ИННОВАЦИОННАЯ ОБРАЗОВАТЕЛЬНАЯ ПРОГРАММА

Проект 2: индивидуальная траектория обучения и качество образования
Цель: ориентированное на требования рынка образовательных услуг улучшение качества подготовки и переподготовки специалистов

Федеральное агентство по образованию

Государственное образовательное учреждение
высшего профессионального образования

Владимирский государственный университет

Кафедра вычислительной техники

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ ПО ДИСЦИПЛИНЕ «СИНТЕЗ ЦИФРОВЫХ УСТРОЙСТВ»

Составитель
Л.А. КАЛЫГИНА

Владимир 2008

УДК 621.39 (07)
ББК 32.88-01я2
М54

Рецензент
Кандидат технических наук, доцент
Владимирского государственного университета
С.Ю. Кириллова

Печатается по решению редакционного совета
Владимирского государственного университета

М54 **Методические** указания к лабораторным работам по дисциплине «Синтез цифровых устройств» / Владим. гос. ун-т ; сост. Л. А. Калыгина. – Владимир : Изд-во Владим. гос. ун-та, 2008. – 40 с.

Включают вопросы изучения средств синтеза цифровых устройств с использованием языка описания аппаратуры VHDL и САПР ПЛИС Xilinx ISE. Содержат четыре лабораторные работы. Первые три посвящены изучению принципов проектирования комбинационных и последовательностных цифровых схем, последняя работа содержит индивидуальные задания и носит творческий характер.

Предназначены для студентов специальности 210402 – Средства связи с подвижными объектами и 230101 – Вычислительные машины, комплексы, системы и сети при изучении курса «Синтез цифровых устройств», могут быть полезны студентам, занимающимся вопросами проектирования цифровых устройств в базисе ПЛИС.

Табл. 1. Ил. 17. Библиогр.: 3 назв.

УДК 621.39 (07)
ББК 32.88-01я2

ПРЕДИСЛОВИЕ

Использование языков описания аппаратуры при проектировании цифровых устройств в настоящее время является одним из основных способов проектирования устройств в базисе программируемых логических схем (ПЛИС) и заказных микросхем. Маршрут проектирования с использованием языка VHDL имеется во всех САПР. Поэтому курс «Синтез цифровых устройств на VHDL» включен в рабочие программы для специальностей 210402 – Средства связи с подвижными объектами и 230101 – Вычислительные машины, комплексы, системы и сети.

Методические указания содержат четыре лабораторные работы. Все лабораторные работы выполняются в среде САПР ISE фирмы Xilinx, которая является одним из лидеров в САПР ПЛИС.

При выполнении первой работы студенты должны изучить полный маршрут проектирования цифрового устройства в САПР ПЛИС, включая анализ результатов синтеза проекта в различных типах ПЛИС. Вторая работа посвящена изучению существующих подходов к описанию схемы на VHDL, конструкций и основных операторов языка. Третья работа – изучение способов проектирования последовательностных цифровых схем и особенностей языка VHDL для описания конечных автоматов. Четвертая работа носит творческий характер – студент самостоятельно выбирает способ и средства проектирования заданного устройства. Первая, вторая и третья лабораторные работы могут выполняться бригадой студентов из двух-трех человек, в четвертой работе каждому студенту выдается индивидуальное задание.

ЛАБОРАТОРНАЯ РАБОТА № 1

ИССЛЕДОВАНИЕ МАРШРУТА ПРОЕКТИРОВАНИЯ ЦИФРОВЫХ УСТРОЙСТВ В САПР XILINX ISE

Цель работы: изучение маршрута проектирования электронных устройств в базисе ПЛИС; знакомство со средствами проектирования САПР ПЛИС Xilinx ISE; временное моделирование полученного устройства. Получение навыков проектирования простых цифровых схем.

1.1. Общий маршрут проектирования устройств на ПЛИС в САПР Xilinx ISE

Процесс проектирования устройств на ПЛИС состоит из следующих шагов:

1. Запуск системы и создание нового (открытие существующего) проекта:

а) Запустить Xilinx ISE > Project Navigator.

Главное окно ISE называется ”Project Navigator”. В этом окне находятся инструментальные панели и консоль сообщений.

Панели, расположенные слева, представляют собой соответственно список файлов текущего проекта – окно «Sources in Project» и список процессов (этапов обработки) проекта – «Processes for Current Source».

б) Для создания нового проекта выбирают соответствующий пункт главного меню (File > New Project). В окне открывающегося мастера (рис. 1) задают имя проекта (Project Name) и размещают проект (Project Location) папки с материалами. Необходимо определить параметры проекта (Project Device Options):

- **Выбор типа ПЛИС –** пункт Device Family. Для FPGA выбрать соответствующие типы (Virtex/ Spartan/ XC4000/ XC5200/ XC3000). Для ПЛИС типа CPLD – (CPLD XC9500/ XC9500/ XC9500XL/ CoolRunner/ CoolRunner-II).

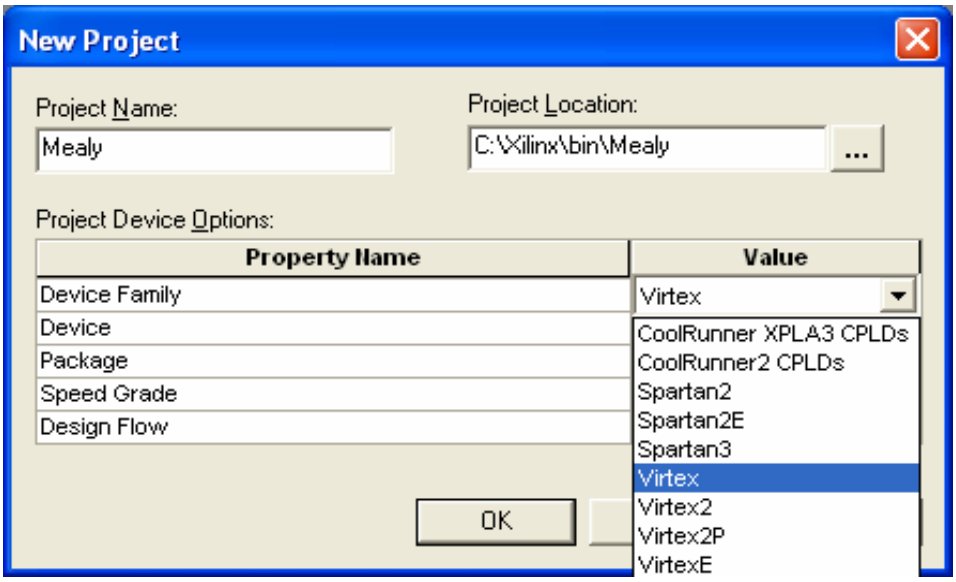


Рис. 1. Диалоговое окно создания нового проекта

- Задать тип микросхемы (Device), тип корпуса (Package) (в лабораторной работе можно оставить значения по умолчанию).
- Задать маршрут проектирования (Design Flow) (рис. 2). Типы XST VHDL и XST Verilog соответствуют описанию на одном из языков описания аппаратуры или графическому вводу, типы EDIF и NGC/NGO предусматривают использование результатов работы внешних средств синтеза. При выполнении лабораторной работы использовать нужно тип XST VHDL.

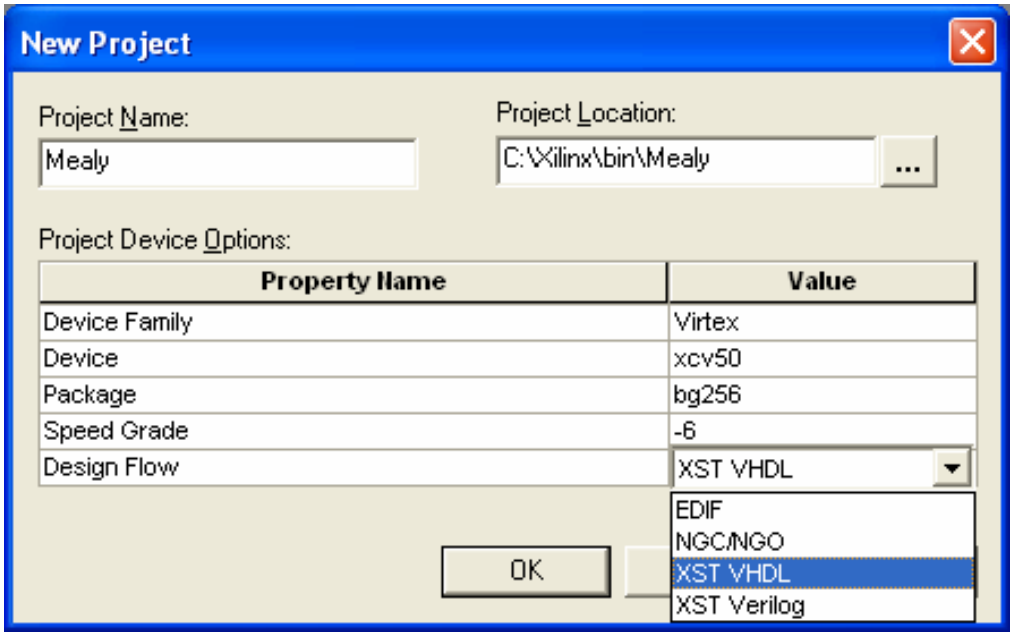


Рис. 2. Диалоговое окно выбора ПЛИС и маршрута проектирования

2. Ввод описания проекта.

а) Для ввода проекта можно использовать графический ввод или описание на языке описания аппаратуры. Для этого в окне «Project Navigator» выполнить Project> New Source, появится окно New, представленное на рис. 3. Для графического ввода выбрать тип Schematic, для VHDL-описания – VHDL Module.

