Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра электронных вычислительных машин

С. А. Байрак, М. М. Татур

АВТОМАТИЗАЦИЯ ПРОЕКТИРОВАНИЯ ЭВМ

Лабораторный практикум
для студентов специальности 1-40 02 01
«Вычислительные машины, системы и сети»
всех форм обучения

Минск 2008

УДК 681.31 (075.8)
ББК 32.973 я73
Б12

Рецензент
вед. науч. сотр. ОИПИ НАН Беларуси,
канд. техн. наук А. А. Дудкин

Байрак, С. А.

Б12 Автоматизация проектирования ЭВМ : лаб. практикум для студ. спец. 1-40 02 01 «Вычислительные машины, системы и сети» всех форм обуч. / С. А. Байрак, М. М. Татур. – Минск : БГУИР, 2008. – 44 с. : ил.
ISBN 978-985-488-319-9

Практикум содержит описание четырех лабораторных работ, тематика которых соответствует учебной программе. Приводятся краткие теоретические сведения по каждой из них и варианты заданий.

УДК 681.31 (075.8)
ББК 32.973 я73

ISBN 978-985-488-319-9

СОДЕРЖАНИЕ

Лабораторная работа №1

1. ПРОЕКТИРОВАНИЕ КОМБИНАЦИОННЫХ УСТРОЙСТВ НА ЯЗЫКЕ VHDL В СРЕДЕ WEBPACK

Цели :

1. Ознакомиться с САПР WebPACK и приобрести навыки работы с ним.
2. Изучить базовый синтаксис языка VHDL.
3. Приобрести навыки проектирования простейших комбинационных схем на языке VHDL.

1.1. Краткое описание САПР WebPACK

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

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

– редактор схемного ввода;
– текстовый редактор с поддержкой языков описания аппаратуры VHDL и Verilog;
– CORE Generator – генератор оптимизированных IP-ядер;
– редактор тестовых воздействий для программы моделирования;
– программа функционального и временного моделирования;
– генератор VHDL/Verilog кода;
– программа автоматического размещения и трассировки ПЛИС;
– программы «ручного» размещения и оптимизации проекта;

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

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

1.1.1. Процесс разработки цифровых устройств в среде WebPACK

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

1. Ввод описания проектируемого устройства в схемотехнической форме или с использованием языков описания аппаратуры (HDL), таких, как VHDL и Verilog.
2. Синтез устройства, то есть преобразование описания устройства, полученного на первом этапе, в описание на уровне логических вентилей.
3. Реализация устройства, то есть преобразование описания устройства на уровне логических вентилей в физическое описание для конкретной микросхемы ПЛИС.
4. Формирование конфигурационной последовательности для микросхемы ПЛИС.

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

На рис. 1.1 представлена обобщенная схема проектирования цифровых устройств в САПР WebPACK.

1.1.2. ProjectNavigator

Как уже отмечалось выше, САПР WebPACK состоит из большого количества разнообразных модулей, каждый из которых выполняет свои определенные функции. Для удобства их использования в WebPACK входит программа-оболочка ***ProjectNavigator***, представляющая собой графический интерфейс пользователя верхнего уровня, интегрирующий в себя всю работу с остальными

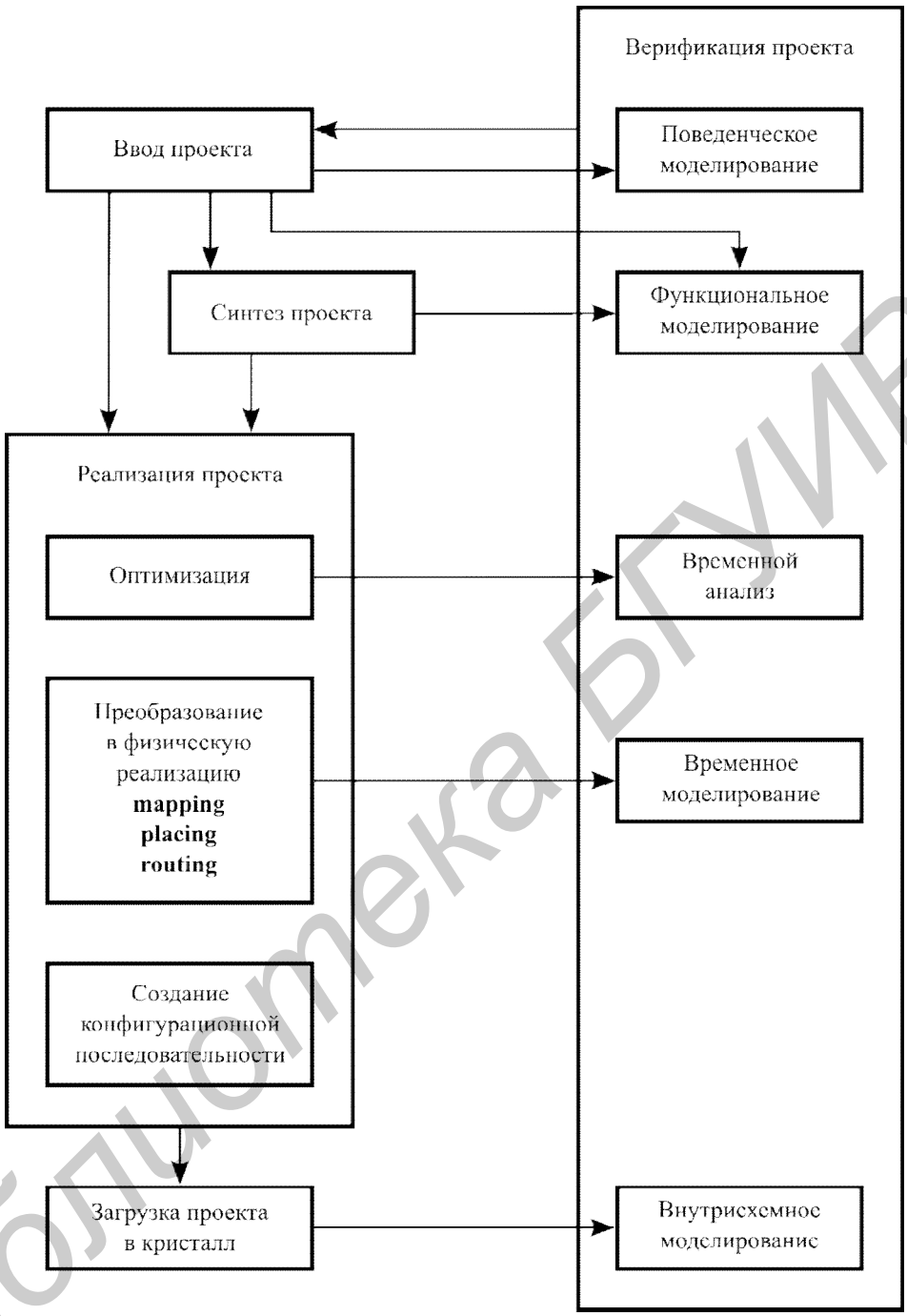

Рис. 1.1. Обобщенная схема проектирования цифровых устройств в САПР WebPACK

модулями САПР. ***ProjectNavigator*** позволяет:

– организовать работу по разработке цифровых устройств в виде проекта;

– добавлять, удалять и редактировать исходные файлы проекта различных типов;

– запускать обработку исходных файлов проекта соответствующими модулями САПР;

– анализировать результаты обработки исходных файлов проекта по развернутым отчетам, предоставляемым в удобной для разработчика форме.

Основное рабочее поле ***ProjectNavigator*** состоит из следующих элементов:

– окно исходных модулей (файлов) проекта (***Sources***);

– окно возможных процедур (процессов) для выбранного исходного модуля проекта (***Processes***);

– окно консольных сообщений программных модулей (***Transcript***);

– основное рабочее окно – окно редактора текстовых HDL-описаний проекта и вывода результатов работы отдельных модулей САПР в развернутом виде.

В окне исходных модулей (файлов) проекта отображается иерархическая структура, состоящая из модулей (файлов). В каждом файле содержится описание проектируемого устройства в графической или текстовой форме, а также другая информация, необходимая для успешного завершения разработки устройства (модули тестовых воздействий, модули временных и топологических ограничений и т.д.). Каждый тип модуля имеет соответствующее графическое обозначение – иконку.

Окно возможных процедур (окно процессов) показывает маршрут обработки выбранного исходного модуля в процессе проектирования устройства. Таким образом, в данном окне подробно отображаются все этапы процесса разработки и программирования ПЛИС. Последовательность и содержание этапов определяется типом исходного модуля и семейством ПЛИС. ***ProjectNavigator*** автоматически показывает в окне процессов структуру процесса проектирования, соответствующую выбранному семейству ПЛИС, исключая тем самым возможные ошибки в последовательности действий разработчика. В этом же

окне указывается информация о дополнительных инструментах, которые могут быть использованы на каждом этапе.

Окно консольных сообщений предназначено для вывода информации программных модулей пакета, работающих в консольном режиме.

1.1.3. Проекты в среде WebPACK

Проектом в САПР WebPACK называется совокупность модулей (файлов), которые содержат информацию, необходимую для выполнения всех этапов процесса разработки цифрового устройства на базе ПЛИС Xilinx. Все модули проекта располагаются в одной папке, название которой совпадает с названием проекта.

Для создания нового проекта необходимо воспользоваться командой ***NewProject*** в меню ***File***, в результате выполнения которой откроется окно мастера создания нового проекта.

На *первом этапе* работы этого мастера необходимо ввести имя проекта, определить каталог, в котором он будет располагаться, а также указать тип исходного файла верхнего уровня. Для всех лабораторных работ тип исходного файла верхнего уровня должен быть ***HDL***.

