МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Магнитогорский государственный технический университет им. Г.И.
Носова»

УТВЕРЖДАЮ
Директор ИЭиАС
_____ В.Р. Храмшин

04.02.2025 г.

РАБОЧАЯ ПРОГРАММА ДИСЦИПЛИНЫ (МОДУЛЯ)

ЯЗЫКИ ОПИСАНИЯ ЦИФРОВОЙ АППАРАТУРЫ (VHDL)

Направление подготовки (специальность)
11.03.04 Электроника и наноэлектроника

Направленность (профиль/специализация) программы
Интернет вещей в промышленной электронике

Уровень высшего образования - бакалавриат

Форма обучения
очная

Институт/ факультет	Институт энергетики и автоматизированных систем
Кафедра	Электроники и микроэлектроники
Курс	3
Семестр	5

Магнитогорск
2025 год

Рабочая программа составлена на основе ФГОС ВО - бакалавриат по направлению подготовки 11.03.04 Электроника и наноэлектроника (приказ Минобрнауки России от 19.09.2017 г. № 927)

Рабочая программа рассмотрена и одобрена на заседании кафедры Электроники и микроэлектроники
15.01.2025, протокол № 5

Зав. кафедрой _____ Д.Ю. Усатый

Рабочая программа одобрена методической комиссией ИЭиАС
04.02.2025 г. протокол № 3

Председатель _____ В.Р. Храмшин

Рабочая программа составлена:
доцент кафедры ЭиМЭ, к.т.н. _____ Швидченко Н.В.

Рецензент:
директор сервисного центра ООО «Техноап-Инжиниринг» , к.т.н.
_____ Суспицын Е.С.

Лист актуализации рабочей программы

Рабочая программа пересмотрена, обсуждена и одобрена для реализации в 2027 - 2028 учебном году на заседании кафедры Электроники и микроэлектроники

Протокол от __ _____ 20__ г. № __
Зав. кафедрой _____ Д.Ю. Усатый

Рабочая программа пересмотрена, обсуждена и одобрена для реализации в 2028 - 2029 учебном году на заседании кафедры Электроники и микроэлектроники

Протокол от __ _____ 20__ г. № __
Зав. кафедрой _____ Д.Ю. Усатый

Рабочая программа пересмотрена, обсуждена и одобрена для реализации в 2029 - 2030 учебном году на заседании кафедры Электроники и микроэлектроники

Протокол от __ _____ 20__ г. № __
Зав. кафедрой _____ Д.Ю. Усатый

Рабочая программа пересмотрена, обсуждена и одобрена для реализации в 2030 - 2031 учебном году на заседании кафедры Электроники и микроэлектроники

Протокол от __ _____ 20__ г. № __
Зав. кафедрой _____ Д.Ю. Усатый

1 Цели освоения дисциплины (модуля)

Целями освоения дисциплины «Языки описания цифровой аппаратуры (VHDL)» являются изучение основных концепций языков описания аппаратуры HDL, базовых понятий, получение навыков написания синтезируемого кода и кода для проведения функциональной верификации на языке VHDL.

2 Место дисциплины (модуля) в структуре образовательной программы

Дисциплина Языки описания цифровой аппаратуры (VHDL) входит в часть учебного плана формируемую участниками образовательных отношений образовательной программы.

Для изучения дисциплины необходимы знания (умения, владения), сформированные в результате изучения дисциплин/ практик:

Дискретная математика

Информатика и информационные технологии

Физика

Математика

Знания (умения, владения), полученные при изучении данной дисциплины будут необходимы для изучения дисциплин/практик:

САПР устройств промышленной электроники

Проектирование цифровой аппаратуры на ПЛИС

Основы проектирования электронной компонентной базы

3 Компетенции обучающегося, формируемые в результате освоения дисциплины (модуля) и планируемые результаты обучения

В результате освоения дисциплины (модуля) «Языки описания цифровой аппаратуры (VHDL)» обучающийся должен обладать следующими компетенциями:

Код индикатора	Индикатор достижения компетенции
ПК-1 Способен разрабатывать структурные и функциональные схемы электронных систем и комплексов, принципиальных схем устройств с использованием средств компьютерного проектирования, проведением проектных расчетов и технико-экономическим обоснованием принимаемых решений	
ПК-1.1	Разрабатывает эскизный проект, включающий: выбор структурной схемы электронного устройства или системы путем сопоставления различных вариантов и их оценки с точки зрения технических и экономических требований; рассчитывает все необходимые показатели структурной схемы электронного устройства или системы, в том числе показатели качества; выбирает и обосновывает схемы вспомогательных устройств
ПК-1.2	Производит технико-экономическое обоснование принятого решения с расчетами себестоимости устройства и стоимости его эксплуатации; сравнивает с аналогами по технико-экономическим характеристикам
ПК-3 Способен разрабатывать поведенческие описания моделей стандартных ячеек	
ПК-3.1	Проводит описание моделей стандартных элементов на поведенческом языке
ПК-3.2	Использует целевые системы автоматизированного проектирования

4. Структура, объём и содержание дисциплины (модуля)

Общая трудоемкость дисциплины составляет 3 зачетных единиц 108 акад. часов, в том числе:

– контактная работа – 73,9 акад. часов:

– аудиторная – 72 акад. часов;

– внеаудиторная – 1,9 акад. часов;

– самостоятельная работа – 34,1 акад. часов;

– в форме практической подготовки – 0 акад. час;

Форма аттестации - зачет с оценкой

Раздел/ тема дисциплины	Семестр	Аудиторная контактная работа (в акад. часах)			Самостоятельная работа студента	Вид самостоятельной работы	Форма текущего контроля успеваемости и промежуточной аттестации	Код компетенции
		Лек.	лаб. зан.	практ. зан.				
1. Языки описания аппаратуры. Основные понятия. Сфера применения. САПР								
1.1 Управление сложность в описании цифровых схем. Уровни представления и формы абстракции.	5	2						ПК-1.1, ПК-1.2
1.2 Основные понятия в HDL. Использование языков описания аппаратуры при проектировании СБИС. Использование языков описания аппаратуры для программирования ПЛИС.		2				Самостоятельное изучение учебной литературы		ПК-1.1, ПК-1.2
1.3 Системы автоматизированного проектирования (САПР). Обзор САПР для различных уровней проектирования. Языки описания аппаратуры HDL. Сквозное проектирование цифровых устройств на основе ПЛИС в САПР ISE WebPACK Xilinx		4	4			Самостоятельное изучение учебной литературы. Подготовка к лабораторным работам	Лабораторная работа	ПК-1.1, ПК-1.2
Итого по разделу		8	4					
2. Описание устройства с помощью VHDL. Базовая структура								
2.1 Базовая структура VHDL-файла. Описание	5	4	4		8	Самостоятельное изучение	Лабораторная работа	

интерфейса и архитектуры устройства. Стандартные библиотеки VHDL.						учебной литературы, подготовка к лабораторным работам		
2.2 Поведенческое и структурное описание архитектуры устройства с помощью VHDL	5	6	6		8	Самостоятельное изучение литературы, подготовка к лабораторным работам	Лабораторная работа	
Итого по разделу		10	10		16			
3. Основные конструкции VHDL								
3.1 Лексические и программные элементы VHDL. Типы данных. Операции с данными.	5	4	4			Самостоятельное изучение учебной литературы, подготовка к лабораторным работам	Лабораторная работа	
3.2 Поведенческое описание устройства. Параллельные и последовательные операторы. Оператор PROCESS.		4	6			Самостоятельное изучение учебной литературы, подготовка к лабораторным работам	Лабораторная работа	
3.3 Структурное описание устройства. Оператор PORT MAP.		4	4			Самостоятельное изучение учебной литературы, подготовка к лабораторным работам	Лабораторная работа	
Итого по разделу		12	14					
4. Функциональная верификация устройства с помощью VHDL								
4.1 Функциональная верификация. Метрики окончания верификации. Простая среда тестирования. Среда тестирования с самопроверкой.	5	4	4		10	Самостоятельное изучение учебной литературы. Подготовка к лабораторным работам	Лабораторная работа	ПК-1.1, ПК-1.2
4.2 Рандомизация тестовых воздействий. Ограниченная рандомизация тестовых воздействий.		2	4		8,1	Самостоятельное изучение учебной литературы. Подготовка к лабораторным занятиям	Лабораторная работа	ПК-1.1, ПК-1.2
Итого по разделу		6	8		18,1			
Итого за семестр		36	36		34,1		зао	
Итого по дисциплине		36	36		34,1		зачет с оценкой	

5 Образовательные технологии

Для реализации предусмотренных видов учебной работы в качестве образовательных технологий в преподавании дисциплины «Языки описания цифровой аппаратуры (VHDL)» используются традиционная и модульно-компетентностная технологии.

Предусмотрены обзорные лекции – для систематизации и закрепления знаний по дисциплине, информационные лекции – для ознакомления со стандартами и справочной информацией, лекции визуализации – для наглядного представления способов решения задач, проблемная лекция – для развития исследовательских навыков и изучения способов решения задач.

Лабораторные занятия по дисциплине проводятся как в традиционной, так и в интерактивной форме. В рамках интерактивного обучения применяются IT-методы (использование сетевых мультимедийных учебников, электронных образовательных ресурсов по данной дисциплине; совместная работа в малых группах (2-3 студента) – прохождение всех этапов и методов выполнения лабораторных работ; индивидуальное обучение.

6 Учебно-методическое обеспечение самостоятельной работы обучающихся

Представлено в приложении 1.

7 Оценочные средства для проведения промежуточной аттестации

Представлены в приложении 2.

8 Учебно-методическое и информационное обеспечение дисциплины

а) Основная литература:

1. Ушенина, И. В. Проектирование цифровых устройств на ПЛИС : учебное пособие для вузов / И. В. Ушенина. — 3-е изд., стер. — Санкт-Петербург : Лань, 2025. — 408 с. — ISBN 978-5-507-52271-2. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/445256 (дата обращения: 03.04.2026). — Режим доступа: для авториз. пользователей.

2. Язык verilog и проектирование цифровых устройств на плис : учебно-методическое пособие / составители Е. В. Богатиков, А. Н. Шебанов. — Воронеж : ВГУ, 2018. — 61 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/171183 (дата обращения: 03.04.2026). — Режим доступа: для авториз. пользователей.

б) Дополнительная литература:

1. Скитев, А. А. Верификация цифровых устройств: курс лекций : учебное пособие / А. А. Скитев. — Москва : НИЯУ МИФИ, 2020. — 92 с. — ISBN 978-5-7262-2696-5. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/175430 (дата обращения: 03.04.2026). — Режим доступа: для авториз. пользователей.

2. Математические модели и методы синтеза в сверхбольших интегральных схемах : учебное пособие / составители Н. И. Червяков Н. И. [и др.]. — Ставрополь : СКФУ, 2016. — 187 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/155293 (дата обращения: 03.04.2026). — Режим доступа: для авториз. пользователей.

в) Методические указания:

Методические рекомендации по выполнению практических заданий представлены в приложении 3.

г) Программное обеспечение и Интернет-ресурсы:

Программное обеспечение

Наименование ПО	№ договора	Срок действия лицензии
MS Office 2007 Professional	№ 135 от 17.09.2007	бессрочно
7Zip	свободно распространяемое ПО	бессрочно

Профессиональные базы данных и информационные справочные системы

Название курса	Ссылка

9 Материально-техническое обеспечение дисциплины (модуля)

Материально-техническое обеспечение дисциплины включает:

6 Учебно-методическое обеспечение самостоятельной работы студентов

По дисциплине «Языки описания цифровой аппаратуры (VHDL)» предусмотрена аудиторная и внеаудиторная самостоятельная работа обучающихся.

Аудиторная самостоятельная работа студентов предполагает выполнение и защиту лабораторных работ.

Лабораторные работы:

Лабораторная работа № 1. Изучение интегрированной среды проектирования для ПЛИС ISE WebPACK. Разработка тестов с помощью временных диаграмм во встроенном симуляторе ISim.

Лабораторная работа № 2. Основные этапы автоматизированной разработки проекта на языке VHDL. Рассмотрение примеров проектов на языке VHDL.

Лабораторная работа № 3. Разработка иерархической схемы проекта на языке VHDL. Программирование, компиляция проекта на языке VHDL.

Лабораторная работа № 4. Моделирование, синтез на языке VHDL.

Лабораторная работа № 5. Разработка проекта в поведенческой форме на языке VHDL.

Лабораторная работа № 6. Разработка проекта в структурной форме на языке VHDL.

Лабораторная работа № 7. Отладка устройства с использованием ПЛИС (отладочная плата NI Digital Electronics FPGA Board на основе ПЛИС Xilinx Spartan-3E).

Перечень вопросов для подготовки к защитам лабораторных работ:

1. Что такое язык описания аппаратуры HDL.
2. Каковы преимущества разработки схемы на базе HDL по сравнению со схемотехническим способом.
3. Что такое логический синтез схемы.
4. Какие САПР разработки ИС вы знаете?
5. Какие САПР для разработки схем на базе ПЛИС вы знаете?
6. Логический синтез ИС на стандартных ячейках.
7. Логический синтез схем на ПЛИС.
8. Что такое критический путь цифровой схемы?
9. Какие языки описания аппаратуры вы знаете?
10. Чем отличаются синтезируемые структуры языка HDL от несинтезируемых?
11. Какими способами можно повысить быстродействие цифровой схемы?
12. В чём заключается компромисс площадь кристалла/быстродействие?
13. Что такое синхронная цифровая схема?
14. Что включает в себя описание интерфейса (entity declaration) в языке VHDL.
15. Какие значения поддерживает тип данных **STD_LOGIC** в языке VHDL.
16. Чем отличается тип данных **STD_LOGIC** от **STD_LOGIC_VECTOR** в языке VHDL.
17. В чём отличие объекта **port** от объекта **signal** в языке VHDL.
18. Какие значения поддерживает тип данных **unsigned** в языке VHDL. Является ли данный тип синтезируемым?
19. Какие значения поддерживает тип данных **integer** в языке VHDL. Является ли данный тип синтезируемым?
20. Является ли высокоимпедансное состояние **Z** синтезируемым в языке VHDL? Если да, то какая схема синтезируется?

21. Каков синтаксис структуры «присваивание по условию» (**conditional signal assignment**) в языке VHDL. В Какую схему данная структура синтезируется?
22. Каков синтаксис структуры «Присваивание по выбору» (**selected signal assignment**) в языке VHDL. В Какую схему данная структура синтезируется?
23. Каков синтаксис оператора **if** в языке VHDL. В Какую схему данная структура синтезируется?
24. Каков синтаксис оператора **case** в языке VHDL. В Какую схему данная структура синтезируется?
25. Для чего в языке VHDL используется структура **process**?
26. Разработать одноразрядную схему сравнения на вентильном уровне на языке VHDL.
27. Разработать на языке VHDL схему дешифратора 2 в 4.
28. Разработать на языке VHDL схему преобразователя двоичного кода в семисегментный.
29. Разработать модуль на VHDL, вычисляющий четырехвходовую функцию XOR (исключающее ИЛИ).
30. Разработать на языке VHDL схему 4-х разрядного счётчика.
31. Разработать на языке VHDL схему 4-х разрядного сумматора чисел со знаком.
32. Для чего применяется generic в языке VHDL.
33. Что значит параметризованная схема?
34. Разработать на языке VHDL схему 8-и разрядного регистра.
35. Разработать на языке VHDL схему сдвигового регистра с параллельной загрузкой.
36. Разработать на языке VHDL схему конечного автомата для детектирования переднего фронта сигнала.
37. Что такое среда тестирования (testbench).
38. Что такое среда тестирования с самопроверкой?
39. Что такое функциональное моделирование?
40. Уровни верификации СБИС.
41. Назначение функциональных блоков stimulus, checker и monitor в среде тестирования?
42. Что такое рандомизация тестовых воздействий (stimulus)
43. Что такое ограниченная (constrained) рандомизация тестовых воздействий
44. Чем отличается кодовое покрытие от функционального покрытия при верификации схемы?
45. Что такое метрика при верификации схемы? Какие метрики используются?
46. Что такое эмуляция схемы?
47. Что такое формальная проверка эквивалентности на этапе синтезирования схемы?
48. Что такое временное моделирование схемы?
49. Написать testbench на VHDL с самопроверкой для одноразрядной схемы сравнения.
50. Написать testbench на VHDL для схемы дешифратора 2 в 4.
51. Написать testbench на VHDL для схемы преобразователя двоичного кода в семисегментный.
52. Написать testbench на VHDL с самопроверкой для 4-х разрядной схемы XOR (исключающее ИЛИ).
53. Написать testbench на VHDL для схемы 4-х разрядного счётчика.
54. Написать testbench на VHDL для схемы 4-х разрядного сумматора чисел со знаком.

55. Написать testbench на VHDL для схемы 8-и разрядного регистра.
56. Написать testbench на VHDL для схемы сдвигового регистра с параллельной загрузкой.
57. Написать testbench на VHDL для схемы детектирования переднего фронта сигнала.
58.
59. Реализовать одноразрядную схему сравнения на базе ПЛИС Spartan 3E. Определить быстродействие схемы и затраченные ресурсы ПЛИС.
60. Реализовать схему дешифратора 2 в 4 на базе ПЛИС Spartan 3E. Определить быстродействие схемы и затраченные ресурсы ПЛИС.
61. Реализовать схему преобразователя двоичного кода в семисегментный на базе ПЛИС Spartan 3E. Определить быстродействие схемы и затраченные ресурсы ПЛИС.
62. Реализовать четырехвходовую функцию XOR (исключающее ИЛИ) на базе ПЛИС Spartan 3E. Определить быстродействие схемы и затраченные ресурсы ПЛИС.
63. Реализовать схему 4-х разрядного счётчика на базе ПЛИС Spartan 3E. Определить быстродействие схемы и затраченные ресурсы ПЛИС.
64. Реализовать схему 4-х разрядного сумматора чисел со знаком на базе ПЛИС Spartan 3E. Определить быстродействие схемы и затраченные ресурсы ПЛИС.
65. Реализовать схему 8-и разрядного регистра на базе ПЛИС Spartan 3E. Определить быстродействие схемы и затраченные ресурсы ПЛИС.
66. Реализовать схему сдвигового регистра с параллельной загрузкой на базе ПЛИС Spartan 3E. Определить быстродействие схемы и затраченные ресурсы ПЛИС.
67. Реализовать схему детектирования переднего фронта сигнала на базе ПЛИС Spartan 3E. Определить быстродействие схемы и затраченные ресурсы ПЛИС.

Внеаудиторная самостоятельная работа обучающихся осуществляется в виде изучения литературы по соответствующему разделу с проработкой материала.

7. Оценочные средства для проведения промежуточной аттестации.

Промежуточная аттестация имеет целью определить степень достижения запланированных результатов обучения по дисциплине (модулю) за определенный период обучения (семестр) и может проводиться в форме зачета, зачета с оценкой, экзамена, защиты курсового проекта (работы).

Данный раздел состоит их двух пунктов: а) Планируемые результаты обучения и оценочные средства для проведения промежуточной аттестации. б) Порядок проведения промежуточной аттестации, показатели и критерии оценивания.

а) Планируемые результаты обучения и оценочные средства для проведения промежуточной аттестации:

Код индикатора	Индикатор достижения компетенции	Оценочные средства
ПК-1: Способен разрабатывать структурные и функциональные схемы электронных систем и комплексов, принципиальных схем устройств с использованием средств компьютерного проектирования, проведением проектных расчетов и технико-экономическим обоснованием принимаемых решений		
ПК-1.1	Разрабатывает эскизный проект, включающий: выбор структурной схемы электронного устройства или системы путем сопоставления различных вариантов и их оценки с точки зрения технических и экономических требований; рассчитывает все необходимые показатели структурной схемы электронного устройства или системы, в том числе показатели качества; выбирает и обосновывает схемы вспомогательных устройств	– *Что такое язык описания аппаратуры HDL.* – *Каковы преимущества разработки схемы на базе HDL по сравнению со схемотехническим способом.* – *Что такое логический синтез схемы.* – *Какие САПР разработки ИС вы знаете?* – *Какие САПР для разработки схем на базе ПЛИС вы знаете?* – *Логический синтез ИС на стандартных ячейках.* – *Логический синтез схем на ПЛИС.* – *Что такое критический путь цифровой схемы?* – *Какие языки описания аппаратуры вы знаете?* – *Чем отличаются синтезируемые структуры языка HDL от несинтезируемых?* – *Какими способами можно повысить быстродействие цифровой схемы?* – *В чём заключается компромисс площадь кристалла/быстродействие?* – *Что такое синхронная цифровая схема?*
ПК-1.2	Производит технико-экономическое обоснование принятого решения с расчетами себестоимости устройства и стоимости его эксплуатации; сравнивает с аналогами по технико-экономическим характеристикам	– *Перечислите основные этапы производства ИС* – *Что включает в себя спецификация на разрабатываемую ИС* – *Какова иерархия проектирования СБИС.* – *Что такое кремниевый уровень проектирования. Какие примитивы применяются на данном уровне.*

		– *Что такое транзисторный уровень проектирования. Какие примитивы применяются на данном уровне.* – *Что такое вентильный уровень проектирования. Какие примитивы применяются на данном уровне.* – *Что такое регистровый уровень проектирования. Какие примитивы применяются на данном уровне.* – *Что такое процессорный уровень проектирования. Какие примитивы применяются на данном уровне.* – *Что такое системный уровень проектирования. Какие примитивы применяются на данном уровне.* – *В чём заключается принцип управления сложностью (абстрагирование) при разработке электроники.* – *Какова современная инфраструктура производства ИС.* – *Что такое IP-блок.* – *Классификация IP-блоков* – *Что представляют собой топологические IP-блоки.* – *Этапы проектирования заказной ИС.* – *Этапы проектирования ИС на стандартных ячейках.* – *Этапы проектирования схемы на базе ПЛИС.*
ПК-3: Способен разрабатывать поведенческие описания моделей стандартных ячеек		
ПК-3.1	Проводит описание моделей стандартных элементов на поведенческом языке	– *Структурное и поведенческое описание цифровой системы.* – *Структура проекта на языке VHDL. Первичный и вторичный модули проекта.* – *Лексические элементы языка VHDL: идентификаторы, разделители, ключевые слова, литералы, комментарии.* – *Декларация объектов языка VHDL (декларация констант, сигналов, переменных). Предопределенные типы данных. Пользовательские типы данных: определение перечислимого типа, численных типов, физического типа, массивов.* – *Декларация объектов языка VHDL (декларация констант, сигналов, переменных). Существенные различия между сигналами и переменными.* – *Атрибуты в языке VHDL:*

