КАЗАХСКИЙ НАЦИОНАЛЬНЫЙ УНИВЕРСИТЕТ ИМЕНИ АЛЬ-ФАРАБИ

Ибраимов М.К

FPGA

Алматы
«Қазақ университеті»
2023

УДК 621.37
ББК 22.3
Ж 27

Рекомендовано к изданию
Ученым советом физико-технического факультета
И Редакционно-издательским советом КазНУ имени аль-Фараби

Рецензенты :

PhD доктор, доцент ***А.К. Саймбетов***
PhD доктор ***С.Н.Ахтанов***

Ибраимов М.К. FPGA / Алматы: Қазақ университеті, 2023. – 351 с.
ISBN 978-601-04-0307-9

Оқу құралы 5В071900-Радиотехника, электроника және телекоммуникациялар мамандық бағыты бойынша оқитын студенттерге арналған. Осы оқу құралында радиоқабылдағыш құрылғылардың элементтер жүйелерінің теориялық негіздері, радиоэлектронды байланыс жүйелердің пайдалануы көрсетілген, сонымен қатар функционалдық түйіндер, электр тізбегінің есептелуі қарастырылған.

УДК 621.37
ББК 22.3

ISBN 978-601-04-0307-9

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Современное состояние цифровой электроники требует использования программируемых интегральных микросхем, которые позволяют реализовывать узлы электронных устройств в программном виде. Описания функционирования цифровых схем на программном уровне ускоряет время проектирования и облегчает тестирования работы электронных устройств. Интегральной схемой для проектирования цифровых устройств на уровне логических схем является программируемая логическая интегральная схема (ПЛИС). На основе ПЛИС в наше время проектируется большая часть цифровых устройств, где происходить цифровая обработка сигналов. Использовав язык описания аппаратуры Verilog, программируется структура ПЛИС.

Первая глава книги рассматривает особенности ПЛИС и ее историю появления. Описываются работа полевых транзисторов, которые являются строительным блоком современных цифровых интегральных схем, а также ПЛИС.

Главы начиная со второй до седьмой описывают синтаксис языка описания аппаратуры Verilog. Рассматриваются уровни проектирования цифровых устройств и их отличие. Проектировать цифровые схемы на Verilog является очень гибким. Потому что, она дает возможность проектировать цифровые схемы на уровне логических вентилей используя соответствующий набор функции или проектировать схемы акцентируя внимания только на ее поведении. В этих главах рассматривается не только синтаксис языка Verilog, но и примеры реализации логических элементов, комбинационных и последовательных устройств. Объясняется различие при проектировании комбинационных и последовательных устройств отдельно и вместе в одном проекте. Приведены примеры и временные диаграммы показывающее результат работы.

Главы с восьмой по одиннадцатой описывают логические уровни сигналов, реализацию логических элементов на уровне

транзисторов, а также алгебру логики. Рассматриваются такие термины как вентиль, который является структурным элементом ПЛИС. Объясняется работа базовых и составных логических операции и элементов, а также описываются реализация логических схем на основе только одной логической функции.

Главы начиная с двенадцатой по четырнадцатой описывает область применения ПЛИС. Рассматривается классификация ПЛИС по разным параметрам. Описывается структурная схема конкретной ПЛИС от компании Xilinx.

После прочтения данной книги и практических применении представленных примеров на Verilog читатель будет понимать область применения ПЛИС, их роль в цифровой электронике и проектировать несложные цифровые схемы на языке описания аппаратуры Verilog.

ГЛАВА 1

ВВЕДЕНИЕ В ПЛИС

Программируемая логическая интегральная схема (ПЛИС, англ. programmable logic device, PLD) – электронный компонент (интегральная микросхема), используемый для создания конфигурируемых цифровых электронных схем. В отличие от обычных цифровых микросхем, логика работы ПЛИС не определяется при изготовлении, а задаётся посредством программирования (проектирования). Для программирования используются программатор и IDE (отладочная среда), позволяющие задать желаемую структуру цифрового устройства в виде принципиальной электрической схемы или программы на специальных языках описания аппаратуры: Verilog, VHDL, AHDL и др. Для понимания работы ПЛИС сперва рассмотрим, как функционируют простые цифровые устройства.

Основой для изготовления ИС служат кристаллы кремния. На поверхности полупроводника можно формировать различные структуры. Если в локальные участки кристалла кремния вводить небольшое количество примесей какого-либо пятивалентного вещества, например мышьяка или сурьмы, можно получить участки полупроводника n-типа, в котором проводимость возникает за счет наличия свободных электронов. Если же в другие участки вводить примеси трехвалентного вещества, например индия, то можно получить участки полупроводника p-типа, где проводимость обеспечивается за счет так называемых «дырок» - нехватки электрона в связях кристаллической решетки. Кроме этого, на поверхность кристалла можно наносить изолирующие слои, и слои металла. Если располагать эти элементы в определенном порядке, можно получать различные электронные компоненты с заданными электрическими свойствами.

Транзисторы являются основным элементом всех цифровых логических компонентов. Однако, работая над проектом, состоящим из нескольких миллионов транзисторов, разработчик не может мыслить на уровне отдельных транзисторов. Поэтому транзисторы объединяются в более абстрактные структуры, называемые вентилями.Логический вентиль – базовый элемент цифровой схемы, выполняющий элементарную логическую операцию, преобразуя таким образом множество входных логических сигналов в выходной логический сигнал. Логика работы вентиля основана на битовых операциях с входными цифровыми сигналами в качестве операндов. При создании цифровой схемы вентили соединяют между собой, при этом выход используемого вентиля должен быть подключён к одному или к нескольким входам других вентилей. В настоящее время в созданных человеком цифровых устройствах доминируют электронные логические вентили на базе полевых транзисторов.

1.1. КМОП-технология

КМОП (комплементарная структура металл-оксид-полупроводник; *англ. CMOS, complementary metal-oxide-semiconductor*) – набор полупроводниковых технологий построения интегральных микросхем и соответствующая ей схемотехника микросхем. Подавляющее большинство современных цифровых ИС изготавливаются по КМОП-технологии.

В технологии КМОП используются полевые транзисторы с изолированным затвором с каналами разной проводимости. Отличительной особенностью схем КМОП по сравнению с биполярными технологиями (ТТЛ, ЭСЛ и др.) является очень малое энергопотребление в статическом режиме (в большинстве случаев можно считать, что энергия потребляется только во время переключения логических состояний). Отличительной особенностью структуры КМОП по сравнению с другими МОП – структурами (n-МОП, p-МОП) является наличие как n –, так и p– канальных полевых транзисторов, локализованных в одном

месте кристалла. Вследствие меньшего расстояния между элементами КМОП – схемы обладают большим быстродействием и меньшим энергопотреблением, однако при этом характеризуются более сложным технологическим процессом изготовления и меньшей плотностью упаковки на поверхности кристалла.

По аналогичной технологии выпускаются дискретные полевые транзисторы с изолированным затвором *(англ. MOSFET, metal-oxide-semiconductor field-effect transistor).*

1.1.1 Полевой (униполярный) транзистор

Полевой (униполярный) транзистор – полупроводниковый прибор, принцип действия которого основан на управлении электрическим сопротивлением токопроводящего канала поперечным электрическим полем, создаваемым приложенным к затвору напряжением.

Электроды полевого транзистора называются:

- исток (*англ. source*) – электрод, из которого в канал испускаются основные носители заряда;
- сток (*англ. drain*) – электрод, через который канал принимает основные носители заряда;
- затвор (*англ. gate*) – электрод, служащий для регулирования поперечного сечения канала.

На рисунках 1.1 а, б показаны графические изображения МОП – транзисторов в советской и российской технической литературе. На рисунках 1.1 в, г показаны упрощенные графические изображения в зарубежной технической литературе.

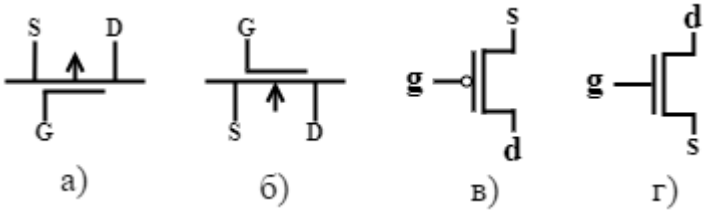


Рисунок 1.1 – Графическое обозначения МОП – транзисторов

В отличие от биполярного транзистора полевой контролируется именно напряжением (уровнем логического «0» иди «1»), а не током, т.е. в открытом состоянии ток через затвор не идёт». Внутри МОП – транзистора как бы «вставлен» защитный диод, который всегда направлен против основного тока (Рис. 1.2).

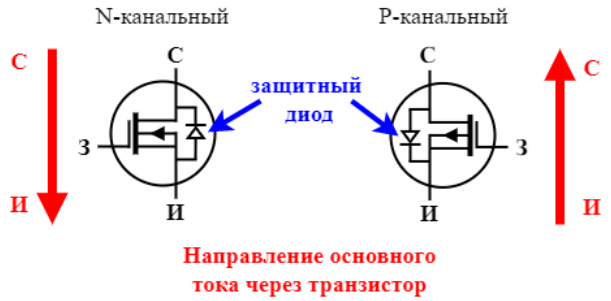


Рисунок 1.2 – Направление токов через МОП – транзисторы

Защитный диод препятствует протеканию тока между стоком и истоком и тем самым защищает данные контакты от перегорания.

МОП-транзисторы бывают двух типов: с p–каналом (pМОП) и с n–каналом (nМОП) (Рис. 1.3).

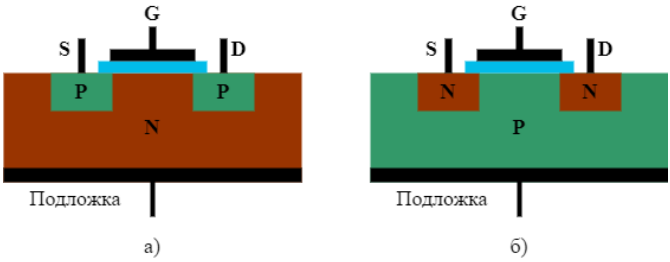


Рисунок 1.3 – Транзистор с p – каналом (а) и с n – каналом (б)

Если в полупроводнике n-типа с отрицательными носителями заряда имеются две области p-типа с положительными носителями заряда, то проводимостью между этими областями можно управлять при помощи изолированного электрода. Если на этом электроде создать отрицательный заряд (Рис. 1.4 позиция 1), то за счет электростатической индукции в полупроводнике возникает область повышенной концентрации положительных зарядов (Рис. 1.4 позиция 2), обеспечивающая проводимость между областями p-типа. Такой транзистор называется p – канальным и открыт тогда, когда на управляющем электроде присутствует отрицательный по отношению к подложке потенциал (Рис. 1.4).

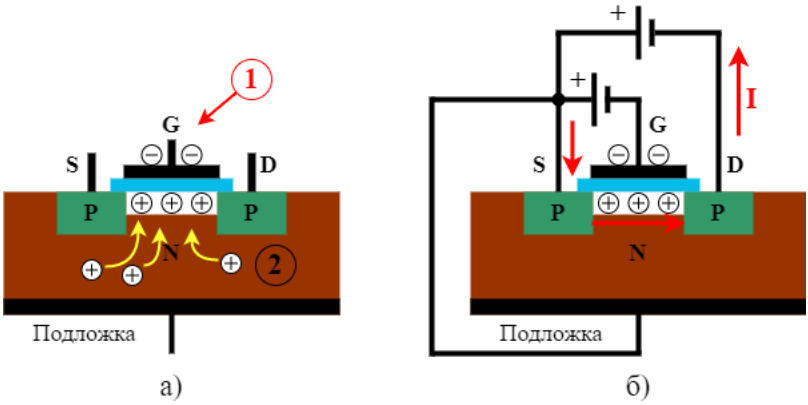


Рисунок 1.4 – Работа pМОП-транзистора

Если. в полупроводнике p – типа с положительными носителями зарядов создать области n – типа с отрицательными носителями, то проводимость появится (Рис. 1.5 позиция 2) при наличии на изолированном электроде положительного заряда (Рис. 1.5 позиция 1). Такой транзистор называется n-канальным, и открывается тогда, когда на управляющем электроде присутствует положительный по отношению к подложке потенциал.

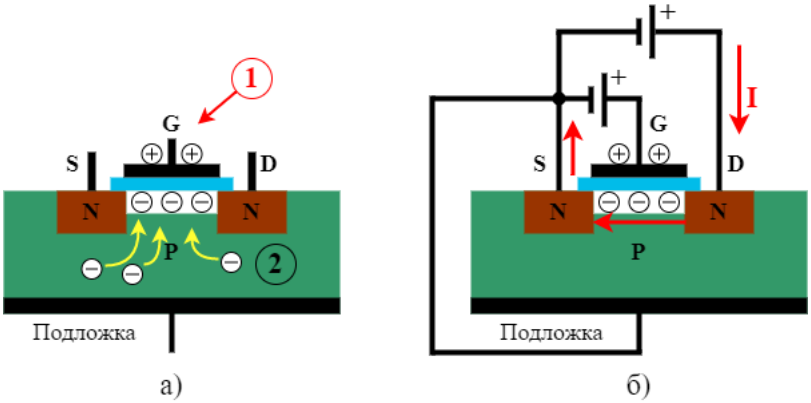


Рисунок 1.5 – Работа nМОП-транзистора

p-канальные и n-канальные транзисторы являются как бы зеркальным отражением друг друга. Пару таких транзисторов, соединенных последовательно называют, комплементарной парой. А к названию технологии, использующей такие пары транзисторов добавилась буква «К» – КМОП (*англ.«CMOS» – complementary metal-oxide-semiconductor*) (Рис. 1.6).

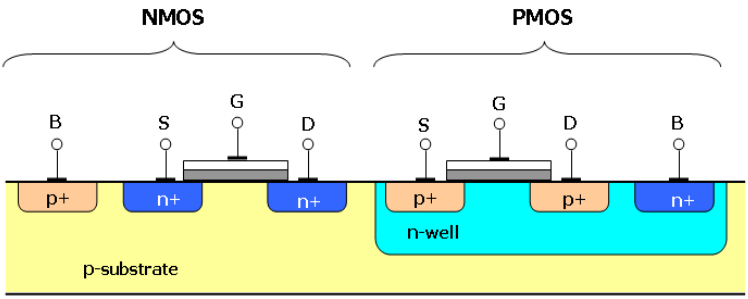


Рисунок 1.6 – p-MOS и n-MOS транзисторы в разрезе. Вывод B (подложка) соединяется с S для n-MOS и с D для p-MOS транзисторов

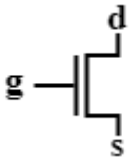
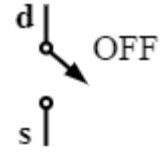
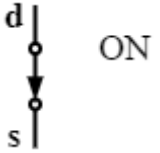
В обоих типах транзистора выделяют следующие части: исток (source – S), сток (drain – D), затвор (gate – G), подложку p-

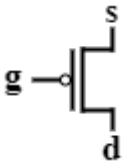
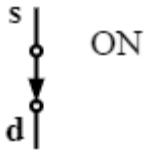
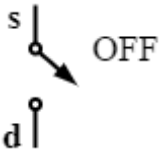
substrate, области с различными проводимостями n+ и p+ и колодец с проводимостью n (для PMOS транзистора). Source или исток является источником основным носителей заряда, drain или сток – это приемник основных носителей заряда. Для обоих типов затвор G электрически изолирован от подложки.

Рассмотрим NMOS транзистор. S и D соединены с областями полупроводника с проводимостью n+. На подложку подается напряжение истока S. Если к затвору не прикладывать положительного потенциала, то проводимость между стоком и истоком очень мала. Однако при прикладывании положительного потенциала (больше потенциала истока) к затвору, между областями n+ образуется n-канал, по которому могут перемещаться основные носители заряда и проводимость возрастает. Аналогично для PMOS транзистора, S и D соединены с областями полупроводника с проводимостью p+. На колодец n-well подается напряжение стока D. Теперь, в отличие NMOS транзистора, если к затвору прикладывать потенциал равный потенциалу стока, то его проводимость будет мала. Если же приложить потенциал меньший чем потенциал стока, то его проводимость возрастет.

На базе КМОП транзисторов можно построить любой логический элемент. В таблице 1.1 сведены данные по работе МОП-транзисторов.

Таблица 1.1. МОП-транзисторы

Тип МОП-транзисторов	Графическое обозначение	Значение напряжения на затворе	
		g = 0	**g = 1**
nMOS			

pMOS			

1.2. Микросхемы

Соединив много транзисторов, можно создать цифровую схему. Получается правда очень громоздко. Тогда родилась идея объединения нескольких транзисторов на одном куске полупроводника (кремния). Так была создана первая интегральная схема (ИС) 1958 г. Первые ИС включали в себя совсем мало транзисторов (десятки), но потом их количество стало расти экспоненциально.

Исторически можно выделить этапы роста плотности транзисторов на интегральной схеме (ИС):

- Малая ИС (МИС) – до 100 элементов в кристалле.
- Средняя ИС (СИС) – до 1000 элементов в кристалле.
- Большая ИС (БИС) – до 10000 элементов в кристалле.
- Сверхбольшая ИС (СБИС) – до 1 миллиона элементов в кристалле.
- Ультрабольшая ИС (УБИС) – до 1 миллиарда элементов в кристалле.

Технология изготовления современных ИС очень сложна. На разработку проекта нового чипа уходит около полутора лет, при этом затраты могут доходить до миллиарда долларов США. Сначала необходимо разработать проект микросхемы в САПР (система автоматического проектирования). Раньше необходимо было задавать расположение каждого транзистора на куске кремния. Теперь, когда типовые линейные размеры транзисторов приближаются к нескольким нм, а число транзисторов на чипе перевалило за миллиард, задавать руками расположение отдельного транзистора не представляется возможным. В современном процессе проектирования используются библиотеки элементов – уже соединенные транзисторы вместе.

Транзисторы объединяются в логические элементы и функции, логические элементы в функциональные узлы, узлы в блоки и интерфейсы. Разработчик может пользоваться уже готовыми элементами при проектировании, и это существенно сокращается время разработки. Непосредственно сам процесс изготовления микросхемы занимает до 8-ми недель и происходит на специализированной фабрике. Сама ИС – это слоистая структура (10–15 слоев), слои бывают полупроводниковыми и металлическими. На полупроводниковых слоях изготовляются транзисторы, на слоях металлизации – соединения между ними и внешние выводы. На полупроводниковых слоях необходимо определить месторасположение каждого транзистора, т.е. указать, где будут области с положительной проводимостью, а где – с отрицательной. Структура слоев задается процессом, называемым фотолитография.

1.3. Классификация микросхем

Теперь, когда мы поняли, как примерно делают микросхемы, мы можем рассмотреть их классификацию. Глобально современные микросхемы можно разделить на два класса:

1. С жестко заданной структурой (заказные микросхемы или *ASIC – Application Specific Integrated Circuit*) ;
2. Со структурой которых можно менять или перепрограммировать.

Рассмотрим сперва ASIC. Сейчас различают 4 класса технологий, по которым делают микросхемы:

1. Вентильные матрицы;
2. Структурированные ASIC;
3. Схемы на стандартных элементах;
4. Полностью заказные микросхемы.

1.3.1. Полностью заказные микросхемы

Проектировщики микросхемы по этой технологии могут создавать все слои (полупроводниковые и металлизации) так как

им угодно, т.е. разработчик может управлять расположением каждого транзистора в микросхеме. Это очень трудоемкий долгий и дорогой процесс. Но при этом получаются наиболее эффективные схемы. Тем не менее, сегодня редко можно встретить микросхему, выполненную по этой технологии. Этот подход чаще используется для проектирования малых критически важных частей будущей микросхемы или при создании базовых логических элементов других технологий.

1.3.2. Схемы на стандартных элементах

Устройство создается из набора заранее определенных компонентов, выполняющих простые логические функции и выполненных по технологии полностью заказных микросхем. Эти компоненты образуют так называемую библиотеку элементов. Библиотека элементов создается производителем микросхемы (заводом), а заказчик или разработчик микросхемы по этой технологии вправе брать любые элементы из этой библиотеки и размещать их в любых частях кристалла. При этом разработка микросхемы становится легче, т.к. проектировщик не управляет расположениями отдельных транзисторов, а оперирует уж готовыми логическими элементами. Хотя при этом подходе все равно необходимо изготавливать все слои. По этой технологии сейчас выполнено большинство высокотехнологичных микросхем.

1.3.3. Вентильные матрицы

Устройство состоит из готовых ячеек, находящихся в определенных местах и выполняющих определенные логические функции. Разработчик вправе только соединять эти элементы произвольным образом для достижения нужной ему функциональности. Этот подход значительно упрощает создание схемы, так как необходимо создать только нужную схему соединений элементов с помощью слоев металлизации (полупроводниковые слои уже созданы заранее). В итоге уменьшается время создания схемы и ее стоимость, однако часть

ресурсов изначальной заготовки может оказаться не востребованной, и многие параметры схемы (максимальная тактовая частота, энергопотребление…) будут хуже по сравнению с той же схемой, но выполненной по технологии, например, схем на стандартных элементах.

1.4. Программируемые логические интегральные схемы (ПЛИС)

ПЛИС являются ярким представителем микросхем, структуру которых можно менять. Состоят они из набора так называемых логических блоков и матрицы программируемых соединений между ними (Рис. 1.7).

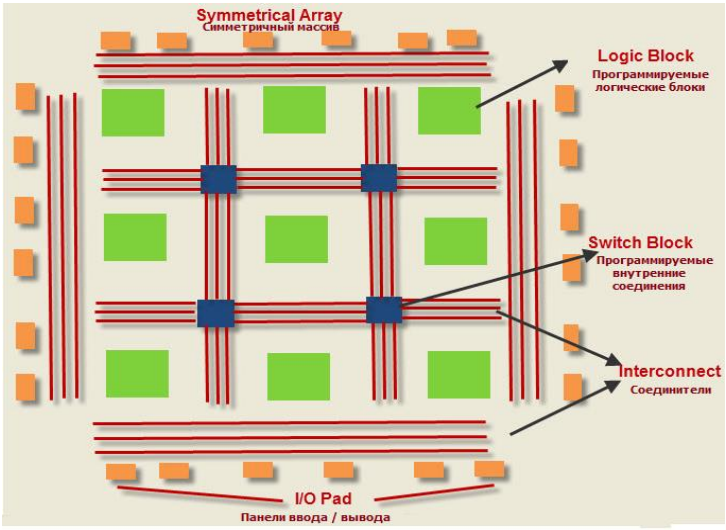


Рисунок 1.7 – Структура ПЛИС

В отличие от ячеек, используемых в технологиях схем на стандартных элементах или вентильных матриц, логические блоки ПЛИС, во-первых, все одинаковые (т.е. нет никаких библиотек элементов), во-вторых, имеют значительно больший

размер, и наконец, могут быть «запрограммированы» на выполнение разных функций.

Как правило, логический блок ПЛИС может быть настроен на реализацию любой логической функции элементов и для хранения одного бита данных в триггере. Внизу представлена упрощенная структура логического блока ПЛИС, включающая в себя таблицу соответствия LUT на 4 входа (здесь для простоты объяснения), триггер и мультиплексор (Рис. 1.8).

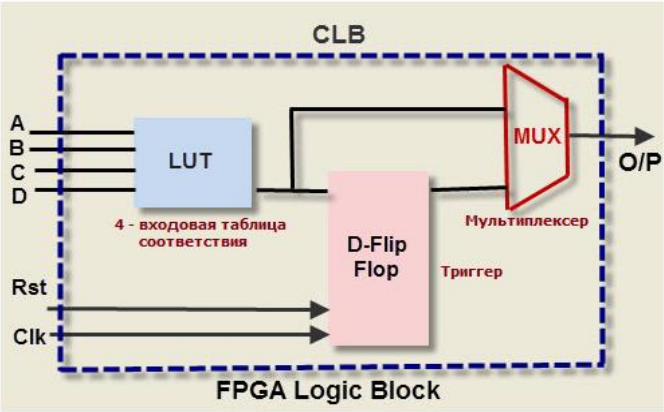


Рисунок 1.8 – Упрощенная структура логического блока (CLB) ПЛИС

Любая логическая функция может быть представлена своей таблицей истинности. Таблица истинности может быть представлена как массив памяти, адресами которой являются аргументы логической функции, а значения, записанные в ячейки этого массива памяти, будут являться значениями функции (Рис. 1.9).

Массивы памяти, представляющие каждую таблицу истинности в каждом логическом блоке, включены в конфигурационную память. Матрица соединений также может быть запрограммирована и позволяет соединять эти блоки в разные конфигурации. Программирование схемы достигается следующим образом. Вся схема ПЛИС «пронизана» ячейками конфигурационного ОЗУ. Каждая ячейка этого ОЗУ – это

элемент, который может содержать либо «0» либо «1». Далее рассмотрим соединение блоков.

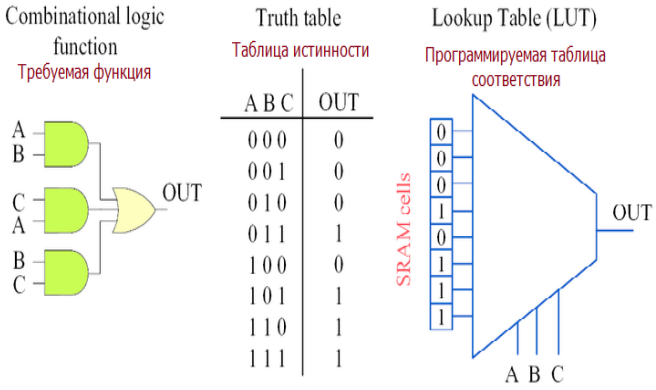


Рисунок 1.9 – Соответствие между LUT и логической функцией

Блоки соединяются проводниками через ключи, к каждому из которых подключена одна ячейка конфигурационного ОЗУ. Если ячейка содержит 1, то ключ замкнут, и соединение есть, если 0, то ключ разомкнут, и соединения нет. Программирование логического блока на выполнение какой-либо функции производится также с помощью этого конфигурационного ОЗУ.

Программирование ПЛИС заключается в процессе записи соответствующих значений в конфигурационную память кристалла. Оно может быть осуществлено просто загрузкой значений с компьютера через соответствующий разъем на плате.

В итоге с использованием ПЛИС можно создавать практически любые цифровые схемы. Конечно, их параметры будут далеки от наилучших сравнительно со схемами, выполненными по технологии схем на стандартных элементах (будет не такая высокая максимальная тактовая частота, больше энергопотребление, часть схемы использоваться не будет, т. к. использовать ресурсы ПЛИС на 100% невозможно). Но при этом цена разработки схемы будет в тысячи раз меньше, и становится возможным разрабатывать свои схемы в учебных и научных целях.

ГЛАВА 2

ВВЕДЕНИЕ В VERILOG

2.1. ПЛИС

ПЛИС позволяет проектировать цифровые схемы. А цифровые схемы это просто набор логических элементов (and, or, nor и др.), соединенных вместе для выполнения конкретной задачи. Проекты, которые вы создаете, могут варьироваться от чего-то такого простого, как счетчик, который мигает светодиодом, до чего-то столь же сложного, как многоядерный процессор. Проектирование цифровых схем на ПЛИС осуществляется по средствам программирования на языках описания аппаратуры. Языки описания аппаратуры (HDL) описывают архитектуру и поведение дискретных электронных систем. Использование HDL – это новая технология проектирования цифровых схем, кардинально отличающаяся от различных схемотехнических подходов, связанных с использованием графических редакторов. Современные языки проектирования позволяют не только описывать структуру и поведения устройства, но также подготавливает последовательность тестовых векторов для анализа, выполнять проверку результатов моделирования, эмулировать внешнюю среду проекта, различными способами отображать результаты моделирования и др.

Другими словами, язык проектирования – это мощный инструмент, используемый на многих этапах представления проекта: от логического синтеза до функционального и временного моделирования.

Наиболее широко используемыми языками проектирования являются Verilog и VHDL.

2.2. Язык описания аппаратуры Verilog

Verilog наряду с конкурирующим языком VHDL, это наиболее распространенный способ программирования FPGA. Вы, вероятно, обнаружили, что программирование ПЛИС с использованием схемы знакомо, поэтому, зачем вам изучать сложный язык программирования, чтобы делать то же самое? Ответ таков: на самом деле, когда проекты становятся все более и более сложными, проще представить дизайн с использованием языка программирования, чем рисовать его. Verilog выглядит как язык программирования, и действительно, вы найдете операторы «if», блоки кода и другие подобные программам конструкции, включая возможность сложения и вычитания чисел.

Одна из главных ошибок начинающих заключается в том, что они думают о коде на HDL как о компьютерной программе, а не как о подспорье для описания цифровой аппаратуры. Если вы не представляете, хотя бы примерно, во что должен синтезироваться ваш код на HDL, то, скорее всего, результат вам не понравится. Ваша цифровая схема может получиться гораздо больше, чем нужно, или может оказаться, что ваш код симулируется правильно, но не может быть реализован в аппаратуре. Вместо этого, вы должны думать над вашей разработкой в понятиях комбинационной логики, регистров и конечных автоматов. Нарисуйте эти блоки на бумаге и покажите, как они будут подключены до того, как вы начнете писать код.

В Verilog модуль может быть определен с использованием четырех различных уровней абстракции.

- Поведенческий (Behavioral) или алгоритмический уровень: это самый высокий уровень абстракции. Модуль может быть реализован с точки зрения алгоритма проектирования. Проектировщику не нужно иметь никаких знаний о реализации оборудования.
- Уровень потока данных (data flow): на этом уровне модуль разработан с указанием потока данных. Проектировщик должен знать, как данные передаются между различными регистрами проекта.

- Уровень вентилей (gate level): модуль реализован с точки зрения логических элементов и взаимосвязей между этими элементами. Проектировщик должен знать схему проекта на уровне логических элементов.
- Уровень переключателя (switch level): это самый низкий уровень абстракции. Конструкция реализована с использованием переключателей / транзисторов.

Моделирование на *уровне вентилей* - фактически самый низкий уровень абстракции, потому что абстракция на уровне коммутатора используется редко. В общем, моделирование на уровне вентилей используется для реализации модулей самого низкого уровня в таких цифровых схемах, как полный сумматор, мультиплексоры и т. д. В Verilog HDL есть примитивы вентилей для всех базовых логических элементов. Таких как, *and, nand, or, nor, ,xor, xnor, not, buf* и т.д. Синтаксис использования примитивов

$$and\ [name\_optional]\ (out, in1, in2, \dots, inN)$$

Здесь, ключевое слово *and* создает копию логического элемента «И», $name\_optional$ имя созданной копии логического элемента, его задавать не обязательно. Аргументы $out, in1, in2, \dots, inN$ является входами и выходами создаваемого элемента, первым всегда записывается единственный выход, а после входы в количестве больше двух.

Проектирование на *уровне потока* данных реализуется использованием оператора непрерывного присваивания *assign.* Синтаксис оператора *assign***:**

$$assign\ [signal\ name] = [expression]$$

Каждое непрерывное присваивания можно рассматривать как часть схемы. Сигнал с левой стороны является выходом, а сигналы, используемые в выражении с правой стороны, являются входами. Выражение описывает функцию этой схемы. Например, рассмотрим утверждение

$$assign\ out = in1\ \&\ in2$$

Это схема, которая выполняет операцию *and*. Когда *in1* или *in2* изменяет свое значение, этот оператор активируется и выражение вычисляется. Новое значение присваивается выходному сигналу *out* после задержки распространения.

