Федеральное государственное автономное образовательное учреждение высшего образования "Национальный Исследовательский Университет ИТМО"
Мегафакультет Компьютерных Технологий и Управления
Факультет Программной Инженерии и Компьютерной Техники

Вариант №4
Лабораторная работа 2

по дисциплине

'Функциональная схемотехника'

Выполнил Студент группы P33102
Лапин Алексей Александрович
Преподаватель:
Васильев С.Е.

г. Санкт-Петербург
2024г.

Содержание

1 Задание

1.1 Описание лабораторной работы

Лабораторная работа №2 посвящена проектированию последовательностной логики на уровне регистровых передач с использованием языка описания аппаратуры Verilog HDL.

В первой части работы предлагается разработать несколько простых блоков цифровой последовательностной логики и объединить их для выполнения заданной функции в одно функционирующее устройство.

Во второй части работы предлагается разработать устройство, управляющее входным потоком данных с помощью одного из указанных алгоритмов обработки.

1.2 Таблица варианта

Вариант	Функция 1	FSM	Функция 2	Разряд ности	Делитель частоты
4	COUNT_FREE	FSM_1	FIFO	32 бит	10

2 Выполнение

2.1 Счётчик

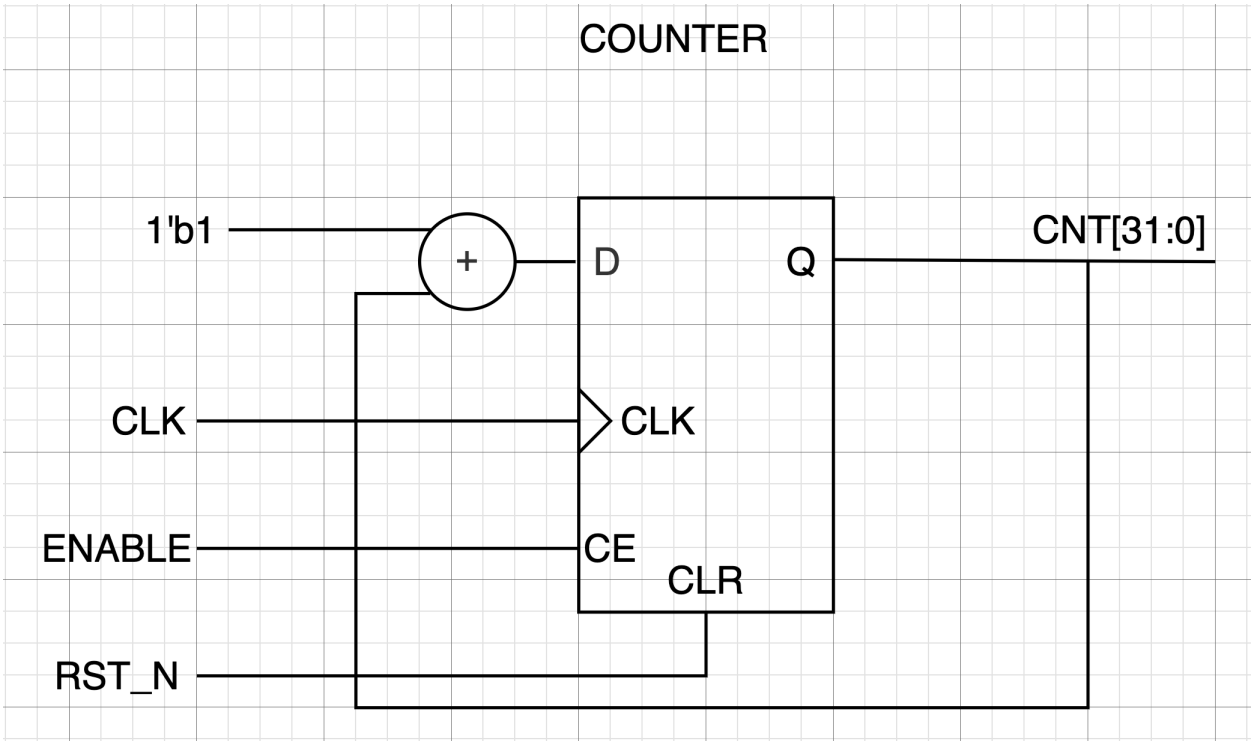

Рис. 1: Синхронный счетчик по переднему фронту с асинхронным сбросом и сигналом разрешения, 32 разряда.

Счетчик тактируется от CLK по переднему фронту. Когда уровень enable высокий, то счетчик инкрементируется, когда уровень низкий — счетчик сохраняет свое значение с предыдущего такта.

По заднему фронту rst_n счетчик асинхронно сбрасывается в 0.

На выходе счетчика по переднему фронту выставляется значение $Q(t-1)+1$ по модулю разрядности счетчика.

