Министерство образования и науки Российской Федерации
Федеральное агентство по образованию
Владимирский государственный университет
Кафедра вычислительной техники

Интегрированные САПР

Методические указания к лабораторным работам

Составители
В.Н. ЛАНЦОВ
Е.В. ГАЛИЧЕВ
М.А. ТРОФИМОВ

Владимир 2005

УДК 658.512.22.01.56
ББК 32.973
И 73

Рецензент
Доктор технических наук, профессор
Владимирского государственного университета
И.Е. Жигалов

Печатается по решению редакционно-издательского совета
Владимирского государственного университета

Интегрированные САПР: метод. указания к лабораторным ра-
И 76 ботам / сост. : В. Н. Ланцов, Е. В. Галичев, М. А. Трофимов; Владим. гос. ун-т. – Владимир: Ред.-издат. комплекс ВлГУ, 2005. – 32 с.

Содержат методические указания для изучения маршрута проектирования электронных устройств в базисе FPGA. В качестве типичной системы автоматизированного проектирования устройств в данном базисе использована САПР Xilinx Foundation Series V4.1. Содержат 4 лабораторные работы, охватывающие весь VHDL маршрут проектирования устройств на ПЛИС.

Предназначены для студентов специальностей 2201 – вычислительные машины, комплексы, системы и сети.

Табл. 2. Ил. 8. Библиогр.: 3.

УДК 658.512.22.01.56
ББК 32.973

ЛАБОРАТОРНАЯ РАБОТА №1

ИЗУЧЕНИЕ ОБЩЕГО VHDL МАРШРУТА ПРОЕКТИРОВАНИЯ УСТРОЙСТВ НА ПЛИС И ЗНАКОМСТВО СО СРЕДОЙ ПРОЕКТИРОВАНИЯ САПР ПЛИС XILINX FOUNDATION SERIES V4.1

Цель работы: изучение VHDL маршрута проектирования электронных устройств в базисе ПЛИС; знакомство со средствами проектирования САПР САПР ПЛИС Xilinx Foundation Series V4.1; изучение отчетов реализации проекта; временное моделирование полученного устройства.

1.1. Основные теоретические и технические сведения

Общий маршрут проектирования устройств на ПЛИС

Процесс проектирования устройств на ПЛИС включает в себя несколько шагов. В рамках лабораторной работы не акцентируется внимание на этапы, свойственные любому проектированию, такие как анализ ТЗ, проработка алгоритмов реализации, разработка и декомпозиция структур и т.д. Маршрут проектирования в САПР ПЛИС начинается с описания проекта. Описание проекта в САПР ПЛИС Xilinx Foundation Series может быть выполнено с помощью языка описания аппаратуры (VHDL, Verilog, ABEL), принципиальных схем или с помощью диаграмм состояний. В этом курсе лабораторных работ рассматривается только VHDL маршрут проектирования, как наиболее перспективный. На рис. 1 представлен общий HDL маршрут проектирования.

После окончания процесса кодирования проекта необходимо выполнить функциональное моделирование для выявления допущенных при описании ошибок. Моделирование исходного кода выполняется достаточно быстро, поэтому рекомендуется его выполнять для больших проектов. Для малых проектов данный этап необязателен. Следует заметить, что в большинстве случаев после синтеза требуется произвести некоторые изменения в описании проекта, поэтому моделированию исходного кода на начальной стадии проектирования обычно выделяется немного времени.

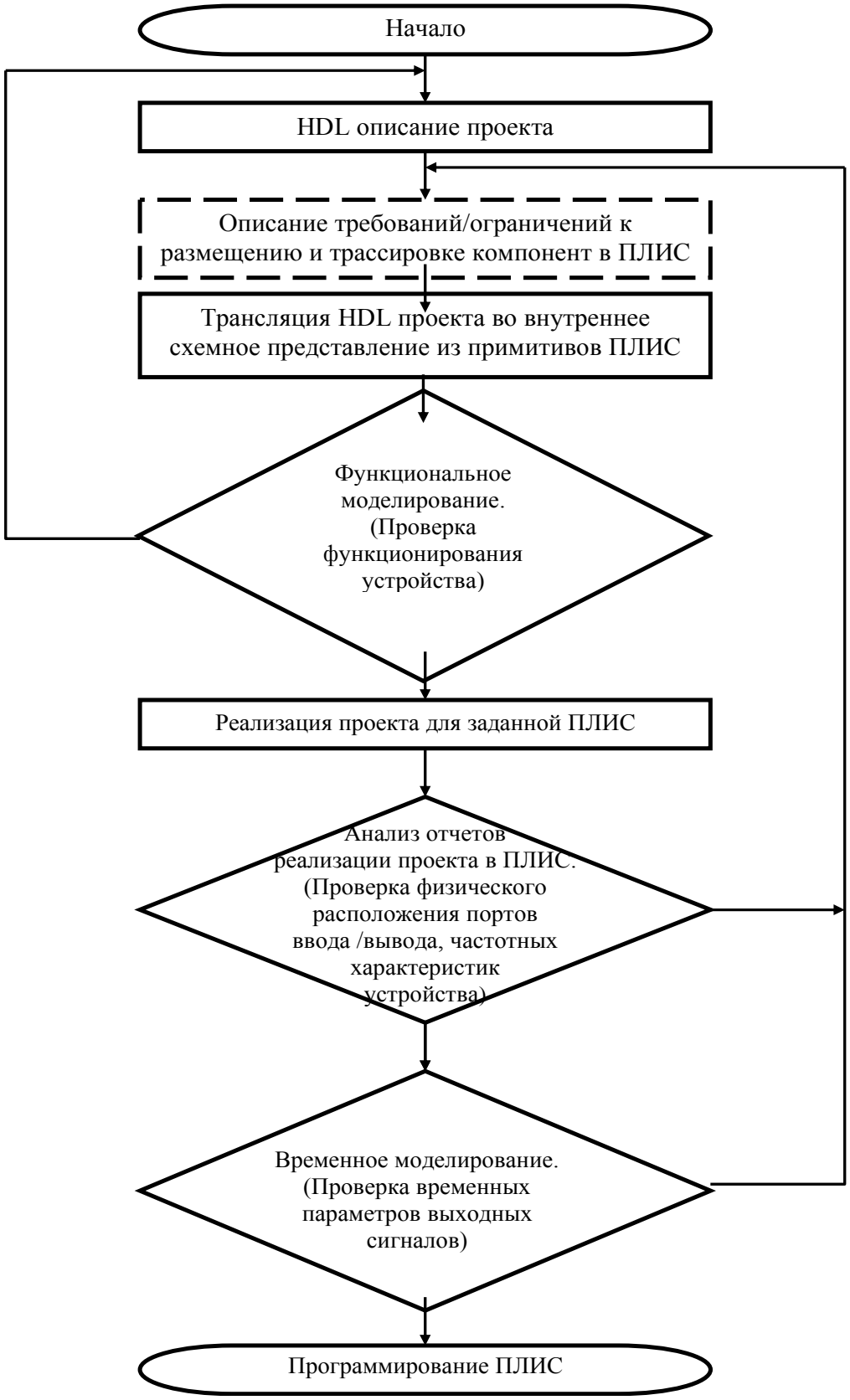


Рис. 1. Общий маршрут проектирования ПЛИС

После синтеза проекта, размещения и трассировки необходимо выполнить повторное моделирование. Этот шаг является необходимым в маршруте проектирования и предназначен не только для верификации функционирования проекта. Моделирование после размещения дает информацию о временных процессах в устройстве с учетом всех задержек. Если временные характеристики (задержки в схеме, синхронизация процессов, тактовая частота и др.) не удовлетворяют запросам пользователя, то необходимо вернуться на несколько шагов назад и внести необходимые коррективы в проект: уточнить спецификации, изменить описание проекта, выбрать другую технологическую базу для реализации устройства, задать другие ограничения на этапе синтеза и т.д.

