Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования
«Томский государственный университет систем управления и радиоэлектроники»
(ТУСУР)

УТВЕРЖДАЮ

Заведующий кафедрой УИ
_____ А.Ф. Уваров
«____»_____2012 г.

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ К ЛАБОРАБОРНЫМ РАБОТАМ, ПРАКТИЧЕСКИМ ЗАНЯТИЯМ И САМОСТОЯТЕЛЬНОЙ РАБОТЕ

по дисциплине

«ОСНОВЫ ПРОЕКТИРОВАНИЯ СИСТЕМ НА КРИСТАЛЛЕ»

Составлена кафедрой «Управление инновациями»

Для студентов, обучающихся
по направлению подготовки 222000.68 «Инноватика»
Магистерская программа «Управление инновациями в электронной технике»

Форма обучения очная

Составитель
Доцент _____ О.Г. Пономарев

«___» _____ 2012 г.

Томск 2012

СОДЕРЖАНИЕ

Лабораторная работа №1. Использование генератора Core Generator System

Введение

Работа проводит вас через весь процесс создания IP Core с помощью системы Xilinx CORE Generator™ и включение этого ядра в ваш дизайн

Цели

По завершению этой работы вы сможете:

- Создавать IP Core, используя Xilinx CORE Generator
- Встраивать IP Core в существующий HDL дизайн
- Выполнять поведенческую симуляцию HDL дизайна, который содержит IP Core

Методика

В этой работе вы будете использовать Xilinx CORE Generator для создания block-RAM, которая содержит программу для PicoBlaze, и тестировать этот дизайн на плате Digilent Spartan-3.

Эта работа состоит из четырех основных частей; вы просмотрите дизайн, создадите IP Core, внедрите IP Core в дизайн и проведете поведенческую симуляцию нового *loopback* модуля.

Создание Файла для инициализации ROM — Шаг 1

Запустите ISE™ Project Navigator и откройте файл проекта.

❶ Для того чтобы запустить Xilinx ISE, выберите **Start → Programs → Xilinx ISE → Project Navigator**

❷ Выберитет **File → Open Project**

c*:\xup\fpgaflow\labs\vhdl\lab5\coregen*

❸ Выберите ***coregen_lab.ise*** и нажмите **Open**

Используя программу из предыдущей лабораторной работы, вы сможете закончить задачу №3 и собрать .coe файл, который будет использован для инициализации ROM.

❶ Откройте файл с программой ассемблера program.psm, расположенный в директории проекта

❷ Завершите задачу №3. Напишите кусок кода, который будет будет возвращать символы, переданные через гипертерминал обратно.

❸ Откройте окно команд, найдите директорию, содержащую компилятор ассемблерной программы и введите командную строку > kcpsm3 program

Замечание: компилятор сгенерирует несколько файлов, включая и .COE, который будет оспользоваться для инициализации памяти.

Создание IP Core — Шаг 2

Создайте IP Core (Single Port Block Memory), и назовите его *program.*

❶ В окне Processes для Source, дважды щелкните на **Create New Source**

Если вы не видите процесс Create New Source, убедитесь, что HDL файл выбран в качестве источника в окне Project.

❷ В диалоге New Source, выберите **IP (CoreGen & Architecture Wizard)** и напечатайте ***program*** в поле для ввода имени (File Name), как показано на **Рис. 1**

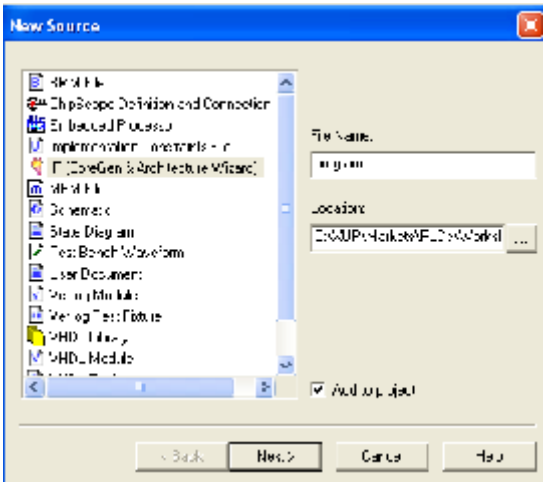

Рисунок 5.1 – Диалог создания нового объекта

❸ Нажмите **Next**

❹ В диалоге Select Core Type, раскройте **Memories & Storage Elements**, далее раскройте **RAMs & ROMs**, и выберите **Dual Port Block Memory v6.1**, как показано на **Рис. 2**

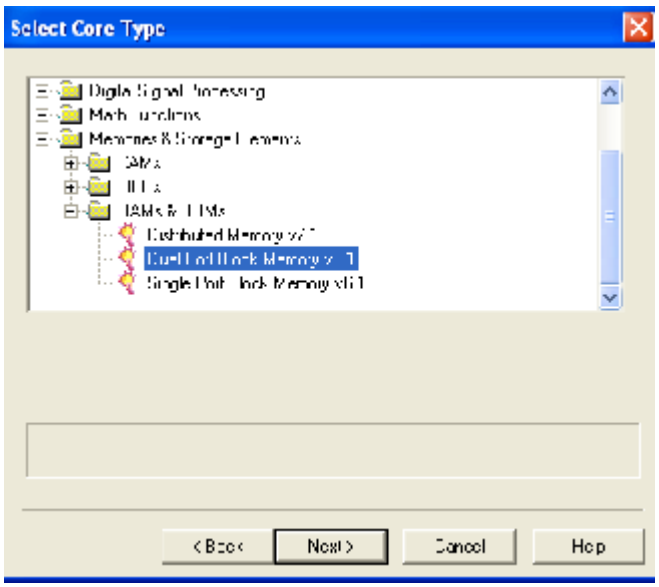

Рисунок 5.2 – Диалог Выбора Типа IP Core

❺ Нажмите **Next** и затем **Finish**

После этого откроется графический интерфейс пользователя системы CORE Generator™ .

Создайте Dual Port Block Memory со следующими параметрами:

- Имя: **program**
- Параметры порта A: **1024 x 18 memory size, Read Only**
- Параметры порта B: **1024 x 18 memory size, Read and Write, Read After Write**

6. Установите следующие параметры, как показано на **Рис. 3**, и нажмите <Next>

   - Component Name: ***program***
   - Width A: **18**
   - Depth A: **1024**
   - Width B: **18**
   - Depth B: **1024**
   - Port A Configuration**: Read Only**
   - Port B Configuration: **Read and Write**
   - Port B Write Mode: **Read After Write**

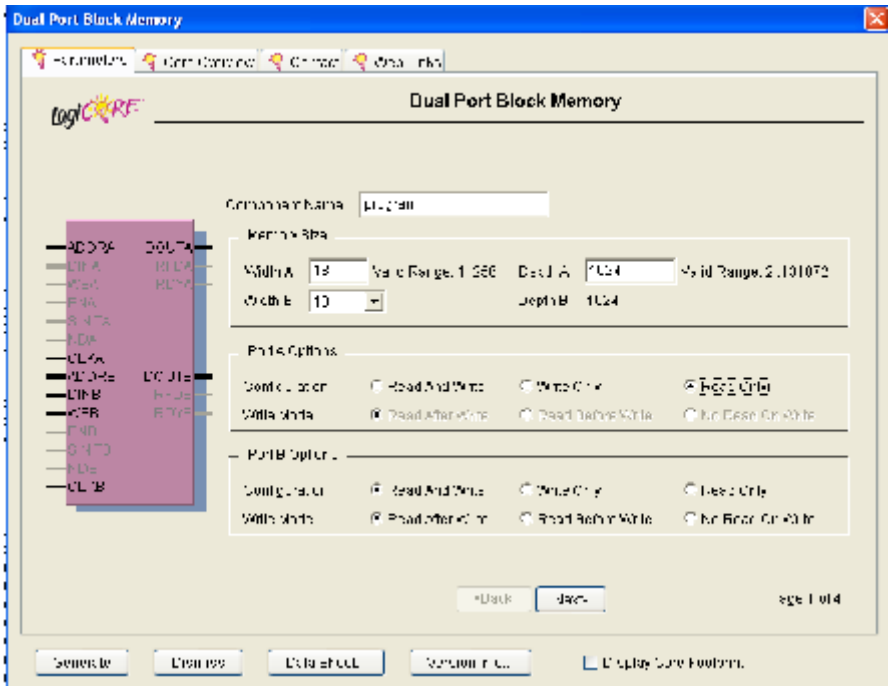

Рисунок 5.3 – Установки при Создании Блок Памяти

❷ Оставьте без изменения значения по умолчанию, как показано на **Рис. 4**, и нажмите <**Next**>.

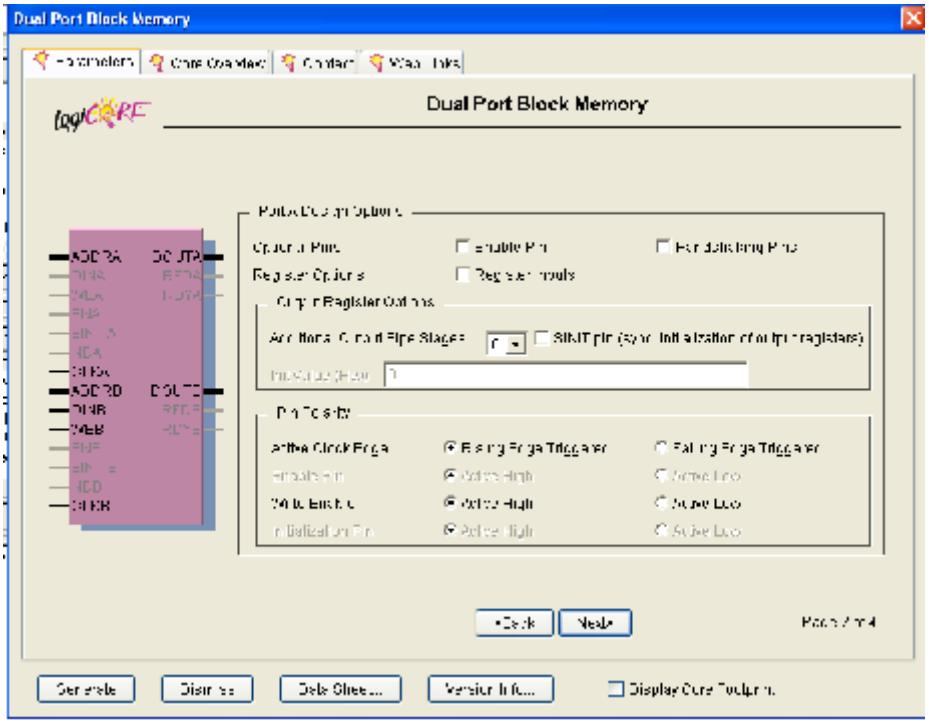

Рисунок 5.4 – Опции дизайна и полярность выводов для порта A

❸ Оставьте без изменения значения по умолчанию, как показано на **Рис. 5**, и нажмите <**Next**>.

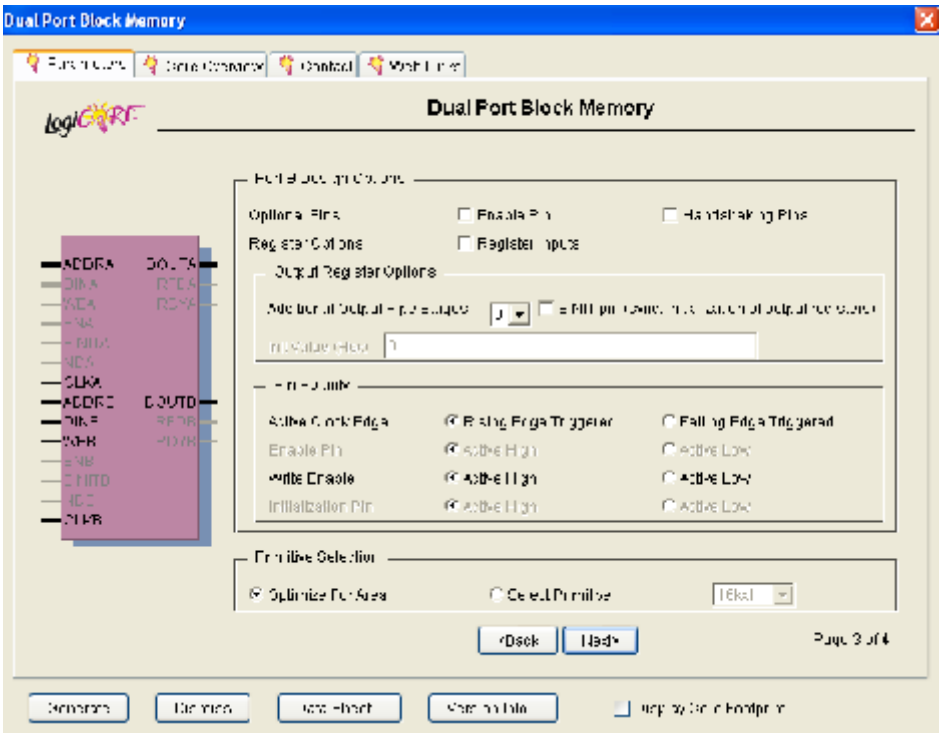

Рисунок 5.5 – Опции дизайна и полярность выводов для порта B

❹ Нажмите **Load Init File** и выберите **program.coe** файл из директории с ассемблерным кодом программы

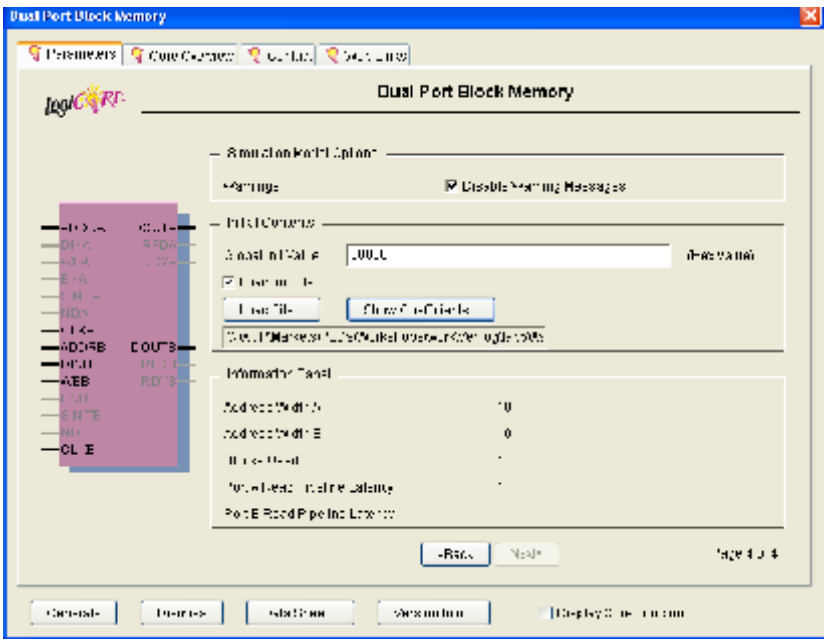

Рисунок 5.6 – Инициализация памяти с программой

❺ Нажмите **Generate** снизу в окне диалога Dual Port Block Memory

❻ В ISE, следуйте в **Project → Add Source** и добавьте файл *program.xco* в проект.

Встраивание Block RAM Core в код на VHDL **Шаг 3**

Встройте IP Core, сгенерированную раньше, в файл *loopback.vhd.*

❶ В окне Project для Sources, дважды щелкните на ***loopback.vhd***

Файл *loopback.vhd* откроется в окне текстового редактора.

❷ Выберите **Edit → Language Templates**

Шаблон для IP Core расположен в секции COREGEN этого окна.

❸ Раскройте **COREGEN**, раскройте **VHDL Component Instantiation**, и выберите ***program***, как показано на **Рис. 7**

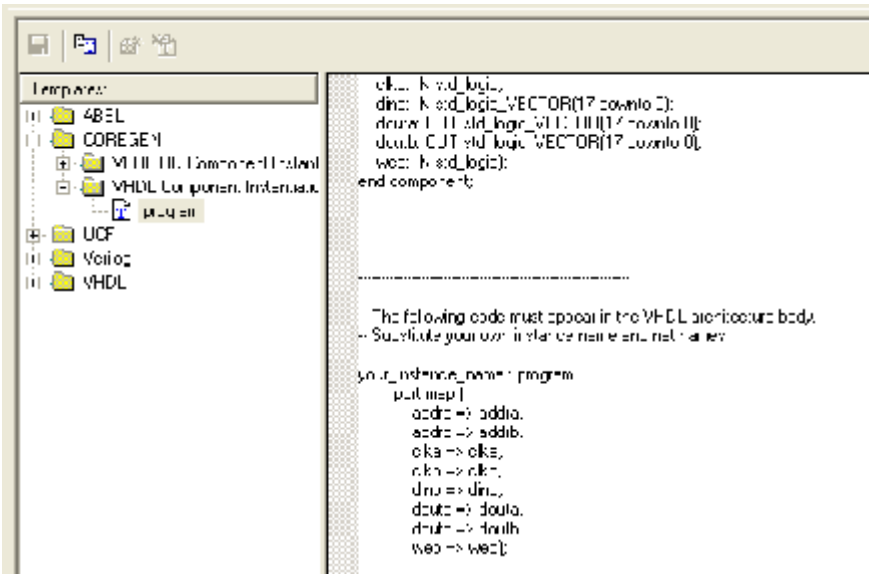

Рисунок 5.7 – Шаблон для реализации

❹ Скопируйте **шаблон** а затем вставьте его в файл *loopback.vhd*

Вставьте **component declaration** в заголовок архитектуры после комментария “---- Insert component declaration for the RAM block here”.

Вставьте **component instantiation** в описание архитектуры после комментария “-- insert component instantiation for the Block RAM here”.

❺ Отредактируйте **instantiation** следующим образом:

```
my_program : program
 port map
 (
   addra => address,
   addrb => "0000000000",
   clka  => clk,
   clkb  => '0',
   dinb  => "0000000000000000",
   douta => instruction,
   doutb => open,
   web   => '0'
 );
```

❻ Выберите **File → Save**

Выполнение Behavioral Simulation **Шаг 4**

Внимательно изучите файл *testbench.vhd* и функциональную модель IP Core.

❶ В окне Sources в Project, дважды щелкните на ***testbench.vhd***

❷ В окне Sources в Project, выберите ***program.xco***

❸ В окне Processes для Source, раскройте панель инструментов **COREGen** и дважды щелкните на **View Verilog/VHDL Functional Model**

Этот файл ссылается на модели из библиотеки симуляций XilinxCoreLib и используется автоматически, если поведенческая симуляция запускается из Project Navigator ISE™.

Пользователям VHDL: Если файл не появляется в текстовом редакторе, нажмите правую кнопку мыши на **View VHDL Functional Model** и выберите **Open Without Updating**.

Используя файл *testbench.vhd* , запустите поведенческую симуляцию для 5000 ns. Просмотрите полученную диаграмму сигналов для подтверждения того, что ядро подсоединено правильно.

❶ В окне Sources в Project, выберите ***testbench.vhd***

❷ В окне Processes для Source, раскройте процесс **Xilinx ISE Simulator**, нажмите правую кнопку мышки на **Simulate Behavioral Model**, и выберите **Properties**

❸ Введите время «Simulation Run Time» - ***5000 ns***

❹ Нажмите **OK**

❺ Дважды щелкните на Simulate Behavioral Model

❻ Изучите диаграмму сигналов и убедитесь, что симуляция происходит так, как и ожидалось.

Тестирование дизайна на кристалле Шаг 5

Откройте сеанс работы с гипертерминалом. Сгенерируйте bitstream, загрузите его в плату Digilent Spartan 3 и протестируйте разработанное приложение.

❶ В директории с проектом, дважды щелкните на terminal.ht для начала сеанса работы с гипертерминалом

❷ С выбранным *loopback.vhd* в Project Navigator, раскройте **Generate Programming file** и дважды щелкните на **Configure Device (iMPACT)**.

❸ Когда **iMPACT** откроется, сконфигурируйте FPGA с помощью кабеля JTAG так, как это делалось в предыдущих лабораторных работах.

- Boundary-Scan Mode
- Automatically connect cable

❹ Нажмите <OK>, когда откроется окно с сообщением о том, что были найдены два устройства в JTAG-цепочке

❺ Назначьте loopback.bit для Spartan-3 xc3s200 и **Bypass** для PROM

❻ Нажмите правую кнопку мышки на устройстве Spartan-3 в **iMPACT** и выберите **program**. Нажмите <OK>.

Замечание: Вы должны увидеть сообщение “Xilinx Rules!” в окне гипертерминала. Далее, любые сообщения, которые вы будете печатать на клавиатуре, также должны появиться в окне гипертерминала.

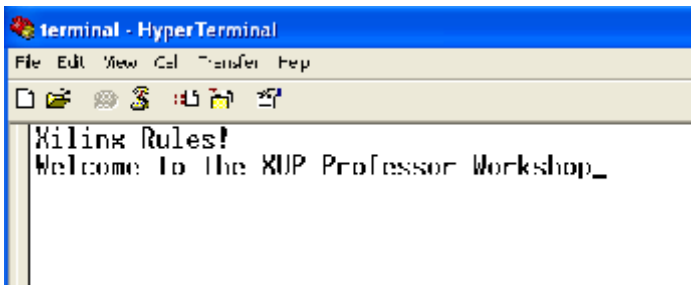

Рисунок 5.9 – Вид окна гипертерминала

Заключение

Используйте систему CORE Generator™ для создания IP Core, которые вы легко можете интегрировать в ваш VHDL-дизайн.

Лабораторная работа №2. Моделирование динамических систем дискретного времени в среде MatLab/Simulink

Simulink является средой моделирования, интегрированной со средой научных расчетов MatLab. Первые шаги по созданию новой модели выполняются в среде MatLab.

1. Запустите MatLab.
2. Выберите рабочую директорию для модели. Пиктограмма, на которую следует нажать, чтобы появилось окно диалога для выбора рабочей директории указана на рис. 1 стрелкой.

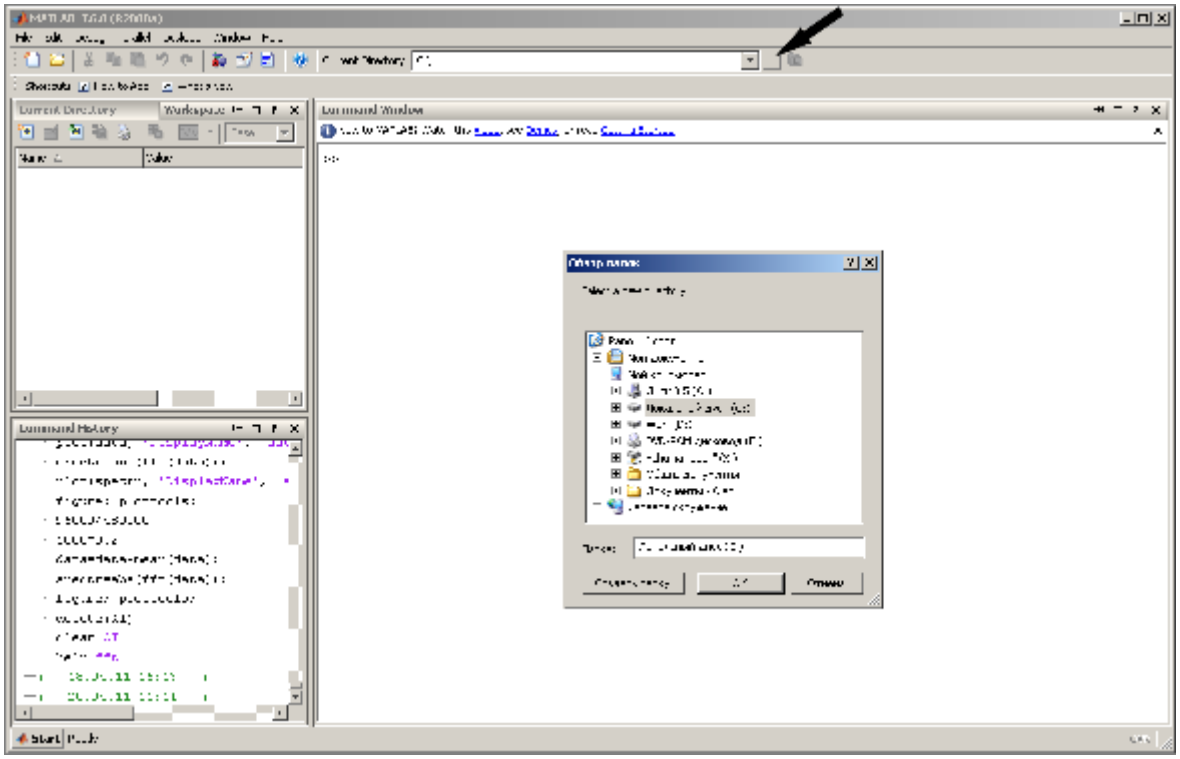

Рисунок 1 – Окно выбора рабочей директории программы MatLab

3. Запустите Simulink, нажав на пиктограмму на панели инструментов MatLab (рис. 2).

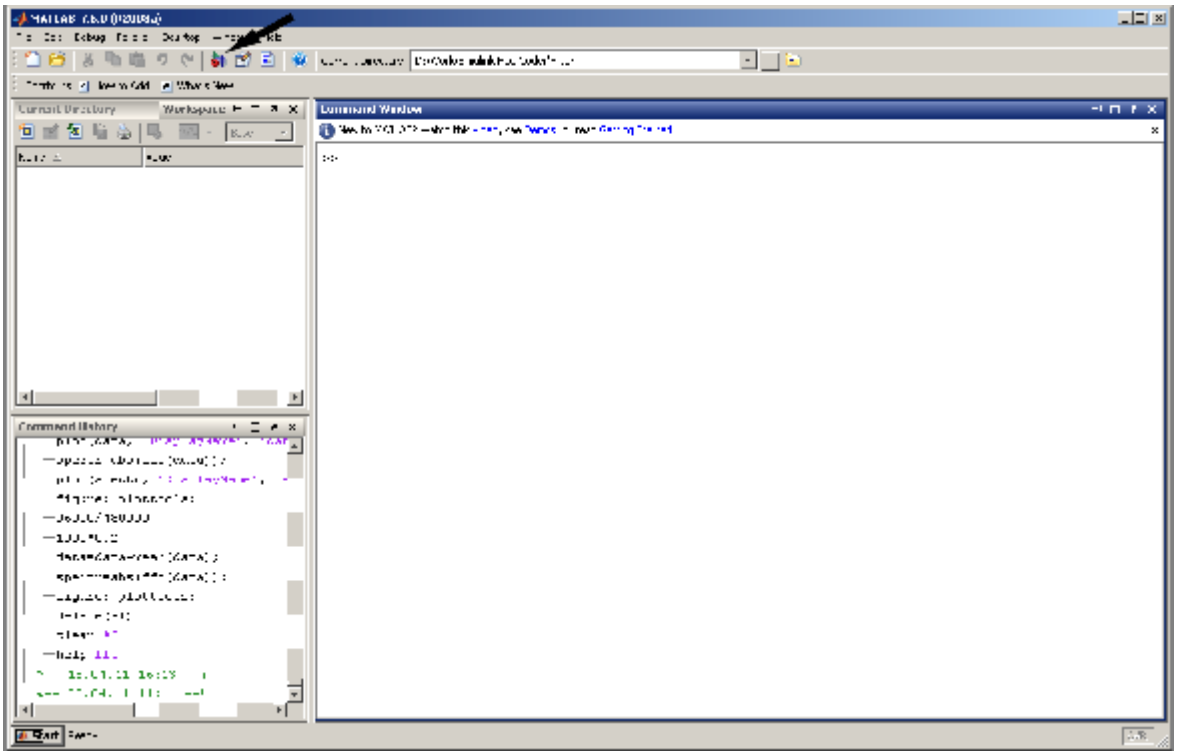

Рисунок 2 – Пиктограмма Simulink на панели инструментов MatLab

В результате откроется окно Simulink Library Browser (рис. 3), в панели инструментов которого находится пиктограмма создания новой модели.

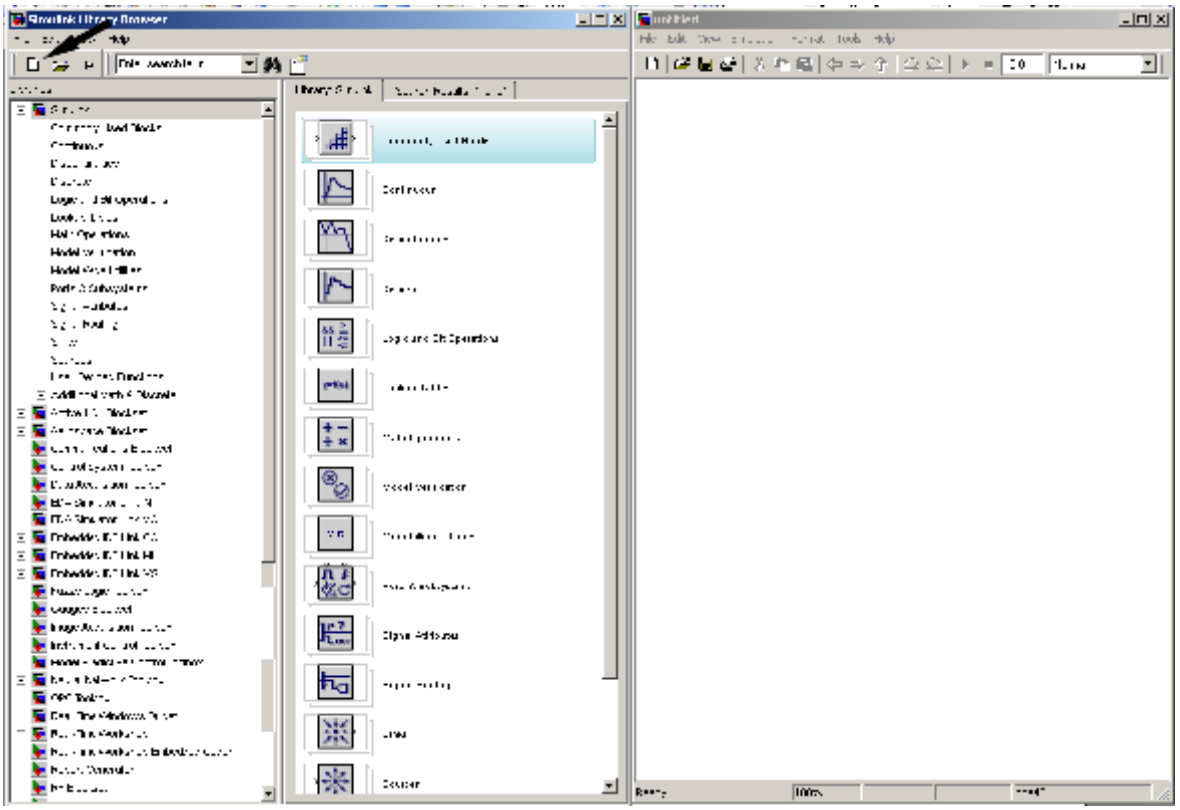

Рисунок 3 – Пиктограмма создания новой модели в панели инструментов Simulink Library Browser

После нажатия на эту пиктограмму откроется окно графического редактора модельных диаграмм. Прежде всего, необходимо сохранить новую модель, нажав на соответствующую пиктограмму в панели инструментов редактора.

4. Произведите настройку параметров модели. Для этого переключитесь в Command Window программы MatLab и наберите команду: **hdlsetup**

По этой команде будет произведена настройка параметров модели, допускающих генерацию из нее в дальнейшем HDL-описания. После настройки параметров и сохранения модели можно приступать к созданию модельной диаграммы (при сохранении модели укажите имя MyFilter). Этот процесс заключается в перетаскивании блоков из Simulink Library Browser и соединении выходных и входных портов блоков сигналами. Далеко не все блоки, предоставляемые Simulink Library Browser, могут быть использованы в модели, предназначенной для генерации HDL. Список всех блоков, пригодных для этой цели, можно получить, набрав в Command Window программы MatLab команду: **hdllib**

5. В разделе Discret в Simulink Library Browser выберите и перетащите в окно модели блок Unit Delay (рис.4). Размножьте блок в окне модели 8 раз (для этого при выделении блока в окне модели необходимо держать нажатой клавишу Ctrl).

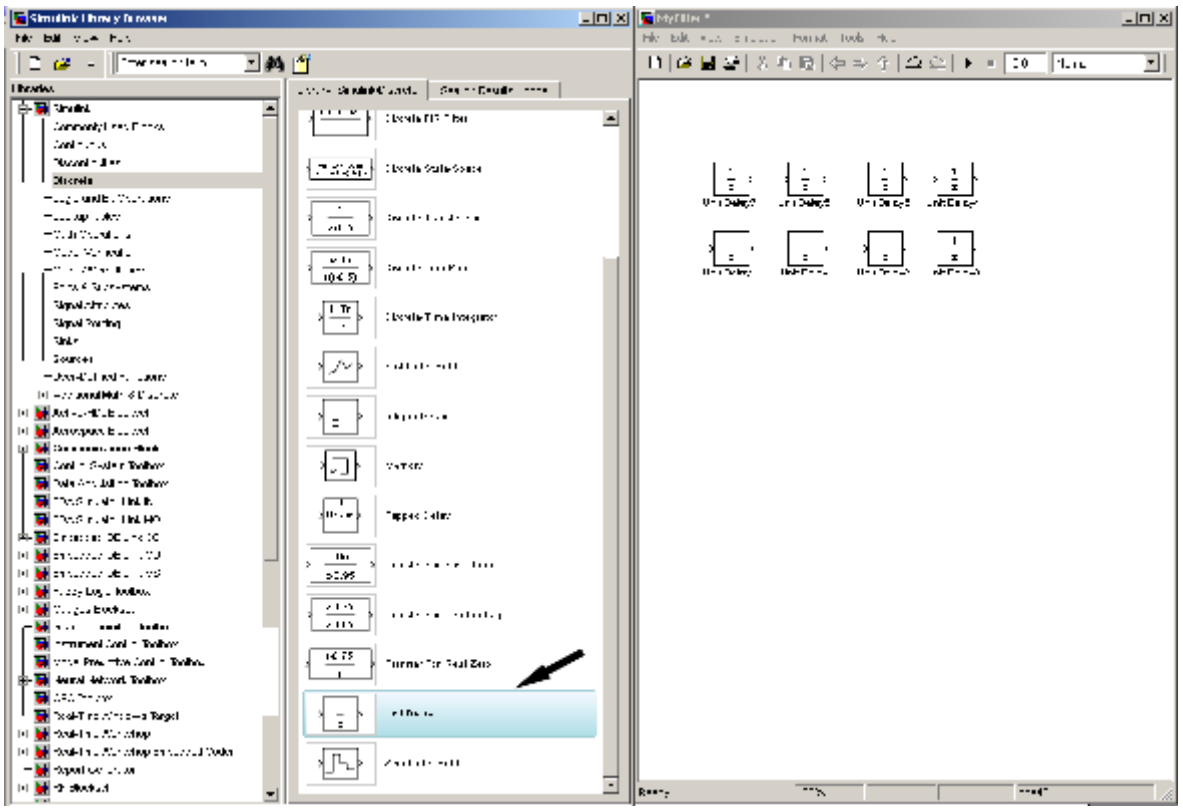

Рисунок 4 – Блок Unit Delay в разделе Discrete программы Simulink Library Browser

6. Разверните 4 блока верхнего ряда на 180 градусов. Для этого выделите эти блоки и в выпадающем меню, доступном по нажатии правой кнопки «мыши», выберите позицию Format/Rotate block (рис. 5).

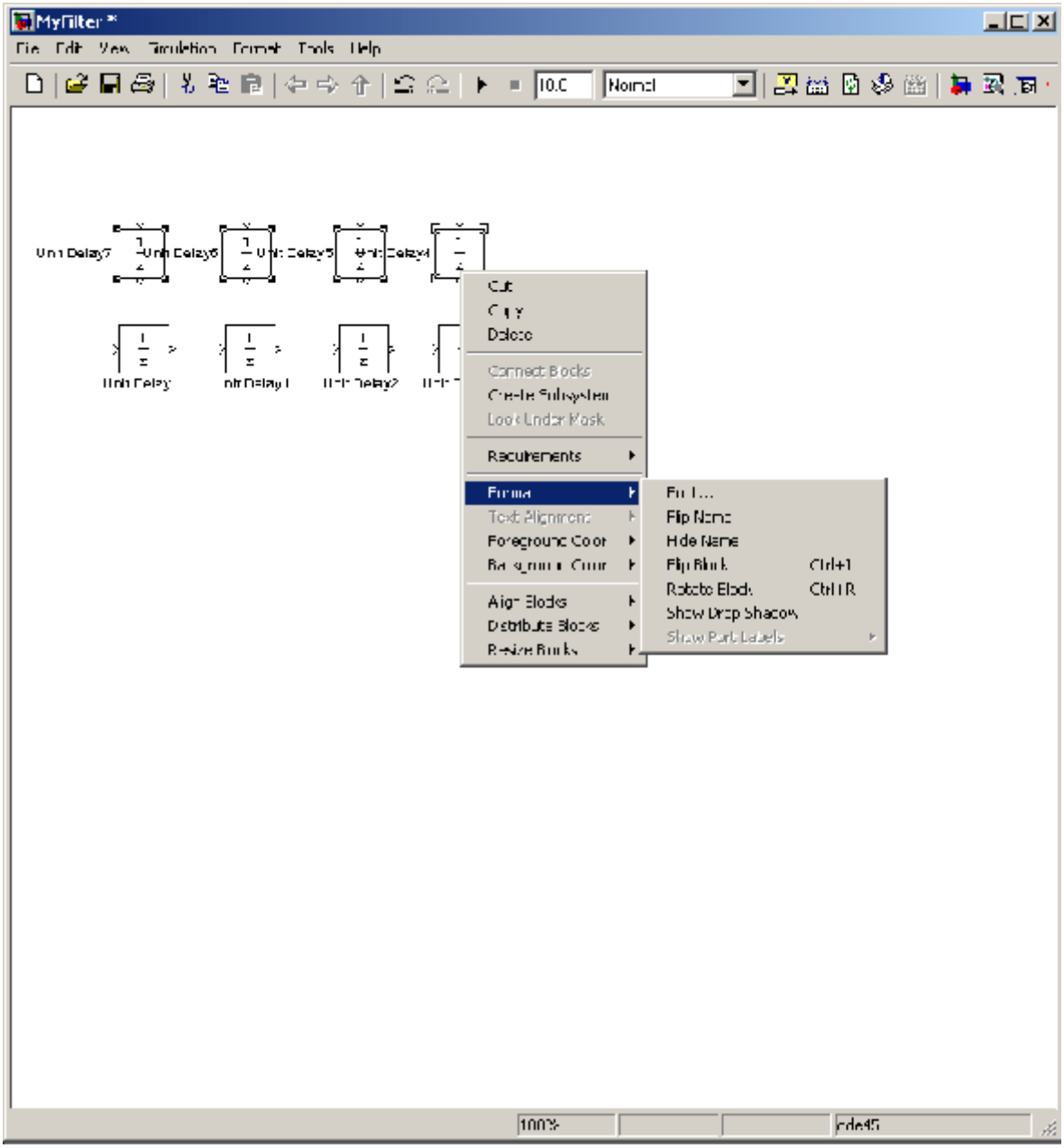

Рисунок 5 – При нажатии на правую кнопку «мыши» становится доступным выпадающее меню

7. Добавьте к модели блок Add из раздела Math Operations в Simulink Library Browser и размножьте его 7 раз. Из этого же раздела добавьте к модели блок Product и размножьте его 4 раза (рис. 6).

