Университет ИТМО

Факультет ПИиКТ

Функциональная схемотехника

Лабораторная работа №2

«Последовательностная логика»

Вариант 2

Работу выполнил

Голиков Д.И.

Группа

P33102

Санкт-Петербург

2024

Оглавление

Введение

Цель работы:

Познакомиться и научиться техникам описания последовательностных схем и блоков на RTL-уровне с использованием языка Verilog HDL. Работа выполняется в Vivado Design Suite.

Вариант:

Функция 1	FSM	Функция 2	Разрядность	Делитель частоты
COUNT_ZEROES	FSM_3	LRU	8 бит	4

Часть 1

Счетчик

Для реализации счетчика на языке Verilog HDL, были использованы следующе сигналы в интерфейсе данного модуля:

Порт	**Тип**	**Описание**
clk	Вх.	Сигнал тактовой частоты
reset	Вх.	Синхронный сигнал сброса
ud	Вх.	Сигнал инкремента/декремента
load	Вх.	Сигнал загрузки значения счетчика из шины данных
data	Вх.	8-ми битная шина данных
count	Вых.	8-ми битная шина счетчика
rst	Вх.	Асинхронный сигнал сброса

```
`timescale 1ns / 1ps
module simple_counter(
    clk, // тактовый сигнал
    reset, // сигнал сброса
    ud, //сигнал инкремента up_down
    load, // сигнал загрузки значения счетчика
    data,
    count, //как идея сюда можно добавить еще один регистр - запись счетчика в дату
    rst // асинхронный сброс
);
  input clk,reset,load,ud,rst;
  input [7:0] data;
  output reg [7:0] count;
  //всегда блок будет выполняться на каждом положительном фронте тактовой частоты
  always@(posedge clk or posedge rst)
  begin
    if(rst)
        count = 0;
    else if(reset)    //Установите счетчик на ноль
      count = 0;
    else if(load)    //загрузите в счетчик значение данных
      count = data;
    else if(ud)        //инкремент
      count = count + 1;
    else            //декремент
      count = count - 1;
  end
endmodule
```

Рисунок 1 – разработанный модуль счетчика.

```
module counter_tb;
  reg clk,reset,load,ud,rst;
  reg [7:0] data;
  wire [7:0] count;
  reg [7:0] master;

  // instance counter design
  simple_counter ct_1(
  .clk(clk),
  .reset(reset),
  .ud(ud),
  .load(load),
  .data(data),
  .count(count),
  .rst(rst)
  );
  //clock generator
  initial begin
    clk = 1'b0;
    repeat(100) #1 clk= ~clk;
   end
  //входной сигнал
   initial begin
        rst = 1'b0;
        reset=1'b1;
        #4 reset=1'b0;
        #11 rst = 1'b1;
        #1 rst = 1'b0;
        #60 reset=1'b1;
        #2 reset=1'b0;
   end
   initial begin
        load = 1'b0;
        #23 load=1'b1;
        #5 load=1'b0;
   end
   initial begin
        ud = 1'b0;
        #2 ud=1'b1;
        #24 ud=1'b0;
   end
   initial begin
        data=8'b11111000;
        #14 data=8'b11111101;
        #2 data=8'b11111111;
   end
```

```
  initial begin
       master <= 8'b00000000;
       #4 master = master + 1 ;
       repeat(5)#2 master = master + 1 ;
       #1 master <= 8'b00000000;
       #1 master = master + 1 ;
       repeat(3) #2 master = master + 1 ;
       #2 master = 8'b11111111;
  end
  initial begin
       $monitor("time=%0d,reset=%b,rst=%b,load=%b,ud=%b,data=%d,count=%d",
                              $time,reset,rst,load,ud,data,count);
  end
  initial begin
       #50 $stop;
  end
endmodule
```

Рисунок 2 – тестирование разработанного модуля.

В качестве эталонной модели был вручную инкрементировано и сброшено значение master.

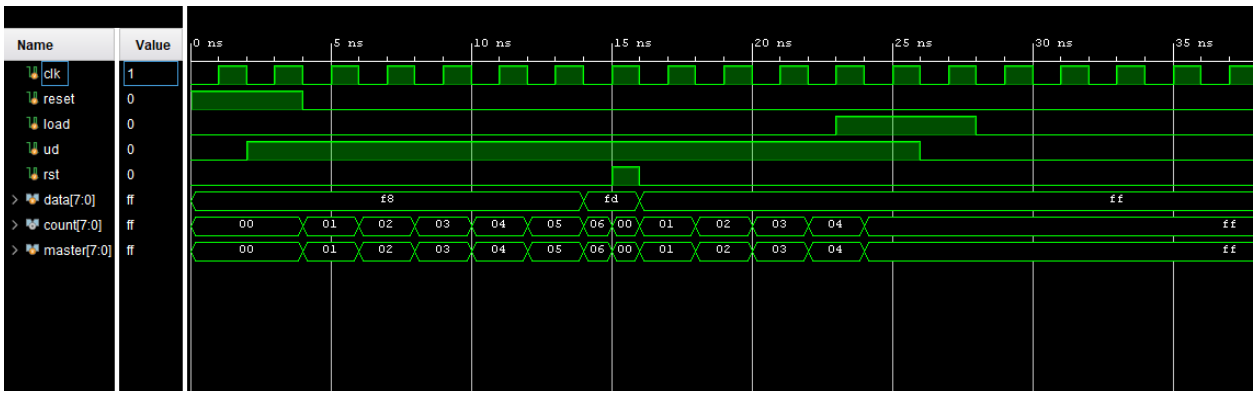


Рисунок 3 – Вывод симуляции тестирования.

```
time=0,reset=1,rst=0,load=0,ud=0,data=248,count=  x
time=1,reset=1,rst=0,load=0,ud=0,data=248,count=  0
time=2,reset=1,rst=0,load=0,ud=1,data=248,count=  0
time=4,reset=0,rst=0,load=0,ud=1,data=248,count=  0
time=5,reset=0,rst=0,load=0,ud=1,data=248,count=  1
time=7,reset=0,rst=0,load=0,ud=1,data=248,count=  2
time=9,reset=0,rst=0,load=0,ud=1,data=248,count=  3
time=11,reset=0,rst=0,load=0,ud=1,data=248,count=  4
time=13,reset=0,rst=0,load=0,ud=1,data=248,count=  5
time=14,reset=0,rst=0,load=0,ud=1,data=253,count=  5
time=15,reset=0,rst=1,load=0,ud=1,data=253,count=  0
time=16,reset=0,rst=0,load=0,ud=1,data=255,count=  0
time=17,reset=0,rst=0,load=0,ud=1,data=255,count=  1
time=19,reset=0,rst=0,load=0,ud=1,data=255,count=  2
time=21,reset=0,rst=0,load=0,ud=1,data=255,count=  3
time=23,reset=0,rst=0,load=1,ud=1,data=255,count=255
time=26,reset=0,rst=0,load=1,ud=0,data=255,count=255
time=28,reset=0,rst=0,load=0,ud=0,data=255,count=255
time=29,reset=0,rst=0,load=0,ud=0,data=255,count=254
time=31,reset=0,rst=0,load=0,ud=0,data=255,count=253
time=33,reset=0,rst=0,load=0,ud=0,data=255,count=252
time=35,reset=0,rst=0,load=0,ud=0,data=255,count=251
time=37,reset=0,rst=0,load=0,ud=0,data=255,count=250
time=39,reset=0,rst=0,load=0,ud=0,data=255,count=249
time=41,reset=0,rst=0,load=0,ud=0,data=255,count=248
time=43,reset=0,rst=0,load=0,ud=0,data=255,count=247
time=45,reset=0,rst=0,load=0,ud=0,data=255,count=246
time=47,reset=0,rst=0,load=0,ud=0,data=255,count=245
time=49,reset=0,rst=0,load=0,ud=0,data=255,count=244
```

Рисунок 4 – Консольный вывод симуляции тестирования.

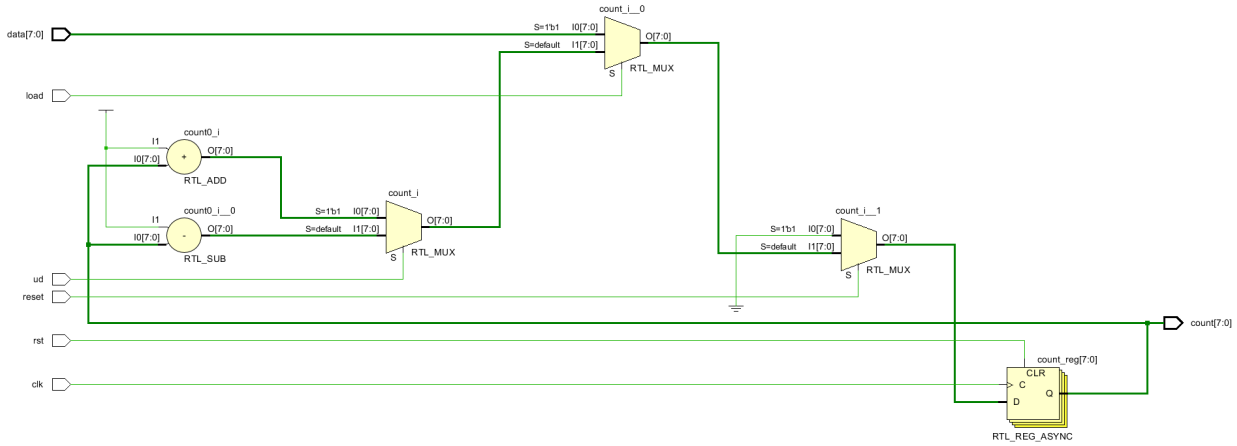


Рисунок 5 – Схема счётчика.

