МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

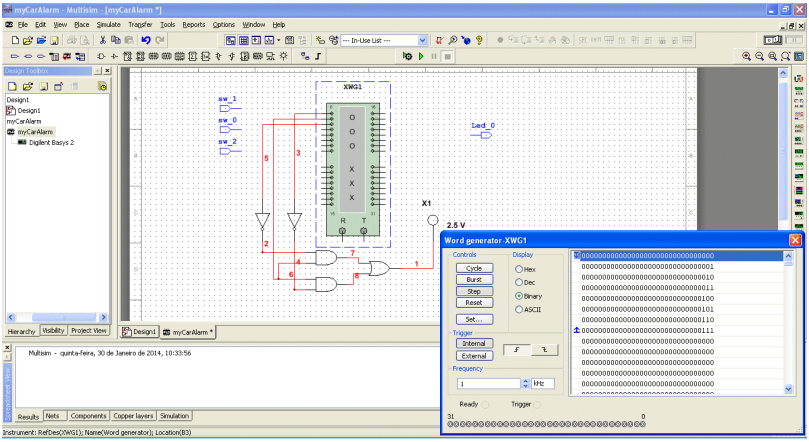
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Воронежский государственный технический университет»

В. Ф. Барабанов, Н. И. Гребенникова,

Д. Н. Донских, С. А. Коваленко

РАЗРАБОТКА И ПРОТОТИПИРОВАНИЕ ЦИФРОВЫХ УСТРОЙСТВ НА ЯЗЫКАХ VHDL И VERILOG

Учебно-методическое пособие



Утверждено учебно-методическим советом университета
в качестве учебно-методического пособия

Воронеж 2018

УДК 681.3.06(07)
ББК 32.973-018я7
Б24

Барабанов, В. Ф.

Разработка и прототипирование цифровых устройств на языках VHDL и Verilog: учебно-методическое пособие [Электронный ресурс]. – Электрон, текстовые и граф. данные (3,7 Мб) / В. Ф. Барабанов, Н. И. Гребенникова, Д. Н. Донских, С. А. Коваленко. – Воронеж: ФГБОУ ВО «Воронежский государственный технический университет», 2018. – 1 электрон.
Б24 опт. диск (CD-ROM): цв. – Систем, требования: ПК 500 и выше; 256 Мб ОЗУ; Windows XP; SVGA с разрешением 1024x768;. Adobe Acrobat; CD-ROM дисковод; мышь. – Загл. с экрана.

ISBN 978-5-7731-0709-5

В пособии рассматриваются основные приемы разработки цифровых устройств на языках VHDL и Verilog, приводятся задания по темам лабораторных работ.

Лабораторная работа № 1 разработана Д. Н. Донских, лабораторная работа № 2 разработана С. А. Коваленко, лабора-торная работа № 3 разработана Н. И. Гребенниковой, лабораторные работы № 4-5 разработаны В. Ф. Барабановым.

Издание соответствует требованиям Федерального государственного образовательного стандарта высшего образования по направлению 09.03.01 «Информатика и вычислительная техника» (профили «Вычислительные машины, комплексы, системы и сети», «Системы автоматизированного проектирования», «Системы автоматизированного проектирования в машиностроении»), дисциплинам «Автоматизация проектирования вычислительных систем», «Программирование на VHDL», «Программирование на Verilog».

Ил. 39. Библиогр.: 4 назв.

УДК 681.3.06(07)
ББК 32.973-018я7

ISBN 978-5-7731-0709-5

ВВЕДЕНИЕ

Программируемые логические интегральные схемы (ПЛИС) – важное направление в современной микроэлектронике. Благодаря своей гибкости ПЛИС стали широко распространены при разработке прототипов цифровых устройств для последующей их реализации на элементной базе специализированных заказных больших интегральных схем (ASIC), а также при мелкосерийном производстве. Специалисты в этой области становятся всё более востребованными на международном рынке труда.

Данный цикл лабораторных работ позволит изучить основные возможности САПР Xilinx ISE Design Suite и среды моделирования ModelSim, освоить навыки разработки и прототипирования цифровых устройств с использованием языков описания аппаратуры Verilog HDL и VHDL.

1. ЛАБОРАТОРНАЯ РАБОТА №1
ЗНАКОМСТВО СО СРЕДОЙ ПРОЕКТИРОВАНИЯ XILINX ISE DESIGN SUITE

1.1. Цель работы

Целью данной лабораторной работы является знакомство со средой проектирования Xilinx ISE Design Suite, обучение созданию моделей цифровых устройств с использованием языков описания аппаратуры Verilog HDL и VHDL, и их функциональному тестированию с помощью средств программной симуляции.

1.2. Краткие теоретические сведения

Для создания проекта нового устройства нужно запустить основную программу среды ISE Project Navigator (например, из главного системного меню, либо с помощью значка на рабочем столе). Затем в строке меню нужно выбрать File-> New_Project (рис. 1.1).

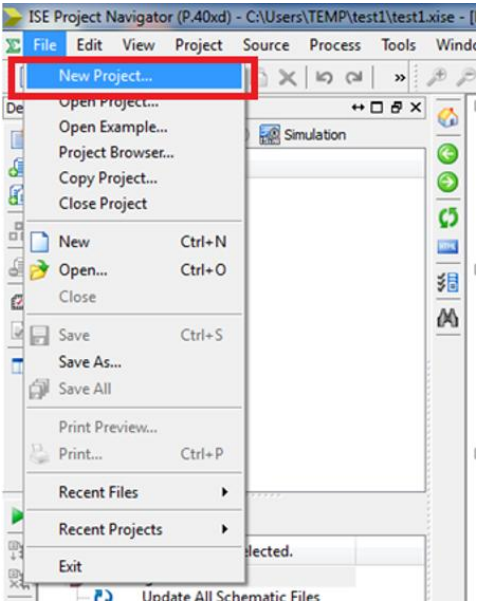


Рис. 1.1. Пример создания нового проекта

В открывшемся окне (New Project Wizard, см. рис. 1.2) необходимо указать имя проекта и выбрать место его размещения на жестком диске. Имя проекта и путь должны содержать только латинские буквы. Для описания модели устройства воспользуемся языком описания аппаратуры (HDL).

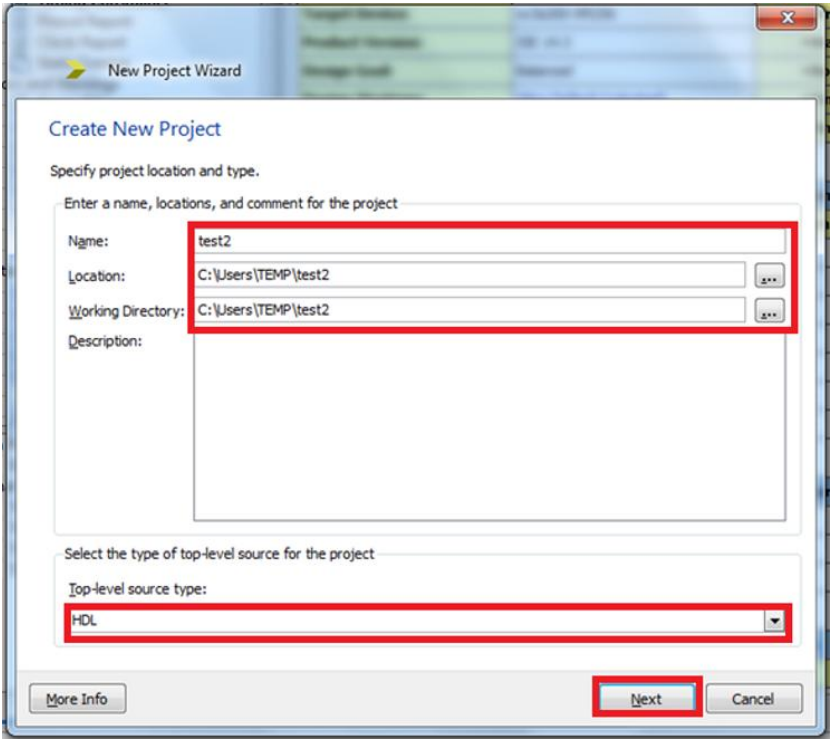


Рис. 1.2. Процесс создания нового проекта

После нажатия кнопки Next появится меню настройки проекта (рис. 1.3). Здесь можно установить вид используемой ПЛИС, её корпус, допустимые параметры скорости, симулятор используемый по умолчанию. Для работы с отладочным комплектом Spartan 3 Starter Kit установите

настройки, как изображено на рис. 1.3. и нажмите кнопку Next.

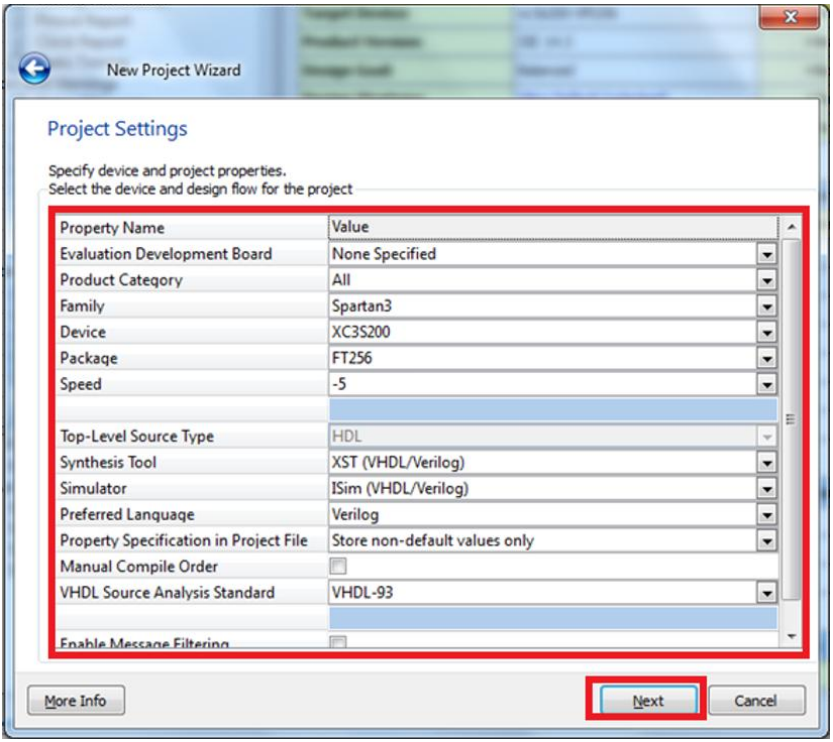


Рис. 1.3. Процесс создания нового проекта

После этого откроется окно со сводной информацией о создаваемом проекте (рис. 1.4). Ознакомьтесь с данной информацией и нажмите кнопку Finish.

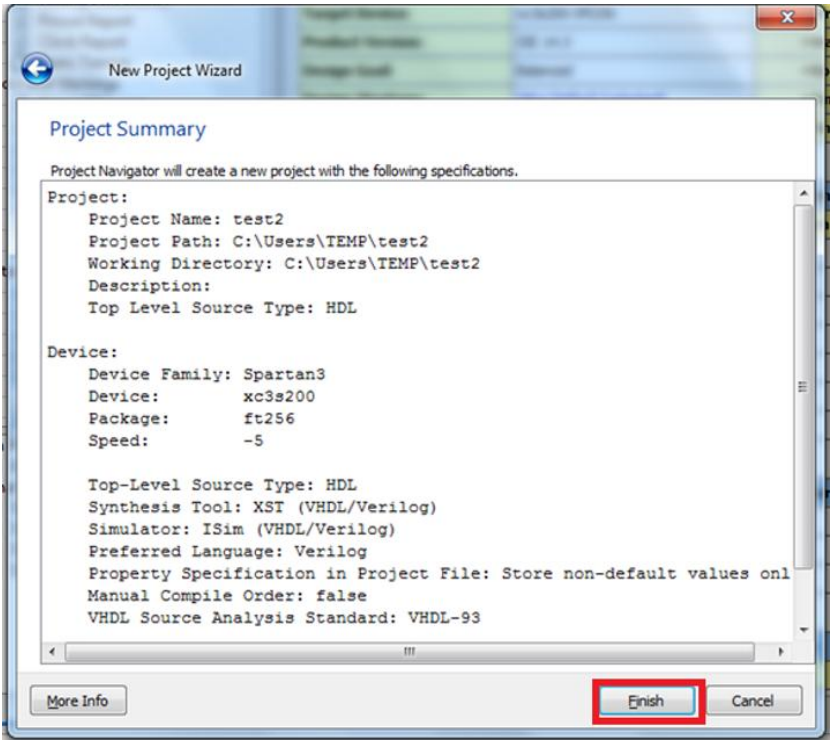


Рис. 1.4. Процесс создания нового проекта

Для создания нового модуля необходимо выбрать пункт меню Project -> New Source (рис. 1.5).

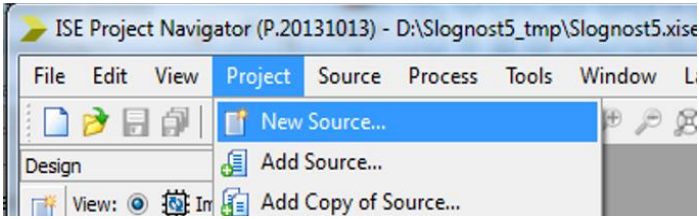


Рис. 1.5. Процесс создания нового проекта, добавление нового модуля

В открывшемся окне нужно выбрать тип исходного файла (в данном случае – модуль на языке Verilog HDL), указать имя файла и место его расположения. Нажать кнопку Next (рис. 1.6).

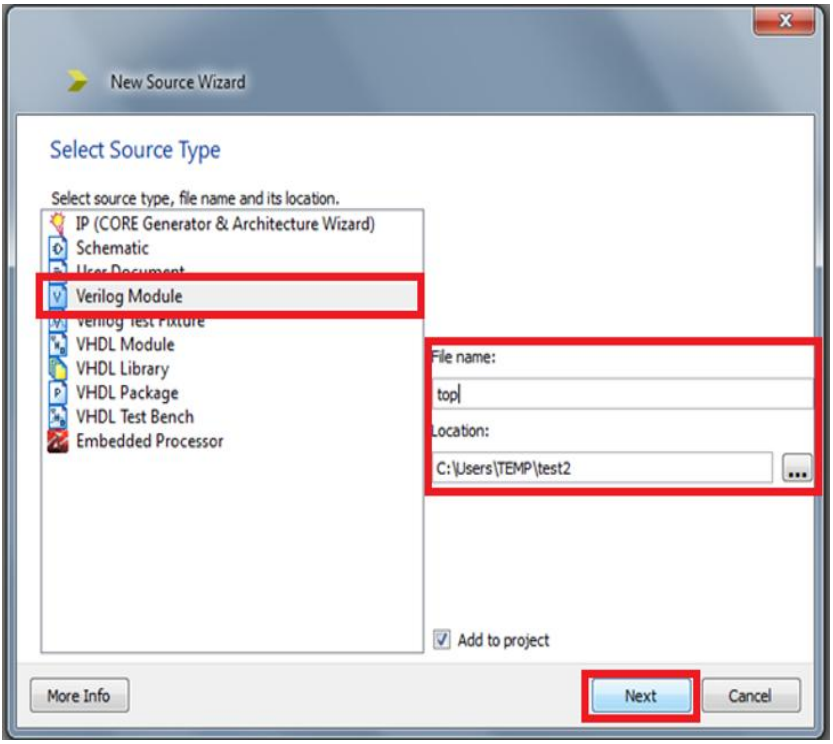


Рис. 1.6. Процесс создания нового проекта, выбор типа модуля

В появившемся меню можно описать интерфейс создаваемого модуля. Здесь можно указать имя сигнала, направление (вход-выход) и разрядность (см. рис. 1.7). Этот шаг можно пропустить, и описать интерфейс модуля вручную в коде.

