

# 1 Введение в Xilinx ISE WebPack версии 10.1i

## 1.1 Внешний вид оболочки

Данное пособие частично знакомит вас с технологией проектирования цифровых устройств на ПЛИС фирмы Xilinx с использованием свободно распространяемого пакета WebPACK™ ISE™ (Integrated Synthesis Environment). Несмотря на то, что этот пакет является бесплатным, он представляет собой полнофункциональную систему автоматизированного проектирования (САПР), которая позволяет выполнить все этапы разработки, начиная с создания проекта и заканчивая программированием кристалла. Данная продукция доступна в интернете: <http://xilinx.com/webpack/>.

Работа с пакетом начинается с запуска Навигатора проектов. Для этого следует запустить приложение  Project Navigator. Рассмотрим управляющую оболочку WebPACK ISE (рис.1.1.1).

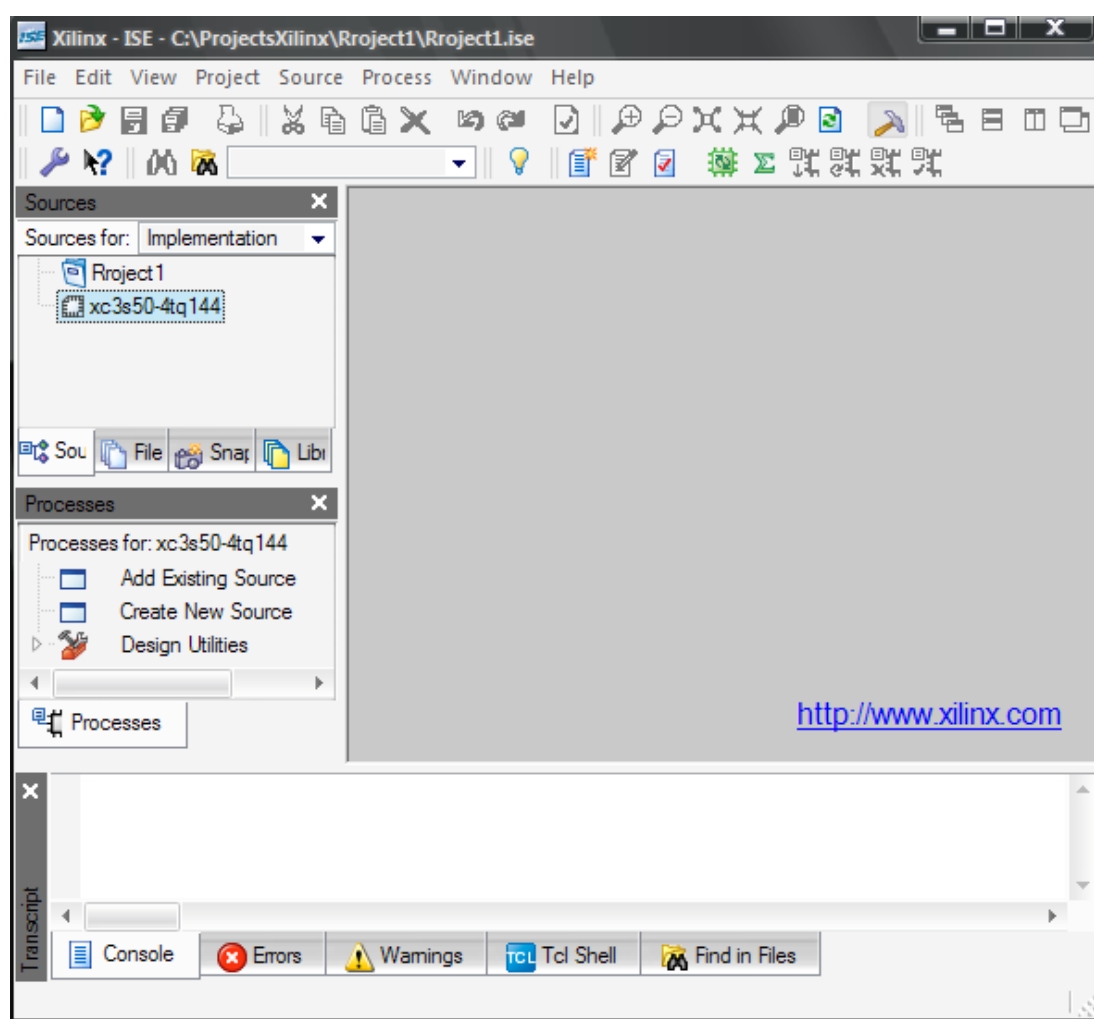


Рисунок 1.1.1.

Сверху на шапочке находится заголовок окна. Под ним, на уровень ниже находится главное меню. Главное меню управляющей оболочки пакета обеспечивает доступ ко всем командам, необходимым для работы с проектом. Каждый пункт главного меню открывает всплывающее меню, в котором находится соответствующая группа команд.

Под главным меню обычно по умолчанию располагается оперативная панель управления. Оперативная панель управления содержит кнопки быстрого доступа, которые дублируют наиболее часто используемые команды Навигатора проекта. Состояние кнопок оперативной панели зависит от типов модулей и процедур, выделенных в окнах исходных модулей и процессов в текущий момент времени. Кнопки, соответствующие недопустимым операциям по отношению к выделенным модулям и процедурам, отображаются серым цветом. Таким образом, обеспечивается необходимая защита от ошибок в процессе проектирования.

Слева-сверху (см. рис.2.1.1) располагается окно исходных модулей (sources). В нем находятся исходные модули, относящиеся непосредственно к проекту. Здесь же можно наблюдать иерархию различных объектов вашего проекта. Исходный модуль, обозначенный на рисунке как xc3s50-4tq144 (модуль устройства), можно открыть двойным кликом мыши. В этом случае откроется окно свойств устройства, эти свойства можно изменить в любой удобный для пользователя момент времени (делать это не рекомендуется, т.к. это может повлиять на подкрепленные к данному источнику модули). Этот модуль является основным, он – своего рода виртуальный кристалл, на который вы накладываете содержимое – различные объекты. Сверху окна исходных модулей находится всплывающий список – отображающий исходные модули для: выполнения, моделирования поведенческого уровня и постпригодного моделирования (source for: Implementation, Behavioral simulation, Post-Fit simulation). Этот список определяет, какие модули отображаются в данном окне. Нарисованная в схемотехническом редакторе схема при сохранении «распадается» на несколько модулей, один – для отображения структуры схемы, другой – для описания поведения схемы в режиме реального времени, задержек схемы и т.п. Всех их одновременно вы не сможете увидеть в данном окне, только при переключении режимов выбора отображаемых исходных модулей. Однако есть модули, одинаковые для всех трех режимов просмотра, например модуль вашего «виртуального кристалла». Снизу окна исходных модулей располагаются закладки: Sources, Files, Snapshot, Libraries. Из этого в рамках учебной программы вам понадобятся только две вкладки: позволяющие видеть исходные модули и файлы. Все модули, созданные ранее можно запускать для отладки, а также свершать над ними различные

действия. Для выполнения особых действий над модулями предусмотрено окно процессов (Process), на рисунке – под окном исходных модулей.

Окно процессов представляет собой эффективный визуальный механизм управления всем ходом проектирования устройства. В этом окне отображается поэтапная последовательность действий (операций), которая должна быть выполнена по отношению к модулю, выделенному в окне исходных модулей проекта. В окне процессов разработчику предоставляется подробный интерактивный план работы с проектом. Содержание окна процедур зависит от типа выделенного исходного модуля и семейства ПЛИС, выбранного для реализации проекта. Информация о маршруте проектирования устройства может быть представлена в сжатом или развернутом виде. При сжатой форме отображения данных в окне процессов показаны лишь основные этапы проектирования. Развернутая форма имеет вид ветвящейся Иерархической структуры, каждый узел которой представляет определенную фазу соответствующего этапа проектирования. Ветви структуры содержат информацию не только о процедурах проектирования, но и о возможности использования дополнительных инструментов, интегрированных с пакетом WebPACK ISE. Для перехода от сжатого формата к подробному необходимо последовательно развернуть каждый узел структуры. Навигатор проекта предоставляет вам три способа управления процессами проектирования. Для первых двух способов необходимо выделить требуемую процедуру в окне процессов, поместив курсор на строку с ее названием и щелкнув левой кнопкой мыши. Далее можно использовать пункт Process главного меню и затем команды соответствующего всплывающему меню. Второй способ заключается в использовании команд всплывающего контекстно-зависимого меню, активизируемого щелчком правой кнопкой мыши. Контекстно-зависимое меню в этом случае содержит команды, дублирующие часто используемые пункты всплывающего меню Process.



*Самый простой и быстрый способ запуска процедур - двойной щелчок левой кнопкой мыши на строке с названием соответствующей процедуры в окне процессов.*

В самом низу располагается окно консольных сообщений. По аналогии с другими компиляторами, в ней высвечиваются сообщения о рисках, ошибках, сведения о состоянии процесса и т.п.

В центре находится панель размещения редакторов (HDL языка, схемотехнического и тестового).



*На некоторых компьютерах панель размещения редакторов не отображается. Разработчиками данного методического пособия не были выявлены причины данной неисправности.*

*Возможно вы столкнетесь с данной проблемой и увидите примерно следующую картину (рис.1.1.2):*

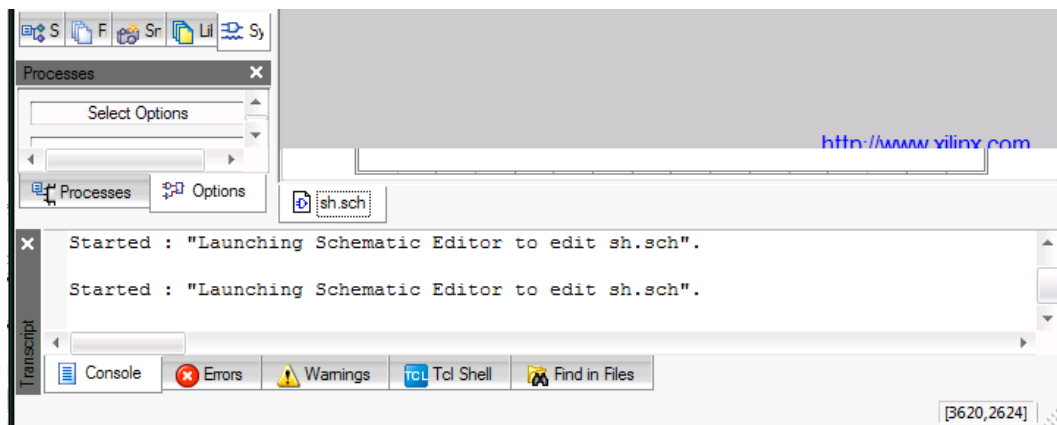



Рисунок 1.1.2.

*Виден только фрагмент редактора, а иногда он вообще отсутствует (на рисунке – отображается фрагмент открывшегося источника sh.sch). Решение проблемы заключается в вынесении редактора в отдельное окно (пиктограмма  в панели оперативного управления) или замены темы оформления операционной системы.*

### **1.1.2 Этапы проектирования в системе проектирования Xilinx ISE10.1i**

Пакет ISE WebPack представляет собой систему сквозного проектирования, которая реализует полный цикл разработки цифровых устройств на основе ПЛИС, включающий этапы создания исходного описания проекта, его синтеза, моделирования, размещения и трассировки, программирования самого кристалла.

Данный пакет поддерживает ПЛИС фирмы Xilinx следующих семейств: Virtex-4 FX, Virtex-4 LX, Virtex-4 SX, Virtex-II Pro, Virtex-II, Virtex-E, Virtex; Spartan-3E, Spartan-3/3L, Spartan-IIe, Spartan-II; CoolRunner-II, CoolRunner XPLA3, XC9500XV, XC9500XL, XC9500.

Основные модули пакета ISE WebPack: редактор схемотехнического ввода; интеллектуальные средства создания HDL-описаний, формирующие шаблоны на основании информации, представляемой пользователем, для языка описания аппаратуры VHDL; модуль генерации тестовых воздействий для программы моделирования HDL Benchner; программа функционального и временного моделирования ModelSim; программа автоматического размещения и трассировки ПЛИС; редактор диаграмм состояний StateCAD; iMPACT - программа загрузки конфигурационной последовательности в ПЛИС FPGA и программирования ПЛИС CPLD и ППЗУ.

Xilinx Embedded Development Kit (EDK) - интегрированный программный пакет для сквозной разработки встраиваемых программируемых процессорных систем на базе ПЛИС Xilinx. Пакет включает программное средство Platform Studio, а также необходимую документацию и IP-ядра, которые могут потребоваться для разработки встраиваемых систем на основе FPGA фирмы Xilinx с встроенными аппаратными ядрами процессора PowerPC и/или софт-процессорами MicroBlaze. Для работы пакета EDK необходим пакет ISE Foundation или ISE WebPack соответствующей версии.

Основные модули в составе пакета:

1. Xilinx Platform Studio (XPS): -графический редактор управления файлами проекта; - интерфейс к подпрограммам EDK и ISE;
2. средства разработки программного обеспечения: -GNU C/C++ компилятор программ для процессоров MicroBlaze и PowerPC; GNU - отладчик для процессоров MicroBlaze;
3. IP-ядра периферийных шин и устройств для процессоров MicroBlaze;
4. IP-ядро процессора MicroBlaze;
5. VHDL модели процессоров и IP-ядер для использования при симуляции.

Проектирование цифровых устройств на основе ПЛИС заключается в выполнении следующих этапов в системе проектирования:

1. Создание принципиальной схемы проектируемого устройства в схемотехническом редакторе Xilinx Ise Design Suite 10.1 (FPGA Editor) или описании данного устройства на языке VHDL или Verilog.
2. Предварительное функциональное (Behavioral Simulation) или временное моделирование для выявления ошибок и проверки работоспособности проектируемого проекта или отдельных его частей.
3. Привязка выводов проекта к входам-выводам кристалла, выбор выходных уровней, критичных цепей (Constraints Editor) и т. д.
4. Запуск автоматизированного размещения проекта в кристалле и анализ генерируемых отчетов для выявления предупреждений и ошибок (Implement Design), а при отсутствии таковых и не критичных переходим к следующему этапу.
5. Верификация проекта, т. е. окончательное временное моделирование (Post-Fit Simulation) после размещения проекта в кристалле при всех реальных задержках распространения сигналов внутри микросхемы ПЛИС.
6. Конфигурирование кристалла ПЛИС с помощью битового потока (iMPACT 10.1i).

Маршрут проектирования цифрового устройства с применением ISE Web PACK версии 10.1i представлен на вытекает из структурной схемы САПР рис.1.1.2.1

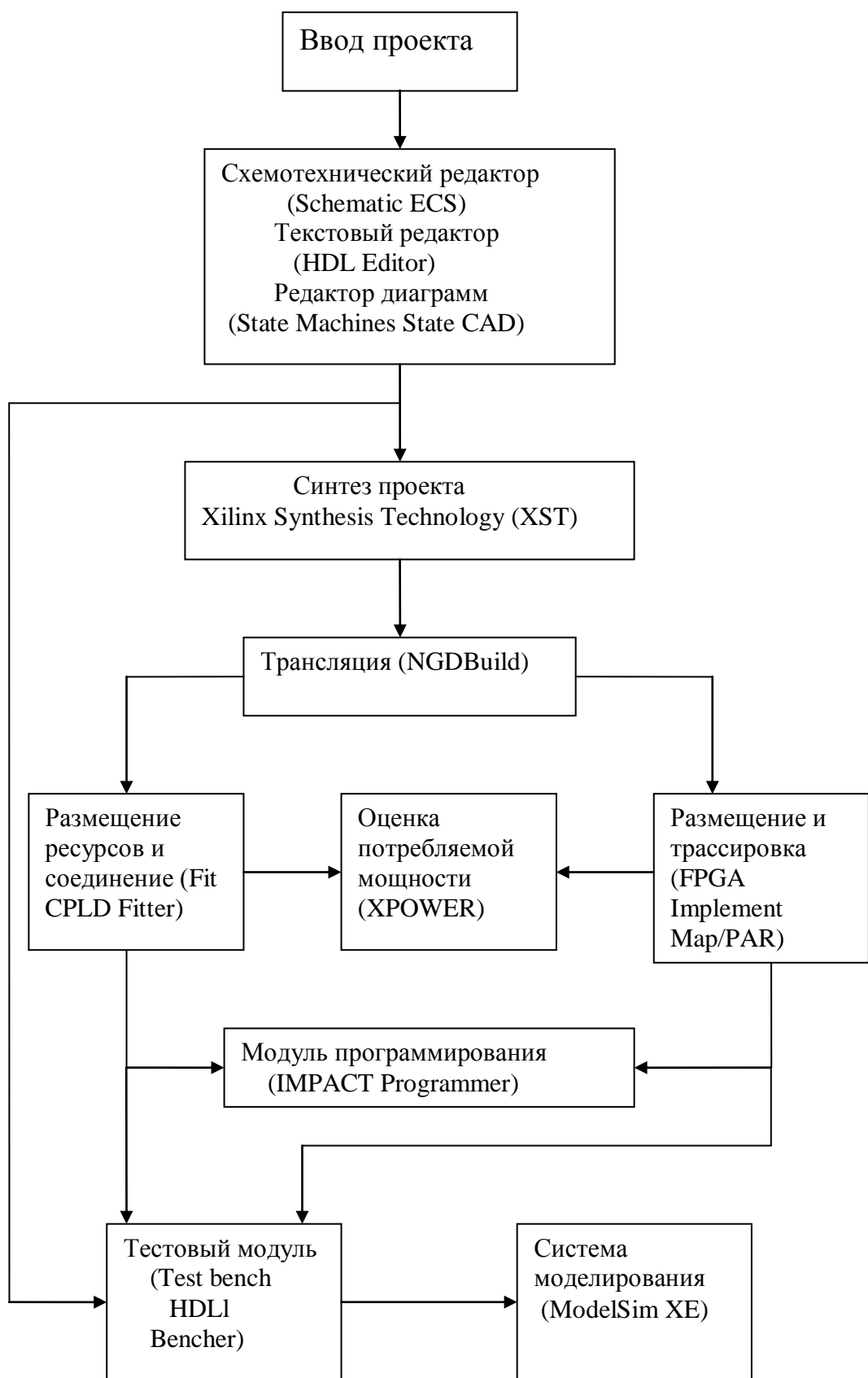


Рисунок 1.1.2.1

Для того, чтобы сконфигурировать ПЛИС необходимо иметь загрузочный JTAG-кабель. Загрузка битового потока осуществляется через специально выделенные конфигурационные выводы с использованием различных способов и режимов загрузки ПЛИС. После удачной загрузки проекта проверяется и отлаживается проект в дальнейшем. В случае необходимости и дальнейшего развития или проектирования проекта на ПЛИС все пройденные этапы повторяются до полного завершения проекта в целом.

## 1.2 Создание проекта

Для создания проекта настоятельно рекомендуется заводить отдельные папки и называть их разными именами. В главном меню управления выберите **File** → **New Project...** Далее вы увидите перед собой окно (рис.1.2.1):

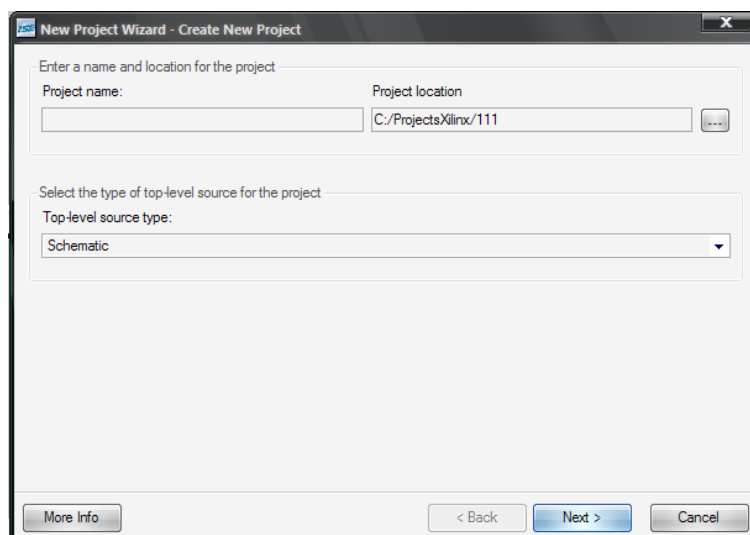


Рисунок 1.2.1

В Top-level source type (тип топ-уровня исходного модуля) выберите schematic (даным действием задается привилегированный режим работы для схематехнических модулей), далее выберите путь, где будет храниться проект в поле Project location.



*Путь и имя проекта должны содержать только английские буквы и цифры. Данный САПР исполнен на английском языке и не распознает других языков. В случае попытки указать путь или имя, содержащие не английские буквы, САПР выдаст сообщение об ошибке.*

Затем укажите имя проекта в поле Project name.



*В имени проекта нельзя использовать пробелы.*

САПР автоматически создает вложенную папку с именем проекта в указанную вами директорию, там будут храниться все файлы проекта. Нажмите Далее (Next).

Далее, если вы сделали все правильно, перед вами появится следующее окно (рис.1.2.2).

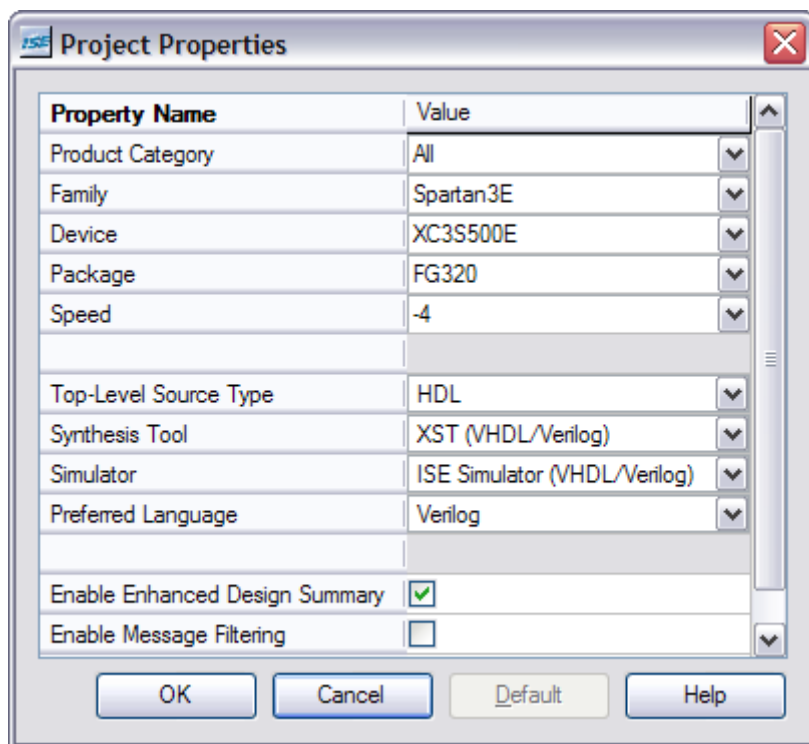


Рисунок 1.2.2.

Установите следующие настройки: в Product Category установите All (данном действием задается выбор использования всех категорий продуктов фирмы Xilinx), семейство кристаллов (Family) установите Spartan 3E (т.к. она более полная и содержит в себе большее число элементов), тип кристаллов (Device) – установите XC3S500E, корпус кристалла (Package) – установите FG320, градацию быстродействия кристаллов (Speed) – установите в (-4). В качестве программы, выполняющий синтез(по описанию строит netlist) – выберите XST (VHDL\Verilog), который поддерживает сразу два языка описания схем: VHDL и Verilog. В качестве моделиатора (программа, которая строит временные диаграммы) выберите ISE Simulator (VHDL/Verilog), как на рисунке. В качестве привилегированного языка (Preferred Language) установите Verilog. Нажмите Далее (Next).

Далее перед вами появится следующее окно (рис.1.2.3).



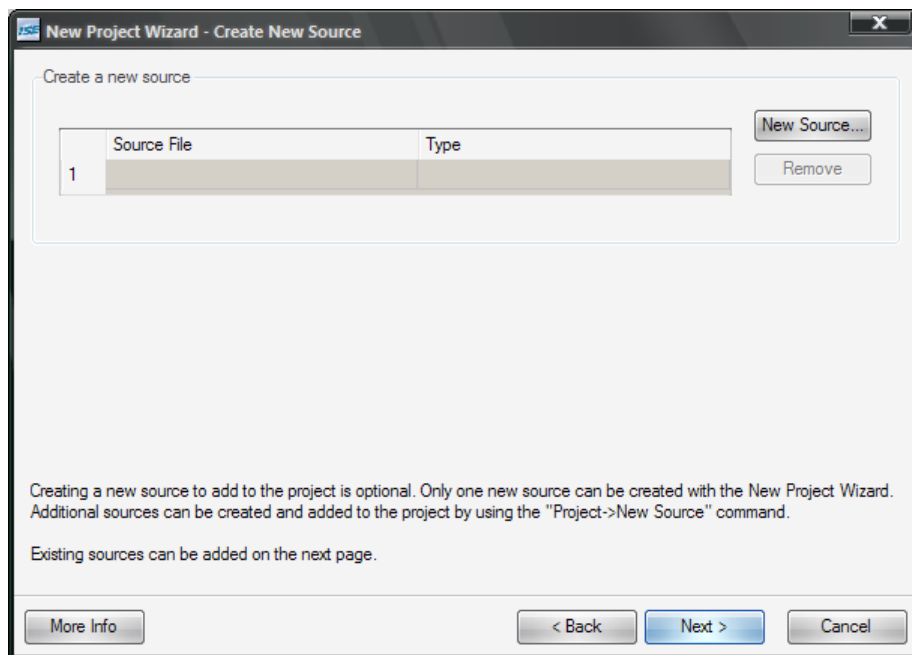


Рисунок 1.2.3.

Вам предлагается сразу создать новый исходный модуль для проекта. Пропустите пока этот момент, к нему вы вернетесь позже. Нажмите Далее (Next).

Перед вами появится следующее окно (рис.1.2.4).

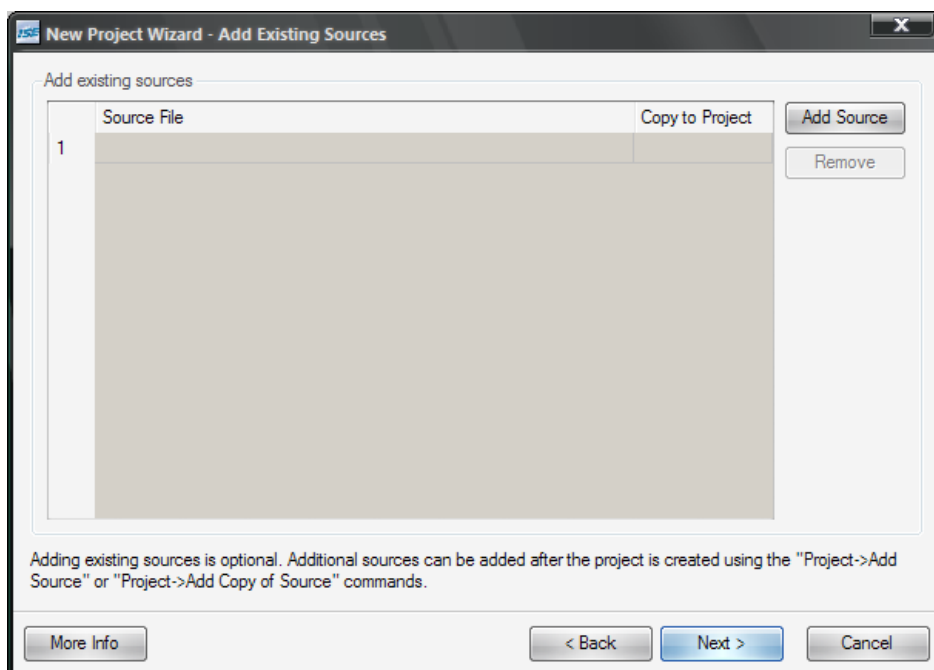


Рисунок 1.2.4.

Здесь вам предлагается копировать исходные модули из другого проекта в текущий. Как и создание объекта, эту процедуру можно будет сделать позже. Пропустите этот момент. Нажмите Далее (Next).

Далее вы должны увидеть следующее окно (рис.1.2.5).

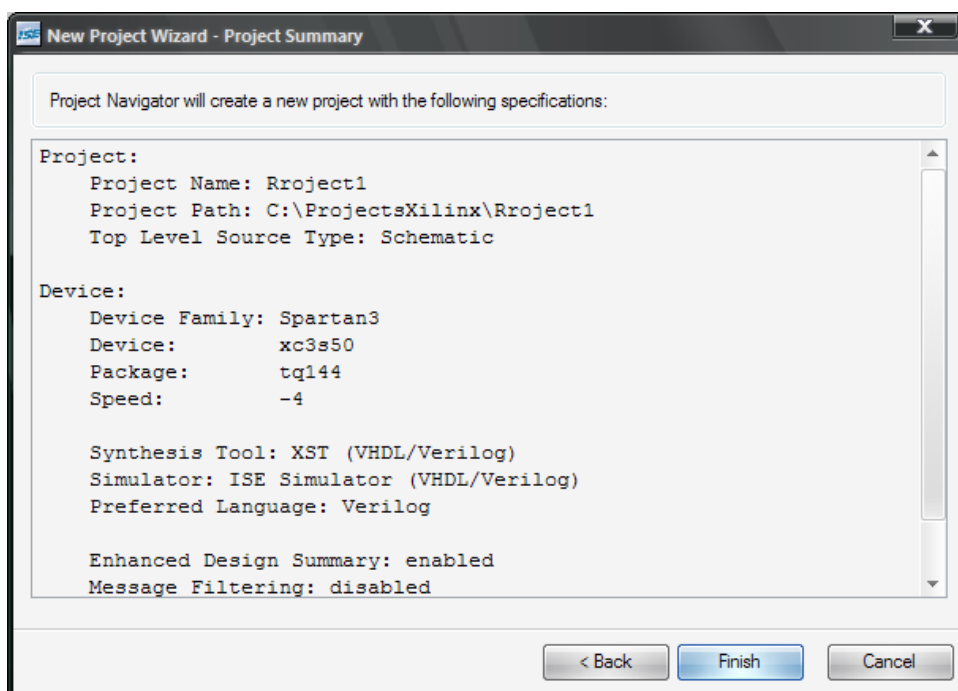


Рисунок 1.2.5.

Здесь вам предоставляется отчет о созданном проекте: имя проекта, путь, серия и т. Нажмите **Завершить** (Finish). Проект создан.

### 1.3 Работа в схемотехническом интерфейсе

Схемотехнический режим проектирования схем является наглядным и легким в освоении. С его помощью вы можете реализовывать схему так, как вам вздумается, практически без ограничений. Также, с его помощью вы можете ставить эксперименты, узнавать свойства неизвестных вам элементов и т.п.

Итак, вам предлагается начать знакомство схемотехнического интерфейса на примере простой конкретной задачи: реализовать схему преобразователя в код Грея.

Что же из себя представляет код Грея? Код Грея (рефлексный двоичный код) — двоичная система нумерования, в которой два соседних значения различаются только в одном двоичном разряде. Изначально предназначался для защиты от ложного срабатывания электромеханических переключателей. Сегодня коды Грея широко используются для упрощения выявления и исправления ошибок в системах связи, а также в формировании сигналов обратной связи в системах управления. 4-х битный код Грея представлен таблицей (табл.1.3.1):

№ п/п	Двоичный код				Код Грея			
	X1	X2	X3	X4	Y1	Y2	Y3	Y4
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1

2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0
5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	1	1	1	1
11	1	0	1	1	1	1	1	0
12	1	1	0	0	1	0	1	0
13	1	1	0	1	1	0	1	1
14	1	1	1	0	1	0	0	1
15	1	1	1	1	1	0	0	0

Таблица 1.2.1.

Наиболее простой способ реализации схемы преобразователя – это использование элементов памяти. Этот способ не только самый простой, но и самый компактный. Для реализации схемы вам понадобится 4 элемента Rom16x1 (4 входа адресуют одну одноразрядную ячейку памяти, таким образом, каждый Rom хранит состояния для определенного разряда).

