Министерство образования Республики Беларусь

Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Автоматизация проектирования вычислительных машин и систем

ОТЧЁТ
к лабораторной работе № 1
на тему

ПРОЕКТИРОВАНИЕ КОМБИНАЦИОННЫХ УСТРОЙСТВ НА ЯЗЫКЕ VHDL В СРЕДЕ VIVADO. ВАРИАНТ № 12

Студенты: гр. 050502
Т.С. Жук
А.В. Крачковский
Е.В. Кравченко

Проверил: В.В. Шеменков

Минск 2023

СОДЕРЖАНИЕ

1 ЦЕЛЬ РАБОТЫ

Ознакомиться с САПР Vivado, изучить базовый синтаксис языка VHDL и выполнить проектирование заданной комбинационной схемы на языке VHDL.

2 ИСХОДНЫЕ ДАННЫЕ К РАБОТЕ

Для выполнения работы используем САПР Vivado 2018.2 и редактор кода Visual Studio Code с расширением (extension) Modern VHDL для подсветки синтаксиса языка VHDL.

В соответствии с полученным вариантом задания необходимо спроектировать устройство на языке VHDL. При этом устройство должно быть реализовано пятью различными способами с использованием:

1. логических операторов и параллельного оператора безусловного присваивания;
2. параллельного оператора условного присваивания;
3. параллельного оператора селективного присваивания (присваивания по выбору) (select);
4. последовательного оператора условия (if);
5. последовательного оператора выбора (case).

После описания заданного устройства всеми пятью способами необходимо создать тестовый модуль для моделирования его работы. При этом моделирование должно выполняться для всех способов реализации.

Для создания тестового модуля необходимо воспользоваться графическим редактором тестовых воздействий. При этом тестовые воздействия должны быть полными. Для заданных комбинационных устройств это означает полный перебор входных значений. После создания тестового модуля необходимо выполнить поведенческое (Behavioural Simulation) моделирование работы устройства для всех способов его описания.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Разработать проектный модуль ‘entity’ изученной комбинационной схемы на языке VHDL;
2. Разработать проектные модули ‘architecture’ для разрабатываемой комбинационной схемы на языке VHDL;
3. Разработать проектный модуль поведенческого моделирования (Behavioural Simulation) работы устройства для всех способов его описания на языке VHDL;
4. Провести поведенческое моделирование и проверить корректность работы реализованной комбинационной схемы.

3 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Согласно заданному варианту необходимо разработать селектор/мультиплексор для двух четырёхразрядных шин (QUADRUPLE 2-LINE TO 1-LINE DATA SELECTORS/MULTIPLEXERS) согласно логической диаграмме представленной на рисунке ниже.

