

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 Р33102
Лапин Алексей Александрович
Преподаватель:
Васильев С.Е.

г. Санкт-Петербург
2024г.

Содержание

| | | |
|----------|--|-----------|
| 1 | Задание | 3 |
| 1.1 | Описание лабораторной работы | 3 |
| 1.2 | Таблица варианта | 3 |
| 2 | Выполнение | 3 |
| 2.1 | Счётчик | 3 |
| 2.1.1 | Разработанный модуль | 4 |
| 2.1.2 | Тестовый план: | 5 |
| 2.2 | Сдвиговый регистр | 7 |
| 2.2.1 | Разработанный модуль | 8 |
| 2.2.2 | Тестовый план: | 9 |
| 2.3 | Конечный автомат | 11 |
| 2.3.1 | Разработанный модуль | 13 |
| 2.3.2 | Тестовый план: | 17 |
| 2.4 | Делитель частоты | 21 |
| 2.4.1 | Разработанный модуль | 21 |
| 2.4.2 | Тестовый план: | 22 |
| 3 | Функция COUNT FREE | 25 |
| 4 | FIFO | 25 |
| 5 | Вывод | 26 |

1 Задание

1.1 Описание лабораторной работы

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

В первой части работы предлагается разработать несколько простых блоков цифровой последовательной 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

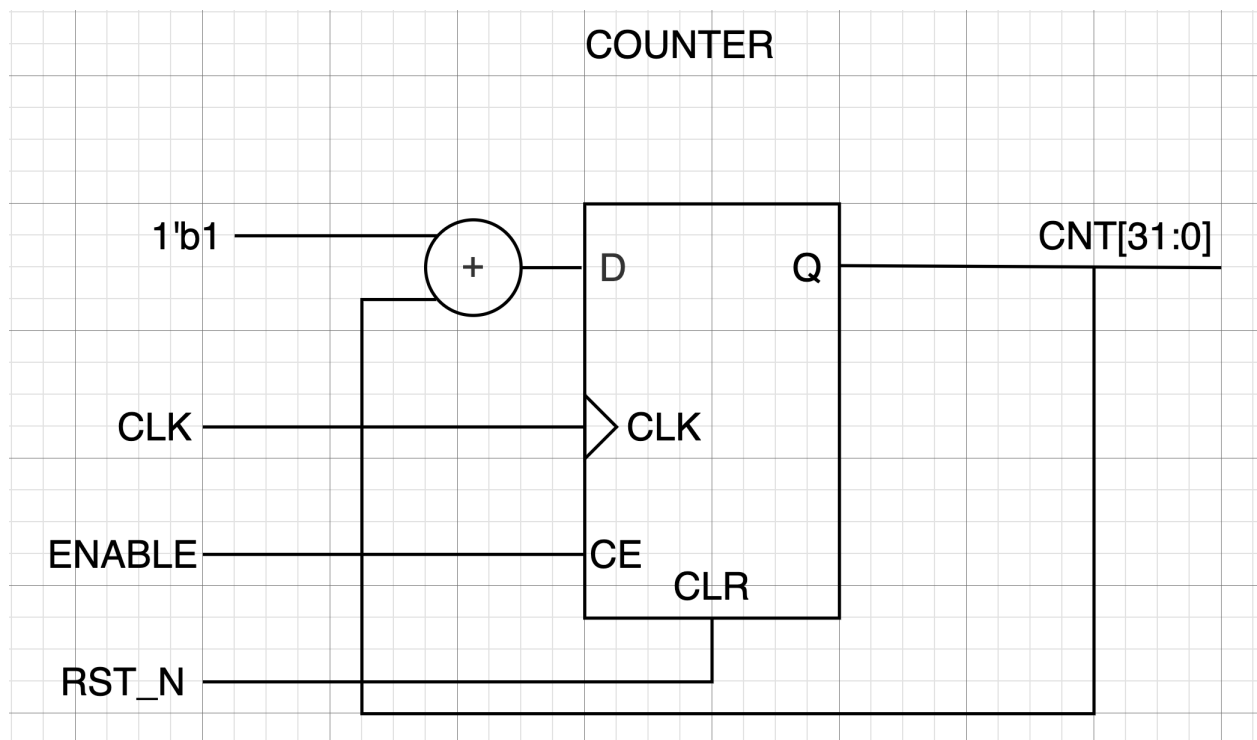


Рис. 1: Синхронный счетчик по переднему фронту с асинхронным сбросом и сигналом разрешения, 32 разряда.

