МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ
ЧЕРНИГОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ

ПРОЕКТИРОВАНИЕ НА VHDL В САПР QUARTUS II

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к самостоятельной работе и к выполнению расчётно-графической работы
по дисциплине “Технологии проектирования компьютерных систем”
для студентов направления подготовки
6.050102 – “Компьютерная инженерия”

Обсуждено и рекомендовано
на заседании кафедры
информационных и компьютерных систем
Протокол № __
от __.__.2013 г.

Чернигов ЧГТУ 2013

Проектування на VHDL в САПР QUARTUS II. Методичні вказівки до самостійної роботи та до виконання розрахунково-графічної роботи з дисципліни "Технології проектування комп'ютерних систем" для студентів напряму підготовки 6.050102 – "Комп'ютерна інженерія"/ Укл.: Вервейко О.І., Красножон О.В. – Чернігів: ЧДТУ, 2013. – 71 с. Рос. мовою.

Составители: Вервейко Александр Иванович, кандидат технических наук, доцент, доцент кафедры информационных и компьютерных систем
Красножон Алексей Васильевич, ассистент кафедры информационных и компьютерных систем

Ответственный за выпуск: Казимир Владимир Викторович, заведующий кафедрой информационных и компьютерных систем, доктор технических наук, профессор

Рецензент: Нестеренко Сергей Александрович, кандидат технических наук, доцент кафедры информационных и компьютерных систем Черниговского государственного технологического университета

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время невозможно представить жизнь без вычислительных устройств, скрытых в бытовой технике, измерительных и медицинских приборах и т.п. Если мысленно охватить всю электронную технику, используемую человеком, то на один универсальный микропроцессор приходятся многие десятки специализированных вычислительных устройств (ВУ).

Проектирование ВУ представляет собой поэтапное преобразование технического описания устройства с уточнением и детализацией на каждом новом этапе. Традиционное проектирование ВУ основано на составлении на каждом этапе различных графических схем. В соответствии с назначением этапа, различают схемы алгоритмов, схемы электрические структурные, принципиальные и функциональные. Повсеместное внедрение САПР электронных схем обеспечивает уменьшение трудозатрат, повышение качества проектов, но не меняет в корне характер проектирования, как ручного описания ВУ.

Использование при разработке вычислительных устройств таких языков проектирования, как VHDL и Verilog позволяет отказаться от составления схем на большинстве этапов разработки.

VHDL (**V**HSIC (Very high speed integrated circuits) **H**ardware **D**escription **L**anguage) – язык описания аппаратуры интегральных схем. Язык проектирования VHDL является базовым языком при разработке аппаратуры современных вычислительных систем.

Язык VHDL предназначен для решения комплекса задач в ходе проектирования и применения цифровых систем, их аппаратных средств, в том числе:

– описания структуры системы, декомпозиции системы на подсистемы, спецификации связей и взаимодействия подсистем;

– спецификации функционирования системы, узлов, блоков, реализуемых функций. Спецификация дается в алгоритмической форме, с использованием привычных для современных специалистов программных конструкций алгоритмического языка, включающих в себя спецификацию временного поведения сигналов и блоков;

– моделирования системы и ее работы на основе четкой спецификации структуры системы, а также функционирования ее компонентов;

– синтеза схемотехнической реализации системы, автоматической генерации детальной структуры на основе строгой спецификации системы на языке VHDL – спецификации на более абстрактном уровне.

Язык VHDL используют для описания структуры и функционирования системы. Такое описание, достаточно формализованное и однозначное, имеет уже самостоятельную ценность, как средство передачи знаний о спроектированной аппаратно реализованной цифровой системе (устройстве, блоке) от разработчика к специалисту, ее применяющему.

Язык VHDL – необычный язык как для программиста так и для разработчика аппаратуры. Это связано с тем, что по своей природе VHDL – язык параллельного программирования. Начинающие программисты иногда путаются в

основах программирования на VHDL, допускают трудно выявляемые ошибки, а их результирующие проекты часто оказываются далеки от оптимальных.

Для избегания появления таких ошибок разработчики должны заниматься изучением возможностей предоставляемых языком VHDL. Знание особенностей языка, существующих в нем возможностей и синтаксических конструкций, во многом избавляет разработчика от необходимости описывать действия, уже присутствующие в языке, самостоятельно. Это значительно сокращает время разработки, уменьшает количество ошибок и позволяет сделать результирующее устройство более оптимальным.

1 САМОСТОЯТЕЛЬНАЯ РАБОТА

1.1 Цель и задачи дисциплины

Дисциплина "Технологии проектирования компьютерных систем" входит в число дисциплин самостоятельного выбора студента цикла подготовки по компьютерной схемо- и системотехники профессиональной программы высшего образования по направлению 6.050102 – "Компьютерная инженерия" и рассматривается кафедрой как составная часть специальной подготовки в области разработки аппаратных средств ЭВМ вместе с дисциплинами "Прикладная теория цифровых автоматов", "БИС МП и ПЛ", "Периферийные устройства" и "Архитектура компьютеров". Необходимыми предпосылками для преподавания дисциплины "Технологии проектирования компьютерных систем" является усвоение студентами материала дисциплин «Математика», «Физика», «Теория электрических цепей", "Компьютерная электроника" и "Компьютерная схемотехника" .

Целью изложения дисциплины "Технологии проектирования компьютерных систем" является формирование у студентов научного уровня инженерного мышления будущего специалиста, содержит в себе теоретическую базу, которая необходима при усвоении прикладных вопросов разработки и эксплуатации аппаратных средств ЭВМ, а также компьютерных систем и сетей. После изучения дисциплины студенты должны иметь знания:

- о построении, принципах действия, основных параметрах и характеристиках сверхбольших интегральных схем программируемой логики (СБИС ПЛ);
- об особенностях построения цифровых устройств на основе (СБИС ПЛ);
- о современных методах проектирования цифровых устройств;
- о языках описания аппаратных средств (ЯОА);
- о методах описания цифровых устройств с использованием МОА;
- о применении редакторов временных диаграмм и диаграмм состояний цифрового автомата для синтеза цифровых устройств;
- об основных параметрах и характеристиках современных систем автоматизированного проектирования (САПР);
- о методиках практического применения САПР синтеза для цифровых устройств различного назначения.

В результате освоения учебной дисциплиной студенты должны уметь:

- анализировать параметры и характеристики СБИС ПЛ с целью получения необходимых параметров и характеристик цифровых устройств;
- проектировать цифровые устройства с использованием ЯОА и редакторов временных диаграмм и диаграмм состояний цифрового автомата;
- разрабатывать специализированные устройства на базе СБИС ПЛ;
- правильного применить СБИС ПЛ в цифровых устройствах и узлах;
- выполнять математическое моделирование цифровых устройств;
- экспериментально определять параметры и характеристики цифровых устройств.

Значение дисциплины для реализации требований квалификационной характеристики специалиста и изучение следующих дисциплин заключается в том, что дисциплина способствует формированию алгоритмического мышления будущего специалиста, создает базу, которая необходима при изучении многих последующих дисциплин.

В связи с интенсивным развитием средств вычислительной техники основное внимание уделено изучению современных методов проектирования цифровых устройств разного назначения на основе СБИС ПЛ, а также приобретению знаний, навыков и практического опыта в проектировании реконфигурируемых устройств с применением современных САПР.

Преподавание дисциплины "Технологии проектирования компьютерных систем" обусловлено необходимостью формирования у студентов четкой системы представлений о целостном комплексе проблем, которые должны быть решены в процессе разработки узлов и специализированных устройств ЭВМ.

Дисциплина является базовой для дисциплин "Архитектура компьютеров", "Проектирование специализированных КС", "Проектирование встроенных КС", "Компьютерные системы", "Телекоммуникационные системы и технологии".

1.2 Содержание дисциплины

1.2.1 Введение

История развития методов проектирования цифровых устройств и ЯОА. Предмет и цели курса.

1.2.2 Методы проектирования цифровых устройств

Тенденции развития цифровых устройств. Традиционные методы проектирования: проектирование с помощью булевых функций, схемотехническое проектирование. Недостатки традиционных методов. Методы разработки: схемотехническое проектирование с использованием САПР, проектирование с помощью языков описания аппаратуры, описание устройства с использованием временных диаграмм его работы, применение диаграмм состояний для описания работы последовательных устройств. Современный цикл проектирования цифровых устройств.

1.2.3 История создания языка VHDL

Первые языка описания аппаратных средств. Этапы развития языка VHDL. Центры поддержки языка VHDL.

1.2.4 Алфавит языка VHDL и его лексические элементы

Алфавит языка. Лексические элементы. Разделители и ограничители. Идентификаторы. Литералы: перечисление, числа, срочные литералы, битовые строки. Комментарии. Ключевые (зарезервированные) слова.

1.2.5 Типы данных

Скалярные типы: перечисление, целые числа, физические типы, числа с плавающей точкой. Составные типы: массивы, записи. Указатели. Файлы. Функции преобразования типов данных.

1.2.6 Имена

Простое имя. Символ оператора. Селективное имя. Индексное имя. Вырезка имени. Имя атрибута. Видимость и область действия имен.

1.2.7 Выражения

Операторы: логические, сравнения, сложения и конкатенации, присвоения знака, сдвига, умножения и деления, смешанные.

Операнды: Name, Literal, Aggregate, Function Call, Qualified Expression, Type Conversion, Allocator, статическое выражение, универсальное выражение.

1.2.8 Представление системы в VHDL

Общая структура описания проекта системы. Сущность проекта системы. Архитектура проекта системы. Предложения VHDL.

1.2.9 Структурное описание цифровых устройств

Сигналы в VHDL. Описание нерегулярных структур. Описание регулярных структур.

1.2.10 Функциональное описание цифровых устройств

Описание комбинационных устройств: особенности применения сигналов, параллельные и последовательные предложения, подпрограммы; примеры описания шифраторов и умножителей.

Описание последовательных устройств: триггеров, регистров, счетчиков, цифровых автоматов, запоминающих устройств.

Применение типа данных RECORD при описании цифровых устройств.

1.2.11 Особенности описания цифровых автоматов в САПР фирмы ALTERA

Основные положения разработки проекта в редакторах временных диаграмм и диаграмм состояний. Правила создания комбинационных и последовательных устройств: триггеров, регистров, цифровых автоматов. Примеры описания цифровых устройств.

1.3 Распределение объема самостоятельной работы

Распределение объема самостоятельной работы указано в таблице 1.1.

Таблица 1.1 – Распределение объема самостоятельной работы по видам работ

Вид работы	Объем, часов
1 Усвоение лекционного материала	6
2 Подготовка к лабораторным работам	4
3 Подготовка отчетов по лабораторным работам	4
4 Работа с методическими материалами, основной и вспомогательной литературой	3
5.Самостоятельное изучение на основе учебной литературы следующих вопросов: 1 Особенности проектирования цифровых устройств в САПР QUARTUS; 2 Методики определения временных параметров цифровых устройств	8
6 Выполнение расчетно-графической работы	8
Всего за курс	**33**

1.4 Экзаменационные вопросы

1 Тенденции развития цифровых устройств.
2 Способы создания быстродействующих устройств.
3 Традиционные методы проектирования цифровых устройств.
4 Новые методы проектирования цифровых устройств.
5 История создания языка VHDL.
6 Алфавит языка VHDL.
7 Лексические элементы.
8 Типы данных языка VHDL.
9 Скалярные типы.
10 Составные типы данных языка VHDL.
11 Array_type_definition (массивы) языка VHDL.
12 Record_type_definition (записи) языка VHDL.
13 Функции преобразования типов данных.
14 Формы имен языка VHDL.
15 Выражения языка VHDL.
16 Операторы языка VHDL.
17 Операнды языка VHDL.
18 Общая структура описания проекта системы.
19 Сущность проекта системы.
20 Параметры настройки системы (Generic) сущности проекта.
21 Порты связи (Port) сущности проекта системы.
22 Архитектура проекта системы.
23 Предложения языка VHDL.

24 Предопределенные атрибуты языка VHDL.

25 Атрибуты для скалярных типов данных и сигналов.

26 Атрибуты для массивов и сигналов.

27 Особенности структурного описания цифровых устройств.

28 Предложение Component Declaration.

29 Сигналы в VHDL.

30 Декларация сигналов в VHDL.

31 Предложение Component Instantiation Statement.

32 Описание нерегулярных структур на примитивах.

33 Описание нерегулярных структур на макрофункциях.

34 Описание нерегулярных структур на компонентах пользователя.

35 Описание нерегулярных структур на параметризированных модулях с применением 2D массивов.

36 Описание регулярных структур.

37 Декларация конфигурации.

38 Конфигурируемая архитектура компонента.

39 Конфигурация компонентов в проекте.

40 Конфигурация компонента и его имен портов.

41 Configuration specification.

42 Спецификация архитектуры компонента.

43 Спецификация компонента и его имен портов.

44 Описание комбинационных устройств.

45 Типы Concurrent Signal Assignment Statements.

46 Conditional Signal Assignment Statement.

47 Selected Signal Assignment.

48 Описание комбинационных устройств с применением 2D массивов.

49 Process Statement.

50 Особенности применения сигналов.

51 Variable Declaration Statement и Variable Assignment Statement.

52 Различия между переменными и сигналами.

53 Особенности применения сигналов и переменных.

54 Формы подпрограмм.

55 Процедуры.

56 Функции.

57 Функции вида Pure и Impure.

58 Виды последовательных предложений.

59 Предложение If Statement.

60 Предложение Case Statement.

61 Предложение Loop Statement.

62 Предложение Loop Statement с итерационной схемой While. Фрагмент программы.

63 Предложение Loop Statement с предложением Next Statement. Фрагмент программы.

64 Предложение Loop Statement с предложением Exit Statement . Фрагмент программы.

65 Предложение Return Statement. Фрагмент программы.

66 Предложение Assertion Statement. Фрагмент программы.

67 Предложение Report statement. Фрагмент программы.

68 Формы предложения Wait Statement.

69 Предложение Wait Statement.

70 Цифровые автоматы.

71 Описание цифрового автомата с автоматической кодировкой состояний.

72 Описание цифрового автомата с кодировкой состояний, задаваемых пользователем.

73 Описание цифрового автомата с применением его диаграммы состояний.

74 Описание цифрового автомата с применением его таблицы состояний.

75 Общие правила создания цифровых устройств с использованием временных диаграмм их работы.

76 Правила создания комбинационных устройств с использованием временных диаграмм их работы.

77 Правила создания регистров с использованием временных диаграмм их работы.

78 Правила создания цифровых автоматов с использованием временных диаграмм их работы.

79 Описание последовательных устройств.

80 Особенности описания D-триггеров.

81 Особенности описания JK-регистров.

82 Особенности описания параллельных счетчиков.

83 Особенности описания последовательных счетчиков.

84 Описание синхронных ОЗУ с помощью параметризированных модулей.

85 Описание асинхронных ОЗУ с помощью параметризированных модулей.

86 Модуль EAB микросхемы FLEX10K фирмы Altera.

87 Применение типа данных RECORD при описании цифровых устройств.

88 Поведенческое описание ОЗУ.

89 Поведенческое описание последовательных ПЗУ.

90 Поведенческое описание параллельных ПЗУ.

2 ОСОБЕННОСТИ ПРИМЕНЕНИЯ VHDL В САПР QUARTUS

2.1 Параллельное предложение Block Statement

Параллельное предложение Block Statement предназначено для иерархической декомпозиции программы на языке VHDL, для управления областью действия описаний переменных, процедур.

Синтаксис предложения Block Statement имеет вид:

```
block_statement ::=
block_label :
        block [ ( guard_expression ) ] [ is ]
                block_header
                block_declarative_part
        begin
                block_statement_part
        end block [ block_label ] ;

block_header ::=
[ generic_clause
[ generic_map_aspect ; ] ]
[ port_clause
[ port_map_aspect ; ] ]

block_declarative_part ::=
{ block_declarative_item }

block_statement_part ::=
{ concurrent_statement }
```

Сторожевое выражение (guarded_expression) – выражение булевского типа, которое служит для определения значения описанного внутри блока сигнала, называемого сторожем (quard). Если изменяет свое значение один из сигналов, входящих в состав сторожевого выражения, то значение сигнала quard немедленно пересчитывается. Использование механизма сторожевых выражений позволяет сгруппировать внутри блока операторы присваивания, для которых условие выполнения включает в себя общий компонент.

Рассмотрим в качестве примера описание D-триггера с помощью блоков.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY vdff1 IS
        PORT (D, clk        : IN std_logic;
                Q, Q1, Q2  : OUT std_logic);
END ENTITY vdff1;
ARCHITECTURE behav OF vdff1 IS
```

```
    BEGIN
      blck: BLOCK (clk = '1') IS
          BEGIN
              Q <= GUARDED D;
      END BLOCK;
      blck1: BLOCK (rising_edge(clk)) IS
           BEGIN
              Q1 <= GUARDED D;
              Q2 <= D;
      END BLOCK;
END ARCHITECTURE behav;
```

Временные диаграммы приведены на рисунке 2.1.

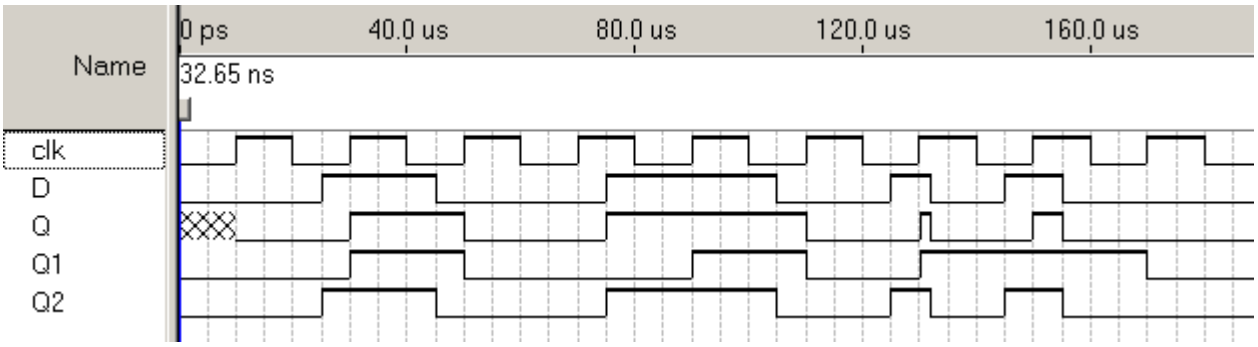

Рисунок 2.1 – Временные диаграммы работы цифровых устройств, описанных предложением Block Statement

Из анализа временных диаграмм следует, что в первом случае синтезирован статический триггер, во – втором . динамический, в третьем – буфер.

Параллельные предложения присваивания сторожевого сигнала может быть двух типов:

– Conditional Signal Assignment Statement;

– Selected Signal Assignment.

Conditional Signal Assignment Statement имеет формат:

```
__label:
__signal <= [ guarded]
                    __expression WHEN __boolean_expression ELSE
                    __expression WHEN __boolean_expression ELSE
                    __expression;
```

Selected Signal Assignment имеет формат:

```
__label:
WITH __expression SELECT
      __signal <= [guarded]
                    __expression WHEN __constant_value,
                    __expression WHEN __constant_value,
```

```
__expression WHEN __constant_value,
__expression WHEN __constant_value;
```

Внешне данные предложения отличаются от других наличием ключевого слова guarded. Это означает, что такой оператор выполняется, когда сигнал-сторож изменяет свое значение. Если сигнал-сторож изменяет свое значение с true на false, драйвер сторожевого сигнала автоматически отсоединяется с использованием нулевой транзакции. При изменении значения сигнала-сторожа с false на true драйвер вновь присоединяется.

Рассмотрим пример описания D-триггера с асинхронной установкой в ноль.