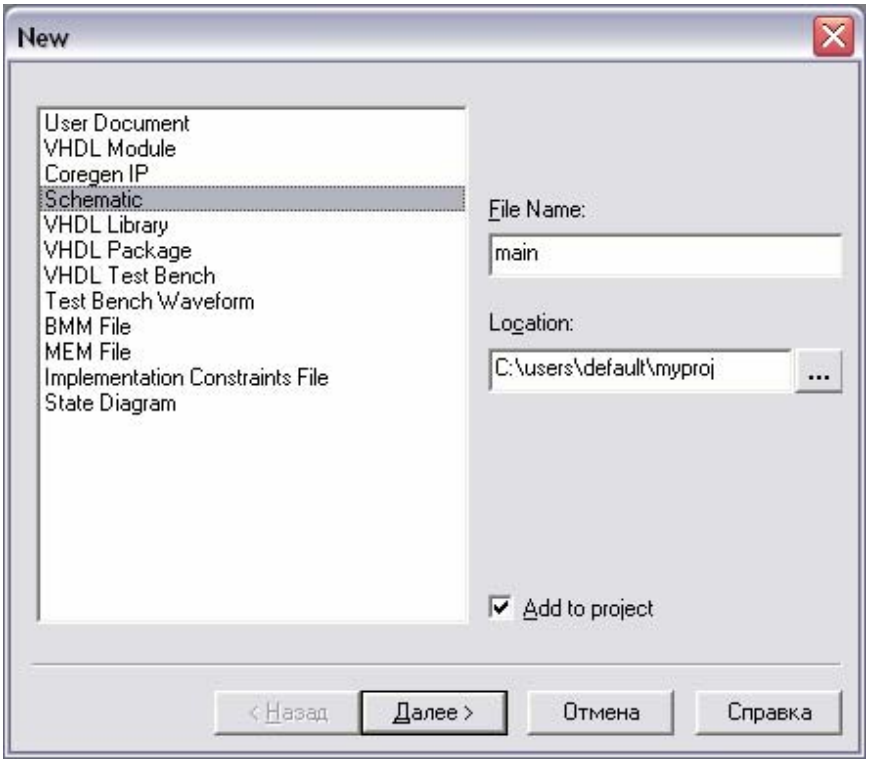


Рис. 3. Выбор схемного описания проекта

Рис. 4. Окно конструктора создания входов и выходов проекта

б) Графический ввод схемы. Определить интерфейс (входы/выходы) объекта. Для графического ввода можно воспользоваться конструктором создания входных и выходных элементов схемы Tools -> Create IOMarkers (рис. 4), указав через запятую имена входных (Inputs) и выходных (Outputs) портов. При использовании конструктора требуется, чтобы на схеме элементов не было.

Схему можно сформировать вручную, перенося элементы с панели компонентов. Подсоединять порты (Add I/O Marker на главной панели) можно только к соединительным линиям, т.е. первоначально нужно «нарисовать» линию (Add Wire на главной панели).

Используя панель компонентов, собрать схему. Готовую схему сохранить.

в) Ввод описания на языке VHDL.

Определение интерфейса представлено на рис. 5. По умолчанию сигналам портов присваивается тип STD_Logic_vector, порядок данных задается позициями MSB, LSB. При необходимости типы сигналов можно изменить в текстовом редакторе.

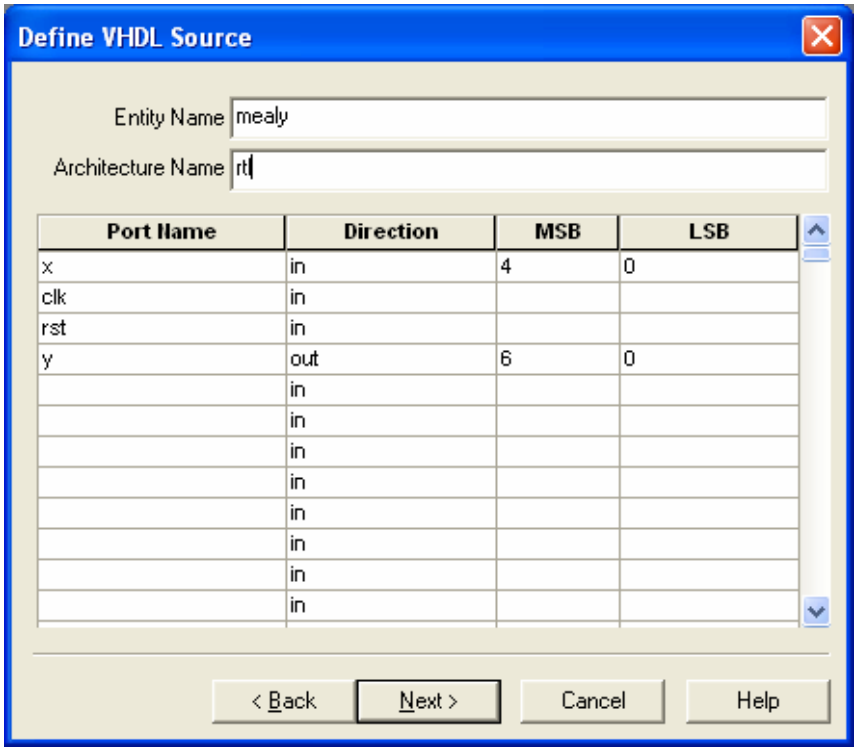


Рис. 5. Диалоговое окно настройки выводов компонента

После завершения работы мастера будет создан шаблон VHDL-файла, который можно редактировать в окне текстового редактора «Project Navigator». По завершении редактирования файл необходимо сохранить.

3. Трансляция компонента во внутреннее схемное представление.

Выбрать компонент в списке файлов проекта, далее в окне доступных процессов появится список операций (процессов), которые можно выполнять для выбранного файла.

Создание схемного представления компонента производится через Design Entry Utilities > Create Schematic Symbol. Двойной щелчок на этом пункте запускает синтез схемотехнического представления компонента, при этом ход синтеза и результаты будут показаны в консоли системных сообщений в нижнем окне. Необходимо анализировать сообщения о результате трансляции, поскольку данная операция производит проверку корректности описания и синтезирует компонент по обычным правилам компиляции. При ошибках в описании, в том числе и методологических, будет выведено сообщение об ошибке и компонент создан не будет.

На рис. 6 показано окно «Project Navigator» после удачного завершения трансляции. В дерево процессов добавляются основные операции, которые соответствуют этапам проектирования устройства – трансляция, синтез, реализация (размещение и трассировка), генерация файла для программирования ПЛИС.

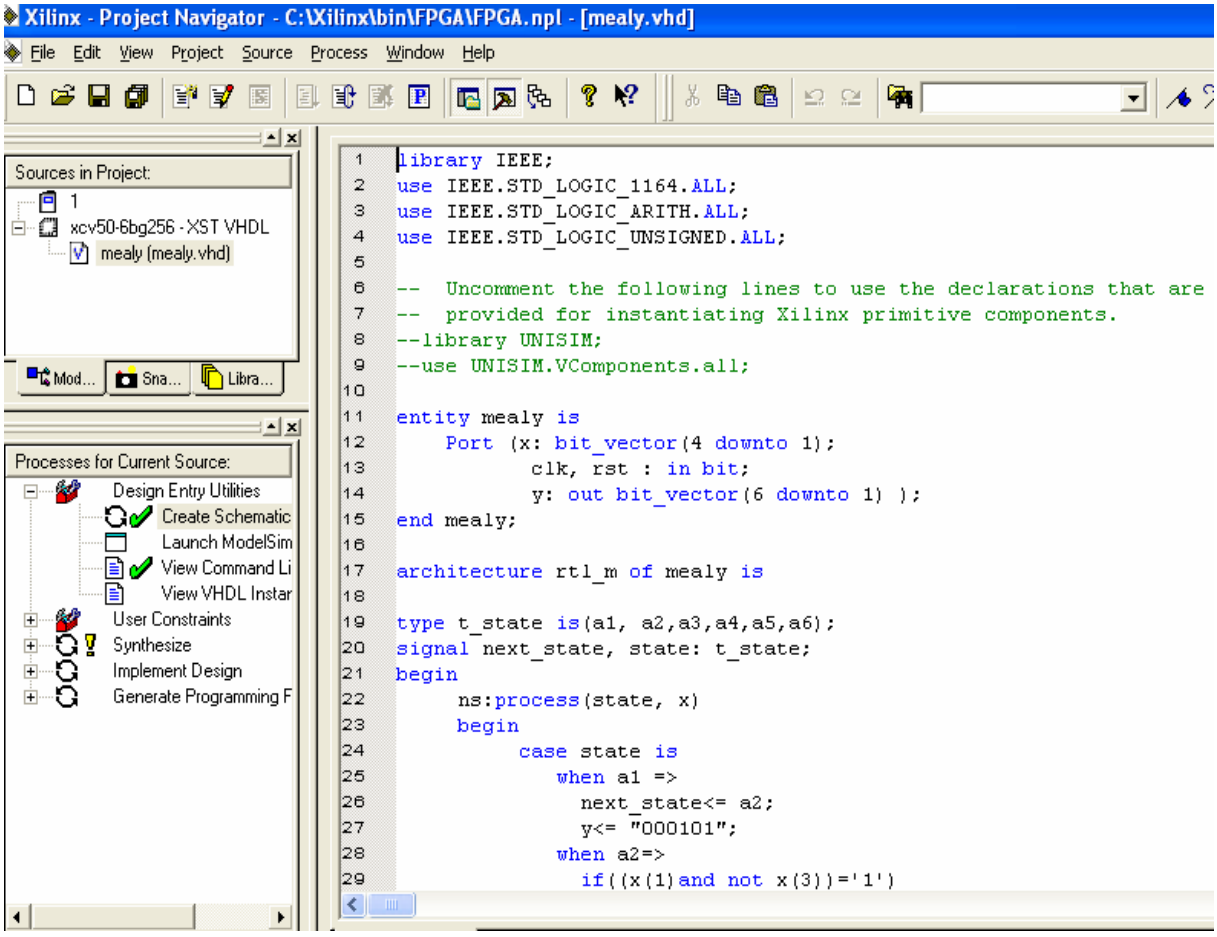


Рис. 6. Окно после создания нового компонента

4. Функциональное моделирование.

а) Для моделирования используется компонента Model Technology Incorporated (ModelSim). Необходимо убедиться, что она подключена к Xilinx ISE. Для определения пути к ModelSim выберите Preferences в меню Edit, и выберите Partners Tool (рис. 7). Путь должен указывать на каталог, который содержит приложение. Изменения вступят в силу после выхода и перезапуска Project Navigator.