На *втором этапе* задаются параметры микросхемы, на которой будет реализовано устройство проекта, а также глобальные параметры процесса проектирования. В табл. 1.1 приведен перечень параметров и значения, которые необходимо установить для успешного выполнения всех лабораторных работ.

На *третьем этапе* можно сразу создать и добавить в проект новые исходные файлы. Эту же операцию можно осуществить и после создания проекта.

На *четвертом этапе* предлагается добавить в проект уже существующие файлы, что также можно сделать и после создания проекта.

На *пятом этапе* выводится вся общая информация по создаваемому проекту. Если значения параметров не требуют корректировки, нужно завершить работу мастера, в результате чего будет создан новый проект.

Таблица 1.1

Наименование параметра	Описание параметра	Значение параметра
Product Category	Возможные применения разрабатываемого устройства. Параметр используется для фильтрации возможных значений следующих двух параметров	***All***
Family	Семейство микросхем для реализации проекта	***Spartan3***
Device	Микросхема для реализации проекта	***XC2S200***
Package	Тип корпуса микросхемы	***PQ208***
Speed	Класс скорости	***-5***
Top-Level Source Type	Тип исходного файла проекта верхнего уровня, который был выбран на предыдущем этапе. Здесь нельзя его изменить	***HDL***
Synthesis Tool	Программа синтеза проекта. В лабораторных работах используется синтезатор, входящий в поставку САПР WebPACK	***XST(VHDL/ Verilog)***
Simulator	Программа для моделирования работы устройств. В лабораторных работах используется программа фирмы Mentor Graphics – Modelsim	***Modelsim***
Preferred Language	Определяет основной язык описания проекта. Используется программами, генерирующими описания на HDL на различных стадиях проектирования устройства	***VHDL***
Enable Enhanced Design Summary	Включает или отключает отображение дополнительной информации в окне ***Design Summary***	***Вкл***
Enable Message Filtering	Включение данной опции позволяет устанавливать фильтры на отображение сообщений в окне ***Design Summary***	***Выкл***
Display Incremental Messages	Опция используется для показа количества новых сообщений последнего используемого модуля в САПР	***Выкл***

Теперь можно приступать к его наполнению, то есть добавлению исходных файлов, содержащих ту или иную информацию о проекте.

1.1.4. Исходные файлы проекта

Проект в САПР WebPACK состоит из набора исходных файлов, которые могут быть разного типа. В табл. 1.2 перечислены типы исходных файлов, которые потребуются при выполнении лабораторных работ.

Таблица 1.2

Наименование типа	Расширение файла	Описание типа исходного файла
VHDL Module	.vhd	Содержит VHDL-код описания разрабатываемого устройства
Test Bench WaveForm	.tbw	Содержит графическое представление тестового воздействия, используемого для моделирования работы устройства
VHDL Test Bench	.vhd	Представляет описание тестового модуля на языке VHDL для выполнения моделирования работы устройства
Implementation Constraints File (User Constraints File)	.ucf	Файл пользовательских временных и топологических ограничений

Для выполнения первой лабораторной работы понадобятся файлы следующих типов:

– VHDL Module;

– Test Bench WaveForm.

Для создания нового исходного файла проекта необходимо выполнить команду ***New Source*** в меню ***Project***. При этом откроется мастер создания нового файла. Здесь на первом этапе нужно будет указать имя создаваемого файла, место его расположения и необходимость автоматического добавления его в проект.

В зависимости от типа добавляемого файла на последующих этапах работы мастера может возникнуть необходимость указания дополнительной информации.

Для файлов типа ***VHDL Module*** на втором этапе можно указать имена проектных модулей ***entity*** и ***architecture***, а также определить интерфейс объекта ими описываемого, то есть порты ввода/вывода. Все это и без всяческих ограничений, накладываемых мастером, можно сделать и после создания файла, непосредственно редактируя файл встроенным редактором, поэтому данный этап можно пропустить. После завершения работы мастера созданный исходный файл проекта откроется в текстовом редакторе для редактирования.

Для файлов типа ***Test Bench WaveForm*** на втором этапе выбирается объект, для которого создается тестовое воздействие, а после создания файла запускается мастер установки параметров тестового воздействия. В этом мастере параметр ***Clock Information*** необходимо установить в значение ***Combinatorial (or internal clock)***, при этом значения остальных параметров можно оставить без изменений. По окончании работы мастера запустится графический редактор, в котором можно задать необходимое тестовое воздействие.

1.1.5. Работа с VHDL-файлами проекта

Редактирование VHDL-файлов выполняется встроенным в WebPACK текстовым редактором. Для проверки синтаксиса VHDL-кода необходимо в окне исходных модулей проекта выбрать соответствующий файл, после чего в окне процессов выполнить команду ***Check Syntax*** элемента списка ***Synthesize–XST***. Результат выполнения данной команды будет отображен в окне консольных сообщений, кроме этого, рядом с самой командой в окне процессов будет отображена соответствующая иконка.

1.1.6. Моделирование работы устройства

Для моделирования работы разрабатываемого устройства сначала создается тестовое воздействие, которое будет подаваться на входы устройства, а затем специализированная программа выполняет само моделирование. Тестовое воз-

действие можно создавать как с использованием встроенного графического редактора тестовых воздействий, так и на языках HDL.

Для моделирования работы устройства, описанного на языках HDL, в состав САПР WebPACK входит программа Modelsim XE Starter компании Mentor Graphics. При запуске процесса моделирования САПР WebPACK подготавливает исходные данные для программы, загружает ее в память, передает ей подготовленные данные для работы и дает команду начала процесса моделирования. После окончания процесса моделирования Modelsim отобразит его результаты в графической форме в отдельном окне.

Как можно видеть из рис. 1.1, моделирование работы устройства может выполняться на разных стадиях его проектирования. Различают функциональное (поведенческое) и временное моделирование.

При функциональном моделировании не учитываются временные параметры элементов разрабатываемых устройств. Поэтому данный тип моделирования используется в основном для проверки соответствия описания разрабатываемого устройства его функциональному назначению.

Временное моделирование в отличие от функционального учитывает задержки, возникающие в разрабатываемом устройстве в ходе его работы. При этом в зависимости от этапа, на котором выполняется временное моделирование, могут учитываться как задержки самих элементов, из которых состоит разрабатываемое устройство, так и задержки линий связи, соединяющих эти элементы.

В табл. 1.3 приведены все возможные варианты моделирования устройств на различных этапах его разработки.

Для всех этапов моделирования, кроме первого (***Behavioural Simulation)***, описание устройства, полученное после выполнения очередного этапа проектирования, преобразуется в описания на языке HDL для передачи системе моделирования. Эта операция выполняется либо автоматически при запуске соответствующего этапа моделирования, либо может быть выполнена непосредственно пользователем через команды в окне процессов.

Таблица 1.3

Этап моделирования	Тип моделирования	Описание
Behavioral Simulation	Функцио-нальное	Выполняется на самом раннем этапе разработки устройства. На вход системы моделирования подается исходное описание устройства без какого-либо преобразования САПР (в случае если исходное описание выполнено на языках HDL)
Post-Synthesis Simulation	Функцио-нальное	Выполняется после синтеза описания устройства. На данном этапе полученное описание устройства является еще архитектурно независимым, то есть не зависит от микросхемы, на которой будет реализовано устройство
Post-Translate Simulation	Функцио-нальное	Выполняется после операции ***Translate*** системой проектирования. На данном этапе полученное описание устройства уже является архитектурно зависимым
Post-Map Simulation	Временное	Выполняется после операции ***MAP***. На данном этапе системе моделирования передаются задержки элементов разрабатываемого устройства без учета задержек линий передачи данных
Post-Place & Route Simulation	Временное	Выполняется после размещения и разводки проекта на микросхеме. Здесь системе моделирования передаются задержки как элементов разрабатываемого устройства, так и задержки линий передачи данных

1.2. Язык описания аппаратуры VHDL

Язык VHDL является фактически международным стандартом в области автоматизации проектирования цифровых систем. Многие современные системы автоматизированного проектирования как заказных, так и программируемых логических интегральных схем используют его в качестве своего входного языка. VHDL предназначен, в первую очередь, для спецификации – точного

описания проектируемых систем и их моделирования на многих этапах проектирования. С помощью VHDL можно моделировать электронные схемы с учетом реальных временных задержек.

1.2.1. Объекты

Объекты языка VHDL могут относиться к одной из трех категорий:

– сигналы;

– переменные;

– константы.

Сигналы (объекты данного типа) используются для передачи информации между модулями проекта или для представления входных и выходных данных проектируемого устройства. Сигнал имеет свойство изменения во времени.

Переменная представляет собой вспомогательную информационную единицу, используемую для описания внутренних операций в программных блоках.

Константой является объект, не изменяющий свое значение при вычислениях. После объявления константы присваивание ей нового значения запрещено.

1.2.2. Типы данных

VHDL – строго типизированный язык. Каждый объект объявляется со своим типом и может присваивать значения только данного типа. Соблюдение правил присваивания объектов и соответствия их типов требует дополнительных усилий разработчика, но благодаря этой особенности описания на языке VHDL имеют высокую надежность и обеспечивают экономию времени при отладке.

Язык VHDL предопределяет некоторый базовый набор типов данных, который не требует явного объявления. Кроме того, пользователь может определять свои типы данных. Различают скалярные и агрегатные типы данных. Объект, отнесенный к скалярному типу, рассматривается как законченная единица

информации. Агрегат представляет упорядоченную совокупность скалярных единиц, объединенных одинаковым именем. Классификация типов данных языка VHDL приведена на рис. 1.2.