```
ENTITY vdff IS
        PORT (d, clk, clr   : IN bit;
              q, q1         : BUFFER bit);
END ENTITY vdff;
ARCHITECTURE behav OF vdff IS
    BEGIN
        blck: BLOCK (clk = '1' OR clr = '1')
           BEGIN
               q <=GUARDED
                       '0' WHEN clr = '1' ELSE
                        D WHEN clk = '1' ELSE
                        unaffected;
               WITH clk&clr SELECT
                       q1 <= GUARDED
                               '0' WHEN "01",
                               d WHEN "10",
                               q1 WHEN others;
        END BLOCK;
END ARCHITECTURE behav;
```

Временные диаграммы D-триггера с установкой в ноль приведены на рисунке 2.2.

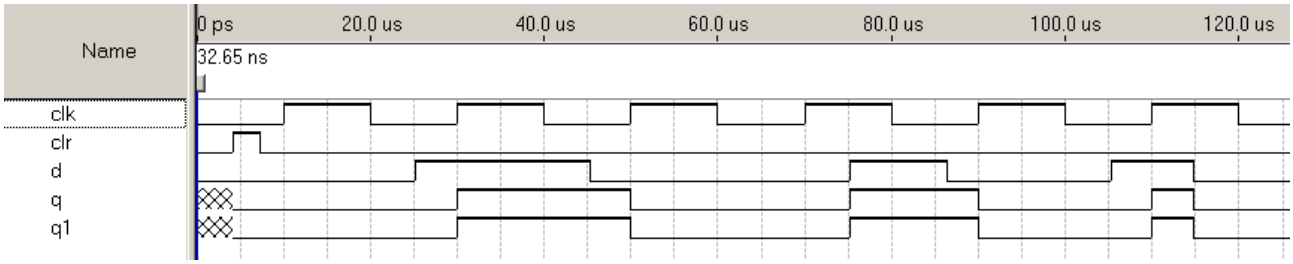

Рисунок 2.2 – Временные диаграммы работы цифровых устройств, описанных с помощью сторожевого сигнала

Из анализа временных диаграмм следует, что в обоих случаях синтезирован статический триггер.

Описанный выше механизм неявного определения сигналов-сторожей на базе сторожевых выражений, задаваемых в заголовке описания блока, оказывается недостаточным для многих моделей. VHDL предоставляет и более общий механизм для определения сигнала-сторожа – механизм явного определения сигнала-сторожа.

При явном определении сигнал-сторож также имеет булевский тип. При изменении его с true на false драйверы сторожевых охраняемых сигналов отключаются, а при изменении с false на true – подключаются.

Явно определенный сигнал-сторож обязательно должен носить имя guard. Его действие распространяется на всю область определения и, соответственно, в этой области не может быть другого сигнала-сторожа.

Рассмотрим особенности применения сигнала-сторожа на примере.

```
ENTITY storog IS
PORT (in1, in2 : IN bit;
            q, q1   : OUT bit);
END storog;
ARCHITECTURE arch OF storog IS
 BEGIN
      blck : BLOCK
       SIGNAL guard : bit;
            BEGIN
                 guard <= in1;
                 q <= guarded in2;
                 q1 <= in1 AND in2;
END BLOCK blck;
END ARCHITECTURE arch;
```

Из анализа временных диаграмм, приведенных на рисунке 2.3, видно, что по выходу q синтезирован статический D-триггер.

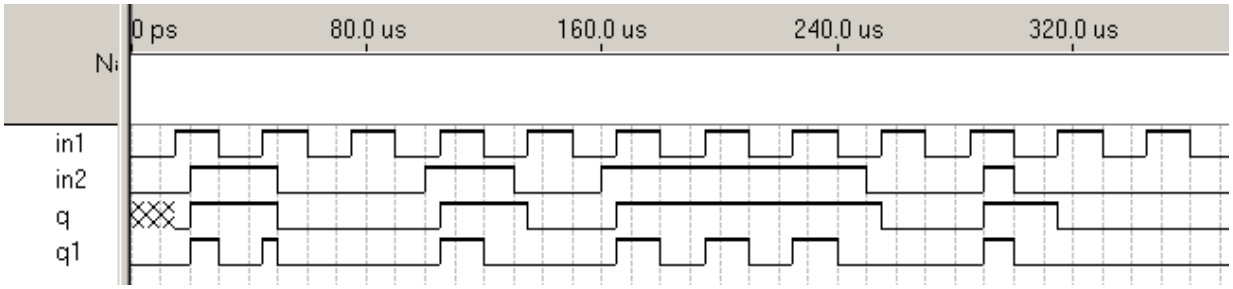

Рисунок 2.3 – Временные диаграммы работы цифровых устройств, описанных с помощью сигнала-сторожа

В сложных проектах иногда возникает необходимость нескольким разработчикам работать в одной архитектуре. Если разработчик будет использовать глобальные имена или параметры настройки, то при возникновении необходимости изменить любое из этих имя необходимо изменять их по всему коду. Во избежание этого используют заголовок блока (block_header).

Заголовок блока явно идентифицирует ряд параметров настройки или сигналов, которые должны быть импортированы из окружающей среды в блок и сопоставлены формальным параметрам настройки или сигналам блока.

```
ENTITY cnt1 IS
  GENERIC (numb :integer:=10; max : integer:=5);
   PORT ( a, b, c, r          :IN bit;
                       q      : OUT integer RANGE 0 TO numb-1);
END ENTITY;
ARCHITECTURE behav OF cnt1 IS
 SIGNAL clka: bit;
        BEGIN
        clka <= a AND b AND c;
         blck: BLOCK IS
                GENERIC (numb_b :integer; max_b : integer);
                GENERIC   MAP (numb_b => numb, max_b => max);
                PORT (clk, res      :IN bit;
                           qb       :BUFFER integer RANGE 0 TO numb_b-1);
                PORT MAP (clk => clka, res => r, qb =>q);
                  BEGIN
                       PROCESS (clk)
                        BEGIN
                          IF r='0' THEN qb <=0;
                            ELSIF (clk'EVENT AND clk ='1')
                                  THEN qb <= qb + 1;
                                       IF qb = max_b THEN qb <=0; END IF;
                              END IF;
                       END PROCESS;
                END BLOCK;
END ARCHITECTURE behav;
```

Из анализа описания следует, что в одной архитектуре можно не задумываться над именами параметров настройки и сигналов. Внутренние параметры настройки и сигналы видны только в блоке, а связь с внешними параметрами настройки и сигналами задают в port map и generic map.

Цифровое устройство по приведенному описанию представляет собой делитель на 5 (смотри рисунок 2.4).

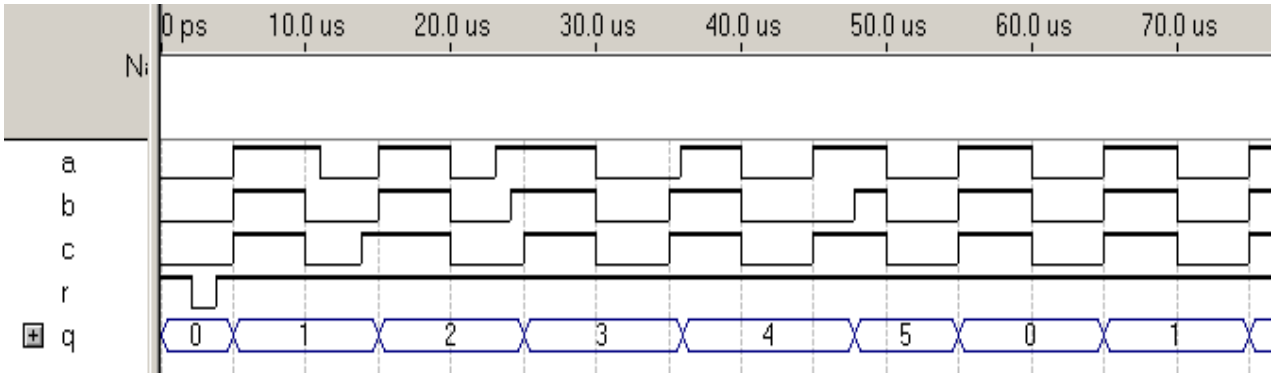

Рисунок 2.4 – Временные диаграммы работы делителя на 5

Декларативная часть позволяет описать объекты, используемые только внутри блока, которому она принадлежит. В декларативной части блока описываются любые типы объектов, которые могут присутствовать в декларативной части архитектурного тела. Все объекты, видимые внутри того архитектурного тела, к которому принадлежит блок, видимы внутри блока. Сами описания блоков располагаются внутри архитектурного тела.

Блок может содержать внутри себя другие блоки. Объекты, которые описаны в декларативной части блока, видны только в этом блоке.

Вложение блоков друг в друга позволяет создавать иерархические структуры моделей.

Рассмотрим пример.

```
ENTITY blck IS
    PORT (a,b,c : IN bit;
          q,q1   : OUT bit);
END blck;
ARCHITECTURE structure OF blck IS
 BEGIN
   block1: BLOCK
     SIGNAL qb : bit;
      BEGIN
        qb <= a AND b;
        q <= qb;
          block2: BLOCK
                SIGNAL q : bit;
                      BEGIN
                        q <= qb AND c;
                        q1 <= q;
          END BLOCK block2;
   END BLOCK block1;
END ARCHITECTURE structure;
```

В приведенном примере применены вложенные блоки, а также пояснена особенность видимости сигналов. Сигналы с именем q объявлены в интерфейсе устройства и во втором блоке. Однако конфликты отсутствуют (смотри рисунок 2.5), поскольку внутренний сигнал q виден только во втором блоке.

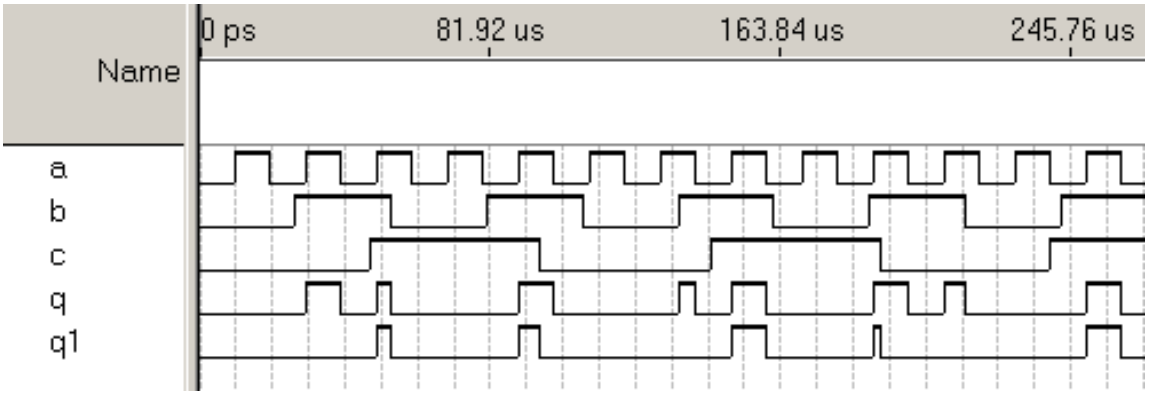

Рисунок 2.5 – Временные диаграммы работы элемента И

Конфигурирование блоков подобно конфигурированию компонентов. Конфигурационная декларация для архитектурного тела, содержащего блоки, должна отражать структуру блоков.

Конфигурационная декларация в этом случае имеет следующий синтаксис:

```
configuration configuration_name of entity_name is
   for (architecture_name | block_label)
   {block_configuration |
      for component_specification [binding_indication;] [block_configuration]
      end for;}
end for
end [configuration] [configuration_name];
```

Из синтаксиса декларации видно, что секции block_configuration могут вкладываться друг в друга.

Рассмотрим в качестве примера описание устройства с применением реконфигурируемого компонента: триггера, который в зависимости от архитектурного тела может выполнять функции T- или D-триггера.

Описание триггера имеет вид.

```
ENTITY dt_blck IS
  PORT ( dt, c, r  : IN bit; : BUFFER bit);
END dt_blck;
ARCHITECTURE d_arch OF dt_blck IS
   BEGIN
    PROCESS (c)
       BEGIN
          IF r = '0' THEN  q <= '0';
            ELSIF (c'EVENT AND c= '1') THEN q <= dt;
               ELSE null;
           END IF;
       END PROCESS;
END d_arch;
ARCHITECTURE t_arch OF dt_blck IS
    BEGIN
```

```
        PROCESS (c)
          BEGIN
            IF r = '0' THEN  q <= '0';
                 ELSIF (c'EVENT AND c= '1') THEN
                   IF dt = '1' THEN q <= NOT q;
                        ELSE null;
                   END IF;
             END IF;
          END PROCESS;
END t_arch;
CONFIGURATION cfg1 OF dt_blck IS -- Configuration Declaration
        FOR t_arch END FOR;
END cfg1;
```

Описание устройства.

```
ENTITY conf_blck IS
  PORT (d, clk, res : IN bit;
        qO, qO1       : OUT bit);
END conf_blck;
ARCHITECTURE arch OF conf_blck IS
   COMPONENT dt_blck
         PORT(dt, c, r : IN bit; q: OUT bit);
   END COMPONENT;
          BEGIN
            level1 : BLOCK
               PORT (dt1, c1, r1: IN bit; q1: OUT bit);
               PORT MAP (dt1 => d, c1 => clk, r1 => res, q1 => qO);
                  BEGIN
                     comp1: dt_blck PORT MAP (dt1, c1, r1, q1);
           END BLOCK level1;
           level2 : BLOCK
                 PORT (dt2, c2, r2: IN bit; q2: OUT bit);
                 PORT MAP (dt2 => d, c2 => clk, r2 => res, q2 => qO1);
                    BEGIN
                       comp2: dt_blck PORT MAP (dt2, c2, r2, q2);
          END BLOCK level2;
END ARCHITECTURE arch;
CONFIGURATION full OF conf_blck IS
  FOR arch
    FOR level1
      FOR comp1: dt_blck
              use entity work.dt_blck(d_arch);
      END FOR;
         END FOR;
         FOR level2
```

```
    FOR comp2: dt_blck
        use entity work.dt_blck(t_arch);
    END FOR;
  END FOR;
 END FOR;
END CONFIGURATION full;
```

Конфигурационной декларацией в первом блоке триггер переведен в режим D-триггера, а во втором – в режим T-триггера, что подтверждается временными диаграммами на рисунке 6.

2.2 Способы использования компонентов пользователя в САПР Quartus

САПР Quartus позволяет создавать компоненты пользователя и подключать их несколькими способами. Выбор конкретного способа зависит от дальнейшего использования компонента в других проектах, необходимости его модификации, а также необходимости существования нескольких одноименных компонентов.

Рассмотрим в качестве примера подключение компонента пользователя, который преобразует десятичную цифру в семисегментный код. Описание сущности и архитектуры компонента на языке VHDL имеет вид:

```
ENTITY dec_to_7seg IS
   PORT( digit : IN INTEGER RANGE 0 TO 15;
         segment : OUT BIT_VECTOR (6 DOWNTO 0) );
END dec_to_7seg;

ARCHITECTURE dec_to_7seg_arh OF dec_to_7seg IS
BEGIN
   WITH digit SELECT
      segment <= "0111111" WHEN 0,
                 "0000110" WHEN 1,
                 "1011011" WHEN 2,
                 "1001111" WHEN 3,
                 "1100110" WHEN 4,
                 "1101101" WHEN 5,
                 "1111101" WHEN 6,
                 "0000111" WHEN 7,
                 "1111111" WHEN 8,
                 "1100111" WHEN 9,
                 "0000000" WHEN OTHERS;
END dec_to_7seg_arh;
```

Компонент может быть использован при описании устройств, в которых применяются статические и динамические семисегментные индикаторы.

Разработаем с использованием различных способов подключения компонента простейшее устройство, которое осуществляет счёт десятичных цифр с определённой частотой и преобразовывает их в семисегментный код. Переход к следующей цифре осуществляется по переднему фронту тактового сигнала, поступающего на вход устройства. После цифры 9 счёт начинается с 0. Временные диаграммы работы устройства приведены на рисунке 2.6.

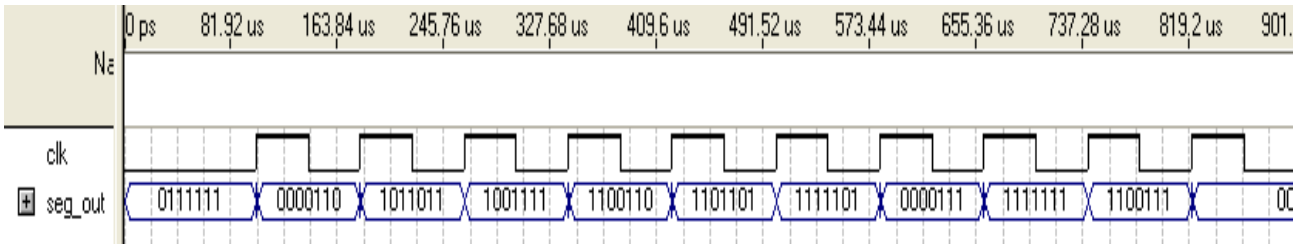

Рисунок 2.6 – Временные диаграммы работы преобразователя кодов

2.2.1 Описание и декларация компонента в одном VHDL-файле

Создавать устройства необходимо в следующей последовательности:

1 создать проект в САПР QUARTUS 2 (File → New Project Wizard);

2 создать файл VHDL (File → New → VHDL File). Поместить в файл описание компонента;

3 описать в этом файле сущность и архитектуру устройства;

4 в архитектуре устройства выполнить декларацию компонента;

5 сохранить файл (File → Save). При этом имя файла должно совпадать с названием сущности устройства.

6 выполнить компиляцию проекта.

Пример описания устройства.

