МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Юго-Западный государственный университет»
(ЮЗГУ)

Кафедра космического приборостроения и средств связи

УТВЕРЖДАЮ:

Проректор по учебной работе

_____ О.Г. Локтионова

«_4_» _05_ 2024 г.

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ КОНСТРУИРОВАНИЯ ЭЛЕКТРОННЫХ СРЕДСТВ

Методические указания к лабораторным занятиям

Курск – 2024

УДК 621.382

Составители: Е.О. Брежнева

Рецензент

Доктор технических наук, профессор Чернецкая И. Е.

Информационные технологии конструирования электронных средств: методические указания к лабораторным занятиям по дисциплине «Информационные технологии конструирования электронных средств» / Юго-Зап. гос. ун-т.; сост.: Е.О. Брежнева Курск, 2024. 91 с.

Содержатся описание и методические рекомендации по изучению принципов работы в системах автоматизированного проектирования цифровых устройств с использованием языка VHDL.

Представлен порядок выполнения лабораторных работ.

Методические указания соответствуют требованиям программы, утвержденной учебно-методическим объединением по специальностям автоматики и электроники (УМО АЭ).

Предназначены для студентов направления подготовки бакалавров 11.03.02, 11.03.03.

Текст печатается в авторской редакции

Подписано в печать 07.05.24. Формат 60×84 1/16.

Усл. печ. л. 5,29. Уч.-изд. л. 4,79. Тираж 100 экз. Заказ 209. Бесплатно.

Юго-Западный государственный университет.

305040, г. Курск, ул. 50 лет Октября, 94

Оглавление

Цель и задачи лабораторных занятий

Изучение современных информационных технологий в области конструирования электронных средств.

Задачи:

1. Познакомиться со структурой и интерфейсом интегрированной средой автоматизированного проектирования ПЛИС.

2. Изучить принципы описания цифровых схем на ПЛИС и приобрести навыки работы с отладочной платой NI Digital Electronics FPGA Board.

3. Изучить методику программирования ПЛИС в среде автоматизированного проектирования.

4. Познакомиться с принципами реализации цифровых схем на языке описания аппаратуры VHDL.

5. Научиться выполнять моделирование схем с использованием современных информационных технологий.

Планируемые результаты обучения

В ходе выполнения лабораторной работы формируются следующие компетенции: ОПК-3, ОПК-4.

Обучающийся должен

знать:

- основы представления данных в системах автоматизированного проектирования;

- источники информации, посвященные вопросам проектирования на ПЛИС;

- основные средства проектирования на ПЛИС;

- описание, функциональные возможности, интерфейс, методы работы в системах проектирования ПЛИС;

- алгоритмы анализа схем в системах автоматизированного проектирования ПЛИС;

уметь:

- выбирать и использовать информационно-коммуникационные технологии при поиске информации;

- применять методы поиска, хранения, обработки, анализа и представления данных с использованием современных информационных технологий;

- находить и анализировать описания программных пакетов для работы с данными и САПР;

- выбирать соответствующую задачам проектирования САПР;

- создавать имитационные модели вычислительных систем;

- математически моделировать цифровые устройства;

владеть:

- навыками работы в сети интернет, с электронными библиотеками и техническими описаниями программных продуктов;

- навыками построения электрических принципиальных схем в САПР в соответствии с НД, оформления отчетной документации с применением информационных технологий;

- навыками выбора адекватной решаемой задачи САПР;

- навыками анализа функциональных возможностей и интерфейса САПР;

- синхронным, асинхронным, сквозным и событийным моделированием;

- навыками представления данных в современных программных пакетах (Excel);

- навыками построения электрических принципиальных схем в САПР;

- технологиями автоматизированного проектирования ЭС;

- навыками моделирования в САПР;

- навыками обработки и представления результатов работы.

Необходимые материально-техническое оборудование и материалы

- Circuit Design Suite 12.0/Система автоматизированного проектирования Xilinx ISE;

- Отладочная плата NI Digital Electronics FPGA Board**;**

- ПЛИС XC3S250E семейства Spartan3E;

- Libreoffice для операционной системы Windows;

- Антивирус Касперского (или ESETNOD);

- ПК (Processor i5-2500, RAM DDR3 4 GB, HDD 320 GB, DVD RW, TFT-монитор 24” 1920x1080).

Инструкция по технике безопасности

1 Общие требования охраны труда

1.1. К самостоятельной работе на персональном компьютере (далее по тексту - ПК) допускаются обучающиеся, прошедшие инструктаж по охране труда, обучение безопасным методам выполнения работ на ПК, проверку знаний требований охраны труда.

1.2. Обучающийся должен знать, что при выполнении работ на ПК на него могут оказывать воздействие следующие вредные и (или) опасные производственные факторы:

- повышенная температура поверхностей ПК;
- повышенная или пониженная температура воздуха рабочей зоны;
- повышенное значение напряжения в электрической цепи, замыкание;
- повышенный уровень статического электричества;
- повышенный уровень электромагнитных излучений;
- повышенная напряженность электрического поля;
- отсутствие или недостаток естественного света;
- недостаточная искусственная освещенность рабочей зоны;
- повышенная яркость света;
- повышенная контрастность;
- прямая и отраженная блесткость;
- зрительное напряжение;
- монотонность трудового процесса.

1.3. При проведении занятий, связанных с работой на ПК. обучающийся обязан:

- соблюдать порядок проведения занятия;

- поддерживать порядок на рабочем столе в течение всего учебного занятия;

- обо всех неисправностях ПК и электропитания немедленно сообщать преподавателю;

- соблюдать регламентированные перерывы в течение учебного занятия.

1.4. Лица, находящиеся в состоянии алкогольного и наркотического опьянения к занятиям, не допускаются.

1.5. Обучающиеся должны соблюдать правила пожарной безопасности, знать места расположения первичных средств пожаротушения и пути эвакуации при пожаре, место хранения аптечки.

1.6. Обучающиеся, невыполняющие и (или) нарушающие инструкцию по охране труда, отстраняются от занятия. Допуск к последующему занятию производится после проверки знаний правил охраны труда обучающегося, нарушившего и (или) не выполнившего инструкцию по охране труда, преподавателем. Для всех обучающихся проводится внеплановый инструктаж.

2 Требования охраны труда перед началом работы

2.1. Проветрить помещение, устранить повышенную подвижность воздуха (сквозняки) и т.д.

2.2. Отрегулировать освещенность на рабочем месте и убедиться в ее достаточности.

2.3. Проверить правильность расположения элементов компьютера в целях исключения неудобных поз и длительных напряжений тела.

2.4. Убрать с рабочего места посторонние предметы, освободить подходы к рабочему месту.

2.5. Убедиться путем внешнего осмотра:

- в исправности кабельных соединений, проводов, вилок, розеток;

- в том, что кабели электропитания ПК и другого оборудования (включая переноски и удлинители) находятся с тыльной стороны рабочего места;

- в максимальной удаленности от рабочего места источника бесперебойного питания для исключения вредного влияния его повышенных магнитных полей.

2.6. Убедиться в отсутствии пыли на экране монитора и клавиатуре, при необходимости протереть их специальной салфеткой.

3 Требования охраны труда во время работы

3.1. Подключение ПК к сети электропитания производить только имеющимися штатными сетевыми кабелями при закрытых кожухах и наличии заземления.

3.2. При включении ПК соблюдать следующую последовательность действий:

- включить сетевой фильтр;

- включить источник бесперебойного питания;

- включить системный блок;

- включить монитор.

3.3. После включения ПК и запуска программы:

- убедиться в отсутствии дрожания и мерцания изображения на экране монитора;

- установить яркость, контрастность, цвет и размер символов, фон экрана, обеспечивающие наиболее комфортное и четкое восприятие изображения.

3.4. Не допускать натягивания, скручивания, перегиба и пережима шнуров электропитания ПК, не допускать нахождения на них каких - либо предметов и соприкосновения их с нагретыми поверхностями.

3.5. Не допускать попадания влаги на поверхность персонального компьютера.

3.6. Не прикасаться к задней панели системного блока при включенном питании.

3.7. Не оставлять включенный ПК без наблюдения.

3.8. Не производить самостоятельно какие-либо виды ремонта ПК.

3.9. Не закрывать вентиляционные отверстия системного блока ПК.

3.10. Не допускать частых отключений и включений ПК в течение занятия.

3.11. Содержать свободными проходы к рабочему месту

3.12. Не загромождать рабочее место.

4 Требования охраны труда по окончании работы

4.1. Под руководством преподавателя отключить ПК от электросети сухими руками, держась за вилку штепсельного соединителя.

4.2. Привести в порядок рабочее место.

План проведения лабораторных работ

1. Знакомство с теоретическим материалом и алгоритмом работы.

2. Выполнение работы согласно алгоритму.

3. Оформление отчета по лабораторной работе.

4. Защита работы.

Теоретические сведения

Программируемые логические интегральные схемы

Программируемые логические интегральные схемы являются одними из самых перспективных элементов цифровой схемотехники. ПЛИС представляет собой микросхему, на кристалле которой расположено большое количество простых логических элементов. Изначально эти элементы не соединены между собой. С помощью электронных ключей, расположенных на этом же кристалле, можно соединять эти элементы в любом

порядке. Электронные ключи управляются памятью, в ячейки которой заносится код конфигурации цифровой схемы. Таким образом, записав в память ПЛИС определенные коды, можно получить цифровое устройство практически любой степени сложности. В отличие от микропроцессоров, в ПЛИС можно организовать алгоритмы цифровой обработки на аппаратном (схемном) уровне. Достоинствами технологии проектирования устройств на основе ПЛИС являются:

- минимальное время разработки схемы;
- одна ПЛИС может заменить сложное устройство, состоящее из множества обычных микросхем цифровой логики, благодаря чему отпадает необходимость в разработке и изготовлении сложных печатных плат;
- быстрое преобразование одной конфигурации цифровой схемы в другую (замена кода конфигурации схемы в памяти);
- для создания устройств на основе ПЛИС не требуется сложное технологическое производство. ПЛИС конфигурируется с помощью персонального компьютера на столе разработчика.

В зависимости от сложности решаемых задач, выбирается тот или иной тип ПЛИС. Если требуется небольшое число триггеров и эквивалентное число вентилей, то подходит CPLD или младшие серии FPGA. Для более крупных проектов практически всегда выбираются кристаллы FPGA большей емкости по числу триггеров и эквивалентных вентилей, иногда требуется использовать несколько кристаллов в проекте. Выбор конкретной

микросхемы целесообразно выполнять после завершения проектирования цифровой схемы.

Система автоматизированного проектирования

Существует множество сред, предназначенных для автоматизированного проектирования электронных средств, в том числен на ПЛИС, например, Circuit Design Suite 12.0.

Одним из мировых лидеров по производству ПЛИС является фирма Xilinx. Для создания цифровых устройств на основе своих изделий Xilinx разработала специальную программную среду Xilinx ISE. Эта среда позволяет:

- с помощью графического редактора ввести в память персонального компьютера электрическую схему;
- проверить работоспособность и исправить ошибки;
- определить параметры и характеристики разработанного устройства;
- промоделировать работу схемы;
- сформировать файл конфигурации для конкретного вида ПЛИС;
- загрузить полученный файл в память интегральной схемы.

Создание проекта

Работа в среде Xilinx ISE начинается с создания проекта. В первую очередь следует запустить программу. Ярлык для запуска может иметь название ISE Project Navigator или ISE Design Suite. Главное окно программы после запуска показано на рис. 1.

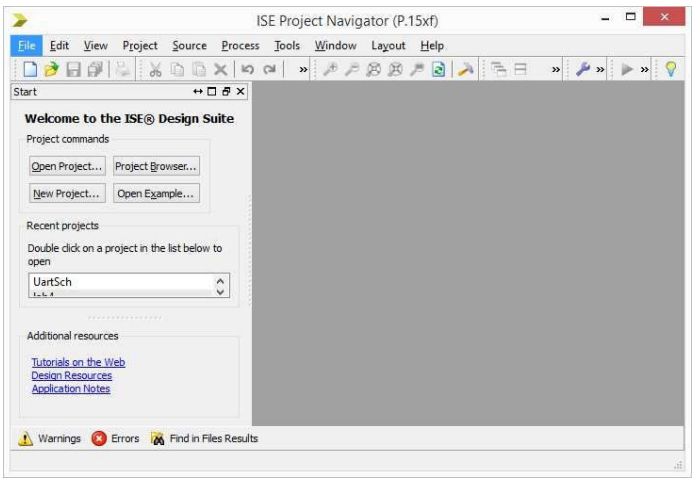


Рис. 1. Главное окно ISE Project Navigator

Для создания проекта необходимо выбрать в меню File пункт

New Project – это запустит мастер создания новых проектов (рис. 2).

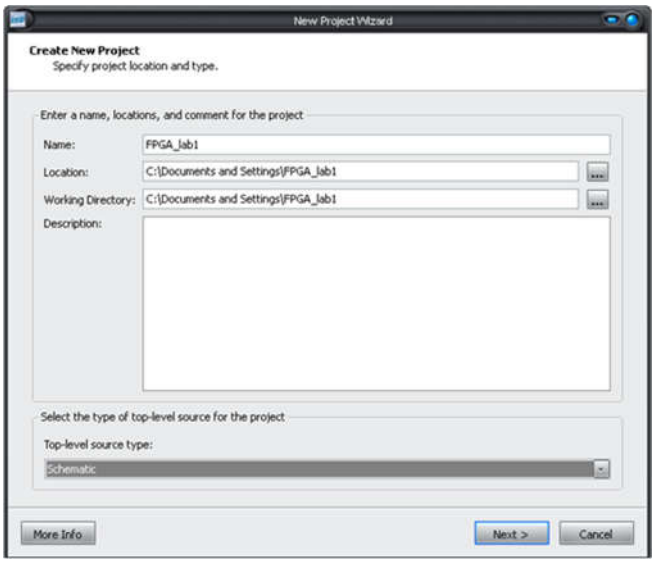


Рис. 2. Создание нового проекта

В открывшемся окне New Project Wizard в поле Name задаем название проекта (может быть любым, в данном случае – FPGA_lab1) и указываем папку для хранения файлов проекта в поле Project Location. **Внимание!** Путь к папке, имя папки и проекта не

должны содержать кириллицы.

Проекты для ПЛИС могут содержать множество файлов различного назначения, однако среди них всегда есть один файл, который является основным и содержит описание проектируемого цифрового устройства. Этот файл называется исходным файлом верхнего уровня. Поле Top-level source type указывает, файл какого типа будет выступать в роли файла верхнего уровня. Здесь необходимо упомянуть о возможных вариантах описания цифрового устройства.

Xilinx ISE позволяет использовать для описания цифровой схемы как графические схемы, так и описания на языках описания аппаратуры. Одно и то же устройство может быть как «нарисовано» из составных логических блоков, так и описано на HDL – разница в этом случае возникает лишь в удобстве для человека. Одним из возможных вариантов организации проекта для ПЛИС является файл верхнего уровня в виде схемы (Schematic), который выступает в роли структурной схемы, объединяющей достаточно крупные и независимые функциональные блоки, описанные с помощью HDL. Такой подход позволяет сочетать наглядность схемного представления и удобство HDL – описания.