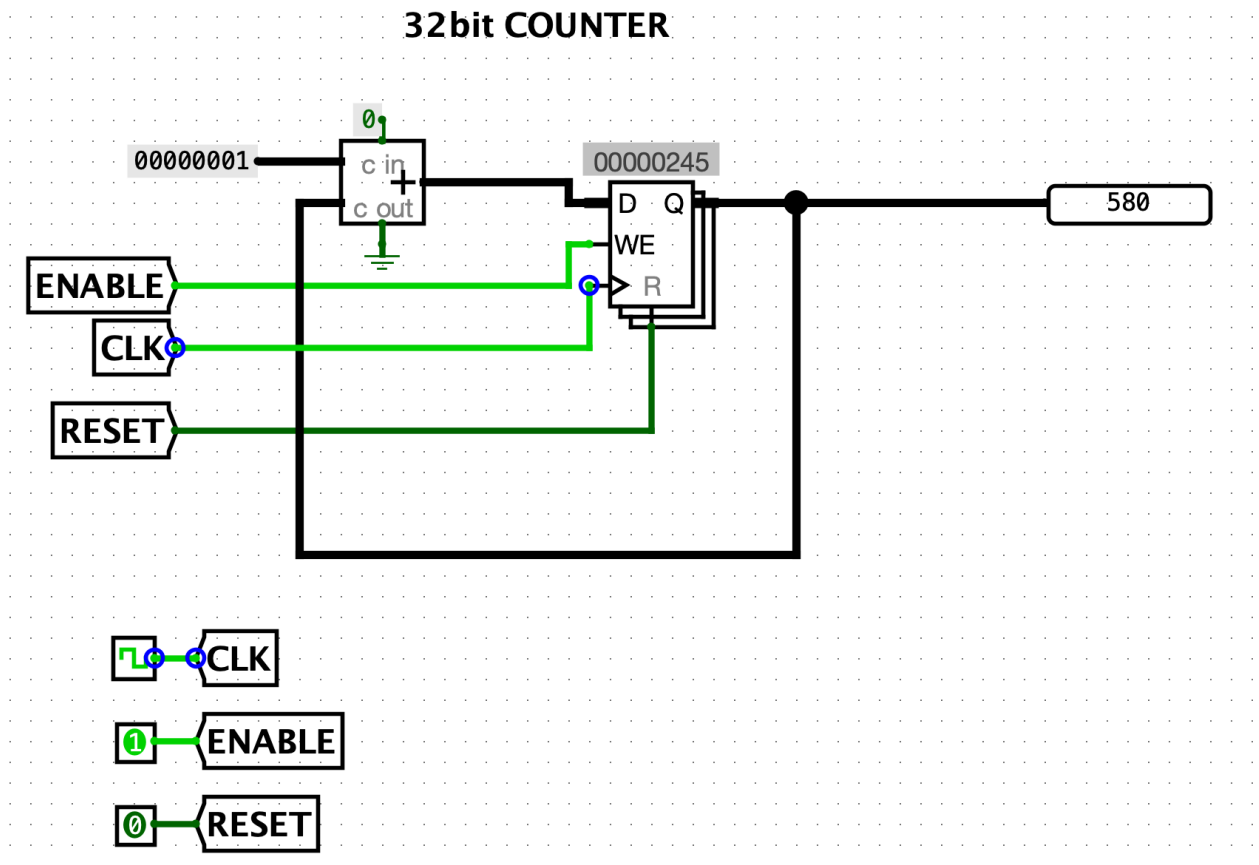


Рис. 2: Синхронный счетчик по переднему фронту с асинхронным сбросом и сигналом разрешения, 32 разряда.

