Министерство образования Республики Беларусь

Учреждение образования «Полоцкий государственный университет»

С. В. Калинцев

АВТОМАТИЗАЦИЯ ПРОЕКТИРОВАНИЯ ЭВМ

Методические указания
к выполнению лабораторных работ для студентов специальности
1-40 02 01 «Вычислительные машины, системы и сети»

Новополоцк
ПГУ
2010

УДК 001.63(075.8)

Одобрено и рекомендовано к изданию методической комиссией факультета информационных технологий в качестве методических указаний (протокол № 1 от 21.01.2010)

Кафедра вычислительных систем и сетей

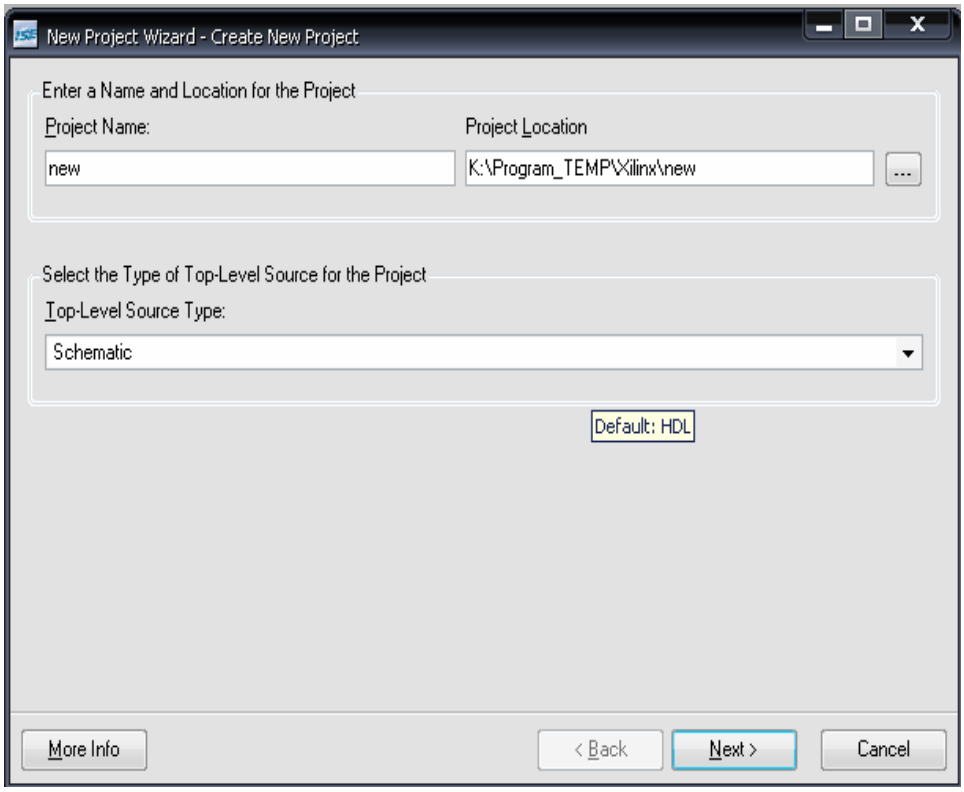
Рецензенты:

канд. техн. наук, доц. каф. радиоэлектроники В. Ф. ЯНУШКЕВИЧ
ст. преп. каф. вычислительных систем и сетей Д. Г. РУГОЛЬ

ОБЩИЕ СВЕДЕНИЯ О СИСТЕМЕ САПР XILINX WEBPACK ISE 11.1

1. Создание проекта
   1.1. Запустить WebPack ISE 11.1. Выбрать пункт меню «File» -> «New project» для создания нового проекта.

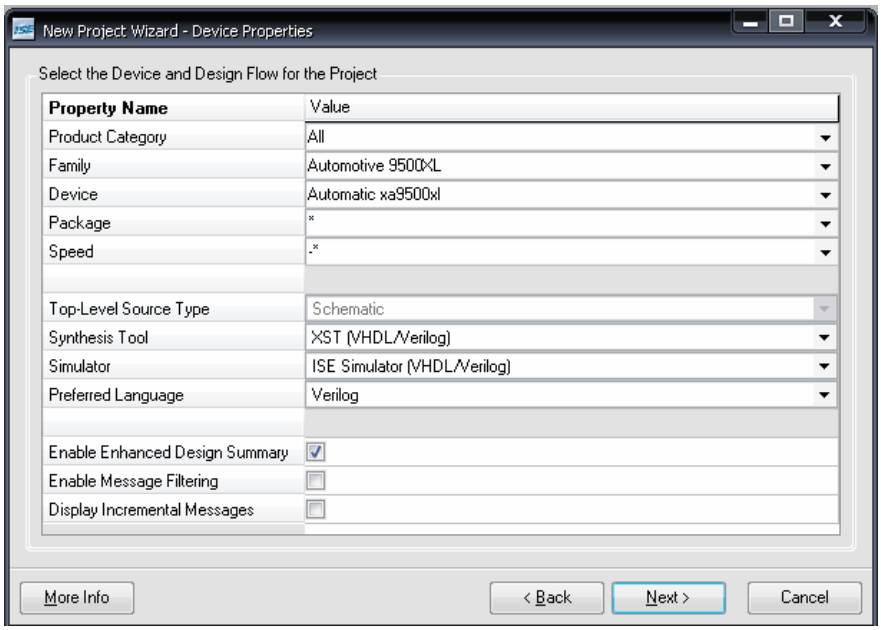
   1.2. В появившемся окне выбрать имя проекта (например, «NEW») и указать тип проекта (HDL или Schematic) в зависимости от категории разрабатываемого или проектируемого устройства.

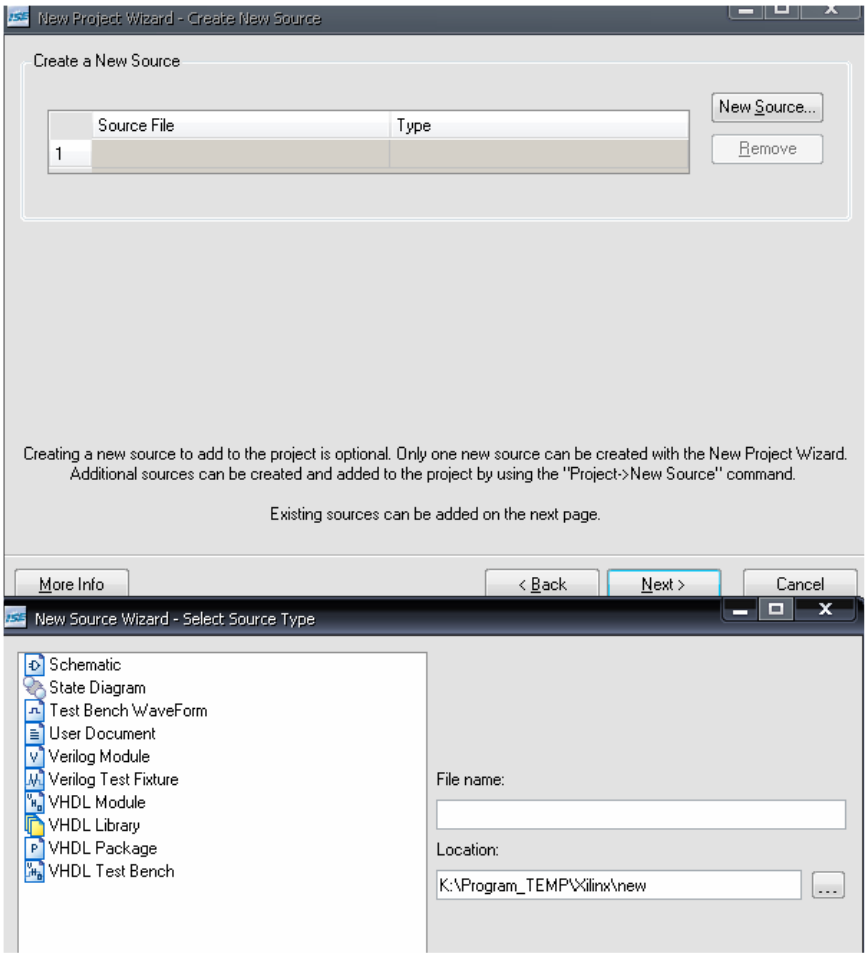


   1.3. В следующем окне выбрать необходимый тип языка программирования (VHDL или VERILOG).

   1.4. Выбрать пункт «new source» в следующем окне и определить тип используемого устройства из списка перечисленных.

   1.5. Завершить создание проекта, при необходимости добавив существующие файлы из других проектов при помощи кнопки «Add existing source».

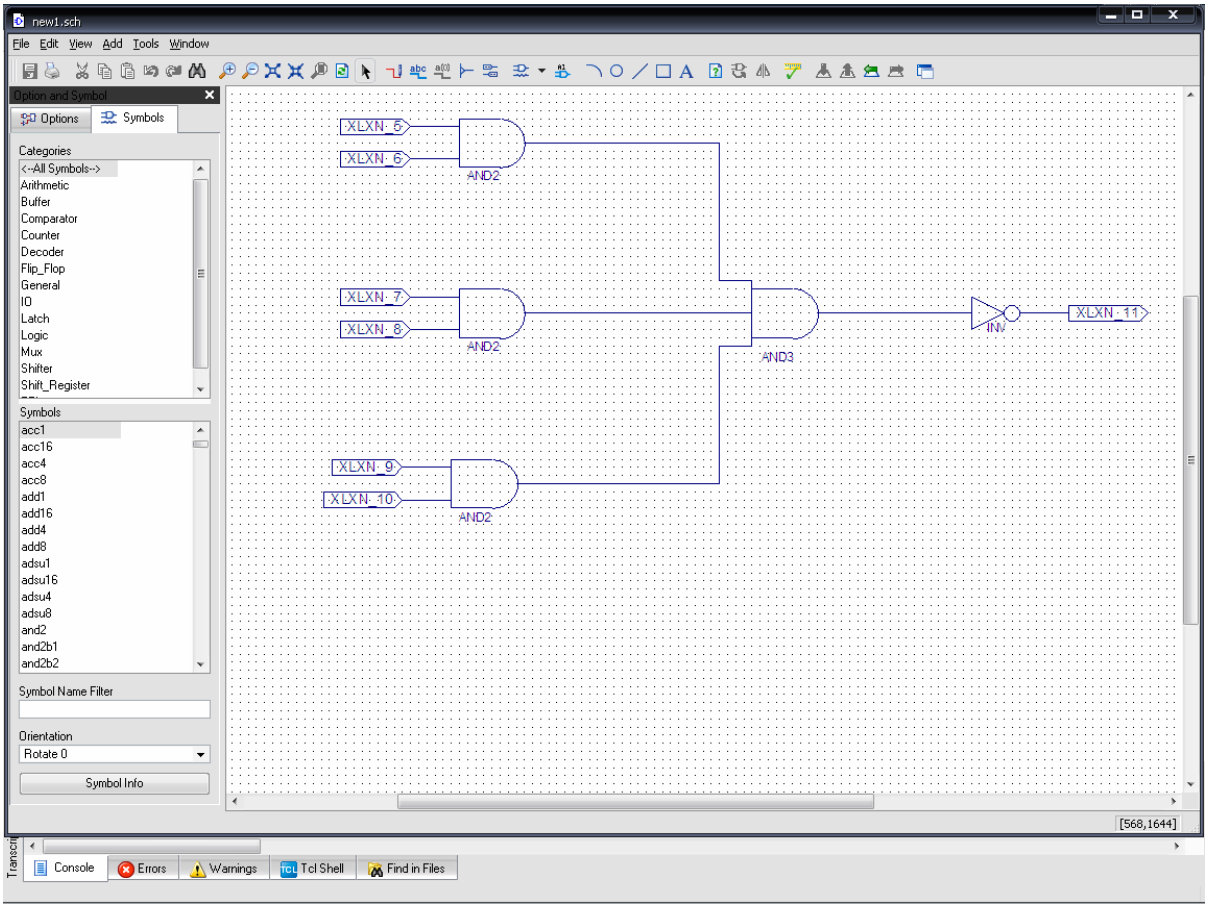

New Project Wizard - Device Properties
Select the Device and Design Flow for the Project
Property Name
Value
Product Category
All
Family
Automotive 9500XL
Device
Automatic xa9500xl
Package
*
Speed
-*
Top-Level Source Type
Schematic
Synthesis Tool
XST (VHDL/Verilog)
Simulator
ISE Simulator (VHDL/Verilog)
Preferred Language
Verilog
Enable Enhanced Design Summary
Enable Message Filtering
Display Incremental Messages
More Info
< Back
Next >
Cancel


New Project Wizard - Create New Source
Create a New Source
Source File
Type
1
New Source...
Remove
Creating a new source to add to the project is optional. Only one new source can be created with the New Project Wizard.
Additional sources can be created and added to the project by using the "Project->New Source" command.
Existing sources can be added on the next page.
More Info
< Back
Next >
Cancel
New Source Wizard - Select Source Type
Schematic
State Diagram
Test Bench WaveForm
User Document
Verilog Module
Verilog Test Fixture
VHDL Module
VHDL Library
VHDL Package
VHDL Test Bench
File name:
Location:
K:\Program_TEMP\Xilinx\new

2. Разработка проекта

В открывшемся справа окне схемотехнического редактора добавить необходимые элементы из библиотеки символов.

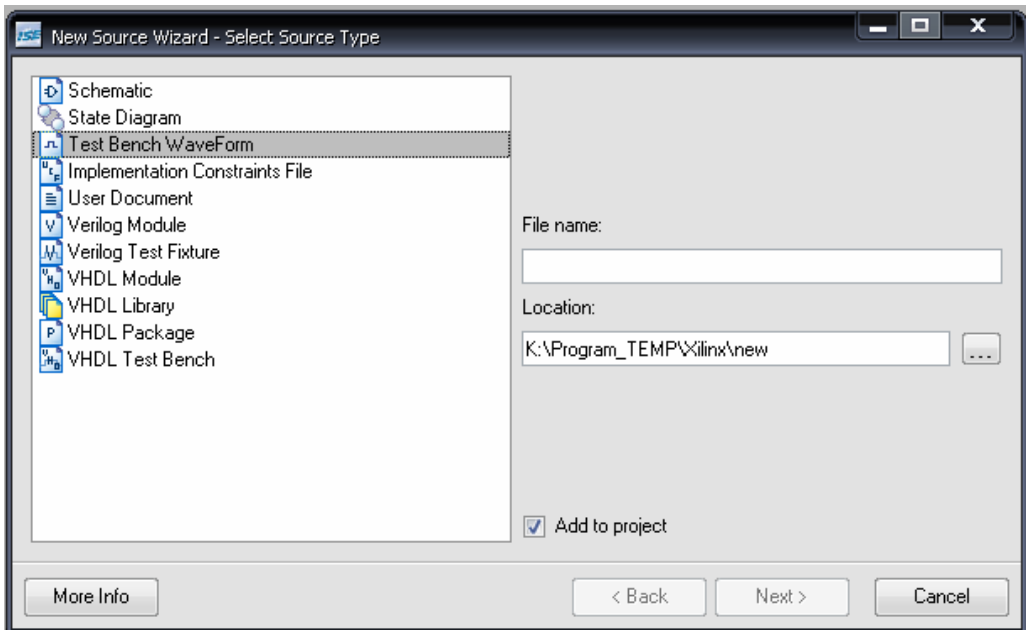
2.1. При помощи библиотеки имеющихся элементов создаем схему, указанную в соответствующем задании.

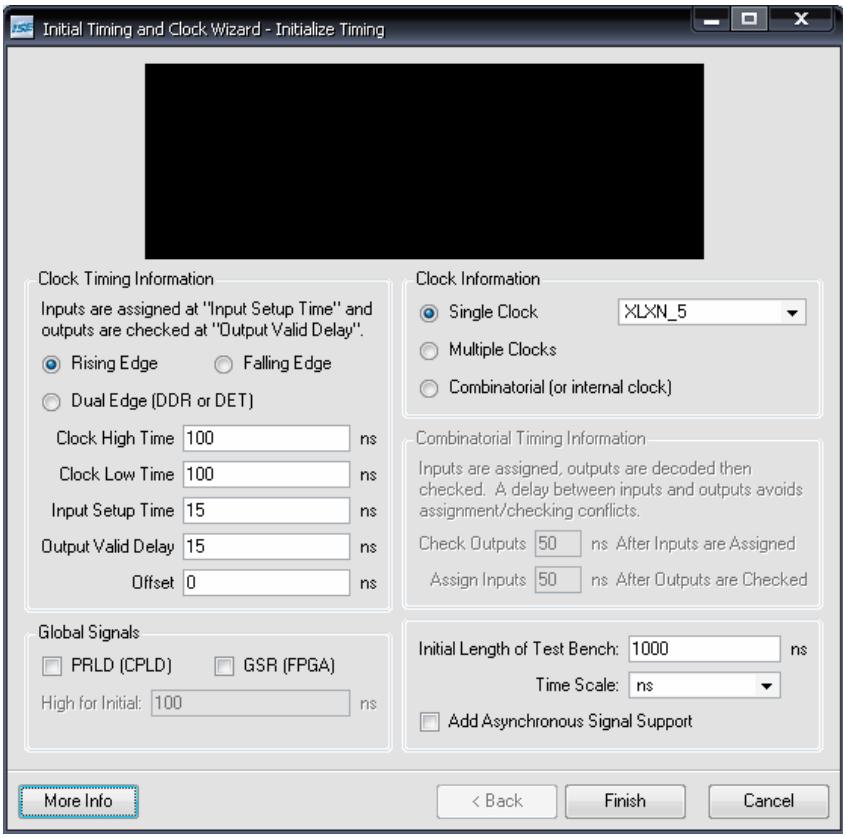


2.2. Компилируем схему, проверяя ее работоспособность. В диалоговом окне внизу будут отображены ошибки, если они допущены при разработке схемы.

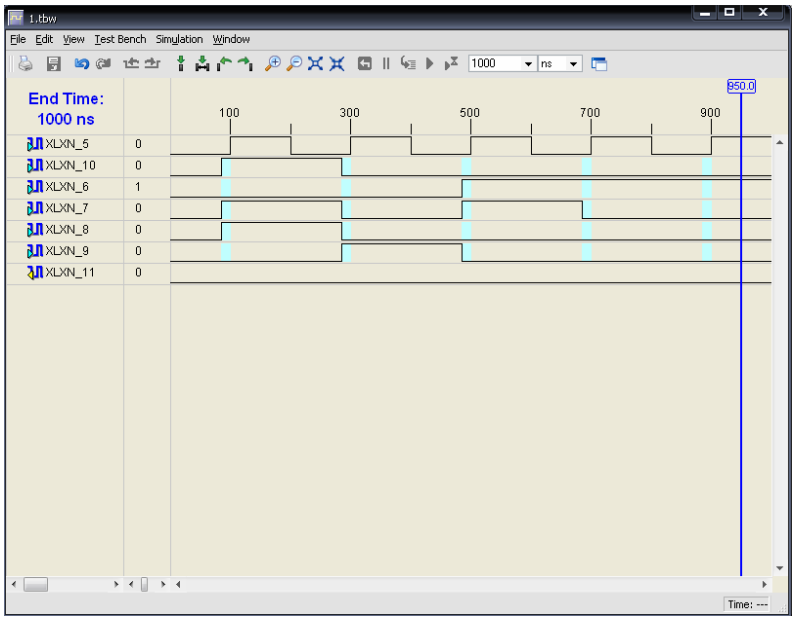
2.3. Создаем файл временной диаграммы при помощи инструмента «Create new source». Выбираем в появившемся окне «Test Bench WaveForm».

2.4. В появившемся окне устанавливаем необходимые нам параметры временной диаграммы. В большинстве случаев целесообразно оставлять их по умолчанию.