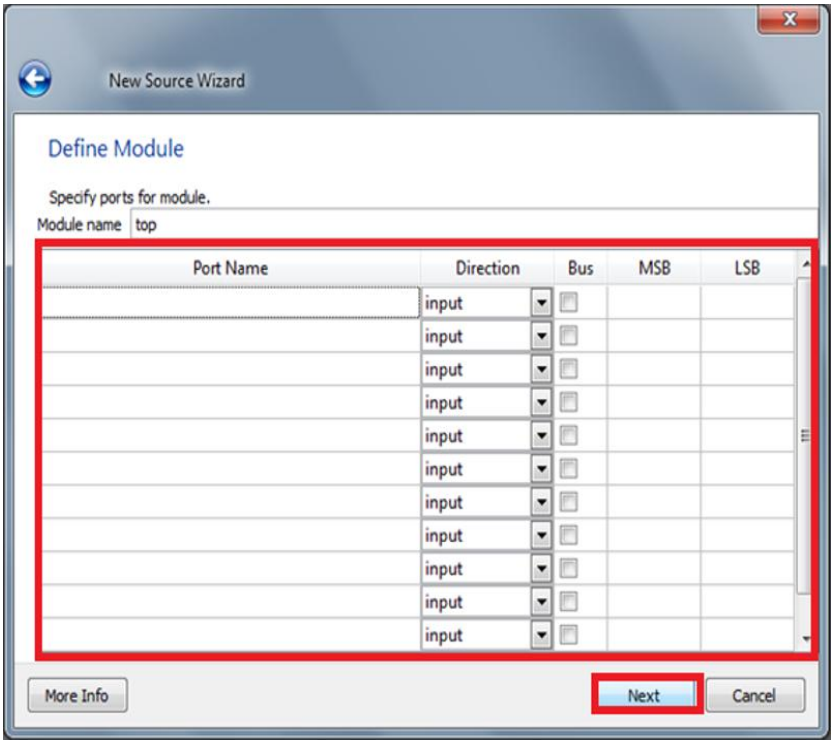


Рис. 1.7. Процесс создания нового проекта, настройка сигналов

Далее необходимо нажать Next, просмотреть сводную информацию по создаваемому модулю, и нажать Finish.

На данном этапе выполнения лабораторной работы необходимо разработать модель цифрового устройства в соответствии с выбранным ранее вариантом задания.

Пример модели цифрового устройства, описанная на языке Verilog HDL представлена на рис. 1.8.

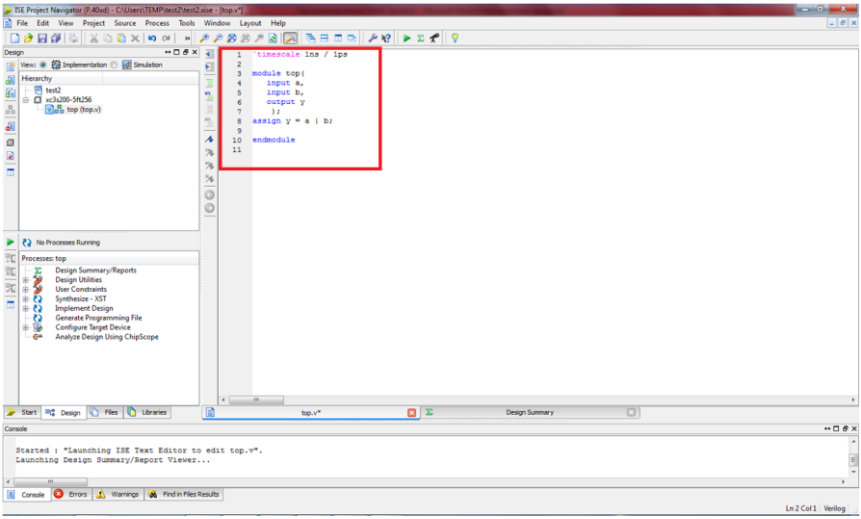


Рис. 1.8. Пример модели устройства

Для тестирования разработанной модели цифрового устройства будем использовать поведенческое моделирование.

Для работы с симулятором нужно перейти в соответствующий режим, выбрав Simulation в левой верхней части экрана (см. рис. 1.9):

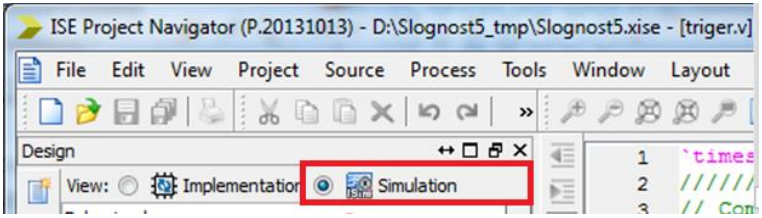


Рис. 1.9. Переход в режим симуляции

Для создания тестовой оболочки нужно выбрать пункт меню Project -> New Source, и указать тип модуля Verilog Test Fixture или VHDL Test Bench, в зависимости от языка, использованного для описания тестируемой модели. Затем нужно указать имя тестового модуля и нажать Next. В открывшемся окне необходимо выбрать модуль, для которого требуется создать тестовое окружение, нажать Next, и затем Finish.

После этого шаблон тестового модуля будет добавлен в проект (пример тестового модуля представлен на рис. 1.10).

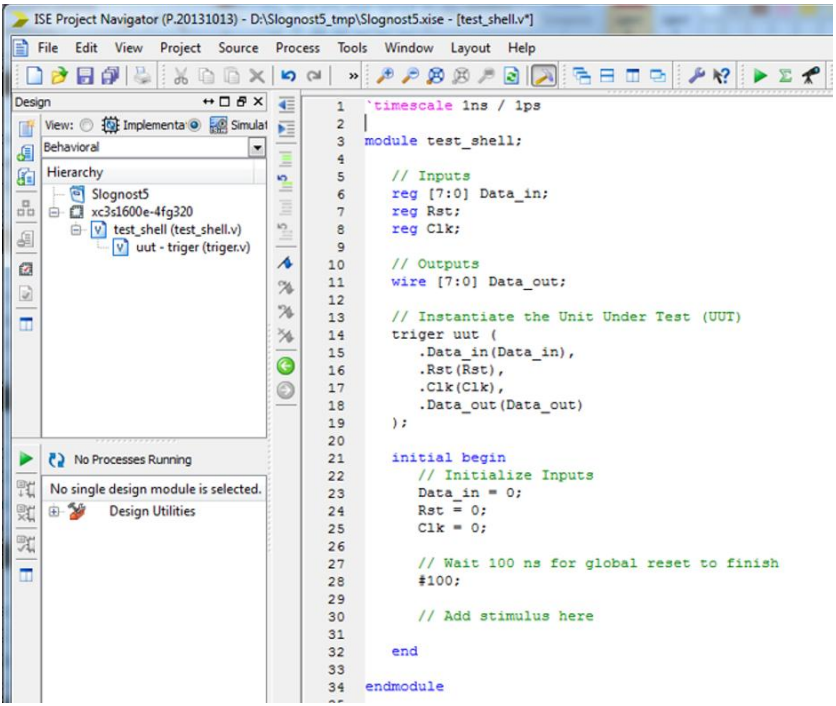


Рис. 1.10. Пример тестового модуля

Далее необходимо задать последовательность внешних сигналов для тестирования разработанного модуля.

Пример последовательности внешних сигналов для тестирования модели D-триггера представлен на рис. 1.11.

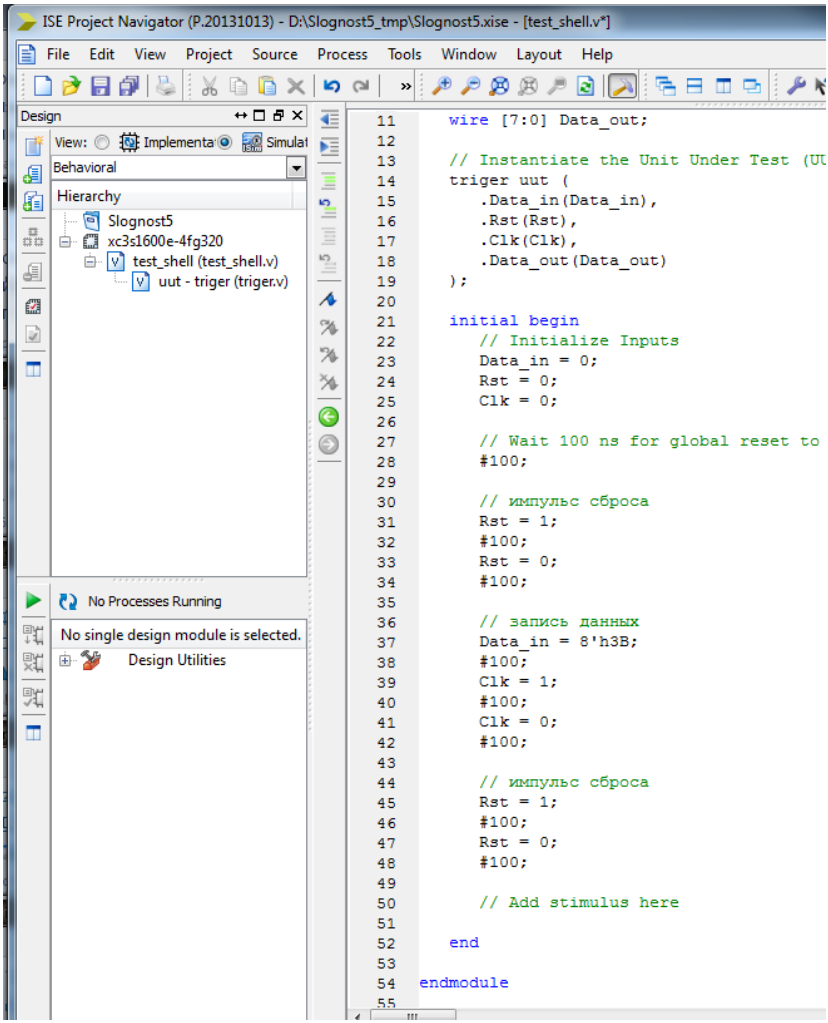


Рис. 1.11. Пример последовательности внешних сигналов

Запуск теста выполняется следующим образом. Выделяем одинарным щелчком мыши тестовый модуль (рис. 1.12).

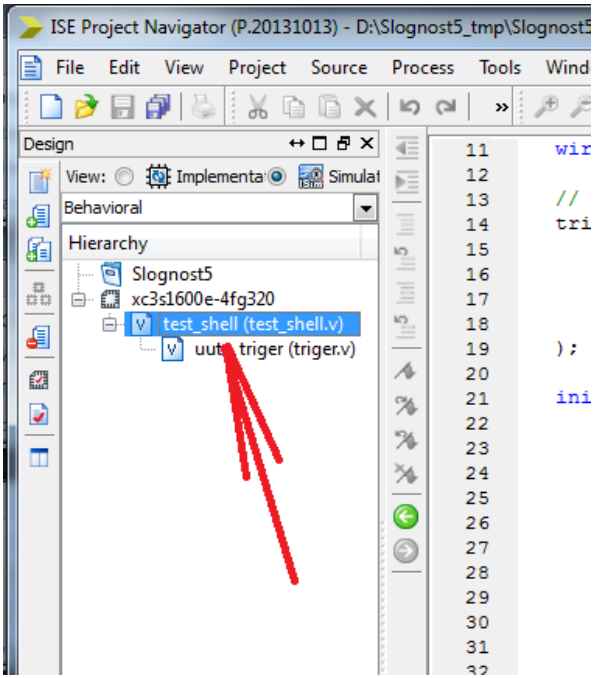


Рис. 1.12. Запуск теста

Затем, в меню Processes запускаем симулятор двойным щелчком мыши (рис. 1.12).

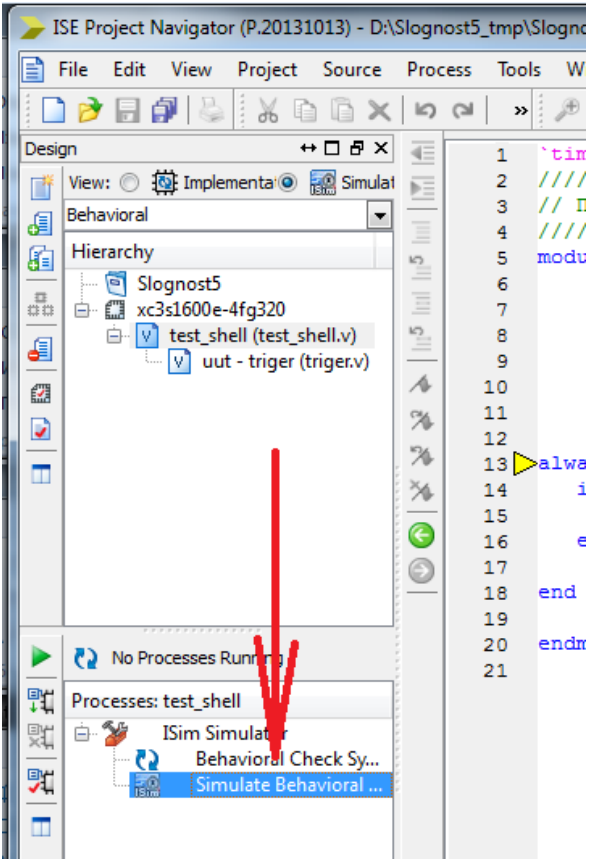


Рис. 1.13. Запуск теста

Если при описании модуля не было допущено ошибок, то запустится окно симулятора (см. рис. 1.14). Для отображения всей временной диаграммы воспользуемся кнопками управления масштабом.

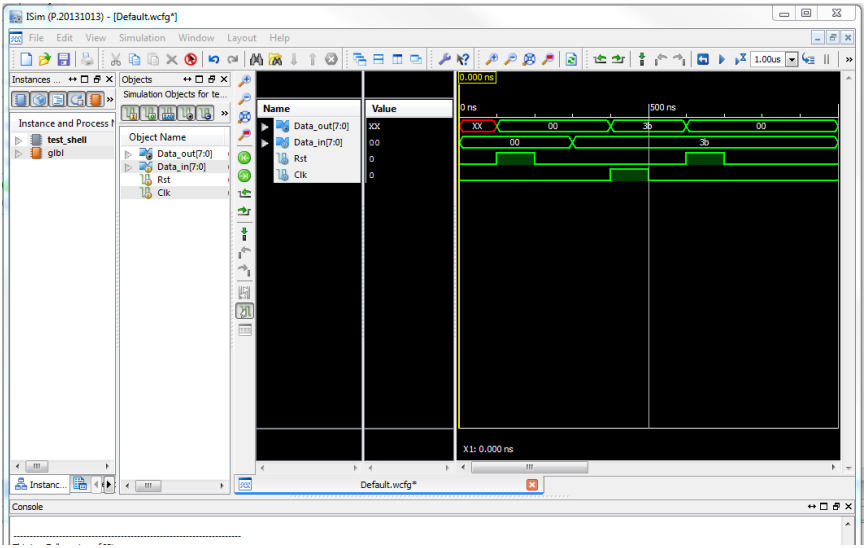


Рис. 1.14. Окно симулятора

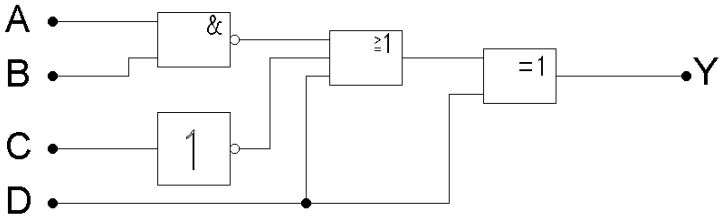
1.3. Задание на лабораторную работу

1) Разработать модель цифровой схемы на языках Verilog HDL и VHDL согласно выбранному варианту.

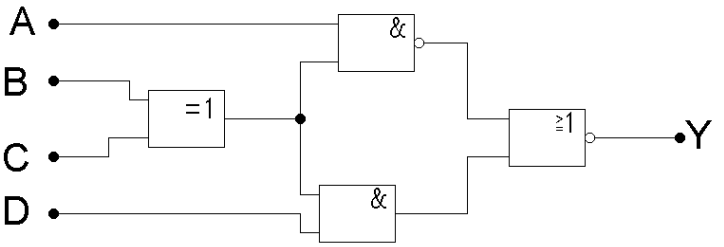
2) Разработать тестовую оболочку для полученных моделей.

