRx*, перед каждым вызовом подпрограммы *TxCh* передачи одного

символа байт, подлежащий пересылке, помещается в переменную *datrs*, а его значение добавляется к содержимому переменной *cntRx*.

Согласно протоколу обмена первым передается байт $5A, затем код $0C сообщения об ошибке, далее код ошибки, нулевой байт и байт контрольной суммы.

Перед завершением подпрограммы *BadCmd* устанавливается режим приема 9-разрядных символов для UART.

Подпрограмма *ExeCmd* начинается очисткой разряда RXCIE регистра UCR, вызывающей запрещение прерывания окончания приема данных UART. Пока выполнение подпрограммы не будет завершено, все поступающие команды будут игнорироваться.

Затем в подпрограмме выполняется сравнение кода команды, хранящегося в переменной *cmdrs* с кодами известных команд, при совпадении с одним из кодов вызывается соответствующая подпрограмма.

Для команды с кодом *cmZero* (заполнить 128 ячеек памяти нулевыми значениями) дополнительно вызывается подпрограмма *xLdZ* заполнения, а для команды с кодом cmAA (записать в 128 ячеек памяти одинаковые байты) для заполнения ячеек извлекается 3-й байт команды.

Перед завершением подпрограммы *ExeCmd* устанавливается режим приема 9-разрядных символов для UART.

Блок 4. Подпрограммы xLdz, xTstRS

```
;ЗАГРУЗИТЬ массив из нулей в ОЗУ
xLdZ:  ldi    YH,high(aBlk)
       ldi    YL,low(aBlk)
       ldi    cnt,128
LdZ1:  st     Y +,tm1
       dec    cnt
       brne   LdZ1
       ret
;КОНЕЦ xLdZ
;Выполнение команд:|
;==================
;Тест RS-232
xTstRS:ldi    ZL,low(FLASHEND*2 + 1)
       ldi    ZH,high(FLASHEND*2 + 1)
       clr    cntRx
       clr    tm1
       clr    tm2
clcCR: lpm
       add    tm1,r0
       adc    tm2,cntRx
       sbiw   ZL,1
       brcc   clcCR
       wdr
       ldi    datrs,FrstRx
```



```
       rcall  TxCh
       mov    cntRx,datrs
       mov    datrs,cmdrs
       rcall  TxCh
       add    cntRx,datrs
       mov    datrs,tm2
       rcall  TxCh
       add    cntRx,datrs
       mov    datrs,tm1
       rcall  TxCh
       add    datrs,cntRx
       ldi    tm1,1<<TXC
       out    USR,tm1
       rcall  TxEnd
xTst6: ret
;КОНЕЦ (Тест RS-232)
;128 байт PC->Sram
xBlPC_S:ldidatrs,FrstRx
       rcall  TxCh
       mov    cntRx,datrs
       mov    datrs,cmdrs
       add    cntRx,datrs
       rcall  TxCh
       ldi    datrs,0
       rcall  TxCh
       rcall  TxCh
       mov    datrs,cntRx
       rcall  TxEnd
       ret
;КОНЕЦ 128 байтов PC->Sram
;128 байтов SRAM->PC
xBlM_PC:
       lds    tm1,aCMD + 1        ;1-й байт команды не хранится в ОЗУ
       sts    aNumBlk,tm1
       lds    tm1,aCMD + 2        ;КОПИРОВАНИЕ № блока из команды
       sts    aNumBlk + 1,tm1     ;в ячейку aNumBlk
xBlM1: ldi    YL,low(aNumBlk)
       ldi    YH,high(aNumBlk)
       clr    tm2
       clr    tm4
       ldi    datrs,FrstRx
       rcall  TxCh
       mov    tm1,datrs
       mov    datrs,cmdrs
       rcall  TxCh
       add    tm1,datrs
       adc    tm4,tm2
       ldi    cnt,130             ;BlNumH:BlNumL + BlkSize
xBlM2: wdr
       ld     datrs,Y +
       rcall  TxCh
```

```
add    tm1,datrs
adc    tm4,tm2
dec    cnt
brne   xBlM2
mov    datrs,tm4
rcall  TxCh
mov    datrs,tm1
rcall  TxEnd
ret
;КОНЕЦ 128 байтов SRAM->PC
```

Работа подпрограммы *xLdZ*, записывающей в 128 ячеек ОЗУ значение, хранящееся в переменной *tm1*, вполне очевидна. Поэтому перед вызовом подпрограммы *xLdZ* переменной *tm1* надо присвоить значение, которым необходимо заполнить ячейки ОЗУ.

Подпрограмма *xTstRS* вычисляет контрольную сумму памяти программ микроконтроллера и посылает ее в составе ответа на команду компьютера ТЕСТ RS-232.

В качестве памяти программ микроконтроллера используется флеш-память, для которой гарантируется сохранение работоспособности после 1000 циклов перезаписи и хранение данных в течение 10 лет.

Поэтому полезно иметь информацию о целостности хранящихся в ней данных. Эту информацию дает контрольная сумма всей памяти программ, передаваемая в компьютер.

В начале подпрограммы *xTstRS* в регистр Z загружается адрес последней ячейки памяти программ. Так как при обращении к памяти программ микроконтроллер обычно извлекает двухбайтные слова, константа FLASHEND, хранящаяся в подключаемом файле 8535def.inc, содержит адрес последней ячейки размером в два байта. Для вычисления контрольной суммы надо извлекать каждый байт отдельно. При удвоении значения FLASHEND получится адрес предпоследней ячейки размером в байт. Поэтому необходимо добавить единицу.

В очищаемых переменных *tm2:tm1* будет накапливаться результат в цикле, начинающемся меткой *clcCR:*.

Команда *lpm* извлекает один байт из памяти программ и помещает его в регистр r0.

После добавления извлеченного байта к содержимому переменных *tm2:tm1* адрес, хранящийся в регистре Z, уменьшается на единицу, а командой *brcc* проверяется, не стал ли он меньше нуля.

> **Замечание.** Если бы проверка производилась командой brne, байт, хранящийся в ячейке с нулевым адресом, не извлекался бы.

После вычисления контрольной суммы памяти программ организуется передача ответа микроконтроллера согласно протоколу обмена. Аналогичная процедура уже рассматривалась при описании подпрограммы *BadCmd* передачи сообщения об ошибке, поэтому при рассмотрении этой и последующих подпрограмм она не рассматривается.

Контрольная сумма ответа микроконтроллера накапливается в переменной *cntRx*, ее передачей завершается подпрограмма *xTstRS*.

Подпрограмма *xBlPC_S* только передает ответ микроконтроллера на команду компьютера «Передать блок 128 байт в контроллер». Сам блок данных загружается в ОЗУ микроконтроллера во время приема уже рассмотренной подпрограммой *GetCmd*.

Подпрограмма *xBlM_PC* извлекает данные из ОЗУ микроконтроллера и передает их в составе ответа на команду компьютера ЗАПРОС БЛОКА 128 байт из контроллера.

Из принятой команды, хранящейся в ОЗУ микроконтроллера, извлекаются 2-й и 3-й байты, определяющие номер блока, и копируются в ячейки с адресом *aNumBlk*. В этой программе двухбайтный номер блока просто отправляется в ответе микроконтроллера перед блоком 128 байт. В исходной программе обслуживания программатора ПЗУ из микросхемы памяти соответствующий блок данных копировался в ОЗУ и вместе с его номером передавался в компьютер. Так как 128 байт данных в ОЗУ размещаются непосредственно за номером блока, они извлекаются единым блоком в 130 байт, начиная с адреса *aNumBlk*.

Отличительной особенностью организации рассматриваемого ответа на команду является вычисление двухбайтной контрольной суммы ответа, которая накапливается в паре переменных *tm4:tm1*, передача которых завершает подпрограмму.

В рассмотренной программе для организации приема и передачи данных по каналу RS-232 прерывание используется только при приеме первого символа, это объясняется преемственностью рассматриваемых программ: один и тот же удовлетворительно работающий блок обмена данными копируется из программы в программу. Можно организовать не менее эффективный обмен данными, пользуясь прерываниями UART как при приеме, так и при передаче, это может несколько сократить объем программы.

5.2.6. Отладка работы UART в AVR Studio

После выполнения обычных действий по созданию проекта, набора Программы связи с компьютером по каналу RS-232 и ее ассемблирования перейдите к отладке.

Программа может использоваться в микроконтроллере ATmega8535, хотя написана она для AT90S8535. Поэтому следует выбрать AT90S8535 в качестве целевого устройства (Debug/Select Platform and Device.../AVR Simulator/AT90S8535). Этим же объясняется включение директивой .include файла 8535def.inc (а не m8535def.inc, связанного с микроконтроллером ATmega8535) а также использование наименования UART (для ATmega8535 эта функция именуется USART, а число и наименования регистров несколько отличаются).

Найдите в окне программы метку *cycle:* и выполните часть программы до этой метки (Ctrl + F10). Выполните 5...7 шагов отладки (F11) — программа все время возвращается на метку *cycle:*, с которой начинается бесконечный цикл.

Откройте окно IO (меню View/New IO View), разверните UART/Status Register и UART/Control Register так, чтобы видеть наименования разрядов. Щелкните правой клавишей мышки по строке Data Register и в открывшемся окне выберите режим Check Box.

Имитируем окончание приема символа UART, установив в Status Register разряд RXC. Следующий шаг отладки вызовет переход на вектор прерывания окончания приема символа UART, по которому расположена команда *rjmp UART_RXC*. Ее выполнение переместит указатель отладчика на начало обработчика прерывания. Выполните часть обработчика до команды *in datrs, UDR*.

Имитируем прием первого байта команды $5A или в двоичном коде 01011010. Для этого установим разряды 6, 4, 3 и 1 в Data Register (окно IO/UART/Data Register). Выполним один шаг отладки, введенные данные должны оказаться в переменной datrs, проверьте ее состояние, вызвав окно Watches.

Так как в принятом байте четное число единиц, то бит четности должен равняться единице (при проверке на нечетность общее число единиц в 9-разрядном символе нечетно). Поэтому в Control Register установим разряд RXB8, хранящий последний бит символа.

Отлаживая программу далее, можно проследить вычисление бита четности, сравнение вычисленного и принятого битов, выполнение проверки значения первого байта команды.

Так как ошибки при приеме первого байта команды не обнаружены, обработчик прерывания выполняется полностью, а после возврата в бесконечный цикл программы будет вызвана подпрограмма *GetCmd* приема оставшихся байтов команды.

Для приема отдельных символов вызывается подпрограмма *RxCh*.

При ее отладке надо повторить действия, описанные для обработчика прерывания *UART_RXC:* выполнить несколько шагов отладки (это будет приводить к возврату на метку *mrx1:*), установить в Status Register разряд RXC, ввести в Data Register значение очередного байта команды, а в Control Register изменить состояние разряда RXB8 в соответствии с тем, каким должен быть бит четности для записанного байта.

Вызовите окно Memory, чтобы увидеть отображение поступающей команды в ОЗУ микроконтроллера.

При необходимости отладить подпрограмму распознавания команд перед меткой *cycle:* установите команду *rjmp RcgCmd*.

При этом в соответствующие ячейки окна Memory введите байты команды. Напомню, что первый байт $5A не записывается в ОЗУ, а байты команды «Записать блок 128 байт в контроллер» разнесены: только один байт, представляющий код команды, записывается в ячейку по адресу *aCMD*, остальные 132 байта помещаются в область ОЗУ, адрес которой *aNumBlk* (0x007E).

5.3. Канал RS-232: программное обеспечение для компьютера

При разработке приложения в Delphi для управления COM-портом компьютера можно использовать обращение к портам ввода/вывода, вызывать функции операционной системы (Windows API) или использовать готовый компонент, который можно найти в сети Internet для среды разработки, в которой выполняется приложение.

Предлагаемое в качестве примера программное обеспечение выполнено в среде разработки Borland Delphi 4.

Программы проверялись в Delphi 5 и в Delphi 6. Для этого в Delphi устанавливались компоненты VaComm, соответствующие версии Delphi.

Других доработок для Delphi 5 не потребовалось.

Для Delphi 6 у компонента VaComm в окне Object Inspector оказалось меньше свойств, поэтому если вы используете Delphi 6, свойства ReadTimeOut, WriteTimeOut, DTRControl, ReadBufSize, WriteBufSize не определяются.

Поскольку в Delphi не предусмотрены функции обращения к портам ввода/вывода компьютера, их придется писать самостоятельно на ассемблере. Программа получится громоздкой и неудобной. А при создании приложений под операционную систему Windows NT/2000/XP такой вариант потребует дополнительных мер для обхода запрета, налагаемого операционной системой на прямые обращения к портам ввода/вывода. Поэтому непосредственное обращение к портам ввода/вывода, без которого нельзя было обойтись на компьютерах с DOS, использовать в приложениях под операционные системы Windows 9x, 2000 или NT нецелесообразно.

При отсутствии компонентов можно пользоваться функциями Windows API, для этого необходимо наличие описания этих функций и их параметров, их можно разыскать в сети. Преимуществом функций Windows API является то, что обращение к ним остается одинаковым независимо от того, пишете вы приложение в Delphi или в C.

Наиболее удобным является использование готовых компонентов для работы с COM-портом, поэтому в рассматриваемых программах используется компонент VaComm из набора компонентов Varian Async32, предлагаемый компанией Varian Software. Этот компонент с файлами помощи и рекомендациями по установке для разных версий Delphi можно найти в сети.

О том, как добавить страницу с компонентом, поддерживающим обращение к COM-порту, в палитру компонентов Delphi описывается в Delphi Help и здесь не рассматривается.

5.3.1. Минимальные сведения о Delphi

На **Рис. 30** показаны окна, появляющиеся при запуске Borland Delphi 4.

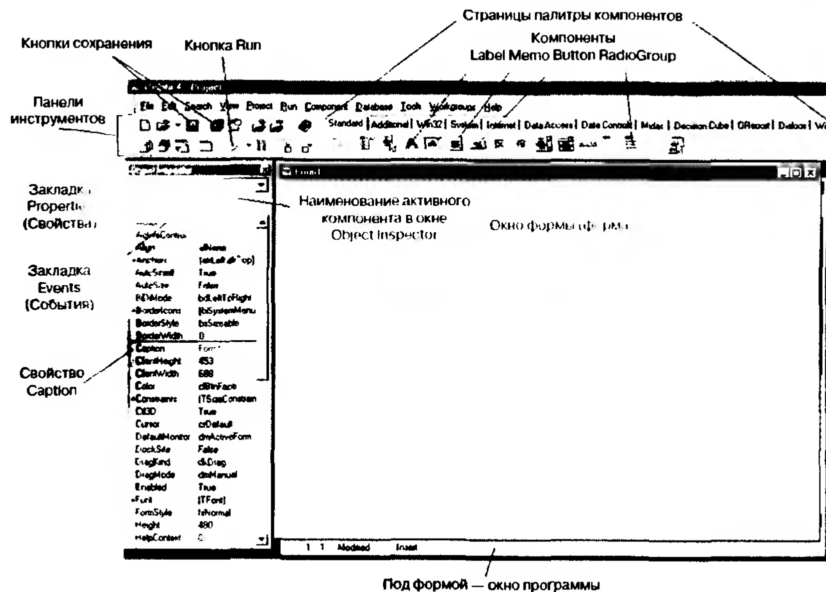


Рис. 30. Окна Delphi 4

Если после запуска Delphi вы перейдете в окно программы, находящееся под формой, то увидите, что программа уже не пуста. По мере помещения компонентов на форму в окно программы будут автоматически добавляться записи, соответствующие помещаемым компонентам.

При описании программы будет указано, какой компонент и с какой страницы палитры компонентов надо поместить на форму. Для поиска компонента на странице надо устанавливать указатель мышки последовательно на каждый компонент страницы и читать появляющуюся подсказку, содержащую наименование компонента.

Для помещения компонента, например Label, на форму на странице Standard надо найти этот компонент, щелкнуть по нему мышкой, затем щелкнуть мышкой в том месте формы, где этот компонент должен быть расположен. Размещенному на форме компоненту автоматически присваивается номер, теперь он называется Label 1.

После помещения на форму компонент активен (на форме он выделен рамкой). В окне Object Inspector в верхней строке представлено его наименование и тип (например, Label 1: TLabel). Если по этой строке щелкнуть мышкой, откроется список компонентов, размещенных на форме, здесь же есть и имя самой формы. Выбрав наименование компонента (или формы) из списка, вы переводите его в активное состояние.

В следующей строке окна находятся закладки Properties (свойства) и Events (события).

Если выбрана закладка Properties, все последующие строки в полях слева содержат свойства активного компонента, поля справа — значения этих свойств.

Свойство Caption введенного компонента имеет значение Label 1. Выделите значение свойства Caption, щелкнув по нему мышкой, введите новое значение «Команда PC». Это изменение касается только надписи, появляющейся на экране, наименование компонента осталось неизменным, оно хранится в свойстве Name.

Если выбрана закладка Events, то все последующие строки в полях слева содержат события, в полях справа (они сейчас пусты) — обработчики этих событий.

5.3.2. Программа обмена данными с микроконтроллером

Описываемая программа отвечает требованиям представленного ранее протокола обмена по каналу RS-232, дополнительно программа обеспечивает выполнение двух команд, не предусмотренных протоколом, что позволяет проверить ответы микроконтроллера при получении им неизвестной команды и команды с неверной контрольной суммой.

Для начала решим, что должна делать программа и как будет выглядеть форма, появляющаяся на экране монитора. Поскольку пользователь должен иметь возможность выбрать одну из команд, предусмотренных протоколом обмена, на форме должен быть список команд, в одной строке списка — одна команда, при щелчке по одной из команд она должна стать активной (как-то выделиться). Для этого в палитре компонентов Delphi есть подходящий компонент RadioGroup. Для выполнения команды нам понадобится кнопка запуска (компонент Button). Щелчок мышкой по кнопке вызовет передачу команды в COM-порт, а дополнительно устанавливаемый в Delphi компонент VaComm обеспечит передачу. Этот же компонент обеспечит и прием ответа микроконтроллера. Переданные команды и принятые ответы должны отражаться на экране так, чтобы можно было увидеть не только последнюю выполненную команду и принятый ответ, но и все предыдущие команды и ответы. Для этого подойдут два компонента Memo — один для команд, другой для ответов.

Для отображения ошибок, которые могут обнаружиться при обмене данными по каналу RS-232, воспользуемся компонентом StatusBar.

Организация приема сообщения под управлением программы компьютера похожа на организацию приема для микроконтроллера. Здесь также понадобится таймер, для этого воспользуемся компонентом Timer. После приема первого символа таймер будет запущен, а по окончании его работы все поступившие данные будут преобразованы и отображены на экране монитора.

В уже рассматривавшейся программе микроконтроллера для обеспечения связи по каналу RS-232 использовалось прерывание переполнения таймера и прерывание окончания приема символа.

Аналоги этих прерываний будут использованы и в программе для компьютера, только в Delphi они называются Events (события), при наступлении события вызывается его обработчик.

Кроме событий приема символа компонента VaComm и окончания работы компонента Timer в программе используем событие щелчка мышки по кнопке Button. Обработчик этого события будет передавать команду в COM-порт.

Теперь, когда понятны требования к программе, перенесем на автоматически появившуюся при запуске Delphi пустую форму с наименованием Form 1 уже перечисленные компоненты, как показано на **Рис. 31**.

Замечание. Для Delphi было бы проще привести здесь файл *.dfm, который содержит в текстовой форме все сведения о размещаемых на форме компонентах, однако выбранная форма изложения позволит без особого труда выполнить подобные программы на другом языке программирования.

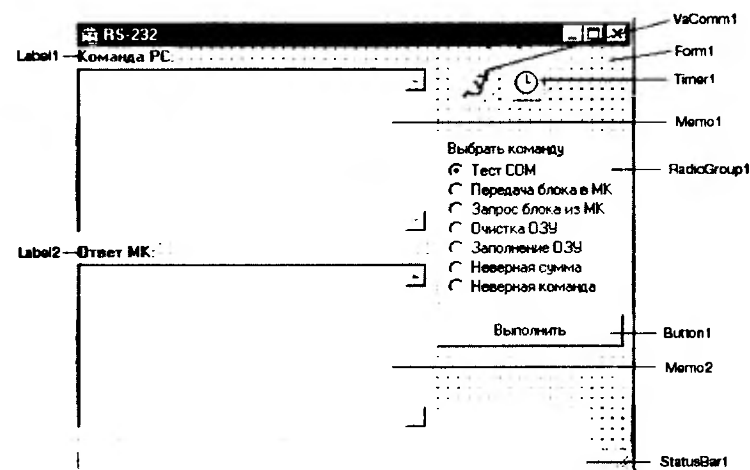


Рис. 31. Форма приложения для связи с микроконтроллером по каналу RS-232

Компоненты Label 1, Label 2, Memo 1, Memo 2, RadioGroup 1 и Button 1 взяты со страницы Standard палитры компонентов, компонент Timer 1 — со страницы System, компонент StatusBar 1 — со страницы Win32, а компонент VAComm1 — со страницы, появившейся в результате установки в Delphi группы компонентов Varian Async32.

Первоначально помещенные на форму компоненты выглядят иначе. Так, новый компонент RadioGroup 1 оказывается пустым. Щелкните по нему мышкой. Сам компонент на форме окажется выделенным, а в окне Object Inspector появится его наименование и список доступных для изменения параметров этого компонента. Найдите свойство Caption и введите вместо значения «RadioGroup 1», установленного по умолчанию, значение «Выбрать команду». В этом же окне перейдите к свойству Items, когда вы щелкнете по нему мышкой, в конце строки справа появится кнопка с тремя точками, щелкните по ней — это приведет к появлению окна String List Editor. Наберите в нем друг под другом следующие наименования кнопок панели RadioGroup 1:

Тест COM
Передача блока в МК
Запрос блока из МК
Очистка ОЗУ
Заполнение ОЗУ
Неверная сумма
Неверная команда

Измените свойство Caption компонентов Label 1, Label 2, Button 1 и самой формы Form 1 в соответствии с приведенным выше рисунком.

В окне Object Inspector установите следующие значения свойств компонента VAComm1, не трогая значения других его свойств.

```
Baudratebr115200
DTRControldtrDisable
MonitorEvents
        ceRxCharTrue
        ceTxEmptyTrue
        ceRxFlagTrue
Options
        coParityCheckTrue
Parity paOdd
PortNum1
ReadBufSize512
ReadTimeout100
WriteBufSize512
WriteTimeout100
```

Для свойств ScrollBars компонентов Memo 1 и Memo 2 выберите значение ssVertica l.

Свойство PortNum (номер порта) должно быть установлено в соответствии с тем, какой COM-порт используется для связи с микроконтроллером. Если по какой-либо причине скорость в 115200 бод не поддерживается, установите меньшую скорость, установите эту же скорость связи для микроконтроллера, внеся соответствующие изменения в его программу. Вероятно, при этом понадобится увеличить значения свойств ReadTimeOut и WriteTimeOut (для Delphi 6 эти свойства отсутствуют).

Теперь добавим обработчики событий для компонентов VAComm1, Timer 1 и Button 1.

В окне Object Inspector выберите из списка компонент Timer 1, перейдите на закладку Events. Для таймера закладка содержит только одно событие OnTimer. Переместите указатель мышки в поле обработчика этого события и выполните мышкой двойной щелчок. В поле автоматически появилось название обработчика Timer 1 Timer, окно программы стало активным, и в нем появился шаблон процедуры обработчика, имеющий вид:

```
procedure TForm1.Timer1Timer(Sender: TObject)
begin
end
```

Для того чтобы назначить обработчик событию приема символа для компонента VAComm1, выберите имя этого компонента из списка в окне Object Inspector, на закладке Events сделайте двойной щелчок мышкой в поле обработчика события OnRxChar. В поле появится наименование обработчика VAComm1 OnRxChar, в окне программы — шаблон этого обработчика.

Нажатие кнопки «Выполнить» вызывается щелчком мышкой по этой кнопке. Этому соответствует событие OnClick компонента Button 1. Назначьте обработчик этому событию.

Замечание. Не выполняйте сохранение программы (Save, Save as) до заполнения шаблонов обработчиков текстом программы. При сохранении пустые шаблоны удаляются, а назначения обработчиков событиям аннулируются. Если сохранение необходимо, введите в каждый шаблон в пустую строку два символа «//», которыми обозначается начало строки комментария.

Сейчас текст программы имеет следующий вид:

```
unit Ucom;
interface
uses Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
                                Dialogs,
VaComm, ComCtrls, ExtCtrls, VaClasses, StdCtrls;
type
TForm1 = class(TForm)
Memo1: TMemo;
Memo2: TMemo;
RadioGroup1: TRadioGroup;
Button1: TButton;
Timer1: TTimer;
VaComm1: TVaComm;
Label1: TLabel;
Label2: TLabel;
StatusBar1: TStatusBar;
procedure Timer1Timer(Sender: TObject);
procedure Button1Click(Sender: TObject);
procedure VaComm1RxChar(Sender: TObject; Count: Integer);
private
{ Private declarations }
public
{ Public declarations }
end;
var Form1 :TForm1;
implementation
{$R *.DFM}
procedure TForm1.Timer1Timer(Sender: TObject);
begin
end;
procedure TForm1.VaComm1RxChar(Sender: TObject; Count: Integer);
begin
end;
procedure TForm1.Button1Click(Sender: TObject);
```

```
begin
end;
end.
```

Листинг начинается заголовком *unit* (модуль), это значит, что мы работаем с модулем, вызываемым основной программой, сама же программа создается автоматически.