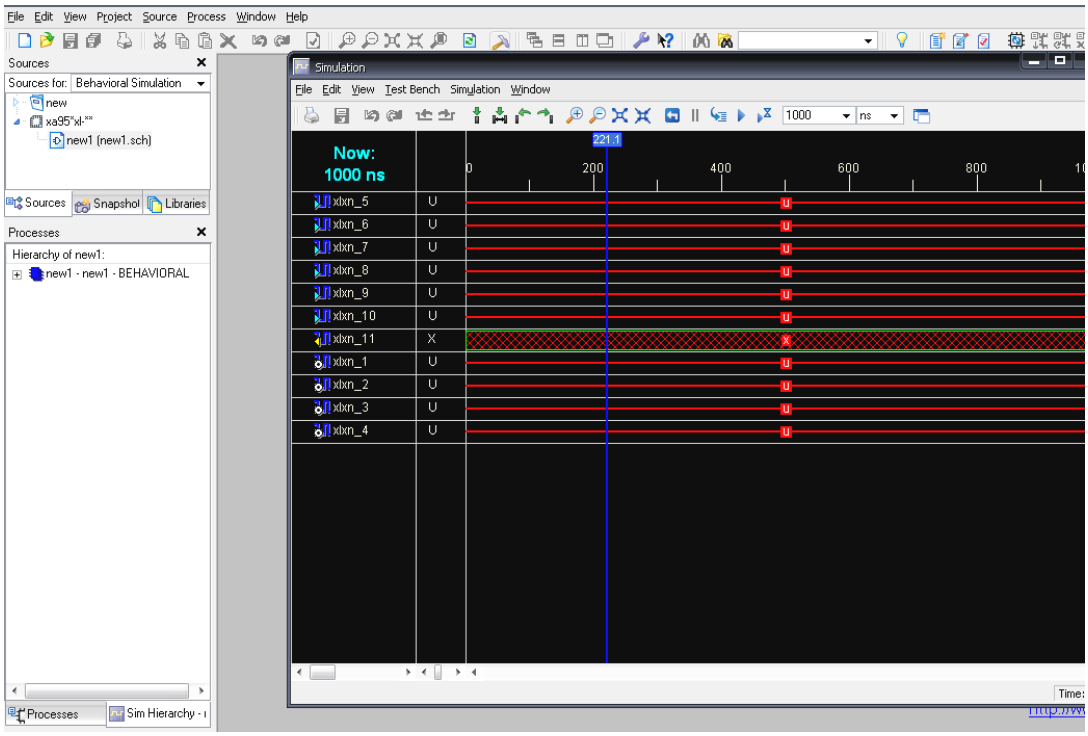




2.5. Устанавливаем необходимые уровни входных сигналов при помощи имеющейся стандартной библиотеки сигналов либо вручную.



2.6. Сохраняем шаблон временной диаграммы. Далее в правом верхнем окне выбираем пункт «Behavioral Simulation», а следом в правом нижнем – пункт «Simulate Behavioral Model». Получаем временную диаграмму моделирования исходной схемы.



2.7. Аналогичным образом поступаем не только при моделировании схематического файла, но и при описании устройства на языке программирования VHDL или VERILOG.

ЛАБОРАТОРНАЯ РАБОТА № 1

Цель работы: ознакомиться с интерфейсом и возможностями сред ActiveHDL, WebPack ISE и получить практические навыки в создании и моделировании простейших комбинационных схем.

Порядок выполнения лабораторной работы:

1. Ознакомиться с общими сведениями о системе Xilinx WebPack ISE 11.1.
2. Получить у преподавателя вариант простейшей схемы и реализовать его в схематическом редакторе. Провести компиляцию заданной схемы и построить временную диаграмму ее работы. Показать результаты проектирования преподавателю.
3. Приступить к выполнению основного варианта задания, указанного преподавателем. Минимизировать логические функции в соответствии с правилами.
4. Выполнив задание, показать результаты преподавателю.
5. Оформить отчет в соответствии с нижеуказанными требованиями.

Отчет по лабораторной работе сдается преподавателю в распечатанном виде (с одной стороны листа формата А4).

Структура отчета:

1. Титульный лист установленного образца (см. прил.).
2. Содержание.
3. Схема, заданная преподавателем, ее описание и временная диаграмма ее работы.
4. Схема варианта лабораторной работы, таблица истинности и минимизации. Временная диаграмма работы схемы.
5. Заключение по результатам проектирования.

Варианты заданий к лабораторной работе № 1

Вариант1

$$Y1 = X4 \wedge \overline{\left(X5 \wedge \overline{(X4)} \wedge X4 \wedge \overline{(X1)}\right)}$$

$$Y2 = X3 \wedge X6 \vee \overline{\left(X2 \wedge \overline{(X4)} \vee X8\right)} \wedge X7 \wedge X8$$

$$Y3 = X4 \wedge \overline{\left(X8 \wedge \overline{\left(X6 \vee \overline{(X3)} \vee X8\right)}\right)}$$

$$Y4 = X4 \wedge \overline{\left(X7 \vee X7 \wedge \overline{(X4)} \wedge \overline{(X5)}\right)}$$

$$Y5 = X4 \vee \overline{\left(X1 \wedge \overline{\left(X2 \vee \overline{(X8)}\right)}\right)}$$

Вариант2

$$Y1 = X1 \wedge \overline{\left(X8 \oplus X8 \vee X6 \wedge \overline{(X3)}\right)} \wedge X5$$

$$Y2 = X4 \vee \overline{\left(X3 \wedge \overline{(X3)}\right)} \wedge X5 \vee \overline{(X4)}$$

$$Y3 = X5 \wedge \overline{(X5 \oplus X3 \oplus X1 \wedge X8 \wedge X7)}$$

$$Y4 = X5 \wedge \overline{\left(X1 \wedge \overline{(X5 \oplus X5 \vee X4)} \vee X1\right)}$$

$$Y5 = X7 \wedge \overline{\left(X5 \wedge \overline{\left(X2 \vee \overline{(X1)} \vee X2\right)}\right)}$$

Вариант3

$$Y1 = X4 \wedge \overline{\left(X8 \oplus X5 \oplus X1 \vee \overline{(X5)} \vee X2\right)}$$

$$Y2 = X4 \oplus X6 \oplus X7 \vee \overline{(X8)} \vee \overline{(X7)}$$

$$Y3 = X8 \oplus X2 \vee \overline{(X4)} \vee \overline{(X7)} \wedge X2$$

$$Y4 = X1 \wedge \overline{(X3 \vee X4 \wedge X6 \oplus X1 \vee X3 \wedge X5 \oplus X5)}$$

$$Y5 = X8 \wedge X5 \wedge \overline{(X1)} \wedge \overline{(X2)} \vee \overline{(X3)}$$

Вариант4

$$Y1 = X5 \wedge X3 \vee \overline{(X4 \oplus X4 \wedge X4 \wedge X6 \vee X7)} \oplus X5$$

$$Y2 = X3 \wedge \overline{(X1 \oplus X1 \vee X3 \vee X8 \wedge X2 \vee X1)}$$

$$Y3 = X4 \wedge \overline{\left(X6 \vee \overline{(X6 \wedge X4)} \vee \overline{(X6)}\right)}$$

$$Y4 = X6 \wedge \overline{\left(X4 \vee \overline{(X2)} \oplus X2 \wedge X1\right)}$$

$$Y5 = X5 \wedge \overline{\left(X4 \vee \overline{(X3)} \wedge \overline{(X1)}\right)}$$

Вариант5

$$Y1 = X2 \oplus X8 \vee \overline{\left(X4 \vee X3 \wedge X1 \wedge X8 \vee \overline{(X2)}\right)}$$

$$Y2 = X4 \vee \overline{(X5)} \wedge X8 \wedge \overline{(X6)} \vee \overline{(X1)}$$

$$Y3 = X2 \oplus X2 \vee X3 \vee X7 \vee X3 \vee \overline{(X1)} \vee X2$$

$$Y4 = X6 \oplus X1 \oplus X1 \wedge \overline{(X2)} \wedge X8 \vee X7$$

$$Y5 = X3 \vee X7 \wedge \overline{(X2)} \vee \overline{(X6)} \wedge \overline{(X8)}$$

Вариант6

$$Y1 = X3 \oplus X4 \oplus X3 \wedge X3 \wedge \overline{(X6)}$$

$$Y2 = X7 \vee \overline{(X4)} \oplus X6 \wedge \overline{(X6)} \wedge X8$$

$$Y3 = X6 \vee X5 \wedge \overline{\left(X1 \vee \overline{(X1)} \oplus X8\right)}$$

$$Y4 = X7 \wedge \overline{\left(X6 \wedge \overline{(X7)} \wedge X7\right)} \vee \overline{(X8)} \wedge X3$$

$$Y5 = X4 \vee \overline{(X7)} \wedge \overline{(X8)} \wedge X3 \vee X6 \vee X4$$

Вариант7

$$Y1 = X6 \oplus X7 \vee \overline{\left(X5 \oplus X5 \wedge \overline{(X2)}\right)} \vee X7$$

$$Y2 = X8 \wedge \overline{(X8)} \wedge X1 \vee \overline{(X8)} \oplus X6$$

$$Y3 = X1 \vee \overline{\left(X8 \vee \overline{\left(X8 \vee X5 \wedge \overline{(X3)} \vee X8\right)} \vee X3\right)}$$

$$Y4 = X1 \vee \overline{(X6)} \vee X8 \oplus X7 \wedge X3 \wedge \overline{(X1)}$$

$$Y5 = X1 \oplus X4 \wedge \overline{(X6)} \vee X3 \oplus X5$$

Вариант8

$$Y1 = X7 \vee X6 \vee X1 \oplus X2 \vee \overline{(X4)} \vee \overline{(X2)}$$

$$Y2 = X5 \wedge \overline{\left(X4 \wedge \overline{(X5)} \wedge \overline{(X3)}\right)} \oplus X2$$

$$Y3 = X7 \vee X1 \wedge X3 \vee \overline{(X5)} \wedge \overline{(X2)} \oplus X5$$

$$Y4 = X3 \vee X6 \wedge X7 \vee X2 \wedge \overline{(X6)} \wedge X6 \oplus X8$$

$$Y5 = X8 \vee \overline{\left(X5 \vee X2 \oplus X5 \wedge X5 \wedge \overline{(X7)}\right)}$$

Вариант9

$$Y1 = X2 \wedge \overline{\left(X1 \wedge X4 \oplus X8 \wedge \overline{(X8)} \right)}$$

$$Y2 = X5 \oplus X5 \oplus X1 \vee \overline{(X6)} \wedge \overline{(X2)} \vee X2$$

$$Y3 = \overline{X5 \wedge \left(X7 \vee \overline{(X6)} \right)} \wedge X4 \oplus X2$$

$$Y4 = X7 \oplus X7 \wedge \overline{(X5)} \oplus X5 \wedge \overline{(X5)}$$

$$Y5 = X3 \oplus X2 \oplus X3 \vee X3 \wedge X4 \vee \overline{(X4)} \wedge X4 \wedge X3$$

Вариант10

$$Y1 = X6 \vee \overline{\left(X3 \wedge \overline{\left(X6 \vee \overline{(X7)} \right)} \right)} \wedge X8$$

$$Y2 = X8 \oplus X6 \oplus X5 \vee X3 \oplus X4 \wedge X8$$

$$Y3 = X7 \vee X8 \wedge \overline{\left(X4 \vee X7 \wedge \overline{(X6)} \vee \overline{(X7)} \right)}$$

$$Y4 = X3 \vee X4 \oplus X8 \vee X3 \wedge X6 \vee X2 \wedge X3 \oplus X1 \vee X2$$

$$Y5 = X7 \vee X3 \wedge \overline{(X7)} \wedge X2 \vee X5 \vee \overline{(X1)}$$

Вариант11

$$Y1 = X3 \vee X2 \wedge \overline{\left(X3 \wedge \overline{(X5)} \vee \overline{(X5)} \right)} \wedge X1$$

$$Y2 = X1 \wedge X4 \vee X3 \oplus X1 \oplus X4 \vee X7 \oplus X1$$

$$Y3 = X2 \vee \overline{\left(X2 \wedge X5 \wedge \overline{(X2 \wedge X4 \vee X6)} \vee \overline{(X7)} \right)}$$

$$Y4 = X8 \vee X4 \wedge X8 \wedge \overline{(X3)} \vee \overline{(X8)} \wedge X6 \wedge X1 \wedge X5$$

$$Y5 = X8 \wedge X7 \wedge X4 \vee \overline{(X1)} \wedge \overline{(X6)} \wedge X7$$

Вариант12

$$Y1 = X7 \wedge X1 \vee \overline{(X1)} \wedge X8 \vee X4 \oplus X2 \vee X6$$

$$Y2 = X5 \wedge \overline{\left(X8 \wedge \overline{(X6)} \oplus X5 \right)} \oplus X4$$

$$Y3 = X5 \wedge \overline{\left(X8 \vee \overline{\left(X3 \vee X5 \wedge X5 \wedge \overline{(X5)} \right)} \right)}$$

$$Y4 = X1 \vee X7 \oplus X8 \wedge X3 \vee X3 \oplus X2 \wedge X3$$

$$Y5 = X4 \wedge X2 \oplus X2 \vee \overline{(X4)} \oplus X7$$

Вариант13

$$Y1 = X5 \wedge X4 \wedge \overline{\left(X5 \vee \overline{(X3)}\right)} \oplus X6$$

$$Y2 = X7 \vee X8 \wedge \overline{\left(X7 \wedge \overline{\left(X4 \wedge \overline{(X4)}\right)}\right)}$$

$$Y3 = X3 \wedge X1 \oplus X7 \vee \overline{(X1)} \wedge \overline{(X8)}$$

$$Y4 = X4 \wedge \overline{\left(X2 \wedge \overline{(X6 \vee X3)} \vee X8\right)} \vee X2$$

$$Y5 = X6 \vee X7 \wedge X8 \oplus X2 \oplus X6 \wedge \overline{(X8)} \vee X5 \vee X5$$

Вариант14

$$Y1 = X7 \vee \overline{\left(X4 \vee X4 \vee \overline{(X3)} \wedge \overline{(X8)}\right)}$$

$$Y2 = X6 \vee \overline{(X5 \vee X5 \oplus X6 \vee X7)} \wedge \overline{(X2)}$$

$$Y3 = X1 \wedge X2 \vee X1 \oplus X7 \vee X3 \vee X4 \vee \overline{(X8)} \vee X5$$

$$Y4 = X6 \wedge X1 \wedge \overline{\left(X5 \oplus X2 \wedge \overline{(X5)} \vee X5\right)}$$

$$Y5 = X7 \vee \overline{\left(X4 \vee X1 \oplus X3 \wedge \overline{(X7 \oplus X8)}\right)}$$

Вариант15

$$Y1 = X3 \wedge \overline{\left(X1 \oplus X7 \wedge X2 \wedge \overline{(X8)}\right)}$$

$$Y2 = X1 \vee X7 \wedge X7 \wedge \overline{\left(X7 \vee X3 \vee \overline{(X5)}\right)} \wedge X1$$

$$Y3 = X2 \vee \overline{\left(X7 \vee X5 \wedge \overline{(X2)} \wedge \overline{(X8)}\right)}$$

$$Y4 = X5 \vee \overline{\left(X6 \wedge X6 \oplus X2 \wedge \overline{(X7)} \oplus X5\right)}$$

$$Y5 = X8 \oplus X6 \wedge X7 \wedge \overline{(X2)} \oplus X4$$

Вариант16

$$Y1 = X2 \wedge X4 \oplus X6 \oplus X5 \oplus X8 \oplus X6 \vee X2$$

$$Y2 = X8 \vee \overline{\left(X5 \vee \overline{(X4 \oplus X3)} \vee X1\right)}$$

$$Y3 = X3 \oplus X4 \wedge \overline{\left(X3 \oplus X1 \vee \overline{(X5)}\right)} \wedge X6$$

$$Y4 = X7 \vee X6 \wedge \overline{\left(X8 \oplus X3 \vee X8 \wedge \overline{(X7)}\right)}$$

$$Y5 = X7 \wedge X1 \vee \overline{\left(X4 \oplus X4 \vee \overline{(X4)}\right)}$$

ЛАБОРАТОРНАЯ РАБОТА № 2

Цель работы: ознакомиться с методами полиномиальной аппроксимации функциональных зависимостей и их реализацией на языке VHDL в среде ActiveHDL.

1. Общие сведения

1.1. На практике при моделировании различных процессов, в частности, экономических, физических, технических, социальных широко используются те или иные способы вычисления приближенных значений функций по известным их значениям в некоторых фиксированных точках.

1.2. Такого рода задачи приближения функций часто возникают при построении приближенных формул для вычисления значений характерных величин исследуемого процесса по табличным данным, полученным в результате эксперимента; при численном интегрировании, дифференцировании, решении дифференциальных уравнений и т.д.; при необходимости вычисления значений функций в промежуточных точках рассматриваемого интервала; при определении значений характерных величин процесса за пределами рассматриваемого интервала, в частности, при прогнозировании.

1.3. Если для моделирования некоторого процесса, заданного таблицей, построить функцию, приближенно описывающую данный процесс на основе метода наименьших квадратов, она будет называться аппроксимирующей функцией (регрессией), а сама задача построения аппроксимирующих функций – задачей аппроксимации.

1.4. В Excel при построении регрессий имеется возможность добавлять выбранные регрессии (линии тренда – trendlines) в диаграмму, построенную на основе таблицы данных для исследуемой характеристики процесса (доступно лишь при наличии построенной диаграммы);

1.5. Полиномиальная линия тренда полезна для описания характеристик, имеющих несколько ярко выраженных экстремумов (максимумов и минимумов). Выбор степени полинома определяется количеством экстремумов исследуемой характеристики. Так, по-

лином второй степени может хорошо описать процесс, имеющий только один максимум или минимум; полином третьей степени – не более двух экстремумов; полином четвертой степени – не более трех экстремумов и т.д.

1.6. В этом случае линия тренда строится в соответствии с уравнением $y = c_0 + c_1x + c_2x^2 + c_3x^3 + c_4x^4 + c_5x^5 + c_6x^6$, где коэффициенты c_0, c_1, c_2, ... c_6 – константы, значения которых определяются в ходе построения.

2. Порядок реализации полиномиальной аппроксимации функциональной зависимости

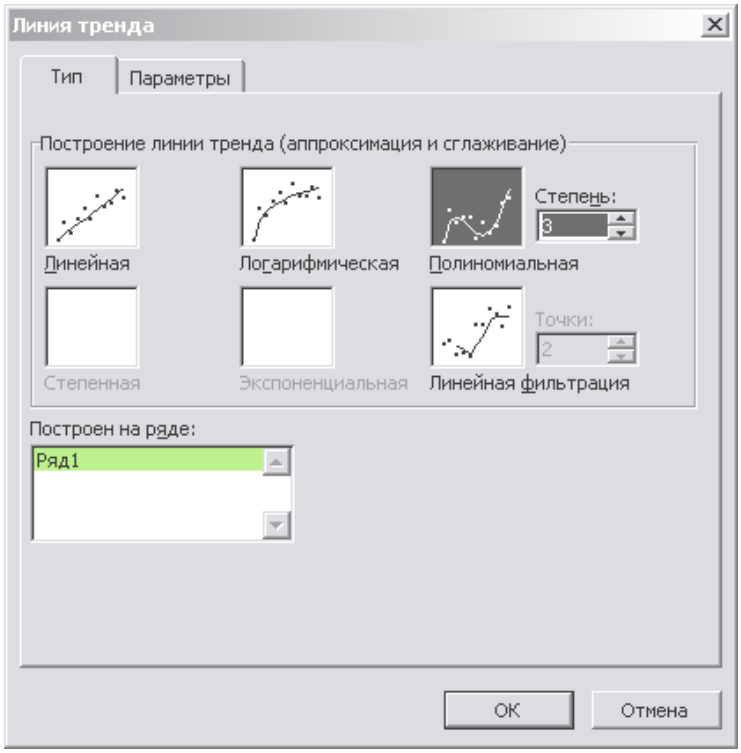
2.1. По графику определить значения функции в точках 1, 2, 3, … .

2.2. Занести значения в таблицу в Microsoft Excel.

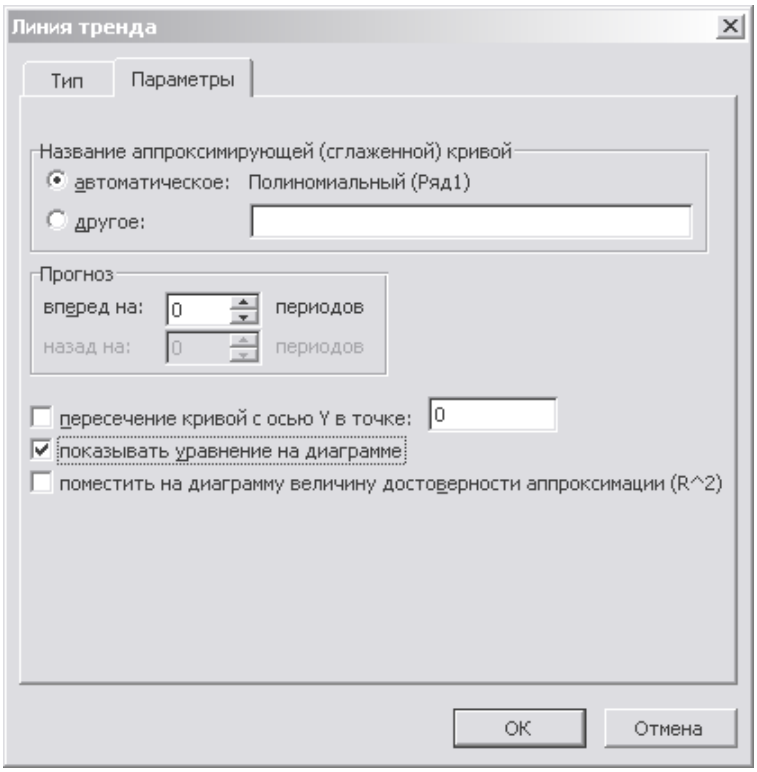
2.3. Построить график по таблице.

2.4. Выбрав график, войти в меню «Диаграмма» – «Добавить линию тренда».