2.1.1 Разработанный модуль

```
module counter
#(
    parameter WIDTH = 32
)
(
    input clk,
    input rst_n,
    input enable,
    output reg [WIDTH - 1:0] cnt
);
    always @(posedge clk or negedge rst_n) begin
        if(!rst_n)
            cnt <= {WIDTH{1'b0}};
        else if (enable)
            cnt <= cnt + 1'b1;
        else
            cnt <= cnt;
    end
endmodule
```

2.1.2 Тестовый план:

Области эквивалентности:

- enable = 1, rst_n = 1 – инкремент
- enable = 1, rst_n = 0 – сброс
- enable = 0, rst_n = 1 – сохранение значения

```
`timescale 1ns/10ps
`include "counter/src/counter.v"

module counter_tb;
    localparam WIDTH = 32;

    reg clk;
    reg rst_n;
    reg enable;
    wire [WIDTH-1:0] cnt;

    reg [WIDTH-1:0] tst_out;

    reg passed = 1'b1;

    counter #(.WIDTH(WIDTH)) counter_dut(
        .clk (clk),
        .rst_n (rst_n),
```

```
19          .enable (enable),
20          .cnt (cnt)
21      );
22
23      /* Clock generation */
24      initial begin
25          clk = 0;
26          forever #10 clk = !clk;
27      end
28
29      task check;
30          begin
31              if(tst_out != cnt) begin
32                  $display("[T=%0g] Test failed: expected %g, got %g",
                        $time, tst_out, cnt);
33                  passed = 1'b0;
34              end
35          end
36      endtask
37
38      integer i;
39      initial begin
40          rst_n = 0;
41          enable = 1;
42          #1 rst_n = 1;
43
44          tst_out = {WIDTH{1'b0}};
45
46          /* Test 1: Count to 10 */
47          for(i = 1; i < 10; i = i + 1) begin
48              @(posedge clk) begin
49                  tst_out = i;
50              end
51              @(negedge clk) check;
52          end
53
54
55          /* Test 2: Reset, Should Be 0 */
56          @(posedge clk) begin
57              rst_n <= 0;
58              tst_out <= 0;
59          end
60          @(negedge clk) check;
61
62          rst_n <= 1;
63
64          /* Test 3: Count to 3 */
65          for(i = 1; i < 3; i = i + 1) begin
66              @(posedge clk) begin
67                  tst_out = i;
68              end
69              @(negedge clk) check;
70          end
```

```
71
72          /* Test 4: Enable = 0, Should Be 3 */
73          @(posedge clk) begin
74              enable <= 0;
75              tst_out <= 3;
76          end
77          repeat(10) @(negedge clk) check;
78
79          /* Test 5: Continue, Enable = 1, Should Be 17 */
80          @(posedge clk) begin
81              enable <= 1;
82              tst_out <= 3;
83          end
84          for(i = 4; i < 17; i = i + 1) begin
85              @(posedge clk) begin
86                  tst_out = i;
87              end
88              @(negedge clk) check;
89          end
90
91          if(passed)
92              $display("[T=%0g] All tests passed", $time);
93          else
94              $display("[T=%0g] Some tests failed", $time);
95          #10; $finish;
96      end
97
98      initial begin
99          $dumpfile("build/counter.vcd");
100         $dumpvars(1);
101     end
102
103
104
105 endmodule
```

Результаты тестирования:

```
[T=720] All tests passed
```

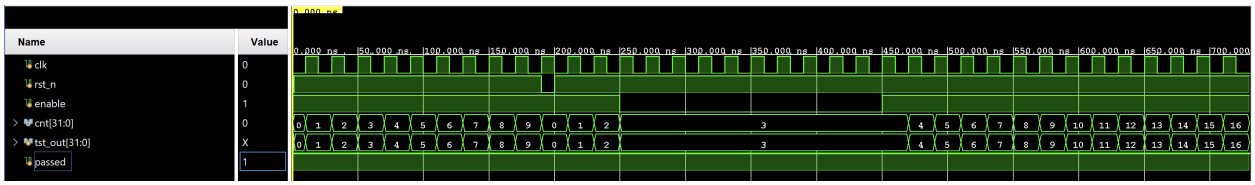

Рис. 2: Временная диаграмма работы счетчика

2.2 Сдвиговый регистр

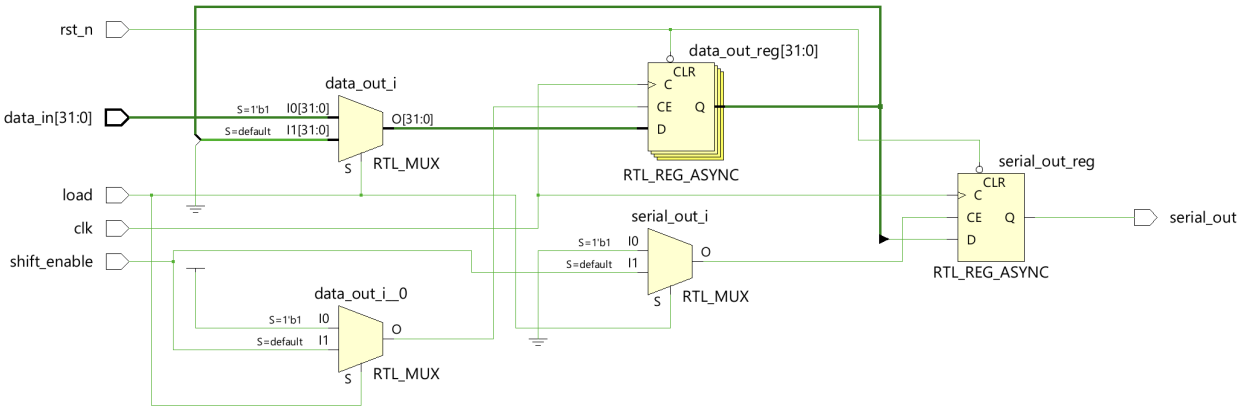

Рис. 3: Схема сдвигового регистра с параллельной загрузкой и последовательным сдвигом вправо

Сдвиговый регистр с параллельной загрузкой и последовательным сдвигом вправо. Каждый фронт тактового сигнала, при наличии активного сигнала разрешения, выполняется операция сдвига вправо. При наличии активного сигнала загрузки, в регистр загружается значение с входа D. По заднему фронту сигнала сброса, регистр сбрасывается в 0.