В результате успешного прохождения всех этапов проектирования ПЛИС получается файл загрузки, который можно использовать для программирования кристалла.

1.2. Менеджер проектов САПР Xilinx Foundation Series 3.1

Менеджер проектов – это средство управления проектами в САПР. Из этой программы доступны все средства ввода, реализации, анализа и синтеза, а также отчеты о проектах.

Рабочая область окна менеджера проектов состоит из трех основных частей (рис. 2):

- область просмотра деревьев проекта (Hierarchy Browser);
- область размещения маршрута проектирования (Project Flowchart);
- окно сообщений (Message Window).

1.2.1 Дерево проекта и дерево реализаций проекта

Область просмотра деревьев проекта содержит закладки Files просмотра иерархической структуры проекта и Version для управления версиями реализаций проектов.

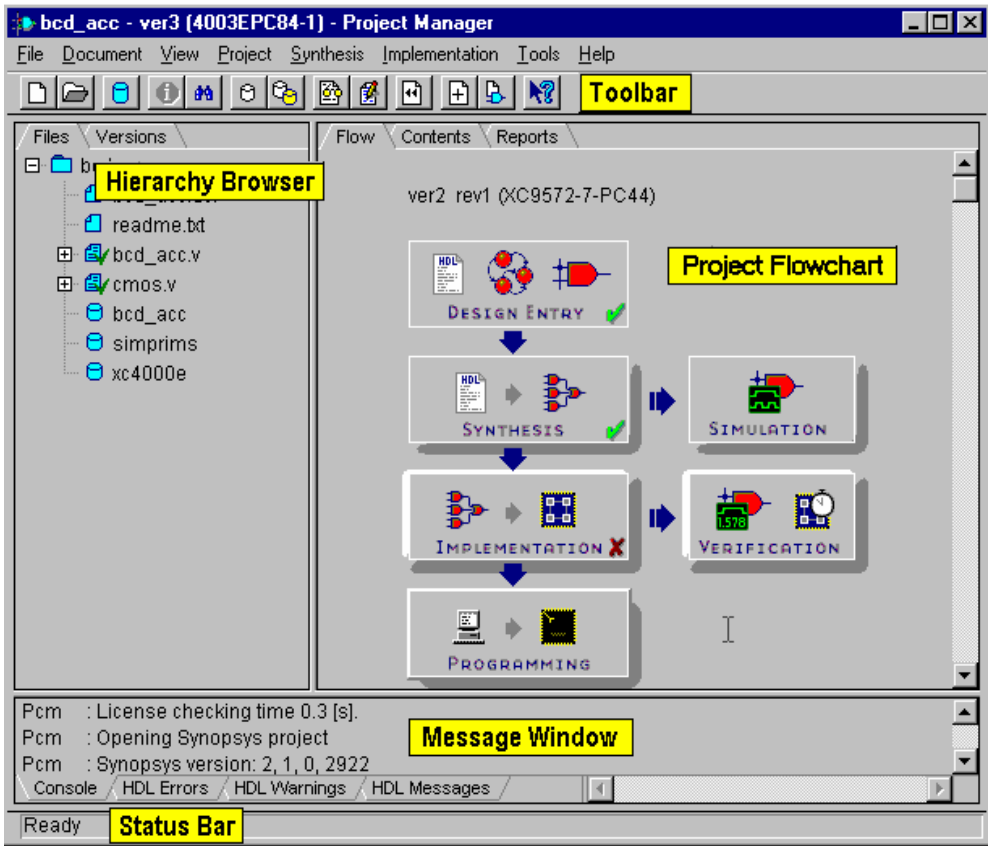


Рис. 2. Окно менеджера проектов

Закладка ***Files*** (рис. 3) отображает древовидную структуру проекта, содержащую:

- *файлы характеристик проекта* (*PDF*-файлы), содержащие информацию о текущем состоянии и составе проекта (файлах проекта, используемых библиотеках и т.д.);
- *документы проекта* с их внутренней иерархией – ключевые файлы каждого проекта, содержащие описания, характеристики и решения проекта, полученные с помощью схемного редактора, диаграмм состояния или HDL-кода;
- *внешние файлы*, подключенные к проекту;
- *библиотеки проекта* – набор библиотек, подключенных к данному проекту, в состав которого входят рабочая библиотека проекта (рабочая директория, автоматически создается при создании проекта), системные библиотеки и добавленные пользовательские библиотеки (редко);
- созданные пользователем VHDL библиотеки.

Иконки иерархического дерева проекта:

– файл описания проекта (находится всегда в вершине дерева);

– различные текстовые файлы;

– HDL-файлы (Verilog, VHDL), при анализе которых не было выявлено синтаксических ошибок;

– HDL-файлы (Verilog, VHDL), анализ которых выявил одну или более синтаксических ошибок;

– HDL-файлы (Verilog, VHDL), для которых анализ не проведен ни разу, либо в них были внесены изменения;

– файлы диаграмм состояния, анализ которых не выявил синтаксических ошибок;

– файлы диаграмм состояния, анализ которых выявил одну или более синтаксических ошибок;

– файлы диаграмм состояния, для которых анализ не проведен ни разу, либо в них были внесены изменения;

– схемы, имеющие обновленный список цепей в формате EDIF;

– схемы, требующие обновления списка цепей в формате EDIF;

– иерархический макрос;

– ABEL модули;

– компонент библиотеки;

– VHDL библиотека.

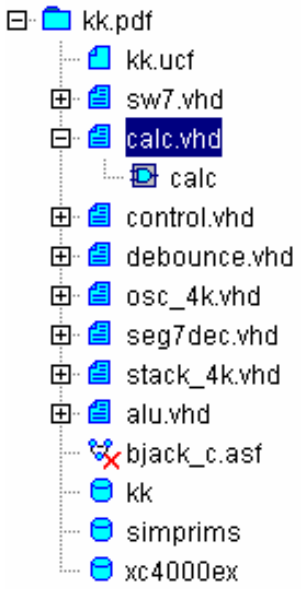


Рис. 3. Пример дерева проекта

Закладка ***Versions*** (рис. 4) отображает иерархическое дерево реализаций проекта в кристалле. Для только что созданного проекта это дерево пустое. Так как каждый проект может иметь несколько реализаций в разных кристаллах, дерево реализации может содержать множество версий реализаций проекта, для дальнейшей работы можно выбрать любую из них. Из каждой версии можно сделать очередной вариант (revision), отличающийся от нее стратегией реализации и определенными пользователем особенностями. При создании нового варианта реализации проекта пользователь может в первую очередь изменять уровень размещения и трассировки элементов в кристалле (от быстрого размещения по времени до глубокой проработки всех особенностей), а

также настраивать параметры реализации, моделирования и конфигурирования (не рекомендуется неопытным пользователям).