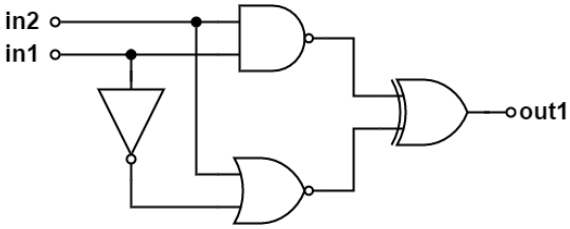
Работа на *поведенческом уровне* подразумевает использование блока *always.* Блок *always* можно представить как черный ящик, поведение которого описывается внутренними процедурными инструкциями. Процедурные инструкции включают в себя большое разнообразие конструкций, но многие из них не имеют четких аппаратных аналогов. Плохо закодированный блок *always* часто приводит к излишне сложной реализации или вообще не может быть синтезирован. Синтаксис использование блока *always:*

```
always@([список чувствительности ])
 begin
 ... процедурные инструкции ...
 end
```

Когда какой-либо сигнал из *списка чувствительности* изменяется или происходит событие, блок *always* активируется и выполняются внутренние *процедурные инструкции*.

2.3. Проектирование схем в Verilog

Как говорилось выше, язык описания аппаратуры Verilog предоставляет несколько вариантов проектирования цифровых схем. Для демонстрации этой возможности рассмотрим логическую схему и ее таблицу истинности показано на рисунке 2.1.



in1	in2	out1
0	0	1
0	1	1
1	0	0
1	1	0

Рисунок 2.1 – Логическая схема и ее таблица истинности

Проектирования схемы на уровне:

Gate level

```
module logic1(
   output out1,
   input in1,
   input in2
);
   wire w1, w2, w3;
   not (w1, in1);
   nand (w2, in1, in2);
   nor (w3, w1, in2);
   xor (out1, w2, w3);
endmodule
```

Data flow

```
module logic1(
   output out1,
   input in1,
   input in2
   );
   assign out1 = !(in1 & in2) ^ (!(!in1 | in2));
endmodule
```

или

```
module logic1(
   output out1,
   input in1,
```

```
    input in2
  );
    wire w1, w2, w3;
    assign w1 = !in1;
    assign w2 = !(in1 & in2);
    assign w3 = !(w1 | in2);
    assign out1 = w2 ^ w3;
  endmodule
```

Behavioral

```
  module logic1(
    output reg out1,
    input in1,
    input in2
  );
    always@(in1 or in2)
    begin
      case({in1, in2})
        2'b00 : out1 = 1'b1;
        2'b01 : out1 = 1'b0;
        2'b10 : out1 = 1'b1;
        2'b11 : out1 = 1'b0;
      endcase
    end
  endmodule
```

ГЛАВА 3

ПРОГРАММИРОВАНИЕ БАЗОВЫХ ЛОГИЧЕСКИХ ЭЛЕМЕНТОВ ЦИФРОВЫХ УСТРОЙСТВ НА ЯЗЫКЕ VERILOG

Элементной базой современных цифровых устройств и систем являются цифровые интегральные схемы.

Цифровая интегральная схема (ИС) – это микроэлектронное изделие, изготовленное методами интегральной технологии, заключенное в самостоятельный корпус и выполняющее определенную функцию преобразования дискретных (цифровых) сигналов. Простейшие преобразования над цифровыми сигналами осуществляют цифровые ИС, получившие названия логических элементов (ЛЭ).

Для описания работы цифровых ИС, а следовательно, и устройств, построенных на их основе, используется математический аппарат алгебры логики или булевой алгебры. Возможность применения булевой алгебры для решения задач анализа и синтеза цифровых устройств обусловлена аналогией понятий и категорий этой алгебры и двоичной системы счисления, которая положена в основу представления преобразуемых устройством сигналов.

Основными понятиями булевой алгебры являются понятия логической переменной и логической функции.

Логической переменной называется величина, которая может принимать одно из двух возможных состояний, одно из которых обозначается символом «0», другое – «1». Сами двоичные переменные чаще обозначают символами *x1, x2, …* В силу определения логические переменные можно называть также двоичными переменными.

Логической (булевой) функцией (обычное обозначение – *y*) называется функция двоичных переменных (аргументов),

которая также может принимать одно из двух возможных состояний: «0» или «1». Значение некоторой логической функции n переменных определяется или задается для каждого набора двоичных переменных. Количество возможных различных наборов, которые могут быть составлены из n аргументов, очевидно, равно 2^n. При этом, поскольку сама функция на каждом наборе может принимать значение «0» или «1», то общее число возможных функций от n переменных равно 2^{2^n}.

Таким образом, множество состояний, которые могут принимать как аргументы, так и функции, равно двум. Для этих состояний в булевой алгебре определяются отношение эквивалентности, обозначаемое символом равенства (=) и три операции:

a) Логического сложения (дизъюнкции), обозначаемые, как + или ∨;
b) Логического умножения (конъюнкции), обозначаемые, как ∧ или &;
c) Логического отрицания (инверсии), обозначаемый, как $\overline{X}$ – операция инверсии (X – символ аргумента или функции).

Приведем описание некоторых, имеющих большое значение в цифровой технике, элементарных логических функций и ЛЭ, реализующих эти функции.

3.1. Инвертор

Функция «Отрицание» – это функция одного аргумента (другие названия функции: инверсия, логическая связь НЕ).

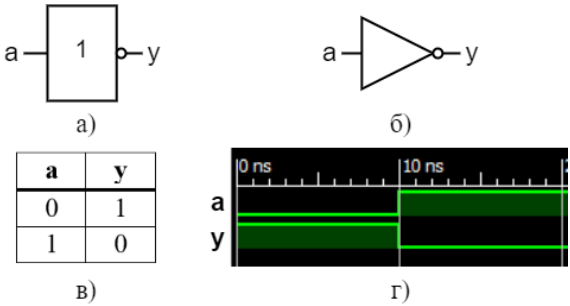
Аналитическая форма задания этой функции:

$$y = \bar{a} \text{ или } y = !\,a$$

где *y* – логическая функция, *a* – аргумент.

Электронный ЛЭ, реализующий функцию «Отрицание» в виде определенных уровней электрических сигналов, называют инвертором или ЛЭ «НЕ» (*англ. NOT*). Условное графическое обозначение (УГО) в стандартах ГОСТ и ANSI, таблица

истинности и временная диаграмма инвертора, полученная в среде XILINX показан на рис. 3.1.



a	y
0	1
1	0

в)

Рисунок 3.1 – УГО по ГОСТ (а), по ANSI (б), таблица истинности (в) и временная диаграмма инвертора полученный в среде XILINX (г)

На языке цифровой техники инверсия означает, что выходной сигнал (y) противоположен входному (a). Листинг кода логического NOT:

```
module not_logic(
  output y,
  input a);
     assign y = !a;
endmodule
```

3.2. Конъюнкция

Функция «Конъюнкция» – это функция двух или большего числа аргументов (другие названия функции: логическое умножение, логическая связь И). Аналитическая форма задания функции двух аргументов:

$$y = a * b \text{ или } y = a \wedge b \text{ или } y = a \,\&\, b$$

где y – логическая функция, a, b – аргументы.

ПРАВИЛО: Функция «Конъюнкция» равна «1» тогда и только тогда, когда все ее аргументы равны «1».

ЛЭ, реализующий функцию «Конъюнкция» называют конъюнктором или ЛЭ «И» (*англ. AND*).

Таблица истинности и временные диаграммы работы двухвходового конъюнктора в среде XILINX показаны на рис. 3.2.

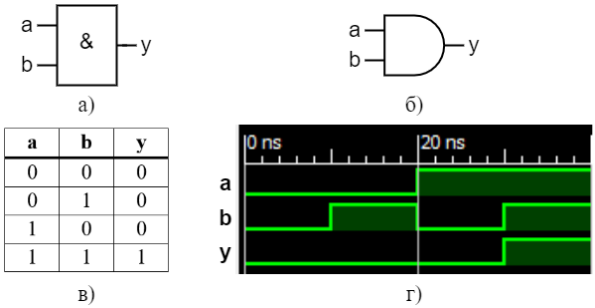


Рисунок 3.2 – УГО по ГОСТ (а), по ANSI (б), таблица истинности (в) и временная диаграмма инвертора полученный в среде XILINX (г)

Листинг кода логического AND:

```
module and_logic(
  output y,
  input a,
  input b);
     assign y = a & b;
endmodule
```

3.3. Дизъюнкция

Функция «Дизъюнкция» – это функция двух или большего числа аргументов (другие названия функции: логическое сложение, логическая связь ИЛИ). Аналитическая форма задания функции двух аргументов:

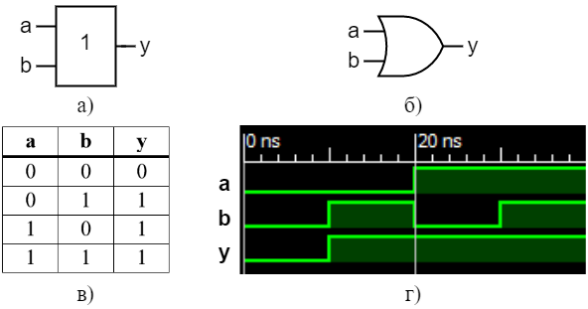
$$y = a + b \text{ или } y = a \vee b \text{ или } y = a \mid b$$

где y – логическая функция, a, b – аргументы.

ПРАВИЛО: Функция «Дизъюнкция» равна «1», если хотя бы один из ее аргументов равен «1».

ЛЭ, реализующий функцию «Дизъюнкция» называют дизъюнктором или ЛЭ «ИЛИ» (*англ. OR*).

УГО по ГОСТ и ANSI, таблица истинности и временная диаграмма в среде XILINX дизъюнктора приведены на рис. 3.3.



a	b	y
0	0	0
0	1	1
1	0	1
1	1	1

в)

Рисунок 3.3 – УГО по ГОСТ (а), по ANSI (б), таблица истинности (в) и временная диаграмма дизъюнктора полученный в среде XILINX (г)

Листинг кода логического OR:

```
module or_logic(
  output y,
  input a,
  input b);
     assign y = a | b;
endmodule
```

3.4. Функция «штрих Шеффера»

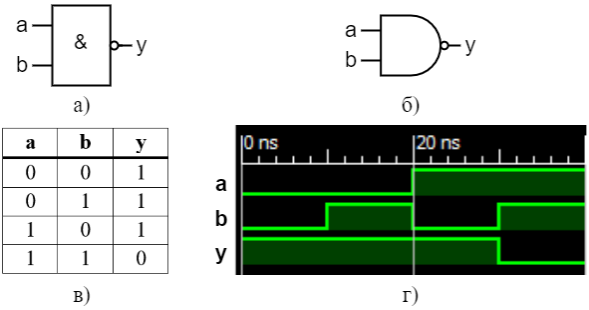
Функция «штрих Шеффера» (другое название функции – логическая связь «И-НЕ») – это функция двух или большего числа аргументов. Аналитическая форма задания функции «И-НЕ» (*англ. NAND*):

$$y = \overline{a * b} \text{ или } y = \overline{a \wedge b}$$

где y – логическая функция, a, b – аргументы.

ПРАВИЛО: Функция «штрих Шеффера» равна «1», если равен «0» хотя бы один из ее аргументов.

УГО по ГОСТ и ANSI, таблица истинности и временная диаграмма в среде XILINX функции «штрих Шеффера» приведены на рис. 3.4.



a	b	y
0	0	1
0	1	1
1	0	1
1	1	0

в)

Рисунок 3.4 – УГО по ГОСТ (а), по ANSI (б), таблица истинности (в) и временная диаграмма «штриха Шеффера» полученный в среде XILINX (г)

Листинг кода логического NAND:

```
module nand_logic(
  output y,
  input a,
  input b);
     assign y = !(a & b);
endmodule
```

3.5. Функция «стрелка Пирса»

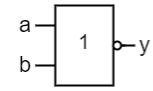
Функция «стрелка Пирса» – это функция двух или большего числа аргументов (другое название функции – логическая связь «ИЛИ-НЕ»). Данная функция является инверсией функции «ИЛИ». Аналитическая форма задания функции двух аргументов функции «ИЛИ-НЕ» (*англ. NOR*):

$$y = \overline{a + b} \text{ или } y = \overline{a \vee b}$$

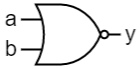
где y – логическая функция, a, b – аргументы.

ПРАВИЛО: Функция «стрелка Пирса» равна «0», если равен «1» хотя бы один из ее аргументов.

УГО по ГОСТ и ANSI, таблица истинности и временная диаграмма в среде XILINX функции «стрелка Пирса» приведены на рис. 3.5.



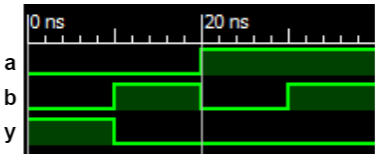
а)



б)

a	b	y
0	0	1
0	1	0
1	0	0
1	1	0

в)



г)

Рисунок 3.5 – УГО по ГОСТ (а), по ANSI (б), таблица истинности (в) и временная диаграмма «стрелки Пирса» полученный в среде XILINX (г)

Листинг кода логического NOR:

```
module nor_logic(
  output y,
  input a,
  input b);
     assign y = !(a | b);
endmodule
```

3.6. Сумма по модулю 2 (М2)

Функция «Сумма по модулю 2 (М2)» – это функция двух или большего числа аргументов. Другое наименование функции «Исключающая ИЛИ» (*англ. XOR*) и аналитическая форма задания (в случае двух аргументов *x1* и *x2*):

$$y = x_1 \oplus x_2$$

Название функции связано с тем, что $x_1 \oplus x_2$ есть арифметическая сумма двоичных чисел *x1* и *x2* в пределах одного

разряда: 0 + 0 = 0; 0 + 1 = 1; 1 + 0 = 1; 1 + 1 = 10. В последнем случае возникает единица переноса в соседний старший разряд, а в разряде самих слагаемых получается ноль. Отсюда широкое применение этого ЛЭ при построении суммирующих устройств.

Для наглядности примера рассмотрим функцию «Исключающая ИЛИ» состоящий из двух аргументов: a, b:

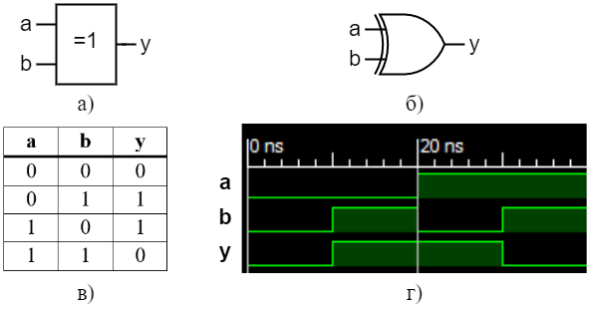
$$y = a \oplus b$$

Имеется очень простой способ проверки таблицы истинности с помощью подсчета количества единиц на входе:

1. Если количество единиц на входе четное на выходе y = 0;
2. Если количество единиц на входе нечетное на выходе y = 1.

ПРЕДУПРЕЖДЕНИЕ: при работе с реальными ЛЭ «Исключающая ИЛИ» может быть обратная логика.

УГО по ГОСТ и ANSI, таблица истинности при прямой логике, временная диаграмма в среде XILINX функции «Исключающая ИЛИ» приведены на рис. 3.6.



a	b	y
0	0	0
0	1	1
1	0	1
1	1	0

Рисунок 3.6 – УГО по ГОСТ (а), ANSI (б) , таблица истинности при прямой логике (в), временная диаграмма в среде XILINX (г) функции «Исключающая ИЛИ»

Листинг кода логического XOR:

```
module xor_logic(
  output y,
  input a,
  input b);
     assign y = a ^ b;
endmodule
```

ГЛАВА 4

ОПИСАНИЕ КОМБИНАЦИОННЫХ УСТРОЙСТВ НА VERILOG

Цифровые схемы разделяются на комбинационные (combinational) и последовательные (sequential). Выходы комбинационных схем зависят только от текущих значений на входах; другими словами, такие схемы комбинирует текущее значение входных сигналов для вычисления значения на выходе. Поведение комбинационной логической схемы можно определить с помощью набора функций вывода. Например, логический элемент – это комбинационная схема.

Комбинационные логические элементы часто группируются в «строительные блоки», используемые для создания сложных систем. Это позволяет абстрагироваться от излишней детализации уровня логических элементов и подчеркнуть функцию «строительного блока». Мультиплексоры и дешифраторы являются часто используемыми блоками.

Схема является комбинационной, если она состоит из соединенных между собой элементов и выполнены следующие условия:

- Каждый элемент схемы сам является комбинационным;
- Каждое соединение схемы является или входом, или подсоединено к
- одному единственному выходу другого элемента схемы;
- Схема не содержит циклических путей: каждый путь в схеме проходит
- через любое соединение не более одного раза.

4.1. Простое назначение сигналов

```
status <= 1;
even <= (p1 & p2) | (p3 & p4);
arith_result <= a + b + c - 1;
```

Последнее выражение может быть изображено в виде следующей схемы (Рис. 4.1):

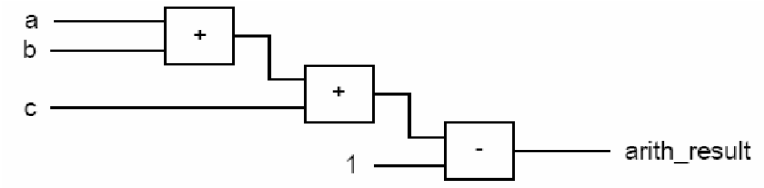


Рисунок 4.1 – Комбинационное устройство

Предположим, что задержки сумматора и блока, вычисляющего разность входных сигналов, одинаковы и равны δ, предположим, что других задержек нет (например, мы не учитываем задержки в проводниках хотя они порой очень существенны). Тогда изменение входного сигнала a отразится на выходном сигнале *arith_result* только через 3δ, ведь сигнал должен пройти через три блока, каждый из которых вносит задержку δ. С другой стороны изменение входного сигнала c отразится на выходном сигнале *arith_result* только через 2δ. Т.е. схема еще и несимметричная. Можно ли оптимизировать схему, чтобы временная задержка стала меньше, и схема стала симметричной?

Воспользуемся коммутативностью и ассоциативностью сложения:

$$a + b + c - 1 = (a + b) + (c - 1)$$

Результат можно посмотреть на Рис. 4.2.

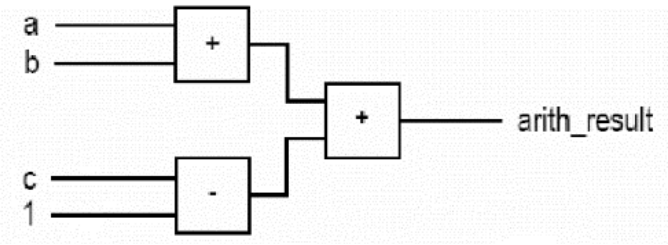


Рисунок 4.2 – Комбинационное устройство

Листинг кода комбинационное устройство на рисунке 4.2:

```
module comb(
  output [4:0]arith_result,
  input [3:0]a,
  input [3:0]b,
  input [3:0]c);

    assign arith_result = (a + b) + (c - 4'd1);

endmodule
```

4.2. Оператор условного назначения сигнала (Тернарный оператор)

Тернарный оператор является оператором использующий три операнда. Синтаксис тернарного оператора:

$$condition\ ?\ expression1 : expression2$$

$condition$ – это место для логического выражения. Если ответом выражение является true, выполняется первое выражения ($expression1$), а если false, то соответственно второе выражения ($expression1$).

Пример 1. Восьмибитовый мультиплексор 4x1. Рассмотрим комбинационную схему, описываемую таблицей истинности (Таблица 4.1).

Таблица 4.1. Таблица работы мультиплексора 4x1

Input **s**	Output **x**
00	a
01	b
10	c
11	d

Листинг кода мультиплексора 4x1:

```
module comb(
 output x,
 input [1:0] s,
 input a,
 input b,
 input c,
 input d);

  assign x = (s == 2'b00) ? a :
             (s == 2'b01) ? b :
            (s == 2'b10) ? c : d;

endmodule
```

Сигнал *x* – это выходной информационный сигнал; *a, b, c, d* – входные информационные сигналы, а s – это сигнал управления, на основании которого осуществляется выбор одного из вариантов.

Пример 2. Бинарный дешифратор. Рассмотрим таблицей истинности представленную в таблице 4.2.

Таблица 4.2. Таблица работы бинарного дешифратора

Input **s**	Output **x**
00	0001
01	0010
10	0100
11	1000

Листинг кода бинарного дешифратора:

```
module bin_decoder(
 output [3:0]x,
 input [1:0] s);

  assign x = (s == 2'b00) ? 4'b0001 :
             (s == 2'b01) ? 4'b0010 :
             (s == 2'b10) ? 4'b0100 : 4'b1000;

endmodule
```

Пример 3. Шифратор приоритета. Рассмотрим таблицей истинности представленную в таблице 4.3.

Таблица 4.3. Таблица работы шифратора приоритета

Input **r**	Output **code**	Output **active**
1---	11	1
01--	10	1
001-	01	1
0001	00	1
0000	00	0

Листинг кода шифратора приоритета:

```
module encoder(
 output [1:0]code,
 output active,
 input [3:0] r);

   assign code = (r[3] == 1'b1) ? 2'b11 :
                 (r[2] == 1'b1) ? 2'b10 :
                 (r[1] == 1'b1) ? 2'b01 : 4'b00;

   assign active = r[3] | r[2] | r[1] | r[0];

endmodule
```

Здесь выходной сигнал *code* определяет, на какой из линий *r[i]* присутствует логическая единица. Причем линия *r[3]* имеет приоритет – если на ней будет 1, то *code* = 11 вне зависимости от других линий.

Поговорим о том, как реализуется оператор условного назначения сигнала в железе. Семантика оператора условного назначения сигнала подразумевает создание приоритетной схемы, т.е. условия, стоящие выше имеют больший приоритет. При синтезе этого оператора реализуются три схемы:

- схема, вычисляющая возможные значения выходного сигнала;
- схема, вычисляющая условия;
- схема определения приоритета.

Чтобы лучше разобраться, что такое схема приоритета введем понятие абстрактного мультиплексора 2x1.

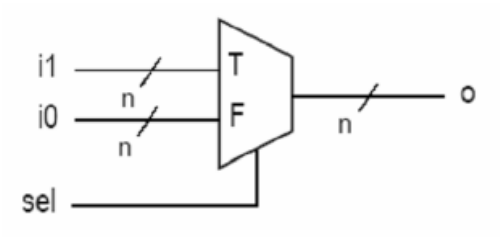


Рисунок 4.3 – Мультиплексор 2x1

Здесь *i1, i2, o – n* - битные шины, а сигнал *sel* однобитный. В зависимости от него на выход *o* идет либо *i1*, либо *i0*.

Когда мы пишем на Verilog выражение:

$$sig = boolean\_expr\ ?\ value\_expr\_1\ :\ value\_expr\_2$$

реализуется схема с мультиплексором, представленная на рисунке 4.4.

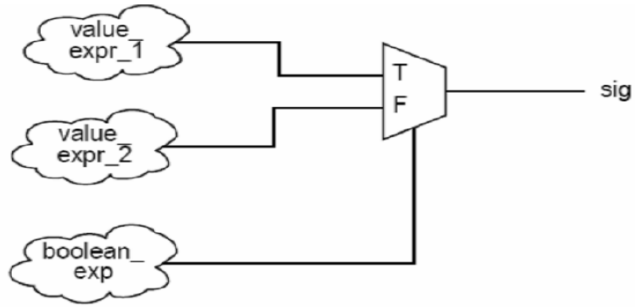


Рисунок 4.4 – Условное назначение сигнала мультиплексора

Каждый следующий уровень ветвления оператора условного назначения сигнала добавляет еще одну ступень в иерархию вместе с еще одним мультиплексором (Рис. 4.5):

sig = boolean_expr_1 ? value_expr_1 :
boolean_expr_2 ? value_expr_2 :
boolean_expr_3 ? value_expr_3 : value_expr_4;

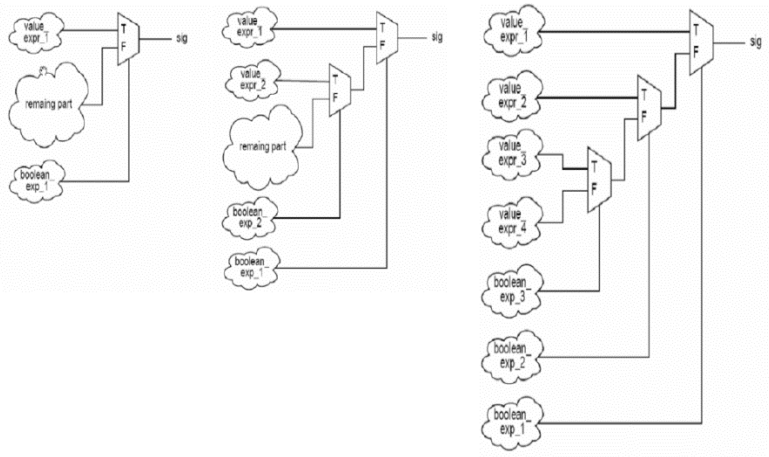


Рисунок 11.5 – Ветвление оператора условного назначения сигнала

При работе с языком Verilog вы всегда должны пытаться мыслить на уровне логических элементов и схем, чтобы получать эффективные реализации, ведь, по сути, Verilog – это не более чем удобный инструмент для описания таких объектов.

ГЛАВА 5

ОПЕРАТОРЫ IF И CASE В VERILOG

В языке Verilog есть средство для описания последовательных алгоритмов с помощью, так называемых, последовательных операторов. Эти операторы должны располагаться внутри специальной конструкции Verilog, называемой *процедурным блоком*. Главной целью введения последовательных операторов в язык Verilog было описание абстрактного поведения цифровых цепей (на более высоком уровне абстракции). Последовательные операторы не всегда имеют однозначное соответствие в железе.

Часто даже они не могут быть синтезированы вовсе. Поэтому для описания схем с помощью процессов и последовательных операторов необходимо придерживаться четких шаблонов, которые однозначно синтезируются. Рассмотрением этих шаблонов мы и займемся. Пусть вас не пугает слово «шаблон»: несмотря на то, что определенные компоненты должны описываться по четкой схеме, у вас останется огромное пространство для творчества при определении взаимодействия этих компонентов.

5.1. Процедурный блок always

Синтаксис использование блока always:

```
always@([список чувствительности ])
begin
  последовательные операторы;
  последовательные операторы;
  ...
end
```

Сам процедурный блок является параллельным оператором. Если в архитектурном теле разместить несколько процессов, то они будут выполняться параллельно. Но внутри каждого процесса находятся операторы, выполняемые последовательно, т.е. один за другим. Список чувствительности – это список запускающих процесс сигналов. Когда один из сигналов, указанных в списке чувствительности, меняет свое значение, процесс запускается, и последовательные операторы выполняются до конца процесса. После этого процесс приостанавливает свою работу до тех пор, пока опять какой-нибудь сигнал из списка чувствительности не изменит свое значение.

Таким образом, процесс может находиться в одном из двух состояний: запущенном и приостановленном. Для комбинационной схемы *все входы* должны быть в списке чувствительности.

Правильно

```
module test(
  output reg y,
  input a, b, c);

  always@(a, b, c)
  begin
     y = a & b | c;
  end
endmodule
```

Неправильно

```
module test(
  output reg y,
  input a, b, c);

  always@(a)
  begin
     y = a & b | c;
  end
endmodule
```

Внутри процесса сигналу можно присвоить значение несколько раз. При этом значение сигнала не изменится до выхода из процесса, а после выхода станет равным последнему присвоенному значению. Получается, что следующие записи просто эквивалентны:

```
always@(a, b, c, d)
  begin
    y = a | c;
    y = a & b;
    y = c & d;
  end
```

```
always@(a, b, c, d)
  begin
    y = c & d;
  end
```

5.2. Оператор if

Оператор if – это один из последовательных операторов, поэтому его можно использовать только внутри оператора процесса. Синтаксис оператора if следующий:

- Первый тип условного оператора без блока else. Оператор выполняется или нет

  if (< $expression$ >) $true\_statement$;

- Второй тип условного оператора с блоком else. Оценивается либо утверждение истинное, либо ложное

  if (< $expression$ >) $true\_statement$;
  $else\ false\_statement$;

- Третьи вид условного оператора, вложенные if-else-if. Выбор из нескольких операторов, но исполняется только первое истинное условие из всех

  if (< $expression1$ >) $true\_statement1$;
  $else\ if$ (< $expression2$ >) $true\_statement2$;
  $else\ if$ (< $expression3$ >) $true\_statement3$;
  . . .
  . . .
  $else\ default\_statement$;

Выражения <$expression$> оценивается, если оно будет истинным (1 или ненулевое значение), то выполнится $true\_statement$. Однако, если это ложное (ноль), то выполняется

$false\_statement$. $true\_statement$ или $false\_statement$ может быть одним оператором или блоком из нескольких выражений. Блок из нескольких выражений должен быть сгруппирован, как правило, с использованием ключевых слов *begin* и *end*.

Рассмотрим его применение на примерах из предыдущего раздела. *Пример 1.* Мультиплексор 4x1.

```
module mux4x1(
  output reg x,
  input [1:0] s,
  input a, b, c, d);

  always@(s, a, b, c, d)
    begin
      if(s == 2'b00)
        x = a;
      else if(s == 2'b01)
        x = b;
      else if(s == 2'b10)
        x = c;
      else
        x = d;
    end
endmodule
```

Пример 2. Бинарный дешифратор.

```
module bin_decoder(
  output reg [3:0]x,
  input [1:0] s);
  always@(s)
    begin
      if(s == 2'b00)
        x = 4'b0001;
      else if(s == 2'b01)
        x = 4'b0010;
      else if(s == 2'b10)
```

```
      x = 4'b0100;
    else
      x = 4'b1000;
  end
endmodule
```

Пример 3. Шифратор приоритета.

```
module encoder(
  output reg [1:0]code,
  output active,
  input [3:0] r);

  always@(r)
    begin
      if(r[3] == 1)
        code = 2'b11;
      else if(r[2] == 1)
        code = 2'b10;
      else if(r[1] == 1)
        code = 2'b01;
      else
        code = 2'b00;
    end
  assign active = r[3] | r[2] | r[1] | r[0];
endmodule
```

Для простых случаев работа оператора if и тернарного оператора эквивалентны.

Отличия:

1. По структуре: внутри if может быть еще один if.
2. По пониманию: легче понять смысл.
3. В одной ветке можно назначить несколько сигналов.

Проблема неполного дерева условий оператора if

В Verilog только одна ветка if обязательна. Остальные ветки (elsif, else) могут быть опущены. Здесь надо быть

аккуратными. Например, рассмотрим компаратор двух чисел, выход которого равен 1, когда они равны.

```
always@(a, b)
  begin
    if(a == b)
      eq = 1;
  end
```

Этот код синтаксически верен, но приводит к неверной реализации. Так как здесь нет ветки else, то, когда числа не равны, не производится никакого действия. Семантика Verilog такова, что в этом случае значение сигнала *eq* не обновляется, и он сохраняет свое значение. Это эквивалентно следующей записи:

```
always@(a, b)
  begin
    if(a == b)
      eq = 1;
    else
      eq = eq;
  end
```

Этот код описывает схему с обратной связью, что приводит к образованию элемента памяти или внутреннему состоянию (обычно так называемых защелок (*latch*)). Очевидно, что мы не это имели в виду. Код необходимо переписать следующим образом:

```
always@(a, b)
  begin
    if(a == b)
      eq = 1;
    else
      eq = 0;
  end
```

Для описания комбинационной цепи необходимо всегда включать ветку else, чтобы избежать образования элемента памяти.