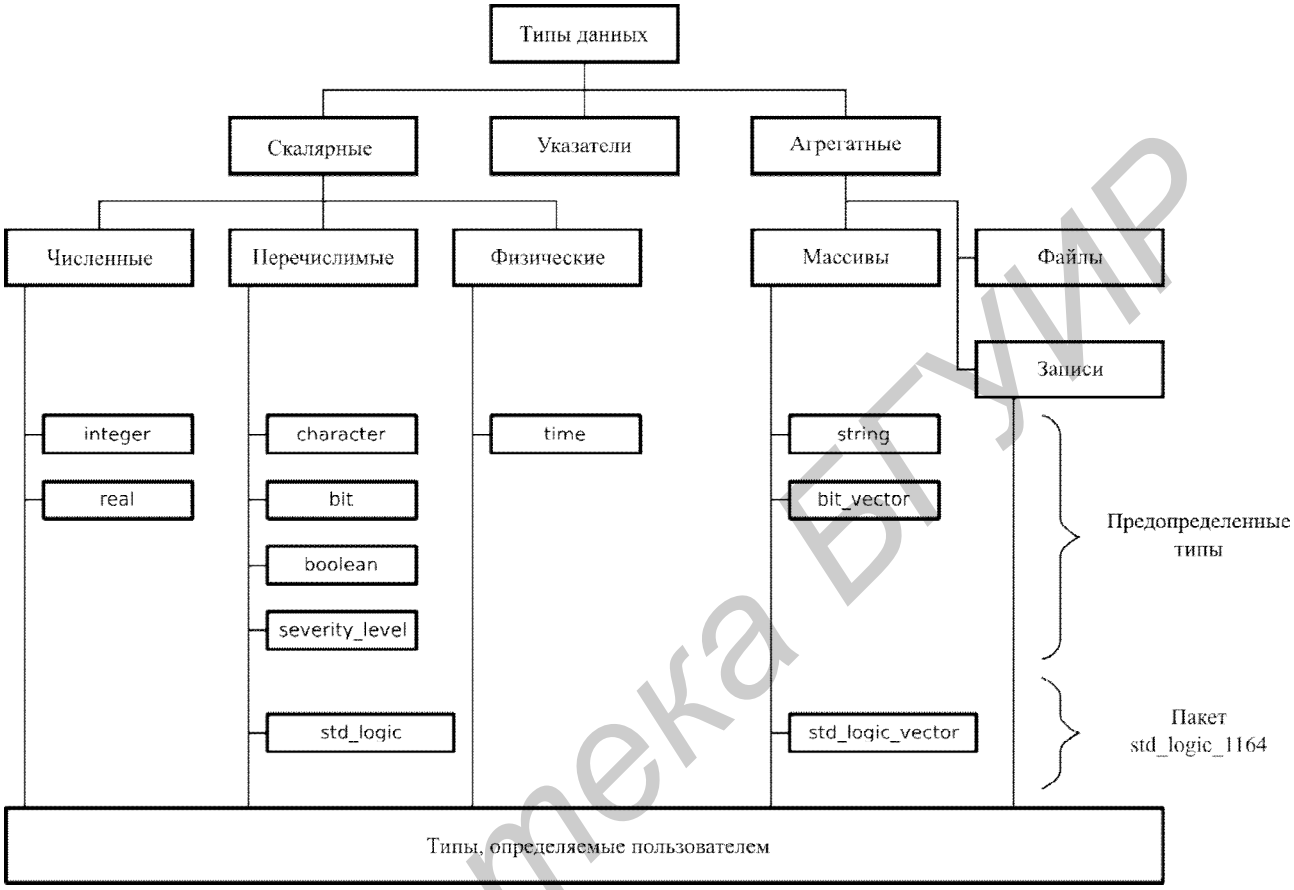

Рис. 1.2. Типы данных языка VHDL

1.2.3. Операторы

Все операторы языка VHDL делятся на последовательные и параллельные.

Последовательные операторы по характеру исполнения подобны операторам традиционных языков программирования. Операторы этого типа могут присутствовать только в параллельном операторе ***PROCESS*** или подпрограмме и выполняются последовательно друг за другом в порядке записи. В табл. 1.4 перечислены все последовательные операторы языка VHDL и дано их краткое описание.

Таблица 1.4

Оператор	Описание
if	Оператор условия
case	Оператор выбора
loop	Оператор цикла
<=	Последовательный оператор присваивания сигналу
:=	Оператор присваивания переменной
	Последовательный оператор вызова процедуры
wait	Оператор ожидания
assert	Последовательный оператор проверки
report	Оператор отображения сообщения
next	Оператор перехода к следующей итерации цикла
exit	Оператор завершения цикла
return	Оператор выхода из подпрограммы
null	Пустой оператор

Ниже приведены примеры использования некоторых последовательных операторов.

Оператор условия *if* предоставляет возможность выполнять различные наборы последовательных операторов в зависимости от каких-либо условий, возникающих в разрабатываемом устройстве в ходе его работы. Пример его использования:

```
not_logic:
 if x='0' then
     y <= '1';
 elsif x='1' then
     y <= '0';
 else
     y <= 'X';
 end if;
```

Оператор выбора *case*, подобно оператору условия, также позволяет выполнять разные наборы последовательных операторов по определенному условию. Пример использования оператора ***case***:

```
not_logic:
 case x is
      when '0' =>
           y <= '1';
      when '1' =>
           y <= '0';
      when others =>
           y <= 'X';
 end case;
```

Параллельные операторы – это такие операторы, каждый из которых выполняется при любом изменении сигналов, используемых в качестве его исходных данных. Результаты исполнения оператора доступны для других параллельных операторов не ранее, чем будут выполнены все операторы, инициализированные общим событием (а может быть и позже, если присутствует выражение задержки). В табл. 1.5 перечислены все параллельные операторы языка VHDL.

Таблица 1.5

Оператор	Описание
`block`	Оператор блока
`process`	Оператор процесса
`generate`	Оператор генерации
`<=`	Параллельный оператор присваивания сигналу
`assert`	Параллельный оператор проверки
	Параллельный оператор вызова процедуры
	Оператор создания экземпляра компонента

Оператор **`process`** является одним из самых важных параллельных операторов. Он предназначен для реализации одного независимого параллельного процесса, описываемого последовательными операторами. Последовательные операторы могут записываться только в теле оператора **`process`**. При моделировании фрагменты алгоритма, заключенные в оператор **`process`**, будут исполняться друг за другом после возникновения в системе «инициализирующего события» – изменения одного из сигналов, перечисленных в списке чувствительности. Если список чувствительности пуст либо отсутствует, процесс без-

условно выполняется при начальном запуске, а также после завершения выполнения последнего оператора. Параллельные операторы в теле процесса не определены. Переменные, как и последовательные операторы, могут использоваться только внутри оператора ***process***. Пример использования оператора ***process***:

```
logic : process(x0,x1,x2)
      variable y1,y2: std_logic;
begin
      y1 := x0 and x1;
      y2 := x1 or x2;
      z <= y1 and y2;
end process;
```

Параллельный оператор присваивания значений сигналу имеет несколько вариантов использования:

– параллельный оператор безусловного присваивания :

```
y <= not x;
```

– параллельный оператор условного присваивания :

```
y <= '1' when x = '0' else '0';
```

– параллельный оператор селективного присваивания :

```
with x select
     y <=      '1' when '0',
          '0' when others;
```

1.2.4. Структура проекта

Проект в системе проектирования на основе языка VHDL представлен совокупностью иерархически связанных текстовых фрагментов, называемых проектными модулями. Различают первичные и вторичные проектные модули.

Первичные проектные модули:

– entity (объект);

– package (пакет);

– configure (кофигурация).

Вторичные проектные модули:

– architecture (архитектура);

– package body (тело пакета).

Первичные и соответствующие им вторичные проектные модули могут сохраняться в различных файлах или записываться в один файл. Важно лишь, чтобы они были скомпилированы в общую проектную библиотеку, причем первичный модуль компилируется раньше подчиненного ему вторичного.

Любая цифровая система может быть представлена как совокупность блоков и связей между ними. Каждый из таких блоков в языке VHDL описывается с помощью проектного модуля ***entity***, который определяет:

– имя объекта (блока);

– порты ввода/вывода объекта (не обязательный);

– настроечные константы (не обязательный);

– дополнительную информацию.

Другими словами, проектный модуль ***entity*** определяет сам объект и его внешний интерфейс. Пример использования проектного модуля ***entity*** для объявления регистра:

```
entity register is
 generic (
      width : integer := 8;
 );
 port (
      input : std_logic_vector(width – 1 downto 0);
      output: std_logic_vector(widht – 1 downto 0)
 )
 constant setup : time := 10 ns;
begin

end register;
```

Проектный модуль ***architecture*** описывает функциональность объекта, определенного модулем ***entity***, и содержит операторы, определяющие его работу. При этом одному модулю ***entity*** может принадлежать несколько модулей ***architecture***. Пример использования проектного модуля ***architecture*** для описания логического элемента «И»:

```
architecture behavioral of and_logic is
      signal temp: std_logic := '0';
begin
      y <= x1 and x2;
end;
```

Модуль ***configuration*** предназначен для сопоставления объектов проекта, компонентов и архитектур и позволяет выполнять следующие задачи:

– специфицировать объект, соответствующий конкретному компоненту проекта;

– специфицировать архитектуру объекта, у которого их несколько.