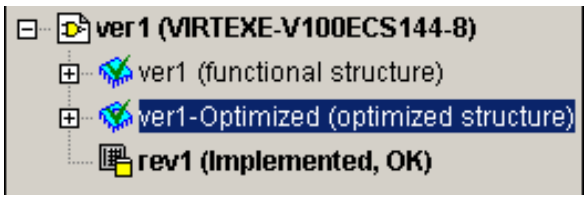


Рис. 4. Пример дерева реализаций проекта

Дерево реализаций содержит иерархию проекта, представляющую иерархию HDL-источников или схем. Элементы иерархии HDL-источников представляются иконками, отражающими их тип и состояние (аналогичны иконкам дерева проекта).

Иконки дерева реализаций для схемных источников:

– вершина дерева, обозначающая схематическое описание проекта;

– версия проекта;

– функциональная структура, полученная в результате синтеза, без оптимизации;

– функциональная структура, полученная в результате синтеза, без оптимизации; знак вопроса показывает, что необходимо заново выполнить синтез, т.к. исходный код был изменен;

– структура, полученная в результате выполнения фазы оптимизации процесса синтеза; оптимизация выполнена с предупреждениями;

– структура, полученная в результате выполнения фазы оптимизации процесса синтеза; оптимизация выполнена без предупреждений и ошибок;

– вариант (revision) версии реализации проекта.

1.2.2. Диаграмма маршрута проектирования

Область размещения маршрута проектирования содержит графическое представление шагов процесса проектирования и показывает пользователю необходимую последовательность операций.

При запуске очередной операции с помощью Диаграммы маршрута проектирования менеджер проектов автоматически контролирует преобразование входных и выходных данных (файлов) между приложениями. Если запускать операции из меню Tool, преобразования данных не происходит.

Кроме того, существенным преимуществом использования Диаграммы маршрута проектирования является автоматическое отслеживание выполненных шагов проектирования. Например, если при созданном HDL-файле нажать на иконку диаграммы маршрута, обозначающую Временное моделирование, Менеджер проекта автоматически выполнит следующие шаги: синтез «synthesis» (разработку и оптимизацию), реализацию «implementation» (создание списка цепей, карты, размещение, трассировку, генерацию временного списка цепей) и запустит Временное моделирование с уже сгенерированным списком цепей в качестве входных параметров. Визуально пройденные шаги маршрута проектирования индицируются зеленой галочкой внизу соответствующей иконки диаграммы маршрута.

Каждый тип проекта имеет свой специфический маршрут проектирования. Существуют два типа проекта:

1) описание (задание) проекта с помощью HDL-кода;
2) схематическое описание проекта.

Тип проекта задается при его создании (рис. 5.) с помощью установки переключателя Flow (ввод) в соответствующее положение:

1) Schematic – для схематического описания проекта;
2) HDL – для описания проекта с помощью HDL-кода.

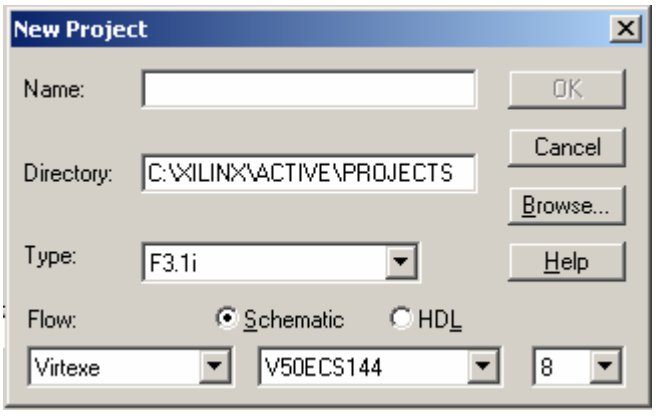


Рис. 5. Создание нового проекта

В зависимости от выбранного типа описания (задания) проекта выделяют два типа маршрута проектирования (рис. 6, 7):

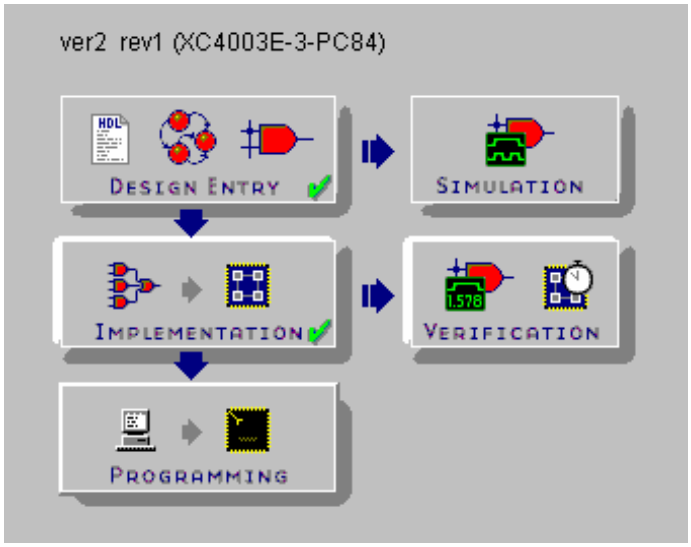


Рис. 6. Маршрут проектирования при схемном описании проекта

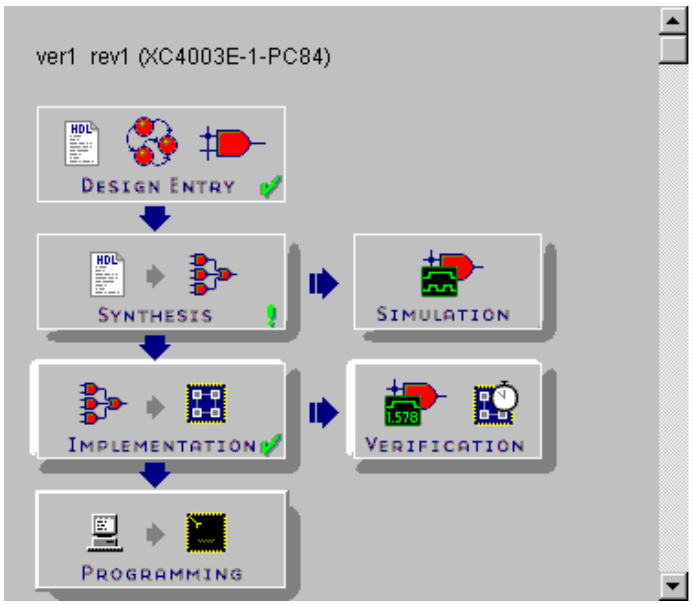


Рис. 7. Маршрут проектирования при использовании HDL-кода

При задании проекта с помощью HDL-кода в маршруте проектирования после этапа Design Entry (задание проекта) появляется этап Synthesis (синтез), который отсутствует в маршруте проектирования для второго типа проекта. Этот этап нужен для преобразования HDL-описания в функциональную модель.

