Министерство образования и науки Украины
Черниговский государственный технологический университет

VHDL – ЯЗЫК ОПИСАНИЯ АППАРАТНЫХ СРЕДСТВ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к выполнению лабораторных работ 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 № 12
от 30 мая 2013 г.

Чернигов ЧГТУ 2013

VHDL – мова опису апаратних засобів. Методичні вказівки до виконання лабораторних робіт з дисципліни "Технології проектування комп'ютерних систем" для студентів напряму підготовки 6.050102 – "Комп'ютерна інженерія"./Укл. О. І. Вервейко, О.В. Красножон – Чернігів: ЧДТУ, 2013. – 77 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, доцент кафедры информационных и компьютерных систем Черниговского государственного технологического университета

СОДЕРЖАНИЕ

1 ИСТОРИЯ СОЗДАНИЯ VHDL 7
2 АЛФАВИТ ЯЗЫКА 11
3 ЛЕКСИЧЕСКИЕ ЭЛЕМЕНТЫ 12
3.1 Разделители и ограничители 12
3.2 Идентификаторы 12
3.3 Ключевые (зарезервированные) слова 13
3.4 Абстрактные литералы 14
3.4.1 Десятичные литералы 14
3.4.2 Литералы с указанием основания системы счисления 14
3.5 Символьные литералы 15
3.6 Строковые литералы 15
3.7 Битовые строки 16
3.8 Комментарии 16
4 ТИПЫ ДАННЫХ 17
4.1 Скалярные типы 17
4.1.1 Перечисления 17
4.1.2 Целые числа 19
4.1.3 Физические типы 19
4.1.4 Числа с плавающей точкой 20
4.2 Составные типы 20
4.2.1 Массивы 21
4.2.2 Записи 22
4.3 Указатели 22
4.4 Файлы 23
5 ИМЕНА 24
5.1 Простое имя 24
5.2 Символ оператора 24
5.3 Селективное имя 25
5.4 Индексное имя 25
5.5 Вырезка имени 26
5.6 Имя атрибута 26
5.7 Видимость и область действия имен 26
6 ВЫРАЖЕНИЯ 27
6.1 Операторы 27
6.1.1 Логические операторы 27
6.1.2 Операторы сравнения 29
6.1.3 Операторы сдвига 29
6.1.4 Аддитивные операторы 29
6.1.5 Мультипликативные операторы 30
6.1.6 Знаковые операторы 31
6.1.7 Смешанные операторы 31
6.2 Операнды 31
6.2.1 Операнд Name 31

6.2.2 Операнд Literal.. 32
6.2.3 Операнд Aggregate.. 32
6.2.4 Операнд Function Call 33
6.2.5 Операнд Qualified Expression 34
6.2.6 Операнд Type Conversion 34
6.2.7 Операнд Allocator 34
6.2.8 Статическое выражение.. 34
6.2.9 Универсальное выражение.. 34
7 ПРЕДСТАВЛЕНИЕ СИСТЕМЫ В VHDL 35
7.1 Общая структура описания проекта системы.............................. 35
7.2 Сущность проекта системы 35
7.3 Архитектура проекта системы 37
7.4 Предложения VHDL... 38
8 СТРУКТУРНОЕ ОПИСАНИЕ ЦИФРОВЫХ УСТРОЙСТВ.................. 39
8.1 Сигналы в VHDL 39
8.2 Описание нерегулярных структур 41
8.3 Описание регулярных структур 43
9 ФУНКЦИОНАЛЬНОЕ ОПИСАНИЕ ЦИФРОВЫХ УСТРОЙСТВ........ 46
9.1 Описание комбинационных устройств... 46
9.1.1 Особенности применения сигналов 47
9.1.2 Предложение Process Statement 47
9.1.3 Предложение Variable Declaration Statement........................... 48
9.1.4 Предложение Variable Assignment Statement........................... 49
9.1.5 Подпрограммы.. 49
9.1.6 Предложение If Statement... 52
9.1.7 Предложение Case Statement... 52
9.1.8 Предложение Loop Statement.. 53
9.2 Описание последовательных устройств.. 54
9.2.1 Описание триггеров 55
9.2.2 Описание регистров 56
9.2.3 Описание счетчиков.. 57
9.2.4 Описание цифровых автоматов 58
10 МОДЕЛИРОВАНИЕ ЦИФРОВЫХ УСТРОЙСТВ............................... 61
10.1 Общие методические рекомендации по выполнению лабораторных работ.. 61
10.2 Лабораторная работа №1. Структурное описание комбинационных устройств... 63
10.3 Лабораторная работа №2. Структурное описание последовательных устройств.. 67
10.4 Лабораторная работа №3. Поведенческое описание комбинационных устройств... 69
10.5 Лабораторная работа №4. Поведенческое описание последовательных устройств.. 73
РЕКОМЕНДОВАННАЯ ЛИТЕРАТУРА 77

ВВЕДЕНИЕ

При создании аппаратных средств компьютерных систем помимо унифицированных устройств, выпускаемых в виде сверх больших интегральных схем (СБИС) массового применения (микропроцессоры, СБИС памяти, процессоры цифровой обработки сигналов, однокристальные микроЭВМ), требуются специализированные устройства, состав и функции которых определяются назначением системы.

Подобного рода устройства могут быть созданы на базе микросхем общего назначения, имеющих средний и малый уровень интеграции. Однако реализация их в виде единой специализированной СБИС позволяет улучшить основные общетехнические показатели всей системы в целом: повысить быстродействие и надежность, уменьшить энергопотребление, габариты, стоимость.

Специализированная СБИС может быть либо заказной, либо реализованной на базе полузаказной СБИС.

Создание заказной специализированной СБИС связано с дорогостоящим проектированием и изготовлением, что при малом объеме выпуска (единичное и мелкосерийное производство) приводит к значительной стоимости получаемых СБИС.

При использовании полузаказных СБИС процесс создания специализированной СБИС разделяется на два этапа. Первый этап – этап массового изготовления – связан с созданием "полуфабриката", имеющего сравнительно небольшую стоимость. Второй этап предполагает допроектирование и специализацию СБИС в соответствии с требованиями заказчика. При этом трудоемкость и стоимость этапов изготовления "полуфабриката" СБИС, его допроектирования и специализации много меньше, чем полного цикла проектирования и изготовления заказной СБИС.

В классе полузаказных СБИС выделяют два отличающихся типом полуфабриката подкласса: базовые матричные кристаллы (БМК) и СБИС программируемой логики (СБИС ПЛ), которые называют также программируемые логические интегральные схемы (ПЛИС).

Полуфабрикат БМК содержит нескоммутированные электронные ключи (транзисторы), которые сгруппированы в базовые ячейки. Для специализации БМК необходимо настроить его на выполнение тех или иных логических функций, что осуществляется соединением транзисторов внутри базовых ячеек и базовых ячеек между собой. Для этого реализуется "на заказ" от одного до трех слоев СБИС. Таким образом, в БМК, в отличие от заказных СБИС, из 15 - 20 слоев "на заказ" выполняется лишь небольшая их часть, что позволяет удешевить и ускорить процесс создания специализированной СБИС на основе БМК. Однако, для специализации БМК, так же как и при создании заказной СБИС, необходимо сложное технологическое оборудование, а время, затрачиваемое на топологическое проектирование (проектирование соединений транзисторов и базовых ячеек) и изготовление специализированной СБИС, составляет не менее двух-трех месяцев.

В СБИС ПЛ, являющейся полуфабрикатом для создания специализированной СБИС, в процессе массового изготовления полностью реализуются как все простейшие функциональные преобразователи, так и соединения между ними, а, кроме того, дополнительные программируемые элементы, позволяющие настроить функциональные преобразователи на выполнение требуемых логических функций и организовать между ними необходимые соединения. Допроектирование и специализация, т.е. настройка программируемых элементов (программирование), СБИС ПЛ могут быть выполнены на одном рабочем месте, оснащенном персональным компьютером, соответствующим пакетом прикладных программ и программатором.

В СБИС ПЛ, в отличие от БМК, соединения между функциональными преобразователями и программируемые элементы занимают намного большую, чем сами функциональные преобразователи, площадь кристалла. Поэтому только с переходом на субмикронную технологию логическая емкость СБИС ПЛ стала сопоставима с логической емкостью БМК. Это обстоятельство, а также:

– меньшая стоимость и сложность цикла допроектирования и специализации СБИС ПЛ;

– возможность внесения изменений в проект на любой стадии проектирования и изготовления специализированной СБИС;

– возможность перепрограммирования СБИС после ее распайки на плате;

– отсутствие необходимости в сложном технологическом оборудовании

определили интенсивное развитие в последние годы направления СБИС ПЛ.

В настоящее время на одном кристалле СБИС ПЛ размещаются миллионы логических элементов и разработка цифровых систем не может осуществляться традиционными методами (проектирование с использованием булевых уравнений, схемное проектирование) из-за их трудоемкости и сложности анализа результатов проектирования. Для решения этой проблемы были созданы специальные искусственные языки описания аппаратуры (ЯОА, HDL, Hardware Description Languages), решающие основные задачи, существенные для проектирования цифровых систем.

1 ИСТОРИЯ СОЗДАНИЯ VHDL

Языки описания аппаратуры появились в начале 60-х годов, в основном, по следующим причинам. Во-первых, требовалось как можно дольше работать с проектом на абстрактном формальном уровне, не опускаясь на уровень мелких деталей и физической реализации. Во-вторых, быстрый рост сложности цифровых систем требовал адекватных средств их разработки и систем автоматизированного проектирования (САПР).

Среди первых ЯОА выделяют LOTIS, EPICURE, CDL, HARGOL, CASSANDRE, DDL. Затем в течение семидесятых годов в Европе и США было разработано много ЯОА, их насчитывалось более 200 – отечественных и зарубежных. Самыми известными являются МОДИС, МОДИС-В78, Автокод, МПЛ, ОСС-2, ФОРОС, EPICURE, CDL, DDL, ISPS, CONLAN, HILO, CASCADE, RAGLAN, ДИСП, САМРИС, СПАРС и др.

Рассмотрим в качестве примера описание схемы, приведенной на рисунке 1.1, в программе ДИСП (диалоговая схема схемотехнического проектирования).

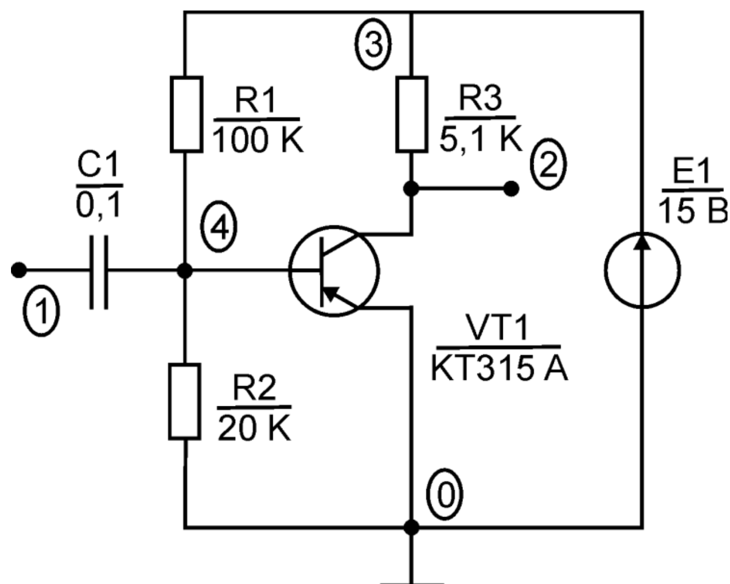

Рисунок 1.1 – Схема транзисторного усилителя

Описание схемы имеет вид:

СХЕМА:	УСИЛИТЕЛЬ;
РАЗРАБОТЧИК:	ИВАНОВ;
ОПИСАНИЕ	
R1 (4,3)	100 K;
R2 (4,0)	20 K;
R3 (2,3)	5,1 K;
C1 (1,4)	0,1 мкФ;
E1 (3,0)	15 В;
VT1 (4,3,0)	КТ315 А; *
NN = 4;	
ЧАСТОТА:	1 Гц, 1 кГц, 1 МГц;

КОНЕЦ;

Директивы "СХЕМА" и "РАЗРАБОТЧИК" идентифицируют схему и разработчика. Со слова "ОПИСАНИЕ" начинается собственно описание схемы. Каждая строка содержит имя модели компонента, номер компонента в схеме, в скобках записаны номера узлов включения компонентов. Для пассивных элементов указаны номиналы, для транзистора – его тип. Последняя строка в описании помечена символом "*" сразу после точки с запятой. Предложение "NN=4;" определяет количество узлов в схеме. Массив "ЧАСТОТА" задает информацию для программы расчета частотных характеристик. По директиве "КОНЕЦ" обработка текста описания прекращается.

Первые языки ЯОА были предназначены, в основном, для описания устройств и использовались при их моделировании.

В 60-е и 70-е годы основной вклад в развитие языков описания аппаратуры внесли США и Европа. Вклад Японии в этот период был менее значительным.

В 1973 году был начат проект CONLAN (CONsensus LANguage) – первая попытка разработки стандарта языка, моделирующего многоуровневые проекты и имеющего формализованную семантику. В 1980 году были опубликованы основные результаты проекта, в 1983 – полное определение языка. В то время были разработаны и использовались другие ЯОА (ELLA, DACAPO III, CASCADE, REGLAN, KARL, VERILOG), однако именно CONLAN стал основой VHDL.

Первоначальным назначением языка VHDL было документирование проектов и выпускаемых электронных изделий. Это позволяло разработчикам и пользователям легко и однозначно понимать выполняемые изделиями функции. В дальнейшем круг задач, решаемых с помощью VHDL, расширился. В 1981 году состоялась первая конференция, на которой были сформулированы требования к языку. В конце 1982 года был выпущен предназначенный для компаний и учреждений документ "Запрос на предложения", в котором появилось первое название языка: VHDL (VHSIC (Very-High Speed Integrated Circuits) Hardware Design and Description Language).

Первыми проект поддержали компании Intermetrix, IBM и Texas Instruments. Язык разрабатывался стремительно, и в 1984 году вышла уже шестая его версия. Версия VHDL 7.2, появившаяся в августе 1985 года, удовлетворяла большинству требований и была принята в качестве стандарта в рамках Министерства Обороны США. Затем развитие и стандартизация языка были продолжены в DATC (техническом комитете по автоматизации проектирования), IEEE (Международный комитет по электротехнике и электронике). В июне 1986 вышла первая рабочая версия руководства по языку (Language Reference Manual). Окончательная версия проекта языка была вынесена на голосование 11 декабря 1987 года и принята в качестве стандарта IEEE STD 1076-1987.

В IEEE все стандарты должны пересматриваться каждые пять лет. Первый пересмотр стандарта на VHDL состоялся в 1992 году, хотя подготовка к этому началась в июне 1990 года с устранения неточностей и слабых мест в

стандарте от 1987 года. Основные принципы построения языка были сохранены, а набор предоставляемых средств расширен. Голосование в декабре 1992 года дало положительные результаты, выявив при этом ряд замечаний, устранение которых привело к версии, принятой в 1993 году.

В июле 1997 года в Париже состоялась вторая конференция "Будущее VHDL" (The future of VHDL), которая позволила выразить свою точку зрения на проблемы проектирования сложных цифровых систем разработчикам языка, разработчикам средств автоматизированного проектирования, базирующихся на этом языке, пользователям языка и средств.

Следующий пересмотр стандарта состоялся в 1998 году. Изменения коснулись, в основном, расширения возможностей языка VHDL в части описания аналоговых устройств.

В настоящее время ведутся исследования, целью которых является разработка фундаментальных направлений и принципов развития языка в будущем. Вопросы будущего языка VHDL тщательно исследуются как в США (главным образом в Министерстве Обороны), так и в Европейском Сообществе.

Существуют два центра, занимающиеся распространением, внедрением, поддержкой и другими вопросами, относящимися к VHDL.

VHDL International – компания, являющаяся посредником между разработчиками языка и его пользователями, издающая и распространяющая ежеквартальный бюллетень, называемый VHDL Times, находится в Санта-Клара, Калифорния, США,

European CAD Standardization Initiative (ECSI – Европейская инициатива в стандартизации САПР) – научно-технический центр, координирующий всю деятельность, относящуюся к VHDL в Европе, и издающий два ежеквартальных бюллетеня: VHDL Newsletter и ECSI Letter. ECSI находится во Франции, в Гренобле. При ECSI существуют национальные группы пользователей языка VHDL, созданные во многих странах Европы. Центр организует конференции, форумы, школы и поддерживает деятельность технического комитета IEEE по автоматизации проектирования.

DATE (Design, Automation and Test in Europe, а в недавнем прошлом ED&TC и EURO-DAC с EURO-VHDL) является центральной европейской конференцией для ученых, ведущих исследования по VHDL и другим языкам описания аппаратуры. VHDL International User's Forum в США, VHDL-Forum в Европе и Asia-Pacific Conference on Hardware Description Languages в Азии являются центральными конференциями для различных типов пользователей языка.

В настоящее время язык VHDL обеспечивает описание проектов различной степени сложности – от простейшего вентиля до целой системы, состоящей из аппаратных и программных частей. Он позволяет строить модели на различных уровнях абстракции, выполнять имитационное моделирование и генерировать временные диаграммы, вести строгое документирование проекта, осуществлять синтез структуры по поведенческому описанию, верифицировать проект формальными методами, автоматически генерировать тесты.

VHDL наследует многие свойства языков программирования высокого уровня, в частности, языка программирования ADA. Дополнительно в язык введены понятия модельного времени, сигнала, события, компонента и другие. Использование VHDL позволяет не привязывать проект заранее к конкретному физическому способу реализации, одна и та же логическая VHDL-реализация является источником генерации различных физических.

