Министерство образования и науки Российской Федерации
Государственное образовательное учреждение высшего профессионального образования
«Томский государственный университет систем управления и радиоэлектроники»

УТВЕРЖДАЮ

Проректор по учебной работе
_____ Л.А. Боков
«____» _____ 2011 г.

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ
К ЛАБОРАТОРНЫМ ЗАНЯТИЯМ

по дисциплине

«Основы проектирования систем на кристалле»

Составлена кафедрой «Управление инновациями»

Для студентов, обучающихся
по направлению подготовки 220600 «Инноватика»
Магистерская программа «Мультимедийные многопроцессорные системы на кристалле»

Форма обучения очная

Составитель
Доцент _____ О.Г. Пономарев

«___» _____ 2011 г.

Томск 2011

Содержание

Лабораторная работа №1. Инструменты разработки приложений для FPGA компании Xilinx

Введение

Эта лабораторная 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 лабораторной работой вы сможете:

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

Методика

Эта лабораторная 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 лабораторных работ вы будете использовать «Spartan-3 board» для того, чтобы проверять результаты вашей работы. Вдобавок к собственно самой плате для успешного выполнения лабораторных 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.

Рис. 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/products/design_resources/proc_central/grouping/picoblaze.htm

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

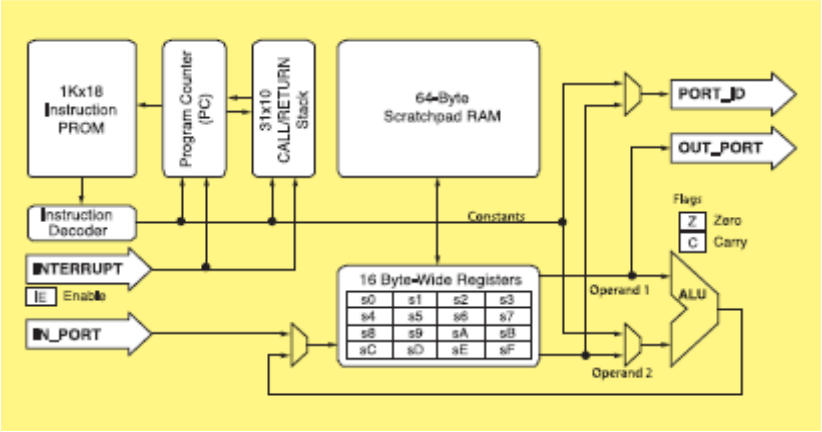


Рис. 2. PicoBlaze блок-схема

Рис. 3 иллюстрирует сигналы интерфейса верхнего уровня микроконтроллера.

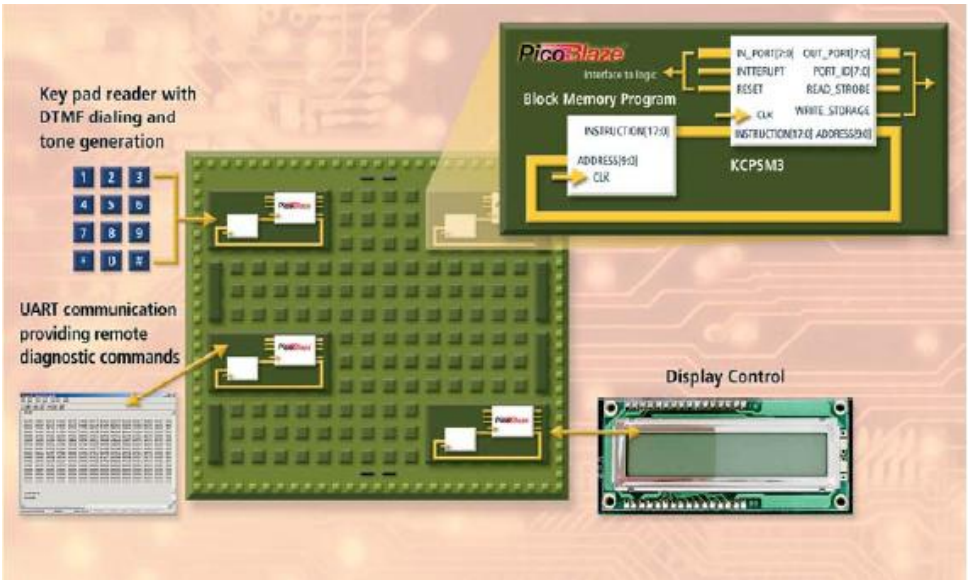


Рис. 3. Интерфейсные соединения и возможные приложения

Средства и методы разработки

В качестве средства для разработки ассемблерных программ для процессора PicoBlaze в этом цикле лабораторных 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).

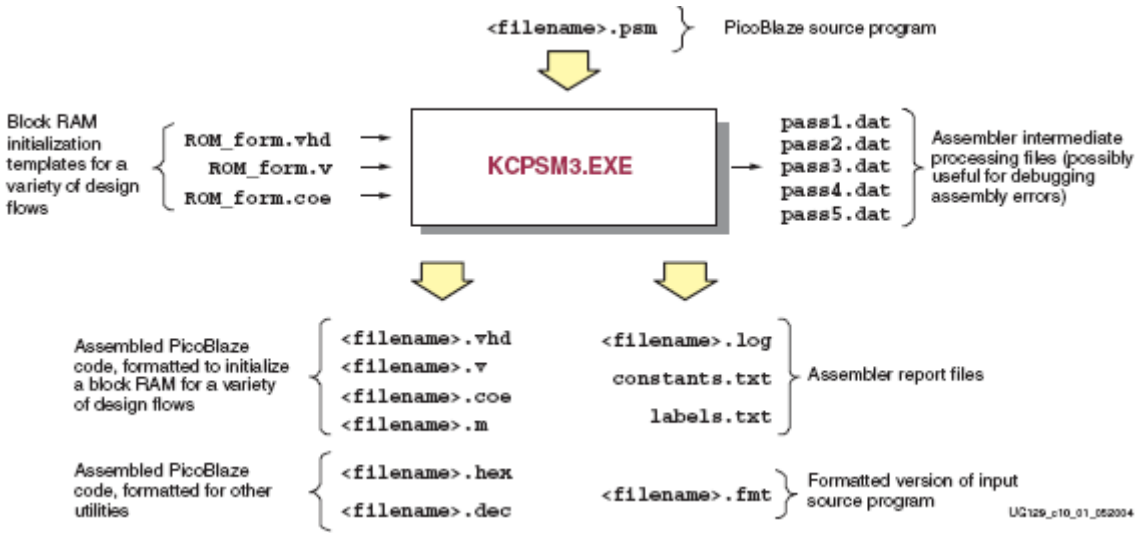


Рис. 4. Входные и выходные файлы для ассемблера KCPSM3

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**)

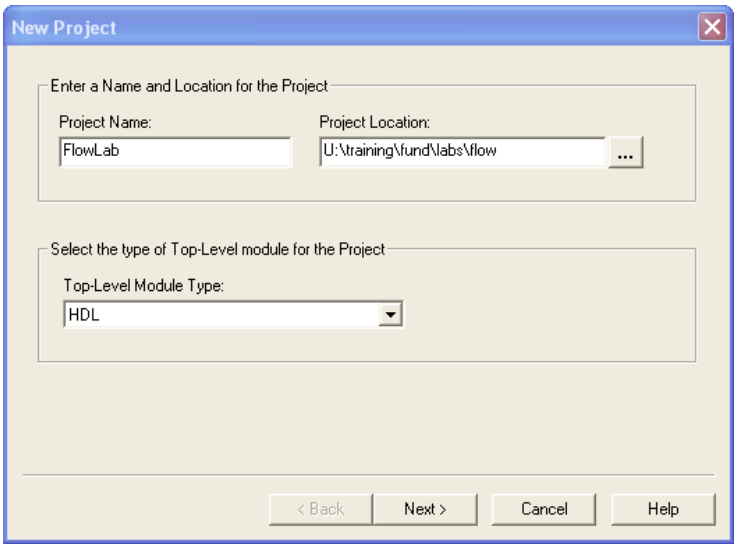


Рис. 5. Мастер создания нового проекта.

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**)

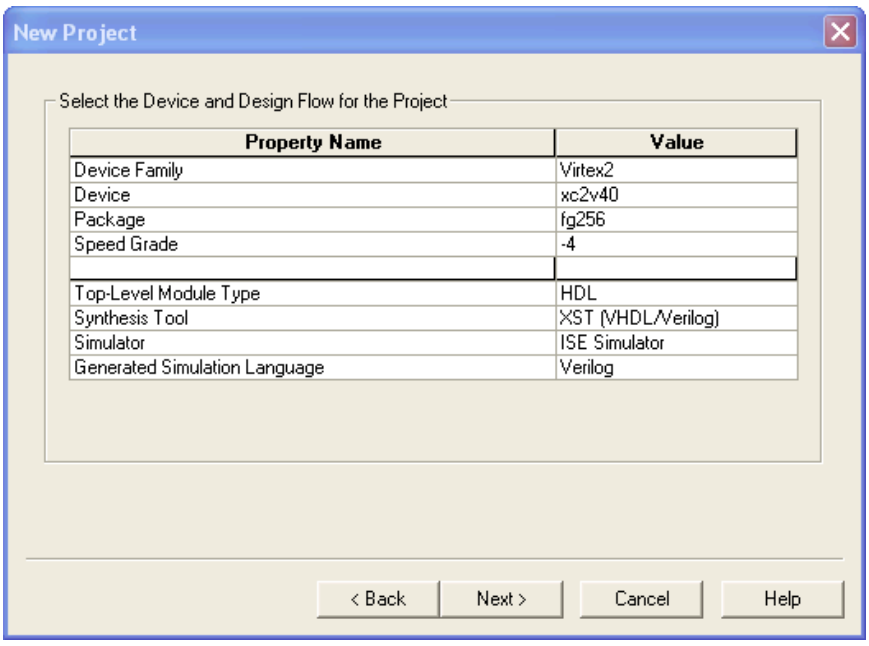


Рис. 6. Диалог установок процесса разработки

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.

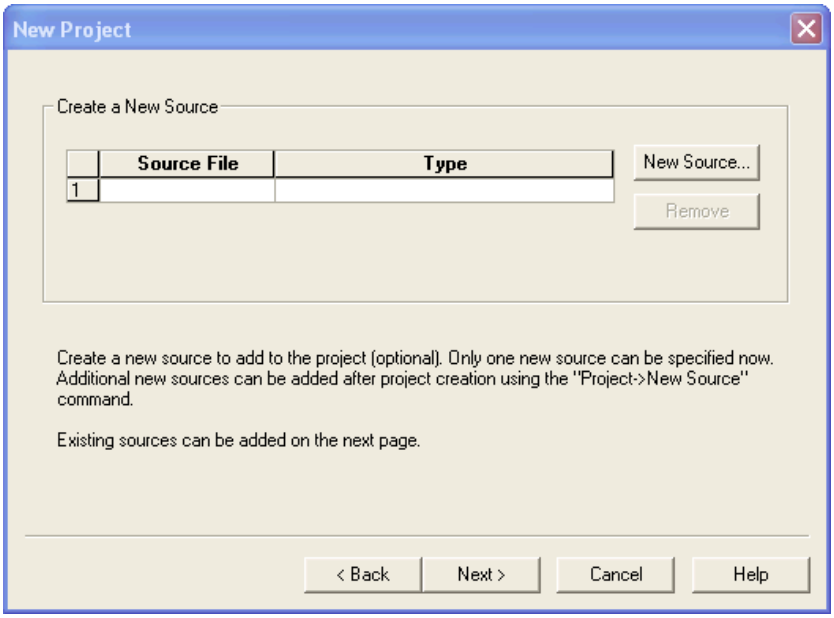


Рис. 7. Диалог Создания Нового Объекта

❼ Нажмите **Next**

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

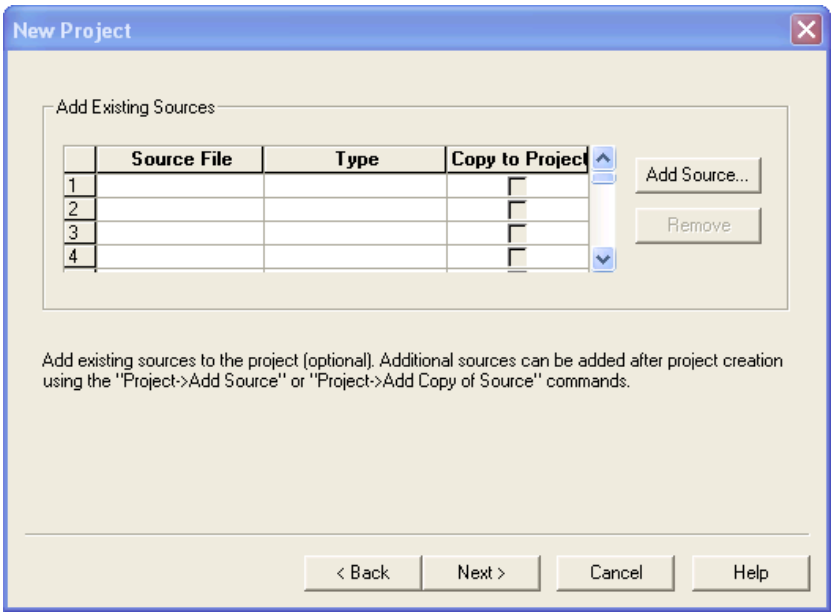


Рис. 8. Диалог добавления в проект существующего документа

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**.