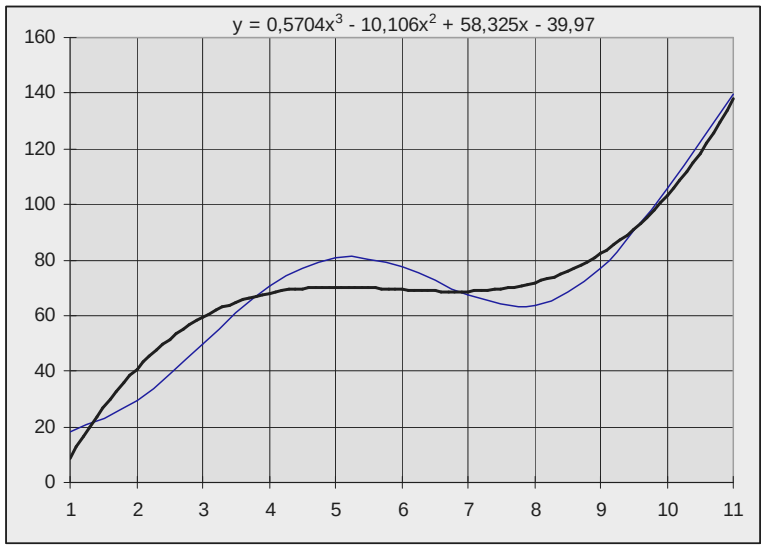
2.5. Выбрать тип аппроксимации «Полиномиальная». Выбрать требуемую степень полинома.



2.6. На вкладке «Параметры» установить флажок «Показывать уравнение на диаграмме».



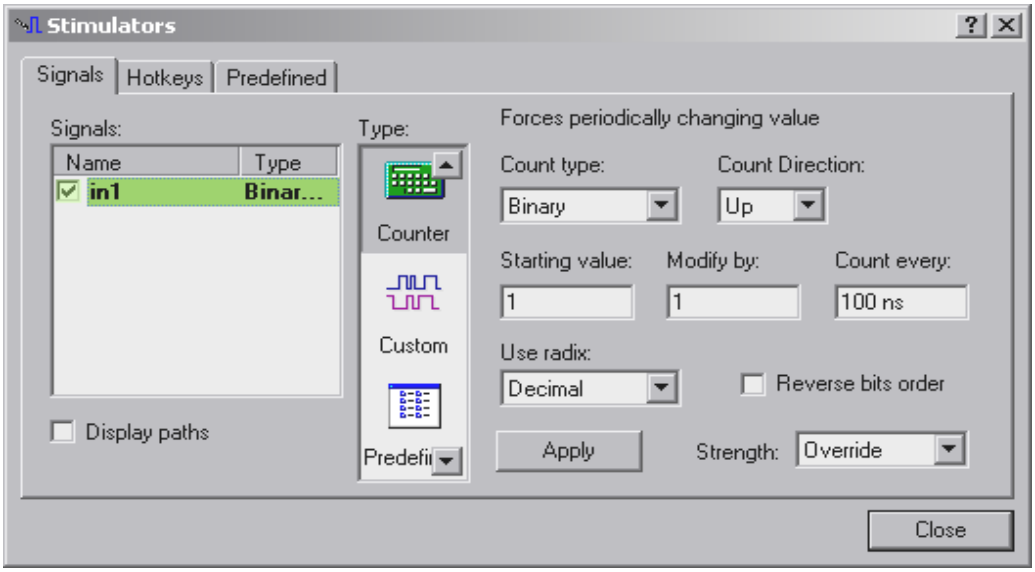
2.7. На полученной диаграмме будет выведено уравнение и график аппроксимированной функции.



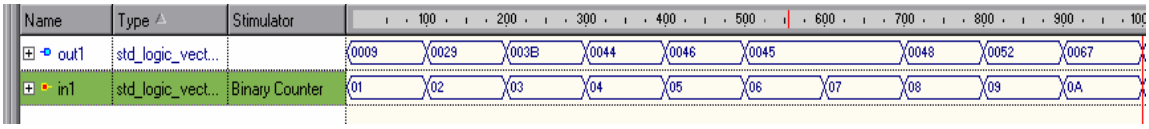
2.8. Создаем архитектуру и процесс в ней. Входные и выходные параметры должны иметь такую размерность, чтобы вмещать все входные и выходные значения. В списке чувствительности процесса должен быть задан входной порт. Вычислить полученный полином с учетом преобразования типов.

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

entity pol is
     port(
            -- Входное число разрядностью 8 бит
            in1 : in std_logic_vector (7 downto 0);
            -- Результирующее число разрядностью 16 бит
            out1 : out std_logic_vector (15 downto 0)
         );
end pol;