Для данной лабораторной работы выбираем тип файла верхнего уровня *Schematic* и переходим к следующему этапу (кнопка *Next*).

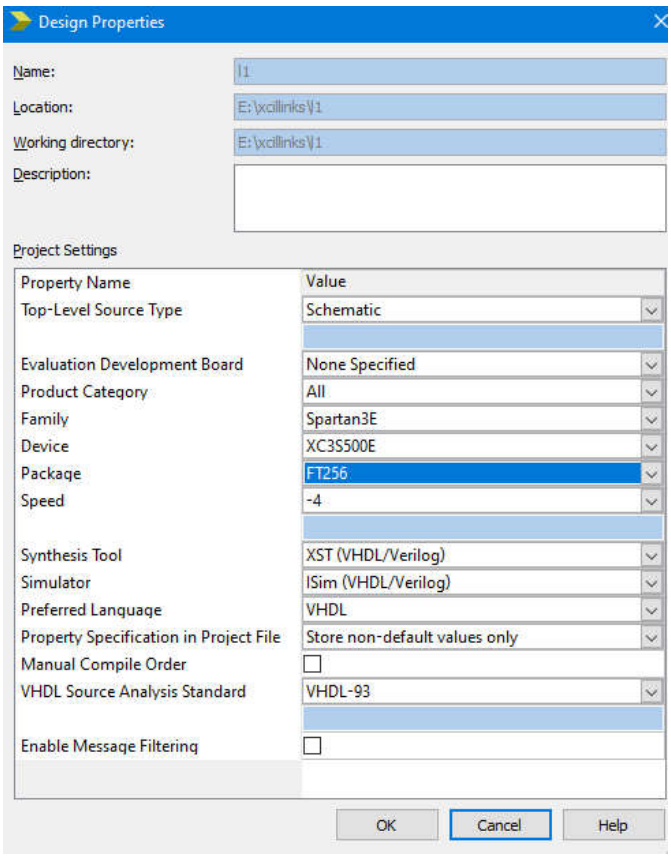


Рис. 3. Начальные настройки проекта

В появившемся окне Project Settings (рис. 3) указывается конкретная микросхема ПЛИС, для которой создается проект. В частности, выбирается семейство микросхем, наименование микросхемы, тип корпуса, в котором она размещается и рейтинг быстродействия. Эта лабораторная работа рассчитаны на использование ПЛИС XC3S250E семейства Spartan3E.

После нажатия на кнопку Next получаем полный список желаемых свойств проекта и нажимаем на кнопку Finish для окончательного подтверждения. После этого мы получаем пустой настроенный проект.

Проект в общем случае состоит из нескольких файлов, описывающих структуру цифрового устройства. Для просмотра

имеющихся файлов, их добавления и удаления используется вкладка Design (рис.4).

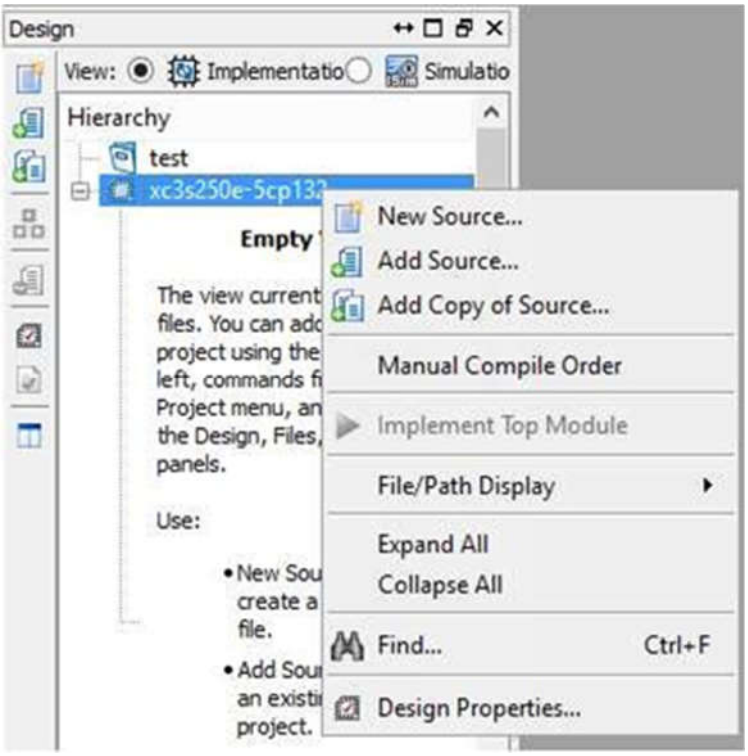


Рис. 4. Окно Design

Для дальнейшей работы в проект необходимо добавить файл верхнего уровня. Так как ранее был указан тип файла *Schematic*, необходимо добавить в проект файл схемного описания. Для этого вызывается контекстное меню (рис. 4), и по нажатию кнопки *New Source* появляется диалоговое окно (рис. 5), в котором необходимо выделить желаемый тип файла (*Schematic*) и задать его имя (в данном случае *main*, но имя может быть любым).

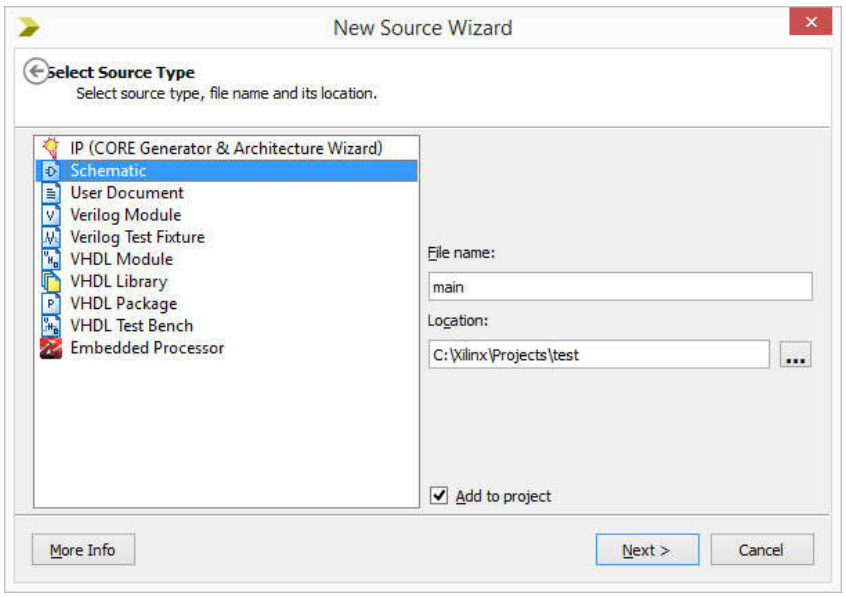


Рис. 5. Добавление файла в проект

После добавления файла автоматически открывается окно редактора принципиальных схем. При этом в окне *Design* в области окна *Hierarchy* появится созданный файл с расширением *.sch*. Данный проект будет пополняться различными файлами и модулями по мере их создания в дальнейшем.

Отладочная плата NI Digital Electronics FPGA Board

В качестве первого проекта будет рассмотрено создание простой комбинационной схемы, предназначенной для программирования ПЛИС, установленной на отладочной плате **NI Digital Electronics FPGA Board**. Внешний вид отладочной платы показан на рис. 6.

На плате установлена ПЛИС серии Spartan 3E, к которой подключено множество периферийных устройств, позволяющих

использовать плату в различных конфигурациях. Для данной работы будут использованы переключатели, кнопки и светодиоды, расположенные на плате.

Для программирования плату необходимо подключить к ПК с помощью USB-кабеля. Программирование может производиться как с помощью средств Xilinx ISE, так и сторонним ПО.

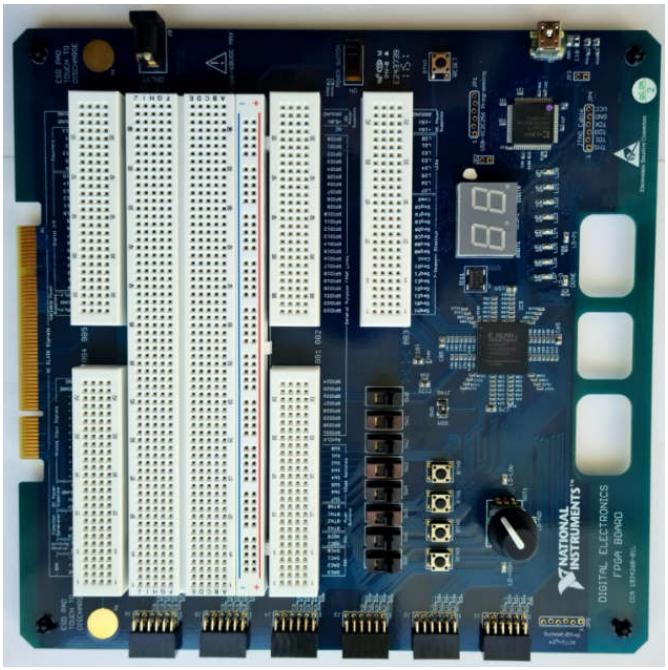
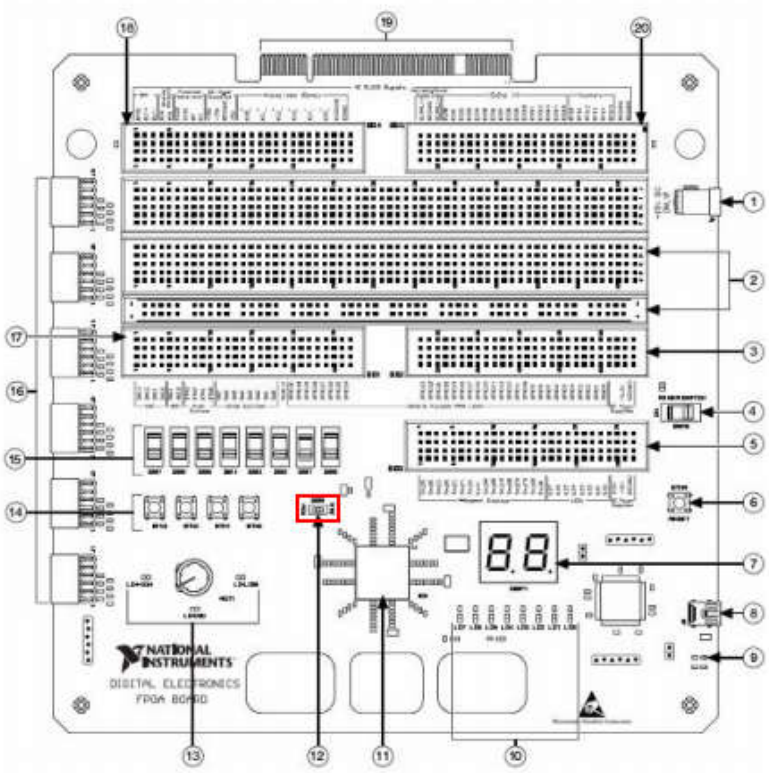


Рис. 6. Внешний вид платы NI Digital Electronics FPGA Board

На рисунке 7 представлена верхняя сторона платы с указанием назначений ее элементов.



1 Разъём подключения источника питания
2 Разъёмы общего назначения зоны макетирования
3 Сигнальный разъём BB2 зоны макетирования
4 Выключатель питания
5 Сигнальный разъём BB3 зоны макетирования
6 Кнопка сброса
7 Семисегментные индикаторы
8 Разъём USB
9 Светодиод LD-G
10 Светодиоды
11 ПЛИС
12 Переключатель SW 9
13 Вращающийся нажимной переключатель
14 Кнопки
15 Движковые переключатели
16 Разъёмы Digilent Pmod
17 Сигнальный разъём BB1 зоны макетирования
18 Сигнальный разъём BB4 зоны макетирования
19 Разъём для подключения к NI ELVIS
20 Сигнальный разъём BB5 зоны макетирования

Рис. 7. Верхняя сторона платы с указанием назначений ее элементов

Создание принципиальной схемы

Схема в Xilinx ISE может создаваться как из готовых библиотечных компонентов, так и из пользовательских компонентов, написанных на одном из языков HDL. Как и на

принципиальных схемах, связи между компонентами обозначаются проводниками или шинами. Если один проводник может соединять только отдельные выводы компонентов, то шина представляет собой группу проводников, к каждому из которых можно подключиться отдельно.

Порт – вывод схемы, который соответствует определенному выводу собственно микросхемы ПЛИС. То есть, можно сказать, что порты служат для «связи с внешним миром» - через них ПЛИС получает и передает сигналы по подключенным к микросхеме ПЛИС проводникам.

Разработку схемы удобнее всего начать с расположения необходимых компонентов на листе. Для ввода библиотечных элементов в схемотехническом редакторе используется панель Symbols (рис. 8), которая содержит элементы выбора: Categories, где можно необходимую группу библиотечных символов; Symbols, в которой отображается список символов из выбранной категории; Symbol Name Filter – данное поле предназначено для поиска по имени символа.

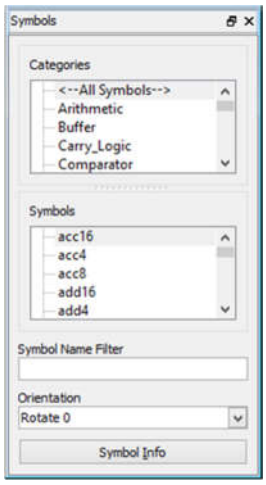


Рис. 8. Вкладка Symbols

Выберем категорию All Symbols и введём в окне Symbol Name Filter, имя элемента and (логическое И) – в окне выбора Symbols появится список элементов, названия которых начинаются на and. Необходим библиотечный символ and2 – двухвыводной элемент И. Выбираем данный символ и перемещаем символ на рабочее поле схемотехнического редактора. Чтобы отменить ввод следующего такого же символа или выбрать другую функцию, можно нажать на клавиатуре клавишу Esc или выбрать на панели инструментов символ с изображением стрелочки, выполняющий функцию указателя выбора.

Аналогично, на схеме располагаются все элементы. Приведем пример обозначения некоторых элементов библиотеки, которые понадобятся при выполнении лабораторной работы:

1. И-НЕ (nand);
2. ИЛИ (or);
3. ИЛИ-НЕ (nor);
4. Искл. ИЛИ (xor);
5. Искл. ИЛИ-НЕ (xnor).

Создание проводников и шин на схеме

Создание проводников и шин на схеме производится одним инструментом – Add Wire (может быть найден в меню Add→Wire). Каждое соединение на схеме входит в определенную сеть (net) с уникальным именем (чаще всего вида XLXN_??). Все проводники,

имеющие одно имя, считаются соединенными, даже если графически эта связь не нарисована. Это позволяет удобно соединять цепи в графическом редакторе, не рисуя длинных соединительных линий, загромождающих схему.

Для смены имени проводника можно либо зайти в контекстное меню, щелкнув на проводнике правой кнопкой мыши, и выбрать Rename selected net…, или воспользоваться инструментом Add Net Name (Add→Net name). Имя цепи указывается в левой панели во вкладке Options.

Шина представляет собой группу проводников и рисуется тем же инструментом, что и проводники. Любой проводник можно превратить в шину, если дать ему имя определенного вида, – точнее, имя проводника должно заканчиваться круглыми скобками, в которых через двоеточие указан интервал номеров для проводников шины (например, mybus(3:0)).