		назначение атрибутов, примеры атрибутов типов, массивов, сигналов. – *Разработать одноразрядную схему сравнения на вентильном уровне на языке VHDL.* – *Разработать на языке VHDL схему дешифратора 2 в 4.* – *Разработать на языке VHDL схему преобразователя двоичного кода в семисегментный.* – *Разработать модуль на VHDL, вычисляющий четырехвходовую функцию XOR (исключающее ИЛИ).* – *Разработать на языке VHDL схему 4-х разрядного счётчика.* – *Разработать на языке VHDL схему 4-х разрядного сумматора чисел со знаком.* – *Разработать на языке VHDL схему 8-и разрядного регистра.* – *Разработать на языке VHDL схему сдвигового регистра с параллельной загрузкой.* – *Разработать на языке VHDL схему конечного автомата для детектирования переднего фронта сигнала.* – *Реализовать одноразрядную схему сравнения на базе ПЛИС Spartan 3E. Определить быстродействие схемы и затраченные ресурсы ПЛИС.* – *Реализовать схему дешифратора 2 в 4 на базе ПЛИС Spartan 3E. Определить быстродействие схемы и затраченные ресурсы ПЛИС.* – *Реализовать схему преобразователя двоичного кода в семисегментный на базе ПЛИС Spartan 3E. Определить быстродействие схемы и затраченные ресурсы ПЛИС.* – *Реализовать четырехвходовую функцию XOR (исключающее ИЛИ) на базе ПЛИС Spartan 3E. Определить быстродействие схемы и затраченные ресурсы ПЛИС.* – *Реализовать схему 4-х разрядного счётчика на базе ПЛИС Spartan 3E. Определить быстродействие схемы и затраченные ресурсы ПЛИС.* – *Реализовать схему 4-х разрядного сумматора чисел со знаком на базе ПЛИС Spartan 3E. Определить быстродействие схемы и затраченные ресурсы ПЛИС.*

		– *Реализовать схему 8-и разрядного регистра на базе ПЛИС Spartan 3E. Определить быстродействие схемы и затраченные ресурсы ПЛИС.* – *Реализовать схему сдвигового регистра с параллельной загрузкой на базе ПЛИС Spartan 3E. Определить быстродействие схемы и затраченные ресурсы ПЛИС.* – *Реализовать схему детектирования переднего фронта сигнала на базе ПЛИС Spartan 3E. Определить быстродействие схемы и затраченные ресурсы ПЛИС.*
ПК-3.2	Использует целевые системы автоматизированного проектирования	– *Последовательные операторы. Оператор присваивания: синтаксическая конструкция, модели задержек* – *Последовательные операторы. Оператор условия* – *Последовательные операторы. Оператор выбора.* – *Последовательные операторы. Оператор ожидания* – *Последовательные операторы. Оператор повторения без итерационной схемы.* – *Последовательные операторы. Оператор повторения с итерационными схемами. Операторы перехода к следующему циклу и выхода из цикла.* – *Параллельные операторы. Оператор процесса* – *Параллельное присваивание* – *Параллельные операторы. Оператор блока.* – *Реализация комбинационных логических схем, заданных в алгебраической форме или табличном представлении, примеры реализации.* – *Разработать двухразрядную схему сравнения на основе двух экземпляров одноразрядной схемы сравнения. Использовать комментарии для описания кода.* – *Разработать на языке VHDL схему дешифратора 3 в 8 на основе экземпляров схемы дешифратора 2 в 4. Использовать комментарии для описания кода.* – *Разработать на языке VHDL схему 16-и разрядного сумматора чисел со знаком на основе экземпляров 4-х разрядного сумматора. Использовать комментарии для описания кода.*

		– *Разработать на языке VHDL схему 8-и разрядного регистра. Использовать комментарии для описания кода.* – *Разработать на языке VHDL схему конечного автомата для реализации защиты от дребезга. Использовать комментарии для описания кода.* *Подготовить проектную документацию: RTL-код и файл ограничений (топологических и временных) для реализации проекта на базе ПЛИС для следующих проектов:* – *Восьмиразрядная схема сдвига с управляющим входом, определяющим направление сдвига.* – *Приоритетный шифратор 8 в 3* – *Преобразователь двоичного кода в двоично-десятичный* – *4-х разрядный сумматор чисел с плавающей точкой.* – *8-и разрядный FIFO буфер* – *4-х разрядный ШИМ* – *Сторожевой таймер* – *Схема стека* – *Арифметико-логическое устройство* – *Регистровый файл* – *Схема деления*

б) Порядок проведения промежуточной аттестации, показатели и критерии оценивания:

Промежуточная аттестация по дисциплине «Языки описания цифровой аппаратуры. VHDL» включает теоретические вопросы, позволяющие оценить уровень усвоения обучающимися знаний, и практические задания, выявляющие степень сформированности умений и владений, проводится в форме зачёта.

Показатели и критерии оценивания зачёта:

– на оценку «отлично» (5 баллов) – обучающийся демонстрирует высокий уровень сформированности компетенций, всестороннее, систематическое и глубокое знание учебного материала, свободно выполняет практические задания, свободно оперирует знаниями, умениями, применяет их в ситуациях повышенной сложности.
– на оценку «хорошо» (4 балла) – обучающийся демонстрирует средний уровень сформированности компетенций: основные знания, умения освоены, но допускаются незначительные ошибки, неточности, затруднения при аналитических операциях, переносе знаний и умений на новые, нестандартные ситуации.
– на оценку «удовлетворительно» (3 балла) – обучающийся демонстрирует пороговый уровень сформированности компетенций: в ходе контрольных мероприятий допускаются ошибки, проявляется отсутствие отдельных знаний, умений, навыков, обучающийся испытывает значительные затруднения при оперировании знаниями и умениями при их переносе на новые ситуации.

– на оценку «неудовлетворительно» (2 балла) – обучающийся демонстрирует знания не более 20% теоретического материала, допускает существенные ошибки, не может показать интеллектуальные навыки решения простых задач.
– на оценку «неудовлетворительно» (1 балл) – обучающийся не может показать знания на уровне воспроизведения и объяснения информации, не может показать интеллектуальные навыки решения простых задач.

ПРИЛОЖЕНИЕ 3

Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Магнитогорский государственный технический университет

им. Г.И. Носова»

(ФГБОУ ВО «МГТУ им. Г.И. Носова»)

Лабораторный практикум

Языки описания цифровой аппаратуры (VHDL)

Разработал: к.т.н., доцент каф. ЭиМЭ Швидченко Н.В.

Магнитогорск, 2023

Содержание

Введение

Традиционное схемное проектирование достигло в своем развитии предела и используется в настоящее время лишь для сравнительно простых проектов разрабатываемой электроники. Основным недостатком схемного описания проектов сложной электроники, включающей большое количество компонентов и модулей, является высокая трудоемкость отладки. На смену схемному описанию пришли языки описания аппаратуры, такие как VHLD и Verilog. В отличие от традиционных языков программирования, описывающих последовательность выполняемых команд, языки описания аппаратуры описывают структуру и связи разрабатываемого устройства. Такое описание разрабатываемого устройства называется структурным (параллельным). Второй подход к описанию устройства называется поведенческим (последовательным). При таком подходе описывается не структура устройства, а порядок его функционирования (поведение). Языки описания аппаратуры включат как параллельные, так и последовательные функции, т.е. с их помощью можно описать как структуру разрабатываемого устройства, так и его поведение в зависимости от входных воздействий. Здесь следует сделать важное замечание:

Последовательное (поведенческое) описание устройства не всегда может быть корректно скомпилировано в синтезируемые конструкции, т.е. синтезированная схема устройства может работать не так, как задумывал разработчик. В тоже время структурное (параллельное) описание однозначно синтезируется в описанную структуру. Таким образом, при описании разрабатываемого устройства следует отдавать предпочтение структурному описанию или, как говорят, использовать синтезируемые конструкции. Последовательное (поведенческое) описание в свою очередь эффективно используется на этапе функционального моделирования работы разрабатываемого устройства. Тем не менее, использование последовательных операторов языков описания аппаратуры для синтеза разрабатываемого устройства допускается при условии жесткого соблюдения ряда правил (данные правила будут рассмотрены в соответствующей лабораторной работе).

Данный лабораторный практикум призван научить основам проектирования электроники с помощью языка описания аппаратуры VHLD. При этом основной упор делается на развитие навыков написания грамотного и самое главное синтезируемого кода, т.е. такого кода, который будет гарантированно скомпилирован в работающую схему. Поэтому вместо подробного описания синтаксиса VHLD и всевозможных конструкций языка практикум ограничивается обзором сравнительно небольшого набора синтезируемых конструкций, использующих десяток типовых шаблонов VHDL-кода, синтезируемых в типовые схемы. Данные шаблоны могут быть легко интегрированы при проектировании больших сложных систем.

Для обеспечения процесса сквозного проектирования вплоть до готового устройства было решено использовать программируемую логику (ПЛИС), а именно отладочную плату **NI Digital Electronics FPGA Board** на основе ПЛИС **Xilinx Spartan-3E**. В качестве программного обеспечения используется САПР **ICE WebPACK** фирмы Xilinx.

1. Описание платы NI Digital Electronics FPGA

Плата **NI Digital Electronics FPGA** является результатом совместного сотрудничества компаний NI и Xilinx, являющихся крупнейшими в мире производителями программного обеспечения измерительных систем и программируемых логических интегральных схем (ПЛИС). Внешний вид платы NI Digital Electronics FPGA представлен на рисунке 1. NI Digital Electronics FPGA Board содержит шесть каналов для ввода аналоговых входных сигналов AI0 – AI5, а также 32 цифровые входные (выходные) линии общего пользования – GPIO31.

Описание сигналов, подаваемых на сигнальные зоны макетирования: ВВ1 – зона макетирования для подключения к ЦАП, АЦП, кнопкам, движковым переключателям, внешнему синхросигналу и линиям общего назначения ПЛИС. ВВ2 - зона макетирования для подключения к линиям общего назначения ПЛИС и источникам питания. ВВ3 - зона макетирования для управления семисегментным индикатором на два знакоместа, светодиодами, подключения к источникам питания ВВ4 - зона макетирования для подключения к сигналам NI ELVIS, включая аналоговые входные сигналы, аналоговые выходные сигналы, функциональные генераторы, источники питания, цифровые входные/выходные сигналы. ВВ5 – зона макетирования для подключения к сигналам NI ELVIS, включая регулируемые источники питания, цифровые входные/выходные сигналы, выходные сигналы счётчиков, общие точки сигналов.

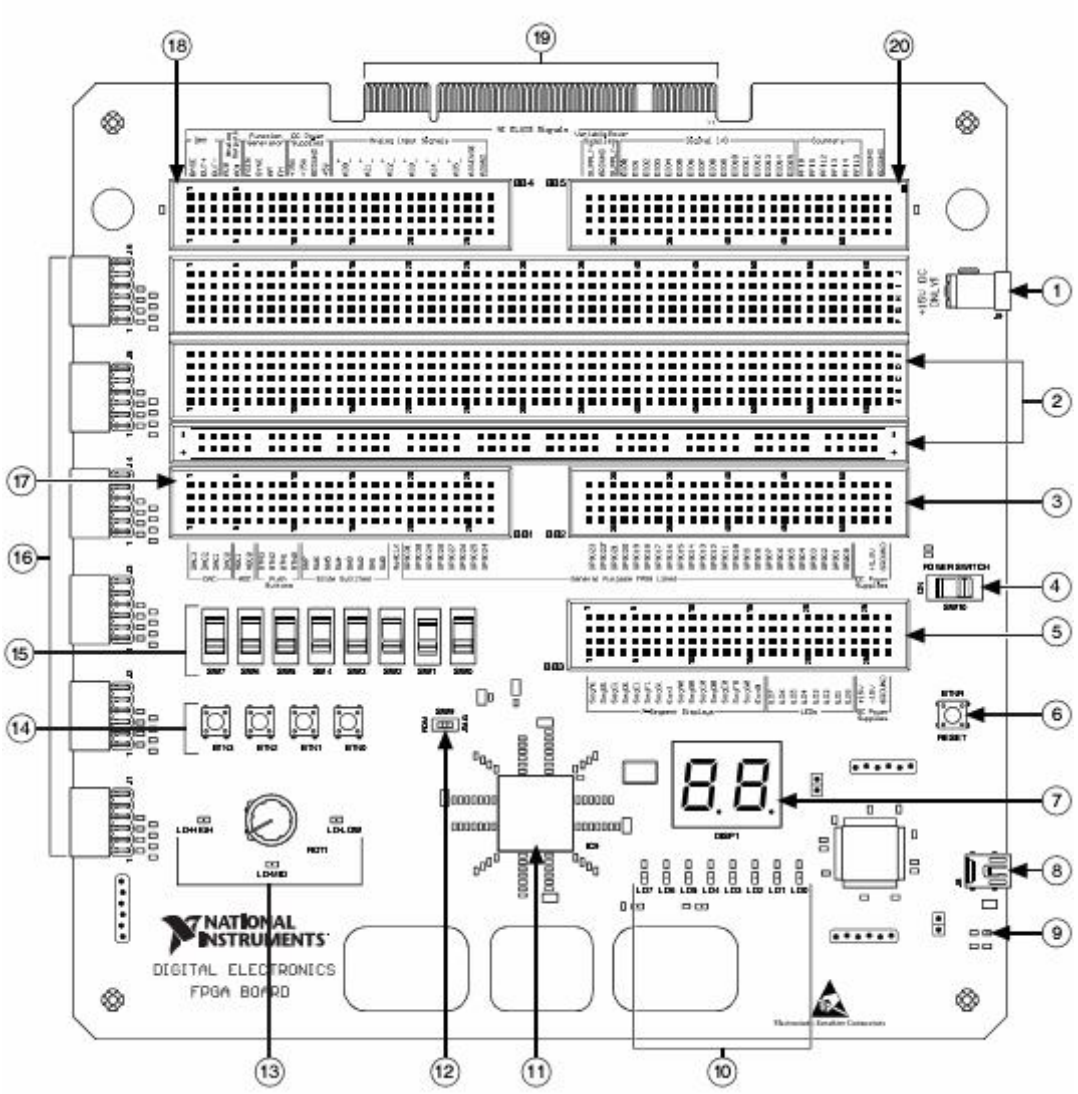


Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.** – Внешний вид платы NI Digital Electronics FPGA Board.

Описание структуры платы: 1 – разъем питания (15В) ; 2 –макетной платы для возможности создания дополнительной обвязки; 3 – блок макетной платы ВВ2; 4 – выключатель питания; 5 – блок макетной платы ВВ3; 6 – кнопка сброса; 7 – семисегментные индикаторы; 8– USB-соединитель; 9 – LD-G- светодиод; 10 – светодиоды; 1 1– FPGA Xilinx Spartan 3E; 12– переключатель прошивки ПЛИС через ROM/JTAG ; 13 – датчик угла поворота с кнопкой; 14 – кнопки; 15 – движковые переключатели; 16 – 6 коннекторов типа PMOD (2x6); 17 – блок макетной платы ВВ1; 18 – блок макетной платы ВВ4; 19 – (NI ELVIS)- соединитель; 20 – блок макетной платы ВВ5; 21 – программатор; 22– кварцевый генератор 50МГц; 23– защита платы от статического электричества

2. Описание системы автоматического проектирования WebPack

WebPACK – это САПР проектирования цифровых устройств на базе микросхем ПЛИС CPLD и FPGA фирмы Xilinx. Данная система является бесплатным вариантом коммерческой САПР этой же фирмы под названием Integrated Synthesis Environment (ISE) и доступна для свободного скачивания через сеть Internet (www.xilinx.com). Основное отличие бесплатной версии от ее платного аналога состоит в отсутствии поддержки микросхем, емкость которых выше 1,5 млн системных вентилей.

WebPACK состоит из набора модулей, каждый из которых выполняет свои специализированные функции. Основные модули пакета следующие:

- редактор схемного ввода;

- текстовый редактор с поддержкой языков описания аппаратуры VHDL и Verilog;

- CORE Generator – генератор оптимизированных IP-ядер;

- редактор тестовых воздействий для программы моделирования;

- программа функционального и временного моделирования;

- генератор VHDL/Verilog кода;

- программа автоматического размещения и трассировки ПЛИС;—программы «ручного» размещения и оптимизации проекта;

- программа загрузки конфигурационной последовательности в ПЛИС FPGA и программирования ПЛИС CPLD и ППЗУ.

Большинство модулей САПР WebPACK имеют как графический интерфейс пользователя, так и интерфейс командной строки. САПР WebPACK может работать под операционными системами Windows, Linux и Sun Solaris.

Процесс разработки цифровых устройств в среде WebPACK состоит из следующих этапов.

1. Ввод описания проектируемого устройства в схемотехнической форме или с использованием языков описания аппаратуры (HDL), таких, как VHDL и Verilog.

2. Синтез устройства, то есть преобразование описания устройства, полученного на первом этапе, в описание на уровне логических вентилей.

3. Реализация устройства, то есть преобразование описания устройства на уровне логических вентилей в физическое описание для конкретной микросхемы ПЛИС.

4. Формирование конфигурационной последовательности для микросхе-

мы ПЛИС.

После каждого из этапов 1,2 и 3 возможно, а в большинстве случаев и необходимо для успешного завершения проекта выполнение процедуры моделирования и верификации полученного описания устройства.

На рисунке 2.1 представлена обобщенная схема проектирования цифровых устройств в САПР WebPACK.

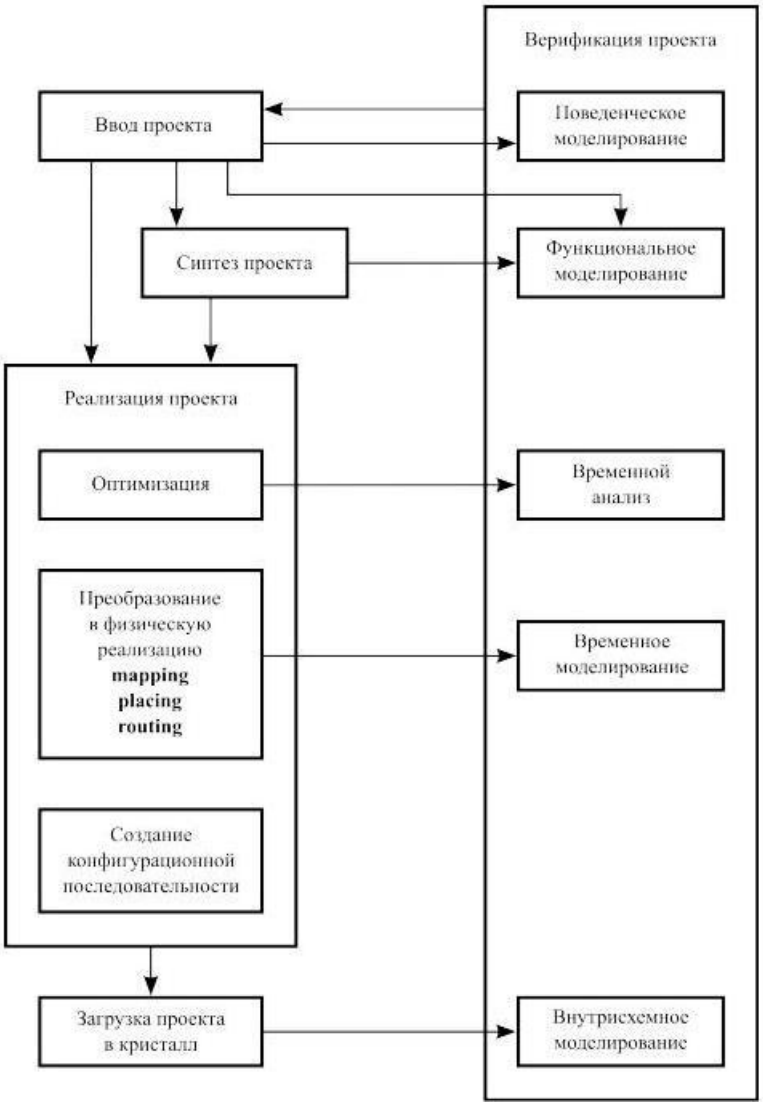


Рисунок 2.1 - Обобщенная схема проектирования цифровых устройств в САПР WebPACK

2.1 Создание проекта

Для начала необходимо запустить ISE Design Suite и создать проект. Для этого зайдите в меню File и создайте новый проект, нажатием на кнопку New Project. Задайте начальные настройки проекта, представленные на рисунке 2.2. Имя проекта укажите латиницей, например eq1, а так же папку, где будут храниться ваши проекты, все папки должны иметь только латинские названия (На английском). В строке Top-level source type выберите HDL – язык описания аппаратуры. Нажмите Next.

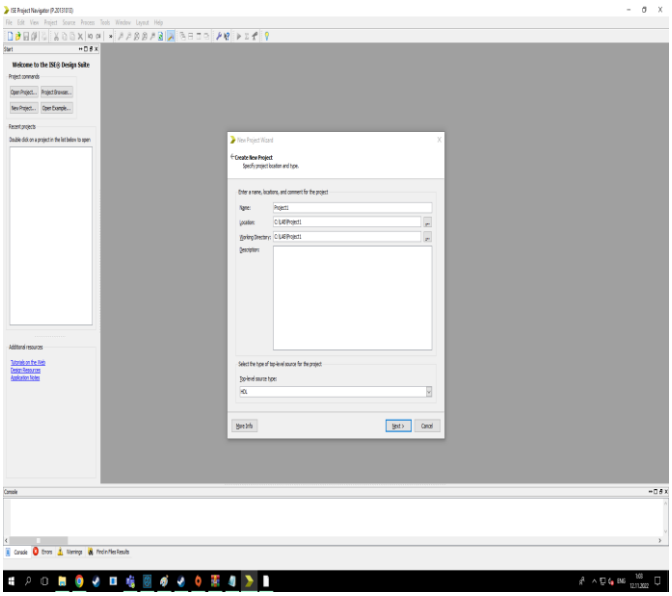


Рисунок 2.2 – начальное окно настроек

Далее, установите настройки для отладочной платы, представленные на рисунке 2.3. Необходимо установить семейство Spartan3E и сама микросхема XC3S500E. А так же выберите тип корпуса микросхемы FT256. Симулятор необходимо установить ISim для проверки схемы в симуляторе. Язык используем VHDL. Нажмите Next и Finish.