Варианты заданий:

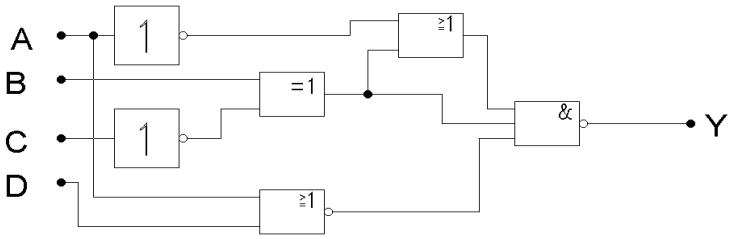
Вариант 1



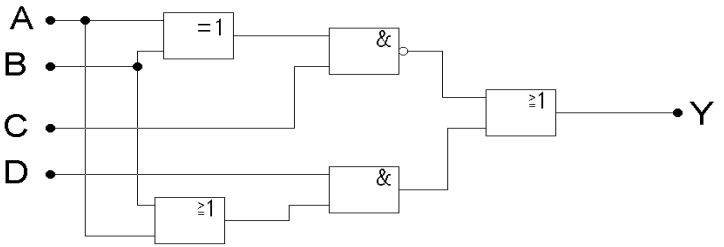
Вариант 2



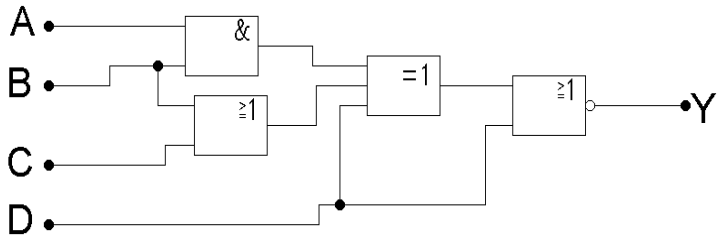
Вариант 3



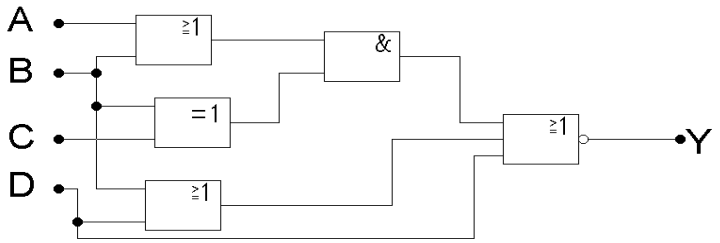
Вариант 4



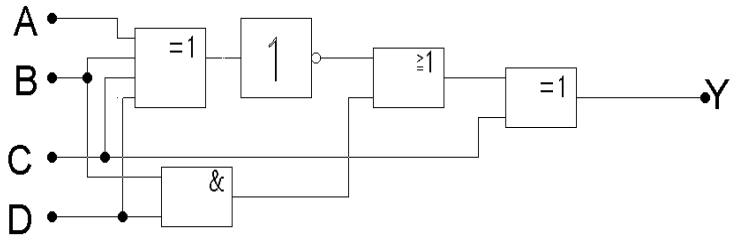
Вариант 5



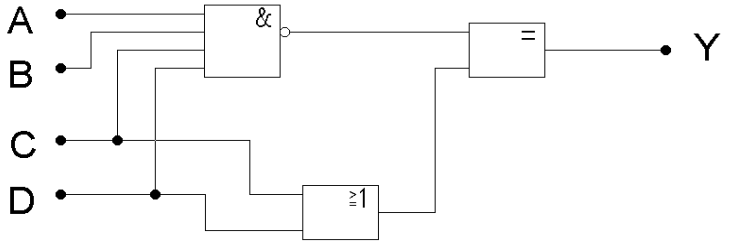
Вариант 6



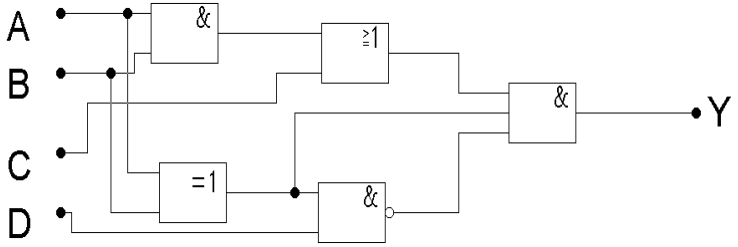
Вариант 7



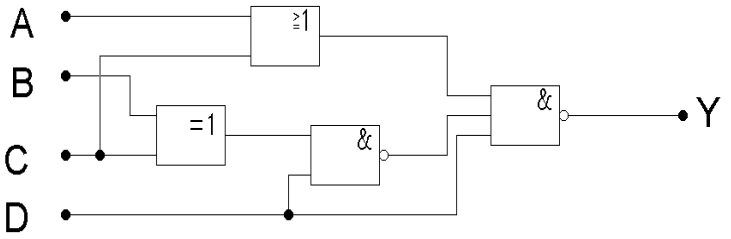
Вариант 8



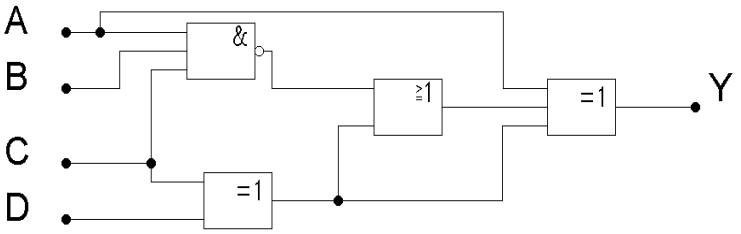
Вариант 9



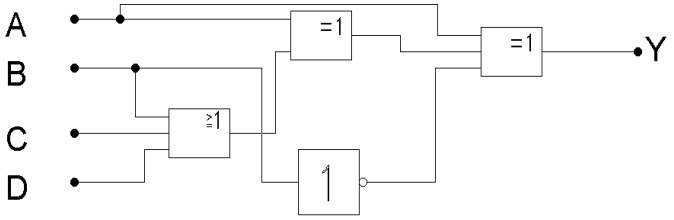
Вариант 10



Вариант 11

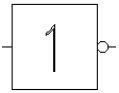


Вариант 12

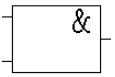


1.4. Дополнительная информация

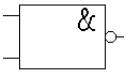
Используемые УГО



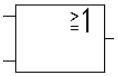
Логический элемент НЕ



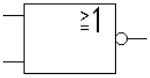
Логический элемент И



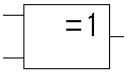
Логический элемент И-НЕ



Логический элемент ИЛИ



Логический элемент ИЛИ-НЕ



Логический элемент
ИСКЛЮЧАЮЩЕЕ ИЛИ

1.5. Требования к отчету по лабораторной работе

Отчет должен содержать следующую информацию по лабораторной работе:

– название работы, цель и задание, согласно варианту;
– код описания модели цифрового устройства;
– временные диаграммы работы разработанной модели.

2. ЛАБОРАТОРНАЯ РАБОТА № 2
РАБОТА С ОТЛАДОЧНЫМ КОМПЛЕКТОМ SPARTAN 3 STARTER KIT

2.1. Цель работы

Целью данной лабораторной работы является разработка моделей цифровых устройств с использованием языков описания аппаратуры Verilog HDL и VHDL, и их последующее тестирование с использованием отладочного комплекта Spartan 3 Starter Kit.

2.2. Краткие теоретические сведения

После завершения этапа разработки модели цифрового устройства и её тестирования средствами симулятора можно перейти к финальному этапу создания устройства на основе ПЛИС-операции конфигурирования интегральной схемы программируемой логики.

Часть выводов ПЛИС в отладочной плате Spartan 3 Starter Kit подключена к светодиодным индикаторам и переключателями. Для выполнения данной лабораторной работы мы будем пользоваться переключателями (SW7- SW0) и индикаторами (LD7-LD0)

Каждый индикатор / переключатель подключается через определенный вывод ПЛИС, перечень которых представлен в табл. 1 и 2 соответственно [3].

Таблица 1

Соответствие выводов ПЛИС светодиодам

Светодиод	LD7	LD 6	LD 5	LD 4	LD 3	LD 2	LD 1	LD 0
FPGA pin	P11	P12	N12	P13	N14	L12	P14	K12

Таблица 2

Соответствие выводов ПЛИС переключателям

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

Определение соответствия входных и выходных портов разработанной модели цифрового устройства и выводов ПЛИС производится в файле пользовательских ограничений (*.ucf). Процесс его создания будет рассмотрен далее.

При выделении модуля верхнего уровня в окне представления иерархической структуры проекта (в режиме Implmentation), в окне «Processes» становятся доступны действия, показанные на рис. 2.1.

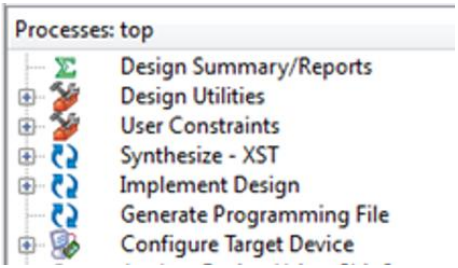


Рис. 2.1. Действия в окне Process

Основные операции над модулем верхнего уровня [1]

1. Synthesize – выполняется синтез кода: проводится проверка синтаксиса, синтез принципиальной схемы на уровне регистровых передач (*register transfer level, RTL*). На данном этапе код модели анализируется, и представляется в базисе имеющихся на выбранном кристалле схемы ресурсов (элементов памяти, генераторов логических функций, триггеров, матриц коммутации цепей и т.п.) Т.е. условные RTL компоненты упаковываются в ресурсы заданного кристалла ПЛИС без размещения компонентов в кристалле.

2. Implement Design – выполняет задачу размещения полученных после синтеза компонентов на заданном кристалле, учитывая пользовательские ограничения, заданные в файле «*.ucf». На этом этапе происходит оптимизации проекта, и привязка сигналов к выводам ПЛИС;

3. Generate Programming File – используя полученное в предыдущем пункте размещение на кристалле, генерируется файл прошивки ПЛИС («*.bit»);

4. Configure Target Device – САПР находит доступные программаторы, производит загрузку заданной прошивки в ПЛИС.

Для конфигурирования интегральной схемы необходимо располагать специальными аппаратными средствами: программатором, с интерфейсом JTAG для соединения с ПЛИС. Программатор должен быть подключен к компьютеру специальными кабелями (в случае комплекта Spartan 3 Starter Kit подключение программатора осуществляется через LPT-порт).

Рассмотрим процесс создания файла пользовательских ограничений (*.ucf). Выберем пункт меню Tools->PlanAhead->I/O Pin Planning (рис. 2.2).

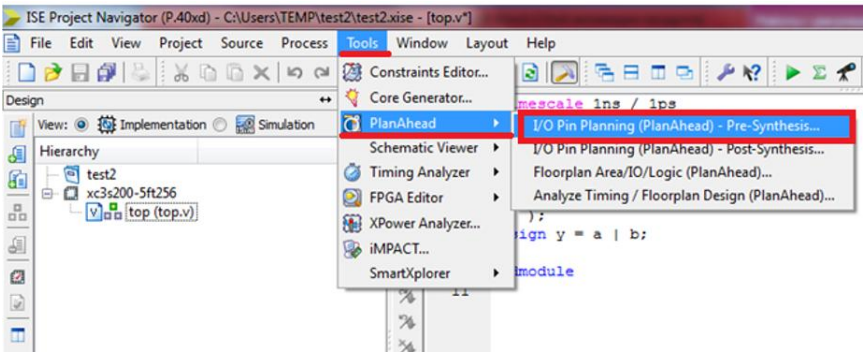


Рис. 2.2. Процесс создания файла пользовательских ограничений

После этого появится предупреждение о необходимости создания файла ограничений (UCF). Нажмите YES (рис. 2.3).

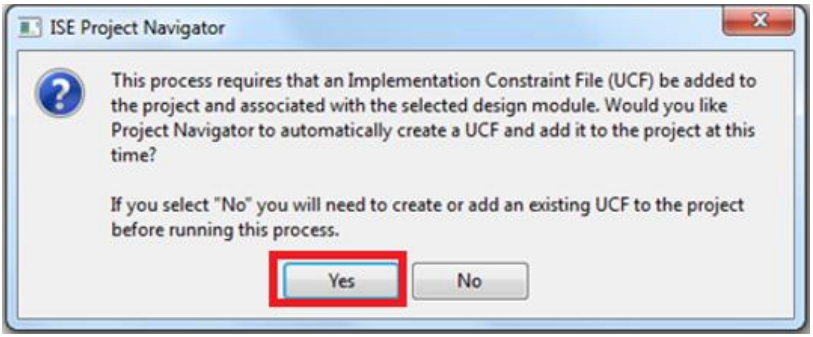


Рис. 2.3. Предупреждение о необходимости создания файла ограничений

Дождитесь запуска приложения PlanAhead. Нажмите на ячейку таблицы, находящуюся на пересечении столбца Site и строки с именем входного/выходного сигнала модели. В выпадающем списке выберите имя используемого контакта на ПЛИС (рис. 2.4).

Когда настройка завершена, сохраните изменения и закройте приложение PlanAhead. Убедитесь, что в проект был добавлен файл с расширением *.ucf содержащий описание соответствия сигналов модели и выводов ПЛИС (рис. 2.5).

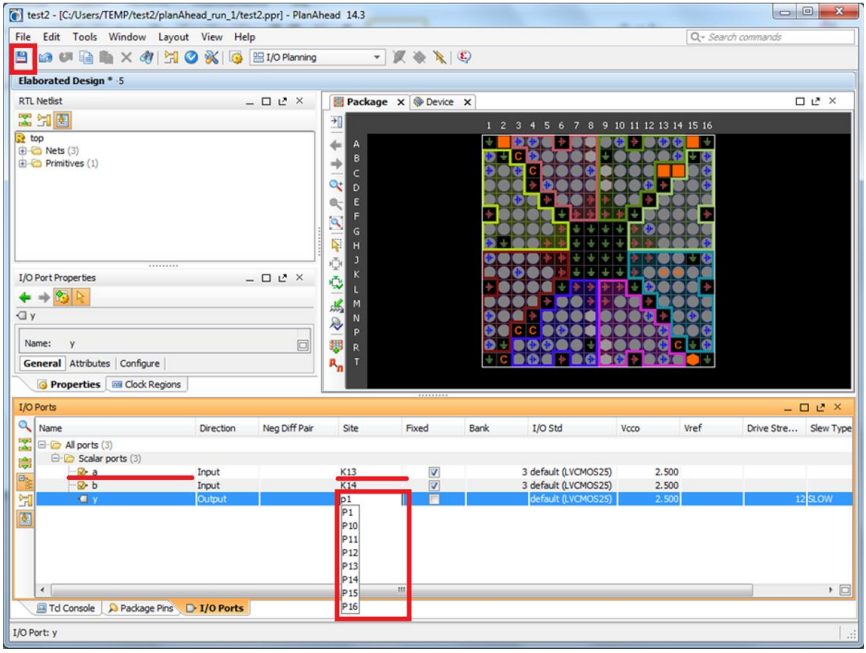


Рис. 2.4. Выбор имени используемого контакта

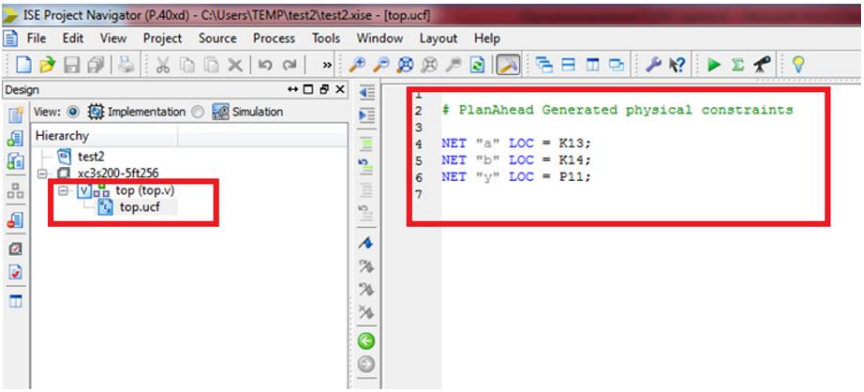


Рис. 2.5. Описание соответствия сигналов модели

Нажмите левой кнопкой на имени файла верхнего уровня в иерархическом представлении проекта, затем дважды кликните по значку Synthesize – XST.

Дождитесь завершения этапа синтеза проекта (рис. 2.6).