Для создания шины необходимо провести проводник, который будет преобразован в шину. Выделив проводник и нажав правую кнопку мыши (рис.9) переименовать проводник, присвоив ему имя, например, sw(7:0). Это означает, что шина носит имя sw, и в ней встречаются проводники с номерами от 7 до 0 – всего 8 проводников. К каждому из проводников можно обратиться отдельно по его номеру – например, sw(3) или sw(7).

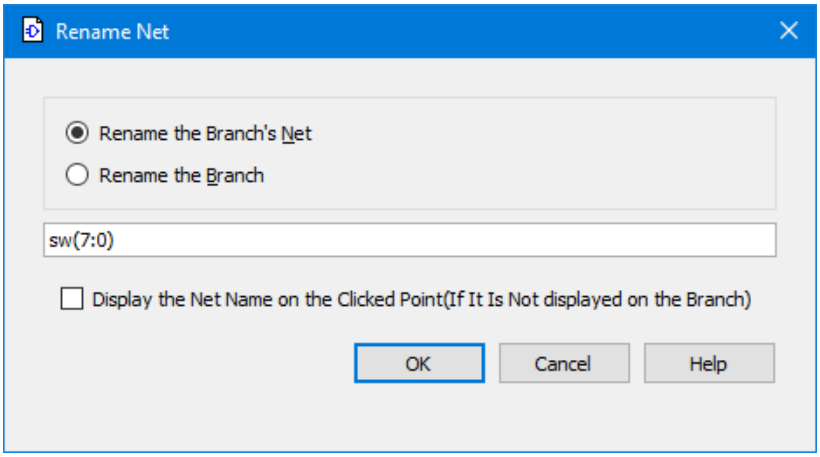


Рис. 9. Создание шины

Для подключения отдельного вывода к шине необходимо воспользоваться инструментом Add Bus Tap, находящимся слева на панели инструментов (рис. 10), для создания отвода от шины и переименовать подходящий к отводу проводник, присвоив ему имя нужного проводника шины.

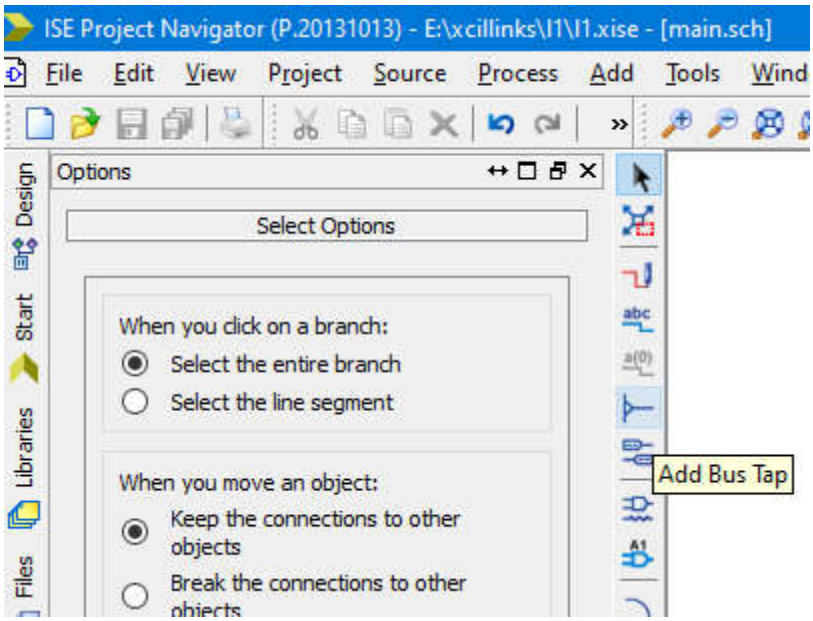


Рис. 10. Подключение вывода к шине

Затем необходимо добавить обозначения портов, чтобы показать, какие из проводников должны быть подключены к физическим выводам микросхемы ПЛИС. Для этого служит инструмент Add I/O Marker, находящийся также слева на панели инструментов.

Назначение выводов

После создания схемы следует провести привязку ее входов/выходов к выводам ПЛИС. На используемой плате установлено 8 переключателей и 4 кнопки, которые могут служить источниками входных сигналов для схемы, а также 8 светодиодов и четырехзнаковый 7-сегментный индикатор, позволяющие выводить на них информацию.

В данной лабораторной работе необходимо подключить переключатели к проводникам шины sw(7:0), кнопки к маркерам btn0…3, а выходы led (5:0) к светодиодам. Для того чтобы задать соответствие маркеров ввода/вывода схемы и физических выводов ПЛИС, необходимо создать новый конфигурационный файл.

С этой целью создается новый файл с расширением *.ucf, в котором описывается привязка выводов схемы к выводам кристалла. Для создания данного файла необходимо в окне Hierarchy навигатора проекта вызвать контекстное меню и выбрать пункт New Source (см. рис. 11). В списке возможных файлов для добавления необходимо выбрать Implementation Constraints File. Имя файла несущественно (в данном случае – param.ucf).

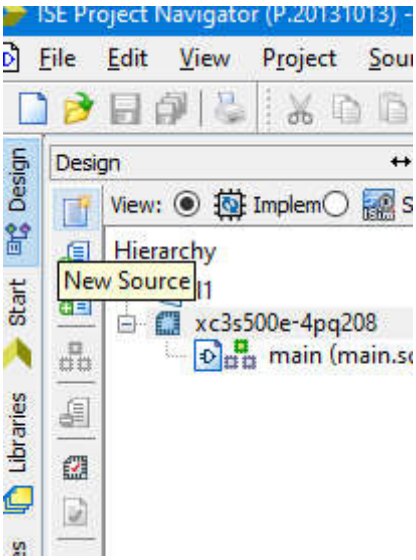


Рис. 11. Создание конфигурационного файла

Файлы *ucf* (*User Constraints Files*) содержат директивы, задающие необходимые временные и топологические ограничения для трассировщика (например, пользователь может указать максимальную частоту определенного сигнала, и для обеспечения данной частоты программа-транслятор будет вынуждена использовать более подходящие элементы). Кроме того, в данном файле может задаваться назначение выводов ПЛИС, то есть, какая цепь схемы (какой маркер ввода/вывода) должна подключаться к какому выводу корпуса ПЛИС. Выводы ПЛИС необходимо определять по документации на отладочную плату.

Ниже приведены параметры *UCF(.ucf)* файлов присвоенные различным аппаратным средствам отладочной платы *NI Digital Electronics FPGA Board.*

Движковые переключатели

В данном подпункте перечислены параметры UCF файла, присвоенные движковым переключателям SW0-SW7 В приведённых ниже параметрах Swx относится к цепи соответствующего движкового переключателя, LOC указывает расположение линии ПЛИС, IOSTANDARD указывает используемый стандарт линий ввода/вывода.

Net «SW0» LOC=»J11» | IOSTANDARD = LVCMOS33;
Net «SW1» LOC=»J12» | IOSTANDARD = LVCMOS33;
Net «SW2» LOC=»H16» | IOSTANDARD = LVCMOS33;
Net «SW3» LOC=»H13» | IOSTANDARD = LVCMOS33;
Net «SW4» LOC=»G12» | IOSTANDARD = LVCMOS33;
Net «SW5» LOC=»E14» | IOSTANDARD = LVCMOS33;
Net «SW6» LOC=»D16» | IOSTANDARD = LVCMOS33;
Net «SW7» LOC=»B16» | IOSTANDARD = LVCMOS33;

Кнопки

В данном подпункте перечислены параметры UCF файла, присвоенные кнопкам BTN0-BTN7. В приведённых ниже параметрах BTNx относится к цепи соответствующей кнопки, LOC указывает расположение линии ПЛИС, IOSTANDARD указывает используемый стандарт линий ввода/вывода.

Net “BTN0” LOC=”C13” | IOSTANDARD = LVCMOS33;

Net “BTN1” LOC=”D12” | IOSTANDARD = LVCMOS33;
Net “BTN2” LOC=”C12” | IOSTANDARD = LVCMOS33;
Net “BTN3” LOC=”C10” | IOSTANDARD = LVCMOS33;

Светодиоды

В данном подпункте перечислены параметры UCF файла, присвоенные светодиодам LED0-LED7. В приведённых ниже параметрах LEDx относится к цепи соответствующего светодиода, LOC указывает расположение линии ПЛИС, IOSTANDARD указывает используемый стандарт линий ввода/вывода, SLEW указывает скорость нарастания выходного напряжения, DRIVE указывает выходной ток источника тока, встроенного в ПЛИС в миллиамперах.

Net “LED0” LOC=”C11” | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;
Net “LED1” LOC=”D11” | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;
Net “LED2” LOC=”B11” | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;
Net “LED3” LOC=”A12” | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;
Net “LED4” LOC=”A13” | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;

Net “LED5” LOC=”B13” | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;
Net “LED6” LOC=”A14” | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;
Net “LED7” LOC=”B14” | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;

Семисегментные индикаторы

В данном подпункте перечислены параметры UCF файла, присвоенные семисегментным индикаторам. В приведённых ниже параметрах SEGxx относится к цепи соответствующего сегмента индикатора, COMx относится к относится к цепи анода соответствующей цифры индикатора. LOC указывает расположение линии ПЛИС, IOSTANDARD указывает используемый стандарт линий ввода/вывода, SLEW указывает скорость нарастания выходного напряжения, DRIVE указывает выходной ток источника тока, встроенного в ПЛИС в миллиамперах.

Net “SEGA0” LOC=”E3” | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;
Net “SEGB0” LOC=”E1” | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;
Net “SEGC0” LOC=”G5” | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;

Net “SEGD0” LOC=”D1” | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;
Net “SEGE0” LOC=”E4” | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;
Net “SEGF0” LOC=”C1” | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;
Net “SEGG0” LOC=”C2” | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;
Net “COM0” LOC=”B2” | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;
Net “SEGA1” LOC=”H6” | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;
Net “SEGB1” LOC=”K2” | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;
Net “SEGC1” LOC=”H3” | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;
Net “SEGD1” LOC=”K1” | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;
Net “SEGE1” LOC=”G4” | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;

Каждая строка задает соответствие обозначенного на схеме порта и физического вывода микросхемы ПЛИС. За ключевым словом NET идет имя цепи, которая должна быть подключена к выводу, обозначенному словом LOC. Между словом LOC и

обозначением вывода должен стоять знак «=». Строка оканчивается точкой с запятой. Комментарии в коде начинаются с символа «#» или пары символов «//» и нужны только для человека (программа-транслятор их игнорирует).

Программирование ПЛИС

После создания схемы и назначения выводов может быть сгенерирован конфигурационный файл для ПЛИС (файл прошивки). Файл имеет расширение. bit и предназначен для загрузки в ПЛИС. Для получения данного файла необходимо во вкладке Design (рис. 12) выделить файл верхнего уровня (в данном случае – main) и в нижней половине вкладки запустить пункт Generate Programming File.

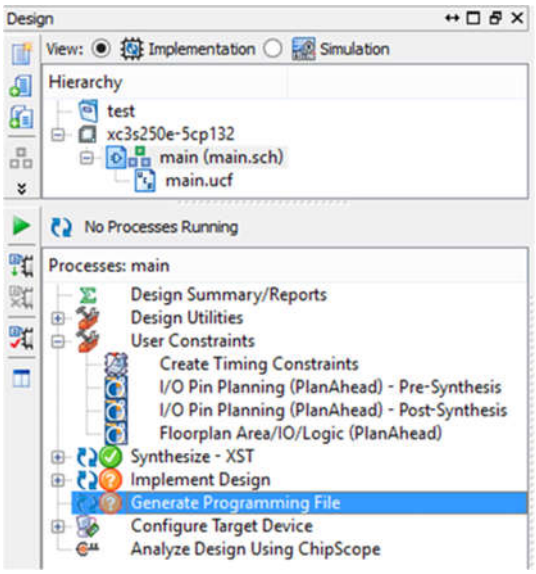


Рис. 12. Создание файла прошивки

При создании файла прошивки проект проходит через множество последовательных преобразований. В первую очередь происходит конвертация всех графических схем проекта в их VHDL описание. Затем запускается этап синтеза (Synthesize). На данном этапе VHDL описание преобразуется в так называемый список связей (netlist). Эти связи устанавливаются между специфичными для ПЛИС аппаратными блоками. Затем на этапе трансляции (Translation) этот список связей преобразуется в файл NGD, в котором учитываются заданные ранее ограничения (ucf) и собираются в один файл все остальные файлы компонентов (например, синтезированные файлы коммерческих компонентов, код которых закрыт). На этапе отображения (Map) проект упаковывается в конкретную ПЛИС – здесь учитываются физические ограничения ПЛИС по числу элементов. Затем наступает этап размещения и трассировки (Place & Route). На этом этапе каждый элемент схемы размещается в ПЛИС учитывая требования трассировки и соблюдения временных ограничений.

Полученный в результате файл с расширением .bit может быть загружен в ПЛИС по интерфейсу JTAG. Для загрузки соответствующим образом сформатированного PROM файла в ЭСППЗУ микросхемы ПЛИС через встроенный в отладочную плату интерфейс USB-JTAG, подключите к ней USB кабель, включите питание платы, переведя выключатель питания в положение ON, и выполните следующие действия:

1. Переведите движковый переключатель отладочной платы NI Digital Electronics FPGA Board в положение JTAG. Движковый переключатель SW 9 в положении JTAG.

2. Запустите программу Xilinx ISE.

3. В меню этой программы выберите следующее: **File» Open Project.** Затем передвигаясь по меню проводника укажите расположение файла.

4. На панели Processes окна проекта раскройте иерахический список **Configure Target Device** (рис. 13).

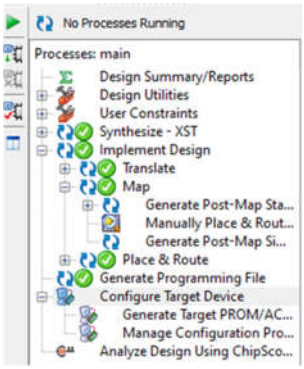


Рис.13

5. Двойным щелчком раскройте пункт **Manage Configuration Project (iMPACT).**

Откроется окно Welcome to iMPACT. Если стартовое окно Welcome to iMPACT не открывается, Вы можете запустить программный модуль iMPACT выполнив следующие действия: **Start » Programs » Xilinx ISE Design Suite » ISE » Accessories » iMPACT.**

6. Выберите **Configure devices using Boundary-Scan (JTAG) and Automatically connect to a cable and identify Boundary-Scan**

chain (этот пункт быть должен выбрать по умолчанию.**)** Так же данный пункт можно найти через **Edit/Launch Wizard**. Нажмите на кнопку **Finish**. Откроется окно Assign New Configuration.

7. В окне Assign New Configuration File выберите конфигурационный файл main.bit (ваше имя файла). Нажмите **Open.** Вы имеете возможность выбрать другие конфигурационные файлы. Если Вы не нуждаетесь в дополнительных конфигурациях, нажмите **Cancel.** Затем раскроется окно Device programming interface.