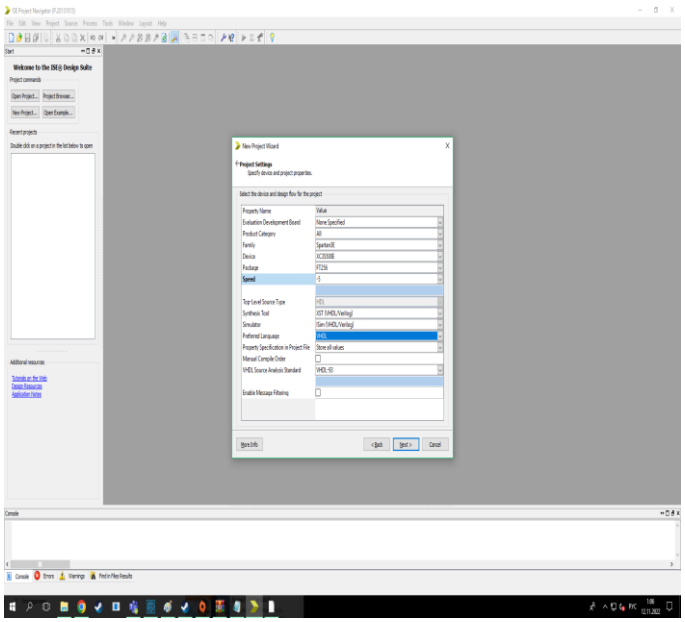


Рисунок 2.3 – окно настроек отладочной платы

Добавление VHDL модуля

Весь проект размещается в дереве проекта, представленном на рисунке 2.4. Для создания рабочего файла, нажмите в нем ПКМ и в контекстном меню вберете New Source. В предложенном списке, создаем VHDL Module, зададим имя файла, расположение и выставим флаг Add to project.

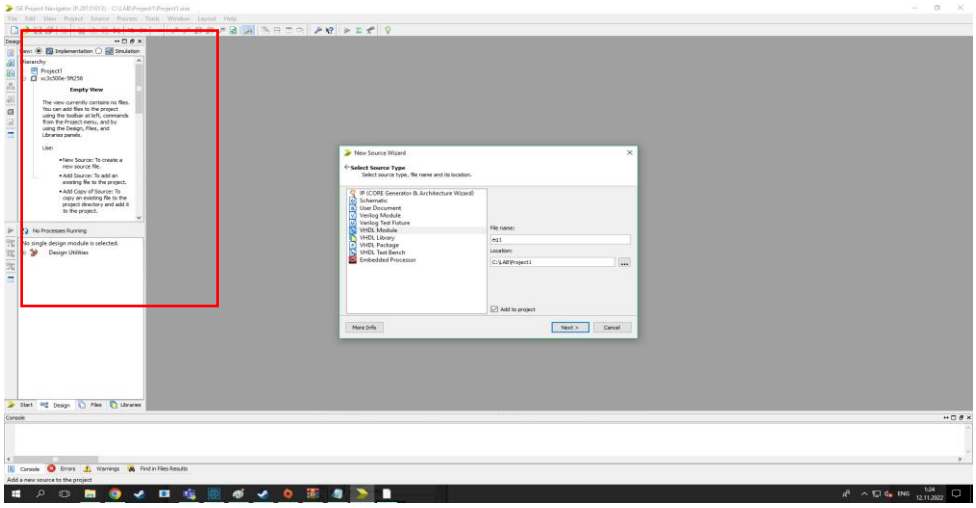


Рисунок 2.4 – настройка VHDL модуля

В открывшемся окне можно изменить имя архитектуры, а также задать переменные. На рисунке 2.5 представлен пример задания переменных A – на вход, B – на выход и четырехразрядная переменная C – на выход. Разрядность переменной задается флагом Bus и задаются старший бит MSB и младший бит LSB. Нажмите Next и Finish.

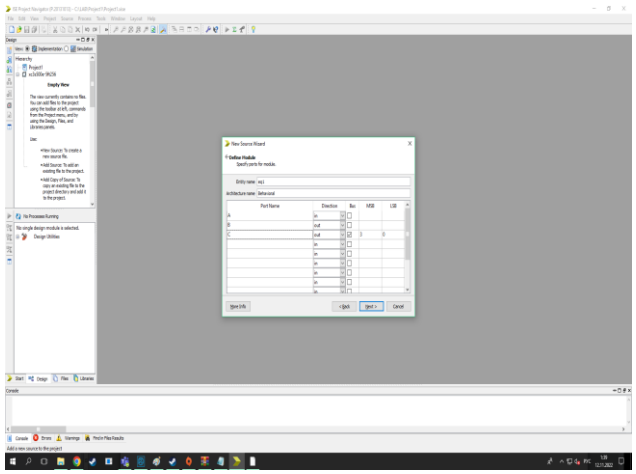


Рисунок 2.5 – Задание переменных

Добавление настроек ограничения

В файле ограничений описываются определенные условия, налагаемые на процессы синтеза и реализации схемы. Для наших целей основным типом ограничения является назначение портов ввода и вывода верхнего уровня и задание минимальной тактовой частоты. В процессе

реализации взаимодействия с портами ввода и вывода, необходимо сопоставить их с физическими выводами устройства, которые уже подключены к плате прототипирования. Команды взаимодействия с портами представлены в приложении 1.

Другой тип ограничений связан с синхронизацией, которая определяет минимальную тактовую частоту для облегчения работы генератора платы. Информация об ограничениях хранится в текстовом файле с расширением .ucf (для файла пользовательских ограничений).

Для создания файла ограничений, нажмите в дереве проект ПКМ и в контекстном меню вберете New Source. В предложенном списке, создаем Implementation Constraints File, представленный на рисунке 2.6, зададим имя файла, расположение и выставим флаг Add to project.

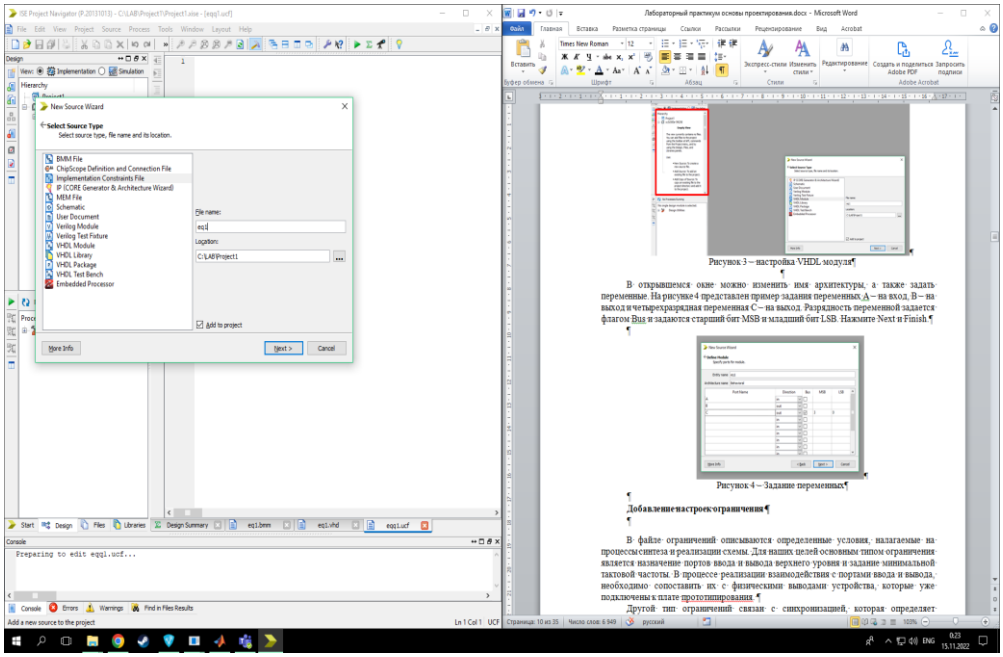


Рисунок 2.6 – Создание файла ограничений

Синтез проекта

После того, как были созданы и написаны модули, необходимо синтезировать проект. В процессе синтеза из файлов HDL-описаний проектируемого устройства формируется файл списка соединений в формате EDIF (Electronic Data Interchange Format). Синтезированный файл представляет собой текстовое (ASCII) описание проекта на более низком логическом уровне в формате, воспринимаемом программами трассировки Xilinx. Если исходные описания проекта представлены в графической, в частности схемотехнической форме, то автоматически выполняется их преобразование в требуемый HDL-формат.

На рисунке 2.7 представлено окно процессов. Здесь располагаются основные узлы управления проектом. Для синтеза проект нажмите левой кнопкой на имени файла верхнего уровня в иерархическом представлении проекта и нажать кнопку Implement Top Module:

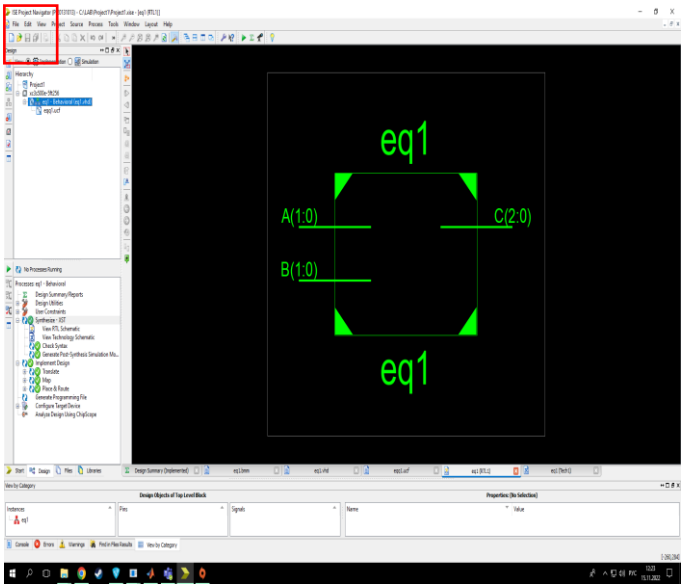


Рисунок 2.7 – Окно процессов ISE WebPACK Design Software

Design Symmary/Reports - основная информация о проекте. Здесь контролируются ресурсы ПЛИС. А также все отчеты о разводке нашего проекта в заданном кристалле можно найти тут.

Design Utilities - здесь расположены вспомогательные утилиты для работы в ISE. К ним относится утилита по созданию схематического образа компонента. А так же обзоры функциональных HDL моделей.

User Constraints - пользовательские ограничения. Собранные из всех файлов, которые были загружены в проект формата «*.ucf». А так же возможность запустить приложение PlanAhead, которое позволяет более подробно задать ограничения для разводчика связей ПЛИС .

Synthesize - XST - синтезатор типа XST. Синтезирует проект для конкретно выбранного типа кристалла ПЛИС. Позволяет посмотреть «RTL» модель написанного или нарисованного блока. Так же проверка синтаксиса на наличие ошибок.

Implement Design - выполняет задачу размещения полученных после синтеза компонентов на заданном кристалле, учитывая пользовательские ограничения, заданные в файле «*.ucf». На этом этапе происходит оптимизации проекта, и привязка сигналов к выводам ПЛИС

Generate Programming File - генератор программного файла, с которого происходит конфигурация ПЛИС.

Configure Target Device - конфигуратор. Используя полученное в предыдущем пункте размещение на кристалле, создает файл прошивки для ПЛИС. А так же отсюда запускается утилита «IMPACT», которую рассмотрим подробнее при загрузке проекта в кристалл, она позволяющая программировать PROM и FPGA.

Analyze Design Using ChipScope - логический анализатор по средствам интерфейса JTAG.

В результате успешного синтеза проекта, напротив выполненных пунктов появится отметка в виде зеленого флага. При наличии ошибок, необходимо внести изменения и двойным щелчком нажать на незавершенный процесс.

Загрузка проекта в кристалл

Для загрузки проекта, необходимо сгенерировать файл прошивки, на рисунке 2.8.

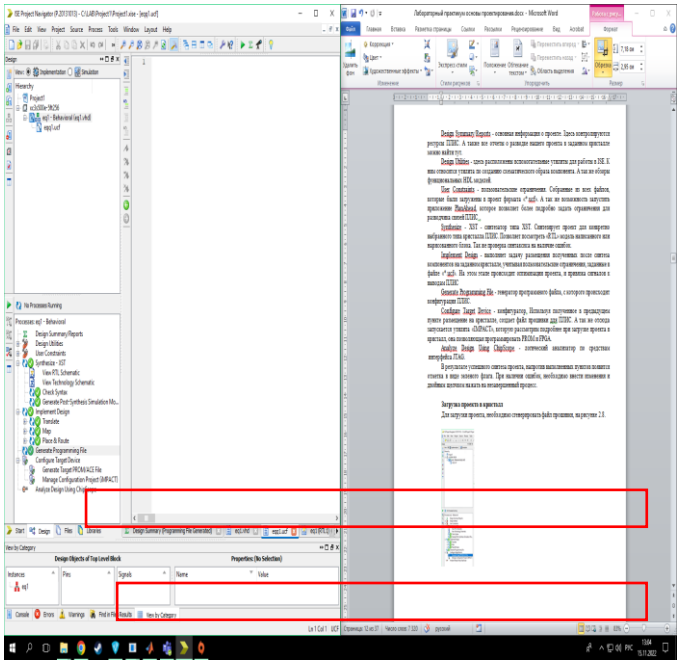


Рисунок 2.7 – Генерация прошивки

Далее в пункте Configure Target Device выбрать Manage Configuration Project (iMPACT). На рисунке 2.8 представлено окно подключения к программатору, в кортом вбираем пункт Boundary Scan – сканирование периферийных устройств.

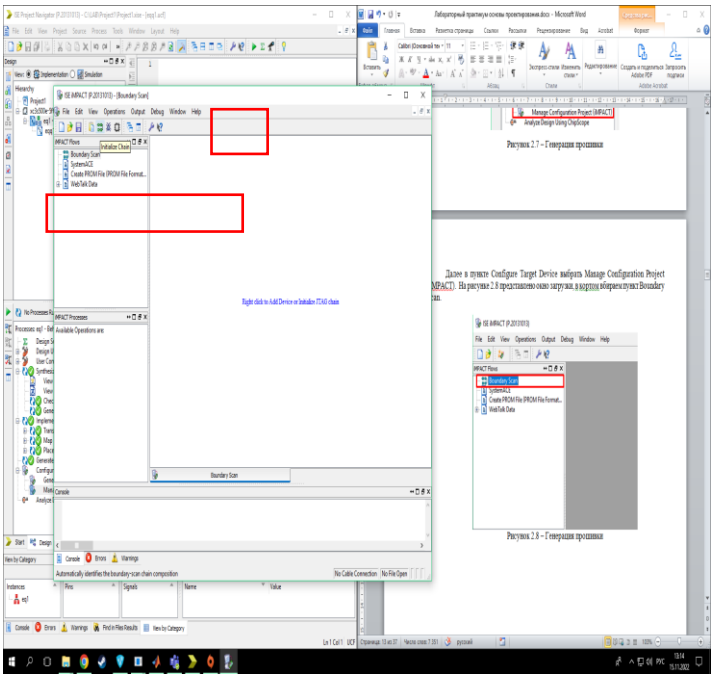


Рисунок 2.8 – Подключение к программатору ПЛИС

Затем нажмите на кнопку Initialize Chain, для того, что бы программа подключилась через программатор к программируемому устройству и определила микросхемы находящиеся на линии программирования. На рисунке 2.9 представлено диалоговое окно указания конфигурационных файлов для ОЗУ и ПЗУ ПЛИС. Нажмите накнопку Yes.

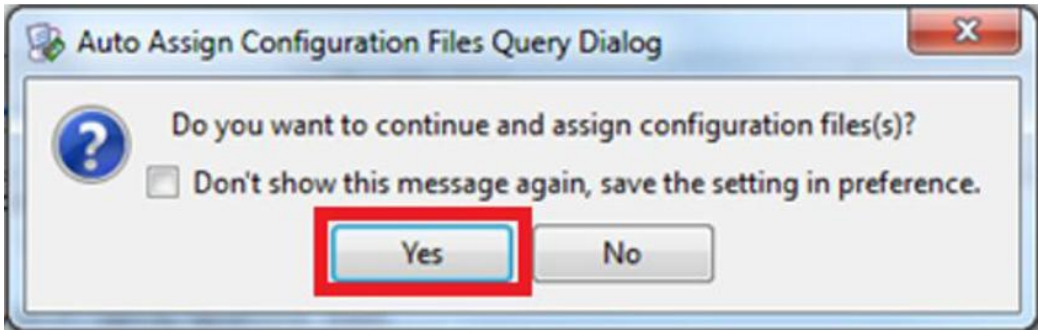


Рисунок 2.9 – Предложение указать конфигурационные файлы

В появившемся списке, найдите свой файл прошивки для ОЗУ ПЛИС на рисунке 2.10. А так же для прошивки ПЗУ ПЛИС на рисунке 2.11.

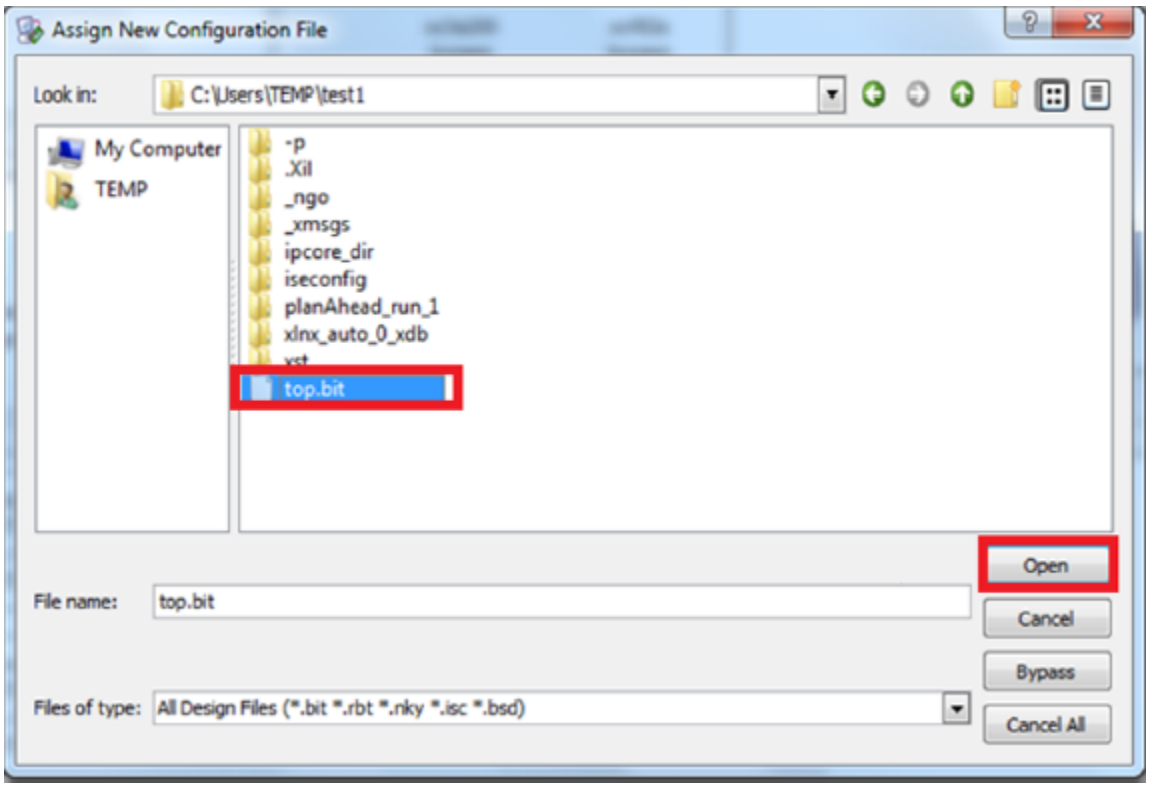


Рисунок 2.10 – Выбор прошивки ОЗУ

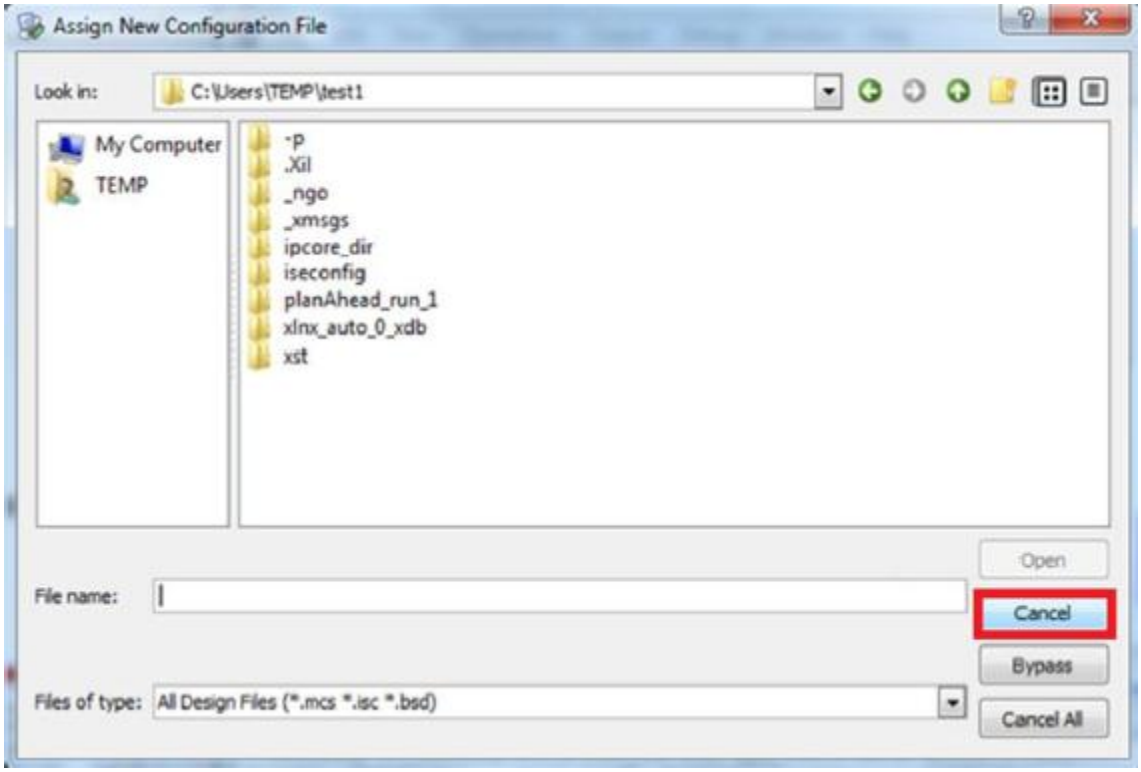


Рисунок 2.11 – Выбор прошивки ПЗУ

В следующем окне настроек программирование, на рисунке 2.12 нажмите Apply.