Сдвиговой регистр

Для реализации сдвигового на языке Verilog HDL, были использованы следующе сигналы в интерфейсе данного модуля:

Порт	**Тип**	**Описание**
clk	Вх.	Сигнал тактовой частоты
reset	Вх.	Синхронный сигнал сброса
enable	Вх.	Сигнал разрешения
load	Вх.	Сигнал загрузки данных из шины данных в регистр
data	Вх.	8-ми битная шина данных
out	Вых.	8-ми битная шина счетчика
rst	Вх.	Асинхронный сигнал сброса
register	Вых.	Шина данных регистра, для отслеживания корректности сдвига. (Может быть убран)

```
1   `timescale 1ns / 1ps
2
3   module shift_register (
4       input clk,                          // тактовый сигнал
5       input reset,                        // сигнал сброса
6       input enable,                       // сигнал разрешения
7       input rst,                          // асинхронный сброс
8       input load,                         // сигнал загрузки данных в регистр
9       input [7:0] data,                   // параллельный вход
10      output reg out,                     //  выход крайней правой
11      output reg [7:0] register
12  );
13
14  always @(posedge rst) begin
15      if (rst) begin
16          out <= 0;
17          register <= 8'b0;
18      end
19  end
20  //always @(posedge load) begin          // Если установлен флаг load, то загружаем данные из даты в регистр
21  //    register = data;
22  //end
23  //reg [7:0] register;                   // объявление регистра
24  always @(posedge clk) begin             //работа с регистром и сдвигами
25      if (load) begin                     // Если установлен флаг load, то загружаем данные из даты в регистр
26          register = data;
27      end else begin
28          if (reset) begin
29              register <= 8'b0;           // сброс регистра
30          end else begin
31          if (enable) begin
32              register [7] <= 0;          // Присвоение крайнему левому значение 0
33              out <= register[0];         // Запоминаем крайний правый
34              register <= {register[7:1]};    // сдвиг вправо
35              end
36          end
37      end
38  end
39
40
41  endmodule
```

Рисунок 6 – разработанный модуль сдвигового регистра.

```
1  `timescale 1ns / 1ps
2  module shift_register_tb;
3      reg clk, reset, enable, rst,load;
4      reg [7:0] data;
5      wire [7:0] register;
6      wire out;
7      reg [7:0] master_data;
8      reg master;
9      shift_register shift(
10     .clk(clk),
11     .reset(reset),
12     .enable(enable),
13     .out(out),
14     .rst(rst),
15     .load(load),
16     .data(data),
17     .register(register)
18     );
19 initial begin
20     clk = 1'b0;
21     repeat(100) #1 clk= ~clk;
22 end
23 initial begin
24     rst = 1'b0;
25     reset = 1'b0;
26     //#4 reset = 1'b0;
27     //#11 rst = 1'b1;
28     //#1 rst = 1'b0;
29     //#60 reset = 1'b1;
30     //#2 reset = 1'b0;
31 end
32 initial begin
33     enable = 1'b1;
34     #20 enable = 1'b0;
35     #10 enable = 1'b1;
36 end
37 initial begin
38
39     load = 1'b0;
40     data = 8'b00110101;
41     #1 load = 1'b1;
42     #1 load = 1'b0;
43     //#3 data = 8'b10000111;
44     data = 8'b11110000;
45     #11 load = 1'b1;
46     #1 load = 1'b0;
47 end
```

```
48  initial begin
49          #1 master_data <= 8'b00110101;
50          repeat(5)#2 master_data = {1'b0, master_data[7:1]};
51          #2 master_data = 8'b11110000;
52          repeat(3)#2 master_data = {1'b0, master_data[7:1]};
53          #12 master_data = {1'b0, master_data[7:1]};
54          repeat(5)#2 master_data = {1'b0, master_data[7:1]};
55  end
56  initial begin
57          master <= 0;
58      #3 master = master_data[0]&1;
59      repeat(8) #2 master = master_data[0]&1;
60      #12 master = master_data[0]&1;
61      repeat(5)#2 master = master_data[0]&1;
62  end
63  initial begin
64      $monitor("time=%0d,reset=%b,rst=%b,enable=%b,data=%b,register=%b,out=%b",
65                              $time,reset,rst,enable,data,register,out);
66  end
67  initial begin
68  #60 $stop;
69  end
70  endmodule
71
```

Рисунок 7 – тестирование разработанного модуля.

В качестве эталонной модели было вручную сдвинут регистр и записано значение логического И с единицей.

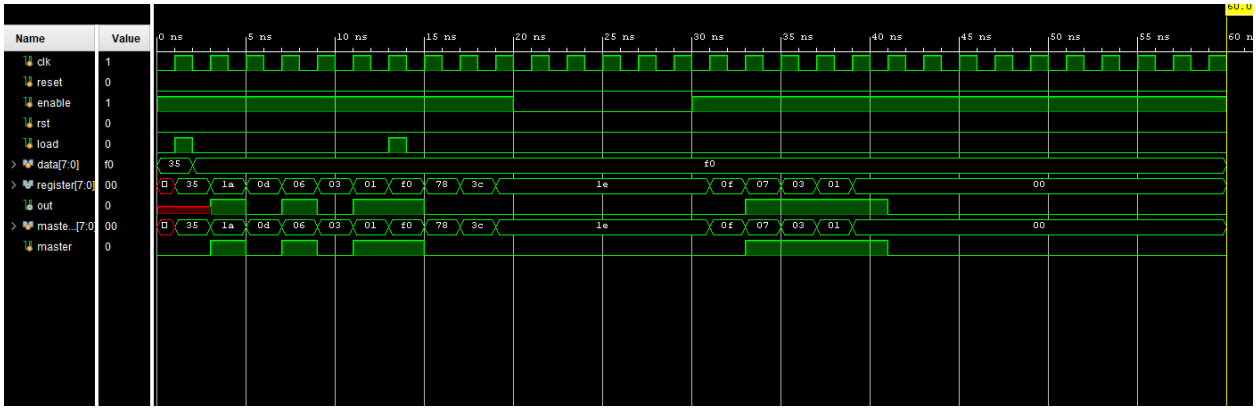


Рисунок 8 – Вывод симуляции тестирования.

```
time=0,reset=0,rst=0,enable=1,data=00110101,register=xxxxxxxx,out=x
time=1,reset=0,rst=0,enable=1,data=00110101,register=00110101,out=x
time=2,reset=0,rst=0,enable=1,data=11110000,register=00110101,out=x
time=3,reset=0,rst=0,enable=1,data=11110000,register=00011010,out=1
time=5,reset=0,rst=0,enable=1,data=11110000,register=00001101,out=0
time=7,reset=0,rst=0,enable=1,data=11110000,register=00000110,out=1
time=9,reset=0,rst=0,enable=1,data=11110000,register=00000011,out=0
time=11,reset=0,rst=0,enable=1,data=11110000,register=00000001,out=1
time=13,reset=0,rst=0,enable=1,data=11110000,register=11110000,out=1
time=15,reset=0,rst=0,enable=1,data=11110000,register=01111000,out=0
time=17,reset=0,rst=0,enable=1,data=11110000,register=00111100,out=0
time=19,reset=0,rst=0,enable=1,data=11110000,register=00011110,out=0
time=20,reset=0,rst=0,enable=0,data=11110000,register=00011110,out=0
time=30,reset=0,rst=0,enable=1,data=11110000,register=00011110,out=0
time=31,reset=0,rst=0,enable=1,data=11110000,register=00001111,out=0
time=33,reset=0,rst=0,enable=1,data=11110000,register=00000111,out=1
time=35,reset=0,rst=0,enable=1,data=11110000,register=00000011,out=1
time=37,reset=0,rst=0,enable=1,data=11110000,register=00000001,out=1
time=39,reset=0,rst=0,enable=1,data=11110000,register=00000000,out=1
time=41,reset=0,rst=0,enable=1,data=11110000,register=00000000,out=0
```

Рисунок 9 – Консольный вывод симуляции тестирования.

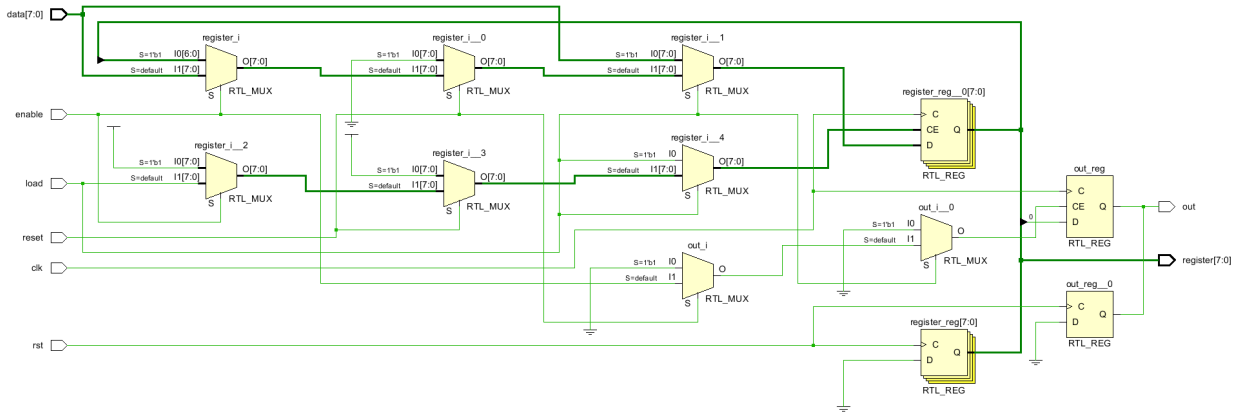


Рисунок 10 – Схема сдвигового регистра.

Конечный автомат

Вариант: FSM_2

Один сумматор, один умножитель.