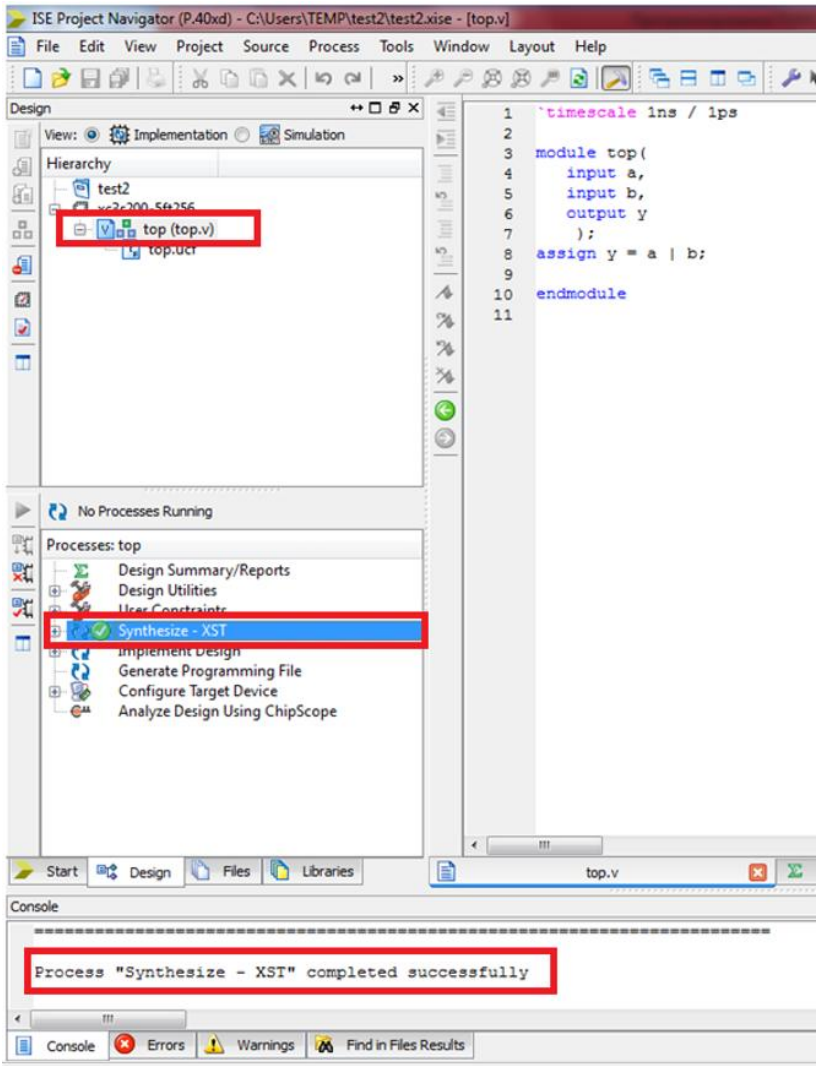


Рис. 2.6. Факт завершения этапа синтеза проекта

Аналогично этапу синтеза запустите имплементацию проекта и дождитесь её окончания (рис. 2.7).

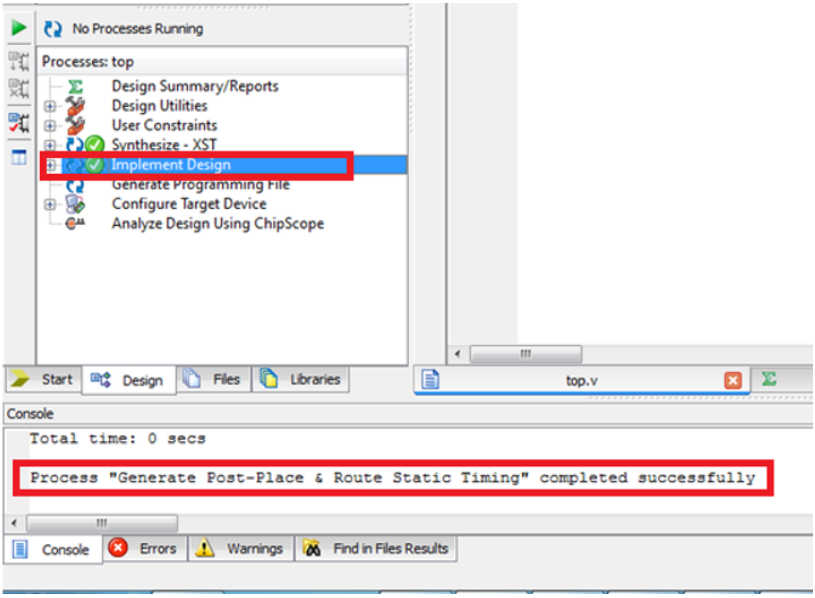


Рис. 2.7. Факт окончания имплементации проекта

Аналогичным образом нужно сгенерировать файл прошивки ПЛИС (рис. 2.8).

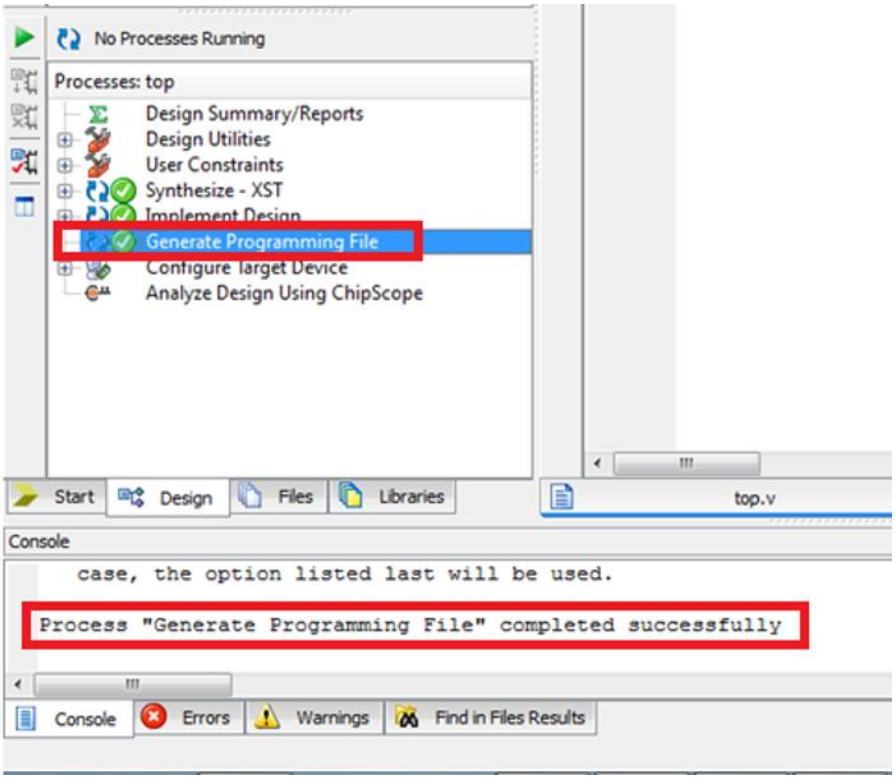


Рис. 2.8. Генерация прошивки

Далее необходимо нажать на значок “+”, расположенный слева от пункта Configure Target Device и выбрать там Manage Configuration Project (iMPACT) (рис. 2.9).

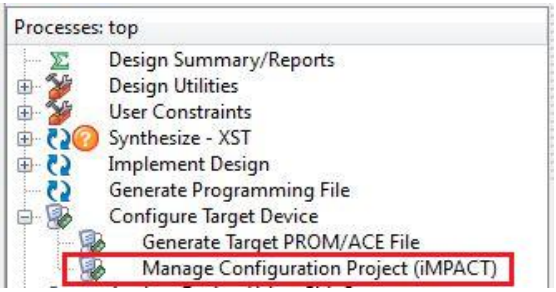


Рис. 2.9. Manage Configuration Project

В открывшемся приложении дважды нажмите на кнопку Boundary Scan и затем на кнопку Initialize Chain для подключения к программатору ПЛИС (рис. 2.10).

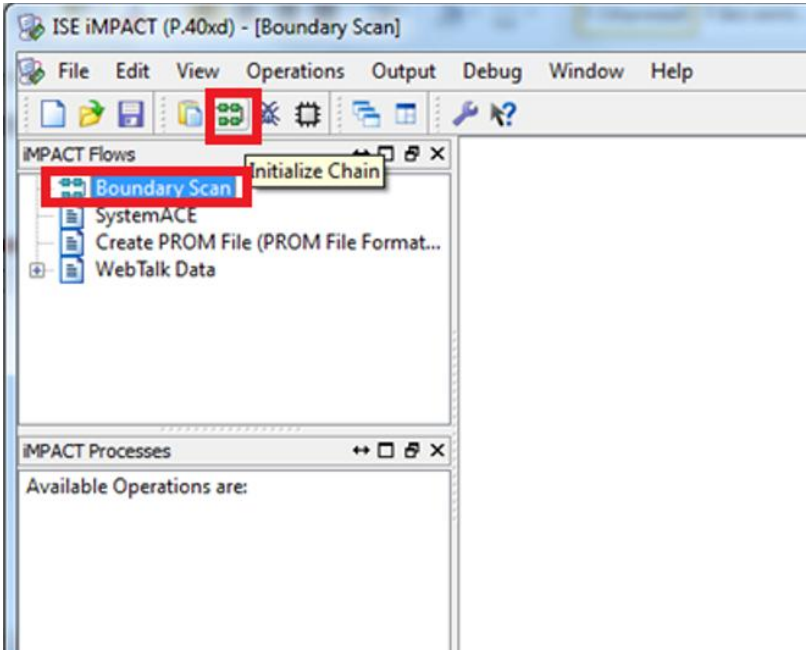


Рис. 2.10. Подключение к программатору ПЛИС

После этого появится предложение указать конфигурационные файлы для ОЗУ и ПЗУ ПЛИС. Нажмите на кнопку Yes (рис. 2.11).

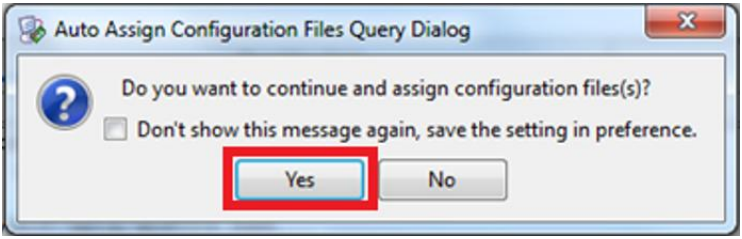


Рис. 2.11. Предложение указать конфигурационные файлы

В появившемся списке выберите файл прошивки для ОЗУ ПЛИС. Нажмите Open (рис. 2.12)..

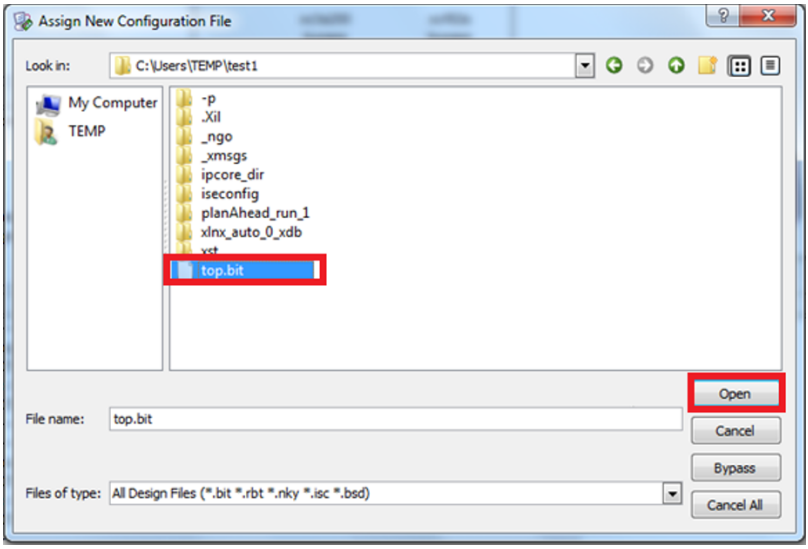


Рис. 2.12. Выбор файла прошивки для ОЗУ

Затем будет предложено выбрать конфигурационный файл для ПЗУ ПЛИС. Нажмите Cancel (рис. 2.13).

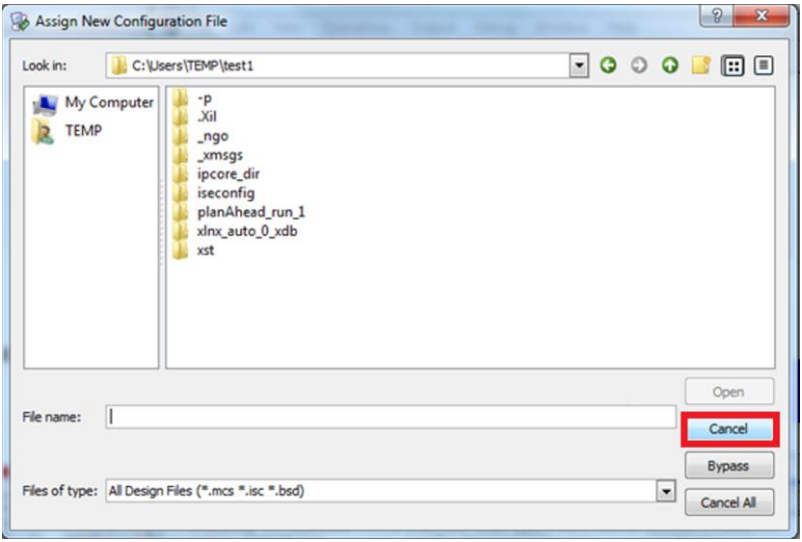


Рис. 2.13. Выбор конфигурационного файла для ПЗУ

В следующем окне оставляем настройки по умолчанию. Нажмите Apply и затем Ok (рис. 2.14).

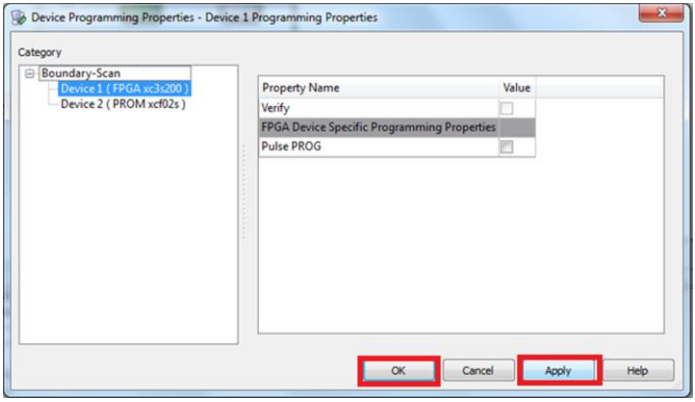


Рис. 2.14. Настройки для программирования

Теперь можно выбрать устройство ПЛИС в списке устройств JTAG цепи и запрограммировать его, нажав кнопку Program (рис. 2.15).

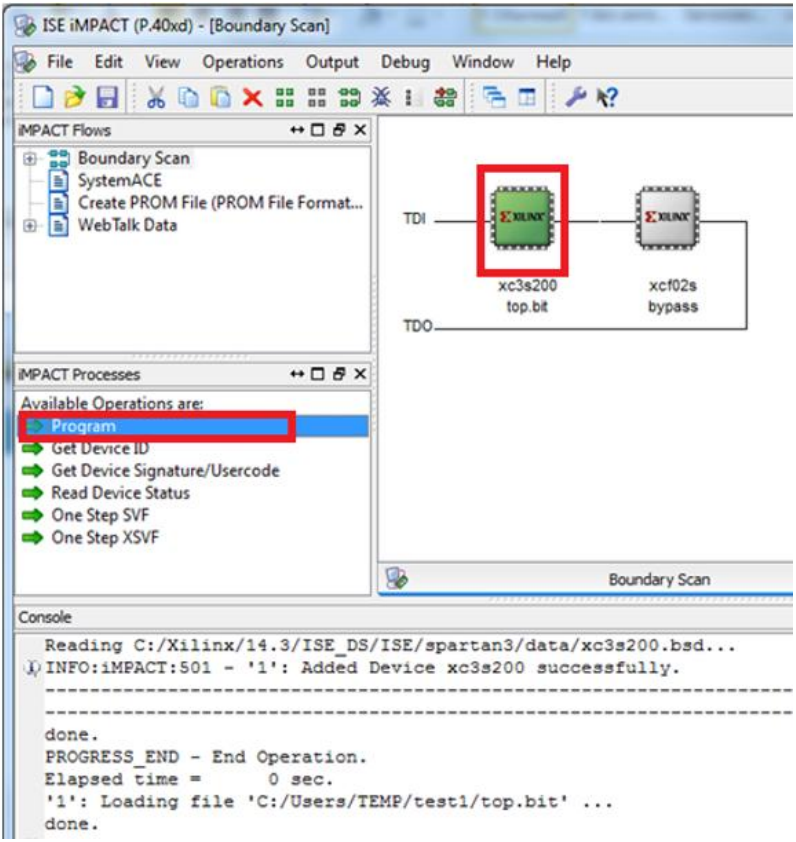


Рис. 2.15. Выбрать устройства ПЛИС в списке устройств JTAG

Дождитесь окончания процесса загрузки. После этого можно протестировать работоспособность устройства (рис. 2.16).

