Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и кибербезопасности
Высшая школа компьютерных технологий и информационных систем

Отчёт по лабораторной работе № 10

Дисциплина: Языки описания аппаратных средств вычислительных систем.

Выполнил студент гр. 5130901/10101 _____ Д.Л. Симоновский
(подпись)

Руководитель _____ А.А. Федотов
(подпись)

“01” ноября 2023 г.

Санкт-Петербург
2023

Оглавление

1. Задача:

На языке Verilog разработать:

- Модуль ss_cntr – параметризированный модуль управления динамическим отображением для 4-разрядного 7-сегментного индикатора.
- Модуль lab_2s – оберточный модуль, осуществляющий подключение к выводам компонента ss_cntr выводов, на плате; задание константных значений выводам компонента ss_cntr; задание коэффициента деления счетчика.

Модуль ss_cntr:

Входы:

- clk – тактовый сигнал.
- rst_n – синхронный сброс, активный уровень – 0.
- [3:0] A – вход данных.
- [3:0] B – вход данных.
- [3:0] C – вход данных.
- [3:0] D – вход данных.

Выходы:

- [6:0] ss – выходы данных для 7-сегментного индикатора.
- [4:1] dig – выходы управления включением разрядов 4-разрядного 7-сегментного индикатора.

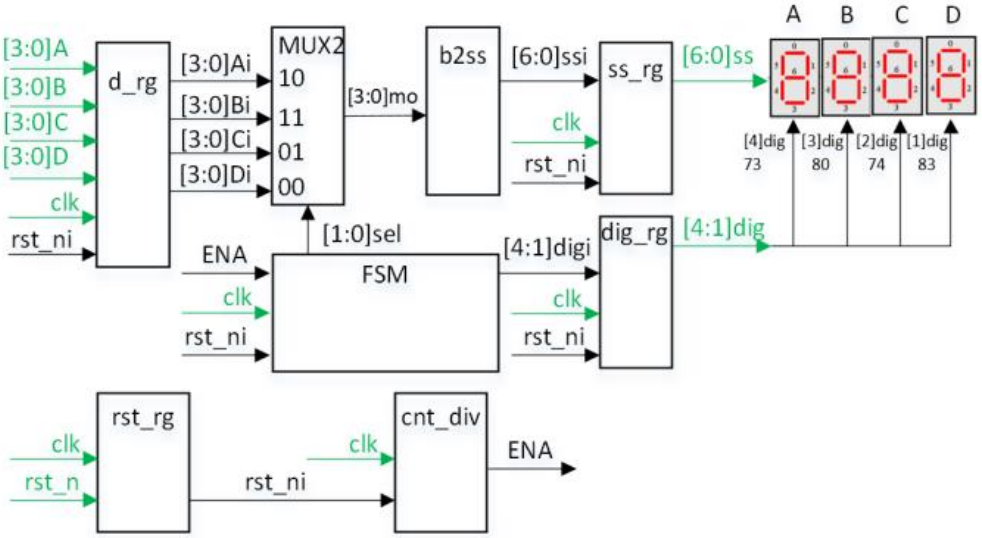

Рис. 1.1. Структура устройства ss_cntr.

Состав устройства:

- rst_rg – синхронизатор сигнала сброса, компонент должен содержать 2 последовательно включенных триггера.
- d_rg – компонент, содержащий 4 регистра для хранения 4-х наборов входных данных.
- MUX2 – параметризированный мультиплексор 4(Nбит) =>1(N бит), реализованный как комбинационная схема. Параметр N – разрядность мультиплексора (базовое значение = 4).
- b2ss – преобразователь двоичного кода в 7-сегментный реализованный как комбинационная схема.
- cnt_div – счетчик-делитель с параметризированным коэффициентом деления.
- FSM – конечный автомат, обеспечивающий управление блоком MUX и формирование сигналов включения разрядов 7-сегментного индикатора.