В основе языка лежат следующие принципы построения:

– поддержка функциональной декомпозиции (функциональная иерархия и рекурсия);

– поддержка структурной декомпозиции (структурная иерархия);

– представление системы в виде параллельно функционирующих взаимодействующих процессов;

– использование абстрактных типов данных;

– использование событийного моделирования;

– поддержка различных уровней абстракции и детализации представления проекта.

Первоначально VHDL создавался как язык описания цифровых схем, в последние годы выполнена большая работа по расширению языка на описание и моделирование аналоговых схем. Принятие цифро-аналогового VHDL в качестве стандарта придаст языку принципиально новые возможности, увеличит его популярность, расширит сферы применения.

2 АЛФАВИТ ЯЗЫКА

Язык VHDL имеет свой алфавит, который представляет собой набор символов, разрешенных к использованию и воспринимаемых компилятором. С помощью этих символов могут быть образованы величины, выражения и операторы данного языка. Алфавит языка составляют:

– символы из набора ISO 8859-1:1987 (International Organization for Standardization);

– составные символы, воспринимаемые компилятором как один символ (смотри таблицу 2.1).

Таблица 2.1 – Составные символы алфавита языка VHDL

Символ	Описание
<=	Меньше или равно, присвоение
>=	Больше или равно
=>	Следует
:=	Присвоение
/=	Не равно
**	Возведение в степень
< >	Границы

3 ЛЕКСИЧЕСКИЕ ЭЛЕМЕНТЫ

Текст на языке VHDL – это последовательность раздельных лексических элементов (лексем). Лексема – минимальное объединение символов, несущее смысл. Различают следующие виды лексем:

- разделитель и ограничитель;
- идентификатор;
- ключевое (зарезервированное) слово;
- абстрактный литерал;
- символьный литерал;
- строковый литерал;
- битовые строки;
- комментарий.

3.1 Разделители и ограничители

Разделители и ограничители служат для разъединения (установки границ) лексических элементов (слов).

Разделителями служат символы: пробел, табуляция и конец строки.

Количество разделителей не имеет значения. Таким образом, следующие выражения для компилятора будут эквивалентны:

```
count:= 2+2;
count := 2 + 2;
count :=      2
+
2;
```

Ограничители - это специальные одиночные символы (в основном наборе символов):

& ' ( ) * + , - . / : ; < = > | [ ]

или составные (парные) символы, указанные в таблице 2.1.

3.2 Идентификаторы

Идентификаторы – это простые пользовательские имена, которые присваиваются некоторому объекту.

Определение (в форме Бэкуса-Наура).

identifier ::= letter { [_] letter | digit}

В программе идентификаторы могут конструироваться из строчных и прописных букв, цифр от 0 до 9 и символа подчеркивания '_' (и только из них!). Кроме того, написание идентификаторов должно подчиняться следующим правилам:

– не может быть зарезервированным словом языка;

– должен начинаться с буквы (не с цифры);

– не может заканчиваться символом подчеркивания '_';

– не должен содержать двух и более последовательных символов подчеркивания '_';

– не может содержать внутри себя пробелы и специальные символы '-', '@', '%'.

В VHDL-коде нет различия между прописными и строчными буквами. Так ident1, IDENT1 и Ident1 – это все одно и то же имя. Примеры идентификаторов приведены в таблице 3.1.

Таблица 3.1 – Примеры идентификаторов

Правильные идентификаторы	Неправильные идентификаторы
carry_out	7AB (начинается с цифры)
Dim_Sum	A@B (специальный символ @)
Count7SUB_2goX	SUM_ (заканчивается подчеркиванием)
AaBBb	PI__A (два подчеркивания подряд)
ExampleOut	Example Out (пробел не допустим)

3.3 Ключевые (зарезервированные) слова

В VHDL-87 зарезервировано 81 ключевое слово, и VHDL-93 дополнительно введены 16 зарезервированных слов (смотри таблицы 3.2 – 3.3).

Таблица 3.2 – Зарезервированные слова VHDL-87

Abs	Access	After	Alias	All
And	Architecture	Array	Assert	Attribute
Begin	Block	Body	Buffer	Bus
Cost	Component	Configuration	Constant	Disconnect
Downto	Else	Elsif	End	Entity
Exit	File	For	Function	Generate
Generic	Guarded	If	In	Inout
Is	Label	Library	Linkage	Loop
Map	Mod	Nand	New	Next
Nor	Not	Null	Of	On
Open	Or	Others	Out	Package
Port	Procedure	Process	Range	Record
Register	Rem	Report	Return	Select
Severity	Signal	Subtype	Then	To
Transport	Type	Units	Until	Use
Variable	Wait	When	While	With
Xor				

Таблица 3.3 – Зарезервированные слова VHDL-93

Group	Impure	Inertial	Literal	Postponed
Pure	Reject	Rol	Ror	Shared
Sla	Sll	Sra	Srl	Unaffected
Xnor				

3.4 Абстрактные литералы

В VHDL имеется два вида абстрактных литералов: реальные литералы и целочисленные литералы. Реальный литерал – абстрактный литерал с точкой, имеющий тип universal_real. Целочисленный литерал – абстрактный литерал без точки, имеющий тип universal_integer.

abstract_literal ::=decimal_literal |based_literal

3.4.1 Десятичные литералы

Десятичные литералы – абстрактные литералы, выраженные в десятичной системе счисления. Они могут быть целыми, реальными или целыми и реальными с экспонентой.

decimal_literal ::=integer [.integer] [exponent]
integer ::=digit {[underline] digit }
exponent ::=E [+] integer |E – integer

Знак экспоненты E может быть строчным либо прописным.
Подчеркивание в десятичном литерале не является значащим.
Экспонента для целого литерала не должна иметь знак минус.
Средства синтеза ПЛИС допускают применение только целых литералов.
Примеры описания чисел:

- целые числа: 21, 0, 1E2, 3e4, 123_000;
- реальные числа: 11.0, 0.0, 0.468, 3.141_592_6;
- реальные числа с экспонентой: 1.23E-11, 1.0E+4, 3.024E+23.

3.4.2 Литералы с указанием основания системы счисления

Литерал с указанием основания системы счисления – абстрактный литерал, выраженный в форме, в которой явно указано основание системы счисления. Основание может быть от двух до шестнадцати.

based_literal ::=
base #based_integer [.based_integer] #[exponent]
base ::=integer
based_integer ::=
extended_digit {[underline] extended_digit }
extended_digit ::=digit |letter

Символ подчеркивания, вставленный между смежными цифрами литерала, не изменяет его значения. Основание и показатель должны быть записаны в

десятичной системе счисления. В литерале могут использоваться буквы от A до F для указания цифр от десяти до пятнадцати. Знак экспоненты E может быть строчным либо прописным.

Примеры:

Целочисленные литералы со значением 255:

2#1111_1111# 16#FF# 016#0FF#

Целочисленные константы со значением 224:

16#E#E1 2#1110_0000#

Вещественные константы со значением 4095.0:

16#F.FF#E+2 2#1.1111_1111_111#E11

3.5 Символьные литералы

Символьные литералы формируется с помощью одного из 191 графических символов (включая пробел) между двумя символами апострофа. Символьный литерал имеет значение, которое принадлежит символьному типу.

character_literal ::='graphic_character'

Примеры:

'A' '*' ''' ' '

3.6 Строковые литералы

Строковый литерал формируется как последовательность букв (возможно пустая), заключенных в двойные кавычки, которые применяют как строковые скобки.

string_literal ::= " {graphic_character} "

Подчеркнем, что 'a' – это символьный литерал, а "a" – строковый, и если, объявив переменную x символьного типа (**VARIABLE** x: CHARACTER), попытаться присвоить ей значение "a" (x := "a";), то это будет ошибкой .

Значением строкового литерала является последовательность символов, соответствующих графическим символам константы строки, кроме кавычек.

Для включения кавычки в строку необходимо ввести две двойные кавычки.

Строковый литерал должен располагаться в одной строке. Для формирования "длинных" строковых литералов может быть употреблена операция конкатенации &.

Длина строкового литерала – количество символов в представленной последовательности.

Примеры строковых литералов:

"Установка времени слишком коротка " --сообщение об ошибке.

" "-- пустой строковый литерал.

" " " A " """" -- три строковых литерала единичной длины.

3.7 Битовые строки

Для задания значений битовым векторам можно воспользоваться не только строковыми литералами ("111000"), но и более удобным представлением в виде битовых строк в 2-ой(B), 8-ой(O) и 16-ой(X) формах с использованием символа '_'.

Формат описания битовых строк.

bit_string_literal ::= base_specifier "[bit_value] "
    base_specifier ::= B | O | X
    bit_value ::= extended_digit { [underline] extented_digit }
        extended_digit ::= digit | letter

Битовые строки формируются как последовательность цифр 0, … , 9 и букв A, … , F (или a, … , f) между двумя кавычками. Подчеркивание в таком литерале не является значащим.

Битовые строки могут быть:

- B – бинарными;
- O – восьмеричными;
- X – шестнадцатеричными.

Вместо прописных букв B, O, X могут употребляться строчные буквы b, o, x.

Длина битовой строки - число бит в последовательности, представляющей литерал. Так, в частности, все литералы X"F_FF", O"7777", B"1111_1111_1111" имеют длину 12 бит.

3.8 Комментарии

Комментарий начинается с двух смежных дефисов и продолжается до конца строки. Компилятор игнорирует текст, начиная с символов "--" до конца строки, т.е. комментарий может включать в себя символы, не входящие в алфавит языка (в частности, русские и украинские буквы).

4 ТИПЫ ДАННЫХ

Тип – поименованное множество значений с некоторыми общими характеристиками.

Подтип – подмножество значений данного типа.

VHDL является языком со строгой типизацией. Это означает, что применение различных типов данных в одной операции является ошибкой. Средства строгого контроля типов играют ответственную роль, поскольку позволяют уточнять намерения разработчика. Язык используется для представления аппаратных проектов в самых разных вариантах, поэтому средства типизации данных приобретают здесь особенно большое значение. Например, они дают разработчику возможность представить группу линий (проводников) шины в виде массива битов или целого числа.

Каждый тип данных в VHDL имеет определенный набор принимаемых значений и набор допустимых операций. В языке предопределено достаточное количество простых и сложных типов, а также имеются средства для образования типов, определяемых пользователем.

Выделяют следующие типы данных языка VHDL:

– скалярные (scalar_type);
– составные (composite_type);
– указатели (access_type);
– файлы (file_type).

4.1 Скалярные типы

Скалярные типы создают значения, которые нельзя разбить на отдельные элементы или поля.

```
scalar_type_definition ::=
        enumeration_type_definition     -- перечисления;
        | integer_type_definition       -- целые значения;
        | floating_type_definition      -- действительные значения;
        | physical_type_definition      -- значения,  имеющие размерность.
```

Все скалярные типы и их подтипы определяются через диапазон своих значений.

4.1.1 Перечисления

Перечисления состоят из списка значений, которые могут быть символами или идентификаторами.

```
enumeration_type_definition ::=
(enumeration _literal { , enumeration _literal } )
        enumeration _literal ::= identifier | character_literal
```

Перечисления описывают по шаблону:

TYPE __enumerated_type_name **IS** (__name, __name, __name);

Например: **TYPE** LOGIC_VOLT **IS** ('0', '5', 'z', 'x');

Вводимые имена перечислений не должны совпадать с предопределенными в языке именами перечислимых типов.

Значения перечислений не должны совпадать. Весь список значений пронумерован слева направо, начиная с нуля, то есть каждое значение имеет соответствующую позицию в списке:

TYPE LIGHT **IS** (active, off, flashing);
Позиция: 0↑ 1↑ 2↑

Значения в разных перечислениях могут совпадать.

Предопределенными перечислениями являются character, bit, boolean, severity_level, специфицированные в пакете "standard":

TYPE CHARACTER **IS** (NUL, SOH, ...);
TYPE BIT **IS** ('0', 'l');
TYPE BOOLEAN **IS** (FALSE, TRUE);
TYPE SEVERITY_LEVEL **IS** (NOTE, WARNING, ERROR, FAILURE).

Примечание – значения boolean (ложный и истинный) не идентичны логическим '0' и '1'.

В пакете “std1164” предопределены перечисления std_ulogic (неразрешимый логический тип с девятизначным алфавитом) и разрешимый подтип std_logic:

TYPE STD_ULOGIC **IS** (
'U', -- неинициализированное значение;
'X', -- неопределенное значение источника с малым выходным сопротивлением;
'0', -- уровень логического нуля источника с малым выходным сопротивлением;
'1', -- уровень логической единицы источника с малым выходным сопротивлением;
'Z', -- высокоимпедансное состояние;
'W', -- неопределенное значение источника с большим выходным сопротивлением;
'L', -- уровень логического нуля источника с большим выходным сопротивлением;
'H', -- уровень логической единицы источника с малым выходным сопротивлением;
'-' -- произвольное);

Для сигналов этого типа запрещено использовать много источников сигнала. Буква u в названии типа std_ulogic сигнализирует о термине неразрешаемый (Unresolved).

Подтип std_logic определен как:

SUBTYPE STD_LOGIC **IS RESOLVED** STD_ULOGIC.

4.1.2 Целые числа

Тип целого задают через диапазон целых чисел.

integer_type_definition ::= range_constraint

Обычно этот диапазон находится между -2.147.483.648 и +2.147.483.647 (диапазон 32-разрядного целого).

Целые числа описывают по шаблону:

TYPE __range_type_name **IS RANGE** __integer **TO** __integer;

Пример описания целых чисел:

TYPE GROUP_INTEGER **IS RANGE -**1025 **TO** 1025;

Предопределенным целым типом является integer, который специфицирован в пакете "standard" как:

TYPE INTEGER **IS RANGE -**2147483648 **TO** 2147483647.

Там же специфицированы и предопределенные подтипы natural и positive по шаблону:

SUBTYPE __subtype name **IS** __type_name **RANGE** __low_value **TO** __high_value;

SUBTYPE NATURAL **IS** INTEGER **RANGE** 0 **TO** INTEGER'**HIGH**;

SUBTYPE POSITIVE **IS** INTEGER **RANGE** 1 **TO** INTEGER'**HIGH**.

Нельзя использовать имена предопределенных типов и подтипов для собственных определений.

4.1.3 Физические типы

Физические типы создают каркас для чисел с реальными размерностями, кратными некоторой базовой единице. Множество допустимых значений задается как диапазон целых чисел (базовых единиц).

Формат описания физических типов.

```
physical_type_definition ::=
        range_constraint
                units
                        base_unit_declaration { secondary_unit_declaration }
                end units
base_unit_declaration ::= identifier ;
secondary_unit_declaration ::= identifier = physical_literal;
physical_literal ::= [ abstract_literal ] unit_name.
```

При объявлении физического типа сначала указывают базовую единицу размерности, а затем последующие единицы размерности как кратные базовой или предшествующим единицам.

Предопределенным физическим типом является time, который специфицирован в пакете "standard" как:

```
TYPE TIME IS RANGE -9223372036854775808 TO 9223372036854775807
    UNITS
        fs;                              -- femtosecond
        ps     = 1000 fs;                -- picosecond
        ns     = 1000 ps;                -- nanosecond
        us     = 1000 ns;                -- microsecond
        ms     = 1000 us;                -- millisecond
        sec    = 1000 ms;                -- second
        min    = 60 sec;                 -- minute
        hr     = 60 min;                 -- hour
END UNITS;
```

Физические типы при описании устройств на ПЛИС обычно не применяются.

4.1.4 Числа с плавающей точкой

Числа с плавающей запятой обеспечивают приближения к вещественным числам. Числа с плавающей запятой применяют для моделей, в которых погрешность вычисления значений не важна или не определена. Тип с плавающей точкой задается через диапазон чисел с плавающей точкой.

floating_type_definition ::= range_constraint

Примеры описания чисел с плавающей точкой:

TYPE RESULT **IS RANGE** 0.0 **TO** 11063.5;

SUBTYPE P_RESULT **IS** RESULT **RANGE** 2765.88 **TO** 8297.63.

Предопределенным типом с плавающей точкой является real, который специфицирован в пакете "standard" как:

TYPE REAL **IS RANGE** **-**1.7E38 **TO** 1.7E38.

Тип real обеспечивает 64-разрядное представление чисел с плавающей точкой (1 разряд – знак числа, 11 разрядов – порядок, 52 разряда – мантисса).

Числа с плавающей запятой при описании устройств на ПЛИС обычно не применяются из-за сложности конструктивной реализации.

4.2 Составные типы

Составной тип – это группа значений под одним именем.

```
composite_type_definition ::=
    array_type_definition          -- массивы;
    record_type_definition         -- записи.
```

4.2.1 Массивы

Массивы объединяют элементы одного типа. Массивы могут иметь любую размерность. Тип элемента массива не может быть file. Можно задать тип массива как с неопределенными, так и с определенными границами. Задание типа с неопределенными границами дает возможность создавать массивы, имеющие один и тот же тип, но различные границы индексов.

array_type_definition ::=
unconstrained_array_definition | constrained_array_definition
unconstrained_array_definition ::=
array (index_subtype_definition { , index_subtype_definition })
of element_subtype_indication
constrained_array_definition ::=
array index_constraint of element_subtype_indication
index_subtype_definition ::=
type_mark range <> index_constraint ::= (discrete_range
{ , discrete_range })
discrete_range::=
discrete_subtype_indication | _range

Массивы можно описывать по шаблону:

TYPE __array_type_name **IS ARRAY** (INTEGER **RANGE** <>) **OF** __type_name;

TYPE __array_type_name **IS ARRAY** (__integer **DOWNTO** __integer) **OF** __type_name;

Пример двух эквивалентных описаний двумерного массива 3*4:

TYPE ARRAY_M **IS ARRAY** (1 **TO** 3, 7 **DOWNTO** 4) **OF** POSITIVE;

TYPE ARRAY_M **IS ARRAY** (INTEGER **RANGE** 1 **TO** 3, INTEGER **RANGE** 7 **DOWNTO** 4) **OF** POSITIVE;

Каждая пара границ массива должна иметь одинаковый тип. Элемент массива в свою очередь может быть массивом.

