Министерство образования Московской области
Государственный университет «Дубна»
Филиал «Протвино»
Кафедра автоматизации технологических процессов и производств



# Проектирование цифровых устройств на базе современных инструментальных средств

УЧЕБНОЕ ПОСОБИЕ

Рекомендовано учебно-методическим советом университета «Дубна» в качестве учебного пособия для студентов, обучающихся по направлениям подготовки «Автоматизация технологических процессов и производств», «Информатика и вычислительная техника», «Физика» (бакалавриат)

Дубна
2021

УДК 004.3, 004.438
ББК
Е

Р е ц е н з е н т:
доктор физ.-математических наук, советник по электронике директора
Института физики высоких энергий им. А.А. Логунова НИЦ «Курчатовский институт»
*В.А. Сенько*

А в т о р ы :
Евсиков А. А., Коковин В. А., Леонов А. П., Сытин А. Н.

**Евсиков, А. А.**

Е Проектирование цифровых устройств на базе современных инструментальных средств : учебное пособие / А. А. Евсиков, В. А. Коковин, А. П. Леонов, А. Н. Сытин. — Дубна: Гос. ун-т «Дубна», 2021. — 86 [2] с.

ISBN 978-5-89847-

Пособие содержит материал, способствующий изучению принципов проектирования и функционирования логических элементов, комбинационных и последовательностных устройств на базе современных программируемых логических интегральных схем с использованием интегрированного пакета Quartus Prime. В пособии даны основы алгебры логики (булевой алгебры), принципов минимизации логических функций. Подробно рассмотрена схемотехника цифровых устройств на базе цифровых матриц. Задания, представленные в пособии, охватывают синтез большинства типовых схем (от счетчиков/регистров до цифровых автоматов) и позволяют получить практические навыки программирования на языке SystemVerilog.

Предназначено для использования при изучении студентами дисциплин «Цифровая электроника» направлений «Автоматизация технологических процессов и производств» и «Физика», «Основы электротехники и электроники систем управления» направления «Информатика и вычислительная техника».

УДК 004.3, 004.438
ББК

ISBN 978-5-89847-

# Оглавление

## Введение

Бурное развитие вычислительных машин после Второй мировой войны, открытие и создание транзистора, прогресс полупроводниковых технологий и появление интегральных микросхем привели к появлению цифровых технологий. Сегодня трудно найти область деятельности человека, в которой отсутствует влияние цифровых технологий. Впервые цифровые технологии были использованы при создании систем сбора и обработки данных в физических экспериментах и системах контроля и управления технологическими процессами. Накопленный опыт использования цифровых технологий широко применяется в финансово-торговых структурах, медицине, транспорте, образовании и других областях деятельности человека. Невысокая стоимость элементной базы, низкие мощностные требования в сочетании с высокой процессорной производительностью позволяют создавать вычислительные системы как для научных, так и для промышленных применений.

Важным этапом в развитии цифровых технологий явилось создание полупроводниковых технологий при производстве интегральных микросхем, промежуточным итогом которого явилось появление микропроцессора. Это послужило основой для перехода к новому этапу развития вычислительной техники, приведшему к разработке персонального компьютера, определившего на долгие годы идеологию архитектуры вычислительных устройств подобного рода. Если раньше ЭВМ использовалась для различных расчетов как средство общего пользования, то теперь появилась возможность огромному числу пользователей решать задачи вычисления и управления в индивидуальном порядке. Это привело к резкому росту системного и прикладного программного обеспечения для персональных компьютеров. В первую очередь следует упомянуть об операционных системах с расширенными функциональными возможностями, которые позволяют упростить процессы ввода-вывода и управления данными, человеко-машинного взаимодействия, управления ресурсами внешней и внутренней памяти, согласованного управления работой всех аппаратно-программных средств персонального компьютера. Сравнительно низкая цена персонального компьютера и небольшие накладные расходы в процессе эксплуатации позволили разработать и использовать большой набор пакетов прикладных программ для индивидуальной работы инженерами различных профессий. Такими пакетами являются программные системы CAD (Computer-Aided Design), CAE (Computer-Aided Engineering) и т.д.

Появление микропроцессоров, при построении систем автоматизации в промышленности, дало толчок к переходу от релейных устройств к программируемым логическим контроллерам (ПЛК). Создание ПЛК заставило стандартизовать программное обеспечение различных производителей, что явилось новым подходом в технологиях программирования.

Дальнейшее развитие цифровых технологий определило появление в обществе всемирной информационной компьютерной сети (интернет), которая базируется на локальных вычислительных сетях различного назначения, объединенных глобальной сетью. Благодаря имеющимся приложениям в глобальной сети, таким как электронная почта, всемирная паутина WWW (World Wide Web), а также различного рода коммуникационным средствам (messengers), появилась возможность резкого ускорения и увеличения обмена информацией.

Все перечисленное выше компоненты и системы сегодня широко используются при создании автоматизированных систем управления технологическими процессами (АСУ ТП) различного назначения в промышленности, науке, медицине, образовании и экономике. В качестве программного обеспечения работы АСУ ТП широко применяются SCADA (**S**upervisory **C**ontrol **A**nd **D**ata **A**cquisition) – системы, предназначенные для разработки и обеспечения работы в реальном времени систем сбора, обработки, отображения и архивирования информации.

Архитектура систем управления (СУ), используемых при автоматизации технологических установок, в своем развитии прошла через определенные этапы. Эти этапы во многом определялись уровнем интеграции, стоимостью элементной базы и программного обеспечения. Архитектура СУ в большей степени зависела от возможностей управляющих вычислительных устройств и интерфейсов связи.

Современные СУ технологическими установками строятся на основе промышленных интерфейсов, мощных вычислительных систем и интеллектуальных датчиков. Можно условно выделить три модели построения систем управления, при этом каждая из моделей имеет свои недостатки и свои достоинства:

- Модель центрального управляющего контроллера, связанного цифровыми сетями с датчиками и исполнительными механизмами. В этой модели все управление выполняется на уровне центрального контроллера, что обуславливает высокую детерминированность реализации алгоритма. К недостаткам такой архитектуры можно отнести трудность масштабируемости и высокие требования к центральному контроллеру.

▪ Модель систем с распределенным управлением, в основу которой положен механизм децентрализованного управления (Distributed Control Systems – DCS) [1]. Данная СУ технологическим процессом позволяет весь алгоритм разбить на небольшие части (законченные функциональные блоки – примитивы) и реализовывать с помощью недорогих и маломощных контроллеров. Связь между отдельными контроллерами выполняется по стандартным сетям. Недостаток – низкоуровневое программирование контроллеров.

▪ Модель СУ, разрабатываемых на основе стандарта МЭК-61499 [2]. Это наиболее перспективная для распределенных систем управления модель. В основе ее лежит концепция распределенных управляющих приложений, реализуемых на функциональных блоках. Программирование выполняется для всей системы на верхнем уровне и загружается в распределенные контроллеры по сети Ethernet. Контроллеры взаимодействуют через потоки данных и управляющих событий.

Современный уровень развития микроэлектроники и технологий управления позволяет создать комбинированную модель управления из представленных выше признаков, отвечающую наиболее полно поставленной задаче.

Существенным недостатком перечисленных *моделей систем управления* является то, что они, как правило, реализуются на контроллерах (вычислителях), построенных с использованием классических моделей вычислений. В настоящее время можно выделить два основных подхода при разработке *моделей вычислений*.

• Во-первых, это традиционная архитектура с использованием вычислителя (микропроцессора), построенного на базе модели фон Неймана [3], которую в литературе принято называть **control flow** (поток команд). Основным недостатком данной модели, с точки зрения быстродействия, является последовательное выполнение команд, реализующих алгоритм. Существует много различных дополнений к данной модели (многоконвейерность обработки команд, многоядерность процессоров и т.д.), которые значительно повысили производительность вычислителей.

• Во-вторых, это реализация архитектуры вычислителей на основе модели вычислений **data flow** (поток данных), предложенной в работах Д. Денниса, например [4]. В этом случае реализуемый алгоритм «зашит» в аппаратуре системы управления в виде требуемых вычислительных и логических операторов, что позволяет распараллеливать различные процессы и неограниченно масштабировать.

Недостатком модели **data flow** до недавнего времени оставалась реализация эффективной по производительности, но узкоспециальной по назначению задачи. При смене алгоритма задачи приходилось модернизировать аппаратную часть системы управления. С появлением технологии программируемых логических интегральных схем (ПЛИС) ситуация изменилась. Появилась возможность реконфигурировать аппаратную реализацию алгоритма удаленно и неограниченное число раз. Кроме того, разработку можно выполнять на языках описания аппаратуры (Verilog, VHDL), что повысило производительность разработки и дало возможность выполнять функциональную и временную симуляцию проекта. Современные ПЛИС в своем составе содержат средства для эффективной реализации модели вычислений **data flow** – цифровые сигнальные процессоры (DSP), которые позволяют вести разработку и реализацию контуров управления высокопроизводительных силовых преобразователей, широко используемых при автоматизации технологических процессов. Реализация функциональных блоков методом **data flow**, лежащих в основе последних двух моделей СУ, представляется перспективной с точки зрения повышения быстродействия обработки информации и отработки заложенных алгоритмов.

Созданные ПЛИС находят все большее применение при разработке цифровых устройств самого различного назначения. В ПЛИС заложены возможности, которые позволяют реализовать на ее основе интегральные схемы с любой функцией цифровой логики. В результате эволюции развития ПЛИС к настоящему времени разработаны и применяются микросхемы, которые можно разбить на два больших класса: ПЛИС с архитектурой CPLD – это сложные программируемые логические приборы (Complex Programmable Logic Device) и программируемые пользователем вентильные матрицы FPGA. FPGA – это ПЛИС, которые имеют большой набор базовых блоков: настраиваемые логические элементы с помощью таблицы перекодировки, блоки сложения-умножения DSP (Digital Signal Processing), PLL (Phase-Locked Loop) для деления и умножения частоты, достаточно большой объем энергозависимой внутренней памяти и функционал для загрузки конфигурации с внешней энергонезависимой памяти. Более подробно с особенностями архитектуры и техническими характеристиками FPGA, например семейства Cyclone IV фирмы Intel, можно познакомиться в [5]. CPLD состоит из нескольких макроячеек, расположенных на одном кристалле. Каждая макроячейка соединена с блоками ввода-вывода, осуществляющими формирование необходимого вида входов или выходов для работы с внешними схемами. Кроме того, все макроячейки и блоки ввода-вывода связаны между собой внутренними параллельными шинами. CPLD содержит меньше базовых элементов, чем

FPGA, но является более быстродействующей. ПЛИС данного типа содержит энергонезависимую конфигурационную память прямо на кристалле, но имеет ограниченное число циклов конфигурирования.

Применение ПЛИС позволяет сократить процесс проектирования и отладки цифровых устройств. Использование ПЛИС обеспечивает максимальную гибкость при необходимости модификации аппаратуры. Проектирование цифровых устройств с применением ПЛИС имеет свои особенности. Для разработки проектов цифровых устройств используются специально созданные системы автоматизированного проектирования, в которых для ввода могут использоваться графические языки описания схем или универсальные текстовые редакторы, позволяющие использовать языки высокого уровня. Обязательным этапом является функциональное и временное моделирование, во время которого проверяется правильность разработанной схемы. Трансляция введенной схемы в битовую загрузочную последовательность часто осуществляется автоматически без вмешательства пользователя. Для программирования микросхем применяются достаточно простые устройства, в том числе широко используемый в вычислительной технике четырехпроводной интерфейс JTAG (Joint Test Action Group) [6], который позволяет не только достаточно просто производить загрузку ПЛИС, но и осуществлять тестирование микросхемы.

Развитие технологии элементной базы цифровой схемотехники позволило создать ПЛИС и, в большинстве случаев, отказаться от специализированных микросхем, минимизировать объем аппаратуры автоматизированных систем управления и повысить ее функциональность. Возрастание сложности и быстродействия интегральных схем, в частности ПЛИС, дает возможность иметь в аппаратуре все большее число функциональных цифровых компонентов, схемотехнически реализовывать всё более многообразные и сложные функции. Разработка цифровых узлов на таких емких (миллионы вентилей) и быстрых схемах, вполне доступных по цене, позволяет проектировать унифицированную аппаратуру различного назначения. Высокая надежность работы ПЛИС позволила, например, реализовать электронику общей таймерной системы управляющей и диагностирующей аппаратуры, системы быстрого вывода ускорительного комплекса Государственного научного центра «Институт физики высоких энергий» на ПЛИС [7; 8]. Инструментальные средства разработки нового поколения, такие как QUARTUS Prime фирмы Intel [9], дают возможность проектировать функционально законченные блоки. Эти блоки представляют собой так называемые мегафункции, которые можно использовать без изменения в различных проектах, содержащих ПЛИС.

Ведущие фирмы, производители ПЛИС, распространяют бесплатные системы проектирования, которые хотя и имеют ограничения по мощности и функциональному назначению по сравнению с платными, тем не менее, позволяют разрабатывать проекты для многих практических приложений. Целью данного учебного пособия является освоение элементов математического аппарата цифровой схемотехники и алгебры логики (булевой алгебры), выполнение анализа и синтеза базовых комбинационных и последовательностных устройств, изучение основных приемов и методов создания проектов в интегрированном пакете Quartus Prime и наблюдение на экране осциллографа сигналов цифровой схемы в контрольных точках. Данное пособие предназначено для студентов старших курсов направлений бакалавриата: «Автоматизация технологических процессов и производств», «Информатика и вычислительная техника», «Физика». Лабораторные работы выполняются на учебных стендах, в составе которых используются современные ПЛИС классов CPLD и FPGA. При выполнении практических и лабораторных работ по курсам, связанным с изучением цифровой электроники, используется бесплатный интегрированный пакет Quartus Prime Lite Edition фирмы Intel.

# Глава 1. Алгебра логики (булева алгебра). Анализ и синтез комбинационных схем

Алгебра логики (булева алгебра) – это раздел математики, изучающий высказывания, рассматриваемые со стороны их логических значений (истинности или ложности) и логических операций над ними. Алгебра логики позволяет закодировать любые утверждения, истинность или ложность которых нужно доказать, а затем манипулировать ими подобно обычным числам в математике.

## 1.1. Числовые коды

В цифровых устройствах обрабатывается и хранится не только числовая, но и алфавитно-цифровая информация, содержащая коды цифр, букв, математических и других символов.

Совокупность всех символов представляет собой алфавит. Для представления каждого символа в цифровом устройстве ему соответствует группа двоичных разрядов, являющаяся слогом.

Десятичные цифры 0, 1, 2, …, 9 кодируются двоичными цифрами с помощью различных кодов. При использовании двоично-десятичного кода 8421 каждая цифра десятичного числа представляется в двоичной форме и изображается четырехразрядным двоичным числом (табл. 1).

Таблица 1. **Двоичное кодирование десятичных цифр**

| Десятичная цифра | Код 8421 | Десятичные цифры | Код 8421 |
|---|---|---|---|
| 0 | 0000 | 5 | 0101 |
| 1 | 0001 | 6 | 0110 |
| 2 | 0010 | 7 | 0111 |
| 3 | 0011 | 8 | 1000 |
| 4 | 0100 | 9 | 1001 |

Этот код аддитивен, т.е. сумма весов разрядов двоично-десятичного кода 8421 равна представляемой цифре и является естественным представлением десятичных цифр в двоичной системе счисления. При двоичном кодировании десятичных цифр используются и другие коды (табл. 2).

Таблица 2. **Использование различных кодов при кодировании десятичных цифр**

| Десятичная цифра | Код 8421 | Код 7421 | Код 2421 | Код 2 из 5 | Код с избытком 3 | Код 3а+2 |
|---|---|---|---|---|---|---|
| 0 | 0000 | 0000 | 0000 | 11000 | 0011 | 00010 |
| 1 | 0001 | 0001 | 0001 | 01100 | 0100 | 00101 |
| 2 | 0010 | 0010 | 0010 | 00110 | 0101 | 01000 |
| 3 | 0011 | 0011 | 0011 | 00011 | 0110 | 01011 |
| 4 | 0100 | 0100 | 0100 | 10001 | 0111 | 01110 |
| 5 | 0101 | 0101 | 1011 | 10100 | 1000 | 10001 |
| 6 | 0110 | 0110 | 1100 | 01010 | 1001 | 10100 |
| 7 | 0111 | 1000 | 1101 | 00101 | 1010 | 10111 |
| 8 | 1000 | 1001 | 1110 | 10010 | 1011 | 11010 |
| 9 | 1001 | 1010 | 1111 | 01001 | 1100 | 11101 |

Эти коды имеют свои особенности. Любая кодовая комбинация кодов 7421 и 2 из 5 содержит не более двух единиц, это используется для обнаружения ошибочных комбинаций. В коде 2421 кодовая комбинация, соответствующая любой из десятичных цифр, представляет собой инверсию кодовой комбинации, соответствующей ее дополнению до девяти. Например, пара взаимодополняющих до девяти цифр 2 и 7 соответствует комбинации 0101 и 1010, каждая из которых образуется как инверсия другой. Код 3а+2 тоже обладает свойством дополнения до девяти, что упрощает выполнение в цифровых устройствах операции над десятичными числами [10].

## 1.2. Алгебра логики

В алгебре логики рассматриваются следующие компоненты: **переменные**, которые могут принимать только два значения **0** и **1** (переменные обозначаются латинскими буквами $x$, $y$, $z$, …, а также $x_0$, $x_1$, …, $x_n$, $y_0$, $y_1$, …, $y_n$ и т.д.), **отношения эквивалентности** (равенства «=»), которые удовлетворяют следующим свойствам:

• рефлексивность – $x=x$;

• симметричность – если $x=y$, то $y=x$;

• транзитивность – если $x=y$ и $y=z$, то $x=z$, отсюда следует принцип, если $x=y$, то в любой формуле, содержащей $x$, в место $x$ можно подставить $y$, и в результате будет получена эквивалентная формула;

• три операции: ***дизъюнкция*** – операция ИЛИ, логическое сложение (обозначается знаком ∨ или +), ***конъюнкция*** – логическое умножение (обозначается знаком ∧, или &, или *), ***отрицание*** – инверсия, операция НЕ (обозначается знаком ’ или чертой над переменной, или над элементами 0 и 1, или над операциями с охватом всех переменных входящих в операцию (например, $\overline{X},\overline{Y}$, или $\bar{1},\bar{0}$, или $\overline{X+Y}$, $x'$, $y_1'$)).

### 1.2.1. Аксиомы алгебры логики

Аксиомы или постулаты математической системы – это набор основных утверждений, про которые мы предполагаем, что они справедливы, и из которых можно вывести все другие свойства системы.

$$\left.\begin{array}{l} x=0, \text{ если } x \neq 1 \\ x = 1, \text{ если } x \neq 0 \end{array}\right\} \quad (1) \qquad \left.\begin{array}{l} 1+1=1 \\ 0*0=0 \end{array}\right\} \quad (2)$$

$$\left.\begin{array}{l} 0+0=0 \\ 1*1=1 \end{array}\right\} \quad (3) \qquad \left.\begin{array}{l} 0+1=1+0=1 \\ 1*0=0*1=0 \end{array}\right\} \quad (4)$$

$$\left.\begin{array}{l} \bar{0}=1 \\ \bar{1}=0 \end{array}\right\} \quad (5)$$

Формула (1) утверждает, что в алгебре логики рассматриваются только двоичные переменные. Формулы (2) – (4) определяют операции дизъюнкции и конъюнкции. Формула (5) определяет операцию отрицания.

### 1.2.2. Теоремы и тождества алгебры логики

На основании аксиом алгебры логики можно вывести ряд теорем и законов.

1. Идемпотентные законы
$$\left.\begin{array}{l} x+x=x \\ x*x=x \end{array}\right\}. \quad (6)$$

2. Коммутативные законы
$$\left.\begin{array}{l} x+y=y+x \\ x*y=y*x \end{array}\right\}. \quad (7)$$

3. Ассоциативные законы
$$\left.\begin{array}{l} (x+y)+z=x+(y+z) \\ (x*y)*z=x*(y*z) \end{array}\right\}. \quad (8)$$

4. Дистрибутивные законы
$$\left.\begin{array}{l} x*(y+z)=x*y+x*z \\ x+y*z=(x+y)(x+z) \end{array}\right\}. \quad (9)$$

5. Законы отрицания
$$\left.\begin{array}{l} x+\bar{x}=1 \\ x*\bar{x}=0 \end{array}\right\}, \quad (10)$$

$$\left.\begin{array}{l} 0+x=x \\ 1*x=x \end{array}\right\}, \quad (11)$$

$$\left.\begin{array}{l} 1+x=1 \\ 0*x=0 \end{array}\right\}. \quad (12)$$

6. Законы двойственности (теоремы де Моргана)

$$\left.\begin{array}{l}\overline{x+y}=\bar{x}*\bar{y}\\ \overline{xy}=\bar{x}+\bar{y}\end{array}\right\}. \qquad (13)$$

7. Закон двойного отрицания $\overline{(\bar{x})}=\bar{\bar{x}}=x.$ (14)

8. Законы поглощения

$$\left.\begin{array}{l}x+xy=x\\ x*(x+y)=x\end{array}\right\}. \qquad (15)$$

9. Операции склеивания

$$\left.\begin{array}{l}xy+x\bar{y}=x\\ (x+y)(x+\bar{y})=x\end{array}\right\}, \qquad (16)$$

$$\left.\begin{array}{l}x+\bar{x}y=x+y\\ x(\bar{x}+y)=xy\end{array}\right\}. \qquad (17)$$

### 1.2.3. Логические функции нескольких переменных

Логическая функция – это логическое выражение, состоящее из логических переменных, связанных между собой с помощью операций алгебры. В соответствии с вышеприведенными аксиомами (1) – (5) функция может принимать в зависимости от значений переменных только два значения «0» и «1» логики. Так как область определения любой функции $n$ переменных конечна ($2^n$ точек), она должна быть задана таблицей значений $f(v_i)=0$ или 1, которые она принимает в точках $V_i$, где $i$ = 0, 1, …, $2^n$–1. Такие таблицы называются таблицами истинности. Набор функций, с помощью которого можно представить (выразить) все логические функции, принято называть функционально-полным или базисом.

Определим параметры таких функций: $n$ – число переменных, $m = 2^n$ – число точек определения (или число точек логического пространства), $N = 2^m$ – число всех функций $n$ переменных.

Для функции одной переменной: $n$ = 1, $m$ = 2, $N$ = 4.

Рассмотрим каждую функцию одной переменной:

- $f_0 = 0$ – нулевая функция,
- $f_1 = x$ – функция повторения,
- $f_2 = \bar{x}$ – функция отрицания,
- $f_3 = 1$ – единичная функция.

Для функции двух переменных: $n$ = 2, $m$ = 4, $N$ = 16. Значения всех функций в точках определения представлены в табл. 3.

Таблица 3. **Таблица истинности всех функций двух переменных**

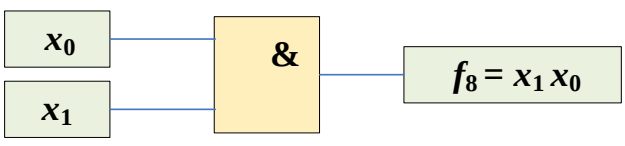
| $Vi$ | $x_1$, $x_0$ | $f_0$ | $f_1$ | $f_2$ | $f_3$ | $f_4$ | $f_5$ | $f_6$ | $f_7$ | $f_8$ | $f_9$ | $f_{10}$ | $f_{11}$ | $f_{12}$ | $f_{13}$ | $f_{14}$ | $f_{15}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 2 | 1 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 3 | 1 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Ниже дано описание основных функций:

***Функция логического умножения (конъюнкция).***

$f_8 = x_1 x_0$ – логическое умножение, описывает работу логического элемента И.

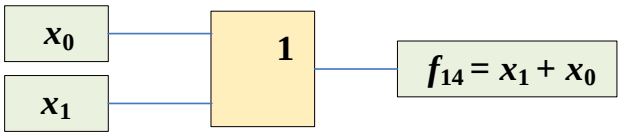
| $V_i$ | $x_1$, $x_0$ | $f_8$ |
|---|---|---|
| 0 | 0 0 | 0 |
| 1 | 0 1 | 0 |
| 2 | 1 0 | 0 |
| 3 | 1 1 | 1 |



***Функция логического сложения (дизъюнкция)***

$f_{14} = x_1 + x_0$ – логическое сложение, описывает работу логического элемента ИЛИ.

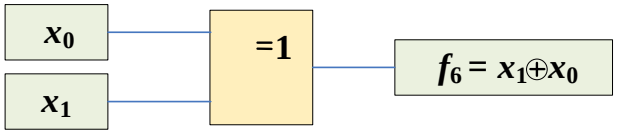
| $V_i$ | $x_1, x_0$ | $f_{14}$ |
|---|---|---|
| 0 | 0 0 | 0 |
| 1 | 0 1 | 1 |
| 2 | 1 0 | 1 |
| 3 | 1 1 | 1 |




***Функция сложение по модулю два (исключающее ИЛИ, неравнозначность)***

$f_6 = x_1 \oplus x_0 = x_0\overline{x_1} + \overline{x_0}x_1$ – сложение по модулю два, применяется для арифметического сложения.

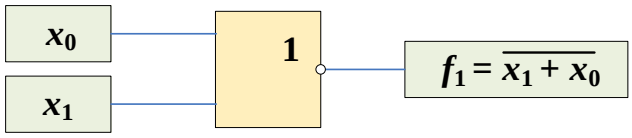
| $V_i$ | $x_1, x_0$ | $f_6$ |
|---|---|---|
| 0 | 0 0 | 0 |
| 1 | 0 1 | 1 |
| 2 | 1 0 | 1 |
| 3 | 1 1 | 0 |




***Функция Пирса логическое сложение с отрицанием, отрицание дизъюнкции (стрелка Пирса ИЛИ-НЕ)***

$f_1 = x_1 \downarrow x_0 = \overline{x_1 + x_0}$ – логическое сложение с отрицанием ИЛИ-НЕ.

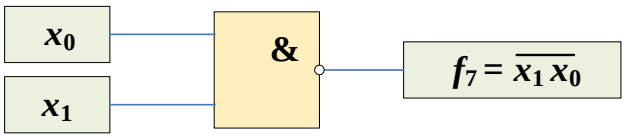
| $V_i$ | $x_1, x_0$ | $f_1$ |
|---|---|---|
| 0 | 0 0 | 1 |
| 1 | 0 1 | 0 |
| 2 | 1 0 | 0 |
| 3 | 1 1 | 0 |




***Функция Шеффера, отрицание от логического умножения (штрих Шеффера И-НЕ)***

$f_7 = x_1 \mid x_0 = \overline{x_0 x_1}$ – логическое умножение с отрицанием И-НЕ.

| $V_i$ | $x_1, x_0$ | $f_7$ |
|---|---|---|
| 0 | 0 0 | 1 |
| 1 | 0 1 | 1 |
| 2 | 1 0 | 1 |
| 3 | 1 1 | 0 |




Функции двух переменных исключительно важны в силу того, что любая логическая функция *n* переменных может быть получена из них методом суперпозиции – подстановкой этих функций вместо переменных в другие функции.

## 1.3. Анализ и синтез комбинационных схем

Логическую схему, имеющую *n* входов $X = \{x_1, x_2, ..., x_n\}$ и *m* выходов $Y = \{y_1, y_2, ..., y_m\}$, можно представить в виде системы уравнений:

$$\left.\begin{aligned} y_1 &= y_1(x_1, x_2, \dots, x_n), \\ y_2 &= y_2(x_1, x_2, \dots, x_n), \\ &\dots \quad \dots \\ y_m &= y_m(x_1, x_2, \dots, x_n) \end{aligned}\right\}. \qquad (18)$$

Логическая схема называется *комбинационной*, если значения множества $Y = \{y_1, y_2, ..., y_m\}$ ее выходов могут быть выражены как система *m* булевых функций от множества входных переменных $X = \{x_1, x_2, ..., x_n\}$, при этом каждая функция $y_i = y_i(x_1, x_2, ..., x_n)$ при $i = 1, 2, \dots, m$ определяет значение на выходе схемы $y_i \in \{0, 1\}$ для любого двоичного набора $(e_1, e_2, \dots, e_n)$, $e_j \in \{0, 1\}$ при $j = 1, 2, \dots, n$, подаваемого на независимые входы схемы.

Система (18) описывает зависимость между входами и выходами схемы, но не дает представления о ее внутренней структуре. Функционирование комбинационной схемы (КС) можно выразить также в виде таблицы истинности, имеющей $2^n$ строк (по строке для каждого набора входных переменных) и $(n+m)$ столбцов ($n$ столбцов для входов и $m$ столбцов для выходов схемы).

Задача синтеза комбинационной схемы состоит в построении схемы для заданной булевой функции или системы булевых функций на основе определенной системы логических элементов. Как правило, исходное описание для синтеза схемы задается либо в виде таблицы истинности, либо в аналитической форме в виде системы (1). При решении задачи синтеза комбинационной схемы, реализующей заданную булеву функцию, предварительно производится минимизация булевой функции.

Минимизация является важным этапом как при создании специализированных интегральных схем, так и при разработке проектов на платформе программируемых логических интегральных схем. В специализированной интегральной схеме для лишних логических элементов и лишних входов необходима лишняя площадь на поверхности кристалла, что приводит к увеличению стоимости микросхем. Далее можно выполнить упрощение минимальной формы путем выделения общих термов (факторизация) и выражения одних функций через другие (декомпозиция).

Анализ комбинационной схемы заключается в первую очередь в формальном описании логической функции, которую реализует эта схема. Получив описание логической функции, можно:

- определить реакцию схемы на различные комбинации входных воздействий;
- преобразовать алгебраическую запись и создать другую структуру схемы, реализующей эту логическую функцию;
- преобразовать алгебраическую запись так, чтобы подогнать ее под имеющийся набор цифровых схем.

На рис. 1 представлена комбинационная схема с тремя входами и одним выходом. Используя только основные аксиомы алгебры логики, можно составить таблицу истинности для схемы с $n$ входами, прослеживая путь от входов к выходам для всех $2^n$ комбинаций входных сигналов. Для каждой такой комбинации определяются сигналы, возникающие на выходах всех вентилей под действием данных входных сигналов.

На рис. 2 у каждой входной линии ($x_1$...$x_3$) выписаны последовательности из восьми логических значений, когда на эти входные линии по очереди подаются сигналы 000, 001,…, 111.

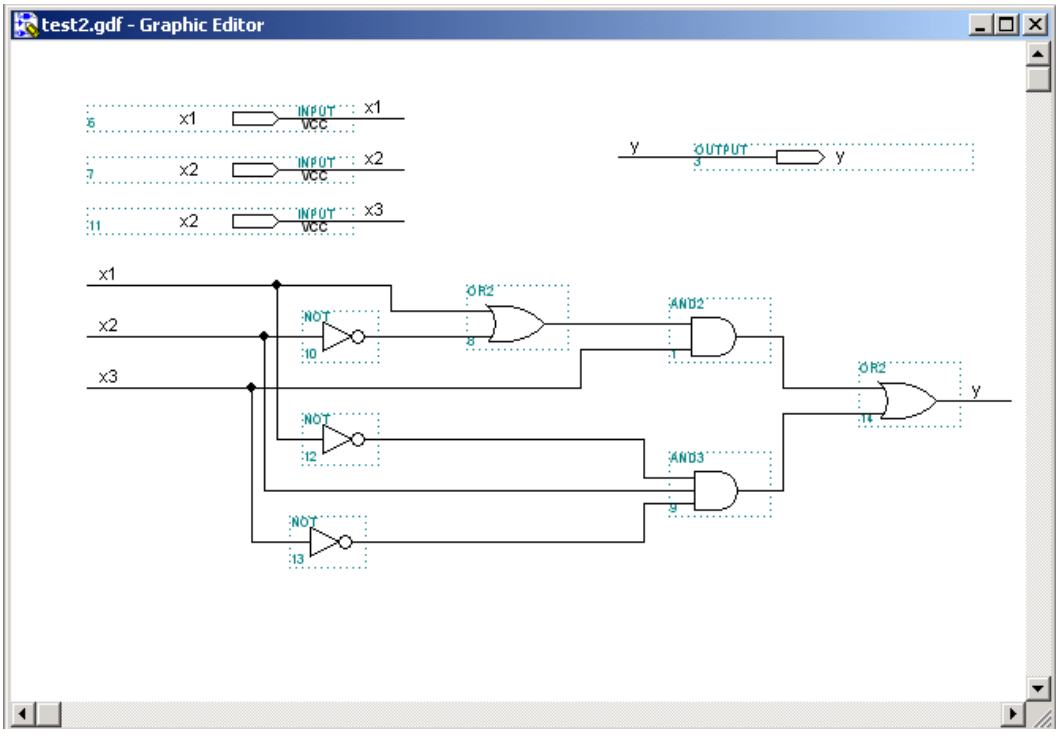



Рис. 1. Комбинационная схема с тремя входами $x_1$… $x_3$ и одним выходом $y$

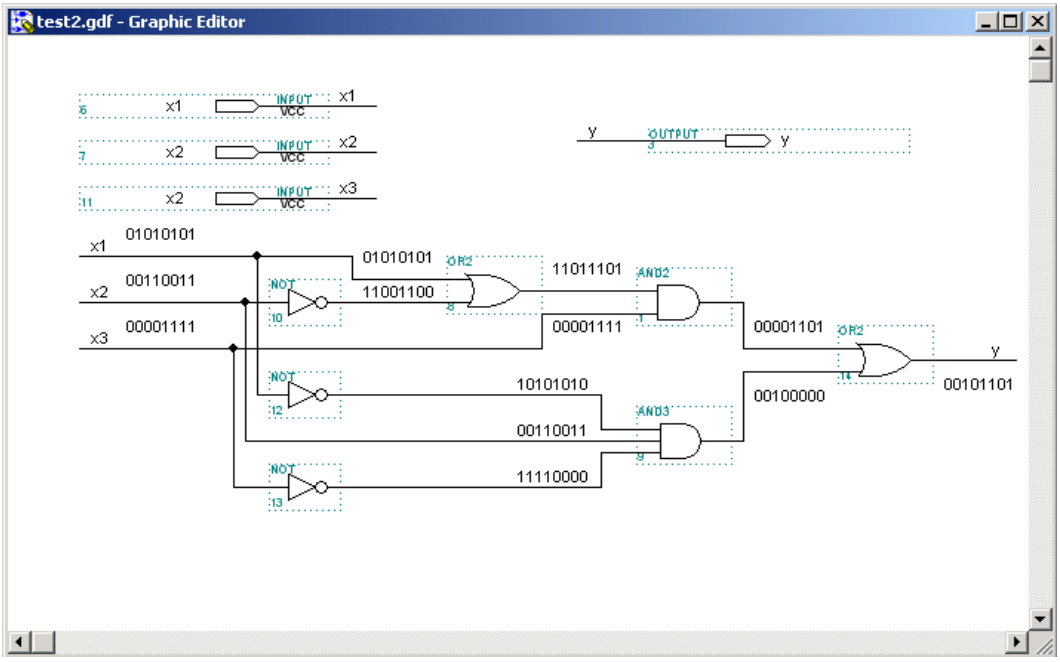



Рис. 2. Логическая схема с комбинациями входных значений (восемь комбинаций)

Составив таблицу истинности для данной схемы, далее можно прямо написать логическое выражение в виде канонической суммы или канонического произведения.

Таблица 4. **Таблица истинности для схемы рис. 1**

| № п/п | *x*3 | *x*2 | *x*1 | *y* |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 2 | 0 | 1 | 0 | 1 |
| 3 | 0 | 1 | 1 | 0 |
| 4 | 1 | 0 | 0 | 1 |
| 5 | 1 | 0 | 1 | 1 |
| 6 | 1 | 1 | 0 | 0 |
| 7 | 1 | 1 | 1 | 1 |

Каноническая сумма логической функции есть сумма минтермов, соответствующих тем строкам таблицы истинности (комбинациям входных сигналов *x* [3…1]), для которых значение функции (*y*) равно 1. Для нашего случая

$$y = \bar{x}_3 \,\&\, x_2 \,\&\, \bar{x}_1 + x_3 \,\&\, \bar{x}_2 \,\&\, \bar{x}_1 + x_3 \,\&\, \bar{x}_2 \,\&\, x_1 + x_3 \,\&\, x_2 \,\&\, x_1.$$

Как уже было сказано выше, знак верхнего подчеркивания над переменной означает инверсию этой переменой. Знак «&» между переменными означает одну из форм записи логического умножения (конъюнкции). Далее запишем каноническое произведение, т.е. произведение макстермов, соответствующее тем комбинациям входных сигналов, для которых значение функции равно 0:

$$y = (x_3 + \bar{x}_2 + \bar{x}_1) \,\&\, (x_3 + \bar{x}_2 + x_1) \,\&\, (x_3 + x_2 + x_1) \,\&\, (x_3 + x_2 + \bar{x}_1).$$

Поскольку число комбинаций входных сигналов растет экспоненциально с увеличением числа входов, то перебирать все возможные комбинации не оптимально. Другой способ получения логического выражения – это алгебраический способ. Для каждого вентиля пишется логическое выражение, начиная от входа и до выхода (рис. 3), при этом учитываются значения функции каждого элемента.

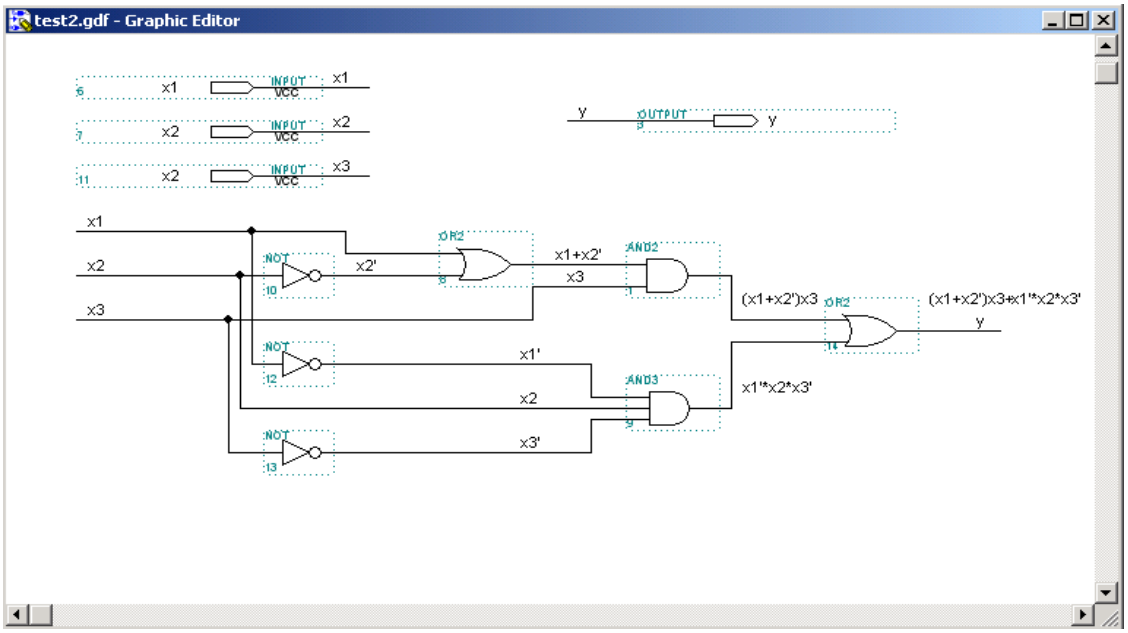



Рис. 3. Логические выражения для различных сигнальных линий

Таким образом, логическое выражение для приведенной схемы выглядит как

$$y = (x_2 + x_1)\& \; x_3 + (x_3 \,\&\, x_2 \,\&\, \bar{x}_1).$$

## 1.4. Минимизация комбинационных схем

Одной из основных задач, возникающих при синтезе комбинационных схем (КС), является минимизация логических функций, которые эти КС реализуют. Чем проще логическое выражение, описывающее функцию, тем проще и дешевле реализующая ее КС.

Существует несколько методов минимизации:

- аналитический, весьма трудоемок и требует сложного подхода, который не всегда виден;
- графический, наиболее нагляден, прост в использовании, но может иметь некоторые ограничения.

Очевидно, что любой метод минимизации может основываться только на тождественном преобразовании логических выражений. Существует несколько способов минимизации булевых функций. Прежде всего это расчетно-табличный метод Куайна–Мак-Класки (Quine–McCluskey method) и метод минимизации с помощью карт Карно или диаграмм Вейча [10].

Метод Куайна–Мак-Класки позволяет с помощью таблиц минимизировать булевы функции. Согласно этому алгоритму, преобразование функции производится в два этапа: первый этап – это переход от канонической формы (СДНФ или СКНФ) к сокращенной, второй – переход от сокращенной формы к минимальной. К достоинствам метода Куайна–Мак-Класки можно отнести следующие положения:

- метод можно применять на большом количестве переменных в САПР, с использованием ЭВМ для минимизации полностью или частично определенных функций;
- функция может быть задана как в СДНФ, так и в СКНФ;
- позволяет последовательно осуществить все этапы минимизации (склеивание и выявление лишних импликант, получение минимальных покрытий).

Рассмотрим минимизацию КС с помощью графического метода – карт Карно. Карта Карно – это графическое представление таблицы истинности, которой задается логическая функция. Карты Карно представляют собой один из табличных способов задания функций и состоят из клеток, каждая из которых соответствует определенной точке *vi* области определения функций. Карты Карно для функции *n* переменных состоят из 2*n* клеток, которые нумеруются числами от 0 до $2^n$–1. Чтобы с помощью такой карты задать функцию *f*(*v*), необходимо в каждую клетку с номером *i* занести значение функции *f*(*vi*) = 0 или 1, которое оно принимает в точке *vi*.

Существуют карты Карно на 2, 3, 4, 5 и 6 переменных (такие карты представлены на рис. 4). Метод Карно основан на законе склеивания. Склеиваются наборы, отличающиеся друг от друга лишь значением одного разряда. Такие наборы называются соседними.

Карно закодировал клетки своей карты так, что в соседних клетках оказались соседние, а значит, склеивающиеся наборы. Соседними могут быть не только отдельные клетки, которые мы назовем элементарными квадратами Карно, но и целые группы соседних клеток (назовем их прямоугольниками Карно).

Под прямоугольником Карно будем понимать некоторую, зачастую разрозненную фигуру покрытия, все соседние клетки которой закодированы соседними наборами. Например, на рис. 4 в поле карты для четырех переменных изображен прямоугольник Карно. Этот прямоугольник состоит из четырех элементарных квадратов Карно (помеченных символом “p”), описываемых наборами *x*4'&*x*3'&*x*2'&*x*1', *x*4'&*x*3'&*x*2&*x*1', *x*4&*x*3'&*x*2'&*x*1' и *x*4&*x*3'&*x*2&*x*1'. Если над логической суммой этих четырех наборов произвести последовательно операции склеивания, то мы аналитически получим результат, соответствующий выражению *x*3'&*x*1'. Карта Карно позволяет получить этот результат графически значительно быстрее и проще.

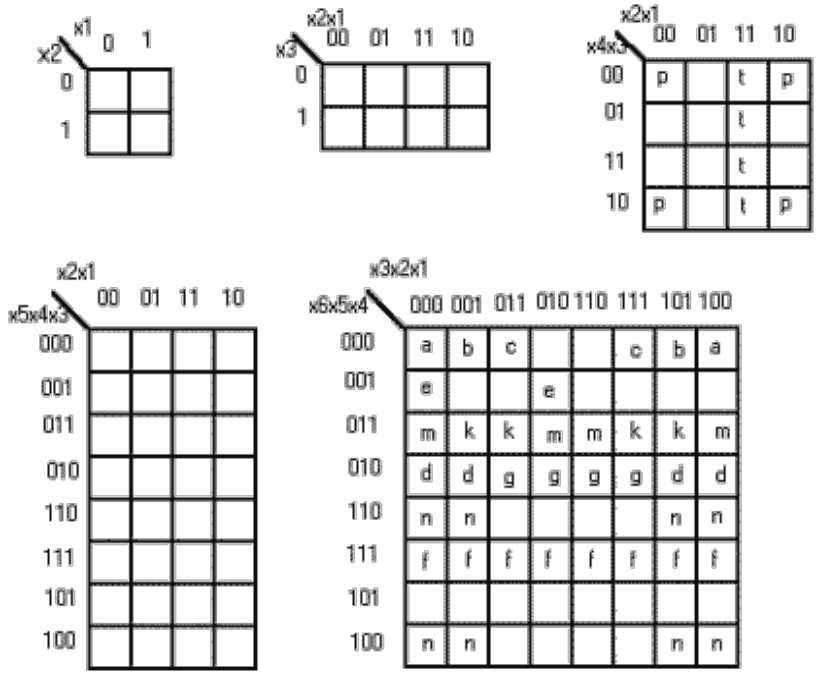



Рис. 4. Карты Карно для 2, 3, 4, 5 и 6 переменных

Для решения этой задачи используем алгоритм графической минимизации с определенными правилами:

- заполнить карту Карно нулями и единицами в соответствии с таблицей истинности;
- покрыть все единичные наборы минимальным количеством прямоугольников Карно, каждый из которых имеет максимальную площадь;
- сторона прямоугольника Карно должна быть кратна степени 2 (т.е. 1, 2, 4, 8…). Таким образом, прямоугольник 3×4 не является прямоугольником Карно.

Выполнив минимизацию комбинационной логической схемы (КЛС), можно осуществлять ее синтез, включающий в себя следующие этапы:

1) задание функции (функций) с помощью таблицы истинности на основе поставленной задачи;
2) алгебраическая запись функции в виде СДНФ и СКНФ, определение более выгодного варианта;
3) минимизация функции (функций);
4) анализ и при необходимости изменение функций на возможность совместной реализации;
5) выбор базиса функции (функций) и приведение ее к этому базису;
6) построение КЛС на логических элементах, удовлетворяющих выбранному базису.

Более подробно с особенностями минимизации КС с помощью карт Карно можно познакомиться в [10].

**Упражнения**

**У1.** Составить таблицу истинности для каждой из следующих логических функций:

а) $y = \bar{x}_2 \& x_1 + x_3 \& \bar{x}_1 + x_3 \& x_2$; б) $y = \bar{x}_1 \& x_2 + x_3 \& \bar{x}_1 + \bar{x}_1 \& x_4$;

в) $y = \bar{x}_1 + x_2 \& (\bar{x}_3 + x_3 \& x_4)$; г) $y = \bar{x}_4 \& x_2 + \bar{x}_2 \& \bar{x}_1 + \bar{x}_1 \& x_2$;

д) $y = \bar{x}_3 \& x_2 + x_3 \& \bar{x}_1 \& x_2$;  е) $y = (x_4 + x_2 + x_2 \& \bar{x}_2) \& (x_1 + x_2 + \bar{x}_2 \& x_4)$.

**У2.** С помощью карт Карно найти минимальное выражение для функций упражнения **У1**, используя операции склеивания и поглощения.

**У4.** Записать СДНФ функции, заданной таблицей истинности и представленной ниже.

| *дес* | $x_4$ | $x_3$ | $x_2$ | $x_1$ | $y$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 2 | 0 | 0 | 1 | 0 | 1 |
| 3 | 0 | 0 | 1 | 1 | 0 |
| 4 | 0 | 1 | 0 | 0 | 1 |
| 5 | 0 | 1 | 0 | 1 | 0 |
| 6 | 0 | 1 | 1 | 0 | 1 |
| 7 | 0 | 1 | 1 | 1 | 0 |
| 8 | 1 | 0 | 0 | 0 | 1 |
| 9 | 1 | 0 | 0 | 1 | 0 |
| 10 | 1 | 0 | 1 | 0 | 0 |
| 11 | 1 | 0 | 1 | 1 | 0 |
| 12 | 1 | 1 | 0 | 0 | 0 |
| 13 | 1 | 1 | 0 | 1 | 1 |
| 14 | 1 | 1 | 1 | 0 | 1 |
| 15 | 1 | 1 | 1 | 1 | 1 |

**Контрольные вопросы**

1. Что изучает алгебра логики?
2. Приведите примеры числового кодирования информации.
3. Приведите примеры теорем и законов алгебры логики.
4. Что такое комбинационная схема?
5. Объясните принцип синтеза комбинационной схемы.
6. Объясните принцип анализа комбинационной схемы.

