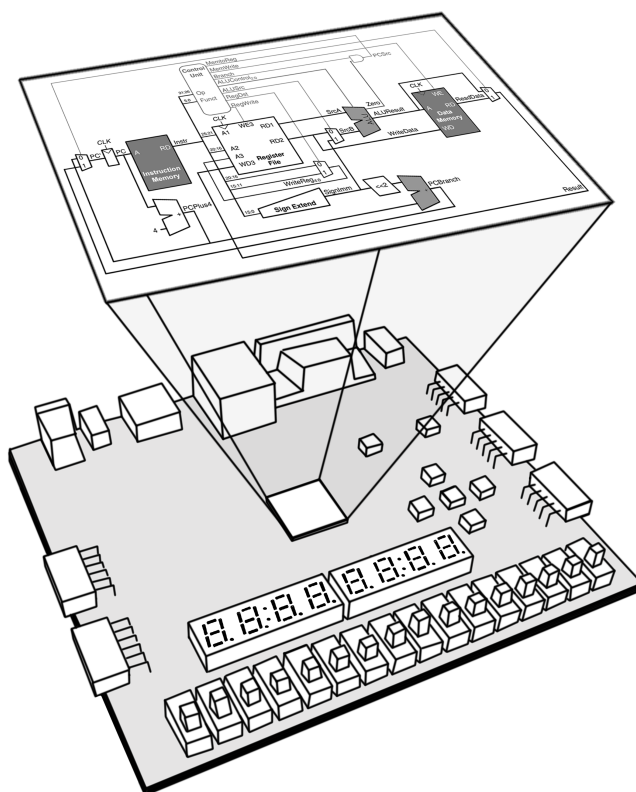


А. А. Антонов, С. В. Быковский, П. В. Кустарев,  
К. А. Кормилицын, В. Ю. Пинкевич

# ФУНКЦИОНАЛЬНАЯ СХЕМОТЕХНИКА

## Практикум



Санкт-Петербург

2019

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ  
ФЕДЕРАЦИИ

УНИВЕРСИТЕТ ИТМО

А. А. Антонов, С. В. Быковский, П. В. Кустарев,  
К. А. Кормилицын, В. Ю. Пинкевич

# ФУНКЦИОНАЛЬНАЯ СХЕМОТЕХНИКА

## Практикум

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

РЕКОМЕНДОВАНО К ИСПОЛЬЗОВАНИЮ В УНИВЕРСИТЕТЕ ИТМО  
по направлению подготовки 09.03.01 «Информатика и вычислительная техника»  
в качестве практикума для реализации основных профессиональных  
образовательных программ высшего образования бакалавриата



Санкт-Петербург

2019

Антонов А.А., Быковский С. В., Кустарев П. В., Кормилицын К. А., Пинкевич В. Ю. Функциональная схемотехника. Практикум. — СПб.: Университет ИТМО, 2019. — 97 с.

**Рецензент:** Поляков В.И., кандидат технических наук, доцент, доцент (квалификационная категория "ординарный доцент") факультета программной инженерии и компьютерной техники, Университета ИТМО.

В учебном пособии представлены теоретические сведения и задания к лабораторным работам по дисциплине «Функциональная схемотехника». Задания лабораторных работ направлены на получение знаний и развитие навыков проектирования цифровых интегральных схем на уровне базовых элементов КМОП-логики — полевых транзисторов, а также на уровне регистровых передач (RTL-уровне) с использованием языка описания аппаратуры Verilog HDL.

В рамках лабораторных работ студенты учатся проектировать аппаратные блоки ускорителей математических вычислений, встроенные схемы самотестирования, контроллеры ввода/вывода, индикации и периферийных устройств, анализировать цифровые схемы с использованием SPICE-моделирования и проводить прототипирование схем на базе микросхем программируемой логики (ПЛИС).



**Университет ИТМО** — ведущий вуз России в области информационных и фотонных технологий, один из немногих российских вузов, получивших в 2009 году статус национального исследовательского университета. С 2013 года Университет ИТМО — участник программы повышения конкурентоспособности российских университетов среди ведущих мировых научно-образовательных центров, известной как проект «5 в 100». Цель Университета ИТМО — становление исследовательского университета мирового уровня, предпринимательского по типу, ориентированного на интернационализацию всех направлений деятельности.

© Университет ИТМО, 2019

© Антонов А.А., Быковский С. В., Кустарев П. В., Кормилицын К. А.,  
Пинкевич В. Ю., 2019

# Содержание

Введение . . . . .	5
<b>1 Лабораторная работа №1. Введение в проектирование цифровых интегральных схем . . . . .</b>	<b>7</b>
1.1 Введение . . . . .	7
1.2 Описание лабораторной работы . . . . .	7
1.3 Основы работы в среде LTspice . . . . .	11
1.3.1 О моделировании в SPICE . . . . .	11
1.3.2 Ограничения области применимости SPICE . . . . .	11
1.3.3 О программе LTspice . . . . .	12
1.3.4 Создание инвертора и моделирование его работы . . . . .	12
1.3.5 Создание иерархических элементов . . . . .	16
1.4 Основы работы в среде Vivado Design Suite . . . . .	19
1.4.1 Схема сумматора . . . . .	19
1.4.2 Описание схемы сумматора на вентиляльном уровне с использованием языка Verilog HDL . . . . .	20
1.4.3 Создание проекта в Vivado Design Suite . . . . .	24
1.4.4 Моделирование схемы . . . . .	31
<b>2 Лабораторная работа №2. Разработка аппаратных ускорителей математических вычислений . . . . .</b>	<b>35</b>
2.1 Введение . . . . .	35
2.2 Описание лабораторной работы . . . . .	35
2.3 Разработка цифровых схем конечных автоматов . . . . .	38
2.4 Алгоритмы машинной арифметики . . . . .	45
2.4.1 Алгоритм умножения . . . . .	45
2.4.2 Алгоритм извлечения квадратного корня . . . . .	49
2.4.3 Алгоритм извлечения кубического корня . . . . .	51
<b>3 Лабораторная работа №3. Проектирование цифровых схем с использованием ПЛИС . . . . .</b>	<b>52</b>

3.1	Введение . . . . .	52
3.2	Описание лабораторной работы . . . . .	52
3.3	Архитектура и принцип работы микросхем программируемой логики . . . . .	54
3.4	Пример создания и загрузки файла конфигурации ПЛИС в среде Vivado Design Suite . . . . .	60
<b>4</b>	<b>Лабораторная работа №4. Проектирование встроенных схем само- тестирования . . . . .</b>	<b>70</b>
4.1	Введение . . . . .	70
4.2	Описание лабораторной работы . . . . .	70
4.3	Устройство и принцип работы регистров LFSR . . . . .	74
4.4	Алгоритм расчета кода CRC8 . . . . .	77
4.5	Дребезг контактов и вариант блока защиты для его устранения . . . . .	79
<b>5</b>	<b>Лабораторная работа №5. Проектирование контроллеров ввода и индикации . . . . .</b>	<b>83</b>
5.1	Введение . . . . .	83
5.2	Описание лабораторной работы . . . . .	83
<b>6</b>	<b>Лабораторная работа №6. Основы проектирования контроллеров периферийных устройств . . . . .</b>	<b>89</b>
6.1	Введение . . . . .	89
6.2	Описание лабораторной работы . . . . .	89
<b>A</b>	<b>Требования к оформлению отчетов по лабораторным работам . . . . .</b>	<b>94</b>
	<b>Список рекомендуемой литературы . . . . .</b>	<b>95</b>

# Введение

Настоящее методическое пособие-практикум содержит описание лабораторных работ, направленных на приобретение навыков проектирования цифровых схем с использованием полупроводниковой элементной базы. Лабораторные работы содержат практические задания как по проектированию схем на низком уровне, на базе полевых транзисторов и КМОП-логики, так и на более высоких уровнях: на вентиляном и уровне регистровых передач (RTL-уровне). В рамках работ также предлагается ознакомиться с проектированием цифровых схем с использованием микросхем программируемой логики (ПЛИС) и получить базовые навыки тестирования и отладки схем на реальной аппаратной платформе с ПЛИС. В практикуме даются рекомендации по работе с инструментальным обеспечением фирмы Xilinx и отладочной платой Nexys 4 DDR (новое название — Nexys A7).

Лабораторные работы охватывают следующие темы:

1. Лабораторная работа №1 «Введение в проектирование цифровых интегральных схем».
2. Лабораторная работа №2 «Разработка аппаратных ускорителей математических вычислений».
3. Лабораторная работа №3 «Проектирование цифровых схем с использованием ПЛИС».
4. Лабораторная работа №4 «Проектирование встроенных схем самотестирования».
5. Лабораторная работа №5 «Проектирование контроллеров ввода и индикации».
6. Лабораторная работа №6 «Проектирование контроллеров периферийных устройств».

В качестве базового набора лабораторных работ на семестр предлагается использовать лабораторные работы под номерами 1, 2, 3, 4. В качестве усложненного варианта может быть предложено выполнение комплекса из лабораторных работ под номерами 1, 5, 6.

В практикуме приводятся описания заданий, требования к отчетам, а также краткие теоретические сведения по выполнению заданий и работе с инструментальным программным обеспечением.

При выполнении лабораторных работ не по порядку, для лучшего понимания требований и рекомендуемых способов выполнения работы, следует обращаться к материалам предыдущих работ.

Данное методическое пособие рекомендовано для проведения лабораторных занятий студентам Университета ИТМО по дисциплине «Функциональная схемотехника» в рамках направлений подготовки 09.03.01 «Информатика и вычислительная техника» и 09.03.04 «Программная инженерия».

Для выполнения лабораторных работ практикума рекомендуется использовать следующее инструментальное обеспечение:

1. LTspice (версия IV или новее).
2. Vivado Design Suite (версия 2017.4 или новее).

# 1 Лабораторная работа №1. Введение в проектирование цифровых интегральных схем

## 1.1 Введение

Лабораторная работа №1 посвящена развитию навыков проектирования схем на уровне транзисторов и с использованием вентиляей, рассматриваются базовые принципы построения схем на КМОП-логике.

Для исследования работы электрических схем инженер-проектировщик может создать прототип устройства на макетной плате или воспользоваться ручными расчётами на бумаге. Однако при разработке интегральной микросхемы создание прототипа может быть крайне дорогостоящим и длительным процессом. Сложность таких схем зачастую не позволяет провести требуемые расчеты вручную. Поэтому при разработке большинства микросхем, а также сложных печатных плат повсеместно применяется компьютерное моделирование. Наиболее известной системой аналогового моделирования является программа SPICE и её многочисленные наследники. Часть заданий лабораторной работы будет выполняться в среде моделирования LTspice, которая относится к классу SPICE-подобных симуляторов электронных схем общего назначения.

В рамках работы также производится знакомство с принципами проектирования схем на вентиляльном уровне с использованием языка описания аппаратуры Verilog HDL.

## 1.2 Описание лабораторной работы

### Цели работы

1. Получить базовые знания о принципах построения цифровых интегральных схем с использованием технологии КМОП.
2. Познакомиться с технологией SPICE-моделирования схем на транзисторах.
3. Получить навыки описания схем базовых операционных элементов (БОЭ) комбинационного типа на вентиляльном уровне с использованием языка описания аппаратуры Verilog HDL.



## Указания к выполнению работы

Лабораторная работа состоит из двух частей.

Первая часть посвящена проектированию цифровых вентилях на полевых транзисторах, построению схем на базе вентилях и знакомству с технологией SPICE-моделирования. Первая часть работы выполняется в программном пакете LTspice. При построении схем вентилях необходимо использовать КМОП-транзисторы с параметрами из файла, предоставленного преподавателем (см. раздел «Основы работы в среде LTspice»).

Вторая часть посвящена знакомству с языком описания аппаратуры Verilog HDL, изучению особенностей его использования для описания схем на вентилях уровне и приобретению навыков тестирования таких схем. Вторая часть работы выполняется с использованием Vivado Simulator, входящего в пакет Vivado Design Suite (см. раздел «Основы работы в среде Vivado Design Suite»).

## Порядок выполнения работы

### Часть 1:

1. Постройте в LTspice на транзисторах схему вентиля, составляющего основу логического базиса согласно варианту задания.
2. Создайте символ для разработанного вентиля как иерархического элемента.
3. С использованием созданного иерархического элемента постройте схему тестирования вентиля.
4. Проведите моделирование работы схемы и определите задержку распространения сигнала через тестируемый вентиль.
5. Определите максимальную частоту изменения входных сигналов, при которой построенная схема сохраняет работоспособность.
6. Постройте БОЭ на базе созданного вентиля согласно варианту задания.
7. Создайте символ для построенного БОЭ.
8. Проведите моделирование работы схемы и определите задержку распространения сигнала через БОЭ.
9. Определите максимальную частоту изменения входных сигналов, при которой построенная схема сохраняет работоспособность.
10. Составьте отчет по результатам выполнения заданий первой части лабораторной работы.

## **Часть 2:**

1. Опишите на Verilog HDL на вентиляльном уровне модуль, реализующий функцию БОЭ в указанном логическом базисе согласно варианту задания.
2. Разработайте тестовое окружение для созданного модуля.
3. Проведите моделирование работы схемы.
4. Составьте отчет по результатам выполнения заданий второй части лабораторной работы.

## **Требования к отчету**

Отчет должен быть оформлен в соответствии с требованиями, представленными в Приложении А.

### **Отчет должен включать:**

1. Титульный лист.
2. Цели работы.
3. Задание в соответствии с вариантом.
4. Отчет о выполнении заданий части 1:
  - схема разработанного вентиля;
  - символ вентиля и схема тестирования;
  - временная диаграмма процесса тестирования вентиля;
  - результат измерения задержки распространения сигнала через вентиль;
  - максимальная частота работы вентиля;
  - схема разработанного БОЭ;
  - символ разработанного БОЭ и схема тестирования;
  - временная диаграмма процесса тестирования БОЭ;
  - результат измерения задержки распространения сигнала через БОЭ;
  - максимальная частота работы БОЭ.
5. Отчет о выполнении заданий части 2:
  - код разработанного модуля БОЭ;
  - код разработанного тестового окружения БОЭ;
  - временная диаграмма процесса тестирования БОЭ.
6. Выводы по работе.

## Задания по вариантам

№ варианта	Логический базис	БОЭ
1	NOR	Демультимплексор «1 в 4»
2	NAND	Полный четырехразрядный компаратор
3	NOR	Схема мажоритарного контроля с 5-ю входами
4	NAND	Позиционный шифратор «8 в 3»
5	NOR	Преобразователь BCD-кода в двоичный код (числа от 0 до 99)
6	NAND	Позиционный дешифратор «3 в 8»
7	NOR	Четырехразрядный двоичный сумматор с переносом
8	NAND	Шифратор кода Грея для трехразрядного двоичного числа
9	NOR	Мультимплексор «4 в 1»
10	NAND	Дешифратор трехразрядного кода Грея

## 1.3 Основы работы в среде LTspice

### 1.3.1 О моделировании в SPICE

SPICE (Simulation Program with Integrated Circuit Emphasis) – программа для моделирования поведения интегральных схем и схем на печатных платах. Изначально разработана в Калифорнийском университете в Беркли в начале 1970-х гг. В дальнейшем на ее основе были созданы другие программы, одной из которых является LTspice.

Необходимость моделирования вызвана тем, что создать дешевый прототип интегральной схемы практически невозможно – для этого ее надо фактически изготовить, что является очень затратным мероприятием. SPICE позволяет промоделировать работу схемы в разных условиях и выявить потенциальные проблемы до того, как она будет изготовлена.

SPICE содержит модели различных электронных компонентов и элементов схем – транзисторов, резисторов, конденсаторов, источников напряжения, усилителей и т.п. Соединяя компоненты, можно построить произвольную схему, а затем подвергнуть ее анализу одним из поддерживаемых способов: передаточная функция, анализ по постоянному току, анализ по переменному току и др.

### 1.3.2 Ограничения области применимости SPICE

Несмотря на то, что SPICE во многих случаях может заменить прототипирование, часто результаты моделирования могут оказаться недостаточно точными или даже совсем не совпадать с работой реальной схемы. Реальные электронные компоненты являются сложными устройствами, работа которых зависит от множества параметров. Результаты моделирования точны настолько, насколько точны модели компонентов.

Модели схем в SPICE свободны от шумов, перекрестных помех, паразитных емкостей и др., если только они явно не внесены в модель проектировщиком. Например, сигнал силой в несколько пикоампер может корректно управлять схемой в SPICE-модели, но в физическом мире будет полностью подавлен шумами и паразитными цепями.

SPICE-моделирование не подходит для исследования излучающих и принимающих схем (антенн). Для задач такого рода используются системы, построенные на численном решении уравнений Максвелла.

SPICE не является лучшим средством для предсказания поломки компонентов. Моделирование не выдаст никакого предупреждения в случае, если будет превышено

максимально допустимое значение тока или напряжения для какого-либо из компонентов. Например, проектировщик может не заметить тока в несколько ампер на одной из цепей при моделировании схемы, но после изготовления такая схема при включении сгорит.

### 1.3.3 О программе LTspice

Загрузить программу LTspice можно с сайта компании Analog Devices по адресу: <https://www.analog.com/en/design-center/design-tools-and-calculators/ltspice-simulator.html>. Программа является бесплатной. Доступны версии под Windows и Mac OS, под Linux LTspice можно запустить в Wine. Версия для Mac OS имеет некоторые отличия от версии для Windows.

LTspice имеет встроенную справку с описанием основных приемов работы. Также в интернете можно найти множество уроков и статей по использованию LTspice.

В лабораторных работах будут использоваться модели транзисторов для техпроцесса 90 нм, полученные путем моделирования. Цифра в названии технологии, как правило, означает минимальную длину канала транзистора. Так как увеличение длины канала транзистора в цифровых схемах не дает никакого положительного эффекта, все транзисторы в рамках выполняемых работ будут иметь длину канала, равную 90 нм. Минимальная допустимая ширина канала обычно в два раза больше минимальной допустимой длины, т.е. для техпроцесса 90 нм минимальная ширина канала равна 180 нм. Увеличение ширины канала увеличивает его проводимость и емкости между затвором и каналом.

### 1.3.4 Создание инвертора и моделирование его работы

Рассмотрим основные принципы работы в среде LTspice и познакомимся с КМОП-схемотехникой на примере простейшего цифрового вентиля – инвертора.

#### Подготовка к работе

Создайте рабочий каталог для сохранения файлов проекта. Загрузите файл 90nm\_bulk.txt с параметрами транзисторов (предоставляется преподавателем) и сохраните его в своем рабочем каталоге. В этом же каталоге следует сохранять создаваемые схемы (файлы \*.asc, \*.asy).

Запустите программу LTspice. Окно программы представлено на рисунке 1.1.

В меню *Tools* → *Color Preferences* можно настроить цвета, в которых изображаются рабочее поле и схема. Необходимо выбирать светлый фон и темные линии при подготовке иллюстраций к отчету.

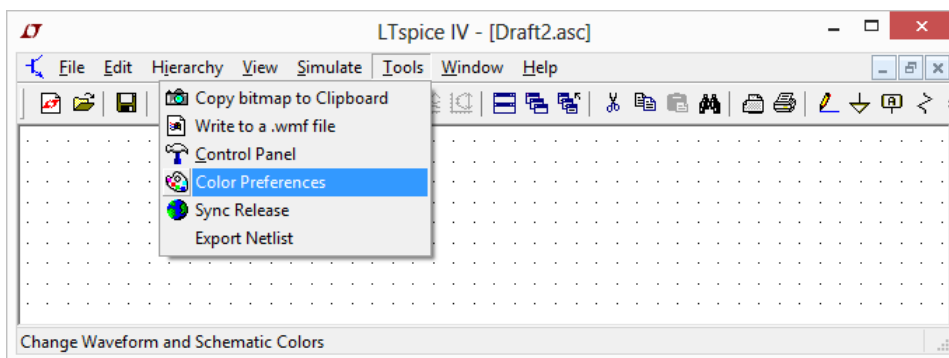


Рисунок 1.1 – Окно программы LTspice

Включить отображение сетки в рабочем поле можно с помощью *View → Show Grid*.

### Создание и симуляция схемы

Создайте новую схему (*File → New Schematic*) и сохраните ее файл в своем рабочем каталоге (*File → Save As...*). Добавьте к схеме SPICE-директиву, подключающую параметры транзисторов (*Edit → SPICE Directive*, рисунок 1.2). Помните, что директивы SPICE начинаются с точки. Разместите директиву на схеме (рисунок 1.3).

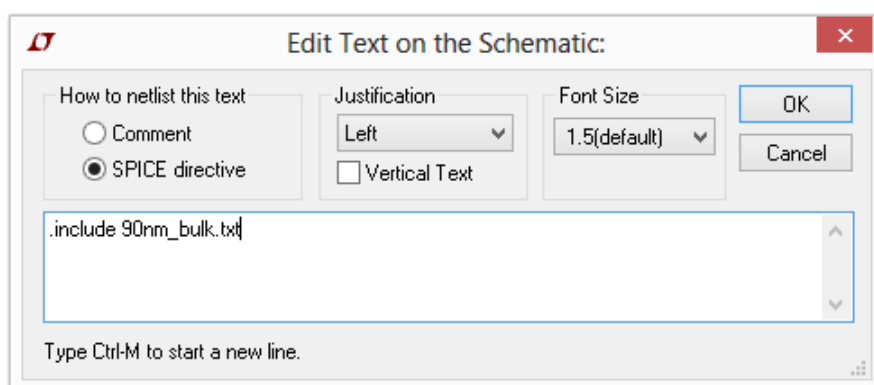


Рисунок 1.2 – Окно ввода директивы

**.include 90nm\_bulk.txt**

Рисунок 1.3 – Директива, вставленная в схему

На рисунке 1.4 показана схема тестирования инвертора – простейшего цифрового вентиля. Схема включает: КМОП-инвертор из двух транзисторов (PMOS и NMOS), источник питания инвертора (VDD1), источник тестового входного напряжения (V1) и нагрузку выхода инвертора (R1 и C1). Нагрузка необходима, чтобы моделировать реальную ситуацию, когда выход инвертора управляет другими вентилями.

Постройте схему. Для этого добавьте на схему транзисторы `pmos4` и `nmos4`, компоненты `voltage`, `res`, `cap` (*Edit* → *Component*) и землю (*Edit* → *Place GND*). Сориентируйте элементы нужным образом (*Edit* → *Rotate* и *Edit* → *Mirror*), а затем соедините проводами (*Edit* → *Draw Wire*). Проводам можно присваивать имена с помощью *Edit* → *Label Net* (только ввести имя, тип порта оставить `None`). Если присвоить двум проводам в разных местах схемы одинаковое имя, это будет эквивалентно их соединению. На схеме тестирования инвертора таким образом соединены все фрагменты провода с именем `VDD`.

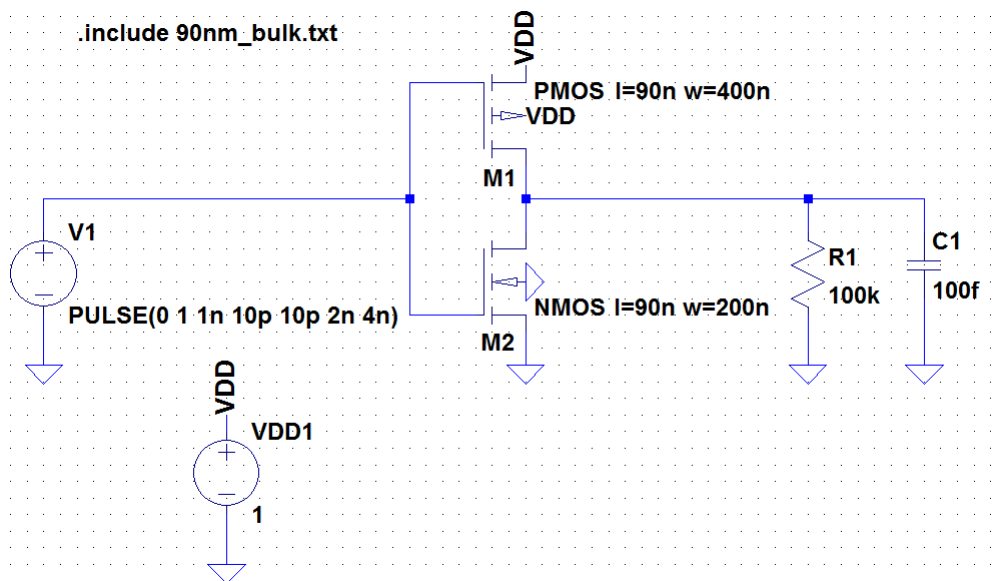


Рисунок 1.4 – Схема тестирования инвертора

Отредактируйте параметры компонентов в соответствии со схемой на рисунке 1.4. Для редактирования параметров необходимо щелкнуть на компоненте правой кнопкой мыши.

Для транзисторов необходимо ввести длину и ширину (`l` и `w`) канала в нанометрах. Заметьте, что ширина канала PMOS-транзистора в два раза больше, чем ширина канала NMOS-транзистора. Это связано с тем, что подвижность дырок примерно в два раза ниже подвижности электронов. Следовательно, чтобы обеспечить одинаковую проводимость каналов, ширину PMOS-канала нужно сделать в два раза больше. Чтобы вывести параметры транзистора рядом с компонентом (как на рисунке 1.4), необходимо настроить видимость параметров компонента: при зажатой клавише `Ctrl` щелкнуть правой кнопкой мыши на компоненте и отметить столбец *Visible* для поля *Value2*.

Для источника питания инвертора `VDD1` задайте постоянное значение напряжения 1 В. Для источника напряжения `V1` задайте параметры, как показано на рисунке 1.5 (для этого в окне параметров необходимо нажать кнопку *Advanced*). Источник

V1 генерирует прямоугольные импульсы (чередование «0» и «1»), которые будут поданы на вход инвертора для его тестирования. Заметьте, что буквы после чисел обозначают приставки кратных и дольных величин в системе СИ (к – кило-, п – нано-, р – пико-, ф – фемто- и т.д.).

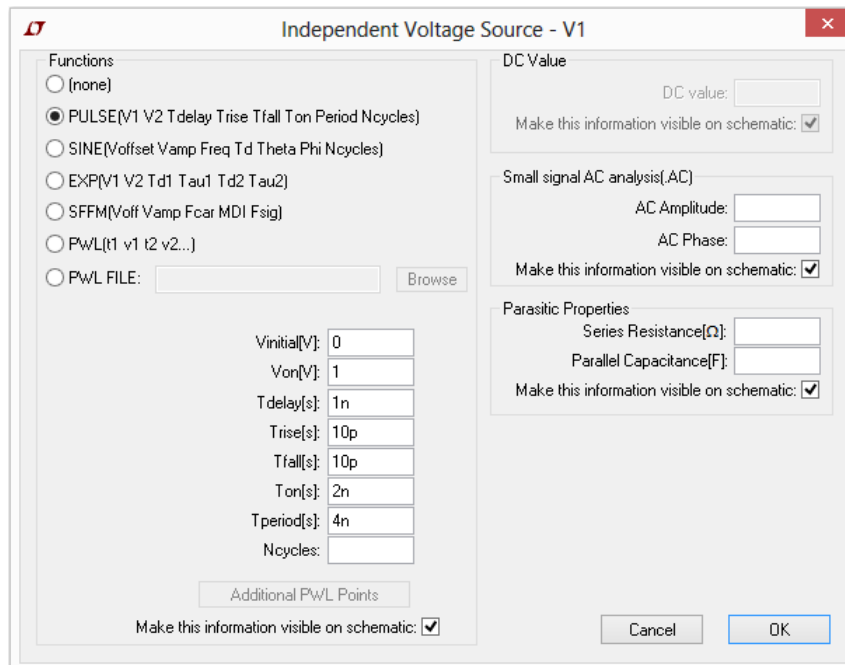


Рисунок 1.5 – Окно настроек источника напряжения

Добавьте на схему настройки анализа переходного процесса (*Edit* → *SPICE Analysis*) с параметрами, как показано на рисунке 1.6. Настройки задают время моделирования 8 нс и шаг моделирования 1 пс.

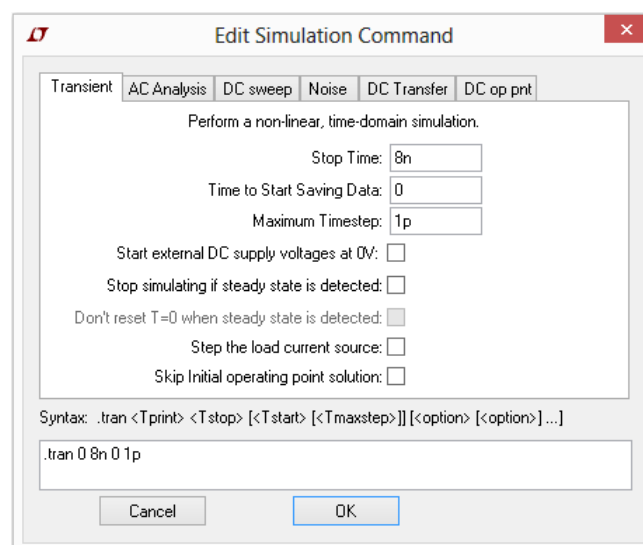


Рисунок 1.6 – Окно настроек процедуры анализа переходного процесса



Запустите симуляцию (*Simulate* → *Run*). Появится окно для вывода временных диаграмм. Наведите курсор мыши на провод перед инвертором; курсор примет вид щупа осциллографа. Щелкните на проводе, чтобы добавить временную диаграмму входного напряжения инвертора на график. Аналогично добавьте диаграмму выходного напряжения. График должен иметь вид, как на рисунке 1.7. Выходное напряжение инвертора противоположно входному, а значит, инвертор работает корректно.

