АВТОНОМНАЯ НЕКОММЕРЧЕСКАЯ ОРГАНИЗАЦИЯ ВЫСШЕГО ОБРАЗОВАНИЯ
«РУССКИЙ УНИВЕРСИТЕТ МЕТАТЕХНОЛОГИЙ»

Кафедра ПС

Отчёт
по лабораторной работе №8
«Разработка устройств обработки сигналов на программных логических интегральных схемах»
по дисциплине «Основы микропроцессорных систем»

Выполнил ст. гр. ПС-11
Поскряков Н.В.
Проверил: Электроник
кафедры ПС
Костицына К.О.

Йошкар-Ола
2025

Теоретические сведения

ПЛИС (программируемая логическая интегральная схема) - это электронное устройство, состоящие из большого количества логических элементов, которое можно программируемым образом настроить на выполнение нужной функции. Использование ПЛИС позволяет существенно ускорить разработку и производство электронной аппаратуры, поскольку значительная часть необходимой логики может быть реализована на одной микросхеме.

Программа может состоять из трёх основных файлов. Схематик, VHDL и файл конфигурации - INV.

Практическая часть

Задание 1

Цель – изучить принцип работы типовых комбинационных схем: шифратора, дешифратора, мультиплексора, демультиплексора. Заполнить таблицы истинности по каждой схеме

Ход работы

1. **Дешифратор 3x8**

Таблица 1 - Таблица истинности дешифратора

D0	D1	D2	S0	S1	S2	S3	S4	S5	S6	S7
0	**0**	**0**	1	0	0	0	0	0	0	0
0	**0**	**1**	0	1	0	0	0	0	0	0
0	**1**	**0**	0	0	1	0	0	0	0	0
0	**1**	**1**	0	0	0	1	0	0	0	0
1	**0**	**0**	0	0	0	0	1	0	0	0
1	**0**	**1**	0	0	0	0	0	1	0	0
1	**1**	**0**	0	0	0	0	0	0	1	0
1	**1**	**1**	0	0	0	0	0	0	0	1

- Файл VHDL

```
39  begin
40     process(D) --вызывается при изменении сигнала D
41     begin
42        case(D) is
43           when "000" => SEG <= "10000000";
44           when "001" => SEG <= "01000000";
45           when "010" => SEG <= "00100000";
46           when "011" => SEG <= "00010000";
47           when "100" => SEG <= "00001000";
48           when "101" => SEG <= "00000100";
49           when "110" => SEG <= "00000010";
50           when "111" => SEG <= "00000001";
51           when others => SEG <= "00000000";
52        end case;
53     end process;
54
55  end Behavioral;
```

Рисунок 1 - VHDL модуль дешифратора

- Подключение в схематике

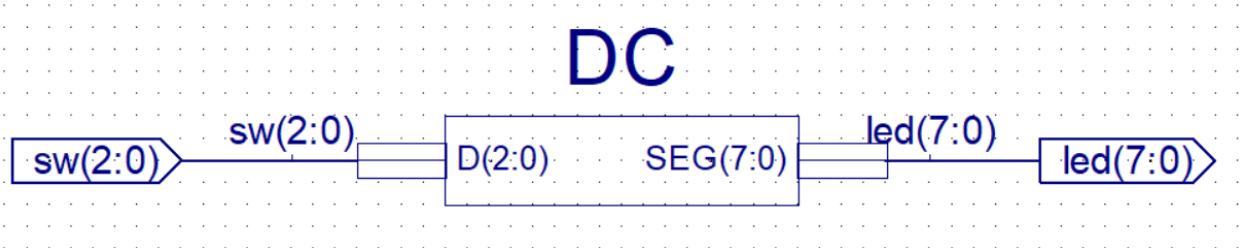

Рисунок 2 - Подключение Дешифратора в схеме

- Конфигурация выходов/входов

```
1  NET "led<0>" LOC="T11";
2  NET "led<1>" LOC="R11";
3  NET "led<2>" LOC="N11";
4  NET "led<3>" LOC="M11";
5  NET "led<4>" LOC="V15";
6  NET "led<5>" LOC="U15";
7  NET "led<6>" LOC="V16";
8  NET "led<7>" LOC="U16";
9
10 NET "sw<2>" LOC="V9";
11 NET "sw<1>" LOC="T9";
12 NET "sw<0>" LOC="T10";
```

Рисунок 3 - Конфигурационный файл Дешифратора

- Модульные тесты для Дешифратора

```
68      D_proc :process
69      begin
70          D <= "000";
71          wait for 100 ns;
72          D <= "001";
73          wait for 100 ns;
74          D <= "010";
75          wait for 100 ns;
76          D <= "011";
77          wait for 100 ns;
78          D <= "100";
79          wait for 100 ns;
80          D <= "101";
81          wait for 100 ns;
82          D <= "110";
83          wait for 100 ns;
84          D <= "111";
85          wait for 100 ns;
86      end process;
```