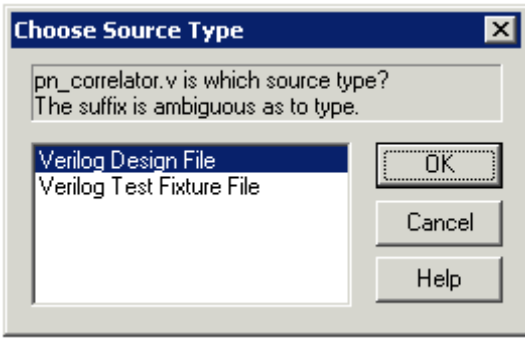


Рисунок 1-9. Выбор типа источника

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.

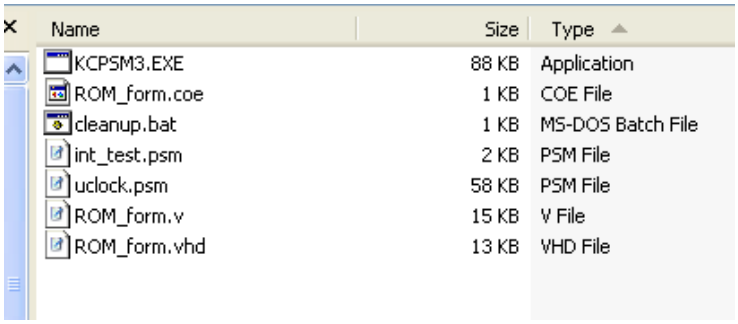


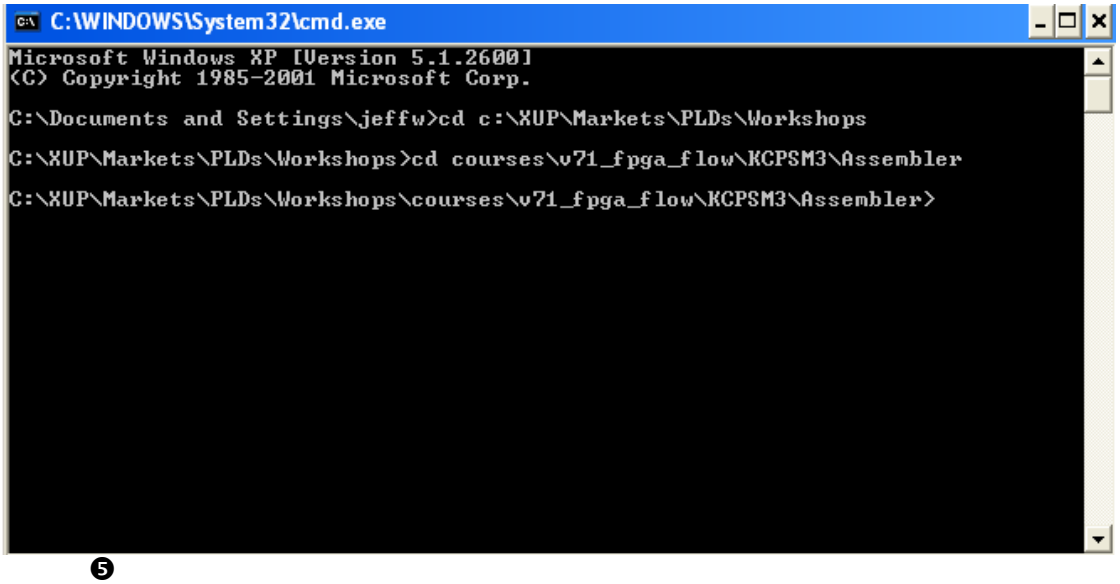
Рис. 10. Содержание Директории с Ассемблером

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 CDROM.

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

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

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



❺

Рис. 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.

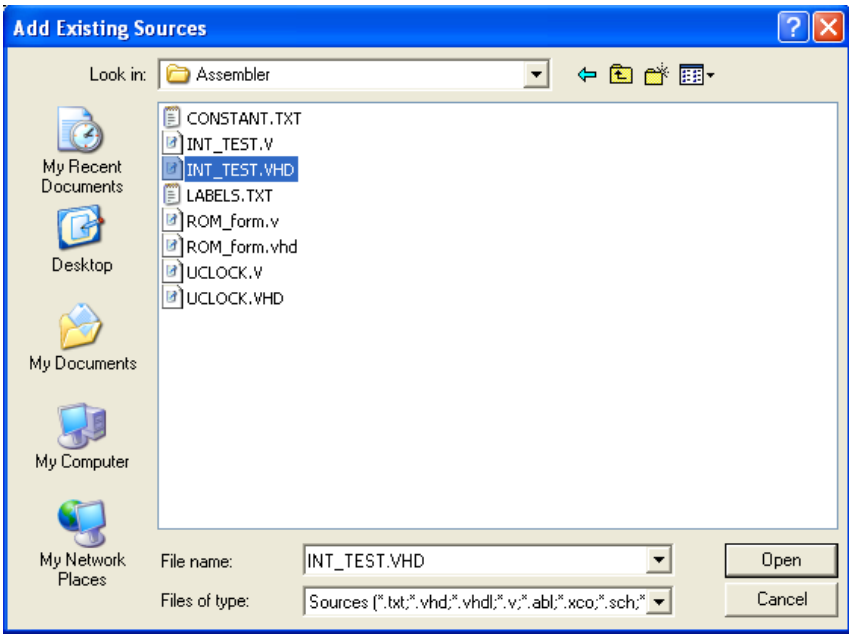


Рис. 12. Добавление int_test HDL (программный файл для ROM) в проект

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

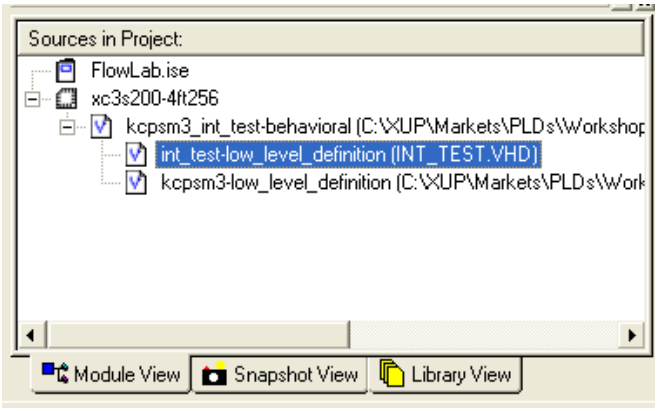


Рис. 13. Иерархия дизайна PicoBlaze контроллера

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.

Моделирование проекта — Шаг 4

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

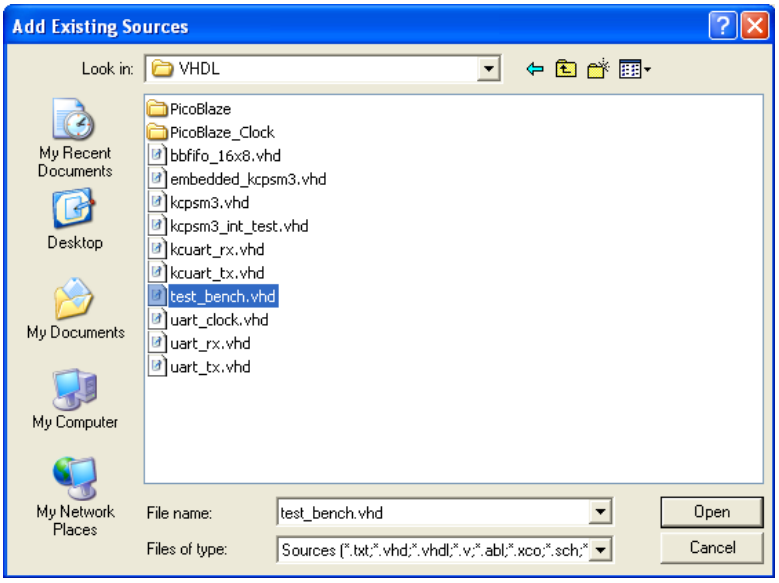


Рис. 14. Месторасположение файла испытательного стенда

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.

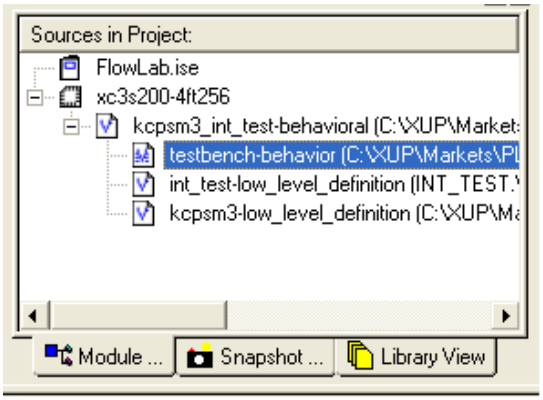


Рисунок 1-15. Иерархия проекта с добавленным файлом Testbench

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>.

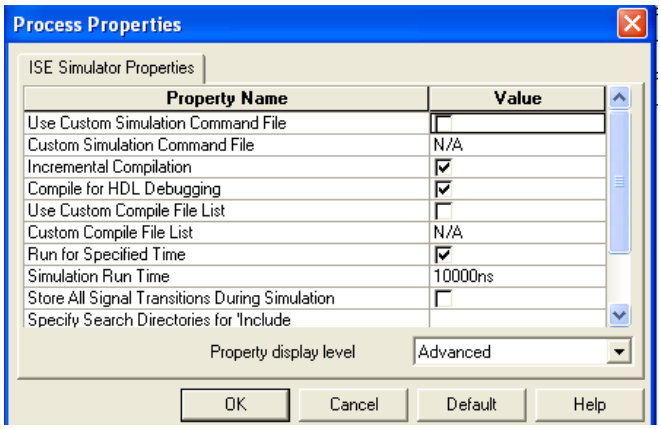


Рис. 16. Свойства iSIM Behavioral Simulation

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.

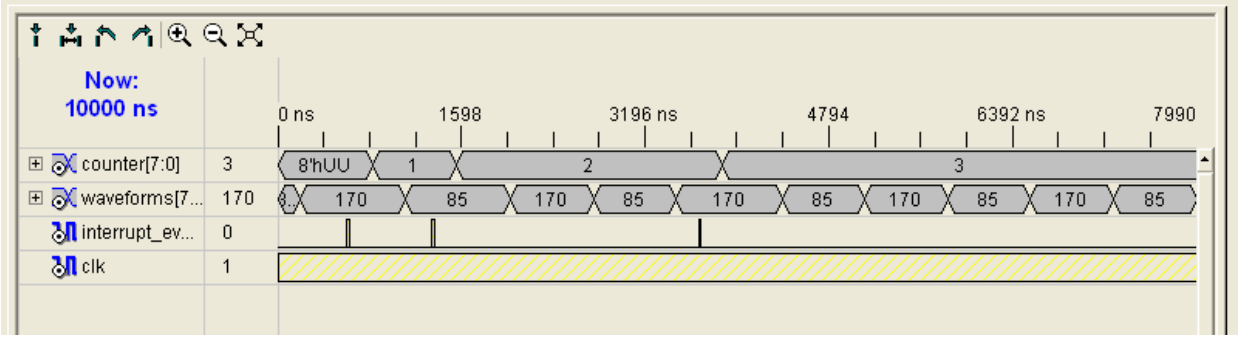


Рис. 17. iSIM HDL Simulator

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.

