БГУИР

Кафедра ЭВМ

Отчет по лабораторной работе № 3
Тема: «Разработка тестовых модулей на языке VHDL в среде WEBPACK»
Вариант 7

Выполнил:

Проверил:

МИНСК 2024

1 ЦЕЛЬ РАБОТЫ

Получить навыки разработки тестовых модулей на языке VHDL.

2 ИСХОДНЫЕ ДАННЫЕ К РАБОТЕ

Задание для лабораторной работы состоит из двух частей.

1. Для комбинационного устройства, разработанного в лабораторной работе №1 (вариант с логическими операциями и параллельным оператором безусловного присваивания), создать два варианта тестовых модулей со встроенной проверкой корректности работы устройства.

В первом варианте исходные воздействия, подаваемые на разработанное устройство, и эталонные результаты работы устройства должны считываться тестовым модулем из текстового файла. Текстовый файл может создаваться любым способом (в текстовом редакторе, программой на языке VHDL, C и т.д.).

На рисунке 2.1 представлен 7 вариант комбинационного устройства, разработанного в лабораторной работе №1.

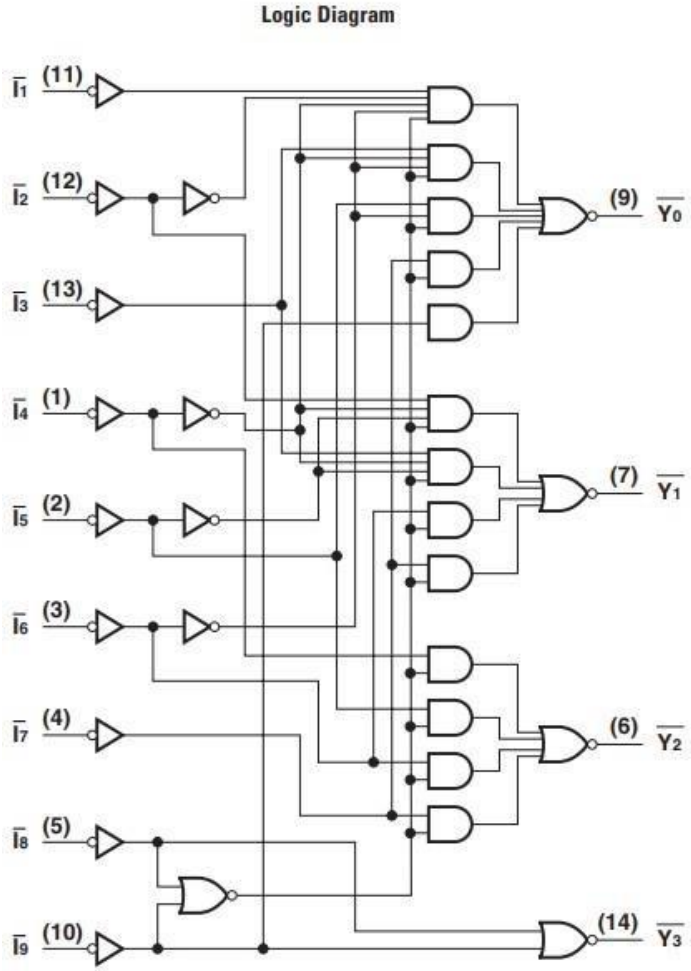

Рисунок 2.1 – Схема комбинационного устройства, разработанного в лабораторной работе №1

Во втором варианте входные тестовые воздействия могут формироваться любым способом (считываться из текстового фала, формироваться непосредственно в тестовом модуле и т.д.), а для формирования эталонных воздействий в качестве эталона необходимо использовать любой другой вариант описания этого же устройства из лабораторной работы №1.

Тестовые воздействия в обоих вариантах должны быть полными. Для комбинационного устройства, разработанного в лабораторной работе №1, это означает полный перебор входных значений. В случае обнаружения ошибки в работе устройства необходимо выдать сообщение на консоль программы моделирования.

2. Для функционального узла последовательного типа, разработанного в лабораторной работе №1, создать тестовый модуль с автоматической проверкой корректности работы устройства. Тестовое воздействие может формироваться любым способом и должно быть по возможности полным, то есть тестировать работу устройства во всех его режимах. В случае обнаружения ошибки в работе устройства необходимо выдать сообщение на консоль программы моделирования.