```
ENTITY dec_to_7seg IS
      PORT( digit : IN INTEGER RANGE 0 TO 15;
            segment : OUT BIT_VECTOR (6 DOWNTO 0) );
END dec_to_7seg;
ARCHITECTURE dec_to_7seg_arh OF dec_to_7seg IS
  BEGIN
    WITH digit SELECT
      segment <= "0111111" WHEN 0,
                 "0000110" WHEN 1,
                 "1011011" WHEN 2,
                 "1001111" WHEN 3,
                 "1100110" WHEN 4,
                 "1101101" WHEN 5,
                 "1111101" WHEN 6,
                 "0000111" WHEN 7,
                 "1111111" WHEN 8,
                 "1100111" WHEN 9,
                 "0000000" WHEN OTHERS;
```

```
END dec_to_7seg_arh;

ENTITY spos1 IS
   PORT(clk : IN BIT;
        seg_out : OUT BIT_VECTOR (0 TO 6));
END spos1;
ARCHITECTURE arch OF spos1 IS
   COMPONENT dec_to_7seg
      PORT( digit       : IN INTEGER RANGE 0 TO 15;
            segment   : OUT BIT_VECTOR (6 DOWNTO 0));
   END COMPONENT dec_to_7seg;
      SIGNAL count: INTEGER RANGE 0 TO 15;
           BEGIN
            comp: dec_to_7seg PORT MAP (count, seg_out);
            PROCESS (clk)
                 BEGIN
                      IF (clk'EVENT and clk='1') THEN
                           count <= count + 1;
                      END IF;
           END PROCESS;
END arch;
```

В окне сообщений на вкладке Processing появятся сообщения:

```
Info: Found 4 design units, including 2 entities, in source file spos1.vhd
    Info: Found design unit 1: dec_to_7seg-dec_to_7seg_arh
    Info: Found design unit 2: spos1-arch
    Info: Found entity 1: dec_to_7seg
    Info: Found entity 2: spos1
Info: Elaborating entity "spos1" for the top level hierarchy
Info: Elaborating entity "dec_to_7seg" for hierarchy "dec_to_7seg:comp"
```

Таким образом, САПР Quartus определил четыре модуля проектирования (design units). Сущностью верхнего уровня является сущность spos1.

Параметры проекта:

– количество логических элементов – 11.

– размер исходного кода проекта – 1239 Байт.

Преимущества:

− работа с единственным исходным файлом в проекте.

Недостатки:

− невозможно использовать компонент в других проектах;

− большой объем исходного файла;

− компонент необходимо декларировать в архитектуре.

2.2.2 Описание компонента в отдельном VHDL-файле, расположенном в каталоге с проектом

Единственным отличием этого способа от предыдущего состоит в том, что описание компонента (сущность и архитектура) выносится в отдельный файл.

Создавать устройства необходимо в следующей последовательности:

1 создать проект в САПР Quartus (File → New Project Wizard);

2 создать файл VHDL (File → New → VHDL File). Поместить в файл описание компонента, которое включает в себя сущность и архитектуру;

3 сохранить файл (File → Save). При этом имя файла должно совпадать с названием сущности компонента;

4 создать ещё один файл VHDL (File → New → VHDL File). Описать в файле сущность и архитектуру устройства;

5 в архитектуре устройства выполнить декларацию и подключение компонента.

6 сохранить файл (File → Save). При этом имя файла должно совпадать с названием сущности устройства.

7 выполнить компиляцию проекта.

Рассмотрим пример.

Описание компонента.

```
ENTITY dec_to_7seg IS
   PORT( digit : IN INTEGER RANGE 0 TO 15;
            segment : OUT BIT_VECTOR (6 DOWNTO 0) );
END dec_to_7seg;
ARCHITECTURE dec_to_7seg_arh OF dec_to_7seg IS
BEGIN
   WITH digit SELECT
      segment <=
               "0111111" WHEN 0,
              "0000110" WHEN 1,
              "1011011" WHEN 2,
              "1001111" WHEN 3,
              "1100110" WHEN 4,
              "1101101" WHEN 5,
              "1111101" WHEN 6,
              "0000111" WHEN 7,
              "1111111" WHEN 8,
              "1100111" WHEN 9,
              "0000000" WHEN OTHERS;
END dec_to_7seg_arh;
```

Описание устройства.

```
ENTITY spos2 IS
   PORT(clk : IN BIT;
```

```
            seg_out   : OUT BIT_VECTOR (0 TO 6));
    END spos2;
    ARCHITECTURE arch OF spos2 IS
      COMPONENT dec_to_7seg
        PORT( digit : IN INTEGER RANGE 0 TO 15;
              segment: OUT BIT_VECTOR (6 DOWNTO 0));
      END COMPONENT dec_to_7seg;
         SIGNAL count: INTEGER RANGE 0 TO 15;
              BEGIN
                comp: dec_to_7seg PORT MAP (count, seg_out);
                PROCESS (clk)
                     BEGIN
                          IF (clk'EVENT and clk='1') THEN
                               count <= count + 1;
                          END IF;
              END PROCESS;
    END arch;
```

В окне сообщений на вкладке Processing появятся сообщения:

```
Info: Found 2 design units, including 1 entities, in source file dec_to_7seg.vhd
    Info: Found design unit 1: dec_to_7seg-dec_to_7seg_arh
    Info: Found entity 1: dec_to_7seg
Info: Found 2 design units, including 1 entities, in source file spos2.vhd
    Info: Found design unit 1: spos2-arch
    Info: Found entity 1: spos2
Info: Elaborating entity "spos2" for the top level hierarchy
Info: Elaborating entity "dec_to_7seg" for hierarchy "dec_to_7seg:comp"
```

Таким образом, САПР Quartus определил четыре модуля проектирования (design units). Сущностью верхнего уровня является сущность spos2.

Параметры проекта:

– количество логических элементов – 11.

– размер исходного кода dec_to_7seg – 667 байт, а spos2 – 570 байт.

Преимущества:

− отдельный файл с описанием компонента можно использовать в других проектах;

− каждый файл отвечает за описание сущности и архитектуры, что упрощает читаемость кода.

Недостатки:

− для использования файла описания компонента в другом проекте его нужно скопировать, а это приводит к нерациональному дублированию кода. Кроме того, при необходимости внести изменения в описание компонента, необходимо их вносить во все скопированные файлы;

− компонент нужно декларировать в архитектуре.

2.2.3 Описание компонента в отдельном VHDL-файле, расположенном вне каталога с проектом

Для устранения основного недостатка второго способа, файл с описанием компонента можно поместить в отдельный каталог и, в дальнейшем, при создании новых проектов указывать этот файл.

Создавать устройства необходимо в следующей последовательности:

1 создать проект в Quartus (File → New Project Wizard);

2 создать каталог вне проекта и поместить в него файл с описанием компонента;

3 создать файл VHDL(File → New → VHDL File). Описать в файле сущность и архитектуру устройства;

4 выполнить в архитектуре устройства декларацию компонента;

8 сохранить файл (File → Save). При этом имя файла должно совпадать с названием сущности устройства;

5 зайти в Project → Add/Remove Files in Project… → Вкладка Files и выбрать файл с описанием компонента;

6 выполнить компиляцию проекта.

Рассмотрим пример разработки устройства.

Предварительно занесем описание компонента dec_to_7seg во вновь созданную директорию libs и подключим его к проекту командой Project → Add/Remove Files in Project… → Вкладка Files. Список проектных файлов показан на рисунке 2.7.

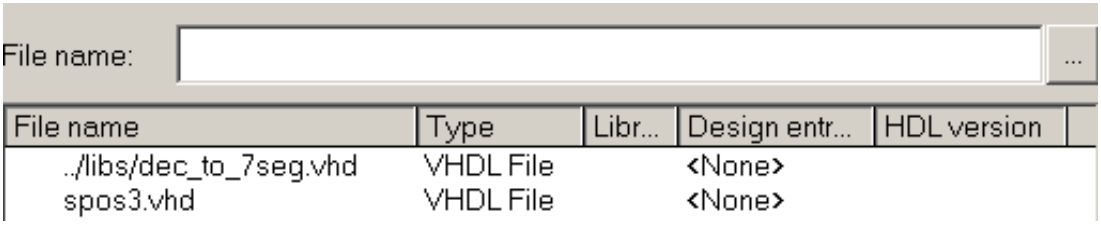

Рисунок 2.7 – Список проектных файлов проекта spos3

Затем создадим в проекте файл spos3.vhd, содержание которого совпадает с файлом spos2.vhd за исключением имени проекта.

Параметры проекта:

– количество логических элементов – 11.

– размер исходного кода dec_to_7seg – 667 байт, а spos3 – 570 байт.

Преимущества:

− можно использовать файл с описанием компонента во многих проектах.

− файл описания можно изменять из любого проекта после его подключения. При этом его содержание будет меняться во всех остальных проектах.

− Недостатки:

− компонент необходимо в архитектуре.

2.2.4 Описание компонента в пакете, расположенном в каталоге с проектом

Создавать устройства необходимо в следующей последовательности:

1 создать проект в Quartus (File → New Project Wizard);

2 создать каталог вне проекта и поместить в него файл с описанием компонента;

3 создать пакет в проекте (File → New → VHDL File). В пакете выполнить декларацию компонента;

4 сохранить файл (File → Save). При этом имя файла должно совпадать с именем пакета;

5 создать файл VHDL в проекте (File → New → VHDL File). Описать в файле сущность и архитектуру устройства. В архитектуре устройства выполнить декларацию компонента не следует. При этом имя файла должно совпадать с названием сущности устройства;

6 подключить с помощью опции в Project → Add/Remove Files in Project… → Вкладка Files файлы с описанием пакета и компонента;

7 подключить пакета добавления в файл описания предложений:

LIBRARY work;

USE work.<имя пакета>.ALL;

Вместо ALL можно указать только имя компонента, тогда остальные декларации из пакета не будут подключены;

8 выполнить компиляцию проекта.

Рассмотрим пример разработки устройства.

Предварительно занесем описание компонента dec_to_7seg и пакета pack_dec_to_7seg во вновь созданную директорию libs и подключим их к проекту командой Project → Add/Remove Files in Project… → Вкладка Files. Список проектных файлов показан на рисунке 2.8.

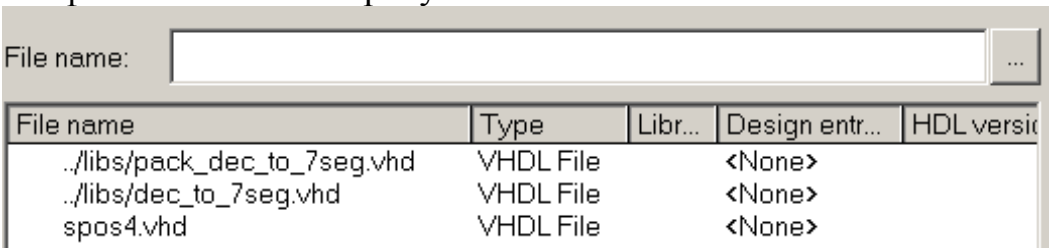

Рисунок 2.8 – Список проектных файлов проекта spos4

Описание пакета.

```
PACKAGE pack_dec_to_7seg IS
   COMPONENT dec_to_7seg
      PORT(digit          : IN INTEGER RANGE 0 TO 15;
                segment   : OUT BIT_VECTOR (6 DOWNTO 0));
   END COMPONENT dec_to_7seg;
END PACKAGE;
```

Описание устройства.

```
LIBRARY work;
USE work.pack_dec_to_7seg. ALL;
ENTITY spos4 IS
   PORT(clk : IN BIT;
            seg_out   : OUT BIT_VECTOR (0 TO 6));
END spos4;
ARCHITECTURE arch OF spos4 IS
        SIGNAL count: INTEGER RANGE 0 TO 15;
                BEGIN
                  comp: dec_to_7seg PORT MAP (count, seg_out);
                        PROCESS (clk)
                            BEGIN
                                IF (clk'EVENT and clk='1') THEN
                                count <= count + 1;
                                 END IF;
                        END PROCESS;
END arch;
```

В окне сообщений на вкладке Processing появятся сообщения:

Info: Found 1 design units, including 0 entities, in source file ../libs/pack_dec_to_7seg.vhd

Info: Found design unit 1: pack_dec_to_7seg

Info: Found 2 design units, including 1 entities, in source file ../libs/dec_to_7seg.vhd

Info: Found design unit 1: dec_to_7seg-dec_to_7seg_arh

Info: Found entity 1: dec_to_7seg

Info: Found 2 design units, including 1 entities, in source file spos4.vhd

Info: Found design unit 1: spos4-arch

Info: Found entity 1: spos4

Info: Elaborating entity "spos4" for the top level hierarchy

Info: Elaborating entity "dec_to_7seg" for hierarchy "dec_to_7seg:comp"

Таким образом, САПР Quartus определил пять модулей проектирования (design units). Сущностью верхнего уровня является сущность spos4.

Параметры проекта:

– количество логических элементов - 11.

– размер исходного кода dec_to_7seg - 667 байт, pack_dec_to_7seg – 209 байт, а spos4 - 455 байт. Размер файла проекта уменьшился, поскольку отсутствует декларация компонента.

Преимущества:

– в архитектуре главной сущности уже не нужно выполнять декларацию компонента.

– пакет позволяет разграничить области имен. Это означает, что можно создать два разноимённых пакета, но с одноименными компонентами внутри. Это удобно при необходимости подключения компонентов с одинаковой функциональностью, но разной реализацией.

Недостатки:
– необходимость создания пакета;
– следует к файлу проекта подключать два файла.

2.2.5 Описание компонента в одном файле с декларацией пакета

Для уменьшения количество подключаемых файлов следует поместить описание самого компонента вместе с декларацией пакета.

Создавать устройства необходимо в следующей последовательности:

1 создать проект в Quartus (File → New Project Wizard).

2 создать каталог вне проекта. Поместить в этот файл декларацию пакета, а также с описание сущности и архитектуры компонента.

3 создать файл проекта VHDL (File → New → VHDL File). Описать в файле сущность и архитектуру устройства. В архитектуре устройства выполнить декларацию компонента не нужно. Сохранить файл (File → Save). При этом имя файла должно совпадать с названием сущности устройства.

4 Выполнить подключение пакета добавлением в файл проекта предложений:

LIBRARY work;
USE work.<имя пакета>.ALL;

Вместо ALL можно указать только имя компонента, тогда остальные декларации из пакета не будут подключены.

5 зайти в Project → Add/Remove Files in Project… → Вкладка Files. Подключитьь файл с декларацией пакета

6 выполнить компиляцию проекта.

Рассмотрим пример разработки устройства.

Создадим описание пакета.

```
PACKAGE pack1_dec_to_7seg IS
   COMPONENT dec_to_7seg
      PORT(digit          : IN INTEGER RANGE 0 TO 15;
           segment        : OUT BIT_VECTOR (6 DOWNTO 0));
   END COMPONENT dec_to_7seg;
END PACKAGE;
ENTITY dec_to_7seg IS
   PORT(   digit          : IN INTEGER RANGE 0 TO 15;
           segment        : OUT BIT_VECTOR (6 DOWNTO 0) );
END dec_to_7seg;
ARCHITECTURE dec_to_7seg_arh OF dec_to_7seg IS
BEGIN
   WITH digit SELECT
      segment <= "0111111" WHEN 0,
            "0000110" WHEN 1,
            "1011011" WHEN 2,
            "1001111" WHEN 3,
            "1100110" WHEN 4,
```

```
                "1101101" WHEN 5,
                "1111101" WHEN 6,
                "0000111" WHEN 7,
                "1111111" WHEN 8,
                "1100111" WHEN 9,
                "0000000" WHEN OTHERS;
END dec_to_7seg_arh;
```

Занесем описание компонента пакета pack1_dec_to_7seg во вновь созданную директорию libs и подключим их к проекту командой Project → Add/Remove Files in Project… → Вкладка Files. Список проектных файлов показан на рисунке 2.9.

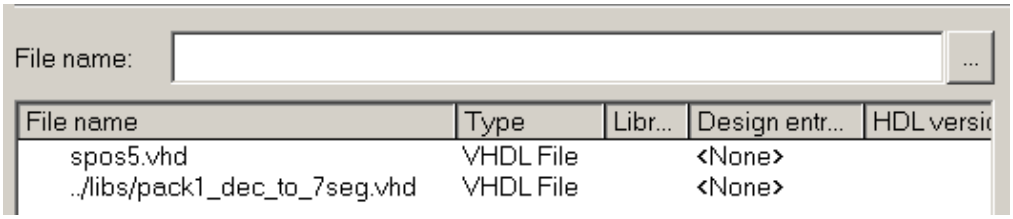

Рисунок 2.9 – Список проектных файлов проекта spos4

Создадим файл проекта spos5.vhd, который совпадает с файлом spos4.vhd за исключением подключения пакета. Предложение Use Clause имеет вид:

USE work.pack1_dec_to_7seg.hex_to_7seg;

При компиляции проекта получены следующие сообщения:

Info: Found 2 design units, including 1 entities, in source file spos5.vhd

Info: Found design unit 1: spos5-arch

Info: Found entity 1: spos5

Info: Found 3 design units, including 1 entities, in source file ../libs/pack1_dec_to_7seg.vhd

Info: Found design unit 1: pack1_dec_to_7seg

Info: Found design unit 2: dec_to_7seg-dec_to_7seg_arh

Info: Found entity 1: dec_to_7seg

Info: Elaborating entity "spos5" for the top level hierarchy

Info: Elaborating entity "dec_to_7seg" for hierarchy "dec_to_7seg:comp"

Видно, что при анализе файла pack1_dec_to_7seg.vhd Quartus нашел три модуля проектирования: декларация пакета, описание архитектуры компонента, описание сущности компонента.

Параметры проекта:

– количество логических элементов – 11.

– размер исходного кода – pack1_dec_to_7seg.vhd – 874 байта, а spos5 – 456 байт.

Преимущество состоит в простоте применения компонента в других проектах.

2.2.6 Подключение компонента, декларированного в пакете, без применения предложения Use Clause

Вместо применения предложения Use Clause тип компонента указывают в формате **<имя библиотеки>.<имя компонента>**.

Создавать устройства необходимо в следующей последовательности:

1 создать проект в Quartus (File → New Project Wizard);

2 создать новый каталог вне проекта и поместить в него файл с декларацией пакета, а также с описанием сущности и архитектуры компонента.

3 создать файл VHDL в проекте (File → New → VHDL File). Описать в файле сущность и архитектуру устройства. В архитектуре устройства выполнить декларацию компонента не нужно. При этом имя файла должно совпадать с названием сущности устройства.

4 при объявлении компонента указать его тип в формате <имя библиотеки>.<имя компонента>

5 зайти в Project → Add/Remove Files in Project… → Вкладка Files и выбрать файл с декларацией пакета и описанием компонента.

6 выполнить компиляцию проекта.

Рассмотрим пример разработки устройства.

Применим пакет pac1_dec_to_7seg.vhd, который находится в каталоге libs.

Описание устройства имеет вид.