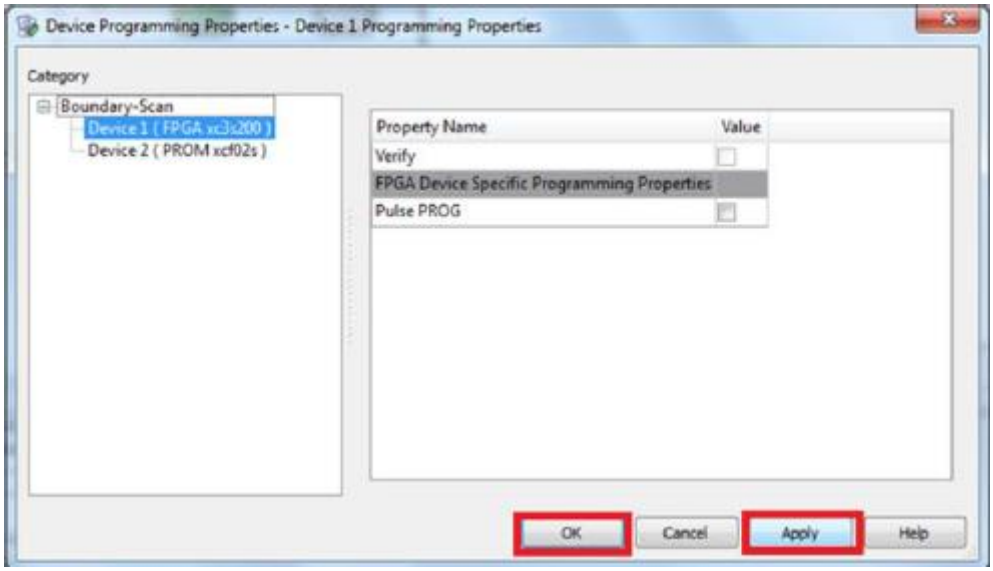


Рисунок 2.12 – Настройки программирования

После чего необходимо выбрать устройство ПЛИС в списке устройств JTAG цепи, представленном на рисунке 2.13. И запрограммировать его, нажав кнопку Program.

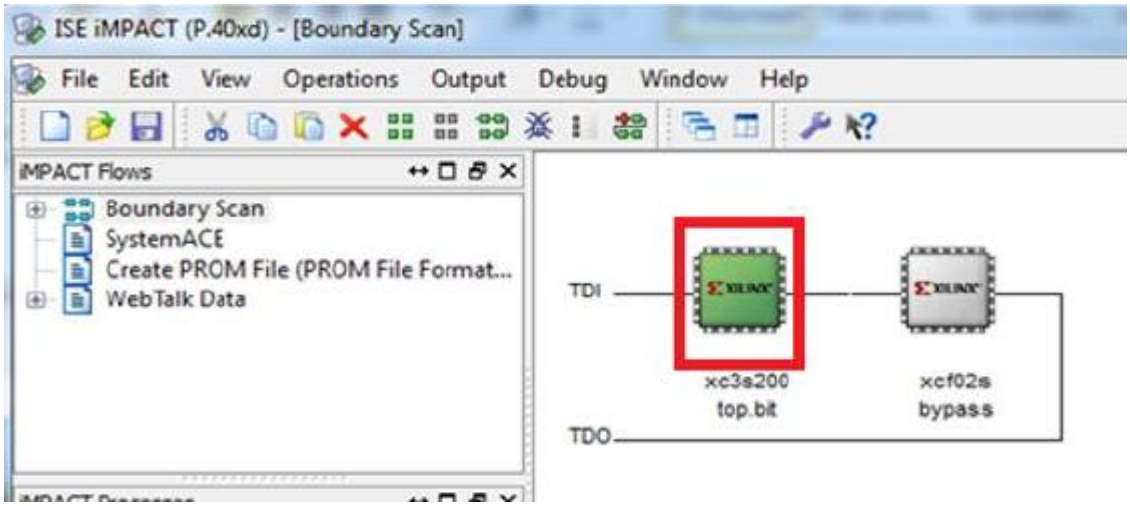


Рисунок 2.13 – Выбрать устройства ПЛИС в списке устройств

JTAG

2.2 Верификация проекта

Верификация это комплекс функциональных проверок, выявляющих, соответствие системы предъявленным к ней требованиям и техническому заданию. В её ходе, проводится валидация, для определения соответствия корректности постановки технического задания здравому смыслу, а так же тестирования отдельных аспектов работы.

Верификация должна осуществляться в соответствии с запланированными мероприятиями с целью удостовериться, что выходные данные проектирования и разработки соответствуют входным требованиям. Записи результатов верификации и всех необходимых действий должны поддерживаться в рабочем состоянии .

Валидация проекта и разработки должна осуществляться в соответствии с запланированными мероприятиями, с целью удостовериться, что полученная в результате продукция соответствует требованиям к установленному или предполагаемому использованию, если оно известно. Где это практически возможно, валидация должна быть завершена до поставки или применения продукции. Записи результатов валидации и всех необходимых действий должны поддерживаться в рабочем состоянии.

В ISim для проведения данного этапа существует интегрированная среда тестирования функционального описания устройства. Схема которой представлена на рисунке 2.14

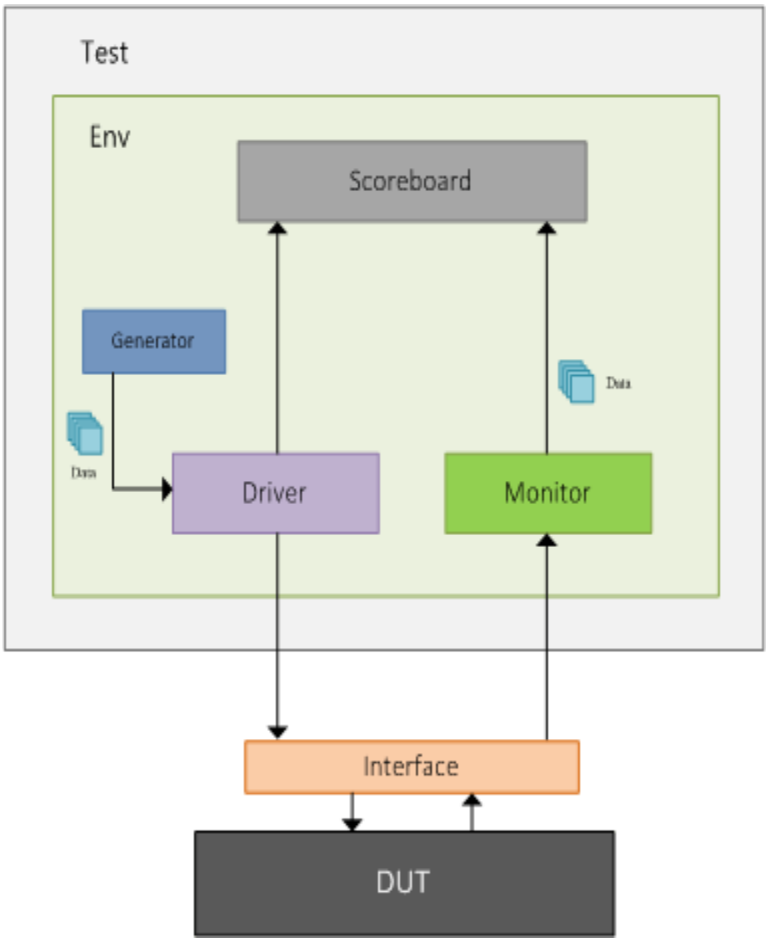


Рисунок 2.14 – Схема тестовой среды программы

DUT – расшифровывается как Design Under Test, обозначающий аппаратный проект, написанный на VHDL после валидации.

Добавление тестового модуля

Для создания тестового модуля, представленного на рисунке 2.15, необходимо в древе проекта нажать ПКМ и в контекстном меню выбрать New Source. В предложенном списке, создаем VHDL Test Bench, зададим имя файла, расположение и выставим флаг Add to project.

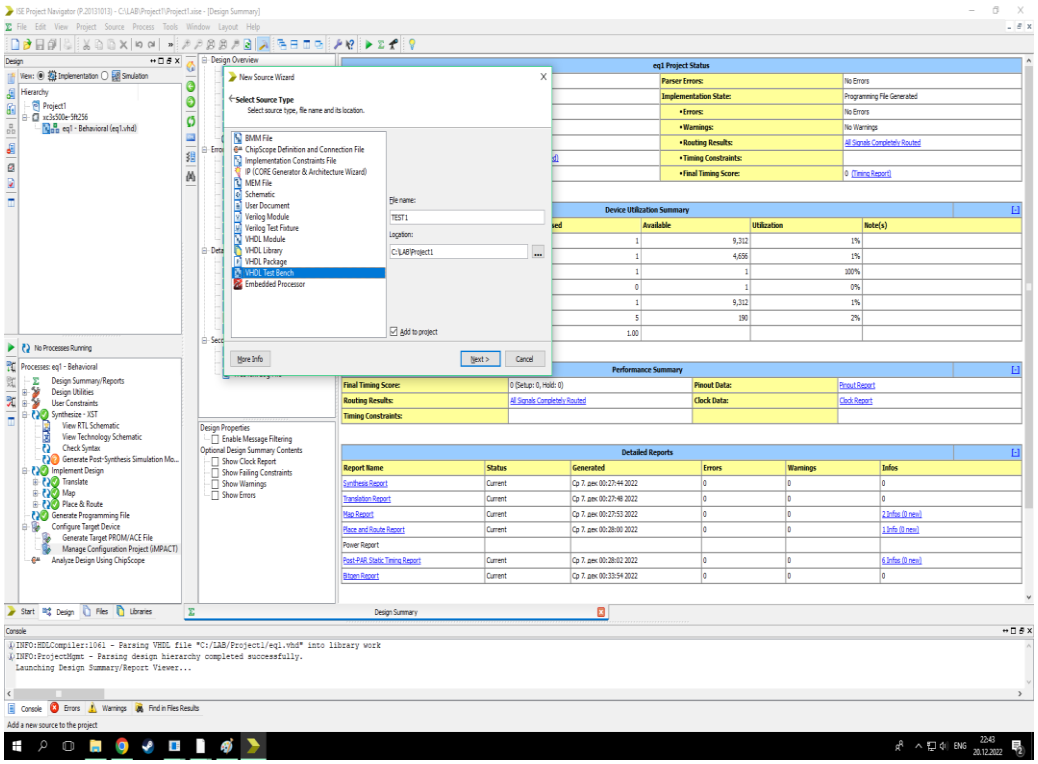


Рисунок 2.15 – Создание Test Bench модуля

Далее, на рисунке 2.16 выбираем проект, который нам нужно протестировать. Убедитесь, что сам проект активен и находится в указанной папке без повреждений.

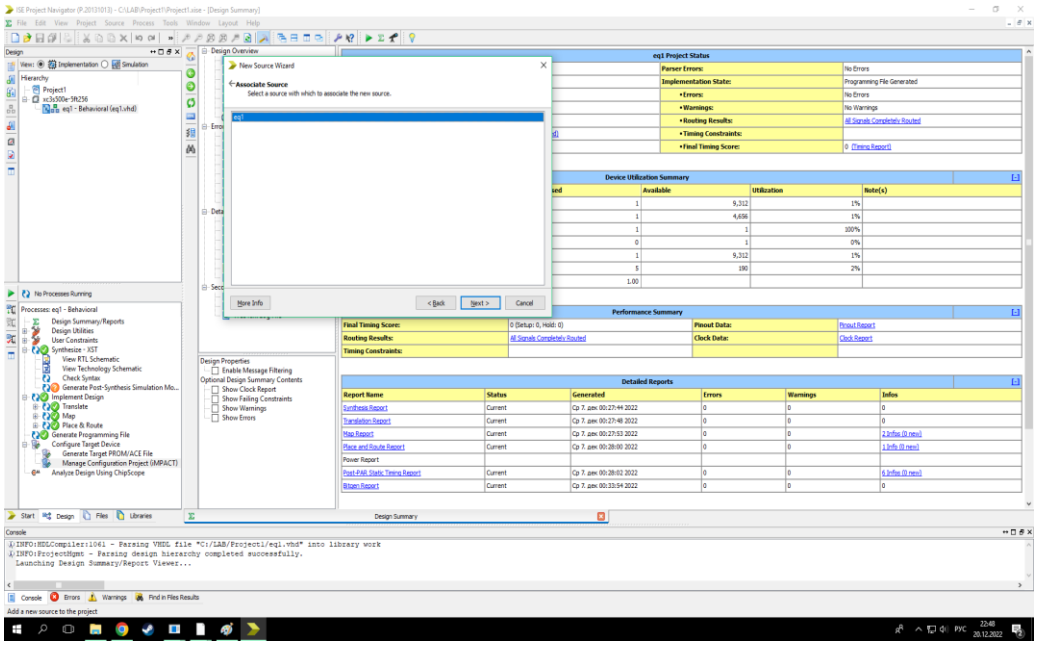


Рисунок 2.16 – Выбор проекта тестирования

Цифровые системы строятся зачастую на основе нескольких более простых унифицированных блоков. При этом на верхнем уровне иерархии описания цифровой системы структура данных унифицированных блоков не раскрывается – они представляются «черными ящиками» реализующими конкретные функции. Структура данных блоков раскрывается на более низком уровне иерархии описания системы. В свою очередь, каждый «черный ящик» может состоять из нескольких еще более простых унифицированных блоков. Структура этих унифицированных блоков раскрывается на еще более низком уровне иерархии описания системы и т.д. Такой подход, значительно упрощающий процесс проектирования и анализа сложных цифровых систем, получил название структурное или иерархическое описание схем. В языках описания аппаратуры HDL унифицированные блоки, которые используются в виде готовых объектов на верхнем уровне описания системы, получили название instance (образ объекта), а процедура размещения блока на верхнем уровне - instantiation.

3 Проектирование комбинационных схем на вентильном уровне

Введение

В данной работе на примере простой схемы сравнения подробно рассмотрена структура синтезируемого VHDL-кода. Описание работы цифровых комбинационных схем в данной работе представлено исключительно на вентильном уровне абстракции (gate level), т.е. применяются только базовые логические вентили, реализующие основные логические операции (НЕ, И, ИЛИ, ИСКЛЮЧАЮЩЕЕ ИЛИ). Рассмотрен структурный (иерархический) подход к проектированию цифровых систем, при котором система на верхнем уровне проектирования рассматривается как соединение более простых унифицированных блоков, которые, в свою очередь, также строятся из более простых блоков и т.д.

3.1 Описание схем в дизъюнктивной нормальной форме

Рассмотрим одноразрядную схему сравнения (1-bit equality comparator) , содержащую два входа *i0*, *i1* и один выход *eq*. Выход *eq* схемы сравнения принимает активное значение при условии, что входы *i0* и *i1* эквивалентны. Таблица истинности (truth table) одноразрядной схемы сравнения представлена в таблице 1.

Таблица 3.1 – Таблица истинности одноразрядной схемы сравнения

i0	i1	eq
0	0	1
0	1	0
1	0	0
1	1	1

Допустим, что для реализации одноразрядной схемы сравнения мы решили использовать базовые логические вентили (logic gates), такие как: *НЕ* (*NOT*), *И* (*AND*), *ИЛИ* (*OR*), *Исключающее ИЛИ* (*XOR*). Одной из форм записи логической функции, описывающей работу схемы, является запись функции в совершенной дизъюнктивной нормальной форме, т.е. в виде суммы минтермов (sum-of-products canonical form) . Логическая функция одноразрядной схемы сравнения, записанная в совершенной дизъюнктивной нормальной форме, имеет вид:

$$eq = i0 \cdot i1 + \overline{i0} \cdot \overline{i1}$$

Один из возможных вариантов VHDL-кода, описывающего рассматриваемую схему сравнения приведен в листинге 1.1. Проанализируем языковые конструкции приведенного кода в следующих подразделах.

Листинг 3.1 – Реализация одноразрядной схемы сравнения на вентильном уровне

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
---------------------------------
entity eq1 is
```

```
        Port ( i0 : in    STD_LOGIC;
               i1 : in    STD_LOGIC;
               eq : out   STD_LOGIC);
end eq1;
---------------------------------
architecture sop_arch of eq1 is
        signal p0 , p1 : STD_LOGIC;
begin
        eq <= p0 or p1 ;                        -- сумма минтермов
        p0 <= (not i0) and (not i1) ;       -- определение минтерма
        p1 <= i0 and i1 ;                       -- определение минтерма
end sop_arch;
```

Основные лексические правила

Код VHDL является нечувствительным к способу написания (регистру) символов (case insensitive), т.е. прописные и строчные буквы являются взаимозаменяемыми. Кроме того в коде VHDL допускается вставлять дополнительные пробелы и пустые строки. Использование дополнительных пробелов и пустых строк позволяет сделать код более понятным.

Идентификатором в языке VHDL называется имя объекта (константы, переменной, функции, сигнала, порта, подпрограммы, объекта проекта, метки). Идентификатор может содержать до 26 букв, цифр и нижних подчеркиваний (не допускается использовать более одного символа нижнего подчеркивания подряд). Начинаться идентификатор должен только с буквы. В примере используются следующие идентификаторы: eq1, i0, i1, eq, sop_arch, p0, p1.

В языке VHDL существуют зарезервированные ключевые слова. В данном учебном пособии все ключевые слова выделены синим цветом (листинг 3.1).

Комментарий начинается с двух смежных дефисов и продолжается до конца строки. При синтезе VHDL-проекта комментарии игнорируются. В данном учебном пособии все комментарии выделены зеленым цветом (листинг 2.1).

Используемые библиотеки

Описание объекта на VHDL состоит из трех частей (листинг 2.1): объявления используемых библиотек (library, use), описания интерфейса объекта (entity) и его внутренней структуры (architecture).

Первые две строчки в примере (листинг 3.1) объявляют используемые библиотеки:

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;
```

В данном случае объявляется библиотека IEEE.STD_LOGIC_1164, что позволяет использовать соответствующие типы данных, операторы и функции, определенные в данной библиотеке.

Интерфейс объекта

Описание интерфейса (entity declaration) включает в себя объявление имени объекта и объявление входных и выходных портов объекта. Ниже приведено описание интерфейса из примера (листинг 1.1):

```
entity eq1 is                                   -- декларация имени объекта
    Port ( i0 , i1 : in    STD_LOGIC;       -- декларация входных портов
           eq : out    STD_LOGIC);              -- декларация выходного порта
end eq1;
```

В данном примере в первой строке объявляется имя объекта eq1. Структура port описывает входные и выходные порты в следующем формате:

имя_порта1, имя_порта2, . . . : режим_сигнала тип_сигнала ;

Для входных портов указывается входной режим сигнала (in), для выходных портов указывается выходной режим сигнала (out). Также предусмотрен режим inout для двунаправленных портов.

Тип данных и операторы

VHDL – язык со строгой типизацией. Это значит, что данные определенного типа могут принимать значения, соответствующие только данному типу (например, логический тип данных BOOLEAN может принимать только значения *ИСТИНА* (*TRUE*) и *ЛОЖНО* (*FALSE*) и никакие другие) и с данными определенного типа могут выполняться операции, соответствующие только данному типу (например, для данных типа BOOLEAN используются только логические операторы). Несмотря на то, что в языке VHDL применяется множество типов данных, в данном учебном пособии используются далеко не все – в основном применяется тип STD_LOGIC и его варианты. Это связано с тем, что основной упор в данном учебном пособии сделан на синтезируемые конструкции языка VHDL, т.е. такие конструкции, на основе которых может быть синтезирована реально работающая схема.

Тип данных STD_LOGIC определен в библиотеке IEEE.STD_LOGIC_1164 и содержит девять возможных значений. Три значения: ‘0’, ’1’ и ’Z’, соответствующие логическому нулю, логической единице и состоянию высокого омического сопротивления (Z-состоянию), могут быть синтезированы. Два других значения ‘U’ и ’X’, которые означают “uninitialize” (“неинициализированный”) и “unknown” (“неизвестный”), могут встретиться при симуляции проекта (например, когда сигналы ‘0’ и ‘1’ соединяются в одной точке). Оставшиеся четыре значения ‘-’, ’H’, ’L’ и ’W’ не используются в данном учебном пособии.

Цифровой сигнал зачастую представляет собой двоичной слово, т.е. набор битов. В данном случае, такой сигнал удобно представлять типом данных STD_LOGIC_VECTOR. Например, 8-ми битный входной порт данных можно объявить следующим образом:

```
a: in    STD_LOGIC_VECTOR (7 downto 0);
```

Запись “(7 downto 0)” означает, что в формате числа старший значащий бит (MSB) располагается крайним слева, а младший значащий бит (LSB) – крайним справа (такая форма записи является общепринятой в электронике, поэтому в данном учебном пособии используется именно этот формат). Для выделения части слова, например старших четырех бит в указанном примере, можно использовать запись “a(7 downto 4)”; для выделения отдельного бита из слова, например нулевого бита в указанном примере, применяется запись “a(0)”.

Для данных типа STD_LOGIC и STD_LOGIC_VECTOR определен ряд логических операторов, таких как: *НЕ* (*NOT*), *И* (*AND*), *ИЛИ* (*OR*), *Исключающее ИЛИ* (*XOR*). Для цифровых слов типа STD_LOGIC_VECTOR указанные логические операции выполняются побитно. В языке VHDL указанные логические операторы обладают одинаковым приоритетом, поэтому для определения порядка выполнения логического выражения необходимо применять скобки, например:

```
(a and b) or (c and d)
```

Структура объекта

Внутренняя структура объекта (architecture body) описывает работу схемы (листинг 3.1):

```
architecture sop_arch of eq1 is
    signal p0 , p1 : STD_LOGIC;
begin
    eq <= p0 or p1 ;                    -- сумма минтермов
    p0 <= (not i0) and (not i1) ;       -- определение минтерма
    p1 <= i0 and i1 ;                   -- определение минтерма
end sop_arch;
```

VHDL позволяет множеству различных структур объекта (architecture body) привязываться к интерфейсу объекта (entity), поэтому структура объекта должна иметь уникальное имя для своей идентификации, например “sop_arch” в примере выше (“sum_of_products architecture”).

Структура объекта (architecture body) может включать в себя секцию для объявления внутренних переменных (сигналов) объекта, вспомогательных констант и т.д. В приведенном примере, дополнительно объявляются две внутренние переменные:

```
signal p0 , p1 : STD_LOGIC;
```

Основная часть описания структуры объекта (architecture body) помещена между ключевыми словами begin и end. В приведенном примере структура объекта описывается тремя выражениями (statement):

```
eq <= p0 or p1 ;
p0 <= (not i0) and (not i1) ;
```

```
p1 <= i0 and i1 ;
```

При этом, в отличие от стандартных языков программирования (например *C*), где выражения (строки программы) выполняются последовательно, в языках описания аппаратуры HDL приведенные выражения (statement) описывают части реализуемой схемы и потому выполняются параллельно (concurrent statements). Переменная слева, при этом, рассматривается как выходной сигнал схемы, выражение справа определяет функцию работы схемы и соответствующие входные сигналы. В качестве примера рассмотрим следующее выражение:

```
eq <= p0 or p1 ;
```

Данное выражение описывает схему, реализующую операцию ИЛИ между входными сигналами *p0* и *p1*. При изменении значений сигналов *p0* и *p1* выражение пересчитывается. Результат расчета присваивается выходному сигналу *eq* через определенную временную задержку, заданную по умолчанию.

Графическое представление объекта представлено на рисунке 3.1. Три части схемы соответствуют трем выражениям (concurrent statements), описанным выше (порядок следования выражений в коде программы не имеет значения, поскольку данные выражения выполняются параллельно и могут следовать в любом порядке). Соединения между частями схемы однозначно определяются именами портов ввода/вывода и внутренних переменных.