Реализуемое выражение: ((A+B) *2+B)

Область допустимых значений:

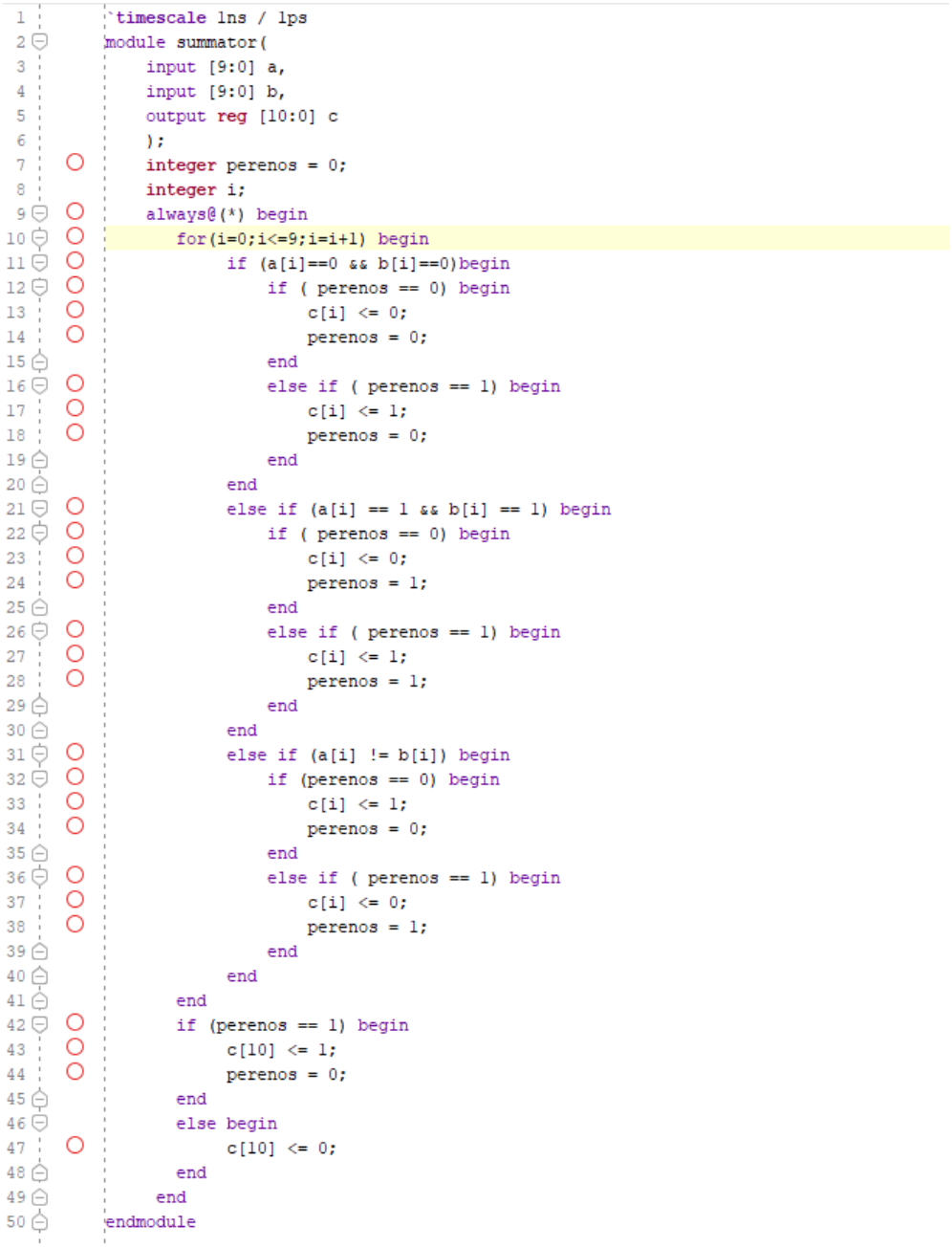
Так как у нас разрядность 8 бит. Следовательно исходные числа восьмибитные при суммировании двух восьмибитных максимально возможных чисел может получиться 9 битное число. При умножении двоичного числа на два, происходит регистровый сдвиг влево на 1, следовательно 9 битное число превращается в 10 битное. В конце надо результат произведения сложить с 8 битным числом, следовательно максимально возможное получаемое в результате выражения число становится 11 битным.

Для того, чтобы собрать конечный автомат, было принято решение для начала сделать сумматор и умножитель, протестировать их, а затем объединить это в конечный автомат.

Сумматор v1

Порт	**Тип**	**Описание**
a	Вх.	Шина первого слагаемого
b	Вх.	Шина второго слагаемого
c	Вых.	Шина результата

Для реализации сумматора без оператора “+”, было принято реализовать суммирование через сравнивание. Для этого была добавлена переменная для бита переноса. Затем побитное сравнение значений и присвоение соответствующих значений для бита переноса и бита результата.



```
`timescale 1ns / 1ps
module summator(
    input [9:0] a,
    input [9:0] b,
    output reg [10:0] c
    );
    integer perenos = 0;
    integer i;
    always@(*) begin
       for(i=0;i<=9;i=i+1) begin
            if (a[i]==0 && b[i]==0)begin
                if ( perenos == 0) begin
                    c[i] <= 0;
                    perenos = 0;
                end
                else if ( perenos == 1) begin
                    c[i] <= 1;
                    perenos = 0;
                end
            end
            else if (a[i] == 1 && b[i] == 1) begin
                if ( perenos == 0) begin
                    c[i] <= 0;
                    perenos = 1;
                end
                else if ( perenos == 1) begin
                    c[i] <= 1;
                    perenos = 1;
                end
            end
            else if (a[i] != b[i]) begin
                if (perenos == 0) begin
                    c[i] <= 1;
                    perenos = 0;
                end
                else if ( perenos == 1) begin
                    c[i] <= 0;
                    perenos = 1;
                end
            end
       end
       if (perenos == 1) begin
            c[10] <= 1;
            perenos = 0;
       end
       else begin
            c[10] <= 0;
       end
     end
endmodule
```

Рисунок 11 – разработанный модуль сумматора.

```
1   `timescale 1ns / 1ps
2
3   module summator_tb;
4       reg [10:0] a,b;
5       wire [10:0] c;
6
7   summator sum (
8       .a(a),
9       .b(b),
10      .c(c)
11      );
12      reg [10:0] out_putin;
13
14      initial begin
15          a = 8'b00000001; // 1
16          b = 8'b00000001; // 1
17          #2
18          a = 8'b00010011; // 19
19          b = 8'b00000101; // 5
20          #2
21          a = 8'b10000101; // 133
22          b = 8'b00000101;  // 5
23          #2
24          a = 8'b00011000; // 24
25          b = 8'b11111100; // 252
26          #2
27          a = 8'b11111111; // 255
28          b = 8'b11111111; // 255
29      end
30
31      initial begin
32          assign out_putin = a + b;
33          $monitor("A=%b , B=%b, SUM=%b, Sum_master=%b", a, b, c, out_putin);
34
35          #10 $stop;
36      end
37
38  endmodule
39
```

Рисунок 11 – тестирование разработанного модуля.

В качестве эталонной модели была выбрана обычная встроенная операция сложения.

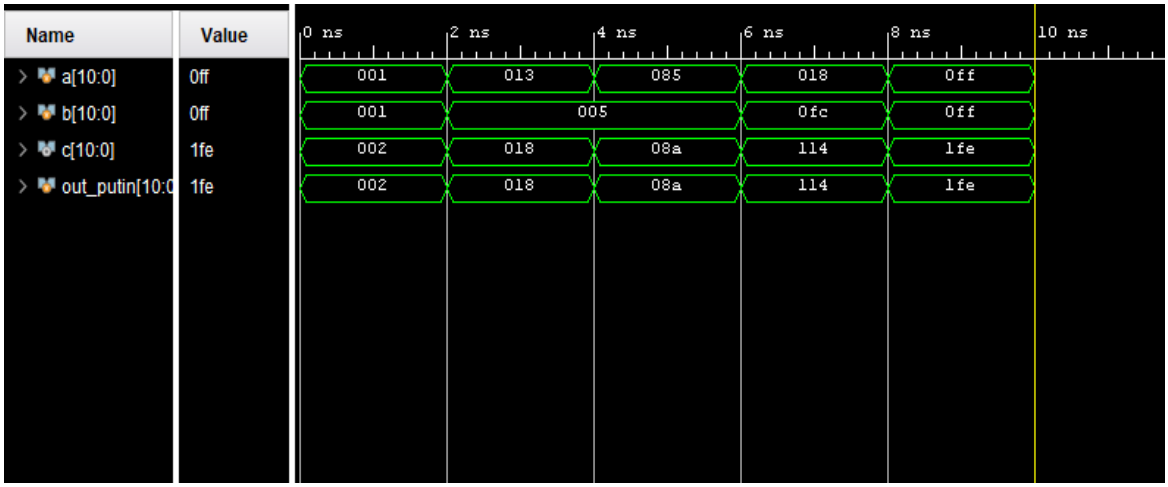


Рисунок 12 – Вывод симуляции тестирования.

```
A=00000000001 , B=00000000001, SUM=00000000010, Sum_master=00000000010
A=00000010011 , B=00000000101, SUM=00000011000, Sum_master=00000011000
A=00010000101 , B=00000000101, SUM=00010001010, Sum_master=00010001010
A=00000011000 , B=00011111100, SUM=00100010100, Sum_master=00100010100
A=00011111111 , B=00011111111, SUM=00111111110, Sum_master=00111111110
$stop called at time : 10 ns : File "C:/Users/Solnyshko/lab2/lab2.srcs/sim_1/new/summator_tb.v" Line 38
```

Рисунок 13 - Консольный вывод симуляции тестирования.

Умножитель v1

Порт	Тип	Описание
AB	Вх.	Шина слагаемого
out	Вых.	Шина результата

Для реализации умножителя было принято решение регистрового сдвига влево на 1 бит и присвоение последнему биту значение 0.

```
1  `timescale 1ns / 1ps
2  module umnozhitel(
3      input [9:0] AB,
4      output reg [10:0] out
5      );
6      always @* begin
7          out <= {AB[9:0], 1'b0};
8      end
9  endmodule
10
```

Рисунок 14 – разработанный модуль умножителя.

```
`timescale 1ns / 1ps
module umnozhitel_master(
    input [7:0] AB,
    output reg [7:0] out_master
    );
    always @* begin
        out_master <= AB*2;
    end

endmodule
```

Рисунок 15 – эталонный модуль умножителя.

```
`timescale 1ns / 1ps
module umnozhitel_tb;
    reg [7:0] AB;
    wire [8:0] out, out_master;

    umnozhitel umozh(
    .AB(AB),
    .out(out)
    );

    umnozhitel_master master(
    .AB(AB),
    .out_master(out_master)
    );


    initial begin
           AB = 8'b00000001;
        #1 AB = 8'b00000010;
        #1 AB = 8'b00000100;
        #1 AB = 8'b00001000;
        #1 AB = 8'b00010000;
        #1 AB = 8'b00100000;
        #1 AB = 8'b01000000;
        #1 AB = 8'b10000000;
        #1 AB = 8'b00110101;
        #1 AB = 8'b00001111;
        #1 AB = 8'b00111111;
    end
    initial begin

        $display("AB = %b, out = %b, out_master= %b",AB,out,out_master);
        #10 $stop;
    end

endmodule
```

Рисунок 16 - тестирование разработанного модуля.