Рисунок 4 - Модульный тест для Дешифратора

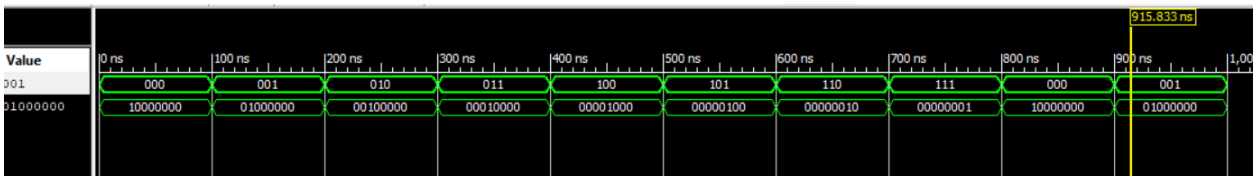

Рисунок 5 - График модульного теста Дешифратора

- Тесты на плате

Рисунок 6 - Результаты тестирования Дешифратора на плате Spartan-6

Вывод: В результате проделанной работы, была составлена таблица истинности Дешифратора. Реализованы VHDL программа устройства и модульные тесты. Верхний меандр отражает входные значения в Дешифратор, нижние выходные значения.

2. Шифратор 8x3

Таблица 2 - Таблица истинности Шифратора

X7	X6	X5	X4	X3	X2	X1	X0	Y2	Y1	Y0
1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0	0	1
0	**1**	**0**	**0**	**0**	**0**	**0**	**0**	0	1	0
0	**0**	**1**	**0**	**0**	**0**	**0**	**0**	0	1	1
0	**0**	**0**	**1**	**0**	**0**	**0**	**0**	1	0	0
0	**0**	**0**	**0**	**1**	**0**	**0**	**0**	1	0	1
0	**0**	**0**	**0**	**0**	**1**	**0**	**0**	1	1	0
0	**0**	**0**	**0**	**0**	**0**	**1**	**0**	1	1	1
0	**0**	**0**	**0**	**0**	**0**	**0**	**1**	1	1	1

- Файл VHDL

```
12    process(x)
13    begin
14       case(x) is
15          when "10000000" => y <= "000";
16          when "01000000" => y <= "001";
17          when "00100000" => y <= "010";
18          when "00010000" => y <= "011";
19          when "00001000" => y <= "100";
20          when "00000100" => y <= "101";
21          when "00000010" => y <= "110";
22          when "00000001" => y <= "111";
23          when others => y <= "000";
24
25       end case;
26    end process;
```

Рисунок 7 - Реализация Шифратора на VHDL

- Подключение в схематике

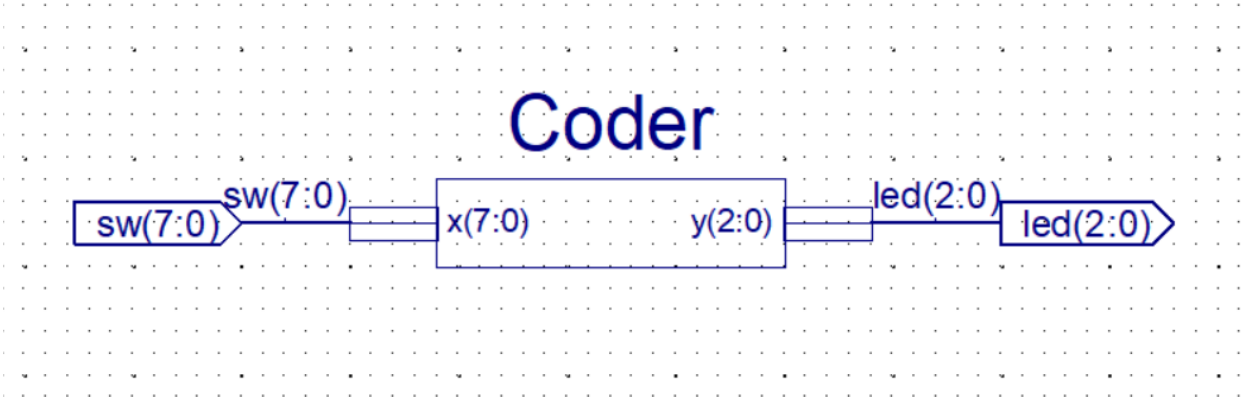

Рисунок 8 - Подключение Шифратора в схематике

- Конфигурация входов/выходов

```
1   NET "sw<7>" LOC="T10";
2   NET "sw<6>" LOC="T9";
3   NET "sw<5>" LOC="V9";
4   NET "sw<4>" LOC="M8";
5   NET "sw<3>" LOC="N8";
6   NET "sw<2>" LOC="U8";
7   NET "sw<1>" LOC="V8";
8   NET "sw<0>" LOC="T5";
9
10  NET "led<2>" LOC="U15";
11  NET "led<1>" LOC="V16";
12  NET "led<0>" LOC="U16";
```

