МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ПРОЕКТИРОВАНИЕ ЦИФРОВЫХ УСТРОЙСТВ НА ПЛИС

Лабораторный практикум
по дисциплине «Цифровые устройства и микропроцессоры»
для студентов очной формы обучения
специальности 11.05.01 «Радиоэлектронные системы и комплексы»
и бакалавров направления 11.03.01 «Радиотехника»

Составитель М. Г. Царёв

Ульяновск
УлГТУ
2017

УДК 004.3+371.693 (076)
ББК 32.971+74.202 я 73
П 79

Рецензент директор УФ ИРЭ РАН доктор технических наук, доцент Сергеев В. А.

Рекомендовано научно-методической комиссией радиотехнического факультета в качестве лабораторного практикума.

П 79 **Проектирование цифровых устройств на ПЛИС** : лабораторный практикум / сост. М. Г. Царёв. – Ульяновск : УлГТУ, 2017. – 77 с.

Лабораторный практикум составлен в соответствии с программой курса «Цифровые устройства и микропроцессоры» и предназначен для исследования на базе учебного лабораторного стенда на основе ПЛИС логических элементов, узлов комбинационной и последовательностной логики, а также архитектуры микропроцессоров.

Адресован студентам дневной формы обучения направления 11.03.01 «Радиотехника» и специальности 11.05.01 «Радиоэлектронные системы и комплексы».

Подготовлен на кафедре «Радиотехника».

УДК 004.3+371.693 (076)
ББК 32.971+74.202 я 73

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Программируемая логическая интегральная схема (ПЛИС) – электронный компонент, используемый для создания цифровых интегральных схем. Логика работы ПЛИС не определяется при изготовлении, а задается разработчиков посредством программирования. Для программирования используются отладочные среды, позволяющие задать желаемую структуру цифрового устройства в виде принципиальной электрической схемы или программы на специальных языках VHDL, Verilog и др. Микросхемы программируемой логики являются одним из наиболее мощных и гибких инструментов для построения цифровых схем. Удобные и доступные системы проектирования позволяют разработчику создавать свои собственные приборы и устройства при минимальных затратах времени и средств.

Настоящий лабораторный практикум дает возможность студентам на лабораторном стенде изучить основные принципы работы и построения комбинационных и последовательностных цифровых устройств на основе ПЛИС.

В первых пяти лабораторных работах последовательно рассмотрено выполнение всех этапов проектирования цифровых устройств от ввода проекта в среде Altera Quartus II до программирования кристалла на лабораторном стенде и отладки проекта в симуляторе ModelSim с использованием файлов Testbench. Представленное описание ориентировано на использование системы автоматизированного проектирования (САПР) Altera Quartus II 13. Последняя лабораторная работа посвящена программированию RISC-процессора Nios II с использованием специального симулятора Nios2Sim.

Данный цикл лабораторных работ позволит студентам освоить САПР Altera Quartus II и получить навыки практической работы по проектированию цифровых устройств и систем, а также написанию программ для процессоров на основе ПЛИС в современной проектной среде.

ОПИСАНИЕ СТЕНДА ALTERA DE2-115

Выполнение первых пяти лабораторных работ рассчитано на использование учебного стенда Altera DE2-115, основой которого является ПЛИС семейства Cyclone IV EP4CE115F29C7. На рис. 1 и 2 показаны основные элементы учебного стенда изучаемой ПЛИС.

Отличительные особенности:

- 50 МГц тактовый генератор, 2 разъема SMA для входа/выхода тактовой частоты;
- 2 МБайт SRAM, 2x64 МБайт SDRAM, 8 МБайт Flash, разъем для SD карт;
- 4 кнопки,18 микропереключателей;
- 18 красных и 9 зеленых пользовательских светодиодов, ЖКИ 16x2;
- 24-разрядный аудио-кодек CD-качества с разъемами линейного входа/ выхода и микрофонного входа;
- VGA ЦАП (8-разрядный высокопроизводительный трехканальный ЦАП) с выходным VGA разъемом. TV декодер (NTSC/PAL/SECAM) и разъем TV входа;
- 2 Gigabit Ethernet PHY с разъемом RJ45;
- USB Host/ Slave контроллер с разъемами USB type A и type B;
- Приемопередатчик RS-232 с 9-контактным разъемом, разъем PS/2 мыши/ клавиатуры, ИК приемник для ДУ;
- 40-контактный разъем расширения с диодной защитой. Разъем для подключения мезонинных модулей (HSMC).

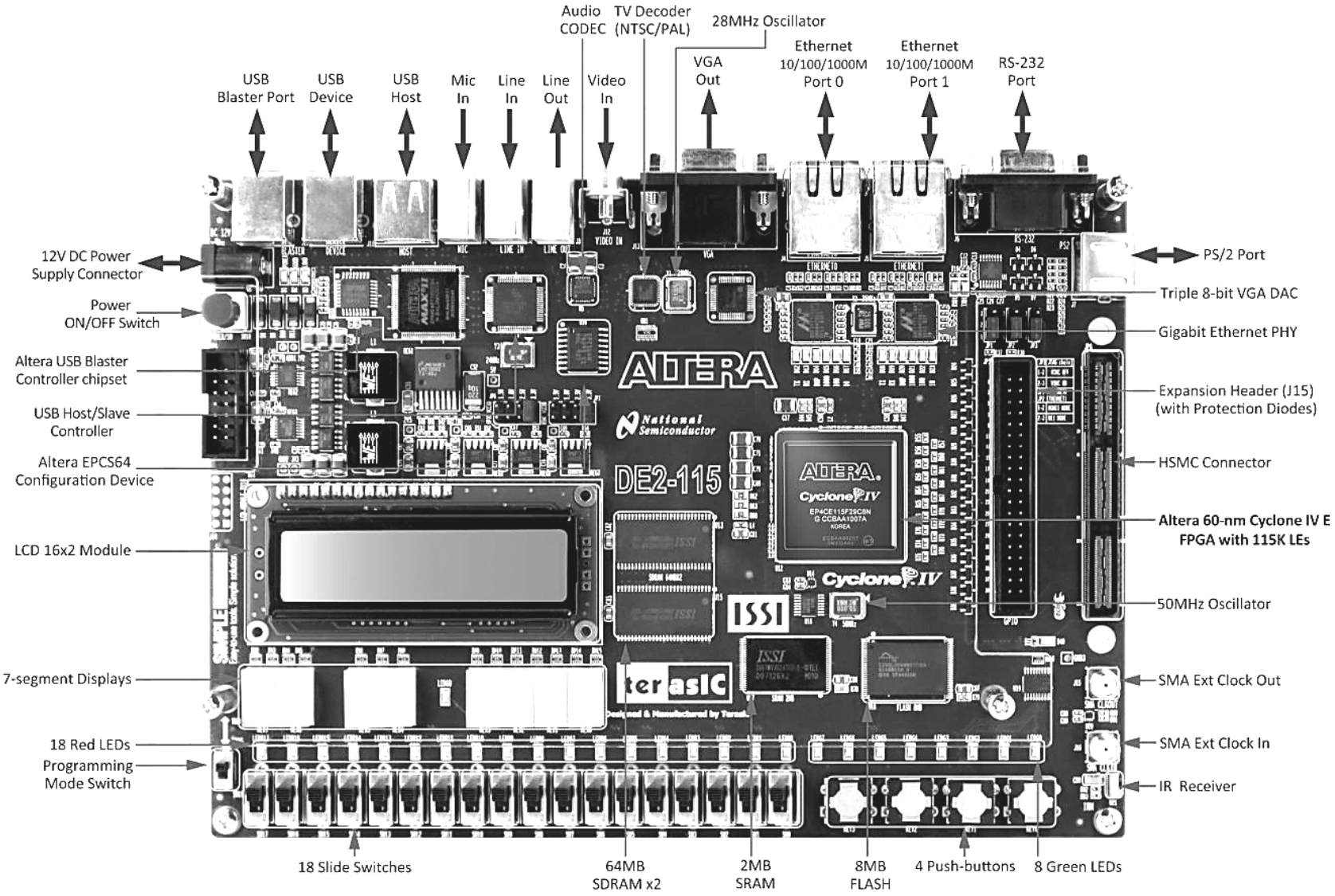


Рисунок 1 – Учебный стенд Altera DE2-115 (верхняя сторона)

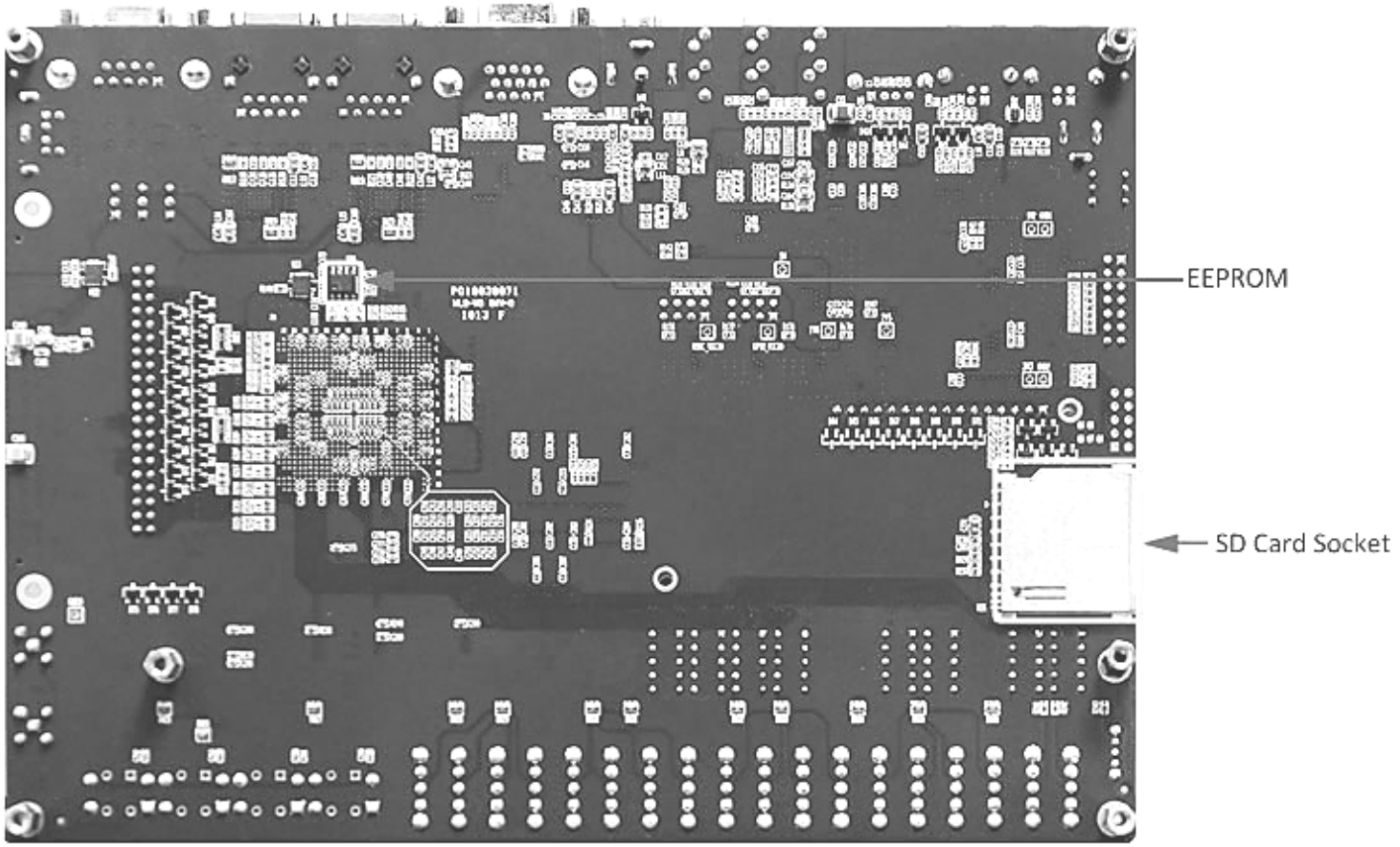


Рисунок 2 – Учебный стенд Altera DE2-115 (нижняя сторона)

ОПИСАНИЕ СИМУЛЯТОРА ПРОЦЕССОРА NIOS II

Выполнение последней лабораторной работы рассчитано на использование программы Nios2Sim в качестве симулятора программного RISC-процессора Nios II. Симулятор позволяет вводить исполняемый код на языке ассемблера, выполнять отладку и наблюдать за содержимым памяти и регистров процессора на каждом шаге выполнения программы.

При запуске симулятора открывается редактор для ввода исполняемой программы на языке ассемблера (рис. 3). Внизу окна отображается панель со служебными сообщениями.

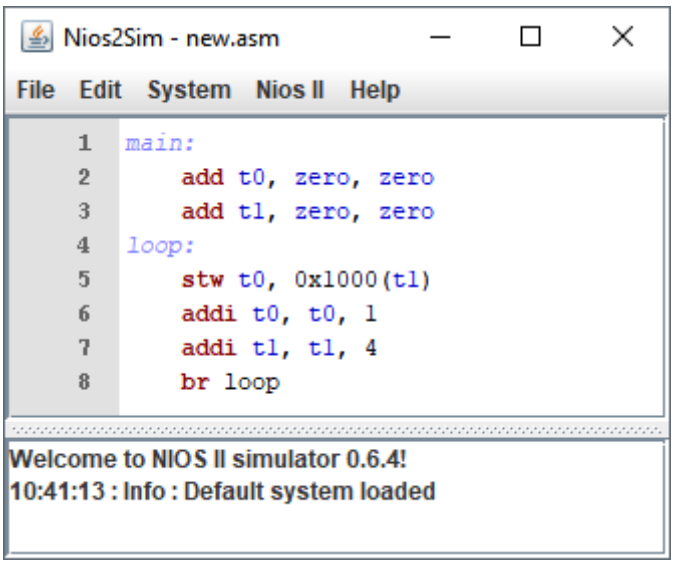


Рисунок 3 – Редактор исходного кода программы Nios2Sim

Меню File позволяет провести стандартные операции с файлами – Создать новый (File → New), загрузить существующий (File → Open…) или сохранить текущий (File → Save или File → Save As…) файл.

После ввода кода его можно запустить на исполнение в меню Nios II → Start Simulation. Окно программы изменит свой вид (рис. 4). Теперь в левой части окна представлены адреса и содержимое памяти ПЗУ процессора с хранящейся в ней программой и соответствующие ей ассемблерные инструкции. В правой части окна в различных вкладках отображены регистры общего назначения и специальные ре-

гистры процессора, память ПЗУ и ОЗУ, а также регистры периферийных устройств.

В данном режиме работы возможно проводить пошаговое выполнение программы (меню Nios II → Execute a Step) и наблюдать за состоянием памяти или регистров, либо запустить выполнение программы (меню Nios II → Run) на 500 шагов.

Текущая исполняемая инструкция помечена синим маркером, а изменяемые регистры помечаются красной заливкой.

Также симулятором поддерживается установка точек останова программы на любой инструкции. Для этого необходимо сделать двойной щелчок левой кнопки мыши в пустом поле слева от инструкции.

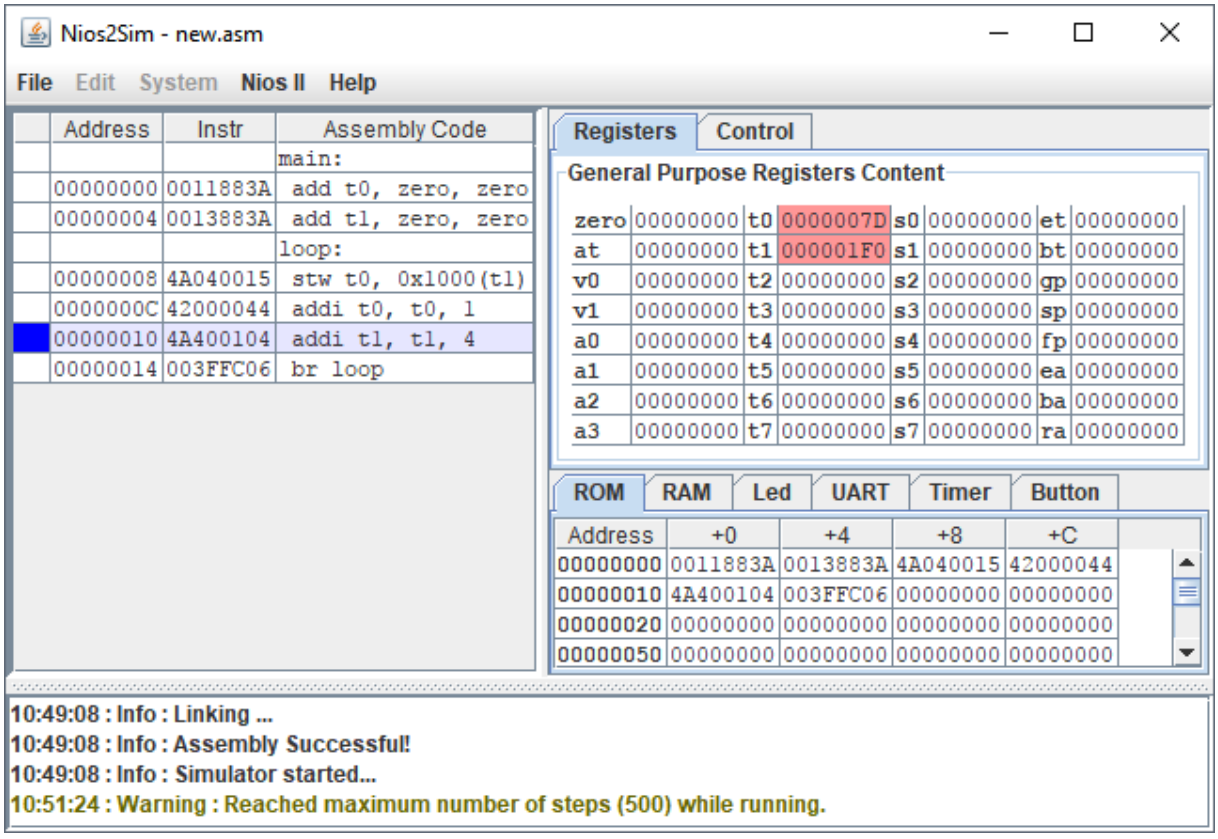


Рисунок 4 – Интерфейс симулятора при симуляции программы

Для перезапуска процесса симуляции следует выбрать в меню Nios II → Restart Simulation, а для завершения – Nios II → End Simulation.