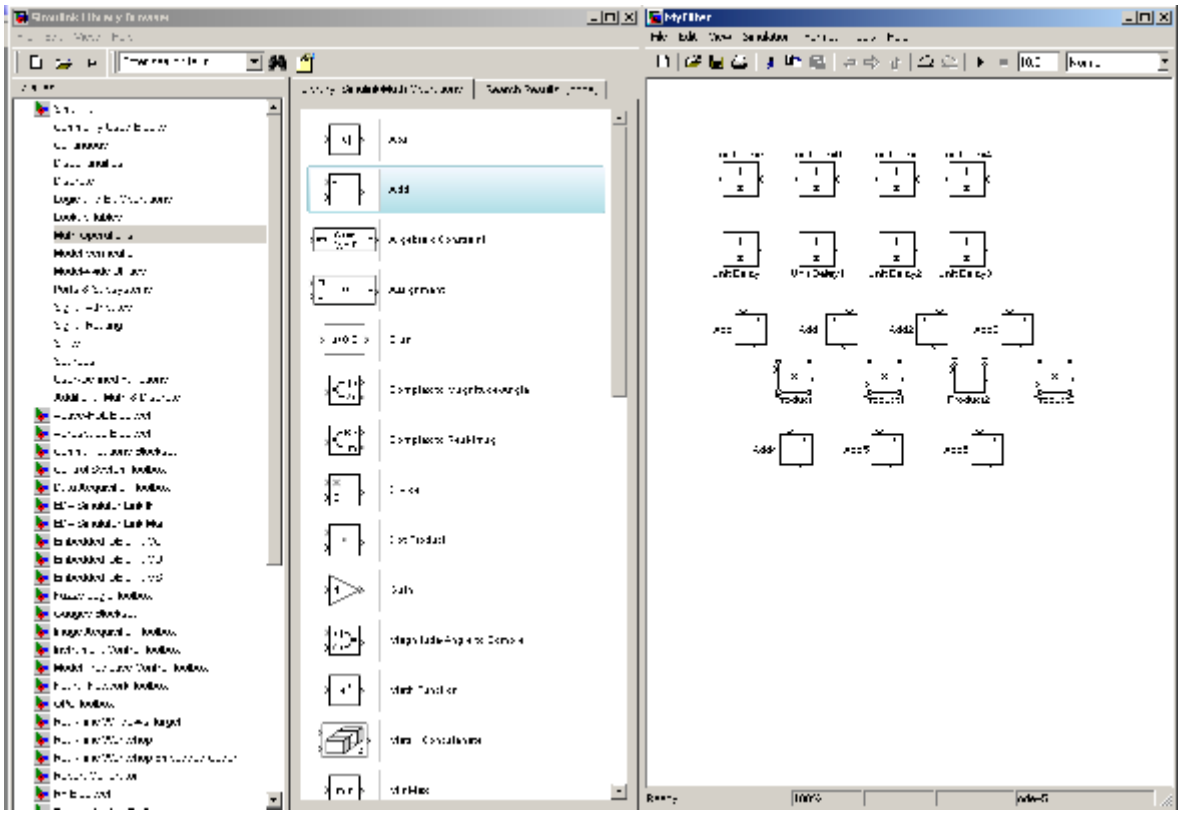

Рисунок 6 – Блок Add находится в разделе Math Operations в Simulink Library Browser

8. Начиная с выходного порта блока Unit Delay соедините порты модели сигналами, как показано на рис. 7. (Для соединения выходного порта блока с входным портом другого блока необходимо нажать на левую кнопку «мыши» на выходном порте и протянуть соединение, не отпуская левой кнопки «мыши», до входного порта. Для создания разветвлений сигнала, необходимо нажать на левую кнопку мыши в точке на сигнале, где создается разветвление, держа при этом нажатой клавишу Ctrl клавиатуры).

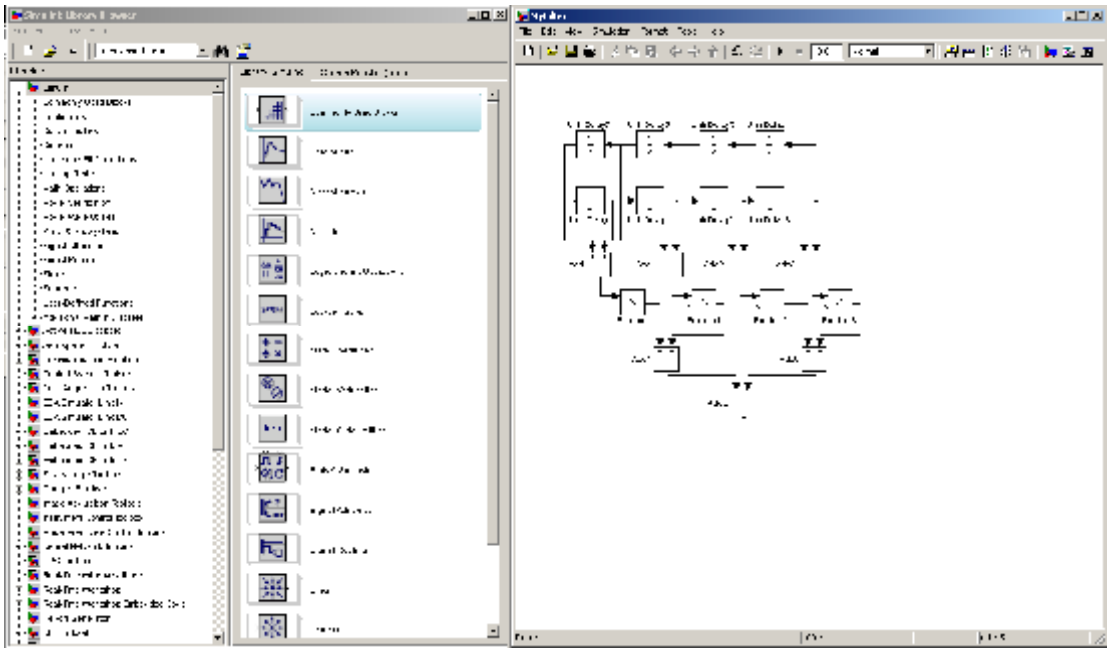

Рисунок 7 – Соединение блоков модели сигналами

9. Из раздела Sources в Simulink Library Browser добавьте к модели блок From Workspace. (Этот блок лучше добавить слева от существующих блоков модели. Если слева оказывается мало места, то существующие блоки можно легко сдвинуть. Для этого необходимо выделить все на диаграмме модели, обведя область с блоками и сигналами «мышью», или нажав комбинацию клавиш Ctrl+A. После этого, используя «мышь» или клавишу «стрелка вправо» клавиатуры можно сдвинуть блоки и сигналы модели на необходимое расстояние вправо).

10. Дважды щелкните по левой кнопке «мыши» над добавленным блоком From Workspace. Откроется окно диалога, позволяющее задать параметры блока (рис. 8). В строке Data введите следующее выражение (в этом выражении используется язык программирования MatLab):

[[0:2000].' cos(2.*pi.*[0:0.001:2].*(1+[0:0.001:2].*75)).'].

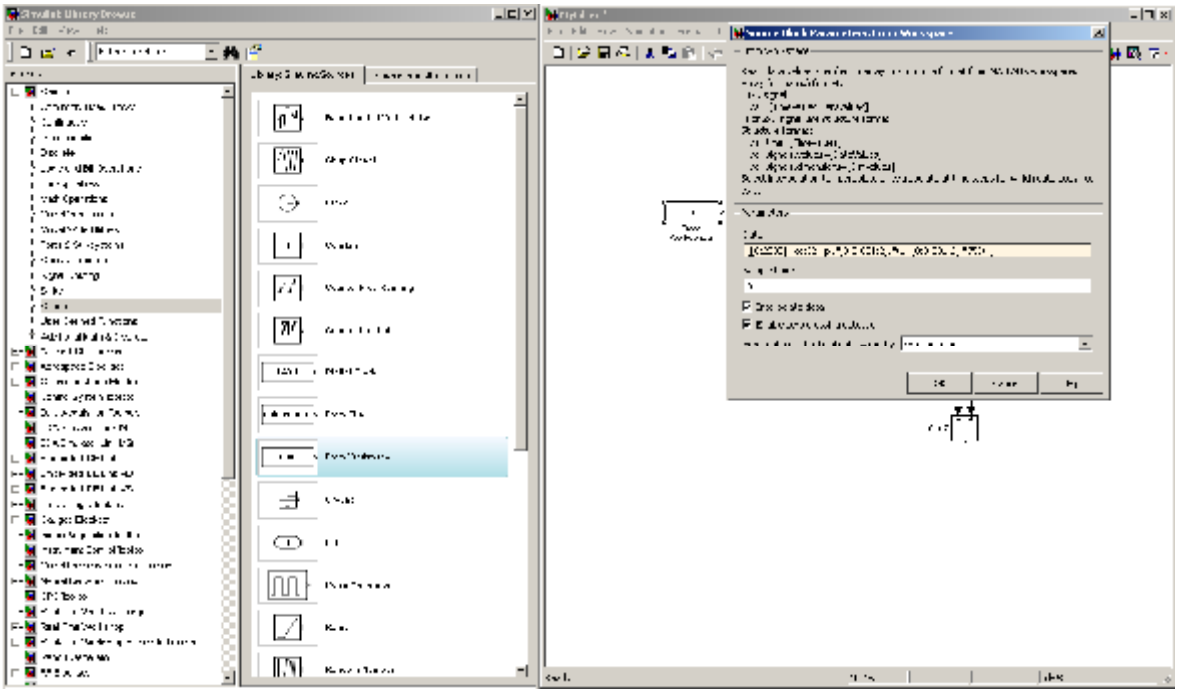

Рисунок 8 – Окно диалога для ввода параметров блока From Workspace откроется при двойном щелчке «мыши» по этому блоку после добавления его к модели

11. Добавьте блок Constant к модели из раздела Sources в Simulink Library Browser. Этот блок будет служить для задания одного из коэффициентов фильтра. В качестве параметра этого блока (дважды щелкнув по нему «мышью») задайте число -0.1339. На вкладке Signal Attributes в поле Output data type задайте fixdt(1,16,10).

12. Размножьте блок Constant 4 раза. В качестве параметра каждого нового блока задайте числа: -0.0838, 0.2026, 0.4064.

13. Добавьте блок Data Type Conversion из раздела Signal Attributes в Simulink Library Browser. В поле Ounput Data Type окна диалога параметров этого блока задайте fixdt(1,16,10).

14. Измените разрядность выходных сигналов умножителей (Product, Product1, Product2, Product3). Для этого двойным щелчком «мыши» на каждом из блоков откройте окно диалога для задания параметров блока. Перейдите на вкладку Signal Attributes и в поле Output data type укажите fixdt(1,16,10). (В предлагаемой схеме на вход каждого из

умножителей поступает по два сигнала, один из которых является 16-ти разрядным, а второй 17-ти разрядным. Результат умножения, таким образом, окажется 33-х разрядным. Simulink не допускает генерации HDL кода по модели, в которой есть более чем 32-х разрядные сигналы)

15. Добавьте блок Scope из раздела Sinks в Simulink Library Browser. Двойным щелчком «мыши» по этому блоку откройте окно Scope и нажмите на пиктограмму Parameters. В появившемся окне в поле Number of Axes установите 2.

16. Соедините добавленные блоки сигналами, как показано на рис. 9.

17. Установите время симуляции в панели инструментов модели равным 2000.

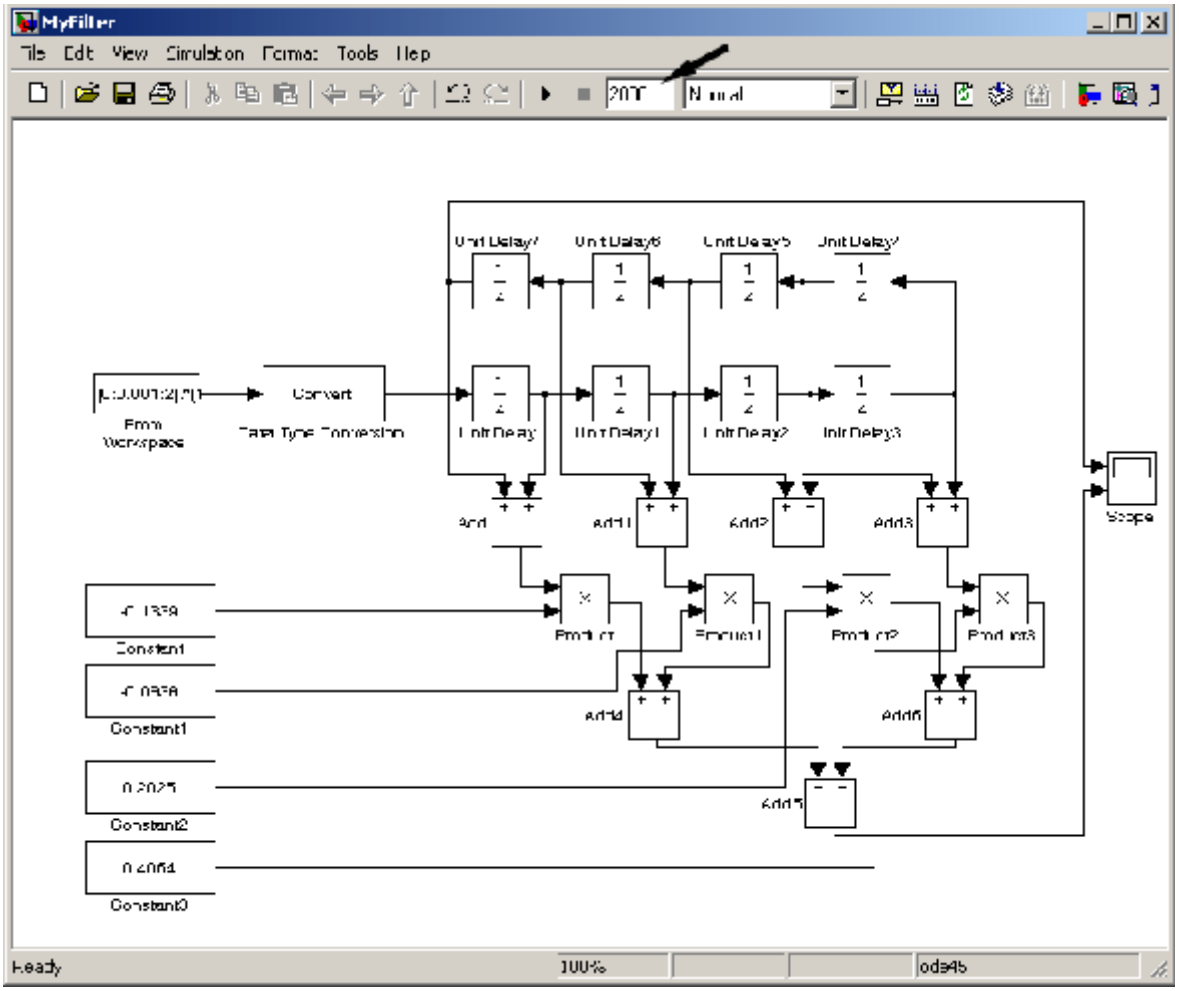

Рисунок 9 – Стрелкой на рисунке указано место, где можно задать время симуляции модели

18. Запустите моделирование, нажав на кнопку Start Simulation на панели инструментов окна модели.

После завершения симуляции в окне Scope будут отображены два графика: верхний – входной сигнал фильтра, задержанный на 8 тактов (временных шагов) работы модели; нижний – результат фильтрации входного сигнала. При необходимости размеры окна Scope легко меняются при помощи «мыши». Щелчок правой кнопкой «мыши» в окне Scope открывает доступ к выпадающему меню. Позиция Autoscale этого меню позволяет автоматически подстраивать масштабы сигнала на каждом графике.

Лабораторная работа №3. Особенности настройки модели для обеспечения возможности генерации HDL

Расширение среды моделирования Simulink, называемое Simulink HDL Coder, позволяет выполнять автоматическую генерацию кода на одном из языков описания аппаратуры (Verilog или VHDL) для цифрового устройства, модель которого создана в Simulink. Это средство в ряде случаев может существенно повысить скорость разработки и отладки цифровых устройств, позволяя разработчику сосредотачиваться на алгоритмах обработки сигналов в разрабатываемом цифровом устройстве и его архитектуре, а не на кодировании этих решений на языке описания аппаратуры.

Процесс генерации VHDL кода по готовой модели очень прост. Во-первых, разработчик должен запустить утилиту Simulink HDL Coder compatibility checker, которая проверит совместимость модели устройства с HDL Coder. Второй шаг – собственно генерация VHDL-описания модели. Третий шаг – генерация VHDL-программы TestBench для тестирования сгенерированного кода.

Здесь необходимо отметить несколько моментов. Прежде всего, если посмотреть на диаграмму модели цифрового фильтра (рис. 9), то станет ясно, что далеко не все блоки модели относятся непосредственно к реализации разработанного фильтра. Так, например, блоки From Workspace, Data Type Conversion, Scope, Constant, Constant1 и т.д. предназначены для моделирования входного (фильтруемого) сигнала, отображения результата фильтрации на графике, задания коэффициентов фильтра. Эти блоки, таким образом, моделируют окружение (среду), в котором функционирует разработанный фильтр. Если говорить в терминах языка описания цифровой аппаратуры, то эти блоки относятся к описанию TestBench. Очевидно, что перед генерацией VHDL-кода для разработанного фильтра, необходимо отделить блоки модели, относящиеся непосредственно к архитектуре фильтра, от тех, что моделируют условия его работы. Такое разделение в Simulink можно осуществить с помощью блока Subsystem и специального файла, называемого Control file.

Объединение набора блоков в Subsystem в Simulink выполнить очень просто. Выполним это в модели цифрового фильтра.

1. С помощью «мыши» выделите в модели те блоки, которые относятся непосредственно к реализации архитектуры фильтра (это все блоки модели за исключением, как уже указывалось, блоков From Workspace, Data Type Conversion, Scope, Constant, Constant1, Constant2, Constant3).
2. Не снимая выделения, нажмите на правую кнопку «мыши» и из выпадающего меню выберите позицию Create Subsystem (рис. 10).

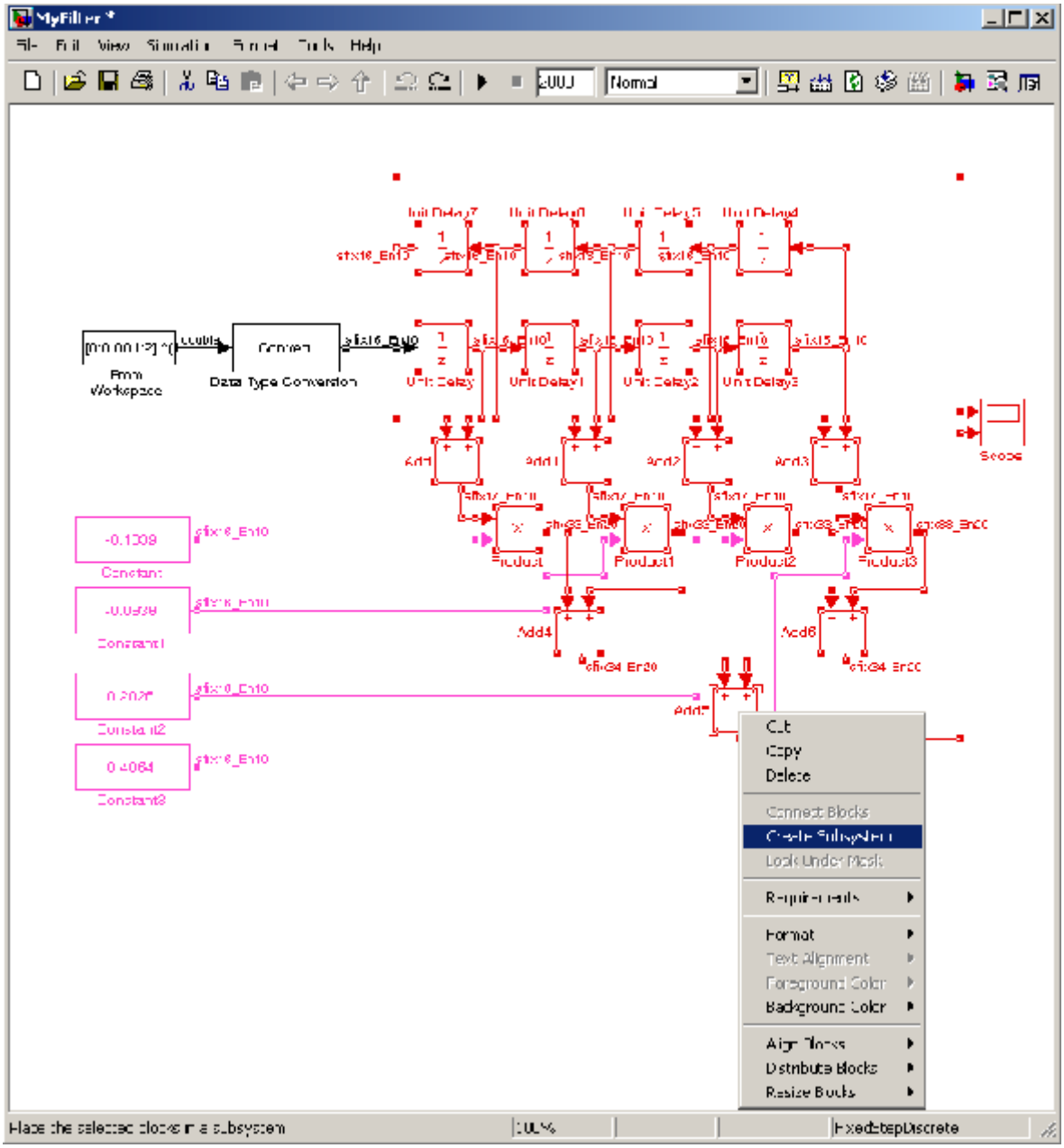

Рисунок 10 – Для объединения блоков в подсистему выберите из контекстного меню, появляющегося при нажатии на правую кнопку «мыши», позицию Create Subsystem

В результате, выделенные блоки будут объединены в подсистему, а в диаграмме модели они будут заменены одним блоком с именем Subsystem. Щелчок левой кнопкой «мыши» по надписи Subsystem позволяет изменить имя подсистемы (например, на Digital Filter). Двойной щелчок левой кнопки «мыши» по блоку подсистемы открывает содержимое подсистемы в отдельном окне. Отметим, что входные и выходные порты подсистемы имеют стандартные названия In1, In2, …, Out1, Out2, … . Их можно поменять на более информативные названия, щелкнув левой кнопкой «мыши» в поле имени порта. Осмысленные названия подсистем и портов в модели существенно повышают ее удобочитаемость. (При необходимости, опять же для повышения читаемости диаграммы модели, размеры блока подсистемы можно поменять с помощью «мыши». Можно также

повернуть блок подсистемы, используя команду Format/Rotate контекстного меню, выпадающего при нажатии на правую кнопку «мыши»).

3. Поменяйте названия портов подсистемы Digital Filter так, чтобы модель выглядела как на рис. 11.

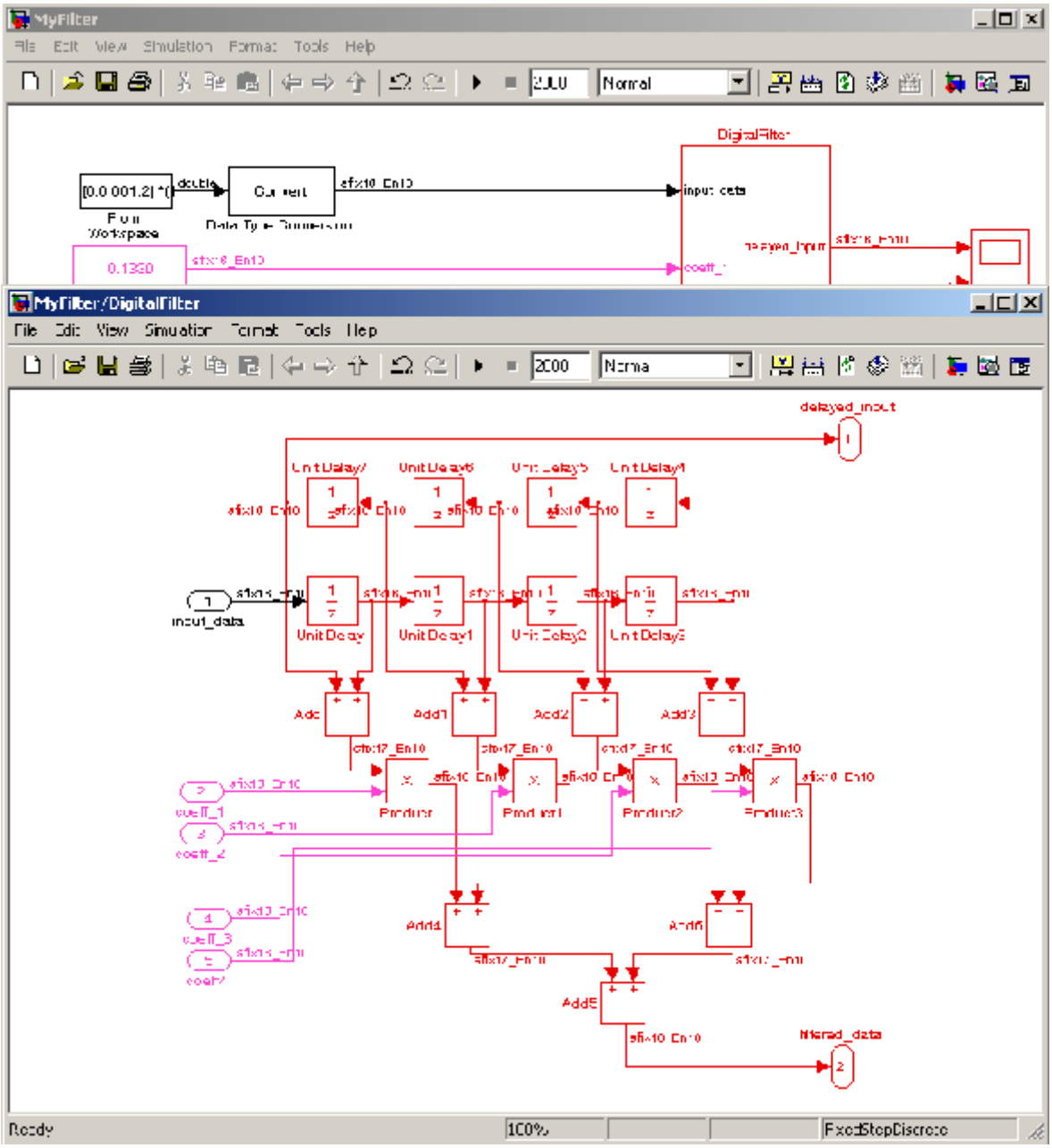

Рисунок 11 – Поменяйте имена входных и выходных портов подсистемы для повышения читаемости диаграммы модели

Следующий шаг, необходимый для правильной генерации HDL-кода – написание Control File. Наиболее просто создать правильный Control File используя скрипт, поставляемый вместе с расширением HDL Coder.

4. В меню Simulation окна модели выберите позицию Configuration Parameters. В левой части открывшегося окна диалога (дерево Select) выберите позицию HDL Coder. Из

выпадающего списка Generate HDL for выберите MyFilter/DigitalFilter. Из выпадающего списка Language выберите VHDL (рис. 12).

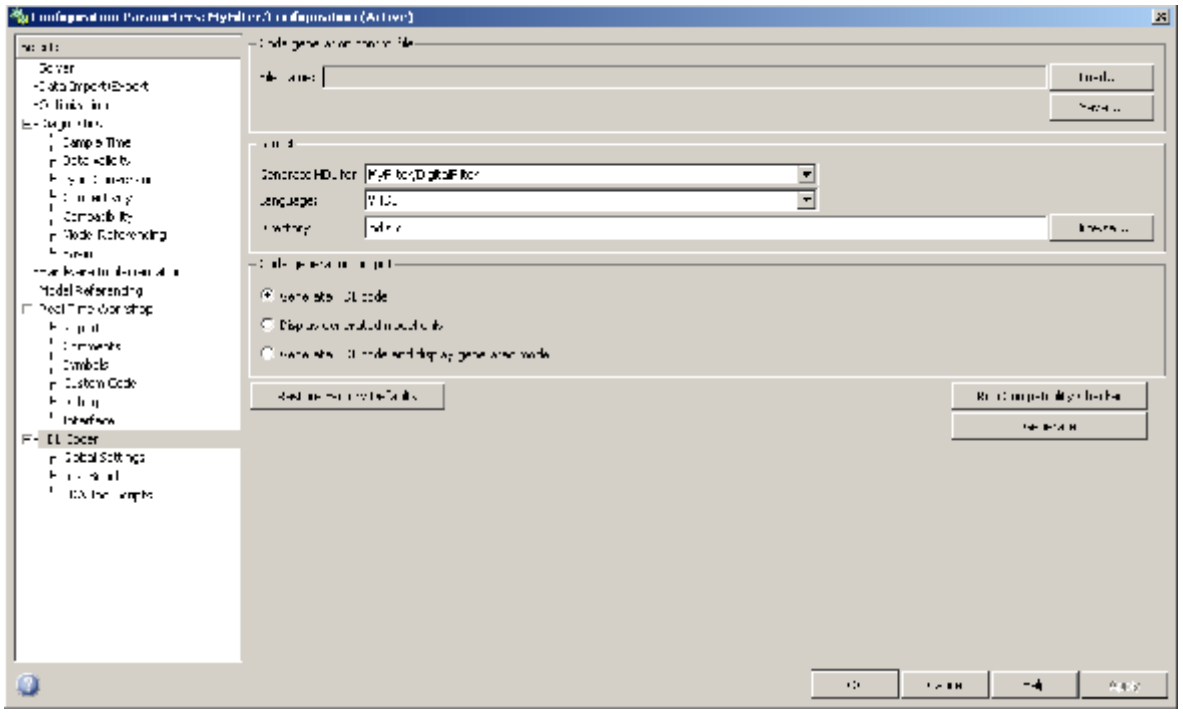

Рисунок 12 – Установите правильные параметры генерации HDL-кода

5. В Command Window среды MatLab наберите команду:

hdlnewcontrolfile(block),

где аргумент block – полный путь к подсистеме, представляющей собой модель цифрового устройства. В нашем случае в качестве аргумента необходимо набрать:

hdlnewcontrolfile('MyFilter/DigitalFilter')

(Путь к подсистеме указывается в одинарных кавычках, элементы пути разделяются символом «/»).

В результате выполнения скрипта в текстовом редакторе MatLab будет создан новый текстовый файл с именем controlfilename. При сохранении этого файла его имя лучше поменять, например, на MyFilterControlFile.

6. Укажите созданный MyFilterControlFile в качетве Code Generation Control File. Для этого переключитесь в окне диалога Configuration Parameters нажмите на кнопку Load и в открывшейся директории выберите для загрузки MyFilterControlFile (рис. 13).

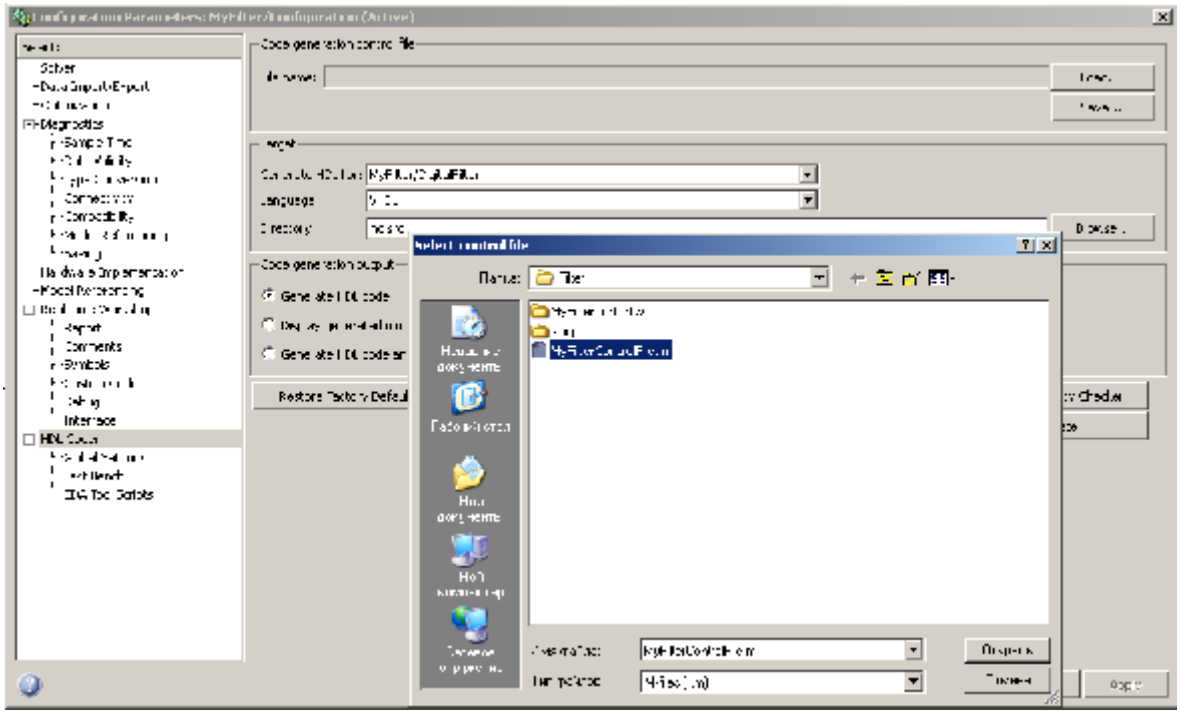

Рисунок 13 – Выберите сгенерированный MyFilterControlFile в качестве Code generation control file

Модель готова к генерации VHDL-кода.

7. В окне диалога Configuration Parameters нажмите на кнопку Run Compatibility Checker.

Утилита Compatibility Checker выполняет проверку модели на совместимость с Simulink HDL Coder. По результатам проверки выполняется генерация HTML-файла, который открывается в WEB-браузере системы (рис. 14).

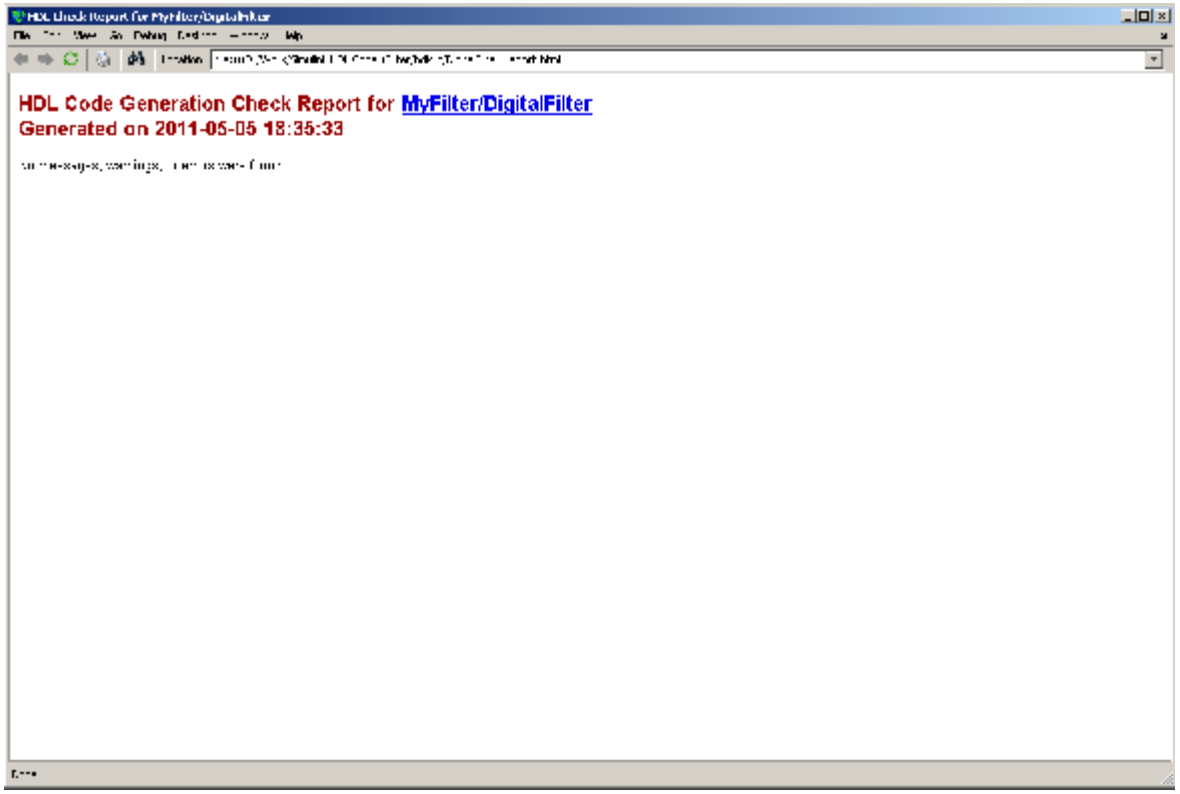

Рисунок 14 – Результаты проверки отображаются в WEB-браузере

8. Выполните генерацию VHDL кода по модели цифрового фильтра. Для этого нажмите на кнопку Generate в окне диалога Configuration Parameters.

В окне Command Window программной среды MatLab после генерации кода появится сообщение, указывающее на успешность выполнения процесса генерации и полный путь к сгенерированному файлу. Отметим, что количество сгенерированных файлов может варьироваться в зависимости от сложности модели.

Для генерации TestBench для модели цифрового фильтра необходимо выполнить следующие шаги.

9. В «дереве» выбора Select диалогового окна Configuration Parameters выберите TestBench.

10. Нажмите на кнопку Generate Test Bench

В результате запустится генерация HDL-кода для тестирующей программы TestBench. По окончании генерации в окне Command Window программной среды MatLab будет указан путь к сгенерированному HDL-файлу.

Несколько слов о control file. Этот файл позволяет производить настройку процесса генерации HDL-кода по модели. Причем, настройку можно производить как для модели в целом, так и для отдельных блоков и подсистем. Настройка выполняется с помощью команды forEach, помещаемой в control file для каждого блока, генерация кода для которого нуждается в настройке. Наиболее просто создать правильный вызов команды forEach с помощью скрипта hdlnewforeach, поставляемого в составе расширения Simulink HDL Coder. Подробности использования команды forEach в control file можно изучить по документации к Simulink HDL Coder.

Лабораторная работа №4. Создание Custom Blocks

Несмотря на то, что с Simulink поставляется богатый набор готовых блоков, которые можно использовать для моделирования цифровых устройств и совместимых с Simulink HDL Coder, в Simulink также предусмотрена возможность создания различных видов пользовательских (Custom) блоков. Функциональность таких блоков пользователь определяет сам. Из всех типов Custom блоков совместимы с HDL Coder только, так называемые, Embedded MatLab Function блоки. Функциональность таких блоков пользователь описывает, создавая программу на языке программирования Embedded MatLab.

Рассмотрим на примере создание модели с использованием Embedded MatLab Function блока. В модели реализуем счетчик, имеющий два двоичных управляющих сигнала: первый – rst, сбрасывает счетчик в нулевое значение; второй – enable, разрешает или запрещает работу счетчика.

1. Создайте новую директорию, например Counter, и укажите ее в качестве Current Directory в программе MatLab.
2. Запустите Simulink и создайте новую модель. Сохраните ее под именем uint8_counter.
3. Выберите Embedded MatLab Function блок из раздела User-Defined Functions в Simulink Library Browser и поместите его в окно модели.
4. Откройте окно Embedded MatLab Editor двойным щелчком «мыши» по изображению блока на диаграмме модели.
5. Замените текст программы на следующий:

```
function y = counter(rst,enable)
persistent counter;

if isempty(counter)
    counter=uint8(0);
end

if rst
    counter=uint8(0);
elseif enable
    counter=counter+1;
end
y=counter;
```

6. Выберите блок Display из раздела Sinks в Simulink Library Browser и поместите его в окно модели.
7. Выберите блок Signal Builder из раздела Sources в Simulink Library Browser и поместите его в окно модели. Двойным щелчком «мыши» откройте окно Signal Builder. Используя меню открывшегося окна создайте два сигнала так, чтобы редактор сигналов выглядел как на рис. 20.

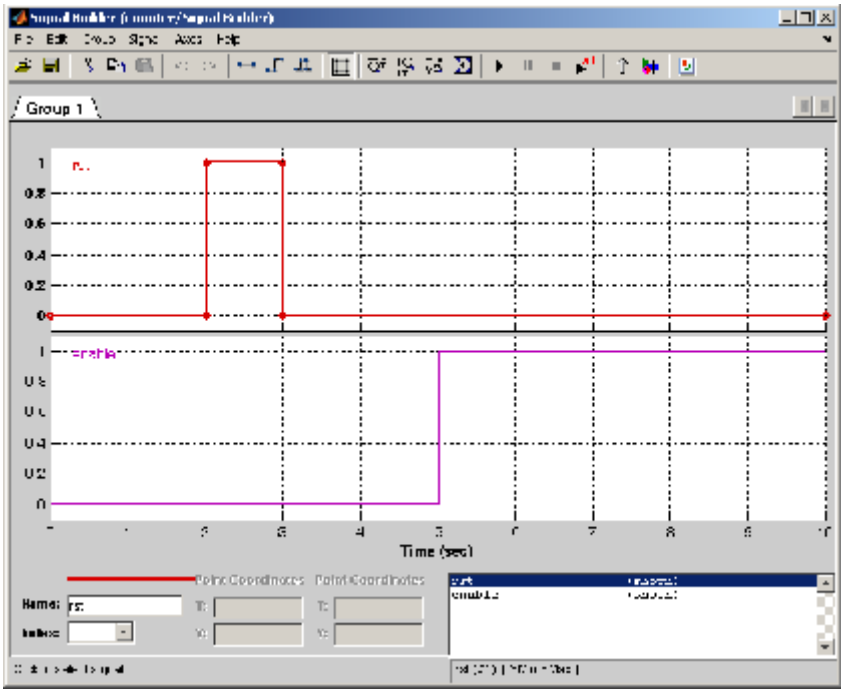

Рисунок 20 – Создайте два сигнала и откорректируйте положение импульсов на временной оси

8. Вызовите окно диалога Simulation Options, выбрав в меню File позицию Simulation Options. Из выпадающего списка Signal values after final time выберите Hold final value. В поле Sample time установите значение 1. Закройте окно диалога Signal Builder.