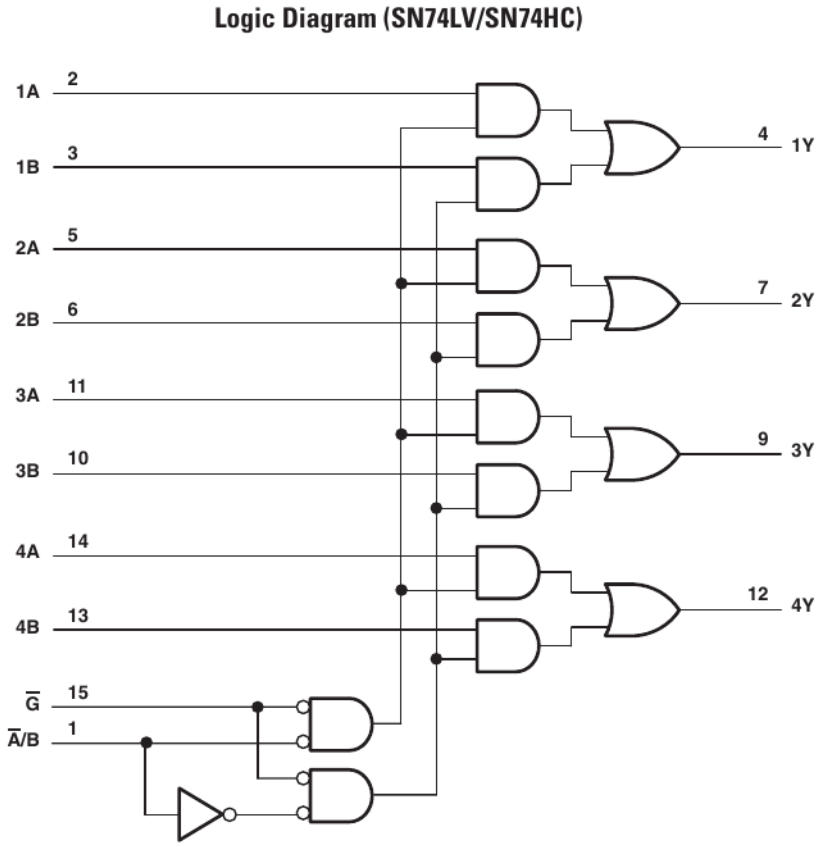

Рисунок 3.1 – Логическая диаграмма заданной комбинационной схемы

Функциональная таблица разрабатываемой комбинационной схемы представлена на рисунке ниже.

FUNCTION TABLE (SN74)

INPUTS				OUTPUT
STROBE	SELECT	A	B	
H	X	X	X	L
L	L	L	X	L
L	L	H	X	H
L	H	X	L	L
L	H	X	H	H

Рисунок 3.2 – Функциональная таблица разрабатываемой комбинационной схемы

4 ВЫПОЛНЕНИЕ РАБОТЫ

Для выполнения лабораторной работы создаём проект с помощью встроенного менеджера проектов в САПР Vivado. С помощью графического интерфейса в режиме диалоговых окон создаём и добавляем в проект исходный файл с описанием заданной комбинационной схемы на языке VHDL ‘SN74.vhd’, а также исходный файл с описанием модуля поведенческого моделирования (Behavioural Simulation) работы устройства для всех способов его описания на языке VHDL ‘Behavioural.vhd’.

Дальнейшее внесение изменений в исходные файлы осуществляем в редакторе кода Visual Studio Code, который предоставляет удобный и привычный интерфейс и богатые возможности по редактированию кода.

Исходный текст разработанных модулей представлен в приложениях А и Б.

4.1 Разработка проектного модуля ‘entity’ изученной комбинационной схемы на языке VHDL

Следуя правилам хорошего тона для достижения гибкости и возможности повторного использования кода в ‘entity SN74’ добавляем параметр ‘generic (WIDTH: positive)’, который будет отвечать за разрядность входных и выходной шин данных.

Добавляем устройству порты ввода:

– порт A типа std_logic_vector(WIDTH-1 downto 0);
– порт B типа std_logic_vector(WIDTH-1 downto 0);
– порт D типа std_logic;
– порт S типа std_logic;

И порты вывода:

– порт Y типа std_logic_vector(WIDTH-1 downto 0)

4.2 Разработка проектных модулей ‘architecture’ для разрабатываемой комбинационной схемы на языке VHDL

После описания того, как выглядит комбинационное устройство снаружи, в модуле ‘entity’ приступаем к разработке модулей ‘architecture’ – того, как это устройство работает изнутри.

Согласно заданию нам необходимо реализовать работу устройства пятью способами, перечисленными в исходных данных к работе.

Отметим, что первый способ представляется нам самым сложным из всех, так как использование синтаксической конструкции ‘(others => X)’ в языке VHDL ограничено и допустимо только там, где не может возникнуть сомнений в размерности результата, например: ‘Y <= (others => ’0’)’.