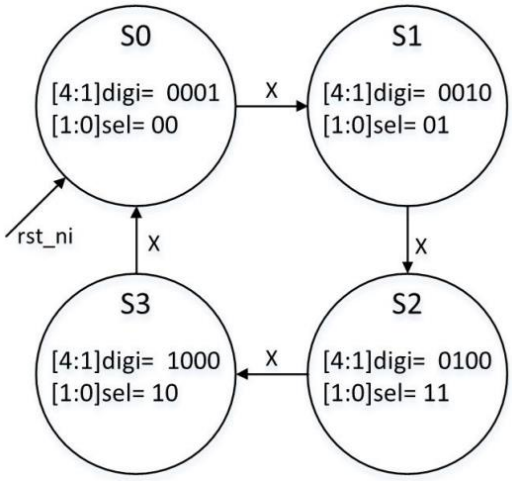

Рис. 1.2. Граф переходов модуля FSM.

- ss_rg – регистр хранения выходных данных 7-сегментного индикатора.
- dig_rg – регистр хранения сигналов управления 7-сегментного индикатора [4:1] dig).

Модуль lab_2s:

Входы:

- clk – тактовый сигнал.
- rst_n – синхронный сброс, активный уровень – 0.

Выходы:

- [6:0] ss – выходы данных для 7-сегментного индикатора.
- [4:1] dig – выходы управления включением разрядов 4-разрядного 7-сегментного индикатора.

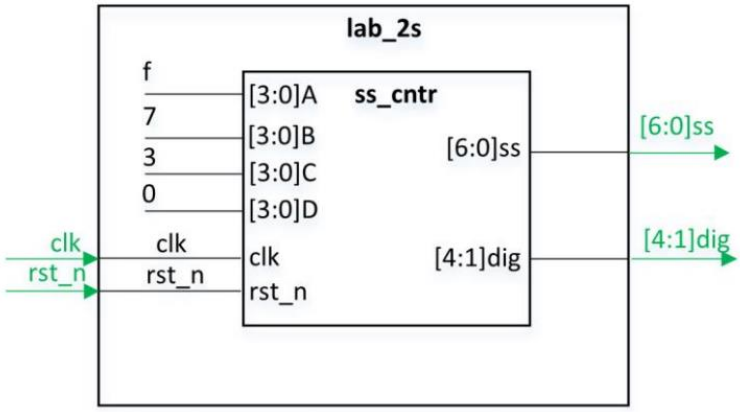

Рис. 1.3. Структура устройства lab_2s.

Состав устройства:

- ss_rst – компонент разработанного модуля.

2. Решение:

Реализуем каждое из устройств, начнем с FSM, это конечный автомат, граф которого приведен на Рис. 1.2, реализуем его следующим образом:

```
1   module FSM
2   (
3           input rst_ni, clk, ENA,
4           output reg [4:1] digi,
5           output reg [1:0] sel
6   );
7
8   parameter S0 = 0, S1 = 1, S2 = 2, S3 = 3;
9   reg [1:0] state = S0;
10
11  always @(posedge clk or negedge rst_ni) begin
12          if (~rst_ni)
13                  state <= S0;
14          else if (ENA) case (state)
15                  S0: state <= S1;
16                  S1: state <= S2;
17                  S2: state <= S3;
18                  S3: state <= S0;
19          endcase
20  end
21
22  always @(state) begin
23          case (state)
24                  S0: begin
25                          digi = 4'b0001;
26                          sel = 2'b00;
27                  end
28                  S1: begin
29                          digi = 4'b0010;
30                          sel = 2'b01;
31                  end
32                  S2: begin
33                          digi = 4'b0100;
34                          sel = 2'b11;
35                  end
36                  S3: begin
37                          digi = 4'b1000;
38                          sel = 2'b10;
39                  end
40          endcase
41  end
42
43  endmodule
```

Рис. 2.1. FSM на Verilog.