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

```

1 module counter
2 #(
3     parameter WIDTH = 32
4 )
5 (
6     input clk,
7     input rst_n,
8     input enable,
9     output reg [WIDTH - 1:0] cnt
10 );
11 always @(posedge clk or negedge rst_n) begin
12     if(!rst_n)
13         cnt <= {WIDTH{1'b0}};
14     else if (enable)
15         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",
33                 $time, tst_out, cnt);
34             passed = 1'b0;
35         end
36     end
37     endtask
38
39     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

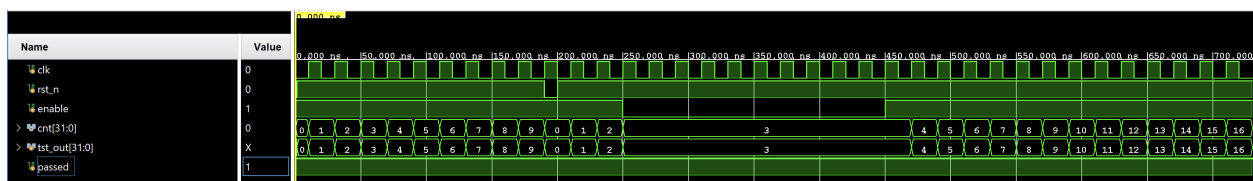


Рис. 3: Временная диаграмма работы счетчика

2.2 Сдвиговый регистр

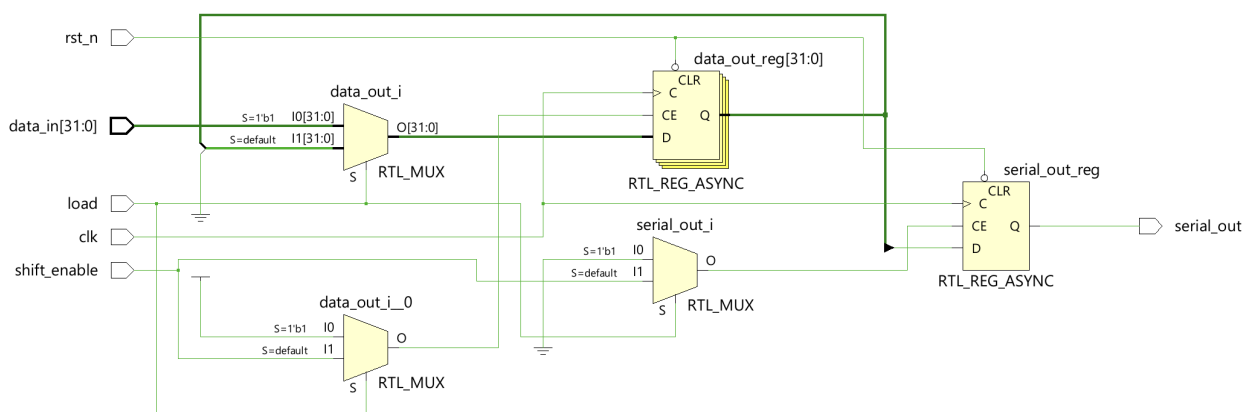


Рис. 4: Схема сдвигового регистра с параллельной загрузкой и последовательным сдвигом вправо

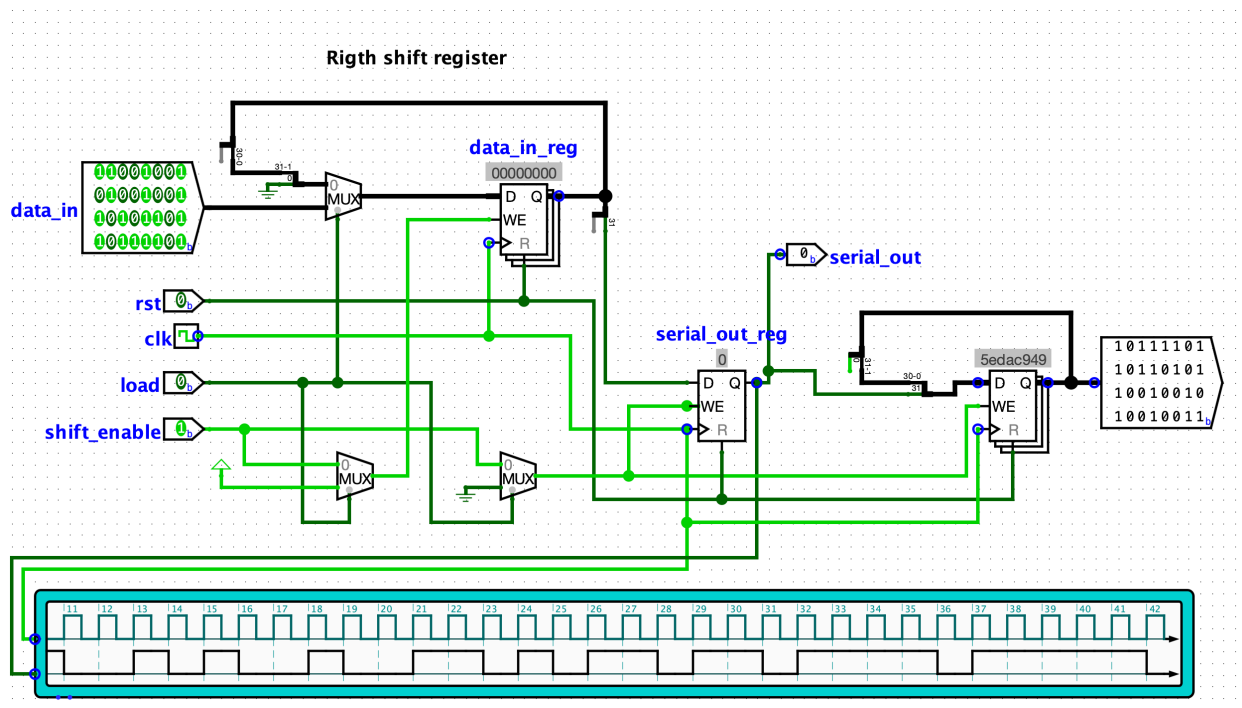


Рис. 5: Схема сдвигового регистра с параллельной загрузкой и последовательным сдвигом вправо

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