Рисунок 9 - Конфигурационный файл Шифратора

- Модульное тестирование

```
36    coder_proc: process
37    begin
38       x <= "10000000";
39       wait for 100 ns;
40       x <= "01000000";
41       wait for 100 ns;
42       x <= "00100000";
43       wait for 100 ns;
44       x <= "00010000";
45       wait for 100 ns;
46       x <= "00001000";
47       wait for 100 ns;
48       x <= "00000100";
49       wait for 100 ns;
50       x <= "00000010";
51       wait for 100 ns;
52       x <= "00000001";
53       wait for 100 ns;
54       x <= "00000000";
55       wait for 100 ns;
56    end process;
```

Рисунок 10 - Модульный тест для Шифратора

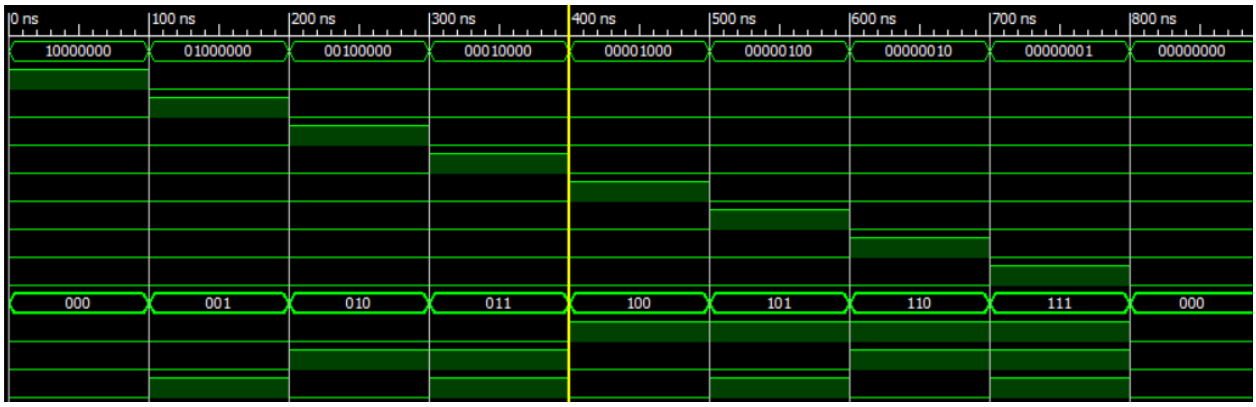

Рисунок 11 - График модульного теста Шифратора

- Тестирование на плате

Рисунок 12 - Результаты тестирования Шифратора на плате Spartan-6

Вывод: В результате проделанной работы, была составлена таблица истинности Шифратора. Реализованы VHDL программа устройства и модульные тесты. Верхний меандр отражает входные значения в Шифратор, нижние выходные значения. При тестировании был рассмотрен случай, когда на вход устройству подаётся фрейм данных не соответствующий таблице истинности. В таком случае Шифратор выдаёт значение “000”.

3. Мультиплексор

Таблица 3 - Таблица истинности Мультиплексера

X2	X1	X0	signal
0	**0**	**0**	Y0
0	**0**	**1**	Y1
0	**1**	**0**	Y2
0	**1**	**1**	Y3
1	**0**	**0**	Y4
1	**0**	**1**	Y5
1	**1**	**0**	Y6
1	**1**	**1**	Y7

- Файл VHDL

```
41  process(x, s)
42     begin
43        case(x) is
44           when "000" => y <= s(0);
45           when "001" => y <= s(1);
46           when "010" => y <= s(2);
47           when "011" => y <= s(3);
48           when "100" => y <= s(4);
49           when "101" => y <= s(5);
50           when "110" => y <= s(6);
51           when "111" => y <= s(7);
52           when others => y <= '0';
53        end case;
54     end process;
```