Проектные модули ***package*** и ***package body*** предназначены для объединения по какому-либо критерию различных описаний языка VHDL в пакет для их последующего использования в различных проектах. При этом модуль ***package*** является декларативной частью пакета, а модуль ***package body*** представляет собой тело пакета, то есть содержит сами описания языка VHDL.

На рис. 1.3 представлено взаимоотношение проектных модулей, а общая схема описания устройства на языке VHDL показана на рис. 1.4.

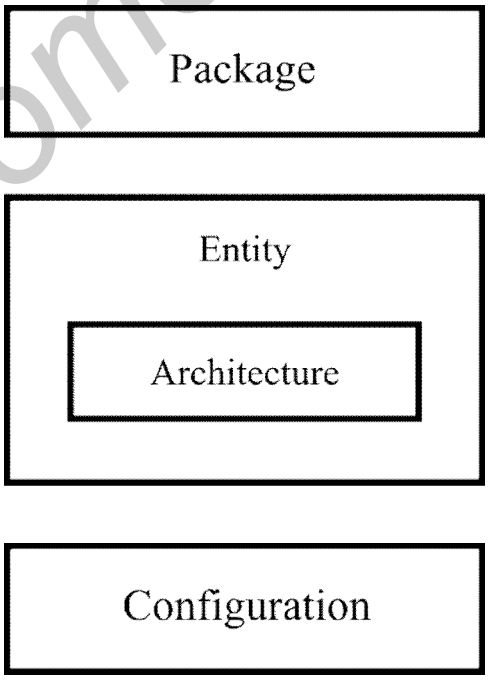

Рис. 1.3. Соотношение между различными модулями VHDL-проекта

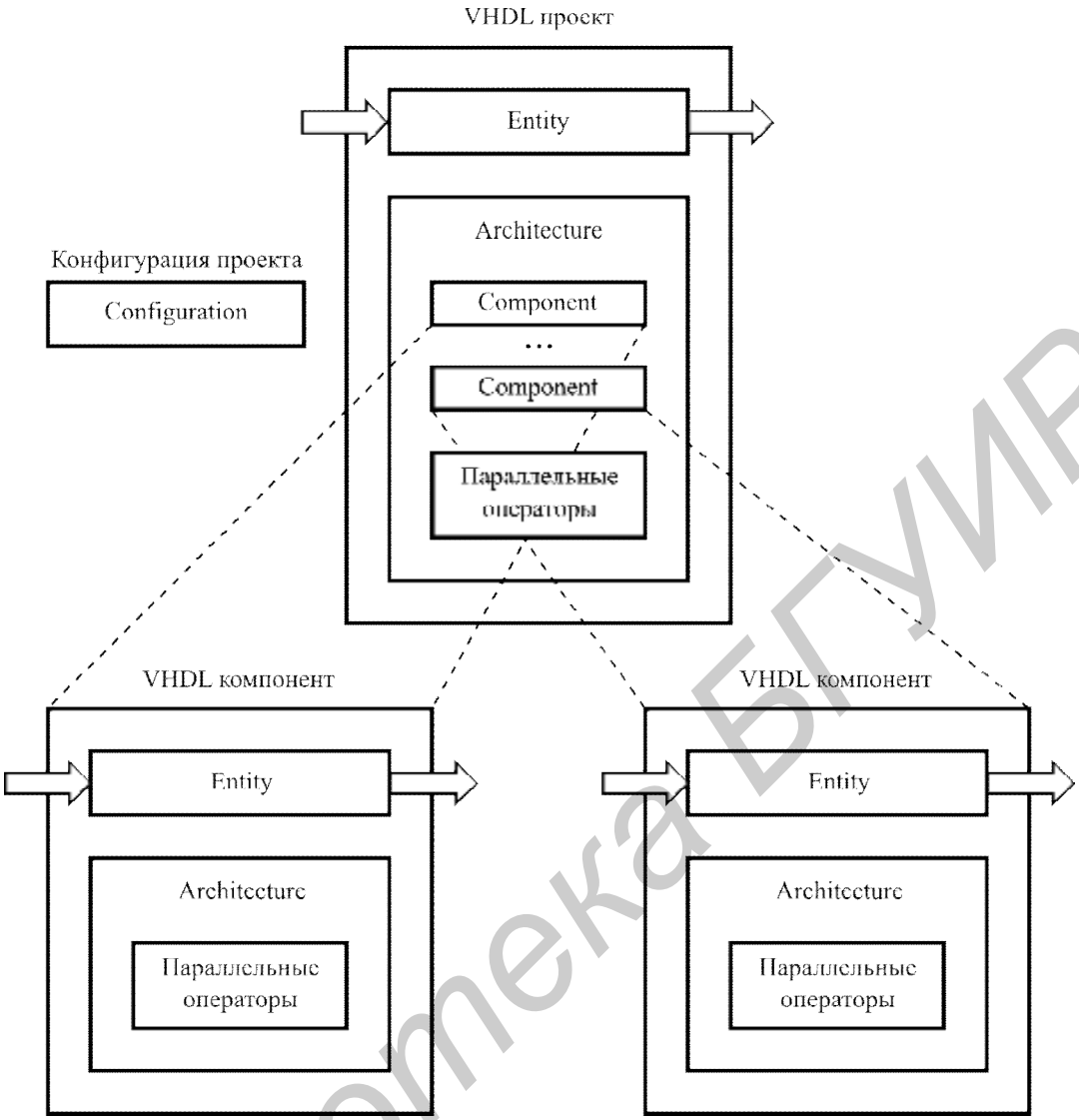

Рис. 1.4. Общая схема описания устройств на языке VHDL

1.3. Задание на лабораторную работу.

В соответствии с полученным вариантом задания необходимо спроектировать устройство на языке VHDL. При этом устройство должно быть реализовано пятью различными способами с использованием :

1) логических операторов и параллельного оператора безусловного присваивания;

2) параллельного оператора условного присваивания;

3) параллельного оператора селективного присваивания (присваивания по выбору) (***select***);

4) последовательного оператора условия (***if***);

5) последовательного оператора выбора (***case***).

Следует учитывать, что варианты 4 и 5 предполагают использование последовательных операторов, следовательно, для их реализации необходимо будет воспользоваться и параллельным оператором ***process***.

После описания заданного устройства всеми пятью способами необходимо создать тестовый модуль для моделирования его работы. При этом моделирование должно выполняться для всех способов реализации. Выполнить это можно двумя путями :

1) реализовать каждое описание устройства отдельным модулем проекта, и создать тестовый модуль для каждого описания в отдельности;

2) реализовать заданное устройство одним проектным модулем ***entity***, для которого необходимо создать пять проектных модулей ***architecture*** – для каждого способа описания в отдельности. После этого можно воспользоваться проектным модулем ***configure*** для выбора текущей архитектуры объекта. В этом случае необходимо создать только один тестовый модуль для тестирования устройства.

Для создания тестового модуля необходимо воспользоваться графическим редактором тестовых воздействий. При этом тестовые воздействия должны быть полными. Для заданных комбинационных устройств это означает полный перебор входных значений.

После создания тестового модуля необходимо выполнить поведенческое (***Behavioral Simulation***) моделирование работы устройства для всех способов его описания.

1.3.1. Варианты заданий

Варианты заданий представлены в таблице 1.6.

Таблица 1.6

Номер варианта	Устройство для реализации
1	Мультиплексор 8 → 1
2	Дешифратор 3 → 8
3	Приоритетный шифратор 8 →3
4	Простейшее 4-разрядное АЛУ ('+','-','+1')
5	Компаратор на 4 разряда

Лабораторная работа №2

2. ПРОЕКТИРОВАНИЕ ФУНКЦИОНАЛЬНЫХ УЗЛОВ ПОСЛЕДОВАТЕЛЬНОГО ТИПА НА ЯЗЫКЕ VHDL В СРЕДЕ WEBPACK

Цели :

1. Приобрести навыки проектирования функциональных узлов последовательного типа (автоматов с памятью) на языке VHDL.
2. Приобрести навыки структурного описания устройств на языке VHDL.

2.1. Атрибуты в языке VHDL

Атрибуты – это данные скалярного типа, отражающие некоторые свойства объектов, используемых в описаниях на языке VHDL. Атрибут имеет имя и тип. Атрибуты подразделяются на предопределенные и определенные пользователем. При выполнении лабораторных работ наиболее полезны предопределенные атрибуты, описание которых приведено ниже. В табл. 2.1 перечислены предопределенные атрибуты типов, в табл. 2.2 – атрибуты агрегатов (массивов), а в табл. 2.3 – атрибуты сигналов.