architecture pol of pol is
begin
    process(
    in1 )
    variable x : INTEGER;
    variable y : INTEGER;
    variable sum : real;
    variable cur : real;
    begin
        -- Исходное целое число переводим в вещественный формат
        x := conv_integer( unsigned( in1 ) );
        cur := real(x);
        -- Аппроксимирующая функция
        -- y = 0,5704x3 - 10,106x2 + 58,325x - 39,97
        sum := ( cur**3 ) * (0.5704) + ( cur**2 ) * (-10.106) + ( cur ) * (58.325) - 39.97;
        -- Результат необходимо преобразовать в целое число,
        -- а затем в битовый вектор
        y := integer( sum );
        out1 <= std_logic_vector( conv_unsigned (y, 16 ) );
    end process;
end pol;
```

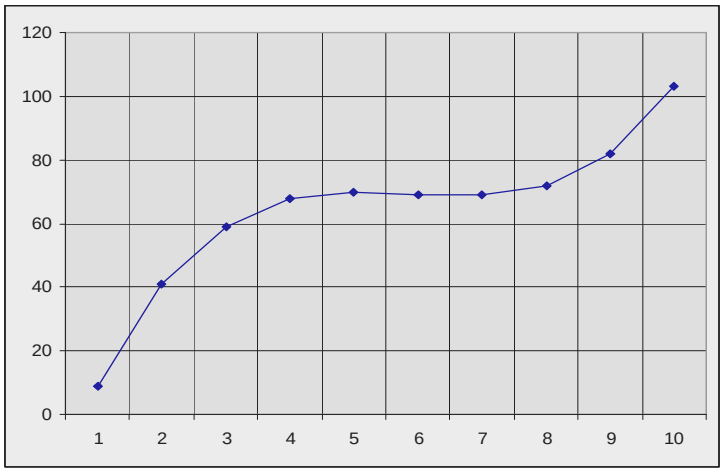
2.9. Создать Waveform, в котором для входного сигнала установлен тип «Binary counter».



2.10. Произвести операцию «Симуляция» для заданного диапазона входных значений.



2.11. Полученные значения свести в таблицу и построить по ним график.



3. Создание синтезируемой реализации аппроксимирующей функции

3.1. При программировании синтезируемых вычислений следует учесть несколько особенностей:

3.1.1. Для синтезируемой операции деления второй операнд должен быть *степенью двойки*. В этом случае операция деления эквивалентна операции сдвига вправо.

3.1.2. Для операции возведения в степень возводимое число должно быть равно *степени двойки*. В этом случае операция эквивалентна сдвигу влево.

3.1.3. При синтезе нельзя использовать тип **real**.

3.2. Возведение в степень необходимо заменить несколькими умножениями.

3.3. Все дробные коэффициенты должны быть приведены к целым числам. Этого можно достигнуть, представив каждый коэффициент как результат деления двух целых чисел. Например, выражение 0.5*x представляется как x*5/10. Необходимо учесть, что целесообразнее сначала выполнять умножение, а затем деление, чтобы не произошло значительной потери точности.

3.4. Все операции деления должны быть приведены к делению на степень двойки. Это необходимо учесть при формировании коэффициентов.

Например, x*0.5 представляется как x*(0.5*16)/16 = x*8/16.

3.5. В связи с невозможностью использования чисел с плавающей точкой и для повышения точности входные значения и результат функции должны быть промасштабированы. Например, входные значения увеличены в 128 раз, и результат должен быть увеличен

в 128 раз. Таким образом, на вход функции должны подаваться значения 128, 256, 384 и т.д., а полученный результат необходимо будет разделить на 128 перед построением графика. При масштабировании чисел необходимо учитывать степени параметра функции.

3.6. Для примера (см. п. 2) промасштабируем все числа в 16 раз. Получим: y : = (x*x*x) * 9 / (16*16*16) + (x*x) * (–162) / (16*16) + x * 933/ 16 - 639;

3.7. Как было указано выше, для повышения точности деление необходимо выполнять как можно позже. Поэтому полученное выражение можно преобразовать так:

y := (x*x*x) * 9 / 16;

y := (y + (x*x) * (–162)) / 16;

y := (y + x * 933)/(16);

y := y – 639;

3.8. При выборе коэффициента масштабирования необходимо предусмотреть, чтобы максимальное вычисляемое значение не превышало максимального значения типа INTEGER. Например, если выбрать масштабирующий коэффициент 256, а не 16, то максимально возможное число (при интервале от 1 до 10) будет $(10*256)^3 = 16777216000$, что заведомо больше 2^{32} и приведет к переполнению.

3.9. Для полученной зависимости провести операцию «Симуляция» и построить график.

3.10. Разместить код на кристалле. При размещении необходимо учитывать, что будет занята значительная площадь кристалла, а на некоторых малых кристаллах размещение может быть неудачным.

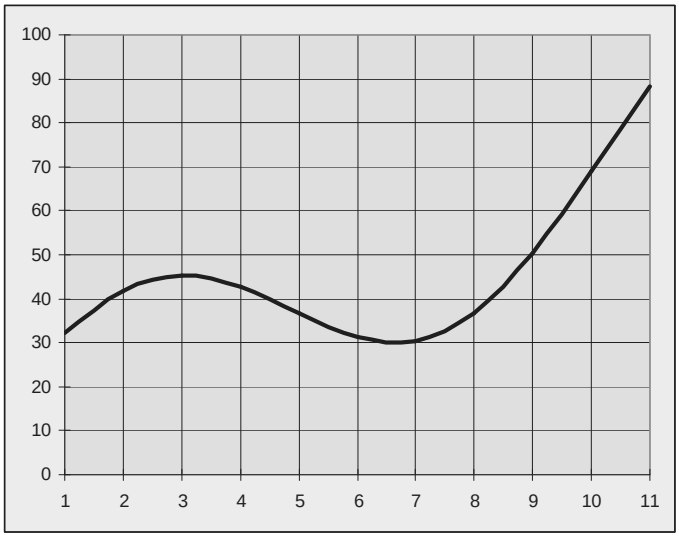
Порядок выполнения работы:

1. Ознакомиться с требованиями методических указаний по выполнению лабораторной работы.
2. Изучить методы полиномиальной аппроксимации функциональных зависимостей.
3. Изучить возможности языка VHDL, применяемые для реализации вычислений полиномов.
4. Реализовать программу на языке VHDL, выполняющую аппроксимацию функциональной зависимости (см. индивидуальное задание).
5. Реализовать синтезируемую программу на языке VHDL, выполняющую аппроксимацию функциональной зависимости.
6. Оформить отчет.

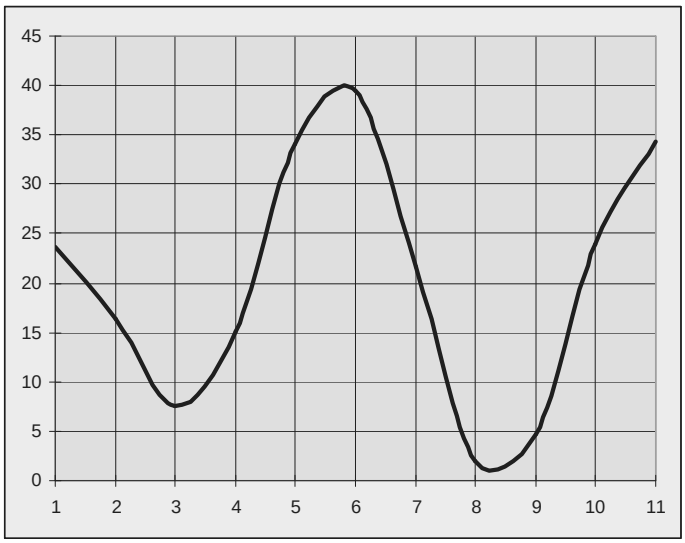
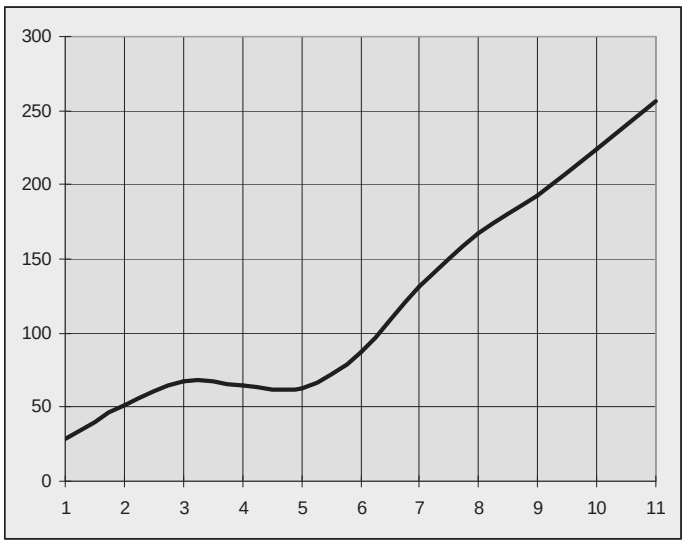
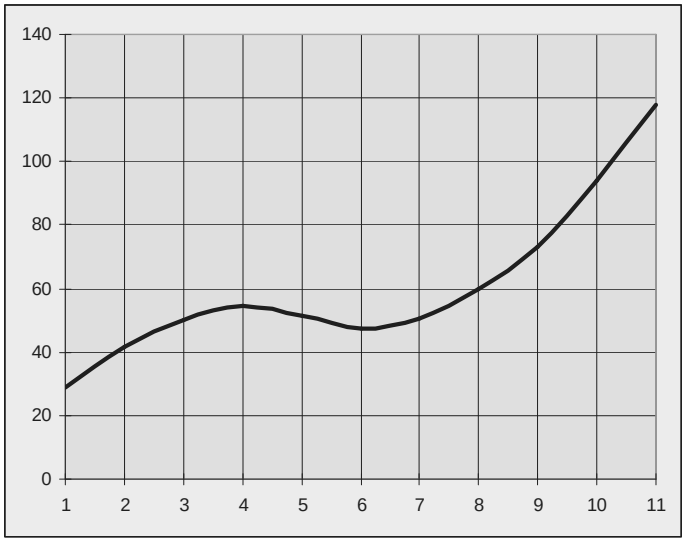
Структура отчета:

1. Название и цель работы.
2. Индивидуальное задание на лабораторную работу.
3. Таблица и график исходной функциональной зависимости.
4. Исходный VHDL-текст программы.
5. Экранная копия тестового Waveform.
6. Таблица и график значений аппроксимирующей функции.
7. Исходный VHDL-текст синтезируемой программы.
8. Экранная копия тестового Waveform для синтезируемой программы.
9. Таблица и график значений аппроксимирующей функции для синтезируемой программы.
10. Вывод по лабораторной работе с учетом индивидуального задания.

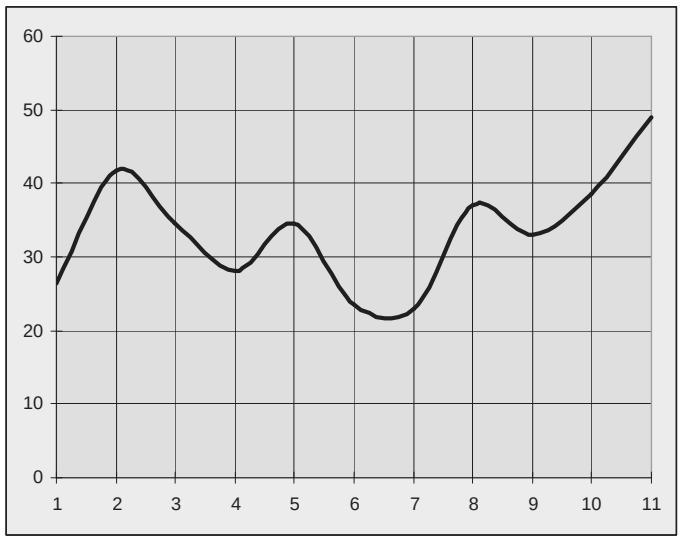
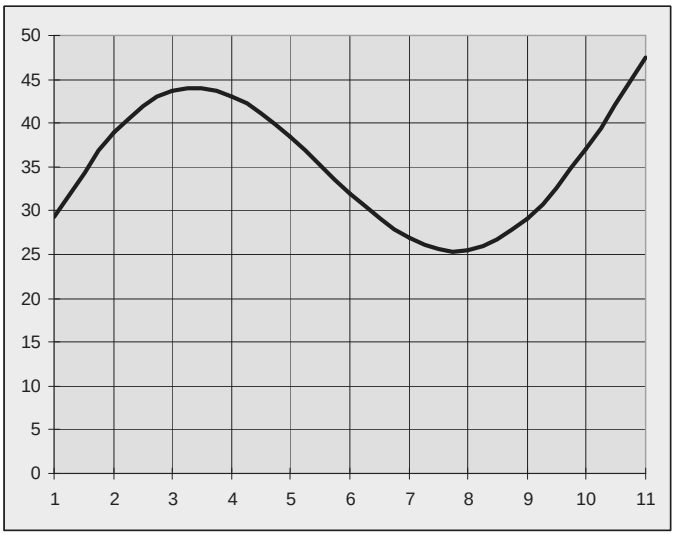
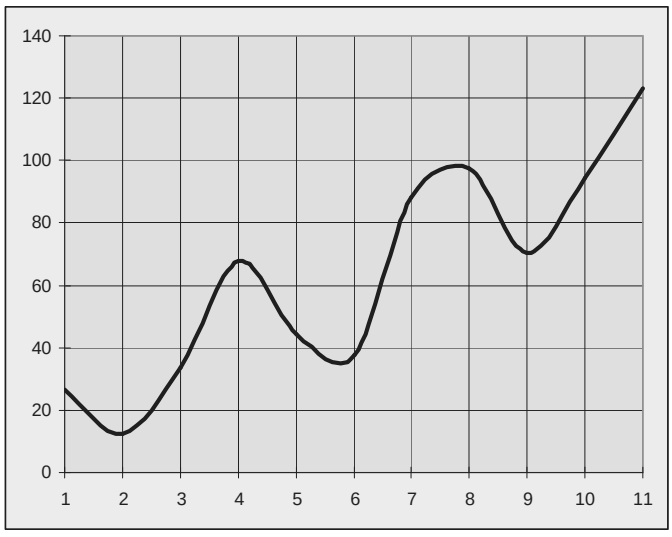
Варианты заданий к лабораторной работе № 2

№ задания	Степень	График
1	3	
2	4	

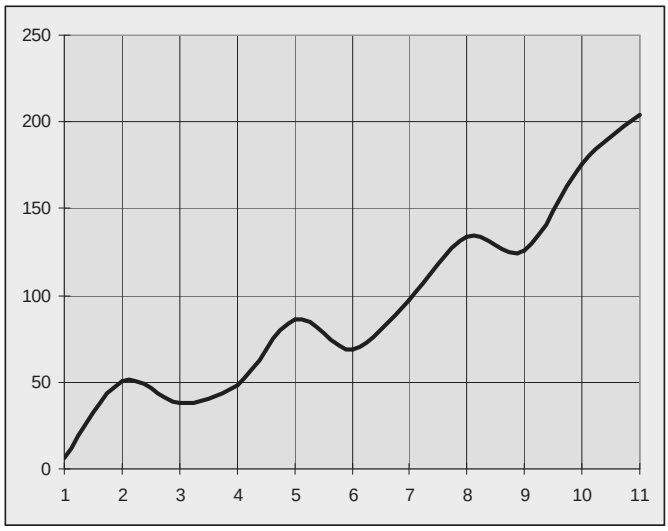
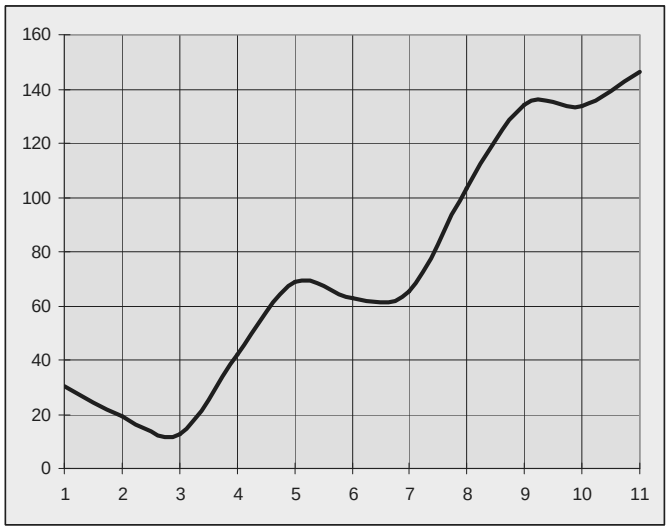
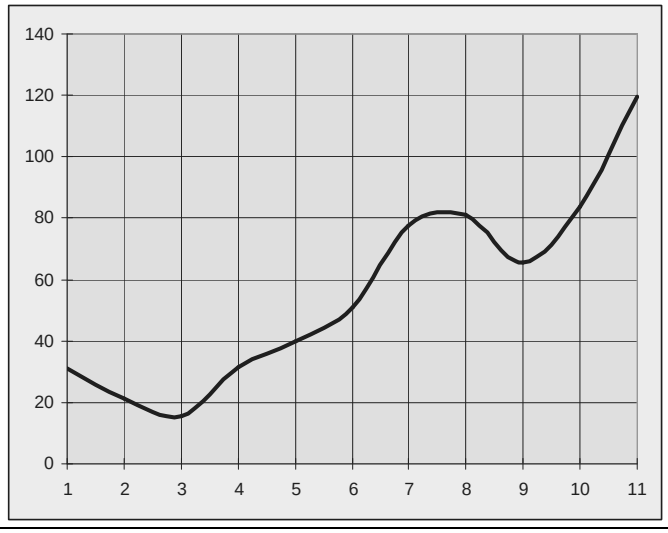
Продолжение табл.

№ задания	Степень	График
3	5	
4	6	
5	3	

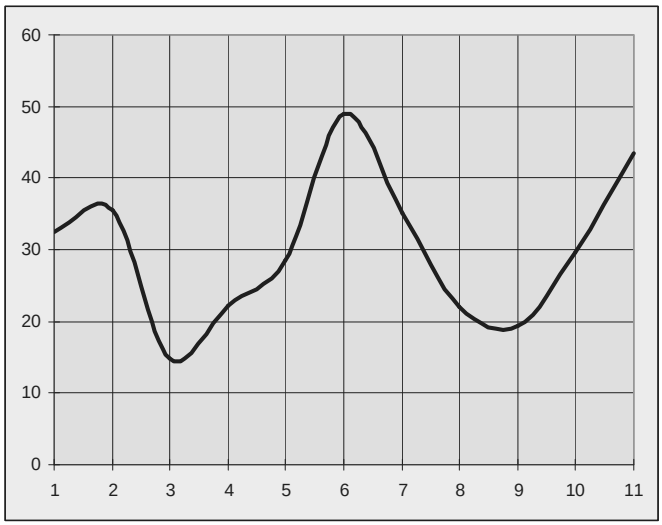
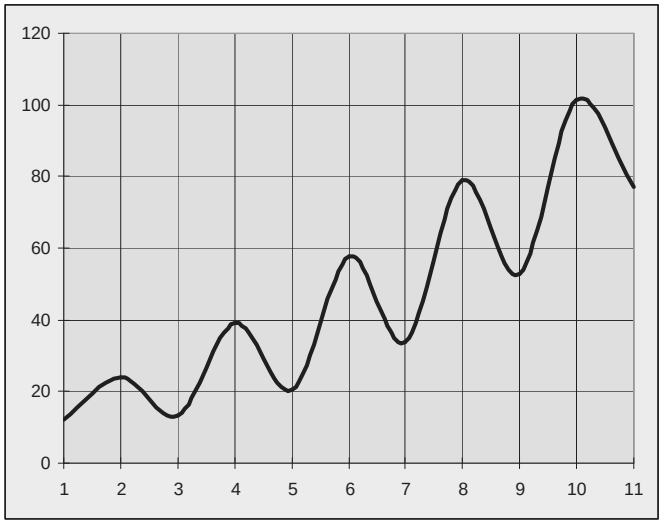
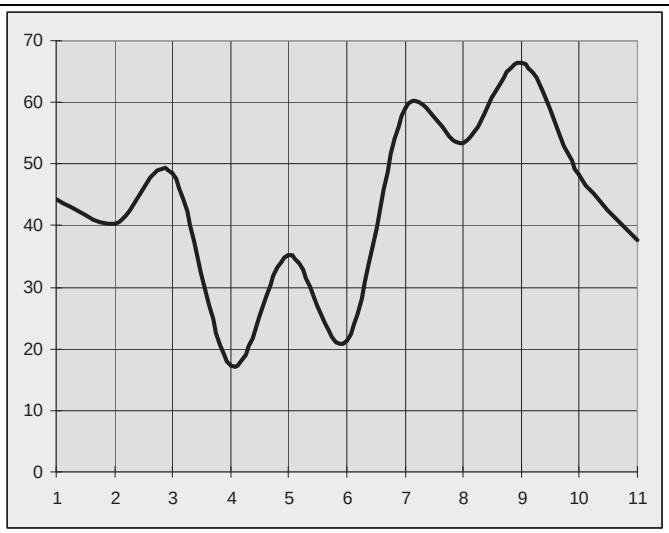
Продолжение табл.

№ задания	Степень	График
6	4	
7	5	
8	6	

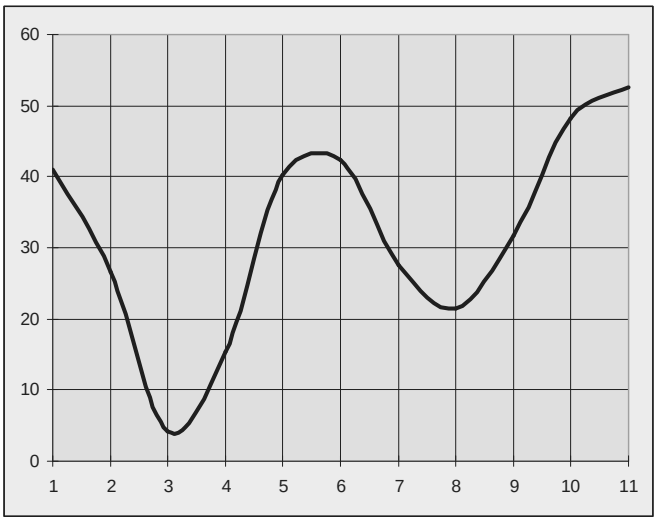
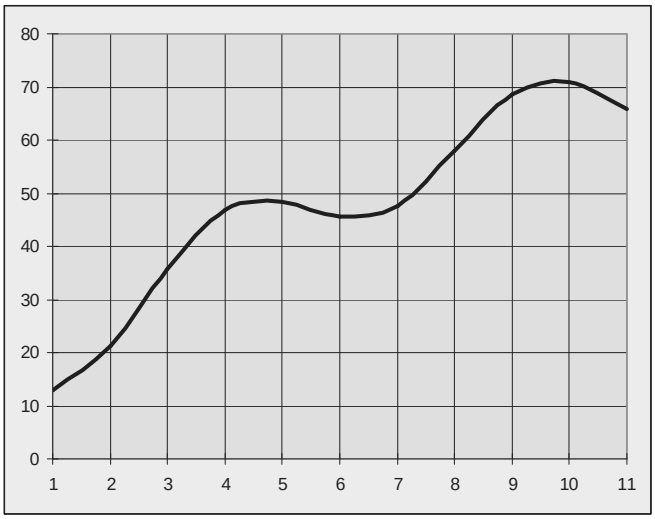
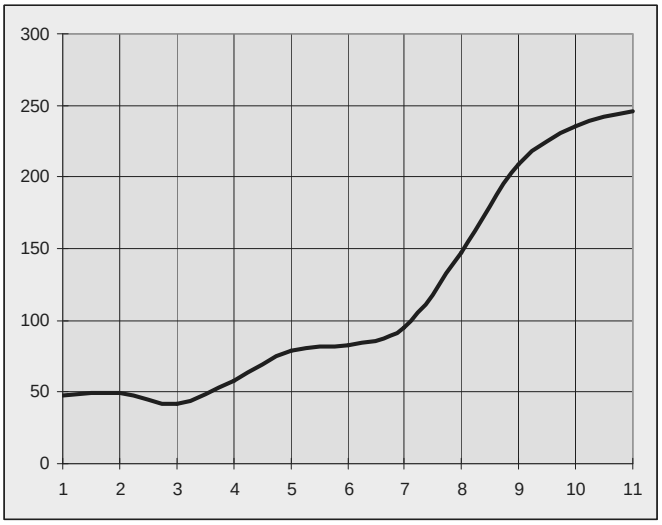
Продолжение табл.

№ задания	Степень	График
9	3	
10	4	
11	5	

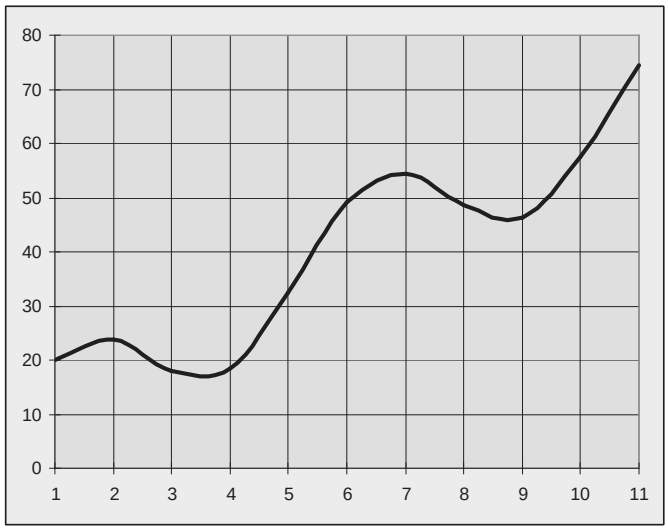
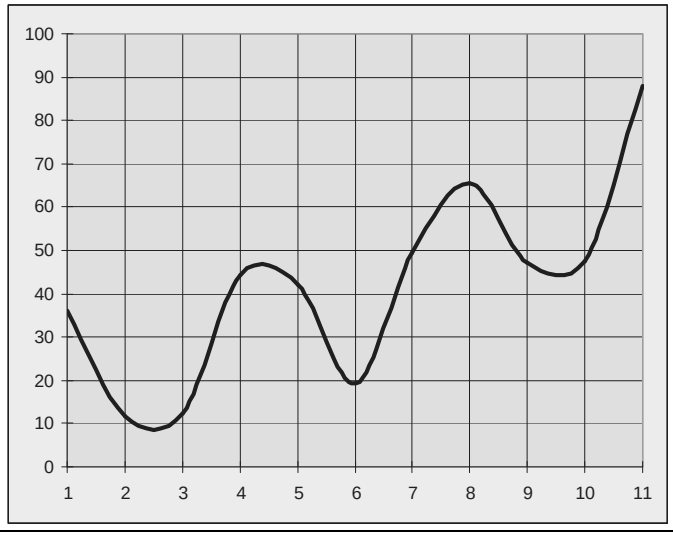
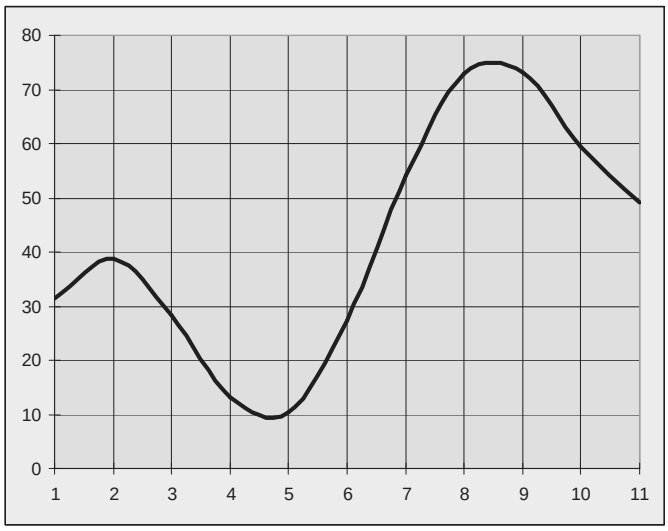
Продолжение табл.

№ задания	Степень	График
12	6	
13	3	
14	4	

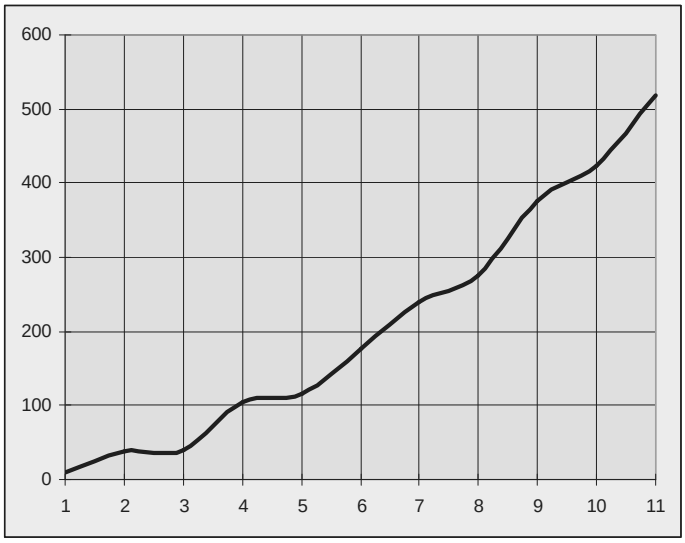
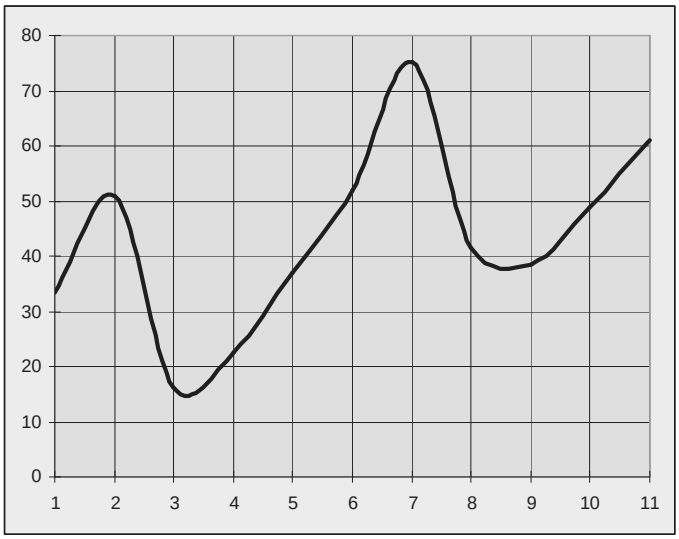
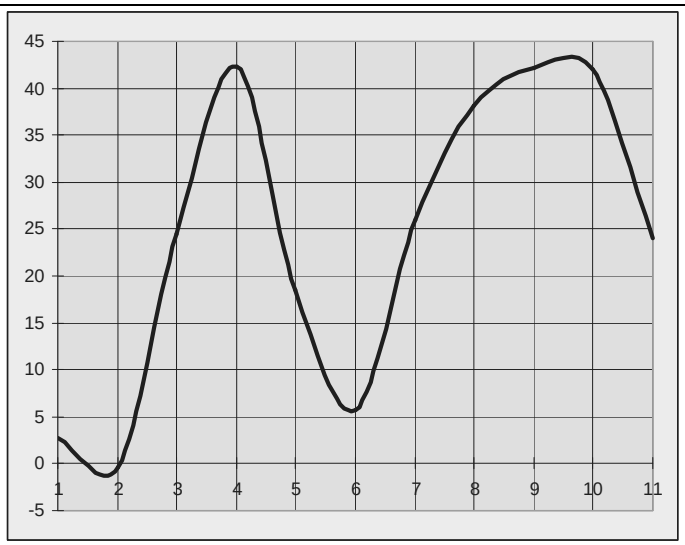
Продолжение табл.

№ задания	Степень	График
15	5	
16	6	
17	3	

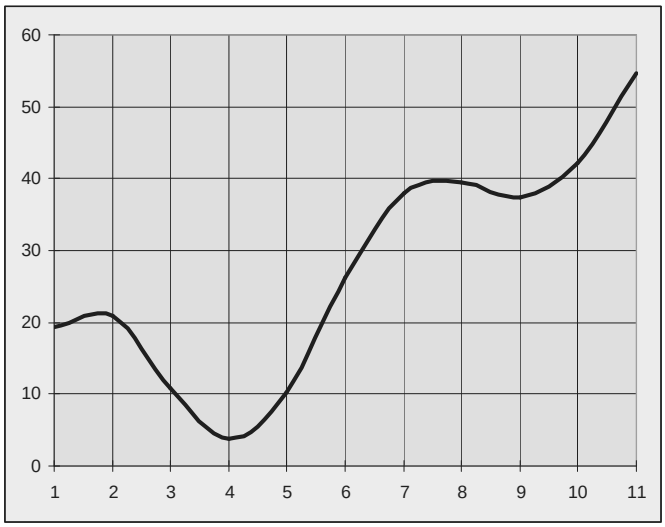
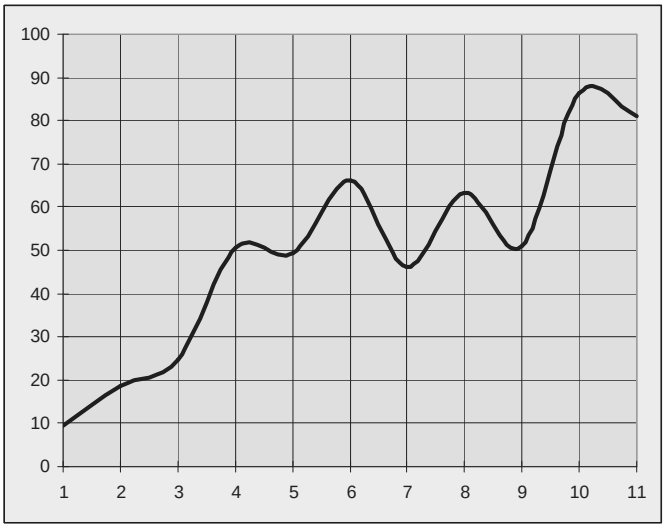
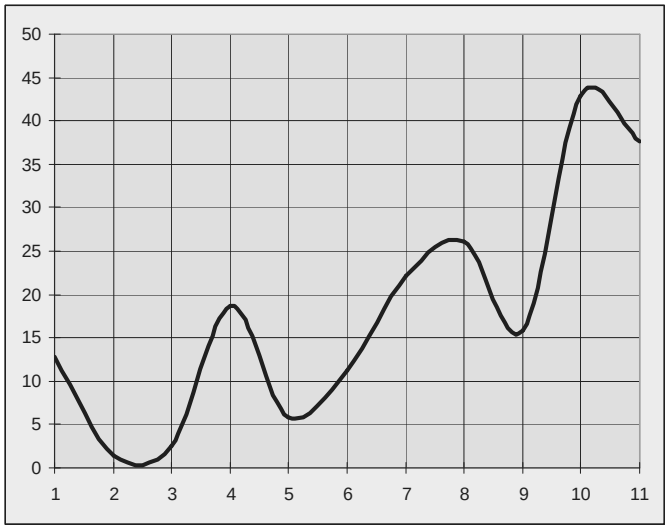
Продолжение табл.

№ задания	Степень	График
18	4	
19	5	
20	6	

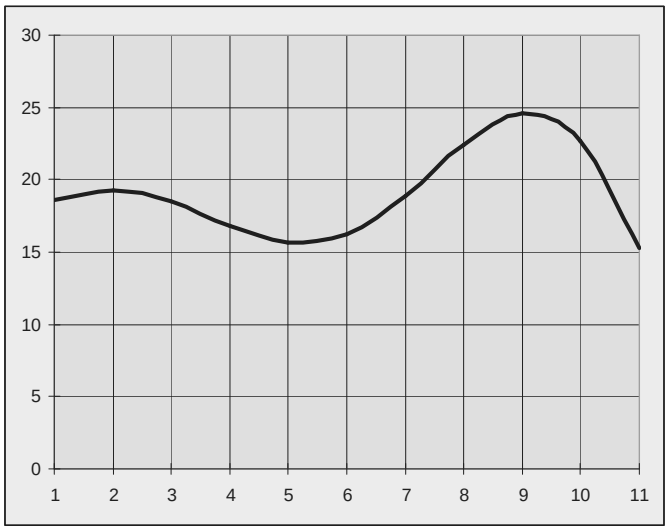
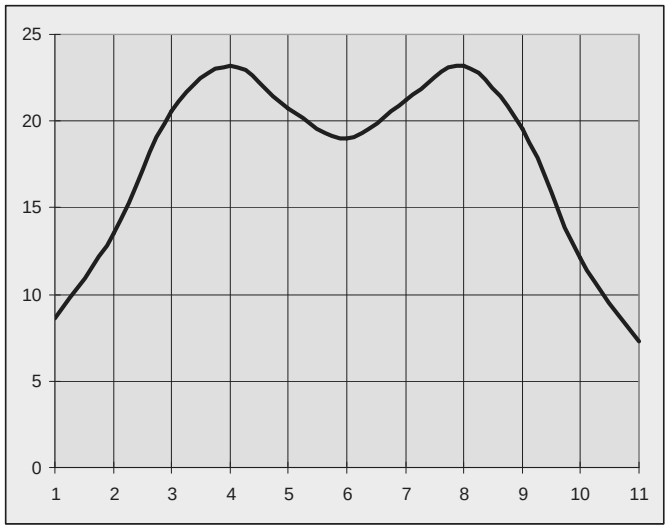
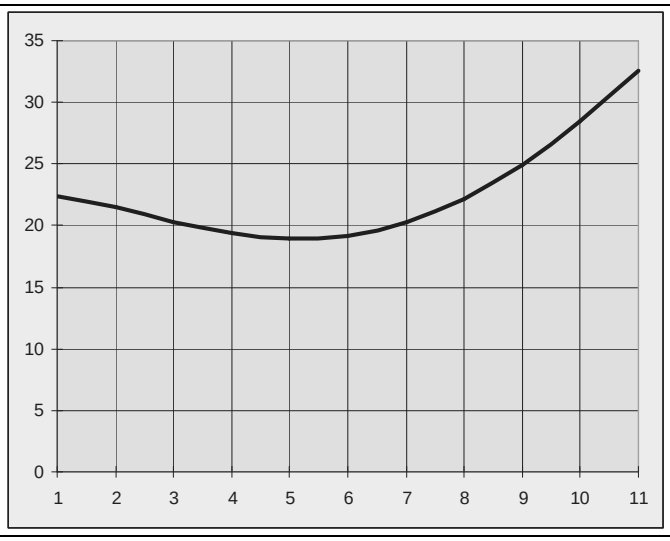
Продолжение табл.

№ задания	Степень	График
21	3	
22	4	
23	5	

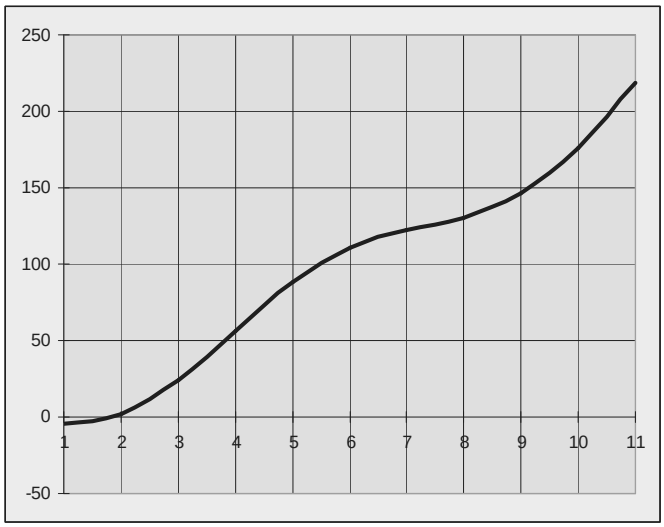
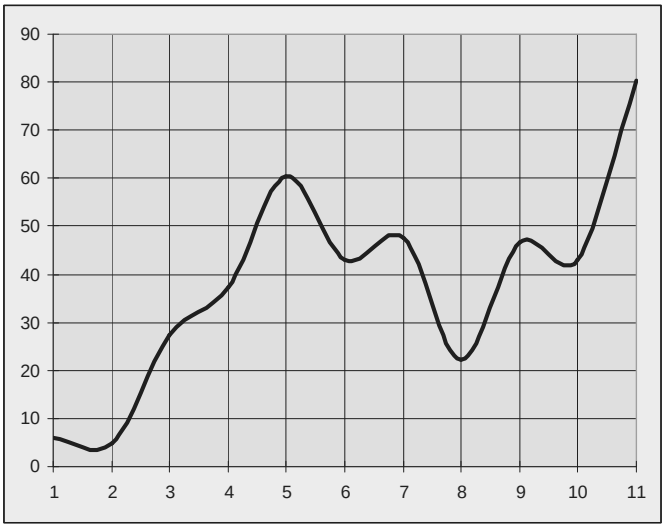
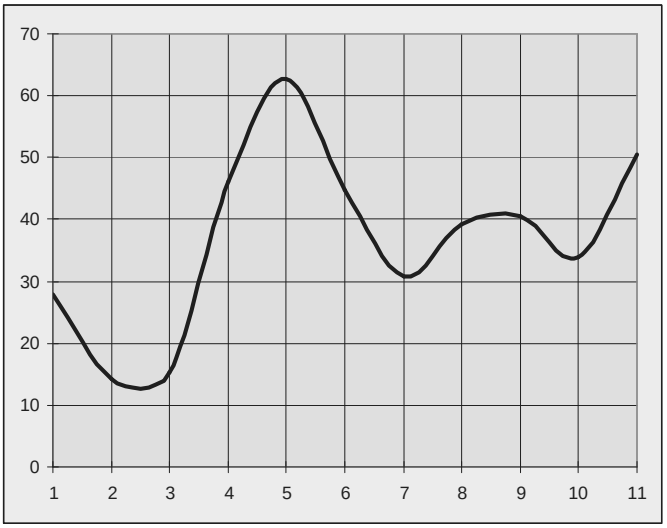
Продолжение табл.

№ задания	Степень	График
24	6	
25	3	
26	4	

Продолжение табл.

№ задания	Степень	График
27	5	
28	6	
29	3	

Окончание табл.

№ задания	Степень	График
30	4	
31	5	
32	6	

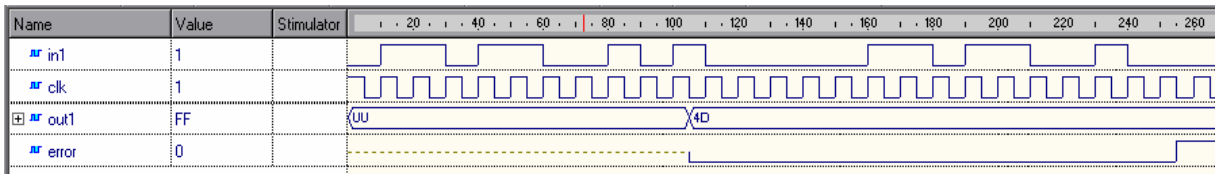
ЛАБОРАТОРНАЯ РАБОТА № 3

Проектирование последовательного порта

Цель работы: ознакомиться с возможностями, предоставляемыми простейшим последовательным портом. Получить практические навыки реализации и тестирования последовательного порта на языке VHDL в среде ActiveHDL.

1. Проектирование приемника последовательного порта.

1.1. Приемник последовательного порта выполняет прием последовательности битов, окаймленной стартовым и стоповым битом, и преобразует ее в байтовое значение. На диаграмме, приведенной ниже, отображен прием одного правильного байта со значением 4Dh и одного неправильного, в котором стоповый бит установлен в 0. По окончании приема ошибочного байта устанавливается сигнал ошибки.



1.2. Простейший приемник последовательного порта содержит:

1.2.1. Линию, по которой поступают входные данные в последовательном представлении. Началом посылки служит стартовый бит. Об окончании посылки сигнализирует стартовый бит. Примем значения стартового и стопового битов равными 1 (in1).

1.2.2. Вход тактирования. Необходим для соблюдения заданной скорости приема (clk). Примем частоту тактирования вдвое большей частоты передачи битов.

1.2.3. Порт, на который выдается полученное по последовательному интерфейсу значение (out1).

1.2.4. Вывод, сигнализирующий об ошибке. Выставляется в 1, если не обнаружен стоповый бит (error).

1.3. Создать проект, который будет содержать:

1.3.1. Модель приемника последовательного порта.

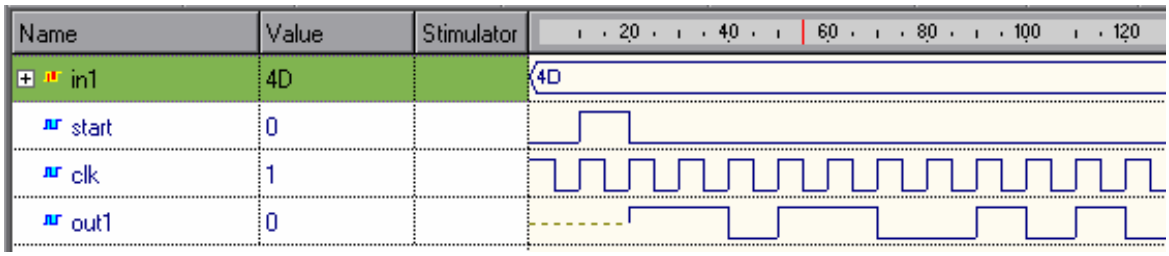
1.3.2. Тестовый модуль для приемника последовательного порта.

1.4. Добиться успешного выполнения тестового модуля.

1.5. Выполнить размещение полученного модуля на кристалле.

2. Проектирование передатчика последовательного порта

2.1. Передатчик последовательного порта представляет собой сдвиговый регистр. При записи какого-либо значения в сдвиговый регистр начинается его побитовый сдвиг и, соответственно, передача с определенной скоростью. При передаче передаваемое значение должно окаймляться стартовым и стоповым битами. Ниже приведена диаграмма для передачи байта со значением 4Dh.



2.2. Простейший передатчик последовательного порта содержит:

2.2.1. Порт, на который выставляется байт для передачи (in1).

2.2.2. Сигнал, по которому начинается передача байта (start).

2.2.3. Вход тактирования. Необходим для соблюдения заданной скорости передачи (clk). Примем частоту тактирования вдвое большей частоты передачи битов.

2.2.4. Линию, по которой передаются данные в последовательном представлении. Началом посылки служит стартовый бит. Об окончании посылки сигнализирует стартовый бит. Примем значения стартового и стопового битов равными 1 (out1).

2.3. Создать проект, который будет содержать:

2.3.1. Модель передатчика последовательного порта.

2.3.2. Тестовый модуль для передатчика последовательного порта.

2.4. Добиться успешного выполнения тестового модуля.

2.5. Выполнить размещение полученного модуля на кристалле.

3. Создание тестового модуля

Для создания тестового модуля, необходимого для расширенного тестирования, требуется:

3.1. Вызвать пункт меню «Tools» ⇒ «Generate Test Bench» («Инструменты» ⇒ «Генерация тестового модуля»).

3.2. Выбрать Entity (Структура) и Architecture (Архитектура), для которой будет создан тестовый модуль.

3.3. Два раза нажать «Далее>>». Ввести имя тестового модуля. Нажать «Далее>>» и «Готово».

3.4. Открыть созданный VHDL-файл с тестовым модулем.

3.5. Добавить в файл тестового модуля код для генерации тестовых последовательностей.

3.6. Откомпилировать проект.

3.7. Выбрать в качестве модуля верхнего уровня (Top Level) пару Entity-Architecture для созданного тестового модуля.

3.8. Создать новый Waveform и добавить в него все сигналы.

3.9. Запустить симуляцию проекта и просмотреть результат в окне Waveform.

4. Дополнительная информация:

4.1. При обработке сигналов, поступающих с тактового генератора, удобно использовать функции:

4.1.1. rising_edge(<имя_сигнала>) – возвращает истину, если в данный момент имеется переход сигнала из нуля в единицу.

4.1.2. falling_edge(<имя_сигнала>) – возвращает истину, если в данный момент имеется переход сигнала из единицы в ноль.

4.2. Условные циклы в языке VHDL организуются при помощи конструкций **while** … **loop** и **for** … **in** … **loop**.

4.3. При вычислении бита контроля по четности проще всего использовать операцию «Сложение по модулю 2» (xor) для всех битов обрабатываемого байта.

Порядок выполнения работы:

1. Ознакомиться с требованиями методических указаний по выполнению лабораторной работы.
2. Изучить требования, предъявляемые к простейшему последовательному порту.
3. Изучить возможности языка VHDL, применяемые для реализации последовательного порта.
4. Реализовать в соответствии с индивидуальным заданием модуль устройства.
5. Подключить соответствующий тестовый модуль и выполнить тестирование.
6. Выполнить размещение устройства на кристалле.
7. Оформить отчет.

Структура отчета:

1. Название и цель работы.
2. Индивидуальное задание на лабораторную работу.
3. Основные характеристики реализуемого устройства.
4. Исходный VHDL-текст реализованного устройства.
5. Копия консоли ActiveHDL с результатом тестирования.
6. Waveform правильной работы устройства с тестовым модулем.
7. Копия экрана размещения устройства на кристалле.
8. Выводы по работе с учетом индивидуального задания.

Базовый исходный код для реализации приемника последовательного порта