Для начала необходимо запрограммировать память, а для этого необходимо сделать заготовку. Программируется память с помощью INIT (сокращение от инициализации). INIT представляет собой 4 разрядный 16-ричный (hex) код. Далее приведена процедура нахождения INIT'ов (рис.1.3.1):

№ п/п	Двоичный код				Код Грея				INIT			
	X1	X2	X3	X4	Y1	Y2	Y3	Y4	Rom1	Rom2	Rom3	Rom4
0	0	0	0	0	0	0	0	0	0	0	C	6
1	0	0	0	1	0	0	0	1				
2	0	0	1	0	0	0	1	1				
3	0	0	1	1	0	0	1	0				
4	0	1	0	0	0	1	1	0	0	F	3	6
5	0	1	0	1	0	1	1	1				
6	0	1	1	0	0	1	0	1				
7	0	1	1	1	0	1	0	0				
8	1	0	0	0	1	1	0	0	F	F	C	6
9	1	0	0	1	1	1	0	1				
10	1	0	1	0	1	1	1	1				
11	1	0	1	1	1	1	1	0				
12	1	1	0	0	1	0	1	0	F	0	3	6
13	1	1	0	1	1	0	1	1				
14	1	1	1	0	1	0	0	1				
15	1	1	1	1	1	0	0	0				

Рисунок 1.3.1.

Порядок построения INIT'ов: для начала необходимо убедиться, что выходные значения (на рисунке **Y**) расположены по порядку (на рисунке: аргументу **X** соответствует значение **Y**, **X**'ы расположены строго по возрастанию от 0 до 15 – это существенно). Затем разбиваются выходные значения по столбцам на группы по 4 значения так, как это показано на рисунке (на рисунке группы объединены овалом). Эти группы являются тетрадрами, и если перевести их в 16-ричный код (hex-код), считывая их **СНИЗУ-ВВЕРХ**, то получится INIT. Итак:

Rom1 имеет INIT: 1111 1111 0000 0000 → FF00

Rom2 имеет INIT: 0000 1111 1111 0000 → 0FF0

Rom3 имеет INIT: 0011 1100 0011 1100 → 3C3C

Rom4 имеет INIT: 0110 0110 0110 0110 → 6666

Заготовка готова. Можно приступить к реализации схемы.

Создайте исходный модуль для схемотехнического режима проектирования. Для этого в главном меню управления выберите **Project** → **New Source** или кликните два раза в окне процессов по процедуре **Create New Source** (не забыв перед этим применить выделение к любому из исходных модулей в окне исходных модулей, иначе эта опция будет недоступной). Перед вами появится следующее окно (рис.1.3.2):

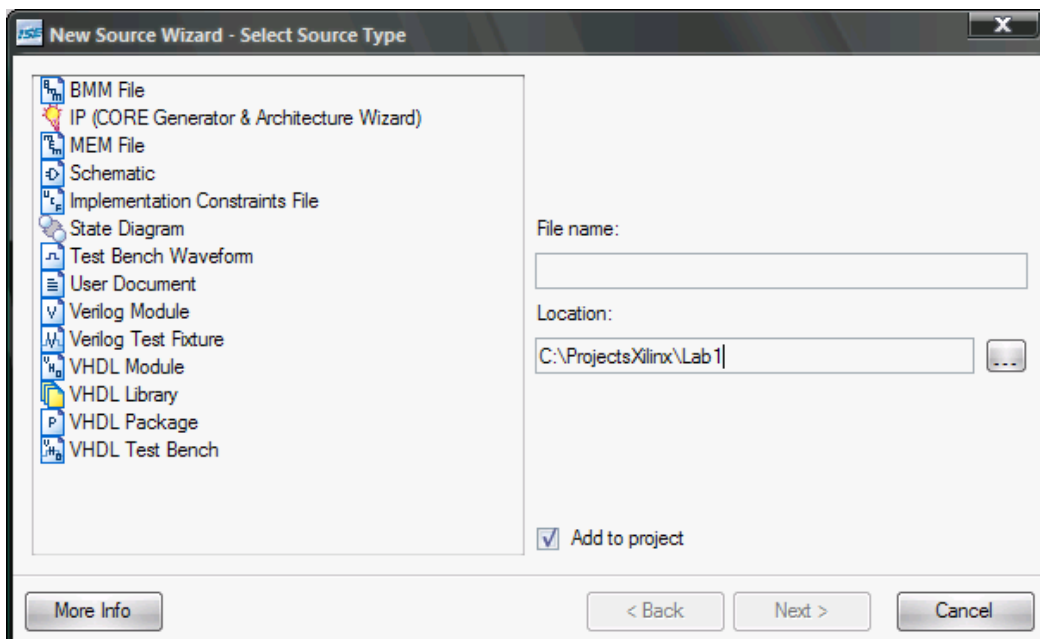


Рисунок 1.3.2.

Вам необходимо выбрать из списка тип исходного модуля (просто выделить его кликнув 1 раз левой кнопкой мышкой) – Schematic, задать название этого модуля (пусть называется shema), путь (на рисунке - Location) изменять не стоит, он автоматически находится в папке с проектом. Не забудьте установить флаг Add to project (добавить в проект), если не стоит. Вам станет доступной кнопка Next. Нажимите ее. Перед вами должно появиться окно с отсчетом о том, что исходный модуль был создан (рис.1.3.3):

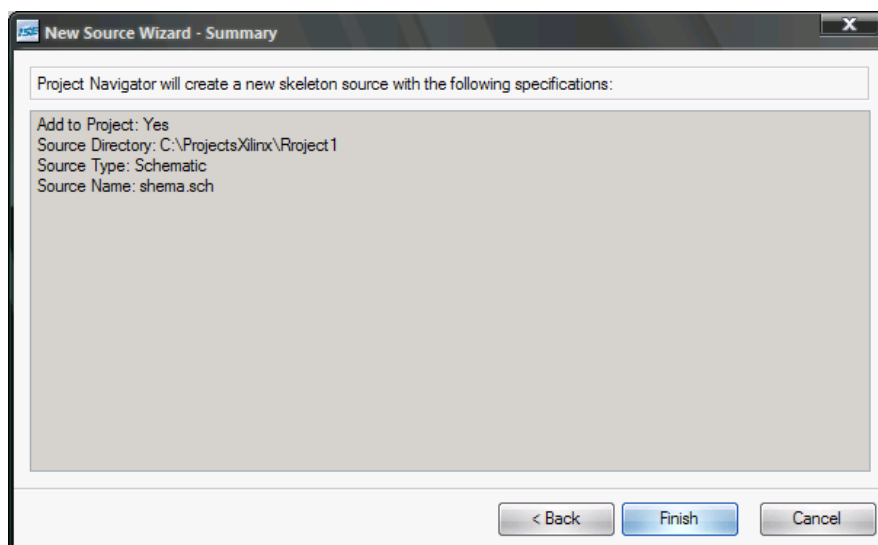


Рисунок 1.3.3.

Нажмите кнопку Finish (закончить).

Перед собой в панели размещения редакторов вы должны увидеть схемотехнический редактор (рис.1.3.4).

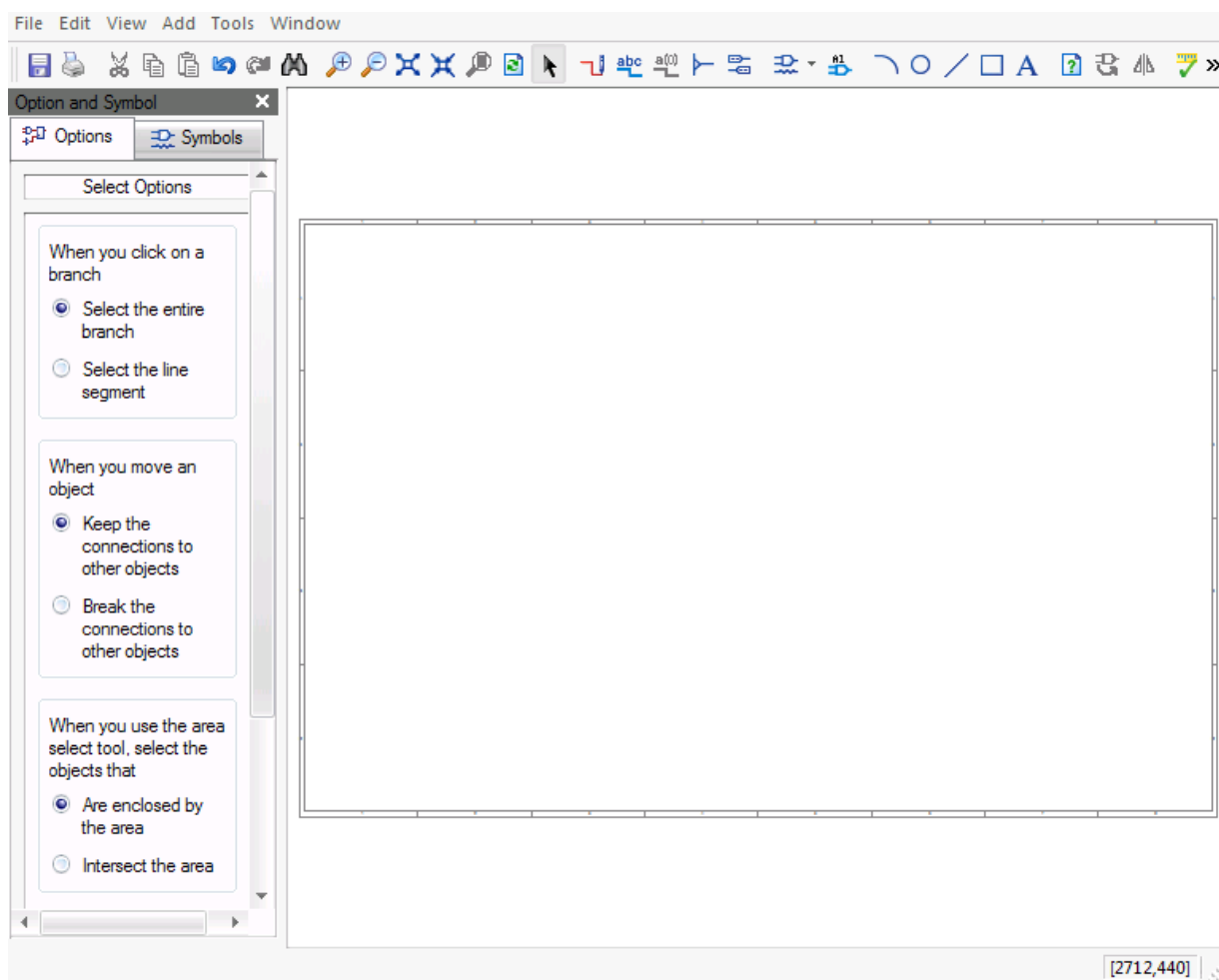


Рисунок 1.3.4.

В центре располагается поле, в рамках которого вы должны разместить схему. Размер поля можно редактировать: для этого кликните правой кнопкой мыши по полю и во всплывшем меню выберете Object Properties (или просто нажмите сочетание клавиш Alt+Enter). Окно с выбором размера представлено на рис.1.3.5.

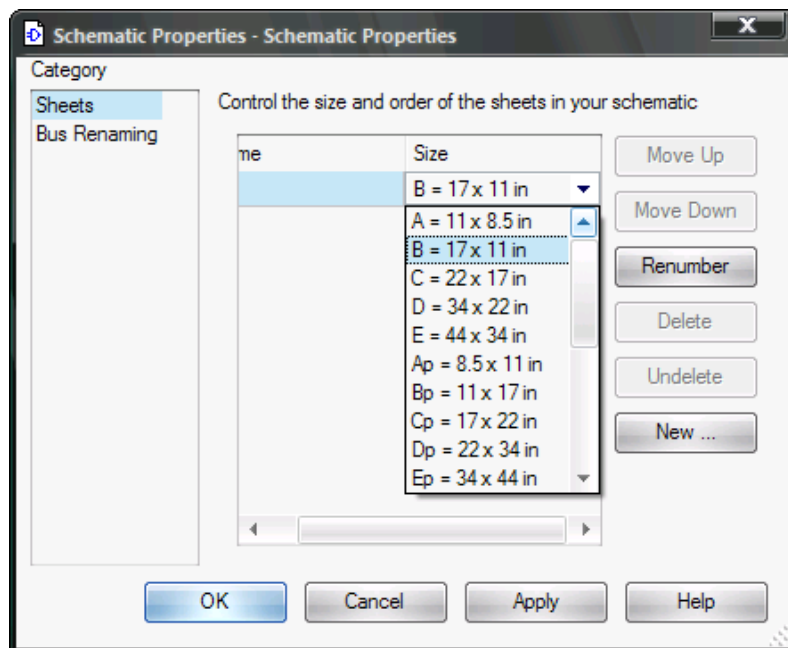


Рисунок 1.3.5.

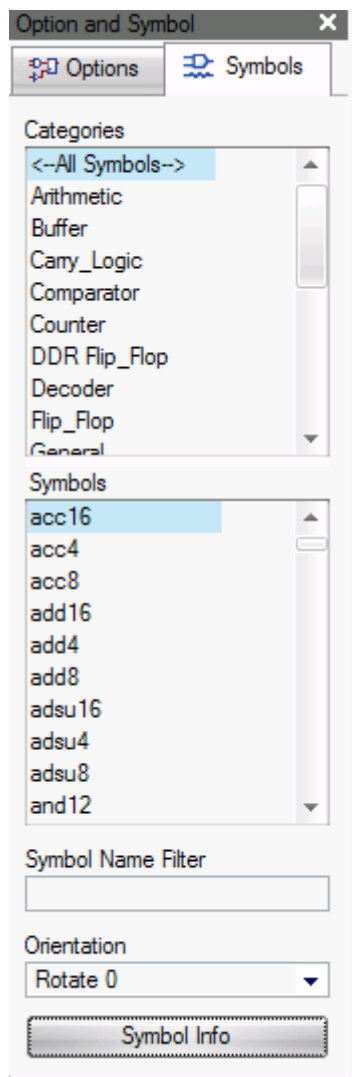


Рисунок 1.3.6.



В категории Sheets, кликнув по колонке Size по текущему размеру, выберете необходимый вам размер поля (для данного примера достаточно размера B). Для справки: категория Bus Renaming содержит в себе опции настроек названий объектов (контроль согласованности имен). Для удобства работы можете изменить исходные параметры.

Слева в схемотехническом редакторе располагается окно Опции и Элементы (Options and Symbols) – рис.1.3.6. Во вкладке Опции содержатся инструкции, связанные с выбранным инструментом в оперативном меню управления редактора. Для удобства работы можете изменить исходные параметры. Содержание данных параметров рассмотрено не было, т.к. оно не существенно. Выбрав вкладку Symbols, вы попадете в меню выбора элемента. Искать элементы можно по категориям (например, если серия кристаллов ограничивает вас в использовании элементов, можно узнать это выбрав категорию и посмотрев что вам доступно). Снизу категорий располагаются сами элементы. Осуществлять поиск необходимых элементов также удобно через текстовое поле «фильтр имен элементов» (Symbol Name Filter). При введении названия элемента автоматически формируется список доступных элементов в поле Symbols, названия которых начинаются на

введенные вами в поле фильтра буквы/символы. Также идет параллельно выбор категории, в которой может находиться вводимый вами элемент. Если при введении названия элемента поле Symbols стало пустым, то возможно элемента с данным названием не существует, или вы задали название неправильно, или данный элемент недоступен для выбранной вами ранее серии кристаллов.

Выбранному элементу можно задать ориентацию (повернуть, отразить) с помощью списка Orientation расположенного внизу окна Options and Symbols. Также можно посмотреть описание выделенного элемента, кликнув кнопку Symbol Info, расположенную все в том же окне Options and Symbols.

Выберите необходимый для решения рассматриваемой задачи элемент rom16x1 (он находится в категории Память(Memory)). К вашему курсору прилипнет образ этого элемента. Разместите его, наведя на необходимую область и кликнув левой кнопкой мыши. Разместите еще 3 таких же элемента.

Для подачи входной информации на вашу схему вам предлагается создать шину данных. Для этого выберите инструмент Add Wire в оперативной панели управления имеющий пиктограмму . Этот инструмент позволяет рисовать линии связи (проводки). Чтобы создать шину, нарисуйте вертикальную линию вдоль всех элементов Rom на значительном расстоянии. Эта будущая шина. Чтобы она стала шиной, выберите инструмент Add Net Name в оперативной панели управления, имеющей пиктограмму . Этот инструмент позволяет называть объекты собственными именами.



*Старайтесь следить за тем, чтобы разные объекты имели разные имена.*

При выборе этого инструмента в окне Options and Symbols во вкладке Options появятся опции, связанные с присваиванием имени объектам. В текстовом поле Name укажите имя объекта, после чего присвойте его ему, кликнув левой кнопкой мыши по объекту. Итак, чтобы шина стала шиной, назовите ее Int(3:0) (4-х разрядная шина с индексацией от 3 до 0), после чего, линия станет толстой. На экране вы должны увидеть примерно следующее (рис.1.3.7):



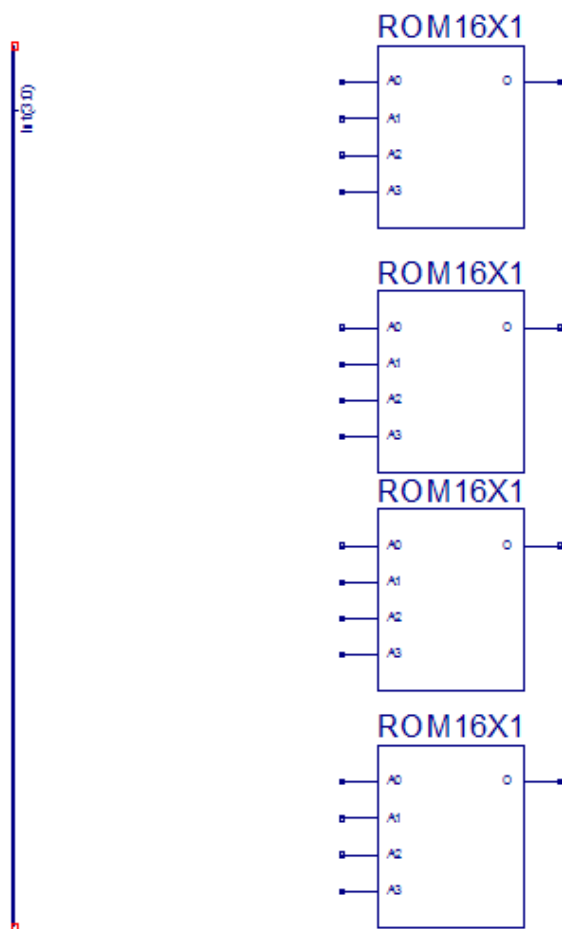




Рисунок 1.3.7.

Использование шины имеет ряд преимуществ: позволяет оптимизировать отображение разводки связей, при тестировании упрощается задание входной информации. Для наглядности выходные сигналы в данном примере не будут объединяться в шину. Выберите снова инструмент Add Wire и протяните короткие горизонтальные линии от выходов Rom элементов. Назовите их Out0, Out1, Out2 и Out3 сверху вниз. Теперь приступайте к разводке входной шины данных. Для этого требуется использовать специальный переходник – между шиной и проводом. Выберите инструмент Add Bus Tap в оперативной панели управления имеющий пиктограмму . При выборе этого инструмента в окне Options and Symbols во вкладке Options появятся опции, связанные с применением этого инструмента. Вам достаточно выбрать ориентацию так, чтобы данный переходник упирался в шину. Сделайте отводы для всех входов элементов Rom (всего их получится 16 штук). Соедините переходники со входами Rom-элементов линиями связи с помощью инструмента Add Wire. Чтобы САПР знал, какой сигнал из всех, содержащихся в шине поступает на тот или иной вход, назовите линии связи Int(0), Int(1), Int(2) и Int(3) сверху вниз для каждого элемента. Чтобы схема полностью функционировала, осталось установить маркеры на те линии, которые

будут участвовать в процессе функциональной верификации (тестирования). Для этого выберите инструмент Add I/O Marker в оперативной панели управления имеющий пиктограмму . Установите маркеры на концы линий (там, где линии обрываются). В итоге у вас должна получиться примерно такая схема (рис.1.3.8):

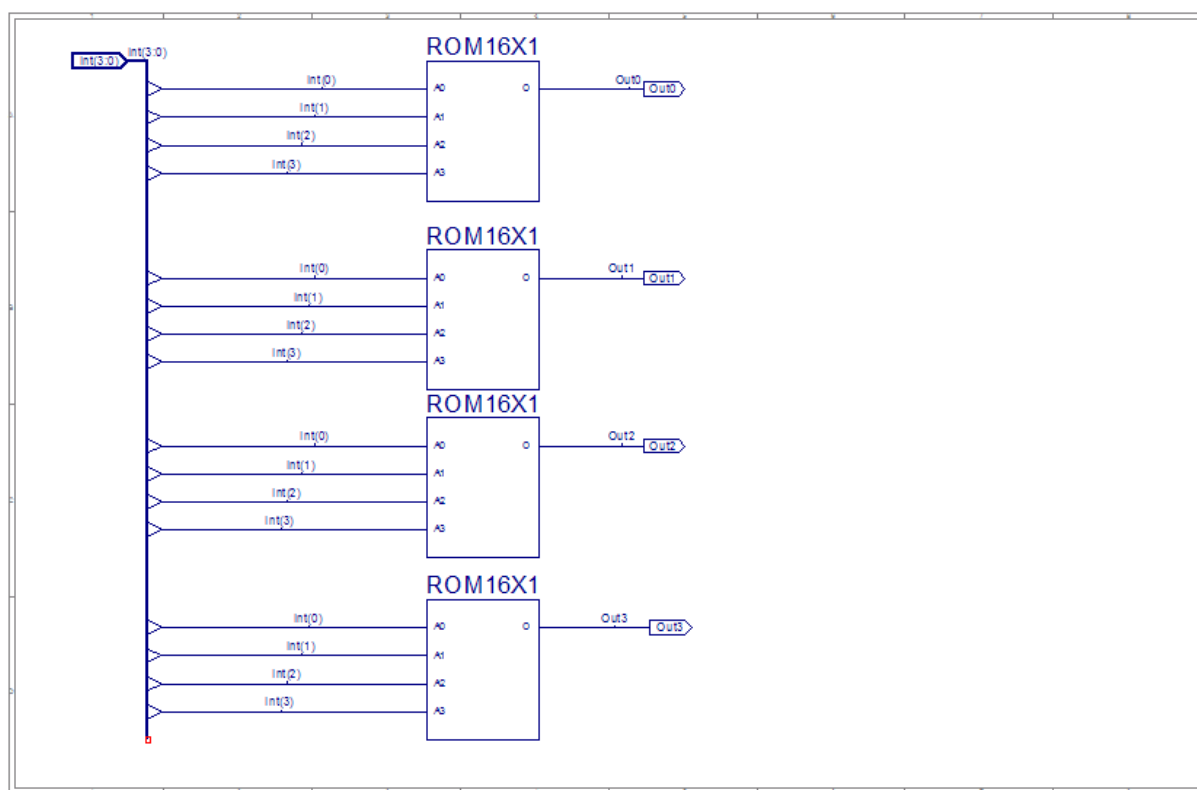


Рисунок 1.3.8.

На данной схеме индекс 0 – обозначает старший разряд, индекс 3 – младший. Осталось лишь задать INIT'ы для элементов Rom, чтобы схема работала так, как ей полагается. Верхний Rom – отвечает за формирование старшего разряда (условно дается обозначение Rom1), нижний – за формирование младшего (Rom4).

Задайте INIT'ы, заготовленные ранее:

Rom1 имеет INIT: FF00

Rom2 имеет INIT: 0FF0

Rom3 имеет INIT: 3C3C

Rom4 имеет INIT: 6666

Чтобы задать INIT дважды кликните по элементу Rom. Перед вами откроется окно (рис.1.3.9):

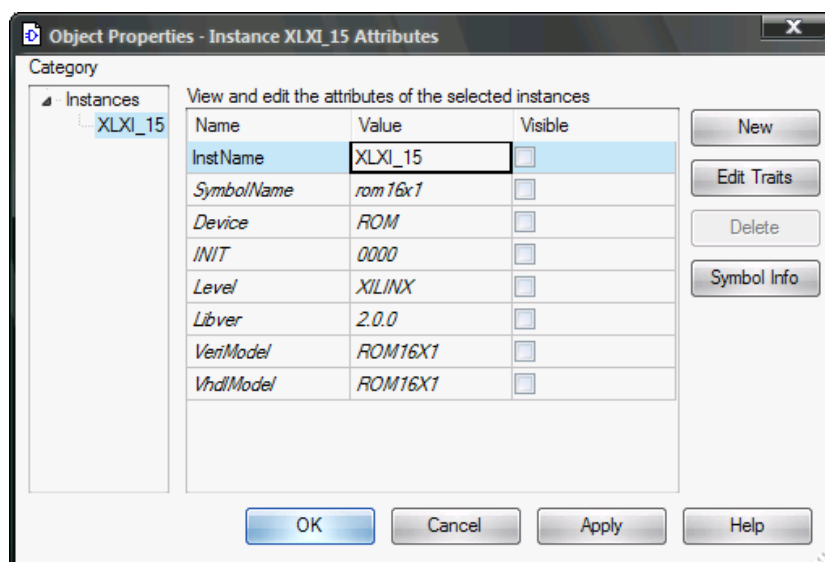



Рисунок 1.3.9.

Каждый элемент имеет свое уникальное имя. Пока для вас это несущественно. Поэтому имя элемента не трогайте. Измените INIT, введя его с клавиатуры (регистр значения не имеет). Также можно выбрать опцию, чтобы тот или иной параметр элемента был виден. Настоятельно рекомендуется INIT делать видимым! Итак, схема реализована. На данном этапе возможно проверить корректность созданной схемы (это ускорит процесс отладки). Для этого щелкните по функции Check Schematic в оперативной панели управления, имеющую пиктограмму . Отчет о проверке высветится в окне консольных сообщений. При необходимости устраните ошибки, затем сохраните изменения, выбрав опцию в главном меню управления **File** → **Save** (или просто нажав сочетание клавиш Ctrl+S).

## 1.4 Функциональная верификация

Для того чтобы осуществить верификацию, необходимо создать соответствующий исходный модуль, где будет храниться информация об условиях верификации (режимы, учет задержек и прочее) и тестовая последовательность (которую вам необходимо задать). Создайте исходный модуль. Для этого в главном меню управления выберите **Project** → **New Source** или кликните два раза в окне процессов по процедуре **Create New Source** (не забыв перед этим применить выделение к любому из исходных модулей в окне исходных модулей, иначе эта опция будет недоступной). Перед вами появится следующее окно (рис.1.4.1):

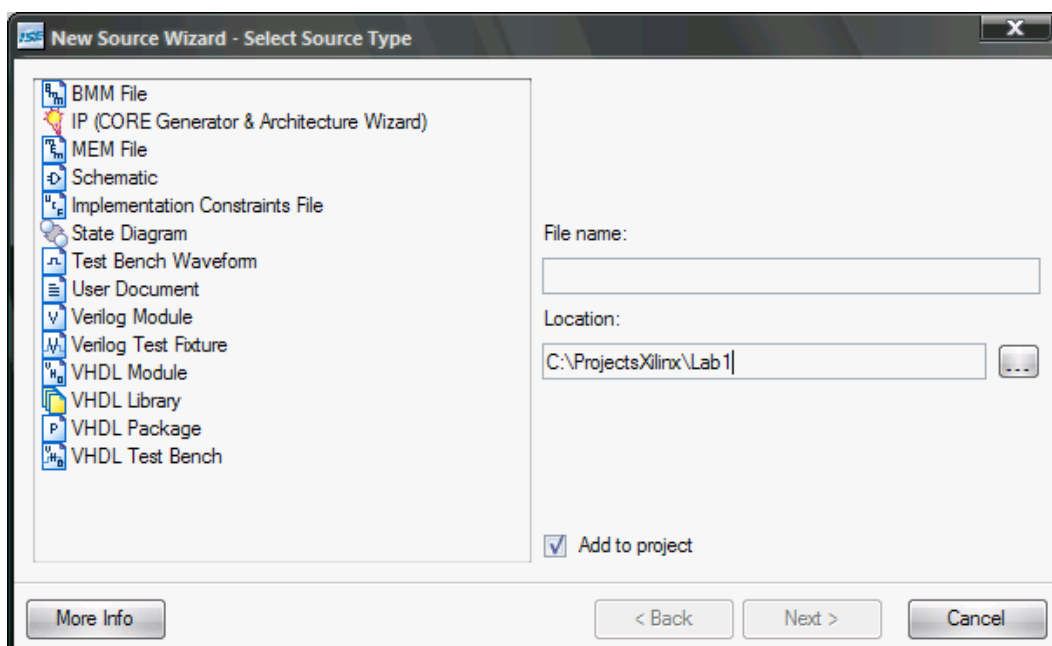


Рисунок 1.4.1.

Вам необходимо выбрать из списка тип исходного модуля (выделить его, кликнув 1 раз левой кнопкой мышки) – Test Bench Waveform, задать название этого модуля (пусть называется test), путь (на рисунке - Location) изменять не стоит, он автоматически находится в папке с проектом. Не забудьте установить флаг Add to project (добавить в проект), если он не стоит. Вам станет доступной кнопка Next. Нажмите ее.

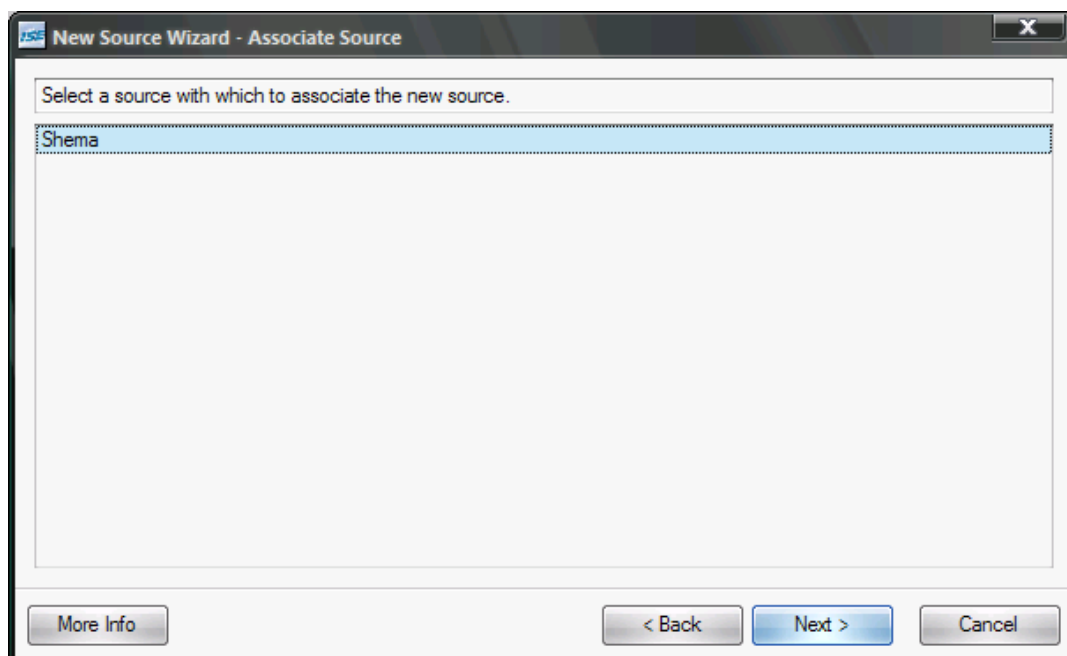


Рисунок 1.4.2.

Перед вами появится окно (рис.1.4.2), в котором вам предлагается выбрать исходный модуль, который вы собираетесь верифицировать. Выберите ваш модуль Shema, нажмите Next.