# Глава 2. Структура учебного стенда CLE-148 для проектирования и проверки функционирования цифровых устройств

Для разработки проектов цифровых устройств в интегрированном пакете Quartus Prime и проверки функционирования будем использовать учебный стенд CLE-148 [11] на базе ПЛИС семейства Cyclone IV (рис. 5). В составе стенда используются следующие комплектующие:

- макетная плата в стандарте промышленного интерфейса ISA;
- плата развития DEO_Nano [12];
- силовые драйверы SN74LVT245 для согласования нагрузки и выходов DEO_Nano;
- силовой драйвер для подключения шагового двигателя;
- силовой драйвер для подключения двигателей постоянного тока RPi Motor Driver [13];
- модуль расширения ДМ-1 [14].

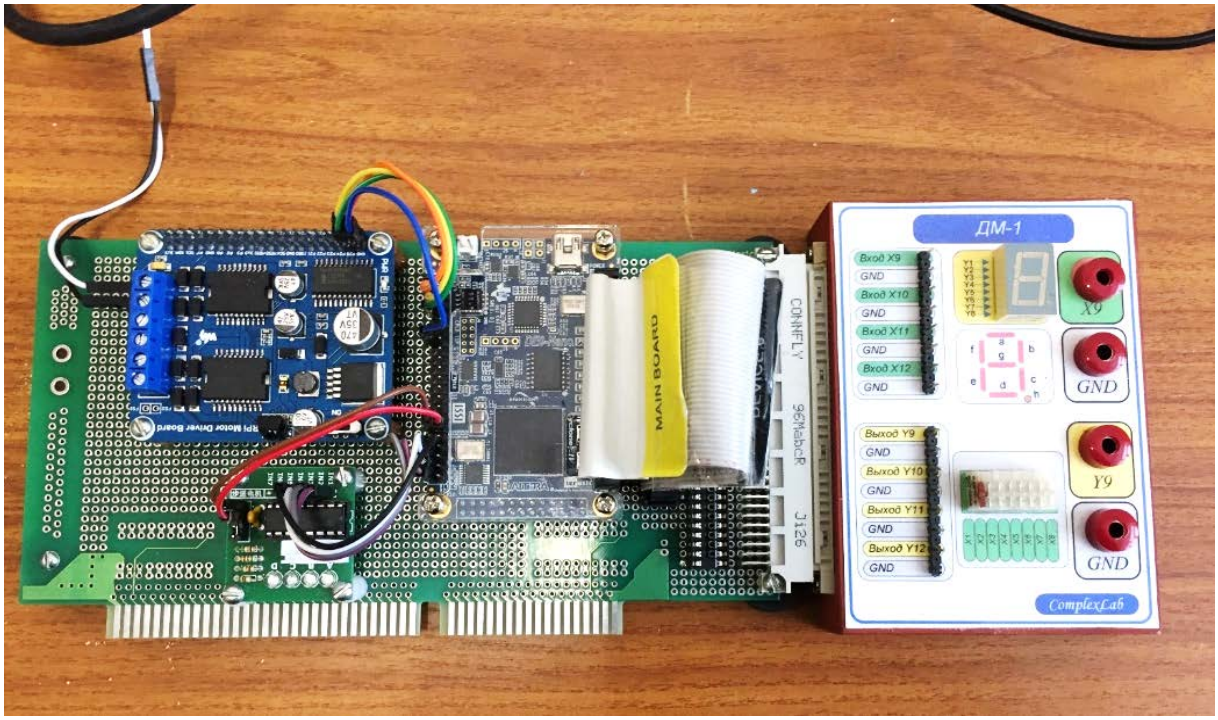



Рис. 5. Учебный стенд CLE-148

Подключение отдельных компонентов к плате развития DEO_Nano выполняется с помощью монтажных соединителей и гибкого шлейфа.

## 2.1. Плата развития DEO-Nano

Плата развития DEO-Nano представляет собой устройство компактного размера на базе FPGA – ПЛИС для разработки широкого спектра портативных дизайн-проектов, включающих как управляющие системы, робототехнику, так и цифровые обучающие приложения. На рис. 6 представлен общий вид DEO-Nano.

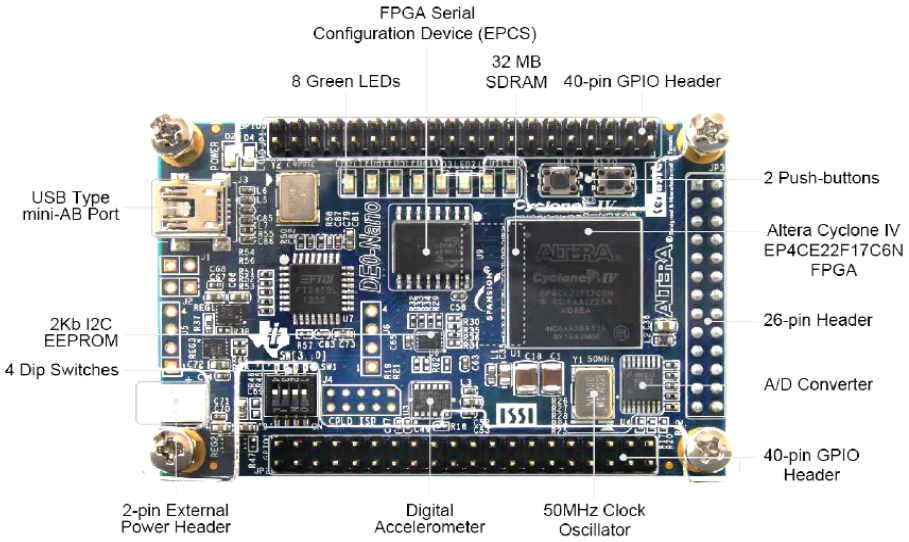



Рис. 6. Функциональные узлы платы развития DEO-Nano

Блок-схема DEO-Nano представлена на рис. 7. К основным характеристикам блоков платы DEO-Nano можно отнести:

- ПЛИС Altera Cyclone IV EP4CE22F17C6N;
- два 40-контактных разъема расширения (GPIOs) обеспечивают 72 ввода/вывода, выводы питания 5V, 3.3V и ножки заземления;
- устройства памяти: 32MB SDRAM, 2KB I2C EEPROM;
- 3-осевой акселерометр с высоким разрешением ADI ADXL345;
- NS ADC128S022, 8-канальный, 12-разрядный аналого-цифровой преобразователь;
- встроенный USB-Blaster для программирования ПЛИС.

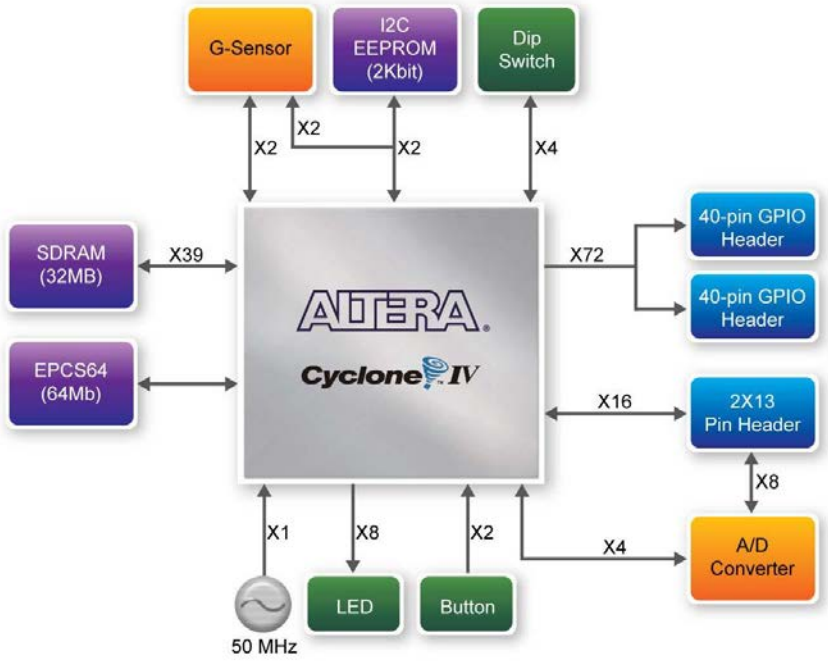



Рис. 7. Блок-схема DEO-Nano

Плата DEO-Nano содержит две кнопки, показанные на рис. 8. Каждая кнопка имеет противодребезговую защиту с помощью схемы триггера Шмидта. Два сигнала от кнопок, пропущенные через триггер Шмидта, называемые KEY0 и KEY1, подключаются непосредственно к Cyclone IV ПЛИС.

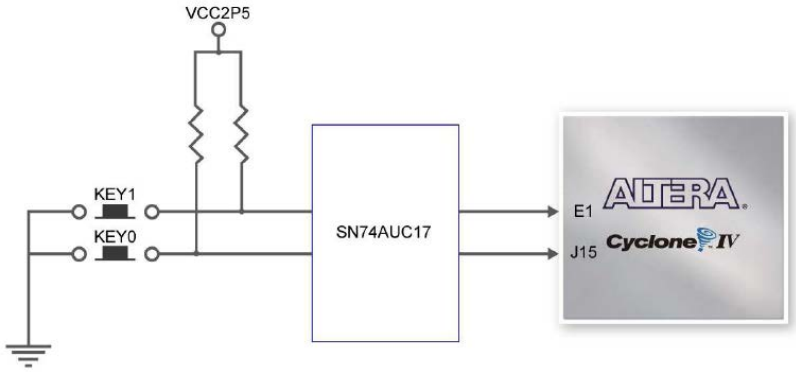



Рис. 8. Системные кнопки KEY0 и KEY1 платы DEO-Nano

Каждая кнопка обеспечивает высокий логический уровень, когда она не нажата, и низкий логический уровень при нажатии. Поскольку кнопки имеют противодребезговую защиту, они подходят для использования в качестве тактового сигнала или сброса входов.

На плате DEO-Nano имеется 8 контролируемых пользователем зеленых светодиодов. Эти светодиоды позволяют пользователям отображать статус и информацию об отладке. Каждый светодиод управляется непосредственно с помощью соответствующих выводов (в режиме выхода) на Cyclone IVE ПЛИС; высокий логический уровень включает светодиод, а низкий выключает его.

Плата DEO-Nano содержит 4 микропереключателя. DIP-переключатель обеспечивает на входе ПЛИС высокий логический уровень, когда он находится в нижнем положении, и низкий уровень логики в верхнем положении. В табл. 5 указаны адреса ножек и назначение контактов. Более подробные характеристики содержатся в паспорте платы развития DEO-Nano.

Таблица 5. **Адреса ножек FPGA для кнопок, светодиодов и DIP-переключателей**

| Имя переменной | Адрес Pin FPGA | Назначение | I/O Standard |
|---|---|---|---|
| KEY[0] | PIN_J15 | Push-button[0] | 3.3V |
| KEY[1] | PIN _E1 | Push-button[1] | 3.3V |
| LED[0] | PIN _A15 | LED Green[0] | 3.3V |
| LED[1] | PIN _A13 | LED Green [1] | 3.3V |
| LED[2] | PIN _B13 | LED Green [2] | 3.3V |
| LED[3] | PIN _A11 | LED Green [3] | 3.3V |
| LED[4] | PIN _D1 | LED Green [4] | 3.3V |
| LED[5] | PIN _F3 | LED Green [5] | 3.3V |
| LED[6] | PIN _B1 | LED Green [6] | 3.3V |
| LED[7] | PIN _L3 | LED Green [7] | 3.3V |
| DIP Switch[0] | PIN _M1 | DIP Switch[0] | 3.3V |
| DIP Switch[1] | PIN _T8 | DIP Switch[1] | 3.3V |
| DIP Switch[2] | PIN _B9 | DIP Switch[2] | 3.3V |
| DIP Switch[3] | PIN _M15 | DIP Switch[3] | 3.3V |

## 2.2. Управляющая плата для двигателей постоянного тока (RPi Motor Driver)

Плата расширения для управления двигателями постоянного тока (DC двигателями) RPi Motor Driver (рис. 9) позволяет подключать к DEO-Nano два двигателя общей мощностью до 400 Вт. Плата предназначена для расширения возможностей управляющих микроконтроллеров и ПЛИС.

Состав функциональных компонентов платы и их основные характеристики:

1) 40-контактный разъем для связи с микроконтроллерами или ПЛИС,
2) разъемы для подключения двигателей постоянного тока,
3) разъем для подключения питания в диапазоне от 7 В до 40 В (*V*+),
4) преобразователь уровней напряжения 74LVC4245AD,
5) силовой драйвер MC33886 для подключения двигателей,
6) регулятор напряжения LM2596-5.0 (вход – 7–40В, выход – 5 В),
7) светодиодный индикатор питания,

8) переключатель выбора источника питания; OFF – контроллер обеспечивает питание управляющей платы, ON – питание управляющей платы обеспечивает питание контроллера,

9) самовосстанавливающийся предохранитель питания 2 А,

10) инфракрасный фотодиод для удаленного управления двигателями (например, с пульта),

11) диоды Шотки для защиты микросхем драйверов двигателей,

12) обратный диод для защиты от переполюсовки, при подключении блока питания.

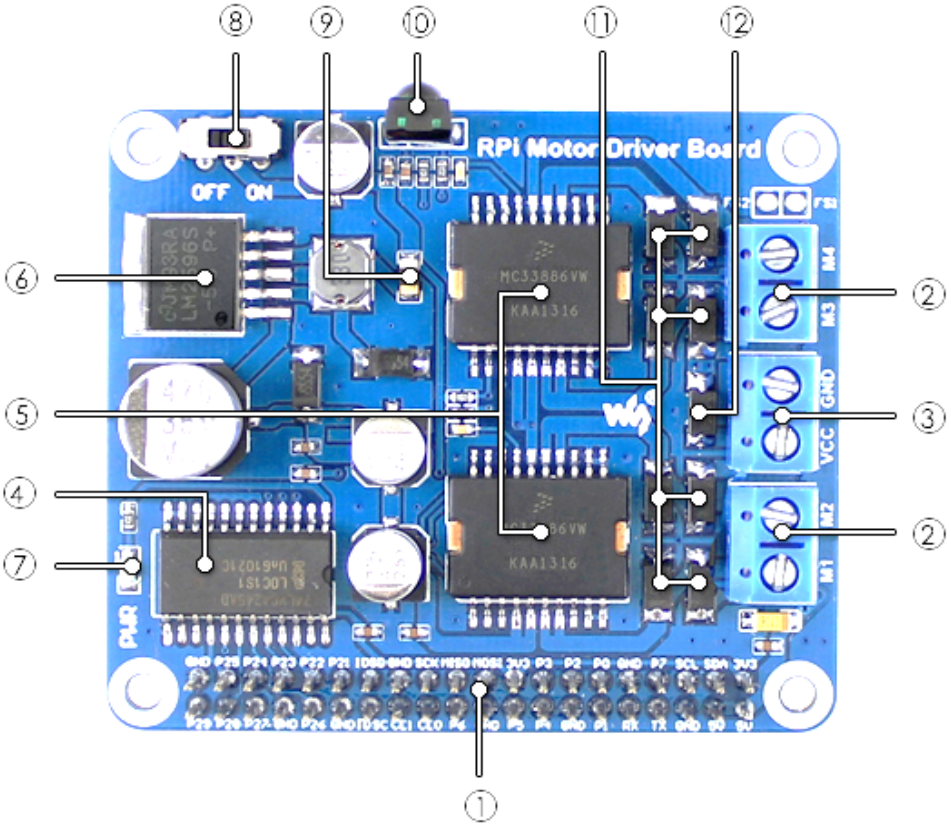



Рис. 9. Управляющая плата RPi Motor Driver

В основе платы расширения RPi Motor Driver – драйвер компании NXP (Freescale) MC33886, представляющий собой H-мост (рис. 10) на MOSFET транзисторах со схемой управления [15].

Многочисленные защитные и рабочие функции (скорость, крутящий момент, направление, динамическое торможение и ШИМ-управление), в дополнение к допустимому току 5,0 А, делают MC33886 очень привлекательным и экономичным решением для управления широким спектром двигателей постоянного тока. Два устройств MC33886 могут использоваться для управления биполярными шаговыми двигателями в обоих направлениях. Как показано на рис. 10, MC33886 представляет собой полностью защищенный монолитный H-мост с контролем неисправности. Чтобы двигатель постоянного тока работал, входные условия должны быть следующими: входная логика D1 – низкий уровень, входная логика D2 – высокий, флаг FS сброшен (высокий логический уровень), при этом одна логика IN – низкая, а другая – высокая, чтобы определить выходную полярность. MC33886 может выполнять динамическое торможение путем одновременного включения обоих полевых МОП-транзисторов верхнего уровня или обоих полевых МОП-транзисторов нижнего уровня в выходном H-мосте; например, высокий логический уровень IN1 и IN2 или низкий логический уровень IN1 и IN2. MC33886 способен управлять индуктивной нагрузкой с рабочим током до 5 А. Работа в режиме ШИМ возможна на частотах до 10 кГц.

На плате RPi Motor Driver установлен 5 В стабилизатор напряжения, обеспечивающий питанием RPi. Для дистанционного управления, например, роботом на RPi в изделии установлен IR-приемник. Плата характеризуется хорошей стабильностью. Целый ряд защитных цепей обеспечивает надежную работу платы драйвера. Среди них можно назвать самовосстанавливающийся предохранитель на 2 А, обеспечивающий безопасность и защиту драйвера от короткого замыкания (от больших токов нагрузки), функцию отключения при снижении напряжения, защиту от неправильной полярности подключения источника питания.

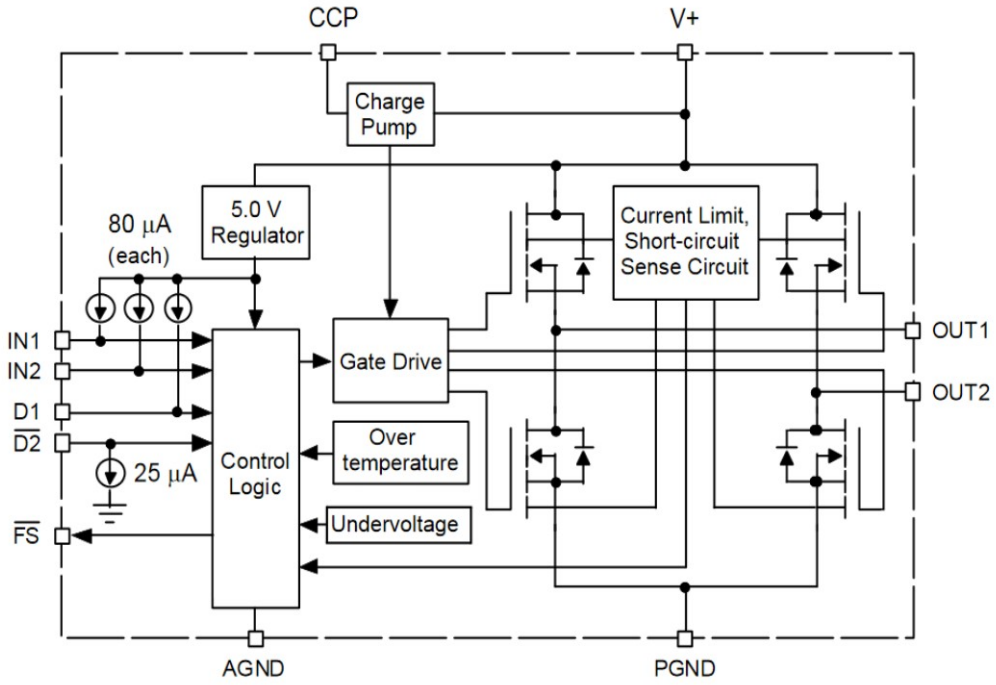



Рис. 10. Блок-диаграмма силового драйвера MC33886

В табл. 6 представлена логика управления двумя двигателями постоянного типа DC1 (переменные управления M1 и M2) и DC2 (переменные управления M3 и M4). Для более подробного изучения работы платы смотрите описание [13].

Таблица 6. **Управляющая логика платы RPi Motor Driver**

| M1 | M2 | M3 | M4 | Режим работы двигателей постоянного тока |
|---|---|---|---|---|
| 1 | 0 | 1 | 0 | Вращение двигателей DC1 и DC2 в прямом направлении |
| 0 | 1 | 0 | 1 | Вращение двигателей DC1 и DC2 в обратном направлении |
| 0 | 0 | 1 | 0 | DC1 остановлен, DC2 вращается в прямом направлении |
| 1 | 0 | 0 | 0 | DC1 вращается в прямом направлении, DC2 остановлен |
| 0 | 0 | 0 | 0 | DC1 и DC2 остановлены |

## 2.3. Плата драйвера шагового двигателя

Для управления шаговым двигателей (ШД) в стенде используется плата на базе микросхемы ULN2003 (рис. 11). Микросхема ULN2003 содержит семь независимых каналов с транзистором Дарлинктона (мощные составные транзисторы) в каждом. Для управления индуктивной нагрузкой в состав каждого канала включен сапрессор (Transient Voltage Suppressor – защитный стабилитрон для ограничения перенапряжения).

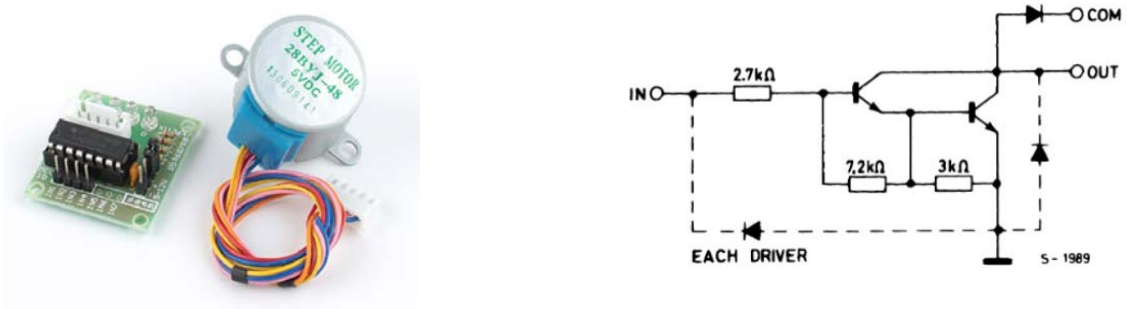



Рис. 11. Плата управления, ШД 28BYJ-48 (слева) и принципиальная схема одного канала ULN2003

На плате четыре светодиода показывают состояние каждой из четырех линий управления шаговым двигателем. При работе стенда в режиме управления ШД можно использовать любой ШД с характеристиками, соответствующими плате управления.

Основные характеристики платы и шагового двигателя 28BYJ-48-5V:

- Максимальный ток нагрузки каждого выхода платы, мА ..... 500
- Напряжение питания платы, В..... 5 (12)
- Тип двигателя ..... 4-фазный униполярный
- Номинальное сопротивление обмоток, Ом ..... 50
- Угол поворота внутреннего вала за один шаг, градусов..... 5, 625
- Количество шагов внутреннего вала на оборот ..... 64
- Максимальная частота формирование сигналов «Шаг», Гц..... 500
- Коэффициент редукции ШД ..... 1/63,68395
- Напряжение питания ШД, В ..... 5
- Крутящий момент не менее, мНм..... 34,3

## 3.4. Модули расширения

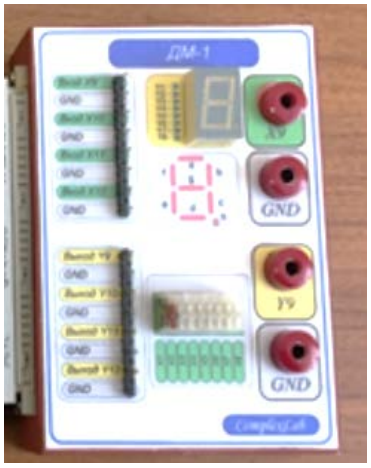



Рис. 12. Модуль расширения ДМ-1

Для расширения функциональных характеристик стенда CLE-148 предусмотрена возможность подключения дополнительных модулей расширения через специальный 96-контактный разъем. Для повышения нагрузочной способности выходных каналов DEO-Nano (8 мА) и управления модулей расширения с током потребления больше 8 мА на макетной плате стенда размещены силовые драйверы SN74LVT245.

Рассмотрим характеристики модуля расширения ДМ-1 (рис. 12). Модуль содержит семисегментный индикатор (8 выходов DEO-Nano), 8 движковых переключателей (подключенных на 8 входов), 8 штырьковых однопиновых разъемов (4 – вход, 4 – выход). Модуль расширения позволяет применить семисегментный индикатор для отображения числа циклов какого-то процесса с использованием разработанного дешифратора. С помощью движковых переключателей можно задать двоичную константу, а однопиновый разъем позволяет подключить осциллограф, внешний генератор или датчики. Адреса выводов ПЛИС стенда для подключения ДМ-1 представлены в паспорте модуля расширения.

**Контрольные вопросы**

1. Какие структурные элементы содержит стенд CLE-148?
2. Какой тип ПЛИС (CPLD или FPGA) содержит плата развития DEO-Nano?
3. Какова нагрузочная способность выходов платы DEO-Nano?
4. Для чего используется в составе стенда плата RPi Motor Driver?
5. Какой тип шагового двигателя используется в учебном стенде CLE-148?
6. Какой максимальный ток нагрузки каждого выхода платы управления ШД?
7. Для чего необходим в составе стенда модуль расширения ДМ-1?

# Глава 3. Разработка проектов на базе ПЛИС в пакете Quartus Prime

Существует достаточно большое количество интегрированных пакетов, которые позволяют не только создать и проверить работоспособность проектов на базе ПЛИС, но и запрограммировать эти ПЛИС, т.е. записать конфигурационный файл в цифровые матрицы либо в энергонезависимую память. Как правило, такие программные пакеты создают производители ПЛИС. На рынке ПЛИС можно выделить две основных фирмы – это Intel с пакетом Quartus Prime [9], которая приобрела фирму Altera и Xilinx с пакетом Vivado Design Suite [16]. Интегрированные пакеты этих фирм имеют условно-бесплатные версии с ограничением по времени использования лицензий. Учебный стенд, на базе которого будем создавать, проверять и управлять внешними исполнительными устройствами, построен с использованием ПЛИС фирмы Intel (Altera), поэтому более подробно будем говорить о пакете Intel Quartus Prime.

## 3.1. Интегрированный пакет QUARTUS PRIME: основные характеристики

Intel Quartus Prime – это программное обеспечение, разработанное Intel для создания электронных устройств на основе цифровых матриц (ПЛИС). Quartus Prime дает возможность анализировать и синтезировать проекты, разработанные в виде схемы или программы на языках описания аппаратуры HDL (Hardware Description Language), что позволяет разработчику скомпилировать их проекты, выполнить анализ синхронизации, изучить диаграммы RTL (Register Transfer Level), смоделировать реакцию проекта на различные входные воздействия. Quartus Prime включает в себя поддержку языков VHDL, Verilog и SystemVerilog для описания аппаратных устройств, визуального редактирования логических схем и моделирования векторных сигналов.

Quartus Prime включает следующие возможности:

- SOPC Builder (System on a Programmable Chip Builder) – инструмент в программном обеспечении Quartus Prime, который устраняет задачи ручной системной интеграции, автоматически генерируя логику межсоединений и создавая тестовую среду для проверки функциональности.
- Qsys – инструмент системной интеграции, являющийся следующим поколением SOPC Builder. Он использует оптимизированную FPGA архитектуру сети на кристалле, которая удваивает производительность по сравнению с SOPC Builder.
- SoC EDS (Embedded Development Suite) – набор инструментов разработки, служебных программ, программного обеспечения времени выполнения и примеров приложений, которые помогут вам разработать программное обеспечение для встроенных систем SoC (System on Chip) FPGA.
- DSP Builder, инструмент, который создает связь между инструментом MATLAB/Simulink и программным обеспечением Quartus Prime, поэтому разработчики FPGA имеют возможности разработки, моделирования и проверки алгоритмов инструментов проектирования на системном уровне MATLAB/Simulink.

Рассмотрим основные этапы разработки проекта (design flow) в пакете Quartus Prime. На рис. 13 изображено дерево разработки проекта с перечислением возможностей и промежуточных результатов на каждом этапе.

Процесс проектирования и компиляции Quartus Prime Integrated Synthesis состоит из следующих этапов:

1. Создание проекта в пакете Quartus Prime с указанием общей информации о проекте, включая имя объекта дизайна верхнего уровня.
2. Создание файлов дизайна в программном обеспечении Quartus Prime или с помощью текстового редактора.
3. В меню «Проект» необходимо выбрать «Добавить/удалить файлы в проекте» и добавить все файлы дизайна в проект Quartus Prime на странице «Файлы» диалогового окна «Настройки».
4. Выполнить настройки компилятора, которые управляют компиляцией и оптимизацией вашего проекта во время синтеза и размещения в кристалле.
5. Добавить временные ограничения, чтобы определить временные требования.
6. Выполнить компиляцию проекта. Чтобы синтезировать дизайн, в меню «Обработка» необходимо выбрать «Начать», а затем запустить «Начать анализ и синтез». Чтобы запустить полный процесс компиляции, включая размещение, маршрутизацию, создание файла программирования и анализ времени, необходимо нажать «Начать компиляцию» в меню «Обработка».

7. После получения результатов синтеза, размещения и маршрутизации, соответствующих вашим требованиям, можно запрограммировать или настроить устройство.

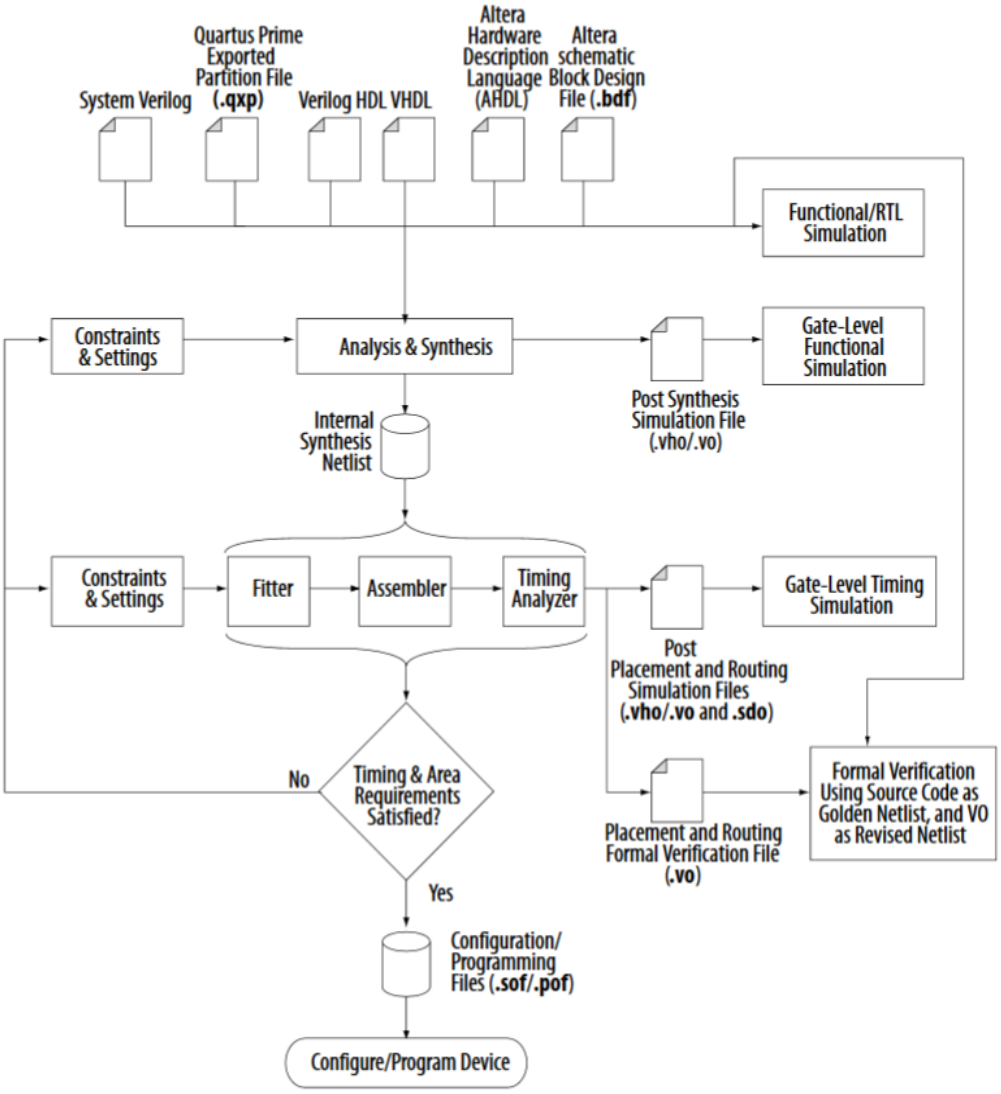



Рис. 13. Дерево последовательности этапов выполнения проекта [9]

## 3.2. Различные способы разработки проектов в пакете Quartus Prime

При разработке электронных устройств используется множество технологий как на аппаратном уровне, так и на программном. Выбор того или иного способа разработки определяется предпочтением разработчика, типом решаемой задачи (тестирование или синтезирование цифровых устройств), программными и аппаратными возможностями. В данном пособии авторы постараются осветить основы различных методов разработки.

### 3.2.1. Разработка проектов в схемном редакторе Quartus Prime

Разработка в схемном редакторе имеет преимущества в том смысле, что интуитивно понятно соединение различных библиотечных элементов (логических элементов, триггеров, счетчиков и т.д.) «проводами», такая разработка достаточна наглядна для человека, который лучше всего воспринимает графические визуальные образы. Кроме того, нет необходимости для изучения новых языков описания аппаратуры. Но этот способ может быть использован для небольших проектов, т.к. с усложнением схемы растет количество соединений и составляющих функциональных элементов.

Вместе с тем нельзя сказать, что такой способ не подходит для разработки сложных проектов. Опыт авторов учебного пособия по разработке электронной аппаратуры системы управления ускорительного комплекса [7; 8] показывает, что наилучший результат дает сочетание схемного и языкового описания модулей на ПЛИС.

### 3.2.2. Языки описания аппаратуры и различные типы описания модулей: структурный, поведенческий

В этом параграфе дадим краткий анализ языков программирования для описания аппаратуры и различные способы ее описания. Для того чтобы спроектировать современную микросхему, от алгоритма до реализации в кремниевых пластинах, необходимы обширные знания по структуре и взаимодействию отдельных элементов электроники, знание алгоритмов этих взаимодействий, а кроме того, умение пользоваться большим количеством специализированных языков программирования. Для описания структурного или поведенческого описания цифровой схемы можно использовать языки Verilog HDL (Verilog Hardware Description Language) [17], VHDL (**V**HSIC (Very high speed integrated circuits) **H**ardware **D**escription **L**anguage) [18] или SystemVerilog [19]. Последний из приведенных языков (SystemVerilog) стал универсальным языком, который помимо классического языка Verilog (с небольшими расширениями), содержит в себе объектно-ориентированный язык для написания тестовых окружений и конструкций утверждений для формальной верификации. Язык SystemVerilog активно развивается, его поддерживают все ведущие производители ПЛИС (например, Intel (Altera), Xilinx и т.д.), поэтому все примеры проектов цифровых устройств, приведенные в данном учебном пособии, будут на этом языке. Основные операторы, типы данных, применяемые в языке SystemVerilog, будут представлены в различных проектах по мере их использования. В данном учебном пособии не было задачи рассказать о всех возможностях и особенностях SystemVerilog. Для более углубленного изучения языка в списке литературы пособия предлагаются ссылки на стандарты и признанные учебники по язык SystemVerilog, например, [20–22].

## 3.3. Разработка комбинационной схемы в схемном редакторе Quartus Prime

Разработаем в схемном редакторе Quartus Prime проект комбинационной схемы с функцией дешифратора. Разработка включает следующие этапы:

- синтез комбинационной схемы дешифратора на основе таблицы истинности;
- алгебраическая запись функции в виде СДНФ и СКНФ, определение более выгодного варианта;
- минимизация функции (функций);
- анализ и при необходимости изменение функций на возможность совместной реализации;
- выбор базиса функции (функций) и приведение ее к этому базису;
- построение комбинационной логической схемы (КЛС) на логических элементах, удовлетворяющих выбранному базису;
- создание проекта в схемном редакторе Quartus Prime;
- компиляция, временное симулирование схемы, загрузка выходного кода в ПЛИС.

Проведем синтез комбинационной логической схемы, реализующей функцию $y = f(x_1, x_2, x_3, x_4)$, в базисах И-ИЛИ-НЕ. Функция $y = f(x_1, x_2, x_3, x_4)$ задана входными наборами таблицы истинности, представленной на рис. 14.

Выполним следующие действия:

- запишем СДНФ (совершенную дизъюнктивную нормальную форму функции);
- минимизируем заданную функцию с помощью карт Карно;
- в пакете Quartus Prime создадим проект, из библиотечных элементов нарисуем получившуюся минимальную функцию в схемном редакторе;
- выполним компиляцию проекта;
- выполним временную и функциональную симуляцию КЛС;
- загрузим конфигурационный файл в ПЛИС;
- с помощью входных движковых переключателей стенда (см. раздел «Учебный стенд CLE-148») зададим входные последовательности КЛС (переменные $x_1, x_2, x_3, x_4$) и смотрим на светодиодном индикаторе значение функции ($y$).

### 3.3.1. Синтез комбинационной схемы, заданной таблицей истинности

Выполним синтез КЛС, реализующей функцию $y = f(x_1, x_2, x_3, x_4)$ в базисах И-ИЛИ. Функция $y = f(x_1, x_2, x_3, x_4)$ задана входными наборами (2, 4, 6, 8, 13, 14, 15) таблицы истинности (см. рис. 14), на которых она равна 1.

1. СДНФ для заданной функции имеет вид: $y = \bar{x_4} \,\&\, \bar{x_3} \,\&\, x_2 \,\&\, \bar{x_1} + \bar{x_4} \,\&\, x_3 \,\&\, \bar{x_2} \,\&\, \bar{x_1} +$

$+ \ \bar{x}_4 \& x_3 \& x_2 \& \ \bar{x}_1 \ + x_4 \& \ \bar{x}_3 \& \ \bar{x}_2 \& \ \bar{x}_1 + x_4 \& x_3 \& \ \bar{x}_2 \& x_1 + x_4 \& x_3 \& x_2 \& \ \bar{x}_1 + x_4 \& x_3 \& x_2 \& x_1.$

2. Минимизируем заданную функцию с помощью карты Карно. Для получения минимальной ДНФ объединяем в группы смежные (соседние) единичные клетки.

| dec | $x_4$ | $x_3$ | $x_2$ | $x_1$ | $y$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 2 | 0 | 0 | 1 | 0 | 1 |
| 3 | 0 | 0 | 1 | 1 | 0 |
| 4 | 0 | 1 | 0 | 0 | 1 |
| 5 | 0 | 1 | 0 | 1 | 0 |
| 6 | 0 | 1 | 1 | 0 | 1 |
| 7 | 0 | 1 | 1 | 1 | 0 |
| 8 | 1 | 0 | 0 | 0 | 1 |
| 9 | 1 | 0 | 0 | 1 | 0 |
| 10 | 1 | 0 | 1 | 0 | 0 |
| 11 | 1 | 0 | 1 | 1 | 0 |
| 12 | 1 | 1 | 0 | 0 | 0 |
| 13 | 1 | 1 | 0 | 1 | 1 |
| 14 | 1 | 1 | 1 | 0 | 1 |
| 15 | 1 | 1 | 1 | 1 | 1 |

| x3x2 \ x1x0 | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 0 | 0 | 1 |
| 01 | 1 | 0 | 0 | 1 |
| 11 | 0 | 1 | 1 | 1 |
| 10 | 1 | 0 | 0 | 0 |

Рис. 14. Таблица истинности (слева) и карта Карно (справа) для функции $y = f(x_1, x_2, x_3, x_4)$

Получившаяся после минимизации функция представляет собой логическое уравнение:

$$y = \ \bar{x}_4 \ \& x_2 \& \ \bar{x}_1 + x_4 \& x_3 \& x_1 + x_4 \& x_3 \& x_2 \ + \ \bar{x}_4 \ \& x_3 \& \ \bar{x}_1 + x_4 \ \& \ \bar{x}_3 \& \ \bar{x}_2 \& \ \bar{x}_1.$$

Создадим проект в интегрированном пакете Quartus Prime.

### 3.3.2. Создание проекта в пакете Quartus Prime

Разработка проекта в схемном редакторе пакета Quartus Prime начинается с запуска программы-мастера по созданию нового проекта (New Project Wizard). Запуск этой программы можно выполнить через графическую иконку (рис. 15) или перейдя по ссылке File/ New Project Wizard. Разработка проекта будет выполняться в несколько этапов.

На первом этапе необходимо определить:

- название проекта и папку, где этот проект (с большим числом файлов после компиляции) будет сохраняться;

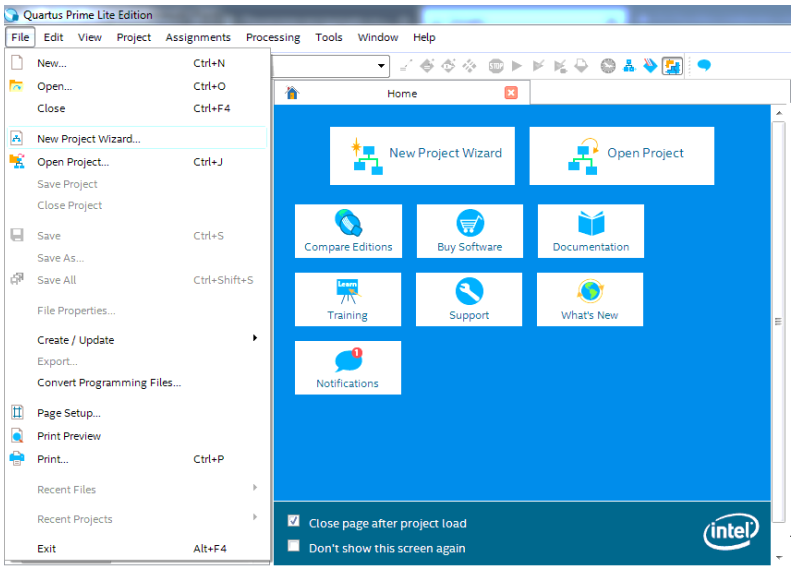



Рис. 15. Главное окно интегрированного пакета Quartus Prime

- название файла верхнего уровня проекта (должно совпадать с названием проекта);
- включить в проект, если необходимо, какие-то готовые части от ранее разработанных проектов и библиотеки;
- определить тип микросхемы (ПЛИС) для этого проекта (эту опцию можно всегда изменить);
- включить для создания проекта необходимые программные инструменты (эти инструменты должны быть установлены до начала работы над проектом).

Не все перечисленные пункты необходимо обязательно выполнять. Например, если при создании проекта не будет выполняться симуляция или еще что-нибудь, то этот пункт просто пропускается. После ознакомления с введением в разработку проекта нажимаем кнопку Next > (рис. 16).

Выполним перечисленные действия по шагам. Необходимо отметить, что любой проект в пакете **Quartus_Prime** состоит из набора функциональных модулей, среди которых есть один, названный модулем верхнего уровня. Каждый модуль состоит из исходного файла проекта (созданного нами) и нескольких служебных файлов различного функционального назначения. Эти служебные файлы появляются после компиляции исходного файла проекта и отвечают за конфигурацию проекта, за размещение (загрузку) выходного файла в ПЛИС и т.д. О некоторых из этих файлов будет рассказано более подробно. Полная информация о структуре модулей и проекта находится в [6].

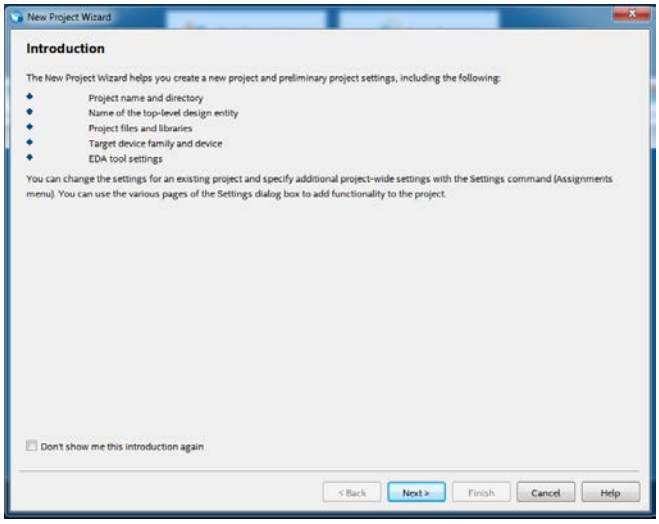



Рис.16. Определение выполняемых действий на первом этапе создания проекта

**Шаг 1.** Создадим рабочую папку **Quartus_Prime_work** на рабочем диске компьютера, в которой будем сохранять файлы различных проектов. В этой папке для текущего проекта по созданию КЛС создадим другую папку с именем **comblogic**. Такие же названия будут у проекта и у файла верхнего уровня – **comblogic** (рис. 17). Название папки может отличаться от названия проекта, названия проекта и файла модуля верхнего уровня проекта должны быть одинаковы. Выберем папку проекта, запишем имя проекта и имя файла модуля верхнего уровня. После этого нажимаем кнопку Next >.

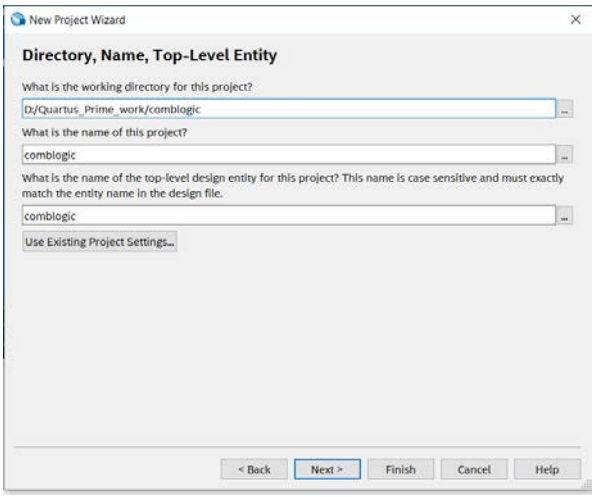



Рис. 17. Выбор папки проекта, названия проекта и файла модуля верхнего уровня

**Шаг 2.** Определяем, какой проект мы создаем: новый (empty project) или воспользуемся шаблоном, который можно скачать с сайта фирмы Intel. Выберем создание нового проекта (рис. 18) и нажимаем кнопку Next >.

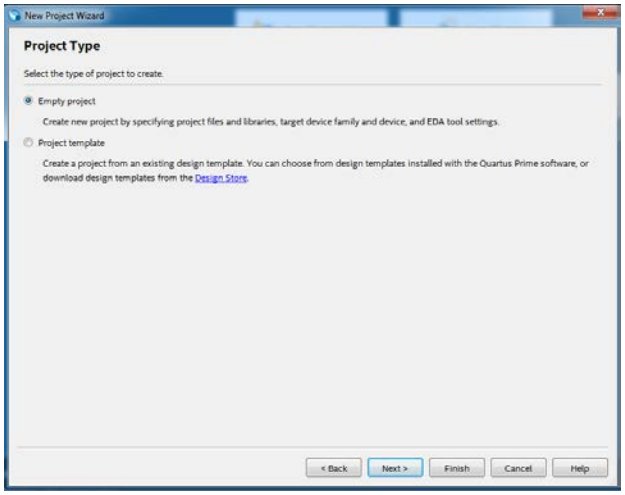



Рис. 18. Создание нового проекта

**Шаг 3.** Определяем, какие дополнительные файлы (например, из предыдущих проектов) мы хотим добавить в текущий проект (рис. 19). Поскольку это первый проект, то необходимо просто нажать кнопку Next >.