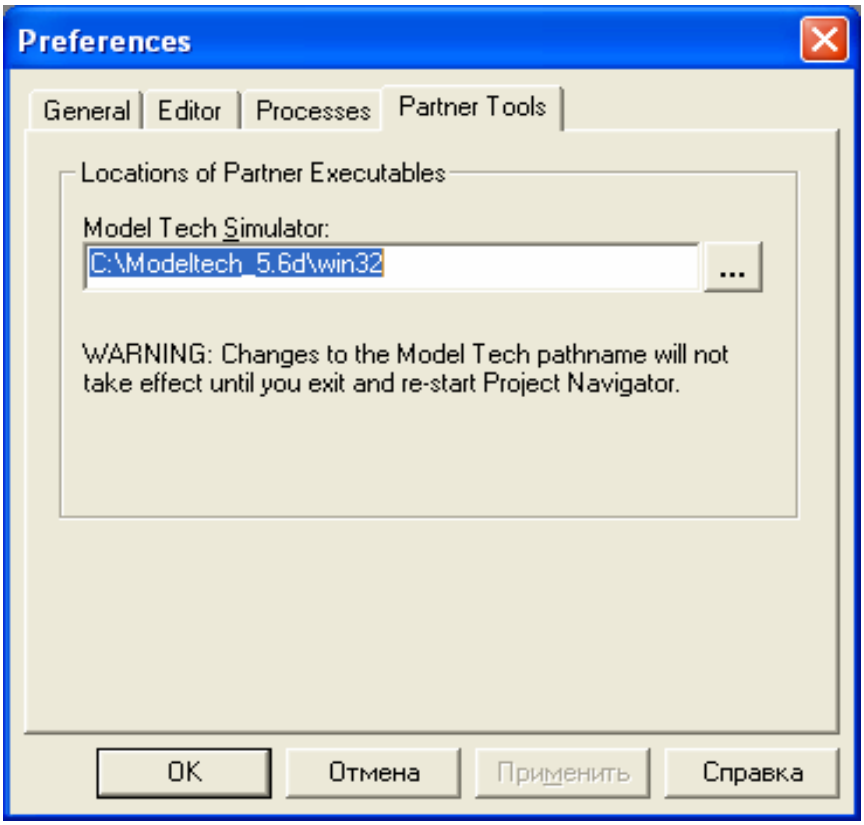


Рис. 7. Диалоговое окно для установления пути (ModelSim)

б) Формирование входных тестовых последовательностей. Добавить в проект новый источник New Source (тип – Test Bench Waveform). Выбрать файл, созданный с помощью схемного редактора или VHDL-файла. Задать тестовые последовательности для входных портов (рис. 8). Сохранить файл.

в) Запуск на моделирование выполняется в окне «Processes for Current Source» Design Entry Utilities > Launch MolelSim Simulator двойным щелчком (рис. 9).

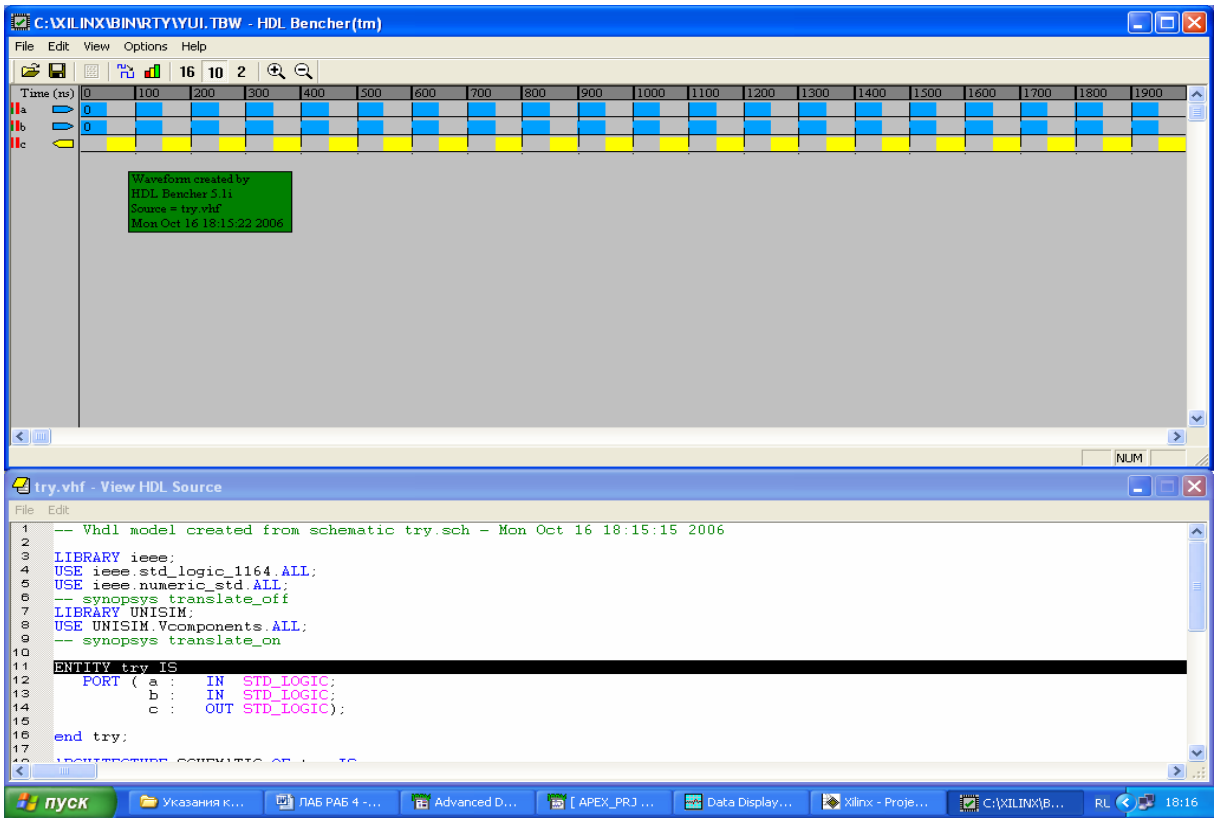


Рис. 8. Окно формирования входных последовательностей

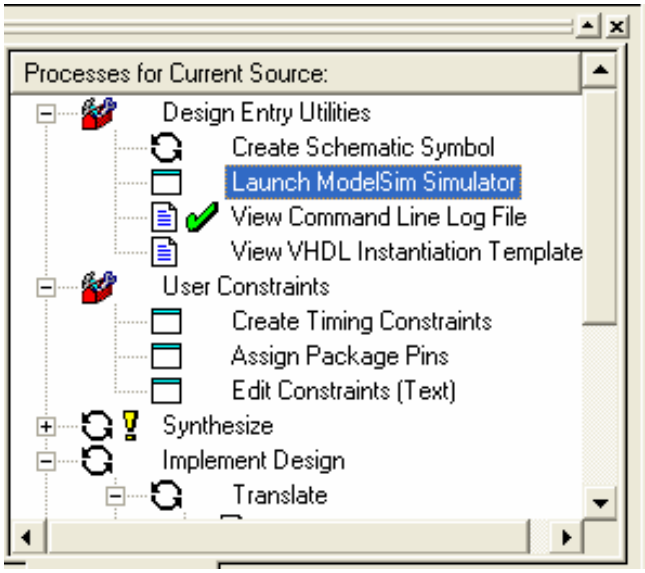


Рис. 9. Моделирование схемотехнического представления компонента

г) Возможны следующие варианты завершения процессов:

– текущий процесс выполнен успешно;

– процедура исполнена без ошибок, но имеются предупреждения;

– при выполнении процесса обнаружены ошибки;

– результаты выполнения процесса устарели (не соответствуют исходным данным).

Ошибки при выполнении трансляции могут возникать в следующих типичных ситуациях:

- в списке файлов не выбран нужный объект. Синтез производится не для объекта, исходный текст которого показан в текстовом

редакторе главного окна, а для объекта, выбранного в списке файлов текущего проекта;

- после редактирования компонента возникла ошибка и активной осталась старая версия. После повторного синтеза компонента убедитесь в том, что компонент оттранслирован без ошибок;
- синтез и реализация, просмотр отчетов и результатов размещения (для FPGA и CPLD) для старой версии проекта.

При отсутствии ошибок на этапе трансляции для простого (не иерархического) проекта автоматически выполняются этапы синтеза (Synthesize) и реализации (Implement Design). Для иерархического проекта обязательно выполнить трансляцию всех составных частей.

Необходимо проанализировать результаты выполнения этапа синтеза и реализации, которые содержатся в соответствующем отчете. Для просмотра отчета удобнее всего дважды щелкнуть левой кнопкой мыши на строке с его названием в окне процессов (рис. 10).

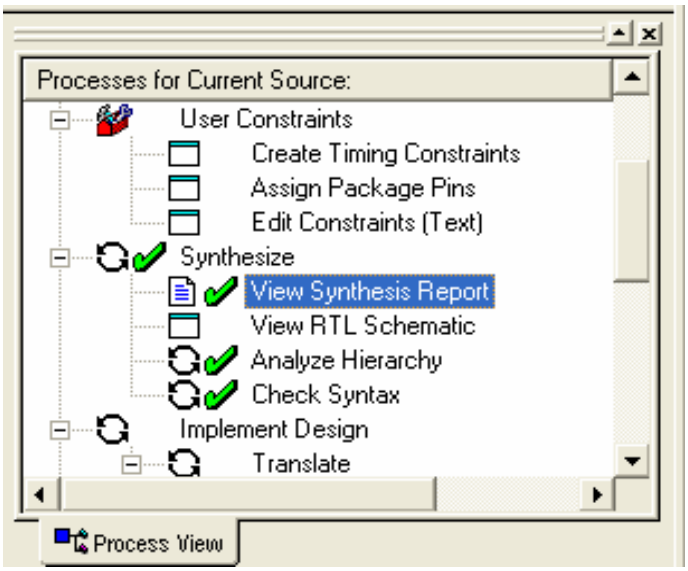


Рис. 10. Выбор отчета результатов синтеза

1.2. Задание к лабораторной работе

1. Изучить маршрут проектирования цифровых устройств в САПР Xilinx ISE.

2. Синтезировать схему для заданной логической функции, минимизировать схему:

- F_1 = Сумма $_{A,B,C,D}$ { 1, 2, 3, 5, 7, 11, 13};
- F_2 = Сумма $_{A,B,C,D}$ { 1, 4, 5, 6, 7, 9, 14, 15};
- F_3 = Сумма $_{A,B,C,D}$ { 0, 2, 5, 7, 8, 10, 13, 15};
- F_4 = Сумма $_{A,B,C,D}$ { 1, 4, 5, 7, 12, 14, 15};
- F_5 = Сумма $_{A,B,C,D}$ {0, 3, 6, 9, 12, 15};
- F_6 = Сумма $_{A,B,C,D}$ {0, 2, 5, 6, 8, 10,11, 14};

• F_7 = Сумма $_{A,B,C,D}$ {2, 3, 4, 6, 8, 12, 14};

• F_8 = Сумма $_{A,B,C,D}$ {1, 3, 6, 8, 9, 11, 14}.

3. Выполнить проектирование полученного устройства, используя ввод в виде принципиальной схемы.

4. Изучить структуру сгенерированного VHDL-описания схемы.

5. Изучить генерируемые системой отчеты.

6. Проанализировать затраченные ресурсы кристалла.

1.3. Содержание отчета

1. Краткое описание средств ввода проекта, реализации и верификации.

2. Описание устройства и интерфейса подключения (VHDL-файл, сгенерированный системой).

3. Результаты реализации устройств в ПЛИС.