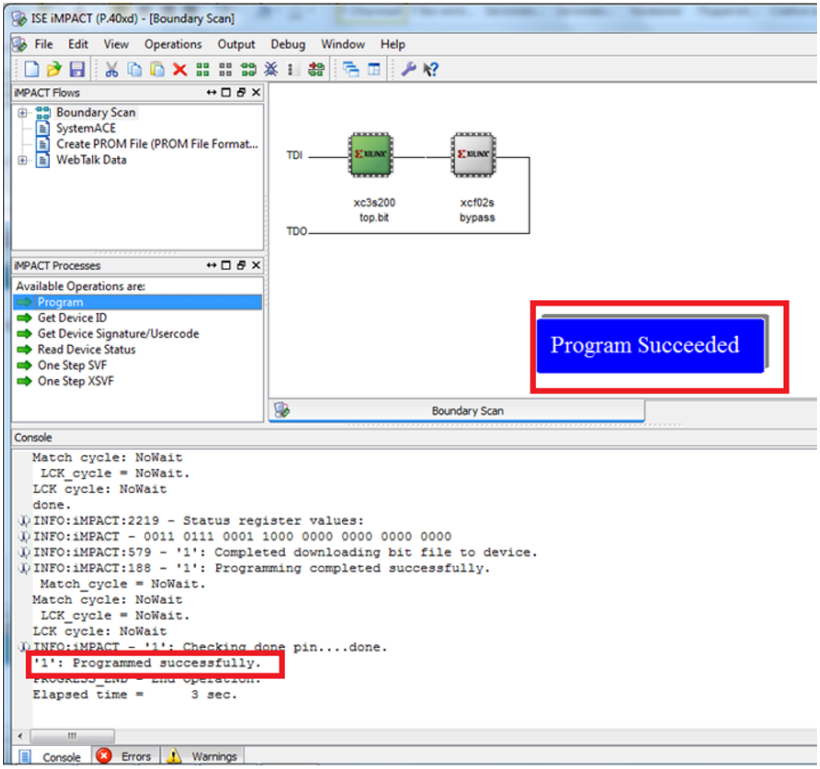


Рис. 2.16. Тестирование работоспособности устройства

2.3. Задание на лабораторную работу

1. Разработать модель цифровой схемы на языках Verilog HDL и VHDL согласно выбранному варианту. Модель должна быть разработана с использованием иерархического описания.

2. Разработать тестовую оболочку для полученной модели.

3. Провести тестирование разработанной модели с использованием отладочного комплекта Spartan 3 Starter Kit.

Варианты заданий.

Задание необходимо выполнить в соответствии с вариантом, выбранным для первой лабораторной работы.

2.4. Требования к отчету по лабораторной работе

Отчет должен содержать следующую информацию по лабораторной работе:

– название работы, цель и задание, согласно варианту;

– код описания модели цифрового устройства;

– временные диаграммы работы полученной модели.

3. ЛАБОРАТОРНАЯ РАБОТА № 3
РАЗРАБОТКА ЦИФРОВЫХ УСТРОЙСТВ С ПАМЯТЬЮ

3.1. Цель работы

Целью данной лабораторной работы является получение навыков разработки и тестирования цифровых устройств с элементами памяти (D-Триггерами).

3.2. Краткие теоретические сведения

При проектировании цифровых устройств широко используются D-триггеры – специальные логические элементы, которые способны запоминать информацию, поданную им на вход. Каждый такой триггер может запомнить один бит информации.

Условное графическое изображение D-Триггера представлено на рис. 3.1.

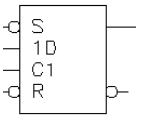


Рис. 3.1. D-триггер с асинхронным сбросом и установкой

Данные с входа записываются на выход D-триггера синхронно с фронтом тактового сигнала, приходящего на вход C. Таким образом, синхронно с каждым импульсом тактовой частоты схема переходит из одного устойчивого состояния в другое. Состояние всей схемы определяется текущими значениями в триггерах.

Для того чтобы поведение схемы было предсказуемым, необходимо произвести начальную установку значений триггеров. Для этого используются входы S(SET) и R(RESET).

Выходное значение триггера сбрасывается в 0 нулевым значением сигнала R, либо устанавливается в 1 нулевым значением сигнала S (в случае триггера с асинхронным сбросом – это происходит независимо от текущего состояния тактового входа C).

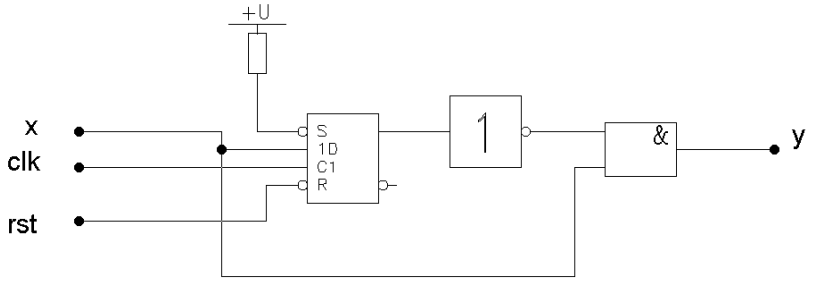
3.3. Задание на лабораторную работу

1. Разработать модели цифровой схемы на языках Verilog HDL и VHDL согласно выбранному варианту.

2. Разработать тестовую оболочку для полученных моделей. При тестировании произвести начальную установку триггеров схемы. Отобразить временные диаграммы сигналов на всех точках схемы, отмеченных в задании (y, y1, y2 и т.д.).

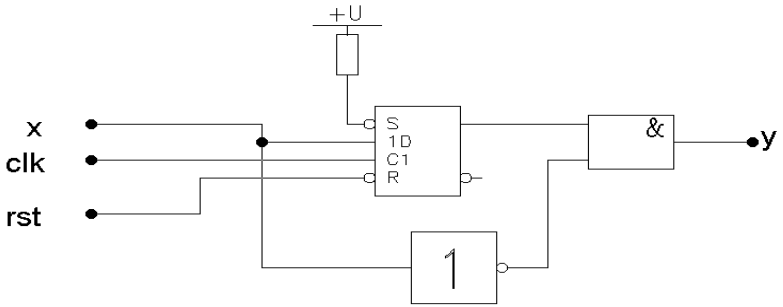
Варианты заданий.

Вариант 1



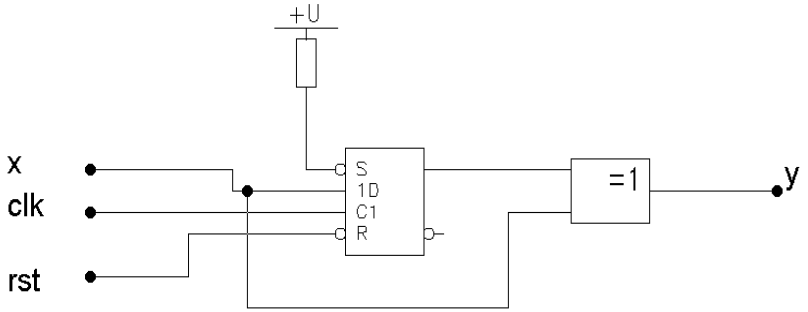
Примечание: на временной диаграмме требуется отобразить реакцию данной цифровой схемы на импульсный сигнал, поданный на вход X. Длительность импульса данного сигнала должна быть не менее двух периодов тактовой частоты, использованной для тактирования схемы.

Вариант 2



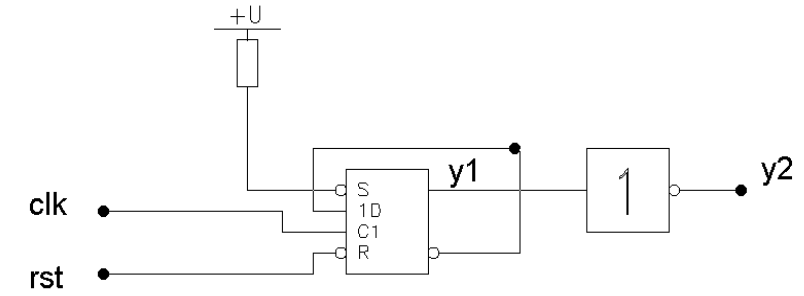
Примечание: на временной диаграмме требуется отобразить реакцию данной цифровой схемы на импульсный сигнал, поданный на вход X. Длительность импульса данного сигнала должна быть не менее двух периодов тактовой частоты, использованной для тактирования схемы.

Вариант 3

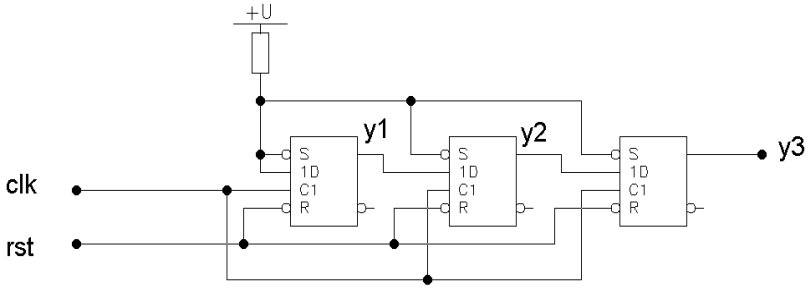


Примечание: на временной диаграмме требуется отобразить реакцию данной цифровой схемы на импульсный сигнал, поданный на вход X. Длительность импульса данного сигнала должна быть не менее трех периодов тактовой частоты, использованной для тактирования схемы.

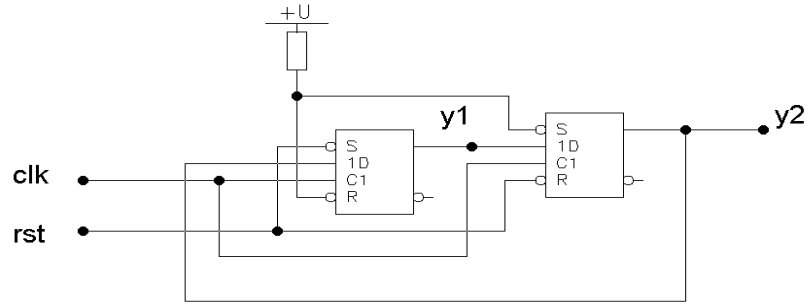
Вариант 4



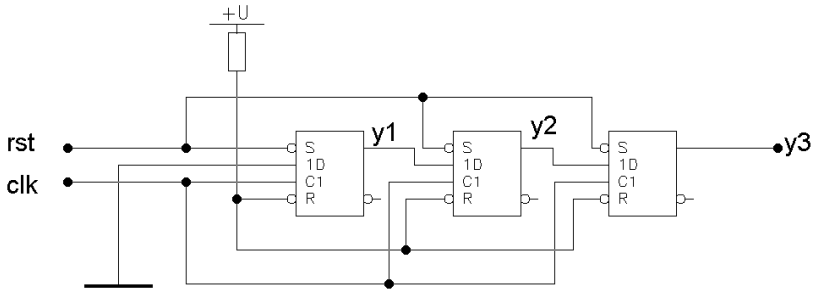
Вариант 5



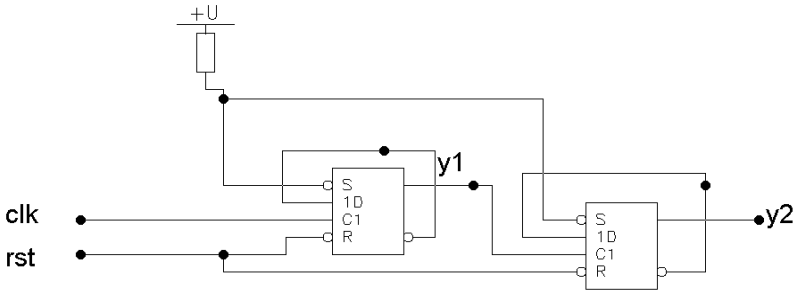
Вариант 6



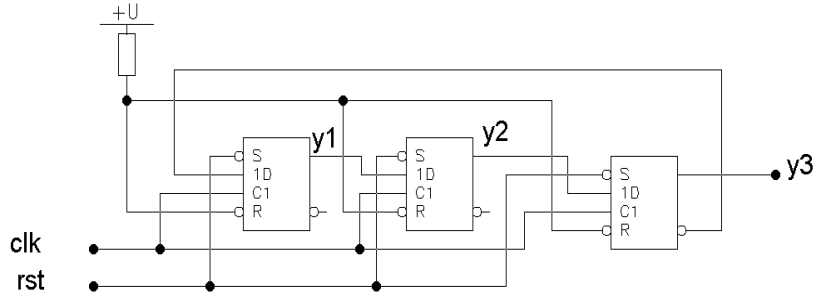
Вариант 7



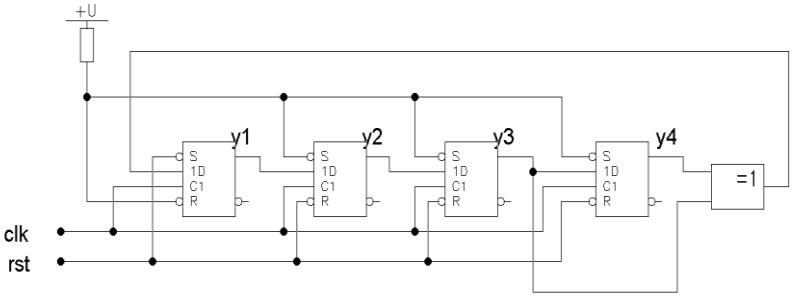
Вариант 8



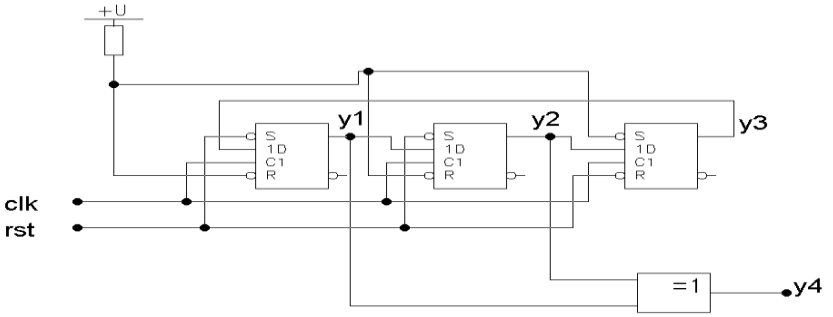
Вариант 9



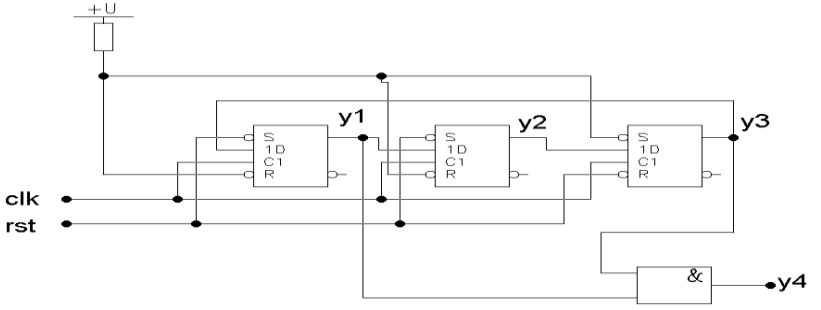
Вариант 10



Вариант 11



Вариант 12



3.4. Требования к отчету по лабораторной работе

Отчет должен содержать следующую информацию по лабораторной работе:

– название работы, цель и задание, согласно варианту;

– код описания модели цифрового устройства;

– временные диаграммы работы разработанной модели.

4. ЛАБОРАТОРНАЯ РАБОТА № 4
РАЗРАБОТКА ЦИФРОВЫХ УСТРОЙСТВ НА ОСНОВЕ КОНЕЧНЫХ АВТОМАТОВ

4.1. Цель работы

Целью данной лабораторной работы является получение навыков разработки и тестирования конечных автоматов.

4.2. Краткие теоретические сведения

Конечные автоматы широко используются при разработке цифровых устройств. Конечный автомат можно представить кортежем A = <A, B, C, δ, λ> , где

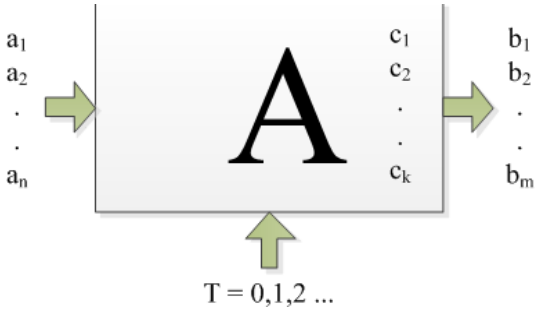


Рис. 4.1. Структурная схема автомата

1. {A} –множество значений на входах автомата.
2. {B} – множество значений на выходах автомата.
3. {C} – а множество, которое представляет внутреннее состояние автомата
4. $\delta = X \times Z \rightarrow Z$ –функции переходов автомата, однозначно определяющие состояние S_t, в которое переходит автомат из состояния S_{t-1}.