1.2.3. Окно сообщений

Окно сообщений содержит следующие закладки для различных классов сообщений:

- ***Console*** отображает информацию о текущем процессе;
- ***HDL Errors*** отображает информацию об ошибках в HDL-файле во время его анализа;
- ***HDL Warning*** отображает предупреждающие сообщения при анализе HDL-файла;
- ***HDL Messages*** отображает сообщения другого рода при анализе HDL-файла.

1.2.4. Альтернативные пути запуска операций проектирования

Кроме диаграммы маршрута проектирования Менеджер проекта имеет альтернативные методы запуска программ Foundation Series:

- *Иерархическое дерево* позволяет открыть любой документ проекта или внешний файл на его сервере приложений.
- *Меню Tools* содержит команды, которые запускают основные операции Foundation Series, сгруппированные в следующие подменю: *Design Entry*, *Physical Implementation*, *Simulation*.

1.3. Варианты заданий

В VHDL-редакторе ввести нижеследующие описания устройств в соответствии с вариантом. Выполнить работу в соответствии с п. 1.3.

1. Сумматор двух беззнаковых целых шестнадцатибитных чисел.

```
-- 16-bit unsigned adder
library IEEE;
use IEEE.std_logic_1164.all,IEEE.std_logic_arith.all,
IEEE.std_logic_unsigned.all;

entity Add16 is port (
        Clk: in STD_LOGIC;
        A: in STD_LOGIC_VECTOR(15 downto 0);
        B: in STD_LOGIC_VECTOR(15 downto 0);
        C: out STD_LOGIC_VECTOR(15 downto 0);
        OVR: out STD_LOGIC);
end Add16;
```

```
architecture Add16_arch of Add16 is
  signal sC: STD_LOGIC_VECTOR(16 downto 0);
begin
  process(Clk)
  begin
     if Clk'Event and Clk='1' then
          sC <= ext(A,17) + ext(B,17);
     end if;
  end process;
  C <= sC(15 downto 0);
  OVR <= sC(16);
end Add16_arch;
```

2. Счетчик с дешифратором

```
-- counter with decoder
library IEEE;
use IEEE.std_logic_1164.all,IEEE.std_logic_arith.all,
IEEE.std_logic_unsigned.all;

entity CntDec is port (
        Clk: in STD_LOGIC;
        Rst: in STD_LOGIC;
        C: in STD_LOGIC;
        D: out STD_LOGIC_VECTOR(15 downto 0));
end CntDec;

architecture CntDec_arch of CntDec is
signal Cnt: STD_LOGIC_VECTOR(15 downto 0);
begin
  process(Clk,Rst)
  begin
     if Rst = '1' then
        Cnt <= x"0000"
     elsif Clk'Event and Clk='1' then
          Cnt <= Cnt + 1;
     end if;
  end process;

     mD: generate
          for i in 0 to 15 loop
               D(i) <= '1' when i = Cnt else '0';
          end loop;
     end generate;
  end CntDec_arch;
```

3. Компаратор

```
-- comparator
library IEEE;
use IEEE.std_logic_1164.all,IEEE.std_logic_arith.all,
IEEE.std_logic_unsigned.all;

entity Comparator is port (
        Clk: in STD_LOGIC;
        CE: in STD_LOGIC;
        Val0: in STD_LOGIC_VECTOR(15 downto 0);
        Val1: in STD_LOGIC_VECTOR(15 downto 0);
        Dev: out STD_LOGIC_VECTOR(15 downto 0);
        Res: out STD_LOGIC);
end Comparator;

architecture Comparator_arch of Comparator is
  signal s0Dev, s1Dev: STD_LOGIC_VECTOR(16 downto 0);
begin
  process(Clk)
  begin
     if Clk'Event and Clk = '1' then
         if CE = '1' then
          s0Dev <= ext(Val1,17) - ext(Val0,17);
          s1Dev <= ext(Val0,17) - ext(Val1,17);
         end if;
     end if;
  end process;

  Dev <= s0Dev(15 downto 0) when s0Dev(16) = '0'
         else s1Dev(15 downto 0) ;
  Res <= s1Dev(16);

end Comparator_arch;
```

4. Устройство, прореживающее поток чисел в 4 раза с усреднением

```
-- Sampler
library IEEE;
use IEEE.std_logic_1164.all,IEEE.std_logic_arith.all,
IEEE.std_logic_signed.all;
```

```
entity Sampler is port (
        Clk: in STD_LOGIC;
        CE: in STD_LOGIC;
        Din: in STD_LOGIC_VECTOR(7 downto 0);
        Dout: out STD_LOGIC_VECTOR(7 downto 0);
        Res: out STD_LOGIC);
end Sampler;

architecture Sampler_arch of Sampler is
  signal Acc: STD_LOGIC_VECTOR(9 downto 0);
  signal Cnt: integer range 0 to 3;
begin
  process(Clk)
  begin
     if Clk'Event and Clk = '1' then
         if CE = '1' then
          Cnt <= Cnt + 1;
          if Cnt = 0 then
            Dout <= Acc(9 downto 2);
            Acc <= sxt(Din,10);
          else
            Acc <= Acc + sxt(Din,10);
          end if;
         end if;
     end if;
  end process;

end Sampler_arch;
```

1.4. Порядок выполнения работы

1. Создать новый проект ПЛИС с VHDL маршрутом проектирования.
2. Имена файлов.
3. Выполнить реализацию проекта для устройств add16 и filter26.
4. Изучить генерируемые системой отчеты.
5. Проанализировать затраченные ресурсы кристалла.

1.5. Содержание отчета

1. Краткое описание средств ввода проекта, реализации и верификации.
2. Описание устройства и интерфейса подключения.
3. Результаты реализации устройств в ПЛИС.
4. Временные диаграммы работы устройства.

1.6. Контрольные вопросы

1. Каковы основные этапы маршрута проектирования электронных устройств в базисе ПЛИС?

2. Является ли функциональное моделирование исходного кода обязательным этапом проектирования? Аргументируйте ответ.

3. Какие этапы проектирования цифровых устройств предшествуют и не покрываются САПР Xilinx Foundation Series?

4. Где можно найти информацию:
   - о максимальной частоте работе устройства;
   - о максимальной задержке на комбинационной логике;
   - о максимальной задержке на трассах между ЛЭ;
   - о размещении портов I/O по физическим выводам кристалла;
   - об объеме устройства в примитивах данной архитектуры (Slice, Flip Flop, LUT и т.д)?

5. Как создавать несколько версий реализаций в одном проекте?

6. Как в системе изменяются параметры размещения и трассировки, какие для этого существуют параметры?

ЛАБОРАТОРНАЯ РАБОТА №2

ИЗУЧЕНИЕ ОСНОВ VHDL ОПИСАНИЯ ПРОЕКТА УСТРОЙСТВА

Цель работы: изучение средств проектирования САПР Xilinx; изучение основ языка VHDL.

2.1. Основные теоретические и технические сведения

Язык VHDL (Very high speed integrated circuits Hardware Description Language) является международным стандартом в области автоматизации проектирования цифровых систем. Это входной язык многих современных систем автоматизированного проектирования как заказных, так и программируемых логических интегральных схем (ПЛИС). VHDL предназначен, в первую очередь, для спецификации проектируемых систем и их моделирования на начальных этапах проектирования – алгоритмическом и логическом. С помощью VHDL можно моделировать электронные схемы с учетом реальных временных задержек.