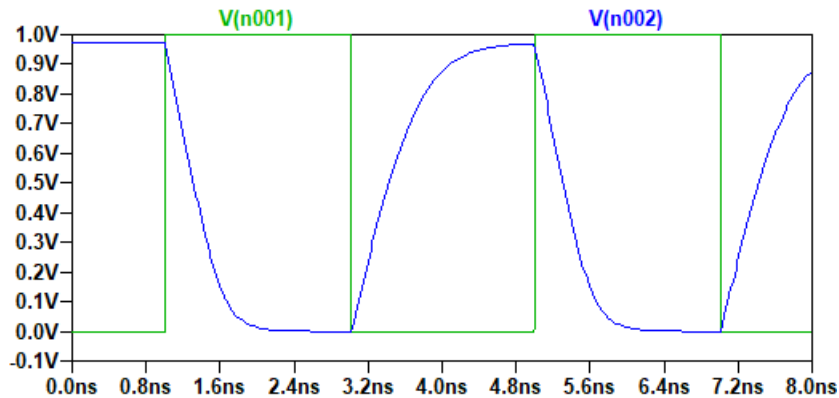


Рисунок 1.7 – Результаты моделирования переходного процесса в инверторе

### 1.3.5 Создание иерархических элементов

Для разработки более сложных схем удобно будет иметь готовый инвертор в библиотеке и вставлять его в новые схемы как иерархический элемент.

Для создания библиотечного элемента требуется создать схему (*File* → *New Schematic*) и символ (*File* → *New Symbol*) с одинаковыми именами, например, *inverter*.

Постройте схему инвертора, как показано на рисунке 1.8.

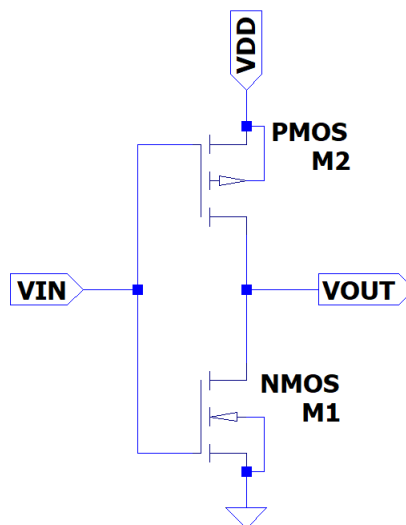


Рисунок 1.8 – Схема инвертора

На схему помещены именованные порты ввода и вывода с помощью *Edit* → *Label Net* и выбора типа порта.

Теперь приступим к созданию символа элемента (*File* → *New Symbol*). В качестве символов элементов следует использовать общепринятые обозначения элементов по стандартам ГОСТ, IEC или ANSI. Нарисуйте ANSI-символ инвертора, как показано на рисунке 1.9. Для рисования используйте команды из меню *Draw*. Добавьте в символ порты (*Edit* → *Add Pin/Port*) с такими же названиями, как и в схеме элемента. Для порта выберите, надо ли отображать название, и если да, то с какой стороны от него будет отображаться точка электрического соединения.

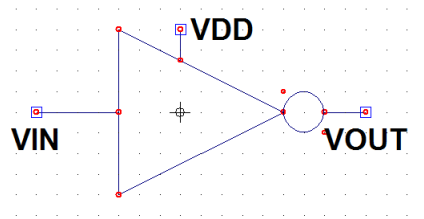


Рисунок 1.9 – Символ инвертора

Теперь инвертор можно вставлять в новые схемы как библиотечный элемент. Постройте схему тестирования инвертора с применением иерархического символа (рисунок 1.10). Промоделируйте схему. Результаты должны совпасть с результатами моделирования предыдущей схемы.

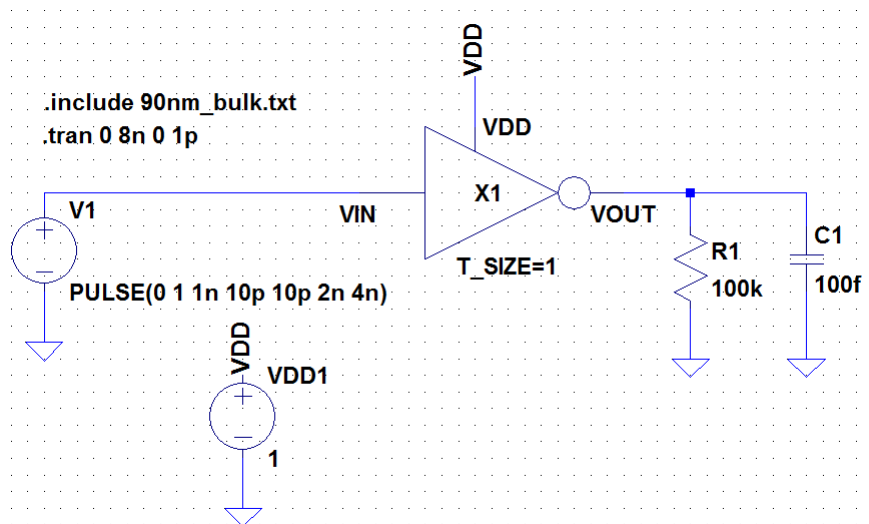


Рисунок 1.10 – Схема тестирования инвертора с использованием разработанного символа

Настроим отображение сигналов на разных координатных сетках. Щелкните правой кнопкой мыши на окне временной диаграммы и выберите пункт *Add plot pane* (рисунок 1.11). На диаграмме появится пустое поле, на которое необходимо пере-

тащить один из двух сигналов. Для этого, щелкнув по окну правой кнопкой, надо выбрать из контекстного меню пункт *Edit* → *Move* и выделить курсором в окне диаграммы прямоугольную область, включающую название требуемого сигнала (например, «V(n001)»). После того как область будет выделена, название сигнала станет перемещаться вместе с курсором. Щелкните по пустой области диаграммы, чтобы добавить в нее выбранный сигнал. Для удаления сигнала необходимо выбрать из контекстного меню инструмент *Edit* → *Delete*, а затем щелкнуть по названиям ненужных сигналов или выделить их. В результате должна получиться диаграмма, показанная на рисунке 1.12. Более подробно об управлении временными диаграммами можно узнать в справке LTspice.

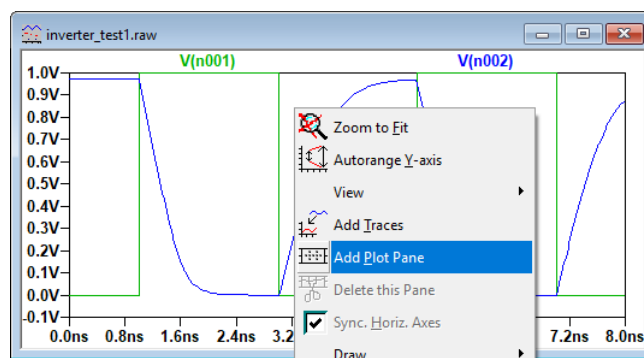


Рисунок 1.11 – Добавление дополнительной области на временную диаграмму

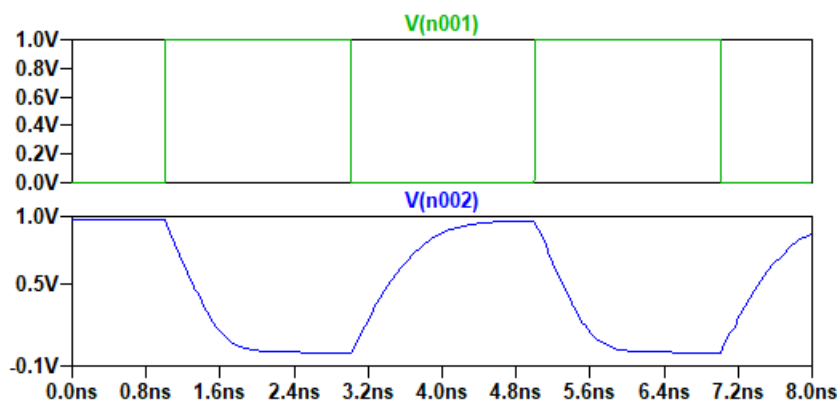


Рисунок 1.12 – Временная диаграмма с разными областями для отображения сигналов

Отредактируйте параметры инвертора. Для этого щелкните по символу правой кнопкой мыши и в строке *PARAMS* введите «*T\_SIZE=4*». Промоделируйте схему еще раз. Длительность переходных процессов при переключении инвертора должна уменьшиться.

## 1.4 Основы работы в среде Vivado Design Suite

В данном разделе описаны принципы работы в системе автоматизированного проектирования (САПР) Vivado Design Suite (далее Vivado) фирмы Xilinx. Версия WebPACK является бесплатной и доступна для скачивания с официального сайта Xilinx (<https://www.xilinx.com/>).

Vivado предназначена для проектирования программного обеспечения микросхем программируемой логики (ПЛИС) фирмы Xilinx. В разделе описан пример проектирования в Vivado версии 2017.4. Описанные рекомендации также применимы и для более ранних и поздних версий с учетом небольших различий версий в элементах пользовательского интерфейса.

### 1.4.1 Схема сумматора

Чтобы ознакомиться с принципами проектирования цифровых схем в Vivado, разберем пример построения схемы неполного одноразрядного сумматора. Таблица истинности рассматриваемой схемы сумматора представлена в таблице 1.1.

Таблица 1.1 – Таблица истинности неполного одноразрядного сумматора

a	b	c
0	0	0
0	1	1
1	0	1
1	1	0

Построим данную схему в базисе И-НЕ. Для этого сначала запишем совершенную дизъюнктивную нормальную форму (СДНФ) функции сумматора:

$$a + b = \bar{a} \cdot b \vee a \cdot \bar{b}. \quad (1.1)$$

Возьмем двойное отрицание и преобразуем СДНФ по правилам де Моргана:

$$\overline{\overline{\bar{a} \cdot b \vee a \cdot \bar{b}}} = \overline{\overline{\bar{a} \cdot b} \cdot \overline{a \cdot \bar{b}}} \quad (1.2)$$

В итоге для выполнения суммирования двух одноразрядных чисел нам понадобится 5 двухвходовых элементов И-НЕ. Схема сумматора на И-НЕ представлена на рисунке 1.13.

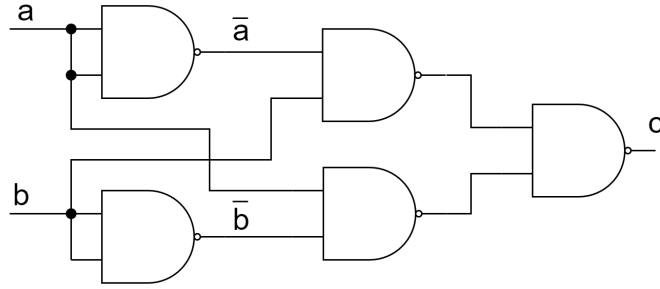


Рисунок 1.13 – Схема сумматора в базисе И-НЕ

### 1.4.2 Описание схемы сумматора на вентиляльном уровне с использованием языка Verilog HDL

Рассмотрим пример описания схемы, изображенной на рисунке 1.13, на языке Verilog HDL, а также пример описания тестового окружения для этой схемы.

В данном разделе мы будем описывать схему на вентиляльном уровне, то есть используя только примитивы вентиляей, доступные в Verilog HDL.

Дадим всем линиям схемы обозначения, которые затем будем использовать в текстовом описании схемы. Аннотированная схема представлена на рисунке 1.14.

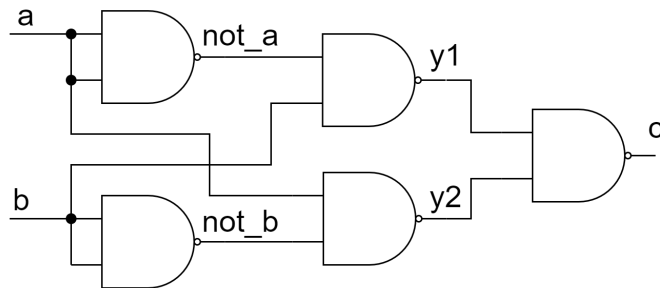


Рисунок 1.14 – Аннотированная схема сумматора в базисе И-НЕ

Полное описание модуля, реализующего функциональность схемы на Verilog HDL, представлено в листинге 1.1.

Модуль является базовой единицей схемы в Verilog HDL. Вся логика и структура схемы описываются внутри модулей. Модули могут входить в состав других модулей, образуя при этом иерархическую структуру целевого устройства. Модули в Verilog обрамляются ключевыми словами **module** и **endmodule** в начале и конце соответственно. Рекомендуется описывать каждый модуль в виде отдельного файла, хотя это не является обязательным.

```
'timescale 1ns / 1ps

module adder(
    input  a,
    input  b,
    output c
);

    wire not_a, not_b, y1, y2;

    nand(not_a, a, a);
    nand(not_b, b, b);

    nand(y1, not_a, b);
    nand(y2, a, not_b);

    nand(c, y1, y2);

endmodule
```

---

В самом начале исходного кода располагается директива **timescale**, которая задает шаг (1ns) и точность (1ps) продвижения модельного времени. Данная директива не является обязательной, и, чтобы она работала, необходимо, чтобы она была одинакова во всех файлах проекта. При этом данная настройка может быть переопределена в настройках симулятора.

После ключевого слова **module** записывается уникальное имя модуля. В нашем случае имя модуля – **adder**. Далее идет определение портов с указанием направления передачи данных: входной (**input**) или выходной (**output**) порт.

По умолчанию тип переменной порта, если он не указан, задается как **wire**. В Verilog также существует ещё один синтезируемый тип переменных и портов – это **reg**. Когда говорится, что данная конструкция языка синтезируема, то подразумевается, что всё, что описано с помощью таких конструкций, может быть транслировано в конкретную цифровую схему. Несинтезируемое подмножество языка используется только для моделирования, например, задания входных и анализа выходных значений, и не может быть транслировано в элементы цифровой схемы.

Переменные **wire** используются для описания только комбинационной логики (схем без памяти). Переменные **reg** могут быть использованы для описания как ком-

бинационной, так и последовательностной логики (схем с памятью). При этом тип **reg** может быть только у выходных портов.

В разбираемом примере мы используем только элементы комбинационной логики, поэтому можно ограничиться использованием типа **wire**. В листинге 1.1 определены линии `not_a`, `not_b`, `y1`, `y2` в соответствии с рисунком 1.14.

Далее следует описание набора вентилях И-НЕ **nand**. В Verilog доступны для использования и другие вентиля, такие как `nor`, `or`, `and`, `not` и `xor`. При описании вентилях первый аргумент задает связь выходного значения с локальной переменной, остальные аргументы – это входные значения.

При описании вентилях порядок их задания не имеет значения. Следует понимать, что при таком описании описывается структура взаимосвязей компонентов, а не последовательность действий.

После того, как модуль сумматора готов, необходимо удостовериться в его работоспособности. Для проверки работоспособности создается специальный модуль тестового окружения, в котором описывается последовательность изменения входных и анализ выходных значений нашего модуля. Описание тестового окружения сумматора представлено в листинге 1.2.

Модуль тестового окружения не взаимодействует с внешним миром, поэтому не имеет портов ввода/вывода. В первой части модуля описано подключение разработанного модуля сумматора и определено подключение его портов. Порты можно подключать простым перечислением, а можно с явным указанием имен портов, как и сделано в приведенном листинге. Второй вариант подключения является более безопасным, так как изменение порядка следования портов в модуле сумматора не приведет к ошибке в соединении портов в тестовом окружении.

Переменные, задающие входные значения, имеют тип **reg**, так как мы должны иметь возможность удерживать тестовые значения на входах модуля, чтобы проанализировать его выход. Условно можно считать, что входные порты сумматора подключаются к регистрам, значения которых мы будем менять в процессе моделирования. Переменная, к которой подключается выходной порт модуля, может иметь только тип **wire**.

Последовательность изменения входных значений задается в процедурном блоке **initial**. Он вызывается в самом начале процесса моделирования и после выполнения всех действий повторно не выполняется. Если определено несколько блоков **initial**, то их выполнение начинается одновременно и идет параллельно, и невозможно предсказать точную последовательность действий, привязанных к одному и тому же моменту времени.

```
'timescale 1ns / 1ps

module adder_tb;

    reg a_in, b_in;
    wire c_out;

    adder adder_1(
        .a(a_in),
        .b(b_in),
        .c(c_out)
    );

    integer i;
    reg [1:0] test_val;
    reg expected_val;

    initial begin
        for (i = 0; i < 4; i = i+1) begin
            test_val = i;
            a_in = test_val[0];
            b_in = test_val[1];
            expected_val = ^test_val;

            #10

            if (c_out == expected_val) begin
                $display("The adder output is correct!!! a_in=%b,
                    b_in=%b c_out=%b", a_in, b_in, c_out);
            end else begin
                $display("The adder output is wrong!!! a_in=%b,
                    b_in=%b c_out=%b, expected=%b", a_in, b_in,
                    c_out, expected_val);
            end
        end
        #10 $stop;
    end
endmodule
```

---



Если процедурный блок **initial** описывает выполнение последовательности действий, то эта последовательность должна быть обрамлена ключевыми словами **begin** и **end**. Это верно для всех процедурных блоков.

Продвижение модельного времени осуществляется с помощью явного указания задержек, предваряемых символом **#**. По умолчанию задержки указываются в наносекундах (нс). Это может быть переопределено с помощью директивы **timescale** или с помощью настроек симулятора.

В листинге 1.2 в теле блока **for** задается последовательность изменения входных значений и производится анализ выходных. Результат анализа выходных значений печатается в консоль симулятора с помощью встроенной функции **\$display**. Встроенные функции в Verilog предваряются символом доллара **\$**. В конце блока **initial** вызывается функция **\$stop**, которая останавливает моделирование.

### 1.4.3 Создание проекта в Vivado Design Suite

Создадим в Vivado проект для схемы сумматора, описанной в предыдущих разделах. Для запуска Vivado находим соответствующий значок на рабочем столе (рисунок 1.15) и щелкаем два раза.



Рисунок 1.15 – Иконка программного комплекса Vivado

В открывшемся окружении выбираем *File* → *New Project* (рисунок 1.16). В открывшемся окне нажимаем *Next* (рисунок 1.17). Задаем имя проекта и нажимаем *Next* (рисунок 1.18). Выбираем *RTL Project* и нажимаем *Next* (рисунок 1.19).

В качестве микросхемы программируемой логики, под которую будет вестись разработка устройства, выбираем XC7A100TCSG324-1 и нажимаем *Next* (рисунок 1.20). Данная микросхема установлена в отладочной плате Nexys A7 (<https://store.digilentinc.com/nexys-a7-fpga-trainer-board-recommended-for-ece-curriculum/>), которая будет в дальнейшем использоваться в работах.

В последнем диалоговом окне нажимаем кнопку *Finish* (рисунок 1.21). После создания проекта откроется окружение, показанное на рисунке 1.22.

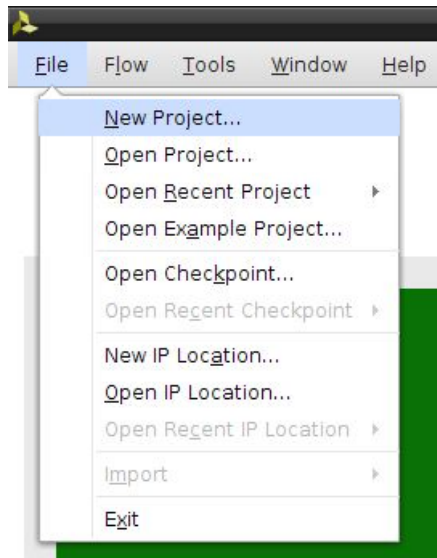


Рисунок 1.16 – Пункт меню создания нового проекта



Рисунок 1.17 – Диалоговое окно создания нового проекта

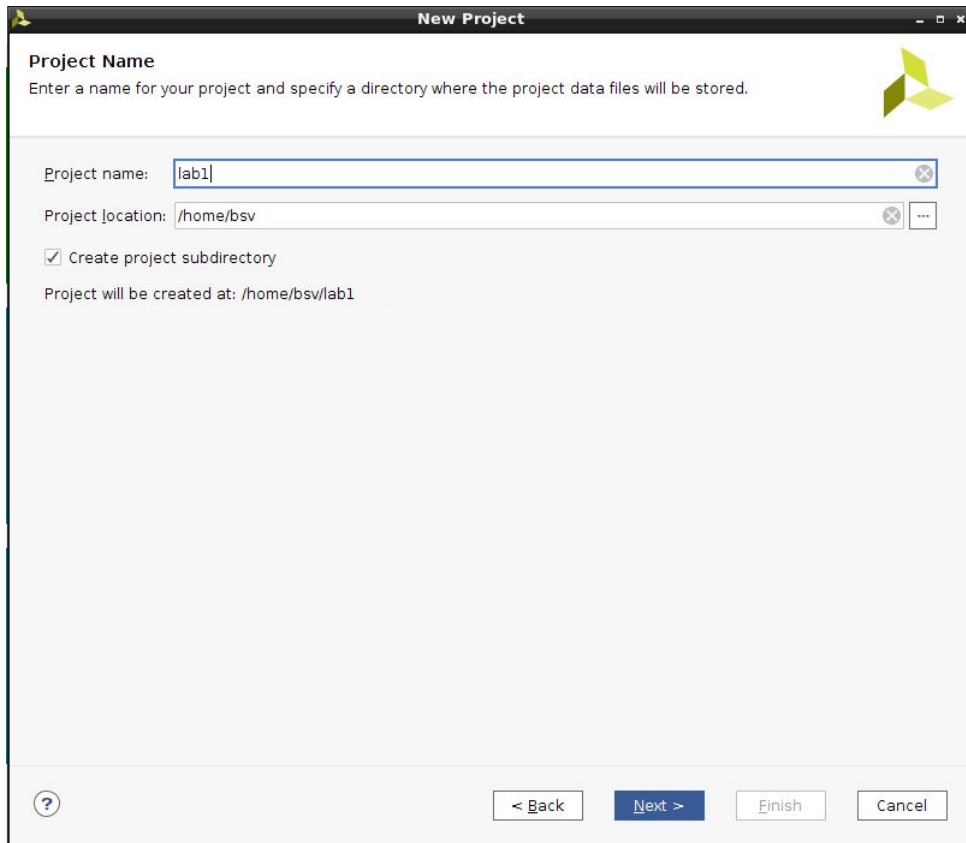


Рисунок 1.18 – Окно ввода имени проекта

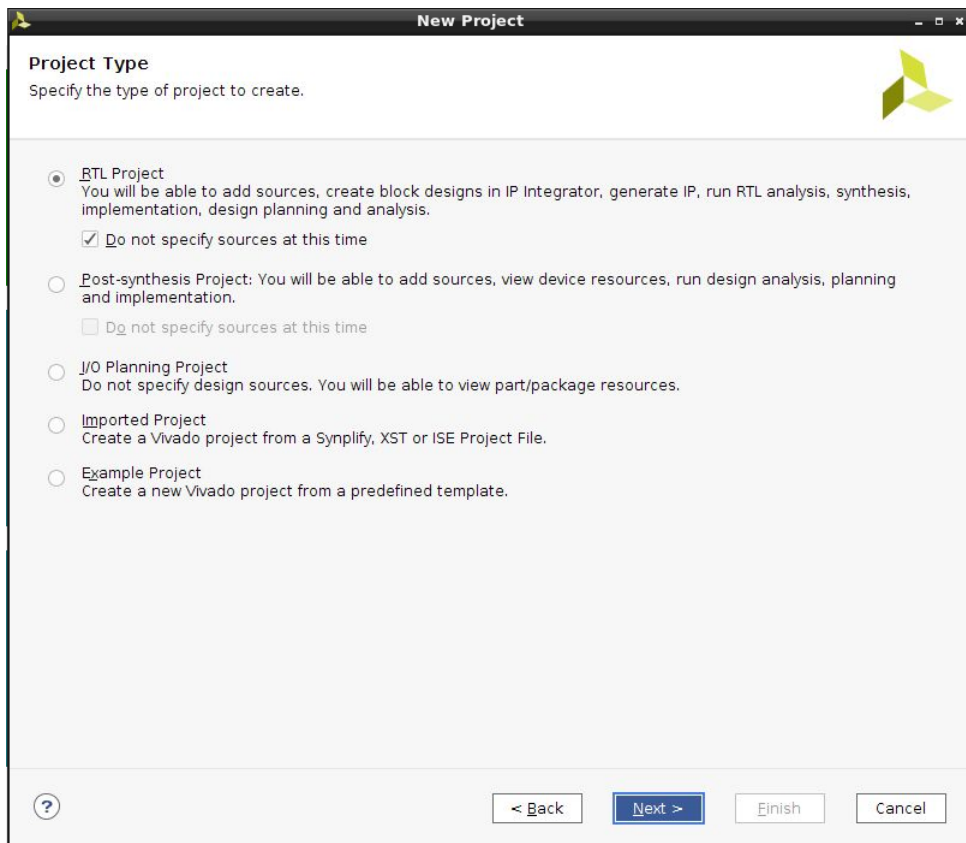


Рисунок 1.19 – Окно выбора типа проекта

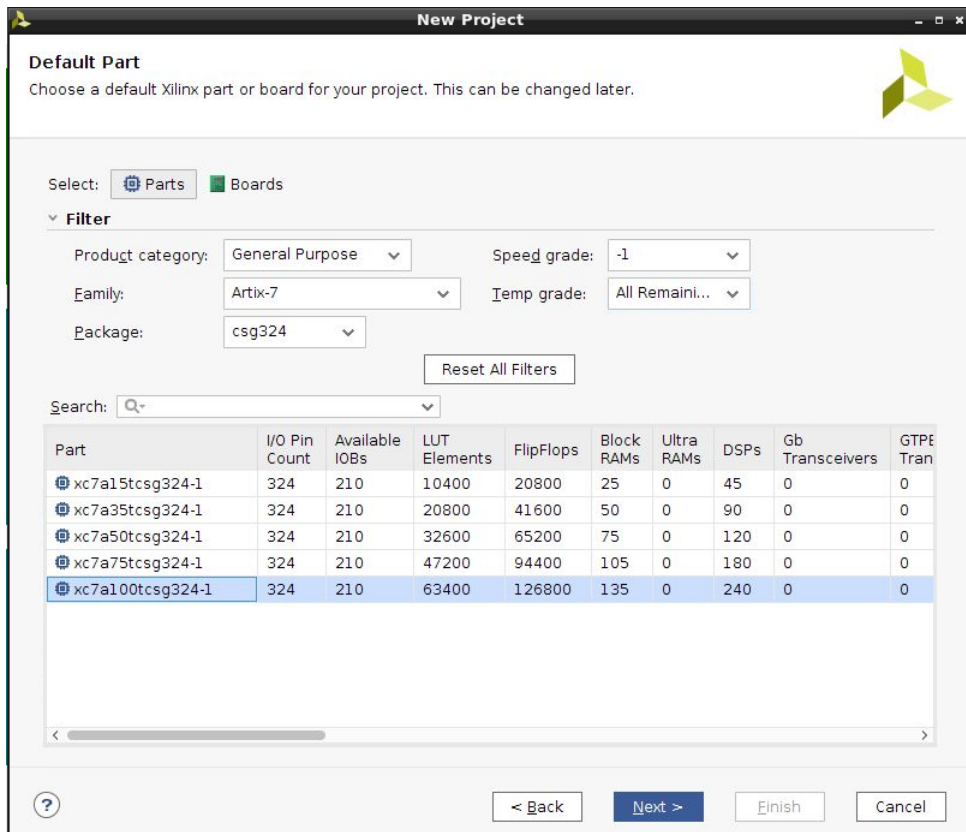


Рисунок 1.20 – Окно выбора целевой микросхемы программируемой логики

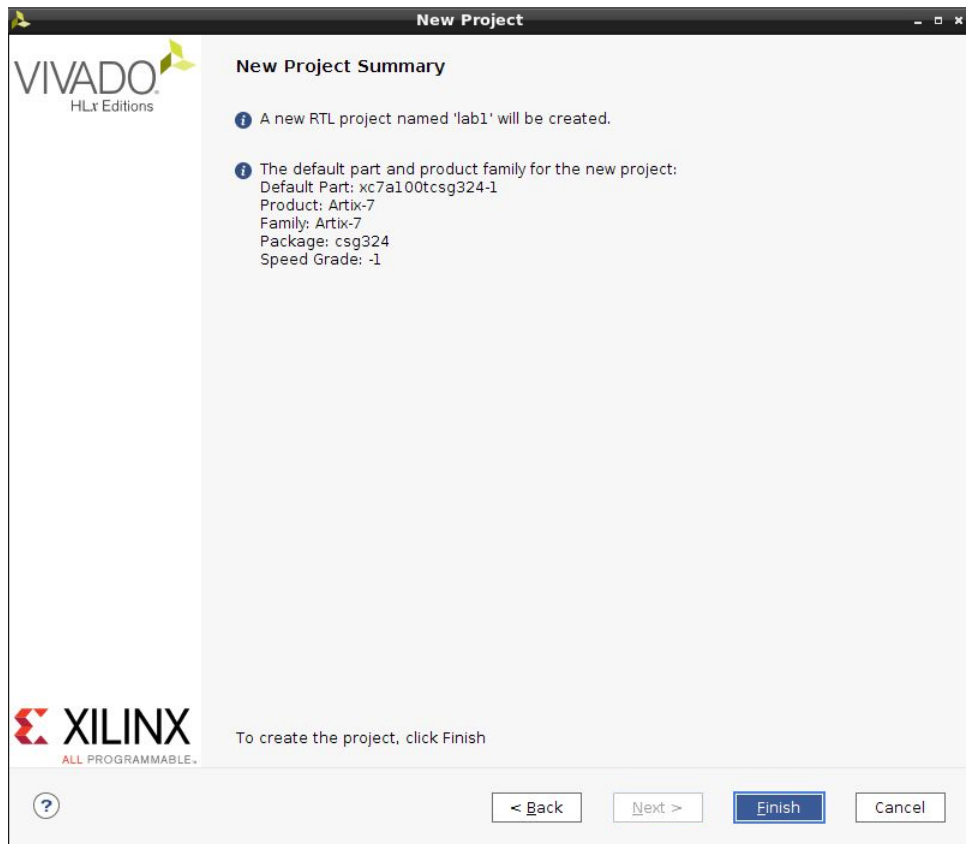


Рисунок 1.21 – Окно с информацией о создаваемом проекте и кнопкой завершения создания нового проекта

Для добавления новых исходных файлов нажимаем на *Add Sources* в левом верхнем углу рабочего окружения (подчеркнуто красным на рисунке 1.22).