5. $\lambda = X \times Z \rightarrow Y$ – функции выходов, они определяют, что находится на выходе автомата в зависимости от входов и внутреннего состояния.

Автомат функционирует дискретно по времени, то есть значения входов, выходов и внутреннее состояние автомата изменяются в дискретные моменты времени.

Выделяют 2 типа автоматов:

Автомат Мура — конечный автомат, выходное значение сигнала в котором зависит лишь от текущего состояния данного автомата, и не зависит напрямую от входных значений.

Автомат Мили — конечный автомат, выходная последовательность которого (в отличие от автомата Мура) зависит от состояния автомата и входных сигналов.

1. Автомат Мили (рис. 4.2.). Описывается системой уравнений:

$$c(t) = \delta(a(t), c(t-1));$$
$$b(t) = \lambda(a(t), c(t-1)).$$

2. Автомат Мура (рис. 4.3.). Описывается системой уравнений:

$$c(t) = \delta(a(t), c(t-1));$$
$$b(t) = \lambda(a(t), c(t)).$$

Граф автомата – это ориентированный связный граф, вершины которого символизируют внутренние состояния автомата, а дуги – переходы из одного состояния в другое.

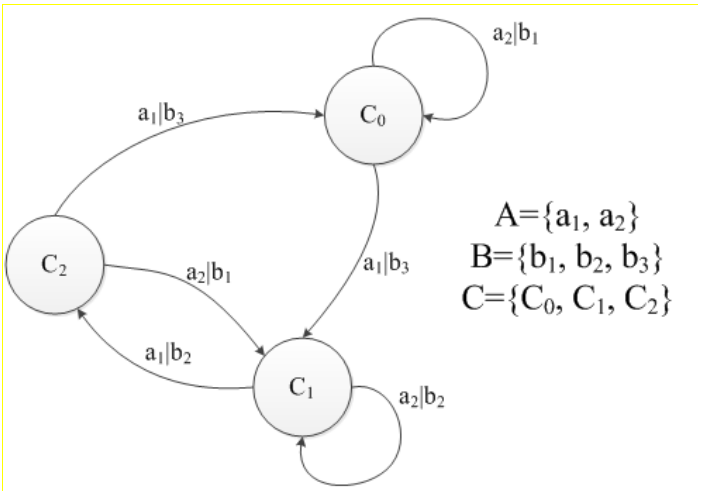


Рис. 4.2. Граф автомата Мили

Для графа Мили на дугах указываются входные и выходные буквы. Выходные буквы пишутся над дугами, символизируя то, что выходное состояние зависит от состояния автомата в предыдущий момент времени.

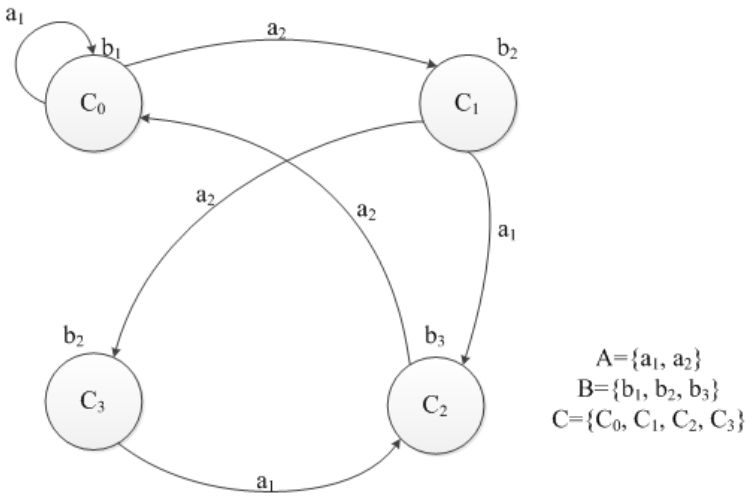


Рис. 4.3. Граф автомата Мура

Для графа автомата Мура на дугах записываются только входные буквы, выходные же указываются около вершин.

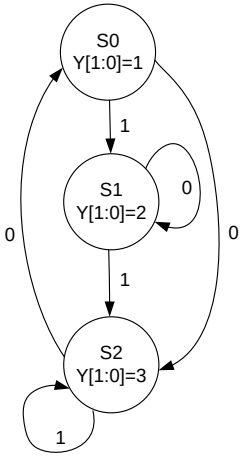
4.3. Задание на лабораторную работу

1. Разработать модель конечного автомата на языках Verilog HDL и VHDL-согласно выбранному варианту.

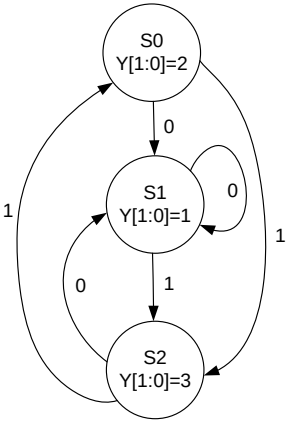
2. Разработать тестовую оболочку, обеспечивающую исчерпывающее тестирование полученной модели (необходимо смоделировать все возможные переходы из каждого состояния в каждое доступное).

Варианты заданий

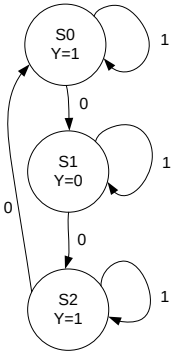
Вариант 1
Начальное состояние: S2



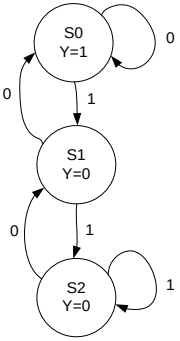
Вариант 2
Начальное состояние: S1



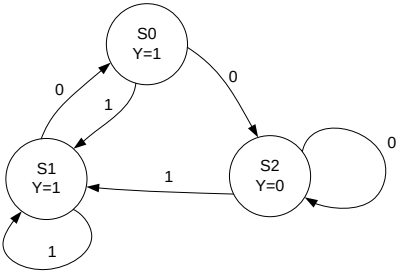
Вариант 3
Начальное состояние: S0



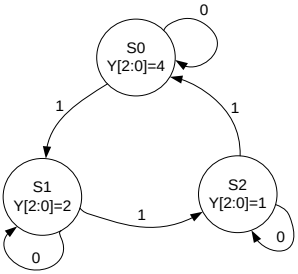
Вариант 4
Начальное состояние: S1



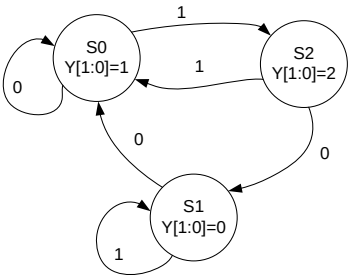
Вариант 5
Начальное состояние: S2



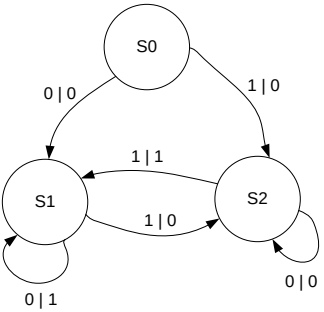
Вариант 6
Начальное состояние: S0



Вариант 7
Начальное состояние: S1



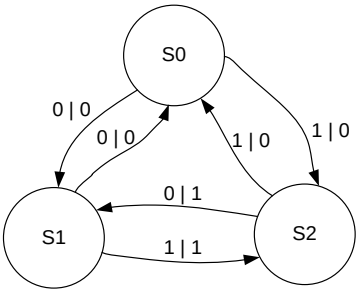
Вариант 8
Начальное состояние: S



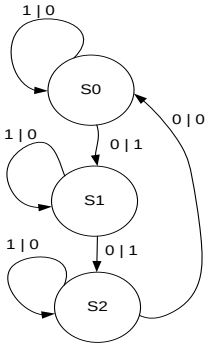
Вариант 9

Вариант 10

Начальное состояние: S1

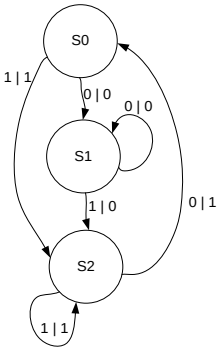


Начальное состояние: S1



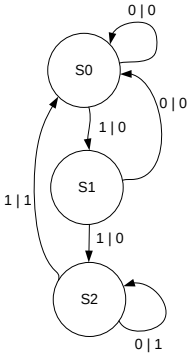
Вариант 11

Начальное состояние: S0



Вариант 12

Начальное состояние: S2



4.4. Требования к отчету по лабораторной работе

Отчет должен содержать следующую информацию по лабораторной работе:

– название работы, цель и задание, согласно варианту;

– код описания модели цифрового устройства;

– временные диаграммы работы разработанной модели.

5. ЛАБОРАТОРНАЯ РАБОТА № 5 РАЗРАБОТКА ПРОТОТИПА ЦИФРОВОГО УСТРОЙСТВА С ИСПОЛЬЗОВАНИЕМ ОТЛАДОЧНОГО КОМПЛЕКТА SPARTAN 3 STARTER KIT

5.1. Цель работы

Целью данной лабораторной работы является получение навыков разработки, тестирования и создания прототипов реальных цифровых устройств с использованием отладочных плат.

5.2. Краткие теоретические сведения

Выполнение каждого из вариантов заданий на данную лабораторную работу требует вывода информации на 7-сегментные индикаторы. В составе отладочной платы Spartan 3 Starter Kit имеется четыре светодиодных 7-сегментных индикатора. Каждый индикатор также содержит отдельный сегмент для отображения точки (рис. 5.1).

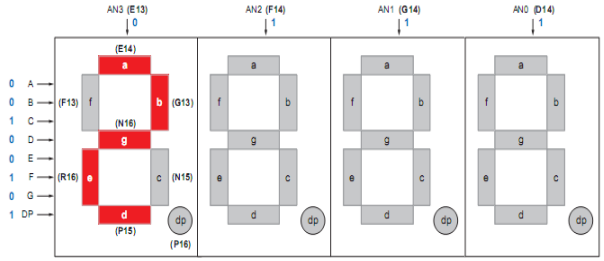


Рис. 5.1. Семисегментные индикаторы

В табл. 5.1 представлены выводы ПЛИС, подключенные к сегментам A-G, и к сегменту DP, отвечающему за отображение точки.

Таблица 5.1

Соответствие выводов ПЛИС сегментам индикатора

Сегмент	вывод ПЛИС
A	E14
B	G13
C	N15
D	P15
E	R16
F	F13
G	N16
DP	P16

В табл. 5.2 представлены выводы ПЛИС, к которым подключены дисплеи AN3 - AN0.

Таблица 5.2

Соответствия выводов ПЛИС выводам, контролирующим включение дисплея

Дисплей	AN3	AN2	AN1	AN0
Вывод ПЛИС	E13	F14	G14	D14

В табл. 5.3 представлены сигналы, которые необходимо подать на сегменты A- G, чтобы отобразить необходимое число от 0 до 9.

Таблица 5.3

Символы дисплея и соответствующие им значения сегментов

Символ	a	b	c	d	e	f	g
0	0	0	0	0	0	0	1
1	1	0	0	1	1	1	1
2	0	0	1	0	0	1	0
3	0	0	0	0	1	1	0
4	1	0	0	1	1	0	0
5	0	1	0	0	1	0	0
6	0	1	0	0	0	0	0
7	0	0	0	1	1	1	1
8	0	0	0	0	0	0	0
9	0	0	0	0	1	0	0

Таким образом, например, чтобы отобразить на первом индикаторе цифру 2 (как изображено на рис. 5.1), необходимо на вывод E13 подать сигнал низкого уровня, а остальные дисплеи «погасить» путем подачи на выводы F14, G14, D14 сигнала высокого уровня. Затем на выводы, соответствующие сегментам a, b, g, e, d, нужно подать сигналы низкого уровня, а на f, C и DP – высокого.

Для отображения сразу нескольких цифр их необходимо отображать на индикаторах поочередно.

Временная диаграмма отображения данных на четырех дисплеях индикатора представлена на рис. 5.2.

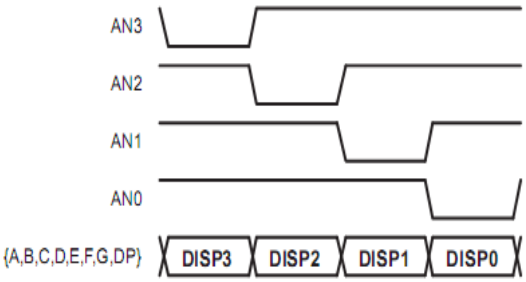


Рис. 5.2. Отображение данных на 4 дисплеях

Для тактирования схемы отображения данных на нескольких дисплеях необходимо использовать частоту порядка 500 Гц. На отладочной плате Spartan 3 Starter Kit установлен кварцевый генератор для тактирования ПЛИС. Тактовая частота генератора - 50МГц. Тактирование ПЛИС осуществляется через вывод T9 [3]. Для получения частоты 500 Гц можно воспользоваться простой схемой делителя частоты на основе счетчика и компаратора. Пример реализации такого делителя частоты используется в составе модуля, представленного в приложении к данной лабораторной работе.

Для отображения каждого из разрядов десятичного числа на нескольких 7-сегментных необходимо получить остаток от деления числа 10. Так как операция деления в

языках описания аппаратуры не является синтезируемой (может использоваться только для тестирования), для её осуществления можно воспользоваться специальным IP-ядром (intellectual property).

Рассмотрим процедуру добавления в проект IP-ядра для реализации операции деления чисел. Выберем пункт меню Project -> New Source. В открывшемся окне, выберем IP (Core generator & Architecture Wizard). Зададим имя добавляемого IP-ядра и нажмем Next (рис. 5.3).

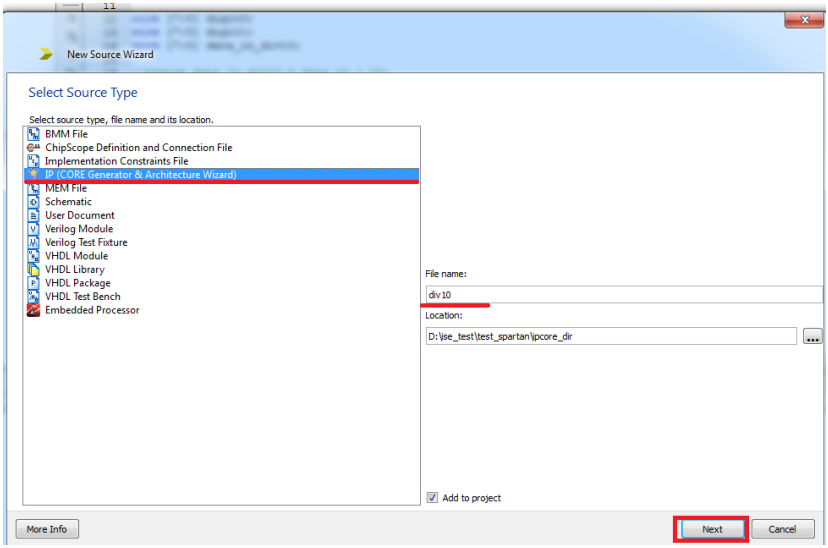


Рис. 5.3. Добавление IP ядра в проект

Из списка доступных IP-ядер, нужно выбирать Divider Generator и затем нажать Next (рис. 5.4).

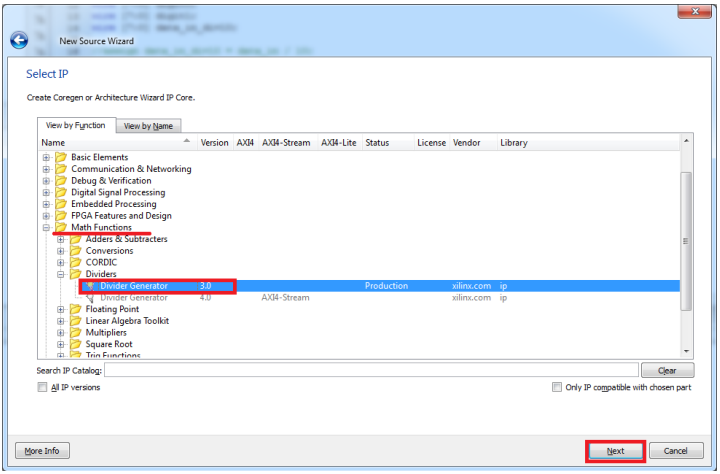


Рис. 5.4. Выбор Divider Generator

В открывшемся окне, ознакомьтесь со сводной информацией о добавляемом в проект IP-ядре, и нажмите Finish. После чего откроется окно, представленное на рис. 5.5.