VHDL – это мощный язык, позволяющий описывать поведение, т.е. алгоритмы функционирования цифровых систем, а также проводить иерархическое функционально-структурное описание систем, имеет средства для описания параллельных асинхронных процессов, регулярных структур. В то же время VHDL имеет все признаки языка программирования высокого уровня: возможность создания новых типов данных, широкий набор арифметических и логических операций и др.

2.1.1. Типы данных в языке VHDL

Для первого знакомства с VHDL рекомендуется использовать стандартные типы данных библиотреки IEEE.std_logic_1164:

std_logic;

std_logic vector;

integer.

Синтаксис подключения библиотеки:

```
library IEEE;
use IEEE.std_logic_1164.all;
```

VHDL имеет гибкие средства для описания пользовательских типов, и большое множество стандартных (см. документацию по VHDL).

2.1.2. Объекты

В VHDL существует три класса объектов: константы, переменные и сигналы. Объявление и использование констант и переменных аналогично использованию их в языках программирования.

Константа – это объект, который при создании получает определенное значение, которое нельзя изменить в процессе работы.

Значение *переменной* может изменяться в ходе работы (во время моделирования). Если начальное значение переменной не задано, то при создании ей присваивается наименьшее значение, возможное для заданного типа. Если объявлена переменная составного типа, то наименьшее значение присваивается каждой составляющей.

Сигналы – объекты, значения которых могут меняться и иметь временные параметры.

2.1.3. Выражения

Использование выражений в языке VHDL аналогично их использованию в языках программирования. *Выражение* – формула, содержащая операнды и операторы, их связывающие. Операндами могут быть сигналы, переменные, литералы, вызовы функций. Операторы могут быть трех типов: логические, арифметические, сравнения.

Логические операторы предназначены для выполнения операций с битовыми и логическими значениями и с массивами значений этих типов. К этим операторам относятся **and, or, not, nor, nand** и **xor**.

Операторы сравнения **=, /=, <, <=, >, >=** всегда имеют два аргумента одного и того же типа и дают результат типа boolean.

Арифметические операторы ***** и / предназначены для выполнения соответствующих операций над данными числовых и физических типов. Деление нацело (**mod**) и остаток от деления (**rem**) применяются только к данным целого типа. Абсолютное значение (**abs**) можно получить для лю-

бого числа. Возведение в степень (**) выполняется для числа любого типа, но показатель степени должен быть целым числом. Если же показатель степени меньше нуля, то в основании степени должно быть число с плавающей точкой.

Оператор конкатенации (**&**) достаточно часто используется при работе с массивами и строками для изменения порядка следования элементов.

Следует особо отметить, что язык VHDL – строго типизированный язык, поэтому в арифметических операциях нельзя смешивать данные разных типов.

В нижеследующей таблице представлены операторы и типы операндов для них.

Тип оператора	Оператор	Тип операндов	Тип результата
Логический	not, and, or, nand, nor, xor	bit, boolean	bit, boolean
Отношения	=, /=, >, <, >=, <=	любой тип	boolean
Арифметический	+, –, *, /, **, mod, rem, abs	целый, реальный, физический	целый, реальный, физический
Конкатенации	&	массив любого типа	массив любого типа

Исходя из контекста VHDL-кода, следует отличать оператор <= (назначение сигнала) и оператор <= (меньше либо равно). Следует также отличать унарные операции присвоения знака "+" и "–" от соответствующих бинарных операций сложения и вычитания.

2.1.4. Структура программы на VHDL

В языке VHDL каждый компонент проектируемого устройства представляется как объект проектирования. Для каждого объекта существует два типа описаний:

1) описание интерфейса;

2) архитектурное тело.

Интерфейс объекта описывает имя, режимы (направление) и типы каждого порта:

entity <имя интерфейса> **is**

[**generic**(<список настраиваемых параметров>);]

[**port**(<список портов>);]

end <имя интерфейса>;

Архитектурное тело может содержать в себе как описание структуры устройства или отдельного компонента, так и алгоритм его функционирования. В архитектурном теле могут содержаться объявления типов, констант, сигналов, компонентов, атрибутов, процедур и функций.

Следует заметить, что для каждого блока интерфейса может существовать несколько архитектурных тел.

Формат описания архитектурного тела:

architecture <имя архитектурного тела> **of** <имя интерфейса> **is**
    [<раздел описаний>]
begin
    <параллельные операторы>
end <имя архитектурного тела>;

2.1.5. Параллельные операторы

Оператор процесса – параллельный оператор, используемый в архитектурном теле и содержащий в себе только последовательные операторы. Формат описания процесса:

[метка]: **process** [(<список чувствительности>)]
        [<объявления>]
        begin
        <последовательные операторы>
    end process [метка];

В разделе объявлений могут быть описаны подпрограммы, типы и подтипы, константы, переменные.

Компоненты позволяют задать сложную иерархию устройства. При этом становится возможной независимая разработка отдельных частей проекта. Объявление компонента выглядит следующим образом:

component <имя компонента>
    [<описание настраиваемых параметров>]
    [<описание интерфейса компонента>]
end component;

Например, для компонента AND2 объявление будет таким:

```
component AND2 port(x1, x2: in bit; y: out bit);
end component;
```

Необходимым условием для моделирования и синтеза устройства является полное определение его составных частей. Перед использованием компонента в проекте он должен быть описан с помощью пары модулей интерфейс-архитектурное тело (entity-architecture), причем имя компонента и список его портов должны совпадать с именем и списком портов интерфейса.

В программе для обращения к конкретному компоненту используется следующая конструкция:

<метка>: <имя компонента>

[**generic map**(<настраиваемые параметры>);]

[**port map**(<фактические параметры интерфейса>);]

2.1.6. Последовательные операторы

Оператор присваивания значения переменной:

<имя переменной>:=<новое значение>;

Условный оператор позволяет реализовывать ветвление алгоритма за счет использования сложных условий:

if <условие 1> **then**

<последовательные операторы>

elsif <условие 2> **then**

<последовательные операторы>

…

elsif <условие N> **then**

<последовательные операторы>

else

<последовательные операторы>

end if;

Селективный оператор может использоваться в процессах, процедурах и функциях. Формат:

case <выражение> **is**

when <значение 1> => <последовательные операторы>

when <значение 2> => <последовательные операторы>

when <значение N> => <последовательные операторы>

end case;

В качестве <значения> может быть либо скалярное значение, либо несколько значений через знак "|", либо диапазон значений (с использованием **to** и **downto**), либо ключевое слово **others**, обозначающее остальные значения.

Операторы цикла позволяют реализовать циклическую обработку данных, как это реализуется в языках программирования:

1) [метка цикла]:

while <условие> **loop**

<последовательность операторов>

end loop [метка цикла];

2) [метка цикла]:

for <переменная цикла> **in** <диапазон значений> **loop**

<последовательность операторов>

end loop [метка цикла];

Внутри циклов могут использоваться дополнительные последовательные операторы:

next [метка цикла] [**when** <условие>];

exit [метка цикла] [**when** <условие>];

2.2. Варианты заданий

Описать следующие устройства на VHDL:

Вариант 1. Компонент распределенной памяти RAM64x4S, используя стандартный компонент кристалла Virtex-E RAM16x1S. Интерфейс подключения RAM64x4S подобен RAM16x1S, объем памяти – 64 слова, ширина слов записи/чтения – 4 бита.