Перед вами должно появиться окно с отсчетом о том, что исходный модуль был создан (рис.1.4.3):

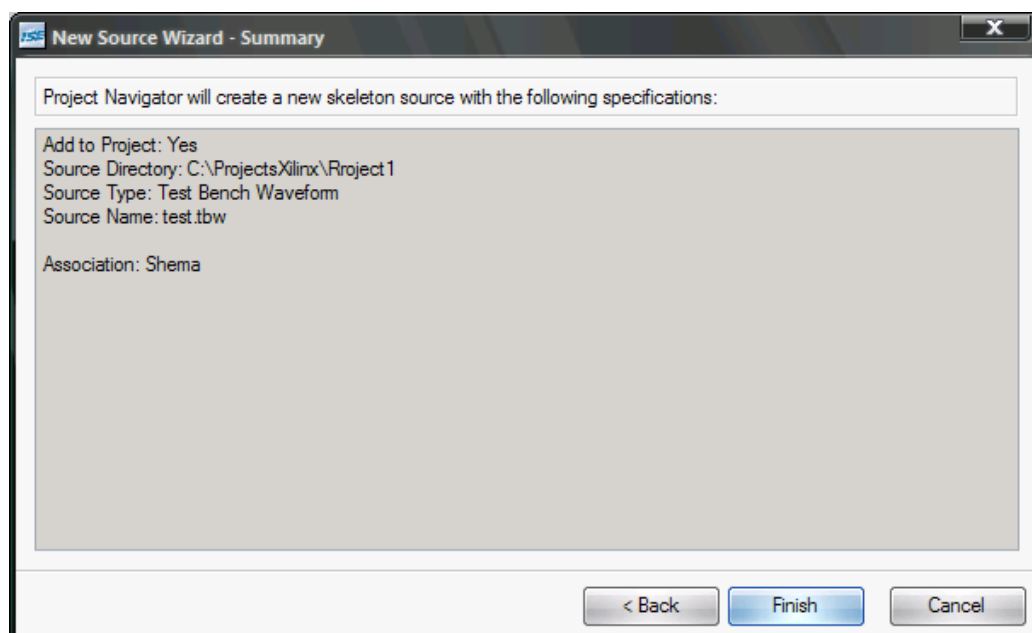


Рисунок 1.4.3.

Нажмите кнопку Finish (закончить).

Далее перед вами появится окно (рис.1.4.4):

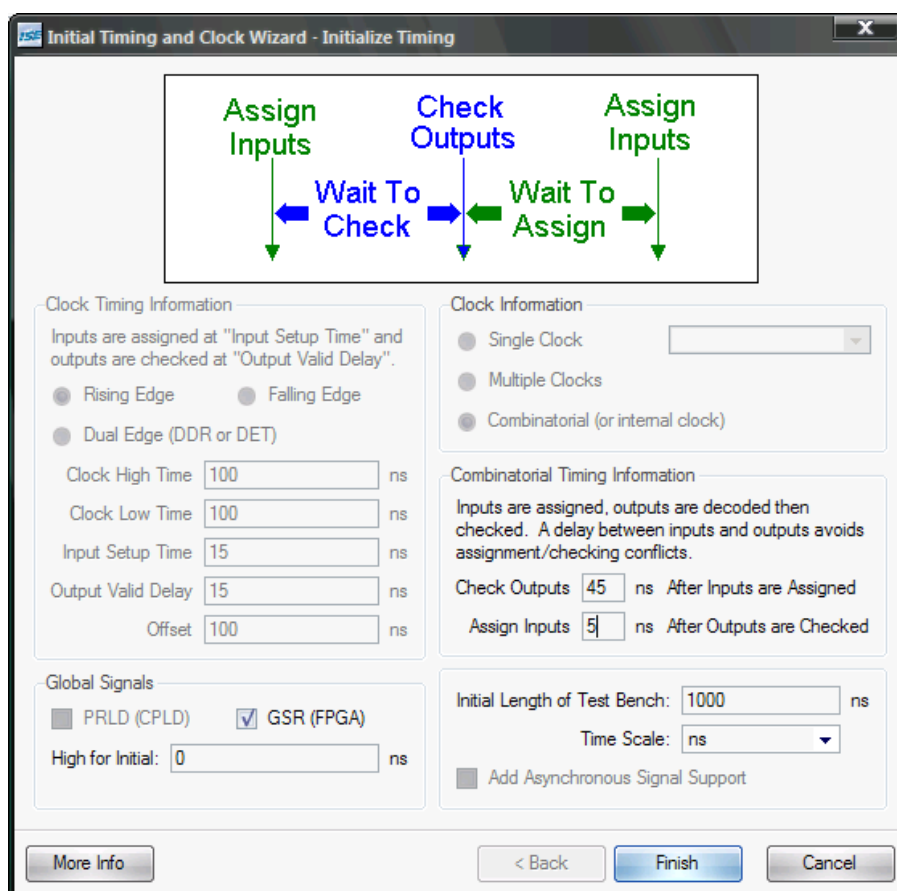


Рисунок 1.4.4.

В этом окне вам предлагается выбрать параметры верификации. В параметрах Clock Timing Information опция Rising Edge означает использование «положительной логики» (восходящий фронт- рабочий), опция Falling Edge означает использование «отрицательной логики» (рабочий - нисходящий фронт), опция Dual Edge (DDR or DET) означает использование «двойной или смешанной логики» (рабочие оба фронта, т.е. восходящий + нисходящий фронты). Параметр Clock High Time указывается время высокого уровня тактового сигнала, Clock Low Time указывается время низкого уровня тактового сигнала, в сумме эти два параметра дают период синхросигнала. Параметр Input Setup Time указывается время предустановки входных значений относительно рабочего фронта, параметр Output Valid Delay (Допустимое время задержки выхода) указывается время задержки выхода относительно рабочего фронта, параметр Offset предотвращает возможность изменения входных сигналов в течении указанного времени (блокировка сигнала в самом начале исполнения). Параметр High for Initial задает задержку для инициализации. В параметрах Clock Information задаются синхросигналы. Опция Signal Clock означает наличие в схеме одного синхросигнала, Multiple Clock – 2 и более, Combinational (or interal clock) означает режим без синхросигнала (эта опция не исключает задание синхросигнала вручную). Опция Check Outputs задает задержку срабатывания между входом и выходом элемента, Assign Inputs – интервал, по истечении которого будет присвоено выходное значение после установки входного, Initial Length of Test Bench – время тестовой последовательности, Time Scale – единицы измерения для задания времени тестовой последовательности. Флаг Add Asynchronous Signal Support включает поддержку асинхронных сигналов (возможность вводить сигналы асинхронно).

Если какие-то опции вам не доступны – значит этого не позволяет задание других параметров, выбор кристалла или конструкция схемы (например, схема асинхронная, т.е. не содержит в себе синхросигналов). Если в недоступных опциях установлены параметры, значит их определил САПР. Подробно получить справку обо всех опциях этого окна можно кликнув на кнопку More Information, которая находится в том же окне.

Начальные установки рекомендуется для поставленной задачи установить такими же, как на рисунке.

Нажмите Finish. Перед вами может предстать следующая картина (рис.1.4.5):

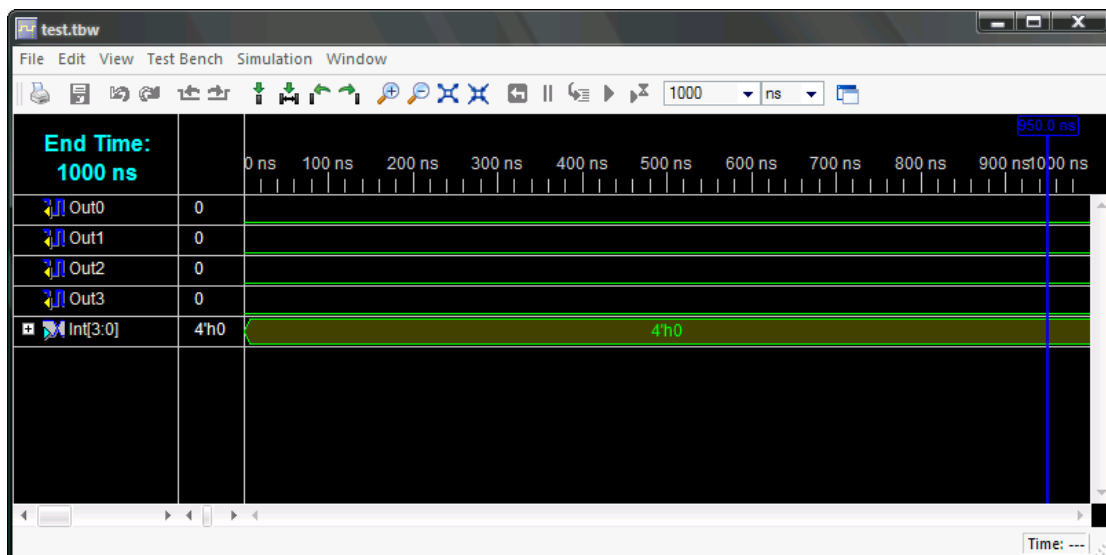


Рисунок 1.4.5.

Цветовое оформление не очень радует глаз. Изменить его можно выбрав опцию в главном меню управления навигатора проектов **Edit** → **Preferences...** Откроется окно предпочтений (рис.1.4.6):

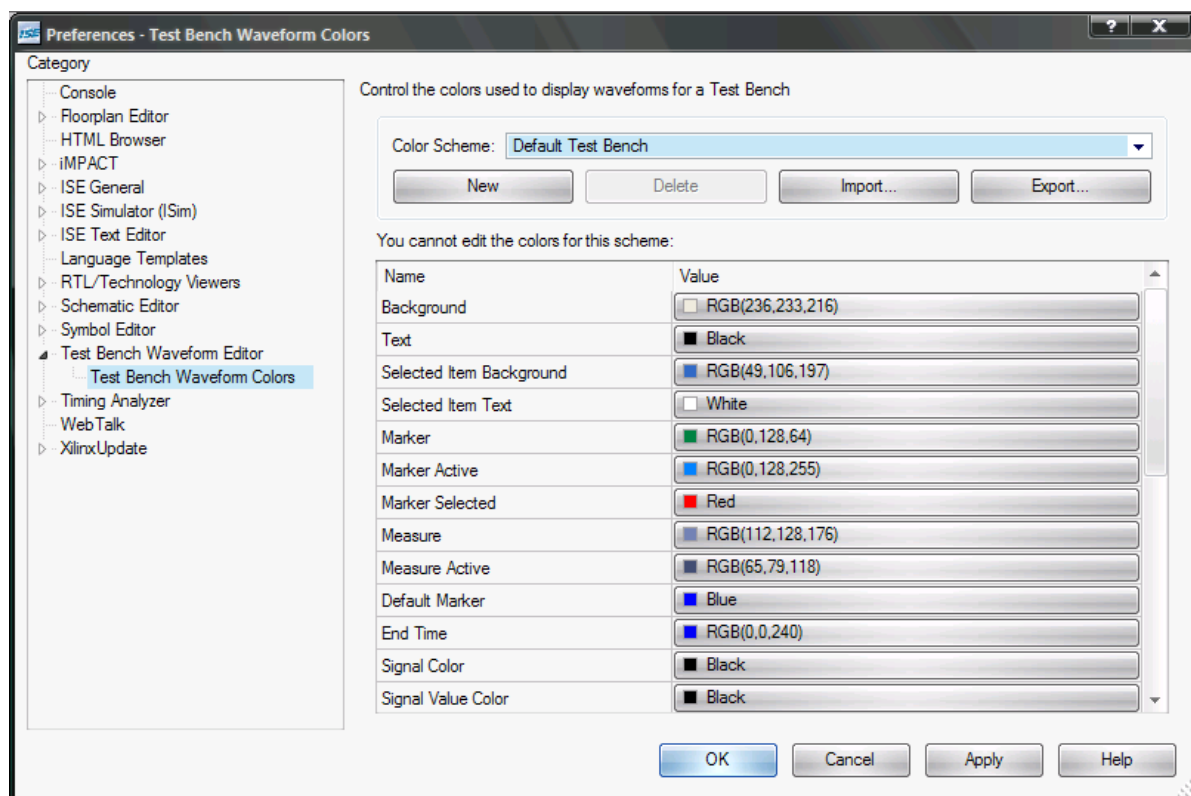


Рисунок 1.4.6.

Слева в категориях выберите **Test Bench Waveform Editor** → **Test Bench Waveform Colors**. Сверху вы найдете выпадающий список, озаглавленный как **Color Scheme**. Вы можете изменить настройки текущей цветовой схемы, установить одну из понравившихся схем или создать свою схему. Таким же образом вы можете изменить цветовую схему в других составляющих САПРа.

После применения настроек вам потребуется перезапустить модуль, чтобы цветовое оформление вступило в силу.

Итак, можно приступать к тестированию. Сначала необходимо задать синхросигнал (последовательность чередующихся логических нулей и единиц). Далее необходимо задать саму тестовую последовательность. Не существует четких алгоритмов, как правильно производить тестирование. Есть лишь указание: тестовая последовательность должна проверять все возможные комбинации и конфигурации. Схема, реализованная в данном пособии простейшая, поэтому несложно догадаться, что охватить все возможные комбинации можно подав на вход последовательно значения от 0 до 15.

Слева от диаграмм находится названия вх/вых значений. В окне временных диаграмм возможно редактирование лишь входных значений (что само по себе является логичным). Около шины Int(3:0) находится значок «+». Нажмите на него. Итак, вы увидите значения на шине и отдельно на каждом разряде. Установить значения на шине можно 2 способами: кликнув левой кнопкой мыши по диаграмме шины в нужном месте откроется окно, куда вы сможете ввести значение в hex-коде (рис.1.4.7).

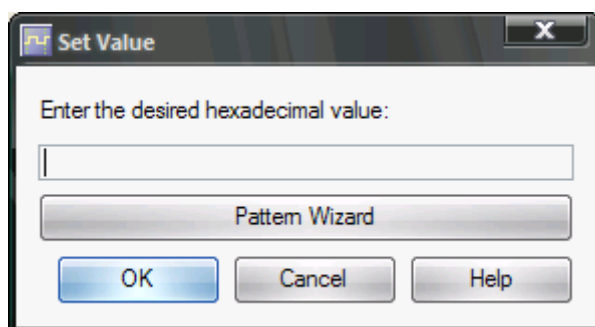


Рисунок 1.4.7.

Так как в схеме шина 4 разрядная, в ней уместается лишь один тетраэдр, поэтому в появившемся окне пишется не более 1 символа (от 0 до F). Второй способ заключается в установке уровня каждого разряда шины в отдельности по средствам клика мышки по временной диаграмме разряда. После заполнения входных данных (тестовых последовательностей), ваша диаграмма будет иметь вид (рис.1.4.8):



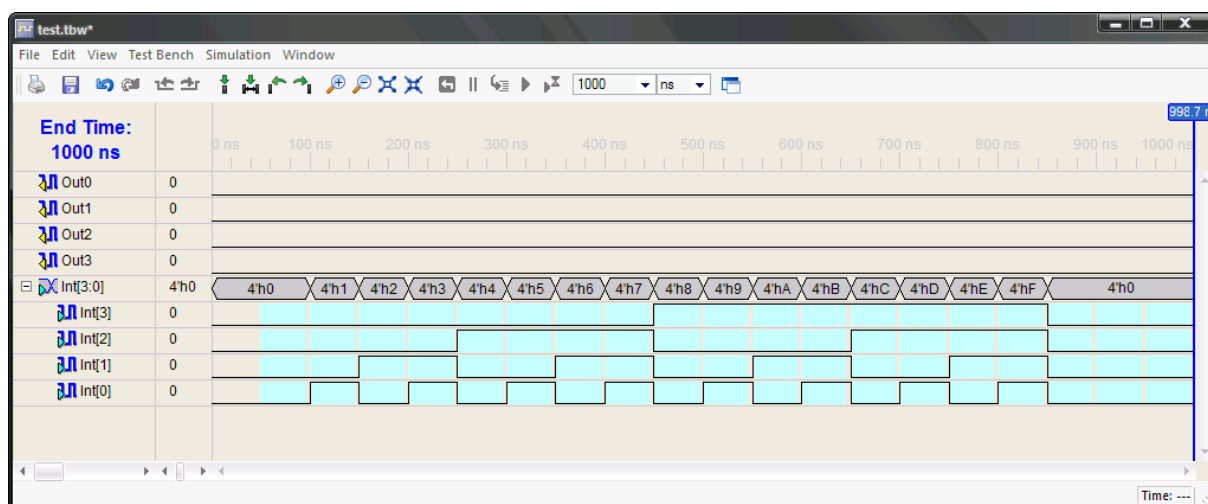


Рисунок 1.4.8.

Для ясности: низкому уровню соответствует логический 0, высокому уровню соответствует логическая единица; надпись 4'h0 означает: **4** – что шина 4-х разрядная, **'h** – что запись числа осуществляется в hex-коде (существует еще binary – двоичный, ставится буква **b** после апострофа, и десятичный код, в таком случае после апострофа ничего не ставится), **0** – что значение на шине равно нулю.

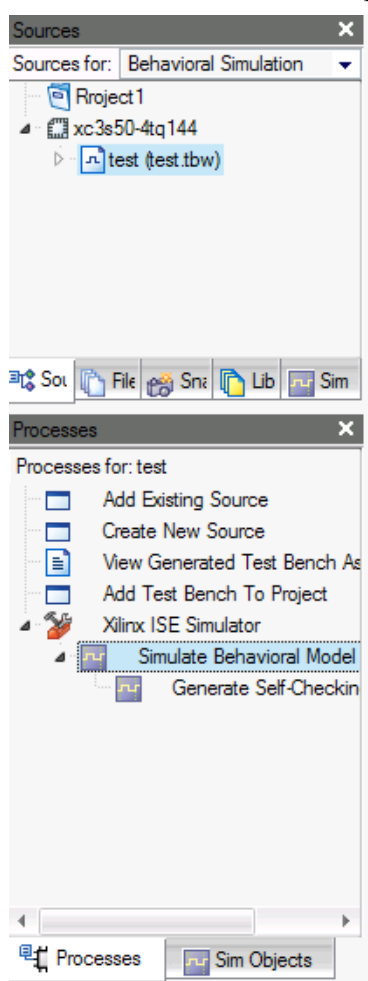


Рисунок 1.4.9.

По окончании ввода тестовой последовательности сохраните файл. Теперь можно запускать верификацию. Для этого в окне исходных модулей (рис.1.4.9) выберите тип отображаемых модулей – «исходные модули для моделирования поведения» с помощью соответствующего выпадающего списка (**Sources for: → Behavioral Simulation**). Найдите исходный модуль тестирования (test.tbw). В окне процессов вы увидите процедуру Xilinx ISE Simulator. Это своего рода «ящик с инструментами», применяемая к выбранному модулю. Раскройте эту процедуру. В ней находится «инструмент» (процедура) Simulate Behavioral Model. Его составной частью является Generate Self-Checking Test Bench. Вам потребуется «весь инструмент» Simulate Behavioral Model. Запустите его двойным кликом мышки. В окне консольных сообщений вы сможете наблюдать за ходом выполнения запущенного вами процесса. Если вы все сделали правильно, то в панели размещения редакторов вы увидите временные диаграммы (рис.1.4.11), смоделированные под ваши тестовые последовательности (в противном случае, вы

увидите в окне консольных сообщений причину невозможности создания поведенческой модели; в таком случае вам придется заняться исправлением допущенных ошибок):

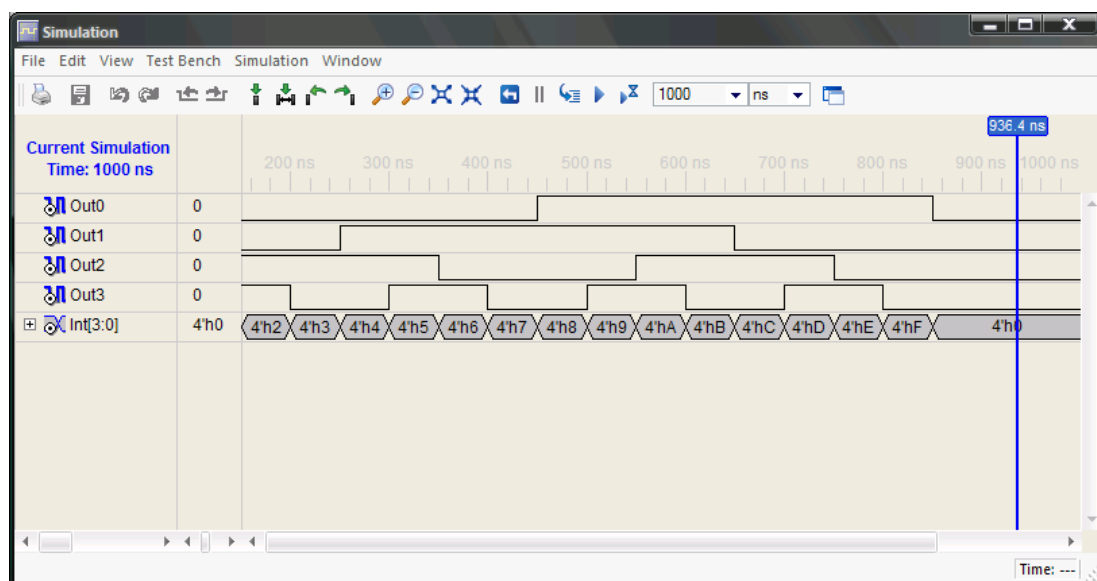



Рисунок 1.4.10.

У вас может возникнуть проблема: отображаемая модель поведения «обрублена» в начале или в конце заданного интервала времени (рис.1.4.10). Чтобы устранить это, необходимо воспользоваться инструментом в оперативной панели инструментов **Zoom To Full View**, имеющим пиктограмму .

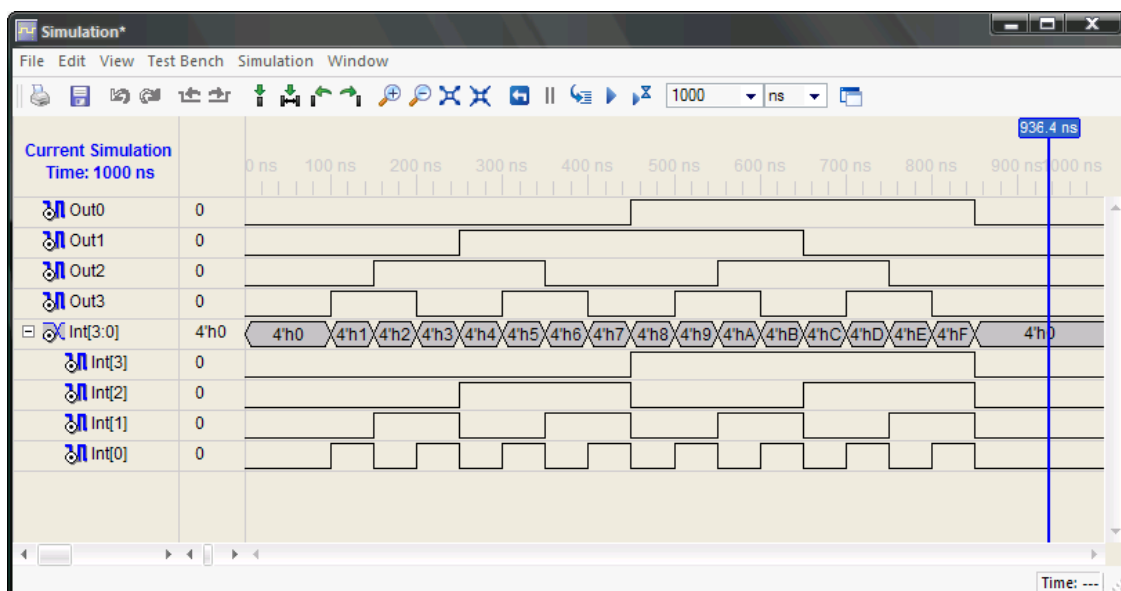


Рисунок 1.4.11.

Можно сделать вывод, что использование шин данных является предпочтительным по отношению к ее не использованию.

## Лабораторные работы

Лабораторные работы должны быть выполнены в САПР Xilinx ISE WebPack v 10.1i.

Лабораторные работы рассмотрены на примере 13 варианта.

### Лабораторная работа № 1

Вариант задания определяется таблицей «варианты заданий на лабораторные работы по дисциплине проектирование ЦУ на ЯОА» (табл. 1):

Вар.\вх.	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	Студенты (бригада)
1	A	B	A	B	C	0	E	7	4	1	A	F	8	A	E	5	
2	9	C	0	D	F	7	6	2	8	E	1	1	A	C	3	A	
3	1	6	C	5	4	6	3	9	B	8	E	C	9	F	0	E	
4	F	7	3	5	8	A	C	D	D	8	B	7	2	4	0	B	
5	6	C	8	1	4	E	2	6	7	A	F	1	0	B	F	4	
6	3	2	A	9	4	F	8	F	E	1	B	8	0	5	1	A	
7	7	B	8	C	F	A	E	0	7	9	C	5	3	7	A	0	
8	E	4	1	0	2	A	C	F	5	F	8	7	8	A	3	4	
9	5	5	B	A	0	4	2	8	F	3	C	E	0	B	D	C	
10	8	1	7	E	B	F	4	7	2	9	1	A	0	1	8	4	
11	0	C	5	7	6	B	9	E	1	C	2	F	C	D	2	4	
12	4	1	8	2	0	C	9	5	A	7	F	E	4	3	7	8	
13	2	5	8	A	8	7	1	8	B	5	9	E	F	4	C	0	
14	7	6	5	4	C	D	E	F	A	9	8	7	3	2	1	0	
15	4	5	6	7	F	E	D	C	7	8	9	A	0	1	2	3	
16	0	4	8	C	1	5	9	D	2	6	A	E	3	7	B	F	
17	2	3	4	5	2	3	4	5	9	A	B	C	F	B	7	3	
18	3	4	B	C	2	5	A	D	1	6	9	E	0	7	8	F	
19	7	6	5	4	3	2	1	0	F	E	D	D	C	8	8	8	
20	1	3	3	5	5	7	7	7	9	8	B	A	D	C	F	E	
21	7	2	2	3	F	8	1	3	B	D	0	C	1	9	A	4	
22	A	C	7	1	F	4	7	2	3	E	0	5	9	6	B	8	
23	4	9	A	9	4	0	4	0	E	7	F	1	6	3	D	A	
24	2	C	C	6	7	E	4	A	C	E	B	F	0	5	D	7	
25	3	9	A	C	0	4	7	5	B	9	E	2	5	1	E	A	
26	A	B	B	E	B	A	0	C	9	D	F	1	7	3	4	8	
27	8	C	2	4	3	9	6	6	7	3	4	E	F	D	2	D	
28	0	5	0	7	6	1	0	4	3	5	F	C	4	A	B	7	
29	F	2	E	4	9	C	D	E	A	D	D	E	A	D	8	A	
30	C	D	D	A	8	7	E	7	8	B	8	C	0	5	F	3	
31	4	8	9	1	3	A	C	F	0	3	5	C	1	9	8	4	
32	1	9	8	8	D	E	A	D	B	A	B	A	8	8	9	1	
33	A	C	5	6	F	B	2	5	0	4	C	A	1	9	5	2	
34	2	0	0	7	2	0	0	8	B	A	B	A	5	3	1	9	
35	F	7	9	E	2	4	7	6	6	C	0	E	5	5	C	D	
36	2	0	0	7	1	9	8	8	D	E	A	D	2	0	0	8	
37	5	A	E	8	F	A	1	4	7	E	0	C	B	A	B	A	
38	A	3	C	A	1	E	1	8	2	6	7	F	D	0	C	9	

39	E	0	F	9	C	E	8	B	9	3	6	4	C	5	6	1	
40	B	0	4	2	7	B	8	D	D	A	C	8	5	3	7	F	
41	4	F	B	0	1	F	A	7	6	2	E	4	8	1	C	6	
42	A	1	5	0	8	B	1	E	8	F	4	F	A	9	3	2	
43	F	E	C	D	B	A	8	9	7	5	5	7	3	5	1	3	
44	2	F	4	E	C	9	E	D	D	A	D	E	D	A	8	A	
45	4	1	2	8	C	0	5	9	A	7	E	F	3	4	8	7	
46	7	1	A	C	7	2	F	4	E	3	5	0	9	6	8	B	

Таблица 1. Варианты для лабораторных работ.

В лабораторной работе №1 требуется построить комбинационный логический узел, имеющий 4 входа  $X_I(3:0)$  и 4 выхода  $Y_O(3:0)$  для каждого из видов абстракции. Каждой входной комбинации ставится в соответствие некоторая выходная комбинация. Задание выдаётся в виде таблицы, в верхней строке которой перечисляются в порядке убывания входные комбинации от «F» (1111) до «0» (0000), а в нижней строке приведены соответствующие им выходные комбинации.

Для каждой из построенных схем необходимо предоставить временную диаграмму работы, на которой последовательным перебором от «0» до «F» или наоборот, будут введены все возможные входные комбинации.

Диаграммы и схемы вставляются в отчёт по лабораторной работе в том виде, как они представлены в САПР.

В данной работе рассмотрено два вида абстракции: описание, приближенное к аппаратуре (вентильный уровень) и алгоритмическое описание (поведенческий уровень). Таким образом, в работе используется:

- 1) 2 ЯОА
- 2) 2 уровня абстракции
- 3) создание иерархии

Вам предлагается создать следующую иерархическую структуру (рис. 1):

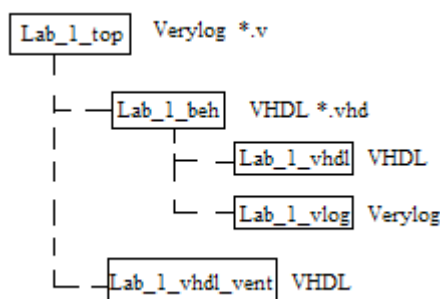


Рисунок 1. Иерархия проекта.

Вентильную реализацию должны содержать в себе 2 модуля: Lab\_1\_top и Lab\_1\_vhdl\_vent. Файлы Lab\_1\_vhdl и Lab\_1\_vlog – должны содержать в себе поведенческое описание. Модуль Lab\_1\_beh – модуль, объединяющий два модуля: Lab\_1\_vhdl и Lab\_1\_vlog.

Таким образом, интерфейс разрабатываемого устройства должен иметь следующий вид (рис. 2):

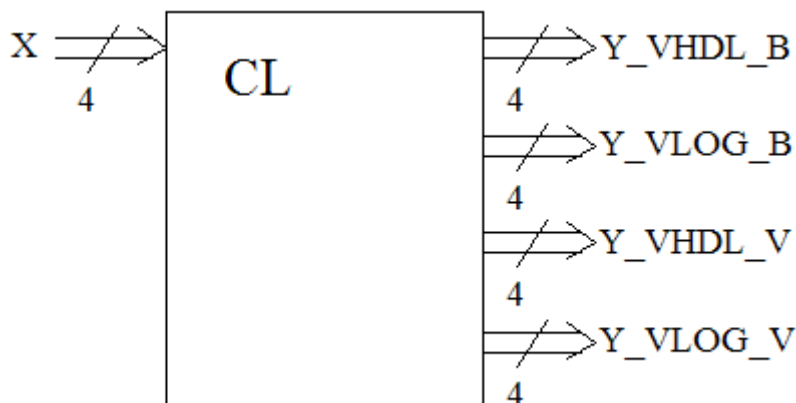


Рисунок 2. Интерфейс устройства.

где Y\_VHDL\_B – выходы модуля Lab\_1\_vhdl (поведенческий уровень), Y\_VLOG\_B – выходы модуля Lab\_1\_vlog (поведенческий уровень), Y\_VHDL\_V – выходы модуля Lab\_1\_vlog (вентильный уровень), Y\_VLOG\_V – выходы модуля Lab\_1\_top (вентильный уровень). Все выходы будут сформированы разными методами, но временные диаграммы у них должны совпадать.

Для начала, создайте проект (создание проекта подробно рассмотрено в разделе «Создание проекта»). В качестве имени вам предлагается выбрать Lab\_1\_HDL, в качестве параметра Top-level source type (тип топ-уровня исходных модулей) **обязательно** выберите HDL (так как топ-файл описан ЯОА).