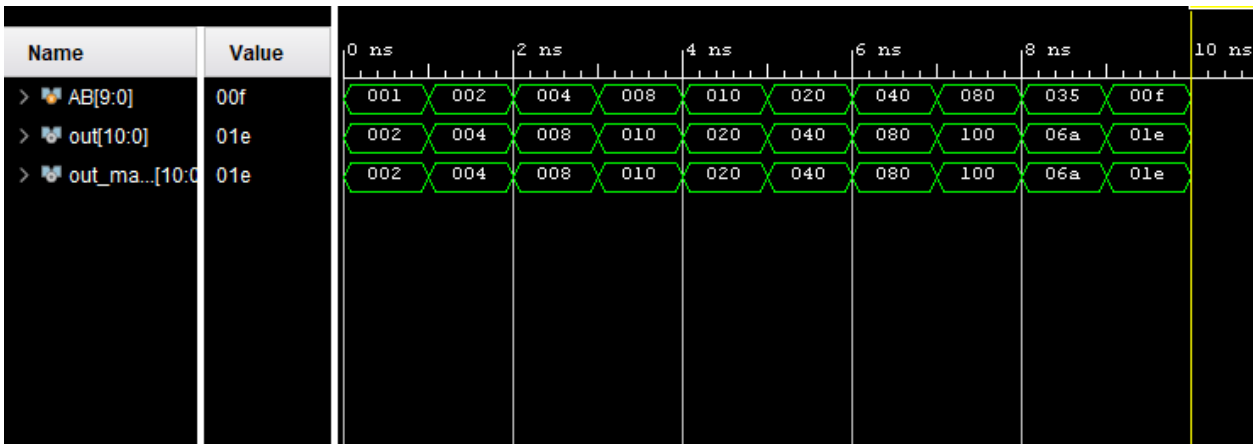


Рисунок 17 – Вывод симуляции тестирования.

```
AB = 0000000001, out = 00000000010, out_master= 00000000010
AB = 0000000010, out = 00000000100, out_master= 00000000100
AB = 0000000100, out = 00000001000, out_master= 00000001000
AB = 0000001000, out = 00000010000, out_master= 00000010000
AB = 0000010000, out = 00000100000, out_master= 00000100000
AB = 0000100000, out = 00001000000, out_master= 00001000000
AB = 0001000000, out = 00010000000, out_master= 00010000000
AB = 0010000000, out = 00100000000, out_master= 00100000000
AB = 0000110101, out = 00001101010, out_master= 00001101010
AB = 0000001111, out = 00000011110, out_master= 00000011110
$stop called at time : 10 ns : File "C:/Users/Solnyshko/lab2/lab2.srcs/sim_1/new/umnozhitel_tb.v" Line 33
```

Рисунок 18 - Консольный вывод симуляции тестирования.

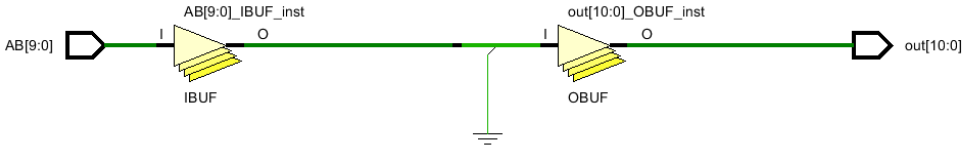


Рисунок 19 – схема умножителя.

Конечный автомат v1

Порт	**Тип**	**Описание**
A	Вх.	Шина первого слагаемого
B	Вх.	Шина второго слагаемого
result	Вых.	Шина результата

```
1   `timescale 1ns / 1ps
2   module avtomat_tb;
3           reg [9:0] A,B;
4           wire [10:0] result;
5
6   avtomat avt(
7       .value1(A),
8       .value2(B),
9       .result(result)
10  );
11  reg [10:0] result_master;
12
13  initial begin
14          A = 8'b00000001; // 1
15          B = 8'b00000001; // 1
16          #2
17          A = 8'b00010011; // 19
18          B = 8'b00000101; // 5
19          #2
20          A = 8'b01000101; // 69
21          B = 8'b00000101;  // 5
22          #2
23          A = 8'b00011000; // 24
24          B = 8'b00111100; // 60
25          #2
26          A = 8'b11111111; // 255
27          B = 8'b11111111; // 255
28  end
29
30  initial begin
31      assign result_master = ((A+B)*2+B);
32      $monitor("A=%b,B=%b,Result=%b,Master = %b", A ,B , result, result_master);
33      #10 $stop;
34  end
35
36  endmodule
37
```

Рисунок 20 - разработанный модуль конечного автомата.

В качестве эталонной модели была выбрана обычные встроенные операции сложения и умножения.

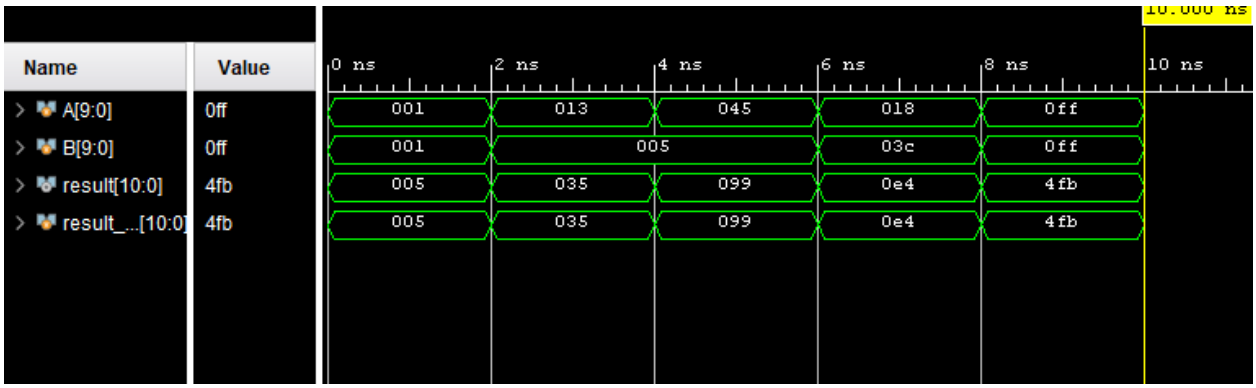


Рисунок 21 – Вывод симуляции тестирования.

```
# run 1000ns
A=0000000001,B=0000000001,Result=00000000101,Master = 00000000101
A=0000010011,B=0000000101,Result=00000110101,Master = 00000110101
A=0001000101,B=0000000101,Result=00010011001,Master = 00010011001
A=0000011000,B=0000111100,Result=00011100100,Master = 00011100100
A=0011111111,B=0011111111,Result=10011111011,Master = 10011111011
$stop called at time : 10 ns : File "C:/Users/Solnyshko/lab2/lab2.srcs/sim_1/new/avtomat_tb.v" Line 33
```

Рисунок 22 - Консольный вывод симуляции тестирования.

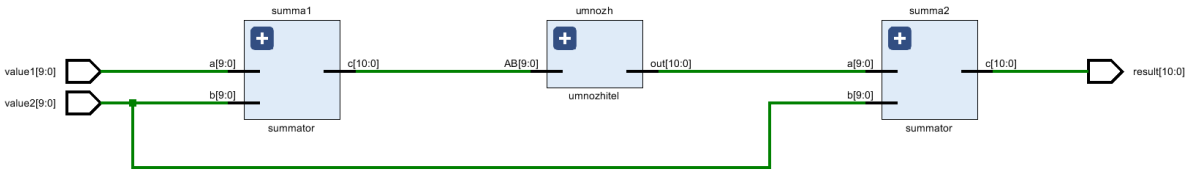


Рисунок 23 – схема конечного автомата.

Для реализации идеи сделать все только через один сумматор и один умножитель можно вручную раскрыть скобки и выполнить операцию умножения для A и B, а затем сложить три слагаемых.

Реализация данной идеи требует переработки сумматора и умножителя.

Сумматор v2

Порт	**Тип**	**Описание**
a	Вх.	Шина первого слагаемого
b	Вх.	Шина второго слагаемого
d	Вх.	Шина третьего слагаемого
c	Вых.	Шина результата

Для реализации сумматора без оператора “+”, было принято реализовать суммирование через сравнивание. Для этого была добавлена переменная для бита переноса. Затем побитное сравнение значений и присвоение соответствующих значений для бита переноса и бита результата.

```
`timescale 1ns / 1ps
module triple_summator(
    input [9:0] a,
    input [9:0] b,
    input [9:0] d,
    output reg [10:0] out
    );
    reg [10:0] c;
    integer perenos = 0;
    integer i,j;
    always@(*) begin
       for(i=0;i<=9;i=i+1) begin
            if (a[i]==0 && b[i]==0)begin
                if ( perenos == 0) begin
                    c[i] <= 0;
                    perenos = 0;
                end
                else if ( perenos == 1) begin
                    c[i] <= 1;
                    perenos = 0;
                end
            end
            else if (a[i] == 1 && b[i] == 1) begin
                if ( perenos == 0) begin
                    c[i] <= 0;
                    perenos = 1;
                end
                else if ( perenos == 1) begin
                    c[i] <= 1;
                    perenos = 1;
                end
            end
            else if (a[i] != b[i]) begin
                if (perenos == 0) begin
                    c[i] <= 1;
                    perenos = 0;
                end
                else if ( perenos == 1) begin
                    c[i] <= 0;
                    perenos = 1;
                end
            end
       end
       if (perenos == 1) begin
            c[10] <= 1;
            perenos = 0;
       end
       else begin
            c[10] <= 0;
```

```
50          end
51        for(j=0;j<=9;j=j+1) begin
52              if (c[j]==0 && d[j]==0)begin
53                  if ( perenos == 0) begin
54                      out[j] <= 0;
55                      perenos = 0;
56                  end
57                  else if ( perenos == 1) begin
58                      out[j] <= 1;
59                      perenos = 0;
60                  end
61              end
62              else if (c[j] == 1 && d[j] == 1) begin
63                  if ( perenos == 0) begin
64                      out[j] <= 0;
65                      perenos = 1;
66                  end
67                  else if ( perenos == 1) begin
68                      out[j] <= 1;
69                      perenos = 1;
70                  end
71              end
72              else if (c[j] != d[j]) begin
73                  if (perenos == 0) begin
74                      out[j] <= 1;
75                      perenos = 0;
76                  end
77                  else if ( perenos == 1) begin
78                      out[j] <= 0;
79                      perenos = 1;
80                  end
81              end
82          end
83          if (perenos == 1) begin
84                out[10] <= 1;
85                perenos = 0;
86          end
87          else begin
88                out[10] <= 0;
89          end
90
91      end
92  endmodule
93
94  ///(A+b)*2+b = 2a+2b+b
```