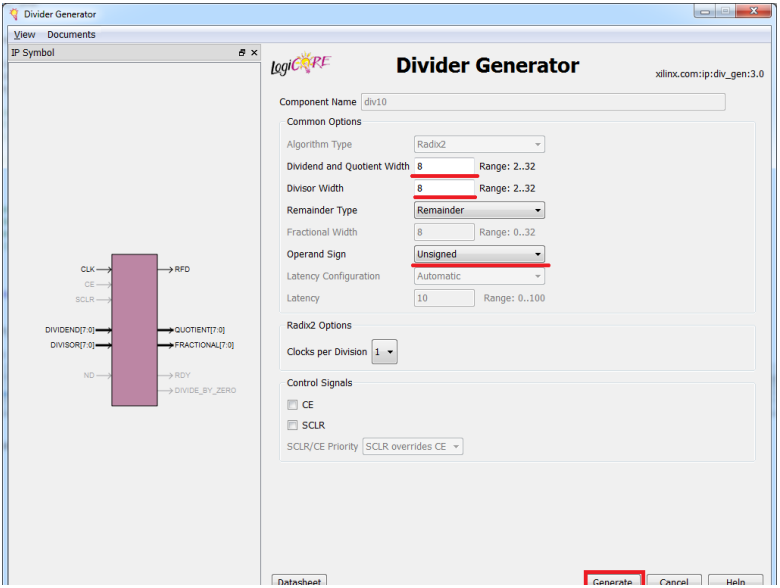


Рис. 5.5. Задание параметров блока делителя

В открывшемся окне задайте параметры операндов блока делителя. Затем, нажмите Generate и дождитесь окончания операции.

Пример использования данного IP-ядра для вывода десятичных чисел от 0 до 99 на 7-ми сегментный индикатор представлен в приложении к данной лабораторной работе.

Некоторые варианты задания предполагают выполнение действий по нажатию кнопки. При выполнении заданий по данным вариантам, для решения проблемы дребезга контактов, можно воспользоваться модулем, представленным ниже. Сигнал, полученный с кнопки, нужно подать на вход din. Очищенный от дребезга контактов сигнал можно получить с выхода dout. Также можно разработать свой модуль для фильтрации дребезга контактов, аналогичный по функционалу.

```
module btn_filter(
   input clk,
   input din,
   output dout
   );
   reg  prev_data;
   reg  [2:0] cntr = 0;
   reg  data_out = 0;
   always @ (posedge clk)
   begin
      prev_data <= din;
      if ( din == prev_data )
         begin
            case(cntr)
               3'b000 : cntr <= 3'b001;
               3'b001 : cntr <= 3'b010;
               3'b010 : cntr <= 3'b011;
               3'b011 : cntr <= 3'b100;
               3'b100 :
               begin
                  data_out    <= din;
```

```
                cntr       <= 3'b100;
              end
              default : cntr <= 0;
            endcase;
          end
        else
          begin
            cntr    <= 0;
          end
    end
    assign dout = data_out;
endmodule
```

Для тактирования данного модуля необходимо использовать медленную тактовую частоту (порядка 10-100 Гц).

5.3. Задание на лабораторную работу

1. Разработать модель цифрового устройства на языках Verilog HDL и VHDL согласно выбранному варианту.

2. Провести функциональное тестирование разработанной модели с использованием отладочного комплекта Spartan 3 Starter Kit.

Варианты заданий

Вариант 1. Разработать устройство, позволяющее считать количество нажатий на кнопку и отображать их на 7-сегментном индикаторе. Предусмотреть возможность сброса количества нажатий.

Вариант 2. Разработать устройство, позволяющее рассчитать сумму двух 4-х битных чисел. Числа задаются с помощью 2-х позиционных переключателей. Результат вывести на 7-сегментный индикатор.

Вариант 3. Разработать устройство, позволяющее подсчитать количество единиц в 8-битном двоичном числе.

Число задается с помощью. 2-х позиционных переключателей. Результат вывести на 7-сегментный индикатор.

Вариант 4. Разработать устройство, функционирующее следующим образом: после одного нажатия на кнопку светодиодный индикатор включается, затем после двух нажатий индикатор выключается. После трех нажатий индикатор снова включается и так далее. Количество нажатий кнопки, требуемых для смены состояния светодиода вывести на 7-сегментный индикатор. Предусмотреть возможность сброса количества требуемых нажатий.

Вариант 5. Разработать устройство, реализующее 4-х битный счетчик кода Грея. Счет производится по нажатию на кнопку. Предусмотреть возможность сброса счетчика. Результат вывести в двоичном виде на светодиодные индикаторы, а также в десятичном виде на 7-сегментные индикаторы.

Вариант 6. Разработать устройство, реализующее 8-ми битный циклический сдвиговый регистр. Направление сдвига задается с помощью 2-х позиционного переключателя. Предусмотреть возможность сброса регистра в начальное состояние (00000001). Результат вывести в двоичном виде на светодиодные индикаторы и также в десятичном виде на 7-сегментные индикаторы.

Вариант 7. Разработать устройство, позволяющее рассчитать произведение двух 3-х битных чисел. Числа задаются с помощью. 2-х позиционных переключателей. Результат вывести на 7-сегментные индикаторы.

Вариант 8. Разработать устройство, формирующее 8-битное псевдослучайное число на основе генератора m-последовательности. Каждое новое число генерируется при нажатии на кнопку. Результат вывести в двоичном виде на светодиодные индикаторы, а также в десятичном виде на 7-сегментные индикаторы.

Вариант 9. Разработать устройство, представляющее собой счетчик, позволяющий вести прямой и обратный счет количества нажатий на кнопку (счет ведется от 0 до 9 или от 9

до 0). Режим работы счетчика задается с помощью 2-х позиционного переключателя. Результат вывести на 7-сегментный индикатор.

Вариант 10. Разработать устройство, позволяющее рассчитать сумму двух 5-битных двоичных чисел. Числа задаются с помощью 2-х позиционных переключателей в 2 этапа: по первому нажатию кнопки запоминается первый операнд, по второму нажатию запоминается второй операнд и производится вычисление. Результат выводится на 7-сегментный индикатор.

Вариант 11. Разработать устройство, позволяющее подсчитать количество одновременно нажатых кнопок на отладочной плате (не более 4). Результат вывести на 7-сегментный индикатор.

Вариант 12. Разработать устройство, позволяющее рассчитать сумму по модулю 2 (исключающее ИЛИ) двух 4-х битных чисел. Числа задаются с помощью 2-х позиционных переключателей. Результат вывести в двоичном виде на светодиодные индикаторы, а также в десятичном виде на 7-сегментные индикаторы.

Вариант 13. Разработать устройство позволяющее выполнить операции И, ИЛИ, исключающее ИЛИ с двумя 3-х битными числами. Числа и вид выполняемой операции задаются с помощью 2-х позиционных переключателей. Результат операции вывести в двоичном виде на светодиодные индикаторы, а также в десятичном виде на 7-сегментные индикаторы.

Вариант 14. Разработать устройство, представляющее собой счетчик с настраиваемым значением инкремента. При каждом нажатии на кнопку значение счетчика увеличивается на 1, 2 3, или 4. Значение инкремента задается с помощью с помощью 2-х позиционных переключателей. Предусмотреть возможность сброса счетчика. Результат вывести на 7-сегментный индикатор.

Вариант 15. Разработать устройство, позволяющее рассчитать разность двух 4-х битных чисел. Числа задаются с

помощью. 2-х позиционных переключателей. Для отображения знака числа, в случае отрицательного результата, использовать средний сегмент (сегмент g) третьего 7-сегментного индикатора. Результат вывести на 7-сегментный индикатор.

5.4. Требования к отчету по лабораторной работе

Отчет должен содержать следующую информацию по лабораторной работе:

- название работы, цель и задание, согласно варианту;
- код описания модели цифрового устройства.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Методические указания к выполнению лабораторных работ по дисциплине «Автоматизированное проектирование вычислительных систем» для студентов специальности 230101 «Вычислительные машины, комплексы, системы и сети» очной и ускоренной форм обучения / ГОУВПО «Воронежский государственный технический университет»; сост. В. Ф. Барабанов, И. Н. Дырда, Д. Ю. Скляднев. Воронеж, 2009. 49 с.

2. Методические указания к выполнению лабораторных работ «Основы проектирования цифровых устройств на языках VHDL и Verilog» по дисциплине «Основы автоматизированного проектирования электронных наноматериалов и 3d изделий микроэлектронной техники» / ГОУВПО «Воронежский государственный технический университет»; сост. В.Ф. Барабанов, С.Л. Подвальный, В.В. Сафронов. Воронеж, 2010. 113 с.

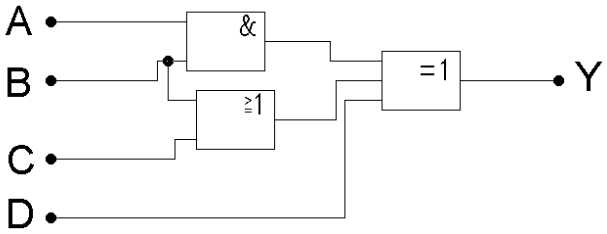
3. Spartan-3 FPGA Starter Kit Board User Guide(UG130 (v1.2) June 20, 2008).

4. Digital Design and Computer Architecture 2nd Edition / David Money Harris & Sarah L. Harris, July 2012, 712 с.

ПРИЛОЖЕНИЕ 1

Пример выполнения задания лабораторной работы 1

Разработать модель цифровой схемы



Код модели на языке Verilog HDL:

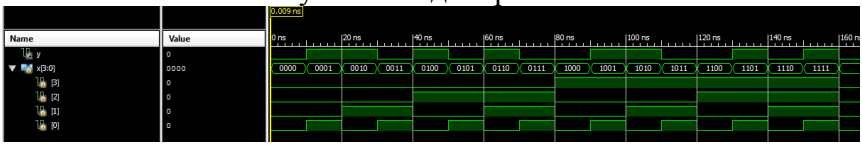
```
`timescale 1ns / 1ps

module top(input x, input y, input z, input c, output res );

        assign res = (x & y) ^ (y | z) ^ c;

endmodule
```

Код тестовой оболочки на языке Verilog HDL:

```
`timescale 1ns / 1ps
module source_laba1;
        reg [3:0] data;
        wire res;
        top uut (.x( data[3] ),  .y(data [2] ), .z(data [1] ), .c(data [0]
), .res( res ));
        integer i;

        initial
        begin
                data = 0;
                for (i = 0; i <= 15; i=i+1)
                begin
                        #10 data = data + 1;
```

```
                end
        end
endmodule
```

Результат моделирования:



Код модели на языке VHDL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity code_lab1 is
    Port ( x : in  STD_LOGIC; y : in  STD_LOGIC; z : in
STD_LOGIC; c : in  STD_LOGIC; res : out  STD_LOGIC);
end code_lab1;

architecture Behavioral of code_lab1 is
begin
        res <= (x and y ) xor ( y or z) xor c;
end Behavioral;
```

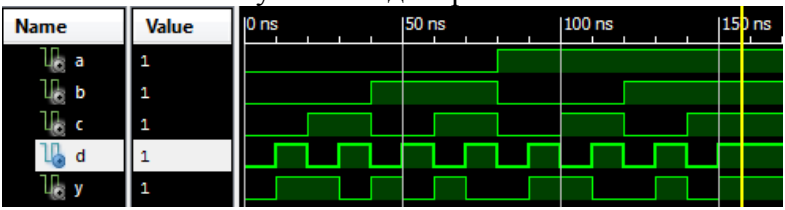
Код тестовой оболочки на языке VHDL:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY tb1 IS
END tb1;

ARCHITECTURE behavior OF tb1 IS
```

```
    COMPONENT code_lab1
    PORT(
         x : IN  std_logic;
         y : IN  std_logic;
         z : IN  std_logic;
         c : IN  std_logic;
         res : OUT  std_logic
        );
    END COMPONENT;

   signal x : std_logic := '0';
   signal y : std_logic := '0';
   signal z : std_logic := '0';
   signal c : std_logic := '0';
   signal res : std_logic;

   constant clock_period : time := 10 ns;

BEGIN

   uut: code_lab1 PORT MAP (
          x => x,
          y => y,
          z => z,
          c => c,
          res => res
        );

   stim_proc: process
   begin
            x <= '0'; y <= '0'; z <= '0'; c <= '0'; wait for 20 ns;
            x <= '0'; y <= '0'; z <= '1'; c <= '0'; wait for 20 ns;
            x <= '0'; y <= '0'; z <= '0'; c <= '1'; wait for 20 ns;
            x <= '0'; y <= '0'; z <= '1'; c <= '1'; wait for 20 ns;
            x <= '0'; y <= '1'; z <= '0'; c <= '0'; wait for 20 ns;
            x <= '0'; y <= '1'; z <= '0'; c <= '1'; wait for 20 ns;
```

```
        x <= '0'; y <= '1'; z <= '1'; c <= '0'; wait for 20 ns;
        x <= '0'; y <= '1'; z <= '1'; c <= '1'; wait for 20 ns;
        x <= '1'; y <= '0'; z <= '0'; c <= '0'; wait for 20 ns;
        x <= '1'; y <= '0'; z <= '0'; c <= '1'; wait for 20 ns;
        x <= '1'; y <= '0'; z <= '1'; c <= '0'; wait for 20 ns;
        x <= '1'; y <= '0'; z <= '1'; c <= '1'; wait for 20 ns;
        x <= '1'; y <= '1'; z <= '0'; c <= '0'; wait for 20 ns;
        x <= '1'; y <= '1'; z <= '0'; c <= '1'; wait for 20 ns;
        x <= '1'; y <= '1'; z <= '1'; c <= '0'; wait for 20 ns;
        x <= '1'; y <= '1'; z <= '1'; c <= '1'; wait for 20 ns;
    wait;
  end process;

END;
```

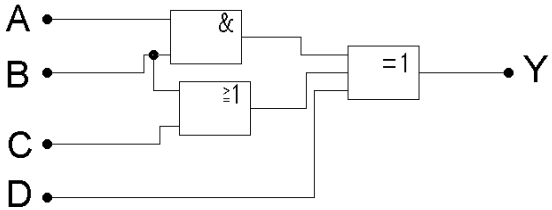
Результат моделирования:



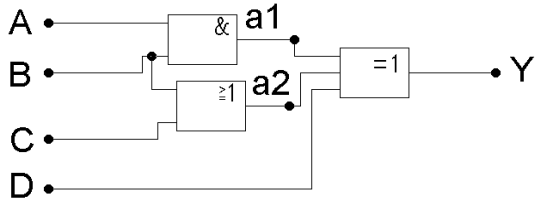
ПРИЛОЖЕНИЕ 2

Пример выполнения задания лабораторной работы 2

Разработать модель цифровой схемы, с использованием иерархического описания.



Для разработки модели с использованием иерархического описания выделим на схеме точки a1 и a2.



Впоследствии, при разработке иерархического описания модели необходимо создать проводники, к которым будут подключены эти промежуточные точки цифровой схемы.

