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

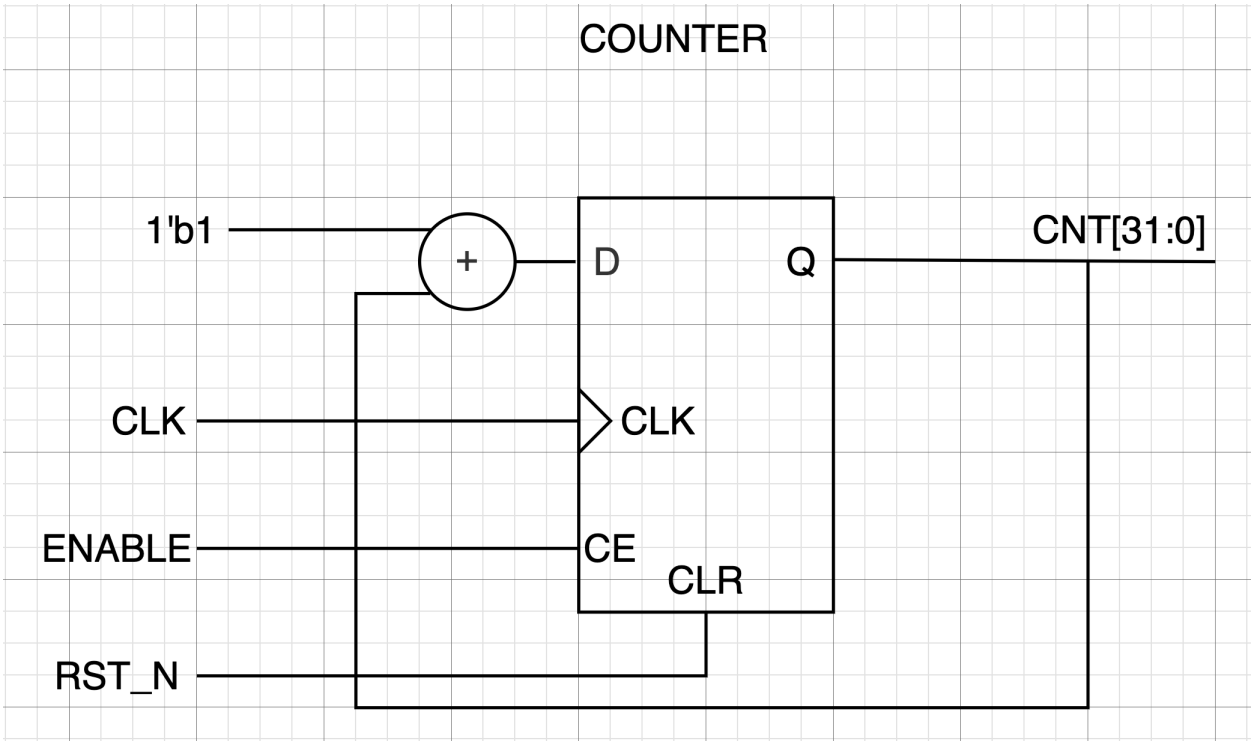



Рис. 1: Синхронный счетчик по переднему фронту с асинхронным сбросом и сигналом разрешения, 32 разряда.

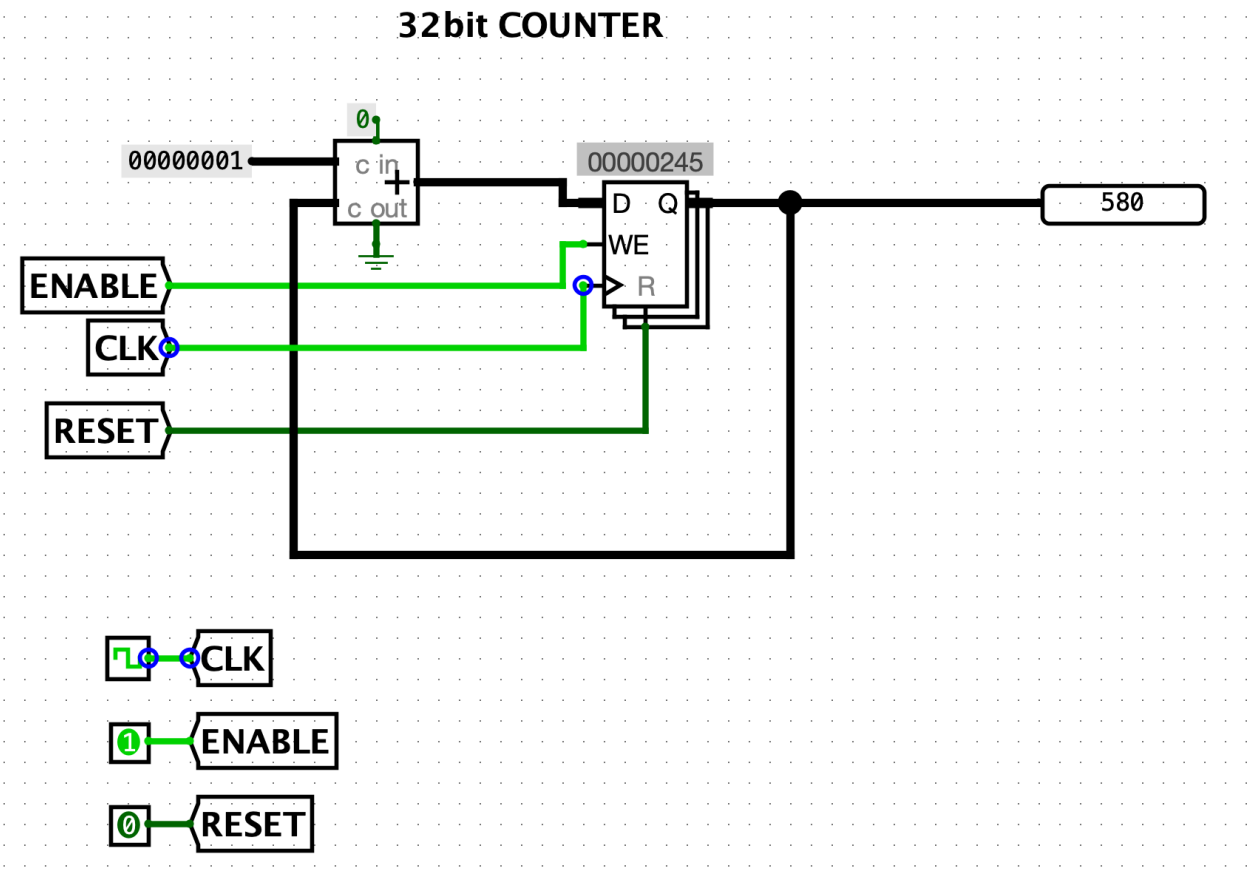



Рис. 2: Синхронный счетчик по переднему фронту с асинхронным сбросом и сигналом разрешения, 32 разряда.