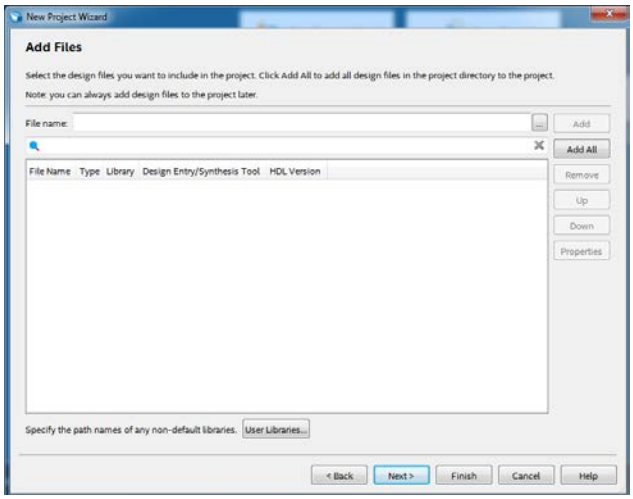



Рис. 19. Выбор дополнительных файлов

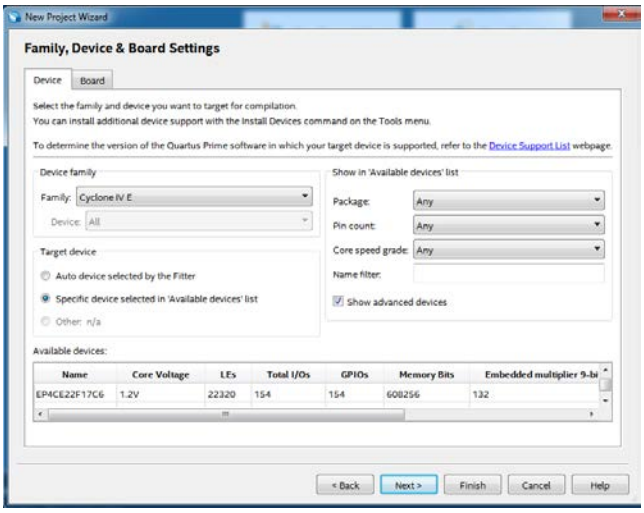



Рис. 20. Выбор семейства ПЛИС и кристалл

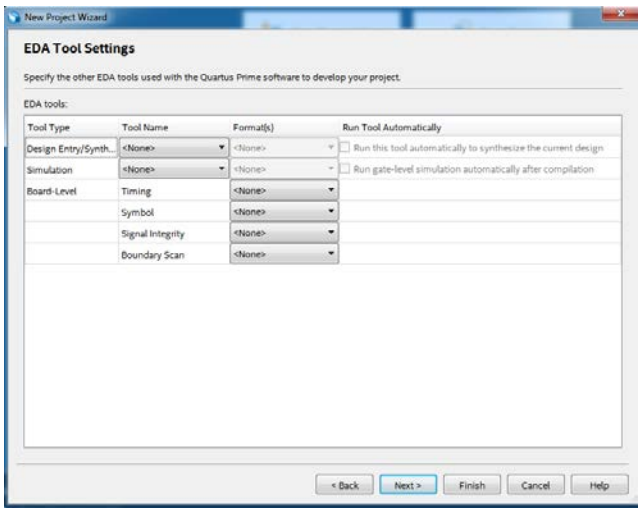



Рис. 21. Выбор дополнительных инструментов

**Шаг 4.** Выбор семейства ПЛИС и кристалла фирмы Intel. Рассмотрим основные критерии при выборе ПЛИС для реализации проектов. Как известно, при выборе элементной базы необходимо руководствоваться следующими критериями отбора:

- быстродействие;
- логическая емкость, достаточная для реализации алгоритма;
- схемотехнические и конструктивные параметры ПЛИС, надежность, рабочий диапазон температур и т.д.;
- стоимость владения средствами разработки, включающая как стоимость программного обеспечения, так наличие и стоимость аппаратных средств отладки;
- стоимость оборудования для программирования ПЛИС или конфигурационных ПЗУ;
- наличие методической и технической поддержки, т.е. наличие комплекта для разработки;
- стоимость микросхем;
- другие дополнительные требования.

Наш проект мы будем реализовывать на базе комплекта разработчика DEO_Nano. В состав комплекта интегрирован кристалл (ПЛИС) семейства Cyclone® IV EP4CE22F17C6N фирмы Intel. Таким образом, необходимо выбрать (рис. 20) семейство (family) Cyclone IV E и кристалл EP4CE22F17C6N. После этого нажимаем кнопку Next >.

**Шаг 5.** Выбор и установка дополнительного программного обеспечения (рис. 21). В проекте мы не будем использовать дополнительные средства автоматизации проектирования электронных устройств (Electronic Design Automation, **EDA**), поэтому нажимаем кнопку Next >. Откроется окно, в котором собраны все данные, полученные в результате выполнения пяти шагов.

Если проверка не обнаружит ошибок, то можно нажать кнопку Finish. Откроется следующий набор окон пакета (рис. 22): окно для ввода в схемном редакторе (или на языках высокого уровня) элементов схемы (справа), окно для отображения процесса компиляции (слева посередине), окно с информацией об используемых файлах (слева вверху) и окно с комментариями и сообщениями (например, об ошибках), которые появляются в процессе компиляции (внизу).

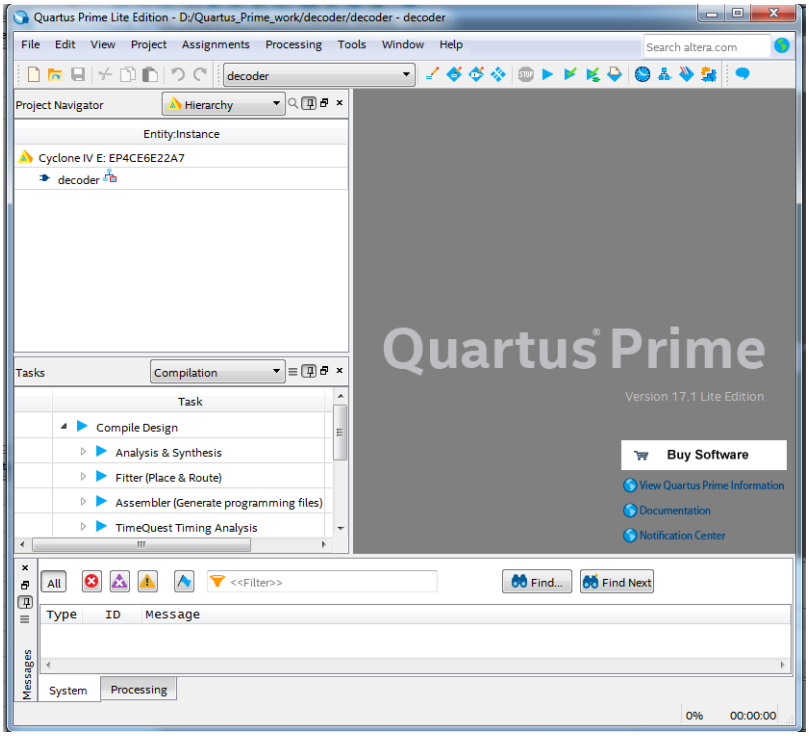



Рис. 22. Рабочие окна пакета **Quartus_Prime Lite Edition**

### 3.3.3. Создание проекта комбинационной логической схемы в схемном редакторе Quartus Prime

В предыдущем параграфе с помощью мастера по созданию нового проекта (New Project Wizard) была создана заготовка проекта, т.е. определена рабочая папка проекта, название проекта и файла верхнего уровня (**comblogic**), выбрано семейство ПЛИС (Cyclone IV E) и кристалл (EP4CE22F17C6N).

Следующим этапом необходимо в схемном редакторе пакета Quartus Prime создать (нарисовать) комбинационную логическую схему на основании синтезированных логических уравнений (§ 3.3.1). Для этого выберем в меню File/New создание нового файла. В открывшемся окне New (рис. 23) выберем Block Diagram/Schematic File. После нажатия на кнопку OK откроется рабочее поле, где будем разрабатывать схему КЛС. Для создания схемы воспользуемся библиотекой примитивов, которая включает огромное количество цифровых элементов (логические элементы, счетчики, регистры, защелки и т.д. Для ввода любого элемента необходимо на рабочем поле правой кнопкой открыть контекстное меню Insert/Symbol и ввести требуемый символ. Эту операцию необходимо выполнить необходимое число раз. Выбранные логические элементы соединить необходимыми связями.

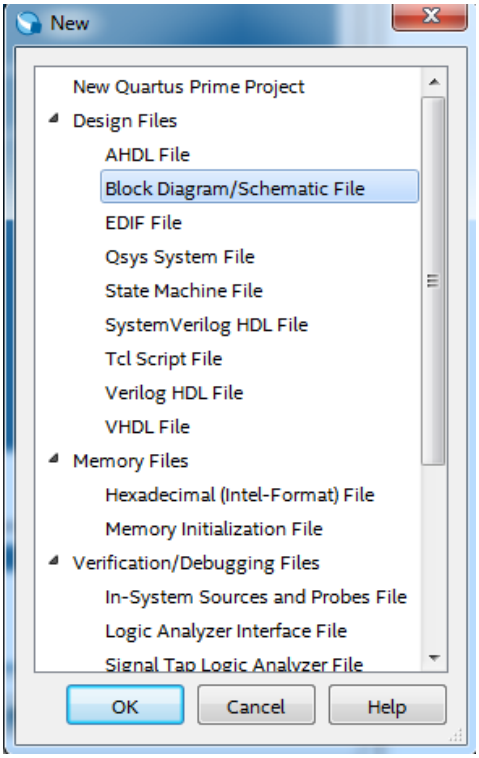



Рис. 23. Выбор типа файла проекта

В минимизированном уравнении КЛС (см. § 3.3.1) используются следующие логические элементы:

AND3, AND4, OR6 и NOT.

$$y = \overline{x_4} \,\&\, x_2 \,\&\, \overline{x_1} + x_4 \,\&\, x_3 \,\&\, x_1 + x_4 \,\&\, x_3 \,\&\, x_2 + \overline{x_4} \,\&\, x_3 \,\&\, \overline{x_1} + x_4 \,\&\, \overline{x_3} \,\&\, \overline{x_2} \,\&\, \overline{x_1}.$$

Кроме того, из библиотеки необходимо взять элементы input и output для связи КЛС с внешними электронными устройствами. Для реализации инверсии входных сигналов КЛС ($x_4$, $x_3$, $x_2$, $x_1$) используем элемент NOT и обозначим их с добавлением буквы $n$ ($nx_4$, $nx_3$, $nx_2$, $nx_1$). На схеме (рис. 24) представлена синтезированная схема КЛС.

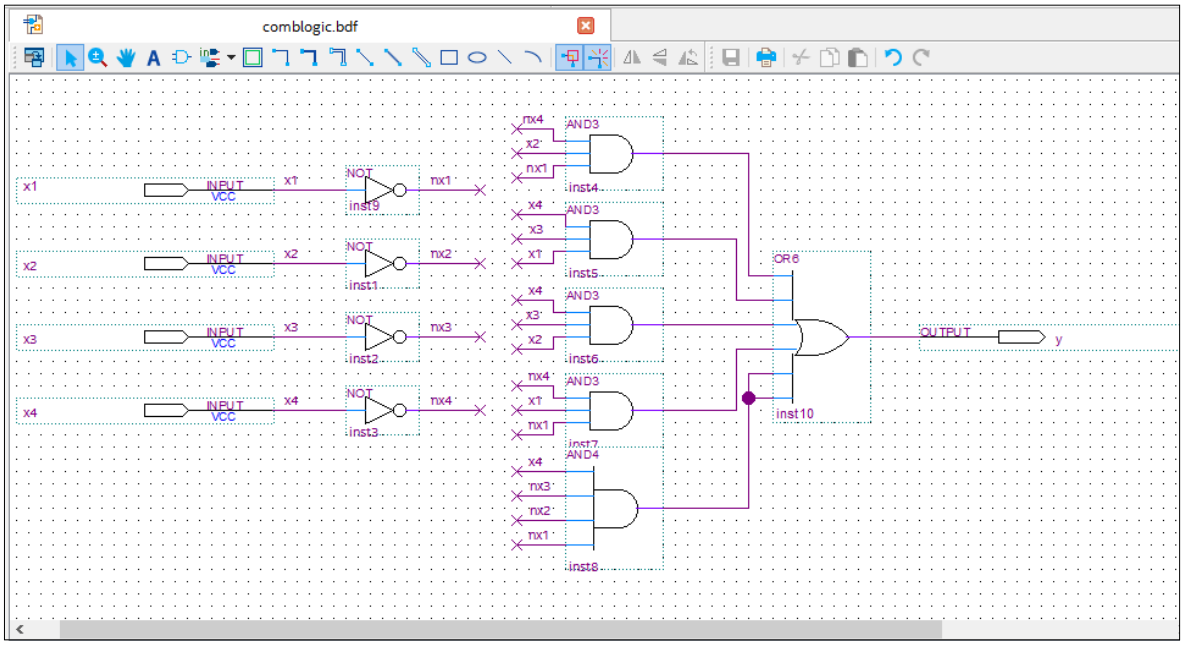



Рис. 24. Синтезированная КЛС

После создания схемы необходимо выполнить компиляцию проекта. Для этого необходимо выбрать пункты меню Processing/Start Compilation (или выбрать на панели инструментов кнопку Start Compilation). Процесс компиляции можно контролировать через окно, представленное на рис. 25. В этом окне отражается процесс компиляции, состоящий из различных этапов: проверка правильности использования введенных элементов (Compile Design), выполнение анализа и синтеза схемы (Analysis & Synthesis), привязка входов и выходов схемы при размещении в реальной микросхеме (Fitter – Place & Route), преобразование исходного представления схемы (в нашем случае графическое) в код ассемблера (Assembler – Generate programming files), создание различных файлов для функциональной и временной симуляции и для программирования микросхемы (ПЛИС).

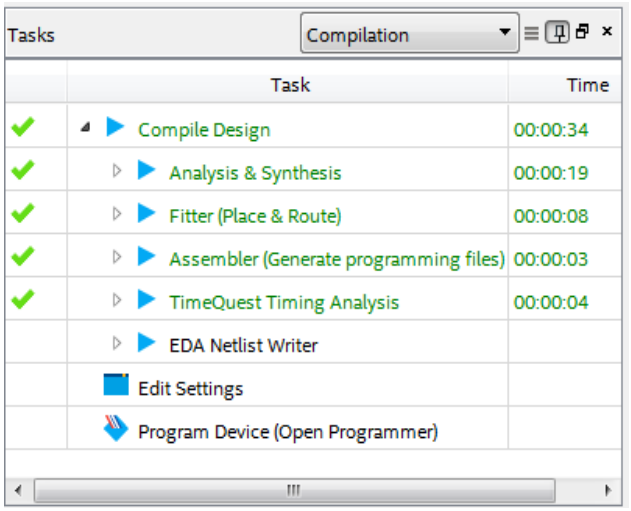



Рис. 25. Окно для контроля компиляции

По окончанию компиляции в основном окне появятся результаты компиляции с перечислением различных данных: исходные данные проекта (имя, дата компиляции и т.д.), тип семейства и микросхемы (Cyclone IV E, EP4CE22F17C6N), число логических элементов, задействованных в проекте, число регистров, число пинов и другая полезная информация. В соседнем окне можно познакомиться с подробной информацией о каждом этапе компиляции с представлением информации об ошибках и способах их устранения. Далее необходимо проверить правильность работы схемы.

Проверить можно несколькими способами. Первый способ проверки заключается в отслеживании пути сигналов от входов к выходам для всех $2n$-комбинаций входных сигналов. Для каждой такой комбинации определяются сигналы, возникающие на выходах всех вентилей под действием данных входных сигналов. В следующем параграфе пособия будет рассмотрен другой способ проверки, который заключается в функциональном и временном симулировании разработанной схемы.

### 3.3.4. Отладка проекта комбинационной логической схемы с помощью функциональной и временной симуляции

Функциональная симуляция связана с проверкой логики работы разработанной схемы. Временная симуляция позволяет проанализировать поведение выходов схемы по отношению ко входам во времени. Кроме того, с помощью временной симуляции можно определить задержки на любом внутреннем или внешнем элементе схемы. Интегрированный пакет Quartus Prime содержит достаточно много средств для функциональной и временной симуляции разработанного проекта. Рассмотрим

симуляцию с помощью программы University Program VWF (Vector Waveform File), которая входит в состав пакета. Эта программа создана специально для использования в университетских курсах по изучению цифровых устройств, реализованных на ПЛИС.

Далее выполним симуляцию работы схемы КЛС, разработанной в предыдущем параграфе. Запустим Quartus Prime и откроем проект comblogic из папки Quartus Prime_work. Далее необходимо выполнить ряд шагов по созданию файла симуляции, выбора параметров и т.д.

**Шаг 1**. Создадим файл для симуляции, для этого выберем в меню File/New. В открывшемся окне New (см. рис. 23) выберем University Program VWF. После нажатия на кнопку OK откроется новая вкладка Simulation Waveform Editor, где будем выполнять симуляцию проекта (рис. 26). Сохраним файл симуляции под именем **Waveform.vwf** (File/Save as) в папке проекта.

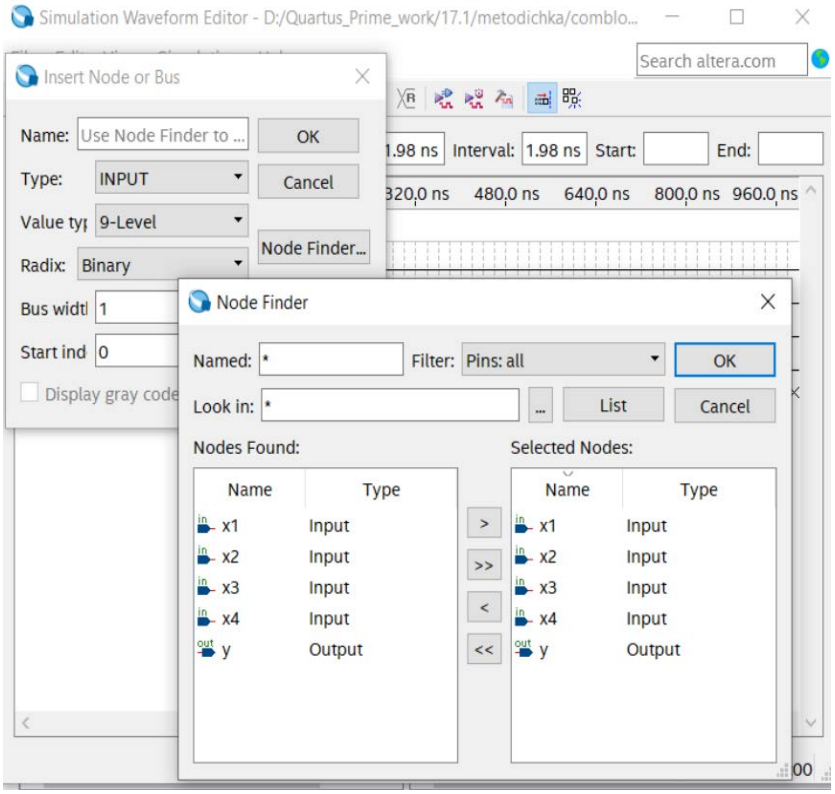



Рис. 26. Окна для ввода входов и выходов схемы

**Шаг 2.** Далее необходимо добавить на поле симуляции входы и выходы КЛС. Для этого последовательно введем команды Edit/Insert/Insert Node or Bus. Откроется окно (Insert Node or Bus), в котором необходимо выбрать кнопку Node Finder. Откроется следующее окно (Node Finder), разделенное на две части. С помощью кнопки List поместим все входы и выходы в левую часть окна и с помощью стрелок переместим необходимые для симуляции переменные. Нажимаем два раза OK.

Слева в окне для симуляции появятся выбранные входные и выходные переменные.

**Шаг 3.** Перед началом задания значений входных сигналов, изменяющихся во времени, необходимо задать два параметра симуляции: шаг временной сетки (Grid Size) и время симуляции (End Time). Шаг временной сетки определяется исходя из требований временного разрешения процессов в исследуемой схеме, а время симуляции связано с числом уникальных состояний устройства (у дешифратора три входных переменных и восемь состояний). Как правило, для удобства симулирования время симуляции выбирается в несколько раз больше числа состояний. Введем команду Edit/Grid Size и в открывшемся окне (Grid Size) в строке период (Period) установим время 10 ns. Далее введем Edit/Set End Time и в открывшемся окне (End Time) зададим время 2 ns.

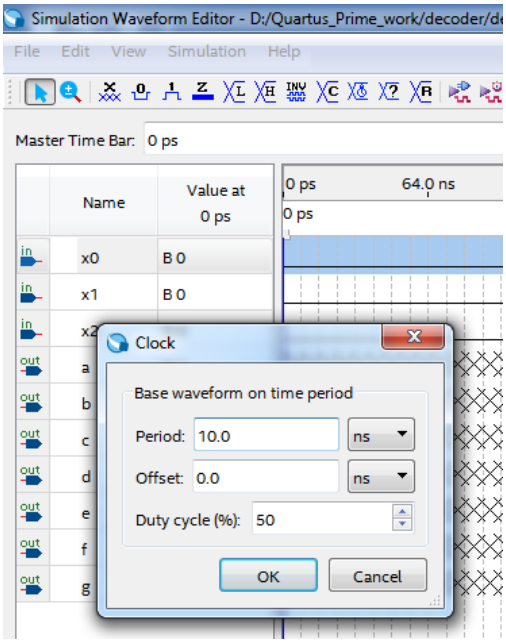



Рис. 27. Задание периода $x_1$

**Шаг 4.** Зададим значения (логические “0” и “1”) входных сигналов $x_4$, $x_3$, $x_2$, $x_1$, для этого выделим на поле введенных сигналов мышкой сигнал $x_1$ и введем команду Edit/Value/Overwrite Clock. Откроется окно (Clock), в котором установим значения периода 100,00 ns, а фазу (Offset) и скважность (Duty cycle) оставим без изменения (рис. 27). После нажатия на кнопку OK в строке переменной $x_1$ формируется периодический сигнал с периодом 100,0 ns и скважностью 50%.

Другой способ задания сигнала – это нарисовать мышкой входной сигнал, формируя логические “0” и “1”. Этот способ удобно использовать, когда сигнал не периодический и с различной скважностью.

Повторим задание значений оставшихся входных переменных $x_4$, $x_3$, $x_2$, причем период $x_1$ зададим равным 200,0 ns, а $x_2$ – 400,0 ns, т.е. период каждой последующей входной переменной возрастает в 2 раза. Далее вводим команду Simulation/Run Functional Simulation и запускаем процесс симуляции. На рисунках представлен результат функциональной (рис. 28) и временной симулировании (рис. 29).

На основание функциональной симуляция проверяется правильность реализации таблицы истинности заданной комбинационной схемы без учета временных задержек.

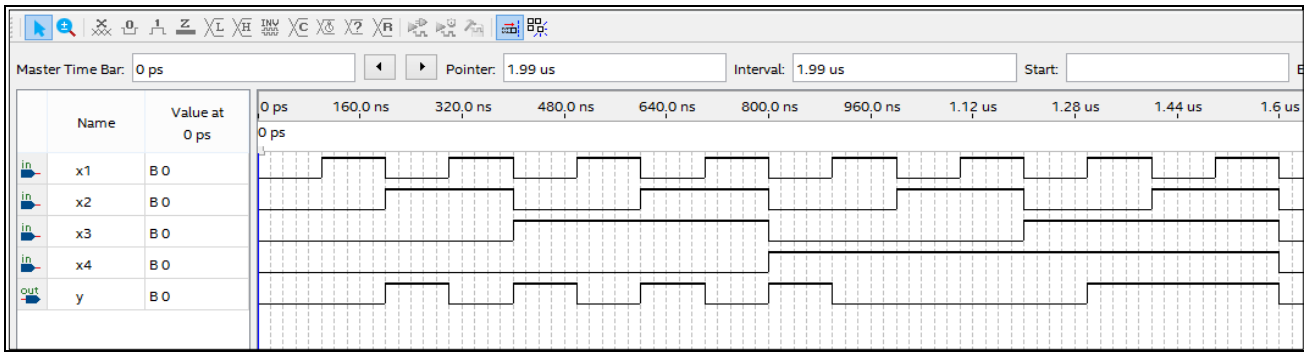



Рис. 28. Функциональное симулирование синтезированной КЛС

При выполнении временного анализа можно посмотреть задержки между появлением входных сигналов и результирующих выходных. На рис. 29 маркер (вертикальная черта) установлен на переднем нарастающем фронте сигнала $x_2$ (отметка на временной шкале 200,0 ns – в окошке Master Time Bar). Подведя мышку к нарастающему фронту (обозначено стрелкой) выходной переменной $y$, можно измерить временную задержку между маркером и передним фронтом переменной $y$ (значение 6,5 ns – в окошке Interval).

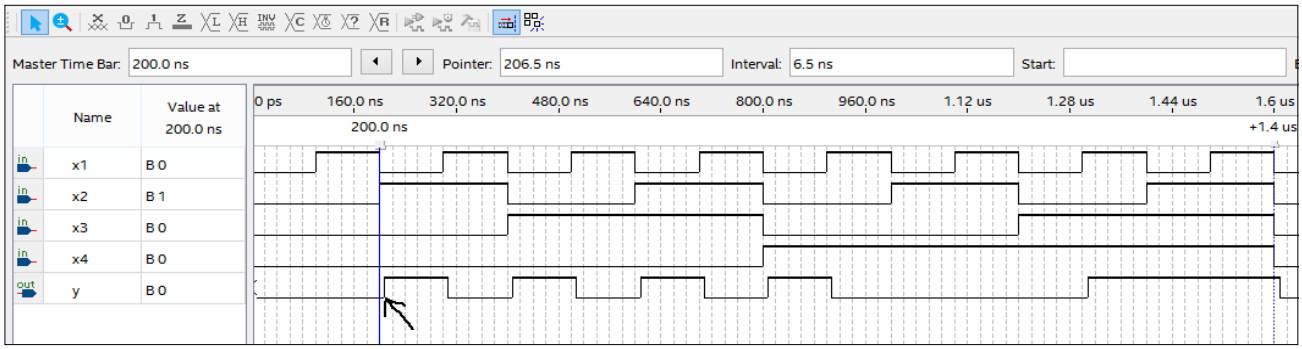



Рис. 29. Временное симулирование синтезированной КЛС

### 3.3.5. Синтез комбинационной схемы дешифратора

Дешифратор – это комбинационная схема с несколькими входами и несколькими выходами, которая преобразует кодированные входные сигналы в кодированные выходные сигналы, причем входные и выходные коды различны. Входной код обычно имеет меньшее число разрядов, чем выходной код, и между входными и выходными кодовыми словами имеется взаимно-однозначное соответствие. В большинстве случаев роль входного кода играет $n$-разрядный двоичный код, где $n$-разрядное двоичное слово представляет одну из $2^n$ различных кодированных величин. Самым распространенным является дешифратор $n \times 2^n$ или полный дешифратор.

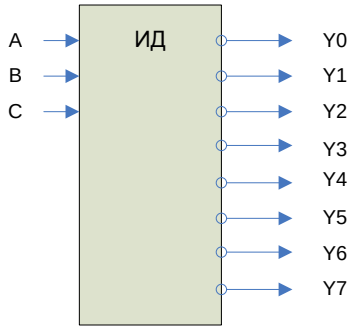



Рис. 30. Функциональная схема полного дешифратора 3×8

Полный дешифратор применяется в том случае, если необходимо активизировать только один из $2^n$-выходов. Каждой комбинации логических уровней на входах будет соответствовать активный уровень на одном из $2^n$-выходов. Обычно $n$ равно 2, 3 или 4. На рис. 30 изображен дешифратор с $n = 3$ входами и $2^n = 2^3 = 8$ выходами. Активным выходным уровнем является уровень логического нуля. На входы C, B, A можно подать следующие комбинации логических уровней: 000, 001, 010, ..., 111, всего 8 комбинаций.

Схема имеет 8 выходов, на одном из которых формируется низкий потенциал, на остальных – высокий. Номер этого единственного выхода, на котором формируется активный (нулевой) уровень, соответ-

ствует числу $n$, определяемому состоянием входов C, B, A следующим образом:

$$Y_j = \overline{m_j(C,B,A)}, \qquad (19)$$

где $j$ = (0,7), $m_j$ – конституанты единицы переменных C, B, A, соответствующего номера. Например, для номера $j$ = 0 выражение (19) будет иметь вид:

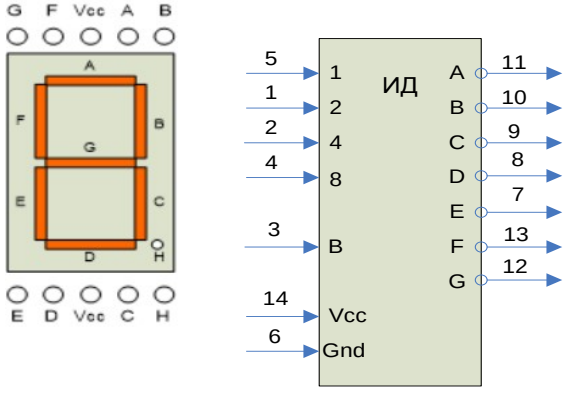
$$Y_0 = \overline{\overline{C} \cdot \overline{B} \cdot \overline{A}},$$

а для $j$ = 3:

$$Y_3 = \overline{\overline{C} \cdot B \cdot A}.$$

Таким образом, если на входы подана комбинация логических уровней 011, то из восьми выходов микросхемы ($Y_0$, $Y_1$, ..., $Y_7$) на выходе $Y_3$ установится логический ноль ($Y_3$ = 0), а все остальные выходы будут иметь уровень логической единицы. Этот дешифратор имеет отрицательный активный выходной сигнал. Дешифраторы могут иметь несколько стробирующих входов.

Дешифраторы изготавливаются в интегральном исполнении и имеют различное функциональное назначение. Например, дешифратор КР514ИД2 является дешифратором семисегментного кода и используется для преобразования двоичного или двоично-десятичного кода в код формирования символов на семисегментных индикаторах. Графическое изображение семисегментного индикатора с общим анодом, расположение выводов и их функции микросхемы КР514ИД2 и неполная таблица истинности дешифратора представлены на рис. 31. Поскольку выходы КР514ИД2 инверсные, а индикатор имеет общие аноды (то есть аноды светодиодов всех сегментов объединены вместе), то гореть светодиоды сегментов будут в том случае, если на соответствующем выходе дешифратора будет логический “0”.




| Входы | | | | | Выходы | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Выбор | | | | B | (0) | (1) | (2) | (3) | (4) | (5) | (6) | Символ |
| 8 | 4 | 2 | 1 | | A | B | C | D | E | F | G | |
| X | X | X | X | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | |
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 2 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 3 |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 4 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 5 |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 6 |
| 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 7 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 8 |

Рис. 31. Графическое изображение семисегментного индикатора (слева направо), функциональной схемы дешифратора КР514ИД2 и таблицы истинности

Выполним синтез дешифратора семисегментного индикатора, заданного таблицей истинности (табл. 7).

Таблица 7. **Таблица истинности дешифратора**

| $x_2$ | $x_1$ | $x_0$ | A | B | C | D | E | F | G | Символ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 2 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 3 |
| 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 4 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 5 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 6 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 7 |

1. Напишем СДНФ для функции каждого выхода дешифратора:

$$Y_0 = A = \overline{x}_2\overline{x}_1x_0 + x_2\overline{x}_1\overline{x}_0,$$

$$Y_1 = B = x_2\overline{x}_1x_0 + x_2x_1\overline{x}_0,$$

$$Y_2 = C = \overline{x}_2x_1\overline{x}_0,$$

$$Y_3 = D = \overline{x}_2\overline{x}_1x_0 + x_2\overline{x}_1\overline{x}_0 + x_2x_1x_0,$$

$$Y_4 = E = \overline{x}_2\overline{x}_1x_0 + \overline{x}_2x_1x_0 + x_2\overline{x}_1\overline{x}_0 + x_2\overline{x}_1x_0 + x_2x_1x_0,$$

$$Y_5 = F = \overline{x}_2\overline{x}_1x_0 + \overline{x}_2x_1\overline{x}_0 + \overline{x}_2x_1x_0 + x_2x_1x_0,$$

$$Y_6 = G = \overline{x}_2\overline{x}_1\overline{x}_0 + \overline{x}_2\overline{x}_1x_0 + x_2x_1x_0.$$

2. Минимизируем получившиеся функции выходов с помощью карт Карно. Функции выходов A, B, C, D уже имеют минимальную форму, поэтому построим карты Карно для выходов E, F и G.

Для функции выходов E

| x2 \ x1x0 | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 |

Склеивая соответствующие наборы, получим: $E = x_0 + x_2\bar{x}_1$.

Для функции выходов F

| x2 \ x1x0 | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 |

После склейки получаем: $F = \bar{x}_2x_1 + \bar{x}_2x_0 + x_1x_0$.

Для функции выходов G

| x2 \ x1x0 | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |

Склеивая наборы, получим: $G = \bar{x}_2\bar{x}_1 + x_2x_1x_0$.

Таким образом, получили минимальные функции выходов A, B, C, D, E, F, G дешифратора:

$A = \bar{x}_2\bar{x}_1x_0 + x_2\bar{x}_1\bar{x}_0,$
$B = x_2\bar{x}_1x_0 + x_2x_1\bar{x}_0,$
$C = \bar{x}_2x_1\bar{x}_0,$
$D = \bar{x}_2\bar{x}_1x_0 + x_2\bar{x}_1\bar{x}_0 + x_2x_1x_0,$
$E = x_0 + x_2\bar{x}_1,$
$F = \bar{x}_2x_1 + \bar{x}_2x_0 + x_1x_0,$
$G = \bar{x}_2\bar{x}_1 + x_2x_1x_0.$

Анализируя получившиеся выражения, мы видим, что в разных функциях выходов есть однородные слагаемые. При составлении схемы это необходимо учесть. На основании полученных уравнений создадим проект дешифратора в схемном редакторе пакета Quartus Prime.

### 3.3.6. Создание проекта комбинационной схемы дешифратора в схемном редакторе пакета Quartus Prime

В § 3.3.2 и 3.3.3 с помощью мастера по созданию нового проекта (New Project Wizard) была создана заготовка проекта, т.е. определена рабочая папка проекта, название проекта и файла верхнего уровня (**comblogic**), выбрано семейство ПЛИС (Cyclone IV E) и кристалл (EP4CE22F17C6N). Создадим новую папку **decoder** и выполним необходимые действия для проекта **decoder**.

Следующим этапом необходимо в схемном редакторе пакета Quartus Prime создать (нарисовать) комбинационную схему дешифратора на основании синтезированных логических уравнений (§ 3.3.5). Для этого выберем в меню File/New создание нового файла. В открывшемся окне New (рис. 32) выберем Block Diagram/Schematic File. После нажатия на кнопку OK откроется рабочее поле, где будем разрабатывать дешифратор. Для создания схемы воспользуемся библиотекой примитивов, которая включает огромное количество цифровых элементов (логические элементы, счетчики, регистры, защелки и т.д.). Для ввода любого элемента необходимо на рабочем поле правой кнопкой открыть контекстное меню Insert/Symbol и ввести требуемый символ. Эту операцию необходимо выполнить необходимое число раз. Выбранные логические элементы соединить необходимыми связями.

В минимизированных уравнениях (§ 3.3.5) функций выходов дешифратора синтезированы следующие логические элементы: NOT, AND3, AND2, OR3, OR2. Кроме того, из библиотеки необходимо взять элементы input и output для связи дешифратора с внешними электронными устройствами.

Для реализации инверсии входных сигналов ($x_0$, $x_1$, $x_2$) используем элемент NOT и обозначим их с добавлением буквы *n* ($nx_0$, $nx_1$, $nx_2$). При разработке схемы было учтено, что некоторые элементы логического умножения повторяются в разных выходных функциях (например, $x_2\bar{x}_1\bar{x}_0$ и $\bar{x}_2\bar{x}_1x_0$). В этом случае нет необходимости загружать из библиотеки дополнительный элемент AND3, а следует использовать выходную функцию уже созданного элемента. На схеме рис. 33 выходные сигналы элементов AND3 (inst8 и inst3) используются как входные элемента OR3 (inst18) при формировании выходного сигнала сегмента "*d*".

После создания схемы необходимо выполнить компиляцию проекта. Для этого необходимо выбрать пункты меню Processing/Start Compilation (или выбрать на панели инструментов кнопку Start Compilation).

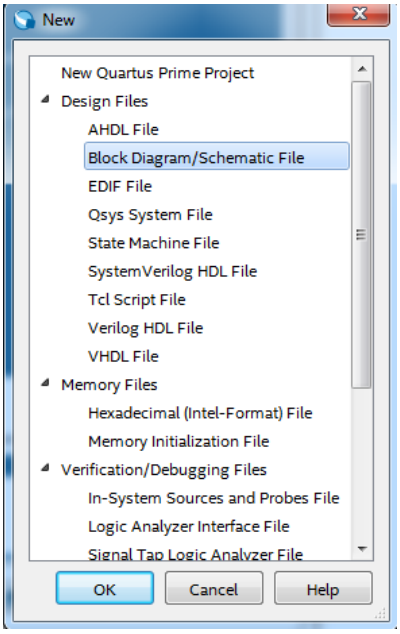


Рис. 32. Выбор типа файла проекта

По окончанию компиляции в основном окне появятся результаты компиляции с перечислением различных данных: исходные данные проекта (имя, дата компиляции и т.д.), тип семейства и микросхемы (Cyclone IV E, EP4CE22F17C6N), число логических элементов, задействованных в проекте, число регистров, число пинов и другая полезная информация. В соседнем окне можно познакомиться с подробной информацией о каждом этапе компиляции с представлением информации об ошибках и способах их устранения.

Далее необходимо проверить правильность работы схемы методом функциональной и временной симуляции проекта. Для этого создадим файл для симуляции через меню File/New. В открывшемся окне New (см. рис. 23, 26) выберем University Program VWF. После нажатия на кнопку OK откроется новая вкладка Simulation Waveform Editor, где будем выполнять симуляцию проекта (рис. 26). Сохраним файл симуляции под именем **Waveform.vwf** (File/Save as) в папке проекта. Порядок задания входных воздействий (вектора входных кодов), шаг временной сетки (Grid Size) и время симуляции (End Time) подробно описаны в § 3.3.4. После ввода команды Simulation/Run Functional Simulation запускается процесс симуляции. На рис. 34 представлен результат функционального симулирования.

Для проверки значения выходных сигналов (переменные *a*, *b*, *c*, *d*, *e*, *f*, *g*) удобно использовать курсор (на рис. 35 вертикальная линия), который перемещается с помощью стрелок. Например, если установить курсор на отметке 50 ns (нарастание импульса переменной $x_0$), то в столбце Value at 50,0 ns можно увидеть значения всех выходов в двоичном коде. Перемещая курсор, можно проверить соответствие состояния входов и выходов дешифратора таблице истинности.

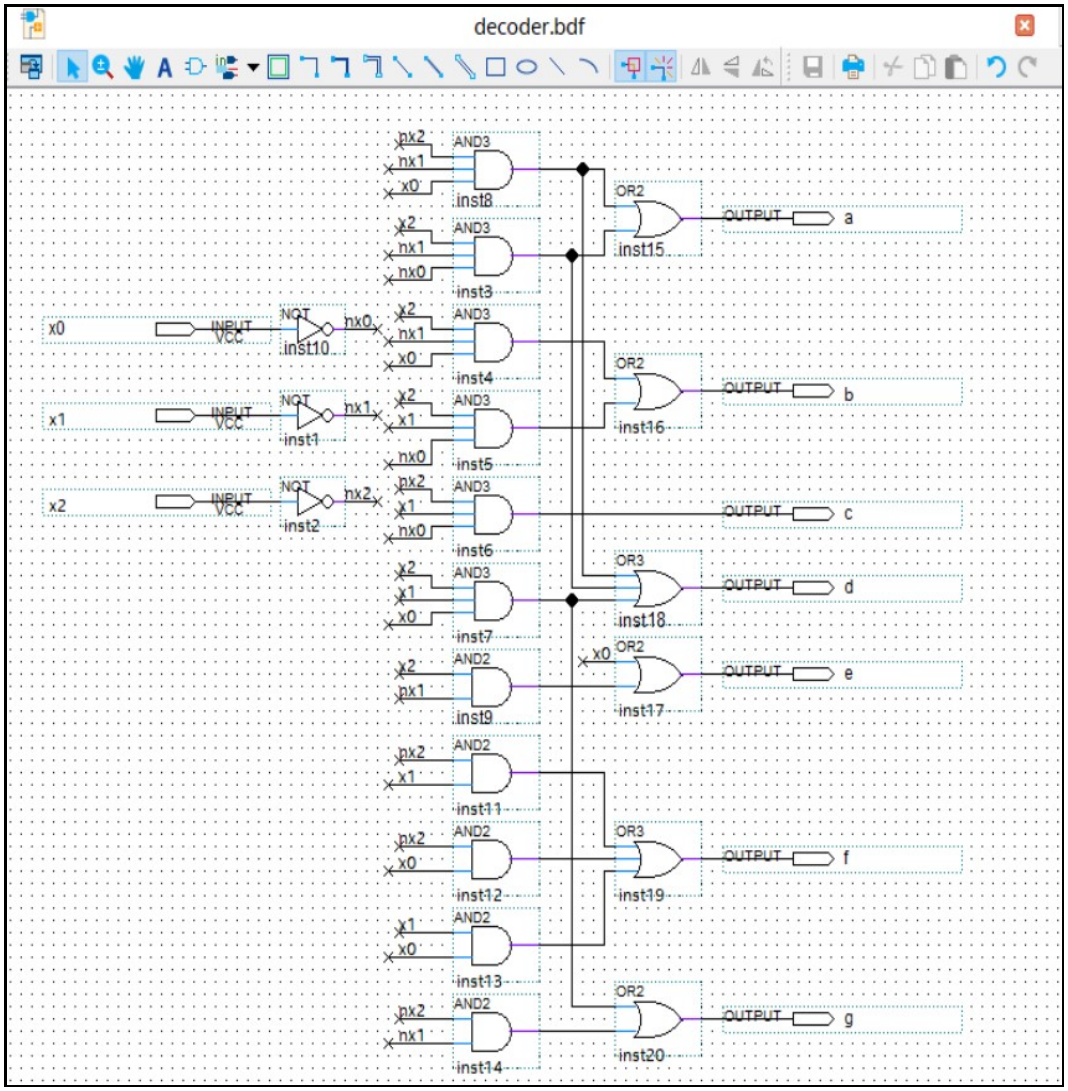



Рис. 33. Схема дешифратора семисегментного кода

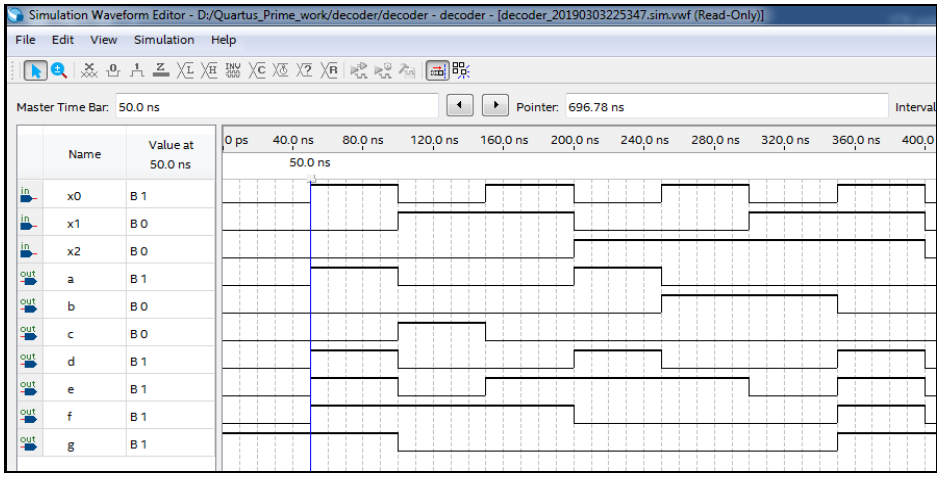



Рис. 34. Результат функционального симулирования схемы дешифратора

На осциллограмме рис. 34 фронты импульсов отдельных сигналов строго синхронны. В реальной схеме такого быть не может, поэтому необходимо выполнить временное симулирование. Вводим команду Simulation/Run Timing Simulation и запускаем процесс симуляции. Результаты временной симуляции представлены на рис. 35.

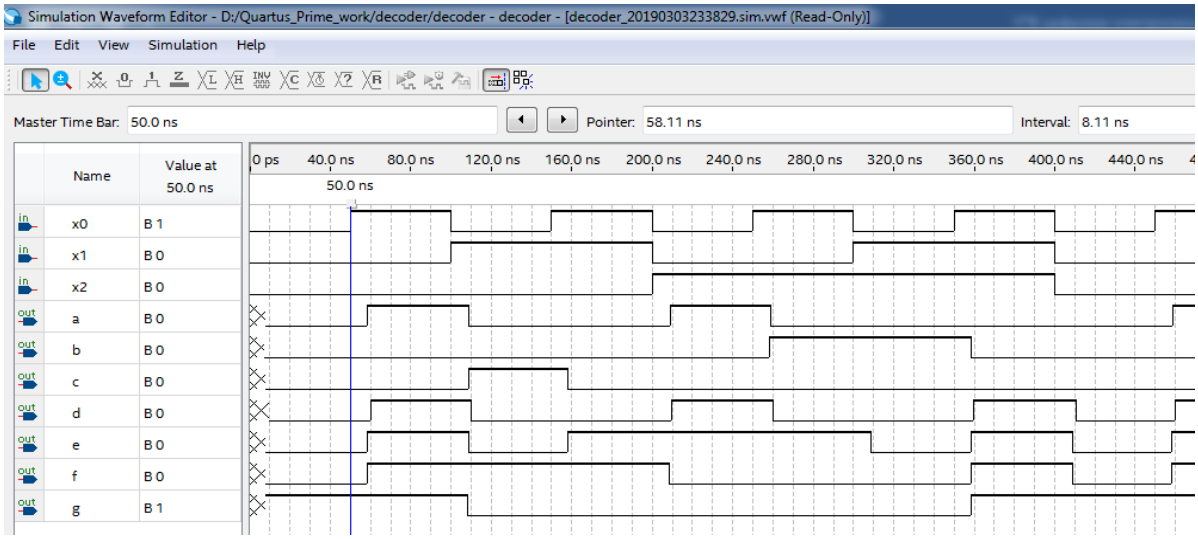



Рис. 35. Временное симулирование схемы дешифратора

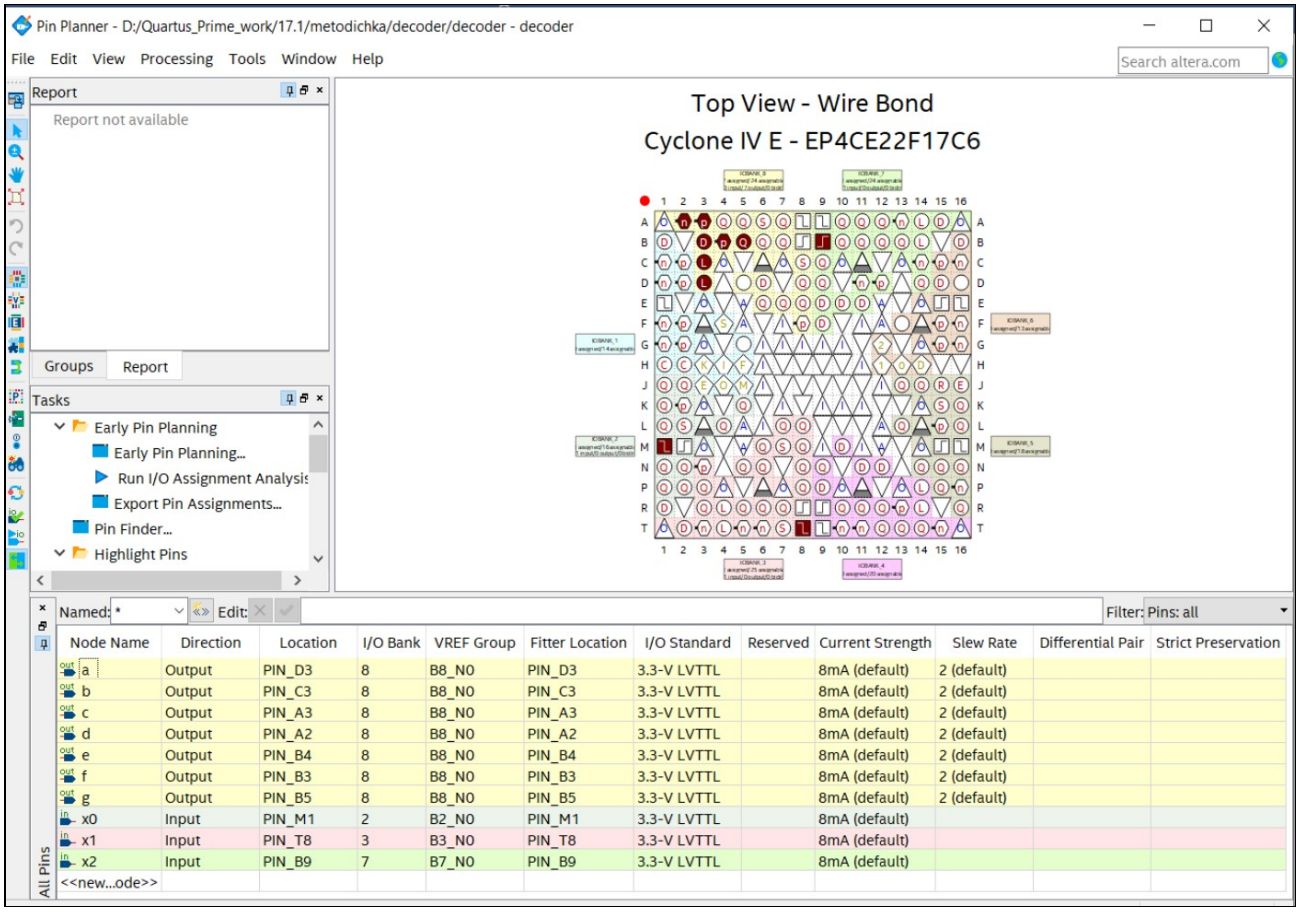
Анализируя временную диаграмму, можно увидеть, что существует задержка между появлением импульса переменной $x_0$ на отметке 50 ns и импульсов переменных *a*, *d*, *e*, *f*. Эту задержку можно измерить между установленным курсором в окне симуляции и курсором мышки. Если поставить курсор мышки на нарастающем фронте импульса сигнала переменной *a* (или *d*, *e*, *f*), то в окошке Interval будет значение задержки 8,11 ns.