Рисунок 13 - Реализация Мультиплексора на VHDL

- Подключение в схематике

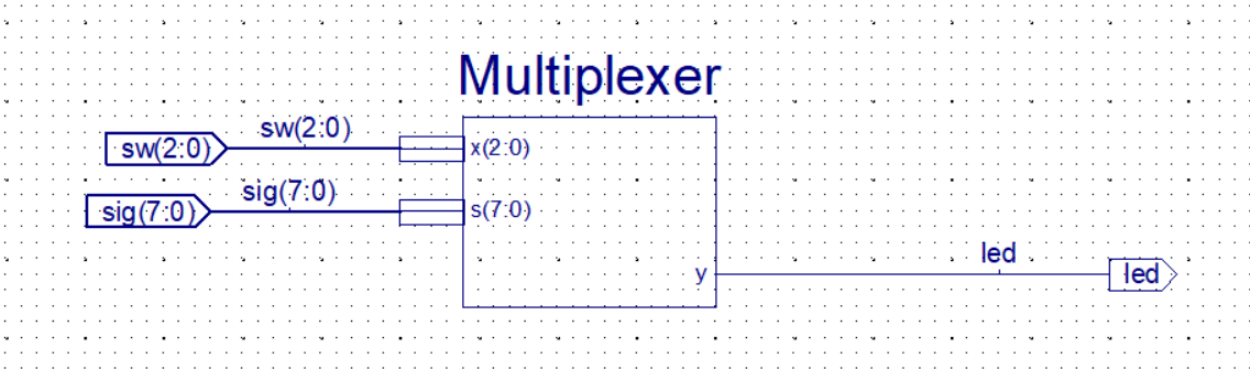

Рисунок 14 - Подключение Мультиплексора в схематике

- Конфигурация входов/выходов

```
1   NET "sw<2>" LOC="C4";
2   NET "sw<1>" LOC="B8";
3   NET "sw<0>" LOC="D9";
4
5   NET "led" LOC="U16";
6
7   NET "sig<7>" LOC="T5";
8   NET "sig<6>" LOC="V8";
9   NET "sig<5>" LOC="U8";
10  NET "sig<4>" LOC="N8";
11  NET "sig<3>" LOC="M8";
12  NET "sig<2>" LOC="V9";
13  NET "sig<1>" LOC="T9";
14  NET "sig<0>" LOC="T10";
```

Рисунок 15 - Конфигурационный файл Мультиплексора

- Модульное тестирование

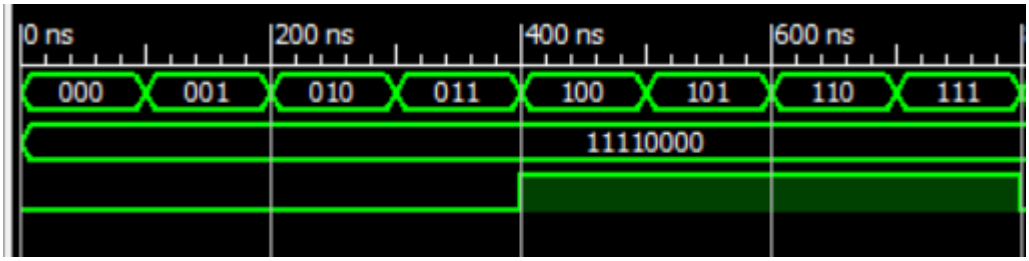

Рисунок 16 - График меандр тестирования Мультиплексора

- Тестирование на плате

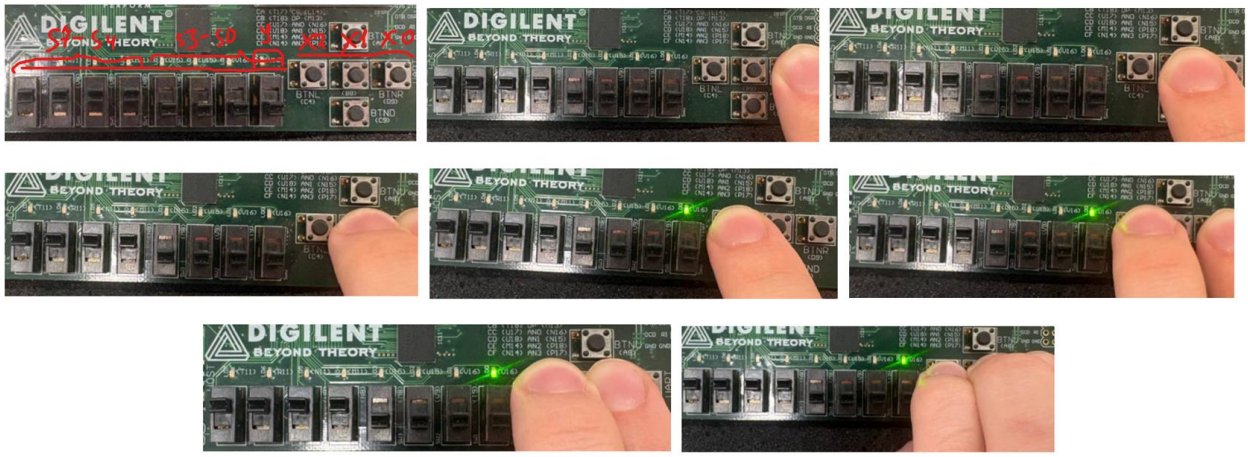

Рисунок 17 - Результаты тестирования Шифратора на плате Spartan-6

Вывод: В результате проделанной работы, была составлена таблица истинности Мультиплексора. Реализованы VHDL программа устройства и модульные тесты. Верхний меандр отражает управляющие входные значения Мультиплексора, ниже находится значения мультиплексирующих сигналов. В самом низу – значение выходного сигнала. Двоичный код управляющего сигнала обозначает номер разряда в шине мультиплексируемых сигналов, значит что управляющий 001 будет направлять на выход S1.