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;
```

```
16          else
17              cnt <= cnt;
18      end
19  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

```
1   `timescale 1ns/10ps
2   `include "counter/src/counter.v"
3
4   module counter_tb;
5       localparam WIDTH = 32;
6
7       reg clk;
8       reg rst_n;
9       reg enable;
10      wire [WIDTH-1:0] cnt;
11
12      reg [WIDTH-1:0] tst_out;
13
14      reg passed = 1'b1;
15
16      counter #(.WIDTH(WIDTH)) counter_dut(
17          .clk (clk),
18          .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
```

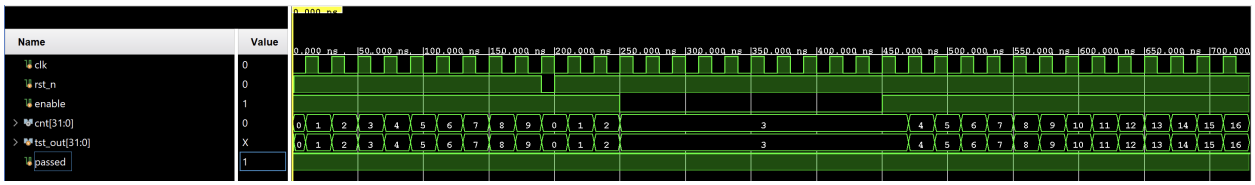



Рис. 3: Временная диаграмма работы счетчика

## 2.2 Сдвиговый регистр

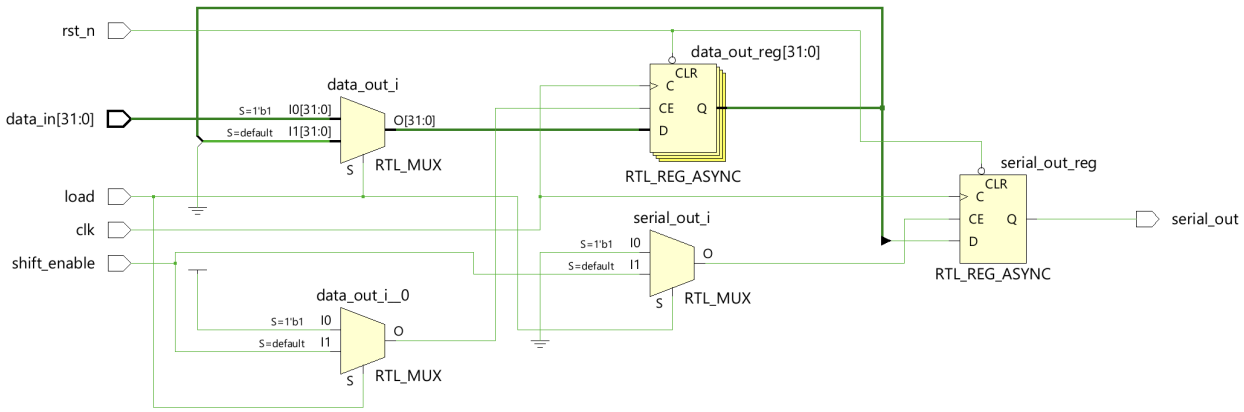



Рис. 4: Схема сдвигового регистра с параллельной загрузкой и последовательным сдвигом вправо

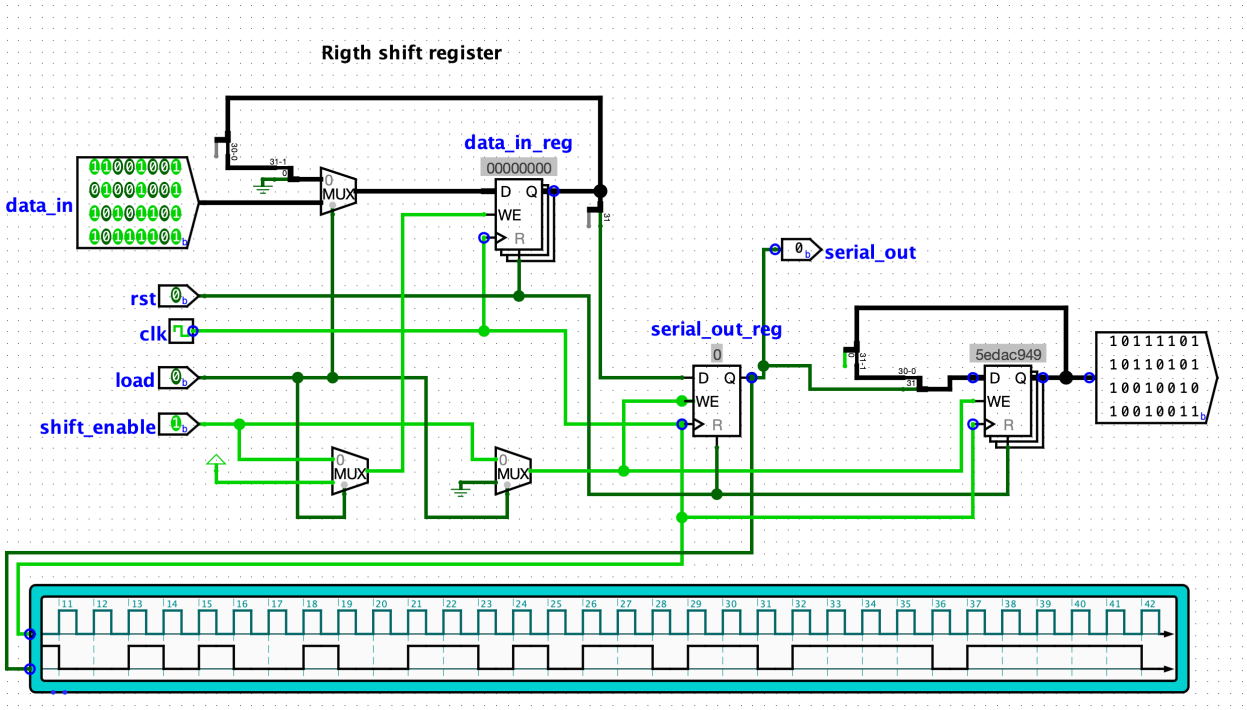



Рис. 5: Схема сдвигового регистра с параллельной загрузкой и последовательным сдвигом вправо