Проблема неполного назначения сигналов в операторе if

В общем случае оператор if имеет несколько веток. Возможно, что сигналу присваивается значение не во всех этих ветках. Хотя синтаксически правильно, это приводит к образованию элемента памяти в схеме (*защелок*). Пример компаратора двух чисел с тремя выходами:

Gt (greater than, a > b), lt (less than, a < b), eq (equal, a=b)

```
always@(a, b)
  begin
    if(a > b)
      gt = 1;
    else if(a == b)
      eq = 1;
    else
      lt = 1;
  end
```

Опять же семантика Verilog подразумевает, что, если сигналу не назначается значение, он его сохраняет. А это приводит к ненужному здесь образованию элемента памяти. Код надо исправить:

Вариант 1

```
always@(a, b)
  begin
    gt = 0;
    eq = 0;
    lt = 0;

    if(a > b)
      gt = 1;
```

Вариант 2

```
always@(a, b)
  begin
    if(a > b)
      begin
        gt = 1;
        eq = 0;
        lt = 0;
      end
```

```
  else if(a == b)
    eq = 1;
  else
    lt = 1;
end
```

```
else if(a == b)
  begin
    gt = 0;
    eq = 1;
    lt = 0;
  end
else
  begin
    gt = 0;
    eq = 0;
    lt = 1;
  end
end
```

Первый вариант кода более красив и реализует прием программирования на Verilog, называемый присвоение значения комбинационному сигналу по умолчанию. Здесь надо вспомнить, что при назначении сигнала в процессе, сигнал принимает последнее присвоенное значение. Рекомендуется использовать именно такой стиль программирования.

5.3. Оператор case

Оператор *case* также является последовательным оператором, и его использование разрешено только внутри процедурного блока. Оператор *case* имеет следующий синтаксис:

```
case (expression)
  alternative1: statement1;
  alternative2: statement2;
  alternative3: statement3;
   . . .
   . . .
  default: default_statement;
endcase
```

Каждый из $statement$ может быть отдельным выражением или блоком нескольких выражений. Блок из нескольких выражений должен быть сгруппирован, как правило, с использованием ключевых слов begin и end. Выражение $expression$ сравнивается с альтернативами в их порядке написания. Если ни одна из альтернатив не совпадает, выполняется $default\_statement$. $default\_statement$ является необязательным.

Рассмотрим применение оператора *case* на примерах из предыдущего раздела. *Пример 1*. Мультиплексор 4x1.

```
module mux4x1(
  output reg x,
  input [1:0] s,
  input a, b, c, d );
  always@(s, a, b, c, d)
    begin
      case(s)
         2'b00 : x = a;
         2'b01 : x = b;
         2'b10 : x = c;
         default : x = d;
      endcase
    end
endmodule
```

Пример 2. Бинарный дешифратор.

```
module bin_decoder(
  output reg [3:0]x,
  input [1:0] s);

  always@(s)
    begin
      case(s)
        2'b00 :  x = 4'b0001;
        2'b01 :  x = 4'b0010;
```

```
        2'b10 :  x = 4'b0100;
        default: x = 4'b1000;
      endcase
    end
endmodule
```

Пример 3. Шифратор приоритета.

```
module encoder(
  output reg [1:0]code,
  output active,
  input [3:0] r);

  always@(r)
    begin
      casex(r)
         4'b1xxx : code = 2'b11;
         4'b01xx : code = 2'b10;
         4'b001x : code = 2'b01;
         default : code = 2'b00;
     endcase
    end
  assign active = r[3] | r[2] | r[1] | r[0];
endmodule
```

Проблема неполного назначения сигналов в операторе CASE

Расмотрим простой пример, представленный ниже:

```
always@(a)
  begin
    case(a)
       3'b000 : high = 1;
       3'b010 : middle = 1;
       default : low <= '1';
    endcase;
  end
```

Подразумевается, что в один момент времени только один из сигналов high, middle и low будет равен 1. Однако при реализации схемы опять возникнет ненужный элемент памяти, так как не все сигналы назначаются за один «пробег» процесса. Решается эта проблема опять же присвоением значений сигналам *по умолчанию.*

5.4 Правила описания комбинационных схем

Перечислим правила, которых вы должны придерживаться при описании комбинационных схем с использованием оператора процесса и последовательных операторов:

1. Все входные сигналы должны быть в списке чувствительности процедурного блока.
2. Дерево условий оператора *if* должно быть полным. Используйте *else.*
3. Выборочные значения в операторе *case* должны быть полными и взаимоисключающими. Используйте *default*.
4. При выходе из процесса всем выходным сигналам необходимо присвоить значение (например, значение по умолчанию в начале процесса).

ГЛАВА 6

ОПИСАНИЕ ПОСЛЕДОВАТЕЛЬНЫХ УСТРОЙСТВ НА VERILOG

Рассмотрим последовательных логических элементов: триггеров, регистров, счетчиков и т. д. Все эти элементы характерны тем, что имеют состояния и способны переходить из одного состояния в другое под действием внешних сигналов. Мы будем работать с элементами, управляемыми тактовым сигналом, причем переходить из одного состояния в другое (или по-другому переключаться) они могут только в узкой окрестности изменения такого сигнала, называемой фронтом сигнала. Различают передний и задние фронты сигнала. Существуют последовательные элементы, переключающиеся по одному фронту (SDR элементы), а есть элементы, работающие по обоим фронтам (DDR элементы). Пример – DDR память.

Зачем нужны эти такты? Такты можно легко считать, измеряя тем самым длительность выполнения некой операции. Эта величина может быть как больше, так и меньше периода тактового сигнала T, но даже если она и меньше T, то мы сможем получить результат ее выполнения только на следующем фронте. В первом приближении такт – это минимальный квант времени в мире синхронных цифровых схем.

Необходимость введения такого кванта времени была обусловлена зависимостью
времени выполнения комбинационной схемы от внешних факторов, прежде всего -температурой. Во времена использования больших интегральных систем (БИС) каждый логический элемент собирался в виде отдельной схемы и имел достаточно большие размеры. Собранная на их основе схема была достаточно громоздкой и, что значительно хуже, требовала кропотливого расчёта времени распространения сигнала по всем

её дорожкам, которое могло изменяться из-за колебаний температуры. Это время определяло, когда результат логической операции будет готов. Для получения стабильного результата необходимо, чтобы входные сигналы не менялись, пока не завершаться все переходные процессы в элементе, т.е. эти данные должны храниться в какой-то памяти. Выходные данные так же должны где-то зафиксироваться, чтобы их можно было передать следующей схеме. При этом время установления выходных данных определяет частоту работы данной схемы. Подобная парадигма позволяет формировать некоторые логические блоки, которые могут работать независимо друг от друга, т. к. выходные триггеры одной схемы могут быть входными у другой.

Другая проблема, которая при этом возникает – если все эти блоки будут работать на разной частоте, то возникает проблема их синхронизации, т. е. данные с одной логической схемы должны передаваться в другую не раньше, чем та готова их принять. Чтобы не усложнять схему и решить эту проблему для всех блоков устанавливают одну частоту, которая определяется наиболее медленным элементом. Именно эта частота и называется тактовой.

6.1. Триггеры

Получается, что нам нужны логические элементы, способные изменять свое состояние (т. е. запоминать результат на выходе комбинационной схемы) только по фронту тактового сигнала.

Разберем, как они работают. Особенностью последовательных логических элементов является зависимость выходного сигнала не только от действующих в настоящий момент на входе логических переменных, но и от тех значений переменных, которые действовали на входе в предыдущие моменты времени. Для выполнения этого условия значения переменных должны быть запомнены логическим устройством. Функцию запоминания значений логических переменных в цифровых системах выполняют так называемые триггеры.

6.1.1. Асинхронный RS триггер

Асинхронный триггер имеет два входа S(set) - установка и R(reset) - сброс и два выхода прямой - Q и инверсный - ~Q. Триггер переходит из текущего состояния X на выходе к состоянию 0, при подаче на вход S нуля и на вход R единицы, а при поступлении на вход S единицы и на вход R нуля триггер переходит к состоянию 1. При нулевых значениях, когда S=R=0 триггер должен сохранять старое значение. Комбинация сигналов S=R=1 не определена (Рис. 1).

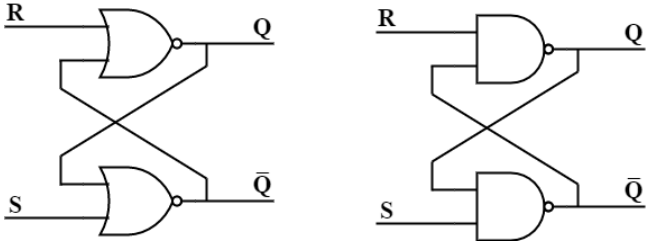


Рисунок 6.1 – RS триггер на базе ЛЭ ИЛИ- НЕ (а) и И-НЕ (б)

Приведенному описанию соответствует схема, приведенная на рисунке. RS триггер можно строить как на элементах «2ИЛИ-НЕ», так и на элементах «2И-НЕ». В первом случае получается триггер с прямыми входами (т. е. с единичным активным уровнем) (Таблица 6.1).

Таблица 6.1. Таблица истинности RS триггера

S	**R**	**Q_{t+1}**
0	0	Q_t
0	1	0
1	0	1
1	1	X

Во втором случае триггер будет управляться нулевым активным уровнем сигналов, и будет называться RS триггером с инверсными входами (Таблица 6.2).

Таблица 6.2. Таблица истинности RS триггера

S	R	Q_{t+1}
0	0	X
0	1	1
1	0	0
1	1	Q_t

6.1.2. RS триггер со статической синхронизацией

К RS триггеру можно добавить сигнал синхронизации C. Когда C=1, то триггер может переключиться, если C=0, то нет. Такой триггер будет называться синхронный RS триггер со статической синхронизацией. Значение слова «статической» вам будет ясен чуть позже. Далее можно добавить асинхронные входы установки и сброса триггера (Рис. 6.2).

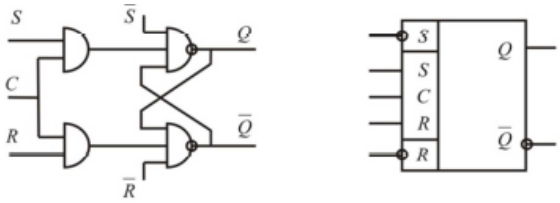


Рисунок 12.2 – RS триггер со статической синхронизацией

6.1.3. D триггер

Вместо двух входов S и R у триггера может быть только один вход D (Data)(Рис. 6.3).

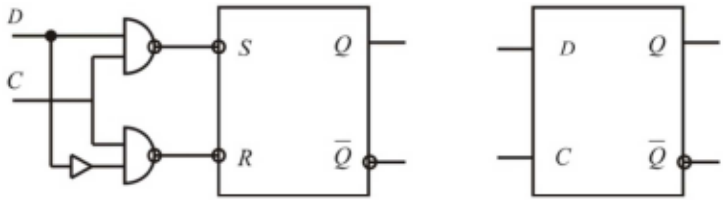


Рисунок 6.3 – D триггер

6.1.4. Триггеры со статическим управлением (защёлки)

Ещё один тип памяти – защёлка (latch). По принципу работы она похожа на D триггер, с той лишь разницей, что в ней используется статический вход синхронизации, который в литературе чаще называется load. На Verilog он реализуется следующим образом:

```
module latch(
  output Q,
  input D,
  input load);

  reg q_tmp = 0;

  always@(load, D)
     begin
       if(load)
         q_tmp <= D;
     end
    assign Q = q_tmp;
endmodule
```

На рисунке 6.4 представлена временная диаграмма работы выше представленного кода защёлки.

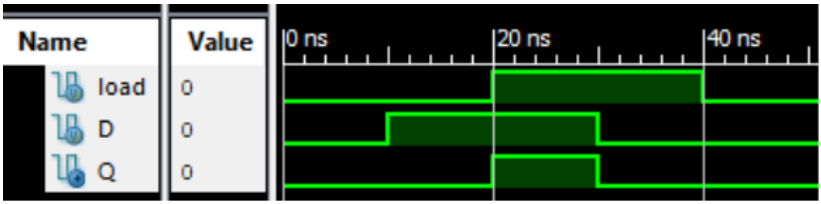


Рисунок 6.4 – Временная диаграмма триггеры со статическим управлением

Необходимо очень аккуратно использовать защёлки в своих дизайнах, а еще лучше – стараться избегать этого вовсе.

6.1.5. D триггер с динамической синхронизацией

Рассмотренные выше триггеры имеют статическую синхронизацию. Это означает, что они управляются уровнем сигнала синхронизации, т.е., этот сигнал выступает в виде разрешения на изменения. Промежуток времени, когда C=1 может быть довольно значителен (половина такта). В течение этого времени триггер просто передает на выход значение входного сигнала. Если вход триггера изменится, то изменится и выход. Обычно в цифровых системах значение триггера меняется только 1 раз за период тактового сигнала, и поэтому вход триггера должен оставаться неизменным в течение всего времени пока C=1, а это накладывает сильные ограничения на входной сигнал и сильно усложняет проектирование схем.

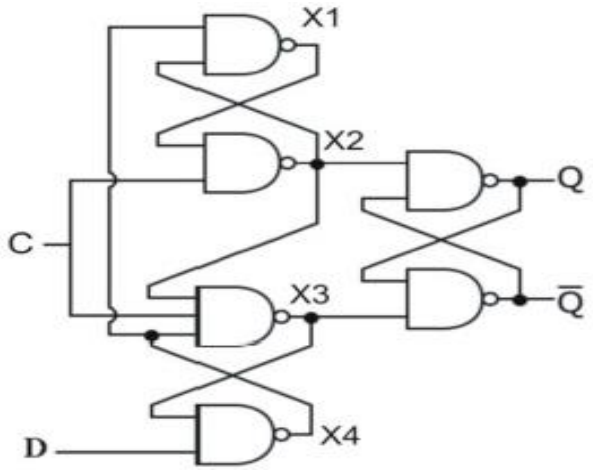


Рисунок 12.5 – D триггер с динамической синхронизацией

Ситуацию можно сильно упростить, если сделать триггер нечувствительным к изменению входного сигнала даже во время действия высокого уровня синхросигнала и заставить его переключаться в течение намного меньшего промежутка времени вокруг так называемого фронта синхроимпульса. Такие триггеры есть, и называются они триггерами с динамической

синхронизацией. Говорят, что они чувствительны не к уровню (как триггеры со статической синхронизацией), а к изменению или фронту синхросигнала. Если изменение состояния триггера происходит по переднему фронту синхросигнала, т. е. по изменению сигнала с 0 на 1, то говорят, что он снабжен прямым динамическим входом. Если же изменение состояния триггера происходит по заднему фронту, т. е. по изменению с 1 на 0, то считают, что он имеет инверсный динамический вход.

Среди триггеров с динамическим управлением широкое распространение получила так называемая схема трех триггеров. Идея построения такой структуры состоит в запоминании сигналов, действовавших на информационных входах в момент изменения сигнала на входе синхронизации. Данная идея реализуется подачей информационных сигналов на основную ячейку памяти (асинхронный RS триггер) не через вспомогательную комбинационную схему, а с использованием дополнительных элементов памяти. Так как асинхронный триггер содержит два информационных входа, то для реализации описанной идеи необходимо два вспомогательных RS - триггеров. Отсюда и название – схема трех триггеров.

Рассмотрим такой триггер. Запишем логические выражения для сигналов x_1, x_2, x_3, x_4.

$$x_1 = \overline{x_2 x_4} = \overline{x_2} + \overline{x_4}$$

$$x_2 = \overline{x_1 C} = \overline{x_1} + \bar{C}$$

$$x_3 = \overline{x_2 x_4 C} = \overline{x_2} + \overline{x_4} + \bar{C}$$

$$x_4 = \overline{x_3 D} = \overline{x_3} + \bar{D}$$

Решим ее относительно x_2 и x_3:

$$\begin{matrix} x_2 = x_2(\overline{x_3} + \bar{D}) + \bar{C} \\ x_3 = x_3 D + \overline{x_2} + \bar{C} \end{matrix} \quad => \quad \begin{matrix} x_{2n+1} = x_{2n}(\overline{x_{3n}} + \bar{D}) + \bar{C} \\ x_{3n+1} = x_{3n} D + \overline{x_{2n}} + \bar{C} \end{matrix} \qquad (1)$$

Рассмотрим возможные значения параметров в последней системе.
а) Пусть C = 0, тогда:

$$x_{2n+1} = 1$$
$$x_{3n+1} = 1 \quad (2)$$

Следовательно, в этом случае основной RS – триггер находится в режиме хранения информации, т. к. на его входах присутствуют пассивные логические сигналы.
б) Пусть поступлении C = 1, подставив (2) в (1) получим:

$$x_{2n+1} = 1(0 + \overline{D}) + 0 = \overline{D}$$
$$x_{3n+1} = D + 0 + 0 = D \quad (3)$$

Подставим (3) в (1) и убедимся, что это состояние устойчивое.

$$x_{2n+1} = \overline{D}(D + \overline{D}) + 0 = \overline{D}$$
$$x_{3n+1} = D + D + 0 = D \quad (4)$$

в) Пусть C = 1, D = 1. Тогда в соответствии с (4):

$$x_{2n+1} = 0$$
$$x_{3n+1} = 1 \quad (5)$$

Триггер установился и стал устойчивым. Проверим это. Пусть в следующий момент времени при действующем синхросигнале C = 1 вход D изменил состояние с «1» на «0». Подставим в (1) значение D = 0 в (5) получим опять:

$$x_{2n+1} = 0$$
$$x_{3n+1} = 1$$

Мы увидим, что после прихода переднего фронта синхросигнала (C = 1) триггер переходит в устойчивое состояние хранения значения, действовавшего на тот момент на информационном входе D. Последующее изменение сигнала D (1 => 0) не привело к изменению состояния триггера.

Теперь для того, чтобы триггер смог изменить свое значение, необходимо подать C=0. Значения X2 и X3 станут равными 1. Триггер будет хранить информацию. Затем после изменения C=1, триггер опять запомнит состояние входа D на момент фронта синхросигнала.

Рассмотренный D триггер с динамической синхронизацией сохраняет значение логической переменной, действующей на информационном входе D в момент каждого
переднего фронта синхросигнала. Иногда требуется, чтобы триггер изменял свое значение не по каждому такому фронту, а только в определенные промежутки времени. Для этого требуется введение дополнительного управляющего сигнала, называемого CE (Clock Enable). Он может быть реализовываться через элемент «И» входов CE и C. Отсюда и название: CE = Clock Enable = Разрешение Тактов. Однако это приводит к внесению дополнительной задержки на синхроный вход, и такой подход не применяется в ПЛИС. Второй подход заключается в использовании мультиплексора в цепи информационного входа D.

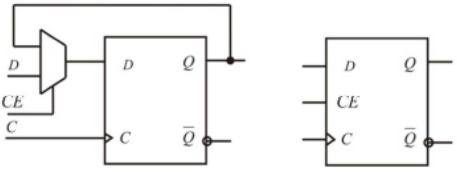


Рисунок 12.6 – Использовании мультиплексора в цепи информационного входа D

Последний рассмотренный триггер является базовым триггером для ПЛИС. В дальнейшем, когда мы будем говорить «D триггер», мы будем иметь в виду D триггер с динамической синхронизацией и входом разрешения работы CE.

6.1.6. Счетный T триггер

Последний триггер, который мы рассмотрим, называется счетный T триггер.

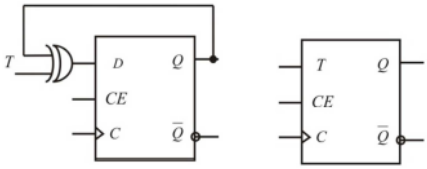


Рисунок 6.7 – T триггер

По определению если в момент прихода синхроимпульса на входе T = 1, то триггер изменяет свое состояние на противоположное. Если T = 0 в момент прихода фронта тактового сигнала, триггер сохраняет свое значение.

6.2. Реализация триггеров на Verilog

Триггеры внутри ПЛИС реализуются не за счёт логических элементов, как было описано выше, они уже реализованы внутри кристалла. В прошивке указывается то, как они должны быть сконфигурены (вид сброса, наличие сигнала разрешения на запись и т. п.). Чтобы трассировщик знал какую именно конфигурацию выбрать необходимо определённым образом описывать поведение сигналов. При описании процессов в списке чувствительности должен находиться только синхросигнал, т. к. только его изменение должно вызывать процесс. Внутри процесса все сигналы, значения которых сохраняются на триггере, должны находиться внутри следующей конструкции:

```
always@(posedge clk)
  begin
    ...
  end
```

где, сигнал *clk* – синхроимпульс.

Приведённый выше код предназначен для случая синхронизации по фронту, если же необходима синхронизация по спаду, то она должна иметь вид:

```
always@(negedge clk)
  begin
    ...
  end
```

6.2.1. Реализация D триггера

Рассмотрим код, реализующий простой D триггер. На входе у него синхроимпульс clk и вход данных D, выход – Q.

```
module D_flip_flop(
  output reg q,
  input d,
  input clk);

  always@(posedge clk)
    begin
       q <= d;
    end
endmodule
```

На рисунке 6.8 представлена временная диаграмма работы D-триггера.

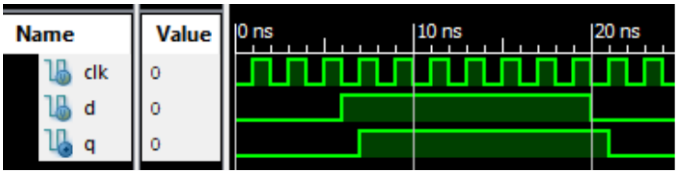


Рисунок 6. 8 – Временная диаграмма D триггера

6.2.2. Реализация D триггера с сигналом разрешения записи

Модернизируем предыдущий пример, добавив сигнал CE – сигнал разрешения.

```
module D_flip_flop(
  output reg q,
  input d,
  input ce,
  input clk);

  always@(posedge clk)
    begin
      if(ce == 1)
        q <= d;
    end
endmodule
```

На рисунке 6.9 представлена временная диаграмма работы D-триггера с разрешающим входом.

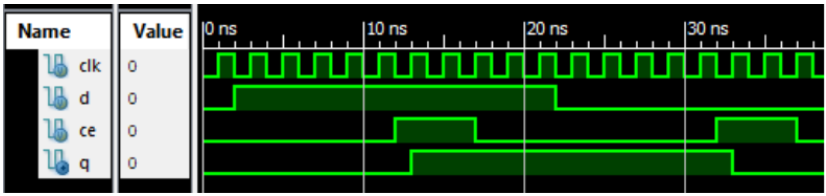


Рисунок 12.9 – Временная диаграмма D триггера с сигналом разрешения записи

6.2.3. Реализация D триггера с синхронным сбросом

Рассмотрим случай сигнала синхронного сброса reset у обычного D триггера.

```
module D_flip_flop(
  output reg q,
  input d,
  input reset,
  input clk);

  always@(posedge clk)
    begin
      if(reset == 1)
        q <= 0;
      else
        q <= d;
    end
endmodule
```

На рисунке 6.10 представлена временная диаграмма работы D-триггера с синхронным сбросом.

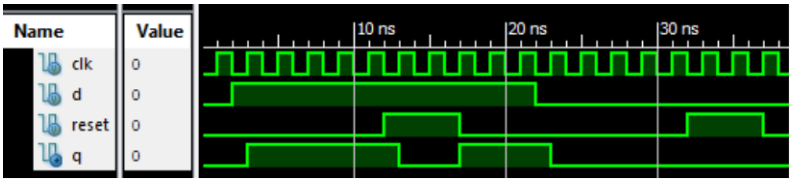


Рисунок 6.10 – Временная диаграмма D триггера с синхронным сбросом

6.2.4. Реализация D триггера с асинхронным сбросом

Рассмотрим случай сигнала асинхронного сброса reset у обычного D триггера. Для этого сам процесс должен реагировать на изменения сигнала reset, независимо от состояния линии clk, поэтому необходимо включить сигнал reset в список чувствительности. Изменения выходного сигнала Q, связанного со сбросом, должно происходить вне блока.

```
module D_flip_flop(
  output reg q,
  input d,
  input reset,
  input clk);

  always@(posedge clk or reset)
    begin
      if(reset == 1)
        q <= 0;
      else
        q <= d;
    end
endmodule
```

На рисунке 6.11 представлена временная диаграмма работы D-триггера с асинхронным сбросом.

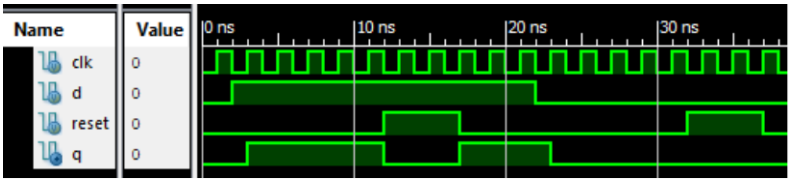


Рисунок 12.11 – Временная диаграмма D триггера с асинхронным сбросом

6.2.5. Реализация T триггера

Рассмотрим простой счётный триггер. У него есть только один входной сигнал – синхроимпульс (clk) и один выходной – данные (Q).

```
module T_flip_flop(
  output q,
  input clk);
  reg q_tmp = 0;
```

```
  always@(posedge clk)
    begin
       q_tmp = !q_tmp;
    end
    assign q = q_tmp;
endmodule
```

На рисунке 6.12 представлена временная диаграмма работы Т-триггера.

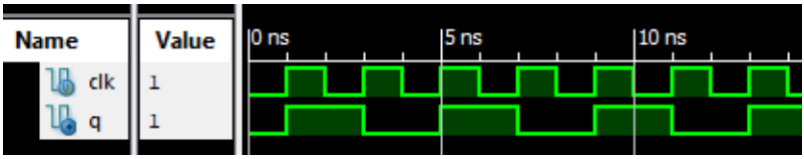


Рисунок 6.12 – Временная диаграмма счётного Т триггера

6.3. Примеры использования триггеров

В качества примера использования рассмотрим детектор фронта, т. е. устройство, которое выдаёт импульс, как только происходит перепад входного сигнала из 0 в 1. Входные сигнала элемента: clk – синхроимпульс, d – внешний сигнал, d_knok – импульс, который сигнализирует о появлении перепада d из 0 в 1.

```
module edgeDetect(
output reg d_knok,
input d,
input clk);
reg d_last = 0;
always@(posedge clk)
  begin
     d_last <= d;
     d_knok <= d & (!d_last);
  end
endmodule
```

На рисунке 6.13 представлена временная диаграмма работы детектора фронта.

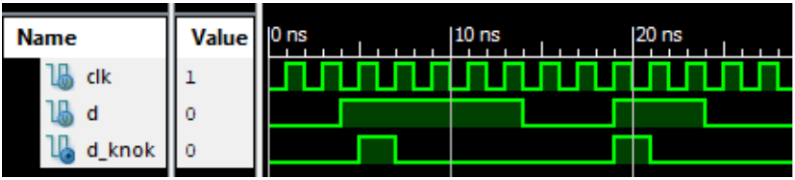


Рисунок 6.13 – Временная диаграмма детектора фронта

ГЛАВА 7

ОПИСАНИЕ СЧЕТНЫХ УСТРОЙСТВ НА VERILOG

7.1. Сложные последовательные элементы

Переходим теперь к соединению триггеров между собой. Один триггер может представлять 1 битовое двоичное число. Для представления n-битного числа требуется n битов. На практике мы часто работаем с 8, 16 и 32 битными числами. Если соединить несколько триггеров вместе, то получится регистр. Регистром называется последовательное устройство, предназначенное для записи, хранения и (или) сдвига информации, представленной в виде многоразрядного двоичного кода. Любой N разрядный регистр состоит из N однотипных ячеек – разрядных схем. При этом каждая разрядная схема, как любое последовательное устройство, состоит из триггера (элемента памяти) и некоторой комбинационной схемы, преобразующей входные воздействия и состояние триггера в выходные сигналы регистра. Ниже на рисунке 7.1 приведена схема регистра (слева) и его условное обозначение (справа).

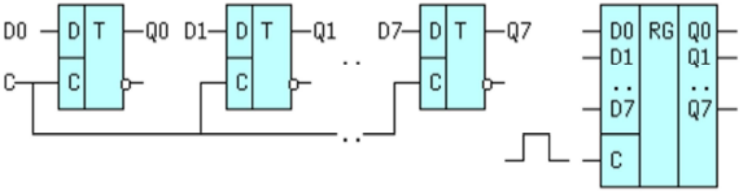


Рисунок 7.1 – Схема регистра

7.2. Сдвиговые регистры

Сдвиговые регистры выполняются на основе триггеров с динамическим синхровходом. Для этого триггеры соединяются последовательно, выходы предыдущих триггеров соединяются со входами последующих. Ниже на рисунке 7.2 приведена схема такого элемента, построенного на D-триггерах (слева) и его условное обозначение (справа):

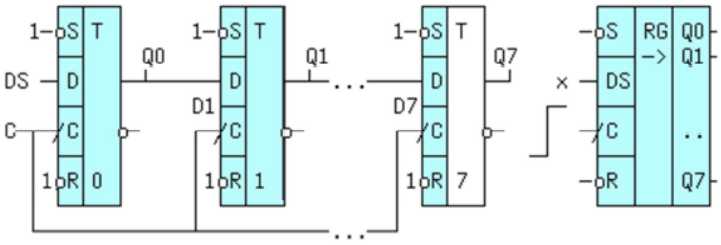


Рисунок 7.2 – Схема сдвигового регистра

С приходом очередного положительного фронта синхроимпульса С длительностью t 0,1, сигнал со входа i-го триггера через время tзд.р. окажется на его выходе и поступит на вход следующего (i+1)-го триггера. Однако на его выход эта информация не перепишется, т. к. длительность активного фронта t0,1 меньше tзд.р.. На этом процесс сдвига данных на один разряд закончится до прихода следующего положительного фронта тактового сигнала. Основные назначения сдвиговых регистров – задержка данных, преобразование из параллельного кода в последовательный и обратно.

7.2.1. Реализация сдвигового регистра

Рассмотрим 8-ми разрядный последовательный сдвиговый регистр. Вход данных обозначим как D, синхроимпульс обозначается как раньше clk, а выходной сигнал – Q.

```
module SReg8(
  output q,
  input d,
  input clk);
  reg [7:0] s = 8'b0;
  always@(posedge clk)
    begin
       s[7:1] <= s[6:0];
       s[0] <= d;
    end
  assign q = s[7];
endmodule
```

В приведённом примере используется шина S для хранения данных сдвигового регистра. Каждая из линий S представляется в виде отдельного триггера, процесс переноса описывается строчкой: s[7:1] <= s[6:0]; а процесс загрузки новых данных: s[0] <= d;. Данные выталкиваются наружу из старшего триггера: q = s[7]. Временная диаграмма работы сдвигового регистра представлена на рисунке 7.3.

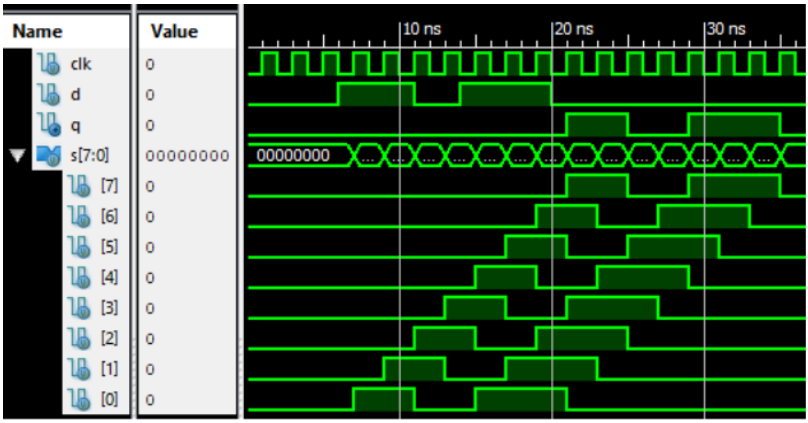


Рисунок 7.3 – Временная диаграмма сдвигового регистра

7.2.2. Реализация сдвигового регистра с параллельной загрузкой

На практике полезным оказывается сдвиговый регистр, в который данные во все триггеры можно загрузить за один такт. К предыдущему примеру добавятся дополнительные сигналы: Load (флаг загрузки данных) и LD (загружаемые данные).

```
module SReg8_ParLoad(
  output q,
  input d,
  input [7:0]ld,
  input load,
  input clk);

  reg [7:0] s = 8'b0;

  always@(posedge clk)
    begin
      if(load == 1)
          s <= ld;
      else
        begin
          s[7:1] <= s[6:0];
          s[0] <= d;
        end
    end
    assign q = s[7];

endmodule
```

На рисунке 7.4 представлена временная диаграмма, показывающая работу сдвигового регистра с параллельной загрузкой.