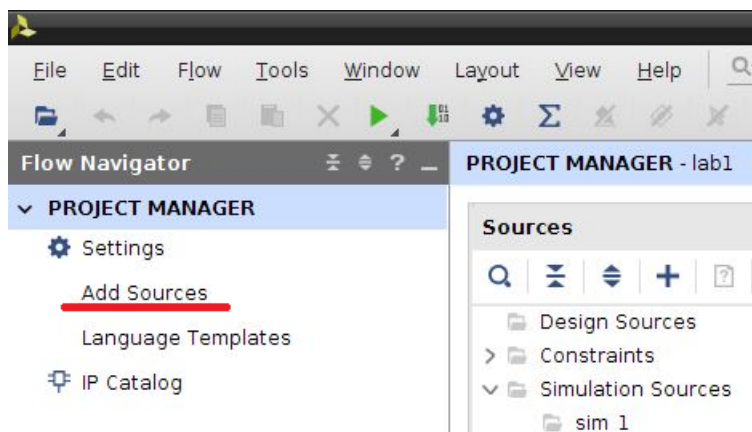


Рисунок 1.22 – Кнопка для добавления исходных файлов в проект

В открывшемся окне выбираем *Add or create design sources* и нажимаем *Next* (рисунок 1.23).

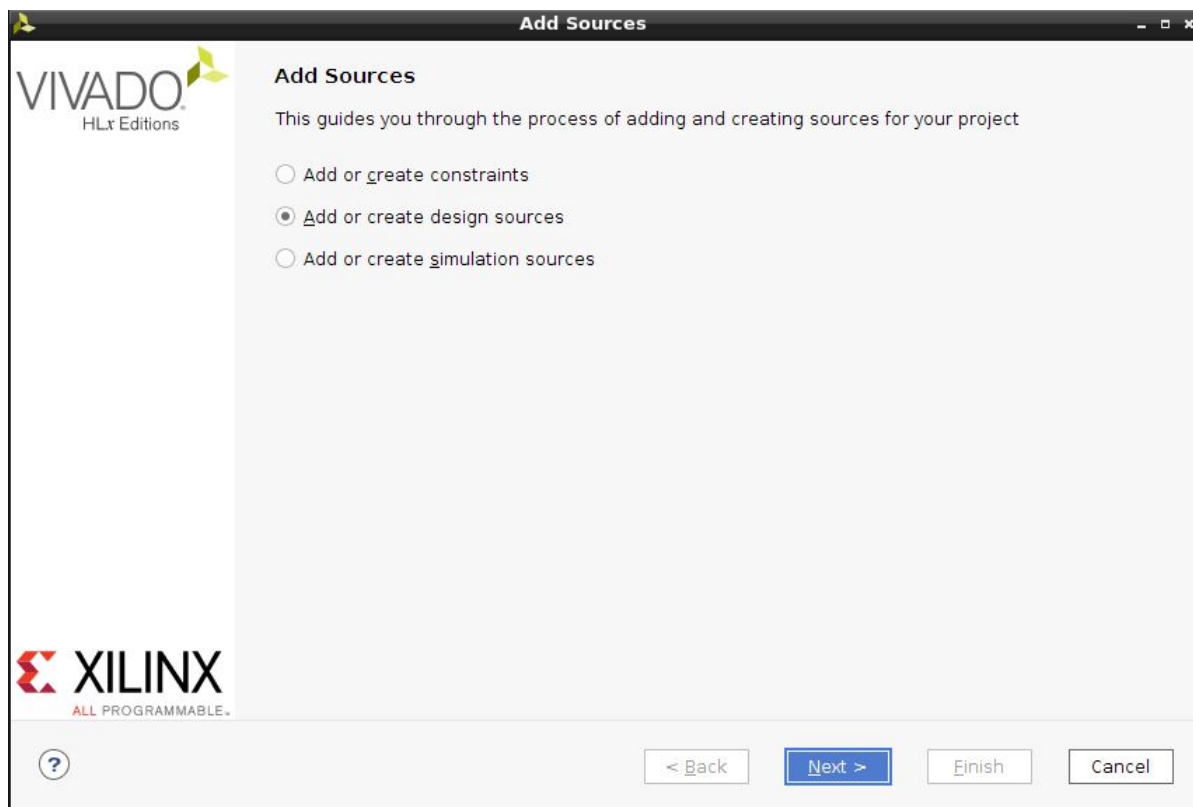


Рисунок 1.23 – Окно выбора типа добавляемых исходных файлов

В открывшемся окне выбираем *Create File* и задаем имя файлу, в котором будем описывать функциональность сумматора (рисунок 1.24).

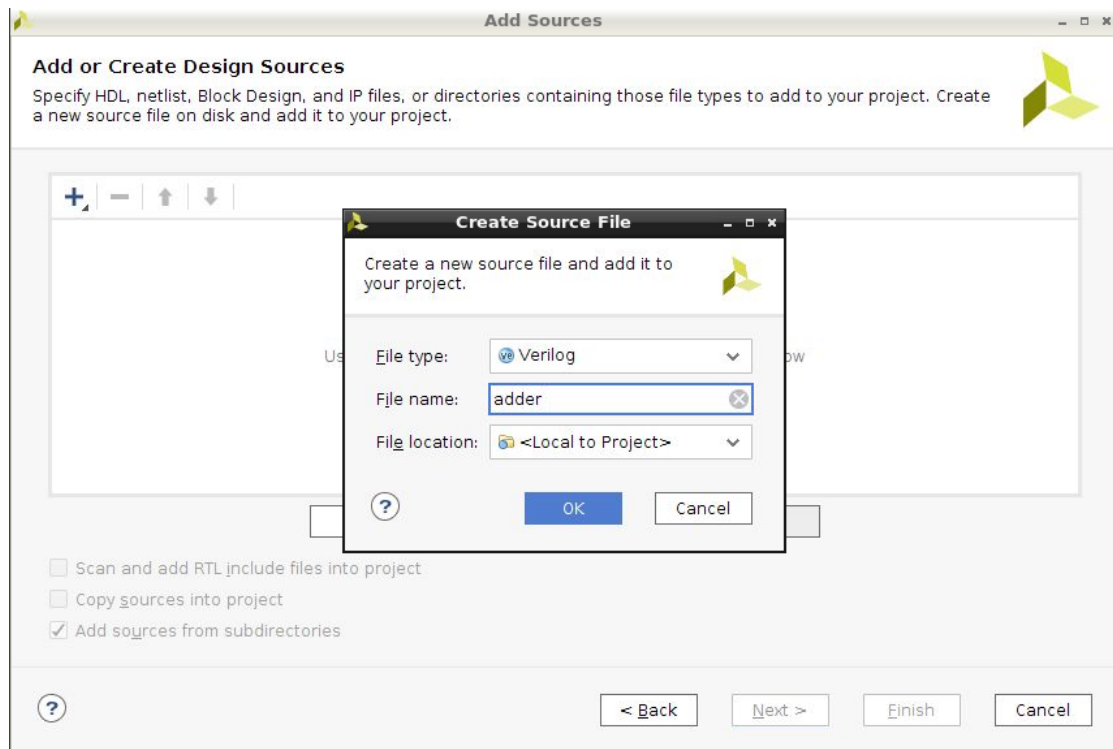


Рисунок 1.24 – Окно с полем ввода имени для создаваемого проектного файла

Нажимаем *Next* и переходим к описанию портов модуля (рисунок 1.25). Данный шаг можно пропустить, нажав *OK*, и явно определить интерфейс модуля в файле модуля.

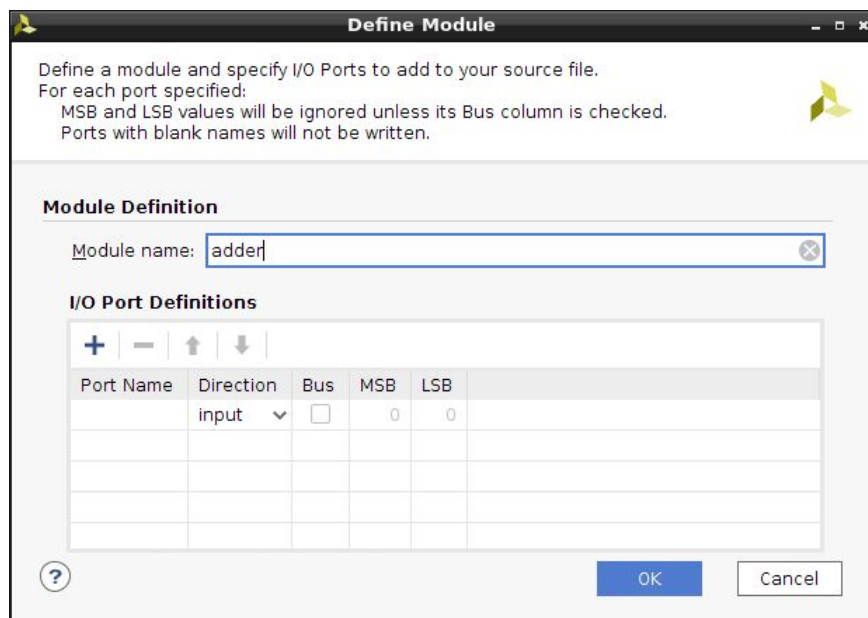
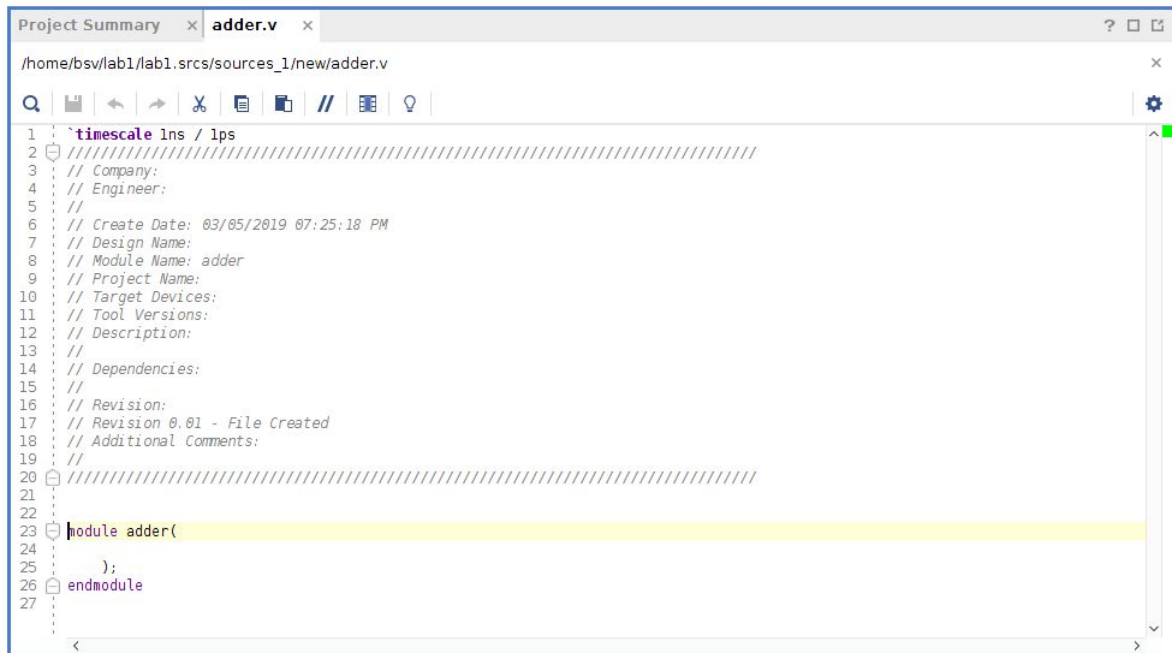


Рисунок 1.25 – Окно для создания портов ввода/вывода нового модуля

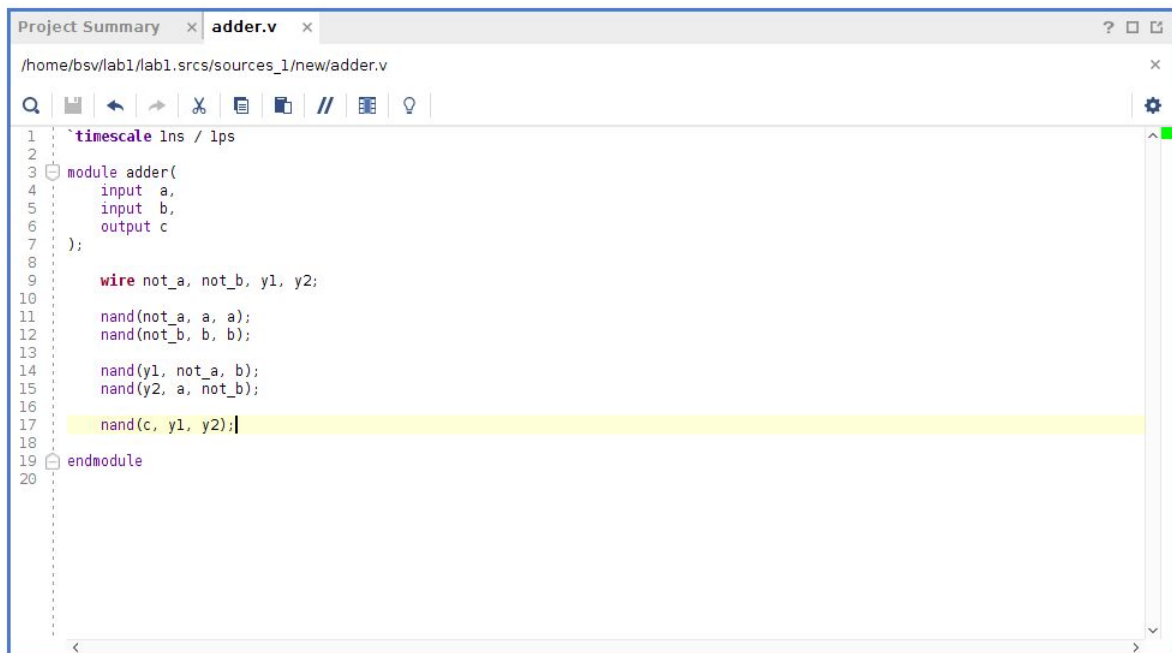
После создания нового файла он откроется для редактирования (рисунок 1.26). Добавим описание модуля (рисунок 1.27).

Теперь у нас есть проект и описание модуля сумматора. Далее необходимо создать тестовое окружение для разработанного модуля.



```
Project Summary x adder.v x
/home/bsv/lab1/lab1.srcs/sources_1/new/adder.v
1 `timescale 1ns / 1ps
2 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////
3 // Company:
4 // Engineer:
5 //
6 // Create Date: 03/05/2019 07:25:18 PM
7 // Design Name:
8 // Module Name: adder
9 // Project Name:
10 // Target Devices:
11 // Tool Versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////
21
22
23 module adder(
24
25 );
26 endmodule
27
```

Рисунок 1.26 – Вид в редакторе содержимого файла модуля после создания



```
Project Summary x adder.v x
/home/bsv/lab1/lab1.srcs/sources_1/new/adder.v
1 `timescale 1ns / 1ps
2
3 module adder(
4     input a,
5     input b,
6     output c
7 );
8
9     wire not_a, not_b, y1, y2;
10
11     nand(not_a, a, a);
12     nand(not_b, b, b);
13
14     nand(y1, not_a, b);
15     nand(y2, a, not_b);
16
17     nand(c, y1, y2);
18
19 endmodule
20
```

Рисунок 1.27 – Вид в редакторе целевого описания модуля

## 1.4.4 Моделирование схемы

Для создания файла тестового окружения вновь нажимаем на *Add Sources* в левом верхнем углу рабочего окружения (подчеркнуто красным на рисунке 1.22). Теперь выбираем *Add or create simulation sources* (рисунок 1.28).

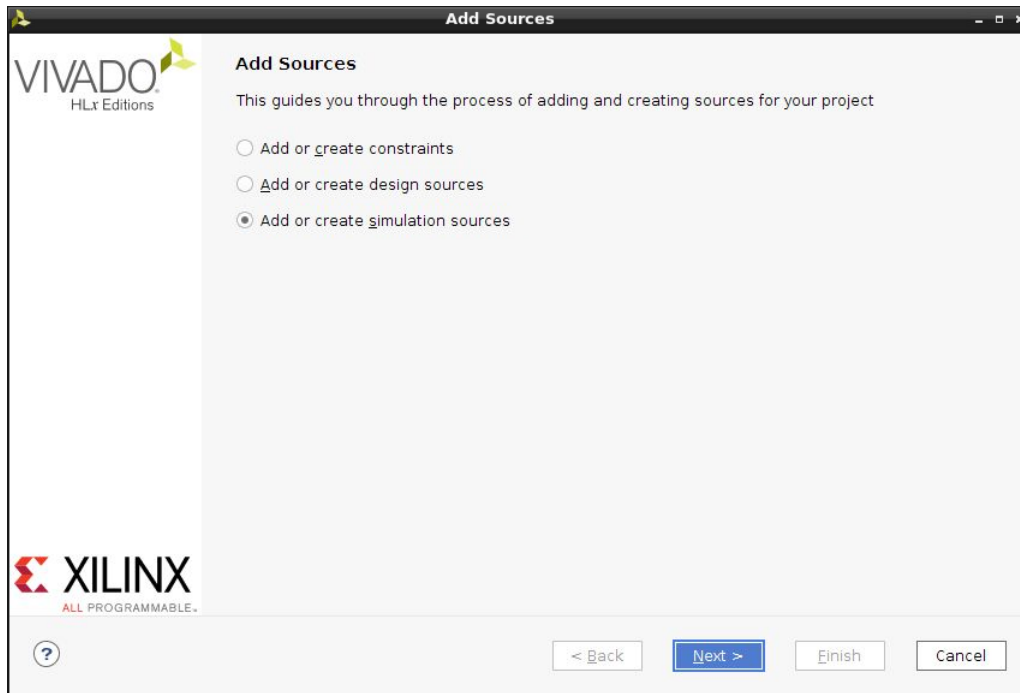


Рисунок 1.28 – Окно добавления файлов для окружения тестирования

В открывшемся окне выбираем *Create File* и задаем имя файлу, в котором будем описывать функциональность тестового окружения сумматора (рисунок 1.29).

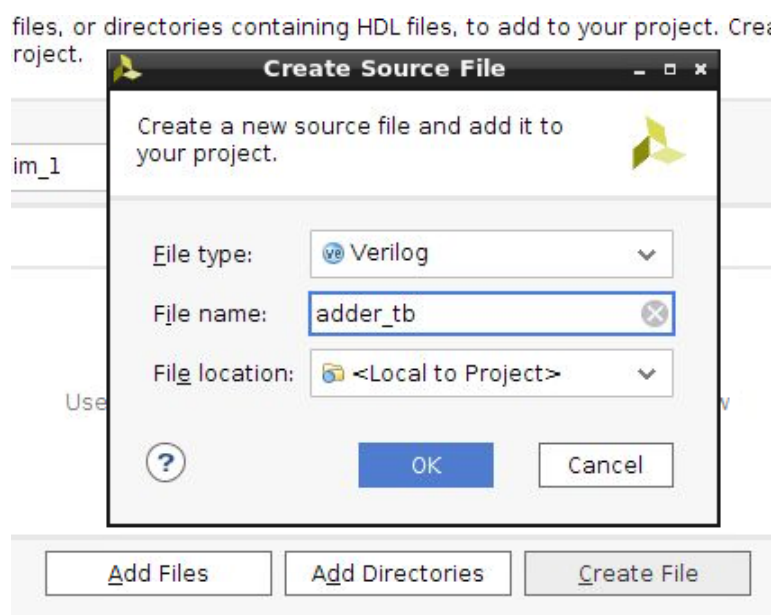
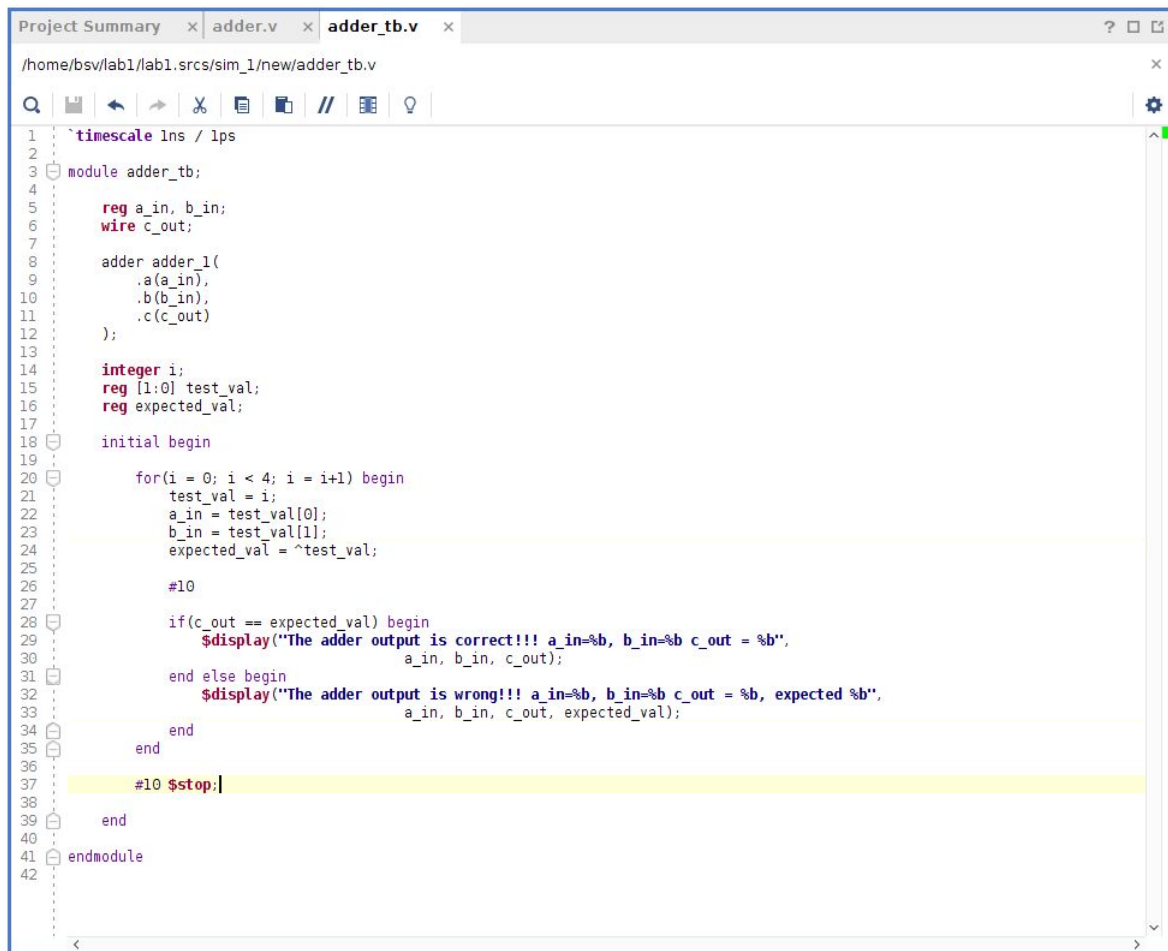


Рисунок 1.29 – Окно ввода имени файла тестового окружения



После создания файла тестового окружения, наполняем его (рисунок 1.30).



```
1  `timescale 1ns / 1ps
2
3  module adder_tb;
4
5      reg a_in, b_in;
6      wire c_out;
7
8      adder adder_1(
9          .a(a_in),
10         .b(b_in),
11         .c(c_out)
12     );
13
14     integer i;
15     reg [1:0] test_val;
16     reg expected_val;
17
18     initial begin
19     :
20         for(i = 0; i < 4; i = i+1) begin
21             test_val = i;
22             a_in = test_val[0];
23             b_in = test_val[1];
24             expected_val = ^test_val;
25
26             #10
27
28             if(c_out == expected_val) begin
29                 $display("The adder output is correct!!! a_in=%b, b_in=%b c_out = %b",
30                     a_in, b_in, c_out);
31             end else begin
32                 $display("The adder output is wrong!!! a_in=%b, b_in=%b c_out = %b, expected %b",
33                     a_in, b_in, c_out, expected_val);
34             end
35         end
36     end
37     #10 $stop;
38
39 end
40
41 endmodule
42
```

Рисунок 1.30 – Вид в редакторе содержимого файла тестового окружения

Для запуска тестового окружения на выполнение необходимо в левой панели выбрать *Run Simulation* и в открывшемся выпадающем меню - *Run Behavioural Simulation* (рисунок 1.31).

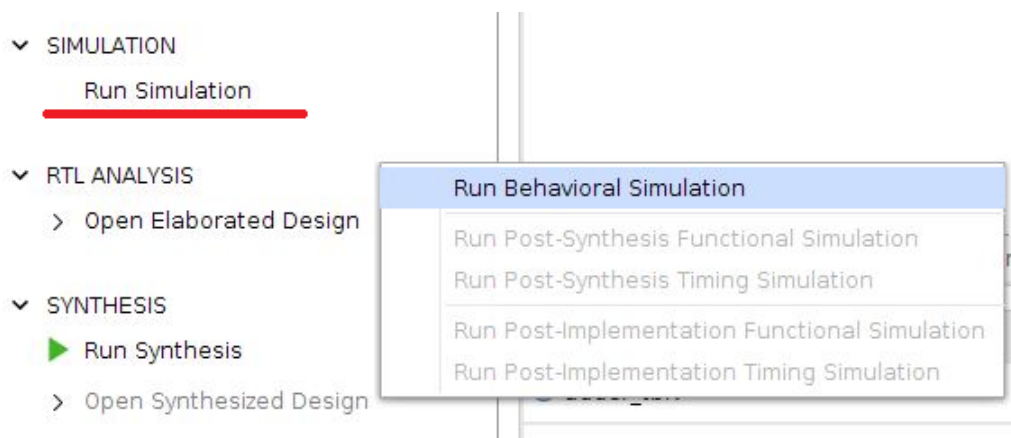


Рисунок 1.31 – Кнопка для запуска моделирования на поведенческом уровне

После запуска процесс моделирования остановится на моменте вызова функции `$stop`. Для просмотра результатов моделирования необходимо переключиться на вкладку с названием, начинающимся с *Untitled* (рисунок 1.32).

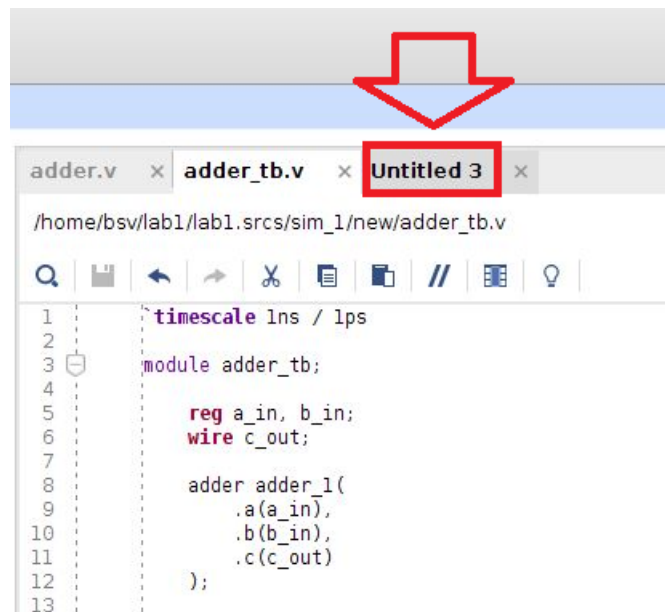


Рисунок 1.32 – Вкладка с результатами моделирования

По умолчанию временная диаграмма имеет масштаб, который не позволяет просмотреть весь процесс моделирования. Чтобы увидеть полную диаграмму, необходимо нажать на кнопку *Zoom Fit* (рисунок 1.33).