2.2.1 Разработанный модуль

```
1  /* 32 - bit Right Shift register with shift_enable */
2  module shift_right
3  #(
4      parameter WIDTH = 32
5  )
6  (
7      input clk,
8      input rst_n,
9      input [WIDTH - 1:0] data_in,
10     input shift_enable,
11     input load,
12     output reg serial_out
13 );
14
15     reg [WIDTH - 1:0] data_out;
16     always @(posedge clk or negedge rst_n) begin
17         if (!rst_n)
18             {serial_out, data_out} <= {1'b0, {WIDTH-1{1'b0}}};
19         else if(load)
20             data_out <= data_in;
21         else if (shift_enable)
22             {data_out[WIDTH-1:0], serial_out} <= {1'b0,
                   data_out[WIDTH-1:0]};
23         else
24             data_out <= data_out;
```

```
    end

endmodule
```

2.2.2 Тестовый план:

1. Протестировать сдвиг при пустом буффере.
2. Протестировать сброс регистра.
3. Протестировать загрузку значения в регистр.
4. Протестировать сдвиг данных после сброса.

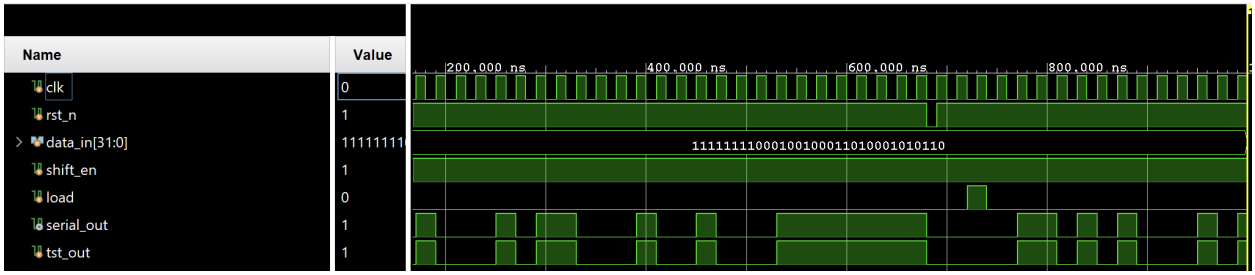

Рис. 4: Временная диаграмма работы сдвигового регистра

```
`timescale 1ns/100ps
`include "shift_right/src/shift_right.v"

module shift_right_tb;
    localparam WIDTH=32;

    reg clk;
    reg rst_n;
    reg [WIDTH-1:0] data_in;
    reg shift_en;
    reg load;
    wire serial_out;

    reg passed = 1;

    reg tst_out;
    reg [WIDTH-1:0] tst_temp;
    reg [WIDTH-1:0] tst_value = 32'hFF123456;

    shift_right #(.WIDTH(WIDTH)) shift_right_dut (
        .clk(clk),
        .load(load),
        .rst_n(rst_n),
        .data_in(data_in),
        .shift_enable(shift_en),
        .serial_out(serial_out)
    );

```

```
29      /* Clock generation */
30      initial begin
31        clk = 0;
32        forever #10 clk = ~clk;
33      end
34
35      task check;
36          begin
37              if(tst_out !== serial_out) begin
38                  $display("[T=%0g] Test failed: expected %g, got %g",
                        $time, tst_out, serial_out);
39                  passed = 0;
40              end
41          end
42      endtask
43
44      integer i;
45
46      initial begin
47          $dumpfile("build/shift_right.vcd");
48          $dumpvars(1);
49          $display("#################### Starting simulation
                    ####################");
50
51          shift_en = 0;
52          rst_n = 1;
53          load = 0;
54
55          tst_temp = tst_value;
56
57          @(negedge clk) begin
58              data_in <= tst_value;
59              load <= 1;
60          end
61          @(negedge clk) begin
62              load <= 0;
63          end
64          shift_en <= 1;
65
66          /* Test: Load value */
67          $display("########## TEST: Load value ##########");
68          for (i = 0; i < 32; i = i + 1) begin
69              @(posedge clk)
70                  {tst_temp[WIDTH-1:0], tst_out} <= {1'b0,
                        tst_temp[WIDTH-1:0]};
71              @(negedge clk) check;
72          end
73
74          /* Test: Reset, Should Be 0 */
75          $display("########## TEST: Reset ##########");
76          rst_n = 0;
77          tst_out = 0;
78          @(posedge clk) check;
```

```
79          rst_n = 1;
80
81          /* Test: Buffer is empty */
82          $display("########## TEST: Buffer is empty ##########");
83          @(posedge clk) check;
84
85          /* Test: Shift right continue */
86          $display("########## TEST: Shift right continue ##########");
87          @(negedge clk) begin
88              tst_temp <= tst_value;
89              data_in <= tst_value;
90              load <= 1;
91          end
92          @(negedge clk) begin
93              load <= 0;
94          end
95          shift_en <= 1;
96
97          for (i = 0; i < 16; i = i + 1) begin
98              @(posedge clk)
99                  {tst_temp[WIDTH-1:0], tst_out} <= {1'b0,
                       tst_temp[WIDTH-1:0]};
100             @(negedge clk) check;
101         end
102         if(passed)
103             $display("[T=%0g] All tests passed", $time);
104         else
105             $display("[T=%0g] Some tests failed", $time);
106         #10; $finish;
107     end
108
109
110 endmodule
```