```
-- реализация простейшего приемника последовательного порта

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

-- порты приемника
-- in1 – вход, на который приходят битовые посылки
-- out1 – результат, полученный при приеме битов
-- error – сигнал ошибки
-- clk – вход для тактирования приемника
entity serial_rx is
        port(
        in1     : in std_logic;
        out1    : out std_logic_vector (7 downto 0);
        error   : out std_logic;
        clk          : in std_logic);
end serial_rx;

architecture behaviour of serial_rx is

begin

-- Здесь должна быть реализация приемника последовательного порта

end behaviour;
```

Тестовый модуль для приемника последовательного порта

```
library ieee;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_1164.all;

entity serial_rx_tb is
end serial_rx_tb;

architecture TB_ARCHITECTURE of serial_rx_tb is
     component serial_rx
     port(
          in1 : in std_logic;
          out1 : out std_logic_vector(7 downto 0);
          error : out std_logic;
          clk : in std_logic );
     end component;

     signal in1 : std_logic;
     signal clk : std_logic;
     signal out1 : std_logic_vector(7 downto 0);
     signal error : std_logic;

begin

     UUT : serial_rx
          port map (
               in1 => in1,
               out1 => out1,
               error => error,
               clk => clk
          );

clk_process:
process
begin

     -- бесконечный цикл, в котором выдаются синхроимпульсы на вход
CLK
     infinite_loop: loop
          clk <= '1';
          wait for 5ns;
          clk <= '0';
          wait for 5ns;
     end loop infinite_loop;
end process;

data_process:
process
begin
     in1 <= '0';
     wait for 10ns;
     -- стартовый бит
```

```
in1 <= '1';
wait for 10 ns;
-- биты данных
in1 <= '1';
wait for 10 ns;
in1 <= '0';
wait for 10 ns;
in1 <= '1';
wait for 10 ns;
in1 <= '1';
wait for 10 ns;
in1 <= '0';
wait for 10 ns;
in1 <= '0';
wait for 10 ns;
in1 <= '1';
wait for 10 ns;
in1 <= '0';
wait for 10 ns;
-- стоповый бит
in1 <= '1';
wait for 10 ns;

assert out1 = "01001101" and error = '0'
report "Ошибка приема первого правильного байта"
severity failure;

in1 <= '0';
wait for 50ns;
-- стартовый бит
in1 <= '1';
wait for 10 ns;
-- биты данных
in1 <= '1';
wait for 10 ns;
in1 <= '0';
wait for 10 ns;
in1 <= '1';
wait for 10 ns;
in1 <= '1';
wait for 10 ns;
in1 <= '0';
wait for 10 ns;
in1 <= '0';
wait for 10 ns;
in1 <= '1';
wait for 10 ns;
in1 <= '0';
wait for 10 ns;
-- ошибочный стоповый бит
```

```
      in1 <= '0';
      wait for 10 ns;

      assert error = '1'
      report "Не обнаружена ошибка во втором байте"
      severity failure;

      in1 <= '0';
      wait for 50ns;
      -- стартовый бит
      in1 <= '1';
      wait for 10 ns;
      -- биты данных
      in1 <= '0';
      wait for 10 ns;
      in1 <= '1';
      wait for 10 ns;
      in1 <= '0';
      wait for 10 ns;
      in1 <= '0';
      wait for 10 ns;
      in1 <= '1';
      wait for 10 ns;
      in1 <= '1';
      wait for 10 ns;
      in1 <= '0';
      wait for 10 ns;
      in1 <= '1';
      wait for 10 ns;
      -- Стоповый бит
      in1 <= '1';
      wait for 10 ns;

      assert out1 = "10110010" and error = '0'
      report "Ошибка приема третьего правильного байта"
      severity failure;

      report "Поздравлем! Приемник последовательного порта успешно про-
шел тестирование!";

      wait;
end process;


end TB_ARCHITECTURE;

configuration TESTBENCH_FOR_serial_rx of serial_rx_tb is
      for TB_ARCHITECTURE
            for UUT : serial_rx
                  use entity work.serial_rx(behaviour);
            end for;
      end for;
end TESTBENCH_FOR_serial_rx;
```

Базовый исходный код для реализации передатчика последовательного порта

```
-- реализация простейшего передатчика последовательного порта

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

-- порты передатчика
-- in1 – вход, на котором устанавливается передаваемый байт
-- out1 – выход для передаваемой битовой последовательности
-- start – сигнал, запускающий передачу данных
-- clk – вход для тактирования приемника
entity serial_tx is
    port(
        in1       : in std_logic_vector(7 downto 0);
        out1      : out std_logic;
        start     : in std_logic;
        clk       : in std_logic);
end serial_tx;

architecture behaviour of serial_tx is

-- Здесь должна быть реализация передатчика последовательного порта

end behaviour;
```

Тестовый модуль для передатчика последовательного порта

```
library ieee;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_1164.all;

entity serial_tx_tb is
end serial_tx_tb;

architecture TB_ARCHITECTURE of serial_tx_tb is
     component serial_tx
     port(
          in1 : in std_logic_vector(7 downto 0);
          out1 : out std_logic;
          start : in std_logic;
          clk : in std_logic );
     end component;

     signal in1 : std_logic_vector(7 downto 0);
     signal start : std_logic;
     signal clk : std_logic;
     signal out1 : std_logic;

begin

     UUT : serial_tx
          port map (
               in1 => in1,
               out1 => out1,
               start => start,
               clk => clk
          );

clk_process:
process
begin

    -- бесконечный цикл, в котором выдаются синхроимпульсы на вход CLK
    infinite_loop: loop
        clk <= '1';
        wait for 5ns;
        clk <= '0';
        wait for 5ns;
    end loop infinite_loop;
end process;
data_process:
```

```
process
begin
      start <= '0';
      -- Устанавливаем значение на передачу
      in1 <= "01001101";
      wait for 10ns;

      -- Запускаем передачу
      start <= '1';
      wait for 10ns;
      start <= '0';

      wait for 5ns;
      assert out1 = '1'
      report "Нет стартового бита!"
      severity failure;

      wait for 10ns;
      assert out1 = '1'
      report "Ошибка в 0-м бите!"
      severity failure;

      wait for 10ns;
      assert out1 = '0'
      report "Ошибка в 1-м бите!"
      severity failure;

      wait for 10ns;
      assert out1 = '1'
      report "Ошибка во 2-м бите!"
      severity failure;

      wait for 10ns;
      assert out1 = '1'
      report "Ошибка в 3-м бите!"
      severity failure;

      wait for 10ns;
      assert out1 = '0'
      report "Ошибка в 4-м бите!"
      severity failure;

      wait for 10ns;
      assert out1 = '0'
      report "Ошибка в 5-м бите!"
      severity failure;
      wait for 10ns;
```