В окне Project Properties (выбор ПЛИС, на котором будет исполнен проект) (рис. 3) выберите следующие установки (для практических работ Вам понадобится использовать ПЛИС определенной модели семейства Spartan 3E):

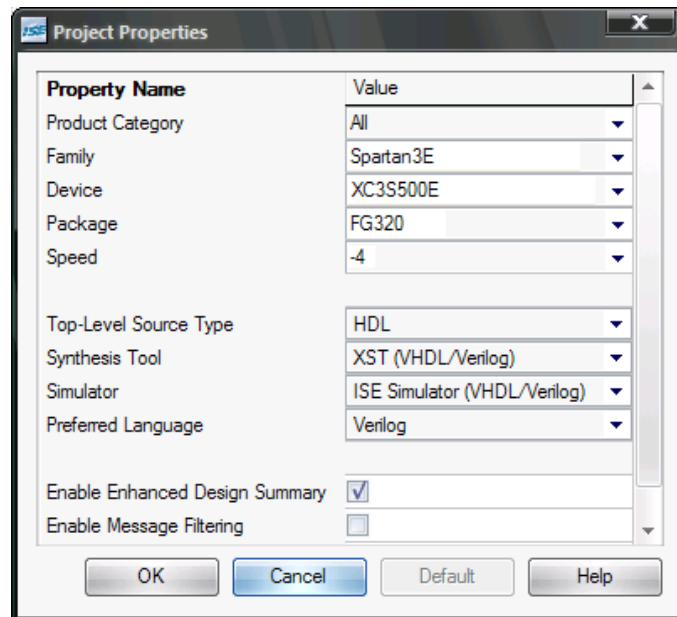


Рисунок 3. Создание проекта.

Затем вам предлагается создать поведенческий модуль VHDL. Для этого создайте исходный модуль типа VHDL Modul. Вам предлагается назвать его Lab\_1\_vhdl. Появится следующее окно (рис. 4):

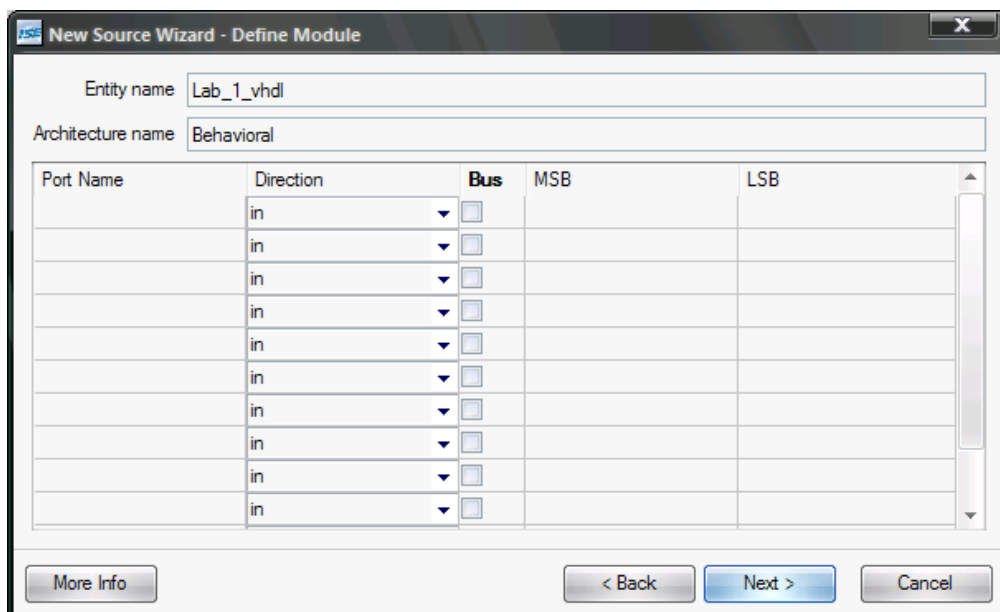


Рисунок 4. Создание проекта.

Здесь вам предлагается заготовить заранее входы и выходы схемы. Вам рекомендуется создать их самостоятельно, поэтому пропустите этот момент и, ничего не изменяя в этих опциях, нажмите Далее. Завершите создание исходного модуля.

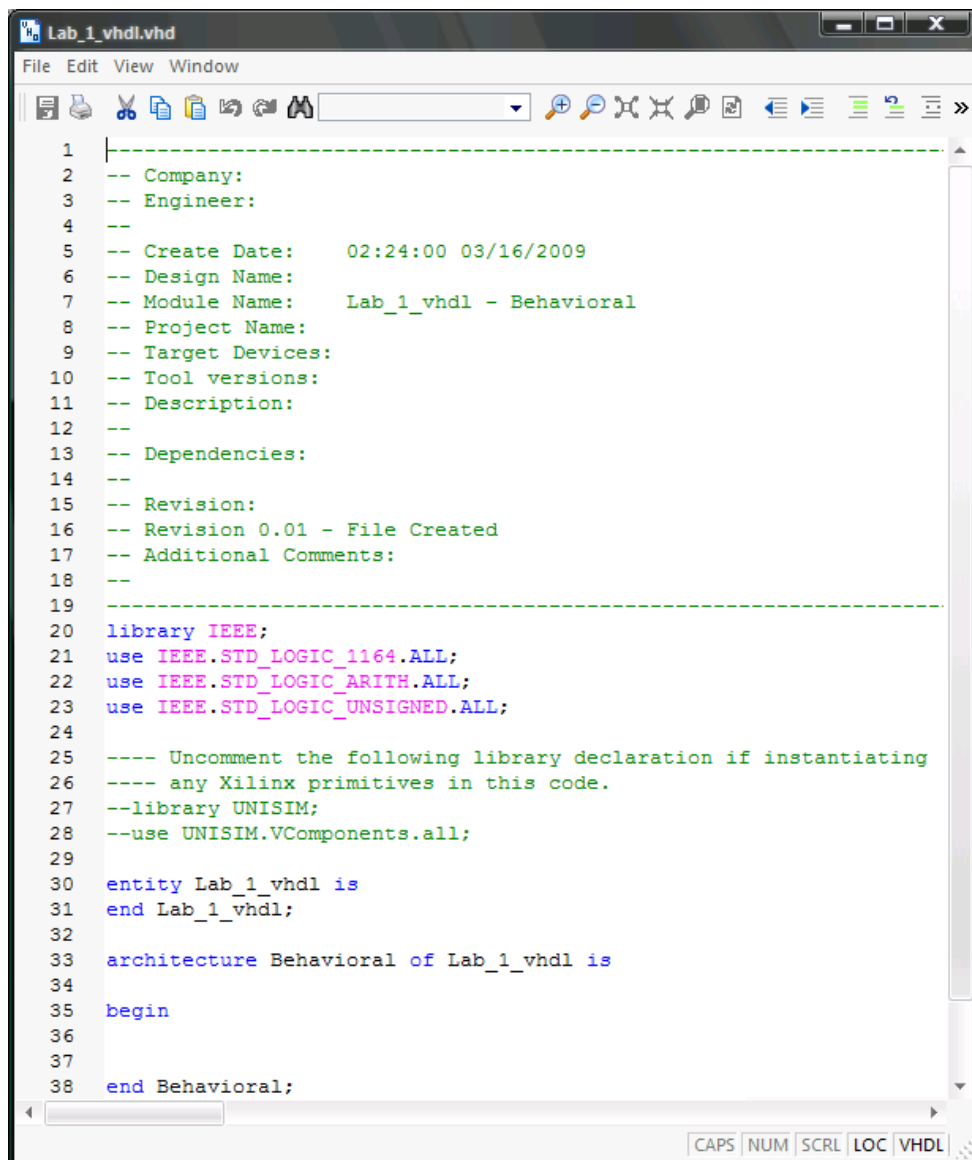


Рисунок 5. Внешний вид VHDL модуля.

В появившемся редакторе (рис. 5) стоит сразу обратить внимание на заготовку:

```
entity Lab_1_vhdl is
end Lab_1_vhdl;
```

```
architecture Behavioral of Lab_1_vhdl is
begin
```

```
end Behavioral;
```

В блоке **entity** описывается интерфейс устройства, в блоке **architecture** описывается его содержание.

Итак, вам будет изложен код с подробным описанием, что делает каждая строка.



*Настоятельно рекомендуется пропечатывать все части описания аппаратуры самостоятельно, без помощи копирования и вставки с целью лучшего запоминания синтаксиса Языков Описания Аппаратуры.*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--подключение библиотек
entity Lab_1_vhdl is
--Заголовок блока описания интерфейса
    Port ( X : in  STD_LOGIC_VECTOR (3 downto 0);
          Y : out STD_LOGIC_VECTOR (3 downto 0));
--описание портов в интерфейсе
--создание 4 разрядных входных и выходной шин
--STD_LOGIC_VECTOR - обозначение шины с данными типа STD_LOGIC
--in и out - указывает на направление работы шины (провода) -
--входная или выходная
end Lab_1_vhdl;
--конец описания интерфейса
architecture Behavioral of Lab_1_vhdl is
--заголовок описания архитектуры
--архитектура имеет название(Behavioral),
--т.к. устройство может содержать в себе несколько архитектур
begin
--запуск, начало описания непосредственно архитектуры
    CL : process(X) begin
--создается новый процесс обработки входных значений X
--CL - имя процесса
--<имя_процесса>:process(<список_чувствительности>)
        case X is
--применение процедуры case к входным значениям X
            when "0000" => Y <= "0000"; -- 0
--когда на вход поступает 0000, на Y выставляется 0000
            when "0001" => Y <= "1100"; -- C
            when "0010" => Y <= "0100"; -- 4
            when "0011" => Y <= "1111"; -- F
            when "0100" => Y <= "1110"; -- E
            when "0101" => Y <= "1001"; -- 9
            when "0110" => Y <= "0101"; -- 5
            when "0111" => Y <= "1011"; -- B
            when "1000" => Y <= "1000"; -- 8
            when "1001" => Y <= "0001"; -- 1
            when "1010" => Y <= "0111"; -- 7
            when "1011" => Y <= "1000"; -- 8
            when "1100" => Y <= "1010"; -- A
            when "1101" => Y <= "1000"; -- 8
            when "1110" => Y <= "0101"; -- 5
            when "1111" => Y <= "0010"; -- 2
```



```

    when others => null;
--в остальных случаях - null (пустой оператор)- ничего не делать
--случай when others фактически не нужен, т.к. были рассмотрены все
--варианты входных значений, но настоятельно рекомендуется данную
--конструкцию использовать при каждом применении оператора case
    end case;
--конец процедуры case
end process;
--конец процесса
end Behavioral;
--конец описания архитектуры

```

Итак, сохраняйте файл. Далее вам рекомендуется создать поведенческий Verilog модуль. Для этого создайте исходный модуль типа Verilog Modul. Вам предлагается назвать его Lab\_1\_vlog. Перед вами снова появится окно, предлагающее сделать заготовки входов и выходов устройства. Пропустите этот момент и завершите создание модуля.

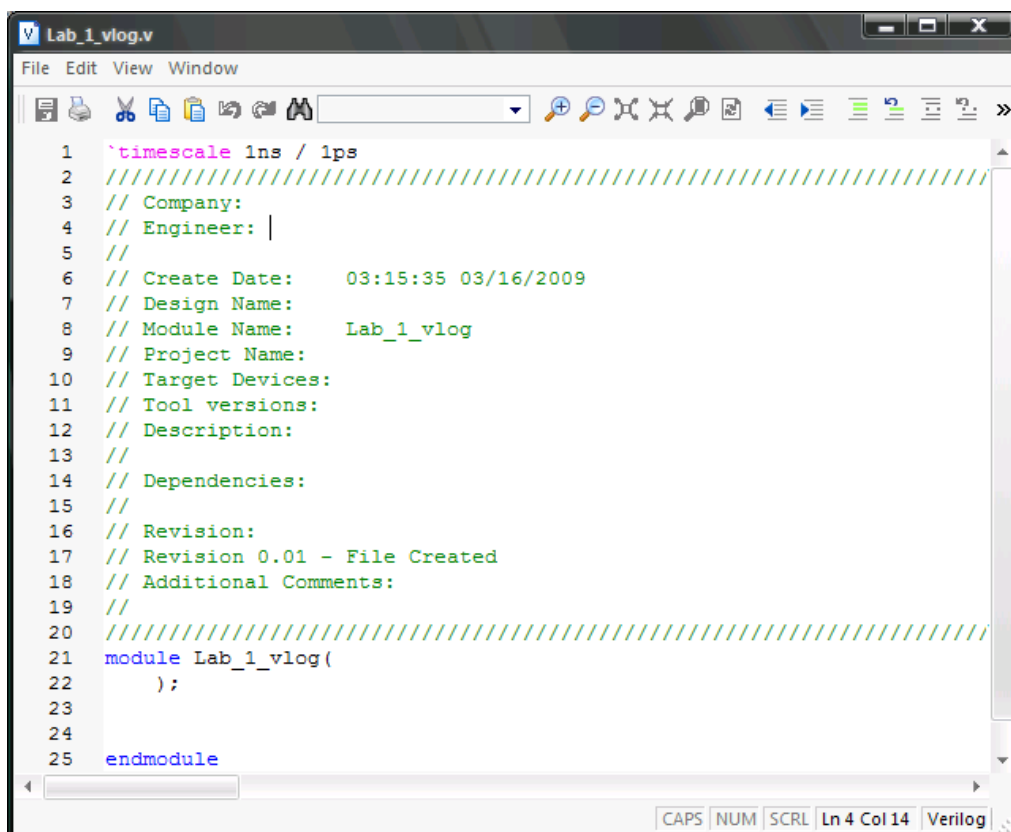


Рисунок 6. Внешний вид Verilog модуля.

В языке Verilog нет разделение на описание интерфейса и описание архитектуры. Все находится в едином блоке (рис. 6):

```
Module Lab_1_vlog();
```

```
endmodule
```

Итак, ниже вам будет изложен код с подробным описанием, что делает каждая строка.

```
`timescale 1ns / 1ps
//директива симулятора - установка шага времени
module Lab_1_vlog(
//Заголовок модуля
    input      [3:0] X,
//описание входов
    output reg [3:0] Y
//описание выходов
//запись reg в данном случае означает, что для шины создается
//регистр, который по сути является буфером
);
    always @ (X)
//аналог процесса в VHDL, x - сигнал, участвующий в процессе в
//качестве аргумента в списке чувствительности
//@ означает, что список чувствительности не может быть пустым
    begin
//начало описания поведения
        case (X)
//применение процедуры case к входным значениям X
            4'hF : Y <= 4'b0010; // 2
//если X равен F (в hex коде), то выходу Y присвоить 0010 (в
//бинарном коде)
            4'hE : Y <= 4'b0101; // 5
            4'hD : Y <= 4'b1000; // 8
            4'hC : Y <= 4'b1010; // A

            4'hB : Y <= 4'b1000; // 8
            4'hA : Y <= 4'b0111; // 7
            4'h9 : Y <= 4'b0001; // 1
            4'h8 : Y <= 4'b1000; // 8

            4'b0111 : Y <= 4'hB; // B
            4'b0110 : Y <= 4'h5; // 5
            4'b0101 : Y <= 4'h9; // 9
            4'b0100 : Y <= 4'hE; // E

            3 : Y <= 15; // F
//если X равен 3 (в десятичном коде), то выходу Y присвоить 15 (тоже
//в десятичном //коде)
            2 : Y <= 4; // 4
            1 : Y <= 12; // C
            0 : Y <= 0; // 0
        endcase
//конец процедуры case
    end
//конец описания поведения
```

```
endmodule
//конец описания модуля
```

Далее вам предлагается создать поведенческий модуль, объединяющий в себе два ранее созданных поведенческих модуля. Для этого создайте исходный модуль типа VHDL Modul. Вам предлагается назвать его Lab\_1\_beh.

Ниже приведено его содержание с подробным описанием.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--подключение библиотек
entity Lab_1_beh is
--заголовок описания интерфейса
    Port ( X : in  STD_LOGIC_VECTOR (3 downto 0);
           Y_VHDL : out  STD_LOGIC_VECTOR (3 downto 0);
           Y_VLOG : out  STD_LOGIC_VECTOR (3 downto 0));
--описание портов в интерфейсе
--описание входных и выходных шин
end Lab_1_beh;
--конец описания интерфейса
architecture Behavioral of Lab_1_beh is
--заголовок описания архитектуры с именем Behavioral
    component Lab_1_vhdl
--создание компонента с названием Lab_1_vhdl
        Port (      X : in  STD_LOGIC_VECTOR (3 downto 0);
                 Y : out  STD_LOGIC_VECTOR (3 downto 0));
--создание входов и выходов компонента
    end component;
--конец описания компонента
    component Lab_1_vlog
        Port (      X : in  STD_LOGIC_VECTOR (3 downto 0);
                 Y : out  STD_LOGIC_VECTOR (3 downto 0));
    end component;
begin
--начало описания архитектуры
    VHDL_MOD : Lab_1_vhdl
--создание интерфейса VHDL_MOD
--прикрепление VHDL модуля с названием Lab_1_vhdl
--прикрепление осуществимо, если VHDL модуль с именем Lab_1_vhdl
--существует в проекте
    port map (
--описание связей
        X => X,
        Y => Y_VHDL);
    VLOG_MOD : Lab_1_vlog
    port map (
```

```

X => X,
Y => Y_VLOG);
end Behavioral;
--конец описания архитектуры

```

Далее вам предлагается создать вентильный модуль VHDL. Вентильный уровень программируется путем задания логического соответствия входов и выходов схемы. Поэтому, прежде чем приступить к созданию вентильного модуля, необходимо найти логические функции для выходных значений. Находить функции вам рекомендуется, используя карты Карно (табл. 2, табл. 3, табл. 4, табл. 5):

Y0				
x0x1 \ x2x3	00	01	11	10
00	0	1	1	0
01	1	1	1	0
11	1	1	0	0
10	1	0	1	0

Y2				
x0x1 \ x2x3	00	01	11	10
00	0	0	1	0
01	1	0	1	0
11	1	0	1	0
10	0	0	0	1

Y1				
x0x1 \ x2x3	00	01	11	10
00	0	1	1	1
01	1	0	0	1
11	0	0	0	1
10	0	0	0	1

Y3				
x0x1 \ x2x3	00	01	11	10
00	0	0	1	0
01	0	1	1	1
11	0	0	0	1
10	0	1	0	1

Слева направо и сверху вниз: Таблица 2, Таблица 3, Таблица 4, Таблица 5.

$Y0 = (|(X3) \& X1 \& X0) + (|(X3) \& X2 \& X0) + (X2 \& X1 \& |(X0)) + (X3 \& X1 \& |(X0)) + (X3 \& |(X2) \& |(X1) \& X0);$

$Y1 = (|(X3) \& X1 \& X0) + (X2 \& X1 \& X0) + (X2 \& |(X1) \& |(X0)) + (X3 \& |(X2) \& X1 \& |(X0));$

$Y2 = (X1 \& |(X0)) + (|(X3) \& |(X2) \& X0) + (|(X3) \& X2 \& |(X0));$

$Y3 = (|(X3) \& X0) + (X2 \& |(X1)) + (X3 \& |(X1) \& |(X0)) + (|(X2) \& X1 \& X0);$



*Вычислять функции рекомендуется с помощью специальных программ типа Karna Minimizer.*

Теперь можно приступать к созданию вентиляльных модулей. Создайте исходный модуль типа VHDL Modul. Вам предлагается назвать его Lab\_1\_vent. В качестве Architecture name вам рекомендуется использовать имя Ventil (по умолчанию указывается имя Behavioral, а вы находитесь в процессе создания не поведенческого, а вентиляльного уровня).

Ниже приведено содержание исходного VHDL модуля с подробным описанием.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--подключение библиотек
entity Lab_1_vhdl_vent is
--заголовок описания интерфейса
    Port ( X0 : in  STD_LOGIC;
           X1 : in  STD_LOGIC;
           X2 : in  STD_LOGIC;
           X3 : in  STD_LOGIC;
           Y0 : out STD_LOGIC;
           Y1 : out STD_LOGIC;
           Y2 : out STD_LOGIC;
           Y3 : out STD_LOGIC);
--описание входов и выходов в виде одноразрядных линий
end Lab_1_vhdl_vent;
--конец описания интерфейса
architecture Ventil of Lab_1_vhdl_vent is
--заголовок описания архитектуры
begin
--начало описания архитектуры
    Y0 <=(not(X3) and X1 and X0 )or
        (not(X3) and X2 and X0 )or
        ( X2 and X1 and not(X0))or
        ( X3 and X1 and not(X0))or
        ( X3 and not(X2) and not(X1) and X0 );
--описание поведения выхода Y0 в форме задания логической функции

    Y1 <=(not(X3) and X1 and X0 )or
        ( X2 and X1 and X0 )or
        ( X2 and not(X1) and not(X0))or
        ( X3 and not(X2) and X1 and not(X0));
--//--
    Y2 <=( X1 and not(X0))or
        (not(X3) and not(X2) and X0 )or
        (not(X3) and X2 and not(X0));
--//--
    Y3 <=(not(X3) and X0 )or
        ( X2 and not(X1) )or
```

```

        (      X3   and                not(X1) and not(X0)) or
        (                not(X2) and      X1   and      X0 );
--//--
end Ventil;
--конец описания архитектуры

```

Наконец в последнюю очередь, вам предлагается создать топовый (главный) модуль, объединяющий в себе два ранее созданные поведенческий и вентильный модули, а также содержащий в себе вентильный уровень описания схемы, написанный на языке Verilog. Для этого создайте исходный модуль типа Verilog Modul. Вам предлагается назвать его Lab\_1\_top.

Ниже приведено содержание исходного Verilog модуля с подробным описанием.

```

`timescale 1ns / 1ps
//директива симулятора – установка шага времени
module Lab_1_top(
//заголовок модуля
    input [3:0] X,
    output [3:0] Y_VHDL_B,
    output [3:0] Y_VLOG_B,
    output [3:0] Y_VHDL_V,
    output [3:0] Y_VLOG_V
);
//описание конечного интерфейса устройства
Lab_1_beh HDL_Beh (
//подключение ранее созданного модуля Lab_1_beh
    .X(X),
    .Y_VHDL(Y_VHDL_B),
    .Y_VLOG(Y_VLOG_B));
//Задание соответствия входов и выходов
assign Y_VLOG_V[0] = ~X[3] &          X[1] &  X[0] |
                    ~X[3] &  X[2] &          X[0] |
                    X[2] &  X[1] & ~X[0] |
                    X[3] &          X[1] & ~X[0] |
                    X[3] & ~X[2] & ~X[1] &  X[0];
//описание поведения выхода Y_VLOG_V[1] в форме задания логической функции
assign Y_VLOG_V[1] = ~X[3] &          X[1] &  X[0] |
                    X[2] &  X[1] &  X[0] |
                    X[2] & ~X[1] & ~X[0] |
                    X[3] & ~X[2] &  X[1] & ~X[0];
//-----
assign Y_VLOG_V[2] =          X[1] & ~X[0] |
                    ~X[3] & ~X[2] &          X[0] |
                    ~X[3] &  X[2] &          ~X[0];
//-----

```

```

assign Y_VLOG_V[3] = ~X[3] & X[0] |
                    X[2] & ~X[1] |
                    X[3] & ~X[1] & ~X[0] |
                    ~X[2] & X[1] & X[0];

//-----
Lab_1_vhdl_vent Ventil (
//подключение ранее созданного модуля Lab_1_vhdl_vent и описание
интерфейса
//Ventil
.X0(X[0]),
.X1(X[1]),
.X2(X[2]),
.X3(X[3]),
.Y0(Y_VHDL_V[0]),
.Y1(Y_VHDL_V[1]),
.Y2(Y_VHDL_V[2]),
.Y3(Y_VHDL_V[3]));
//задание списка соответствия входов и выходов
endmodule
//конец описания модуля

```

Лабораторная работа на данном этапе почти выполнена. Вам остается пройти стадию функциональной верификации проекта. Для этого создайте исходный модуль типа Test Bench Waveform. Вам предлагается назвать его Lab\_1\_test\_top. Создание данного модуля описано в разделе «Функциональная верификация». В качестве тестовой последовательности вам необходимо ввести последовательность чисел от 15 до 0 в hex-коде (т.е. от F до 0). После проведения моделирования поведенческой модели, все выходные значения должны совпасть (рис. 7).

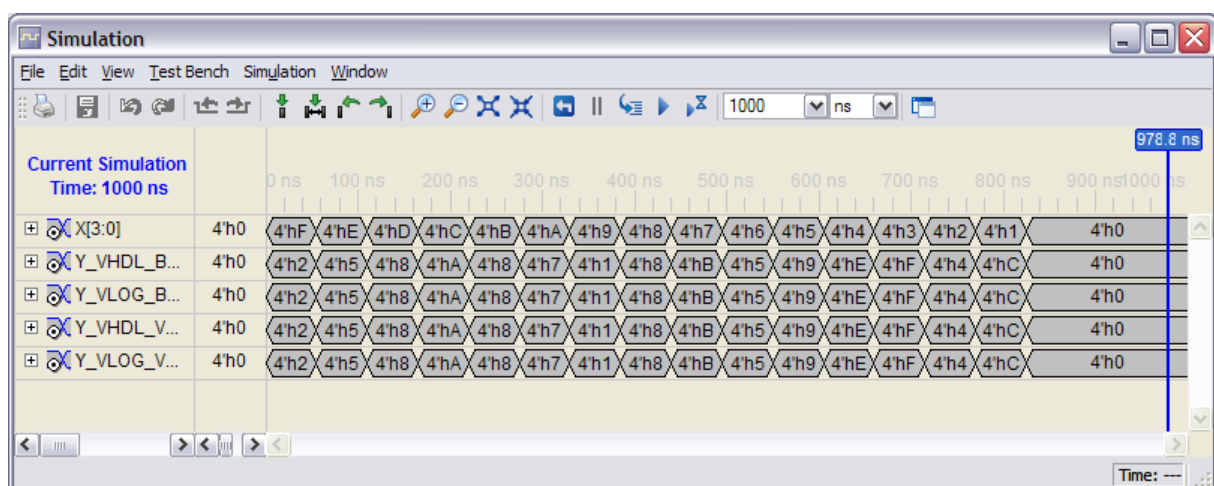


Рисунок 7. Временная диаграмма работы устройства.

## Лабораторная работа № 2

В лабораторной работе №2 требуется построить цифровой узел с элементами памяти, представляющий собой генератор последовательности. Последовательность генерируется в виде бесконечного циклического набора 4 разрядных слов. Генератор представляет собой счетчик имеющий модуль счета **16**. Каждому состоянию ставится в соответствие некоторая выходная комбинация. Задание выдаётся в виде таблицы, в верхней строке которой перечисляются в порядке возрастания входные комбинации от «0» (0000) до «F» (1111), а в нижней строке приведены соответствующие им выходные комбинации (табл. 1).

Структурная схема генератора последовательности представлена на рисунке (рис. 8):

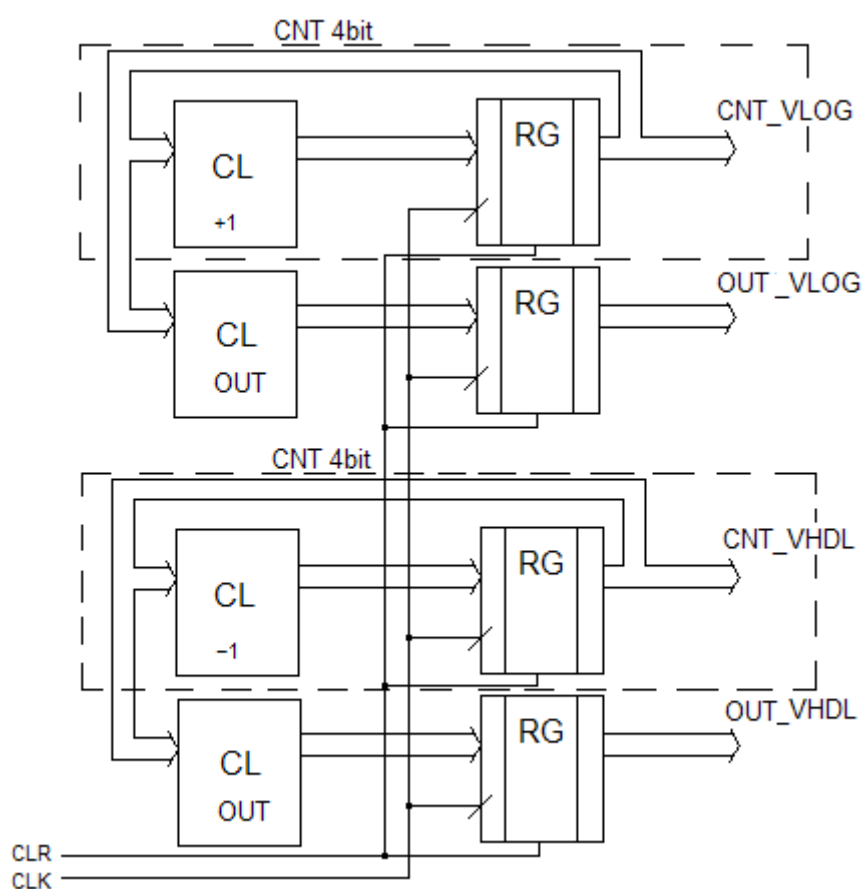


Рисунок 8. Структурная схема генератора последовательности.

Пунктирными линиями обведены счетчики, вырабатывающие состояния, CNT – состояние счетчика, OUT – Выходная шина, генерирующая последовательность. Студенту требуется создать 2 генератора: на языке Verilog, выдающий последовательность в прямом порядке, и на языке VHDL, выдающий последовательность в обратном порядке (табл. 6):

CNT_VLOG	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
OUT_VLOG	0	C	4	F	E	9	5	B	8	1	7	8	A	8	5	2



CNT_VHDL	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
OUT_VHDL	2	5	8	A	8	7	1	8	B	5	9	E	F	4	C	0

Таблица 6. Задание на лабораторную работу (13 вариант).

Для каждой из построенных схем необходимо представить временную диаграмму работы, на которой должны отображаться состояния и сама последовательность.

Диаграммы и схемы вставляются в отчёт по лабораторной работе в том виде, как они представлены в САПР.

В данной работе рассмотрен один вид абстракции - поведенческий уровень.

Вам предлагается создать следующую иерархическую структуру (рис. 9):

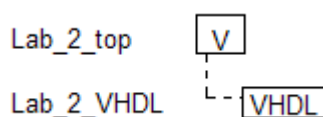


Рисунок 9. Иерархия проекта.

Таким образом, интерфейс разрабатываемого устройства должен иметь следующий вид (рис. 10):

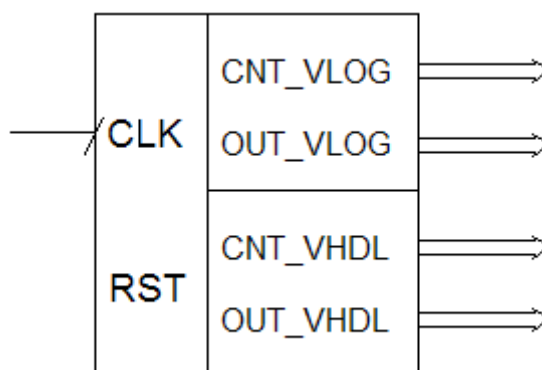


Рисунок 10. Интерфейс устройства.

- где CNT\_VLOG и OUT\_VLOG – выходы модуля Lab\_2\_top, CNT\_VHDL и OUT\_VHDL – выходы модуля Lab\_2\_VHDL.

Создайте проект, следуя инструкции, описанной в разделе «Лабораторная работа по моделированию № 1». Затем вам предлагается создать поведенческий модуль VHDL (следуйте инструкциям, описанной в разделе «Лабораторная работа по моделированию № 1»). Вам предлагается назвать его Lab\_2\_VHDL.

Ниже приведено его содержание с подробным описанием.

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--подключение библиотек
  
```

```

entity Lab_2_VHDL is
--Заголовок блока описания интерфейса
    Port ( CLK    : in  STD_LOGIC;
          RST     : in  STD_LOGIC;
          CNT_O   : out STD_LOGIC_VECTOR (3 downto 0);
          VAR_O   : out STD_LOGIC_VECTOR (3 downto 0));
--создание линий синхросигнала, сброса и двух 4 разрядных
--выходных шин
end Lab_2_VHDL;
--конец описания интерфейса
architecture Behavioral of Lab_2_VHDL is
--заголовок описания архитектуры
signal STATE : STD_LOGIC_VECTOR (3 downto 0);
--создание локальной виртуальной шины типа «сигнал», которая
--впоследствии станет шиной состояния счетчика
begin
--начало описания архитектуры
    CNT : process(CLK, RST) begin
--создание нового процесса, содержащего инструкции по поведению
--генератора последовательности
--в скобках – список чувствительности от входных параметров
        if(RST = '1') then
--если сигнал сброса активен
            STATE <= "1111";
--сброс STATE в состояние «1111» (начальное состояние для
--реверсивного счетчика)
            VAR_O <= "0010";
--присвоение выходной шине значения, соответствующей состоянию
--«1111»
            elsif(CLK'event and CLK = '1') then
--если сигнал CLK изменил свое значение и стал равен '1'
                STATE <= STATE - 1;
--декремент счетчика
--когда счетчик доходит до нуля никакой ошибки не возникает, т.к.
--модуль счета равен 16-и
--если модуль счета < 2^n, где n –разрядность шины, то вместо строки
--декремента виртуального сигнала надо было бы написать:
--if(STATE=0)then STATE <= <модуль_счета - 1>;
--else STATE <= STATE -1;
--end if;
                case STATE is
--применение условного оператора case к виртуальному сигналу STATE
                    when "1111" => VAR_O <= "0101"; -- 5
                    when "1110" => VAR_O <= "1000"; -- 8
                    when "1101" => VAR_O <= "1010"; -- A
                    when "1100" => VAR_O <= "1000"; -- 8
                    when "1011" => VAR_O <= "0111"; -- 7
                    when "1010" => VAR_O <= "0001"; -- 1

```

```

when "1001" => VAR_O <= "1000"; -- 8
when "1000" => VAR_O <= "1011"; -- B
when "0111" => VAR_O <= "0101"; -- 5
when "0110" => VAR_O <= "1001"; -- 9
when "0101" => VAR_O <= "1110"; -- E
when "0100" => VAR_O <= "1111"; -- F
when "0011" => VAR_O <= "0100"; -- 4
when "0010" => VAR_O <= "1100"; -- C
when "0001" => VAR_O <= "0000"; -- 0
when "0000" => VAR_O <= "0010"; -- 2
--присвоение выходной шине значения, соответствующей следующему
--состоянию т.к. происходит задержка при присвоении выходного
--сигнала в 1 такт, необходимо присваивать значение следующего
--значения
    when others => null;
--в остальных случаях - null (пустой оператор)- ничего не делать
end case;
--конец процедуры case
End if;
end process;
--конец процесса
CNT_O <= STATE;
--присвоение шине состояния значения виртуальной шины STATE
end Behavioral;
--конец описания архитектуры

```

Сохраните файл. Далее вам предлагается создать топовый (главный) поведенческий модуль Verylog (следуйте инструкциям, описанной в разделе «Лабораторная работа по моделированию № 1»). Вам предлагается назвать его Lab\_2\_top.