Результаты тестирования:

```
########################## Starting simulation ##########################
############ TEST: Load value ############
############ TEST: Reset ############
########## TEST: Buffer is empty ##########
########## TEST: Shift right continue ##########
[T=1060] All tests passed
```

2.3 Конечный автомат

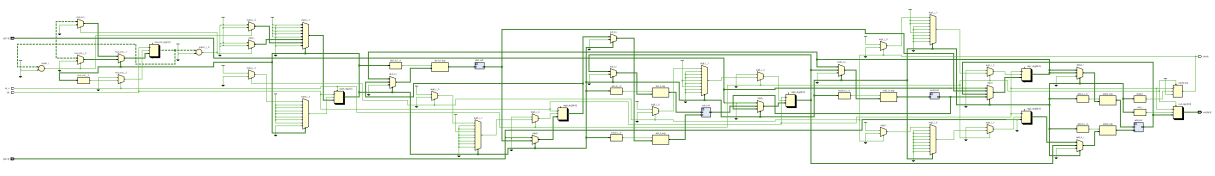

Рис. 5: Схема конечного автомата

Конечный автомат состоит из 7 состояний, которые соответвуют последовательным этапам вычисления функции $(A/2 + B) * 8 + (A - B/2) * 4$. Чтобы автомат перешел в начальное состояние, необходимо подать сигнал сброса rst. Когда автомат закончит работу он устанавливает сигнал ready в 1 и переходит в первое состояние, где ожидает сигнала сброса.

2.3.1 Разработанный модуль

Сумматор

```
‘timescale 1ps/1ps
module adder
#(
    parameter WIDTH = 32
)
(
    input [WIDTH - 1:0] x, y,
    output [WIDTH - 1:0] z
);
    assign z = x + y;
endmodule
```

Делитель на 2

```
‘timescale 1ps/1ps
module div2
#(
    parameter WIDTH = 32
)
(
    input [WIDTH - 1:0] in,
    output [WIDTH - 1:0] out
);
    assign out = in >> 1;
endmodule
```

Умножитель на 8

```
‘timescale 1ps/1ps
module mul2
#(
    parameter WIDTH = 32
)
(
    input [WIDTH - 1:0] in,
    output [WIDTH - 1:0] out
);
    assign out = in << 1;
endmodule
```

Вычитатель

```
‘timescale 1ps/1ps
module sub
#(
    parameter WIDTH = 32
```

```
)
(
    input [WIDTH - 1:0] x, y,
    output [WIDTH - 1:0] z
);
    assign z = x + (~y) + 1;

endmodule
```