Вариант 2*. Компонент, реализующий комплексное умножение. Ширина слов операндов реальных и мнимых частей по 8 бит, операнды знаковые. Для реализации компонента использовать умножитель, созданный в CORE Generator.

Вариант 3*. Компонент, вычисляющий квадрат знакового числа. Разрядность входных чисел – 8 бит. Для реализации компонента использовать умножитель, созданный в CORE Generator.

Вариант 4[1]*. Сглаживающий фильтр. Компонент выполняет усреднение с апертурой, равной четверти потока 8-битных знаковых чисел. Другими словами на каждое входное число компонент выдает среднее арифметическое четырех последних чисел. Накопление суммы организовать по

[1] *Для устройств ***вариантов 2, 3 и 4*** должны быть управляющие входы CE, DV. Сигнал CE определяет активность устройства (Clock Enable). Сигнал DV (Data valid) устанавливается в ‘1’ на период одного такта синхросигнала в момент, когда данные на входе устройства верны. Вычисленный результат должен сохраняться на выходе до момента появления следующего результата.

следующему алгоритму: последнее вошедшее число добавляется к аккумулятору, а число, пришедшее четырьмя числами ранее, вычитается из аккумулятора. Очередь задержки чисел организовать на сдвиговых регистрах SRL16E.

2.3. Порядок выполнения работы

1. Проработать алгоритм функционирования компонента.
2. Разработать структурную схему алгоритма.
3. Выполнить VHDL описание разработанной структуры.
4. Убедиться в работоспособности разработанного компонента с помощью функционального моделирования.

2.4. Содержание отчета

1. Интерфейс подключения разработанного компонента.
2. Описание функционирования компонента с необходимыми диаграммами сигналов.
3. Укрупненная структурная схема компонента.
4. VHDL описание компонента.

2.5. Контрольные вопросы

1. Какие типы данных определены в стандарте языка VHDL?
2. Какие типы операторов существуют в VHDL?
3. Какова структура описания устройства на языке VHDL?
4. Перечислите последовательные операторы. Приведите примеры.
5. Перечислите параллельные операторы. Приведите примеры.
6. Какие операторы могут использоваться в архитектурном теле?
7. Какие операторы могут использоваться в процессе?
8. Перечислите классы объектов, существующие в VHDL. Приведите примеры объявления и использования объектов в VHDL-описаниях.
9. Привести пример определения, описания и подключения компонента.

ЛАБОРАТОРНАЯ РАБОТА №3

РЕАЛИЗАЦИЯ КОНЕЧНЫХ АВТОМАТОВ В VHDL

Цель работы: познакомиться со способами описания конечных автоматов устройств в VHDL, научиться анализировать описание устройства для выделения его структуры реализации.

3.1. Основные теоретические и технические сведения

3.1.1. Описание проекта на основе диаграммы состояний

Широкое распространение в практике проектирования дискретных устройств получила модель конечного (абстрактного) автомата.

Конечный автомат К определяется как набор

$$К = (A, Z, W, \delta, \lambda, a_1),$$

где $A = \{ a_1, ..., a_q \}$ – множество (алфавит) состояний (имеются в виду внутренние состояния автомата);

$Z = \{ Z_1, ..., Z_N \}$ – множество входных сигналов (входной алфавит);

$W = \{ w_1, ..., w_M \}$ – множество выходных сигналов (выходной алфавит);

δ – функция переходов, определяющая состояние автомата в момент времени t+1 в зависимости от состояния автомата и входного сигнала в момент времени t, иначе говоря, $a_s=\delta(a_m,z)$, где a_m – состояние автомата в момент времени t; z – входной сигнал в момент времени t; a_s – состояние автомата в момент времени t+1;

λ – функция выходов. Если функция λ по состоянию автомата a_m и входному сигналу z_n, определяет значение выходного сигнала $w_m = \lambda (a_m, z_n)$, то конечный автомат называется *автоматом* ***Мили***, если же $w_m = \lambda (a_m)$, то конечный автомат называется *автоматом* ***Мура****;*

a_1 - начальное состояние автомата в момент времени t =0, a, $\in$ A.

Функционирование автомата происходит следующим образом: в дискретные моменты времени t = 0, 1, 2, 3,... на вход устройства поступает

входной сигнал – один из элементов множества Z, а на выходе появляется выходной сигнал – один из элементов множества W, в этот же момент времени t автомат из состояния a_m переходит в состояние a_s, в котором он будет находиться в момент времени t+1.

Разработаны различные формы задания конечных автоматов. Ниже-следующая таблица представляет собой пример задания конечного автомата.

Входные сигналы	Состояния			
	a_1	a_2	a_3	a_4
Z_1	a_2/w_1	A_2/w_1	a_1/w_2	a_1/w_4
Z_2	a_4/w_5	A_3/w_3	a_4/w_4	a_3/w_5

Алфавит состояний A = {a_1, a_2, a_3, a_4}, q = 1,...,4. Входной алфавит Z образуют сигналы z_1, z_2, т.е. Z = {z_1 , z_2 }, n = 1, 2. Выходной алфавит W образуют сигналы w_1, ..., w_5, т.е. W = {w_1 w_2 ,w_3 ,w_4, w_5}, m = 1,...,5. На пересечении строки z_n, и столбца a_q в таблице находится состояние a_s, в которое должен перейти автомат из состояния a_q под воздействием сигнала z_n. После косой черты в этой же графе таблицы указывается выходной сигнал, выдаваемый автоматом в состоянии a_q при поступлении на его вход сигнала z_n. Иначе говоря, данная таблица является способом задания функций δ, λ и называется *совмещенной таблицей переходов и выходов.*

Следующий VHDL-код реализует поведение конечного автомата, заданного в виде графа на рис. 8. Поведение автомата представляется в виде совокупности трех взаимодействующих состояний.