Модуль имеет блок *interface* и блок *implementation* — это стандартное построение модуля. Блок *interface* создан автоматически, его трогать не будем, а все дополнения будем вносить в блок *implementation* ниже следующей строки:

*{$R *.DFM}*

Наберите следующий текст под указанной строкой:

```
const
//Коды команд компьютера:
cmRS        = $20; //ТЕСТ RS-232
cmBlkPC_S   = $21; //ПЕРЕДАТЬ БЛОК 128 байт в МК
cmBlkS_PC   = $22; //ЗАПРОСИТЬ БЛОК 128 байт из МК
cmZero      = $23; //ЗАПОЛНИТЬ ОЗУ нулями
cmAA        = $24; //ЗАПОЛНИТЬ ОЗУ 3-м байтом команды
arsz        = 150; //размер массива данных
Err         = 'Ошибка '; //Постоянная часть сообщения об ошибке
var
aBt    :array[1..arsz]of byte;
wCMD    :integer;
sCMD    :string;
function VAString(var SC:string):string;
var S,S1,S2:string;
I :integer;
begin
while SC[Length(SC)] = ' ' do SetLength(SC,Length(SC) - 1);          //1
S2:= SC;                                                              //2
S:= '';                                                               //3
//Преобразование шестнадцатеричных символов строки в десятичные:
while Length(S2)>0 do                                                 //4
begin
S1:= '$';                                                             //5
I:= 1;                                                                //6
while (S2[I]<>' ') and (I< = Length(S2)) do                           //7
begin
S1:= S1 + S2[I];                                                      //8
Inc(I);                                                               //9
end;                                                                  //10
Delete(S2,1,I);                                                       //11
S:= S + Chr(StrToInt(S1));                                            //12
end;                                                                  //13
Result:= S;                                                           //14
end;                                                                  //15
```


При заполнении шаблонов обработчиков текстом следите за числом строк *begin* и *end.*

После заполнения текстом программы обработчики событий должны приобрести следующий вид:

```
//Обработчики событий:
procedure TForm1.VaComm1RxChar(Sender: TObject; Count: Integer);
begin
Timer1.Interval:= 35;
Timer1.Enabled:= True;
end;
procedure TForm1.Button1Click(Sender: TObject);
var I,Sum:integer;
Trans:boolean;
begin
if not VaComm1.Active then VaComm1.Open;                              //16
wCMD:= RadioGroup1.ItemIndex + cmRS;                                  //17
sCMD:= '5A' + Format(' %.2x',[wCMD]);                                 //18
Sum:= $5A + wCMD;                                                     //19
case RadioGroup1.ItemIndex of                                         //20
0,2,3: sCMD:= sCMD + ' 00 00 00' + Format(' %.2x',[Hi(Sum)]) +
       Format(' %.2x',[Lo(Sum)]);                                     //21
1: begin {Передача блока из PC в ОЗУ МК}
sCmd:= sCmd + ' 00 00';                                               //22
for I:= 1 to 128 do begin                                             //23
Sum:= Sum + I;                                                        //24
sCMD:= sCMD + Format(' %.2x',[I]);                                    //25
end;                                                                  //26
sCMD:= sCMD + Format(' %.2x',[Hi(Sum)]) +
          Format(' %.2x',[Lo(Sum)]);                                  //27
end;                                                                  //28
4: begin
Sum:= Sum + $AE;                                                      //29
sCMD:= sCMD + ' AE 00 00' + Format(' %.2x',[Hi(Sum)]) +
          Format(' %.2x',[Lo(Sum)]);                                  //30
end;
5: sCmd:= '5A 20 00 00 00 00 6A';                                     //31
6: sCmd:= '5A 44 00 00 00 00 9E';                                     //32
end;                                                                  //33
Trans:= VaComm1.WriteText(VAString(sCMD));                            //34
if not Trans then sCMD:= 'Ошибка передачи';                           //35
Memo1.Lines.Add(sCMD);                                                //36
end;                                                                  //37
//====== Button1Click
procedure TForm1.Timer1Timer(Sender: TObject);
var S:string;
MsgLng,I,Sum:integer;
begin
Timer1.Enabled:= false;                                               //38
MsgLng:= VaComm1.ReadBuf(aBt,arsz);                                   //39
```

```
S:= '';                                                         //40
for I:= 1 to MsgLng do S:= S + Format('%.2x',[aBt[I]]) + ' ';   //41
//Проверка первого байта ответа МК
if (MsgLng>0) and (aBt[1]<>$>I$5A) then begin                   //42
S:= S + ' Первый байт <>5A ' + Format('%.2x',[aBt[1]]) + '
                                 ' + IntToStr(MsgLng);          //43
StatusBar1.Panels[0].Text:= Err + ': первый байт в ответе
                                 программатора <>5A';           //44
aBt[2]:= 0;                                                     //45
end;                                                            //46
//Проверка длины ответа
if (MsgLng>0) and (aBt[2]>0) and (MsgLng<>5) and (MsgLng<>134)
then begin                                                      //47
S:= S + ' Неверное число байтов';                               //48
StatusBar1.Panels[0].Text:= Err + ': неверное число байтов в ответе
                                 программатора';                //49
aBt[2]:= 0;                                                     //50
end;                                                            //51
//Проверка контрольной суммы ответа
if (MsgLng>0) and (aBt[2]>0) then begin                         //52
Sum:= 0;                                                        //53
case aBt[2] of                                                  //54
cmBlkS_PC: begin                                                //55
for I:= 1 to MsgLng - 2 do Sum:= Sum + aBt[I];                  //56
if Sum<>aBt[MsgLng - 1]*256 + aBt[MsgLng] then
                                                                //57
S:= S + ' Неверная CS';                                         //58
aBt[2]:= 0;                                                     //59
end                                                             //60
else begin                                                      //61
for I:= 1 to MsgLng - 1 do Sum:= Sum + aBt[I];
                                                                //62
if Lo(Sum)<>aBt[MsgLng] then                                    //63
S:= S + ' Неверная CS';                                         //64
aBt[2]:= 0;                                                     //65
end;                                                            //66
end;                                                            //67
end;                                                            //68
case aBt[2] of                                                  //69
$0C: begin                                                      //70
case aBt[3] of                                                  //71
$E0: StatusBar1.Panels[0].Text:= Err +
            'четности при передаче';                            //72
$E1: StatusBar1.Panels[0].Text:= Err +
            'контрольной суммы';                                //73
$E2: StatusBar1.Panels[0].Text:= Err +
            'команда компьютера не опознана';                   //74
end;                                                            //75
end;                                                            //76
end;                                                            //77
if S<>'' then Memo2.Lines.Add(S);                               //78
```



```
end;                                                            //79
//===== Timer1
//========== Конец обработчиков
end. //Конец модуля                                             //80
```

Для удобства описания программы в конце строк в виде комментария указаны номера операторов программы.

Замечание. При попытке набрать текст программы целиком, без обязательного в Delphi назначения обработчиков, как это было описано выше, программа работать не будет.

5.3.3. Описание работы программы

После выбора команды из списка и нажатия кнопки «Выполнить» событие OnClick компонента Button 1 вызывает процедуру *procedure TForm1.Button1Click(Sender: TObject);*.

Оператор 16 — первый оператор процедуры. Посредством обращения к функции *VAComm1.Active* проверяется, открыт ли COM-порт. Если нет, то вызывается процедура его открытия *VAComm1.Open*.

Оператор 17. Выбор команд обеспечивает компонент RadioGroup 1, а его свойство *ItemIndex* хранит порядковый номер выбранной команды в списке. Команда «Тест COM» имеет нулевой порядковый номер и код cmRS, код каждой следующей команды на единицу больше кода предыдущей (смотрите блок *const* в программе). В результате в переменной *wCMD* оказывается код команды, выбранной пользователем.

Оператор 18. Функция *Format* преобразует код команды в строку, параметр функции ‘ *%.2x*’ требует начать строку пробелом (пробел перед знаком %), после которого будет 2 символа (.2 после знака %), представляющих шестнадцатеричное число (символ x). Второй параметр функции — преобразуемое число. Например, для команды /Тест COM/ переменная wCMD хранит значение $20, тогда результат функции — строка ‘ 20’. В результате операции строка *sCMD* примет вид ‘5A 20’.

Оператор 19. В переменной *Sum* — результат суммирования первых двух байт команды.

Оператор 20. Начало конструкции:

```
case INDEX of
Ind1: оператор 1;
...
IndK: оператор K;
...
IndN: оператор N
end;
```

Ее окончание — оператор 33. Если *INDEX = Ind 1*, выполняется оператор 1. В конструкции выполняется проверка для всех значений Ind 1...Ind N на равенство значению INDEX. Если *INDEX = Ind K*, выполняется оператор K.

Здесь проверяется порядковый номер выбранной команды в списке.

Оператор 21. Для команд с порядковыми номерами 0 и 3 три байта, следующих за кодом команды, равны нулю, поэтому контрольная сумма команды определяется первыми двумя байтами. Функции Format определяют строки в том же формате, что и для оператора 18. Причем при первом вызове функции определяется строка для старшего байта суммы (функция *Hi(Sum)*), при втором вызове — для младшего байта (функция *Lo(Sum)*). Функции *Hi* и *Lo* аналогичны функциям *high* и *low*, использовавшимся при программировании микроконтроллера. В результате строка *sCMD* будет содержать команду «Тест COM» для *ItemIndex* = 0 или команду «Запрос блока из МК» при *ItemIndex* = 3.

Если выбрана команда с порядковым номером 1 («Передача блока в МК»), то выполняется группа операторов с 22-го по 28-й.

Оператор 22. К строке *sCMD* добавляется два нуля, представляющих номер блока. Строка приобретает вид '5A 21 00 00'.

Оператор 23 начинает цикл, который повторится 128 раз, переменная цикла *I* будет изменяться от единицы до 128, окончание цикла — оператор 27. В каждом цикле значение переменной цикла прибавляется к контрольной сумме команды (оператор 24), преобразуется в строку функцией *Format* и добавляется к строке *sCMD* (оператор 25). По окончании цикла в строке оказывается 132 байта.

Оператор 28. Как и в операторе 21, к строке добавляется преобразованная в строковый вид двухбайтная контрольная сумма.

Для команды с порядковым номером 4 («Заполнение ОЗУ») к контрольной сумме добавляется значение байта $AE (оператор 29), выбранного в качестве заполнителя для ОЗУ микроконтроллера. К строке *sCMD* добавляется строковое представление байта-заполнителя, двух нулей и двухбайтной контрольной суммы (оператор 30).

Для команд с порядковыми номерами 5 и 6 с заведомо внесенными ошибками соответствующие им значения строковых констант присваиваются переменной *sCMD* (операторы 31 и 32).

Оператор 34. Сформированная в виде строки команда преобразуется функцией *VAString*. Преобразованная ею команда передается через COM-порт посредством вызова функции *VAComm1.WriteText*. Если передача прошла успешно, функция принимает значение true, в противном случае — false, это значение присваивается булевой переменной *Trans*.

Оператор 35. Если при передаче произошла ошибка, значит, значение переменной *Trans* = false и в переменную *sCMD*, хранившую до этого строку переданной команды, записывается сообщение об ошибке.

Оператор 36. Переданная команда или сообщение об ошибке передачи, хранящееся в переменной *sCMD*, добавляется в окно Memo 1 (окно «Команда PC»).

Строка *sCMD* удобна для просмотра команды пользователем, но не подходит для передачи с помощью функции *VAComm1.WriteText*. Рассмотрим функцию *VAString*, преобразующую строку команды в пригодный для передачи вид.

Оператор 1. Используемые в операторе функции определяют число символов в строке *SC*, процедура *SetLength* устанавливает длину строки *SC* (первый параметр процедуры), равную значению второго параметра. В операторе проверяется, не является ли последний символ строки пробелом, если да, то строка укорачивается на один символ (пробел в конце строки удаляется). Действие конструкции *while ... do ...* продолжается до тех пор, пока в конце строки *SC* не останется ни одного символа пробела.

Обратите внимание на то, что при описании функции (*function VAString(var SC:string):string;*) перед параметром функции *SC* стоит ключевое слово *var*. Это приводит к тому, что при вызове функции изменения внутреннего параметра функции *SC* приведут к изменениям и параметра *sCMD* (как это сделано в программе (*Trans:= VAComm1.WriteText(VAString(sCMD));*). То есть, если в конце строки *sCMD* были пробелы, они исчезнут после обращения к функции *VAString(sCMD)*.

Операторы 2, 3. Строка *SC* копируется в строку *S2*, чтобы дальнейшие преобразования производились над строкой *S2* и не отразились на строке *SC*, а значит, и на строке *sCMD*. Строка *S* пуста.

Конструкция *while* <выражение> *do* <оператор> начинается оператором 4. Пока длина строки *S2* больше нуля, выполняется группа операторов с 5-го по 12-й.

Оператором 7 начинается новая конструкция *while ... do ...*, в которой к строке *S1* добавляются символы из строки *S2* (оператор 7) до тех пор, пока не будет обнаружен пробел (выражение *S2[I]*<>' ') или строка *S2* не закончится (выражение *I*< = *Length(S2)*).

Оператор 11. Скопированный из строки фрагмент удаляется вместе с пробелом.

Оператор 12. Функция *StrToInt* преобразует строку вида '$5A' в число $5A, функция *Char* определяет символ, код которого равен $5A. Полученный символ добавляется к строке *S*.

> Если для команды «Неверная команда» строка *S2* первоначально имела вид '5A 44 00 00 00 00 9E', то в строке *S1* после первой итерации окажется значение '$5A', в строке *S* — символ, код которого равен $5A (это символ 'Z'), а строка *S2* примет вид '44 00 00 00 00 9E'. После второй итерации *S1* = '$44'; *S2* = '00 00 00 00 9E'; *S* = 'ZD' и так далее. То есть исходному фрагменту команды '5A 44 ' соответствует передаваемый с помощью функции *VAComm1.WriteText* в COM-порт фрагмент 'ZD'.
> В Delphi есть и другая форма записи строки. Запись *S*:= #$5A#$44; эквивалентна двум следующим записям:
> *S*:= Char($5A) + Char($44); и *S*:= 'ZD';.
> Значок # в первой записи обозначает, что следующее за ним число $5A — это код символа. Такая запись не берется в кавычки, она используется для записи строчных констант, содержащих неотображаемые на экране символы (такие, как символ #0).

Если микроконтроллер работает нормально, то после получения команды он передаст ответ. Поступление первого байта ответа через COM-порт вызовет событие прием символа. Обработчик этого события *procedure TForm1.VaComm1RxChar(Sender: TObject; Count: Integer);* содержит лишь два оператора. Первый из них устанавливает интервал счета таймера в 35 мс. Такой же интервал был выбран и для приема последовательности в 134 байта в микроконтроллере. Второй оператор запускает счет таймера.

Событием завершения работы таймера вызывается обработчик procedure *TForm1.Timer1Timer(Sender: TObject);*. К этому моменту ответ от микроконтроллера должен быть принят полностью.

Оператор 38 запрещает работу таймера.

Оператор 39. Функцией *VAComm1.ReadBuf* принятые байты ответа помещаются в массив *aBt* размером *arsz*, а их действительное количество помещается в переменную *MsgLng*.

Оператор 41. Принятые байты ответа микроконтроллера преобразуются в строку *S*, аналогичную по структуре строке команды *sCMD*.

Оператор 42. Если первый принятый байт не равен $5A, выполняются операторы 43...45: в строку *S* к принятой последовательности добавляется сообщение об ошибке «первый байт не 5А», дополнительное сообщение об ошибке выводится в компоненте *StatusBar 1*. Во второй байт массива *aBt[2]* записывается ноль, чтобы последующие проверки принятого сообщения не проводились.

Операторы 47...51. Если *aBt[2]* не равно нулю и длина сообщения не равна 5 или 134, в строку *S* добавляется сообщение о неверном числе байтов, сообщение выводится также в *StatusBar 1*, а элемент массива *aBt[2]* очищается.

Оператором 52 начинается проверка контрольной суммы поступившего ответа. Если код поступившего сообщения (второй его байт) равен *cmBlkS_PC*, то поступило 134 байта и контрольная сумма представлена двумя последними байтами.

Оператор 56 — цикл вычисления контрольной суммы для (*MsgLng* – 2) или 132 байта.

Оператор 57 сравнивает вычисленную по 132 байтам сумму с контрольной суммой, переданной в последних двух байтах ответа микроконтроллера. Если эти суммы не одинаковы, к строке *S* добавляется сообщение о неверной контрольной сумме, а элемент массива *aBt[2]* очищается.

Для любого другого ответа микроконтроллера контрольная сумма представлена одним байтом, и операторы 61...65 выполняют проверку контрольной суммы для этих ответов.

Операторы 69 и 70 проверяют состояние *aBt[2]*, если в этом элементе обнаружен код ошибки, переданный микроконтроллером, конструкция *case* (оператор 71) проверяет код ошибки в элементе *aBt[3]*.

Операторы 72...74 выполняются соответственно для кодов ошибок $E0...$E2, помещая определяемое ими сообщение в компонент *StatusBar 1*.

Завершается процедура обработчика выводом строки *S* в окно *Memo 2* (Ответ МК).

5.3.4. Сохранение, запуск, использование программы

После набора программы сохраните модуль Unit 1 и проект Project 1 в удобной вам директории, пользуясь кнопками сохранения или меню File/Save, и запустите выполнение программы кнопкой Run.

Запустив программу, вы можете проверить соответствие команд компьютера и ответов микроконтроллера протоколу обмена. Команда «Передача блока в МК» передаст массив в 128 байт, значения элементов которого изменяются от одного до 128, этот массив записывается в 128 ячеек ОЗУ микроконтроллера. Команда «Очистка ОЗУ» очистит те же ячейки ОЗУ, а команда «Заполнение ОЗУ» заполнит ячейки байтом $AE.

После каждой из этих команд можно прочитать ОЗУ, выполнив команду «Запрос блока из МК».

Две последние команды вызовут передачу микроконтроллером сообщений об ошибках.

5.4. Программа-монитор связи через COM-порты

Если у разработчика нет готовой программы для компьютера, которая могла бы поддерживать связь с микроконтроллером, удобно воспользоваться программой-монитором, которая позволит ввести команду вруч-

ную, определить номер порта и параметры связи. Для этого надо будет освоить процесс создания формы и определения обработчиков событий.

Перенесите на форму компоненты, как показано на **Рис. 32**. Еще не встречавшиеся нам компоненты SpinEdit находятся на странице Samples палитры компонентов, а Edit — на странице Standard.

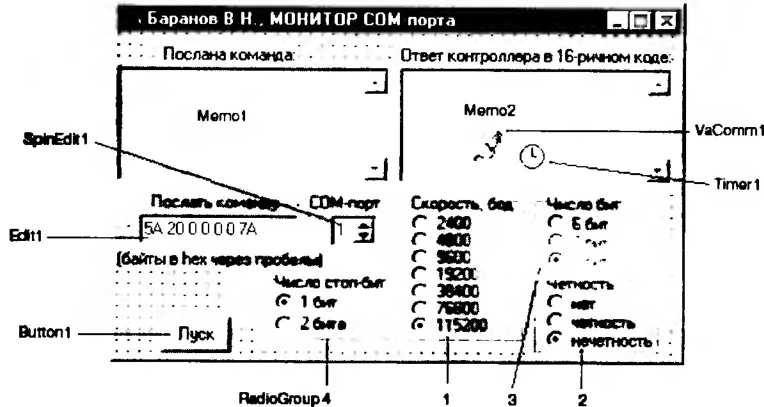


Рис. 32. Форма программы «Монитор»

Соблюдайте порядок установки компонентов RadioGroup и Memo.

Компоненты Label установите самостоятельно и приведите их свойство Caption в соответствие рисунку. Для следующих компонентов установите указанные значения свойств.

Компонент Edit 1:

```
Text   5A 20 0 0 0 0 7A            ;ВУКВЫ «А» – латинские, между
                                   ;байтами – по одному пробелу
```

Компонент SpinEdit 1:

```
Increment          1
MaxValue           4
MinValue           1
```

Компонент Timer 1:

```
Interval           200
Enabled            false
```

Компоненты Memo 1 и Memo 2:

```
ScrollBars         ssVertical
```

Компоненты RadioGroup 1...4:


```
Caption            ;ИЗМЕНИТЬ для каждого в соответствии с Рис. 32
Items              ;ВВЕСТИ списки как на Рис. 32
ItemIndex6         ;Для RadioGroup 1
       2           ;Для RadioGroup 2 и 3
       0           ;Для RadioGroup 4
```

Компонент VAComm1:

```
AutoOpen           false
Baudrate           br115200
Databits           db8
MonitorEvent
ceRxChar           true
ceRxEmpty          true
ceRxFlag           true
Options
coParityCheck      true
Parity             paOdd
PortNum            1
ReadBufSize        4096
ReadTimeout        1000
Stopbits           sbl
WriteBufSize       2048
WriteTimeout       300
```

Теперь свяжем события с их обработчиками, как это делалось для предыдущей программы. События те же. В окне Object Inspector перейдите на закладку Events.

Для компонента Button 1 выполните двойной щелчок мышкой в поле обработчика для события OnClick.

Для компонента Timer 1 повторите действия в поле обработчика OnTimer, а для компонента VAComm1 — в поле обработчика события OnRxChar.

Дополнительно определим обработчик события создания формы. В окне Object Inspector в списке компонентов выберите Form 1 и определите обработчик события OnCreate.

После указанных действий в окне программы появились шаблоны процедур обработчиков определенных нами событий.

В этой программе надо просто заполнить шаблоны обработчиков текстом, после чего программа должна приобрести следующий вид:

```
unit U_RS232;
interface
uses
Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
VaClasses, VaComm, StdCtrls, ExtCtrls, Spin, VaSource, TeEngine, Series,
TeeProcs, Chart;
type
TForm1 = class(TForm)
```

```
Label1: TLabel;
Label3: TLabel;
Label2: TLabel;
Button1: TButton;
SpinEdit1: TSpinEdit;
VaComm1: TVaComm;
Timer1: TTimer;
Memo2: TMemo;
RadioGroup1: TRadioGroup;
RadioGroup2: TRadioGroup;
RadioGroup3: TRadioGroup;
Edit1: TEdit;
Memo1: TMemo;
Label4: TLabel;
Label5: TLabel;
RadioGroup4: TRadioGroup;
procedure Button1Click(Sender: TObject);
procedure VaComm1RxChar(Sender: TObject; Count: Integer);
procedure Timer1Timer(Sender: TObject);
procedure FormCreate(Sender: TObject);
private
{ Private declarations }
public
{ Public declarations }
end;
var
Form1: TForm1;
implementation
{$R *.DFM}
procedure TForm1.Button1Click(Sender: TObject);
var S,S1,S2:string;
I:integer;
Trans:boolean;
begin
case RadioGroup1.ItemIndex of
0: VAComm1.Baudrate:= br2400;
1: VAComm1.Baudrate:= br4800;
2: VAComm1.Baudrate:= br9600;
3: VAComm1.Baudrate:= br19200;
4: VAComm1.Baudrate:= br38400;
5: VAComm1.Baudrate:= br57600;
6: VAComm1.Baudrate:= br115200;
end;
case RadioGroup2.ItemIndex of
0: VAComm1.Parity:= paNone;
1: VAComm1.Parity:= paEven;
2: VAComm1.Parity:= paOdd;
end;
case RadioGroup3.ItemIndex of
0: VAComm1.Databits:= db6;
1: VAComm1.Databits:= db7;
```



```
2: VAComm1.Databits:= db8;
end;
case RadioGroup4.ItemIndex of
0: VAComm1.Stopbits:= sb1;
1: VAComm1.Stopbits:= sb2;
end;
if VaComm1.Active then VaComm1.Close;
VaComm1.PortNum:= SpinEdit1.Value;
VaComm1.Open;
S1:= Edit1.Text;
S:= '';
while S1[Length(S1)] = ' ' do SetLength(S1,Length(S1) - 1);
while Length(S1)>0 do begin
S2:= '$'; I:= 1;
while (S1[I]<>' ') and (I< = Length(S1)) do begin S2:= S2 + S1[I]; Inc(I);
                                                end;
Delete(S1.1,I);
S:= S + Chr(StrToInt(S2));
end;
Trans:= VaComm1.WriteText(S);
if not Trans then Memo1.Lines.Add(' Ошибка передачи')
else Memo1.Lines.Add(Edit1.Text);
end;
procedure TForm1.VaComm1RxChar(Sender: TObject; Count: Integer);
begin
Timer1.Enabled:= True;
end;
procedure TForm1.Timer1Timer(Sender: TObject);
const  arsz = 150;
var S:string;
aBt:array[1..arsz]of byte;
I,J:integer;
begin
Timer1.Enabled:= False;
I: = VaComm1.ReadBuf(aBt,arsz);
S:= '';
for J:= 1 to I do S:= S + Format('%.2x',[aBt[J]]) + ' ';
Memo2.Lines.Add(S);
end;
procedure TForm1.FormCreate(Sender: TObject);
begin
Memo1.Clear;
Memo2.Clear;
end;
end.
```

Если вы поняли, как работает предыдущая программа, то разобраться в программе «Монитор» для вас не составит труда.

Здесь процедуры обработчиков упрошены, а отличительной особенностью является то, что выбранный пользователем режим связи из списков

(скорость, число бит, четность, число стоп-битов, номер COM-порта) анализируется при каждом нажатии кнопки «Пуск» и введенная вручную команда в окошко Edit 1 передается в COM-порт.

Дополнительных комментариев программа не требует.

Вы можете проверить выполнение тех же команд, которые использовались в предыдущей программе. Только будет затруднительно набрать 134 байта для команды /Передать блок в МК/.

5.5. Использование функций Windows API для обращения к COM-порту

Компоненты, поддерживающие связь через COM-порт компьютера, как правило, обращаются к функциям Windows API. Поэтому полезно рассмотреть программу, использующую эти функции непосредственно.

Поскольку основная задача данного раздела — ознакомление с функциями Windows API, программа будет максимально проста.

Разместите на форме компоненты Edit и Button со страницы Standard палитры компонентов Delphi.

В компонент Edit 1 будет вводиться команда, а после приема ответа микроконтроллера сюда же будет выводиться этот ответ.

Для компонента Edit 1 определите значение свойства /Text 5A 20 0 0 0 0 7A/. Это команда /Тест COM/, которая появится в компоненте при запуске программы, и пользователю останется лишь нажать кнопку. Буквы «А» — латинские, между байтами команды должно быть только по одному пробелу.

Для компонента Button 1 определите обработчик события OnClick.

После добавлений, которые следует выполнить, программа должна принять следующий вид:

```
unit UAPI;
interface
uses
Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
                                   StdCtrls;
type
TForm1 = class(TForm)
Edit1: TEdit;
Button1: TButton;
procedure Button1Click(Sender: TObject);
private
{ Private declarations }
public
{ Public declarations }
end;
```



```
var
Form1: TForm1;
implementation
{$R *.DFM}
procedure TForm1.Button1Click(Sender: TObject);
const arsz = 150;
var DCB:TDCB;
hndCOM:THandle;
A,B:array[0..arsz] of char;
Num:cardinal;
I,J:integer;
S,S1,S2:string;
begin
hndCOM:= CreateFile(PChar('COM1'),GENERIC_READ + GENERIC_WRITE,
0,nil,OPEN_EXISTING,FILE_ATTRIBUTE_NORMAL,0);                       //1
DCB.BaudRate:= 115200;                                              //2
DCB.ByteSize:= 8;                                                   //3
DCB.Parity:= 1;                                                     //4
DCB.StopBits:= 2;                                                   //5
SetCommState(hndCOM, DCB);                                          //6
S1:= Edit1.Text;                                                    //7
S:= '';                                                             //8
J:= 0;                                                              //9
while S1[Length(S1)] = ' ' do SetLength(S1,Length(S1) - 1);         //10
while Length(S1)>0 do begin                                         //11
S2:= '$'; I: = 1;                                                   //12
while (S1[I]<>' ') and (I< = Length(S1)) do begin
S2:= S2 + S1[I]; Inc(I); end;                                       //13
Delete(S1.1,I);                                                     //14
A[J]:= Chr(StrToInt(S2));                                           //15
Inc(J);                                                             //16
end;                                                                //17
WriteFile(hndCOM, A, J, Num, nil);                                  //18
ReadFile(hndCOM, B, 5, Num, nil);                                   //19
S1:= '';                                                            //20
for I:= 0 to Num - 1 do                                             //21
S1:= S1 + Format('%.2x',[Ord(B[I])]) + ' ';                         //22
Edit1.Text:= S1;                                                    //23
Button1.Enabled:= false;                                            //24
end;                                                                //25
end.                                                                //26
```

Оператор 1. Производится попытка создать файл 'COM1', если файл создан успешно, параметру *hndCOM*, тип которого *THandle*, присваивается значение описателя файла. Это значение еще понадобится для обращений к функциям Windows API.

Операторы 2...5 записывают параметры COM-порта, соответствующие нашему протоколу обмена, в структуру BDE.

Оператор 6. Вызывается функция установки состояния COM-порта, определенного нами в структуре BDE.

Операторы 7...17 выполняют те же действия, что и рассмотренная ранее функция *VAString* (подраздел 5.4.2), с той лишь разницей, что значения подлежащих передаче через COM-порт символов помещаются не в строку, а в буфер (массив A типа char). Число подлежащих передаче элементов массива оказывается в переменной *J*, оно равно числу байтов в передаваемой команде.

Оператор 18. Вызывается функция записи содержимого буфера A в файл с описателем *hndCOM*, а это, как было определено, COM-порт; количество подлежащих записи элементов *J*; *Num* — количество элементов, которые удастся записать.