Рисунок 24 – разработанный модуль сумматора.

```
1   `timescale 1ns / 1ps
2
3   module triple_summator_tb;
4       reg [9:0] a,b,d;
5       wire [10:0] c;
6       reg [10:0] out_putin;
7       triple_summator summa (
8           .a(a),
9           .b(b),
10          .d(d),
11          .out(c)
12      );
13
14  initial begin
15          a = 8'b00000001; // 1
16          b = 8'b00000001; // 1
17          d = 8'b00000001; // 1
18          #2
19          a = 8'b00010011; // 19
20          b = 8'b00000101; // 5
21          d = 8'b01110001; //
22          #2
23          a = 8'b10000101; // 133
24          b = 8'b00000101;  // 5
25          d = 8'b00000101; //
26          #2
27          a = 8'b00011000; // 24
28          b = 8'b11111100; // 252
29          d = 8'b00110001; //
30          #2
31          a = 8'b11111111; // 255
32          b = 8'b11111111; // 255
33          d = 8'b11111111; //
34      end
35
36      initial begin
37          assign out_putin = a + b + d;
38          $monitor("A=%b , B=%b, D,=%b SUM=%b, Sum_master=%b", a, b, d, c, out_putin);
39
40          #10 $stop;
41      end
42
43  endmodule
44
```

Рисунок 25 – тестирование разработанного модуля.

В качестве эталонной модели была выбрана обычная встроенная операция сложения.

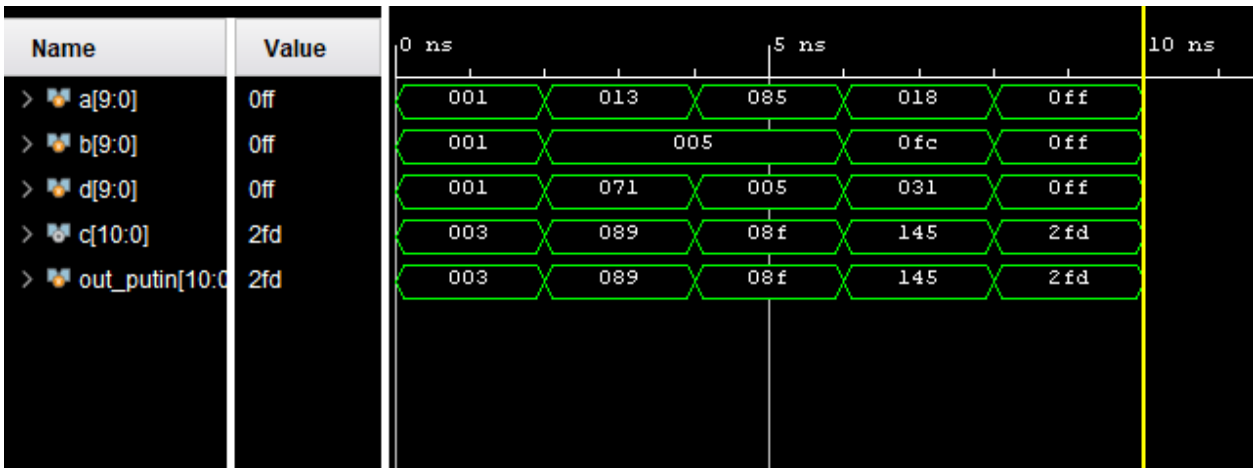


Рисунок 26 – Вывод симуляции тестирования.

```
# run 1000ns
A=0000000001 , B=0000000001, D,=0000000001 SUM=00000000011, Sum_master=00000000011
A=0000010011 , B=0000000101, D,=0001110001 SUM=00010001001, Sum_master=00010001001
A=0010000101 , B=0000000101, D,=0000000101 SUM=00010001111, Sum_master=00010001111
A=0000011000 , B=0011111100, D,=0000110001 SUM=00101000101, Sum_master=00101000101
A=0011111111 , B=0011111111, D,=0011111111 SUM=01011111101, Sum_master=01011111101
$stop called at time : 10 ns : File "C:/Users/Solnyshko/lab2/lab2.srcs/sim_1/new/triple_summator_tb.v" Line 40
```

Рисунок 27 - Консольный вывод симуляции тестирования.

Умножитель v2

Порт	**Тип**	**Описание**
A	Вх.	Шина слагаемого A
B	Вх.	Шина слагаемого B
outA	Вых.	Шина результата A
outB	Вых.	Шина результата B

Для реализации умножителя было принято решение регистрового сдвига влево на 1 бит и присвоение последнему биту значение 0.

```
`timescale 1ns / 1ps
module double_umnozh(
input [9:0] A,
input [9:0] B,
    output reg [10:0] outA,
    output reg [10:0] outB
    );
    always @* begin
        outA <= {A[9:0], 1'b0};
        outB <= {B[9:0], 1'b0};
    end
endmodule

```

Рисунок 28 – разработанный модуль умножителя.

```
`timescale 1ns / 1ps

module Double_umnozh_tb;
    reg [9:0] A,B;
    wire [10:0] outA,outB;
    reg [10:0] outA_master,outB_master;

    double_umnozh umozh(
    .A(A),
    .B(B),
    .outA(outA),
    .outB(outB)
    );

initial begin
           A = 8'b00000001; B = 8'b00000001;
        #1 A = 8'b00000010; B = 8'b00000010;
        #1 A = 8'b00000100; B = 8'b00000100;
        #1 A = 8'b00001000; B = 8'b00001000;
        #1 A = 8'b00010000; B = 8'b00010000;
        #1 A = 8'b00100000; B = 8'b00100000;
        #1 A = 8'b01000000; B = 8'b01000000;
        #1 A = 8'b10000000; B = 8'b10000000;
        #1 A = 8'b00110101; B = 8'b00110101;
        #1 A = 8'b00001111; B = 8'b00001111;
        #1 A = 8'b11111111; B = 8'b11111111;
    end
    initial begin
        assign outA_master = A*2;
        assign outB_master = B*2;

        $monitor("A = %b, B =%b, outA = %b, outB= %b, outA_master= %b, outB_master = %b"
                                ,A,B,outA,outB,outA_master, outB_master);
        #10 $stop;
    end

endmodule

```

Рисунок 29 - тестирование разработанного модуля.

В качестве эталонной модели была выбрана обычная встроенная операция умножения на 2.

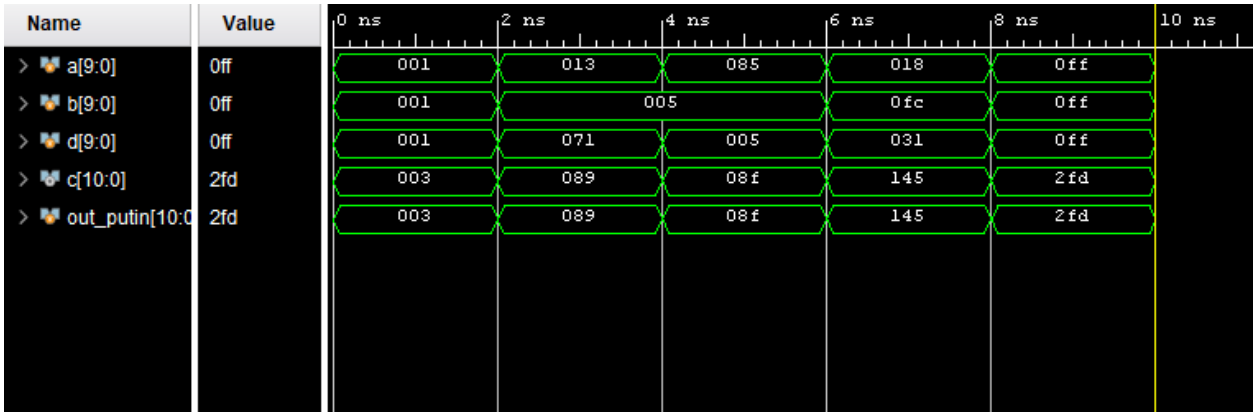


Рисунок 30 – Вывод симуляции тестирования.

```
# run 1000ns
A=0000000001 , B=0000000001, D,=0000000001 SUM=00000000011, Sum_master=00000000011
A=0000010011 , B=0000000101, D,=0001110001 SUM=00010001001, Sum_master=00010001001
A=0010000101 , B=0000000101, D,=0000000101 SUM=00010001111, Sum_master=00010001111
A=0000011000 , B=0011111100, D,=0000110001 SUM=00101000101, Sum_master=00101000101
A=0011111111 , B=0011111111, D,=0011111111 SUM=01011111101, Sum_master=01011111101
$stop called at time : 10 ns : File "C:/Users/Solnyshko/lab2/lab2.srcs/sim_1/new/triple_summator_tb.v" Line 40
```

Рисунок 31 - Консольный вывод симуляции тестирования.

Конечный автомат v2

Порт	**Тип**	**Описание**
A	Вх.	Шина первого слагаемого
B	Вх.	Шина второго слагаемого
result	Вых.	Шина результата

```
1   `timescale 1ns / 1ps
2   module Avtomat_v2_tb;
3           reg [9:0] A,B;
4           wire [10:0] result;
5
6   Avtomat_v2 avt(
7       .value1(A),
8       .value2(B),
9       .result(result)
10  );
11  reg [10:0] result_master;
12
13  initial begin
14          A = 8'b00000001; // 1
15          B = 8'b00000001; // 1
16          #2
17          A = 8'b00010011; // 19
18          B = 8'b00000101; // 5
19          #2
20          A = 8'b01000101; // 69
21          B = 8'b00000101;  // 5
22          #2
23          A = 8'b00011000; // 24
24          B = 8'b00111100; // 60
25          #2
26          A = 8'b11111111; // 255
27          B = 8'b11111111; // 255
28  end
29
30  initial begin
31      assign result_master = ((A+B)*2+B);
32      $monitor("A=%b,B=%b,Result=%b,Master = %b", A ,B , result, result_master);
33      #10 $stop;
34  end
35
36  endmodule
```

Рисунок 32 - разработанный модуль конечного автомата.