```

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;
25     end
26
27 endmodule

```

2.2.2 Тестовый план:

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

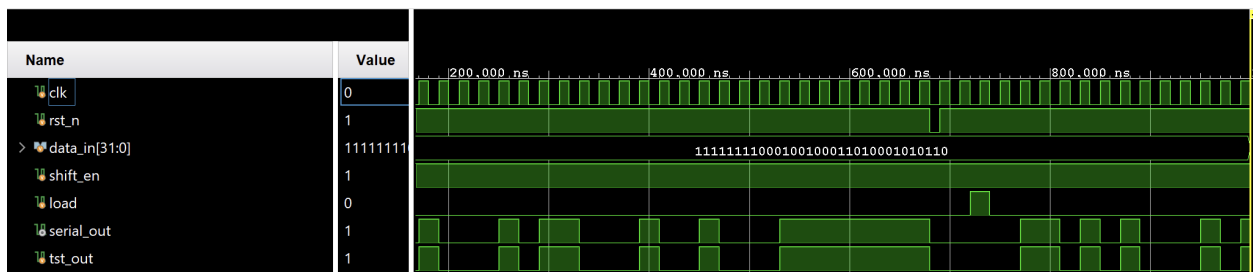


Рис. 6: Временная диаграмма работы сдвигового регистра

```

1  'timescale 1ns/100ps
2  'include "shift_right/src/shift_right.v"
3
4  module shift_right_tb;
5      localparam WIDTH=32;
6
7      reg clk;
8      reg rst_n;
9      reg [WIDTH-1:0] data_in;
10     reg shift_en;
11     reg load;
12     wire serial_out;
13
14     reg passed = 1;
15
16     reg tst_out;
17     reg [WIDTH-1:0] tst_temp;
18     reg [WIDTH-1:0] tst_value = 32'hFF123456;
19
20     shift_right #(.WIDTH(WIDTH)) shift_right_dut (
21         .clk(clk),
22         .load(load),
23         .rst_n(rst_n),
24         .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",
39                 $time, tst_out, serial_out);
40             passed = 0;
41         end
42     end
43     endtask
44
45     integer i;
46
47     initial begin
48         $dumpfile("build/shift_right.vcd");
49         $dumpvars(1);
50         $display("##### Starting simulation
51             #####");
52
53         shift_en = 0;
54         rst_n = 1;
55         load = 0;
56
57         tst_temp = tst_value;
58
59         @(negedge clk) begin
60             data_in <= tst_value;
61             load <= 1;
62         end
63         @(negedge clk) begin
64             load <= 0;
65         end
66         shift_en <= 1;
67
68         /* Test: Load value */
69         $display("##### TEST: Load value #####");
70         for (i = 0; i < 32; i = i + 1) begin
71             @(posedge clk)
72                 {tst_temp[WIDTH-1:0], tst_out} <= {1'b0,
73                     tst_temp[WIDTH-1:0]};
74             @(negedge clk) check;
75         end
76
77         /* 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,
100                 tst_temp[WIDTH-1:0]};
101         @(negedge clk) check;
102     end
103     if(passed)
104         $display("[T=%0g] All tests passed", $time);
105     else
106         $display("[T=%0g] Some tests failed", $time);
107     #10; $finish;
108
109 end
110 endmodule

```

Результаты тестирования:

```

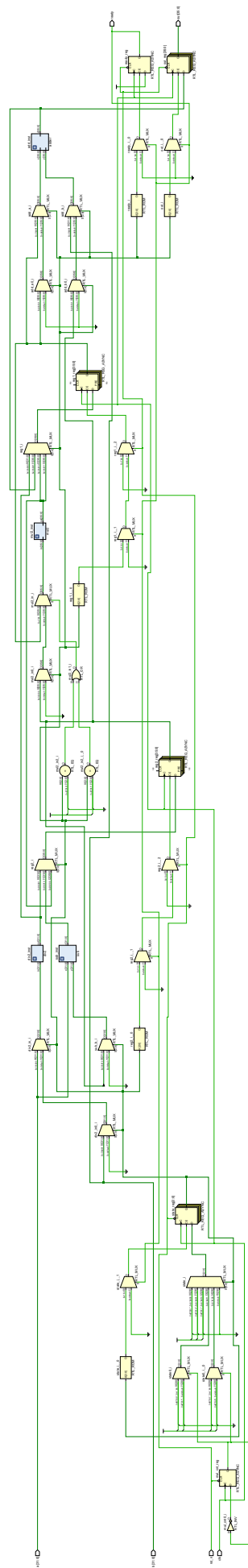
##### Starting simulation #####
##### TEST: Load value #####
##### TEST: Reset #####
##### TEST: Buffer is empty #####
##### TEST: Shift right continue #####
[T=1060] All tests passed

```

2.3 Конечный автомат

Схема конечного автомата

Конечный автомат состоит из 7 состояний, которые соответствуют 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