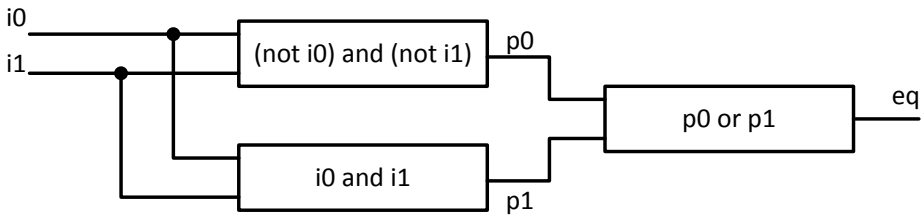


Рисунок 3.1 – Графическое представление реализации одноразрядной схемы сравнения

Двухразрядная схема сравнения

Увеличим разрядность схемы сравнения до двух. Пусть имеется входные 2-х разрядные сигналы *a* и *b* и выходной сигнал *aeqb*. Выходной сигнал *aeqb* принимает активное значение при одновременном равенстве и старших и младших разрядов *a* и *b*. Как и в случае одноразрядной схемы, описание реализуем в дизъюнктивной нормальной форме:

$$aeqb = \overline{a}_1 \cdot \overline{b}_1 \cdot \overline{a}_0 \cdot \overline{b}_0 + \overline{a}_1 \cdot \overline{b}_1 \cdot a_0 \cdot b_0 + a_1 \cdot b_1 \cdot \overline{a}_0 \cdot \overline{b}_0 + a_1 \cdot b_1 \cdot a_0 \cdot b_0 ,$$

где a_1, b_1 – старшие разряды сигналов *a* и *b* соответственно; a_0, b_0 – младшие разряды сигналов *a* и *b* соответственно;

Код реализации двухразрядной схемы сравнения приведен в листинге 3.2

Листинг 3.2 – Реализация двухразрядной схемы сравнения на вентильном уровне

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity eq2 is
      Port ( a, b : in    STD_LOGIC_VECTOR (1 downto 0);
                  aeqb : out    STD_LOGIC);
end eq2;

architecture sop_arch of eq2 is
         signal p0, p1, p2, p3 : STD_LOGIC;
begin
          -- сумма минтермов
         aeqb <= p0 or p1 or p2 or p3 ;
          -- определение минтермов
         p0 <= ((not a(1)) and (not b(1))) and ((not a(0)) and (not b(0))) ;
         p1 <= ((not a(1)) and (not b(1))) and ((a(0)) and (b(0))) ;
         p2 <= ((a(1)) and (b(1))) and ((not a(0)) and (not b(0))) ;
         p3 <= ((a(1)) and (b(1))) and ((a(0)) and (b(0))) ;
end sop_arch;
```

Входные порты *a* и *b* в данном случае объявляются как STD_LOGIC_VECTOR. Реализация структуры объекта аналогична реализации одноразрядной схемы – отличие только в количестве минтермов.

3.2 Структурное описание схем

В качестве альтернативы для реализации двухразрядной схемы сравнения (листинг 3.2) применим описанный выше структурный подход – в качестве унифицированного блока (instance) для построения двухразрядной схемы сравнения используется готовая одноразрядная схема (листинг 3.1). Структура двухразрядной схемы сравнения, реализованная указанным способом, представлена на рисунке 3.2. В приведенной структуре используется две одноразрядные схемы сравнения eq_bit0_unit и eq_bit1_unit, осуществляющие сравнение младших и старших бит входных сигналов соответственно и элемент *И*, на вход которого подаются сигналы с выходов указанных схем. Таким образом, выходной сигнал *aeqb* принимает активное значение при одновременном равенстве и старших и младших разрядов *a* и *b*.

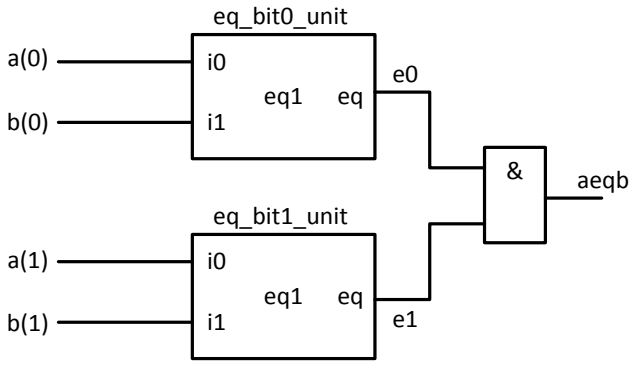


Рисунок 3.2 – Реализация двухразрядной схемы сравнения на основе одноразрядных блоков

Соответствующий код реализации двухразрядной схемы сравнения приведен в листинге 3.3 (Описание интерфейса (entity declaration) аналогично коду в листинге 3.2, поэтому в листинге 3.3 не приводится).

Листинг 3.3 – Описание структуры двухразрядной схемы сравнения

```
architecture struc_arch of eq2 is
        signal e0, e1 : STD_LOGIC;
begin
         -- вызов (instantiate) блока eq1 для сравнения младших битов
        eq_bit0_unit: entity work.eq1 (sop_arch)
                port map (i0 => a(0), i1 => b(0), eq => e0) ;
        -- вызов (instantiate) блока eq1 для сравнения старших битов
        eq_bit1_unit: entity work.eq1 (sop_arch)
                port map (i0 => a(1), i1 => b(1), eq => e1) ;
        aeqb <= e0 and e1 ;
         -- a и b равны, если равны одновременно и младшие и старшие биты
        aeqb <= e0 and e1 ;
end struc_arch;
```

Приведенный код (листинг 3.3) включает две процедуры вызова блока, реализующего одноразрядную схему сравнения (листинг 1.1). Рассмотрим подробно синтаксис процедуры вызова (instantiation statement) на примере вызова блока eq_bit0_unit:

```
eq_bit0_unit: entity work.eq1 (sop_arch)

        port map (i0 => a(0), i1 => b(0), eq => e0) ;
```

Процедура вызова начинается с объявления уникального имени вызываемого блока, в данном случае – **eq_bit0_unit**. Затем указывается название библиотеки (library name), в которой вызываемый блок находится – по умолчанию вызываемые блоки располагаются в рабочей библиотеке **work** создаваемого проекта. Идентификаторы **eq1** и **sop_arch** – имена интерфейса и архитектуры вызываемого блока соответственно (листинг 1.1). Вторая часть процедуры вызова заключается в привязке портов ввода вывода (port mapping) вызываемого блока к структуре проекта (рисунок 1.2). Таким образом, процедура вызова (instantiation statement) реализует схему, заключенную в «черный ящик», чья функция определяется в отдельном блоке (instance).

3.3 Задание на лабораторную работу – разработать схему «больше чем»

Схема «больше чем» сравнивает между собой 2 входа ***a*** и ***b*** и активизирует выход при условии, если ***a*** больше ***b***. Требуется разработать 4-х разрядную схему «больше чем» на вентильном уровне. Для этого:

1. Разработать таблицу истинности 2-х разрядной схемы «больше чем» и записать логику ее работы в совершенной дизъюнктивной нормальной форме. На основании полученного выражения, разработать VHDL-код, используя только логические операторы (вентили).
2. Разработать тестовый модуль (testbench) для моделирования работы 2-х разрядной схемы.
3. Используя 4 ползунковых переключателя для организации входа схемы и 1 светодиод для организации выхода, синтезировать схему и загрузить конфигурационный файл в отладочную плату. Протестировать схему.
4. Используя разработанную 2-х разрядную схему «больше чем», 2-х разрядную схему сравнения и минимальное количество дополнительной логики спроектировать 4-х разрядную схему «больше чем». Сначала нарисовать структурную схему, а затем, согласно схеме, разработать соответствующий VHDL-код.
5. Разработать тестовый модуль (testbench) для моделирования работы 4-х разрядной схемы.
6. Используя 8 ползунковых переключателя для организации входа схемы и 1 светодиод для организации выхода, синтезировать схему и загрузить конфигурационный файл в отладочную плату. Протестировать схему.

4 Проектирование комбинационных схем на регистровом уровне

Введение

В первой работе рассматривались простые схемы, описанные на вентильном уровне абстракции (gate level), с применением базовых логических вентилей, реализующих основные логические операции. В данной работе рассматривается проектирование схем средней степени интеграции, таких как сумматоры, дешифраторы, мультиплексоры и т.д. Описание подобных схем выполняется уже не на вентильном уровне абстракции (gate level), а на так называемом регистровом или функциональном уровне (RT-level). На данном уровне абстракции описывается поведение схемы (ее функция). В начале работы выполнен обзор операторов и конструкций, применяемых для описания схем на регистровом уровне.

В дополнение к логическим операторам, рассмотренным в первой главе, для синтеза комбинационных схем применяются операторы сравнения (relational operators) и некоторые арифметические операторы (arithmetic operators). В данной главе рассматриваются данные операторы и разнообразные VHDL-конструкции на их основе. В таблицах 4.1 и 4.2 приведены основные операторы VHDL, применяемые для синтеза схем, и типы данных, с которыми данные операторы работают.

Таблица 4.2 – Операторы и соответствующие типы данных стандартной библиотеки VHDL-93 IEEE.STD_LOGIC_1164 package

оператор	описание	тип данных операндов	тип данных результата
a ** b	возведение в степень	Integer	integer
a * b	умножение	тип integer используется для констант и границ массивов. **Не синтезируются!**	
a / b	деление		
a + b	суммирование		
a - b	вычитание		
a & b	конкатенация	1-D array, element	1-D array
a = b	равенство	любой	boolean
a /= b	не равенство		
a < b	меньше чем	scalar или 1-D array	boolean
a <=b	меньше чем или равно		
a > b	больше чем		
a >=b	больше чем или равно		
not a	отрицание	boolean, std_logic, std_logic_vector	такой же как у операндов
a and b	И		
a or b	ИЛИ		
a xor b	исключающее ИЛИ		

Таблица 4.2 – Операторы и соответствующие типы данных библиотеки IEEE.NUMERIC_STD package

оператор	описание	тип данных операндов	тип данных результата
a * b	арифметические операции	unsigned, natural signed, integer	unsigned signed
a + b			
a - b			
a = b	операции сравнения	unsigned, natural signed, integer	boolean boolean
a /= b			
a < b			
a <=b			
a > b			
a >=b			

Операторы сравнения

В стандарте VHDL используется шесть операторов сравнения: **=** (равенство), **/=** (не равенство), **<** (меньше чем), **<=** (меньше чем или равно), **>** (больше чем), **>=** (больше чем или равно). Данные операторы выполняют операции сравнения между операндами одного типа и возвращают результат логического типа BOOLEAN. В данном учебном пособии тип данных BOOLEAN напрямую не используется, однако операции сравнения применяются в различных условных конструкциях VHLD, которые определяют структуру схемы при ее синтезировании. Операторы сравнения синтезируются в виде компараторов.

Арифметические операторы

В стандарте VHDL арифметические операторы определены для типов данных INTEGER (целочисленный) и NATURAL (натуральные числа). Данные типы используются для констант и границ массивов, но не для синтеза.

Библиотека IEEE.NUMERIC_STD package

Для применение арифметических операторов при синтезе схем используется дополнительная библиотека IEEE.NUMERIC_STD package (таблица 4.2). Библиотека IEEE.NUMERIC_STD package добавляет два дополнительных типа данных: unsigned (беззнаковый) и signed (знаковый) и определяет операторы сравнения и арифметические операторы для этих типов данных. Типы данных unsigned и signed определены как массив элементов типа STD_LOGIC. Данный массив интерпретируется, как двоичное представление знакового или беззнакового целого числа. Для использования операторов из библиотеки IEEE.NUMERIC_STD package (таблица 4.2) ее нужно объявить вначале проекта:

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.NUMERIC_STD.ALL;          -- объявление библиотеки numeric_std
```

Конвертация типов данных

Поскольку VHDL является языком со строгой типизацией, STD_LOGIC_VECTOR, unsigned и signed трактуются как различные типы данных, несмотря на то, что все они представляют собой массив элементов типа STD_LOGIC. Для конвертации данных из одного типа в другой используются функции конвертации (таблица 4.3). Обратите внимание, что данные типа STD_LOGIC_VECTOR представляют собой набор битов и не интерпретируются как число, поэтому данные типа STD_LOGIC_VECTOR не могут быть конвертированы в целочисленный тип INTEGER напрямую и наоборот.

Таблица 4.3 – Конвертация типов данных

Тип данных a	Конвертировать в …	Функция конвертации
unsigned, signed	std_logic_vector	std_logic_vector(a)
signed, std_logic_vector	unsigned	unsigned(a)
unsigned, std_logic_vector	signed	signed(a)
unsigned, signed	integer	to_integer(a)
natural	unsigned	to_unsigned(a, size)
integer	signed	to_signed(a, size)

В листинге 4.1 приведены примеры правильной и неправильной конвертации типов данных. Пусть в проекте объявлены следующие данные.

Листинг 4.1 – Описание структуры двухразрядной схемы сравнения

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
. . .
signal s1, s2, s3, s4, s5, s6 : STD_LOGIC_VECTOR(3 downto 0);
signal u1, u2, u3, u4, u5, u6 : UNSIGNED(3 downto 0);
. . .
```

Для начала рассмотрим следующие выражения:

```
u1 <= s1; -- Ошибка типа (type mismatch)

u2 <= 5;  -- Ошибка типа (type mismatch)

s2 <= u3; -- Ошибка типа (type mismatch)

s3 <= 5;  -- Ошибка типа (type mismatch)
```

Все приведенные выражения ошибочны. Необходимо данные, находящиеся в правой части выражений привести к типу данных, находящихся в левой части:

```
u1 <= unsigned(s1);                                  -- Выражение верно

u2 <= to_unsigned(5, 4);                             -- Выражение верно

s2 <= std_logic_vector(u3);                          -- Выражение верно

s3 <= std_logic_vector(to_unsigned(5, 4));           -- Выражение верно
```

Обратите внимание, что для последнего выражения использовались две функции конвертации.

Теперь рассмотрим выражения, использующие арифметические операторы. Следующие выражения верны поскольку оператор «+» определен для типов данных unsigned и natural в библиотеке IEEE.NUMERIC_STD package:

```
u4 <= u1 + u2;          -- Выражение верно. Оба операнда типа unsigned

u4 <= u2 + 1;           -- Выражение верно. Операнды типа unsigned и natural
```

С другой стороны, следующие выражения являются неверными, поскольку для данных типа STD_LOGIC_VECTOR арифметические операции не определены:

```
s5 <= s1 + s2;      -- Выражение не верно. Оператор «+» не определен для
                                используемых типов данных

s5 <= s2 + 1;       -- Выражение не верно. Оператор «+» не определен для
                                используемых типов данных
```

Для решения проблемы необходимо выполнить конвертацию операндов к типу unsigned (или signed), выполнить их сложение, а затем результат конвертировать обратно к типу STD_LOGIC_VECTOR:

```
S5 <= std_logic_vector(unsigned(s1) + unsigned(s2));        -- Выражение верно

S5 <= std_logic_vector(unsigned(s2) + 1);                   -- Выражение верно
```

Оператор конкатенации

Оператор конкатенации «&» соединяет (склеивает) сегменты элементов в одну последовательность (массив). Листинг 4.2 иллюстрирует использование конкатенации.

Листинг 4.2 – Описание конкатенации

```
signal a1 : STD_LOGIC;
signal a4 : STD_LOGIC_VECTOR(3 downto 0);
signal b8, c8, d8 : STD_LOGIC_VECTOR(7 downto 0);
. . .
b8 <= a4 & a4;                  -- Соединяет два 4-х разрядных слова a4 в одно
                                   8-ми разрядное b8
```

```
c8 <= a1 & a1 & a4 & “00”;                    -- Соединяет два бита a1, 4-х разрядное слово
                                                 a4 и 2-х разрядное слово «00» в одно 8-ми
                                                 разрядное слово c8

d8 <= b8(3 downto 0) & c8(3 downto 0);-- Соединяет 4 младших разряда 8-ми разрядного
                                                 слова b8 и 4 младших разряда 8-ми разрядного
                                                 слова c8 в одно 8-ми разрядное слово d8
```

Оператор конкатенации «&» при синтезе схемы выполняется путем соединения входных и выходных сигналов, таким образом для его реализации требуются только «проводники».

Основное применение оператора конкатенации «&» - операции сдвига (shifting operations). В примере ниже оператор «&» применяется для реализации сдвига данных на фиксированное количество бит:

```
signal a : STD_LOGIC_VECTOR(7 downto 0);
signal rot, shl, sha : STD_LOGIC_VECTOR(7 downto 0);
signal b8, c8, d8 : STD_LOGIC_VECTOR(7 downto 0);
. . .
rot <= a(2 downto 0) & a(8 downto 3);          -- Сдвиг вправо на 3 бита

shl <= “000” & a(8 downto 3);                  -- Сдвиг вправо на 3 бита с заполнением
                                                  старших бит нулями (логический сдвиг)

sha <= a(8) & a(8) & a(8) & a(8 downto 3); -- Сдвиг вправо на 3 бита с расширением
                                                  числа по знаку (арифметический сдвиг)
```

Z-состояние

Тип данных STD_LOGIC помимо значений ‘0’ и ’1’ содержит Z-состояние (высокоимпедансное состояние), характеризующее разомкнутую цепь. Z-состояние синтезируется исключительно с помощью трехстабильных буферов (tri-state buffer). Условное обозначение и таблица истинности трехстабильного буфера представлены на рисунке 4.1 и в таблице 4.4 соответственно.

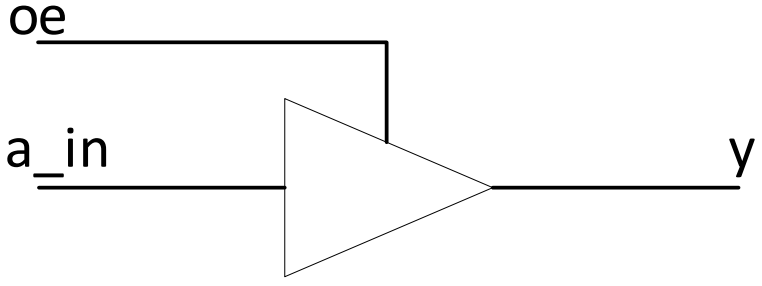


Рисунок 4.1 – Условное обозначение трехстабильного буфера

Таблица 4.4 – Таблица истинности трехстабильного буфера

oe	y
0	Z
1	a_in

Работа трехстабильного буфера контролируется входом разрешения oe (output enable) когда **oe** = ‘1’, входной сигнал передается на выход буфера, когда же **oe** = ‘0’ выходная цепь буфера разомкнута, т.е. буфер находится в Z-состоянии. Код, описывающий трехстабильный буфер выглядит следующим образом:

```
y   <= a_in when oe = ’1’ else ’Z’;
```

Приведенное выше выражение называется «присваиванием по условия» и будет рассмотрено далее. Основное применение трехстабильного буфера – реализация двунаправленного порта ввода/вывода (bidirectional port) для более рационального использования контактов микросхем (I/O pins). Пример двунаправленного порта показан на рисунке 4.2.

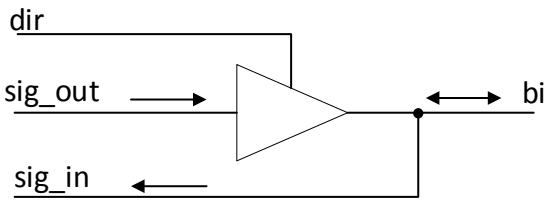


Рисунок 4.2 – Двунаправленный порт

Сигнал dir управляет направлением передачи данных. Когда dir = ‘0’ трехстабильный буфер находится в Z-состоянии и сигнал sig_out заблокирован – вывод bi используется как входной порт (sig_in). Когда dir = ‘1’ вывод bi используется как выходной порт и сигнал sig_out выдается во внешнюю цепь. VHDL-код, описывающий двунаправленный порт (рисунок 4.2) приведен в листинге 4.3.

Листинг 4.3 – Описание двунаправленного порта

```
entity bi_demo is
      Port (bi : inout    STD_LOGIC;
                . . .)
begin
          . . .
          bi <= sig_out when dir=’l’ else ’Z’;
          sig_in <= bi;
```

. . .

Обратите внимание что порт **bi** должен объявляться как inout, т.е. двунаправленный.

2.1.4 Выводы

VHLD – язык со строгой типизацией данных, поэтому при проектировании схем начинающими разработчиками часто можно видеть ошибку «Type mismatch». Ниже резюмируются основные правила, позволяющие избежать подобных ошибок. Поскольку данное учебное пособие нацелено прежде всего на синтезируемые конструкции VHLD, правила приводятся для синтезируемых типов данных и операторов:

- При объявлении входных и выходных портов (entity declaration) и внутренних сигналов (signals), не задействованных в арифметических операциях, используйте типы данных STD_LOGIC и STD_LOGIC_VECTOR.
- Используйте значение ‘Z’ только для синтезирования трехстабильных буферов.
- При объявлении внутренних сигналов (signals), участвующих в арифметических операциях, используйте библиотеку IEEE.NUMERIC_STD package и загружаемые с ней типы данных unsigned (беззнаковый) и signed (знаковый).
- Используйте функции конвертации (таблица 2.3) для преобразования данных типа STD_LOGIC_VECTOR в unsigned или signed и наоборот.
- Используйте встроенный тип данных integer и арифметические операции для задания констант и граничных значений массивов, но не для синтеза.
- Используйте результаты операторов сравнения, имеющих тип данных BOOLEAN в различных условных конструкциях VHLD (раздел 2.2), но не для синтеза.

4.1 Использование параллельных операторов присваивания

«Присваивание по условию» (conditional signal assignment) и «присваивание по выбору» (selected signal assignment) являются параллельными конструкциями (concurrent statements) в VHLD. Функционально данные конструкции напоминают операторы if и case в традиционных языках программирования, но в отличие от традиционных языков конструкции в VHLD выполняются параллельно, а не последовательно. Данные конструкции определяют структуру схемы при ее синтезировании.

Присваивание по условию (conditional signal assignment)

Синтаксис «присваивания по условию» приведен ниже:

```
signal_name <=   value_expr_1 when boolean_expr_1 else

                 value_expr_2 when boolean-expr-2 else

                 . . .
```

value_expr_n;

Булевы выражения (**boolean_expr**) поочередно проверяются пока одно из них не оказывается истинным и соответствующее выражение (**value_expr**) присваивается сигналу **signal_name**. В случае если все булевы выражения ложны сигналу присваивается значение **value_expr_n**.

Структура «присваивание по условию» синтезируется в каскадную приоритетную схему на базе мультиплексоров 2 в 1. Условное обозначение и таблица истинности мультиплексора 2 в 1 показаны на рисунке 4.3 и таблице 4.5 соответственно.

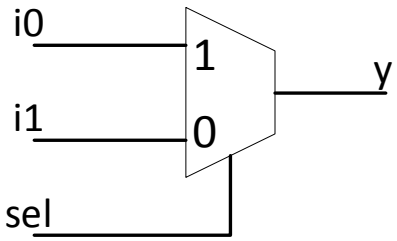


Рисунок 4.3 – Схема мультиплексора 2 в 1

Таблица 4.5 – Таблица истинности мультиплексора 2 в 1

sel	y
0 (false)	i0
1 (true)	i1

Рассмотрим пример синтеза следующего кода:

```
r <=    a + b + c when m = n else
        a - b     when m > n else
        c +1 ;
```

Каскадная приоритетная схема, в данном случае, реализуется с помощью соединения двух мультиплексоров 2 в 1 (рисунок 4.4).

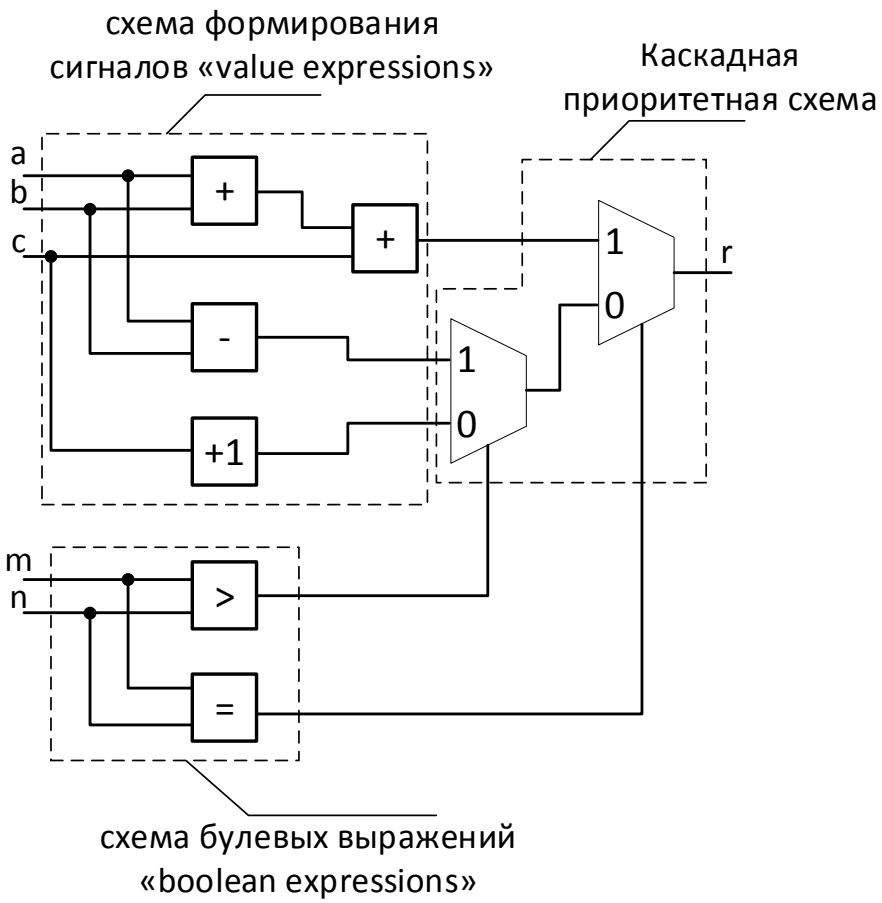


Рисунок 4.4 – Синтезированная схема, реализующая конструкцию «присваивание по условию»

Схема работает следующим образом: если истинно первое булево выражение (m = n), то сумма сигналов (a + b + c) передается на выход схемы y; если же истинно второе булево

выражение (m > n), то на выход y передается разность сигналов (a - b); во всех остальных случаях (m < n) на выход схемы передается сигнал (c + 1).

Обратите внимание, что схема формирования сигналов (value expressions circuit) и схема булевых выражений (boolean expressions circuit) выполняются параллельно (рисунок 4.4). Выходные сигналы схемы булевых выражений используются как сигналы выбора в мультиплексорах, а выходные сигналы схемы формирования являются информационными сигналами, из которых выбирается необходимый сигнал и подается на выход схемы r. Количество ступеней (мультиплексоров) в каскадной приоритетной схеме (рисунок 4.4) соответствует количеству when-else условий в конструкции. Большое количество when-else условий приводит к синтезированию длинной цепи каскадов и, как следствие, к значительной временной задержке (propagation delay) работы схемы.

Ниже приведены два простых примера использования конструкции «присваивание по условию» (conditional signal assignment). Первым примером является приоритетный 4-х разрядный шифратор (priority encoder). Данный шифратор имеет четыре входных запроса (requests): r(4), r(3), r(2) и r(1), которые объединены в 4-х разрядный вход r, причем старший бит r(4) имеет наивысший приоритет. Выходной сигнал pcode представляет собой двоичный код входного запроса с наивысшим приоритетом. Таблица истинности приоритетного шифратора приведена в таблице 4.6. Синтезируемый HDL код приведен в листинге 4.4.

Таблица 4.6 – Таблица истинности 4-х разрядного приоритетного шифратора

Input r	Output pcode
1×××	100
01××	011
001×	010
0001	001
0000	000

Листинг 4.4 – Синтез приоритетного шифратора с применение конструкции «присваивание по условию» (conditional signal assignment)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity prio_encoder is
    Port (
        r : in   STD_LOGIC_VECTOR (4 downto 1);
        pcode : out   STD_LOGIC_VECTOR (2 downto 0)
        );
end prio_encoder;

architecture cond_arch of prio_encoder is
begin
    pcode   <=   "100" when (r(4) = 1) else
                 "011" when (r(3) = 1) else
                 "010" when (r(2) = 1) else
                 "001" when (r(1) = 1) else
```

```
                        “000” ;
end cond_arch;
```

В приведенном коде (листинг 4.1) сначала проверяется состояние запроса r(4) и, в случае если он активен, на выход передается код “100”. Если запрос r(4) не активен проверяется состояние следующего по приоритету запроса, т.е. r(3) и т.д. пока не будут проверены остальные запросы.

Вторым примером использования конструкции «присваивание по условию» (conditional signal assignment) для синтеза схем является двоичный дешифратор. Двоичный дешифратор n-в-2^n преобразует двоичный n-разрядный код в унитарный 2^n-разрядный (код, в котором активным является только один бит). Таблица истинности двоичного дешифратора 2-в-4 приведена в таблице 4.7. Схема дешифратора дополнительно имеет вход разрешения работы **en** (enable). Синтезируемый HDL код дешифратора приведен в листинге 4.5.

Таблица 4.7 – Таблица истинности двоичного дешифратора 2-в-4

Input			Output
en			y
a(1)		a(0)	
0	×	×	0000
1	0	0	0001
1	0	1	0010
1	1	0	0100
1	1	1	1000

Листинг 4.5 – Синтез двоичного дешифратора с применение конструкции «присваивание по условию» (conditional signal assignment)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity decoder_2_4 is
    Port (
            a : in    STD_LOGIC_VECTOR (1 downto 0);
            en : in    STD_LOGIC;
            y : out    STD_LOGIC_VECTOR (3 downto 0)
            );
end prio_encoder;

architecture cond_arch of decoder_2_4 is
begin
        y    <=    “0000” when (en = ‘0’) else
                   “0001” when (a = “00”) else
                   “0010” when (a = “01”) else
                   “0100” when (a = “10”) else
                   “1000” ;          -- a = “11”
```

```
end cond_arch;
```

В приведенном коде (листинг 4.5) изначально проверяется состояние сигнала разрешения **en**, если сигнал активный, т.е. **en** = ‘1’, то последовательно проверяются 4 входные комбинации сигнала **a**.

Присваивание по выбору (selected signal assignment)

Синтаксис «присваивания по выбору» приведен ниже:

```
with sel select
        sig <=      value_expr_1 when choice_1 ,

                    value_expr_2 when choice_2 ,

                    . . .

                    value_expr_n when others;
```

Функционально «присваивание по выбору» напоминает оператор case в традиционных языках программирования. Данная конструкция присваивает необходимое значение (**value_expression**) выходному сигналу **sig** в зависимости от значения сигнала выбора **sel**. Каждый конкретный выбор (**choice_i**) представляет собой конкретное значение или набор значений **sel**. Каждый выбор должен включать в себя любое значение **sel** не чаще одного раза, но при этом все вероятные значения **sel** должны присутствовать в структуре. Для отражения в структуре неиспользуемых вариантов используется служебное слово **others**, которое используется в обязательном порядке и включает в себя также не синтезируемые значения (‘X’, ’U’ и т.д.).

Структура «Присваивание по выбору» синтезируется в схему на базе мультиплексора. Рассмотрим пример синтеза в листинге 4.6.

Листинг 4.6 – Синтез структуры присваивания по выбору

```
signal sel : STD_LOGIC_VECTOR (1 downto 0);

. . . .

r <=    a + b + c when “00” ,
```

```
a - b          when “01” ,
c +1           when others;
```

В данном случае сигнал **sel** может принимать 4 значения (“00”, ”01”, ”10”, “11”). Поэтому схема будет синтезирована в виде мультиплексора 4 в 1 с сигналом выбора **sel** (рисунок 4.5). Условное обозначение и таблица истинности мультиплексора 4 в 1 показаны на рисунке 4.6 и таблице 4.8 соответственно.

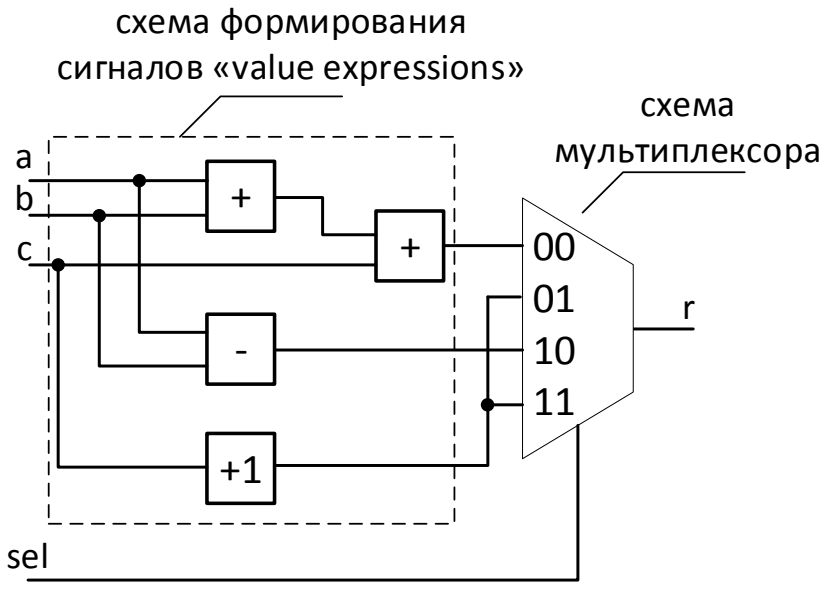


Рисунок 4.5 – Синтезированная схема, реализующая конструкцию «присваивание по выбору»

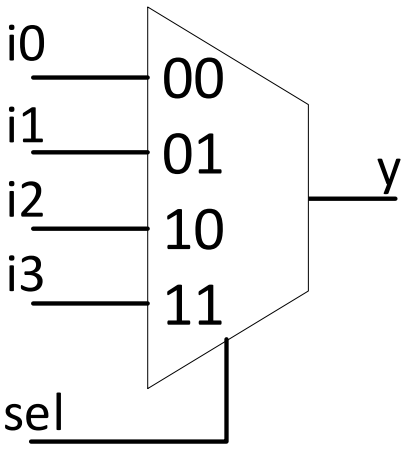


Рисунок 4.6 – Схема мультиплексора 4 в 1

Таблица 4.8 – Таблица истинности мультиплексора 4 в 1

sel	y
00	i0
01	i1
10	i2
11	i3

В результате схема (рисунок 4.5) работает следующим образом: сумма сигналов a+b+c передаётся на выход r при значении сигнала sel “00”; разность сигналов a-b передаётся на выход r при значении сигнала sel “10”; сигнал c+1 передаётся на выход схемы r при значении sel “01”

4.2 Использование последовательных структур

Последовательные структуры в VHDL используются для описания поведения системы, которое происходит последовательно, поэтапно, в определенном порядке. Синтезаторы HDL распознают определенные идиомы и превращают их в конкретные последовательные схемы. Код, написанный в ином стиле, может правильно симулироваться, но в синтезированной схеме могут оказаться как грубые, так и труднораспознаваемые ошибки.

Описание process

Для облегчения моделирования системы VHDL содержит ряд последовательных, операторов, которые выполняются последовательно. Они включают большое разнообразие конструкций, но многие из них не имеют четкие аппаратные аналоги. Для синтеза последовательной структуры мы ограничиваем использование процесса и создаем список чувствительности – sensitivity list. В листинге 4.7 представлен упрощённый синтаксис процесса со списком чувствительности.

Листинг 4.7 – Синтез структуры присваивания по выбору

```
process(sensitivity-list)

begin

        sequential statement;
        sequential statement;

... end process;
```

Здесь список входных сигналов (sensitivity list) - это список сигналов, которые будут использоваться в процессе. Внутри блока process описывается последовательность операторов, которые будут выполняться при изменении одного или нескольких входных сигналов.

Таким образом, process в VHDL используется для описания поведения системы в ответ на изменение входных сигналов. Он является одним из основных элементов языка VHDL и используется для описания различных видов поведения, от простых до сложных.

Описание if statement

Оператор IF в VHDL используется для проверки условий и выполнения определенных действий в зависимости от результата проверки. Он может использоваться как самостоятельный оператор, так и в составе последовательных структур. В листингаз 4.8, 4.9, 4.10 приведены примеры использования оператора IF в последовательных структурах VHDL.

Листинг 4.8 – Оператор if в процессе

```
process (a, b)
    begin
      if a = '1' then
        c <= d;
      end if;
      if b = '1' then
        c <= e;
      end if;
 end process;
```

В этом примере значение сигнала c изменяется в зависимости от значений сигналов a и b. Если a = '1', то c присваивается значение d, а если b = '1', то c присваивается значение e. Оператор IF используется для определения условия, при котором будет происходить изменение сигнала c.

Листинг 4.9 – Оператор if внутри другой последовательной структуры

```
process (a, b)
    begin
      if a = '1' then
        if b = '1' then
          c <= d;
        else
          c <= e;
        end if;
```

```
        end if;

end process;
```

В этом примере значение сигнала c изменяется в зависимости от значений сигналов a и b. Если a = '1', то происходит проверка значения сигнала b. Если b = '1', то c присваивается значение d, а если b = '0', то c присваивается значение e.

Листинг 4.10 – Оператор if с несколькими условиями

```
process (a, b)
    begin
        if a = '1' and b = '1' then
            d <= c;
        elsif a = '1' and b = '0' then
            d <= not c;
        else
            d <= '0';
        end if;
end process;
```

В этом примере значение сигнала d изменяется в зависимости от значений сигналов a, b и c. Если a = '1' и b = '1', то d присваивается значение c. Если a = '1' и b = '0', то d присваивается значение not c. В противном случае d присваивается значение '0'.

Таким образом, оператор IF в VHDL широко используется внутри последовательных структур для проверки условий и выполнения определенных действий в зависимости от результата проверки. Он позволяет создавать более сложные логические конструкции и сделать описание поведения системы более гибким и удобным.

Описание CASE statement

Оператор case в VHDL используется для проверки условий и выполнения определенных действий в зависимости от результата проверки. Он может использоваться как самостоятельный оператор, так и в составе последовательных структур. В листингаз 4.11, 4.12 приведены примеры использования оператора case в последовательных структурах VHDL.

Листинг 4.11 – Упрощенный синтаксис оператора Case

```
process (a, b)

        begin
        case sel is
        when choice-1 =>
                    sequential statements;
        when choice-2 =>;
                     sequential statements;
        when others =>
                     sequential statements ;
        end case;
 end process;
```

Оператор case использует сигнал sel для выбора набора последовательных операторов для выполнения.

Как и в операторе присвоения выбранного сигнала, выбор (т. е. выбор-i) должен быть действительным. Значение или набор допустимых значений sel, и выбор должен быть взаимоисключающим, и все включительно. Обратите внимание, что остальные термины в конце охватывают неиспользуемые значения.

Оператор case и параллельный оператор присваивания выбранного сигнала несколько похожий. Два оператора эквивалентны, если каждая ветвь оператора case содержит только один оператор последовательного назначения сигнала.

Например, часть листинга 4.12 переписать в именном виде.

```
        with sel select
                r <= a + b + c when "00",
                             a - b when "10",
                             c + l when others;
        Можно переписать как:
        process(a,b,c,sel)
        begin
                case sel is
                        when "00" =>
                                        r <= a + b + c;
                        when "10" =>
                                  r <= a - b;
                        when others =>
                                 r <= c + 1;
        end case;
        end    process;
```

Листинг 4.12 – Оператор if с несколькими условиями

Как и в операторе присваивания выбранного сигнала, оператор case предполагает аналогичное мультиплексирование структуру в процессе синтеза.

4.3 Сравнение последовательных и параллельных структур

В предыдущих подразделах показано, что простые операторы if и case эквивалентны оператору условные и выбранные операторы присвоения сигналов. Однако оператор if или case позволяет любое количество и любой тип последовательных операторов в своих ветвях и, таким образом, более гибкий и универсальный. Дисциплинированное использование может сделать код более описательным и даже схема более эффективная.

Это можно проиллюстрировать двумя сегментами кода. Сначала рассмотрим схему на листинге 4.13, которая сортирует значения двух входных сигналов и направляет их на большой и малый выходы. Это может быть делается с помощью двух операторов условного присваивания сигнала.

Листинг 4.13 – Направление сигналов

```
large <= a when a > b else
                    b;
small <= b when a > b else
                    a;
```

Поскольку в коде есть два оператора отношения (т.е. два >), программное обеспечение синтеза может вывести

два компаратора больше чем. Та же самая функция может быть закодирована одним оператором if в листинге 4.14:

```
process (a, b)
begin
        if a > b then
            large <= a;
            small <= b;
```

```
            else
                large <= b;
                small <= a;
                  end if ;
    end ;
```

Листинг 4.14 – Направление сигналов с оператором if

Код состоит только из одного реляционного оператора. Далее в листинге 4.15, давайте рассмотрим схему, которая направляет максимальное значение трех входных сигналов на выход. Это можно четко описать вложенными двухуровневыми операторами if:

Листинг 4.15 – Направление сигналов с оператором if

```
    process (a,b,c)
    begin
            if (a > b) then
                if (a > c) then
                    max <= a;
                else
                    max <= c;
                end if ;
            else
                if (b > c) then
                    max <= b;
                else
                    max <= c;
                end if ;
            end if ;
    end process;
```

Мы так же можем преобразовать оператор if в одноуровневый оператор условного присваивания сигнала в листинге 4.16.

Листинг 4.16 – Направление сигналов с оператором case

```
max <= a when ((a > b) and (a > c)) else
                    c when (a > b) else
                    b when (b > c) else
                    c;
```

Поскольку вложение не допускается, код становится менее интуитивным. Если необходимо использовать параллельные операторы, лучшей альтернативой является описание схемы с тремя условными операторами присваивания сигнала в листинге 4.17.

Листинг 4.17 – Листинг 4.16 – Направление сигналов с тремя условными операторами