Конечный автомат

```
`include "fsm/src/sub.v"
`include "fsm/src/adder.v"
`include "fsm/src/mul2.v"
`include "fsm/src/div2.v"
module fsm
#(
    parameter WIDTH = 32,
    parameter OP_WIDTH = WIDTH + 3
)
(
    input clk,
    input [WIDTH - 1:0] a, b,
    input rst_n,
    output reg[OP_WIDTH-1:0] out,
    output reg ready
);

    localparam [3:0]
        S0 = 4'b0000,
        S1 = 4'b0001,
        S2 = 4'b0010,
        S3 = 4'b0011,
        S4 = 4'b0100,
        S5 = 4'b0101,
        S6 = 4'b0110,
        S7 = 4'b0111;
    reg [3:0] state;

    /* registers to store intermediate results */
    reg[OP_WIDTH - 1:0] reg1, reg2, reg3, reg4;

    reg[1:0] mul_cnt = 0;

    /* instantiate div2 module */
    reg [OP_WIDTH - 1:0] div2_in;
    wire [OP_WIDTH - 1:0] div2_out;
    div2 #(.WIDTH(OP_WIDTH)) div2_inst (
        .in(div2_in),
        .out(div2_out)
    );

    /* instantiate adder module */
    reg [OP_WIDTH - 1:0] add_a, add_b;
```

```
44      wire [OP_WIDTH - 1:0] add_z;
45      adder #(.WIDTH(OP_WIDTH)) add_inst (
46          .x(add_a),
47          .y(add_b),
48          .z(add_z)
49      );
50
51      /* instantiate mul2 module */
52      reg [OP_WIDTH - 1:0] mul2_in;
53      wire [OP_WIDTH - 1:0] mul2_out;
54      mul2 #(.WIDTH(OP_WIDTH)) mul2_inst (
55          .in(mul2_in),
56          .out(mul2_out)
57      );
58
59      /* instantiate sub module */
60      reg [OP_WIDTH - 1:0] sub_a, sub_b;
61      wire [OP_WIDTH - 1:0] sub_z;
62      sub #(.WIDTH(OP_WIDTH)) sub_inst (
63          .x(sub_a),
64          .y(sub_b),
65          .z(sub_z)
66      );
67
68      always @(posedge clk, negedge rst_n) begin
69          if (!rst_n) begin
70              state <= S0;
71              out <= {WIDTH + 4{1'b0}};
72              ready <= 1'b0;
73          end
74          else begin
75              case (state)
76                  S0: begin
77                      if(!ready) begin
78                          /* load A and B */
79                          reg1 <= a;
80                          reg2 <= b;
81                          reg3 <= a;
82                          reg4 <= b;
83                          state <= S1;
84                      end
85                  end
86                  S1: begin
87                      /* reg1 = (A / 2) */
88                      reg1 <= div2_out;
89                      state <= S2;
90                  end
91                  S2: begin
92                      /* reg1 = (A / 2 + B) */
93                      reg1 <= add_z;
94                      state <= S3;
95                  end
96                  S3: begin
```

```
97                      /* reg4 = (B / 2) */
98                      reg4 <= div2_out;
99                      state <= S4;
100                 end
101                 S4: begin
102                     /* reg3 = (A - B / 2)*/
103                     reg3 <= sub_z;
104                     state <= S5;
105                 end
106                 S5: begin
107                     /* reg1 = ((A / 2) + B) * 8 */
108                     if(mul_cnt !== 2'd2) begin
109                         reg1 <= mul2_out;
110                         mul_cnt <= mul_cnt + 1;
111                         state <= S5;
112                     end
113                     else begin
114                         mul_cnt <= 0;
115                         reg1 <= mul2_out;
116                         state <= S6;
117                     end
118                 end
119                 S6: begin
120                     /* reg3 = (A - B / 2) * 4 */
121                     if (mul_cnt !== 2'd1) begin
122                         reg3 <= mul2_out;
123                         mul_cnt <= mul_cnt + 1;
124                         state <= S6;
125                     end
126                     else begin
127                         mul_cnt <= 0;
128                         reg3 <= mul2_out;
129                         state <= S7;
130                     end
131                 end
132                 S7: begin
133                     /* out = ((A / 2) + B) * 8 + (A - B / 2) * 4 */
134                     out <= add_z;
135                     state <= S0;
136                     ready <= 1'b1;
137                 end
138                 default:
139                     state <= state;
140             endcase
141         end
142     end
143
144     /* Connect inputs of adder, mul2, sub and div2 */
145     always@(*) begin
146         case (state)
147             S1: begin
148                 div2_in = reg1;
149             end
```

```
            S2: begin
                add_a = reg1;
                add_b = reg2;
            end
            S3: begin
                div2_in = reg4;
            end
            S4: begin
                sub_a = reg3;
                sub_b = reg4;
            end
            S5: begin
                mul2_in = reg1;
            end
            S6: begin
                mul2_in = reg3;
            end
            S7: begin
                add_a = reg1;
                add_b = reg3;
            end
            default: begin
                div2_in = 0;
                add_a = 0;
                add_b = 0;
                mul2_in = 0;
                sub_a = 0;
                sub_b = 0;
            end
        endcase
    end
endmodule
```

2.3.2 Тестовый план:

1. Протестировать правильность вычисления функции на любых валидных входных данных.
2. Протестировать сброс автомата.
3. Протестировать максимальные значения входных аргументов. $a = 2^{32} - 1\ and\ b = 2^{32} - 1$
4. Протестировать значения 0 входных аргументов. $a = 0 and b = 0$
5. Протестировать минимальные значения входных аргументов. $a = -2^{31}\ and\ b = -2^{31}$

```
`timescale 1ps/1ps
`include "fsm/src/fsm.v"
module fsm_tb;
    localparam WIDTH = 32;
    localparam OP_WIDTH = WIDTH + 3;
```

```
6       reg clk;
7       reg [WIDTH - 1:0] a, b;
8       reg rst_n;
9       wire ready;
10      wire [OP_WIDTH-1:0] out;
11      reg passed = 1;
12      reg [OP_WIDTH-1:0] tst_out;
13      fsm #(.WIDTH(WIDTH)) fsm_dut (
14          .clk(clk),
15          .rst_n(rst_n),
16          .a(a),
17          .b(b),
18          .out(out),
19          .ready(ready)
20      );
21
22      /* Clock generation */
23      initial begin
24          clk = 0;
25          forever #10 clk = ~clk;
26      end
27
28      task check;
29          begin
30              if(tst_out !== out) begin
31                  $display("[T=%0g] Test failed: expected %g, got %g",
                        $time, tst_out, out);
32                  passed = 0;
33              end
34          end
35      endtask
36
37      task calfunc(input [WIDTH - 1:0] a, b, output [OP_WIDTH-1:0]
           out);
38          begin
39              out = ((a / 2) + b) * 8 + (a - (b / 2)) * 4;
40          end
41      endtask
42
43      initial begin
44          $dumpfile("build/fsm.vcd");
45          $dumpvars(1);
46
47          $monitor("[T=%0d] state->%0d, reg1=%0d, reg2=%0d, reg3=%00d,
               reg4=%0d, mul_cnt=%0d, ready=%b, out=%0d",
48              $time, fsm_dut.state, fsm_dut.reg1, fsm_dut.reg2,
                   fsm_dut.reg3, fsm_dut.reg4, fsm_dut.mul_cnt,
                   fsm_dut.ready, fsm_dut.out);
49
50          rst_n = 0;
51
52          /* Test: a = 6 and b = 4 */
53          $display("[T=%0g] Test 1: a = 6, b = 4", $time);
```

```
54          @(negedge clk) begin
55              a <= 32'd6;
56              b <= 32'd4;
57              rst_n <= 1;
58          end
59          while (!ready) begin
60              @(posedge clk);
61          end
62          calfunc(a, b, tst_out);
63          check;
64
65          /* Test: Reset */
66          $display("[T=%0g] Test 2: Reset", $time);
67          @(negedge clk) begin
68              rst_n <= 0;
69              a <= 32'd1234;
70              b <= 32'd5678;
71          end
72          tst_out = 0;
73          @(posedge clk) check;
74
75          @(negedge clk) begin
76              rst_n <= 1;
77          end
78
79          while(!ready) begin
80              @(posedge clk);
81          end
82          calfunc(a, b, tst_out);
83          @(posedge clk) check;
84
85          /* Test: Max values a = 2^32 - 1 and b = 2^32 - 1 */
86          $display("[T=%0g] Test 3: a = 2^32 - 1, b = 2^32 - 1",
               $time);
87          @(negedge clk) begin
88              // a = 2^31 - 1
89              a <= 32'h7FFFFFFF;
90              // b = 2^31 - 1
91              b <= 32'h7FFFFFFF;
92              rst_n <= 1;
93          end
94          while (!ready) begin
95              @(posedge clk);
96          end
97          calfunc(a, b, tst_out);
98          @(posedge clk) check;
99
100         /* Test: Zero values a = 0 and b = 0 */
101         $display("[T=%0g] Test 4: a = 0, b = 0", $time);
102         @(negedge clk) begin
103             a <= 32'h0;
104             b <= 32'h0;
105             rst_n <= 0;
```

```
106         end
107         @(negedge clk) begin
108             rst_n <= 1;
109         end
110         while (!ready) begin
111             @(posedge clk);
112         end
113         calfunc(a, b, tst_out);
114         @(posedge clk) check;
115 
116         /* Test: Min values a = -2^31 and b = -2^31 */
117         $display("[T=%0g] Test 5: a = -2^31, b = -2^31", $time);
118         @(negedge clk) begin
119             // a = -2^31
120             a <= 32'h80000000;
121             // b = -2^31
122             b <= 32'h80000000;
123             rst_n <= 0;
124         end
125         @(negedge clk) begin
126             rst_n <= 1;
127         end
128         while (!ready) begin
129             @(posedge clk);
130         end
131         calfunc(a, b, tst_out);
132         @(posedge clk) check;
133 
134         if(passed)
135             $display("[T=%0g] All tests passed", $time);
136         else
137             $display("[T=%0g] Some tests failed", $time);
138         #30; $finish;
139     end
140 
141 endmodule
```

Результаты тестирования:

```
[T=0] Test 1: a = 6, b = 4
[T=250] Test 2: Reset
[T=530] Test 3: a = 2^32 - 1, b = 2^32 - 1
[T=550] Test 4: a = 0, b = 0
[T=830] Test 5: a = -2^31, b = -2^31
[T=1110] All tests passed
```

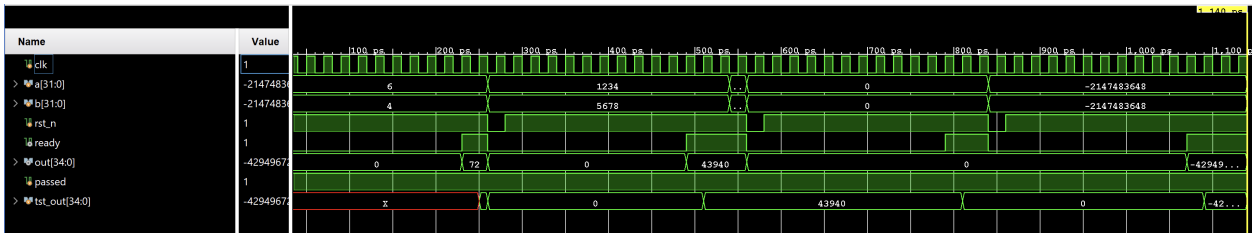

Рис. 6: Временная диаграмма работы конечного автомата

2.4 Делитель частоты

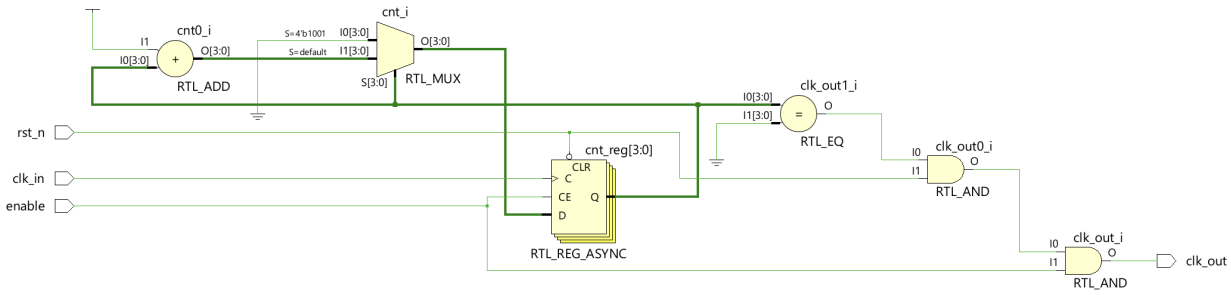

Рис. 7: Схема делителя частоты, уменьшает частоту на 10 раз

При подаче асинхронного сигнала сброса rst d-триггер сбрасывается. С каждым тактом инкрементируется значение регистра cnt_reg до тех пор, пока cnt не станет равным значению DIV_CNT-1. Тогда значение регистра clk_out становиться равным 1 и обнуляется значение cnt.