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

```
/* 32 - bit Right Shift register with shift_enable */
module shift_right
#(
    parameter WIDTH = 32
)
(
    input clk,
    input rst_n,
    input [WIDTH - 1:0] data_in,
    input shift_enable,
    input load,
    output reg serial_out
);

    reg [WIDTH - 1:0] data_out;
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n)
            {serial_out, data_out} <= {1'b0, {WIDTH-1{1'b0}}};
        else if(load)
            data_out <= data_in;
        else if (shift_enable)
```

```
            {data_out[WIDTH-1:0], serial_out} <= {1'b0,
               data_out[WIDTH-1:0]};
        else
            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.

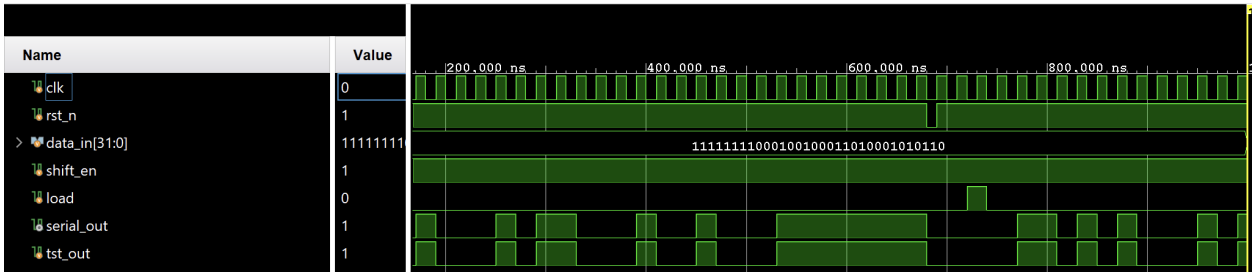



Рис. 6: Временная диаграмма работы сдвигового регистра

```
`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),
```

```
25          .shift_enable(shift_en),
26          .serial_out(serial_out)
27      );
28
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 */
```

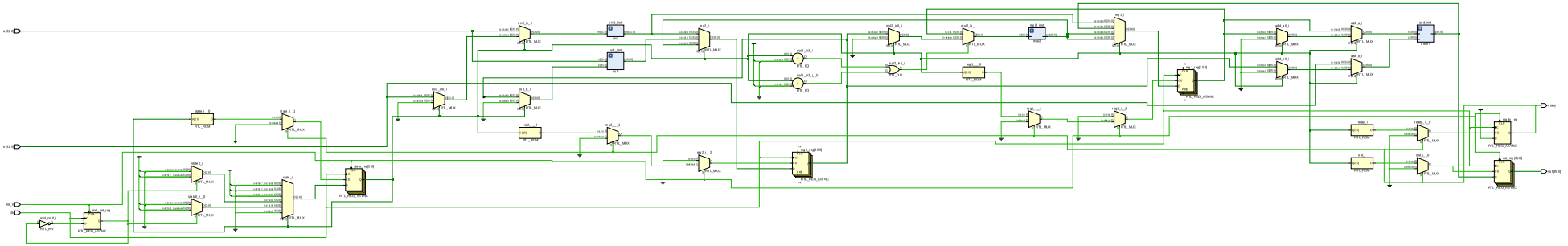
```
75          $display("########## TEST: Reset ##########");
76          rst_n = 0;
77          tst_out = 0;
78          @(posedge clk) check;
79          rst_n = 1;
80
81          /* Test: Buffer is empty */
82          $display("########## TEST: Buffer is empty ##########");
83          @(posedge 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

Схема конечного автомата



Конечный автомат состоит из 6 состояний, которые соответвуют последовательным этапам вычисления функции $(A/2 + B) * 8 + (A - B/2) * 4$.

Входные данные:

- clk: тактовый сигнал
- a, b: входные данные шириной WIDTH бит
- rst_n: асинхронный сигнал сброса

Работа модуля:

- Модуль fsm использует конечный автомат с шестью состояниями (S0, S1, S2, S3, S4, S5) для выполнения последовательности операций над входными данными.
- В каждом состоянии выполняются определенные операции, такие как деление на 2 (div2), сложение (adder), умножение на 2 (mul2) и вычитание (sub).
- Результаты промежуточных операций сохраняются в регистрах reg1 и reg2.
- Переходы между состояниями происходят на каждом такте тактового сигнала clk, если сигнал ready неактивен.
- В состоянии S0:
  - Значение a делится на 2 с помощью модуля div2, результат сохраняется в reg1.
- В состоянии S1:
  - Значение b делится на 2 с помощью модуля div2, результат сохраняется в reg2.
  - Значение reg1 складывается с b с помощью модуля adder, результат сохраняется в reg1.
- В состоянии S2:
  - Значение a вычитается из reg2 с помощью модуля sub, результат сохраняется в reg2.
  - Значение reg1 умножается на 2 с помощью модуля mul2, результат сохраняется в reg1.
- В состоянии S3:
  - Значение reg1 умножается на 2 с помощью модуля mul2, результат сохраняется в reg1.
  - Если счетчик mul_cnt равен 0, переход в состояние S4, иначе остаемся в S3
- В состоянии S4:
  - Значение reg2 умножается на 4 с помощью модуля mul2, результат сохраняется в reg2.
  - Если счетчик mul_cnt равен 0, переход в состояние S5, иначе остаемся в S4.
- В состоянии S5:

– Значения reg1 и reg2 складываются с помощью модуля adder, результат сохраняется в out.

– Сигнал ready устанавливается в 1, указывая на готовность результата.

Выходные данные:

- out: выходные данные, содержащие результат вычислений
- ready: сигнал готовности результата

### 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 + 4,
    parameter DIV_WIDTH = WIDTH - 1,
    parameter MUL1_WIDTH = WIDTH + 3,
    parameter MUL2_WIDTH = WIDTH,
    parameter ADDER1_WIDTH = WIDTH,
    parameter ADDER2_WIDTH = WIDTH + 3,
    parameter SUB_WIDTH = WIDTH - 2
)
(
    input clk,
    input [WIDTH - 1:0] a, b,
    input rst_n,
    output reg[OP_WIDTH-1:0] out,
    output reg ready
);

    localparam [2:0]
        S0 = 3'b000,
        S1 = 3'b001,
        S2 = 3'b010,
        S3 = 3'b011,
        S4 = 3'b100,
        S5 = 3'b101;

    reg [2:0] state;
    reg mul_cnt = 0;