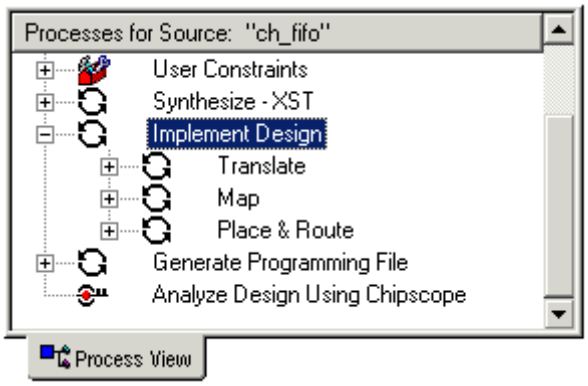


Рисунок 1-17. Окно Processes

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 игнорированы.

❹ Прочитайте некоторые сообщения о предупреждениях.

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

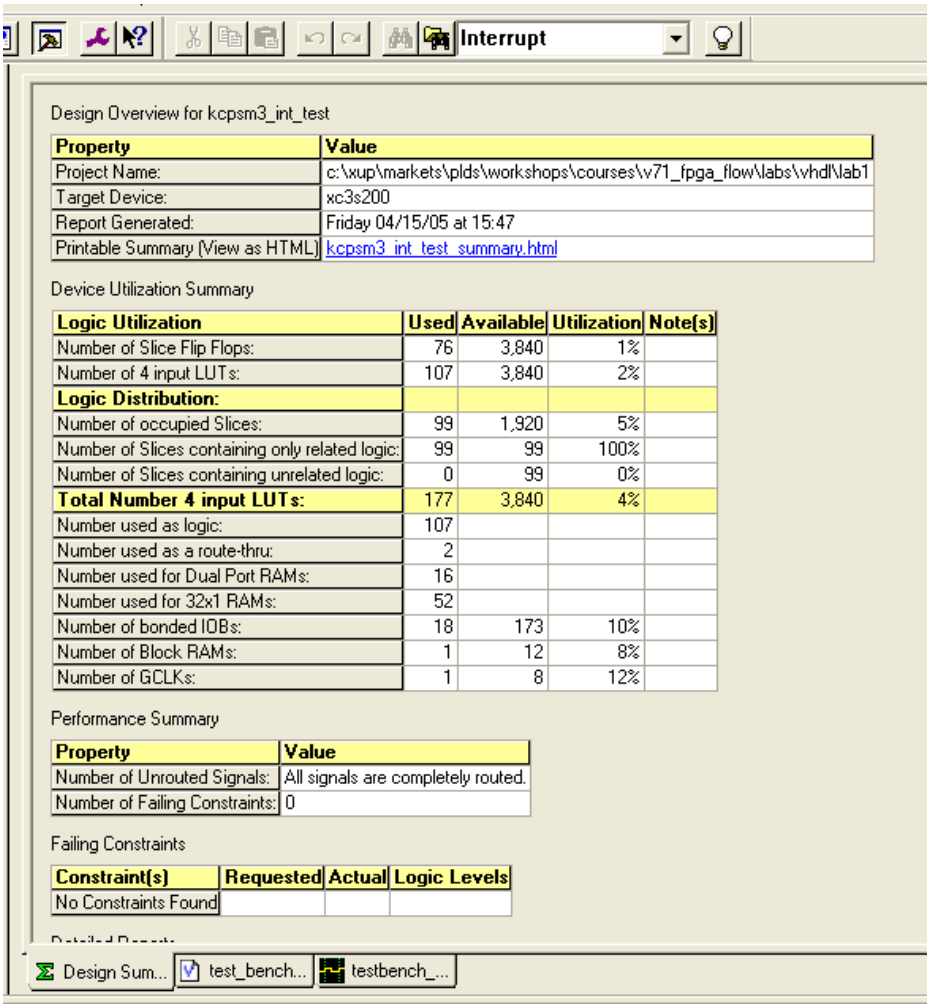


Рис. 18. Design Summary

Заключение

В этой лабораторной 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 лабораторной работы является знакомство с процессором PicoBlaze. Необходимо внимательно ознакомиться с текстами ассемблерных программ, очень полезно изучить VHDL-код процессора. Все тексты содержатся на прилагаемом CD-ROM.

В следующей лабораторной 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.

Лабораторная работа №2. Знакомство с Architecture Wizard и редактором PACE

Ведение

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

Цели

После выполнения этой лабораторной 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 лабораторная 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.

1. Если вы уже закрыли ISE™ Project Navigator, выберите **Start → Programs → Xilinx ISE → Project Navigator**
2. Выберите **File → Open Project** в Project Navigator

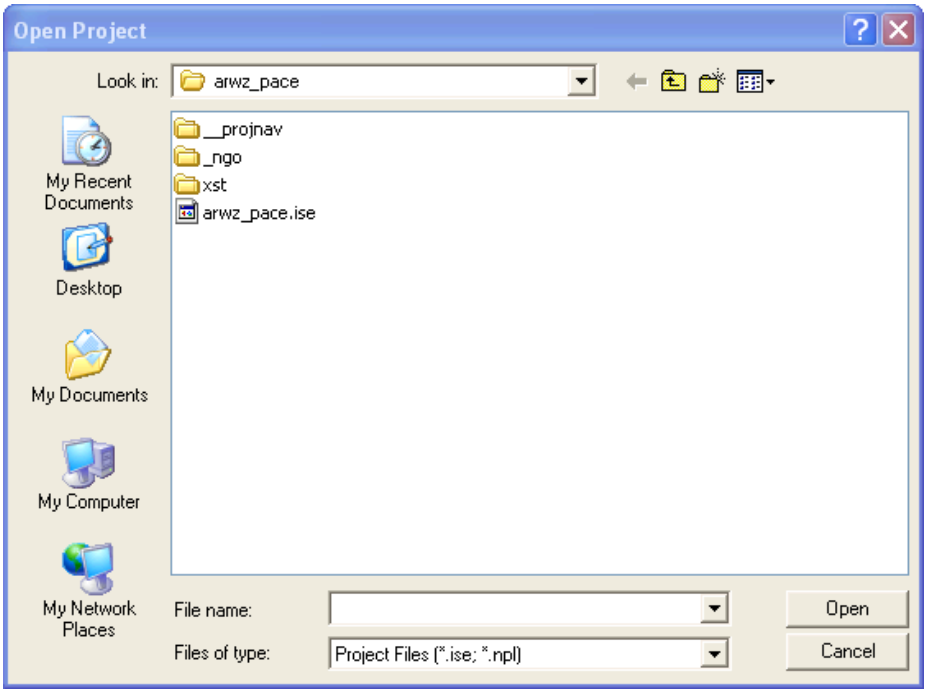


Рис. 1. Окно Открыть Проект

3. Найдите директорию *c:\xup\fpgaflow\labs\vhdl\lab2\arwz_pace* и выберите ***arwz_pace.ise***
4. Нажмите **Open**

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

1. В окне Processes для Source window, дважды щелкните **Create New Source**
2. 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**)

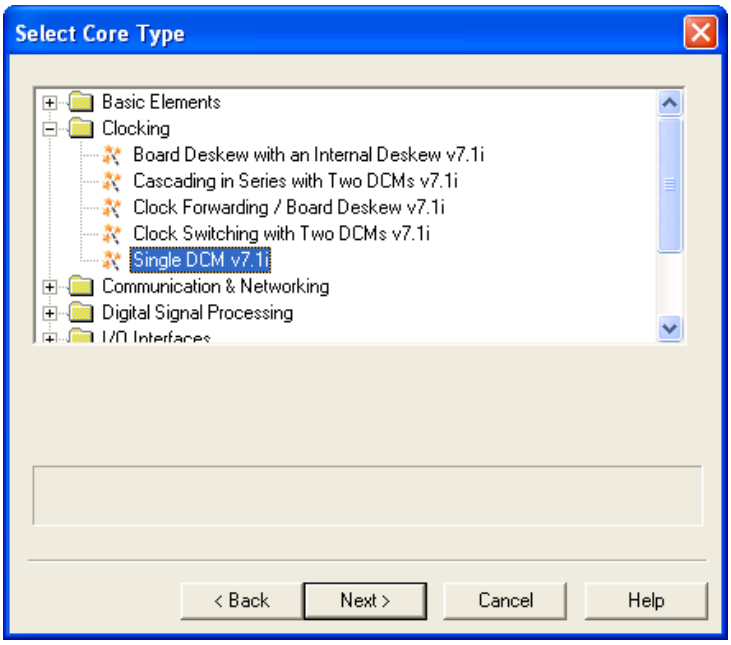


Рис. 2. Окно выбора Мастера Архитектуры

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**

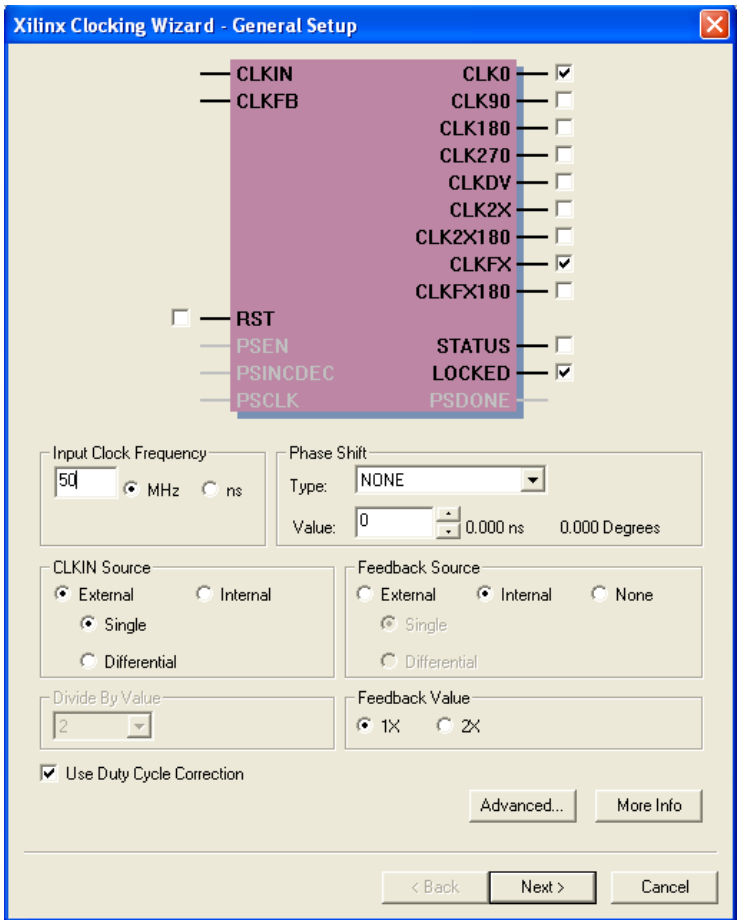


Рис. 3. Xilinx Clocking Wizard – General Setup Window

❼ Нажмите <**Next**>

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

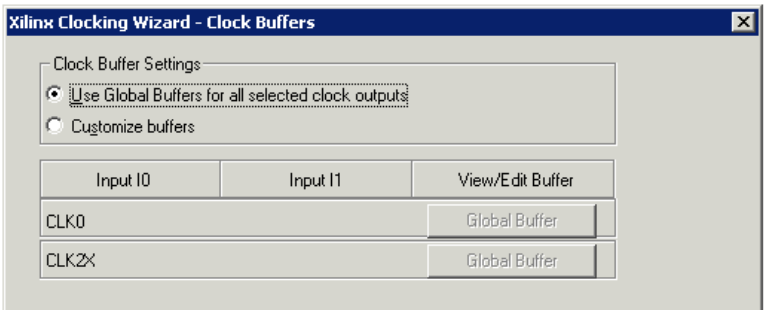


Рис. 4. Xilinx Clocking Wizard – Clock Buffers Window

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**.