Выполним компиляцию и синтез, чтоб удостовериться, что схема действительно скомпилировалась, как конечный автомат:

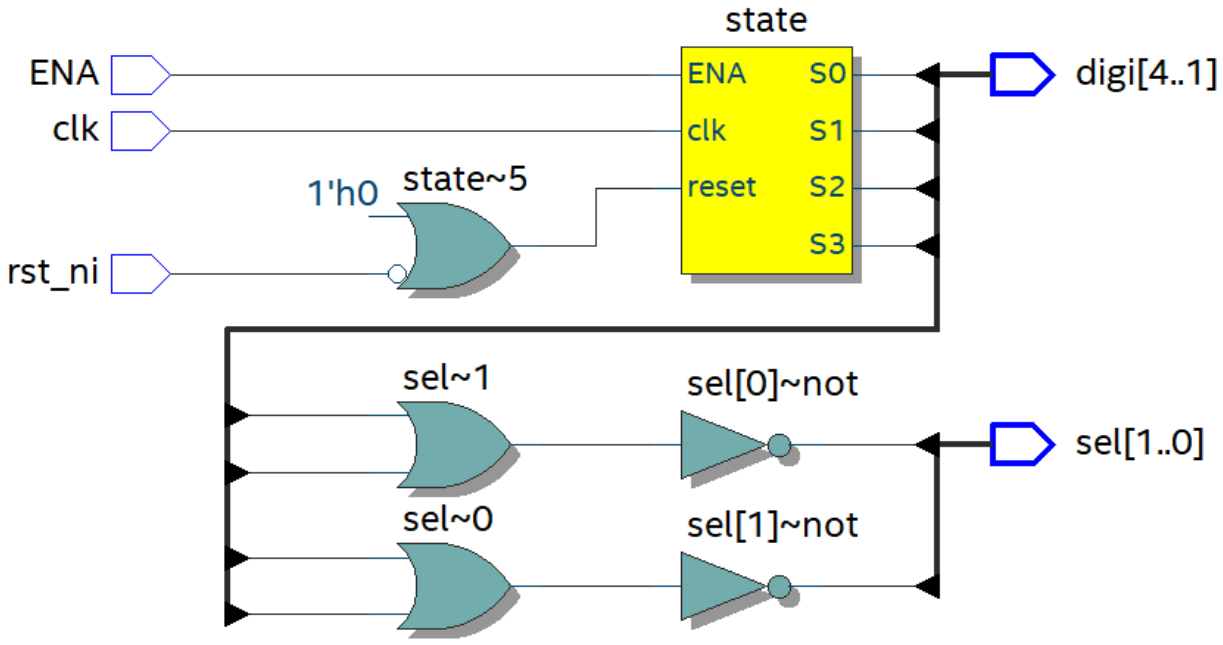

Рис. 2.2. RTL Viewer.

Далее реализуем тестовый модуль FSM_tb:

```
module FSM_tb ();
reg rst_ni, clk, ENA;
reg [4:1] digi;
reg [1:0] sel;

FSM FSM (rst_ni, clk, ENA, digi, sel);

localparam CLK_PERIOD = 20;

initial begin
        clk = 1'b0;
        forever #(CLK_PERIOD / 2) clk = ~clk;
end

function check (
        input[4:1] exp_digi,
        input[1:0] exp_sel
);
        if (exp_digi != digi || exp_sel != sel) begin
                $display("Expected: %d, %d\nReal: %d, %d", exp_digi, exp_sel, digi, sel);
                $stop;
        end
endfunction


initial begin
        rst_ni = 1'b1;
        ENA = 1'b1;
        #CLK_PERIOD
        check(4'b0010, 2'b01);
        #CLK_PERIOD
        check(4'b0100, 2'b11);
        #CLK_PERIOD
        check(4'b1000, 2'b10);
        ENA = 1'b0;
        #CLK_PERIOD
        check(4'b1000, 2'b10);
        ENA = 1'b1;
        #CLK_PERIOD
        check(4'b0001, 2'b00);
        #CLK_PERIOD
        check(4'b0010, 2'b01);
        rst_ni = 1'b0;
        #(CLK_PERIOD / 10)
        check(4'b0001, 2'b00);
        rst_ni = 1'b1;
        #(CLK_PERIOD / 10 * 9)
        check(4'b0010, 2'b01);
        $display("All tests have been passed!");
        $stop;
end

endmodule
```

Рис. 2.3. FSM_tb на Verilog.