```
signal ac-max , bc-max: std-logic;
….
ac-max <= a when (a > c) else
                    c;
bc-max <= b when (b > c) else
                    c;
max <= ac-max when (a > b) else
                    bc-max ;
```

4.4 Задание на лабораторную работу – разработать схему четырехразрядного приоритетного шифратора

Схема четырехразрядного приоритетного шифратора построена с помощью структуры «Присваивание по условию».

1. Разработать схему четырехразрядного приоритетного шифратора с соблюдением таблицы истинности 4.9.

Таблица 4.9 – Таблица истинности 4-х разрядного приоритетного шифратора

Input r	Output pcode
1×××	100
01××	011
001×	010
0001	001
0000	000

2. Разработать тестовый модуль (VHDL Test Bench) и проверить работу схемы во встроенном симуляторе ISim.
3. Нарисовать синтезированную схему приоритетного шифратора.

5 Проектирование последовательностных схем

Последовательная схема - это схема с памятью, которая формирует внутреннее состояние схемы.

В отличие от комбинационной схемы, в которой выходной сигнал является функцией только входного сигнала, выходной сигнал последовательной схемы является функцией входного сигнала и внутреннего состояния. Методология синхронного проектирования является наиболее часто используемой практикой при проектировании последовательной схемы. В этой методологии все элементы хранилища управляются (т.е. синхронизируются) с помощью глобальных часов сигнал и данные отбираются и сохраняются на восходящем или нисходящем фронте тактового сигнала.

Это позволяет разработчикам отделять компоненты хранилища от схемы и значительно упрощает процесс разработки. Эта методология является наиболее важным принципом при разработке большой и сложной цифровой системы и лежит в основе большинства алгоритмов синтеза, верификации и тестирования. Все проекты, представленные в книге, следуют этой методологии. Одним из ключевых элементов последовательностной схемы является состояний, которые определяют текущее состояние схемы на основе полученных входных данных. В каждом состоянии определено, какие операции будут выполнены в следующем состоянии, а также какое состояние будет следующим, в зависимости от полученных входных данных. Другим важным аспектом последовательностных схем является использование триггеров, которые позволяют хранить и обрабатывать данные в различных состояниях. Триггеры могут использоваться для считывания данных, сохранения данных и обработки данных перед последующей передачей на выход.

Использование последовательностных схем в VHDL позволяет разработчикам создавать комплексные схемы, которые могут обрабатывать большие объемы данных с высокой точностью и эффективностью. Такие схемы часто используются в современных электронных устройствах, например, в автомобильной промышленности, коммуникационных системах и во многих других областях. Конечный результат является описанием поведения системы в синтаксисе, который может быть скомпилирован и выполнен на ПЛИС.

5.1 Описание синхронных систем

Самым основным компонентом запоминающего устройства в последовательной схеме является D триггер срабатывающего по положительному фронту, показана на рисунке 5.1. Значение сигнала d отсчитывается по восходящему фронту сигнала clk и сохраняется в FF

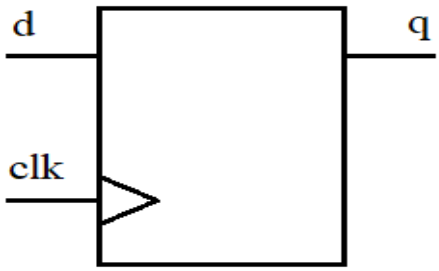


Рисунок 5.1 – D триггер с разрешающим сигналом

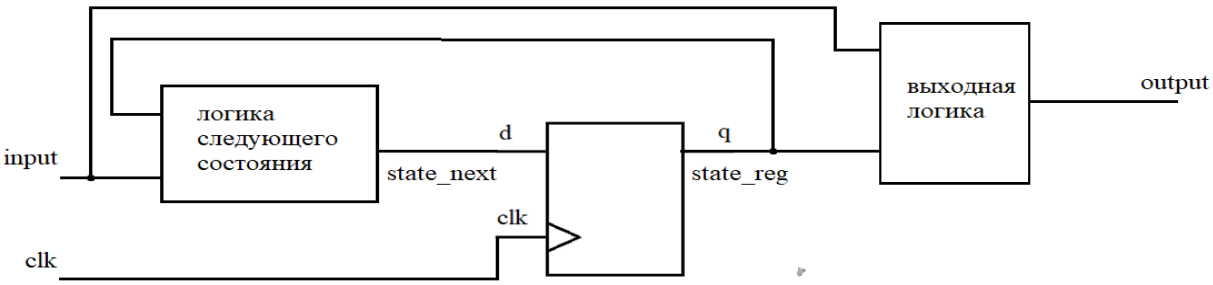


Рисунок 5.2 – Структурная схема синхронной системы

Структурная схема синхронной системы показана на рисунке 5.2. Она состоит из следующих частей:

- State reg: набор D триггеров, управляемых одним и тем же тактовым сигналом.
- Next-stale logie: комбинационная логика, которая использует выходные данные регистра для определения нового значения регистра.
- Output logic: комбинационная логика, которая генерирует выходной сигнал отключения.

Одним из наиболее сложных аспектов проектирования последовательной схемы является обеспечение того, чтобы синхронизация системы не нарушала ограничений по времени настройки и удержания.

В асинхронной системе компоненты хранилища сгруппированы вместе и обрабатываются как единый регистр, как показано на рисунке 5.2. Нам нужно выполнить анализ синхронизации только для одной памяти компонента. Синхронизация последовательной схемы характеризуется fmax - максимальной тактовой частотой, которая определяет, с какой скоростью может работать circait.

Обратная величина fmax задает Tcomb минимальный тактовый период, который может быть интерпретирован как интервал между двумя фронтами дискретизации тактового сигнала. Чтобы обеспечить правильную работу, следующее значение должно быть сгенерировано и стабилизировано в течение этого интервала. Как уже упоминалось, максимальная задержка

распространения логики следующего состояния равна Tcomb. Минимальный тактовый период может быть получен путем сложения задержек по формуле 5.1 распространения и ограничение по времени настройки замкнутого контура на рисунке 5.2:

$$Tclock = Tcq + Tcomb + Tsetup \quad (5.1)$$

Во время синтеза программное обеспечение Xilinx проанализирует синтезированную схему и покажет fnux в отчете. Мы также можем указать желаемую рабочую частоту в качестве ограничения синтеза, и программное обеспечение синтеза попытается получить схему, удовлетворяющую этому требованию (т.е. схему, fmax которой равен или больше чем желаемая рабочая частота), например, если мы используем генератор частотой 50 МГц (т.е. с периодом 20 нс) на плате прототипа в качестве источника тактовой частоты, fmax последовательной схемы должен превышать эту частоту (т.е. период должен быть меньше 20 нс).

5.2 D триггер

Разработка нашего кода осуществляется в соответствии с базовой блок-схемой на рисунке 5.2. Главное – отделить компонент памяти (например, регистр) от системы. Как только регистр изолирован, оставшаяся часть представляет собой чистую комбинационную схему, и схемы кодирования и анализа, рассмотренные в предыдущих главах, могут быть применены соответствующим образом. Хотя такой подход может привести к код временами немного более громоздкий, это помогает нам лучше визуализировать архитектуру схемы и избежать непреднамеренного использования памяти и незначительных ошибок.

На листинге 5.1 представлено описание работы D триггера без синхронного сброса с использованием структуры process с обязательным описанием признаков чувствительности.

Листинг 5.1 – D триггер без синхронного сброса

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity d_ff is
      Port (
               clk: in    STD_LOGIC;
               d: in    STD_LOGIC;
               q : out    STD_LOGIC;
end d_ff;

architecture arch of d_ff is
begin
          process(clk)
          begin
             if (clk'event and clk='l') then
```

```
                q <= d;
            end if ;
            end process;
end arch;
```

Восходящий фронт проверяется выражением clk'event и clk='1', которое представляет, что в сигнале clk произошло изменение (т.е. "событие"), и новым значением является 'l'. Если это условие истинно, значение d сохраняется в q, а если это условие ложно, q сохраняет свое предыдущее значение (т.е. запоминает значение, выбранное ранее).

Обратите внимание, что в список конфиденциальных данных включен только сигнал clk. Это согласуется с тем фактом, что сигнал d дискретизируется только на восходящем фронте сигнала clk, и изменение его значения не вызывает какого-либо немедленного отклика.

DF с асинхронным сбросом ADFF может содержать сигнал асинхронного сброса, как. Сигнал очищает D FF до "O" в любое время и не контролируется тактовым сигналом. На самом деле он имеет более высокий приоритет, чем регулярный входной сигнал. Использование асинхронного сигнала сброса нарушает методологию синхронного проектирования, и поэтому его следует избегать при нормальной эксплуатации. Его основное применение заключается в выполнении инициализации системы. Например, мы можем сгенерировать короткий импульс сброса, чтобы вернуть систему в исходное состояние. состояние после включения питания. Код для D FF с асинхронным сбросом показан в листинг 5.2.

Листинг 5.2 – D триггер с синхронным сбросом

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity d_ff_reset is
      Port (
               clk, reset: in    STD_LOGIC;
               d: in    STD_LOGIC;
               q : out    STD_LOGIC;
end d_ff_reset;

architecture arch of d_ff_reset is
begin
          process(clk, reset)
          begin
            if (reset='l') then
                     q <='0';
            elsif (clk'event and clk='l') then
                     q <= d;
            end if ;
           end process;
end arch;
```

5.3 Регистр

Регистр - это набор различий, которые управляются одними и теми же сигналами синхронизации и сброса. Как и D триггер, регистр может иметь дополнительный асинхронный сигнал сброса и асинхронный сигнал включения. Код идентичен коду DF, за исключением того, что тип данных массива, std-logicvector, необходим для соответствующих входных и выходных сигналов. Например, 8-разрядный регистр с асинхронным сбросом показан в листинге 5.3.

Листинг 5.3 – 8-разрядный регистр

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity reg_reset is
      Port (
               clk, reset: in    STD_LOGIC;
               d: in    STD_LOGIC_VECTOR (7 downto 0);
               q : out    STD_LOGIC_VECTOR (7 downto 0);
end reg_reset;

architecture arch of reg_reset is
begin
          process(clk, reset)
          begin
                   if (reset='l') then
                            q <=(others=>'0')
             elsif (clk'event and clk='l') then
                       q <= d;
             end if ;
           end process;
end arch;
```

Обратите внимание, что выражение (others=>'0') означает, что всем элементам присваивается значение '0' в данном случае эквивалентно 00000000.

5.4 Сдвиговый регистр

Свободно работающий регистр сдвига сдвигает свое содержимое влево или вправо на одну позицию в каждом такте. Другого управляющего сигнала нет. Код для N-разрядного свободно работающего регистра сдвига вправо показан в листинге 5.4.

Листинг 5.4 – Свободный сдвиговый регистр

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
entity free_run_shift_reg is
      Port (
               clk, reset: in    STD_LOGIC;
               s_in: in    STD_LOGIC;
               s_out: out    STD_LOGIC;
end free_run_shift_reg;

architecture arch of free_run_shift_reg is
    signal r_reg : std-logic-vector (N-1 downto 0) ;
    signal r_next : std-logic-vector (N-1 downto 0) ;
begin
          process(clk, reset)
          begin
                   if (reset='l') then
                            r_reg <=(others=>'0');
             elsif (clk'event and clk='l') then
                       r_reg <= r_next;
             end if ;
           end process;
          s_out <= r_reg(0);
end arch;
```

5.5 Регистровый файл

Регистровый файл - это набор регистров с одним входным портом и одним или несколькими выходными портами.

Сигнал адреса записи, w_addr, указывает, где хранить данные, а сигнал адреса чтения, r_addr, указывает, где извлекать данные. Рейсовый файл обычно используется в качестве быстрого временного хранилища. Код для параметризованного регистрового файла 2 в степени W показан в листинге 5.5.

В этом проекте определены два универсальных файла. Общий параметр W определяет количество битов адреса, что подразумевает, что в файле содержится 2^W слов w, а общий параметр B определяет количество битов в слове.

Листинг 5.5 – Регистровый файл

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity reg_file is
generic(
B: integer:=8; -- number of bits
W: integer:=2 -- number of address bits);
      Port (
               clk, reset: in    STD_LOGIC;
               w_addr , r_addr : in std_logic_vector (W-1 downto 0) ;
               w_data: in std_logic_vector (B-1 downto 0) ;
               r_data: out std_logic_vector (B-1 downto 0) ) ;
         end reg_file;
architecture arch of reg_file is
type reg_file_type is array (2**W-1 downto 0) of
  std_logic_vector (B-1 downto 0) ;
signal array_reg: reg_file_type;
begin
           process(clk, reset)
           begin
              if (reset='l') then
              array_reg <= (others=>(others=>'0'));
                 elsif (clk'event and clk='l') then
                 if wr_en='l' then
              array_reg( to_integer (unsigned(w_addr))) <= w_data; end if ;
              end if ;
    end process;
r-data <= array-reg(to-integer(unsigned(r_addr)));
end arch;
```

5.6 Двоичный счетчик

Автономный двоичный счетчик многократно обрабатывает двоичную последовательность. Например, 4-разрядный двоичный счетчик ведет отсчет от "0000" до "1111". Код для параметризованного N-разрядного двоичного счетчика, работающего в автономном режиме, показан в листинге 5.6.

Листинг 5.6 – Двоичный счетчик

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity eq11 is
Generic (N: integer := 4);
      Port ( clk : in    STD_LOGIC;
                  reset : in    STD_LOGIC;
                  q : out    STD_LOGIC_VECTOR (N-1 downto 0);
                  max_tick : out    STD_LOGIC);
end eq11;

architecture Behavioral of eq11 is

signal r_reg: unsigned(N-1 downto 0) ;
signal r_next : unsigned (N-1 downto 0) ;
begin
-- описание регистра
  process (clk,reset)
  begin
     if (reset = '1') then
                    r_reg <= (others => '0');
          elsif (clk'event and clk = '1') then
               r_reg <= r_next;
     end if;
  end process;
-- логика след.состояния
  r_next <= r_reg + 1;
  q <= std_logic_vector(r_reg);
  max_tick <= '1' when (r_reg = 2**N-1) else '0';
  end Behavioral;
```

После имплементации, во вкладке Tools => Schematic viewer можно увидеть синтезированную технологическую схему, которая представлена на рисунке 5.3.

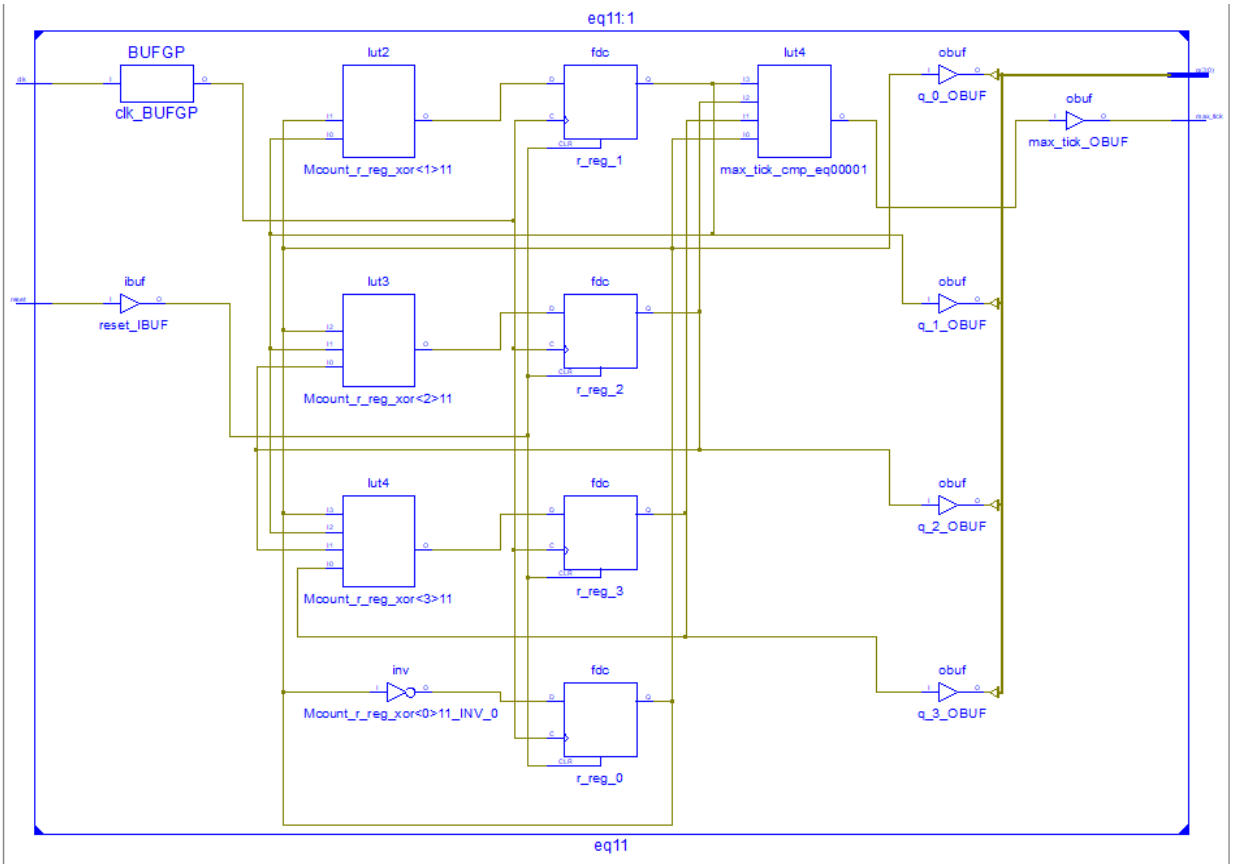


Рисунок 5.3 – Структурная схема двоичного счетчика

Проверка работы счетчика осуществляется написанием файла тестирования Test Bench. Для этого добавляем модуль тестирования для проекта счетчика. В режиме симуляции. В файле тестирования внесем изменения в разделе процесса симуляции, для подачи сигнала сброса начального положения, представленном в листинге 5.7.

Листинг 5.6 – Двоичный счетчик

```
-- Stimulus process
stim_proc: process
    begin
        -- hold reset state for 100 ns.
                reset <= '1';
        wait for 10 ns;
                reset <= '0';

       -- wait for clk_period*10;