```
1 'timescale 1ps/1ps
2 module adder
3 #(
4     parameter WIDTH = 32
5 )
6 (
7     input [WIDTH - 1:0] x, y,
8     output [WIDTH - 1:0] z
9 );
10     assign z = x + y;
11 endmodule
```

Делитель на 2

```
1 'timescale 1ps/1ps
2 module div2
3 #(
4     parameter WIDTH = 32
5 )
6 (
7     input [WIDTH - 1:0] in,
8     output [WIDTH - 1:0] out
9 );
10     assign out = in >> 1;
11 endmodule
```

Умножитель на 8

```
1 'timescale 1ps/1ps
2 module mul2
3 #(
4     parameter WIDTH = 32
5 )
6 (
7     input [WIDTH - 1:0] in,
8     output [WIDTH - 1:0] out
9 );
10     assign out = in << 1;
11 endmodule
```

Вычитатель

```
1 'timescale 1ps/1ps
2 module sub
3 #(
4     parameter WIDTH = 32
5 )
6 (
7     input [WIDTH - 1:0] x, y,
```

```

8     output [WIDTH - 1:0] z
9 );
10    assign z = x + (~y) + 1;
11
12 endmodule

```

Конечный автомат

```

1  'include "fsm/src/sub.v"
2  'include "fsm/src/adder.v"
3  'include "fsm/src/mul2.v"
4  'include "fsm/src/div2.v"
5  module fsm
6  #(
7      parameter WIDTH = 32,
8      parameter OP_WIDTH = WIDTH + 3
9  )
10 (
11     input clk,
12     input [WIDTH - 1:0] a, b,
13     input rst_n,
14     output reg [OP_WIDTH-1:0] out,
15     output reg ready
16 );
17
18     localparam [3:0]
19         S0 = 4'b0000,
20         S1 = 4'b0001,
21         S2 = 4'b0010,
22         S3 = 4'b0011,
23         S4 = 4'b0100,
24         S5 = 4'b0101,
25         S6 = 4'b0110,
26         S7 = 4'b0111;
27     reg [3:0] state;
28
29     /* registers to store intermediate results */
30     reg [OP_WIDTH - 1:0] reg1, reg2, reg3, reg4;
31
32     reg [1:0] mul_cnt = 0;
33
34     /* instantiate div2 module */
35     reg [OP_WIDTH - 1:0] div2_in;
36     wire [OP_WIDTH - 1:0] div2_out;
37     div2 #(.WIDTH(OP_WIDTH)) div2_inst (
38         .in(div2_in),
39         .out(div2_out)
40     );
41
42     /* instantiate adder module */
43     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
150     S2: begin
151         add_a = reg1;
152         add_b = reg2;

```