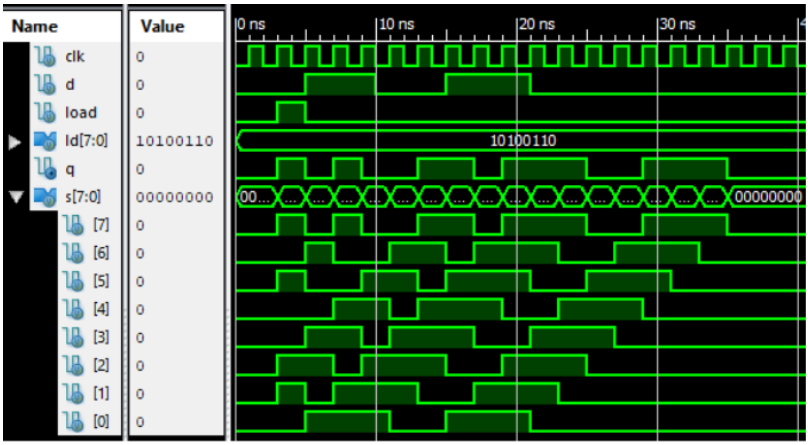


Рисунок 7.4 – Временная диаграмма сдвигового регистра с параллельной загрузкой

7.2.3. Реализация сдвигового регистра с параллельным выходом

Рассмотрим пример реализации сдвигового регистра с параллельным выходом по аналогии с простым сдвиговым регистром, который приводился раньше. Его отличие заключается в том, что выход Q – шина.

```
module SReg8(
  output [7:0]q,
  input d,
  input clk);
  reg [7:0] s = 8'b0;
  always@(posedge clk)
    begin
       s[7:1] <= s[6:0];
       s[0] <= d;
    end
    assign q = s;
endmodule
```

На рисунке 7.5 представлена временная диаграмма, показывающая работу сдвигового регистра с параллельным выходом.

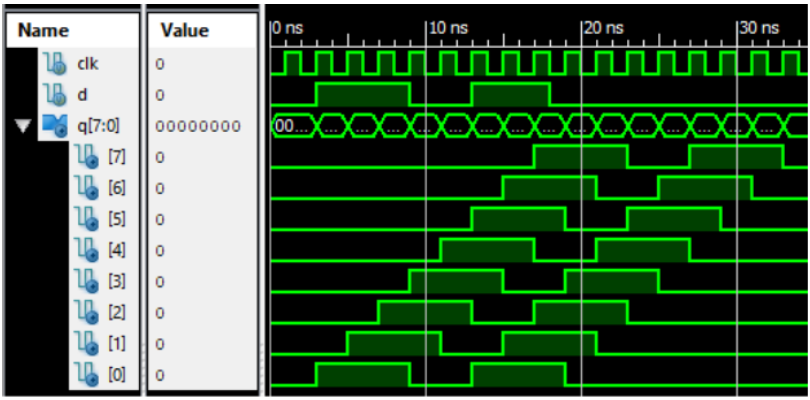


Рисунок 7.5 – Временная диаграмма сдвигового регистра с параллельным выходом

Возможна реализация различных сдвиговых регистров: с параллельной загрузкой, реверсивные регистры (направление сдвига определяется состоянием одного из входов), частично параллельный регистр (не все триггеры выставляют данные на выходную шину) и т.п. На практике все эти виды регистров полезны для перевода параллельного кода (шины) в последовательный код (линию) и обратно, что используется при передаче данных в таких протоколах как USB, RS232, Ethernet и т.п.

7.3. Счетчики

Счётчик – последовательная схема, которая преобразует поступающие на вход импульсы в параллельный код, эквивалентный количеству пришедших импульсов. Счётчики могут считать импульсы в различных системах отсчёта (двоичный, двоично-десятичный), иметь сигнал разрешения на счёт, возможность параллельной загрузки состояния счётчика,

выдача сигнала о переполнении, либо о достижении какого-то значения. Ниже приведена схема самой простой реализации четырёхбитного счётчика (Рис. 7.6).

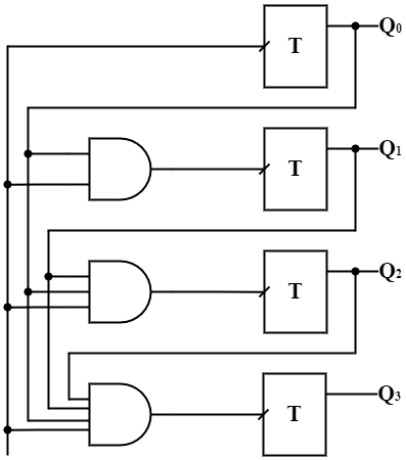


Рисунок 7.6 – Счетчик

Будем считать, что нумерация триггеров идёт от верхнего (0-й) до нижнего (3-й). В начальном состоянии все регистры находятся в 0-м состоянии, при приходе синхроимпульса своё состояние на 1 меняет только 0-й T триггер, так как на остальные поступают только 0. На следующем такте он же меняется обратно в 0, а 1-й – в высокое состояние. На следующем такте меняет своё состояние на 1 только 0-й триггер. 2-й триггер не может этого сделать, т. к. зависит от предыдущего состояния 0-го триггера. И так далее до тех пор, пока все они не будут в высоком состоянии. Следующим их состоянием с приходом синхроимпульсом будет 0, так называемое переполнение счётчика.

7.3.1. Реализация счётчика

Рассмотрим реализацию 4-битного счётчика, который считает каждый импульс. Входной сигнал - clk (синхроимпульс), выходной сигнал – Q.

```
module counter4bit(
  output [3:0]q,
  input clk);

  reg [3:0] counter = 4'b0;

  always@(posedge clk)
    begin
       counter <= counter + 1;
    end
     assign q = counter;

endmodule
```

На рисунке 7.7 представлена временная диаграмма работы счетчика.

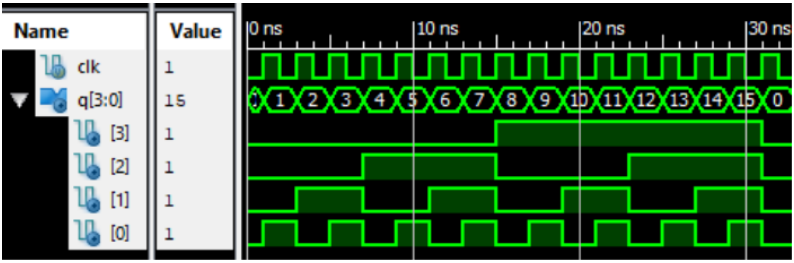


Рисунок 7.7 – Временная диаграмма 4-битного счётчика

7.3.2. Реализация реверсивного счётчика с параллельной загрузкой

Помимо прямого счёта, т. е. инкрементального, бывают и реверсивные счётчики, которые считают в обратном направлении. Это может оказаться полезным, когда необходимо отсчитать какое-то количество тактов и выдать сигнал об окончании счёта.

Рассмотрим реверсивный счётчик, который выдаёт сигнал об окончании счёта. В качестве входов у него будет синхроимпульс (clk), флаг загрузки данных (load) и шину данных (DL). Выходной сигнал CR – окончание счёта.

```
module RevCounter4(
  output cr,
  input [3:0]dl,
  input load,
  input clk);

  reg [3:0] counter = 4'b1111;

  always@(posedge clk)
    begin
      if(load == 1)
        counter <= dl;
     else
       counter <= counter - 1;
    end
    assign cr = (counter == 0) ? 1 : 0;

endmodule
```

На рисунке 7.8 представлена временная диаграмма работы реверсивного счётчика с параллельной загрузкой.

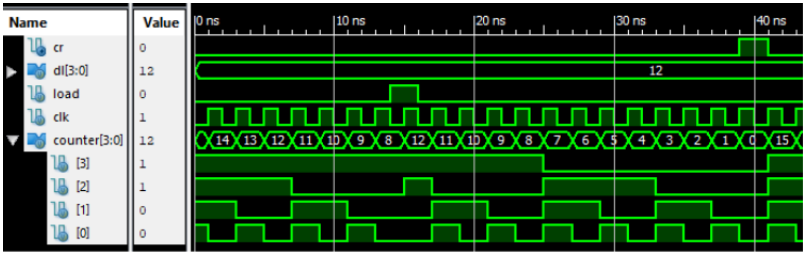


Рисунок 7.8 – Временная диаграмма реверсивного счётчика с параллельной загрузкой

ГЛАВА 8

БАЗОВЫЕ ЛОГИЧЕСКИЕ ВЕНТИЛИ И СТРУКТУРЫ

Транзистор является основным элементом всех цифровых логических компонентов. Однако, работая над проектом, состоящим из нескольких миллионов транзисторов, разработчик не может мыслить на уровне отдельных транзисторов. Поэтому транзисторы объединяются в более абстрактные структуры, называемые вентилями.

8.1. Состояния логического сигнала

8.1.1. Система логических значений

Полной системой для представления логических величин является четырехзначная система (Таблица 8.1).

Таблица 8.1. Четырехзначная логическая система

Значения	**Описание**
0	Установка «0» или подтягивание к «0»
1	Установка «1» или подтягивание к «1»
Z	«Плавающее» состояние или высокий импеданс
X	Неопределенное состояние

Если сигнал подтягивается к низкому уровню напряжения (Gnd), то он является логическим «0», если же подтянут к высокому уровню напряжения (Vdd), то он является логической «1».

В логическом моделировании линия, которая не управляется посредством подтягивания к верхнему (pull - up) или нижнему (pull - down) уровню напряжения, принимает значение Z.

Состояние линии или проводного соединения, которое управляется как высоким, так и низким значениями сигналов, в результатах моделирования представляется значением X.

8.1.2. Третье состояние логического сигнала

По-другому еще называется Z - состояние. Элементы, имеющие третье Z – состояние используются там, где необходима передача информации по одной линии от нескольких источников к одному или нескольким приемникам. В таком случае только один источник сигнала (драйвер) должен быть в активном состоянии. Все остальные драйверы должны быть переведены в третье состояние, характеризующееся тем, что выходное сопротивление элемента становится практически бесконечным. В третьем состоянии выход схемы не «управляет» своим выходным потенциалом. Вместо этого потенциал на выходе элемента в третьем состоянии будет равен выходному потенциалу другого элемента, подключенного к этой электрической цепи (Рис. 8.1).

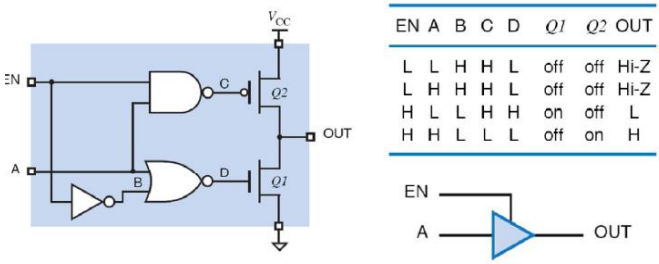


Рисунок 8.1 – Схема тристабильного элемента

Достигается этот эффект за счет следующего. Тристабильный элемент имеет вход EN, который в буквальном смысле разрешает передачу значения входного сигнала A на линию OUT. Если EN = 1, то в зависимости от сигнала A открыт транзистор Q1 или Q2, и на выходе OUT появляется потенциал равный A. Если же EN = 0, то вне зависимости от сигнала A оба транзистора Q1 и Q2 закрыты и ток через них не течет. Таким

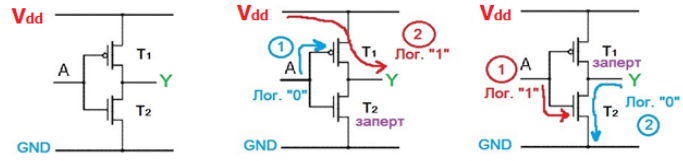
образом, получается, что выходная линия OUT оказывается отрезанной и от потенциала 0 и от потенциала 1. Говорят, что линия OUT оказывается «болтающейся в воздухе».

8.2. Представление логических функций

Логическая функция может представляться различными способами. Простейшее табличное представление функции, в котором перечисляется все комбинации входных и выходных значений, называется таблицей истинности. Алгебраический формат представления более компактен и позволяет манипулировать функциями и объединять их. Очень часто логические функции представляют в графической форме.

8.2.1. КМОП – инвертор (вентиль НЕ)

Инвертор или логическая функция НЕ (NOT), является логическим вентилем, выход которого является дополнением его входа. Структура транзисторного уровня, работа данного вентиля и таблица истинности показаны на рис. 8.2.



№	A	Транзистор T1	Транзистор T2	Y
1	0	Открыт	Заперт	1
2	1	Заперт	Открыт	0

г)

Рисунок 8.2 – КМОП – вентиль НЕ: обозначение (а), работа вентиля при значении входа равной «0» (б), работа вентиля при значении входа равной «1» (в) и таблица истинности вентиля НЕ (г)

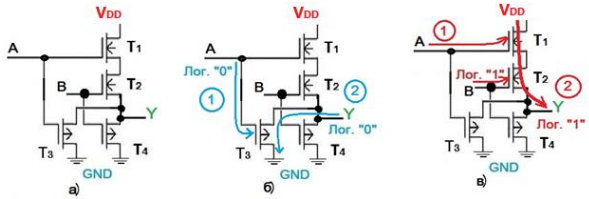
Вход A = «0». На затвор транзисторов подается логический «0», открывается верхний pMOS – транзистор T1 и положительное напряжение V_{dd} через сток – исток подается на

выход логическую «1». Нижний nMOS- транзистор Т2 заперт (рис. 8.2, б).

Вход А = «1». На затвор транзисторов подается логическая «1», открывается нижний nMOS-транзистор Т2 и на выходе появляется логический «0». Выходной сигнал через сток – исток подтягивается к низкому уровню напряжения (Gnd). Верхний pMOS- транзистор Т1 заперт (рис. 8.2 в).

8.2.2. КМОП – вентиль И

КМОП – вентиль И (AND) состоит из двух последовательно соединенных nMOS- транзисторов (Т1 и Т2) для подтягивается выхода вентиля к высокому уровню напряжения (Vdd) и двух параллельно соединенных pMOS- транзисторов (Т3 и Т4) для подтягивания выхода вентиля к низкому уровню напряжения (Gnd), (рис. 8.3).



№	A	B	Транзистор T_1	Транзистор T_2	Транзистор T_3	Транзистор T_4	Y
1	0	0	Заперт	Заперт	Открыт	Открыт	0
2	0	1	Заперт	Открыт	Открыт	Заперт	0
3	1	0	Открыт	Заперт	Заперт	Открыт	0
4	1	1	Открыт	Открыт	Заперт	Заперт	1

г)

Рисунок 8.3 – КМОП – вентиль И: обозначение (а), работа вентиля при значении одного из входов равной «0» (б), работа вентиля при значении обеих входов равной «1» (в) и таблица истинности вентиля И (г)

КМОП – вентиль И работает по следующим правилам:

a) Любой логический «0» на входе вентиля, на выходе дает логический «0» (см. рис. 8.3 б);
b) На выходе вентиля И будет логическая «1», если все входы равны логической «1» (см. рис. 8.3 в).

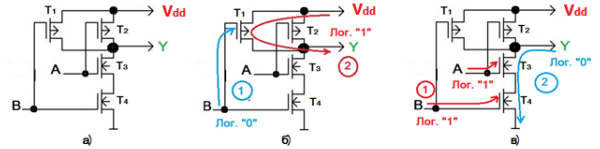
Рассмотрим примеры:

Вход A = «0», позиции 1 и 2 таблицы истинности. На затвор транзисторов T1 и T3 подается логический «0», открывается первый pMOS-транзистор T3 и выходной сигнал через сток – исток транзистор T3 подтягивается к низкому уровню напряжения (Gnd) (Рис. 8.3 б). Транзистор T1 заперт.

Входы A = B = «1», позиция 4 таблицы истинности. На затвор всех транзисторов подается логическая «1», открываются nMOS- транзисторы T1 и T2. Положительное напряжение Vdd через стоки – истоки открытых транзисторов T1 и T2 подает на выход логическую «1» (Рис. 8.3 в). Нижние pMOS-транзисторы T3 и T4 заперты.

8.2.3. КМОП – вентиль И-НЕ

КМОП – вентиль И-НЕ состоит из двух последовательно соединенных nMOS- транзисторов (T3 и T4) для подтягивается выхода вентиля к низкому уровню напряжения (Gnd), и двух параллельно соединенных pMOS- транзисторов (T1 и T2) для подтягивания выхода вентиля к высокому уровню напряжения (Vdd) (рис. 8.4).



№	A	B	Транзистор T_1	Транзистор T_2	Транзистор T_3	Транзистор T_4	Y
1	0	0	Открыт	Открыт	Заперт	Заперт	1
2	0	1	Заперт	Открыт	Заперт	Открыт	1
3	1	0	Открыт	Заперт	Открыт	Заперт	1
4	1	1	Заперт	Заперт	Открыт	Открыт	0

г)

Рисунок 8.4 – КМОП – вентиль И-НЕ: обозначение (а), работа вентиля при значении одного из входов равной «0» (б), работа вентиля при значении обеих входов равной «1» (в) и таблица истинности вентиля И- НЕ (г)

КМОП – вентиль И- НЕ работает по следующим правилам:

а) Любой логический «0» на входе вентиля, на выходе дает логическую «1» (см. рис. 8.4 б);

b) На выходе вентиля И- НЕ будет логический «0», если все входы равны логической «1» (см. рис. 8.4 в).

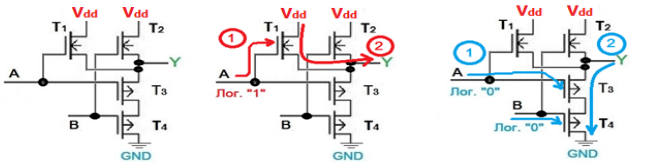
Рассмотрим примеры:

Вход В = «0», позиции 1 и 3 таблицы истинности. На затвор транзисторов Т1 и Т4 подается логический «0», открывается первый pMOS – транзистор Т1 и выходной сигнал через сток – исток транзистора Т1 подает на выход логическую «1». Транзистор Т4 заперт (см. рис. 8.4 б);

Входы А = В = «1», позиция 4 таблицы истинности. На затвор всех транзисторов подается логическая «1», открываются nMOS- транзисторы Т3 и Т4. Положительное напряжение Vdd через стоки – истоки открытых транзисторов Т3 и Т4 подтягивается к низкому уровню напряжения (Gnd). Верхние pMOS-транзисторы Т1 и Т2 заперты. (Рис. 8.4 в).

8.2.4. КМОП – вентиль ИЛИ

КМОП – вентиль ИЛИ (OR) состоит из двух параллельно соединенных nMOS- транзисторов (Т1 и Т2) для подтягивания выхода вентиля к высокому уровню напряжения (Vdd) и двух последовательно соединенных pMOS- транзисторов (Т3 и Т4) для подтягивается выхода вентиля к низкому уровню напряжения (Gnd), (рис. 8.5).



№	A	B	Транзистор T_1	Транзистор T_2	Транзистор T_3	Транзистор T_4	Y
1	0	0	Заперт	Заперт	Открыт	Открыт	0
2	0	1	Заперт	Открыт	Открыт	Заперт	1
3	1	0	Открыт	Заперт	Заперт	Открыт	1
4	1	1	Открыт	Открыт	Заперт	Заперт	1

г)

Рисунок 8.5 – КМОП – вентиль ИЛИ: обозначение (а), работа вентиля при значении одного из входов равной «1» (б), работа вентиля при значении обеих входов равной «0» (в) и таблица истинности вентиля ИЛИ (г)

КМОП – вентиль ИЛИ работает по следующим правилам:

a) Любой логическая «1» на входе вентиля, на выходе дает логическую «1» (см. рис. 8.5 б);
b) На выходе вентиля ИЛИ будет логическая «0», если все входы равны логическому «0» (см. рис. 8.5 в).

Рассмотрим примеры:

Вход A = «1», позиции 3 и 4 таблицы истинности. На затвор транзисторов T1 и T4 подается логическая «1», открывается первый nMOS- транзистор T1 и выходной сигнал через сток – исток транзистора T1 подает на выход логическую «1». Транзистор T3 заперт (см. рис. 8.5 б);

Входы A = B = «0», позиция 1 таблицы истинности. На затвор всех транзисторов подается логический «0», открываются pMOS-транзисторы T3 и T4 и выходной сигнал через стоки – истоки транзисторов T3 и T4 подтягивается к низкому уровню напряжения (Gnd).

8.2.5. КМОП – вентиль ИЛИ-НЕ

КМОП – вентиль ИЛИ-НЕ (OR-NOT) состоит из двух параллельно соединенных nMOS-транзисторов для подтягивания выхода вентиля к низкому уровню напряжения (Gnd), и двух последовательно соединенных pMOS-транзисторов для подтягивается выхода вентиля к высокому уровню напряжения (Vdd) (рис. 8.6).

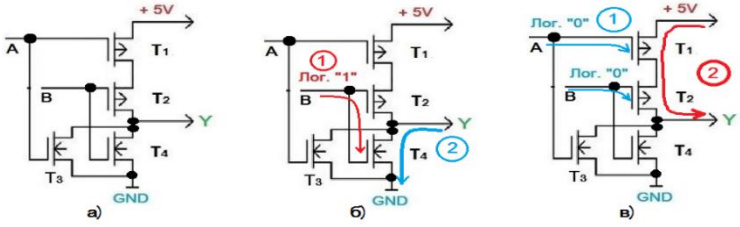


Рисунок 8.6 – КМОП – вентиль ИЛИ-НЕ: обозначение (а), работа вентиля при значении одного из входов равной «1» (б), работа вентиля при значении обеих входов равной «0» (в) и таблица истинности вентиля ИЛИ- НЕ (г)

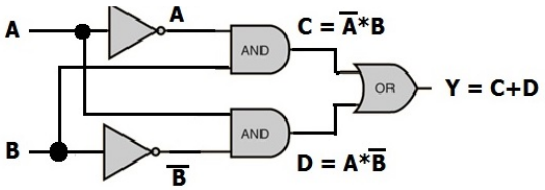
При подаче на любой из входов логической «1» (или оба входа A = B = «1»), открывается затвор одного из двух параллельно соединенных nMOS- транзисторов (или обе) через стоки – истоки которых выход вентиля подтягивается к низкому уровню напряжения (Gnd). Обе последовательно соединенных верхних pMOS-транзисторов в это время будут заперты (рис. 8.6 б).

Только при подаче на оба входа логического «0», открываются затворы двух последовательно соединенных pMOS – транзисторов (A = B = «0») и на выход подается логическая «1». При открытых затворах верхних последовательно соединенных двух транзисторов, через их стоки – истоки выход вентиля подтягивается к высокому уровню напряжения (Vdd). Обе параллельно соединенных нижних nMOS- транзисторов в это время будут заперты (рис. 8.6 в).

8.2.6. КМОП – вентиль Исключающее ИЛИ

Путем различных комбинаций логические элементы можно выражать друг через друга, а также строить из них схемы с более сложными логическими функциями. Любую логическую функцию можно выразить при помощи минимального набора базовых логических элементов. Такой минимальный набор называется базисом. Наиболее прост для понимания базис, состоящий из трех типов элементов – И, ИЛИ, НЕ. На практике, в цифровых устройствах чаще применяются базисы, состоящие из элементов И-НЕ или ИЛИ-НЕ, поскольку такие элементы проще составлять из транзисторов. Любая логическая схема может быть представлена в любом базисе.

Например, в виде комбинации логических элементов ИЛИ - НЕ можно представить логический элемент «Исключающее ИЛИ» (XOR). Этот элемент должен выдавать логическую единицу только в том случае, если два логических значения на его входах не равны друг другу. Другими словами - если единице равен не какой-нибудь из его входов, как в обычном элементе «ИЛИ», а ТОЛЬКО ОДИН из его входов (рис. 8.7, 8.8).



а)

№	A	B	(-A)	(-B)	C = (- A) * B	D = A * (-B)	Y = C + D
1	0	0	1	1	0	0	0
2	0	1	1	0	1	0	1
3	1	0	0	1	0	1	1
4	1	1	0	0	0	0	0

б)

Рисунок 8.7 – КМОП – вентиль Исключающее ИЛИ (а), таблица истинности (б)

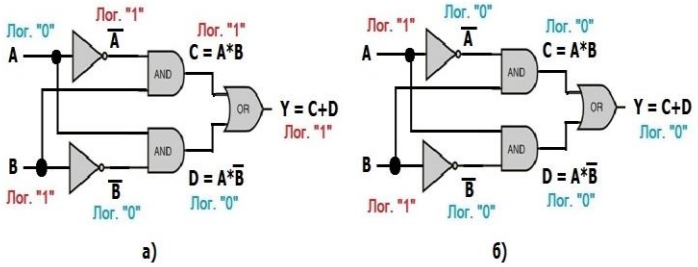
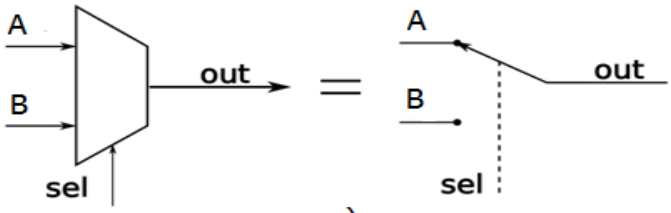


Рисунок 8.8 – Примеры работы вентиля Исключающее ИЛИ

8.2.7. Мультиплексор

В проектировании ПЛИС очень полезной логической структурой, является мультиплексор, который передает на выход значение одного из входов в зависимости от значения на адресном входе Sel (Select). Обозначение и принцип работы мультиплексора (а), таблица выбора переменной (б) приведены на рис. 8.9. Таблица выбора переменной (8.9, б) указывает: если Sel = 1, то из таблицы выбирается значения столбца B, иначе выбирается значения столбца A.



а)

№	Sel	A	B	Out
1	0	0	*	0
2	0	1	*	1
3	1	*	0	0
4	1	*	1	1

б)

Рисунок 8.9 – Описание мультиплексора (а), таблица выбора переменной (б)

8.2.8. Функциональный генератор (LUT- элемент)

Функциональный генератор, или LUT - элемент (Look – up Table – таблица преобразований), представляет собой небольшой блок памяти объемом 2^n одноразрядных слов с n адресными входами и одним выходом. блок памяти может программироваться для реализации любой функции с n входами и одним выходом. Эта структура м. б. реализована на базе схем с перемычками или блока запоминающих ячеек и является основой для построения программируемых логических устройств (Рис. 8.10).

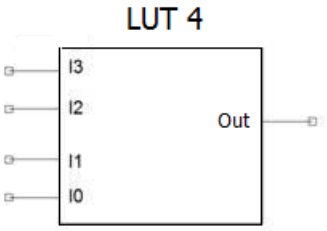


Рисунок 8.10 – LUT-элемент с 4 входами

ПЛИС с архитектурой FPGA (Field Programmable Gate Array) организованы в виде массива логических ячеек, состоящих из блоков комбинаторной логики LUT – элементов, мультиплексоров и триггеров (на рисунке обозначены FF). В исходном состоянии FPGA не реализует какой-то конкретной схемы, а функциональность приобретает после операции программирования – занесения в специальную конфигурационную память значений, управляющих цифровыми ключами, соединяющими отдельные элементы на кристалле FPGA. Также конфигурационная память управляет логическими генераторами, мультиплексорами и другими цифровыми узлами, которые в разных схемах могут выполнять различные функции (Рис.8.11).

Логические ячейки (logiccell) являются основным программируемым элементом, определяющим возможности ПЛИС FPGA. Основными компонентами ячейки являются логический генератор и триггер. Ячейки (т. е. LUT и соответствующие им триггеры и мультиплексоры) объединяются в секции (slice). Секции, в свою очередь, объединяются в конфигурируемые логические блоки (КЛБ, ConfigurableLogicBlock, CLB). Схема секции FPGA серии 7 показана на рис. 8.12.

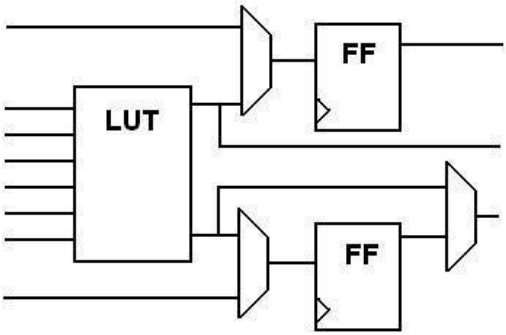


Рисунок 8.11 – Упрощенная схема логической ячейки FPGA

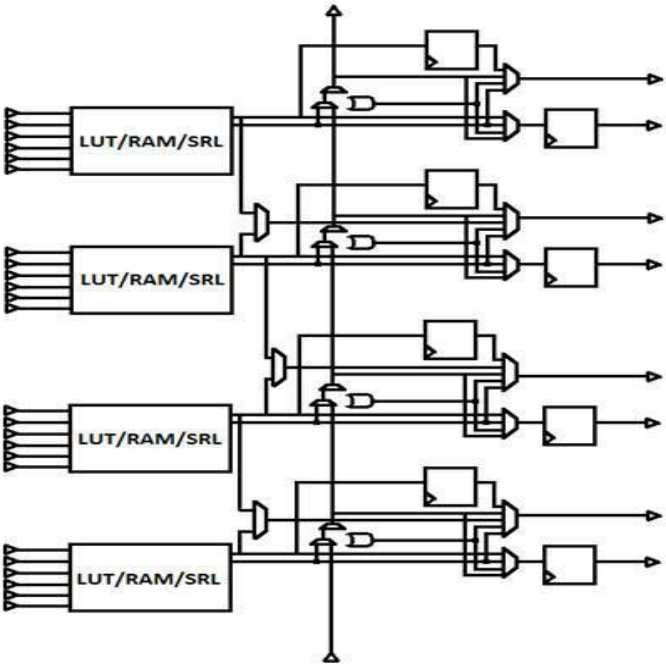


Рисунок 8.12 – Схема секции FPGA серии 7

ГЛАВА 9

АЛГЕБРА ЛОГИКИ

С развитием производства ИС возникло необходимость программирования их для специальных задач. Производство ИС относительно дорого. Изготовление ИС происходит посредством ряда технологических операций. Сперва, разрабатываются транзисторные каскады, которые собираются в логические структуры и далее производятся связи между ними. Изготовление микросхемы напоминает проявление фотографии с помощью масок. Маски – это стеклянные пластины, немного больше кремниевых пластин, на которые нанесены фотографически уменьшенные логические структуры. Они переносятся на полупроводниковые пластины с помощью светочувствительного лака, освещения и напыления. Окупаемость данной продукции экономически выгодно при изготовлении партии более 5000 штук. В настоящее время разработаны ИС, которые программируются производителем или самими пользователями.

Схемы с большими логическими структурами называются программируемыми вентильными матрицами. Вентильные матрицы – это схемы – полуфабрикаты, которые состоять из вентилей (логические элементы) и могут соединяться между собой программно по желанию пользователя. Линии связи производятся масками и данный вид программирования называется – масочным программированием.