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

```
40      .out(div2_out)
41      );
42
43      /* instantiate adder module */
44      wire [OP_WIDTH - 1:0] add_a, add_b;
45      wire [OP_WIDTH - 1:0] add_z;
46      adder #(.WIDTH(OP_WIDTH)) add_inst (
47          .x(add_a),
48          .y(add_b),
49          .z(add_z)
50      );
51
52      /* instantiate mul2 module */
53      wire [OP_WIDTH - 1:0] mul2_in;
54      wire [OP_WIDTH - 1:0] mul2_out;
55      mul2 #(.WIDTH(OP_WIDTH)) mul2_inst (
56          .in(mul2_in),
57          .out(mul2_out)
58      );
59
60      /* instantiate sub module */
61      wire[WIDTH - 1:0] sub_b;
62      wire [WIDTH - 1:0] sub_z;
63      sub #(.WIDTH(WIDTH)) sub_inst (
64          .x(a),
65          .y(sub_b),
66          .z(sub_z)
67      );
68
69      // Additional registers to manage values consistently
70      reg [OP_WIDTH - 1:0] reg1, reg2;
71
72      always @(posedge clk, negedge rst_n) begin
73          if (!rst_n) begin
74              state <= S0;
75              out <= {WIDTH + 4{1'b0}};
76              mul_cnt <= 0;
77              ready <= 1'b0;
78          end
79          else begin
80              mul_cnt <= !mul_cnt;
81              if(!ready) begin
82                  case (state)
83                      S0: begin
84                          /* reg1 = (A / 2) */
85                          reg1 <= div2_out;
86                          state <= S1;
87                      end
88                      S1: begin
89                          /* reg2 = (B / 2) */
90                          reg2 <= div2_out;
91                          /* reg1 = (A / 2 + B) */
92                          reg1 <= add_z;
```

```
93                          state <= S2;
94                      end
95                      S2: begin
96                          /* reg2 = (A - B / 2)*/
97                          reg2 <= sub_z;
98                          /* reg1 = ((A / 2) + B) * 2 */
99                          reg1 <= mul2_out;
100                         state <= S3;
101                     end
102                     S3: begin
103                         /* reg1 = ((A / 2) + B) * 2 * 4 */
104                         reg1 <= mul2_out;
105                         state <= !mul_cnt ? S4 : S3;
106                     end
107                     S4: begin
108                         /* reg2 = (A - B / 2) * 4 */
109                         reg2 <= mul2_out;
110                         state <= !mul_cnt ? S5 : S4;
111
112                     end
113                     S5: begin
114                         /* out = ((A / 2) + B) * 8 + (A - B / 2) * 4
                               */
115                         out <= add_z;
116                         ready <= 1'b1;
117                     end
118                     default: begin
119                         state <= S0;
120                     end
121                 endcase
122             end
123         end
124     end
125
126     assign div2_in = state == S0 ? a :
127             state == S1 ? b : 0;
128     assign sub_b = state == S2 ? reg2 : 0;
129     assign mul2_in = state == S2 || state == S3 ? reg1 :
130                      state == S4 ? reg2 : 0;
131     assign add_a = state == S1 ? reg1:
132                      state == S5 ? reg1 : 0;
133     assign add_b = state == S1 ? b : state == S5 ? reg2 : 0;
134 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 + 4;
    reg clk;
    reg [WIDTH - 1:0] a, b;
    reg rst_n;
    wire ready;
    wire [OP_WIDTH-1:0] out;
    reg passed = 1;
    reg [OP_WIDTH-1:0] tst_out;
    fsm #(.WIDTH(WIDTH)) fsm_dut (
        .clk(clk),
        .rst_n(rst_n),
        .a(a),
        .b(b),
        .out(out),
        .ready(ready)
    );

    /* Clock generation */
    initial begin
        clk = 0;
        forever #10 clk = ~clk;
    end

    task check;
        begin
            if(tst_out !== out) begin
                $display("[T=%0g] Test failed: expected %g, got %g",
                   $time, tst_out, out);
                passed = 0;
            end
        end
    endtask

    task calfunc(input [WIDTH - 1:0] a, b, output [OP_WIDTH-1:0]
       out);
        begin
            out = ((a / 2) + b) * 8 + (a - (b / 2)) * 4;
        end
    endtask

    initial begin
        $dumpfile("build/fsm.vcd");
        $dumpvars(1);

```

```
47          $monitor("[T=%0d] state->%0d, reg1=%0d, reg2=%0d, add_a=%0d,
               add_b=%0d, div2_in=%0d, mul2_in=%0d, mul_cnt=%0d,
               ready=%b, out=%0d",
48              $time, fsm_dut.state,fsm_dut.reg1,
                   fsm_dut.reg2,fsm_dut.add_a, fsm_dut.add_b,
                   fsm_dut.div2_in, fsm_dut.mul2_in, 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'd5678;
70              b <= 32'd1234;
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              rst_n <= 0;
89          end
90          @(negedge clk) begin
91              // a = 2^32 - 1
92              a <= 32'hFFFFFFFF;
93              // b = 2^32 - 1
```

```
94              b <= 32'hFFFFFFFF;
95              rst_n <= 1;
96          end
97          while (!ready) begin
98              @(posedge clk);
99          end
100         calfunc(a, b, tst_out);
101         @(posedge clk) check;
102
103         /* Test: Zero values a = 0 and b = 0 */
104         $display("[T=%0g] Test 4: a = 0, b = 0", $time);
105         @(negedge clk) begin
106             a <= 32'h0;
107             b <= 32'h0;
108             rst_n <= 0;
109         end
110         @(negedge clk) begin
111             rst_n <= 1;
112         end
113         while (!ready) begin
114             @(posedge clk);
115         end
116         calfunc(a, b, tst_out);
117         @(posedge clk) check;
118
119         if(passed)
120             $display("[T=%0g] All tests passed", $time);
121         else
122             $display("[T=%0g] Some tests failed", $time);
123         #30; $finish;
124     end
125
126 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
```