2.4.1 Разработанный модуль

```
1  /* Frequency clock divider */
2  module freq_div
3  #(
4      parameter DIV_CNT = 10,
5      parameter WIDTH = $clog2(DIV_CNT)
6  )
7  (
8      input clk_in,
9      input rst_n,
10     input enable,
11     output clk_out
12 );
13     reg [WIDTH-1:0] cnt;
14
15     always @(posedge clk_in or negedge rst_n) begin
16         if(!rst_n)
17             cnt <= {WIDTH{1'b0}};
18         else if (enable)
19             if (cnt == DIV_CNT - 1)
```

```
                cnt <= {WIDTH{1'b0}};
            else
                cnt <= cnt + 1'b1;
        else
            cnt <= cnt;
    end

    assign clk_out = (cnt == 0 && rst_n && enable) ? 1'b1 : 1'b0;
endmodule
```

2.4.2 Тестовый план:

1. Протестировать правильное деление частоты на 10.
2. Протестировать сброс делителя.
3. Протестировать сигнал разрешения.
4. Протестировать продолжение работы после сброса.

```
`timescale 1ns/1ps
`include "freq_div/src/freq_div.v"

module freq_div_tb;
    localparam DIV_CNT = 10;
    reg  clk_in;
    reg  rst_n;
    reg enable;
    wire clk_out;
    freq_div #(.DIV_CNT(DIV_CNT)) fd_dut(
        .clk_in (clk_in),
        .rst_n (rst_n),
        .enable (enable),
        .clk_out (clk_out)
    );

    reg passed = 1;
    reg tst_out;

    /* Clock generation */

    initial begin
        clk_in = 0;
        forever #10 clk_in = !clk_in;
    end

    task check;
        begin
            if(tst_out != clk_out) begin
                $display("[T=%0g] Test failed: expected %g, got %g",
                   $time, tst_out, clk_out);
                passed = 1'b0;
            end
```

```
33          end
34      endtask
35
36      integer i;
37      initial begin
38          $dumpfile("build/freq_div.vcd");
39          $dumpvars(1);
40          $monitor("[T=%0d] enable=%b, rst_n=%b, cnt=%0d, clk_out=%b",
               $time, enable, rst_n, fd_dut.cnt, clk_out);
41
42
43          rst_n = 0;
44          enable = 1;
45          #1 rst_n = 1;
46
47          tst_out = 1'b0;
48
49          /* Test 1: Count test */
50          $display("[T=%0g] Test 1: Count test", $time);
51          for(i = 1; i < 98; i = i + 1) begin
52              @(posedge clk_in) begin
53                  if(i % DIV_CNT == 0)
54                      tst_out = 1'b1;
55                  else tst_out = 1'b0;
56              end
57              @(negedge clk_in) begin
58                  check;
59              end
60          end
61
62          /* Test 2: Reset, Should Be 0 */
63          $display("[T=%0g] Test 2: Reset, Should Be 0", $time);
64          @(posedge clk_in) begin
65              rst_n <= 0;
66              tst_out <= 0;
67          end
68          @(negedge clk_in) check;
69
70          #1 rst_n = 1;
71
72          /* Test 3: Continue count */
73          $display("[T=%0g] Test 3: Continue count", $time);
74          for(i = 1; i < 98; i = i + 1) begin
75              @(posedge clk_in) begin
76                  if(i % DIV_CNT == 0)
77                      tst_out = 1'b1;
78                  else tst_out = 1'b0;
79              end
80              @(negedge clk_in) begin
81                  check;
82              end
83          end
84
```

```
85          /* Test 4: Enable disable */
86          $display("[T=%0g] Test 4: Enable disable", $time);
87          rst_n = 0;
88          enable <= 0;
89          @(posedge clk_in) begin
90              tst_out <= 0;
91              rst_n <= 1;
92          end
93          repeat(93) @(negedge clk_in) check;
94
95          if(passed)
96              $display("[T=%0g] All tests passed", $time);
97          else
98              $display("[T=%0g] Some tests failed", $time);
99          #30; $finish;
100
101     end
102 endmodule
```

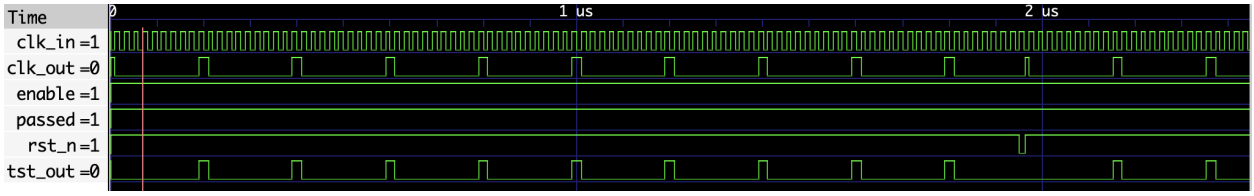

Рис. 8: Временная диаграмма работы делителя частоты

Результаты тестирования:

```
[T=1] Test 1: Count test
[T=1940] Test 2: Reset, Should Be 0
[T=1961] Test 3: Continue count
[T=3900] Test 4: Enable disable
[T=5760] All tests passed
```