Вентили могут быть собраны полупроводниковой пластине в некоторые наборы элементов, называемых стандартными ячейками, которые затем собираются в более сложные схемы. Данные вентили программируются производителем и ориентированы на конечного заказчика.

9.1. Аксиомы и тождества алгебры логики

9.1.1. Переменные и постоянные величины

В алгебре логики есть понятия переменных и постоянных величин (констант). Но, в отличие от обычной алгебры они могут иметь только два значения: 0 или 1, соответствующим логическим состояниям. Следовательно, переменные и постоянные величины в алгебре логики являются бинарными величинами. Входные/выходные величины (переменные, постоянные) в алгебре логики принято обозначать буквами латинского алфавита – A, B, C, …. Их можно изобразить в виде замкнутых/разомкнутых выключателей или проводов, при этом замкнутый ключ выключателя соответствует переменной величине – логической «1», а разомкнутый ключ выключателя соответствует переменной величине – логическому «0» (Рис. 9.1).

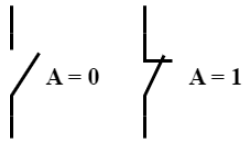


Рисунок 9.1 – Представление возможных состояний переменной A

Постоянные величины можно изобразить в виде «фиксированных переключателей». Если данный переключатель постоянно разомкнут (разрыв линии), то постоянная величина имеет значение – логического «0», если переключатель постоянно замкнут (неразрывное соединение – обычный провод), то постоянная величина имеет значение – логической «1» (Рис. 9.2).

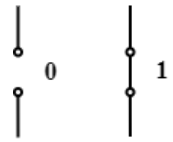
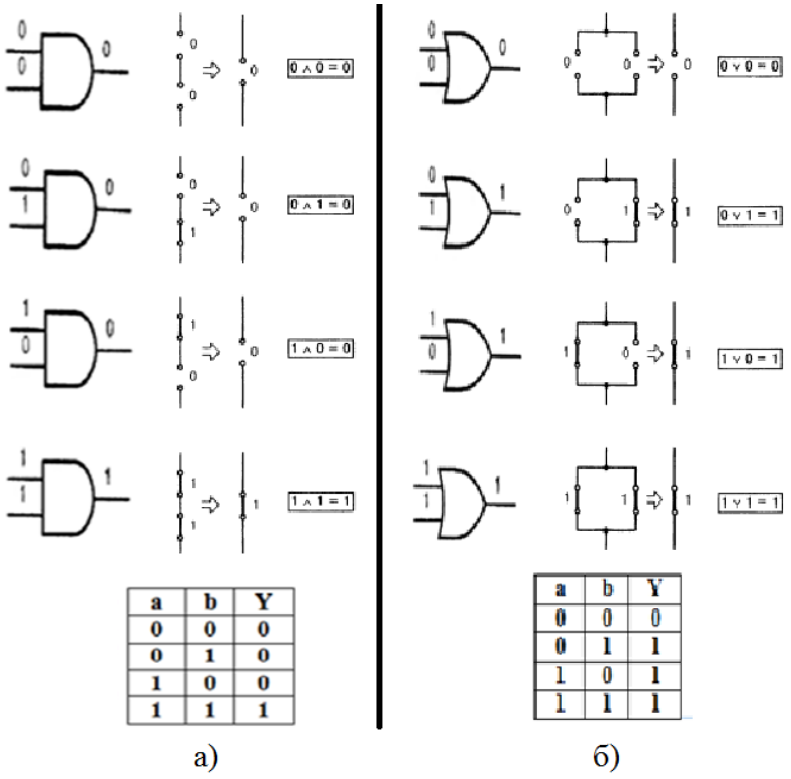


Рисунок 9.2 – Представление возможных состояний константы

9.1.2. Законы алгебры логики

Основные законы алгебры логики являются правилами, действующими для логических операций над постоянными величинами:

а) Логическое умножение (И). Схематично данную операцию можно представить в виде последовательных соединений ключей. Логическая «1» на выходе будет только в том случае, если оба ключа одновременно замкнуты, иначе на выходе всегда логический «0» (Рис. 9.3 а).



a	b	Y
0	0	0
0	1	0
1	0	0
1	1	1

a	b	Y
0	0	0
0	1	1
1	0	1
1	1	1

Рисунок 9.3 – Законы логического умножения И (а) и логического сложения ИЛИ (б)

б) Логическое сложение (ИЛИ). Схематично операцию можно представить в виде параллельно соединенных ключей. Логический «0» на выходе будет только в том случае, если оба ключа одновременно разомкнуты, иначе на выходе всегда логическая «1» (Рис. 9.3 б).

в) Логическое отрицание (НЕ). Входной логический «0» на выходе инвертируется в логическую «1», логическая «1» на входе, на выходе инвертируется в логический «0» (Рис. 9.4).

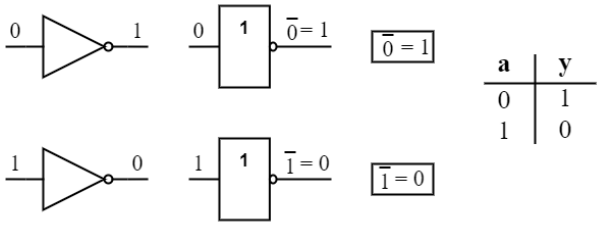


Рисунок 9.4 – Закон логического отрицания

9.1.3. Аксиомы алгебры логики

Правила для логических операций переменной величины с константой или переменной величины с самим собой или ее инвертированным значением называются аксиомами. Все девять аксиом пронумерованы от 1 до 9 и далее при их использовании мы будем ссылаться на данную нумерацию (Рис. 9.5, 9.6).

Обозначим некоторую переменною как А. Все, что верно для А, верно для любой переменной величины. Для представления инверсии применен нормально – замкнутый контакт. Он замкнут, если главный выключатель разомкнут. И размыкается при замыкании главного выключателя, т. о. при один из последовательно включенных ключей всегда разомкнут и в линии имеет место разрыв (0). Все четыре возможные аксиомы логического умножения показаны на рис. 9.5 а.

Аксиомы для логического сложения ИЛИ можно изобразить в виде параллельно включенных контактов (Рис. 9.5 б).

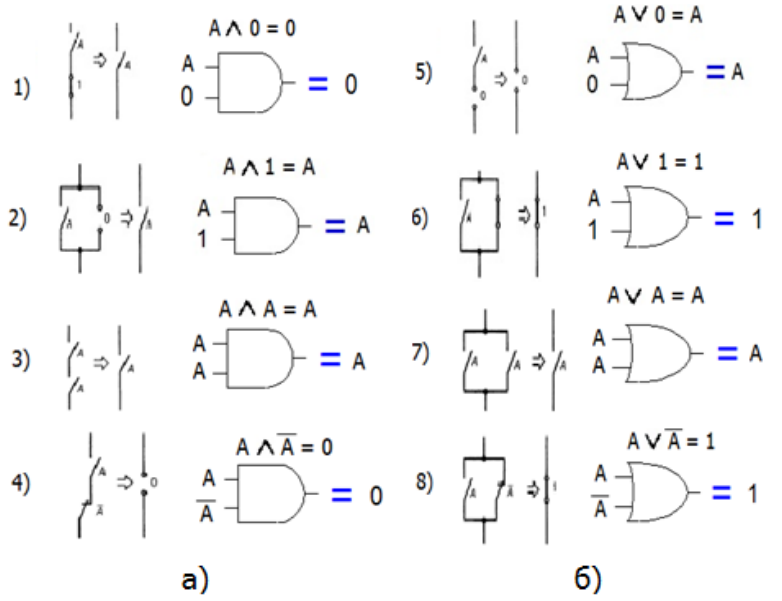


Рисунок 9.5 – Аксиомы логического умножения И (а) и логического сложения ИЛИ (б)

Если переменная инвертируется и затем еще раз инвертируется, то она принимает первоначальное значение. Два штриха инверсии над переменной не меняют ее состояния (Рис. 9.6).

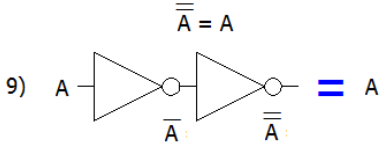


Рисунок 9.6 – Аксиома двойного логического отрицания

9.1.4. Закон коммутативности и ассоциативности

Закон коммутативности также называют переместительным законом и его можно сформулировать для операций умножения и сложения следующим образом:

а) Результат операции логического умножения И не зависит от порядка обработки переменных (Рис. 9.7 а).

б) Результат операции логического сложения ИЛИ не зависит от порядка обработки переменных (Рис. 9.7 б).

$$Z = A \wedge B \wedge C = C \wedge A \wedge B \qquad Z = A \vee B \vee C = C \vee A \vee B$$

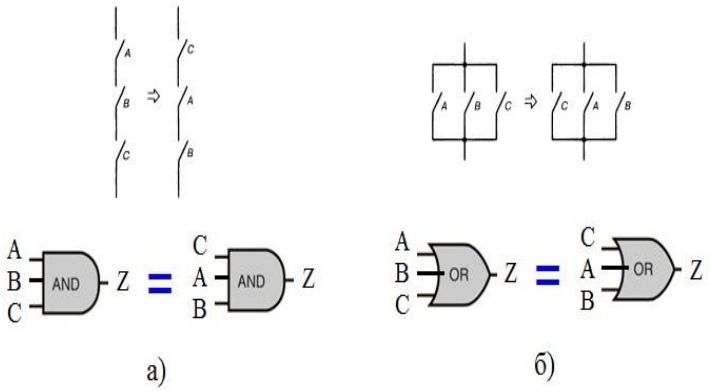


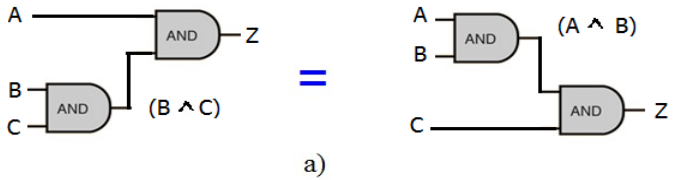
Рисунок 9.7 – Переместительный закон для логического умножения И (а) и логического сложения ИЛИ (б)

Закон ассоциативности также называют сочетательным законом и его можно сформулировать для операций умножения и сложения следующим образом:

а) Результат операции логического умножения И не зависит от порядка обработки переменных (Рис. 9.8 а).

б) Результат операции логического сложения ИЛИ не зависит от порядка обработки переменных (Рис. 9.8 б).

$$Z = A \wedge (B \wedge C) = (A \wedge B) \wedge C$$



а)

$$Z = A \vee (B \vee C) = (A \vee B) \vee C$$

б)

Рисунок 9.8 – Сочетательный закон для логического умножения И (а) и логического сложения ИЛИ (б)

9.1.5. Дистрибутивный закон

Дистрибутивный закон также называют распределительным законом. Распределительный закон для операций логического умножения по отношению к сложению играет большую роль на практике по преобразованию логических выражений.

а) Конъюнктивный распределительный закон:

$$Z = A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$$

Переменная А в операции логического умножения «распределяется» по переменным В и С. Оба контакта А могут коммутироваться только одновременно, узлы 1 и 2 можно соединить без изменения действия схемы. Схема на рисунке доказывает правильность данного тождества (Рис. 9.9 а). Чтобы показать правильность закона можем проверить его с помощью таблицы истинности. Состояния переменных в колонках Х и У

одинаковы. Значит, конъюнктивный распределительный закон верен (Рис. 9.9 б).

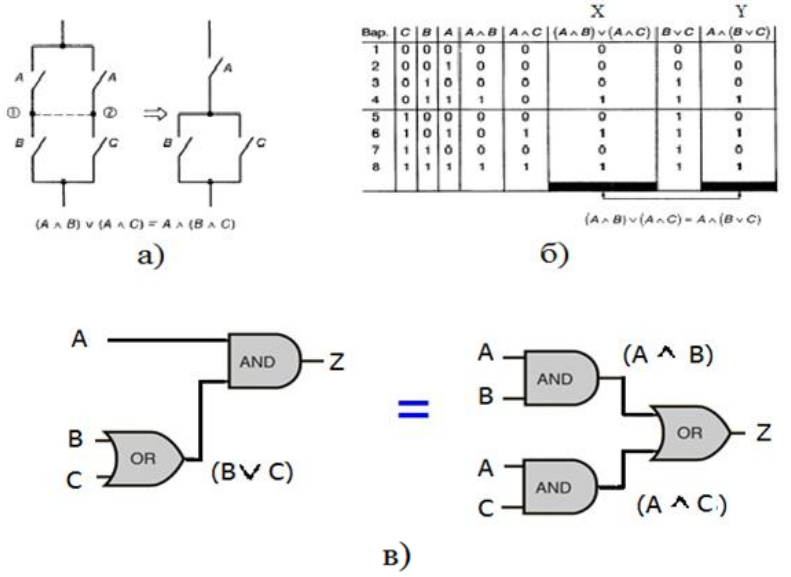
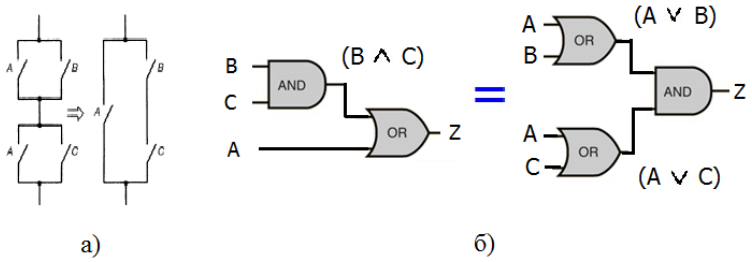


Рисунок 9.9 – Конъюнктивный распределительный закон (а), его таблицы истинности (б) и схемотехническое изображение (в)

б) Дизъюнктивный распределительный закон:

$$Z = A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$$

Переменная А в операции логического сложения «распределяется» по переменным В и С. Оба контакта А могут коммутироваться только одновременно, схему преобразовать как показано на рисунке, без изменения действия схемы (Рис. 9.10).



а) б)

Рисунок 9.10 – Дизъюнктивный распределительный закон

9.1.6. Перемещение инверсии в КМОП-логике

Большинство разработчиков думают в терминах элементов И и ИЛИ, но в КМОП-логике, для которой предпочтительны элементы И-НЕ и ИЛИ-НЕ. В этом случае можно воспользоваться перемещением инверсии, чтобы преобразовать схему в элементы И-НЕ, ИЛИ-НЕ

Прямолинейное решение заключается в простой замене каждого элемента И на И – НЕ с инвертором, а каждого элемента И- на И-НЕ с инвертором, как это показано на рисунке 9.11.

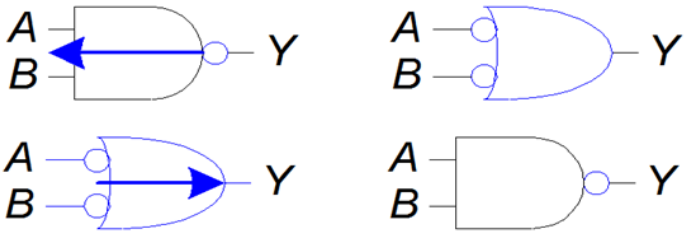


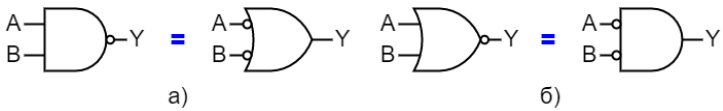
Рисунок 9.11 – Перемещении знака инверсии

Заметьте, что инверторы изображены с отрицанием на входе, а не на выходе, чтобы подчеркнуть, что последовательное двойное отрицание не меняет логику работы схемы и может быть отброшено.

9.1.7. Теоремы де Моргана

Теоремы де Моргана имеют большое практическое значение при упрощении инвертируемых выражений для логических операций с элементами И-НЕ и ИЛИ-НЕ. Существуют две теоремы де Моргана. Согласно теоремам, взаимно меняется тип логической операции (И или ИЛИ) (Рис. 9.12).

$$Y = \overline{A \wedge B} = \bar{A} \vee \bar{B} \qquad Y = \overline{A \vee B} = \bar{A} \wedge \bar{B}$$



а) б)

Рисунок 9.12 – Первая (а) и вторая (б) теоремы де Моргана

Теоремы де Моргана действуют также и для логических операций с большим количеством переменных.

$$Y = \overline{A \wedge B \wedge C \wedge D \wedge \ldots} = \bar{A} \vee \bar{B} \vee \bar{C} \vee \bar{D} \vee \ldots$$

$$Y = \overline{A \vee B \vee C \vee D \vee \ldots} = \bar{A} \wedge \bar{B} \wedge \bar{C} \wedge \bar{D} \wedge \ldots$$

9.1.8. Приоритеты логических операций

Логические операции с более высоким приоритетом выполняются перед другими логическими операциями. В алгебре логики более высоким приоритетом обладают операции логического умножения И, следовательно он выполняется в первую очередь.

Если в выражении алгебре логики присутствуют операции логического умножения и сложения, то переменные, связанные с логическим умножением, должны читаться так, как будто взяты в скобки.

$$Z = A \vee B \wedge C \Rightarrow A \vee (B \wedge C)$$

ГЛАВА 10

ПРИМЕНЕНИЯ ВЕНТИЛЕЙ

10.1. Функции И-НЕ и ИЛИ-НЕ

10.1.1. Логические элементы построенные на ИЛИ-НЕ

Элемент *ИЛИ-НЕ*. Из первой теоремы де Моргана следует, что элемент логического умножения всегда м. б. заменен элементом ИЛИ и несколькими элементами НЕ:

$\overline{A \wedge B} = \bar{A} \vee \bar{B}$ – первая теорема де Моргана

$$\overline{\overline{A \wedge B}} = \overline{\bar{A} \vee \bar{B}}$$

$$A \wedge B = \overline{\bar{A} \vee \bar{B}}$$

Отсюда следует правило: Любая логическая функция м. б. реализована только на элементах ИЛИ и НЕ.

Вентили на элементах ИЛИ и НЕ можно реализовать как элемент ИЛИ-НЕ (Рис. 10.1).

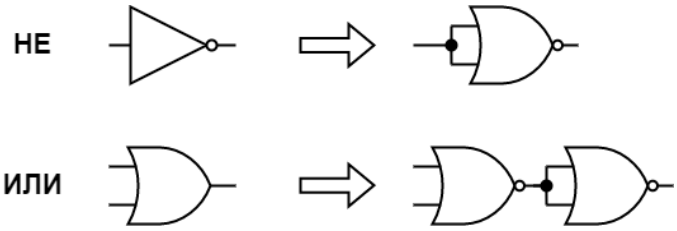


Рисунок 10.1 – Вентиль ИЛИ на элементах ИЛИ-НЕ

Если соединить входы элемента ИЛИ-НЕ вместе, то получается элемент НЕ (Рис. 10.1, а). Элемент ИЛИ получается путем инвертирования выхода ИЛИ-НЕ. Для этого к выходу элемента ИЛИ-НЕ последовательно подключается еще один элемент ИЛИ-НЕ, который действует как элемент НЕ (Рис. 10.1, б).

Элемент И-НЕ. Вентиль И м. б. образован согласно уравнению, следующему из первой теоремы де Моргана:

$$A \wedge B = \overline{\bar{A} \vee \bar{B}}$$

Для получения инверсий А и В применяют два элемента ИЛИ-НЕ. Для логической операции сложения используется еще один элемент ИЛИ-НЕ (Рис. 10.2).

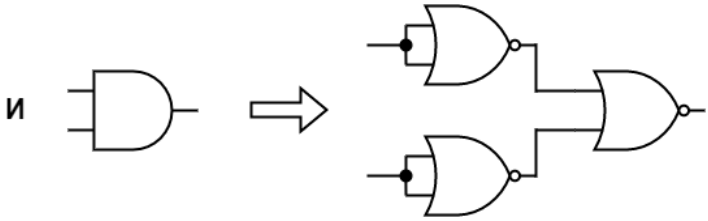


Рисунок 10.2 – Вентиль И на элементах ИЛИ-НЕ

Итак, элементы ИЛИ-НЕ можно использовать как универсальные логические элементы.

10.1.2. Логические элементы построенные на И-НЕ

Вентиль И. Из второй теоремы де Моргана следует, что логический элемент ИЛИ м. б. заменен логическим элементом И и несколькими элементами НЕ:

$\overline{A \vee B} = \bar{A} \wedge \bar{B}$ – вторая теорема де Моргана

$$\overline{\overline{A \vee B}} = \overline{\bar{A} \wedge \bar{B}}$$
$$A \vee B = \overline{\bar{A} \wedge \bar{B}}$$

Отсюда следует правило: Любая логическая функция м. б. реализована только на элементах И и НЕ.

Вентиль НЕ можно реализовать на элементах И-НЕ. Если соединить входы элемента И-НЕ вместе, то получается элемент НЕ (Рис. 10.3, а). Элемент И получается путем последовательно включения двух элементов И-НЕ, второй элемент которого действует как элемент НЕ (Рис. 10.3, б).

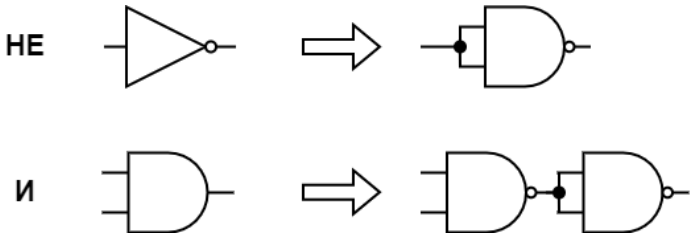


Рисунок 10.3 – Вентиль И собранный на элементах И-НЕ

Вентиль *ИЛИ*. Из второй теоремы де Моргана следует:

$$A \vee B = \overline{\bar{A} \wedge \bar{B}}$$

Для реализации вентиля ИЛИ требуется два элемента И-НЕ, включенные как элемент НЕ. Еще один последовательно включенный элемент И-НЕ используется для логического умножения с инверсией (Рис. 10.4).

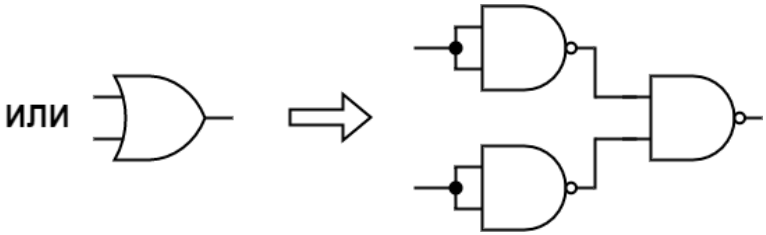


Рисунок 10.4 – Вентиль ИЛИ, собранный на элементах И-НЕ

Элементы И-НЕ, так же как элементы ИЛИ-НЕ можно использовать как универсальные логические элементы.

10.2. Примеры применения вентилей

Для синтеза цифровых схем только на элементах И-НЕ и/или ИЛИ-НЕ требуется многошаговые преобразование уравнений алгебры логики. Данные преобразования м. б. произведены разными способами.

Преобразуем уравнение так, чтобы соответствующая схема содержала только элементы И-НЕ:

$$Z = (\bar{A} \wedge \bar{B} \wedge C) \vee (A \wedge \bar{B} \wedge \bar{C})$$

Путь, обычно ведущий к цели кратчайшей дорогой, начинается с двойного отрицания, т. к. оно не меняет результат выражения:

$$Z = \overline{\overline{(\bar{A} \wedge \bar{B} \wedge C) \vee (A \wedge \bar{B} \wedge \bar{C})}}$$

Применяем второй закон де Моргана: $Y = \overline{A \vee B} = \bar{A} \wedge \bar{B}$. Нижнюю инверсия разбивается на две части по количеству элементов заключенных в скобки, и операция дизъюнкция меняется на операцию конъюнкция:

$$Z = \overline{\overline{(\bar{A} \wedge \bar{B} \wedge C)} \wedge \overline{(A \wedge \bar{B} \wedge \bar{C})}}$$

И далее учитывая, что двойное отрицание дает само значение переменной:

$$Z = \overline{\left(\bar{\bar{A}} \wedge \bar{\bar{B}} \wedge \bar{C}\right) \wedge \left(\bar{A} \wedge \bar{\bar{B}} \wedge \bar{\bar{C}}\right)}$$

Получаем окончательное уравнение:

$$Z = \overline{(A \wedge B \wedge \bar{C}) \wedge (\bar{A} \wedge B \wedge C)}$$

Синтезированная на основе преобразованного уравнения схема (Рис. 10.5).

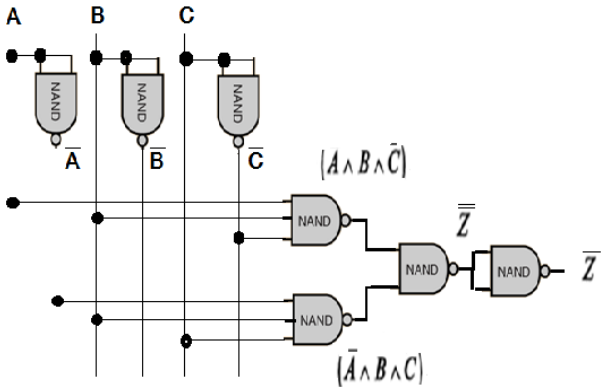


Рисунок 10.5 – Схема только на элементах И-НЕ

Если потребуется преобразовать нормальную форму ИЛИ так, чтобы соответствующая схема содержала только элементы ИЛИ-НЕ, то рекомендуется:

1. Дважды инвертировать каждую полную конъюнкцию;
2. Каждую нижнюю черту инверсии преобразовать в соответствии с первой теоремой де Моргана;
3. Еще раз подвергнуть уравнение двойной инверсии.

Пример: Дано

$$Z = (\bar{A} \wedge \bar{B} \wedge C) \vee (A \wedge \bar{B} \wedge \bar{C})$$

$$\mathbf{1}) \; Z = \overline{\overline{(\bar{A} \wedge \bar{B} \wedge C)} \vee \overline{(A \wedge \bar{B} \wedge \bar{C})}}$$

$$\mathbf{2}) \; \overline{\overline{(\bar{A} \wedge \bar{B} \wedge C)}} = \overline{\left(\bar{\bar{A}} \wedge \bar{\bar{B}} \wedge \bar{C}\right)}$$
$$\overline{\overline{(A \wedge \bar{B} \wedge \bar{C})}} = \overline{\left(\bar{A} \wedge \bar{\bar{B}} \wedge \bar{\bar{C}}\right)}$$
$$Z = \overline{(A \wedge B \wedge \bar{C})} \vee \overline{(\bar{A} \wedge B \wedge C)}$$

$$\mathbf{3}) \; Z = \overline{\overline{(A \wedge B \wedge \bar{C})} \vee \overline{(\bar{A} \wedge B \wedge C)}}$$

Синтезированная на основе преобразованного уравнения схема (Рис. 10.6).

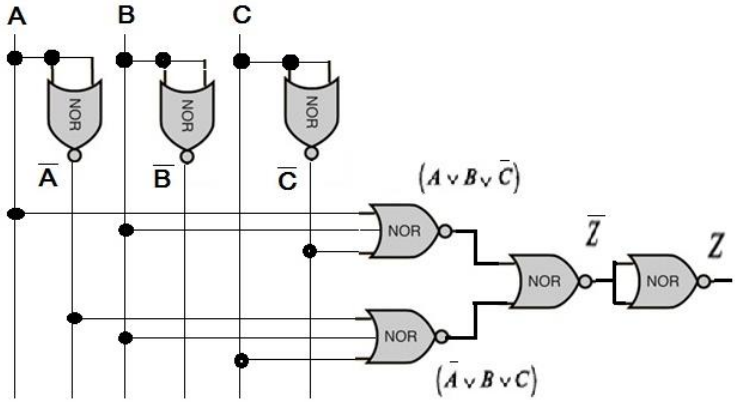


Рисунок 10.6 – Схема только на элементах ИЛИ-НЕ

На практике часто возникают трудности при преобразовании алгебраических уравнений в логические операции И-НЕ и ИЛИ-НЕ. Можно способ избежать этих преобразований. Отдельные основные элементы можно заменить элементами И-НЕ и ИЛИ-НЕ, как указано на рисунках 10.1–10.6. При этом получается более сложные схемы, которые, тем не менее легко упрощать. Часто встречаются друг за другом вентили НЕ, которые взаимно уничтожаются, т. к. двойное отрицание не меняет результат выражения. Вследствие чего схема значительно упрощается.

Пример такого упрощения показано ниже. Задана функция (Рис. 10.7):

$$Z = (\bar{A} \wedge \bar{B} \wedge C) \vee (A \wedge \bar{B} \wedge \bar{C})$$

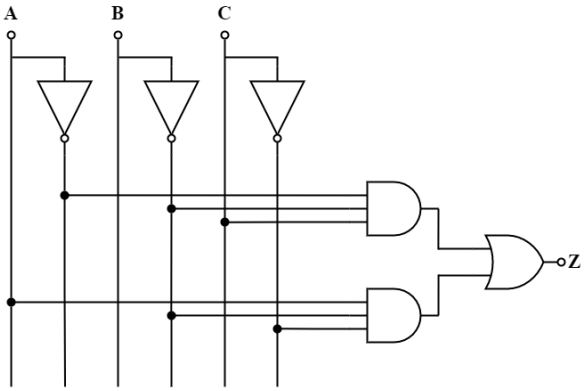


Рисунок 10.7 – Схема на основных элементах

Преобразуем эту схему в схему, состоящую только на элементах И-НЕ. Для этого каждый основной элемент схемы заменяется соответствующим ему элементом И-НЕ. Последовательно включенные элементы НЕ вычеркиваются. В результате получается схема на рис. 10.5. (Рис. 10.8).

$$Z = \overline{\overline{(\bar{A} \wedge \bar{B} \wedge C)} \wedge \overline{(A \wedge \bar{B} \wedge \bar{C})}}$$

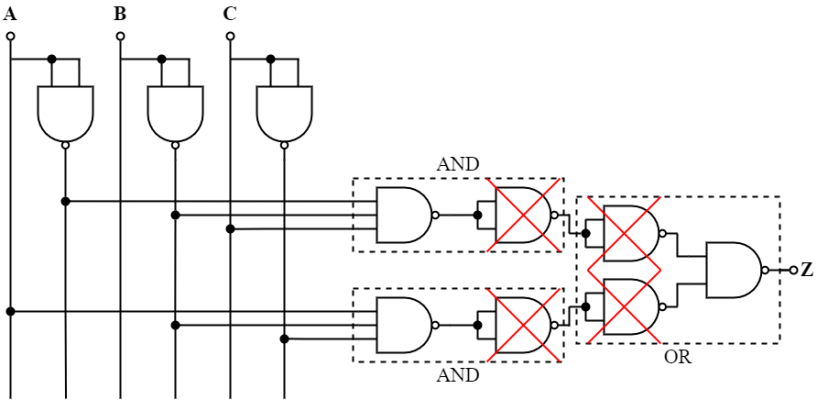


Рисунок 10.8 – Схема только на элементах И-НЕ

ГЛАВА 11

СИНТЕЗ СХЕМ. НОРМАЛЬНАЯ ФОРМА ЗАПИСИ

11.1. Синтез схем на логических элементах по заданным условиям

Цифровые электронные схемы на логических элементах применяются в качестве схем управления для самых различных задач контроля и регулирования технологических объектов. Под синтезом схемы понимают ее проектирование (разработку).
Перед началом синтеза схема д. б. четко и однозначно сформулирована задача, которую должна решать данная схема.
В первую очередь нужно назначить входные переменные. В качестве символов применяют заглавные буквы алфавита, начиная с первой, с индексом или без. Словесное описание часто бывают неоднозначными.

Входные переменные, например: $A, B, C, D, E, F, G, E_1, E_2, E_3$.