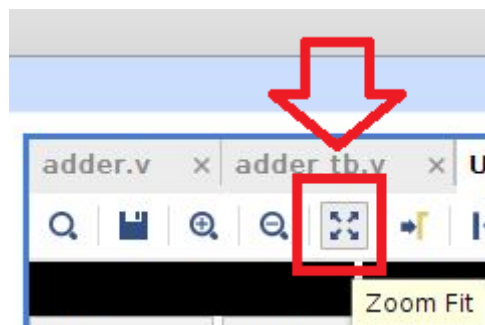


Рисунок 1.33 – Кнопка для просмотра полной временной диаграммы

На рисунке 1.34 представлена временная диаграмма всего процесса моделирования. Можно видеть все комбинации входных сигналов, значение выходного сигнала, а также вывод результатов анализа выходного значения схемы в консоли симулятора.

Стоит отметить, что при проведении описанного выше поведенческого моделирования схемы никак не учитываются задержки распространения сигналов через вентили.

The screenshot displays a behavioral simulation environment. The top window title is "SIMULATION - Behavioral Simulation - Functional - sim\_1 - adder\_tb".

**Scope Panel:** Shows a tree view with "adder\_tb" expanded to "adder", which contains "adder\_1" and "gbl".

**Objects Panel:** Lists variables and their values:
 

Name	Value	Data Type
a_in	1	Logic
b_in	1	Logic
c_out	0	Logic
i[31:0]	4	Array
test_val[1...]	3	Array
expected...	0	Logic

**Waveform Viewer:** Shows a timing diagram for variables a\_in, b\_in, c\_out, i[31:0], test...[0], and exp...val. The time axis ranges from 0 ns to 40 ns. The test...[0] signal is shown as a sequence of values: 0, 1, 2, 3.

**Tcl Console:** Contains the following text:
 

```

    # run 1000ns
    The adder output is correct!!! a_in=0, b_in=0 c_out = 0
    The adder output is correct!!! a_in=1, b_in=0 c_out = 1
    The adder output is correct!!! a_in=0, b_in=1 c_out = 1
    The adder output is correct!!! a_in=1, b_in=1 c_out = 0
    INFO: [USF-XSim-96] XSim completed. Design snapshot 'adder_tb_behav' loaded.
    INFO: [USF-XSim-97] XSim simulation ran for 1000ns
    
```

Рисунок 1.34 – Окно с результатами моделирования схемы

## **2 Лабораторная работа №2. Разработка аппаратных ускорителей математических вычислений**

### **2.1 Введение**

Лабораторная работа №2 посвящена разработке цифровых схем ускорителей математических операций на уровне регистровых передач (register transfer level, RTL) с использованием языка описания аппаратуры Verilog HDL.

В работе предлагается разработать цифровой блок со сложной логикой работы для выполнения заданной вариантом арифметической операции. Разработка должна осуществляться с учетом установленных ограничений на формат входных и выходных данных, а также на использование базовых арифметических блоков сложения и умножения.

В процессе реализации алгоритма работы разрабатываемого блока развиваются навыки проектирования цифровых схем конечных автоматов. Конечные автоматы используются для реализации управляющей логики блока и синхронизации шагов вычислительного процесса.

### **2.2 Описание лабораторной работы**

#### **Цель работы**

Получить навыки описания арифметических блоков на RTL-уровне с использованием языка описания аппаратуры Verilog HDL.

#### **Указания к выполнению работы**

Лабораторная работа посвящена знакомству с техниками описания схем арифметических блоков на RTL-уровне с использованием языка Verilog HDL. Работа выполняется в Vivado Design Suite.

#### **Порядок выполнения работы**

1. Разработайте и опишите на Verilog HDL схему, вычисляющую значение функции в соответствии с заданными ограничениями согласно варианту задания.
2. Определите область допустимых значений функции.

3. Разработайте тестовое окружение для разработанной схемы. Тестовое окружение должно проверять работу схемы не менее, чем на 10 различных тестовых векторах.
4. Проведите моделирование работы схемы и определите время вычисления результата. Схема должна тактироваться от сигнала с частотой 100 МГц.
5. Составьте отчет по результатам выполнения работы.

## Требования к отчету

Отчет должен быть оформлен в соответствии с требованиями, представленными в Приложении А.

### Отчет должен включать:

1. Титульный лист.
2. Цель работы.
3. Задание в соответствии с вариантом.
4. Схему (рисунок) разработанного блока вычисления функции, заданной вариантом, в терминах базовых операционных элементов (БОЭ) (т.е. используя мультиплексоры/демультиплексоры, шифраторы/дешифраторы, компараторы, регистры, счетчики и др.).
5. Описание работы разработанного блока, начиная с подачи входных данных и заканчивая получением результата.
6. Область допустимых значений для разработанного блока.
7. Результат тестирования разработанного блока (временные диаграммы).
8. Время вычисления результата при частоте тактового сигнала в 100 МГц.
9. Выводы по работе.

## Задания по вариантам

Вариант	Функция	Ограничения
1	$y = a^2 + \sqrt[3]{b}$	1 сумматор и 2 умножителя
2	$y = a^3 + \sqrt[2]{b}$	2 сумматора и 1 умножитель
3	$y = \sqrt{a^2 + b^2}$	1 сумматор и 2 умножителя
4	$y = \sqrt{a + \sqrt[3]{b}}$	2 сумматора и 1 умножитель
5	$y = 3a + 2 \cdot \sqrt[3]{b}$	1 сумматор и 2 умножителя
6	$y = a \cdot b + a^3$	2 сумматора и 1 умножитель
7	$y = a \cdot \sqrt{b}$	2 сумматора и 2 умножителя
8	$y = a \cdot \sqrt[3]{b}$	1 сумматор и 2 умножителя
9	$y = \sqrt[3]{a} + \sqrt{b}$	2 сумматора и 1 умножитель
10	$y = \sqrt[3]{a + \sqrt[2]{b}}$	2 сумматора и 2 умножителя

## Комментарии к заданиям

В качестве входных данных необходимо использовать целые беззнаковые числа с разрядностью 8 бит.

Разрядность выходного значения выбирается в соответствии с областью допустимых значений функции. Результат вычислений не должен выходить за границы формата представления выходного значения блока.

Ограничения накладываются на количество используемых блоков суммирования и умножения. В разработанной схеме должен быть использован блок умножения, реализующий последовательный алгоритм умножения «в столбик». В качестве основы можно взять реализацию, представленную в разделе 2.4.1.

Сумматор реализуется с помощью встроенного в Verilog оператора суммы «+». Сдвиги реализуются с помощью встроенных операторов сдвига «<<<» и «>>>».

Для вычисления квадратного и кубического корней могут использоваться алгоритмы, представленные в разделе 2.4.

При выполнении заданий необходимо использовать беззнаковую целочисленную арифметику.

## 2.3 Разработка цифровых схем конечных автоматов

Конечные автоматы в цифровой схемотехнике используются для реализации логики управления функциональных блоков. С помощью конечных автоматов задается алгоритм работы функционального блока посредством определения состояний в алгоритме управления и условий переходов между состояниями. Автоматы реализуются в виде цифровых аппаратных блоков и выполняют функции устройства управления в составе функциональных узлов схемы.

Существует множество видов конечных автоматов. На практике для реализации функциональности блоков управления чаще всего используют либо автомат Мура, либо автомат Мили.

Обобщенная структурная схема автомата Мура представлена на рисунке 2.1.

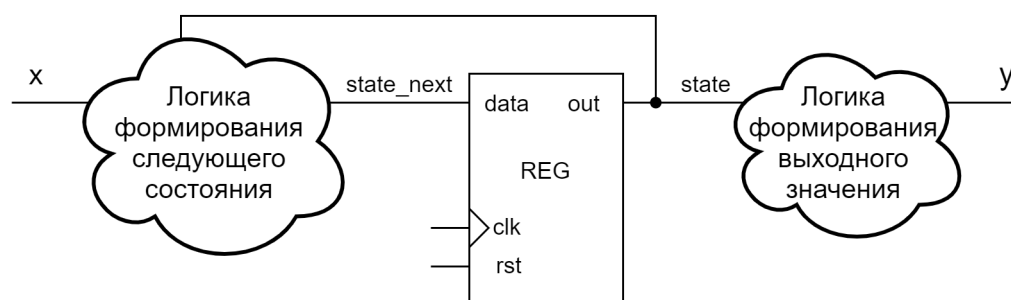


Рисунок 2.1 – Обобщенная схема автомата Мура

Из схемы видно, что выходное значение автомата зависит только от текущего состояния автомата и не зависит напрямую от входных значений. Работа автомата синхронизируется по тактовому сигналу clk. На каждом такте clk происходит запись нового состояния state\_next в регистр состояния. «Логика формирования следующего состояния» и «логика формирования выходного значения» являются комбина-

ционными схемами, то есть не имеют памяти – их выход изменяется при изменении входных сигналов.

Пример графа переходов автомата Мура представлен на рисунке 2.2.

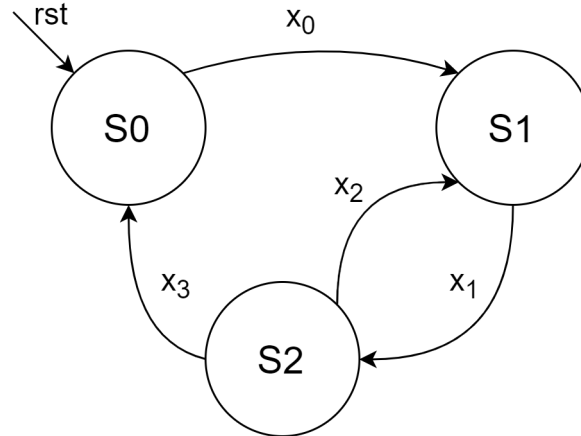


Рисунок 2.2 – Пример графа переходов автомата Мура

В графе кружками обозначаются состояния, а дугами – переходы. Переход осуществляется, если значение входа автомата равно значению, указанному над дугой. В представленном примере вход автомата на каждом такте работы может принимать одно значение из множества  $\{x_0, x_1, x_2, x_3\}$ .

Начальным состоянием является  $S_0$ . В него автомат переходит из любого состояния при активном значении сигнала сброса  $rst$  и формирует на выходе значение  $y_0$ .

Пример таблицы переходов автомата Мура представлен в таблице 2.1.

Таблица 2.1 – Пример таблицы переходов автомата Мура

S/rst,X	rst	$x_0$	$x_1$	$x_2$	$x_3$
$S_0$	$S_0/y_0$	$S_1/y_1$	$S_0/y_0$	$S_0/y_0$	$S_0/y_0$
$S_1$	$S_0/y_0$	$S_1/y_1$	$S_2/y_2$	$S_1/y_1$	$S_1/y_1$
$S_2$	$S_0/y_0$	$S_2/y_2$	$S_2/y_2$	$S_1/y_1$	$S_0/y_0$

В первой строке таблицы перечислены значения входа автомата, включая ситуацию подачи сигнала сброса  $rst$  на автомат, а в первой колонке перечислены возможные состояния. В ячейках таблицы указаны значения следующего состояния и выхода автомата при условии, что в данном состоянии на вход автомата было передано соответствующее значение. Например, если автомат находился в состоянии  $S_0$  и ему на вход было передано  $x_0$ , то он перейдет в состояние  $S_1$  и сформирует на выходе значение  $y_1$ . Это обозначено в таблице как ячейка  $S_1/y_1$  на пересечении столбца  $x_0$  и строки  $S_0$ .

Реализация автомата Мура на языке Verilog HDL представлена в листинге 2.1. В листинге явно выделены структурные части, обозначенные на рисунке 2.1. На



практике возможно объединение разных структурных частей в рамках одних и тех же процедурных блоков.

---

Листинг 2.1 – Описание автомата Мура на Verilog HDL

---

```
module moore(  
    input wire clk ,  
    input wire rst ,  
  
    input wire [1:0] x ,  
    output wire [1:0] y  
);  
  
    localparam S0 = 0, S1 = 1, S2 = 2;  
    localparam X0 = 0, X1 = 1, X2 = 2, X3 = 3;  
  
    reg [1:0] state;  
    reg [1:0] state_next;  
  
    // логика формирования следующего состояния  
    always @*  
        case(state)  
            S0: state_next = (x == X0) ? S1 : S0;  
            S1: state_next = (x == X1) ? S2 : S1;  
            S2: state_next = (x == X2) ? S1 :  
                (x == X3) ? S0 : S2;  
            default: state_next = state;  
        endcase  
  
    // логика формирования выходного значения  
    assign y = state;  
  
    // регистр состояния  
    always @(posedge clk)  
        if (rst) begin  
            state <= S0;  
        end else begin  
            state <= state_next;  
        end  
  
endmodule
```

---

Обобщенная структурная схема автомата Мили представлена на рисунке 2.3.

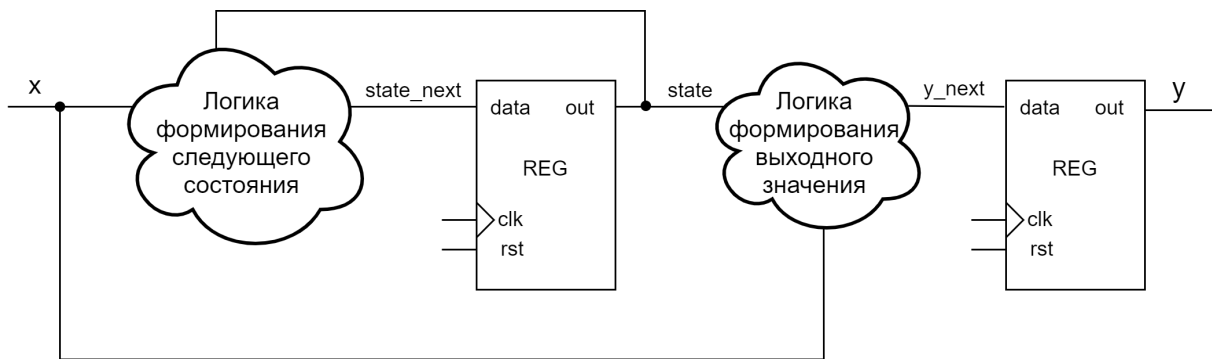


Рисунок 2.3 – Обобщенная схема автомата Мили

В автомате Мили выходное значение автомата зависит не только от текущего состояния, но и от входных значений. Пример графа переходов автомата Мили представлен на рисунке 2.4.

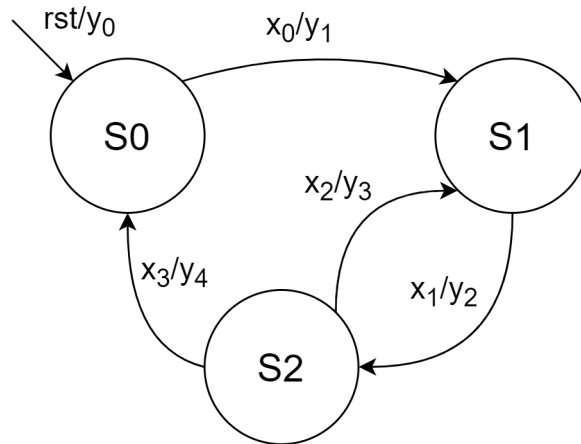


Рисунок 2.4 – Пример графа переходов автомата Мили

Каждый переход автомата Мили аннотируется значением, которое автомат формирует на выходе. Например, переход из состояния  $S_0$  в  $S_1$  происходит, если на вход автомата подано значение  $x_0$ . При этом на выходе формируется значение  $y_1$ . Если же автомат переходит из состояния  $S_2$  в состояние  $S_1$  при наличии на входе  $x_2$ , то на выходе формируется значение  $y_3$ . Таким образом, в одном и том же состоянии  $S_1$  автомат может иметь два разных значения на выходе, а именно  $y_1$  или  $y_3$ .

Пример таблицы переходов автомата Мили представлен в таблице 2.2.  $y_{prev}$  – это значение выхода автомата на предыдущем шаге в соответствующем состоянии. Таким образом, в автомате Мили невозможно узнать выходное значение только по текущему состоянию автомата.

Реализация автомата Мили на языке Verilog HDL представлена в листинге 2.2. В листинге явно выделены структурные части, обозначенные на рисунке 2.3. На

практике возможно объединение разных структурных частей в рамках одних и тех же процедурных блоков.

Таблица 2.2 – Пример таблицы переходов автомата Мили

S/rst,X	rst	$x_0$	$x_1$	$x_2$	$x_3$
S0	S0/ $y_0$	S1/ $y_1$	S0/ $y_{prev}$	S0/ $y_{prev}$	S0/ $y_{prev}$
S1	S0/ $y_0$	S1/ $y_{prev}$	S2/ $y_2$	S1/ $y_{prev}$	S1/ $y_{prev}$
S2	S0/ $y_0$	S2/ $y_{prev}$	S2/ $y_{prev}$	S1/ $y_3$	S0/ $y_4$

Листинг 2.2 – Описание автомата Мили на Verilog HDL

```

module mealy(
    input wire clk ,
    input wire rst ,

    input wire [1:0] x ,
    output reg [2:0] y
);

    localparam S0 = 0, S1 = 1, S2 = 2;

    localparam X0 = 0, X1 = 1, X2 = 2, X3 = 3;

    localparam Y0 = 0, Y1 = 1, Y2 = 2,
                Y3 = 3, Y4 = 4;

    reg [1:0] state;
    reg [1:0] state_next;
    reg [2:0] y_next;

    // логика формирования следующего состояния
    always @*
        case (state)
            S0: state_next = (x == X0) ? S1 : S0;
            S1: state_next = (x == X1) ? S2 : S1;
            S2: state_next = (x == X2) ? S1 :
                            (x == X3) ? S0 : S2;
            default: state_next = state;
        endcase

```

```

// логика формирования выходного значения
always @* begin
    y_next = y;
    case (state)
        S0: if (x == X0) y_next = Y1;
        S1: if (x == X1) y_next = Y2;
        S2: begin
            if (x == X2) y_next = Y3;
            if (x == X3) y_next = Y4;
        end
        default: y_next = y;
    endcase
end

// регистр выходного порта y
always @(posedge clk)
    if (rst) begin
        y <= Y0;
    end else begin
        y <= y_next;
    end
end

// регистр состояния
always @(posedge clk)
    if (rst) begin
        state <= S0;
    end else begin
        state <= state_next;
    end
end

endmodule

```

---

Автомат Мили может быть однозначно преобразован в автомат Мура, и наоборот. На рисунке 2.5 представлен граф переходов автомата Мура, реализующий функциональность автомата Мили, изображенного на рисунке 2.4. Таблица переходов получившегося автомата Мура представлена в таблице 2.3.

Как видно из рисунка 2.5, чтобы избежать неоднозначности в определении выхода, исходные состояния  $S0$  и  $S1$  были разбиты на два:  $S0.1$ ,  $S0.2$  и  $S1.1$ ,  $S1.2$

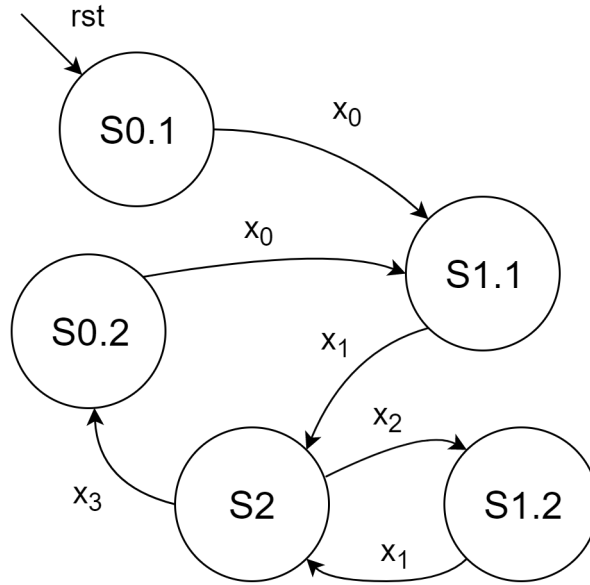


Рисунок 2.5 – Граф переходов автомата Мура, эквивалентного автомату Мили, представленному на рисунке 2.4

Таблица 2.3 – Таблица переходов автомата Мура, эквивалентного автомату Мили, представленному на рисунке 2.4

S/rst,X	rst	$x_0$	$x_1$	$x_2$	$x_3$
S0.1	S0.1/ $y_0$	S1.1/ $y_1$	S0.1/ $y_0$	S0.1/ $y_0$	S0.1/ $y_0$
S0.2	S0.1/ $y_0$	S1.1/ $y_1$	S0.2/ $y_4$	S0.2/ $y_4$	S0.2/ $y_4$
S1.1	S0.1/ $y_0$	S1.1/ $y_1$	S2/ $y_2$	S1.1/ $y_1$	S1.1/ $y_1$
S1.2	S0.1/ $y_0$	S1.2/ $y_3$	S2/ $y_2$	S1.2/ $y_3$	S1.2/ $y_3$
S2	S0.1/ $y_0$	S2/ $y_2$	S2/ $y_2$	S1.2/ $y_3$	S0.2/ $y_4$

соответственно. Теперь выходное значение автомата полностью определяется его состоянием, как показано в таблице 2.3.

Таким образом, граф переходов автомата Мили имеет более компактный вид по сравнению с графом автомата Мура, но вместе с этим более сложную логику формирования выходного значения, так как необходимо учитывать не только текущее состояние автомата, но и значение его входа.

## 2.4 Алгоритмы машинной арифметики

В цифровой технике более сложные арифметические операции, такие как умножение, деление, извлечение корня и др., сводятся к более простым (сложению, вычитанию и сдвигу) и реализуются с помощью базовых операционных элементов цифровой схемотехники.

В данном разделе будут рассмотрены алгоритмы умножения, извлечения квадратного и кубического корней. Понимание данных алгоритмов будет необходимо для успешного выполнения задания лабораторной работы. Для алгоритма умножения будет приведен пример его реализации в виде аппаратного блока.

### 2.4.1 Алгоритм умножения

Рассмотрим алгоритм умножения «в столбик». Пример умножения двух 4-разрядных беззнаковых чисел приведен на рисунке 2.6.

$$\begin{array}{r} \phantom{000} 1 \ 0 \ 1 \ 1 \\ * \phantom{000} 0 \ 1 \ 1 \ 0 \\ \hline \phantom{000} 0 \ 0 \ 0 \ 0 \\ + \phantom{000} 1 \ 0 \ 1 \ 1 \\ + \phantom{00} 1 \ 0 \ 1 \ 1 \\ + \phantom{000} 0 \ 0 \ 0 \ 0 \\ \hline 1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \end{array}$$

Рисунок 2.6 – Пример умножения «в столбик»

В примере перемножаются числа  $1011_2 = 11_{10}$  и  $0110_2 = 6_{10}$ . Произведение получилось равным  $1000010_2 = 66_{10}$ . В процессе вычисления результата выполнялось умножение каждого разряда второго операнда на первый операнд и сложение частных результатов. Умножение на 0 и 1 выполняется простейшим образом: разряды второго операнда можно рассматривать как условие, определяющее необходимость сложения первого операнда с полученным на предыдущих шагах частным результатом. Каждое сложение выполняется со сдвигом на один разряд относительно предыдущего шага.

Важно, что для сохранения результата требуется отводить регистр, имеющий разрядность вдвое большую, чем разрядность операндов. Это необходимо, чтобы избежать переполнения формата и получения неверного результата.

Схема алгоритм умножения «в столбик» представлена на рисунке 2.7.

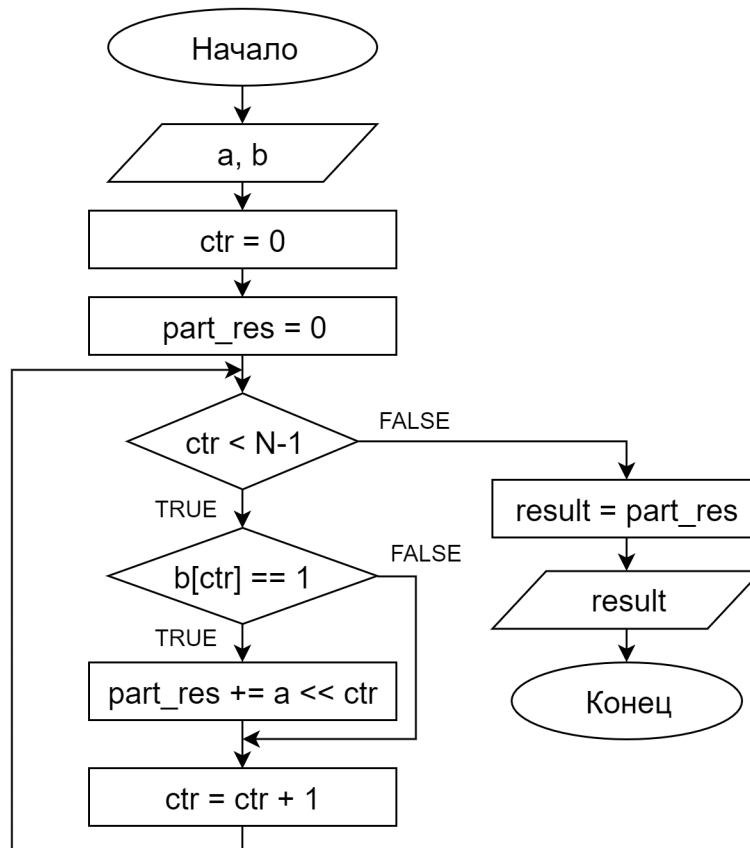


Рисунок 2.7 – Схема алгоритма умножения «в столбик»

На рисунке 2.7 символом  $N$  обозначена разрядность операндов  $a$  и  $b$ . Частная сумма на каждом шаге помещается в переменную  $part\_res$ . Переменная  $ctr$  используется как счетчик шагов алгоритма. С помощью  $b[ctr]$  обозначается выборка бита с номером  $ctr$  из второго операнда  $b$ . Нумерация битов начинается с 0. В конце расчета результат помещается в переменную  $result$ .

Теперь спроектируем аппаратный блок умножения. Сначала определим интерфейс блока. Так как вычисления выполняются за несколько шагов, то необходим сигнал, по которому блок должен начать вычисления ( $start\_i$ ), сигнал, определяющий занятость блока вычислениями ( $busy\_o$ ), а также сигнал сброса ( $rst\_i$ ) и тактовый сигнал ( $clk\_i$ ). Тактовый сигнал или сигнал синхронизации необходим для реализации пошагового алгоритма – каждый шаг будет выполняться с приходом очередного положительного фронта тактового сигнала. Помимо этого, понадобятся входные шины данных для передачи значений операндов ( $a\_bi$  и  $b\_bi$ ) к блоку,

а также одна выходная шина ( $y\_bo$ ) для передачи результатов вычислений. Для примера примем, что операнды представлены беззнаковыми 8-разрядными целыми числами. Выходная шина данных при этом будет иметь разрядность в 16 бит, то есть вдвое большую, чем разрядность операндов. Интерфейс проектируемого блока представлен на рисунке 2.8.



Рисунок 2.8 – Интерфейс аппаратного блока умножения

Устройство управления ходом вычислительного процесса удобно выполнить как конечный автомат Мили, граф переходов которого представлен на рисунке 2.9.

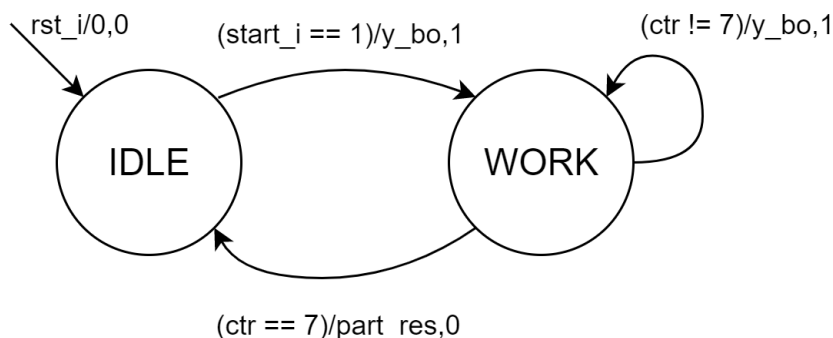


Рисунок 2.9 – Граф переходов автомата устройства управления блока умножения

Управляющий автомат имеет два состояния: ожидание (IDLE) и работа (WORK). В состоянии IDLE блок ожидает прихода управляющего сигнала  $start\_i$  и переходит к началу вычислений. Все шаги алгоритма реализуются в состоянии WORK.