ПРАВИЛА ВЫПОЛНЕНИЯ ЛАБОРАТОРНЫХ РАБОТ

Подготовка к работе

При подготовке к работе следует по конспектам лекций и рекомендованной литературе изучить теоретический материал, относящийся к данной работе, ознакомиться с описанием работы и продумать ответы на контрольные вопросы, составить краткий план выполнения лабораторной работы.

Выполнение работ в лаборатории

Лабораторные работы выполняются каждым студентом индивидуально либо небольшими бригадами по 2-3 человека и только в часы, предусмотренные расписанием. Выполнению работы предшествует проверка готовности студента к работе – студент должен ответить на все вопросы преподавателя по теории предстоящей работы и методике ее выполнения. Если результаты проверки готовности будут признаны удовлетворительными, студент получает допуск к работе.

При выполнении работы студент должен быть внимательным, проявлять самостоятельность и не допускать порчи оборудования. По всем неясным вопросам обращаться к преподавателю.

Работа в лаборатории считается законченной только после просмотра полученных результатов преподавателем и их утверждения. Отчет по лабораторной работе принимается преподавателем на последующих занятиях.

Порядок оформления отчета и зачет по работе

Отчет о выполненной лабораторной работе составляется каждым студентом индивидуально на компьютере (формат бумаги А4, шрифт Times New Roman, кегль 14, интервал полуторный) и должен содержать в себе материалы согласно заданию конкретной лабораторной работы. Пример оформления титульного листа отчета представлен в Приложении Г.

Зачет по работе студент получает только после предоставления и защиты отчета.

ЛАБОРАТОРНАЯ РАБОТА №1
СИНТЕЗ ЛОГИЧЕСКИХ СХЕМ.
ЗНАКОМСТВО С САПР ALTERA QUARTUS II

1. Цель работы

Целью работы является знакомство с САПР для ПЛИС Altera Quartus II, а также изучение способов синтеза логических схем и минимизации логических функций.

2. Краткие теоретические сведения

2.1. Способы синтеза логических схем

Любая логическая схема без памяти полностью описывается таблицей истинности. Эта таблица является исходной информацией для синтеза схемы на основе логических элементов «И», «ИЛИ», «НЕ». Для разработки требуемого цифрового устройства сначала на основе таблицы истинности записывают его логическое выражение. Затем с целью упрощения цифрового устройства минимизируют его логическое выражение и далее разрабатывают схему, реализующую полученное логическое выражение. Логические выражения можно получить двумя способами:

- на основе совершенной дизъюнктивной нормальной формы;
- на основе совершенной конъюнктивной нормальной формы.

Совершенная дизъюнктивная нормальная форма (СДНФ) представляется суммой групп. Каждая группа представляет собой ***минтерм*** – произведение, в которую входят все переменные.

Например,

$$F(A,B,C) = \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C}$$

Совершенная конъюнктивная нормальная форма (СКНФ) представляется произведением групп. Каждая группа представляет собой ***макстерм*** – сумму, в которую входят все переменные.

Например,

$$F(A,B,C) = (\overline{A} + B + C) \cdot (A + \overline{B} + C) \cdot (A + B + \overline{C})$$

Если схема имеет несколько выходов, то каждый выход описывается своей функцией. Такая система функций называется системой собственных функций.

Пример:

Таблица 1.1

Таблица истинности для функции трех переменных

A	B	C	Y	Минтерм	Макстерм
0	0	0	0	$\overline{A}\,\overline{B}\,\overline{C}$	$A+B+C$
0	0	1	1	$\overline{A}\,\overline{B}C$	$A+B+\overline{C}$
0	1	0	0	$\overline{A}B\overline{C}$	$A+\overline{B}+C$
0	1	1	1	$\overline{A}BC$	$A+\overline{B}+\overline{C}$
1	0	0	0	$A\overline{B}\,\overline{C}$	$\overline{A}+B+C$
1	0	1	0	$A\overline{B}C$	$\overline{A}+B+\overline{C}$
1	1	0	1	$AB\overline{C}$	$\overline{A}+\overline{B}+C$
1	1	1	1	ABC	$\overline{A}+\overline{B}+\overline{C}$

СДНФ составляется на основе таблицы истинности путем суммирования (логическое «ИЛИ») всех тех минтермов, для которых выход *Y* имеет значение 1.

$$Y = \overline{A}\cdot\overline{B}\cdot C + \overline{A}\cdot B\cdot C + A\cdot B\cdot\overline{C} + A\cdot B\cdot C$$

СКНФ составляется на основе таблицы истинности путем произведения (логическое «И») всех макстермов, для которых выход *Y* имеет значение 0.

$$Y = (A+B+C)\cdot(A+\overline{B}+C)\cdot(\overline{A}+B+C)\cdot(\overline{A}+B+\overline{C})$$

На основе полученных выражений можно составить схему устройства, реализующего заданную функцию. Схема устройства, полученная на основе СДНФ, изображена на рис. 1.1, а на основе СКНФ на рис. 1.2.

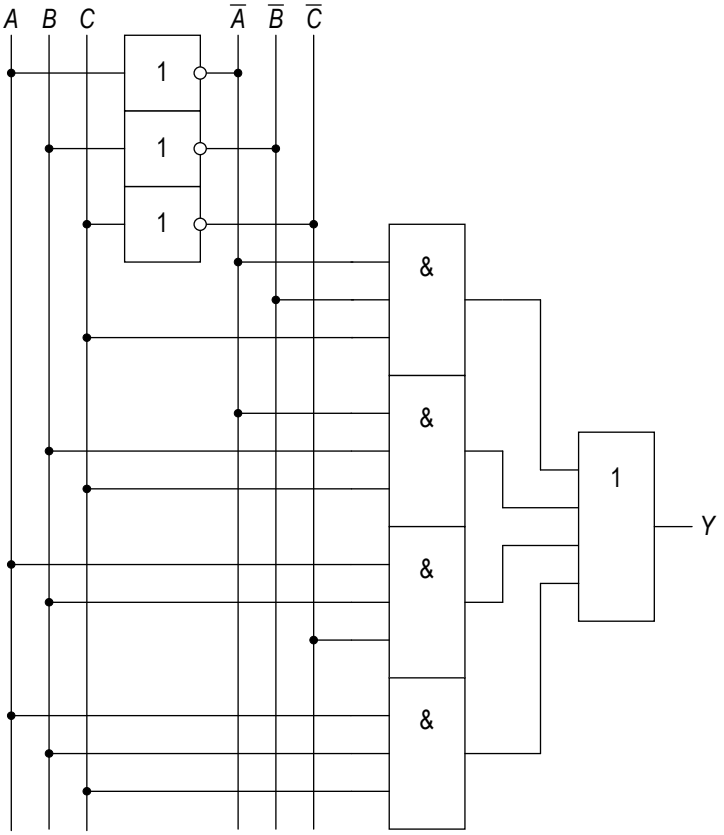


Рисунок 1.1 – Схема устройства, полученная на основе СДНФ

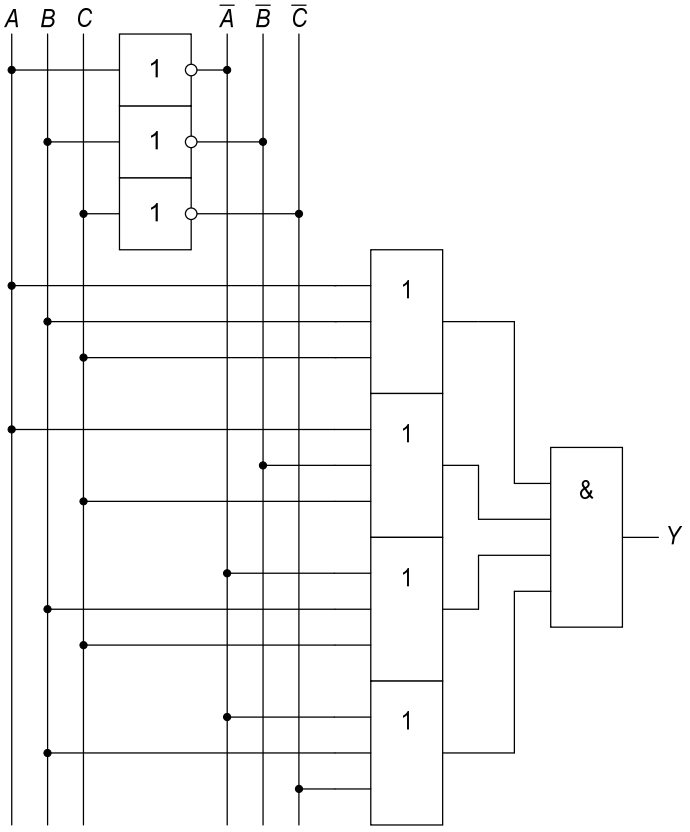


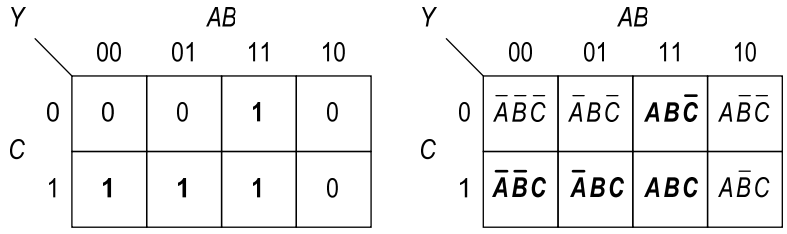
Рисунок 1.2 – Схема устройства, полученная на основе СКНФ

2.2. Карты Карно

С целью упрощения схемы цифрового устройства применяют минимизацию функций. ***Карты Карно*** представляют собой наглядный метод для упрощения булевых уравнений.

На рис. 1.3 показана карта Карно для функции, описанной в табл. 1.1. Верхняя строка карты дает 4 возможных значения для переменных *A* и *B*. Левая колонка дает 2 возможных значения переменной *C*. Каждая клетка карты Карно соответствует строке таблицы истинности и содержит значение функции *Y* из этой строки. Например, верхняя левая клетка соответствует первой строке таблицы истинности и показывает, что значение функции *Y* будет равно 0, когда *ABC* = 000. Как и каждая строка в таблице истинности, каждая клетка карты Карно представляет собой отдельный минтерм.

Каждая клетка (минтерм) отличается от соседней изменением только одной переменной. Это значит, что соседние клетки различаются только в значении одного литерала, значение которого «истинно» в одной клетке и «ложно» в соседней. Переменные *A* и *B* комбинируются в верхней строке в виде ***кода Грея***: 00, 01, 11, 10. В отличие от битового порядка по возрастанию величины (00, 01, 10, 11), в коде Грея соседние записи отличаются только на один разряд.



а б

Рисунок 1.3 – Функция трех переменных:
а – карта Карно, б – карта Карно с минтермами

Карты Карно также «закольцованы». Клетка с самого правого края таблицы является соседней с самой левой, так как они отличаются только в одной переменной (*A*). Можно свернуть карту в цилиндр, соединив края, и даже в этом случае соседние клетки также будут отличаться только в одной переменной.

На карте Карно на рис. 1.3 содержатся четыре единицы, что соответствует числу минтермов в уравнении. Чтение минтермов из карт Карно в точности соответствует чтению СДНФ из таблицы истинности.

Карты Карно помогают делать упрощение графически, обводя единицы в соседних клетках овалами, как показано на рис. 1.4. Для каждого овала пишут соответствующую ему импликанту. Переменные, для которых прямая и комплементарная формы попадают в один овал, исключаются.

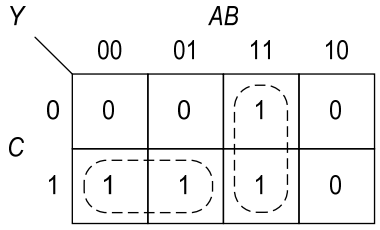


Рисунок 1.4 – Минимизация при помощи карты Карно

Таким образом, получаем функцию $Y = \overline{A}C + AB$, на основе которой составим новую схему устройства (рис. 1.5).

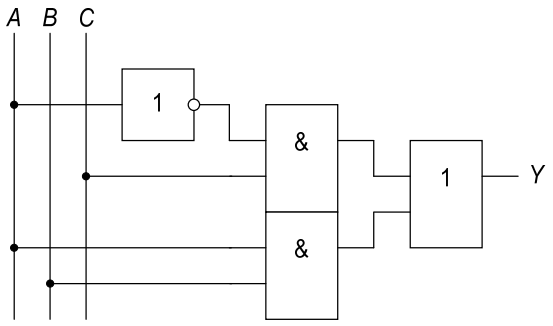


Рисунок 1.5 – Схема устройства, полученная после минимизации логической функции

Сравните с рис. 1.1 и 1.2, схема получилась гораздо компактнее.

Правила для нахождения минимального уравнения из карт Карно следующие:

- Использовать меньше всего овалов, необходимых для покрытия всех 1;
- Все клетки в каждом овале обязаны содержать 1 (или безразличные значения X);

- Каждый овал должен охватывать блок, число клеток которого в каждом направлении равно степени двойки (то есть 1, 2, 4, 8 и т. п.);
- Объединять клетки можно только по горизонтали или вертикали;
- Каждый овал должен настолько большим, насколько это возможно;
- Овал может связывать края карты Карно;
- Единица на карте Карно может быть обведена сколько угодно раз, если это позволяет уменьшить число овалов, которые будут использоваться.

3. Задание к работе

1. Синтезировать логическую схему на основе двухуровневой логики (СКНФ или СДНФ) по таблице истинности своего варианта (см. табл. 1.3).

2. Запустить среду Altera Quartus II и создать новый проект (меню File → New Project Wizard…). Мастер создания проекта содержит 5 шагов:

- Шаг 1 – Имя проекта и директория его сохранения (рис. 1.6).
  Внимание! Не допускать использования в именах файлов проекта и в пути их сохранения кириллические символы. Для сохранения каждого проекта использовать отдельную папку на диске.

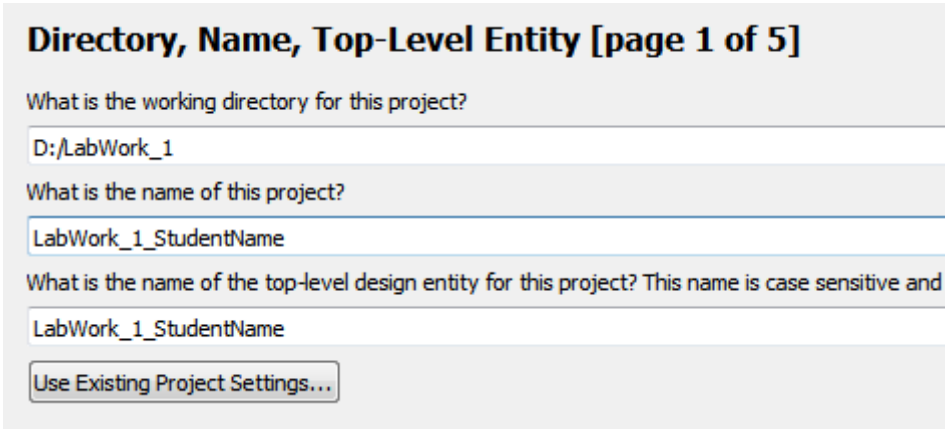


Рисунок 1.6 – Выбор директории и имени проекта при создании проекта

- Шаг 2 – Добавление в проект других необходимых файлов. В данной работе этот этап необходимо пропустить.
- Шаг 3 – Выбор семейства и наименования ПЛИС, под которую создается проект. Необходимо указать семейство Cyclone IV E, наименование EP4CE115F29C7. Чтобы ускорить процесс выбора типа ПЛИС, можно использовать фильтры по корпусу, числу выводов, классу скорости, а также по имени, введя символы, входящие в наименование (см. рис. 1.7).

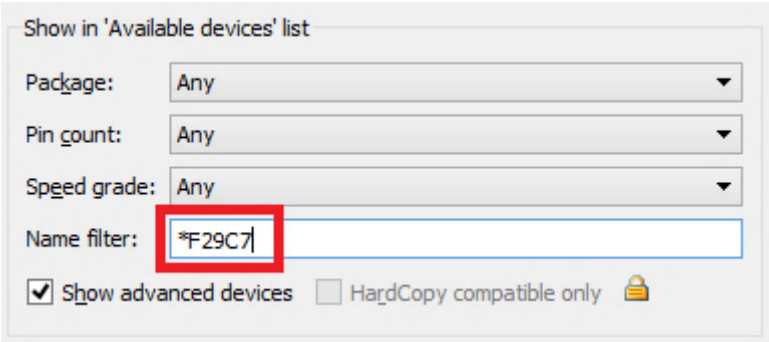


Рисунок 1.7 – Использование фильтра по имени при выборе типа ПЛИС

- Шаг 4 – Выбор инструментов для синтеза, отладки и симуляции. В данной работе этот этап необходимо пропустить.
- Шаг 5 – На последнем шаге отображается суммарная информация по проекту. Для завершения создания проекта нужно нажать кнопку «Finish».

3. Добавить в проект новый файл (меню File → New…) типа «Block Diagram/Schematic File» и начертить синтезированную схему, используя инструменты «Orthogonal Node Tool» (соединительные проводники), «Pin Tool» (входные и выходные порты), «Symbol Tool» (логические элементы). Altera Quartus II уже содержит в своей библиотеке стандартные символы простых логических элементов («and3», «or4», «not» и т. п.) (см. рис. 1.8).

4. Сохранить файл, не изменяя наименования файла, и выполнить анализ и синтез проекта (Processing → Start → Start Analysis & Synthesis). В случае отсутствия ошибок сделать назначение выводов (меню Assignments → Pin Planner) в поле Location нижней части окна Pin Planner в соответствии с первой и второй строками табл. 1.2.