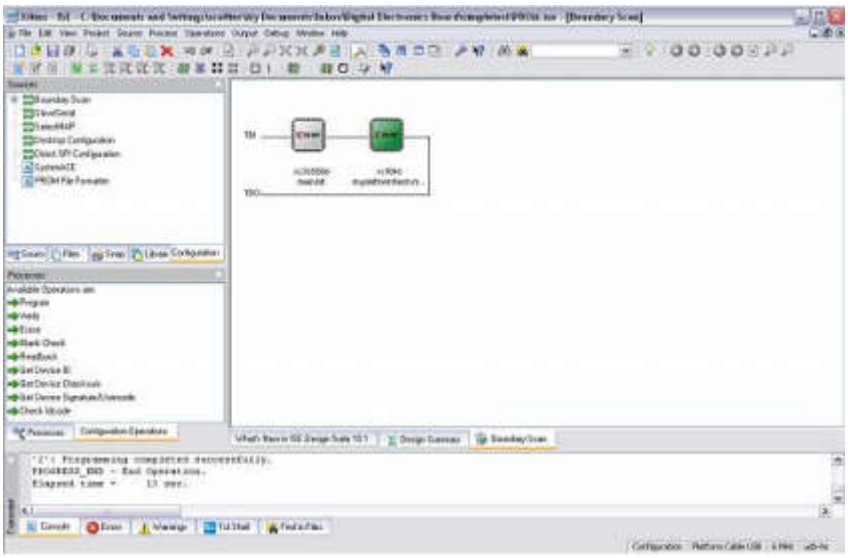


Рис.14 Окно проекта PROM.ise

8. В окне Device programming interface выберите **Device 1 (FPGA, xc3s500e).** Для программирования устройства 1 (Devce 1). На отладочной плате Digital Electronics FPGA Board установлена микросхема ПЛИС Xilinx XC3S500E Spartan-3E.

9. Нажмите на кнопку **ОК.** В окне проекта. Раскроется панель Boundary Scan.

10. Нажмите правой клавишей мыши наведя её на иконку **xcf04s file?,** в выпадающем меню выберите пункт **Assign New Configuration File** для того чтобы ассоциировать PROM file (.mcs) с сЭСППЗУ XCF04S в JTAG цепочке.

11. Раскройте нажатием правой клавиши мыши иконку **myplatformflash.mcs** и в выпадающем меню выберите пункт **Program.**

Реализация схемы на языке VHDL

Рассмотренная схема, изображенная на рис. 7, может быть задана не только графической схемой, но и на языке описания аппаратуры (HDL). Существует множество языков HDL, многие из которых являются модификациями или расширениями друг друга. В дальнейшем будет рассматриваться VHDL (VHSIC (Very high speed integrated circuits) Hardware Description Language).

Следует отметить, что языки HDL, говоря строго, не являются в полном смысле языками программирования. Если, например, в языке С последовательно встречаются строки:

a = b; c = a;

то в ходе выполнения программы данные действия будут выполнены последовательно: сначала переменная a станет равна b, а затем переменная c примет значение a. На языке VHDL аналогичная конструкция не описывает последовательность:

a <= b; c <= a;

Данная запись будет означать лишь то, что сигнал a примет значение b, а сигнал c примет значение a, но последовательность изменений сигнала здесь не определена. Возможно, при генерации схемы данные сигналы будут заданы одним проводником, и понятие последовательности вообще не будет иметь смысла.

Рассмотрим порядок создание описания цифровой схемы на VHDL.

Сначала создается файл исходных текстов VHDL (вкладка Design→New Source→VHDL Module). После указания имени файла в появившемся окне необходимо указать все имеющиеся входы и выходы схемы (рис. 15).

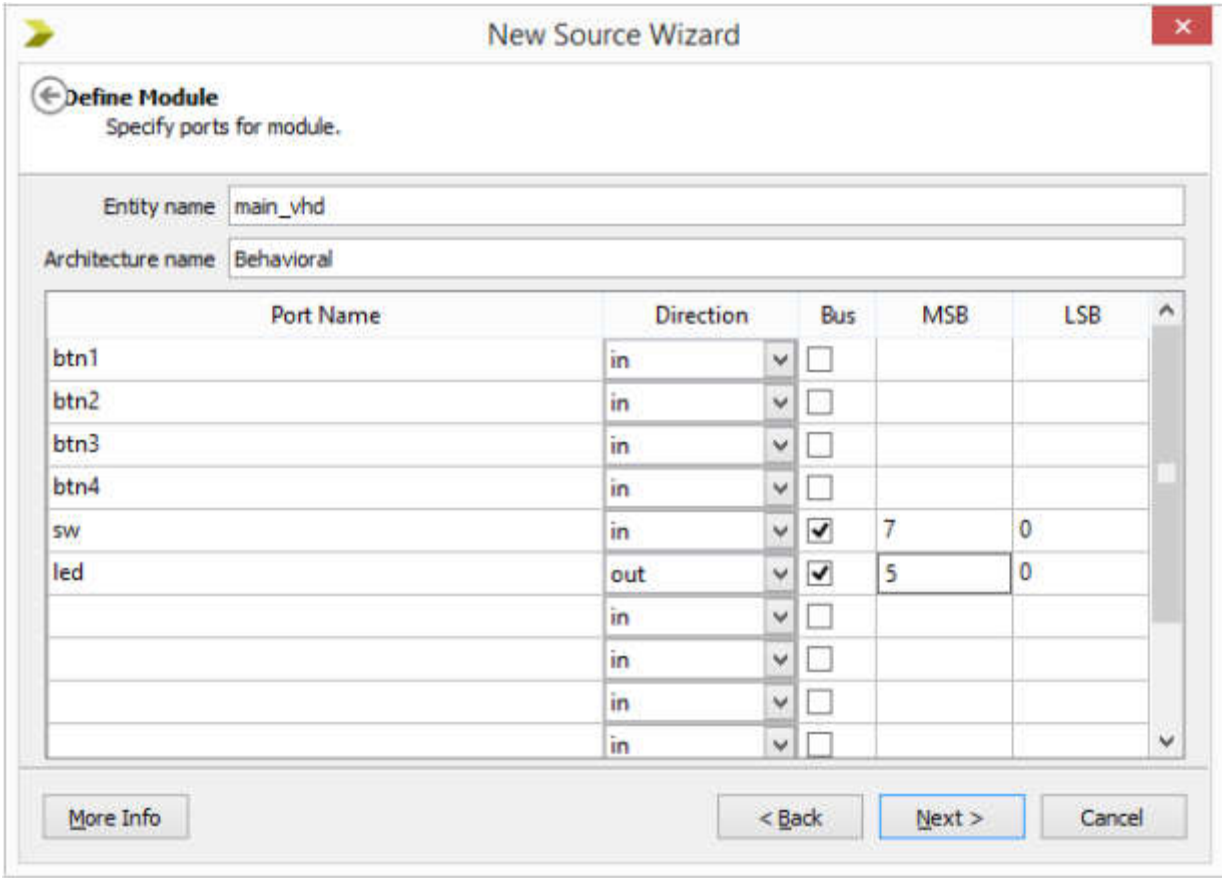


Рис. 15. Задание портов схемы

Обратите внимание на обозначение шин sw и led: необходимо указать, что порт является шиной, и ввести верхний и нижний индексы проводников (аналогично схеме, значения 7 и 0 для шины sw показывают, что шина содержит 8 проводников с номерами от 7 до 0).

По завершении этой операции нажимаем Next – появляется окно, в котором описан результат инициализации выводов схемы. Проверив правильность обозначения выводов, завершаем этот этап нажатием Finish. После этого будет создан файл модуля со следующим базовым текстом:

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

-- Uncomment the following library declaration if
using
-- arithmetic functions with Signed or Unsigned
values
--use IEEE.NUMERIC_STD.ALL;

entity main_vhd is
  Port ( btn1 : in  STD_LOGIC;
         btn2 : in  STD_LOGIC;
         btn3 : in  STD_LOGIC;
         btn4 : in  STD_LOGIC;
         sw : in  STD_LOGIC_VECTOR (7 downto 0);
         led : out  STD_LOGIC_VECTOR (5 downto 0));
end main_vhd;

architecture Behavioral of main_vhd is

begin

end Behavioral;
```

Рассмотрим подробнее структуру файла на VHDL. Все VHDL программы начинаются с двух объявлений.

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

Первая строка указывает на использование стандартной библиотеки IEEE (IEEE – Institute of Electrical and Electronics Engineers – Институт Инженеров Электротехники и Электроники. Данный институт отвечает за разработку множества современных стандартов в области электроники). Следующая строка, начинающаяся со слова use, указывает на использование пакета STD_LOGIC_1164. Этот пакет описывает типы данных STD_LOGIC и STD_LOGIC_VECTOR. Тип STD_LOGIC соответствует единичному проводнику и может принимать значения 0 или 1. В свою очередь, тип STD_LOGIC_VECTOR описывает шину из таких проводников.

Далее по тексту программы расположен комментарий. Комментарии в VHDL начинаются с двух дефисов (--) и нужны только для человека, читающего код.

Одним из основных понятий VHDL является entity – «элемент», «объект». Вся цифровая схема состоит из элементов различной степени сложности, и сами элементы в свою очередь могут состоять из других элементов. Запись «entity main_vhd is» указывает на начало описания нового компонента с именем main_vhd (имя может быть любым, но должно состоять из латинских букв).

Главное, что содержит entity, – описание входных и выходных сигналов данного элемента. То есть entity описывает элемент как некоторый «чёрный ящик», о котором известно только то, какие сигналы он принимает, а какие отдаёт.

Описание входов и выходов происходит в круглых скобках после ключевого слова Port:

```
Port ( btn1 : in  STD_LOGIC;
       btn2 : in  STD_LOGIC;
       btn3 : in  STD_LOGIC;
       btn4 : in  STD_LOGIC;
       sw :   in  STD_LOGIC_VECTOR (7 downto 0);
       led :  out STD_LOGIC_VECTOR (5 downto 0)
);
```

Внутри расположены описания входных и выходных сигналов, аналогичные тем, что были на исходной схеме. Рассмотрим первую строку: здесь btn1 – имя порта, in – указывает, что порт работает на вход (соответственно далее слово out обозначает порт, работающий на выход), STD_LOGIC – тип порта (в данном случае порт представляет собой одиночный проводник).

Аналогично описываются и шины – они отличаются тем, что имеют тип STD_LOGIC_VECTOR и в конце указывается их ширина. Описание компонента заканчивается строкой end main_vhd (если элемент носит другое имя, на месте main_vhd должно быть именно оно).

Конечно, для описания работы схемы недостаточно описать только её входы и выходы. Собственно логика работы описывается внутри блока architecture («архитектура»). Начало блока архитектура задается строкой architecture Behavioral of main_vhd is,

где Behavoiral – название архитектуры (может быть любым). Соответственно между строками begin и end Behavioral описывается собственно логика работы компонента.

Моделирование работы схемы

Перед тем, как использовать созданное описание цифровой схемы для конфигурации ПЛИС, необходимо убедиться, что схема функционирует в точном соответствии с заданной логикой работы и соблюдением временных характеристик. Для этого проводится функциональное моделирование работы схемы или отдельных её узлов при помощи симулятора ISim. Симулятор может быть запущен следующим образом (рис. 16): необходимо переключиться в режим симуляции в окне Design, выделить исследуемую схему и запустить пункт Simulate Behavioral Model.

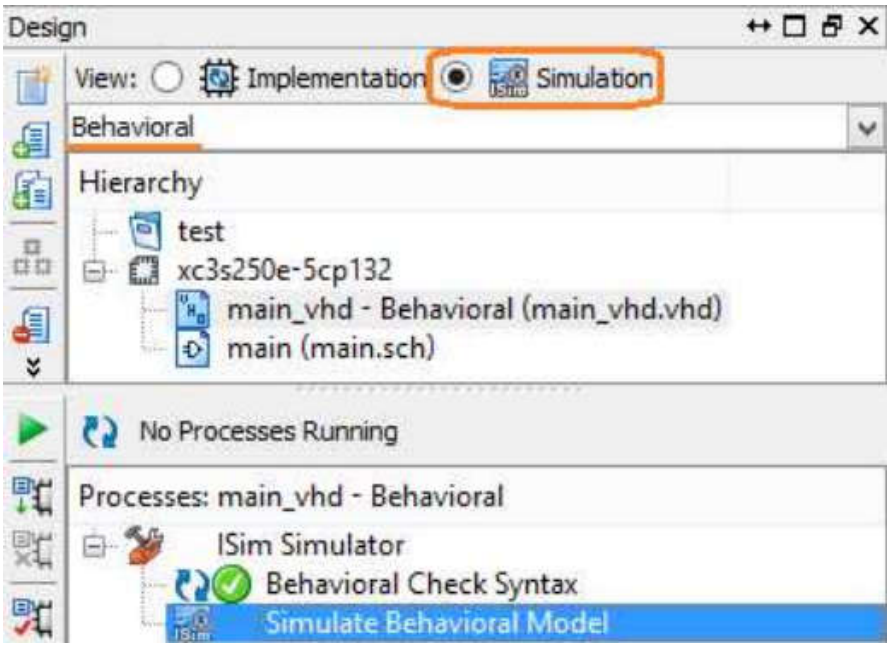


Рис. 16. Запуск симулятора Isim

Однако запущенный таким образом симулятор не сможет запустить моделирование, так как для него не заданы входные воздействия на моделируемую схему. Для решения этой задачи необходимо включить в проект файл на VHDL, содержащий необходимую информацию.

С помощью команды New Source в проект необходимо включить файл типа VHDL Test Bench и в окне Associate Source указать файл, который будет моделироваться. В данном примере это файл main_vhd.

После создания появится файл, в котором автоматически создана большая часть необходимого текста, однако присутствует и часть, которая для данной симуляции не нужна. Рассмотрим данный файл подробнее.

Первые строки:

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

уже встречались при создании VHDL-описания компонента и определяют используемые библиотеки.

Далее следует описание компонента main_tb, который и подаёт сигналы на моделируемый элемент.

```
ENTITY main_tb IS
END main_tb;
```

Так как данный компонент не синтезируемый (не предназначен для генерации кода прошивки ПЛИС), списка портов у него нет.

Строка

```
ARCHITECTURE behavior OF main_tb IS
```

открывает блок кода, описывающий поведение тестирующего компонента.

Следующий блок кода

```
  COMPONENT main_vhd
  PORT(
       btn1 : IN  std_logic;
       btn2 : IN  std_logic;
       btn3 : IN  std_logic;
       btn4 : IN  std_logic;
       sw : IN  std_logic_vector(7 downto 0);
       led : OUT  std_logic_vector(5 downto 0)
      );
  END COMPONENT;
```

объявляет, что внутри компонента main_tb содержится компонент main_vhd (который и будет тестироваться) и указываются его порты.

Далее указываются имеющиеся внутри компонента сигналы:

```
    --Inputs
       signal btn1 : std_logic := '0';
       signal btn2 : std_logic := '0';
       signal btn3 : std_logic := '0';
       signal btn4 : std_logic := '0';
       signal sw : std_logic_vector(7 downto 0) :=
 (others => '0');
    --Outputs
       signal led : std_logic_vector(5 downto 0);
       constant <clock>_period : time := 10 ns;
```

Эти проводники должны подключаться к тестируемому компоненту main_vhd и подавать на его входы тестирующие сигналы. Здесь необходимо обратить внимание на последнюю

строку – она была сгенерирована автоматически и вызывает ошибку компиляции. Дело в том, что изначально предполагается, что моделируемая схема содержит логику, для которой необходимо подать на вход моделируемого элемента периодический прямоугольный тактовый сигнал определенной частоты, но так как в нашей схеме нет сигнала, который отмечен как тактовый, то была сгенерирована заглушка вида <clock>. Как следует из слова constant, эта строка задает постоянное число типа time (время), которое равно 10 нс. Для наших целей эта константа не нужна и может быть удалена.

Далее следует ключевое слово begin, указывающее на то, что описание входящих в архитектуру элементов закончено и начинается описание их взаимосвязей.