Для задания состояний автомата в архитектурном теле определяется перечислимый тип TState. Предполагается, что начальное состояние автомата равно State_A. Начальное состояние явно не указывается, так как при инициализации будет взят элемент нижней границы перечислимого типа – это и будет элемент State_A.

```
-- описание архитектуры устройства
architecture arh_FTM of FTM is
-- описание перечисляемого типа
type Tstate is (State_A, State_B, State_C)
signal State: TState;
begin
process(clk,reset) -- описание процесса функционирования
   if reset='1' then  -- асинхронный сброс
  State<=State_A;  -- установка состояния после сброса
```

```
   ELSIF clk'Event and clk='1' then
  Case State is -- выбор текущего состояния
     When State_A => <описание функционирования
                      в состоянии A>
          If <Условие 1> then
               State<=State_B; -- переход в состояние B
          End if;
          If <условие 4> then
               State<=State_C; -- переход в состояние C
          End if;
     When State_B => <описание функционирования
                      в состоянии B>
          If <Условие 2> then
               State<=State_A;
          End if;
          If <условие 3> then
               State<=State_C;
          End if;
     When State_C => <описание функционирования
                      в состоянии C>
          If <Условие 5> then
               State<=State_D;
          End if;
     When others => null;
  End case;
   end if;
end process;   --  завершение процесса функционирования
end arh_FTM;   --  Конец описания архитектуры
```

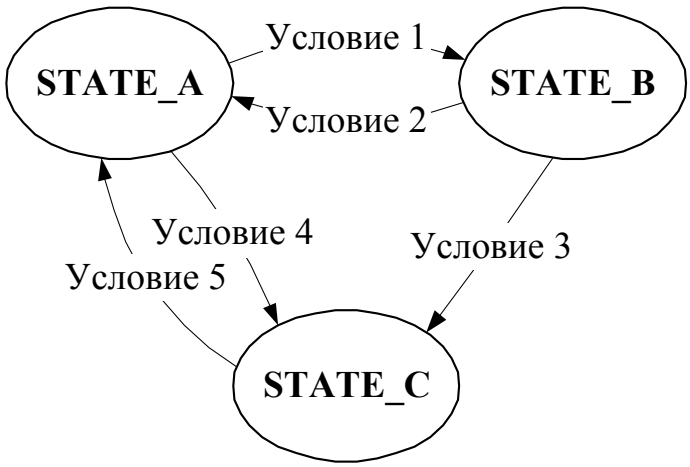


Рис. 8. Диаграмма состояний конечного автомата

3.1.2. Другие способы описания состояний устройства

В качестве сигнала, описывающего состояние устройства может быть использован не только перечисляемый тип, но и, например

STD_LOGIC_VECTOR. В этом случае состояние устройства можно представать в виде сдвигового регистра. Таким образом, переход из одного состояния в другое можно выполнять простой операцией сдвига. Обычно, при таком описании в сдвиговом регистре в любой момент находится одна единичка в одном из битов, однозначно определяющим состояние устройства. Единичка помещается в регистр стартовым событием устройства. Преимуществом данного способа является простота формирования управляющих сигналов от машины состояний к компонентам структуры устройства.

Описать машину состояний можно также, присвоив каждому состоянию свой уникальный номер, то есть в частном случае машина состояний может превратиться в обычный счетчик. Достоинством данного описания является компактность реализации самой машины состояний. К недостаткам данного способа можно отнести увеличение логических функций при формировании управляющих сигналов для компонент структуры при больших машинах состояний.

Выбор того или иного способа представления машины состояний определятся структурой устройства и целью оптимизации устройства по быстродействию либо площади реализации в кристалле.

3.2. Варианты заданий

Вариант 1. Создать VHDL описание контроллера игрового автомата, вытаскивающего игрушки. Задача контроллера состоит в управлении шаговыми двигателями автомата в соответствии с управлением играющего. Принцип работы автомата заключается в следующем: автомат входит в режим работы только после опускания монеты в принимающее устройство и цикл работы ограничен временем; пользователь может с помощью джойстика управлять в ограниченной плоскости захватывающим устройством; пользователь может нажать кнопку захвата игрушки из короба; захватывающее устройство по нажатию кнопки или по истечению времени делает попытку захватить игрушку из текущего положения в его плоскости движения и затем возвращается в точку старта, отпуская игрушку. В разрабатываемом контроллере должны быть реализованы все необходимые входные сигналы для обеспечения описанного алгоритма работы и выходные сигналы, обеспечивающие перемещение в плоскости захватывающего устройства, команды на захват и отпускание игрушки.

Вариант 2. Создать VHDL описание контроллера игрового автомата «Счастливый номер». Принцип работы автомата заключается в следующем: автомат постоянно последовательно перебирает числа от 0 до 999 с частотой 500 Гц, после опускания монеты в приемное устройство автомат отображает мгновенное состояние счетчика на цифровом индикаторе, в случае выпадения чисел определенных как выигрышные, автомат выдает выигрыш монетами в установленном размере для конкретного числа. Установить следующие размеры выигрышей: 1) если все три десятичные цифры одинаковые – 6 монет; 2) если старшая цифра равна 5, а две остальные одинаковые – 2 монеты; 3) если три цифры соседи и в порядке возрастания – 1 монета. Реализовать в контроле автомата встроенный генератор, счетчик, сигналы начала работы и отсчитывания монет.

Вариант 3. Создать VHDL описание контроллера игрового автомата «Счастливый номер». Играющий должен вовремя выпустить торпеду со своего корабля по линейно движущейся цели перед ним вдалеке цели. Имеется счетчик подбитых кораблей. Количество боеприпасов ограничено на игровой сеанс.

Вариант 4. Контроллер системы лифтов здания. В подъезде здания находится два лифта, управляемые одним контроллером. У контроллера на каждом этаже по две кнопки запроса лифта для спуска или подъема соответственно. Пусть здание имеет четыре этажа. На запрос должен реагировать тот лифт, который в данный момент свободен или движется по пути в том же направлении. Внутри каждого лифта имеется по четыре кнопки, соответствующие этажам и кнопка закрытия дверей – старта движения. Лифт должен останавливаться последовательно на всех указанных этажах.

3.3. Порядок выполнения работы

1. Проанализировать задание.
2. Выделить входные управляющие сигналы.
3. Определить основные компоненты устройства.
4. Выделить сигналы управления компонентами устройства.
5. Создать диаграмму состояний работы устройства.

6. Создать VHDL-описание контроллера устройства в соответствии с заданием.
7. Выполнить синтез проекта и функциональное моделирование.
8. Выполнить реализацию проекта и временной анализ.
9. Сопоставить результаты функционального и временного моделирования. Сделать выводы.

3.4. Содержание отчета

1. Задание на лабораторную работу.
2. Диаграмма состояний.
3. Обобщенная структурная схема устройства.
4. VHDL-описания устройств в соответствии с заданиями.
5. Результаты функционального и временного моделирования.
6. Выводы по работе.

3.5. Контрольные вопросы

1. Что такое конечный автомат?
2. Какие типы конечных автоматов существуют?
3. Какими способами в VHDL можно описать состояния устройства?
4. Какова общая структура программы на VHDL, реализующей автомат Мили?
5. Какова общая структура программы на VHDL, реализующей автомат Мура?

ЛАБОРАТОРНАЯ РАБОТА №4

ИЗУЧЕНИЕ СРЕДСТВ АНАЛИЗА И ОПТИМИЗАЦИИ РАЗМЕЩЕНИЯ УСТРОЙСТВ В ПЛИС

Цель работы: изучить средства САПР, позволяющие повысить качество размещения, предельные возможности конкретной реализации; научиться управлять трассировкой различных примитивов в кристалле; сделать выводы о влиянии качеством размещения и трассировки на предельные параметры функционирования устройства.

4.1. Основные теоретические и технические сведения

FloorPlaner – графическое средство, позволяющее полностью вручную или автоматизированно размещать логические элементы на топологической карте кристалла.