Задание типа с неопределенными границами – очень удобная возможность для описания аппаратуры. Это дает как полную совместимость объектов данного типа, так и большую гибкость в параметризации описания. Примерами таких типов могут служить предопределенные типы bit_vector и string, специфицированные в пакете "standard", типы signed и unsigned, специфицированные в пакете "arith", а также типы std_ulogic_vector и std_logic_vector, специфицированные в пакете "std1164":

TYPE BIT_VECTOR **IS ARRAY** (NATURAL **RANGE** <>) **OF** BIT;

TYPE STRING **IS ARRAY** (POSITIVE **RANGE** <>) **OF** CHARACTER;

TYPE UNSIGNED **IS ARRAY** (NATURAL **RANGE** <>) **OF** STD_LOGIC;

TYPE SIGNED **IS ARRAY** (NATURAL **RANGE** <>) **OF** STD_LOGIC;

TYPE STD_ULOGIC_VECTOR **IS ARRAY** (NATURAL **RANGE** <>) **OF** STD_ULOGIC;

TYPE STD_LOGIC_VECTOR **IS ARRAY** (NATURAL **RANGE** <>) **OF** STD_LOGIC;

Направление и границы диапазона индексов не содержатся в определении указанных типов и должны быть указаны непосредственно при объявлении объектов данных типов.

Тип unsigned предназначен для описания беззнаковых чисел, которые могут быть больше или равны нулю. Левый крайний бит имеет наибольший вес. Например:

UNSIGNED ("0110") представляет +6;

UNSIGNED ("1010") представляет +10.

Тип signed предназначен для описания знаковых чисел, которые могут быть положительными, отрицательными или равными нулю. Левый крайний бит указывает знак числа. Например:

SIGNED ("0110") представляет +6;

SIGNED ("1010") представляет **-**6.

4.2.2 Записи

Записи – это составной тип данных, элементы которых могут иметь различные типы.

```
record_type_definition ::=
    record
        element_declaration
        { element_declaration }
    end record
     element_declaration ::= identifier_list: element_subtype_definition;
          element_subtype_definition ::= subtype_indication
```

Все имена элементов (полей) записи должны быть различными.

Пример описания данных составного типа:

```
TYPE CLOCK_TIME IS RECORD
    hour              : INTEGER RANGE 1 TO 24;
    minute, second    : INTEGER RANGE 1 TO 60;
    ampm              : HALF_DAY;
END RECORD;
```

4.3 Указатели

Указатель – это объект (переменная), значение которого есть адрес ячейки в оперативной памяти, где хранятся данные определенного типа, исключая тип файл, и не имеющие имени. Доступ к этим данным возможен только через указатели. Конструкция access_type_definition позволяет объявить необходимый тип указателя.

Указатели при описании устройств на ПЛИС обычно не применяются.

4.4 Файлы

Файл – это специально организованная структура данных, которая имеет имя и рассматривается в процессе пересылки и обработки как единое целое на диске. Перед тем как работать с файлами, необходимо объявить типы этих файлов.

Определение.

file_type_definition ::= file of type_mark

После этого можно объявлять и сами файловые объекты.

Пример описания:

TYPE ROM_DATA **IS FILE OF** INTEGER; -- объявление файлового типа.

FILE ini_rom : ROM_DATA **IS** "$HOME/INI/ini_rom_file"; -- файловый объект, куда будут читаться данные из файла ini_rom_file.

Объявление файлового объекта вызывает неявное соответствующее определение всех подпрограмм доступа к нему. В примере подразумеваемые определения для VHDL-87 будут иметь вид:

PROCEDURE read (ini_rom : **INOUT** ROM_DATA; value **OUT** : INTEGER);

PROCEDURE write (ini_rom : **INOUT** ROM_DATA; value **IN** : INTEGER);

FUNCTION endfile (ini_rom : **IN** ROM_DATA) **RETURN** BOOLEAN;

Файлы открываются только на чтение или только на запись (в примере файл ini_rom_file открывается на чтение). Функция endfile возвращает false, если нет возможности прочесть очередную запись. Процедура read читает очередную запись на переменную value. Процедура write пишет значение value в конец файла.

Файлы при описании устройств на ПЛИС обычно не применяются.

5 ИМЕНА

При любом объявлении объекта определяют идентификатор, который допускает ссылку на такой объект. Однако не всегда возможно указать объект или часть его, используя только его идентификатор. Более общая форма ссылки к объектам - *имя*.

Существуют шесть форм имен.

```
name ::=
        simple_name        -- простое имя;
        | operator_symbol  -- символ оператора;
        | selected_name    -- селективное имя;
        | indexed_name     -- индексное имя;
        | slice_name       -- вырезка имени;
        | attribute_name   -- имя атрибута.
```

5.1 Простое имя

Простое имя – это идентификатор. Имя может быть создано:

- посредством объявления;
- при использовании метки;
- при создании цикла;
- при создании псевдонима объекта.

Формат описания простого имени.

simple_name ::= identifier
identifier ::= letter {[_] letter | digit}

Примеры описания простых имен:

ARCHITECTURE behavel_shifter **OF** shifter **IS** -- behavel_shifter – простое имя.

SIGNAL reg_com: BIT_VECTOR (0 **TO** 15); -- reg_com – простое имя.

proc_shifter: **PROCESS BEGIN** -- proc_shifter – имя поименованного процесса.

FOR k **IN** 1 **TO** 12 **LOOP** -- k – имя переменной цикла.

VARIABLE e: BIT_VECTOR (0 **TO** 3); -- создание псевдонима.

ALIAS f : BIT_VECTOR (0 **TO** 3) is **E;**

e := "0000"; f:="1111";

5.2 Символ оператора

Символ оператора представляет собой имена только предопределенных операторов в их изначальном или перегруженном смысле.

operator_symbol ::= string_literal

Примеры описания символа оператора:

FUNCTION "+"(L, R : BOOLEAN) **RETURN** BOOLEAN **IS** -- "+" – символ (имя) предопределенного оператора плюс.

5.3 Селективное имя

Селективное имя служит для указания объекта, объявленного внутри другого объекта или в библиотеке.

Формат описания селективного имени.

selected_name ::= prefix . suffix
prefix ::= name | function_call
suffix ::= simple_name | character_literal | operator_symbol | all

Селективное имя состоит из префикса, который является названием составного объекта, точки (.) и суффикса, который может быть простым именем, символьной константой, символом оператора или зарезервированным словом **all.** В последнем случае имя обращается ко всем объектам, объявленным в составном объекте.

Примеры описания селективных имен:

TYPE ARRAY_3 **IS ARRAY** (POSITIVE **RANGE** <>, POSITIVE **RANGE** <>, POSITIVE **RANGE** <>) **OF** INTEGER;
TYPE MY_RECORD **IS RECORD**
    my_array_3 : ARRAY_3 (1 **TO** 4, 0 **TO** 7, 7 **DOWNTO** 0);
    my_vec   : BIT_VECTOR(1 **TO** 16);
END RECORD;
VARIABLE A_B_C: MY_RECORD;

Здесь селективные имена могут иметь вид:
A_B_C.my_array_3 или A_B_C.my_vec.

При необходимости применения всего содержимого, например, библиотеки "arithmetic", следует указать зарезервированное слово **ALL** (**USE** arithmetic.**ALL**;).

5.4 Индексное имя

Индексное имя указывает элемент массива, который обозначен списком выражений.

Формат описания индексного имени.

indexed_name ::= prefix (expression { , expression })

Число выражений в списке должно соответствовать размерности массива, а значения выражений должны находиться в пределах соответствующих индексных диапазонов.

Для приведенных выше переменных индексными именами являются:
A_B_C.my_array_3(ss-l, ss*2, 4+ss)   A_B_C.my_vec(8).

Индексное имя имеет ту же самую форму, что и вызов функции. В таком случае интерпретация имени зависит от контекста. Если интерпретация такого имени невозможна, то имя неоднозначно.

5.5 Вырезка имени

Вырезки имен позволяют манипулировать с частями одномерных массивов, то есть префикс здесь должен иметь тип одномерного массива.

Формат описания вырезки имени.

slice_name ::= prefix (discrete_range)

Например, для приведенных выше переменных допустимо присваивание:

A_B_C.my_vec := my_big_vec(6**DOWNTO**1); --my_big_vec(6 DOWNTO 1) и есть вырезка.

5.6 Имя атрибута

Для удобства создания моделей в язык VHDL введено понятие атрибута. Атрибут – это значение, связанное с поименованным объектом. В языке существует ряд предопределенных атрибутов для таких объектов, как массивы, блоки, сигналы, типы (Array, Block, Signal, Type). Кроме этого, можно объявить, а затем специфицировать (связать с поименованными объектами и задать значения) свои (пользовательские) атрибуты. Пользовательские атрибуты можно создать для устройств, архитектурных тел, конфигураций, процедур, функций, пакетов, типов, подтипов, констант, сигналов, переменных, компонентов, меток (Entity Declaration, Architecture Body, Configuration, Procedure, Function, Package, Type, Subtype, Constant, Signal, Variable, Component, Label).

Формат описания имени атрибута.

attribute_name ::= prefix ' attribute_designator [(expression)]
  attribute_designator ::= attribute_simple_name

Пример описания имени атрибута:
TYPE COLOUR **IS** (white, blue, red);
Значение выражения COLOUR'**VAL**(2) вернет red.

5.7 Видимость и область действия имен

Каждому месту в тексте программы соответствует некоторое видимое множество имен, которое должно быть непротиворечивым – однозначно воспринимаемым. И наоборот, введение (объявление) имени в том или ином месте текста определяет область в тексте программы, где это объявление действительно.

Правила видимости и установки границ действия имен в VHDL те же, что и в большинстве языков программирования.

6 ВЫРАЖЕНИЯ

Выражение – это формула, которая используется для вычисления нового значения, или одиночный термин, имеющий значение.

Выражение можно рассматривать как совокупность бинарных выражений, имеющих левый операнд, правый операнд и оператор, связывающий левый и правый операнды (x+y). В результате вычисления бинарного выражения получается новый операнд, который вступает далее в бинарные отношения со своими соседями. Унарное выражение рассматривают как бинарное выражение, в котором отсутствует левый операнд. Константы могут употребляться в выражениях.

6.1 Операторы

Предопределенные операторы языка рассмотрим по группам. При необходимости эти операторы можно перегружать: переопределять их семантику и расширять область их применимости для различных типов. Список всех операторов языка VHDL приведен в таблице 6.1.

Строки в таблице 6.1 располагаются в порядке старшинства (от низшего к высшему) операторов. Операторы, находящиеся в одной строке, обладают одинаковым старшинством (приоритетом). Таким образом, операции нижней строки в таблице 6.1 обладают наибольшим приоритетом и выполняются первыми.

Таблица 6.1 – Классификация операций

Класс операций	Операции						
Логические	and	nand	or	nor	xor	xnor (VHDL-93)	not
Сравнения	=	/=	<	<=	>	>=	
Сложения и конкатенации	+	-	&				
Присвоение знака	+	-					
Сдвига	sll	srl	sla	sra	rol	ror	
Умножения и деления	*	/	mod	rem			
Смешанные	**	abs					

6.1.1 Логические операторы

Определение.

logical_operator ::= and | nand | or | nor | xor | nxor | not

Логические операторы выполняют следующие функции: and – логическое 'и'; nand – логическое 'и-не'; or – логическое 'или'; nor –логическое 'или-не'; xor – логическое 'исключающее или'; nxor – логическое 'исключающее или-не'; not – логическое отрицание.

В логических бинарных выражениях массивы должны быть одинаковой длины. Вычисления здесь производятся над парами элементов, равно отстоящими от левой границы. Результатом является массив, индексация элементов в котором совпадает с индексацией элементов левого операнда, то есть их подтипы совпадают.

Оператор not является логическим унарным.

Логические операторы выполняются для следующих типов данных:

− boolean;

− bit, bit_vector;

− std_logic, std_logic_vector;

− std_ulogic, std_ulogic_vector.

Логические операторы and, nand, or, nor, xor, nxor имеют одинаковое старшинство и выполняются слева направо в выражениях. Операция not имеет более высокое старшинство и выполняется прежде других операторов. В сложных логических выражениях порядок выполнения операторов регулируется скобками. Рекомендуется применять скобки в затруднительных случаях. Например, для выражения

Z <= A **AND NOT** B **OR** C;

будет только отрицание B, в выражении

Z <= A **AND NOT** (B **OR** C);

будет отрицание подвыражения B **OR** C, находящегося в скобках.

В языке VHDL можно применять три способа вызова операторов:

− префиксный;

− обычный (инфиксный);

− с использованием квалифицирующего выражения.

Три способа вызова операторов применены в примере, описанном ниже.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY operat IS
        PORT (      op1, op2            : IN std_logic_vector(3 downto 0);
                    res1, res2, res3    : OUT std_logic_vector(3 downto 0));
END operat ;

ARCHITECTURE maxpld OF operat IS
BEGIN
        res1 <= op1 AND op2;
        res2 <= "AND"(op1,op2);
        res3 <= std_logic_vector'(op1 AND op2);
END maxpld;
```

6.1.2 Операторы сравнения

Определение.

relational_operator ::= = | / = | < | <= | > | > =

Операторы сравнения предназначены для выполнения следующих операций: = – равно; /= – не равно; < – меньше; <= – меньше-равно; > – больше; >= – больше-равно.

Операторы сравнения выполняются для следующих типов данных:

– std_logic_vector;
– std_ulogic_vector;
– signed;
– unsigned;
– integer.

6.1.3 Операторы сдвига

В VHDL-93 были введены предопределенные операторы сдвига. Операторы сдвига можно использовать, когда левым операндом является одномерный массив из элементов типа bit (bit_vector) или boolean, а правым операндом является любое неотрицательное целое.

shift_operator ::= sll | srl | sla | sra | rol | ror

sll (shift left logical) – сдвиг левый логический. Освобождающиеся элементы массива заполняются значением, определенным по умолчанию для данного типа (для типа bit это '0').

srl (shift right logical) – сдвиг правый логический. Освобождающиеся элементы массива заполняются значением, определенным по умолчанию для данного типа.

sla (shift left arithmetic) – сдвиг левый арифметический. Освобождающиеся элементы заполняются значениями крайнего правого элемента массива.

sra (shift right arithmetic) – сдвиг правый арифметический. Освобождающиеся элементы заполняются значениями крайнего левого элемента массива.

rol (rotate left logical) – сдвиг циклический левый логический.

ror (rotate right logical) – сдвиг циклический правый логический.

Операторы сдвига имеют одинаковое старшинство с мультипликативными операторами.

6.1.4 Аддитивные операторы

Определение.

adding_operator ::= + | - | &

Аддитивные операторы предназначены для выполнения операций суммирования, вычитания и конкатенации.

Операторы суммирования и вычитания выполняются для следующих типов данных:

– std_logic_vector;
– std_ulogic_vector;
– integer;
– signed;
– unsigned.

Один из операндов может иметь тип integer или std_ulogic.

Оператор конкатенации & служит для объединения двух одномерных массивов, одномерного массива и скаляра, двух скаляров. Любой скаляр здесь рассматривается как одномерный массив, элементы которого индексируются в диапазоне 1 To 1. Скаляры и элементы массивов, участвующие в конкатенации, могут быть любого типа, но эти типы должны совпадать.

В результате конкатенации образуется одномерный массив большей длины. Индекс этого массива непрерывен и значение его левого индекса совпадает со значением левого индекса левого операнда. Конкатенация является удобным средством при описании устройств на регистровом уровне для объединения различных разрядов нескольких регистров в один вектор.

Пример.

```
PROCESS (clk_sig)
   VARIABLE register_5_bit: BIT_VECTOR (5 TO 9);
   VARIABLE register_4_BIT: BIT_VECTOR (6 DOWNTO 3);
   VARIABLE register_9_BIT: BIT_VECTOR (9 DOWNTO 1);
BEGIN
   register_9_bit := register_5_bit & register_4_bit;
   register_5_bit := '1' & register_4_bit;
   register_4_bit := '1'&'0'&'1'&'0';    --эквивалентно: register_4_bit:="1010"
END PROCESS;
```

В VHDL-93 допустимо объединять только восходящие (n1 To n2), или нисходящие (nl Downto n2) массивы. В первом случае левый индекс результата совпадает с левым индексом левого операнда, во втором случае правый индекс результата совпадает с правым индексом правого операнда.

6.1.5 Мультипликативные операторы

Определение.

multiplying_operator ::= * | / | mod | rem

Мультипликативные предназначены для выполнения операций умножения, деления, нахождения модуля и остатка целочисленного деления.

Целочисленное деление отвечает тождеству: (-A)/B = - (A/B) = A/(-B).

Выражение (A mod B) имеет знак B, абсолютное значение меньше абсолютной величины B, и определяется отношением: A = B*целое + (A mod B).

Выражение (A rem B) имеет знак A, абсолютное значение меньше абсолютной величины B, и в терминах целочисленного деления определяется отношением: A = (A/B)*B + (A rem B).

6.1.6 Знаковые операторы

Определение.

sign ::= + | -

Знаковые операнды допустимы для операндов, имеющих скалярные типы. Они не могут следовать непосредственно за мультипликативными, аддитивными операторами или смешанным оператором "экспонента".

Эти выражения ошибочны:
(A*-B), (C/+D), (F**-G), (H+-I).
Здесь необходимы скобки:
A * (- B), C / (+ D), F ** (- G), H + (-I).

6.1.7 Смешанные операторы

Определение.

miscellaneous_operator ::= ** | abs

Смешанные операнды предназначены для возведения числа в степень или получения абсолютного значения.

6.2 Операнды

Определение.

```
primary ::=
        name | literal | aggregate | function_call |
        qualified_expression | type_conversion | allocator | (expression)
```

Перечисленные выше операнды могут участвовать в выражениях.

6.2.1 Операнд Name

В качестве операнда можно использовать:
– имя объекта;
– имя атрибута, возвращающего значение;
– значение (имя перечислимого типа).
Пример attribute_name.