9. В окне Configuration Parameters модели в «дереве» Select выберите Solver. В поле Fixed step size установите значение 1.

10. Добавьте к диаграмме модели два блока Data Type Conversion из категории Signal Attributes в Simulink Library Browser. В настройках этих блоков установите в поле Output Data Type в значение Boolean.

11. Соедините блоки сигналами так, чтобы диаграмма модели выглядела как на рис. 21.

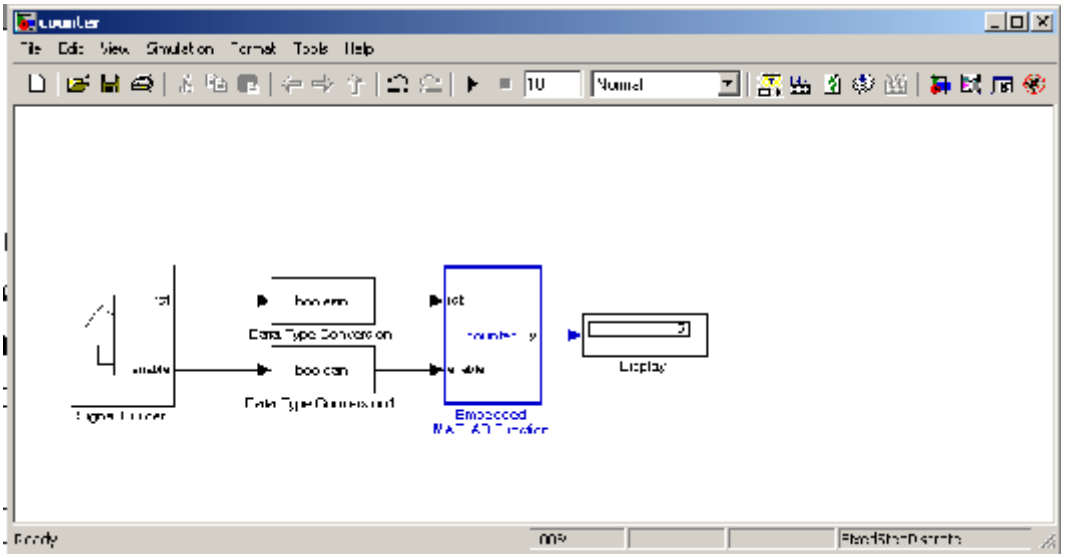

Рисунок 21 – Вид диаграммы модели, использующей Embedded MatLab function блок

12. Проверьте правильность работы модели, используя Simulink Debugger.

Разберемся с кодом, описывающим функциональность блока counter. Очевидно, что аргументы функции counter в коде программы отображаются во входные порты блока counter. Возвращаемое значение – y – является выходным портом. Отметим, что для задания нескольких выходных портов у Embedded MatLab function блока, имена портов перечисляются в квадратных скобках через запятую. Так, например, конструкция:

```
function [x,y,z] = counter(rst,enable)
```

описывает блок с тремя выходными портами с именами x, y и z.

Следующая конструкция:

```
persistent counter;
```

объявляет переменную counter как статическую переменную (persistent), значения которой сохраняются между вызовами функции. Необходимо заметить, что по умолчанию все переменные в Embedded MatLab function являются локальными и их значения не сохраняются между вызовами функции (вызов Embedded MatLab function происходит на каждом временном шаге работы модели при симуляции). Указанная выше конструкция является просто объявлением переменной. Захват памяти и инициализация переменной counter производится следующими операторами:

```
if isempty(counter)
    counter=uint8(0);
end
```

Здесь функция isempty возвращает значение TRUE, если переменная counter еще не существует. Таким образом, инициализация и захват памяти под статическую переменную производится только один раз при инициализации модели.

Еще одно важное замечание. По умолчанию, в языке MatLab все переменные считаются переменными типа double. Все остальные типы данных необходимо указывать явно при присвоении переменной значения. Остальные конструкции, используемые в функции counter, думается, не требую каких-либо пояснений.

И еще одно замечание. Вместе с Simulink HDL Coder поставляется богатая библиотека примеров Embedded MatLab function блоков. Доступ к этому набору готовых

блоков с редактируемой функциональностью можно получить, набрав в Command Window программной среды MatLab команду

eml_hdl_design_patterns

Практическое занятие №1. Инструменты разработки приложений для FPGA компании Xilinx

Введение

Эта работа познакомит вас с программными инструментами компании Xilinx - ISE™. Программные средства ISE™ представляют собой систему сквозного проектирования, которая реализует полный цикл разработки проектов на FPGA архитектуре. Система включает в себя инструменты для создания исходных описаний, синтеза, моделирования, размещения, трассировки и собственно программирования кристалла.

Цели

После ознакомления с этой работой вы сможете:

- Ориентироваться в этапах разработки проекта на FPGA-платформе.
- Выяснить особенности специализированной платы Digilent Spartan-3.
- Перечислить особенности PicoBlaze контроллера.

Методика

Эта работа включает в себя четыре этапа: вы создадите новый проект, добавите файлы дизайна в проект, промоделируете (simulate) дизайн и, наконец, скомпилируете файлы, подготовив их, таким образом, к загрузке в FPGA. Ниже, для каждого этапа, вы найдете сопроводительные инструкции и иллюстрации, которые предоставят вам всю информацию, необходимую для выполнения основного задания.

Ознакомление с платой «Spartan-3 Board»

На протяжении всех работ вы будете использовать «Spartan-3 board» для того, чтобы проверять результаты вашей работы. Вдобавок к собственно самой плате для успешного выполнения работ требуется источник питания, JTAG-кабель для программирования FPGA и кабель для подключения к последовательному порту персонального компьютера RS232. Внешний вид платы приведен на рис. 1.

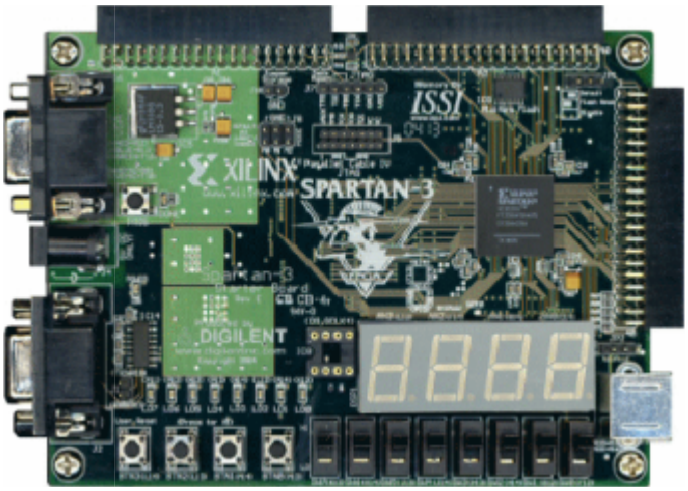

Рисунок 1.1 – Внешний вид платы «Digilent Spartan-3»

Основные особенности платы

- Содержит Spartan-3 xc3s200-4ft256 FPGA
  - 4,320 логических ячеек (480 CLBs)
  - 30 Килобит распределенной памяти (RAM)
  - 216K блочной памяти (BlockRAM)
  - 12 умножителей18x18
  - 4 контроллера управления временем (DCM)
  - 173 пользовательских контактов ввода/вывода
  - 76 дифференциальных пар ввода/вывода
- Xilinx XCF02S Platform-Flash 2 Мбита
- 1024 байт быстрой асинхронной SRAM
- 3-битный, 8-цветный VGA порт
- 9-контактный RS232 последовательный порт
- PS/2-порт
- Четыре цифровых, 7-сегментных светодиодных дисплея
- Восемь переключателей
- Восемь сигнальных светодиодов
- Четыре программируемые кнопки
- 50 Мегагерцовый кварцевый резонатор в качестве источника тактового сигнала
- режим конфигурации FPGA выбирается комбинацией перемычек
- Кнопка перезагрузки FPGA
- Светодиодный индикатор успешной конфигурации FPGA
- Три разъема (40 контактов) расширения
- JTAG порт для конфигурации FPGA
- Digilent JTAG кабель для подключения платы к параллельному порту компьютера
- Светодиодный индикатор питания
- Встроенные источники стабилизированного питания 3.3V, 2.5V, 1.2V

Общие представления о PicoBlaze

В этом параграфе представлена короткая информация о микроконтроллере PicoBlaze. В процессе изучения этого параграфа вы научитесь использовать PicoBlaze для создания нового дизайна, который будет демонстрировать некоторые особенности программных инструментов ISE Project Navigator и платы Digilent Spartan-3. Эти лабораторные работы не предполагают сколько-нибудь глубокого изучения микроконтроллера PicoBlaze. Более подробная информация о PicoBlaze может быть найдена в руководстве пользователя PicoBlaze.

PicoBlaze – свободно распространяемый!

PicoBlaze – свободно распространяемый 8-битный микроконтроллер: Самая подробная информация находится на веб-сайте PicoBlaze

http://www.xilinx.com/support/documentation/ip_documentation/ug129.pdf

Характеристики PicoBlaze:

PicoBlaze 8-битный микроконтроллер, разработанный для использования в FPGA серий Virtex™ и Spartan™ и CoolRunner™-II CPLD. Базовый вариант PicoBlaze поддерживает от 57 до 59 различных 16- или 18-битных инструкций, 16 8-битных регистров общего назначения, до 256 непосредственно и косвенно адресуемых портов, сброс и маскируемые прерывания. PicoBlaze контроллер для Spartan-3, Virtex-4, Virtex-II, и Virtex-II Pro также включают 64-байтную быструю RAM.

Как показано на рис. 2, PicoBlaze микроконтроллер обладает следующими характеристиками:

- 8-битные регистры общего назначения
- 1K-память инструкций, автоматически загружаемая во время конфигурации FPGA
- 8-битное ALU с CARRY и ZERO флагами
- 64-байтная внутренняя быстрая RAM
- 256 входных и 256 выходных портов для связи с внешними источниками сигналов
- Автоматический CALL/RETURN стек на 31-положение
- Предсказываемая производительность: всегда 2 процессорных такта на каждую выполняемую команду, рабочая частота до 200 МГц или 100 операций в секунду на Virtex-II Pro FPGA
- Быстрая обработка прерываний (в худшем случае 5 тактов)
- Ассемблер, поддержка симуляции

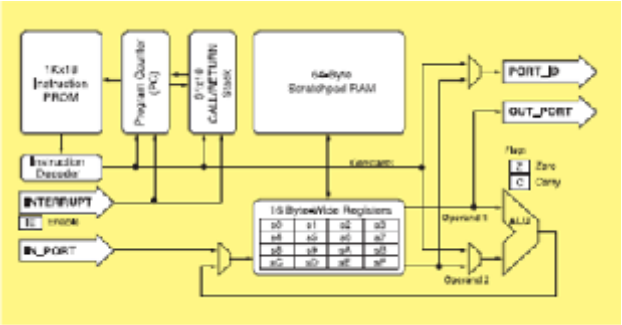

Рисунок 1.2 – PicoBlaze блок-схема

Рис. 3 иллюстрирует сигналы интерфейса верхнего уровня микроконтроллера.

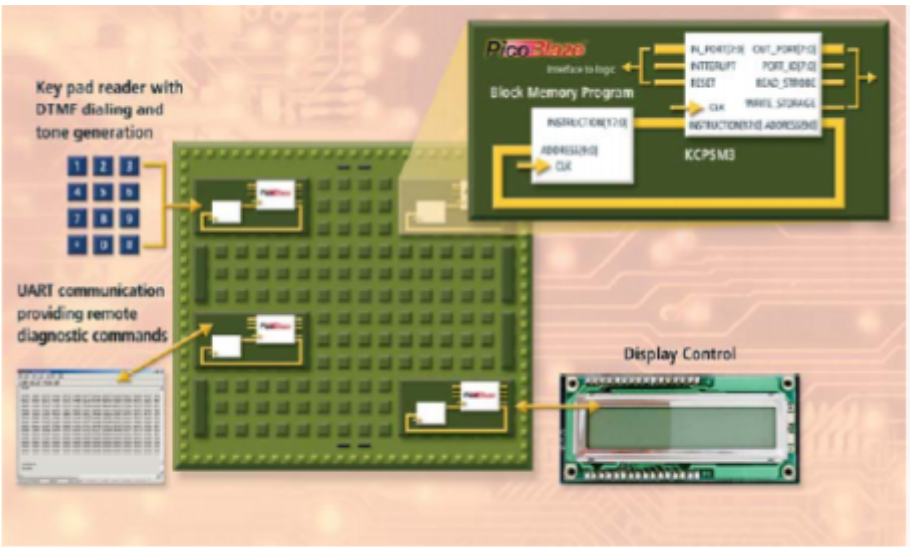

Рисунок 1.3 – Интерфейсные соединения и возможные приложения

Средства и методы разработки

В качестве средства для разработки ассемблерных программ для процессора PicoBlaze в этом цикле работ предлагается использовать любой стандартный текстовый редактор (Notepad, WordPad). Написанная программа может быть скомпилирована с помощью программы KCPSM3. KCPSM3 поставляется как простой DOS-исполняемый файл с тремя шаблонами. Сохранять написанные в текстовом редакторе ассемблерные программы для PicoBlaze нужно с расширением PSM и использовать не более 8 символов для имени файла.

Входные и выходные файлы

Ассемблер читает из 4–х входных файлов и создает 15 выходных (Рис. 4). Сначала ассемблер читает основной код для PicoBlaze , <filename>.psm, и три шаблонных файла, которые инициализируют внутреннюю память FPGA (block RAM).

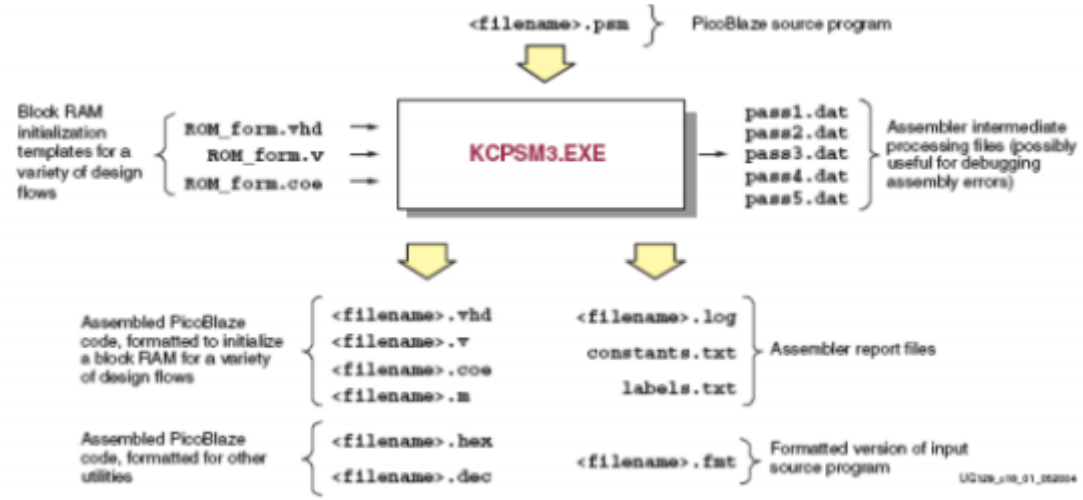

Рисунок 1.4 – Входные и выходные файлы для ассемблера KCPSM3

Если ваш ассемблерный код не содержит ошибок, KCPSM3 создаст несколько выходных файлов, которые затем можно использовать в проекте по программированию FPGA.

Давайте начнем **Шаг 1-ый**

Запустите ISE Project Navigator и создайте новый проект.

❶ Выберете **Start → Programs → Xilinx ISE 7.1i → Project Navigator**

> Могут появляться некоторые всплывающие сообщения относительно чтения сетевых папок или запуска обновлений через Интернет. Вы можете игнорировать эти сообщения и продолжить.

❷ В Project Navigator, выберете **File → New Project**

Откроется Мастер Нового Проекта (New Project Wizard) (**рисунок 1-5**)

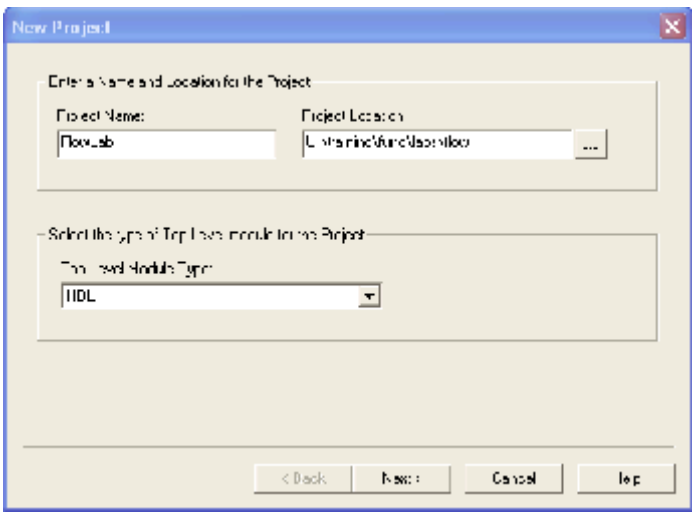

Рисунок 1.5 – Мастер создания нового проекта.

❸ В поле Project Name, напечатайте ***FlowLab*** (как пример)

❹ В поле Project Location, используйте кнопку "**…**" для выбора рабочей директории проекта, и затем нажмите <OK>. В качестве рабочей директории необходимо указать *c:\xup\fpgaflow\labs\vhdl\lab1*

❺ Нажмите **Next**

Появится следующий диалог "Device and Design Flow" (**Рис. 6**)

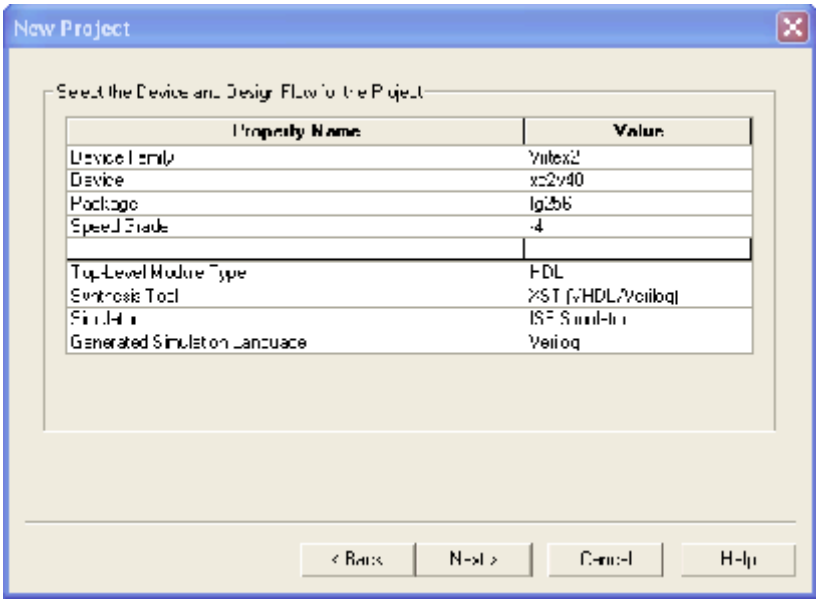

Рисунок 1.6 – Диалог установок процесса разработки

❻ Выберете следующие установки и нажмите **Next**:

Device Family: **Spartan3**
Device: **xc3s200**
Package: **ft256**
Speed Grade: **–4**
Synthesis Tool: **XST (VHDL/Verilog)**
Simulator: **ISE Simulator**
Generated Simulation Language: **VHDL** or **Verilog**

Появится диалог создания нового объекта «Create New Source» (**Рис. 7**). Вы можете использовать этот диалог для создания нового HDL файла, установив имя модуля и портов. Все необходимые файлы уже были созданы для вас в этом проекте.

Рисунок 1.7 – Диалог Создания Нового Объекта

7. Нажмите **Next**

Появится диалог «Добавить Существующий Объект» «Add Existing Sources» (**Рис. 8**).

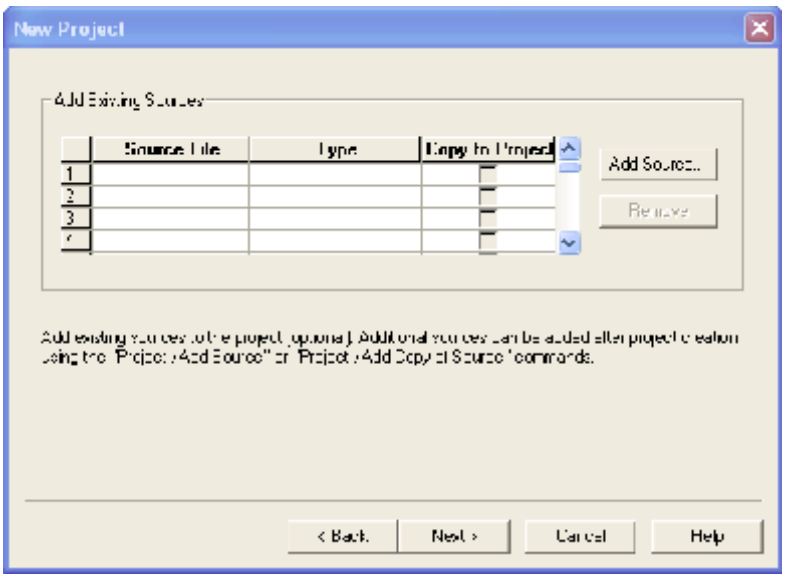

Рисунок 1.8 – Диалог добавления в проект существующего документа

Добавьте Файлы Дизайна в Проект **Шаг 2-ой**

Добавьте HDL файлы в проект.

❶ Нажмите **Add Source** и найдите местонахождения папки c:*\xup\fpgaflow\KCPSM3\VHDL*

❷ Выберете VHDL *kcpsm3_int_test* и *kcpsm3* файлы и затем нажмите **Open**.

❸ Диалог «Choose Source Type» Выбор Типа Базовых файлов откроется для каждого Verilog или VHDL файла (**Рис. 9**). Выберете **VHDL** для каждого HDL файла и нажмите **OK**.

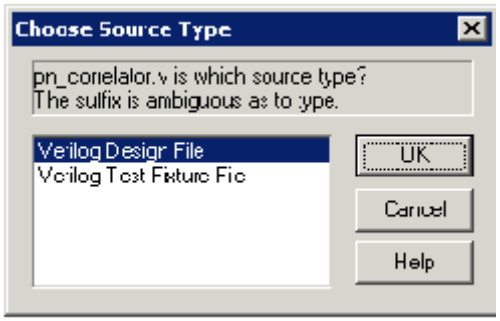

Рисунок 1.9 – Выбор типа источника

Замечание: Вы должны заметить модуль, именованный как **int_test** в иерархическом списке просмотра помеченный красным вопросом. Этот модуль – BlockRAM который будет впоследствии содержать инструкции для контроллера PicoBlaze.

Сборка Приложения **Шаг 3-ий**

Вместе с PicoBlaze контроллером поставляется пример PSM файла **init_test.psm**. Вам нужно скомпилировать его для создания инструкций ROM, которые будут интегрированы вместе с PicoBlaze контроллером.

❶ Откройте Windows Explorer и найдите место расположения директории Assembler, находящейся в поддиректории KCPSM3 (c:\xup\fpgaflow\KCPSM3\Assembler)

Замечание: KCPSM3.exe ассемблер и ROM_form* заготовка с соответствующими двумя PSM файлами должны находиться в одной и той же директории. Помните, что скомпилированные файлы будут расположены в директории, содержащей ассемблер и файлы-заготовки. Может быть, вполне разумно скопировать ассемблер и заготовки в директорию с вашим

проектом. В этом случае вы будете иметь все необходимые файлы в рабочем окружении.

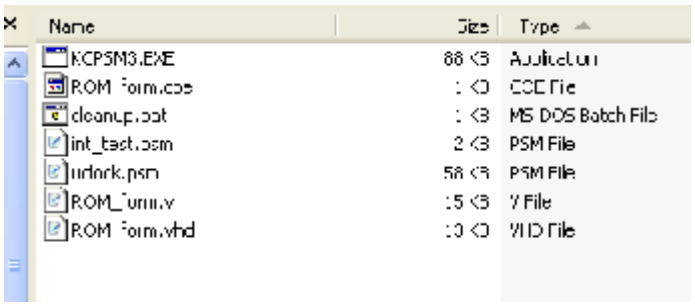

Рисунок 1.10 – Содержание Директории с Ассемблером

❷ Откройте файл **int_test.psm** и просмотрите код. Разберитесь в том, как работает и что делает программа на ассемблере. Описание набора команд и ассемблера процессора PicoBlaze содержится на прилагаемом компакт-диске.

❸ Откройте командное окно **Start → Programs → Accessories → Command Prompt**

❹ Найдите директорию с ассемблером, используя команду **cd**

> cd c:\xup\fpgaflow\KCPSM3\Assembler

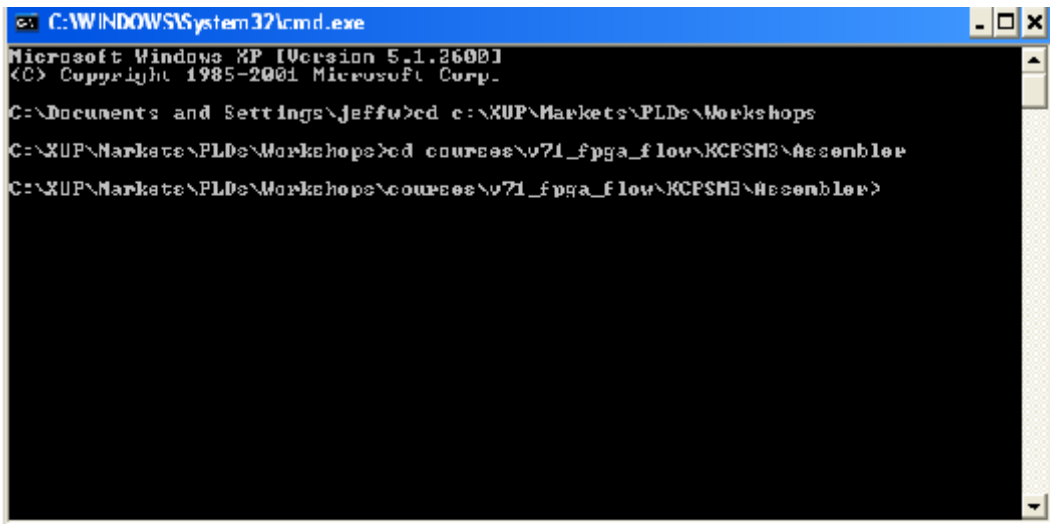

Рисунок 1.11 – Окно Команд

❺ Введите следующую команду в командной строке для получения файла для прошивки ROM

kcpsm3 int_test.psm

Замечание: Вы должны увидеть теперь несколько новых файлов в ассемблерной директории, которые начинаются init_test*, включая VHDL (int_test.vhd) и Verilog (int_test.v) файлы программирования ROM.

❻ В ISE Project Navigator, выберите команду меню **Project → Add Source** и найдите **int_test.vhd** файл в директории c:\xup\fpgaflow\KCPSM3\Assembler.

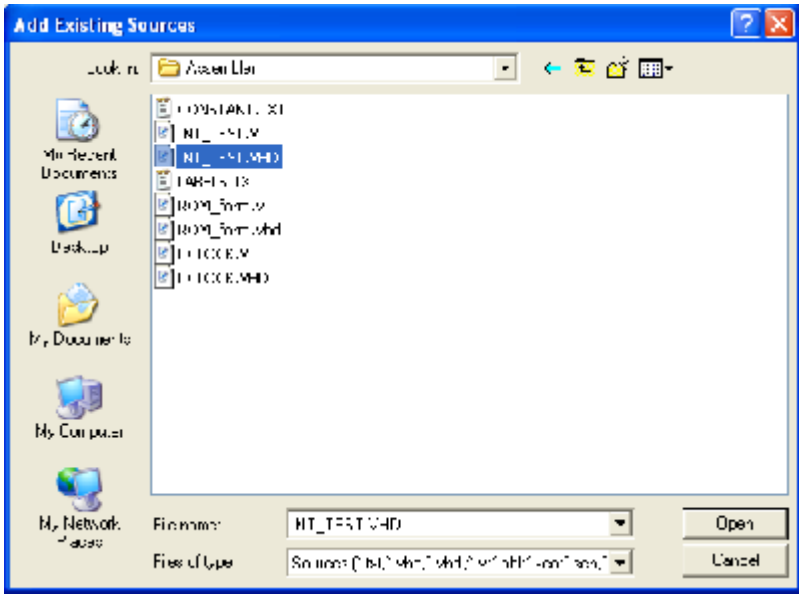

Рисунок 1.12 – Добавление int_test HDL (программный файл для ROM) в проект

❼ Нажмите **Open** и затем **OK** для того чтобы добавить **INIT_TEST** как VHDL/Verilog Design File в проект (**Рис. 13**).

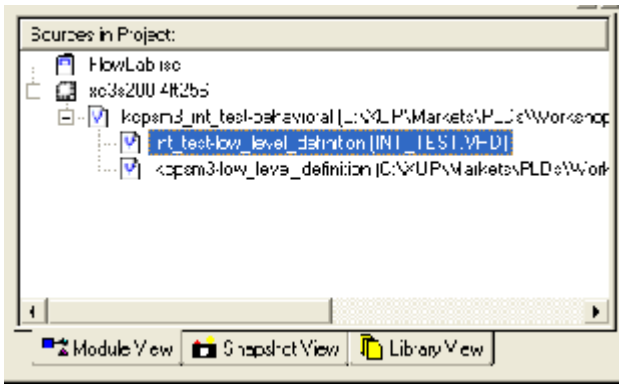

Рисунок 1.13 – Иерархия дизайна PicoBlaze контроллера

Замечание: Файл верхнего уровня “top-level kcpsm3_int_test.vhd” содержит реализацию файла для программирования ROM int_test.vhd. После добавления этого кода в проект, значок с красным вопросом исчезнет.

Добавьте программу ***testbench.vhd*** и просмотрите код. Затем запустите behavioral моделирование, используя Xilinx iSIM, и проанализируйте результаты.

❶ В Sources in Project окне выберете **Project → Add Source** и найдите c:\xup\fpgaflow\KCPSM3\VHDL

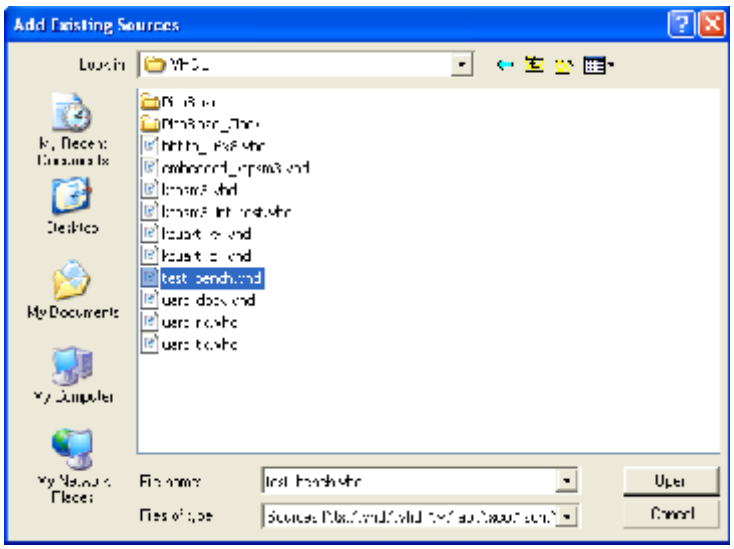

Рисунок 1.14 – Месторасположение файла испытательного стенда

❷ Выберете **test_bench.vhd** и нажмите <Open>

❸ В диалоге **Choose Source Type**, выберете **Test Bench File** и нажмите <OK> для того, чтобы добавить программу testbench в проект.

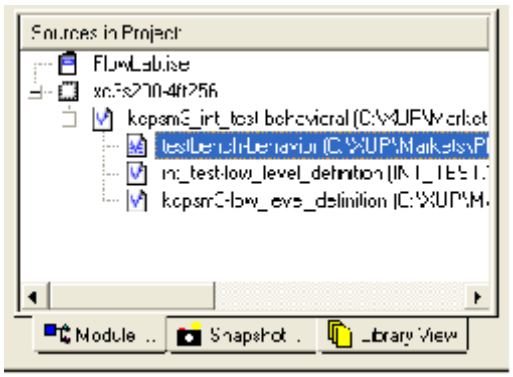

Рисунок 1.15 – Иерархия проекта с добавленным файлом Testbench

❹ Выбрав Testbench, разверните окно инструментов Xilinx ISE Simulator в поле Processes for Source window, щелкните правой кнопкой мышки на **Simulate Behavioral Model**, и выберете Properties.

❺ Введите число 10000 для **Simulation Run Time** и нажмите <OK>.

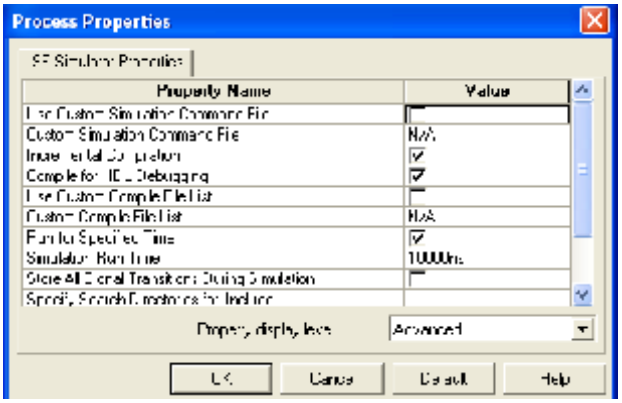

Рисунок 1.16 – Свойства iSIM Behavioral Simulation

❻ Дважды щелкните мышкой на **Simulate Behavioral Model**

После окончания моделирования в рабочей области Project Navigator появятся две закладки. На одной из них отображаются результаты моделирования в виде временной диаграммы. На второй закладке приведен текст тестирующей программы Testbench.

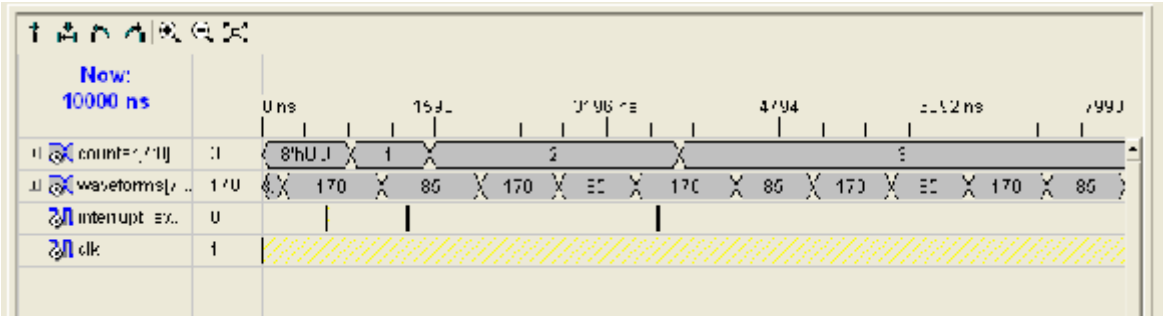

Рисунок 1.17 – iSIM HDL Simulator

❺ Выберите закладку **waveform** для того, чтобы просмотреть результаты моделирования. В этом окне можно увеличивать и перемещать просматриваемый участок временной диаграммы. Убедитесь, что результаты моделирования совпадают с ожидаемыми результатами.

❻ Закройте окно «**simulator windows**». Нажмите **Yes** для подтверждения того, что вы действительно хотите прекратить симуляцию.

Реализация дизайна — Шаг 5

В процессе реализации дизайна будут созданы несколько файлов отчета. Содержание этих отчетов будет рассмотрено более подробно в следующей лабораторной работе.

❶ Выберите самый верхний уровень иерархии дизайна (файл ***kcpsm3_int_test.vhd***) в окне Project.

❷ В окне Processes дважды щелкните на **Implement Design** (**Рис. 17**)

Учтите, что при реализации проекта будут выполняться все процессы, необходимые для имплементации. В этом случае Synthesis будет выполнен перед Implementation.

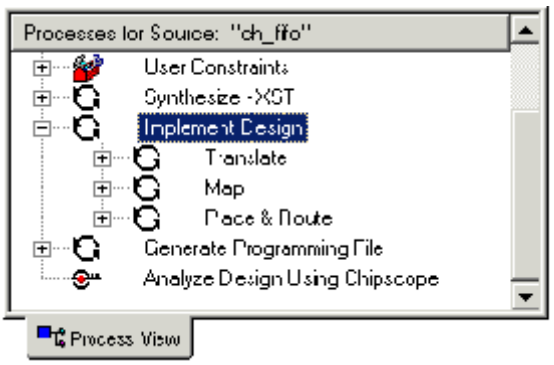

Рисунок 1.17 – Окно Processes

❸ Пока идет процесс имплементации, нажмите + на строке **Implement Design** для того чтобы увидеть составляющие процесса имплементации.

После завершения каждого из этапов имплементации, может появиться один из ниже перечисленных символов:

Зеленая галочка сигнализирует об успешном завершении
Желтый восклицательный знак – предупреждение (warning)
Красный X – ошибка.

В этом дизайне вы можете наблюдать несколько сообщений о предупреждениях, которые можно просто игнорировать.

❹ Прочитайте некоторые сообщения о предупреждениях.

❺ По окончании имплементации вы можете ознакомиться с результатами, дважды щелкнув на **View Design Summary** (**Рисунок 1-18**).

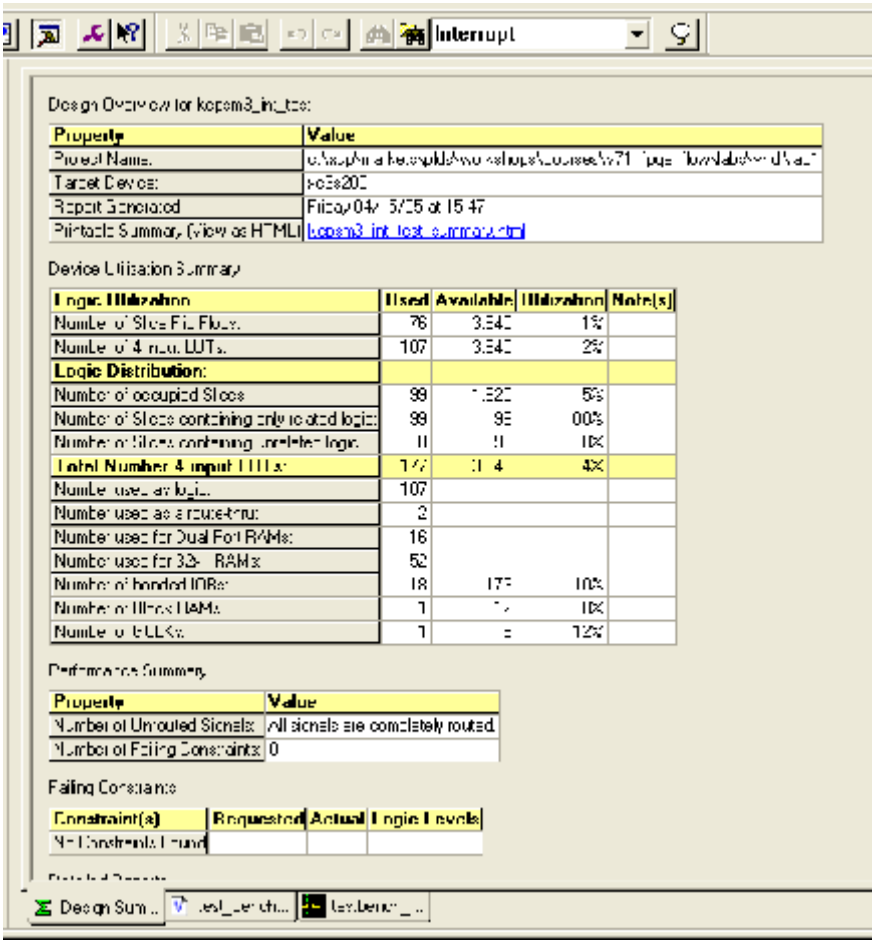

Рисунок 1.18 – Design Summary

Заключение

В этой работе вы ознакомились с основными этапами разработки приложений на FPGA с помощью средств разработки компании Xilinx ISE™ . А именно: создание проекта, добавление файлов в проект, моделирование, симуляцию и имплементацию проекта.