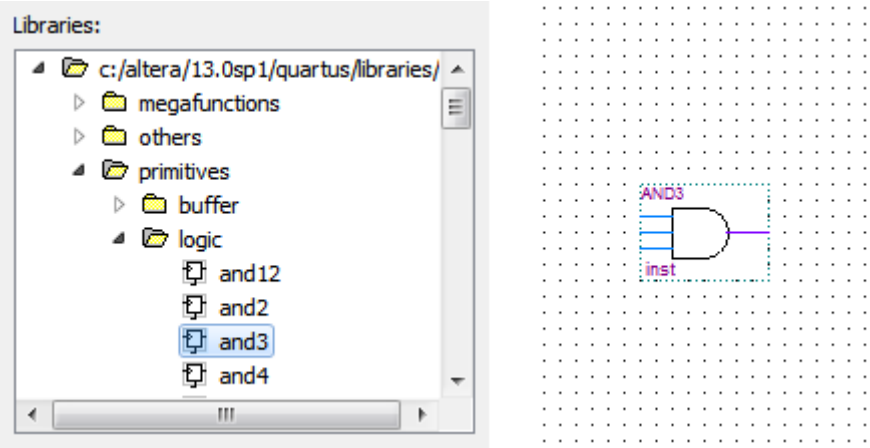


Рисунок 1.8 – Библиотека стандартных логических элементов в Altera Quartus II

5. Выполнить полную компиляцию проекта (Processing → Start Compilation). В процессе компиляции создается конфигурационный файл *.sof в папке с проектом в подпапке «output_files». Он используется при программировании ПЛИС.

Таблица 1.2

Назначение выводов схемы

Сигнал	**A**	**B**	**C**	**Y**
Вывод ПЛИС	AC27	AC28	AB28	G19
На стенде	SW2	SW1	SW0	LEDR0

6. В присутствии преподавателя подключить лабораторный стенд к компьютеру и включить на стенде питание.

7. Запрограммировать учебный стенд (меню Tools → Programmer) и, устанавливая на входах схемы с помощью переключателей SW2..SW0 все возможные кодовые комбинации из таблицы истинности и наблюдая за светодиодом LEDR0, проверить работу схемы.

8. Составить минимизированную логическую функцию по таблице истинности своего варианта (см. табл. 1.3) при помощи карты Карно. Синтезировать схему по минимизированной функции.

9. Начертить новую схему в Altera Quartus II в том же файле и выполнить анализ и синтез проекта. Затем сделать назначение выводов и полную компиляцию проекта.

10. Запрограммировать учебный стенд и проверить работу схемы на ее соответствие заданной таблице истинности.

11. Выключить питание лабораторного стенда и отключить его от компьютера.

12. Сделать выводы по работе.

4. Содержание отчета

1. Цель работы.
2. Заданная таблица истинности и синтезированная схема.
3. Карта Карно, синтезированная логическая функция и схема.
4. Выводы по работе.

5. Варианты заданий

Таблица 1.3

Варианты заданий

Вариант			**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
A	**B**	**C**	**Y**														
0	0	0	0	1	0	0	1	0	1	0	1	0	1	0	0	0	1
0	0	1	1	0	0	1	0	0	0	1	1	1	0	1	0	1	1
0	1	0	0	1	1	1	1	0	1	0	0	1	0	0	1	0	0
0	1	1	1	1	1	0	0	1	0	1	0	0	1	0	0	1	0
1	0	0	1	0	1	0	1	1	1	0	0	0	1	1	1	0	1
1	0	1	1	0	0	1	0	1	1	1	0	0	1	0	1	0	1
1	1	0	0	1	1	0	0	0	0	1	1	1	0	1	1	1	0
1	1	1	0	0	0	1	1	1	0	0	1	1	0	1	0	1	0

6. Контрольные вопросы

1. Что такое СДНФ/СКНФ?
2. Как записать СДНФ/СКНФ, используя таблицу истинности?
3. Как лучше синтезировать логическое устройство (на основе СДНФ или СКНФ), если значение функции в таблице истинности имеет больше нулей, чем единиц?
4. Как разработать логическое устройство, если оно имеет несколько выходов?

5. Что такое минимизация логического выражения, и какие существуют способы минимизация?

6. Каким образом осуществляется минимизация функции при помощи карты Карно?

7. Расскажите, как определить таблицу истинности логического устройства экспериментально, используя лабораторный стенд.

8. Напишите таблицы истинности для основных логических вентилей (И, ИЛИ, НЕ, исключающее ИЛИ).

9. Что такое проект в Altera Quartus II? Для чего необходима компиляция проекта в Altera Quartus II и как ее выполнить?

10. Для чего проводится операция назначения выводов в Altera Quartus II?

ЛАБОРАТОРНАЯ РАБОТА №2
ИССЛЕДОВАНИЕ КОМБИНАЦИОННЫХ СХЕМ. ЗНАКОМСТВО С ЯЗЫКАМИ ПРОЕКТИРОВАНИЯ АППАРАТУРЫ

1. Цель работы

Целью работы является изучение принципов действия комбинационных схем: дешифратора, преобразователя кода для семисегментного индикатора, сумматора, а также знакомство с языками проектирования аппаратуры.

2. Краткие теоретические сведения

2.1. Языки проектирования аппаратуры

VHDL[*] – язык описания аппаратуры интегральных схем. VHDL является базовым языком при разработке аппаратуры современных вычислительных систем. Был разработан в 1983 г. по заказу Министерства обороны США с целью формального описания логических схем для всех этапов разработки электронных систем, начиная модулями микросхем и заканчивая крупными вычислительными системами.

Программа на VHDL состоит из одного или нескольких файлов. В одном файле размещаются одна или несколько пар элементов «объект проекта – архитектура объекта». *Объект проекта* – это интерфейсная часть, в которой указана информация о том, как включать данный объект внутри другого объекта, находящегося на более высоком уровне иерархии. А в *архитектуре объекта* описан алгоритм его функционирования. У одного объекта может быть несколько архитектур, соответствующих разным алгоритмам функционирования, проектам на различных этапах проектирования.

В примерах кода здесь и далее *курсивным шрифтом* указаны комментарии, **полужирным** – ключевые слова рассматриваемого языка.

---

[*] Подробнее основы языка VHDL описаны в [4].

Объект проекта выглядит следующим образом:

```
library std;                  -- подключение библиотеки STD
use std.standard.all;         -- используются все типы пакета
                                 STANDARD библиотеки STD
entity FUNC_NAND is           -- название объекта - FUNC_NAND
     port (                   -- объявление портов
          A :  in BIT;        -- входной порт A типа BIT
          B :  in BIT;        -- входной порт B типа BIT
          C : out BIT);       -- выходной порт C типа BIT
end FUNC_NAND;                -- конец объявления объекта
```

Ключевое слово *port* открывает описание входов-выходов (портов) объекта. Слово *in* указывает на вход, а *out* – на выход. BIT – это тип порта, который, согласно определению этого типа в пакете *standard*, принимает значения 0 и 1.

Описание архитектуры:

```
architecture BEHAVIORAL of FUNC_NAND is -- начало архитектуры
                                           BEHAVIORAL объекта
                                           FUNC_NAND
-- начало описательной (поведенческой) части архитектуры
begin
     C <= not (A and B);              -- параллельный оператор
end BEHAVIORAL;                       -- конец архитектуры
```

Раздел архитектуры состоит из декларативной и описательной частей. В декларативной части объявляются используемые внутри объекта сигналы, константы, специальные (свои собственные) типы, атрибуты, процедуры и функции, описания этих атрибутов, процедур и функций. Описательная часть состоит из списка параллельных операторов. Все параллельные операторы выполняются одновременно, их порядок в списке не имеет никакого значения.

Verilog[*], Verilog HDL – еще один язык описания аппаратуры, используемый для описания и моделирования электронных систем. Verilog был создан в 1984 году в фирме Automated Integrated Design Systems (позднее переименованной в Gateway Design Automation) как собственное средство моделирования. После поглощения последней

[*] Подробнее основы языка Verilog описаны в [5].

фирмой Caddence, язык стал получать все более широкое распространение среди разработчиков и стал не менее популярен, чем VHDL.

Синтаксис Verilog очень похож на синтаксис языка C, что упрощает его освоение. Препроцессор Verilog очень похож на препроцессор языка C, и основные управляющие конструкции *if*, *while* и др. подобны одноименным конструкциям языка C. Программа на Verilog обычно состоит из нескольких модулей. Синтаксис объявления модуля выглядит следующим образом:

```
module FUNC_NAND (A, B, C);
      input A, B;           // Входные порты
      output C;             // Выходной порт

      assign C = ~(A & B);  // Операция И-НЕ
endmodule
```

В заголовке модуля описывается список портов. В теле модуля описывается его функциональность.

В проекте, особенно сложном, бывает много модулей, соединенных между собой. Прежде всего, нужно заметить, что обычно в проекте всегда есть один ***модуль самого верхнего уровня*** (***top level***). Он состоит из нескольких других модулей. Те в свою очередь могут содержать еще модули и так далее. Необязательно, чтобы все модули были написаны на одном языке описания аппаратуры. Совсем наоборот. Довольно удобно и наглядно иметь модуль самого верхнего уровня выполненным в виде схемы, состоящей из модулей более низкого уровня. Эти модули могут быть написаны разными разработчиками, на разных языках (Verilog, VHDL) и даже выполнены в виде схем.

SystemVerilog является надмножеством языка Verilog стандарта 2005 года, но с бóльшими возможностями для верификации и моделирования разработок.

Примечание – при именовании программных объектов (проектов, интерфейсов, архитектур, пакетов, сигналов, переменных, констант, компонентов и др.) в коде следует придерживаться правил, приведенных в Приложении А данного пособия.

2.2. Дешифратор (декодер)

В общем случае ***дешифратор*** (декодер) – это устройство, преобразующее цифровой сигнал в какой-либо кодировке в другую, не закодированную форму. *Классический дешифратор* представляет собой устройство для преобразования *N*-разрядного позиционного двоичного кода в единичный выходной сигнал на одном из 2^N выходов. При каждой входной комбинации сигналов на одном из выходов появляется логическая 1. Таким образом, по единичному сигналу на одном из выходов можно судить о входной кодовой комбинации. Таблица истинности классического дешифратора с двумя входами изображена в таблице 2.1.

Таблица 2.1

Таблица истинности двухразрядного дешифратора

X1	X0	Y3	Y2	Y1	Y0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Для построения схемы дешифратора по таблице истинности воспользуемся методикой, изложенной в лабораторной работе №1. Например, устройство должно иметь 4 выхода. Для каждого выхода записываем логическое выражение. На основе СДНФ:

$$Y0 = \overline{X1} \cdot \overline{X0}, \quad Y1 = \overline{X1} \cdot X0, \quad Y2 = X1 \cdot \overline{X0}, \quad Y3 = X1 \cdot X0$$

По этой системе выражений несложно построить схему требуемого дешифратора (рис. 2.1).

2.3. Преобразователь кода для семисегментного индикатора

Наиболее широко преобразователи кодов известны применительно к цифровым индикаторам. Например, часто востребованы преобразователи позиционного двоичного кода в десятичные цифры.

При помощи семисегментного индикатора (рис. 2.2а) можно высветить десять арабских цифр (рис. 2.3). Такие индикаторы изготавливаются в двух вариантах – с общим анодом (ОА, рис. 2.2б) и с об-

щим катодом (ОК, рис. 2.2в). Следует заметить, что «зажигание» сегмента индикатора с ОА происходит логическим нулем, а с ОК – логической единицей.

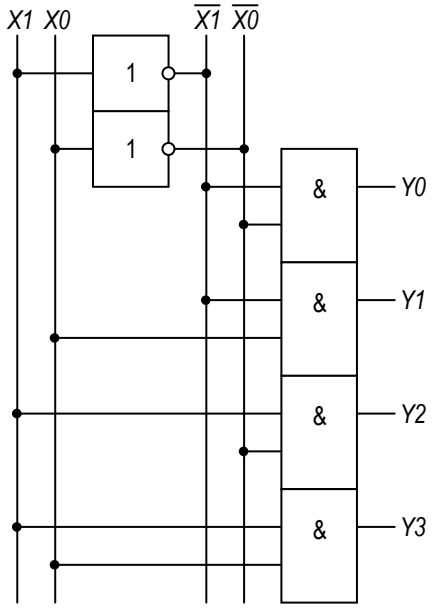
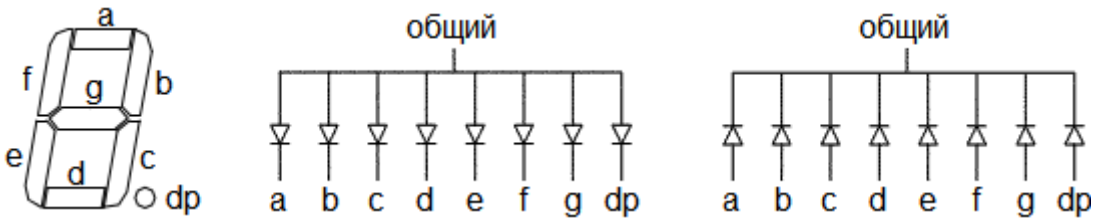


Рисунок 2.1 – Схема дешифратора



а б в

Рисунок 2.2 – Семисегментный индикатор с точкой:
а – внешний вид, б – схема с общим анодом, в – схема с общим катодом

Рисунок 2.3 – Пример отображения цифр на семисегментном индикаторе

Очевидно, что двоичный код должен иметь не менее 4-х разрядов (2^4 = 16, что больше 10). Условное графическое обозначение представлено на рис. 2.4.

Таблица 2.2

Таблица истинности преобразователя кода

Цифра	D3	D2	D1	D0	Ha	Hb	Hc	Hd	He	Hf	Hg
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1
-	Остальные комбинации				0	0	0	0	0	0	1

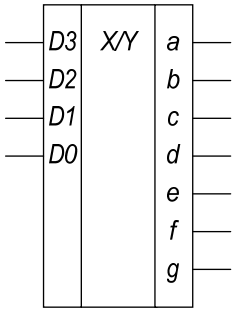


Рисунок 2.4 – Условное графическое обозначение преобразователя кода

2.4. Сумматор

Сложение – одна из самых распространенных операций в цифровых системах.

Полусумматор имеет два входа (A и B) и два выхода (S и $C_{вых}$). S – это сумма A и B. Если же и A, и B равны 1, то выход S должен стать равным 2, но такое число не может быть представлено в виде одного двоичного разряда. В этом случае результат указывается вместе с переносом $C_{вых}$ в следующий разряд. Полусумматор может быть построен из элементов XOR («исключающее ИЛИ») и AND («логическое И»).

Таблица 2.3

Таблица истинности полусумматора

A	**B**	**$C_{вых}$**	**S**
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$S = A \oplus B \qquad C_{вых} = AB$$

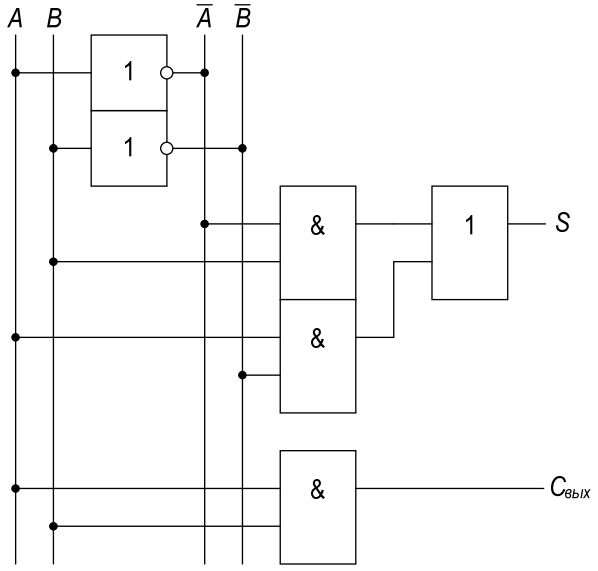


Рисунок 2.5 – Схема полусумматора

Полный сумматор кроме выхода переноса имеет вход переноса с предыдущего разряда сложения $C_{вх}$.

$$S = A \oplus B \oplus C_{вх}$$

$$C_{вых} = AB + AC_{вх} + BC_{вх}$$

N-разрядный сумматор складывает два *N*-разрядных числа (*A* и *B*), а также входной перенос $C_{вх}$ и формирует *N*-разрядный результат *S* и выходной перенос $C_{вых}$. Такой сумматор называется ***сумматором с распространяющимся переносом***, так как выходной перенос одного разряда переходит в следующий разряд.

Таблица 2.4

Таблица истинности полного сумматора

$C_{вх}$	A	B	$C_{вых}$	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

3. Задание к работе

1. Запустить среду Altera Quartus II и создать новый проект с параметрами:

- Имя проекта: LabWork_2_StudentName
- Семейство ПЛИС: Cyclone IV E
- Тип ПЛИС: EP4CE115F29C7

Остальное оставить по умолчанию.

2. Добавить в проект три новых файла типа «Verilog HDL File» или «VHDL File» (на выбор) и описать схемы дешифратора 2×4, преобразователя кода для семисегментного индикатора и полного сумматора.

Внимание! Необходимо учесть, что на стенде семисегментные индикаторы подключены по схеме с *общим катодом*.

3. Добавить в проект файл типа «Block Diagram/Schematic File» и назначить его файлом верхнего уровня (Project → Set as Top-Level Entity).

4. Выполнить анализ и синтез проекта.

5. Выполнить генерацию схемного символа из HDL-описания для каждого модуля по отдельности (File → Create/Update → Create Symbol Files from Current File). После этой операции символы станут доступны в библиотеке инструмента «Symbol Tool» в подпапке «Project».

6. В файле схемы, созданном в п. 3, начертить общую схему, содержащую в себе все исследуемые модули.

7. Выполнить анализ и синтез проекта и сделать назначение выводов в соответствии с таблицами 2.5-2.7. После этого выполнить полную компиляцию проекта.

Таблица 2.5

Назначение выводов дешифратора 2×4

Сигнал	**X1**	**X0**	**Y3**	**Y2**	**Y1**	**Y0**
Вывод ПЛИС	AC27	AC28	F21	E19	F19	G19
На стенде	SW2	SW1	LEDR3	LEDR2	LEDR1	LEDR0

Таблица 2.6

Назначение выводов преобразователя кода
для семисегментного индикатора

Сигнал	**D3**		**D2**		**D1**		**D0**	
Вывод ПЛИС	Y23		Y24		AA22		AA23	
На стенде	SW17		SW16		SW15		SW14	
Сигнал	**Ha**	**Hb**	**Hc**	**Hd**	**He**	**Hf**	**Hg**	
Вывод ПЛИС	AD17	AE17	AG17	AH17	AF17	AG18	AA14	
На стенде	HEX7							

Таблица 2.7

Назначение выводов полного сумматора

Сигнал	**$C_{вх}$**	**A**	**B**	**$C_{вых}$**	**S**
Вывод ПЛИС	AC26	AB27	AD27	E18	F18
На стенде	SW5	SW4	SW3	LEDR5	LEDR4

8. Подключить к компьютеру лабораторный стенд, включить его и запрограммировать. Подавая различные входные сигналы и наблюдая за состояниями светодиодных индикаторов, проанализировать работу исследуемых схем.