Это тест 2 класса, все ожидаемые значения записаны прямо в тесте, для удобства. Помимо самих переходов проверена работа входа ENA и rst_ni. Все тесты выполняются корректно, wave выглядит следующим образом:

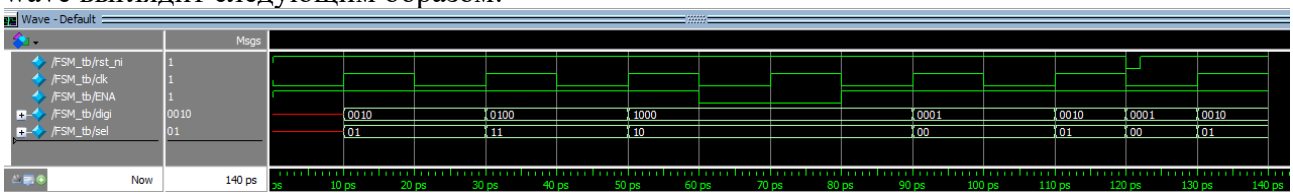

Рис. 2.4. Wave для FSM_tb.

На ней видно, что модуль работает корректно, в том числе асинхронный сброс.

Далее реализуем модуль MUX:

```verilog
module MUX
#(parameter N = 4)
(
        input [N - 1:0] Ai, Bi, Ci, Di,
        input [1:0] sel,
        output reg [N - 1:0] mo
);

always @*
        case (sel)
                2'b10: mo <= Ai;
                2'b11: mo <= Bi;
                2'b01: mo <= Ci;
                2'b00: mo <= Di;
        endcase

endmodule
```

Рис. 2.5. MUX на Verilog.

Этот модуль выполнен в соответствии с этой схемой:

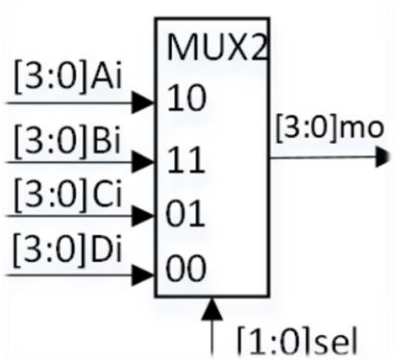

Рис. 2.6. Схема модуля MUX.

Удостоверимся, что полученная схема действительно комбинационная, как этого требует задание:

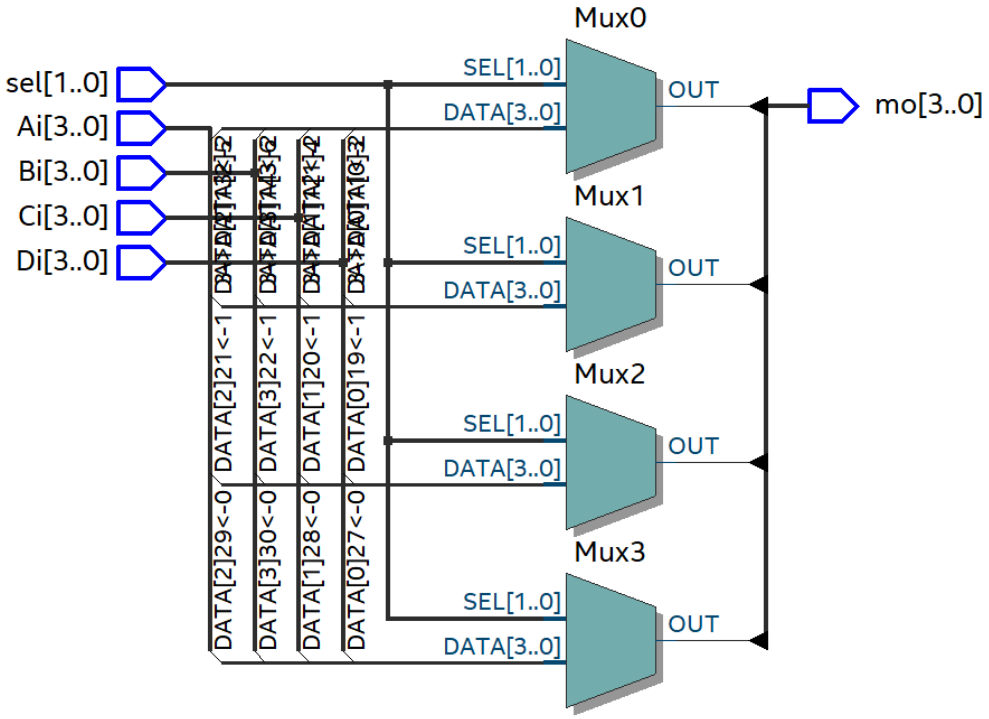

Рис. 2.7. RTL Viewer модуля MUX.

Выполним тестирование этого модуля, используя тестовый модуль MUX_tb:

```
module MUX_tb ();
localparam N = 4;
reg [N - 1:0] Ai, Bi, Ci, Di;
reg [1:0] sel;
reg [N - 1:0] mo;

MUX #(.N(N)) MUX2 (Ai, Bi, Ci, Di, sel, mo);

reg [N - 1:0] excepted_result;
integer input_file, answers_file, char_1, char_2;

initial begin
        `define eof 32'hffff_ffff
        input_file = $fopen("input_file.txt", "r");
        answers_file = $fopen("except_output.txt", "r");
        char_1 = $fgetc(input_file);
        char_2 = $fgetc(answers_file);
        while (char_1 != `eof && char_2 != `eof) begin
                $ungetc(char_1, input_file);
                $ungetc(char_2, answers_file);
                $fscanf(input_file, "%b", Ai);
                $fscanf(input_file, "%b", Bi);
                $fscanf(input_file, "%b", Ci);
                $fscanf(input_file, "%b", Di);
                $fscanf(input_file, "%b", sel);
                $fscanf(answers_file, "%b", excepted_result);

                #1
                if (excepted_result != mo) begin
                        $display("Incorrect output:\nExcepted: %b\nActual: %b", excepted_result, mo);
                        $stop;
                end
                #1
                char_1 = $fgetc(input_file);
                char_2 = $fgetc(answers_file);
        end
        $display("All test have been passed!");
        $fclose(input_file);
        $fclose(answers_file);
        $stop;
end

endmodule
```

Рис. 2.8. Тестовый модуль MUX_tb.

Данный тестовый модуль берет значения из входного файла и сравнивает со значением из выходного. Модуль MUX работает корректно, все ожидаемые данные совпали с реальными. Wave выглядит следующим образом:

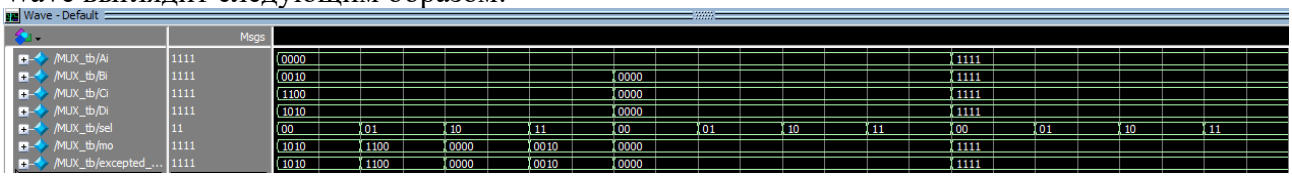

Рис. 2.9. Wave для MUX_tb.

Реализуем модуль ss_rg:

```
module ss_rg
(
        input [6:0] ssi,
        input clk,
        input rst_ni,
        output reg [6:0] ss
);

always @(posedge clk) begin
        if (~rst_ni)
                ss <= 1'b0;
        else
                ss <= ssi;
end

endmodule
```

Рис. 2.10. ss_rg на Verilog.

Реализуем модуль dig_rg:

```
module dig_rg
(
        input [4:1] digi,
        input clk,
        input rst_ni,
        output reg [4:1] dig
);

always @(posedge clk) begin
        if (~rst_ni)
                dig <= 1'b0;
        else
                dig <= digi;
end

endmodule
```

Рис. 2.11. dig_rg на Verilog.