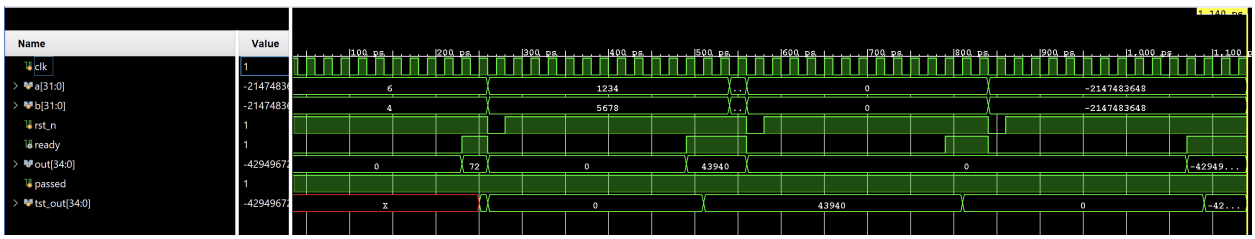



Рис. 7: Временная диаграмма работы конечного автомата

## 2.4 Делитель частоты

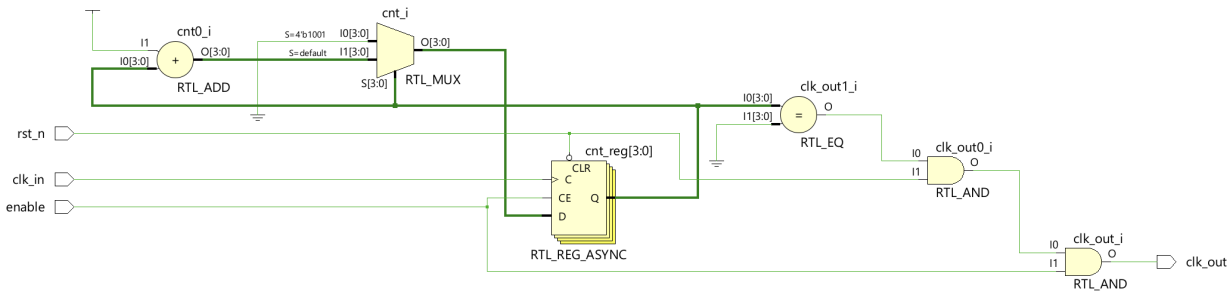



Рис. 8: Схема делителя частоты, уменьшает частоту на 10 раз

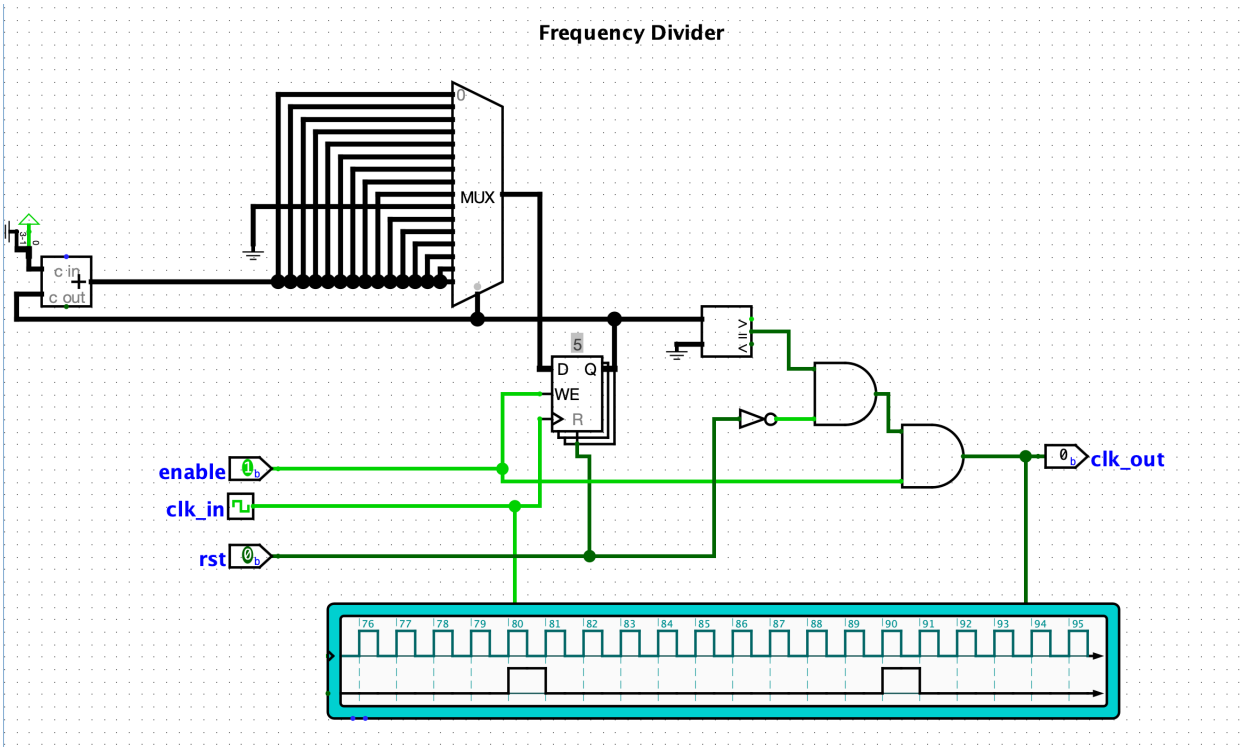



Рис. 9: Схема делителя частоты, уменьшает частоту на 10 раз

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,
```

```
    parameter WIDTH = $clog2(DIV_CNT)
)
(
    input clk_in,
    input rst_n,
    input enable,
    output clk_out
);
    reg [WIDTH-1:0] cnt;

    always @(posedge clk_in or negedge rst_n) begin
        if(!rst_n)
            cnt <= {WIDTH{1'b0}};
        else if (enable)
            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;
```