Одним из важнейших моментов при выполнении этой работы является знакомство с процессором PicoBlaze. Необходимо внимательно ознакомиться с текстами ассемблерных программ, очень полезно изучить VHDL-код процессора.

В следующей работе вы будете изучать некоторые детали отчетов, которые предоставляет средство разработки Xilinx ISE™. Сможете узнать, насколько удачно был имплементирован дизайн, определить достигли вы нужной производительности или нет и многое другое.

Практическое занятие №2. Architecture Wizard

Ведение

Эту работу можно рассматривать как предварительное знакомство с Architecture Wizard и редактором PACE.

Цели

После выполнения этой работы вы сможете:

- Использовать Architecture Wizard для настройки компонентов проекта. В частности DCM (Digital Clock Management) компонент
- Встроить компонент DCM в дизайн
- Использовать редактор PACE для установки связей между выводами FPGA-микросхемы и сигналами проекта
- Имплементировать дизайн и проверить правильность подключения всех выводов

Методика

Эта работа включает в себя четыре основополагающих для проектирования в ISE шага. Вы будете использовать Architecture Wizard для настройки DCM компонента, встраивать DCM компонент в VHDL код, использовать PACE редактор для назначения выводов микросхемы, имплементировать дизайн, загружать код в FPGA и, наконец, тестировать дизайн на демонстрационной плате.

Обзор

В этой лабораторной работе используется готовый проект «UART Real-Time Clock». Подробную информацию об этом проекте можно найти в документе UART_real_time_clock.pdf.

Рассматриваемый дизайн реализует таймер реального времени с подсчетом времени в часах, минутах и секундах с возможностью будильника. Необычность этого дизайна в том, что для наблюдения за временем и установки времени будильника используется последовательный порт UART. Сообщения могут передаваться в текстовом виде с помощью таких простых приложений персонального компьютера, как Hyperterminal.

В виде команд могут использоваться обычные ASCII символы. Команда выполняется после введения символа «возврат каретки» или “Enter”. Проект готов к работе, когда на экране высвечивается строка “KCPSM3>”.

Программа “uclock” распознает верхний и нижний регистры вводимых символов и конвертирует их в верхний регистр. В случае ввода неправильной команды появится сообщение “syntax error”, некорректное значение времени будет сопровождаться

сообщением “Invalid Time”. Сообщение “overflow error” появится, если команды будут передаваться быстрее, чем FPGA-приложение сможет их обработать, хотя это маловероятно при использовании hyperterminal. (т.е. буфер приемника UART переполнится).

Дизайн требует 55 МГц тактовой частоты. Поскольку на плате Spartan-3 имеется только 50 МГц осциллятор, используемый в качестве генератора синхросигнала, необходимо преобразовать частоту генератора. Такого сорта задача стояла перед многими разработчиками на FPGA, и все они решали её примерно одним путем. Эта задача достаточно проста и вы можете самостоятельно написать HDL-код, который будет выполнять преобразование частоты. Многие производители инструментария для разработчиков (Xilinx в их числе) предлагают быстрый способ разработки элементарных модулей. Далее вы научитесь, как с помощью Мастера Архитектуры быстро создать модуль управления временем DCM (Digital Clock Manager).

Использование Мастера Архитектуры для Конфигурирования DCM Шаг 1

Откройте существующий проект.

❶ Если вы уже закрыли ISE™ Project Navigator, выберите **Start → Programs → Xilinx ISE → Project Navigator**

❷ Выберите **File → Open Project** в Project Navigator

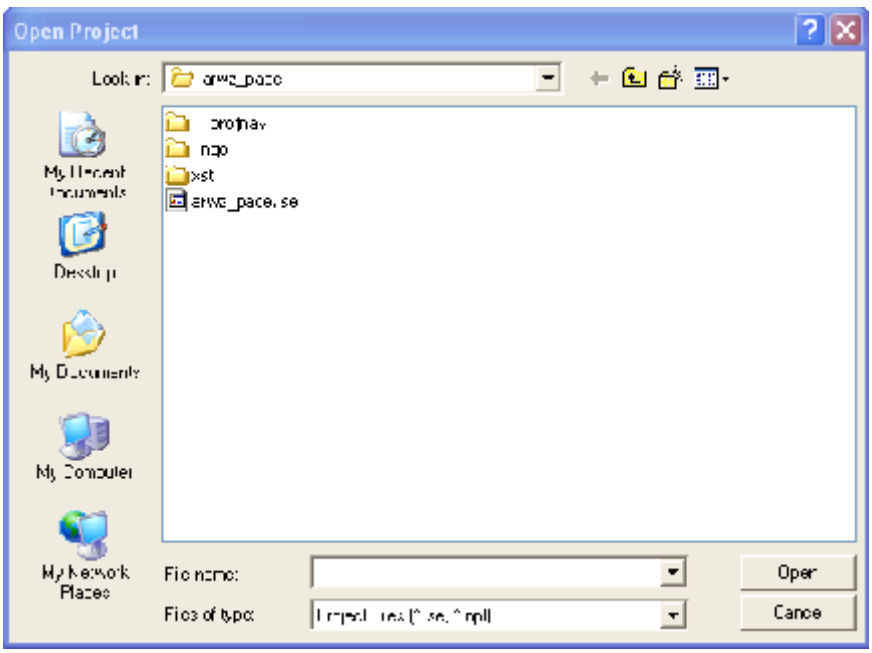

Рисунок 2.1 – Окно «Открыть проект»

❸ Найдите директорию *c:\xup\fpgaflow\labs\vhdl\lab2\arwz_pace* и выберите ***arwz_pace.ise***

❹ Нажмите **Open**

В поставляемой версии дизайна отсутствует DCM компонент. Используйте Мастер Архитектуры для конфигурирования DCM компонента под ваши требования.

❶ В окне Processes для Source window, дважды щелкните **Create New Source**

❷ В окне «New Source», выберите **IP (CoreGen & Architecture Wizard)** и введите имя файла ***my_dcm***

❸ Нажмите **Next**

❹ В окне «Select Core Type», раскройте **Clocking** и выберите **Single DCM** (**Рис. 2**)

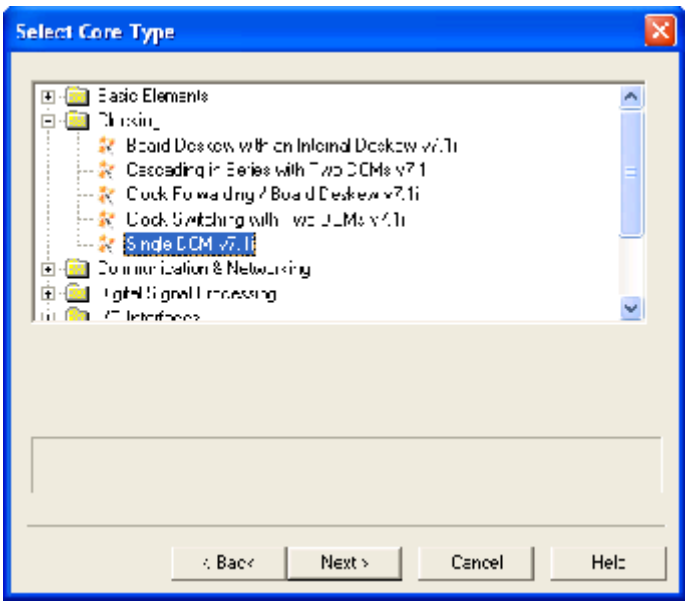

Рисунок 2.2 – Окно выбора Мастера Архитектуры

❺ Нажмите **Next,** и затем **Finish**

❺ В мастере настроек временных параметров «Clocking Wizard», в окне «General Setup», установите следующие опции (**Рис. 3**):

- CLK0, CLKFX и LOCKED параметры: **Использовать**
- RST: **Не использовать**
- Входная тактовая частота (Input Clock Frequency): **50 MHz**

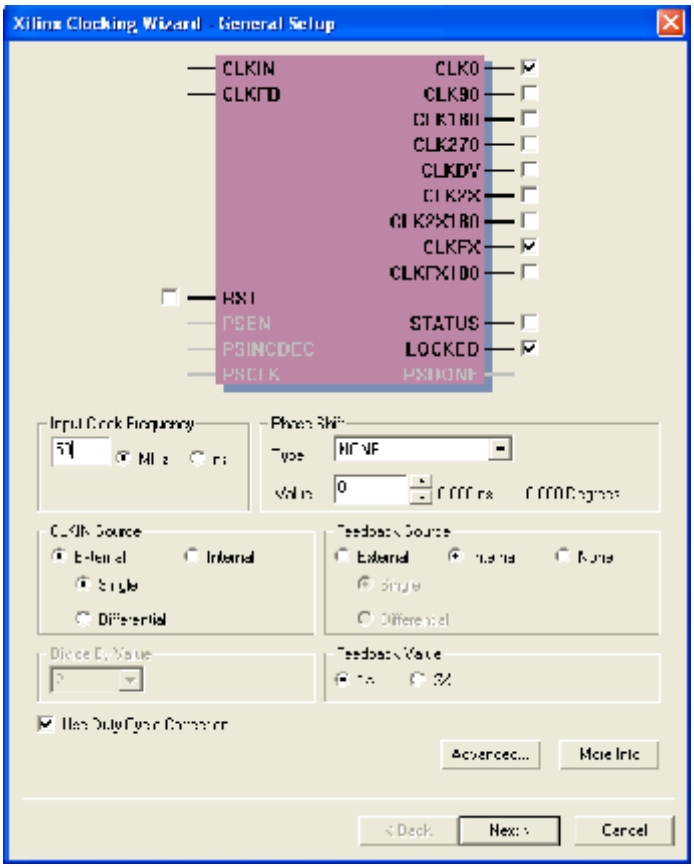

Рисунок 2.3 – Xilinx Clocking Wizard – General Setup Window

❼ Нажмите <**Next>**

❽ В окне «Xilinx Clocking Wizard – Clock Buffers» (**Рис. 4**),сохраните настройки без изменения и нажмите <**Next>**

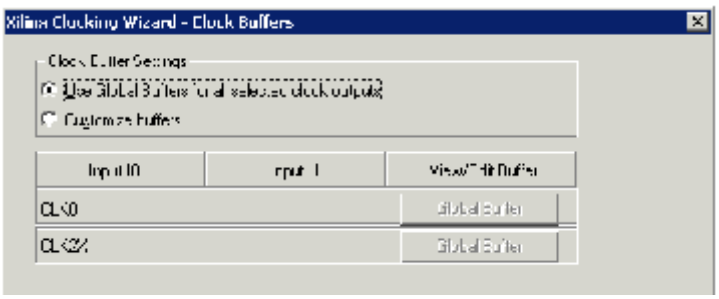

Рисунок 2.4 –Xilinx Clocking Wizard – Clock Buffers Window

❾ В диалоге «**Xilinx Clocking Wizard – Clocking Frequency Synthesizer»**, введите значение 55 МГц для выходной частоты, нажмите <Next> и затем <Finish>.

Замечание: Если вы не видите файл my_dcm.xaw в иерархии дизайна, добавьте его вручную через **Project → Add Source**.

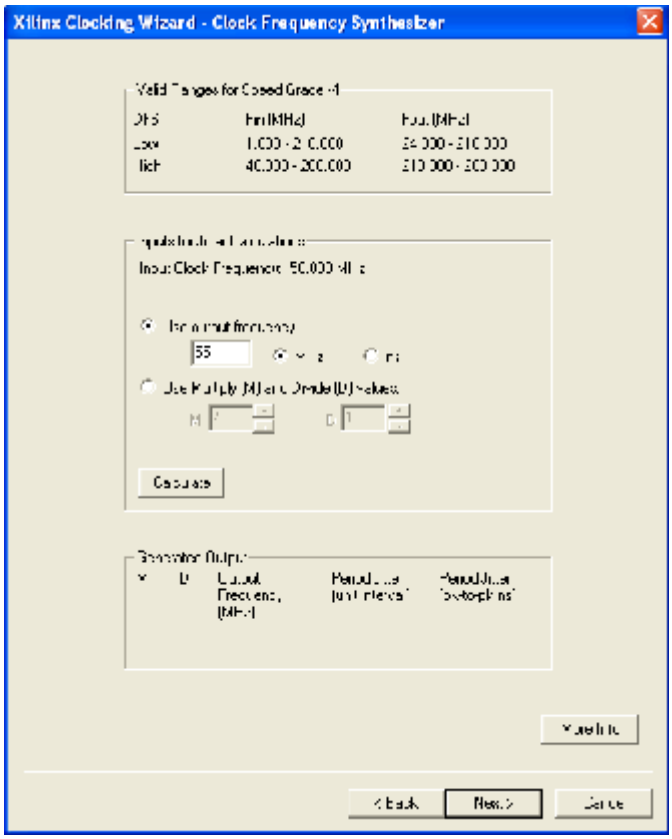

Рисунок 2.5 – Уточнение выходной частоты DCM

Убедитесь что новый файл (***my_dcm.xaw***) добавлен в окно Sources проекта (**Рис. 6**). Этот файл не будет включен в иерархию проекта до тех пор, пока он не будет инициирован в одном из HDL файлов проекта.

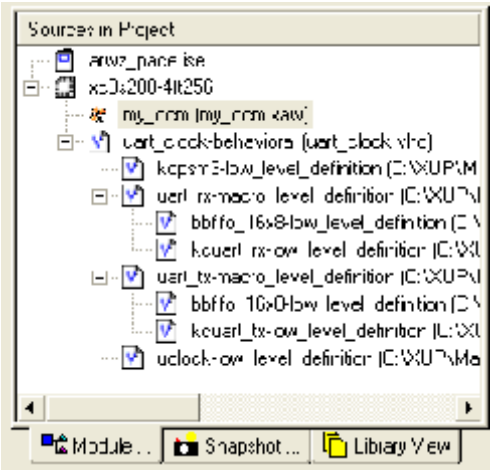

Рисунок 2.6 – DCM компонент в иерархии проекта

Теперь, когда все необходимые файлы были созданы, вы можете реализовать DCM компонент в вашем дизайне. Скопируйте и заполните текст из Заготовки Реализации (Instantiation Template) в *uart_clock.vhd* и подсоедините сигналы.

❶ С выбранным файлом *my_dcm.xaw* , следуйте в окно Processes для Source и дважды нажмите **View HDL Source** для проверки кода, сгенерированного Мастером Архитектуры

> Если файл не появляется в текстовом редакторе, ещё раз дважды нажмите **View HDL Source**

Этот файл содержит следующие компоненты для реализации: IBUFG, DCM, и два BUFG.

Входной синхросигнал *CLKIN_IN* подключен к буферу IBUFG, который подсоединен к DCM. Два выходных тактовых сигнала подключены к BUFG-компонентам.

❷ В окне Sources, в Project, дважды нажмите на ***uart_clock.vhd*** для того, чтобы открыть код в текстовом редакторе.

❸ Выберите ***my_dcm.xaw*** в окне Sources в Project

❹ В окне Processes для Source, дважды нажмите **View HDL Instantiation Template** для того, чтобы открыть Заготовку в окне текстового редактора

> Если заготовка не открылась в текстовом редакторе, попробуйте ещё раз.

❺ В Заготовке *my_dcm.vhi*, скопируйте **объявление компонента (начинается с *COMPONENT my_dcm* и заканчивается после *END COMPONENT;*)** и вставьте в ***uart_clock.vhd*** под комментарием -- *Insert DCM component declaration here*

❻ В HDL заготовке *my_dcm.vhi*, скопируйте **реализацию компонент (начинается с**

Inst_my_dcm: my_dcm* и до конца файла)** и вставьте в ***uart_clock.vhd ниже комментария -- *Insert DCM component instantiation here*

❼ Закончите реализацию заполнением соединений порта следующим образом:

```
Inst_my_dcm: my_dcm

 PORT MAP(
CLKIN_IN                => clk,
```

```
    CLKFX_OUT           => clk55MHz,
CLKIN_IBUFG_OUT => open,
CLK0_OUT            => open,
 LOCKED_OUT         => lock
 );
```

❽ Добавьте описание сигналов для выхода 55 МГц в DCM ниже комментариев -- Signals for DCM, следующим образом:

```
signal clk55MHz : std_logic;
```

Добавьте порт lock в port-секцию uart_clock следующим образом:

```
entity uart_clock is
   Port (
          tx : out std_logic;
          rx : in std_logic;
       alarm : out std_logic;
         clk : in std_logic;

        lock : out std_logic
        );
end uart_clock;
```

Замечание: этот порт (**lock)** будет зажигать светодиод led1 на плате Spartan-3. Светодиод будет сигнализировать о том, что DCM успешно захватил частоту 50 MHz от внешнего генератора.

❿ Нажмите **File → Save** для сохранения файла

Обратите внимание на то, что файл *my_dcm.xaw* теперь появился на своем месте в иерархии дизайна.

Использование PACE для Установки Местоположения Контактов Шаг 3

Закончили дизайн тем, что установили DCM.

Большинство FPGA проектов требуют назначения входных/выходных контактов ещё до того, как дизайн будет закончен. Редактор PACE поможет легко назначить контакты и проверить то, что выбранные контакты соответствуют всем правилам DRC для I/O банков.

В этой работе вы будете использовать PACE для назначения местоположения контактов для вашего дизайна.

❶ В окне Sources (Project), выберите файл верхнего уровня дизайна ***uart_clock.vhd***

❷ В окне Processes для Source, раскройте пункт меню **User Constraint** и дважды щелкните на **Assign Package Pins** для того, чтобы открыть PACE

Нажмите “yes” в ответ на предложение добавить UCF файл в проект. Дизайн должен быть синтезирован до запуска PACE.

❸ Просмотрите окно Design Object List, и изучите имеющиеся сигналы.

В колонке Loc введите адрес для каждого внешнего контакта для связи нашего дизайна с интерфейсом платы Digilent Spartan-3. Для этого, в документации к плате Digilent найдите имена контактов FPGA Spartan, соответствующие следующим условиям:

- clk : соединен с 50 MHz осциллятором *–для справки- (T9)*
- lock *:* соединен с led0 *–для справки (K12)*
- alarm : соединен с led1 *–для справки (P14)*
- rx : соединен с контактом, который принимает последовательные данные от Maxim MAX3232 *–для справки (T13)*
- tx : соединен с контактом, который передает последовательные данные в Maxim MAX3232 *–для справки (R13)*

Просмотрите ваше назначение контактов в отношении внутренней логики.

❶ В окне Device Architecture, увеличьте масштаб просматриваемого правого нижнего угла так, чтобы были видны номера выходных контактов (**Рис. 8**)

Цветные квадратики неподалеку от I/O контактов сигнализируют о том, какие из контактов находятся в том же I/O банке. Вы можете легко увидеть, что все контакты назначены в соответствующие порты.

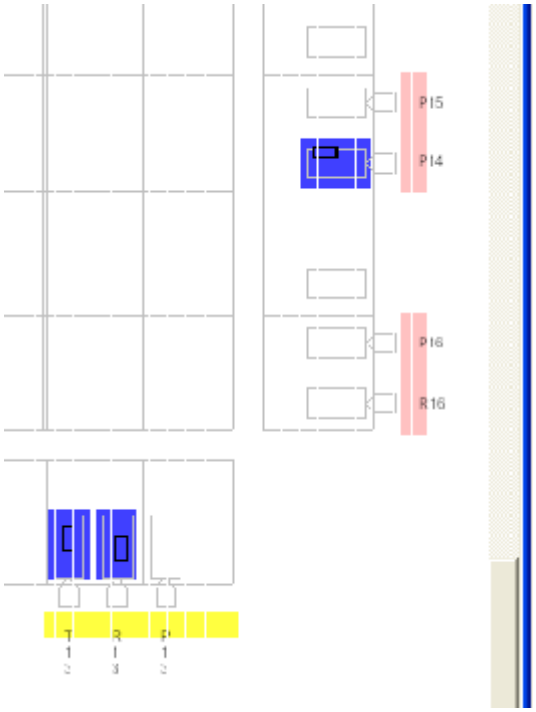

Рисунок 2.8 – Окно Архитектура Устройства

❷ Нажмите **каждый раскрашенный I/O контакт**. Соответствующие записи будут отмечаться в окне Design Object List

❸ В окне Design Object List, подтвердите, что назначение местоположения контактов было изменено.

❹ Нажмите **File → Save** для того, чтобы сохранить изменения.

❺ В диалоге Edit -> Preferences->Logic выберите **XST Default: <>** в качестве разделителя I/O Bus и нажмите **OK**

❻ Нажмите **File → Exit** для того чтобы закрыть PACE

❼ В окне Processes для Source (Project Navigator), раскройте **User Constraints** и дважды щелкните на **Edit Constraints (Text)** для того чтобы просмотреть файл ограничений *uart_clock.ucf* , созданный редактором PACE. Просмотрите UCF файл для подтверждения того, что все ограничения были записаны.

Проверка PAD отчета и начало сеанса работы с Hyperterminal Шаг 4

Создайте и проверьте PAD отчет для уверенности, что назначения контактов были реализованы. Начните сеанс работы с hyperterminal.

❶ Убедитесь, что файл верхнего уровня ***uart_clock.vhd*** выбран в окне Sources в Project

❷ В окне Processes для Source, раскройте **Implement Design** и **Place & Route**

❸ Дважды нажмите на **Pad Report**

Навигатор Проекта (Project Navigator) автоматически определит, какой из процессов должен быть запущен. Отчет будет открыт после завершения процесса Place & Route.

❹ Просмотрите отчет и подтвердите, что количество и наименование контактов соответствует вашим намерениям.

❺ Откройте сеанс работы с hyperterminal следующим образом: **Start → All Programs →Accessories → Communications → HyperTerminal**

❻ Дайте сессии имя, нажмите <OK>, и установите COM1 в качестве порта для соединения.

❼ Нажмите на кнопку Configure и установите следующие параметры для установок порта. Нажмите <OK>, когда закончите.

- Baud rate of 38400
- 8 data bits
- No parity bits
- 1 stop bit
- No flow control

Рисунок 2.9 – Установки для соединения через последовательный порт

❽ В Установках нажмите ASCII Setup вкладку и затем установите значения параметров так, как показано на Рис. 10

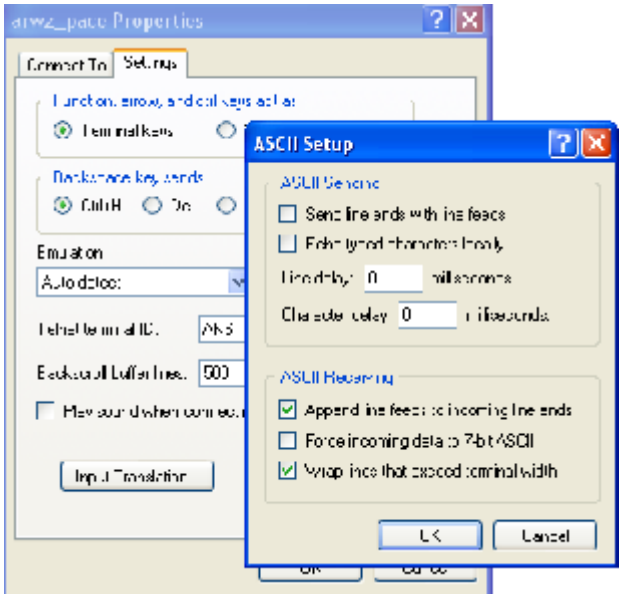

Рисунок 2.10 – ASCII Установки для соединения через последовательный порт

Загрузка Кода	**Шаг 5**

Создайте bitstream и поместите его в FPGA

❶ Выделите uart_clock.vhd и дважды щелкните на Generate Programming File для создания bitstream (битовый поток), который будет размещен в FPGA.

❷ Когда процесс генерации bitstream будет закончен, раскройте **Generate Programming File** и дважды щелкните на Configure Device (iMPACT).

❸ Выберите **Boundary-Scan Mode,** нажмите <next> и затем <finish>.

Нажмите <OK>, когда появится диалог Boundary-Scan Chain Contents Summary.

❹ Когда появится диалог «**Assign New Configuration File»**, выберите uart_clock.bit файл для устройства xc3s200 (первый в JTAG цепочке) и нажмите <open>.

Замечание: будет появляться сообщение, предупреждающее о том, что запуск часов будет осуществлен с JTAG часов. нажмите <OK>.

❺ Выберите «Bypass» для устройства xcf02s EPROM (второе устройство в цепочке) и просмотрите установки JTAG цепочки в окне iMPACT.

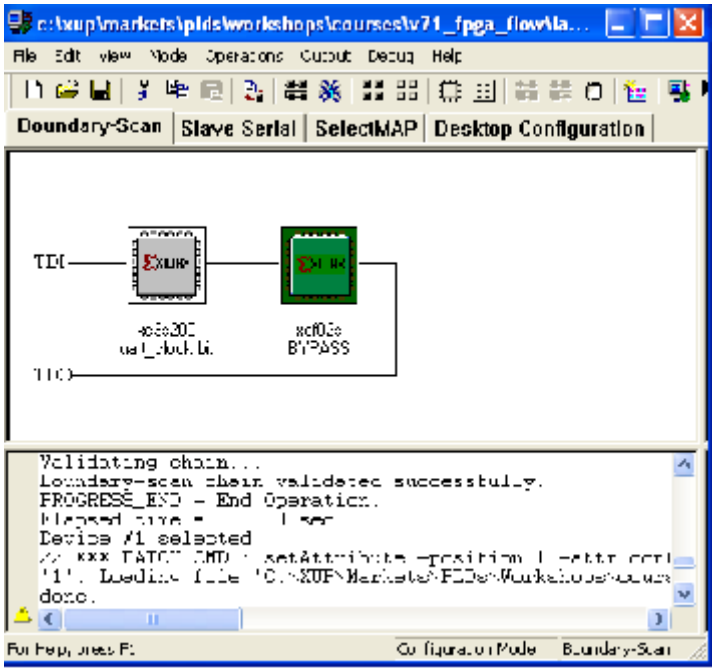

Рисунок 2.11 – JTAG-цепочка с установленным конфигурационным файлом

❻ Соедините JTAG кабель с платой Spartan-3 и включите питание на плате.

❼ Нажмите правую кнопку мышки на устройстве xc3s200 в окне iMPACT, выберите «Program», и нажмите <OK> в диалоге «**Programming Options**». Замечание: Теперь вы должны увидеть в окне гипертерминала приглашение «KCPSM3>».

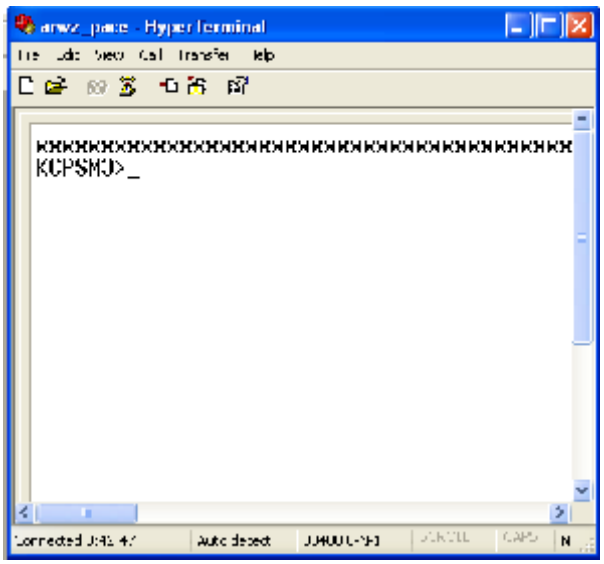

Рисунок 2.12 – Последовательная связь с PicoBlaze

Вы познакомитесь с командами управления «UART real-time clock», которые описаны в документе «UART_real_time_clock.pdf».

❶ Введите команду “time” в командную строку, для того чтобы узнать текущее время. Напоминаем, что индикация времени осуществляется в формате hh:mm:ss.

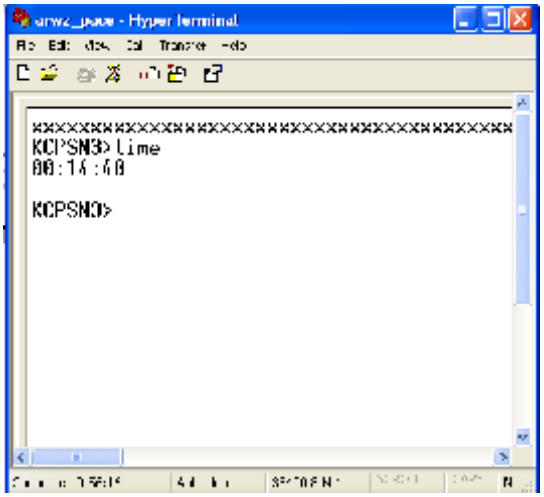

Рисунок 2.13 – Текущее Время

❷ Введите команду “alarm” в командную строку, для того чтобы определить установленное время для будильника. Формат времени hh:mm:ss

Замечание: будильник пока не активирован

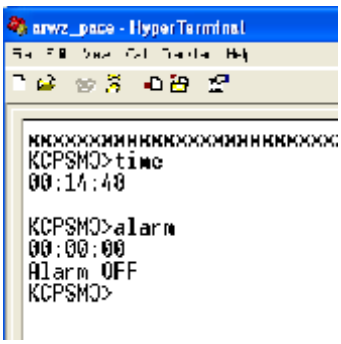

Рисунок 2.14 – Время и Статус будильника

❸ Введите команду “alarm on” для активации будильника

❹ Введите команду “alarm 00:00:30” для установки времени будильника на 30 секунд.

❺ Введите команду “time 00:00:00” для установки текущего времени

Замечание: Вы должны заметить, что светодиод led0 на плате Digilent Spartan-3 загорится через 30 секунд.

❻ Введите команду “alarm off” для выключения будильника.

Вывод

В этой работе вы использовали Мастер Архитектуры при создании и конфигурировании DCM компонента. Вы также реализовали компонент в вашем дизайне. Вы использовали PACE-редактор для назначения выходных контактов микросхемы. Наконец вы загрузили готовый для выполнения на FPGA код и провели тесты непосредственно в микросхеме.

Практическое занятие №3. Временные ограничения Timing constraints

Введение

В этой работе вы будете использовать Timing Constraints (временные ограничения) для увеличения рабочей частоты готового проекта. Вы также будете использовать Post-Map Static Timing Report и Post-Place & Route Static Timing Report для анализа производительности модифицированного дизайна.

Цели

По завершению этой работы вы сможете:

- Использовать редактор Xilinx Constraints Editor для установки временных ограничений
- Разобраться со структурой отчета Post-Map Static Timing Report
- Использовать Post-Place и Route Static Timing Report для определения наибольшей задержки сигналов при их прохождении по FPGA-микросхеме

Ссылки

Документация, ссылки на которую будут сделаны в этой лабораторной, могут быть найдены на сайте Xilinx http://www.xilinx.com.

- PicoBlaze User Guide
- Spartan-3 Data Sheet
- Digilent Spartan-3 Board Data Sheet
- Platform Flash In-System Programmable Configuration PROMs data sheet

Описание Дизайна

В этой работе вы реализуете встроенную процессорную систему с несколькими периферийными устройствами. Большинство систем уже представлено вам в предыдущих работах, тем не менее, полезно ознакомиться более подробно с описанием системы для лучшего понимания.

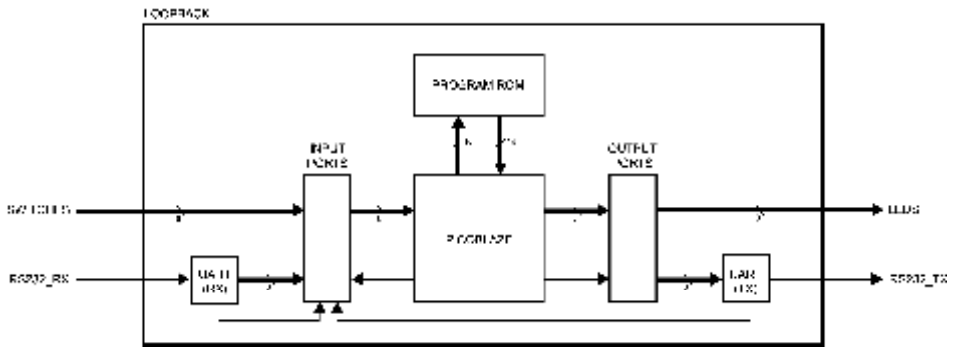

Рисунок 3.1 – PicoBlaze System

Главная задача этой работы, разработать программу на PicoBlaze-ассемблере для реализации кольцевого теста. Кольцевой тест – тест, в котором сигнал посылается в устройство и возвращается обратно через него же. Может использоваться как способ проверки того, что устройство работает правильно.

Первый «loopback» тест будет отзываться включением светодиодов на плате. То есть вы будете передавать сигнал пальцами, а принимать глазами. Второй тест будет отвечать данными через последовательный порт RS232. В этом случае настольный компьютер будет посылать, и принимать посланные данные через последовательный порт.

На рис. 1 схематично описывается рассматриваемая система. Вы можете видеть, что система имеет 8 переключателей на входе и приемник последовательного сигнала. Кроме этого, необходимо помнить о сигналах Reset и Clock, которые обязаны быть в каждом дизайне.

clk	сигнал тактовой частоты clock, 50 МГц от осциллятора
rst	сигнал сброса reset
rs232_rx	входной последовательный сигнал приемника
switches[7:0]	входной сигнал от 8-битного переключателя

Также на рис. 1 представлены выходные сигналы. Это 8 светодиодов (LED) и последовательный передатчик.

rs232_tx	выходной последовательный сигнал
leds[7:0]	8 светодиодов

Вы должны реализовать данную систему с помощью поставляемых исходных кодов, а затем разработать собственную программу для процессора PicoBlase. Разработка программы разделится на три части: Сначала вам будет нужно передать короткое сообщение после сброса. Затем вы должны реализовать две loopback функции:

- Отклик светодиодов на изменение положения переключателей
- Отклик системы на данные, поступающие на последовательный вход интерфейса RS232

После успешного завершения этой работы, вы достигнете понимания того, как нужно использовать PicoBlaze для реализации простых процессорных систем.

Анатомия отчета

Отчет Timing report дает возможность увидеть цепи, ваши ограничения Timing constraints не дали ожидаемого результата, и почему это произошло.

Графический интерфейс утилиты Timing Analyzer содержит три окна (**Рис. 2**). Вы можете использовать Hierarchical Browser, для быстрой навигации по всему большому отчету.

Окно Path Detail в правом нижнем углу содержит текст отчета.

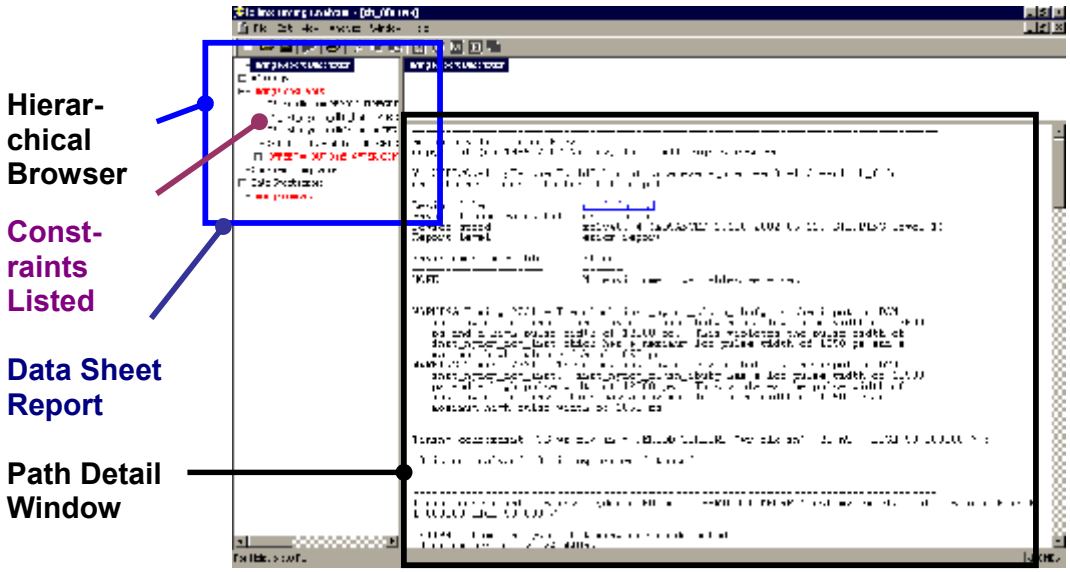

Рисунок 3.2 – Графический Интерфейс пользователя утилиты Timing Analyzer

Подробный Анализ (Detailed Path Analysis) (**Рис. 3**) содержит информацию о задержках сигналов для каждой цепи внутри микросхемы FPGA, включая следующее:

- Slack (провисание)— разница между значением Constraint и реальной временной задержкой сигнала в данной цепи (отрицательное значение slack говорит о том, что при трассировке цепи не удалось достигнуть желаемого результата)
- Источник и приемник сигнала в той или иной цепи
- Задержки сигнала на каждом отрезке цепи
- Коэффициент разветвления (Fanout) каждой цепи в зависимости от задержки
- Общая задержка в каждой цепи
- Процентное соотношение между логикой, выполняющей алгоритм обработки сигнала и ресурсами, потраченными на межсоединения. Это соотношение может быть индикатором того, что та или иная цепь проложена внутри FPGA неудачно и нуждается в коррекции.

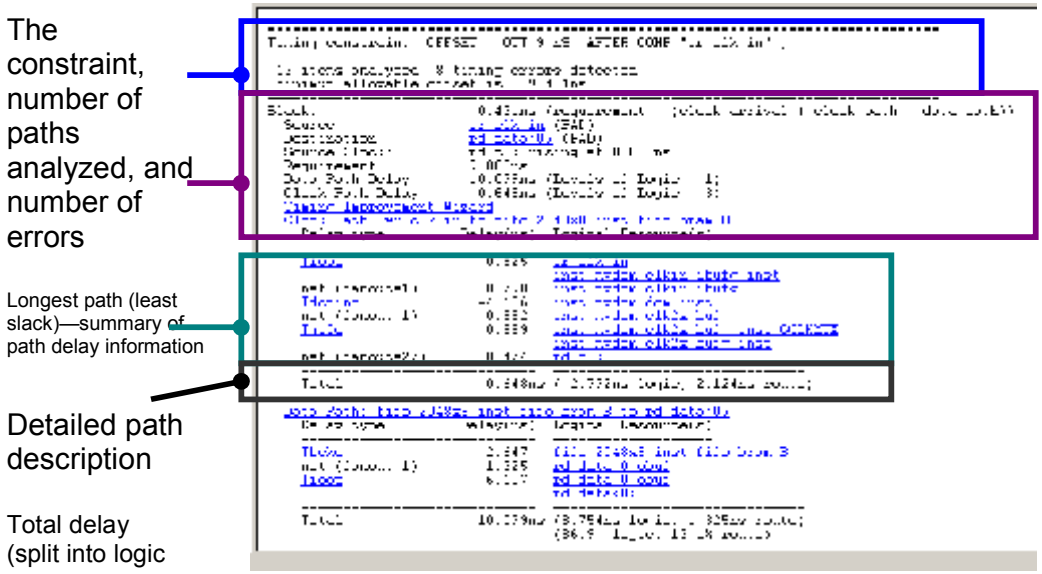

Рисунок 3.3 – Подробный Анализ

Методика

В этой работе вы создадите простую встроенную систему и установите для неё общие временные ограничения (Global Timing Constraints).

Заготовка для Ассемблера **Шаг 1**

Запустите ISE Project Navigator и откройте проект *time_const* . Скомпилируйте файл-заготовку program.psm, для генерации файла program.vhd, который будет содержать инструкции для процессора PicoBlaze.

❶ Выберите **Start → Programs → Xilinx ISE 7.1i → Project Navigator**

❷ Выберите **File → Open Project** в Project Navigator

❸ Найдите директорию c:\xup\fpgaflow\labs\VHDL\lab3

❹ Выберите ***time_const.ise***

❺ Щелкните **Open** и просмотрите верхний уровень дизайна.