Изначально эта проблема была решена грубым способом: через вызов функции-обёртки для этой конструкции с ”принудительным” указанием размерности результата:

```
function fill_vector(X: std_logic) return std_logic_vector is
    variable tmp: std_logic_vector(WIDTH-1 downto 0);
begin
    tmp := (others => X);
    return tmp;
end function;
```

Хотя этот способ и справляется со своим назначением, но является ”топорным” и вероятнее всего привнесёт дополнительные проблемы при усложнении внутренней логики. Иными словами, этот подход не следует лучшим практикам и является ‘попыткой программировать на языке описания аппаратуры интегральных схем VHDL’. Правильным в данной ситуации будет введение внутренних сигналов, которые необходимы для работы устройства:

```
architecture architecture1 of SN74 is
    -- Internal signal
    signal Enable:  std_logic_vector(WIDTH-1 downto 0) := (others =>
       '0');
    signal SelectB: std_logic_vector(WIDTH-1 downto 0) := (others =>
       '0');
begin
    Enable  <= (others => not D);
    SelectB <= (others => S);
    Y       <= Enable and ((A and not SelectB) or (B and SelectB));
end architecture;
```

Реализация остальных архитектур также использует внутренний сигнал:

```
...
  signal DS: std_logic_vector(1 downto 0) := (others => '0');
...
  DS <= D & S;
...
```

Это позволяет нам обрабатывать оба управляющих сигнала одними и теми же синтаксическими конструкциями, не пытаясь реализовать так называемую ”вложенность” операторов (например, switch-case, with-select).

4.3 Разработка проектного модуля поведенческого моделирования (Behavioural Simulation) работы устройства для всех способов его описания на языке VHDL

Несмотря на то, что симуляция не является синтезируемым устройством, мы считаем необходимым ввести некоторые параметры и константы, вычисляемые на основе этих параметров, в ‘entity Behavioural’ для последующего использования:

```
generic (DL_WIDTH: positive := 4);
constant ITERATIONS: positive := 2 ** (2 * DL_WIDTH + 2);
constant DELAY:      time     := 1000000 ps / ITERATIONS;
constant MAX_VALUE:  positive := 2 ** DL_WIDTH - 1;
```

Таким образом, всё время моделирования (1000000 ps) равномерно бьётся на равные интервалы времени.

Заметим, что модуль разработан для одновременного тестирования всех реализаций устройства, что позволяет моментально замечать несоответствия в работе различных архитектур, если таковые имеются. Для этого выходной сигнал ‘Y’ дублируем 5 раз, переименовав в ‘Y_aN’, где N – номер архитектуры, а также используем ‘generic map’ и ‘port map’, чтобы связать порты с сигналами моделирования.

Непосредственно алгоритм поведенческого моделирования, со сменой входных сигналов выглядит следующим образом:

```
for int_d in 0 to 1 loop
    for int_s in 0 to 1 loop
        for int_a in 0 to MAX_VALUE loop
            for int_b in 0 to MAX_VALUE loop
                D <= to_std_logic(int_d);
                S <= to_std_logic(int_s);
                A <= to_std_logic_vector(int_a);
                B <= to_std_logic_vector(int_b);
                wait for DELAY;
            end loop;
        end loop;
    end loop;
end loop;
```

Как можно заметить выше, в основном теле цикла мы используем две функции: ‘to_std_logic’ и ‘to_std_logic_vector’, задача которых конвертировать тип ‘integer’ в тип ‘std_logic’ и ‘std_logic_vector’ соответственно. Так как это устройство не синтезируемо использование функций не является чем-то крамольным и вполне допустимо для обеспечения читабельности кода.

4.4 Проведение поведенческого моделирования и проверка корректности работы реализованной комбинационной схемы

Результаты проведения поведенческого моделирования представлены на рисунках ниже.