Ниже приведено содержание исходного Verylog модуля с подробным описанием.

```

`timescale 1ns / 1ps
//директива симулятора – установка шага времени
module Lab_2_top(
//Заголовок модуля
    input          CLK,
    input          RST,
    output [3:0] CNT_VHDL,
    output [3:0] OUT_VHDL,
    output [3:0] CNT_VLOG,
    output reg [3:0] OUT_VLOG
);
//создание линий синхросигнала, сброса и четырех 4 разрядных
выходных шин
reg [3:0] STATE;
//создание виртуальной 4-х разрядной шины

```

```

always @ (posedge CLK, posedge RST)
//условие синхронизации: здесь идет срабатывание по одному любому из
//указанных в списке чувствительности сигналу
//posedge – обозначение позитивного фронта
if(RST) STATE <= 0;
//если сигнал сброса активен (=1) – сброс счетчика в нуль
else
//иначе
STATE <= STATE + 1;
//инкремент счетчика
//если модуль счета < 2^n, где n –разрядность шины, то вместо строки
//инкремента виртуального сигнала надо было бы написать
//дополнительные условные преобразования
//if(STATE=<модуль_счета - 1>) STATE <=0;
//else STATE <= STATE + 1;
assign CNT_VLOG = STATE;
//непрерывное присвоение выходной шине значения, соответствующей
//текущему состоянию
always @ (posedge CLK, posedge RST)
//описание поведения счетчика; условие синхронизации: по
//синхросигналу и сигналу сброса
begin
//начало процесса
if(RST) OUT_VLOG <= 0;
//описание поведения «выхода»; условие синхронизации: по
// сигналу сброса
else
//иначе (при активном сигнале синхронизации)
case (STATE)
//применение условного оператора case к виртуальному сигналу STATE
4'h0 : OUT_VLOG <= 4'hC;
4'h1 : OUT_VLOG <= 4'h4;
4'h2 : OUT_VLOG <= 4'hF;
4'h3 : OUT_VLOG <= 4'hE;

4'h4 : OUT_VLOG <= 4'h9;
4'h5 : OUT_VLOG <= 4'h5;
4'h6 : OUT_VLOG <= 4'hB;
4'h7 : OUT_VLOG <= 4'h8;

4'h8 : OUT_VLOG <= 4'h1;
4'h9 : OUT_VLOG <= 4'h7;
4'hA : OUT_VLOG <= 4'h8;
4'hB : OUT_VLOG <= 4'hA;

4'hC : OUT_VLOG <= 4'h8;
4'hD : OUT_VLOG <= 4'h5;
4'hE : OUT_VLOG <= 4'h2;
4'hF : OUT_VLOG <= 4'h0;

```

```

//присвоение выходной шине значения, соответствующей текущему
//состоянию
endcase
//конец процедуры case
end
//конец описания поведения «выхода»
Lab_2_VHDL INT_MOD (
//прикрепление из модуля Lab_2_VHDL сигналов
.CLK(CLK),
.RST(RST),
.CNT_O(CNT_VHDL),
.VAR_O(OUT_VHDL));
//задание соответствия между сигналами
endmodule
//конец модуля

```

Лабораторная работа на данном этапе почти выполнена. Вам остается пройти стадию верификации проекта. Для этого создайте исходный модуль типа Test Bench Waveform. Вам предлагается назвать его Lab\_1\_test\_top. Создание данного модуля описано в разделе «Функциональная верификация». Вам необходимо лишь задать сигнал (чередование высокого и низкого уровней), а также проверить работу сигнала сброса (рис. 11).

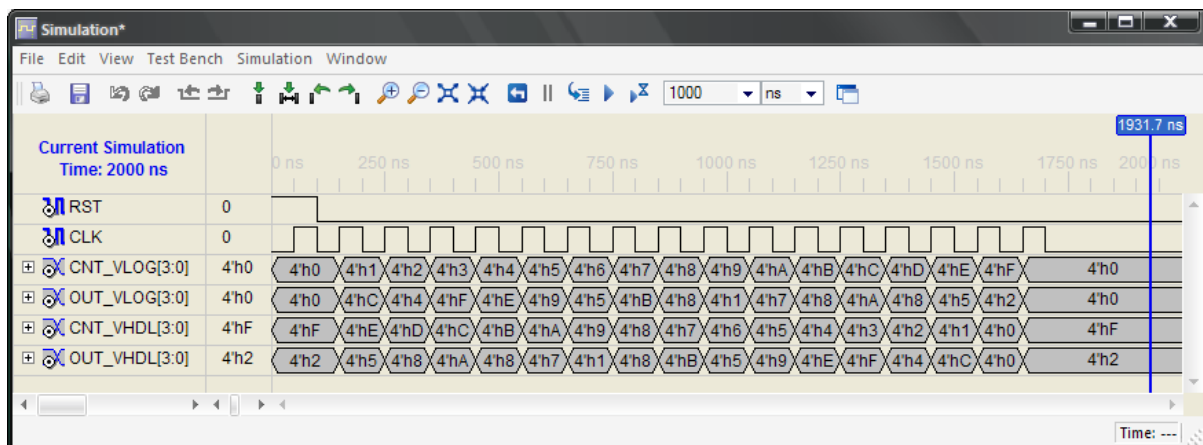


Рисунок 11. Временная диаграмма работы устройства.

### Лабораторная работа № 3

В лабораторной работе №3 требуется построить цифровой узел с элементами памяти, представляющий собой программируемый генератор последовательности, который выдает процессору серию из 16 прерываний, сопровождаемых статусными комбинациями из последовательности варианта, причем временной интервал между прерываниями в последовательности задается программным способом в виде записи количества тактов синхросигнала. После записи начинается выдача

прерываний с заданным интервалом до тех пор, пока не произойдет 16-е прерывание.

К устройству предъявляется требование синхронности (т.е. оно должно срабатывать по синхросигналу). Поэтому требуется использовать описание устройства в 1 процессе.

Последовательность соответствует варианту, указанному в таблице заданий (табл. 1). Студентам с четными вариантами требуется построить схему на VHDL, с нечетными – на Verilog.

Интерфейс разрабатываемого устройства будет иметь следующий вид (рис.12):

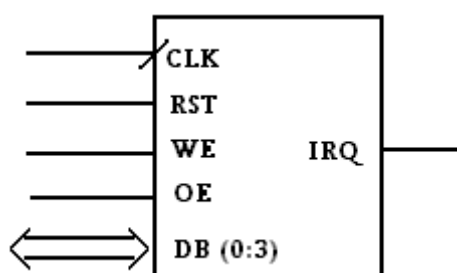


Рисунок 12. Интерфейс устройства.

В данном устройстве (рис.13) содержатся следующие сигналы:

- CLK – синхросигнал (входной)
- RST – сигнал сброса (входной)
- WE – входной сигнал разрешения на запись
- OE – входной сигнал разрешения на чтение
- IRQ – выходной сигнал прерываний

Также устройство должно иметь шину:

- DB – тристабильная шина данных (имеет тип inout)

Более детальное изображение устройства указано на следующем рисунке. Данное устройство включает:

- CNT – счетчики, у первого модуль счета программируется посредством ШД, а у второго он постоянен и равен 16;
- CL – комбинационная логика, генерирующая заданную последовательность.

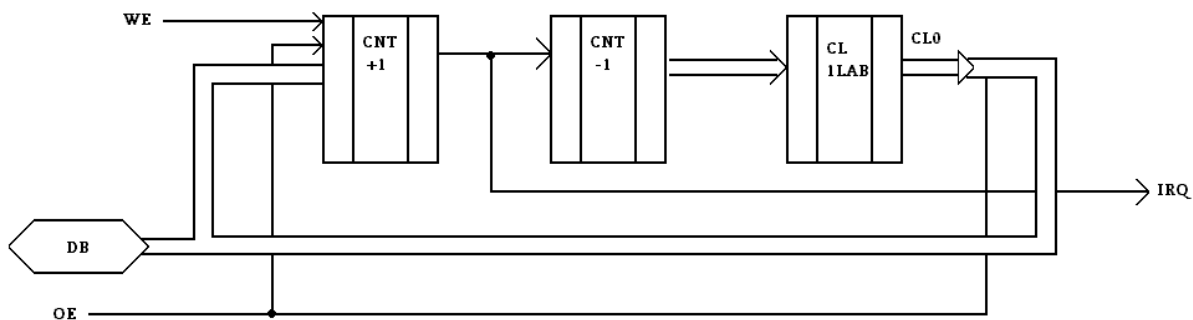


Рисунок 13. Структурная схема программируемого генератора последовательности.

Тристабильный буфер представляет собой следующую структуру (рис.14):

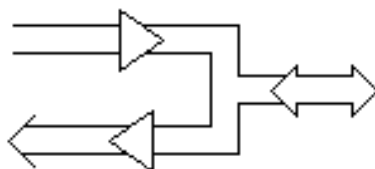


Рисунок 14. Структурная схема тристабильного буфера.

Т.е. он является выходной, совмещенной с входной шиной. Тристабильный буфер строится по следующему принципу:

- Если шина свободна от чтения ( $DB='Z'$ ) – можно производить запись
- Если на шину ничего подается – можно производить чтение
- В остальных случаях шина должна находиться в состоянии высокого импеданса ( $DB='Z'$ )

Читаемая информация определяется внутренней шиной состояния (рис. 15). Во избежание конфликтов она должна в каждый момент времени содержать в себе какое-либо значение (должна быть определена):

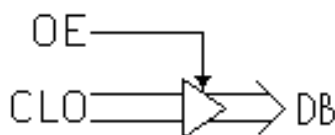


Рисунок 15. Интерфейс тристабильного буфера.

Еще один конфликт связан с тем, что значения одновременно могут пытаться подаваться и считываться с тристабильной шины одновременно.

В этих двух случаях значение шины принимает значение 'X' – неопределенности.

Чтобы ознакомиться с синтаксисом применения тристабильного буфера необходимо выбрать закладку в главном меню управления **Edit** → **Language Templates...** и перейти далее **Verilog** (или **VHDL**) → **Synthesis Constructs** → **Coding Examples** → **Tristate Buffers**. Здесь содержится

синтаксическое описание построения тристабильного буфера. Также вы можете ознакомиться с синтаксисом использования других устройств. **Внимание!** Если у вас не отображается **Language Templates...** на панели размещения редакторов, вы не сможете вынести этот объект в отдельное окно. С этой целью вам придется синтаксис использования тристабильного буфера.

### Verilog:

DB = OE ? CLO : 4'hZ;

Здесь описано поведение шины данных: если OE='1', то шина данных должна выдавать значение шины CLO (шина внутреннего состояния); если OE='0', то шина данных находится в состоянии высокого импеданса; в этом состоянии можно производить запись на шину.

### VHDL:

DB <= CLO when OE = '1' else 4'hZ;

По аналогии с Verilog.

При функциональной верификации для данного устройства вам предлагается использовать следующие настройки (рис. 16):

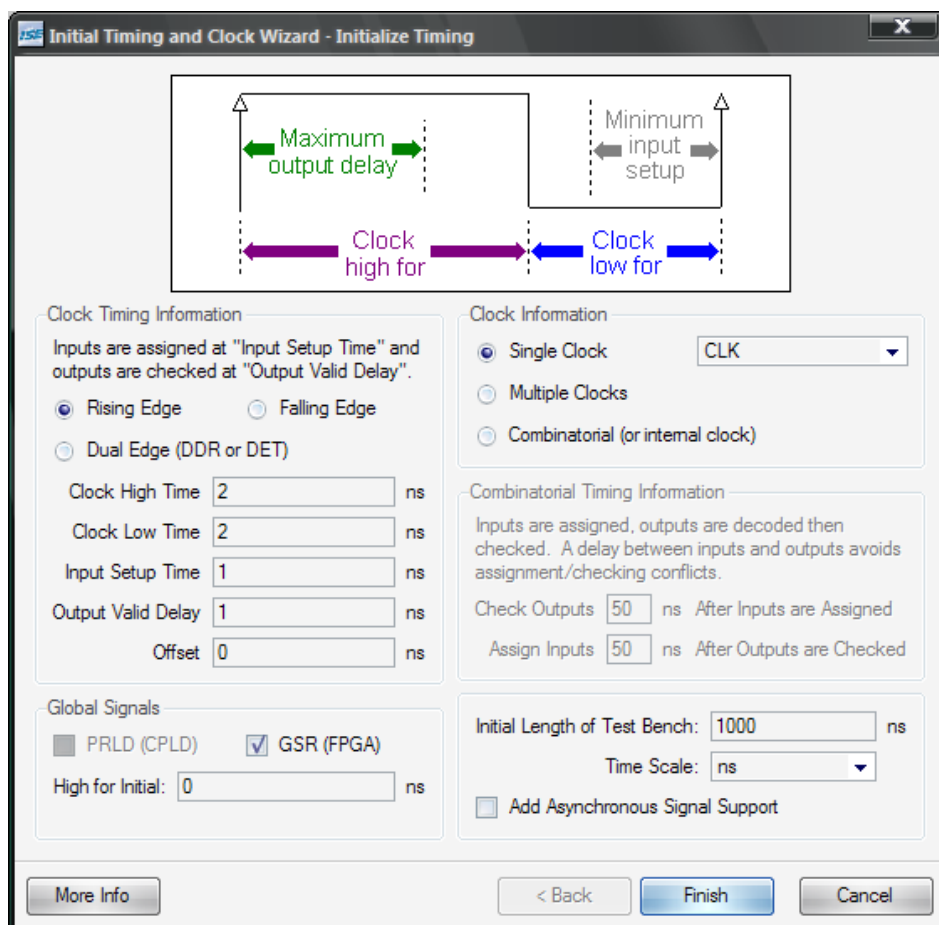


Рисунок 16. Настройки временных параметров симуляции.

В качестве Single Clock (сигнала синхронизации) выберите ваш синхросигнал (в этом случае по окончании создания модуля этот сигнал примет вид импульсного сигнала прямоугольной формы), Clock Hide и



Low Time (время высокого и низкого сигнала) выберите 2 (делать больше нет особого смысла). В качестве Input Setup Time & Output Valid Delay (времени установки входных и допустимой задержки установки выходных значений) задайте 1 (эти задержки должны быть меньше времени высокого и низкого уровней сигналов соответственно). Offset необходимо задать нулевым.

В процессе построения устройства вы должны получить следующую временную диаграмму (рис. 17):

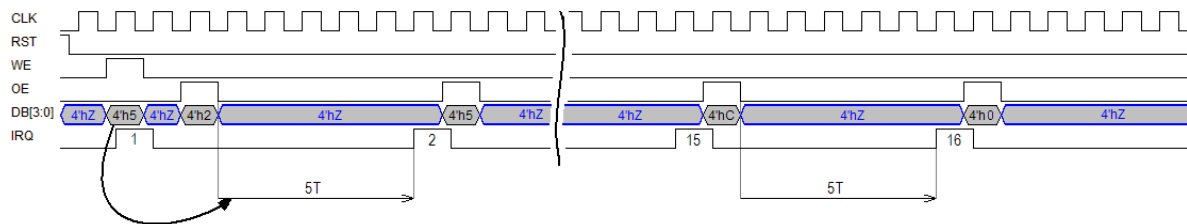


Рисунок 17. Временная диаграмма работы устройства.

*Считается, что входные значения, поступающие на устройство, исключают конфликтные ситуации, к примеру, на устройство не может подаваться сигнал чтения и записи одновременно, поэтому особые случаи рассматривать не требуется.*



*Список чувствительности процессов должен включать только линию сброса и синхронизации.*

## Лабораторная работа № 4

Целью данной работы является изучение работы с IP блоками и с VHDL библиотеками. В лабораторной работе №3 требуется построить синхронный цифровой узел, представляющий собой генератор последовательности. Последовательность генерируется в виде бесконечного циклического набора 4 разрядных слов. Генератор представляет собой счетчик имеющий модуль счета **16**, реализованный в виде IP блока. Каждому состоянию ставится в соответствие некоторая выходная комбинация. Задание выдаётся в виде таблицы, в верхней строке которой перечисляются в порядке возрастания входные комбинации от «0» (0000) до «F» (1111), а в нижней строке приведены соответствующие им выходные комбинации (табл.1.5.1.1). Данный комбинационный узел (ставящий в соответствие состоянию счетчика выходную комбинацию) должен быть реализован в виде отдельного библиотечного элемента.

Структурная схема генератора последовательности представлена на рисунке (рис. 18):

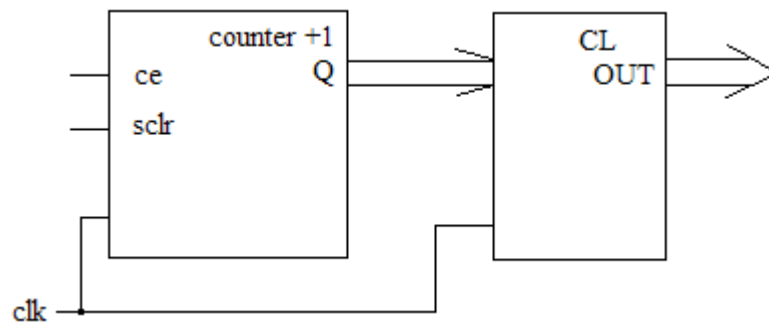


Рисунок 18. Структурная схема устройства.

Counter +1 - счетчик, вырабатывающий состояния, Q – состояние счетчика, OUT – Выходная шина, генерирующая последовательность. Студенту требуется создать генератор на языке VHDL, выдающий последовательность в прямом порядке (табл. 7):

Q	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
OUT	0	C	4	F	E	9	5	B	8	1	7	8	A	8	5	2

Таблица 7. Задание на лабораторную работу (13 вариант).

Вам предлагается создать следующую иерархическую структуру (рис. 19):

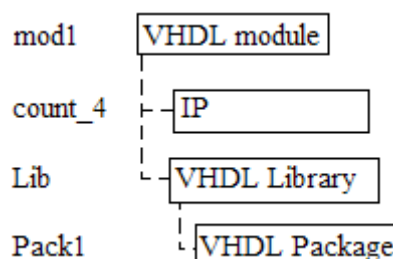


Рисунок 19. Иерархическая структура проекта.

Таким образом, интерфейс разрабатываемого устройства должен иметь следующий вид (рис. 20):

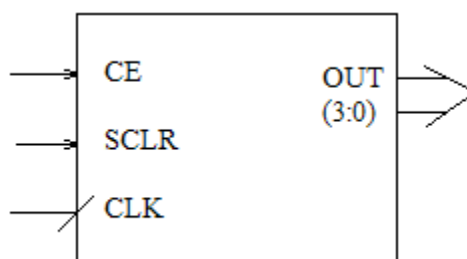


Рисунок 20. Интерфейс устройства.

- где OUT – выход модуля mod1, CE – разрешающий сигнал, SCLR – сигнал сброса счетчика, который сбрасывает счетчик на следующем после срабатывания такте.

Создайте проект, следуя инструкции, описанной в разделе «Лабораторная работа по моделированию № 1» (рис. 21). Затем создайте IP блок со счетчиком. Вам предлагается назвать его count\_4.

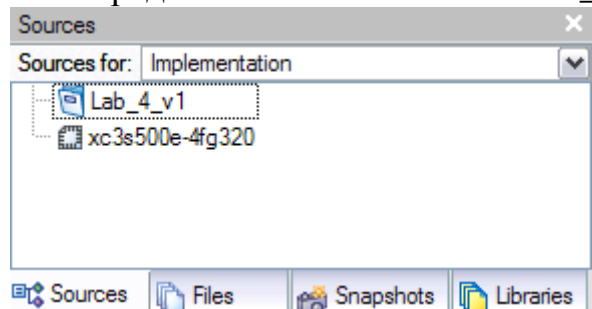


Рисунок 21. Начальная иерархическая структура проекта.

Для этого щелкните правой кнопкой мыши по полю **Sources** и во всплывшем меню выберите пункт **New Source...**

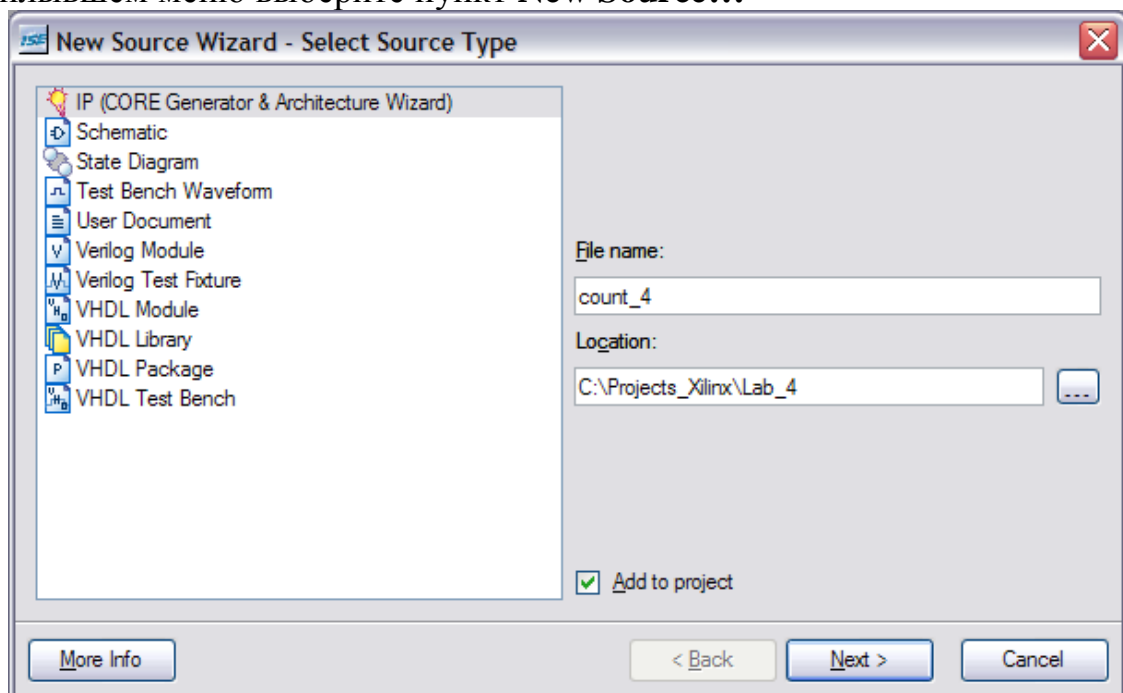


Рисунок 22. Окно создания модуля.

В данном окне (рис. 22) выберите тип создаваемого модуля **IP (CORE Generator & Architecture Wizard)**. Нажмите **Next**.

Перед вами появится окно выбора IP блока (рис. 23).

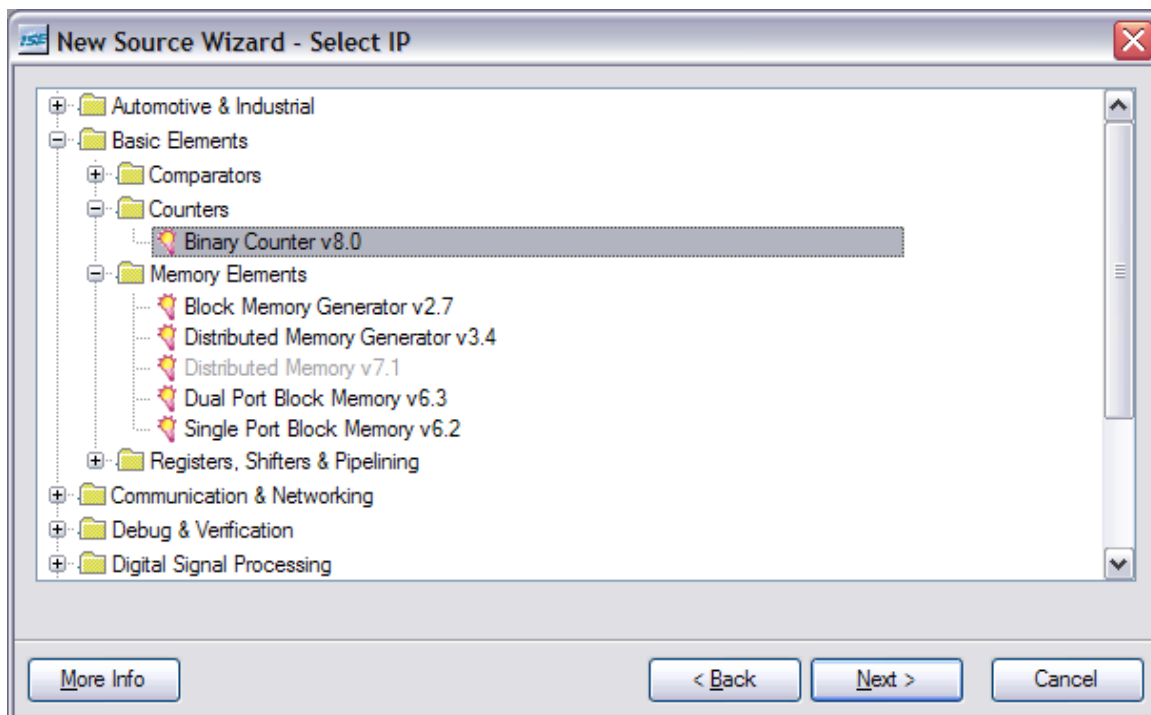


Рисунок 23. Окно выбора IP блока.

Здесь Вам следует выбрать IP блок двоичного счетчика, для этого перейдите по директории **Basic Elements -> Counters -> Binary Counter v8.0**.



*В связи с тем, что данная версия САПР является свободнораспространяемой, некоторые IP блоки не доступны для общего пользования. Такие IP блоки выделены серым цветом, как например Distributed Memory Generator v2.7 на рис.*

23.

Нажмите **Next**.

Перед вами появится окно отчета выбора IP блока (рис. 24).

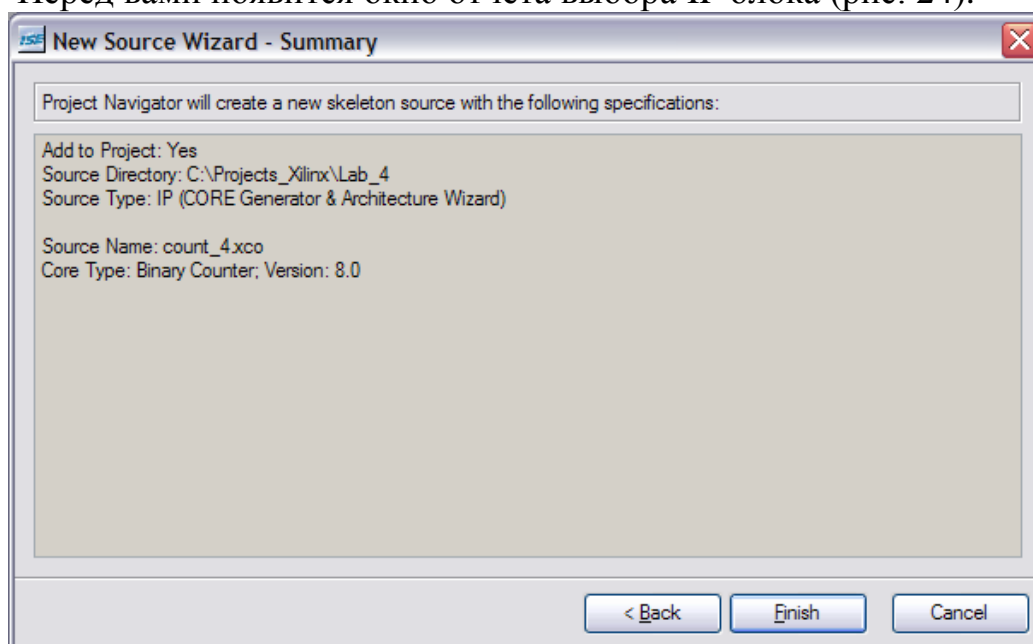


Рисунок 24. Окно отчета выбора IP блока.

Нажмите **Finish**.

Перед вами появится окно выбора настроек IP блока (рис. 25). Для получения подробной информации о настройках данного IP блока воспользуйтесь кнопкой **View Data Sheet**. При нажатии этой кнопки откроется руководство по настройкам в формате .pdf на английском языке.

В качестве имени IP блока Вам предлагается оставить count\_4. Глубину счетчика (**Output Width**) задайте равную 4 разрядам. Шаг счета (**Step Value**) задайте равную 1. Флаг **Restrict Count** (установка конечного состояния счетчика) оставьте неактивным. Данный флаг нужен, например, для установления конечного состояния счетчика (модуль счета) равную 9: разрядность счетчика в таком случае тоже составляла бы 4 разряда. В Вашем случае Вам нужен модуль счета равный 16, поэтому установка конечного состояния счетчика здесь не требуется. В качестве параметра **Count Mode** (направление счета) выберите **UP**. В этом случае счетчик будет инкрементироваться.