4. Содержание отчета

1. Цель работы.

2. Схемы исследования дешифратора, преобразователя кода для семисегментного индикатора, сумматора.

3. Исходные коды модулей дешифратора, преобразователя кода для семисегментного индикатора, сумматора.

3. Таблицы истинности для каждой схемы.

4. Выводы по работе.

5. Контрольные вопросы

1. Объясните принципы работы дешифратора и шифратора.

2. Как синтезировать дешифратор с произвольной разрядностью?

3. Как работает преобразователь кода для семисегментного индикатора? Как устроен семисегментный индикатор?

4. Как работают мультиплексор и демультиплексор?

5. Как работает полный сумматор? В чем заключается его отличие от полусумматора?

6. Что такое бит переноса в сумматоре?

7. Что такое файл верхнего уровня в Altera Quartus II?

8. Опишите структуру VHDL-файла (Verilog-файла).

ЛАБОРАТОРНАЯ РАБОТА №3
ИССЛЕДОВАНИЕ ТРИГГЕРОВ И РЕГИСТРОВ

1. Цель работы

Целью работы является экспериментальное исследование работы различных типов триггеров и регистров.

2. Краткие теоретические сведения

2.1. Триггеры

Триггеры – простые последовательностные схемы, запоминающие один бит информации. Использование триггеров позволяет реализовывать устройства оперативной памяти (то есть памяти, информация в которой хранится только на время вычислений). Однако триггеры могут использоваться и для построения некоторых цифровых устройств с памятью, таких как счетчики, преобразователи последовательного кода в параллельный или цифровые линии задержки.

Одной из простейших последовательностных схем является *RS-триггер*, состоящий, как показано на рис. 3.1, из двух перекрестно включенных элементов ИЛИ-НЕ. У триггера есть два входа – R и S и два выхода Q и $\overline{Q}$.

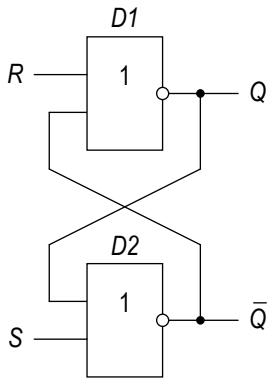


Рисунок 3.1 – Схема RS-триггера

Таблица переходов, приведенная ниже, иллюстрирует четыре возможных состояния триггера. Входы R и S отвечают за сброс (reset) и установку (set) значения на выходе Q соответственно.

Таблица 3.1

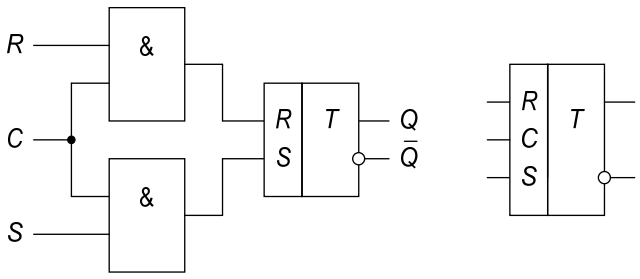
Таблица переходов RS-триггера

S	**R**	**Q**	$\overline{\mathbf{Q}}$	**Режим**
0	0	$Q_{пред}$	$\overline{Q_{пред}}$	Режим хранения
0	1	0	1	Режим установки 0
1	0	1	0	Режим установки 1
1	1	*	*	Запрещенный режим

Запрещенный режим не используется, так как состояния выходов триггера в общем случае не определены и будут зависеть от его конкретной реализации.

Рассмотренный выше триггер является *асинхронным*.

У *синхронного RS-триггера* кроме *R* и *S* входов имеется *вход синхронизации* (*тактируемый вход*, *clock*). Его схема построения на основе асинхронного RS-триггера и условное обозначение показаны на рис. 3.2.



а б

Рисунок 3.2 – Синхронный RS-триггер:
а – схема, б – условное обозначение

Состояние выходов тактируемого RS-триггера может изменяться только в моменты прихода тактовых импульсов. В противном случае элементы И «заперты» и не пропускают сигналы *R* и *S*. В этом случае говорят, что схема работает синхронно.

RS-триггер неудобен из-за необычного поведения в запрещенном режиме. ***D-триггер*** (*D-защелка*, *D-latch*) (рис. 3.3) решает эту проблему. Он построен на основе синхронного RS-триггера и имеет

вход данных *D*, определяющий, каким будет следующее состояние, и вход тактового сигнала *C*.

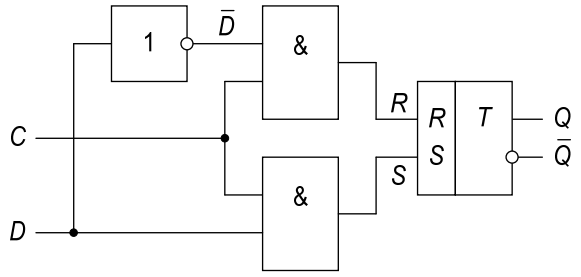


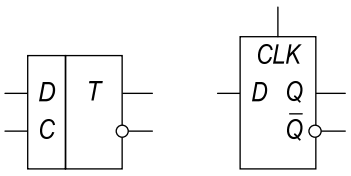
Рисунок 3.3 – Схема D-триггера

Таблица 3.2

Таблица истинности D-триггера

C	**D**	**Q**	$\overline{\mathbf{Q}}$	**Режим**
0	X	$Q_{\text{пред}}$	$\overline{Q_{\text{пред}}}$	Режим хранения
1	0	0	1	«Прозрачный» режим
	1	1	0	

Когда $C = 1$, говорят, что D-триггер «прозрачен», т. е. он пропускает данные *D* на выход *Q*, как если бы он являлся обычным буфером. Когда $C = 0$ – «непрозрачен», т. е. не пропускает новые данные с входа *D* на выход *Q*, а *Q* сохраняет свое предыдущее значение. Условное обозначение D-триггера представлено на рис. 3.4.



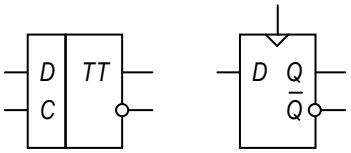
а б

Рисунок 3.4 – Условное обозначение D-триггера
а – отечественное, б – зарубежное

Состояние D-триггера изменяется непрерывно, пока $C = 1$. Такой триггер называется *статическим*.

D-триггер, синхронизируемый фронтом (*динамический*), может быть построен из двух включенных последовательно статических D-триггеров, у которых тактовые сигналы являются булевыми допол-

нениями друг друга. Условное обозначение такого D-триггера приведено на рис. 3.5.



а б

Рисунок 3.5 – Условное обозначение D-триггера, синхронизируемого фронтом: а – отечественное, б – зарубежное

Динамический D-триггер копирует значение с D на Q по заднему или переднему (зависит от реализации триггера) фронту тактового импульса и помнит это состояние все остальное время. Таким образом вход *D* определяет новое, будущее состояние триггера, а фронт определяет конкретный момент времени, когда состояние будет обновлено.

Работа динамического D-триггера, работающего по заднему фронту синхроимпульсов, представлена в таблице 3.3.

Таблица 3.3

Таблица переходов динамического D-триггера

C	**D**	**Q**	$\overline{\mathbf{Q}}$	**Режим**
0	X	$Q_{\text{пред}}$	$\overline{Q_{\text{пред}}}$	Режим хранения
1				
⌐				
⌐	0	0	1	Переключение
⌐	1	1	0	

T-триггер – это счетный триггер. У данного триггера имеется только один вход *T*. После поступления на этот вход импульса, состояние триггера меняется на противоположное. Счетным он называется потому, что Т-триггер как бы подсчитывает количество импульсов, поступивших на его вход. При поступлении второго импульса Т-триггер способен отобразить только младший разряд числа $2_{10} = 10_2$, поэтому на его выходе снова возникает логический ноль.

Т-триггер можно синтезировать из любого двухступенчатого триггера. Рассмотрим пример синтеза Т-триггера из динамического D-триггера. Для того чтобы превратить D-триггер в счетный, необходимо ввести цепь обратной связи с инверсного выхода этого триггера на вход, как показано на рис. 3.6.

Таблица 3.4

Таблица переходов Т-триггера

T	**Q**	$\overline{\mathbf{Q}}$	**Режим**
0	$Q_{\text{пред}}$	$\overline{Q_{\text{пред}}}$	Хранение
1			
⌐			
⌙	0	1	Переключение
	1	0	

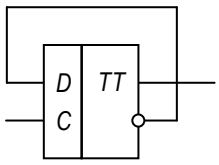


Рисунок 3.6 – Схема Т-триггера на основе двухступенчатого D-триггера

Существует еще одно представление Т-триггера. При разработке схем синхронных двоичных счетчиков важно осуществлять одновременную запись во все его триггеры. В этом случае вход *T* триггера служит только для разрешения изменения состояния на противоположное, а синхронизация производится отдельным входом *C*. Подобная схема Т-триггера приведена на рис. 3.7.

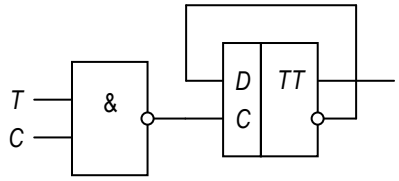
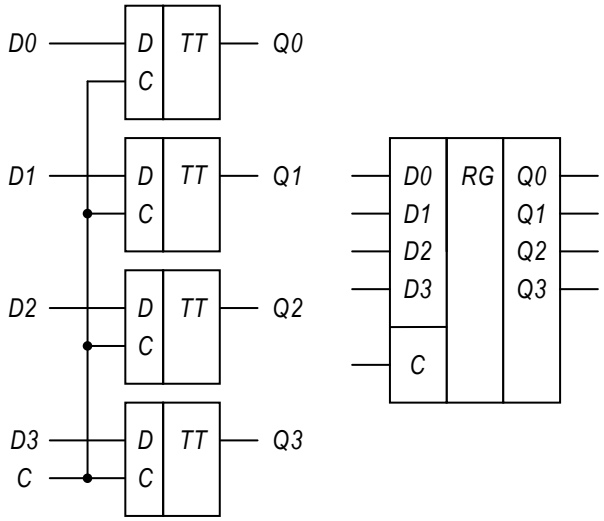


Рисунок 3.7 – Схема синхронного Т-триггера

2.2. Регистры

Регистры предназначены для хранения и преобразования многоразрядных двоичных чисел. ***N-разрядный регистр*** – набор из N триггеров с общим тактовым сигналом. Таким образом, все биты регистра обновляются одновременно. Регистр является ключевым блоком при построении большинства последовательностных схем. На рис. 3.8 показана схема и обозначение четырехразрядного регистра со входами $D3..D0$ и выходами $Q3..Q0$.



а б

Рисунок 3.8 – Четырехразрядный регистр:
а – схема, б – условное обозначение

Сдвигающий регистр имеет вход тактового сигнала, последовательный вход S_{in}, последовательный выход S_{out} и N параллельных выходов. По каждому заднему фронту тактового импульса в первый триггер регистра записывается новый бит со входа S_{in} а содержимое следующих триггеров сдвигается вперед.

Сдвигающий регистр может быть построен из N последовательно соединенных триггеров (рис. 3.9). Такой регистр можно рассматривать как *последовательно-параллельный преобразователь*.

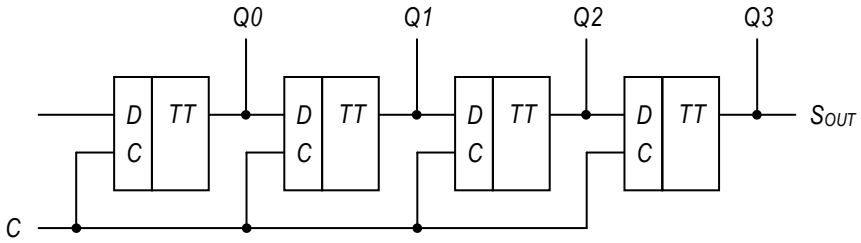


Рисунок 3.9 – Схема 4-разрядного сдвигового регистра

В *параллельно-последовательный преобразователь* параллельно загружается *N* бит, которые затем последовательно (по одному биты за раз) поступают на выход. Схемотехника параллельно-последовательного преобразователя и сдвигающего регистра подобны.

3. Задание к работе

Запустить среду Altera Quartus II и создать новый проект с параметрами:

- Имя проекта: LabWork_3_StudentName
- Семейство ПЛИС: Cyclone IV E
- Тип ПЛИС: EP4CE115F29C7

  Остальное оставить по умолчанию.

3.1. Исследование RS-триггеров

1. Добавить в проект новый файл схемы и начертить схему асинхронного RS-триггера.

2. Выполнить анализ и синтез проекта и сделать назначение выводов в соответствии с таблицей 3.5. После этого выполнить полную компиляцию проекта.

Таблица 3.5

Назначение выводов асинхронного RS-триггера

Сигнал	**R**	**S**	**Q**
Вывод ПЛИС	AC27	AC28	G19
На стенде	SW2	SW1	LEDR0

3. Запрограммировать учебный стенд и, наблюдая за состояниями светодиодных индикаторов, проанализировать работу исследуемой схемы.

4. Преобразовать схему в синхронный RS-триггер, выполнить анализ и синтез проекта и сделать назначение выводов в соответствии с таблицей 3.6. После этого выполнить полную компиляцию проекта.

Ввод сигнала синхронизации производить с кнопки KEY3. При ее нажатии поступает сигнал низкого логического уровня, а при отпускании – высокого, поэтому в цепь сигнала синхронизации необходимо ввести инвертор.

Таблица 3.6

Назначение выводов синхронного RS-триггера

Сигнал	**C**	**R**	**S**	**Q**
Вывод ПЛИС	R24	AC27	AC28	G19
На стенде	KEY3	SW2	SW1	LEDR0

5. Запрограммировать учебный стенд и, наблюдая за состояниями светодиодных индикаторов, проанализировать работу исследуемой схемы.

3.2. Исследование D-триггера

1. Начертить схему динамического D-триггера на основе стандартного библиотечного элемента «DFF» (рис. 3.10).

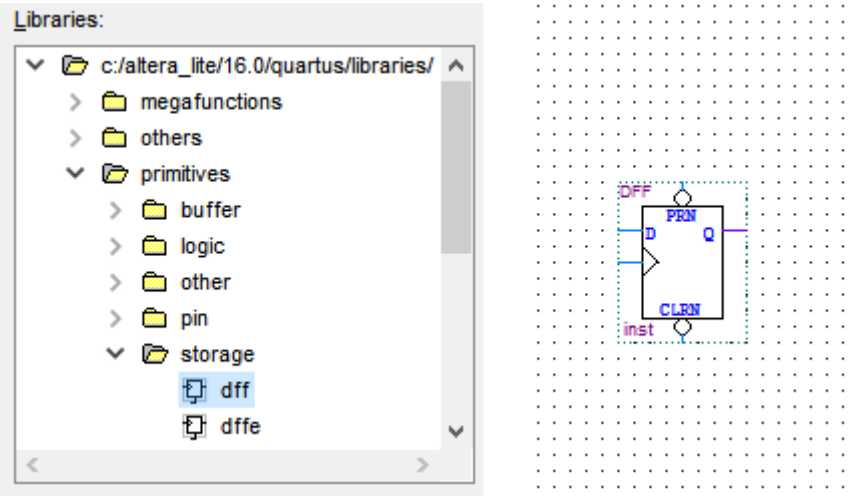


Рисунок 3.10 – D-триггер в библиотеке стандартных логических элементов

2. Выполнить анализ и синтез проекта и сделать назначение выводов в соответствии с таблицей 3.7. После этого выполнить полную компиляцию проекта.

Таблица 3.7

Назначение выводов динамического D-триггера

Сигнал	**D**	**C**	**Q**
Вывод ПЛИС	AC27	R24	G19
На стенде	SW2	KEY3	LEDR0

3. Запрограммировать учебный стенд и, наблюдая за состояниями светодиодных индикаторов, проанализировать работу исследуемой схемы.

3.3. Исследование Т-триггера

1. Начертить схему Т-триггера на основе стандартного библиотечного элемента «TFF».

2. Выполнить анализ и синтез проекта и сделать назначение выводов в соответствии с таблицей 3.8. После этого выполнить полную компиляцию проекта.

Внимание! При использовании ввода информации с кнопок лабораторного стенда возникает дребезг контактов. Описание этого явления и способ борьбы с ним – модуль «button_debouncer» – приведены в Приложении Б. Для того чтобы использовать модуль «button_debouncer», необходимо перенести код, приведенный в Приложении Б, в файл «debouncer.v», а затем включить его в проект. Для этого следует вызвать меню «Assigment → Settings» и на вкладке Files добавить файл с модулем – необходимо указать путь к файлу, нажать кнопку «Add», затем «Apply» и «OK».

Таблица 3.8

Назначение выводов Т-триггера

Сигнал	**G_50MHz**	**T**	**C**	**Q**
Вывод ПЛИС	Y2	AC27	R24	G19
На стенде	50 MHz Oscillator	SW2	KEY3	LEDR0

3. Запрограммировать учебный стенд и, наблюдая за состояниями светодиодных индикаторов, проанализировать работу исследуемой схемы.

3.4. Исследование параллельного регистра

1. Начертить схему параллельного 4-разрядного регистра (см. рис. 3.8).

2. Выполнить анализ и синтез проекта и сделать назначение выводов в соответствии с таблицей 3.9. После этого выполнить полную компиляцию проекта.

Таблица 3.9

Назначение выводов параллельного регистра

Сигнал	**G_50MHz**	**C**	**D3**	**D2**	**D1**	**D0**
Вывод ПЛИС	Y2	R24	AD27	AC27	AC28	AB28
На стенде	50 MHz Oscillator	KEY3	SW3	SW2	SW1	SW0

Сигнал	**Q3**	**Q2**	**Q1**	**Q0**
Вывод ПЛИС	F21	E19	F19	G19
На стенде	LEDR3	LEDR2	LEDR1	LEDR0

3. Запрограммировать учебный стенд и поочередно произвести запись нескольких 4-разрядных двоичных чисел в регистр.

Наблюдая за состояниями светодиодных индикаторов, проанализировать работу исследуемой схемы.