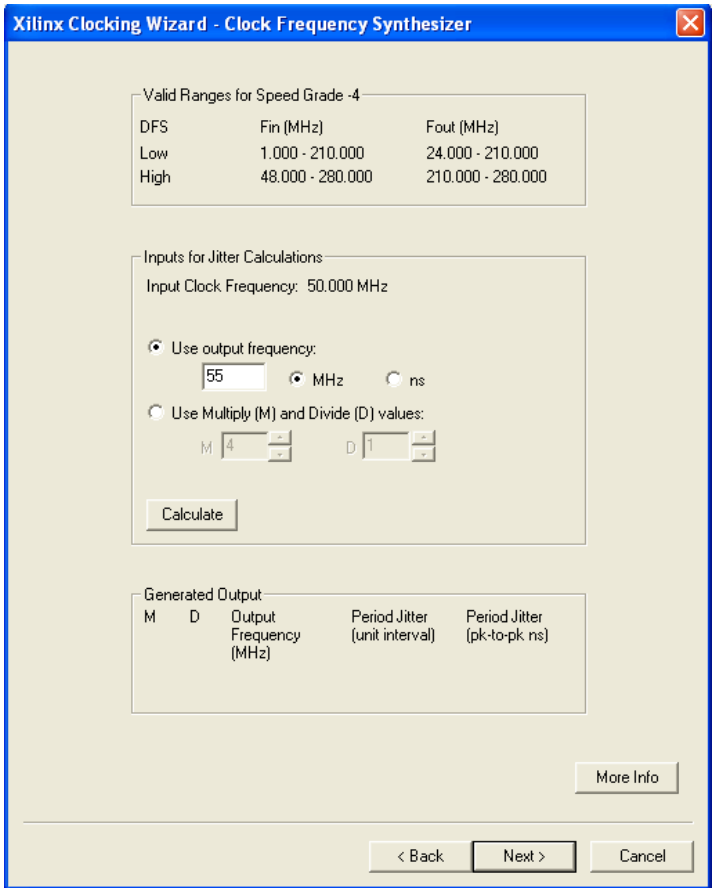


Рис. 5. Уточнение выходной частоты DCM

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.

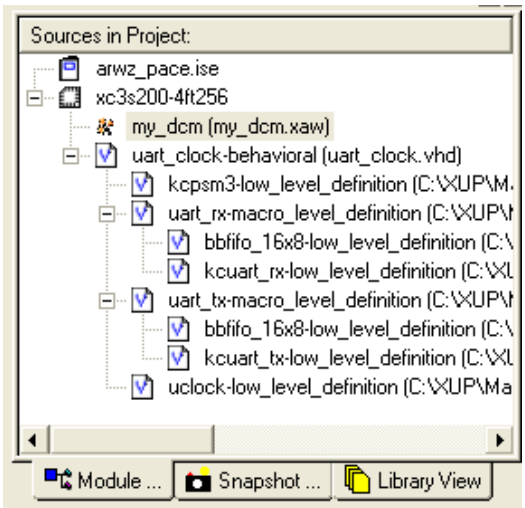


Рисунок 2-6. DCM компонент в иерархии проекта

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**

❺ ***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.

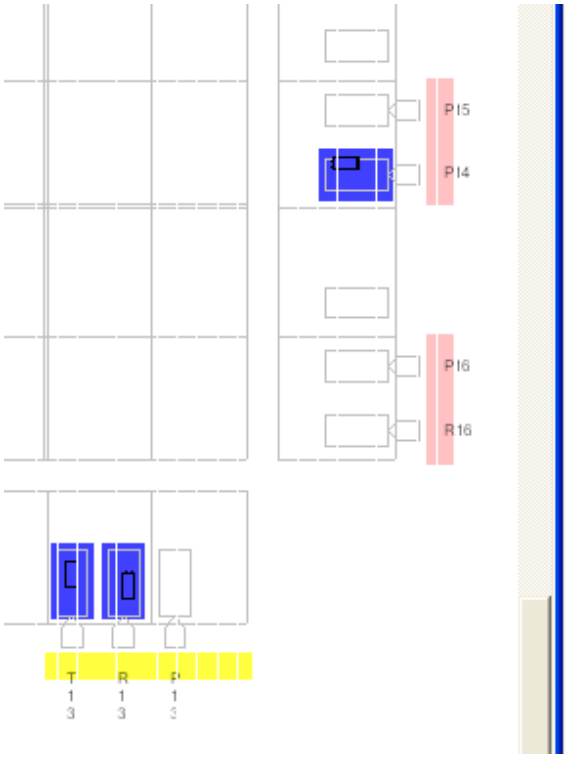


Рисунок 2-8. Окно Архитектура Устройства

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

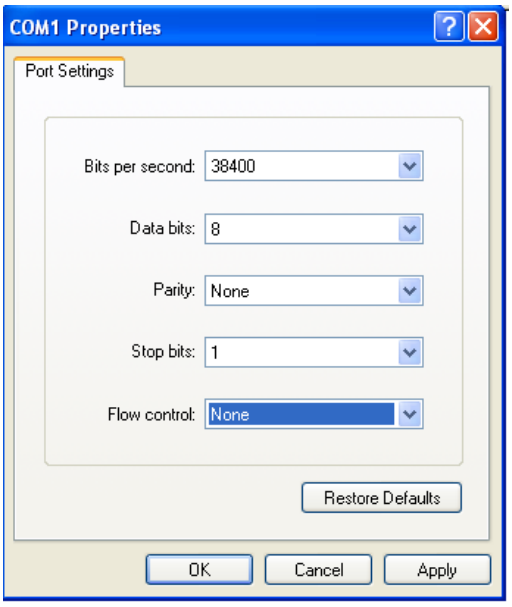


Рис. 9. Установки для соединения через последовательный порт

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

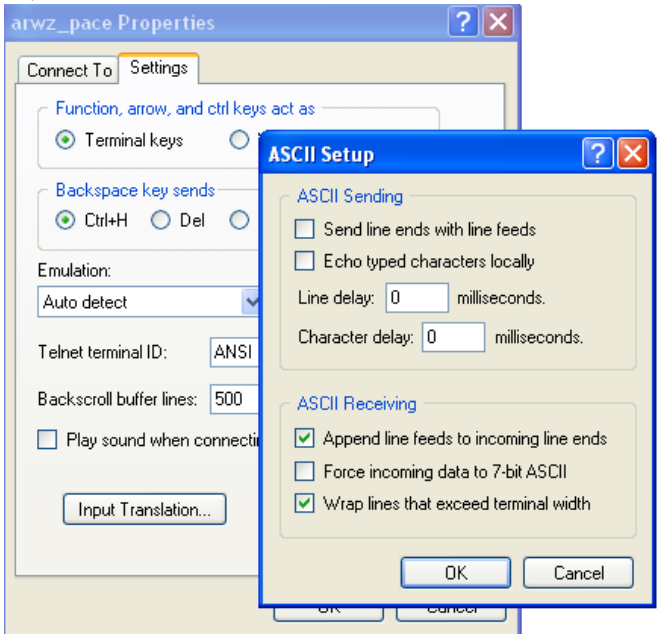


Рис. 10. ASCII Установки для соединения через последовательный порт

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.

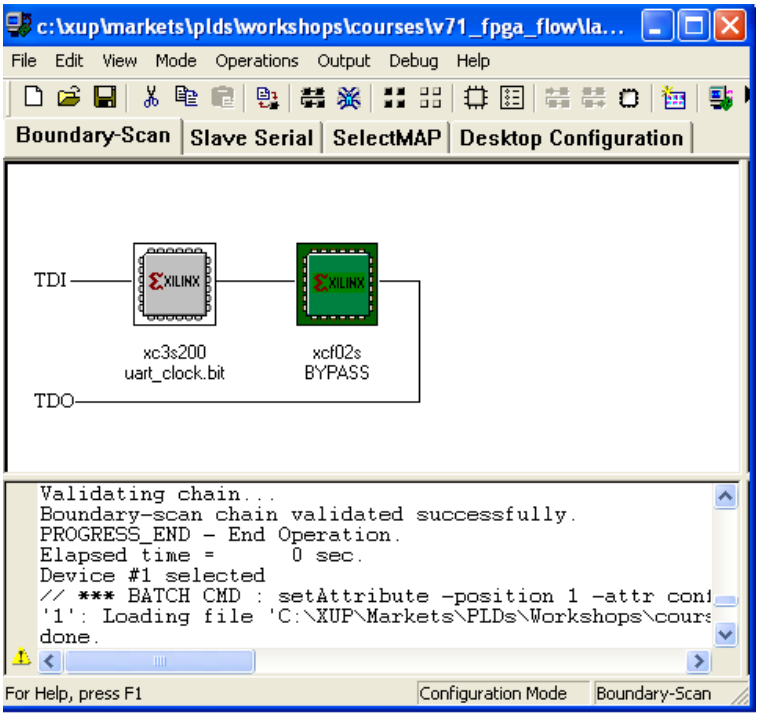


Рис. 11. JTAG-цепочка с установленным конфигурационным файлом

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>».

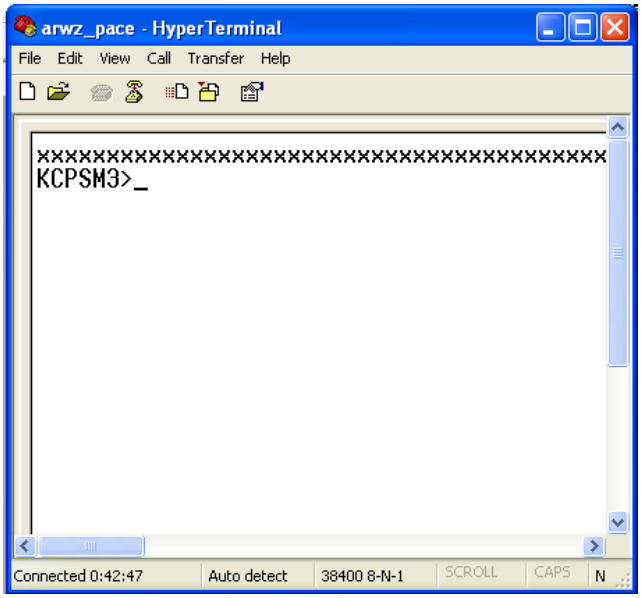


Рис. 12. Последовательная связь с PicoBlaze

Функционирование UART Real-Time Clock Шаг 6

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. Напомнаем, что индикация времени осуществляется в формате hh:mm:ss.

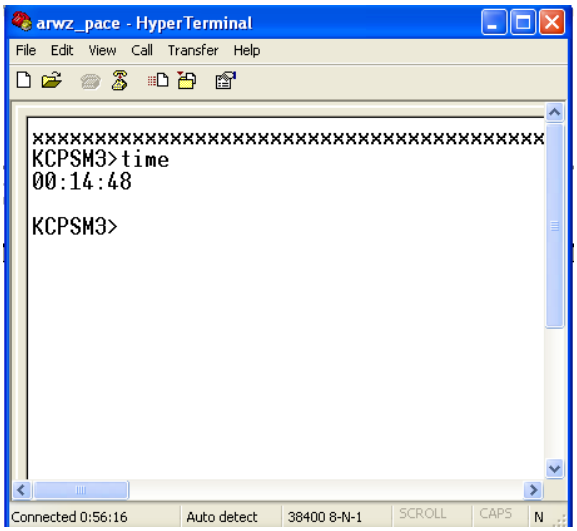


Рис. 13. Текущее Время

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

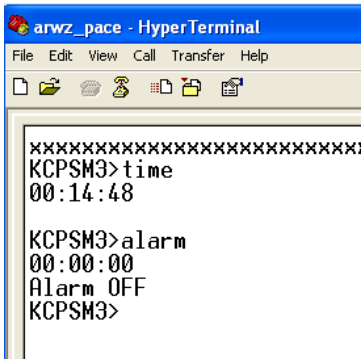


Рис. 14. Время и Статус будильника

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 лабораторной 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.

Лабораторная работа №3. Задание временных ограничений (Global Timing Constraints)

Введение

В этой лабораторной 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 лабораторной 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.