4. Временные диаграммы работы устройства.

1.4. Контрольные вопросы

1. Синтез цифровых схем, заданных логической функцией: описание таблицей истинности, карты Карно, минимизация логических схем.

2. Основные этапы маршрута проектирования электронных устройств в базисе ПЛИС.

3. Содержание отчетов САПР, информация:

– о максимальной частоте работы устройства;

– максимальной задержке на комбинационной логике;

– максимальной задержке на трассах между логическими элементами;

– размещении портов I/O по физическим выводам кристалла;

– об объеме устройства в примитивах данной архитектуры (Slice, Flip Flop, LUT и т.д.).

ЛАБОРАТОРНАЯ РАБОТА № 2

СТРУКТУРНОЕ, ПОВЕДЕНЧЕСКОЕ И ПОТОКОВОЕ ОПИСАНИЕ КОМБИНАЦИОННЫХ ЛОГИЧЕСКИХ СХЕМ НА VHDL

Цель работы: изучение основных конструкций языка VHDL, типов описания схем на VHDL и методов проектирования комбинационных схем.

2.1. Краткие теоретические сведения

Комбинационной является такая логическая схема, выходные сигналы которой зависят только от текущих значений входных сигналов.

Основные сведения о языке VHDL содержатся в [1, 2].

При проектировании логических схем на VHDL используются следующие подходы: структурное описание схемы, потоковое и поведенческое описание.

Структурное описание содержит описание элементов схемы и связей между ними. Основными операторами VHDL, используемыми для этого, являются операторы *component, for-generate, entity.* Структурное проектирование является обязательным при проектировании на верхнем уровне иерархии проекта. Пример структурного описания устройства для обнаружения простых чисел приведен ниже.

```
-- Пример 1
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
-- объявление объекта
entity prime is
port(
N: in std_logic_vector(3 downto 0);
```

```
F: out std_logic
);
end prime;

architecture prime_arch of prime is
-- объявление сигналов, внутренних точек схемы
signal N3_L, N2_L, N1_L: std_logic;
signal N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0: std_logic;
-- объявление компонентов схемы, элементы должны быть в библиотеках
component inv is port(I: in std_logic; O: out std_logic);end component;
component and2 is port(I0,I1: in std_logic;O: out std_logic);end component;
component and3 is port(I0,I1,I2: in std_logic;O: out std_logic);end component;
component or41 is port(I0,I1,I2,I3: in std_logic; O: out std_logic);end component;

begin
U1: inv port map (N(3),N3_L); -- операторы component
U2: inv port map (N(2),N2_L);
U3: inv port map (N(1),N1_L);
U4: and2 port map (N3_L,N(0),N3L_N0);
U5: and3 port map (N3_L,N2_L,N(1),N3L_N2L_N1);
U6: and3 port map (N2_L,N(1),N(0),N2L_N1_N0);
U7: and3 port map (N(2),N1_L,N(0), N2_N1L_N0);
U8: or41 port map (N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0, F);
end prime_arch;
```

При *потоковом описании* функционирование схемы описывается в терминах потоков данных и выполняемых операций над этими данными. Основным оператором потокового проектирования является параллельный сигнальный оператор присваивания. Для операций над данными используют встроенные операторы *and*, *or* и *not*. При потоковом описании можно использовать оператор *select*.

Пример потокового описания устройства обнаружения простых чисел с использованием параллельного сигнального оператора:

```
-- Пример 2
architecture prime2_arch of prime is
signal N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0:
std_logic;
begin
N3L_N0 <= not N(3) and N(0);
N3L_N2L_N1 <= not N(3) and not N(2) and N(1);
N2L_N1_N0 <= not N(2) and N(1) and N(0);
N2_N1L_N0 <= N(2) and not N(1) and N(0);
F <= N3L_N0 or N3L_N2L_N1 or N2L_N1_N0 or N2_N1L_N0;
end prime2_arch;
```

Пример потокового описания логической схемы с использованием оператора условного присваивания:

```
-- Пример 3
architecture prime3_arch of prime is
signal N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0:
std_logic;
begin -- использование условного присваивания

N3L_N0 <= '1' when N(3)='0' and N(0)='1' else '0';
N3L_N2L_N1 <= '1' when N(3)='0' and N(2)='0' and N(1)='1' else '0';
N2L_N1_N0 <= '1' when N(2)='0' and N(1)='1' and N(0)='1' else '0';
N2_N1L_N0 <= '1' when N(2)='1' and N(1)='0' and N(0)='1' else '0';
F <= N3L_N0 or N3L_N2L_N1 or N2L_N1_N0 or N2_N1L_N0;
end prime3_arch;
```

Пример потокового описания логической схемы с использованием оператора избирательного присваивания:

```
-- Пример 4
architecture prime4_arch of prime is
begin --использование избирательного присваивания
with N select
```

```
F <= '1' when "0001",
'1' when "0010",
'1' when "0011"| "0101"| "0111",
'1' when "1011"| "1101",
'0' when others;
end prime4_arch;
```

Объявление объекта в примерах 2 – 4 такое же, как в примере 1.

Поведенческое проектирование. Стиль описания устройства, опирающийся на описание поведения выполняемых устройством функций без привязки к структуре устройства, называется поведенческим. Строго говоря, поведенческое описание можно создать, используя операторы потокового проектирования. Условимся называть поведенческим описание, использующее оператор *process* и последовательные операторы – последовательный сигнальный оператор присваивания, операторы *if, case, loop, for*. Примеры поведенческого проектирования устройства обнаружения простых чисел приведены в примерах 5 – 8.

Пример поведенческого описания логической схемы с использованием сигнального оператора присваивания:

```
-- Пример 5
architecture prime5_arch of prime is
begin
process(N)
variable N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0:
std_logic;
begin
N3L_N0 := not N(3) and N(0);
N3L_N2L_N1 := not N(3) and not N(2) and N(1);
N2L_N1_N0 := not N(2) and N(1) and N(0);
N2_N1L_N0 := N(2) and not N(1) and N(0);
F <= N3L_N0 or N3L_N2L_N1 or N2L_N1_N0 or N2_N1L_N0;
End process;
end prime5_arch;
```

Пример поведенческого описания логической схемы с использованием оператора *if:*

```
-- Пример 6
architecture prime6_arch of prime is
begin
process(N)
variable NI: INTEGER;
begin
NI:= CONV_INTEGER(N);
If NI=1 or NI=2 then F<='1';
Elsif NI=1 or NI=5 or NI=7 or NI=11 or NI=13 then F<='1';
Else F <= '0';
End if;
End process;
end prime6_arch;
```

Пример поведенческого описания логической схемы с использованием оператора *case:*

```
--Пример 7
architecture prime7_arch of prime is
begin
process(N)
begin
case CONV_INTEGER(N) is
when 1 => F<='1';
when 2 => F<='1';
when 3|5|7|11|13 => F<='1';
when others => F<='0';
end case;
end process;
end prime7_arch;
```

Пример поведенческого описания логической схемы с использованием оператора *for:*

```
--Пример 8
entity prime8 is
```

```
port(
N : in std_logic_vector(15 downto 0);
F : out std_logic
);
end prime;

architecture prime8_arch of prime8 is
begin
process(N)
variable NI,I: INTEGER;
variable prime: boolean;
begin
NI:= CONV_INTEGER(N);
Prime:=true;
If NI=1 or NI=2 then null;
Else
for I in 2 to 253 loop
if (NI mod I)=0 then
prime :=false;
end if;
end loop;
End if;
If prime then F<='1'; else F<='0'; end if;
End process;
end prime8_arch;
```

2.2. Задание к лабораторной работе

1. Изучить основные конструкции программ и операторы языка VHDL:

- структура программы, объявление объекта, объявление архитектуры;
- структурное описание – операторы – *Component, for-generate, entity;*
- потоковое описание – параллельные сигнальные операторы присваивания, оператор *select;*

• поведенческое описание – операторы *process, if, case, loop, for.*

2. Сформировать VHDL-описание проекта (из лабораторной работы № 1), используя различные конструкции языка (два варианта потокового описания и два варианта поведенческого).

3. Выполнить полный цикл проектирования цифрового устройства, включая этап реализации в ПЛИС типа CPLD для трех вариантов реализации проекта.

4. Провести анализ результатов синтеза цифровой схемы в зависимости от используемых конструкций языка VHDL.

2.3. Содержание отчета

1. Тексты VHDL-описаний проекта (четыре варианта), отчеты по синтезу.

2. Выводы.

2.4. Контрольные вопросы

1. Структура программ на языке VHDL.
2. Структурное описание проекта, применение и синтаксис операторов *Component, for-generate, entity.*
3. Использование и синтаксис параллельных операторов *process, if, case, loop, for,* параллельные сигнальные операторы присваивания, оператор *select.*
4. Потоковое описание проекта.
5. Поведенческое описание проекта.
6. Объяснить полученные результаты.

ЛАБОРАТОРНАЯ РАБОТА № 3

СИНТЕЗ ПОСЛЕДОВАТЕЛЬНОСТНЫХ ЛОГИЧЕСКИХ СХЕМ НА VHDL

Цель работы: изучение возможностей языка VHDL для описания последовательностных схем.

3.1. Краткие теоретические сведения

Последовательностной называется такая логическая схема, выходные сигналы которой определяются не только текущими значениями входных сигналов, но зависят также от последовательности значений входных сигналов в предыдущие моменты времени.

Основным инструментом реализации последовательностной логики являются конечные автоматы.

Для описания конечного автомата на VHDL используются все конструкции языка VHDL и способы описания комбинационных схем, а также дополнительные возможности языка, позволяющие описывать особенности последовательностных схем:

– средства VHDL для описания схем с обратными связями (необходимы при структурном описании конечного автомата);

– описание тактируемых схем и способов моделирования синхропоследовательностей;

– конструкции языка для поведенческого описания конечных автоматов.