4. Демультиплексор

Таблица 4 - Таблица истинности Демультиплексора

X2	X1	X0	S7	S6	S5	S4	S3	S2	S1	S0
0	**0**	**0**	0	0	0	0	0	0	0	Y
0	**0**	**1**	0	0	0	0	0	0	Y	0
0	**1**	**0**	0	0	0	0	0	Y	0	0
0	**1**	**1**	0	0	0	0	Y	0	0	0
1	**0**	**0**	0	0	0	Y	0	0	0	0
1	**0**	**1**	0	0	Y	0	0	0	0	0
1	**1**	**0**	0	Y	0	0	0	0	0	0
1	**1**	**1**	Y	0	0	0	0	0	0	0

- Файл VHDL

```
41      process(X, Y)
42          begin
43              S <= "00000000";
44              if(Y = '1')then
45                  case(X) is
46                      when "000" => S(0) <= Y;
47                      when "001" => S(1) <= Y;
48                      when "010" => S(2) <= Y;
49                      when "011" => S(3) <= Y;
50                      when "100" => S(4) <= Y;
51                      when "101" => S(5) <= Y;
52                      when "110" => S(6) <= Y;
53                      when "111" => S(7) <= Y;
54                      when others => S <= "00000000";
55                  end case;
56              end if;
57          end process;
```

Рисунок 18 - Реализаяция Демультиплексора на VHDL

- Подключение в схематике

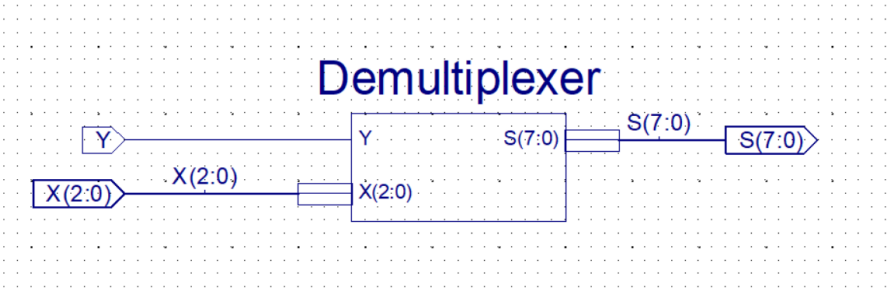

Рисунок 19 - Подключение Демультиплексора в схематике

- Модульное тестирование

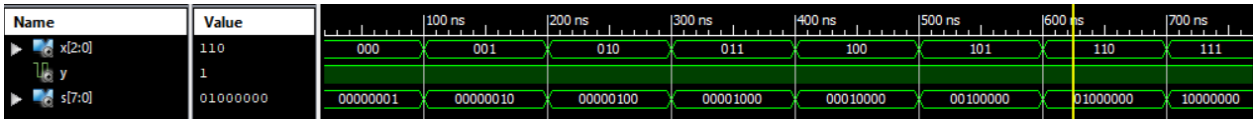

Рисунок 20 - График меандр тестирования Демультиплексора

- Тестирование на плате

Рисунок 21 - Результаты тестирования Демультиплексора на плате Spartan-6

Вывод: В результате проделанной работы, была составлена таблица истинности Демультиплексора. Реализованы VHDL программа устройства и модульные тесты. Управляющая шина данных ‘X’, выбирает в какой порт S демультиплексировать сигнал Y. Важно обратить внимание, что остальные порты S должны быть “обнулены”.

Задание 2

Цель – модифицировать дешифратор для семисегментного индикатора. Сделать 4x16 версию, недостающие цифры заменить буквами.

Ход работы

- Файл VHDL

```
12    process(y)
13    begin
14       case(y) is
15                                  --ABCDEFGP
16          when "0000" => x <= "11111100";--0
17          when "0001" => x <= "01100000";
18          when "0010" => x <= "11011010";
19          when "0011" => x <= "11110010";
20          when "0100" => x <= "01100110";
21          when "0101" => x <= "10110110";
22          when "0110" => x <= "10111110";
23          when "0111" => x <= "11100000";--7
24                                  --ABCDEFGP
25          when "1000" => x <= "11111110";--8
26          when "1001" => x <= "11110110";
27          when "1010" => x <= "11101110";
28          when "1011" => x <= "00111110";
29          when "1100" => x <= "10011100";
30          when "1101" => x <= "01111010";
31          when "1110" => x <= "10011110";
32          when "1111" => x <= "10001110";--F
33          when others => x <= "00000000";
34       end case;
35    end process;
```