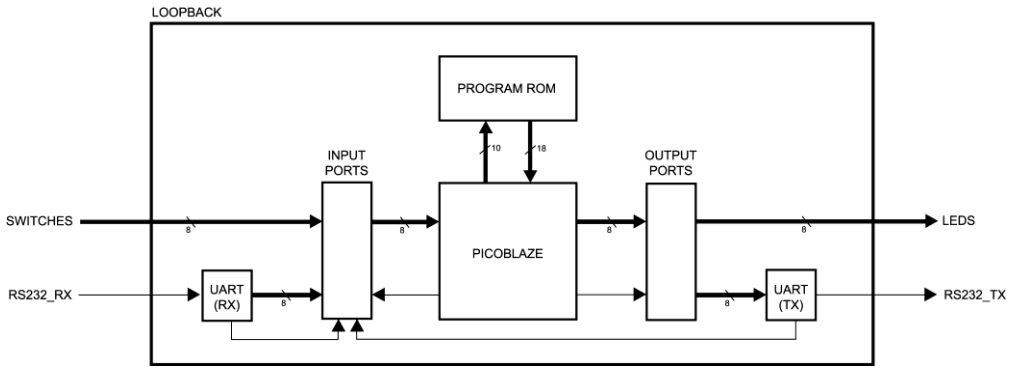


Рис. 1. PicoBlaze System

Главная задача этой лабораторной 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 лабораторной, 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.

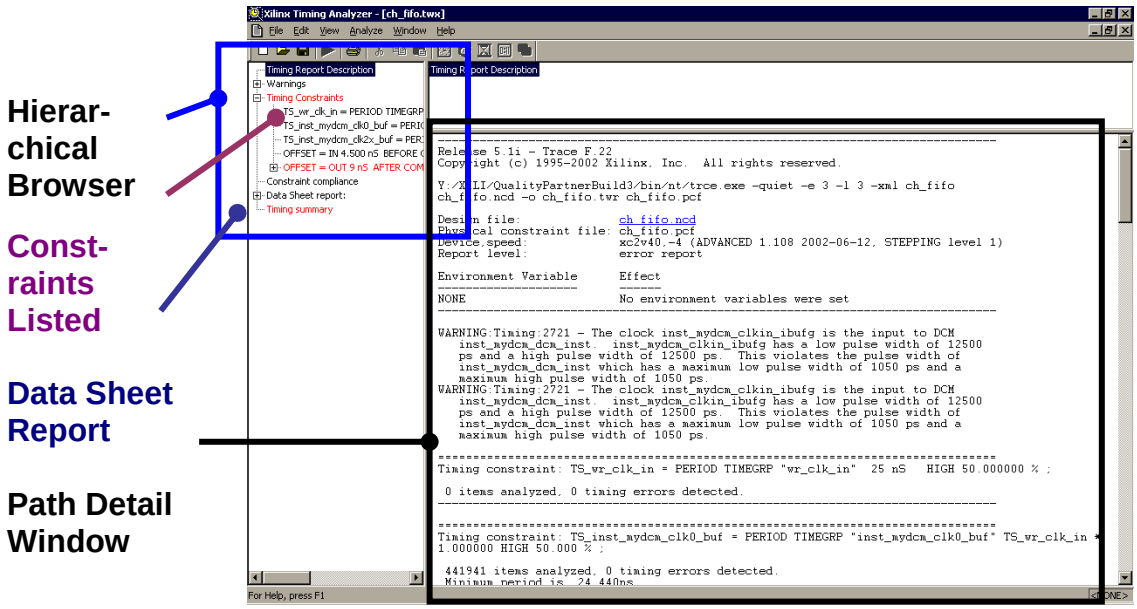


Рис. 2. Графический Интерфейс пользователя утилиты Timing Analyzer

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.

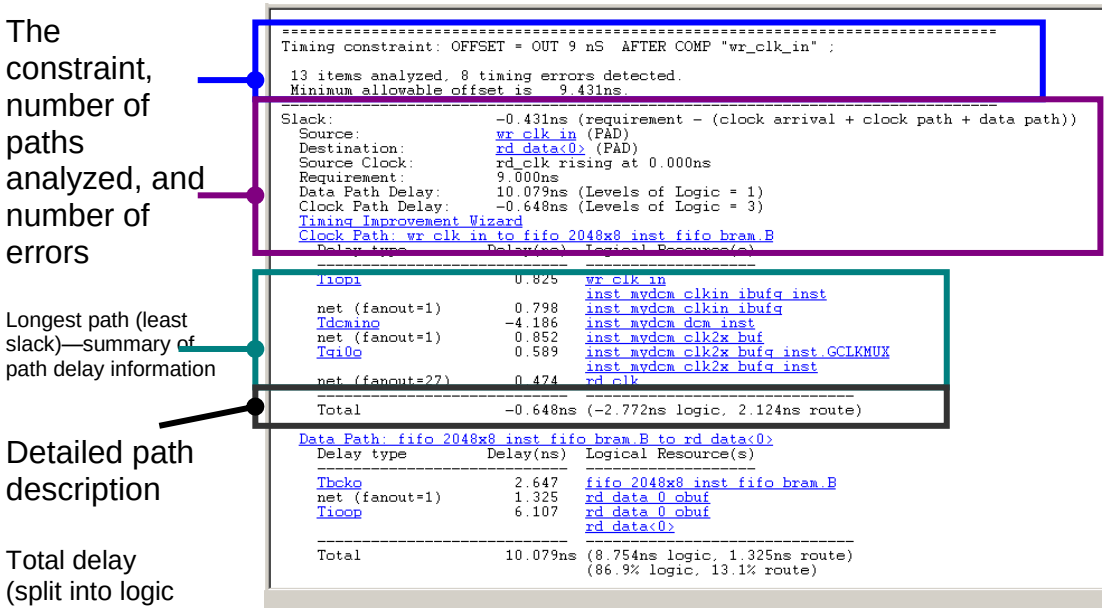


Рис.3. Подробный Анализ

Методика

В этой лабораторной 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**)

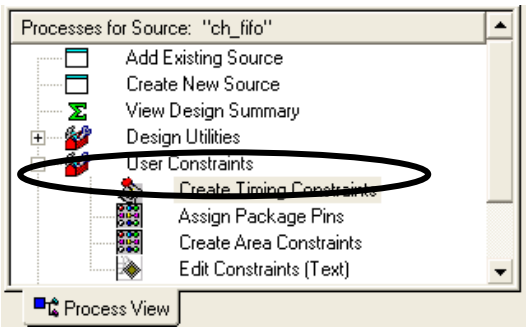


Рис. 4. Окно Процессов

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**).

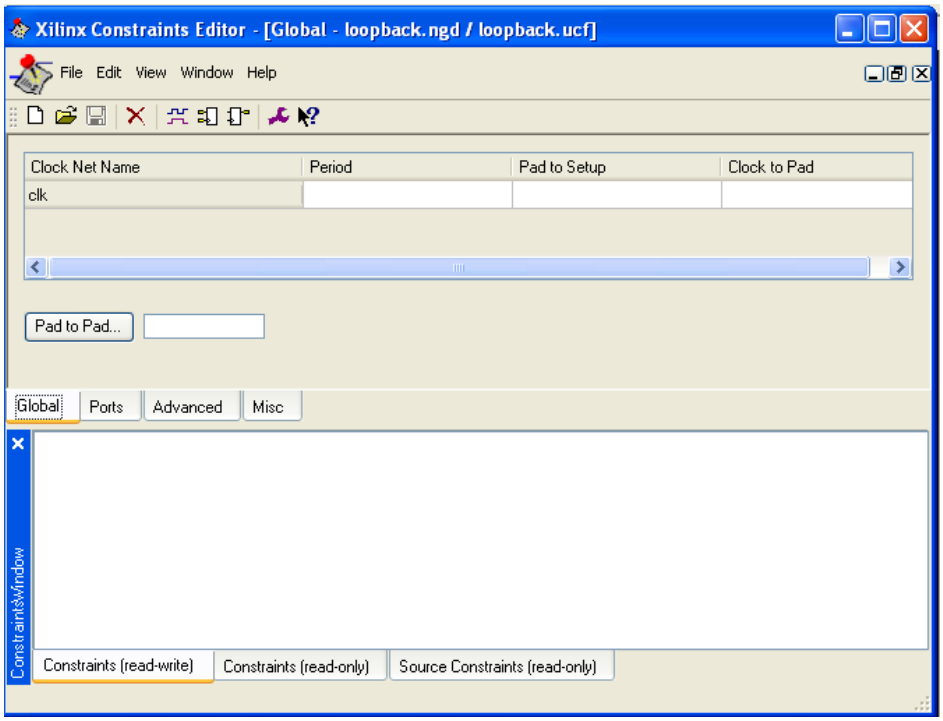


Рис. 5. Закладка Global

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

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

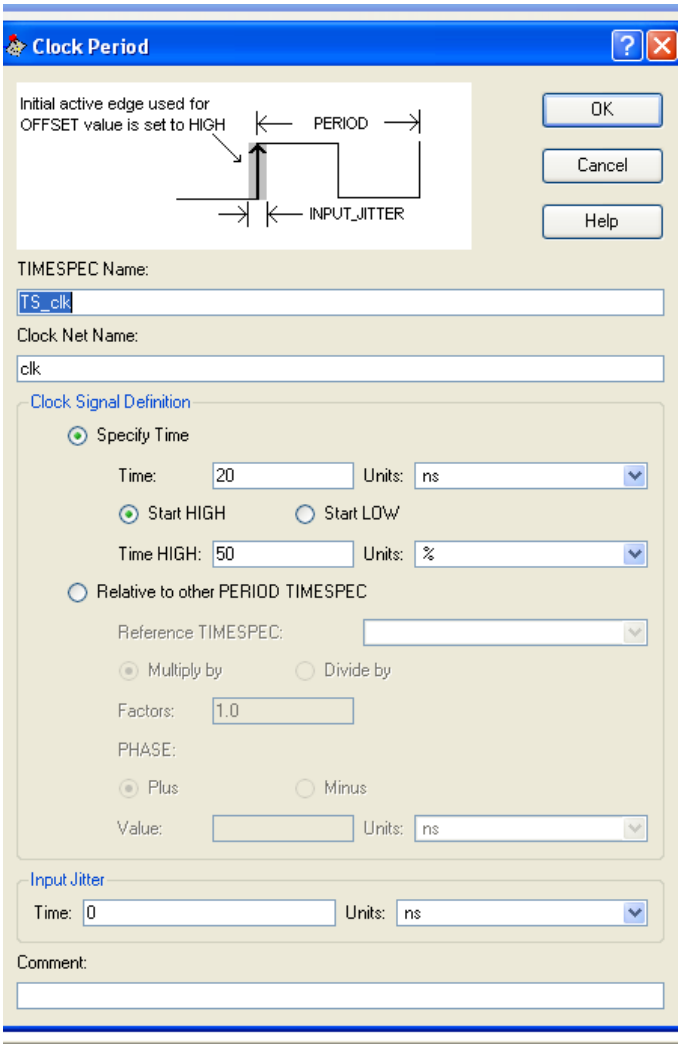


Рис. 6. Диалог установки ограничений

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**

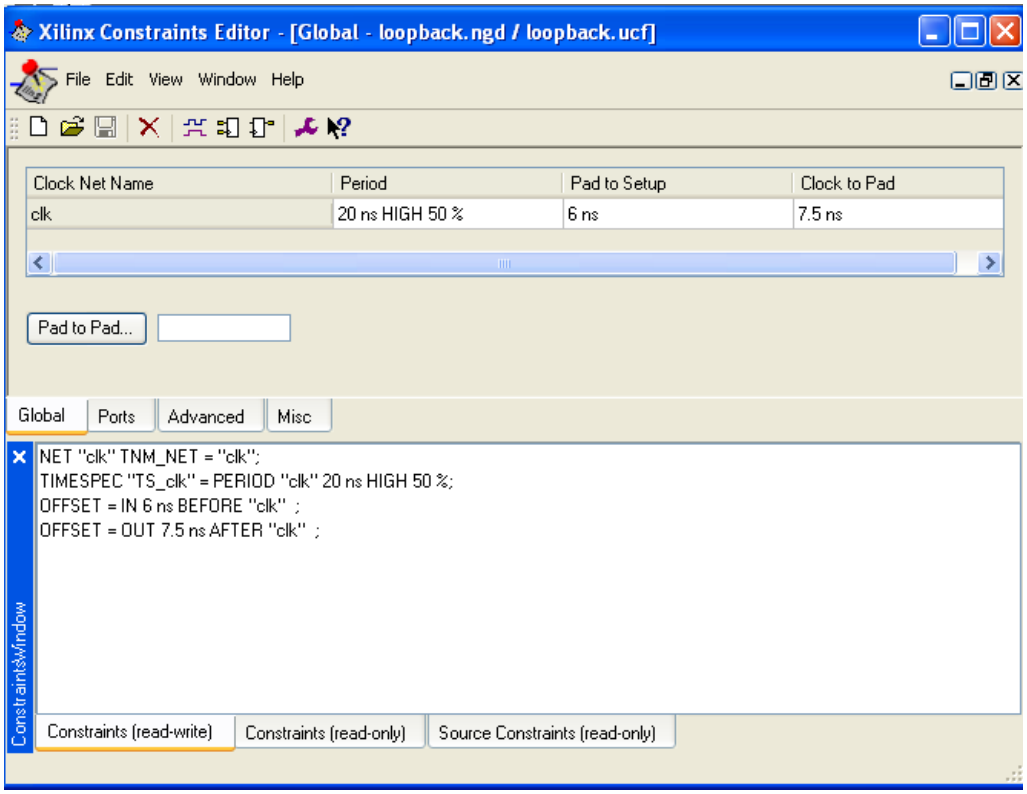


Рисунок 3-7. Закладка Global в редакторе Constraints Editor

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 лабораторной работе вы будете назначать местоположение контактов (Табл. 1) вручную редактируя файл UCF и пользуясь технической документацией на плату Digilent Spartan-3.