        -- insert stimulus here

        wait;
    end process;
```

В приложении Б представлен процесс симуляции двоичного счетчика от 1 до 15.

5.7 Задание на лабораторную работу – Разработать на VHDL схему сдвига вправо и влево.

Разработать на VHDL схему 8-разрядную схему сдвига, обеспечивающую сдвиг 8-разрядного слова на заданное количество бит влево или вправо рисунок 5.4. Количество бит, на которые нужно сдвинуть слово задается 3-х разрядным управляющим сигналом amt, а направление сдвига (вправо/влево) задается одноразрядным сигналом lr. Порядок разработки:

1. Разработать на VHDL 8-разрядную схему сдвига вправо, используя структуру «присваивание по выбору».

2. Разработать на VHDL 8-разрядную схему сдвига влево, используя структуру «присваивание по выбору».

3. Используя готовые блоки «сдвиг вправо» и «сдвиг влево» разработать итоговую универсальную схему сдвига вправо/влево рисунок 5.5.

4. Разработать тестовый модуль (VHDL Test Bench) и проверить работу схемы во встроенном симуляторе ISim.

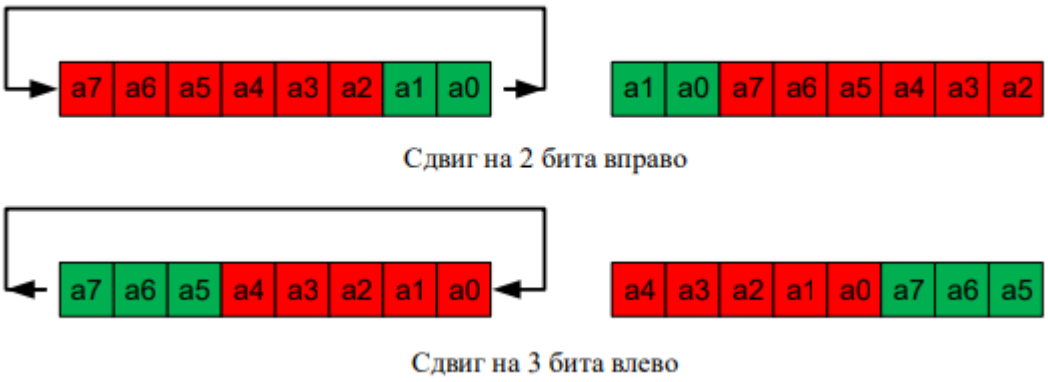


Рисунок 5.4 – Пример работы схемы сдвига

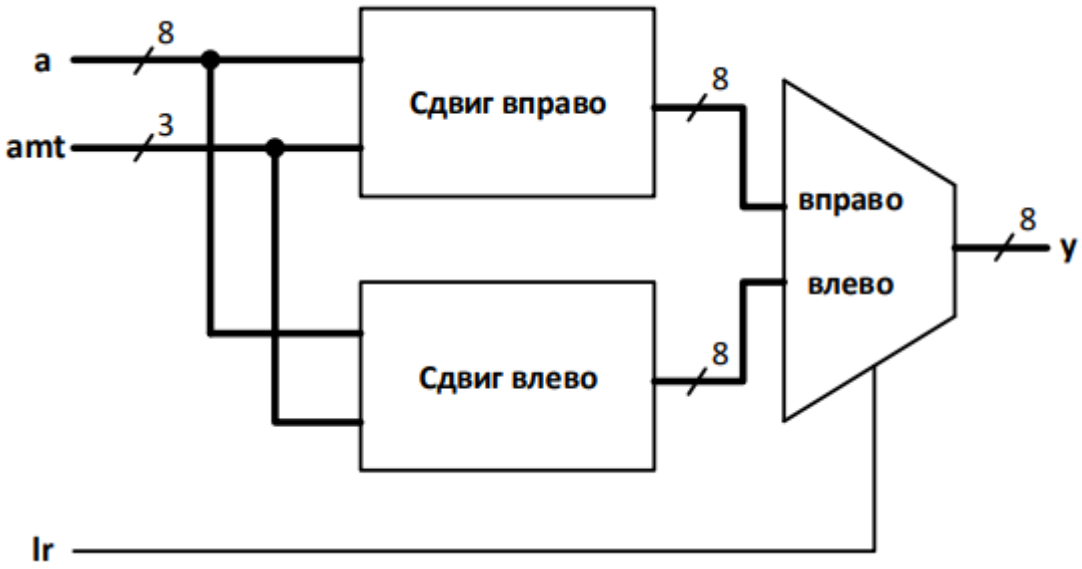


Рисунок 5.5 – Универсальная схема сдвига вправо и влево

5.8 Задание на лабораторную работу – Разработать на VHDL схему 8-разрядного универсального счетчика

Разработать на VHDL схему 8-разрядного универсального счетчика представленного на рисунке 5.6. Данный счетчик позволяет считать в обоих направлениях (вверх и вниз) и ставить счетчик на паузу. Вход dir определяет направление счёта (если 1 – счётчик считает вверх, если 0 – вниз), а вход pause останавливает счёт (если 0 – выполняется счёт, если 1 - останавливается). Вход clk – тактовый, reset – асинхронный сброс счётчика в ноль. Создать технологическую схему.

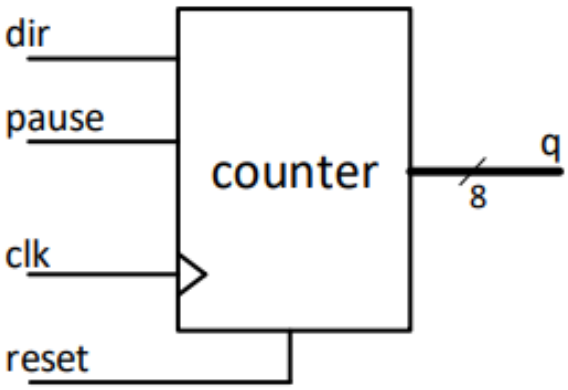


Рисунок 5.6 – Схема восьмиразрядного счетчика

6 Конечные автоматы

Конечный автомат (FSM) используется для моделирования системы, которая переходит между конечным числом внутренних состояний. Переходы зависят от текущего состояния и внешних входных данных. В отличие от обычной последовательной схемы, переходы состояний FSM не демонстрируют простой, повторяющийся паттерн. Его логика следующего состояния обычно строится с нуля и иногда известна как “случайная” логика. Это отличается от логики следующего состояния обычной последовательной схемы, которая состоит в основном из “структурированных” компонентов, таких как инкрементеры и переключатели. На практике основное применение FSM заключается в том, чтобы выступать в качестве контроллера большой цифровой системы, которая проверяет внешние команды и состояние и активирует соответствующие управляющие сигналы для управления работой канала передачи данных, который обычно состоит из обычных последовательных компонентов.

6.1 Автомат Мили Мура

Базовая структурная схема FSM такая же, как и у обычной последовательной схемы, и повторена на рис. 6.1. Он состоит из регистра состояний, логики следующего состояния и логики вывода. FSM известен как машина Мура, если выходные данные являются функцией только состояния, и известен как машина Мили, если выходные данные являются функцией состояния и внешнего ввода. Оба типа выходных данных могут существовать в сложном FSM, и мы просто называем его содержащим выходные данные Мура и Мили выходы.

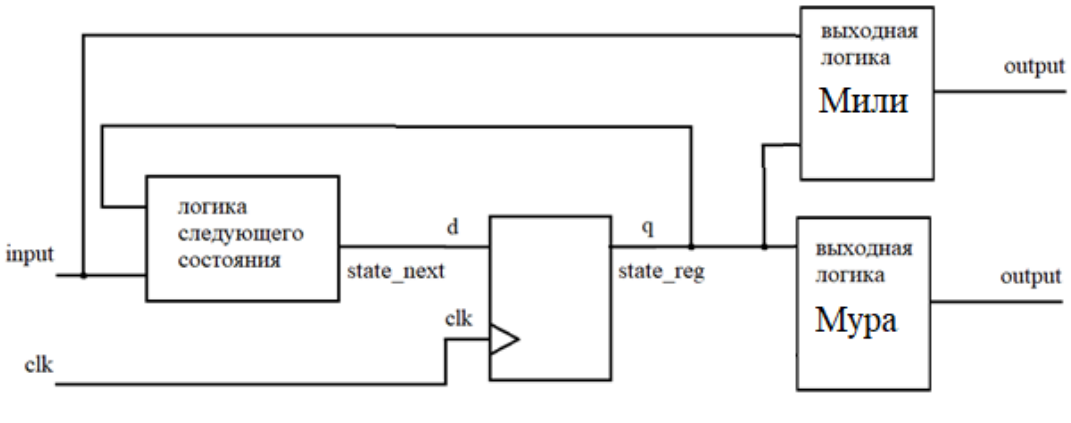


Рисунок 6.1 – Структурная схема синхронной системы

Результаты Мура и Мили схожи, но не идентичны. В автомате Мура выходная логика зависит только от текущего состояния, тогда как в автомате Мили выходная логика зависит еще и от выходных сигналов. Понимание их тонких различий является ключом к разработке контроллера.

Рассмотрим пример работы автомата Мура на управлении светофорами на перекрестке, показанная на рисунке 6.2

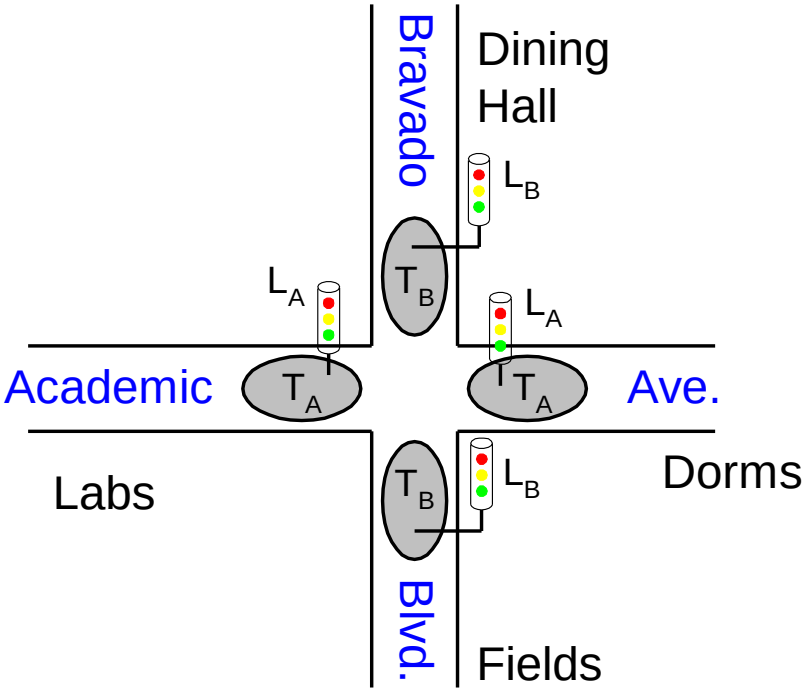


Рисунок 6.2 – Система управления светофорами

Для управления необходимо работа с диаграммой переходов, показанная на рисунке 6.3. В этой диаграмме представлены основные состояние системы со значениями каждого выхода в отдельных кругах, которые преобразуются по дугам переходов.

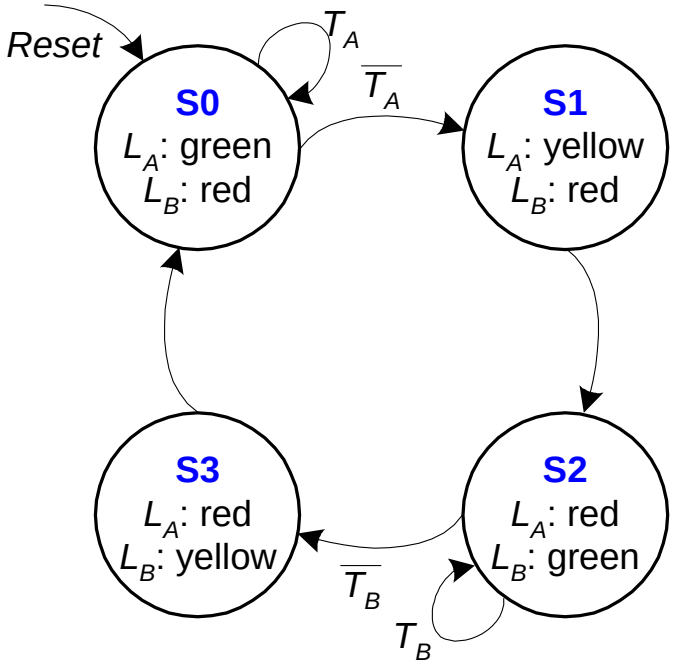


Рисунок 6.3 – Диаграмма переходов автомата Мура

Результатом обобщения значений диаграммы состояний является таблица состояний 6.1. На которой указаны следующие состояние в зависимости от предыдущих.

Таблица 6.3 – Таблица переходов конечного автомата

Текущее состояние S	Входы T_A	Входы T_B	Следующее состояние S'
S0	0	X	S1
S0	1	X	S0
S1	X	X	S2
S2	X	0	S3
S2	X	1	S2
S3	X	X	S0

Для перевода значений необходима кодировка состояний по таблице 6.2.

Таблица 6.2 – Таблица кодирования состояний

Состояние	Кодирование	Выходы	Кодирование
S0	00	Зеленый	00
S1	01	Желтый	01
S2	10	Красный	10
S3	11	X	X

Записать булевые выражения через ДНФ в одну таблицу с двоичным кодированием всех состояний, так как зависимость выходного сигнала зависит как от текущего состояния, так и от входных данных. В таблице 6.3 указаны все значения после преобразований по формулам 6.1.

Таблица 6.3 – Таблица переходов конечного автомата с двоичным кодированием

Текущее состояние		Входы		Следующее состояние	
S1	S0	TA	TB	S1	S'0
0	0	0	X	0	1
0	0	1	X	0	0
0	1	X	X	1	0
1	0	X	0	1	1
1	0	X	1	1	0
1	1	X	X	0	0

$$S'_1 = S_1 \oplus S_0$$

$$S'_0 = S_1 S_0 T_{A} + S_1 S_0 T_{B} \quad (6.1)$$

Таким образом, рассмотрев формирование состояний на примере автомата Мура. Сравним характеры и временные диаграммы обоих автоматов на рисунке 6.4. Для того, что бы обраобоать сигнал – «11100011110011011».

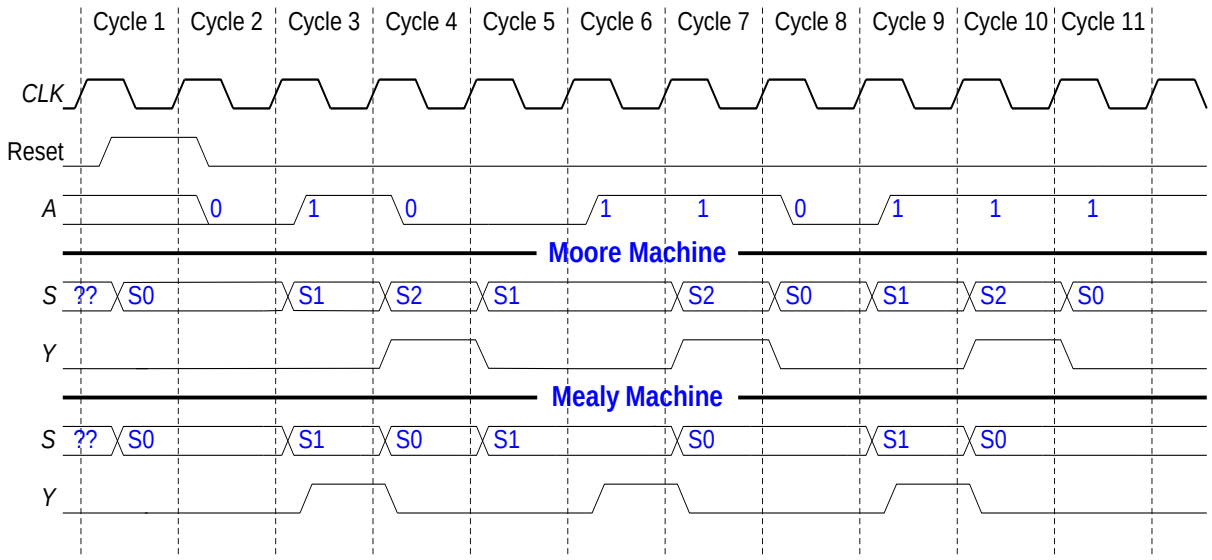


Рисунок 6.4 – Временные диаграммы работы автомата Мура и Мили

В автомате Мура, реакция схемы на комбинацию «01» жестко привязана к работе тактового сигнала , каждый выходной сигнал соответствует сигналам CLK при наличии требуемой пары. Тогда как автомат Мили не зависит от тактового сигнала, а напрямую изменяется по

изменению входного сигнала. Удобство автомата Мили является его быстродействие в сравнении с более синхронным автоматом Мура.

Используем шаблон работы автоматов в ISE Design Suite 14.7. Для этого перейдем во вкладку Edit далее выбираем раздел Laguage Templates… На рисунке 6.5. выберем шаблон машины Мура.

В шаблоне необходимо прописать работу автомата Мура на основании работы диаграммы.

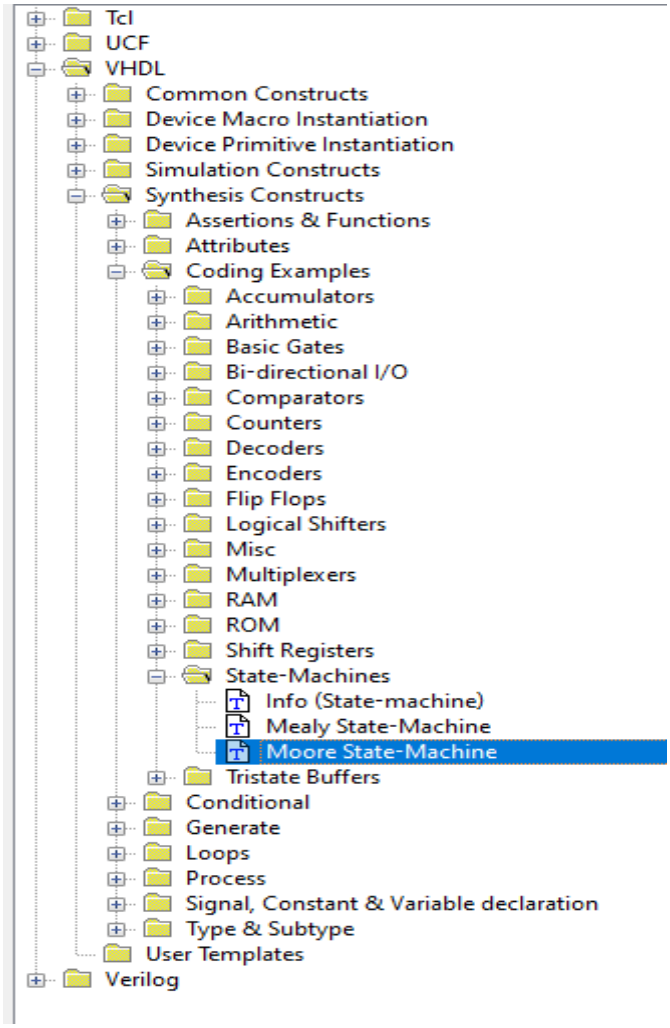


Рисунок 6.5 – Выбор шаблона автомата Мура

В листинге 6.1 необходимо указать правила реакции автомата Мура, для запуска симуляции, представленной в приложении В.

Листинг 5.5 – Регистровый файл

```
NEXT_STATE_DECODE: process (state, <input1>, <input2>, ...)
    begin
        case (state) is
```

```
            when st1_<name> =>
                if <input_1> = '1' then
                    next_state <= st2_<name>;
                end if;
            when st2_<name> =>
                if <input_2> = '1' then
                    next_state <= st3_<name>;
                end if;
            when st3_<name> =>
                next_state <= st1_<name>;
            when others =>
                next_state <= st1_<name>;
        end case;
    end process;
```

6.2 Задание на лабораторную работу – Схема счетчика количества автомобилей на парковке.

Разработать на VHDL схему счетчика количества автомобилей на парковке на рисунке 6.6. Два фотодатчика a и b используются для детектирования автомобилей – когда автомобиль перекрывает пространство между излучателем и приёмником, датчик вырабатывает сигнал ‘1’ (рисунок 1). Например, при въезде автомобиля на парковку наблюдается следующая последовательность сигналов с датчиков:

- Датчики a и b «открыты» - сигналы с датчиков “00”.
- Датчик a «закрыт», датчик b «открыт» - сигналы с датчиков “10”.
- Датчик a «закрыт», датчик b «закрыт» - сигналы с датчиков “11”.
- Датчик a «открыт», датчик b «закрыт» - сигналы с датчиков “01”.
- Датчики a и b «открыты» - сигналы с датчиков “00”.

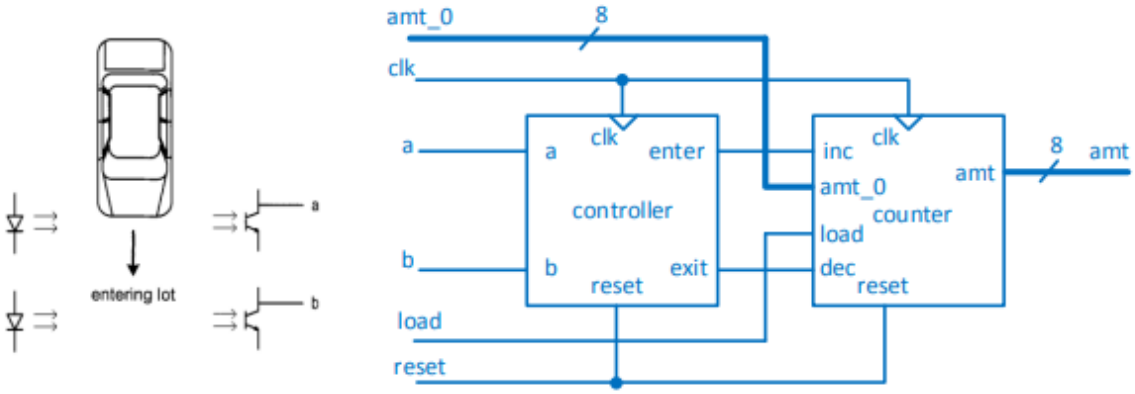


Рисунок 6.6 – Схема счетчика количества автомобилей на парковке

1. Разработать на VHDL схему конечного автомата для контроля въезда/выезда автомобиля на парковку. На вход схемы подаются сигналы с датчиков a и b, тактовый сигнал clk и сигнал асинхронного сброса reset. На выходе схемы – два сигнала enter и exit, соответствующие

въезду и выезду автомобиля соответственно. Длительность сигналов “1” на выходе enter (при въезде автомобиля) и exit (при выезде автомобиля) – 1 такт.

2. Разработать схему универсального счетчика с возможностью прямого и обратного счёта. Счетчик имеет входы inc для прямого счёта и dec для обратного счёта, тактовый сигнал clk и сигнал асинхронного сброса reset. Также предусмотрен 8-ми разрядный вход amt_0 для загрузки из ПЗУ начального количества автомобилей, имеющихся на парковке, и сигнал load для управления загрузкой amt_0. На выход схемы выдается 8-ми разрядный сигнал amt, соответствующий текущему количеству автомобилей на парковке.

3. Создать общий проект, в котором объединить схему контроллера въезда/выезда и счётчик. Разработать testbench проекта и проверить работу в симуляторе.

4. К отчёту помимо файлов .vhd должна быть приложена диаграмма переходов конечного автомата.

5. Соблюдать синхронность всех сигналов, кроме сигнала «Reset».

ПРИЛОЖЕНИЕ А
Примеры команд управления в User Constraint File

В данном ghbkj;tybb перечислены параметры UCF файла, присвоенные движковым переключателям SW0-SW7 В приведённых ниже параметрах Swx относится к цепи соответствующего движкового переключателя, LOC указывает расположение линии ПЛИС, IOSTANDARD указывает используемый стандарт линий ввода/вывода.

```
Net "SW0" LOC="J11" | IOSTANDARD = LVCMOS33;

Net "SW1" LOC="J12" | IOSTANDARD = LVCMOS33;

Net "SW2" LOC="H16" | IOSTANDARD = LVCMOS33;

Net "SW3" LOC="H13" | IOSTANDARD = LVCMOS33;

Net "SW4" LOC="G12" | IOSTANDARD = LVCMOS33;

Net "SW5" LOC="E14" | IOSTANDARD = LVCMOS33;

Net "SW6" LOC="D16" | IOSTANDARD = LVCMOS33;

Net "SW7" LOC="B16" | IOSTANDARD = LVCMOS33;
```

В данном подпункте перечислены параметры UCF файла, присвоенные кнопкам BTN0-BTN7. В приведённых ниже параметрах BTNx относится к цепи соответствующей кнопки, LOC указывает расположение линии ПЛИС, IOSTANDARD указывает используемый стандарт линий ввода/вывода.

```
Net "BTN0" LOC="C13" | IOSTANDARD = LVCMOS33;

Net "BTN1" LOC="D12" | IOSTANDARD = LVCMOS33;

Net "BTN2" LOC="C12" | IOSTANDARD = LVCMOS33;

Net "BTN3" LOC="C10" | IOSTANDARD = LVCMOS33;
```

Светодиоды В данном подпункте перечислены параметры UCF файла, присвоенные светодиодам LED0-LED7.

В приведённых ниже параметрах LEDx относится к цепи соответствующего светодиода, LOC указывает расположение линии ПЛИС, IOSTANDARD указывает используемый стандарт линий ввода/вывода, SLEW указывает скорость нарастания выходного напряжения, DRIVE указывает выходной ток источника тока встроенного в ПЛИС в миллиамперах.

```
Net "LED0" LOC="C11" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;

Net "LED1" LOC="D11" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;

Net "LED2" LOC="B11" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;
```

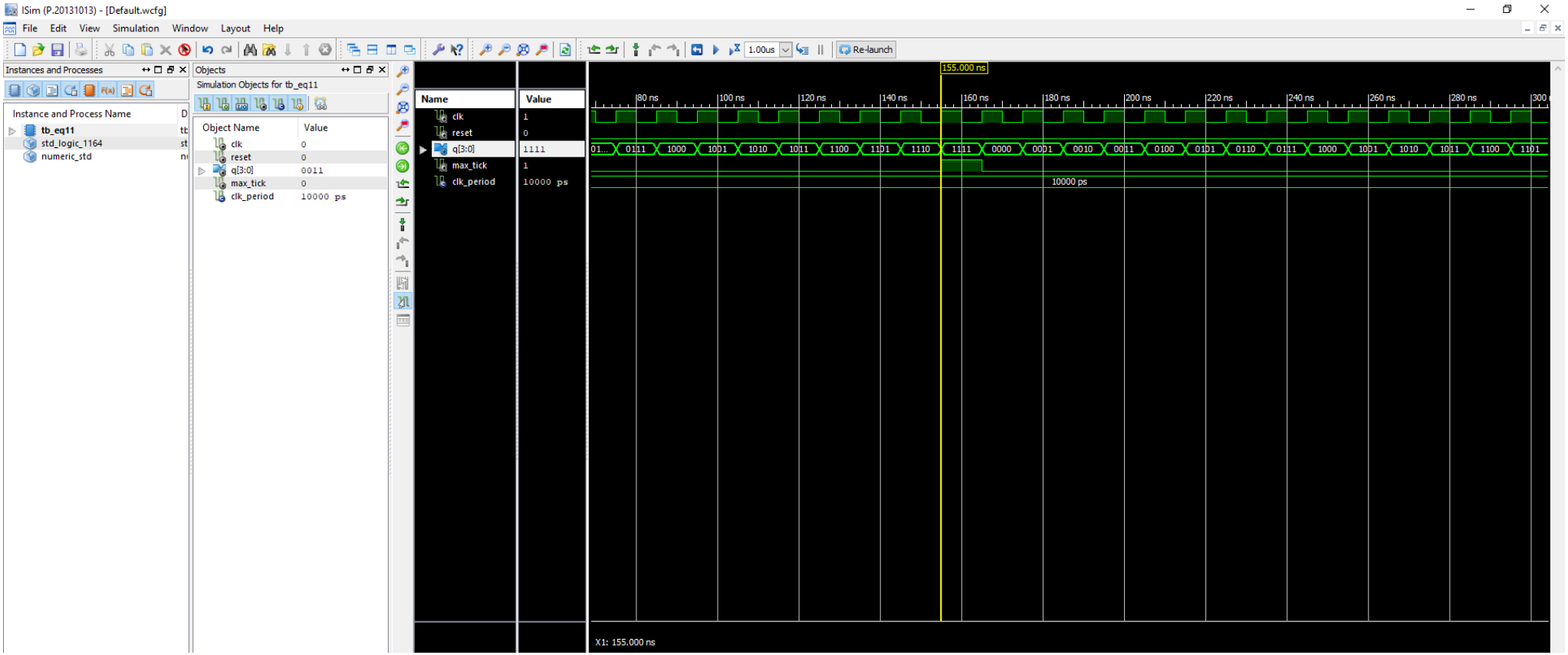
```
Net "LED3" LOC="A12" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;

Net "LED4" LOC="A13" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;

Net "LED5" LOC="B13" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;

Net "LED6" LOC="A14" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;

Net "LED7" LOC="B14" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;
```

Семисегментные индикаторы

В данном подпункте перечислены параметры UCF файла, присвоенные семисегментным индикаторам. В приведённых ниже параметрах SEGxx относится к цепи соответствующего сегмента индикатора, COMx относится к относится к цепи анода соответствующей цифры индикатора. LOC указывает расположение линии ПЛИС, IOSTANDARD указывает используемый стандарт линий ввода/вывода, SLEW указывает скорость нарастания выходного напряжения, DRIVE указывает выходной ток источника тока встроенного в ПЛИС в миллиамперах.

```
Net "SEGA0" LOC="E3" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;

Net "SEGB0" LOC="E1" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;

Net "SEGC0" LOC="G5" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;

Net "SEGD0" LOC="D1" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;

Net "SEGE0" LOC="E4" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;

Net "SEGF0" LOC="C1" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;

Net "SEGG0" LOC="C2" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;

Net "COM0" LOC="B2" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;

Net "SEGA1" LOC="H6" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;

Net "SEGB1" LOC="K2" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;

Net "SEGC1" LOC="H3" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;

Net "SEGD1" LOC="K1" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;

Net "SEGE1" LOC="G4" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;

Net "SEGF1" LOC="J2" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;

Net "SEGG1" LOC="G3" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;

Net "COM1" LOC="G2" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;
```

ПРИЛОЖЕНИЕ Б
Пошаговый процесс симуляции двоичного счетчика



ПРИЛОЖЕНИЕ В
Пошаговый процесс симуляции автомата Мура