МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Безопасности жизнедеятельности

ОТЧЕТ

по лабораторной работе №2
по дисциплине «Языки проектирования аппаратуры»
Тема: Представление комбинационных схем и простых триггерных устройств

Студенты гр. 1308 _____ Лепов А. В.

_____ Мельник Д. А.

Преподаватель _____ Гречухин М. Н.

Санкт-Петербург
2024

Цель работы.

Выполнить модификацию и осуществить моделирование комбинационной схемы на основе различных способов задания логической функции, а также исследовать использование комбинационной схемы в сочетании с запоминающим элементом.

Задание на лабораторную работу

Вариант 4: функция 10011100

1. Создать в системе QuestaSim новый проект, включив в него файл Lab2.v. Изменить текст программы Lab2.v так, чтобы реализовать одну из функций таблицы. Сигнал clock при выполнении п.п. 1 – 7 не используется, и соответствующие присваивания закомментированы.

2. Выполнить компиляцию проекта.

3. Запустить процедуру моделирования, вызвав опцию Simulate системы моделирования. Открыть окна наблюдения Source, Process, Signals, Wave.

4. Выполнить моделирование в автоматическом режиме. Просмотреть временную диаграмму, убедиться в правильности реализации логической функции. Сохранить программу в новом файле.

5. Модифицировать программу так, чтобы функция вычислялась на основе разложения Шеннона по одной или двум переменным. Выполнить п.п. 2 и 4.

6. Выполнить описание той же логической функции с использованием оператора case или UDP (по заданию преподавателя). Выполнить п.п. 2 и 4.

8. Вызвать в окно редактора программу, сохраненную в п. 4. Раскомментировать присваивания сигналу clock. Отметим, что в представленном в образце варианте единичное состояние сигнала clock «охватывает» момент изменения сигналов x_0 – x_2.

9. Добавить сигнал и оператор, описывающие триггер типа D со статическим управлением, на информационный вход которого поступает выход комбинационной схемы. Выполнить п.п. 2 и 4.

10. Скорректировать программу так, чтобы реализовался триггер с динамическим управлением. Выполнить п.п. 2 и 4. Сравнить результаты по п.п. 9 и 10.

11. Описать схему, задерживающую сигнал с выхода комбинационной схемы еще на один такт. Для этого добавить описание дополнительного триггера с динамическим управлением и тем же сигналом синхронизации clock, а информационный вход соединить с выходом «первого» триггера. Для корректного описания такой синхронной задержки присваивания должны быть неблокирующими. Выполнить п.п. 2 и 4. А что если здесь использовать блокирующие присваивания?

12. Выполнить компиляцию и моделирование на схемотехническом уровне одного из вариантов описания комбинационной схемы. Для этого воспользоваться синтезирующей САПР Quartus II в соответствии с рекомендациями Приложения 2 данного пособия.

При выполнении этого этапа требуется коррекция программы:

– изменить заголовок программы, определив входы и выходы модуля, объявить режимы портов и типы данных, передаваемых через них;

– исключить оператор initial, тестовые воздействия будут формироваться в отдельном файле.

Выполнив компиляцию, изучить RTL-представление и физическую реализацию (Schematic view) устройства. Оценить затраты и быстродействие схемы.

Содержание этого пункта может изменяться в зависимости от используемого в практикуме оборудования.

Выполнение работы

1. Метод индекса из массива

Первым методом реализации комбинационной схемы является применение восьмиразрядного двоичного числа (массива битов).

Первоначально задаётся массив битов: 8’b00111001 (в обратном порядке из-за особенности следования индексов в числе : 7 - 0).

На выход подаётся элемент этого битового массива с десятичным индексом, равным получаемой на вход трёхзначной двоичной комбинации (например, на вход поступила комбинация 101, следовательно, на выход будет подан элемент массива с индексом 5).

Описание модуля lab2.v:

```
module lab2(
    input [2:0] x,
    input wire clock,
    output wire error,
    output reg z
);

parameter truth_table = 8'b00111001;

assign error = (z == truth_table[x[2] * 4 + x[1] * 2 + x[0]]) ? 0 : 1;

always @(x)
begin
    z = truth_table[x[2] * 4 + x[1] * 2 + x[0] * 1]; // выдача элемента массива
end

endmodule
```

Как видно, для детектирования ошибки функции был использован выходной сигнал ***error*** который сравнивает получаемое значение ***z*** с эталонным из массива ***truth_table.***

Тестирование этого и последующих модулей осуществляется средствами отдельного модуля (тестбенча). Тестбенчи для всех вариаций этого модуля будут одинаковыми.