3 Функция COUNT FREE

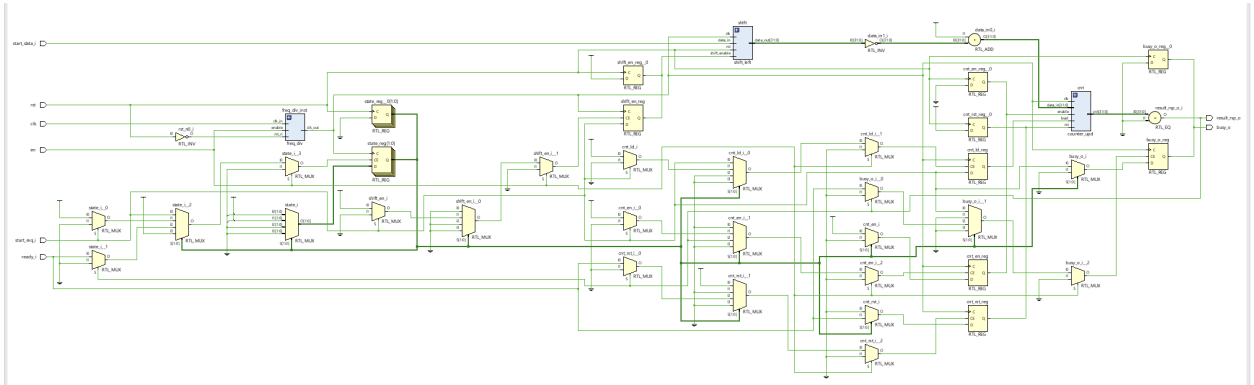

Рис. 9: Схема функции COUNT_FREE

Результаты тестирования:

```
[T=0] Test 1: Reset
[T=222] Test 2: Count
[T=4440] All tests passed
```

4 FIFO

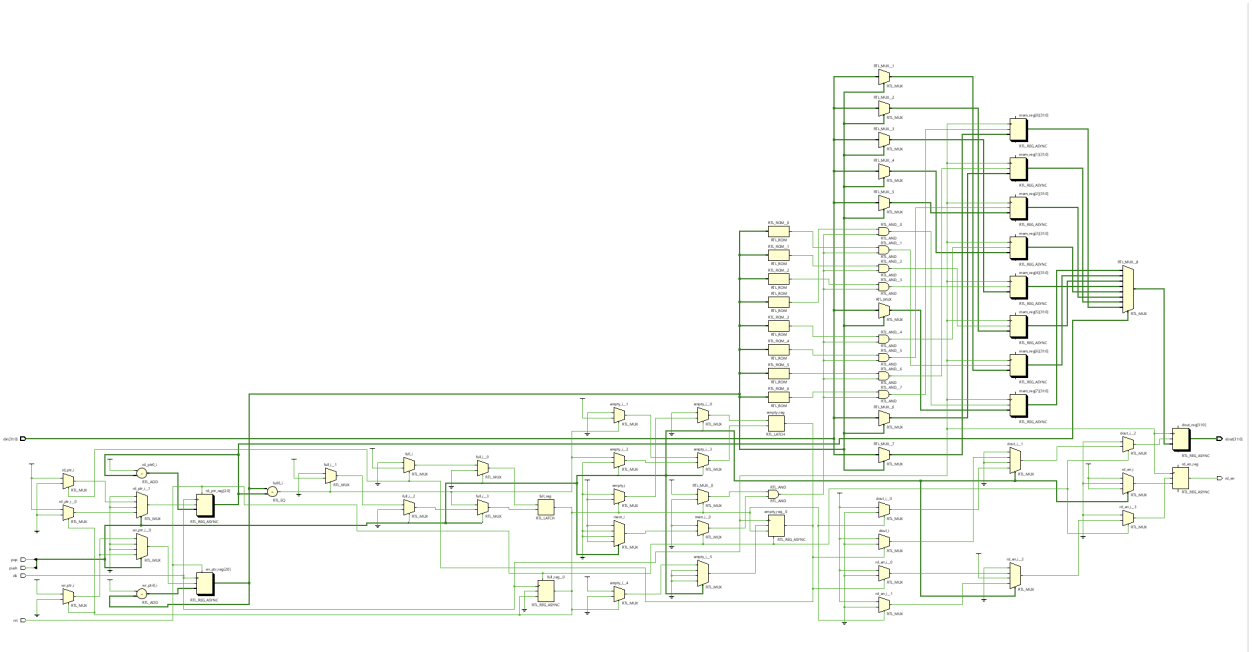

Рис. 10: Схема FIFO

Результаты тестирования:

```
Test 1: Basic write/read
Test 2: Overflow condition
Test 3: Underflow condition
Test 4: PUSH and POP on the same clock cycle
[T=675] All tests passed
```