```
19
20      /* Clock generation */
21
22      initial begin
23          clk_in = 0;
24          forever #10 clk_in = !clk_in;
25      end
26
27      task check;
28          begin
29              if(tst_out != clk_out) begin
30                  $display("[T=%0g] Test failed: expected %g, got %g",
                        $time, tst_out, clk_out);
31                  passed = 1'b0;
32              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
```

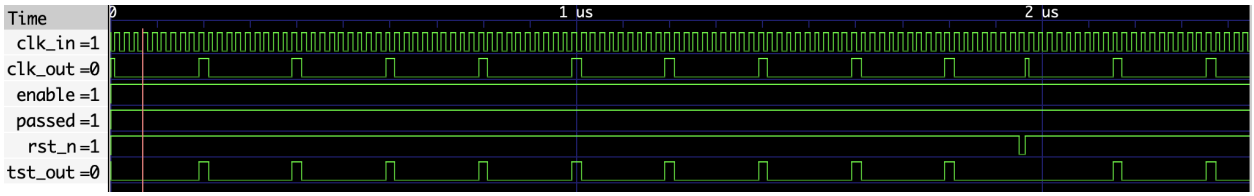



Рис. 10: Временная диаграмма работы делителя частоты

**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

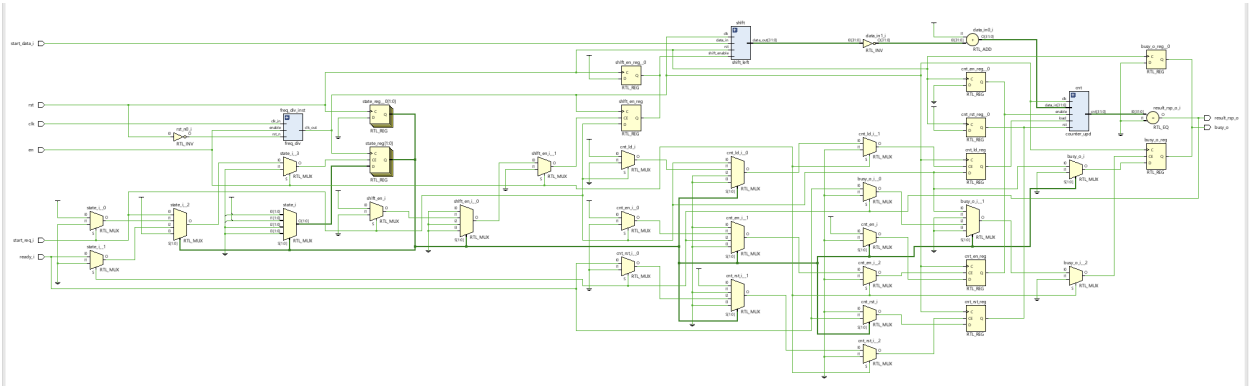


Рис. 11: Схема функции COUNT_FREE

Задание:

Необходимо после прошествия определенного количества времени сформировать однобитный сигнал запроса. Информацию о количестве тактов, после которого необходимо формировать сигнал, устройство получает по однобитному последовательностному порту запроса. Необходимые сигналы в интерфейсе вашего модуля:

| **Порт** | **Тип** | **Описание** |
|---|---|---|
| clk | Вх. | Сигнал тактовой частоты. |
| rst | Вх. | Асинхронный сигнал сброса. |
| start_req_i | Вх. | Сигнал запроса. Представляет собой сигнал валидности для однобитного сигнала входных данных. |
| start_data_i | Вх. | Однобитный сигнал данных, представляющий собой один разряд исходного числа. |
| ready_i | Вх. | Сигнал готовности внешнего устройства принять результат. |
| 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. |

Входные данные:

- clk: тактовый сигнал
- en: сигнал разрешения работы модуля
- rst: сигнал сброса

- start_req_i: запрос на начало подсчета
- start_data_i: входные данные для подсчета
- ready_i: сигнал готовности приема результата

Работа модуля:

- Модуль freq_div делит частоту тактового сигнала clk и генерирует сигнал fd_clk с пониженной в 10 раз частотой.
- Модуль shift_left последовательно сдвигает входные данные start_data_i в регистр data_out на каждом такте fd_clk, пока shlft_en активен.
- Модуль counter_upd считает количество тактовых сигналов.
- Модуль count_free управляет работой shift_left и counter_upd с помощью конечного автомата с тремя состояниями: WAIT, READ и COUNT.
  - В состоянии WAIT модуль ожидает запроса start_req_i. Когда запрос получен, модуль переходит в состояние READ.
  - В состоянии READ модуль разрешает сдвиг данных в shift_left и загружает значение в счетчик counter_upd (сколько тактов ему надо отсчитать). Когда start_req_i становится неактивным, модуль переходит в состояние COUNT.
  - В состоянии COUNT модуль разрешает счетчику counter_upd считать количество тактов. Когда счетчик достигает нужного значения, модуль ждет сигнала ready_i, чтобы отправить результат и перейти обратно в состояние WAIT.

Выходные данные:

- result_rsp_o: результат подсчета, равный 1, когда счетчик достигает нуля, и 0 в остальных случаях.
- busy_o: сигнал занятости, указывающий, что модуль выполняет подсчет.

**Результаты тестирования:**

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

# 4 FIFO

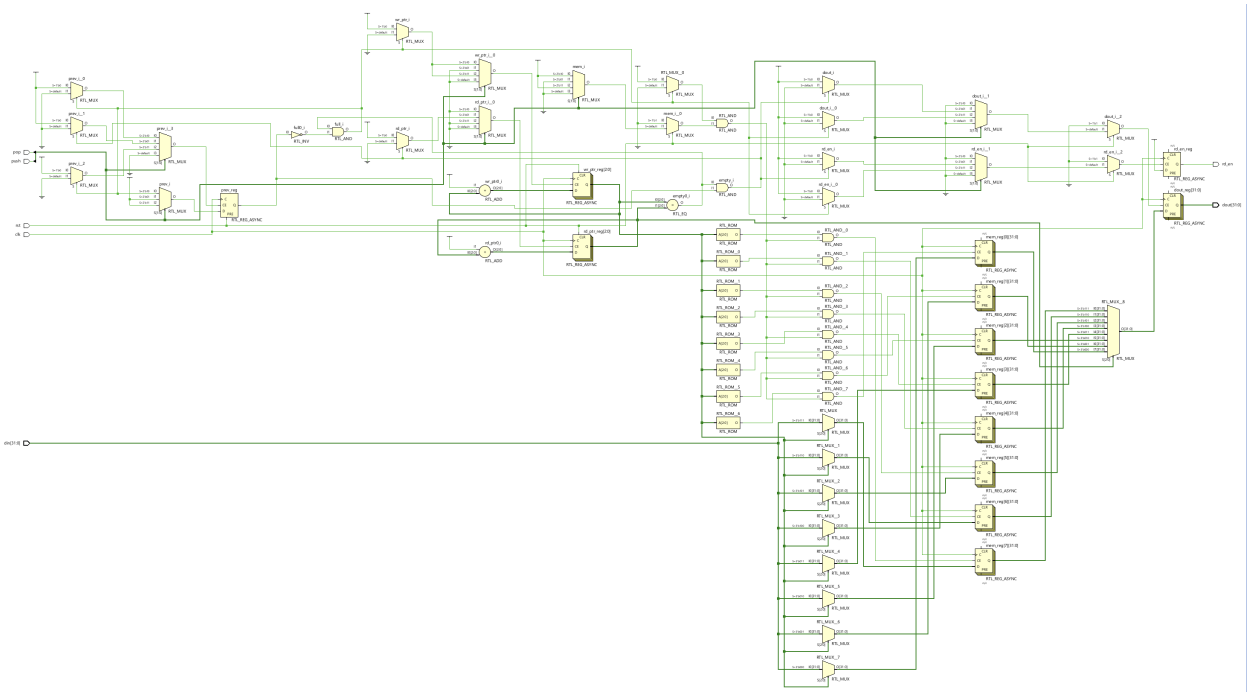


Рис. 12: Схема FIFO

Входные данные:

- clk: тактовый сигнал
- rst: сигнал сброса
- din: входные данные
- push: сигнал записи данных в fifo
- pop: сигнал чтения данных из fifo

Работа модуля:

- Модуль fifo использует два указателя: rd_ptr (указатель чтения) и wr_ptr (указатель записи) для отслеживания головы и хвоста буфера.
- Данные хранятся в памяти mem, которая представляет собой массив регистров размером $2^{PTR\_WIDTH}$.
- Когда сигнал push активен и буфер не заполнен, входные данные din записываются в память mem по адресу wr_ptr, и указатель записи wr_ptr увеличивается на 1.
- Когда сигнал pop активен и буфер не пуст, данные из памяти mem по адресу rd_ptr выводятся на выход dout, сигнал rd_en устанавливается в 1, указывая на то, что данные доступны для чтения, и указатель чтения rd_ptr увеличивается на 1.
- Если оба сигнала push и pop активны одновременно, и буфер не заполнен, происходит сначала запись, потом чтение данных.

- Сигналы empty и full указывают на состояние буфера: пустой (empty) или заполненный (full). Они определяются на основе значений указателей чтения и записи, а также предыдущего действи (push или pop).

Выходные данные:

- dout: выходные данные
- rd_en: сигнал, указывающий на то, что данные доступны для чтения

**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
```

# 5 Вывод

В ходе выполнения лабораторной работы были разработаны и протестированы следующие устройства:

- Счетчик с асинхронным сбросом и сигналом разрешения.
- Сдвиговый регистр с параллельной загрузкой и последовательным сдвигом вправо.
- Конечный автомат, вычисляющий функцию $(A/2 + B) * 8 + (A - B/2) * 4$.
- Делитель частоты, уменьшающий частоту на 10 раз.
- Устройство COUNT_FREE, считающее количество единиц во входном потоке.
- FIFO, реализующая очередь с фиксированным размером.