```
PROCESS
   VARIABLE x: INTEGER;
   TYPE info IS ARRAY (INTEGER RANGE <>) OF INTEGER;
   VARIABLE z: info (1 TO 10);
BEGIN
   x := z'RIGHT +10;
```

END PROCESS;

Атрибут right возвращает значение правой границы массива "z", то есть 10, при этом "x" принимает значение 20.

6.2.2 Операнд Literal

Литерал служит для задания значений объектов языка. В VHDL имеется пять типов литералов.

```
literal ::=
        numeric_ literal          -- числа (целые и реальные): 0, 2E4, 3.14 ns;
       |enumeration_literal       -- перечисления: '0', ram;
       | string_ literal          -- строки (строковый литерал): "are not";
       | bit_string_literal       -- битовые строки: b"111_111" o"7_7", x"OF";
       | null                     -- значение указателя (access_type) в никуда.
```

Числа могут быть представлены в виде абстрактных и физических литералов.

```
numeric_literal ::=
abstract_literal
| physical_literal
```

Литералы перечисления – это литералы, применяемые для описания типа данных - перечисления. В качестве этих литералов применяют идентификаторы и символьные литералы.

Строковые литералы и битовые строки были рассмотрены в 4 разделе.

Литерал null представляет собой нулевое значение для любого типа.

Примеры.

VARIABLE result: INTEGER := 1024*8; -- 1024 и 8 - литералы; 1024*8 - статическое выражение.

VARIABLE answer: CHARACTER := 'x'; -- 'x' - символьный литерал.

CONSTANT STRING_l: STRING := "011" & "true"; -- "011" и "true" - литералы; "011" & "true" – статическое выражение, которое присваивается строковой константе STRING_l.

6.2.3 Операнд Aggregate

Агрегат – это базовая операция, объединяющая одно или нескольких значений в массив или запись.

Элементы этого объединения связываются (ассоциируются) при вычислениях (например, при присвоении агрегата одномерному массиву) позиционно (в этом случае конструкция choices => отсутствует), или поименованно.

```
aggregate ::= (element_association { , element_association })
element_association ::= [choices =>] expression
choices ::= choice { | choice }
choice ::=
simple_expression
```

| discrete_range
| element_simple_name
| others.

При именованном присвоении сначала формируется цель – множество элементов массива, затем, после символов "=>" – то, что надо записать в эту цель, и далее, через разделитель ", ", формируется следующая цель.

Позиционное связывание в агрегатах должно предшествовать поименованным ассоциациям. Поименованная ассоциация "Others =>" может быть использована (при необходимости) только в конце агрегата. Допустимо только однократное связывание. Если агрегат имеет единственное значение, то оно должно быть связано поименованно.

Опишем, для примера, присвоение значений переменной:

VARIABLE q : BIT_VECTOR (0 **TO** 3).

Возможно 7 форм присвоения значений с помощью агрегатов:

1 позиционная
q := ('0', '1', '1', '0');
2 именованная
q := (0=>'0', 2=> '1', 1=> '1', 3=> '0');
3 смешанная
q := ('0', '1', 1=> '1', 3=> '0');
4 с использованием вырезки имен
q := (0 =>'0', 1 TO 2 =>'1', 3=> '0');
5 с использованием зарезервированного слова OTHERS (слово OTHERS можно использовать в случае применения только позиционной или именованной ассоциации)
q := (1 TO 2 =>'1', OTHERS => '0');
q := ('0', '1', OTHERS => '0');
6 с применением только слова OTHERS, если необходимо всем разрядам установить одно и то же значение
q := (OTHERS => '0');
q := (OTHERS => '1');
7. с перечислением разрядов, которым следует установить одинаковые значения. Перечисляемые разряды отделяют друг от друга вертикальной чертой
q := (0|3 =>'0', 1|2 => '1').

6.2.4 Операнд Function Call

Вызов функции приводит к выполнению тела функции. Он определяет имя функции, которая будет вызвана и фактические параметры, которые должны быть связаны с формальными параметрами функции. В результате выполнения функции формируется значение с типом, определенным при объявлении функции.

function_call:: = function_name [(actual_parameter_part)]

Каждому формальному параметру должен соответствовать фактический параметр.

6.2.5 Операнд Qualified Expression

Позволяет однозначно указывать тип или подтип операнда.

qualified_expression ::= type_mark'(expression)| type_mark'aggregate

Необходимость использования квалифицирующего выражения возникает при наличии перегруженных (overload) операторов, функций или операндов, то есть когда, например, существуют одинаково именованные операторы, работающие с различными типами операндов и, естественно, имеющие различное функционирование.

6.2.6 Операнд Type Conversion

Конверсия типа используется для перевода одного типа данных в другой тип данных.

type_conversion ::= type_mark (expression)

6.2.7 Операнд Allocator

Предназначен для создания анонимных объектов, доступ к которым осуществляется через указатели (access) на эти объекты.

allocator ::= new subtype_indication | new qualified_expression

Использование динамически размещаемых объектов удобно не только при задании необходимого объема аппаратуры (ROM, RAM и пр.), но и при моделировании таких структур, как FIFO, LIFO. Сами динамически размещаемые объекты, естественно, должны быть детерминированных размеров.

Обычно в САПР проектирования устройств на ПЛИС не поддерживается.

6.2.8 Статическое выражение

Для вычисления значений констант, параметров настройки и начальных значений всех других объектов языка используются выражения, которые являются статическими (Static Expression), то есть вычисляются единожды в процессе выполнения программы. Кроме этого, статические выражения участвуют при определении типов.

6.2.9 Универсальное выражение

Универсальное выражение (Universal Expression) – это выражение, тип результата которого есть универсальное целое или универсальное действительное. Обычно в САПР не поддерживается.

7 ПРЕДСТАВЛЕНИЕ СИСТЕМЫ В VHDL

Система предназначена для выполнения заданных преобразований, для чего она должна:

– получить некоторые входные данные от окружающей ее среды;

– выполнить преобразования;

– вывести некоторые данные.

Из анализа приведенного следует, что система должна иметь связи с окружающей ее средой, которые называют интерфейсом.

Интерфейс системы описан в VHDL его сущностью (Entity), которая является основной единицей проекта для любой системы.

Выполнение преобразований осуществляется внутренней частью системы или телом (Body), которая называется архитектурой (Architecture).

Для получения дополнительных возможностей систем применяют пакеты (Package) и библиотеки (Library).

7.1 Общая структура описания проекта системы

Структуру проекта в общем случае описывают по шаблону:

```
-- Context Clauses
      -- LIBRARY __library_name;                         -- Library Clause
      -- USE __library_name.__package_name.ALL;     -- Use Clause
--  Library Units
      --  Package Declaration (optional)
      --  Package Body (optional)
      --  Entity Declaration
      --  Architecture Body
```

В структуре проекта последовательно описывают применяемые стандартные библиотеки (Library Clause) и входящие в них пакеты (Use Clause), пакеты пользователя (Package Declaration и Package Body), интерфейс объекта (Entity Declaration) и его архитектуру (Architecture Body).

В самом простейшем виде описание может содержать только Entity Declaration и Architecture Body.

7.2 Сущность проекта системы

Любой проект системы в VHDL следует начинать с декларации сущности (Entity Declaration), которую описывают следующим образом:

```
ENTITY __entity_name IS
        GENERIC ( __parameter_name    : __TYPE    := __default_value);
        PORT       (__port                    :__mode __TYPE );
 END ENTITY __entity_name;
```

Имя сущности (__entity_name), формально названное идентификатором, предназначено для целей документирования. Рекомендуется задавать имя с учетом функций, выполняемых системой.

Сущность обеспечивает спецификацию интерфейса системы и обычно включает в себя два элемента:

– параметры настройки системы (Generic);

– порты связи (Port), которые передают информацию к системе и от нее (системные вводы и выводы).

Параметр настройки системы (Generic) представляет собой канал статической информации, которая будет сообщена системе из окружающей среды. Он используется для описания постоянных значений. Например, он может задавать разрядность шины данных, адреса, направления счета, модуль счета и т.д.

Описание параметров настройки состоит из ключевого слова Generic и списка параметров, заключенных в скобки , например:

```
GENERIC (WIDTH          : INTEGER   := 8;
         DEPTH          : INTEGER   := 15;
         INDATAWIDTH    : POSITIVE  := 8);
```

Если значение не определено и система используется в иерархической структуре, то значение должно быть определено через текущий компонент (Component Instantiation).

Порт связи (Port) язык VHDL определяет как канал для динамической связи между сущностью и окружающей средой. На практике эти каналы (сигналы) формируют интерфейс системы, поэтому каждый порт должен быть точно определен.

Каждый порт определяется своим предложением порта (Port Clause), например:

```
PORT (SIGNAL i0    : IN    BIT; -- входные данные
      SIGNAL i1    : IN    BIT; -- входные данные
      SIGNAL sel   : IN    BIT; -- выбор адреса
      SIGNAL y     : OUT   BIT); -- выходные данные
```

Port Clause состоит из следующих элементов:

– ключевое слово Signal (необязательно);

– имя порта;

– Mode порта;

– Type порта;

– начальное значение, которому предшествует символ := (необязательно);

– комментарий, описывающий порт (необязательный, но рекомендуемый).

Для каждого порта определяют его режим (Mode), указывающий направление потока информации в этой точке коммутации.

Имеется пять доступных мод: In, Out, Inout, Buffer, Linkage.

Мода In – интерфейсный объект можно только читать (изнутри).

Мода Out – интерфейсный объект можно переустанавливать (изнутри), но не читать.

Мода Inout – интерфейсный объект можно читать и переустанавливать.

Мода Buffer – интерфейсный объект должен переустанавливаться (изнутри) только одним источником. В отличие от интерфейсного объекта с модой Out его можно читать.

Мода Linkage – интерфейсный объект можно читать и переустанавливать, при этом все другие интерфейсные объекты, связанные с ним, также должны иметь моду Linkage.

По умолчанию используется мода In.

Для интерфейсных объектов Constant возможна только мода In.

Сущность, как большинство конструкций VHDL, заканчивается ключевым словом End.

7.3 Архитектура проекта системы

В VHDL архитектура проекта системы (Architecture Body) может быть описана как:

```
ARCHITECTURE architecture_name OF entity_name IS
        architecture_declarations
BEGIN
        concurrent_statements
END [ ARCHITECTURE ] [ architecture_name ];
```

Архитектура описывает внутренние отношения между портами ввода и вывода объекта. Она состоит из двух частей: объявления и параллельных предложений.

Первая (декларативная) часть архитектуры может содержать объявления типов, подтипов, сигналов, констант, подпрограмм (функций и процедур), атрибутов, компонент и групп.

Параллельные предложения в теле архитектуры определяют отношения (связи) между входами и выходами. Эти отношения могут быть определены, используя различные типы предложений и стили: структурный, функциональный или смешанный.

Структурное описание основано на конкретизации компонентов и линий связи между ними и интерфейсом. Оно позволяет создавать иерархические проекты: от простых схем до очень сложных компонентов, описывающих полные подсистемы.

Функциональное описание определяет алгоритм преобразования входных сигналов в выходные. Данное описание может быть двух видов: потоковое и поведенческое.

Архитектура может содержать предложения, которые одновременно содержат структурное и функциональное описание. Такое описание архитектуры называют смешанным.

7.4 Предложения VHDL

Рекомендуемая структура проекта с учетом применяемых основных параллельных и последовательных предложений и места их расположения приведена на рисунке 7.1.

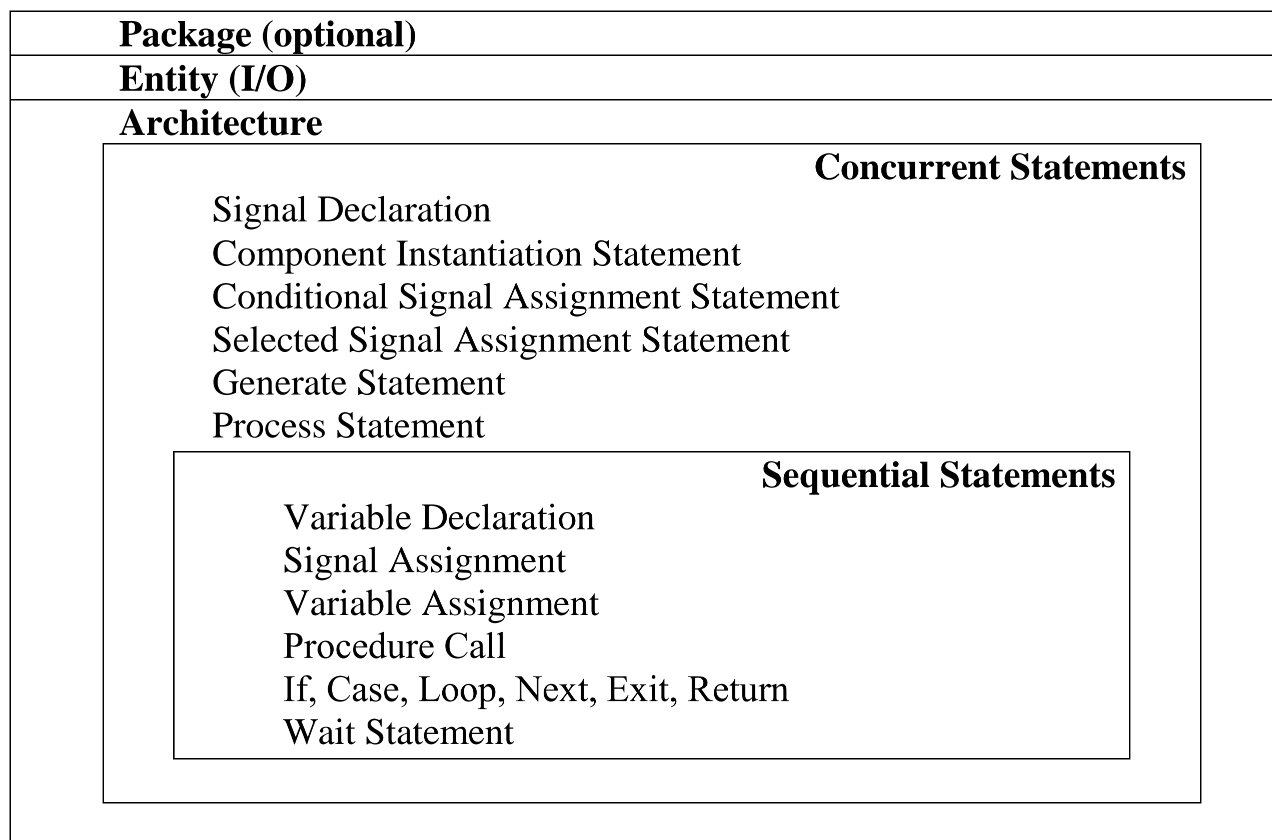

Рисунок 7.1 – Общая структура VHDL-описания

Последовательные предложения (Sequential Statements) выполняются в порядке их появления в VHDL-коде.

Порядок выполнения параллельных предложений не связан с порядком их появления внутри архитектурного тела. Параллельные предложения активизируются сигналами, которые употребляются для связи параллельных предложений.

8 СТРУКТУРНОЕ ОПИСАНИЕ ЦИФРОВЫХ УСТРОЙСТВ

При структурном описании цифровых устройств должны быть выполнены следующие операции:

– определен интерфейс связи с внешней средой;

– декларированы входящие компоненты;

– заданы внутренние сигналы, определяющие необходимые электрические линии связи в устройстве;

– специфицированы все вхождения каждого компонента и линии связи между ними и интерфейсом.

Интерфейс связи с внешней средой указывают в Entity Declaration.

Декларацию компонентов осуществляют с помощью предложения Component Declaration по шаблону:

```
COMPONENT __component_name
    GENERIC(__parameter_name : STRING   := __default_value;
            __parameter_name : INTEGER := __default_value);
    PORT(__input_name, __input_name       : IN      STD_LOGIC;
         __bidir_name, __bidir_name       : INOUT   STD_LOGIC
         __output_name, __output_name     : OUT     STD_LOGIC);
END COMPONENT;
```

Предложение Component Declaration объявляет компоненты со всеми именами и параметрами интерфейсных объектов (параметрами настройки и сигналами портов).

Ключевое слово Generic служит для передачи параметров, чаще всего таких, как разрядность и других настраиваемых параметров.

Список портов, начинающийся с ключевого слова Port, определяет имя, направление (режим – Mode) и тип каждого порта. Тип порта – это тип сигнала, ассоциированного с данным портом. Направление порта может быть: In – входной порт, Out – выходной, Inout – двунаправленный.

Компонент может быть определен, например, в пакете или архитектуре. Если компонент применяют в архитектуре, то он должен быть объявлен в декларативной части перед ключевым словом Begin. В таком случае компонент можно использовать только в архитектуре. Более универсальный подход состоит в том, чтобы объявить компонент в пакете. Такой компонент виден в любой архитектуре, которая использует этот пакет.

8.1 Сигналы в VHDL

Электрические сигналы играют ключевую роль в действии электронных элементов. Фактически трудно вообразить прибор, который не посылает или не получает некоторые сигналы. По этой причине сигналы – наиболее существенные объекты любого электронного прибора. В языке VHDL сигналы играют важную роль в описании связи между узлами или блоками. В отличие от пере-

менных в классических языках программирования VHDL-сигналы содержат информацию о прошлых, настоящих и будущих значениях. Эта информация называется историей сигнала. Она позволяет проводить анализ временных зависимостей внутри VHDL системы.

Электронные проекты основаны на компонентах и сигнальных линиях, соединяющих эти компоненты. Сигнальные линии могут быть выполнены одноразрядными и многоразрядными.

Одноразрядная связь представлена сигнальной линией, которая имеет единственное двоичное значение в любой момент. Примером такого сигнала могут быть импульсы синхронизации.

Некоторые проекты включают многоразрядные линии связи, названные шинами или векторами, которые передают информацию как комбинацию некоторых двоичных значений (например, шины адреса или данных).

Спецификация VHDL, в простейшем случае, допускает два типа сигналов, которые названы bit (для единичных сигналов) и bit_vector (для шин). В обоих случаях каждая сигнальная линия может иметь значение '0' или '1'.

В случае единичного сигнала следует указать, что этот сигнал имеет тип bit.

Описание шин – более комплексно. Во-первых, следует указать, что это – шина, прибавляя слово: _vector. При этом формируется утверждение bit_vector. Во-вторых, в списке должна быть явно задана разрядность шины и порядок следования битов, например, является ли разряд номер 7 старшим значащим битом (MSB), а номер 0 – самый младшим битом (LSB) в шине, или наоборот. Эта информация обеспечивается через диапазон вектора, которая всегда указывается в круглых скобках, например (7 Downto 0). Диапазон вектора определяется в VHDL использованием двух ключевых слов: To или Downto. Первое служит для определения увеличивающегося диапазона, а второе – уменьшающегося диапазона.

В VHDL выделяют сигналы внутренние и внешние.

Внешние сигналы – сигналы, которые соединяют систему с внешним миром. Другими словами, они формируют интерфейс системы.

Внутренние сигналы, которые не видимы из внешнего мира, являются частью внутренней архитектуры системы. Они обеспечивают связь между внутренними узлами и устройствами и могут быть описаны предложением Signal Declaration Statement по шаблону:

SIGNAL __signal_name : __type_name;

Вхождения компонентов и линии связи между ними и интерфейсом специфицируют с помощью предложения Component Instantiation Statement.

Предложение Component Instantiation Statement создает вхождения компонента внутри устройства, подсоединяет сигналы к этому вхождению и связывает (при необходимости) параметры настройки в устройстве с параметрами настройки вхождения компонента по шаблону:

__instance_name: __component_name