### 3.3.7. Настройка входов и выходов проекта decoder для загрузки в ПЛИС учебного стенда

После компиляции проекта **decoder**, проверки работы дешифратора с помощью функционального и временного симулирования (см. § 3.3.6) необходимо выполнить несколько настроек для входов и выходов дешифратора. Проект decoder использует всего три входа (переменные $x_2$, $x_1$, $x_0$) и семь выходов (*a*, *b*, *c*, *d*, *e*, *f*, *g*). Таким образом, для реализации проекта необходимо всего 10 выводов ПЛИС. Но ПЛИС стенда CLE148 Cyclone IV EP4CE22F17C6N имеет 154 вывода общего назначения (ввода/вывода). При компиляции проекта необходимо определить (настроить), в каком состоянии будут находиться неиспользуемые выводы ПЛИС. Для этого в Quartus Prime выберем в меню закладку Assignments/Device→Device and Pin Options, а затем выберем категорию Unused Pins. Пакет Quartus Prime позволяет выбрать несколько вариантов определения резервных (неиспользуемых в проекте) пинов ПЛИС: “As input tri-stated with weak pull up” (как вход с тремя состояниями с подтягивающими к питанию резисторами), “As input tri-stated” (как вход с тремя состояниями), “As input tri-stated with bus-hold circuitry” (как вход с тремя состояниями с удержанием шины), “As output driving an unspecified signal” (как выход, который управляет неопределенным сигналом), “As output driving ground” (как выход подключенный к «земле», минусовому полюсу питания). Для выбора того или иного варианта необходимо знать, как разведены выводы ПЛИС на печатной плате. Самый безопасный вариант определения резервных выводов – это “As input tri-stated with weak pull up”. При этом варианте выводы будут определены как входы в высокоимпедансном состоянии с небольшой подтяжкой к питанию. Это означает, что на этих входах всегда будет логическая «1». Выберем этот вариант настройки неиспользуемых выводов ПЛИС и нажмем OK.

Далее необходимо присвоить входным и выходным переменным дешифратора конкретные адреса выводов ПЛИС. Для задания двоичных значений входным переменным $x_2$, $x_1$, $x_0$ будем использовать движковые переключатели (DIP Switch [2...0]) стенда, адреса которых представлены в табл. 5. Как указано в описании стенда (§ 2.1), DIP-переключатель обеспечивает на входе ПЛИС высокий логический уровень, когда он находится в нижнем положении, и низкий уровень логики в верхнем положении. Адреса выводов ПЛИС (которые будут использоваться как выходы) для управления сегментами семисегментного индикатора (*a*, *b*, *c*, *d*, *e*, *f*, *g*), расположенного на модуле расширения ДМ-1, представлены в Паспорте ДМ-1.

Для задания адресов используемых выводов ПЛИС (в нашем случае Cyclone IV EP4CE22F17C6N) в Quartus Prime применяется специальный инструмент, названный “Pin Planner”. Этот инструмент можно вызвать по ссылке Assignments → Pin Planner. На рис. 36 изображено окно инструмента Pin Planner.




| Node Name | Direction | Location | I/O Bank | VREF Group | Fitter Location | I/O Standard | Reserved | Current Strength | Slew Rate | Differential Pair | Strict Preservation |
|---|---|---|---|---|---|---|---|---|---|---|---|
| a | Output | PIN_D3 | 8 | B8_N0 | PIN_D3 | 3.3-V LVTTL | | 8mA (default) | 2 (default) | | |
| b | Output | PIN_C3 | 8 | B8_N0 | PIN_C3 | 3.3-V LVTTL | | 8mA (default) | 2 (default) | | |
| c | Output | PIN_A3 | 8 | B8_N0 | PIN_A3 | 3.3-V LVTTL | | 8mA (default) | 2 (default) | | |
| d | Output | PIN_A2 | 8 | B8_N0 | PIN_A2 | 3.3-V LVTTL | | 8mA (default) | 2 (default) | | |
| e | Output | PIN_B4 | 8 | B8_N0 | PIN_B4 | 3.3-V LVTTL | | 8mA (default) | 2 (default) | | |
| f | Output | PIN_B3 | 8 | B8_N0 | PIN_B3 | 3.3-V LVTTL | | 8mA (default) | 2 (default) | | |
| g | Output | PIN_B5 | 8 | B8_N0 | PIN_B5 | 3.3-V LVTTL | | 8mA (default) | 2 (default) | | |
| x0 | Input | PIN_M1 | 2 | B2_N0 | PIN_M1 | 3.3-V LVTTL | | 8mA (default) | | | |
| x1 | Input | PIN_T8 | 3 | B3_N0 | PIN_T8 | 3.3-V LVTTL | | 8mA (default) | | | |
| x2 | Input | PIN_B9 | 7 | B7_N0 | PIN_B9 | 3.3-V LVTTL | | 8mA (default) | | | |
| <<new...ode>> | | | | | | | | | | | |

Рис. 36. Окно Pin Planner для настройки выводов ПЛИС проекта **decoder**

В центре окна располагается схематичное представление ПЛИС Cyclone IV EP4CE22F17C6N. Изображение корпуса микросхемы можно выводить в двух режимах: Top View и Bottom View. По умолчанию выводится изображение Top View, при этом первый вывод микросхемы располагается слева вверху. В нижней части Pin Planner расположено окно All Pins, организованное в виде таблицы. Рассмотрим основной функционал инструмента для настройки выводов ПЛИС. После компиляции проекта в левом столбце All Pins – Node Name будут отображаться названия всех переменных, являющихся входами или выходами ПЛИС-проекта. В столбце таблицы Direction автоматически определяется направление сигнала – input или output. В столбец таблицы Location необходимо занести адреса выводов ПЛИС, которые можно взять из описания стенда (или на рис. 36). Важный раздел таблицы – столбцы I/O Standard и Current Strength. Можно выбрать в широком диапазоне амплитуду входных и выходных сигналов, а также ток выходных сигналов. Выберем эти параметры так, как представлено на рис. 36. Для сохранения назначений контактов ПЛИС в отдельном файле в окне Pin Planner выберем File/Export и сохраним с именем **decoder.csv**. Файл **decoder.csv** будет содержать информацию о назначениях контактов и может быть импортирован в другие проекты. Выполнив все эти настройки, необходимо еще раз запустить компиляцию проекта **decoder**.

### 3.3.8. Загрузка проекта decoder в ПЛИС учебного стенда

Как уже было сказано ранее, ПЛИС в учебном стенде CLE148 Cyclone IV EP4CE22F17C6N относится к Field-Programmable Gate Array – «программируемая пользователем вентильная матрица». Это означает, что конфигурация нашего проекта сохраняется в ячейках статической памяти, изготовленной по стандартной технологии CMOS. Поскольку память является энергозависимой, то при включении питания необходимо вновь загружать конфигурацию проекта. Достоинством такой технологии является многократное (практически неограниченное) конфигурирование ПЛИС. При загрузке конфигурации проектов в ПЛИС учебного стенда используется специальный загрузочный кабель, который связывает персональный компьютер (ПК) и ПЛИС на печатной плате стенда.

Загрузочный кабель фирмы Intel (download cable) может иметь нескольких видов [23]. Более ранние версии загрузочного кабеля подключались к параллельному порту (принтерный порт ПК), но уже современные версии используют USB-порт. В случае USB-версии загрузочного кабеля на печатной плате должна быть аппаратная поддержка USB-интерфейса и интерфейсов программирования ПЛИС. Одним из удачных и многофункциональных программаторов (загрузочный кабель плюс ап-

паратная поддержка в отдельном корпусе), разработанный фирмой Altera (Intel), является программатор, названный USB-Blaster [24]. Этот программатор имеет три интерфейса программирования: Joint Test Action Group (JTAG) – специализированный аппаратный интерфейс на базе стандарта IEEE 1149.1 [6] для программирования и отладки сложных цифровых микросхем и устройств, интерфейсы Active Serial и Passive Serial. Для учебных целей вполне достаточно загрузки ПЛИС от ПК через USB-порт, используя JTAG интерфейс. Но если мы хотим обеспечить созданному цифровому устройству автономность, то необходимо использовать постоянную конфигурационную память и один из интерфейсов – Active Serial или Passive Serial.

Для конфигурации FPGA в автономных устройствах используются постоянные запоминающие устройства (ПЗУ) параллельного и последовательных типов. Для загрузки в основном применяются два способа: загрузка из внешнего параллельного ПЗУ или микропроцессора (режим Passive Parallel) и загрузка из последовательного конфигурационного ПЗУ (режимы Passive Serial и Active Serial). Фирма Intel выпускает последовательные конфигурационные ПЗУ, поддерживающие различные режимы конфигурации. Выпускаются и однократно программируемые, и репрограммируемые конфигурационные ПЗУ. Репрограммируемые конфигураторы могут быть прошиты непосредственно на плате целевого устройства по последовательному интерфейсу [25]. Семейство FPGA фирмы Intel Cyclone кроме режимов Passive Parallel и Passive Serial поддерживают еще один режим конфигурации – Active Serial. Режим Active Serial поддерживается последовательными конфигурационными ПЗУ EPCS1, EPCS4, EPCS16, EPCS64. Режим Active Serial имеет две отличительные особенности: источником тактового сигнала конфигурации (DCLK) является FPGA (в отличие от режима Passive Serial, где источником тактового сигнала в этом режиме является конфигурационное ПЗУ или загрузочный кабель); конфигурационное ПЗУ программируется через те же выводы, через которые оно загружает проект в FPGA, что позволяет сделать программатор универсальным.

На плате развития DEO_Nano, которая входит в состав стенда CLE148 для автономной загрузки FPGA используется последовательное конфигурационное ПЗУ EPCS64. Таким образом, для того чтобы наше цифровое устройство работало автономно (не нужно было каждый раз для загрузки ПЛИС использовать ПК), необходимо предварительно записать в конфигурационное ПЗУ «образ» нашего проекта **decoder**.

Рассмотрим оба варианта загрузки (конфигурирования) с помощью программатора USB-Blaster: загрузка проекта **decoder** в ПЛИС непосредственно от ПК и запись (программирование) «образа» проекта в конфигурационную память EPCS64. Прежде чем выполнять процесс программирования (конфигурирования) ПЛИС, необходимо убедиться, что операционная система ПК (будем предполагать, что это Windows 10) имеет драйвер (driver) USB-Blaster. Подробная инструкция установки драйвера для устройства “Altera USB Blaster” находится на сайте производителя платы развития DEO_Nano фирмы Terasic [12].

### Конфигурирование FPGA от персонального компьютера через интерфейс JTAG программатора USB-Blaster

После выполнения компиляции проекта формируется большое число файлов различного назначения (смотрите папку проекта). Выходные файлы проекта, связанные с конфигурацией, находятся в папке “output files” проекта **decoder**. Для программирования статической памяти FPGA необходимо использовать файл SRAM Object File (.sof) – **decoder.sof**.

Для программирования ПЛИС интегрированный пакет Quartus Prime содержит утилиту Programmer (рис. 37), который находится в меню Tools/Programmer.

Для использования инструмента Programmer при программировании ПЛИС необходимо выполнить несколько настроек и подключений:

1. Подключите USB-кабель к портам персонального компьютера и DEO_Nano.
2. Вставьте в 96-контактный разъем стенда модуль расширения ДМ-1.
3. Запустите инструмент Programmer.
4. В окне Mode выберите JTAG.
5. Если в поле рядом с кнопкой Hardware Setup (выбор аппаратного обеспечения) отображается надпись “No Hardware”, то запустите процесс выбора кнопкой Hardware Setup. Откроется окно выбора аппаратного обеспечения. Выберите из возможного списка USB Blaster и закройте окно.
6. В открытом окне Programmer необходимо выбрать конфигурационный файл **decoder.sof** с помощью кнопки Add File, если он еще не выбран.
7. В поле Program/Configure поставьте галочку.
8. Кнопкой Start запустите процесс загрузки, контролируя в окне Progress ход загрузки (до 100%).

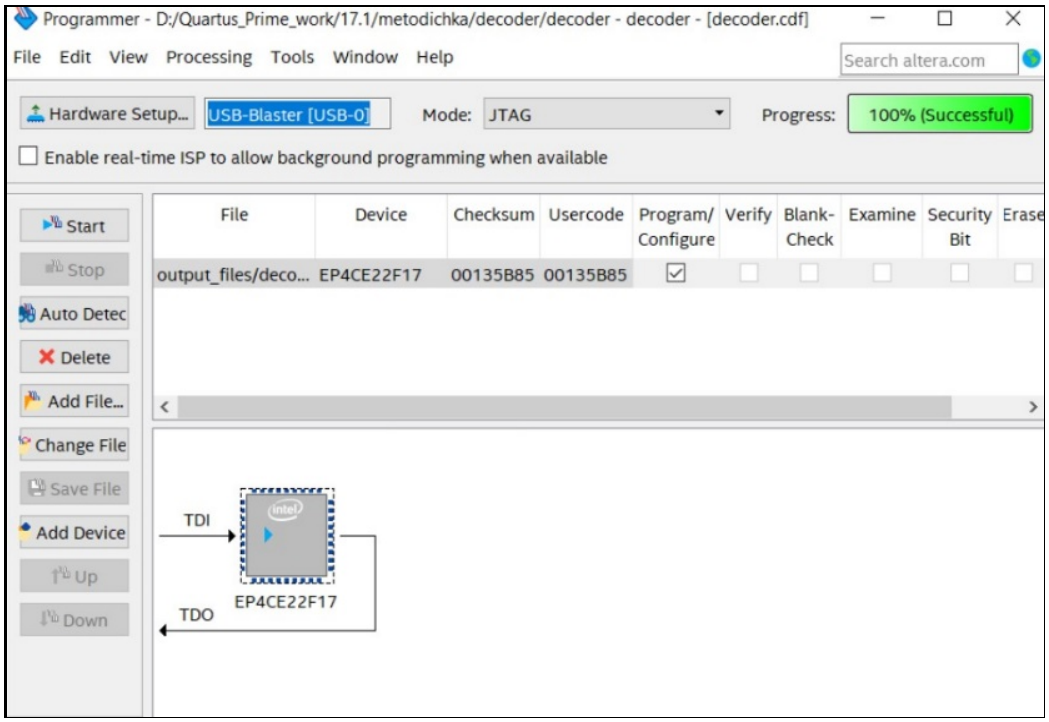



Рис. 37. Окно инструмента Programmer для программирования ПЛИС

После конфигурирования FPGA-стенда проектом **decoder** сразу начинает работать дешифратор. На индикаторе модуля ДМ-1 появляется символ, соответствующий двоичному коду входных переменных.

### Программирование последовательной конфигурационной памяти EPCS64 от персонального компьютера через интерфейс JTAG программатора USB-Blaster

Как было сказано выше, для обеспечения автономности стенда необходимо конфигурационный файл проекта загрузить (запрограммировать) в конфигурационную память. Поскольку плата DEO-Nano не содержит в своем составе отдельного разъема для программирования памяти, то необходимо использовать метод формирования «моста» между USB-Blaster и памятью с помощью FPGA (рис. 38).

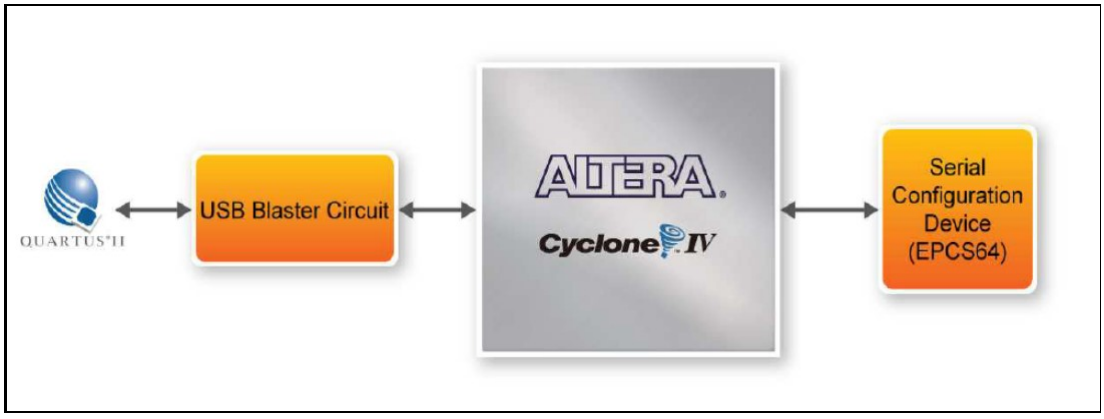



Рис. 38. Метод программирования конфигурационной памяти через FPGA [4a]

Суть метода заключается в следующем: вначале создается загрузчик для последовательной памяти (Serial Flash Loader [26]), который загружается через интерфейс JTAG программатора USB-Blaster в FPGA, затем с помощью этого загрузчика конфигурационный файл уже через Active Serial интерфейс загружается в EPCS64. Для выполнения этих операций необходимо исходный файл **decoder.sof** конвертировать в специальный формат JTAG Indirect Configuration File (.jic) и создать загрузчик (Serial Flash Loader). Эти две операции выполняются инструментом, расположенным по ссылке File/ Convert Programming Files (рис. 39).

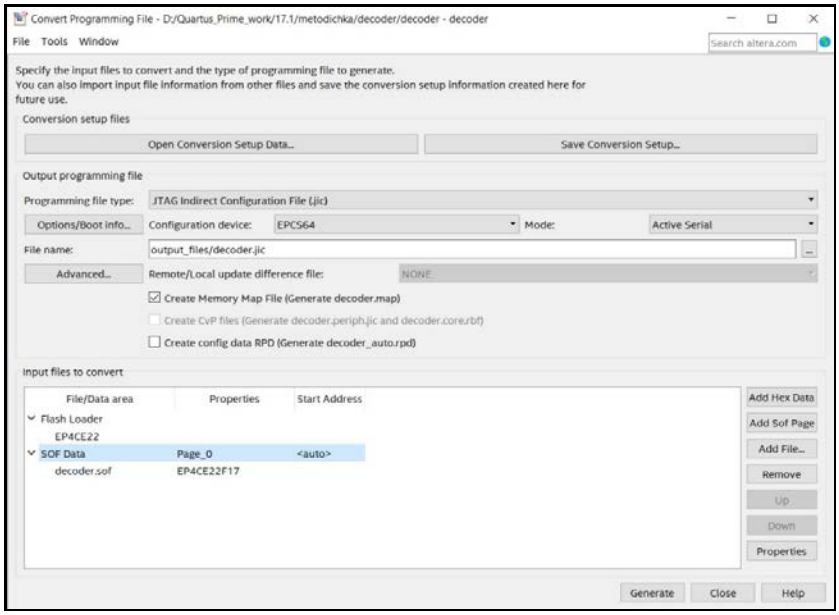



Рис. 39. Создание загрузчика и конвертирование конфигурационного файла

Для правильной конвертации необходимо выполнить некоторые обязательные установки:

1) в окне Programming file type необходимо выбрать JTAG Indirect Configuration File (.jic);

2) в окне Configuration device – EPCS64 (последовательное конфигурационное ПЗУ платы DEO-Nano);

3) в окне Mode – Active Serial;

4) в окне File name – **decoder.jic** (это имя выходного файла после конвертации).

После указанных установок в окне File/Data area должны появиться две строки – Flash Loader и SOF Data (см. рис. 39). Далее необходимо указать для какой FPGA необходимо создать загрузчик. Для этого выделяем строчку Flash Loader и нажимаем кнопку Add Device (выбираем FPGA нашего стенда EP4CE22). Для выбора исходного файла конвертации выделяем строчку с SOF Data и нажимаем кнопку Add File (выбираем файл **decoder.sof**). Кнопкой Generate запускаем процесс создания загрузчика и конвертирования конфигурационного файла.

Процесс программирования последовательного конфигурационного ПЗУ EPCS64 платы DEO-Nano повторяет действия, описанные в разделе «Конфигурирование FPGA от персонального компьютера через интерфейс JTAG программатора USB-Blaster» с использованием утилиты Programmer. Отличие заключается в выборе файла **decoder.jic**, а не **decoder.sof**. После успешного программирования ПЗУ (в окне Progress должно появиться 100% выполнения) необходимо отключить питание платы DEO-Nano, а затем снова включить. После этого содержимое конфигурационного файла переписывается (загружается) из конфигурационной памяти в ПЛИС, а наш дешифратор сразу начинает преобразовывать двоичный код в код семисегментного индикатора.

Для проверки работы цифрового устройства «дешифратор семисегментного индикатора» необходимо задать с помощью DIP-переключателей входные переменные $x_2$, $x_1$, $x_0$ (все восемь комбинаций) и визуально проконтролировать изображения символов на индикаторе от 0 до 7.

## 3.4. Разработка комбинационной схемы на языке высокого уровня

Как упоминалось в предыдущих параграфах, проект разработки цифровых устройств (как и комбинационных схем) можно выполнять на языках описания аппаратуры. В настоящее время при разработке проектов цифровой электроники используются два основных языка описания аппаратуры (Hardware Description Language, HDL) – VHDL и SystemVerilog. Язык Verilog был разработан компанией Gateway Design Automation в 1984 г. как фирменный язык для симуляции логических схем. В 1989 г. Gateway приобрела компания Cadence [27] и Verilog стал открытым стандартом в 1990 г. под управлением сообщества Open Verilog International. Язык стал стандартом IEEE (институт инженеров по электротехнике и электронике (IEEE) – профессиональное сообщество, ответственное за многие компьютерные стандарты) в 1995 г. [28]. В 2005 г. язык был расширен для упорядочивания и лучшей поддержки моделирования и верификации систем. Эти расширения были объединены в единый стандарт, который сейчас называется SystemVerilog (стандарт IEEE 1800-2009). Файлы языка SystemVerilog обычно имеют расширение .sv. По мнению авторов работы [22], язык VHDL более

многословный и громоздкий. В данном учебном пособии при разработке проектов на ПЛИС будем использовать язык HDL – SystemVerilog. Описание структуры и синтаксиса языка SystemVerilog будет самое минимальное, т.е. такое, которое поможет понять работу того или иного проекта. SystemVerilog – сложный язык со множеством операторов. Не все из них могут быть синтезированы в аппаратуре. В данном пособие будет использоваться синтезируемое подмножество языка. Для более подробного изучения данного языка можно использовать описание стандарта SystemVerilog [19] или большой русскоязычный веб-ресурс по ссылке [20]. Рассмотрим основные характеристики синтезируемого подмножества языка SystemVerilog.

Числа в SystemVerilog указываются в двоичной, восьмеричной, десятичной или шестнадцатеричной системе счисления (основания 2, 8, 10 и 16 соответственно). Размер, т.е. количество бит, может быть также указан, свободные разряды заполняются нулями. Подчеркивания в числах игнорируются и могут быть полезными, лишь когда требуется разбить длинное число на более читаемые фрагменты. Формат для объявления констант – *N'Bvalue*, где *N* – размер в битах, *B* – буква, указывающая на основание и *value* – значение. Например, 9'*h*25 определяет 9-битное число со значением $25_{16} = 37_{10} = 000100101_2$. В языке SystemVerilog для обозначения основания системы счисления используются следующие буквы: "*b*" – для основания 2, "*o*" – для основания 8, "*d*" – для основания 10 и "*h*" – для основания 16. Если основание опущено, то по умолчанию оно равно 10. Если не указан размер, то предполагается, что число имеет столько же бит, сколько и выражение, в котором оно используется. Недостающие старшие разряды дополняются нулями автоматически до полного размера. Например, если *w* – 6-битная шина, то assign *w* = 'b11 присваивает *w* значение 000011. В табл. 8 даны примеры записи констант в различных системах счисления.

Таблица 8. **Записи констант с различным основанием**

| Запись | Число бит | Основание | Значение | Двоичный код |
|---|---|---|---|---|
| 3'b101 | 3 | 2 | 5 | 101 |
| 'b11 | ? | 2 | 3 | 11 |
| 8'b11 | 8 | 2 | 3 | 00000011 |
| 8'b1010_1011 | 8 | 2 | 171 | 10101011 |
| 3'd6 | 3 | 10 | 6 | 110 |
| 8'hAB | 8 | 16 | 171 | 10101011 |
| 42 | ? | 10 | 42 | 000101010 |

Основным типом данных, используемым в синтезируемом SystemVerilog, является тип **logic**. Переменная типа **logic** может принимать четыре значения: 1, 0, *x, z*. Тип переменной, обозначенной через *x*, означает неизвестное значение, *z* означает высокоимпедансное состояние (плавающее состояние). Можно создавать вектора типа logic различной длины, например: logic [1:0] data; (двухбитный вектор). Битовые операторы манипулируют однобитовыми сигналами или многоразрядными шинами.

Рассмотрим пример 5-разрядного инвертора на языке SystemVerilog и схему внутренней структуры проекта с помощью RTL (Register Transfer Level) Viewer. Вызов RTL Viewer – Tools/Netlist Viewers/RTL Viewer. RTL Viewer позволяет просматривать схему внутренней структуры списка соединений проекта на уровне регистровых передач.

Вектор *a*[4:0] (рис. 40) представляет собой 5-битную шину. Биты, от старшего к младшему, записываются так: *a*[4], *a*[3], *a*[2], *a*[1] и *a*[0]. Такой порядок битов называется *little-endian*, т.к. младший бит имеет наименьший битовый номер. Можно назвать шину *a*[5:1], и тогда *a*[5] был бы старшим. Или мы могли бы написать *a*[0:4], и тогда порядок битов от старшего к младшему был бы следующим: *a*[0], *a*[1], *a*[2], *a*[3] и *a*[4]. Такой порядок битов называется *big-endian*.

Конструкция приложений на основе данного языка состоит из набора модулей (module), каждый из которых имеет интерфейс ввода-вывода – портов (I/O interface) – и описание его функции, которое может быть структурным или поведенческим. Эти модули сформированы в иерархию и взаимодействуют между собой. Оператор непрерывного присваивания assign перевычисляется каждый раз, когда изменяется какая-либо из переменных в правой части, поэтому этот оператор может описывать только комбинационную логику [27].

В SystemVerilog, как и в других HDL-языках, *z*-состояние используется для указания на плавающее значение. Использование *z*-состояния, в частности, полезно для описания буфера с тремя состояниями, состояние выхода которого является плавающим, когда на вход разрешения подан логический "0". Такой буфер используется в логических элементах, выходы которых объединены шиной, причем эта шина может управляться несколькими буферами с тремя состояниями, только один из которых должен быть активен. Шиной называется средство передачи информации, разделяемое

между несколькими логическими компонентами. В разные моменты времени по шине может передаваться информация от разных источников. На рис. 41 представлена программная реализация 4-разрядного тристабильного буфера. Если этот буфер активирован (*en* = 1), то состояние на выходе будет таким же, как и на входе. Если буфер не активирован, то состояние на выходе назначается плавающим значением (*z*). При описании выходов буфера в разделе портов используется конструкция output tri, что означает, что выход буфера может принимать *z*-состояние.

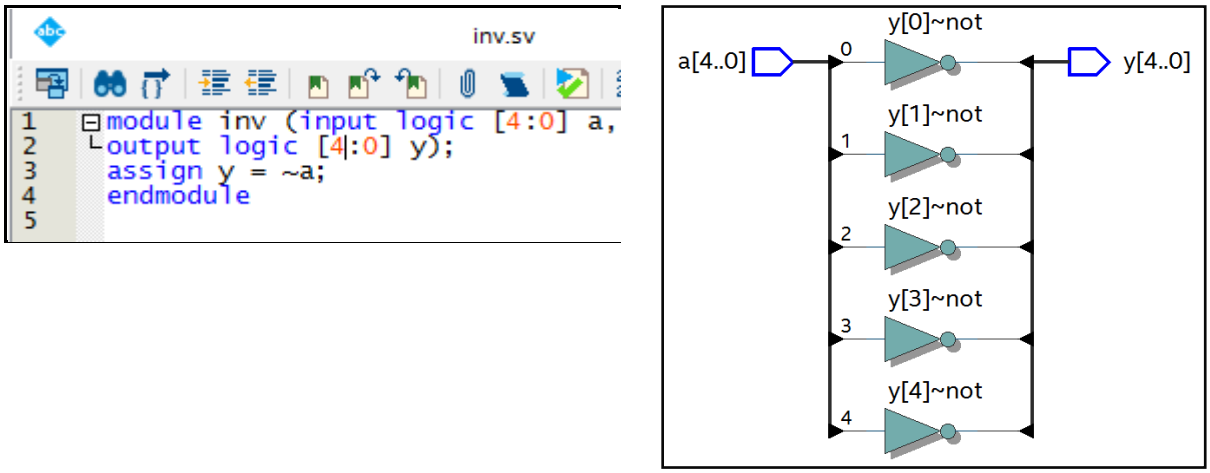



Рис. 40. Программа на языке SystemVerilog 5-битового инвертора (слева) и его изображение в RTL Viewer

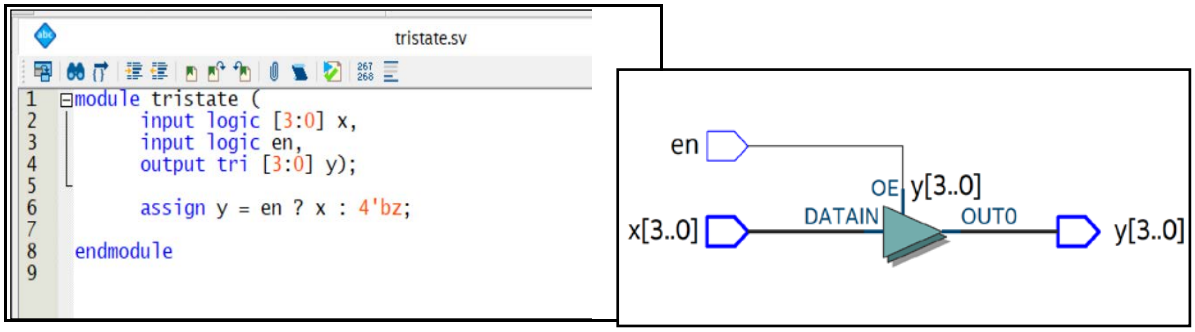



Рис. 41. Программа 4-разрядного тристабильного буфера (слева) и его изображение в RTL Viewer

Условный оператор “?:” выбирает между вторым и третьим выражениями, руководствуясь первым выражением. Первое выражение (в примере – сигнал en) называется условие (condition). Если условие принимает значение 1, то оператор выбирает второе выражение (сигнал *x*). Если условие принимает значение 0, то оператор выбирает третье выражение (4’*bz*, плавающее *z*-состояние всех четырех сигналов). Далее мы покажем, как создать проект на языке высокого уровня SystemVerilog и получить графическое изображение в RTL-виде.

Разработаем дешифратор семисегментного индикатора на SystemVerilog. Разработку проекта дешифратора будем выполнять в проекте **decoder** (см. § 3.3.6), для этого откроем пакет Quartus_Prime и загрузим проект. После того как проект **decoder** откроется, создадим новый файл с именем **sevenseg.sv**. Последовательность шагов по созданию нового файла приведена в § 3.3.6. Выберем в меню File/New создание нового файла. В открывшемся окне New выберем SystemVerilog HDL File. После нажатия на кнопку OK откроется рабочее поле, где будем разрабатывать программу дешифратора на языке SystemVerilog. На рис. 42 представлена программа на языке SystemVerilog дешифратора семисегментного индикатора.

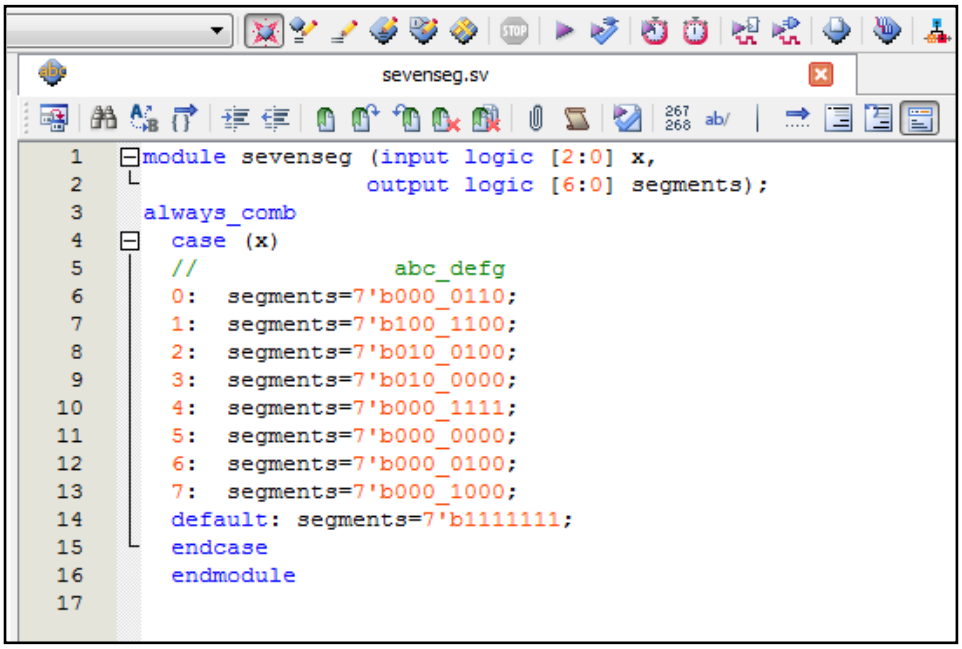



Рис. 42. Программа на языке SystemVerilog дешифратора семисегментного индикатора в пакете Quartus Prime

Операторы always можно использовать для создания триггеров, защелок или комбинационной логики в зависимости от списка чувствительности и оператора. Из-за подобной гибкости языка при синтезе аппаратных блоков можно непреднамеренно получить нежелательную конфигурацию. Во избежание таких ошибок в SystemVerilog введены операторы always_ff, always_latch и always_comb. Оператор always_ff ведет себя так же, как always, но используется только тогда, когда подразумевается синтез триггеров, и позволяет инструментальной среде в противном случае выдавать предупреждение.

Оператор always_comb исполняет выражения внутри оператора always каждый раз, когда изменяется любой из сигналов в правой части <= или = (в этом примере) оператора always. Равенство = в операторе always называется блокирующим присваиванием, в отличие от неблокирующего присваивания <=. В SystemVerilog хорошей практикой является использование блокирующих присваиваний для комбинационной логики и неблокирующих – для последовательностной. Оператор case проверяет значение *x*; если *x* равно 0, выполнится действие после двоеточия, т.е. установка segments в 0000110. Аналогично проверяются другие значения *x* вплоть до 9 (обратите внимание, что по умолчанию система счисления десятичная). Условие default (по умолчанию) – удобный способ определить выход для всех случаев, не упомянутых явно, тем самым гарантируя в результате комбинационную логику. В SystemVerilog операторы case обязаны находиться внутри операторов always [19].

Если проверить правильность работы схемы методом функциональной и временной симуляции проекта, то мы увидим такой же результат, как и в § 3.3.6.

Далее, при описании последовательностных схем и примеров на языке SystemVerilog, будем знакомиться с синтаксисом и конструкциями языка.

## 3.5. Разработка последовательностных устройств в пакете Quartus Prime

В предыдущих параграфах мы познакомились с классом цифровых устройств, названных комбинационными схемами. Второй класс цифровых устройств – это логические схемы, выходные сигналы которых определяются не только текущими значениями входных сигналов, но и зависят от последовательности значений входных сигналов в прошлом. В большинстве последовательностных схем изменение выходов происходит в моменты времени, задаваемые внешним тактовым сигналом от независимого источника. Различают два типа простейших последовательностных устройств – защелки и триггеры. В цифровой электронике принято триггером (flip-flop) называть последовательностную схему, в которой значения выходных сигналов изменяются только в моменты времени, задаваемые тактовым сигналом. Название «защелка» (latch) используется для последовательностной схемы, выходы которой чувствительны к изменению входных сигналов (но зависят не только от входных сигналов, но и от текущего состояния) непрерывно в течение всего времени [10]. Рассмотрим краткие сведения из теории об особенностях работы этих простейших цифровых устройств.

### 3.5.1. SR-защелка

Рассмотрим простейшую, так называемую SR-защелку. На рис. 43 показана SR-защелка (set-reset latch) на вентилях ИЛИ-НЕ. У этой схемы два входа S и R и два выхода Q и QN. Сигнал QN представляет собой инверсию сигнала Q. Создадим проект защелки в схемном редакторе и на языке SystemVerilog, причем обе защелки будут находиться в одном проекте **sr_latch**. Сначала в схемном редакторе нарисуем входы S и R и выходы Q и QN. Затем на двух элементах ИЛИ-НЕ (nor2) создадим защелку. Выполним компиляцию и убедимся, что ошибок нет.

Далее создадим новый файл. Выберем в меню File/New создание нового файла. В открывшемся окне New выберем SystemVerilog HDL File. После нажатия на кнопку OK откроется рабочее поле, где будем разрабатывать программу SR-защелки на языке SystemVerilog. Сохраним файл под именем **sr_latchsv.sv**. Выходы этой защелки обозначим через Q1 и QN1. Создадим два компонента nor2 и связи как на защелке в схемном редакторе. В выражении nor (Q1, R, QN1) в скобках слева направо: Q1 – выход элемента nor, R и QN1 – его входы. По аналогии с первым формируется второй элемент nor2.

На рис. 43 показаны обе защелки в графическом и текстовом виде.

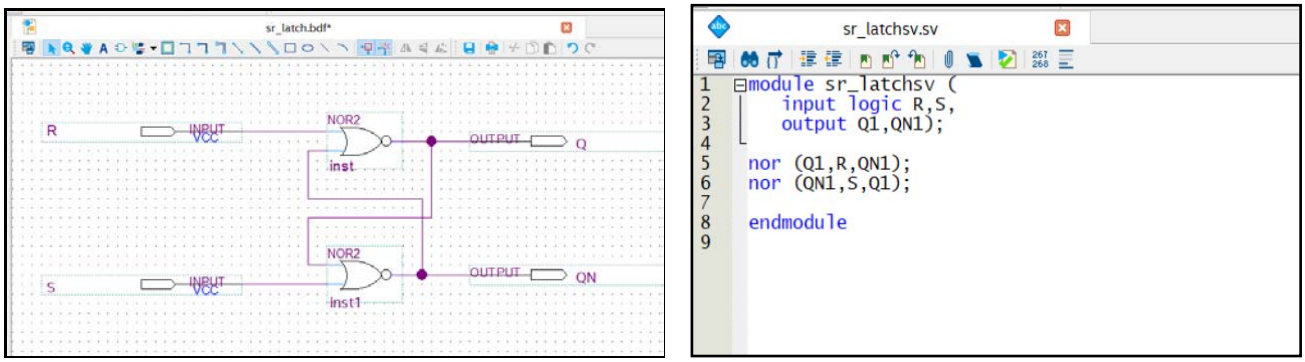



Рис. 43. SR-защелка: схема на элементах nor2 и на языке SystemVerilog

Из защелки sr_latchsv.sv создадим компонент, для этого выберем File/Create/Update/Create Symbol Files for Current File. После успешной операции создания в библиотеке логических компонентов (в разделе Project) появится компонент с названием sr_latchsv. Поместим этот компонент на рабочий стол проекта (рис. 44). Запустим процесс компиляции и при ее успешном завершении создадим файл для симуляции. Полезно с помощью RTL Viewer просмотреть структуру созданного проекта.

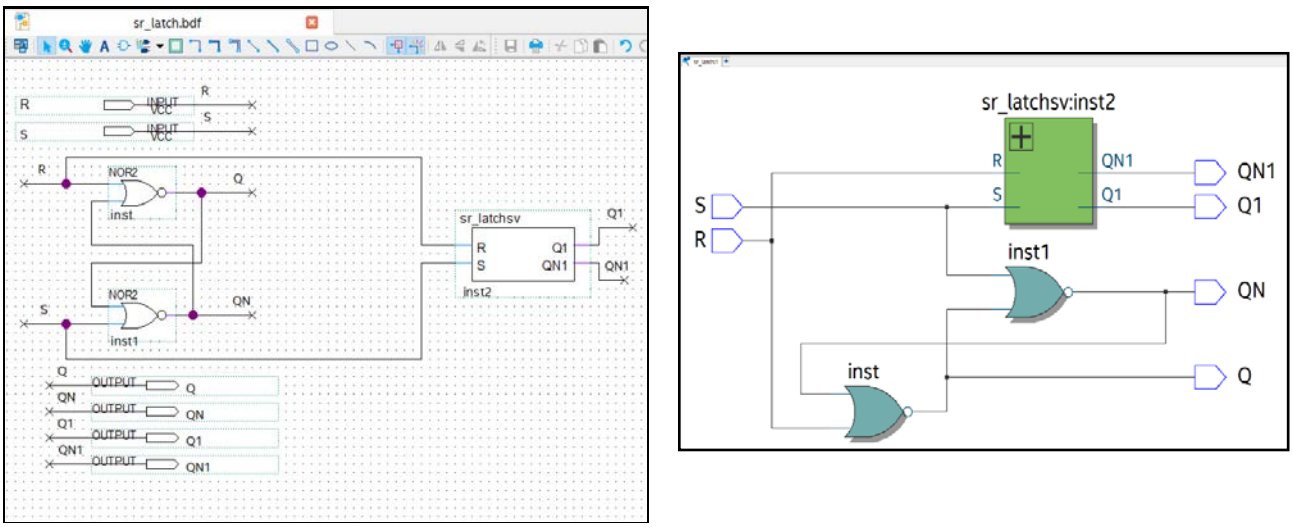



Рис. 44. Две SR-защелки и их представление в RTL Viewer

Если оба входных сигнала S и R равны 0, то схема ведет себя аналогично элементу с двумя устойчивыми состояниями, т.е. на выходах сохраняется последнее состояние (last Q и last QN). Меняя состояния S и R, можно заставить схему переходить в требуемое состояние. Сигнал S (активное состояние “1”) устанавливает (set) состояние, при котором выходной сигнал Q равен “1”. Сигнал R сбрасывает (reset) или очищает схему и выход Q становится равным “0”. На рис. 45 представлены результаты временного симулирования работы SR-защелки.

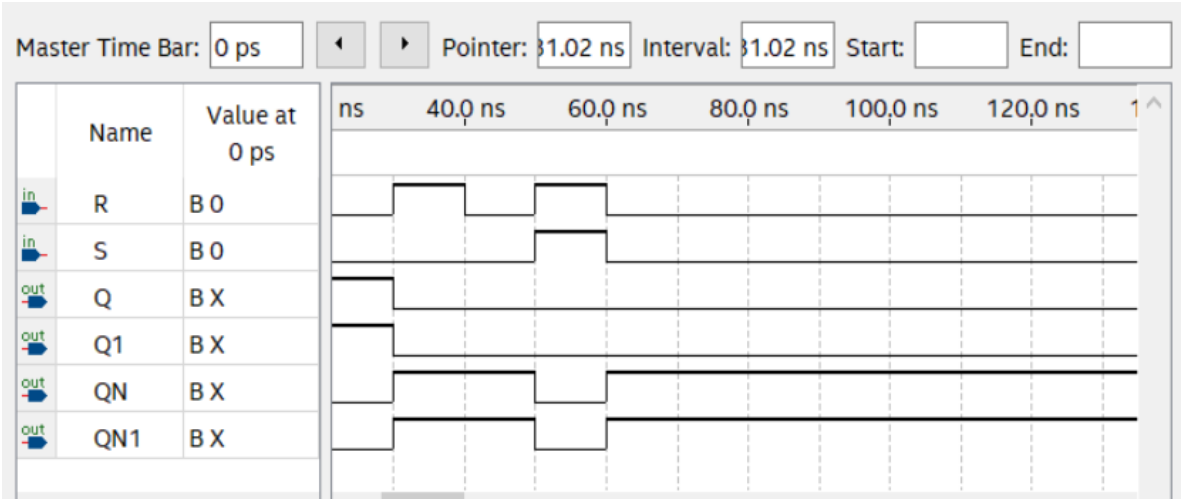



Рис. 45. Временная диаграмма работы SR-защелки

SR-защелки чувствительны к входным сигналам S и R в течение всего времени. Если мы хотим, чтобы выходные сигналы изменялись только в строго определенное время (во время действия разрешающего сигнала, например C), то схему защелки необходимо видоизменить. Реализуем защелку видоизмененной схемы не на элементах ИЛИ-НЕ (nor2), а на элементах И-НЕ (nand2). На рис. 46 представлена схема, имеющая разрешающий сигнал C, с элементами стробирования и SR-защелка.