❻ Откройте приглашение на ввод команды **Start → Programs → Accessories → Command Prompt**

❼ Войдите в директорию с Ассемблером, которая содержит заготовки программ (program.psm):

> cd c:\xup\fpgaflow\labs\vhdl\lab3\assembler

❽ Введите следующую команду для компиляции программной заготовки и генерации файла ROM с программой для PicoBlase:

> kcpsm3 program

Замечание: Эта заготовка синтаксически правильная, но функционально бесполезная. На следующих этапах работы Вы придадите этой программе функциональность.

❾ В ISE, добавьте сгенерированный файл ROM HDL в проект.

❿ Проведите проверку синтаксиса. Установите маркер на верхнем уровне дизайна и дважды щелкните на **Check Syntax** в пункте Synthesis в окне процессов.

Установка ограничений для Глобального Расчета Времени Шаг 2

Запустите Constraints Editor.

❶ В окне Sources , выберите файл верхнего уровня дизайна ***loopback.vhd/.v***

❷ В окне Processes для Source, раскройте пункт **User Constraints** и дважды щелкните на **Create Timing Constraints** (**Рис. 4**)

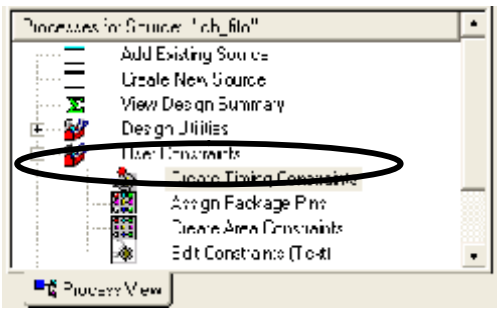

Рисунок 4 – Окно Процессов

Проект в настоящее время не имеет UCF файлов, связанных с ним. Project Navigator предложит создать его автоматически.

❸ Нажмите **Yes для создания нового** UCF файла. Назовите его *loopback.ucf* и добавьте в проект.

Когда Constraints Editor открывается, закладка Global выбирается по умолчанию (**Рис. 5**).

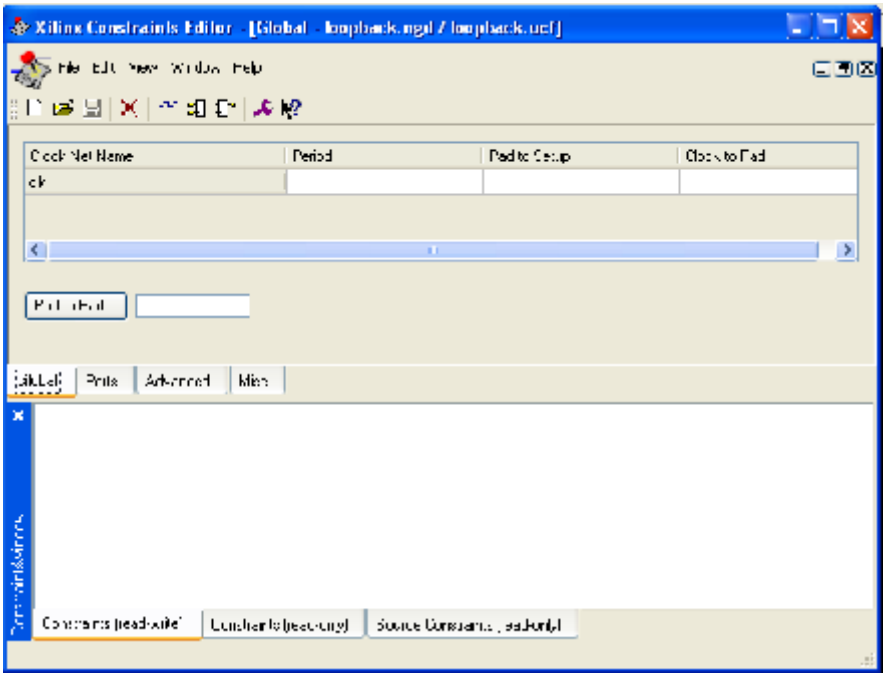

Рисунок 3.5 – Закладка Global

Введите ограничение PERIOD в *20 ns* для *clk*.

1. В колонке Period**,** дважды щелкните на поле, соответствующее сигналу ***clk*** . Откроется диалог Clock Period, с помощью которого можно ввести ограничения на PERIOD (**Рис. 6**).

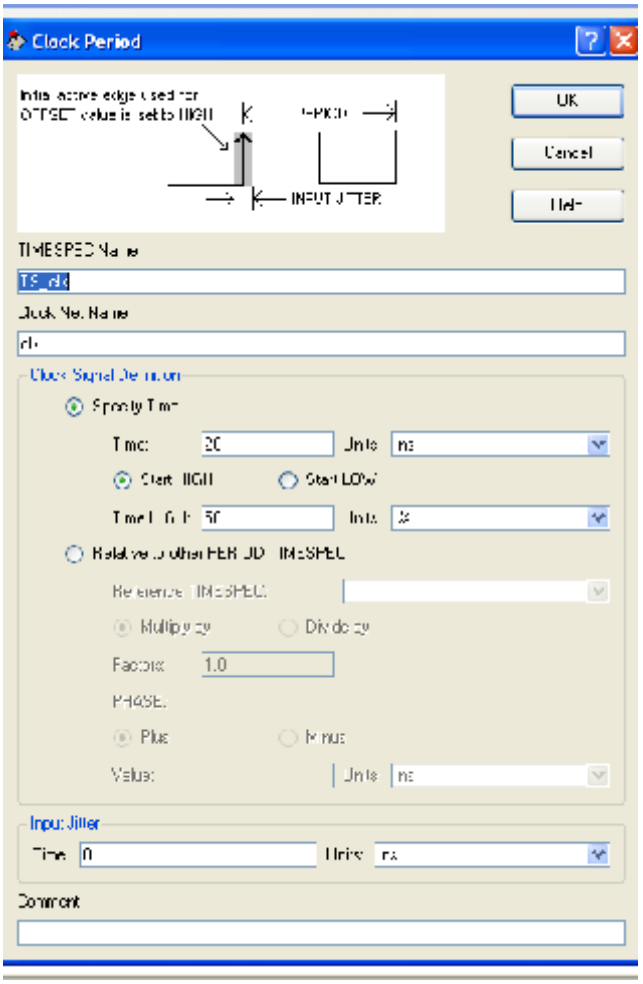

Рисунок 3.6 – Диалог установки ограничений

❷ Введите в позицию Specify Time значение **20 ns**

❸ Нажмите **OK**

Введите ограничения в 6 наносекунд на OFFSET IN и 7,5 наносекунд для OFFSET OUT для сигнала *clk*. Затем сохраните ограничения и выйдите из редактора.

❶ Дважды щелкните на пустое поле под заголовком колонки **Pad to Setup,** введите значение **6** для OFFSET и нажмите **“Ok”**. Таким образом, вы ввели ограничение на OFFSET IN (**Рисунок 3-7**)

❷ Дважды щелкните на пустое поле под заголовком колонки **Clock to Pad,** введите значение **7,5** для OFFSET и нажмите **“Ok”.** Таким образом, вы ввели ограничение на OFFSET OUT.

❸ Выберите **File → Save**

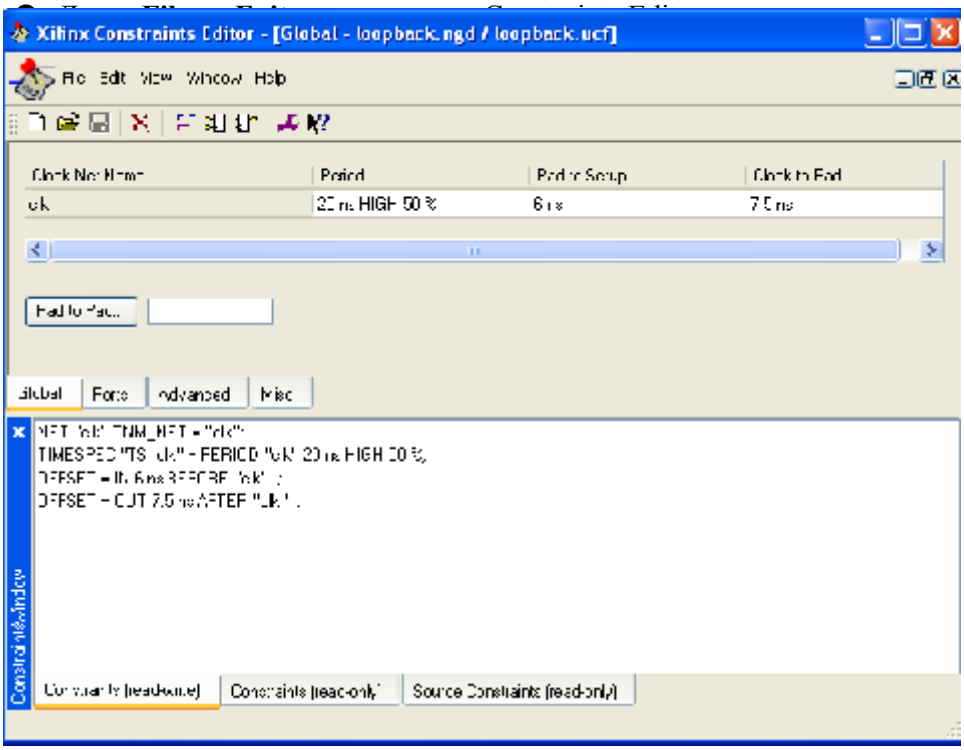

Рисунок 3.7 – Закладка Global в редакторе Constraints Editor

Ввод ограничений на местоположения контактов **Шаг 3**

Для большинства FPGA-дизайнов требования по размещению входных/выходных выводов должны быть реализованы на этапе разработки. В этой работе вы будете назначать местоположение контактов (Табл. 1) вручную редактируя файл UCF и пользуясь технической документацией на плату Digilent Spartan-3.

Таблица 3.1 – Список Вх/Вых контактов

Наименование контакта	**Направление**	**Размерность**	**Назначение на Spartan-3 Board**
Clk	Input	1 bit	50 MHz clock oscillator
Rst	Input	1 bit	Use one of four push buttons
Switches	Input	8 bits	8 Slide switches
Rs_232_rx	Input	1 bit	RS-232 serial receive
Leds	Output	8 bits	8 LEDs
Rs_232_tx	Output	1 bit	RS-232 serial transmit

❶ Установите маркер на верхний уровень дизайна и раскройте **User Constraints.** Дважды щелкните на **Edit Constraints (Text)**.

❷ Откройте «Spartan-3 Starter Kit User Guide», и назначьте местоположение контактов перечисленных в Таблице 3-1.

Для шины данных может быть применен следующий формат записи:
NET “led<0>” LOC = P12;
NET “led<1>” LOC = P13;

❸ Сохраните UCF файл после завершения.

Реализация Дизайна и Анализ временных задержек Шаг 4

Реализуйте дизайн (Design Implementation). Просмотрите отчет «Post-Map Static Timing Report» и «Post-Place & Route Static Timing Report» и заполните таблицы Chart 1 и Chart 2 на основе данных полученных из отчетов.

❶ В окне Processes для Source, раскройте процесс **Implement Design**, и далее раскройте процесс **Map**

Если вы не видите процесс Implement Design, убедитесь, что в окне Sources выбран файл ***loopback.vhd*** .

❷ Раскройте процесс **Generate Post-Map Static Timing**

❸ Нажмите правую кнопку мышки на **Post-Map Static Timing Report** и выберите **Rerun All**

При выполнении шагов реализации дизайна будут генерироваться отчеты Post-Map Static Timing Report. После этого они должны открыться в Timing Analyzer. Используйте эти отчеты для того, чтобы убедиться в том, что ограничения, наложенные вами, реалистичные и могут быть приняты этапом Place & Route.

Во время заполнения таблиц, определите ограничения на PERIOD, которые были установлены на сигнал clk.

Если отчет не содержит временных ограничений, значит Project Navigator не вернулся на операцию Translate. Повторите Шаг 3, для уверенности в том, что вы нажали правую кнопку мышки на **Post-Map Static Timing Report** и выбрали **Rerun All**.

1. Заполните строку Post-Map в этой таблице:

Chart 1	PERIOD constraint	OFFSET IN constraint	OFFSET OUT constraint
Constraint	20 ns	6 ns	7.5 ns
Post-Map			

Сравните ваш ответ с правильными ответами в конце этих указаний к работе.

❹ Покиньте **Timing Analyzer**

❺ В окне Processes для Source, раскройте процессы **Place & Route** и **Generate Post-Place & Route Static Timing**

❻ Дважды нажмите на **Post-Place & Route Static Timing Report**

2. Заполните строку Post-P&R в этой таблице:

Chart 2	PERIOD constraint	OFFSET IN constraint	OFFSET OUT constraint
Constraint	20 ns	6 ns	7.5 ns
Post-P&R			

Сравните ваш ответ с правильными ответами в конце этих указаний к работе.

Создание программы и Выполнение HDL Симуляции Шаг 5

Теперь, когда hardware-часть вашего проекта готова, вы разработаете ассемблерную программу для PicoBlaze, чтобы выполнить три задания. Заготовка программы содержит несколько определений констант и структурирована так, чтобы эти задания можно было выполнить независимо друг от друга. Первая задача наиболее легкая, третья наиболее сложная.

После того, как вы напишете код программы, вы должны скомпилировать программу на языке ассемблер заново.

```
;=================================================================
; Actual assembly program goes here...
;=================================================================

cold_start:     LOAD     s0, all_clear     ; zero out reg s0
```

```
        ; Задача #2
                ; Напишите код для вывода короткого сообщения(10
                ; символов или меньше) на последовательный порт.

loop:       LOAD    s0, all_clear    ; zero out reg s0 (nop)

        ; Задача #1
        ; Напишите код для чтения состояния переключателей и
                ; последующей записи их в порт светодиодов

rs232_echo:    LOAD    s0, all_clear    ; zero out reg s0 (nop)



        ; Задача #3
        ; Напишите код для проверки получения байта от приемника
                ; последовательного порта. Если это случилось, отправьте
                ;его обратно в передающий порт. Если ничего не
                ; случилось, то ничего не делайте

        JUMP    loop            ; loop again

;=====================================================================
;
;=====================================================================
```

Вы создаете код, необходимый для выполнения loopback теста в задаче #1 и запускаете его на PicoBlase процессоре. Как только ROM файл будет сгенерирован, вы добавите testbench в дизайн и выполните симуляцию поведения переключателей и светодиодов.

❶ Напишите код для чтения состояния переключателей и последующей записи их в порт светодиодов

Замечание: Ссылайтесь на константы, объявленные в заготовке для значения портов.
Подсказка: Вам надо написать только две строчки кода

❷ Как только код написан, скомпилируйте его заново

❸ В ISE, добавьте файл testbench в дизайн **Project → Add Source**. Выберите testbench.vhd из директории лабораторной №3.

❹ Выберите VHDL testbench и нажмите OK

❺ Установите время окончания симуляции 5000 ns и запустите симуляцию.

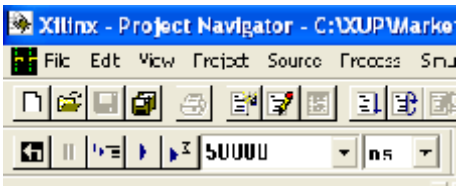

Рисунок 3.8 – Установка времени окончания симуляции

❻ Проанализируете результаты симуляции и закройте окно, как только закончите.

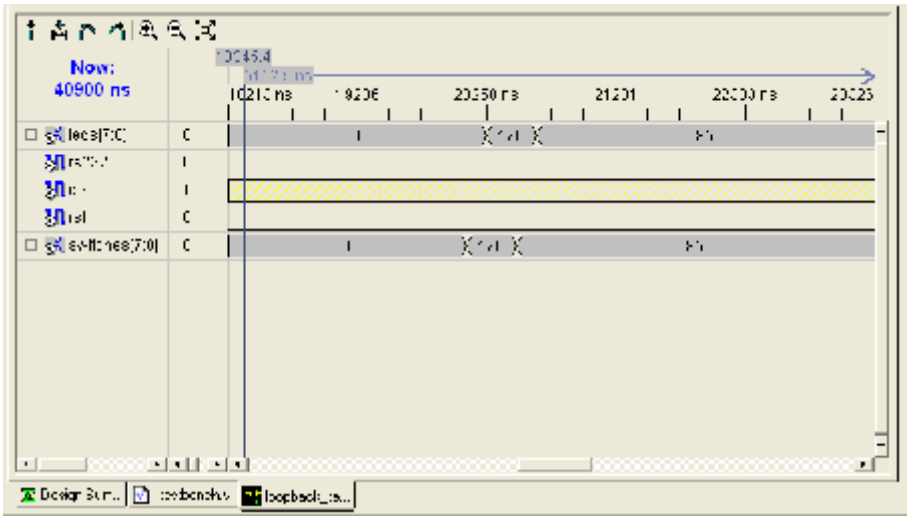

Рисунок 3.9 – Результаты Behavioral Simulation

Замечание: В примере с симуляцией бинарное значение 10101010 (170 десятичное) было установлено для входного порта PicoBlase. На выходе мы получили то же самое значение (рис. 9).

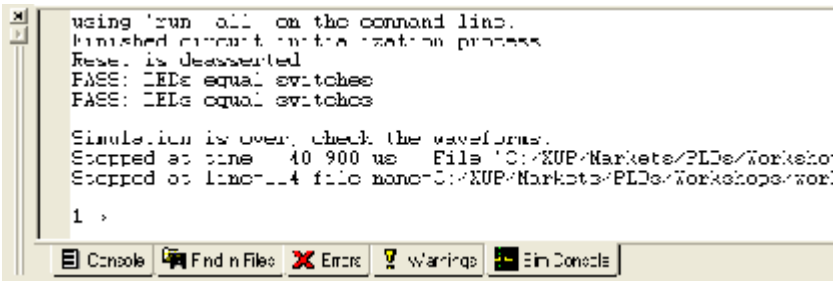

Рисунок 3.10 – Просмотр сообщений в Консоли Симулятора

На плате Digilent Spartan-3 установлена перепрограммируемая flash-память, которая позволяет хранить большие конфигурационные файлы для программирования Xilinx FPGA. Плата Digilent Spartan-3 содержит 2 Mbit xcf02s flash-память (для программирования FPGA требуется всего 1,047,616 конфигурационных бит). В этом разделе вы будете использовать утилиту iMPACT для генерации форматированного Intel MCS файла для записи во flash-память.

❶ Дважды нажмите на **Generate Programming File** для генерации bitstream.

❷ Раскройте процесс Generate Programming File и дважды нажмите на **Generate PROM, ACE, or JTAG**

❸ Выберите PROM файл и затем нажмите Next для продолжения.

❹ Установите формат файла MCS для Xilinx PROM (рисунок 3-11), имя файла и нажмите Next для продолжения.

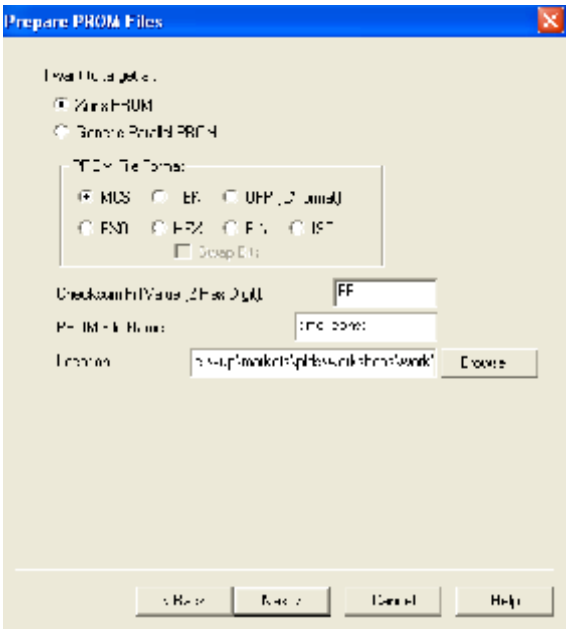

Рисунок 3.11 – Подготовка PROM файла

❺ Выберите **xcf02s** Platform Flash PROM из выпадающего списка, нажмите кнопку **Add** и затем нажмите <Next> для продолжения

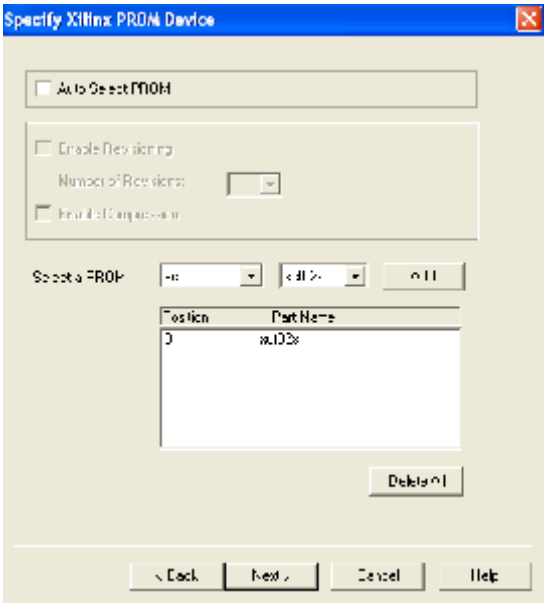

Рисунок 3.12 – Установка xcf02s PROM для платы Digilent Spartan-3

❻ Нажмите <next>, нажмите на Add File, и затем выберите loopback.bit файл

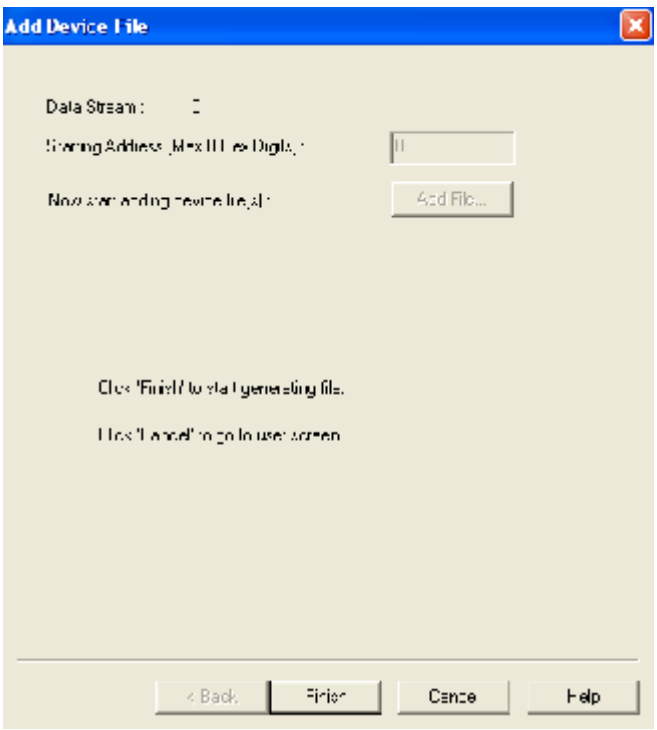

Рисунок 3.13 – Добавление bit файла в проект

❼ Нажмите Finish и затем Yes для генерации файла в MCS формате. Оставьте iMPACT открытым.

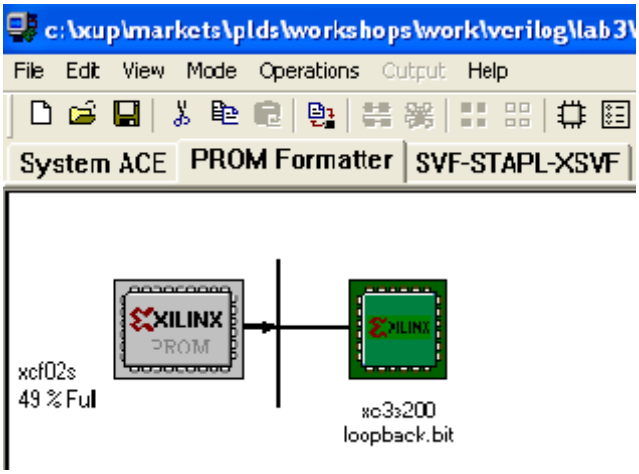

Рисунок 3.14 – PROM файл теперь отформатирован

Конфигурирование FPGA и Запуск LoopBack Теста Шаг 7

На этом шаге вы будете переключать конфигурационные режимы и конфигурировать flash-память с помощью MCS-файла, полученного на предыдущем шаге. После этого вы загрузите программный бит-код в FPGA из PROM и протестируете loopback на плате Digilent board.

❶ В утилите iMPACT выберите следующий пункт меню: Mode → Configuration Mode,

❷ Выберите следующий пункт меню: File → Initialize Chain. Нажмите <OK>, когда отрывшийся диалог говорит о том, что два устройства было обнаружено в JTAG-цепочке.

❸ Нажмите <Bypass>, когда нужно назначить конфигурационный файл для xc3s200

❹ Добавьте **time_const.mcs** файл для устройства xcf02s Platform Flash

❺ Нажмите правую кнопку мышки на xcf02s в окне iMPACT и выберите program.

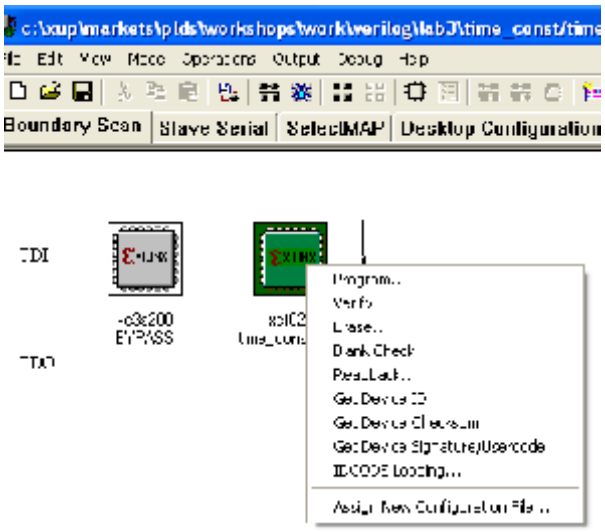

Рисунок 3.15 – Программирование xcf02s Platform Flash PROM

Замечание: Если программирование прошло успешно, вы увидите сообщение об этом.

❺ Нажмите кнопку сброса на плате или выключите/включите питание Digilent для конфигурирования Spartan-3 из flash-памяти и пощелкайте выключателями.

Заключение

В этой работе вы использовали Редактор Constraint Editor компании Xilinx для задания общих временных ограничений (Global Timing Constraints). Вы также научились понимать отчеты о времени прохождения сигналов по цепям внутри FPGA. Использование этой технологии – лучший путь для того, чтобы проверить производительность вашего продукта без выполнения полного цикла разработки.

Вы должны убедиться в том, что ваши ограничения произвели должный эффект. Для этого вы должны использовать Place&Rout процесс и оценить реальную производительность по отчету Post-Map Static Timing Report.

Правильные ответы

Представляемые ответы соответствуют оригинальной сборке проекта. Ваши результаты могут несколько отличаться в зависимости от того, какую версию ISE вы используете.

1. Результаты в таблице Post-Map:

Chart 1	**PERIOD constraint**	**OFFSET IN constraint**	**OFFSET OUT constraint**
Constraint	25 ns	6 ns	7.5 ns

Post-Map	~8 ns	~4 ns	~7.5 ns

2. Результаты в таблице Post-P&R:

Chart 2	**PERIOD constraint**	**OFFSET IN constraint**	**OFFSET OUT constraint**
Constraint	20 ns	6 ns	7.5 ns
Post-P&R	~15 ns	~5 ns	~7.5 ns

Практическое занятие №4. Настройка

Введение

Эта работа исследует процесс использования различных вариантов синтеза для того, чтобы дизайн для FPGA получил максимальную производительность.

Цели

По завершению этой работы вы сможете:

- Использовать «Keep Hierarchy» и веерный вариант синтеза для облегчения отладки проекта и улучшения результатов синтеза
- Читать отчет, создаваемый утилитой XST, для определения качества результатов синтеза

Методика

Вы будете менять настройки XST и анализировать полученные результаты.

Вы просмотрите дизайн, синтезированный с установками по умолчанию. Будете изменять настройки и просматривать результаты синтеза в RTL Viewer.

Просмотр Дизайна Шаг 1

Запустите ISE™ Project Navigator и откройте файл проекта *synth_lab.ise*.

❶ Для того чтобы открыть ISE, нажмите **Start → Programs → Xilinx ISE → Project Navigator**

❷ Выберите **File → Open Project**

- c*:\xup\fpgaflow\labs\vhdl\lab4*

❸ Выберите *synth_lab.*ise и нажмите Open

Обновите файл program.psm, созданный в лабораторной работе №3 так, чтобы была завершена задача №2 “Xilinx Rules!”. Скомпилируйте программу для генерации программного ROM файла. Добавьте ROM файл в проект.

❶ Откройте program.psm (расположенный в директории Ассемблера), используя любой текстовый редактор, и завершите задачу №2.

Подсказка: Все необходимые ASCII символы включены в список констант в начале программы. Для показа одного символа необходимы только две инструкции, load и output.

❷ Откройте в командной строке директорию, содержащую изменяемую программу.

❸ Введите следующую команду в командную строку для компиляции программы и генерации ROM файла

```
> kcpsm3 program
```

❹ Добавьте файл program.vhd в проект

Синтез и Имплементация с использованием настроек по умолчанию Шаг 2

Синтезируйте файл *loopback.vhd*.

❶ В окне Sources выберите ***loopback.vhd***

❷ В окне Processes для Source, дважды щелкните на **Synthesize - XST**

Просмотрите Отчет о Синтезе и ответьте на вопросы 1 и 2.

❶ В окне Processes для Source, раскройте процесс **Synthesize** и дважды нажмите на **View Synthesis Report**

❷ В поле для поиска наберите ***Timing Summary*** и нажмите ***<Enter>***

3. Сделайте заметку об ожидаемой тактовой частоте для дальнейшего сравнения.

_____

_____

_____

❸ В поле для поиска введите ***Device Utilization*** в окне **Find** и нажмите **Next**

4. Введите числа об используемых ресурсах в таблицу

Slices	
Slice Flip Flops	
4 input LUTs	
IOBs	
BRAMs	
Global Clocks	

❹ Закройте отчет

❺ Раскройте **Implement Design** и дважды щелкните на **Place & Route**

❻ Раскройте **Place & Route** и дважды щелкните на **View/Edit Placed Design (Floorplanner)**

❼ Просмотрите иерархию дизайна и размещение уровня, обратите внимание на то, что дизайн разбросан по всей поверхности чипа.

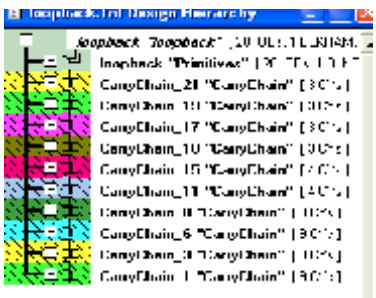

Рисунок 4.1 – Просмотр Иерархии Дизайна в Floorplanner

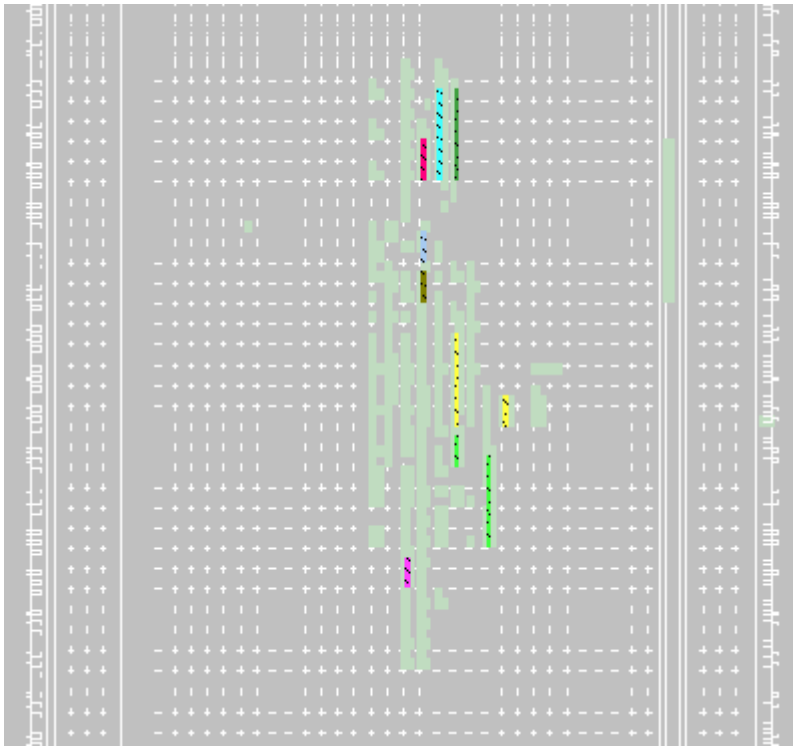

Рисунок 4.2 – Топология разбросанного дизайна

❸ Выйдите из Floorplanner

Изменение Настроек Синтеза — Шаг 3

В основном, HDL-дизайн – это набор иерархических блоков (компонентов). Сохранение иерархии дает преимущество быстрого синтеза, поскольку оптимизация может быть сделана в отдельных частях дизайна (внутри каждого компонента) меньшей сложности. Тем не менее, очень часто, соединение иерархических блоков дает хорошие результаты, поскольку процессы оптимизации применяются на всю используемую логику.

По умолчанию дизайн синтезируется в неорганизованный netlist. Вы будете менять параметры синтезатора для генерации иерархического netlist и исследовать полученные эффекты по использованию площади кристалла и тактовых частот.

Измените свойства синтеза как показано ниже и синтезируйте проект заново.

- Поддерживать Иерархию неизменной (Keep Hierarchy): **Yes**

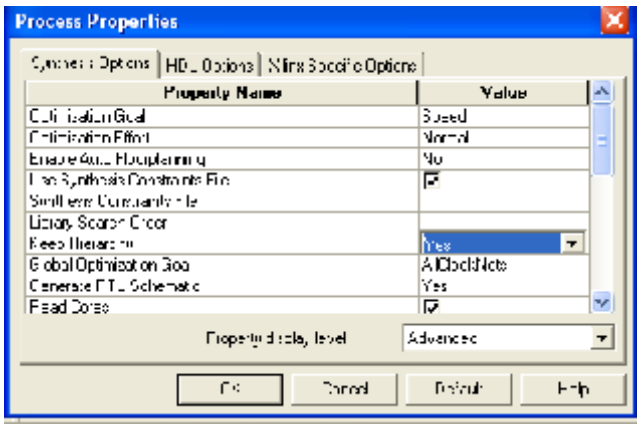

Рисунок 4.3 – Свойства Процесса Synthesis

❶ В окне Processes для Source, нажмите правую кнопку мыши на **Synthesize** и выберите **Properties**

❷ Установите для строки Keep Hierarchy значение Yes как показано на **Рис. 3**

❸ Нажмите <OK> и синтезируйте дизайн заново

❹ В поле для поиска наберите ***Timing Summary*** и нажмите ***<Enter>***

5. Сделайте заметку об ожидаемой тактовой частоте для дальнейшего сравнения.

_____

❺ В поле для поиска введите ***Device Utilization*** в окне **Find** и нажмите **Next**

6. Введите числа об используемых ресурсах в таблицу

Slices	
Slice Flip Flops	
4 input LUTs	
IOBs	
BRAMs	
Global Clocks	

❻ Введите ***Fanout*** в поле поиска и нажмите **Find Next**

5. Какая цепь имеет наибольшее разветвление и каким данным принадлежит?

_____

В этом разделе вы уменьшите максимальное разветвление (Fanout).

❶ Нажмите правую кнопку мышки на **Synthesize – XST** и выберите **Properties**
❷ В закладке **Xilinx Specific Options**, введите значение 50 для **Max Fanout**. Нажмите <OK>

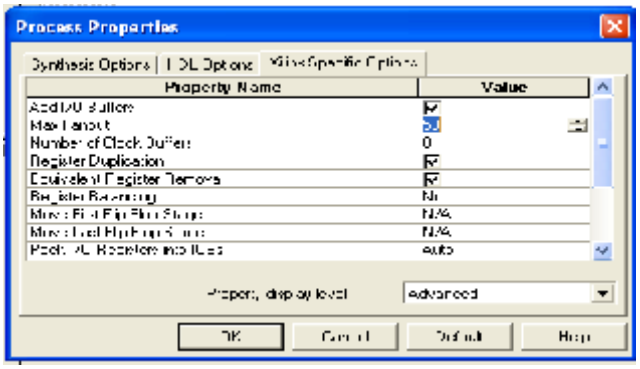

Рисунок 3.4 – Специальные установки

❸ Дважды щелкните на **Synthesize – XST** для того, чтобы синтезировать проект заново.
❹ Откройте отчет синтезатора и выполните поиск по слову **fanout**

6. Какая цепь имеет наибольшее разветвление и каким данным принадлежит?

_____

❺ В поле для поиска наберите ***Timing Summary*** и нажмите ***<Enter>***

7. Какая максимальная частота?

_____

❻ Раскройте Implement Design и дважды щелкните на Place & Route
❼ Откройте планировщик топологии и просмотрите иерархию дизайна.

Рисунок 4.5 – Просмотр Иерархии Дизайна в Floorplanner

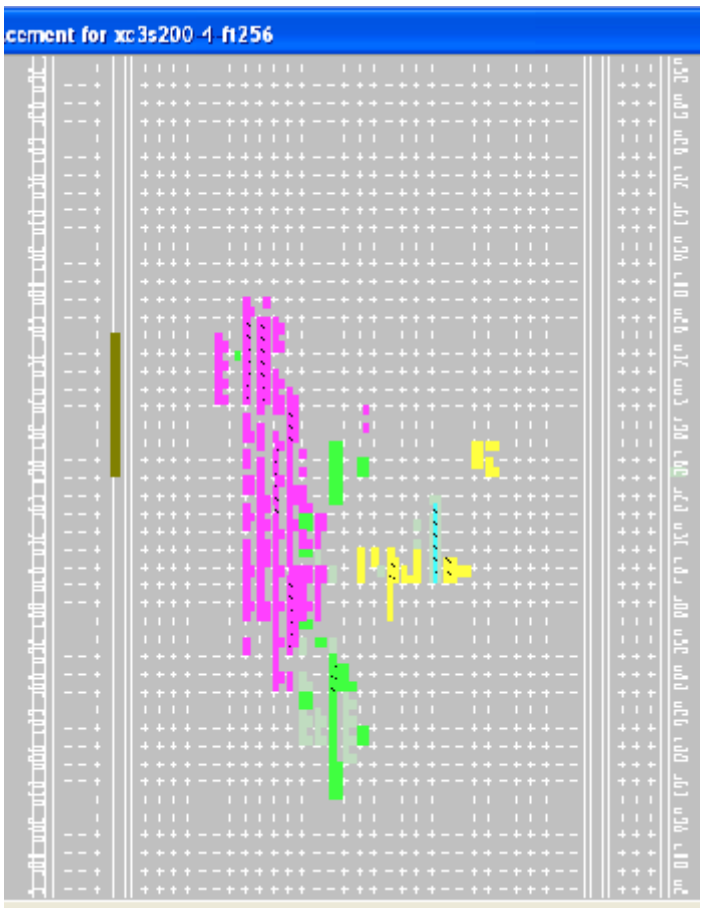

Рисунок 4.6 – Топология иерархического дизайна

Загрузка и тестирование системы **Шаг 5**

В этом разделе, вы создадите программный бит файл и сконфигурируете FPGA с помощью загрузочного JTAG кабеля.

❶ Соедините JTAG-кабель к плате Digilent Spartan-3.

❷ Откройте windows explorer и дважды щелкните на terminal.ht в директории проекта для того чтобы открыть сеанс связи.

❸ Добавьте в проект файл loopback.ucf, который содержит информацию о расположении контактов и временных ограничениях.

❹ Раскройте пункт меню Generate Programming file и дважды щелкните на **Configure Device (iMPACT)**

❺ Когда появится диалог Configure Devices, проверьте, что выбран режим **Boundary-Scan Mode**, нажмите <Next> и затем <Finish>

❻ Нажмите <OK>, когда появится диалог, сигнализирующий о том, что найдено два устройства в JTAG-цепочке

❼ Назначьте файл loopback.bit для устройства xc3s200 и выберите режим bypass для flash-памяти

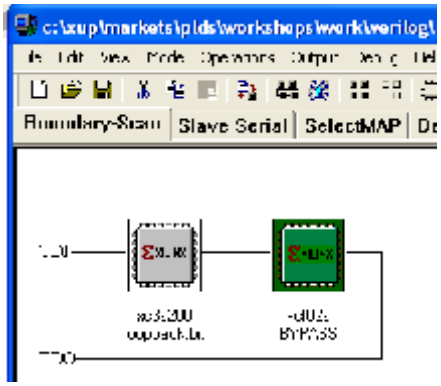

Рисунок 4.8 – Назначение конфигурационного файла

❽ Нажмите правую кнопку на xc3s200 и выберите Program.