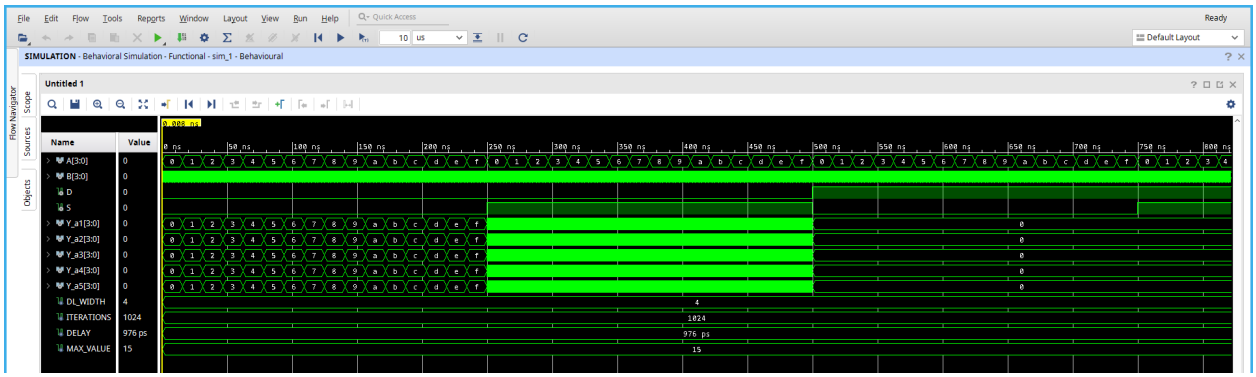

Рисунок 4.1 – Поведенческое моделирование реализованной комбинационной схемы. 800 ns моделирования

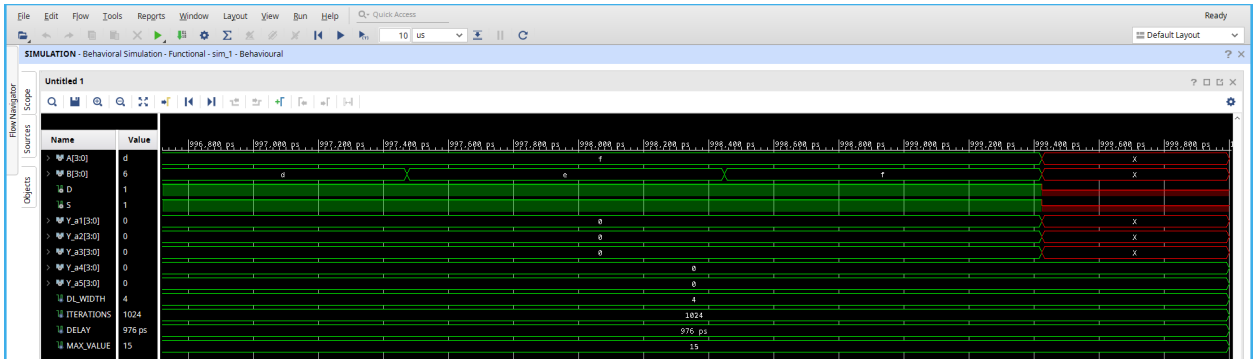

Рисунок 4.2 – Поведенческое моделирование реализованной комбинационной схемы. Обработка некорректных входных сигналов

Так как результаты проведения моделирования не разнятся от архитектуры к архитектуре и соответствуют таблице истинности, приведённой в разделе ‘Теоретические сведения’, заключаем, что реализованная комбинационная схема работает корректно.

5 ВЫВОДЫ

В процессе выполнения работы мы ознакомились с САПР Vivado, изучили базовый синтаксис языка VHDL и заданную комбинационную схему и выполнили её проектирование на языке VHDL.

Полученные знания были применены для решения задач, возникших в ходе работы:

1. Разработки проектного модуля ‘entity’ изученной комбинационной схемы на языке VHDL;
2. Разработки проектных модулей ‘architecture’ для разрабатываемой комбинационной схемы на языке VHDL;
3. Разработки проектного модуля поведенческого моделирования (Behavioural Simulation) работы устройства для всех способов его описания на языке VHDL;
4. Проведения поведенческого моделирования и проверки корректности работы реализованной комбинационной схемы.