Затем назначают выходные переменные. В качестве символов применяют заглавные буквы алфавита, начиная с последней.

Выходные переменные, например: Z, Y, X, V_1, V_2, V_3.

Далее необходимо обговорить при каких условиях переменные равны «1» или «0».

После этого приступаем к составлению таблицы истинности. Здесь станет ясно, являлось ли словесное описание однозначным. Если при составлении таблицы истинности

встречаются неясность, то ее надо сразу устранить путем обсуждения с остальными разработчиками и заказчиком.

Таблицы истинности однозначно определяет, как будет работать проектируемая схема.

После составления таблицы истинности подбираются логические элементы, на которых ее можно реализовать. Следует максимально упростить схему.

Для синтеза схемы можно выделить пять шагов:

1. Описание функции требуемой схемы;
2. Назначение входных и выходных переменных и присвоение значений «1» или «0»;
3. Составление таблицы истинности;
4. Определение необходимых логических операций;
5. Упрощение и при необходимости преобразование схем.

Пример синтеза схемы, предотвращающую пуск лифта при определенных условиях:

Шаг 1. Описание функции требуемой схемы

Лифт не может двигаться при открытой двери. Также не может трогаться при перегрузке. Для пуска необходимо нажать кнопку.

Шаг 2. Назначение входных и выходных переменных

Входная переменная A назначается для дверного контакта. A = 1 – контакт замкнут, A = 0 – контакт разомкнут.

Входная переменная B назначается для перегрузки. B = 1 – перегрузка, B = 0 – нет перегрузки.

Входная переменная C назначается для кнопки. C = 1 – кнопка нажата, C = 0 – не нажата.

Выходная переменная Z. Z = 1 – лифт может двигаться, Z = 0 – запрет движению лифта.

Шаг 3. Составление таблицы истинности

Имеем три переменных. Следовательно, таблицы истинности имеет 8 возможных вариантов. Лифт может двигаться тогда, когда дверь заперта (A = 1), нет перегрузки (B =

0) и кнопка нажата (C = 1). Все данные условия выполняются в варианте №6 таблицы истинности. Для данного варианта Z = 1, а для остальных Z = 0 (Рис. 11.1).

Вар.	C	B	A	Z
1	0	0	0	0
2	0	0	1	0
3	0	1	0	0
4	0	1	1	0
5	1	0	0	0
6	1	0	1	1
7	1	1	0	0
8	1	1	1	0

Рисунок 11.1 – Таблица истинности работы лифта

Шаг 11. Определение необходимых логических операций

Для такой простой задачи применим метод подбора. Z = 1 только тогда, когда A = 1, B = 0 и C = 1. Если подать вход B на инвертор НЕ, то на выходе элемента получим «1». При A = 1, - B = 1 и C = 1 на входе элемента И имеем три «1» - состояния. Выход элемента – Z. Состояние Z = 1 означает, что к выходу Z подано напряжение +5 В и данное напряжение запускает реле (Рис. 11.2).

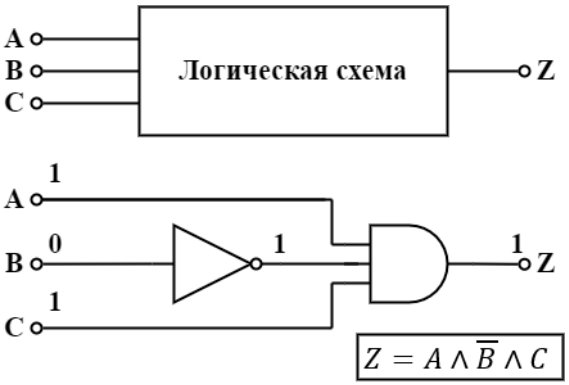


Рисунок 11.2 – Цифровая схема безопасности лифта

Шаг 5. Упрощение и при необходимости преобразование схем

Схему на рис. 11.2 упростить нельзя. Однако ее можно преобразовать. Предположим, что у нас на руках только элементы ИЛИ – НЕ. Тогда функцию $Z = A \wedge \bar{B} \wedge C$ можно преобразовать:

$$Z = A \wedge \bar{B} \wedge C = \overline{\overline{A \wedge \bar{B} \wedge C}} = \overline{\bar{A} \vee B \vee \bar{C}}$$

Схема, построенная на элементах ИЛИ–НЕ (Рис. 11.3).

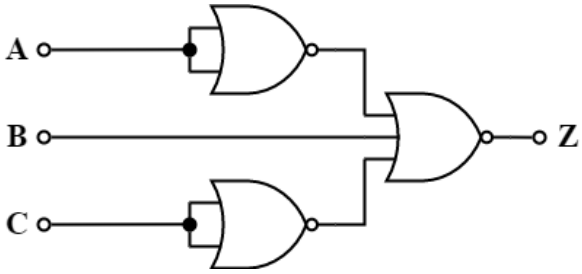


Рисунок 11.3 – Цифровая схема безопасности лифта на элементах ИЛИ-НЕ

11.2. Нормальная форма записи

Стандартизированные формы записи выражений называются в математике нормальными формами. Для определения целей необходимо логические функции приводить в нормальную форму.

11.2.1. Нормальная форма операции логического сложения ИЛИ

Данную форма записи называется также нормальной дизъюнктивной формой записи и является формой записи уравнения алгебры логики, в котором полные конъюнкции связаны друг с другом операцией логического сложения.

Под полной конъюнкцией понимают операцию логического умножения, в которой участвуют все имеющиеся переменные или их инвертированные значения.

Если имеются переменные A и B, то получится четыре возможные полные конъюнкции:

$A \wedge B$ $\quad \overline{A} \wedge B$ $\quad A \wedge \overline{B}$ $\quad \overline{A} \wedge \overline{B}$

Нормальная форма операции ИЛИ состоит из нескольких полных конъюнкций, которые логически складываются операцией ИЛИ. Она может состоят также из одной единственной полной конъюнкции.

Рассмотрим таблицу истинности для $\overline{A} \wedge \overline{B}$, который имеет только одно 1 – состояние (Рис. 11.4).

Вар.	B	A	$\overline{B}$	$\overline{A}$	$\overline{A} \wedge \overline{B}$
1	0	0	1	1	$1 \Rightarrow \overline{A} \wedge \overline{B}$
2	0	1	1	0	0
3	1	0	0	1	0
4	1	1	0	0	0

Рисунок 11.4 – Таблица истинности $\overline{A} \wedge \overline{B}$

Из этого следует вывод:

Количество 1 – состояний в выходном столбце таблицы истинности равно количеству полных конъюнкций нормальной формы ИЛИ.

Рассмотрим таблицу истинности для $A \wedge B$ (Рис. 11.5).

Вар.	B	A	Z
1	0	0	0
2	0	1	0
3	1	0	0
4	1	1	$1 \Rightarrow A \wedge B$

а)

Вар.	B	A	Z
1	0	0	$1 \Rightarrow \overline{A} \wedge \overline{B}$
2	0	1	$1 \Rightarrow A \wedge \overline{B}$
3	1	0	$1 \Rightarrow \overline{A} \wedge B$
4	1	1	$1 \Rightarrow A \wedge B$

б)

Рисунок 11.5 – Таблица истинности полной конъюнкции $A \wedge B$ (а) и соответствие полных конъюнкций возможным 1 – состояниям (б)

Из этого следует вывод:

Если в рассмотренном варианте таблицы истинности переменная принимает значение «0», то в соответствующей полной конъюнкции она инвертируется. Если в рассмотренном варианте таблицы истинности переменная принимает значение «1», то в соответствии полной конъюнкции она не инвертируется.

Соответствие полных конъюнкций возможным 1 – состояниям показано на рис. 11.5 б.

Рассмотрим пример: Дана таблица истинности (Рис. 11.6). Определить соответствующую нормальную форму ИЛИ. Каждое 1 – состояние в Z – столбце соответствует полной конъюнкции. 0 – состояния в Z – столбце не рассматриваются.

Вар.	C	B	A	Z
1	0	0	0	0
2	0	0	1	$1 \Rightarrow A \wedge \overline{B} \wedge \overline{C}$
3	0	1	0	0
4	0	1	1	0
5	1	0	0	$1 \Rightarrow \overline{A} \wedge \overline{B} \wedge C$
6	1	0	1	0
7	1	1	0	0
8	1	1	1	$1 \Rightarrow A \wedge B \wedge C$

Рисунок 11.6 – Таблица истинности

Рассмотрим 2-ю позицию таблицы истинности. Переменная A равна «1», поэтому она не инвертируется в полной конъюнкции. Переменные B и C равны «0». Они инвертируется в полной конъюнкции Т. о. полная конъюнкция для 2-й позиции имеет вид:

$$A \wedge \overline{B} \wedge \overline{C}$$

Соответственно для позиций 5 и 8 записываем полные конъюнкции. Нормальная форма ИЛИ является суммой всех полных конъюнкций:

$$Z = (A \wedge \overline{B} \wedge \overline{C}) \vee (\overline{A} \wedge \overline{B} \wedge C) \vee (A \wedge B \wedge C)$$

Эта нормальная форма ИЛИ представляет содержание таблицы истинности на рис. 11.6. Как и любое другое уравнение алгебры логики, нормальную форму можно преобразовать в таблицы истинности (Рис. 11.7).

Вар.	C	B	A	$\overline{B}$	$\overline{A}$	$A \wedge \overline{B} \wedge \overline{C}$	$\overline{A} \wedge \overline{B} \wedge C$	$A \wedge B \wedge C$	Z
1	0	0	0	1	1	0	0	0	0
2	0	0	1	1	0	1	0	0	1
3	0	1	0	0	1	0	0	0	0
4	0	1	1	0	0	0	0	0	0
5	1	0	0	1	1	0	1	0	1
6	1	0	1	1	0	0	0	0	0
7	1	1	0	0	1	0	0	0	0
8	1	1	1	0	0	0	0	1	1

Рисунок 11.7 – Обратное преобразование нормальной формы ИЛИ в таблицу истинности

С помощью нормальной формы ИЛИ возможно для любой заданной или составленной по описанию таблицы истинности записать соответствующие уравнения алгебры логики. Метод подбора уже не нужен. Можно без особых трудностей осуществлять синтез логических схем.

11.2.2. Нормальная форма операции логического сложения И

Данную форма записи называется также нормальной конъюнктивной формой записи и является формой записи уравнения алгебры логики, в котором полные дизъюнкции связаны друг с другом операцией логического умножения.

Под полной дизъюнкцией понимают операцию логического сложения, в которой участвуют все имеющиеся переменные или их инвертированные значения.

Если имеются переменные A и B, то получится четыре возможные полные конъюнкции:

$A \wedge B$ $\quad \overline{A} \wedge B$ $\quad A \wedge \overline{B}$ $\quad \overline{A} \wedge \overline{B}$

Нормальная форма операции ИЛИ состоит из нескольких полных дизъюнкций, которые логически перемножаются операцией И. Она может состоят также из одной единственной полной дизъюнкции.

Если работа с нормальной формой ИЛИ не вызывает затруднения, то нормальная форма И уже не нужна. Нормальную форму И можно легко превратить в нормальную форму ИЛИ.

Пример: перевести нормальную форму И в нормальную форму ИЛИ.

$$Z = (A \wedge \overline{B}) \vee (\overline{A} \wedge B)$$

Произведем операцию двойного отрицания:

$$Z = \overline{\overline{(A \wedge \overline{B}) \vee (\overline{A} \wedge B)}}$$

Согласно 2-й теореме де Моргана:

$$\overline{\overline{(A \wedge \overline{B}) \vee (\overline{A} \wedge B)}} = \overline{\overline{(A \wedge \overline{B})} \wedge \overline{(\overline{A} \wedge B)}}$$

$$Z = \overline{\overline{(A \wedge \overline{B})} \wedge \overline{(\overline{A} \wedge B)}}$$

Согласно 1-й теореме де Моргана:

$$\overline{\overline{(A \wedge \overline{B})}} = \overline{(\overline{A} \vee \overline{\overline{B}})} = \overline{(\overline{A} \vee B)}$$

$$\overline{\overline{(\overline{A} \wedge B)}} = \overline{(\overline{\overline{A}} \vee \overline{B})} = \overline{(A \vee \overline{B})}$$

$$Z = \overline{(\overline{A} \vee B)} \vee \overline{(A \vee \overline{B})}$$

На рисунке 11.8 показана схема на базе выше указанной формулы.

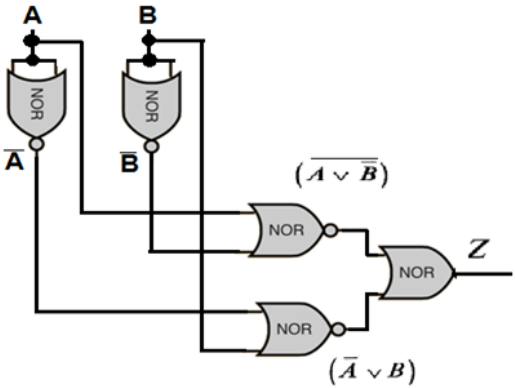


Рисунок 11.8 – Схема в нормальной форме ИЛИ

Если еще раз произведем операцию двойного отрицания:

$$Z = \overline{\overline{(\overline{A} \vee B)} \vee \overline{(A \vee \overline{B})}}$$

Получим схемы, где все элементы построены на базе ИЛИ –НЕ (Рис. 11.9).

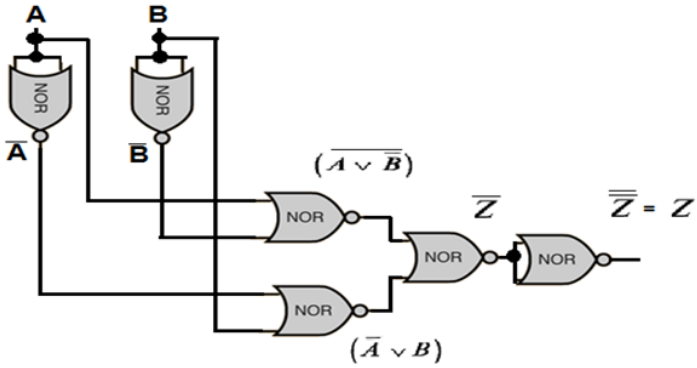


Рисунок 11.9 – Схема в нормальную форму ИЛИ – НЕ

11.3. Упрощение и преобразование нормальной формы ИЛИ

11.3.1. Упрощение нормальной формы ИЛИ

Нормальная форма ИЛИ воспроизводит содержание таблицы истинности в виде логического уравнения. По этому уравнению м. б. синтезирована нужная схема. По нормальной форме ИЛИ можно синтезировать схему, удовлетворяющей таблице истинности.

Часто такие схемы не являются самым простым из возможных вариантов. Во многих случаях нормальные формы ИЛИ можно упростить. Данные упрощения м. б. сделаны с помощью алгебры логики.

Пример: упростить нормальную форму ИЛИ

$$Z = (A \wedge B) \vee (A \wedge \overline{B})$$

Т. к. обе полные конъюнкции содержат переменную А, то она с помощью распределительного закона м. б. вынесена за скобки:

$$Z = A \wedge (B \vee \overline{B})$$

Выражение $B \vee \overline{B} = 1$, поэтому:

$$Z = A \wedge 1$$

Логическое сложение переменной с «1» дает в итоге переменную и результат упрощения будет иметь вид:

$$Z = A$$

Пример: упростить нормальную форму ИЛИ:

$$Z = \underset{\boxed{1}}{(\overline{A} \wedge B \wedge C)} \vee \underset{\boxed{2}}{(\overline{A} \wedge B \wedge \overline{C})} \vee \underset{\boxed{3}}{(\overline{A} \wedge \overline{B} \wedge C)} \vee \underset{\boxed{4}}{(\overline{A} \wedge \overline{B} \wedge \overline{C})}$$

Сначала упрощаем полные конъюнкции (1) и (2). $\overline{A} \wedge B$ рассматривается как одна переменная и выносятся за скобки:

$$\left(\left(\overline{A} \wedge B\right) \wedge C\right) \vee \left(\left(\overline{A} \wedge B\right) \wedge \overline{C}\right) = \left(\overline{A} \wedge B\right) \wedge \left(C \vee \overline{C}\right) = \\ = \left(\overline{A} \wedge B\right) \wedge 1 = \overline{A} \wedge B$$

Далее упрощаем полные конъюнкции (3) и (4). $\overline{A} \wedge \overline{B}$ рассматривается как одна переменная и выносятся за скобки:

$$\left(\left(\overline{A} \wedge \overline{B}\right) \wedge C\right) \vee \left(\left(\overline{A} \wedge \overline{B}\right) \wedge \overline{C}\right) = \left(\overline{A} \wedge \overline{B}\right) \wedge \left(C \vee \overline{C}\right) = \\ = \left(\overline{A} \wedge \overline{B}\right) \wedge 1 = \overline{A} \wedge \overline{B}$$

Получаем:

$$Z = \left(\overline{A} \wedge B\right) \vee \left(\overline{A} \wedge \overline{B}\right)$$

Инверсию А выносим за скобки:

$$Z = \overline{A} \wedge (B \vee \overline{B})$$

Т. к. $B \vee \overline{B} = 1$, поэтому:

$$Z = \overline{A} \wedge 1$$

Логическое сложение переменной с «1» дает в итоге переменную и результат упрощения будет иметь вид:

$$Z = \overline{A}$$

11.3.2. Преобразование нормальной формы ИЛИ

Схема, которая строится согласно нормальной форме ИЛИ должна базироваться на основных логических элементах. Во многих случаях можно использовать другие элементы, например И-НЕ или ИЛИ-НЕ. Нормальная форма ИЛИ в этих случаях д. б. преобразована.

Перевести нормальную форму ИЛИ на элементы И-НЕ очень просто. Нормальная форма ИЛИ сначала подвергается двойному отрицанию. Затем нижняя черта инверсии разделяется согласно второй теореме де Моргана.

Пример:

$$Z = \left(\overline{A} \wedge B \wedge \overline{C}\right) \vee \left(A \wedge \overline{B} \wedge C\right)$$

Двойное отрицание:

$$Z = \overline{\overline{\left(\overline{A} \wedge B \wedge \overline{C}\right) \vee \left(A \wedge \overline{B} \wedge C\right)}}$$

Применение второй теоремы де Моргана:

$$Z = \overline{\overline{\left(\overline{A} \wedge B \wedge \overline{C}\right)} \wedge \overline{\left(A \wedge \overline{B} \wedge C\right)}}$$

Схема построенная только элементами И-НЕ (Рис. 11.10)

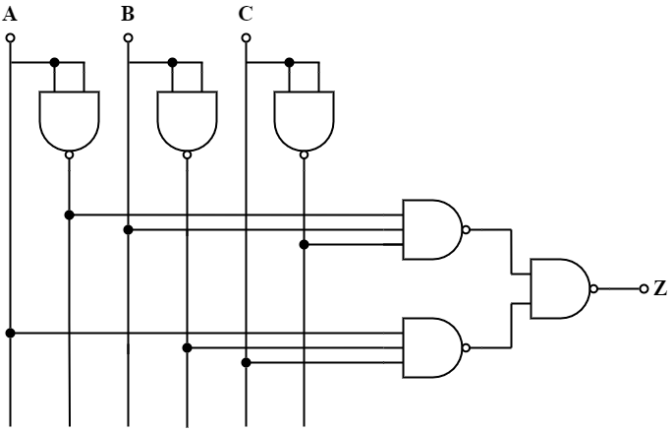


Рисунок 11.10 – Схема, построенная на элементах И-НЕ

Если необходимо преобразовать нормальную форму ИЛИ так, чтобы схема состояла из элементов ИЛИ-НЕ, то

рекомендуется дважды инвертировать каждую полную конъюнкцию и каждую нижнюю черту инверсии согласно первой теореме де Моргана. Затем вся схема еще раз подвергается двойному отрицанию.

$$Z = \left(\overline{A} \wedge B \wedge \overline{C}\right) \vee \left(A \wedge \overline{B} \wedge C\right)$$

Двойное отрицание:

$$Z = \overline{\overline{\left(\overline{A} \wedge B \wedge \overline{C}\right)}} \vee \overline{\overline{\left(A \wedge \overline{B} \wedge C\right)}}$$

Применение первой теоремы де Моргана:

$$Z = \overline{\left(A \vee \overline{B} \vee C\right)} \vee \overline{\left(\overline{A} \wedge B \wedge \overline{C}\right)}$$

Повторное двойное отрицание:

$$Z = \overline{\overline{\left(A \vee \overline{B} \vee C\right)}} \vee \overline{\overline{\left(\overline{A} \vee B \vee \overline{C}\right)}}$$

Схема построенная только элементами ИЛИ-НЕ (Рис. 11.11).

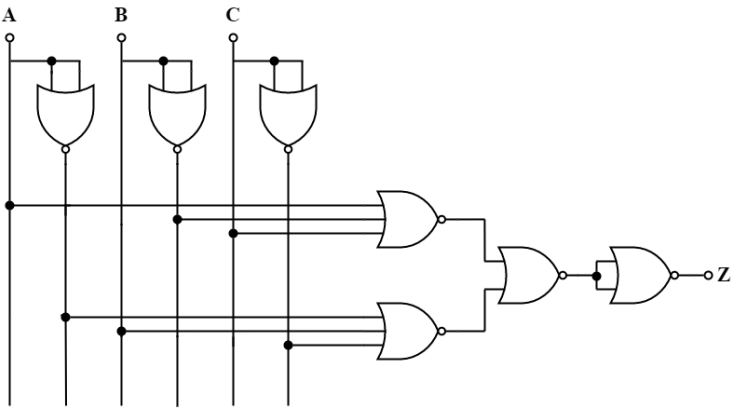


Рисунок 11.11 – Схема, построенная на элементах ИЛИ-НЕ

ГЛАВА 12

ПРОГРАММИРУЕМЫЕ ЛОГИЧЕСКИЕ СХЕМЫ

В цифровых устройствах используются два типа запоминающих устройств (ЗУ):

- постоянное запоминающее устройство (ПЗУ);
- оперативное запоминающее устройство (ОЗУ).

ПЗУ называют *энергонезависимыми* устройствами, т. к. хранимая в них информация не разрушается при отключении питания устройства. Другие компоненты устройства могут считывать информацию из ПЗУ, но не могут записывать в них новые данные.

В ОЗУ данные могут обновляться. Такие устройства называют *энергозависимыми*, т. к. хранимая в них информация не сохраняются при отключении питания устройства. ОЗУ впоследствии стали основой устройств ПЛИС.

12.1. Простая программируемая функция

12.1.1. Схема простой программируемой функции

Рассмотрим простую программируемую функцию. На входах таких функций присутствуют инверторы (операция НЕ), следовательно, любой сигнал м. б. представлен как в прямом, так и инверсной форме (Рис. 12.1).

Особого внимания заслуживают расположение предполагаемых перемычек. При их отсутствии на все входы И через нагрузочные резисторы подается логическая «1». Это значит, что на выходе И всегда присутствует состояние логической «1». Чтобы сделать простую программируемую функцию необходим некий механизм для изменения состояния

выхода элемента И. Такой управляющий механизм достигается с помощью перемычек.

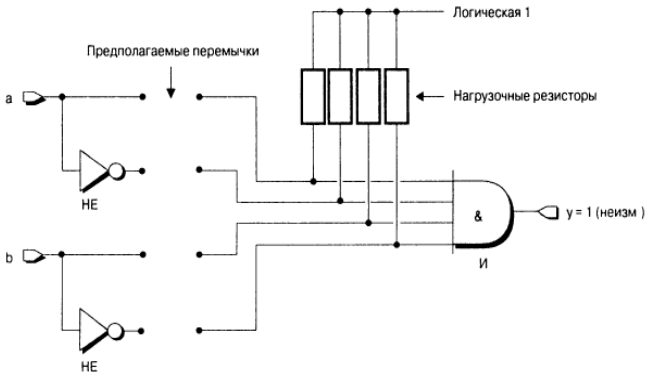


Рисунок 12.1 – Простая программируемая функция

12.1.2. Метод плавких перемычек

Одним из первых методов, позволявший пользователям программировать собственные устройства, был и до сих пор существующий метод плавких перемычек. При использовании данного метода устройство изготавливается со всеми возможными соединениями, каждое из которых представляет собой плавкую перемычку (Рис. 12.2).

Принцип действия этих перемычек аналогичен принципу действия предохранителей. При превышении тока потребления перегорает предохранитель, что приводит к разрыву цепи, по которой поступает ток.

Плавкие перемычки имеют микроскопические размеры, поскольку в кристалле используются те же технологии что при изготовлении транзисторов и проводников. Изначально все перемычки целы. Это означает, что в незапрограммированном состоянии на выходе функции И всегда логический «0».

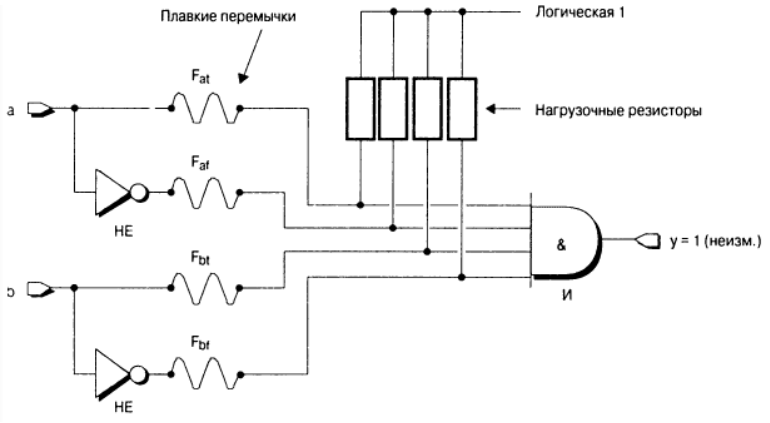


Рисунок 12.2 – Устройство с незапрограммированными плавкими перемычками

Разработчик могут выборочно удалять ненужные плавкие перемычки, подавая на входы устройства импульсы относительно высокого напряжения и большого тока, что приводить к выжиганию плавких перемычек (Рис. 12.3).

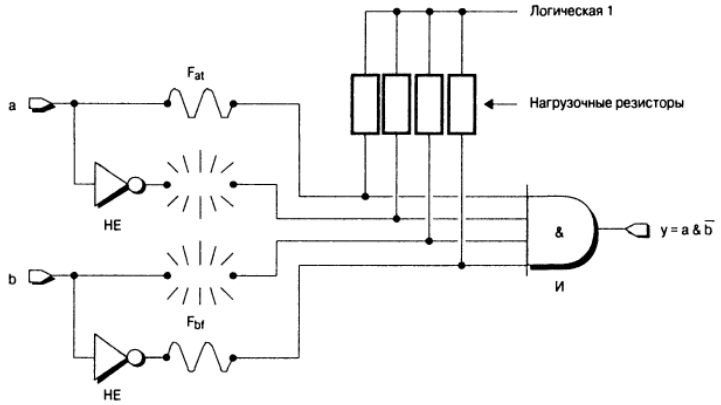


Рисунок 12.3 – Плавкие перемычки после программирования

Например, удалим перемычки F_{at} и F_{bt}, как показано на рис. 12.3. Удаление этих перемычек приведет к отсоединению инверсного входа A и прямого выхода B от логического элемента И. Через нагрузочные резисторы на этих входах устанавливается логическая «1». Это обстоятельство вынуждает устройство формировать новую функцию:

$$Y = A \,\&\, \overline{B}$$

Устройства, конфигурирование которых основано на методе плавких перемычек, являются однократно программируемыми устройствами, т. к. после прожига плавкая перемычка не м. б. заменена или возвращена в первоначальное состояние. Метод плавких перемычек в ПЛИС не применяется.

12.1.3. Метод наращиваемых перемычек

Данный метод прямо противоположен методу плавких перемычек. В этом случае каждое конфигурируемое соединение имеет линию связи, называемое наращиваемой перемычкой. В незапрограммированном состоянии наращиваемая перемычка имеет очень высокое сопротивление, что её можно рассматривать как разомкнутую цепь (Рис. 12.4).

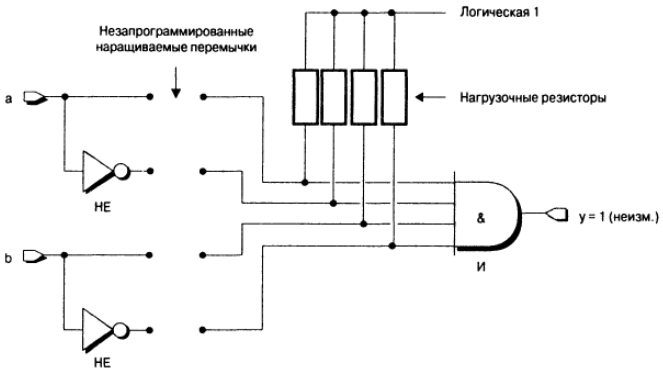


Рисунок 12.4 – Наращиваемые перемычки до программирования

Однако перемычки могут «наращиваться» (программироваться) с помощью подаваемых на входы устройства импульсов относительно высокого напряжения и большого тока. Например, если добавить наращиваемые перемычки в цепь инверсного входа A и прямого входа B, то устройство реализует функцию (Рис. 12.5):

$$Y = \overline{A} \,\&\, B$$

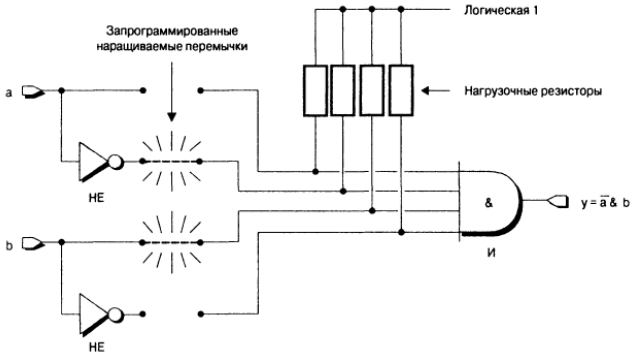


Рисунок 12.5 – Наращиваемые перемычки после программирования

12.1.4. Устройства, программируемые фотошаблоном

Типовые устройства ПЗУ также называются программируемые фотошаблоном или масочно-программируемыми, поскольку данные в них жестко прошиваются в процессе производства с помощью фотошаблона. Рассмотрим транзисторную ячейку ПЗУ, которая может хранить один бит данных (Рис. 12.6).

Стандартное устройство ПЗУ состоит из некоторого количества строк (адреса) и столбцов (данные), которые вместе образуют массив данных. К каждому столбцу подключен один нагрузочный резистор, который поддерживает на выводе столбца логическую «1». В каждом пересечении столбца и строки

присутствует транзистор и, при необходимости, перемычка. Наличие/отсутствие перемычки задается фотошаблоном.

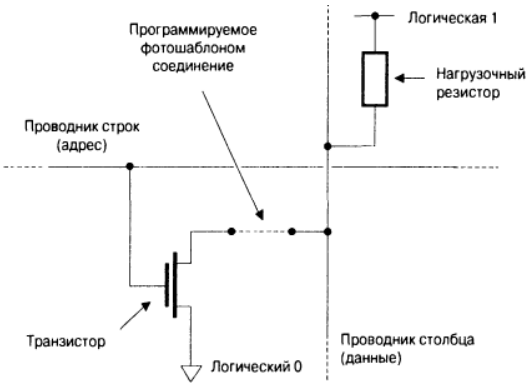


Рисунок 12.6 – Транзисторная ячейка ПЗУ, программируемая фотошаблоном

12.1.5. ППЗУ

Недостаток масочно–программируемых устройств состоит в том, что их производство является очень дорогостоящим.

Первые программируемые постоянное запоминающее устройство (ППЗУ) были разработаны в 1970 году. Для создания этих устройств использовался метод плавких перемычек из нихрома (Рис. 12.7).