Описание схем с обратными связями. Сигнал выходного порта (тип *out*) нельзя использовать в качестве внутреннего сигнала, для описания обратной связи выходного сигнала схемы необходимо использовать тип выходного порта *buffer*. Пример программы для SR-защелки приведен ниже. Чтобы устройство с обратными связями можно было использовать в составе сложных схем, не переопределяя тип порта на *buffer* (это связано с проблемами при реализации устройства в ПЛИС), можно объявить дополнительный сигнал и использовать его для реализации обратной связи, сигнал выходного порта определять через этот дополнительный сигнал:

```
entity Vsrlatch is
        Potr ( S, R: in STD_LOGIC;
               Q, QN: buffer STD_LOGIC);
end Vsrlatch;

architecture Vsrlatch_arch of Vsrlatch is

begin
QN <= S nor Q;
Q <= R nor QN;

End Vsrlatch_arch;
```

```
entity Vsrlatch is
 Potr (S, R: in STD_LOGIC;
 Q, QN: out STD_LOGIC);
 end Vsrlatch;

architecture Vsrlatch_arch of
                    Vsrlatch is
signal Q_d, QN_d: STD_LOGIC;
begin
        QN_d <= S nor Q_d;
        Q_d <= R nor QN_d;
        Q <= Q_d;
        QN <= QN_d;
End Vsrlatch_arch;
```

Описание тактируемых схем. Для описания переключающегося по фронту элемента используется признак *event*, который можно присоединить к имени сигнала для получения переменной типа *boolean*. Полученная переменная принимает значение *true* при изменении значения сигнала и *false* при неизменном значении. Передний фронт тактового сигнала *CLK* можно определить следующим условием:

if CLK'event and CLK='1' then …

Булева переменная *CLK'event* принимает значение *true* в моменты переднего и заднего фронта синхроимпульса, *CLK'event and CLK='1'* принимает значение *true* только в момент переднего фронта синхроимпульса.

При анализе тактируемых схем необходимо моделировать синхропоследовательность, для этого можно использовать признак *after*, который позволяет присваивать значение сигналу не мгновенно, а через заданный промежуток времени (необходимо указывать единицу измерения времени, если это не секунда).

Пример моделирования синхропоследовательности приведен ниже:

```
-- semi_period – половина периода
process
begin
```

```
while (true) loop
clk <= 1, 0 after semi_period;
wait until semi_period;
end loop;
end process;
```

Поведенческое описание конечного автомата

1. При кодировании состояний конечного автомат на VHDL можно использовать следующие конструкции языка:

– перечислимый тип. В этом случае состояния будут кодироваться программой синтеза.

Пример объявления перечислимого типа и использования переменной этого типа:

```
architecture Behavioral of avtomat is
type STATE_TYPE is (INIT, S1, S2…);
signal CS, NS: STATE_TYPE;
. . .
```

– использование признака *enum_encoding* для определения кодов состояний. Этот признак определен в большинстве средств синтеза, для его использования необходимо указать соответствующую библиотеку:

```
. . .
library SYNOPSYS;
use SYNOPSYS.attributes.all;
. . .
architecture Behavioral of avtomat is
type STATE_TYPE is (INIT, S1, S2…);
attribute enum_encoding of STATE_TYPE: type is
“0000 0001 0010 …”;
signal CS, NS: STATE_TYPE;
. . .
```

– использование обычной логики и констант:

```
architecture Behavioral of avtomat is
subtype STATE_TYPE is STD_LOGIC_VECTOR (1 to 4);
constant INIT: STATE_TYPE:= “0000”;
constant S1: STATE_TYPE:= “0001”;
```

```
. . .
signal CS, NS: STATE_TYPE;
. . .
```

2. В теле архитектуры объявляют два сигнала этого типа: один для текущего, второй для следующего состояния автомата (*signal CS, NS: STATE_TYPE;*).

3. Объявляют два процесса: один для установки текущего состояния автомата, второй – для установки выходного сигнала и следующего состояния автомата на основе входного сигнала и текущего состояния.

Для описания конечного автомата применяются таблицы состояний или диаграммы состояний. Перед тем как приступать к описанию цифрового устройства, необходимо проанализировать конечный автомат, оптимизировать условия перехода, закодировать сигналы и состояния.

Пример конечного автомата приведен на рис. 11.

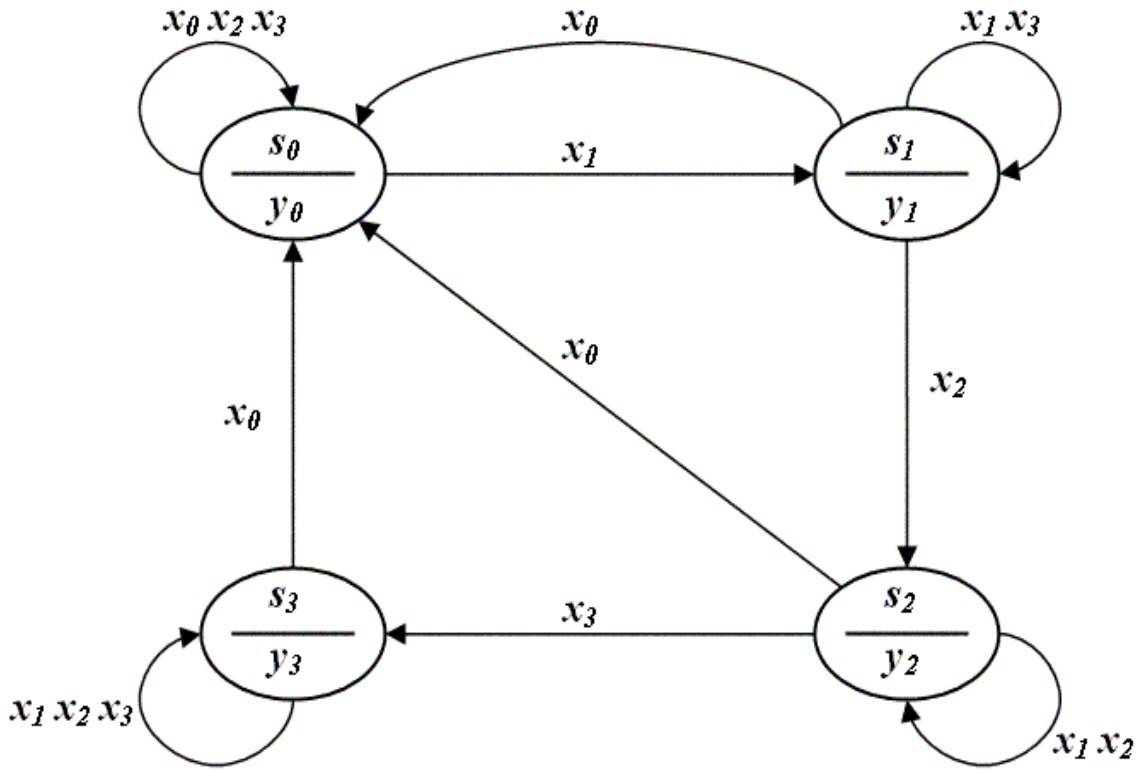


Рис. 11. Схема конечного автомата

Из диаграммы конечного автомата видно, что по сигналу x_0 происходит сброс автомата в начальное состояние. На схеме сигнал сброса *rs* обозначается ●⟶.

В цифровой технике принято выделять для сброса отдельный вход, остальные входные сигналы закодируем так: x_1 = 00, x_2 = 01, x_3 = 10. Выходные сигналы закодируем по порядку: y_0 = 00, y_1 = 01, y_2 = 10, y_3 = 11. Необходимость в кодировании состояния возникает не всегда (они могут совпадать с выходными сигналами). Для данного автомата: s_0 = 00, s_1 = 01, s_2 = 10, s_3 = 11. Входной сигнал обозначим *di*, выходной – *d*o, состояние назовём *q*. В результате получаем схематическое описание автомата, представленное на рис. 12.

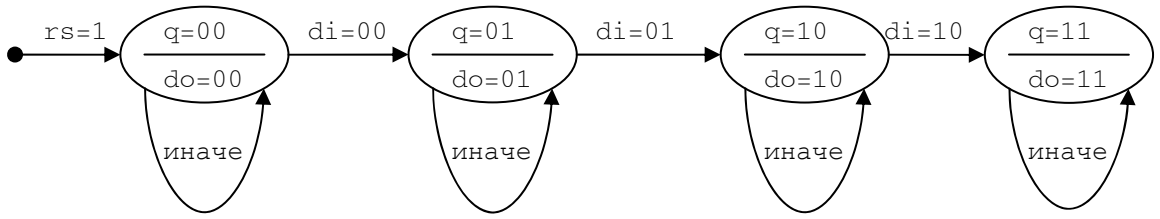


Рис. 12. Модифицированное представление конечного автомата

Синтез конечных автоматов при помощи StateCAD

В САПР Xilinx ISE есть встроенное средство для автоматической генерации VHDL-описания конечных автоматов – *StateCad*. С его помощью можно быстро создавать описание довольно сложных конечных автоматов, оптимизированное для типа ПЛИС (CPLD или FPGA). Ниже приведена последовательность шагов для описания диаграммы состояний в *StateCad*.

1. Создать новый проект (тип маршрута VHDL: XST VHDL).

2. Выполнить команду в меню Project → New Source. В появившемся окне выбрать State Diagram и указать имя (длина имени должна быть не более 8 символов). Это запустит *State CAD*.

3. На панели инструментов *StateCAD* нажать кнопку Draw State Mashines.

4. На первом шаге указать число состояний автомата (в примере – 4) и форму представления shape (в примере – row) (рис. 13). На остальных шагах оставить значения, предлагаемые по умолчанию.

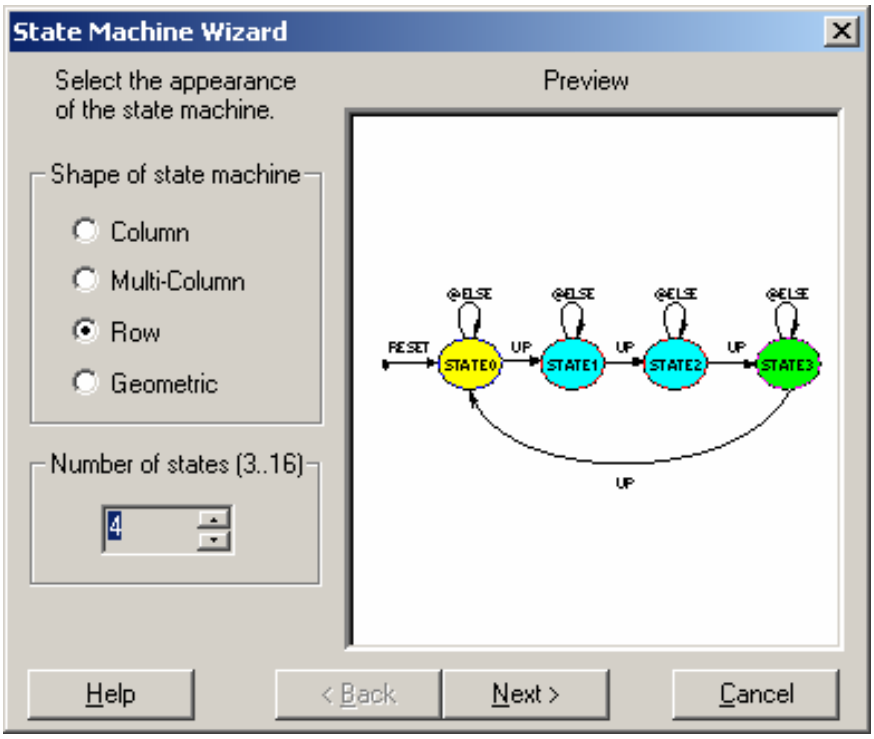


Рис. 13. Окно редактора описания конечных автоматов

5. Добавить вектора (Add vector) для входных и выходных сигналов (сигнал сброса и синхросигнал добавляются автоматически), установить их разрядность. В примере – два вектора *di* и *do*, разрядность 1:0 (рис. 14).