Оператор 19. Вызывается функция считывания из файла с описателем *hndCOM* (из COM-порта) в буфер *B*; число считываемых элементов 5; *Num* — количество элементов, которые удастся считать.

Операторы 20...22 преобразуют принятые символы, хранящиеся в массиве *B*, в пригодный для просмотра вид (строка *S*).

Оператором 23 содержимое строки *S* отображается компонентом *Edit 1*.

Оператор 24 запрещает работу кнопки *Button 1*.

Запрещение работы кнопки вызвано тем, что порт уже открыт и инициализирован, а следующее нажатие кнопки вызвало бы новую попытку открыть порт, что привело бы к ошибке. Эти неприятности вы сможете легко устранить самостоятельно.

Основное неудобство при использовании функции *ReadFile* — это необходимость считывания заданного числа элементов. Если же задать заведомо большее число элементов, например не 5, а 150, то программа зависнет.

В описаниях функций Windows API, размещенных в сети Internet, можно найти и другие функции для работы с COM-портом, такие как WriteComm, ReadComm, OpenComm. Работа этих функций мало отличается от действий уже рассматривавшихся функций, а по удобству использования им также не сравниться с компонентами для COM-порта.

Глава 6. **Организация аналоговых выходов для микроконтроллера**

В этой главе рассматриваются два способа преобразования цифровых кодов, с которыми оперирует микроконтроллер, в аналоговый сигнал. В одном из них к микроконтроллеру подключается дополнительная микросхема цифро-аналогового преобразователя (ЦАП), преобразующая цифровые коды в уровни напряжений, во втором используется режим PWM (Pulse Width Modulator — широтно-импульсный модулятор (ШИМ)) таймера T1 микроконтроллера. В обоих случаях выходной сигнал интегрируется (усредняется) по времени.

На **Рис. 33** представлены графики, помогающие понять оба способа цифро-аналогового преобразования.

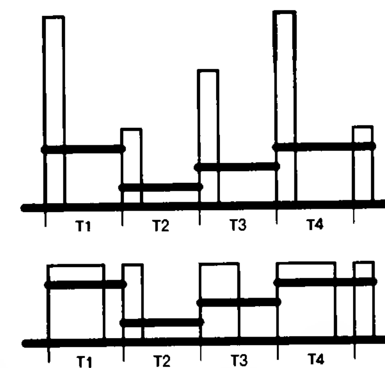


Рис. 33. Усреднение сигналов для двух методов цифро-аналогового преобразования

Импульсные последовательности на обоих графиках имеют одинаковые периоды T1...T4. На верхнем графике код преобразован в амплитуду напряжения при одинаковой длительности импульсов. Для нижнего графика коду соответствует ширина импульса, а амплитуды импульсов одинаковы. При этом площади импульсов в одноименных периодах одинаковы для двух графиков.

Усредненные значения сигнала за период изображены жирными линиями, для обоих графиков они одинаковы.

Верхний график представляет метод преобразования код-амплитуда, нижний — метод код-ширина импульсов или метод широтно-импульсной модуляции (ШИМ).

Для верхнего графика можно выбрать ширину импульса равной длительности периода, тогда жирные линии на этом графике и будут представлять сигнал, который обычно присутствует на выходе микросхемы ЦАП.

6.1. Преобразование кода в ширину импульса

6.1.1. ЦАП и генератор пилообразного напряжения с PWM

Все знакомо в представленной на **Рис. 34** схеме. Пояснения требует только цепочка R_2C_2. Она подключена к контакту PD5, его альтернативное наименование OC1A. Когда микроконтроллер работает в режиме PWM (ШИМ, широтно-иппульсная модуляция), на этом контакте формируется широтно-импульсный сигнал, а цепочка R_2C_2 интегрирует этот сигнал.

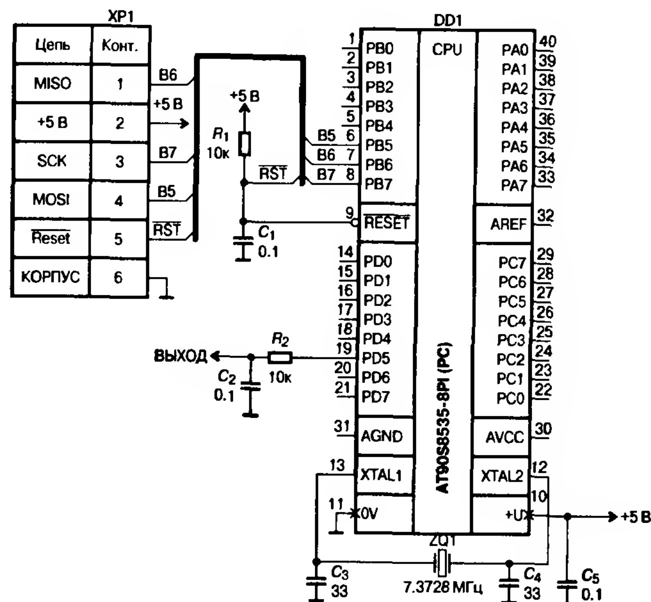


Рис. 34. Схема ЦАП с широтно-импульсным модулятором

Значения сопротивления резистора R_2 и емкости конденсатора C_2 определяются периодом импульсов: чем больше значение произведения R_2C_2, тем меньше пульсаций в выходном сигнале, но при этом снижается скорость изменения выходного сигнала.

6.1.2. Таймер Т1 микроконтроллера в режиме PWM

PWM — это наименование режима таймера Т1 микроконтроллера. В режиме PWM счетчик таймера Т1 выполняет непрерывный счет импульсов, поступающих от предварительного делителя частоты. Как и в обычном режиме, предварительный делитель может быть запрограммирован для деления тактовой частоты микроконтроллера на 1, 8, 64, 256 или 1024.

При подсчете поступающих импульсов содержимое счетчика сначала увеличивается от нуля до предельного значения (счет вверх), а после достижения верхнего значения уменьшается снова до нуля (счет вниз). Когда содержимое счетчика достигает нуля, возникает прерывание переполнения таймера Т1.

Предельное значение, до которого происходит счет, определяется выбранным разрешением. Для разрешения в 8 разрядов предельное значение составляет $2^8 - 1$ или 255, для разрешения в 9 разрядов — 511, для 10-разрядного разрешения — 1023.

Для 10-разрядного разрешения в течение одного цикла при счете вверх поступает 0...1023 импульса (1024 импульса), при счете вниз — 1022...1 импульс (1022 импульса), всего 2046 импульсов.

В микроконтроллере есть две пары регистров сравнения OCR1AH:OCR1AL и OCR1BH:OCR1BL. Содержимое счетчика постоянно сравнивается с содержимым регистров OCR1AH:OCR1AL (или OCR1BH:OCR1BL). При их совпадении логический уровень на контакте OC1A (PD5) (или на контакте OC1B (PD4)) изменится на противоположный. Если состояние счетчика совпало с содержимым регистров OCR1AH:OCR1AL, произойдет прерывание сравнения А. Если совпадение произошло с содержимым регистров OCR1BH:OCR1BL, произойдет прерывание сравнения В. Конечно, эти прерывания должны быть разрешены, также должно быть и обшее разрешение прерываний.

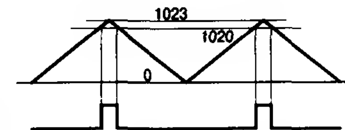


Рис. 35. Формирование импульсов в режиме PWM

На **Рис. 35** изображено изменение состояния счетчика в режиме PWM от нуля до 1023 (счет вверх), а затем снова до нуля (счет вниз).

В регистры сравнения занесено значение 1020. Снизу показана сформированная импульсная последовательность. Прерывание переполнения таймера Т1 происходит, когда состояние счетчика достигает нуля.

6.1.3. Программа для генератора PWM

```
.include "c:\avr\def\m8535def.inc"
.CSEG
.org 0
        rjmp RESET ; Reset Handler
        nop    ;rjmp EXT_INT0 ; IRQ0 Handler
        nop    ;rjmp EXT_INT1 ; IRQ1 Handler
        nop    ;rjmp TIM2_COMP ; Timer2 Compare Handler
        nop    ;rjmp TIM2_OVF ; Timer2 Overflow Handler
```

```
        nop   ;rjmp TIM1_CAPT ; Timer1 Capture Handler
        nop   ;rjmp TIM1_COMPA ; Timer1 CompareA Handler
        nop   ;rjmp TIM1_COMPB ; Timer1 CompareB Handler
        rjmp TIM1_OVF ; Timer1 Overflow Handler
        nop   ;rjmp TIM0_OVF ; Timer0 Overflow Handler
        nop   ;rjmp SPI_STC; ; SPI Transfer Complete Handler
        nop   ;rjmp USART_RXC ; USART RX Complete Handler
        nop   ;rjmp USART_DRE ; UDR Empty Handler
        nop   ;rjmp USART_TXC ; USART TX Complete Handler
        nop   ;ADC Conversion Complete Interrupt Handler
        nop   ;rjmp EE_RDY ; EEPROM Ready Handler
        nop   ;rjmp ANA_COMP ; Analog Comparator Handler
.def    temp  = r16
.def    cnt   = r18
Reset:  ldi   temp,high(RAMEND)
        out   SPH,temp
        ldi   temp,low(RAMEND)
        out   SPL,temp
        ldi   temp,15
        out   WDTCR,temp
        wdr
        ldi   temp,1<<PD5
        out   DDRD,temp
        out   PORTD,temp
        ldi   temp,1<<TOIE1
        out   TIMSK,temp
        ldi   temp,(1<<PWM11) + (1<<PWM10) + (1<<COM1A1) + (1<<COM1A0)
        out   TCCR1A,temp
        ldi   temp,1
        out   TCCR1B,temp
        ldi   temp,1<<SE
        out   MCUCR,temp
        sei
Cycle:  ldi   XL,low(1023)
        ldi   XH,high(1023)
Step:   sleep
        sbiw  XL,1
        brne  Step
        rjmp  Cycle
TIM1_OVF:
        out   OCR1AH,XH
        out   OCR1AL,XL
        wdr
        reti
```

В блоке векторов прерываний устанавливается вектор прерывания переполнения таймера T1. При возникновении прерывания будет вызвана подпрограмма обработки прерывания *TIM1_OVF*.

Меткой *Reset:* начинается инициализация микроконтроллера: определяется стек, устанавливается время срабатывания сторожевого таймера.

Для того чтобы выходной сигнал появился на линии PD5 (OC1A) микроконтроллера, выводом значения 1<<PD5 в порты DDRD и PORTD эта линия организуется как выход, на ней устанавливается ВЫСОКИЙ уровень.

Установкой разряда TOIE1 в регистре TIMSK разрешается прерывание переполнения таймера T1.

Установкой разрядов PWM11 и PWM10 выбирается 10-разрядный режим PWM, а установкой разрядов COM1A1 и COM1A0 определяется положительная полярность выходных импульсов на контакте OC1A, как это изображено на **Рис. 35**.

Выбор коэффициента деления предварительного делителя частоты такой же, как при обычной работе с таймером, запись единицы в регистр TCCR1B означает, что коэффициент деления равен единице, на счетчик таймера T1 подается сигнал с тактовой частотой микроконтроллера (7.3728 МГц).

Установка разряда SE в регистре MCUCR разрешает переход микроконтроллера в режим Idle при выполнении команды *sleep*, а командой *sei* разрешается работа всех прерываний.

Меткой *Cycle:* начинается бесконечно повторяющийся цикл программы. В 16-разрядном регистре X (пара регистров XH:XL) хранится значение, с которым сравнивается состояние счетчика. Первоначально это значение равно 1023, значит, в первом цикле счета ширина импульса будет нулевой (**Рис. 35**).

Команда *sleep* переводит микроконтроллер в режим Idle, который прерывается, когда состояние счетчика таймера T0 достигнет нулевого значения. Тогда вызывается обработчик прерывания *TIM1_OVF* и в регистры сравнения выводится содержимое регистра X, а сторожевой таймер сбрасывается.

После возврата из обработчика прерывания содержимое регистра X уменьшается на единицу. Пока оно не достигло нуля, цикл, начинающийся меткой *Step:*, будет повторяться, а длительность импульса на выходе будет увеличиваться на время, равное двум периодам тактовой частоты микроконтроллера.

Когда в регистре X останется нулевое значение (импульс на линии OC1A при этом самый широкий, его длительность равна 2046 периодам тактовой частоты микроконтроллера), в него вновь будет занесено значение 1023.

Время, за которое ширина импульса на линии OC1A изменится от нуля до максимума, равно длительности одного цикла счета таймера T1, умноженной на число ступенек 1024 (0...1023), при тактовой частоте 7.3728 МГц и единичном коэффициенте предварительного делителя частоты это составит $2046 \cdot 1024 / 7.3728 = 0.284$ с. Значит, частота пилообразного напряжения составит 3.5 Гц.

> **Замечание.** Режим ШИМ можно организовать программно, без использования режима PWM микроконтроллера, так, что частота получаемого сигнала увеличится вдвое при том же разрешении. Для этого счет надо вести только в одну сторону, например от 1023 до нуля. При совпадении содержимого регистров сравнения с содержимым счетчика необходимо устанавливать ВЫСОКИЙ уровень на выходной линии (это может быть любая линия портов ввода/вывода), а при переполнении таймера T1 (при достижении нулевого значения счетчиком) в счетчик снова заносить значение 1023, не останавливая счета, а на выходе устанавливать НИЗКИЙ уровень.

6.2. Преобразование кода в амплитуду импульса

6.2.1. Генератор пилообразного напряжения

Несмотря на невысокое быстродействие микроконтроллера, на его основе можно реализовать генераторы, преобразователи, сумматоры и устройства цифровой обработки аналоговых сигналов. Так как микроконтроллер не обладает аналоговыми выходами, для получения аналоговых сигналов необходимо использовать цифро-аналоговые преобразователи.

Один из возможных вариантов реализации аналогового выхода — использование дополнительной микросхемы ЦАП. Такие микросхемы выпускаются как с параллельной, так и с последовательной загрузкой преобразуемых данных. Последовательная загрузка обычно осуществляется через SPI (Serial Peripheral Interface — последовательный интерфейс для подключения периферийных устройств), при этом микроконтроллер соединяется с микросхемой ЦАП двумя сигнальными линиями. Выбор способа загрузки данных в микросхему ЦАП зависит от конкретной задачи. Параллельный способ загрузки выполняется в несколько раз быстрее последовательного, но требует использования большего числа линий ввода/вывода микроконтроллера, а возможно, и дополнительной микросхемы для буферизации данных.

В настоящем разделе рассматривается несколько устройств, выполненных на основе одной схемы, в которой используется последовательная загрузка данных в микросхему ЦАП через SPI микроконтроллера. Схема устройства приведена на **Рис. 36**.

В схеме используется микроконтроллер ATmega8 (DD1), обладающий встроенным 6-канальным 10-разрядным АЦП. В качестве аналоговых входов АЦП используются линии порта С микроконтроллера.

Микросхема TLC5615 (DA1) корпорации Texas Instruments, представляет собой 10-разрядный ЦАП с интерфейсом SPI. Микросхема подклю-

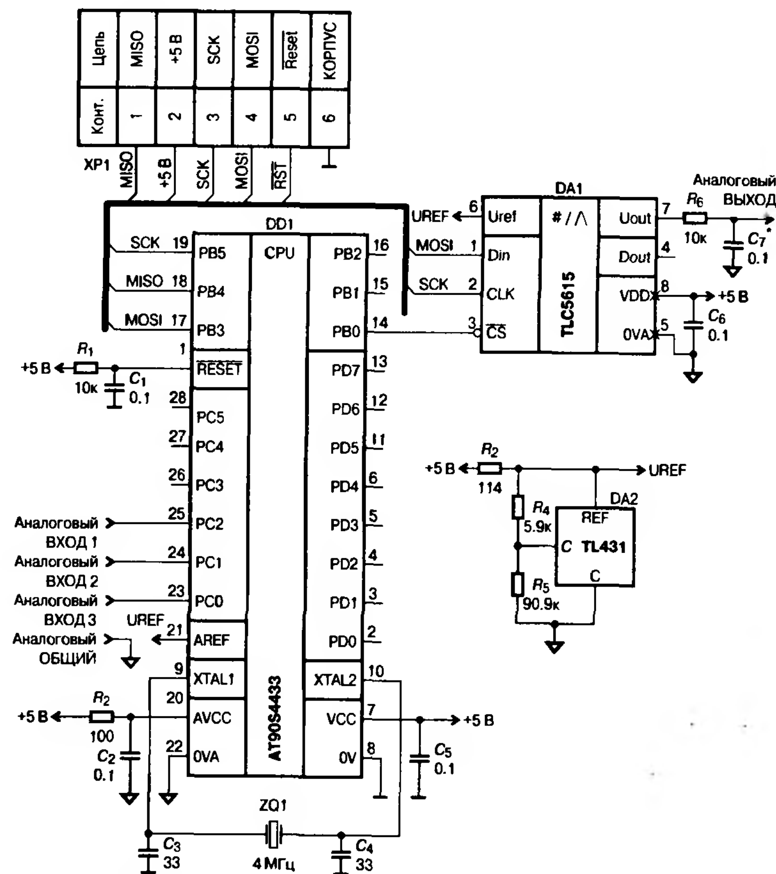


Рис. 36. Подключение микросхемы ЦАП к микроконтроллеру через SPI

чается к линиям SCK и MOSI микроконтроллера. Эти же линии вместе с линиями MISO и RESET используются и для программирования микроконтроллера через разъем XP1. Поэтому в рабочем режиме отключение программатора обязательно. Резистор R_6 и конденсатор C_7, подключаемые к аналоговому выходу микросхемы ЦАП, служат для сглаживания сигнала. Выходное напряжение ЦАП может изменяться от 0 В до уровня опорного напряжения U_{REF}, подаваемого на вход микросхемы ЦАП.

Использование интегрального источника опорного напряжения TL431 (DA2) обеспечивает стабильность опорного напряжения U_{REF}, что снижа-

ет до минимума ошибки преобразования, возникающие при работе АЦП и ЦАП. В этой микросхеме регулировка напряжения осуществляется так, чтобы поддерживать на контакте А напряжение в 2.5 В ($U_{конт.А}$).

Напряжение U_{REF} и сопротивления резисторов R_4 и R_5 связаны выражением: $U_{REF} = U_{конт.А}(1 + R_4/R_5) + I_{конт.А}R_4$. Ток $I_{конт.А}$, протекающий через контакт А микросхемы, составляет 0.01 А.

При использовании указанных на схеме резисторов напряжение U_{REF} составляет примерно 2.7 В.

Опорным напряжением U_{REF}, подаваемым на контакт AREF микроконтроллера, определяется и допустимый диапазон входных напряжений на входах АЦП микроконтроллера PC0, PC1 и PC2. Для того чтобы цифровой код, полученный в результате работы АЦП, соответствовал входному напряжению, это напряжение не должно выходить за границы диапазона $0...U_{REF}$.

Для защиты от напряжения, превышающего напряжение питания микроконтроллера, а также от напряжения ниже 0 В входы микроконтроллера, связанные с АЦП, снабжены ограничительными диодами, как показано на **Рис. 37**. Максимальный допустимый ток через эти диоды составляет 1 мА.

Для проверки работы устройств, реализуемых по приведенной ранее схеме, понадобится подавать переменное синусоидальное напряжение от низкочастотного генератора. Для того чтобы синусоидальное напряжение, изменяющееся в диапазоне от минус U_M до плюс U_M, могло быть подано на вход АЦП микроконтроллера, оно должно быть преобразовано таким образом, чтобы напряжению $-U_M$ на входе АЦП соответствовало напряжение, немного превышающее 0 В, а напряжению $+U_M$ — напряжение, чуть меньшее U_{REF}. Для этого поступающее на вход АЦП напряжение должно быть смещено на величину $U_{REF}/2$. В этом случае можно будет полностью использовать диапазон АЦП. Простейшая схема, реализующая такое преобразование, изображена на **Рис. 38**.

Рис. 37. Защита входов АЦП микроконтроллера от перенапряжения

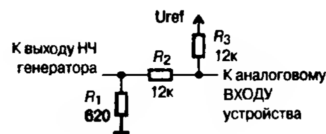


Рис. 38. Устройство смещения напряжения

Эта схема обеспечивает для генератора стандартную нагрузку в 600 Ом. Резисторы R_2 и R_3 представляют собой делитель для сигнала генератора. В то же время они обеспечивают постоянное смещение в точке соединения,

равное половине напряжения U_{REF}. Входное сопротивление со стороны АЦП составляет порядка 100 МОм и практически не оказывает влияния на работу устройства смещения напряжения.

Для того чтобы напряжение смещения было точно равно половине U_{REF}, резисторы R_2, R_3 должны быть прецизионными, выходное сопротивление источника сигнала (в данном случае генератора) близко к нулю, а поступающее напряжение не должно содержать постоянной составляющей.

Прежде всего надо проверить связь микроконтроллера через SPI с микросхемой ЦАП. Проще всего для этого реализовать генератор пилообразного напряжения. Для этого в микросхему ЦАП с небольшим интервалом будем передавать коды, начиная с нуля с приращением в единицу. После достижения кодом максимального значения, а для 10-разрядного ЦАП оно составляет $2^{10} - 1$ (1023), весь цикл повторится. Минимальному коду будет соответствовать нулевое напряжение на выходе ЦАП, максимальному — напряжение U_{REF}.

6.2.2. Программа для генератора пилообразного напряжения

```
;Посылка кода 0,…,1023,0,…,1023,… в 10-разрядный ЦАП TLC5615
.include "c:\avr\def\m8def.inc"
.def temp = r16
.CSEG
.org 0
       rjmp RESET ; Reset Handler
       nop ; EXT_INT0 ; IRQ0 Handler
       nop ; EXT_INT1 ; IRQ1 Handler
       nop ; TIM1_CAPT ; Timer1 Capture Handler
       nop ; TIM1_COMP ; Timer1 compare Handler
       nop ; TIM1_OVF ; Timer1 Overflow Handler
       nop ; TIM0_OVF ; Timer0 Overflow Handler
       rjmp SPI_STC ; SPI Transfer Complete Handler
       nop ; USART_RXC ; USART RX Complete Handler
       nop ; USART_DRE ; UDR Empty Handler
       nop ; USART_TXC ; USART TX Complete Handler
       nop ; ADC ; ADC Conversion Complete Interrupt Handler
       nop ; EE_RDY ; EEPROM Ready Handler
       nop ; ANA_COMP ; Analog Comparator Handler
.MACRO Pause
       nop
       nop
       nop
       nop
.ENDMACRO
RESET: cli
       ldi    r16,low(RAMEND);ИНИЦИАЛИЗАЦИЯ стека
       out    SPL,r16
       ldi    temp,(1<<PB0) + (1<<PB3) + (1<<PB5)
```

```
        out     DDRB,temp           ;НАПРАВЛЕНИЕ передачи линий порта B
        ldi     temp,1<<PB0
        out     PORTB,temp          ;УСТАНОВКА линии PB0
        ldi     temp,((1<<SPIE) + (1<<SPE) + (1<<MSTR))
        out     SPCR,temp           ;ИНИЦИАЛИЗАЦИЯ SPI
        ldi     temp,1<<SE
        out     MCUCR,temp          ;ОПРЕДЕЛЕНИЕ режима SLEEP

        ldi     temp,(1<<WDE) + (1<<WDP2) + (1<<WDP1) + (WDP0)
        out     WDTCR,temp          ;ИНИЦИАЛИЗАЦИЯ сторожевого таймера
        sei
Cycle:  ldi     XH,high(1024*4)
        ldi     XL,low(1024*4)
Step:   sbiw    XL,4
        rcall   Send
        breq    Cycle
        rjmp    Step
SPI_STC:wdr
        reti
Send:   cbi     PORTB,PB0
        Pause
        out     SPDR,XH
        sleep
        out     SPDR,XL
        sleep
        sbi     PORTB,PB0
        ret
```

Приведенная программа начинается блоком векторов прерываний, в котором используется не рассматривавшееся ранее прерывание окончания передачи SPI (далее будем именовать его прерыванием SPI). Как следует из названия, прерывание возникает после окончания передачи данных как данные, записанные в регистр SPDR (регистр данных SPI).

Определение макроса *Pause* начинается директивой *.MACRO* и заканчивается директивой *.ENDMACRO*; содержимое макроса находится между строками с указанными директивами.

Меткой *RESET:* начинается основная часть программы. После инициализации стека определяется направление передачи линий порта B. Линии PB0, PB3 и PB5 определяются как выходные, линия PB1 определяется как вход, остальные линии порта B в работе устройства не участвуют.

На линию PB0, соединенную с контактом CS микросхемы ЦАП, выводится ВЫСОКИЙ уровень, запрещающий работу ЦАП.

Для инициализации SPI в регистре управления SPCR устанавливаются разряды SPIE, SPE и MSTR. Установка разряда SPIE разрешает работу прерывания SPI, установка разряда SPE разрешает работу SPI, а установка разряда MSTR переводит SPI в режим ведущего (Master).

Поскольку остальные разряды регистра SPCR сброшены, то передача по SPI начинается старшим разрядом данных (так как DORD = 0), пауза в передаче данных сопровождается НИЗКИМ уровнем на линии SCK (SPOL = 0), частота тактовых импульсов SPI вчетверо ниже таковой частоты микроконтроллера (SPR1, SPR0 = 0). Состояние разряда SPHA не влияет на работу устройства, так как им определяется прием данных по линии MISO, которая не подключена к микросхеме ЦАП и данные по ней не поступают.

Установкой разряда SE в регистре MCUCR выбирается режим Idle, в который будет переходить микроконтроллер после выполнения команды *sleep*.

Сторожевой таймер запускается на подсчет самого длинного возможного интервала в 1.9 с при напряжении питания 5 В.

Командой *sei* разрешается работа прерываний.

Меткой *Cycle:* начинается бесконечный цикл программы. В регистры XH:XL загружается значение 1024×4. Максимальный 10-разрядный код, который может быть преобразован микросхемой ЦАП TLC5615, выражается числом $2^{10} - 1$ (1024 – 1), но особенность микросхемы такова, что передача должна осуществляться посылкой двух байт, причем 10 информационным битам должны предшествовать 4 нулевых бита, а за информационной посылкой должны следовать два пустых бита. Это означает, что код, подлежащий преобразованию, должен быть сдвинут влево на два разряда, что эквивалентно умножению на четыре.

Перед посылкой данных в регистр SPI его значение уменьшается на 4 командой *sbiw*, значит, максимальное передаваемое значение окажется равным 1023×4. Если в результате выполнения команды содержимое регистра X достигнет нуля, будет установлен флаг Z регистра флагов.

Вызываемая подпрограмма *Send* обеспечивает передачу данных по SPI, для этого первой командой подпрограммы на линии PB0 микроконтроллера устанавливается НИЗКИЙ уровень, разрешающий запись в микросхему ЦАП.

Вызов макроса *Pause*, фактически эквивалентный помещению в программу его содержимого (4 команды *nop*), обеспечивает задержку в 4 такта микроконтроллера перед началом передачи.

Передача начинается посылкой старшего байта в регистр SPDR, после чего командой *sleep* микроконтроллер переводится в режим ожидания.

По окончании передачи прерыванием SPI микроконтроллер выводится из режима ожидания и вызывается обработчик прерывания *SPI_STC*.

В обработчике лишь сбрасывается сторожевой таймер. После возврата из обработчика в подпрограмме *Send* выполняется следующая команда загрузки младшего передаваемого байта в регистр SPDR. Микроконтроллер снова переводится в режим ожидания, а по окончании передачи данных по SPI вызывается обработчик прерывания *SPI_STC*, после возврата из него в

подпрограмме *Send* выполняется команда установки ВЫСОКОГО уровня на линии PB0 микроконтроллера, запрещающая прием данных микросхемой ЦАП.