```
GENERIC MAP (__parameter_name => __parameter_value,
             __parameter_name => __parameter_value)
PORT MAP (__component_port => __connect_port,
          __component_port => __connect_port);
```

__instance_name – обязательная метка для компонента, так как каждый компонент может быть реализован более одного раза в каждой архитектуре.

В секции Generic Map уточняются параметры компонента для реализуемого проекта.

В секции Port Map указываются линии связи или сигналы между компонентами и интерфейсом.

Сигналы, подключаемые к компоненту, могут быть описаны двумя способами:

– Named Port Association (соответствие имен), при котором каждый порт подключают к сигналу, используя явное имя порта. Порядок описания портов не имеет значения. Соответствие сигналов и портов указывают символом “ =>”;

– Positional Port Association (соответствие позиций), при котором сигналы перечисляются в порядке следования портов в декларации компонента. Таким образом, связи между сигналами и портами определяются их позицией. К недостаткам данного способа стоит отнести сложность определения связей между сигналами и портами при большом количестве сигналов.

8.2 Описание нерегулярных структур

Проекты в САПР Quartus могут быть, в частности, реализованы на следующих типах компонентов:

- примитивах;
- макрофункциях;
- мегафункциях;
- параметризированных модулях;
- компонентах MegaWizard;
- компонентах пользователя;
- компонентах MegaCore;
- компонентах AMPP (Altera Megafunction Partner Program).

Отметим особенности создания проектов.

Примитивы Quartus специфицированы в пакете maxplus2 библиотеки altera. Триггер dff в этом пакете описан как:

```
component dff
    port ( D     : in STD_LOGIC;
           CLK   : in STD_LOGIC;
           CLRN: in STD_LOGIC;
           PRN   : in STD_LOGIC;
           Q     : out STD_LOGIC);
end component;
```

Принципиальная схема сдвигающего регистра на примитивах и временные диаграммы его работы приведены на рисунках 8.1 и 8.2 соответственно.

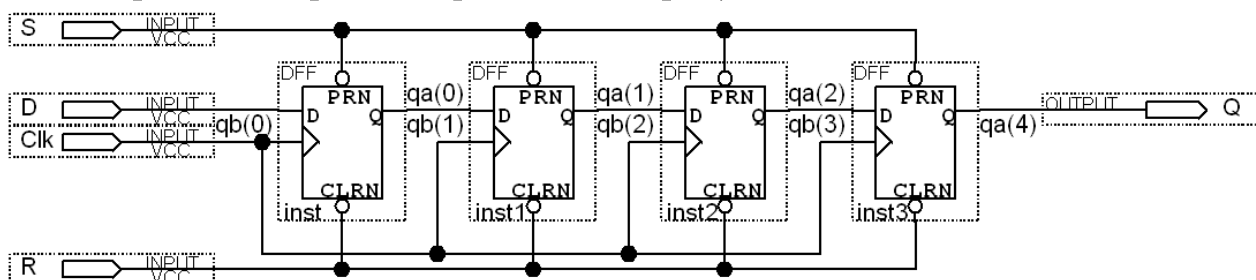

Рисунок 8.1 – Принципиальная схема сдвигающего регистра

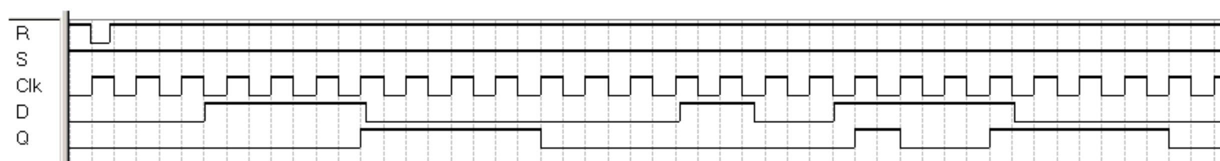

Рисунок 8.2 – Временные диаграммы работы сдвигающего регистра

При описании нерегулярных структур необходимо декларировать все компоненты, ввести внутренние сигналы (для примера на рисунке 8.1 введены сигналы q(0) …q(1)) и указать все линии связи между компонентами и интерфейсом.

```
LIBRARY altera;
USE altera.maxplus2.all;
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY vhdl_shift IS
        PORT(s, r, clk, d  : IN    STD_LOGIC;
                 q             : OUT        STD_LOGIC);
END vhdl_shift;
ARCHITECTURE arch OF vhdl_shift IS
        SIGNAL qa          : STD_LOGIC_VECTOR(0 TO 2);
                BEGIN
                        dff0: dff      PORT MAP (d, clk, r, s, qa(0));
                        dff1: dff      PORT MAP (qa(0), clk, r, s, qa(1));
                        dff2: dff      PORT MAP (qa(1), clk, r, s, qa(2));
                        dff3: dff      PORT MAP (qa(2), clk, r, s, q);
END arch;
```

Пакет std_logic_1164 указан для определения типа данных std_logic, с которыми работает триггер dff.

Сигналы подключены к портам компонента с использованием соответствия позиций, что позволило упростить описание проекта.

Описания проекта на основе других компонентов аналогичны за исключением отличий в указаниях о месте расположения описания требуемого компонента.

Структурное описание устройств с применением Named Port Association и Positional Port Association на языке VHDL позволяет описывать произвольные структуры. Однако такое описание весьма трудоемко и менее наглядно в сравнении со схемным представлением проекта. Преимущества языка VHDL проявляются при описании регулярных структур. Для этих целей применяют предложение генерирования Generate Statement (For Generate, If Generate) и предложение цикла Loop Statement (For Statement, While Statement).

8.3 Описание регулярных структур

Предложение Generate Statement упрощает описание регулярных структур цифровых устройств. Обычно оно используется, чтобы определить группу идентичных компонентов, используя только одно определение компонента и повторяя его, используя механизм Generate. Упрощенный синтаксис этого предложения имеет вид:

```
__generate_label:
FOR __index_variable IN __range GENERATE
   __statement;
   __statement;
END GENERATE;
```

```
__generate_label:
IF__expression GENERATE
   __statement;
   __statement;
END GENERATE;
```

For Generate используется для описания регулярных структур, в которых компоненты и их связи идентичны.

Однако часто регулярные структуры содержат некоторые отличия в компонентах и связях между ними. В таких случаях применяется конструкция If Generate. Рассмотрим пример описания четырехразрядного сдвигающего регистра, отличия в котором состоят в подключении первого и последнего триггеров (смотри рисунок 8.1).

Можно создать несколько вариантов описания регистра, которые отличаются друг от друга сложностью описания. Рассмотрим два из них.

В первом случае применены предложения For Generate и If Generate из-за неидентичности подключения первого и последнего триггеров. Для соединения триггеров между собой декларированы сигналы qa(0)… qa(2).

```
LIBRARY altera;
USE altera.maxplus2.all;
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY vhdl_shift1 IS
        PORT(s, r, clk, d  : IN    STD_LOGIC;
                q          : OUT STD_LOGIC);
END vhdl_shift1;

ARCHITECTURE arch OF vhdl_shift1 IS
    SIGNAL qa              : STD_LOGIC_vector(0 TO 2);
        BEGIN
                gen1:
                FOR I IN 0 TO 3 GENERATE
                        gen2:
                                IF I = 0 GENERATE
                                  dff1: dff PORT MAP (d, clk, r, s, qa(0));
                                END GENERATE;
                        gen3:
                                IF I > 0 AND I < 3 GENERATE
                                  dff1: dff PORT MAP (qa(I-1), clk, r, s, qa(I));
                                END GENERATE;
                        gen4:
                                IF I =3 GENERATE
                                  dff1: dff PORT MAP (qa(I-1), clk, r, s, q);
                                END GENERATE;
                END GENERATE;
        END arch;
```

Во втором случае с целью упрощения описания регистра увеличена разрядность внутреннего сигнала qb до 5 разрядов. Описание имеет вид.

```
LIBRARY altera;
USE altera.maxplus2.all;
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY vhdl_shift2 IS
        GENERIC  (wdth          : INTEGER :=3);
        PORT     (s, r, clk, d  : IN    STD_LOGIC;
                  q             : OUT         STD_LOGIC);
END vhdl_shift2;

ARCHITECTURE arch OF vhdl_shift2 IS
    SIGNAL qb : STD_LOGIC_vector(0 TO wdth+1);
        BEGIN
```

```
        qb(0) <= d; q <= qb(wdth+1);
            gen1:
              FOR I IN 0 TO wdth GENERATE
                 dff1: dff PORT MAP (qb(I), clk, r, s, qb(I+1));
              END GENERATE;
END arch;
```

Анализ описания регистра позволяет выделить основные специфические этапы описания регулярных структур после создания описаний входящих компонентов:

– указание параметра регулярности структуры width (Generic Clause);
– определение входящих компонентов (Component Declaration);
– задание внутренних сигналов (Signal Declaration);
– задание регулярности структуры (For Generate);
– указание отличий в регулярности структуры (If Generate);
– задание линий связи (Component Instantiation Statement).

Применение параметра регулярности структуры width позволяет просто изменять разрядность проектируемых устройств.

9 ФУНКЦИОНАЛЬНОЕ ОПИСАНИЕ ЦИФРОВЫХ УСТРОЙСТВ

9.1 Описание комбинационных устройств

Комбинационными называют устройства, сигналы на выходах которых в любой момент времени однозначно определяются сочетанием сигналов на входах и не зависят от их предыдущих состояний. Примерами комбинационных устройств являются дешифраторы, мультиплексоры, сумматоры и т.д.

Комбинационные устройства в VHDL реализуют обычно с помощью Concurrent Signal Assignment Statements или Process Statements по формату:

```
ARCHITECTURE arch OF gate IS
BEGIN
      Concurrent Signal Assignments
      Process Statements
      Other Concurrent Statements
END arch;
```

Возможны следующие типы Concurrent Signal Assignment Statements:

− Simple Signal Assignment Statement;
− Conditional Signal Assignment Statement;
− Selected Signal Assignment Statement.

Simple Signal Assignments обычно описывают по шаблону:

```
__signal <= __expression;
```

Его применяют для описания простейших цифровых устройств, например, логических элементов.

```
ENTITY and4 IS
     PORT ( in1, in2, in3, in4       : IN   BIT;
               q                     : OUT  BIT);
END and4;
ARCHITECTURE and4_arch OF and4 IS
     BEGIN
          q <= in1 AND in2 AND in3 AND in4;
END and4_arch;
```

Приведенная программа описывает элемент 4И.

Conditional Signal Assignment Statement указывает список назначений целевому сигналу в результате анализа булевых выражений по шаблону:

```
__label:
__signal <=  __expression WHEN __boolean_expression ELSE
              __expression WHEN __boolean_expression ELSE
              __expression;
```

Selected Signal Assignment указывает возможные выражения для целевого сигнала в зависимости от значения сигнала выбора по шаблону:

```
__label:
WITH __expression SELECT
        __signal <=  __expression WHEN __constant_value,
                     __expression WHEN __constant_value,
                     __expression WHEN __constant_value,
                     __expression WHEN __constant_value;
```

9.1.1 Особенности применения сигналов

Основная цель поведенческой спецификации VHDL состоит в том, чтобы описать реакцию выходов на входы. И входы, и выходы – сигналы, так что реакции выходных сигналов даются как назначения сигнала. Однако, использование сигналов в процессах регулируется в соответствии с тремя важными ограничениями:

– сигналы не могут быть объявлены внутри процессов;

– любое присвоение сигналу выполняется только после приостановки процесса. До этого все сигналы хранят их предыдущие значения;

– только последнее присвоение сигналу внутри процесса эффективно.

Таким образом, в процессе не имеет смысла присваивать больше одного значение сигналу.

Ограничения на использование сигналов имеют серьезное значение, что приводит к следующим последствиям:

– сигналы хранят только последнее присвоенное значение, поэтому они не могут использоваться для хранения промежуточных или временных данных внутри процессов;

– новые значения присваиваются после приостановки процесса, что делает анализ проекта довольно громоздким.

Указанные ограничения определяют потребность в объекте, который объявлен внутри процесса и обеспечил бы временное хранение данных. Такой объект в VHDL существует и назван переменной.

9.1.2 Предложение Process Statement

Process Statement – это параллельное предложение языка, в котором на основе множества последовательных предложений описывается поведение всего устройства или его части. Способ описания аппаратуры посредством множества взаимодействующих между собой процессов является наиболее естественным.

Определение Process Statement для комбинационных устройств.

```
[ process_label :]
PROCESS [ ( sensitivity_list) ]
      process_declarative_part
BEGIN
      process_statement_part
END PROCESS [ process_label ] ;
```

Метка процесса (process_label) необязательна, однако если она есть в конце (после слов end process), то она должна быть и в начале перед словом process.

В списке чувствительности (sensitivity_list) указывают имена сигналов, при изменении значений которых данный процесс выполняется. В ином случае (список чувствительности у процесса отсутствует) последовательное выполнение предложений этого процесса возобновляется непосредственно вслед за завершением выполнения его последнего предложения. Использование списка чувствительности позволяет оптимизировать время счета, например, при описании синхронных схем. Наличие у процесса списка чувствительности эквивалентно наличию Wait Statement в конце описания этого процесса с идентичным списком чувствительности. При наличии списка чувствительности в этом процессе нельзя использовать предложение Wait (ни в нем самом, ни в процедурах, в нем вызывающихся). Все имена сигналов в списке чувствительности должны быть видимыми и статическими (вычисляться при наличии индексов через статические выражения). Очевидно, что для корректного моделирования необходимо, чтобы процесс без списка чувствительности содержал предложение Wait Statement, т.к. в противном случае моделирование зациклится на выполнении этого процесса.

В декларативной части процесса (process_declaration_part) могут быть: декларации подпрограмм, тела подпрограмм, декларации подтипов, типов, констант, переменных, файлов, альтернативных имен, конфигураций, атрибутов, спецификации атрибутов.

Сигналы не могут быть декларированы внутри процессов.

Структура процесса имеет вид:

```
__process_label:
PROCESS (__signal_name, __signal_name, __signal_name)
        VARIABLE __variable_name : STD_LOGIC;
        VARIABLE __variable_name : STD_LOGIC;
BEGIN
        -- Signal Assignment Statement
        -- Variable Assignment Statement
        -- Procedure Call Statement
        -- If Statement
        -- Case Statement
        -- Loop Statement
END PROCESS __process_label;
```

Рассмотрим основные предложения, входящие в состав процессов (Process Statement).

9.1.3 Предложение Variable Declaration Statement

Декларацию переменных осуществляют по шаблону:

```
VARIABLE __variable_name : __type_name;
```

Переменные допустимы только в процессах и подпрограммах.

9.1.4 Предложение Variable Assignment Statement

Переменные ограничены процессами, они объявляются в процессах или подпрограммах и не могут быть использованы вне их. Присвоение значения переменной выглядит подобно присвоению значения сигнала.

__variable_name := __expression;

Присваивание значения переменной осуществляется символом ' := '. Такое присваивание мгновенно и каждой переменной могут быть назначены новые значения столько раз, сколько это необходимо.

Укажем различия между локальными переменными и сигналами.

Локальные переменные декларируются и видны только внутри процесса или подпрограммы. Сигналы не могут быть декларированы внутри процесса или подпрограммы.

Новое значение локальной переменной немедленно корректируется, когда выполняется оператор присваивания. Понятие времени не ассоциируется с понятием переменной. Оператор назначения сигнала корректирует сначала драйвер сигнала. Когда процесс станет приостановленным, сигнал корректируется. Поэтому использование сигналов ведет к двухпроходному моделированию. С точки зрения системы моделирования переменные применять целесообразнее.

Только сигналы могут употребляться для связывания параллельных операторов. Порты, декларируемые в entity, являются сигналами. Аргументы подпрограмм могут быть сигналами или переменными.

В VHDL-описаниях логических (цифровых) схем сигнал употребляется для соединения элементов. Переменные обычно употребляются как временные значения в процессе или алгоритме описания функции.

9.1.5 Подпрограммы

Подпрограммы имеют две формы – процедуры и функции. Вызов процедуры есть предложение, в то время как вызов функции возвращает значение в выражении. Указателем функции (операторным символом) может быть идентификатор или строковый литерал. Строковый литерал не может быть именем процедуры.

Процедура – подпрограмма, которая определяет алгоритм для вычисления значений или функциональное поведение устройства.

Упрощенный синтаксис.