3.5. Исследование сдвигающего регистра

1. Начертить схему сдвигающего 4-разрядного регистра (см. рис. 3.9).

2. Выполнить анализ и синтез проекта и сделать назначение выводов в соответствии с таблицей 3.10. После этого выполнить полную компиляцию проекта.

Таблица 3.10

Назначение выводов параллельного регистра

Сигнал	**G_50MHz**		**C**	**S_{in}**
Вывод ПЛИС	Y2		R24	AB28
На стенде	50 MHz Oscillator		KEY3	SW0
Сигнал	**Q3**	**Q2**	**Q1**	**Q0**
Вывод ПЛИС	F21	E19	F19	G19
На стенде	LEDR3	LEDR2	LEDR1	LEDR0

3. Запрограммировать учебный стенд и, наблюдая за состояниями светодиодных индикаторов, проанализировать работу исследуемой схемы.

4. Содержание отчета

1. Цель работы.
2. Схемы исследования триггеров и регистров и их условно-графические обозначения.
3. Таблицы переходов схем триггеров.
4. Выводы по работе.

5. Контрольные вопросы

1. Начертить схему RS-триггера на логических элементах ИЛИ-НЕ и пояснить принцип его работы.
2. Чем синхронный RS-триггер отличается от асинхронного? Как он реализуется?
3. Пояснить по таблице переходов работу D-триггера.
4. Какой характерной особенностью обладает периодическая последовательность импульсов на входе Т-триггера?
5. Каким преимуществом обладает динамический D-триггер?
6. Чем определяется разрядность регистров?
7. Параллельный регистр: его схема и принцип работы.
8. Последовательно-параллельный регистр: его схема и принцип работы.

ЛАБОРАТОРНАЯ РАБОТА №4
ИССЛЕДОВАНИЕ И СИНТЕЗ СЧЕТЧИКОВ

1. Цель работы

Целью работы является изучение универсального двоичного счетчика и приобретение навыков в построении и экспериментальном исследовании счетчиков.

2. Краткие теоретические сведения

Счетчик – устройство для подсчёта числа входных импульсов. Они используются для построения схем таймеров или для выборки инструкций из ПЗУ в микропроцессорах. Они могут использоваться как делители частоты в управляемых генераторах частоты (синтезаторах). При использовании в цепи фазовой автоподстройки счетчики могут быть использованы для умножения частоты как в синтезаторах, так и в микропроцессорах.

Основные параметры счетчика:

- M (модуль счета) – число устойчивых состояний;
- E (емкость) – максимальное число, которое может быть записано в счетчик ($E = M - 1$);

2.1. Простейший асинхронный двоичный счетчик

Простейший асинхронный счетчик представляет собой несколько последовательно включенных счетных триггеров (рис. 4.1). Напомним, что по каждому входному импульсу счетный триггер изменяет свое состояние на противоположное.

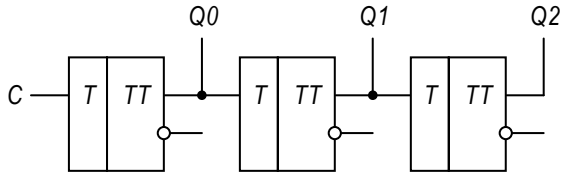


Рисунок 4.1 – Простейший суммирующий асинхронный счетчик

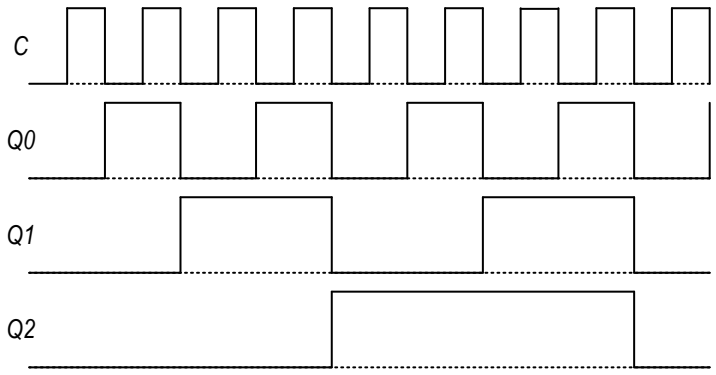


Рисунок 4.2 – Временная диаграмма работы суммирующего асинхронного счетчика

Для того чтобы разобраться, как работает схема двоичного счетчика, воспользуемся временными диаграммами сигналов на входе и выходах этой схемы, приведенными на рис. 4.2.

Пусть первоначальное состояние всех триггеров счетчика будет нулевым. Это состояние видно на временных диаграммах. После поступления на вход счетчика тактового импульса (который воспринимается по заднему фронту) первый триггер изменяет свое состояние на противоположное, то есть единицу. Так как по приходу первого импульса изменилось состояние первого триггера, то этот триггер содержит младший разряд двоичного числа (единицы).

Подадим на вход счетчика еще один тактовый импульс. Значение первого триггера снова изменится на прямо противоположное. На этот раз на выходе первого триггера, а значит и на входе второго триггера, сформируется задний фронт. Это означает, что второй триггер тоже изменит свое состояние на противоположное. Это отчетливо видно на временных диаграммах, приведённых на рис. 4.2.

Продолжая анализировать временную диаграмму, можно определить, что на выходах приведенной схемы счетчика последовательно появляются цифры от 0 до 7. Эти цифры записаны в двоичном виде. При поступлении на счетный вход счетчика очередного импульса, содержимое его триггеров увеличивается на 1. Поэтому такие счетчики получили название суммирующих двоичных счетчиков. Если информацию снимать с инверсных выходов триггеров, то получится вычитающий счетчик.

2.2. Счетчик с произвольным модулем счета

Для построения такого счетчика можно использовать двоичный счетчик, у которого модуль счета M должен быть больше модуля счета разрабатываемого счетчика с произвольным модулем счета.

Пусть нужно сделать счетчик с $M = 10$. У четырехразрядного счётчика модуль счета равен 16 (больше 10). Схема счетчика представляет собой 4 последовательно включенных счетных триггера, у которых есть вход сброса R (см. рис. 4.3). Число 10 в двоичной системе счисления представляется 1010. Когда на выходах счетчика будет код 1010, на выходе элемента «И» появится логическая единица, которая запустит схему гашения. Длительность импульса на выходе схемы гашения должна быть достаточна для надежного сброса всех триггеров счетчика в 0. Разряды числа 1010, равные 1, подаются на схему «И» с прямых выходов триггеров, а равные 0 – с инверсных. Таким образом, как только счетчик досчитает до 10, произойдет обнуление всех триггеров, и счет продолжится с кода 0000.

В качестве схемы гашения может быть применен RS-триггер. Сигнал на входе R счетчика будет действовать в течение одного периода входных импульсов.

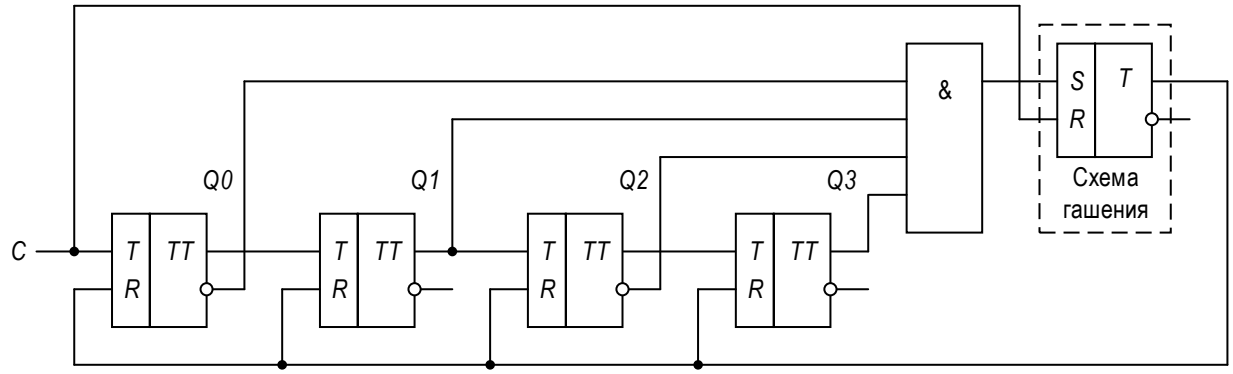


Рисунок 4.3 – Счетчик с модулем счета $M = 10$

Условно-графическое обозначение суммирующего двоичного счетчика на принципиальных схемах приведено на рисунке 4.4. В двоичных счетчиках обычно предусматривают вход обнуления микросхемы R, который позволяет записать во все триггеры счетчика нулевое значение. Это состояние иногда называют исходным состоянием счетчика.

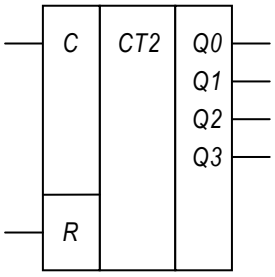


Рисунок 4.4 – Условное графическое обозначение
четырехразрядного двоичного счетчика

2.3. Кольцевой счетчик

В рассмотренных схемах счетчиков быстродействие всей схемы определяется временем распространения сигнала от входа до выхода самого старшего разряда. При этом получается, что чем больше требуемый коэффициент деления, тем больше двоичных разрядов счетчика требуется для реализации этого делителя. Тем большее время требуется для распространения сигнала от входа синхронизации счетчика, до его выхода, и тем меньше будет предельная частота, которую можно подавать на вход этого делителя.

Чтобы обойти эту неприятную особенность, нужно, чтобы счетчик подготавливал свое новое состояние в промежутках между тактовыми импульсами и только записывал его по приходу нового импульса.

Простая схема, реализующая данный принцип, – это *схема кольцевого счетчика*. Такой счетчик можно построить на основе сдвигового регистра. Схема кольцевого счетчика и временные диаграммы его работы приведены на рисунке 4.5.

Как видно, при счете от первого разряда к последнему распространяется «волна» единиц, а затем «волна» нулей. Для устранения этого эффекта вводится дополнительное комбинационное устройство (КУ) для формирования правильных чисел (рис. 4.6).

В качестве преимущества схемы кольцевого счетчика можно отметить то, что ее быстродействие зависит только от времени задержки одного триггера. Это означает, что на кольцевых счетчиках можно реализовывать самые быстродействующие делители частоты.

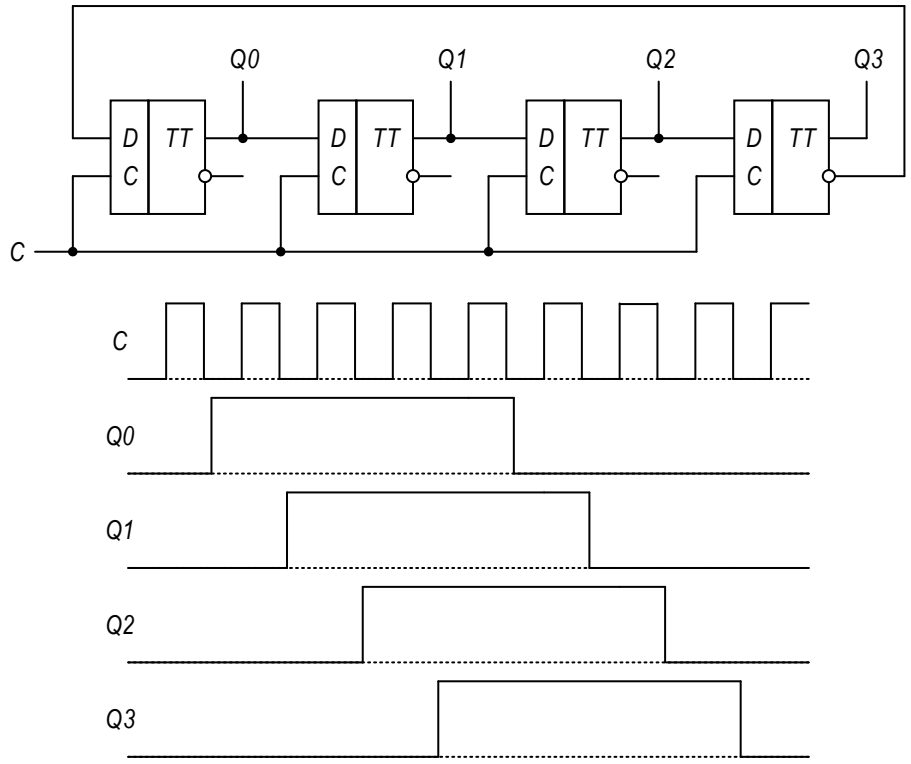


Рисунок 4.5 – Схема и временные диаграммы кольцевого счетчика

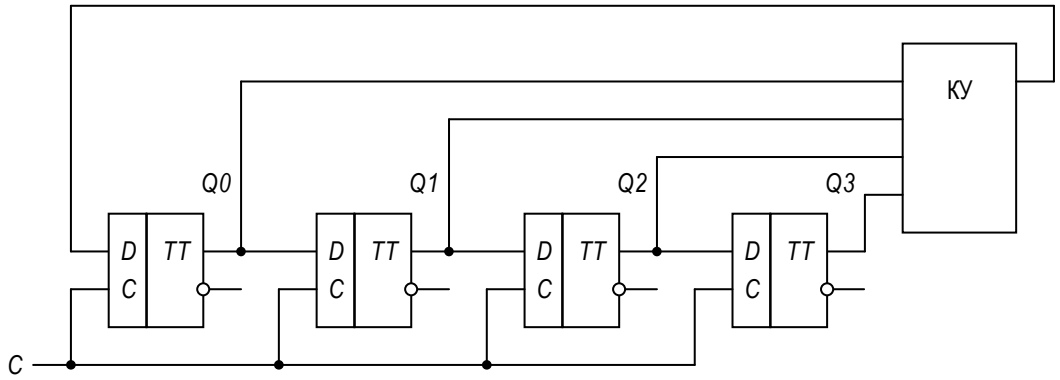


Рисунок 4.6 – Схема двоичного счетчика на основе кольцевого счетчика

3. Задание к работе

Запустить среду Altera Quartus II и создать новый проект с параметрами:

- Имя проекта: LabWork_4_StudentName
- Семейство ПЛИС: Cyclone IV E
- Тип ПЛИС: EP4CE115F29C7
  Остальное оставить по умолчанию.

3.1. Исследование работы реверсивного счетчика режимах суммирования, вычитания и параллельной загрузки

1. Изучить описание и режимы работы реверсивного счетчика SN74193 (отечественный аналог КР1533ИЕ7) в Приложении В.

2. Добавить в проект новый файл схемы и начертить схему счетчика на основе библиотечного элемента «74193» (данный модуль является моделью реальной микросхемы SN74193/КР1533ИЕ7).

3. Выполнить анализ и синтез проекта и сделать назначение выводов в соответствии с таблицей 4.1. После этого выполнить полную компиляцию проекта.

Таблица 4.1

Назначение выводов счетчика

Сигнал	**G_50MHz**	**CU**	**CD**	**D3**	**D2**	**D1**	**D0**
Вывод ПЛИС	Y2	R24	N21	AD27	AC27	AC28	AB28
На стенде	50 MHz Oscillator	KEY3	KEY2	SW3	SW2	SW1	SW0

Сигнал	**LOAD**	**RESET**	**Q3**	**Q2**	**Q1**	**Q0**
Вывод ПЛИС	M21	M23	G21	G22	G20	H21
На стенде	KEY1	KEY0	LEDG7	LEDG6	LEDG5	LEDG4

4. Запрограммировать учебный стенд и, наблюдая за состояниями светодиодных индикаторов, проанализировать работу счетчика в трех режимах – суммирование, вычитание, параллельная загрузка. В последнем режиме на параллельный вход счетчика необходимо загрузить число в соответствии со своим вариантом (см. табл. 4.2).

3.2. Исследование счетчика с произвольным модулем счета

1. Изменить исходную схему таким образом, чтобы счетчик считал до числа, определенного вариантом (см. табл. 4.2).

2. Выполнить полную компиляцию проекта.

3. Запрограммировать учебный стенд и, наблюдая за состояниями светодиодных индикаторов, убедиться в правильной работе схемы.

4. Содержание отчета

1. Цель работы.

2. Схема исследования счетчика.

3. Временные диаграммы входных и выходных импульсов в режимах суммирования и вычитания.

4. Схема исследования счетчика с произвольным модулем счета.

5. Временные диаграммы входных и выходных импульсов счетчика с произвольным модулем счета.

6. Выводы по работе.

5. Варианты заданий

Таблица 4.2

Варианты заданий

Вариант	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Число	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

6. Контрольные вопросы

1. Объяснить принцип работы суммирующего счетчика.

2. Изобразить временные диаграммы работы суммирующего счетчика.

3. Объяснить принцип работы вычитающего счетчика.

4. Изобразить временные диаграммы работы вычитающего счетчика.

5. Объяснить принцип работы счетчика с произвольным модулем счета.

6. В чем заключается преимущество кольцевого счетчика перед асинхронными?

7. Назовите основные параметры двоичных счетчиков.

8. Где применяются счетчики?

ЛАБОРАТОРНАЯ РАБОТА №5
ВЕРИФИКАЦИЯ HDL-ПРОЕКТА

1. Цель работы

Целью работы является изучение возможностей верификации HDL-проектов в среде ModelSim.

2. Краткие теоретические сведения

Верификация – в общем случае, это проверка совпадения результатов работы разрабатываемого устройства и требований, которые к нему предъявляются.

HDL-языки имеют в своем составе необходимый инструментарий для проведения верификации и отладки проектов ПЛИС. В качестве среды симуляции и отладки предлагается использовать программу ModelSim, входящую в состав САПР Altera Quartus II.

2.1. Параллельный оператор PROCESS в VHDL

Оператор *process* относится к классу операторов параллельной обработки, что дает возможность использовать его для проектирования цифровых устройств комбинаторного типа. В то же время он является родительским оператором для всех других выполняемых операторов, которые разрешено располагать в его теле. Говоря другими словами, все выполняемые операторы, располагаемые в теле оператора *process*, выполняются по последовательному принципу. Оператор *process* может использоваться как в явном виде, так и в неявном.

Явно заданный оператор *process* – это основная конструкция для поведенческой формы описания проектов. Различают два подкласса явно заданного оператора ***process***: со списком чувствительности и без списка чувствительности.