Таблица 2.1

Атрибут	Описание	Атрибутируемый тип
1	2	3
T'left	Левая граница значений ***T***	Скалярный
T'right	Правая граница значений ***T***	Скалярный
T'low	Нижняя граница значений ***T***	Численный, физический
T'high	Верхняя граница значений ***T***	Численный, физический
T'image(X)	Строка символов, представляющая значение ***X***	Любой
T'pos(X)	Позиция значения ***X*** в наборе значений ***T***	Перечислимый

Окончание табл. 2.1

1	2	3
T'val(N)	Значение элемента в позиции ***N*** в наборе значений ***T***	Перечислимый, физический, целый
T'leftof(X)	Значение в наборе значений ***T***, записанное в позиции слева от ***X***	Перечислимый, физический, целый
T'rightof(X)	Значение в наборе значений ***T***, записанное в позиции справа от ***X***	Перечислимый, физический, целый
T'pred(X)	Значение в наборе значений ***T*** на одну позицию меньшее ***X***	Перечислимый, физический, целый
T'succ(X)	Значение в наборе значений ***T*** на одну позицию большее ***X***	Перечислимый, физический, целый

Таблица 2.2

Атрибут	Описание
A'left(N)	Левая граница диапазона индексов ***N***-й координаты массива ***A***
A'right(N)	Правая граница диапазона индексов ***N***-й координаты массива ***A***
A'low(N)	Нижняя граница диапазона индексов ***N***-й координаты массива ***A***
A'high(N)	Верхняя граница диапазона индексов ***N***-й координаты массива ***A***
A'range(N)	Диапазон индексов ***N***-й координаты массива ***A***
A'reverse_range(N)	Обратный диапазон индексов ***N***-й координаты массива ***A***
A'length(N)	Диапазон индексов ***N***-й координаты массива ***A***

Таблица 2.3

Атрибут	Тип атрибута	Описание
1	2	3
S'delayed(T)	Тот же, что у ***S***	Значение ***S***, существовавшее на время ***T*** перед вычислением атрибута
S'event	***boolean***	Сигнализирует об изменении сигнала

Окончание табл. 2.3

1	2	3
S'stable	***boolean***	***S'stable*** = ***not S'event***
S'active	***boolean***	***true***, если присвоение сигналу выполнено, но значение еще не изменено (не закончен временной интервал, заданный выражением ***after***)
S'quiet	***boolean***	***S'active*** = ***not S'quiet***
S'last_event	***time***	Время от момента вычисления атрибута до последнего перед этим изменения сигнала
S'last_active	***time***	Время от момента вычисления атрибута до последнего присвоения значения сигналу (не совпадает с ***last_event*** при наличии слова ***after*** в определяющем выражении)

Атрибуты агрегатов удобно использовать для перебора их элементов в цикле. Например, побитовую операцию ***и*** двух массивов одинаковой размерности ***test_array_x*** и ***test_array_y*** можно описать следующим образом:

```
for i in test_array_x'range
      result_array := test_array_x(i) & test_array_y(i);
end for;
```

Здесь предполагается, что массивы ***result_array***, ***test_array_x*** и ***test_array_y*** являются переменными, а данная конструкция используется внутри оператора ***process***.

Атрибуты сигналов являются эффективным средством анализа поведения сигнала во времени и используются, например, для определения момента перехода сигнала из одного состояния в другое. Эта возможность, в свою очередь, часто используется при описании синхронных динамических функциональных узлов, то есть тех, которые изменяют свое состояние при переходах синхросигнала из 0 в 1 и наоборот. Ниже приведен пример использования атрибута сигнала для описания схемы, которая по фронту синхросигнала (***clk***), то есть при изменении его значения с 0 на 1, присваивает значение сигнала ***x*** сигналу ***y***.

```
process (clk)
begin
      if clk'event and clk = '1' then
           y <= x;
      end if;
end process;
```

2.2. Виды представления описаний проектов на языке VHDL

Классически принято выделять три различных стиля описания аппаратных архитектур на зыке VHDL :

– структурное описание;

– потоковое описание;

– поведенческое описание.

2.2.1. Структурное описание

При структурном описании (structural description) объекта проекта его архитектура представляется в виде иерархии связанных компонентов. Каждый экземпляр компонента представляет собой часть проекта, которая также может быть описана объектом проекта низшего уровня, состоящим из связанных компонентов. Таким образом может быть построена иерархия объектов, представляющих в конечном итоге весь проект.

Структурный тип описания отражает декомпозицию проекта на компоненты и делает акцент на соединениях, которые должны быть проведены между компонентами. При этом сами компоненты могут быть абстрактными функциональными устройствами или могут представлять отдельный вентиль, микросхему, плату или целую подсистему. Иерархия может представлять как структурное разбиение, так и функциональную декомпозицию разрабатываемого устройства.

2.2.2. Потоковое описание

При потоковом описании (data-flow description) объекта проекта его архитектура представляется в виде множества параллельных регистровых операций,

каждая из которых управляется вентильными сигналами. Потоковое описание соответствует стилю описания, используемому в языках регистровых передач.

В потоковом описании акцент делается на потоке информации между элементами с вентильным управлением. Этот поток информационного обмена регулируется и направляется управляющими элементами, которые логически отделены от путей прохождения данных. Так как пути данных показаны явно, то при потоковом описании не уделяется внимание структуре возможных реализаций.

2.2.3. Поведенческое описание

Поведенческий стиль описания определяет последовательно выполнимый код процедурного типа, подобный коду на языках программирования общего назначения. Основная причина использования поведенческого описания заключается в том, что данный стиль описания определяет с любой желаемой степенью точности функционирование устройства без определения его структуры.

Большая часть способов описания комбинационных устройств из первой лабораторной работы представляла собой именно поведенческое описание.

2.3. Использование компонентов в языке VHDL

Для использования объекта, описанного отдельным модулем ***entity***, необходимо выполнить следующие действия.

1. В декларативной части проектного модуля ***architecture*** объявить прототип используемого объекта, воспользовавшись оператором объявления компоненты.

2. В операторной части проектного модуля ***architecture*** создать экземпляр требуемого объекта, воспользовавшись параллельным оператором создания экземпляра компонента.

Ниже приведен пример использования регистра, описанного отдельным проектным модулем ***entity***.

```
architecture reg_use of reg_use is
 component register is
      generic (
           width : integer := 8
      )
      port (
           input : std_logic_vector(width - 1 downto 0);
           output: std_logic_vector(width – 1 downto 0)
      );
 end component;
 signal input_int : std_logic_vector(7 downto 0);
 signal output_int : std_logic_vector(7 downto 0);
begin
 register_inst : register
      generic map (
           width => 8
      )
      port map (
           input => input_int;
           output => output_int
      );
end reg_use;
```

2.4. Задание для лабораторной работы

Задание для лабораторной работы состоит из двух частей.

1. В соответствии с полученным вариантом задания разработать на языке VHDL синхронный динамический триггер. Для его моделирования разработать тестовое воздействие в соответствии с рис. 2.1. Тестовое воздействие можно реализовать с помощью встроенного в САПР WebPACK графического редактора. Выполнить поведенческое (***Behavioral Simulation***) моделирование работы заданного триггера.

2. В соответствии с полученным вариантом задания на базе триггера, разработанного в первой части лабораторной работы, реализовать на языке VHDL функциональный узел последовательного типа. Описание функционального узла должно быть выполнено в структурном виде, то есть отражать структуру, а не поведение данного устройства. Функциональные схемы устройств последовательного типа, отражающие их структуру, предоставляются преподавателем.

3. Для реализованного функционального узла последовательного типа

разработать по возможности полное тестовое воздействие и провести поведенческое (***Behavioral Simulation***) моделирование его работы. Данное тестовое воздействие должно проверять (моделировать) работу заданного устройства во всех режимах его работы.

На рис. 2.1 представлены условные графические обозначения (УГО) синхронных динамических триггеров и диаграммы их работы.

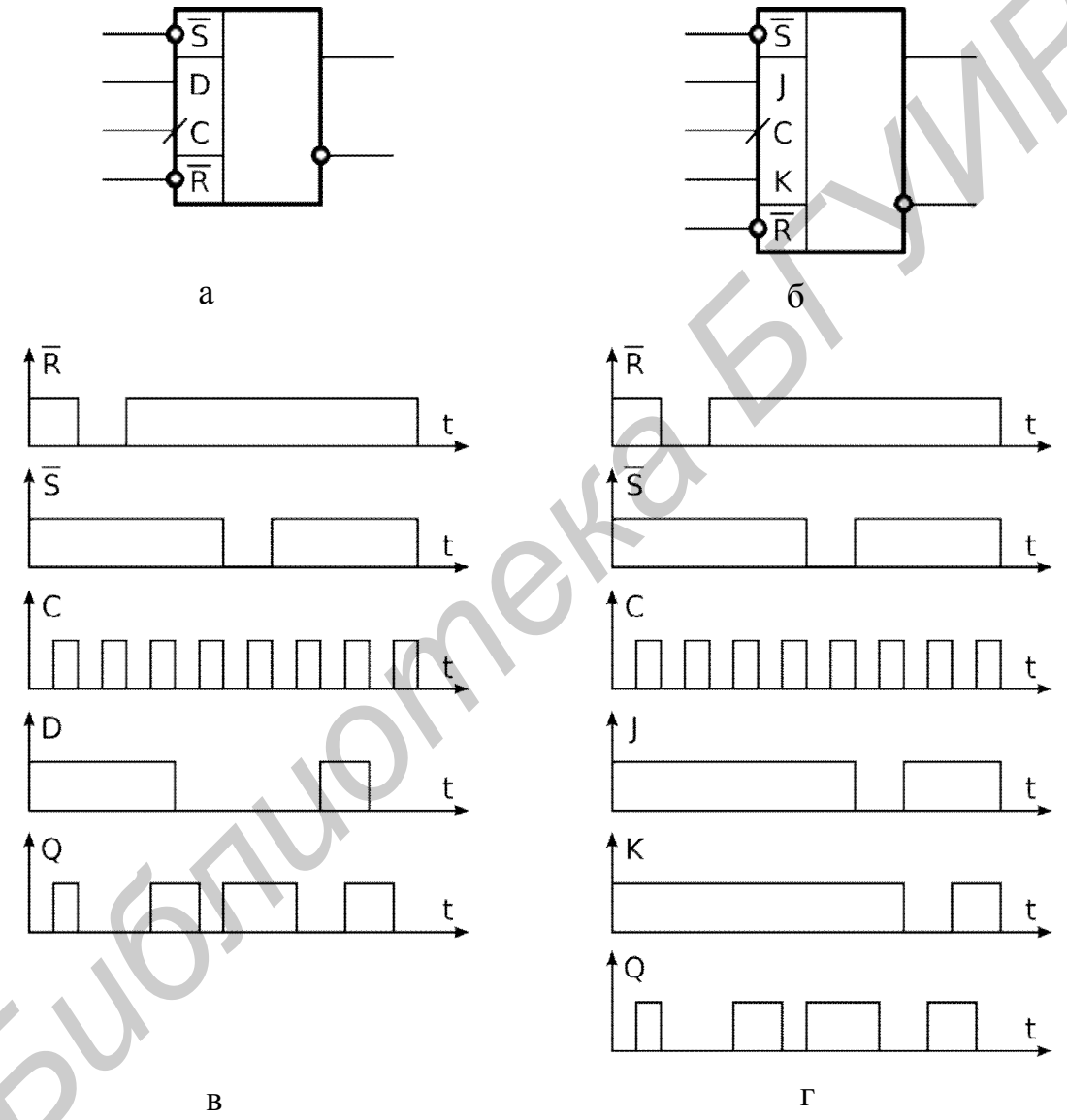

Рис. 2.1. УГО и диаграммы работы динамических триггеров :
а – синхронный динамический D-триггер с асинхронными входами сброса и установки; б – синхронный динамический JK-триггер с асинхронными входами сброса и установки; в – диаграммы работы D-триггера;
г – диаграммы работы JK-триггера

2.4.1. Варианты заданий

Варианты заданий представлены в табл. 2.4.