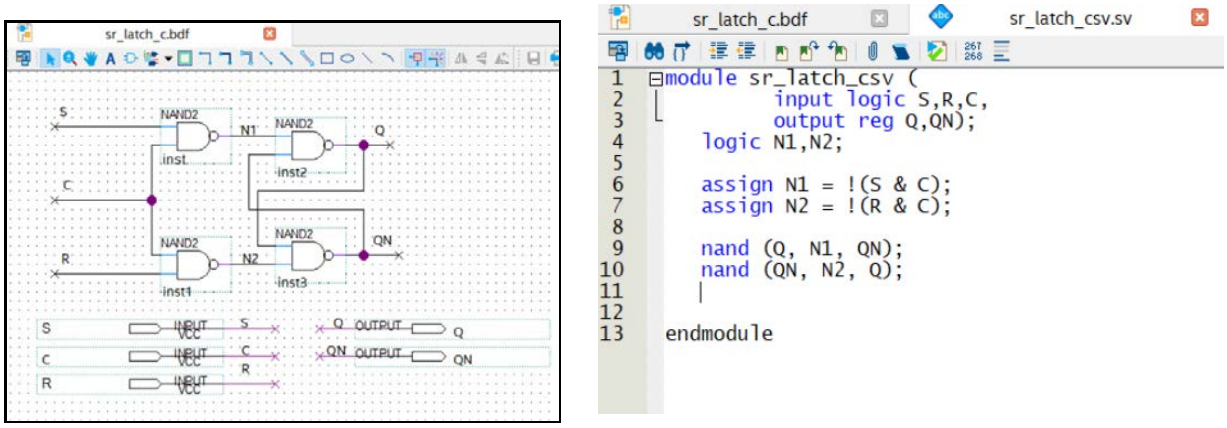



Рис. 46. SR-защелка с входом разрешения C: схема на вентилях И-НЕ (слева), на языке SystemVerilog

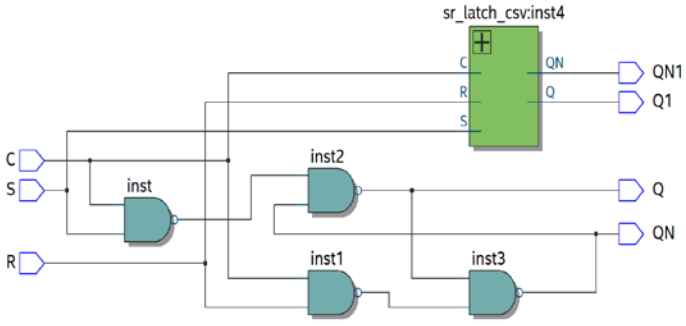



Рис. 47. Две SR-защелки с разрешением и их представление в RTL Viewer

При C = 1 данная схема ведет себя как SR-защелка, а при C = 0 она удерживается в прежнем состоянии. Из защелки sr_latch_csv.sv создадим компонент, для этого выберем File/Create/Update/ Create Symbol Files for Current File. После успешной операции создания в библиотеке логических компонентов, в разделе Project, появится компонент с названием sr_latch_csv. Поместим этот компонент на рабочий стол проекта **sr_latch_c.bdf**. На рис. 47 представлена схема двух SR-защелок с разрешением и RTL-структура этой схемы. На рис. 48 приведены временные диаграммы работы схемы при различных входных сигналах. Если оба сигнала (S и R) равны 1 в момент, когда сигнал C переходит из 1 в 0 (отрицательный фронт сигнала), то схема ведет себя подобно SR-защелке при одновременном переходе сигналов S и R на неактивный уровень. В этом случае следующее состояние непредсказуемо, и выходная цепь может стать метастабильной (подробнее о метастабильности триггера рассказано в § 4.4.1).

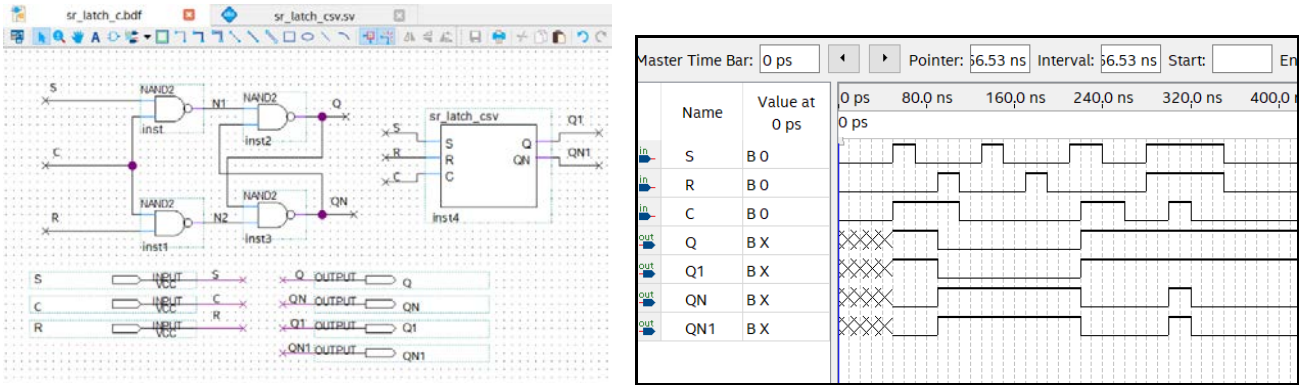



Рис. 48. Временная диаграмма работы SR-защелок с разрешением

### 3.5.2. D-защелка

В цифровой схемотехнике очень часто бывают нужны защелки, чтобы просто запомнить биты информации, когда каждый бит поступает по отдельной сигнальной линии и его необходимо сохранить. В этом случае удобно воспользоваться D-защелкой. Создадим проект защелки в схемном редакторе и на языке SystemVerilog, причем обе защелки будут находиться в одном проекте d_latch.bdf. Сначала в схемном редакторе нарисуем входы D и C и выходы Q и QN. Затем на четырех элементах И-НЕ (nand2) создадим защелку. Схема D-защелки отличается от схемы SR-защелки только тем, что сигнал R заменен на инверсию сигнала D. На рис. 49 показана схема D-защелки, состоящая из SR-защелки с входом разрешения и дополнительного инвертора (not), который формирует S- и R-сигналы из единственного входа D.

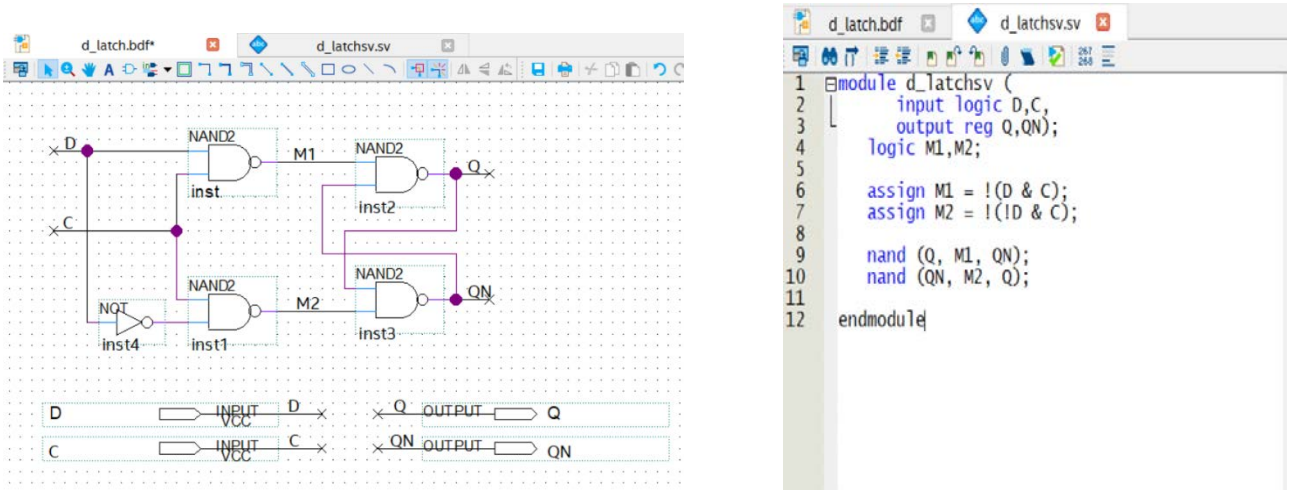



Рис. 49. D-защелка с входом разрешения C: схема на вентилях И-НЕ (слева), на языке SystemVerilog

По аналогии с проектом в предыдущем параграфе (SR-защелка) выполним описание D-защелки на языке SystemVerilog и создадим из файла d_latchsv.sv компонент. Поместим этот компонент на рабочее поле проекта d_latch.bdf. Выполним компиляцию и убедимся, что ошибок нет.

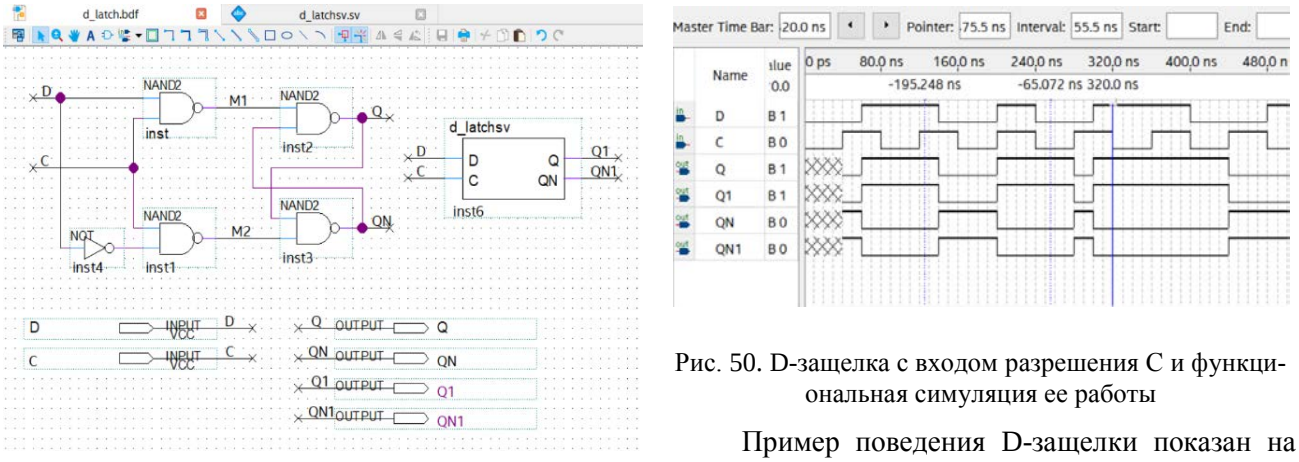



Рис. 50. D-защелка с входом разрешения C и функциональная симуляция ее работы

Пример поведения D-защелки показан на

рис. 50. Когда сигнал C активный, выходной сигнал Q (Q1) повторяет значение входного сигнала D. Когда сигнал C снимается, защелка запирается; выходной сигнал Q (Q1) сохраняет свое последнее значение и больше не реагирует на изменение сигнала D. В D-защелке нет проблемы, имеющейся в SR-защелке (S=R=1), но остаются затруднения, связанные с метастабильностью. В частности, в окрестности отрицательного фронта сигнала C (вертикальный маркер на рис. 50) существует интервал времени, в котором входной сигнал D не должен изменяться. Этот интервал начинается за время $t_{setup}$ до отрицательного фронта сигнала C (это время называется временем *установления*) и заканчивается спустя время $t_{hold}$ после отрицательного фронта C (это время называется временем *удержания*). Если входной сигнал *D* изменяется внутри этого интервала ($t_{setup}$ + $t_{hold}$), то значение сигнала на выходе защелки непредсказуемо.

### 3.5.3. D-триггер, переключающийся по фронту

Рассмотрим схему, состоящую из двух D-защелок и двух инверторов. Создадим проект **d_flip** с использованием ранее созданных D-защелок. Запустим мастер создания нового проекта New Project Wizard. На вкладке мастера, предлагающего добавить в проект файлы (Add files – рис. 18), необходимо выбрать файл d_latchsv.sv, который на языке SystemVerilog описывает D-защелку. После создания файла в схемном редакторе d_flip.bdf и добавления его в проект d_flip создадим из файла d_latchsv.sv компонент d_latchsv, как описано в § 3.5.1. Созданная схема называется D-триггером, переключающимся по положительному фронту тактового сигнала. На рис. 51 показа схема, в которой опрос ее входа D и изменение ее выходных сигналов Q и QN происходит только в моменты времени, задаваемые положительным фронтом сигнала CLK. В этой схеме первая защелка называется ведущей (master): при значении CLK, равном 0, она открыта и ее выходной сигнал повторяет входной. Когда сигнал CLK переходит в состояние 1 (положительный фронт), ведущая защелка запирается и ее выходной сигнал переносится во вторую защелку, называемую ведомой (slave), и защелкивается там. Ведомая защелка открыта в течение всего времени, пока значение CLK остается равным 1, но изменение сигнала на ее выходе возможно только в самом начале этого интервала, поскольку ведущая защелка заперта и сигнал на ее выходе остается неизменным. Для изучения работы D-триггера в схему добавлен промежуточный сигнал QM, с выхода ведущей защелки.

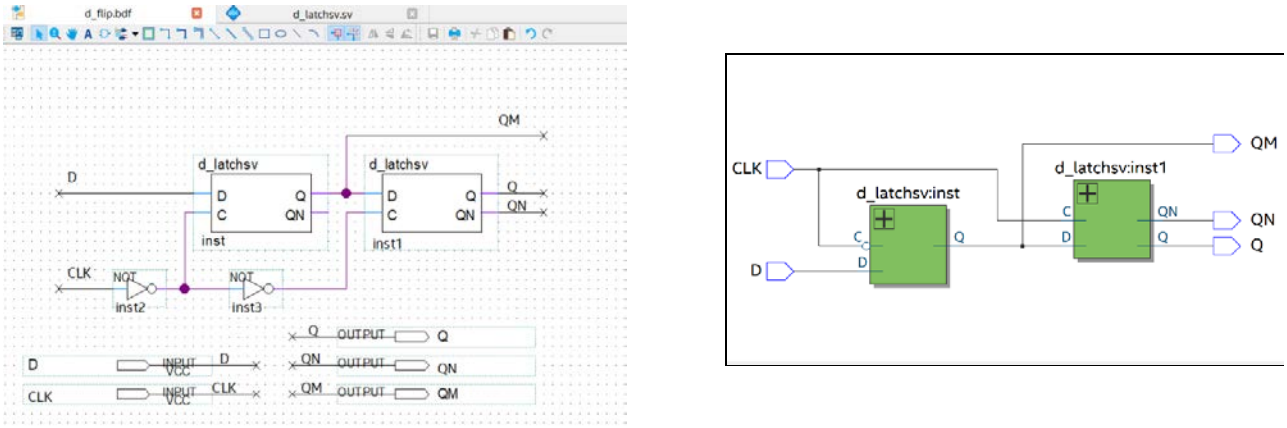



Рис. 51. D-триггер: схема на D-защелках (слева) и представление в RTL Viewer

Рассмотрим работу приведенной схемы после выполнения функциональной и временной симуляции проекта. На рис. 52 показаны временные диаграммы работы D-триггера.

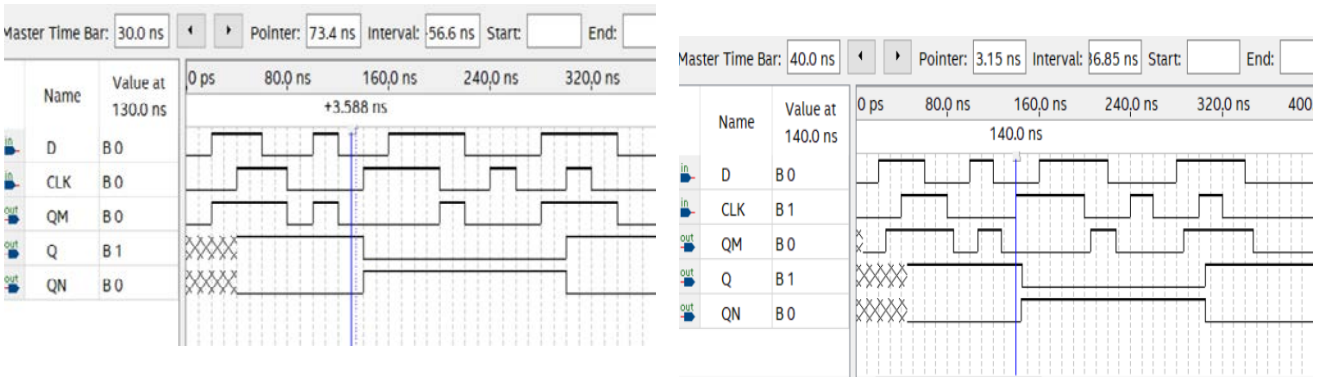



Рис. 52. Симуляция работы D-триггера: функциональная (слева) и временная

Необходимо обратить внимание, что сигнал QM изменяется только при CLK, равным 0. Когда CLK становится равным 1, текущее значение QM переносится на выход Q, тогда как изменение сигнала QM невозможно до тех пор, пока CLK снова не станет равным 0.

Так же, как и в случае D-защелки, у D-триггера есть интервал времени, состоящий из времени установления и времени удержания, в течение которого сигнал на входе D не должен меняться. Этот сигнал находится в окрестности *положительного фронта* сигнала CLK.

### 3.5.4. Регистр на основе D-триггеров, переключающихся по фронту

*Регистром* называют совокупность из двух или большего числа триггеров с общим входом тактового сигнала. Регистры часто применяют для промежуточного хранения набора связанных между собой (функционально однотипных) битов, например байт данных в компьютере. Но можно в регистре сохранять и не связанные биты, единственное ограничение – все биты будут запоминаться в регистре в один и тот же момент времени. Регистры можно классифицировать по способу приема и выдачи данных. По этому признаку различают параллельные, последовательные (сдвиговые) и параллельно-последовательные. В параллельных регистрах прием и выдача слов производится одновременно. В последовательных регистрах биты информации принимаются и выдаются разряд за разрядом, т.е. происходит сдвиг данных по разрядам от входа к выходу или обратно (реверсивный сдвиговый регистр). Параллельно-последовательные регистры имеют входы-выходы одновременно параллельного и последовательного типов. В таких регистрах можно принимать информацию в параллельном виде, а передавать в последовательном.

Как правило, обрабатываемая в цифровых системах информация [4] представляется словами, состоящими из 8, 16 или 32 битов. Поэтому разрядность регистров кратна восьми, но бывают и исключения, когда разрядность равна полубайту. В сложных проектах используется много регистров, выходы которых объединены в так называемые шины данных. Для считывания информации из определенного регистра используются специальные буферные элементы с тремя состояниями, т.е. с *z*-состоянием. Такие элементы рассмотрены в предыдущих параграфах. В это (третье) состояние элемент переводится с помощью специального управляющего сигнала. Таким образом, при считывании информации из регистров по шине данных только один регистр подключен, а выходы остальных находятся в третьем состоянии.

Разработаем 4-разрядный регистр, состоящий из переключающихся по фронту D-триггеров. Для этого из библиотеки примитивов на рабочее поле скопируем элементы: dff – D-триггер с входами предустановки, tri – буферный элемент с тремя состояниями и not – инвертор. Создадим схему регистра. Входная информация подается по линиям данным d[3..0]. Как видно из рис. 53, регистр имеет четыре D-триггера, которые одновременно записывают входную информацию по нарастающему фронту сигнала clk, которая с небольшой задержкой (задержка определяется срабатыванием D-триггера, как определено в предыдущем параграфе) появляется на выходах триггеров.

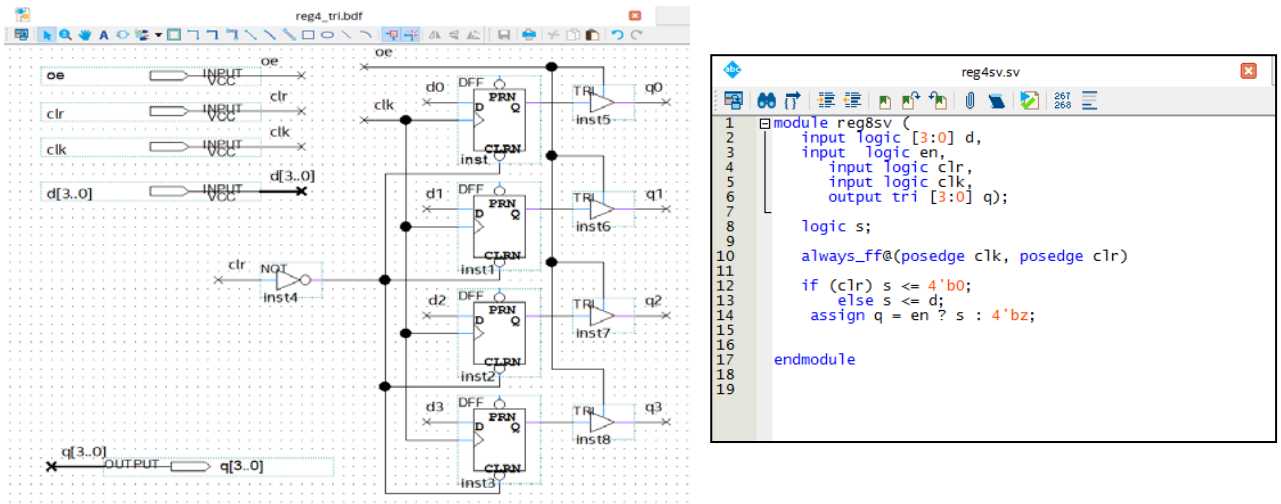



Рис. 53. 4-разрядный регистр на переключающихся по фронту D-триггерах и выходами с тремя состояниями: реализация в схемном редакторе (слева), на языке SystemVerilog

Сигнал с выхода каждого триггера поступает на буфер с тремя состояниями и далее на соответствующий выходной элемент схемы регистра (output). Установка всех триггеров в исходное состояние выполняется с помощью сигнала clr. Реализованный на языке SystemVerilog регистр использует always_ff со списком чувствительности из сигналов clk и clr.

Рассмотрим временные диаграммы работы регистра (рис. 54). Для этого зададим во временной симуляции разные значения входных сигналов d[3..0] в шестнадцатеричном коде (равными 0×4, 0×7, 0×A). Например, по переднему фронту сигнала clk (маркер на рис. 54) код 0×4 фиксируется на выходах q[3..0] D-триггеров регистра. Информация достоверна на выходах регистра q[3..0] только при активном управляющем сигнале oe. Все остальное время выходы регистра находятся в третьем состоянии (третье состояние обозначено Z).

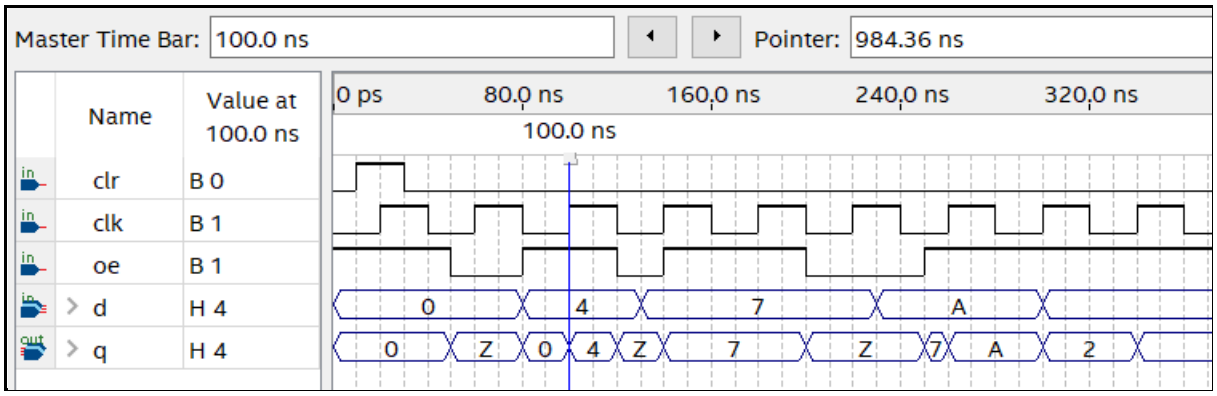



Рис. 54. Временная диаграмма работы параллельного регистра

### 3.5.5. Счетчики с последовательным переносом

Счетчиком называется тактируемая последовательностная схема, выполненная на *n-триггерах*, диаграмма состояний которых представляет собой кольцо, т.е. за последним состоянием следует первое. Счетчики позволяют вести подсчет электрических импульсов, количество которых (поступивших на тактовый вход счетчика) представляется обычно в параллельном коде.

Счетчики подразделяются на *асинхронные* и *синхронные*. У синхронных счетчиков все разрядные триггеры синхронизируются параллельно одними и теми же синхроимпульсами, поступающими из источника этих импульсов. Асинхронные счетчики имеют последовательную синхронизацию, т.е. каждый последующий разрядный триггер синхронизируется выходными импульсами триггера предыдущего разряда.

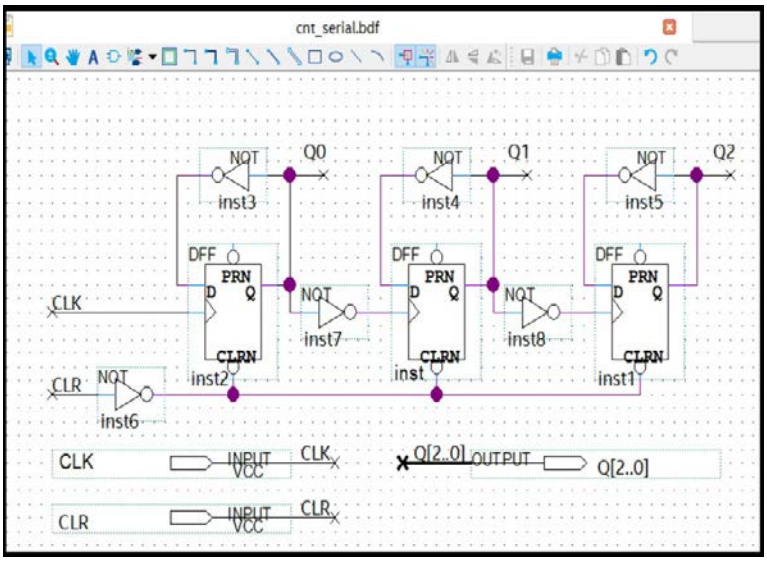



Рис. 55. 3-разрядный счетчик на D-триггерах

Асинхронные счетчики иногда называют счетчиками с последовательным переносом или последовательными, а синхронные счетчики – параллельными. Разработаем 3-разрядный счетчик с последовательным переносом на D-триггерах и общим сигналом сброса CLR (рис. 55). У этого счетчика каждый разрядный D-триггер включен по схеме счетного триггера, т.е. инверсный выход соединен с D-входом. Прямые выходы триггеров служат выходами счетчика соответствующего разряда (Q0, Q1, Q2), а инверсные выходы каждого триггера являются тактовыми. Прямые выходы триггеров служат выходами счетчика соответствующего разряда (Q0, Q1, Q2), а инверсные выходы каждого триггера являются тактовыми сигналами для следующего триггера. Поскольку счетчик суммирующий, то каждый положительный фронт сигнала CLK увеличивает значение счетчика на единицу. Из временной диаграммы (функциональная симуляция, рис. 56) следует, что содержимое того или иного разряда счетчика меняется на противоположное тогда и только тогда, когда значение инверсного выхода триггера предыдущего разряда меняется с 0 на 1. Это соответствует двоичному счету в прямом направлении, когда бит, хранящийся в данном разряде, изменяется с 1 на 0, возникает перенос информации в следующий разряд. Такой счетчик называется счетчиком с последовательным переносом, поскольку информация о переносе поочередно передается от младшего разряда к старшему.

На рис. 56 видно, что при активном сигнале сброса CLR все выходы счетчика принимают нулевое состояние. Временная симуляция счетчика показывает, что при каждом положительном фронте сигнала CLK происходит увеличение значения счетчика от 0 до 7 и увеличение задержки между нарастающим фронтом CLK и нарастающим фронтом разрядных выходов Q[2..0].

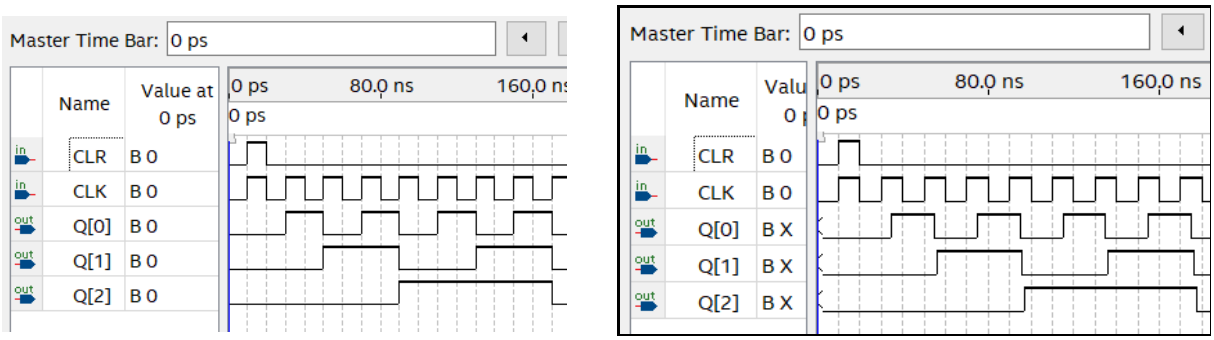



Рис. 56. Временные диаграммы работы счетчика: функциональная симуляция (слева) и временная

Рассмотренный тип счетчиков может быть использован в цифровых устройствах «умеренного» быстродействия, когда частота следования синхроимпульсов не превышает критического значения, при котором время задержки установки триггеров последних (старших) разрядов счетчика становится соизмеримым с длительностью периода входных тактовых импульсов. В связи с этим, асинхронные счетчики строятся на относительно небольшое количество разрядов, т.к. при большем количестве разрядов выходные сигналы триггеров старших разрядов появляются позднее, чем управляющие фронты синхроимпульсов (поступающих на вход первого триггера).

### 3.5.6. Синхронные счетчики

В синхронном счетчике к тактовым входам всех триггеров подводится один и тот же тактовый сигнал CLK, так что изменение значений сигналов на выходах всех триггеров происходит в один и тот же момент времени. Синхронные счетчики могут быть с последовательным переносом (синхронный последовательный счетчик) и параллельным переносом (синхронный параллельный счетчик). Синхронные последовательные счетчики быстрее асинхронных последовательных счетчиков, но если период тактового сигнала слишком мал, то изменение в младшем разряде (перенос информации) может не успеть дойти за это время до старшего разряда.

Разработаем схему 4-разрядного синхронного параллельного счетчика. Для этого из библиотеки примитивов пакета Quartus Prime на рабочее поле скопируем элементы dffe – D-триггер с входами предустановки и разрешения, and2 – элемент 2И, and3 – элемент 3И and4 – элемент 4И, not – инвертор. Счетчик будет иметь сигнал сброса CLR, тактовый сигнал CLK и сигнал разрешения счета CNT_EN. На рис. 57 представлен синхронный параллельный счетчик на D-триггерах, включенных в счетном режиме. Каждый триггер имеет вход разрешения работы и входы предустановки.

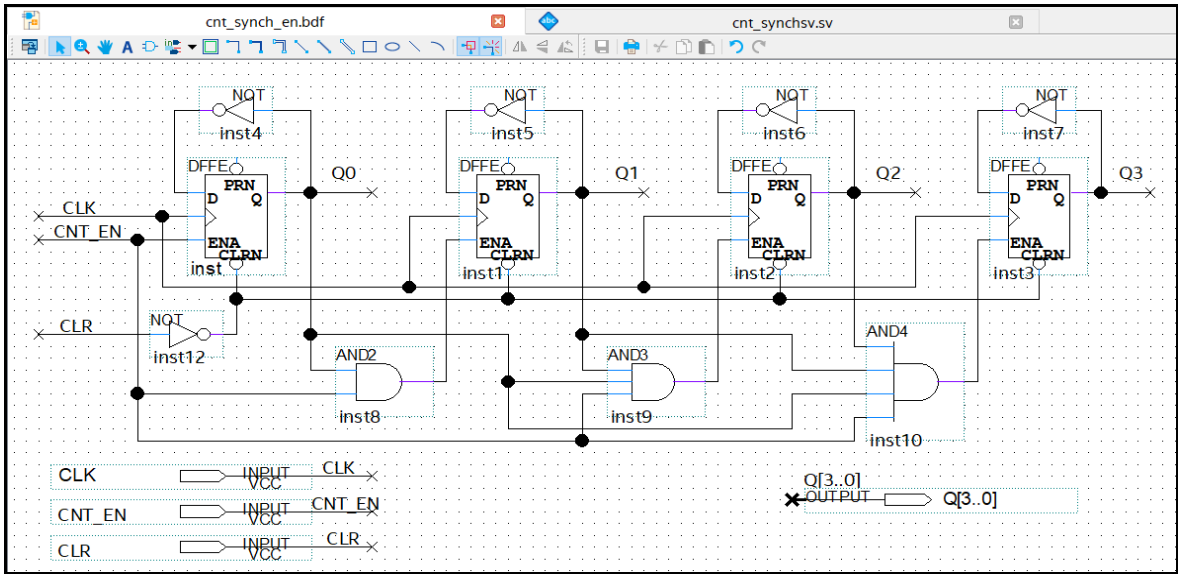



Рис. 57. 4-разрядный синхронный параллельный счетчик на D-триггерах

Если сигнал разрешения счета CNT_EN активный (лог.1), то счетчик меняет свое состояние (рис. 58), в противном случае выходы счетчика остаются без изменений.

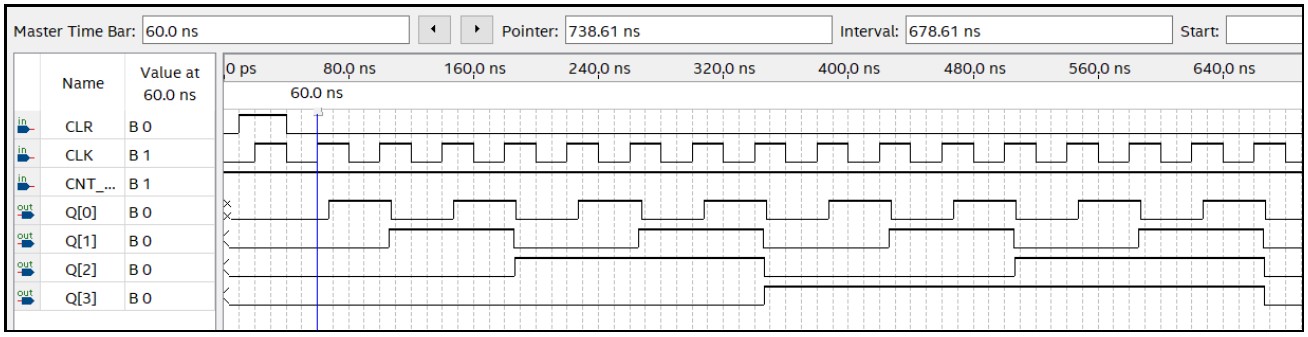



Рис. 58. Временная диаграмма работы синхронного счетчика

Поскольку счетчик имеет одну общую линию тактирования (синхронизации), состояние триггеров меняется синхронно, т.е. те триггеры, которые по переднему фронту тактового сигнала должны изменить свое состояние, делают это одновременно, что существенно повышает быстродействие синхронных счетчиков.

На рис. 59 представлен 4-разрядный синхронный счетчик с асинхронным сбросом, реализованный на языке SystemVerilog, и временная диаграмма его работы (временное симулирование). Как уже было сказано в § 3.4, при формировании последовательностных устройств с помощью оператора always_ff используется оператор «<=», что означает неблокирующее присваивание, т.е. запоминание вычисленных значений в элементах памяти (триггеры, регистры и т.д.) происходит одновременно при появлении положительного фронта сигнала CLK (posedge CLK). Противоположно неблокирующему блокирующее присваивание («=») заставляет вычислять выражения, описанные в always-блоке, последовательно. Выражение if (CLR) заставляет асинхронно устанавливать (присваивать) выходы разрядных триггеров в «0». В противном случае, если (else) переменная CNT_EN в значении логической «1», то значение счетчика увеличивается (QL +1'd1).

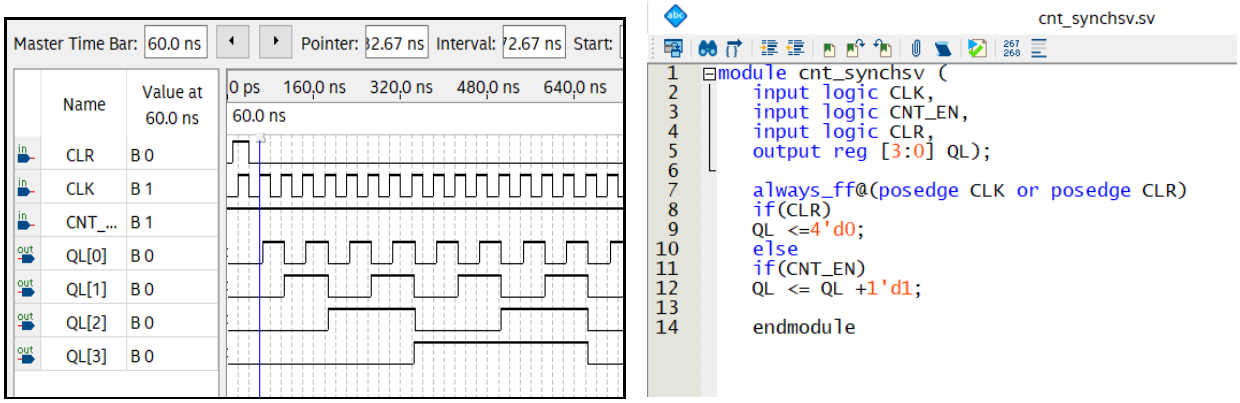



Рис. 59. Синхронный счетчик на языке SystemVerilog (справа) и временная диаграмма его работы

### 3.5.7. Счетчики с произвольным коэффициентом счета

Принцип построения счетчиков с произвольным коэффициентом счета (с коэффициентом пересчета, отличающимся от двоичного) состоит в исключении нескольких состояний обычного двоичного счетчика, являющихся избыточными. При этом избыточные состояния исключаются с помощью обратных связей внутри счетчика.

Число избыточных состояний для любого счетчика определяется из следующего выражения:

$$M = 2^m - К_{сч},$$

где $M$ – число запрещенных состояний; $К_{сч}$ – требуемый коэффициент счета; $2^m$ – число устойчивых состояний двоичного счетчика. Задача синтеза счетчика с произвольным коэффициентом счета заключается в определении необходимых обратных связей и минимизации их числа. Требуемое количество триггеров определяется из выражения

$$n = [\log_2 К_{сч}],$$

где [$\log_2 К_{сч}$] – двоичный логарифм заданного коэффициента пересчета $К_{сч}$, округленный до ближайшего целого числа. В каждом отдельном случае приходится применять какие-то конкретные методы получения требуемого коэффициента пересчета. Существует несколько методов получения счетчиков с заданным коэффициентом пересчета $К_{сч}$. Один их этих методов заключается в немедленном сбросе в “0” счетчика, установившегося в комбинацию, соответствующему числу $К_{сч}$.

Разработаем проект цифровой схемы, включающий делитель частоты с переменным коэффициентом деления на синхронных счетчиках. Пусть требуемый коэффициент пересчета $К_{сч}$ = 19, а название проекта **counter_synch**.

Выполним следующие действия:

- для изучения работы синхронных счетчиков в пакете Quartus Prime создать проект, включающий синхронный счетчик 74193 из библиотеки пакета Quartus Prime (megafunctions → others → → maxplus2 → 74193);
- в схемном редакторе разработать схему и выполнить компиляцию проекта;
- в редакторе **University Program VWF** задать входные и выходные сигналы счетчика и выполнить функциональное и временное моделирование схемы;
- разработать проект цифровой схемы, включающий делитель частоты с переменным коэффициентом деления на синхронных счетчиках 74193;
- выполнить компиляцию проекта;
- с помощью входных переключателей стенда (см. «Руководство по эксплуатации учебного стенда CLE-148») задать коды делителя, соответствующие коэффициенту деления $К_{сч}$ и смотреть на осциллографе входные и выходные частоты делителя.

Выполним последовательно указанные действия.

1. Создадим проект **counter_synch.bdf** (рис. 60) и выполним компиляцию.

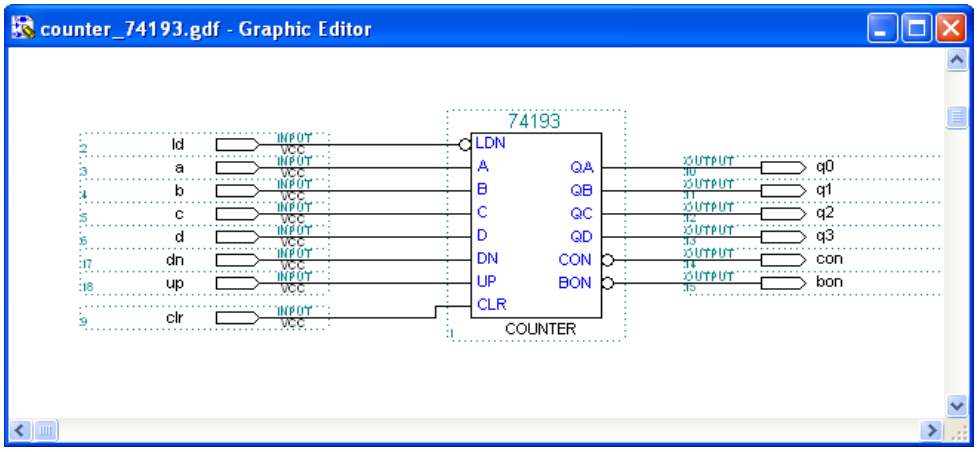



Рис. 60. Синхронный счетчик со входом параллельной загрузки

2. В редакторе University Program VWF задаем входные и выходные сигналы (рис. 61) счетчика и выполняем временное моделирование. Счетчик 74193 (аналог К155ИЕ7) является 4-разрядным синхронным счетчиком с параллельной загрузкой. Параллельная загрузка (сигналом ld) предполагает предварительную установку выходов счетчика в состояние, кодовая комбинация которого задана на входах счетчика.

3. Выполним расчет делителя по заданному коэффициенту деления $К_{сч}$.

Разработаем схему делителя частоты на базе счетчика 74193. Прежде всего, необходимо вычислить двоичный код для загрузки. Пусть требуемый коэффициент пересчета, как указано выше, $К_{сч}$ = 19.

• Требуемое количество разрядных триггеров определяется из выражения

$$n = [\log_2 К_{сч}] = [\log_2 19] = [\sim 5] = 5.$$

Поскольку счетчик 74193 имеет число разрядов, кратное 4, то нам необходимо использовать два элемента 74193. С учетом этого примем $n$ = 8.

• Определим число избыточных состояний $M$:

$$M = 2^n - К_{сч} = 2^8 - 19 = 256 - 19 = 237.$$

• Переведем десятичное число «237» в двоичный код восьмиразрядного числа ($n=8$):

$$237_{10} = 10001001_2.$$

Таким образом, получили следующие исходные данные для разработки делителя:

- Число разрядов счетчика делителя $n=8$;
- Параллельный код для предзагрузки в счетчики – 11101101 (0xED – шестнадцатеричный код числа).

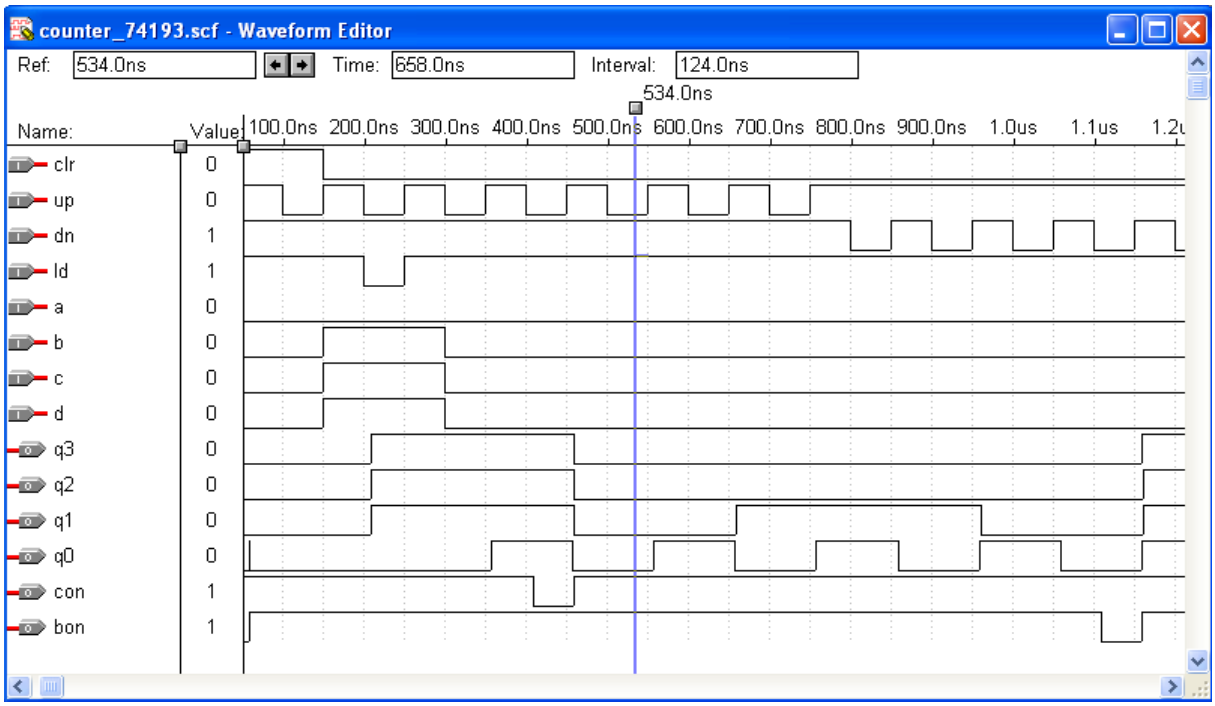



Рис. 61. Временная диаграмма работы синхронного счетчика 74193

На рис. 62 изображена схема делителя с коэффициентом пересчета, определяемым двоичным числом, подаваемым на линии данных D[7..0].

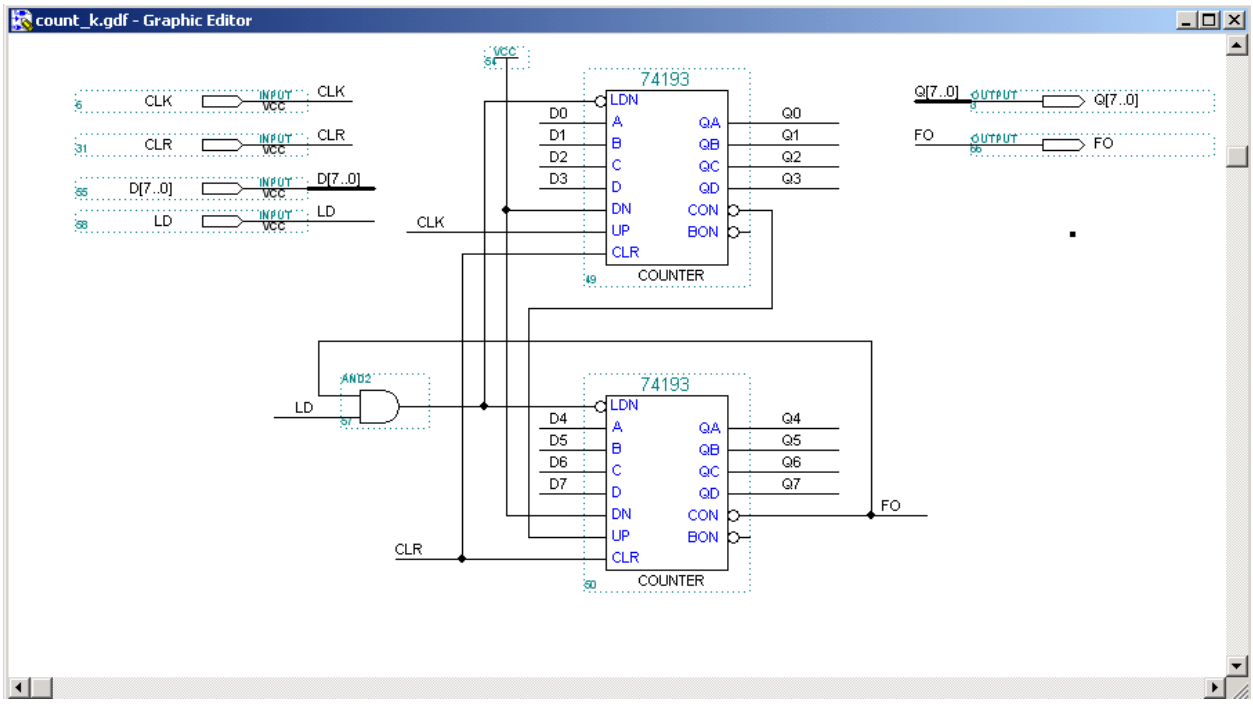



Рис. 62. Схема делителя частоты

4. Выполняем временное моделирование синтезированной схемы (рис. 63).