Рисунок 22 - Реализация сегментного дешифратора на VHDL

- Подключение в схематике

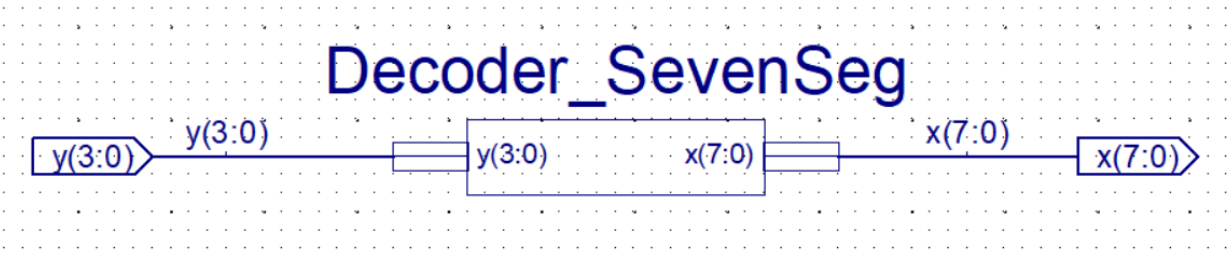

Рисунок 23 - Подключение сегментного дешифратора на схематике

- Модульное тестирование

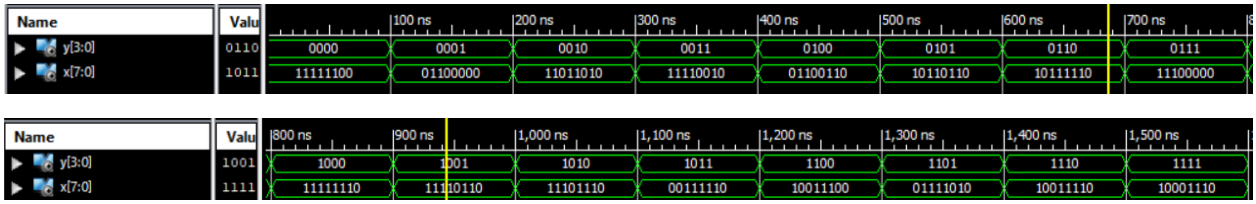

Рисунок 24 - Модульное тестирование сегментного дешифратора

- Тестирование на плате

Рисунок 25 - Результаты тестирования сегментного дешифратора на плате Spartan-6

Вывод: В результате проделанной работы, был реализован дешифратор для сегментного индикатора. Каждому двоичному числу от 0000 до 1111, соответствует своя цифра от ‘0’ до ‘F’. Точка в рассчёт не бралась, она всегда выключена. Индексация сегментов в фрейме символа идёт от большего разряда, к меньшему и выглядит вот так: **ABCDEFGP,** где каждая буква отвечает за свой сегмент соответственно даташиту семисегментного блока KW4-281 (который установлен на плату Spartan-6). Буква ‘P’ обозначает точку.

Задание 3

Цель – составить таблицу истинности для сегмента F и минимизировать её, с помощью карт карно. После, создать на минимизированной основе этот сегмент в схематике.

Таблица 5 - Таблица истинности сегмента F

X3	X2	X1	X0	F
0	0	0	0	**1**
0	0	0	1	**0**
0	0	1	0	**0**
0	0	1	1	**0**
0	1	0	0	**1**
0	1	0	1	**1**
0	1	1	0	**1**
0	1	1	1	**0**
1	0	0	0	**1**
1	0	0	1	**1**
1	0	1	0	**1**
1	0	1	1	**1**
1	1	0	0	**1**
1	1	0	1	**0**
1	1	1	0	**1**
1	1	1	1	**1**