Тестбенч lab2_tb.v:

```
`timescale 1 ns / 1 ns

module lab2_tb;

reg [2:0] x;
reg clock;
reg [3:0] takt;
wire error;
wire z;

lab2 lab2_inst (
  .x(x),
  .clock(clock),
  .error(error),
  .z(z)
);

initial begin
//   x = 3'b000;
x = 3'bzzz;
for (takt = 0; takt != 8;takt = takt+1)
  begin
  #10
  #2.5  x = takt;
  $display("takt = ", takt, " ", x[2], x[1], x[0], " ", z);
  end
end

endmodule
```

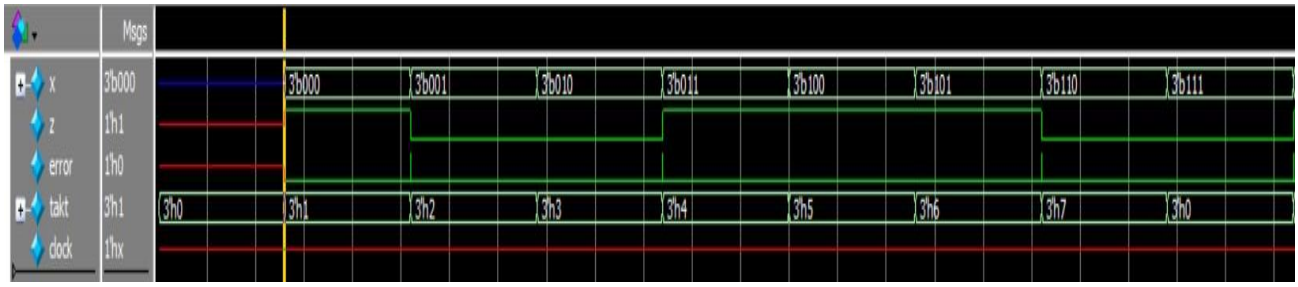

Рисунок 1. Диаграмма моделирования модуля lab2.v

На временных диаграммах ***error*** указывает на существующие риски в моменты переключения сигналов.

2. *Метод логической функции*

Данный метод подразумевает применение в модуле логической функции *z = f (x0, x1, x2)*, результат которой подаётся на выход схемы.

A_{10}	$x_0\,x_1\,x_2$	z
0	000	1
1	001	0
2	010	0
3	011	1
4	100	1
5	101	1
6	110	0
7	111	0

Отсюда получим функцию:

$$z = \overline{x_2} \cdot \overline{x_1} \cdot \overline{x_0} + \overline{x_2} \cdot x_1 \cdot x_0 + x_2 \cdot \overline{x_1} \cdot \overline{x_0} + x_2 \cdot \overline{x_1} \cdot x_0$$
$$= \overline{x_2} \cdot (\overline{x_1} \cdot \overline{x_0} + x_1 \cdot x_0) + x_2 \cdot \overline{x_1} \cdot (\overline{x_0} \cdot x_0)$$
$$= \overline{x_2} \cdot (\overline{x_1} \cdot \overline{x_0} + x_1 \cdot x_0) + x_2 \cdot \overline{x_1}$$

Описание модуля lab2_1.v:

```
module lab2_1(
    input [2:0] x,
    input wire clock,
    output wire error,
    output reg z
);

parameter truth_table = 8'b00111001;

assign error = (z == truth_table[x[2] * 4 + x[1] * 2 + x[0]]) ? 0 : 1;

always @(x)
begin
    z = ~x[2] && (~x[1] && ~x[0] || x[1] && x[0]) || x[2] && ~x[1];
end

endmodule
```

Рисунок 2. Временное моделирование модуля lab2_1.v

3. *Метод разложения Шеннона*

Ниже приведено описание модуля *lab2_2.v* и результат его временного моделирования.