Таблица 2.4

Номер варианта	Тип триггера	Имя файла описания функционального узла
1	D-триггер	1.pdf
2	JK-триггер	2.pdf
3	D-триггер	3.pdf
4	JK-триггер	4.pdf
5	D-триггер	5.pdf
6	JK-триггер	6.pdf
7	D-триггер	7.pdf
8	JK-триггер	8.pdf
9	D-триггер	9.pdf
10	JK-триггер	10.pdf
11	D-триггер	11.pdf
12	JK-триггер	12.pdf
13	D-триггер	13.pdf
14	JK-триггер	14.pdf
15	D-триггер	15.pdf
16	JK-триггер	16.pdf
17	D-триггер	17.pdf
18	JK-триггер	18.pdf
19	D-триггер	19.pdf
20	JK-триггер	20.pdf
21	D-триггер	21.pdf
22	JK-триггер	22.pdf
23	D-триггер	23.pdf
24	JK-триггер	24.pdf
25	D-триггер	25.pdf

Лабораторная работа №3

3. РАЗРАБОТКА ТЕСТОВЫХ МОДУЛЕЙ НА ЯЗЫКЕ VHDL В СРЕДЕ WEBPACK

Цель: приобрести навыки разработки тестовых модулей на языке VHDL.

3.1. Общие сведения по написанию тестовых модулей на языке VHDL

Основная цель тестовых модулей – проверить функционирование разрабатываемого устройства путем подачи на его вход тестовых воздействий и анализа результатов моделирования его работы. При этом анализ результатов моделирования может выполняться автоматически самим тестовым модулем либо вручную разработчиком.

В целом, тестовый модуль выполняет следующие задачи :

– создает экземпляр объекта для тестирования (моделирования);

– формирует на входе созданного экземпляра объекта тестовое воздействие;

– при необходимости выводит в консоль программы моделирования различную информацию;

– при необходимости проводит автоматическую проверку функционирования тестируемого устройства.

Общая структура тестового модуля показана на рис. 3.1.

Тестовый модуль на языке VHDL представляется проектными модулем ***entity***, который не содержит внешнего интерфейса, то есть портов ввода/вывода. При этом в проектном модуле ***architecture***, представляющем реализацию тестового модуля, создается экземпляр тестируемого объекта, на вход которого подаются сформированные тестовые воздействия.

Тестовые воздействия могут формироваться внутри тестового модуля с помощью конструкций языка VHDL или могут считываться из внешнего источника, например из файла.

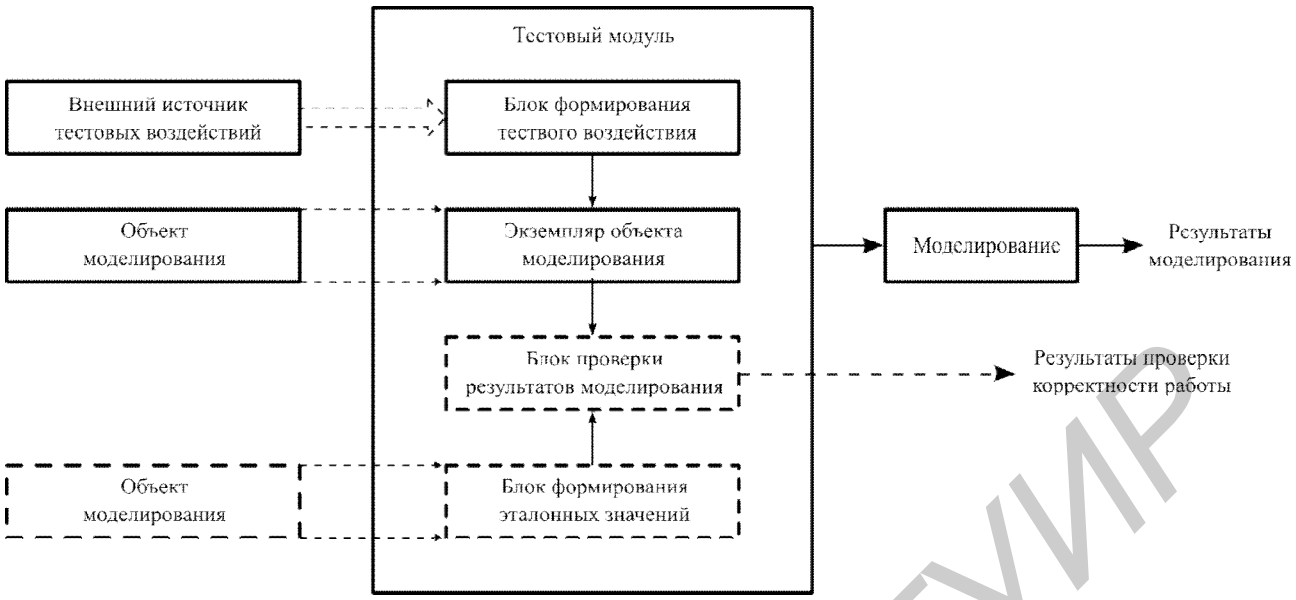

Рис. 3.1. Общая структура тестового модуля

Автоматическая проверка корректности работы устройства выполняется путем сравнения результатов работы устройства, то есть значений его выходных портов, с эталонными значениями. Эталонные значения, в свою очередь, могут быть сформированы также разными способами.

Первый способ – это получить данные значения извне, например считать из файла. Здесь предполагается, что эталонные значения были сформированы каким-либо образом ранее, например созданы самим разработчиком тестируемого устройства.

Второй способ заключается в формировании эталонных значений непосредственно самим тестовым модулем с помощью конструкций языка VHDL. Фактически в этом случае создается модель поведения тестируемого устройства в целом или какой-либо его части в отдельности.

Третий способ заключается в использовании эталонного объекта для получения эталонных значений. В этом случае кроме экземпляра тестируемого объекта дополнительно создается экземпляр эталонного объекта. При этом входные векторы тестовых воздействий подаются одновременно как на тестируемый, так и на эталонный объекты. В результате моделирования на выходе эталонного объекта формируются значения эталона, с которыми

можно сравнивать результаты работы тестируемого объекта. В качестве эталонного объекта часто используют поведенческое описание разрабатываемого устройства, которое редко бывает синтезируемым, зато его можно быстро разработать и отладить.

На рис. 3.2 показана структура тестового модуля в терминах проектных модулей языка VHDL.

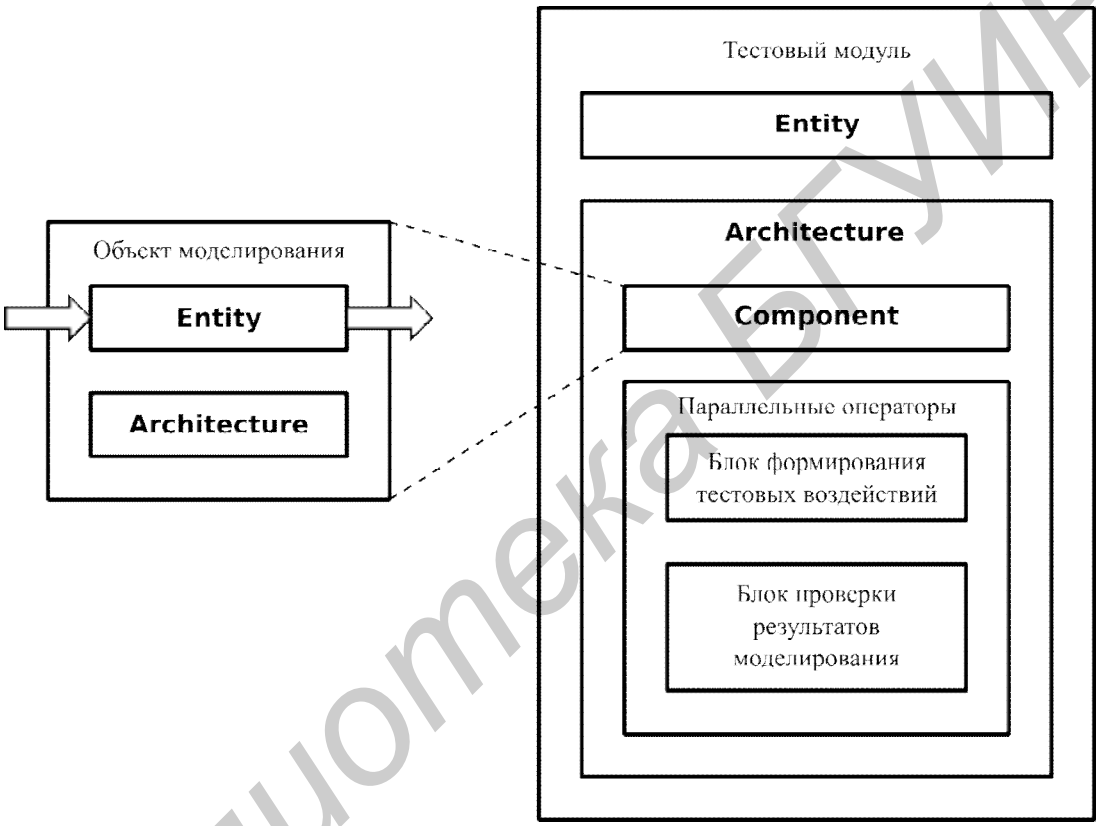

Рис. 3.2. Структура тестового модуля на языке VHDL

3.2. Работа с файлами в языке VHDL

Для работы с файлами в VHDL можно воспользоваться двумя пакетами.