Возможности:

- работа как на высоком уровне иерархии (компоненты), так и с низкоуровневыми элементами (площадки ввода-вывода (I/O PADs), функциональные генераторы (LUT), тристабильные буфера, триггеры, блоки памяти и т.п.);
- захват и размещение сложных шаблонов, для работы с повторяющимися блоками;
- автоматическое распределение по столбцам и строкам;
- гибкое отображение соединений;
- поиск ЛЭ или узлов по имени;
- перестановка в иерархии устройства для упрощения топологической карты;
- группировка по соединению или по функциям блоков;
- возможность чтения файлов различных стандартов;
- привязка и разрыв RP макросов (RPM);
- поиск проблем размещения.

При работе с FloorPlaner’ом обычно используются четыре основные окна:

- **Design Hierarchy** – иерархическое дерево компонентов;
- **Design Nets** – отображает идентификаторы соединений;
- **Placement** – отображает расстановку элементов, выполненную в автоматическом режиме;

- **Editable FloorPlan** – чистый шаблон кристалла для размещения на нем компонентов.

4.1.1. Иерархическое дерево компонентов

Это окно служит для наглядного представления структуры устройства и удобства выбора определенных компонентов. Каждая ветвь выделяется своим цветом. Цвета могут быть переопределены с помощью пункта меню FloorPlaner->Меню->Edit->Colors.

Основные функции, доступные в этом окне: выбор компонентов или ЛЭ, навигация по уровням иерархии, поиск и сортировка по различным критериям, сборка в группу набора компонентов и/или ЛЭ. Доступ к этим функциям осуществляется через FloorPlaner->Меню->Hierarchy или меню, вызываемое по нажатию правой кнопки мыши.

4.1.2. Окно идентификаторов соединений (Nets)

Позволяет отобразить как все соединения, так и только связанные с текущим выбранным блоком. Выбор режима производится в меню, вызываемом по нажатию правой кнопки мыши.

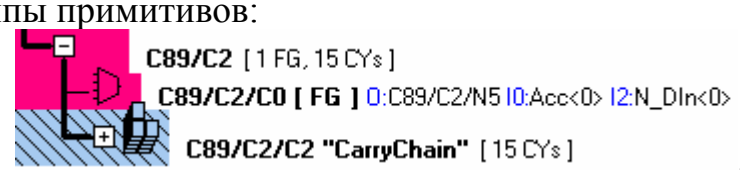
4.1.3. Окна расстановки элементов

Окно размещения (Placement windows) – служит только для отображения существующего расположения компонентов на кристалле.

В окне редактируемого размещения (Editable FloorPlan) формируется новая топологическая структура размещения компонентов. Из окна иерархии на окно структуры выносятся компоненты.

При установке компонента, ЛЭ могут быть ориентированы в горизонтальном или вертикальном направлении. Также может быть включен режим поэлементной установки. Выбор режима осуществляется с помощью одной из следующих кнопок на панели управления:  .

Неустановленные элементы помечаются в иерархии в виде отдельных ЛЭ или группы примитивов:



Пиктограммы некоторых ЛЭ приведены далее (цвет роли не играет):

 - триггер (Flip-Flop);

 - контакт ввода-вывода (I/O PAD);

 - функциональный генератор (LUT).

В некоторых случаях бывает удобно объединять набор компонентов с сохранением их взаиморасположения в один блок. Такой блок называется RPM (Relationally Placed Macros) – макрос взаимного расположения. Группа компонентов выделяется и затем объединяется командой Floor-Planer->Меню->Pattern->Bind. Обратная операция – разъединение: Floor-Planer->Меню->Pattern
->UnBind.

Некоторые компоненты, созданные, например программой Core Generator, имеют заранее описанный RP макрос. Такие компоненты нельзя разъединить.

4.1.4. Рекомендации по размещению

- Расстояние между различными соединяющимися блоками нужно делать по возможности минимальным, для уменьшения задержек на соединениях.
- Задержка распространения сигнала, подающегося на несколько блоков, будет определяться самой длинной линией. Взаиморасположение блоков в этом случае должно сокращать самое длинное соединение.
- Число линий связи ограничено. При сложных соединениях используются обходные соединения. Поэтому следует, по возможности, избегать большого количества пересечений линий соединения.
- Синхросигнал располагают рядом со специальным буфером.

4.2. Порядок выполнения работы

1. Создайте определенное преподавателем устройство.
2. Выполните синтез (Synthesis) и реализацию (Implementation) для кристалла V50CS144, семейства Virtex.
3. Обратите внимание на максимальную частоту работы устройства.
4. Запустите FloorPlaner: Project Manager->Меню->Tools->FloorPlaner. Вынесите на чистый шаблон все компоненты из Design Hierarchy, назначив порты устройства определенным преподавателем ножкам. Проверьте наличие ошибок размещения:
   FloorPlaner->Меню->FloorPlan->Check FloorPlan
5. Сохраните план размещения.
6. Установите новый план размещения для реализации
   Project Manager->Меню->Implementation->Set FloorPlan file(s).
7. Проведите повторную реализацию (согласитесь на запрос повторного выполнения этапов Map – Place&Route – Timing). Обратите внимание на новое значение максимальной частоты.

4.3. Варианты заданий

Разместить в одной ПЛИС два устройства, одновременно реализованных по заданию соответствующего варианта лабораторных работ № 2 и № 3. Каждое из устройств должно сгруппироваться в правильной прямоугольной области кристалла. Описание размещения должно содержать как примеры группировки подмножества компонент, так и отдельных примитивов. Синхронные сигналы управления должны быть реализованы в триггерах портов ввода/вывода. Создать ограничения на время распространения отдельных сигналов компонент.

4.4. Порядок выполнения работы

Создать VHD описание устройства, объединяющего два устройства реализованных по заданию соответствующего варианта лабораторных работ № 2 и № 3.

4.5. Содержание отчета

1. Задание на лабораторную работу.
2. Обобщенная структурная схема устройства.
3. VHDL-описания устройств верхнего уровня объединения компонент.
4. Изображение размещения компонент в кристалле ПЛИС.
5. Выводы по работе.

4.6. Контрольные вопросы

1. Как размещение элементов влияет на максимальную частоту устройства?
2. С какими логическими элементами вы сталкивались при работе с FloorPlaner? Каково их назначение?
3. Можно ли изменить логику функционирования устройства, используя FloorPlaner?

Рекомендательный библиографический список

1. Foundation series 3.1i Quick Start Guide
2. Xilinx The programmable logic data book. 1999
3. Xilinx Foundation Series 3.1i Help.
4. Vivtex-II Platform FPGA Handbook, 2000 xilix, Inc.

ОГЛАВЛЕНИЕ

ИНТЕГРИРОВАННЫЕ САПР

Методические указания к лабораторным работам

Составители

ЛАНЦОВ Владимир Николаевич
ГАЛИЧЕВ Евгений Витальевич
ТРОФИМОВ Михаил Александрович
Ответственный за выпуск-зав. кафедрой, профессор В.Н. Ланцов

Редактор Л.В.Пукова
Компьютерная верстка С.В. Павлухиной

ЛР № 020275. Подписано в печать
Формат 60ч84/16. Бумага для множит. техники. Гарнитура Таймс.
Печать на ризографе. Усл. печ.1,86. Уч.-изд. л.1,96.Тираж 100 экз.
Заказ
Редакционно-издательский комплекс
Владимирского государственного университета.
600000, Владимир, ул. Горького, 87.