```
PROCEDURE procedure_name (formal_parameter_list) IS
        procedure_declarations
    BEGIN
        sequential statements
END PROCEDURE procedure_name;
```

В качестве формальных параметров можно использовать константы (Constant), переменные (Variable), сигналы (Signal) и файлы (File). Класс каждого параметра определяется соответствующим зарезервированным словом.

В случае констант, переменных и сигналов режим параметра определяет направление информационного потока, и таким образом решается какие формальные параметры могут считываться или записываться внутри процедуры. Параметры типа файл не имеют никакого назначенного режима.

Имеются три доступные режима: In, Out, и Inout. Когда объявлен режим In, а класс объекта не определен, тогда по умолчанию считается, что объект – константа. В случае режимов Inout и Out, заданный по умолчанию класс – переменная. При вызове процедуры формальные параметры заменяются фактическими параметрами. В случае формальных параметров типа сигнала, переменной и файла фактические параметры должны быть объекты того же самого класса.

Тело процедуры определяет алгоритм процедуры, составленный из последовательных инструкций. При вызове процедуры начинает выполняться последовательность инструкций, объявленных внутри тела процедуры.

Тело процедуры состоит из декларативной и описательной частей подпрограммы.

В декларативной части могут быть объявлены: декларации подпрограмм, тела подпрограмм, декларации типов, подтипов, констант, переменных, файлов, альтернативных имен, атрибутов, спецификации атрибутов.

Описательную часть подпрограммы помещают между зарезервированными словами начало (Begin) и конец (End). Ключевое слово процедура (Procedure) и имя процедуры могут следовать за зарезервированным словом конец (End).

Процедура может содержать любые последовательные предложения, включая предложение ожидания (Wait). Предложение Wait не может использоваться в процедурах, которые вызываются из процесса со списком чувствительности или из функции.

Вызов процедуры (предложение Procedure Call Statement) – это последовательная или параллельная инструкция в зависимости от того, где она используется. Последовательная процедура вызова выполняется всякий раз, когда управление достигает ее. Параллельная процедура вызова активизируется всякий раз, когда любой из ее параметров в режиме в In или Inout изменяет свои значения.

Все фактические параметры в процедуре вызова должны иметь тот же самый тип, что и формальные параметры, которые они заменяют.

Функция – подпрограмма, которая определяет алгоритм для вычисления значений или описывает поведение устройства. Важная особенность функции состоит в том, что она используется как выражение, которое возвращает значение указанного типа. В этом заключается основное отличие функций от другого типа подпрограмм – процедур.

Формат описания функции.

```
FUNCTION function_name (parameters) RETURN type IS
        declarations
    BEGIN
        sequential statements
END FUNCTION function_name;
```

Результат, возвращенный функцией, может иметь или скалярный или комплексный тип.

Имя функции, которое появляется после зарезервированного слова функция (Function), может быть или идентификатором или символом оператора (если функция определяется оператором). Спецификация новых функций для существующих операторов допустима в VHDL и названа перегрузкой оператора (Operator Overloading).

Функция имеет только входные параметры, которые могут быть константами, сигналами или файлами. Поскольку все параметры функции являются входами, то их режим (Mode) можно не указывать.

Рассмотрим в качестве примера использование функций и процедур для описания простейших логических элементов 4 исключающее или и 2-2и-или.

```
ENTITY call_proced IS
   PORT(a, b, c, d      : IN BIT;    --сигнал типа BIT
        q1, q2          : OUT BIT);
END call_proced;

ARCHITECTURE rtl OF call_proced IS
--функция, реализующая многовходовый элемент "исключающее или"
FUNCTION f_xor (a_f, b_f, c_f, d_f : BIT) RETURN BIT IS
   VARIABLE xor_v       : BIT;
BEGIN
   xor_v := a_f XOR b_f XOR c_f XOR d_f;
   RETURN xor_v;
END f_xor;
--процедура, реализующая логический элемент 2-2и-или
PROCEDURE pr_lr (a_pr, b_pr, c_pr, d_pr: IN BIT; q2_pr: OUT BIT) IS
BEGIN
   q2_pr := (a_pr AND b_pr) OR (c_pr AND d_pr);
END pr_lr;
BEGIN
--процесс вызова функции и процедуры
p0: PROCESS (a, b, c, d)
   VARIABLE v_q2: BIT;
BEGIN
   q1 <= f_xor (a, b, c, d);
   pr_lr (a, b, c, d, v_q2);
   q2 <= v_q2;
```

```
END PROCESS p0;
END rtl;
```

Подпрограммы делают проекты более компактными, легко модифицируемыми, читаемыми и верифицируемыми.

9.1.6 Предложение If Statement

Конструкция If Statement позволяет выбрать для выполнения последовательное предложение (группу последовательных предложений) на основе проверки заданных условий по шаблону:

```
IF__expression THEN __statement; __statement;
    ELSIF__expression THEN __statement; __statement;
    ELSE __statement; __statement;
END IF;
```

Условно можно выделить три формы If statement:

– If-Then;
– If-Then-Else;
– If-Then-Elsif-Else.

Предложение If Statement контролирует выполнение последовательных операторов. Оно содержит по крайней мере одно булево условие, указанное после ключевого слова If. Остающиеся условия определены после выражения Elsif. Выражение Else обрабатывается как Else If true Then. Условия оцениваются одно за другим, пока любое из них не становится истинным или не имеется больше условий, которые нужно проверить. Если условие истинно, то выполняется последовательность инструкций, указанных после выражения Then. Если условие не встречается, то управление переходит к следующему за предложением If Statement выражению.

Выражение Elsif используется, когда нужно проверить различные вложенные условия. Это может использоваться для расположения по приоритетам условий и сигналов.

9.1.7 Предложение Case Statement

Предложение Case Statement выбирает одну из альтернатив. Оно обычно осуществляется по шаблону:

```
CASE __expression IS
    WHEN __constant_value => __statement; __statement;
    WHEN __constant_value => __statement; __statement;
    WHEN OTHERS => __statement; __statement;
END CASE;
```

Избранная альтернатива (случай) определяется значением выражения. Выражение (expression) должно быть дискретного типа или типа одноразмерного массива символов, значения которых могут быть представлены как строки

или строка битов. Выбор (__constant_value) должен быть такого же типа как выражение. Все возможные значения expression должны присутствовать в __constant_value для предложения Case. Каждый выбор можно использовать только один раз. Если используется зарезервированное слово Others ("все случаи, кроме перечисленных"), то оно должна стоять последним и отдельно от других выборов.

Предложение Case применяют для описания конечных автоматов, дешифраторов, программ микропроцессора и т.д. Используя предложение Case, приведем пример описания дешифратора.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY decod IS
    PORT (bin   : IN    STD_LOGIC_VECTOR(3 DOWNTO 0);
          dec   : OUT   STD_LOGIC_VECTOR(9 DOWNTO 0));
END decod;

ARCHITECTURE decod_arch OF decod IS
BEGIN
    convert:
    PROCESS (bin)
    BEGIN
       CASE bin IS
          WHEN "0000" =>        dec <= "0000000001";
          WHEN "0001" =>        dec <= "0000000010";
          WHEN "0010" =>        dec <= "0000000100";
          WHEN "0011" =>        dec <= "0000001000";
          WHEN "0100" =>        dec <= "0000010000";
          WHEN OTHERS =>        dec <= "0000000000";
       END CASE;
    END PROCESS convert;
END decod_arch;
```

Использование зарезервированного слова Others позволяет заметно упростить описание неполных дешифраторов.

Предложение Case Statement рекомендуется использовать вместо предложения If Statement, если имеется такая возможность. Это позволяет упростить сложность конструктивной реализации проектируемого устройства.

9.1.8 Предложение Loop Statement

Предложение Loop Statement позволяет выполнить серию последовательных предложений ноль и более раз в зависимости от итерационной схемы.

```
loop_label:
      WHILE condition LOOP
            sequence_of_statements
```

```
        END LOOP loop_label;
loop_label:
        FOR loop_parameter IN range LOOP
                sequence_of_statements
        END LOOP loop_label;
```

Предложение Loop Statement может иметь несколько различных форм в зависимости от итеративной схемы, предшествующей зарезервированному слову Loop. Если итерационная схема опущена, то цикл будет выполняться бесконечное число раз до тех пор, пока не встретится Return Statement (выход из подпрограммы), или Exit Statement (выход из цикла). При этом в Exit Statement должно выполниться соответствующие условие, если оно присутствует.

В итерационной схеме condition – это выражение типа boolean и цикл выполняется до тех пор, пока оно истинно.

В итерационной схеме For количество итераций задается через диапазон. Параметр цикла можно использовать только внутри цикла и нельзя модифицировать.

9.2 Описание последовательных устройств

Последовательными называют устройства, сигналы на выходах которых в любой момент времени определяются не только сочетанием сигналов на входах, но и некоторыми внутренними сигналами (состояниями), учитывающими предыдущие входные воздействия. Примерами последовательных устройств являются регистры, счетчики, триггеры, цифровые автоматы и т.д.

Последовательные устройства в VHDL, в основном, создают с помощью предложения Process Statement, которое позволяет создавать устройства, контролируемые сигналом синхронизации. Последовательные устройства рекомендуется создавать по шаблону:

```
__process_label:
PROCESS
        VARIABLE __variable_name : STD_LOGIC;
        VARIABLE __variable_name : STD_LOGIC;
BEGIN
        WAIT UNTIL __clk_signal = '1';
        -- Signal Assignment Statement
        -- Variable Assignment Statement
        -- Procedure Call Statement
        -- If Statement
        -- Case Statement
        -- Loop Statement
END PROCESS __process_label;
```

Основное отличие приведенного шаблона от шаблона для описания комбинационных устройств состоит в применении предложения Wait.

Предложение Wait Statement приостанавливает выполнение процесса. Конструкция Wait Statement имеет три формы:

– Wait On sensitivity_list;
– Wait Until condition;
– Wait For time_expression.

Sensitivity_list описывает множество сигналов, на изменение значений которых отзывается предложение Wait.

Condition представляет собой булевское выражение, определяющее условие, при котором ожидание заканчивается и выполнение процесса возобновляется.

Time_expression указывает максимальное время, на которое приостанавливается выполнение процесса.

Конструкции sensitivity_list, condition, time_expression можно использовать в любом необходимом для моделирования сочетании.

Последовательное выполнение предложений в процессе приостанавливается на предложении Wait и возобновляется с выполнения следующего за Wait предложения. Возобновление процесса происходит в момент времени изменения какого-либо сигнала из sensitivity_list, если condition равно true или по истечении времени ожидания, заданного в time_expression. Имена сигналов в sensitivity_list должны быть статическими и видимыми. При использовании в sensitivity_list составных типов сигналов (массивов и др.) все элементы этих сигналов будут принадлежать sensitivity_list.

Можно использовать предложение Wait внутри процедуры, вызываемой из процесса, но нельзя использовать внутри функции, или внутри процедуры, вызываемой из функции. Нельзя использовать предложение Wait при наличии у процесса списка чувствительности (ни в самом процессе, ни в процедуре, вызываемой из процесса).

Обычно в САПР описания цифровых устройств поддерживается только конструкция

```
WAIT UNTIL __clk_name = '1';
```

причем она должна быть первой в предложении Process Statement.

9.2.1 Описание триггеров

Триггеры обычно создают с помощью различных форм предложения If Statements, которое помещают внутри предложения Process Statement. Следующий пример показывает описание D-триггера с применением первой формы предложения If Statement.

```
ENTITY latch IS
    PORT (enable, data    : IN BIT;
              q            : OUT BIT) ;
END latch;

ARCHITECTURE maxpid OF latch IS
```

```
BEGIN
    latch  : PROCESS (enable, data)
    BEGIN
        IF(enable = '1') THEN
            q <= data;
        END IF;
    END PROCESS latch;
END maxpid;
```

Анализ временных диаграмм работы D-триггера (смотри рисунок 9.1) позволяет выявить особенности его работы:

– после инициализации на выходе триггера устанавливается неопределенное состояние;

– триггер переходит в определенное состояние по приходу единичного уровня сигнала синхронизации;

– в триггере реализовано статическое управление.

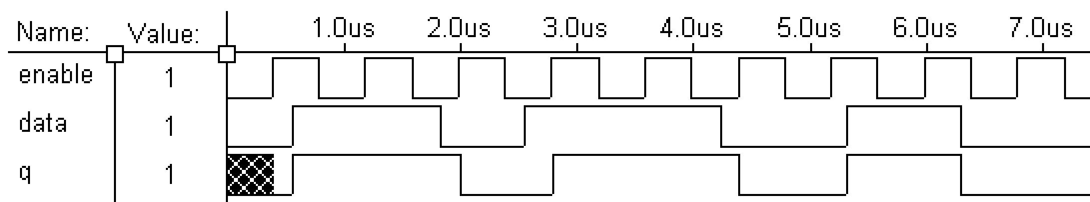

Рисунок 9.1 – Временные диаграммы работы D-триггера

9.2.2 Описание регистров

Для описания работы регистров могут быть применены практически все предложения VHDL, однако наиболее простое описание получают при использовании предложений Wait Statement, If Statement, Loop Statement и оператора конкатенации.

Предложение Wait Statement позволяет довольно просто создавать, в частности, многоразрядные регистры.

```
ENTITY reg_unt IS
    PORT (d      : IN BIT_VECTOR(0 TO 7);
              clk   : IN BIT;
              q;     : OUT BIT_VECTOR(0 TO 7))
END reg_unt;

ARCHITECTURE maxpid OF reg_unt IS
BEGIN
    PROCESS
    BEGIN
        WAIT UNTIL clk = '1';
        q <= d;
    END PROCESS;
```

END maxpid;

Анализ временных диаграмм описанного устройства (смотри рисунок 9.2) показывает, что оно представляет собой восьмиразрядный регистр с динамическим управлением.

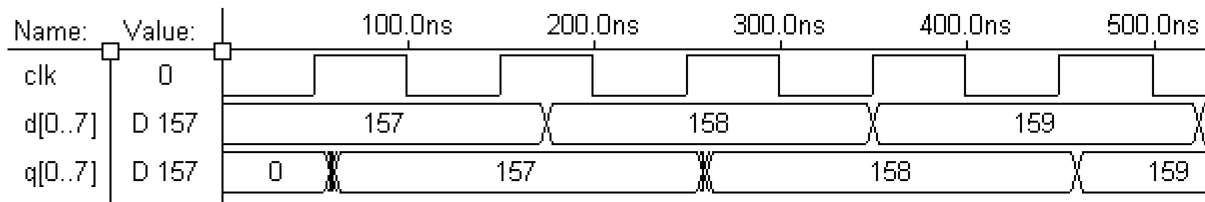

Рисунок 9.2 – Временные диаграммы работы регистра

9.2.3 Описание счетчиков

Счетчик представляет собой последовательное устройство для подсчета количества импульсов, поступивших на его вход в определенный промежуток времени. Счетчик может быть создан с помощью Register Inference, который приведен, в частности, в [8] или в подразделе Implementing Counters помощи к САПР, а также посредством функционального описания. Счетчики обычно создают с помощью предложения If Statement, в котором указывают активный фронт сигнала синхронизации, а также дополнительную логику для управления режимами работы счетчика. Предложение If Statement и дополнительная логика должны быть расположены внутри предложения Process Statement.

Рассмотрим описание синхронного реверсивного счетчика с установкой в ноль.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY counter IS
        PORT (clk, clear, up_down       : IN STD_LOGIC;
                q                       : OUT INTEGER RANGE 0 TO 255);
END counter;
ARCHITECTURE a OF counter IS
        BEGIN
        PROCESS (clk)
                VARIABLE        cnt             : INTEGER RANGE 0 TO 255;
                VARIABLE        direction       : INTEGER;
                        BEGIN
                                IF(up_down = '1') THEN          direction       := 1;
                                        ELSE                    direction       := -1;
                                END IF;
                                IF(clk'EVENT AND clk = '1') THEN
                                        IF clear = '0' THEN             cnt     := 0;
                                                ELSE            cnt:= cnt + direction;
                                        END IF;
```

```
                    END IF;
                q <= cnt;
            END PROCESS;
        END a;
```

Временные диаграммы работы счетчика приведены на рисунке 9.3, откуда следуют особенности работы счетчика:

– счетчик работает в режимах вычитания и сложения;

– изменение состояний счетчика, а также его установка в ноль происходят по переднему фронту сигнала синхронизации.

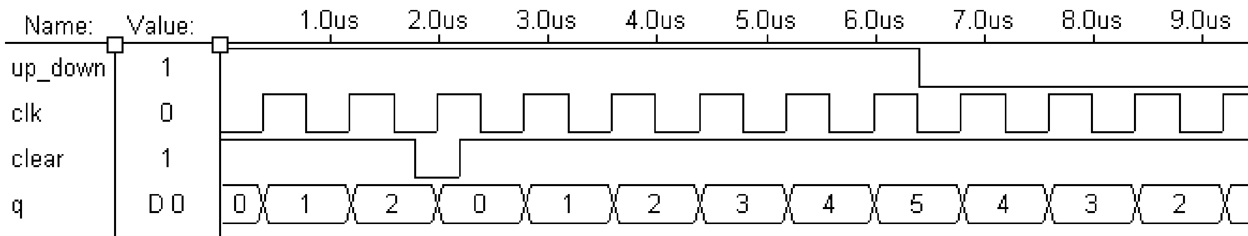

Рисунок 9.3 – Временные диаграммы работы счетчика

9.2.4 Описание цифровых автоматов

Цифровой автомат представляет собой последовательное устройство, которое переходит из одного состояния в другое под воздействием управляющих импульсов. Для создания цифрового автомата необходимо декларировать его состояния посредством Enumeration Type, а затем применить предложение Process Statement с целью определения его состояний и логики перехода между ними.

Рассмотрим описание цифрового автомата, который выделяет из входного сигнала последовательность 110. Работу цифрового автомата обычно описывают с помощью диаграммы состояний, пример которой приведен на рисунке 9.4.

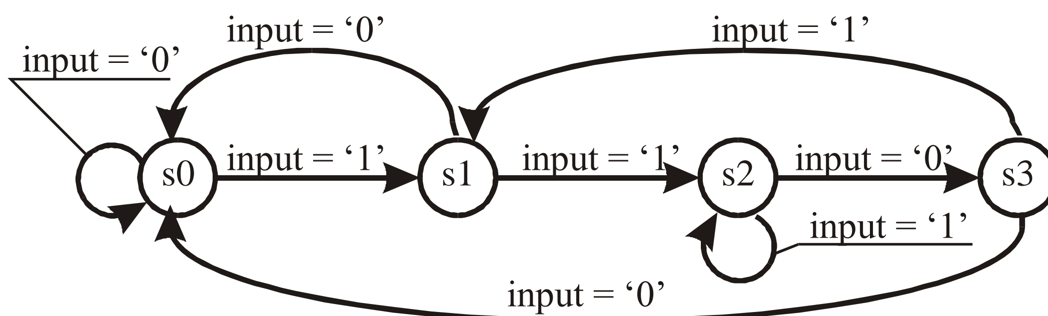

Рисунок 9.4 – Диаграмма состояний цифрового автомата

Описание цифрового автомата по диаграмме состояний, приведенной на рисунке 9.4, приведено ниже.