Замечание: Вы должны увидеть в окне гипертерминала сообщение Xilinx Rules!

❾ Поменяйте положение переключателей и убедитесь, что программа работает правильно

Дополнительные задания

Запустите RTL Viewer и ответьте на вопросы, поставленные ниже.

В окне Processes для Source, раскройте процесс **Synthesize** и дважды щелкните на **View RTL Schematic**

Редактор ECS Schematic Editor предоставляет возможность просмотра принципиальной схемы разработанного дизайна launches.

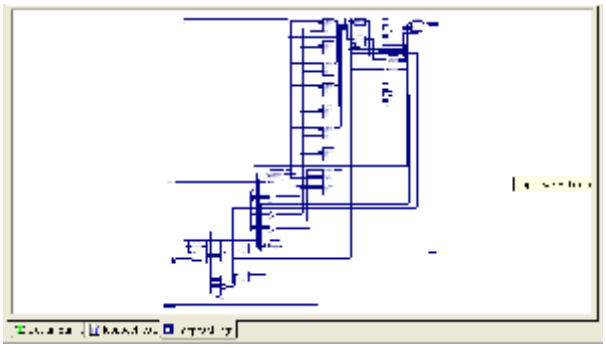

Рисунок 4.7 – Просмотр RTL-схемы

8. Сколько блоков внутренней памяти (RAM64x1S) использует процессор *PicoBlaze*?

_____

Заключение

Пытаясь улучшить производительность вашего дизайна, начинайте с определения и поиска узких мест по окончании синтеза. Убедитесь, что ваш код позволяет достичь максимальной производительности. Следующий шаг после этого, попытаться использовать настройки синтеза. Различные настройки могут помочь вам достичь более высокой производительности, но больше всего, конечно, производительность зависит от дизайна. Понимание того, какая настройка доступна, и какой эффект наступит после её применения сделает вас настоящим мастером HDL-программирования. Не стесняйтесь и не ленитесь исследовать эти возможности экспериментальным путем.

Ответы

1. Оценка частоты зависит от версии XST, используемой Вами.

2.

Slices	197
Slice Flip Flops	147
4 input LUTs	277
IOBs	20
BRAMs	1
Global Clocks	1

3. Оценка частоты зависит от версии XST, используемой Вами. После изменений настроек синтеза дизайн работает лучше.

4.

Slices	199
Slice Flip Flops	147
4 input LUTs	277
IOBs	20

BRAMs	1
Global Clocks	1

5.

port_id<2> with fanout of 77
Data path: my_kcpsm3/reg_loop_register_bit_2 to transmit/buf_0/register_bit_3

6.

Port_id<0> with a fanout of 43
Data Path: my_kcpsm3/reg_loop_register_bit_0 to transmit/buf_0/register_bit_3

7.

Minimum period: 13.716ns (Maximum Frequency: 72.907MHz)

Самостоятельная работа №1. Библиотеки и пакеты в языке VHDL.

Языки описания аппаратуры (Hardware Description Language), является формальной записью, которая может быть использована на всех этапах разработки цифровых электронных систем. Это возможно вследствие того, что язык легко воспринимается как машиной, так и человеком он может использоваться на этапах проектирования, верификации, синтеза и тестирования аппаратуры также как и для передачи данных о проекте, модификации и сопровождения. Наиболее универсальным и распространенным языком описания аппаратуры является VHDL. На этом языке возможно как поведенческое, так структурное и потоковое описание цифровых схем[1].

Язык VHDL используется во многих системах для моделирования цифровых схем, проектирования программируемых логических интегральных микросхем, базовых матричных кристаллов, заказных интегральных микросхем

С точки зрения программиста язык VHDL состоит как бы из двух компонент – общеалгоритмической и проблемно-ориентированной.

Общеалгоритмическая компонента VHDL- это язык, близкий по синтаксису и семантике к современным языкам программирования типа Паскаль, С и др. Язык относится к классу строго типизированных. Помимо встроенных (пакет STANDART) простых (скалярных) типов данных: целый, вещественный, булевский, битовый, данных типа время, данных типа ссылка (указатель) пользователь может вводить свои типы данных (перечислимый, диапазонный и др.).

Помимо скалярных данных можно использовать агрегаты: массивы array, в том числе и битовые векторы bit_vector, и символьные строки string, записи record, файлы file.

Последовательно выполняемые (последовательные) операторы VHDL могут использоваться в описании процессов, процедур и функций. Их состав включает:

- оператор присваивания переменной (:=);
- последовательный оператор назначения сигналу (<=);
- последовательный оператор утверждения (assert);
- условный (if);
- выбора (case);
- цикла (loop);
- пустой оператор (null);
- оператор возврата процедуры- функции (return);
- оператор последовательного вызова процедуры.

Язык поддерживает концепции пакетного и структурного программирования. Сложные операторы заключены в операторные скобки: if- end if; process- end process; case- end case; loop- end loop и т. д.

Различаются локальные и глобальные переменные. Область “видимости” локальных переменных ограничена пределами блока (процессного, процедурного, оператора блока, оператора описания архитектуры).

Фрагменты описаний, которые могут независимо анализироваться компилятором и при отсутствии ошибок помещаться в библиотеку проекта (рабочую библиотеку Work),называются проектными пакетами design unit. Такими пакетами могут быть объявление интерфейса объекта проекта entity, объявление архитектуры architecture, объявление конфигурации configuration, объявление интерфейса пакета package и объявление тела пакета package body.

Модули проекта, в свою очередь, можно разбить на две категории: первичные и вторичные. К первичным пакетам относятся объявления пакета, объекта проекта, конфигурации. К вторичным- объявление архитектуры, тела пакета. Один или несколько модулей проекта могут быть помещены в один файл, называемый файлом проекта (design file).

Каждый проанализированный модуль проекта помещается в библиотеку проекта (design library) и становится библиотечным модулем (library unit).

Каждая библиотека проекта в языке VHDL имеет логическое имя (идентификатор).

По отношению к сеансу работы с VHDL- системой существует два класса рабочих библиотек проекта: рабочие библиотеки и библиотеки ресурсов.

Рабочая библиотека- это библиотека WORK, с которой в данном сеансе работает пользователь и в которую помещается пакет, полученный в результате анализа пакета проекта.

Библиотека ресурсов - это библиотека, содержащая библиотечные модули, ссылка на которые имеется в анализируемом модуле проекта.

В каждый конкретный момент времени пользователь работает с одной рабочей библиотекой и произвольным количеством библиотек ресурсов.

Модули, как и в обычных алгоритмических языках, - это средство выделения из ряда программ и подпрограмм общих типов данных, переменных, процедур и функций, позволяющее упростить, в частности, процесс их замены.

Так же, как в описаниях проектируемых систем разделяются описания интерфейсов и тел, в VHDL у пакета разделяются описание интерфейса и тела пакета. По умолчанию предусмотрено подключение стандартных пакетов STANDART и TEXT 10. ПакетSTANDART, в частности, содержит описание булевских операций над битовыми данными и битовыми векторами. Нестандартные пакеты реализуются пользователями, желающими более точно отобразить свойства описываемых ими объектов. Например, можно в пользовательском пакете переопределить логические операции И, ИЛИ и НЕ и перейти от булевского (0, 1) к многозначному (1, 0,X? Z) алфавиту моделирования.

Проблемно- ориентированная компонента позволяет описывать цифровые системы в привычных, разработчику понятиях и терминах. Сюда можно отнести:

- понятие модельного времени now.
- данные типа time, позволяющие указывать время задержки в физических единицах
- данные вида сигнал signal, значение которых изменяется не мгновенно, как у обычных переменных, а с указанной задержкой, а также специальные операции и функции над ними
- Средства объявления объектов entity и их архитектур architecture.

Если говорить про операторную часть проблемно- ориентированной компоненты, то условно ее можно разделить на средства поведенческого описания аппаратуры (параллельные процессы и средства их взаимодействия); средства потокового описания (описание на уровне межрегистровых передач) – параллельные операторы назначения сигнала (<=) с транспортной transport или инерциальной задержкой передачи сигналов и средства структурного описания объектов (операторы конкретизации компонент с заданием карт портов port map и карт настройки generic map, объявление конфигурации и т. д.).

Параллельные операторы VHDL включают:

- оператор процесса process;
- оператор блока block;
- параллельный оператор назначения сигналу <=;
- оператор условного назначения сигналу when;
- оператор селективного назначения сигналу select;
- параллельный оператор утверждения assert;
- параллельный оператор вызова процедуры;
- оператор конкретизации компоненты port map;
- оператор генерации конкретизации generate;

Как видно из этого перечня, последовательные и параллельные операции назначения, вызова процедуры и утверждения различаются контекстно, то есть внутри процессов и процедур они последовательные, вне- параллельные.

Базовым элементом описания систем на языке VHDL является блок. Блок содержит раздел описаний данных и раздел параллельно исполняемых операторов. Частным случаем блока является описание архитектуры объекта. В рамках описания архитектуры могут использоваться внутренние, вложенные блоки. Наряду со всеми преимуществами блочной структуры программы и ее соответствия естественному иерархическому представлению структуры проекта операторы блока языка VHDL позволяют устанавливать, условия охраны (запреты) входа в блок. Только при истинности значения охранного выражения управление передается в блок и инициирует выполнение операторов его тела.

Алфавит языка

Как и любой другой язык программирования, VHDL имеет свой алфавит – набор символов, разрешенных к использованию и воспринимаемых компилятором. В алфавит языка входят:

1. Латинские строчные и прописные буквы:

A, B, . . . , Z и a, b, . . . , z

2. Цифры от 0 до 9.
3. Символ подчеркивания “_” (код ASCII номер 95).

Из символов, перечисленных в пп.1–3 (и только из них!) могут конструироваться идентификаторы в программе. Кроме того, написание идентификаторов должно подчиняться следующим правилам:

- идентификатор не может быть зарезервированным словом языка;
- идентификатор должен начинаться с буквы;
- идентификатор не может заканчиваться символом подчеркивания “_”;
- идентификатор не может содержать двух последовательных символов подчеркивания “__”;

Примеры корректных идентификаторов:

cont, clock2, full_add

Примеры некорректных идентификаторов:

1clock, _adder, add__sub, entity

Следует отметить что прописные и строчные буквы не различаются, т.е. идентификаторы clock и CLOCK являются эквивалентными.

1. Символ “пробел” (код 32), символ табуляции (код 9), символ новой строки (коды 10 и 13).

Данные символы являются разделителями слов в конструкциях языка. Количество разделителей не имеет значения. Т.о. следующие выражения для компилятора будут эквивалентны:

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

2. Специальные символы, участвующие в построении конструкций языка:

+ – * / = < > . , ( ) : ; # ' " |

3. Составные символы, воспринимаемые компилятором как один символ:

<= >= => := /=

Разделители между элементами составных символов недопустимы.

Комментарии

Признаком комментария являются два символа тире ("—"). 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 буквы).

Числа

В стандарте языка определены числа как целого, так и вещественного типа. Однако средства синтеза ПЛИС допускают применение только целых чисел. Целое число в VHDL может быть представлено в одной из четырех систем счисления: двоичной, десятичной, восьмеричной и шестнадцатеричной. Конкретные форматы написания числовых значений будут описаны далее при рассмотрении различных типов языка.

К разновидности числовых значений можно отнести также битовые строки.

Символы

Запись символа представляет собой собственно символ, заключенный в одиночные кавычки. Например:

'A', '*', ' '

В средствах синтеза ПЛИС область применения символов ограничена использованием их в качестве элементов перечислимых типов.

Строки

Строки представляют собой набор символов, заключенных в двойные кавычки. Чтобы включить двойную кавычку в строку, необходимо ввести две двойных кавычки. Например:

" A string"

"A string in a string ""A string"" "

Типы данных.

Подобно высокоуровневым языкам программирования, VHDL является языком со строгой типизацией. 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.

Необходимо отметить, что в данном пособии рассматриваются не все типы данных, определенные в стандарте, а только те, которые поддерживаются средствами синтеза ПЛИС.

Простые типы

Следующие простые типы являются предопределенными:

1. BOOLEAN (логический) – объекты данного типа могут принимать значения FALSE (ложь) и TRUE (истина).
2. INTEGER (целый) – значения данного типа представляют собой 32-разрядные числа со знаком. Объекты типа INTEGER могут содержать значения из диапазона $-(2^{31}-1)\ldots 2^{31}-1$ (-2147483647 … 2147483647).
3. BIT (битовый) – представляет один логический бит. Объекты данного типа могут содержать значение '0' или '1'.
4. STD_LOGIC (битовый) – представляет один бит данных. Объекты данного типа могут принимать 9 состояний. Данный тип определен стандартом IEEE 1164 для замены типа BIT.
5. STD_ULOGIC (битовый) – представляет один бит данных. Объекты данного типа могут принимать 9 состояний. Данный тип определен стандартом IEEE 1164 для замены типа BIT (см. Примечание).
6. ENUMERATED (перечислимый) – используется для задания пользовательских типов.
7. SEVERITY_LEVEL – перечислимый тип, используется только в операторе ASSERT.
8. CHARACTER – символьный тип.

Примечание

В действительности тип STD_ULOGIC является базовым типом для типа STD_LOGIC, т.е. объекты обеих типов могут принимать одно и то же множество значений и имеют одинаковый набор допустимых операций. Единственное различие между типами заключается в том, что для типа STD_ULOGIC не определена функция разрешения (resolving function). В языке VHDL функция разрешения используется для определения значения сигнала, имеющего несколько источников (драйверов).

Пример: Выходы двух буферов с тремя состояниями подключены к одной цепи X. Пусть выход одного буфера установился в состояние 'Z', а выход другого – в состояние '1'. Функция разрешения определяет, что в этом случае значение сигнала X будет равно '1'.

Поскольку для типа STD_ULOGIC не определена функция разрешения, сигналы этого типа могут иметь только один источник.

Далее рассматривается только тип STD_LOGIC, однако все сказанное будет справедливо и для типа STD_ULOGIC. На практике, в подавляющем большинстве случаев достаточно использования типа STD_LOGIC.

Сложные типы

Из всей совокупности сложных типов, определенных в стандарте языка, для синтеза логических схем используются только массивы (тип ARRAY) и записи (тип RECORD). Однако тип RECORD поддерживается не всеми средствами синтеза и в данном пособии рассмотрен не будет.

Следующие типы-массивы являются предопределенными:

1. BIT_VECTOR – одномерный массив элементов типа BIT;
2. STD_LOGIC_VECTOR – одномерный массив элементов типа STD_LOGIC;
3. STD_ULOGIC_VECTOR – одномерный массив элементов типа STD_ULOGIC;
4. STRING – одномерный массив элементов типа CHARACTER;

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

Описание простых типов

Тип BOOLEAN

Тип BOOLEAN является перечислимым типом. Объект данного типа может принимать значения FALSE (ложь) и TRUE (истина), причем FALSE эквивалентно 0, а TRUE эквивалентно 1.

Все три типа объектов в VHDL (константы, переменные и сигналы) могут иметь тип BOOLEAN. Таким объектам может быть присвоено только значение типа BOOLEAN.

Пример

```
PROCESS (a, b)
VARIABLE cond : BOOLEAN;
BEGIN
cond := a > b;
IF cond THEN
output <= '1';
ELSE
output <= '0';
END IF;
END PROCESS;
```

Операторы отношения

Значения типа BOOLEAN могут участвовать в выражениях. Операторы отношения (=, /=, <, <=, >, >=) определены для операндов типа BOOLEAN и одномерных массивов,

содержащих элементы типа BOOLEAN. Результат выражения также имеет тип BOOLEAN.

(Как для всех перечислимых типов, операции сравнения над одномерными массивами типа BOOLEAN производятся поэлементно, начиная с крайнего левого элемента).

Логические операторы

Для операндов типа BOOLEAN и одномерных массивов, содержащих элементы типа BOOLEAN определены все логические операции (AND, OR, NAND, NOR, XOR и NOT). Тип и размер операндов должны быть одинаковыми. Тип и размер результата такой же как тип и размер операндов.

Оператор конкатенации

Оператор конкатенации также определен для операндов типа BOOLEAN и одномерных массивов, содержащих элементы типа BOOLEAN. Результат выражения представляет собой одномерный массив, содержащий элементы типа BOOLEAN; размер массива равен сумме размеров операндов.

Другие операторы

Другие операции над операндами типа BOOLEAN не определены.

Тип INTEGER

Стандарт VHDL определяет тип INTEGER для использования в арифметических выражениях. По умолчанию объекты типа INTEGER имею размерность 32 бита и представляют целое число в интервале $-(2^{31}-1)$ . . . $2^{31}-1$ (-2147483647 . . . 2147483647). Стандарт языка позволяет также объявлять объекты типа INTEGER, имеющие размер меньше 32 бит, используя ключевое слово RANGE, ограничивающее диапазон возможных значений:

SIGNAL X : INTEGER RANGE –127 TO 127

Данная конструкция определяет X как 8-битное число.

Кроме того, можно определить ограниченный целый тип, используя следующую конструкцию:

TYPE *имя_типа* **IS RANGE** *диапазон_индексов***;**

диапазон_индексов определяется следующим образом:

m **TO** *n*

n **DOWNTO** *m*

где *m*, *n* – целочисленные константы, *m* <= *n*.

Пример

TYPE byte_int 0 TO 255;

TYPE signed_word_int is range –32768 TO 32768;

TYPE bit_index is range 31 DOWNTO 0;

Значения типа INTGER записываются в следующей форме:

[основание #] разряд { [_] разряд} [#]

По умолчанию "основание" принимается равным 10. Допустимыми также являются значения 2, 8, 16.

При записи числа допускается использование одиночных символов подчеркивания, которые не влияют на результирующее значение.

Пример

CONSTANT min : INTEGER := 0;

CONSTANT group : INTEGER := 13_452; – – эквивалентно 13452

CONSTANT max : INTEGER := 16#FF#;

Допустимое использование

Операторы отношения

Значения типа INTEGER могут участвовать в выражениях. Операторы отношения (=, /=, <, <=, >, >=) определены для операндов типа INTEGER и одномерных массивов, содержащих элементы типа INTEGER. Результат выражения имеет тип BOOLEAN.

Арифметические операторы

Операторы +, –, ABS допустимы для операндов типа INTEGER. Результат выражения имеет тип INTEGER.

Операторы *, /, MOD, REM допустимы в следующих случаях:

- если оба операнда являются константами (CONSTANT);
- если второй операнд является константой и его значение равно 2^n, где n = 0, 1, 2, 3….

Применение операторов *, /, MOD, REM недопустимо если оба операнда являются сигналами (SIGNAL) или переменными (VARIABLE).

Оператор возведения в степень (**) как правило не поддерживается средствами синтеза.

Другие операторы

Другие операции над операндами типа INTEGER не определены.

Тип BIT

Объект данного типа может принимать значение '0' (лог.0) или '1' (лог.1).

Примечание

Стандартом IEEE 1164 определена замена типа BIT на более гибкий тип STD_LOGIC. Поэтому использование типа BIT в новых разработках не рекомендуется.

Допустимое использование

Операторы отношения

Значения типа BIT могут участвовать в выражениях. Операторы отношения (=, /=, <, <=, >, >=) определены для операндов типа BIT и одномерных массивов, содержащих элементы типа BIT. Результат выражения имеет тип BOOLEAN.

(Как для всех перечислимых типов, операции сравнения над одномерными массивами типа BIT производятся поэлементно, начиная с крайнего левого элемента).

Логические операторы

Для операндов типа BIT и одномерных массивов, содержащих элементы типа BIT определены все логические операции (AND, OR, NAND, NOR, XOR и NOT). Тип и размер операндов должны быть одинаковыми. Тип и размер результата такой же как тип и размер операндов.

Оператор конкатенации

Оператор конкатенации также определен для операндов типа BIT и одномерных массивов, содержащих элементы типа BIT. Результат выражения представляет собой одномерный массив, содержащий элементы типа BIT; размер массива равен сумме размеров операндов.

Другие операторы

Другие операции над операндами типа BIT не определены.

Тип STD_LOGIC

Типы STD_LOGIC является перечислимым типом. Объекты типа STD_LOGIC могут принимать 9 значений: '0', '1', 'Z', '–', 'L', 'H', 'U', 'X', 'W'.

Для синтеза логических схем используются только первые четыре:

'0' – логический "0";

'1' – логическая "1";

'Z' – третье состояние;

'–' – не подключен.

Допустимое использование

Операторы отношения

Значения типа STD_LOGIC могут участвовать в выражениях. Операторы отношения (=, /=, <, <=, >, >=) определены для операндов типа STD_LOGIC и одномерных массивов, содержащих элементы типа STD_LOGIC. Результат выражения имеет тип BOOLEAN.

(Как для всех перечислимых типов, операции сравнения над одномерными массивами типа STD_LOGIC производятся поэлементно, начиная с крайнего левого элемента).

Логические операторы

Для операндов типа STD_LOGIC и одномерных массивов, содержащих элементы типа STD_LOGIC определены все логические операции (AND, OR, NAND, NOR, XOR и NOT). Тип и размер операндов должны быть одинаковыми. Тип и размер результата такой же как тип и размер операндов.

Оператор конкатенации

Оператор конкатенации также определен для операндов типа STD_LOGIC и одномерных массивов, содержащих элементы типа STD_LOGIC. Результат выражения представляет собой одномерный массив, содержащий элементы типа STD_LOGIC; размер массива равен сумме размеров операндов.

Другие операторы

Другие операции над операндами типа STD_LOGIC не определены.

Перечислимый тип

Перечислимый тип – это такой тип данных, при котором количество всех возможных значений конечно. Строго говоря, все описанные выше типы являются перечислимыми.

Применение перечислимых типов преследует две цели:

- улучшение смысловой читаемости программы;
- более четкий и простой визуальный контроль значений.

Наиболее часто перечислимый тип используется для обозначения состояний конечных автоматов.

Перечислимый тип объявляется путем перечисления названий элементов-значений. Объекты, тип которых объявлен как перечислимый, могут содержать только те значения, которые указаны при перечислении.

Элементы перечислимого типа должны быть идентификаторами или символами, которые должны быть уникальными в пределах одного типа. Повторное использование названий элементов в других перечислимых типах разрешается.

Объявление перечислимого типа имеет вид:

TYPE *имя_типа* **IS (** *название_элемента* [**,** *название_элемента*] **);**

Пример

```
Type State_type IS (stateA, stateB, stateC);
VARIABLE State : State_type;
.
.
.
State := stateB
```

В данном примере объявляется переменная *State*, допустимыми значениями которой являются *stateA*, *stateB*, *stateC*.

Примеры предопределенных перечислимых типов:

```
TYPE SEVERITY_LEVEL IS (NOTE, WARNING, ERROR, FAILURE);
TYPE BOOLEAN IS (FALSE, TRUE);
TYPE BIT IS ('0', '1');
TYPE STD_LOGIC IS ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '–');
```

Любой перечислимый тип имеет внутреннюю нумерацию: первый элемент всегда имеет номер 0, второй – номер 1 и т.д. Порядок нумерации соответствует порядку перечисления.

Допустимое использование

Операторы отношения

Значения определенных пользователем перечислимых типов могут участвовать в выражениях. Операторы отношения (=, /=, <, <=, >, >=) определены как для перечислимых типов, так и для одномерных массивов, содержащих элементы этих типов. Результат выражения имеет тип BOOLEAN.

Оператор конкатенации

Оператор конкатенации определен для операндов имеющих перечислимый тип и одномерных массивов, содержащих элементы перечислимого типа. При этом оба операнда должны быть одного типа. Результат выражения представляет собой одномерный массив, тип элементов которого равен типу операндов; размер массива равен сумме размеров операндов.

Другие операторы

К операндами перечислимых типов применим оператор указания типа. Данный оператор используется для уточнения типа объекта в случае если одно и то же название элемента используется различными типами.

Пример

```
TYPE COLOR IS (green, red, tellow, orange);
TYPE FRUIT IS (orange, apple, pear);
VARIABLE VC : COLOR;
VARIABLE VF : FRUIT;
VC := COLOR'orange;
VF := FRUIT'orange;
```

Другие операции над операндами перечислимых типов определенных пользователем не определены.

Тип SEVERITY_LEVEL

Переменные этого типа используются только в операторе ASSERT и игнорируются при синтезе логических схем.

Переменные типа SEVERITY_LEVEL могут принимать следующие значения: NOTE, WARNING, ERROR и FAILURE.

Тип CHARACTER

Перечислимый тип. Значением объекта данного типа может быть любой символ из набора ASCII (128 первых символов).

Массивы

Массив (тип "массив") является сложным типом. Массив представляет собой упорядоченную структуру однотипных данных. Массив имеет диапазон индексов, который может быть возрастающим либо убывающим. На любой элемент массива можно сослаться, используя его индекс. Несмотря на то, что стандартом языка допускается использование массивов любой размерности, для синтеза ПЛИС используются только одномерные (поддерживаются всеми средствами синтеза) и двумерные (поддерживаются ограниченным числом средств синтеза) массивы.

Также можно сослаться на часть одномерного массива, используя вместо индекса диапазон индексов.

Существуют две разновидности типа "массив": ограниченный (*constrained*) и неограниченный (*unconstrained*).

- Объявление ограниченного типа определяет границы диапазона индексов (число элементов массива) в каждом измерении при определении типа.
- Объявление неограниченного типа не определяет границы диапазона индексов. В этом случае границы диапазона устанавливаются при объявлении конкретного экземпляра объекта данного типа.

Объявление ограниченного типа "массив" имеет вид:

TYPE *имя_типа* **IS**
ARRAY (*диапазон_индексов* [**,** *диапазон_индексов*] **)**
OF *тип_элемента***;**

диапазон_индексов может определяться двумя способами:

1) явным заданием границ диапазона

m **TO** *n*

n **DOWNTO** *m*

где *m*, *n* – целочисленные константы, *m* <= *n*.

2) с использованием идентификатора ограниченного подтипа. В этом случае значения границ подтипа являются значениями границ индекса массива. Описание подтипов см. далее.

Объявление неограниченного типа "массив" имеет вид:

TYPE *имя_типа* **IS**

ARRAY (*тип_индекса*

[**,** *тип_индекса*] **)**

OF *тип_элемента***;**

тип_индекса определяется следующим образом:

подтип **RANGE <>**

где *подтип* может быть:

INTEGER – индекс находится в диапазоне $-(2^{31}-1) \ldots 2^{31}-1$;

NATURAL – индекс находится в диапазоне $0 \ldots 2^{31}-1$;

POSITIVE – индекс находится в диапазоне $1 \ldots 2^{31}-1$;

Примеры

1) Объявление ограниченного массивного типа:

```
TYPE word IS ARRAY (31 DOWNTO 0) OF STD_LOGIC;
TYPE register_bank IS ARRAY (byte RANGE 0 TO 132) OF INTEGER;
TYPE memory IS ARRAY (address) OF word; -- двумерный массив
```

2) Объявление неограниченного массивного типа:

```
TYPE logic IS ARRAY (INTEGER RANGE <>) OF BOOLEAN;
```

3) Объявление двумерного массива:

```
TYPE Reg IS ARRAY (3 DOWNTO 0) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
TYPE transform IS ARRAY (1 TO 4, 1 TO 4) OF BIT;
```

Как было сказано, в языке имеется несколько предопределенных типов "массив". Их объявления выглядят следующим образом:

```
TYPE STRING IS ARRAY (POSITIVE RANGE <>) OF CHARACTER;
TYPE BIT_VECTOR IS ARRAY (NATURAL RANGE <>) OF BIT;
TYPE STD_LOGIC_VECTOR IS ARRAY (NATURAL RANGE <>) OF
STD_LOGIC;
TYPE STD_ULOGIC_VECTOR IS ARRAY (NATURAL RANGE <>) OF
STD_ULOGIC;
```

Объявление объекта типа "неограниченный массив" должно содержать ограничения на индекс. Диапазон изменения индексов может быть ограничен:

1. с использованием ключевых слов TO или DOWNTO:

```
TYPE data_memory_type IS ARRAY (INTEGER RANGE <>) OF BIT;
. . .
VARIABLE data_ memory : data_memory_type(0 TO 255);
```

2. путем использования диапазона начального значения:

```
CONSTANT part_id : STRING :="M38006";
```

СТРОКИ, БИТОВЫЕ СТРОКИ И АГРЕГАТЫ

Строки,битовые строки,агрегаты (*strings*, *bit strings*, *aggregates*) используются для конструирования значений обьектов массивных типов. Они могут использоваться в любом месте, где допускается значение типа массив, например, как начальное значение константы или операнд в выражении.

Строковая запись может быть использована для представления значений как объектов некоторых предопределенных типов (*string*,*bit_vector*, *std_logic_vector*), так и для любого одномерного массива, элементы которого имеют тип *character*; например:

```
TYPE bit6 IS ('U', '0', '1', 'F', 'R', 'X');
TYPE bit6_data IS ARRAY (POSITIVE RANGE<>) OF bit6;
. . .
SIGNAL data_bus : bit6_data(15 DOWNTO 0);
.
data_bus <= "UUUUUUUUFFFFFFFF";
```

VHDL позволяет компактно описывать битовые строки (значение типа *bit_vector* или *std_logic_vector*) в базисе 2, 8 и 16. Например:

```
CONSTANT clear : bit_vector := B"00_101_010";
CONSTANT empty : bit_vector := O"052";
CONSTANT null : bit_vector := X"2A";
```

Все три константы имеют одно и то же значение. Отметим, что символы подчеркивания могут использоваться в любом месте битовой строки для облегчения чтения. Расширенными цифрами (extended digits) для шестнадцатеричного представления являются буквы от A до F, причем могут использоваться как прописные, так и строчные буквы.

Массивные агрегаты используются для присваивания объектам типа массив значений. Массивные агрегаты формируются при помощи позиционной (positional) записи, поименованной (named) записи или комбинации этих двух форм. Рассмотрим пример.

Предположим, что имеются следующие описания

```
TYPE ArayType IS ARRAY (1 TO 4) OF CHARACTER;
. . .
VARIABLE Test : ArrayType;
```

и требуется, чтобы переменная *Test* содержала элементы 'f', 'o', 'o', 'd' в указанном порядке.

Позиционная запись имеет вид:

```
Test := ('f', 'o', 'o', 'd');
```

Агрегат в данном случае записывается как список значений элементов, разделенных запятыми. Первое значение назначается элементу с наименьшим значением индекса (крайнему левому).

Поименованная запись имеет вид:

```
Test := (1=>'f', 3=>'o', 4=>'d', 2=>'o');
```

В этом случае агрегат также является списком, элементы которого разделены запятыми, однако элементы списка имеют формат:

позиция => значение

Порядок перечисления элементов при поименованной записи не имеет значения.

Комбинированная запись имеет вид:

```
Test := ('f', 'o', 4=>'d', 3=>'o');
```

В этом случае сначала записываются элементы, присваиваемые с использованием позиционной записи, а оставшиеся элементы присваиваются с использованием поименованной записи.

При формировании агрегата с использованием поименованной (или комбинированной) записи вместо номера позиции можно указывать ключевое слово **OTHERS**, которое определяет значение для всех элементов, которые еще не были описаны в агрегате. Например

```
Test := ('f', 4=>'d', OTHERS=>'0');
```

Подтипы

Использование подтипов позволяет объявлять объекты, принимающие ограниченный набор значений из диапазона, допустимого для базового типа.

Подтипы применяются в двух случаях:

1) Подтип может ограничить диапазон значений базового скалярного типа (ограничение по диапазону). В этом случае объявление подтипа выглядит следующим образом

SUBTYPE *имя_подтипа* **IS** *имя_базового_типа* **RANGE** *диапазон_индексов***;**

диапазон_индексов определяется следующим образом:

m **TO** *n*

n **DOWNTO** *m*

где *m*, *n* – целочисленные константы, *m* <= *n*.

Пример

Предположим что разработчик желает создать сигнал *A* типа *severity* и что *A* может принимать значения только *OKAY*, *NOTE* и*WARNING*.

```
TYPE sevirity IS (OKAY, NOTE, WARNING, ERROR, FAILURE);
SUBTYPE go_status IS severity RANGE OKAY TO WARNING;
SIGNAL A : go_status;
```

Базовый тип и ограничение диапазона могут быть включены непосредственно в объявление объекта. Объявление сигнала A, эквивалентное приведенному выше будет выглядеть следующим образом

```
SIGNAL A : severity RANGE OKAY TO WARNING;
```

Другие примеры:

```
SUBTYPE pin_count IS INTEGER RANGE 0 TO 400;
SUBYUPE digits IS character range '0' TO '9';
```

Подтипы, объявленные таким образом, могут также участвовать в описании ограниченных массивных типов (см. "*Массивы*").

2) Подтип может определить границы диапазона индексов для неограниченного (*unconstrained*) массивного типа. В этом случае объявление подтипа выглядит следующим образом

SUBTYPE *имя_подтипа* **IS** *имя_базового_типа* **(***диапазон_индексов* [**,** *диапазон_индексов*] **);**

диапазон_индексов определяется следующим образом:

m **TO** *n*

n **DOWNTO** *m*

где *m*, *n* – целочисленные константы, *m* <= *n*.

Такое использование подтипа может быть удобно при наличии большого числа объектов некоторого типа с одинаковыми ограничениями на индексы.

Пример

```
TYPE bit6_data IS ARRAY (POSITIVE RANGE <>) OF bit6;
SUBTYPE data_store IS bit6_data (63 DOWNTO 0);
SIGNAL A_reg, B_reg,C_reg : data_store;
VARIABLE temp : data_store;
```

В языке имеются два предопределенных числовых подтипа *natural* и *positive*, которые определены как:

```
SUBTYPE NATURAL IS INTEGER RANGE 0 TO highest_integer;
SUBTYPE POSITIVE IS INTEGER RANGE 1 TO highest_integer;
```

4.4. Операторы VHDL

4.4.1. Основы синтаксиса

Исходный текст программы на VHDL состоит из последовательностей операторов, записанных с учетом следующих правил:

- каждый оператор – это последовательность слов, содержащих буквы английского алфавита, цифры и знаки пунктуации

- слова разделяются произвольным количеством пробелов, табуляций и переводов строки
- операторы разделяются символами “;”
- в некоторых операторах могут встречаться списки объектов, разделяемые символами “,” или “;”

Комментарии могут быть включены в текст программы с помощью двух подряд идущих символов “--”. После появления этих символов весь текст до конца строки считается комментарием.

Для указания системы счисления для констант могут быть применены спецификаторы:

- B – двоичная система счисления, например B”0011”
- O– восьмеричная система счисления, например O”3760”
- H – шестнадцатеричная система счисления, например H”F6A0”

Объекты

Объекты являются контейнерами для хранения различных значений в рамках модели. Идентификаторы объектов содержать буквы, цифры и знаки подчеркивания. Идентификаторы должны начинаться с буквы, не должны заканчиваться знаком подчеркивания, и знаки подчеркивания не могут идти подряд. Прописные и строчные буквы в VHDL не различаются. Все объекты должны быть явно объявлены перед использованием, за исключением переменной цикла в операторе ***for***, которая объявляется по умолчанию.

Каждый объект характеризуется типом и классом. Типы разделяются на предопределенные в VHDL и определяемые пользователем. Тип показывает, какого рода данные может содержать объект. Класс показывает, что можно сделать с данными, содержащимися в объекте. В VHDL определены следующие классы объектов:

- ***Constant*** – константы. Значение константы определяется при ее объявлении и не может быть изменено. Константы могут иметь любой из поддерживаемых типов данных.
- ***Variable*** – переменные. Значение, хранимое в переменной, меняется везде, где встречается присваивание данной переменной. Переменные могут иметь любой из поддерживаемых типов данных.
- ***Signal*** – сигналы. Сигналы представляют значения, передаваемые по проводам и определяемые присвоением сигналов (отличным от присвоения переменных). Сигналы могут иметь ограниченный набор типов (обычно *bit*, *bit_vector*, *std_logic*,*std_logic_vector*, *integer*, и, возможно, другие, в зависимости от среды разработки).

Повторное использование присваивания сигналов в наборе параллельных операторов не допускается. В наборе последовательных операторов такое присваивание допустимо и даст значение сигнала, соответствующее последнему по порядку присваиванию.

Синтаксис объявления объектов:

Constant { name [, name] } : Type [(index_range [, index_range])] := initial_value;

Variable { name [, name] } : Type [(index_range [, index_range])] [:= initial_value];

Signal { name [, name] } : Type [(index_range)];

Диапазон значений индексов задается в виде *int_value* ***to*** *int_value* или *int_value* ***downto*** *int_value.*

Атрибуты

Атрибуты (или иначе свойства) определяют ктеристики объектов, к которым они относятся. Стандарт VHDL предусматривает как предопределенные, так и определяемые пользователем атрибуты, однако современные инструментальные средства в большинстве

своем поддерживают только предопределенные атрибуты. Для обращения к атрибутам объекта используется символ “’” (например*A1’left*).

В VHDL определены следующие атрибуты:

‘left – левая граница диапазона индексов массива

‘right – правая граница диапазона индексов массива

‘low – нижняя граница диапазона индексов массива

‘high – верхняя граница диапазона индексов массива

‘range – диапазон индексов массива

‘reverse_range – обращенный диапазон индексов массива

‘length – ширина диапазона индексов массива

4.4.4. Компоненты

Объявление компонента определяет интерфейс к модели на VHDL (***entity*** и ***architecture***), описанной в другом файле. Обычно объявление компонента совпадает с соответствующим объявлением ***entity***. Они могут различаться только значениями по умолчанию. Эти значения используются, когда какой-либо из отводов компонента остается не присоединенным (ключевое слово***open***) при установке компонента в схему.

Оператор объявления компонента может находиться внутри объявления ***architecture*** или в заголовке пакета (***package***). Соответствующие компоненту объявления ***entity*** и ***architecture*** не обязательно должны существовать в момент анализа схемы. В момент моделирования или синтеза должны существовать объявления entity и architecture для компонентов, которые не только объявлены, но и установлены в схему. Это позволяет, например, конструктору задать объявления библиотечных элементов, а реальное их описание (объявления entity и architecture) задавать по мере использования этих элементов в конструкции.

Объявление компонента записывается следующим образом:

Component name

[port(port_list);]

end component;

Выражения

Выражения могут содержать следующие операторы: преобразование типа, ***and***, ***or***, ***nand***, ***nor***, ***xor***, =, /=, <, <=, >, >=, +, -, &, *, /,***mod***, ***rem***, ***abs***, ***not***.

В зависимости от избранной САПР при синтезе может поддерживаться подмножество приведенных выше операторов. Порядок вычисления выражений определяется приоритетом операторов:

and, ***or***, ***nand***, ***nor***, ***xor*** – самый низкий приоритет

=, /=, <, <=, >, >=

+, -, & (бинарные)

+, - (унарные)

*, /, ***mod***, ***rem***

abs, ***not*** – высший приоритет

Операторы с более высоким приоритетом выполняются раньше. Чтобы изменить такой порядок используются скобки.

При моделировании (но не при синтезе) схемы возможно также описание формы сигнала в виде выражения. Записывается оно следующим образом:

Value_expression [**after** time_expression]

{ , value_expression [**after** time_expression] }

Операторы

С помощью операторов описывается алгоритм, определяющий функционирование схемы. Они могут находиться в теле функции, процедуры или процесса.

Wait until *condition;*

Приостанавливает выполнение процесса, содержащего данный оператор до момента выполнения условия.

Signal <= expression

Оператор присваивания сигнала устанавливает его значение равным выражению справа.

Variable := expression

Оператор присваивания устанавливает значение переменной равным выражению справа.

Procedure_name (parameter { , parameter })

Оператор вызова процедуры состоит из имени процедуры и списка фактических параметров.

If condition then
Sequence_of_statements
{ Elsif condition then
Sequence_of_statements }
[else
sequence_of_statements]
end if ;

Оператор ***if*** используется для ветвления алгоритма по различным условиям.

Case expression is
When choices_list => sequence_of_statements;
{ When choices_list => sequence_of_statements; }
When others => sequence_of_statements;
End case;

Оператор ***case*** подобно оператору ***if*** задает ветвление алгоритма. Значения в списках разделяются символом "|". Когда значение выражения встречается в одном из списков значений, выполняется соответствующая последовательность операторов. Если значение выражения не присутствует ни в одном из списков, то выполняется список операторов, соответствующий ветви ***when others***.