- Минимизация с помощью Карно

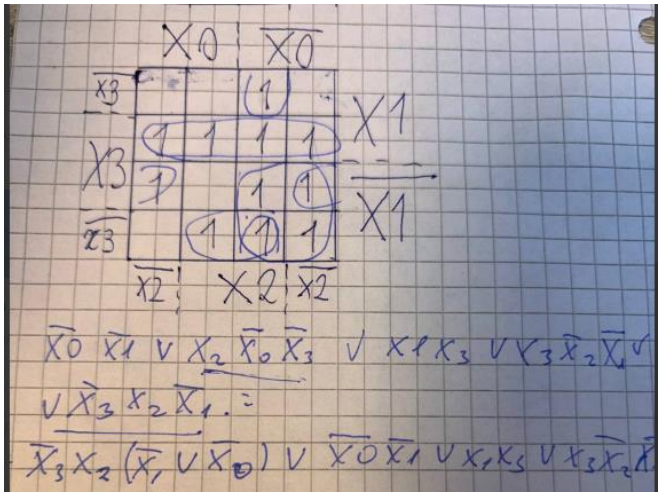

Рисунок 26 - Минимизированная функция

- Реализация в схематике

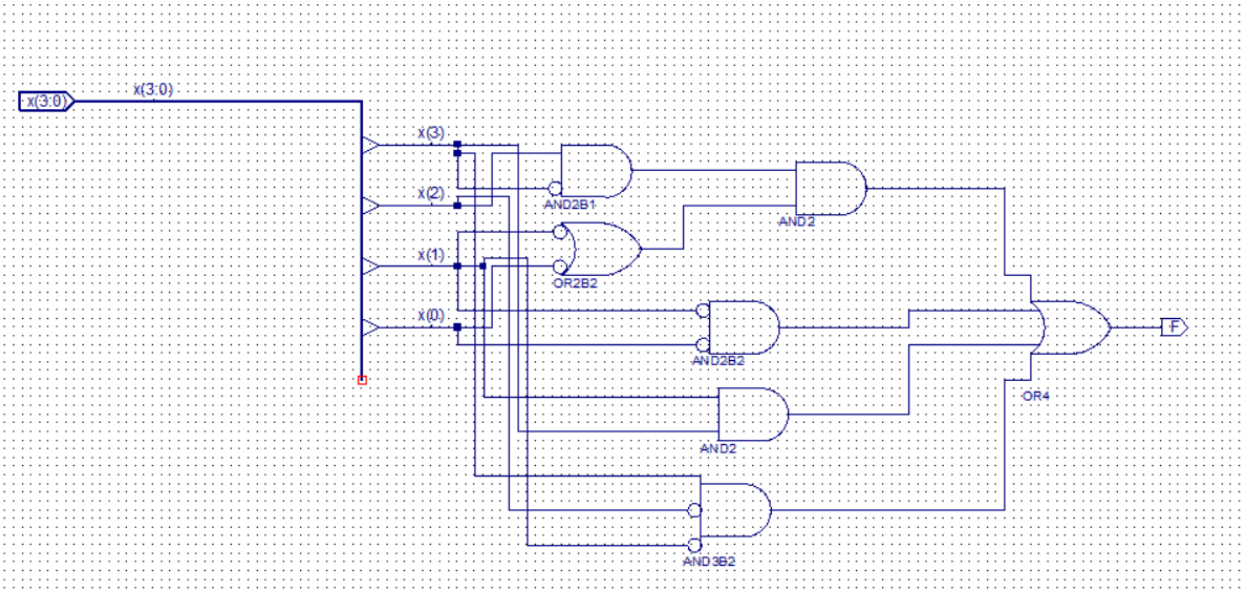

Рисунок 27 - Реализация минимизированной функции на схематике

- Модульное тестирование

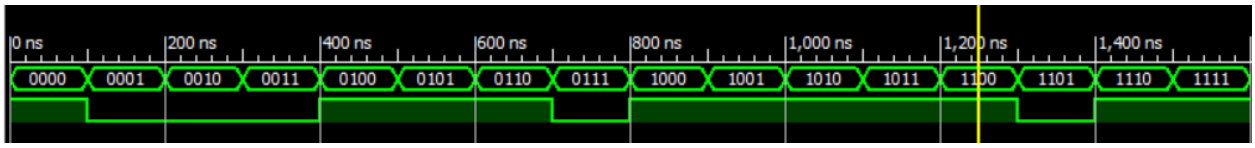

Рисунок 28 - Меандр тестирования схематики

- Тестирование на плате

Рисунок 29 - Результаты тестирования минимизированной функции на плате Spartan-6

Вывод: в результате проделанной работы была составлена минимизированная функция сегмента ‘F’, с помощью карты Карно. В результате тестирования – таблица истинности, построенная на основе минимизированной функции, полностью совпадают с первоначальной таблицей истинности сегмента ‘F’.

Задание 4

Цель – собрать схему устройства динамической индикации. Создать модули дешифратора, мультиплексора, делителя частоты

Реализуем полное управление всеми сегментами и возможность выводить на каждом сегменте отдельное число. Интерфейс будет состоять из **Сигнала разрешения записи, двух свитчей выбора разряда** и **четырёх свитчей выбора числа** от 0 до П (последние две цифры это С и П, чтобы написать ПС:11)

Делитель частоты:

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_UNSIGNED.ALL;
4
5  entity Divider is
6      Port ( V10 : in  STD_LOGIC;
7             Q : out  STD_LOGIC_VECTOR (1 downto 0));
8  end Divider;
9
10 architecture Behavioral of Divider is
11 signal sig : std_logic_vector(1 downto 0) := "00";
12 begin
13    process(V10) --вызывается при изменении сигнала D
14    variable counter : integer := 0;
15    begin
16       if (V10'event and V10 = '1') then
17          if (counter = 400000) then
18             counter := 0;
19             sig <= sig + '1';
20          else
21             counter := counter + 1;
22          end if;
23       end if;
24
25    end process;
26    Q <= sig;
27 end Behavioral;
```

Рисунок 30 - Реализация делителя частоты в VHDL

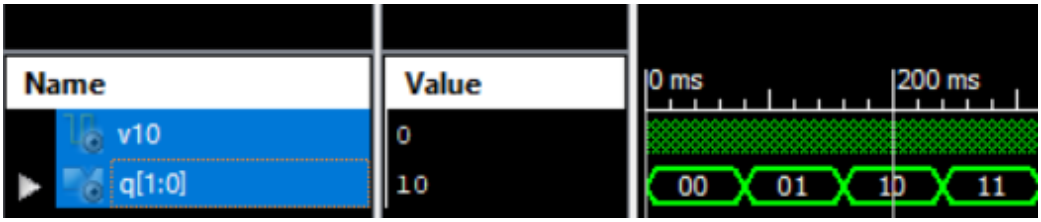

Рисунок 31 - Тестирование делителя частоты

Демультиплексор для управления свитчами:

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


entity Num_demultiplexer is
    Port ( ENABLE : in  STD_LOGIC;
           DIGIT : in  STD_LOGIC_VECTOR (1 downto 0);
           NUMBER : in  STD_LOGIC_VECTOR (3 downto 0);
           NUM_0 : out  STD_LOGIC_VECTOR (3 downto 0);
           NUM_1 : out  STD_LOGIC_VECTOR (3 downto 0);
           NUM_2 : out  STD_LOGIC_VECTOR (3 downto 0);
           NUM_3 : out  STD_LOGIC_VECTOR (3 downto 0));
end Num_demultiplexer;

architecture Behavioral of Num_demultiplexer is

begin
   process(ENABLE, DIGIT, NUMBER)
   begin
      if (ENABLE = '1') then
         case(DIGIT) is
            when "00" => NUM_0 <= NUMBER;
            when "01" => NUM_1 <= NUMBER;
            when "10" => NUM_2 <= NUMBER;
            when "11" => NUM_3 <= NUMBER;
            when others => null;
         end case;
      end if;
   end process;

end Behavioral;
```