3 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

3.1 Общие сведения по написанию тестовых модулей на языке VHDL

Основная цель тестовых модулей – проверить функционирование разрабатываемого устройства путем подачи на его вход тестовых воздействий и анализа результатов моделирования его работы. При этом анализ результатов моделирования может выполняться автоматически самим тестовым модулем либо вручную разработчиком.

В целом, тестовый модуль выполняет следующие задачи:

- создает экземпляр объекта для тестирования (моделирования);
- формирует на входе созданного экземпляра объекта тестовое воздействие;
- при необходимости выводит в консоль программы моделирования различную информацию;
- при необходимости проводит автоматическую проверку функционирования тестируемого устройства.

Тестовый модуль на языке VHDL представляется проектными модулем `entity`, который не содержит внешнего интерфейса, то есть портов ввода/вывода. При этом в проектном модуле `architecture`, представляющем реализацию тестового модуля, создается экземпляр тестируемого объекта, на вход которого подаются сформированные тестовые воздействия.

Тестовые воздействия могут формироваться внутри тестового модуля с помощью конструкций языка VHDL или могут считываться из внешнего источника, например из файла.

Автоматическая проверка корректности работы устройства выполняется путем сравнения результатов работы устройства, то есть значений его выходных портов, с эталонными значениями. Эталонные значения, в свою очередь, могут быть сформированы также разными способами.

Первый способ – это получить данные значения извне, например считать из файла. Здесь предполагается, что эталонные значения были сформированы каким-либо образом ранее, например созданы самим разработчиком тестируемого устройства.

Второй способ заключается в формировании эталонных значений непосредственно самим тестовым модулем с помощью конструкций языка VHDL.

Фактически в этом случае создается модель поведения тестируемого устройства в целом или какой-либо его части в отдельности.

Третий способ заключается в использовании эталонного объекта для получения эталонных значений. В этом случае кроме экземпляра тестируемого объекта дополнительно создается экземпляр эталонного объекта. При этом входные векторы тестовых воздействий подаются одновременно как на тестируемый, так и на эталонный объекты. В результате моделирования на выходе эталонного объекта формируются значения эталона, с которыми
можно сравнивать результаты работы тестируемого объекта. В качестве эталонного объекта часто используют поведенческое описание разрабатываемого устройства, которое редко бывает синтезируемым, зато его можно быстро разработать и отладить.

3.2 Работа с файлами в языке VHDL

Для работы с файлами в VHDL можно воспользоваться двумя пакетами.

1. Пакет std.textio предоставляет функции для работы с файлами с использованием стандартных типов данных: `bit`, `bitvector`, `boolean`, `character`, `integer`, `real`, `string`, `time`.

2. Пакет ieee.std_logic_textio предоставляет функции для работы с файлами с использованием типов данных, определенных пакетом std_logic_1164: `std_ulogic`, `std_ulogic_vector`, `std_logic`, `std_logic_vector`.

Работа с текстовыми файлами выполняется в два этапа. При этом используются два типа функций:

1) readline или writeline;

2) read или write.

Какая конкретно функция должна использоваться, зависит от типа операций работы с файлами: чтение или запись.

Функции readline и writeline работают непосредственно с файлами и строками в них. Так функция readline выполняет чтение строки из файла, а функция writeline – запись строки в файл. Эти функции определены в пакете std.textio и там же описан тип данных, определяющий считываемые или записываемые строки, – тип line. Функции read и write работают со строками и позволяют считать (read) из строки или записать (write) в строку значения, определенные различными типами данных. Ниже приведен пример использования вышеперечисленных функций для работы с файлами.

```
architecture work_file_example of work_file_example is
 file test_file : text;
begin
 process
 variable file_status : file_open_status;
 variable current_line: line;
 variable test_value : std_logic_vector(7 downto 0);
 begin file_open(file_status,test_file,"test_file");
 read_line(test_file,current_line);
 read(current_line,test_value);
 end process;
end  work_file_example;
```