В качестве эталонной модели была выбрана обычные встроенные операции сложения и умножения.

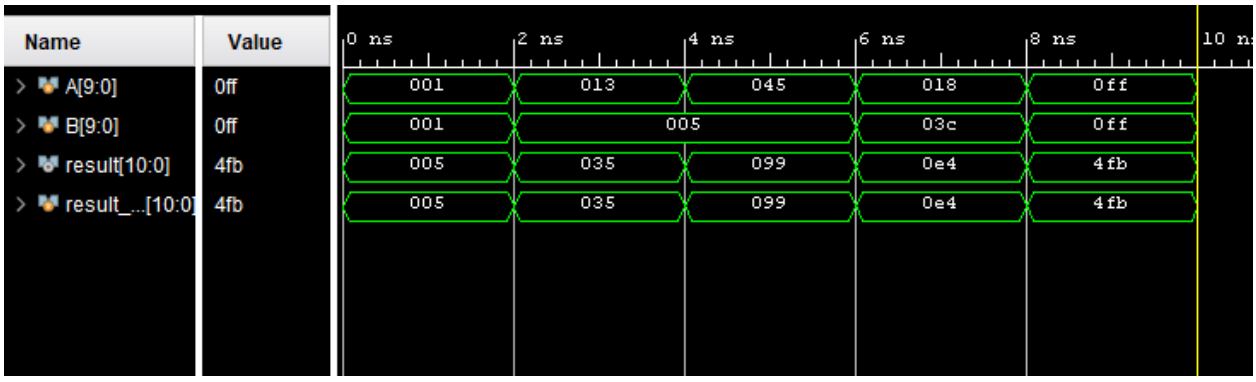


Рисунок 33 – Вывод симуляции тестирования.

```
A=0000000001,B=0000000001,Result=00000000101,Master = 00000000101
A=0000010011,B=0000000101,Result=00000110101,Master = 00000110101
A=0001000101,B=0000000101,Result=00010011001,Master = 00010011001
A=0000011000,B=0000111100,Result=00011100100,Master = 00011100100
A=0011111111,B=0011111111,Result=10011111011,Master = 10011111011
$stop called at time : 10 ns : File "C:/Users/Solnyshko/lab2/lab2.srcs/sim_1/new/Avtomat_v2_tb.v" Line 33
```

Рисунок 34 - Консольный вывод симуляции тестирования.

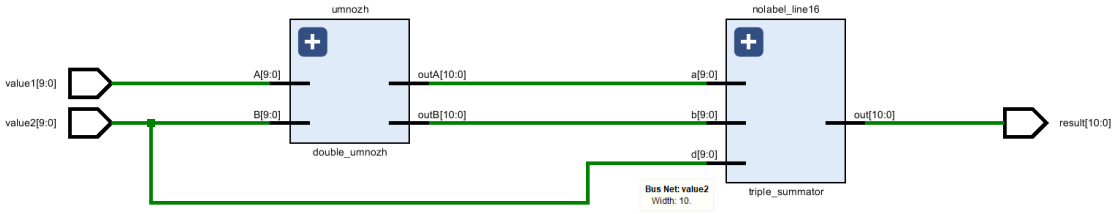


Рисунок 35 – схема конченого автомата через 1 сумматор и 1 умножитель.

Делитель частоты.

Порт	**Тип**	**Описание**
clk	Вх.	Сигнал тактовой частоты
rst_n	Вх.	Сигнал сброса
o_clk	Вых.	Сигнал деленной тактовой частоты

```
1  `timescale 1ns / 1ps
2
3  module Frequency_Divider(
4      input clk,
5      input rst_n, //импульс сброса
6      output reg o_clk
7      );
8   reg [1:0] cnt;
9
10 always@(posedge clk or negedge rst_n) begin
11   if (!rst_n)
12     cnt <= 0;
13   else if (cnt == 3) // 0 ~ 3
14     cnt <= 0;
15   else
16     cnt <= cnt + 1;
17 end
18
19 always@(posedge clk or negedge rst_n) begin
20   if (!rst_n)
21     o_clk <= 0;
22   else if (cnt < 2) // 0 ~ 1
23     o_clk = 0;
24   else              // 2 ~ 3
25     o_clk = 1;
26 end
27
28 endmodule
29
```

Рисунок 36 - разработанный модуль делителя частоты.

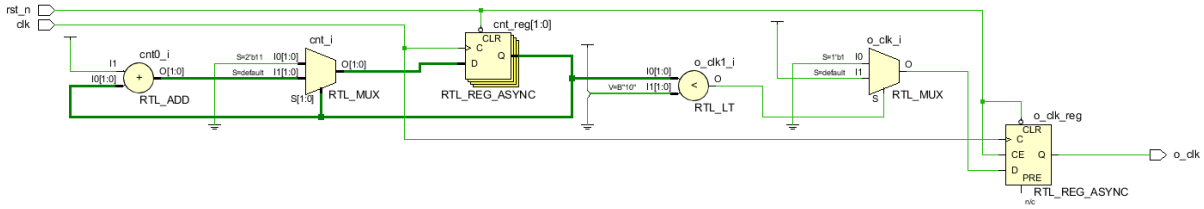


Рисунок 37 – схема разработанного модуля

```
1   `timescale 1ns / 1ps
2
3   module Frequency_Divider_tb;
4   reg clk;
5   reg rst_n;
6   wire o_clk;
7   reg master_clk;
8
9   Frequency_Divider u0 (
10    .clk(clk),
11    .rst_n(rst_n),
12    .o_clk(o_clk)
13  );
14
15  initial begin
16    clk   = 1'b0;
17    rst_n = 1'b0;
18    #1 rst_n = 1'b1;
19  end
20
21  // 50MHz clk
22  initial begin
23      clk = 1'b0;
24      repeat(100) #1 clk= ~clk;
25      end
26   initial begin
27      #1
28      master_clk = 1'b0;
29      repeat(100) #4 master_clk = ~master_clk;
30      end
31  initial begin
32      $monitor("clk=%b,rst_n=%b,o_clk=%b", clk ,rst_n , o_clk);
33      #100 $stop;
34  end
35
36  endmodule
```

Рисунок 37 - тестирование разработанного модуля.

В качестве эталонной модели был запущен тактовый сигнал в 4 раза больший, нежели изначальный.

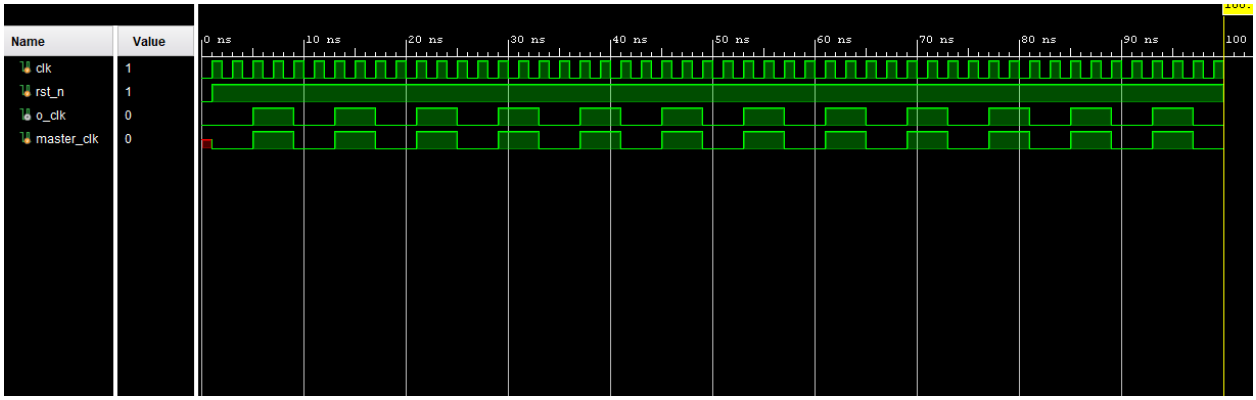


Рисунок 38 – Вывод симуляции тестирования.

```
# run 1000ns
clk=0,rst_n=0,o_clk=0
clk=1,rst_n=1,o_clk=0
clk=0,rst_n=1,o_clk=0
clk=1,rst_n=1,o_clk=0
clk=0,rst_n=1,o_clk=0
clk=1,rst_n=1,o_clk=1
clk=0,rst_n=1,o_clk=1
clk=1,rst_n=1,o_clk=1
clk=0,rst_n=1,o_clk=1
clk=1,rst_n=1,o_clk=0
clk=0,rst_n=1,o_clk=0
clk=1,rst_n=1,o_clk=0
clk=0,rst_n=1,o_clk=0
clk=1,rst_n=1,o_clk=1
clk=0,rst_n=1,o_clk=1
clk=1,rst_n=1,o_clk=1
clk=0,rst_n=1,o_clk=1
clk=1,rst_n=1,o_clk=0
clk=0,rst_n=1,o_clk=0
clk=1,rst_n=1,o_clk=0
clk=0,rst_n=1,o_clk=0
clk=1,rst_n=1,o_clk=1
clk=0,rst_n=1,o_clk=1
clk=1,rst_n=1,o_clk=1
clk=0,rst_n=1,o_clk=1
clk=1,rst_n=1,o_clk=0
clk=0,rst_n=1,o_clk=0
clk=1,rst_n=1,o_clk=0
```

Рисунок 39 - Консольный вывод симуляции тестирования.