Описание модуля lab2_2.v:

```
module lab2_2(
    input [2:0] x,
    input wire clock,
    output wire error,
    output reg z
);

parameter truth_table = 8'b00111001;

assign error = (z == truth_table[x[2] * 4 + x[1] * 2 + x[0]]) ? 0 : 1;

always @(x)
begin
    if (~x[2])
        z = ~(x[1] ^ x[0]);
    else
        z = ~x[1];
end

endmodule
```

Рисунок 3. Результат временного моделирования модуля lab2_2.v

4. Метод case

Ниже приведены реализация модуля через перебор всех возможных комбинаций входных воздействий в блоке ***case.***

Описание lab2_3.v:

```
module lab2_3(
    input [2:0] x,
    input wire clock,
    output wire error,
    output reg z
);

parameter truth_table = 8'b00111001;

assign error = (z == truth_table[x[2] * 4 + x[1] * 2 + x[0]]) ? 0 : 1;

always @(x)
begin
    case (x)
        3'b000 : z = 1'b1;
        3'b001 : z = 1'b0;
        3'b010 : z = 1'b0;
        3'b011 : z = 1'b1;
        3'b100 : z = 1'b1;
        3'b101 : z = 1'b1;
        3'b110 : z = 1'b0;
        3'b111 : z = 1'b0;

        default: z = 1'bz;
    endcase
end

endmodule
```

Рисунок 4. Результат временного моделирования модуля lab2_3.v

5. *Метод UDP*

Описание модуля lab2_4.v:

```
module lab2_4(
    input [2:0] x,
    input wire clock,
    output wire error,
    output reg z
);

parameter truth_table = 8'b00111001;

my_logic udp (
    .zt(z),
    .a(x[2]),
    .b(x[1]),
    .c(x[0])
);

assign error = (z == truth_table[x[2] * 4 + x[1] * 2 + x[0]]) ? 0 : 1;

endmodule

primitive my_logic(zt,a,b,c);
output zt;
input a,b,c;
table
      // a, b, c : zt;
         0  0  0 : 1;
         0  0  1 : 0;
         0  1  0 : 0;
         0  1  1 : 1;
         1  0  0 : 1;
         1  0  1 : 1;
         1  1  0 : 0;
         1  1  1 : 0;
endtable
endprimitive
```

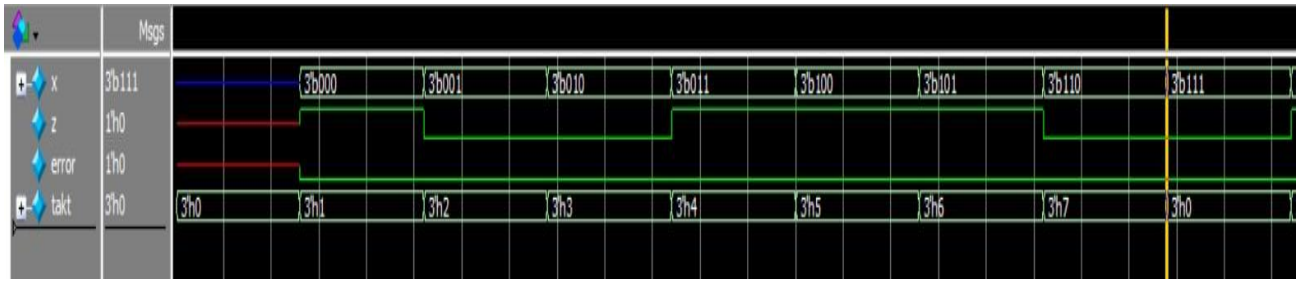

Рисунок 5. Временная диаграмма моделирования модуля lab2_4.v

6. Введение в схему статического D-триггера

Для реализации модуля с применением статического D-триггера потребовалось ввести тактирующий сигнал ***clock.***

В данном случае сущность статического D-тригерра реализуется через выходную переменную ***D***, которая находится в блоке ***always*** зависящем от тактового сигнала и входных воздействий ***x.***

Так как триггер статический, ***D*** может меняться только при условии истинности тактового сигнала.

Описание модуля lab2_5.v:

```
module lab2_5(
    input [2:0] x,
    input wire clock,
    output wire error,
    output reg D,
    output reg z
);

parameter truth_table = 8'b00111001;

assign error = (z == truth_table[x[2] * 4 + x[1] * 2 + x[0]]) ? 0 : 1;

always @(x)
begin
    z = ~x[2] && (~x[1] && ~x[0] || x[1] && x[0]) || x[2] && ~x[1];
end

always @(z, clock) begin
    if (clock) begin
        D = z;
    end
end

endmodule
```

Для тестирования модуля с тактовым сигналом требуются изменения в тестбенче: необходимо добавить «бесконечное» изменение входного тактового сигнала.