```
ENTITY spos6 IS
   PORT(clk : IN BIT;
        seg_out : OUT BIT_VECTOR (0 TO 6));
END spos6;
ARCHITECTURE arch OF spos6 IS
        SIGNAL count: INTEGER RANGE 0 TO 15;
                BEGIN
                  comp: work.dec_to_7seg PORT MAP (count, seg_out);
                     PROCESS (clk)
                        BEGIN
            IF (clk'EVENT and clk='1') THEN
                                count <= count + 1;
            END IF;
          END PROCESS;
END arch;
```

Параметры проекта:

количество логических элементов – 11.

размер исходного кода – pack1_dec_to_7seg.vhd – 874 байта, а spos6 – 411 байт.

Преимущества:

– минимальный размер файла проекта;

– содержимое пакета не отображается на всю область видимости VHDL-файла, что позволяет исключить проблему одинаковых идентификаторов.

Недостаток – при многоразовом использовании компонента следует всё время указывать имя библиотеки.

2.2.7 Описание пакета, компонента и устройства в одном файле

Создавать устройства необходимо в следующей последовательности:

1 создать проект в Quartus (File → New Project Wizard);

2 создать файл VHDL в проекте (File → New → VHDL File). Описать пакет и в нём выполнить декларацию компонента.

3 подключить пакет в созданном файле добавлением в файл предложений:

LIBRARY <имя библиотеки>;

USE <имя библиотеки>.<имя пакета>.ALL;

4 описать в файле сущность и архитектуру устройства;

5 сохранить файл (File → Save). При этом имя файла должно соответствовать названию сущности устройства.

6 выполнить компиляцию проекта.

Пример описания устройства.

```
PACKAGE pack1_dec_to_7seg IS
   COMPONENT dec_to_7seg
      PORT(digit          : IN INTEGER RANGE 0 TO 15;
           segment        : OUT BIT_VECTOR (6 DOWNTO 0));
   END COMPONENT dec_to_7seg;
END PACKAGE;

ENTITY dec_to_7seg IS
   PORT( digit : IN INTEGER RANGE 0 TO 15;
         segment : OUT BIT_VECTOR (6 DOWNTO 0) );
END dec_to_7seg;
ARCHITECTURE dec_to_7seg_arh OF dec_to_7seg IS
BEGIN
   WITH digit SELECT
      segment <= "0111111" WHEN 0,
                 "0000110" WHEN 1,
                 "1011011" WHEN 2,
                 "1001111" WHEN 3,
                 "1100110" WHEN 4,
                 "1101101" WHEN 5,
                 "1111101" WHEN 6,
                 "0000111" WHEN 7,
                 "1111111" WHEN 8,
                 "1100111" WHEN 9,
                 "0000000" WHEN OTHERS;
END dec_to_7seg_arh;
```

```
LIBRARY work;
USE work.pack1_dec_to_7seg. ALL;
ENTITY spos7 IS
   PORT(clk         : IN BIT;
            seg_out   : OUT BIT_VECTOR (0 TO 6));
END spos7;
ARCHITECTURE arch OF spos7 IS
        SIGNAL count: INTEGER RANGE 0 TO 15;
                BEGIN
                  comp: dec_to_7seg PORT MAP (count, seg_out);
                  PROCESS (clk)
                        BEGIN
                          IF (clk'EVENT and clk='1') THEN
                             count <= count + 1;
                           END IF
                END PROCESS;
END arch;
```

Параметры проекта:

– количество логических элементов – 11.

– размер исходного кода – spos7 - 1334 байт.

Преимущества:

– при одноразовом использовании компонента минимальны усилия по управлению файлами;

– пакет даёт возможность создать разные пространства имён.

Недостатки:

− максимальный размер исходного файла проекта;

− трудность использования файла для других проектов.

2.2.8 Подключение компонента из библиотеки

В Quartus библиотека представляет собой каталог, в котором находятся файлы проектных модулей (design units).

Библиотеки бывают двух видов: библиотки проекта (Project Libraries) – подключаемые единожды в один проект и глобальные – подключаемые во все проекты, которые создаются в Quartus. Поэтому настройки библиотеки проекта хранятся в файле qsf, который входит в проект, а настройки глобальных библиотек в файле системы (quartus2.ini).

Для подключения библиотеки необходимо указать путь к определёному каталогу. По умолчанию все созданные пакеты помещаются в псевдо-библиотеку work, независимо от того, в каких каталогах размещены файлы с декларацией этих пакетов.

Создавать устройства необходимо в следующей последовательности:

1 создать проект в Quartus (File → New Project Wizard).

2 создать каталог библиотеки проекта. В нем создать файл с декларацией пакета, а также с описанием сущности и архитектуры компонента.

3 создать файл VHDL в проекте (File → New → VHDL File). Описать в файле сущность и архитектуру устройства. В архитектуре устройства декларацию компонента не проводить. Сохранить файл (File → Save). При этом имя файла должно совпадать с названием сущности устройства.

4 Выполнить подключение пакета путём добавления в файл предложений:

LIBRARY <имя библиотеки>;

USE <имя библиотеки>.<имя пакета>.ALL;

5 зайти в Project → Add/Remove Files in Project… → Вкладка Libraries. Выбрать каталог библиотеки, нажать Add, OK;

6 зайти в Project → Add/Remove Files in Project… → Вкладка Files и указать имя файл с декларацией пакета и описанием сущности и архитектуры компонента.

7 Выполнить компиляцию проекта.

Пример описания устройства.

Файл с декларацией пакета, а также с описанием сущности и архитектуры компонента помещен в каталог libs.

```
LIBRARY libs;
USE libs.pack1_dec_to_7seg. ALL;
ENTITY spos8 IS
   PORT(clk       : IN BIT;
           seg_out : OUT BIT_VECTOR (0 TO 6));
END spos8;
ARCHITECTURE arch OF spos8 IS
        SIGNAL count: INTEGER RANGE 0 TO 15;
                BEGIN
                   comp: dec_to_7seg PORT MAP (count, seg_out);
                      PROCESS (clk)
                         BEGIN
                            IF (clk'EVENT and clk='1') THEN
                               count <= count + 1;
                            END IF;
                       END PROCESS;
END arch;
```

Списки библиотек и проектных файлов показаны на рисунке 2.10 и 2.11.

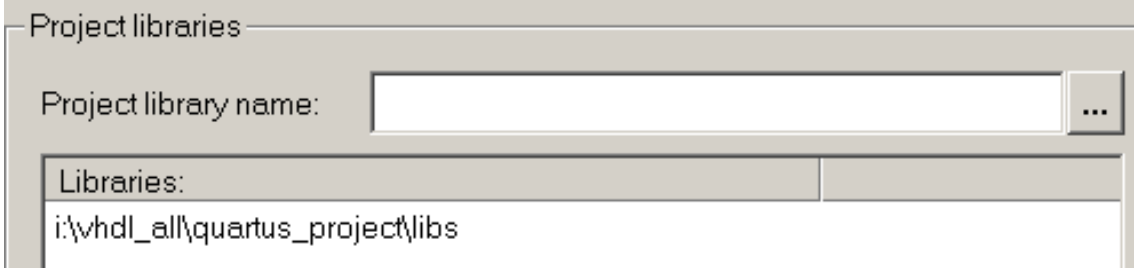

Рисунок 2.10 – Список библиотек проекта spos8

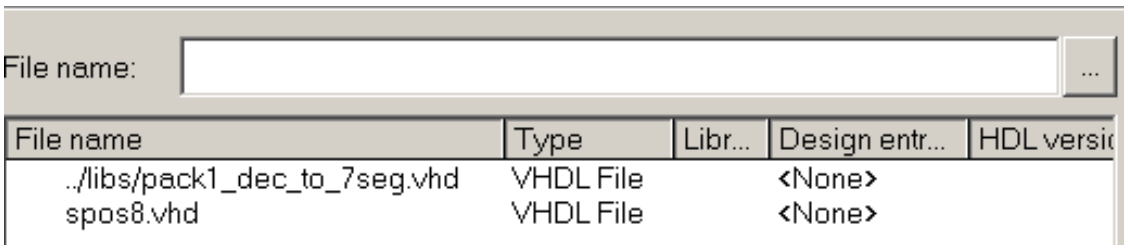

Рисунок 2.11 – Список проектных файлов проекта spos8

Параметры проекта:

– количество логических элементов – 11.

– размер исходного кода – pack1_dec_to_7seg.vhd – 874 байта, spos8 – 454 байта.

Преимущества:

– библиотеки позволяют объединить несколько проектных файлов в единое целое и разграничить области видимости имен с другим библиотеками на уровне пакетов;

– независимость пути к проектным файлам от дерева файловой системы. Позволяет намного проще переносить библиотеки между системами.

Недостатки:

- необходимость дополнительно указывать путь к библиотеке.

2.2.9 Сравнительный анализ способов подключения компонентов

Сравним рассмотренные способы по сложности реализации и объему файлов проекта (смотри таблицу 2.1).

Таблица 2.1 – Объем файлов проекта и сложность реализации способов

Номер	Объем файлов проекта (байт)		Суммарный объем файлов	Сложность реализации
1	**spos1**	**1239**	1239	11 элементов
2	dec_to_7seg	667	1237	
	spos2	**570**		
3	dec_to_7seg	667	1237	
	spos3	**570**		
4	pack_dec_to_7seg	209	1331	
	dec_to_7seg	667		
	spos4	**455**		
5	pack1_dec_to_7seg	874	1330	
	spos5	**456**		
6	pack1_dec_to_7seg	874	1285	
	spos6	**411**		
7	**spos7**	**1334**	1334	
8	pack1_dec_to_7seg	874	1328	
	spos8	**454**		

Следует стремиться подключить компонент самым простым способом, то есть при использовании компонента определить только необходимые настройки. Выполнять же декларацию компонента в проекте, в котором он используется, не является удачным решением. В этом случае размер кода больше, возрастает вероятность допустить ошибку. Кроме того, при изменении интерфейсной части компонента придётся изменять ещё и декларацию компонента, причём в каждом проекте, который использует данный компонент. Намного удобнее поместить декларацию в пакет, и в дальнейшем просто подключать этот пакет в каждый проект.

В пакете можно сгруппировать несколько компонентов со схожей функциональностью для того, чтобы логически систематизировать набор компонентов. Таким образом, когда разработчику необходимо решить определённую задачу, он воспользуется несколькими компонентами с нужными функциями, для решения другой похожей задачи ему понадобится этот же набор компонентов. Для проектов, решающих другой круг задач, разработчику понадобится другой набор компонентов, то есть другой пакет. К тому же в пакетах можно декларировать не только компоненты, но и функции, константы, типы.

При разработке иногда возникают случаи, когда необходимо определить несколько компонентов с одинаковыми именами. В этом случае необходимо разделить (разграничить) пространства имён. Выполнить такое разграничение можно либо при применении пакетов (одноименные компоненты подключаются в разноимённые пакеты), либо при применении библиотек. Библиотеки помогают разграничить пространство имён на уровне пакетов, то есть одноимённые пакеты могут использоваться в разноимённых библиотеках.

2.3 Предложения Assertion Statement и Report Statement

В процессе разработки устройств на языках описания аппаратуры перед проверкой работоспособности на реальном устройстве целесообразно выполнить предварительное тестирование написанной программы на виртуальном устройстве, называемом симулятором.

Симулятор (от англ. simulate – моделировать) – специальное программное обеспечение (существующее независимо, либо входящее в состав САПР), способное воспроизвести работу реального устройства. Процесс такого воспроизведения называется моделированием. В результате процесса моделирования разработчик может оценивать изменение состояний выходов микросхемы в зависимости от заданных состояний входов.

Результат моделирования традиционно представляется в виде временных диаграмм, на которых отображены зависимости изменения выходных сигналов от входных сигналов. Однако такое представление результатов имеет ряд недостатков: отсутствует возможность просмотреть результаты вычисления определённых выражений или переменных; нет возможности оценить разветвленные алгоритмы программы; кроме того, нет возможности автоматически проконтролировать значения выходных сигналов, и это приходится делать каждый раз

“визуально”. При многократном тестировании устройств такой традиционный подход является неэффективным и некачественным.

Дополнительным подходом к тестированию является программная проверка результатов работы устройства и отображение их в журнале симулятора.

Журнал симулятора – это поток текстовой информации, который генерируется симулятором в процессе выполнения моделирования. Обычно отображение такого потока реализуется путём отображения текстовой информации в так называемой консоли – виртуальной области дисплея, доступной пользователю для взаимодействия с потоком. Через консоль пользователь может не только просматривать информацию, но и вводить команды для управления процессом моделирования (например, загрузка скомпилированных файлов в симулятор, запуск процесса моделирования). Вместе с консолью текстовый поток может помещаться в файл, который используют для сохранения результатов моделирования.

Итак, в основе дополнительного подхода лежит механизм передачи данных из исходного VHDL-кода в текстовый поток журнала с помощью специальных предложений утверждения и отчета.

2.3.1 Оператор утверждения (Assertion statement)

Предложение Assertion Statement проверяет заданное условие и генерирует отчет об ошибке, если это условие не выполнилось. Формат предложения:

assertion_statement ::= [label:] assertion ;
assertion ::=
assert condition
[**report** expression]
[**severity** expression]

Условие (condition) в обязательном предложении **assert** должно быть выражение типа Boolean. Необязательное предложение **report** должно включать выражение типа STRING, которое определяет сообщение, выводимое в результирующий отчет. Если предложение **report** не указать, то в отчет будет выведена строка: "Assertion violation." Необязательное предложение **severity** указывает степень серьёзности утверждения. Выражение может принимать одно из четырех значений:

NOTE – сообщение (не важное утверждение);
WARNING – предупреждение;
ERROR – ошибка;
FAILURE – отказ.

Если предложение **severity** не указано, то автоматически устанавливается степень серьёзности ERROR.

При невыполнении условия в предложении **assert** будут выполнены предложения **report** и **severity** (если они присутствуют) и на их основе составлено сообщение об ошибке, которое будет выведено в отчет.

Предложение Assert Statement может использоваться как в качестве параллельного предложения, так и в качестве последовательного.

Сообщение об ошибке состоит, по крайней мере, из:

– указания, что данное сообщение сгенерировано предложением утверждения;

– имени модуля проекта, содержащего это утверждение;

– значения строки сообщения;

– значения уровня серьёзности.

2.3.2 Предложение Report statement

С помощью этого предложения можно выполнять безусловную генерацию сообщений в отчёте. Формат предложения:

report_statement ::=
[label :]
report expression
[**severity** expression] ;

Обязательное предложение **report** должно включать выражение типа STRING, которое определяет сообщение, выводимое в результирующий отчет. Необязательное предложение **severity** указывает степень серьёзности утверждения. Может принимать такие же значения, как и в операторе утверждения. Если предложение не указано, то автоматически устанавливается степень серьёзности NOTE.

Предложение Report Statement может использоваться в САПР Quartus только в качестве последовательного предложения.

Сообщение об ошибке состоит, по крайней мере, из:

– указания, что данное сообщение сгенерировано предложением отчета;

– имени модуля проекта, содержащего это утверждение;

– значения строки сообщения;

– значения уровня серьёзности.

2.3.3 Применение предложений Assertion Statement и Report Statement в различных САПР

Продемонстрируем применение утверждений и отчётов в языке VHDL на примере простого дешифратора, описание которого имеет вид:

```
ENTITY decod IS
  PORT (x          : IN bit_vector (0 TO 1);
        Y, Z       : OUT bit_vector(0 TO 3));      BEGIN
 ASSERT false REPORT "Compilation entity" SEVERITY WARNING;
 ASSERT false SEVERITY WARNING;
END decod;
ARCHITECTURE desh_arch OF decod IS
  FUNCTION case_f(vxod:IN bit_vector (0 TO 1))RETURN bit_vector IS
    VARIABLE case_out:bit_vector(0 TO 3);
```

```
        BEGIN
          CASE vxod IS
                WHEN "00"=>case_out:="1000";
                WHEN "10"=>case_out:="0100";
                WHEN "01"=>case_out:="0010";
                WHEN "11"=>case_out:="0001";
                END CASE;
                RETURN case_out;
         END case_f;
       BEGIN
             ASSERT false REPORT "Concurrent Statement"
                          SEVERITY WARNING;
          PROCESS (X)
            BEGIN
             ASSERT false REPORT "Sequential Statement"
                         SEVERITY WARNING;
             REPORT "Calculation z" SEVERITY WARNING;
                   z<=case_f(X);
             REPORT "Calculation y" SEVERITY WARNING;
                   y<=case_f(X);
          END PROCESS;
    END desh_arch;
```

В САПР Quartus сообщения в консоль выводятся только при компиляции описания цифрового устройства. Сообщения имеют вид:

Warning (10651): VHDL Assertion Statement at decod.vhd(5): assertion is false - report "Compilation entity" (WARNING)

Warning (10651): VHDL Assertion Statement at decod.vhd(6): assertion is false - report Assertion violation (WARNING)

Warning (10651): VHDL Assertion Statement at decod.vhd(21): assertion is false - report "Concurrent Statement" (WARNING)

Warning (10651): VHDL Assertion Statement at decod.vhd(24): assertion is false - report "Sequential Statement" (WARNING)

Warning (10636): VHDL Report Statement at decod.vhd(25): "Calculation z" (WARNING)

Warning (10636): VHDL Report Statement at decod.vhd(27): "Calculation y" (WARNING)

Сообщения позволяют контролировать этап, на котором в данный момент находится компиляция описания, что важно при создании сложных цифровых устройств.

При моделировании в САПР Quartus сообщения по предложениям утверждения и отчеты в консоль и файлы отчеты не выводятся. Поэтому рассмотрим особенности работы в САПР ModelSim Altera Starter Edition 6.5e на следующем примере синхронного RS-триггера:

```
ENTITY rs_ff IS
```

```
   PORT (R,S, clk  : in bit;
              Q, nQ     : buffer bit);
END ENTITY rs_ff;
ARCHITECTURE rsff OF rs_ff IS
  BEGIN
    PROCESS(clk)
      BEGIN
        IF (clk'EVENT AND clk = '1') THEN
          IF (R = '1' AND S ='0') THEN  Q <= '0'; nQ <= '1';
            ELSIF (R = '0' AND S ='1') THEN  Q <= '1'; nQ <= '0';
               ELSIF (R = '0' AND S ='0') THEN Q <= Q; nQ <= nQ ;
                  ELSIF (R = '1' AND S ='1') THEN Q <= '0'; nQ <= '0';
               ASSERT (R AND S)='0' REPORT "R = S= 1" SEVERITY note;
          END IF;
        END IF;
   END process;
END rsff;
```

При моделировании в консоли появится сообщение о запрещенной комбинации (смотри рисунок 2.12), что позволяет довольно таки просто определять критические ситуации.

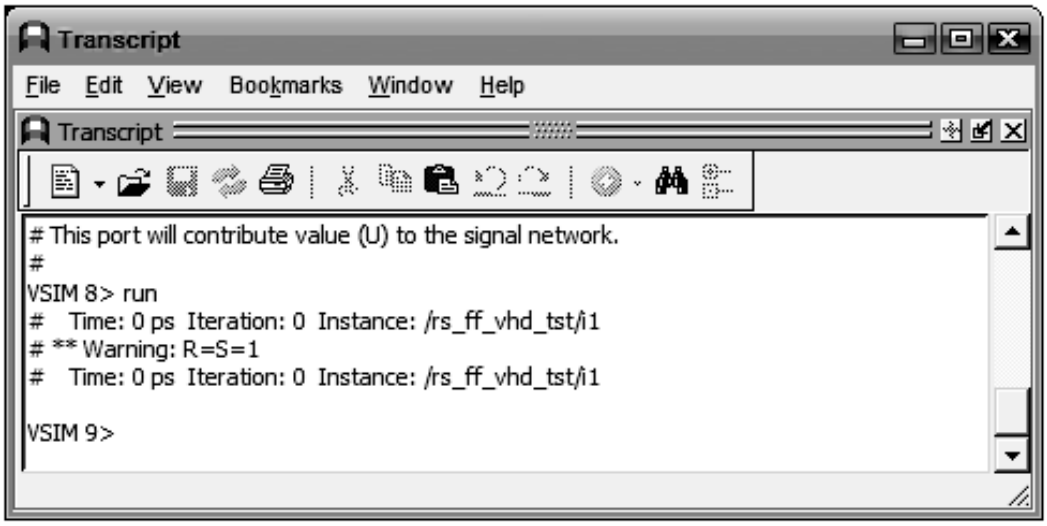

Рисунок 2.12 – Сообщения в консоли САПР ModelSim

3 РАСЧЁТНО-ГРАФИЧЕСКАЯ РАБОТА

3.1 Цель и порядок выполнения расчётно-графической работы

Расчётно-графическая работа является одним из видов учебной работы, в процессе выполнения которой студент для решения поставленных задач применяет на практике те знания, которые были приобретены им при освоении теоретических основ учебной дисциплины. В результате он закрепляет полученные ранее знания, расширяет их, а также приобретает навыки и умения практического решения типовых задач учебной дисциплины.

Темой расчётно-графической работы по дисциплине “Технологии проектирования компьютерных систем” является поведенческое описание схемотехнических устройств на языке VHDL.

Целью выполнения расчётно-графической работы является приобретение практических навыков поведенческого описания сложных схемотехнических устройств средствами языка VHDL.

Выполнение расчётно-графической работы включает следующие обязательные этапы:

– анализ известных методов построения заданного устройства;

– разработку схемы алгоритма функционирования устройства;

– разработку поведенческого описания на языке VHDL последовательного или комбинационного устройства согласно варианту задания;

– анализ работоспособности описанного устройства и оценка соответствия заданным требованиям по временным диаграммам;

– исправление выявленных ошибок, а также внесение изменений (при необходимости);

– выводы о проделанной работе;

– оформление отчёта о выполнении расчётно-графической работы.

Каждое задание для выполнения расчётно-графической работы состоит из нескольких уровней. Каждый последующий уровень имеет более высокую сложность и оценивается более высоким количеством баллов по сравнению с предыдущими.

Выполнение задания более высокого уровня подразумевает обязательное выполнение всех заданий более низких уровней,

Практическая часть расчётно-графической работы выполняется при помощи инструментальных средств САПР Quartus II от компании Altera. Рекомендуется использовать номера версий указанного САПР, начиная с 9.0 и выше.

3.2 Требования к отчёту о выполнении расчётно-графической работы

Отчёт о выполнении расчётно-графической работы оформляется на стандартных листах формата А4 в соответствии с требованиями СОККР-2001 кафедры ИКС.

Отчёт о выполнении расчётно-графической работы должен содержать следующие обязательные структурные элементы:

– титульный лист с указанием темы расчётно-графической работы;

– анализ известных методов построения заданного устройства (объёмом не менее 4-х листов формата А4), раскрывающие особенности внутреннего построения и функционирования, как самого устройства, указанного в задании, так и класса схемотехнических устройств, к которому оно относится;

– описание варианта задания, обязательно содержащее таблицу задания, состоящую из тех уровней, которые были выполнены непосредственно;

– разработка схемы алгоритма функционирования устройств (с необходимым текстовым описанием);

– разработка поведенческого описания устройства на языке VHDL, представленное с необходимым форматированием и содержащее подробные комментарии (смотри пункт 3.4.4);

– анализ работоспособности описанного устройства и оценка соответствия заданным требованиям (этот анализ проводится на основании полученных временных диаграмм функционирования устройства), содержащий необходимые пояснения, причём на приведенных временных диаграммах должны быть представлены все комбинации входных сигналов, как указано в задании;

– выводы (здесь необходимо произвести подведение итогов проделанной работы, оценку полученных результатов, указать особенности применённых предложений языка VHDL).

Каждый из структурных элементов нумеруется, как отдельный раздел и начинается с нового листа.

3.3 Варианты заданий

Ниже, в таблицах 3.1 – 3.15, приводятся варианты заданий для выполнения расчётно-графической работы по дисциплине “Технологии проектирования компьютерных систем”.

Задание №1

Разработать поведенческое описание синхронного счётчика.

Таблица 3.1 – Задание №1

Уровень I (0 – 5 баллов)	
Интерфейс устройства	Счётчик содержит асинхронный вход сброса Res, асинхронный вход установки Set, счётный вход Clk, вход разрешения счёта Enable, выход Result[...]. Разрядность выходного сигнала Result[...] счётчика задается как статический параметр, значение которого должно быть не менее 4.
Поведение	Если на входе Res удерживается низкий уровень сигнала, на выходе счётчика устанавливается низкий уровень сигнала (независимо от состояния остальных входов). Когда на входе Set удерживается низкий уровень сигнала, на выходе счётчика устанавливается высокий уровень сигнала (независимо от состояния остальных входов). Если же на обоих входах Res и Set одновременно удерживается низкий уровень, выход счётчика переходит в Z-состояние. При условии, что на входах Res, Set, Enable удерживается высокий уровень сигнала, счётчик осуществляет насчёт импульсов, поступающих на его счётный вход по переднему фронту импульса. В случае, когда на вход Enable подается низкий уровень, счётчик прекращает насчёт импульсов и хранит текущее значение.
Уровень II (5 – 10 баллов)	
Интерфейс устройства	Дополнительно счётчик содержит вход Dir, который управляет направлением счёта.
Поведение	Если на входе Dir удерживается низкий уровень сигнала, счётчик работает как суммирующий. Когда на входе Dir удерживается высокий уровень сигнала, счётчик работает как вычитающий.
Уровень III (10 – 15 баллов)	
Интерфейс устройства	Дополнительно счётчик содержит вход данных Data[...], разрядность которого равна разрядности самого счётчика и вход загрузки данных Load.
Поведение	Если на входе Load удерживается высокий уровень, в счётчик загружается значение, установленное на входе Data[...] независимо от состояния входа Enable. Низкий уровень на входе Load никак не влияет на работу счётчика.

Задание №2

Разработать поведенческое описание параллельного сумматора беззнаковых целых чисел.

Таблица 3.2 – Задание №2

Уровень I (0 – 5 баллов)	
Интерфейс устройства	Сумматор содержит асинхронный вход сброса Res, вход тактирования Clk, вход разрешения счёта Enable, входы Operand1[...] и Operand2[...] слагаемых, выход результата Result[...], выход переноса Carryout. Разрядность входных операндов Operand1[...], Operand2[...] и выхода Result[...] сумматора задается как статический параметр, значение которого должно быть не менее 4.
Поведение	Если на входе Res удерживается низкий уровень сигнала, на выходах сумматора устанавливается низкий уровень сигнала (независимо от состояния остальных входов). При условии, что на входах Res и Enable удерживается высокий уровень сигнала, сумматор осуществляет сложение чисел, поступающих на его входы, по переднему фронту импульса тактирования Clk. В случае, когда на вход Enable подается низкий уровень, сумматор не производит сложение и хранит результат последней операции.
Уровень II (5 – 10 баллов)	
Интерфейс устройства	Дополнительно сумматор содержит вход Check и выход Parity контроля чётности вычисленного результата.
Поведение	Когда на входе Check удерживается низкий уровень, на выходе Parity устанавливается высокий (если количество единичных разрядов в полученном результате нечётно) либо низкий (если количество единиц чётно) уровень. Высокий уровень на входе Check никак не влияет на работу сумматора, а выход Parity в этом случае переводится в Z-состояние.
Уровень III (10 – 15 баллов)	
Интерфейс устройства	Дополнительно сумматор содержит выход сравнения операндов Equal[...], разрядностью 2.
Поведение	После выполнения операции сложения на этом выходе устанавливаются следующие значения: - 00, если оба входных операнда одинаковые; - 10, если значение на входе Operand1[...] больше; - 01, если значение на входе Operand2[...] больше;

Задание №3

Разработать поведенческое описание параллельного вычитателя беззнаковых целых чисел.

Таблица 3.3 – Задание №3

Уровень I (0 – 5 баллов)	
Интерфейс устройства	Вычитатель содержит асинхронный вход сброса Res, вход тактирования Clk, вход разрешения счёта Enable, входы уменьшаемого Operand1[...] и вычитаемого Operand2[...], выход результата Result[...], выход заёма Borrow. Разрядность входных операндов Operand1[...], Operand2[...] и выхода Result[...] вычитателя задаётся как статический параметр, значение которого должно быть не менее 4.
Поведение	Если на входе Res удерживается низкий уровень сигнала, на выходах вычитателя устанавливается низкий уровень сигнала (независимо от состояния остальных входов). При условии, что на входах Res и Enable удерживается высокий уровень сигнала, вычитатель осуществляет вычитание чисел, поступающих на его входы, по заднему фронту импульса тактирования Clk. В случае, когда на вход Enable подается низкий уровень, вычитатель не производит вычитание и хранит результат последней операции.
Уровень II (5 – 10 баллов)	
Интерфейс устройства	Дополнительно вычитатель содержит вход Check и выход Parity контроля чётности вычисленного результата.
Поведение	Когда на входе Check удерживается высокий уровень, на выходе Parity устанавливается высокий (если количество единичных разрядов в полученном результате нечётно) либо низкий (если количество единиц чётно) уровень. Низкий уровень на входе Check никак не влияет на работу сумматора, а выход Parity в этом случае переводится в Z-состояние.
Уровень III (10 – 15 баллов)	
Интерфейс устройства	Дополнительно вычитатель содержит выход сравнения операндов Equal[...], разрядностью 2.
Поведение	После выполнения операции вычитания на этом выходе устанавливаются следующие значения: - 11, если оба входных операнда одинаковые; - 10, если значение на входе Operand1[...] меньше; - 01, если значение на входе Operand2[...] меньше;

Задание №4

Разработать поведенческое описание цифрового компаратора беззнаковых целых чисел.

Таблица 3.4 – Задание №4

Уровень I (0 – 5 баллов)	
Интерфейс устройства	Компаратор содержит асинхронный вход сброса Res, вход тактирования Clk, вход разрешения сравнения Enable, входы операндов Operand1[...] и Operand2[...], выход условия равно Eq, выход условия первый больше Gr, выход условия первый меньше Le. Разрядность входных операндов Operand1[...] и Operand2[...] компаратора задаётся как статический параметр, значение которого должно быть не менее 4.
Поведение	Если на входе Res удерживается низкий уровень сигнала, на выходах компаратора устанавливается низкий уровень сигнала (независимо от состояния остальных входов). При условии, что на входах Res и Enable удерживается высокий уровень сигнала, компаратор осуществляет сравнение чисел, поступающих на его входы, по переднему фронту импульса Clk. В результате сравнения на выходе, для которого выполнилось условие сравнения, устанавливается высокий уровень сигнала. В случае, когда на вход Enable подается низкий уровень, все выходы компаратора переводятся в Z-состояние.
Уровень II (5 – 10 баллов)	
Интерфейс устройства	Дополнительно компаратор содержит вход инверсии операндов Inv.
Поведение	Если на входе Inv удерживается высокий уровень сигнала, данные в компаратор поступают без инверсии. Когда же на входе Inv удерживается низкий уровень сигнала, каждый бит данных инвертируется.
Уровень III (10 – 15 баллов)	
Интерфейс устройства	Дополнительно компаратор содержит выход анализа операндов Odd_Even[...], разрядностью 2.
Поведение	После выполнения операции сравнения на этом выходе устанавливаются следующие значения: - 00, если оба входных операнда чётные; - 11, если оба входных операнда нечётные; - 10, если значение на входе Operand1[...] нечётное; - 01, если значение на входе Operand2[...] нечётное;

Задание №5

Разработать поведенческое описание цифрового компаратора беззнаковых целых чисел.

Таблица 3.5 – Задание №5

Уровень I (0 – 5 баллов)	
Интерфейс устройства	Компаратор содержит асинхронный вход сброса Res, вход тактирования Clk, вход разрешения сравнения Enable, входы операндов Operand1[...] и Operand2[...], выход условия не равно Neq, выход условия первое больше или равно Greq, выход условия первое меньше или равно Leeq. Разрядность входных операндов Operand1[...] и Operand2[...] компаратора задаётся как статический параметр, значение которого должно быть не менее 4.
Поведение	Если на входе Res удерживается низкий уровень сигнала, на выходах компаратора устанавливается низкий уровень сигнала (независимо от состояния остальных входов). При условии, что на входах Res и Enable удерживается высокий уровень сигнала, компаратор осуществляет сравнение чисел, поступающих на его входы, по заднему фронту импульса Clk. В результате сравнения на выходе, для которого выполнилось условие сравнения, устанавливается высокий уровень сигнала. В случае, когда на вход Enable подается низкий уровень, все выходы компаратора переводятся в Z-состояние.
Уровень II (5 – 10 баллов)	
Интерфейс устройства	Дополнительно компаратор содержит вход инверсии операндов Inv.
Поведение	Если на входе Inv удерживается низкий уровень сигнала, данные в компаратор поступают без инверсии. Когда же на входе Inv удерживается высокий уровень сигнала, каждый бит данных инвертируется.
Уровень III (10 – 15 баллов)	
Интерфейс устройства	Дополнительно компаратор содержит выход анализа операндов Even_Odd[...], разрядностью 2.
Поведение	После выполнения операции сравнения на этом выходе устанавливаются следующие значения: - 00, если оба входных операнда чётные; - 11, если оба входных операнда нечётные; - 10, если значение на входе Operand1[...] нечётное; - 01, если значение на входе Operand2[...] нечётное;

Задание №6

Разработать поведенческое описание параллельного регистра.

Таблица 3.6 – Задание №6

Уровень I (0 – 5 баллов)	
Интерфейс устройства	Регистр содержит асинхронный вход сброса Res, асинхронный вход установки Set, вход разрешения записи Wr, вход разрешения чтения Rd, вход данных Data[...], выход данных Result[...]. Разрядность входного Data[...] и выходного Result[...] сигнала данных регистра задаётся как статический параметр, значение которого должно быть не менее 4.
Поведение	Если на входе Res удерживается низкий уровень сигнала, на выходе регистра устанавливается низкий уровень сигнала (независимо от состояния остальных входов). Когда на входе Set удерживается низкий уровень сигнала, на выходе регистра устанавливается высокий уровень сигнала (независимо от состояния остальных входов). В случае, когда на входах Res и Set одновременно удерживается низкий уровень, выход регистра переходит в Z-состояние. При условии, что на входах Res и Set удерживается высокий уровень сигнала, запись данных в регистр осуществляется по переднему фронту импульса на входе Wr. Если же на входах Res, Set и Rd удерживается высокий уровень сигнала, осуществляется чтение данных из регистра. Если на входе Rd удерживается низкий уровень сигнала, и на каждом из выходов регистра устанавливается низкий уровень сигнала.