```

153         end
154     S3: begin
155         div2_in = reg4;
156     end
157     S4: begin
158         sub_a = reg3;
159         sub_b = reg4;
160     end
161     S5: begin
162         mul2_in = reg1;
163     end
164     S6: begin
165         mul2_in = reg3;
166     end
167     S7: begin
168         add_a = reg1;
169         add_b = reg3;
170     end
171     default: begin
172         div2_in = 0;
173         add_a = 0;
174         add_b = 0;
175         mul2_in = 0;
176         sub_a = 0;
177         sub_b = 0;
178     end
179 endcase
180 end
181 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 $b = -2^{31}$

```

1 'timescale 1ps/1ps
2 'include "fsm/src/fsm.v"
3 module fsm_tb;
4     localparam WIDTH = 32;
5     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",
32                 $time, tst_out, out);
33             passed = 0;
34         end
35     endtask
36
37     task calfunc(input [WIDTH - 1:0] a, b, output [OP_WIDTH-1:0]
38         out);
39     begin
40         out = ((a / 2) + b) * 8 + (a - (b / 2)) * 4;
41     end
42     endtask
43
44     initial begin
45         $dumpfile("build/fsm.vcd");
46         $dumpvars(1);
47
48         $monitor("[T=%0d] state->%0d, reg1=%0d, reg2=%0d, reg3=%00d,
49             reg4=%0d, mul_cnt=%0d, ready=%b, out=%0d",
50             $time, fsm_dut.state, fsm_dut.reg1, fsm_dut.reg2,
51             fsm_dut.reg3, fsm_dut.reg4, fsm_dut.mul_cnt,
52             fsm_dut.ready, fsm_dut.out);
53
54         rst_n = 0;
55
56         /* Test: a = 6 and b = 4 */
57         $display("[T=%0g] Test 1: a = 6, b = 4", $time);
58         @(negedge clk) begin
59             a <= 32'd6;
60             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",
87         $time);
88     @(negedge clk) begin
89         // a = 2^31 - 1
90         a <= 32'h7FFFFFFF;
91         // b = 2^31 - 1
92         b <= 32'h7FFFFFFF;
93         rst_n <= 1;
94     end
95     while (!ready) begin
96         @(posedge clk);
97     end
98     calfunc(a, b, tst_out);
99     @(posedge clk) check;
100
101     /* Test: Zero values a = 0 and b = 0 */
102     $display("[T=%0g] Test 4: a = 0, b = 0", $time);
103     @(negedge clk) begin
104         a <= 32'h0;
105         b <= 32'h0;
106         rst_n <= 0;
107     end
108     @(negedge clk) begin
109         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

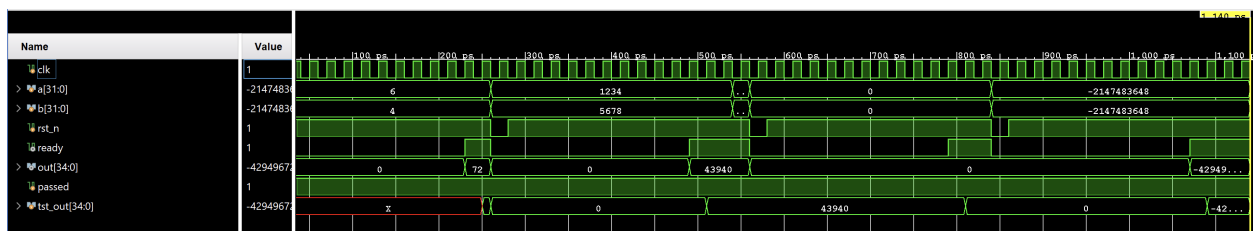


Рис. 7: Временная диаграмма работы конечного автомата

2.4 Делитель частоты

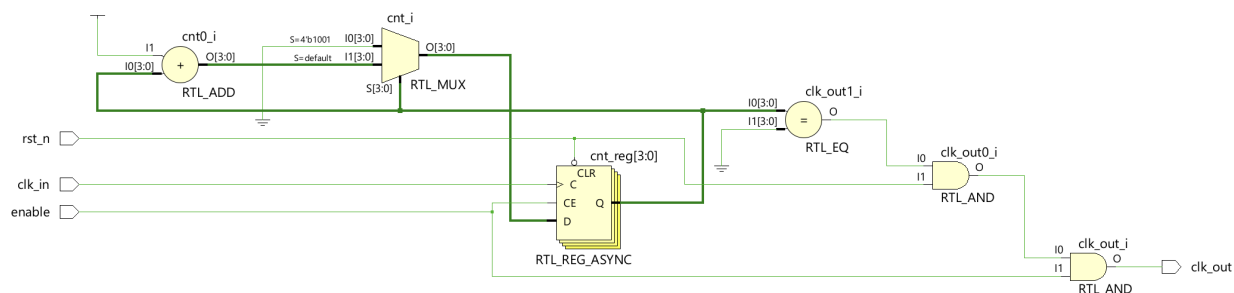


Рис. 8: Схема делителя частоты, уменьшает частоту на 10 раз

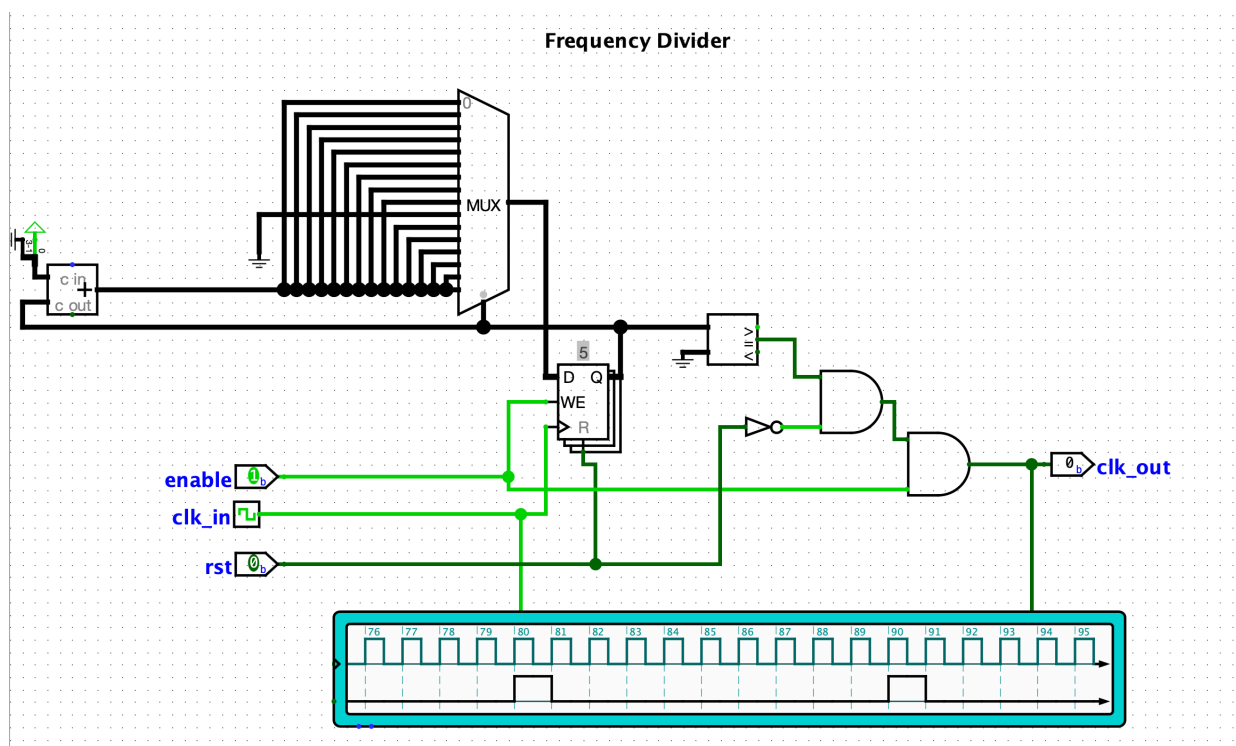


Рис. 9: Схема делителя частоты, уменьшает частоту на 10 раз

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 становится 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)
20                cnt <= {WIDTH{1'b0}};
21            else
22                cnt <= cnt + 1'b1;
23        else
24            cnt <= cnt;
25    end
26
27    assign clk_out = (cnt == 0 && rst_n && enable) ? 1'b1 : 1'b0;
28 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.