В представленном графе над каждым переходом указано условие перехода и через косую черту значение выходных сигналов  $y\_bo$  и  $busy\_o$ . Так, например, переход из состояния IDLE в состояние WORK производится по выполнению условия  $(start\_i == 1)$ . На выходе  $y\_bo$  при этом сохраняется предыдущее значение, а сигнал  $busy\_o$  принимает значение 1.

Реализация алгоритма умножения «в столбик» на языке Verilog HDL представлена в листинге 2.3. В представленном описании логика формирования следующего состояния, регистр состояния и логика формирования выходных значений автомата Мили объединены в один процедурный блок.



```
module mult(  
    input clk_i ,  
    input rst_i ,  
  
    input [7:0] a_bi ,  
    input [7:0] b_bi ,  
    input start_i ,  
  
    output busy_o ,  
    output reg [15:0] y_bo  
);  
  
    localparam IDLE = 1'b0;  
    localparam WORK = 1'b1;  
  
    reg [2:0] ctr;  
    wire [2:0] end_step;  
    wire [7:0] part_sum;  
    wire [15:0] shifted_part_sum;  
    reg [7:0] a, b;  
    reg [15:0] part_res;  
    reg state;  
  
    assign part_sum = a & {8{b[ctr]}};  
    assign shifted_part_sum = part_sum << ctr;  
    assign end_step = (ctr == 3'h7);  
    assign busy_o = state;  
  
    always @(posedge clk_i)  
        if (rst_i) begin  
            ctr <= 0;  
            part_res <= 0;  
            y_bo <= 0;  
  
            state <= IDLE;  
        end else begin
```

```

    case (state)
      IDLE:
        if (start_i) begin
          state <= WORK;

          a      <= a_bi;
          b      <= b_bi;
          ctr    <= 0;
          part_res <= 0;
        end
      WORK:
        begin
          if (end_step) begin
            state <= IDLE;
            y_bo <= part_res;
          end

          part_res <= part_res + shifted_part_sum;
          ctr <= ctr + 1;
        end
      endcase
    end
  endmodule

```

---

## 2.4.2 Алгоритм извлечения квадратного корня

В данном разделе представлен целочисленный алгоритм извлечения квадратного корня с округлением до ближайшего меньшего целого числа.

$$y = \lfloor \sqrt{x} \rfloor$$

Округление до ближайшего меньшего числа означает, что если мы извлекаем квадратный корень из числа, для которого квадратный корень не является целым, то в качестве результата берется наименьшая целая часть. Например, при извлечении квадратного корня из 10 получим 3,16, и в качестве результата берем 3.

$$\lfloor \sqrt{10} \rfloor = \lfloor 3,16 \rfloor = 3$$

Схема алгоритма вычисления квадратного корня с использованием только операций сложения, вычитания, сравнения и сдвига представлена на рисунке 2.10. В схеме алгоритма символом  $N$  обозначена разрядность входного операнда  $x$ .

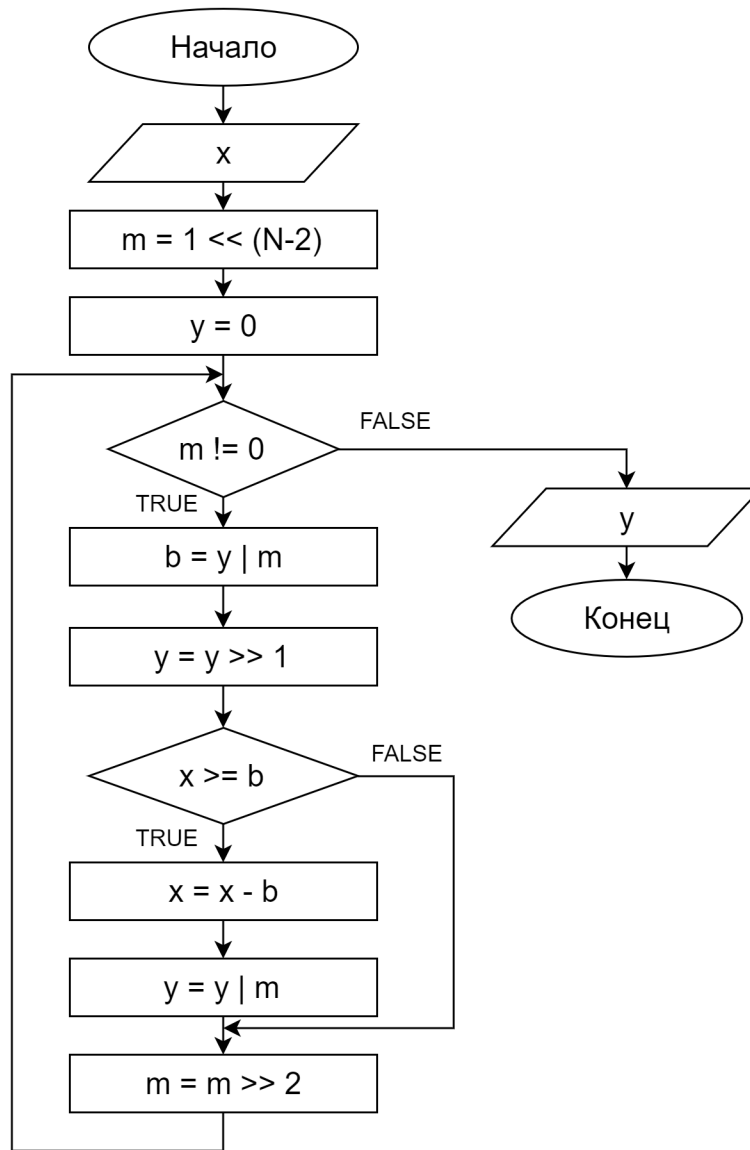


Рисунок 2.10 – Схема алгоритма вычисления квадратного корня

### 2.4.3 Алгоритм извлечения кубического корня

В данном разделе представлен целочисленный алгоритм извлечения кубического корня с округлением до ближайшего меньшего целого числа.

$$y = \lfloor \sqrt[3]{x} \rfloor$$

Схема алгоритма вычисления кубического корня с использованием только операций умножения, сложения, вычитания, сравнения и сдвига представлена на рисунке 2.11. В схеме алгоритма символом N обозначена разрядность входного операнда x.

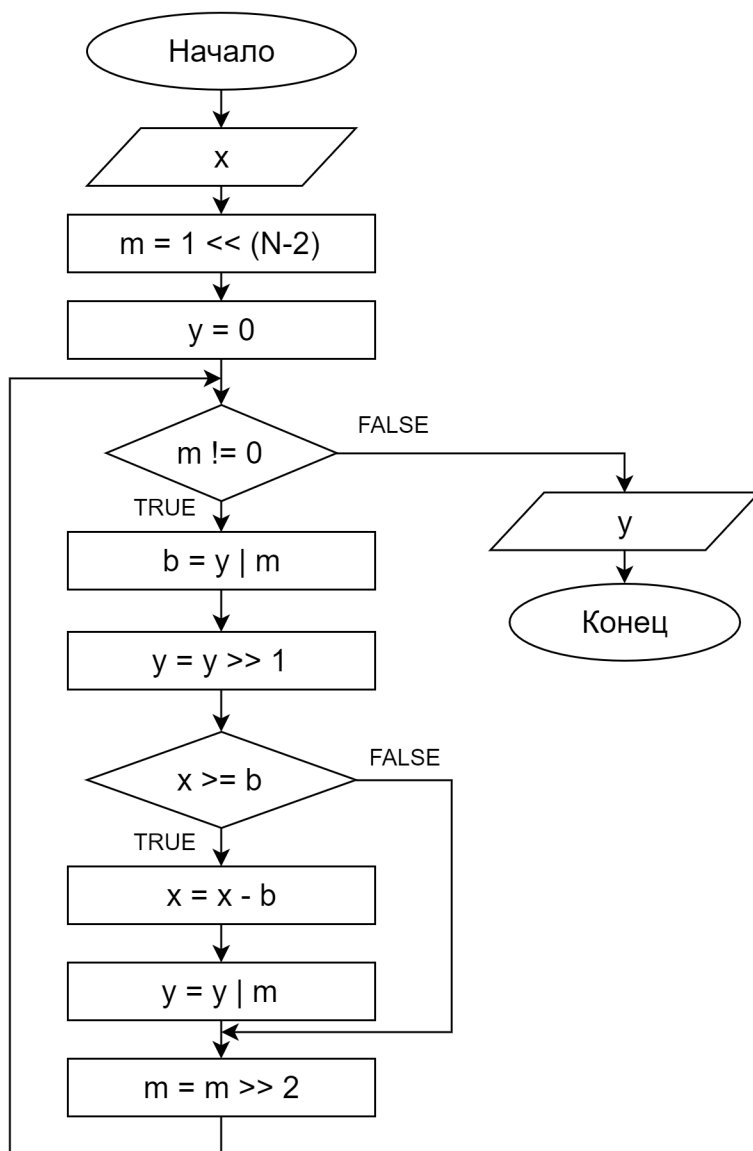


Рисунок 2.11 – Схема алгоритма вычисления кубического корня

## **3 Лабораторная работа №3. Проектирование цифровых схем с использованием ПЛИС**

### **3.1 Введение**

Лабораторная работа №3 посвящена получению навыков прототипирования цифровых устройств с использованием микросхем программируемой логики (ПЛИС).

В работе предлагается запустить цифровой блок ускорения математических операций, разработанный в лабораторной работе №2, на отладочной плате Nexys 4 DDR с ПЛИС XC7A100T-1CSG324C. Для успешного запуска проект должен быть написан на синтезируемом подмножестве языка Verilog HDL и содержать корректные описания базовых операционных элементов цифровой схемотехники.

В процессе выполнения студенты учатся связывать проект с реальным внешним окружением посредством подключения портов ввода/вывода проекта к реальным ножкам микросхемы ПЛИС. Таким образом, студенты получают базовые навыки работы с дискретными портами ввода/вывода, подключенными к светодиодам, движковым переключателям и кнопкам на используемой отладочной плате.

### **3.2 Описание лабораторной работы**

#### **Цель работы**

Получить навыки разработки цифровых устройств на базе программируемых логических интегральных схем (ПЛИС).

#### **Указания к выполнению работы**

Работа выполняется в Vivado Design Suite и с использованием отладочной платы Nexys 4 DDR (новое название Nexys A7). Пример создания и загрузки файла конфигурации (прошивки) ПЛИС описан в разделе 3.4.

#### **Порядок выполнения работы**

1. Доработайте схему функционального блока, разработанного в лабораторной работе №2, в соответствии с рисунком 3.1. Необходимо добавить возможность работы с блоком посредством дискретных портов ввода/вывода, подключенных к переключателям, светодиодам и кнопкам платы Nexys 4 DDR:

- значения операндов должны вводиться с помощью переключателей (SW);
  - результат должен выводиться на светодиоды (LEDS);
  - с целью повышения удобства работы пользователя допускается использование дополнительных кнопок, переключателей и светодиодов;
  - интерфейс пользователя должен обеспечивать возможность многократного проведения вычислений без постоянного нажатия на кнопку сброса.
2. Разработайте тестовое окружение и проведите моделирование.
  3. Проведите синтез и размещение схемы для ПЛИС XC7A100T-1CSG324C, входящей в состав отладочной платы Nexys 4 DDR.
  4. Определите количество и тип используемых ресурсов ПЛИС после размещения схемы.
  5. Проверьте работоспособность схемы на отладочной плате Nexys 4 DDR.
  6. Составьте отчет по результатам выполнения работы.

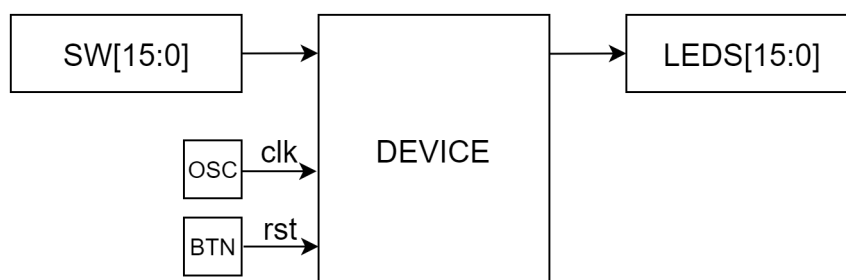


Рисунок 3.1 – Схема сопряжения функционального блока с дискретными портами ввода/вывода и источником синхросигнала: SW – переключатели, LEDS – светодиоды, BTN – тактовая кнопка, OSC – генератор синхросигнала

## Требования к отчету

Отчет должен быть оформлен в соответствии с требованиями, представленными в Приложении А.

### Отчет должен включать:

1. Титульный лист.
2. Цель работы.
3. Задание в соответствии с вариантом.
4. Схему (рисунок) сопряжения разработанного блока и устройств ввода/вывода (переключателей, светодиодов, кнопок и др.). Копирование схемы из задания не допускается. На схеме должно быть указано актуальное количество используемых устройств ввода/вывода.
5. Описание алгоритма работы пользователя. Приводится схема алгоритма и краткое описание порядка взаимодействия пользователя с устройством, то есть указывается, какие кнопки/переключатели в какой последовательности нажимать/переключать, где смотреть и как интерпретировать результат работы.
6. Результат тестирования блока в симуляторе (временные диаграммы).
7. Таблицу с данными об использовании ресурсов ПЛИС.
8. Выводы по работе.

## 3.3 Архитектура и принцип работы микросхем программируемой логики

Программируемая логическая интегральная схема (ПЛИС; англ. programmable logic device, PLD) — это интегральная схема с конфигурируемой логикой работы.

В отличие от обычных цифровых микросхем, логика работы ПЛИС не определяется при изготовлении, а задаётся посредством программирования. При этом под программой понимается не последовательность инструкций, которые надо исполнить, а конфигурация реакции внутренних логических элементов на изменения входных сигналов, а также схема соединений этих элементов.

Для программирования используется специальное аппаратное (программатор) и программное обеспечение, позволяющее задать желаемую структуру цифрового устройства в виде схемы или описания на специальных языках описания аппаратуры: Verilog, VHDL и др.

В англоязычной литературе для микросхем ПЛИС существует два различных термина:

- CPLD (Complex Programmable Logic Device) – разновидность ПЛИС, в которых конфигурируются соединения между логическими элементами с уже определенной функциональностью;
- FPGA (Field-Programmable Gate Array) – разновидность ПЛИС, в которых конфигурируются соединения между логическими элементами и функциональность самих элементов.

В настоящее время граница между CPLD и FPGA всё больше стирается. Некоторые семейства CPLD и FPGA отличаются лишь возможностью хранения конфигурации в энергонезависимой памяти. Микросхемы CPLD обычно содержат меньше логических элементов, и файл конфигурации занимает меньший объем памяти, чем конфигурация микросхем FPGA. Поэтому интеграция конфигурационной памяти на кристалл микросхемы CPLD является не очень затратной по ресурсам, а микросхемы FPGA требуют внешней памяти для хранения своей конфигурации. Возможность конфигурирования функциональности логических элементов при этом присутствует и там, и там, хотя имеются исключения. Кроме этого, некоторые производители микросхем FPGA интегрируют на кристалл конфигурационную память для микросхем средней емкости. Например, Xilinx так поступала в случае своей линейки FPGA Spartan-3AN, но это не широко распространенная ситуация.

Далее, говоря о ПЛИС, мы будем подразумевать микросхемы типа FPGA, так как они чаще используются для реализации сложных цифровых устройств (процессоров, аппаратных ускорителей и пр.). Базовыми элементами микросхем ПЛИС являются:

- конфигурируемые логические блоки или макроячейки, реализующие требуемую логическую функцию;
- программируемые линии связи между макроячейками;
- программируемые блоки ввода/вывода, обеспечивающие связь вывода микросхемы с внешним миром;
- блоки памяти;
- аппаратные IP-ядра (Intellectual Property core, IP-core) – логические блоки или блоки данных, которые реализуют законченную функциональность (процессор, Ethernet-контроллер, контроллер PCIe и пр.); также могут быть программируемыми и конфигурируемыми.

Пример структуры микросхемы ПЛИС представлен на рисунке 3.2. На рисунке представлена ПЛИС с конфигурируемыми макроячейками, встроенным процессор-



ным ядром, блоками памяти BRAM, DSP-блоками для быстрого выполнения операций умножения и сложения, высокоскоростными приемопередатчиками.

Аппаратные IP-блоки контроллеров ввода/вывода обычно включаются в более дорогие устройства ПЛИС. Это позволяет сэкономить ресурсы конфигурируемых макроячеек и использовать уже готовый блок для организации взаимодействия с внешними устройствами.

Структурная схема конфигурируемой макроячейки представлена на рисунке 3.3.

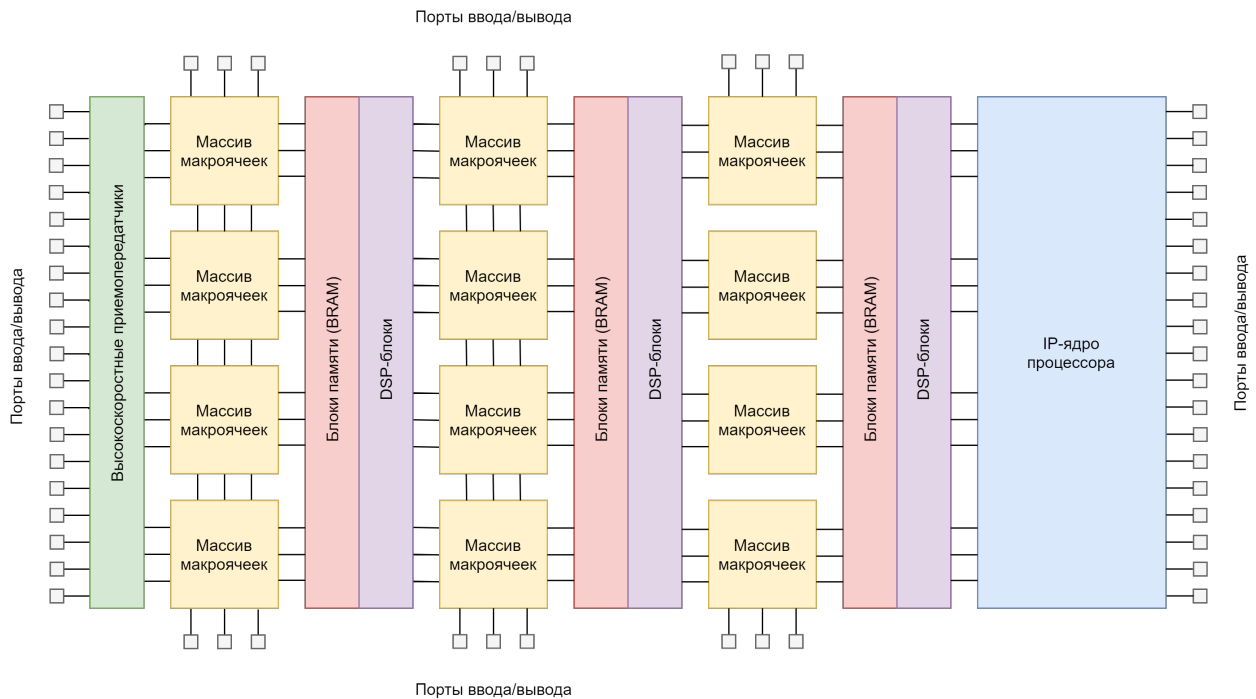


Рисунок 3.2 – Пример структуры микросхемы ПЛИС

Внутренняя структура макроячейки определяется фирмой-производителем ПЛИС и может включать различные компоненты. Можно выделить основные компоненты, присущие большинству видов макроячеек:

- LUT – Look-Up Table (таблица преобразования);
- D-триггер (FF, flip-flop).

С помощью LUT-ячеек реализуется комбинационная логика. Структурная схема LUT представлена на рисунке 3.4.

LUT представляет собой схему из регистров и мультиплексоров. Входы каждого мультиплексора подключаются к одноразрядным регистрам  $D_0, D_1, \dots$ . Эти регистры хранят значения, которые будут поданы на выход при определенной комбинации управляющих сигналов. Входы LUT являются управляющими сигналами внутренних мультиплексоров. Таким образом, перезаписывая содержимое регистров  $D_0, D_1, \dots$ , мы можем менять реакцию LUT на значения входных сигналов и тем самым реализовывать различные элементы цифровой схемотехники.

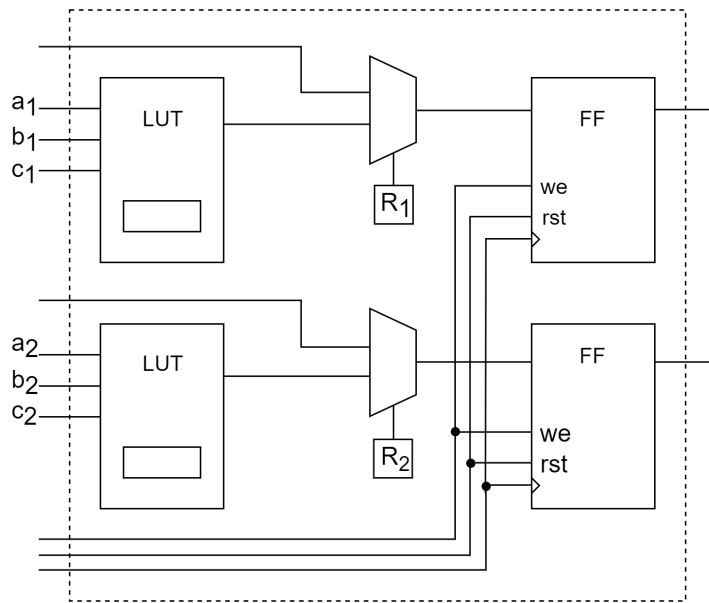


Рисунок 3.3 – Схема конфигурируемой макроячейки

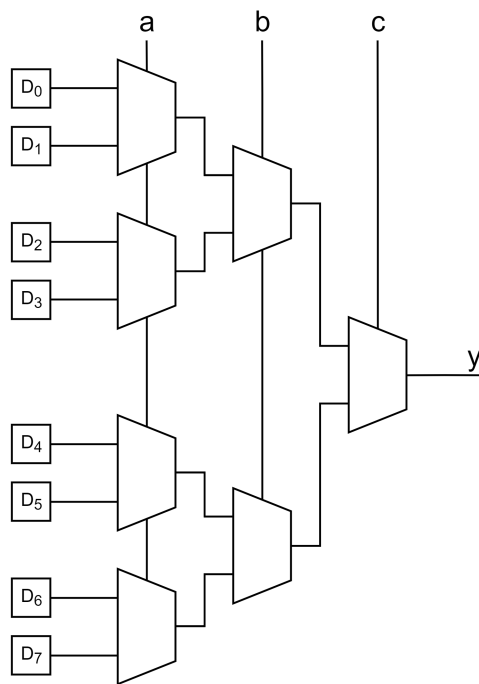


Рисунок 3.4 – Схема внутреннего устройства трехвходового LUT

Пример реализации комбинационной схемы с использованием таблицы LUT представлен на рисунке 3.5.

С помощью D-триггеров реализуется последовательная логика, то есть элементы с памятью: регистры, конечные автоматы, счетчики и др. Однобитовые регистры конфигурации  $R_1, R_2$  необходимы для соединения выхода LUT с триггерами. Сигналы с этих однобитовых регистров поступают на управляющие входы коммутационных мультиплексоров.

Разные микросхемы ПЛИС отличаются друг от друга количеством LUT и триггеров в макроячейке, а также количеством входов каждого LUT.

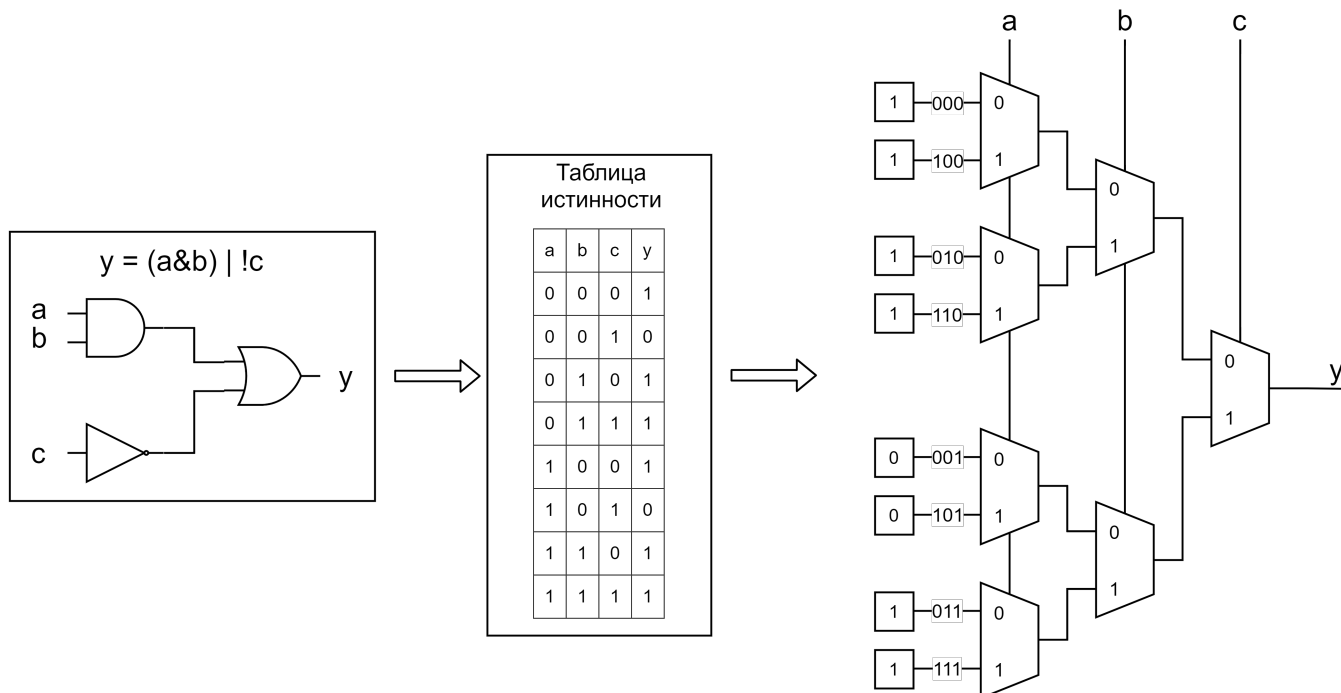


Рисунок 3.5 – Пример конфигурации LUT на выполнение функции комбинационной схемы

На сегодняшний день можно выделить две основные архитектуры ПЛИС: островная (island-style или mesh-based FPGA) и иерархическая.

Островная архитектура представлена на рисунке 3.6 и характерна для большинства ПЛИС.

На рисунке 3.6 используются следующие обозначения:

- CLB (Configurable Logic Block) – конфигурируемая макроячейка;
- СВ (Connection Box) – это набор программируемых мультиплексоров, подключающихся непосредственно к входам и выходам блоков CLB, а также портам ввода/вывода;
- СВ (Switch Box) – это набор программируемых мультиплексоров, позволяющих настроить связь между блоками СВ.

Таким образом, блоки СВ и СВ позволяют настроить соединение конфигурируемых блоков CLB.

В островных ПЛИС все макроячейки равнозначны и соединены одинаковым способом. Они рассматриваются подобно островам в океане конфигурируемых узлов коммутации (блоки СВ и СВ) и линий связи (линии, соединяющие СВ и СВ). Отсюда и название – «островная архитектура».

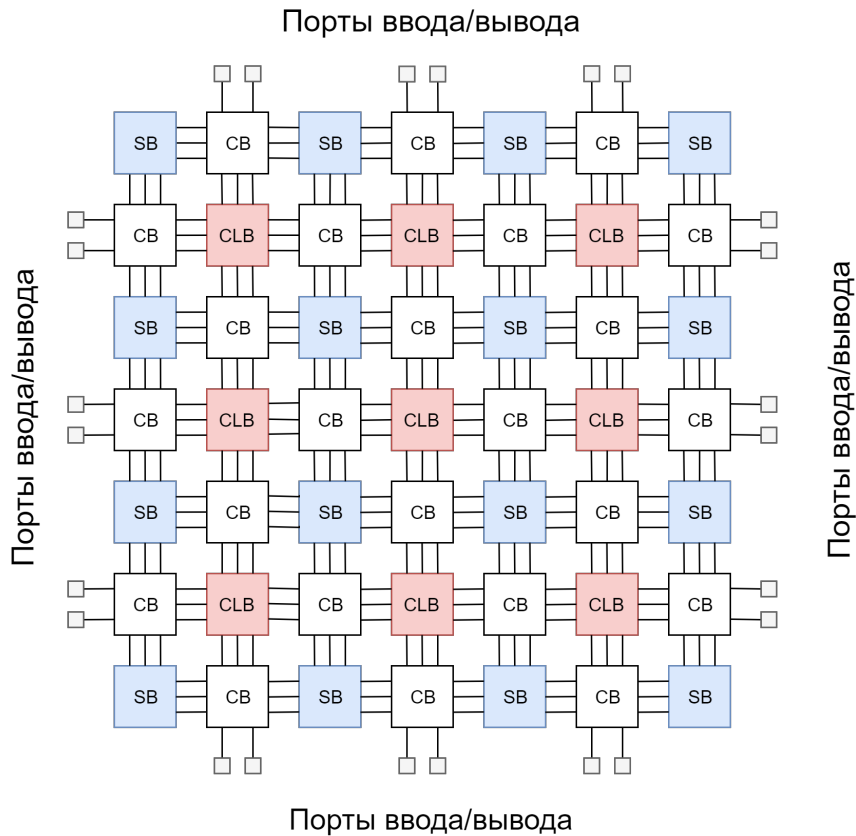


Рисунок 3.6 – Островная архитектура ПЛИС

Островная архитектура характерна для микросхем фирмы Xilinx, а также семейств Cyclone и Stratix фирмы Intel (бывшая Altera).