Описание тестбенча lab2_5_tb.v:

```
`timescale 1 ns / 1 ns

module lab2_5_tb;

  reg [2:0] x;
  reg  clock;
  wire  error;
  reg [2:0] takt;
  wire z;
  wire  D;


  lab2_5  lab2_5_inst (
    .x(x),
    .clock(clock),
    .error(error),
    .z(z),
    .D(D)
  );

initial begin
  clock = 1'b0;
end

always #5 clock = ~clock;

initial begin
//   x = 3'b000;
x = 3'bzzz;
for (takt = 0; takt != 8;takt = takt+1)
  begin
  #10
  #2.5  x = takt;
  $display("takt = ", takt, " ", x[2], x[1], x[0], " ", z);
  end
end

endmodule
```

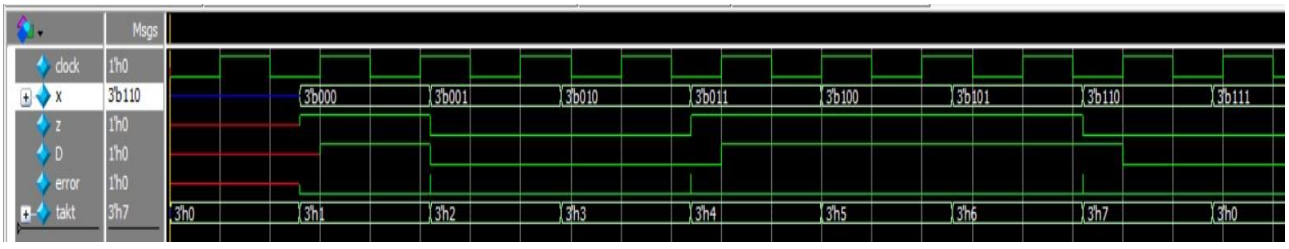

Рисунок 6. Временное моделирование модуля lab2_5.v

7. *Добавление динамического D-триггера*

Основным отличием реализации динамического триггера от статического является зависимость блока ***always*** от переднего фронта тактового сигнала (теперь ***always @(posedge clock)***). Для сравнения в схеме оставлен статический триггер (***D_s***).

Описание модуля lab2_6.v:

```
module lab2_6(
    input [2:0] x,
    input wire clock,
    output wire error,
    output reg D_s, D_d,
    output reg z
);

parameter truth_table = 8'b00111001;

assign error = (z == truth_table[x[2] * 4 + x[1] * 2 + x[0]]) ? 0 : 1;

always @(x)
begin
    z = ~x[2] && (~x[1] && ~x[0] || x[1] && x[0]) || x[2] && ~x[1];
end

always @(clock, z)
    if (clock) D_s = z;

always @(posedge clock)
    D_d = z;


endmodule
```

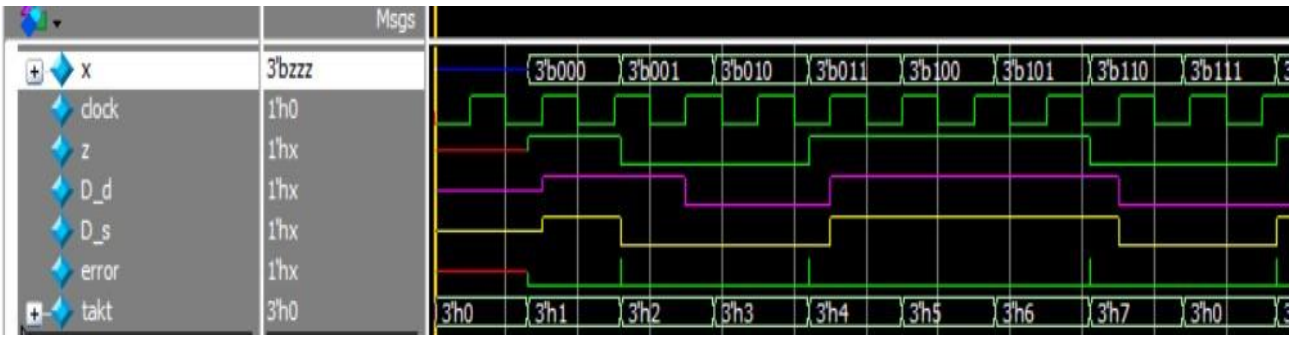

Рисунок 7. Результат временного моделирования модуля lab2_6.v

8. Сдвиг выходного сигнала средствами D-триггеров.