```
      assert out1 = '1'
      report "Ошибка в 6-м бите!"
      severity failure;

      wait for 10ns;
      assert out1 = '0'
      report "Ошибка в 7-м бите!"
      severity failure;

      wait for 10ns;
      assert out1 = '1'
      report "Нет стопового бита!"
      severity failure;

    report "Поздравлем! Передатчик последовательного порта успешно прошел тестирование!";

    wait;
end process;

end TB_ARCHITECTURE;

configuration TESTBENCH_FOR_serial_tx of serial_tx_tb is
     for TB_ARCHITECTURE
          for UUT : serial_tx
               use entity work.serial_tx(behaviour);
          end for;
     end for;
end TESTBENCH_FOR_serial_tx;
```

Варианты заданий к лабораторной работе № 3

Вариант	Разрядность	Контроль	Стоповые биты	Инверсные значения битов
1	5	Четность	1	да
2	6	Четность	1	нет
3	7	Четность	1	да
4	8	Четность	1	нет
5	5	Нечетность	1	да
6	6	Нечетность	1	нет
7	7	Нечетность	1	да
8	8	Нечетность	1	нет
9	5	Четность	1,5	да
10	6	Четность	1,5	нет
11	7	Четность	1,5	да
12	8	Четность	1,5	нет
13	5	Нечетность	1,5	да
14	6	Нечетность	1,5	нет
15	7	Нечетность	1,5	да
16	8	Нечетность	1,5	нет
17	5	Четность	2	да
18	6	Четность	2	нет
19	7	Четность	2	да
20	8	Четность	2	нет
21	5	Нечетность	2	да
22	6	Нечетность	2	нет
23	7	Нечетность	2	да
24	8	Нечетность	2	нет
25	5	Четность	2,5	да
26	6	Четность	2,5	нет
27	7	Четность	2,5	да
28	8	Четность	2,5	нет
29	5	Нечетность	2,5	да
30	6	Нечетность	2,5	нет
31	7	Нечетность	2,5	да
32	8	Нечетность	2,5	нет

ЛАБОРАТОРНАЯ РАБОТА № 4

Проектирование АЛУ

Цель работы: ознакомиться с функциями, реализуемыми простейшим АЛУ. Получить практические навыки реализации и тестирования АЛУ на языке VHDL в среде ActiveHDL.

Простейшее АЛУ содержит:

- два входа для операндов;
- вход для указания выполняемой операции;
- выход для результата;
- выходы для флагов;
- блоки, выполняющие операции;
- блоки, устанавливающие значения флагов.

Простейшее АЛУ реализует:

- суммирование;
- вычитание;
- умножение;
- побитовое И;
- побитовое ИЛИ;
- побитовое исключающее ИЛИ;
- побитовое НЕ;
- циклический сдвиг влево, через перенос;
- циклический сдвиг вправо, через перенос;
- арифметический сдвиг влево, через перенос;
- арифметический сдвиг вправо, через перенос.

Простейшее АЛУ устанавливает значения флагов:

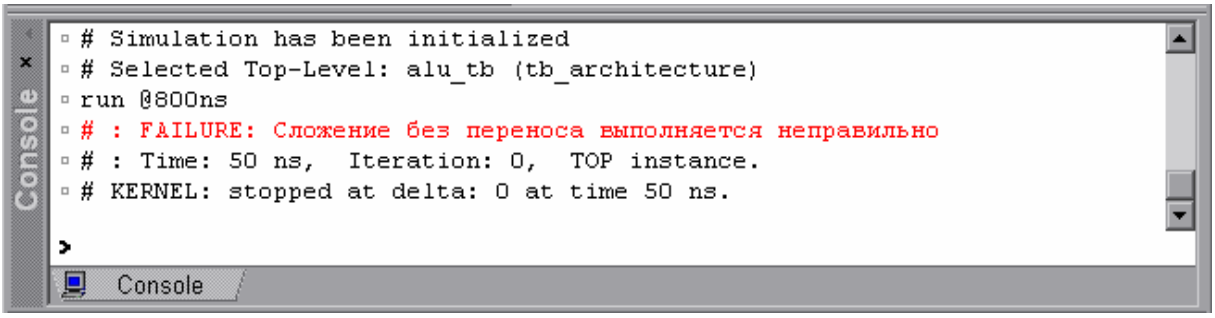
- Z – флаг нулевого результата;
- CY – флаг переноса/заема;
- OV – флаг переполнения;
- P – флаг четности.

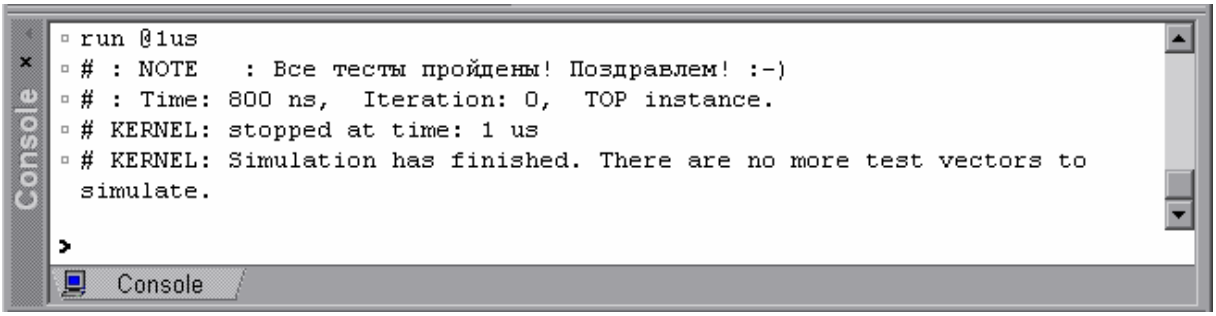
1. Добавить в новый проект файл с исходным VHDL-кодом и файл с тестовым модулем.

2. Реализовать в VHDL-коде АЛУ необходимые операции с учетом влияния результата на значения флагов (см. табл).

Операция / Затрагиваемые флаги	+	-	*	And	Or	Xor	Not	Rol	Ror	Shl	Shr
Z	□	□	□	□	□	□	□				
CY	□	□						□	□	□	□
OV			□								
P	□	□	□	□	□	□	□	□	□	□	□

3. Исключить из тестового модуля тесты для операций, которые АЛУ не должно реализовать.
4. Полностью выполнить тестовый модуль и получить сообщение о правильной работе АЛУ.
5. Выдаваемые тестовым модулем сообщения отображаются в консоли ActiveHDL.

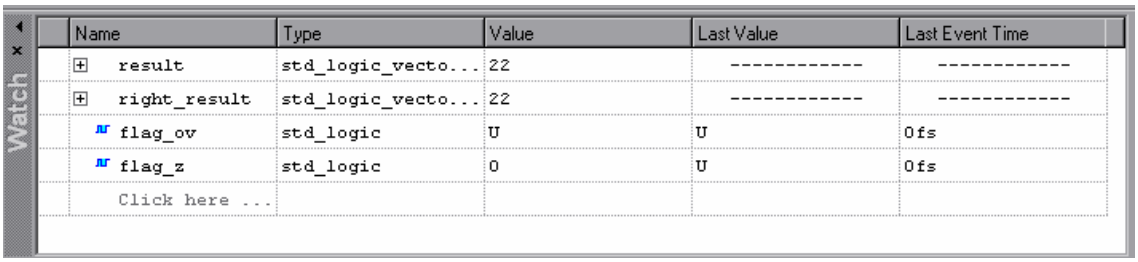




6. При моделировании АЛУ удобно применять встроенный в ActiveHDL отладчик, который позволяет:
   а) установить в любом месте кода контрольную точку, по достижении которой выполнение будет приостановлено. Горячая клавиша – F9;

б) выполнить код пошагово: F7 – с входом в выполняемые функции/процедуры; F8 – без входа в выполняемые функции/процедуры; F10 – с выходом из текущей функции/процедуры;

в) просмотреть значение любой переменной/сигнала. Вызов окна Watch – горячая клавиша Alt-4. Добавление переменной/сигнала в список просмотра – двойной щелчок по надписи «Click here…».



Порядок выполнения работы:

1. Ознакомиться с требованиями методических указаний по выполнению лабораторной работы.
2. Изучить требования, предъявляемые к простейшим АЛУ.
3. Изучить возможности языка VHDL, применяемые для реализации АЛУ.
4. Реализовать процедуры для установки флагов Z, CY, OV по результату выполнения операции.
5. Реализовать выполнение операций, заданных в индивидуальном задании.
6. Модифицировать тестовый модуль и выполнить тестирование полученного кода.
7. Оформить отчет.

Структура отчета:

1. Название и цель работы.
2. Индивидуальное задание на лабораторную работу.
3. Исходный VHDL-текст реализованного АЛУ.
4. Исходный код реализованного тестового модуля.
5. Копия текста из консоли ActiveHDL с результатом тестирования.
6. Вывод по работе с учетом индивидуального задания.

Исходный код простейшего АЛУ с несколькими реализованными функциями

```
-- реализация простейшего АЛУ

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

-- порты АЛУ
-- reset - сброс
-- op_code - код операции
-- in1 - первый операнд
-- in2 - второй операнд
-- out1 - результат
-- flag_z - флаг нулевого результата
-- flag_cy - флаг переноса
-- flag_ov - флаг переполнения
entity alu is
    port(
        reset   : in std_logic;
        op_code : in  std_logic_vector (3 downto 0);
        in1     : in  std_logic_vector (7 downto 0);
        in2     : in  std_logic_vector (7 downto 0);
        out1    : out std_logic_vector (7 downto 0);
        flag_z  : out std_logic;
        flag_cy : out std_logic;
        flag_ov : out std_logic);
end alu;

architecture behaviour of alu is

--Коды операций, которые выполняет АЛУ
constant  alu_op_add : std_logic_vector (3 downto 0)  := "0001";
constant  alu_op_sub : std_logic_vector (3 downto 0)  := "0010";
constant  alu_op_mul : std_logic_vector (3 downto 0)  := "0011";
constant  alu_op_and : std_logic_vector (3 downto 0)  := "0100";
constant  alu_op_or  : std_logic_vector (3 downto 0)  := "0101";
constant  alu_op_xor : std_logic_vector (3 downto 0)  := "0110";
constant  alu_op_ror : std_logic_vector (3 downto 0)  := "0111";
constant  alu_op_rol : std_logic_vector (3 downto 0)  := "1000";
constant  alu_op_not : std_logic_vector (3 downto 0)  := "1001";

-- Процедура, вычисляющая по значению результата значение флага OV
procedure calc_overflow_flag (
    result          : in  std_logic_vector (15 downto 0);
    overflow_flag   : out std_logic ) is

    begin
```

```
            -- Проверяются старшие 8 бит результата
        if( result( 15 downto 8 ) /= "00000000" ) then
                -- Если они не нулевые, то установить флаг переполнения
            overflow_flag := '1';
        else
                -- Если они нулевые, то сбросить флаг переполнения
            overflow_flag := '0';
        end if;

    end calc_overflow_flag;


begin
    process(
    reset,
    op_code,
    in1,
    in2 )

        -- Временные переменные для хранения значений внутри процесса
        variable tmp_in1, tmp_in2 : std_logic_vector (15 downto 0);
        variable res : std_logic_vector (15 downto 0);
        variable res_z, res_cy, res_ov : std_logic;
    begin

    if reset = '0' then

        -- Обнуляем внутренние переменные и записываем в них
        -- входные значения
        -- Внутренние переменные имеют вдвое большую разрядность, чем
        -- входные для того, чтобы иметь возможность устанавливать
        -- флаги
        tmp_in1 := "0000000000000000";
        tmp_in2 := "0000000000000000";
        tmp_in1( 7 downto 0 ) := in1;
        tmp_in2( 7 downto 0 ) := in2;

        -- Определяем, какой код операции установлен
        case op_code is
            when alu_op_add =>
            -- Установлен код операции сложения
            -- Выполняем сложение
            res := tmp_in1 + tmp_in2;

            when alu_op_mul =>
            -- Умножение
```

```
                -- Максимальное значение результата умножения
                -- имеет разрядность в 2 раза больше, чем операнды,
                -- поэтому в качестве множителей берутся не 16-
разрядные
                -- tmp_in1 и tmp_in1, а 8 разрядные in1 и in2
            res := in1 * in2;
            -- Устанавливаем флаг OV
            calc_overflow_flag( res, res_ov );

            when alu_op_rol =>
            -- Циклический сдвиг влево
            res( 0 ) := tmp_in1( 7 );
            res( 7 downto 1 ) := tmp_in1( 6 downto 0 );

            when others =>
            -- Неизвестный код операции
            res := "ZZZZZZZZZZZZZZZZ";

        end case;

    end if;

    -- Выдаем в порты 8 младших бит результата и
    -- значения флагов
    out1 <= res( 7 downto 0 );
    flag_z <= res_z;
    flag_cy <= res_cy;
    flag_ov <= res_ov;

end process;

end behaviour;
```

Тестовый модуль для АЛУ

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

    -- Add your library and packages declaration here ...

entity alu_tb is
end alu_tb;

architecture TB_ARCHITECTURE of alu_tb is
    -- Component declaration of the tested unit
--Коды операций, которые выполняет АЛУ
constant  alu_op_add : std_logic_vector (3 downto 0)  := "0001";
constant  alu_op_sub : std_logic_vector (3 downto 0)  := "0010";
constant  alu_op_mul : std_logic_vector (3 downto 0)  := "0011";
constant  alu_op_and : std_logic_vector (3 downto 0)  := "0100";
constant  alu_op_or  : std_logic_vector (3 downto 0)  := "0101";
constant  alu_op_xor : std_logic_vector (3 downto 0)  := "0110";
constant  alu_op_ror : std_logic_vector (3 downto 0)  := "0111";
constant  alu_op_rol : std_logic_vector (3 downto 0)  := "1000";
constant  alu_op_not : std_logic_vector (3 downto 0)  := "1001";

    component alu
    port(
        reset : in std_logic;
        op_code : in std_logic_vector(3 downto 0);
        in1 : in std_logic_vector(7 downto 0);
        in2 : in std_logic_vector(7 downto 0);
        out1 : out std_logic_vector(7 downto 0);
        flag_z : out std_logic;
        flag_cy : out std_logic;
        flag_ov : out std_logic );
    end component;

    -- Stimulus signals - signals mapped to the input and inout
ports of tested entity
    signal reset : std_logic;
    signal op_code : std_logic_vector(3 downto 0);
    signal in1 : std_logic_vector(7 downto 0);
    signal in2 : std_logic_vector(7 downto 0);
    -- Observed signals - signals mapped to the output ports of
tested entity
    signal out1 : std_logic_vector(7 downto 0);
    signal flag_z : std_logic;
    signal flag_cy : std_logic;
    signal flag_ov : std_logic;

    -- Add your code here ...
begin
```

```
    -- Unit Under Test port map
    UUT : alu
        port map (
            reset => reset,
            op_code => op_code,
            in1 => in1,
            in2 => in2,
            out1 => out1,
            flag_z => flag_z,
            flag_cy => flag_cy,
            flag_ov => flag_ov
        );

    -- Add your stimulus here ...
-- Процесс, генерирующий тестовые последовательности
test_data_generator:
process

variable result : std_logic_vector (7 downto 0);
variable right_result : std_logic_vector (7 downto 0);

begin

    -- Подаем Reset на АЛУ -- пока не выполняются никакие действия
    reset <= '1';

    -- Тестирование сложения без переноса
    -- Задаем код операции
    op_code <= alu_op_add;
    -- Задаем первый операнд
    in1 <= "00001100";
    -- Задаем второй операнд
    in2 <= "00010110";
    -- Сбрасываем Reset, тем самым запуская работу АЛУ
    reset <= '0';
    -- Пауза
    wait for 50ns;

    -- Записываем в одну переменную полученный результат,
    -- а в другую – правильный результат
    result := out1;
    right_result := "00100010";

    -- Содержимое блока assert будет выполнено только тогда,
    -- когда нарушается указанное условие, т.е. если полученный
    -- результат равен правильному и флаг CY также установлен
    -- правильно, блок assert выполнен не будет
    assert ( result = right_result ) and ( flag_cy = '0' )
    -- Выводим сообщение о том, что АЛУ функционирует неправильно
    report "Сложение без переноса выполняется неправильно"
```

```
-- Установлен самый жесткий тип проверки assert -- failure
-- В случае если условие неистинно, выполнение моделирования
-- будет прервано
severity failure;

-- Снова останавливаем работу АЛУ
reset <= '1';

-- Тестирование сложения с переносом
op_code <= alu_op_add;
in1 <= "10001100";
in2 <= "10010110";
reset <= '0';
wait for 50ns;

result := out1;
right_result := "00100010";
assert ( result = right_result ) and ( flag_cy = '1' )
report "Сложение с переносом выполняется неправильно"
severity failure;

reset <= '1';

-- Тестирование вычитания без установки флага Z
op_code <= alu_op_sub;
in1 <= "00010100";
in2 <= "00001100";
reset <= '0';
wait for 50ns;

result := out1;
right_result := "00001000";
assert ( result = right_result ) and ( flag_z = '0' )
report "Вычитание без установки флага Z выполняется неправильно"
severity failure;

reset <= '1';

-- Тестирование вычитания с установкой флага Z
op_code <= alu_op_sub;
in1 <= "00010100";
in2 <= "00010100";
reset <= '0';
wait for 50ns;

result := out1;
right_result := "00000000";
assert ( result = right_result ) and ( flag_z = '1' )
report "Вычитание с установкой флага Z выполняется неправильно"
severity failure;
```

```
reset <= '1';

-- Тестирование умножения без переполнения
op_code <= alu_op_mul;
in1 <= "00010100";
in2 <= "00001100";
reset <= '0';
wait for 50ns;

result := out1;
right_result := "11110000";
assert ( result = right_result ) and ( flag_ov = '0' )
report "Умножение без переполнения выполняется неправильно"
severity failure;

reset <= '1';

-- Тестирование умножения С переполнением
op_code <= alu_op_mul;
in1 <= "00110100";
in2 <= "00001100";
reset <= '0';
wait for 50ns;

result := out1;
right_result := "01110000";
assert ( result = right_result ) and ( flag_ov = '1' )
report "Умножение с переполнением выполняется неправильно"
severity failure;

reset <= '1';

-- Тестирование побитового И без установки флага Z
op_code <= alu_op_and;
in1 <= "00110100";
in2 <= "00001100";
reset <= '0';
wait for 50ns;

result := out1;
right_result := "00000100";
assert ( result = right_result ) and ( flag_z = '0' )
report "Побитовое И без установки флага Z выполняется неправильно"
severity failure;

reset <= '1';

-- Тестирование побитового И с установкой флага Z
op_code <= alu_op_and;
in1 <= "00110100";
in2 <= "00001000";
```

```
reset <= '0';
wait for 50ns;

result := out1;
right_result := "00000000";
assert ( result = right_result ) and ( flag_z = '1' )
report "Побитовое И с установкой флага Z выполняется неправильно"
severity failure;

reset <= '1';


-- Тестирование побитового ИЛИ без установки флага Z
op_code <= alu_op_or;
in1 <= "00110100";
in2 <= "00001100";
reset <= '0';
wait for 50ns;

result := out1;
right_result := "00111100";
assert ( result = right_result ) and ( flag_z = '0' )
report "Побитовое ИЛИ без установки флага Z выполняется неправильно"
severity failure;

reset <= '1';

-- Тестирование побитового ИЛИ с установкой флага Z
op_code <= alu_op_or;
in1 <= "00000000";
in2 <= "00000000";
reset <= '0';
wait for 50ns;

result := out1;
right_result := "00000000";
assert ( result = right_result ) and ( flag_z = '1' )
report "Побитовое ИЛИ с установкой флага Z выполняется неправильно"
severity failure;

reset <= '1';

    -- Тестирование побитового исключающего ИЛИ без установки флага Z
op_code <= alu_op_xor;
in1 <= "00110100";
in2 <= "00001100";
reset <= '0';
wait for 50ns;

result := out1;
right_result := "00111000";
assert ( result = right_result ) and ( flag_z = '0' )
```

```
    report "Побитовое исключающее ИЛИ без установки флага Z выполня-
ется неправильно"
    severity failure;

    reset <= '1';

    -- Тестирование побитового исключающего ИЛИ с установкой флага Z
    op_code <= alu_op_xor;
    in1 <= "00110100";
    in2 <= "00110100";
    reset <= '0';
    wait for 50ns;

    result := out1;
    right_result := "00000000";
    assert ( result = right_result ) and ( flag_z = '1' )
    report "Побитовое исключающее ИЛИ с установкой флага Z выполняется
неправильно"
    severity failure;

    reset <= '1';

    -- Тестирование побитового НЕ без установки флага Z
    op_code <= alu_op_not;
    -- Операция имеет только один операнд. Значение in2
    -- несущественно
    in1 <= "00110100";
    reset <= '0';
    wait for 50ns;

    result := out1;
    right_result := "11001011";
    assert ( result = right_result ) and ( flag_z = '0' )
    report "Побитовое НЕ без установки флага Z выполняется неправильно"
    severity failure;

    reset <= '1';

    -- Тестирование побитового НЕ с установкой флага Z
    op_code <= alu_op_not;
    in1 <= "11111111";
    reset <= '0';
    wait for 50ns;

    result := out1;
    right_result := "00000000";
    assert ( result = right_result ) and ( flag_z = '1' )
    report "Побитовое НЕ с установкой флага Z выполняется неправильно"
    severity failure;

    reset <= '1';
```

```
    -- Тестирование циклического сдвига влево
    op_code <= alu_op_rol;
    in1 <= "10110100";
    reset <= '0';
    wait for 50ns;

    result := out1;
    right_result := "01101001";
    assert ( result = right_result )
    report "Циклический сдвиг влево выполняется неправильно"
    severity failure;

    reset <= '1';

    -- Тестирование циклического сдвига вправо
    op_code <= alu_op_ror;
    in1 <= "00110101";
    reset <= '0';
    wait for 50ns;

    result := out1;
    right_result := "10011010";
    assert ( result = right_result )
    report "Циклический сдвиг вправо выполняется неправильно"
    severity failure;

    reset <= '1';

    report "Все тесты пройдены! Поздравлем! :-)";
    wait;

end process;

end TB_ARCHITECTURE;

configuration TESTBENCH_FOR_alu of alu_tb is
    for TB_ARCHITECTURE
        for UUT : alu
            use entity work.alu(behaviour);
        end for;
    end for;
end TESTBENCH_FOR_alu;
```

Варианты заданий к лабораторной работе № 4

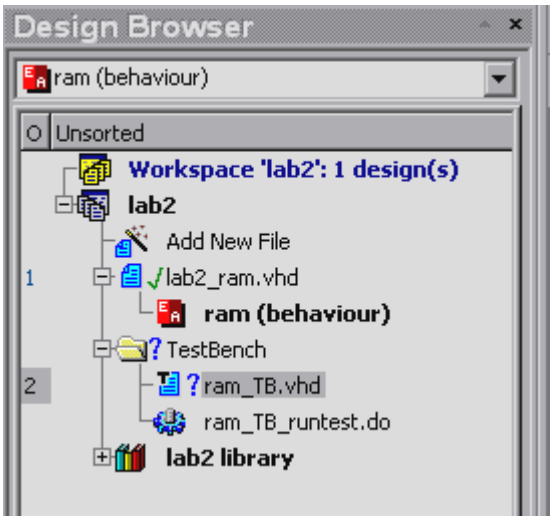
Вариант	Задание 1	Задание 2	Задание 3
1	+	and	ror
2	–	and	ror
3	*	and	ror
4	+	or	ror
5	-	or	ror
6	*	or	ror
7	+	xor	ror
8	–	xor	ror
9	*	xor	rol
10	+	not	rol
11	–	not	rol
12	*	not	rol
13	+	and	rol
14	–	and	rol
15	*	and	rol
16	+	or	rol
17	–	or	shr
18	*	or	shr
19	+	xor	shr
20	–	xor	shr
21	*	xor	shr
22	+	not	shr
23	–	not	shr
24	*	not	shr
25	+	and	shl
26	–	and	shl
27	*	and	shl
28	+	or	shl
29	–	or	shl
30	*	or	shl
31	+	xor	shl
32	–	xor	shl

ЛАБОРАТОРНАЯ РАБОТА № 5

Моделирование памяти (ОЗУ) при помощи VHDL

Цель работы: ознакомиться с различными типами оперативной памяти. Получить практические навыки реализации и тестирования оперативной памяти на VHDL в среде ActiveHDL.