1. Пакет ***std.textio*** предоставляет функции для работы с файлами с использованием стандартных типов данных: ***bit***, ***bitvector***, ***boolean***, ***character***, ***integer***, ***real***, ***string***, ***time***.

2. Пакет ***ieee.std_logic_textio*** предоставляет функции для работы с файлами с использованием типов данных, определенных пакетом ***std_logic_1164***: ***std_ulogic***, ***std_ulogic_vector***, ***std_logic***, ***std_logic_vector***.

Работа с текстовыми файлами выполняется в два этапа. При этом используются два типа функций:

1) ***readline*** или ***writeline***;

2) ***read*** или ***write***.

Какая конкретно функция должна использоваться, зависит от типа операций работы с файлами: чтение или запись.

Функции ***readline*** и ***writeline*** работают непосредственно с файлами и строками в них. Так функция ***readline*** выполняет чтение строки из файла, а функция ***writeline*** – запись строки в файл. Эти функции определены в пакете ***std.textio*** и там же описан тип данных, определяющий считываемые или записываемые строки, – тип ***line***. Функции ***read*** и ***write*** работают со строками и позволяют считать (***read***) из строки или записать (***write***) в строку значения, определенные различными типами данных. Ниже приведен пример использования вышеперечисленных функций для работы с файлами.

```
architecture work_file_example of work_file_example is
 file test_file : text;
begin
 process
      variable file_status : file_open_status;
      variable current_line: line;
      variable test_value      : std_logic_vector(7 downto 0);
 begin
      file_open(file_status,test_file,"test_file");
      read_line(test_file,current_line);
      read(current_line,test_value);
 end process;
end work_file_example;
```

В этом примере выполняется считывание значения типа ***std_logic_vector*** в переменную ***test_value*** из файла ***test_file***. Стоит отметить, что это лишь один из вариантов работы с файлами и вышеперечисленные функции не единственные, которые можно использовать для этого.

3.3. Задание для лабораторной работы

Задание для лабораторной работы состоит из двух частей.

1. Для комбинационного устройства, разработанного в лабораторной работе №1 (вариант с логическими операциями и параллельным оператором безусловного присваивания), создать два варианта тестовых модулей со встроенной проверкой корректности работы устройства.

В первом варианте исходные воздействия, подаваемые на разработанное устройство, и эталонные результаты работы устройства должны считываться тестовым модулем из текстового файла. Текстовый файл может создаваться любым способом (в текстовом редакторе, программой на языке VHDL, C и т.д.).

Во втором варианте входные тестовые воздействия могут формироваться любым способом (считываться из текстового фала, формироваться непосредственно в тестовом модуле и т.д.), а для формирования эталонных воздействий в качестве эталона необходимо использовать любой другой вариант описания этого же устройства из лабораторной работы №1.

Тестовые воздействия в обоих вариантах должны быть полными. Для комбинационного устройства, разработанного в лабораторной работе №1, это означает полный перебор входных значений. В случае обнаружения ошибки в работе устройства необходимо выдать сообщение на консоль программы моделирования.

2. Для функционального узла последовательного типа, разработанного в лабораторной работе ~~№1,~~ создать тестовый модуль с автоматической проверкой корректности работы устройства. Тестовое воздействие может формироваться любым способом и должно быть по возможности полным, то есть тестировать работу устройства во всех его режимах. В случае обнаружения ошибки в работе устройства необходимо выдать сообщение на консоль программы моделирования.

Лабораторная работа №4

4. РАЗРАБОТКА И РЕАЛИЗАЦИЯ ЦИФРОВЫХ УСТРОЙСТВ НА БАЗЕ МАКЕТНОЙ ПЛАТЫ SET-STARTERKIT

Цели :

1. Изучить полный цикл проектирования цифровых устройств на базе ПЛИС с использованием САПР WebPACK.
2. Получить навыки проектирования и отладки цифровых устройств на базе ПЛИС.

4.1. Полный цикл проектирования цифровых устройств в САПР WebPACK

Процесс проектирования цифровых устройств на базе FPGA (см. рис. 1.1) состоит из следующих этапов: ввод описания, синтез, преобразование в физическую реализацию, создание конфигурационного файла и его передача в микросхему FPGA (её программирование). Моделирование описания, включающее в себя как функциональное, так и временное моделирование, выполняется на разных этапах процесса проектирования для контроля полученного описания устройства. Далее более подробно опишем назначение каждого этапа в отдельности.

4.1.1. Ввод описания

Ввод описания разрабатываемого устройства состоит из следующих этапов:

– создание проекта;

– создание новых или добавление в проект уже существующих исходных файлов проектов, включая файлы временных и топологических ограничений;

– ввод описания проектируемого устройства;

– установка значений временных и топологических ограничений.

В предыдущих лабораторных работах выполнялись все этапы ввода проекта, кроме последнего, так как информация, устанавливаемая на этом этапе, необходима только начиная с синтеза проекта.

4.1.2. Синтез проекта

После ввода проекта и выполнения необязательного функционального моделирования необходимо выполнить синтез проекта. Это можно сделать через окно исходных модулей проекта, воспользовавшись командой ***Synthesize-XST***. Синтез может осуществляться как встроенным синтезатором, так и внешней программой, поставляемой в составе других САПР, выполняющих те же функции. В лабораторных работах будет использоваться встроенная программа-синтезатор ***XST***. Во время синтеза выполняется преобразование исходного описания устройства в описание на базе логических блоков и функциональных узлов, для которых впоследствии может быть создана эффективная физическая реализация на базе микросхем FPGA. Фактически выполняется синтез устройства на базе стандартных функциональных схемотехнических узлов.

Результат синтеза выводится в консоль, а также может быть выведен на экран в отдельном окне с помощью команды ***View Synthesis Report***. Кроме этого, результат синтеза можно посмотреть и в графической форме, воспользовавшись командами окна процессов ***View RTL Schematic*** и ***View Technology Schematic***.

Описание устройства, полученное на этом этапе, в большинстве случаев является еще аппаратно независимым, то есть не зависит от микросхемы, на которой оно будет реализовано.

4.1.3. Преобразование в физическую реализацию

Результат синтеза описания устройства используется для получения его физической реализации на заданной микросхеме FPGA. Получение физической реализации выполняется в несколько этапов:

1) ***Translate***;

2) ***Map***;

3) ***Place and Route***;

4) создание конфигурационного файла.

Выполнить данные этапы можно, воспользовавшись соответствующими командами из окна процессов САПР WebPACK (***Processes***).

На *первом* этапе (***Translate***) выполняется преобразование описания, полученного после синтеза проекта, в описание устройства на базе примитивов WebPACK, каждый из которых помещается в базу данных используемых элементов.

На *втором* этапе (***Map)*** каждый элемент из базы данных, полученной на предыдущем этапе, реализуется на компонентах, входящих в состав заданной микросхемы FPGA.

На *третьем* этапе (***Place and Route)*** выполняется размещение и разводка описания устройства, полученного на предыдущем этапе на заданной микросхеме.

На *последнем* этапе выполняется генерация конфигурационного файла (файла прошивки). Для этого используется команда ***Generate Programming File*** окна процессов программы ***ProjectNavigator***.

4.1.4. Загрузка конфигурационного файла в микросхему FPGA

После того как получен конфигурационный файл для прошивки FPGA, необходимо загрузить его в микросхему.

Лабораторная работа выполняется с использованием платы SET-StarterKit, ядром которой является микросхема FPGA Spartan II XC2S200. Данная плата не позволяет напрямую загружать конфигурационную информацию в FPGA. Вместо этого необходимо сформировать файл прошивки для микросхемы FLASH-памяти XC18V02, размещенной на этой же плате, и загрузить его в эту микросхему, воспользовавшись интерфейсом JTAG. После включения питания либо

при нажатии кнопки «Program» конфигурационная информация из этой микросхемы будет автоматически загружена в FPGA.

Для формирования файла прошивки FLASH-памяти можно воспользоваться командой ***Generate PROM, ACE or JTAG File*** окна процессов программы ***ProjectNavigator***, при этом загрузится специализированный модуль САПР WebPACK – ***IMPACT***, предназначенный для формирования файлов прошивок конфигурационной последовательности и загрузки этих файлов в микросхемы посредством использования различных интерфейсов.