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

Табл. 1. Список Вх/Вых контактов

- 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 лабораторной работы.

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 лабораторной работы.

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.

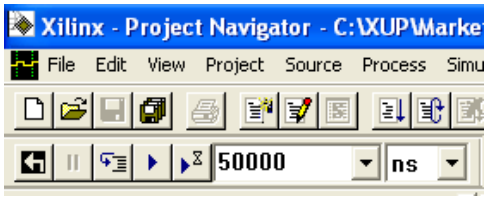


Рис. 8. Установка времени окончания симуляции

❻ Проанализируете результаты симуляции и закройте окно как только закончите.

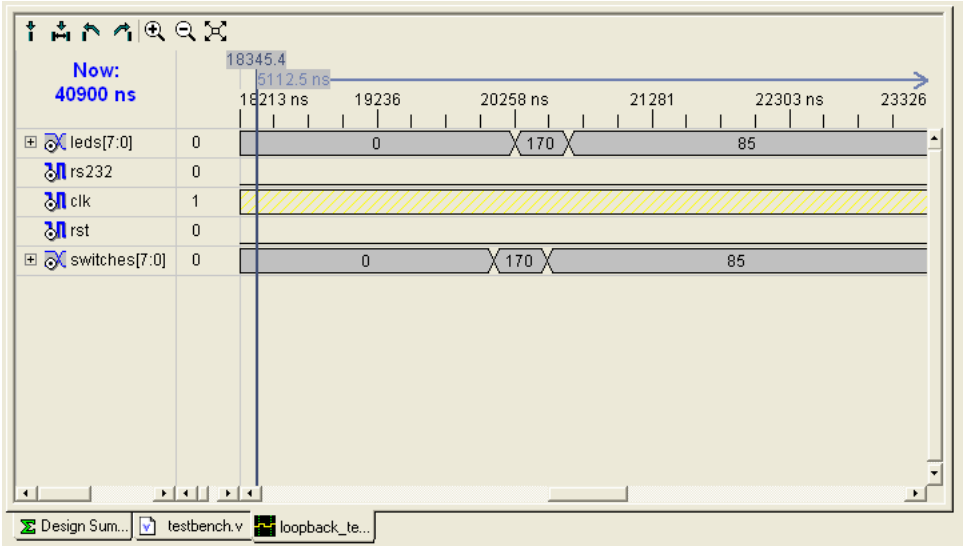


Рис. 9. Результаты Behavioral Simulation

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).

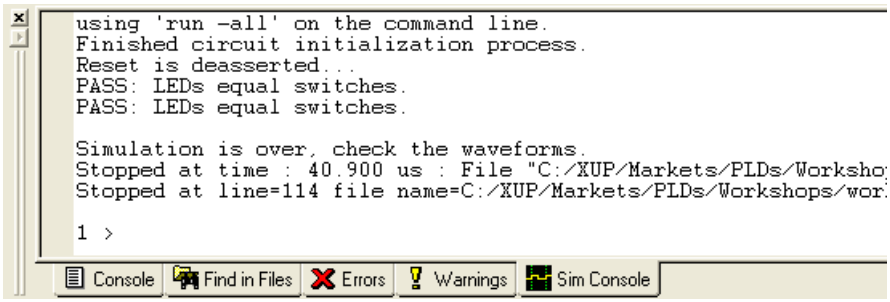


Рис. 10. Просмотр сообщений в Консоли Симулятора

Генерация PROM файла для Flash памяти Шаг 6

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.

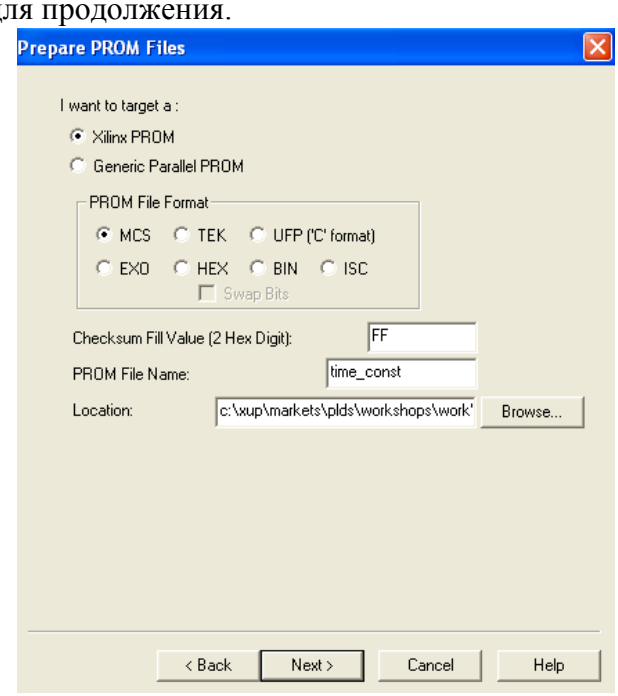


Рис. 11. Подготовка PROM файла

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

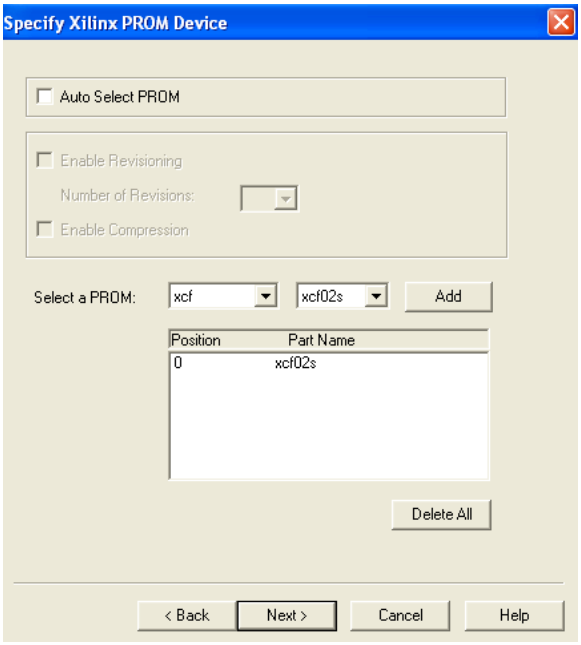


Рис. 12. Установка xcf02s PROM для платы Digilent Spartan-3

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

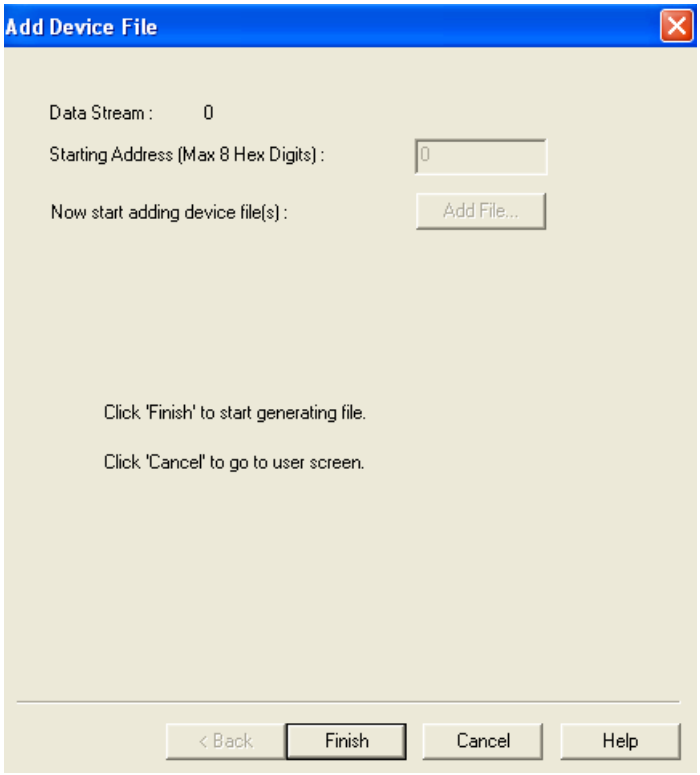


Рис. 13. Добавление bit файла в проект

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

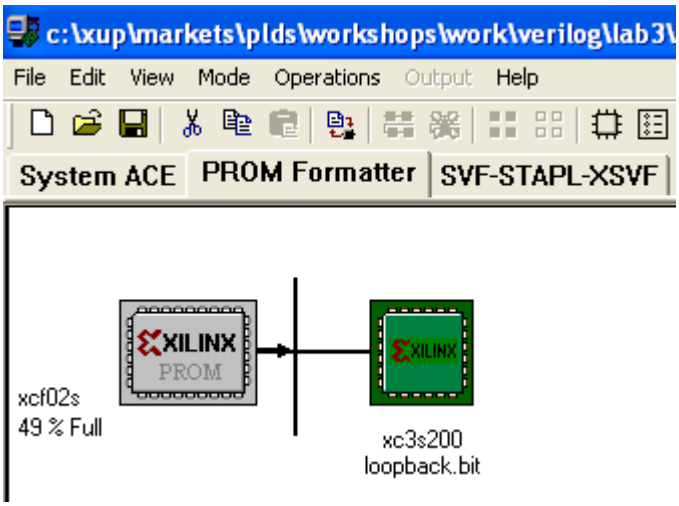


Рис. 14. PROM файл теперь отформатирован

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.

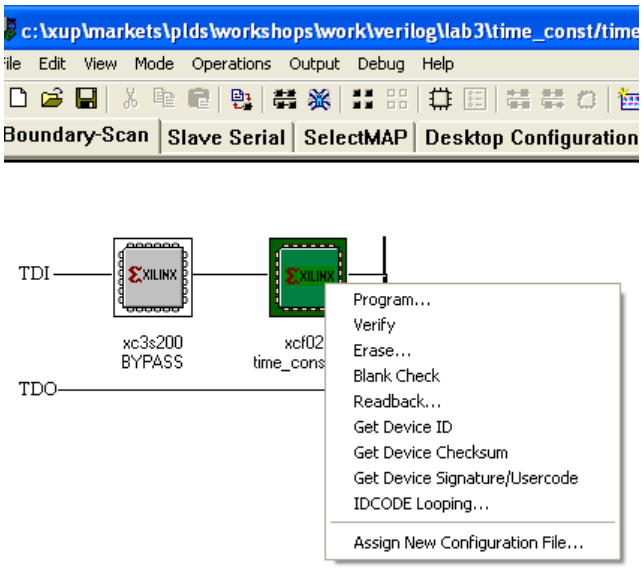


Рис. 15. Программированиеxcf02s Platform Flash PROM

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 лабораторной 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

Лабораторная работа №4. Синтез (XST)

Введение

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

Цели

По завершению этой лабораторной 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 синтезирования 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.