ПРИЛОЖЕНИЕ А

(*обязательное)*

Исходный текст модуля SN74

Файл ‘SN74.vhd’ содержит следующий код:

```
1  ----------------------------------------------------------------------------------
2  -- Company:
3  -- Engineer:
4  --
5  -- Create Date: 16.09.2023 17:37:05
6  -- Design Name:
7  -- Module Name: SN74 - architecture1..5
8  -- Project Name:
9  -- Target Devices:
10 -- Tool Versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
19 ----------------------------------------------------------------------------------


20
21
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24
25 -- Category: QUADRUPLE 2-LINE TO 1-LINE DATA SELECTORS/MULTIPLEXERS
26 -- Implementation: SN74LV/SN74HC
27 entity SN74 is
28     generic (
29         -- Width of quadruple lines
30         WIDTH: positive
31     );
32     port (
33         -- Input quadruple lines
34         A: in  std_logic_vector(WIDTH-1 downto 0);
35         B: in  std_logic_vector(WIDTH-1 downto 0);
36         -- Disable/Enable (not G)
37         D: in  std_logic;
38         -- Select (not A/B)
39         S: in  std_logic;
40         -- Output quadruple line
41         Y: out std_logic_vector(WIDTH-1 downto 0)
42     );
43 end entity;
44
45 -- Function table (SN74)
46 -- |---|---|---|---||---|
47 -- | D | S | A | B || Y |
48 -- |---|---|---|---||---|
49 -- | H | X | X | X || L |
50 -- | L | L | L | X || L |
```

```
51  -- | L | L | H | X || H |
52  -- | L | H | X | L || L |
53  -- | L | H | X | H || H |
54  -- |---|---|---|---||---|
55
56  -- Implementation 1
57  -- Parallel unconditional assignment is required (+ logical operators)
58  architecture architecture1 of SN74 is
59      -- Internal signal
60      signal Enable:  std_logic_vector(WIDTH-1 downto 0) := (others =>
            '0');
61      signal SelectB: std_logic_vector(WIDTH-1 downto 0) := (others =>
            '0');
62  begin
63      Enable  <= (others => not D);
64      SelectB <= (others => S);
65      Y       <= Enable and ((A and not SelectB) or (B and SelectB));
66  end architecture;
67
68  -- Implementation 2
69  -- Parallel conditional assignment is required
70  architecture architecture2 of SN74 is
71      -- Internal signals
72      signal DS: std_logic_vector(1 downto 0) := (others => '0');
73  begin
74      DS <= D & S;
75      Y  <= A               when DS = "00" else
76            B               when DS = "01" else
77            (others => '0') when DS = "10" else
78            (others => '0') when DS = "11" else
79            (others => 'X');
80  end architecture;
81
82  -- Implementation 3
83  -- Parallel selected assignment is required
84  architecture architecture3 of SN74 is
85      -- Internal signals
86      signal DS: std_logic_vector(1 downto 0) := (others => '0');
87  begin
88      DS <= D & S;
89      with DS select
90          Y <= A               when "00",
91               B               when "01",
92               (others => '0') when "10",
93               (others => '0') when "11",
94               (others => 'X') when others;
95  end architecture;
96
97  -- Implementation 4
98  -- Sequential conditional is required (if)
99  architecture architecture4 of SN74 is
100     -- Internal signals
101     signal DS: std_logic_vector(1 downto 0) := (others => '0');
102 begin
```

```
103     DS <= D & S;
104     process (A, B, D, S) begin
105         if    DS = ”00” then Y <= A;
106         elsif DS = ”01” then Y <= B;
107         elsif DS = ”10” then Y <= (others => ’0’);
108         elsif DS = ”11” then Y <= (others => ’0’);
109         else                 Y <= (others => ’X’);
110         end if;
111     end process;
112 end architecture;
113 
114 -- Implementation 5
115 -- Sequential selection operator is required (case)
116 architecture architecture5 of SN74 is
117     -- Internal signals
118     signal DS: std_logic_vector(1 downto 0) := (others => ’0’);
119 begin
120     DS <= D & S;
121     process (A, B, D, S) begin
122         case DS is
123             when ”00”   => Y <= A;
124             when ”01”   => Y <= B;
125             when ”10”   => Y <= (others => ’0’);
126             when ”11”   => Y <= (others => ’0’);
127             when others => Y <= (others => ’X’);
128         end case;
129     end process;
130 end architecture;
```