Функция “COUNT_ZEROES”

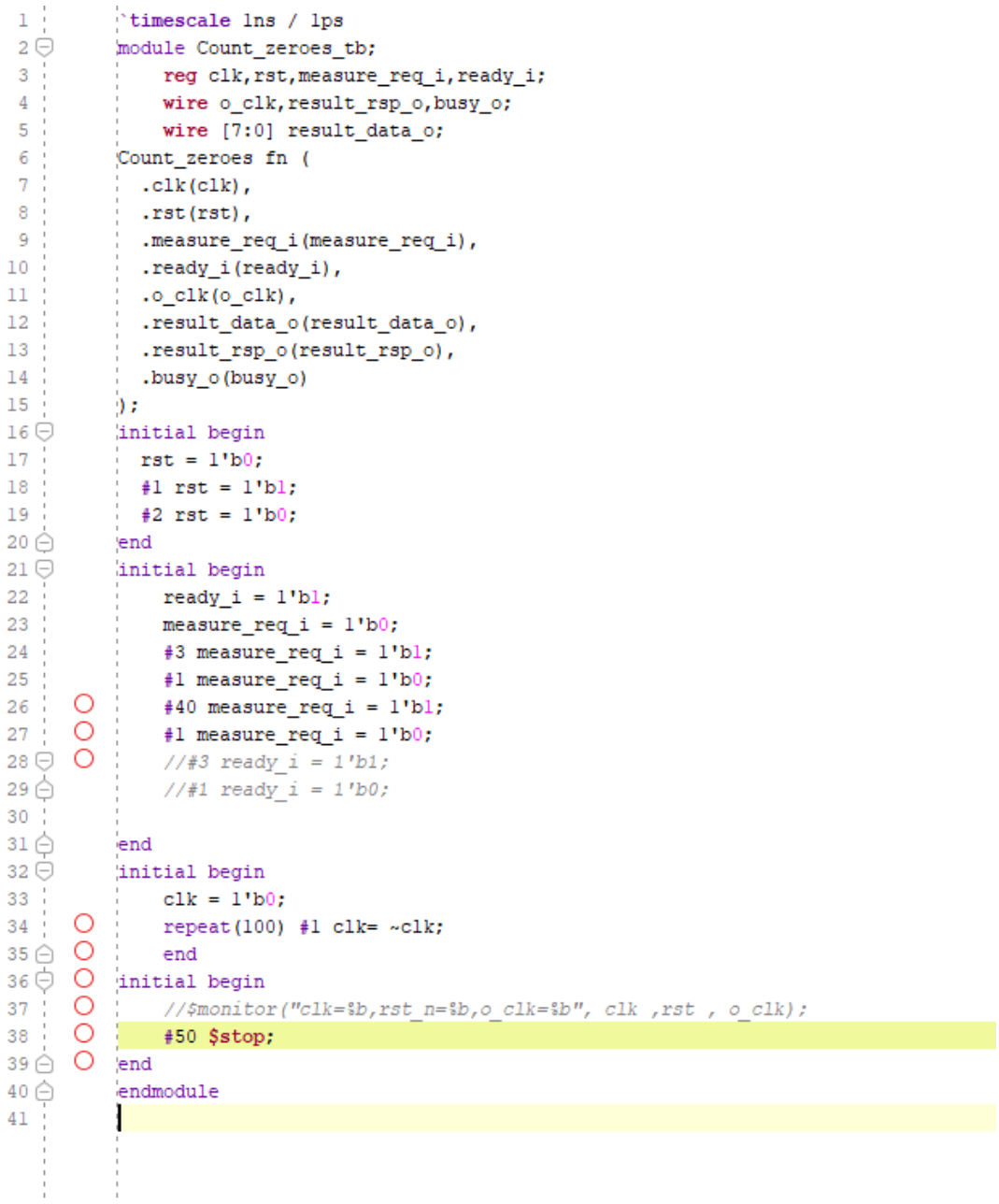
За заданный промежуток времени необходимо рассчитать количество спадов тактового сигнала из делителя частоты. После расчёта на выходные порты необходимо подать число нулей в двоичном представлении рассчитанного числа.

Порт	Тип	Описание
clk	Вх.	Сигнал тактовой частоты.
rst	Вх.	Асинхронный сигнал сброса
measure_req_i	Вх.	Сигнал регулирующий временной промежуток. При подаче в первый раз запускает временной

		отсчёт. При подаче во второй раз останавливает временной отсчёт.
ready_i	Вх.	Сигнал готовности внешнего устройства принять результат
o_clk	Вых.	*Необязательный* выход тактового сигнала из делителя частоты
result_data_o	Вых.	Шина данных результата
result_rsp_o	Вых.	Сигнал готовности результата. Выставляется в высокий уровень тогда и только тогда, когда на шине данных установлен корректный результат. Держится в высоком состоянии ровно один период тактового сигнала, когда внешнее устройство готово принять результат; в противном случае держится в высоком состоянии до тех пор, пока устройство не будет готово принять результат
busy_o	Вых.	Сигнал занятости устройства

```
1   `timescale 1ns / 1ps
2
3   module Count_zeroes(
4       input clk,              ///Сигнал тактовой частоты
5       input rst,                ///Асинхронный сигнал сброса
6       input measure_req_i,
7       input ready_i,
8       output wire o_clk,      ///необязательный выход (для наглядности)
9       output wire [7:0] result_data_o,
10      output reg result_rsp_o,
11      output reg busy_o
12      );
13      reg lets_count;
14
15      Frequency_Divider delitel(
16          .clk(clk),
17          .rst_n(rst),
18          .o_clk(o_clk)  //переводит изначальную тактовую частоту в тактовую частоту делителя
19      );
20      simple_counter count(
21          .clk(o_clk),
22          .rst(rst),
23          .ud(lets_count),
24          .count(result_data_o)
25
26      );
27      always@(*)begin
28          if (result_rsp_o == ready_i) begin
29              #1
30              result_rsp_o <=0;
31          end
32
33      end
34      always@(posedge measure_req_i) begin
35          if(measure_req_i && lets_count == 1) begin
36              lets_count <=0;
37              busy_o <= 0;
38              result_rsp_o <= 1;
39
40          end
41          else begin
42              lets_count <=1;
43              busy_o <=1;
44              result_rsp_o <= 0;
45          end
46      end
47  endmodule
48
```

Рисунок 40 – разработанный модуль функции COUNT_ZEROES.



```
`timescale 1ns / 1ps
module Count_zeroes_tb;
    reg clk,rst,measure_req_i,ready_i;
    wire o_clk,result_rsp_o,busy_o;
    wire [7:0] result_data_o;
Count_zeroes fn (
  .clk(clk),
  .rst(rst),
  .measure_req_i(measure_req_i),
  .ready_i(ready_i),
  .o_clk(o_clk),
  .result_data_o(result_data_o),
  .result_rsp_o(result_rsp_o),
  .busy_o(busy_o)
);
initial begin
  rst = 1'b0;
  #1 rst = 1'b1;
  #2 rst = 1'b0;
end
initial begin
    ready_i = 1'b1;
    measure_req_i = 1'b0;
    #3 measure_req_i = 1'b1;
    #1 measure_req_i = 1'b0;
    #40 measure_req_i = 1'b1;
    #1 measure_req_i = 1'b0;
    //#3 ready_i = 1'b1;
    //#1 ready_i = 1'b0;

end
initial begin
    clk = 1'b0;
    repeat(100) #1 clk= ~clk;
    end
initial begin
    //$monitor("clk=%b,rst_n=%b,o_clk=%b", clk ,rst , o_clk);
    #50 $stop;
end
endmodule
```

Рисунок 41 - тестирование разработанного модуля.

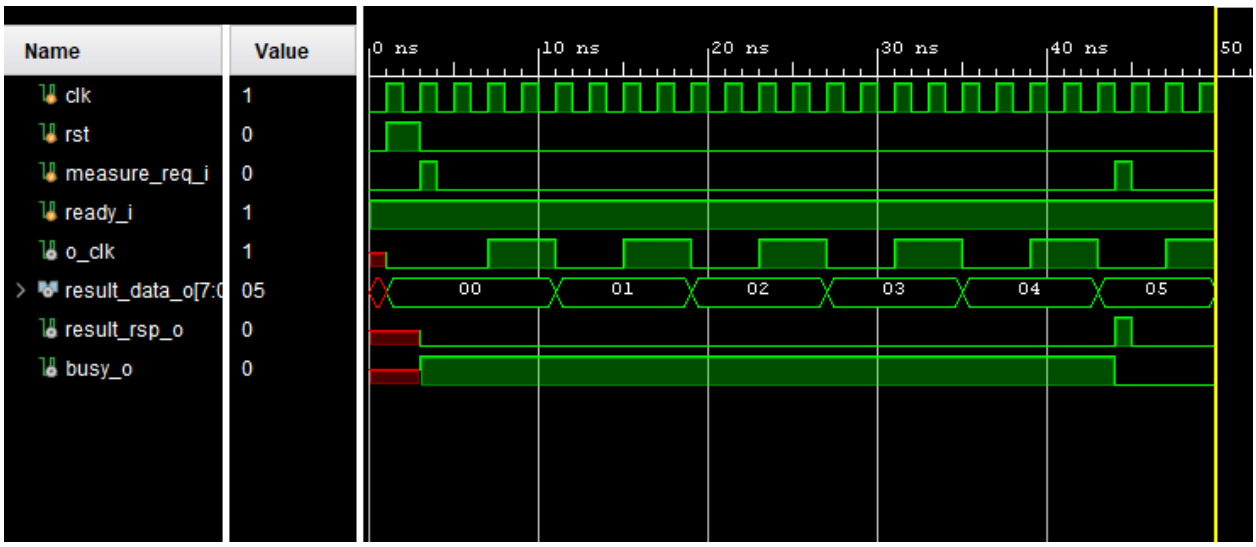


Рисунок 42 – Вывод симуляции тестирования.

На временной диаграмме видно, что подсчёт идёт со спада тактовой частоты делителя частоты, также работа счетчика подкрепляется сигналом занятости и значение о корректности и готовности результата идет ровно один такт, что полностью соответствует заданию.

Часть 2

В рамках второй части задания необходимо реализовать модуль, выполняющий определенную процедуру обработки входного потока данных. Все внутренние элементы памяти вашего модуля должны тактироваться от делителя частоты и обладать сигналами разрешения и асинхронного сброса.

Модуль “BUFFER_LRU”

Модуль-буффер из восьми элементов, в котором вытеснение происходит по алгоритму LRU. Интерфейс подачи транзакции должен включать в себя шину данных и соответствующий ей сигнал валидности данных. Выходной

интерфейс представляет собой текущее состояние всех восьми элементов буффера.