Список чувствительности (*sensitivity list*) – это набор сигналов, изменение значения которых вызывает немедленный запуск на выполнение оператора *process*. Если список чувствительности не определен, внутри такого процесса должен содержаться оператор *wait*, чтобы предоставить разработчику возможность управлять запуском и

остановкой соответствующего оператора *process*. Оператора без списка чувствительности всегда запускается на выполнение автоматически в момент начала процесса моделирования работы проекта.

Синтаксис явно заданного оператора *process* имеет следующий вид (в квадратных скобках указаны необязательные конструктивные элементы):

```
[метка_процесса:] process [(список_чувствительности)] [is]
[операторы_объявлений_процесса] -- раздел объявлений
begin                  -- раздел выполняемых операторов
     -- Тут размещаются операторы следующих типов:
     -- установки значений сигналов;
     -- присвоения значений переменным;
     -- вызовы процедур;
     -- операторы case, exit, if, loop, next, null или wait;
end process [метка_процесса];
```

Необходимо отметить, что присвоение значения переменной в теле оператора *process* обозначается символами «:=», а установка значения сигнала символами «<=».

2.2. Последовательные операторы VHDL

Оператор *if* относится к семейству последовательных операторов, располагаемых в разделе выполняемых операторов явно заданного процесса. Оператор *if* ответствен за выполнение того или иного блока последовательных операторов в зависимости от истинности одного или нескольких условий (condition). Синтаксис оператора *if* имеет следующий вид:

```
if условие_1 then
     блок_операторов_1;
[elsif условие_2 then
     блок_операторов_2;]
...
[else
     блок_операторов_3;]
end if;
```

Оператор *wait* приостанавливает выполнение тела оператора process до тех пор, пока не происходит очередное событие.

Язык VHDL поддерживает несколько форм оператора *wait*, а именно:

```
wait until условие;   -- ждать выполнения условия
wait for время;       -- ждать заданное время, например, 25ns
wait on сигнал;       -- ждать изменения значения сигнала
wait;                 -- бесконечное ожидание
```

2.3. Testbench

Testbench – это виртуальный «стенд тестирования» цифровой схемы. Создается среда, внутрь которой помещается тестируемая схема как компонент. То есть файл Testbench – это такой же HDL-файл, компонентом которого является тестируемая схема, описанная на HDL-языке.

Внутри этой среды описывается генерация входных сигналов для схемы, которые затем подаются на вход для последующего анализа выходов тестируемой схемы. При этом сам Testbench не имеет внешнего интерфейса.

Затем этот файл симулируется в программе симуляторе, которая будет выполнять действия, описанные в этом файле и представлять результат в графическом виде в виде диаграмм изменения сигналов.

Изменяемым параметром будет являться время – симулятор последовательно будет увеличивать время с необходимым разрешением и последовательно вычислять значения сигналов.

Пример Testbench-файла, описанного на VHDL:

```
library ieee;
use ieee.std_logic_1164.all;

entity temp_vhd_tst is
end temp_vhd_tst;

architecture temp_arch of temp_vhd_tst is
      signal x1 : std_logic;
      signal x2 : std_logic;
      signal x3 : std_logic;
      signal y  : std_logic;
-- объявление тестируемой схемы
component temp
      port (
            x1 : in std_logic;
```

```
            x2 : in std_logic;
            x3 : in std_logic;
            y  : out std_logic
      );
end component;
begin
      -- вставка тестируемой схемы
      i1 : temp
      -- связь между портами и сигналами
      port map (
            x1 => x1,
            x2 => x2,
            x3 => x3,
            y => y
      );

      init : process
      begin
            -- однократно выполняющийся код
            wait;
      end process init;

      always : process
      -- список чувствительности
      begin
      -- код, выполняющийся после изменения любого из сигналов
      из списка чувствительности
      end process always;
end temp_arch;
```

2.4 Операторы проверки условий и вывода сообщений в консоль

Язык VHDL допускает встраивание в файл Testbench проверку выполнения определенных условий. Это производится с помощью ***оператора assert***. Пример его использования показан ниже:

```
assert SUM_EXPECTED = SUM_OUT -- условие для проверки
report "ERROR: output SUM is incorrect" severity warning;
```

После ключевого слова *assert* указывается условие, которое предполагается истинным (при нормальной работе модуля оно должно выполняться). Если условие не выполняется, в консоль выводится сообщение, указанное после ключевого слова ***report***. После *severity* указывается уровень важности проверяемого условия.

Уровень важности может быть следующим:

- *note* – «примечание», это сообщение не является ошибкой и носит информационный характер;
- *warning* – «предупреждение», такие сообщения подсчитываются, но их наличие не является основанием для признания результатов некорректными;
- *error* – «ошибка», результат моделирования признается некорректным, моделирование продолжается;
- *failure* – «сбой», аналогично ошибке, но выполнение немедленно прерывается.

По умолчанию сообщения имеют уровень *error*.

Назначение уровня важности производится самим разработчиком теста. Предполагается, что сообщения уровня «сбой» будут присваиваться ошибкам, которые не могут быть достаточно легко исправлены, или относятся к «ситуациям, которые никогда не могут возникнуть». Возникновение сообщения такого уровня должно означать, что в процессе моделирования возникли существенные проблемы, и производить дальнейший анализ не имеет смысла.

3. Задание к работе

3.1. Симуляция проекта вручную

1. Запустить среду Altera Quartus II и создать проект схемы в соответствии со своим вариантом (см. табл. 5.1). Следует иметь в виду, что ModelSim не поддерживает симуляцию файлов схем, поэтому предпочтительнее описать схему на одном из HDL-языков.

Примечание. Для симуляции схемы требуется преобразовать ее в HDL-описание (File → Create/Update → Create HDL File from Current File). В открывшемся окне выбрать любой удобный язык. Открыть полученный файл и добавить его в проект (Project → Add Current File to Project). Назначить его файлом верхнего уровня (Project → Set as Top-Level Entity), а файл со схемой (расширение *.bdf) удалить из проекта (Project → Add/Remove Files in Project).

2. Выполнить анализ и синтез проекта, назначение выводов и полную компиляцию проекта. В случае успешной компиляции, запус-

тить симуляцию проекта (Tools → Run Simulation Tool → RTL Simulation). При этом следует выбрать язык, соответствующий файлу верхнего уровня проекта. Altera Quartus II запустит программу ModelSim.

3. В открывшемся окне ModelSim запустить симуляцию своего проекта (Simulate → Start Simulation) и выбрать файл верхнего уровня проекта в каталоге «work». На панели Objects появятся порты и сигналы, имеющиеся в проекте. Выделить входные и выходные порты и перетащить их, захватив мышью, в панель Wave.

4. Задать сигналы на входных портах таким образом, чтобы они образовывали инкрементирующийся двоичный код (Wave → Clock…), параметр First Edge задать «Falling». Установить общее время симуляции, достаточное для симуляции всех возможных состояний входов, и запустить симуляцию (Simulate → Run → Run 100).

5. Программа проведет симуляцию заданного времени работы схемы и отобразит результат в виде временной диаграммы на панели Wave. Проверить соответствие диаграммы таблице истинности заданной функции.

6. Завершить работу программы ModelSim.

3.2. Testbench

1. В Altera Quartus II выполнить генерацию шаблона файла Testbench (Processing → Start → Start Test Bench Template Writer). На панели Messages найти строку с информацией о расположении сгенерированного файла (см. рис. 5.1).

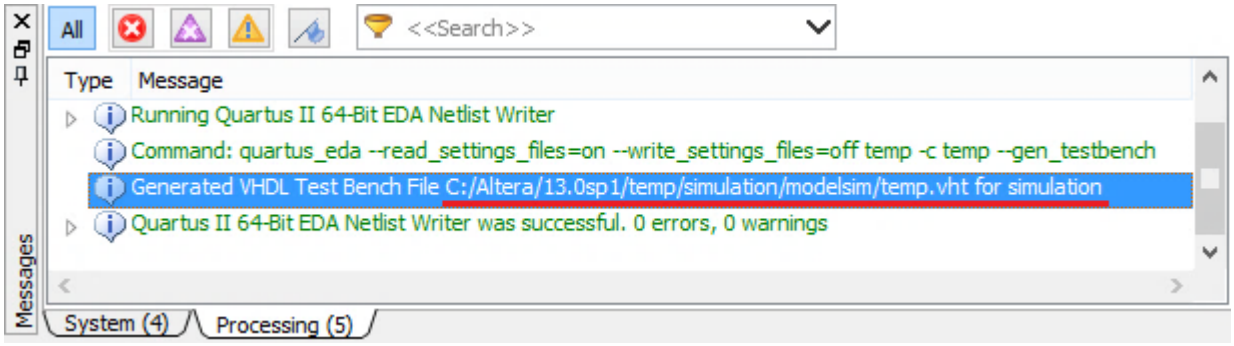


Рисунок 5.1 – Пример сообщения Quartus II о расположении файла Testbench

2. Выполнить привязку файла Testbench к проекту. Для этого открыть настройки проекта (Assignments → Settings), раздел Simulation и добавить новый Testbench, следуя указаниям рисунка 5.2.

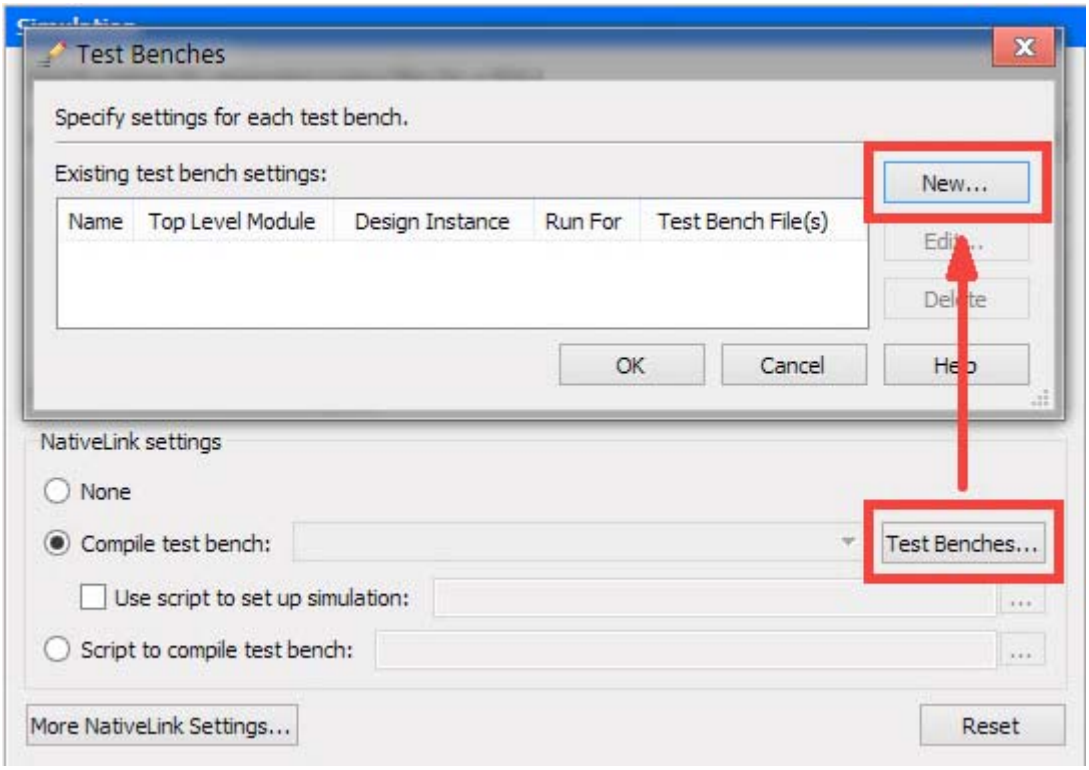


Рисунок 5.2 – Настройки симуляции, добавление Testbench

В открывшемся окне настроек (рис. 5.3) ввести:

- Test bench name: произвольное имя;
- Top level module in test bench: имя файла Testbench верхнего уровня (указан в исходном тексте файла Testbench как имя объекта *entity*);
- Simulation period: период симуляции в соответствии с п. 3.1;
- Test bench and simulation files: выбрать Testbench-файл и нажать кнопку Add.

3. Открыть Testbench-файл в среде Altera Quartus II (File → Open) и ввести код, описывающий окружение для тестирования проекта: генерация входных сигналов и вывод консольных сообщений о старте и завершении теста.

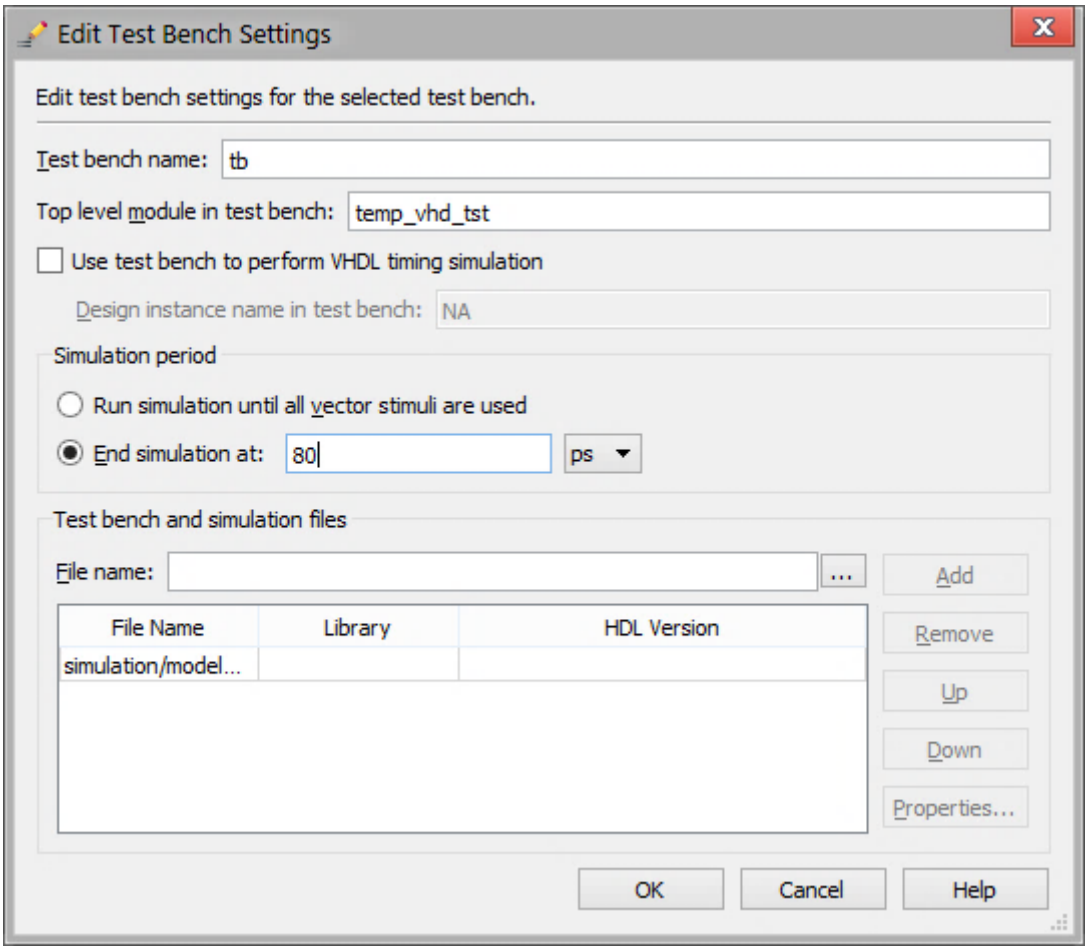


Рисунок 5.3 – Настройки Testbench

4. Выполнить компиляцию проекта. В случае успешной компиляции запустить симуляцию проекта (Tools → Run Simulation Tool → RTL Simulation).

5. Результат симуляции отобразится в виде диаграммы на панели Wave. Текстовые сообщения выводятся в панели Transcript. Убедиться, что выводится сообщение об окончании тестировании.

6. Завершить работу программы ModelSim.

4. Содержание отчета

1. Цель работы.
2. HDL-описание схемы и ее таблица истинности по варианту.
3. Временная диаграмма симуляции.
4. Исходный код файла Testbench.
5. Выводы по работе.

5. Варианты заданий

Таблица 5.1

Варианты заданий

Вариант	**1**	**2**	**3**	**4**	**5**
Функция	$A(B+C)$	$A+BC$	$\overline{ABC}$	$\overline{A}+\overline{BC}$	$A \oplus B \oplus C$
Вариант	**6**	**7**	**8**	**9**	**10**
Функция	$\overline{A \oplus B \oplus C}$	ABC	$A+B+C$	$\overline{AB+C}$	$\overline{A+B+C}$
Вариант	**11**	**12**	**13**	**14**	**15**
Функция	$A \oplus BC$	$\overline{A+B \oplus C}$	$\overline{A}B+C$	$\overline{A+B}+C$	$\overline{AB \otimes C}$

6. Контрольные вопросы

1. Что такое симуляция проекта? Для чего она проводится?

2. Объясните принцип тестирования проектов для ПЛИС с использованием файлов Testbench.

3. Опишите структуру файла Testbench. Чем он отличается от VHDL-файла?

4. Что такое компонент в VHDL? Как он реализуется?

5. Поясните работу оператора *process*. Каково его назначение в языке VHDL?

6. Как работает оператор условия в языке VHDL? Приведите пример использования оператора *if*.

7. Приведите пример описания периодического сигнала.

8. Как вывести сообщение при тестировании проекта?

ЛАБОРАТОРНАЯ РАБОТА №6
ИЗУЧЕНИЕ АРХИТЕКТУРЫ И ПРИНЦИПОВ РАБОТЫ RISC-ПРОЦЕССОРА

1. Цель работы

Целью работы является изучение архитектуры и системы команд процессора Nios II, а также приобретение навыков его программирования на языке ассемблера.

2. Краткие теоретические сведения

Процессор Nios II – это 32-разрядная процессорная RISC архитектура для применения во встраиваемых системах (soft processor или программный процессор), разработанная специально для ПЛИС фирмы Altera. Nios II является развитием архитектуры Nios и находит применение в различных встраиваемых приложениях – от систем цифровой обработки сигналов до разнообразных устройств управления.

Существуют 3 различные версии конфигурации процессора.

- *Nios II/f (fast)* – версия, предназначенная для достижения максимальной производительности. Конфигурация имеет широкий набор опций для оптимизации процессора по производительности.
- *Nios II/s (standart)* – стандартная версия. Данная конфигурация требует меньше ресурсов для реализации и характеризуется меньшей производительностью.
- *Nios II/e (economy)* – экономичная версия. Данная конфигурация требует наименьшее количество ресурсов для реализации, но обладает ограниченным набором возможностей.