ППЗУ, когда находится в незапрограммированном состоянии содержит плавкие перемычки. В этом случае при переводе строки в активное состояние будут включаться все транзисторы, подсоединенные к этой строке, вынуждая, т. о., все столбцы принимать значение логического «0». Разработчик может по своему усмотрению выборочно удалять ненужные плавкие перемычки. При их удалении на выходе ячейки будет формироваться уровень логической «1».

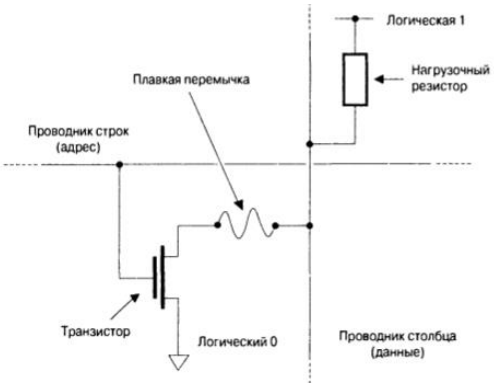


Рисунок 12.7 – Ячейка ПЗУ на основе транзистора с плавкой перемычкой

Данные ППЗУ изначально предназначались для использования в качестве устройств памяти. Однако, разработчики нашли им применение при реализации простых логических функций, таких как таблицы соответствия и конечные автоматы.

12.1.6. СППЗУ

Рассмотренные ранее устройства м. б. запрограммированы только один раз. В связи с этим возникла идея создания таких устройств, которые можно было бы перепрограммировать. Одним из вариантов этой идеи явилось стираемые программируемые постоянное запоминающее устройство (СППЗУ), которые были разработаны в 1971 г.

СППЗУ – транзистор имеет такую же структуру, как стандартный МОП – транзистор, но с дополнительным (вторым) плавающим затвором из поликристаллического кремния, изолированного слоями оксида кремния (Рис. 12.8).

В незапрограммированном состоянии плавающий затвор не заряжен и не влияет на работу обычного затвора. Чтобы запрограммировать транзистор, необходимо приложить к контактам затвора и стока относительно высокое напряжение +12 В. При этом транзистор резко включается и быстрые электроны

преодолевают слой оксида кремния направляясь в плавающий затвор. Этот процесс называется инжекцией горячих электронов. После снятия внешнего сигнала программирования электроны остаются в плавающем затворе. Их заряд стабилен и при соблюдении правил эксплуатации не рассеиваются в течение 10 лет. Накопленные на плавающем затворе заряды блокируют нормальную работу обычного затвора, и, т. о. позволяют различать программированные и непрограммированные ячейки. Благодаря этому свойству данные транзисторы можно использовать как ячейки памяти (Рис. 12.9).

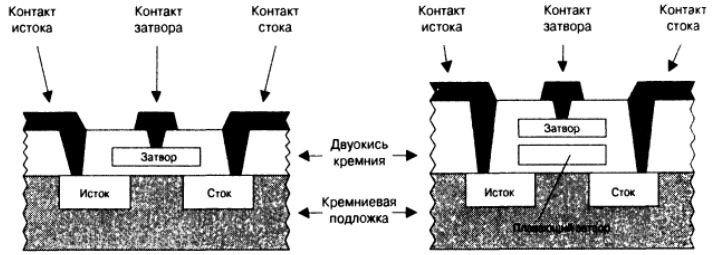


Рисунок 12.8 – Сравнение МОП-и СППЗУ-транзисторов

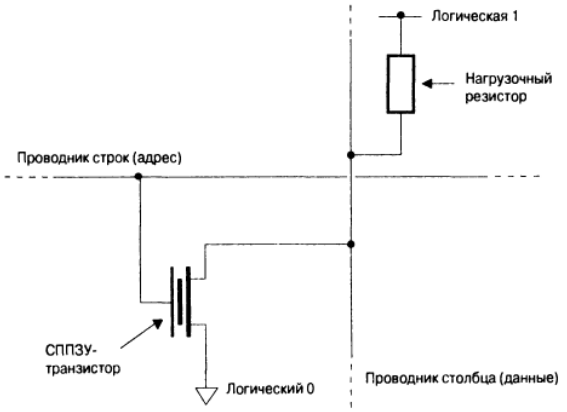


Рисунок 12.9 – Ячейка памяти на основе СППЗУ – транзистора

Стирание ячеек памяти – процесс «вытекания» электронов из плавающего затвора. Энергия, необходимая для «вытекания» электронов обеспечивается с помощью источника ультрафиолетового (УФ) излучения. Микросхемы СППЗУ поставляются в керамическом или пластиковом корпусе, наверху которого имеется небольшое кварцевое окно. Для стирания памяти надо поместить СППЗУ в контейнер с УФ – источником.

12.1.7. ЭСППЗУ

Ячейка ЭСППЗУ (электрически стираемые программируемые постоянное запоминающее устройство) приблизительно 2,5 раза больше, чем эквивалентная ей ячейка СППЗУ, т. к. она состоит из двух отстоящих друг от друга транзисторов (Рис. 12.10).

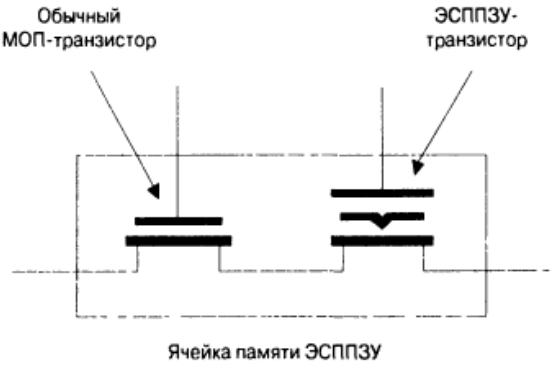


Рисунок 12.10 – Ячейка памяти ЭСППЗУ

Подобно СППЗУ, ЭСППЗУ тоже имеет плавающий затвор, но этот затвор окружен очень тонким изолирующим слоем оксида кремния. Второй МОП - транзистор используется для стирания ячейки памяти.

12.2. Логические схемы, программируемые пользователем (PLD)

Логические схемы, программируемые пользователем называются PLD (англ. Programmable Logic Devices, рус. ПЛУ - программируемые логические устройства).

Логические связи любой схемы могут быть выражены в нормальной форме ИЛИ, т. е. логической суммы ИЛИ полных конъюнкций. Для схем с двумя входными переменными получаются четыре полных конъюнкций (Рис. 12.11).

Вар.	*A*	*B*	*Z*	
1	0	0	1	$\overline{A} \vee \overline{B}$
2	0	1	1	$\overline{A} \vee B$
3	1	0	1	$A \vee \overline{B}$
4	1	1	1	$A \vee B$

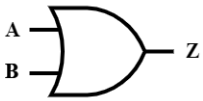


Рисунок 12.11 – Логической суммы ИЛИ полных конъюнкций

Программируемая комбинаторная схема с двумя входами будет иметь следующий вид (Рис. 12.12). Здесь с косой чертой обозначены программируемые связи.

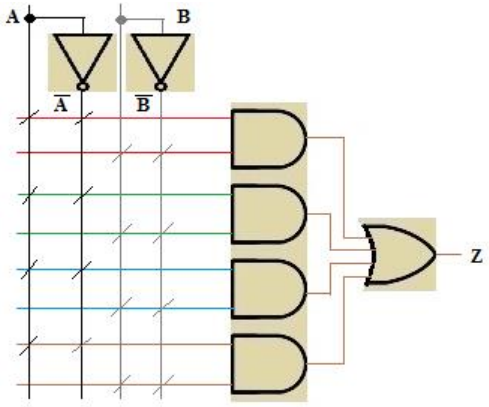


Рисунок 12.12 – Программируемая комбинаторная схема с двумя входами

В схемотехнике при графическом изображений PLD – схем используются следующие условные обозначения состояний узлов (Рис. 12.13).

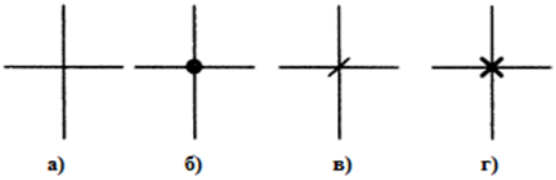


Рисунок 12.13 – Графические условные обозначения состояний узлов – пересечение линий, связи нет (а), неизменная (постоянная) связь (б), программируемый узел, непроводящая связь (в), программируемый узел, проводящая связь (г)

Все возможные нормальные формы ИЛИ при двух переменных показаны на следующем рисунке. Программирование состоит в том, чтобы правильно перемкнуть линии управления. В местах, где должен быть проводящая связь ставиться крестик (Рис. 12.14, 12.15).

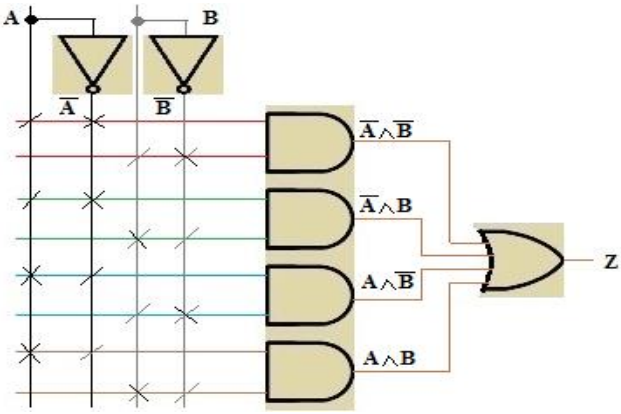


Рисунок 12.14 – Все возможные нормальные формы ИЛИ для двух переменных

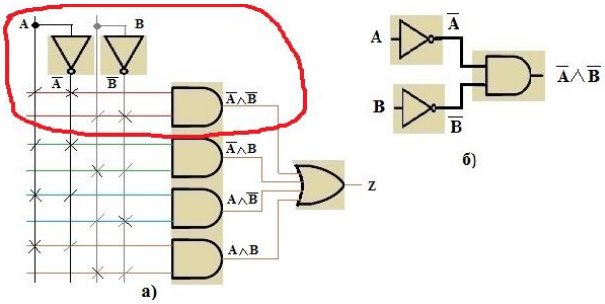


Рисунок 12.15 – Пояснение связей

Типичная PLD – схема, состоящая из программируемых И – связи, также называемая И – матрицей (AND Array) и ИЛИ – связи (ИЛИ – матрицей, OR Array) представлена на следующем рисунке. Все программируемые связи выполнены проводящими. Связи разделяют в соответствии с заявленными нормальными формами ИЛИ и/или другими уравнениями алгебры логики и получают необходимую схему. Ненужные элементы отключаются. Для PLD – схем с большим количеством элементов получается очень много проводников, усложняющие чтение схем (Рис. 12.16).

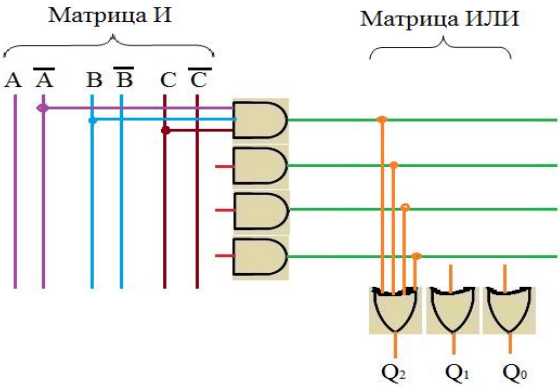


Рисунок 12.16 – PLD – схема с многожильными проводами

Для улучшения наглядности проводники изображаются как шины в виде косой черты с указанием количества проводов в ней (Рис. 12.17).

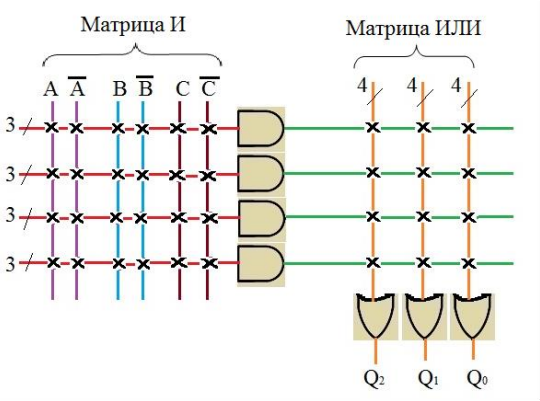


Рисунок 12.17 – PLD – схема с шинами

12.3. Программируемая матричная логика (PAL)

Программируемая матричная логика (PAL – Programmable Array Logic) имеют программируемую пользователем И – матрицу и фиксированную ИЛИ – матрицу. Входные сигналы усиливаются входными усилителями имеющих два выхода: прямой и инверсный (Рис. 12.18).

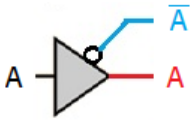


Рисунок 12.18 – Входной усилитель

Схема PAL – матрицы состоит из подключаемых к программируемым входам И – матриц входных сигналов (максимальное количество-6). Программируемые узлы обозначены косой чертой, что означает, что данная связь является

непроводящей. Выходы двух И – элементов подключены к ИЛИ – элементу. С этой схемой можно производить четыре различных связи. Они действуют на выходах от Q0 до Q3. Ненужные связи остаются неиспользованными. (Рис. 12.19).

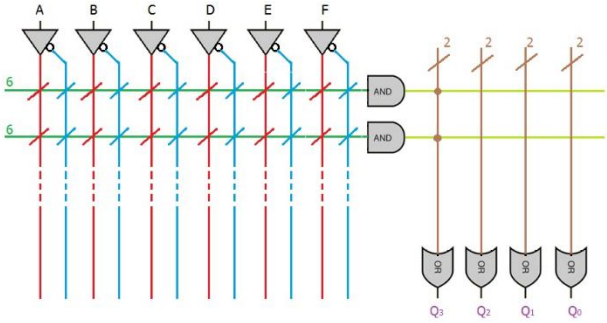


Рисунок 12.19 – Программируемая PAL – схема

Покажем, как программируется следующая функция. Величина Z поступает на выход Q3 (Рис. 12.20):

$$Z = \left(\overline{A} \wedge B \wedge \overline{C} \wedge D \wedge \overline{E} \wedge F\right) \vee \left(A \wedge B \wedge C \wedge \overline{D} \wedge E \wedge \overline{F}\right)$$

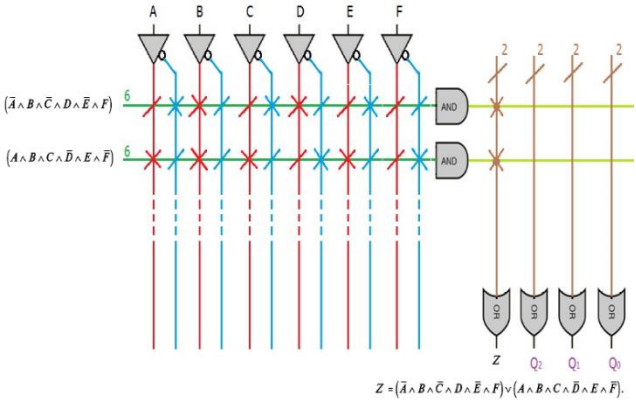


Рисунок 12.20 – Программированная PAL – схема

12.4. Программируемые пользователем логические матрицы (FPLA)

FPLA (Программируемые пользователем логические матрица) – имеет программируемые И – матрицу и ИЛИ – матрицу. И – матрица может производить все желаемые полные конъюнкции. С помощью ИЛИ – матрицы они подаются на ИЛИ – элементы с выходами от Q0 до Q3. Схема может обеспечить связи для четырех нормальных форм ИЛИ (Рис. 12.21).

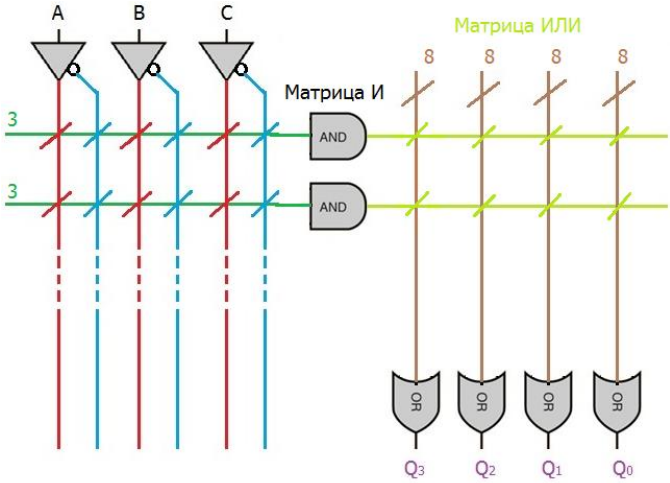


Рисунок 12.21 – Программируемая FPLA – схема

ГЛАВА 13

ПРИМЕНЕНИЯ ПЛИС. КЛАССИФИКАЦИЯ ПЛИС

13.1. Применения ПЛИС

13.1.1. Альтернатива «рассыпной логике»

До появления ПЛИС, когда перед конструкторами цифровых устройств вставала необходимость в создании некоторой цифровой схемы, у них было два варианта: использовать заказную микросхему ASIC, либо собирать эту схему из множества ИС малой интеграции, выполняющих простые функции – микросхем так называемой рассыпной логике. Разработка заказной микросхемы - очень дорогой процесс, и он может быть окуплен только при массовом серийном выпуске продукта. Собирать схему из рассыпной логики связано с большими трудностями по соединению этих элементов на интегральной схеме. Таким способом собрать большую схему в разумных (на современном уровне) габаритах невозможно, либо она будет занимать несколько шкафов и потреблять киловатты энергии. С появлением ПЛИС жизнь сильно упростилась, так как в одной микросхеме теперь можно было реализовать сложную схему, непосредственно по проекту разработчика.

13.1.2. Прототипирование микросхем

В одной или нескольких ПЛИС можно реализовать прототип будущего проекта заказной микросхемы и проверить его функциональность перед тем, как начинать долгий и дорогостоящий процесс ее изготовления.

13.1.3. Высокоскоростная обработка сигналов (радио, звук, видео)

В ПЛИС можно реализовать многие алгоритмы обработки потока данных (одномерных – звук, радиосигналы; двумерных – видео) на аппаратном уровне. Такая схема, по сути, будет являться специализированным вычислителем, способная обрабатывать данные только по одному алгоритму (в отличии от процессора общего назначения, на котором можно считать по любому алгоритму). Но за счет того, что она будет создана для реализации одного и только одного алгоритма, она будет работать максимально эффективно. В итоге быстродействие алгоритма будет намного выше, чем, например, при реализации на сигнальном или универсальном процессоре.

В таком ключе ПЛИС применяются при кодировании/декодировании сигналов радиосигналов сотовой связи или 4G WiMAX и LTE стандартов – там, где нужна высокоскоростная обработка в реальном времени. Также часто в реальном времени по фиксированным алгоритмам требуется обработка видео сигнала при кодировании/декодировании и при приеме/передачи. В системах видео наблюдения (отслеживание определенных событий при анализе экспериментальных данных в различных областях науки, или, например, в системах помощи водителю автомобиля) также могут использоваться ПЛИС. Множество статей о применении FPGA в области потоковой обработки сигналов можно найти на страницах онлайн журнала DSP-FPGA.

Обработка сигнала с видео камер с в реальном времени и принятие решений – очень горячая тема на сегодняшний день. Тысячи людей бьются над задачей научить машины видеть, понимать, что они видят, и реагировать на это. На сайте www.embedded-vision.com представлено много материалов на эту тему – в том числе и как использовать FPGA для обработки видео сигнала в реальном времени.

13.1.4. Высокопроизводительные реконфигурируемые вычисления

Эта тема весьма актуальна на данный момент. ПЛИС могут использоваться как со-процессоры (ускорители) для универсальных процессоров, берущие на себя вычисления наиболее математически интенсивных участков алгоритма. Как было сказано в предыдущем параграфе, в ПЛИС можно реализовать специализированный вычислитель со структурой, соответствующей выполняемому участку алгоритма. Тогда этот вычислитель будет наиболее эффективно использовать аппаратные ресурсы и решать задачу быстрее чем универсальный процессор. Дело в том, что процессор по сути последовательный вычислитель по программе, заложенной в памяти. Непосредственно полезными вычислениями он занимается довольно редко, а большую часть тактов выполняется другие задачи: выборка команды и данных из памяти, пересылка данных, ожидание готовности данных, обслуживание операционной системы.

ПЛИС в этом отношении могут быть более эффективны. В них можно организовать параллельные потоки вычислений, каждый такт которых будет использован для выполнения какой-либо математической операции: сложение, умножением и т. д. Эффективная специализированная схема может обогнать процессор потенциально в 100 раз при реализации одного и того же логического алгоритма. ПЛИС - ы используются как сопроцессоры во многих суперкомпьютерах. Отдельной областью применения FPGA является построение суперкомпьютера полностью на FPGA.

Множество примеров применения ПЛИС можно найти на страницах бесплатного журнала XCell, который выпускает Xilinx, его онлайн версию можно найти в следующем сайте: http://www.xilinx.com/publications/xcellonline/.

13.2. Классификация ПЛИС

Классификация ПЛИС использует структурный признак, который даёт наиболее полное представление о классе задач, пригодных для решения на ПЛИС. Общепринятой оценкой логической ёмкости ПЛИС является число эквивалентных вентилей, определяемое как среднее число вентилей 2И-НЕ, необходимых для реализации эквивалентного проекта на ПЛИС и базовом матричном кристалле (БМК). Эта оценка весьма условна, поскольку ПЛИС не содержат вентилей 2И-НЕ в чистом виде, однако для проведения сравнительного анализа различных архитектур она вполне пригодна. Основным критерием такой классификации является наличие, вид и способы коммутации элементов логических матриц. До настоящего времени в литературе отсутствует единая терминология по интегральным микросхемам ПЛИС, из-за чего в разных источниках они могут различаться.

Рассмотрим историю развития архитектур ПЛИС. В конце 1970 годов на рынке появились ПЛИС с программируемыми логическими матрицами, имеющие программируемые матрицы И и ИЛИ. В зарубежной литературе соответствующими этому классу аббревиатурами являются FPLA (Field Programmable Logic Array) и FPLS (Field Programmable Logic Sequencers). В СССР появились аналоги данных ПЛИС (см. рисунок 13.1) К556РТ1, РТ2, РТ21. Недостаток такой архитектуры – слабое использование ресурсов программируемой матрицы ИЛИ.

Идя по пути совершенствования такой архитектуры, разработчики ПЛИС предложили более простую архитектуру программируемой матричной логики (PAL – Programmable Array Logic и GAL – Gate Array Logic). Это ПЛИС, имеющие программируемую матрицу И и фиксированную матрицу ИЛИ. У ПЛИС GAL на выходе имеется триггер. К этому классу относится широкая номенклатура ПЛИС небольшой степени интеграции. В качестве примеров можно привести ИС производства СССР - КМ1556ХП4, ХП6, ХП8, а также ранние разработки (середина – конец 80-х годов) ПЛИС фирм Intel, Altera и др. Однако за любую унификацию надо платить избыточностью.

Другим подходом уменьшения избыточности программируемой матрицы (ПМ) ИЛИ является программируемая макрологика. ПЛИС, построенные по данной архитектуре, содержат единственную ПМ И - НЕ или ИЛИ - НЕ, но за счёт многочисленных инверсных обратных связей способны формировать сложные логические функции.

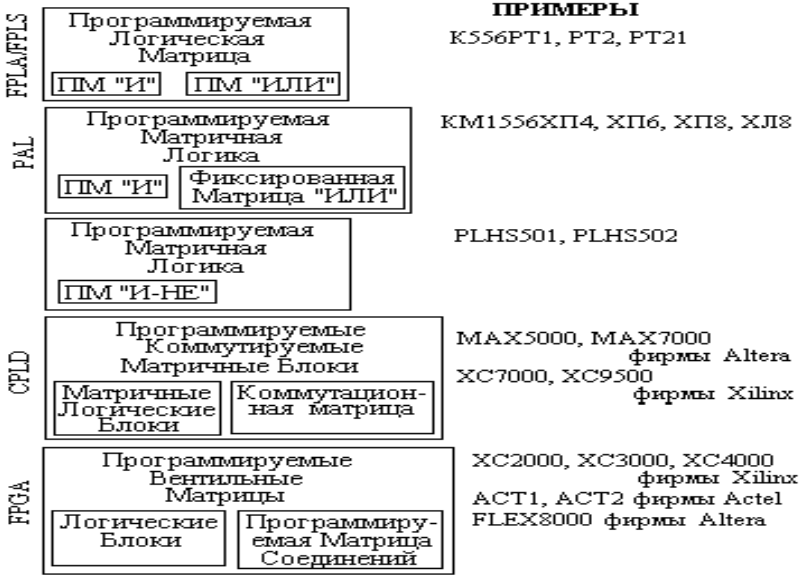


Рисунок 13.1 – Виды ПЛИС

Перечисленные архитектуры ПЛИС, содержащие небольшое число ячеек, к настоящему времени морально устарели и применяются для реализации относительно простых устройств, для которых не существует готовых ИС средней степени интеграции. Для реализации серьёзных алгоритмов управления они не пригодны.

В начале 80-х годов на мировой рынок микроэлектронных изделий выходят три ведущие фирмы-производители ПЛИС. В 1984 г. основана компания Xilinx, Inc. (www.xilinx.com), в 1985г. – компания Actel Corporation (www.actel.com), в 1988 г. – фирма

Altera Corporation (www.altera.com). Эти три компании занимают до 80 % всего рынка ПЛИС и являются основными разработчиками идеологии их применения. Если ранее ПЛИС являлись одним из множества продуктов, выпускаемых такими гигантами, как Intel, AMD и др., то, начиная с середины 80-х годов, на рынке ПЛИС происходит специализация и законодателями мод становятся фирмы, специализирующиеся на разработке и производстве ПЛИС.

С новыми производителями появились и новые архитектуры. ИС ПМЛ (PLD) имеют архитектуру, весьма удобную для реализации цифровых автоматов. Развитие этой архитектуры – CPLD (Complex Programmable Logic Devices) – ПЛИС, содержащие несколько логических блоков (ЛБ), объединённых коммутационной матрицей. Каждый ЛБ представляет собой структуру типа ПМЛ, т. е. программируемую матрицу И и фиксированную матрицу ИЛИ. ПЛИС типа CPLD, как правило, имеют высокую степень интеграции (до 10 000 эквивалентных вентилей, до 256 макроячеек). К этому классу относятся ПЛИС семейства MAX5000 и MAX7000 фирмы Altera, схемы XC7000 и XC9500 фирмы Xilinx.

Другой тип архитектуры ПЛИС – программируемые вентильные матрицы (ПВМ), состоящие из логических блоков (ЛБ) и коммутирующих путей – программируемых матриц соединений. Логические блоки таких ПЛИС состоят из одного или нескольких относительно простых логических элементов, в основе которых лежит таблица перекодировки (ТП – Look-Up Table, LUT), программируемый мультиплексор, D-триггер, а также цепи управления. Таких простых элементов может быть достаточно большое количество, у современных ПЛИС емкостью до 1 миллиона вентилей число логических элементов достигает нескольких десятков тысяч. За счет такого большого числа логических элементов они содержат значительное число триггеров, а также некоторые семейства ПЛИС имеют встроенные реконфигурируемые модули памяти (РМП – Embedded Array Block, EAB). Это делает ПЛИС данной архитектуры весьма удобным средством реализации алгоритмов цифровой обработки сигналов, основными операциями в

которых являются перемножение, умножение на константу, суммирование и задержка сигнала. Вместе с тем возможности комбинационной части таких ПЛИС ограничены, поэтому совместно с ПВМ применяют CPLD для реализации управляющих и интерфейсных схем. В зарубежной литературе такие ПЛИС получили название FPGA (Field Programmable Gate Array- программируемый полем массив вентилей). К FPGA (ПВМ) относятся ПЛИС XC2000–XC4000 фирмы Xilinx, ACT1, ACT2 фирмы Actel, а также семейства FLEX8000 фирмы Altera.

Множество конфигурируемых логических блоков (Configurable Logic Blocks, CLBs) объединяются с помощью матрицы соединений. Характерными для FPGA архитектурами являются элементы ввода/вывода (Input/Output Blocks, IOBs), позволяющие реализовать двунаправленный ввод / вывод, третье состояние и т. п.

Особенностью современных ПЛИС является возможность тестирования узлов с помощью порта JTAG (B-scan), а также наличие внутреннего генератора (Osc) и схем управления последовательной конфигурацией.

Фирма Altera пошла по пути развития FPGA архитектур и предложила в семействе FLEX10K так называемую двухуровневую архитектуру матрицы соединений. Логические элементы (ЛЭ) объединяются в группы – логические блоки (ЛБ). Внутри логических блоков ЛЭ соединяются посредством локальной программируемой матрицы соединений, позволяющей соединять любые ЛЭ. Логические блоки связаны между собой и с элементами ввода-вывода посредством глобальной программируемой матрицы соединений (ГПМС). Локальная и глобальная матрицы соединений имеют непрерывную структуру – для каждого соединения выделяется непрерывный канал.

13.3. Перспектива развития ПЛИС

Дальнейшее развитие архитектур идёт по пути создания комбинированных архитектур, сочетающих удобство реализации алгоритмов цифровой обработки сигналов на базе таблиц

перекодировок и реконфигурируемых модулей памяти, характерных для FPGA структур и многоуровневых ПЛИС с удобством реализации цифровых автоматов на CPLD архитектурах. Так, ПЛИС APEX20K Altera содержит в себе логические элементы всех перечисленных типов, что позволяет применять ПЛИС как основную элементную базу для «систем на кристалле» (System-On-Chip, SOC) В основе идеи SOC лежит интеграция всей электронной системы в одном кристалле (например, в случае ПК такой чип объединяет процессор, память и т.д.). Компоненты этих систем разрабатываются отдельно и хранятся в виде файлов параметризируемых модулей. Окончательная структура SOC-микросхемы выполняется на базе этих «виртуальных компонентов» с помощью программ систем автоматизации проектирования (САПР) электронных устройств EDA (Electronic Design Automation). Благодаря стандартизации в одно целое можно объединять «виртуальные компоненты» от разных разработчиков.

13.4. Критерии выбора ПЛИС

При выборе элементной базы устройств управления руководствуются следующими критериями отбора:

- быстродействием;
- логическая ёмкость, достаточная для реализации алгоритма;
- схемотехнические и конструктивные параметры ПЛИС, надёжность, рабочий диапазон температур, стойкость к ионизирующим излучениям и т. п.;
- стоимость владения средствами разработки, включающая как стоимость программного обеспечения, так и наличие, и стоимость аппаратных средств отладки;
- стоимость оборудования для программирования ПЛИС или конфигурационных ПЗУ;
- наличие методической и технической поддержки;
- наличие и надёжность поставщиков;
- стоимость микросхем.

ГЛАВА 14

АРХИТЕКТУРА ПЛИС ФИРМЫ XILINX

14.1. Архитектура ПЛИС фирмы XILINX

ПЛИС является одной из современной микросхем. Разработка цифрового управляющего устройства (ЦУУ) на базе ПЛИС осуществляется с помощью высокотехнологичного САПР. В общем случае ПЛИС похожа на СБИС, если прошивка конфигурации будет выполнена один раз на заводе – изготовителе. Стоимость последней будет ниже стоимости микросхемы ПЛИС. Но ПЛИС может изменять свою конфигурацию для каждого конкретного проекта. Для проектирование цифровых устройств разработчик должен знать:

- архитектуру ПЛИС, принцип её действия;
- САПР цифровых устройств на базе ПЛИС.