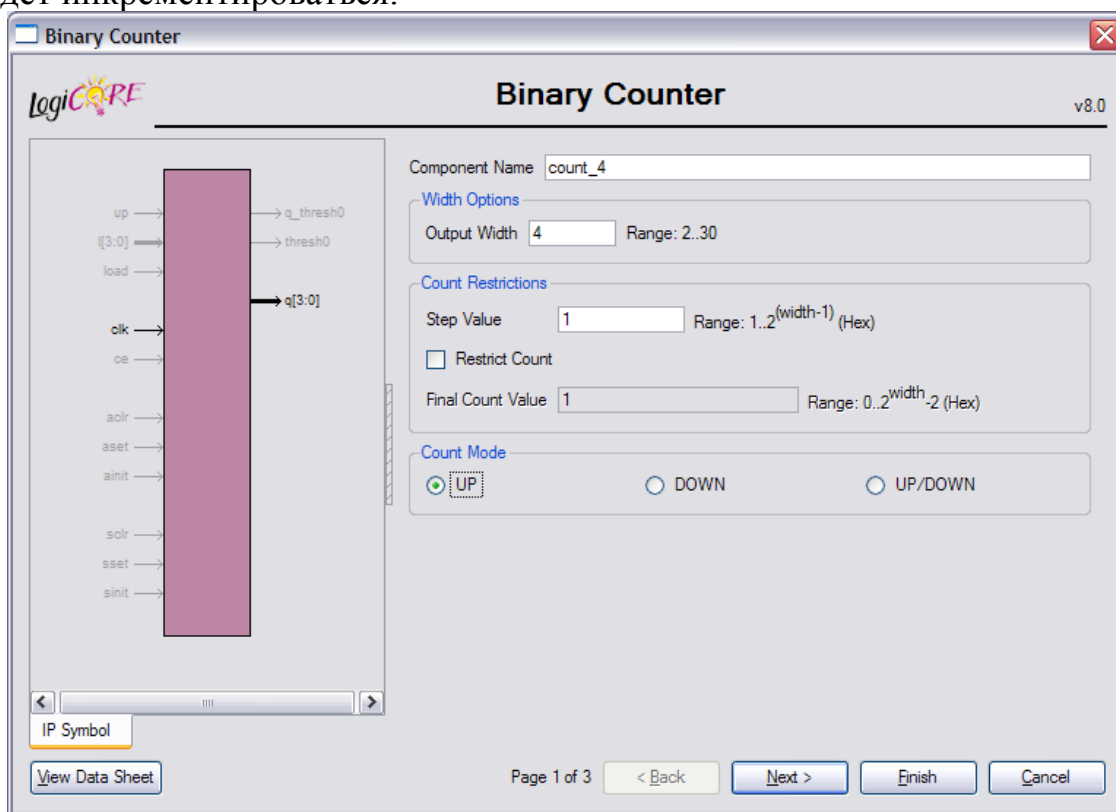


Рисунок 25. Окно выбора настроек IP блока.

Нажмите **Next**. Следующее появившееся окно (рис. 26) содержит дополнительные настройки, касающиеся дополнительных управляющих входов. В рамках данной лабораторной работы Вам требуется создать синхронный цифровой узел, поэтому в группе асинхронных настроек (**Asynchronous Settings**) все флаги должны быть неактивными, начальное инициализируемое значение поставьте равной нулю.

В группу синхронных настроек (**Synchronous Settings**) находятся 3 флага: **Set**, **Clear**, **Init**. Флаг **Set** добавляет выводы к счетчику,

позволяющие установить счетчик в конечное состояние (все сигналы состояния переходят в высокий уровень). Флаг **Clear** добавляет выходы к счетчику, позволяющие сбросить счетчик. Флаг **Init** добавляет выходы к счетчику, позволяющие пользователю в любой момент времени (по синхросигналу) установить заранее зарегистрированное в поле **Synchronous Init Value** состояние счетчика. Сделайте активным флаг **Clear**.

Следующая группа настроек **Clock Enable** позволяет добавить разрешающий сигнал и установить его приоритет по отношению к сигналу сброса (если таковой установлен настройками). Сделайте флаг **CE** активным и выберите приоритет сигнала сброса над разрешающим сигналом.

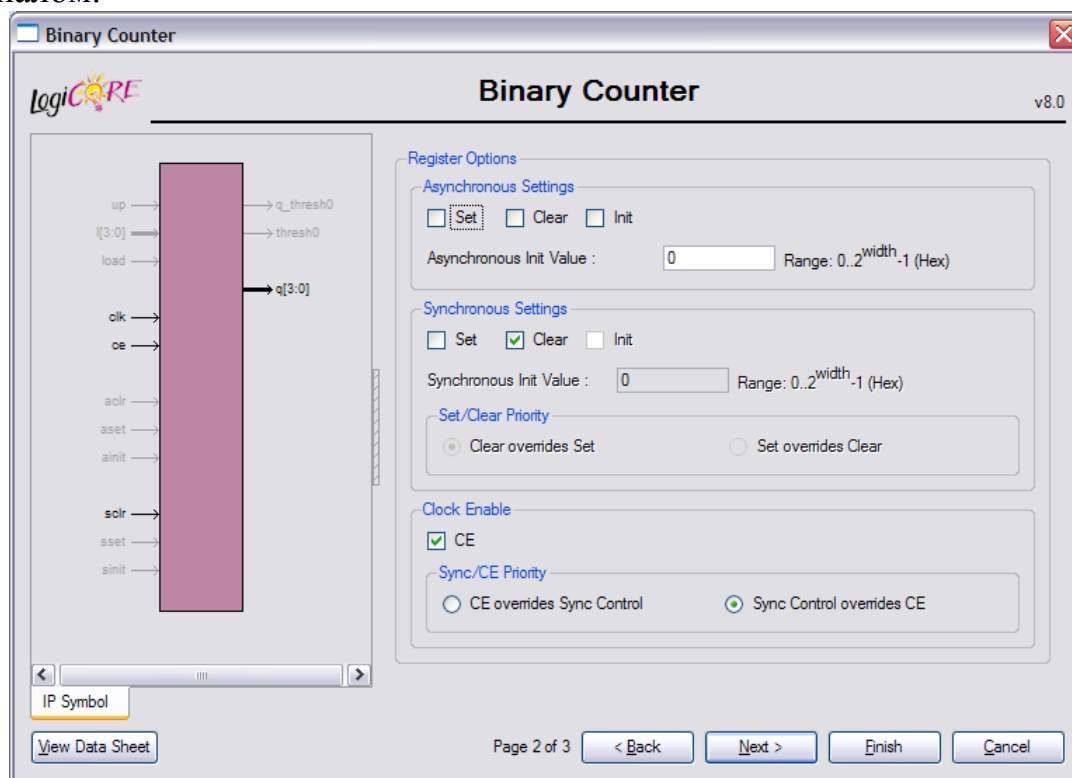


Рисунок 26. Окно выбора дополнительных настроек IP блока.

Нажмите **Next**. Следующее появившееся окно (27) содержит дополнительные настройки, касающиеся загрузки состояния. Они позволяют добавить контакты, позволяющие пользователю задать произвольное состояние счетчика, установить приоритет разрешающего сигнала над загрузкой состояния, а также установить пороговые параметры.

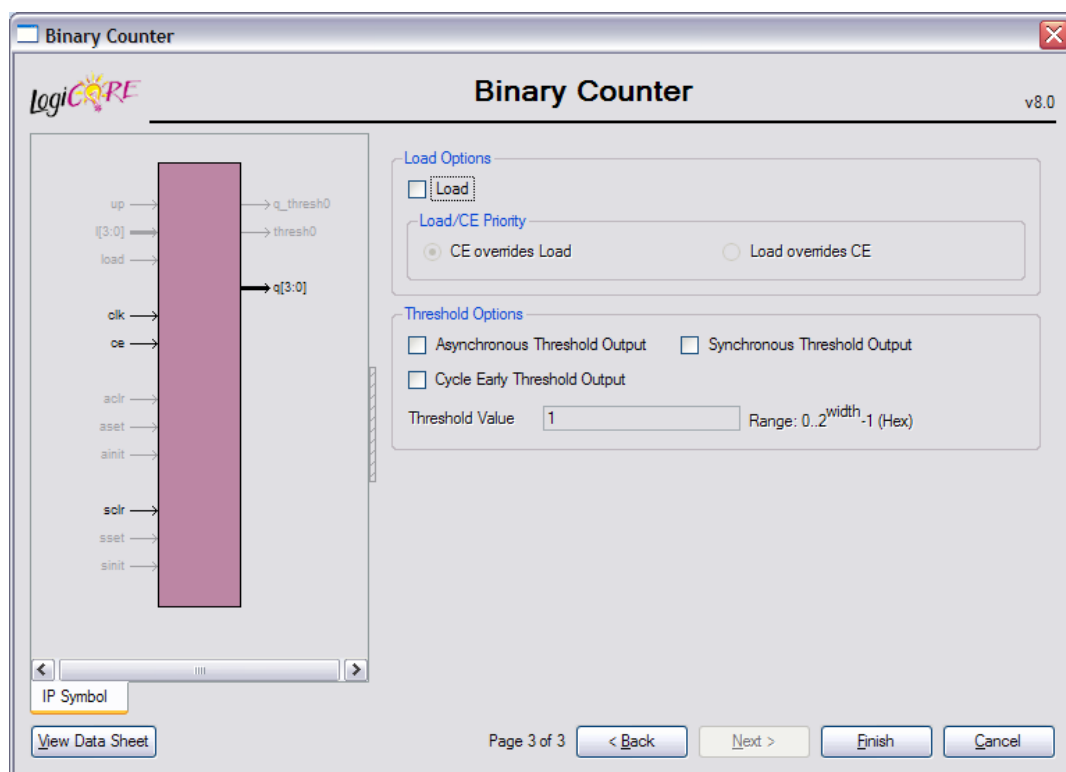


Рисунок 27. Окно выбора настроек загрузки IP блока.

Данные установки в контексте данной лабораторной работы не требуются. Обратите внимание, в левой стороне окна изображен интерфейс IP блока, с имеющимися входными и выходными сигналами. Нажмите **Finish**.

После проделанных действий счетчик в виде IP блока будет создан. Т.к. IP блоки являются интеллектуальной собственностью фирмы производителя, Вы не сможете увидеть его код на языке описания аппаратуры. Для того чтобы изменить настройки IP блока, кликните два раза по нему в поле **Sources** (рис. 28).

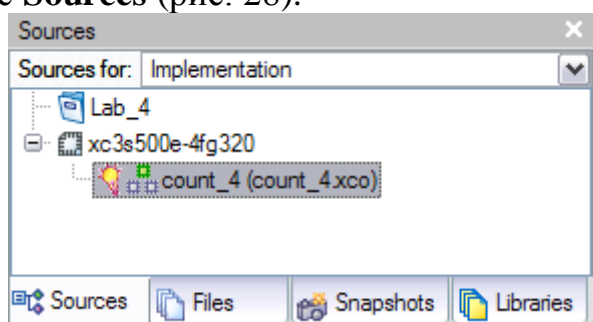


Рисунок 28. Промежуточная иерархия проекта.

Далее создайте VHDL библиотеку, в которую Вы должны будете поместить библиотечный элемент (Package). Для этого щелкните правой кнопкой мыши по полю **Sources** и во всплывшем меню выберите пункт **New Source...**

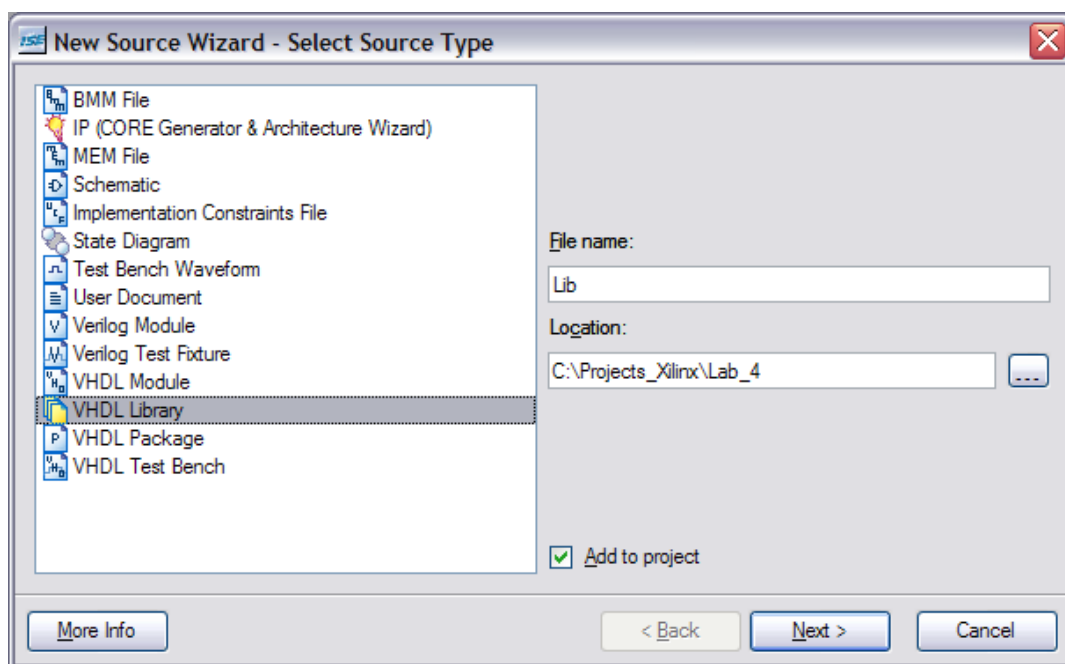


Рисунок 29. Окно создания модуля.

В данном окне выберите тип создаваемого модуля **VHDL Library** (рис. 29). Нажмите **Next**.

Перед вами появится окно отчета о создании VHDL библиотеки (рис. 30).

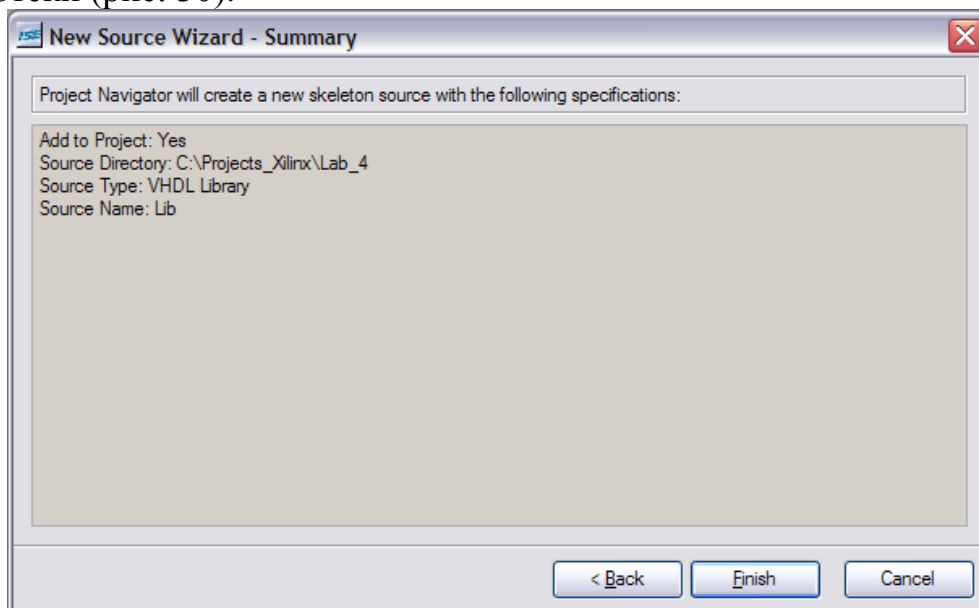


Рисунок 30. Окно отчета о создании VHDL библиотеки.

Нажмите **Finish**. Созданную библиотеку Вы можете наблюдать в поле Sources во вкладке Libraries. Теперь создайте Библиотечный элемент, ставящий в соответствие состоянию счетчика выходную комбинацию.



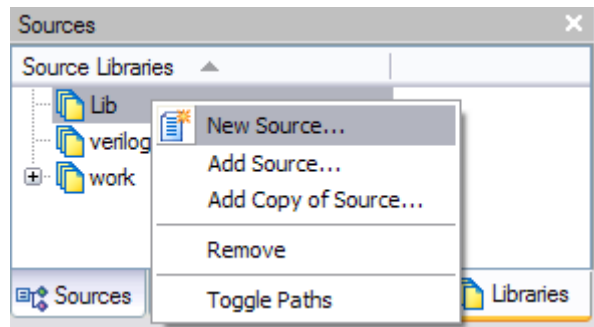


Рисунок 31. Создание библиотечного элемента.

Для этого щелкните правой кнопкой мыши по созданной библиотеке и во всплывшем меню выберите пункт **New Source...** (рис. 31).

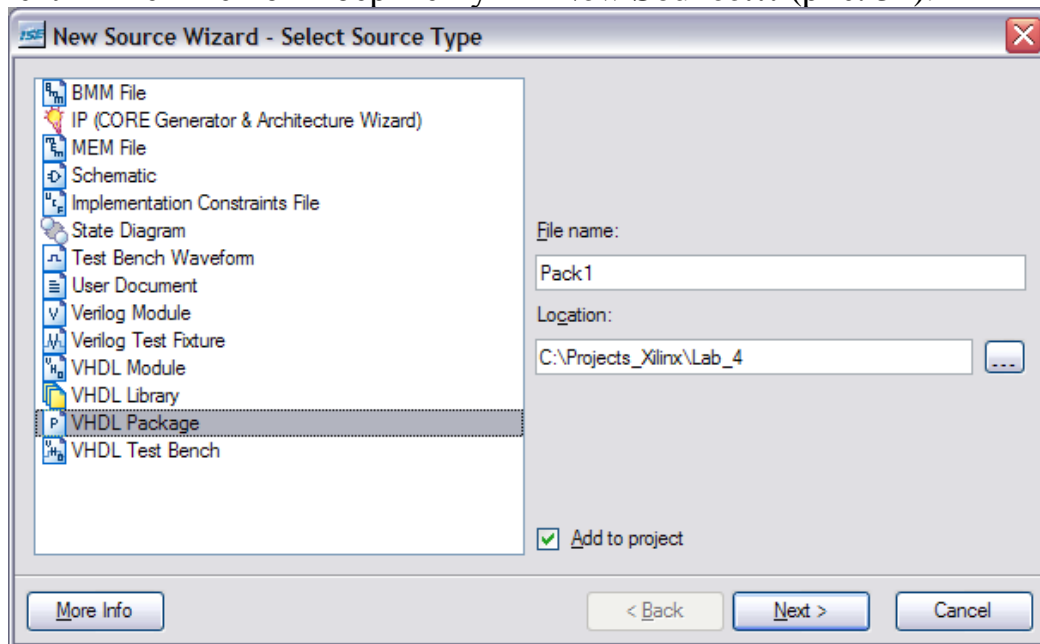


Рисунок 33. Окно создания модуля.

Создаваемый библиотечный элемент рекомендуется назвать Pack1. Выберите тип создаваемого модуля **VHDL Package**. Нажмите **Next**.

Перед вами появится окно отчета о создании библиотечного элемента (рис. 34).

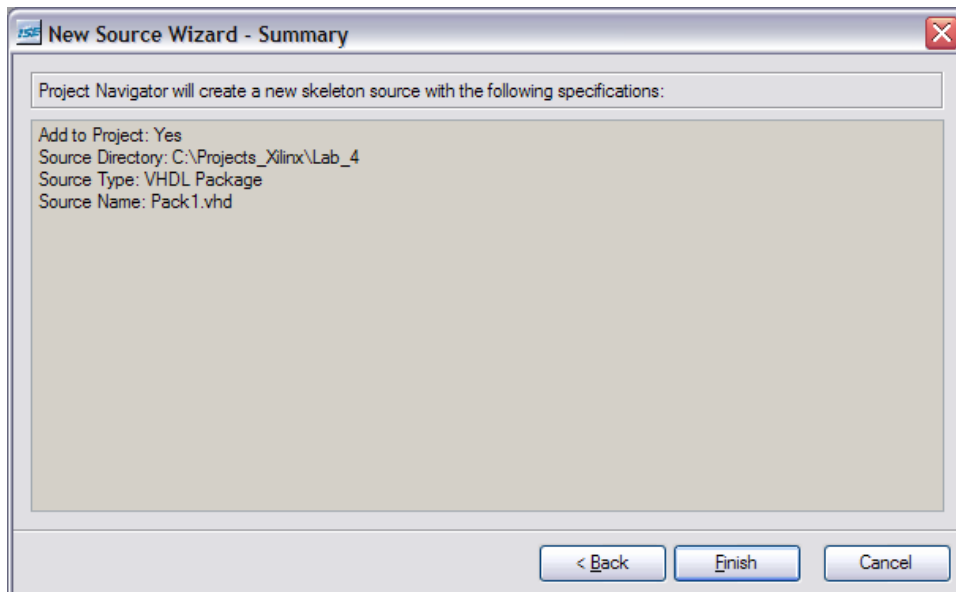


Рисунок 34. Окно отчета о создании библиотечного элемента.

Нажмите **Finish**. Перед Вами откроется текстовый редактор, относящийся к библиотечному элементу. Сразу же стоит обратить внимание, что в библиотечном элементе поддерживается ограниченный набор стандартных библиотек.

На данном этапе Вам требуется создать описание функций, ставящих в соответствие состоянию счетчика выходную комбинацию на языке VHDL. Для этого Вам пригодятся вентиляные описания выходных значений, взятые из лабораторной работы №1. Также Вы можете воспользоваться оператором **case**.

Ниже приведено содержание библиотечного элемента с подробным описанием.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
--подключение стандартных библиотек
package Pack1 is
--операторная часть библиотечного элемента
    function part1 (signal X3,X2,X1,X0 : STD_LOGIC ) return
STD_LOGIC;
    function part2 (signal X3,X2,X1,X0 : STD_LOGIC ) return
STD_LOGIC;
    function part3 (signal X3,X2,X1,X0 : STD_LOGIC ) return
STD_LOGIC;
    function part4 (signal X3,X2,X1,X0 : STD_LOGIC ) return
STD_LOGIC;
--регистрация функций

end Pack1;
--конец операторной части
package body Pack1 is
--исполнительная часть
```

```

function part1 (signal X3,X2,X1,X0 : STD_LOGIC ) return STD_LOGIC
is
--после слова return стоит тип возвращаемой переменной
variable Y0      : STD_LOGIC;
--создание локальной переменной
begin
    Y0 :=(not(X3) and                X1 and      X0 )or
    (not(X3) and      X2 and                X0 )or
    (                X2 and      X1 and not(X0))or
    (      X3 and                X1 and not(X0))or
    (      X3 and not(X2) and not(X1) and      X0 );
    return Y0;
end part1;
--описание функции
function part2(signal X3,X2,X1,X0 : STD_LOGIC) return STD_LOGIC is
variable Y1:STD_LOGIC;
begin
    Y1 :=(not(X3) and                X1 and      X0 )or
    (                X2 and      X1 and      X0 )or
    (                X2 and not(X1) and not(X0))or
    (      X3 and not(X2) and      X1 and not(X0));
return Y1;
end part2;

function part3(signal X3,X2,X1,X0 : STD_LOGIC) return STD_LOGIC is
variable Y2:STD_LOGIC;
begin
Y2 :=(
                X1 and not(X0))or
    (not(X3) and not(X2) and                X0 )or
    (not(X3) and      X2 and                not(X0));
return Y2;
end part3;

function part4(signal X3,X2,X1,X0 : STD_LOGIC) return STD_LOGIC is
variable Y3:STD_LOGIC;
begin
Y3 :=(not(X3) and                X0 )or
    (                X2 and not(X1)                )or
    (      X3 and                not(X1) and not(X0))or
    (                not(X2) and      X1 and      X0 );
return Y3;
end part4;
end Pack1;
--конец исполнительной части

```

Не забывайте периодически сохранять все изменения в проекте (Save all). Обратите внимание на иерархию проекта (см. поле **Sources** вкладка **Sources**). Пока библиотечный элемент не был использован, он условно закрепляется за старшим в иерархии модулем. После непосредственного

подключения этого библиотечного элемента в модуль, его не будет видно в данной иерархии (рис. 35).

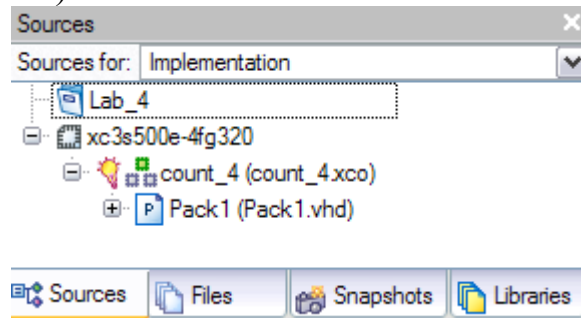


Рисунок 35. Промежуточная иерархия проекта.

Далее Вам необходимо создать VHDL модуль, который должен содержать подключение созданного библиотечного элемента, а также счетчика, реализованного в виде IP блока.

Для того чтобы подключить созданный библиотечный элемент, необходимо сначала подключить соответствующую библиотеку, а затем и сам библиотечный элемент. В демонстрационном случае это будет выглядеть следующим образом:

```
library IEEE;
library Lib;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use Lib.Pack.ALL;
--подключение библиотек
```

Создайте интерфейс устройства в соответствии с заданием на лабораторную работу.

Создайте отдельный сигнал (в качестве регистра), куда будут считываться состояния счетчика:

```
signal X: STD_LOGIC_VECTOR(3 downto 0);
```

Свяжите контакты основного VHDL модуля с контактами счетчика (как это делается – см. лабораторную работу №1). Выходное значение счетчика свяжите с отдельным сигналом накопления состояний счетчика (в демонстрационном примере – с сигналом X).

Создайте процесс, чувствительный к синхросигналу, в котором будут вызываться функции из библиотечного элемента.

```
Y(0)<=part1(X(3),X(2),X(1),X(0));
Y(1)<=part2(X(3),X(2),X(1),X(0));
Y(2)<=part3(X(3),X(2),X(1),X(0));
Y(3)<=part4(X(3),X(2),X(1),X(0));
```

По завершении выполнения всех предписанных действий, Вам необходимо верифицировать проект. Для этого создайте исходный модуль типа Test Bench Waveform. Создание данного модуля описано в разделе «Функциональная верификация». Вам необходимо манипулировать разрешающим сигналом и сигналом сброса. Для корректного моделирования работы устройства, установите ненулевое время инициализации, т.к. использование библиотечного элемента требует определенной предустановки. В ходе выполнения данной лабораторной работы у Вас в итоге должна получиться следующая временная диаграмма (рис. 36).

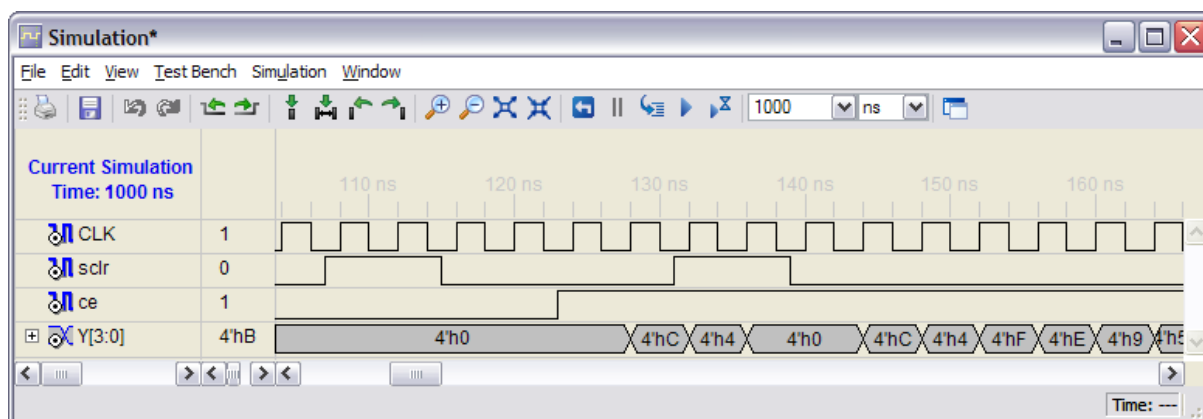


Рисунок 36. Временная диаграмма работы устройства.

## Лабораторная работа № 5

До сих пор для верификации в рамках лабораторных работ была использована верификация с помощью программного пакета Test Bench. Данный пакет позволяет наглядно в графическом режиме менять подаваемые сигналы с течением времени посредством мышки. В более поздних моделях САПР Xilinx ISE от этого отказались. Верификация в них происходит посредством отдельного файла верификации, описанного на языках описания аппаратуры. Этот метод имеет ряд преимуществ, по сравнению с методом, использующим Test Bench:

- большая точность определения временных параметров;
- аналитический характер разработки тестов;
- большая скорость разработки тестов (для больших и средних проектов).

Поэтому в 4 лабораторной работе Вам предлагается создать проект с устройством и сделать для него текстовый файл для верификации. В качестве устройства требуется построить синхронный комбинационный логический узел, имеющий 4 входа X (3:0) и 4 выхода Y (3:0). Каждой входной комбинации ставится в соответствие некоторая выходная комбинация. Задание выдаётся в виде таблицы, в верхней строке которой перечисляются в порядке убывания входные комбинации от «F» (1111) до

«0» (0000), а в нижней строке приведены соответствующие им выходные комбинации (табл. 1).

Интерфейс разрабатываемого устройства будет иметь следующий вид (рис. 37):

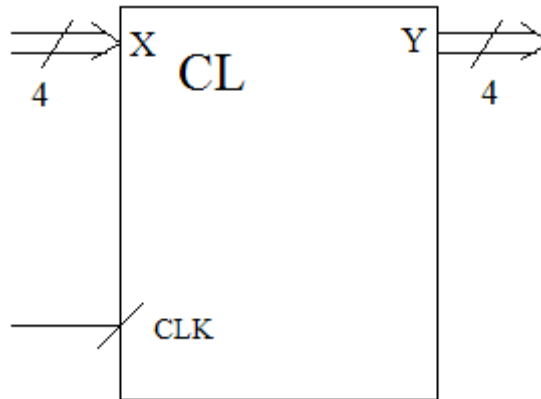


Рисунок 37. Интерфейс устройства.

Создайте устройство, описанное выше на любом удобном для Вас языке описания аппаратуры.

Создайте тестовый VHDL модуль. Для этого щелкните правой кнопкой мыши по полю **Sources** и во всплывшем меню выберите пункт **New Source...**

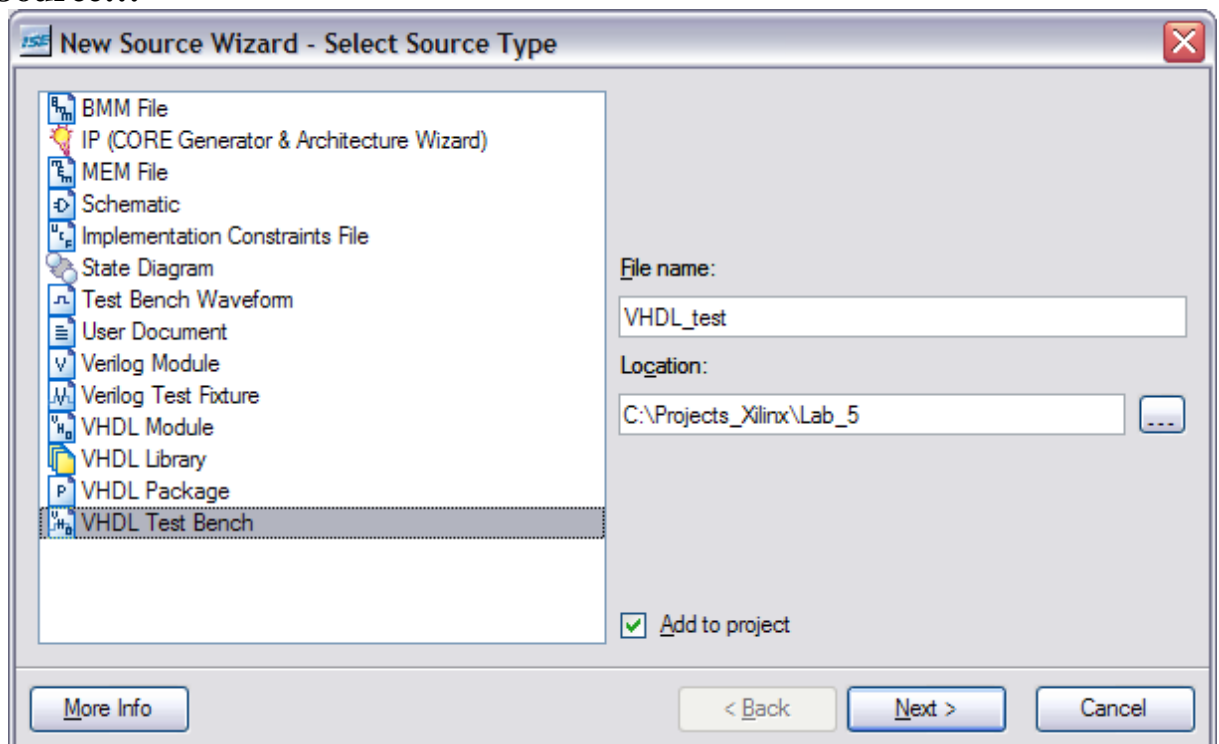


Рисунок 38. Окно создания модуля.

Этот модуль рекомендуется назвать VHDL\_test. В списке предлагаемых типов модулей выберите **VHDL Test Bench** (рис. 38).

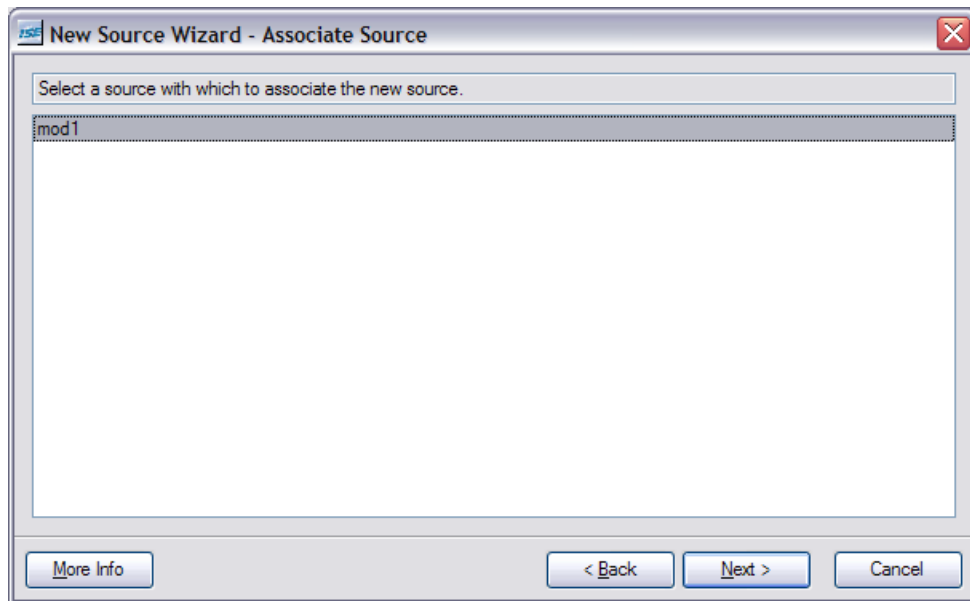


Рисунок 39. Окно прикрепления модуля.

Прикрепите его к созданному модулю с описанием устройства (рис. 39).

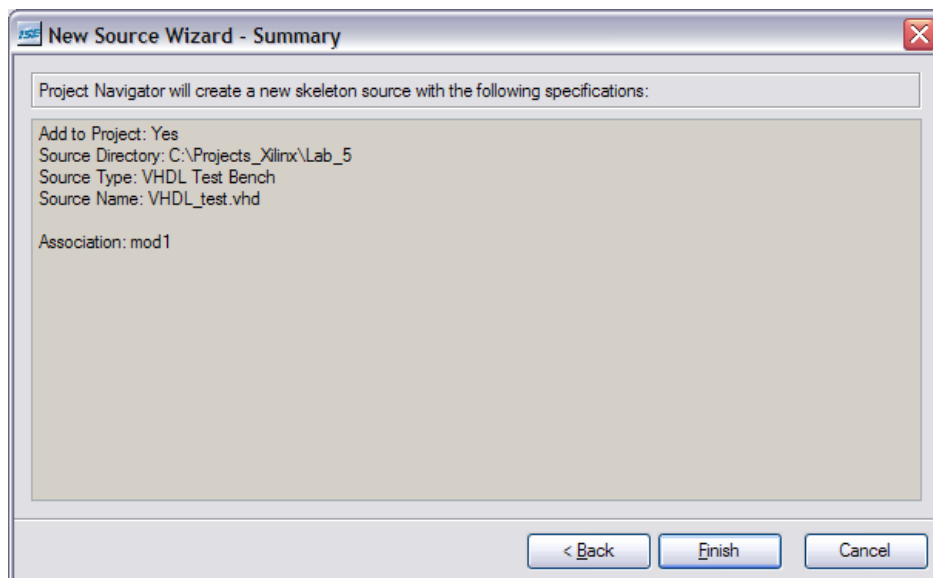


Рисунок 40. Окно отчета создания тестового модуля.

После создания тестового модуля перед Вами появится заготовка:

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;
-- подключение библиотек
ENTITY VHDL_test IS
END VHDL_test;
-- т.к. модуль является тестовым, он не имеет интерфейса
ARCHITECTURE behavior OF VHDL_test IS
    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT mod1