Команда *breq* вызовет переход на метку *Cycle:* лишь если флаг Z окажется установленным, в противном случае выполнение следующей команды приведет к переходу на метку *Step:*.

То есть передаваемые по SPI данные будут уменьшаться от 1023×4 до нуля, затем в регистр X снова загрузится значение 1024×4 и все повторится.

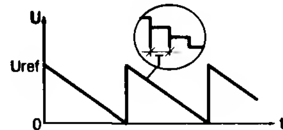


Рис. 39. Выходное напряжение ЦАП

Важно, что ни одна команда подпрограммы *Send* или подпрограммы обработчика прерывания *SPI_STC* не влияют на состояние флага Z, так как эти подпрограммы выполняются между командой *sbiw*, определяющей состояние флага, и командой *breq*, выполнение которой определяется его состоянием.

В результате выполнения программы на аналоговом выходе микросхемы ЦАП сигнал будет иметь вид, представленный на **Рис. 39**.

Напряжение на выходе ЦАП изменяется ступенчато, всего 1024 ступеньки, каждая ступенька удерживается в течение времени T.

Определим период пилообразного напряжения. Для этого надо вычислить интервал времени T.

При инициализации регистра SPCR (регистр управления SPI) тактовая частота передачи по SPI была определена вчетверо меньше тактовой частоты микроконтроллера. Тактовая частота микроконтроллера определяется частотой резонанса использованного в схеме кварцевого резонатора. Она составляет 4 МГц, значит, тактовая частота передачи по SPI равна 1 МГц.

Длительность одного периода этой частоты составляет 1 мкс, поэтому для передачи 16 разрядов данных по SPI требуется 16 мкс.

Можно подсчитать вручную время, требующееся на выполнение команд одного цикла, начинающегося меткой *Step:*, а можно определить его при отладке, воспользовавшись возможностями, открывающимися при использовании окна Processor.

Во втором случае надо помнить, что режим ожидания, вызываемый командой *sleep*, отладчиком не поддерживается и время вызова обработчика прерывания *SPI_STC* надо добавлять вручную, так как после команды *sleep* симулирования прерывания не произойдет. Оно произойдет через 36 тактов микроконтроллера после загрузки регистра SPDR. Это означает, что симуляция работы SPI выполняется, но она не связана с командой *sleep*.

Расчет показывает, что интервал времени T одной ступеньки напряжения равен 27 мкс, а значит, период получаемого на выходе ЦАП сигнал составит $27 \times 1024 = 27648$ мкс, а частота составит $1/0.027648 = 36$ Гц.

Замечание. Если из полученных 27 мкс интервала T выделить время, требующееся на обработку двух прерываний (6 мкс), и время передачи двух байт данных (16 мкс), то остаток составит 5 мкс. Еще на две микросекунды можно сократить интервал за счет отказа от организации подпрограммы и вызова макроса Pause. Остаток (3 мкс) примерно в 9 раз меньше интервала T. Примерно такой интервал T потребовался бы при использовании микросхемы 10-разрядного ЦАП с параллельным способом загрузки данных. Но при этом надо было бы задействовать не менее 11 линий микроконтроллера.

Для микроконтроллера использование метода преобразования кода в амплитуду импульса с выводом кода через SPI дало примерно десятикратное увеличение частоты сигнала по сравнению с методом преобразования кода в ширину импульса. Поэтому второй метод более интересен в практическом плане, и именно он используется в устройствах, которые описаны ниже.

6.2.3. Генератор синусоидального сигнала

Для того чтобы вычислять коды, которые необходимо передавать в микросхему ЦАП для получения синусоидального сигнала, микроконтроллеру понадобится много времени, что в свою очередь резко понизит частоту выходного сигнала. Удобнее воспользоваться табличным методом: записать в память программ микроконтроллера несколько значений функции (в данном случае синуса) при одинаковом приращении времени.

Проще всего определить значения для одного периода. Но если необходимо получить функцию синуса, достаточно определить значения лишь для четверти периода, остальные значения вычисляются по простым формулам.

Вычислить значения синусоиды можно, например, в Delphi.

На **Рис. 40** изображена форма с полученными значениями синусоиды.

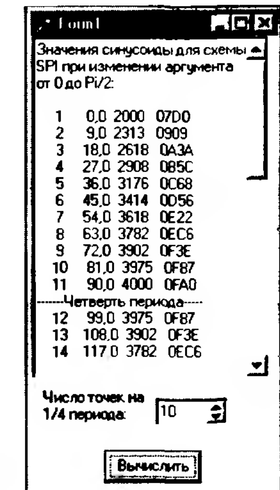


Рис. 40. Форма Delphi с вычисленными значениями синусоиды

На форме размещены уже знакомые вам компоненты, а при нажатии кнопки «Вычислить» (Button1) выполняется следующая процедура:

```
procedure TForm1.Button1Click(Sender: TObject);
var I,Cod:integer;
Step,W:real;
begin
Memo1.Clear;                                                   //1
Memo1.Lines.Add('Значения синусоиды для схемы SPI при изменении
аргумента от 0 до Pi/2:');                                     //2
Memo1.Lines.Add('');                                           //3
Step:= 90/SpinEdit1.Value;                                     //4
W:= 0; I:= 1;                                                  //5
while W<360 + Step do begin                                    //6
Cod:= Round(4*(500 + 500*sin(PI*W/180)));                      //7
Memo1.Lines.Add(Format(' %4d',[I]) +
Format(' %7.1f',[W]) + Format(' %4d',[Cod]) +
Format(' %.4x',[Cod]));                                        //8
Inc(I);                                                        //9
W:= W + Step;                                                  //10
if W = 90 + Step then Memo1.Lines.Add(
'-------Четверть периода-----');                               //11
end;                                                           //12
end;                                                           //13
```

Шаг изменения (в градусах) аргумента Step вычисляется в операторе 4 исходя из того, что на 90 градусов (1/4 периода синусоиды) должно приходиться количество точек, задаваемое в компоненте SpinEdit1.

Код в операторе 7 рассчитывается с учетом особенностей микросхемы ЦАП TLC5615. Для передачи код должен быть умножен на четыре, постоянное смещение должно составлять не более половины от максимального значения кода 1023 (выбрано значение 500), это же значение имеет и амплитуда кода синусоиды, тогда код синусоиды будет изменяться от нуля до тысячи.

Вычисление производится для нуля и для 90 градусов, в связи с этим число точек, помещаемых в память микроконтроллера, должно быть на одну больше, чем задано в компоненте SpinEdit1.

В первой колонке окна выводится номер точки, во второй — значение аргумента в градусах, в третьей — значение кода в десятичном виде, в четвертой — код в шестнадцатеричном представлении, он понадобится при отладке программы в AVR Studio.

Коды для одной четверти периода, которые будут использоваться в программе микроконтроллера, отделены линией от остальных кодов всего периода. Вывод всех кодов периода значительно облегчит проверку программы микроконтроллера в режиме отладки.

6.2.4. Программа для генератора синусоидального сигнала

```
;Посылка кода синусоидального сигнала в 10-разрядный ЦАП TLC5615
.include "c:\avr\def\m8def.inc"
.def   temp = r16
.def   cnt = r17
.CSEG
.org 0
       rjmp RESET ; Reset Handler
       nop ; EXT_INT0 ; IRQ0 Handler
       nop ; EXT_INT1 ; IRQ1 Handler
       nop ; TIM1_CAPT ; Timer1 Capture Handler
       nop ; TIM1_COMP ; Timer1 compare Handler
       nop ; TIM1_OVF ; Timer1 Overflow Handler
       nop ; TIM0_OVF ; Timer0 Overflow Handler
       rjmp SPI_STC ; SPI Transfer Complete Handler
       nop ; USART_RXC ; USART RX Complete Handler
       nop ; USART_DRE ; UDR Empty Handler
       nop ; USART_TXC ; USART TX Complete Handler
       nop ; ADC ; ADC Conversion Complete Interrupt Handler
       nop ; EE_RDY ; EEPROM Ready Handler
       nop ; ANA_COMP ; Analog Comparator Handler
.MACRO Pause
       nop
       nop
       nop
       nop
.ENDMACRO
RESET: cli
       ldi    r16,low(RAMEND); Main program start
       out    SPL,r16
       ldi    temp,0
       out    PORTB,temp
       sbi    PORTB,PB0
       ldi    temp,(1<<PB0) + (1<<PB2) + (1<<PB3) + (1<<PB5)
       out    DDRB,temp
       ldi    temp,((1<<SPIE) + (1<<SPE) + (1<<MSTR))
       out    SPCR,temp
       ldi    temp,1<<SE
       out    MCUCR,temp
       ldi    temp,(1<<WDE) + (1<<WDP2) + (1<<WDP1) + (WDP0)
       out    WDTCR,temp
       sei
Cycle: ldi    ZH,high(Sinus*2)
       ldi    ZL,low(Sinus*2)
       ldi    cnt,11
qrt1:  rcall  Clc
       adiw   ZL,1
       dec    cnt
       brne   qrt1
       ldi    cnt,10
```

```
        sbiw   ZL,4
qrt2:   rcall  Clc
        sbiw   ZL,3
        dec    cnt
        brne   qrt2
        ldi    cnt,10
        adiw   ZL,4
qrt3:   rcall  ClcMinus
        adiw   ZL,1
        dec    cnt
        brne   qrt3
        ldi    cnt,9
        sbiw   ZL,4
qrt4:   rcall  ClcMinus
        sbiw   ZL,3
        dec    cnt
        brne   qrt4
        rjmp   Cycle
SPI_STC:wdr
        reti
Clc:    nop
        nop
        nop
        nop
        lpm
        mov    XH,r0
        adiw   ZL,1
        lpm
        mov    XL,r0
        rcall  Send
        ret
ClcMinus:
        ldi    XH,high(2000*2)
        ldi    XL,low(2000*2)
        lpm
        mov    YH,r0
        adiw   ZL,1
        lpm
        mov    YL,r0
        sub    XL,YL
        sbc    XH,YH
        rcall  Send
        ret
Send:   cbi    PORTB,PB0
        Pause
        out    SPDR,XH
        sleep
        out    SPDR,XL
        sleep
        sbi    PORTB,PB0
        ret
```



```
;Данные для 1/4 периода синусоиды:
Sinus:
.db    high(2000),low(2000)
.db    high(2313),low(2313)
.db    high(2618),low(2618)
.db    high(2908),low(2908)
.db    high(3176),low(3176)
.db    high(3414),low(3414)
.db    high(3618),low(3618)
.db    high(3782),low(3782)
.db    high(3902),low(3902)
.db    high(3975),low(3975)
.db    high(4000),low(4000)
```

За основу для генератора синусоидального сигнала взята программа для генератора пилообразного напряжения, а внесенные изменения касаются основного цикла программы, начинающегося меткой *Cycle:*. В конце программы добавлен блок данных, начинающийся меткой *Sinus:*. Это 11 двухбайтных значений кода для четверти периода синусоиды, полученных с помощью описанной в предыдущем разделе программы Delphi.

Для получения первой четверти периода синусоидального сигнала надо просто извлекать эти данные и передавать по SPI в микросхему ЦАП с помощью подпрограммы *Send*, так же, как это делалось в программе для генератора пилообразного напряжения.

В организации оставшихся трех четвертей периода нам поможет следующая иллюстрация.

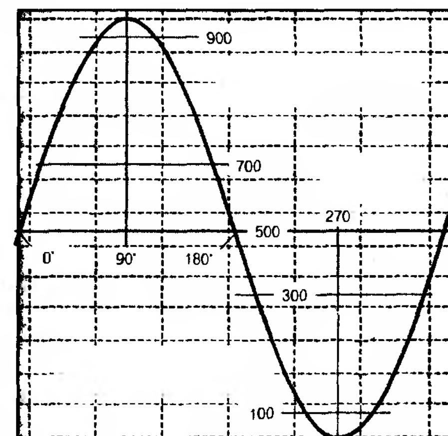


Рис. 41. Фрагмент синусоидального сигнала

Величина изображенного на **Рис. 41** сигнала исчисляется в единицах — это значения цифрового кода сигнала. Сигнал имеет амплитуду 500 и постоянную составляющую, равную 500. Началу сигнала соответствует 0° и значение кода 500.

> **Замечание.** Показанные на рисунке значения кода выбраны для большей наглядности, они не соответствуют реальным значениям, используемым в программе, кроме того, они не увеличены в 4 раза, что сделано в программе в соответствии с требованиями для микросхемы ЦАП TLC5615.

При изменении аргумента от 0° до 90° (первая четверть периода) значение сигнала изменяется в порядке 500, 700, ..., 900, 1000, а в программе из блока данных, начинающегося меткой *Sinus:*, код должен извлекаться в порядке от первого до последнего (одиннадцатого) значения, всего 11 значений.

Для второй четверти периода (до 180°) порядок изменения значения сигнала следующий: 900, ..., 700, 500. Это значит, что для этой части периода нужно извлекать код в порядке от предпоследнего значения (десятого) до первого, всего 10 значений.

Для третьей четверти периода (до 270°) потребуются несложные вычисления. Порядок изменения сигнала: 300, ..., 100, 0. Перепишем этот ряд, используя значения сигнала для первой четверти периода и значение постоянной составляющей: 500 – (700 – 500), ..., 500 – (900 – 500), 500 – (1000 – 500) или 2·500 – 700, ..., 2·500 – 900, 2·500 – 1000. Значит, для 3-й четверти периода надо извлекать код в порядке от второго до одиннадцатого, всего 10 значений, а каждое извлеченное значение вычитать из удвоенного значения смещения.

Для 4-й четверти периода порядок следования кода сигнала следующий: 100, ..., 300. При использовании значений первой четверти периода этот ряд примет вид: 500 – (900 – 500), ..., 500 – (700 – 500) или 2·500 – 900, ..., 2·500 – 700. Для 4-й четверти периода код надо извлекать в порядке от предпоследнего (десятого) до второго, всего 9 значений, и каждое извлеченное значение вычитать из удвоенного значения смещения.

Если бы при организации последней четверти периода извлекалось и первое значение, то при последующем повторе всего периода это значение повторялось бы.

Теперь перейдем к программе и при ее рассмотрении коснемся только происшедших изменений.

Основной цикл программы начинается меткой *Cycle:*, в регистр Z загружается начальный адрес кодов синусоиды в памяти программ, а в счетчик *cnt* — число извлекаемых кодов для первой четверти периода синусоиды.

Меткой *qrt1:* начинается цикл извлечения и вывода кодов через SPI в микросхему ЦАП для первой четверти периода. В вызываемой подпрограмме *Clc* командой *lpm* извлекается старший байт кода, адрес которого — в регистре Z, извлеченный байт помещается в регистр r0.

О четырех командах *nop* будет сказано ниже.

Извлеченное значение кода копируется в регистр XH.

Значение адреса в регистре Z увеличивается на единицу, а командой *lpm* извлекается младший байт кода. Он копируется в регистр XL, а подпрограмма *Send* отправляет код, хранящийся в регистре X (XH:XL) через SPI.

После возврата из подпрограммы *Clc* адрес, хранящийся в регистре Z, увеличивается на единицу, теперь в регистре Z оказывается адрес старшего байта следующего кода. Содержимое счетчика *cnt* декрементируется, и цикл повторяется, пока содержимое счетчика не достигнет нуля.

По окончании первой четверти периода регистр Z указывает на несуществующее 12-е значение кода, а передачу надо начинать с предпоследнего, 10-го значения. А так как каждое значение кода представлено двумя байтами, содержимое регистра Z уменьшается на 4.

Меткой *qrt2:* начинается цикл для второй четверти периода. Отличие его от предыдущего в том, что после извлечения очередного кода адрес в регистре Z уменьшается на 3.

Для 3-й четверти цикл начинается меткой *qrt3:*, а порядок извлечения кодов такой же, как для 1-й четверти периода.

Для 4-й четверти начало цикла обозначено меткой *qrt4:*, порядок извлечения кодов — как для 2-й четверти периода.

В обоих циклах вызывается подпрограмма *ClcMinus*. В этой подпрограмме в регистр X записывается удвоенное значение постоянной составляющей, извлекаемые командой *lpm* значения кодов помещаются в регистр Y, а затем они вычитаются из значений, хранящихся в регистре X. Результат — содержимое регистра X — передается подпрограммой *Send* через SPI в микросхему ЦАП.

Как вы заметили, для организации первой половины периода используется подпрограмма *Clc*, для второй половины — подпрограмма *ClcMinus*. Во второй подпрограмме используется дополнительно 2 команды *ldi* и команды *sub* и *sbc*. Эти команды увеличивают интервал времени, в течение которого напряжение остается неизменным, то есть происходит удлинение ступенек во второй половине периода сигнала. Для компенсации в подпрограмму *Clc* введено 4 команды *nop*.

> **Замечание.** Если команды, связанные с извлечением кодов (кроме самого первого) и с их вычислением, разместить между командой вывода данных в регистр SPDR и командой *sleep*, тогда на эти действия не будет расходоваться время, а максимальная частота генерируемого сигнала будет ограничиваться только передачей по SPI. Это же справедливо и для программы генератора пилообразного напряжения.

6.3. Определение пространственного модуля сигнала

Пусть необходимо определить модуль (длину) вектора по величинам трех его проекций на ортогональные оси X, Y и Z. Напряжения, пропорциональные проекциям x, y и z, поступают от трех датчиков, подключенных к аналоговым входам 1, 2 и 3 схемы, изображенной на **Рис. 37**. Эти напряжения могут изменяться от нуля до U_{REF}. Напряжение выходного сигнала устройства, пропорционального модулю вектора, должно также изменяться от нуля до U_{REF}.

Частота выборки входных сигналов — порядка 1 кГц.

6.3.1. Алгоритм программы

Используем таймер T0 для организации временных интервалов между выборками. Если использовать коэффициент предварительного деления, равный 64, то при отсчете 62 импульсов до переполнения счетчика таймера T0 частота составит около 1.008 кГц. Значит, период работы таймера T0 около 0.001 с или 1000 мкс.

При переполнении счетчика таймера T0 АЦП выполнит три цикла преобразования — по одному для каждого входного сигнала. Выберем коэффициент деления частоты для АЦП равным 16. Тогда тактовая частота АЦП составит 4 МГц/16 = 0.125 МГц. Преобразование АЦП выполняется за 14 тактов, значит, время, необходимое для выполнения преобразований для трех сигналов, составит $14 \times 3/0.125 = 336$ мкс.

При периоде работы таймера T0 1000 мкс на вычисления и вывод сигнала по SPI остается более 650 мкс. По полученным значениям сигналов вычисляется модуль вектора: $M = (x^2 + y^2 + z^2)^{1/2}$.

Определим максимально возможную разрядность суммы $x^2 + y^2 + z^2$. Так как в микроконтроллере 10-разрядный АЦП, то максимальное значение суммы составит $3 \times (2^{10})^2$ или 3×2^{20}, то есть для размещения суммы достаточно трех байт (24 разряда). Для хранения значений x^2, y^2, z^2 также потребуется по три байта. Для получения квадратов воспользуемся операцией умножения: $x^2 = x \times x$, $y^2 = y \times y$, $z^2 = z \times z$.

Для вычисления корня квадратного воспользуемся следующим методом: для первой итерации квадратный корень q_1 произвольного числа N определяется как $q_1 = N/200 + 2$, для последующих итераций квадратный корень $q_i = (N/q_{i-1} + q_{i-1})/2$. Число итераций резко сокращается, когда значение q_1 близко к значению корня. Попробуем уточнить значение q_1.

Рассмотрим двоичные числа /0b1000 0000/ и /0b0100 0000/ (0b — указывает, что число двоичное). Первое из чисел вдвое больше второго, а в двоичном виде они отличаются разрядностью значащих цифр (самым старшим разрядом, содержащим единицу).

Квадрат первого числа в двоичном виде /0b0100 0000 0000 0000/, квадрат второго — /0b0001 0000 0000 0000/.

В первом числе количество значащих разрядов 8, а в квадрате этого числа — 15, для второго числа количество разрядов соответственно равно 7 и 13.

Можно показать, что, если значащие разряды обрезать справа наполовину, получится приближенное значение квадратного корня, отличающегося от реального не более чем в 1.41 (корень из двух) раза.

В программе для этого надо определить разрядность значащих цифр квадрата числа (например, для первого она составляет 15 или 0b1111), разделим ее на два (в двоичном виде сдвинуть вправо 1 раз, получим 0b111 или 7), а затем сдвигаем значение квадрата числа вправо 7 раз.

При этом надо выполнить проверку: если квадрат числа равен нулю или единице, то и корень равен нулю или единице, а вычисления не выполняются.

При вычислении квадратного корня дробная часть будет отбрасываться, а вычисление считается законченным, когда $|q_i - q_{i-1}| \le 1$.

При максимальных значениях x, y и z, равных 1023, величина модуля составит $(1023^2 \cdot 3)^{1/2}$ или $1023 \cdot 1.73$. Так как полученное значение передается по SPI в микросхему 10-разрядного ЦАП, оно не должно превышать 1023. Поэтому перед передачей значение модуля необходимо уменьшить вдвое.

6.3.2. Листинг программы вычисления модуля

Группы команд, обеспечивающие работу АЦП, SPI, таймера T0, подробно рассматривались в предшествующих программах. В данном разделе комментируются только внесенные в них изменения.

Блок 1. Векторы прерываний, инициализация, основной цикл программы

```
.include "c:\avr\def\m8def.inc"
.def   c1 = r1
.def   c2 = r2
.def   c3 = r3
.def   s4 = r4
.def   s3 = r19
.def   s1 = r20
.def   s2 = r21
.def   tm1 = r16
.def   tm2 = r17
.def   Chnl = r18                    ;НОМЕР канала АЦП
.def   cnt = r18                     ;СЧЕТЧИК
.equ   tick = 62
.equ   aADC = $60                    ;АДРЕС для 3 x 2 байта АЦП
.equ   aADC2 = aADC
.equ   aADC1 = aADC + 3
.equ   aADC0 = aADC + 6
```

```
.equ   aSUM = $70              ;АДРЕС 3-байтной суммы квадратов
.equ   adrS3 = aSum
.equ   adrS2 = aSum + 1
.equ   adrS1 = aSum + 2
.equ   aSqr = $80              ;2 байта квадратного корня
.equ   aSq2 = aSqr
.equ   aSq1 = aSqr + 1
.CSEG
.org 0
       rjmp   RESET ; Reset Handler
       nop    ; EXT_INT0 ; IRQ0 Handler
       nop    ; EXT_INT1 ; IRQ1 Handler
       nop    ; TIM1_CAPT ; Timer1 Capture Handler
       nop    ; TIM1_COMP ; Timer1 compare Handler
       nop    ; TIM1_OVF ; Timer1 Overflow Handler
       rjmp   TIM0_OVF ; Timer0 Overflow Handler
       rjmp   SPI_STC; ; SPI Transfer Complete Handler
       nop    ; USART_RXC ; USART RX Complete Handler
       nop    ; USART_DRE ; UDR Empty Handler
       nop    ; USART_TXC ; USART TX Complete Handler
       rjmp   ADC ; ADC Conversion Complete Interrupt Handler
       nop    ; EE_RDY ; EEPROM Ready Handler
       nop    ; ANA_COMP ; Analog Comparator Handler
RESET: cli
       ldi    tm1,low(RAMEND)
       out    SPL,tm1          ;ИНИЦИАЛИЗАЦИЯ стека
       ldi    tm1,(1<<WDE) + (1<<WDP2) + (1<<WDP1) + (WDP0)
       out    WDTCR,tm1        ;ИНИЦИАЛИЗАЦИЯ сторожевого таймера
       wdr
       ldi    tm1,1<<PB0       ;ИНИЦИАЛИЗАЦИЯ порта B
       out    PORTB,tm1
       ldi    tm1,(1<<PB0) + (1<<PB2) + (1<<PB3) + (1<<PB5)
       out    DDRB,tm1
       ldi    tm1,((1<<SPIE) + (1<<SPE) + (1<<MSTR))
       out    SPCR,tm1         ;ИНИЦИАЛИЗАЦИЯ SPI
       ldi    tm1,0
       out    TCCR0,tm1        ;ОСТАНОВИТЬ таймер T0
       ldi    tm1,(1<<TOIE0)
       out    TIMSK,tm1        ;РАЗРЕШИТЬ прерывание таймера T0
       ldi    tm1,256 - tick   ;ПРЕРЫВАНИЯ таймера T0 с
                               ;интервалом 1 мс
       out    TCNT0,tm1
       ldi    tm1,3
       out    TCCR0,tm1
       sei                     ;ГЛОБАЛЬНОЕ разрешение прерываний
Cycle: wdr
       rjmp Cycle
;=================
```

В программе используются прерывание аппаратного сброса *Reset*, прерывание переполнения таймера T0, прерывание окончания передачи по SPI и прерывание окончания преобразования АЦП. В блоке векторов прерываний установлены прерывания, указывающие на соответствующие обработчики *TIM0_OVF, SPI_STC и ADC*.

Меткой *RESET:* начинается группа команд, определяющих режим работы микроконтроллера. В бесконечном цикле программы, начинающемся меткой *Cycle:*, выполняется сброс сторожевого таймера, а выполнение преобразований сигналов, вычислений и передачи кода по SPI вызывается по прерыванию таймера T0.

Блок 2. Обработчики прерываний

```
;Обработчики прерываний:
TIM0_OVF:
       ldi    tm1,($ff - tick)
       out    TCNT0,tm1
       ldi    Chnl,2
       ldi    YL,low(aADC2)
       ldi    YH,high(aADC2)
       ldi    tm1,(1<<ADEN) + (1<<ADSC) +
              (1<<ADIE) + (1<<ADPS2) + (1<<ADPS0)
mADC:  out    ADMUX,Chnl
       out    ADCSR,tm1
       sbr    tm1,1<<SE
       out    MCUCR,tm1
       sleep
       in     tm1,ADCL
       in     tm2,ADCH
       st     Y +,tm2
       st     Y +,tm2
       st     Y +,tm1
       dec    Chnl
       brpl   mADC
       clr    tm1
       out    ADCSR,tm1
       rcall  ClcModul
       lds    XL,aSq1
       lds    XH,aSq2
       rcall  SendSpi
       reti
;КОНЕЦ TIM0_OVF
;-----------
;Обработчик прерывания АЦП:
ADC:   reti
;-----------
;Обработчик прерывания SPI:
SPI_STC:     reti
;-----------
;КОНЕЦ обработчиков прерываний
;============================
```

Первой парой команд обработчика прерываний *TIM0_OVF* производится перезапуск счетчика таймера T0 без его остановки. Это обеспечивает возникновение очередного прерывания переполнения таймера T0 через 1 мс.

Замечание. Поскольку коэффициент предварительного деления частоты для таймера T0 выбран равным 64, то время выполнения команд, которые могут быть выполнены быстрее, чем за 64 такта микроконтроллера, начиная с момента прерывания, не увеличит интервал между прерываниями таймера T0.

Следующая группа из 16 команд обеспечивает работу АЦП. Номер канала хранится в переменной *Chnl*, преобразование выполняется для второго ($АЦП_2$), первого ($АЦП_1$), а затем для нулевого канала ($АЦП_0$). Результат каждого преобразования — два байта данных, но для каждого канала АЦП отводится три ячейки памяти, так как впоследствии в эти же ячейки будет помещен возведенный в квадрат результат преобразования АЦП. Для того чтобы сохранить порядок адресации, старшим байтом результата преобразования АЦП заполняются две ячейки памяти из трех отведенных.