```
ENTITY state110 IS
   PORT (clk, input, reset        : IN Bit;
               output                 : OUT Bit);
END state110;

ARCHITECTURE archstatemachr OF state110 IS
    TYPE State_type IS (s0, s1, s2, s3);       -- состояния автомата
    SIGNAL state : State_type;                 -- служебный сигнал
      BEGIN
         PROCESS (clk)
           BEGIN
              IF reset = '1' THEN   state <= s0;
                  ELSIF (clk'EVENT AND clk = '1') THEN
                CASE state IS
                     WHEN s0 =>
                        IF input = '1' THEN
                             state <= s1;
                        ELSE
                             state <= s0;
                        END IF;
                      WHEN s1 =>
                        IF input = '1' THEN
                             state <= s2;
                        ELSE
                             state <= s0;
                        END IF;
                      WHEN s2 =>
                         IF input = '1' THEN
                             state <= s2;
                         ELSE
                             state <= s3;
                         END IF;
                      WHEN s3 =>
                         IF input = '1' THEN
                             state <= s1;
                         ELSE
                             state <= s0;
                         END IF;
                   END CASE;
           END IF;
         END PROCESS;

                  output <= '1' WHEN state = s3 ELSE '0';
```

END archstatemachr;

Если необходим автомат без установки в ноль, то из входных сигналов убирают сигнал reset, а следующие строки

IF reset = '1' **THEN** state <= s0;
ELSIF (clk'**EVENT AND** clk = '1') **THEN**
преобразуют к виду
IF (clk'**EVENT AND** clk = '1') **THEN**

Описание цифрового автомата содержит Process Statement, который активизируется по переднему фронту каждого импульса сигнала синхронизации (смотри рисунок 9.5).

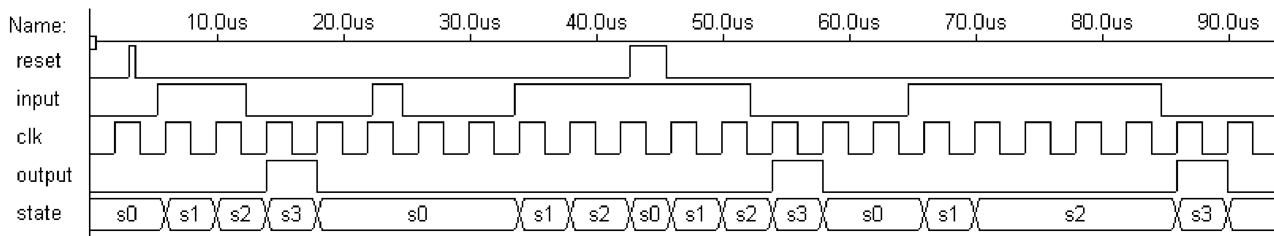

Рисунок 9.5 – Временные диаграммы работы цифрового автомата

Сигнал state определяет текущее состояние цифрового автомата. Декларация state_type определяет состояния s0, s1, s2 и s3.

После инициализации цифровой автомат переходит в состояние s0. В дальнейшем предложение Case Statement определяет условия перехода между состояниями по переднему фронту сигнала clk.

10 МОДЕЛИРОВАНИЕ ЦИФРОВЫХ УСТРОЙСТВ

10.1 Общие методические рекомендации по выполнению лабораторных работ

Изучение дисциплины “Технологии проектирования компьютерных систем” эффективно лишь тогда, когда наряду с овладением теорией студенты, в условиях проведения лабораторного практикума, самостоятельно создают цифровые устройства различного назначения, знакомятся с САПР, применяемыми в промышленности.

Предлагаемые методические указания посвящены изучению языка описания аппаратных средств VHDL и САПР Quartus II, которая отличается от других известных систем:

– простотой и удобством в обращении;

– развитой системой помощи, которая включает различные шаблоны для описания цифровых устройств на языках VHDL, AHDL, Verilog;

– обеспечением сквозного проектирования электронной аппаратуры;

– развитыми библиотеками компонентов;

– наличием студенческих версий;

– развитой и удобной для разработчика системой меню;

– интуитивно понятным графическим интерфейсом пользователя;

– простотой получения лицензии на использование САПР в учебных целях.

В ходе выполнения лабораторных работ студенты должны:

– изучить язык описания аппаратных средств VHDL;

– получить знания о методах практического применения ЯОА для проектирования цифровых устройств различного назначения;

– освоить методологию автоматизированного проектирования;

– усвоить основы теории цифровых устройств и методы их проектирования.

Подготовка к лабораторной работе предусматривает обязательное изучение студентами теоретического материала.

Перед выполнением лабораторных работ студенты должны:

– внимательно ознакомиться с краткими теоретическими сведениями, приведенными в лабораторных работах;

– изучить встроенные типовые компоненты САПР Quartus II;

– произвести синтез цифрового устройства или разработать алгоритм его работы в соответствии с установленным вариантом.

Лабораторную работу рекомендуется начинать с математического моделирования примера цифрового устройства, приведенного в каждой лабораторной работе с указанием параметров расчета. Затем следует приступить к иссле-

дованию цифрового устройства, синтезированного с учетом установленного варианта задания.

Отчет о выполнении каждой лабораторной работы обязательно должен содержать такие структурные элементы:

- название лабораторной работы;
- цель работы;
- краткие теоретические сведения;
- порядок выполнения работы;
- результаты исследования и анализа параметров и характеристик исследуемого электронного устройства;
- особенности функционирования САПР Quartus II, выявленные в ходе выполнения лабораторной работы;
- выводы.

10.2 Лабораторная работа №1. Структурное описание комбинационных устройств

Цель работы: научиться описывать комбинационные устройства на структурном уровне.

1 Порядок выполнения работы

1.1 Изучите методики описания комбинационных устройств на структурном уровне в языке VHDL.

1.2 Изучите специфические особенности проектирования комбинационных устройств на структурном уровне в САПР Quartus.

1.3 Синтезируйте схему устройства с учетом установленного варианта задания по таблице 10.1, при этом разрядность входных и/или выходных сигналов должна быть не менее четырех.

Следует обратить особое внимание на необходимость наличия регулярных структур в синтезируемом устройстве.

Таблица 10.1 – Варианты заданий

№ варианта	Устройство	
	Название	**На основе компонентов**
1	Шифратор	Примитивов
2	Дешифратор	Макрофункций
3	Преобразователь двоично-десятичного кода в семисегментный	Примитивов
4	Мультиплексор	Мегафункций
5	Демультиплексор	Примитивов
6	Элемент контроля четности	Макрофункций
7	Вычитатель	Конструктора MegaWizard
8	Умножитель	Конструктора MegaWizard
9	Компаратор	Мегафункций
10	Преобразователь двоичного кода в код Грея	Макрофункций

На рисунке 10.1 в качестве примера приведены результаты синтеза мультиплексора на макрофункциях.

1.4 Создайте в Text Editor САПР Quartus структурное описание устройства с использованием и без использования предложения Generate Statement.

Примеры описания мультиплексора без использования и с использованием предложения Generate Statement приведены ниже.

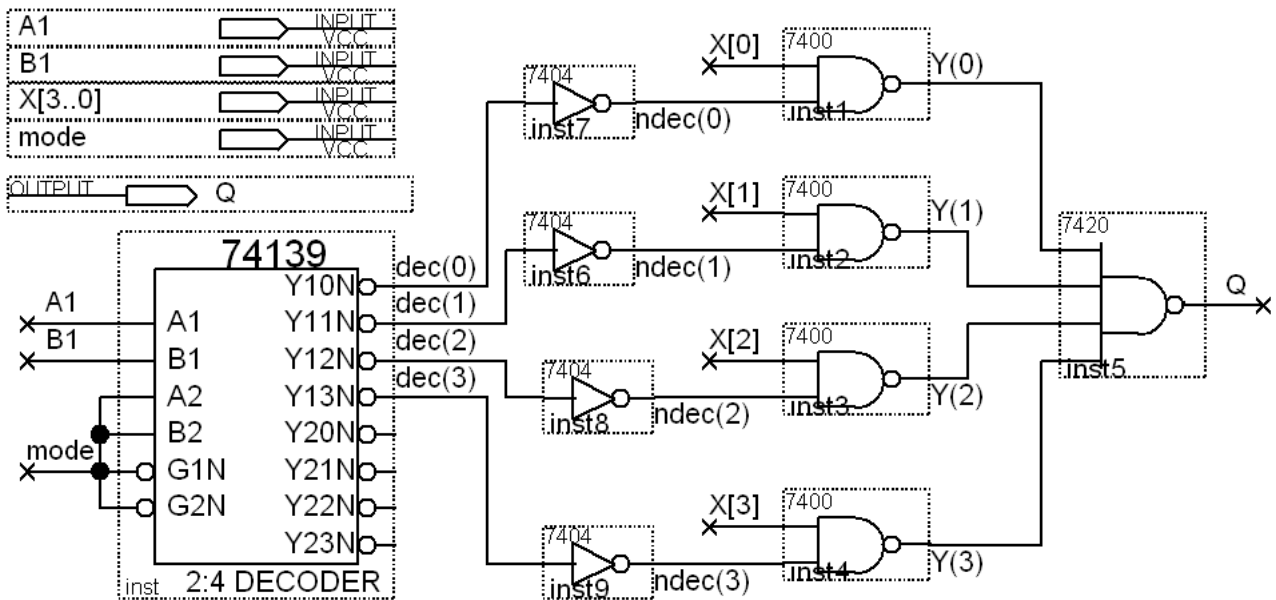

Рисунок 10.1 – Схема мультиплексора на макрофункциях

Описание мультиплексора без использования предложения Generate Statement.

```
LIBRARY altera;
USE altera.maxplus2.all;
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY vhdl_mux IS
   PORT(A1,B1, mode  : IN    STD_LOGIC;
             X              : IN    STD_LOGIC_VECTOR(0 TO 3);
             q              : OUT STD_LOGIC);
END vhdl_mux;
ARCHITECTURE a OF vhdl_mux IS
    SIGNAL dec, ndec, Y        : STD_LOGIC_ VECTOR (0 TO 3);
       BEGIN
        decod: a_74139 PORT MAP (mode,B1,A1,mode,mode, mode,dec(0),
                                  dec(1), dec(2),dec(3),open,open,open,open);
        inv1: a_7404  PORT MAP (dec(0), ndec(0));
        inv2: a_7404  PORT MAP (dec(1), ndec(1));
        inv3: a_7404  PORT MAP (dec(2), ndec(2));
        inv4: a_7404  PORT MAP (dec(3), ndec(3));
        vnand1: a_7400  PORT MAP (X(0), ndec(0), Y(0));
        vnand2: a_7400  PORT MAP (X(1), ndec(1), Y(1));
        vnand3: a_7400  PORT MAP (X(2), ndec(2), Y(2));
        vnand4: a_7400  PORT MAP (X(3), ndec(3), Y(3));
        vnand:  a_7420  PORT MAP (Y(0), Y(1), Y(2), Y(3),q);
END a;
```

Описание мультиплексора с использованием предложения Generate Statement.

```
IBRARY altera;
USE altera.maxplus2.all;
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY vhdl_mux1 IS
      PORT(A1,B1, mode      : IN   STD_LOGIC;
              X             : IN   STD_LOGIC_VECTOR(0 TO 3);
              q             : OUT       STD_LOGIC);
END vhdl_mux1;
ARCHITECTURE a OF vhdl_mux1 IS
  SIGNAL dec, ndec, Y : STD_LOGIC_VECTOR(0 TO 3);
      BEGIN
         decod: a_74139 PORT MAP (mode,B1,A1,mode,mode,mode,dec(0),
                          dec(1),dec(2),dec(3),open, open,open, open);
            gen1:
                  FOR I IN 0 TO 3 GENERATE
                        inv: a_7404 PORT MAP (dec(I),ndec(I));
                  END GENERATE;
            gen2:
                  FOR I IN 0 TO 3 GENERATE
                        vndn: a_7400 PORT MAP (X(I),ndec(I),Y(I));
                  END GENERATE;
         vnand: a_7420 PORT MAP (Y(0),Y(1),Y(2),Y(3),q);
END a;
```

1.5 Выполните в Compiler САПР Quartus компиляцию схемы.

1.6 Создайте в Waveform Editor файл входных воздействий для моделирования.

1.7 Выполните в Simulator моделирование созданного устройства и проанализируйте временные диаграммы его работы.

На рисунке 10.2 в качестве примера приведены временные диаграммы работы четырехразрядного сумматора

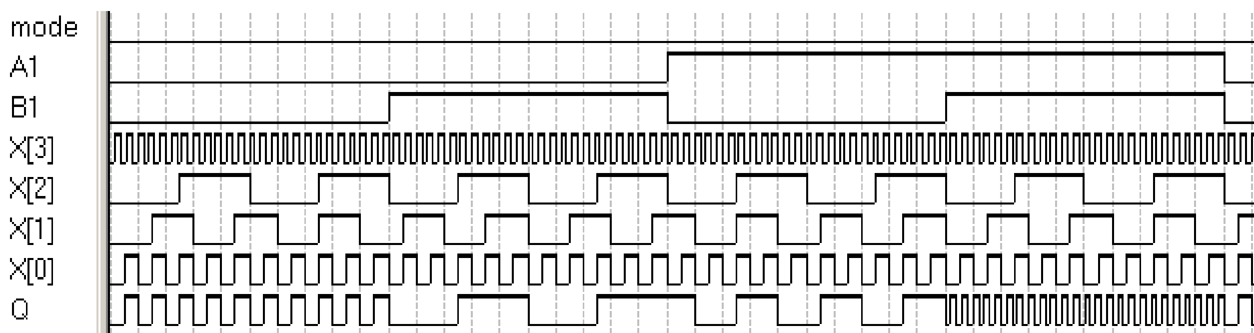

Рисунок 10.2 – Временные диаграммы работы мультиплексора

1.8 Выполните повторную компиляцию схемы при другом типе микросхемы.

1.9 Выполните моделирование при новом типе микросхемы.

1.10 Сравните временные диаграммы работы созданного устройства при различных типах микросхем.

2 Контрольные вопросы

2.1 Укажите правильные и неправильные идентификаторы: Decoder_1, _Decoder_1, 2FFT, sig_N, NOT_Ack, NOT-Ack, Sig_#N, FFT.

2.2 В каких разделах VHDL-кода можно объявлять сигналы: Library, Package, Entity, Architecture, Process, Block, Procedure, Function, Statement.

2.3 Отметьте ограничения на использование сигналов в языке VHDL.

2.4 Для каждого из утверждений ответьте, является ли оно правильным или ложным:

– все элементы массива должны быть одного и того же типа;

– значение true типа boolean эквивалентно значению 1 типа bit;

– константы могут употребляться в выражениях.

2.5 Какие средства языка VHDL применяют для соединения компонентов?

2.6 К каким предложениям (последовательным или параллельным) относится предложение Generate Statement?

2.7 Правильно ли, что структурное описание состоит из компонент и сигналов?

10.3 Лабораторная работа №2. Структурное описание последовательных устройств

Цель работы: научиться описывать последовательные устройства на структурном уровне.

1 Порядок выполнения работы

1.1 Изучите методики синтеза последовательных устройств на структурном уровне в языке VHDL.

1.2 Изучите специфические особенности проектирования последовательных устройств на структурном уровне в САПР Quartus.

1.3 Синтезируйте схему устройства с учетом установленного варианта задания по таблице 10.2, при этом разрядность входных и/или выходных сигналов должна быть не менее четырех. Тип триггеров – произвольный, если иное не оговорено в задании.

Следует обратить особое внимание на необходимость наличия регулярных структур в синтезируемом устройстве.

Таблица 10.2 – Варианты заданий

№ варианта	Устройство
1	Вычитающий счетчик с параллельным переносом на D-триггерах
2	Асинхронный вычитающий счетчик со сквозным переносом
3	Реверсивный счетчик с параллельным переносом на JK-триггерах
4	Асинхронный суммирующий счетчик с последовательным переносом
5	Асинхронный суммирующий счетчик со сквозным переносом
6	Синхронный вычитающий счетчик со сквозным переносом
7	Синхронный суммирующий счетчик с параллельным переносом
8	Синхронный суммирующий счетчик со сквозным переносом
9	Асинхронный реверсивный счетчик со сквозным переносом
10	Параллельно-последовательный регистр
11	Преобразователь последовательного кода в параллельный
12	Преобразователь параллельного кода в последовательный

Для примера на рисунке 8.1 приведена синтезированная схема сдвигающего регистра на триггерах dff.

1.4 Создайте в Text Editor описание схемы устройства с использованием и без использования предложения Generate Statement.

Примеры описания сдвигающего регистра с использованием и без использования предложения Generate Statement приведены в подразделах 8.2 и 8.3 соответственно.

1.5 Выполните в Compiler САПР Quartus компиляцию схемы.

1.6 Создайте в Waveform Editor файл входных воздействий для моделирования.

1.7 Выполните в Simulator моделирование созданного устройства и проанализируйте временные диаграммы его работы.

На рисунке 8.2 в качестве примера приведены временные диаграммы работы сдвигающего регистра, описанного в подразделе 8.3.

1.8 Выполните повторную компиляцию схемы при другом типе микросхемы.

1.9 Выполните моделирование при новом типе микросхемы.

1.10 Сравните временные диаграммы работы созданного устройства при различных типах микросхем.

2 Контрольные вопросы

2.1 Правильно ли, что компоненты, декларируемые в архитектурном теле, должны специфицироваться полностью, т.е. вместе с их интерфейсом и выполняемыми функциями?

2.2 В какой части VHDL-кода можно употреблять предложения If, Case, For Loop. Выберите правильный ответ:

– Architecture;

– Entity;

– Package;

– Process.

2.3 Правильно ли, что все компоненты должны быть описаны на поведенческом уровне?

2.4 Какая часть VHDL-кода содержит параллельные предложения. Выберите правильный ответ:

– Entity;

– Process;

– Package;

– Architecture.

2.5 Как должны быть подобны компонент (Component) и соответствующий интерфейс (Entity)? Выберите правильный ответ:

– Entity и Component должны иметь одно и то же имя, но порты могут различаться;

– имена Entity и Component могут различаться, но имена портов должны быть одинаковы;

– Entity и Component должны иметь одинаковые имена и должны иметь одинаковые имена портов.

2.6 Являются ли порты сигналами?

2.7 Правильно ли утверждение: ”Режим порта специфицирует направление потока данных через порт”?

10.4 Лабораторная работа №3. Поведенческое описание комбинационных устройств

Цель работы: научиться описывать комбинационные устройства на поведенческом уровне.

1 Порядок выполнения работы

1.1 Изучите методики синтеза комбинационных устройств на поведенческом уровне в языке VHDL.

1.2 Изучите специфические особенности проектирования комбинационных устройств на поведенческом уровне в САПР Quartus.

1.3 Разработайте описание цифрового устройства с учетом установленного варианта задания по таблице 10.3, при этом разрядность входных и/или выходных сигналов должна быть не менее четырех.

Таблица 10.3 – Варианты заданий

№ варианта	Устройство
1	Шифратор
2	Дешифратор
3	Преобразователь двоичного кода в двоично-десятичный
4	Мультиплексор
5	Демультиплексор
6	Сумматор
7	Вычитатель
8	Вычитатель в обратном коде
9	Вычитатель в дополнительном коде
10	Умножитель
11	Компаратор
12	Элемент контроля четности
Примечание – Устройства необходимо реализовать с использованием подпрограмм и без них.	

1.4 Создайте в Text Editor описание схемы устройства.

Для примера рассмотрим описание преобразователя двоично-десятичного кода в семисегментный, который имеет вид.

Входные сигналы: bin3, bin2, bin1, bin0 – двоичный код.

Выходные сигналы: a, b, c, d, e, f, g – семисегментный код.

Алгоритм работы преобразователя приведен в таблице 10.4.

Таблица 10.4 – Алгоритм работы преобразователя

Входной код				Выходной код						
bin3	bin2	bin1	bin0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
Примечание – При значениях кода, не указанных в таблице, по всем выходным сигналам должны быть установлены нулевые уровни.										

Пример описания преобразователя двоичного кода в семисегментный на языке VHDL приведен ниже.