```

```

PORT(
    X : IN  std_logic_vector(3 downto 0);
    Y : OUT std_logic_vector(3 downto 0);
    CLK : IN  std_logic
);
END COMPONENT;
--Inputs
signal X : std_logic_vector(3 downto 0) := (others => '0');
signal CLK : std_logic := '0';
--Outputs
signal Y : std_logic_vector(3 downto 0);
--создание локальных сигналов/регистров
BEGIN
    -- Instantiate the Unit Under Test (UUT)
    uut: mod1 PORT MAP (
        X => X,
        Y => Y,
        CLK => CLK
    );
    -- Связывание локальных сигналов и сигналов тестируемого модуля
    -- No clocks detected in port list. Replace <clock> below with
    -- appropriate port name
    constant <clock>_period := 1ns;

    -- процесс генератора синхросигнала
    <clock>_process :process
    --т.к. в модуле из-за верхнего регистра (CLK вместо clk) не был
    --распознан синхросигнал, здесь предлагается вписать его вместо
    --надписи <clock>
    begin
        <clock> <= '0';
        wait for <clock>_period/2;
        <clock> <= '1';
        wait for <clock>_period/2;
    end process;
    -- Stimulus process
    stim_proc: process
    begin
        -- hold reset state for 100ms.
        wait for 100ms;
        --время для инициализации
        wait for <clock>_period*10;
        --время общего моделирования (период синхросигнала*10)
        wait;
    end process;
END;

```

Строку с константой необходимо переместить до начала описания архитектуры (после объявления сигнала Y). <clock> необходимо заменить сигналом синхронизации (CLK). Как видно из синтаксиса, для задания



входных значений сигнала необходимо создавать процесс (по одному процессу на группу параллельно изменяющихся подаваемых значений).

Создайте отдельный процесс для подачи сигнала X. Задайте период такта синхросигнала равной 10 наносекунд. Подавайте значение X за 5 наносекунд до восходящего фронта синхросигнала (каждое следующее значение X образуется с помощью его инкремента). При запуске симуляции у Вас должна получиться следующая диаграмма (рис. 41):

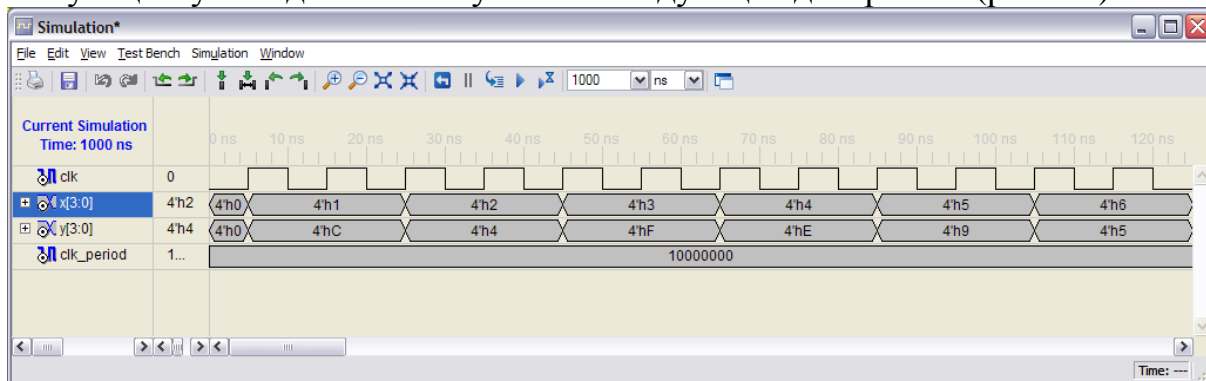


Рисунок 41. Временная диаграмма работы устройства.

Создайте тестовый Verilog модуль. Для этого щелкните правой кнопкой мыши по полю **Sources** и во всплывшем меню выберите пункт **New Source...**

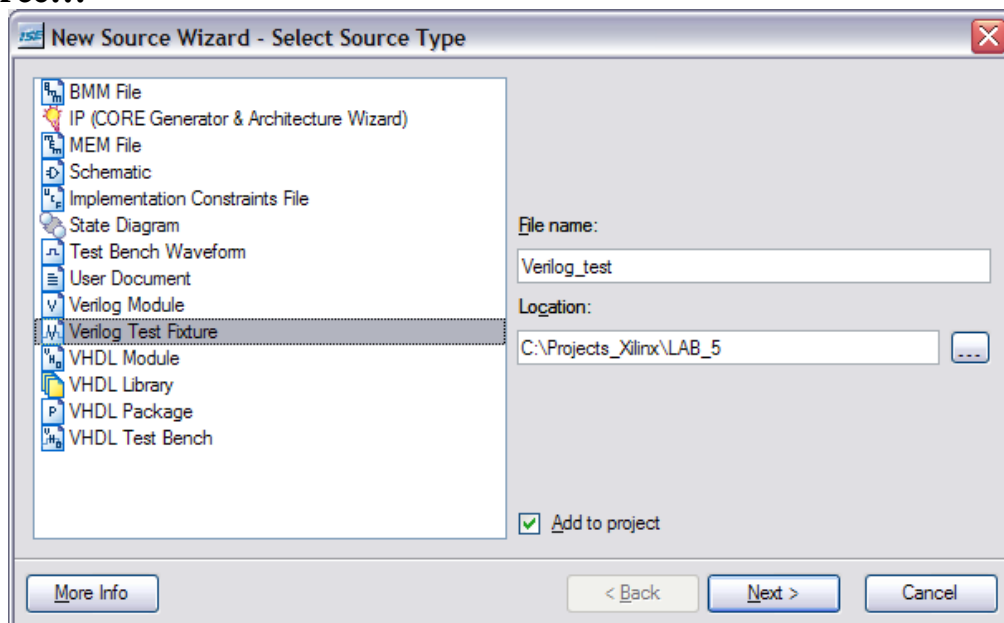


Рисунок 42. Окно создания модуля.

Этот модуль рекомендуется назвать Verilog\_test. В списке предлагаемых типов модулей выберите **Verilog Test Fixture** (рис. 42).

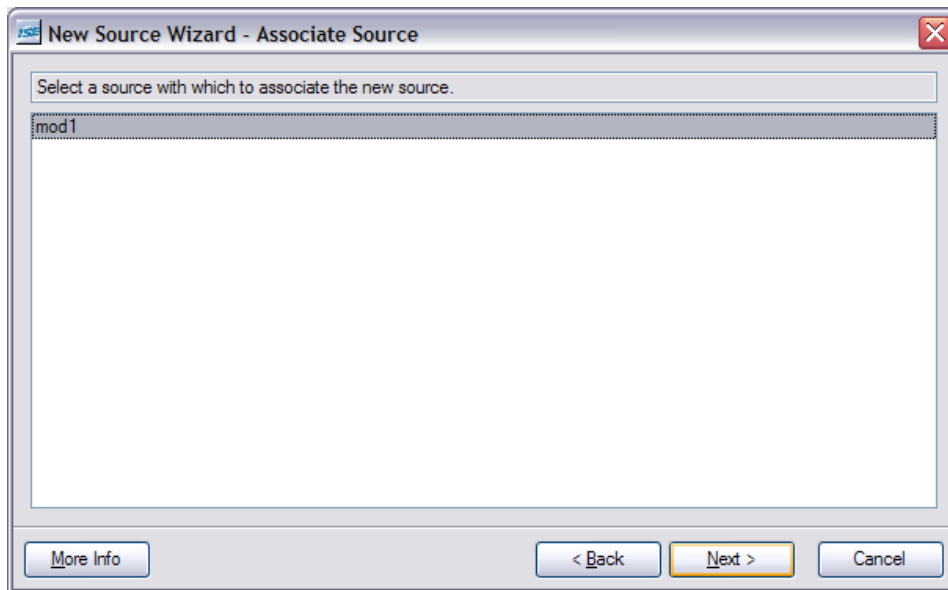


Рисунок 43. Окно прикрепления модуля.

Прикрепите его к созданному модулю с описанием устройства (рис. 43).

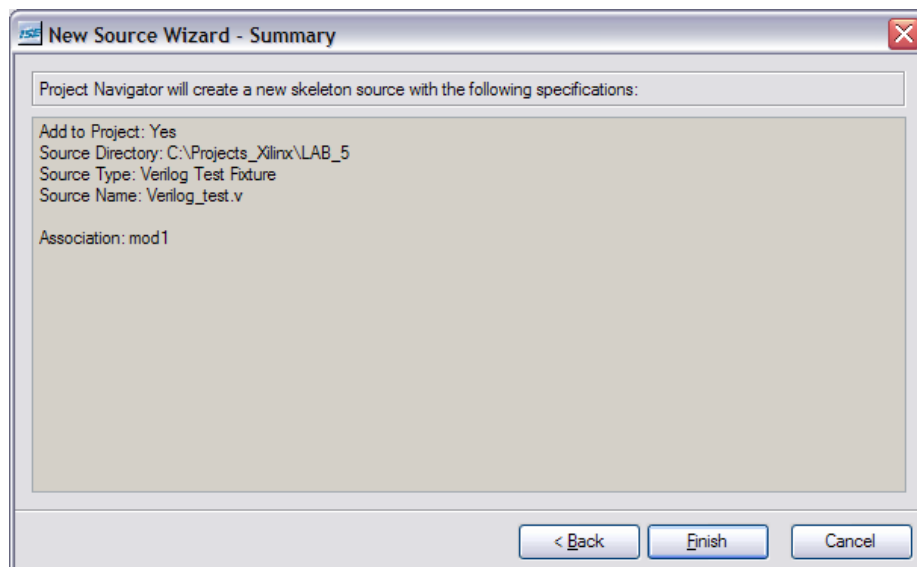


Рисунок 43. Окно отчета создания тестового модуля.

После создания тестового модуля перед Вами появится заготовка:

```
`timescale 1ns / 1ps
module Verilog_test;

    // Inputs
    reg [3:0] X;
    reg CLK;

    // Outputs
    wire [3:0] Y;
    // Создание локальных сигналов
    // Instantiate the Unit Under Test (UUT)
    mod1 uut (
```

```

        .X(X),
        .Y(Y),
        .CLK(CLK)
    );
    // Связывание локальных сигналов и сигналов тестируемого модуля
    initial begin
        // Initialize Inputs
        X = 0;
        CLK = 0;
        //Задание начальных значений
        // Wait 100 ns for global reset to finish
        #100;

        // Add stimulus here

    end

endmodule

```

Для того чтобы задать поступление входных сигналов требуется создать отдельный процесс (по одному процессу на группу параллельно изменяющихся подаваемых значений). Задержки (с помощью оператора «#») измеряются в единицах, указанных вначале после **`timescale** (исходно – #1 эквивалентно одной наносекунде).

Вот пример создания процесса для синхросигнала:

```
always #10 clk=~clk;
```

Вставьте этот процесс в тестовый модуль. Аналогичным образом создайте процесс, подающий значение X за 5 наносекунд до восходящего фронта синхросигнала (каждое следующее значение X образуется с помощью его инкремента). Запустите симуляцию.

В итоге временные диаграммы у Вас должны совпасть.

## Практические работы

В данном разделе подробно описывается исполнения лабораторных работ на отладочной плате Spartan-3E Starter Kit.

### Практическая работа № 1

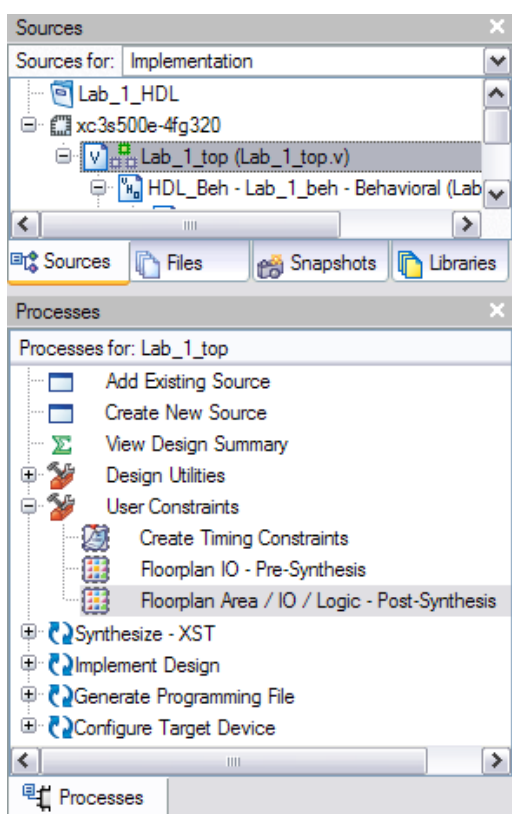
Перед началом конфигурирования ПЛИС на отладочной плате, следует ее сначала установить. Для этого подключите отладочную плату посредством интерфейса USB к вашему компьютеру; подайте питание на

плату, подключив ее к розетке через специальный адаптер. Включите плату с помощью ползункового переключателя (расположенного около порта питания), переключив его в положение ON.



*Оберегать плату от механических повреждений, от влаги, от влияния чрезмерных температур, не класть на нее металлические предметы.*

После проделанной процедуры, операционная система выдаст сообщение «Найдено новое оборудование» и предложит Вам установить его. Установите его, используя автоматическую установку.



Теперь можно приступать к конфигурированию ПЛИС. Для этого запустите проект с первой лабораторной работой на Xilinx WebPACK ISE. Убедитесь в том, что в качестве выбранной проектируемой ПЛИС стоит xc3s500e-4fg320. В окне исходных модулей установите закладку **Sources**, в выпадающем меню выберите **Sources for: → Implementation**, кликните левой кнопкой мыши по топовому файлу. В окне **Processes** выберите инструмент **User Constrains → Floorplan Area/ IO/ Logic Post-Synthesis** и запустите его посредством двойного клика мыши (рис. 1).

Рисунок 1.

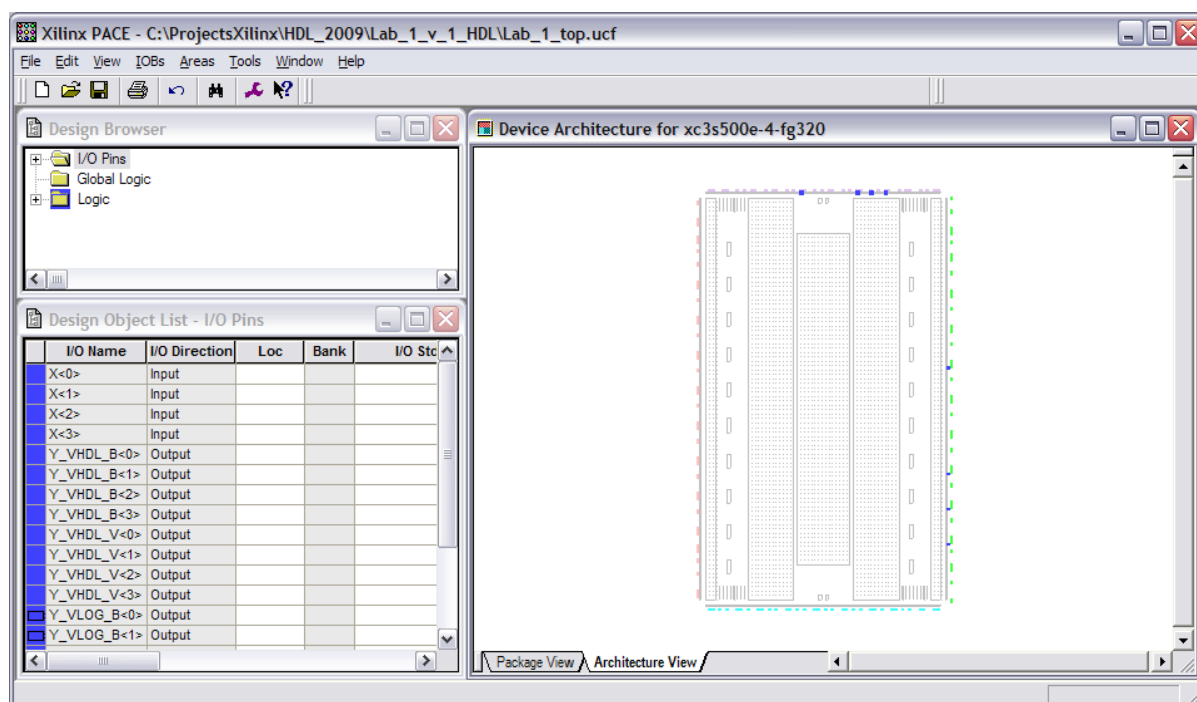


Рисунок 2. Окно плана размещения контактов на плате

С помощью этой программы (рис. 2) вам предлагается задать соответствие входных и выходных сигналов с элементами платы. Для этого разверните окно **Design Object List – I/O Pins** (рис. 3). В позиции Loc (сокращение от Location – «местоположение») укажите соответствие входным значениям ползунковые переключатели, выходным сигналам – светодиоды. Т.к. светодиодов на плате 8 шт, а выходных сигналов 16 шт, Вам предстоит проделать весь перечень процедур конфигурации ПЛИС дважды, чтобы проверить правильность срабатывания.

The screenshot shows the Xilinx PACE software interface with the 'Design Object List - I/O Pins' window open. The table displays detailed configuration for each I/O pin, including location, bank, standard, termination, slew, delay, and clock settings.

I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name	Local Clock
X<0>	Input	h13	BANK1					PULLUP			Unknown		<input type="checkbox"/>
X<1>	Input	h14	BANK1					PULLUP			Unknown		<input type="checkbox"/>
X<2>	Input	h18	BANK1					PULLUP			Unknown		<input type="checkbox"/>
X<3>	Input	n17	BANK1					PULLUP			Unknown		<input type="checkbox"/>
Y_VHDL_B<0>	Output	f12	BANK0								Unknown		<input type="checkbox"/>
Y_VHDL_B<1>	Output	e12	BANK0								Unknown		<input type="checkbox"/>
Y_VHDL_B<2>	Output	e11	BANK0								Unknown		<input type="checkbox"/>
Y_VHDL_B<3>	Output	f11	BANK0								Unknown		<input type="checkbox"/>
Y_VHDL_V<0>	Output	c11	BANK0								Unknown		<input type="checkbox"/>
Y_VHDL_V<1>	Output	d11	BANK0								Unknown		<input type="checkbox"/>
Y_VHDL_V<2>	Output	e9	BANK0								Unknown		<input type="checkbox"/>
Y_VHDL_V<3>	Output	f9	BANK0								Unknown		<input type="checkbox"/>
Y_VLOG_B<0>	Output										Unknown		<input type="checkbox"/>
Y_VLOG_B<1>	Output										Unknown		<input type="checkbox"/>
Y_VLOG_B<2>	Output										Unknown		<input type="checkbox"/>
Y_VLOG_B<3>	Output										Unknown		<input type="checkbox"/>
Y_VLOG_V<0>	Output										Unknown		<input type="checkbox"/>
Y_VLOG_V<1>	Output										Unknown		<input type="checkbox"/>
Y_VLOG_V<2>	Output										Unknown		<input type="checkbox"/>
Y_VLOG_V<3>	Output										Unknown		<input type="checkbox"/>

Рисунок 3. Список контактов устройства.

В позиции Termination («срабатывание») укажите положение ползунковых переключателей, соответствующих логической единице (PULLUP или PULLDOWN).

**!ВАЖНО!** Название элементов можно посмотреть на самой плате. Они подписываются под каждым элементом и заключены в скобки: <Название\_элемента>

Закройте приложение. Программа спросит вас, сохранять ли изменения в файле Lab\_1\_top.ucf – нажмите да. Вы можете открыть этот файл с помощью текстового редактора (например - Блокнота) и увидеть его содержимое:

#PACE: Start of Constraints generated by PACE

#PACE: Start of PACE I/O Pin Assignments

```
NET "X<0>" LOC = "113" | PULLUP ;  
NET "X<1>" LOC = "114" | PULLUP ;  
NET "X<2>" LOC = "h18" | PULLUP ;  
NET "X<3>" LOC = "n17" | PULLUP ;  
NET "Y_VHDL_B<0>" LOC = "f12" ;  
NET "Y_VHDL_B<1>" LOC = "e12" ;  
NET "Y_VHDL_B<2>" LOC = "e11" ;  
NET "Y_VHDL_B<3>" LOC = "f11" ;  
NET "Y_VHDL_V<0>" LOC = "c11" ;  
NET "Y_VHDL_V<1>" LOC = "d11" ;  
NET "Y_VHDL_V<2>" LOC = "e9" ;  
NET "Y_VHDL_V<3>" LOC = "f9" ;
```

#PACE: Start of PACE Area Constraints


#PACE: Start of PACE Prohibit Constraints

#PACE: End of Constraints generated by PACE

Впоследствии, для быстрой корректировки вы можете вносить изменения в сам файл, не запуская приложение Xilinx PACE.

Выберите в окне **Processes** Инструмент **Manage Configuration Project (iMPACT)** и запустите его посредством двойного клика мышью (рис. 4).



*Если на каком-то этапе напротив одного из инструментов вы увидите значок «красный кружочек с крестиком» , значит Вы допустили ошибку в проекте. Соответственно для того, чтобы сконфигурировать ПЛИС, вам необходимо устранить ошибки.*

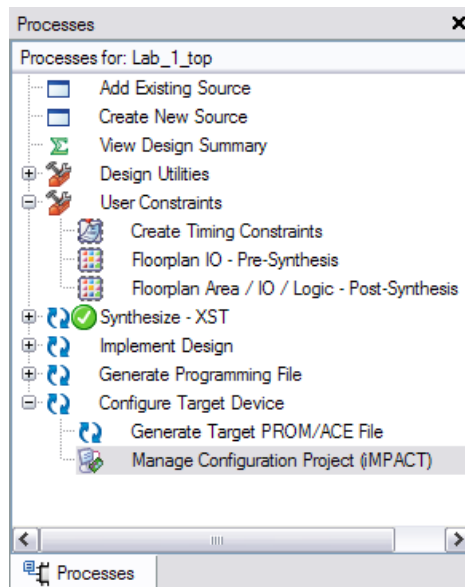


Рисунок 4. Окно процессов.

Перед вами появится окно (рис. 5):

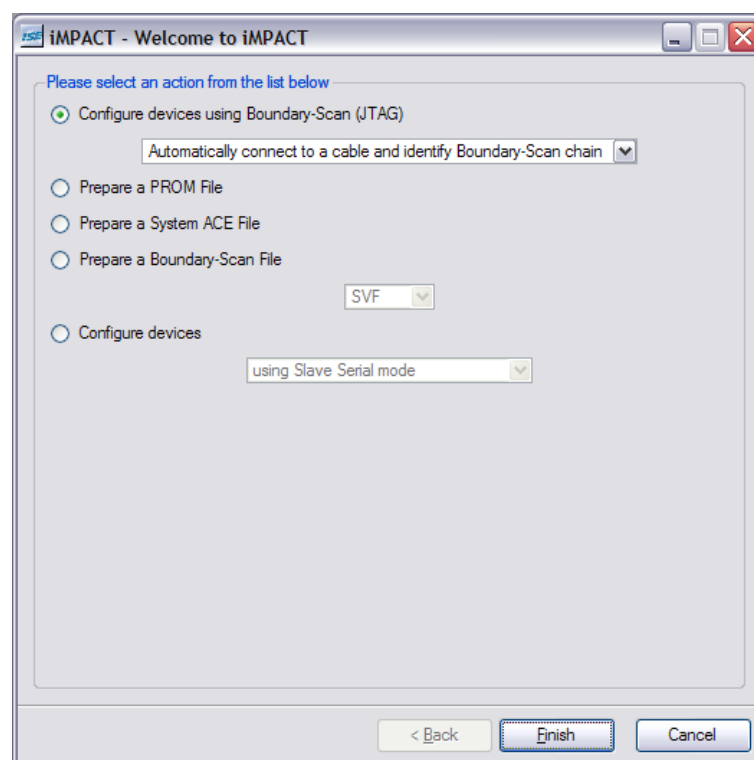


Рисунок 5. Окно программирования платы.

Нажмите **Finish**. Далее перед вами появится окно (рис. 6), где вам предлагается выбрать файл для конфигурирования FPGA. Выберите его (он должен находиться в папке с проектом и скорее всего будет единственным доступным).

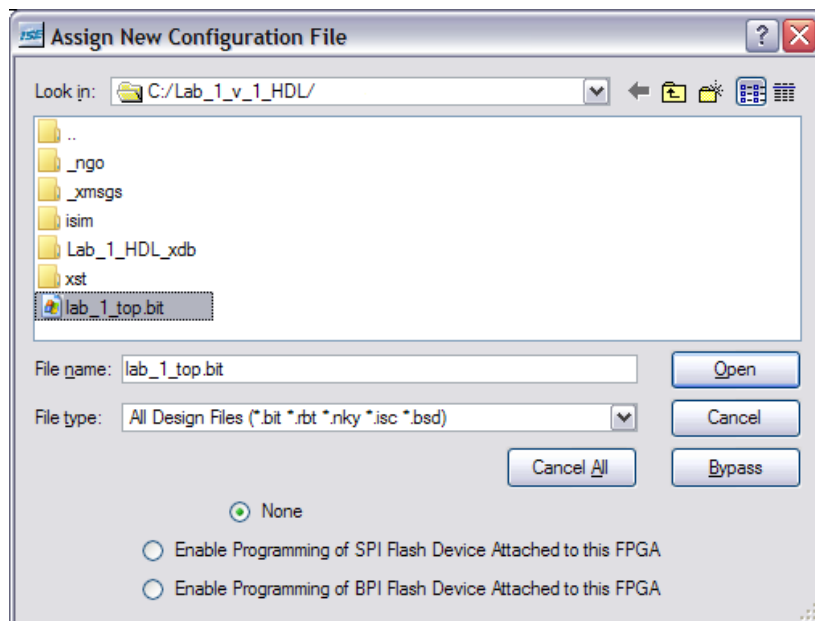


Рисунок 6. Окно подключения конфигурационного файла.

Нажмите **Open**. Далее перед вами появится окно, где вам предлагается выбрать файл для конфигурирования PROM (рис. 7).