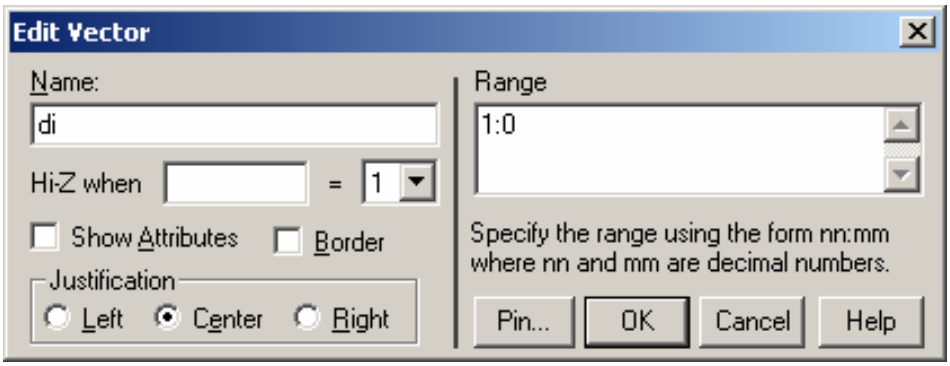


Рис. 14. Окно описания векторов входных и выходных сигналов

6. Установить условие перехода для каждого перехода, для этого необходимо дважды щёлкнуть на стрелке и заполнить поле Condition. Окно для стрелки up State0 приведено на рис. 15.

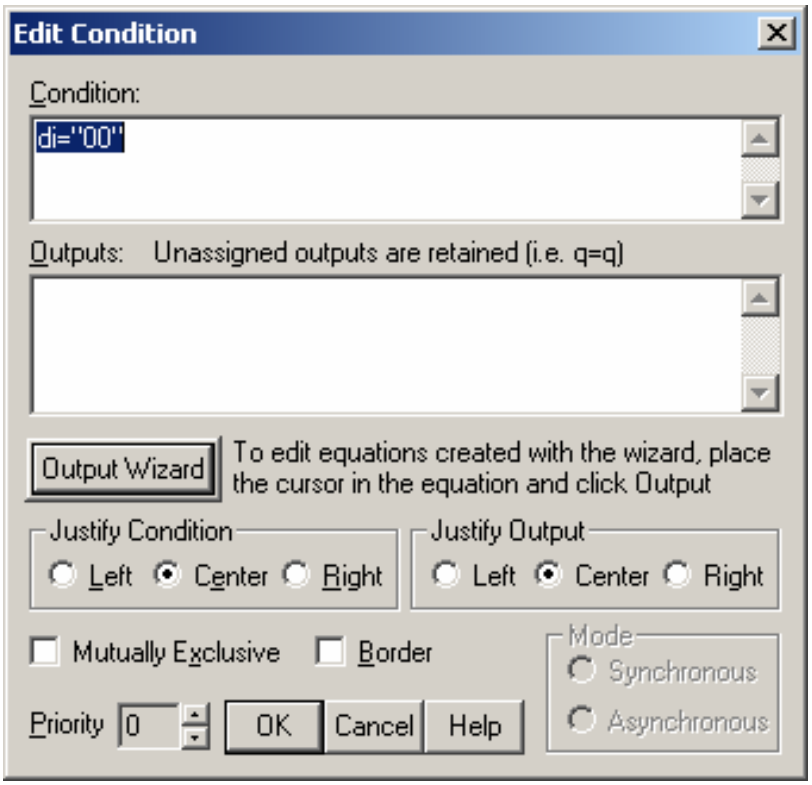


Рис. 15. Окно описания переходов автомата

7. Определить выходной сигнал состояния, для этого дважды кликнуть на нём и заполнить поле Outputs. Окно для состояния State0 приведено на рис. 16.

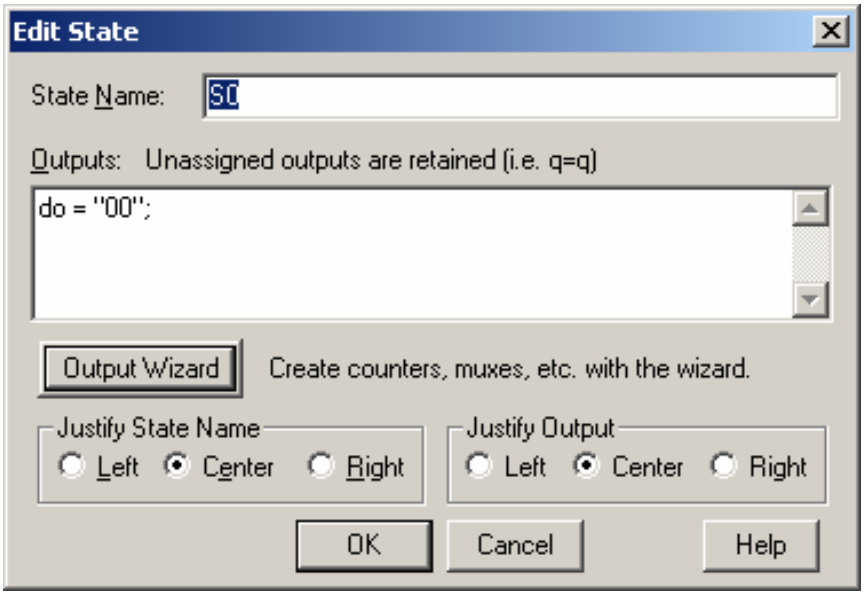


Рис. 16. Окно описания состояния

8. Выполнить п. 6 и 7 для всех переходов и состояний. Вид диаграммы для примера – на рис. 17.

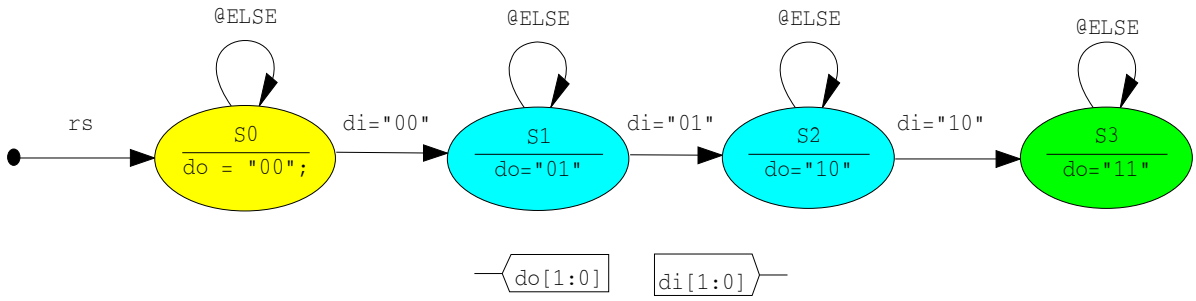


Рис. 17. Диаграмма состояний конечного автомата

9. Выполнить команды Optimize (для настройки параметров) и Generate HDL. Результат – VHDL-описание конечного автомата.

10. Закрыть *StateCAD* и добавить сгенерированный файл в проект.

Примеры поведенческого описания конечных автоматов приведены в [1].

3.2. Задание к лабораторной работе

1. Изучить основные конструкции программ и операторы языка VHDL, используемые при проектировании последовательностных схем.

2. Составить по заданной диаграмме состояний конечного автомата таблицу состояний. Реализовать синхронный (для чётных вариантов) или асинхронный (для нечётных) сброс.

3. Составить два варианта VHDL-описаний конечного автомата, выполнить полный цикл проектирования на ПЛИС.

4. Получить с помощью *StateCAD* VHDL-описание конечного автомата, выполнить полный цикл проектирования на ПЛИС. Проанализировать полученный VHDL-код.

5. Провести анализ результатов синтеза цифровой схемы в зависимости от используемых конструкций языка VHDL.

3.3. Содержание отчета

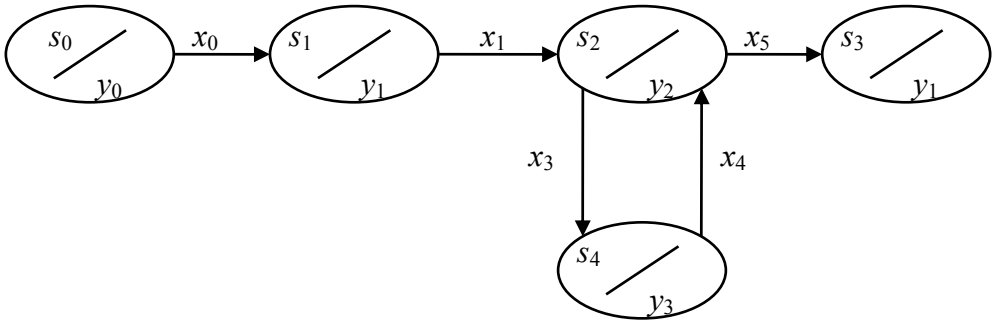
1. Диаграмма состояний и таблица переходов автомата.
2. VHDL-описание (три варианта) конечного автомата, отчеты по синтезу.
3. Выводы.

3.4. Контрольные вопросы

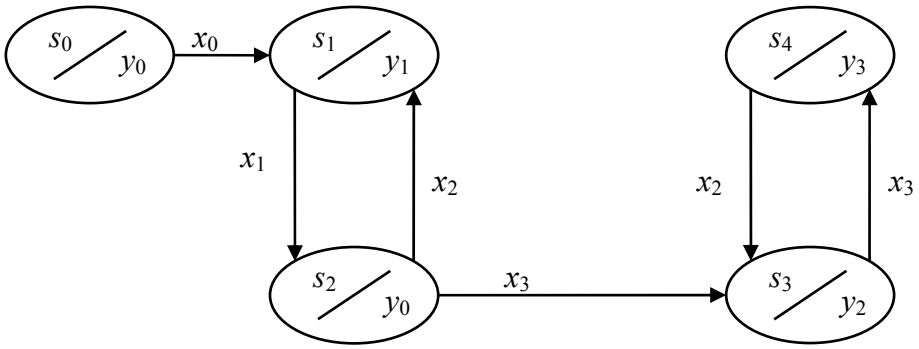
1. Синтез конечных автоматов по таблице переходов.
2. Приемы создания программы, описывающей таблицу переходов конечных автоматов на языке VHDL.
3. Структурное описание конечных автоматов.
4. Синтаксис конструкций VHDL для описания конечных автоматов.
5. Объяснить полученные результаты.