[loop_label :]
for *loop_index_variable* ***in*** *range* ***loop***
sequence_of_statements
end loop *[loop_label] ;*

Оператор цикла позволяет многократно выполнить последовательность операторов. Диапазон значений задается в виде *value1* ***to****value2* или *value1* ***downto*** *value2*. Переменная цикла последовательно принимает значения из заданного диапазона. Количество итераций равно количеству значений в диапазоне.

Return *expression ;*

Этот оператор возвращает значение из функции.

Null

Пустой оператор, не выполняет никаких действий.

Интерфейс и тело объекта.

Полное VHDL- описание объекта состоит как минимум из двух отдельных описаний: описание интерфейса объекта и описание тела объекта (описание архитектуры).

Интерфейс описывается в объявлении объекта **entity declaration** и определяет входы и выходы объекта, его входные и выходные порты **ports** и параметры настройки **generic**. Параметры настройки отражают тот факт, что некоторые объекты могут иметь управляющие входы, с помощью которых может производиться настройка экземпляров объектов в частности, задаться временем задержки.

Например, у объекта Q1 три входных порта X1, X2, X3 и два выхода У1, У2. Описание его интерфейса на VHDL имеет вид:

Entity Q1 is
Port (X1, X2, X3: in real; Y1, Y2: out real);

End Q1.

Порты объекта характеризуются направлением потока информации. Они могут быть:

- входными (in)
- выходными (out)
- двунаправленными (inout)
- двунаправленными буферными (buffer)
- связными (linkage)
- А также имеют тип, характеризующий значения поступающих на них сигналов:
- целый (integer)
- вещественный (real)
- битовый (bit)
- символьный (character)

Тело объекта специфицирует его структуру или поведение. Его описание по терминологии VHDL содержится в описании его архитектуры **architecture.**

VHDL позволяет отождествлять с одним и тем же интерфейсом несколько архитектур. Это связано с тем, что в процессе проектирования происходит проработка архитектуры объекта: переход от структурной схемы к электрической принципиальной, от поведенческого к структурному описанию.

Средства VHDL для отображения структур цифровых систем базируются на представлении о том, что описываемый объект **entity**представляет собой структуру из компонент **component** соединяемых друг с другом линиями связи. Каждая компонента, в свою очередь, является объектом и может состоять из компонент низшего уровня (иерархия объектов). Взаимодействуют объекты путем передачи сигналов **signal** по линиям связи. Линии связи подключаются к входным и выходным портам компонент. В VHDLсигналы отождествляются с линиями связи.

Имена сигналов и имена линий связи совпадают (они отождествляются). Для сигналов (линий), связывающих компоненты друг с другом, необходимо указывать индивидуальные имена.

Описание структуры объекта строится как описание связей конкретных компонент, каждая из которых имеет имя, тип и карты портов. Карта портов **port map** определяет соответствие портов компонент поступающим на них сигналам, можно интерпретировать карту портов как разъем, на который приходят сигналы и в который вставляется объект-компонента.

Принятая в VHDL форма описания связей конкретных компонент имеет следующий вид:

Имя: тип связи (сигнал, порт).

Например, описание связей объекта Q1, представленного на рис. 3 выглядит следующим образом:

К1: SM port map (X1, X2, S);

К3: M port map (S, Y1);

К2: SM port map (S, X3, Y2);

Здесь К1, К2, К3- имена компонент; SM,M- типы компонент; X1, X2, X3, S, Y1,Y2- имена сигналов, связанных с портами.

Полное VHDL описание архитектуры STRUCTURA объекта Q1 имеет вид:

```
Architecture STRUCTURA of Q1 is
Component SM port (A, B: in real; C: out real);
End component;
Component M port (E: in real; D: out real);
End component;
```

```
Signal S: real;
Begin
K1: SM port map (X1, X2, S);
K3: M port map (S, Y1);
K2: SM port map (S, X3, Y2);
End STRUCTURA;
```

Средства VHDL для отображения поведения описываемых архитектур строится на представлении их как совокупности параллельно взаимодействующих процессов. Понятие процесса **process** относится к базовым понятиям языка VHDL.

Архитектура включает в себя описание одного или нескольких параллельных процессов. Описание процесса состоит из последовательности операторов, отображающих действия по переработке информации. Все операторы внутри процесса выполняются последовательно. Процесс может находиться в одном из двух состояний- либо пассивном, когда процесс ожидает прихода сигналов запуска или наступления соответствующего момента времени, либо активном- когда процесс исполняется.

Процессы взаимодействуют путем обмена сигналами.

В общем случае в поведенческом описании состав процессов не обязательно соответствует составу компонент, как это имеет место в структурном описании

Поведение VHDL- объектов воспроизводится на ЭВМ, и приходится учитывать особенности воспроизведения параллельных процессов на однопроцессорной ЭВМ. Особая роль в синхронизации процессов отводится механизму событийного воспроизведения модельного времени **now.**

Когда процесс вырабатывает новое значение сигнала перед его посылкой на линию связи, говорят, что он вырабатывает будущее сообщение **transaction**. С каждой линией связи (сигналом) может быть связано множество будущих сообщений. Множество сообщений для сигнала называется его драйвером **driver**.

Т.о., драйвер сигнала - это множество пар: время – значение (множество планируемых событий).

VHDL реализует механизм воспроизведения модельного времени, состоящий из циклов. На первой стадии цикла вырабатываются новые значения сигналов. На второй стадии процессы реагируют на изменения сигналов и переходят в активную фазу. Эта стадия завершается, когда все процессы перейдут снова в состояние ожидания. После этого модельное время становится равным времени ближайшего запланированного события, и все повторяется.

Особый случай представляет ситуация, когда в процессах отсутствуют операторы задержки. Для этого в VHDL предусмотрен механизм так называемой дельта - задержки.

В случае дельта – задержек новый цикл моделирования не связан с увеличением модельного времени. В приведенном выше примере новое значение сигнала У1 вырабатывается через дельта- задержку после изменения сигнала S.

Другая способность VHDL- процессов связана с так называемыми разрешенными **resolved** сигналами. Если несколько процессов изменяют один и тот же сигнал, (сигнал имеет несколько драйверов), в описании объектов может указываться функция разрешения. Эта функция объединяет значения из разных драйверов и вырабатывает одно. Это позволяет, например, особенности работы нескольких элементов на общую шину.

В языке VHDL для наиболее часто используемых видов процессов – процессов межрегистровых передач – введена компактная форма записи.

Полное описание архитектуры POVEDENIE объекта Q1 в этом случае имеет следующий вид:

```
Architecture POVEDENIE of Q1 is
Signal S: real;
```

```
Begin
Y1<=S;
Y2<=S+X3 after 10 ns;
S<=X3+X2 after 10 ns;
End POVEDENIE;
```

Описание простого объекта.

Для иллюстрации возможностей VHDL рассмотрим пример проектирования простой комбинационной схемы, назовем ее объект F.Объект проекта F имеет два входа A1 и A2 и два выхода B1 и B2.

Объявление объекта проекта F.

Entity F is

Port (A1, A2: in BIT; B1, B2: out BIT)

Сигналы принимают значения 1 или 0 в соответствии с таблицей истинности.

Входы		Выходы	
A1	A2	B1	B2
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

Поведенческое описание архитектуры

Вариант описания архитектуры BEHAVIOR объекта F использует условный оператор **if** языка VHDL и учитывает, что только при обоих входах A1 и A2, равных 1, выходы B1=1 и B2=0. В остальных случаях наоборот B1=0 и B2=1

```
Architecture BEHAVIOR of F is
Begin
Process
Begin
Wait on (A1, A2)
If (A1='1') and (A2='1')
Then B1<='1'; B2<='0';
End if;
End process;
End;
```

В каждом процессе может быть только 1 оператор **wait on.** Второй вариант поведенческого описания архитектуры объекта F, назовем его BEHAVIOR_F, использует выбор **case** языка VHDL и учитывает то свойство функции F, что для первых трех строк ее значение не меняется. В заголовке процесса указан список чувствительности процесса **process (A1, A2).** Это указание эквивалентно оператору **wait on (A1, A2)** в начале описания процесса.

```
Architecture BEHAVIOR_F of F is
Begin
Process (A1,A2);
Begin
--&-операция
```

```
case (A1& A2) is
--первые три строки таблицы
when “00”/ “01”/ “10”=> B1<=’0’; B2<=’1’
--последняя строка таблицы
when “11” => B1<=‘1’; B2<=’0’
end case
end process
end BEHAVIOR_F;
```

Потоковая форма

В процессе проектирования объекта F могут быть предложены различные варианты его функциональных схем:

Описание архитектуры объекта F может быть таким:

```
Architecture F_A of F is
Begin
--каждому вентилю сопоставлен оператор назначения сигнала
B1<= A1 and A2;
B2<= not (A1 and A2);
End;
```

Здесь каждому элементу сопоставлен процесс, отображающий последовательность преобразования входной информации и передачи ее на выход. Процесс представлен в форме оператора параллельного назначения сигнала. Операторы назначения сигнала**(<=)** срабатывают параллельно при изменении хотя бы одного из сигналов в своих правых частях.

Другой вариант описания архитектуры F_B. Здесь вентили включены последовательно.

```
Architecture F_B of F is
Signal X: bit
Begin
B2<= not (X);
X <= A1 and A2;
B1 <= X;
End;
```

Промежуточный сигнал X введен в описание архитектуры F_B объекта F потому, что в описании интерфейса объекта F порт B1 объявлен выходным, то есть с него нельзя считывать сигнал и запись B2<= not(B1) была бы не корректной.

Сигнал B2 вырабатывается только после изменения сигнала X. Оператор B2<= not(X) сработает только тогда, когда изменится сигнал X, то есть после оператора X<= A1 and A2, т. к. он реагирует только на изменение сигнала в своей правой части. С учетом задержки E1=10 нс., а E2=5нс описание архитектуры будет иметь вид

```
Architecture F_B_TIME of F is
Signal X: bit
Begin
--задержка на B1- 10 нс.
--задержка на B2- 5 нс.
B1<=X;
B2<= not (X) after 5 ns;
X<= A1 and A2 after 10 ns;
End;
```

Через 10 нс. после изменения одного из входных сигналов (A1 или A2) может измениться выходной сигнал B1, и с задержкой 5 нс. после него изменится B2.

Структурное описание архитектуры.

Описание архитектуры представляет собой структуру объекта как композицию

компонент, соединенных между собой и обменивающихся сигналами. Функции, реализуемые компонентами в явном виде, в отличие от предыдущих примеров в структурном описании не указываются. Структурное описание включает описание интерфейсов компонент, из которых состоит схема и их связей. Полные (интерфейс + архитектура) описания объектов- компонент должны быть ранее помещены в проектируемую библиотеку, подключенную к структурному описанию архитектуры.

```
Library work;
use work.all
--подключение рабочей библиотеки work, содержащей описание объекта соответствующего компоненте INE2
Architecture CXEM_F_C of F is
--ниже интерфейсы компоненты INE2
Component INE2
Port (X1, X2: in bit; Y: out bit);
End component;
--ниже описание связей экземпляров компонент
Signal X: bit;
Begin
E1: INE2 port map (A1, A2, X);
E2: INE2 port map (X, X, B1);
B2<= X;
End;
```

В описании архитектуры CXEM_F_C объекта F сначала указан интерфейс компонент, из которых строится схема. Это компоненты типа INE2 с двумя входными и одним выходным портом. Затем после *begin* идут операторы конкретизации компонент. Для каждого экземпляра компоненты следуют: ее имя или карта портов, указывающая соответствие портов экземпляра компоненты поступающим на них сигналам. Например, для компоненты по имени E1 типа INE2 на порт X1 подан сигнал A1, на порт X2- сигнал A2. Порядок конкретизации безразличен, так как это параллельные операторы. Для того чтобы описание F было полным, в данном случае в рабочей библиотеке проекта work необходимо иметь описание интерфейса и архитектуры некоторого объекта, сопоставляемого компоненте INE2. Обозначим этот объект в библиотеке как LA3. Его описание:

```
Entity LA3 is
Port (X, Y: in bit; Z: out bit);
End LA3;
Architecture DF_LA3 of LA3 is
Begin
Z<= not (X and Y) after 10ns;
End;
```

У объекта LA3 может быть насколько архитектур. В примерах дан вариант потокового описания архитектуры DF_LA3 объекта LA3, который содержит оператор назначения сигналу Z инверсного значения конъюнкции сигналов X и Y с задержкой 10 нс.

Описание конфигурации.

Конфигурацию можно рассматривать как аналог списка компонентов для проекта. Оператор конфигурации (идентифицируемый ключевым словом “configuration”), определяет, какие реализации должны быть использованы, и позволяет изменять связи компонентов в вашем проекте во время моделирования и синтеза.

Конфигурации не являются обязательными, независимо от того, насколько сложен описываемый проект. При отсутствии конфигурации, стандарт VHDL

определяет набор правил, который обеспечивает конфигурацию по умолчанию; например, в случае, когда предусмотрено более одной реализации для блока, последняя скомпилированная реализация получит приоритет, и будет связана с объектом.

Обозначение типа компоненты в описании архитектуры CXEM_F_C объекта F и обозначение соответствующего объекта проекта в библиотеке могут не соответствовать друг другу. Связывание обозначений осуществляется в форме объявления конфигурации. Для того чтобы задать информацию о том, что использованная при описании архитектуры CXEM_F_C объекта F компонента INE2соответствует библиотечному объекту LA3 и варианту его архитектуры под названием DF_LA3, необходимо объявить конфигурацию **configuration**. Конфигурация V1 указывает, что из рабочей библиотеки проекта **library work** для архитектуры CXEM_F_C объекта F для компонент с именами E1 и E2 типа INE2 следует использовать архитектуру DF_LA3 объекта LA3.

```
Library WORK
--подключается рабочая библиотека проекта
configuration V1 of F is
--конфигурация по имени V1 объекта F
use WORK. all;
--используются все (all) компоненты библиотеки WORK
for CXEM_F_C
--для архитектуры CXEM_F_C
--компоненты E1, E2 соответствуют объекту LA3 с архитектурой DF_LA3 из библиотеки WORK
for E1,E2: INE2
use entity LA3 (DF_LA3);
end for;
end for;
end V1;
```

Векторные сигналы и регулярные структуры.

Одним из средств повышения компактности описаний цифровых устройств является использование векторных представлений сигналов и операций над ними. Например, пусть некоторый объект FV выполняет туже функцию, что и объект F, но над 20- разрядными двоичными векторами AV1 и AV2.

Его описание определяет порты как двоичные векторы:

```
Entity FV is
Port (AV1, AV2: in bit_vector (1 to 20);
BV1, BV2: out bit_vector (1 to 20));
End FV1;
```

Поведенческое описание архитектуры FV в потоковой форме использует операции над битовыми векторами.

```
Architecture BECHAV_FV of FV is
Begin
BV2<= not (AV1 and AV2);
BV1<= AV1 and AV2;
End BECHAV_FV;
```

Структурное описание архитектуры FV для варианта реализации объекта FV как совокупности объектов F, представленное ниже, выполнено с использованием оператора генерации конкретизации. Это позволяет повысить компактность описаний регулярных фрагментов схем.

```
Architecture STRUCT_FV of FV is
Component F port (X1, X2: in bit; Y1, Y2: out bit);
```

End component;
Begin
--первая компонента конкретизирована обычным способом с использованием позиционного соответствия сигналов портам
K1: F port map (AV1 (1), AV2 (1), BV1 (1), BV2 (1));
--вторая компонента конкретизирована с использованием ключевого способа указания соответствия сигналов ее портам
K2: F port map (AV1 (2)=>X1, BV1 (2)=>Y1, AV2 (2)=>X2, BV2 (2) =>Y2);
--компоненты К3 - К20 конкретизированы с использованием оператора генерации, позволяющего компактно описывать регулярные фрагменты схем
for I in 3 to 20 generate
K(I): F port map (AV1 (1), AV2 (1), BV1 (1), BV2 (1));
End generate;
End STRUCT_FV;

Задержки сигналов и параметры настройки.

Объект с задержкой можно представить как бы состоящим из двух- идеального элемента и элемента задержки.

В языке VHDL встроены две модели задержек – инерциальная и транспортная.

Инерциальная модель предполагает, что элемент не реагирует на сигналы, длительность которых меньше порога, равного времени задержки элемента. Транспортная модель лишена этого ограничения.

Инерциальная модель по умолчанию встроена в оператор назначения сигнала языка VHDL. Например, оператор назначения *Y<=X1 and X2 after 10 ns;* описывает работу вентиля 2И и соответствует инерциальной модели. Указание на использование транспортной модели обеспечивается ключевым словом **transport** в правой части оператора назначения. Например, оператор *YT<=transport X1 and X2 after 10 ns;* отображает транспортную модель задержки вентиля.

Задержка может быть задана не константой, а выражением, значение которого может конкретизироваться для каждого экземпляра объекта, используемого как компонента. Для этого ее следует задать как параметр настройки в описании интерфейса объекта.

Приведенное ниже описание объекта 12 включает описание интерфейса и тела 12 с инерциальной задержкой, заданной как параметр настройки

Entity 12 is
--параметр настройки T по умолчанию равен 10 нс
Generic (T: time = 10 ns);
Port (X1, X2: in bit; Y: out bit);
End 12;
Architecture A1_inert of 12 is
Begin
Y<= X1 and X2 after T;
End A1_inert;
Architecture A1_transport of 12 is
Begin
Y<= transport X1and X2 after 10 ns;
End;

Ниже представлен вариант описания архитектуры иллюстрирующей возможность использования параметра настройки (задержка E1 равна 5 нс., E2- 20 нс.) и возможность совмещения структурного и поведенческого описаний в одной архитектуре:

Architecture MIX_8_a of F is
Component 12

```
Generic (T: time);
Port (X1, X2: in bit; Y: out bit );
End component;
Begin
E1: 12 generic map (5 ns );
Port map ( A1, A2, B1);
E2: B2<= not ( A1 and A2 ) after 20 ns;
End;
```

Более сложной представляется ситуация, когда необходимо отобразить в описании архитектуры объекта тот факт, что задержки фронта и среза сигналов не совпадают или зависят от путей прохождения сигналов в схеме и ее предыдущего состояния.

Одним из вариантов описания инерциального поведения вентиля 2И с разными задержками фронта и среза может быть следующим:

```
Architecture INERT of 12 is
Begin
Process ( X1, X2 );
Variable Z: bit;
Begin
--выход идеального вентиля
Z=X1 and X2;
if Z=’ 1’ and Z’DELAYED=’ 0’ then
--срез
Y<=’ 0’ after 3 ns
End if;
End process;
End;
```

Атрибут Z’DELAYED дает предыдущее значение сигнала.

При описании более сложных ситуаций следует учитывать особенности реализации механизма учета задержек сигналов в VHDL. С сигналом ассоциируется драйвер- множество сообщений о планируемых событиях в форме пар время-значение сигнала.

В случае транспортной задержки, если новое сообщение имеет время большее, чем все ранее запланированные, оно включается в драйвер последним. В противном случае предварительно уничтожаются все сообщения, запланированные на большее время.

В случае инерциальной задержки также происходит уничтожение всех сообщений, запланированных на большее время. Однако разница в том, что происходит анализ событий, запланированных на меньшее время, и если значение сигнала отличается от нового , то они уничтожаются.

Атрибуты сигналов и контроль запрещенных ситуаций.

Описания систем могут содержать информацию о запрещенных ситуациях, например, недопустимых комбинациях сигналов на входах объектов, рекомендуемых длительностях или частотах импульсов и т. п. Например, в вентиле 2И возникает риск сбоя в ситуациях, когда фронт одного сигнала перекрывает срез другого.

Входные сигналы X1 и X2 изменяются в противоположном направлении, и время их перекрытия меньше необходимого, допустим равного 10 нс.

Средством отображения информации о запрещенных ситуациях в языке VHDL является оператор утверждения (оператор контроля, оператор аномалии) **assert**. В нем помимо контролируемого условия, которое не должно быть нарушено, т. е. должно быть истинным, записывается сообщение **report** о нарушении и уровень серьезности ошибки **severity**.

Для приведенного примера в описании архитектуры вентиля 2И может быть вставлено утверждение о том, что все будет нормально, если внутренний сигнал Z будет равен 1 не менее чем 10 нс., иначе идет сообщение об ошибке.

Для этого необходимо, чтобы в момент среза сигнала его задержанное на 10 нс. значение было равно 1. Если оно равно 0, то длительность сигнала меньше 10 нс.

Время предыдущего события в сигнале Z можно получить атрибутом LAST_EVENT.

```
Architecture C1 of 12 is
Signal Z: bit =’0’;
Begin
Process ( X1, X2 );
Z<= X1 and X2;
Assert not (Z=’0’ and not Z’STABLE and Z’DELAYED (10 ns)= ‘0’)
Report ”риск сбоя в 1 в вентиле 12”
Severity worning;
Y<= transport Z after 10 ns;
End C1;
```

Более полное представление о предопределенных атрибутах сигналов можно получить из таблицы 4.1 Помимо предопределенных, пользователь может вводить дополнительные атрибуты для сигналов и других типов данных.

Таблица 4.1. Предопределенные атрибуты сигналов

Пример	Тип результата	Пояснения
S’QUIET(T)	Boolean	TRUE, если сигнал S пассивен на интервале T.
S’TRANSACTION	Bit	Инвертируется S каждый раз, когда S активен (изменяется)
S’STABLE(T)	Boolean	TRUE, если не было событий за интервал T.
S’DELAYED(T)	Signal	Предыдущее значение S в момент NOW-T
S’ACTIVE	Boolean	TRUE, если сигнал активен
S’LAST_ACTIVE	Time	Время, когда сигнал последний раз был активен

S'EVENT	Boolean	TRUE, если происходит событие в S
S'LAST_VALUE	Signal	Значение сигнала передпоследним событием в нем
S'LAST_EVENT	Time	Время последнего события в S

Алфавит моделирования и пакеты.

Описание пакета VHDL задается ключевым словом **package** и используется, чтобы собирать часто используемые элементы конструкции для использования глобально в других проектах. Пакет можно рассматривать как общую область хранения, используемую, чтобы хранить такие вещи как описания типов, констант, и глобальные подпрограммы. Объекты, определенные в пределах пакета можно использовать в любом другом проекте на VHDL, и можно откомпилировать в библиотеки для дальнейшего повторного использования.

Пакет может состоять из двух основных частей: описания пакета и дополнительного тела пакета. Описание пакета может содержать следующие элементы:

- Объявления типов и подтипов
- Объявления констант
- Глобальные описания сигналов
- Объявления процедур и функций
- Спецификация атрибутов
- Объявления файлов
- Объявления компонентов
- Объявления псевдонимов
- операторы включения

Пункты, появляющиеся в пределах описания пакета могут стать видимыми в других проектах с помощью оператора включения.

Если пакет содержит описания подпрограмм (функций или процедур) или определяет одну или более задерживаемых констант (константы, чья величина не задана), то в дополнение к описанию необходимо тело пакета. Тело пакета (которое определяется с использованием комбинации ключевых слов "package body"), должно иметь то же имя как соответствующее описание пакета, но может располагаться в любом месте проекта (оно не обязано располагаться немедленно после описания пакета).

Отношение между описанием и телом пакета отчасти напоминает отношение между описанием и реализацией элемента (тем не менее, может быть только одно тело пакета для каждого описания пакета). В то время как описание пакета обеспечивает информацию, необходимую для использования элементов, определенных в пределах этого пакета(список параметров для глобальной процедуры, или имя определенного типа или подтипа), фактическое поведение таких объектов, как процедуры и функции, должно определяться в пределах тела пакета.

Приведенные выше описания объекта F базировались на стандартных средствах языка VHDL- сигналы представлялись в алфавите ' 1', ' 0 ', логические операции И, ИЛИ, НЕ также определялись в этом алфавите. Во многих случаях

приходиться описывать поведение объектов в других алфавитах. Например, в реальных схемах сигнал кроме значений ‘1’ и ‘0’ может принимать значение высокого импеданса ‘Z’ (на выходе буферных элементов) и неопределенное значение ‘X’, например, отражая неизвестное начальное состояние триггеров.

Переход к другим алфавитам осуществляется в VHDL с помощью пакетов. У пакета, как и у объекта проекта, различают объявление интерфейса **package** и объявление тела объекта **package body**.

Ниже приводиться пример пакета P4 для описания объектов в четырехзначном алфавите представления сигналов (‘X’, ‘0’, ‘1’, ‘Z’), X- значение не определено, Z- высокий импеданс. В этом пакете приводится тип KONTAKT для представления сигналов в четырехзначном алфавите, и определяются функции NOT и AND над ними.

В ТТЛ логике высокий импеданс на входе воспринимается как 1, что учитывается в таблице функции AND.

Ниже следует объявление пакета P4:

```
Package P4 is
--перечислимый тип
type KONTAKT is (‘X’, ‘0’, ‘1’, ‘Z’ );
function “NOT” (X: in KONTAKT) return KONTAKT;
function “AND” (X1, X2: in KONTAKT) return KONTAKT;
end P4;
```

Ниже следует объявление тела пакета P4:

```
Package body P4 is
Function “NOT” (A: in KONTAKT) return KONTAKT is
Begin
If A=’X’ then return ‘X’
Else if A=’1’ then return ‘0’
Else if A=’0’ then return ‘1’
Else return ‘Z’
End if;
End “NOT”;
Function “AND” ( A1, A2: in KONTAKT ) return KONTAKT is
Begin
If (A1=’0’ ) or (A2=’X’ ) then return to ‘0’
Else if (A1= ‘X’ ) or (A2=’0’ ) or (A2= ’X’) and (A1=’0’ ) then return ‘X’
Else return to’1’
End if;
End “AND”;
End P4;
```

Пример использования пакета P4 при описании объекта F_P4. Этот объект отличается от F, т. к. у него другой интерфейс.

```
--подключается ( use) пакет P4, все его функции (ALL)
use P4.ALL;
entity F_P4 is
port (A1, A2: in KONTAKT; B1, B2: out KONTAKT )
end F_P4;
Описание архитектуры F_P4_a
Architecture F_P4_a of F is
Begin
B2<= not ( A1 and A2 );
B1<= A1 and A2;
End F_P4_a;
```

Из этого примера видно, что в ряде случаев изменение алфавита моделирования не требует внесения изменений в описания объектов.

Например, переход к семизначному алфавиту (‘0’, ‘1’, ‘X’, ‘Z’, ‘F’, ‘S’, ‘R’), где тип KONTAKT имеет дополнительные значения: F- фронт, S- срез, R- риск сбоя, потребует только создания нового пакета и подключения его к объявлению объекта F_P4. Изменение в других частях описаний объекта проекта не потребуется.

Описание монтажного “или” и общей шины.

В цифровой аппаратуре используются монтажные ИЛИ (И) и двунаправленные шины на элементах с тремя состояниями выхода.

Монтажное ИЛИ:

```
Signal Y: WIRED_OR bit;
K1: process
Begin
Y<= X1 and X2
End process K1;
K2: process
Y<= X3 and X4;
End process K2;
```

Общая шина на элементах с трех стабильными выходами:

```
Type A3 is (‘0’, ‘1’, ‘Z’);
Signal D: CHIN A3;
C1: process
Begin
If E1=’0’ then D<=’ Z’
Else D<= X1 and X2
End if;
End process C1;
C2: process
If E2=’0’ then D<=’Z’
Else D<= X3 and X4
End process C2;
```

Если каждой компоненте (К1, К2) схемы сопоставить процесс, то имеем два параллельных процесса, каждый из которых вырабатывает свой выходной сигнал.

В языке VHDL предусмотрен механизм разрешения конфликтов, возможных в подобных ситуациях, когда сигнал имеет несколько драйверов. Функция разрешения обычно описывается в пакете, а ее имя указывается при описании соответствующего сигнала. Например, тело функции разрешения WIRED_OR (монтажное ИЛИ) имеет следующий вид:

```
Function WIRED_OR (INPUTS: bit_vector) return bit is
Begin
For I in INPUTS’RANGE loop
If INPUTS( I ) = ‘1’ then
Return ‘1’ ;
End if;
End loop;
Return ‘0’;
End;
```

Драйверы сигнала INPUTS неявно рассматриваются как массив, границы которого определяются атрибутом ‘RANGE.

Функция сканирует драйверы сигнала и, если хоть один из них равен ‘1’, возвращает значение ‘1’, иначе ‘0’.

Функция разрешения SHIN для шины на элементах с тремя с тремя

состояниями выходами может быть такой:

```
Type A3 is (‘0’, ‘1’, ‘Z’);
Type VA3 is array ( integer range <> of A3 );
Function SHIN (signal X: VA3 ) return A3 is
Variable VIXOD: A3:= ‘Z’;
Begin
For I in X’RANGE loop
I f X( I) /= ‘Z’ then
VIXOD:= X ( I);
Exit;
End if;
End loop;
Return VIXOD;
End SHIN;
```

Предполагается, что может быть включен, (то есть, не равен ‘Z’) только один из драйверов входных сигналов.

Организации, поддерживающие развитие vhdl

Министерство обороны США в начале 80-х годов финансировало разработку многоуровнего языка VHDL, стандартизовало его и обязало своих поставщиков цифровых микросхем представлять в составе документации их описание на VHDL. Это можно рассматривать как важный, но только первый шаг к обязательности формальных моделей для всех видов выпускаемой электронной техники. В связи с возлагаемой на VHDL особой ролью, интерес к нему в США и в Европе огромен, созданы Американская и Европейская группы, занимающиеся всем комплексом вопросов, связанных с внедрением VHDL, как то:

- уточнение семантики языка,
- разработка методологии описания различных классов ЦУ,
- разработка внутренних форматов представления VHDL-моделей в САПР для обеспечения совместимости разрабатываемых продуктов,
- создание анализаторов, позволяющих контролировать синтаксис и семантику VHDL-моделей,
- создание справочно-обучающих систем и резидентных справочников по VHDL, позволяющих писать VHDL- модели под управлением и контролем системы ,
- создание мощных систем моделирования, использующих в качестве входного VHDL.

Спонсорами работ по развитию VHDL являются: Air Force Wright Aeronautical Laboratories, Avionics Laboratory, Air Force Systems Command, United States Air Force, Wright-Patterson Air Force Base , Ohio 45433.

В России работы по языку VHDL поддерживаются Российским научно-исследовательским институтом информационных систем (РосНИИИС), Московским институтом электронного машиностроения (кафедра "Специализированные вычислительные комплексы" МИЭМ), Томским политехниеским университом (кафедра"Вычислительной техники"), Международный центр по информатике и электронике, НИИ "Квант", Ассоциация заинтересованных в применении VHDL.

Самостоятельная работа №2. Функциональные особенности и состав тестовой платы Xilinx Spartan 3 Starter Kit.

Большинство производителей FPGA-микросхем предлагают их на рынке не только в виде отдельных чипов, но и в составе специально разработанных плат. Каждая такая плата, помимо собственно FPGA, может содержать несколько дополнительных микросхем (например, микросхемы памяти), стандартные разъемы для подсоединения внешних устройств, специальный разъем (или разъемы) для программирования FPGA-микросхемы, осциллятор, генерирующий синхросигнал. Все это позволяет разрабатывать на базе такой платы цифровые устройства, что называется «без паяльника», занимаясь только программированием FPGA-логики.

На рынке сейчас предлагается множество FPGA-плат, различающихся как типом FPGA-микросхемы, так и количеством окружающих ее на плате дополнительных устройств. В этой главе мы рассмотрим одно из простейших, но, тем не менее, достаточно мощных решений, предлагаемое фирмой Xilinx – плату Xilinx Spartan 3 Starter Kit (S3SK).

Основным элементом S3SK является FPGA-микросхема – Xilinx Spartan 3 XC3S200FT256 (рис. 1, маркер 1). Эта микросхема содержит 200 000 элементарных логических вентилей, двенадцать блоков памяти по 18 Кбит каждый, двенадцать 18-разрядных умножителей, четыре блока DCM (Digital Clock Manager), 173 перепрограммирумых на различные стандарты портов ввода-вывода. Помимо этого основного элемента плата S3SK содержит:

- микросхему энергонезависимой FLASH-памяти объемом 2 Мбита – Xilinx XCF02S; эта память подсоединена к FPGA и используется обычно для хранения bitstream, которым FPGA программируется, после выключения питания платы (маркер 2);
- две микросхемы SRAM-памяти – ISSI IS61LV25616AL-10T – это быстрая асинхронная 10-наносекундная SRAM; каждая микросхема содержит по 256К 16-разрядных ячеек памяти (маркер 4);
- один 3-разрядный VGA порт (маркер 5);
- 9-штырьковый RS232 последовательный порт (маркер 6);
- Maxim MAX3232 RS232 преобразователь (маркер 7);
- PS/2 порт для подключения «мыши» или клавиатуры (маркер 9);
- четырехпозиционный 7-сегментный LED дисплей (маркер 10);
- восемь двухпозиционных переключателей (маркер 11);
- восемь светодиодов для индикации (маркер 12);
- четыре кнопочных переключателя (маркер 13);
- 50-мегагерцовый кристаллический осциллятор, подсоединенный к FPGA как генератор синхросигнала (маркер 14);
- разъем для подключения внешнего генератора синхросигнала (маркер 15);
- кнопочный переключатель, для запуска реконфигурации FPGA (маркер 16);
- светодиод (LED), индицирующий успешную инициализацию FPGA (маркер 17);
- три 40-контактных разъема для подключения дополнительных устройств, совместимых с S3SK (маркеры 19,20,21);
- три порта для подключения различных вариантов JTAG кабелей (маркеры 22, 23, 24);
- AC-разъем для подключения стандартного источника питания +5 Вольт (маркер 25);
- светодиод, индицирующий наличие питания на плате (маркер 26);
- три регулятора напряжения – 3.3 В, 2.5 В, 1.2 В (маркеры 27, 28, 29).

На рис. 1 схематично изображено положение перечисленных компонентов на плате S3SK. Номера маркеров в кружочках соответствуют номерам в списке.

Разберемся с тем, как перечисленные выше компоненты подключены к микросхеме

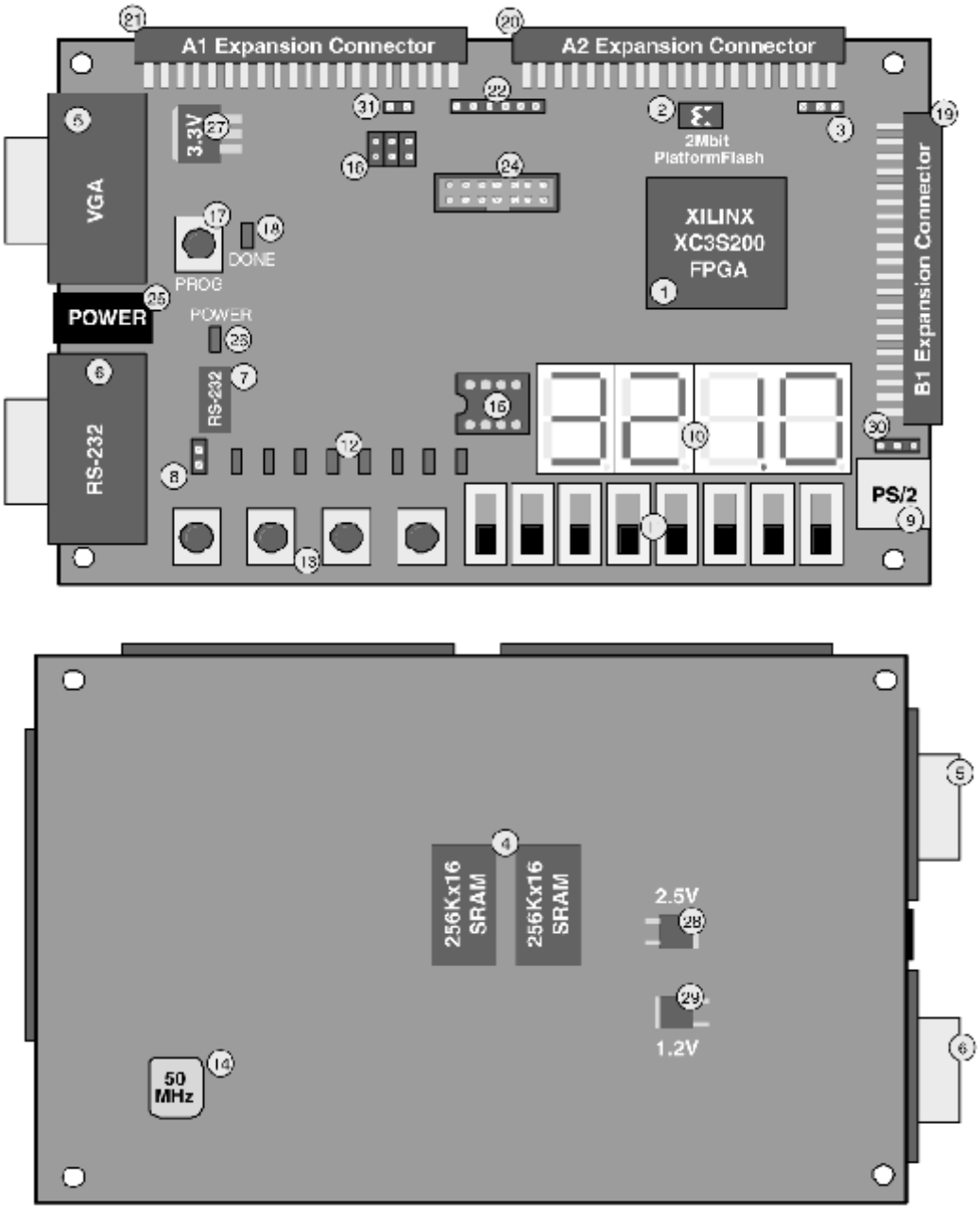

Рис. 1. Схематичное изображение расположения компонентов на лицевой (сверху) и оборотной (снизу) сторонах платы S3SK

XC3S200FT256.

Начнем с быстрой асинхронной памяти. Быстрая память представлена двумя микросхемами, расположенными на оборотной стороне платы S3SK. Каждая микросхема представляет собой массив из 256 К 16-разрядных ячеек. Обе микросхемы памяти объединены общими сигналами разрешения записи (Write Enable, WE), разрешения чтения (Output Enable, OE) и общей шиной адреса (A[17:0]) (рис. 2). При этом каждая микросхема имеет отдельный сигнал выбора микросхемы (Chip Enable, CE), выбора старшего (UB) или младшего (LB) байта 16-разрядной ячейки памяти.

Контакты XC3S200FT256, подключенные к выводам микросхем памяти, приведены

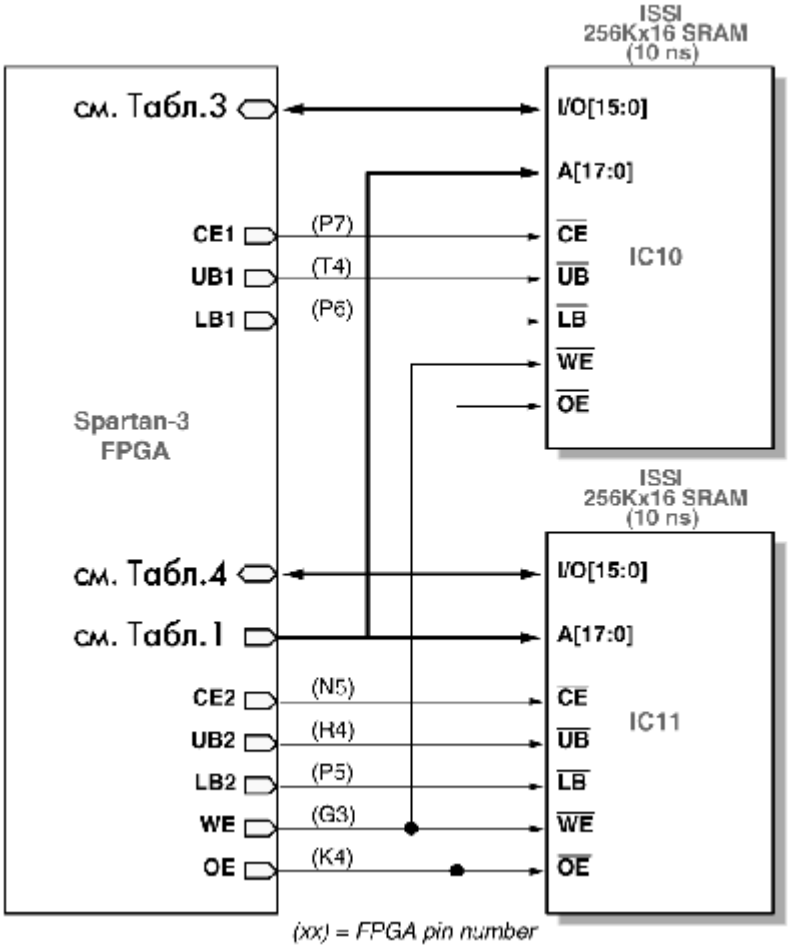

Рис. 2. Подключение быстрой асинхронной памяти к XC3S200FT256

в таблицах 1, 2, 3, 4. В этих же таблицах приведены контакты разъема расширения A1 (маркер 21 на рис. 1). При таком подключении, память можно использовать несколькими способами. Во-первых, если оба сигнала CE1 и CE2 установлены в низкий уровень, то доступны обе микросхемы IC10 и IC11 и память представляет собой массив 32-разрядных слов длиной 256 К. Во-вторых, установка одного из сигналов CE1 или CE2 в высокий уровень деактивирует соответствующую микросхему. В результате память представляет собой массив 16-разрядных слов длиной 512 К. В-третьих, использование сигналов UB1, LB1, UB2, LB2 позволяет адресоваться к старшим и младшим байтам в каждом 16-разрядном слове каждой микросхемы памяти. В результате память представляет собой массив 8-разрядных ячеек (байт) длиной 1 М.