По окончании преобразований вызывается подпрограмма вычисления модуля *ClcModul*, результат работы подпрограммы, сохраненный в ячейках с адресами *aSq1*, *aSq2*, помещается в регистры XH:XL, а вызываемая подпрограмма *SendSpi* обеспечивает передачу содержимого этих регистров по SPI.

Обработчики прерываний *ADC* и *SPI_STC* содержат по одной команде ***reti*** возвращения из обработчика.

Блок 3. Подпрограммы ClcModul и SendSPI

```
;Подпрограммы:
ClcModul:
        ldi    YL,low(aADC2)
        ldi    YH,high(aADC2)
        rcall  Sqr
        rcall  Sqr
        rcall  Sqr
        clr    s3
        clr    s2
        clr    s1
        ldi    YL,low(aADC2)
        ldi    YH,high(aADC2)
        rcall  Sum
        rcall  Sum
        rcall  Sum
;ldi    s3,$2f
;ldi    s2,$e8
;ldi    s1,$03
        sts    aS3,s3
```



```
        sts    aS2,s2
        sts    aS1,s1
        ldi    XL,2
        ldi    XH,0
        clr    c3
        cp     s1,XL
        cpc    s2,XH
        cpc    s3,c3
        brge   Clc1
        mov    XL,s1
        rjmp   Clc2
Clc1:   rcall  Sqrt
Clc2:   ret
;КОНЕЦ ClcModul
;==============
SendSpi:lslXL
        rol    XH
        cbi    PORTB,PB0
        out    SPDR,XH
        ldi    tm1,1<<SE
        out    MCUCR,tm1
        sleep
        out    SPDR,XL
        ldi    tm1,1<<SE
        out    MCUCR,tm1
        sleep
        sbi    PORTB,PB0
        ret
;КОНЕЦ SendSpi
;=============
```

В подпрограмме *ClcModul* организовано вычисление модуля вектора по трем его проекциям x, y и z, представленным результатами преобразований $АЦП_2$, $АЦП_1$ и $АЦП_0$.

Три следующих друг за другом вызова подпрограммы *Sqr* обеспечивают возведение в квадрат результатов преобразований $АЦП_2$, $АЦП_1$ и $АЦП_0$.

Для вычисления суммы квадратов трижды вызывается подпрограмма *Sum*, результат суммирования накапливается в переменных *s3:s2:s1* и сохраняется в трех ячейках памяти с адресами *aS3*, *aS2* и *aS1*.

Замечание. Закомментированные команды загрузки в переменные *s3:s2:s1* значения $2fe803 (или другого меньшего значения) оставлены для отладки и проверки подпрограммы вычисления квадратного корня. Представленная величина — максимально возможное значение суммы квадратов: 3×1023^2.

Подпрограмма вычисления квадратного корня правильно работает, если подкоренное выражение больше единицы. Для этого значение кон-

трольной суммы сравнивается с числом 2. Если значение контрольной суммы оказывается равным нулю или единице, то и значение корня квадратного равно нулю или единице. Оно помещается в регистры XH:XL, в противном случае вызывается подпрограмма вычисления корня *Sqrt*.

Подпрограмма *SendSPI* аналогична использовавшимся в предыдущих программах подпрограммам передачи данных по SPI. При описании алгоритма программы было показано, что полученное значение модуля вектора должно быть уменьшено вдвое. Выводимые значения необходимо умножить на 4 для передачи данных в микросхему TLC5615. В итоге значения, подлежащие выводу по SPI, увеличиваются лишь в два раза посредством сдвига содержимого XH:XL влево.

Блок 4. Подпрограммы Sqr и Sum

```
;Вычисление квадрата числа:
Sqr:  adiw  YL,1
      ld    s2,Y +
      ld    s1,Y +
      mov   c2,s2
      mov   c1,s1
;________________________
;ВЫЧИСЛЕНИЕ произведения 2 байта × 2 байта:
mul2: ldi   cnt,8*2
      clr   s4
      clr   s3
mul21: sbrs s1,0
      rjmp  mul22
      add   s3,c1
      adc   s4,c2
mul22: lsr  s4
      ror   s3
      ror   s2
      ror   s1
      dec   cnt
      brne  mul21
      sbiw  YL,3
      st    Y +,s3
      st    Y +,s2
      st    Y +,s1
      ret
;КОНЕЦ Sqr
;---
;Вычисление суммы квадратов:
Sum:  ld    c3,Y +
      ld    c2,Y +
      ld    c1,Y +
      add   s1,c1
      adc   s2,c2
      adc   s3,c3
```



```
      ret
;КОНЕЦ Sum
;---
```

В подпрограмме *Sqr* для возведения двухбайтного числа в квадрат используется умножение двух одинаковых чисел. Возводимое в квадрат число копируется из ячеек памяти, адрес которых определяется состоянием регистра Y, в пару переменных *s2:s1* и в переменные *c2:c1*.

Меткой mul2: начинается подпрограмма вычисления произведения двухбайтных чисел.

Замечание. Программа оперирует 10-разрядными числами, но на подпрограммы *mul2* и *Div3_2* ограничение десятью разрядами не распространяется.

Алгоритм вычисления произведения разберем на примере умножения однобайтных чисел. Ниже показано, как можно выполнить умножение двоичных чисел — оно очень напоминает умножение десятичных чисел, за исключением итогового суммирования. Как и при десятичном умножении, каждая следующая строка результата сдвигается на одну позицию влево относительно предыдущей строки. Слева от промежуточных результатов показан номер, равный числу сдвигов множимого, справа от них — множитель в виде столбца. Если разряд множителя равен нулю, то стоящее слева от него сдвинутое множимое умножается на ноль (вычеркивается) и в итоговом суммировании не участвует.

ПРИМЕР УМНОЖЕНИЯ ОДНОБАЙТНЫХ ЧИСЕЛ

Число сдвигов множимого	Множимое							1	0	0	1	0	1	1	0	
	Множитель						×	0	0	1	0	0	1	0	1	Мно-житель
0							+	1	0	0	1	0	1	1	0	1
1							~~+~~	~~0~~	~~0~~	~~1~~	~~0~~	~~1~~	~~1~~	~~0~~		
2					+	1	0	0	1	0	1	1	0.			
3					~~+~~	~~0~~	~~0~~	~~1~~	~~0~~	~~1~~	~~1~~	~~0~~				
4				~~+~~	~~0~~	~~0~~	~~1~~	~~0~~	~~1~~	~~1~~	~~0~~					
5		+	1	0	0	1	0	1	1	0						
6		~~+~~	~~0~~	~~0~~	~~1~~	~~0~~	~~1~~	~~1~~	~~0~~							
7	~~+~~	~~0~~	~~0~~	~~1~~	~~0~~	~~1~~	~~1~~	~~0~~								
2-байтный результат																

Заметим, что число разрядов итоговой суммы равно суммарному количеству разрядов множимого и множителя, вычисление произведения можно свести к сдвигу и суммированию, а суммирование не обязательно выполнять после выполнения всех сдвигов.

Рассмотрим один из алгоритмов умножения целых беззнаковых чисел. Для результата умножения двух однобайтных чисел резервируем два байта, старший из них очистим, в младший поместим множитель: 00000000 00100101. Организуем циклы, в которых каждый раз выполняются следующие действия:

- анализируется значение младшего разряда результата (он же младший разряд множителя);
- если это значение равно единице, к старшему байту результата будет прибавляться множимое 10010110;
- результат сдвигается вправо;
- сдвиги повторяются, пока множитель не покинет результат (так как множитель 8-разрядный, число сдвигов, а значит, и циклов равно восьми).

Первый цикл. В помеченном символом ^ младшем разряде результата находится единица, поэтому к старшему разряду прибавляется множимое:

0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1
1	0	0	1	0	1	1	0	M							^

Результат суммирования:

1	0	0	1	0	1	1	0	0	0	1	0	0	1	0	1
							L	M							^

Результат сдвигается вправо (буквой L помечен младший разряд результата, буквой M — старший разряд множителя):

0	1	0	0	1	0	1	1	0	0	0	1	0	0	1	0
								L	M						^

Второй цикл. После первого цикла в младшем разряде результата 0 выполняется только сдвиг вправо. Результат сдвига:

0	0	1	0	0	1	0	1	1	0	0	0	1	0	0	1
									L	M					^

Третий цикл. В младшем разряде 1 выполняется суммирование:

0	0	1	0	0	1	0	1	1	0	0	0	1	0	0	1
1	0	0	1	0	1	1	0		L	M					^

После суммирования:

1	0	1	1	0	1	1	1	1	0	0	0	1	0	0	1

Результат сдвигается вправо:

0	1	0	1	1	0	1	1	1	1	0	0	0	1	0	0
										L	M				^

Для завершения умножения должно быть выполнено еще пять циклов, пока в результате сдвигов вправо результат не вытеснит множитель за пределы разрядной сетки, а младший разряд результата не окажется в нулевом (крайнем правом) разряде сетки.

В подпрограмме *mul2* используется рассмотренный алгоритм умножения. Для результата резервируются переменные *s4...s1*. Два старших байта *s4* и *s3* очищаются, в младшие из ячеек памяти, хранящих результат преобразования АЦП, помещается множитель. В переменные *c2:c1* записывается множимое, а в счетчик *cnt* помещается число, равное количеству разрядов в двухбайтном множителе.

Меткой *mul21* обозначено начало цикла.

Команда *sbrs* проверяет состояние младшего (нулевого) разряда результата, если оно равно единице, следующая команда *rjmp* не выполняется, а множимое (*c2:c1*) добавляется к старшим байтам результата.

Содержимое результата сдвигается вправо, а значение, хранящееся в счетчике *cnt*, декрементируется.

Цикл повторяется до обнуления счетчика.

Адрес, хранящийся в регистре Y, уменьшается на 3, чтобы поместить трехбайтный результат умножения в те же ячейки, из которых извлекался результат преобразования АЦП.

Подпрограмма *Sum* вызывается для суммирования двух трехбайтных чисел, добавляемое значение — возведенный в квадрат результат одного преобразования АЦП — извлекается из ОЗУ микроконтроллера, сумма накапливается в переменных *s3:s2:s1*.

Блок 5. Подпрограммы Sqrt и Div3_2

```
;Вычисление квадратного корня:
Sqrt: ldi    YL,low(aSum)
      ldi    YH,high(aSum)
      ld     s3,Y +
      ld     s2,Y +
      ld     s1,Y +
      ldi    cnt,8*3 + 1
MSB:  dec    cnt
      lsl    s1
```

```
        rol     s2
        rol     s3
        brcc    MSB
        lsr     cnt
        ldi     YL,low(aSum)
        ldi     YH,high(aSum)
        ld      s3,Y +
        ld      s2,Y +
        ld      s1,Y +
MSB1:   lsr     s3
        ror     s2
        ror     s1
        dec     cnt
        brne    MSB1
        sts     aSq2,s2
        sts     aSq1,s1
SqN:    ldi     cnt,8 + 1
        rcall   Div3_2
        andi    s2,$0f
        lds     c1,aSq1
        lds     c2,aSq2
        add     c1,s1
        adc     c2,s2
        lsr     c2
        ror     c1
        lds     XH,aSq2
        lds     XL,aSq1
        sts     aSq2,c2
        sts     aSq1,c1
        sub     XL,c1
        sbc     XH,c2
        breq    SqOK
        sbiw    XL,1
        breq    SqOK
        adiw    XL,2
        breq    SqOK
        rjmp    SqN
SqOK:   ret
;---
Div3_2:lds      s3,adrS3
        lds     s2,adrS2
        lds     s1,adrS1
        lds     c2,aSq2
        lds     c1,aSq1
        clr     s4
        clr     c3
dv:     sbrc    c2,7
        rjmp    dv1
        lsl     c1
        rol     c2
        inc     cnt
```



```
        rjmp    dv
dv1:    cp      s2,c1
        cpc     s3,c2
        cpc     s4,c3
        brcs    dv2
        sub     s2,c1
        sbc     s3,c2
        sbc     s4,c3
        sec
        rjmp    dv3
dv2:    clc
dv3:    rol     s1
        rol     s2
        rol     s3
        rol     s4
        dec     cnt
        brne    dv1
        ret
```

Подпрограмма *Sqrt* вызывается для вычисления квадратного корня. Для получения первого приближенного значения корня числа определяется количество значащих разрядов в этом числе, затем число сдвигается вправо так, что число значащих разрядов уменьшается вдвое.

Для этого в трехбайтную переменную *s3:s2:s1* загружается число, из которого надо извлечь корень, а в переменную *cnt* — число разрядов в трех байтах плюс единица.

В цикле, начинающемся меткой *MSB*, содержимое переменных *s3:s2:s1* сдвигается влево до тех пор, пока единица из старшего значимого разряда числа не выйдет за границу разрядной сетки, при каждом сдвиге содержимое счетчика *cnt* декрементируется.

Для представленного числа:

0	0	1	0	0	1	0	1	1	0	0	0	1	0	0	1	1	0	0	0	1	0	0	1

понадобится выполнить 3 сдвига влево, а счетчик будет содержать номер старшего значимого разряда числа, равный 22.

Команда сдвига вправо *lsr* уменьшает содержимое счетчика вдвое.

В переменные *s3:s2:s1* снова загружается исходное число, которое в цикле, начинающемся меткой *MSB1:*, сдвигается вправо, пока счетчик *cnt* не окажется пуст. Для приведенного числа будет выполнено 11 сдвигов вправо. Два байта результата записываются в ячейки памяти, предназначенные для хранения корня.

Меткой *SqN:* начинается цикл определения последующих приближенных значений корня.

Исходное число делится на последнее приближенное значение корня. Для этого вызывается подпрограмма *Div3_2*, выполняющая деление

3-байтного числа на 2-байтное. Целочисленный результат такого деления не короче одного байта. Перед вызовом подпрограммы в счетчик *cnt* помещается число разрядов в одном байте плюс единица. После работы подпрограммы результат оказывается в 10 младших разрядах переменной *s2:s1*, и команда *andi* очищает старшие разряды переменной *s2* от возможного мусора.

Последнее приближенное значение квадратного корня копируется из памяти в переменные *c2:c1*, к нему добавляется результат деления, хранящийся в переменных *s2:s1*, сумма, находящаяся теперь в переменных *c2:c1*, делится пополам посредством сдвига вправо содержимого этих переменных.

Теперь в переменных *c2:c1* — новое приближенное значение корня.

Из памяти в переменные XH:XL копируется предыдущее приближенное значение корня, на его место в памяти записывается новое значение корня.

Новое значение корня вычитается из старого значения. Если в результате трех последующих проверок разница оказывается равной 0, 1 или –1, вычисление корня завершается.

Прежде чем рассматривать подпрограмму *Div3_2*, разберемся, как выполняется деление двоичных чисел.

Ниже приведены 3 примера. Если для умножения выполнялись операции сдвиг — суммирование, то для деления — операции сдвиг — вычитание.

ПРИМЕР 1

		Делимое								Делитель				
		1	1	0	0	1	1	0	0	1	1	1	1	
Вычитания нет: остаток 0 < делителя		~~1~~	~~1~~	~~1~~	~~1~~					0	1	1	0	1
Остаток 1:		1	1	0	0	1				Частное				
	–		1	1	1	1								
Остаток 2:			1	0	1	0	1							
		–		1	1	1	1							
Остаток 3:				0	1	1	0	0						
Вычитания нет			–		~~1~~	~~1~~	~~1~~	~~1~~						
Остаток 4:					1	1	0	0	0					
				–		1	1	1	1					
Остаток деления:						1	0	0	1					

Как видно из приведенного примера, деление двоичных чисел похоже на деление десятичных чисел. В приведенном примере проблем не возникает. Если делитель превышает остаток, вычитание не производится. В этом случае единицы в нашем примере зачеркнуты. Тогда в остаток сносится разряд справа, а делитель смещается вправо. После этого производится вычитание делителя из остатка.

Необходимо отметить, что только Остаток 3 в приведенном примере имеет одинаковую с делителем разрядность, Остаток 1, Остаток 2 и Остаток 4 имеют большую разрядность. Этот факт важен для алгоритма деления.

ПРИМЕР 2

```
  1  1  0  0  1  1  0  0  0  0  0  1
– 0  0  0  1
  1  0  1  1
```

Здесь первый остаток оказывается больше делителя, что делает невозможным дальнейшее вычисление без дополнительных мер.

Подобная неприятность при делении десятичной дроби на десятичную дробь устранялась тем, что делимое и делитель одновременно умножались на десять (сдвигались на один десятичный разряд влево), пока оба числа не становились целыми.

Для процедуры деления двоичных чисел поступим аналогично: оба числа будем сдвигать влево, пока в делителе единица не окажется в старшем разряде, а пустые места справа заполним нулями.

ПРИМЕР 3

```
  1  1  0  0  1  1  0  0  0  0  0  1  0  0  0
– 1  0  0  0                       1  1  0  0  1  1  0  0
  0  1  0  0  1
  –  1  0  0  0
     0  0  0  1  1  0  0
           –  1  0  0  0
              0  1  0  0  0
              –  1  0  0  0
                          0
```

После трех сдвигов делимого и делителя влево операция деления выполняется без труда.

Предлагаемый здесь алгоритм деления несколько отличается от обычно использующёгося алгоритма с восстановлением остатка. Он выполняется немного медленнее, но его работа более близка к действиям, выполняемым человеком при делении чисел.

К переменным, хранящим делимое и делитель, независимо от их длины, слева добавляется по одному байту, перед началом деления добавленные байты очищаются. Вернемся к Примеру 1: там остаток 0 оказался меньше делителя. Значит, делимое надо сдвинуть влево относительно делителя и произвести вычитание. Но если слева от делимого нет хотя бы одного разряда, то вместо Остатка 1, равного 11001, получится остаток 1001, результат деления будет неверным. Именно для этого слева от делимого добавляется очищенный байт.

В счетчик циклов заносится разница между числом разрядов делимого и делителя плюс единица.

Делитель сдвигается влево, пока в его старшем разряде не окажется единица, при каждом сдвиге содержимое счетчика инкрементируется.

Эта операция эквивалентна сдвигу делимого и делителя влево. Но если сдвиг делителя не требует увеличения его разрядности (справа добавляется столько нулей, сколько их удалилось слева), то при сдвиге делимого могут потребоваться дополнительные разряды справа. Вместо этого мы увеличим содержимое счетчика, что приведет к увеличению числа операций вычитания — сдвиг без наращивания разрядности делимого.

Результат помещается в переменные, хранящие делимое, по мере сдвига содержимого переменных влево. Это происходит так же, как при умножении: один разряд делимого при сдвиге выходит за границу разрядной сетки слева, освобождая место справа для одного разряда результата.

В предлагаемом алгоритме перед вычитанием выполняется сравнение остатка и делителя. Если остаток больше делителя, выполняется вычитание, если меньше, вычитание не выполняется. Делимое сдвигается влево в любом случае.

> **Замечание.** В алгоритме деления с восстановлением остатка вычитание производится в каждом цикле. Если результат оказывается отрицательным, делимое восстанавливается либо суммированием, либо из памяти, либо из дополнительных переменных, хранящих предшествующее значение делимого.

Повторение циклов вычитания — сдвиг завершается после обнуления счетчика.

В подпрограмме *Div3_2* делимое копируется из памяти в переменные *s3:s2:s1*, делитель — в переменные *c2:c1*. Переменные *s4* и *c3* очищаются и добавляются к делимому и делителю слева.

Теперь делитель составляют переменные *c3:c2:c1*, делимое — переменные *s4:s3:s2:s1*. В процессе выполнения подпрограммы переменные *s4:s3:s2:s1* будут хранить промежуточные остатки и результат.

Меткой *dv:* начинается цикл сдвига влево делителя до появления единицы в старшем (7-м) разряде переменной *c2*. При каждом сдвиге содержимое счетчика *cnt* инкрементируется.

Меткой *dv1:* обозначено начало цикла вычитания — сдвига.

Переменные *s4:s3:s2* и *c3:c2:c1* сравниваются, если вторая из них (делитель) больше первой, флаг С в регистре флагов оказывается установленным, вычитание не выполняется, так как происходит переход на метку *dv2:*. Если же флаг С сброшен после сравнения переменных, вычитание выполняется.

Отметим, что состояние флага С противоположно значению, которое мы должны добавить к частному: если вычитание выполнить нельзя, то к частному добавляется ноль (а С = 1), если вычитание производится, то к частному добавляется единица (С = 0).

Это добавляемое к частному значение получаем инвертированием состояния флага С (командой *sec*, если С = 0, или командой *clc* при С = 1) перед сдвигом делимого, начинающимся меткой *dv3:*. В процессе сдвига делимого новое значение флага С, как очередной разряд результата деления, оказывается в младшем разряде делимого.

Выполнение цикла продолжается до достижения счетчиком *cnt* нулевого значения.

Следует заметить, что результат не полностью вытесняет остаток из переменной *s4:s3:s2:s1*, в нашей программе результат занимает 10 младших разрядов переменной или младший байт *s1* и два младших байта переменной *s2*, поэтому после выполнения подпрограммы старшие 6 разрядов переменной *s2* очищались (смотрите подпрограмму *Sqrt*).

6.4. Цифровой фильтр

Основными узлами цифрового фильтра являются:

- устройство выборки аналогового сигнала;
- АЦП, преобразующий подлежащий фильтрации аналоговый сигнал в цифровой код;
- память, хранящая несколько последних выборок сигнала;
- вычислительный блок, в котором по этим выборкам рассчитывается выходной код отфильтрованного сигнала;
- ЦАП, преобразующий выходной код в аналоговый сигнал.

Все перечисленные узлы уже имеются в схеме, представленной на **Рис. 19** (устройство выборки аналогового сигнала входит в состав АЦП микроконтроллера). Сами выборки сигнала в виде цифрового кода могут

храниться в ОЗУ микроконтроллера. Необходимые вычисления выходного кода производятся в самом микроконтроллере, а преобразование выходного кода в аналоговый сигнал осуществляет микросхема ЦАП TLC5615.

На основе такой схемы разработаем низкочастотный цифровой фильтр Чебышева 4-го порядка, частота среза полосы пропускания фильтра 30 Гц, неравномерность частотной характеристики в полосе пропускания 0.5 дБ, частота выборки аналогового сигнала 1024 Гц.

Вопросы цифровой фильтрации рассматриваются в специальной литературе, здесь же отметим, что представленной информации достаточно для определения коэффициентов фильтра.

Сами коэффициенты, а также характеристики фильтра можно получить на странице

http://www-users.cs.york.ac.uk/~fisher/mkfilter/trad.html

На этой же странице можно получить и формулы для вычисления очередного кода отфильтрованного сигнала. Для фильтра Чебышева 4-го порядка с указанными параметрами эти формулы имеют вид:

$GAIN = 40929.18778;$

$x[0] = x[1]; x[1] = x[2]; x[2] = x[3]; x[3] = x[4];$

$y[0] = y[1]; y[1] = y[2]; y[2] = y[3]; y[3] = y[4];$

x[4] = <значение очередной выборки>*/GAIN;*

$$y[4] = (x[0] + x[4]) + 4\cdot(x[1] + x[3]) + 6\cdot x[2] + (-0.8022616140\cdot y[0]) + (3.3574942168\cdot y[1]) + (-5.3028655427\cdot y[2]) + (3.7472420208\cdot y[3]).$$

Из формул следует, что для определения фильтра низкой частоты 4-го порядка выходной код y[4] определяется значениями четырех предыдущих выборок входного сигнала *(x[0]...x[3])*, четырьмя предыдущими выходными кодами *(y[0]...y[3])* и значением новой выборки входного сигнала x[4].

Значения *x[0]...x[3]* и *y[0]...y[3]* хранятся в памяти, причем самые старые из них *x[0]* и *y[0]*.

Вычисление начинается с уничтожения самых старых значений выборки входного сигнала *x[0]* и выходного кода *y[0]*, на их место записываются более свежие значения *x[1]* и *y[1]*, остальные значения также сдвигаются из переменной *x[i]* в *x[i – 1]*, из *y[i]* в *y[i – 1]*. Новое значение выборки, разделенное на коэффициент *GAIN*, записывается в переменную *x[4]*.

По последним значениям *x[0]...x[4]* и *y[0]...y[3]* определяется новое значение выходного кода *y[4]*, которое сохраняется в памяти и посылается в ЦАП, преобразующий его в аналоговый сигнал.

При необходимости такое вычисление можно организовать на ассемблере. Для этого даже не обязательно выполнять вычисления с плавающей запятой: все коэффициенты можно домножить на константу так, чтобы они стали целыми, учесть эту константу вместе с коэффициентом GAIN при вычислении значения *x[4]* по значению выборки.

Однако проше воспользоваться пакетами, позволяющими программировать микроконтроллеры AVR серии ATmega на языках более высокого уровня, чем ассемблер. Для этого подойдет, например, пакет IAR Embedded Workbench для AVR. Программу в IAR Embedded Workbench можно написать на языке программирования С. Кроме того, поставляемая вместе с компилятором С библиотека содержит массу стандартных функций, что значительно упрощает написание программ.

К сожалению, бесплатно распространяется лишь демо-версия программы, имеюшая существенные ограничения по объему кода, записываемого в память программ микроконтроллера. Эту версию можно найти на сайте *http://www.iar.com/*, на страничке http://www.iar.com/Products/EW/EW_Product.asp?name=EWAVR&manufacturer=Atmel находится информация о том, что нужно сделать, чтобы скачать демо-версию программы.

С учетом вышесказанного здесь приведен пример программы на С, написанной для микроконтроллера AVR серии ATmega, с тем, чтобы продемонстрировать, насколько проще организовать вычисления на языке программирования более высокого уровня, чем ассемблер.