В этом примере выполняется считывание значения типа std_logic_vector в переменную test_value из файла test_file. Стоит отметить, что это лишь один из вариантов работы с файлами и вышеперечисленные функции не единственные, которые можно использовать для этого

4 ВЫПОЛНЕНИЕ РАБОТЫ

Ниже представлен текст программы для 1-го варианта:

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

entity priority_encoder is
    Port (
        inputs : in std_logic_vector(8 downto 0);
        outputs : out std_logic_vector(3 downto 0)
    );
end  priority_encoder;

architecture Behavioral of priority_encoder is
begin
    process(inputs)
    begin
        -- Инициализация выходов
        outputs <= "0000";

        -- Приоритетный кодировщик
        if inputs(8) = '1' then
            outputs <= "1001"; -- 9
        elsif inputs(7) = '1' then
            outputs <= "1000"; -- 8
        elsif inputs(6) = '1' then
            outputs <= "0111";  -- 7
```

```
        elsif inputs(5) = '1' then
            outputs <= "0110";  -- 6
        elsif inputs(4) = '1' then
            outputs <= "0101";  -- 5
        elsif inputs(3) = '1' then
            outputs <= "0100";  -- 4
        elsif inputs(2) = '1' then
            outputs <= "0011";  -- 3
        elsif inputs(1) = '1' then
            outputs <= "0010";  -- 2
        elsif inputs(0) = '1' then
            outputs <= "0001";  -- 1
        else outputs <= "0000"; -- 0
        end if;
    end process;
end Behavioral;
```

Testbench для проверки правильности работы:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use  IEEE.STD_LOGIC_UNSIGNED.ALL;
use std.textio.all;
use  IEEE.std_logic_textio.all;

entity testbench is
end testbench;

architecture behavior of testbench is

    signal input_lines : std_logic_vector(8 downto 0);
    signal output_lines : std_logic_vector(3 downto 0);
    signal expected_output : std_logic_vector(3 downto 0);

    component bcd_priority_encoder
        port (
            input_lines : in std_logic_vector(8 downto 0);
            output_lines : out std_logic_vector(3 downto 0)
        );
    end component;

    -- Объявление файлового типа
    file file_input : text; -- Правильное объявление файла

begin

    uut: bcd_priority_encoder port map (
        input_lines => input_lines,
        output_lines => output_lines
    );

    process
        variable line_input : line;
        variable temp_input : std_logic_vector(8 downto 0);
        variable temp_expected : std_logic_vector(3 downto 0);
        variable temp_output : std_logic_vector(3 downto 0);
    begin
        -- Открытие файла для чтения
        file_open(file_input,  "input_data.txt",  read_mode);

        while not endfile(file_input) loop
            readline(file_input, line_input);
```

```
            read(line_input, temp_input);
            read(line_input, temp_expected);
            read(line_input, temp_output);


            input_lines <= temp_input;
            expected_output <= temp_expected;
            output_lines <= temp_output;

            wait for 10 ns; -- Задержка для симуляции

            assert output_lines = expected_output
            report "Ошибка: выходные данные не совпадают с ожидаемыми"
            severity error;
        end loop;

        file_close(file_input);

        wait;
    end process;

end behavior;
```

На рисунке 4.1 показана временная диаграмма, полученная в результате выполнения программы первым способом.

Рисунок 4.1 – Временная диаграмма №1

Ниже представлен текст программы для 2-го варианта:

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

entity priority_encoder is
    Port (
        inputs : in std_logic_vector(8 downto 0);
        outputs : out std_logic_vector(3 downto 0)
```

```
    );
end  priority_encoder;

architecture Behavioral of priority_encoder is
begin
    process(inputs)
    begin
        -- Инициализация выходов
        outputs <= "0000";  -- Значение по умолчанию