```
BEGIN
 -- Instantiate the Unit Under Test (UUT)
   uut: main_vhd PORT MAP (
          btn1 => btn1,
          btn2 => btn2,
          btn3 => btn3,
          btn4 => btn4,
          sw => sw,
          led => led
        );
```

Здесь uut – имя компонента, main_vhd – тип компонента, а внутри блока port map перечисляются соединения данного компонента и сигналов. В данном случае имена сигналов и портов компонента совпадают, поэтому запись вида btn1=>btn1 обозначает

подключение к порту btn1 компонента uut типа main_vhd сигнала по имени btn1.

Код, идущий дальше, формирует периодический сигнал.

```
<clock>_process :process
begin
  <clock> <= '0';
  wait for <clock>_period/2;
  <clock> <= '1';
  wait for <clock>_period/2;
end process;
```

Аналогичным образом выполняется и второй процесс в данном коде:

```
stim_proc: process
begin
   wait for 100 ns;
   wait for <clock>_period*10;
   wait;
end process;
```

Этот код необходим для формирования сигнала сброса схемы. Обратите внимание на последнюю инструкцию wait – после неё не указано время ожидания, и таким образом процесс здесь остановится и будет ждать вечно.

После того, как файл тестовых воздействий создан, может быть запущен симулятор. Для этого необходимо во вкладке Design выделить файл с main_tb (файл с тестовыми сигналами) и в нижнем окне Processes запустить пункт Simulate Behavioral Model.

После открытия окна симулятора необходимо выполнить следующие действия:

1) сбросить текущие настройки  ;

2) установить время симуляции 1 us 

;

3) запустить симуляцию на заданное время  .

Пример результата моделирования представлен на рис. 17.

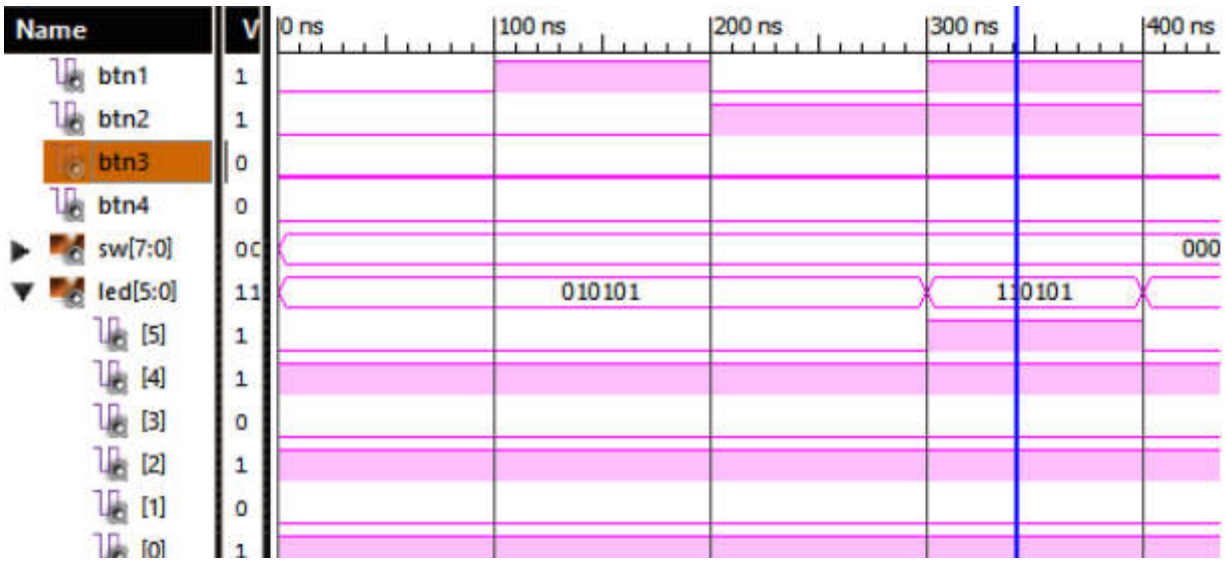


Рис. 17 – Результата моделирования

Следует обратить внимание на то, что, если не все сигналы имеют определенное значение, результат симуляции может быть ошибочным. Так, значение U в поле Value для сигнала означает, что его состояние не определено.

Проектирование арифметических устройств на языке VHDL

Сначала создается новый проект (рисунок 18), а затем файл исходных текстов VHDL (вкладка Design→New Source→VHDL Module, рисунок 19). После указания имени файла в появившемся

окне можно указать все имеющиеся входы и выходы схемы (на данном этапе указание портов можно не осуществлять).

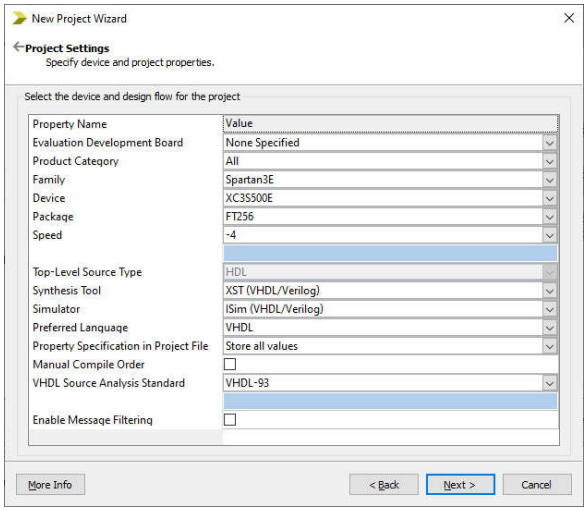


Рисунок 18

По завершении этой операции нажимаем Next – появляется окно, в котором описан результат инициализации выводов схемы. Проверив правильность обозначения выводов, завершаем этот этап нажатием Finish.

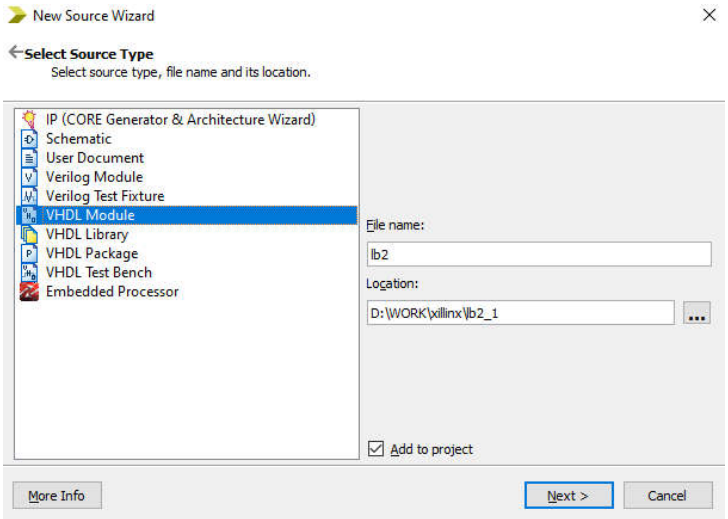


Рисунок 19

Далее появляется рабочее окно, содержащее исходную структуру программы на языке VHDL (рисунок 20). Все VHDL программы начинаются с двух объявлений.

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

```
13  -- Dependencies:
14  --
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  --
19  ----------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22
23  -- Uncomment the following library declaration if using
24  -- arithmetic functions with Signed or Unsigned values
25  --use IEEE.NUMERIC_STD.ALL;
26
27  -- Uncomment the following library declaration if instantiating
28  -- any Xilinx primitives in this code.
29  --library UNISIM;
30  --use UNISIM.VComponents.all;
31
32  entity lb2 is
33  end lb2;
34
35  architecture Behavioral of lb2 is
36
37  begin
38
39
40  end Behavioral;
41
42
```

Рисунок 20

Первая строка указывает на использование стандартной библиотеки IEEE. Следующая строка, начинающаяся со слова use, указывает на использование пакета STD_LOGIC_1164. Этот пакет описывает типы данных STD_LOGIC и STD_LOGIC_VECTOR. Тип STD_LOGIC соответствует единичному проводнику и может

принимать значения 0 или 1. В свою очередь, тип STD_LOGIC_VECTOR описывает шину из таких проводников.

Раздел IEEE_STD_LOGIC_ARITH (use IEEE.STD_LOGIC_ARITH.ALL) содержит шаблоны функций преобразования, которые относятся к следующим типам данных: SIGNED, UNSIGNED, SMALL_INT, INTEGER, STD_ULOGIC, STD_LOGIC и STD_LOGIC_VECTOR. Эти функции определены в пакете std_logic_arith библиотеки IEEE.

В разделе IEEE-STD_LOGIC_SIGNED (use IEEE.STD_LOGIC_SIGNED.ALL), входящем в состав папки Conversion Functions, представлен шаблон вызова функции преобразования массивов с учетом знака, которая применяется по отношению к данным типа STD_LOGIC_VECTOR. Определение этой функций приведено в пакете std_logic_signed библиотеки IEEE.

Раздел IEEE-STD_LOGIC_UNSIGNED, представленный в папке Conversion Functions, содержит шаблон функции преобразования массивов без учета знака, которая используется по отношению к операндам типа STD_ LOGIC_VECTOR. Определение этой функции приведено в пакете std_logic_unsigned библиотеки IEEE.

Далее по тексту программы расположен комментарий. Комментарии в VHDL начинаются с двух дефисов (--) и нужны только для человека, читающего код.

Структура проекта

Проект в VHDL определяется как совокупность связанных проектных пакетов. Проектными пакетами (design unit) называются независимые (external) фрагменты описаний, которые можно независимо анализировать компилятором и помещать в рабочую библиотеку проекта (Work).

Проектными пакетами могут быть:

• объявление интерфейса объекта проекта (entity);

• объявление архитектуры (architecture);

• объявление конфигурации (configuration);

• объявление интерфейса пакета (package);

• объявление тела пакета (package body).

Можно выделить две категории модулей проекта: первичные и вторичные. К первичным относятся объявления пакета, объекта проекта, конфигурации, к вторичным — объявление архитектуры, тела пакета. Файл, в котором размещаются один или несколько модулей проекта, называется файлом проекта (design file).

Все проанализированные модули помещаются в библиотеку проекта (design library) и становятся библиотечными модулями (library unit). Существует два класса библиотек проекта: рабочие библиотеки и библиотеки ресурсов. Рабочая библиотека — это библиотека Work, с которой в данном сеансе работает пользователь и в которую помещается пакет, полученный в результате анализа пакета проекта. Библиотека ресурсов — это библиотека, содержащая библиотечные модули, используемые в анализируемом

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

Описание объектов проекта

Полное описание модели объекта проекта состоит из следующих частей:

а) описание интерфейса объекта проекта (entity), включающее:

— Port (списки входных и выходных сигналов);

— Generic (настраиваемые параметры модели, например, generic (data_width: natural:=10););

Entity описывает элемент как некоторый «чёрный ящик», о котором известно только то, какие сигналы он принимает, а какие отдаёт.

Описание входов и выходов происходит в круглых скобках после ключевого слова Port:

```
Port ( s1 : out  signed( data_width+3 downto 0); -- выход
       a1 : in      signed ( data_width downto 0); -- вход
       a2 : in      signed (data_width downto 0);
       a3 : in      signed ( data_width downto 0);
       a4 : in      signed ( data_width downto 0));
```

Рассмотрим первую строку: здесь s1 – имя порта, out – указывает, что порт работает на вывод, signed – тип порта, в скобках – разрядность данных.

б) описание архитектуры объекта проекта (ARCHITECTURE), включающее:

— объявление констант, переменных и дополнительных (внутренних) сигналов

```
signal sig1, sig2, sig3, sig4: signed( data_width*2+1 downto 0);
constant b1 : signed (data_width downto 0) :="01100000001";;
```

— операторную часть, представляющую собой описание объекта проекта на структурном или поведенческом уровне (начинается с begin, оканчивается - end Behavioral;);

в) (только для структурной формы описания) описание конфигурации (configuration), задающей подключение библиотеки моделей элементов и выборку их в качестве компонентов структуры.

В VHDL существуют два основных уровня описания архитектуры объектов — поведенческий и структурный.

Поведенческий уровень. На поведенческом уровне описание объектов проекта представляется в VHDL в виде набора параллельных процессов. Организация процессов обеспечивается введением оператора процесса и оператора параллельного присваивания сигналов, также представляющего процессы. Это определяет две формы описания объектов на поведенческом уровне — потоковую и процессную.

Потоковая форма. В потоковой форме описания объекта проекта его архитектура представлена в виде множества параллельных операций. Для сигналов вводятся специальные операторы параллельного присваивания "<=", являющиеся эквивалентами операторов присваивания ":=" для простых переменных, но имитирующие параллельные процессы с сигналами. Это реализуется искусственным приемом выполнения

этих операторов на каждом шаге моделирования с бесконечно малой дельта-задержкой.

Процессная форма. В процессной форме описание объекта производится при помощи процессов. Процесс в VHDL определяет независимую повторяющуюся последовательность операторов и представляет поведение некоторой части проекта. После того, как последний оператор последовательности выполнен, выполнение начинается с первого оператора процесса. Для определения процесса используется оператор process, который состоит из объявлений (после ключевого слова process) и операторной части, которая начинается после слова begin. В объявлении процесса допускается создавать переменные, в то время как объявлять сигналы в этой части нельзя.

Предложения в операторной части выполняются строго последовательно (в отличие от параллельного назначения потоковой формы описания и конкретизации компонентов в структурном стиле) и включают, подобно языкам высокого уровня, операторы присваивания, условные операторы и операторы циклов. Оператор "<=" здесь используется как оператор назначения сигнала (в отличие от потоковой формы, где он используется как оператор параллельного присваивания) и также выполняется последовательно.

Процессы в VHDL могут быть вложенными; также существует понятие пассивного процесса — процесса, в котором (явно или

опосредовано) не производится назначение сигнала ни в одном из его операторов.

Поведенческий уровень описания объектов проекта является базовым, поскольку любые схемы, по крайней мере на самом нижнем уровне, представлены элементами, реализация которых выполнена на уровне поведения.

Структурный уровень описания объектов проекта. На структурном уровне объект представлен в виде иерархии связанных компонентов, на низшем уровне компоненты реализуются при помощи поведенческого описания. В структурном описании после объявления архитектуры следуют объявления компонентов, используемых в архитектуре. В теле описания архитектуры для создания экземпляров компонентов используются предложения конкретизации компонентов. Предложение конкретизации компонента начинается с метки, за которой следует имя компонента, а затем, если это необходимо, операторы назначения карты параметров (generic map) и карты портов (port map).

Каждый экземпляр компонента должен быть сопоставлен с парой объект—архитектура. По умолчанию осуществляется поиск объекта с совпадающим с компонентом именем, списком портов и параметров, однако данное поведение можно изменить при помощи описания конфигурации.

Арифметические операции в VHDL

На рисунке 21 представлены основные арифметические операции языка VHDL.

Не все операторы могут быть автоматически синтезированы (т.е. превращены в реальную схему). Например, оператор деления \ не может быть синтезирован автоматически. При этом все операторы могут использоваться при моделировании. За каждым оператором, который может быть синтезирован, стоят реальные аппаратные ресурсы, т.е. знак +(*) реализуется блоком сумматора (умножителя).

Операции языка VHDL

Обозначение	Название
not	логическое НЕ
and	логическое И
or	логическое ИЛИ
nand	логическое И-НЕ
nor	логическое ИЛИ-НЕ
xor	исключающее ИЛИ
xnor	эквивалентность
=	равно
/=	не равно
<	меньше
<=	меньше либо равно
>	больше
>=	больше либо равно
+	сложение, присвоение знака +
–	вычитание, присвоение знака -
&	конкатенация
*	умножение
/	деление
mod	модуль
rem	остаток
**	возведение в степень
abs	абсолютное значение

Рисунок 21

Исходя из контекста VHDL-кода следует отличать оператор <= (назначение сигнала) и оператор <= (меньше либо равно).

При сложении (вычитании) вектора (SIGNED, UNSIGNED) и целого числа (NATURAL, INTEGER) результирующий вектор (независимо от значения целого числа) имеет размерность вектора.

При умножении двух векторов, размерность результирующего вектора равна сумме размеров векторов-аргументов:

```
signal l_u : unsigned(3 downto 0) := "1101";  -- "1101" = (13)
signal r_u : unsigned(2 downto 0) := "100";   -- "100" = (4)
signal y_u : unsigned(6 downto 0);
...
y_u <= l_u * r_u;       -- результат y_u = "0110100" (52)
```

Реализация нелинейной функции

Нелинейная функция может быть реализована прямым вычислением на отдельном микроконтроллере (МК), МК встроенном в ПЛИС в виде IP-модуля и табличным методом. При этом таблица может использовать внешнее выделенное ПЗУ или встроенную память ПЛИС. В современных ПЛИС табличный метод реализуется на основе встроенной оперативной памяти.

Память на языке VHDL может быть описана массивом (*array*) векторов. Разрядность вектора определяется разрядностью ячейки памяти, а количество векторов — количеством ячеек в модуле памяти:

```
type initmem is array(0 to 2**data_width-1) of bit_vector(data_width-1 downto 0);
```

Например, для модуля памяти из 128 ячеек, каждая из которых содержит 8 бит, необходимо объявить массив, в котором

содержится 128 вектора, каждый из которых является восьмиразрядным.

Описание памяти приводится после команды объявления архитектуры:

architecture Behavioral of init_mem is.

При описании блоков памяти содержимое ячеек необходимо определять при написании программы. Возможно использование нескольких вариантов определения содержимого памяти:

1. создание константы или сигнала типа «массив»;
2. использование оператора case;
3. использование *.txt файла и атрибутов синтеза (объявляется в блоке архитектуры после сигналов и констант):

```
attribute RAM_STYLE : string; -- Block/Signal (RAM_STYLE) включает в себя шаблон варианта записи атрибута RAM_STYLE,
attribute RAM_STYLE of ram : signal is "BLOCK";
```

Предварительно создается функция построчного чтения данных из текстового файла – содержимого ячеек памяти.

```
impure function InitRamFromFile (InitRamFile : in string) return initmem is  -- функция чтения из файла
   FILE ram_file         : text is in InitRamFile;   -- декларация файла чтения памяти
   variable line_ram : line;       -- присваивается значение переменной  строка читаемая из файла
   variable ram       : initmem;
begin
   for I in initmem'range loop    -- цикл перебора
       readline (ram_file, line_ram);   -- чтение всех строк файла
       read (line_ram, ram(I));
   end loop;
   return ram;
end function;

 signal ram : initmem := InitRamFromFile("ram_file1.txt");
```

Описание оперативных запоминающих устройств (ОЗУ) отличается от описания постоянно запоминающих устройств (ПЗУ) тем, что в ОЗУ можно осуществлять запись.

Работа ОЗУ может быть описана таблицей 1:

Таблица 1 – Режимы работы ОЗУ

we	*clk*	Режим работы
0	clk'event **and** clk = '1'	Чтение
1	clk'event **and** clk = '1'	Запись

Команда чтения данных на языке VHDL может быть представлена следующей записью:

```
mem_data <=to_stdlogicvector(ram(conv_integer(addr)));
```

где *addr* – указание адреса ячейки для чтения. Так как адрес используется как индекс массива памяти, тип данных для адреса – integer.

Ниже приведен пример описания процесса работы памяти на языке VHDL:

```
process (clk, ram, mem_data, sl )
    begin
       if clk'event and clk = '1' then
if we = '1' then
             ram(conv_integer(addr)) <= to_bitvector(data_in);    -- запись
          end if;
             mem_data <=to_stdlogicvector(ram(conv_integer(addr))); -- чтение


       end if;


    end process;
```

Чтение/запись текстовых файлов

Процесс проектирования любого электронного устройства включает этапы, требующие проведения экспериментальных исследований. Применение специализированных исследовательских стендов позволяет автоматизировать операции

ввода исходных данных и вывода результатов исследования в требуемом формате для их дальнейшей обработки.

В среде Xilinx ISE такая задача может быть решена реализацией функций чтения/записи текстовых файлов с расширением *.txt на языке VHDL.

Применение таких функций требует подключения специализированных библиотек:

use STD.textio.all;

use ieee.std_logic_textio.all;.

Необходимо помнить, что исследовательский стенд на языке VHDL, реализующий функции чтения и записи текстовых файлов, не может быть синтезирован на ПЛИС и используется только для решения задач моделирования электронного устройства на этапе проведения экспериментальных работ (о чем свидетельствует значок  напротив Implement Design).

Кроме указания специализированных библиотек, в блоке «architecture» необходимо объявить соответствующие текстовые файлы:

file file_VECTORS : text;.

Команды чтения и записи реализуются в виде процесса, в котором учавствуют переменные variable, объявляемые в том же блоке:

variable v_ILINE1   : line;.

Для открытия текстового файла используется следующая командная строка:

file_open(file_VECTORS, "input_vectors1.txt", read_mode);.

Ниже представлены команды чтения и записи в текстовый файл на языке VHDL:

```
readline(file_VECTORS, v_ILINE1);
read(v_ILINE1, v_ADD_TERM1);