6.4.1. Листинг С-программы цифрового фильтра

```
#include <c:\iar\ \inc\iom8.h>
#include <c:\iar\ \inc\ina90.h>
#include <c:\iar\ \inc\stdio.h>
#include <c:\iar\ \inc\math.h>
#define SPIE (0x80) /*for SPI's SPCR*/
#define SPE (0x40)
#define DORD (0x20)
#define MSTR (0x10)
#define CPOL (0x08)
#define CPHA (0x04)
#define CPR1 (0x02)
#define CPR0 (0x01)
#define SE (0x20) /*for MCUCR sleep enable*/
#define WDE (0x80) /*for WDTR*/
#define WDP2 (0x40)
#define WDP1 (0x20)
#define WDP0 (0x10)
#define TOIE0 (0x02)
#define TOV0 (0x02)
#define ADEN (0x80)
#define ADSC (0x40)
#define ADFR (0x20)
#define ADIF (0x10)
#define ADIE (0x08)
#define ADPS2 (0x04)
#define ADPS1 (0x02)
```

```
#define ADPS0 (0x01)
#define ADCNum (0x00)
/*Конец определения наименований разрядов регистров микроконтроллера*/
#define KZ 3
/*1024Hz 30Hz cutting, Gain = 40928/4 = 10232*/
#define K_Hz 256 - 107
#define Gain 10232
#define ky0 - 0.8022616140
#define ky1 3.3574942168
#define ky2 - 5.3028655427
#define ky3 3.7472420208
/*512Hz 30Hz cutting, Gain = 2832/4 = 708*/
unsigned char bh,bl;
unsigned char *il,*ih;
void *ptri;
unsigned short i;
signed long j;
signed short k;
float y0,y1 = 1449505,y2 = 1449505,y3 = 1449505,y4,z;
unsigned short x0,x1 = 512,x2 = 512,x3 = 512,x4,x;
#pragma vector=SPI_STC_vect
__interrupt void SPI_interrupt(void)
{
}
#pragma vector=ADC_vect
__interrupt void ADC_interrupt(void)
{ADCSR = 0;
}
#pragma vector=TIMER0_OVF_vect
__interrupt void Tim0_interrupt(void)
{TCNT0 = K_Hz;
TCCR0 = KZ;
_WDR();
_SEI();
ADCSR = ADEN + ADSC + ADIE + ADPS2 + ADPS1;
MCUCR = SE;
_SLEEP();
bl = ADCL;
bh = ADCH;
bh = bh&0x03;
x0 = x1;
x1 = x2;
x2 = x3;
x3 = x4;
x4 = bl + 256*bh;
y0 = y1;
y1 = y2;
y2 = y3;
y3 = y4;
x = x0 + x4 + 4*(x1 + x3) + 6*x2;
y4 = (ky0*y0 + ky1*y1 + ky2*y2 + ky3*y3 + x);
```



```
j = y4;
j = j/Gain;
i = j;
PORTB = 0;
SPDR = *ih;
_SLEEP();
SPDR = *il;
_SLEEP();
PORTB = 1;
}
void main(void)
{
ptri = &i;
ih = ptri;
il = ih++;
WDTCR = WDE + WDP2 + WDP1 + WDP0;
DDRB = 0x2D; /*Init PortB for SPI work SCK + MOSI + SS + PB0 =
                                 00101101 = 0x2D — outputs*/
_CLI();
PORTB = 1;
SPCR = SPIE + SPE + MSTR; /* + CPR1 + CPR0;*/
ADMUX = ADCNum;
TIMSK = 2;
TIFR = 2;
TCNT0 = K_Hz;
TCCR0 = KZ;
_SEI();
cycle:
_WDR();
goto cycle;
}
```

Как и в AVR Studio, листинг начинается директивами *#include*, подключающими к проекту стандартные библиотеки. Имена папок, в которых находятся подключаемые файлы, необходимо установить в соответствии с тем, где эти файлы окажутся установленными на вашем компьютере, вы можете разыскать эти папки, воспользовавшись поиском (Пуск/Найти/Файлы и папки, ввести, например, ina90.h). Первый подключаемый файл определяет наименования регистров, векторов прерываний микроконтроллера ATmega8, второй файл содержит функции, общие для микроконтроллеров серии ATmega. Третий и четвертый файлы не используются в данной программе, но часто используются в других программах, длину кода они не увеличивают, поэтому нет смысла убирать их. Третий файл содержит стандартные функции ввода/вывода языка С, четвертый файл содержит разнообразные функции, в том числе математические.

> **Замечание.** Поочередно закомментировав директивы #include, можно откомпилировать программу и по сообщениям об ошибках определить, какие именно функции и определения содержат подключаемые файлы библиотеки.

Подключаемые файлы для С-программы не содержат наименований отдельных разрядов регистров микроконтроллера, как это было в подключаемых файлах *.inc для ассемблера. Для удобства программирования директивами *#define* имена разрядов определяются такими же, какими они были определены в техническом описании микроконтроллера и в файле *.inc для ассемблера.

Последующие определяемые константы нам известны:

- *KZ*, равное трем, — это значение, загружаемое в регистр TCCR0 микроконтроллера для установки коэффициента предварительного деления частоты (64), подаваемой на таймер T0;
- *K_Hz*, равное 256 – 107, — значение, загружаемое в регистр TCNT0; загрузка констант *KZ* и *K_Hz* в указанные регистры обеспечивает частоту выборки входного сигнала в 1024 Гц;
- константа *GAIN* определяется вчетверо меньшей, чем значение, полученное при расчете. Это позволит отправлять в микросхему ЦАП выходной код без дополнительного умножения его на четыре (как было сказано выше, 10-разрядный код, посылаемый в микросхему TLC5615, надо умножать на 4);
- константы *ky0...ky3* — это расчетные коэффициенты фильтра, на которые в формуле домножаются значения *y[0]...y[3]* соответственно.

Дальше определяются типы переменных, а значения *y1...y3*, *x1...x3*, соответствующие значениям выходных кодов *y[1]...y[3]* и выборок *x[1]...x[3]* соответственно, определяются предварительно. Значения выборок приравниваются среднему между максимально и минимально возможным значением выборки: (0 + 1023)/2, что приблизительно составляет 512. Значения кодов получены расчетным путем для указанных значений выборок без учета коэффициента *GAIN* (об этом будет сказано далее). Без предварительного определения эти значения определяются как нулевые, что приводит к получению выходного кода, значительно отличающегося от реального в нескольких начальных выборках, а также к скачку напряжения на выходе фильтра после включения.

Подпрограмма *SPI_interrupt* обработки прерывания окончания передачи по SPI вызывается вектором прерывания *[SPI_STC_vect]*. Никаких операций в подпрограмме не выполняется.

Подпрограмма *ADC_interrupt* обработки прерывания окончания преобразования АЦП вызывается вектором прерывания *[ADC_vect]*. В регистр ADCSR управления АЦП заносится нулевое значение.

Подпрограмма *Tim0_interrupt* обработки прерывания переполнения таймера T0 вызывается вектором прерывания *[TIMER0_OVF0_vect]*.

Записью в регистр TCNT0 константы *K_Hz* таймер T0 перезапускается на счет следующего интервала времени между выборками АЦП.

Функция *_WDR()* сбрасывает сторожевой таймер микроконтроллера, а функция *_SEI()* устанавливает глобальное разрешение аппаратных прерываний, как команды *wdr* и *sei* из списка команд ассемблера.

Записью в регистр ADCSR значения, необходимого для запуска АЦП в нужном режиме, инициализируется преобразование АЦП, а для ожидания его окончания в регистре MCUCR устанавливается разряд SE и вызывается функция *_SLEEP()*.

После окончания преобразования и выполнения подпрограммы *ADC_interrupt* обработки прерывания АЦП происходит переход к следующему за функцией *_SLEEP()* оператору.

В переменные *bl* и *bh* из регистров АЦП ADCL и ADCH помещаются соответственно младший и старший байты результата преобразования 10-разрядного АЦП.

Старшие 6 разрядов переменной *bh* дополнительно очищаются логическим умножением ее значения на 3 (00000011 в двоичном виде).

Значения выборок и кодов передаются от более новых к старым, вытесняя самые старые значения, хранившиеся в переменных *x0* и *y0*.

Вычисление значения *x*, определяемого только значениями выборок *x0...x4*, выполняется отдельно, так как выборки представлены беззнаковыми целыми числами. Операции с ними выполняются гораздо быстрее и требуют значительно меньше ресурсов микроконтроллера, чем операции с кодами *y0...y4*, представленными в формате с плавающей запятой.

> **Замечание.** Для значения, определенного как беззнаковое целое, компилятором С отводится 2 байта, для значения с плавающей запятой 4 байта, для длинного целого – также 4 байта.

Полученное значение *y4* передается переменной *j*, предназначенной для хранения чисел формата длинное целое. Содержимое переменной *j* делится на коэффициент *GAIN*, а полученный результат передается переменной *i*, определенной как беззнаковое целое.

> **Замечание.** Представление результата в виде беззнакового целого значения обоснованно, так как диапазон выходного кода цифрового фильтра (y4/GAIN) близок к диапазону значений выборок (0...1023). Некоторое превышение выходным кодом значения 1023 может наблюдаться за счет неравномерности частотной характеристики (0.5 дБ) в полосе пропускания фильтра.

Вероятно, вы обратили внимание на то, что в формуле на коэффициент *GAIN* делятся значения выборок входного сигнала, а в программе деление выполняется при получении выходного кода. С математической точки зрения это не сказывается на результате, но, если значения выборок делить на коэффициент *GAIN*, превышающий 40000, результат всегда будет меньше нуля. Поэтому его надо будет хранить в формате с плавающей запятой. Такое хранение потребует больше памяти микроконтроллера и более длительных вычислений. Решение, принятое в программе относительно деления на коэффициент *GAIN*, позволило хранить выборки в их исходном виде и оперировать с ними как с двухбайтными целыми числами.

Полученное значение выходного кода, хранящееся в переменной *i*, передается по SPI в микросхему ЦАП. Сначала старший байт **ih* отправляется в регистр SPDR, а микроконтроллер переводится в режим Idle вызовом функции *_SLEEP()*. По окончании передачи и обработки прерывания SPI в регистр SPDR передается младший байт **il* выходного кода. Передача сопровождается НИЗКИМ уровнем на линии PB0 микроконтроллера.

Подпрограмма *main* начинается определением связи между переменной *I* и переменными *ptri*, *ih*, *il*, определенными как указатели. Первая переменная определена как указатель на любой объект, независимо от его размера. Переменные *ih* и *il* являются указателями на беззнаковый символ, попросту говоря, на один байт.

Занимаемое указателем место зависит от адресного пространства, в котором указатель оперирует. Для ОЗУ до 256 байт указатель (адрес) занимает 1 байт, для ОЗУ размером от 257 байт до 64 килобайт указатель (адрес) на любую переменную должен быть двухбайтным и так далее. Размеры всех указателей в программе одинаковы, разными могут быть размеры объектов, на которые они указывают.

Операция **ih* означает, что используется объект, адрес которого хранится в *ih*, в данном случае это переменная, определенная как беззнаковый символ.

Операция *&i* определяет адрес объекта *i*.

Операция *ptri = &i* записывает адрес двухбайтной переменной *i* в указатель *ptri*.

Фактически *ptri* хранит адрес старшего из двух байт который и передается указателю на однобайтную переменную *ih*, теперь адрес старшего байта переменной *i* — в указателе *ih*. Операция *il = ih + +* заносит в указатель на однобайтное значение *il* адрес следующего объекта в ОЗУ (младшего байта кода). Теперь **ih* представляет старший байт переменной, **il* — младший байт.

> **Замечание.** Операция *ih = &i* вызовет ошибку компиляции, так как размер переменной *i* не совпадает с размером объекта, на который может указывать переменная *ih*, в качестве посредника и используется указатель на любой объект *ptri*.

В оставшейся части подпрограммы *main*, являющейся аналогом подпрограммы обработчика *RESET* в программах на ассемблере, выполняется обычная инициализация узлов микроконтроллера: сторожевого таймера (регистр WDTCR), порта B (регистры DDRB и PORTB), SPI (регистр SPCR), АЦП (установка номера канала в регистре ADMUX), таймера T0 (регистры TCCR0, TCNT0, TIMSK и TIFR).

Меткой *Cycle:* начинается бесконечный цикл программы. После вызова функции *_WDR()*, выполняющей сброс сторожевого таймера, происходит возврат на метку *Cycle:*.

С-программа цифрового фильтра по структуре мало отличается от написанной на ассемблере программы определения пространственного модуля. Выигрыш состоит в том, что организация вычислений не требует от программиста тех усилий, которые понадобились бы при написании программы на ассемблере.

В справочной системе IAR Embedded Workbench 2.28 есть хороший материал для обучения работе в среде разработки программ на С (Help/Embedded Workbench Guide). Там можно найти все необходимое для создания проекта и его отладки.

Обычно трудности вызывает установка некоторых опций проекта и содержание подключаемого линкером файла *.xcl, определяющего свойства конкретного типа микроконтроллера.

Откройте окно опций проекта: Project/Options.

Слева в окне Options находится окошко Category, содержащее тип опций. Первый из них General. Для этого типа опции надо выбрать модель процессора (Processor configuration) по объему доступного ОЗУ и памяти программ, а также модель памяти (Memory model) для компилятора С.

Для микроконтроллера ATmega8 подойдет такая модель микроконтроллера: -v1,Max 64 Kbyte data, 8 Kbyte code. При этом доступны модели памяти Tiny и Small. Если выбрать модель Tiny, то используемые в программе указатели смогут обращаться к памяти в пределах начальных 256 байт ОЗУ, при выборе модели Small возможно обращение указателей к оперативной памяти до 64K, хотя на самом деле у микроконтроллера может быть ОЗУ объемом 512 байт.

Понадобится также определить тип генерируемого компилятором С файла. Для этого в опциях категории XLINK надо перейти на закладку Output.

Если на панели Format закладки Output выбрать строку Debug info, компилятор С генерирует файл с расширением *.d90. Тогда в C-Spy (отладчик, поставляемый в составе пакета) производится отладка С-программы. Для получения файла *.hex, пригодного для записи в память программ микроконтроллера, на панели Format выберите строку Other, а в окошке Output format из списка выберите тип формата файла mpds-i. В этом случае отладка в C-Spy выполняется на уровне команд ассемблера.

Замечание. Файл *.hex может быть загружен и отлажен в AVR Studio.

Несколько слов о частоте среза цифрового фильтра: этот параметр прямо пропорционален частоте выборки. Это значит, что уменьшение частоты выборки, например, вдвое вызовет уменьшение частоты среза также в два раза, то есть частотой среза можно управлять с сохранением остальных параметров фильтра.

Один из способов управления — изменение тактовой частоты микроконтроллера. Для этого вместо кварцевого резонатора на контакт XTAL1 микроконтроллера подается сигнал от перестраиваемого импульсного генератора, однако при изменении его частоты будет изменяться и тактовая частота микроконтроллера, а значит, и время выполнения команд.

Еще один способ — изменение интервалов между прерываниями переполнения таймера Т0.

Проверка показала, что рассмотренная С-программа нормально работает при указанных параметрах схемы и фильтра (кроме частоты его среза, конечно), пока частота выборки не превысит 1024 Гц. В то же время программа, написанная на ассемблере, позволила увеличить частоту выборки примерно вдвое за счет сокращения времени на вычислительные операции.

Вывод. Программирование для микроконтроллера на С или на другом языке высокого уровня значительно сокращает исходный текст программ и время, затрачиваемое программистом, но в критических случаях, когда требуется обеспечить максимальное быстродействие и минимальный объем кода, лучше выполнять программирование на ассемблере.

Приложение 1. Как получить необходимые материалы через сеть Internet

Запустите на подключенном к сети Internet компьютере браузер, например, Internet Explorer или Netscape, введите ссылку на материал, который вы хотите загрузить (скачать) или прочитать.

Если вы ввели ссылку на файл, например, на astudio4.exe (файл самораспаковывающегося архива AVR Studio 4.08), то при его обнаружении на экране монитора появится окно с запросом на сохранение файла, выберите подходящую директорию или создайте новую и сохраните в ней файл.

Если была введена ссылка на страничку, например, со ссылками на техническую документацию, при ее обнаружении страничка будет выведена на экран. Слева от наименования каждого микроконтроллера находятся ссылки на его техническое описание в формате pdf, скачайте описание выбранного микроконтроллера, щелкнув мышкой по его ссылке.

Для работы с микроконтроллером надо скачать его полное описание (Complete), краткое описание (Summary) годится только для ознакомления с возможностями микроконтроллера.

Поскольку сайты регулярно обновляются, возможно изменение как ссылок, так и имен размещаемых файлов. Если при попытке открыть какую-то из ссылок появилось сообщение о том, что адрес не обнаружен, следует самостоятельно разыскать необходимые файлы.

Для этого можно воспользоваться поисковой системой сайта корпорации «Atmel» (*http://www.atmel.com*), в окошке поиска (Search) введите ключевое слово или фразу, например «avr studio» или «avr datasheets», в результатах поиска будут ссылки на все материалы сайта, содержащие ключевую фразу или слово.

Файл astudio4.exe велик. Если же обращение к сети идет медленно, связь с сервером, хранящим нужные вам файлы, может оборваться во время передачи. Если не удалось скачать файл с помощью браузера из-за обрывов связи, понадобится download manager — программа, позволяющая производить загрузку файлов в несколько приемов, причем загрузка может быть продолжена через несколько дней с того места закачиваемого файла, на котором произошел обрыв связи. Следует помнить, что существуют серверы, не поддерживающие такой режим.

К указанным программам относится FlashGet (ищите по адресу: *http://www.amazesoft.com/fgrushlp.zip*),

Net Vampire (можно скачать архив netvampire.zip размером 807K, находящийся по адресу: *http://www.netvampire.com/ftp/netvampire.zip*) и другие. Если скачать такой файл затруднительно, можно попробовать получить архив netvampire.zip, воспользовавшись службой почтовой рассылки emailfile

http://www.emailfile.com/?http://www.netvampire.com/ftp/netvampire.zip и получить netvampire.zip по электронной почте.

Если эти адреса не обнаруживаются, надо искать аналогичную программу, пользуясь поисковой машиной — сайтом, поддерживающим поиск в сети по ключевым словам.

Download manager лучше искать, пользуясь англоязычными поисковыми машинами. Вот несколько адресов таких машин:

http://www.altavista.com
http://www.google.com
http://www.yahoo.com

Выбрав поисковую машину, введите ее адрес в браузере и дождитесь загрузки основной странички. В строку поиска (Search) введите ключевые слова «download manager» и нажмите кнопку Search. Поисковая машина даст ссылки на зарегистрированные ею сайты, содержащие ваши ключевые слова, среди них будут и сайты, предлагающие программу Download Manager.

Не ограничивайтесь использованием одной поисковой машины: список сайтов, зарегистрированных на разных машинах, не одинаков. Это касается поиска программатора, его можно поискать на русскоязычных поисковых машинах.

Вот адреса некоторых из них:

http://www.aport.ru
http://www.google.ru
http://www.rambler.ru
http://www.yandex.ru

Просто введите в строку запроса открывшейся страницы фразу «программатор микроконтроллера avr» и выберите несколько найденных ссылок для ознакомления с характеристиками программаторов.

В каждой поисковой машине обеспечивается более сложный поиск, чем обнаружение ключевой фразы. Об этом вы сможете узнать, вызвав справочную систему конкретной поисковой машины.

Приложение 2. **Устройства, облегчающие отладку контроллера в составе системы**

При работе с контроллерами часто приходится подсоединять различные устройства. Делать это с использованием пайки не удобно, особенно в полевых условиях. На **Рис. 43** изображены различные переходники и индикаторы, позволяющие существенно облегчить жизнь при отладке работы контроллеров.

Рис. 43. **Переходники и светодиодные индикаторы**

Все переходники сделаны на основе контактов (штырь и гнездо) от разъемов типа 2РМ. Гнездо снабжено пружинящим обхватом, обеспечивающим надежный контакт и хорошую фиксацию штыря. Кроме того, гнездо хорошо фиксируется на контактах различных разъемов.

Слева на рисунке представлены штырь и гнездо, извлеченные из вилки и розетки типа 2РМ соответственно. Для извлечения контактов надо либо распилить металлическую оболочку разъема, либо выбить изолятор вместе с контактами.

В следующем вертикальном ряду рисунка представлен пружинный зажим с захваченной им иглой. Таким зажимом удобно подсоединяться к контактам микросхем, установленных на плату, к проводникам. Как зажим, так и игла заканчиваются гнездом.

Ниже представлены переходники «штырь — штырь» и «гнездо — гнездо». Далее — разветвитель «штырь — три гнезда».

Справа расположены два светодиодных индикатора. Маркировка R на одном из индикаторов означает, что внутри изолирующей трубки уже есть токоограничивающий резистор. Полярность подключения определяется по красной трубке (на рисунке она темная), надетой на проводник, который следует подключать к источнику напряжения +5 В. Второй индикатор не имеет резистора. Свет виден со стороны выходящего из трубки провода.

Используя подобный набор, можно легко подсоединить зажим к индикатору через переходник «штырь — штырь», а общий провод контроллера соединить с несколькими приборами через разветвитель «штырь — три гнезда». Для этого удобно использовать такие гнезда и штыри на концах кабелей приборов, а также на концах проводов для подключения к источникам питания.

Удобно иметь запас таких переходников и разветвителей «штырь — три гнезда», «гнездо — три штыря», а также кабелей и проводов с такими контактами.

Изолирующие трубки выполняют три функции:

- защищают проводник от обламывания около контакта;
- не должны сползать при разъединении контактов;
- должны надежно изолировать контакты при случайных касаниях токоведущих частей схемы.

Поэтому длина трубки должна быть примерно в полтора раза больше длины контакта. Диаметр трубки надо выбирать так, чтобы она плотно садилась на контакт, а натягивать ее следует со стороны припаянного провода, пока контакт не успел остыть. Гнездо должно быть утоплено в трубке, а при соединении штыря с гнездом изолирующие их трубки должны полностью скрыть контакты.

Часто встречающаяся в практике задача — это формирование на контактах контроллера выходных уровней в зависимости от уровней на его входных контактах. Обычно интерес представляет взаимное изменение уровней на большом количестве выводов контроллера. Для этого удобно воспользоваться устройством, схема которого изображена на **Рис. 44**.

В качестве индикаторов в схеме используются линейные шкалы АЛС317В с общим анодом. На **Рис. 45** представлено устройство, изготовленное по такой схеме.

Четыре корпуса линейных шкал АЛС317В, установленных впритык друг к другу, образуют единую линейную шкалу на 20 разрядов (темная вертикальная полоса посередине платы). От линейной шкалы влево и вправо идут дорожки к резисторам R1...R20, сами резисторы расположены с другой стороны платы.

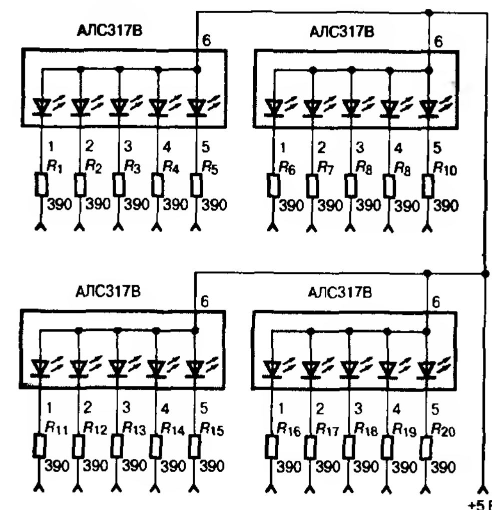


Рис. 44. Схема электрическая индикатора «Линейная шкала»

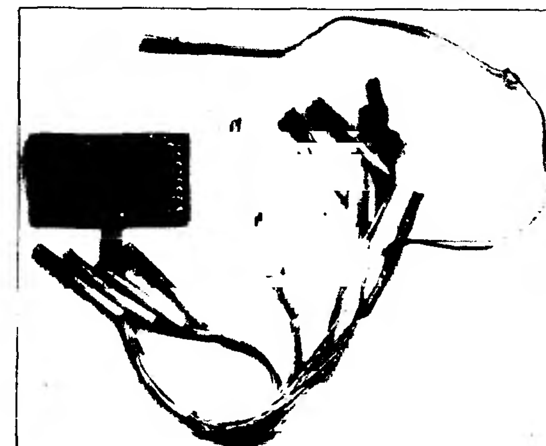


Рис. 45. Индикатор «Линейная шкала»

Имитация свечения шести горизонтальных сегментов линейной шкалы сделана искусственно — ни один из проводников устройства не подключен к источнику сигнала.

Все проводники разделены на группы по 4 проводника. Каждая группа перехвачена трубкой. Длина трубок для каждой группы различна, длина каждого из четырех проводников также различна. Это позволяет легко определить последовательность подключения проводников к световым сегментам линейной шкалы. Например, самый короткий проводник в группе, перехваченной самой короткой трубкой, подсоединен к самому младшему сегменту линейной шкалы.

На трубках, изолирующих контакты (ими заканчивается каждый проводник), можно маркером записать номер контакта разъема контроллера, к которому должен подключаться данный проводник в конкретной разработке. Такую надпись можно стереть и заменить другой при переходе к другой разработке.

На плате у проводника «+5В» необходимо нанести соответствующую маркировку, чтобы через некоторое время не возник вопрос, подключать проводник к общему проводу или к питанию контролируемого устройства.

Еще одна задача при отладке — подсчет числа импульсов, поступающих на контроллер или генерируемых им, а также определение состояния контакта. Для этого подойдет щуп, представленный на **Рис. 46**.

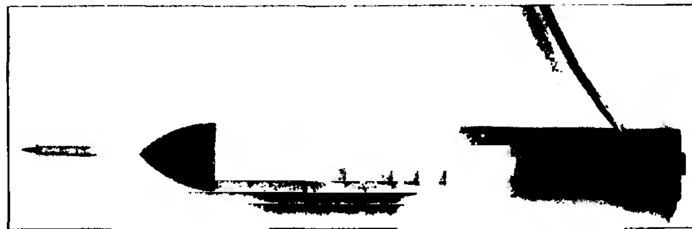


Рис. 46. Щуп счетчика числа импульсов

Первый индикатор отражает состояние контролируемой цепи (показание «1» соответствует ВЫСОКОМУ уровню, «0» — НИЗКОМУ, «С» — высокоимпедансному состоянию или отсутствию контакта). Три последующих индикатора позволяют подсчитать без переполнения до 999 импульсов. В торце щупа расположена кнопка сброса счетчика импульсов. На схеме (**Рис. 47**) представлен каскад определения состояния линии контролируемой цепи. Под ним расположен первый каскад счетчика с индикатором (обведен штриховой линией). Два других каскада ничем не отличаются от первого.

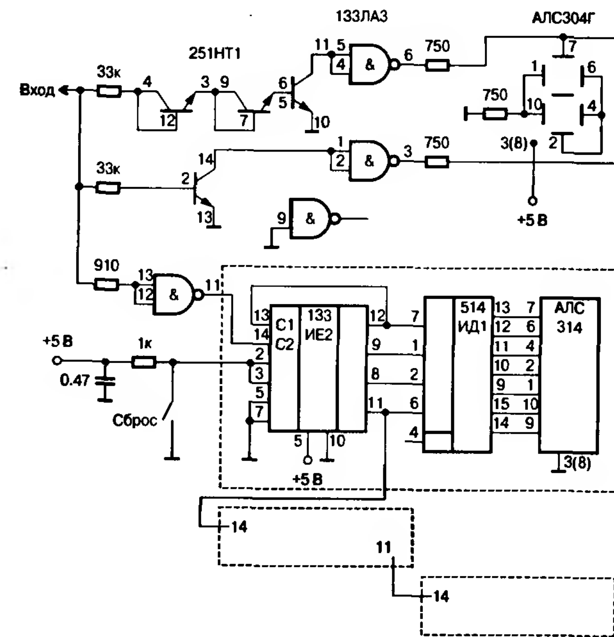


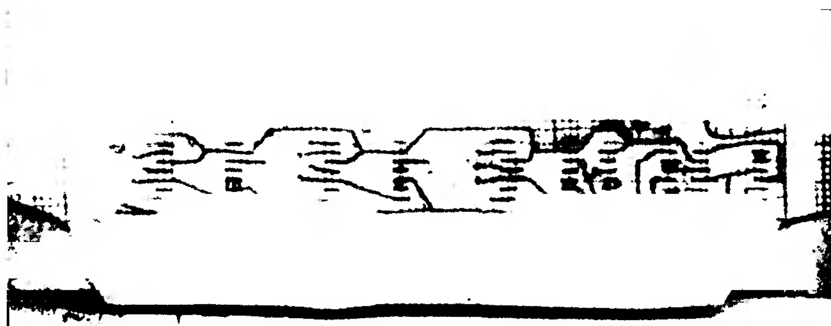
Рис. 47. Схема щупа

Ниже (**Рис. 48**) представлена печатная плата со стороны установки индикаторов (а) и со стороны установки микросхем счетчиков (б). Слева установлена кнопка сброса (микропереключатель).

Вся конструкция помещалась в самодельный корпус из намотанной на трубку соответствующего диаметра стеклоткани, которая пропитывалась эпоксидной смолой. Углубление в пластилине с вставленной в него иглой заливалось эпоксидной смолой. Полученная заготовка обтачивалась так, чтобы она плотно входила в корпус. Колпачок кнопки слегка утоплен в корпусе. Поэтому щуп можно удерживать, прижимая корпус сверху одним пальцем к контролируемому контакту, и нажать кнопку сброса счетчика.