Уровень II (5 – 10 баллов)	
Интерфейс устройства	Дополнительно регистр содержит вход Inv, который инвертирует каждый бит входных данных перед записью.
Поведение	Если на входе Inv удерживается высокий уровень сигнала, данные в регистр записываются без инверсии. Когда же на входе Inv удерживается низкий уровень сигнала, каждый бит данных инвертируется.
Уровень III (10 – 15 баллов)	
Интерфейс устройства	Дополнительно регистр содержит вход Check и выход Parity контроля чётности хранимых данных.
Поведение	Если на входе Check удерживается высокий уровень, на выходе Parity устанавливается высокий (если количество единичных разрядов для хранимых в регистре данных нечётно) либо низкий (если количество единиц чётно) уровень. Низкий уровень на входе Check никак не влияет на работу регистра, а выход Parity в этом случае переводится в Z-состояние.

Задание №7

Разработать поведенческое описание преобразователя двоично-десятичного кода в семисегментный.

Таблица 3.7 – Задание №7

Уровень I (0 – 5 баллов)	
Интерфейс устройства	Преобразователь содержит асинхронный вход сброса Res, вход разрешения преобразования Enable, вход загрузки данных Load, вход данных BinCode[...], выход 7SegCode[...]. Разрядность входного сигнала BinCode[...] задается как статический параметр, значение которого равно 4. Разрядность выходного сигнала 7SegCode[...] задается как статический параметр, значение которого равно 7.
Поведение	Если на входе Res удерживается низкий уровень сигнала, на выходе преобразователя устанавливается низкий уровень сигнала (независимо от состояния остальных входов). При условии, что на входах Res и Enable удерживается высокий уровень сигнала, преобразователь осуществляет преобразование двоичного кода, поступающего на его вход данных по переднему фронту импульса тактирования Load. В случае, когда на вход Enable подается низкий уровень, все выходы преобразователя переводятся в Z-состояние.
Уровень II (5 – 10 баллов)	
Интерфейс устройства	Дополнительно преобразователь содержит вход Inv, который инвертирует каждый бит выходного кода.
Поведение	Если на входе Inv удерживается высокий уровень сигнала, выходной код устанавливается без инверсии. Когда же на входе Inv удерживается низкий уровень сигнала, каждый бит выходного кода инвертируется.
Уровень III (10 – 15 баллов)	
Интерфейс устройства	Дополнительно преобразователь содержит вход Check и выход Parity контроля чётности семисегментного кода.
Поведение	Если на входе Check удерживается низкий уровень, на выходе Parity устанавливается высокий (если количество единичных разрядов в выходном коде нечётно) либо низкий (если количество единиц чётно) уровень. Высокий уровень на входе Check никак не влияет на работу преобразователя, а выход Parity в этом случае переводится в Z-состояние.

Задание №8

Разработать поведенческое описание последовательного регистра.

Таблица 3.8 – Задание №8

Уровень I (0 – 5 баллов)	
Интерфейс устройства	Регистр содержит асинхронный вход сброса Res, асинхронный вход установки Set, вход разрешения записи Wr, вход разрешения чтения Rd, вход тактирования Clk, вход данных Data, выход данных Result.
Поведение	Если на входе Res удерживается низкий уровень сигнала, на выходе регистра устанавливается низкий уровень сигнала (независимо от состояния остальных входов). Когда на входе Set удерживается низкий уровень сигнала, на выходе регистра устанавливается высокий уровень сигнала (независимо от состояния остальных входов). В случае, когда на входах Res и Set одновременно удерживается низкий уровень, выход регистра переходит в Z-состояние. При условии, что на входах Res, Set и Wr удерживается высокий уровень сигнала, запись данных в регистр осуществляется по заднему фронту импульса на входе Clk. При этом, если на входе Wr удерживается низкий уровень, в регистр записывается 0. Если же на входах Res, Set и Rd удерживается высокий уровень сигнала, осуществляется чтение данных из регистра. Если на входе Rd удерживается низкий уровень сигнала, на выходе регистра устанавливается низкий уровень сигнала.
Уровень II (5 – 10 баллов)	
Интерфейс устройства	Дополнительно регистр содержит вход Inv, который инвертирует каждый бит входных данных перед записью в регистр.
Поведение	Если на входе Inv удерживается низкий уровень сигнала, данные в регистр записываются без инверсии. Когда же на входе Inv удерживается высокий уровень сигнала, каждый бит данных инвертируется.
Уровень III (10 – 15 баллов)	
Интерфейс устройства	Дополнительно регистр содержит вход Check и выход Parity контроля чётности хранимых данных.
Поведение	Если на входе Check удерживается низкий уровень, на выходе Parity устанавливается высокий (если количество единичных разрядов для хранимых в регистре данных нечётно) либо низкий (если количество единиц чётно) уровень. Высокий уровень на входе Check никак не влияет на работу регистра, а выход Parity в этом случае переводится в Z-состояние.

Задание №9

Разработать поведенческое описание преобразователя двоичного кода в код Грея.

Таблица 3.9 – Задание №9

Уровень I (0 – 5 баллов)	
Интерфейс устройства	Преобразователь содержит асинхронный вход установки Set, вход разрешения преобразования Enable, вход загрузки данных Load, вход данных BinCode[...], выход GreyCode[...]. Разрядность входного сигнала BinCode[...] и выходного GreyCode[...] задается как статический параметр, значение которого равно 5.
Поведение	Если на входе Set удерживается низкий уровень сигнала, на выходе преобразователя устанавливается высокий уровень сигнала (независимо от состояния остальных входов). При условии, что на входах Set и Enable удерживается высокий уровень сигнала, преобразователь осуществляет преобразование двоичного кода, поступающего на его вход данных по заднему фронту импульса тактирования Load. В случае, когда на вход Enable подается низкий уровень, все выходы преобразователя переводятся в Z-состояние.
Уровень II (5 – 10 баллов)	
Интерфейс устройства	Преобразователь должен дополнительно содержать вход Inv, который инвертирует каждый бит выходного кода.
Поведение	Если на входе Inv удерживается высокий уровень сигнала, выходной код устанавливается без инверсии. Когда же на входе Inv удерживается низкий уровень сигнала, каждый бит выходного кода инвертируется.
Уровень III (10 – 15 баллов)	
Интерфейс устройства	Дополнительно преобразователь содержит вход Check и выход Parity контроля чётности кода Грея.
Поведение	Если на входе Check удерживается низкий уровень, на выходе Parity устанавливается высокий (если количество единичных разрядов в выходном коде нечётно) либо низкий (если количество единиц чётно) уровень. Высокий уровень на входе Check никак не влияет на работу преобразователя, а выход Parity в этом случае переводится в Z-состояние.

Задание №10

Разработать поведенческое описание делителя с произвольным постоянным коэффициентом деления, построенного по методу принудительного задания коэффициента пересчета в вычитающем счётчике.

Таблица 3.10 – Задание №10

Уровень I (0 – 10 баллов)	
Интерфейс устройства	Делитель содержит асинхронный вход сброса Res, асинхронный вход установки Set, счётный вход Clk, выход Dev. Значение коэффициента деления N задается как статический параметр и должно быть не менее 11.
Поведение	Если на входе Res удерживается низкий уровень сигнала, на выходе делителя устанавливается низкий уровень сигнала (независимо от состояния остальных входов). Когда на входе Set удерживается низкий уровень сигнала, на выходе делителя устанавливается высокий уровень сигнала (независимо от состояния остальных входов). При условии, что на входах Res и Set одновременно удерживается низкий уровень, выход делителя переходит в Z-состояние. В случае, когда на входах Res и Set удерживается высокий уровень сигнала, делитель осуществляет насчёт импульсов, поступающих на его счётный вход, по переднему фронту, формируя один импульс на выходе на каждые N входных импульсов. Длительность высокого уровня в сформированном импульсе составляет половину периода тактового сигнала.
Уровень II (10 – 15 баллов)	
Интерфейс устройства	Дополнительно делитель содержит вход установки коэффициента деления Koef[...], разрядность которого должна быть не менее 4, и вход загрузки коэффициента деления Load.
Поведение	Если на входе Load удерживается высокий уровень, в делитель загружается значение коэффициента деления, получаемого со входа Koeff[...], при этом на выходе делителя удерживается низкий уровень сигнала и он не осуществляет насчёт импульсов. Низкий уровень на входе Load никак не влияет на работу делителя.

Задание №11

Разработать поведенческое описание делителя с произвольным постоянным коэффициентом деления, построенного по методу исключения лишних состояний.

Таблица 3.11 – Задание №11

Уровень I (0 – 10 баллов)	
Интерфейс устройства	Делитель содержит асинхронный вход сброса Res, асинхронный вход установки Set, счётный вход Clk, выход Dev. Значение коэффициента деления N задается как статический параметр и должно быть не менее 9.
Поведение	Если на входе Res удерживается низкий уровень сигнала, на выходе делителя устанавливается низкий уровень сигнала (независимо от состояния остальных входов). Когда на входе Set удерживается низкий уровень сигнала, на выходе делителя устанавливается высокий уровень сигнала (независимо от состояния остальных входов). При условии, что на входах Res и Set одновременно удерживается низкий уровень, выход делителя переходит в Z-состояние. В случае, когда на входах Res и Set удерживается высокий уровень сигнала, делитель осуществляет насчёт импульсов, поступающих на его счётный вход, по переднему фронту, формируя один импульс на выходе на каждые N входных импульсов. Длительность высокого уровня в сформированном импульсе составляет половину периода тактового сигнала.
Уровень II (10 – 15 баллов)	
Интерфейс устройства	Дополнительно делитель содержит вход установки коэффициента деления Koef[...], разрядность которого должна быть не менее 4, и вход загрузки коэффициента деления Load.
Поведение	Если на входе Load удерживается низкий уровень, в делитель загружается значение коэффициента деления, получаемого со входа Koeff[...], при этом на выходе делителя удерживается высокий уровень сигнала и он не осуществляет насчёт импульсов. Высокий уровень на входе Load никак не влияет на работу делителя.

Задание №12

Разработать поведенческое описание параллельно-последовательного регистра.

Таблица 3.12 – Задание №12

Уровень I (0 – 5 баллов)	
Интерфейс устройства	Регистр содержит асинхронные входы сброса Res и установки Set, входы разрешения записи Wr и чтения Rd, вход тактирования Clk, вход данных Data[...], выход данных Result. Разрядность входных данных Data[...] задаётся как статический параметр, значение которого должно быть не менее 4.
Поведение	Если на входе Res удерживается низкий уровень сигнала, на выходе регистра устанавливается низкий уровень сигнала (независимо от состояния остальных входов). Когда на входе Set удерживается низкий уровень сигнала, на выходе регистра устанавливается высокий уровень сигнала (независимо от состояния остальных входов). В случае, когда на входах Res и Set одновременно удерживается низкий уровень, выход регистра переходит в Z-состояние. При условии, что на входах Res, Set и Wr удерживается высокий уровень сигнала, запись данных в регистр осуществляется по переднему фронту импульса на входе Clk. Если на входе Wr удерживается низкий уровень, в регистр записывается 0. Если же на входах Res, Set и Rd удерживается высокий уровень сигнала, то осуществляется чтение данных из регистра. Если на входе Rd удерживается низкий уровень сигнала, на выходе регистра устанавливается низкий уровень сигнала.
Уровень II (5 – 10 баллов)	
Интерфейс устройства	Дополнительно регистр содержит вход Inv, который инвертирует каждый бит входных данных перед записью.
Поведение	Если на входе Inv удерживается высокий уровень сигнала, данные в регистр записываются без инверсии. Когда же на входе Inv удерживается низкий уровень сигнала, каждый бит данных инвертируется.
Уровень III (10 – 15 баллов)	
Интерфейс устройства	Дополнительно регистр содержит вход Check и выход Parity контроля чётности хранимых данных.
Поведение	Если на входе Check удерживается высокий уровень, на выходе Parity устанавливается высокий (если количество единичных разрядов для хранимых в регистре данных нечётно) либо низкий (если количество единиц чётно) уровень. Низкий уровень на входе Check никак не влияет на работу регистра, а выход Parity в этом случае переводится в Z-состояние.

Задание №13

Разработать поведенческое описание последовательно-параллельного регистра.

Таблица 3.13 – Задание №13

Уровень I (0 – 5 баллов)	
Интерфейс устройства	Регистр содержит асинхронные входы сброса Res и установки Set, вход разрешения записи Wr и чтения Rd, вход тактирования Clk, вход данных Data, выход данных Result[...]. Разрядность выходных данных Result[...] задаётся как статический параметр, значение которого должно быть не менее 4.
Поведение	Если на входе Res удерживается низкий уровень сигнала, на выходе регистра устанавливается низкий уровень сигнала (независимо от состояния остальных входов). Когда на входе Set удерживается низкий уровень сигнала, на выходе регистра устанавливается высокий уровень сигнала (независимо от состояния остальных входов). В случае, когда на входах Res и Set одновременно удерживается низкий уровень, выход регистра переходит в Z-состояние. При условии, что на входах Res, Set и Wr удерживается высокий уровень сигнала, запись данных в регистр осуществляется по заднему фронту импульса на входе Clk. Если на входе Wr удерживается низкий уровень, в регистр записывается 0. Если же на входах Res, Set и Rd удерживается высокий уровень сигнала, осуществляется чтение данных из регистра. Если на входе Rd удерживается низкий уровень сигнала, на выходе регистра устанавливается низкий уровень сигнала.
Уровень II (5 – 10 баллов)	
Интерфейс устройства	Дополнительно регистр содержит вход Inv, который инвертирует каждый бит входных данных перед записью.
Поведение	Если на входе Inv удерживается низкий уровень сигнала, данные в регистр записываются без инверсии. Когда же на входе Inv удерживается высокий уровень сигнала, каждый бит данных инвертируется.
Уровень III (10 – 15 баллов)	
Интерфейс устройства	Дополнительно регистр содержит вход Check и выход Parity контроля чётности хранимых данных.
Поведение	Если на входе Check удерживается низкий уровень, на выходе Parity устанавливается высокий (если количество единичных разрядов для хранимых в регистре данных нечётно) либо низкий (если количество единиц чётно) уровень. Высокий уровень на входе Check никак не влияет на работу регистра, а выход Parity в этом случае переводится в Z-состояние.

Задание №14

Разработать поведенческое описание делителя с произвольным постоянным коэффициентом деления, построенного по методу искусственного принудительного насчёта импульсов в счётчике.

Таблица 3.14 – Задание №14

Уровень I (0 – 10 баллов)	