Иерархическая архитектура представлена на рисунке 3.7. В ней сделан расчет на то, что, располагая часто обменивающиеся данными блоки ближе друг к другу, можно выиграть в количестве используемых ресурсов и производительности системы. Всё пространство конфигурируемых блоков делится на кластеры (комнаты). Внутри кластеров конфигурируемые ячейки соединены тесной сетью линий связи и узлов коммутации, а между кластерами связь организуется с помощью небольшого количества коммутационных узлов. Таким образом, наилучшая скорость передачи данных обеспечивается там, где это необходимо, то есть внутри кластеров. А при организации связи между самими кластерами удастся сэкономить ресурсы коммуникационной инфраструктуры.

Иерархическая архитектура характерна для старых семейств Flex10K и APEX фирмы Intel (бывшая Altera).

Информацию о типе архитектуры ПЛИС можно найти в документации на соответствующую микросхему. Однако названия блоков одинакового назначения могут быть разными из-за маркетинговой политики фирм-производителей.

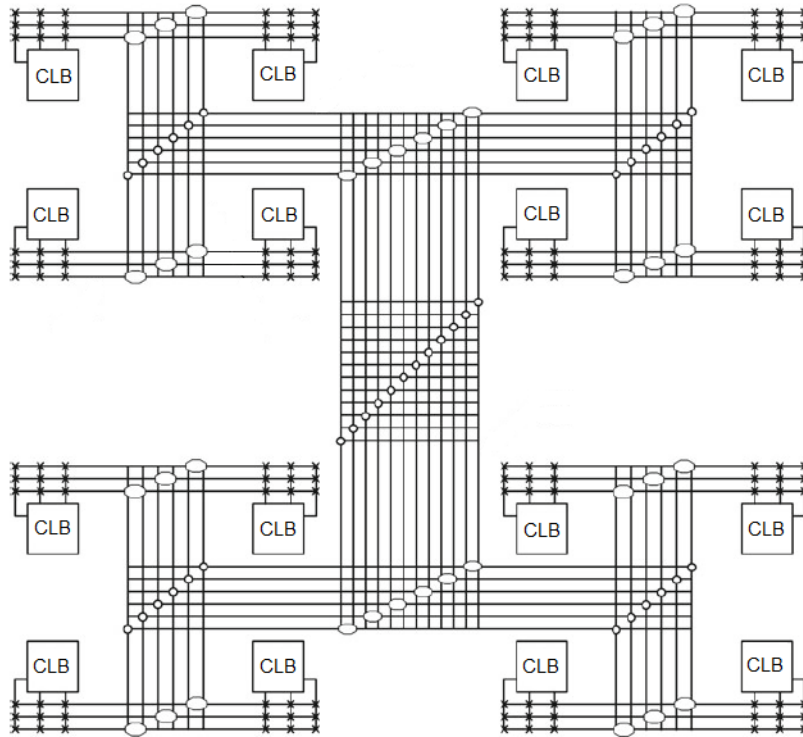


Рисунок 3.7 – Иерархическая архитектура ПЛИС

### 3.4 Пример создания и загрузки файла конфигурации ПЛИС в среде Vivado Design Suite

В данном разделе описывается процесс создания файла конфигурации (прошивки) для ПЛИС XC7A100T-1CSG324C, входящей в состав отладочной платы Nexys 4 DDR, а также последующая загрузка файла в устройство с помощью Vivado Design Suite.

Процесс создания файла конфигурации для ПЛИС в общем случае включает следующие основные этапы:

- логический синтез;
- отображение схемы на ресурсы ПЛИС, физическое размещение;
- генерация файла.

Иногда весь процесс называют термином «синтез» или «компиляция», но необходимо понимать, что он содержит много шагов, которые нельзя путать. В системах разных производителей эти шаги могут называться по-разному, но суть остается почти одинаковой.

Перейдем к созданию прошивки. Предполагается, что проект устройства уже создан и отлажен в симуляторе. Следующим шагом является логический синтез схемы,

в процессе которого описание на языке Verilog HDL переводится в базис логических примитивов (базовых операционных элементов, вентилях и т.п.).

Для запуска логического синтеза необходимо в левой панели Vivado Design Suite нажать на кнопку *Run Synthesis* (рисунок 3.8).



Рисунок 3.8 – Кнопка запуска процесса синтеза

После нажатия на кнопку появится диалоговое окно с настройками синтеза (рисунок 3.9). В окне настроек можно указать, в какое количество потоков (jobs) будет производиться синтез. Для продолжения синтеза необходимо нажать *ОК*.

После завершения синтеза откроется диалоговое окно с выбором следующего действия (рисунок 3.10). При первом выполнении синтеза необходимо открыть синтезированный проект, чтобы установить соответствие виртуальных портов ввода/вывода, имеющих в проекте, реальным портам ПЛИС. Для этого в открывшемся окне выбираем *Open Synthesized Design* и нажимаем *ОК*.

Чтобы перейти к редактированию назначения портов, надо в правом верхнем углу открывшегося окна синтезированного проекта выбрать режим *I/O Planning* (рисунок 3.11).

В режиме *I/O Planning* необходимо задать соответствия портов в нижней части окна проекта (область выделена красным на рисунке 3.12). Для всех дискретных

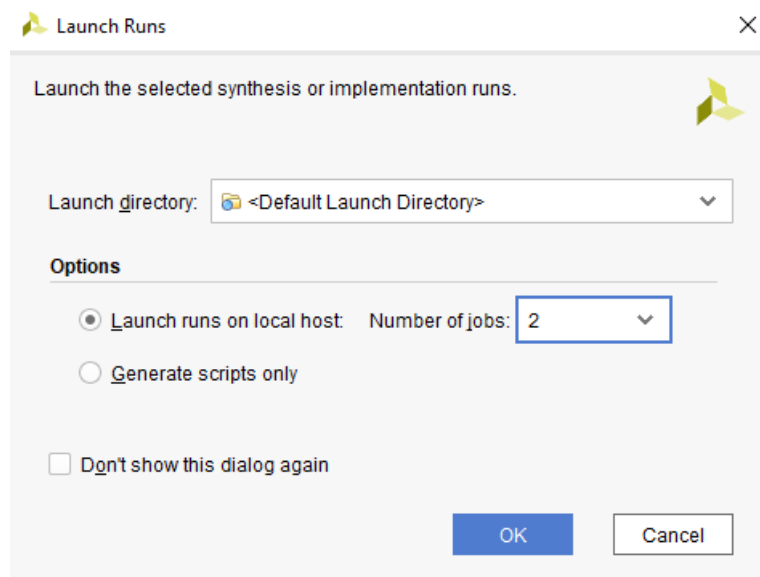


Рисунок 3.9 – Диалоговое окно настройки синтеза

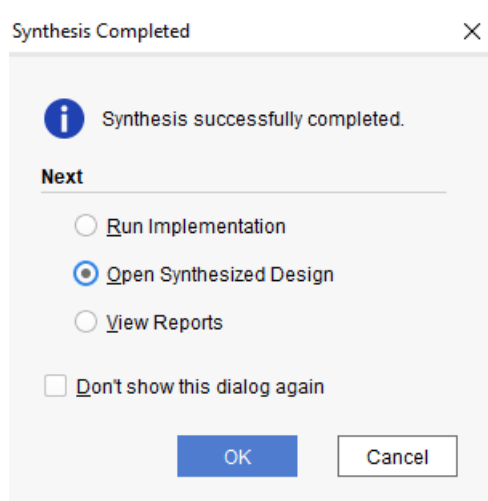


Рисунок 3.10 – Диалоговое окно открытия синтезированного проекта

портов ввода/вывода (переключатели, светодиоды и кнопки) необходимо в качестве значения параметра *IO Std* задавать «LVCMOS33».

Информацию о названии реальных ножек ПЛИС и о том, к чему они подключены на плате, можно взять из официальной документации к плате. Подключение дискретных портов ввода/вывода представлено на рисунке 3.13.

После завершения редактирования привязок необходимо сохранить изменения. Это можно сделать комбинацией клавиш «Ctrl+S». При первом сохранении будет показано окно с предложением ввести имя для вновь создаваемого файла ограничений, в котором будут сохранены установленные настройки портов (рисунок 3.14). Вводим имя и сохраняем ограничения в файл XDC.

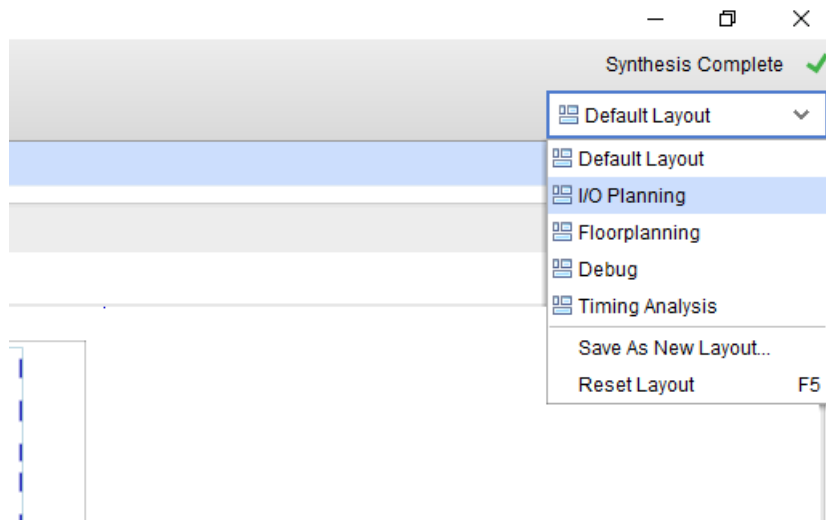


Рисунок 3.11 – Выбор режима I/O Planning

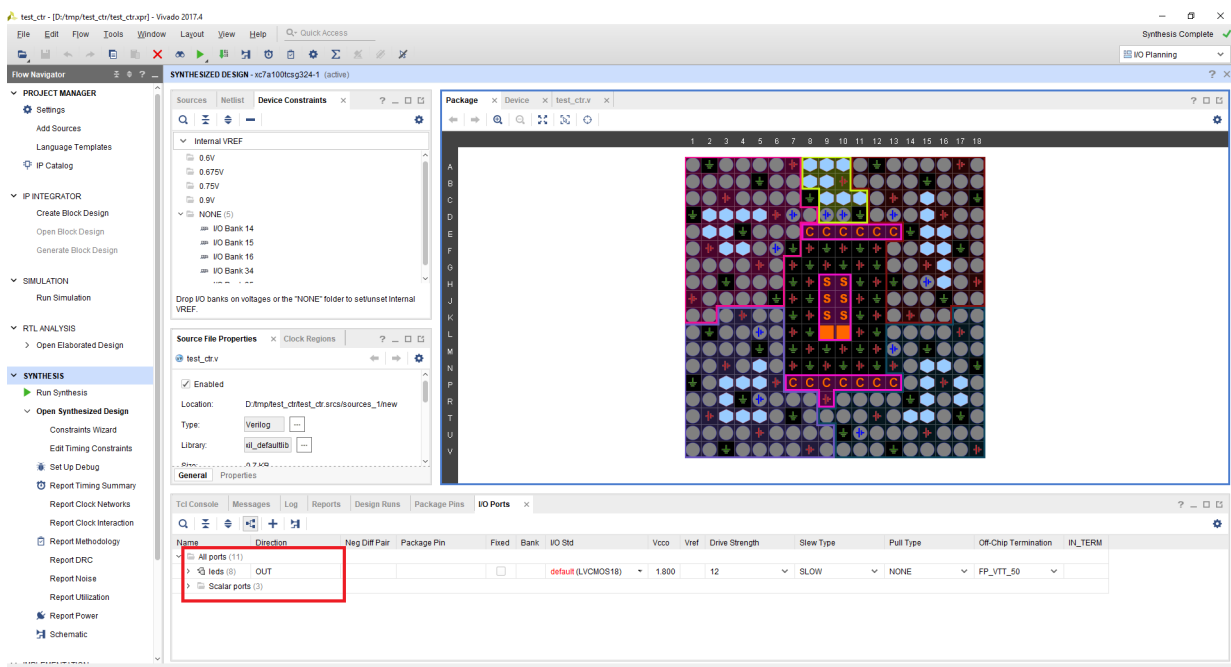


Рисунок 3.12 – Интерфейс редактирования привязки портов/ввода вывода

Заданные ограничения можно просмотреть в текстовом виде, щелкнув дважды на названии файла в панели исходных файлов проекта (рисунок 3.15).

Нам необходимо также добавить ограничение на частоту тактового сигнала. На рисунке 3.16 необходимая строчка выделена цветом. Выставляем ограничение на период в 10 нс, что соответствует 100 МГц. Эта информация необходима Vivado для правильного временного анализа схемы после размещения её компонентов на ПЛИС. Если временные ограничения не будут соблюдаться, Vivado выдаст предупреждение.



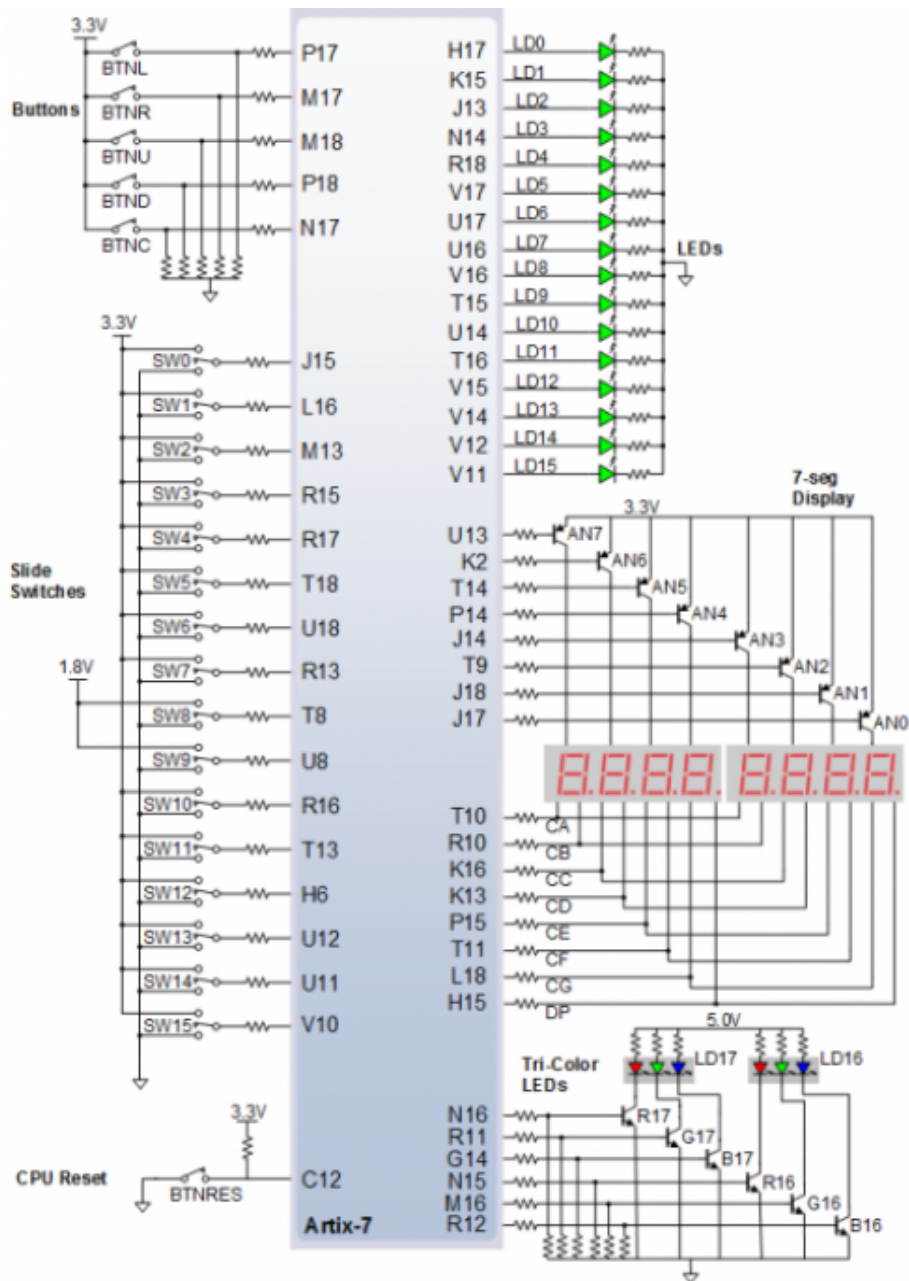


Рисунок 3.13 – Схема подключения дискретных портов ввода/вывода ПЛИС XC7A100T-1CSG324C на плате Nexys 4 DDR

После задания ограничений необходимо перейти к этапу отображения схемы на ресурсы ПЛИС и физическому размещению. Эти шаги запускаются нажатием на *Run Implementation* в левой части интерфейса проекта Vivado (рисунок 3.17).

После успешного размещения схемы на ПЛИС будет предложено выполнить генерацию файла прошивки. Выбираем *Generate Bistream* и нажимаем *OK* (рисунок 3.18).

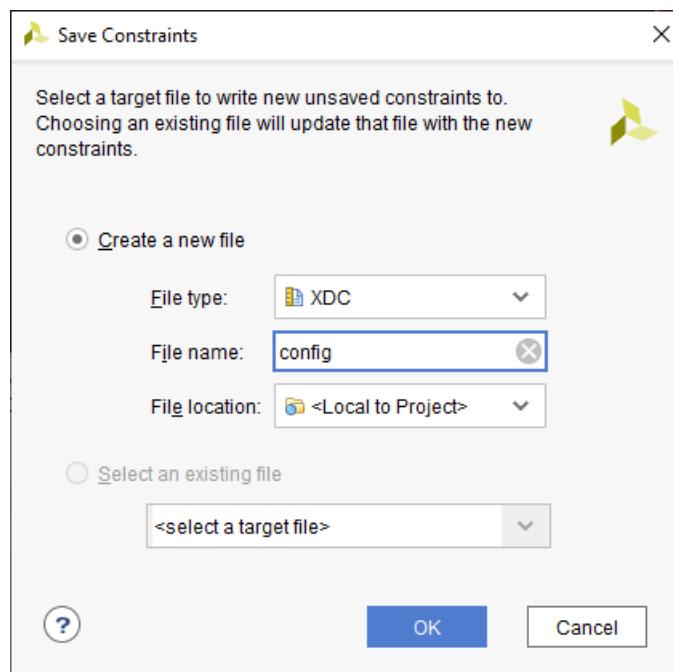


Рисунок 3.14 – Диалоговое окно сохранения ограничений проекта

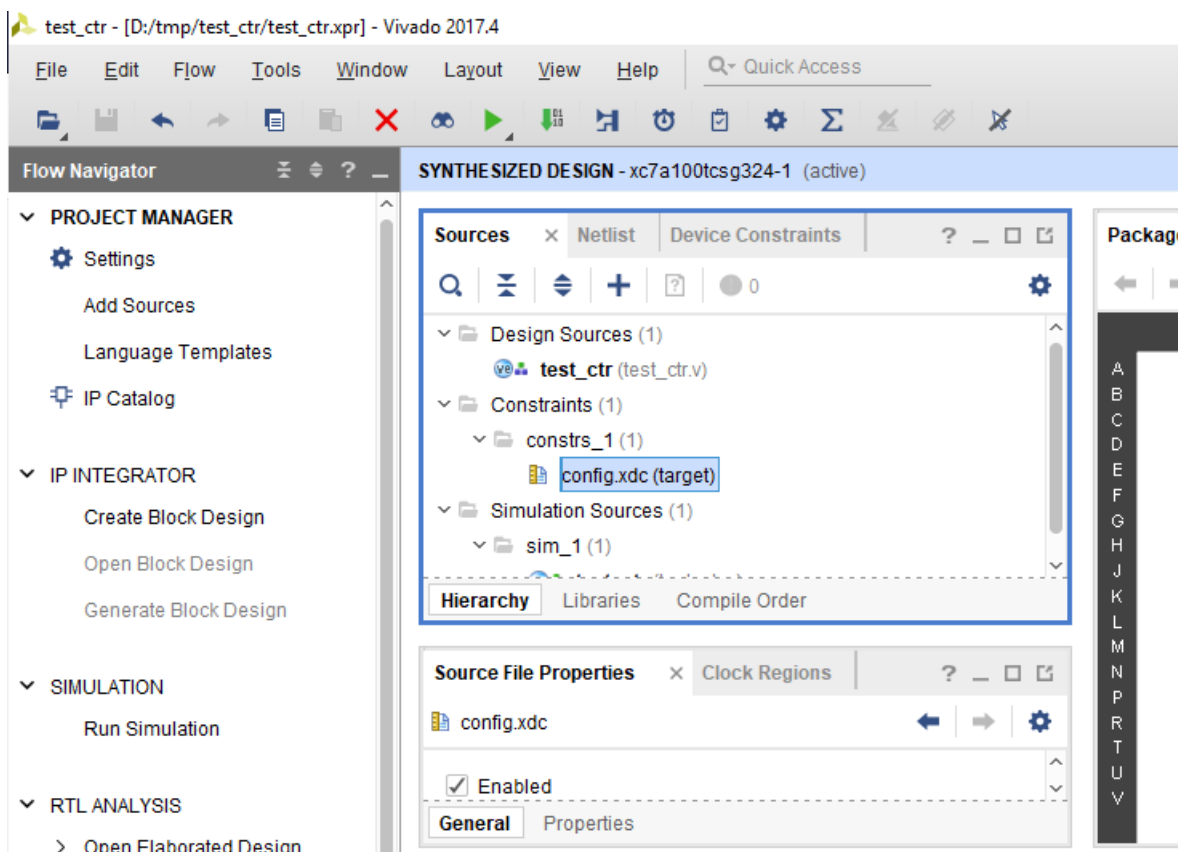


Рисунок 3.15 – Расположение файла ограничений проекта

После успешной генерации файла прошивки отобразится окно с предложением открыть проект после размещения (рисунок 3.19). Можно нажать *Cancel*. Пока нам не требуется вносить изменения в размещение на ПЛИС.

```

14 set_property PACKAGE_PIN V17 [get_ports {leds[5]]
15 set_property PACKAGE_PIN U17 [get_ports {leds[6]]
16 set_property PACKAGE_PIN U16 [get_ports {leds[7]]
17 set_property IOSTANDARD LVCMOS33 [get_ports clk]
18
19 set_property PACKAGE_PIN E3 [get_ports clk]
20 create_clock -add -name clk -period 10.00 -waveform {0 5} [get_ports { clk }];
21
22 set_property PACKAGE_PIN P18 [get_ports rst]
23 set_property PACKAGE_PIN J15 [get_ports sw]
24 set_property IOSTANDARD LVCMOS33 [get_ports rst]
25 set_property IOSTANDARD LVCMOS33 [get_ports sw]
26

```

Рисунок 3.16 – Содержимое файла ограничений проекта

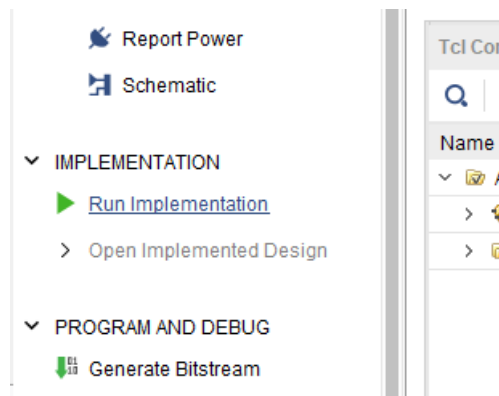


Рисунок 3.17 – Кнопка запуска процесса отображения и размещения схемы на ресурсах ПЛИС

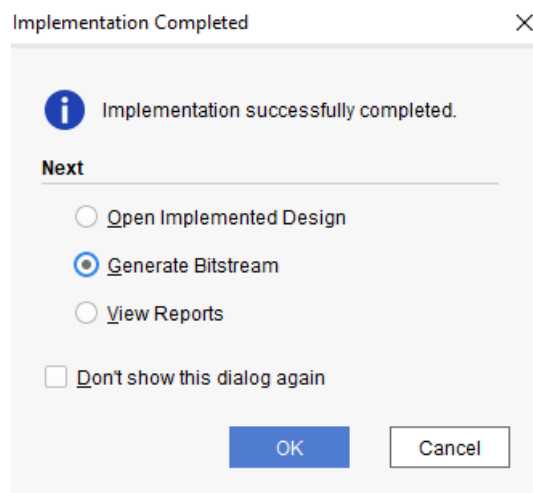


Рисунок 3.18 – Окно с предложением генерации прошивки ПЛИС

Файл прошивки имеет расширение \*.bit и располагается в каталоге с результатами размещения проекта. Мы можем посмотреть количество ресурсов, занимаемых проектом на ПЛИС в панели *Design Runs* (рисунок 3.20).

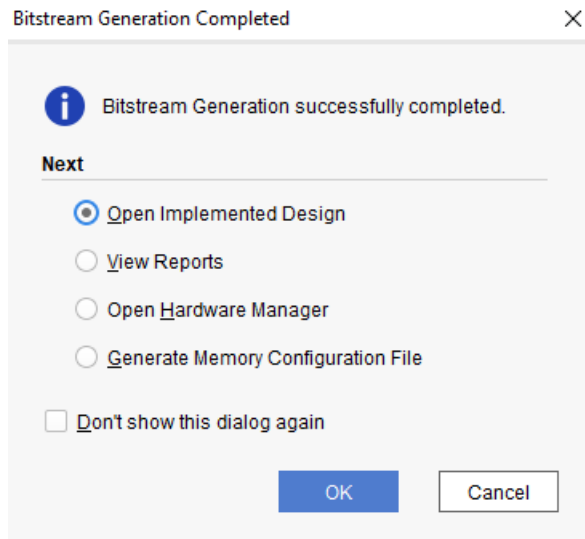


Рисунок 3.19 – Окно с предложением открытия проекта после размещения

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP
✓ synth_1	constrs_1	synth_design Complete!								46	40	0.00	0	0
✓ impl_1	constrs_1	write_bitstream Complete!	5.229	0.000	0.266	0.000	0.000	0.098	0	46	40	0.00	0	0

Рисунок 3.20 – Информация по занимаемым ресурсам ПЛИС

Также информацию о ресурсах можно просмотреть, нажав на символ суммы в верхней инструментальной панели Vivado (рисунок 3.21). Откроется отчет об использовании ресурсов (рисунок 3.22).

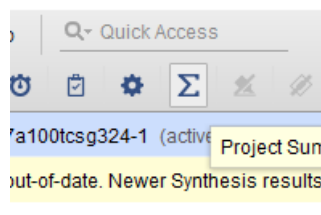


Рисунок 3.21 – Кнопка просмотра отчета по занимаемым ресурсам ПЛИС

Для проверки работоспособности схемы необходимо загрузить полученную прошивку на ПЛИС. Это можно сделать с помощью Hardware Manager, входящего в состав Vivado. Также можно загрузить прошивку с помощью приложения IMPACT, входящего в состав ISE Design Suite – предшественника Vivado. Далее будет описан процесс загрузки прошивки с помощью Hardware Manager.

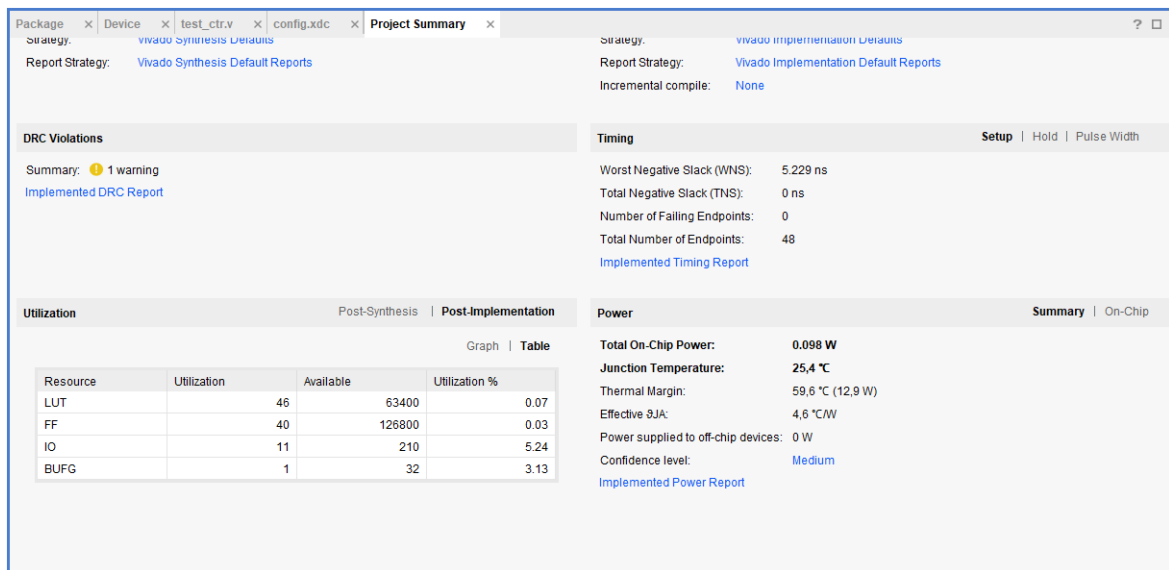


Рисунок 3.22 – Отчет по занимаемым ресурсам ПЛИС

Для открытия Hardware Manager необходимо нажать на *Open Hardware Manager* на левой панели Vivado (рисунок 3.23).