Рисунок 32 - Реализация демультиплексора числа на VHDL

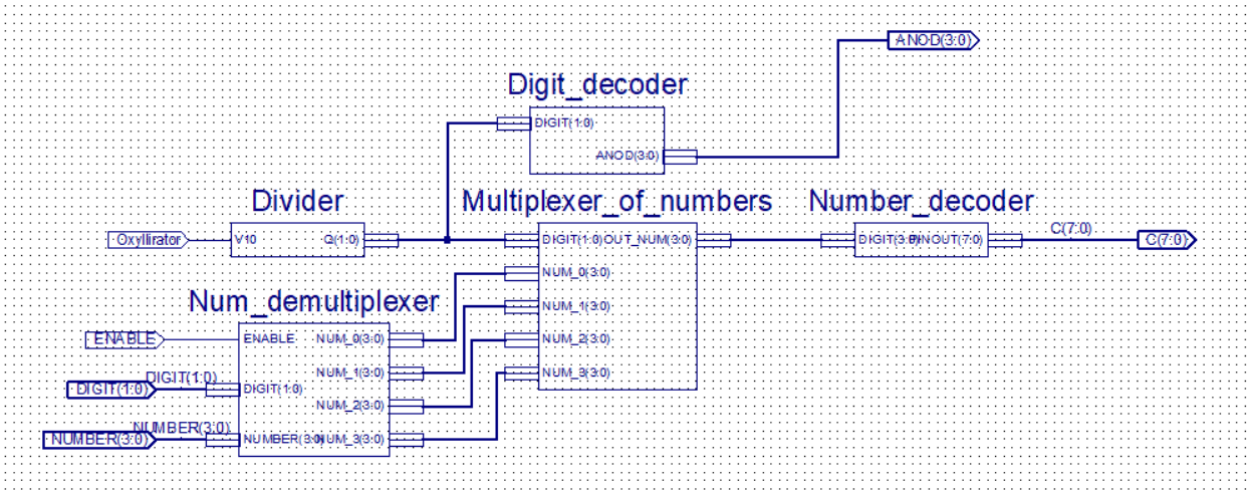

Рисунок 33 - Схема управления динамической индикации семисегментного индикатора

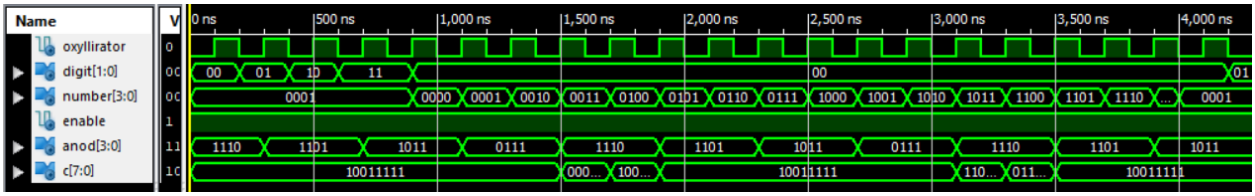

Рисунок 34 - Тестирование схемы управления динамической индикацией семисегментного индикатора

Рисунок 35 - Результаты тестирования схемы на плате Spartan-6

Рисунок 36 - Вывод ПС11 на блок с индикаторами

Вывод: В результате проделанной работы была реализована схема динамической индикации блока семисегментных индикаторов. С помощью свитчей E, Digit и Num можно соответственно управлять разрешением записи, выбором разряда и цифрой, которая будет отображаться. Схема может выводить произвольные числа в 16-ричной СС, с учётом замены двух последних цифр на С и П. Динамичность переключения обеспечивает тактовый генератор на 100МГц, который был понижен до 64 Гц с помощью тактового делителя.

Общий вывод: В результате лабораторной работы были постепенно изучены комбинационные схемы, собраны дешифраторы, мультиплексоры, делители частот, демультиплексоры. Были изучены способы минимизирования логических функций, с помощью метода Карт Карно. Изучены принципы динамической индикации модуля семисегментной индикации. Разработана и протестирована схема динамической индикации, позволяющая в полной мере взаимодействовать с блоком семисегментных индикаторов. Ссылка на проект в GitHub