Интерфейс устройства	Делитель содержит асинхронный вход сброса Res, асинхронный вход установки Set, счётный вход Clk, выход Dev. Значение коэффициента деления N задается как статический параметр и должно быть не менее 10.
Поведение	Если на входе Res удерживается низкий уровень сигнала, на выходе делителя устанавливается низкий уровень сигнала (независимо от состояния остальных входов). Когда на входе Set удерживается низкий уровень сигнала, на выходе делителя устанавливается высокий уровень сигнала (независимо от состояния остальных входов). При условии, что Если на входах Res и Set одновременно удерживается низкий уровень, выход делителя переходит в Z-состояние. В случае, когда на входах Res и Set удерживается высокий уровень сигнала, делитель осуществляет насчёт импульсов, поступающих на его счётный вход, по заднему фронту, формируя один импульс на выходе на каждые N входных импульсов. Длительность высокого уровня в сформированном импульсе составляет половину периода тактового сигнала.
Уровень II (10 – 15 баллов)	
Интерфейс устройства	Дополнительно делитель содержит вход установки коэффициента деления Koeff[...], разрядность которого должна быть не менее 4, и вход загрузки коэффициента деления Load.
Поведение	Если на входе Load удерживается высокий уровень, в делитель загружается значение коэффициента деления, получаемого со входа Koeff[...], при этом на выходе делителя удерживается низкий уровень сигнала и он не осуществляет насчёт импульсов. Низкий уровень на входе Load никак не влияет на работу делителя.

Задание №15

Разработать поведенческое описание оперативного запоминающего устройства.

Таблица 3.15 – Задание №15

Уровень I (0 – 5 баллов)	
Интерфейс устройства	ОЗУ содержит шину адреса Addr[...], входную и выходную шины данных Data_in[...] и Data_out[...] соответственно, вход чтения/записи данных Rd_Wr, вход тактирования Clk. Разрядность шины адреса Addr [...] задается как статический параметр, значение которого не менее 5. Разрядность шины данных Data_in[...] и Data_out[...] задается как статический параметр, значение которого не менее 8.
Поведение	Если на входе Rd_Wr удерживается высокий уровень сигнала, запись данных, установленных на шине Data_in[...], осуществляется по переднему фронту тактового импульса Clk в ячейку, адрес которой установлен на шине Addr[...]. В случае, когда на входе Rd_Wr удерживается низкий уровень сигнала, чтение данных, установленных на шине Data_out[...], осуществляется по переднему фронту сигнала Clk из ячейки, адрес которой установлен на шине Addr[...].
Уровень II (5 – 10 баллов)	
Интерфейс устройства	ОЗУ дополнительно содержит асинхронный вход сброса Res и асинхронный вход установки Set.
Поведение	Если на входе Res удерживается низкий уровень сигнала, все биты всех ячеек ОЗУ принимают нулевое значение (независимо от состояния остальных входов). Когда же на входе Set удерживается низкий уровень сигнала, все биты всех ячеек ОЗУ принимают единичное значение (независимо от состояния остальных входов). При условии, что на входах Res и Set одновременно удерживается низкий уровень сигнала, содержимое ячеек ОЗУ не изменяется, а выход Data_out[...] переходит в Z-состояние.
Уровень III (10 – 15 баллов)	
Интерфейс устройства	ОЗУ дополнительно содержит вход Inv, который инвертирует каждый бит входных данных перед записью.
Поведение	Если на входе Inv удерживается низкий уровень сигнала, данные записываются без инверсии. Когда же на входе Inv удерживается высокий уровень сигнала, каждый бит входных данных инвертируется.

3.4 Пример выполнения расчётно-графической работы

Ниже представлен пример выполнения расчётно-графической работы по дисциплине “Технологии проектирования компьютерных систем”, в котором разрабатывается синхронный счётчик с параллельным переносом.

3.4.1 Анализ известных методов построения счётчиков

Счётчики. Счетчиками называют последовательные устройства, предназначенные для подсчета и запоминания числа импульсов, поданных в определенном временном интервале на его счетный вход.

Классифицируют счетчики по направлению счета, по способу организации схемы переноса, по коэффициенту пересчета, по наличию синхронизации переключения счетчиков.

По направлению счета:

а) суммирующие;

б) вычитающие;

в) реверсивные.

По способу организации схемы переноса:

а) счетчики с последовательным переносом;

б) счетчики со сквозным переносом;

в) счетчики с параллельным переносом;

г) счетчики с комбинированным переносом.

По коэффициенту пересчета:

а) двоичные;

б) двоично-десятичные;

в) с произвольным постоянным коэффициентом пересчета;

г) с переменным коэффициентом счета и т.д.

По наличию синхронизации переключения счетчиков:

а) асинхронные;

б) синхронные.

Суммирующие счетчики. У этих счетчиков отсутствует общая для всех разрядов синхронизация, и переход разрядов в новые состояния происходит последовательно разряд за разрядом, начиная от входного, на который поступают импульсы (смотри рисунок 3.1).

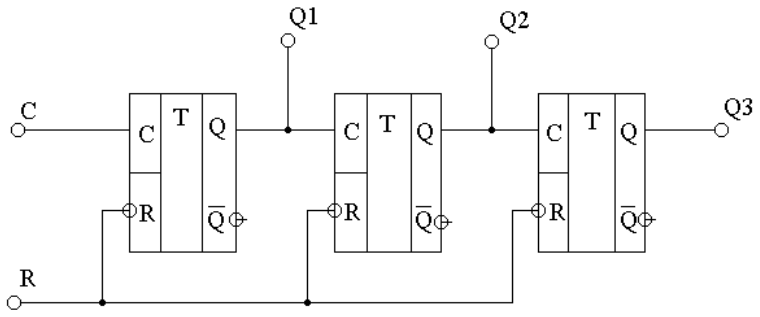

Рисунок 3.1 – Асинхронный суммирующий счётчик с последовательным переносом

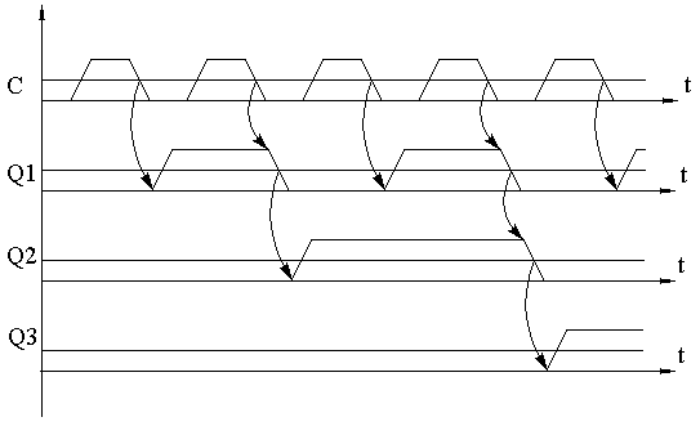

Рисунок 3.2 – Временная диаграмма работы асинхронного суммирующего счётчика с последовательным переносом

Из анализа рисунка 3.2 следует, что признаком смены состояний любого из разрядов счетчика является переключение предыдущего разряда (или входного сигнала) из состояния “1” в “0”, причем старший разряд переключается в “1”, если все предыдущие разряды переключаются из “1” в “0”.

Из временной диаграммы видно, что время установления счетчика $t_{уст.с}$ зависит от количества последовательно переключающихся разрядов и для N-разрядного счетчика изменяется в пределах:

$$t_{уст.т} \le t_{уст.с} \le N * t_{уст.т} = t_{уст.max} \qquad (3.1)$$

Максимальная частота следования импульсов fmax определяется предельной частотой подключения первого триггера.

Если требуется дешифрировать каждое состояние счетчика, то до подачи очередного счетного импульса все разряды должны установиться в новое состояние на время t_0, где t_0 – время дешифрации состояний счетчика. В этом случае максимально допустимая частота смены состояний $f_{c.max}$ определяется по наихудшему времени установления:

$$f_{c.max} = (t_0 + N * t_{уст.т})^{-1} \qquad (3.2)$$

Достоинства – минимальные затраты микросхем и минимум электрических связей.

Главный недостаток – низкое быстродействие.

Вычитающие счётчики. Вычитающий счетчик получается последовательным соединением инверсных выходов предыдущих разрядов со счетными входами последующих разрядов Т-триггера (смотри рисунок 3.3).

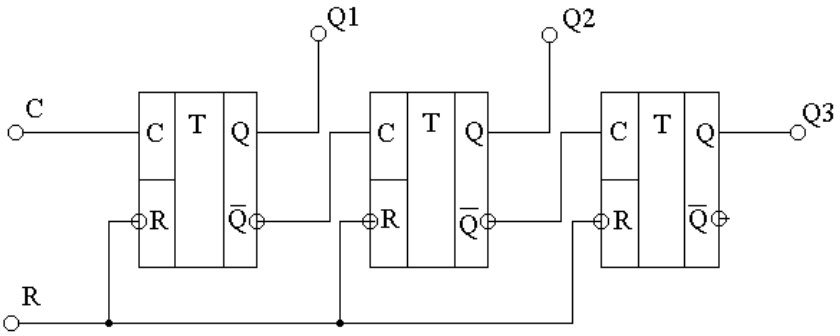

Рисунок 3.3 – Асинхронный вычитающий счётчик с последовательным переносом

Т-триггеры переключаются отрицательными фронтами входного сигнала или сигнала с инверсного выхода предыдущего триггера, что соответствует положительному фронту (смотри рисунок 3.4).

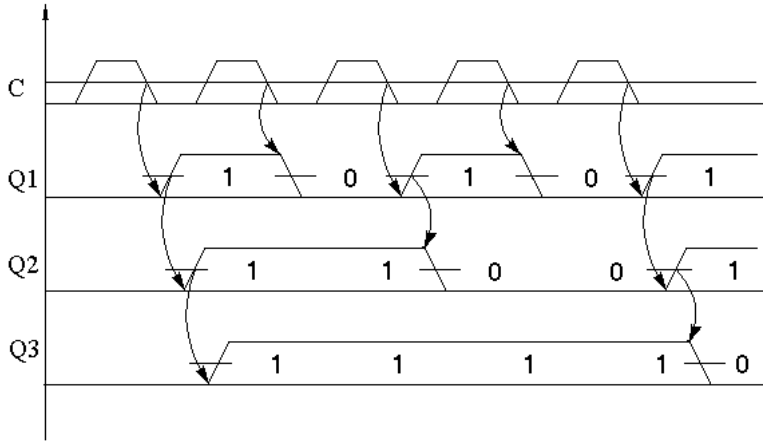

Рисунок 3.4 – Временная диаграмма работы асинхронного вычитающего счётчика с последовательным переносом

На временных диаграммах переключение триггеров DD2 и DD3 синхронизировано положительными фронтами выходов Q1 и Q2.

Таким образом, единственное отличие между суммирующими и вычитающими счетчиками состоит в организации цепей переноса из младших разрядов в старшие.

Все типы счетчиков могут быть использованы в цифровых устройствах “умеренного” быстродействия, когда частота следования синхроимпульсов не превышает критического

Реверсивные счётчики. Реверсивный счетчик – это счетчик с управляемым направлением счета (смотри рисунок 3.5).

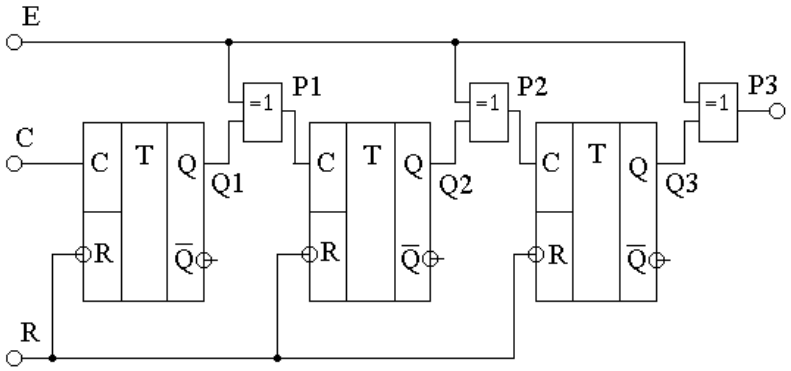

Рисунок 3.5 – Асинхронный реверсивный счётчик

Для построения реверсивного счетчика необходимо между разрядами включить логическую схему, обеспечивающую связь счетного входа второго и последующего триггеров с выходами Q (суммирование) или $\overline{Q}$ (вычитание) предыдущих разрядов.

Возможны различные реализации логических схем (смотри рисунок 3.6) между разрядами (E=0 – суммирование; E=1 – вычитание).

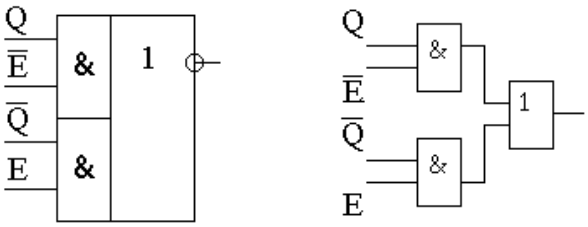

Рисунок 3.6 – Логическая схема для построения реверсивного счётчика

Включение дополнительных логических элементов между разрядами увеличивает время установления.

Синхронные или параллельные счётчики. К синхронным или параллельным относят счетчики, в которых переключение разрядов происходит одновременно, независимо от удаленности разряда от счетного входа. Это достигается подачей на все триггеры синхронизирующих импульсов, которые положительным или отрицательным фронтом вызывают переключение триггеров. Благодаря этому достигается минимальное время установления счетчика $t_{уст.с}$, которое не превышает время установления одного триггера $t_{уст.т}$ Тем самым обеспечивается максимальная частота смены состояний счетчика:

$$f_{c.max} = \frac{1}{t_0 + t_{уст.с}} = \frac{1}{t_0 + t_{уст.т}} \qquad (3.3)$$

Схема простейшего синхронного счетчика показана на рисунке 3.7

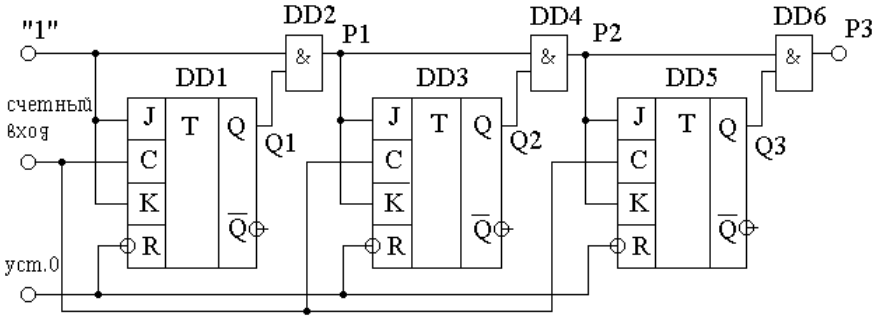

Рисунок 3.7 – Синхронный суммирующий счётчик со сквозным переносом

Здесь сигнал переноса формируется последовательно по мере распространения “1” через DD2, DD4, DD6 и т.д. Накопление задержки за счет этих вентилей обуславливает взаимное смещение счетных импульсов T и импульсов переноса. До тех пор, пока совпадение импульсов T и P_i не нарушается, счетчик работает без сбоев с максимально возможной частотой счета.

Очевидно, для этого необходимо, чтобы выполнялось условие:

$$(N-1)t_{зд.р.ср} < \frac{1}{f_c} - t_{вх} = T_{вх} - t_{вх} \qquad (3.4)$$

Ограничение сверху на N, либо на частоту f_c входных импульсов T, может оказаться технически неприемлемым и обуславливает использование других счетных вариантов счетчиков с дополнительными затратами ИМС.

Принципиальной предпосылкой для построения синхронных счетчиков произвольной разрядности N является следующая закономерность: i-й разряд счетчика переключается синхроимпульсом в новое состояние, если все (i - 1) триггеры младших разрядов к моменту прихода синхроимпульса находятся в “1”. Сигнал, разрешающий переключение триггера i-го разряда в противоположное состояние, должен сформировать логический элемент, фиксирующий состояние “1” всех (i - 1) младших разрядов.

С ростом номера разряда увеличивается количество входов логических элементов цепи переноса, поэтому соответственно с ростом количества разрядов N усложняется схема счетчика.

3.4.2 Описание варианта задания

Разработать поведенческое описание синхронного счётчика с параллельным переносом.

Таблица 3.16 – Задание №16

Уровень I (0 – 5 баллов)	
Интерфейс устройства	Счётчик содержит асинхронный вход сброса Res, асинхронный вход установки Set, счётный вход Clk, вход разрешения счёта Enable, выход Result [...]. Разрядность выходного сигнала Result [...] счётчика задается как статический параметр, значение которого должно быть не менее 4.
Поведение	Если на входе Res удерживается низкий уровень сигнала, на выходе счётчика устанавливается низкий уровень сигнала (независимо от состояния остальных входов). Когда на входе Set удерживается низкий уровень сигнала, на выходе счётчика устанавливается высокий уровень сигнала (независимо от состояния остальных входов). Если же на входах Res и Set одновременно удерживается низкий уровень, выход счётчика переходит в Z-состояние. При условии, что на входах Res, Set, Enable удерживается высокий уровень сигнала, счётчик осуществляет насчёт импульсов, поступающих на его счётный вход по заднему фронту импульса. В случае, когда на вход Enable подается низкий уровень, счётчик прекращает насчёт импульсов и хранит своё текущее значение.
Уровень II (5 – 10 баллов)	
Интерфейс устройства	Счётчик должен дополнительно содержать вход Dir, который управляет направлением счёта.
Поведение	Если на входе Dir удерживается высокий уровень сигнала, счётчик работает как вычитающий. Когда же на входе Dir удерживается низкий уровень сигнала, счётчик работает как суммирующий.
Уровень III (10 – 15 баллов)	
Интерфейс устройства	Дополнительно счётчик содержит вход данных Data [...], разрядность которого равна разрядности самого счётчика, вход загрузки данных Load.
Поведение	Если на входе Load удерживается высокий уровень, в счётчик загружается значение, установленное на входе Data [...] независимо от состояния входа Enable. Низкий уровень на входе Load никак не влияет на работу счётчика.

3.4.3 Разработка схемы алгоритма функционирования синхронного реверсивного счётчика с параллельным переносом

Исходя из анализа известных методов построения и функционирования счётчиков, с учётом всех требований задания, можно предложить следующую схему алгоритма функционирования (смотри рисунок 3.8).

Поведение заданного синхронного счётчика возможно описать при помощи единственного предложения PROCESS, в список чувствительности которого следует указать 3 внешних сигнала – Res, Set, Clk.

Ввиду особенностей описания и использования сигналов в языке VHDL при описании заданного устройства удобно использовать вспомогательный внутренний сигнал sresault. Таким образом, внутри предложения PROCESS появляется возможность осуществлять необходимые изменения и действия с указанным вспомогательным сигналом, а состояние выходов самого счётчика изменять по окончанию выполнения процесса. Для осуществления необходимых действий следует также объявить вспомогательную переменную buf.

По условию задания входы Res и Set счётчика являются асинхронными. Это означает, что сигналы на этих входах напрямую влияют на работу устройства, не зависимо от сигналов на остальных входах. Поэтому, они должны анализироваться в первую очередь. Причём, случай, когда на каждом из этих входов удерживается уровень логического '0' должен анализироваться самым первым, в противном случае – выходы счётчика не всегда смогут переключаться в Z-состояние.

Если ни на одном из асинхронных входов счётчика не удерживается активный уровень сигнала, работа счётчика определяется по сигналу на входе тактирования Clk. По условию задания, счётчик может изменить своё состояние по переднему фронту импульса на этом входе.

После поступления тактового импульса следует проанализировать состояние входа загрузки данных load. Если на нём удерживается уровень логической '1', то происходит загрузка данных в счётчик – выходное состояние счётчик становится равным состоянию, заданному на входах Data[...], в противном случае происходит насчёт импульсов.

Возможность осуществлять насчёт поступивших импульсов определяется уровнем сигнала на входе enable. Если на нём удерживается низкий уровень сигнала, то счётчик переходит в режим хранения, т.е. не изменяет своего состояния. В противном случае – выходное состояние счётчика может изменится в соответствии с состоянием на входе Dir.

Если на входе Dir удерживается уровень логической '1', то счётчик работает как вычитающий, т.е. значение на его выходах декрементируется. В противном случае – происходит инкремент выходного значения счётчика.

Перед окончанием выполнения процесса, следует присвоить выходам счётчика новое состояние, значение которого определяется вспомогательным сигналом.

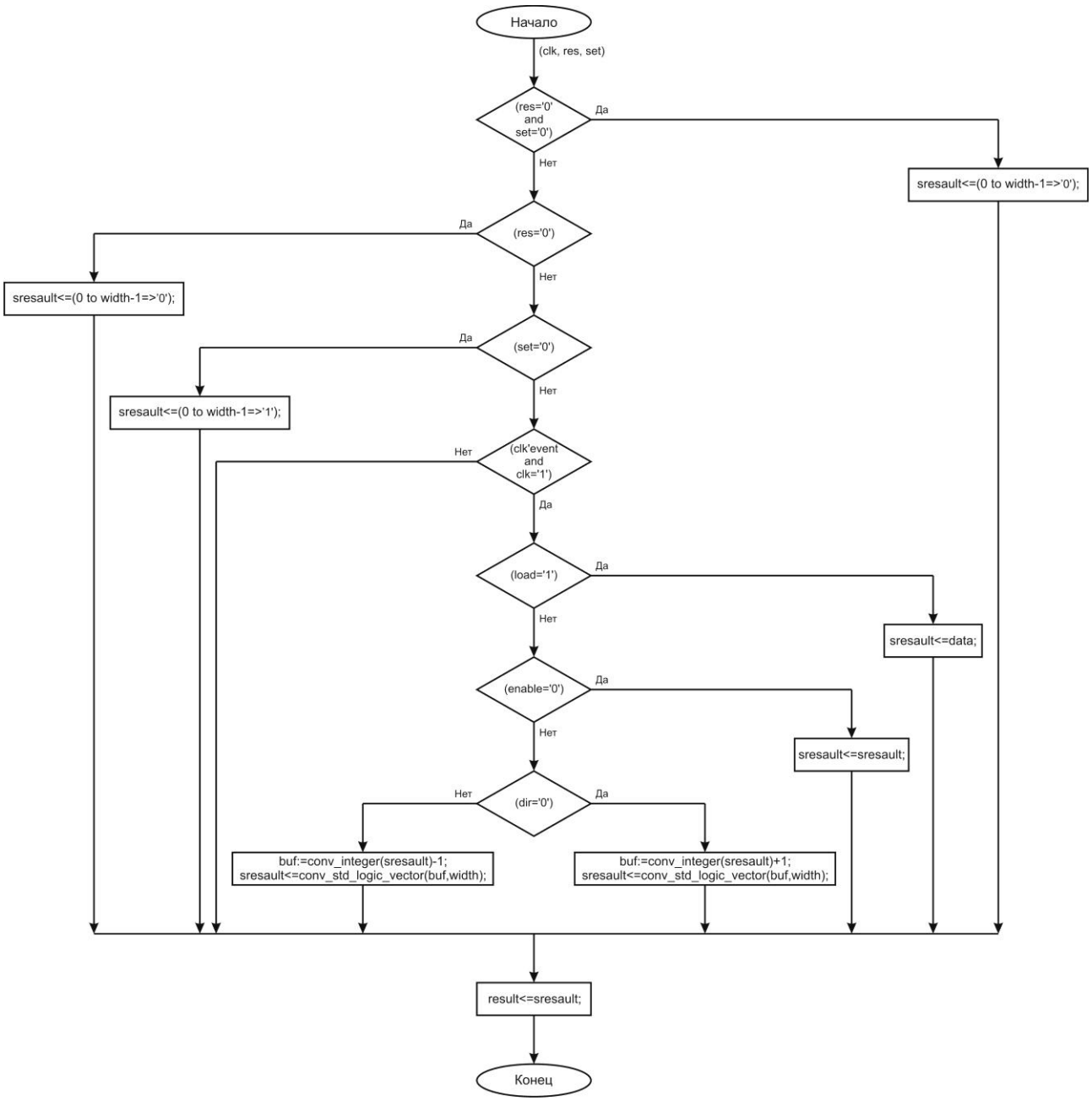

Рисунок 3.8 – Схема алгоритма функционирования устройства

3.4.4 Разработка поведенческого описания синхронного реверсивного счётчика с параллельным переносом на языке VHDL