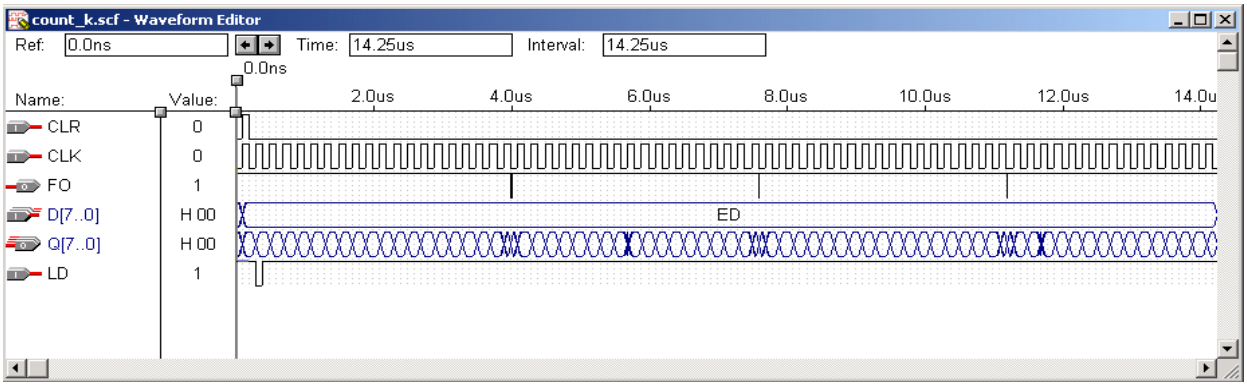



Рис. 63. Временная диаграмма работы синхронного счетчика с $К_{сч}$ = 19

5. Исследуем с помощью осциллографа разработанную схему. Для отображения входных и выходных сигналов делителя используем разные каналы осциллографа. Замерим длительность периода входного и выходного сигналов.

## 3.6. Конечные автоматы

«Конечный автомат» – это общее название последовательностных цифровых схем, которые принимают следующее состояние в зависимости от входных сигналов и текущего состояния. Кроме того, «конечные автоматы» характеризуются конечным числом состояний. Простейший «конечный автомат» – это триггер. Слово «тактируемый» указывает на тот факт, что элементы памяти в конечном автомате (триггеры) имеют тактовый вход, а слово «синхронный» означает, что на тактовые входы всех триггеров подается один и тот же тактовый сигнал. Состояние такого конечного автомата изменяется только на очередном такте, а между тактами остается неизменным.

### 3.6.1. Тактируемые синхронные конечные автоматы

На рис. 64 приведена общая структура тактируемого синхронного конечного автомата. Память состояния представляет собой набор из *n*-триггеров, в которых хранится текущее состояние автомата; всего имеется $2^n$ различных состояний. Все триггеры подключены к общему источнику тактового сигнала, который позволяет им изменять состояние на каждом такте. Следующее состояние конечного автомата определяется логикой (функцией) переходов F и является функцией текущего состояния и входного воздействия. Выходные сигналы определяются выходной логикой G и также зависят от текущего состояния и входного воздействия. Оба блока F и G являются строго комбинационными схемами.

Пусть

$A = \{a_1, a_2, .., a_q\}$ – множество внутренних состояний конечного автомата;

$Z = \{z_1, z_2, .., z_n\}$ – множество входных сигналов;

$W = \{w_1, w_2, .., w_m\}$ – множество выходных сигналов.

Тогда

$$a_s = F\,(a_m, z),$$

где $a_m$ – состояние автомата в момент времени $t$;

$z$ – входной сигнал в момент времени $t$;

$a_s$ – состояние автомата в момент времени $t$+1.

$$w_m = G\,(a_q, z_n).$$

Последовательностная схема (рис. 64), выход которой зависит как от состояния, так и от входа, называется *автоматом Мили* (Mealy machine).

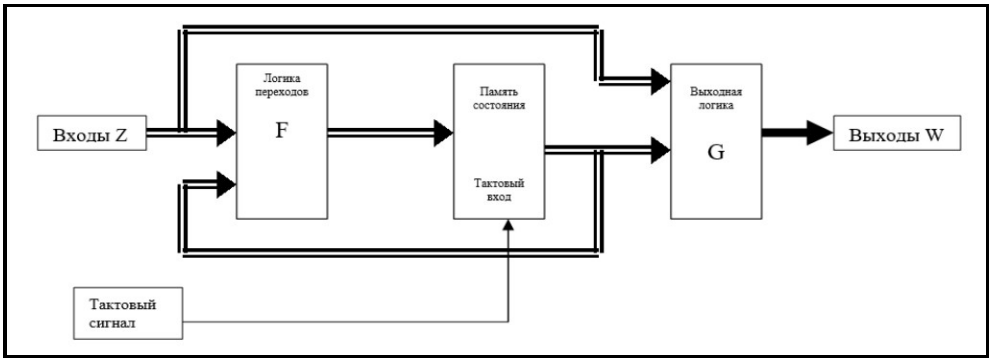



Рис. 64. Структура конечного автомата (автомат Мили)

Память состояния конечного автомата может быть построена на D-триггерах, переключающихся по положительному фронту. В этом случае все события происходят в моменты времени, соответствующие нарастающим фронтам тактового сигнала. Возможно также использование в памяти состояния JK-триггеров и RS-триггеров.

В некоторых приложениях, где используются конечные автоматы, выход зависит только от текущего состояния автомата:

$$w_m = G\,(a_q).$$

Такая схема (рис. 65) называется *автоматом Мура* (Moore machine).

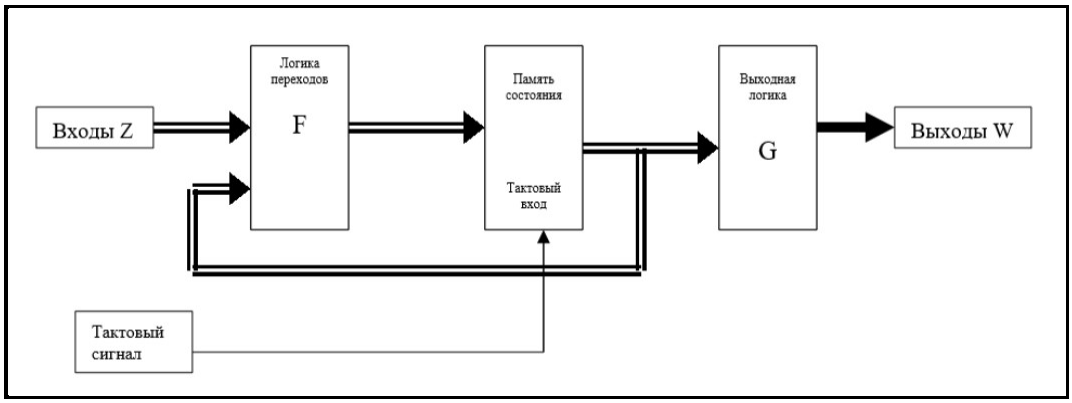



Рис. 65. Структура конечного автомата (автомата Мура)

Единственное различие между автоматом Мили и автоматом Мура заключается в том, как вырабатываются выходные сигналы.

Функциональное поведение триггера можно описать формально с помощью характеристического уравнения, посредством которого следующее состояние триггера задается как функция его текущего состояния и значений сигналов на его входах. В табл. 9 перечислены характеристические уравнения для различных триггеров. Принято считать, что символ * (звездочка) в записи $Q^*$ означает «следующее значение Q».

Таблица 9. **Характеристические уравнения триггерных устройств**

| Тип схемы | Характеристическое уравнение |
|---|---|
| Переключающийся по фронту D-триггер | $Q^* = D$ |
| D-триггер с входом разрешения | $Q^* = (EN \,\&\, D) + (\overline{EN} \,\&\, Q)$ |
| Двухтактный RS-триггер | $Q^* = S + (\overline{R} \,\&\, Q)$ |
| Двухтактный JK-триггер | $Q^* = (J \,\&\, \overline{Q}) + (\overline{K} \,\&\, Q)$ |
| Переключающийся по фронту JK-триггер | $Q^* = (J \,\&\, \overline{Q}) + (\overline{K} \,\&\, Q)$ |
| T-триггер | $Q^* = \overline{Q}$ |
| T-триггер с входом разрешения | $Q^* = (EN \,\&\, \overline{Q}) + (\overline{EN} \,\&\, Q)$ |

Анализ тактируемых синхронных конечных автоматов выполняется в три основных этапа:

- определяются функции переходов и выхода F и G;
- функции F и G используются для построения таблицы «состояние/выход», которой полностью задаются следующие состояния и выходы схемы при всех возможных комбинациях текущего состояния и текущих значений входных сигналов;
- вычерчивается диаграмма состояний, представляющая информацию, полученную на предыдущем шаге, в графической форме.

Последний этап не обязательный.

### 3.6.2. Микропрограммные автоматы на ПЗУ

На основе микропрограммных автоматов можно строить устройства, которые работают по довольно сложным алгоритмам и выполняют различные функции, определяемые входными сигналами. В микропрограммном автомате выходные сигналы зависят от входных сигналов и от текущего состояния автомата. Но логика работы памяти состояния автомата задается микропрограммой, зашитой в ПЗУ. Структура микропрограммных автоматов (рис. 66), как правило, содержит три основных узла:

- постоянное запоминающее устройство;
- регистр, срабатывающий по фронту (например, на D-триггерах);
- узел, вырабатывающий тактовый сигнал.

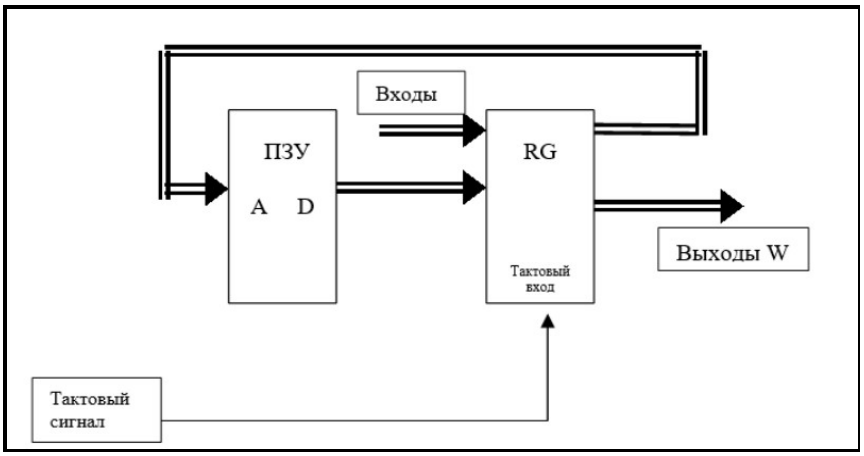



Рис. 66. Структура микропрограммного автомата

ПЗУ имеет M адресных разрядов и N разрядов данных. Если входы автомата имеют L разрядов, то регистр должен иметь (N+L) разрядов. Данные записываются в регистр RG по положительному фронту тактового сигнала. Часть выходных разрядов регистра используется для управления адресом ПЗУ, другая часть служит для формирования выходных сигналов.

Алгоритм и условия правильной работы схемы следующий:

- в каждом такте ПЗУ выдает код данных, определяя не только выходные сигналы, но и адрес ПЗУ в следующем такте;
- на формирование адреса в следующем такте влияют также входные сигналы;
- за один период тактовой частоты должны успеть сработать регистр и ПЗУ.

Отметим некоторые простые последовательные действия, из которых могут складываться алгоритмы работы микропрограммного автомата:

1) последовательный перебор адресов ПЗУ для выдачи последовательности выходных сигналов;
2) периодическое повторение последовательности адресов ПЗУ для повторения последовательности выходных сигналов;
3) остановка в каком-то адресе ПЗУ для ожидания изменения входного сигнала.

На микропрограммных автоматах, разработанных по структурной схеме (см. рис. 66), можно построить сложные устройства. Например, можно построить цифровой генератор функции или автомат, декодирующий код Манчестер-II в двоичный.

### 3.6.3. Разработка микропрограммного автомата в пакете Quartus Prime

Задача разрабатываемого микропрограммного автомата заключается в управлении цифровыми объектами по заданной таблице состояний. Сам алгоритм работы автомата, фактически «программа», будет реализован в «железе». В качестве объектом будем использовать D-триггер со входами сброса

и установки. Обозначим объекты через Т (Т1…Т4) и выполним анализ таблицы состояний объектов (табл. 10), которая определяет поведение объектов во времени.

Таблица 10. **Таблица состояний объектов**

| Объект | Номер такта | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| Т1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| Т2 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| Т3 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Т4 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |

По приведенной таблице состояний составим таблицу управляющих сигналов. Поскольку объектами являются триггеры, то удобно управлять ими через R-входы (установка в ‘0’) и S-входы (установка в ‘1’), подавая на них управляющие сигналы. Управляющие сигналы должны иметь активный уровень логический «0», т.к. R- и S-входы D-триггеров инверсны.

В табл. 11 показано, при каком состоянии адресного счетчика необходимо формировать те или иные управляющие сигналы. В таблице: Q1, Q2, Q3, Q4 – выходы счетчика, Т1, Т2, Т3, Т4 – триггеры, S1, S2, S3, S4 – управляющие сигналы, подаваемые на S-входы триггеров, R1, R2, R2, R4 – управляющие сигналы, подаваемые на R-входы триггеров. Синтезируем функции R и S для соответствующих объектов, согласно состояниям объектов, заданных в табл. 10. В нашем примере эти функции достаточно простые. В более сложных задачах комбинационная логика минимизируется с помощью карт Карно.

Таблица 11. **Соответствие кодовых комбинаций выходов счетчика и управляющих сигналов**

| Q4 | Q3 | Q2 | Q1 | Номер такта | Т4 | Т3 | Т2 | Т1 |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | | | | |
| 0 | 0 | 0 | 1 | 1 | | | | |
| 0 | 0 | 1 | 0 | 2 | | | | |
| 0 | 0 | 1 | 1 | 3 | S4 | | | S1 |
| 0 | 1 | 0 | 0 | 4 | | | S2 | |
| 0 | 1 | 0 | 1 | 5 | R4 | S3 | | R1 |
| 0 | 1 | 1 | 0 | 6 | | | | |
| 0 | 1 | 1 | 1 | 7 | | | R2 | S1 |
| 1 | 0 | 0 | 0 | 8 | | | | |
| 1 | 0 | 0 | 1 | 9 | | R3 | | |
| 1 | 0 | 1 | 0 | 10 | S4 | | | |
| 1 | 0 | 1 | 1 | 11 | | | S2 | |
| 1 | 1 | 0 | 0 | 12 | | | | R1 |
| 1 | 1 | 0 | 1 | 13 | | | | |
| 1 | 1 | 1 | 0 | 14 | | | R2 | |
| 1 | 1 | 1 | 1 | 15 | R4 | | | |

Из табл. 11 видно, что некоторые управляющие сигналы за весь цикл работы автомата (16 тактов) формируются несколько раз (например R1, S1 и т.д.). Такие сигналы с одинаковым индексом можно объединить. Составим логические выражения для управляющих сигналов:

$$R1 = Y5 \,\&\, Y12 \qquad S1 = Y3 \,\&\, Y7$$
$$R2 = Y2 \,\&\, Y14 \qquad S2 = Y4 \,\&\, Y11$$
$$R3 = Y9 \qquad S3 = Y5$$
$$R4 = Y5 \,\&\, Y15 \qquad S4 = Y3 \,\&\, Y10$$

Разработаем в пакете Quartus Prime схему микропрограммного цифрового автомата (рис. 67). На счетный вход счетчика подадим тактовый сигнал CLK, а на вход сброса триггеров и счетчика подадим сигнал CLR.

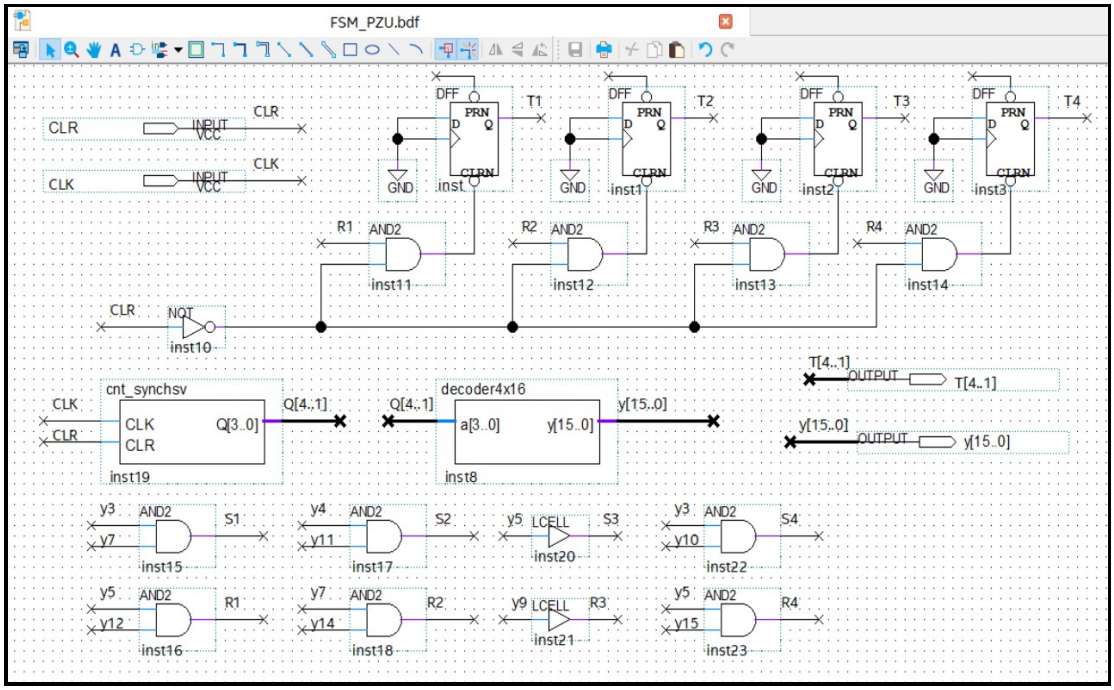



Рис. 67. Цифровой микропрограммный автомат

Выходную логику микропрограммного автомата построим на дешифраторе 4×16 с позиционным кодом на выходе. Позиционный код содержит всего один выходной разряд в состоянии логического «0» в каждом такте (например, для второго такта выходной код будет 1111111111111101). Адресный счетчик цифрового автомата необходим для формирования дешифратором кода, соответствующего текущему такту. Дешифратор и адресный счетчик напишем на языке SystemVerilog (рис. 68).

decoder4x16.sv

```
module decoder4x16 (input logic [3:0]a,
                    output logic [15:0]y);

  always_comb
  case(a)

    0: y=16'b11111111_11111110;
    1: y=16'b11111111_11111101;
    2: y=16'b11111111_11111011;
    3: y=16'b11111111_11110111;
    4: y=16'b11111111_11101111;
    5: y=16'b11111111_11011111;
    6: y=16'b11111111_10111111;
    7: y=16'b11111111_01111111;
    8: y=16'b11111110_11111111;
    9: y=16'b11111101_11111111;
   10: y=16'b11111011_11111111;
   11: y=16'b11110111_11111111;
   12: y=16'b11101111_11111111;
   13: y=16'b11011111_11111111;
   14: y=16'b10111111_11111111;
   15: y=16'b01111111_11111111;

  endcase
  endmodule
```

cnt_synchsv.sv

```
module cnt_synchsv (
    input logic CLK,
    input logic CLR,
    output reg [3:0] Q);

    always_ff@(posedge CLK or posedge CLR)
   if(CLR)
     Q <= 4'd0;
   else
     Q <= Q + 1'd1;

endmodule
```

Рис. 68. Программа на языке SystemVerilog дешифратора 4×16 (слева) и синхронного счетчика

## Упражнения

**У1.** Провести синтез комбинационной логической схемы, реализующей функцию $y = f(x_1, x_2, x_3, x_4)$, в базисах И-ИЛИ-НЕ. Функция $y = f(x_1, x_2, x_3, x_4)$ задана входными наборами таблицы истинности, определяемой вариантом.

Выполнить следующие действия:

- записать СДНФ (совершенную дизъюнктивную нормальную форму функции);
- минимизировать заданную функцию с помощью карт Карно;
- в пакете Quartus Prime создать проект, из библиотечных элементов нарисовать получившуюся минимальную функцию в графическом редакторе Block Diagram/Schematic File;
- выполнить компиляцию проекта;
- в редакторе University Program VWF выполнить временное симулирование КЛС;
- загрузить конфигурационный файл в ПЛИС стенда;

- с помощью входных переключателей стенда (см. «Руководство по эксплуатации учебного стенда» CLE-148) задавать входные последовательности КЛС (переменные $x_1$, $x_2$, $x_3$, $x_4$) и смотреть на светодиодном индикаторе значение функции ($y$).

**Варианты:**

| ВАРИАНТ 1 | | | | | |
|---|---|---|---|---|---|
| Vi | $x_4$ | $x_3$ | $x_2$ | $x_1$ | $y$ |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 2 | 0 | 0 | 1 | 0 | 1 |
| 3 | 0 | 0 | 1 | 1 | 0 |
| 4 | 0 | 1 | 0 | 0 | 1 |
| 5 | 0 | 1 | 0 | 1 | 0 |
| 6 | 0 | 1 | 1 | 0 | 1 |
| 7 | 0 | 1 | 1 | 1 | 0 |
| 8 | 1 | 0 | 0 | 0 | 1 |
| 9 | 1 | 0 | 0 | 1 | 1 |
| 10 | 1 | 0 | 1 | 0 | 0 |
| 11 | 1 | 0 | 1 | 1 | 0 |
| 12 | 1 | 1 | 0 | 0 | 0 |
| 13 | 1 | 1 | 0 | 1 | 1 |
| 14 | 1 | 1 | 1 | 0 | 0 |
| 15 | 1 | 1 | 1 | 1 | 1 |

| ВАРИАНТ 2 | | | | | |
|---|---|---|---|---|---|
| Vi | $x_4$ | $x_3$ | $x_2$ | $x_1$ | $y$ |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 |
| 2 | 0 | 0 | 1 | 0 | 0 |
| 3 | 0 | 0 | 1 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 | 0 |
| 6 | 0 | 1 | 1 | 0 | 0 |
| 7 | 0 | 1 | 1 | 1 | 0 |
| 8 | 1 | 0 | 0 | 0 | 1 |
| 9 | 1 | 0 | 0 | 1 | 1 |
| 10 | 1 | 0 | 1 | 0 | 0 |
| 11 | 1 | 0 | 1 | 1 | 1 |
| 12 | 1 | 1 | 0 | 0 | 1 |
| 13 | 1 | 1 | 0 | 1 | 1 |
| 14 | 1 | 1 | 1 | 0 | 0 |
| 15 | 1 | 1 | 1 | 1 | 0 |

| ВАРИАНТ 3 | | | | | |
|---|---|---|---|---|---|
| Vi | $x_4$ | $x_3$ | $x_2$ | $x_1$ | $y$ |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 2 | 0 | 0 | 1 | 0 | 0 |
| 3 | 0 | 0 | 1 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 | 1 |
| 6 | 0 | 1 | 1 | 0 | 0 |
| 7 | 0 | 1 | 1 | 1 | 1 |
| 8 | 1 | 0 | 0 | 0 | 0 |
| 9 | 1 | 0 | 0 | 1 | 1 |
| 10 | 1 | 0 | 1 | 0 | 0 |
| 11 | 1 | 0 | 1 | 1 | 0 |
| 12 | 1 | 1 | 0 | 0 | 0 |
| 13 | 1 | 1 | 0 | 1 | 1 |
| 14 | 1 | 1 | 1 | 0 | 0 |
| 15 | 1 | 1 | 1 | 1 | 1 |

| ВАРИАНТ 4 | | | | | |
|---|---|---|---|---|---|
| Vi | $x_4$ | $x_3$ | $x_2$ | $x_1$ | $y$ |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 2 | 0 | 0 | 1 | 0 | 0 |
| 3 | 0 | 0 | 1 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 | 0 |
| 6 | 0 | 1 | 1 | 0 | 0 |
| 7 | 0 | 1 | 1 | 1 | 1 |
| 8 | 1 | 0 | 0 | 0 | 1 |
| 9 | 1 | 0 | 0 | 1 | 1 |
| 10 | 1 | 0 | 1 | 0 | 0 |
| 11 | 1 | 0 | 1 | 1 | 0 |
| 12 | 1 | 1 | 0 | 0 | 0 |
| 13 | 1 | 1 | 0 | 1 | 1 |
| 14 | 1 | 1 | 1 | 0 | 0 |
| 15 | 1 | 1 | 1 | 1 | 1 |

| ВАРИАНТ 5 | | | | | |
|---|---|---|---|---|---|
| Vi | $x_4$ | $x_3$ | $x_2$ | $x_1$ | $y$ |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 2 | 0 | 0 | 1 | 0 | 0 |
| 3 | 0 | 0 | 1 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 | 1 |
| 5 | 0 | 1 | 0 | 1 | 1 |
| 6 | 0 | 1 | 1 | 0 | 0 |
| 7 | 0 | 1 | 1 | 1 | 1 |
| 8 | 1 | 0 | 0 | 0 | 0 |
| 9 | 1 | 0 | 0 | 1 | 1 |
| 10 | 1 | 0 | 1 | 0 | 0 |
| 11 | 1 | 0 | 1 | 1 | 1 |
| 12 | 1 | 1 | 0 | 0 | 0 |
| 13 | 1 | 1 | 0 | 1 | 1 |
| 14 | 1 | 1 | 1 | 0 | 0 |
| 15 | 1 | 1 | 1 | 1 | 1 |

| ВАРИАНТ 6 | | | | | |
|---|---|---|---|---|---|
| Vi | $x_4$ | $x_3$ | $x_2$ | $x_1$ | $y$ |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 |
| 2 | 0 | 0 | 1 | 0 | 1 |
| 3 | 0 | 0 | 1 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 | 0 |
| 6 | 0 | 1 | 1 | 0 | 0 |
| 7 | 0 | 1 | 1 | 1 | 0 |
| 8 | 1 | 0 | 0 | 0 | 1 |
| 9 | 1 | 0 | 0 | 1 | 1 |
| 10 | 1 | 0 | 1 | 0 | 0 |
| 11 | 1 | 0 | 1 | 1 | 0 |
| 12 | 1 | 1 | 0 | 0 | 0 |
| 13 | 1 | 1 | 0 | 1 | 1 |
| 14 | 1 | 1 | 1 | 0 | 0 |
| 15 | 1 | 1 | 1 | 1 | 1 |

| ВАРИАНТ 7 | | | | | |
|---|---|---|---|---|---|
| Vi | $x_4$ | $x_3$ | $x_2$ | $x_1$ | $y$ |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 2 | 0 | 0 | 1 | 0 | 0 |
| 3 | 0 | 0 | 1 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 | 1 |
| 5 | 0 | 1 | 0 | 1 | 1 |
| 6 | 0 | 1 | 1 | 0 | 0 |
| 7 | 0 | 1 | 1 | 1 | 1 |
| 8 | 1 | 0 | 0 | 0 | 1 |
| 9 | 1 | 0 | 0 | 1 | 1 |
| 10 | 1 | 0 | 1 | 0 | 1 |
| 11 | 1 | 0 | 1 | 1 | 0 |
| 12 | 1 | 1 | 0 | 0 | 0 |
| 13 | 1 | 1 | 0 | 1 | 1 |
| 14 | 1 | 1 | 1 | 0 | 0 |
| 15 | 1 | 1 | 1 | 1 | 1 |

| ВАРИАНТ 8 | | | | | |
|---|---|---|---|---|---|
| Vi | $x_4$ | $x_3$ | $x_2$ | $x_1$ | $y$ |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 2 | 0 | 0 | 1 | 0 | 0 |
| 3 | 0 | 0 | 1 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 | 0 |
| 6 | 0 | 1 | 1 | 0 | 0 |
| 7 | 0 | 1 | 1 | 1 | 1 |
| 8 | 1 | 0 | 0 | 0 | 1 |
| 9 | 1 | 0 | 0 | 1 | 1 |
| 10 | 1 | 0 | 1 | 0 | 1 |
| 11 | 1 | 0 | 1 | 1 | 0 |
| 12 | 1 | 1 | 0 | 0 | 0 |
| 13 | 1 | 1 | 0 | 1 | 1 |
| 14 | 1 | 1 | 1 | 0 | 1 |
| 15 | 1 | 1 | 1 | 1 | 1 |

**У2.** Провести синтез дешифратора двоичного 3-разрядного входного кода в семисегментный выходной код. Функция преобразования определяется таблицей истинности соответствующего варианта.

Выполнить следующие действия:

- записать СДНФ для каждого выхода дешифратора;
- минимизировать заданную функцию с помощью карт Карно для каждого выхода;
- в пакете Quartus Prime создать проект, из библиотечных элементов нарисовать схему дешифратора в графическом редакторе Block Diagram/Schematic File;
- выполнить компиляцию проекта;
- в редакторе University Program VWF выполнить временное симулирование схемы;
- загрузить конфигурационный файл в ПЛИС стенда;
- с помощью входных переключателей стенда (см. «Руководство по эксплуатации учебного стенда CLE-148») задать входные последовательности дешифратора и смотреть на индикаторе отображение символов, заданных таблицей истинности.

**Варианты:**

ВАРИАНТ 1

| $X_2$ | $X_1$ | $X_0$ | A | B | C | D | E | F | G | Символ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 3 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 4 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 5 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 6 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 7 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 8 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 9 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | A |

ВАРИАНТ 2

| $X_2$ | $X_1$ | $X_0$ | A | B | C | D | E | F | G | Символ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | C |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | D |
| 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | E |
| 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | F |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | O |
| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 2 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 3 |

ВАРИАНТ 3

| $X_2$ | $X_1$ | $X_0$ | A | B | C | D | E | F | G | Символ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 6 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 7 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 8 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 9 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | A |
| 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | B |
| 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | C |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | D |

ВАРИАНТ 4

| $X_2$ | $X_1$ | $X_0$ | A | B | C | D | E | F | G | Символ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 9 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | A |
| 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | B |
| 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | C |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | D |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | E |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | F |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | O |

ВАРИАНТ 5

| $X_2$ | $X_1$ | $X_0$ | A | B | C | D | E | F | G | Символ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 2 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 3 |
| 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | C |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | D |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | E |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | F |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | O |

ВАРИАНТ 6

| $X_2$ | $X_1$ | $X_0$ | A | B | C | D | E | F | G | Символ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 9 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | A |
| 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | B |
| 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | C |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | D |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 6 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 7 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 8 |

ВАРИАНТ 7

| $X_2$ | $X_1$ | $X_0$ | A | B | C | D | E | F | G | Символ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 9 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | A |
| 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | B |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 2 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 3 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 7 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 8 |

ВАРИАНТ 8

| $X_2$ | $X_1$ | $X_0$ | A | B | C | D | E | F | G | Символ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 9 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | A |
| 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 4 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 5 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 6 |
| 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 7 |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 2 |

**У3.** Разработать проект цифровой схемы в пакете Quartus Prime, включающей 8 независимых D-триггеров с асинхронным сбросом, используя логические элементы.

**У4.** Спроектировать D-защелку с асинхронным сбросом, используя логические элементы.

**У5.** Разработать проект цифровой схемы в пакете Quartus Prime – логическую схему восьмиразрядного счетчика, реализованную на D-триггерах. Выполнить функциональную и временную симуляцию и проверить работоспособность схемы.

**У6.** Разработать проект цифровой схемы в пакете Quartus Prime, включающей 8 независимых D-триггеров с асинхронным сбросом, используя логические элементы.

**У7.** Разработать проект цифровой схемы в пакете Quartus Prime, включающий делитель частоты с переменным коэффициентом деления на синхронных счетчиках. Коэффициент деления $К_{сч}$ задается соответствующим вариантом.

Выполнить следующие действия:

- в пакете Quartus Prime создать проект, включающий синхронный счетчик 74193 из библиотеки пакета Quartus Prime (megafunctions → others → maxplus2 → 74193);
- в графическом редакторе Block Diagram/Schematic File разработать схему и выполнить компиляцию проекта;
- в редакторе University Program VWF задать входные и выходные сигналы счетчика и выполнить функциональное и временное симулирование схемы;
- с помощью входных переключателей стенда (см. «Руководство по эксплуатации учебного стенда CLE-148») задать коды делителя, соответствующие коэффициенту деления $К_{сч}$, и смотреть на осциллографе входные и выходные частоты делителя.

**Варианты:**

1) $К_{сч}$ = 53,, 2) $К_{сч}$ = 24, 3) $К_{сч}$ = 19, 4) $К_{сч}$ = 109, 5) $К_{сч}$ = 135, 6) $К_{сч}$ = 106, 7) $К_{сч}$ = 48, 8) $К_{сч}$ = 38, 9) $К_{сч}$ = 74, 10) $К_{сч}$ = 126, 11) $К_{сч}$ = 83, 12) $К_{сч}$ = 124, 13) $К_{сч}$ = 119, 14) $К_{сч}$ = 99, 15) $К_{сч}$ = 65, 16) $К_{сч}$ = 145.

**У8.** Разработать в пакете Quartus Prime 32-разрядный синхронный реверсивный счетчик (Up/Down counter). Он имеет входы Reset и Up. Когда вход Reset установлен в “1”, все выходы сбрасываются в “0”. В противном случае, если Up = 1, счетчик увеличивает свое значение, а когда Up = 0 – уменьшает.

**У9.** Разработать проект цифрового автомата в пакете Quartus Prime, управляющего набором объектов и работающего по цикловому алгоритму. В качестве объектов управления использовать четыре D-триггера. Поведение объектов задается таблицей состояний соответствующего варианта. Нарисовать циклограмму работы автомата.

Выполнить следующие действия:

- в графическом редакторе Block Diagram/Schematic File пакета Quartus Prime создать проект, реализующий цифровой автомат для управления четырьмя объектами (поведение объектов индицировать на светодиодах стенда CLE-148);
- выполнить компиляцию проекта;
- в редакторе University Program VWF задать входные и выходные сигналы цифрового автомата и выполнить временное симулирование схемы (сравнить временную диаграмму работы автомата и заданную таблицу состояний объектов);
- загрузить конфигурационный файл в ПЛИС;
- проверить правильность работы автомата, наблюдая за работой объектов с помощью светодиодов стенда.

**Варианты:**

| Объект | ВАРИАНТ 1 | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Номер такта | | | | | | | | | | | | | | | |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| T1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| T2 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| T3 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| T4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |

| Объект | ВАРИАНТ 2 | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Номер такта | | | | | | | | | | | | | | | |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| T1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| T2 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| T3 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| T4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |

| Объект | ВАРИАНТ 3 | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Номер такта | | | | | | | | | | | | | | | |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| T1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| T2 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| T3 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| T4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |

| Объект | ВАРИАНТ 4 | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Номер такта | | | | | | | | | | | | | | | |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| T1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| T2 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| T3 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| T4 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Объект | ВАРИАНТ 5 | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Номер такта | | | | | | | | | | | | | | | |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| T1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| T2 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| T3 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| T4 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |

| Объект | ВАРИАНТ 6 | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Номер такта | | | | | | | | | | | | | | | |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| T1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| T2 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| T3 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| T4 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |

| Объект | ВАРИАНТ 7 | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Номер такта | | | | | | | | | | | | | | | |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| T1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| T2 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| T3 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| T4 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |

| Объект | ВАРИАНТ 8 | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Номер такта | | | | | | | | | | | | | | | |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| T1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| T2 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| T3 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| T4 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |

## Контрольные вопросы

1. Какими способами можно задать комбинационную схему?
2. Напишите программу на языке SystemVerilog цифрового устройства «дешифратор» размерности 3×8.
3. Чем отличается устройство «защелка» от «триггера»?
4. Что такое «регистр» и чем отличаются параллельные, последовательные (сдвиговые) и параллельно-последовательные регистры?
5. Чем отличается синхронный счетчик от асинхронного счетчика?
6. Какое устройство называется «конечный автомат»? Приведите примеры конечных автоматов.
7. Чем отличается автомат Мура от автомата Мили?
8. Напишите характеристическое уравнение D-триггера.

# Глава 4. Разработка проектов в пакете Quartus Prime для цифрового управления исполнительными устройствами

Реализация модели управления технологическим процессом предполагает наличие датчиков, поставляющих информацию о состоянии технологического процесса, вычислителя, обрабатывающего информацию от датчиков и управляющего исполнительными устройствами, с помощью которых изменяются параметры процесса. Любая цифровая система управления – это, как правило, синхронная система, выполненная на различных платформах: ПЛИС, микроконтроллеры, ПЛК и т.д. Проектирование цифровых систем достаточно сложная задача. Для обеспечения правильной синхронизации всех элементов системы необходимо при проектировании придерживаться определенных правил. В работе [29] сформулированы основные подходы для согласования работы отдельных компонентов во времени при проектировании синхронных устройств:

1. В пределах домена синхронизации все изменения состояния происходят по одному фронту одного тактового сигнала.
2. В рамках одного такта состояние изменяется только один раз. Неоднократное изменение нарушает правило 1.
3. Сигналы, поступающие извне домена, перед использованием должны быть синхронизированы с локальным тактовым генератором. Только в этом случае можно проводить временной анализ.
4. Сигналы сброса триггеров не используются для смены состояния. Они служат для приведения системы в начальное состояние (состояние после включения питания) и не являются частью нормальной процедуры смены состояния.

## 4.1. Примеры реализации нестандартных цифровых устройств на ПЛИС

Исходя из вышеперечисленных рекомендаций, одной из важных задач, которые решает цифровая система управления наряду с другими, – это синхронизация сигналов, поступающих от внешних датчиков, с внутренней тактовой частотой системы управления. Эту задачу иногда требуется разделить на две подзадачи: формирование входного импульса требуемой длительности из сигнала от датчика (а иногда и формирование требуемой амплитуды) и, собственно, сама синхронизация. Для правильного решения задачи синхронизации работы последовательностных устройств необходимо познакомиться с явлением, названным метастабильностью триггера. В последующих разделах рассмотрим решения этих задач, которые можно будет использовать при создании проектов на ПЛИС для управления исполнительными устройствами.

### 4.4.1. Метастабильность триггера

В § 3.5.3 даны понятие и структура D-триггера, представляющего собой цифровую схему, состоящую из двух D-защелок и двух инверторов. Эта схема имеет информационный D-вход и вход тактового сигнала CLK, по положительному фронту которого D-триггер меняет свое состояние на выходах Q и QN. На рис. 69 представлена временная диаграмма работы любой синхронной последовательностной схемы, построенной на базе триггеров. На этом рисунке входы схемы (inputs) связаны с D-входами, выходы (outputs) – с Q-выходами триггеров. На все триггеры подается один тактирующий сигнал CLK. В качестве основы синхронной последовательностной схемы может быть любой тактируемый (синхронный) триггер.

После перехода 0→1 тактового сигнала (переднего фронта тактового импульса) выход (или выходы) схемы могут начать изменяться не ранее, чем через время $t_{ccq}$ (задержка реакции CLK-to-Q), и должны принять стационарное значение не позднее чем через время $t_{pcq}$ (задержка распространения CLK-to-Q). Эти величины представляют собой наименьшую и наибольшую задержки схемы соответственно. Для того чтобы фиксация была выполнена корректно, информационный вход (или входы) схемы должен быть стабильным в течение некоторого времени предустановки (setup time) $t_{setup}$ перед передним фронтом тактового сигнала и не должен изменяться в течение времени удержания (hold time) $t_{hold}$ после переднего фронта тактового сигнала. Сумма времен предустановки и удержания называется апертурным временем схемы, это общее время, в течение которого информационный входной сигнал должен быть стабилен для его фиксации на выходе [1]. Невыполнение данных требований приводит к неопределенному состоянию триггера, которое и называется метастабильностью.

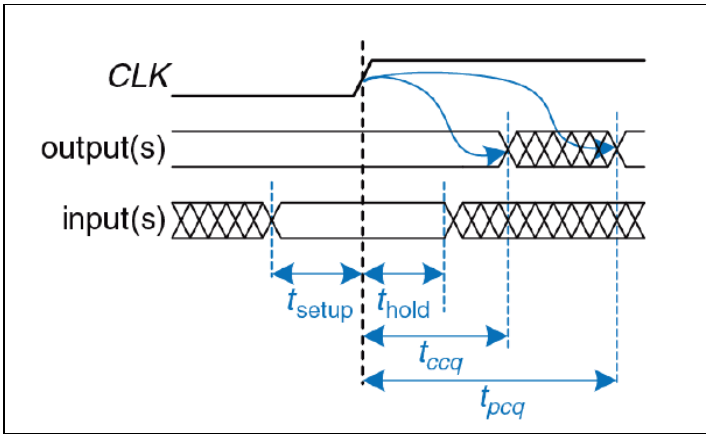



Рис. 69. Временная спецификация синхронной последовательностной схемы [1]

Таким образом, для правильной работы необходимо, чтобы сигналы на входах синхронной последовательностной схемы были стабильны в течение времени предустановки до фронта тактового импульса и времени удержания после этого фронта. Выполнение этих требований гарантирует, что в процессе фиксации значения информационного входа триггером он не будет изменяться.

Временные параметры работы триггера $t_{setup}$ и $t_{hold}$ зависят от того, как именно в микросхеме выполнен триггер, от технологии его изготовления, эксплуатационных режимов и нагрузки на выходах, особенно на тех выходах, которые не буферизированы. Чем более крутые фронты имеют входные сигналы, и чем меньше нестабильность фронтов у этих сигналов, тем меньшее время необходимо для предустановки сигнала. Появление метастабильности выхода триггера может вызвать асинхронный сброс, поэтому рекомендуется сигнал внешнего (асинхронного) сброса синхронизовать с тактовой частотой схемы, в которой используется триггер.

Изготовители ПЛИС нормируют параметры для своих триггеров. Для того чтобы определить вероятность отказа триггера, используется статистический параметр – среднее время безотказной работы (среднее время между отказами) (MTBF, Mean Time Between Failures), который и описывает характеристику метастабильности триггера. Изготовители рассчитывают среднее время безотказной работы в зависимости от ширины окна времени, в течение которого изменение во входном сигнале заставляет триггер становиться нестабильным. Кроме этого, для вычисления среднего времени безотказной работы используется частота входного сигнала и значение синхрочастоты, управляющей триггером [29].

### 4.1.2. Формирование импульса требуемой длительности из входного сигнала

Формирователи импульсов широко используются в измерительных устройствах, в схемах, где необходимо подавить дребезг коммутируемых механических устройств и т.д. Схема цифрового формирователя импульса из произвольного входного сигнала разрабатывается исходя из поставленной задачи. Это может быть формирователь с требуемой длительностью или параметрический формирователь с задаваемым параметром (при создании формирователя на базе ПЛИС). Все цифровые формирователи импульсов представляют из себя одновибраторы, реализованные с использованием кварцевого тактового генератора, триггеров, счетчиков. Такие формирователи можно разделить на две группы: в первом варианте выходной импульс формирователя синхронизируется с входным, и в этом случае длительность сформированного импульса может отличаться от заданной на величину меньшую периода тактового генератора, во втором – синхронизируется с тактовым генератором формирователя, и его длительность соответствует заданной (кратная периоду тактовой частоты).

Разработаем формирователь в пакете Quartus Prime на SystemVerilog. Создадим проект с именем **form_impulse** по методике, описанной в § 3.3.2. Верхний уровень проекта создадим в Block Diagram/Schematic File, а саму программу на SystemVerilog. Такой подход позволяет более наглядно проанализировать созданное устройство, и, кроме того, созданный из программы формирователя законченный символ с именем form_pulse позволяет использовать его в других проектах с заданием необходимого значения параметра N.

Формирователь позволяет сформировать выходной сигнал, привязанный к внутреннему clk-домену, из внешнего асинхронного события (сигнала). Длительность сформированного сигнала задается параметром N. При значении параметра N = 0, 1, 2, ... длительность выходного импульса равна N +1 тактов тактовой частоты clk. Кроме того, формирователь позволяет заблокировать действие повторяющихся внешних событий на заданное число тактов (равное длительности сформированного сигнала) до окончания текущего выходного импульса. Введение в программу параметра дли-

тельности N дает возможность задавать длительность импульса без выполнения перекомпиляции программы. На рис. 70 представлена структурная схема формирователя и его программная реализация.

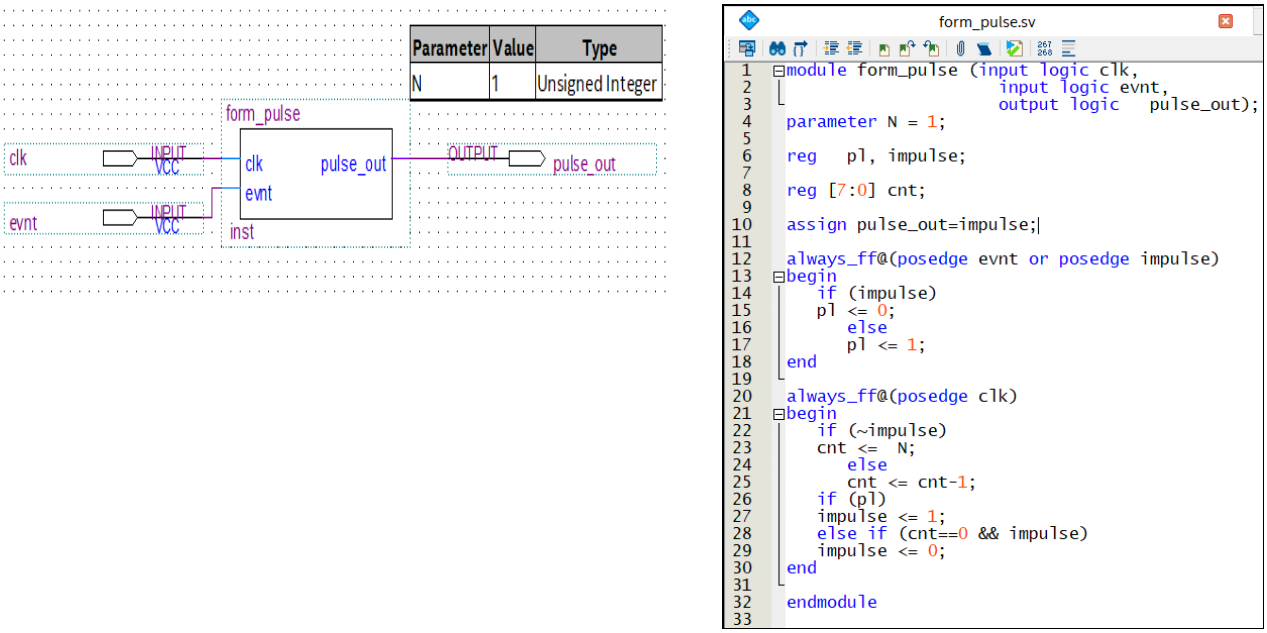



Рис. 70. Параметрический формирователь импульса из входного события evnt (параметр N = 1): блок схема (слева) и текст программы на языке SystemVerilog

Рассмотрим структуру программы формирователя. Как видно из перечня портов (рис. 70) в описании модуля (module form_pulse), он содержит два входа (clk – тактовый сигнал, evnt – внешнее событие) и один выход (pulse_out – сформированный импульс с заданной длительностью). Внутренние переменные pl и impuls являются одноразрядными регистрами, а cnt [7:0] – восьмиразрядным. Разрядность регистра cnt определяется максимальным значением параметра N, $0 \le N \le 255$.

В программе используются два блока кода, реализованных на операторах always_ff (об операторах always_ff более подробно рассказано в § 3.5.6). В списке чувствительности оператора always_ff@(posedge evnt or posedge impulse) анализируются передние фронты сигналов evnt и impulse. Конструкция if–else проверяет условие и решает, будет или нет выполнена часть кода. Если условие верно (impulse активный), код выполнится. Иначе выполнится другая часть кода (после else). Таким образом, выполнение кода внутри оператора always_ff, при активном сигнале impuls, присваивает переменной pl значение «0», в противном случае – “1”.