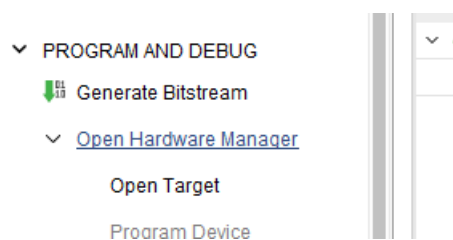


Рисунок 3.23 – Кнопка открытия режима прошивки ПЛИС

После открытия Hardware Manager необходимо подключить плату Nexys 4 DDR к компьютеру и включить питание платы. Затем нажать на кнопку автоопределения подключенного устройства – *Auto Connect* (рисунок 3.24).

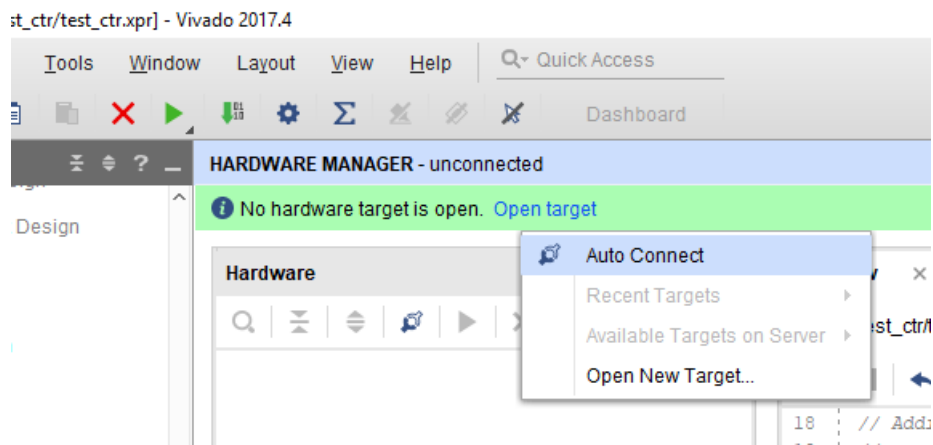


Рисунок 3.24 – Кнопка автоопределения платы с ПЛИС

После успешного определения ПЛИС необходимо щелкнуть правой кнопкой мышки на ней и выбрать *Program Device* (рисунок 3.25).

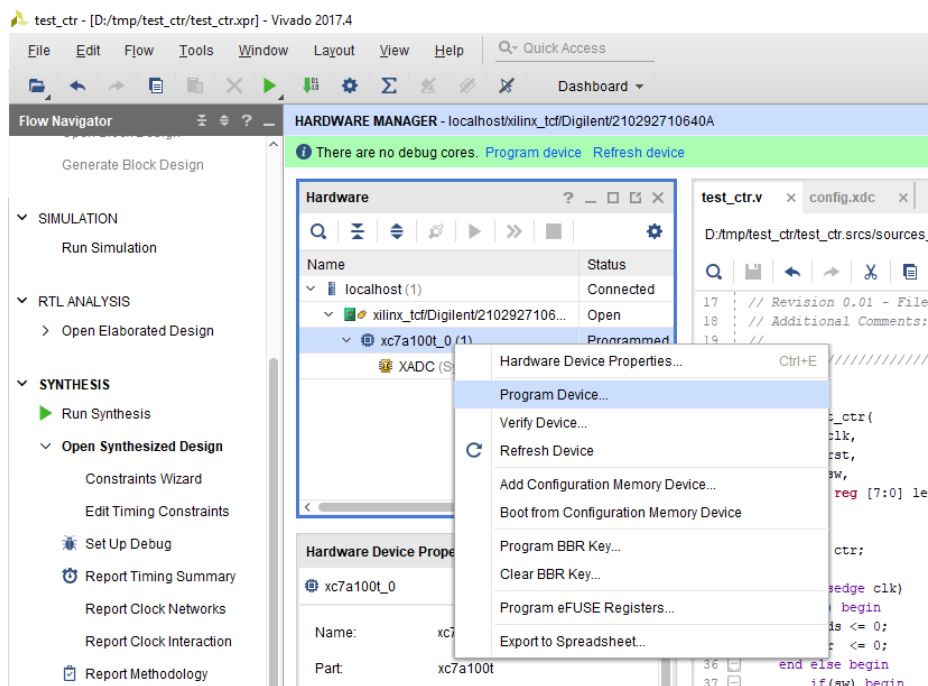


Рисунок 3.25 – Меню программирования ПЛИС

После нажатия на *Program Device* откроется диалоговое окно с выбором файла прошивки. Если Hardware Manager запускается из текущего проекта, как в данном примере, то путь к файлу прописывается автоматически. Для программирования нажимаем *Program* (рисунок 3.26).

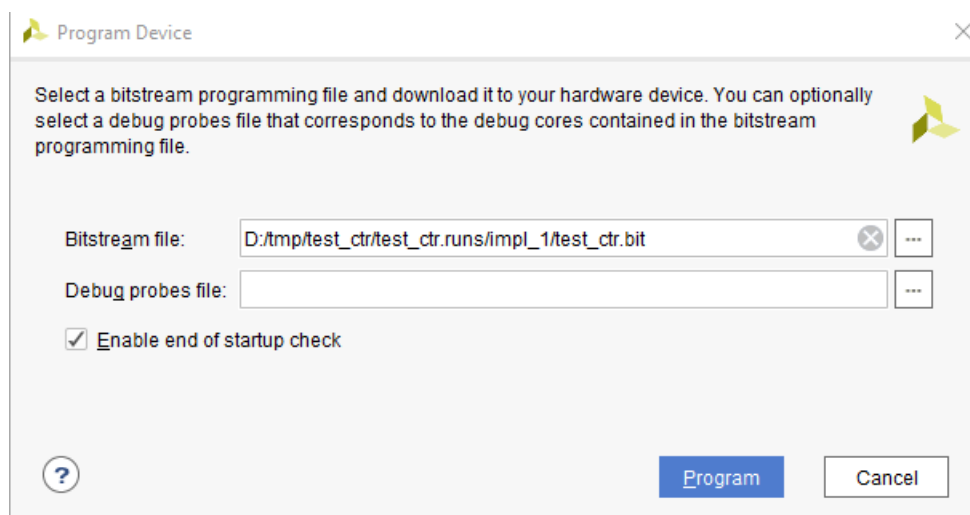


Рисунок 3.26 – Окно выбора файла прошивки

После успешного завершения процесса программирования можно увидеть на плате результат работы спроектированной схемы.

## **4 Лабораторная работа №4. Проектирование встроенных схем самотестирования**

### **4.1 Введение**

Лабораторная работа №4 посвящена вопросам разработки встроенных схем самотестирования для проектов на ПЛИС.

В процессе моделирования схемы в симуляторе невозможно точно учесть реальные временные задержки, а моделирование секунды жизни системы может занимать минуты и часы реального времени. В связи с этим возникает проблема тестирования системы с реальными задержками прохождения сигналов через элементы, а также перебором всех необходимых тестовых векторов за минимальное время. Это возможно осуществить с помощью встроенных блоков самотестирования, так как они работают в реальном окружении и в реальном времени.

Блоки самотестирования позволяют проверить систему без подключения к внешнему оборудованию, и их наличие актуально на всем протяжении эксплуатации системы. В процессе эксплуатации из-за воздействия различных факторов окружающей среды (например, перегрева, повышенной влажности, присутствия электромагнитного излучения и пр.) может происходить деградация элементной базы устройства, что приводит к сбоям в работе системы. Регулярный запуск процесса самотестирования может помочь вовремя выявить нарушения работоспособности системы и обезопасить окружение от последствий сбоев.

### **4.2 Описание лабораторной работы**

#### **Цель работы**

Получить навыки разработки встроенных схем самотестирования (BIST, built-in self-test) для цифровых устройств, реализованных на базе программируемых логических интегральных схем (ПЛИС).

#### **Указания к выполнению работы**

Лабораторная работа посвящена знакомству с особенностями разработки встроенных схем самотестирования для цифровых устройств, реализованных на базе микросхем ПЛИС. Работа выполняется в Vivado Design Suite и с использованием отладочной платы Nexys 4 DDR (новое название Nexys A7).

## Порядок выполнения работы

1. Доработайте схему функционального блока, разработанного в лабораторной работе №3, в соответствии с рисунком 4.1. На рисунке данный блок обозначен как DUT (Design under Test). Необходимо добавить в схему возможность выполнять самотестирование по внешней команде – по факту нажатия кнопки. Схема самотестирования должна удовлетворять следующим требованиям:
  - 1.1. Вход и выход из режима самотестирования должен выполняться по факту нажатия кнопки BTN.
  - 1.2. Модуль «BIST control logic» должен управлять коммутацией сигналов в режиме тестирования. Данный модуль также должен хранить количество переходов в режим самотестирования с момента подачи питания на схему.
  - 1.3. Должна быть поддержана возможность подавать вместо операндов функции значения с регистров сдвига с линейной обратной связью LFSR (Linear-Feedback Shift Register). Регистры LFSR будут выполнять функции генераторов псевдослучайных чисел. Полиномы, на базе которых работают регистры LFSR, определяются в варианте задания.
  - 1.4. Значение выхода функционального блока (DUT) в режиме тестирования должно проходить через модуль расчета циклического избыточного 8-битного кода CRC8 (Cyclic Redundancy Check). По значению CRC8 в конце тестирования определяется корректность работы схемы. Использование такого подхода позволяет сэкономить память для записи истории тестирования и для списка эталонных значений на каждой итерации тестирования. Полином для CRC8 определяется в варианте задания.
  - 1.5. Результат работы блока в режиме самотестирования должен представлять собой значение кода CRC8 после 256 итераций тестирования. Одной итерацией тестирования называется расчет результата на одной комбинации входных операндов.
  - 1.6. В режиме тестирования на светодиоды должно выводиться значение CRC8 и количество переходов схемы в режим самотестирования с момента подачи питания на схему. В режиме нормальной работы на светодиоды выводится результат расчета функционального блока (DUT).
2. Разработайте тестовое окружение и проведите моделирование схемы. Определите корректное значение CRC8 в конце процесса самотестирования, то есть после 256 итераций смены входных операндов.



3. Проведите синтез и размещение схемы для ПЛИС XC7A100T-1CSG324C, входящей в состав отладочной платы Nexys 4 DDR.
4. Определите количество и тип используемых ресурсов ПЛИС после размещения схемы.
5. Проверьте работоспособность схемы на отладочной плате Nexys 4 DDR в нормальном режиме и в режиме самотестирования.
6. Составьте отчет по результатам выполнения работы.

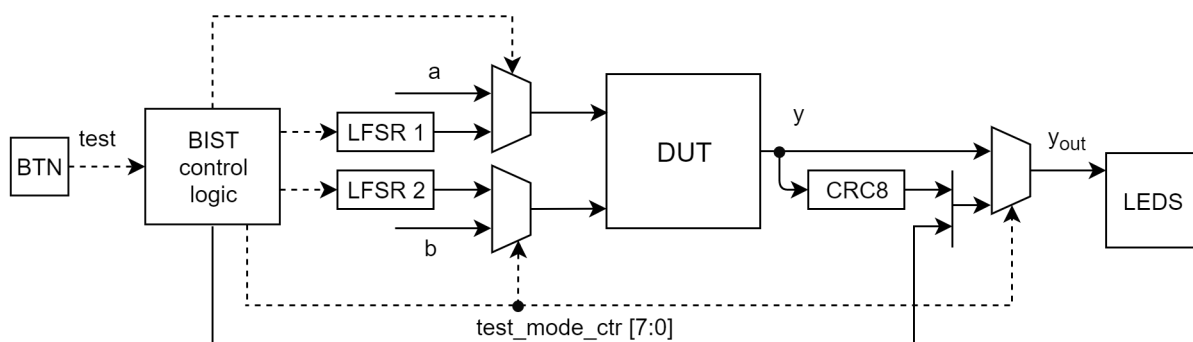


Рисунок 4.1 – Схема сопряжения функционального блока со схемой тестирования: BTN – тактовая кнопка, LEDES – светодиоды, DUT (Design under Test) – модуль из лабораторной работы №3, LFSR 1 и LFSR 2 (Linear-Feedback Shift Register) – регистры сдвига с линейной обратной связью, CRC8 (Cyclic Redundancy Check) – модуль расчета циклического избыточного 8-битного кода

На рисунке 4.1 интерфейсы ввода/вывода блока DUT показаны упрощенно. Блок DUT должен иметь интерфейсы, аналогичные тем, что были в лабораторной работе №3.

## Требования к отчету

Отчет должен быть оформлен в соответствии с требованиями, представленными в Приложении А.

### Отчет должен включать:

1. Титульный лист.
2. Цель работы.
3. Задание в соответствии с вариантом.
4. Схемы (рисунки) устройства блока «BIST control logic», регистров LFSR и модуля расчета CRC8.

5. Результат тестирования блока в симуляторе (временные диаграммы).
6. Время моделирования режима самотестирования в симуляторе.
7. График с плотностью распределения значений операндов, которые перебираются в режиме самотестирования, с указанием области допустимых значений.
8. Процент количества значений операндов, которые попали в область допустимых значений.
9. Таблицу с используемыми ресурсами ПЛИС.
10. Выводы по работе.

### Задания по вариантам

№ вар.	Полином LFSR 1	Полином LFSR 2	Полином CRC8
1	$y = 1 + x + x^2 + x^4 + x^8$	$y = 1 + x^2 + x^3 + x^5 + x^8$	$y = 1 + x^2 + x^3 + x^4 + x^8$
2	$y = 1 + x^2 + x^4 + x^5 + x^8$	$y = 1 + x^2 + x^3 + x^7 + x^8$	$y = 1 + x^4 + x^5 + x^7 + x^8$
3	$y = 1 + x^3 + x^4 + x^5 + x^8$	$y = 1 + x + x^3 + x^5 + x^8$	$y = 1 + x^4 + x^6 + x^7 + x^8$
4	$y = 1 + x + x^5 + x^6 + x^8$	$y = 1 + x^2 + x^6 + x^7 + x^8$	$y = 1 + x^3 + x^5 + x^7 + x^8$
5	$y = 1 + x^3 + x^5 + x^7 + x^8$	$y = 1 + x^4 + x^6 + x^7 + x^8$	$y = 1 + x^4 + x^5 + x^6 + x^8$
6	$y = 1 + x + x^6 + x^7 + x^8$	$y = 1 + x^2 + x^4 + x^5 + x^8$	$y = 1 + x^2 + x^4 + x^7 + x^8$
7	$y = 1 + x^4 + x^6 + x^7 + x^8$	$y = 1 + x + x^6 + x^7 + x^8$	$y = 1 + x^3 + x^5 + x^6 + x^8$
8	$y = 1 + x^3 + x^4 + x^6 + x^8$	$y = 1 + x^3 + x^6 + x^7 + x^8$	$y = 1 + x + x^3 + x^7 + x^8$
9	$y = 1 + x^5 + x^6 + x^7 + x^8$	$y = 1 + x + x^3 + x^5 + x^8$	$y = 1 + x + x^3 + x^4 + x^8$
10	$y = 1 + x^3 + x^6 + x^7 + x^8$	$y = 1 + x^2 + x^4 + x^5 + x^8$	$y = 1 + x^4 + x^5 + x^6 + x^8$

### 4.3 Устройство и принцип работы регистров LFSR

LFSR (Linear-Feedback Shift Register) – регистр сдвига с линейной обратной связью. Основное назначение данного регистра – генерирование псевдослучайных значений. Структурную схему LFSR можно увидеть на рисунке 4.2. Как показано на рисунке 4.2, для работы данного регистра необходимы сдвиговый регистр и функция обратной связи. Входными параметрами для алгоритма генерации псевдослучайных значений являются начальное состояние сдвигового регистра и значение порождающего полинома. Обобщенная форма полинома представлена в формуле (4.1).

$$P = c_L * x^L + \dots + c_2 * x^2 + c_1 * x^1 + 1 \quad (4.1)$$

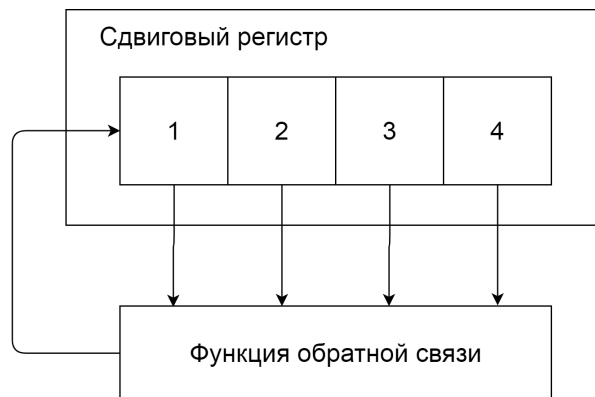


Рисунок 4.2 – Структурная схема LFSR-регистра

Как показано на рисунке 4.2, значение младшего бита сдвигового регистра определяется функцией обратной связи и зависит от значений всех битов регистра до их сдвига. Перед расчетом значения функции обратной связи значения битов сдвигового регистра умножаются на коэффициент  $c$ , который может принимать значения 0 или 1 (задается полиномом). При этом коэффициент для младшего разряда всегда равен 1. Далее получившиеся коэффициенты попадают на функцию обратной связи. В качестве функции обратной связи наиболее часто используется сложение по модулю 2 (**XOR**) или его логическая инверсия (**XNOR**). На рисунке 4.3 представлена структурная схема реализации блока обратной связи с использованием функции **XOR**.

Таким образом, новое значение младшего разряда сдвигового регистра можно рассчитать по формуле (4.2).

$$x_0 = (x_0 * c_0) \oplus (x_1 * c_1) \oplus (x_2 * c_2) \dots \oplus (x_{L-1} * c_{L-1}) \quad (4.2)$$

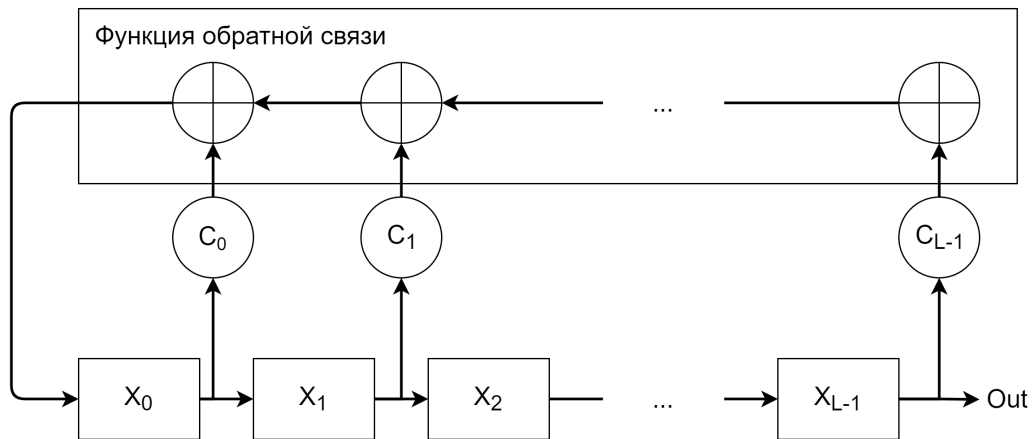


Рисунок 4.3 – Структурная схема LFSR-регистра и блока обратной связи на XOR

По формуле (4.2) можно увидеть, что количество аппаратных элементов **XOR** будет определяться количеством коэффициентов, равных 1. Коэффициенты, в свою очередь, задаются в бинарном полиноме.

В формуле (4.3) представлен пример 5-разрядного полинома.

$$P = 1 * x^5 + 0 * x^4 + 1 * x^3 + 0 * x^2 + 0 * x^1 + 1 = x^5 + x^3 + 1 \quad (4.3)$$

Таким образом для данного примера достаточно будет использовать всего 3 аппаратных **XOR**, поскольку коэффициент **c**, равный 1, установлен только для 5, 3 и 0 битов сдвигового регистра.

На сегодняшний день существуют два варианта реализации регистра LFSR: внешний (external, рисунок 4.4) и внутренний (internal, рисунок 4.5).

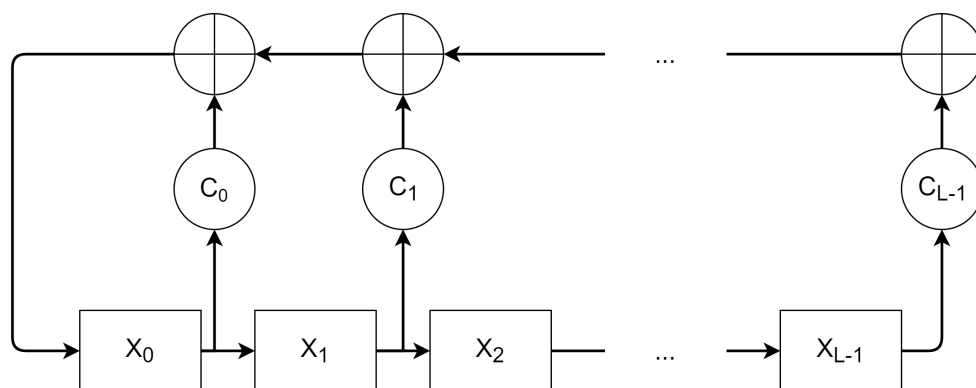


Рисунок 4.4 – Внешний (external) LFSR-регистр

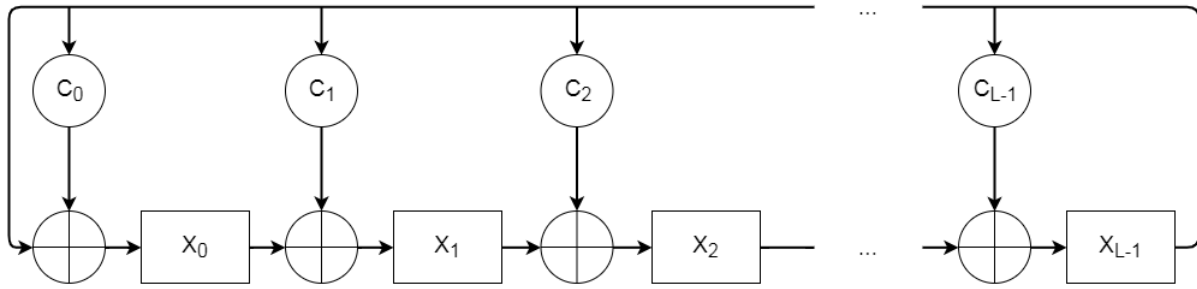


Рисунок 4.5 – Внутренний (internal) LFSR-регистр

Для внешнего LFSR-регистра результат функции обратной связи записывается в младший разряд сдвигового регистра, а в остальные разряды записывается значение предыдущего бита. Таким образом, внешний LFSR реализует следующую последовательность действий:

1. Установить начальное значение сдвигового регистра.
2. Произвести умножение каждого разряда сдвигового регистра на заданный коэффициент  $c$ .
3. Рассчитать значение нового бита на основе функции обратной связи.
4. Произвести сдвиг регистра.
5. В освободившийся разряд записать новое значение, полученное на шаге 3 (в результате будет получено новое псевдослучайное число).
6. Повторять шаги 2-5 для получения новых псевдослучайных чисел.

В случае внутреннего LFSR-регистра новое значение каждого разряда является результатом операции сложения по модулю 2 с результатом умножения старшего разряда на коэффициенты порождающего полинома.

## 4.4 Алгоритм расчета кода CRC8

CRC (Cyclic Redundancy Check) – циклический избыточный код, который предназначен для проверки целостности данных. CRC8 является одной из разновидностей кода CRC. Алгоритм расчета CRC8 подразумевает, что выходная контрольная сумма записывается в 8-разрядный регистр. Значение CRC может быть рассчитано по формуле (4.4).

$$R(x) = P(x) * x^N \text{ mod } G(x), \quad (4.4)$$

где  $\mathbf{R}(\mathbf{x})$  – многочлен, представляющий значение CRC;  $\mathbf{P}(\mathbf{x})$  – многочлен, коэффициенты которого представляют входные данные;  $\mathbf{G}(\mathbf{x})$  – порождающий полином;  $\mathbf{N}$  – степень многочлена.

Значение CRC является остатком от деления многочлена, соответствующего входным данным, на фиксированный порождающий полином. Порождающий полином представлен в формуле (4.5).

$$G(x) = c_L * x^L + \dots + c_2 * x^2 + c_1 * x^1 + c_1 * x^0, \quad (4.5)$$

где  $c$  является коэффициентом, который может принимать значения 0 или 1.

Для расчета CRC8 используют восьмиразрядный сдвиговый регистр. На рисунке 4.6 представлена схема вычисления CRC8 с полиномом  $x^8 + x^5 + x^4 + 1$ .

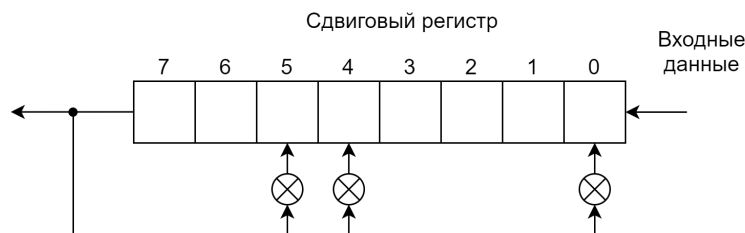


Рисунок 4.6 – Схема расчета CRC8 с порождающим полиномом  $x^8 + x^5 + x^4 + 1$

В процессе расчета CRC8 в восьмиразрядный регистр побитово поступают входные данные со стороны младшего разряда. Если выдвигаемый (старший) бит равен 1, то происходит сложение по модулю 2 коэффициентов порождающего полинома и содержимого регистра.

Для обработки данных по словам в 8 бит, то есть побайтово, можно воспользоваться следующим алгоритмом:

1. Установить начальное значение CRC.
2. Загрузить указанное слово в регистр данных.

3. Произвести операцию сложения по модулю 2 со значением регистра данных и текущим значением CRC.
4. Загрузить результат в сдвиговый регистр.
5. Произвести сдвиг данных в сторону старших разрядов.
6. Если выдвинутый бит равен 0, перейти к шагу 8.
7. Произвести операцию сложения по модулю 2 значения сдвигового регистра с коэффициентами полинома.
8. Загрузить результат в регистр CRC.
9. Повторить шаги с 3 по 8 (количество повторений равно длине слова, то есть 8 раз).
10. Повторить шаги с 2 по 9 (количество повторений равно количеству слов).

Поскольку значение полинома и начальное значение регистра CRC нам известны, и эти данные не меняются со временем, то можно заранее рассчитать значение CRC для входных данных и сохранить эти значения в память. Таким образом, алгоритм расчета нового значения CRC сводится к чтению данных из памяти, где входное значение будет выступать в роли адреса для доступа к памяти. При этом скорость работы данного алгоритма ограничена скоростью операции чтения данных из памяти. Табличный метод будет работать значительно быстрее, но при этом будет требовать больше ресурсов памяти на кристалле.

Для реализации табличного метода необходимо выполнить следующую последовательность действий:

1. Установить начальное значение CRC.
2. Загрузить слово данных в регистр данных.
3. Произвести операцию сложения по модулю 2 значения регистра данных и текущего значения CRC.
4. Полученный на шаге 3 результат подать в качестве адреса ячейки памяти.
5. Сохранить считанное из памяти значение в регистр CRC.
6. Повторить шаги с 2 по 5 (количество повторений равно количеству слов).

## 4.5 Дребезг контактов и вариант блока защиты для его устранения

Дребезг контактов – явление, возникающее в электрических и электронных переключателях, при котором они вместо некоторого стабильного сигнала выдают на выходе случайные высокочастотные колебания.

Причина возникновения дребезга заключается в механической конструкции кнопок и переключателей, которая не позволяет мгновенно зафиксировать контакт. На рисунке 4.7 показан пример временных диаграмм идеального и реального нажатий на кнопку. Как видно из диаграмм, на реальном объекте может возникать хаотичное изменение состояния входного сигнала. При неправильной обработке нажатия кнопки программная логика может воспринимать данное поведение как многократное нажатие кнопки. Таким образом, для нормальной работы схемы сигнал предварительно необходимо обработать. При проектировании схем обработки необходимо понимать, что дребезг может возникать как в моменты переключения кнопки или переключателя, так и во время удержания.