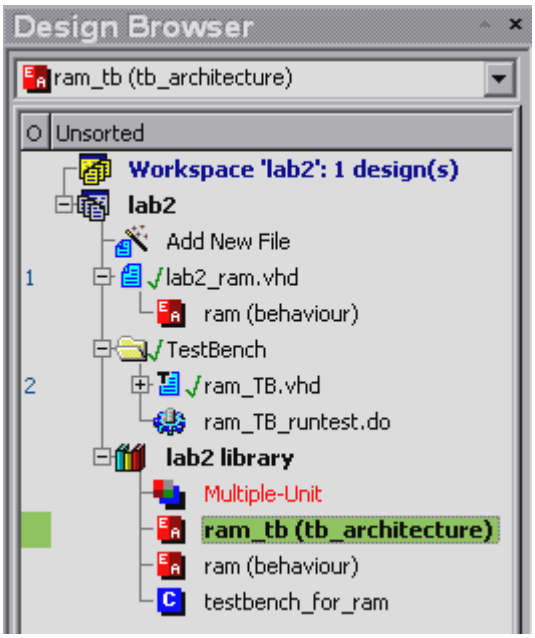
1. Моделирование памяти с произвольным доступом. (RAM – Random Access Memory).
   - 1.1. Память с произвольным доступом позволяет осуществлять чтение/запись данных по любому адресу в произвольном порядке. Память с произвольным доступом содержит:
     - 1.1.1. Шину адреса.
     - 1.1.2. Шину данных.
     - 1.1.3. Управляющие линии:
       - 1.1.3.1. Выбор микросхемы.
       - 1.1.3.2. Разрешение записи.
       - 1.1.3.3. Вход тактирования.
   - 1.2. Создать новый проект.
   - 1.3. Добавить в проект пустой файл исходного кода VHDL.
   - 1.4. Записать в созданный файл исходный код для реализации простейшей памяти с произвольным доступом.
   - 1.5. Для создания тестового модуля, который необходим для расширенного тестирования памяти, вызвать пункт меню «Tools» $\Rightarrow$ «Generate Test Bench» («Инструменты» $\Rightarrow$ «Генерация тестового модуля»).
   - 1.6. Выбрать Entity (Структура) и Architecture (Архитектура), для которой будет создан тестовый модуль. Для листинга это «ram» и «behaviour» соответственно.
   - 1.7. Два раза нажать «Далее>>». Ввести имя тестового модуля. Нажать «Далее>>» и «Готово».
   - 1.8. Открыть созданный VHDL-файл с тестовым модулем.



1.9. Добавить в файл тестового модуля код для генерации тестовых последовательностей.

1.10. Откомпилировать проект.

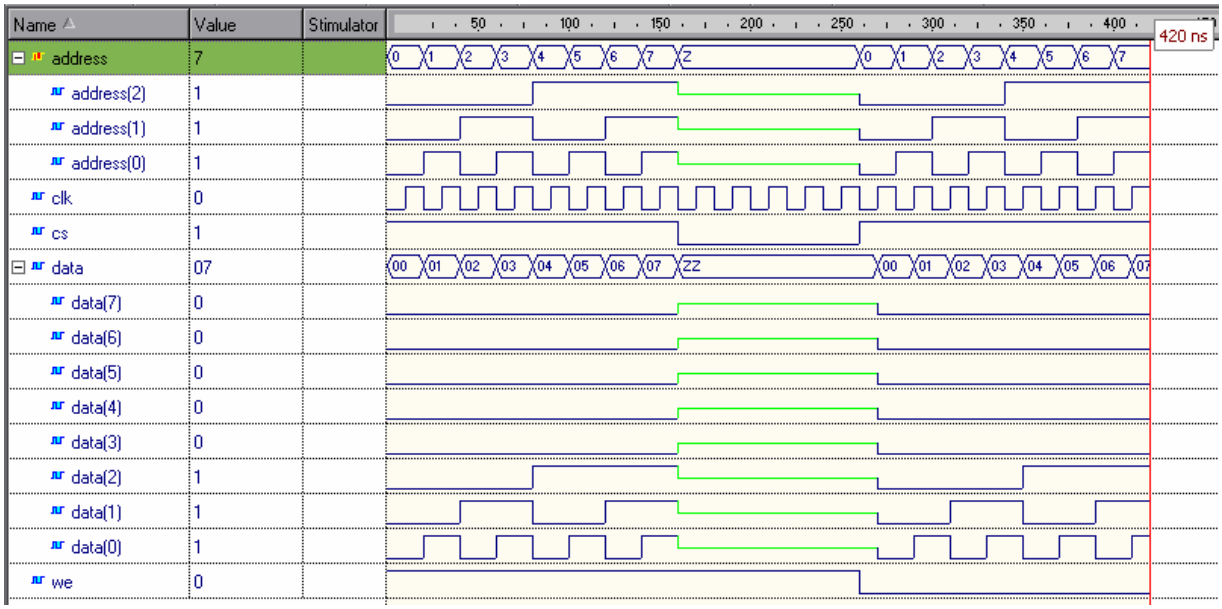
1.11. Выбрать в качестве модуля верхнего уровня (Top Level) пару Entity – Architecture для созданного тестового модуля.



1.12. Создать новый Waveform и добавить в него все 5 сигналов (address, data, cs, we, clk).

1.13. Запустить симуляцию проекта и просмотреть результат в окне Waveform. Убедиться, что считанные значения равны записанным для соответствующих адресов. На рисунке виден этап записи в память, когда для каждого адреса на шину данных выставляется требуемое значение. Затем пауза, когда шины

данных и адреса находятся в высокоимпедансном состоянии. После чего – этап чтения, во время которого память выставляет на шину данных значения, расположенные по соответствующим адресам.



1.14. Выполнить размещение полученного модуля на кристалле. Дополнительная информация – в. п. 3.1.

1.15. Выполнить все вышеприведенные пункты – в. п. 3.2.

2. Моделирование стековой памяти (LIFO – Last In First Out).

2.1. Стековая память позволяет производить операции чтения-записи с вершиной стека. При записи значения в стек указатель вершины стека увеличивается и по адресу вершины стека записывается новое значение. При чтении значения из стека возвращается значение из вершины стека и указатель стека уменьшается. Стековая память для доступа к ней содержит:

2.1.1. Шину данных.

2.1.2. Управляющие линии:

2.1.2.1. Выбор микросхемы.

2.1.2.2. Разрешение записи.

2.1.2.3. Вход тактирования.

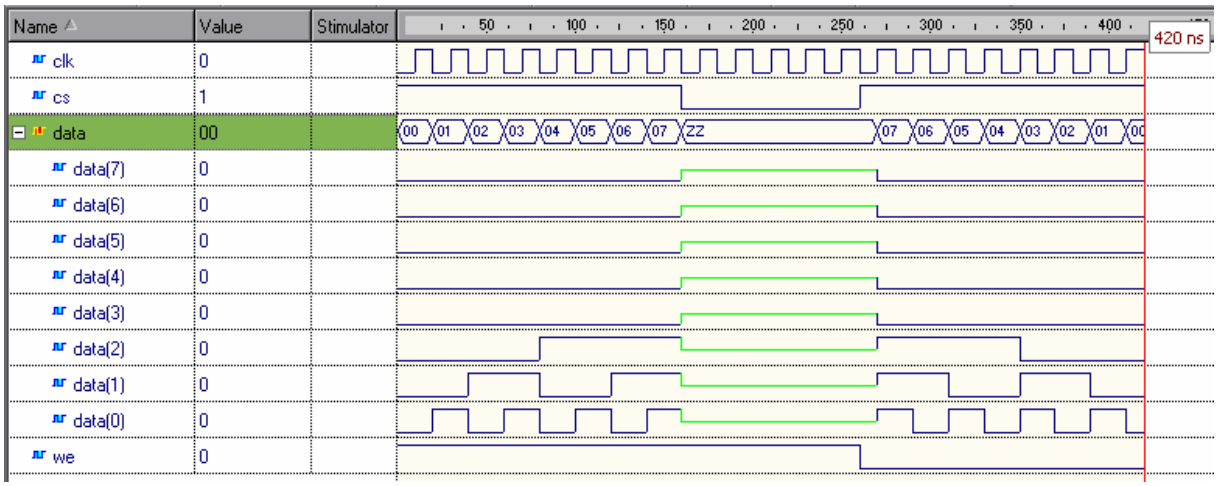
2.2. Добавить в проект пустой файл исходного кода VHDL.

2.3. Записать в созданный файл исходный код для реализации простейшего стека.

2.4. Создать тестовый модуль для модуля стека.

2.5. Добавить в файл тестового модуля код для генерации тестовых последовательностей (прил. 4).

2.6. Произвести моделирование стека и просмотреть результаты моделирования при помощи Waveforms. На приведенном рисунке видно три этапа работы: запись, пауза, чтение. На время паузы сигнал CS устанавливается в ноль.



2.7. Выполнить размещение полученного модуля на кристалле. Дополнительная информация – в п. 3.1.

2.8. Выполнить все вышеприведенные пункты с использованием блочной памяти. Дополнительная информация – в п. 3.2.

3. Дополнительная информация:

3.1. При размещении на кристалле необходимо исключить из синтеза тестовый модуль. Для этого в окне опций синтеза в контекстном меню соответствующего модуля необходимо снять флажок «Include to synthesis» (включать в синтез).

3.2. Использование блочной памяти:

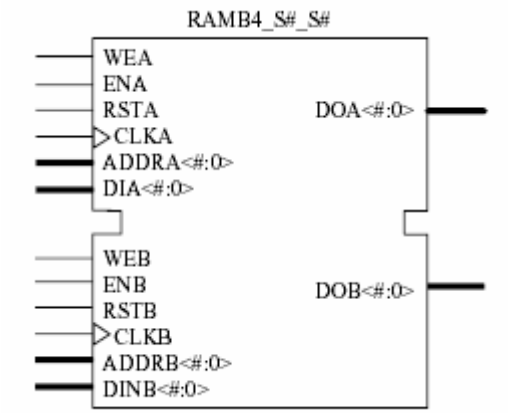
3.2.1. Блочная память создана в дополнение к распределенной памяти, которая имеет небольшую емкость.

3.2.2. Общий объем блочной памяти зависит от числа блоков Block Select RAM на кристалле.

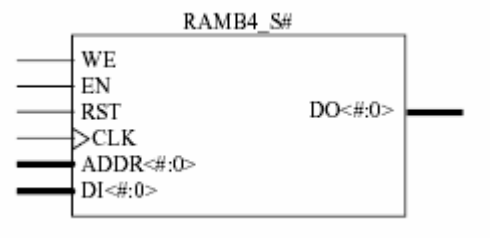
3.2.3. Каждый блок памяти – это полностью синхронная двухпортовая RAM с независимым управлением для каждого порта. Размерность шины данных для обоих портов может быть сконфигурирована независимо. Каждый порт памяти может быть использован как отдельная однопортовая память.

3.2.4. В стандартных библиотеках существует набор примитивов, позволяющих обращаться к блочной памяти.

3.2.5. Графическое обозначение интерфейса примитива двухпортовой блочной памяти:



3.2.6. Графическое обозначение интерфейса примитива однопортовой блочной памяти:



3.2.7. На графических схемах обозначены следующие входы-выходы:

3.2.7.1. CLK – тактирование памяти.

3.2.7.2. EN – разрешение работы памяти. Если на этом входе установлен неактивный сигнал, то запись в память не может выполняться, а на выходных линиях данных сохраняется предыдущее состояние.

3.2.7.3. WE – разрешение записи.

3.2.7.4. RST – сброс.

3.2.7.5. ADDR – шина адреса.

3.2.7.6. DI – входная шина данных. Предназначена для записи данных в память.

3.2.7.7. DO – выходная шина данных. Предназначена для чтения данных из памяти.

3.2.8. Существуют различные примитивы памяти, применение которых зависит от необходимой разрядности шин адреса и данных. Основной характеристикой примитива памяти

является разрядность порта памяти, которая фактически является разрядностью шины данных примитива. Ниже в таблице приведены возможные значения разрядности и соответствующие им определения шины адреса и шины данных.

Разрядность	Количество ячеек	Шина адреса	Шина данных
1	4096	ADDR<11:0>	DATA<0>
2	2048	ADDR<10:0>	DATA<1:0>
4	1024	ADDR<9:0>	DATA<3:0>
8	512	ADDR<8:0>	DATA<7:0>
16	256	ADDR<7:0>	DATA<15:0>

3.2.9. В зависимости от разрядности портов можно определить название необходимого примитива памяти:

Название примитива	Разрядность порта A	Разрядность порта B
RAMB4_S1	1	Отсутствует
RAMB4_S1_S1		1
RAMB4_S1_S2		2
RAMB4_S1_S4		4
RAMB4_S1_S8		8
RAMB4_S1_S16		16
RAMB4_S2	2	Отсутствует
RAMB4_S2_S2		2
RAMB4_S2_S4		4
RAMB4_S2_S8		8
RAMB4_S2_S16		16
RAMB4_S4	4	Отсутствует
RAMB4_S4_S4		4
RAMB4_S4_S8		8
RAMB4_S4_S16		16
RAMB4_S8	8	Отсутствует
RAMB4_S8_S8		8
RAMB4_S8_S16		16
RAMB4_S16	16	Отсутствует
RAMB4_S16_S16		16

3.2.10. Простейшим случаем использования примитива блочной

памяти является отображение его портов на порты пользовательского модуля. Использование блочной памяти в таком варианте включает в себя следующие шаги:

3.2.10.1. Подключение библиотеки, в которой находятся примитивы блочной памяти, *library unisim.*

3.2.10.2. Описание компонента используемого примитива блочной памяти:

```
component ramb4_s8
        port(
             DI : in std_logic_vector(7 downto 0);
             EN : in std_ulogic;
             WE : in std_ulogic;
             RST : in std_ulogic;
             CLK : in std_ulogic;
             ADDR : in std_logic_vector(8 downto 0);
             DO : out std_logic_vector(7 downto 0));
end component;
```

3.2.10.3. Описание пользовательского модуля, использующего блочную память:

```
  entity ram is
    port (address: in std_logic_vector(2 downto 0);
          data: inout std_logic_vector(7 downto 0) :=
          "ZZZZZZZZ";
          cs, we, clk: in std_logic);
  end;
```

3.2.10.4. Отображение портов на компонент блочной памяти. Следует обратить внимание на то, что неиспользуемые разряды шины адреса и/или данных должны быть отображены на константные значения. В противном случае произойдет ошибка компиляции.

```
U_RAMD: ramb4_s8   port map(
             EN =>cs,
             WE =>we,
             ADDR(8 downto 3) => "000000",
             ADDR(2 downto 0) => address,
             DO => data_out,
             DI => data_in,
             RST => '0',
             CLK => clk);
```

Порядок выполнения работы:

1. Ознакомиться с требованиями методических указаний по выполнению лабораторной работы.
2. Изучить возможности моделирования памяти языка VHDL.
3. Изучить возможности создания тестовых VHDL-модулей в среде ActiveHDL.
4. Реализовать по индивидуальному заданию память с произвольным доступом и стековую память:
   - 4.1. Память должна иметь размер, указанный в индивидуальном задании.
   - 4.2. Память с произвольным доступом должна хранить данные только в промежутке адресов, указанном в индивидуальном задании.
   - 4.3. В памяти с произвольным доступом должна быть предусмотрена нормальная работа при попытке чтения/записи по несуществующему адресу.
   - 4.4. В стековой памяти должна быть предусмотрена нормальная работа в случае переполнения стека и при попытке извлечения значения из пустого стека.
5. Повторить пункт 4, но с использованием блочной памяти.
6. Протестировать работу полученного кода.

Структура отчета:

1. Название и цель работы.
2. Индивидуальное задание на лабораторную работу.
3. Исходный VHDL-текст реализованной памяти с произвольным доступом и стековой памяти.
4. Исходный VHDL-текст реализованной памяти с произвольным доступом и стековой памяти с применением блочной памяти.
5. Исходный VHDL-текст реализованных тестовых модулей.
6. Набор тестовых и результирующих воздействий. Воздействия должны отображать функционирование памяти как в нормальных, так и в крайних (несуществующие адреса, переполнение стека, чтение из пустого стека) условиях.

Тестовый модуль для памяти с произвольным доступом

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

        -- Add your library and packages declaration here ...

entity ram_tb is
end ram_tb;

architecture TB_ARCHITECTURE of ram_tb is
        -- Component declaration of the tested unit
        component ram
        port(
                address : in std_logic_vector(2 downto 0);
                data : inout std_logic_vector(7 downto 0);
                cs : in std_logic;
                we : in std_logic;
                clk : in std_logic );
        end component;

        -- Stimulus signals - signals mapped to the input and inout
ports of tested entity
        signal address : std_logic_vector(2 downto 0);
        signal cs : std_logic;
        signal we : std_logic;
        signal clk : std_logic;
        signal data : std_logic_vector(7 downto 0);
        -- Observed signals - signals mapped to the output ports of
tested entity

        -- Add your code here ...

begin

        -- Unit Under Test port map
        UUT : ram
                port map (
                        address => address,
                        data => data,
                        cs => cs,
                        we => we,
                        clk => clk
                );

    -- Add your stimulus here ...