```
ENTITY Binseg IS
PORT ( bin                  : IN     BIT_VECTOR(3 DOWNTO 0);
        a, b, c, d, e, f, g  : OUT    BIT);
END binseg;
ARCHITECTURE binseg_arch OF binseg IS
    SIGNAL seg :      BIT_VECTOR(6 DOWNTO 0);
BEGIN
    PROCESS (bin)
    BEGIN
        CASE bin IS
            WHEN "0000" => seg <= "1111110";
            WHEN "0001" => seg <= "0110000";
            WHEN "0010" => seg <= "1101101";
            WHEN "0011" => seg <= "1111001";
            WHEN "0100" => seg <= "0110011";
            WHEN "0101" => seg <= "1011011";
            WHEN "0110" => seg <= "1011111";
            WHEN "0111" => seg <= "1110000";
            WHEN "1000" => seg <= "1111111";
            WHEN "1001" => seg <= "1111011";
            WHEN OTHERS => seg <= "0000000";
        END CASE;
    END PROCESS convert;
    a<=seg(6); b<=seg(5); c<=seg(4); d<=seg(3);
    e<=seg(2); f<=seg(1); g<=seg(0);
END binseg_arch;
```

Пример использования функций и процедур для описания цифровых устройств приведен в разделе 9 настоящего документа.

1.5 Выполните в Compiler САПР Quartus компиляцию схемы.

1.6 Создайте в Waveform Editor файл входных воздействий для моделирования.

1.7 Выполните в Simulator моделирование созданного устройства и проанализируйте временные диаграммы его работы.

На рисунке 10.3 в качестве примера приведены временные диаграммы работы преобразователя двоично-десятичного кода в семисегментный.

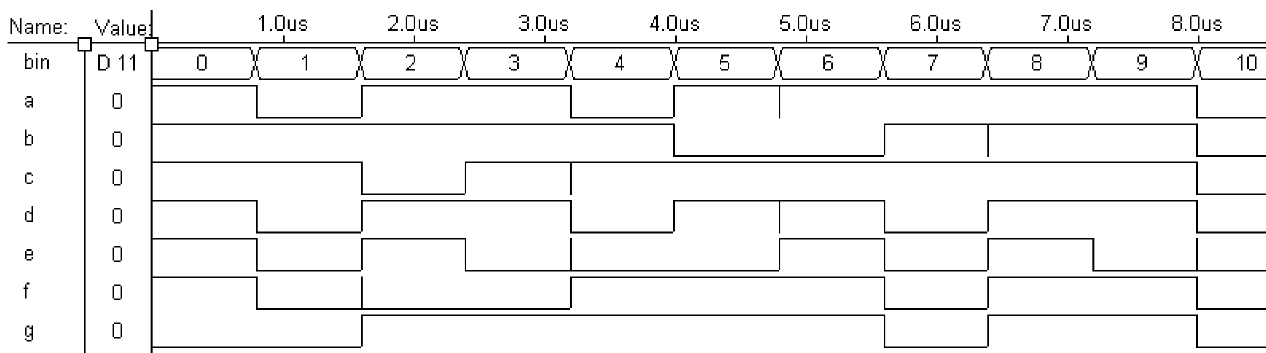

Рисунок 10.3 – Временные диаграммы работы преобразователя двоично-десятичного кода в семисегментный

1.8 Выполните повторную компиляцию схемы при другом типе микросхемы.

1.9 Выполните моделирование при новом типе микросхемы.

1.10 Сравните временные диаграммы работы созданного устройства при различных типах микросхем.

2 Контрольные вопросы

2.1 В какой части VHDL-кода необходимо указывать тип сигнала, который декларируется? Выберите правильный ответ:

– когда сигнал декларируется;

– когда сигнал употребляется первый раз в коде;

– в пакете;

– нет необходимости декларировать тип сигнала.

2.2 Какая часть VHDL-кода содержит последовательные операторы. Выберите правильный ответ:

– процесс (перед ключевым словом Begin);

– архитектурное тело;

– процесс (после ключевого слова Begin);

– пакет.

2.3 Путь имеется фрагмент VHDL-кода.

```
SIGNAL a_bus : BIT_VECTOR(3 DOWNTO 0);
SIGNAL z_bus : BIT_VECTOR(3 DOWNTO 0);
SIGNAL a_bit, b_bit, c_bit, d_bit : BIT;
```

```
byte <= (OTHERS => '1');
z_bus <= a_bit & b_bit;
a_bus <= ('1', b_bit, '0', d_bit);
a_bus(0 TO 1) <= (OTHERS => '0');
```

Какие строки корректны. Выберите правильный ответ. Обоснуйте его.

– только 2 и 4;

– только 3 и 4;

– только 1 и 2;

– только 1 и 3.

2.4 В чем состоит отличие между параллельными и последовательными предложениями?

2.5 Где в VHDL-коде может быть декларирована локальная переменная?

2.6 Правильно ли утверждение: “Каждый порт должен быть специфицирован с его режимом (Mode)”?

10.5 Лабораторная работа №4. Поведенческое описание последовательных устройств

Цель работы: научиться описывать последовательные устройства на поведенческом уровне.

1 Порядок выполнения работы

1.1 Изучите методики синтеза последовательных устройств на поведенческом уровне в языке VHDL.

1.2 Изучите специфические особенности проектирования последовательных устройств на поведенческом уровне в САПР Quartus.

1.3 Разработайте описание цифрового устройства с учетом установленного варианта задания по таблице 10.5, при этом разрядность входных и/или выходных сигналов должна быть не менее четырех.

Таблица 10.5 – Варианты заданий

№ варианта	Устройство
1	Асинхронный суммирующий счетчик
2	Асинхронный вычитающий счетчик
3	Асинхронный реверсивный счетчик
4	Синхронный суммирующий счетчик
5	Синхронный вычитающий счетчик
6	Синхронный реверсивный счетчик
7	Параллельный регистр
8	Последовательный регистр
9	Параллельно-последовательный регистр
10	Последовательно-параллельный регистр
11	Преобразователь последовательного кода в параллельный
12	Преобразователь параллельного кода в последовательный
Примечание – Устройства необходимо реализовать с использованием подпрограмм и без них.	

1.4 Разработайте схему алгоритма работы цифрового устройства.

Для примера на рисунке 10.4 приведена схема алгоритма работы синхронного суммирующего счетчика с параллельной загрузкой данных и сигналом разрешения.

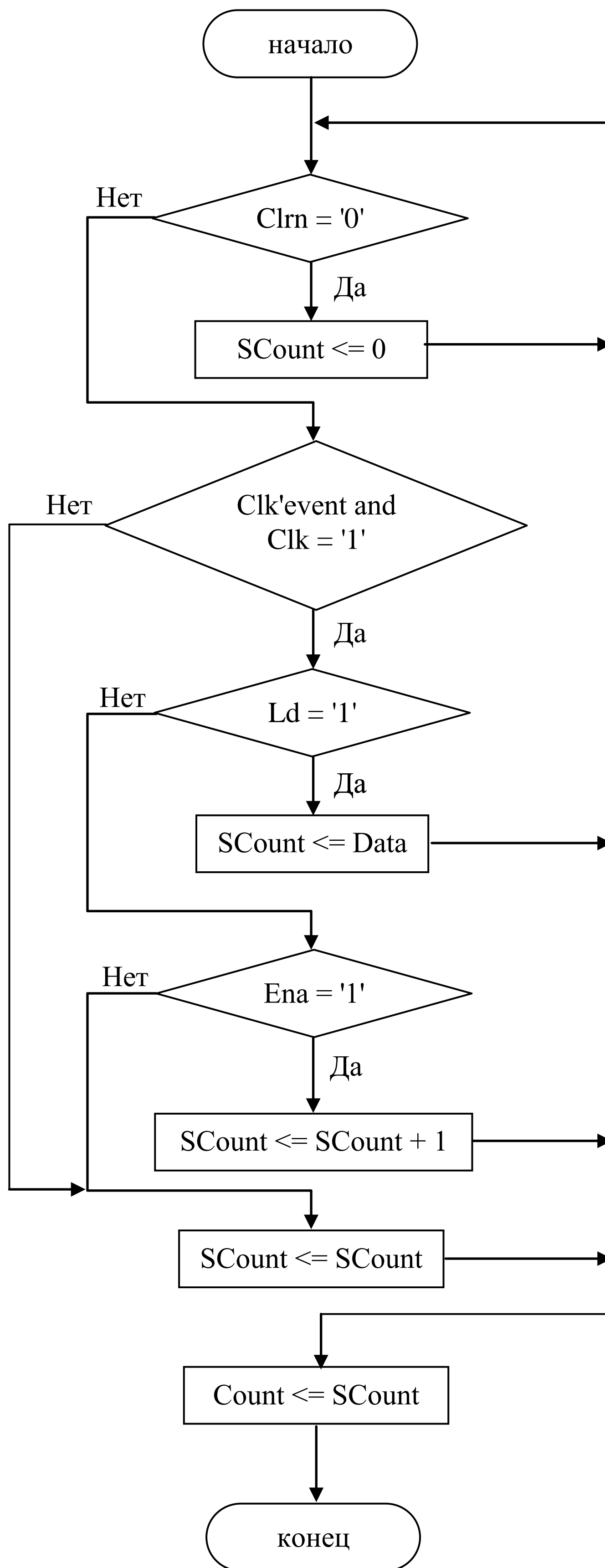

Рисунок 10.4 – Схема алгоритма работы синхронного суммирующего счетчика

1.5 Создайте в Text Editor описание цифрового устройства.

Пример описания синхронного суммирующего счетчика приведен ниже.

```
ENTITY uncounter IS
GENERIC (WIDTH : INTEGER := 16);              --коэффициент пересчета
PORT (data : IN  INTEGER RANGE 0 TO WIDTH-1;    --вход данных
      clk    : IN  BIT;                        --вход синхронизации
      clrn   : IN  BIT;                        --асинхронная установка в 0
      ena    : IN  BIT;                        --вход разрешения счета
      ld  : IN     BIT;                        --разрешение загрузки данных
      count : OUT INTEGER RANGE 0 TO WIDTH-1);     --счетный выход
END uncounter;

ARCHITECTURE archuncounter OF uncounter IS
SIGNAL scount : INTEGER RANGE 0 TO WIDTH-1;     --служебный сигнал
BEGIN
    PROCESS (clk, clrn, ena, ld)
      BEGIN
        IF clrn = '0' THEN scount <= 0;
         ELSIF (clk'EVENT AND clk = '1')        --по переднему фронту
         THEN
           IF ld = '1' THEN                     --установка начального зна-
              scount <= data;                   чения счетного выхода
           ELSE
             IF ena = '1' THEN                  --если разрешен счет
                 scount <= scount + 1;          --инкрементировать
             ELSE
                 scount <= scount;              --не изменять
             END IF;
           END IF;
        END IF;
    END PROCESS;
    count <= scount;
END ArchUnCounter;
```

1.6 Выполните в Compiler САПР Quartus компиляцию схемы.

1.7 Создайте в Waveform Editor файл входных воздействий для моделирования.

1.8 Выполните в Simulator моделирование созданного устройства и проанализируйте временные диаграммы его работы.

На рисунке 10.5 приведены временные диаграммы работы синхронного счетчика.

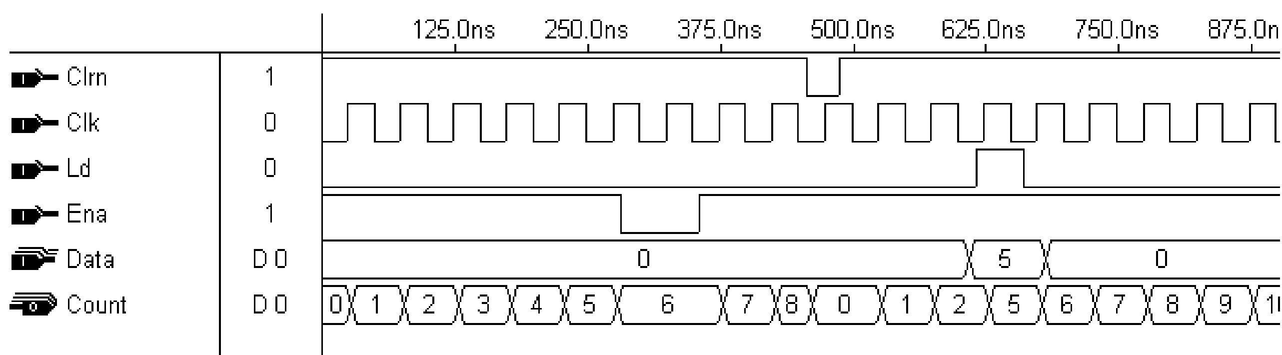

Рисунок 10.5 – Временные диаграммы работы синхронного счетчика

1.9 Выполните повторную компиляцию схемы при другом типе микросхемы.

1.10 Выполните моделирование при новом типе микросхемы.

1.11 Сравните временные диаграммы работы созданного устройства при различных типах микросхем.

2 Контрольные вопросы

2.1 Как много архитектурных тел может быть связано с одним Entity? Выберите правильный ответ:

– одно или более;

– более одного;

– только одно;

– ни одного.

2.2 Правильны ли утверждения:

– структурное VHDL-описание может быть иерархичным;

– поведенческое VHDL-описание может быть иерархичным;

– смешанное (структурно-поведенческое) VHDL-описание не может быть иерархичным.

2.3 Перечислите различия между локальными переменными и сигналами в языке VHDL.

2.4 Может ли процесс иметь список чувствительности и предложение Wait внутри предложения Process?

2.5 Может ли процесс иметь:

– несколько списков чувствительности;

– несколько предложений Wait внутри предложения Process;

– несколько списков сигналов запуска и несколько предложений Wait внутри предложения Process.

2.6 Правильно ли, что все предложения внутри процесса выполняются один за другим?

2.7 Должен ли каждый процесс иметь имя (метку)?

2.8 Правильно ли, что имя процесса специфицируется после ключевого слова Process?

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.