На **Рис. 49** показан пример использования переходников и линейной шкалы: щуп осциллографа подключен к контакту микроконтроллера с помощью пружинного зажима, непосредственно к разъему платы подключены восемь световых сегментов линейной шкалы. Общие провода источника питания и осциллографа подключены к разъему через разветвитель.

а)



б)

Рис. 48. Расположение на печатной плате индикаторов АЛС304, АЛС314 (*а*) и счетчиков 133ИЕ2 (*б*)

Рис. 49. Щуп осциллографа и индикатор «Линейная шкала»

Приложение 3. **Программатор**

В качестве основы для программатора выбрана схема, представленная на **Рис. 50**.

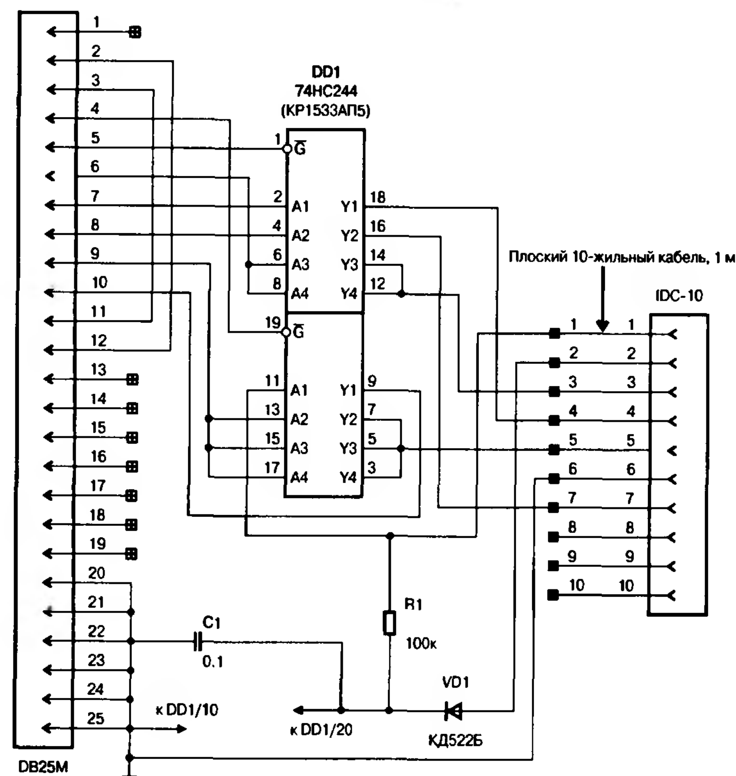


Рис. 50. Схема программатора для микроконтроллеров AVR

О конструкции программатора

Рис. 51. Программатор для микроконтроллеров AVR

Плата программатора размещается в корпусе разъема DB-25M. Существует две модификации корпусов этого разъема. Удобнее использовать пластмассовый корпус на защелках: в нем легко размещается печатная плата с микросхемой DD1. Для программирования микроконтроллеров AVR длина кабеля может достигать одного метра.

На **Рис. 51** показан программатор (крышка снята).

Печатная плата программатора

На **Рис. 52** показаны две стороны платы программатора.

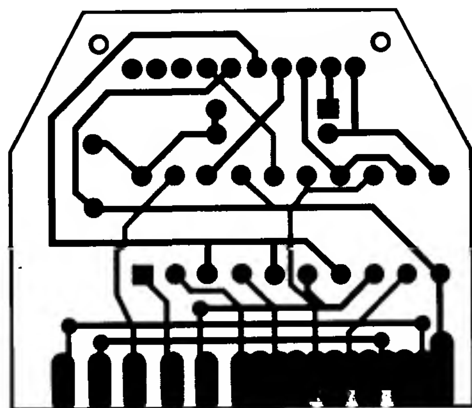


Рис. 52. Печатная плата программатора

Плата вдвигается между жвумя рядами контактов разъема DB-25M, затем дорожки платы припаиваются к накрывающим их контактам разъема. Со стороны монтажа некоторые контакты разъема могут достать до ближайшей горизонтальной дорожки, в таком случае эти контакты надо предварительно укоротить.

Использование контактов LPT-порта в программаторе для микроконтроллеров AVR

В **Табл. 7** приводится описание контактов LPT-порта компьютера, которые используются в программаторе.

При использовании порта LPT1 программное управление производится через порты 378h, 379h и 37Ah; для порта LPT2 — через 278h, 279h и 27Ah.

Таблица 7

Контакт	Наименованне контакта	Порт	Номер бита	Направление	Инверсия
2	DATA 0	378h (278h)	0	выход	нет
3	DATA 1	378h (278h)	1	выход	нет
4	DATA 2	378h (278h)	2	выход	нет
5	DATA 3	378h (278h)	3	выход	нет
6	DATA 4	378h (278h)	4	выход	нет
7	DATA 5	378h (278h)	5	выход	нет
8	DATA 6	378h (278h)	6	выход	нет
9	DATA 7	378h (278h)	7	выход	нет
10	ACK	379h (279h)	6	вход	нет
11	BUSY	379h (279h)	7	вход	есть
12	PE	379h (279h)	5	вход	нет
18...25	0V	—	—	—	—

Направление ВЫХОД означает, что информация через контакт выводится из компьютера. ВЫХОД (ВХОД) — возможны оба направления передачи через контакт, но в программаторе контакт работает как ВЫХОД. Наличие ИНВЕРСИИ означает, что единице в разряде порта будет соответствовать НИЗКИЙ уровень, нулю — ВЫСОКИЙ уровень на контакте.

Как управлять состоянием указанных в таблице линий LPT-порта

Простая программа на языке программирования Pascal иллюстрирует организацию управления и позволяет проверить работу LPT-порта компьютера.

Для проверки состояния линий порта подойдет тестер в режиме измерения постоянного напряжения, уровни напряжений должны измеряться на указанных контактах разъема LPT-порта относительно любого из контактов 0 В. Вместо тестера удобно использовать светодиод с резистором (см. Приложение 2).

> Для подключения тестера, светодиода с резистором или перемычек к разъему LPT-порта используйте только изолированные контакты соответствующего диаметра, аналогичные описанным в Приложении 2. Это защитит порт от выхода из строя.

```
const bit0 = 1; bit1 = 2; bit2 = 4; bit3 = 8; bit4 = 16; bit5 = 32;
                                        bit6 = 64; bit7 = 128;
var   B: byte;
begin
Port[$378]:= bit7 + bit5;               {в порт 378h посылается 128 + 32, высокие
                                        уровни на контактах 9 и 7}
Readln;                                 {проверьте уровни на контактах 2, 3, 7, 8
                                        и 9, затем нажмите ENTER}
Port[$37A]:= bit1 + bit2;               {на контакте 14 низкий уровень
                                        (с инверсией), на контакте 16 высокий
                                        уровень}
Readln;                                 {проверьте уровни на контактах 14 и 16,
                                        затем нажмите ENTER}
while true do begin                     {в бесконечном цикле проверяется
                                        состояние порта 379h}
B:= Port[$379];                         {на контактах, не соединенных с землей,
                                        высокие уровни}
Writeln(B);                             {исходное состояние порта 127:
                                        1 + 2 + 4 + 8 + 16 + 32 + 64 (нет 128,
                                        так как разряд 7 с инверсией)}
end;                                    {соединение контактов 11, 12, 13 или 15 с
                                        любым контактом 0V изменит состояние
                                        порта, что отразится на экране}
end.
```

Для удобства работы с программой перед операторами *Readln* можно добавить операторы

```
Write                                   ('В порт … выведено значение …, после
                                        проверки состояния порта нажать Enter');
```

*Формат файлов *.hex, загружаемых в память микроконтроллера AVR*

В результате ассемблирования в AVRStudio можно получить файл в формате *.hex. Именно он содержит информацию, предназначенную для загрузки в память программ микроконтроллера. Такой файл можно просмотреть с помощью Блокнота, имеющегося в операционной системе Windows.

В этом же формате можно хранить данные для EEPROM микроконтроллера (формат файла определялся интуитивно, поэтому возможны поправки).

Рассмотрим фрагмент файла *.hex, генерированного AVR Studio для записи в память программ микроконтроллера AVR.

Для наглядности в строки введены интервалы. Каждая строка начинается двоеточием. Каждая последующая пара символов представляет собой один байт в шестнадцатеричном коде. Четыре байта в начале и один байт в конце строки являются служебными. Между ними находятся информационные байты, которые загружаются в память.

В начале и в конце файла находятся служебные строки, в которых информационные байты отсутствуют.

Ниже приведены две начальные и три конечные строки одной из программ:

:10	0000	00	B6C0	0000	0000	0000	0000	0000	0000	0000	7A
:10	0010	00	0000	0000	0000	0000	0000	24D0	7CE0	50FD	43
						...					
:10	0E20	00	2A1D	3B1D	4C1D	5D1D	6E1D	7F1D	2797	7993	4F
:0E	0E30	00	6993	5993	4993	3993	2993	1993	0895	1F	
:00	0000	01	FF								

Первый байт — число байтов данных в строке; для первой строки это 10, значит, в первой, второй и третьей строках по 10Н байт (10Н = 16); в предпоследней строке их 14 (0ЕН = 14).

Следующие два байта — адрес ячейки памяти, по которому записывается первый байт данных строки (остальные байты данных строки записываются в последующие ячейки памяти), третий байт отличается от нуля только в служебных строках. Он равен единице в последней строке, являющейся, очевидно, признаком конца файла, и равен двум в первой строке файла *.hex, загружаемого в EEPROM микроконтроллера AVR (такая первая строка — признак файла с информацией для EEPROM).

Далее — пары байтов данных, каждая пара составляет ассемблерную команду или половину команды микроконтроллера AVR.

Последний байт строки — контрольная сумма. Если просуммировать все байты строки вместе с контрольной суммой, то младший байт результата должен равняться нулю.

Алгоритм последовательной загрузки программы в микроконтроллер AVR

Для начала следует ознакомиться с разделом Serial Downloading главы Memory Programming полного технического описания какого-нибудь микроконтроллера AVR. Ниже остановимся лишь на моментах, которые не были описаны вовсе или вызывали сомнение и потребовали проверки.

Если сравнить пары байтов в файле *.hex с парами байтов при просмотре Program Memory в AVR Studio, то оказывается, что в файле *.hex пара выгля-

дит так: C04F, а в Program Memory так: 4FC0, то есть в Program Memory байты переставлены местами. Последовательность байтов в файле *.hex соответствует последовательности передачи байтов в микроконтроллер.

При подаче питания на микроконтроллер следует выполнять инструкцию Programming Enable (см. Serial Programming Instruction Set). Передача этой инструкция обязательна не только перед записью в память, но и перед считыванием, а также перед стиранием данных.

При программировании микроконтроллера через его SPI-интерфейс данные передаются в оба направления, но последовательность передачи через порт и приема ответа от микроконтроллера важна и она должна быть следующей:

1. Записать информацию на выходную линию LPT-порта, считать информацию с входной линии порта при SCK = 0 (то есть информация, выставленная компьютером, записывается в микроконтроллер AVR, а информация, переданная в это же время микроконтроллером, принимается компьютером).
2. Установить SCK (SCK = 1).
3. Сделать паузу.
4. Сбросить SCK (SCK = 0, при этом информация с выходной линии микроконтроллера AVR поступает через программатор в компьютер).

Пример программного обеспечения программатора

Предлагаемая программа приводится лишь как пособие для разработчика программатора. Программа тестировалась лишь на трех разных по быстродействию компьютерах и не предназначалась для распространения, поэтому ответственности за неполадки, связанные с ее использованием, автор не несет.

Программа написана давно, некоторые операторы, функции и процедуры можно значительно упростить, а ее интерфейс оставляет желать лучшего, тем не менее она довольно долго выполняла все необходимые задачи.

Программа нормально работает с файлами, полученными в результате ассемблирования в AVR Studio, а так же с файлами, генерируемыми IAR Embedded Workbench.

Из-за непосредственного обращения к портам ввода/вывода программа может работать в операционных системах Windows 95/98, в более поздних версиях такое обращение к портам блокируется.

При необходимости вы сможете внести изменения в программу самостоятельно.

Наименование проекта — PrgAVR. Имя файла проекта — PrgAVR.dpr. Имя файла модуля, листинг которого приведен ниже, — UPrgAvr.pas. Имя файла формы проекта – PrgAVR.dfm.

Форма программатора

Размещение компонентов на форме приводится на **Рис. 53**. Вид формы показан во время выполнения программы, поэтому компоненты LC1, LC2, BtChk, OpenDialog1 не видны, так как LC1 и LC2 содержат пустую строку в свойстве Caption, свойство BtChk.Visible = false, а компонент OpenDialog1 визуально отображается только при открытии файла.

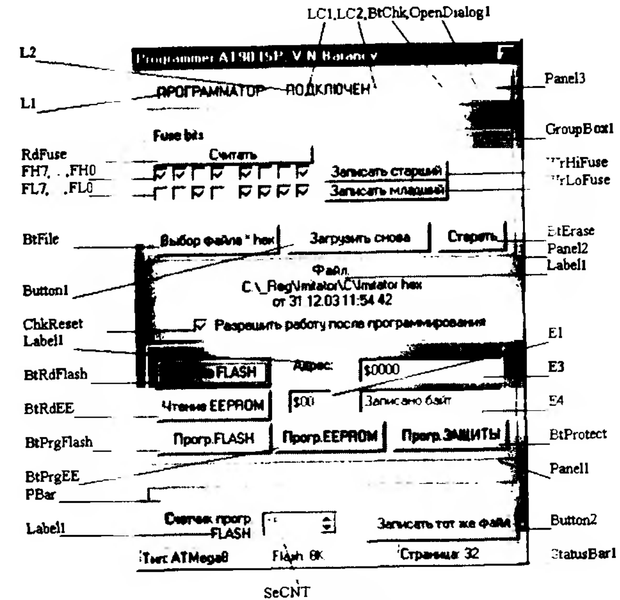


Рис. 53. Размещение компонентов на форме программатора

Типы используемых компонентов:

- BtChk, BtErase, BtFile, BtPrgEE, BtPrgFlash, BtProtect, BtRdEE, BtRdFlash, Button1, Button2, ChkReset, RdFuse, WrHiFuse, WrLoFuse: TButton,
- E1, E3, E4: TEdit,
- FL7...FL0, FH7..FH0: TcheckBox,
- GroupBox1: TgroupBox,
- L1, L2, LC1, LC2, Label1, Label2, Label3: TLabel,
- OpenDialog1: TOpenDialog,
- Panel1...Panel3: TPanel,
- PBar: TProgressBar,
- SeCNT: TspinEdit,
- StatusBar1: TStatusBar.

Об использовании Fuse-байтов

На форме программатора под кнопкой «Считать» расположено 8 окошек FH7...FH0, соответствующих разрядам 7...0 старшего Fuse-байта, а под ними — окошки FL7...FL0, соответствующие разрядам младшего Fuse-байта.

Если при программировании микроконтроллеров серии AT90 Fuse-байтам можно было не уделять внимания, то при программировании микроконтроллеров серии ATmega без их использования не обойтись, так как в продажу эти микроконтроллеры поступают с установкой Fuse-байтов, соответствующей работе от внутреннего *RC*-генератора с тактовой частотой около 1 МГц, для работы от внешнего кварцевого генератора необходимо изменить состояние соответствующих разрядов Fuse-байтов.

Для микроконтроллеров серии ATmega состояние Fuse-байтов определяет, будет ли микроконтроллер работать от внутреннего *RC*-генератора или от внешнего кварцевого либо керамического резонатора, либо просто от внешней RC-цепочки, возможна ли загрузка памяти программ во время работы микроконтроллера (использование функции BootLoader), возможность использования контакта RESET в качестве дополнительной линии ввода/вывода, а для микроконтроллеров ATmega8515 и ATmega8535 – возможно ли использование программ, написанных для AT90S8515 и AT90S8535.

Количество Fuse-байтов, назначение разрядов различны для разных микроконтроллеров серии ATmega, ищите их в техническом описании на конкретный микроконтроллер. На форме представлено состояние Fuse-байтов, обеспечивающее работу микроконтроллера ATmega8 от кварцевого резонатора частотой 3...8 МГц.

> Будьте особенно внимательны при изменении состояния Fuse-байтов, так, для микроконтроллера ATmega8 установка разряда SPIEN (окошко FH5 на Форме, Рис.53) приведет к невозможности дальнейшего последовательного программирования через SPI-интерфейс, а сброс разряда RSTDISBL (FH7 на Рис.53) позволит использовать линию Reset как дополнительную линию PC6 порта C, что также сделает невозможным дальнейшее программирование через SPI-интерфейс.
> Восстановить микроконтроллер в этих случаях можно только с помощью параллельного программатора, более дорогого и сложного.