        -- Приоритетный кодировщик с использованием операторов
        case inputs is
            when "100000000" =>  -- 9
                outputs <= "1001";
            when "010000000" =>  -- 8
                outputs <= "1000";
            when "001000000" =>  -- 7
                outputs <= "0111";
            when "000100000" =>  -- 6
                outputs <= "0110";
            when "000010000" =>  -- 5
                outputs <= "0101";
            when "000001000" =>  -- 4
                outputs <= "0100";
            when "000000100" =>  -- 3
                outputs <= "0011";
            when "000000010" =>  -- 2
                outputs <= "0010";
            when "000000001" =>  -- 1
                outputs <= "0001";
            when others => -- Все нули
                outputs <= "0000";
        end case;
    end process;
end Behavioral;
```

Testbench для проверки правильности работы:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use  IEEE.STD_LOGIC_UNSIGNED.ALL;
use std.textio.all;
use  IEEE.std_logic_textio.all;

entity testbench is
end testbench;

architecture behavior of testbench is

    signal input_lines : std_logic_vector(8 downto 0);
    signal output_lines : std_logic_vector(3 downto 0);
    signal expected_output : std_logic_vector(3 downto 0);

    component bcd_priority_encoder
        port (
            input_lines : in std_logic_vector(8 downto 0);
            output_lines : out std_logic_vector(3 downto 0)
        );
    end component;

    -- Объявление файлового типа
    file file_input : text;  -- Правильное объявление файла
```

```
begin

    uut: bcd_priority_encoder port map (
        input_lines => input_lines,
        output_lines => output_lines
    );

    process
        variable line_input : line;
        variable temp_input : std_logic_vector(8 downto 0);
        variable temp_expected : std_logic_vector(3 downto 0);
        variable temp_output : std_logic_vector(3 downto 0);
    begin
        -- Открытие файла для чтения
        file_open(file_input,  "input_data.txt",  read_mode);

        while not endfile(file_input) loop
            readline(file_input, line_input);
            read(line_input, temp_input);
            read(line_input, temp_expected);
            read(line_input, temp_output);


            input_lines <= temp_input;
            expected_output <= temp_expected;
            output_lines <= temp_output;

            wait for 10 ns; -- Задержка для симуляции

            assert output_lines = expected_output
            report "Ошибка: выходные данные не совпадают с ожидаемыми"
            severity error;
        end loop;

        file_close(file_input);

        wait;
    end process;

end behavior;
```

На рисунке 4.2 показана временная диаграмма, полученная в результате выполнения программы вторым способом.

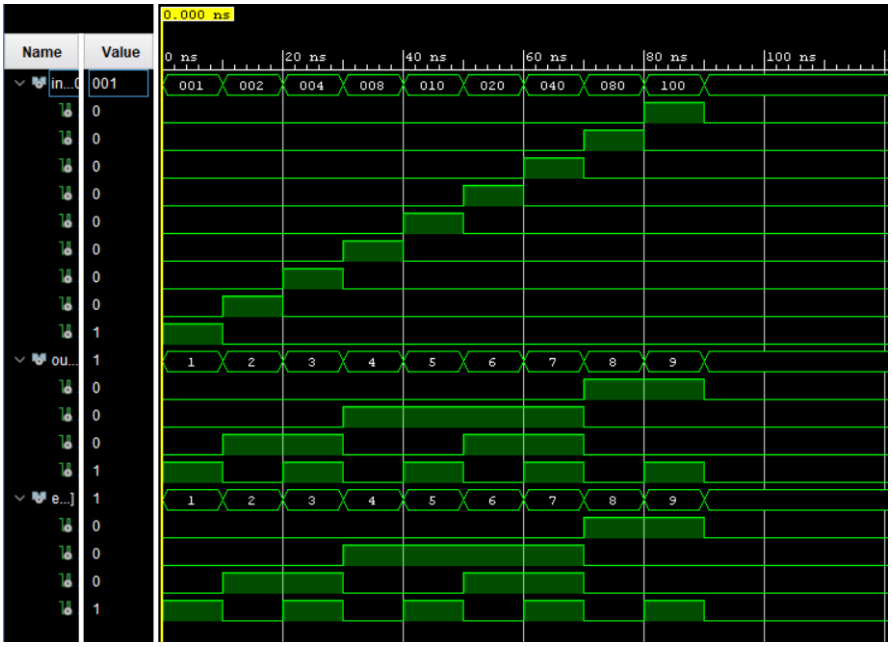

Рисунок 4.2 – Временная диаграмма №2

5 ВЫВОДЫ

В ходе выполнения лабораторной работы были получены навыки разработки тестовых модулей на языке VHDL.