2.1. Архитектура процессора Nios II

Процессор Nios II имеет RISC архитектуру, в которой все арифметические и логические операции выполняются над операндами, хранящимися в регистрах общего назначения. Обмен информацией

между регистрами и памятью осуществляется путем выполнения специальных команд «Load» и «Store».

Длина машинного слова процессора Nios II составляет 32 бита. Такую же разрядность имеют и его регистры. Процессор использует раздельные шины для команд и данных, то есть построен по гарвардской архитектуре. Структурная схема процессора Nios II представлена на рис. 6.1.

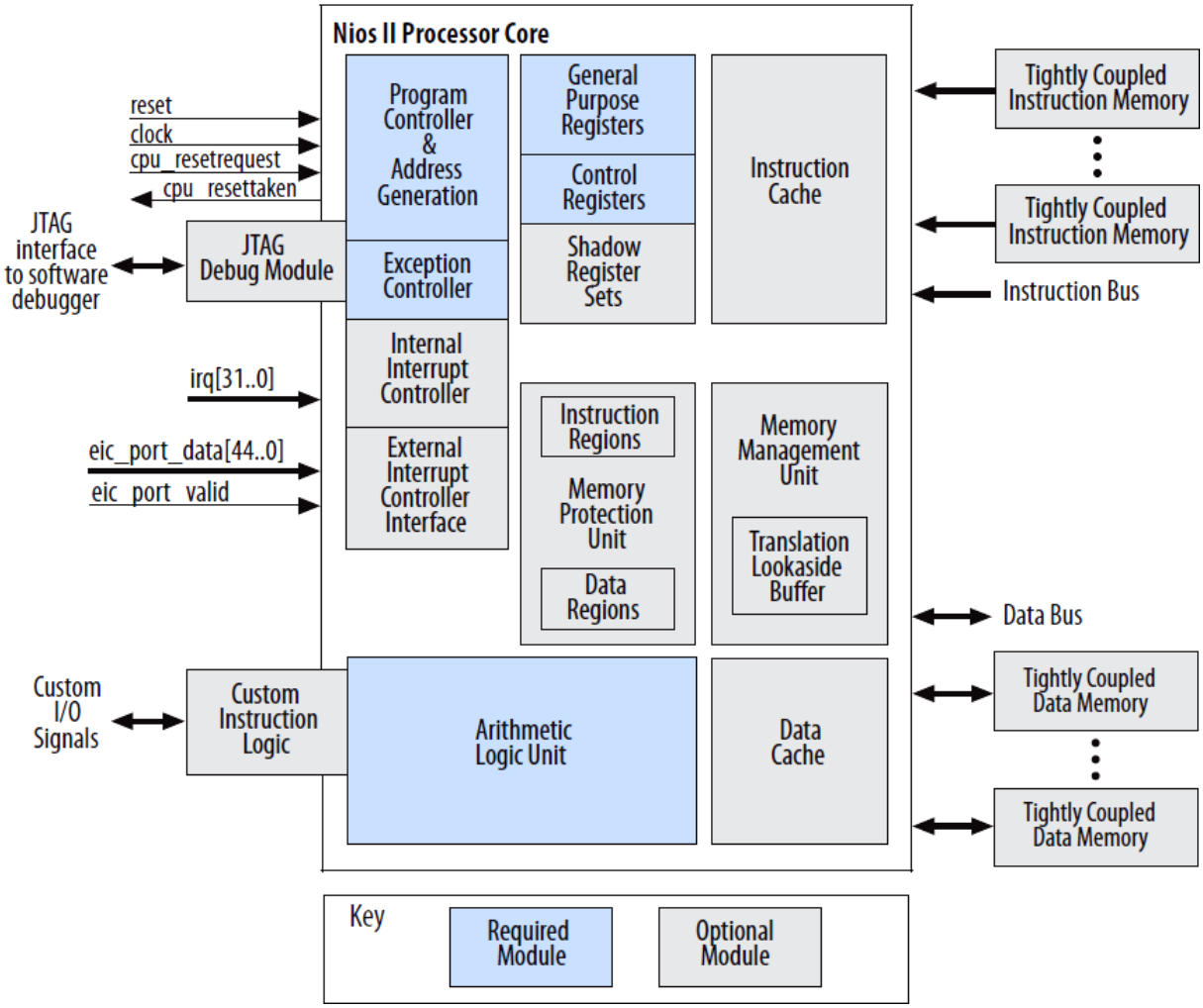


Рисунок 6.1 – Структурная схема процессора Nios II

Процессор может функционировать в двух режимах – в режиме супервизора (процессор переключается в данный режим после поступления сигнала сброса), где ему разрешается выполнять все инструкции и осуществлять любые функции, и режиме пользователя, в котором ограничена возможность выполнения определенных инструкций

системного назначения (доступен только при наличии модуля управления памятью MMU или модуля защиты памяти MPU).

Архитектурой процессора Nios II поддерживается однородный регистровый файл, состоящий из тридцати двух регистров общего назначения (РОН) и до тридцати двух управляющих регистров. Все регистры имеют имена, распознаваемые ассемблером. Регистры общего назначения и управляющие регистры, представлены в таблицах 6.1 и 6.2, соответственно. В состав процессора может входить до 32 управляющих регистров. Их общее количество зависит от наличия модулей защиты памяти или управления памятью.

Таблица 6.1

Регистры общего назначения процессора Nios II

Регистр	**Имя**	**Назначение**
r0	zero	Регистр нуля, всегда содержит значение 0x00000000
r1	at	Временный регистр, используется ассемблером
r2, r3	v0, v1	Возвращаемое значение
r4..r7	a0..a3	Регистры аргументов
r8..r15	t0..t7	Временные регистры
r16..r23	s0..s7	Сохраняемые регистры
r24	et	Временный регистр для обработки исключений
r25	bt	Временный регистр, используемый при отладке
r26	gp	Глобальный указатель
r27	sp	Указатель стека
r28	fp	Указатель кадра
r29	ea	Адрес возврата из исключений (недоступен в пользовательском режиме)
r30	ba	Возврат из контрольной точки (используется только модулем отладки JTAG)
r31	ra	Адрес возврата при вызове подпрограмм

Опционально, для ускорения контекстного переключений, процессор может иметь до 63 наборов теневых (shadows) регистров. Для работы с ними процессор имеет две специальные инструкции, позволяющие перемещать данные между наборами регистров. Текущий используемый набор теневых регистров отображает поле CRS статусного регистра.

Для адресации процессор Nios II использует 32-битный адрес с побайтовой адресацией (порядок Little-endian). С помощью соответствующих команд можно записывать или считывать слова (32 бита), полуслова (16 бита) и байты данных (8 бит).

В процессоре Nios II определены несколько способов адресации. *Непосредственная адресация* – в команде присутствует 16-битный операнд (он может дополняться до 32 разрядов при выполнении арифметических операций); *регистровая адресация* – операнды находятся в регистрах процессора; *относительная регистровая адресация* – эффективный адрес операнда вычисляется суммированием содержимого регистра и знакового 16-разрядного числа, находящегося в самой команде, которое определяет смещение; *косвенная регистровая адресация* – содержимое регистра является эффективным адресом операнда (этот способ эквивалентен предыдущему при нулевом смещении), *абсолютная адресация* – 16-битный абсолютный адрес операнда определен смещением относительно регистра нуля.

Таблица 6.2

Управляющие регистры процессора Nios II

Регистр	**Имя**	**Назначение**
ctl0	status	Регистр состояния: бит 0 PIE – разрешение внешних прерываний, бит 1 U – режим работы процессора, биты 2..32 – зарезервированы
ctl1	estatus	Копия регистра состояния во время прерываний: бит 0 EPIE – сохраненный бит PIE, бит 1 EU – сохраненный бит U, биты 2..32 – зарезервированы
ctl2	bstatus	Копия регистра состояния во время точек останова: бит 0 PIE – сохраненный бит PIE, бит 1 U – сохраненный бит U, биты 2..32 – зарезервированы
ctl3	ienable	Регистр разрешения прерываний
ctl4	ipending	Регистр активных прерываний
ctl5	cpuid	Уникальный идентификатор процессора для многопроцессорных систем

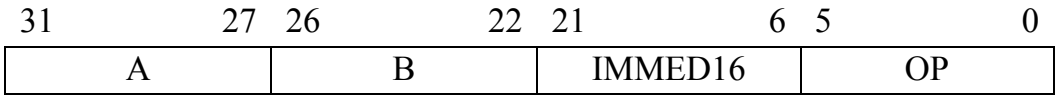
Процессор содержит аппаратную обработку исключений, включая аппаратные прерывания. Также он может иметь дополнительный интерфейс с внешним контроллером прерываний (EIC), чтобы ускорить обработку прерываний в комплексной системе.

В состав процессорного ядра также могут входить и специализированные модули, например, JTAG UART, который служит для обмена информацией с компьютером, и JTAG Debug, необходимый для выполнения отладки программного обеспечения с помощью компьютера.

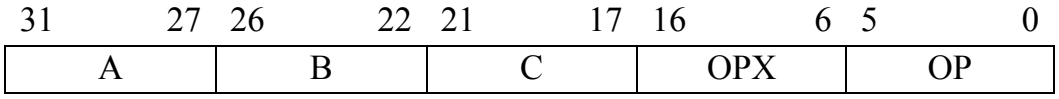
2.2. Система команд процессора Nios II

Исполнимые команды процессора Nios II кодируются 32-разрядными словами. Используются 3 различных формата команд:

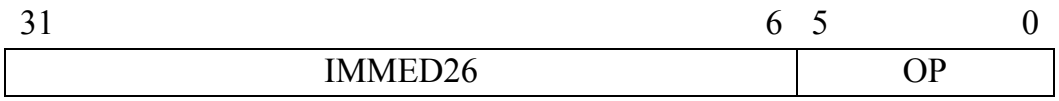
- **I-тип**. Команда содержит поля A и B шириной по 5 бит, использующиеся для определения регистров, поле IMMED16, используемое для указания непосредственных операндов (при необходимости может быть расширено до 32 бит), и поле кода операции (КОП, OP) шириной 6 бит;



- **R-тип**. Команда содержит поля A, B и C шириной по 5 бит, использующиеся для определения регистров, поле OPX для расширения кода операций, и поле кода операции (КОП, OP) шириной 6 бит;



- **J-тип**. Команда содержит поле IMMED26, используемое для указания непосредственных операндов, и поле кода операции (КОП, OP) шириной 6 бит.



2.3. Список команд процессора Nios II

Ниже приведен неполный список команд ассемблера процессора Nios II.

Команды load, store

Для обмена информацией между регистрами общего назначения процессора и оперативной памятью определены несколько команд.

ldw *rB, byte_offset (rA)* – загрузка слова из оперативной памяти в регистр rB, адрес операнда в оперативной памяти определяется сложением содержимого регистра rA и смещения byte_offset;

stw *rB, byte_offset (rA)* – сохранение слова из регистра rB в оперативную память по адресу rA + смещение byte offset.

Арифметические команды

add *rC, rA, rB* и ***addi*** *rB, rA, IMMED* – сложение; в первом случае оба операнда (rA, rB) расположены в регистрах общего назначения, во втором случае один из операндов (IMMED) представлен числом, результат вычисления в обоих случаях помещается в РОН (rC);

sub *rC, rA, rB* – вычитание; команда вычитания константы заменяется операцией сложения с отрицательной константой (–IMMED);

mul *rC, rA, rB* – умножение;

mulxss *rC, rA, rB* – умножение двух знаковых чисел;

mulxsu *rC, rA, rB* – умножение знакового и беззнакового чисел;

mulxuu *rC, rA, rB* – умножение двух беззнаковых чисел;

Логические команды

Следующие пары команд предназначены для выполнения логических операций.

and *rC, rA, rB* и ***andi*** *rD, rA, IMMED16* – побитовое логическое И;

or *rC, rA, rB* и ***ori*** *rB, rA, IMMED16* – побитовое логическое ИЛИ;

xor *rC, rA, rB* и ***xori*** *rB, rA, IMMED16* – побитовое сложение по модулю 2 (исключающее ИЛИ);

nor *rC, rA, rB* – побитовое логическое И с отрицанием.

Дополнительные три команды ***andhi***, ***orhi***, ***xorhi*** выполняют операции со старшей половиной слова в регистре. Младшая часть непосредственного операнда при этом дополняется нулями до полной ширины слова.

Команды сдвига

Команды логического сдвига выполняют сдвиг содержимого регистра rA влево или вправо на количество разрядов, заданных либо пятью младшими разрядами регистра rB, либо непосредственным операндом IMMED5. Освободившиеся разряды заполняются нулями. Результат заносится в регистр rC.

sll *rC, rA, rB* и ***slli*** *rC, rA, IMMED5* – логический сдвиг влево;

srl *rC, rA, rB* и ***srli*** *rC, rA, IMMED5* – логический сдвиг вправо;

Команды арифметического сдвига выполняют сдвиг содержимого регистра rA вправо на количество разрядов, заданных либо пятью младшими разрядами регистра rB, либо непосредственным операндом IMMED5. Освободившиеся разряды заполняются знаковым битом. Результат заносится в регистр rC.

sra *rC, rA, rB* и ***srai*** *rC, rA, IMMED5* – арифметический сдвиг вправо.

Команды циклического сдвига выполняют сдвиг содержимого регистра rA влево или вправо на количество разрядов, заданных либо пятью младшими разрядами регистра rB, либо непосредственным операндом IMMED5. Освободившиеся разряды заполняются вытесненными битами. Результат заносится в регистр rC.

ror *rC, rA, rB* – циклический сдвиг вправо;

rol *rC, rA, rB* и ***roli*** *rC, rA, IMMED5* – циклический сдвиг влево;

Команды пересылки

Для копирования данных из одного регистра в другой либо записи константы в РОН можно применить команды сложения add, указав в качестве второго операнда регистр нуля, либо addi, указав нулевую константу.

Команды сравнения

Группа команд сравнения предназначена для сравнения содержимого регистров.

cmplt *rC, rA, rB* – запись в регистр rC единицы, если rA < rB (с учетом знака), или нуля в противном случае;

cmpltu *rC, rA, rB* – запись в регистр rC единицы, если rA < rB (без учета знака), или нуля в противном случае;

cmpeq *rC, rA, rB* – запись в регистр rC единицы, если rA = rB, или нуля в противном случае;

cmpne *rC, rA, rB* – запись в регистр rC единицы, если rA ≠ rB, или нуля в противном случае;

cmpge *rC, rA, rB* – запись в регистр rC единицы, если rA ≥ rB (с учетом знака), или нуля в противном случае;

cmpgeu *rC, rA, rB* – запись в регистр rC единицы, если rA ≥ rB (без учета знака), или нуля в противном случае.

Команды сравнения с непосредственным операндом

Следующая группа команд предназначена для сравнения содержимого регистра с непосредственным операндом.

cmplti *rB, rA, IMMED16* – запись в регистр rC единицы, если rA < IMMED16 (с учетом знака), или нуля в противном случае;

cmpltui *rB, rA, IMMED16* – запись в регистр rC единицы, если rA < IMMED16 (без учета знака), или нуля в противном случае;

cmpeqi *rB, rA, IMMED16* – запись в регистр rC единицы, если rA = IMMED16, или нуля в противном случае;

cmpnei *rB, rA, IMMED16* – запись в регистр rC единицы, если rA ≠ IMMED16, или нуля в противном случае;

cmpgei *rB, rA, IMMED16* – запись в регистр rC единицы, если rA ≥ IMMED16 (с учетом знака), или нуля в противном случае;

cmpgeui *rB, rA, IMMED16* – запись в регистр rC единицы, если rA ≥ IMMED16 (без учета знака), или нуля в противном случае.

Команды переходов

Команды переходов применяются для ветвления программы.

jmp *rA* – безусловный переход по адресу в rA;

jmpi *Label* – безусловный переход по адресу Label, адрес перехода вычисляется как $PC_{31..28}$: IMMED26 x 4;

br *Label* – безусловный переход по адресу Label, адрес перехода вычисляется как PC + 4 + σ(IMM16);

blt *rA, rB, Label* и ***bltu*** *rA, rB, Label* – команды перехода по адресу Label, если rA < rB, с учетом и без учета знака соответственно;

bge *rA, rB, Label* и ***bgeu*** *rA, rB, Label* – команды перехода по адресу Label, если rA ≥ rB, с учетом и без учета знака соответственно;

beq *rA, rB, Label* – переход по адресу Label, если rA = rB.

bne *rA, rB, Label* – переход по адресу Label, если rA ≠ rB.

Команды вызова подпрограммы и возврата из нее

call *Label* – адрес следующей команды сохраняется в регистре ra и выполняется переход, адрес перехода вычисляется как $PC_{31..28}$: IMMED26 x 4.

callr *rA* – адрес следующей команды сохраняется в регистре ra и выполняется переход по адресу, хранящемуся в регистре rA.

ret –возврат из подпрограммы, выполняется переход по адресу, хранящемуся в регистре rA.

Команды управления

Запись в регистры управления процессора Nios II или чтение из них выполняются при помощи следующих команд:

rdctl *rC, ctlN* – копирование содержимого регистра ctlN в rC;

wrctl *ctlN, rA* – копирование содержимого регистра rA в ctlN.

3. Задание к работе

1. Составить блок-схему алгоритма программы в соответствии с вариантом задания.

2. Запустить программу Nios2Sim и ввести код программы на языке ассемблера. Исходный массив объявить в памяти программы. Результат выводить в оперативную память.

3. Запустить код на исполнение (меню Nios II → Start Simulation) и выполнить пошаговое выполнение программы (меню

Nios II → Execute a Step), наблюдая за состоянием памяти или регистров.

4. Содержание отчета

1. Цель работы.
2. Задание к работе схемы.
3. Блок-схема алгоритма программы.
4. Исходный код программы.
5. Выводы по работе.