```
LIBRARY ieee; --подключение библиотеки ieee
USE ieee.std_logic_1164.all; --подключение пакетов этой библиотеки
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY count IS
GENERIC (width: INTEGER:= 4);
PORT (
        data: IN STD_LOGIC_VECTOR (0 TO width-1);  --вход данных
        load: IN BIT; --вход загрузки
```

```
        clk: IN BIT; --вход синхронизации
        res: IN BIT; --асинхронный сброс
        enable: IN BIT; --вход разрешения счета
        set: IN BIT; --асинхронная установка
        dir: IN BIT; --вход задания направление счёта
        result: OUT STD_LOGIC_VECTOR (0 TO width-1)); --выходы
END count;

ARCHITECTURE arc_counter OF count IS
SIGNAL sresault: STD_LOGIC_VECTOR (0 TO width-1);
--сигнал для хранения результата
BEGIN

PROCESS (clk, res, set)
VARIABLE buf: INTEGER; --буферная переменная
BEGIN
        IF(res= '0' AND set= '0') THEN
                sresault<= (0 TO width-1=> 'Z'); --Z-состояние на выходе
        ELSIF(res= '0') THEN
                sresault<= (0 TO width-1=> '0');
--если на асинхронном входе Res удерживается низкий уровень сигнала,
--то на выходе счётчика устанавливается низкий уровень сигнала
        ELSIF(set= '0') THEN
                sresault<= (0 TO width-1=> '1');
--если на асинхронном входе Set удерживается низкий уровень сигнала,
--то на выходе счётчика устанавливается высокий уровень сигнала
        ELSIF (clk'EVENT AND clk= '1') THEN
--если пришёл передний фронт синхроимпульса
                IF(load= '1') THEN
                        sresault<= data;
--если на входе Load удерживается высокий уровень, то в счётчик
--загружается значение, установленное на входе Data
                ELSIF(enable= '0') THEN
                        sresault<= sresault;
--если на входе enable удерживается низкий уровень, то счётчик
--прекращает насчёт импульсов и хранит своё текущее значение
                ELSIF(dir= '0') THEN
--если на входе Dir удерживается высокий уровень сигнала,
--то счётчик работает как суммирующий
                        buf:= CONV_INTEGER(sresault)+ 1;
                        sresault<= CONV_STD_LOGIC_VECTOR(buf, width);
                ELSE
--в противном случае счётчик работает как вычитающий
                        buf:= CONV_INTEGER(sresault)- 1;
                        sresault<= CONV_STD_LOGIC_VECTOR(buf, width);
```

```
            END IF;
        END IF;
        result<= sresault; --изменение выходного состояния счётчика
END PROCESS;

END arc_counter;
```

3.4.5 Анализ работоспособности синхронного реверсивного счётчика с параллельным переносом и оценка соответствия заданным требованиям

Если на входе Res удерживается низкий уровень сигнала, на выходе счётчика устанавливается низкий уровень сигнала независимо от состояния остальных входов (смотри рисунок 3.9).

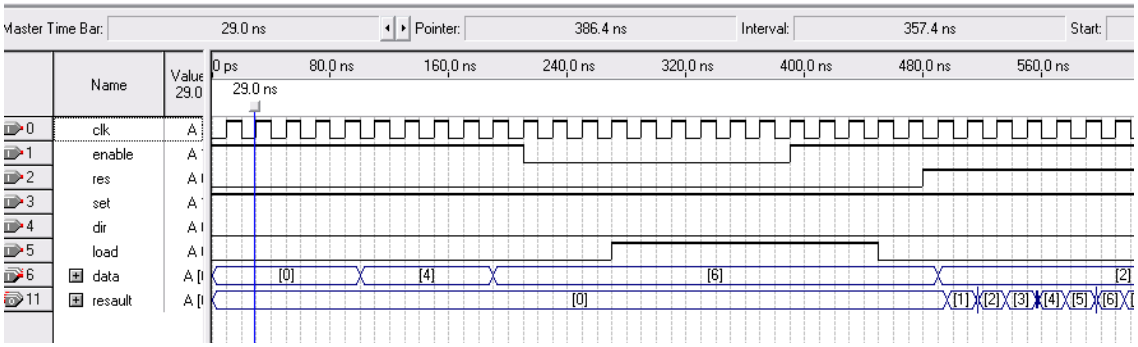

Рисунок 3.9 – На входе Res установлен низкий уровень сигнала

Когда на входе Set удерживается низкий уровень сигнала, на выходе счётчика устанавливается высокий уровень сигнала независимо от состояния остальных входов (смотри рисунок 3.10).

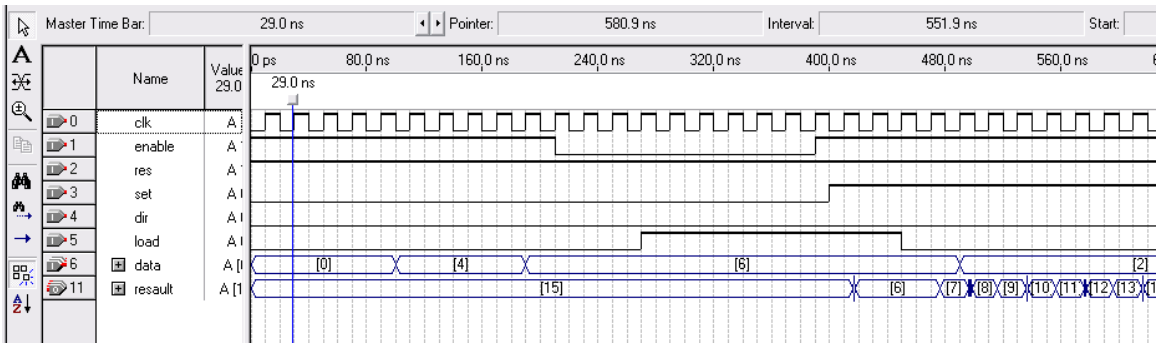

Рисунок 3.10 – На входе Set установлен низкий уровень сигнала

Если же на входах Res и Set одновременно удерживается низкий уровень сигнала, выход счётчика переходит в Z-состояние (смотри рисунок 3.11).

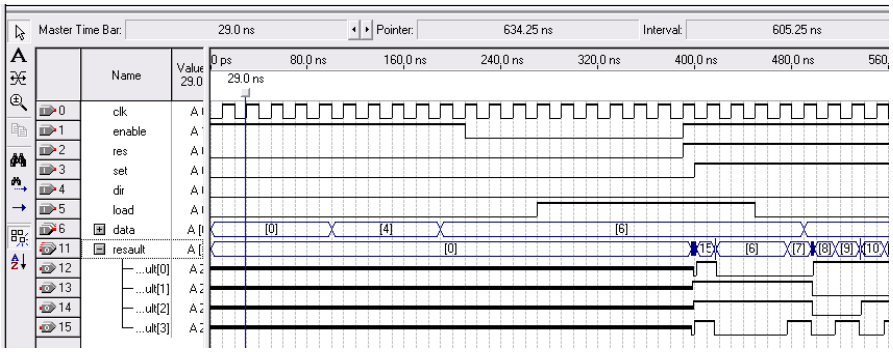

Рисунок 3.11 – На входах Set и Res установлен низкий уровень сигнала

При условии, что на входах Res, Set, Enable удерживается высокий уровень сигнала, счётчик осуществляет насчёт импульсов, поступающих на его счётный вход по заднему фронту импульса (смотри рисунок 3.12).

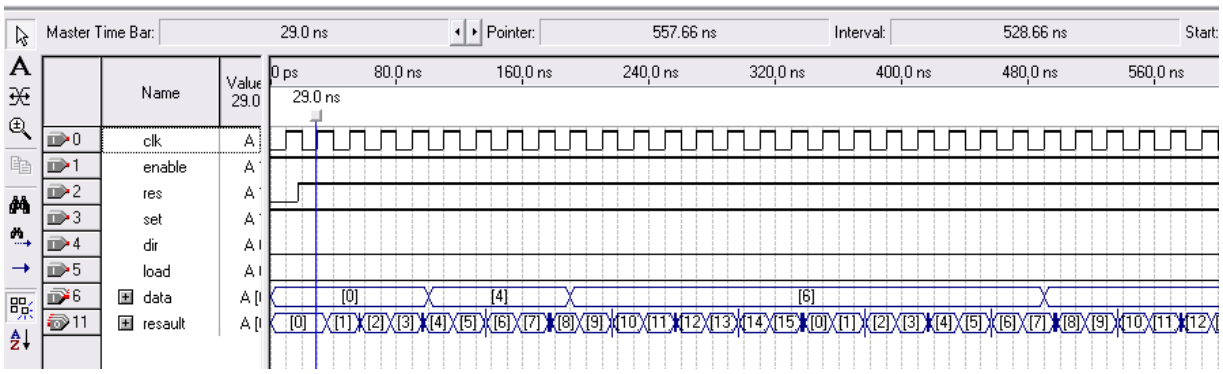

Рисунок 3.12 – На входах Res, Set, Enable удерживается высокий уровень сигнала

В случае, когда на вход Enable подается низкий уровень, счётчик прекращает насчёт импульсов и хранит своё текущее значение (смотри рисунок 3.13).

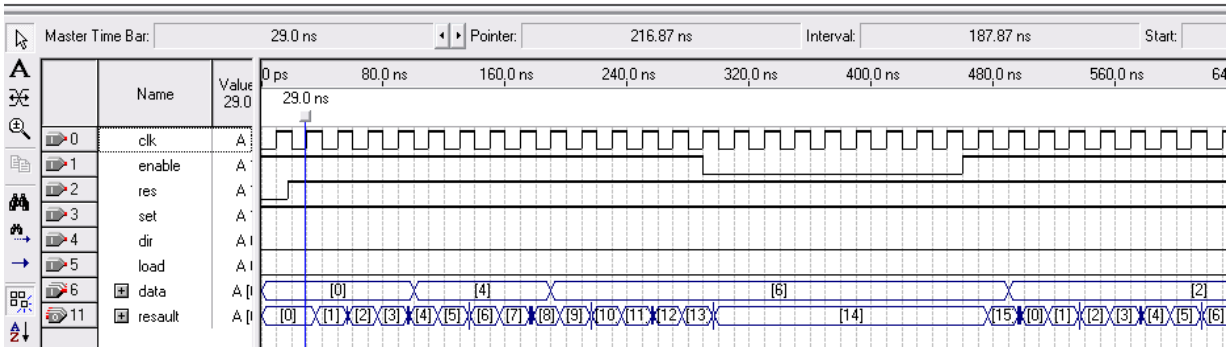

Рисунок 3.13 – На входе Enable удерживается низкий уровень сигнала

Если на входе Dir удерживается высокий уровень сигнала, счётчик работает как вычитающий (смотри рисунок 3.14).

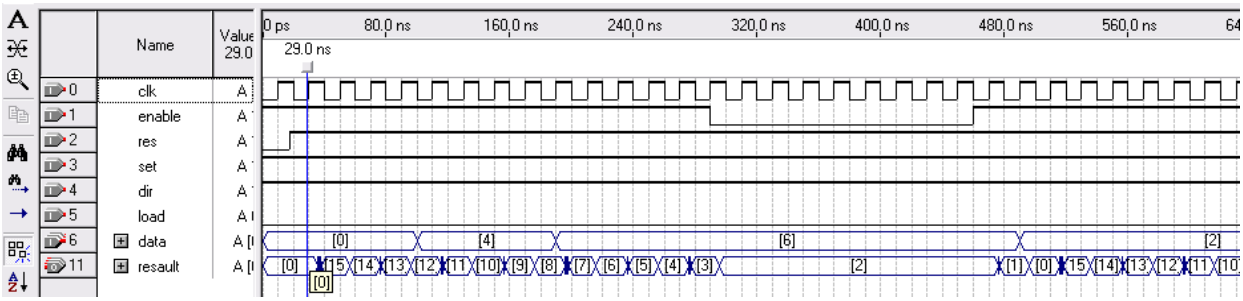

Рисунок 3.14 – На входе dir удерживается высокий уровень сигнала

Когда же на входе Dir удерживается низкий уровень сигнала, счётчик работает как суммирующий (смотри рисунок 3.15).

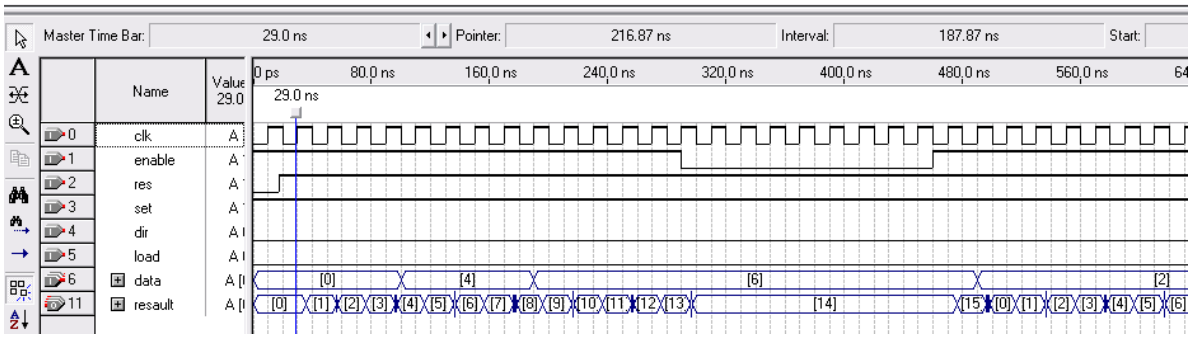

Рисунок 3.15 – На входе dir удерживается низкий уровень сигнала

Если на входе Load удерживается высокий уровень, в счётчик загружается значение, установленное на входе Data [...] независимо от состояния входа Enable. Низкий уровень на входе Load никак не влияет на работу счётчика (смотри рисунок 3.16).

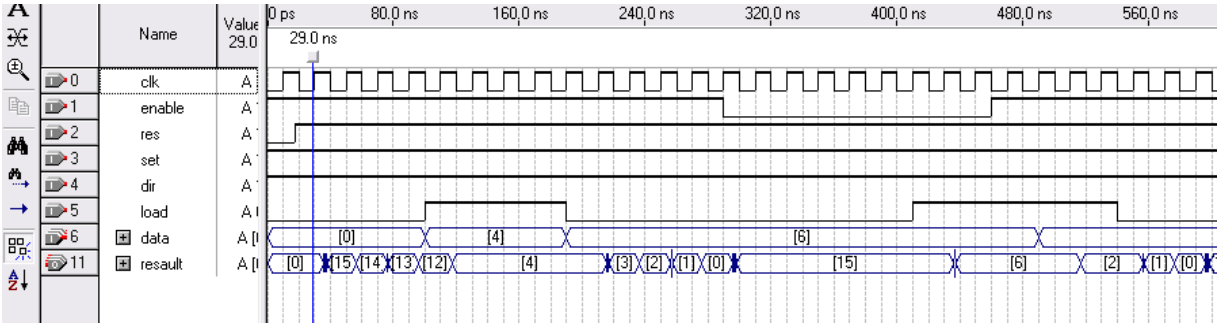

Рисунок 3.16 – На входе Load удерживается высокий и низкий уровень

Как видно из анализа полученных временных диаграмм, разработанный на языке VHDL синхронный реверсивный счётчик с параллельным переносом соответствует всем требованиям, каждого из трёх уровней задания.

3.4.6 Выводы

В ходе выполнения расчетно-графической работы был проведен теоретический анализ существующих методов построения синхронных счётчиков.

Синхронные счётчики – это счётчики, в которых переключение разрядов происходит одновременно, независимо от удаленности разряда от счетного входа.

По заданию на расчетно-графическую работу было разработано поведенческое описание синхронного счетчика с параллельным переносом на языке VHDL. В ходе создания синхронного счетчика с параллельным переносом были использования следующие синтаксические конструкции языка VHDL:

– If Statement – позволяет осуществлять выбор предложения (или группы последовательных предложений) для выполнения на основании результатов проверки выполнения указанных условий;

– Process Statement – это параллельное предложение языка, в котором на основе множества последовательных предложений описывается поведение всего устройства или некоторой его части;

– Variable Declaration Statement – используется для декларации переменных внутри тела процесса;

– Signal Declaration Statement – используется для описания внутренних сигналов устройства, которые не видимы из внешнего мира и являются частью внутренней архитектуры системы и предназначены для обеспечения связи между внутренними узлами и устройствами.

Разработанный синхронный счётчик с параллельным переносом в дальнейшем может быть использован, как библиотечное устройство.

РЕКОМЕНДОВАННАЯ ЛИТЕРАТУРА

1 Антонов А.П. Язык описания цифровых устройств AlteraHDL. Практический курс. – М.: ИП РадиоСофт, 2001. – 224 с.

2 Бибило П.Н. Основы языка VHDL. – Москва: Солон-Р, 2000. – 205 с.

3 Казеннов Г.Г. Основы проектирования интегральных схем и систем. – М.: БИНОМ. Лаборатория знаний, 2005. – 295с.

4 Комолов Д.А. Мяльк Р.А., Зобенко А.А., Филоппов А.С. Системы автоматизированного проектирования фирмы Altera MAX+plus II и Quartus II. Краткое описание самоучитель. – М.: ИП РадиоСофт, 2002. – 352с.: ил.

5 Соловьев В.В. Проектирование цифровых систем на основе программируемых логических интегральных схем. 2-е изд., стереотип. – М.: Горячая линия – Телеком, 2007. – 636 с.

6 Стешенко В.Б. ПЛИС фирмы ALTERA: проектирование устройств обработки сигналов. - М.: Додека, 2000. - 127 с.

7 Стешенко В.Б. ПЛИС фирмы Altera: элементная база, система проектирования и языки описания аппаратуры. – М.: Издательский дом «Додэка XXI», 2002. – 576 с.

8 Суворова Е. А., Шейнин Ю. Е. Проектирование цифровых систем на VHDL. – СПб.: БХИ-Петербург, 2003, - 576 с.

9 Яицков А.С. VHDL – язык описания аппаратных средств. – М.: Латмэс, 1998. – 119 с.

10 IEEE Std 1076-2000 // IEEE Standard VHDL. Language Reference Manual. – New York: IEEE, 2000/ - 290 с.

11 Introduction to the Quartus II Software. – San Jose: Altera Corporation, 2009. – 229 с.

12 Quartus II Handbook Version 9.0. – San Jose: Altera Corporation, 2009. – 2490 с.