ПРИЛОЖЕНИЕ Б

(*обязательное)*

Исходный текст модуля Behavioural

Файл ‘Behavioural.vhd’ содержит следующий код:

```
 1 ----------------------------------------------------------------------------------
 2 -- Company:
 3 -- Engineer:
 4 --
 5 -- Create Date: 16.09.2023 17:37:36
 6 -- Design Name:
 7 -- Module Name: Behavioural - simulation
 8 -- Project Name:
 9 -- Target Devices:
10 -- Tool Versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
19 ----------------------------------------------------------------------------------
20
21
22 library IEEE;
23 use IEEE.NUMERIC_STD.ALL;
24 use IEEE.STD_LOGIC_1164.ALL;
25
26 entity Behavioural is
27     generic (
28         -- Data line width (for units under test)
29         -- Max value is 8!!! Higher values result it 0ps DELAY
30         DL_WIDTH: positive := 4
31     );
32     constant ITERATIONS: positive := 2 ** (2 * DL_WIDTH + 2);
33     constant DELAY:      time     := 1000000 ps / ITERATIONS;
34     constant MAX_VALUE:  positive := 2 ** DL_WIDTH - 1;
35 end entity;
36
37 architecture simulation of Behavioural is
38     -- Signals
39     signal A:    std_logic_vector(DL_WIDTH-1 downto 0) := (others =>
          '0');
40     signal B:    std_logic_vector(DL_WIDTH-1 downto 0) := (others =>
          '0');
41     signal D:    std_logic                             := '0';
42     signal S:    std_logic                             := '0';
43     signal Y_a1: std_logic_vector(DL_WIDTH-1 downto 0) := (others =>
          '0');
44     signal Y_a2: std_logic_vector(DL_WIDTH-1 downto 0) := (others =>
          '0');
45     signal Y_a3: std_logic_vector(DL_WIDTH-1 downto 0) := (others =>
          '0');
```

```
46      signal Y_a4: std_logic_vector(DL_WIDTH-1 downto 0) := (others =>
           '0');
47      signal Y_a5: std_logic_vector(DL_WIDTH-1 downto 0) := (others =>
           '0');
48      -- Utility functions
49      -- Convert value `V` to std_logic
50      function to_std_logic(V: integer) return std_logic is begin
51          if    V = 0 then return '0';
52          elsif V = 1 then return '1';
53          else             return 'X';
54          end if;
55      end function;
56      -- Convert value `V` to std_logic_vector
57      function to_std_logic_vector(V: integer) return std_logic_vector is
            begin
58          return std_logic_vector(to_unsigned(V, DL_WIDTH));
59      end function;
60  begin
61      -- Instantiate units under test (UUT-s)
62      uut1: entity work.SN74(architecture1) -- architecture1 :: passed
63          generic map (WIDTH => DL_WIDTH)
64          port map (A => A, B => B, D => D, S => S, Y => Y_a1);
65      uut2: entity work.SN74(architecture2) -- architecture2 :: passed
66          generic map (WIDTH => DL_WIDTH)
67          port map (A => A, B => B, D => D, S => S, Y => Y_a2);
68      uut3: entity work.SN74(architecture3) -- architecture3 :: passed
69          generic map (WIDTH => DL_WIDTH)
70          port map (A => A, B => B, D => D, S => S, Y => Y_a3);
71      uut4: entity work.SN74(architecture4) -- architecture4 :: passed
72          generic map (WIDTH => DL_WIDTH)
73          port map (A => A, B => B, D => D, S => S, Y => Y_a4);
74      uut5: entity work.SN74(architecture5) -- architecture5 :: passed
75          generic map (WIDTH => DL_WIDTH)
76          port map (A => A, B => B, D => D, S => S, Y => Y_a5);
77      -- Simulation process
78      process begin
79          for int_d in 0 to 1 loop
80              for int_s in 0 to 1 loop
81                  for int_a in 0 to MAX_VALUE loop
82                      for int_b in 0 to MAX_VALUE loop
83                          D <= to_std_logic(int_d);
84                          S <= to_std_logic(int_s);
85                          A <= to_std_logic_vector(int_a);
86                          B <= to_std_logic_vector(int_b);
87                          wait for DELAY;
88                      end loop;
89                  end loop;
90              end loop;
91          end loop;
92          D <= 'X';
93          S <= 'X';
94          A <= (others => 'X');
95          B <= (others => 'X');
96          wait;
```

```
97      end process;
98  end architecture;
```