write(v_OLINE, v_SUM, right, c_WIDTH);
writeline(file_RESULTS, v_OLINE);
```

По завершению работы с текстовым файлом, его необходимо закрыть:

file_close (file_VECTORS);.

1. Лабораторная работа №1 «Создание принципиальной схемы»

1.1 Алгоритм работы

1. Создать новый проект. Выяснить какие существуют варианты описания цифрового устройства. Определиться с типом файла верхнего уровня (заполнить поле Top-level source type).

2. Используя данные, указанные на ПЛИС отладочной платы, заполнить окно Project Settings.

3. Изучить принцип работы и собрать электрическую принципиальную схему, представленную на рисунке 22.

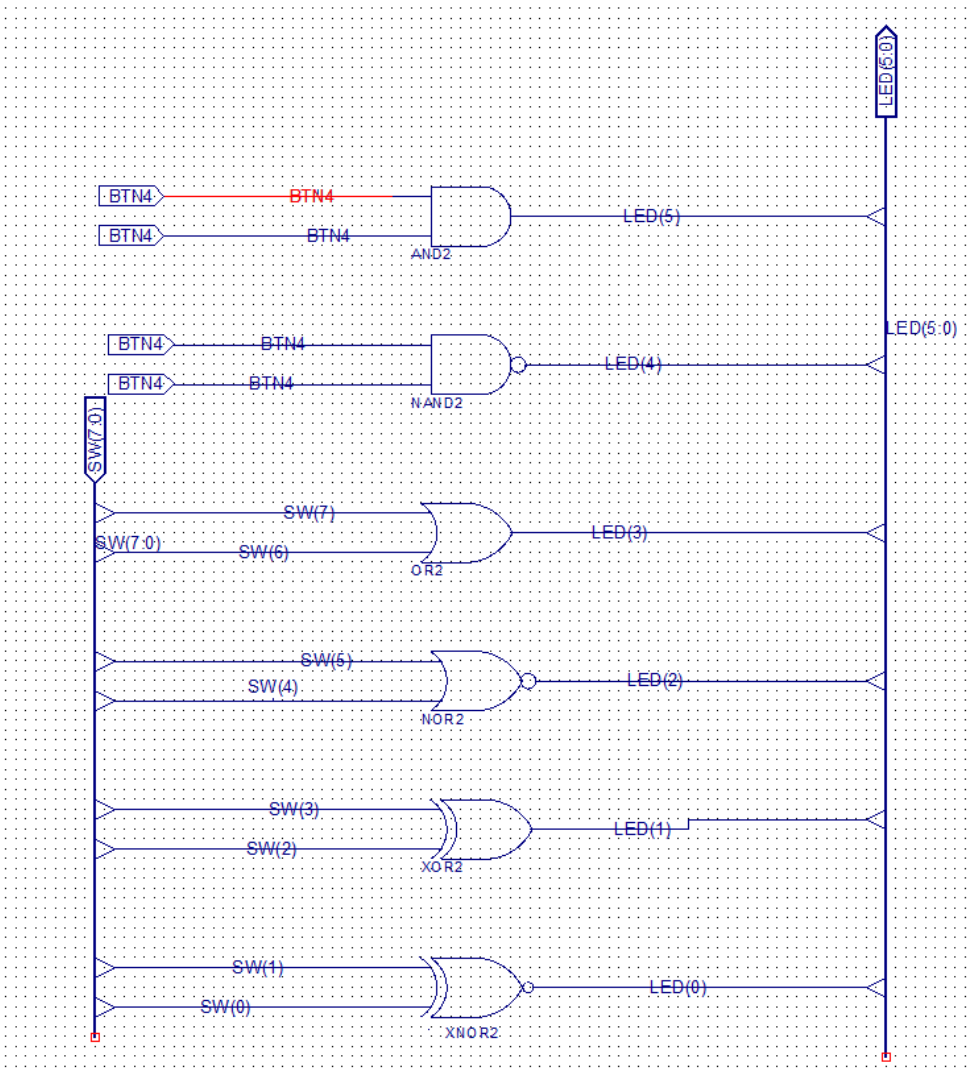


Рисунок 22 – Исследуемая схема

3. Привести таблицы истинности для всех логических элементов схемы.

4. Описать в отчете процесс управления светодиодами (при каких входных комбинациях они светятся).

5. Синтезировать схему, убедиться в отсутствии ошибок.

6. Изучить меню процессов и описать, сопровождая фотографиями.

7. Выполнить моделирование, подтвердить соответствие п.4.

1.2 Форма отчета обучающегося о выполненной лабораторной работе

Отчёт должен быть оформлен в соответствии с требованиями ГОСТ и содержать:

1. Титульный лист;

2. Оглавление;

3. Наименование работы, цель исследований;

4. Таблицы истинности логических элементов, исследуемую электрическую принципиальную схему и описание принципа ее функционирования;

5. Описание меню процессов и соответствующие схемы синтезированного цифрового устройства;

6. Результаты моделирования схемы (временную диаграмму);

7. Анализ диаграммы и выводы.

8. Ответы на контрольные вопросы;

9. Перечень литературы, использованной при подготовке и выполнении работы (ссылки на используемую техническую документацию).

1.3 Контрольные вопросы

Дать письменные ответы на приведенные ниже вопросы.

1. Что такое программируемая логическая интегральная схема?
2. Расскажите о преимуществах ПЛИС.
3. Что такое файл конфигурации ПЛИС?

4. Каким образом производится конфигурирование ПЛИС?

5. Каково назначение системы автоматизированного проектирования?

6. Перечислите и кратко охарактеризуйте этапы создания проекта в САПР.

7. Приведите условное графическое изображение основных логических элементов в соответствии с российскими стандартами и в САПР.

8. Перечислите логические элементы, используемые при управлении светодиодами в схеме.

9. Запишите таблицы истинности для используемых в схеме логических элементов.

10. Какие возможны способы описания цифровой схемы в САПР?

2. Лабораторная работа №2 «Программирование ПЛИС и работа с отладочной платой»

2.1 Алгоритм работы

1. Создать файл с расширением *.ucf, в котором описать привязку выводов схемы к выводам кристалла.

2. Создать конфигурационный файл для ПЛИС (файл прошивки). Файл имеет расширение. bit и предназначен для загрузки в ПЛИС.

3. Выполнить программирование ПЛИС.

4. С помощью отладочной платы осуществить проверку функционирования спроектированной схемы.

5. Устранить ошибки, в случае их обнаружения.

2.2 Форма отчета обучающегося о выполненной лабораторной работе

Отчёт должен быть оформлен в соответствии с требованиями ГОСТ и содержать:

1. Титульный лист;

2. Оглавление;

3. Наименование работы, цель исследований;

4. Описание файла привязки выводов схемы к выводам кристалла;

5. Результаты отладки функционирования спроектированной схемы на плате.

6. Ответы на контрольные вопросы;

7. Перечень литературы, использованной при подготовке и выполнении работы (ссылки на используемую техническую документацию).

2.3 Контрольные вопросы

Дать письменные ответы на приведенные ниже вопросы.

1. Что представляет собой файл конфигурации ПЛИС?

2. Каким образом производится конфигурирование ПЛИС?

3. Перечислите и кратко охарактеризуйте этапы создания проекта в САПР.

4. Перечислите порядок действий при программировании ПЛИС в САПР.

5. Расскажите устройство отладочной платы.

6. Опишите процесс управления светодиодом № 5.

7. Как осуществляется управление светодиодом № 1?

8. Как осуществляется управление светодиодом № 3?

9. Как осуществляется управление светодиодом № 2?

10. Какие логические элементы используются в схеме управления светодиодами?

3. Лабораторная работа №3 «Задание схемы на языке описания аппаратуры (HDL)»

3.1 Алгоритм работы

1. Изучить пункт теоретического раздела методички «Реализация на языке VHDL».

2. Создать файл исходных текстов VHDL.

3. Задать все имеющиеся входы и выходы схемы (порты).

4. Описать работу схемы на языке программирования VHDL.

5. Выполнить моделирование с целью проверки корректности работы схемы управления светодиодами.

3.2 Форма отчета обучающегося о выполненной лабораторной работе

Отчёт должен быть оформлен в соответствии с требованиями ГОСТ и содержать:

1. Титульный лист;

2. Оглавление;

3. Наименование работы, цель исследований;

4. Скриншот окна задания входов и выходов схемы;

5. Программный код.

6. Результаты моделирования.

7. Ответы на контрольные вопросы;

8. Перечень литературы, использованной при подготовке и выполнении работы (ссылки на используемую техническую документацию).

3.3 Контрольные вопросы

Дать письменные ответы на приведенные ниже вопросы.

1. На что указывает первая строка программного кода IEEE?

2. Чем отличаются типы данных STD_LOGIC и STD_LOGIC_VECTOR?

3. Где и как в программном коде задается тип данных?

4. Какова форма записи комментариев на языке VHDL?

5. Перечислите основные блоки программного кода на языке VHDL.

6. Что содержит программный блок «entity»?

7. Как осуществляется задание входов и выходов в программном коде?

8. В каком блоке описывается логика работы программного кода?

9. Какой строкой задается начало блока архитектуры?

10. Какие команды используются при описании схемы управления светодиодами на языке программирования?

4. Лабораторная работа №4 «Моделирование схемы с использованием Test Bench»

4.1 Алгоритм работы

1. Создать файл типа VHDL Test Bench.

2. Доработать исходный программный код, внеся необходимые правки, в соответствии с теоретическим разделом «Моделирование работы схемы».

3. Задать время задержки значений всех входных воздействий достаточным для демонстрации работы устройства при моделировании.

4. Осуществить моделирование схемы управления светодиодами, проверить на соответствие таблицам истинности.

4.2 Форма отчета обучающегося о выполненной лабораторной работе

Отчёт должен быть оформлен в соответствии с требованиями ГОСТ и содержать:

1. Титульный лист;

2. Оглавление;

3. Наименование работы, цель исследований;

4. Программный код;

5. Результаты моделирования;

6. Ответы на контрольные вопросы;

7. Перечень литературы, использованной при подготовке и выполнении работы (ссылки на используемую техническую документацию).

4.3 Контрольные вопросы

Дать письменные ответы на приведенные ниже вопросы.

1. Какие существуют способы задания значений входных воздействий в САПР?

2. Каково назначение файла Test Bench?

3. Опишите структуру программного кода файла Test Bench.

4. С какой целью генерируется строка программного кода «constant <clock>_period..».?

5. Что следует за ключевым словом «begin» в программном коде?

6. Что перечисляется внутри блока port map?

7. Запишите часть программного кода, отвечающего за формирование периодического сигнала.

8. Как выбирается время просмотра симуляции?

9. Что означают символы «U» в окне симуляции?

10. Какие изменения исходного кода потребовались в ходе создания файла Test Bench?

5. Лабораторная работа №5 «Проектирование специализированного арифметического устройства»

5.1 Алгоритм работы

1. На языке VHDL создать специализированное арифметическое устройство, реализующее следующее математическое выражение (n=4):

$$S_1 = \sum_{i=1}^{n} a_i b_i + c$$

.

Значение констант принять равными:

b_1="01110"; b_2="01111"; b_3="00010"; b_4="00101"; c= "00101100".

Перевести значения констант в десятичную систему счисления и занести их в результирующую таблицу отчета.

При задании разрядности векторов использовать константу:

1. a1: in signed (**data_width** downto 0); (пояснить для чего);
2. data_width: natural:=4.

1.1 Для исключения эффекта переполнения перед операцией сложения каждое произведение подлежит дополнительной обработке.

1.2 После операции умножения осуществляется возврат размерности вектора каждого произведения до размерности исходных данных (без знака), отбрасыванием младших разрядов.

1.3 Перед сложением вводится три дополнительных старших разряда: "000" – в случае, если исходное произведение является

положительным числом, "111" – отрицательным. Количество вводимых старших разрядов определяется числом слагаемых (при 5 слагаемых – 3 разряда). Рекомендуется использовать функцию when – else.

2. По завершению редактирования VHDL описания устройства выполнить синтез устройства и проверку файла на наличие синтаксических ошибок (Synthesize – XST).

Об отсутствии ошибок свидетельствуют:

– надпись в окне консоли (отчетов): «Process "Synthesize - XST" completed successfully»;

– зеленые значки напротив пунктов меню окна процессов «Synthesize – XST» и «Implement Design».

При наличии ошибок или предупреждений напротив пунктов меню окна процессов «Synthesize – XST» и «Implement Design» появятся красные или желтые значки соответственно.

Сохранить код программы в отчет.

3. Сформировать отчет синтеза устройства, щелкнув правой кнопкой мыши по пункту «Synthesize – XST» и выбрав пункт View Text Report в окне процессов.

Этот отчет содержит параметры и опции, используемые во время синтеза, протокол ошибок и предупреждений, извлеченные макросы, использование ячеек устройства, и оценку временных характеристик.

Изучить отчет синтеза, выписать наименование, количество, разрядность оборудования задействованного для реализации

устройства. Скопировать из отчета синтеза полный список задействованной элементной базы, расшифровать названия.

Пояснить назначение ячейки, представленной на рисунке 23, принцип работы LUT и функциональное назначение относительно проектируемого специализированного арифметического устройства.

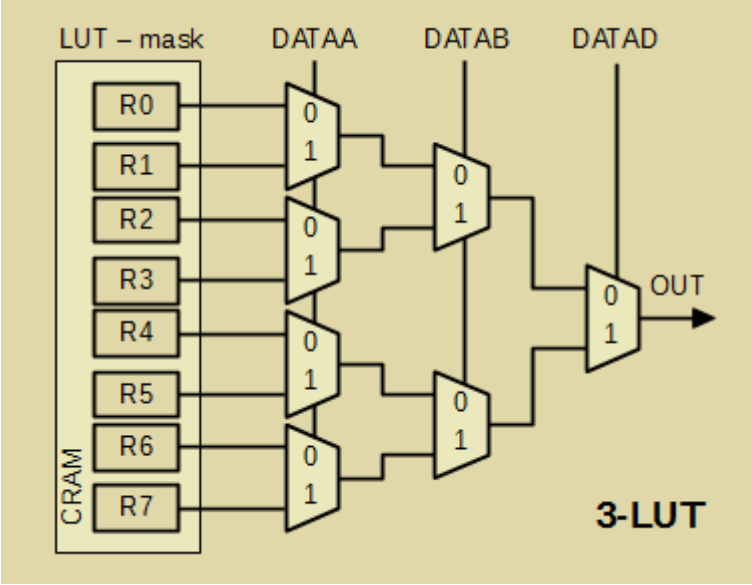


Рисунок 23

Укажите процент задействованных в проекте устройств от их общего числа для данной ПЛИС.

4. Изучить RLT-структуру проекта (раскрыть блок Synthesize – XST, выбрать пункты «View RLT Schematic»). Осуществить просмотр схемы уровня логических преобразований и сохранить в отчет.

Изучить и сохранить в отчет технологическую схему устройства («View Technology Schematic»).

5. Открыть окно симуляции проекта (Simulation – Simulate Behavioral Model). Задать значения входных портов в соответствии с вариантом задания (таблица 2), нажатием правой кнопки мыши по наименованию порта и выбором пункта Force Constant. Значение переменной указывается в поле Force to Value (рисунок 24).

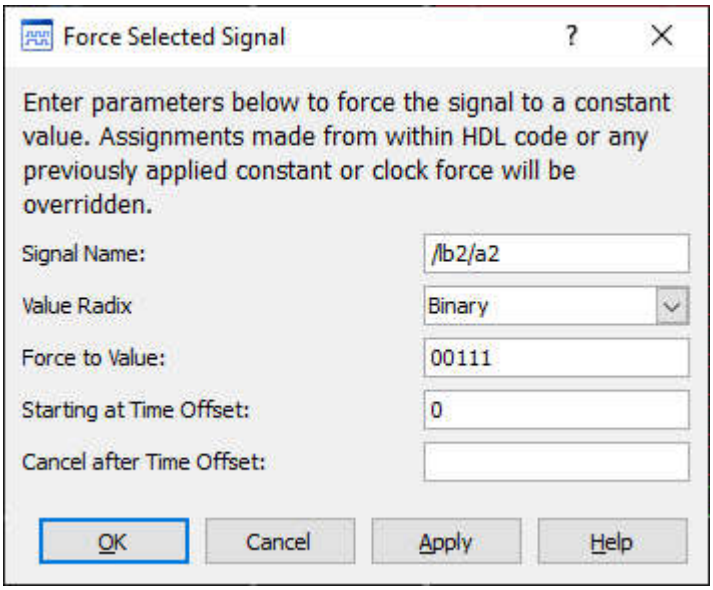


Рисунок 24

Таблица 2– Варианты заданий

№	1	2	3	4	5	6	7	8	9	10
a_1	00111	01111	11111	01111	00011	11111	00000	010101	01101	11111
a_2	11111	10111	10101	10001	10101	10101	11111	00001	10011	11011
a_3	10101	11101	00000	10101	11101	01010	10111	11111	11101	01111
a_4	01110	01111	00001	01010	01010	10011	11111	00001	01111	10101

6. Запустить симуляцию, нажатием на пиктограмму  . Заполнить таблицу 3 и сохранить в отчет.

Таблица 3 – Результаты симуляции схемы устройства

a_n дв./дес.		b_n дв./дес.		c дв./дес.		ab дв./дес.		$\Sigma\, ab + c$ дв./дес.	

7. Рассчитать погрешность вычисления спроектированного арифметического устройства для заданных значений в десятичной системе счисления.

5.2 Форма отчета обучающегося о выполненной лабораторной работе

Отчёт должен быть оформлен в соответствии с требованиями ГОСТ и содержать:

1. Титульный лист;

2. Оглавление;

3. Наименование работы, цель исследований;

4. Программу на языке описания аппаратуры (HDL) с комментариями;

5. Схемы спроектированного устройства на разных уровнях;

6. Результаты моделирования схемы и отчет синтеза устройства;

7. Таблицу с результатами моделирования и выводы.

9. Ответы на контрольные вопросы;

10. Перечень литературы, использованной при подготовке и выполнении работы (ссылки на используемую техническую документацию).

5.3 Контрольные вопросы

Дать письменные ответы на приведенные ниже вопросы.

1. Что такое программируемая логическая интегральная схема?

2. Как расшифровывается аббревиатура LUT? Что реализуется с помощью LUT в ПЛИС?

3. Как преобразовать число из десятичной системы в двоичную и обратно?

4. Что представляет собой двоичное число в дополнительном коде?

5. Какие библиотеки необходимо подключать при работе со знаковыми числами?

6. Какое и сколько оборудования использовано для реализации специализированного арифметического устройства на ПЛИС?

7. Какой процент оборудования задействован для реализации данного устройства от общего числа оборудования данной ПЛИС? Где можно ознакомиться с данной информацией?

8. Для решения каких задач может быть использовано разработанное специализированное арифметическое устройство?

9. С какой целью осуществляется возврат размерности вектора каждого произведения до размерности исходных данных (без знака), отбрасыванием младших разрядов?

10. С какой целью перед результирующей операцией сложения в произведения вводятся три дополнительных старших разряда?

6. Лабораторная работа №6 «Табличный синтезатор нелинейной функции»

6.1 Алгоритм работы

1. Формирование таблицы нелинейного преобразования в Excel.

1.1 Создать таблицу в Excel:

— первый столбец (масштабированное целочисленное значение аргумента к заданной шкале) заполняется числами от 0 до $N=2^n-1$ с шагом 1 ($n=4$ – разрядность входных сигналов a_i без знака);

— второй столбец (аргумент X) заполняется как произведение первого столбца на шаг (шаг вычисляется по следующей формуле: $6/N$);

— третий столбец — истинные значения функции Y нелинейного преобразования (Y=1/(1+EXP(X)));

— четвертый столбец — масштабированные значения функции ($Y*N$);

— пятый столбец (адрес памяти) получается преобразованием первого столбца в целочисленный двоичный код;

— шестой столбец (значения сохраняемые в ячейках памяти) получается преобразованием четвертого столбца в целочисленный двоичный код.

Функция Excel для преобразования в двоичный код =ОСНОВАНИЕ(ячейка;2;n) (подходит для чисел более 511).

1.2 Построить график функции (X, Y), сохранить в отчет, сравнить с сигмоидой. Сделать выводы.

1.3 Записать значения шестого столбца в текстовый файл с наименованием ram_file1.txt.

2. Создание блока ОЗУ для хранения функции нелинейного преобразования.

2.1 Запустить программу и создать новый проект для работы с VHDL Module. Поместить файл ram_file1.txt в папку проекта.

2.2 Скопировать и вставить в рабочее окно приведенный ниже программный код, заменив цифры соответствующими командами.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;
use std.textio.all; -- библиотека чтения текстовых файлов


entity init_mem is
generic ( data_width: natural:=4);
 Port ( clk : in  STD_LOGIC; -- сигнал тактирования
   we : in  STD_LOGIC; -- сигнал управления чтение/запись
   data_in : in  STD_LOGIC_VECTOR ( data_width-1 downto 0); -- вход
    data : out  STD_LOGIC_VECTOR (data_width downto 0); -- выход
```

```
a1 : in      signed ( data_width downto 0); -- сигнал 1
 a2 : in      signed (data_width downto 0); -- сигнал 2
  a3 : in      signed ( data_width downto 0); -- сигнал 3
   a4 : in      signed ( data_width downto 0)); -- сигнал 4
end init_mem;

architecture Behavioral of init_mem is
```

1 – Объявить массив памяти

```
   impure function InitRamFromFile (InitRamFile : in string) return initmem is
     FILE ram_file          : text is in InitRamFile;    -- декларация файла чтения
памяти
     variable line_ram : line;
     variable ram       : initmem;
   begin
     for I in initmem'range loop
        readline (ram_file, line_ram);
        read (line_ram, ram(I));
     end loop;
     return ram;
   end function;

 signal ram: initmem := InitRamFromFile("ram_file1.txt"); -- сигнал чтения из
текстового файла
signal mem_data: std_logic_vector ( data_width-1 downto 0);
signal addr, mem_data_const, s13, s14: std_logic_vector (data_width-1 downto 0);
 signal sig1, sig2, sig3, sig4: signed( data_width*2+1 downto 0);
        signal sig12, sig22, sig32, sig42: signed( data_width+3 downto 0);
      signal sig11, sig21, sig31, sig41: signed(data_width-1 downto 0);
        signal s1: signed ( data_width+3 downto 0);
          signal s12 : signed (data_width-1 downto 0);
          signal   mem_data1,   mem_data_const1   :   std_logic_vector(data_width
downto 0);
```

```
        constant b1 : signed (data_width downto 0) :="01110"; -- объявление константы и присваивание ей значения
        constant b2 : signed (data_width downto 0) :="01111";
        constant b3 : signed (data_width downto 0) :="00010";
        constant b4 : signed (data_width downto 0) :="00101";
constant b5 : signed (data_width+3  downto 0) :="00101100";

        attribute RAM_STYLE : string; -- Block/Signal (RAM_STYLE) включает в себя шаблон варианта записи атрибута RAM_STYLE, позволяющего учитывать в процессе синтеза определенного сигнала реализацию макросов элементов оперативной памяти на базе модулей Block RAM
        attribute RAM_STYLE of ram : signal is "BLOCK";

        begin

        sig1   <= a1 * b1; -- умножение сигнала на весовой коэффициент
        sig11 <= sig1(data_width*2 downto data_width+1); -- возврат к разрядности исходных данных без знака, отбрасыванием младших разрядов
        sig12(data_width-1 downto 0)<=sig11; -- заносится результат в вектор большей размерности для контроля эффекта переполнения при сложении
        sig12 (data_width+3 downto data_width)<="0000" when (sig1(data_width*2+1)='0')  else "1111"; -- замена нулями 4 старших разряда, когда старший бит вектора произведения равен 0 (положительное число), иначе четырьмя единицами

        sig2   <= a2 * b2;
        sig21 <= sig2(data_width*2 downto data_width+1);
        sig22(data_width-1 downto 0)<=sig21;
        sig22 (data_width+3 downto data_width)<="0000" when (sig2(data_width*2+1)='0')  else "1111";

        sig3    <= a3 * b3;
        sig31 <= sig3(data_width*2 downto data_width+1);
```

```
sig32(data_width-1 downto 0)<=sig31;
sig32 (data_width+3 downto data_width)<="0000" when (sig3(data_width*2+1)='0') else "1111";

sig4  <= a4 * b4;
sig41 <= sig4(data_width*2 downto data_width+1);
sig42(data_width-1 downto 0)<=sig41;
sig42 (data_width+3 downto data_width)<="0000" when (sig4(data_width*2+1)='0') else "1111";

s1  <= sig12 + sig22 + sig32 + sig42 + b5 ; -- суммирование произведений

-- преобразование сигнала с выхода сумматора для использования его в качестве адреса ячейки памяти

s12 <= s1(data_width+2 downto 3); -- возврат к размерности исходных данных (отбрасывание знакового разряда и младших разрядов)
s13 <= std_logic_vector(s12);
s14  <= s13 when (s1(data_width+3)='0') else ((not s13) +1); -- преобразование адреса из дополнительного в прямой код (т.к. в памяти хранится только та часть функции, где аргумент положителен)

addr <= s14 ; -- адрес ячейки памяти для считывания
```