3.5. Варианты конечных автоматов

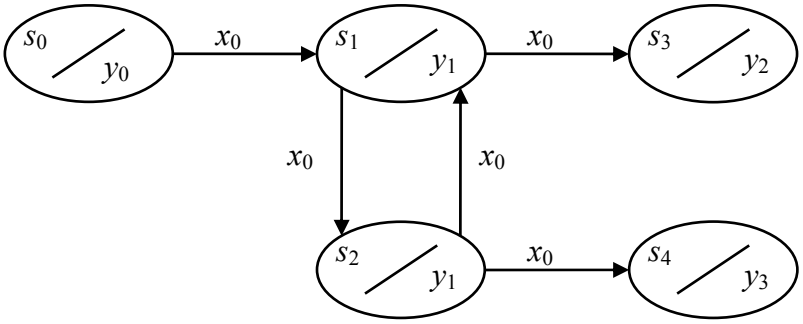
Вариант 1



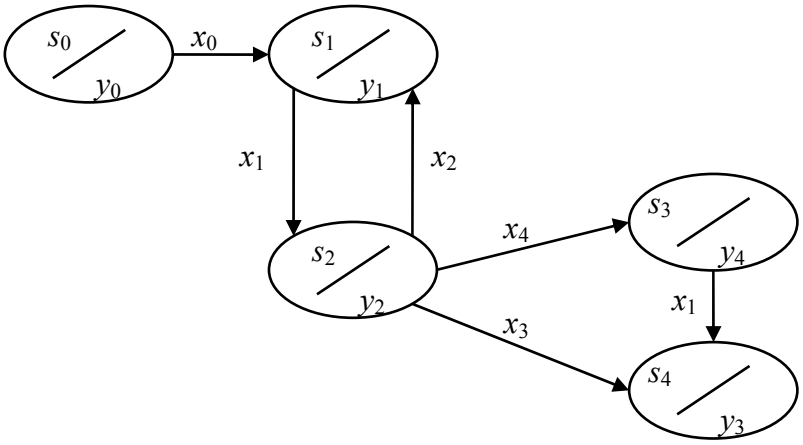
Вариант 2



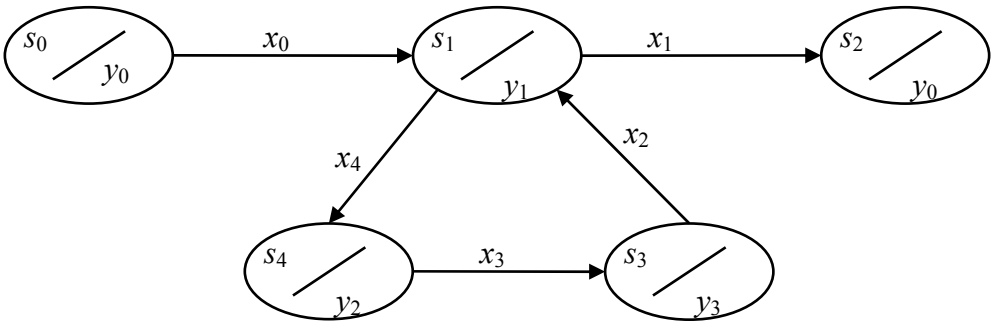
Вариант 3



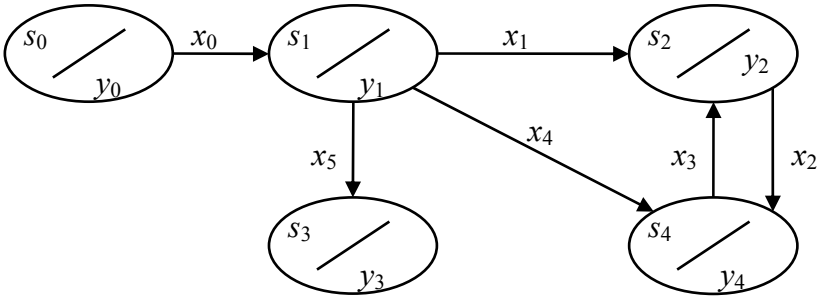
Вариант 4



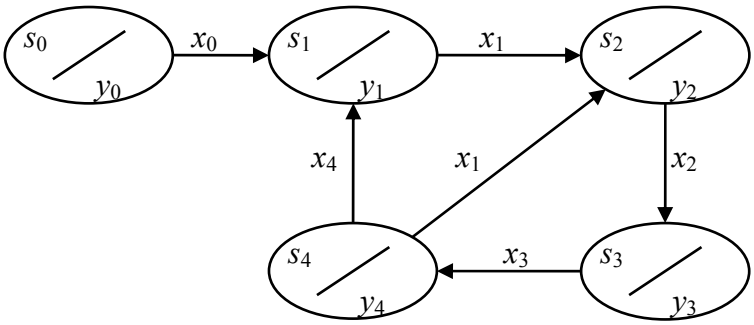
Вариант 5



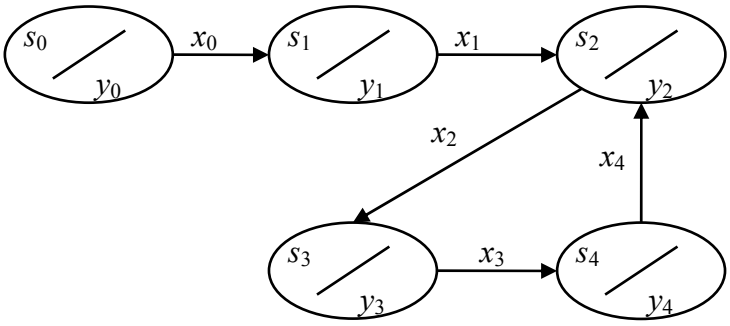
Вариант 6



Вариант 7



Вариант 8



ЛАБОРАТОРНАЯ РАБОТА № 4

РАЗРАБОТКА ЯДРА ЦИФРОВОГО БЛОКА НА БАЗЕ ЯЗЫКА VHDL

Цель работы: изучение правил создания цифровых блоков на VHDL, разработка ядра цифрового блока.

4.1. Краткие теоретические сведения

При проектировании различных классов устройств рекомендуется придерживаться следующих правил.

Общие правила проектирования:

– *спецификация*. Перед началом написания кода надо проверить существующие ядра и написать спецификацию. Спецификация должна содержать интерфейс устройства как чёрного ящика, описание функций с ссылкой на стандарты. Желательно использовать шаблон спецификации;

– *проектная документация.* Если над ядром работает группа разработчиков, проектная документация должна быть написана до начала программирования; это условие того, что разрабатываемые блоки будут способны работать совместно без дополнительных затрат на их интеграцию;

– *структура директорий.* Необходимо использовать строго оговоренную структуру директорий и она обязательно должна присутствовать в проектной документации и документации пользователя.

Правила разработки на системном уровне:

– рекомендуется писать наглядные комментарии, писать комментарии к каждому новому назначению или блоку;

– для иерархического проекта рекомендуется создавать модуль верхнего уровня, включающий модули более низких уровней, не использовать в модуле верхнего уровня логику;

– желательно сохранять одно и то же имя сигнала на различных уровнях иерархии.

Правила разработки на уровне регистровых передач:

– не рекомендуется смешивать логику, работающую по верхнему и нижнему уровню сигнала. Желательно придерживаться логики одного верхнего уровня;

– *сброс (Reset).* Рекомендуется создавать асинхронный сброс, работающий по верхнему уровню сигнала для всех триггеров. Такой сброс должен быть синхронизирован с тактовым генератором на верхнем уровне проекта и не должен требовать мультиплексирования на каждом входе триггера. Рекомендуется, чтобы во время сброса все двунаправленные порты находились в состоянии входа;

– *синхросигналы.* При наличии нескольких синхросигналов сигналы, которые пересекают различные области тактирования, должны быть дискретизированы до и после пересечения таких областей для предотвращения нестабильных состояний. Рекомендуется не использовать синхросигнал или сигнал сброса как данные или как сигнал запуска. Не использовать данные как синхросигнал или сигнал сброса. Желательно использовать минимальное количество областей синхросигналов в одном проекте;

– *шины.* Строго рекомендуется сравнивать шины одинаковой длины, начинать индексировать шину с 0, использовать преобразование MSB в LSB (старший значащий разряд в младший значащий разряд). При этом бит 0 должен являться младшим значащим (LSB). Не делать шины шире, чем это необходимо. Если возможно, делать шину данных зауженной, а взамен увеличивать шину адреса;

– *тристабильные схемы.* Рекомендуется избегать использования внутренних тристабильных сигналов. Они используются только для внутреннего мониторинга;

– *память.* Рекомендуется применять синхронные одно- и двухпортовые групповые блоки памяти;

– *кодирование для синтеза.* Желательно использовать синхронный метод проектирования. Это позволяет избежать проблем с синтезом, временной верификацией и при моделировании. Строго рекомендуется не использовать элементов задержки. Это приводит к проблемам синтеза и временной верификации;

– рекомендуется все внешние входы/выходы ядра *буферизировать;*

– синхронизировать внутренние интерфейсы ядра;

– избегать использования регистров-защёлок (latches);

– желательно избегать использования триггеров с отрицательным перепадом синхроимпульса;

– *порты ввода/вывода*. Рекомендуется называть порты ядра, опираясь на соглашения, показанные в таблице с обязательным занесением в проектную документацию;

Правила именования портов ввода/вывода

Порт	Описание
*_*i*	Входной порт ядра
*_*o*	Выходной порт ядра
*_*io*	Двухсторонний порт ядра
*_*clk_i*	Порт входного синхросигнала ядра
*_*clk_o*	Порт выходного синхросигнала ядра
*_*rst_i*	Входной порт сигнала сброса
*_*rst_o*	Выходной порт сигнала сброса

– не стоит использовать другие аббревиатуры кроме *_clk_* и *_rst_* для обозначения синхросигнала и сигнала сброса. Например, не надо использовать *reset* или *clock* для избежания проблем при синтезе;

– рекомендуется использовать *n для обозначения сигналов, работающих по низкому уровню.