Для реализации данного модуля был воссоздан LRU (Least Recently Used) алгоритм основан на следующем принципе: он отслеживает время последнего обращения к каждому элементу данных и заменяет элемент, к которому было обращение давнее всего, когда требуется освободить место для нового элемента.

Для того, чтобы все внутренние элементы памяти модуля тактировались от делителя частоты, был использован Делитель частоты из первой части лабораторной работы.

```
12      Frequency_Divider delitel ( // делитель частоты
13        .clk(clk),
14        .rst_n(rst),
15        .o_clk(o_clk)
16      );
17
```

Рисунок 43 – Делитель частоты в модуле LRU.

Для воссоздания таймера к каждой ячейке буффера был привязан свой собственный Counter из первой части лабораторной работы (таких 8).

```
simple_counter count1( //инициализация таймеров на каждую переменную буфера
    .clk(o_clk),
    .rst(rst1),
    .ud(razresh),
    .count(coun1)
);
simple_counter count2(
    .clk(o_clk),
    .rst(rst2),
    .ud(razresh),
    .count(coun2)
);
simple_counter count3(
    .clk(o_clk),
    .rst(rst3),
    .ud(razresh),
    .count(coun3)
);
```

Рисунок 44 – Назначения тикового таймера.

Порт	**Тип**	**Описание**
clk	Вх.	Сигнал тактовой частоты.
rst	Вх.	Асинхронный сигнал сброса
razresh	Вх.	Сигнал разрешения.
data	Вх.	Шина входящих данных
o_clk	Вых.	Необязательный. Показывает сигнал тактовой частоты с делителя частоты.
d1-d8	Вых.	Необязательный. Показывает записанные значения в буффер.
coun1-coun8	Вых.	Необязательный. Показывает таймер на каждое значение.
result	Вых.	Выходной интерфейс представляет собой текущее состояние (таймера) всех восьми элементов буффера, где каждый элемент это каждые 8 бит.

```
1   `timescale 1ns / 1ps
2
3   module LRU_Buffer_tb;
4       reg clk; //takt signal
5       reg rst; // асинхронный сигнал сброса
6       reg razresh;
7       wire o_clk;
8       reg [7:0] data;
9       wire [7:0] d1,d2,d3,d4,d5,d6,d7,d8;
10      wire [7:0] coun1,coun2,coun3,coun4,coun5,coun6,coun7,coun8;
11      wire [63:0] result;
12      LRU_Buffer b1(
13          .clk(clk),
14          .rst(rst),
15          .razresh(razresh),
16          .o_clk(o_clk),
17          .data(data),
18          .d1(d1),.d2(d2),.d3(d3),.d4(d4),.d5(d5),.d6(d6),.d7(d7),.d8(d8),
19          .coun1(coun1),.coun2(coun2),.coun3(coun3),.coun4(coun4),.coun5(coun5),.coun6(coun6),.coun7(coun7),.coun8(coun8),
20          .result(result)
21          );
22      initial begin
23          clk = 1'b0;
24          repeat(200) #1 clk= ~clk;
25      end
26      initial begin
27          razresh = 1'b1;
28          rst=1'b0;
29          #1 rst = 1'b1;
30          #1 rst = 1'b0;
31      end
32      initial begin
33              data = 8'b00000000;
34          #7 data = 8'b00000001;
35          #8 data = 8'b00000010;
36          #8 data = 8'b00000100;
37          #8 data = 8'b00001000;
38          #8 data = 8'b00010000;
39          #8 data = 8'b00100000;
40          #8 data = 8'b01000000;
41          #8 data = 8'b10000000;
42          #8 data = 8'b10000001;
43          #8 data = 8'b00000010;
44          #8 data = 8'b00000100;
45          #8 data = 8'b00001000;
46          #8 data = 8'b00010000;
47          #8 data = 8'b00100000;
48          #8 data = 8'b01000000;
49          #8 data = 8'b10000000;
50          #8 data = 8'b00000001;

51          #8 data = 8'b00000010;
52          #8 data = 8'b00000100;
53          #8 data = 8'b00001000;
54          #8 data = 8'b00010000;
55          #8 data = 8'b00100000;
56          #8 data = 8'b01000000;
57          #8 data = 8'b10000000;
58      end
59      initial begin
60          #200 $stop;
61      end
62  endmodule
```

Рисунок 45 - тестирование разработанного модуля.

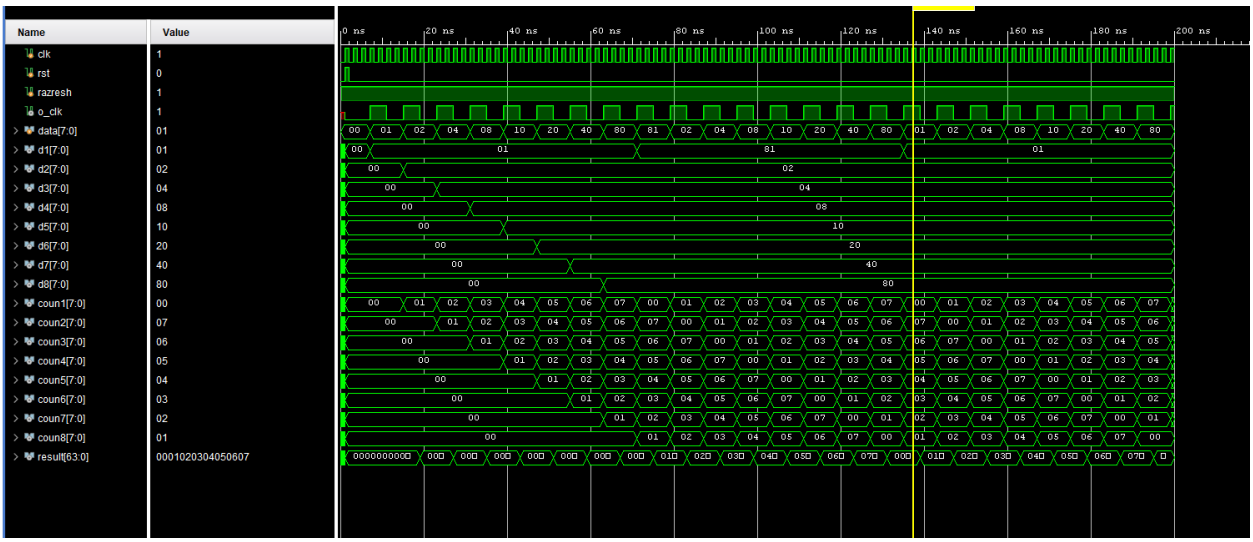


Рисунок 46 - Вывод симуляции тестирования.

Заключение

В результате выполнения лабораторной работы были выявлены некоторые трудности тестирования.

Для тестирования счётчика не было придумано никаких эталонных моделей, потому что сам счётчик работает от тактового сигнала и каждый сигнал на вход поступает несколько дополнительных сигналов или же настроек для модуля. Следовательно, и эталонная модель должна принимать в себя все те же самые сигналы, для корректного результата. Считаю эталонную модель для счётчика ненужной, потому что эталонная модель будет полностью или частично копировать написанную. (Исправлено)

В тестировании сдвигового регистра была аналогичная ситуация, что и с счётчиком. В принципе для эталонной модели можно было бы сделать обычный остаток от деления на два, но опять же, сдвиговый регистр работает на каждый такт, к тому же имеет в себе несколько сигналов, таких как сброс, загрузка данных и сигнал разрешения. По сути, сама сложность написания сдвигового регистра заключалась в том, чтобы придумать каким способом

можно сдвинуть регистр и при этом сохранить сдвинутый бит. Решение данной проблемы может быть простое копирования модуля сдвигового регистра с изменением одной строчки на остаток от деления. Было принято решение не делать из мухи слона. Однако, если бы это было сделано, то не был бы написан этот текст. (Исправлено)

Создание конечного автомата было занятием интересным. Там мне не надо было делать дополнительные ненужные модули, потому что математику можно было написать и в самом тестовом окружении. Проблема возникла тогда, когда был написан конечный автомат v1, потому что, посмотрев на RTL схему, сумматора оказалось 2, а в задании должен был быть задействован лишь 1. Решением данной проблемы оказалось созданием сумматора с тремя входами и умножителем с двумя входами.

С Делителем частоты проблем не возникло, ведь за эталонную модель можно было запустить еще один тактовый сигнал превосходящий начальный в четыре раза.

Первая функция — это первая серьезная сложность данной лабораторной работы, потому что функция должна включать в себя компоненты, написанные ранее, а они не всегда были подходящими, а значит их надо было изменять и заново тестировать, потому что хотелось быть уверенным, что ошибки в функции не были связаны с ошибками в маленьких, написанных ранее компонентах. После написания функции, ручной проверкой было выявлено, что функция работает корректно. Эталонная модель будет представлять обычный инкремент при нуле делителя тактовой частоты.

В результате создания сложно-функционального блока я столкнулся с множеством проблем, в плане реализации высокоуровневой задачи. Как передавать и запоминать правильно данные, как запускать и обнулять таймер каждый вызов записанного выражения, все эти вопросы так или инче влияли на всё то, что было в итоге написано. К подходу создания с нуля надо

подходить ответственно, потому что первые два раза, когда я думал, что сейчас все запишу и оно сразу всё заработает, тестовая модель валилась на этапе делителя частоты. Следовательно создание такой сложной функции надо начинать, поэтапно вводя новые переменные и новые методы.

После создания сложно-функционального блока, было визуально протестировано, на соответствие введённых данных и порядку записи. Тестирование эталонной моделью не представляю возможным. Если бы Verilog HDL имел функционал других языков программирования, с использованием массивов, листов или методов insert, append, то создание эталонной моделью было бы не таким сложным. В то время, как для написании её в нынешних реалиях нужно задуматься над тем, как сделать так, чтобы новая модель имела такой же функционал как и написанная, но использовала другие алгоритмы. Также, я отказался вставлять RTL схемы сложно-функционального блока и функции, так как, было использовано множество операторов if(), из-за чего, схема была перегружна. Возможно, решением этой проблемы, если её можно считать таковой, была бы замена операторов на базовые логические операции с записью значений в некую переменную, но опять же, нужно подумать, как бы сделать так, чтобы обойтись без оператора if().