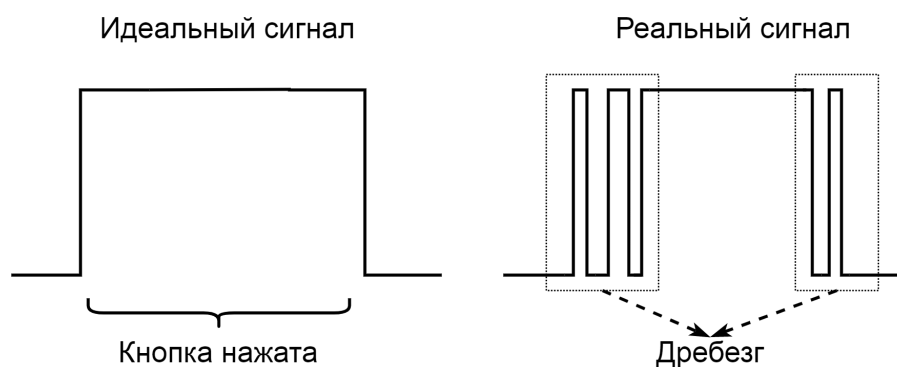


Рисунок 4.7 – Временные диаграммы нажатия на кнопку

В электронике одной из самых простых и распространенных схем для устранения подобного рода помех является RC-фильтр нижних частот (LPF, Low Pass Filter), который представлен на рисунке 4.8. Данная схема за счет значительного времени зарядки и разрядки конденсатора преобразует динамически изменяющийся высокочастотный сигнал в постоянное значение.

На верхней части рисунка 4.9 показан входной сигнал (LPF input), поступающий на RC-фильтр, на нижней части – выходной (LPF output). Как видно из рисунка 4.9, на входном сигнале имеются высокочастотные помехи, но их период значительно меньше постоянной времени RC-цепи. Поэтому на выходном сигнале данные помехи не наблюдаются. Таким образом, для корректной работы этой схемы необходимо правильно подобрать постоянную времени RC-цепи.



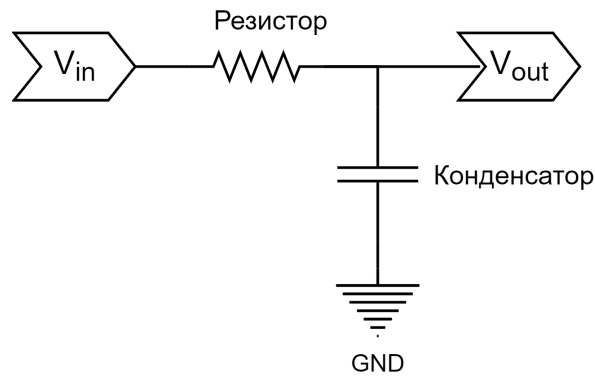


Рисунок 4.8 – RC-фильтр нижних частот

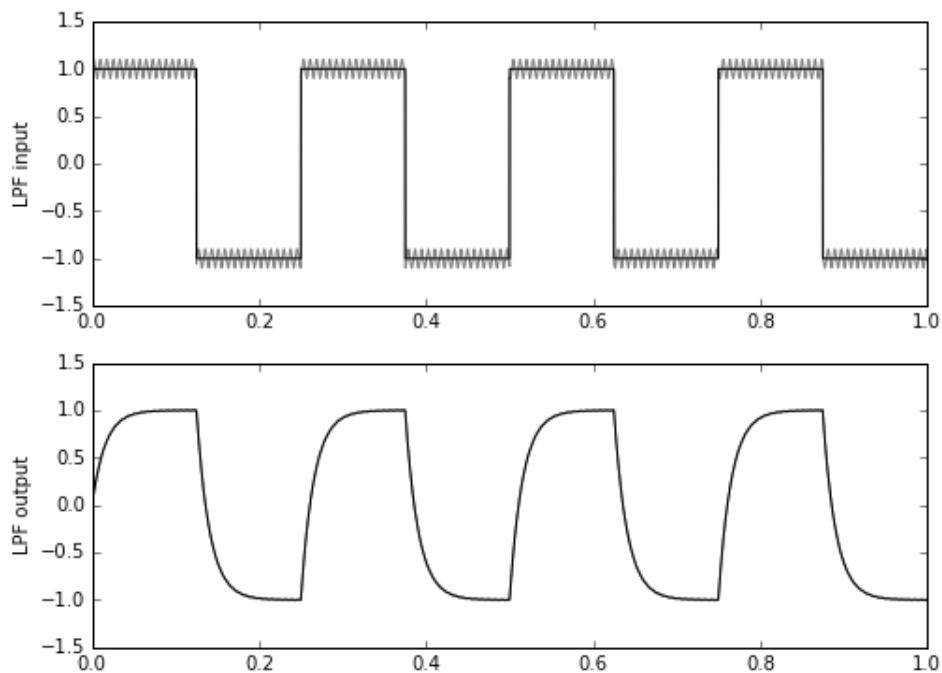


Рисунок 4.9 – Временные диаграммы RC-фильтра нижних частот

Аналогичный фильтр можно использовать и для обработки события нажатия кнопки в проектах на основе ПЛИС. Таким образом, задача обработки кнопки сводится к реализации фильтра нижних частот с использованием элементов цифровой схемотехники. Структурная схема данной реализации представлена на рисунке 4.10.

Как видно из рисунка, входной сигнал поступает на триггер, после чего переходит на компаратор. На компараторе происходит его сравнение с текущим выходным сигналом. Если входное значение отличается от выходного, то считается, что произошло событие (кнопку нажали или отпустили). С этого момента устанавливается сигнал разрешения счетчика, при этом счетчик будет изменять свое значение при каждом фронте тактового сигнала. В противном случае счетчик находится в сброшенном состоянии. Значение счетчика поступает на вход еще одного компаратора,

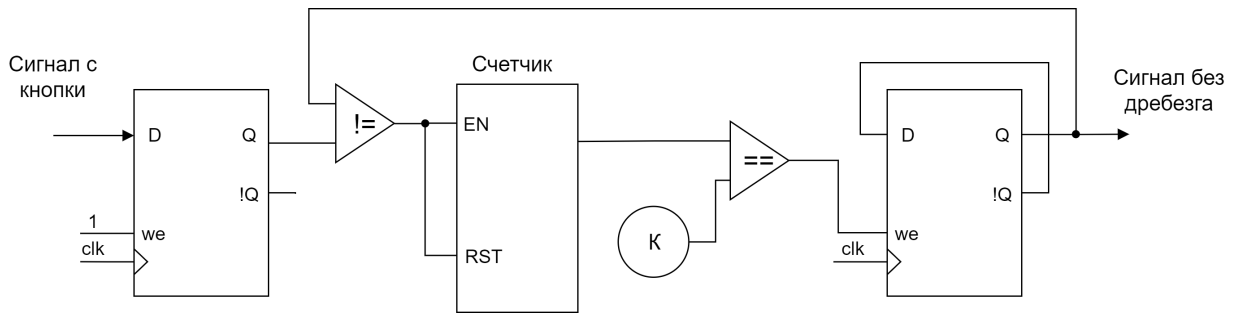


Рисунок 4.10 – Реализация фильтра нижних частот на цифровых элементах

который проверяет его значение на равенство константе **К**. Значение **К**, аналогично постоянной времени RC-цепи, подбирается для каждой задачи в соответствии с особенностями данной задачи. При равенстве счетчика значению **К** устанавливается сигнал разрешения обновления данных на выходном триггере. Новым значением выходного сигнала является его инверсное значение.

Разработанная схема имеет один большой недостаток – она не учитывает, что событие нажатия кнопки по определению является асинхронным по отношению к внутренней логике, реализованной на ПЛИС. Смена логического состояния на входной ножке ПЛИС может совпасть с моментом переключения принимающего триггера, и есть вероятность, что триггер окажется в метастабильном состоянии. Это может привести к непредсказуемым результатам. Следовательно, необходимо избавиться от потенциально возможного метастабильного состояния. Добиться этого можно за счет использования схем синхронизации. Простейшей схемой синхронизации могут служить два последовательно подключенных триггера, тактируемые от одного тактового сигнала. Доработанную схему фильтра можно увидеть на рисунке 4.11. Как видно на рисунке, входной сигнал сначала поступает на вход цепи синхронизации, состоящей из двух триггеров, и только после этого поступает на вход фильтра.

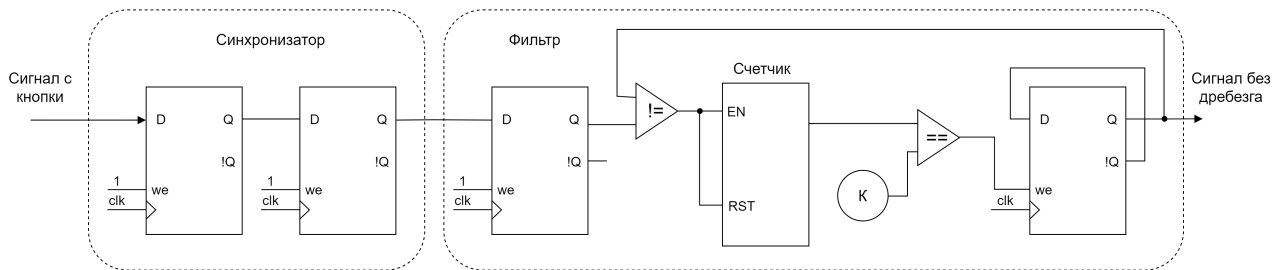


Рисунок 4.11 – Программная реализация фильтра нижних частот со схемой синхронизации

Также в ряде задач необходимо выделять момент нажатия или отпускания кнопки. Необходимого поведения можно добиться путем подачи сигнала out на схему выделения момента изменения сигнала. На рисунке 4.12 представлена схема выде-

ления переднего фронта сигнала, а на рисунке 4.13 представлена схема выделения заднего фронта (спада). На рисунке 4.14 представлена схема выделения момента любого изменения сигнала.

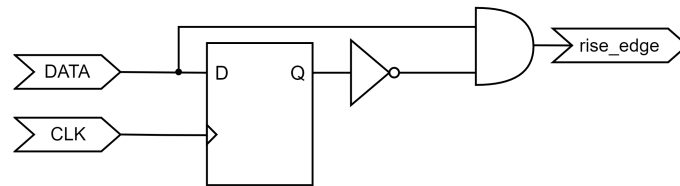


Рисунок 4.12 – Схема выделения переднего фронта сигнала

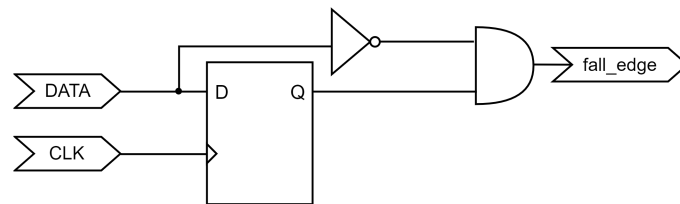


Рисунок 4.13 – Схема выделения заднего фронта (спада) сигнала

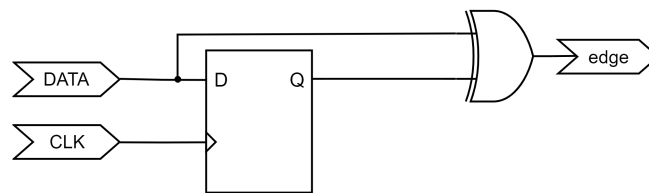


Рисунок 4.14 – Схема выделения момента изменения сигнала

## **5 Лабораторная работа №5. Проектирование контроллеров ввода и индикации**

### **5.1 Введение**

Лабораторная работа №5 посвящена развитию навыков работы с дискретными портами ввода, подключенными к переключателям и кнопкам, а также освоению принципов взаимодействия с простейшими устройствами индикации – светодиодами и семисегментными индикаторами.

В работе предлагается разработать устройство аппаратного контроллера ввода и индикации с использованием языка описания аппаратуры Verilog HDL. После этого необходимо выполнить прототипирование устройства на отладочной плате Nexys 4 DDR с ПЛИС XC7A100T-1CSG324C.

Для успешного запуска проекта на ПЛИС необходимо использовать синтезируемое подмножество языка Verilog HDL.

### **5.2 Описание лабораторной работы**

#### **Цель работы**

Освоение работы с базовыми дискретными элементами ввода/вывода: светодиоды, кнопки, позиционные переключатели, семисегментные индикаторы.

#### **Указания к выполнению работы**

Работа выполняется в Vivado Design Suite и с использованием отладочной платы Nexys 4 DDR (новое название Nexys A7).

#### **Порядок выполнения работы**

1. Разработайте архитектуру устройства, указанного в варианте задания.
2. Опишите RTL-модель устройства с использованием языка Verilog HDL.
3. Разработайте тестовое окружение и план тестирования разработанной RTL-модели.
4. Проведите отладку и тестирование RTL-модели в симуляторе.

5. Выполните синтез RTL-модели устройства на аппаратные ресурсы ПЛИС Artix-7 XC7A100T-1CSG324C.
6. Проведите прототипирование разработанного устройства на отладочной плате Nexys 4 DDR.
7. Составьте отчет по результатам выполнения работы.

## **Требования к отчету**

Отчет должен быть оформлен в соответствии с требованиями, представленными в Приложении А.

### **Отчет должен включать:**

1. Титульный лист.
2. Цель работы.
3. Задание в соответствии с вариантом.
4. Структурную схему RTL-модели разрабатываемого устройства. Структурную схему сопроводить комментариями, поясняющими алгоритм работы устройства.
5. Описание структуры тестового окружения и плана тестирования.
6. Временные диаграммы, доказывающие корректность работы разработанного устройства. Временные диаграммы сопроводить комментариями.
7. Выводы по работе.

## Задания по вариантам

### Вариант 1

Реализовать модуль управления трехцветным светодиодом LD17 отладочной платы Nexys 4 DDR.

Светодиод должен в непрерывном режиме отображать следующую цветовую анимацию:

1. В течение 1 с плавно загорается красный светодиод и остается в активном состоянии.
2. В течение 1 с плавно загорается синий светодиод и остается в активном состоянии.
3. В течение 1 с плавно загорается зеленый светодиод и остается в активном состоянии.
4. В течение 1 с плавно гаснет зеленый светодиод.
5. В течение 1 с плавно гаснет синий светодиод.
6. В течение 1 с плавно гаснет красный светодиод.

Шаги 1-6 непрерывно циклически повторяются.

Плавное включение-выключение необходимо реализовать с использованием широтно-импульсной модуляции (ШИМ) сигнала, подаваемого на светодиод определенного цвета. Яркость светодиода варьируется изменением скважности сигнала.

## Вариант 2

Реализовать модуль светодиодной индикации, выводящий анимацию по заданному алгоритму на линейку из 16 светодиодов отладочной платы Nexys 4 DDR. Кадры анимации представлены в таблице 5.1.

Таблица 5.1 – Описание кадров анимации варианта 2

Кадр	Состояние линейки светодиодов LED[15:0]														
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
3	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1
4	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1
5	1	1	1	1	0	0	0	0	0	0	0	1	1	1	1
6	1	1	1	1	1	0	0	0	0	0	1	1	1	1	1
7	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1
8	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1
9	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
10	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0
11	0	0	1	1	1	1	1	1	1	1	1	1	1	0	0
12	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0
13	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0
14	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0
15	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0
16	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0

Кадры 1-16 непрерывно циклически повторяются. Должно быть реализовано четыре режима скорости смены кадров:

1.  $SW[1:0] = 0$  – анимация остановлена, отображается последний кадр.
2.  $SW[1:0] = 1$  – смена каждые 1000 мс.
3.  $SW[1:0] = 2$  – смена каждые 500 мс.
4.  $SW[1:0] = 3$  – смена каждые 200 мс.

Режим скорости анимации задается с помощью двух переключателей  $SW[1:0]$ .

### Вариант 3

Реализовать модуль светодиодной индикации, выводящий анимацию по заданному алгоритму на линейку из 16 светодиодов отладочной платы Nexys-4 DDR. Кадры анимации представлены в таблице 5.2.

Таблица 5.2 – Описание кадров анимации варианта 3

Кадр	Состояние линейки светодиодов LED[15:0]														
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
3	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1
4	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1
5	1	1	1	1	0	0	0	0	0	0	0	1	1	1	1
6	1	1	1	1	1	0	0	0	0	0	1	1	1	1	1
7	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1
8	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1
9	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
10	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1
11	1	1	1	1	1	1	0	0	0	0	1	1	1	1	1
12	1	1	1	1	1	0	0	0	0	0	1	1	1	1	1
13	1	1	1	1	0	0	0	0	0	0	0	1	1	1	1
14	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1
15	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1
16	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Кадры 1-16 непрерывно циклически повторяются. Должно быть реализовано четыре режима скорости смены кадров:

1.  $SW[1:0] = 0$  – анимация остановлена, отображается последний кадр.
2.  $SW[1:0] = 1$  – смена каждые 1000 мс.
3.  $SW[1:0] = 2$  – смена каждые 500 мс.
4.  $SW[1:0] = 3$  – смена каждые 200 мс.

Режим скорости анимации задается с помощью двух переключателей  $SW[1:0]$ .



## Вариант 4

Разработать модуль таймера с отображением значений на линейке семисегментных индикаторов.

При реализации таймера используются четыре семисегментных индикатора для отображения минут, два индикатора для отображения секунд, два индикатора для отображения миллисекунд.

Предусмотреть возможность сброса значения таймера по факту нажатия на кнопку BTNC.

## Вариант 5

Разработать контроллер бегущей строки с отображением анимации на линейке семисегментных индикаторов.

На семисегментные индикаторы должна выводиться строка HELLO по алгоритму, представленному в таблице 5.3.

Таблица 5.3 – Описание кадров анимации варианта 5

Кадр	Состояния индикаторов							
1	-	-	-	-	-	-	-	-
2	О	-	-	-	-	-	-	-
3	L	О	-	-	-	-	-	-
4	L	L	О	-	-	-	-	-
5	E	L	L	О	-	-	-	-
6	H	E	L	L	О	-	-	-
7	-	H	E	L	L	О	-	-
8	-	-	H	E	L	L	О	-
9	-	-	-	H	E	L	L	О
10	-	-	-	-	H	E	L	L
11	-	-	-	-	-	H	E	L
12	-	-	-	-	-	-	H	E
13	-	-	-	-	-	-	-	H

Кадры 1-13 непрерывно циклически повторяются. Предусмотреть возможность сброса анимации в начало по факту нажатия на кнопку BTNC.

## **6 Лабораторная работа №6. Основы проектирования контроллеров периферийных устройств**

### **6.1 Введение**

Лабораторная работа №6 направлена на развитие навыков проектирования контроллеров последовательных интерфейсов ввода/вывода, а также аппаратных блоков обработки данных с внешних датчиков и устройств ввода/вывода. Для развития базовых навыков студентам предлагается разработать контроллеры для таких интерфейсов, как SPI, UART, I2C, PS/2, а также аппаратные блоки обработки данных с датчиков температуры, освещенности, прикосновений, джойстика, мыши и клавиатуры.

Разработанные проекты необходимо протестировать и отладить с использованием отладочной платы Nexys 4 DDR с ПЛИС XC7A100T-1CSG324C и необходимых устройств ввода/вывода.

### **6.2 Описание лабораторной работы**

#### **Цели работы**

1. Знакомство с принципами работы последовательных интерфейсов ввода/вывода SPI, UART, I2C, PS/2.
2. Изучение основ разработки аппаратных контроллеров периферийных устройств.
3. Изучение основ работы с цифровыми датчиками.

#### **Указания к выполнению работы**

Работа выполняется в Vivado Design Suite и с использованием отладочной платы Nexys 4 DDR (новое название Nexys A7) с необходимыми устройствами ввода/вывода.

#### **Порядок выполнения работы**

1. Разработайте архитектуру устройства, указанного в варианте задания. Необходимо разделить контроллер интерфейса и блок прикладной логики.

2. Опишите RTL-модель устройства с использованием языка Verilog HDL. Контроллер интерфейса должен находиться в отдельном модуле.
3. Разработайте тестовое окружение и план тестирования разработанной RTL-модели.
4. Проведите отладку и тестирование RTL-модели в симуляторе.
5. Выполните синтез RTL-модели устройства на аппаратные ресурсы ПЛИС Artix-7 XC7A100T-1CSG324C.
6. Проведите прототипирование разработанного устройства на отладочной плате Nexys 4 DDR.
7. Составьте отчет по результатам выполнения работы.

## **Требования к отчету**

Отчет должен быть оформлен в соответствии с требованиями, представленными в Приложении А.

### **Отчет должен включать:**

1. Титульный лист.
2. Цели работы.
3. Задание в соответствии с вариантом.
4. Структурную схему RTL-модели разрабатываемого устройства. Структурную схему сопроводить комментариями, поясняющими алгоритм работы устройства.
5. Описание структуры тестового окружения и плана тестирования.
6. Временные диаграммы, доказывающие корректность работы разработанного устройства. Временные диаграммы сопроводить комментариями.
7. Выводы по работе.

## Задания по вариантам

### Вариант 1

Разработать контроллер, взаимодействующий с датчиком температуры ADT7420. Контроллер должен реализовывать следующие функции:

1. Поддерживать обмен данными с датчиком температуры посредством интерфейса I2C.
2. Выводить на линейку светодиодов одно из двух значений в зависимости от текущего состояния переключателя SW[0]:
  - 0 – ID устройства датчика;
  - 1 – текущая температура в градусах Цельсия.

### Вариант 2

Разработать контроллер, взаимодействующий с датчиком освещенности PmodALS. Контроллер должен реализовывать следующие функции:

1. Поддерживать обмен данными с датчиком освещенности посредством интерфейса SPI.
2. При значении переключателя SW[0] = 0 показывать факт наличия света в аудитории с помощью светодиодов: LED[15:0] = 0xFFFF – свет выключен, LED[15:0] = 0x0000 – свет включен. Таким образом, все светодиоды должны загораться, когда свет в аудитории выключается, и гаснуть – когда свет в аудитории включен.
3. При значении переключателя SW[0] = 1 показывать на светодиодах текущее значение освещенности, считанное с датчика.

### Вариант 3

Разработать контроллер, взаимодействующий с датчиком прикосновения PmodCDC1. Контроллер должен реализовывать следующие функции:

1. Поддерживать обмен данными с датчиком прикосновения посредством интерфейса I2C.
2. Выводить на линейку светодиодов одно из четырех значений в зависимости от текущего состояния переключателей SW[1:0]:
  - 0x0 – константа 0x00;

- 0x1 – текущее значение интенсивности прикосновения к кнопке BTN1;
- 0x2 – текущее значение интенсивности прикосновения к кнопке BTN2;
- 0x3 – выводить на светодиод LED[0] факт прикосновения к кнопке BTN1, а на светодиод LED[1] – факт прикосновения к кнопке BTN2 (прикосновение есть – соответствующий светодиод горит, прикосновения нет – светодиод не горит).

#### Вариант 4

Разработать контроллер, взаимодействующий с джойстиком PmodJSTK. Контроллер должен реализовывать следующие функции:

1. Поддерживать обмен данными с платой джойстика посредством интерфейса SPI.
2. Выводить на линейку светодиодов одно из четырех значений в зависимости от текущего состояния переключателей SW[1:0]:
  - 0x0 – константа 0x00;
  - 0x1 – текущее смещение джойстика по координате X;
  - 0x2 – текущее смещение джойстика по координате Y;
  - 0x3 – выводить на светодиоды LED[2:0] значения кнопок платы джойстика, на LED[3] – факт перемещения джойстика вверх, на LED[4] – вниз, на LED[5] – влево, на LED[6] – вправо (перемещение есть – соответствующий светодиод горит, перемещения нет – светодиод не горит).

#### Вариант 5

Разработать контроллер, взаимодействующий с компьютером по последовательному интерфейсу UART. Контроллер должен реализовывать следующие функции:

1. Поддерживать обмен данными с компьютером по интерфейсу UART (скорость 9600 бит/с, без контроля четности, один стоп-бит, длина слова восемь бит).
2. Постоянно выводить последнее принятое слово на светодиоды LED[7:0].
3. Работать в двух режимах в зависимости от состояния переключателя SW[0]:
  - 0 – режим «эхо» (принятое по UART слово отправляется обратно);
  - 1 – с периодичностью в 1000 мс отправляется строка «Hello world».

## Вариант 6

Разработать контроллер, взаимодействующий с клавиатурой по интерфейсу PS/2. Контроллер должен реализовывать следующие функции:

1. Поддерживать обмен данными с клавиатурой по интерфейсу PS/2.
2. Выводить на светодиоды скан-код последней нажатой клавиши.
3. Изменять состояние светодиодов клавиатуры при нажатии кнопок на отладочной плате (BTNL – Num Lock, BTNC – Caps Lock, BTNR – Scroll Lock).
4. Изменять состояние светодиодов клавиатуры при нажатии соответствующих кнопок на клавиатуре.

## Вариант 7

Разработать контроллер, взаимодействующий с мышью по интерфейсу PS/2. Контроллер должен реализовывать следующие функции:

1. Поддерживать обмен данными с мышью по интерфейсу PS/2.
2. Конфигурировать мышь в потоковый режим передачи данных (stream).
3. При значении переключателя  $SW[0] = 0$  выводить на светодиоды LED[7:0] слово статуса мыши (1 байт информационной посылки от мыши).
4. При значении переключателя  $SW[0] = 1$  выводить на светодиоды последнее смещение мыши по осям X и Y. Смещение по оси X отображается на светодиодах LED[7:0], смещение по оси Y – на LED[15:8].

## **А Требования к оформлению отчетов по лабораторным работам**

1. Отчет выполняется в виде самостоятельного документа. Материал, изложенный в отчете, должен быть понятен без дополнительных комментариев со стороны исполнителей.
2. Отчет выполняется как текстовый документ в соответствии с ГОСТ 2.105-95.
3. Размер шрифта основного текста 12-14 pt (Times, Calibri или аналогичный), межстрочный интервал 1-1,5, поля с краев листа не менее 2 см.
4. Листы отчета должны быть пронумерованы, кроме титульного листа, который считается первым.
5. Если отчет содержит большое количество листов, рекомендуется добавлять лист с содержанием отчета (разделы и номера листов).
6. На титульном листе указывается следующая информация:
  - название университета;
  - название факультета;
  - название дисциплины;
  - номер и тема лабораторной работы;
  - вариант лабораторной работы;
  - фамилия, инициалы и номер группы каждого исполнителя;
  - фамилия и инициалы преподавателя;
  - текущий год.
7. Обязательны нумерация и подписи к рисункам и таблицам, а также ссылки на них в тексте отчета.
8. Схемы и временные диаграммы должны быть темными на светлом фоне.
9. В распечатанном отчете линии на схемах и временных диаграммах должны быть четко видны.
10. На временных диаграммах не допускается наложение графиков сигналов друг на друга, если это мешает их однозначному восприятию. В таких случаях сигналы должны быть разнесены на отдельные координатные сетки.

## Список рекомендуемой литературы

1. Дэвид М. Харрис, Сара Л. Харрис. Цифровая схемотехника и архитектура компьютера. – 2-е изд. – 2013. – 1662 с.
2. Жан М. Рабаи, Ананта Чандракасан, Боривож Николич. Цифровые интегральные схемы. Методология проектирования. – 2-е изд. – М.: «Вильямс», 2016. – 912 с.
3. Угрюмов Е.П. Цифровая схемотехника: учеб. пособие для вузов. – 3-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2010. – 816 с.: ил.
4. Harris D., Harris S. Digital Design and Computer Architecture. – 2nd edition. – Morgan Kaufmann, 2012. – P. 712.
5. Weste N.H.E., Harris D.M. CMOS VLSI design: A circuits and systems perspective. – 4th edition. – Pearson, 2010. – P. 880.
6. Хоровиц П., Хилл У., Искусство схемотехники./Пер. с англ. – 6-е изд. – М.: Мир, 2003. – 704 с.
7. Rabaey J., Chandrakasan A., Nikolic B. Digital Integrated Circuits. – Pearson, 2003. – P. 800.
8. Baker R. J. CMOS: Circuit Design, Layout, and Simulation. – 3rd edition. – Wiley-IEEE Press, 2019. – P. 1280.
9. Точчи Р., Уидмер Дж., Нил С. Цифровые системы. Теория и практика. – 8-е изд. – М.: «Вильямс», 2004. – 1024 с.
10. Сохор Ю.Н. Моделирование устройств в пакете LTspice/SwCAD. Учебно-методическое пособие. Псковск. гос. политехн. ин-т. – Псков: Издательство ППИ, 2008. – 165 с.



Антонов Александр Александрович  
Быковский Сергей Вячеславович  
Кустарев Павел Валерьевич  
Кормилицын Константин Александрович  
Пинкевич Василий Юрьевич

## **ФУНКЦИОНАЛЬНАЯ СХЕМОТЕХНИКА**

### **Практикум**

В авторской редакции

Редакционно-издательский отдел Университета ИТМО

Зав. РИО

Н. Ф. Гусарова

Подписано к печати

Заказ №

Тираж

Отпечатано на ризографе

**Редакционно-издательский отдел**  
**Университета ИТМО**  
197101, Санкт-Петербург, Кронверкский пр., 49