После загрузки модуля ***IMPACT*** автоматически запускается мастер, позволяющий облегчить процесс выполнения необходимой операции, которая может быть выбрана на первой вкладке данного мастера. Для формирования файла прошивки FLASH-памяти на *первом* этапе необходимо выбрать операцию ***Prepare a PROM File***.

На *втором* этапе работы мастера (вкладка ***Prepare PROM Files***) необходимо указать устройство, для которого будут сформированы прошивка (***Xilinx PROM***), тип файла прошивки (***MCS***), имя файла прошивки (***PROM File Name***) и путь, куда будет записан сформированный файл.

На *третьем* этапе (вкладка ***Specify Xilinx PROM Device***) создается список микросхем памяти, для которых будет генерироваться файл прошивки. Здесь необходимо выбрать тип используемой микросхемы FLASH-памяти (***XC18V02***) и добавить ее в список.

На *последнем* этапе выводится для проверки общая информация по выполняемой операции, после чего запрашивается имя файла конфигурационной последовательности, сгенерированного на предыдущих этапах. После выбора необходимого файла и отрицательного ответа на запрос о добавления другого устройства необходимый файл прошивки FLASH-памяти будет сгенерирован модулем ***IMPACT***, о чем будет сообщено всплывающим информационным окном.

После получения файла прошивки FLASH-памяти можно приступать к его

загрузке в микросхему. Для этого в модуле IMPACT необходимо перейти в режим работы с использованием JTAG интерфейса (***Boundary Scan)***. После этого загрузка файла прошивки в микросхему FLASH-памяти также осуществляется в несколько этапов.

На *первом* этапе необходимо запустить процедуру JTAG сканирования для определения на макетной плате микросхем, поддерживающих JTAG. В случае успешного завершения этой операции будут обнаружены две микросхемы: ПЛИС FPGA Spartan II (XC2S200) и конфигурационная FLASH-память (XCV18V02).

На *втором* этапе на запрос файла для программирования FLASH-памяти (файл с расширением ***MCS***) указывается файл, сгенерированный ранее. При этом на запрос файла прошивки для FPGA (файл с расширением ***BIT***) необходимо нажать кнопку ***BYPASS***, что означает отсутствие конфигурационного файла для FPGA.

После этого необходимо, воспользовавшись контекстным меню по нажатию правой кнопки мыши на изображении FLASH-памяти, вызвать процедуру ее программирования.

После успешного завершения процедуры загрузки конфигурационной последовательности во FLASH-память можно воспользоваться кнопкой «Program» для загрузки конфигурационной информации в FPGA.

4.1.5. Подключение FPGA на плате SET-StarterKit

Ядром платы SET-StarterKit является микросхема ПЛИС FPGA Spartan II (XC2S200), тактируемая сигналом внешнего тактового генератора с частотой 50 Мгц. Кроме этого, на плате есть набор светодиодов, подключенных к выводам микросхемы FPGA, работающим в режиме выходов, и набор переключателей, подключенных к выводам микросхемы FPGA, работающим в режиме входов. Схема подключения описанных выводов микросхемы FPGA изображена на рис. 4.1.

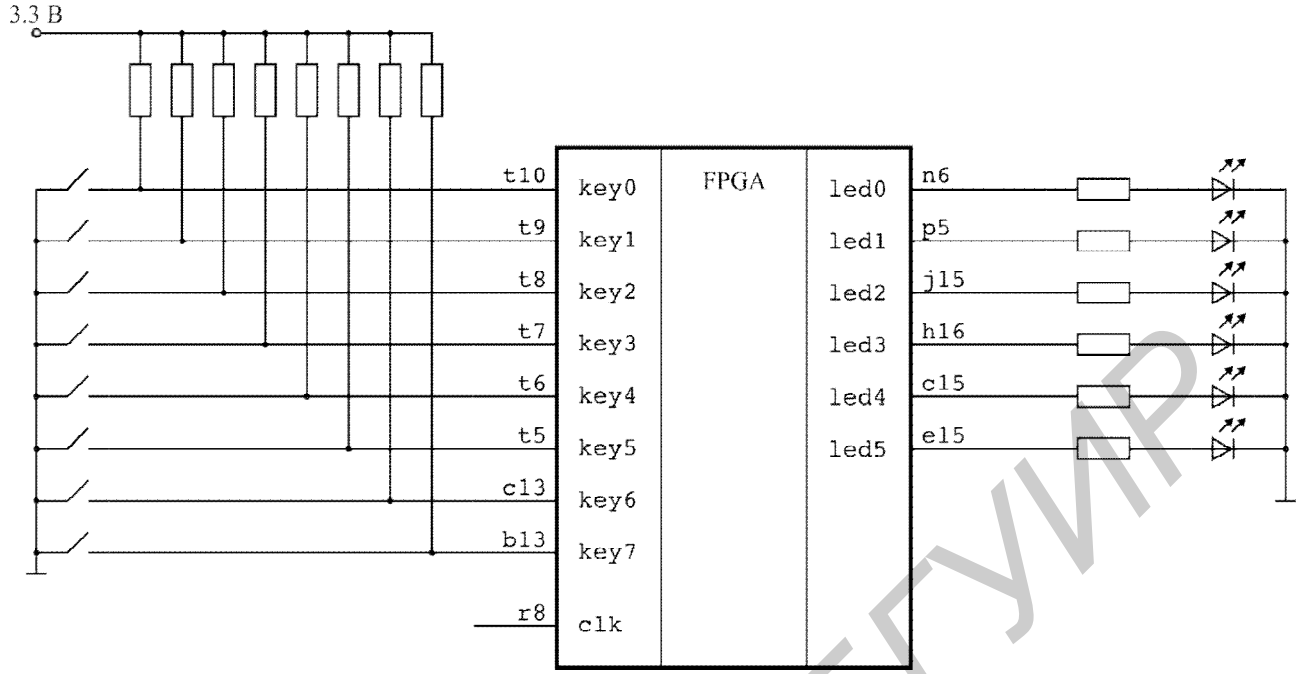

Рис. 4.1. Схема подключения микросхемы FPGA на макетной плате

4.1.6. Файл временных и топологических ограничений

Файл временных и топологических ограничений содержит дополнительную информацию для программ синтеза, размещения и трассировки. Этот файл имеет текстовый формат, каждая строка которого представляет собой выражение, описывающее соответствующий параметр. Чаще всего файл временных и топологических ограничений используют для задания соответствия между входами/выходами разработанного устройства и выводами микросхемы FPGA.

Ниже приведен пример файла временных и топологических ограничений, который можно с небольшими изменениями использовать в своем проекте для выполнения лабораторной работы:

```
# Синхросигнал 50Мгц от внешнего генератора
net clk loc = r8;
# Светодиодные индикаторы
net leds<0> loc = n6;
net leds<1> loc = p5;
net leds<2> loc = j15;
net leds<3> loc = h16;
net leds<4> loc = c15;
net leds<5> loc = e15;
```

```
# Переключатели
net keys<0> loc = t10;
net keys<1> loc = t9;
net keys<2> loc = t8;
net keys<3> loc = t7;
net keys<4> loc = t6;
net keys<5> loc = t5;
net keys<6> loc = c13;
net keys<7> loc = b13;
```

4.2. Задание для лабораторной работы

На базе своего варианта функционального узла последовательного типа, разработанного во второй лабораторной работе, спроектировать и реализовать на базе макетной платы одно из следующих устройств.

1. Счетчик, увеличивающийся (уменьшающийся) на 1 с частотой, заданной преподавателем. При этом в качестве компонента обязательно использовать счетчик, разработанный в лабораторной работе №2. Выходы данного счетчика подключить к выводам микросхемы FPGA, соединенным со светодиодами (см. рис. 4.1), а управляющие входы счетчика – к выводам микросхемы FPGA, подключенным к переключателям.

2. Регистр, сохраняющий значения с частотой, заданной преподавателем. В качестве компонента обязательно использовать регистр, разработанный в лабораторной работе №2. Выходы регистра соединить с выводами микросхемы FPGA, подключенными к светодиодам, а управляющие входы – с выводами микросхемы FPGA, подключенными к переключателям.

Для разработанного устройства создать тестовое воздействие и выполнить его функциональное (поведенческое) моделирование .

Выполнить синтез спроектированного устройства. При необходимости внести коррективы в его описание.

Провести функциональное моделирование описания устройства, полученного после его синтеза.

Выполнить реализацию (получение физического описания) устройства на базе микросхемы FPGA Spartan II XC2S200-6FG256.

Провести временное моделирование полученного описания устройства и добиться его корректной работы. При необходимости внести изменения в описание проектируемого устройства.

Сгенерировать файл конфигурационной последовательности для микросхем FPGA и FLASH-памяти.

Выполнить программирование микросхемы на макетной плате и убедиться в корректной работе спроектированного устройства.

Св. план 2008, поз. 55

Учебное издание

Байрак Сергей Анатольевич
Татур Михаил Михайлович

АВТОМАТИЗАЦИЯ ПРОЕКТИРОВАНИЯ ЭВМ

Лабораторный практикум
для студентов специальности 1-40 02 01
«Вычислительные машины, системы и сети»
всех форм обучения

Редактор Н. В. Гриневич
Корректор М. В. Тезина
Компьютерная верстка Е. Г. Бабичева

Подписано в печать 15.08.2008. Формат 60x84 1/16. Бумага офсетная.
Гарнитура «Таймс». Печать ризографическая. Усл. печ. л. 2,79.
Уч.-изд. л. 2,3. Тираж 120 экз. Заказ 193.

Издатель и полиграфическое исполнение: Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
ЛИ №02330/0056964 от 01.04.2004. ЛП №02330/0131666 от 30.04.2004.
220013, Минск, П. Бровки, 6