Вопрос: На что влияет не учитывание всех учитывать всех возможных комбинации входных данных?

Ответ:

- В комбинационной возникают засчелки

```
`timescale 1ns / 1ps

module test(
    input [1:0] sel,
    input [7:0] a, b,
    output reg [7:0] out
);
    always @(*) begin
        case (sel)
            2'b00: out = a;
            2'b01: out = b;
        endcase
    end
endmodule
```

Синтезатор не может определить, какое значение должен принимать выход out для нерассмотренных комбинаций входов. Поэтому он может реализовать выход как latch (триггер с асинхронным установлением), который будет сохранять предыдущее значение

| Rule ID | RTL Name | RTL Hierarchy | Message Body | File Name |
|---|---|---|---|---|
| INFER | | | | |
| INFER-1 | | | | |
| INFER-1# 1 | out_reg | test | Inferred latch for signal 'out_reg' | test.v : 10 |
| INFER-2 | | | | |
| INFER-2# 1 | | test | Case statement conditions not fully specified | test.v : 9 |

Рис. 13: Vivado сигнализирует о возникновении засчелки

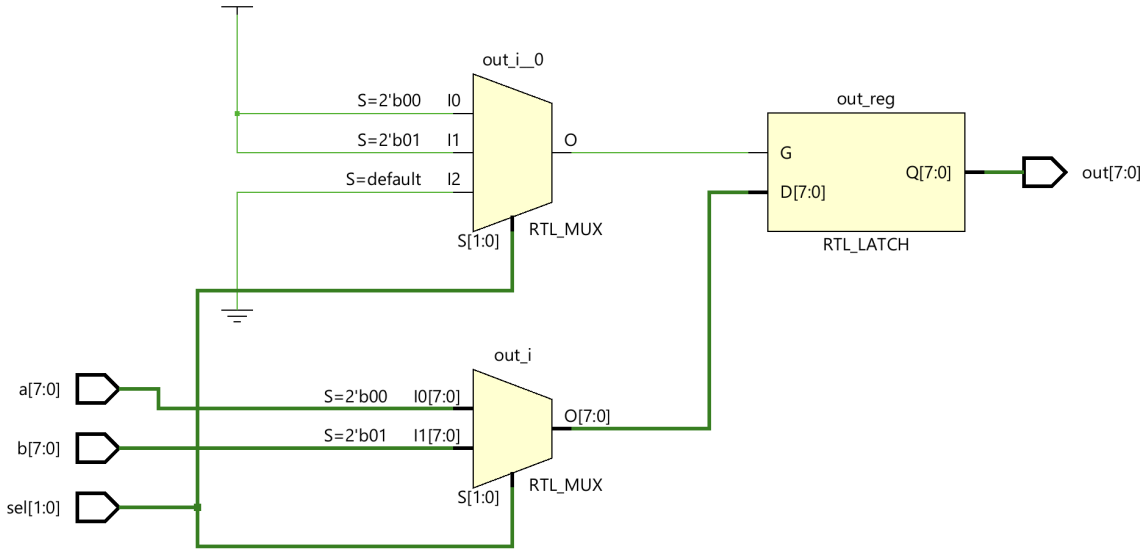



Рис. 14: Засчелка

Избавляемся от засчелки:

```
‘timescale 1ns / 1ps

module test(
    input [1:0] sel,
    input [7:0] a, b,
    output reg [7:0] out
);
    always @(*) begin
        case (sel)
            2'b00: out = a;
            2'b01: out = b;
            default: out = a;
        endcase
```

```
    end
endmodule
```

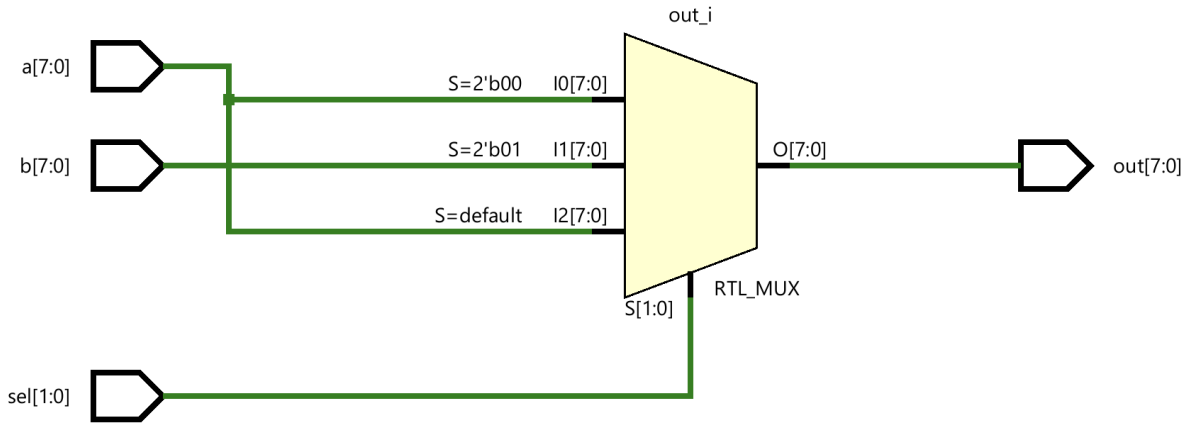



Рис. 15: Без засчелки

- В последовательностной логике появляется дополнительный мультиплексор, если не было управления разрешением (enable)

```
‘timescale 1ns / 1ps

module test(
    input [1:0] sel,
    input clk,
    input [7:0] a, b,
    output reg [7:0] out
);
always @(posedge clk) begin
    case (sel)
        2'b00: out <= a;
        2'b01: out <= b;
    endcase
end
endmodule
```