Код модели на языке Verilog HDL:

```
`timescale 1ns / 1ps

module code_and2(input a, input b, output res);
        assign res = a & b;
endmodule

module code_or2(input a, input b, output res);
```

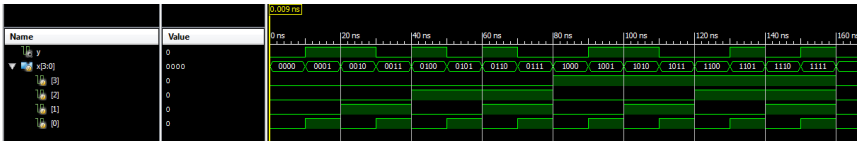
```
	assign res = a | b;
endmodule

module code_xor2(input a, input b, input c, output res);
	assign res = a ^ b ^ c;
endmodule

module top(input x, input y, input z, input c, output res);

wire x1, x2;

code_and2	code_and2_1	( .x1( x ), .x2( y ), .y( x1 ) );
code_or2	code_or2_1	( .x1( y ), .x2( z ), .y( x2 ) );
code_xor2	code_xor2_1	( .x1( x1 ), .x2( x2 ), .x3( c ), .y( res )
);

endmodule
```

Код тестовой оболочки на языке Verilog HDL:

```
`timescale 1ns / 1ps
module top_tb;
	reg [3:0] data;
	wire res;
	top uut (
		.a(data[3]),
		.b(data[2]),
		.c(data[1]),
		.d(data[0]),
		.y(res)
	);

integer i;
	initial
	begin
		data = 0;
```

```
            for (i = 0; i <= 15; i=i+1)
            begin
                #10 data = data + 1;
            end
        end
endmodule
```

Результат моделирования:



Код модели на языке VHDL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity code_AND2 is
    Port ( a : in  STD_LOGIC;
           b : in  STD_LOGIC;
           res : out  STD_LOGIC);
end code_AND2;

architecture Behavioral of code_AND2 is
begin
res <= a and b;
end Behavioral;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity code_OR2 is
    Port ( a : in  STD_LOGIC;
```

```
        b : in  STD_LOGIC;
        res : out  STD_LOGIC);
end code_OR2;

architecture Behavioral of code_OR2 is
begin
res <= a or b;
end Behavioral;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity my_XOR3 is
    Port ( a: in  STD_LOGIC;
        b: in  STD_LOGIC;
         c: in  STD_LOGIC;
        res: out  STD_LOGIC);
end my_XOR3;

architecture Behavioral of my_XOR3 is
begin
res <= a xor b xor c;
end Behavioral;


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity code_lab2 is
    Port (x: in  STD_LOGIC;
        y: in  STD_LOGIC;
        z: in  STD_LOGIC;
        c: in  STD_LOGIC;
        res: out  STD_LOGIC);
end code_lab2;

architecture Behavioral of code_lab2 is
```

```
component  code_AND2
    Port (a: in  STD_LOGIC;
          b: in  STD_LOGIC;
          res: out  STD_LOGIC);
end component ;

component  code_OR2
    Port (a: in  STD_LOGIC;
          b: in  STD_LOGIC;
          c: out  STD_LOGIC);
end component ;

component  my_XOR3
    Port (a: in  STD_LOGIC;
          b: in  STD_LOGIC;
           c: in  STD_LOGIC;
          res: out  STD_LOGIC);
end component ;

          signal x1: STD_LOGIC;
          signal x2: STD_LOGIC;

begin
c1: code_AND2 port map(
                          a => x,
                          b => y,
                          res => x1);
c2: code_OR2 port map(
                          a => y,
                          b => z,
                          res => x2);

c3: my_XOR3 port map(
                          a => x1,
                          b => x2,
```

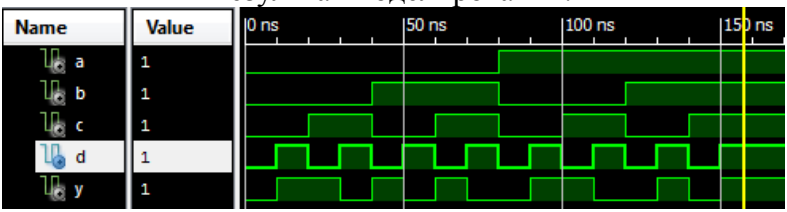
```
                    c => c,
                    res => res);

end Behavioral;
```

Код тестовой оболочки на языке VHDL:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY test_lab2 IS
END test_lab2;

ARCHITECTURE behavior OF test_lab2 IS

    COMPONENT code_lab2
    PORT(x: IN std_logic; y: IN std_logic; z: IN  std_logic; c: IN
std_logic; res: OUT  std_logic);
    END COMPONENT;

   signal x: std_logic := '0';
   signal y: std_logic := '0';
   signal z: std_logic := '0';
   signal c: std_logic := '0';

   signal res : std_logic;
   constant clock_period : time := 10 ns;

BEGIN

   uut: code_lab2 PORT MAP ( x => x, y => y, z => z, c => c, res
=> res);

   stim_proc: process
   begin
         x <= '0'; y <= '0'; z <= '0'; c <= '0'; wait for 20 ns;
```

```
        x <= '0'; y <= '0'; z <= '0'; c <= '1'; wait for 20 ns;
        x <= '0'; y <= '0'; z <= '1'; c <= '0'; wait for 20 ns;
        x <= '0'; y <= '0'; z <= '1'; c <= '1'; wait for 20 ns;
        x <= '0'; y <= '1'; z <= '0'; c <= '0'; wait for 20 ns;
        x <= '0'; y <= '1'; z <= '0'; c <= '1'; wait for 20 ns;
        x <= '0'; y <= '1'; z <= '1'; c <= '0'; wait for 20 ns;
        x <= '0'; y <= '1'; z <= '1'; c <= '1'; wait for 20 ns;
        x <= '1'; y <= '0'; z <= '0'; c <= '0'; wait for 20 ns;
        x <= '1'; y <= '0'; z <= '0'; c <= '1'; wait for 20 ns;
        x <= '1'; y <= '0'; z <= '1'; c <= '0'; wait for 20 ns;
        x <= '1'; y <= '0'; z <= '1'; c <= '1'; wait for 20 ns;
        x <= '1'; y <= '1'; z <= '0'; c <= '0'; wait for 20 ns;
        x <= '1'; y <= '1'; z <= '0'; c <= '1'; wait for 20 ns;
        x <= '1'; y <= '1'; z <= '1'; c <= '0'; wait for 20 ns;
        x <= '1'; y <= '1'; z <= '1'; c <= '1'; wait for 20 ns;
        wait;
    end process;

END;
```

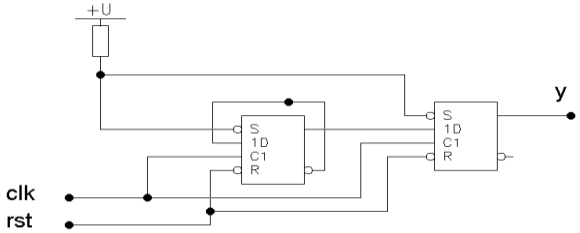
Результат моделирования:



ПРИЛОЖЕНИЕ 3

Пример выполнения задания лабораторной работы 3

Разработать модель цифровой схемы:



Код модели на языке Verilog HDL[4]:

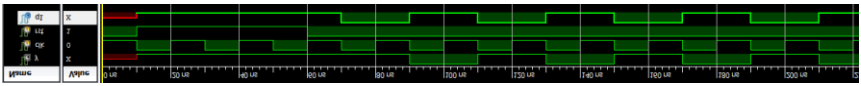
```
`timescale 1ns / 1ps

module dTrigger (input d, input clk, input reset, input s, output reg
q, output reg qn);

always @ ( posedge clk or negedge reset)
if (~reset)
        begin
                q  <= 1'b0;
                qn <= 1'b1;
        end
else if (~s)
        begin
                q        <= 1'b1;
                qn <= 1'b0;
        end
else
        begin
                q <= d;
                qn <= ~d;
        end
endmodule
```

```
module top(input clk, input reset, output res);

wire q1n, q1;

dTrigger  dTrigger_1 ( .d( q1n ), .clk( clk ), .reset( reset ), .s( 1'b1
), .q( q1 ), .qn( q1n ) );
dTrigger  dTrigger_2 ( .d( q1 ),   .clk( clk ), .reset( reset ), .s( 1'b1
), .q( reset ) );

endmodule
```

Код тестовой оболочки на языке Verilog HDL:

```
`timescale 1ns / 1ps

module top_tb;
	reg clk;
	reg reset;
	wire res;

	top uut (.clk(clk),.rst(reset),.y(res));

	initial begin
		clk = 0;
		reset = 1;

		#20 reset = 0;
		#100 reset = 1;
	end

	always
		clk = #20 ~clk;

endmodule
```

Результат моделирования:



Код модели на языке VHDL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity dTrigger is
port(d:in std_logic; clk:in std_logic; reset:in std_logic; s:in
std_logic; q:out std_logic;qn:out std_logic);
end entity;

architecture rtl of dTrigger is

begin
   process (clk, reset, s) begin
      if (reset = '0') then
         q  <= '0';
          qn <= '1';
      elsif (s = '0') then
         q  <= '1';
          qn <= '0';
      elsif (rising_edge(c)) then
         q <= d;
          qn  <= not d;
      end if;
   end process;
end architecture;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity code_lab3 is
   Port (clk : in  STD_LOGIC; reset : in  STD_LOGIC; res : out
STD_LOGIC);
end code_lab3;
```

```
architecture Behavioral of code_lab3 is

component dTrigger Port (d:in std_logic; clk:in  std_logic; reset:in
std_logic; s:in std_logic; q:out std_logic; qn:out std_logic);
end component ;
        signal q1n: STD_LOGIC;
        signal q1: STD_LOGIC;
begin
c1: dTrigger port map(
        d => q1n,
        clk => clk,
        reset => reset,
        s => '1',
        q => q1,
        qn => q1n );
c2: dTrigger port map(
        d => q1,
        clk => clk,
        reset => reset,
        s => '1',
        q => res);
end Behavioral;
```

Код тестовой оболочки на языке VHDL:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY tb2 IS
END tb2;

ARCHITECTURE behavior OF tb2 IS
   COMPONENT code_lab3
   PORT(clk : IN  std_logic; reset : IN  std_logic; res : OUT
std_logic);
```

```
    END COMPONENT;

   signal clk : std_logic := '0';
   signal reset : std_logic := '1';
   signal res : std_logic;
   constant clk_period : time := 20 ns;

BEGIN
   uut: code_lab3 PORT MAP (clk => clk, reset => reset, res =>
res);
   clk_process :process
   begin
         clk <= '0';
         wait for clk_period/2;
         clk <= '1';
         wait for clk_period/2;
   end process;

   stim_proc: process
   begin
     wait for clk_period*5;
         reset <= '0'; wait for clk_period*2;
         reset <= '1';
     wait;
   end process;
END;
```

Результат моделирования:

ПРИЛОЖЕНИЕ 4

Пример выполнения задания лабораторной работы 4

Код модели автомата Мили на языке Verilog HDL

```
module mili(input c, data, reset, output reg [1:0] res);

    reg [1:0] condition;
    parameter s0 = 0, s1 = 1, s2 = 2, s3 = 3;

    always @ (posedge c or posedge reset) begin
        if (reset)
            state <= S0;
        else
            case (state)
                S0:
                    if (data)
                    begin
                        condition <= s1;
                    end
                    else
                    begin
                        condition <= s1;
                    end
                S1:
                    if (data)
                    begin
                        condition <= s2;
                    end
                    else
                    begin
                        condition <= s1;
                    end
                S2:
```

```
                        if (data)
                        begin
                                condition <= s3;
                        end
                        else
                        begin
                                condition <= s1;
                        end
                S3:
                        if (data)
                        begin
                                condition <= s2;
                        end
                        else
                        begin
                                condition <= s3;
                        end
        endcase
end

always @ (condition or data)
begin
        case (state)
                s0:
                        if (data)
                        begin
                                res = 2'b00;
                        end
                        else
                        begin
                                res = 2'b10;
                        end
                s1:
                        if (data)
                        begin
                                res = 2'b01;
```

```
                end
                else
                begin
                        res = 2'b00;
                end
        s2:
                if (data)
                begin
                        res = 2'b10;
                end
                else
                begin
                        res = 2'b01;
                end
        s3:
                if (data)
                begin
                        res = 2'b11;
                end
                else
                begin
                        res = 2'b00;
                end
    endcase
end

endmodule
```

Код модели автомата Мура на языке Verilog HDL

```
module moore (input  c, data, reset, output reg [1:0] res);

    reg [1:0] conditional;
    parameter s0 = 0, s1 = 1, s2 = 2, s3 = 3;

    always @ (state) begin
```

```
                case (conditional)
                        s0:
                                res = 2'b01;
                        s1:
                                res = 2'b10;
                        s2:
                                res = 2'b11;
                        s3:
                                res = 2'b00;
                        default:
                                res = 2'b00;
                endcase
        end
        always @ (posedge c or posedge reset) begin
                if (reset)
                        conditional <= s0;
                else
                        case (conditional)
                                s0:
                                        state <= S1;
                                s1:
                                        if (data)
                                                state <= S2;
                                        else
                                                state <= S1;
                                s2:
                                        if (data)
                                                state <= S3;
                                        else
                                                state <= S1;
                                s3:
                                        if (data)
                                                state <= S2;
                                        else
                                                state <= S3;
                        endcase
        end

endmodule
```

ПРИЛОЖЕНИЕ 5

Пример использования IP ядра Divider Generator для вывода двоичного числа от 00 до 99 в десятичном виде на 7-ми сегментный индикатор отладочной платы Spartan 3 Starter Kit [4].

```
`timescale 1ns / 1ps
module top( input [7:0] data, input c, output reg [6:0] code, output
[3:0] stb);

wire clk_in;

reg [19:0] div;
reg c_out = 0;
always @ (posedge c)
begin
   if (div == 100000)
   begin
      div <= 0;
      c_out  <= ~c_out ;
   end
   else
      div <= div+1;
end

assign clk_in = c_out;

reg [1:0] sh_reg = 1;
wire [7:0] digit0;
wire [7:0] digit1;
wire [7:0] data_10;
```

```
reg  [6:0] digit0_code;
reg  [6:0] digit1_code;

//IP ядро для реализации операции деления.
div10 u_div10_1(.clk(clk_in), .dividend(data), .divisor(8'hA),
.quotient(data_10), .fractional());

div10 u_div10_2(.clk(clk_in), .dividend(data), .divisor(8'hA),
.quotient(), .fractional(digit0));

div10 u_div10_3(.clk(clk_in), .dividend(data_10), .divisor(8'hA),
.quotient(), .fractional(digit1));

always @ (posedge clk_in)
begin
        case(digit0)
                8'h00 : digit0_code <= 7'b0000001;
                8'h01 : digit0_code <= 7'b1001111;
                8'h02 : digit0_code <= 7'b0010010;
                8'h03 : digit0_code <= 7'b0000110;
                8'h04 : digit0_code <= 7'b1001100;
                8'h05 : digit0_code <= 7'b0100100;
                8'h06 : digit0_code <= 7'b0100000;
                8'h07 : digit0_code <= 7'b0001111;
                8'h08 : digit0_code <= 7'b0000000;
                8'h09 : digit0_code <= 7'b0000100;
                default : digit0_code <= 0;
        endcase

        case(digit1)
                8'h00 : digit1_code <= 7'b0000001;
                8'h01 : digit1_code <= 7'b1001111;
                8'h02 : digit1_code <= 7'b0010010;
```

```
                8'h03 : digit1_code <= 7'b0000110;
                8'h04 : digit1_code <= 7'b1001100;
                8'h05 : digit1_code <= 7'b0100100;
                8'h06 : digit1_code <= 7'b0100000;
                8'h07 : digit1_code <= 7'b0001111;
                8'h08 : digit1_code <= 7'b0000000;
                8'h09 : digit1_code <= 7'b0000100;
                default : digit1_code <= 0;
        endcase

        sh_reg = ~sh_reg;

        case(sh_reg)
                2'b01 : code_out <= digit0_code;
                2'b10 : code_out <= digit1_code;
                default : code_out <= 0;
        endcase

end

assign stb[1:0] = sh_reg;
assign stb[3:2] = 2'b11;

endmodule
```

Содержимое файла пользовательских ограничений

```
NET "code_out[0]" LOC = N16;
NET "code_out[1]" LOC = F13;
NET "code_out[2]" LOC = R16;
NET "code_out[3]" LOC = P15;
NET "code_out[4]" LOC = N15;
NET "code_out[5]" LOC = G13;
NET "code_out[6]" LOC = E14;
```

```
NET "stb[0]" LOC = G14;
NET "stb[1]" LOC = D14;
NET "stb[2]" LOC = E13;
NET "stb[3]" LOC = F14;

NET "c" LOC = T9;
NET "data[7]" LOC = K13;
NET "data[6]" LOC = K14;
NET "data[5]" LOC = J13;
NET "data[4]" LOC = J14;
NET "data[3]" LOC = H13;
NET "data[2]" LOC = H14;
NET "data[1]" LOC = G12;
NET "data[0]" LOC = F12;
```

ОГЛАВЛЕНИЕ

Учебное издание

Барабанов Владимир Федорович
Гребенникова Наталия Ивановна
Донских Дмитрий Николаевич
Коваленко Сергей Александрович

РАЗРАБОТКА И ПРОТОТИПИРОВАНИЕ ЦИФРОВЫХ УСТРОЙСТВ НА ЯЗЫКАХ VHDL И VERILOG

Учебно-методическое пособие

Редактор Е. А. Кусаинова

Подписано к изданию 23.11.2018.
Объем данных 3,7 Мб.

ФГБОУ ВО «Воронежский государственный технический университет»
394026 Воронеж, Московский просп., 14