Правила создания проектов на языке описания аппаратуры VHDL. Они учитывают особенности языка VHDL и возможности современных систем автоматизированного синтеза для ПЛИС. Эти правила относятся к стадии разработки моделей блоков на уровне регистровых передач:

– необходимо использовать тип std_logic для внешних портов. Нельзя назначать сигналам неизвестное значение 'x'. В некоторых случаях могут создаваться некорректные результаты при моделировании и синтезе. Использовать значения по умолчанию или инициализацию сигналов и переменных можно, хотя и не желательно, только для моделирования, но не синтеза (`variable B:INTEGER:=0;`). Такое назначение может привести к расхождениям при моделировании и реализации. Необходимо использовать сигнал сброса для задания всех сигналов и переменных;

– нельзя использовать порты с буфером для чтения выходных значений. Взамен этого нужно добавлять другие переменные или сигналы с тем же выходным значением. Это связано с тем, что порты буферного типа не могут быть соединены с портами другого типа и поэтому данный тип buffer распространяется на порты всего проекта;

– рекомендуется использовать определение компонент и констант для каждого ядра в одиночном модуле. Нужно писать один VHDL-элемент проекта в одном файле. Имя файла должно быть такое же, как имя элемента;

– желательно не смешивать в проекте различные стандарты VHDL (например не смешивать конструкции VHDL 87 и VHDL 93);

– желательно компилировать каждый блок в отдельную библиотеку. Рекомендуется пользоваться константами и настраиваемыми параметрами (generics) для размера буферов, ширины шин и всех других параметров компонента;

– *кодирование для синтеза.* Строго рекомендуется читать из переменных, затем писать в них (чтение перед записью). Если переменные сначала записываются, а затем считываются, то это приводит к созданию длинной комбинации логики и триггеров-защёлок (или регистров-защёлок). Это следует из того, что переменные получают своё значение сразу же, а не так как сигналы. Далее приведен пример неправильной конструкции, которой необходимо избегать:

```
PROCESS (CLK, RST_n)
Variable out_var : std_logic;
BEGIN -- PROCESS
IF RST_n = '0' THEN
out_var <= '0';
outsign2 <= '0';
ELSIF CLK'event AND CLK = '1' THEN
Outsign2 <= out_var; ФФ чтение
out_var <= input1 and input2; ФФ запись
END IF;
END PROCESS;
```

– необходимо включать все сигналы, которые считываются внутри комбинированного процесса, в его список чувствительности. Это делают для предотвращения появления нежелательных триггеров-защёлок;

– рекомендуется избегать использования длинных выражений if-then-else, а взамен использовать оператор case. Это предотвращает появление декодеров высокого приоритета и делает код более легко читаемым;

– нельзя использовать такие выражения, как ‘(b <= a after X ns)’ или ‘wait for X ns;’. Данные выражения используются только для моделирования;

– рекомендуется использовать сигнал clock enable (CE), как это показано ниже, с одним синхросигналом на процесс и не использовать два различных процесса – один для синхронизации, а другой для комбинаторной логики. Это связано с тем, что некоторые системы синтеза сами находят сигнал CE и назначают его. В противном случае ножка CE не будет задействована, что приведет к появлению дополнительной логики.

```
PROCESS (CLK, RST_n)
BEGIN -- PROCESS
IF RST_n = '0' THEN
Outsignal <= '0';
ELSIF CLK'event AND CLK = '1' THEN
IF (CE = '1') THEN
Outsignal <= '1';
END IF;
END IF;
END PROCESS;
```

– желательно писать тестовые последовательности в двух частях – одна часть для генерации данных для проверки функционирования, а вторая часть для генерирования временных параметров протоколов шинных интерфейсов и их проверки;

– *заголовок файла.* Рекомендуется использовать стандартный заголовок в начале каждого файла. Заголовок содержит базовую информацию о проекте, файле, авторе, лицензии и др. Все пункты заголовка должны быть оговорены в проектной документации.

4.2. Задание на лабораторную работу

1. Изучить правила создания программных ядер цифровых блоков.

2. Разработать ядро цифрового блока на языке VHDL согласно варианту задания.

3. Синтезировать устройство в ПЛИС типа CPLD и FPGA.

4.3. Содержание отчета

1. Спецификация устройства.
2. Описание тестовых последовательностей.
3. Текст программы на языке VHDL.
4. Отчеты по синтезу для двух типов ПЛИС.
5. Выводы.

4.4. Контрольные вопросы

1. Понятие ядра цифрового блока. Виды ядер.
2. Содержание спецификации ядра цифрового блока.
3. Принципы формирования тестовых последовательностей (для разработанного ядра).
4. Обоснуйте выбор типа проекта, использованный при разработке ядра цифрового блока.
5. Основные правила разработки программных ядер, использованные при реализации ядра.

4.5. Варианты заданий

Все устройства должны иметь сигнал сброса и синхросигнал.

1. Одноканальная 32-разрядная регистровая память, 16 регистров.

Вход: адрес (номер) регистра; код операции 0/1 – чтение / запись; данные для записи.

Выход: прочитанные (/записанные) данные.

2. Реализовать функцию перевода двоичного числа в десятичное.

Вход: двоичное 32-разрядное число.

Выход: число типа Integer.

3. Реализовать блочный перемежитель двоичных чисел, перемежитель меняет местами строки и столбцы квадратной матрицы. Маска для перемежения постоянная.

Вход: массив чисел 32 × 32.

Выход: флаг (1 – выходные данные сформированы, 0 – данных нет); выходные данные или массив «0».

4. Реализовать 32-разрядный сдвиговый регистр. Регистр осуществляет логический или арифметический сдвиг на один разряд влево/ вправо.

Вход: входные данные;

направление сдвига – вправо/влево;

тип сдвига.

Выход: выходные данные.

5. Реализовать 32-разрядный сдвиговый регистр. Регистр осуществляет циклический сдвиг на указанное число разрядов.

Вход: входные данные;

число разрядов для сдвига;

направление сдвига вправо/влево;

Выход: выходные данные.

6. Реализовать АЛБ процессора, обрабатывающего беззнаковые числа. Реализуемые операции: сложение, вычитание.

Вход: входные данные;

код операции.

Выход: выходные данные;

флаги: переполнение, меньше нуля.

7. Реализовать АЛБ процессора. Реализуемые операции – AND, OR, NOR. На обоих портах A и B стоят управляемые инверторы.

Вход: входные 32-разрядные данные;

код операции;

сигналы управления входами (инвертировать/ не инвертировать).

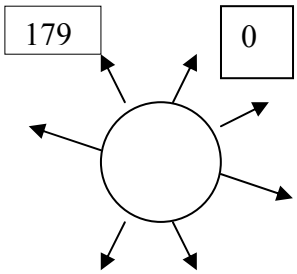
Выход: выходные данные.

8. Реализовать АЛБ. Реализуемые операции – сравнение двух знаковых чисел с фиксированной точкой. Отрицательные числа представлены в дополнительном коде.

Вход: входные 32-разрядные данные.

Выход: выходные данные (01 – A>B, 10 – A<B, 11 – A=B).

9. Разработать устройство обработки информации от круглого радара, определяющее направление движения наблюдаемого объекта (по часовой стрелке, против часовой стрелки, неподвижен). Всего на радаре 180 датчиков, расположенные по окружности. От каждого датчика поступает сигнал 0/1 (нет объекта/ датчик фиксирует объект).



Вход: данные от датчиков радара.

Выход: флаг наличия объекта;
номер датчика;
направление движения (01 – по часовой стрелке, 10 – против часовой стрелки, 11 – объект неподвижен).

10. Счетчик со сбросом, загрузкой, сигналом разрешения – устройство подсчитывает количество изменений входных данных с последнего сигнала «сброс». Счет выполняется только при наличии сигнала «разрешение счета».

Вход: входные 32-разрядные данные;
сброс;
разрешение счета.

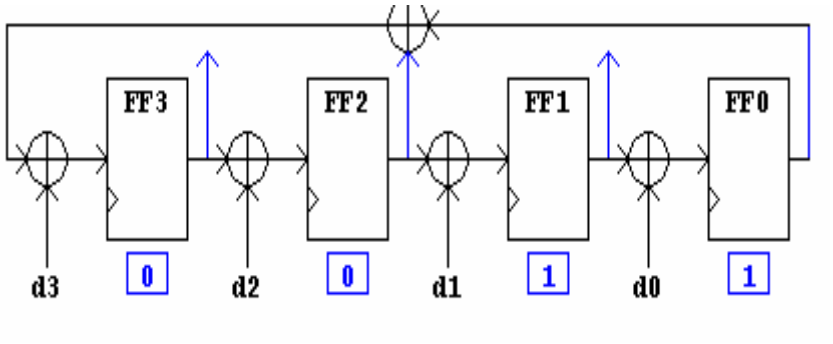
Выход: выходные данные.

11.Устройство управления индикацией семисекционного цифрового табло.

Вход: двоичный код арабской цифры (от 0 до 9).

Выход: сигналы управления.

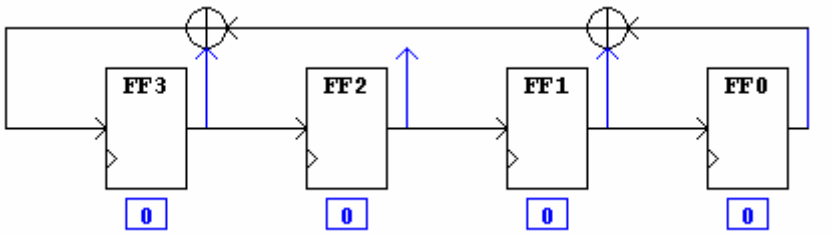
12.Устройство, представленное на рисунке. Компоненты устройства – D-триггер и сумматор по модулю 2.



Вход: d0 … d3.

Выход: вектор состояний триггеров.

13. Реализовать устройство. Компоненты устройства – D-триггер и сумматор по модулю 2.



Выход: вектор состояний триггеров.

14. Устройство умножения/ деления на число, являющееся степенью 2.

Вход: входные 32-разрядные данные;
код операции;
степень 2.

15. Сверточный кодер с параметрами:

– скорость кода ½;

– кодовое ограничение 4;

– задающий полином [13 05].

РЕКОМЕНДАТЕЛЬНЫЙ БИБЛИОГРАФИЧЕСКИЙ СПИСОК

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

2. Куликов, К. В. Основные проблемы проектирования систем на одном кристалле / К. В. Куликов // Новые методологии проектирования изделий микроэлектроники (New design methodologies) : материалы междунар. науч.-техн. конф. – Владимир, 2004. – С. 35 – 37.

3. Уэйкерли, Дж. Ф. Проектирование цифровых устройств : в 2т. / Дж. Ф. Уэйкерли. – М. : Постмаркет, 2002. – Т. 1. – 544 с.; Т. 2. – 528 с. – ISBN 5-901095-12-X.

ОГЛАВЛЕНИЕ