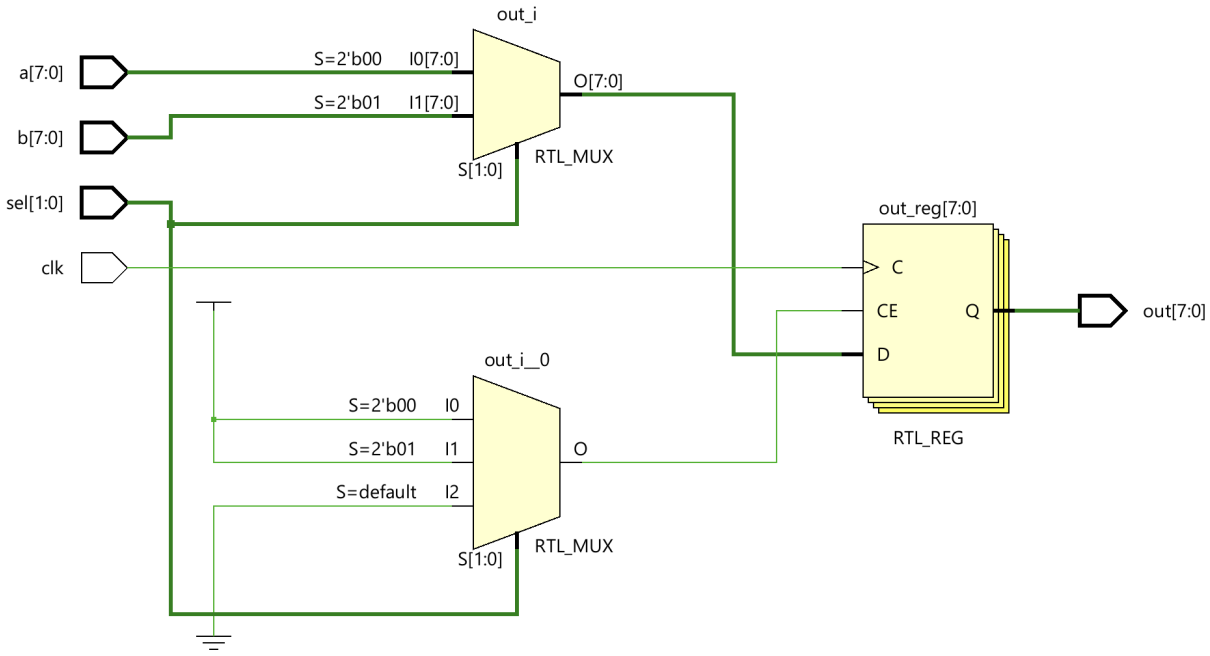



Рис. 16: Дополнительный мультиплексор

```
‘timescale 1ns / 1ps

module test(
    input [1:0] sel,
    input clk,
    input [7:0] a, b,
    output reg [7:0] out
);
always @(posedge clk) begin
    case (sel)
        2'b00: out <= a;
        2'b01: out <= b;
        default: out <= a;
    endcase
end
endmodule
```

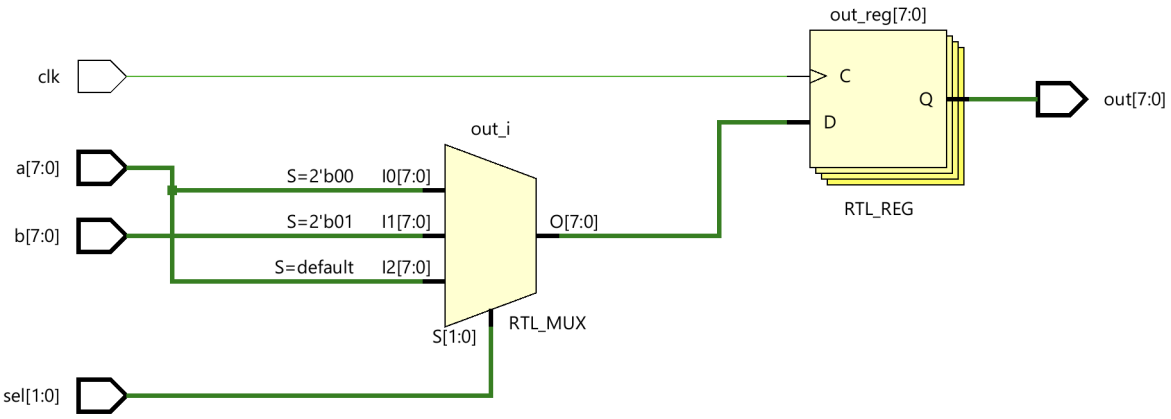



Рис. 17: Один мультиплексор

Вопрос: 18-19 counter что будет если убрать else ,что если оставить

```
‘timescale 1ns / 10ps

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
```

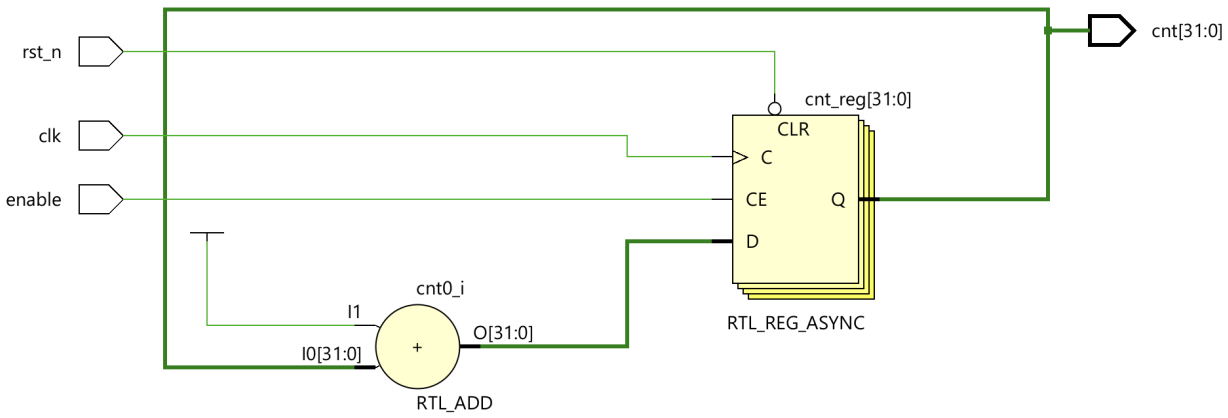



Рис. 18: RTL схема счетчика

Ничего не измениться так как и без else были описаны все возможные входные значения.

**Вопрос, задание: сократить количество FF**

| RQA Score | QoR Suggestions | LUT | FF | BRAM | URAM | DSP | Start | Elapsed | Run Strategy | Report Strategy | Part |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 286 | 182 | 0 | 0 | 0 | 4/20/24, 10:39 PM | 00:01:00 | Vivado Synthesis Defaults (Vivado Synthesis 2023) | Vivado Synthesis Default Reports (Vivado Synthesis 2023) | xc7a100tcs |

Рис. 19: Было

| Total Power | Failed Routes | Methodology | RQA Score | QoR Suggestions | LUT | FF | BRAM | URAM | DSP | Start | Elapsed | Run Strategy | Report Strategy |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | 208 | 113 | 0 | 0 | 0 | 4/22/24, 11:45 AM | 00:00:52 | Vivado Synthesis Defaults (Vivado Synthesis 2023) | Vivado Synthesis Default Reports (Vivado Synth |
| | | | | | | | | | | | | Vivado Implementation Defaults (Vivado Implementation 2023) | Vivado Implementation Default Reports (Vivad |

Рис. 20: Стало