Второй блок always_ff в списке чувствительности содержит тактовый сигнал clk, и код этого оператора (блока операторов) выполняется только при появлении переднего фронта clk. Поскольку в коде этого оператора (always_ff) содержится несколько операторов, то этот код обрамляется конструкцией begin/end. Блок операторов содержит уже две конструкции if–else. В первой конструкции анализируется инверсное состояние однобитового регистра (триггера) impulse. Если ~impulse (инверсия impulse) равна «1», в регистр cnt [7:0] загружается значение параметра N, в противном случае выполняется код cnt<=cnt-1. Вторая конструкция представляет собой вложенное if, когда необходимо проверить несколько условий. В этой конструкции анализируется состояние переменной pl. Если pl = “1”, то переменной impulse присваивается значение “1”. В противном случае (else) выполняется проверка на равенство “0” регистра cnt [7:0]. Важно отметить, что все операторы always будут выполняться одновременно, когда одно или несколько условий, перечисленные в его описании, выполняются.

Оператор непрерывного присваивания assign работает следующим образом: как только переменные справа от знака «=» меняются, то результат слева от знака заново вычисляется. Таким образом, в выражении assign pulse_out = impulse при изменении переменной impulse меняется и переменная pulse_out. Если в программе используется несколько операторов assign, то они выполняются параллельно.

На рис. 71 представлено RTL-представление программы form_pulse, из которого видно, что на информационном входе регистра cnt [7:0] находится мультиплексор, передающий на регистр либо число N в инверсном коде, либо значение после сумматора. Выход регистра проверяется на равенство “0”. Условием выбора мультиплексора является инверсный сигнал impulse. Таким образом, реализуется код второго блока операторов always_ff. Из RTL-представления видно, что переменные pl и impuls реализованы на D-триггерах.

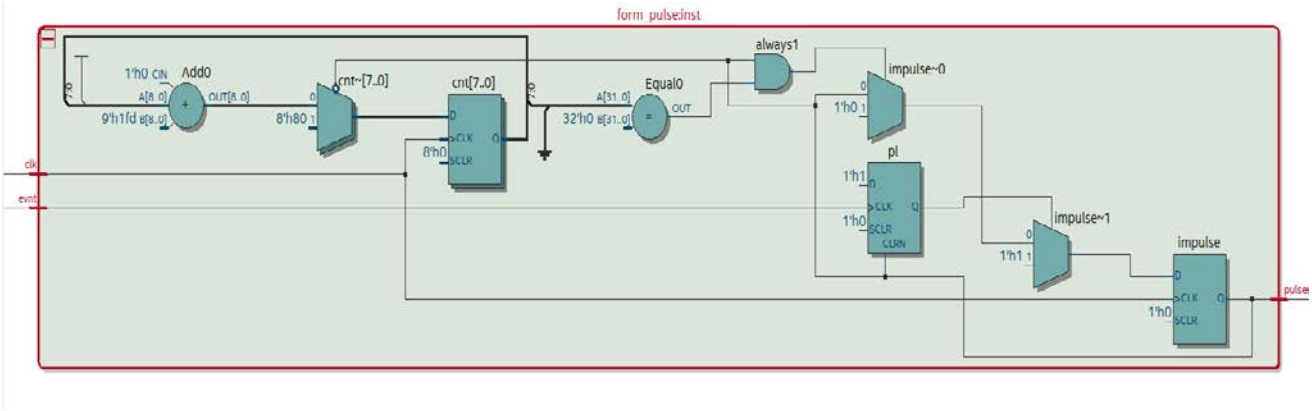



Рис. 71. RTL-представление программы form_pulse

Работа программы проверена на симуляторе. На рис. 72 представлено временное симулирование формирователя. При значении N = 1 длительность выходного импульса составляет два периода тактовой частоты. Минимальное время между сигналами evnt (между передними фронтами) составляет четыре периода сигнала clk или два периода после завершения предыдущего выходного импульса. Курсор, установленный на временной метке 130 ns, показывает запуск сформированного импульса, а следующий сигнал уже не формируется, поскольку событие evnt пришло раньше установленного минимального времени.

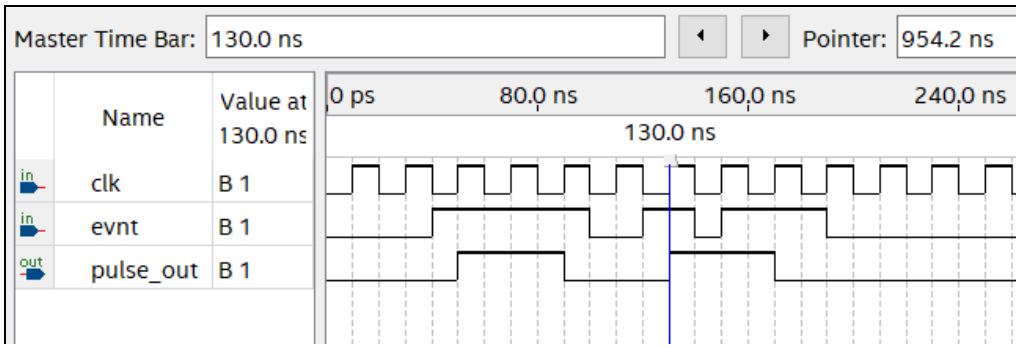



Рис. 72. Временная диаграмма работы формирователя

Кроме выполнения задачи формирования выходного сигнала заданной длительности реализованный формирователь решает еще одну очень важную задачу – синхронизацию асинхронного входного сигнала evnt с тактовой частотой clk, что необходимо для правильной работы синхронной схемы.

### 4.1.3. Делитель частоты

Рассмотрим два варианта разработки делителей частоты. Первый вариант будет реализован на библиотечной мегафункции ALTPLL пакета Quartus Prime, а вторую реализуем программно с возможностью параметрической настройки.

Вначале рассмотрим понятие PLL и принцип его работы. В работе [30] рассмотрена структура устройства, названная генератором со схемой подстройки частоты, который встроен в ПЛИС (рис. 73).

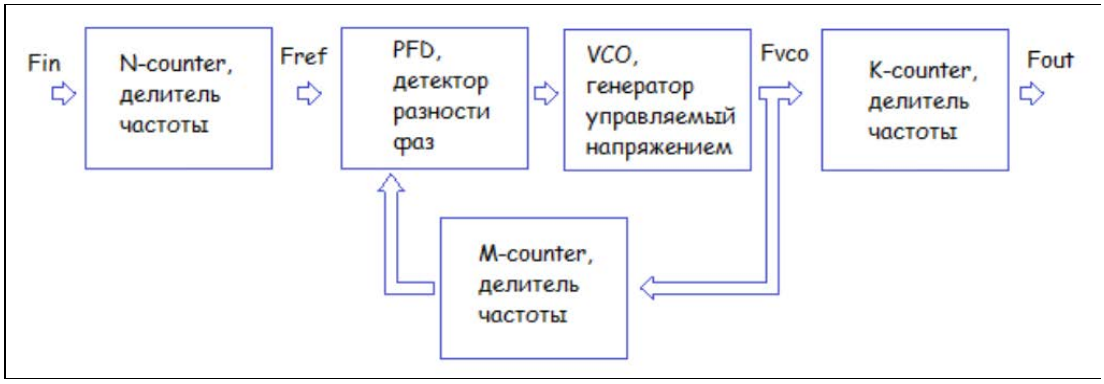



Рис. 73. Упрощенная структура PLL-устройства [23]

PLL (Phase-Locked Loop) – это специальный генератор со схемой подстройки частоты, это генератор, управляемый напряжением (VCO – voltage-controlled oscillator). В отечественной литературе эквивалентом PLL-устройства описывается устройство фазовой автоподстройки частоты (ФАПЧ), которая представляет собой систему автоматического регулирования, подстраивающей фазу управляемого генератора (VCO) так, чтобы она была равна фазе опорного сигнала либо отличалась на известную функцию от времени. Таким образом, в PLL-устройстве реализовано сравнение фаз сигнала входной частоты и сигнала выходной частоты. Измеренная разность фаз этих частот через отрицательную обратную связь как раз и управляет частотой генератора, фиксируя ее на заданном значении. На вход PLL-устройства подается входная частота опорного генератора Fin. Далее на счетчике N-counter опорная частота делится и получается другая частота Fref, которая поступает на фазовый детектор PFD (Phase-Frequency Detector). Фазовый детектор сравнивает фазы частот Fref с той, что поступает с делителя M-counter.

Разность фаз фильтруется и управляет генератором VCO. На выходе управляемого генератора – новая частота Fvco. Фазовый детектор подает управляющее воздействие на генератор VCO до тех пор, пока не выполнится условие Fref = Fvco/M. При этом условии частоты, подаваемые на детектор фаз, равны. Таким образом, например, если M=2, то частота генератора VCO должна получиться в 2 раза выше частоты Fref.

Последний этап – частота Fvco делится на выходном счетчике K-counter. Подбирая коэффициенты N, M, K, можно синтезировать довольно большой диапазон частот. Более сложные компоненты PLL позволяют перезагружать коэффициенты в процессе работы системы.

В основе стенда CLE-148 используется ПЛИС семейства Cyclone IVE фирмы Intel. В ПЛИС EP4CE22 семейства Cyclone IVE встроено 4 независимых блока PLL. Эти блоки можно каскадировать, используя специальные «глобальные входы». Есть возможность управлять фазой частоты на выходе PLL.

На рис. 74 представлена упрощенная блок-диаграмма PLL-устройства Cyclone IVE [31]. В этой структуре присутствуют те же компоненты, что и на рис. 73. По спецификации, представленной в [32], можно выбрать один из четырех источников опорной частоты (Fin), коэффициенты N, M и K, которые могут изменяться в диапазоне от 1 до 512. Cyclone IV PLL содержит 5 выходов: C0, C1, C2, C3, C4. Каждый из выходов может быть самостоятельно сконфигурирован на заданную частоту путем записи в выходные счетчики-делители (C0…C4) необходимого значения (максимальное 512).

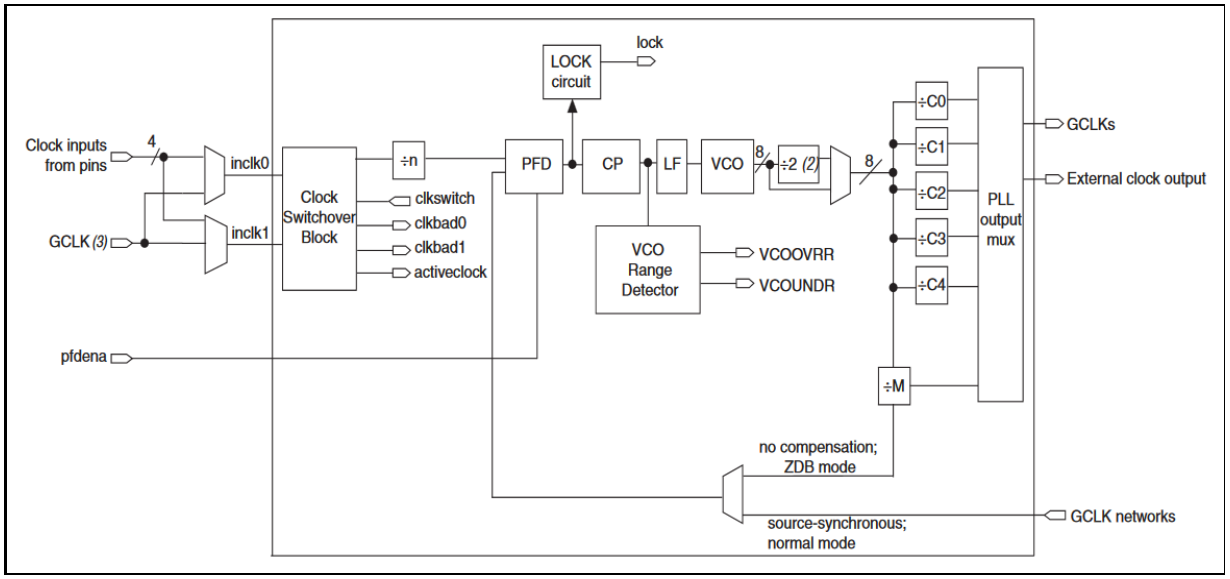



Рис. 74. Блок-диаграмма PLL-устройства Cyclone IV E [32]

Создадим проект делителя частоты в Quartus Prime под именем **pll_divider**, в основе которого будем использовать библиотечную мегафункцию ALTPLL. Присвоим проекту и файлу верхнего уровня имя **pll_divider**. Поставим задачу разработать делитель частоты, на входе которого будет частота 50 MHz, а выход должен иметь пять различных частот: 120 MHz, 10 MHz, 20 MHz, 30 MHz и 40 MHz. Делитель частоты будем разрабатывать в схемном редакторе. Создадим проект **pll_divider** по методике, предложенной в § 3.3.2. Не забудьте указать имя ПЛИС EP4CE22F17C6, входящей в состав DEO-Nano. Из библиотеки на рабочее поле вставим один входной и пять выходных портов. Мегафункцию ALTPLL будем брать из IP-каталога Tools → IP Catalog. С правой стороны рабочего

поля откроется дерево IP Catalog, из которого выберем Installed IP → Library → Basic Functions → → Clocks; PLLs and Resets → PLL → ALTPLL. В открывшемся окне присвоим имя pll_div и язык описания (Verilog) настраиваемой мегафункции. После нажатия «OK» откроется первое из двенадцати окон настройки ALTPLL (рис. 75).

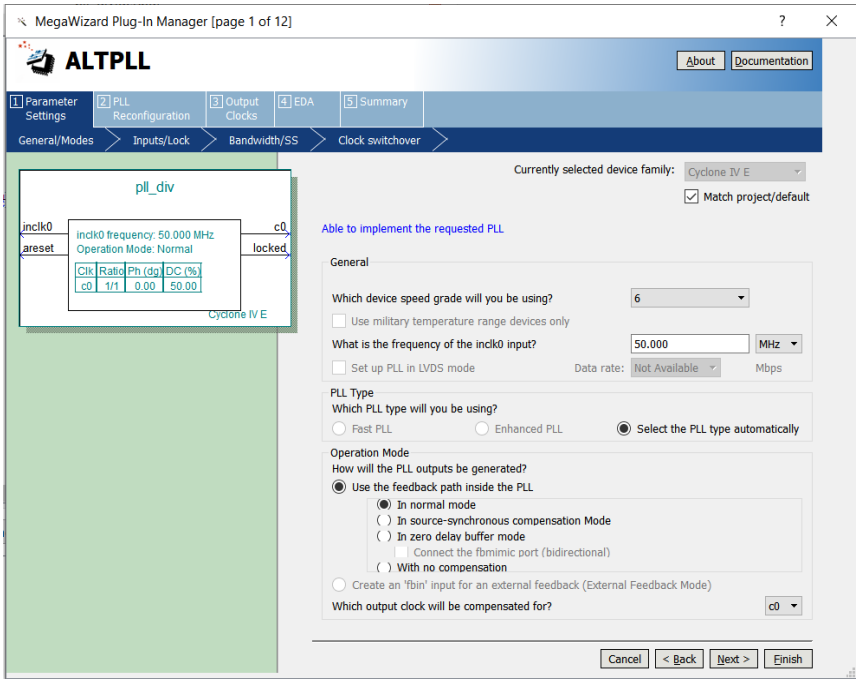



Рис. 75. Окно настройки ALTPLL

В этом окне необходимо указать значение входной частоты – частоты генератора, установленного на плате DEO-Nano (50 MHz), указать быстродействие (speed grade) ПЛИС, в нашем случае это значение равно 6. Кроме того, установим моду работы как нормальную (normal mode). Более подробную информацию по работе PLL и настройкам мегафункции ALTPLL можно посмотреть в документации Intel [32]. Во втором окне снимем галочки, устанавливающие асинхронный вход сброса PLL areset и выход locked. Окна 3, 4 и 5 пропустим. В окнах 6, 7, 8, 9 и 10 установим параметры частоты на выходах c0...c4 – запланированные выходные частоты 120 MHz, 10 MHz, 20 MHz, 30 MHz и 40 MHz и скважность выходных сигналов, равную 50. Окно 11 пропускаем, а в 12 отмечаем требуемые файлы для проекта. Нам необходим файл символа настроенной мегафункции ALTPLL (pll_div.bsf). Далее предлагается добавить выбранные файлы в проект. Символ ALTPLL с настроенными параметрами будет сохранен в библиотеке проекта в разделе Project. Добавим символ с именем pll_div на рабочее поле проекта. На рис. 76 представлен проект pll_divider, созданный в схемном редакторе, а на рис. 77 – временная диаграмма.

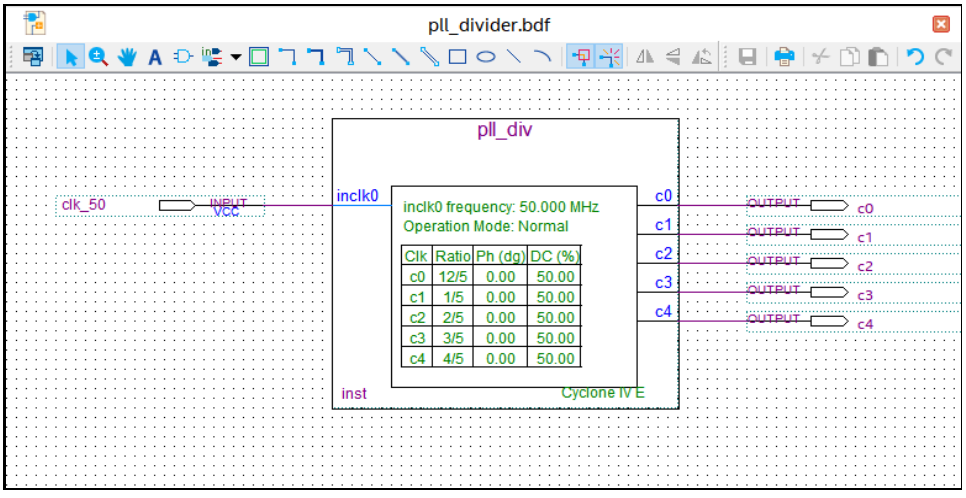



Рис. 76. Делитель частоты на базе мегафункции ALTPLL

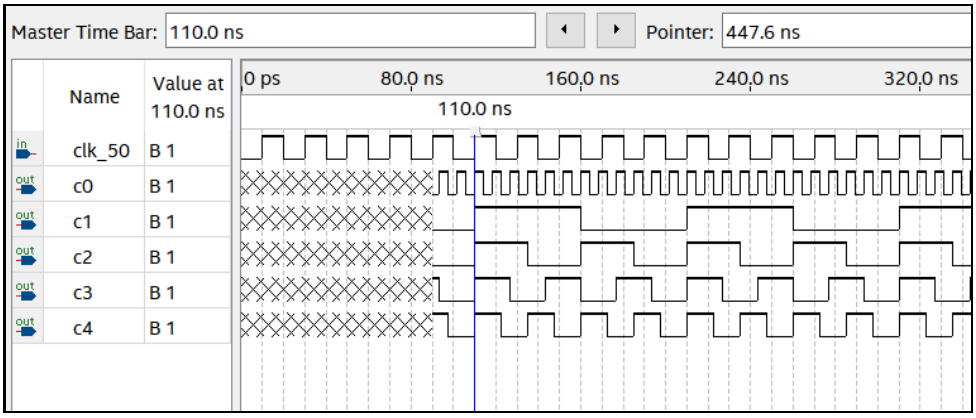



Рис. 77. Временная диаграмма работы делителя pll_divider

Из временной диаграммы видно, что выход на режим делителя составляет порядка 90 ns. Далее формируется заданный набор частот с синхронными фронтами.

Второй вариант делителя реализуем программно с возможностью параметрической настройки. Для этого создадим проект делителя частоты в Quartus Prime под именем clk_div на основе проекта, представленного в работе [33]. Поскольку у нас не будет задействована технология PLL, то значения выходных частот должны быть меньше входной частоты clk_50 = 50 MHz. Пусть это будут выходные частоты Fast = 10 MHz и Slow = 1 MHz. Частоту Fast = 10 MHz можно использовать для моделирования цифровых устройств с частотами реального генератора платы DEO-Nano, т.е. задавая частоту домена верхнего уровня, равной 50 MHz. Напишем программу на языке SystemVerilog с именем clk_divsv (рис. 78 справа) и создадим символ с этим же именем. Включим созданный символ в файл верхнего уровня clk_div.bdf (рис. 78 слева). Обозначим вход делителя как clk_50, а выходы – ce_fast и ce_slow. Особенность программы clk_divsv заключается в том, что в программе на этапе компиляции выполняются арифметические действия над параметрами clk, Fast и Slow. На рис. 79 выполнена временная симуляция делителя с разными значениями параметра Slow.

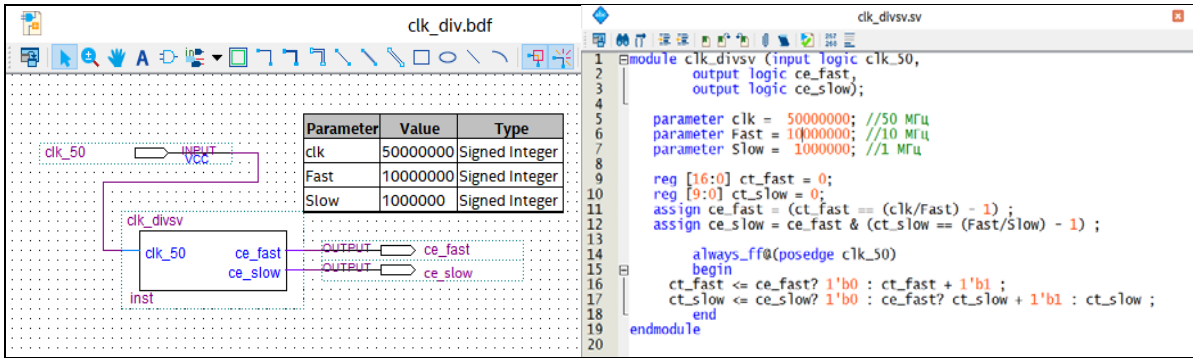



Рис. 78. Делитель частоты: справа программа clk_divsv, слева символ с входными и выходными портами

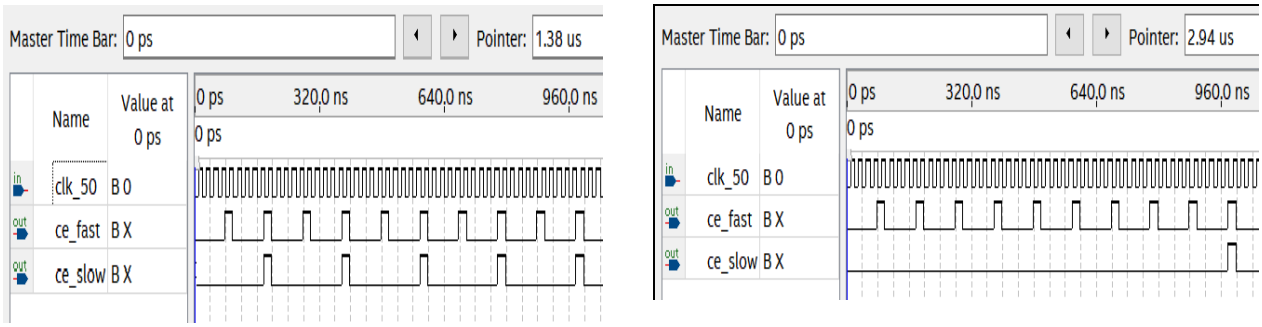



Рис. 79. Временная симуляция делителя частоты: слева параметр Slow = 5000000, справа – Slow = 1000000

## 4.2. Разработка ШИМ-управления двигателем постоянного тока на базе учебного стенда CLE-148

В гл. 2 дано описание учебного стенда CLE-148, с помощью которого можно изучать основы разработки проектов на платформе ПЛИС. Кроме этого, данный стенд по составу оборудования и по своим характеристикам позволяет разрабатывать достаточно сложные проекты для решения задач автоматизации [34], вычислительной обработке данных (например, сжатие [35] или фильтрация [36] данных в реальном времени). Для управления двигателями постоянного тока в составе стенда используется управляющая плата RPi Motor Driver.

Коллекторные двигатели постоянного тока являются наиболее доступными и распространенными в использовании в самых разнообразных электронных устройствах и элементах систем автоматизации. К основным достоинствам данных двигателей можно отнести доступную цену и простоту схем управления. Второе достоинство необходимо рассматривать в контексте его реального применения с конкретной нагрузкой и в конкретных условиях, а именно – строить схему управления в зависимости от типа решаемой задачи [37].

Если необходимо регулировать скорость коллекторного двигателя без ее стабилизации, то для этой цели используются как аналоговые, так и импульсные схемы прямого управления без обратной связи. Аналоговые регуляторы применяются для управления маломощными двигателями и выполняются, как правило, на основе схем стабилизации напряжения иногда с возможностью ограничения максимального тока для защиты двигателя и нагрузки. Но наиболее часто используются регуляторы с широтно-импульсной модуляцией (ШИМ). В отличие от аналоговых схемы управления с ШИМ обладают значительно более высоким КПД [38]. В некоторых применениях ШИМ-управления необходима стабилизация напряжения питания, т.к. постоянная составляющая их выходного напряжения зависит не только от отношения длительности импульса к периоду импульсной последовательности, но и от амплитуды.

ШИМ – метод, используемый для регулирования среднего тока, подаваемого на обмотку двигателя, с целью изменения скорости вращения. Можно говорить об управлении мощностью, подводимой к нагрузке. В рассматриваемом примере ШИМ нагрузкой является обмотка двигателя. Фактически процесс управления методом ШИМ заключается в изменении скважности импульсов управляющего сигнала. Скважность прямоугольного импульса – это отношение периода сигнала к длительности импульса. Если обозначить период сигнала через $T$, а длительность импульса через переменную $\tau$, то скважность $Q$ будет равна

$$Q = \frac{T}{\tau}. \tag{20}$$

Обратная величина скважности – это коэффициент заполнения $S$. Коэффициент заполнения удобно отображать в процентах:

$$S = \frac{\tau * 100\%}{T}. \tag{21}$$

Коэффициент заполнения импульсного сигнала равен времени (выраженного в процентах), в течении которого амплитуда сигнала в пределах одного периода соответствует логической «1». При коэффициенте заполнения, равном 50%, говорят, что импульсный сигнал представляет собой «меандр», т.е. в каждом периоде длительность импульса $\tau$ равна половине периода сигнала $T$.

В ШИМ период прямоугольного сигнала поддерживается постоянным, а время, в течение которого амплитуда сигнала остается логической «1», изменяется или модулируется. Рабочий цикл и среднее значение постоянного тока сигнала могут варьироваться. ШИМ является мощным методом управления аналоговыми цепями с помощью выхода цифровой системы. Метод ШИМ используется в различных разделах электроники и вычислительной техники:

- в телекоммуникации – где ширина импульсов соответствует определенным значениям данных, закодированных на одном конце и декодированных на другом;
- при регулировании напряжения – выходное напряжение в системе регулятора напряжения можно регулировать до требуемого уровня путем изменения режима работы ШИМ;
- при регулировании мощности – можно изменять среднюю мощность, которая является функцией модулированного рабочего цикла;
- при формировании звуковых и световых эффектов – изменение интенсивности в светодиоде является результатом изменения среднего напряжения и тока через светодиод из-за ШИМ.

В основе практически любой цифровой схемы управления на базе ШИМ присутствуют три основных элемента: регистр (регистр-счетчик), счетчик и цифровой компаратор (рис. 80).

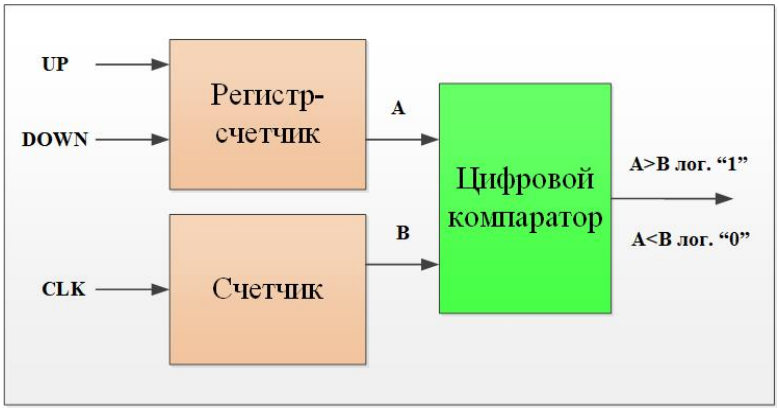



Рис. 80. Структура упрощенного цифрового ШИМ-модуля

«Регистр-счетчик» необходим для задания длительности импульса τ (переменная *A*) путем управления сигналами up-down состоянием встроенного счетчика. Накопительный «Счетчик» непрерывно увеличивает свое значение (переменная *B*) с каждым импульсом тактового сигнала clk. «Цифровой компаратор» сравнивает значения переменных *A* и *B* и формирует на выходе компаратора значение сигнала с заданным условием: если $A > B$, то логическая “1”, если $A < B$, то “0”. При фиксированной частоте сигнала clk период ШИМ-сигнала определяется разрядностью счетчика. Как было сказано выше, процесс управления методом ШИМ заключается в изменении скважности импульсов *Q* управляющего сигнала при неизменном периоде *T*. Период можно определить с помощью выражения

$$T = t * 2^n, \qquad (22)$$

где *t* – период тактового сигнала clk, а *n* – разрядность счетчика. Выбор разрядности счетчика определяется требованиями к градациям ширины импульса за период, чем больше разрядность, тем более плавная регулировка. Для нашего проекта выберем разрядность счетчика, равную 4, т.е. число градаций регулировки равняется 16. В § 2.2 при описании параметров платы RPi Motor Driver было сказано, что работа этой платы в режиме ШИМ возможна на частотах до 10 кГц. Отсюда нетрудно вычислить минимальное значение периода *T*, которое будет равно 100 мкс. Из выражения (22) можно вычислить период *t*. Период тактового сигнала *clk* равен $t = T / 2^n = 100 / 16 = 6{,}25$ мкс. Соответственно, частота clk равна $f_{clk} = 1/ t = 1 / 6{,}25$ мкс = 160 кГц.

Рассмотрим структуру проекта в ПЛИС платы DEO-Nano для ШИМ-управления вентилятором, в основе которого DC-двигатель. Плата DEO-Nano имеет в своем составе генератор на 50 МГц, который тактирует цифровые устройства. Выполненный выше расчет показал, что для ШИМ-управления с частотой 10 кГц необходима частота 160 кГц (при градации регулировки, равной 16). Деление частоты выполним с помощью ранее разработанного параметрического делителя clk_divsv. Будем управлять скважностью сигнала pwma (рис. 81), поступающего на плату RPi Motor Driver, с помощью кнопок KEY0 и KEY1. Если сигнал up/down (KEY1), поступающий на счетчик, равен “1”, то формирование импульса en (KEY0) приведет к увеличению длительности импульса сигнала pwma и уменьшению скважности. Смена значения сигнала up/down на противоположное приведет к возможности уменьшения длительности импульса сигнала pwma и увеличению скважности.

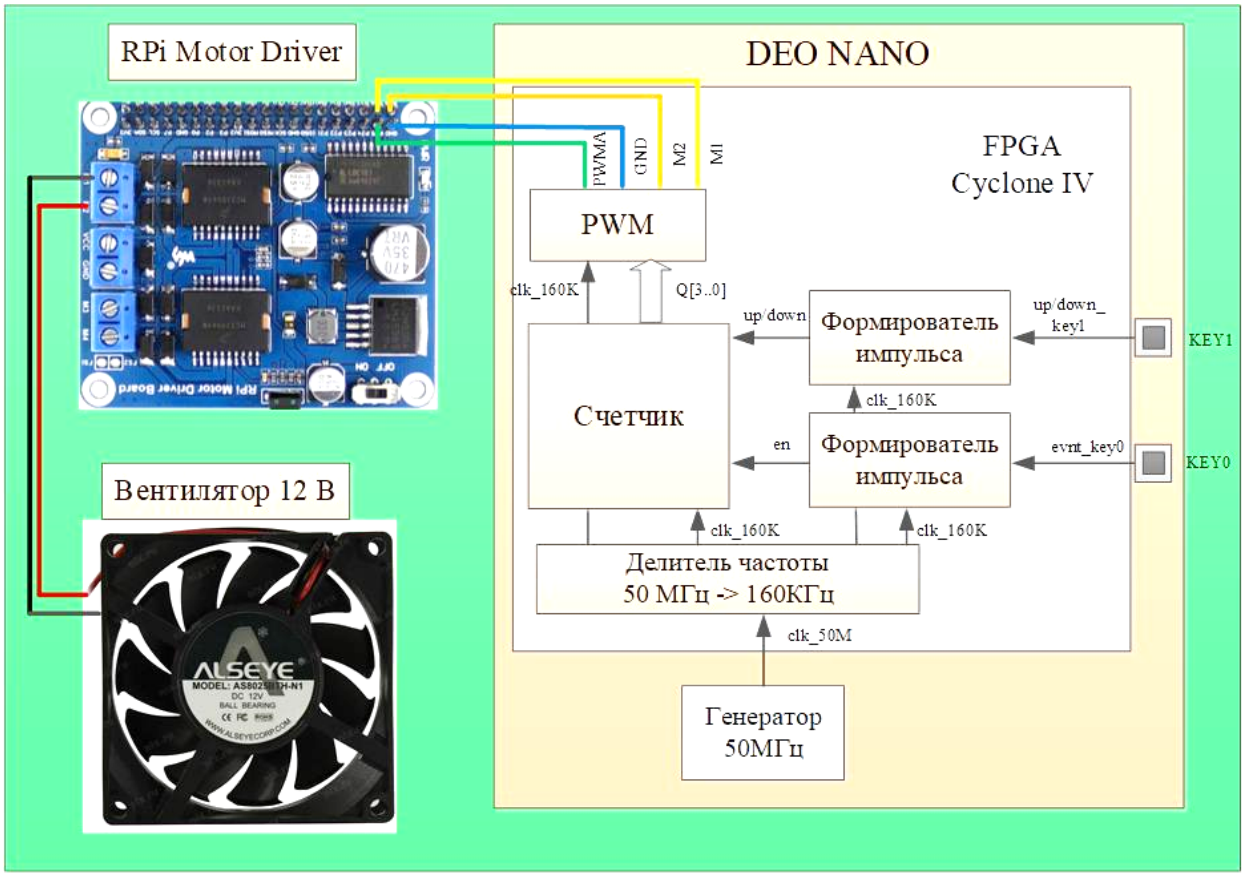



Рис. 81. Структура проекта по ШИМ-управлению DC-двигателем на базе стенда CLE-148

С целью устранения дребезга кнопок каждый из сигналов, evnt_ key0 и up/down_key1, необходимо пропустить через формирователь, проект которого рассмотрен в § 4.1.2. Кроме того, сигналы evnt_key0 и up/down_key1 являются асинхронными по отношению к работе остальных устройств проекта.

Создадим проект с именем **pwm_cle148** в пакете Quartus Prime и рассмотрим более подробно ШИМ-управление вентилятором. При разработке проекта будем придерживаться методики, рассмотренной выше: верхний уровень проекта pwm_cle148.bdf создадим в Block Diagram/Schematic File, а функциональные устройства (делитель частоты, формирователи и т.д.) напишем на языке SystemVerilog.

На рис. 82 представлена структура проекта верхнего уровня pwm_cle148.bdf. Схема содержит три входных порта: clk_50M – тактовый сигнал от генератора 50 МГц платы DEO-Nano, evnt_key0 и up_down_key1 – сигналы, связанные с двумя кнопками, о назначении которых говорилось выше. Выходные порты проекта можно условно поделить на «рабочие» и «демонстрационные». К «рабочим» необходимо отнести следующие порты: PWMA – основной модулируемый сигнал, поступающий на двигатель, M1 и M2 – сигналы управления для платы RPi Motor Driver, комбинация которых задает направление вращения двигателя. «Рабочих» портов достаточно для ШИМ-управления частоты вращения двигателя. Для временной и функциональной симуляции созданы «демонстрационные» выходные порты: q [3...0] – вектор уставки, задающий длительность импульса, clk_160K – частота 160 кГц для синхронизации устройств проекта (расчет частоты представлен выше), en – управляющий сигнал модуля counter, привязанный к домену тактовой частоты проекта, up_down – статический управляющий сигнал модуля counter, который определяет направление изменения значения счетчика.

В проекте использованы модули цифровых устройств, которые были рассмотрены ранее: параметрический делитель частоты clk_divsv и формирователи импульсов form_pulse. Их назначение описано выше. Как было рассмотрено ранее (рис. 80), при ШИМ-управлении для задания длительности импульса необходимо формировать уставку с помощью «регистра-счетчика». В проекте для этих целей используется модуль counter. На рис. 83 дано описание счетчика на SystemVerilog и временная симуляция. Рассмотрим подробнее программу counter.sv. Кроме реализации непосредственно счетчика в программе решается задача синхронизации входного сигнала en. Для этой цели добавлен сдвиговый регистр reg [1:0] sync, содержимое которого анализируется в первом блоке always_ff со списком чувствительности (posedge clk or posedge clr). Если при очередном нарастающем фронте переменной

clk старший разряд регистра sync [1] находится в состоянии “1”, то состояние регистра обнуляется. В противном случае (else) выполняется сдвиг (sync <= {sync [0], en}).

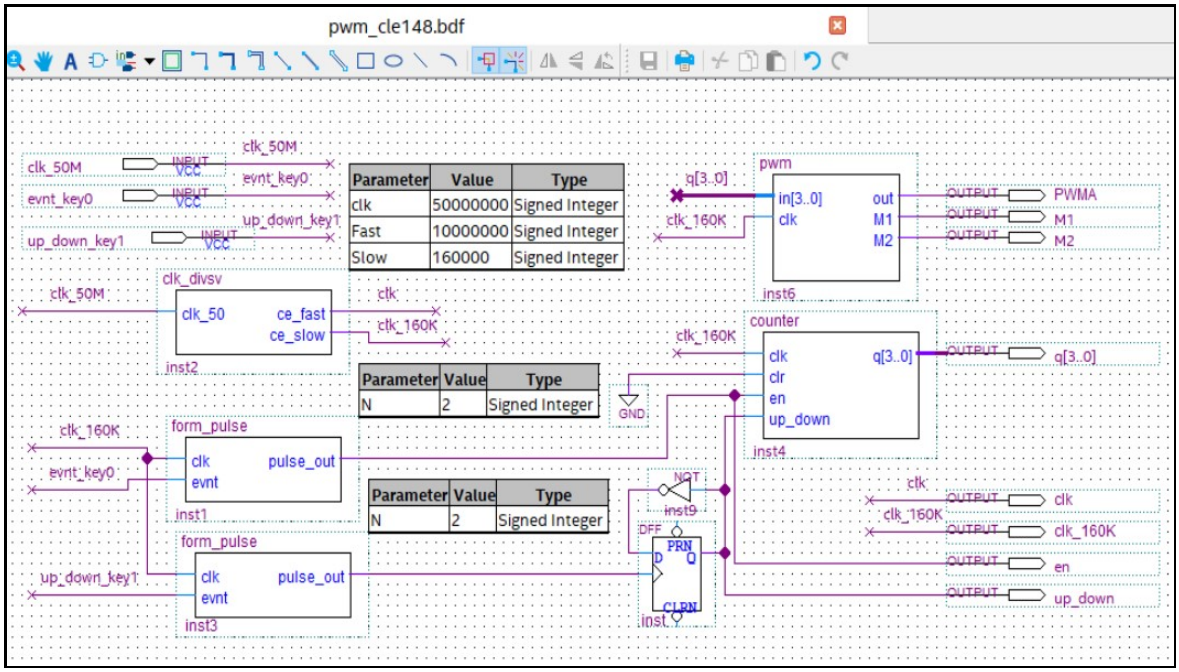



Рис. 82. Структура проекта pwm_cle148.bdf. для ШИМ-управления двигателем постоянного тока

В проекте использованы модули цифровых устройств, которые были рассмотрены ранее: параметрический делитель частоты clk_divsv и формирователи импульсов form_pulse. Их назначение описано выше.

При выполнении временной симуляции установлены следующие исходные данные: сигнал en приходит три раза с различной длительностью и в различных фазах по отношению к сигналу clk, сигнал up_down установлен в логическую “1” (значения счетчика будут увеличиваться с приходом en), в список сигналов симуляции добавлены внутренние выходы сдвигового регистра sync [1:0]. На временной диаграмме (рис. 83) работы счетчика видно, что после появления сигнала en по первому нарастающему фронту clk выполняется сдвиг (sync [0] переводится в состояние “1”), по второму выполняется еще один сдвиг (sync [1] переводится в состояние “1”). Затем сдвиговый регистр обнуляется, а значение счетчика увеличивается. Во втором блоке always_ff@ (posedge clk or posedge clr) анализируются сигналы sync [1] и up_down. При активном сигнале sync [1] значение счетчика либо увеличивается (up_down в состояние “1”), либо уменьшается (up_down в состояние “0”). В схему проекта pwm_cle148.bdf введен делитель на 2 на D-триггере (элемент DFF). Он позволяет при каждом нажатии кнопки key1 (для формирования импульса up_down_key1) получить статический сигнал up_down, который задает режимы работы счетчика.

counter.sv

```
1  module counter (input logic clk, clr,
2                  input logic en, up_down,
3                  output reg[3:0] q
4                  );
5  reg [1:0] sync;
6
7    always_ff@(posedge clk or posedge clr)
8      begin
9      if(clr)
10     sync <= 0;
11       else
12     if(sync [1])
13     sync <= 0;
14      else
15        sync <= { sync [0], en};
16
17     end
18
19   always_ff@(posedge clk or posedge clr)
20     if (clr)
21       q <= 0;
22     else
23
24        if (sync [1])
25
26            if (up_down)
27              q <= q + 1;
28            else
29              q <= q - 1;
30 endmodule
```

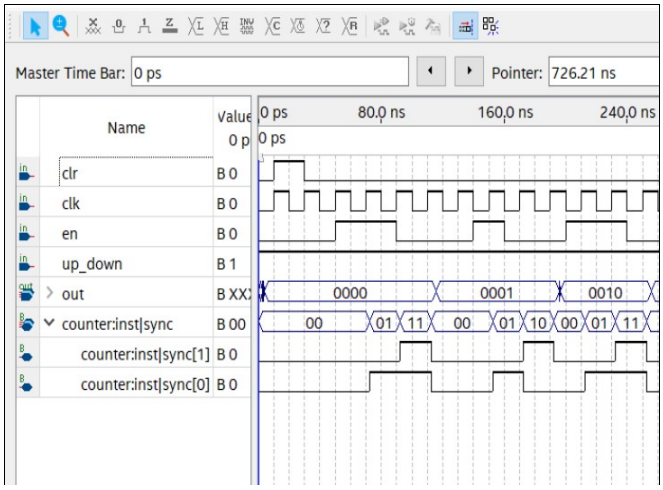



Рис. 83 Счетчик counter.sv с синхронизацией входного сигнала en (слева) и временная симуляция

Рассмотрим работу устройства, представленного модулем pwm. Это устройство содержит компаратор, который сравнивает значение уставки, формируемой счетчиком counter.sv и счетчиком в модуле pwm. Как было сказано ранее, для управления двигателями постоянного тока в составе стенда используется управляющая плата RPi Motor Driver. В нашем эксперименте используется вентилятор на базе коллекторного двигателя, который вращается в одну сторону. Поэтому, согласно табл. 6, необходимо задать вращение вентилятора в прямом направлении, присвоив переменным M1 и M2 следующие значения: M1 = 1’b1, M2 = 1’b0 (рис. 84). Модуль pwm имеет входные порты: clk – тактовый сигнал частотой 160 кГц, in [3:0] – четырехразрядный вектор уставки длительности импульса. Выходные порты: out – сигнал ШИМ-управления (pwma в проекте, см. рис. 82), M1 и M2 – для задания направления вращения. Внутренний счетчик counter [3:0] модуля pwm имеет разрядность, равную разрядности уставки, приходящей на вход in [3:0]. Блок always_ff@ (posedge clk) содержит оператор if–else, в котором сравниваются уставка и текущее значение счетчика counter.

pwm.sv

```
1  module pwm(input logic [3:0] in,
2             input logic clk,
3             output logic out,
4             output logic M1,
5             output logic M2);
6
7      assign M1 = 1'b1;
8      assign M2 = 1'b0;
9
10     reg[3:0] counter;
11
12     always_ff@(posedge clk)
13 begin
14     if(counter<=in)
15      out <=1'b1;
16     else
17      out <=1'b0;
18     counter<=counter+4'b1;
19 end
20
21  endmodule
```

Рис. 84. Выходной модуль pwm.sv проекта pwm_cle148.bdf

Рассмотрим работу схемы проекта на основе функциональной симуляции, представленной на рис. 85. Входная частота 50 МГц делится до частоты 160 кГц с помощью модуля clk_divsv.


Рис. 85. Функциональная симуляция проекта по ШИМ-управлению двигателем постоянного тока

Для того чтобы увидеть изменение длительности импульса сигнала PWMA (при неизменном периоде этого сигнала) от минимального значения до максимального, зададим сигнал evnt_key0 в виде периодического сигнала. Сигнал up_down установим в состояние логической “1”, для того чтобы наблюдать непрерывное нарастание значение уставки, а следовательно, и длительности импульса сигнала PWMA до максимального значения. Для задания вращения вентилятора в прямом направлении переменным M1 и M2 присвоены следующие значения: M1 = 1’b1, M2 = 1’b0. Анализируя рис. 85, можно сделать следующие выводы:

- Для изменения скорости вращения двигателя необходимо изменять четырехразрядный вектор уставки q[3...0] с помощью сигнала evnt_key0. Число разрядов этого вектора определяет плавность изменения скорости (то есть ускорение).
- Направление изменения скорости вращения (уменьшение или увеличение) определяет сигнал up_down, который изменяется сигналом up_down_key1 (то есть кнопкой KEY1).
- Направление вращения двигателя задается комбинацией сигналов M1 и M2.

## 4.3. Разработка управления шаговым двигателем на базе учебного стенда CLE-148

Шаговый двигатель – это электромеханическое устройство, преобразующее сигнал управления в угловое (или линейное) перемещение ротора с фиксацией его в заданном положении без устройств

обратной связи. Одним из главных преимуществ шаговых двигателей является возможность осуществлять точное позиционирование и регулировку скорости без датчика обратной связи.

По виду обмоток шаговые двигатели разделяются на два типа: униполярные и биполярные шаговые двигатели. По конструкции их делят еще на три вида:

- с переменным магнитным сопротивлением (высокая точность, низкий крутящий момент, низкая цена);
- с постоянным магнитом (низкая точность, высокий крутящий момент, низкая цена);
- гибридный (высокая точность, высокий крутящий момент, высокая цена).

Статор и ротор ШД с переменным магнитным сопротивлением (реактивного ШД) имеют явно выраженные полюсы (зубцы). На зубцах статора размещаются обмотки возбуждения, питаемые от электронного коммутатора. Ротор выполнен из ферромагнитного материала и не имеет обмоток возбуждения (пассивный).

Отличительная особенность реактивного ШД заключается в неравенстве зубцов статора $z_c$ и ротора $z_\text{р}$ (обычно $z_\text{р} > z_c$). Вследствие такой конструкции при каждом переключении обмоток статора ротор совершает шаг α, равный разности полюсных делений статора $\tau_c$ и ротора $\tau_\text{р}$, а именно

$$\alpha = \tau_\text{с} - \tau_\text{р} = 360°/z_c - 360°/z_\text{р} = 360° \cdot (z_\text{р} - z_c)/(z_\text{р} \cdot z_c).$$

Уменьшив разность чисел зубцов $z_c$ и $z_\text{р}$, можно снизить шаг ротора. Практически эту разность выбирают четной, что улучшает использование ШД.

Реактивные ШД при простоте и технологичности конструкции, малых размерах шагов имеют существенный недостаток – незначительные мощность и синхронизирующий момент, что ограничивает их применение.