```

1 `timescale 1ns/1ps
2 `include "freq_div/src/freq_div.v"
3
4 module freq_div_tb;
5     localparam DIV_CNT = 10;
6     reg  clk_in;
7     reg  rst_n;
8     reg  enable;
9     wire clk_out;
10    freq_div #(.DIV_CNT(DIV_CNT)) fd_dut(
11        .clk_in (clk_in),
12        .rst_n  (rst_n),
13        .enable (enable),
14        .clk_out (clk_out)
15    );
16
17    reg passed = 1;
18    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",
31                  $time, tst_out, clk_out);
32              passed = 1'b0;
33          end
34      end
35  endtask
36
37  integer i;
38  initial begin
39      $dumpfile("build/freq_div.vcd");
40      $dumpvars(1);
41      $monitor("[T=%0d] enable=%b, rst_n=%b, cnt=%0d, clk_out=%b",
42          $time, enable, rst_n, fd_dut.cnt, clk_out);
43
44      rst_n = 0;
45      enable = 1;
46      #1 rst_n = 1;
47
48      tst_out = 1'b0;
49
50      /* Test 1: Count test */
51      $display("[T=%0g] Test 1: Count test", $time);
52      for(i = 1; i < 98; i = i + 1) begin
53          @(posedge clk_in) begin
54              if(i % DIV_CNT == 0)
55                  tst_out = 1'b1;
56              else tst_out = 1'b0;
57          end
58          @(negedge clk_in) begin
59              check;
60          end
61      end
62
63      /* Test 2: Reset, Should Be 0 */
64      $display("[T=%0g] Test 2: Reset, Should Be 0", $time);
65      @(posedge clk_in) begin
66          rst_n <= 0;
67          tst_out <= 0;
68      end
69      @(negedge clk_in) check;

```