Реализуем модуль d_rg:

```
module d_rg
(
        input [3:0] A, B, C, D,
        input clk, rst_ni,
        output reg [3:0] Ai, Bi, Ci, Di
);

always @(posedge clk) begin
        if (~rst_ni)
                {Ai, Bi, Ci, Di} <= 1'b0;
        else
                {Ai, Bi, Ci, Di} <= {A, B, C, D};
end

endmodule
```

Рис. 2.12. d_rg на Verilog.

Модуль rst_rg взят из lab_1s:

```
module rst_rg
(
         input clk, rst_n,
         output reg rst_n_i
);

reg rst_temp;

always @(posedge clk) begin
         rst_temp <= rst_n;
         rst_n_i <= rst_temp;
end

endmodule
```

Рис. 2.13. rst_rg на Verilog.

Модуль b2ss возьмем из lab3_2:

```
module b2ss
(
       input [3:0] a,
       output reg [6:0] d7seg
);

reg [6:0] ss_arr[15:0];

initial begin
       ss_arr[0] = 7'h3F; //0
       ss_arr[1] = 7'h06; //1
       ss_arr[2] = 7'h5B; //2
       ss_arr[3] = 7'h4F; //3
       ss_arr[4] = 7'h66; //4
       ss_arr[5] = 7'h6D; //5
       ss_arr[6] = 7'h7D; //6
       ss_arr[7] = 7'h07; //7
       ss_arr[8] = 7'h7F; //8
       ss_arr[9] = 7'h6F; //9
       ss_arr[10] = 7'h77; //A
       ss_arr[11] = 7'h7C; //B
       ss_arr[12] = 7'h39; //C
       ss_arr[13] = 7'h5E; //D
       ss_arr[14] = 7'h79; //E
       ss_arr[15] = 7'h71; //F
end

assign d7seg = ss_arr[a];

endmodule
```

Рис. 2.14. b2ss на Verilog.

Модуль cnt_div возьмем из lab_1s:

```
module cnt_div
#(parameter N = 5)
(
        input clk, rst,
        output reg ena
);

integer counter = 0;

always @(posedge clk)
        if (~rst) begin
                counter <= 0;
                ena <= 0;
        end else
                if (counter == N - 1) begin
                        counter <= 0;
                        ena <= 1;
                end else begin
                        counter <= counter + 1;
                        ena <= 0;
                end

endmodule
```

Рис. 2.15. cnt_div на Verilog.

Создадим структурное описание модуля ss_cntr:

```
module ss_cntr
#(parameter divider = 5)
(
         input [3:0] A, B, C, D,
         input clk, rst_n,
         output reg [6:0] ss,
         output reg [4:1] dig
);

reg rst_ni;

rst_rg rst_rg (clk, rst_n, rst_ni);

reg ENA;
reg [3:0] Ai, Bi, Ci, Di;

cnt_div #(.N(divider)) cnt_div (clk, rst_ni, ENA);
d_rg d_rg (A, B, C, D, clk, rst_ni, Ai, Bi, Ci, Di);

reg [1:0] sel;
reg [4:1] digi;

FSM FSM (rst_ni, clk, ENA, digi, sel);

reg [3:0] mo;

MUX MUX2 (Ai, Bi, Ci, Di, sel, mo);

reg [6:0] ssi;

b2ss b2ss (mo, ssi);
ss_rg ss_rg (ssi, clk, rst_ni, ss);
dig_rg dig_rg (digi, clk, rst_ni, dig);

endmodule
```

Рис. 2.16. ss_cntr на Verilog.

Это параметризированное устройство, где параметр – коэффициент деления.