Для сдвига сигнала применяется регистр из двух динамических D-триггеров. Выход первого триггера подключён ко входу второго. Присваивания неблокирующие.

Описание модуля lab2_7.v:

```
module lab2_7(
    input [2:0] x,
    input wire clock,
    output wire error,
    output reg D_d1, D_d2,
    output reg z
);

parameter truth_table = 8'b00111001;

assign error = (z == truth_table[x[2] * 4 + x[1] * 2 + x[0]]) ? 0 : 1;

always @(x)
begin
    z = ~x[2] && (~x[1] && ~x[0] || x[1] && x[0]) || x[2] && ~x[1];
end

always @(posedge clock)
    D_d1 <= z;

always @(posedge clock)
    D_d2 <= D_d1;


endmodule
```

Рисунок 8. Временное моделирование модуля lab2_7.v

9. *Компиляция проекта в САПР Quartus II*

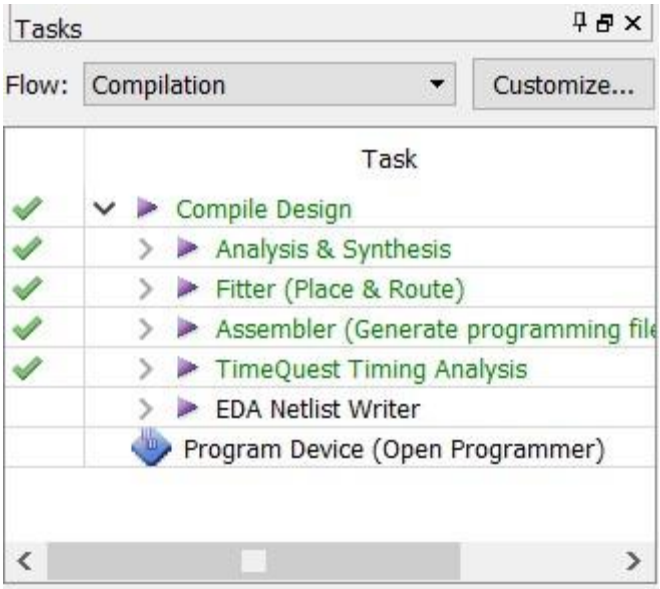

Рисунок 9. Ход компиляции

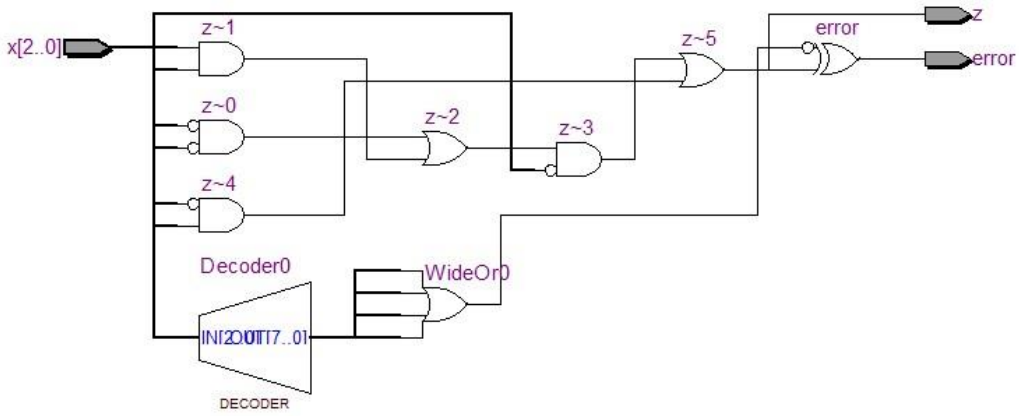

Рисунок 10. Синтезированная схема

Вывод

В ходе выполнения лабораторной работы были изучены различные методы описания комбинационных схем на языке Verilog. Были рассмотрены реализации D-триггеров и способы их применения как в одиночном виде, так и в регистровом.