2 - чтение содержимого последней ячейки памяти

```
mem_data_const1 <= ext(mem_data_const, data_width+1);  -- введение нуля в старший разряд
```

3 - процесс работы блочной памяти

```
  if s1(data_width+3)='0' then
mem_data1 <= ext(mem_data, data_width+1);  -- введение нуля в старший разряд
data <= mem_data1;
    else
```

```
mem_data1 <= ext(mem_data, data_width+1);
data <= (mem_data_const1 - mem_data1); -- определение разности между значениями хранящимся в крайней ячейки памяти и считанным по адресу
end if;
end process;

end Behavioral;
```

3. Разработать алгоритм программы по коду, представленному в п.2. Разбить программу на блоки по функциональному назначению, дать наименование каждому блоку и привести краткое описание выполняемых в нем операций.

Пример: Блок 1 – Функция чтения текстового файла (рисунок 25). Реализует построчное чтение значений, хранящихся в текстовом файле. Используется, как способ записи данных в ячейки памяти.

```
1   impure function InitRamFromFile (InitRamFile : in string) return initmem is
      FILE ram_file      : text is in InitRamFile;  -- декларация файла чтения памяти
      variable line_ram : line;          -- присваивается значение переменной, строка читаемая из файла
      variable ram      : initmem;
    begin
      for I in initmem'range loop   -- цикл перебора
        readline (ram_file, line_ram); -- чтение строк файла
        read (line_ram, ram(I));
      end loop;
      return ram;
    end function;
```

Рисунок 25

4. Осуществить проверку программного кода и запустить симуляцию устройства, задав значения сигналов из таблицы вариантов заданий предыдущей лабораторной работы.

Перед запуском симуляции установить значения управляющих сигналов (*clk, we*) соответствующими процессу чтения данных из памяти (рисунок 26).

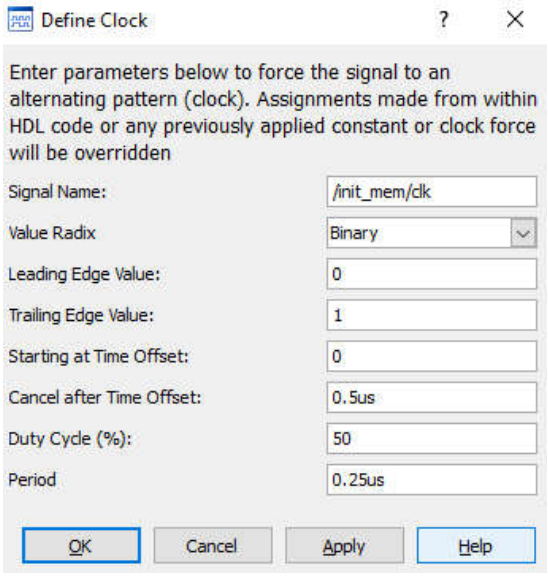


Рисунок – 26

5. Проанализировать результаты симуляции и сохранить в отчет, убедиться в правильности функционирования блока памяти, пояснить методику проверки.

6.2 Форма отчета обучающегося о выполненной лабораторной работе

Отчёт должен быть оформлен в соответствии с требованиями ГОСТ и содержать:

1. Титульный лист;

2. Оглавление;

3. Наименование работы, цель исследований;

4. Программу на языке описания аппаратуры (HDL) с комментариями (необходимо выделить вставленные самостоятельно недостающие блоки программы);

5. Схемы спроектированного устройства на разных уровнях;

6. Результаты моделирования схемы и отчет синтеза устройства;

7. Алгоритм программы, функциональные блоки и их описание.

9. Ответы на контрольные вопросы;

10. Перечень литературы, использованной при подготовке и выполнении работы (ссылки на используемую техническую документацию).

6.3 Контрольные вопросы

Дать письменные ответы на приведенные ниже вопросы.

1. Что такое программируемая логическая интегральная схема?

2. Как и в какой части программного кода объявляется блочная память VHDL?

3. Какая команда используется для чтения содержимого ячейки памяти?

4. Какие способы записи содержимого памяти существуют?

5. Чем отличается ОЗУ от ПЗУ с точки зрения функционирования и реализации на языке VHDL?

6. Какое оборудование используется для реализации блочной памяти на ПЛИС? Где можно ознакомиться с данной

информацией?

7. Почему в памяти хранится только часть нелинейной функции?

8. Какое свойство нелинейной функции позволяет реализовать данный способ?

9. Опишите последовательность создания функции в Excel.

10. Как обратиться к ячейкам памяти?

7. Лабораторная работа №7 «Чтение и запись текстовых файлов»

7.1 Алгоритм работы

1. Создать и сохранить в папку нового проекта текстовые файлы с соответствующими названиями:

input_vectors1.txt, input_vectors2.txt, output_results.txt.

Получить от преподавателя значения слагаемых, записать и сохранить их в соответствующих текстовых файлах.

2. Скопировать и вставить в рабочее окно приведенный ниже программный код, заменив цифры соответствующими командами.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
```

1 – подключение специализированных библиотек;

```
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;
```

```
entity example_file_io_tb is
end example_file_io_tb;
architecture behave of example_file_io_tb is
```

2 – объявление текстовых файлов;

```
 constant c_WIDTH : natural := 2;
signal r_ADD_TERM1 : std_logic_vector(c_WIDTH-1 downto 0)
:= (others => '0');
signal r_ADD_TERM2 : std_logic_vector(c_WIDTH-1 downto 0)
:= (others => '0');
        signal w_SUM       : std_logic_vector(c_WIDTH downto 0);
signal  r_ADD_TERM13  :  std_logic_vector(c_WIDTH-1  downto
0) := (others => '0');
 signal  r_ADD_TERM23  :  std_logic_vector(c_WIDTH-1  downto
0) := (others => '0');
        signal w_SUM1 : std_logic_vector(c_WIDTH-1 downto 0);
        signal w_SUM2 : std_logic_vector(c_WIDTH-1 downto 0);
        signal rst:     std_logic := '1';
       signal clk:     std_logic := '0';
        signal eor:     std_logic;

begin
 process (rst, clk)
```

3 – объявление переменных, для считывания и записи строки текстовых файлов;

```
variable v_ADD_TERM1 : std_logic_vector(c_WIDTH-1 downto 0);
variable v_ADD_TERM2 : std_logic_vector(c_WIDTH-1 downto 0);
        variable v_SUM : std_logic_vector(c_WIDTH-1 downto 0);
variable cnt:          integer range 0 to 160 := 0; -- defaults to 0

begin
    if rst = '1' then
    eor <= '0';
    elsif rising_edge(clk) then
    if cnt < 100 then
            if cnt = 0 then
```

4 – открытие текстовых файлов для чтения исходных данных и записи результата;

```
    eor <= '1';
    end if;
```

5 – чтение из текстового файла значения первого слагаемого;

```
r_ADD_TERM1 <= v_ADD_TERM1;
```

6 – чтение из текстового файла значения второго слагаемого;

```
r_ADD_TERM2 <= v_ADD_TERM2;
v_SUM :=   v_ADD_TERM1 + v_ADD_TERM2;
w_SUM2 <=v_SUM;
```

7 – запись в текстовый файл результата суммирования;

```
        cnt := cnt + 1;
        end if;
        if cnt = 100 then
```

8 **– закрытие текстовых файлов;**

```
            eor <= '0';
        end if;
      end if;
    end process;
  w_SUM1 <=  r_ADD_TERM1 + r_ADD_TERM2;
end behave;
```

3. Добавить комментарии к командам, составить алгоритм работы программы, разбив на функциональные блоки. Кратко описать назначение каждого блока.

4. Проверить правильность кода и открыть окно симуляции, нажав на пиктограмму  в окне процессов (рисунок 27). Установить значение rst равным нулю (в Force constant), clk (в разделе Force clock) – в соответствии с рисунком 28.

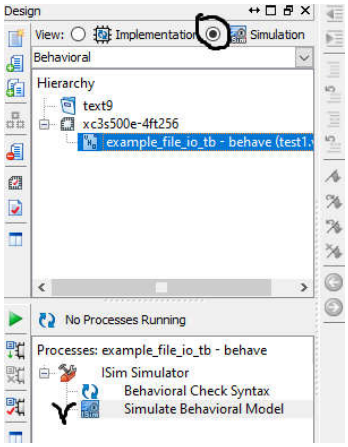


Рисунок 27

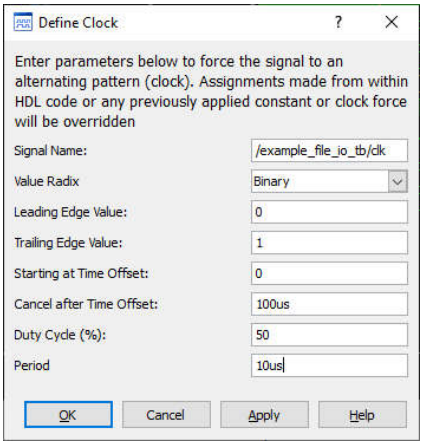


Рисунок 28

Запустить симуляцию, нажав на соответствующую пиктограмму , предварительно установив время просмотра в соответствии с рисунком 29.

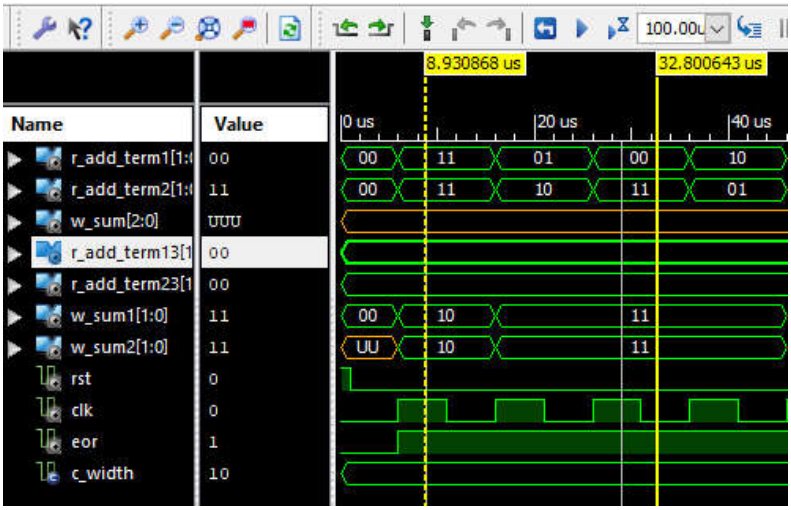


Рисунок 29

5. Проанализировать полученную временную диаграмму, проверить правильность значений, сохраненных в файле output_results.txt. Сделать выводы. В случае необходимости скорректировать программный код, установки режима симуляции, отразить в отчеты результаты корректировки.

7.2 Форма отчета обучающегося о выполненной лабораторной работе

Отчёт должен быть оформлен в соответствии с требованиями ГОСТ и содержать:

1. Титульный лист;

2. Оглавление;

3. Наименование работы, цель исследований;

4. Программу на языке описания аппаратуры (HDL) с комментариями (необходимо выделить вставленные самостоятельно недостающие блоки программы);

5. Результаты моделирования схемы, значения слагаемых и результатов суммирования, сохранённые в текстовый файл, сведенные в результирующую таблицу отчета;

6. Алгоритм программы, функциональные блоки и их описание.

7. Результаты корректировки программного кода.

8. Ответы на контрольные вопросы;

9. Перечень литературы, использованной при подготовке и выполнении работы (ссылки на используемую техническую документацию).

7.3 Контрольные вопросы

Дать письменные ответы на приведенные ниже вопросы.

1. Чем отличаются символы языка VHDL := и <=, какой из них применяется в процессах?

2. Какой альтернативный вариант программной реализации чтения данных из файла для данной задачи вы можете предложить?

3. Назовите отличие переменных от сигналов?

4. Есть ли в программном коде лишние командные строки, какие из них можно убрать, не изменив результатов моделирования?

5. С какой целью в программном коде использован оператор if?

6. Запишите программный код для закрытия текстового файла.

7. Как и в какой части программного кода объявляются текстовые файлы?

8. Как объявить переменные, для считывания и записи строки текстовых файлов?

9. Запишите программный код для открытия текстовых файлов с целью чтения исходных данных и записи результата.

10. Запишите программный код чтения из текстового файла значения первого слагаемого.

Заключение

По завершению лабораторных работ обучающийся должен

знать:

- основы представления данных в системах автоматизированного проектирования;
- источники информации, посвященные вопросам проектирования на ПЛИС;
- основные средства проектирования на ПЛИС;
- описание, функциональные возможности, интерфейс, методы работы в системах проектирования ПЛИС;
- алгоритмы анализа схем в системах автоматизированного проектирования ПЛИС;

уметь:

- выбирать и использовать информационно-коммуникационные технологии при поиске информации;

- применять методы поиска, хранения, обработки, анализа и представления данных с использованием современных информационных технологий;

- находить и анализировать описания программных пакетов для работы с данными и САПР;

- выбирать соответствующую задачам проектирования САПР;

- создавать имитационные модели вычислительных систем;

- математически моделировать цифровые устройства;

владеть:

- навыками работы в сети интернет, с электронными библиотеками и техническими описаниями программных продуктов;

- навыками построения электрических принципиальных схем в САПР в соответствии с НД, оформления отчетной документации с применением информационных технологий;

- навыками выбора адекватной решаемой задачи САПР;

- навыками анализа функциональных возможностей и интерфейса САПР;

- синхронным, асинхронным, сквозным и событийным моделированием;

- навыками представления данных в современных программных пакетах (Excel);

- навыками построения электрических принципиальных схем в САПР;

- технологиями автоматизированного проектирования ЭС;

- навыками моделирования в САПР;

- навыками обработки и представления результатов работы.