Реализуем модуль-обертку lab_2s:

```
module lab_2s
#(parameter divider = 5)
(
        (* altera_attribute = "-name IO_STANDARD \"3.3-V LVCMOS\"", chip_pin = "23" *)
        input clk,
        (* altera_attribute = "-name IO_STANDARD \"2.5-V\"", chip_pin = "64" *)
        input rst_n,
        (* altera_attribute = "-name IO_STANDARD \"3.3-V LVCMOS\"", chip_pin = "84, 76, 85, 77, 86, 133, 87" *
        output reg [6:0] ss,
        (* altera_attribute = "-name IO_STANDARD \"3.3-V LVCMOS\"", chip_pin = "73, 80, 74, 83" *)
        output reg [4:1] dig
);

ss_cntr #(divider) ss_cntr (4'hF, 4'h7, 4'h3, 4'h0, clk, rst_n, ss, dig);

endmodule
```

Рис. 2.17. lab_2s на Verilog.

Выполним компиляцию и получим следующую схему на RTL Viewer:

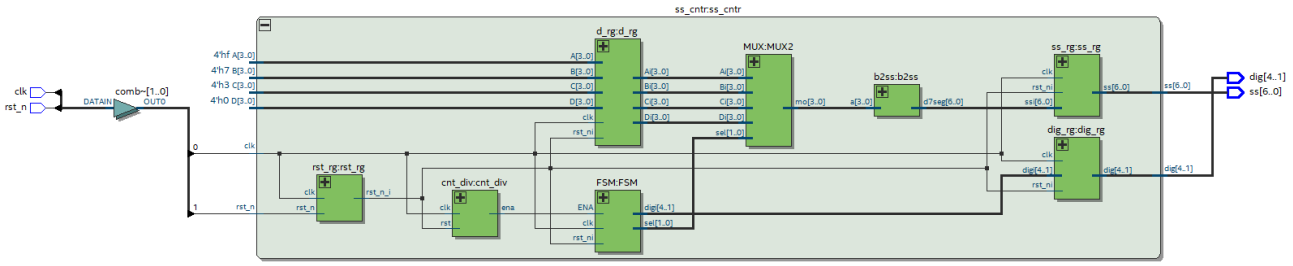

Рис. 2.18. RTL Viewer для lab_2s.

Создадим тест 1 класса для этого модуля:

```
module lab_2s_tb ();

reg clk, rst_n;
reg [6:0] ss;
reg [4:1] dig;

lab_2s #(5) lab2_s (clk, rst_n, ss, dig);
localparam CLK_PERIOD = 20;

initial begin
        clk = 1'b0;
        forever #(CLK_PERIOD / 2) clk <= ~clk;
end

initial begin
        rst_n = 1'b1;
        #(CLK_PERIOD * 50)
        rst_n = 1'b0;
        #(CLK_PERIOD * 10)
        rst_n = 1'b1;
        #(CLK_PERIOD * 10)
        $stop;
end

endmodule
```

Рис. 2.19. lab_2s_tb на Verilog.

Запустим его и получим следующую Wave:

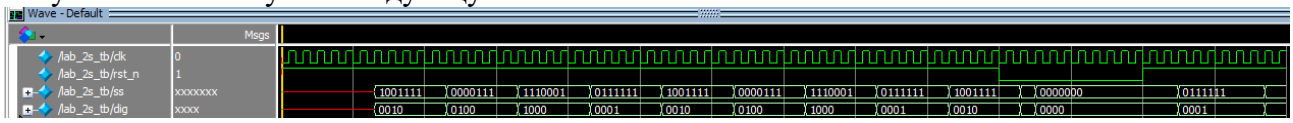

Рис. 2.20. Wave для модуля lab_2s_tb.

Устройство работает в соответствии с ожиданиями.

3. Вывод:

В ходе лабораторной работы было разработано устройство для динамической индикации на семисегментном индикаторе. Данное устройство позволяет выводить любое значение на индикатор.

Устройство было протестировано как в лаборатории на стенде, так и с использованием тестовых модулей, которые продемонстрировали корректность разработанной схемы.

Видно, что язык Verilog сильно упрощает разработку таких сложных устройств, по сравнению с вводом схем, используя блочную диаграмму. Так же ModelSim позволяет провести более комплексное тестирование устройства, что также ускоряет разработку.