```

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

```

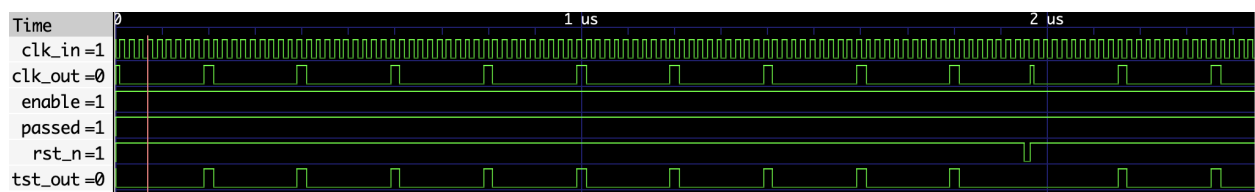


Рис. 10: Временная диаграмма работы делителя частоты

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

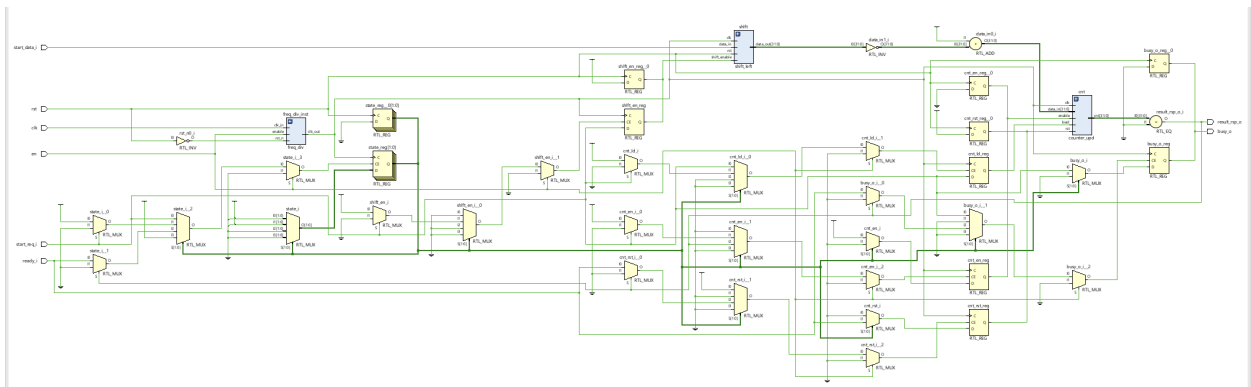


Рис. 11: Схема функции COUNT_FREE

Результаты тестирования:

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

4 FIFO

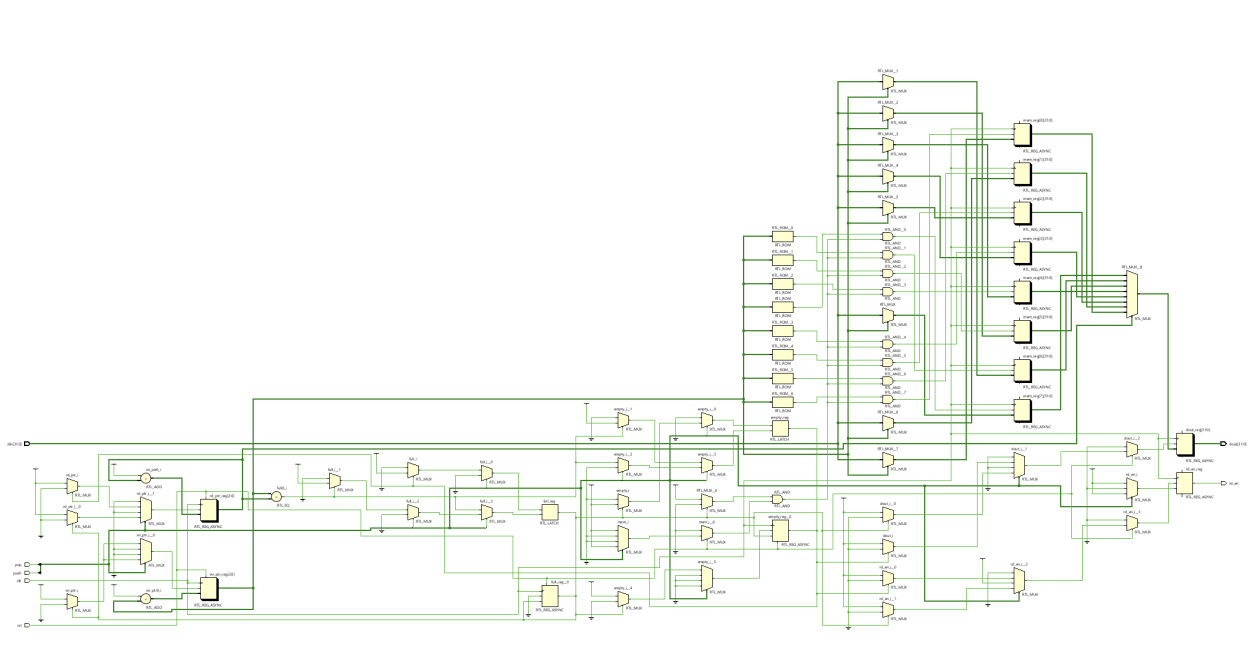


Рис. 12: Схема FIFO

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

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 логике это ни на что не влияет.