Таблица 1

Разряд шины адреса	Контакт XC3S200FT256	Контакт разъема расширения A1
A17	L3	35
A16	K5	33
A15	K3	34
A14	J3	31
A13	J4	32
A12	H4	29
A11	H3	30
A10	G5	27
A9	E4	28
A8	E3	25
A7	F4	26
A6	F3	23
A5	G4	24
A4	L4	14
A3	M3	12
A2	M4	10
A1	N3	8
A0	L5	6

Таблица 2

Сигнал	Контакт XC3S200FT256	Контакт разъема расширения A1
OE	K4	16
WE	G3	18

Таблица 3.
Подключение шины данных микросхемы IC10 (рис. 2) к выводам FPGA

Сигнал	Контакт XC3S200FT256
IO15	R1
IO14	P1
IO13	L2
IO12	J2
IO11	H1
IO10	F2
IO9	P8
IO8	D3
IO7	B1
IO6	C1
IO5	C2
IO4	R5
IO3	T5
IO2	R6
IO1	T8
IO0	N7
CE1 (Chip Enable IC10)	P7
UB1 (Upper byte enable IC10)	T4
LB1 (Lower byte enable IC10)	P6

Таблица 4.
Подключение шины данных микросхемы IC11 (рис. 2) к выводам FPGA

Сигнал	Контакт XC3S200FT256
IO15	N1
IO14	M1
IO13	K2
IO12	C3
IO11	F5
IO10	G1
IO9	E2
IO8	D2
IO7	D1
IO6	E1
IO5	G2
IO4	J1
IO3	K1
IO2	M2
IO1	N2
IO0	P2
CE2 (Chip Enable IC11)	N5
UB2 (Upper byte enable IC11)	R4
LB2 (Lower byte enable IC11)	P5

Рассмотрим подключение следующего компонента на плате S3SK – 4-позиционного 7-сегментного жидкокристаллического дисплея. Для уменьшения количества сигналов, необходимых для управления дисплеем, все четыре позиции имеют общие управляющие выводы (рис. 3). В скобках на рисунке указаны выводы XC3S200FT256, подключенные к указанным контактам дисплея.

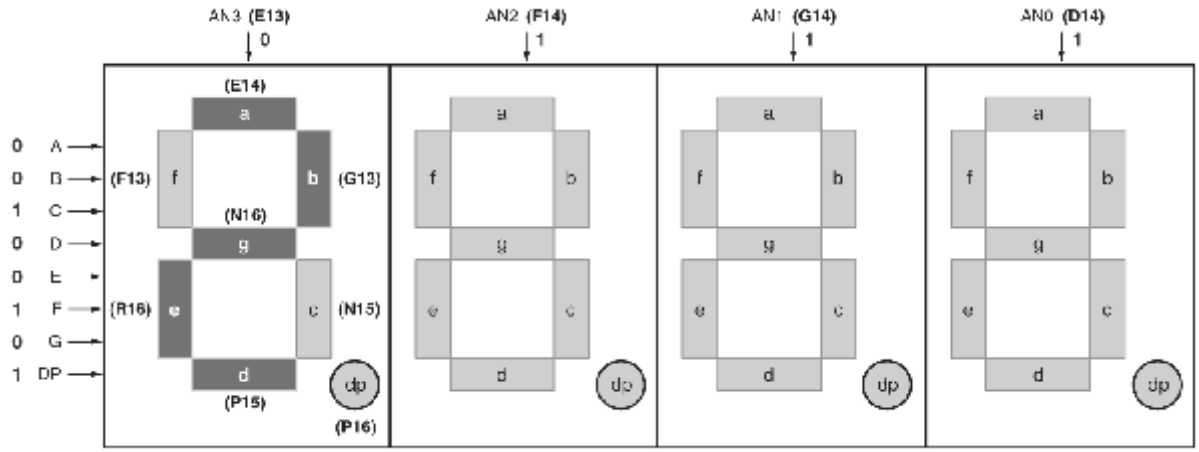

Рис. 3. Управляющие сигналы 4-позиционного 7-сегментного дисплея. Вектор (A,B,C,D,E,F,G,DP) определяет светящиеся сегменты на дисплее. Вектор (AN0,AN1,AN2,AN3) задает одну из четырех активных позиций (один из четырех разрядов дисплея). Активным считается нулевой уровень. На рисунке приведен пример управляющих сигналов, при которых в самой левой позиции на экране высвечивается цифра 2

Одновременный вывод символов во все четыре позиции экрана, таким образом, невозможен. Эффект одновременной работы всех четырех позиций достигается за счет инерционности человеческого глаза при быстром последовательном переключении управляющих сигналов AN0, AN1, AN2, AN3, с одновременной сменой данных на шине (A,B,C,D,E,F,G,DP) (рис. 4).

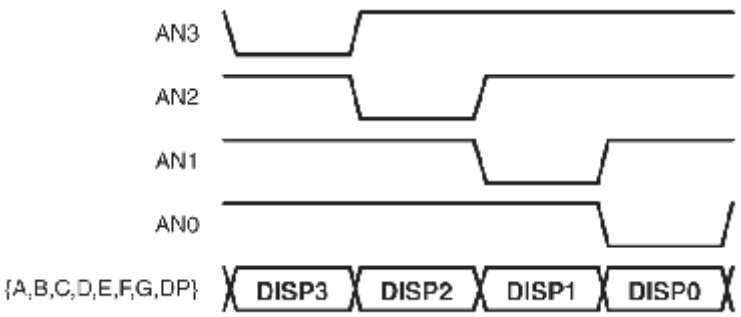

Рис. 4. Общий поток данных (A,B,C,D,E,F,G,DP), управляющих свечением сегментов, мультиплексируется сигналом (AN0,AN1,AN2,AN3)

Такой подход позволяет управлять выводом информации на дисплей, используя всего 12 управляющих сигналов (вместо 32, если каждый сегмент в каждой позиции управляется со своего собственного вывода).

В нижнем правом углу S3SK находится 8 переключателей, которые подключены к выводам XC3S200FT256, как показано в табл. 5. Все переключатели поименованы: с SW0 до SW7. Переключатель, названный SW0, занимает самую левую позицию, SW7 – самую правую (рис. 1).

Табл. 5

Переключатель	SW7	SW6	SW5	SW4	SW3	SW2	SW1	SW0
Контакт FPGA	K13	K14	J13	J14	H13	H14	G12	F12

Когда какой-либо переключатель находится во включенном состоянии, он подает на соответствующий вывод FPGA уровень логической единицы. В противном случае – нуль. Необходимо заметить, что скорость срабатывания переключателей, то есть время, за которое сигнал после переключения устанавливается в 0 или 1, составляет около 2 мс.

Левее переключателей (рис. 1, маркер 13) находятся четыре кнопочных выключателя. Нажатие на любую из четырех кнопок подает высокий уровень (логическая единица) на контакт FPGA, к которому кнопка подключена. Кнопки поименованы: BTN0 – BTN3. Самая левая кнопка – BTN0, самая правая – BTN3. Выводы FPGA, подключенные к кнопкам, приведены в табл. 6.

Табл. 6

Кнопка	BTN3	BTN2	BTN1	BTN0
Контакт FPGA	L14	L13	M14	M13

Над кнопками (рис. 1, маркер 12) расположены восемь светодиодов. Светодиоды поименованы: LD0 – LD7. Самый левый светодиод – LD0, самый правый – LD7. Контакты FPGA, подключенные к светодиодам, приведены в табл. 7. Для того чтобы «зажечь» тот или иной светодиод, на соответствующий контакт FPGA надо подать высокий уровень, то есть уровень логической единицы.

Плата S3SK содержит VGA-порт (рис. 1, маркер 5), подключенный к стандартному 15-штырьковому разъему DB15 (рис. 5). Это позволяет использовать стандартный кабель

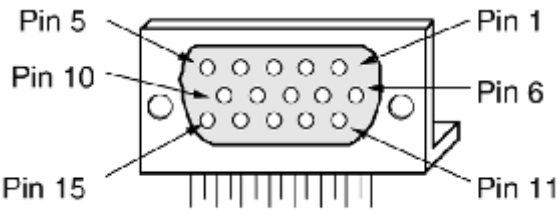

Рис. 5. VGA-разъем DB15 (вид спереди)

для подключения к S3SK монитора. Схема подключения разъема DB15 к контактам FPGA приведена на рис. 6 (в скобках указаны номера контактов FPGA).

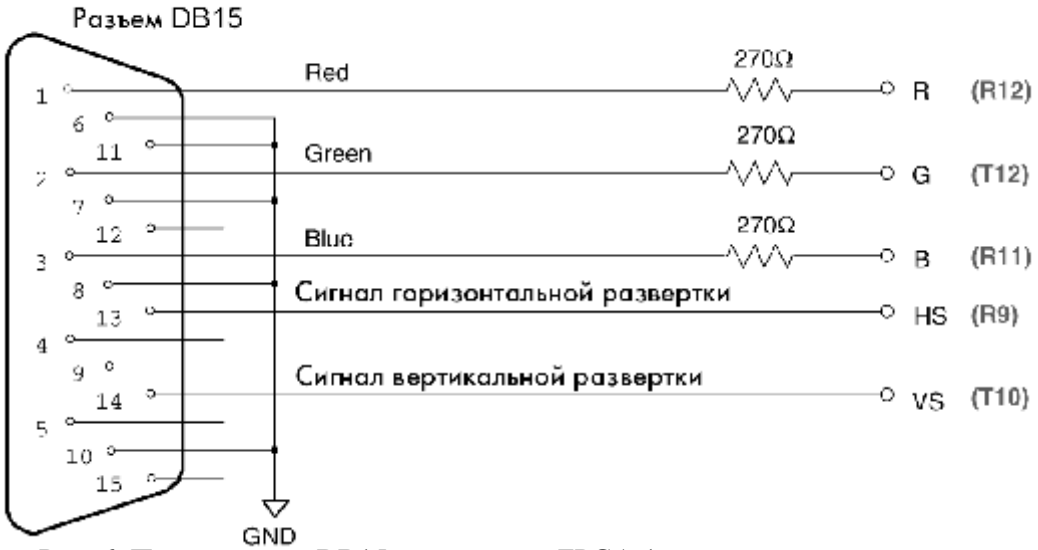

Рис. 6. Подключение DB15 к контактам FPGA (номера контактов указаны в скобках справа на рисунке)

Из рис. 6 ясно, что при таком подключении реализовано 3-разрядное представление цвета. Возможные варианты управляющих сигналов (R, G, B) и соответствующие им цвета представлены в табл. 7.

Табл. 7

Red (R)	Green (G)	Blue (B)	Результирующий цвет
0	0	0	Черный
1	0	0	Красный
1	1	0	Желтый
1	0	1	Пурпурный (Magenta)
0	1	0	Зеленый
0	0	1	Синий
0	1	1	Голубой (Cyan)
1	1	1	Белый

Разберемся теперь с сигналами горизонтальной и вертикальной развертки. В ЭЛТ-мониторах изображение на экране появляется в результате «засветки» электронным пучком покрытого люминесцентным слоем экрана. Положение электронного пучка контролируется отклоняющими напряжениями по горизонтали и по вертикали. Напряжение, контролирующее отклонение луча по горизонтали, линейно нарастает во времени до некоторого максимального значения. Луч в результате пробегает по экрану слева направо (рис. 7). Аналогично, но медленнее, меняется и напряжение, отклоняющее луч по вертикали. Скорость изменения отклоняющих напряжений контролируется

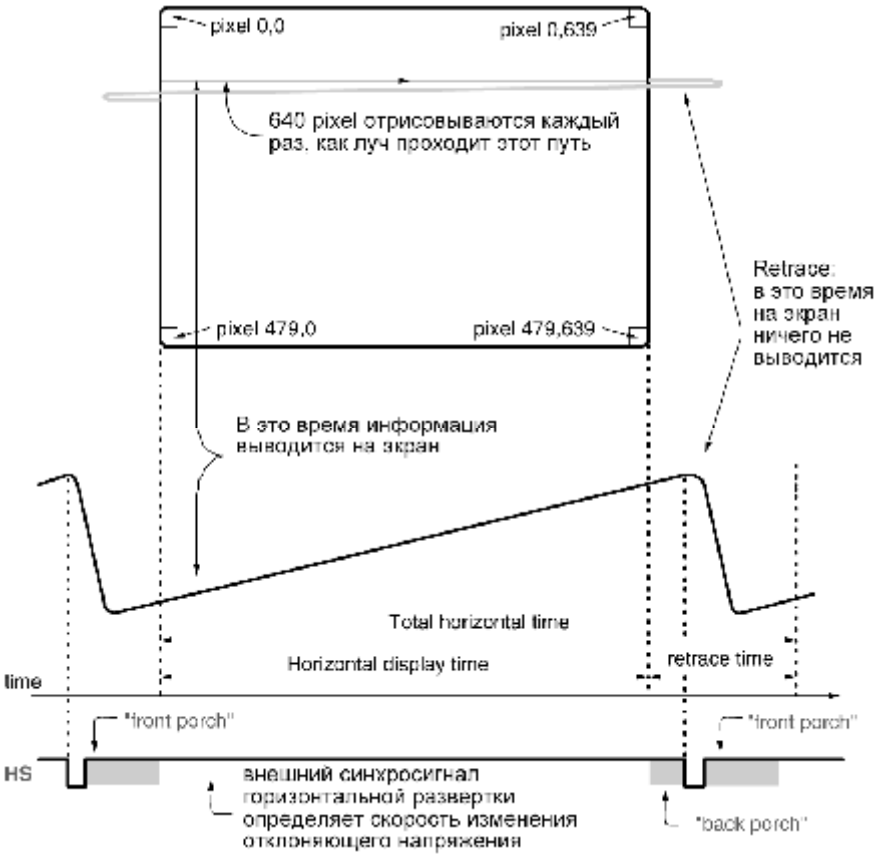

Рис. 7. Скорость изменения отклоняющих напряжений по вертикали и горизонтали задаются внешними синхросигналами. На рисунке представлен сигнал горизонтальной развертки HS (внизу), отклоняющее напряжение и траектория пучка

внешним синхросигналом.

В табл. 8 приведены соотношения времен при выводе на экран монитора информации в режиме 640x480 при частоте вертикальной развертки (обновлении экрана) 60 Гц. Частота, с которой при этом должен меняться сигнал RGB, составляет 25 МГц.

Обозначения иллюстрируются на рис. 8. Столбцы «Периодов RGB» содержат отношение времени в данной строке к периоду изменения RGB-сигнала. Если для формирования синхросигналов использовать счетчики, считающие количество синхроимпульсов, следующих с частотой 25 МГц, то значения в этих столбцах соответствуют значениям, при которых счетчики обнуляются.

Табл. 8

Символ на рис. 8	Параметр	Вертикальная развертка			Горизонтальная развертка	
		Время (микросек)	Периодов RGB	Линий развертки	Время (микросек)	Периодов RGB
T_S	Период синхросигнала	16700	416800	521	32	800
T_{DISP}	Время вывода на экран	15360	384000	480	25,6	640
T_{PW}	Длительность импульса	64	1600	2	3,84	96
T_{FP}	Front porch	320	8000	10	0,64	16
T_{BP}	Back porch	928	23200	29	1,92	48

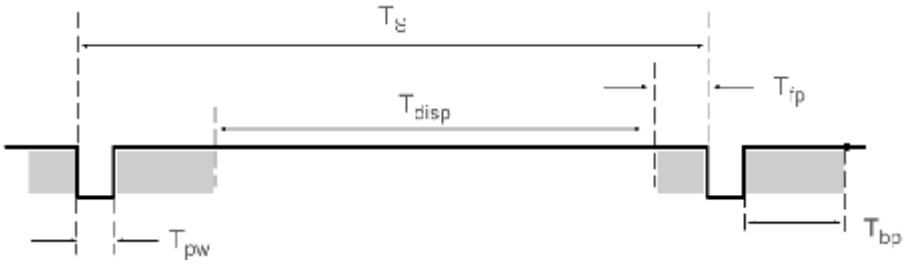

Рис. 8. Внешний синхросигнал VGA-монитора. Времена T_{FP}, T_{BP} и T_{PW} в сумме дают время Retrace

Часто для управления видеосигналом используют следующий прием. Задают синхросигнал, соответствующий скорости изменения RGB-сигнала. Период этого сигнала, по сути дела, задает время на вывод одного пиксела на экран. В нашем случае частота этого сигнала равна 25 МГц. Сигнал горизонтальной развертки генерируется счетчиком, считающим импульсы этого RGB-сигнала. С периодом T_S счетчик обнуляется (в нашем случае, досчитав до 800), выдавая импульс длительностью T_{PW}. Таким образом, число, содержащееся в счетчике в каждый момент времени, соответствует горизонтальной позиции выводимого на экран пиксела. Здесь необходимо только учитывать, что во время T_{PW} и T_{BP} в начале каждого периода развертки и во время T_{FP} в конце позиция пиксела

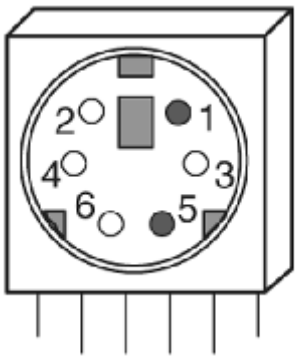

Рис. 9. Разъем PS/2. Вид спереди

находится как бы за пределами видимой области.

Еще один счетчик генерирует сигнал вертикальной развертки, считая импульсы синхросигнала горизонтальной развертки. Его значение в каждый момент времени определяет номер строки, в которую производится вывод информации.

На плате S3SK находится еще один стандартный порт (рис. 1, маркер 9), подсоединенный к стандартному разъему. Это порт PS/2, служащий для подключения стандартной клавиатуры или стандартной «мыши» (рис. 9). В табл. 9 приведен перечень сигналов, передаваемых по PS/2 соединению, и подключение их к контактам FPGA.

Табл. 9.

Контакт разъема PS/2	Сигнал	Контакт XC3S200FT256
1	Данные (Data, PS2D)	M15
2	Зарезервировано	–
3	Земля (GND)	Земля (GND)
4	Рабочее напряжение	–
5	Синхросигнал (CLK, PS2C)	M16
6	Зарезервировано	–

Из таблицы видно, что для передачи информации и клавиатура, и «мышь» используют два сигнала: собственно сигнал данных и синхросигнал. Сигнал данных представляет собой 11-разрядные слова. В эти 11 бит входят стартовый, стоповый биты и бит контроля четности.

Временные соотношения на PS/2 шине представлены на рис. 10. и в табл. 10.

Табл. 10

Обозначение на рисунке	Параметр	Минимальное значение (мс)	Максимальное значение (мс)
T_{CK}	Полупериод синхросигнала, время высокого или низкого уровня	30	50
T_{SU}	Задержка синхросигнала после стартового бита	5	25
T_{HLD}	Задержка данных после фронта синхросигнала	5	25

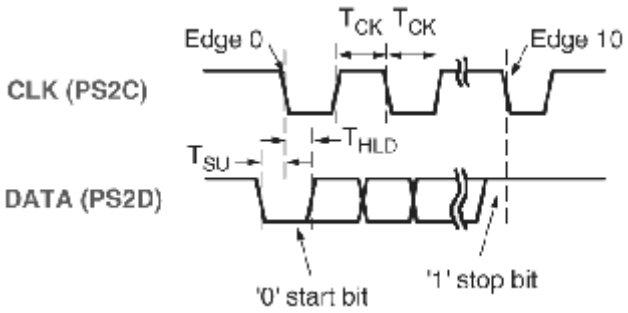

Рис. 10. Временные соотношения на шине PS/2

Пока нет данных для передачи, оба сигнала CLK и DATA находятся в состоянии логической единицы. Первый бит данных предваряется стартовым битом, т.е. сигнал на шине данных переходит в состояние 0. С задержкой T_{SU} после этого перепада запускается синхросигнал CLK (10 импульсов подряд). С задержкой T_{HLD} после каждого заднего (из 1 в 0) фронта синхросигнала значение сигнала DATA устанавливается в состояние 0 или 1 в соответствии со значением текущего бита передаваемой информации. Передача пакета завершается установкой сигнала DATA в состояние 1 после 9-го синхроимпульса (стоп-бит).

Данные, передаваемые клавиатурой и «мышью» внутри 11-разрядного пакета, организованы по-разному.

Клавиатура в каждом пакете данных передает скан-код нажатой клавиши. Скан-коды большинства клавиш клавиатуры приведены на рис. 11. Из рисунка видно, что некоторые клавиши, называемые «extended» имеют 2-байтный скан-код: старший байт скан-кода

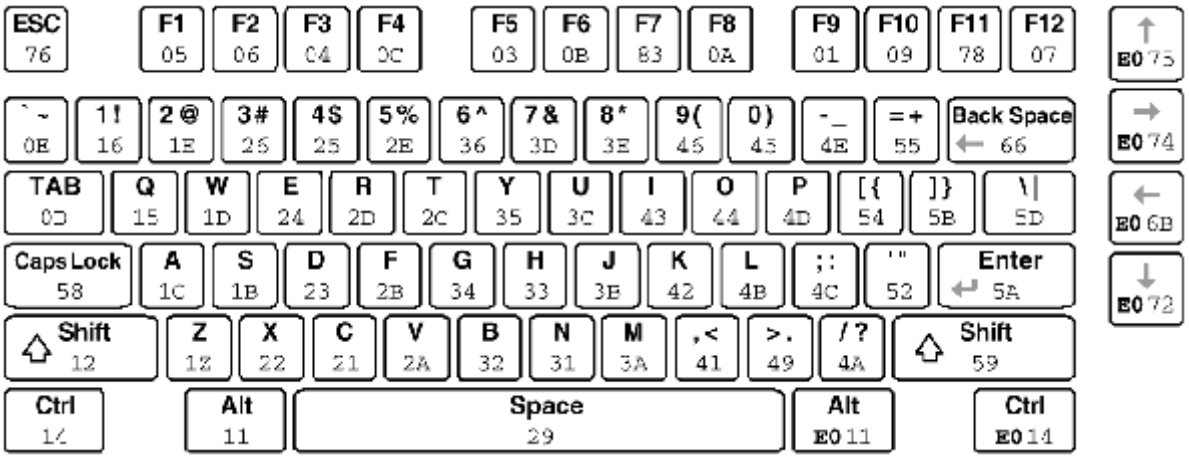

Рис. 11. Скан-коды большинства клавиш клавиатуры

таких клавиш – это код 0xE0. При отпускании нажатой клавиши клавиатура передает код 0xF0, а затем скан-код отпущенной клавиши. Отпускание «extended» клавиши приводит к передаче кодов 0xE0, 0xF0, младший байт скан-кода отпущенной клавиши.

Если клавиша удерживается в нажатом состоянии, то клавиатура повторяет передачу скан-кода этой клавиши каждые 100 миллисекунд.

Клавиатура передает скан-коды 11-битовыми пакетами. Первый бит в пакете – всегда 0. Это стартовый бит. Затем следует 8 бит скан-кода клавиши. Передача осуществляется в обратном порядке. То есть первым передается младший бит (LSB), затем старший. После 8 бит скан-кода следует 1 бит контроля четности. Передача пакета завершается передачей стоп-бита, который всегда равен 1. Биты сигнала DATA отделяются во времени друг от друга по задним фронтам синхросигнала. С началом каждого пакета данных клавиатура на проводе CLK генерирует последовательность из 11 синхроимпульсов. Синхросигнал генерируется с частотой от 20 до 30 кГц.

При перемещении, «мышь» генерирует три 11-разрядных пакета на проводе DATA и синхросигнал CLK. В противном случае эти сигналы имеют высокий уровень. Структура пакетов данных представлена на рис. 12.

Рис. 12. Структура трех 11-разрядных пакетов данных, генерируемых «мышью» при движении

Из рисунка видно, что в каждом пакете 8 бит данных обрамлены старт-стоповыми битами и битом четности. Второй и третий пакеты содержат относительные координаты «мыши». Значения битов XS и YS в первом пакете определяют знак смещения. XS=0 – «мышь» движется вправо, XS=1 – влево, YS=0 – вперед, YS=1 – назад. Передача трех пакетов повторяется каждые 50 миллисекунд. Чем быстрее движется «мышь», тем большие значения принимают относительные координаты во втором и третьем пакетах.

При переполнении 8-разрядных координат, устанавливаются в 1 биты XV и YV в первом пакете. Биты L и R в первом пакете индицируют нажатие левой и правой кнопок «мыши». При нажатии кнопки соответствующий бит устанавливается в 1.

Клавиатура и «мышь» в качестве питающего напряжения используют напряжение, подаваемое на провода разъема PS/2. Большинство современных клавиатур и манипуляторов «мышь» могут одинаково хорошо работать и от напряжения 3,3 В, и от 5 В. Устаревшие модели могут требовать подачи напряжения только 5 В. Хотя для FPGA предпочтительнее использовать напряжение 3,3 В, на плате S3SK имеется переключатель, обозначенный JP2 (рис. 1, маркер 30), положение перемычки которого определяет напряжение на PS/2-шине (рис. 13).

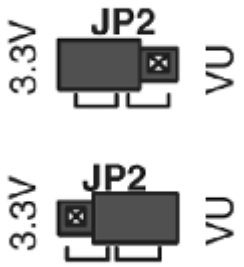

Рис. 13. Положение перемычки на переключателе JP2 определяет напряжение, используемое сигналами на PS/2-шине в качестве высокого уровня: сверху – для напряжения 3,3 В, снизу – для 5 В

На рис. 1 маркером 6 отмечен стандартный DB9 разъем последовательного порта RS-232 (рис. 14). Маркером 7 на рис. 1 обозначен конвертор напряжений Maxim MAX3232, обеспечивающий согласование уровней напряжения на выходах FPGA и на сигнальных проводах последовательного порта. Схема подключения DB9 к контактам FPGA приведена на рис. 15.

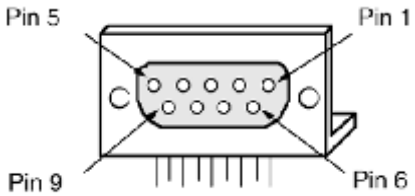

Рис. 14. Разъем DB9. Вид спереди

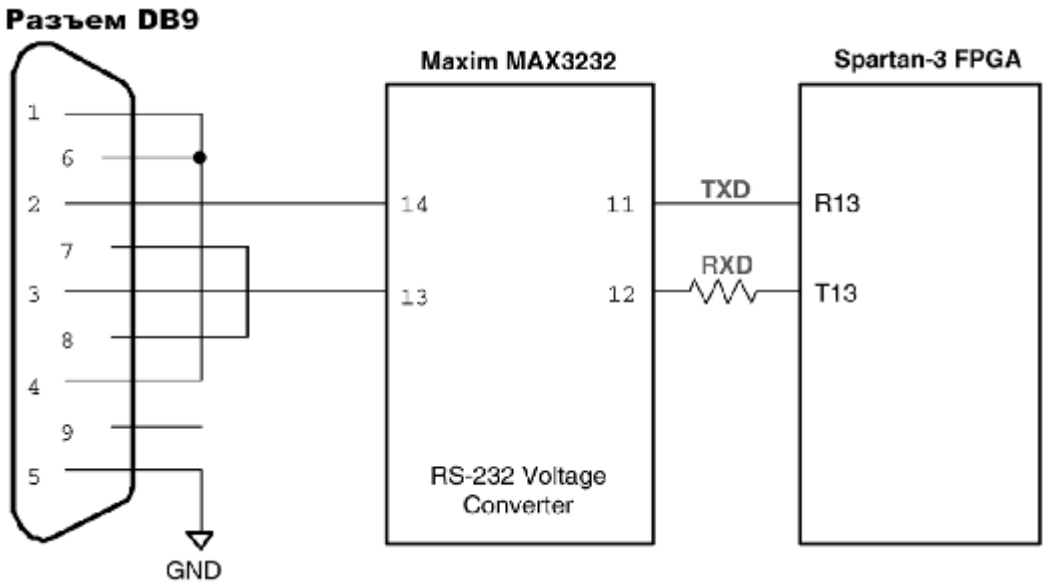

Рис. 15. Подключение контактов FPGA к разъему DB9

Передача по RS-232 осуществляется асинхронно, т.е. передающее устройство может начинать передачу в произвольный момент времени. Передача производится на определенной скорости, известной и передатчику, и приемнику. В состоянии ожидания передатчик держит высокий уровень сигнала TXD. Перепад сигнала из высокого состояния в нулевое означает начало передачи – стартовый бит. Далее на оговоренной скорости передающее устройство посылает пакет из 8 бит передаваемой информации (младший бит байта передается первым). Передача пакета данных завершается передачей стопового бита, который устанавливает сигнал в высокое состояние. Приемное устройство по переднему фронту стартового бита запускает внутренний синхросигнал, частота которого задается скоростью передачи. Например, если оговоренная скорость передачи 38400 бит/с, то соответствующая частота внутреннего синхросигнала – 38400 Гц. Приемник интерпретирует значение принимаемого сигнала как значение следующего бита передаваемых данных по каждому переднему фронту синхросигнала. Максимальное количество бит данных в каждом передаваемом пакете равно 8 (рис. 16). После восьмого бита данных должен обязательно следовать стоповый бит, устанавливающий

Рис. 16. Сигнал RXD и внутренний синхросигнал приемника

передаваемый сигнал в высокий уровень. Если по переднему фронту десятого синхроимпульса приемник не обнаруживает высокий уровень принимаемого сигнала, то последний принятый пакет считается ошибочным и принятые данные отбрасываются.

На оборотной стороне платы S3SK находится 50 МГц осциллятор Epson SG-8002JF, который можно использовать в качестве источника синхросигнала для FPGA (рис. 1, маркер 14). Кроме того, на лицевой стороне S3SK расположен 8-штырьковый разъем, предназначенный для подключения внешнего источника синхросигнала (рис. 1, маркер 15). Сигнал Epson SG-8002JF подключен к контакту T9 микросхемы FPGA. Синхросигнал внешнего источника, подключенного к разъему, передается на контакт D9. Для преобразования частоты, смещения фазы и других преобразований синхросигнала можно использовать блоки DCM, имеющиеся в составе FPGA XC3S200.

Плата S3SK содержит микросхему FLASH-памяти – XCF02S. Эта энергонезависимая память обычно используется для хранения bitstream-данных, конфигурирующих FPGA. Таким образом, программу, по которой должна работать FPGA-микросхема, можно загрузить как непосредственно в FPGA, так и в энергонезависимую память. В первом случае после выключения питания все данные, записанные в XC3S200, будут утеряны. Во втором – данные для программирования FPGA хранятся во FLASH-памяти и могут быть загружены в FPGA в любой момент. Для загрузки необходимо нажать кнопку PROG (рис. 1, маркер 17). После успешной загрузки загорается индикатор DONE (рис. 1, маркер 18). Объем FLASH-памяти – 2 Мбита. (Напомним, что для конфигурирования FPGA достаточно 1 Мбита данных).

Прежде всего отметим, что для использования FLASH для конфигурирования FPGA все три перемычки на переключателе J8 (рис. 1, маркер 16) должны быть установлены (рис. 17). Кроме того, имеется три возможных режима работы FLASH-памяти. Эти режимы контролируются перемычками на переключателе JP1 (рис. 1 маркер 3). Возможные положения перемычек иллюстрируются на рис. 18.

Рис. 17. Для использования FLASH-памяти все перемычки переключателя должны быть установлены

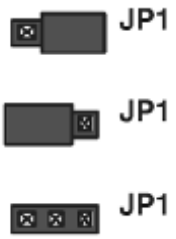

Рис. 18. FLASH-память используется для конфигурирования FPGA (сверху).
FLASH-память используется для конфигурирования FPGA, после чего остается доступной для чтения. FPGA может прочитать дополнительные данные из FLASH-памяти (посередине).
FLASH-память не доступна для использования (снизу)

Схема подключения FLASH к FPGA при использовании второго режима (когда возможно чтение дополнительных данных из памяти после конфигурации FPGA) приведена на рис. 19.

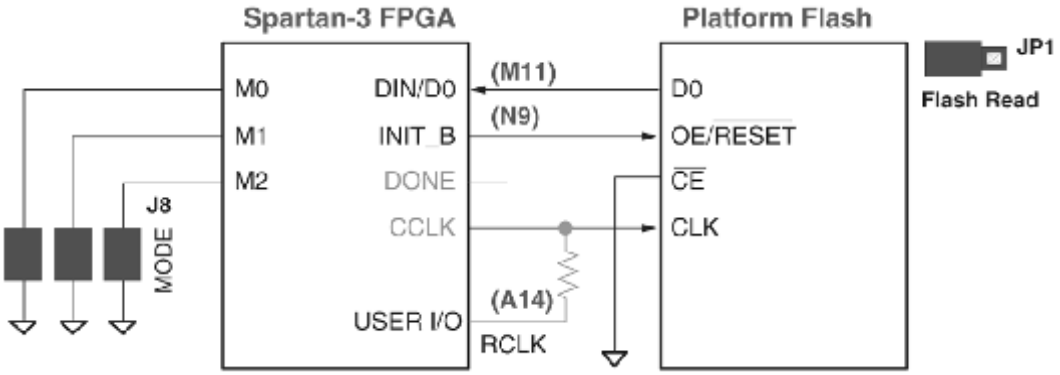

Рис. 19. Подключение FLASH к FPGA для чтения дополнительных данных после конфигурации (в скобках указаны номера контактов XC3S200FT256)

Для чтения дополнительных данных после конфигурации FPGA необходимо установить в высокий уровень сигнал INIT_B (N9) и подать синхросигнал RCLK (A14). XCF02S является последовательной памятью, т.е. адрес, по которому производится чтение бита информации обновляется (увеличивается) после каждого чтения (т. е. по каждому импульсу CLK). После конфигурации FPGA, таким образом, адрес для чтения из FLASH-памяти оказывается установлен на следующий бит за последним битом кофигурационных данных. По синхросигналу RCLK дополнительные данные начинают поступать на вывод DIN (M11) FPGA.

Плата S3SK содержит JTAG-порт, который может использоваться как для загрузки данных (bitstream) в XC3S200 или в XCF02S, так и для тестирования загруженных программ. Соответственно FPGA и FLASH-память объединены в единую JTAG-цепочку (рис. 20). В комплект поставки S3SK входит Digilent JTAG3 кабель, позволяющий

подключать параллельный порт персонального компьютера к разъему J7 на плате (рис. 1, маркер 22). Для загрузки данных в устройства из JTAG-цепочки в состав Xilinx ISE WebPack включена программа iMPACT. Все контакты JTAG-кабеля подписаны на плате и на разъеме кабеля. Это помогает избежать неправильного подключения кабеля к S3SK.

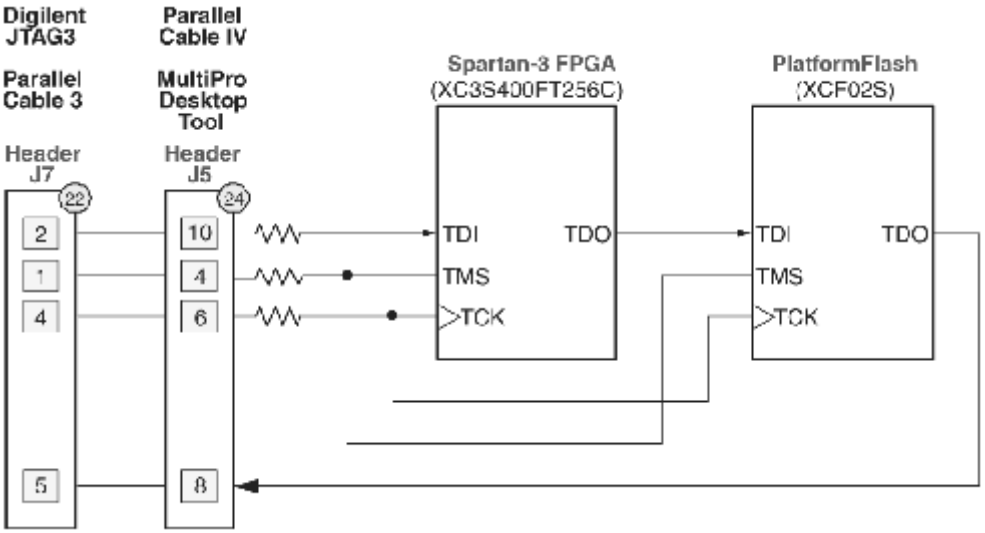

Рис. 20. JTAG-цепочка на плате S3SK. Номера слева обозначают контакты разъемов J7 и J5

В комплект поставки S3SK входит стандартный преобразователь напряжения на +5 В, предназначенный питания платы. Источник питания подключается к разъему на плате (рис. 1, маркер 25). Никаких дополнительных выключателей питания на плате не предусмотрено. Включение питания индицируется на плате светодиодом (рис. 1, маркер 26).

Самостоятельная работа №3. Возможности, архитектура, система команд микропроцессора PicoBlaze.

Микропроцессор PicoBlaze™ представляет собой компактный, производительный, экономичный и полностью встроенный 8-битный RISC-микроконтроллер, оптимизированный для семейства Spartan®-3, с поддержкой Virtex®-5, Spartan-6 и Virtex ®-6. PicoBlaze – это микроконтроллер, который позволят выполнять задачи управления и простой обработки данных.

PicoBlaze оптимизирован под создание высокоэффективных систем с низкой стоимостью развертывания. Он занимает 96 FPGA-блоков (слайсов), что составляет всего лишь 12,5% размера FPGA XC3S50 и 0,3% FPGA XC3S5000. В типичных реализациях FPGA один элемент блочной памяти хранит до 1024 программных инструкций, которые автоматически загружаются во время конфигурирования FPGA. Даже с такой эффективностью использования ресурсов PicoBlaze микроконтроллер выполняет порядка 44-100 миллионов операций в секунду (MIPS) в зависимости от семейства FPGA и класса скорости.

Ядро микроконтроллера PicoBlaze полностью встроено в целевую FPGA и не требует внешних ресурсов. Микроконтроллер PicoBlaze является чрезвычайно гибким. Основная функциональность легко расширяется и усиливается с помощью подключения дополнительной логики FPGA на вход микроконтроллера и к выходным портам.

Микроконтроллер PicoBlaze обеспечивает богатый и гибкий интерфейс ввода/вывода при значительно меньших затратах, чем у готовых контроллеров. Также набор периферийных устройств PicoBlaze может быть настроен в соответствии с конкретными особенностями, функциями и требованиями к стоимости конкретного приложения. Так как PicoBlaze поставляется в виде синтезируемых исходных кодов VHDL, ядра могут быть перенесены на последующие архитектуры FPGA, что позволяет постоянно модифицировать продукт. Будучи интегрированным в FPGA, PicoBlaze-микроконтроллер уменьшает площадь печатной платы, стоимость разработки и набор комплектующих.

PicoBlaze FPC поддерживает набор средств разработки, включая ассемблер, графическую интегрированную среду разработки (IDE), графический симулятор набора команд, исходный код на языке VHDL и имитационные модели. PicoBlaze микроконтроллер также поддерживается в среде разработки Xilinx System Generator.

Для получения дополнительной информации смотрите соответствующую документацию[2].

ЛИТЕРАТУРА:

1. Стешенко В.Б. ПЛИС фирмы Altera: элементная база, система проектирования и языки описания аппаратуры. – М.: Издательский дом «Додэка-XXI», 2007. – 576 с.

2. PicoBlaze 8-bit Embedded Microcontroller User Guide for Spartan-3, Spartan-6, Virtex-5, and Virtex-6 FPGAs UG129 (v2.0) January 28, 2010. Режим доступа: http://www.ujk.edu.pl/informatyka/pardyka/ug129.pdf.