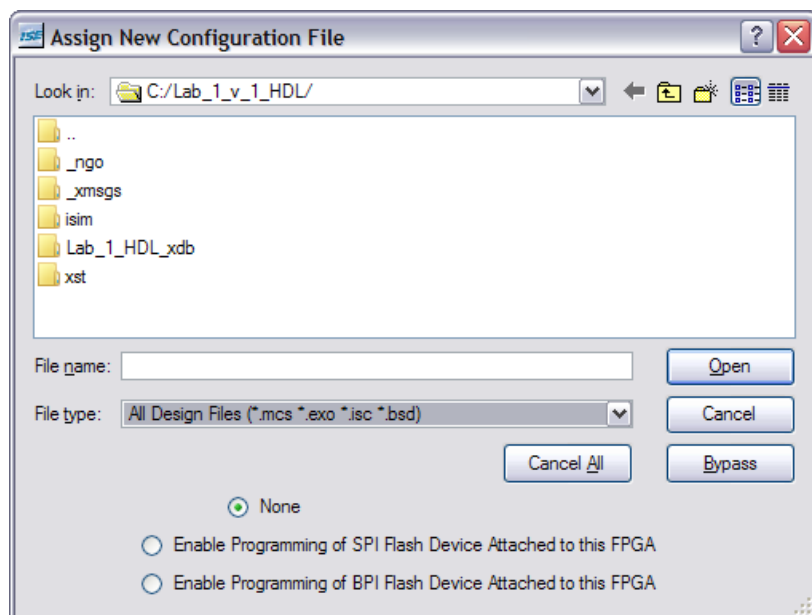


Рисунок 7. Окно подключения конфигурационного файла.

Нажмите **Bypass**. Далее перед вами появится окно, где вам предлагается выбрать файл для конфигурирования CPLD (рис. 8).



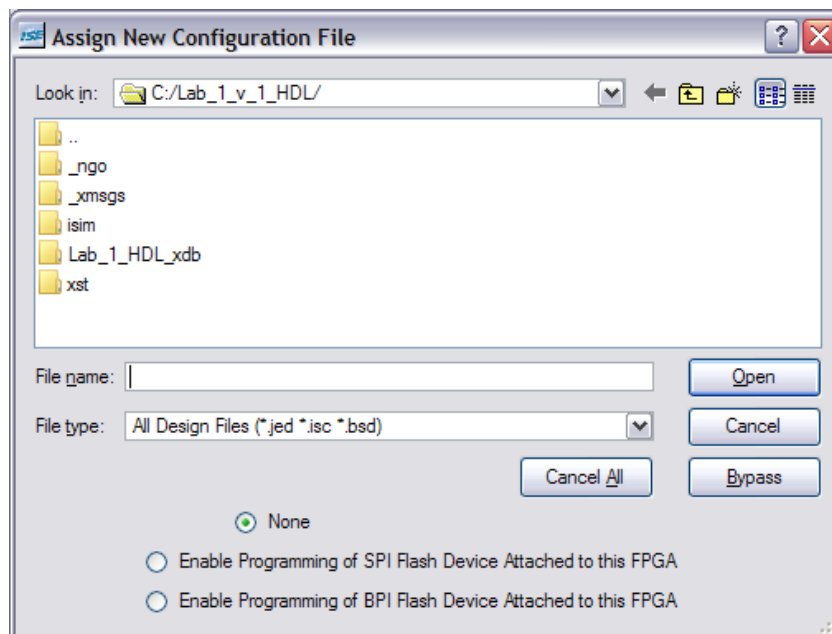


Рисунок 8. Окно подключения конфигурационного файла.  
Снова нажмите **Bypass**. Далее перед вами появится окно, где вам предлагается выбрать опции для конфигурирования (рис 9).

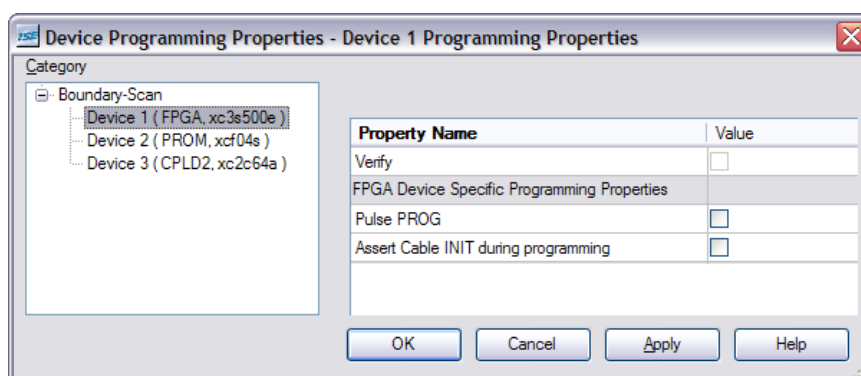


Рисунок 9. Окно программируемых настроек.  
Нажмите **ОК**. В панели размещения редакторов вы увидите следующую картину (рис 10):

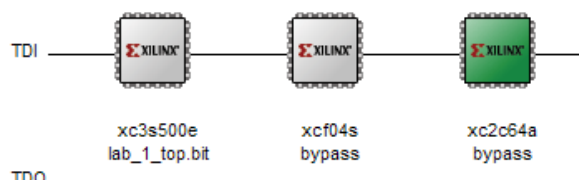


Рисунок 10. Интерфейс платы.

Из нее видно, что на плате располагается не только ПЛИС FPGA xc3s500e-fg320, но и другие платы (PROM и CPLD). Кликните правой кнопкой мыши по ПЛИС xc3s500e-fg320 и в выпадающем меню выберите опцию **Programm** (рис 11).

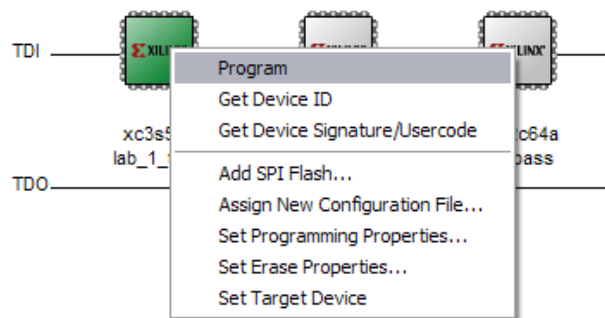


Рисунок 11. Программирование платы.

Если конфигурирование платы прошло успешно, в панели размещения редакторов появится надпись **Program Succeeded**. Теперь, посредством ползунковых переключателей вы можете проверить корректность написания проекта. При переключении ползунковых переключателей должны загораться соответствующие выходным значениям светодиоды. Для сброса конфигурации используйте кнопку **PROG** на плате.

## Практическая работа № 2

В практической работе № 2 требуется реализовать устройство, описанное в лабораторной работе № 2 на отладочной плате S3ESK. В качестве синхросигнала требуется указать генератор частоты на плате с названием «с9». Также требуется использовать светодиодные индикаторы для вывода выходной последовательности. Т.к. генератор синхросигнала имеет тактовую частоту 50МГц, вам следует дополнить проект поворотным регулятором, регулирующим тактовую частоту. В противном случае светодиоды будут переключаться слишком быстро, незаметно для человеческого глаза.

Для того чтобы использовать поворотный регулятор необходимо в проекте описать для него контроллер, который будет делить тактовую частоту. Описанный контроллер есть в разделе 2.1.2 «Описание используемых элементов».

В качестве примера использования поворотного регулятора приводится пример устройства, который выводит состояния счетчика. Проект имеет следующую иерархическую структуру (рис. 12):

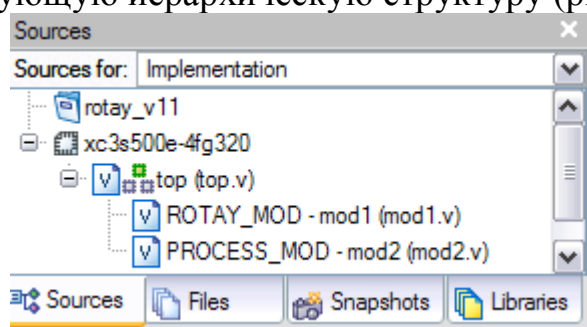


Рисунок 12. Иерархия проекта.

Модуль mod2 представляет собой синхронный прямой счетчик с сигналом сброса, mod1 – контроллер поворотного регулятора, top – модуль, где описаны соединения контактов модулей mod1 и mod2.

Содержимое модуля mod2:

```
`timescale 1ns / 1ps
module mod2(
    input clk_rot,
    output reg [7:0] A,
    input rst);
always @ (posedge clk_rot)
begin
if (rst) A<=0;
else A<=A+1;
end
endmodule
```

Содержимое модуля mod1:

```
`timescale 1ns / 1ps
module mod1
#(parameter DATA_SIZE = 26 )
(
    input clk,
    input r_a,
    input r_b,
    input res,
    output clk_rot
);
parameter SUM = 18'hFFFF3;
reg [DATA_SIZE-1:0]count_i;
reg [DATA_SIZE-1:0]counter;
wire sync_rot_b;
wire sync_rot_a;
reg event_rot_l;
reg event_rot_r;
reg rotary_q1;
reg rotary_q2;
reg rotary_q1_dly;
reg rotary_q2_dly;
wire tmp_a;
reg tr_a1;
reg tr_a2;
wire tmp_b;
reg tr_b1;
reg tr_b2;
reg clk_state ;
```

```

assign clk_rot = clk_state;
assign sync_rot_a = tr_a2;
assign tmp_a      = tr_a1;
assign sync_rot_b = tr_b2;
assign tmp_b      = tr_b1;
always @(posedge clk)
    begin //(1)
        // Синхронизация сигналов от поворотного переключателя
        tr_a1 <= r_a;
        tr_a2 <= tmp_a;
        tr_b1 <= r_b;
        tr_b2 <= tmp_b;

        //////////////////////////////////////
        // Вычисления состояний
        case ({sync_rot_b, sync_rot_a})
            0: rotary_q1 <= 1'b0;
            1: rotary_q2 <= 1'b0;
            2: rotary_q2 <= 1'b1;
            3: rotary_q1 <= 1'b1;
        endcase
        rotary_q1_dly <= rotary_q1;
        rotary_q2_dly <= rotary_q2;
        event_rot_l <= rotary_q2_dly && !rotary_q1_dly &&
rotary_q1;
        event_rot_r <= !rotary_q2_dly && !rotary_q1_dly &&
rotary_q1;
        if (res)
            begin
                count_i <= {DATA_SIZE{1'b0}};
                counter <= {DATA_SIZE{1'b0}};
            end
        else
            case ({event_rot_r, event_rot_l})
                3'b00 : count_i <= count_i;
                3'b01 : count_i <= count_i + SUM;
// Поворот влево
                3'b10 : count_i <= count_i - SUM;
// Поворот вправо
                3'b11 : count_i <= count_i;
                default: count_i <= count_i;
            endcase
        counter<=counter+1;
        if(count_i<=counter)
            begin
                counter <= {DATA_SIZE{1'b0}};
                clk_state<=clk_state+1;
            end
    end //(1)

```

```
endmodule
```

Содержимое файла top:

```
`timescale 1ns / 1ps
module top(
    input clk,
    input r_a,
    input r_b,
    input res,
    output [7:0] A,
    input rst
);
wire clk_rot;
mod1 ROTAY_MOD (
    //прикрепление из модуля mod1 сигналов
    .clk(clk),
    .r_a(r_a),
    .r_b(r_b),
    .res(res),
    .clk_rot(clk_rot));
mod2 PROCESS_MOD (
    //прикрепление из модуля mod2 сигналов
    .A(A),
    .rst(rst),
    .clk_rot(clk_rot));
endmodule
```

План размещения контактов на отладочной плате (рис. 13):

I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name	Local Clock
A<0>	Output	F12	BANK0								Unknown		<input type="checkbox"/>
A<1>	Output	E12	BANK0								Unknown		<input type="checkbox"/>
A<2>	Output	E11	BANK0								Unknown		<input type="checkbox"/>
A<3>	Output	F11	BANK0								Unknown		<input type="checkbox"/>
A<4>	Output	C11	BANK0								Unknown		<input type="checkbox"/>
A<5>	Output	D11	BANK0								Unknown		<input type="checkbox"/>
A<6>	Output	E9	BANK0								Unknown		<input type="checkbox"/>
A<7>	Output	F9	BANK0								Unknown		<input type="checkbox"/>
clk	Input	C9	BANK0								Unknown		<input type="checkbox"/>
res	Input	V16	BANK2					PULLDOWN			Unknown		<input type="checkbox"/>
rst	Input	K17	BANK1					PULLDOWN			Unknown		<input type="checkbox"/>
r_a	Input	K18	BANK1					PULLUP			Unknown		<input type="checkbox"/>
r_b	Input	G18	BANK1					PULLUP			Unknown		<input type="checkbox"/>

Рисунок 13. План размещения контактов на отладочной плате.

### Практическая работа № 3

В практической работе № 2 требуется реализовать устройство, описанное в лабораторной работе № 4 на отладочной плате S3ESK. Вывод информации требуется осуществлять на ЖК-дисплей.

Для того чтобы использовать ЖК дисплей необходимо в проекте описать для него контроллер. Описанный контроллер есть в разделе 2.1.2 «Описание используемых элементов». Данные должны выдаваться

последовательно в режиме реального времени, поэтому в данной работе требуется поставить делитель частоты для того, чтобы ЖК-дисплей успевал инициализироваться, а также для того, чтобы отображаемые выходные значения не менялись слишком быстро.

ЖК-дисплей имеет 32 адресуемые ячейки, которые выводятся на дисплей:

Character Display Addresses																Undisplayed Addresses			
1	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	...	27
2	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	...	67
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	...	40

Рисунок 14. Адресуемые ячейки вывода информации ЖК-дисплея.

ЖК-дисплей имеет собственную таблицу выводимых значений (рис. 15):

DB7	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	
DB6	0	0	0	0	1	1	1	1	0	0	1	1	1	1	1	
DB5	0	1	1	0	0	1	1	1	1	0	0	1	1			
DB4	0	0	1	0	1	0	1	0	1	0	1	0	1			
xxxx0000																
xxxx0001																
xxxx0010																
xxxx0011																
xxxx0100																
xxxx0101																
xxxx0110																
xxxx0111																
xxxx1000																
xxxx1001																
xxxx1010																
xxxx1011																
xxxx1100																
xxxx1101																
xxxx1110																
xxxx1111																

Рисунок 15. Таблица выводимых значений.

- где CG RAM – область символов, программируемых пользователем.

В качестве примера использования ЖК-дисплея приводится пример устройства, который выводит состояния счетчика. Проект имеет следующую иерархическую структуру:

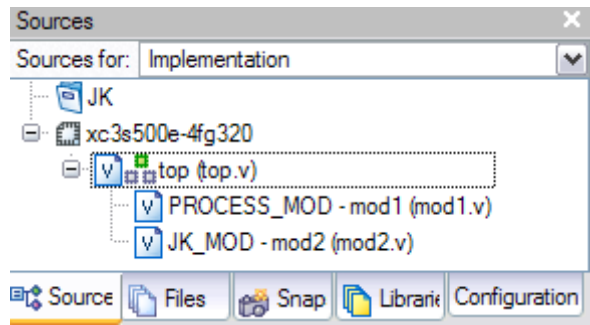


Рисунок 16. Иерархия проекта.

Модуль mod1 представляет собой синхронный прямой счетчик с сигналом сброса, mod2 – контроллер ЖК-дисплея, top – модуль, где описаны соединения контактов модулей mod1 и mod2, а также поставлен делитель частоты для модуля mod1.

Содержимое модуля mod1:

```
`timescale 1ns / 1ps
module mod1(
    input clk_rot,
    output reg [7:0] A,
    input rst);
always @ (posedge clk_rot)
begin
if (rst) A<=0;
else A<=A+1;
end
endmodule
```

Интерфейс ЖК-дисплея:

Содержимое модуля mod2:

```
`timescale 1ns / 1ps
module mod2(
    input CLK,
    input on,
    output E1,
    output RS1,
    output [3:0] SF1,
    input [7:0] A
);
reg E, RS;
reg [3:0] SF;
reg flag1;
reg flag2;
integer cnt=0;
always @(posedge CLK) begin
if (A[3:0]>9) flag1<=1;
```

```

else flag1<=0;
if (A[7:4]>9) flag2<=1;
else flag2<=0;
if (on==1) begin
cnt=cnt+1;
case (cnt)
// инициализация дисплея
750000 : begin SF<=4'b0011; E<=1; end
750012 : begin SF<=4'b0000; E<=0; end
1000012 : begin SF<=4'b0111; E<=1; end
1000024 : begin SF<=4'b0000; E<=0; end
1005024 : begin SF<=4'b0011; E<=1; end
1005036 : begin SF<=4'b0000; E<=0; end
1007036 : begin SF<=4'b0110; E<=1; end
1007048 : begin SF<=4'b0000; E<=0; end
// конфигурация дисплея
1009060 : begin SF<=4'b0010; E<=1; end // function set
1009072 : begin SF<=4'b0000; E<=0; end
1009122 : begin SF<=4'b1000; E<=1; end
1009134 : begin SF<=4'b0000; E<=0; end

1011134 : begin SF<=4'b0000; E<=1; end // entry mode set
1011146 : begin SF<=4'b0000; E<=0; end
1011196 : begin SF<=4'b0110; E<=1; end
1011208 : begin SF<=4'b0000; E<=0; end

1013208 : begin SF<=4'b0000; E<=1; end // display on/off
1013220 : begin SF<=4'b0000; E<=0; end
1013270 : begin SF<=4'b1100; E<=1; end
1013282 : begin SF<=4'b0000; E<=0; end

1015282 : begin SF<=4'b0000; E<=1; end // clear display
1015294 : begin SF<=4'b0000; E<=0; end
1015344 : begin SF<=4'b0001; E<=1; end
1015356 : begin SF<=4'b0000; E<=0; end
// Вывод сообщения
1097356 : begin SF<=4'b1000; E<=1; end // установка начального
//положения курсора
1097368 : begin SF<=4'b0000; E<=0; end
1097418 : begin SF<=4'b0000; E<=1; end
1097430 : begin SF<=4'b0000; E<=0; end

1099430 : begin SF<=4'b0100; E<=1; RS<=1; end // write DD ram
1099442 : begin SF<=4'b0000; E<=0; RS<=0; end // 'A' - 00
1099492 : begin SF<=4'b0001; E<=1; RS<=1; end
1099504 : begin SF<=4'b0000; E<=0; RS<=0; end

1101504 : begin SF<=4'b0011; E<=1; RS<=1; end // '=' - 01
1101516 : begin SF<=4'b0000; E<=0; RS<=0; end

```



```

1101566 : begin SF<=4'b1101;    E<=1; RS<=1; end
1101578 : begin SF<=4'b0000;    E<=0; RS<=0; end

1103578 : begin SF<={1'b0,flag2,~flag2,~flag2};    E<=1; RS<=1; end
// "значение A[7:4]" - 02
1103590 : begin SF<=4'b0000;    E<=0; RS<=0; end
1103640 : begin SF<=A[7:4]-flag2*9;    E<=1; RS<=1; end
1103652 : begin SF<=4'b0000;    E<=0; RS<=0; end

1105652 : begin SF<={1'b0,flag1,~flag1,~flag1};    E<=1; RS<=1; end
// "значение A[3:0]" - 03
1105664 : begin SF<=4'b0000;    E<=0; RS<=0; end
1105714 : begin SF<=A[3:0]-flag1*9;    E<=1; RS<=1; end
1105726 : begin SF<=4'b0000;    E<=0; RS<=0; end

1107726 : begin SF<=4'b0010;    E<=1; RS<=1; end // ' ' - 04
1107738 : begin SF<=4'b0000;    E<=0; RS<=0; end
1107788 : begin SF<=4'b0000;    E<=1; RS<=1; end
1107800 : begin SF<=4'b0000;    E<=0; RS<=0; end

1109800 : begin SF<=4'b0010;    E<=1; RS<=1; end // ' ' - 05
1109812 : begin SF<=4'b0000;    E<=0; RS<=0; end
1109962 : begin SF<=4'b0000;    E<=1; RS<=1; end
1109974 : begin SF<=4'b0000;    E<=0; RS<=0; end

1111974 : begin SF<=4'b0010;    E<=1; RS<=1; end // ' ' - 06
1111986 : begin SF<=4'b0000;    E<=0; RS<=0; end
1112036 : begin SF<=4'b0000;    E<=1; RS<=1; end
1112048 : begin SF<=4'b0000;    E<=0; RS<=0; end

1114048 : begin SF<=4'b0010;    E<=1; RS<=1; end // ' ' - 07
1114060 : begin SF<=4'b0000;    E<=0; RS<=0; end
1114110 : begin SF<=4'b0000;    E<=1; RS<=1; end
1114122 : begin SF<=4'b0000;    E<=0; RS<=0; end

1116122 : begin SF<=4'b0010;    E<=1; RS<=1; end // ' ' - 08
1116134 : begin SF<=4'b0000;    E<=0; RS<=0; end
1116184 : begin SF<=4'b0000;    E<=1; RS<=1; end
1116196 : begin SF<=4'b0000;    E<=0; RS<=0; end

1118196 : begin SF<=4'b0010;    E<=1; RS<=1; end // ' ' - 09
1118208 : begin SF<=4'b0000;    E<=0; RS<=0; end
1118258 : begin SF<=4'b0000;    E<=1; RS<=1; end
1118270 : begin SF<=4'b0000;    E<=0; RS<=0; end

1120270 : begin SF<=4'b0010;    E<=1; RS<=1; end // ' ' - 0A
1120282 : begin SF<=4'b0000;    E<=0; RS<=0; end
1120332 : begin SF<=4'b0000;    E<=1; RS<=1; end
1120344 : begin SF<=4'b0000;    E<=0; RS<=0; end

```

```

1122344 : begin SF<=4'b0010;    E<=1; RS<=1; end // ' ' - 0B
1122356 : begin SF<=4'b0000;    E<=0; RS<=0; end
1122406 : begin SF<=4'b0000;    E<=1; RS<=1; end
1122418 : begin SF<=4'b0000;    E<=0; RS<=0; end

1124418 : begin SF<=4'b0010;    E<=1; RS<=1; end // ' ' - 0C
1124430 : begin SF<=4'b0000;    E<=0; RS<=0; end
1124480 : begin SF<=4'b0000;    E<=1; RS<=1; end
1124492 : begin SF<=4'b0000;    E<=0; RS<=0; end

1126492 : begin SF<=4'b0010;    E<=1; RS<=1; end // ' ' - 0D
1126504 : begin SF<=4'b0000;    E<=0; RS<=0; end
1126554 : begin SF<=4'b0000;    E<=1; RS<=1; end
1126566 : begin SF<=4'b0000;    E<=0; RS<=0; end

1128566 : begin SF<=4'b0010;    E<=1; RS<=1; end // ' ' - 0E
1128578 : begin SF<=4'b0000;    E<=0; RS<=0; end
1128628 : begin SF<=4'b0000;    E<=1; RS<=1; end
1128640 : begin SF<=4'b0000;    E<=0; RS<=0; end

1130640 : begin SF<=4'b0010;    E<=1; RS<=1; end // ' ' - 0F
1130652 : begin SF<=4'b0000;    E<=0; RS<=0; end
1130702 : begin SF<=4'b0000;    E<=1; RS<=1; end
1130714 : begin SF<=4'b0000;    E<=0; RS<=0; end

1132714 : begin SF<=4'b1100;    E<=1; RS<=0; end // перевод курсора
//на следующую строку
1132726 : begin SF<=4'b0000;    E<=0; RS<=0; end
1132776 : begin SF<=4'b0000;    E<=1; RS<=0; end
1132788 : begin SF<=4'b0000;    E<=0; RS<=0; end

1134788 : begin SF<=4'b0010;    E<=1; RS<=1; end // ' ' - 40
1134800 : begin SF<=4'b0000;    E<=0; RS<=0; end
1134850 : begin SF<=4'b0000;    E<=1; RS<=1; end
1134862 : begin SF<=4'b0000;    E<=0; RS<=0; end

1138836 : begin SF<=4'b0010;    E<=1; RS<=1; end // ' ' - 41
1138848 : begin SF<=4'b0000;    E<=0; RS<=0; end
1138898 : begin SF<=4'b0000;    E<=1; RS<=1; end
1138910 : begin SF<=4'b0000;    E<=0; RS<=0; end

1140910 : begin SF<=4'b0010;    E<=1; RS<=1; end // ' ' - 42
1140922 : begin SF<=4'b0000;    E<=0; RS<=0; end
1140972 : begin SF<=4'b0000;    E<=1; RS<=1; end
1140984 : begin SF<=4'b0000;    E<=0; RS<=0; end

1142984 : begin SF<=4'b0010;    E<=1; RS<=1; end // ' ' - 43
1142996 : begin SF<=4'b0000;    E<=0; RS<=0; end

```

```

1143046 : begin SF<=4'b0000;    E<=1; RS<=1; end
1143058 : begin SF<=4'b0000;    E<=0; RS<=0; end

1145058 : begin SF<=4'b0010;    E<=1; RS<=1; end // ' ' - 44
1145070 : begin SF<=4'b0000;    E<=0; RS<=0; end
1145120 : begin SF<=4'b0000;    E<=1; RS<=1; end
1145132 : begin SF<=4'b0000;    E<=0; RS<=0; end

1147132 : begin SF<=4'b0010;    E<=1; RS<=1; end // ' ' - 45
1147144 : begin SF<=4'b0000;    E<=0; RS<=0; end
1147194 : begin SF<=4'b0000;    E<=1; RS<=1; end
1147206 : begin SF<=4'b0000;    E<=0; RS<=0; end

1149206 : begin SF<=4'b0010;    E<=1; RS<=1; end // ' ' - 46
1149218 : begin SF<=4'b0000;    E<=0; RS<=0; end
1149268 : begin SF<=4'b0000;    E<=1; RS<=1; end
1149280 : begin SF<=4'b0000;    E<=0; RS<=0; end

1151280 : begin SF<=4'b0010;    E<=1; RS<=1; end // ' ' - 47
1151292 : begin SF<=4'b0000;    E<=0; RS<=0; end
1151342 : begin SF<=4'b0000;    E<=1; RS<=1; end
1151354 : begin SF<=4'b0000;    E<=0; RS<=0; end

1153354 : begin SF<=4'b0010;    E<=1; RS<=1; end // ' ' - 48
1153366 : begin SF<=4'b0000;    E<=0; RS<=0; end
1153416 : begin SF<=4'b0000;    E<=1; RS<=1; end
1153428 : begin SF<=4'b0000;    E<=0; RS<=0; end

1155428 : begin SF<=4'b0010;    E<=1; RS<=1; end // ' ' - 49
1155440 : begin SF<=4'b0000;    E<=0; RS<=0; end
1155490 : begin SF<=4'b0000;    E<=1; RS<=1; end
1155502 : begin SF<=4'b0000;    E<=0; RS<=0; end

1157502 : begin SF<=4'b0010;    E<=1; RS<=1; end // ' ' - 4A
1157514 : begin SF<=4'b0000;    E<=0; RS<=0; end
1157564 : begin SF<=4'b0000;    E<=1; RS<=1; end
1157576 : begin SF<=4'b0000;    E<=0; RS<=0; end

1159576 : begin SF<=4'b0010;    E<=1; RS<=1; end // ' ' - 4B
1159588 : begin SF<=4'b0000;    E<=0; RS<=0; end
1159638 : begin SF<=4'b0000;    E<=1; RS<=1; end
1159650 : begin SF<=4'b0000;    E<=0; RS<=0; end

1161650 : begin SF<=4'b0010;    E<=1; RS<=1; end // ' ' - 4C
1161662 : begin SF<=4'b0000;    E<=0; RS<=0; end
1161712 : begin SF<=4'b0000;    E<=1; RS<=1; end
1161724 : begin SF<=4'b0000;    E<=0; RS<=0; end

1163724 : begin SF<=4'b0010;    E<=1; RS<=1; end // ' ' - 4D

```

```

1163736 : begin SF<=4'b0000;    E<=0; RS<=0; end
1163786 : begin SF<=4'b0000;    E<=1; RS<=1; end
1163798 : begin SF<=4'b0000;    E<=0; RS<=0; end

1165798 : begin SF<=4'b0010;    E<=1; RS<=1; end // ' ' - 4E
1165810 : begin SF<=4'b0000;    E<=0; RS<=0; end
1165860 : begin SF<=4'b0000;    E<=1; RS<=1; end
1165872 : begin SF<=4'b0000;    E<=0; RS<=0; end

1167946 : begin SF<=4'b0010;    E<=1; RS<=1; end // ' ' - 4F
1167958 : begin SF<=4'b0000;    E<=0; RS<=0; end
1168008 : begin SF<=4'b0000;    E<=1; RS<=1; end
1168020 : begin SF<=4'b0000;    E<=0; RS<=0; end
endcase
end
else cnt=0;
end

assign SF1=SF;
assign E1=E;
assign RS1=RS;

endmodule

```

Содержимое файла top:

```

`timescale 1ns / 1ps
module top(
    input clk,
    input rst,
    output E1,
    output RS1,
    output [3:0] SF1
);
reg on;
reg clk_1;
wire [7:0] A;
reg [24:0] count;
mod1 PROCESS_MOD (
    //прикрепление из модуля mod2 сигналов
    .A(A),
    .rst(rst),
    .clk(clk_1));
mod2 JK_MOD (
    //прикрепление из модуля mod1 сигналов
    .CLK(clk),
    .E1(E1),
    .RS1(RS1),
    .SF1(SF1),

```

```

        .on(on),
        .A(A));
always @ (posedge clk)
begin
count<=count+1;
if(count==0)
begin
    on<=1;
    clk_1<=clk_1+1;
end
else if(count== 25'h11D295) on<=0;
end
endmodule

```

План размещения контактов на отладочной плате (рис 17):

I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name	Local Clock
clk	Input	c9	BANK0	LVC MOS33	N/A	3.30							<input type="checkbox"/>
E1	Output	M18	BANK1	LVC MOS33	N/A	3.30			SLOW		Unknown		<input type="checkbox"/>
rst	Input	K17	BANK1	LVTTL	N/A	3.30		PULLDOWN			Unknown		<input type="checkbox"/>
RS1	Output	L18	BANK1	LVC MOS33	N/A	3.30			SLOW		Unknown		<input type="checkbox"/>
SF1<0>	Output	r15	BANK1	LVC MOS33	N/A	3.30			SLOW		Unknown		<input type="checkbox"/>
SF1<1>	Output	r16	BANK1	LVC MOS33	N/A	3.30			SLOW		Unknown		<input type="checkbox"/>
SF1<2>	Output	p17	BANK1	LVC MOS33	N/A	3.30			SLOW		Unknown		<input type="checkbox"/>
SF1<3>	Output	m15	BANK1	LVC MOS33	N/A	3.30			SLOW		Unknown		<input type="checkbox"/>

Рисунок 17. План размещения контактов на отладочной плате.

## ТЗ магистерской диссертации

### Тема магистерской диссертации:

«Средства поддержки дистанционного обучения проектированию цифровых устройств на ПЛИС».

### Обоснование актуальности темы:

Актуальность данной работы заключается в малой эффективности существующих на сегодняшний день ЭУМК для дистанционного обучения.

### Объект исследования:

Средства дистанционного обучения проектированию ПЛИС на САПР.

### Цель исследования:

Создание подхода к составлению учебных материалов в рамках дистанционного обучения по дисциплине «Проектирование цифровых устройств на ПЛИС» и по схожим дисциплинам.

### Задачи исследования:

1. Обзор логических структур ЭУМК.
2. Разработка курса лекций по проектированию цифровых устройств на ПЛИС.
3. Разработка методических указаний по использованию САПР.

4. Разработка методических указаний по эксплуатации обучающего стенда.
5. Разработка, апробация и реализация модулей лабораторных работ по проектированию цифровых устройств на ПЛИС.
6. Разработка, апробация и реализация модулей лабораторных работ по проектированию цифровых устройств на ПЛИС.
7. Исследование и обобщение научных трудов по способам и методам составления учебных материалов в рамках дистанционного обучения по дисциплине «Проектирование цифровых устройств на ПЛИС» и по схожим дисциплинам.

**Ожидаемые результаты исследования:**

Создание требований к составлению учебных материалов в рамках дистанционного обучения по дисциплине «Проектирование цифровых устройств на ПЛИС» и по схожим дисциплинам, а также создание ЭУМК «Средства поддержки дистанционного обучения проектированию цифровых устройств на ПЛИС».