5. Варианты заданий

1. В массиве все четные элементы обнулить.
2. В массиве все нечетные элементы заменить на цифру 1.
3. В массиве все элементы, стоящие после нечетных, обнулить.
4. В массиве все элементы, стоящие перед четными, заменить на цифру 9.
5. В массиве все элементы, стоящие между перед минимальным, заменить на 0.
6. В массиве все элементы, стоящие после минимального, заменить на цифру 7.
7. В массиве все элементы, стоящие перед максимальным, заменить на 0.
8. В массиве все элементы, стоящие после максимального, заменить на 0.
9. В массиве все нечетные элементы, стоящие после максимального, заменить на 0.
10. В массиве все четные элементы, стоящие перед минимальным, заменить на 0.
11. В массиве все нечетные элементы по модулю большие 5, и стоящие после максимального, заменить на 0.
12. В массиве все четные элементы по модулю большие 4, и стоящие перед минимальным, заменить на 0.
13. В массиве все элементы, кратные 2 и 3, обнулить.
14. Отсортировать массив по убыванию.

15. Отсортировать массив по возрастанию.

Массив должен иметь размерность больше чем 1×20 и быть заполнен произвольными числами.

6. Контрольные вопросы

1. Что такое Nios II? Что такое система с процессором Nios II?

2. Какие существуют конфигурации процессора Nios II? В чем их различие?

3. Что такое регистровый файл? Поясните его структуру и назначение.

4. Назовите виды адресации процессора Nios II.

5. Назовите форматы команд процессора Nios II с примерами.

6. Какими командами языка ассемблера осуществляется обмен информацией между регистрами и памятью?

7. Как на языке ассемблера организуются циклы?

8. Как на языке ассемблера организуются условные конструкции?

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. Харрис, Д. М. Цифровая схемотехника и архитектура компьютера (перевод) / Д. М. Харрис, С. Л. Харрис. – Нью Йорк : Издательство Morgan Kaufman, 2015. – 1662 с.

2. Бойт, К. Цифровая электроника / К. Бойт. – Москва : Техносфера, 2007. – 472 с.

3. Ефремов, Н. В. Введение в систему автоматизированного проектирования Quartus II: учебное пособие / Н. В. Ефремов. – Москва : ГОУ ВПО МГУЛ, 2011. – 147 с.

4. Бабак, В. П. VHDL. Справочное пособие по основам языка / В. П. Бабак, А. Г. Корченко, Н. П. Тимошенко, С. Ф. Филоненко. – Москва : Издательский дом «Додэка-XXI», 2008. – 224 с.

5. Соловьев, В. В. Основы языка проектирования цифровой аппаратуры Verilog / В. В. Соловьев. – Москва : Горячая линия – Телеком, 2014. – 206 с.

6. Нарышкин, А. К. Цифровые устройства и микропроцессоры: учебное пособие для вузов / А. К. Нарышкин. – Москва : Академия, 2008.

7. Nios II Classic Processor Reference Guide [Электронный ресурс] // Intel FPGA and SoC, 2017. URL: https://www.altera.com/en_US/pdfs/literature/hb/nios2/n2cpu_nii5v1.pdf (дата обращения: 26.09.2017).

ПРИЛОЖЕНИЕ А
ИДЕНТИФИКАТОРЫ ЯЗЫКОВ VHDL И VERILOG

Идентификаторы – определяемые пользователем слова, используемые для обозначения программных объектов (проектов, интерфейсов, архитектур, пакетов, переменных, компонентов, сигналов, констант и др.).

В языке VHDL при выборе идентификаторов следует соблюдать следующие правила:

- Идентификаторы не должны совпадать с ключевыми словами;
- Идентификатор может содержать прописные и (или) строчные символы *латинского* алфавита и цифры, разделенные *одним* символом подчеркивания;
- Первым символом в идентификаторе должен быть символ латинского алфавита. Символ подчеркивания не может быть первым или последним символом идентификатора;
- Использовать подряд два символа подчеркивания в идентификаторе запрещается;
- Строчные и прописные символы латинского алфавита в идентификаторе считаются одинаковыми, т. е. идентификаторы «And2», «AND2» или «and2» равнозначны;
- Длина идентификатора по стандарту не ограничена, но может ограничиваться в конкретном компиляторе.

Для языка Verilog существует еще несколько ограничений:

- Системные имена начинаются с символа ($);
- Директивы компилятора начинаются с символа ' (апостроф);
- Компилятор различает строчные и прописные буквы.

Примеры корректных идентификаторов: X10, x_10, My_gate1.
Примеры некорректных индикаторов: _X10, gate@inp, 2gate-in.

ПРИЛОЖЕНИЕ Б
ПОДАВЛЕНИЕ ДРЕБЕЗГА КОНТАКТОВ

В учебном стенде Altera DE2-115 для ввода данных используются кнопки и переключатели. Если переключатели, как правило, используются для задания статического уровня, то кнопка используется в качестве генератора коротких импульсов. В этом случае схема должна реагировать не на статический уровень, а на факт нажатия, то есть на переход сигнала из одного логического состояния на выводе ПЛИС в другое. При использовании механических коммутаторов (кнопки, тумблеры и прочие ключи) всегда возникает такое явление, как дребезг контактов.

Дребезг контактов – явление, возникающее в электрических и электронных переключателях, при котором они вместо некоторого стабильного переключения производят случайные многократные неконтролируемые замыкания и размыкания контактов (происходит в момент переключения) (см. рис. В.1). Иными словами – это явление, вызванное неизбежным несовершенством технологии изготовления переключателей.

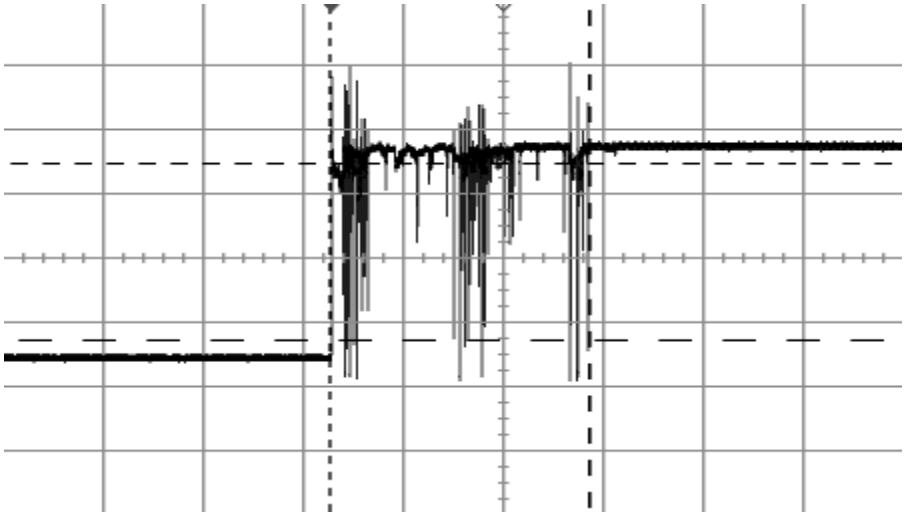


Рисунок В.1 – Дребезг контактов на экране осциллографа

Ниже приведен листинг кода универсального модуля обработки сигнала с механических переключателей для подавления эффекта дребезга контактов. В иностранной литературе аналогичное устройст-

во или блок называется «debouncer». Модуль написан на языке Verilog.

```
module button_debouncer
#(
	parameter CNT_WIDTH = 16
)
(
	input clk_i,
	input rst_i,
	input sw_i,
	output reg sw_state_o,
	output reg sw_down_o,
	output reg sw_up_o
);

reg [1:0] sw_r;
always @(posedge rst_i or posedge clk_i)
	if (rst_i)
		sw_r <= 2'b00;
	else
		sw_r <= {sw_r[0], ~sw_i};

reg [CNT_WIDTH-1:0] sw_count;

wire sw_change_f = (sw_state_o != sw_r[1]);
wire sw_cnt_max = &sw_count;

always @(posedge rst_i or posedge clk_i)
	if (rst_i)
	begin
		sw_count <= 0;
		sw_state_o <= 0;
	end
	else if (sw_change_f)
	begin
		sw_count <= sw_count + 'd1;
		if(sw_cnt_max) sw_state_o <= ~sw_state_o;
	end
	else sw_count <= 0;

always @(posedge clk_i)
begin
	sw_down_o <= sw_change_f & sw_cnt_max & ~sw_state_o;
	sw_up_o <= sw_change_f & sw_cnt_max & sw_state_o;
end

endmodule
```

Для ввода сигнала непосредственно с кнопки имеется входной порт *sw_i*. Вход для системных тактовых импульсов, с которыми синхронно работает вся остальная схема – *clk_i*. Вход для асинхронного сброса всей схемы – *rst_i*. В целях универсальности предусмотрено два раздельных выхода для события «кнопка нажата» и события «кнопка отпущена» – *sw_down_o* и *sw_up_o*. Выход *sw_state_o* служит для отражения состояния кнопки (длительности нажатия), естественно, после устранения эффекта дребезга.

Пример подключения схемы подавления дребезга контактов кнопки показан на рис. В.2. Сигнал с кнопки должен быть пропущен через модуль (подается на вход *BUTTON*), вместо того, чтобы идти напрямую к исследуемой схеме. Кроме этого, на модуль должен подаваться тактовый сигнал с генератора (вход *G_50MHz*).

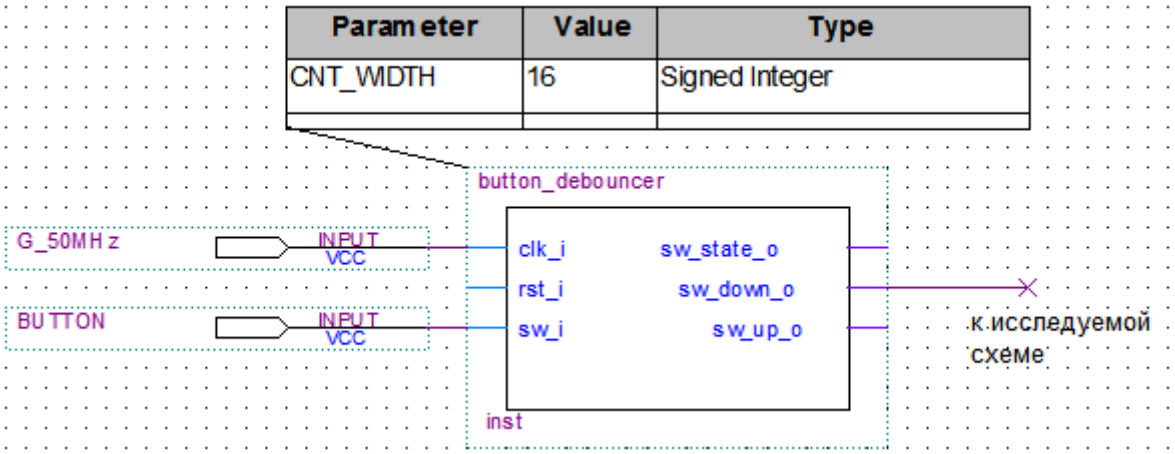


Рисунок В.2 – Пример подключения схемы подавления дребезга контактов

ПРИЛОЖЕНИЕ В
ОПИСАНИЕ МИКРОСХЕМЫ SN74193 (К1533ИЕ7)

Микросхема SN74193 (КР1533ИЕ7) представляет собой двоичный четырехразрядный реверсивный счетчик синхронного типа.

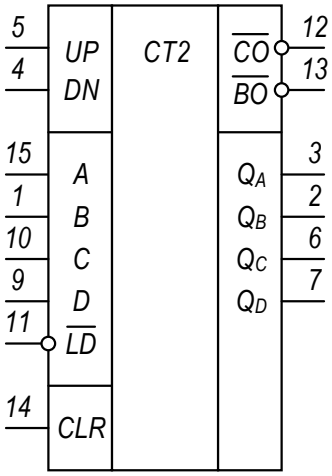


Рисунок В.1 – Условно-графическое обозначение микросхемы SN74193 (КР1533ИЕ7)

Таблица В.1

Назначение выводов микросхемы SN74193 (КР1533ИЕ7)

Номер вывода	**Имя вывода**	**Функция вывода**
1	B	Вход информационный B
2	Q_B	Выход B
3	Q_A	Выход A
4	DN	Вход «Обратный счет»
5	UP	Вход «Прямой счет»
6	Q_C	Выход C
7	Q_D	Выход D
8	GND	Общий вывод
9	D	Вход информационный D
10	C	Вход информационный C
11	$\overline{LD}$	Вход стробирования предварительной записи
12	$\overline{CO}$	Выход «Прямой перенос»
13	$\overline{BO}$	Выход «Обратный перенос»
14	CLR	Вход сброса
15	A	Вход информационный A
16	VCC	Вывод питания

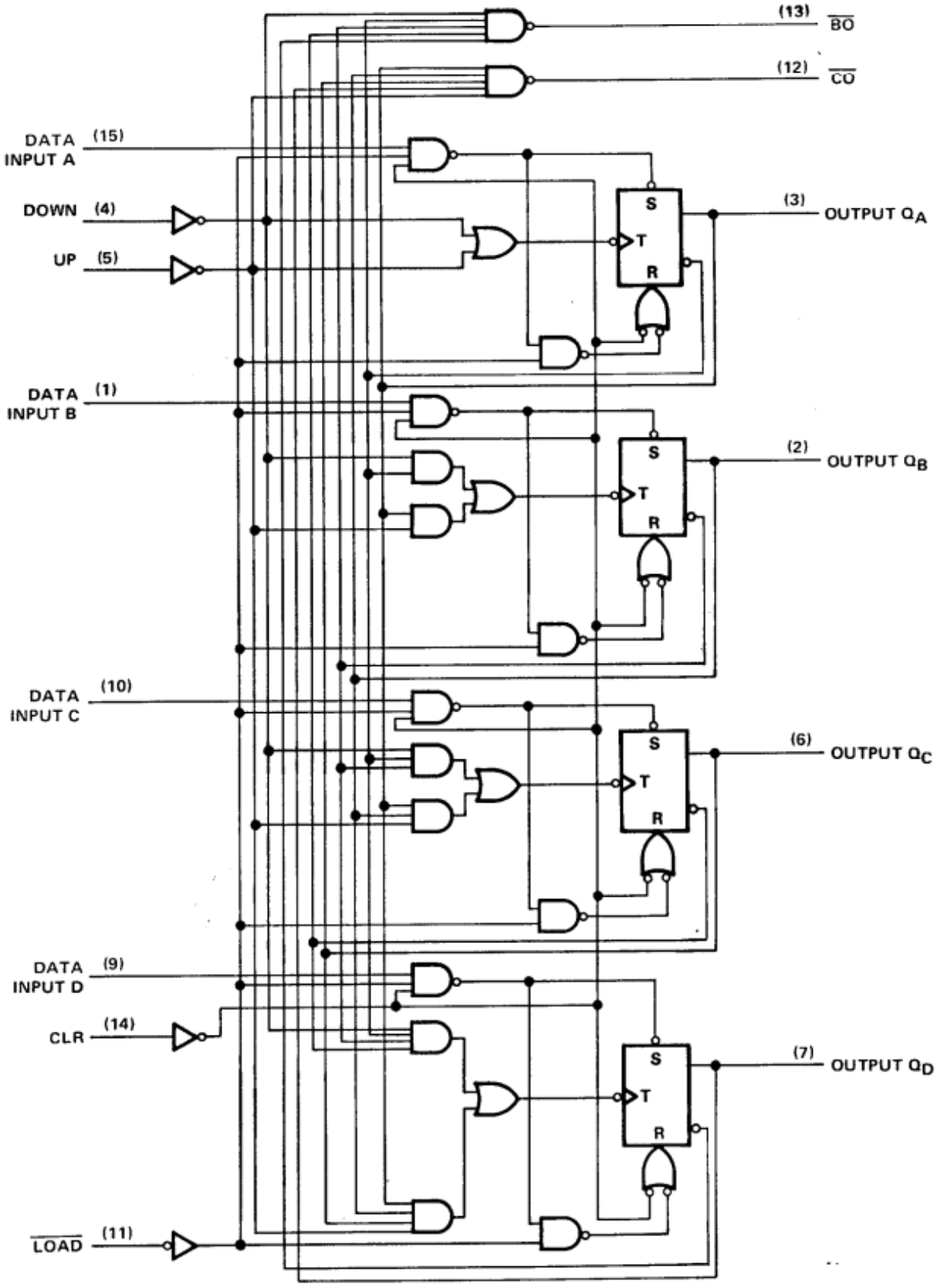


Рисунок В.2 – Функциональная схема микросхемы SN74192 (КР1533ИЕ7)

Положительный импульс напряжения по входу *CLR* устанавливает выходы счетчика в исходное состояние – уровень «логического 0» на счетных выходах. Для предварительной установки счетчика в определенное состояние необходимо на информационные входы подать соответствующие уровни, а на вход стробирования предварительной записи подать отрицательный импульс напряжения. Для осуществления прямого счета на вход *DN* подается высокий уровень напряжения, а на вход прямого счета *UP* – положительные импульсы. Счет будет вестись от того числа, которое было предварительно записано в счетчик. После заполнения счетчика выходы устанавливаются в состояние высокого уровня, а на выходе прямого переноса появится отрицательный импульс переноса счета в старший разряд. Аналогично счетчик работает в режиме обратного счета.

Таблица В.2

Режимы работы микросхемы SN74193 (КР1533ИЕ7)

Режим работы	**Вход**			
	CLR	$\overline{\text{LD}}$	UP	DN
Сброс выходов в лог. 0	1	X	X	X
Запись информации	0	0	X	X
Неактивное состояние	0	1	1	1
Счет прямой	0	1	⌐¬	1
Счет обратный	0	1	1	⌐¬

ПРИЛОЖЕНИЕ Г
ПРИМЕР ОФОРМЛЕНИЯ ТИТУЛЬНОГО ЛИСТА ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ

Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
Радиотехнический факультет
Кафедра «Радиотехника»

ОТЧЕТ
по лабораторной работе №1
«НАЗВАНИЕ РАБОТЫ»
по дисциплине «Цифровые устройства и микропроцессоры»

Выполнили:
студенты гр. РТбд-31

_____ / И.В. Иванов
_____ / А.И. Петров

Проверил:
_____ / М.Г. Царёв

Ульяновск, 2017

Учебное издание
Проектирование цифровых устройств на ПЛИС
Лабораторный практикум

Составитель ЦАРЁВ Михаил Григорьевич

Редактор Н.А. Евдокимова

Подписано в печать 29.12.2017. Формат 60×84/16.
Усл. печ. л. 4,65. Тираж 50 экз. Заказ 419. ЭИ №1078.

Ульяновский государственный технический университет
432027, г. Ульяновск, ул. Сев. Венец, д. 32.
ИПК «Венец» УлГТУ, 432027, г. Ульяновск, ул. Сев. Венец, д. 32.