14.1.1. Назначение ПЛИС

ПЛИС предоставляет разработчикам большой спектр кристаллов с различной технологией производства, степенью интеграции, архитектурой, быстродействием, потребляемой мощностью и напряжением питания, выпускаемых в различных типах корпусов и в нескольких вариантах исполнения, включая промышленное, военное и радиационно-стойкое. Кристаллы в полной мере реализуют преимущества ПЛИС:

- высокое быстродействие;
- возможность перепрограммирования непосредственно в системе;
- высокая степень интеграции позволяет разместить ЦУУ на одном кристалле и снизить время и затраты на трассировку и производство печатных плат;

- сокращение времени цикла разработки и производства устройства;
- наличие мощных инструментов САПР, позволяющих устранить возможные ошибки в процессе проектирования устройства;
- сравнительно низкая стоимость (в пересчете на один логический вентиль);
- возможность последующей реализации проектов ПЛИС для серийного производства в виде заказных СБИС, что снижает их себестоимость.

До недавнего времени, несмотря на все достоинства ПЛИС Xilinx, существовало обстоятельство сдерживающее их применение (особенно недорогих кристаллов при разработке несерийных устройств) – необходимость дополнительных затрат на приобретение пакета программных средств проектирования и программирования. Чтобы устранить это препятствие, фирма Xilinx предоставила разработчикам возможность использование бесплатного ПО – пакет WebPACK™ ISE™ (Integrated Synthesis Environment).

14.1.2. Архитектура ПЛИС

Рассмотрим более подробно архитектуру ПЛИС, простейших микросхем фирмы Xilinx. Основной особенностью ПЛИС является наличие трёх типов элементов, конфигурация которых может изменяться разработчиком при проектировании ЦУУ. Этими элементами являются:

- блоки ввода/вывода (БВВ) (IOB – Input/Output Block),
- конфигурируемые логические блоки (КЛБ) (CLB - Configurable Logic Block),
- межсоединения (Interconnection) (Рис. 14.1).

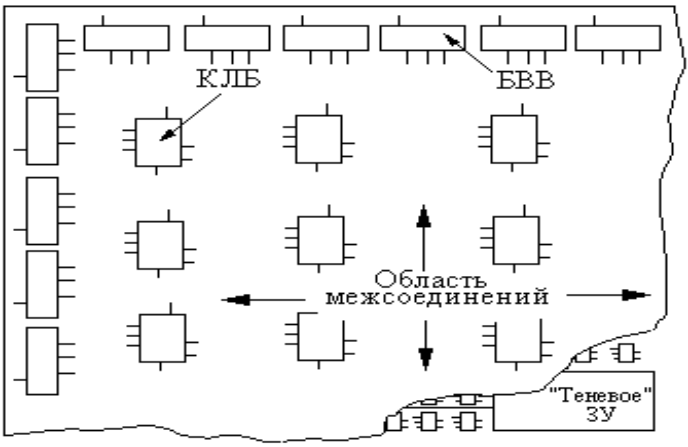


Рисунок 14.1 – Структура кристалла ПЛИС

Общая структура кристалла ПЛИС показана на рисунке, из которой видно, что БВВ располагаются по периферии кристалла, а КЛБ – в виде матрицы в центре, между ними расположены конфигурируемые межсоединения.

Любой отдельный блок ввода/вывода может быть настроен для выполнения функций буферов: входного, выходного, с тремя состояниями, с запоминанием и других, и обеспечения требуемого вида сопряжения с внешними схемами. Конфигурируемые логические блоки предназначены для выполнения простых логических функций от нескольких переменных, а также функций триггера. Цепи межсоединений служат для формирования сложных логических функций и построения узлов, состоящих из многих КЛБ и БВВ. Логические функции ПЛИС и межсоединения определяются данными, хранящимися во внутренних статических запоминающих элементах («теневом» ЗУ), схема которых приведена на рисунке 14.2, а схема реализации логической функции – на рисунке 14.3.

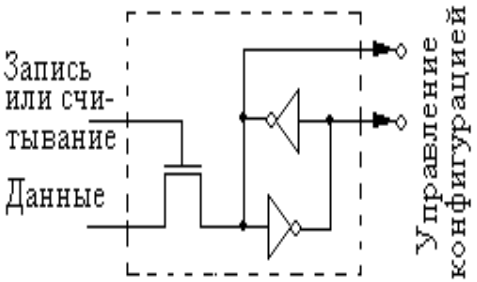


Рисунок 14.2 – Элемент памяти «теневого» ЗУ

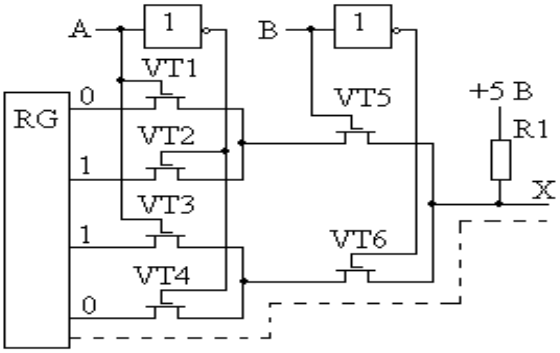


Рисунок 14.3 – Реализация функции на МДП - транзисторах в «теневом» ЗУ

Для реализации требуемой функции в регистр RG (элементы памяти «теневого» ЗУ) записывается нужная информация. Например, для функции «Сумма по модулю два» это 0110. Транзисторы VT1…VT6 переводятся в проводящее состояние напряжением логической 1 на затворе и подключают к выходу X соответствующие разряды регистра. Если A=0 и B=0, то открыты транзисторы VT4 и VT14.В этом случае сигнал на выходе определяется состоянием четвёртого разряда регистра (на рисунке этот путь показан пунктиром), т. е. X = 0.

Полная таблица истинности для данного состояния регистра выглядит следующим образом.:

A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

Из таблицы следует, что схема в этом случае реализует функцию:

$$X = \overline{A}B + A\overline{B}$$

что соответствует функции «Сумма по модулю два». Для получения другой функции следует записать в регистр RG другую информацию.

Программируемые межсоединения в ПЛИС позволяют объединять входы и выходы любых БВВ и КЛБ. Все межсоединения представляют собой сетку вертикальных и горизонтальных металлических сегментов, в местах пересечения которых расположены программируемые переключающие точки (транзисторы – англ. Physical Interconnect Point (PIP) – точка физического межсоединения). PIP позволяют реализовать любой требуемый маршрут цепи и получить для критических цепей задержку менее 0,1 нс. Для разводки по всему кристаллу сигналов с минимальной задержкой служат так называемые длинные линии (ДЛ – англ. LL – Long line) и тактовые буферы. На кристаллах многих ПЛИС имеется внутренний быстродействующий инвертирующий усилитель, позволяющий с помощью внешнего кварцевого резонатора и двух резисторов создавать кварцевый генератор, используемый в проекте. Схема генератора активизируется в начале загрузки конфигурации, что позволяет стабилизировать генератор. Реальное внутреннее подключение генератора задерживается до завершения загрузки конфигурации.

Отличительной особенностью ПЛИС является возможность перепрограммирования функций КЛБ, БВВ и межсоединений путём перезагрузки во внутреннее («теневое»)

ЗУ микросхемы (МС) информации о её конфигурации. Это позволяет получать разные устройства на одном и том же кристалле ПЛИС в динамическом режиме, т. е. в течение малого времени и во время работы МС в составе устройства. Автоматическую загрузку информации о конфигурации обеспечивают специальные схемы на кристалле ПЛИС. Исходные данные о конфигурации могут находиться в ПЗУ, установленном на печатной плате рядом с ПЛИС, или в файле на диске.

14.1.3. Эксплуатационные параметры ПЛИС

Гибкая архитектура ПЛИС позволяет создавать различные управляющие устройства с широким спектром возможных параметров по быстродействию, температурному диапазону, напряжению питания, потребляемой мощности и т. п.

Быстродействие ПЛИС обеспечивается применением современной технологии: частота переключения одного триггера в счётном режиме для разных модификаций МС может составлять от 50 до 320 МГц, а время выработки логической функции на КЛБ – от 15 до 1 нс.

Температурный диапазон работы ПЛИС зависит от исполнения: МС в коммерческом исполнении работоспособны при температурах от 0 до +70 °С; в промышленном – от –40 до +85 °С; в военном – от –55 до +125 °С.

Напряжение питания в активном режиме составляет +5,0 или +3.3 В, в пассивном режиме +2.3 В (без потери конфигурации).

Потребляемая мощность ПЛИС, как и всех МС на КМОП - структурах, существенно зависит от типа МС и её номинального напряжения питания, частоты переключения элементов и сложности схемы. Для реальных схем потребляемая мощность одной ПЛИС в динамическом режиме составляет от 0.1 до 4 Вт и более. В статическом режиме, т. е. когда элементы не переключаются, потребляемая мощность ПЛИС составляет единицы милливатт.

14.1.4. Построение устройств на ПЛИС

К входам и выходам ПЛИС подключаются другие МС, тумблеры, элементы индикации и т. д., составляющие другую часть разрабатываемого устройства.

Так как при выключенном питании в ПЛИС не содержится полезной информации, то основная проблема заключается во вводе в ПЛИС информации о требуемой конфигурации. Существует несколько способов ввода конфигурации в ПЛИС, различающихся количеством задействованных выводов, сложностью управления и т. п.

14.1.5. Принцип действия ПЛИС

В простейшем случае процесс конфигурирования ПЛИС осуществляется следующим образом. Предварительно программа конфигурации длиной несколько тысяч бит, автоматически получаемая в результате проектирования устройства, заносится в МС ПЗУ. Затем ПЗУ устанавливается на печатную плату рядом с ПЛИС и соединяется с ней по определённой схеме. После включения питания ПЛИС сама переписывает из ПЗУ в своё «теневое» ЗУ информацию о конфигурации и начинает выполнять заданные при проектировании функции.

Время, необходимое для загрузки программы конфигурации в ПЛИС, зависит от объёма их «теневого» ЗУ и частоты тактового сигнала CCLK, которая, как правило, не должна превышать 2 МГц. Типичное время загрузки 1 бита программы конфигурации составляет 1 мкс. Время конфигурирования составляет от нескольких единиц до нескольких сотен миллисекунд для разных ПЛИС и разных режимов их конфигурирования.

14.2. Архитектура ПЛИС серии XC2000Xilinx

Для примера рассмотрим ПЛИС серии XC2000, который состоят из двух семейств XC2000 с номинальным напряжением питания – 5 В и XC2000L – 3.3 В. Микросхемы этих семейств (XC2064, XC2018 и XC2064L, XC2018L) имеют одинаковую архитектуру и различаются лишь параметрами, зависящими от напряжения питания (потребляемой мощностью, быстродействием, уровнями входных и выходных сигналов). Некоторые параметры МС этой серии приведены в таблице 14.1.

Таблица 14.1 – Параметры ПЛИС серии XC2000

ПЛИС	Число вентилей 2И-НЕ	Число логических блоков	Максимальное число доступных входов/выходов	Объём памяти конфигурации, бит
XC2064	600-1000	64	58	12038
XC2018	1000–1500	100	74	17838

14.2.1. Общая структура ПЛИС

Общая структура ПЛИС серии XC2000 показана на рисунке 14.4.

Микросхема XC2064 содержит 64 КЛБ (матрица 8x8), XC2018 – 100 КЛБ (матрица 10x10). В обозначении микросхемы XC2064 последние две цифры соответствуют количеству КЛБ. Начиная с МС XC2018, последние две цифры соответствуют максимальной эквивалентной ёмкости, а именно 1800 вентилей (реальная ёмкость для XC2018 оказалась меньше) (см. таблицу 14.1).

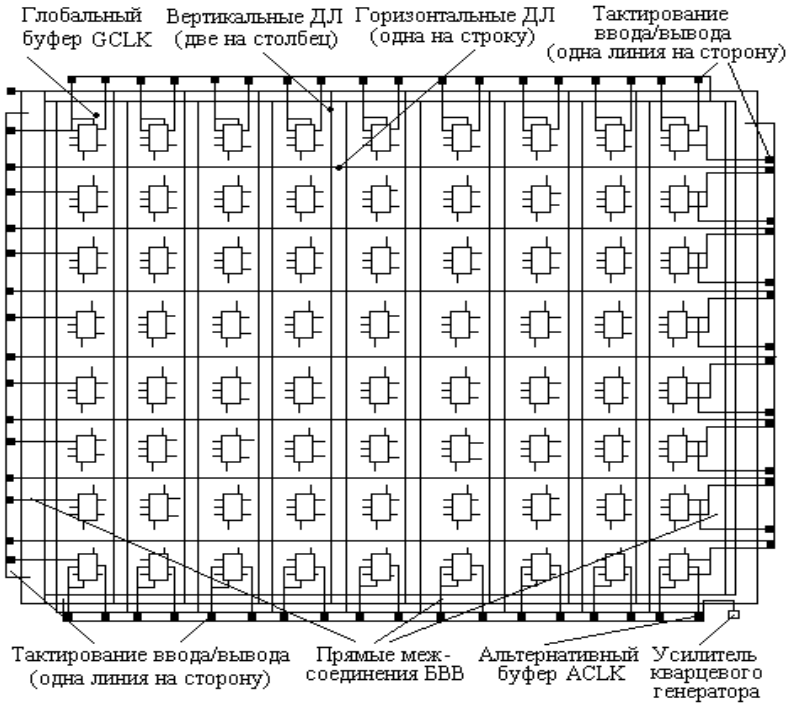


Рисунок 14.4 – Структура ПЛИС серии XC2000 фирмы Xilinx (XC2064)

14.2.2. Блок ввода/вывода (БВВ)

Служит для выбора пользователем конфигурации сопряжения внешних выводов корпуса МС с внутренней частью ПЛИС. Большинство выводов МС связано с конфигурируемыми БВВ, за исключением выводов питания, «земли» и некоторых других. Располагаются БВВ по всем четырём сторонам кристалла ПЛИС.

На рисунке 14.5 показана структура БВВ ПЛИС серии XC2000. Каждый БВВ включает программируемый входной канал (через вывод 1) и программируемый выходной буфер. Входной буфер обеспечивает согласование внешних сигналов,

поступающих на выводы корпуса ПЛИС, и внутренних логических сигналов.

Порог входного буфера можно запрограммировать на совместимость с уровнями либо ТТЛ – схем (пороговое напряжение 1.4 В), либо схем на КМОП –структурах (пороговое напряжение 2.2 В).

Буферизованный входной сигнал поступает на вход данных триггера и на один из входов программируемого мультиплексора. Выходной сигнал триггера поступает на другой вход мультиплексора. Наличие программируемого мультиплексора даёт возможность пользователю выбрать либо прямой ввод сигнала, либо ввод с запоминанием на триггере. Расположенные входы на одной стороне кристалла БВВ, используют общие синхронизирующие импульсы, поступающие на вход К. Триггеры сбрасываются в 0 в процессе установления конфигурации, а также входом RESET, имеющим активный нижний уровень (при переходе этого сигнала в логический 0).

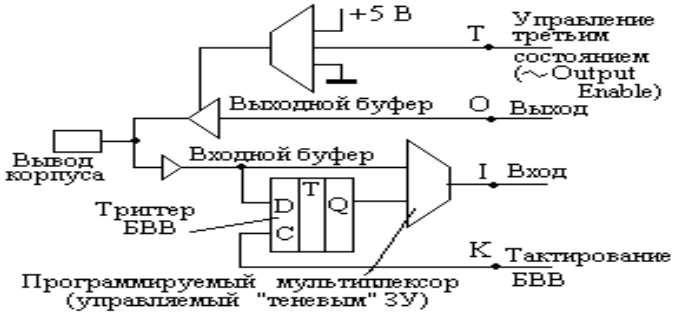


Рисунок 14.5 – Блок ввода/вывода ПЛИС серии XC2000

Вывод 0 является источником данных для выходного буфера. Каждый БВВ может содержать простой буфер или же буфер с тремя состояниями. Пользователь может удалить выходной буфер из БВВ. Тогда данный вывод корпуса будет работать только как вход.

Пользователь также может управлять третьим состоянием выходного буфера, подсоединяя к выводу Т, либо специальный

сигнал, либо сам выходной сигнал. Если на вывод T подан сигнал высокого уровня (логическая 1), то выход буфера переходит в третье состояние и вывод корпуса МС имеет большой импеданс. То есть активным значением управляющего сигнала на выводе T является логическая 1, что и отражено в обозначении выходного буфера. В обозначения буферов ряда других МС аналогичный сигнал носит название OE (Output Enable – разрешение выхода) и его действующее значение, как правило, соответствует логическому 0.

В случае соединения цепей управляющего сигнала T и вывода O одного и того же БВВ выходной буфер этого блока становится эквивалентным каскаду с открытым коллектором (стоком).

Выходные буферы в БВВ могут вырабатывать сигналы, соответствующие по уровням схемам на КМОП – структурах. Нагрузочная способность БВВ по току равна 4 мА; номинальная емкостная нагрузка 50 пф. Каждый блок имеет также схемы защиты от электростатического пробоя и режима защёлкивания по входу.

Имена БВВ для МС во всех типах корпусов, кроме PGA, состоят из бук -
вы P и порядкового номера вывода корпуса (P1, P2…). Если БВВ не подсоединён к выводу МС, то его имя начинается с буквы U. Для корпуса PGA имя БВВ соответствует номеру вывода МС (A1, F5, U7, W10…). Один и тот же кристалл может размещаться в корпусах как с большим количеством выводов, что делает доступными все БВВ, так и в корпусах с малым количеством выводов, что уменьшает габаритные размеры МС.

14.2.3. Конфигурируемые логические блоки

Расположены в виде матрицы в центре кристалла ПЛИС. Строки и столбцы матрицы обозначаются буквами A, B, C, D и т. д., которые используются для задания имени (адреса) блока, например, блоки AA, AD, HH. Разработчик аппаратуры может присвоить блоку новое имя для обозначения, например узла

схемы, занимаемого этим блоком. В этом случае блок будет иметь два имени.

На рисунке 14.6 показана структура КЛБ. В КЛБ можно выделить три части:

- комбинаторный узел, служащий для выработки логических функций;
- запоминающий элемент (триггер) для хранения значения одной из логической функции;
- узел соединений (мультиплексор), предназначенный для внутренних соединений и управления.

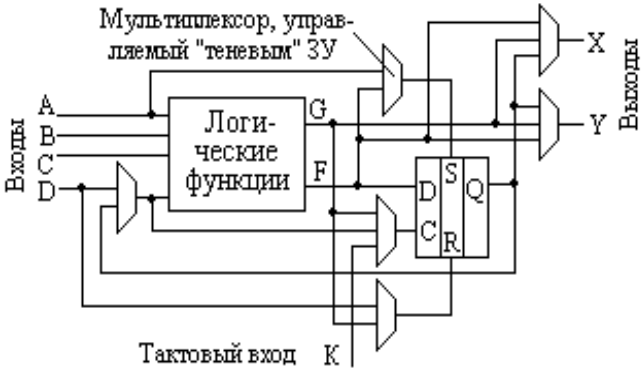


Рисунок 14.6 – Конфигурируемый логический блок ПЛИС серии XC2000

Каждый КЛБ имеет два выхода X и Y, четыре входа общего назначения A, B, C и D, а также специальный вход синхронизации (тактовый вход) K.

В МС серии XC2000 можно генерировать любую логическую функцию четырёх переменных (максимум), т. к. для этого используется 16-разрядный регистр «теневого» ЗУ. Время распространения сигнала через комбинаторный узел не зависит от генерируемой функции. Каждый блок может выполнять любую функцию четырёх переменных или любые две функции трёх переменных (см. рисунок 13.9). Переменные для логической функции могут поступать с четырёх входов A, B, C, D и выхода Q триггера.

Запоминающий элемент КЛБ может быть сконфигурирован либо как D-триггер, запоминающий состояние на входе данных по переднему фронту тактового сигнала (FF-триггер), либо как триггер-защёлка (Latch), срабатывающий по заднему фронту. Полярность тактового сигнала триггера может программироваться (то есть тактовый сигнал может иметь вид положительного импульса, так и отрицательного импульса).

Узел соединений с помощью программируемых мультиплексоров обеспечивает различные пути прохождения сигналов от входов до выходов КЛБ, позволяя тем самым изменять общую конфигурацию.

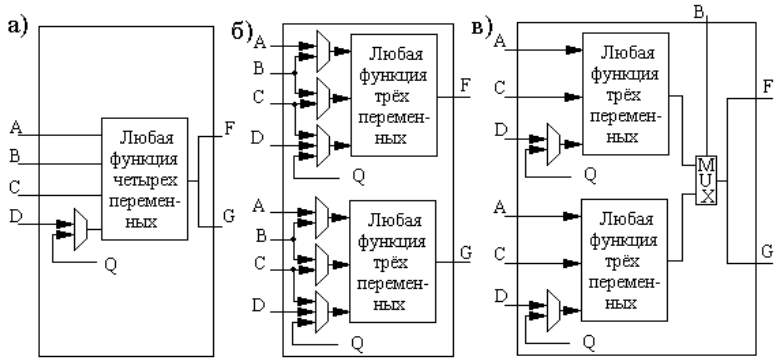


Рисунок 14.7 – Варианты конфигурации функций в МС серии XC2000: а) – одна функция четырёх переменных; б) – две функции трёх переменных; в) – динамический выбор одной из двух функций трёх переменных

14.3. Программируемые межсоединения и кварцевый генератор ПЛИС серии XC2000 фирмы Xilinx

14.3.1. Программируемые межсоединения

В ПЛИС позволяют объединять входы и выходы любых БВВ и КЛБ и получать для некоторых целей задержку менее 0.1 нс. Все межсоединения представляют собой сетку вертикальных и горизонтальных металлических сегментов, в месте пересечения которых расположены транзисторы, выполняющие роль

программируемых связных точек (Phisical Interconnect Points, PIP), что даёт возможность реализовать требуемый маршрут.
Программируемые межсоединения подразделяются на межсоединения общего назначения, прямые соединения и длинные линии.

Межсоединения общего назначения (General Purpuse Interconnect) используются для передачи сигналов между любыми блоками (см. рисунок 14.8). Они содержат переключающие матрицы (Switch Matrix) для изменения направления и разветвления сигнала.

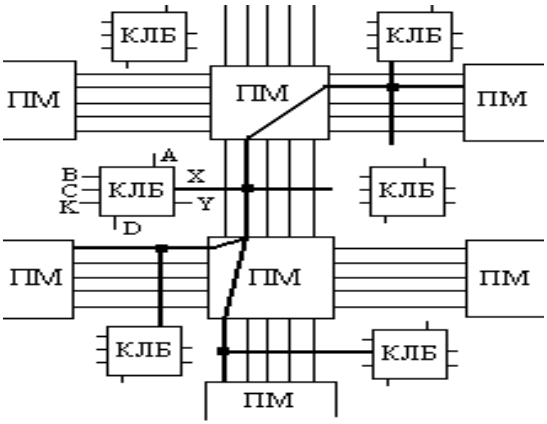


Рисунок 14.8 – Конфигурация межсоединений общего назначения

Однако переключающие матрицы вносят существенную задержку в распространение сигнала.

Прямые линии (Direct Interconnect) обеспечивают практически нулевую задержку между соседними логическими блоками (см. рисунок 14.10), а также между БВВ и близлежащими к ним КЛБ (см. рис. 14.6).

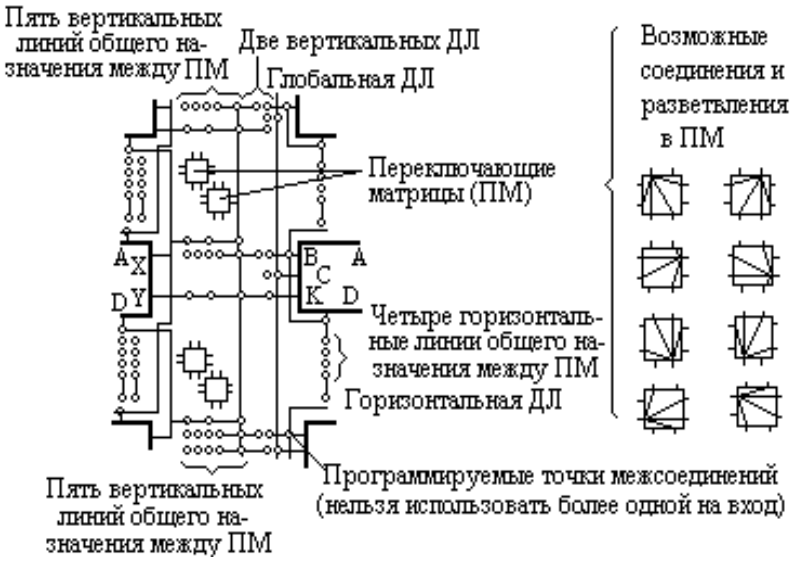


Рисунок 14.9 – Структура переключающих матриц ПЛИС

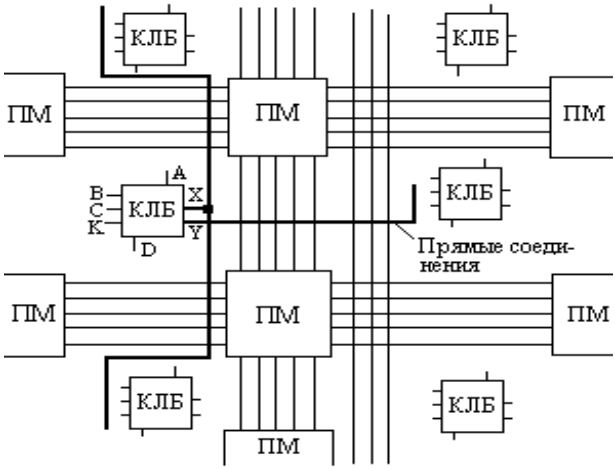


Рисунок 14.10 – Прямые соединения логических блоков ПЛИС

Длинные линии (Long Line), проходящие мимо ПМ (см. рисунок 14.11), также позволяют подводить ко всем логическим блокам сигналы с практически нулевой задержкой.

В ПЛИС серии XC2000 имеются три вида ДЛ:

- горизонтальные (по одной на каждую строку КЛБ);
- вертикальные (по две на каждый столбец КЛБ);
- глобальная (одна, проходящая возле каждого столбца КЛБ).

В верхнем левом углу кристалла (см. рис. 14.4) возле логического блока AA расположен глобальный тактовый буфер (Global Buffer, GCLK), предназначенный для подачи через ДЛ одного сигнала к входам B и K всех логических блоков.

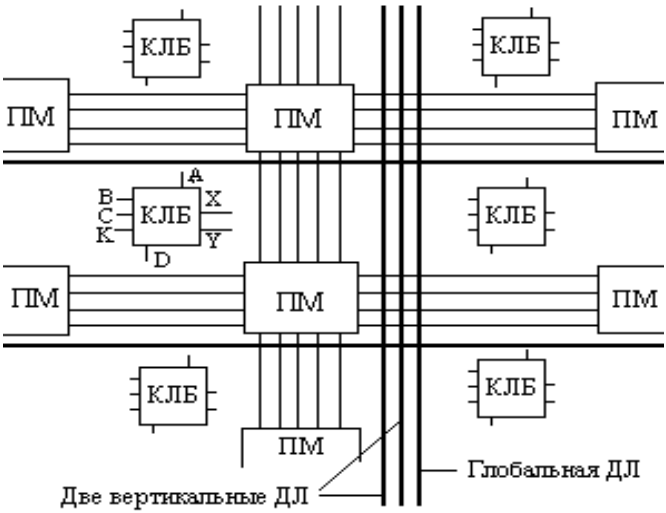


Рисунок 14.11 – Конфигурация длинных линий ПЛИС

Второй альтернативный (Alternate Buffer) тактовый буфер ACLK, расположенный в правом нижнем углу кристалла, также позволяет через ДЛ подвести один сигнал ко всем логическим блокам (к выводам B, C и K). Время распространения сигнала по ДЛ к любому КЛБ равно ~0.

Длинные линии могут подключаться к выходам КЛБ по отдельности, создавая быстродействующие локальные тактовые цепи, проходящие вдоль одного столбца или строки КЛБ. К горизонтальной ДЛ можно подключить выход X логического блока, к вертикальной – выход Y.

14.3.2. Кварцевый генератор

Для создания кварцевого генератора (Crystal Oscillator), используемого в проектируемом устройстве, на кристалле ПЛИС серии XC2000 фирмы Xilinx имеется внутренний быстродействующий инвертирующий усилитель, который связан с двумя выводами МС, расположенными в нижнем правом углу кристалла и имеющими имена XTAL1 и XTAL2 (см. рис. 14.12 и таблицу 14.2). Рекомендуемые параметры компонентов, приведённых на рис.14.12, составляют R1 = 0.5-1 МОм; R2 = 0-1 kОм; C1, C2 = = 10-40 пф; BQ1 = 1-20 МГц.

Таблица 14.2 – Номера выводов для кварцевого генератора ПЛИС серии XC2000

Тип корпуса	48 DIP	68 PLCC	68 PGA	84 PLCC	84 PGA
XTAL1	33	46	J10	56	K11
XTAL2	30	43	L10	53	L11

Тактовый сигнал от кварцевого генератора снимается с альтернативного тактового буфера ACLK. Схема генератора становится активной в начале загрузки конфигурации, что позволяет стабилизировать генератор к моменту её завершения. Реальное внутреннее подключение генератора задерживается до завершения загрузки конфигурации.

Если кварцевой генератор не используется, то БВВ XTAL1 и XTAL2 могут быть использованы, как и любые другие конфигурируемые БВВ.

Выше были рассмотрены основные архитектурные особенности и принципы построения ПЛИС семейства XC2000. Здесь не приводится информация о назначении контактов для различных корпусов, потребляемой мощности и т. д. Это связано

с тем, что данная информация легко доступна на CD Xilinx, так и в Internet.

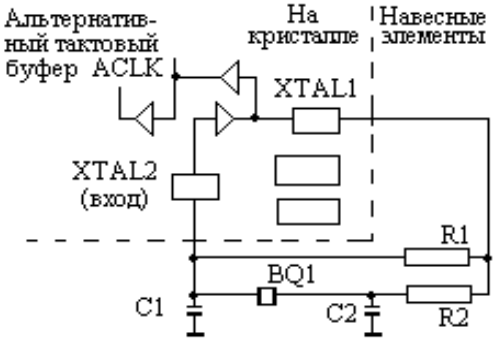


Рисунок 14.12 – Кварцевый генератор

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Максфилд К. Проектирование на ПЛИС. Курс молодого бойца. - М.: Изд. дом «Додэка-XXI», 2007. - 408 с.
2. M. Morris Mano, Michael D. Ciletti. Digital Design With an Introduction to the Verilog HDL. -Pearson; 5th edition, 2012. -576p.
3. Харрис Д. и Харрис С. Цифровая схемотехника и архитектура компьютера / пер. с англ. / - второе издание. Издательство Morgan Kaufman © English Edition 2013
4. Charles Roth, Lizy K. John, Byeong Kil Lee. Digital Systems Design Using Verilog. - Cengage Learning; 1st edition, 2015. -608p.
5. Наваби З. Проектирование встраиваемых систем на ПЛИС / пер. с англ. / - М.: ДМК Пресс, 2016. -464 с.
6. Ronald Tocci, Neal Widmer, Gregory Moss. Digital Systems Principles and applications. -Pearson; 12th edition,2016. -1004 p.
7. Cem Unsalan, Bora Tar. Digital System Design with FPGA: Implementation Using Verilog and VHDL. McGraw Hill; 1st edition, 2017. -400p.
8. Жармакин Б. К. Программирование элементов цифровой электроники на языке VHDL. -Алматы.: Эпиграф, 2018. -180с.
9. M. Rafiquzzaman, Steven A. McNinch. Digital Logic: With an Introduction to Verilog and FPGA-Based Design. - Wiley, 2019. -444p.
10. Barun Raychaudhuri. Electronics: Analog and Digital. -Cambridge University Press; New edition, 2022. -500p.