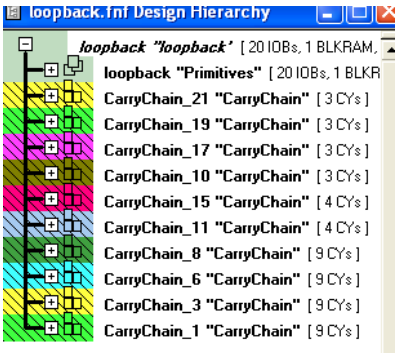


Рис. 1. Просмотр Иерархии Дизайна в Floorplanner

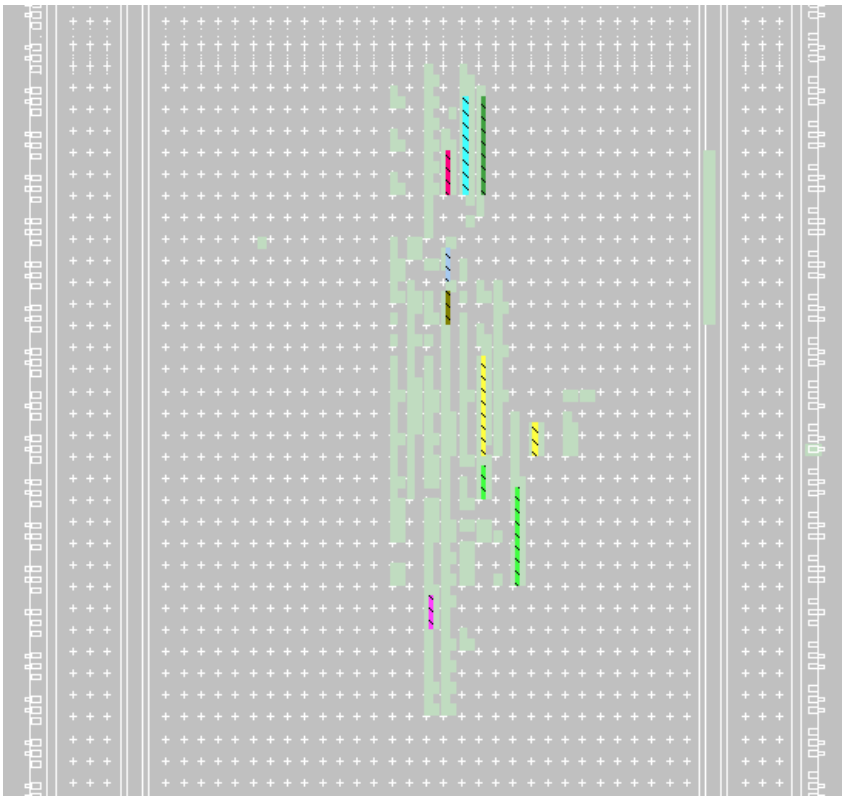


Рис. 2. Топология разбросанного дизайна

8. 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**

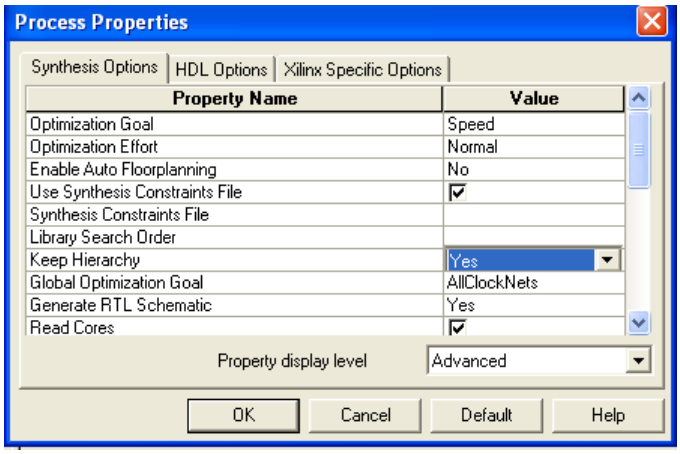


Рис. 3. Свойства Процесса Synthesis

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>

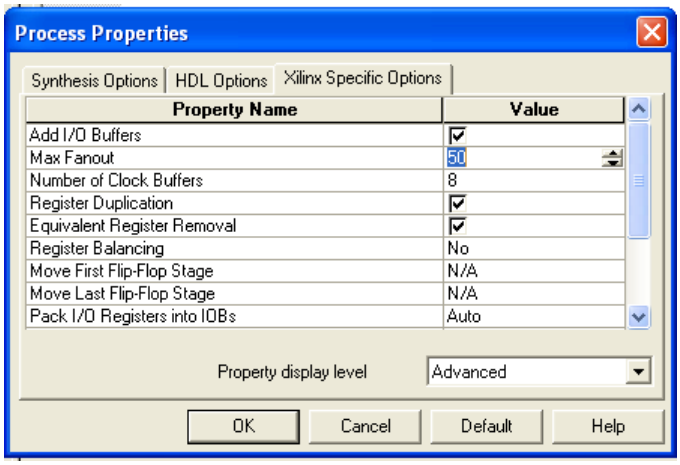


Рис. 4. Специальные установки

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.

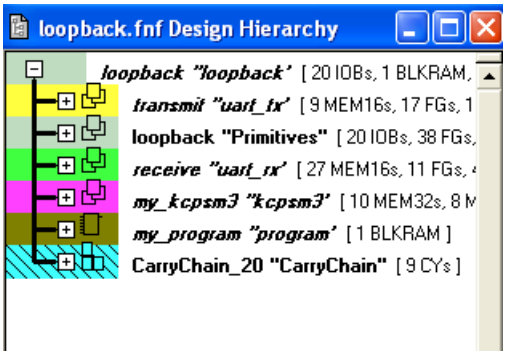


Рис. 5. Просмотр Иерархии Дизайна в Floorplanner

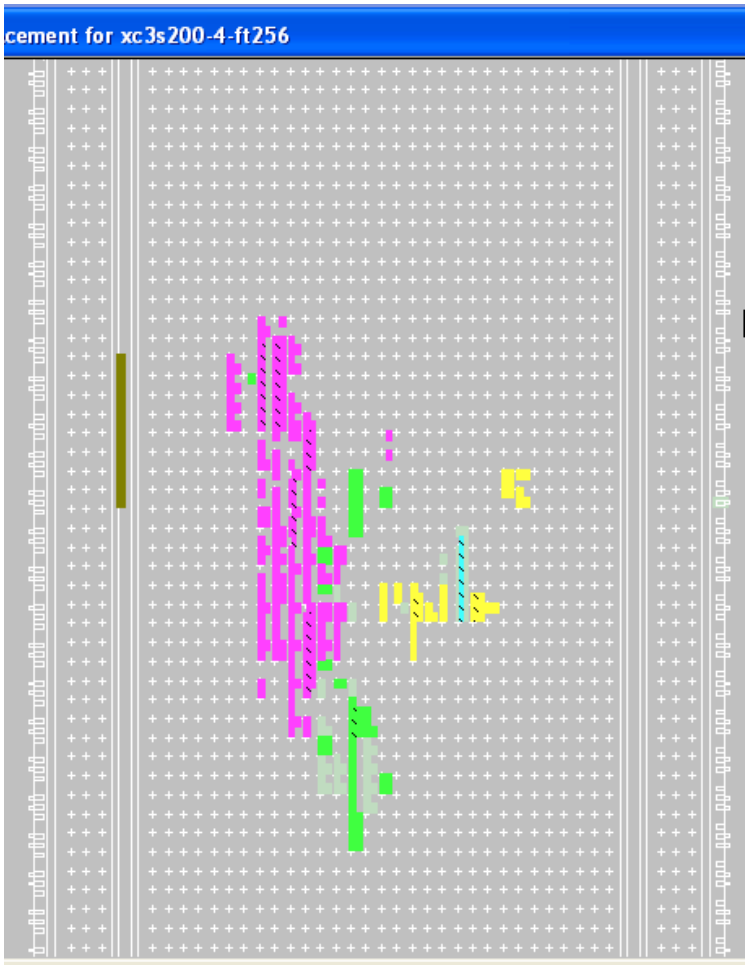


Рис. 6. Топология иерархического дизайна

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

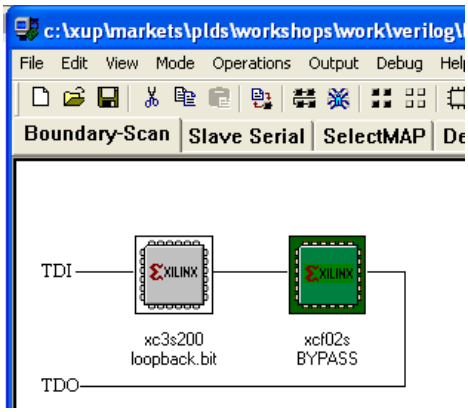


Рис. 8. Назначение конфигурационного файла

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.

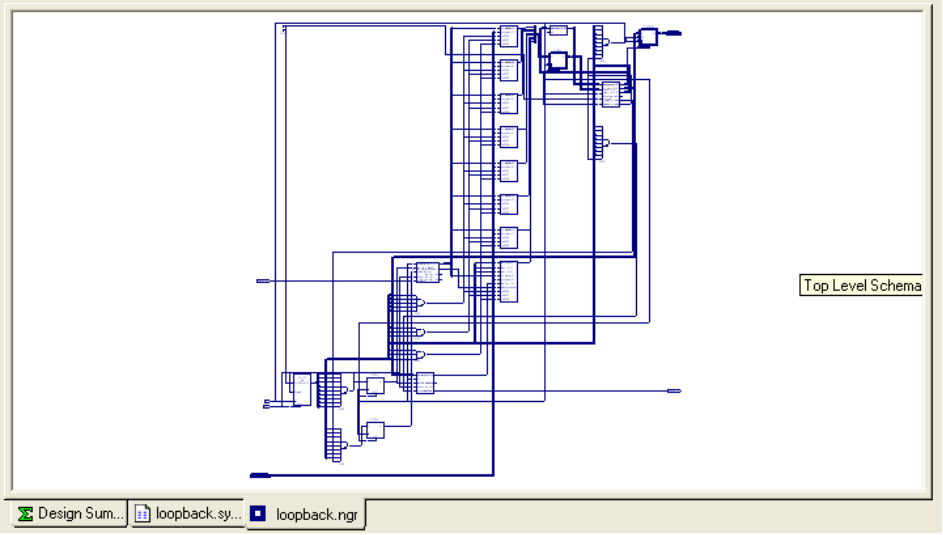


Рис. 7. Просмотр RTL-схемы

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, нобольше 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)

Лабораторная работа №5. CORE Generator System

Введение

Лабораторная 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 лабораторной 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 лабораторной 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 лабораторная 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**