Этого недостатка лишены индукторные ШД, в которых для увеличения синхронизирующего момента ротор подмагничивается со стороны статора с помощью постоянных магнитов или дополнительной обмотки возбуждения.

В шаговых двигателях с постоянным магнитом, как следует из названия, есть постоянный магнит, который ориентируется в зависимости от полярности магнитного поля.

Гибридные двигатели сочетают в себе лучшие черты двигателей с переменным магнитным сопротивлением и двигателей с постоянными магнитами.

Биполярный двигатель имеет одну обмотку в каждой фазе, которая для изменения направления магнитного поля должна переполюсовываться драйвером (рис. 86*а*). Управляющие сигналы подаются на полюса A, B, C и D. Для такого типа двигателя требуется мостовой драйвер или полумостовой с двухполярным питанием. Всего биполярный двигатель имеет две обмотки и, соответственно, четыре вывода. Униполярный двигатель также имеет одну обмотку в каждой фазе, но от середины обмотки сделан отвод в виде дополнительных полюсов AB и CD (рис. 86*б*). Это позволяет изменять направление магнитного поля, создаваемого обмоткой, простым переключением половинок обмотки. При этом существенно упрощается схема драйвера. Драйвер должен иметь только 4 простых ключа. Таким образом, в униполярном двигателе используется другой способ изменения направления магнитного поля. Униполярный двигатель с двумя обмотками и отводами тоже можно использовать в биполярном режиме, если отводы оставить неподключенными. У четырехобмоточного ШД каждая обмотка имеет отдельные выводы. При соответствующем соединении обмоток такой двигатель можно использовать как униполярный, так и биполярный.

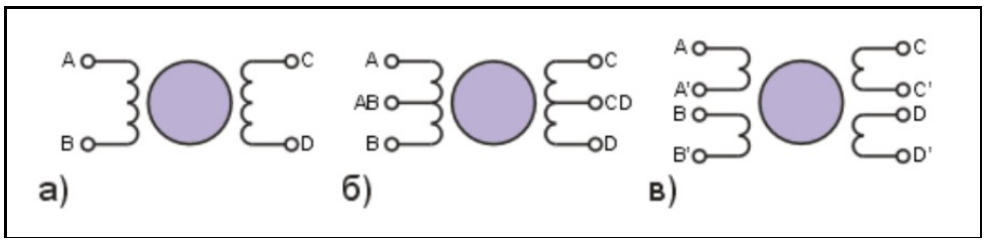



Рис. 86. Различные типы ШД: а) биполярный, б) униполярный, в) четырехобмоточный

Если сравнивать между собой биполярный и униполярный двигатели, то биполярный имеет более высокую удельную мощность. При одних и тех же размерах биполярные двигатели обеспечивают больший момент.

### 4.3.1. Способы управления фазами шагового двигателями

Решение задачи цифрового управления ШД требует формализации алгоритма с получением кодовых последовательностей. Существует три основных режима управления шаговым двигателем: полношаговый, полушаговый и микрошаговый. В работе [39] представлена графическая интерпретация управления ШД на основе разных режимов. Основным принципом работы шагового двигателя является создание вращающегося магнитного поля, которое заставляет ротор поворачиваться. Вращающееся магнитное поле создается статором, обмотки которого соответствующим образом запитываются (рис. 87).

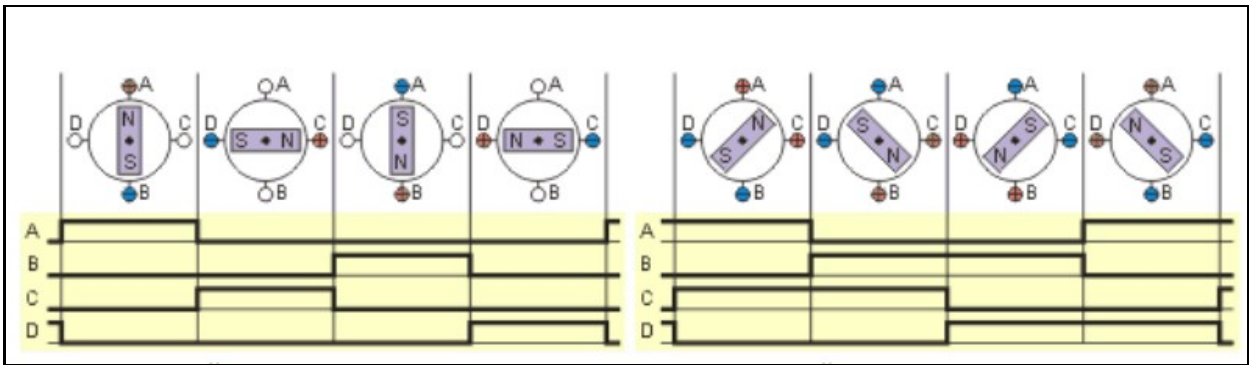



Рис. 87. Полношаговый режим управления: включена одна фаза (слева), включено две фазы [28]

В полношаговом режиме управления (full step) обеспечивается попеременная коммутация фаз, при этом они не перекрываются, и в один момент времени включена только одна фаза (рис. 87 слева) или две. На рисунке шаговый двигатель представлен в виде четырехполюсного статора и ротора в виде постоянного магнита, вращающегося по часовой стрелке. Ниже стилизованного изображения ШД в различных состояниях показаны значения напряжения (которые изменяются во времени), подаваемые на соответствующие полюса.

Точки равновесия ротора для каждого шага совпадают с «естественными» точками равновесия ротора у не запитанного двигателя. Недостатком этого способа управления является то, что для биполярного двигателя в один и тот же момент времени используется 50% обмоток, а для униполярного – только 25%. Это означает, что в таком режиме не может быть получен полный момент. При этом способе управления ротор фиксируется в промежуточных позициях между полюсами статора (см. рис. 87) и обеспечивается примерно на 40% больший момент, чем в случае одной включенной фазы.

Полушаговый режим управления ШД (half step) представляет собой комбинацию двух вариантов полношагового управления (рис. 88). Этот режим позволяет увеличить разрешающую способность двигателя, т.е. угловое перемещение ротора составляет половину угла шага по сравнению с полношаговым режимом. На рисунке дано графическое представление вращения внутреннего магнита (ротора) по часовой стрелке при подаче переменного напряжения на полюса. Из рисунка видно, что в тактах управления происходит чередование подключения полюсов к управляющему напряжению: в 1, 3, 5 и 7 тактах задействовано по одной обмотке (два полюса), а в 2, 4, 6 и 8 тактах – по две обмотки (четыре полюса).

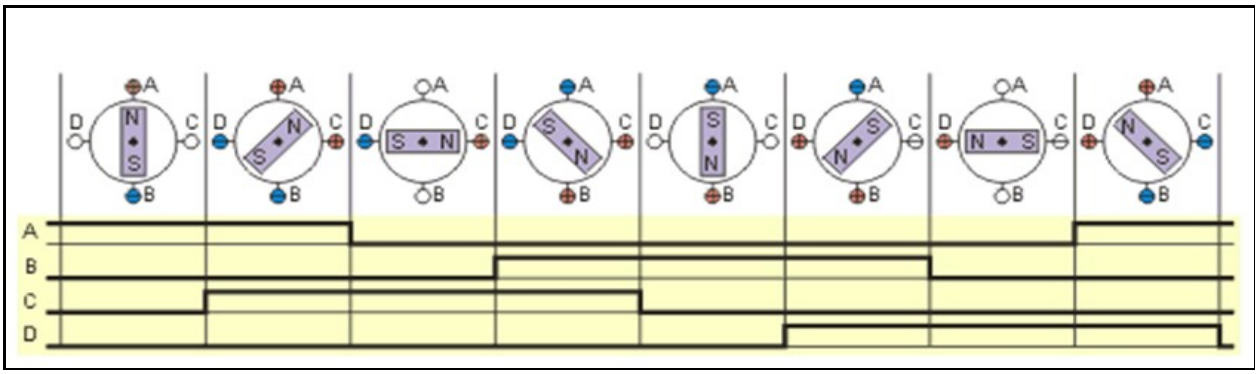



Рис. 88. Полушаговый режим управления ШД [28]

Недостатком полушагового режима является колебание момента от такта к такту. В тех положениях ротора, когда запитана одна фаза, момент составляет примерно 70% от полного, когда запитаны две фазы.

Для получения еще большего числа шагов двигателя применяют *дробление шага (микрошаговый режим)*, используя полушаговый режим, но токи обмоток распределяют неравномерно. Магнитное поле статора смещается между полюсов, смещается и положение ротора. Как правило, диспропорция токов между фазами происходит с определенной дискретностью, микрошагом. Современные системы управления обеспечивают перемещение ШД с дискретностью более 1000 микрошагов на оборот вала.

Важнейшим параметром ШД является *частота приемистости.* Максимальная частота управляющих импульсов, при которой возможен пуск ШД из неподвижного состояния без выпадания из синхронизма (пропуска шагов), называется *частотой приемистости* $f_{КПР}$. Чем больше механическая инерция ротора и момент нагрузки на валу, тем меньше $f_{КПР}$.

Преобразуем изменения управляющего напряжения в двоичный код. Поставим в соответствие полюсам A, B, C и D разряды четырехразрядного двоичного кода, где полюс A соответствует младшему разряду, а D – старшему. Высокий уровень напряжения в каждом такте будет соответствовать логической “1”, а низкий – логическому “0”. Запишем эти вектора в двоичном коде, результаты сведем в таблицу.

Таблица 12. **Соответствие уровней напряжения на полюсах статора двоичному коду**

| Номер такта | Уровень напряжения на полюсах | | | | Двоичный код | | | |
|---|---|---|---|---|---|---|---|---|
| | D | C | B | A | 4 | 3 | 2 | 1 |
| 1 | низкий | низкий | низкий | высокий | 0 | 0 | 0 | 1 |
| 2 | низкий | высокий | низкий | высокий | 0 | 1 | 0 | 1 |
| 3 | низкий | высокий | низкий | низкий | 0 | 1 | 0 | 0 |
| 4 | низкий | высокий | высокий | низкий | 0 | 1 | 1 | 0 |
| 5 | низкий | низкий | высокий | низкий | 0 | 0 | 1 | 0 |
| 6 | высокий | низкий | высокий | низкий | 1 | 0 | 1 | 0 |
| 7 | высокий | низкий | низкий | низкий | 1 | 0 | 0 | 0 |
| 8 | высокий | низкий | низкий | высокий | 1 | 0 | 0 | 1 |

### 4.3.2. Разработка проекта в пакете Quartus Prime для управления шаговым двигателем

В состав стенда CLE-148 включена семиканальная плата ключей (задействовано 4 канала) для управления от DEO-Nano униполярным шаговым двигателем 28BYJ-48. Описание характеристик платы и ШД представлены в § 2.3. На рис. 89 представлена структура стенда для управлением ШД.

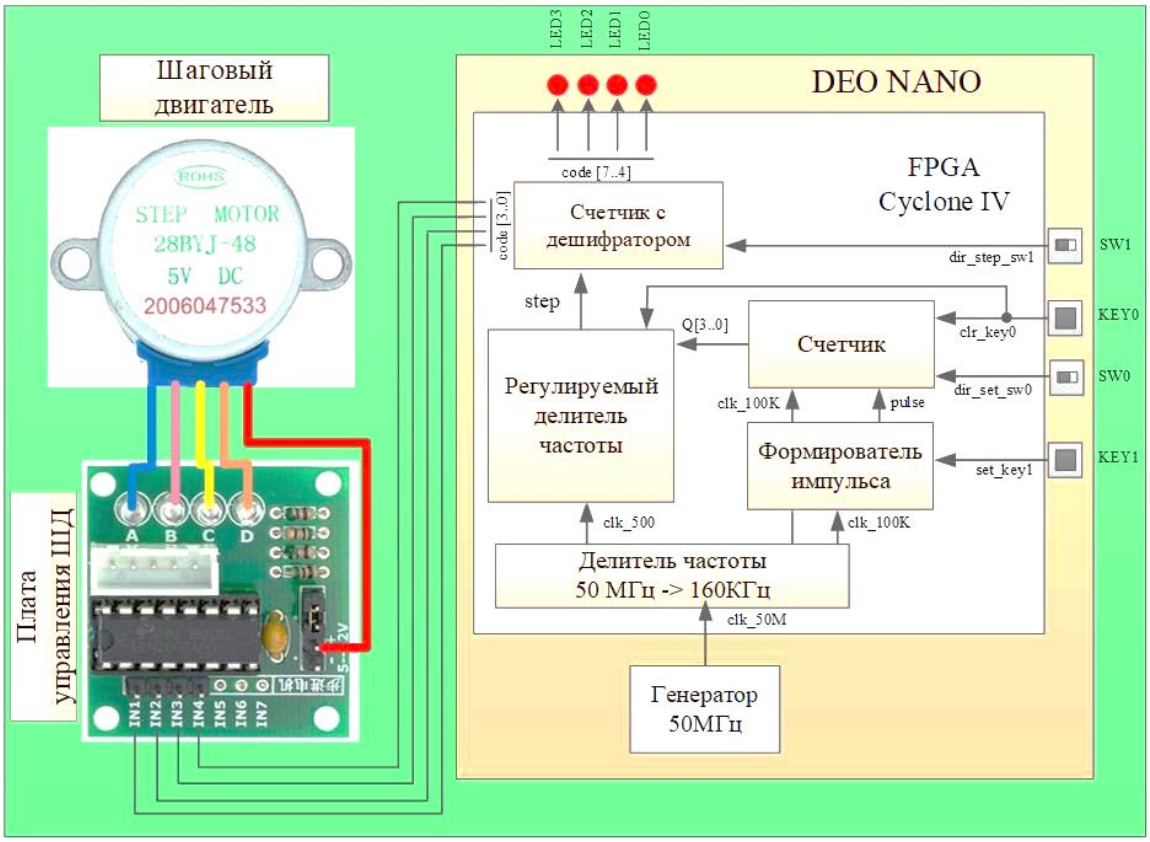



Рис. 89. Структура проекта по управлению шаговым двигателем на базе стенда CLE-148

Конфигурация стенда для управления ШД требует использования двух кнопок (KEY0 и KEY1) и двух движковых микропереключателей (SW0 и SW1). С помощью кнопки KEY0 формируется сигнал сброса clr_key0 для модулей «Счетчик» и «Регулируемый делитель частоты». Вторая функция сигнала clr_key0 – установить в «Счетчике» фиксированное (начальное) значение частоты вращения ШД, предварительно записанное как параметр. Кнопка KEY1 является источником сигнала set_key1, который, пройдя через «Формирователь импульса», позволяет изменять скорость вращения ШД. Направление изменения скорости (уменьшение или увеличение) задается с помощью микропереключателя SW0 (сигнал dir_set_sw0). Микропереключатель SW1 задает направление вращения вала ШД с помощью сигнала dir_step_sw1, поступающего на выходной модуль «Счетчик с дешифратором». Выходной вектор Q [3..0] модуля «Счетчик» определяет уставку для формирования частоты поступления сигналов step.

Поскольку ШД, используемый в стенде, является униполярным шаговым двигателем, то дополнительные полюса AB и CD (рис. 86*б*) необходимо соединить с выводом питания +5 V (правый провод на рис. 89). Сигналы управления на плату ШД поступают непосредственно от разъема DEO-Nano. Плата DEO-Nano имеет в своем составе генератор на 50 МГц, который тактирует цифровые устройства проекта. Характеристики ШД 28BYJ-48 ограничивают частоту поступления сигналов step величиной 500 Гц, а соответственно, и частоту смены управляющих кодов, представленных в табл. 12. Таким образом, для формирования сигналов step необходима частота 500 Гц, а для формирователя импульса и работы модуля «Счетчик» (см. рис. 89) можно ограничиться частотой 100 кГц. Как и в предыдущем проекте, деление частоты выполним с помощью разработанного параметрического делителя clk_divsv.

Создадим проект с именем **stepmotor_cle148** в пакете Quartus Prime и рассмотрим более подробно управление шаговым двигателем. При разработке проекта будем придерживаться методики, рассмотренной выше: верхний уровень проекта stepmotor_cle148.bdf создадим в Block Diagram/Schematic File, а функциональные устройства (делитель частоты, формирователи и т.д.) напишем на языке SystemVerilog.

На рис. 90 представлена структура проекта stepmotor_cle148.bdf. Схема содержит пять входных портов: clr_key0 – сигнал сброса и инициализации, clk_50M – тактовый сигнал от генератора 50 МГц платы DEO-Nano, set_key1 – установка скорости вращения вала ШД, dir_set_sw0 – задание направления изменения скорости и dir_step_sw1 – задание направления вращения вала ШД. Как и в предыдущем проекте, в stepmotor_cle148.bdf использованы выходные порты для непосредственного управления ШД («рабочие» выходные порты), так и выходы для организации функционального и временного симулирования. Как видно из структуры проекта (см. рис. 89), достаточно иметь 4 канала для управления ШД и 4 контрольных канала для индикации (LED3...LED0).

Рассмотрим модули проекта, структура которого представлена на рис. 90. Модули clk_divsv и form_pulse уже были использованы в предыдущем проекте. Модуль counter с синхронизацией входного сигнала отличается от использованного в проекте только дополнительным параметром: parameter setting = 4'b0100. Параметр setting необходим для задания начальной скорости вращения вала ШД при формировании сигнала сброса clr_key0. По умолчанию его значение равно parameter setting = 4'b0100.

Для создания возможности изменять скорость ШД с помощью внешней кнопки, т.е. изменять частоту сигнала step, был разработан модуль clk_div_stepmotor – регулируемый делитель частоты. На рис. 91 приведен текст программы модуля и временная симуляция его работы.

В программе clk_div_stepmotor.sv есть внутренний регистр reg [3:0] count, который подсчитывает тактовые сигналы clk_step. Процедурный оператор always_ff@ в списке чувствительности содержит сигналы clk_step и clr. Внутри always_ff@ находятся операторы, определяющие алгоритм работы программы. Поскольку эти операторы разные, то, как было описано ранее, эти операторы обрамляются конструкцией begin–end. Внутри этой конструкции используется цепочка операторов if-else-if-else, в каждом из которых используются такая же конструкция begin–end, но ее функция другая – обеспечить заданную последовательность выполняемых действий.

При появлении в списке чувствительности сигнала сброса clr внутренний регистр count и выходной сигнал clk_out обнуляются. Все остальные действия выполняются по переднему фронту сигнала clk_step. Если значение регистра count сравнялось со значением входного сигнала in, то выполняем обнуление count и инвертирование сигнала clk_out (clk_out <= ~ clk_out). В противном случае (else) значение count увеличиваем на единицу (count <= count +1’b1), а значение clk_out оставляем без изменений. Таким образом, частота выходного сигнала clk_out все время соответствует значению уставки in. Если значение 4-разрядного вектора in изменяется, то изменяется и частота сигнала clk_out. Это хорошо видно на временной симуляции рис. 91. При значении in [3..0] = 4’b0100 частота сигнала clk_out составляет порядка 5 МГц, а при in [3...0] = 4’b0111 меньше 3 МГц.

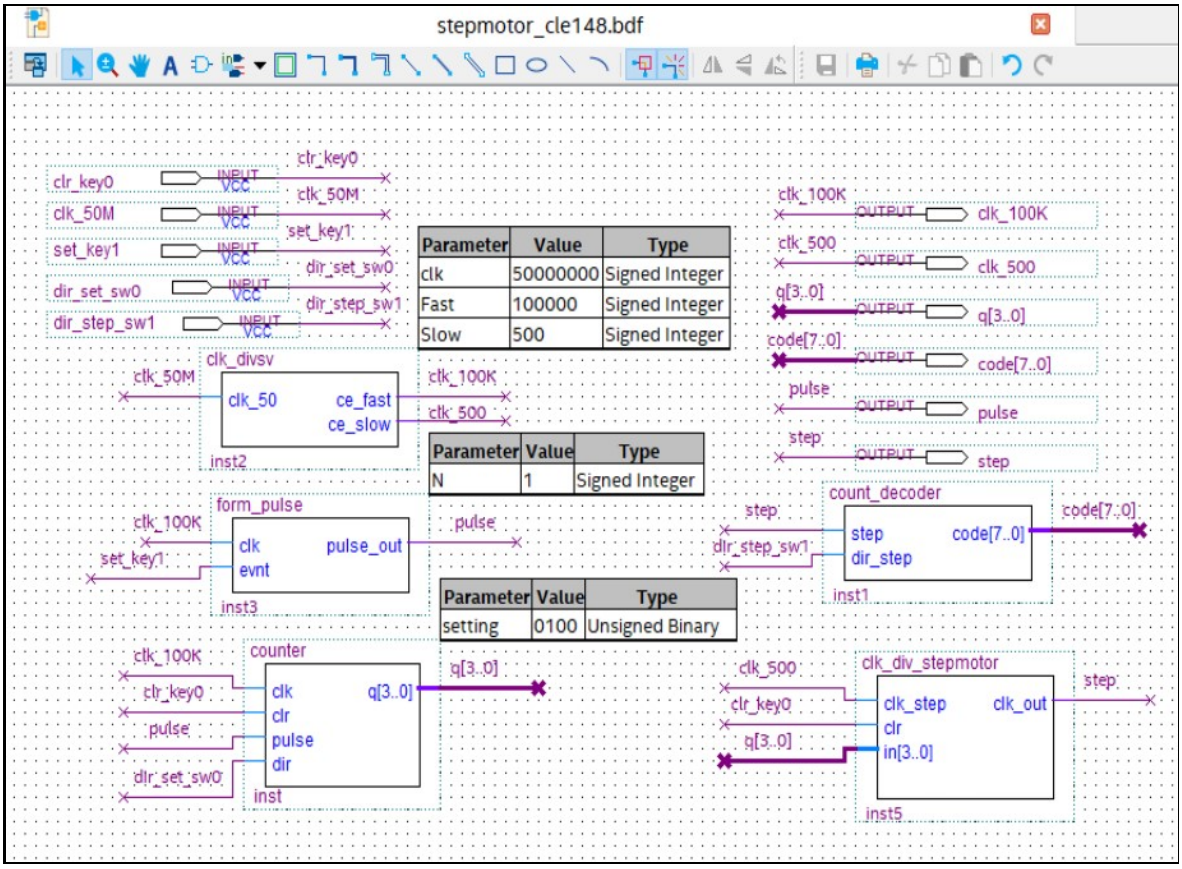



Рис. 90. Структура проекта stepmotor_cle148 для управления шаговым двигателем


Рис. 91. Программа регулируемого делителя частоты clk_div_stepmotor.sv и временная симуляция его работы

Выходной модуль count_decoder проекта совмещает в себе два устройства: счетчик и дешифратор. Модуль имеет два входных порта: для переменной step, синхронизирующей работу ШД с внешними процессами, и dir_step, задающей направление вращения вала двигателя. Выходной порт передает значение 8-разрядного вектора code [7...0].

В задачу этого модуля входит: преобразование поступающих сигналов step в 3-разрядный двоичный код с помощью счетчика и декодирование этого кода в управляющий код, рассчитанный в § 4.3.1 и представленный в табл. 12. Вектор code [7...0] модуля count_decoder, кроме 4-разрядного управляющего кода, содержит такой же код для индикации на светодиодах платы DEO-Nano.

В программе count_decoder.sv используется два процедурных оператора always (рис. 92). Как было сказано выше, сигналы, значения которым присвоены в этих операторах, сохраняют свое состояние до тех пор, пока не придет очередной сигнал из списка чувствительности. В текущей программе используется два различных типа always оператора. Для создания счетчика (последовательностная схема) используется оператор always_ff, а для дешифратора (комбинационная схема) – always_comb. В программе используется внутренняя переменная counter [2:0] – 3-разрядный регистр, который является аргументом для оператора case (counter). Оператор case проверяет значение counter; если значение

переменной counter равно 0, выполнится действие после двоеточия, т.е. значение выходной переменной code установится в 8’b00010001. Аналогично проверяются другие значения counter вплоть до 7. Условие default (по умолчанию) – удобный способ определить выход для всех случаев, не упомянутых явно, тем самым гарантируя в результате комбинационную логику. В SystemVerilog операторы case обязаны находиться внутри операторов always. Функциональная симуляция работы модуля дают возможность проанализировать взаимодействие сигналов step и dir_step и формирование выходных управляющих кодов code [7...0]. На рис. 92 показан реверс вращения вала ШД при смене сигнала dir_step с “1” на “0” (вертикальный курсор).

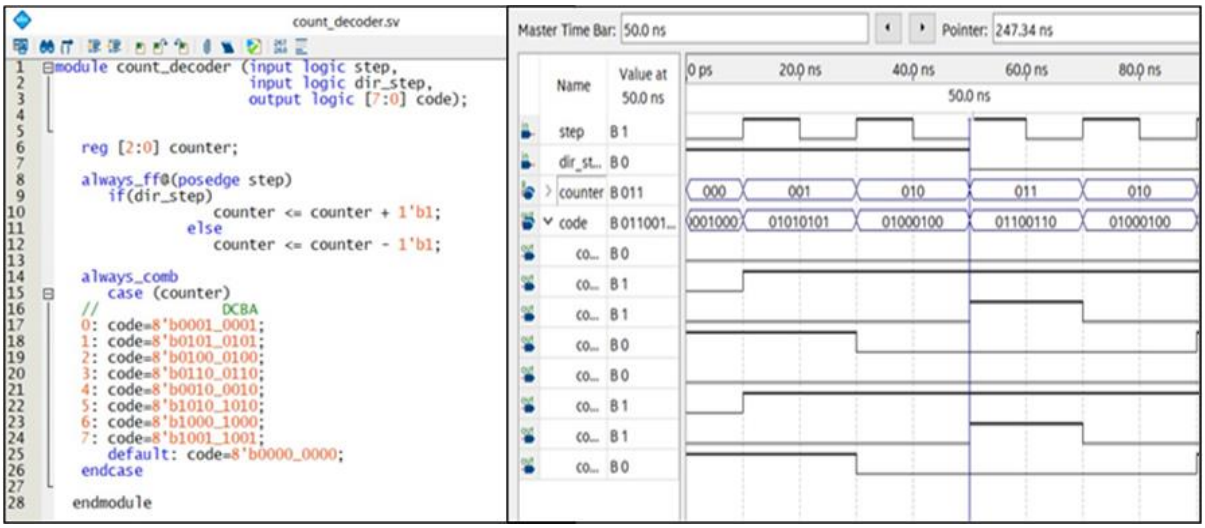



Рис. 92. Программа счетчика с дешифратором count_decoder.sv и функциональная симуляция работы

На рис. 93 показана временная симуляция всего проекта stepmotor_cle148, на которой представлены не только все входы и выходы проекта, но и внутренняя переменная pulse.

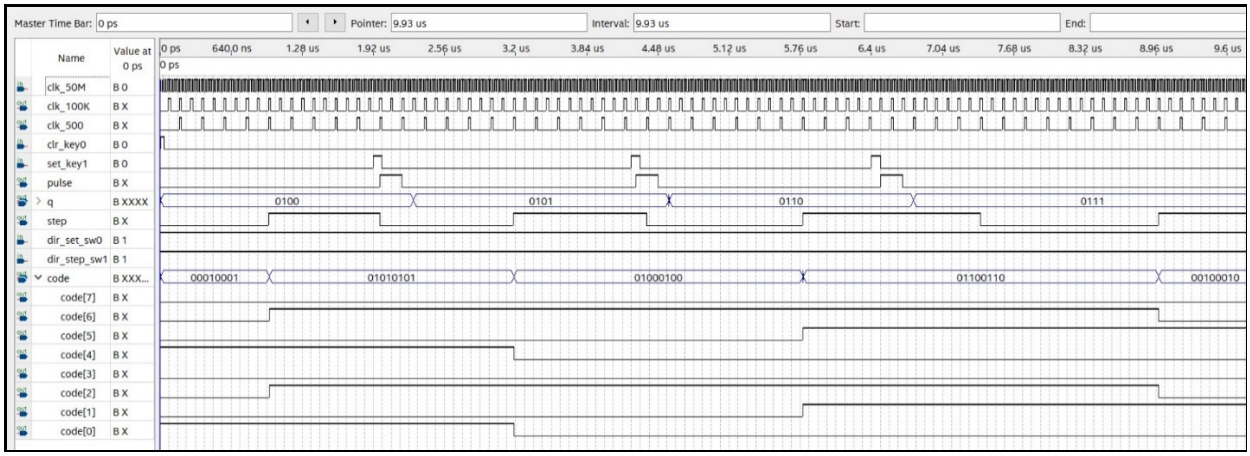



Рис. 93. Временная симуляция проекта stepmotor_cle148

Для того чтобы увидеть все сигналы на одном экране, были заданы следующие значения синтезированных частот: clk_100K = 10 МГц, а clk_500 = 5 МГц. На рис. 93 видно, что с каждым сформированным импульсом на входе set_key1 увеличивается значение уставки q[3..0] (от значения 0100 до 0111) и увеличивается период сигнала step, определяющего скорость вращения вала ШД. При формировании каждого сигнала step изменяется значение управляющего кода code [7...0] согласно кодировке дешифратора программы count_decoder.sv (см. рис. 92).

# Глава 5. Электронные системы сбора и обработки информации

**Магистрально-модульная система КАМАК.** Важнейшим компонентом при построении научных и промышленных автоматизированных комплексов являются электронные системы сбора и обработки информации. Первыми системами такого типа можно считать появление в 1972 г. европейского электронного стандарта CAMAC (Computer Automated Measurement and Control) [40; 41], определяющего организацию магистрально-модульной системы, предназначенной для сбора и обработки данных в физическом эксперименте. Следует заметить, что слово «контроллер» было использовано в данном стандарте впервые. CAMAC (ГОСТ определяет эту систему как КАМАК [42]) представляет собой систему, предназначенную для связи измерительных устройств с цифровой аппаратурой обработки данных. Система построена по модульному (блочному) принципу. Наименьшая конструктивная единица системы – функциональный модуль (или станция). Функциональные модули размещаются в каркасе-крейте (рис. 94).

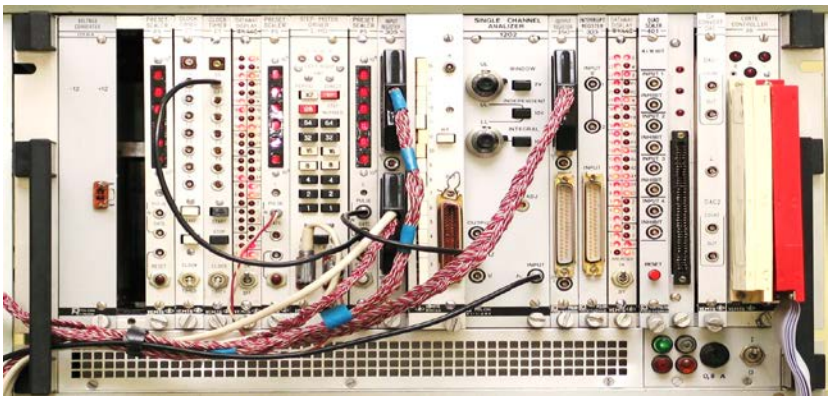


Рис. 94. Вид передней панели каркаса КАМАК

Структура измерительной электронной системы на основе аппаратуры КАМАК иллюстрируется рис. 95. Модули каркаса КАМАК управляются определенным набором команд. Текущее состояние аппаратуры КАМАК отображается соответствующими индикаторами на передней панели крейта. Программное управление и определение состояния аппаратуры осуществляется через регистр управления и состояния.

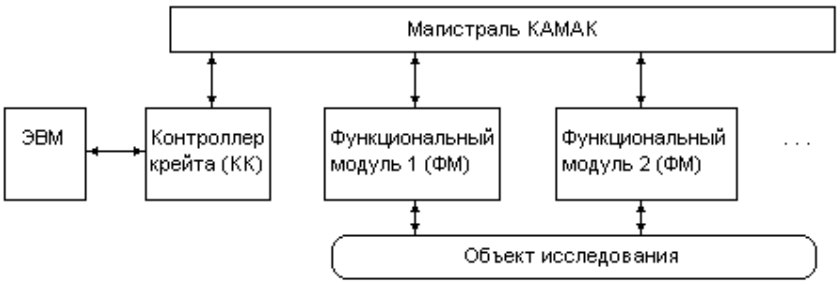



Рис. 95. Структура системы на основе аппаратуры КАМАК

Многокрейтовая система КАМАК состоит из одной или нескольких ветвей, каждая из которых имеет свою магистраль ветви (МВ), являющуюся средством связи между драйвером и крейт-контроллерами. Во время каждой операции на ветви драйвер может сообщаться максимум с семью крейт-контроллерами.

**Магистрально-модульная система VMEbus.** Название VMEbus произошло от VERSA module Eurocard и bus. Стандарт (спецификация VMEbus revison A) был разработан группой производителей магистрально-модульной аппаратуры в 1981 г., в число которых входили компании Motorola CG [43] и Signetics [44]. Представители этих фирм объединили свои усилия с целью создания документа, который позволил бы различным производителям разрабатывать изделия целиком или частями с полной совместимостью между собой. Создатели архитектуры VMEbus поставили перед собой задачу разработать новую шину, полностью независимую от типа используемого микропроцессора, с возможностью простого увеличения емкости информационных каналов от 16 до 32 бит, опирающуюся на надежный механический стандарт и позволяющую независимым поставщикам создавать совме-

стимые продукты. Шина была объявлена открытой, что стимулировало разработку оборудования, совместимого с новой стандартной архитектурой, третьими фирмами. Система VME состоит из интерфейсных логических схем, набора функциональных модулей, варьируемого в зависимости от конфигурации, и четырех групп сигнальных линий: передачи данных, арбитрации, прерываний и служебной шины.

VME включает в себя асинхронную параллельную шину передачи данных (рис. 96), содержащую функциональные модули: ведущего, ведомого, следящего монитора, таймера магистрали. Магистраль передачи данных позволяет процессорам передавать двоичные данные со скоростью, удовлетворяющей требованиям самых современных 32-разрядных процессоров. Структура магистрали обеспечивает совместное применение в одной системе 8-, 16- и 32-разрядных модулей, работающих на разных частотах.

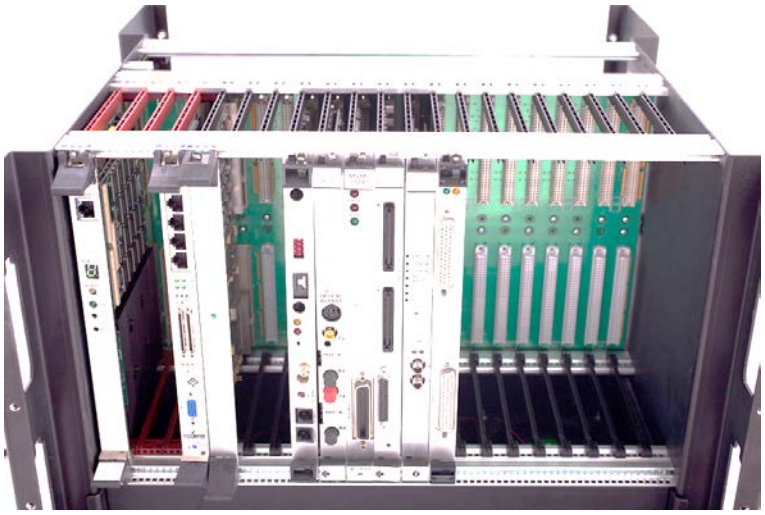


Рис. 96. Магистрально-модульная система VME-6U

*Ведущий* запускает цикл магистрали (последовательность уровней и фронтов на линиях магистрали, в результате которой происходит передача адреса и данных между ведущим и ведомым) для обмена данными с ведомым. Ведомый воспринимает цикл, начатый ведущим. При выполнении определенных условий отвечает на него, передавая или принимая данные.

*Следящий* монитор отслеживает передачу данных по магистрали. При обращении по присвоенным ему адресам сигнализирует размещенной на плате схеме. Например, сигналом может быть запрос прерывания процессору. В такой конфигурации процессор *A*, записывая данные в ячейку глобальной памяти, находящуюся под наблюдением монитора на плате процессора *B*, вызовет прерывание процессора *B*.

*Таймер магистрали* контролирует время ответа и заканчивает слишком длинные циклы специальным сигналом BERR. Так как VME использует асинхронный протокол, ведущий обязан ждать ответа ведомого для завершения цикла. Для предотвращения зависания (при попытках обращения к несуществующим ячейкам) таймер магистрали автоматически выставляет сигнал ошибки BERR по истечении определенного времени, определяемого характеристиками конкретной системы.

Расширением VMEbus является стандарт VXI (VMEbus eXtention for Instrumentation – расширение VMEbus для измерительной техники) – одно из направлений развития шины. Основываясь на шине VMEbus и полностью включая ее как подмножество, VXI представляет собой самостоятельный стандарт на контрольно-измерительную и управляющую аппаратуру. Спецификации VXIbus публиковались в 1987 (версия 1), 1998 (версия 2) и 2004 гг. (версия 3). VXI был принят в качестве стандарта IEEE как IEEE Std 1155-1992 [45].

**Магистрально-модульная система CompactPCI.** Семейство спецификаций CompactPCI® [46] определяет компактный (348×232 мм) модульный и масштабируемый подход к созданию систем, подходящих для чрезвычайно широкого спектра промышленных, коммерческих, аэрокосмических, военных, измерительных приборов, сбора данных, связи, телефонии, управления машиной и человеко-машинный интерфейс-приложений. CompactPCI, впервые представленная в 1999 г., продолжала развивать новые приложения.

Успех CompactPCI обусловлен в немалой степени его использованием параллельной шины PCI (Peripheral Component Interconnect, «взаимосвязь периферийных компонентов» – шина ввода-вывода для подключения периферийных устройств к материнской плате компьютера) в качестве основной ши-

ны данных. PCI была первой универсальной, независимой от процессора компьютерной шиной, которая была принята всеми основными производителями микропроцессоров. Сотни процессоров и тысячи периферийных чипов используют PCI. Благодаря использованию недорогого кремния и программного обеспечения, разработанного для PCI, CompactPCI стала самой популярной в мире модульной архитектурой открытого компьютера, предназначенной для встроенных приложений (рис. 97).

Стандарт имеет следующие отличительные черты:

- унифицированные размеры Eurocard в соответствии с IEEE 1101.1;
- компактные соединители с шагом 2 мм;
- вертикальное расположение плат (речь идет об охлаждении);
- высокая устойчивость к ударным и вибрационным воздействиям;
- металлическая передняя панель съемных плат (карт);
- соединители, предназначенные для пользователя, как на передней, так и на задней панели плат;
- применение стандартных шасси от разных изготовителей;
- каскадное исполнение выводов питания для режима горячей замены;
- поддержка восьми слотов в базовой конфигурации (с расширением при использовании мостов).

Рис. 97. Пример модулей магистрально-модульной системы CompactPCI

В 2011 г. была принята базовая спецификация CompactPCI Serial, тем самым было пополнено семейство популярных спецификаций для построения встраиваемых компьютерных систем CompactPCI. CompactPCI Serial четко прописывает назначение контактов четырех типов интерконнектов и одну контрольную шину I2C на разъемах системных контроллеров и разъемах периферии. На физическом уровне каждый канал состоит из двух дифференциальных пар, обеспечивающих передачу данных от источника к приемнику и обратно [47]. Стандарт CompactPCI Serial позволяет упростить процесс создания промышленной системы. Ее разработчик имеет выбор из четырех типов интерконнектов, что позволяет существенно сократить количество мостов для связи периферии с системным контроллером.

Разработчики систем реального времени и разработчики прикладных высокопроизводительных комплексов, предназначенных, например, для обработки преобразований Фурье, используют CompactPCI Serial как спецификацию, предоставляющую интерконнект обмена данными с низкими задержками и высокими скоростями межмодульного обмена, позволяющими создавать кластеры, объединяющие ресурсы нескольких модулей.

**Контрольные вопросы**

1. Какие значимые достижения предшествовали появлению цифровых технологий?
2. Развитию каких направлений способствовало появление персональных компьютеров?
3. Что способствовало резкому увеличению обмена информацией?
4. Какая магистрально-модульная система появилась первой, как она называлась и для каких целей предназначалась?
5. Объясните понятие «каркасный контроллер».
6. Какие задачи ставили разработчики системы VMEbus?
7. Какова особенность структуры магистрали VMEbus?
8. Что определяет семейство спецификаций магистрально-модульного стандарта CompactPCI®?

## Библиографический список


1. International Standard IEC 61804-2. Function blocks (FB) for process control. Part 2. Second edition, IEC, Geneva, 2006. 58 p.
2. International Standard IEC 61499. Function blocks for industrial-process measurement and control systems. Part 1: Architecture / International Electrotechnical Commission. Geneva, 2005. 111 p.
3. Burks A. W., Goldstine H. H., Neumann J. Preliminary Discussion of the Logical Design of an Electronic Computing Instrument. Institute for Advanced Study, Princeton, N. J., July 1946.
4. Dennis J. Data flow ideas for supercomputers. IEEE Society, 28th international conference, San Francisco, 1984. P. 15–19.
5. FPGA Cyclone IV. – Электрон. дан. – URL: https://www.intel.ru/content/dam/www/programmable/us/en/pdfs/literature/hb/cyclone-iv/cyclone4-handbook.pdf.
6. IEEE Std. 1149.1. Standard Test Access Port and Boundary-Scan Architecture. – URL: http://grouper.ieee.org/groups/1149/1/.
7. Коковин В.А., Комаров В.В. Контроллер таймерной сети общей таймерной системы ускорительного комплекса ИФВЭ // «НАУЧТЕХЛИТИЗДАТ», Приборы и Системы. Управление, Контроль, Диагностика. 2005. № 6. С. 15–20.
8. Коковин В.А., Комаров В.В. Мониторинг и диагностика общей таймерной системы // «НАУЧТЕХЛИТИЗДАТ», Приборы и Системы. Управление, Контроль, Диагностика. 2005. № 9. С. 46–50.
9. Intel® Quartus® Prime Software Suite. – Электрон. дан. – URL: https://www.intel.ru/content/www/ru/ru/software/programmable/quartus-prime/overview.html.
10. Уэйкерли Дж. Ф. Проектирование цифровых устройств. Москва: Постмаркет, 2002. 544 с.
11. Учебный стенд CLE-148. Паспорт и техническое описание.
12. Плата развития DEO-Nano. – Электрон. дан. – URL: https://www.terasic.com.tw/cgi-bin/page/archive.pl?No=593.
13. RPi Motor Driver Board. – Электрон. дан. – URL: https://www.waveshare.com/wiki/RPi_Motor_Driver_Board.
14. Модуль расширения ДМ-1. Паспорт и техническое описание.
15. MC33886. – Электрон. дан. – URL: https://www.nxp.com/docs/en/data-sheet/MC33886.pdf.
16. Vivado Design Suite Evaluation and WebPACK. – Электрон. дан. – URL: https://www.xilinx.com/products/design-tools/vivado/vivado-webpack.html.
17. IEEE Standard Verilog® Hardware Description Language. – Электрон. дан. – URL: https://inst.eecs.berkeley.edu/~cs150/fa06/Labs/verilog-ieee.pdf.
18. IEEE Standard for VHDL Register Transfer Level (RTL) Synthesis. – Электрон. дан. – URL: https://standards.ieee.org/standard/1076_6-2004.html.
19. 1800-2009-IEEE Standard for SystemVerilog-Unified Hardware Design, Specification, and Verification Language. – Электрон. дан. – URL: https://ieeexplore.ieee.org/document/5354441.
20. Все о языке SystemVerilog. – Электрон. дан. – URL: http://systemverilog.ru/.
21. Томас Д. Логическое проектирование и верификация систем на SystemVerilog / пер. с англ. А.А. Слинкина, А.С. Камкина, М.М. Чупилко. М.: ДМК Пресс, 2019. 384 с.: ил.
22. Харрис Дэвид М., Харрис Сара Л. Цифровая схемотехника и архитектура компьютера. – Электрон. дан. – URL: http://community.imgtec.com/downloads/digital-design-and-computer-architecture-russian-edition-second-edition.
23. Intel® FPGA Download Cable User Guide. – Электрон. дан. – URL: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug_usb_blstr.pdf.
24. Altera USB Blaster Driver Installation Instructions. – Электрон. дан. – URL: http://www.terasic.com.tw/wiki/Altera_USB_Blaster_Driver_Installation_Instructions.
25. Березин В. В., Золотухо Р. Н. Программирование последовательных конфигурационных ПЗУ фирмы Altera по JTAG-интерфейсу // Компоненты и технологии. 2005. № 2. С. 126–128.
26. AN 370: Using the Intel® FPGA Serial Flash Loader IP Core with the Intel® Quartus® Prime Software. – Электрон. дан. – URL: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/an/an370.pdf.
27. https://www.cadence.com/.
28. https://www.ieee.org/.
29. Каршенбойм И. Асинхронные частоты, пересечение клоковых доменов и синхронизация // Компоненты и технологии. 2009. № 2. С. 102–107.
30. Ковач Н. Использование PLL. – Электрон. дан. – URL: https://marsohod.org/11-blog/212-pll.
31. Clock Networks and PLLs in Cyclone IV Devices. – Электрон. дан. – URL: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/cyclone-iv/cyiv-51005.pdf.
32. ALTPLL (Phase-Locked Loop) IP Core User Guide. – Электрон. дан. – URL: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug_altpll.pdf.
33. Делитель частоты. – Электрон. дан. – URL: https://nabbla1.livejournal.com/204719.html.
34. Kokovin V.A., Sytin A.N. The processing of information from sensors in intelligent systems // Journal of Physics: Conference Series. 2017. Vol. 803, N. 1, 012075. doi:10.1088/1742-6596/803/1/012075.
35. Kokovin V., Uvaysov S., Uvaysova S. Lossless Compression Algorithm For Use In Telecommunication Systems, Control and Communications (SIBCON) // IEEE. 2016. DOI: 10.1109/SIBCON.2016.7491839.

36. Строгонов А.В. Проектирование цифровых фильтров в системе Matlab/Simulink и САПР ПЛИС Quartus // Компоненты и технологии. 2008. N 6. С. 32–36.
37. Рентюк В. Управление двигателем постоянного тока // Компоненты и технологии. 2014. № 10.
38. Евсиков А.А., Коковин В.А., Леонов А.П. Автоматизированный электропривод с частотным управлением: учебное пособие. Дубна: Гос. ун-т «Дубна», 2020. 121 с.
39. Ридико Л. Контроллер шагового двигателя. – Электрон. дан. – URL: http://kazus.ru/shemes/showpage/0/843/1.html.
40. CAMAC a Modular Instrumentation System for Data Handling. ESONE Committee, EUR 4100e, 1972.
41. Sytin A. Programmable Lam-Grader Type-258 // CERN CAMAC Note 70-80. 1978. May.
42. ГОСТ 27080-93 Система КАМАК. Модульная система технических средств для обработки данных.
43. https://www.motorola.com.
44. https://segnetics.com.
45. Introduction to VME / VXI / VXS Standards. – Электрон. дан. – URL: https://file.wiener-d.com/documentation/General/WIENER_VME_VXI_VXS_introduction_1.0.pdf.
46. CompactPCI Base Specification PICMG 2.0. – Электрон. дан. – URL: https://www.picmg.org/openstandards/compactpci/.
47. Открытые спецификации для построения модульных встраиваемых компьютерных систем // Информатизация и Системы Управления в Промышленности. 2012. № 5(41). – Электрон. дан. – URL: https://isup.ru/articles/15/3175/.

*Учебное издание*

**Евсиков** Александр Александрович,
**Коковин** Валерий Аркадьевич
**Леонов** Анатолий Петрович,
**Сытин** Александр Николаевич

# Проектирование цифровых устройств на базе современных инструментальных средств

УЧЕБНОЕ ПОСОБИЕ

Редактор Ю. С. Цепилова
Технический редактор Ю. С. Цепилова
Компьютерная верстка Ю. С. Цепилова
Корректор Ю. С. Цепилова

Подписано в печать 2021. Формат 60×90/8. Усл. печ. л. 10,75.
Тираж 40 экз. Заказ № .

ГБОУ ВО МО «Университет «Дубна»
141980, г. Дубна Московской обл., ул. Университетская, 19