-- Процесс, генерирующий тактовые импульсы
clk_generator:
process
begin
        clk <= '0';
        wait for 10ns;
        clk <= '1';
        wait for 10ns;
end process;

-- Процесс, генерирующий тестовые последовательности
```

```
data_generator:
process
-- Переменные для хранения генерируемого адреса и данных
variable integer_address: integer := 0;
variable integer_data: integer := 0;
begin
        -- Выбираем микросхему и переходим в режим записи
        we <= '1';
    cs <= '1';
    -- Циклически перебираем адреса от 0 до 7
        while integer_address <= 7 loop
        -- Выводим адрес и данные
                address  <=  std_logic_vector(  conv_unsigned(  inte-
ger_address, 3 ) );
        data <= std_logic_vector( conv_unsigned( integer_data, 8 ) );
        -- Инкремент адреса и данных
                integer_address := integer_address + 1;
        integer_data := integer_data + 1;
                -- Пауза
        wait for 20ns;
    end loop;
    -- Некоторое время не взаимодействуем с памятью
        cs <= '0';
    address <= "ZZZ";
    data <= "ZZZZZZZZ";
    wait for 100ns;

        -- Выбираем микросхему и переходим в режим чтения
        -- Двунаправленный порт данных оставляем в высокоимпендансном
        -- состоянии, что бы можно было читать данные из памяти
        we <= '0';
    cs <= '1';

    -- Циклически перебираем адреса от 0 до 7
        integer_address := 0;
    while integer_address <= 7 loop
        -- Выдаем адрес на шину
                address  <=  std_logic_vector(  conv_unsigned(  inte-
ger_address, 3 ) );
                -- Инкрементируем адрес
        integer_address := integer_address + 1;
                -- Пауза
        wait for 20ns;
    end loop;
    -- Конец процесса
        wait;

end process;

end TB_ARCHITECTURE;

configuration TESTBENCH_FOR_ram of ram_tb is
        for TB_ARCHITECTURE
                for UUT : ram
                        use entity work.ram(behaviour);
                end for;
        end for;
end TESTBENCH_FOR_ram;
```

Тестовый модуль для стековой памяти

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

        -- Add your library and packages declaration here ...

entity stack_tb is
end stack_tb;

architecture TB_ARCHITECTURE of stack_tb is
        -- Component declaration of the tested unit
        component stack
        port(
                data : inout std_logic_vector(7 downto 0);
                cs : in std_logic;
                we : in std_logic;
                clk : in std_logic );
        end component;

        -- Stimulus signals - signals mapped to the input and inout
ports of tested entity
        signal cs : std_logic;
        signal we : std_logic;
        signal clk : std_logic;
        signal data : std_logic_vector(7 downto 0);
        -- Observed signals - signals mapped to the output ports of
tested entity

        -- Add your code here ...

begin

        -- Unit Under Test port map
        UUT : stack
                port map (
                        data => data,
                        cs => cs,
                        we => we,
                        clk => clk
                );

        -- Add your stimulus here ...

-- Процесс, генерирующий тактовые импульсы
clk_generator:
process
begin
        clk <= '0';
        wait for 10ns;
        clk <= '1';
        wait for 10ns;
end process;

-- Процесс, генерирующий тестовые последовательности
```

```
data_generator:
process
-- Переменные для хранения генерируемого адреса и данных
variable integer_data: integer := 0;
begin
        -- Выбираем микросхему, но не устанавливаем никакой режим
        cs <= '1';
    we <= '1';
    -- Циклически записываем в стек числа от 0 до 7
        while integer_data <= 7 loop
        -- Выводим данные
        data <= std_logic_vector( conv_unsigned( integer_data, 8 )
);
                -- Строб записи на 20нс
                wait for 20 ns;
                -- Инкремент данных
        integer_data := integer_data + 1;
    end loop;
    -- Некоторое время не взаимодействуем с памятью
        cs <= '0';
    data <= "ZZZZZZZZ";
    wait for 100ns;

        -- Выбираем микросхему, но не устанавливаем никакой режим
        -- Двунаправленный порт данных оставляем в высокоимпендансном
        -- состоянии, чтобы можно было читать данные из памяти
        we <= '0';
    cs <= '1';

    -- Выполняем 8 чтений из стековой памяти
    integer_data := 0;
        while integer_data <= 7 loop
        -- Выдаем строб на чтение
                wait for 20 ns;
                -- Инкрементируем счетчик циклов
        integer_data := integer_data + 1;
    end loop;
    -- Конец процесса
        wait;

end process;


end TB_ARCHITECTURE;

configuration TESTBENCH_FOR_stack of stack_tb is
        for TB_ARCHITECTURE
                for UUT : stack
                        use entity work.stack(behaviour);
                end for;
        end for;
end TESTBENCH_FOR_stack;
```

Варианты заданий к лабораторной работе № 5

Вариант	Разрядность	Размер стека	Начальный адрес	Конечный адрес
1	5		10	50
2	6	15		
3	7		30	60
4	8	13		
5	9		10	60
6	10	11		
7	5		30	55
8	6	9		
9	7		10	65
10	8	16		
11	9		30	50
12	10	14		
13	5		10	60
14	6	12		
15	7		30	60
16	8	10		
17	9		10	55
18	10	8		
19	5		30	65
20	6	15		
21	7		10	50
22	8	13		
23	9		30	60
24	10	11		
25	5		10	60
26	6	9		
27	7		30	55
28	8	16		
29	9		10	65
30	10	14		
31	5		30	50

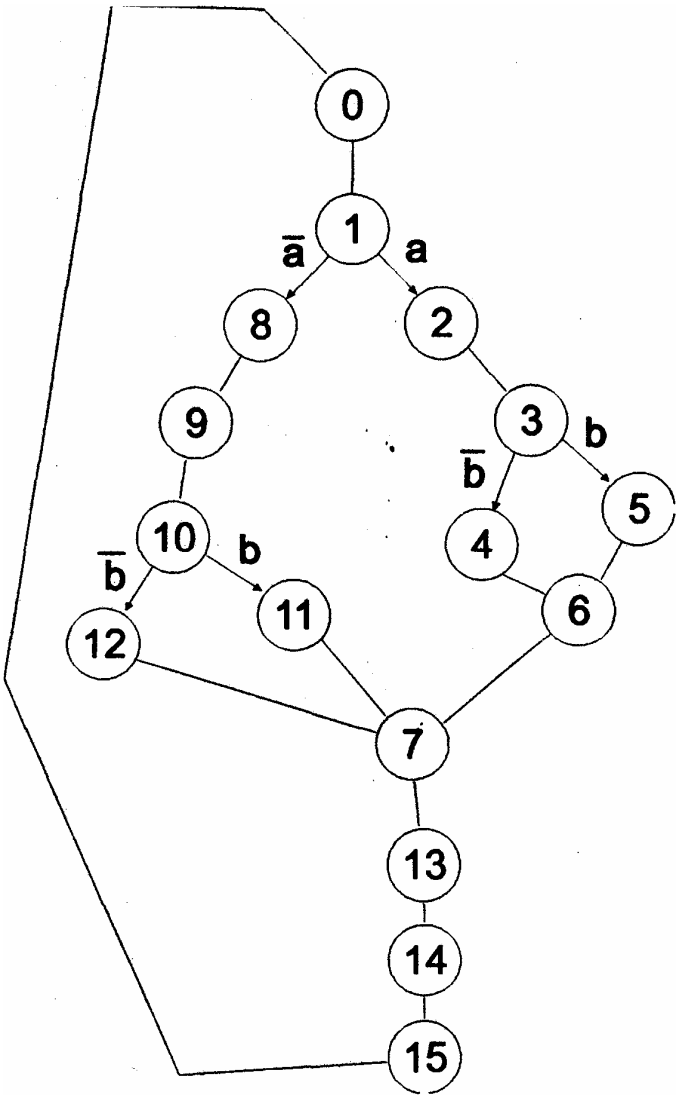
ЛАБОРАТОРНАЯ РАБОТА № 6

Проектирование цифрового автомата

Цель работы: графический ввод схем, компилирование и моделирование на примере автомата.

Задание: ввести схему автомата, выполнить компиляцию и моделирование.

Исходные данные: граф состояний.



Значения состояний взять из табл. 1 для своего варианта.

Таблица 1

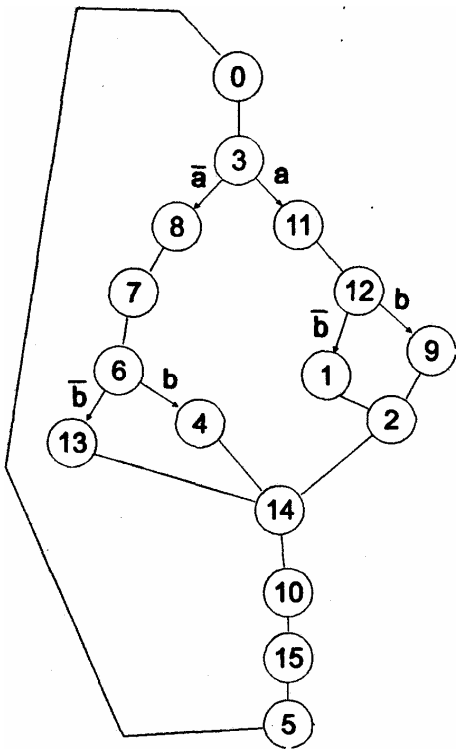
№ вар.	Состояния графа															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	3	11	12	1	9	2	14	8	7	6	4	13	10	15	5
2	5	0	3	11	12	1	9	2	14	8	7	6	4	13	10	15
3	15	5	0	3	11	12	1	9	2	14	8	7	6	4	13	10
4	10	15	5	0	3	11	12	1	9	2	14	8	7	6	4	13
5	13	10	15	5	0	3	11	12	1	9	2	14	8	7	6	4
6	4	13	10	15	5	0	3	11	12	1	9	2	14	8	7	6
7	6	4	13	10	15	5	0	3	11	12	1	9	2	14	8	7
8	7	6	4	13	10	15	5	0	3	11	12	1	9	2	14	8
9	8	7	6	4	13	10	15	5	0	3	11	12	1	9	2	14
10	14	8	7	6	4	13	10	15	5	0	3	11	12	1	9	2
11	2	14	8	7	6	4	13	10	15	5	0	3	11	12	1	9
12	9	2	14	8	7	6	4	13	10	15	5	0	3	11	12	1
13	1	9	2	14	8	7	6	4	13	10	15	5	0	3	11	12
14	12	1	9	2	14	8	7	6	4	13	10	15	5	0	3	11
15	11	12	1	9	2	14	8	7	6	4	13	10	15	5	0	3
16	3	11	12	1	9	2	14	8	7	6	4	13	10	15	5	0
17	0	5	15	10	13	4	6	7	.8	14	2	9	1	12	11	3
18	3	0	5	15	10	13	4	6	7	8	14	2	9	1	12	11
19	11	3	0	5	15	10	13	4	6	7	8	14	2	9	1	12
20	12	11	3	0	5	15	10	13	4	6	7	8	14	2	9	1
21	1	12	11	3	0	5	15	10	13	4	6	7	8	14	2	9
22	9	1	12	11	3	0	5	15	10	13	4	6	7	8	14	2
23	2	9	1	12	11	3	0	5	15	10	13	4	6	7	8	14
24	14	2	9	1	12	11	3	0	5	15	10	13	4	6	7	8
25	8	14	2	9	1	12	11	3	0	5	15	10	13	4	6	7
26	7	8	14	2	9	1	12	11	3	0	5	15	10	13	4	6
27	6	7	8	14	2	9	1	12	11	3	0	5	15	10	13	4
28	4	6	7	8	14	2	9	1	12	11	3	0	5	15	10	13
29	13	4	6	7	8	14	2	9	1	12	11	3	0	5	15	10
30	10	13	4	6	7	8	14	2	9	1	12	11	3	0	5	15
31	15	10	13	4	6	7	8	14	2	9	1	12	11	3	0	5
32	5	15	10	13	4	6	7	8	14	2	9	1	12	11	3	0

Пример выполнения варианта № 1

Составляем таблицу перекодировки состояний автомата и их двоичного кода:

№ состояния	№ состояния из табл. 1	Двоичный код q3,q2,ql,q0
0	0	0000
1	3	0011
2	11	1011
3	12	1100
4	1	0001
5	9	1001
6	2	0010
7	14	1110
8	8	1000
9	7	0111
10	6	0110
11	4	0100
12	13	1101
13	10	1010
14	15	1111
15	5	0101

Подставляем новые значения в граф состояний:



Составляем таблицу истинности автомата:

Старое состояние		Условие	Новое состояние	
№	код		№	КОД
0	0000	–	3	0011
3	0011	A = 0	8	1000
3	0011	A = 1	11	1011
8	1000	–	7	0111
7	0111	–	6	0110
6	0110	B = 0	13	1101
6	0110	B = 1	4	0100
13	1101	–	14	1110
14	1110	–	10	1010
10	1010	–	15	1111
15	1111	–	5	0101
5	0101	–	0	0000
11	1011	–	12	1100
12	1100	B = 0	1	0001
12	1100	B = 1	9	1001
1	0001	–	2	0010
2	0010	–	14	1110
4	0100	–	14	1110
9	1001	–	2	0010

После составления таблицы истинности автомата создается функциональная схема в САПР XILINX без минимизации.

Структура отчета:

1. Титульный лист.
2. Содержание.
3. Задание к лабораторной работе.
4. Таблица истинности и схема автомата.
5. Реализация в графическом редакторе.
6. Временная диаграмма работы автомата.
7. Выводы с подтверждением по таблице истинности.

ЛАБОРАТОРНАЯ РАБОТА № 7

Проектирование устройств на базе ПЛИС фирмы XILINX (ч.1)

Цель работы: схемотехническое проектирование конкретных устройств на базе САПР XILINX WebPack ISE 11.1

Порядок выполнения лабораторной работы:

1. В окне графического редактора создать схему устройства, собранного на основе логических элементов.
2. В редакторе сигналов создать файл с тестовыми сигналами.
3. Проверить соответствие выходного сигнала логике работы устройства.
4. Построить это же устройство, используя готовый узел устройства из библиотеки, и повторно протестировать работу схемы. На основании анализа временных диаграмм сравнить быстродействие обоих вариантов схемы.
5. Исследовать работу созданной схемы с помощью подходящих тестовых сигналов, оценить временные параметры готового устройства. Созданные файлы проекта (графический файл схемы, файл испытательных сигналов) сохранить для выполнения следующей лабораторной работы.
6. Оформить отчет в соответствии с требованиями и сдать преподавателю.

Структура отчета:

1. Титульный лист.
2. Содержание.
3. Задание к лабораторной работе.
4. Реализация в графическом редакторе.
5. Временная диаграмма работы устройства.
6. Выводы с подтверждением правильности проектирования.

Варианты индивидуальных заданий

№ вар.	Задание
1	19-входовый мультиплексор с входом разрешения
2	17-выходной демультиплексор с входом разрешения
3	18-разрядный синхронный суммирующий счетчик с асинхронным сбросом
4	14-разрядный синхронный вычитающий счетчик с входом асинхронной установки
5	12-разрядный синхронный реверсивный счетчик с асинхронным сбросом
6	9-разрядный параллельный сумматор с входом и выходом переноса
7	12-разрядный цифровой компаратор с входами и выходами, позволяющими обеспечить наращивание разрядности
8	18-разрядный регистр сдвига с параллельной загрузкой и последовательным выходом
9	18-разрядный регистр сдвига с последовательным входом и параллельным выходом
10	6-разрядная схема контроля четности с параллельной загрузкой и выходом четности
11	4-разрядный счетчик Джонсона с выходным дешифратором, формирующим двоичный код
12	14-разрядный регистр хранения с параллельной загрузкой и Z-состоянием выходной шины
13	Цифровая линия задержки 8-разрядного слова на 12 тактов с параллельными входом и выходом
14	АЛУ, обеспечивающее сложение или вычитание 4-разрядных операндов в зависимости от состояния управляющего входа
15	Формирователь пачек импульсов, длина которых задается 4-разрядным словом (тактирование от внешнего синхросигнала)
16	Двоично-десятичный синхронный счетчик емкостью 100
17	Делитель частоты на 37
18	Устройство, выделяющее каждый пятый бит из последовательного цифрового сигнала
19	10-разрядный реверсивный регистр сдвига с параллельной загрузкой
20	6-разрядная схема контроля четности с параллельной загрузкой и выходом нечетности
21	12-разрядный регистр хранения с общей шиной для чтения и записи
22	Одновибратор, длительность импульса которого задается параллельным 4-разрядным кодом (тактовый сигнал – внешний)
23	13-разрядный синхронный счетчик с Z-состоянием на выходе и асинхронным сбросом
24	18-разрядный измеритель временных интервалов с запуском по фронту, остановкой счета по срезу и асинхронным сбросом
25	18-разрядный «Бегущий огонь» с внешним тактированием и 8 режимами, задаваемыми внешним параллельным сигналом
26	16-разрядный счетчик импульсов, поступающих по 3-м независимым линиям. Необходим асинхронный сброс
27	Формирователь секундных, минутных и часовых импульсов из входного тактового сигнала частотой 32768 Гц
28	АЛУ, обеспечивающее сравнение или вычитание 4-разрядных операндов в зависимости от состояния управляющего входа
29	12-разрядный двоичный счетчик с параллельной загрузкой и чтением по одной шине
30	Мультиплексор структуры 12x4 с Z-состоянием на выходе

ЛАБОРАТОРНАЯ РАБОТА № 8

Проектирование устройств на базе ПЛИС фирмы XILINX (ч. 2)

Цель работы: проектирование на языке VERILOG конкретных устройств на базе САПР XILINX WebPack ISE 11.1.

Порядок выполнения лабораторной работы:

1. В окне редактора языка создать программу, обеспечивающую работу заданного устройства.
2. В редакторе сигналов создать файл с тестовыми сигналами.
3. Проверить соответствие выходного сигнала логике работы устройства.
4. Построить это же устройство, используя готовый узел устройства из библиотеки, и повторно протестировать работу схемы. На основании анализа временных диаграмм сравнить быстродействие обоих вариантов схемы.
5. Исследовать работу созданной схемы с помощью подходящих тестовых сигналов, оценить временные параметры готового устройства.

Оформить отчет в соответствии с требованиями и сдать преподавателю.

Структура отчета:

1. Титульный лист.
2. Содержание.
3. Задание к лабораторной работе.
4. Реализация на языке программирования.
5. Временная диаграмма работы устройства.
6. Выводы с подтверждением правильности проектирования.

Варианты индивидуальных заданий

№ вар.	Задание
1	19-входовый мультиплексор со входом разрешения
2	17-выходной демультиплексор со входом разрешения
3	18-разрядный синхронный суммирующий счетчик с асинхронным сбросом
4	14-разрядный синхронный вычитающий счетчик со входом асинхронной установки
5	12-разрядный синхронный реверсивный счетчик с асинхронным сбросом
6	9-разрядный параллельный сумматор с входом и выходом переноса
7	12-разрядный цифровой компаратор с входами и выходами, позволяющими обеспечить наращивание разрядности
8	18-разрядный регистр сдвига с параллельной загрузкой и последовательным выходом
9	18-разрядный регистр сдвига с последовательным входом и параллельным выходом
10	6-разрядная схема контроля четности с параллельной загрузкой и выходом четности
11	4-разрядный счетчик Джонсона с выходным дешифратором, формирующим двоичный код
12	14-разрядный регистр хранения с параллельной загрузкой и Z-состоянием выходной шины
13	Цифровая линия задержки 8-разрядного слова на 12 тактов с параллельными входом и выходом
14	АЛУ, обеспечивающее сложение или вычитание 4-разрядных операндов в зависимости от состояния управляющего входа
15	Формирователь пачек импульсов, длина которых задается 4-разрядным словом (тактирование от внешнего синхросигнала)
16	Двоично-десятичный синхронный счетчик емкостью 100
17	Делитель частоты на 37
18	Устройство, выделяющее каждый пятый бит из последовательного цифрового сигнала
19	10-разрядный реверсивный регистр сдвига с параллельной загрузкой
20	6-разрядная схема контроля четности с параллельной загрузкой и выходом нечетности
21	12-разрядный регистр хранения с общей шиной для чтения и записи
22	Одновибратор, длительность импульса которого задается параллельным 4-разрядным кодом (тактовый сигнал – внешний)
23	13-разрядный синхронный счетчик с Z-состоянием на выходе и асинхронным сбросом
24	18-разрядный измеритель временных интервалов с запуском по фронту, остановкой счета по срезу и асинхронным сбросом
25	18-разрядный «Бегущий огонь» с внешним тактированием и 8 режимами, задаваемыми внешним параллельным сигналом
26	16-разрядный счетчик импульсов, поступающих по 3-м независимым линиям. Необходим асинхронный сброс
27	Формирователь секундных, минутных и часовых импульсов из входного тактового сигнала частотой 32768 Гц
28	АЛУ, обеспечивающее сравнение или вычитание 4-разрядных операндов в зависимости от состояния управляющего входа
29	12-разрядный двоичный счетчик с параллельной загрузкой и чтением по одной шине
30	Мультиплексор структуры 12x4 с Z-состоянием на выходе

ЛИТЕРАТУРА

1. Соловьев, В.В. Программируемые логические интегральные схемы и их применение / В.В. Соловьев, А.Г. Васильев. – Минск: Белорусская наука, 1998.
2. Скляров, В.А. Синтез автоматов на матричных БИС / В.А. Скляров; под ред. С.И. Баранова. – Минск: Наука и техника, 1984.
3. Логическое проектирование СБИС: пер с япон. / К. Киносита [и др.]. – М.: Мир, 1988.
4. Проектирование СБИС: пер. с япон. / М. Вамапебэ.[и др.]. – М.: Мир, 1988.
5. Курейчик, В.М. Математическое обеспечение конструкторского и технологического проектирования с применением САПР: учебник для вузов / В.М. Курейчик. – М.: Радио и связь, 1990.
6. Разработка САПР. В 10 кн. / под ред. А.В. Петрова. – М.: Высш. шк., 1990.
7. Автоматизация проектирования БИС: практ. пособие. В 6 кн. Кн. 1: Принципы и методология построения САПР БИС / под ред. Г.Г. Казеннова. – М.: Высш. шк., 1990.
8. Автоматизация проектирования БИС: практ. пособие. В 6 кн. Кн. 3: Рабочие станции в проектировании БИС / под ред. Г.Г. Казеннова. – М.: Высш. шк., 1990.
9. Интеграция данных в САПР БИС. Направления практической реализации / Ю.Н. Белеков [и др.]. – М.: Радио и связь, 1990.
10. Оре, О. Теория графов / О. Оре. – М.: Мир, 1980.

Приложение

Пример оформления титульного листа

Министерство образования Республики Беларусь

Учреждение образования «Полоцкий государственный университет»

Кафедра вычислительных систем и сетей

ОТЧЕТ
по лабораторной работе № ___
_____
«название работы»
по предмету «Автоматизация проектирования ЭВМ»

Выполнил(а): /Фамилия И.О. студента/

Группа _____

Проверил: /Фамилия И.О. преподавателя/

НОВОПОЛОЦК
20__г.

СОДЕРЖАНИЕ

Учебное издание

КАЛИНЦЕВ Сергей Викторович

АВТОМАТИЗАЦИЯ ПРОЕКТИРОВАНИЯ ЭВМ

Методические указания
к выполнению лабораторных работ для студентов специальности
1-40 02 01 «Вычислительные машины, системы и сети»

Редактор *Т. В. Булах*

Подписано в печать 24.06.10. Формат 60×84/16. Бумага офсетная.
Ризография. Усл.-печ. л. 4,64. Уч.-изд. л. 3,2. Тираж 25 экз. Заказ 1036.

Издатель и полиграфическое исполнение –
учреждение образования «Полоцкий государственный университет»

ЛИ № 02330/0548568 от 26.06.2009 ЛП № 02330/0494256 от 27.05.2009

Ул. Блохина, 29, 211440, г. Новополоцк.