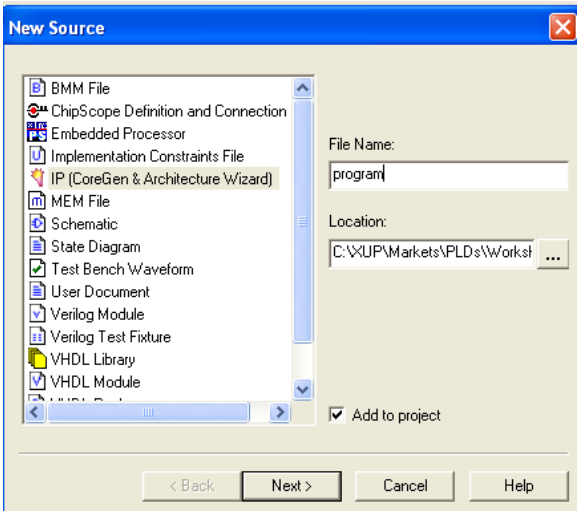


Рис. 1. Диалог создания нового объекта

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**

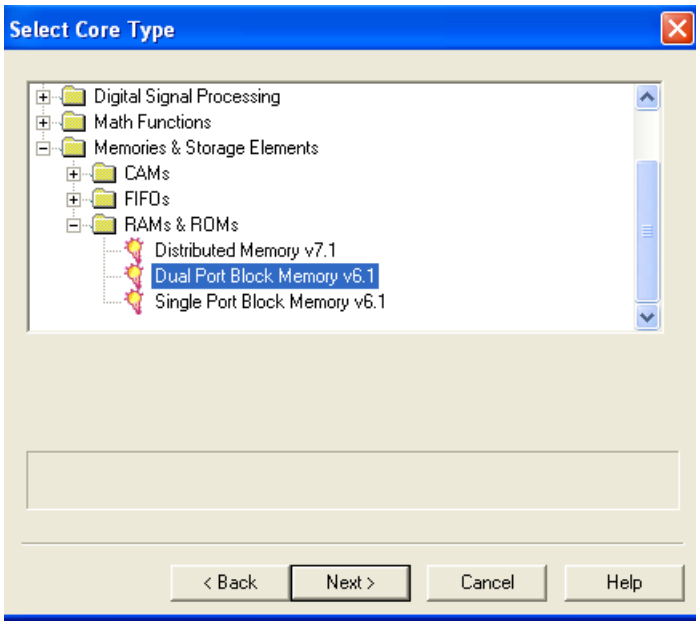


Рис. 2. Диалог Выбора Типа IP Core

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**

❶ 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**

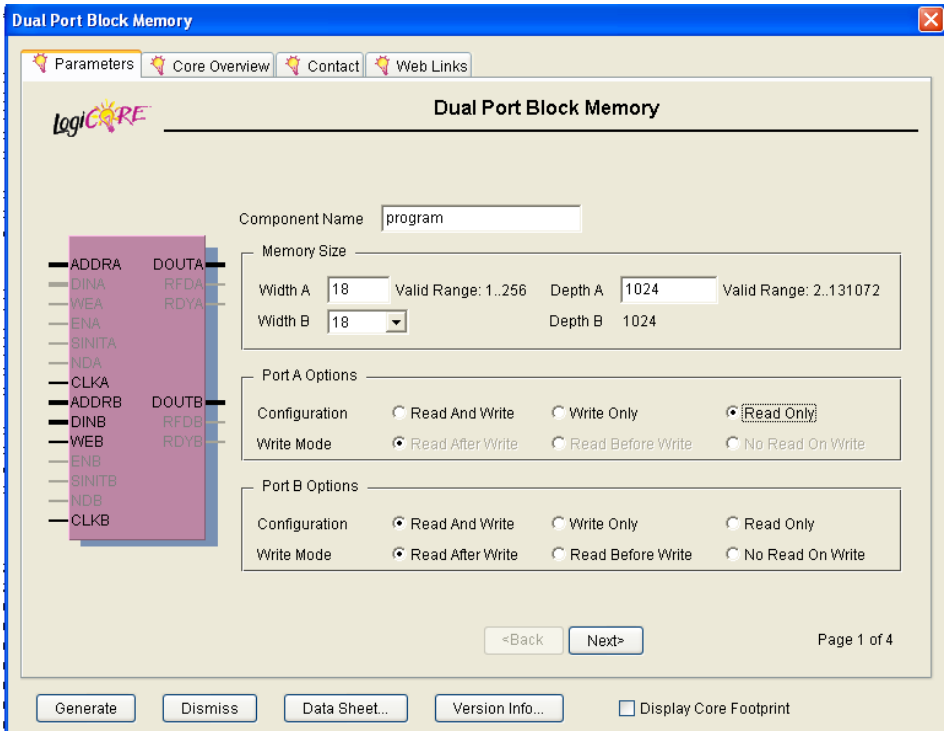


Рис. 3. Установки при Создании Блок Памяти

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

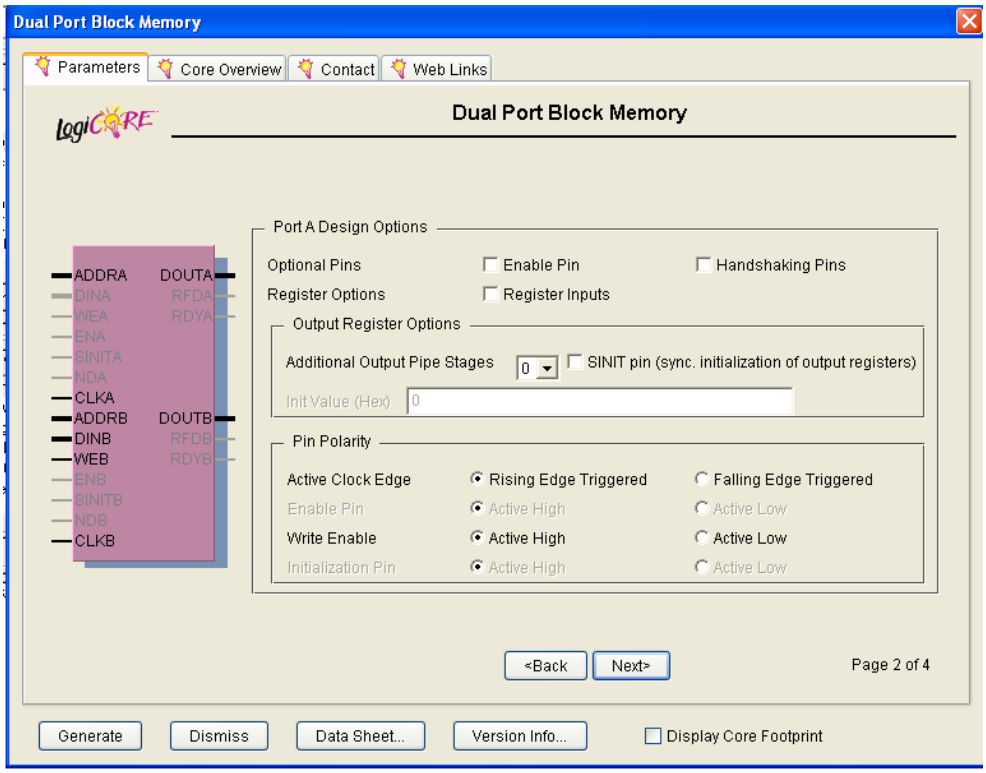


Рис. 4. Опции дизайна и полярность выводов для порта A

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

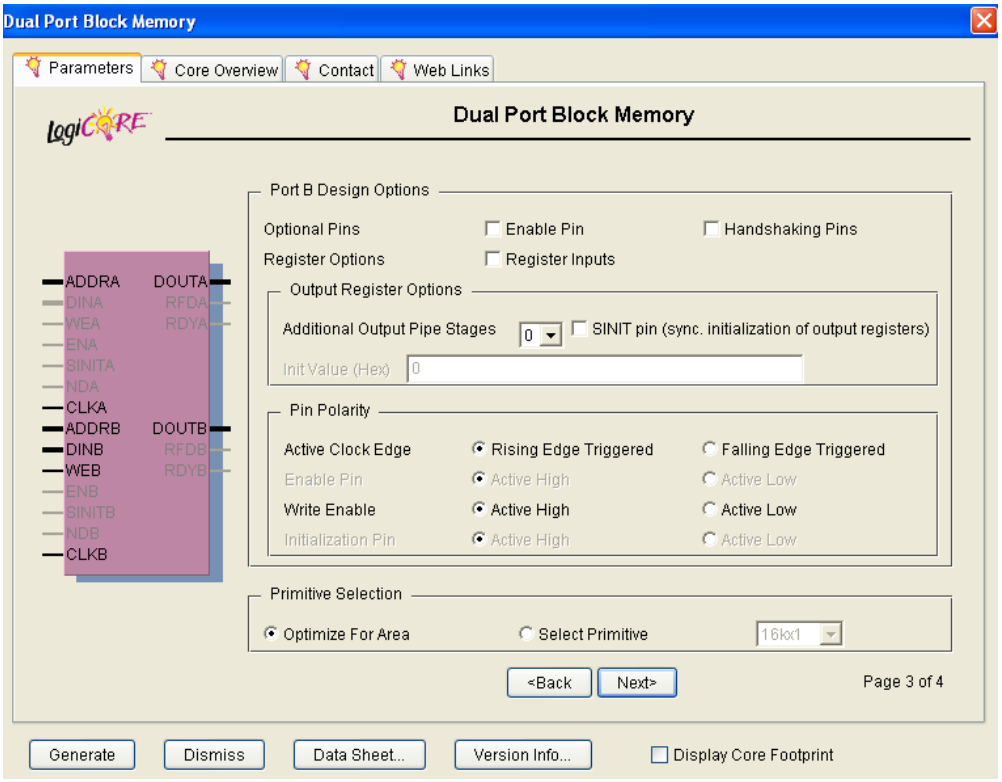


Рис. 5. Опции дизайна и полярность выводов для порта B

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

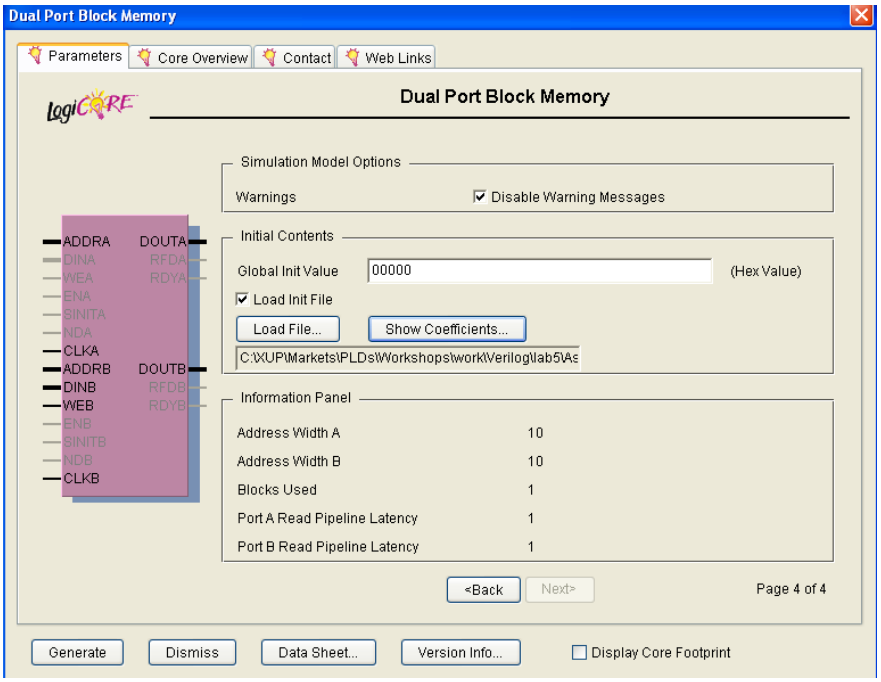


Рис. 6. Инициализация памяти с программой

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**

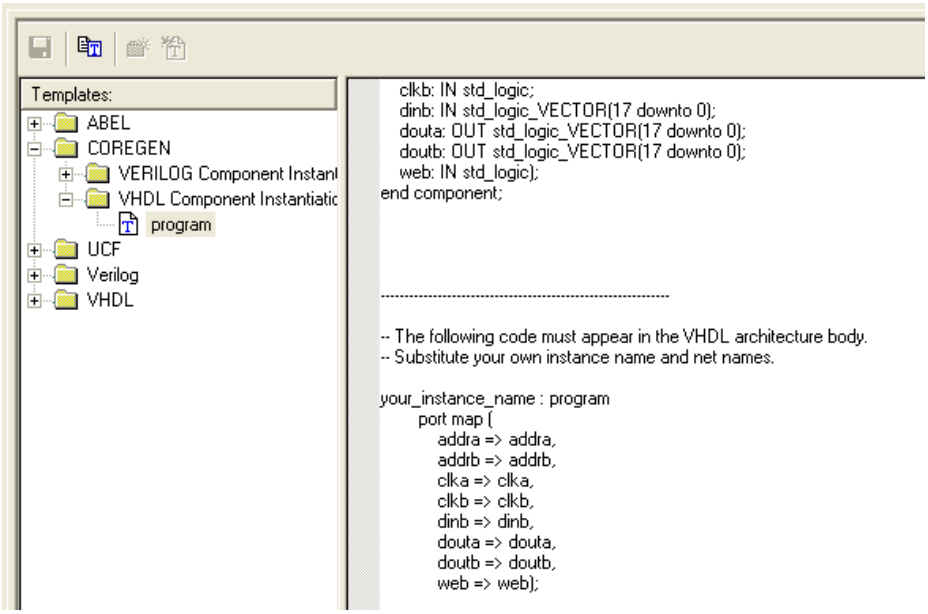


Рис. 7. Шаблон для реализации

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
- Automatcially 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.

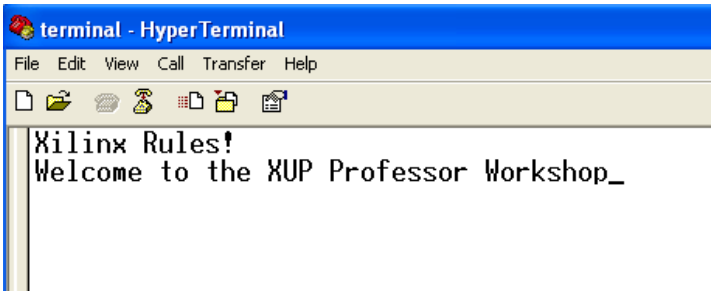


Рис. 9. Вид окна гипертерминала

Заключение

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