Листинг программы

Кроме обычного определения процедур нажатия кнопок, для компонента E1 необходимо определить процедуру для события OnKeyPress. Создание обработчиков в Delphi рассматривалось в разделе программирования COM-порта.

```
//Prg_Mega_Isp.exe [DlPrm] [DelMs]
//DlPrm=500 for my PC and =5 for old PCs
unit UPrg_Mega_Isp_Str;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ExtCtrls, ComCtrls, IniFiles, Spin;

type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Panel2: TPanel;
    Panel3: TPanel;
    LC1: TLabel;
    LC2: TLabel;
    L1: TLabel;
    L2: TLabel;
    OpenDialog1: TOpenDialog;
    Label1: TLabel;
    PBar: TProgressBar;
    BtPrgEE: TButton;
    Label3: TLabel;
    BtProtect: TButton;
    StatusBar1: TStatusBar;
    Button1: TButton;
    ChkReset: TCheckBox;
    BtChk: TButton;
    BtRdFlash: TButton;
    BtPrgFlash: TButton;
    BtRdEE: TButton;
    E1: TEdit;
    E3: TEdit;
    BtFile: TButton;
    BtErase: TButton;
    E4: TEdit;
    Button2: TButton;
    GroupBox1: TGroupBox;
    FH7: TCheckBox;
    FH6: TCheckBox;
```

```
    FH5: TCheckBox;
    FH4: TCheckBox;
    FH3: TCheckBox;
    FH2: TCheckBox;
    FH1: TCheckBox;
    FH0: TCheckBox;
    FL7: TCheckBox;
    FL6: TCheckBox;
    FL5: TCheckBox;
    FL4: TCheckBox;
    FL3: TCheckBox;
    FL2: TCheckBox;
    FL1: TCheckBox;
    FL0: TCheckBox;
    RdFuse: TButton;
    WrLoFuse: TButton;
    WrHiFuse: TButton;
    Label2: TLabel;
    SECnt: TSpinEdit;
    procedure BtFileClick(Sender: TObject);
    procedure BtChkClick(Sender: TObject);
    procedure BtRdFlashClick(Sender: TObject);
    procedure BtPrgFlashClick(Sender: TObject);
    procedure BtEraseClick(Sender: TObject);
    procedure ElKeyPress(Sender: TObject; var Key: Char);
    procedure BtPrgEEClick(Sender: TObject);
    procedure BtRdEEClick(Sender: TObject);
    procedure BtProtectClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure RdFuseClick(Sender: TObject);
    procedure WrLoFuseClick(Sender: TObject);
    procedure WrHiFuseClick(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure Label2DblClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation
{$R *.DFM}
const
     PW      =$378;
```



```
     RST     =$80;
     MOSI    =$20;
     SCK     =$10;
     ChkACK  =$01;
     Pause   =0;

     PR      =PW+1;
     MISO    =$40;
     ACK     =$20;

var  DBt, PageSize                      :byte;
     sFile, DTFile                      :string;
     PosStr,Size,ArrSize,AddrOrg        :longint;
     bIn, GetBt                         :byte;
     GetWd, DlPrm, DlyMs                :word;
     FileOK, Reload, PrgReady           :boolean;
     Buf,BWr                            :pByteArray;
     IniFName                           :string;
     IniFile                            :TIniFile;

procedure DOut(Port,Num:word);
begin
Asm
mov dx,Port
mov ax,Num
out dx,ax
end;
end;//DOut

procedure DIn(var Num:byte; Port:word);
var B:byte;
begin
Asm
mov dx,Port
in al,dx
mov B,al
end;
Num:=B;
end;//DIn

procedure Delay(Ticks:dword); {Задержка в милисекундах}
var dw : dword;
begin
dw:=GetTickCount+Ticks;
while GetTickCount<dw do;
end;

procedure Dly;
var J:integer;
```

```
begin
for J:=1 to DlPrm do;
end;

procedure RstAVR;
begin
DBt:=Rst; DOut(PW,DBt); Delay(30);
DBt:=0; DOut(PW,DBt);
end;

procedure SetAVR;
begin
DBt:=$c; DOut(PW,DBt);
end;

function RdBit:byte;
begin
DIn(Result,PR);
Inc(DBt,SCK); DOut(PW,DBt);
Result:=Result and MISO div MISO;
Dly;
Dec(DBt,SCK);
DOut(PW,DBt); Dly;
end;

procedure WrBit(WrBt:byte);
begin DBt:=WrBt; DOut(PW,DBt); bIn:=RdBit; end;

function SendBt(BSnd:byte):byte;
var I:integer;
begin
Result:=0;
for I:=0 to 7 do begin
  WrBit((BSnd and $80 shr 7)*MOSI);
  BSnd:=BSnd shl 1;
  Result:=(Result shl 1)+bIn;
end;
Dly;Dly;
end;

function ProgEnable:boolean;
var IPE:integer;
    BPE:byte;
begin
Delay(30);  //Delay more than 20 ms
IPE:=0; GetBt:=0;
while (GetBt<>$53) and (IPE<32) do begin
  SendBt($ac);
  SendBt($53);
```



```
  BPE:=SendBt($ff);
  GetBt:=BPE;
  BPE:=SendBt($ff);
  GetWd:=BPE;
  Delay(10);
  Inc(IPE);
end;
if GetBt<>$53 then Result:=False else Result:=True;
end;

function ReadSignature:word;
var BLow,BHigh:byte;
begin
  SendBt($30);
  SendBt($00);
  SendBt($01);
  BLow:=SendBt($ff);
  SendBt($30);
  SendBt($00);
  SendBt($02);
  BHigh:=SendBt($ff);
  Result:=(BHigh shl 8)+ BLow;
end;

procedure ChipErase;
begin
SendBt($ac);
SendBt($80);
SendBt($ff);
SendBt($ff);
end;

function ProgRead(Addr:word):word;
begin
SendBt($28);     //High byte reading
SendBt(Hi(Addr));
SendBt(Lo(Addr));
Result:=SendBt($00) shl 8;
//Delay(DlyMs);
SendBt($20);     //Low byte reading
SendBt(Hi(Addr));
SendBt(Lo(Addr));
Result:=Result+SendBt($00);
end;

function ProgWrite(AWr:word;Wd:word):boolean;
var I,wR:word;
begin
wR:=not(Wd); I:=0;//ProgRead(AWr);
```

```
while (Hi(wR)<>Hi(Wd)) and (I<10) do begin
  SendBt($48);       //High byte write
  SendBt(Hi(AWr));
  SendBt(Lo(AWr));
  SendBt(Hi(Wd));
  Delay(DlyMs);
  wR:=ProgRead(AWr);
  Inc(I);
end;
if Hi(Wd)=$7f then Delay(DlyMs);
if I<10 then begin
  wR:=not(Wd); I:=0;//ProgRead(AWr);
  while (Lo(wR)<>Lo(Wd)) and (I<10) do begin
    SendBt($40);      //Low byte write
    SendBt(Hi(AWr));
    SendBt(Lo(AWr));
    SendBt(Lo(Wd));
    Delay(DlyMs);
    wR:=ProgRead(AWr); I:=0;
    Inc(I);
  end;
  if Lo(Wd)=$7f then Delay(DlyMs);
end;
if I<10 then Result:=True else Result:=False
end;

procedure LdWdToPage(AWr:byte;Wd:word);
begin
  SendBt($40);       //Low byte write
  SendBt($00);
  SendBt(AWr);
  SendBt(Lo(Wd));

  SendBt($48);      //High byte write
  SendBt($00);
  SendBt(AWr);
  SendBt(Hi(Wd));
end;

procedure LoadPage(PageAddr:word);
var L:longword;
begin
  SendBt($4c);      //High byte write
  SendBt(Hi(PageAddr));
  SendBt(Lo(PageAddr));
  SendBt($FF);
  L:=GetTickCount+10;
  while GetTickCount<L do;
end;
```



```
function EERead(Addr:word):byte;
begin
SendBt($a0);
SendBt(Hi(Addr));
SendBt(Lo(Addr));
Result:=SendBt($00);
end;

procedure EEWrite(Addr:word; Bt:byte);
begin
if EERead(Addr)<>Bt then begin
  SendBt($c0);
  SendBt(Hi(Addr));
  SendBt(Lo(Addr));
  SendBt(Bt);
  while EERead(Addr)<>Bt do;
  if (Bt=$7f) or (Bt=$80) then Delay(DlyMs);
end;
end;

function SignRead(Addr:word):byte;
begin
SendBt($30);
SendBt($00);
SendBt(Lo(Addr));
Result:=SendBt($00);
end;

procedure LockWrite(Bt:byte);
begin
SendBt($ac);
SendBt((Bt shl 1) or $f9);
SendBt($ff);
SendBt(Bt or $fc);
end;

function AdapterOK:boolean;//Проверка закоротки конт.7 и 10
var      Bt:byte;
begin
Result:=True;
DlyMs:=4;
DlPrm:=500; //DlPrm:=5 for old PCs
if ParamCount>0 then begin
  DlPrm:=StrToInt(ParamStr(1));
  if ParamCount=2 then
    DlyMs:=StrToInt(ParamStr(2));
end;
DBt:=ChkACK; DOut(PW,DBt); Delay(30);
```

```
DIn(Bt,PR);
if Bt and ACK=0 then Result:=False;
DOut(PW,0); Delay(30);
DIn(Bt,PR);
if Bt and ACK=ACK then Result:=False;
end;

function VoltageOK:boolean;
begin
Result:=True;
{DOut(PW,Enbl); Delay(30);
DIn(Bt,PR);
if Bt and Busy=0 then Result:=False;}
end;

procedure GetProgArr; //Result: BWr^[0..ArrSize-1]
var S,sAdr,sNum,sTyp,sB  :string;
    J,Adr,Num,Typ,K,MaxAddr      :integer;
begin
  MaxAddr:=0;
  PosStr:=0;
  while PosStr<Size do begin
    S:='';
    while (Char(Buf^[PosStr])<>#13) do begin
      S:=S+Char(Buf^[PosStr]);
      Inc(PosStr);
    end;
    Inc(PosStr,2);    //#13 и ":" пропустить для перехода к след строке
    sNum:='$'+Copy(S,2,2); Val(sNum,Num,K);  //число байт в строке
    sAdr:='$'+Copy(S,4,4); Val(sAdr,Adr,K);  //нач.адрес строки во FLASH
    if Adr+Num>MaxAddr then MaxAddr:=Adr+Num;

    sTyp:='$'+Copy(S,8,2); Val(sTyp,Typ,K);  //sTyp = 00 для данных
    Delete(S,1,9);  //стереть в строке слева все до данных
    J:=1;
    while (Typ=0) and (J<2*Num) do begin
      sB:='$'+Copy(S,J,2);
      Val(sB,BWr^[Adr],K);
      Inc(Adr);
      Inc(J,2);
    end;
  end;
  ArrSize:=MaxAddr;
end;  //GetProgArr; result in BWr^[0..ArrSize-1]

procedure TForm1.BtFileClick(Sender: TObject);
var
    F     :file;
    FStm  :TStream;
```



```
    I     :longword;
    FHandle:integer;
begin
BtPrgFlash.Caption:='Прорг.FLASH';
BtPrgFlash.Font.Color:=clBlack;
if not Reload then if OpenDialog1.Execute then
                                    sFile:=OpenDialog1.FileName;
if FileExists(sFile) then begin

  FHandle:=FileOpen(SFile,fmOpenRead);
  DTFile:=' от '+DateToStr(FileDateToDateTime(FileGetDate(FHandle)));
  DTFile:=DTFile+' '+TimeToStr(FileDateToDateTime(FileGetDate(FHandle)));
  FileClose(FHandle);

  FileOK:=True;
  Label1.Caption:='Файл: '+sFile+DTFile;
  Label1.Font.Color:=clBlack;
  Label1.Left:=(Form1.Width-Label1.Width) div 2;
  AssignFile(F,sFile); Reset(F,1);
  Size:=FileSize(F);
  CloseFile(F);
  FStm:=TFileStream.Create(sFile,fmOpenReadWrite);
  ArrSize:=Size;
  GetMem(Buf,Size);
  GetMem(BWr,Size);
  FStm.Read(Buf^,Size);
  FStm.Free;
  for I:=0 to Size-1 do BWr^[I]:=$ff;
  GetProgArr;
  FreeMem(Buf,Size);
end;
Reload:=false;
end;

procedure TForm1.BtChkClick(Sender: TObject);
  var sType, sFlash:string;
begin
BtPrgFlash.Caption:='Прорг.FLASH';
BtPrgFlash.Font.Color:=clBlack;
if AdapterOK then begin
  LC1.Caption:='ПОДКЛЮЧЕН';
  LC1.Font.Color:=clBlack;
    if VoltageOK then begin
      LC2.Caption:='';
      LC2.Font.Color:=clBlack;
//
      RstAVR;
      if ProgEnable then PrgReady:=true
                    else PrgReady:=false;
```

```
      if not PrgReady then begin
        LC1.Caption:='Программатор не готов';
        LC1.Font.Color:=clRED;
      end;
      case ReadSignature of
       $0693: begin PageSize:=32; sType:='ATMega8515'; sFlash:='8K'; end;
       $0793: begin PageSize:=32; sType:='ATMega8'; sFlash:='8K'; end;
       $0893: begin PageSize:=32; sType:='ATMega8535'; sFlash:='8K'; end;
       $0394: begin PageSize:=64; sType:='ATMega16'; sFlash:='16K'; end;
       $0197: begin PageSize:=128; sType:='ATMega103'; sFlash:='128K'; end;
       $0297: begin PageSize:=128; sType:='ATMega128'; sFlash:='128K'; end;

       $0590: begin PageSize:=0; sType:='ATtiny12'; sFlash:='1K'; end;
       $0690: begin PageSize:=0; sType:='ATtiny15'; sFlash:='1K'; end;
       $0991: begin PageSize:=0; sType:='ATtiny26'; sFlash:='2K'; end;

       $0191: begin PageSize:=0; sType:='AT90S2313'; sFlash:='2K'; end;
       $0192: begin PageSize:=0; sType:='AT90S4414'; sFlash:='4K'; end;
       $0392: begin PageSize:=0; sType:='AT90S4433'; sFlash:='4K'; end;
       $0193: begin PageSize:=0; sType:='AT90S8515'; sFlash:='8K'; end;
       $0393: begin PageSize:=0; sType:='AT90S8535'; sFlash:='8K'; end
       else   begin PageSize:=0; sType:=''; sFlash:=''; end;
      end;
      StatusBar1.Panels[0].Text:='Тип: '+sType;
      StatusBar1.Panels[1].Text:='Flash: '+sFlash;
      StatusBar1.Panels[2].Text:='Страница: '+IntToStr(PageSize);
    end        else begin
      LC2.Caption:='ОТСУТСТВУЕТ';
      LC2.Font.Color:=clRed;
    end;
end          else begin
  LC1.Caption:='НЕ ПОДКЛЮЧЕН';
  LC1.Font.Color:=clRed;
  LC2.Caption:='';
end;
end;

procedure TForm1.BtRdFlashClick(Sender: TObject);
var A:word;
    S:string;
begin
BtChk.OnClick(self);
if not PrgReady then Exit;
S:=E1.Text; A:=StrToInt(S);
A:=ProgRead(A);
E3.Text:=Format('$%.2x',[Hi(A)])+Format('%.2x',[Lo(A)]);
if ChkReset.Checked then SetAVR;
end;
```



```
procedure TForm1.BtPrgFlashClick(Sender: TObject);
var       PageNum,I,A,W:word;
          WdOnPage:byte;
begin
BtChk.OnClick(self);
if not PrgReady then Exit;
PBar.Position:=0;
A:=0;
if (AdapterOK) and (VoltageOK) and (FileOK) then begin
  I:=0; GetWd:=0;
  if PageSize=0 then begin
    while I<=ArrSize-1 do begin
      W:=BWr^[I+1];
      W:=(W shl 8)+BWr^[I];
      if GetWd mod (ArrSize div 20)=0      if ProgWrite(A,W) then
                                      Inc(GetWd);
        then PBar.Position:=PBar.Position+10;
      Inc(A);
      Inc(I,2);
    end;
  end else begin
    PageNum:=0;
    while I<=ArrSize do begin
      WdOnPage:=0;
      while WdOnPage<PageSize do begin
        if I<ArrSize then begin W:=BWr^[I+1]; W:=(W shl 8)+BWr^[I]; end
                    else W:=$ffff;
        LdWdToPage(WdOnPage,W);
        Inc(WdOnPage);
        Inc(GetWd);
        if GetWd mod (ArrSize div 20)=0
          then PBar.Position:=PBar.Position+10;
        Inc(I,2);
      end;
      LoadPage(PageNum*PageSize);
      Inc(PageNum);
    end;
  end;
  E3.Text:=IntToStr(ArrSize);
  E4.Text:=IntToStr(GetWd*2);
  SECnt.Value:=SECnt.Value+1;
  if ChkReset.Checked then SetAVR;
end else begin
  BtPrgFlash.Caption:='АДАПТЕР/ФАЙЛ!';
  BtPrgFlash.Font.Color:=clRed;
end;
end;

procedure TForm1.BtEraseClick(Sender: TObject);
```

```
begin
BtChk.OnClick(self);
if not PrgReady then Exit;
ChipErase;
end;

procedure TForm1.E1KeyPress(Sender: TObject; var Key: Char);
begin
if Key=#13 then BtRdFlashClick(Sender);
end;

procedure TForm1.BtPrgEEClick(Sender: TObject);
var       I,A,W:word;
begin
BtChk.OnClick(self);
if not PrgReady then Exit;
PBar.Position:=0;
if (AdapterOK) and (VoltageOK) and (FileOK) then begin
  A:=AddrOrg;
  I:=0; GetWd:=0;
  while I<=ArrSize do begin
    W:=BWr^[I];
    EEWrite(A,W);
    Inc(GetWd);
    if GetWd mod (ArrSize div 20)=0
      then PBar.Position:=PBar.Position+10;
    Inc(A);
    Inc(I);
  end;
  E3.Text:=IntToStr(ArrSize);
  E4.Text:=IntToStr(GetWd);
  if ChkReset.Checked then SetAVR;
end                                             else begin
  BtPrgFlash.Caption:='АДАПТЕР/ФАЙЛ!';
  BtPrgFlash.Font.Color:=clRed;
end;
end;

procedure TForm1.BtRdEEClick(Sender: TObject);
var A:word;
    S:string;
begin
BtChk.OnClick(self);
if not PrgReady then Exit;
S:=E1.Text; A:=StrToInt(S);
A:=EERead(A);
E3.Text:=Format('$%.2x',[Lo(A)]);
if ChkReset.Checked then SetAVR;
end;
```



```
procedure TForm1.BtProtectClick(Sender: TObject);
var Bt:byte;
begin
BtChk.OnClick(self);
if not PrgReady then Exit;
Bt:=0;
LockWrite(Bt);
if ChkReset.Checked then SetAVR;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
IniFName:=GetCurrentDir+'\Avr_Prg.ini';
IniFile := TIniFile.Create(IniFName);
if IniFile.SectionExists('LastFile') then begin
 Reload:=true;
 sFile:=IniFile.ReadString('LastFile','FName',sFile);
 BtFile.OnClick(self);
end;

if IniFile.SectionExists('Counter') then
 SECnt.Value:=(IniFile.ReadInteger('Counter','Cnt',-1))
                                     else SECnt.Value:=0;
IniFile.Free;

StatusBar1.Panels[0].Width:=StatusBar1.Width div 3;
StatusBar1.Panels[1].Width:=StatusBar1.Panels[0].Width;
StatusBar1.Panels[2].Width:=StatusBar1.Panels[0].Width;
//StatusBar1.Panels[3].Width:=StatusBar1.Panels[0].Width;
Label1.Left:=(Form1.Width-Label1.Width) div 2;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
Reload:=true;
BtFile.OnClick(self);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
Button1.OnClick(self);
BtErase.OnClick(self);
BtPrgFlash.OnClick(self);
end;

procedure TForm1.RdFuseClick(Sender: TObject);
var BLow,BHigh:byte;
begin
```

```
  BtChk.OnClick(self);
  if not PrgReady then Exit;
  SendBt($50);
  SendBt($00);
  SendBt($00);
  BLow:=SendBt($ff);
  SendBt($58);
  SendBt($08);
  SendBt($00);
  BHigh:=SendBt($ff);
  if BLow and $80>0 then FL7.Checked:=true else FL7.Checked:=false;
  if BLow and $40>0 then FL6.Checked:=true else FL6.Checked:=false;
  if BLow and $20>0 then FL5.Checked:=true else FL5.Checked:=false;
  if BLow and $10>0 then FL4.Checked:=true else FL4.Checked:=false;
  if BLow and $08>0 then FL3.Checked:=true else FL3.Checked:=false;
  if BLow and $04>0 then FL2.Checked:=true else FL2.Checked:=false;
  if BLow and $02>0 then FL1.Checked:=true else FL1.Checked:=false;
  if BLow and $01>0 then FL0.Checked:=true else FL0.Checked:=false;
  if BHigh and $80>0 then FH7.Checked:=true else FH7.Checked:=false;
  if BHigh and $40>0 then FH6.Checked:=true else FH6.Checked:=false;
  if BHigh and $20>0 then FH5.Checked:=true else FH5.Checked:=false;
  if BHigh and $10>0 then FH4.Checked:=true else FH4.Checked:=false;
  if BHigh and $08>0 then FH3.Checked:=true else FH3.Checked:=false;
  if BHigh and $04>0 then FH2.Checked:=true else FH2.Checked:=false;
  if BHigh and $02>0 then FH1.Checked:=true else FH1.Checked:=false;
  if BHigh and $01>0 then FH0.Checked:=true else FH0.Checked:=false;
end;

procedure TForm1.WrLoFuseClick(Sender: TObject);
var BLow:byte;
begin
  BtChk.OnClick(self);
  if not PrgReady then Exit;
  BLow:=0;
  if FL7.Checked=true then BLow:=BLow+$80;
  if FL6.Checked=true then BLow:=BLow+$40;
  if FL5.Checked=true then BLow:=BLow+$20;
  if FL4.Checked=true then BLow:=BLow+$10;
  if FL3.Checked=true then BLow:=BLow+$08;
  if FL2.Checked=true then BLow:=BLow+$04;
  if FL1.Checked=true then BLow:=BLow+$02;
  if FL0.Checked=true then BLow:=BLow+$01;
  SendBt($ac);
  SendBt($a0);
  SendBt($ff);
  SendBt(BLow);
end;

procedure TForm1.WrHiFuseClick(Sender: TObject);
```



```
var BHigh:byte;
begin
  BtChk.OnClick(self);
  if not PrgReady then Exit;
  BHigh:=0;
  if FH7.Checked=true then BHigh:=BHigh+$80;
  if FH6.Checked=true then BHigh:=BHigh+$40;
  if FH5.Checked=true then BHigh:=BHigh+$20;
  if FH4.Checked=true then BHigh:=BHigh+$10;
  if FH3.Checked=true then BHigh:=BHigh+$08;
  if FH2.Checked=true then BHigh:=BHigh+$04;
  if FH1.Checked=true then BHigh:=BHigh+$02;
  if FH0.Checked=true then BHigh:=BHigh+$01;
  SendBt($ac);
  SendBt($a8);
  SendBt($ff);
  SendBt(BHigh);
end;

procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
IniFile := TIniFile.Create(IniFName);
IniFile.WriteString('LastFile','FName',sFile);
IniFile.WriteInteger('Counter','Cnt',SECnt.Value);
IniFile.Free;
end;

procedure TForm1.Label2DblClick(Sender: TObject);
begin
SECnt.Value:=0;
end;

end.
```

Приложение 4. 8-разрядные RISC-микроконтроллеры фирмы Atmel

Прибор	Флэш [Кбайт]	EEPROM [Кбайт]	SRAM [байт]	Макс. кол-во Вв/Выв	f_{max} [МГц]	V_{CC} [В]	8/16-бит таймеры	ШИМ-каналы	RTC	SPI	UART	TWI	ISP	10-бит АЦП	Аналоговый компаратор	Детектор пониж.	Сторожевой таймер	Встроенный генератор	Аппаратный умножитель	Прерывания	Внешние прерывания	Память программ	Корпус
Automotive AVR																							
ATmega48 Au*	4	0.256	512	23	16	2.7...5.5	2/1	6	●	1+USART	1	●	●	8	●	●	●	●	●	26	26	●	MLF-32, TQFP-32
ATmega88 Au*	8	0.512	1024	23	16	2.7...5.5	2/1	6	●	1+USART	1	●	●	8	●	●	●	●	●	26	26	●	MLF-32, TQFP-32
ATmega168 Au*	16	0.512	1024	23	16	2.7...5.5	2/1	6	●	1+USART	1	●	●	8	●	●	●	●	●	26	26	●	MLF-32, TQFP-32
AT90CAN128 Au*	128	4	4096	53	16	2.7...5.5	2/2	8	●	●	2	—	●	8	●	●	●	●	●	34	8	●	LQFP-64, MLF-64
LCD AVR																							
ATmega169	16	0.5	1024	54	16	1.8...5.5	2/1	4	●	1+USI	1	USI	●	8	●	●	●	●	●	23	17	●	MLF-64, TQFP-64
ATmega329	32	1	2048	54	16	1.8...5.5	2/1	4	●	1+USI	1	USI	●	8	●	●	●	●	●	25	17	●	MLF-64, TQFP-64, TQFP-100
ATmega3290	32	1	2048	69	16	1.8...5.5	2/1	4	●	1+USI	1	USI	●	8	●	●	●	●	●	25	32	●	MLF-64, TQFP-64, TQFP-100
ATmega649	64	2	4096	54	16	1.8...5.5	2/1	4	●	1+USI	1	USI	●	8	●	●	●	●	●	25	17	●	MLF-64, TQFP-64, TQFP-100
ATmega6490	64	2	4096	69	16	1.8...5.5	2/1	4	●	1+USI	1	USI	●	8	●	●	●	●	●	25	32	●	MLF-64, TQFP-64, TQFP-100
Lighting AVR																							
AT90PWM2	8	0.5	512	19	16	2.7...5.5	1/1	7	●	1	●	—	●	8	●	●	●	●	●	—	4	●	SOIC-24
AT90PWM3	8	0.5	512	19	16	2.7...5.5	1/1	12	●	1	●	—	●	11	●	●	●	●	●	—	4	●	MLF-32, SOIC-32 (440 mil)
megaAVR																							
ATmega48	4	0.256	512	23	20	1.8...5.5	2/1	6	●	1+USART	1	●	●	8	●	●	●	●	●	26	26	●	MLF-32, TQFP-32, PDIP-28
ATmega8	8	0.5	1024	23	16	2.7...5.5	2/1	3	●	1	1	●	●	8	●	●	●	●	●	18	2	●	MLF-32, TQFP-32, PDIP-28
ATmega88	8	0.5	1024	23	20	1.8...5.5	2/1	6	●	1+USART	1	●	●	8	●	●	●	●	●	26	26	●	MLF-32, TQFP-32, PDIP-28
ATmega8515	8	0.5	512	35	16	2.7...5.5	1/1	3	—	1	1	—	●	—	—	●	●	●	●	16	3	●	PLCC-44, MLF-44, TQFP-44, PDIP-40
ATmega8535	8	0.5	512	32	16	2.7...5.5	2/1	4	—	1	1	●	●	8	●	●	●	●	●	20	3	●	PLCC-44, MLF-44, TQFP-44, PDIP-40
ATmega16	16	0.5	1024	32	16	2.7...5.5	2/1	4	●	1	1	●	●	8	●	●	●	●	●	20	3	●	MLF-44, TQFP-44, PDIP-40
ATmega162	16	0.5	1024	35	16	1.8...5.5	2/2	6	●	1	2	—	●	—	●	●	●	●	●	28	3	●	MLF-44, TQFP-44, PDIP-40
ATmega164	16	0.512	1024	32	20	1.8...5.5	2/1	6	●	1+USART	2	●	●	8	●	●	●	●	●	31	32	●	MLF-44, TQFP-44, PDIP-40
ATmega165	16	0.5	1024	54	16	1.8...5.5	2/1	4	●	1+USI	1	USI	●	8	●	●	●	●	●	23	17	●	MLF-64, TQFP-64
ATmega168	16	0.5	1024	23	20	1.8...5.5	2/1	6	●	1+USART	1	●	●	8	●	●	●	●	●	26	26	●	MLF-32, TQFP-32, PDIP-28

Прибор	Флэш [Кбайт]	EEPROM [Кбайт]	SRAM [байт]	Макс. кол-во Вв/Выв	f_{max} [MHz]	V_{CC} [B]	8/16-бит таймеры	ШИМ-каналы	RTC	SPI	UART	TWI	ISP	10-бит АЦП	Аналоговый компаратор	Детектор пониж.	Сторожевой таймер	Встроенный генератор	Аппаратный умножитель	Прерывания	Внешние прерывания	Память программ	Корпус
ATmega32	32	1	2048	32	16	2.7...5.5	2/1	4	●	1	1	●	●	8	●	●	●	●	●	19	3	●	MLF-44, TQFP-44, PDIP-40
ATmega324	32	1	2048	32	20	1.8...5.5	2/1	6	●	1+USART	2	●	●	8	●	●	●	●	●	31	32	●	MLF-44, TQFP-44, PDIP-40
ATmega325	32	1	2048	54	16	1.8...5.5	2/1	4	●	1+USI	1	USI	●	8	●	●	●	●	●	23	17	●	MLF-64, TQFP-64
ATmega3250	32	1	2048	69	16	1.8...5.5	2/1	4	●	1+USI	1	USI	●	8	●	●	●	●	●	32	17	●	MLF-64, TQFP-64, TQFP-100
ATmega64	64	2	4096	54	16	2.7...5.5	2/2	8	●	1	2	●	●	8	●	●	●	●	●	34	8	●	MLF-64, TQFP-64
ATmega640	64	4	8192	86	16	1.8...5.5	2/4	16	●	1+USART	4	●	●	16	●	●	●	●	●	57	32	●	TQFP-100
ATmega644	64	2	4096	32	20	1.8...5.5	2/1	6	●	1+USART	1	●	●	8	●	●	●	●	●	31	32	●	MLF-44, TQFP-44, PDIP-40
ATmega645	64	2	4096	54	16	1.8...5.5	2/1	4	●	1+USI	1	USI	●	8	●	●	●	●	●	23	17	●	MLF-64, TQFP-64
ATmega6450	64	2	4096	69	16	1.8...5.5	2/1	4	●	1+USI	1	USI	●	8	●	●	●	●	●	32	17	●	MLF-64, TQFP-64, TQFP-100
ATmega128	128	4	4096	53	16	2.7...5.5	2/2	8	●	1	2	●	●	8	●	●	●	●	●	34	8	●	MLF-64, TQFP-64
ATmega1280	128	4	8192	86	16	1.8...5.5	2/4	16	●	1+USART	4	●	●	16	●	●	●	●	●	57	32	●	TQFP-100
ATmega1281	128	4	8192	54	16	1.8...5.5	2/4	9	●	1+USART	2	●	●	8	●	●	●	●	●	48	17	●	MLF-64, TQFP-64
ATmega2560	256	4	8192	86	16	1.8...5.5	2/4	16	●	1+USART	4	●	●	16	●	●	●	●	●	57	32	●	TQFP-100
ATmega2561	256	4	8192	54	16	1.8...5.5	2/4	9	●	1+USART	2	●	●	8	●	●	●	●	●	48	17	●	MLF-64, TQFP-64
Smart Battery AVR																							
ATmega406	40	0.512	2048	18	1	4...25	1/1	1	●	—	—	●	●	—	●	●	●	●	●	23	4	●	LQFP-48
tinyAVR																							
ATtiny11	1	—	—	6	6	2.7...5.5	1/—	—	—	—	—	—	—	—	●	—	●	●	—	4	1	—	PDIP-8, SOIC-8 (209 mil)
ATtiny12	1	0.064	—	6	8	1.8...5.5	1/—	—	—	—	—	—	●	—	●	●	●	●	—	5	1	—	PDIP-8, SOIC-8 (209 mil)
ATtiny13	1	0.064	64B+ 32 reg	6	20	1.8...5.5	1/—	2	—	—	—	—	●	4	●	●	●	●	—	9	6	●	SOIC-8 (150 mil), SOIC-8 (209 mil), PDIP-8, MLF-20
ATtiny15L	1	0.0625	—	6	1.6	2.7...5.5	2/—	1	—	—	—	—	●	4	●	●	●	●	—	8	1(+5)	—	SOIC-8 (209 mil), PDIP-8
ATtiny2313	2	0.128	128	18	20	1.8...5.5	1/1	4	—	USI	1	USI	●	—	●	●	●	●	—	8	2	●	SOIC-20, PDIP-20, MLF-20
ATtiny24	2	0.128	128	12	20	1.8...5.5	1/1	4	●	USI	—	USI	●	8	●	●	●	●	—	17	12	●	PDIP-14, SOIC-14, MLF-20
ATtiny25	2	0.128	128	6	20	1.8...5.5	2/—	4	●	USI	—	USI	●	4	●	●	●	●	—	15	7	●	MLF-20, SOIC-8 (209 mil), PDIP-8
ATtiny26	2	0.125	128	16	16	2.7...5.5	2/—	2	—	USI	—	USI	●	11	●	●	●	●	—	11	1	—	MLF-32, SOIC-20, PDIP-20
ATtiny28L	2	—	32	11	4	1.8...5.5	1/—	—	—	—	—	—	—	—	●	—	●	●	—	5	2(+8)	—	MLF-32, TQFP-32, PDIP-28
ATtiny44	4	0.256	256	12	20	1.8...5.5	1/1	4	●	USI	—	USI	●	8	●	●	●	●	—	17	12	●	PDIP-14, SOIC-14, MLF-20
ATtiny45	4	0.256	256	6	20	1.8...5.5	2/—	4	●	USI	—	USI	●	4	●	●	●	●	—	15	7	●	MLF-20, SOIC-8 (209 mil), PDIP-8
ATtiny84	8	0.512	512	12	20	1.8...5.5	1/1	4	●	USI	—	USI	●	8	●	●	●	●	—	17	12	●	PDIP-14, SOIC-14, MLF-20
ATtiny85	8	0.512	512	6	20	1.8...5.5	2/—	4	●	USI	—	USI	●	4	●	●	●	●	—	15	7	●	MLF-20, SOIC-8 (209 mil), PDIP-8

*Au — Automotive

Примечание: RISC — это аббревиатура от Reduced Instruction Set Computer (компьютер с сокращенным набором команд), в отличие от CISC — аббревиатура от Comlex Instruction Set Computer (компьютер с полным набором команд).

Материалы, размещенные на компакт-диске

После запуска компакт-диска открывается окно главного меню:

- Исходные тексты программ
- AVR Studio
- Документация на микроконтроллеры (Data Sheets)
- Документация по применению (App Notes)
- Acrobat Reader

В разделе «Исходные тексты программ» представлены листинги всех программ, приведенных в книге.

Раздел «AVR Studio» содержит дистрибутивы последней версии AVR Studio 4.12 build 462 с дополнением Service Pack 1, а также версии 4.08, в которой автор проводил отладку всех программ. Здесь же располагаются следующие документы: описание протокола обмена данными AVRISP mkII, руководство по разработке приложений на базе микроконтроллеров с внутрисистемным перепрограммированием флэш-памяти, информацию об изменениях в AVR Studio 4, описание набора команд микроконтроллеров AVR, рекламные материалы по пакету AVR Studio, а также вводный курс для начинающих разработчиков на AVR.

В разделе «Документация на микроконтроллеры» приводится полная англоязычная документация на все выпускаемые в настоящий момент 8-разрядные микроконтроллеры AVR компании ATMEL.

В раздел «Документация по применению» включено большое количество фирменных материалов на английском языке по микроконтроллерам AVR с сайта http://www.atmel.com.

Из главного меню можно установить бесплатную программу Acrobat Reader для чтения файлов в формате pdf.

КОМПОНЕНТЫ ДЛЯ ТВОРЧЕСТВА

- ЭЛЕКТРОННЫЕ КОМПОНЕНТЫ
- ЭЛЕКТРОМЕХАНИЧЕСКИЕ КОМПОНЕНТЫ
- СИСТЕМЫ ПРОМЫШЛЕННОЙ АВТОМАТИКИ
- КОМПЛЕКТУЮЩИЕ ДЛЯ ВОЛС
- КОМПОНЕНТЫ ДЛЯ БЕСПРОВОДНОЙ СВЯЗИ

WWW.EFO.RU

ПРЯМЫЕ ДИСТРИБЬЮТОРСКИЕ И ПАРТНЕРСКИЕ ОТНОШЕНИЯ

ВПЕРВЫЕ В РОССИИ
ВСЯ ИНФОРМАЦИЯ
ПО ПРОДУКЦИИ
КОРПОРАЦИИ ATMEL
НА ОДНОМ DVD-ДИСКЕ

ЗВОНИТЕ И ЗАКАЗЫВАЙТЕ!

ВЕДУЩИЕ ПОЗИЦИИ СРЕДИ ПОСТАВЩИКОВ ЭЛЕКТРОННЫХ КОМПОНЕНТОВ РОССИИ

ОПТИМАЛЬНЫЕ УСЛОВИЯ ДЛЯ СОТРУДНИЧЕСТВА

КВАЛИФИЦИРОВАННАЯ ТЕХНИЧЕСКАЯ ПОДДЕРЖКА

ООО «ЭФО» ▪ ПОСТАВКА ПРОДУКЦИИ И ТЕХНИЧЕСКАЯ ПОДДЕРЖКА

САНКТ-ПЕТЕРБУРГ:
Т.: (812) 327-8654
Ф.: (812) 320-1819
E-MAIL: ZAV@EFO.RU

МОСКВА:
ТЕЛ./ФАКС: (495) 933-0743
E-MAIL: MOSCOW@EFO.RU

ЕКАТЕРИНБУРГ:
ТЕЛ./ФАКС: (343) 378-4122
E-MAIL: URAL@EFO.RU

КАЗАНЬ:
ТЕЛ./ФАКС: (843) 518-7920
E-MAIL: KAZAN@EFO.RU

РОСТОВ-НА-ДОНУ:
ТЕЛ./ФАКС: (863) 220-3679
E-MAIL: ROSTOV@EFO.RU

Система менеджмента качества сертифицирована по ГОСТ Р ИСО 9001-2001, МС ИСО 9001-2000 и СРПП ВТ в системах «Военный Регистр» и «Оборонсертифика»