



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

**«МИРЭА - Российский технологический университет»**

**РТУ МИРЭА**

---

Институт Информационных Технологий  
Кафедра Вычислительной Техники (ВТ)

**ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2**

**«Управление семисегментными индикаторами»**

по дисциплине

**«Схемотехника устройств компьютерных систем»**

Выполнил студент группы  
ИВБО-12-23

Туктаров Т.А.

Принял преподаватель кафедры ВТ

Дуксина И.И.

Лабораторная работа выполнена

«\_\_»\_\_\_\_\_2025 г.

«Зачтено»

«\_\_»\_\_\_\_\_2025 г.

Москва 2025

## **АННОТАЦИЯ**

Данная работа включает в себя 8 рисунков и 9 листингов. Количество страниц в работе — 24.

# СОДЕРЖАНИЕ

Введение .....	4
1 Создание необходимых модулей на Verilog HDL .....	5
1.1 Описание дополнительных модулей .....	5
1.2 Создание модуля управления семисегментными индикаторами .....	12
1.3 Создание модуля верхнего уровня .....	15
2 Создание тестового модуля и его верификация .....	18
3 Создание файла проектных ограничений и верификация на плате .....	23
ЗАКЛЮЧЕНИЕ .....	24
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ .....	30

## ВВЕДЕНИЕ

В данной лабораторной работе рассматриваются вопросы индикации при помощи семисегментных индикаторов, объединённых в дисплей в составе отладочной платы серии Xilinx Nexys[1-2]. Разрабатываемое устройство представляет собой устройство хранения истории ввода[3]. Ввод очередного значения осуществляется при помощи движковых переключателей платы Xilinx Nexys. Для подтверждения ввода используется кнопка BTN\_C платы Xilinx Nexys. Хранимая история ввода должна отображаться при помощи семисегментных индикаторов в составе платы Xilinx Nexys. Последнее введённое значение должно отображаться на крайнем правом индикаторе правого дисплея. Далее, справа налево должны располагаться ранее введённые значения в порядке их прихода (чем раньше пришло значение, тем левее оно располагается). Количество хранимых значений прямо пропорционально количеству индикаторов. Для отладочной платы Nexys A7 число хранимых значений равно 8.

# 1 СОЗДАНИЕ НЕОБХОДИМЫХ МОДУЛЕЙ НА VERILOG HDL

## 1.1 Описание дополнительных модулей

В работе будут использованы модули счётчика, делителя частоты, синхронизатора и фильтра дребезга контактов. Рассмотрим каждый модуль.

Начнём с модуля счётчика. Название модуля – «count». Модуль имеет следующие параметры: «MODULE», отвечающий за модуль счёта, по умолчанию равный 2; «STEP», отвечающий за шаг счёта, по умолчанию равный 1. Модуль обладает следующими входными портами: «clk» - синхросигнал, «reset» - сброс, «enable» – разрешающий сигнал, «direction» - направление счёта; выходной регистр «cnt». С помощью оператора «initial» происходит инициализация значения «cnt» значением 0. Далее в блоке «always» при поступлении переднего фронта синхросигнала «clk» происходит изменение выходного регистра «cnt». При единице на «reset» значение счётчика «cnt» будет сброшено в ноль, иначе при единице на «enable» будет изменено значение «cnt» в соответствии с направлением, модулем и шагом счёта.

Реализация модуля представлена в Листинге 1.1. RTL-схема представлена на Рисунке 1.1.

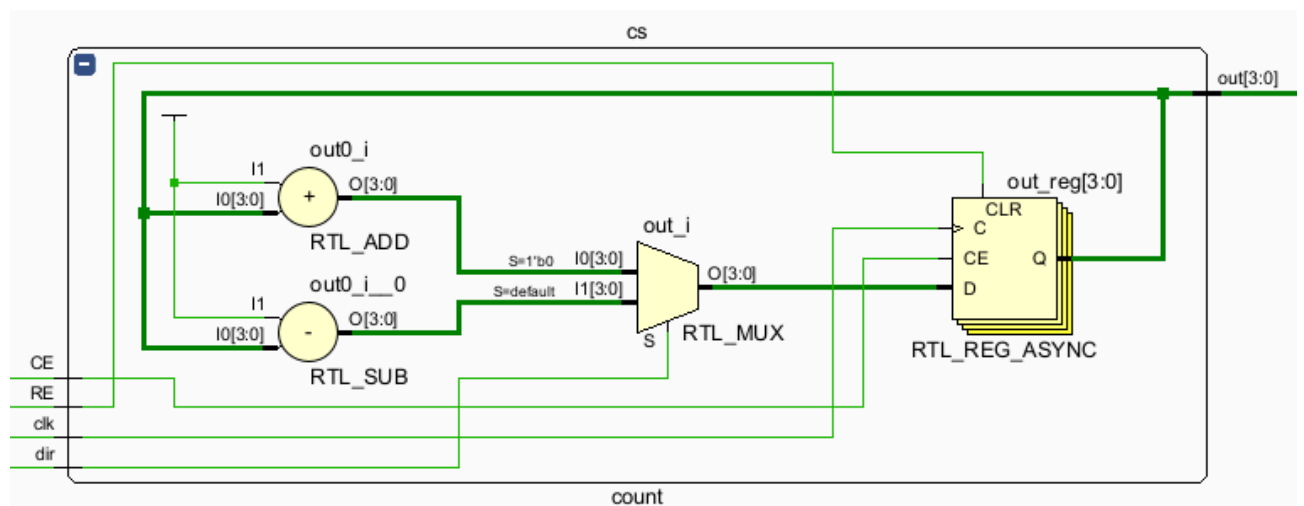
*Листинг 1.1 – Реализация модуля параметризованного универсального реверсивного счётчика*

```

`timescale 1ns / 1ps
module count# (step = 1, mod = 16) (
    input clk,
    input dir,
    input RE, //Reset
    input CE, //Clock enable
    output reg [$clog2(mod) -1:0] out
);
    initial out = 0;

    always@(posedge clk or posedge RE)
    begin
        if (RE)
            out <= 0; // Reset
        else if (CE) begin
            if (dir == 0)
                out <= (out + step) % mod;
            else
                out <= (mod + out - step) % mod;
        end
    end
end
endmodule

```



**Рисунок 1.1 - RTL-схема модуля универсального параметризованного реверсивного счётчика**

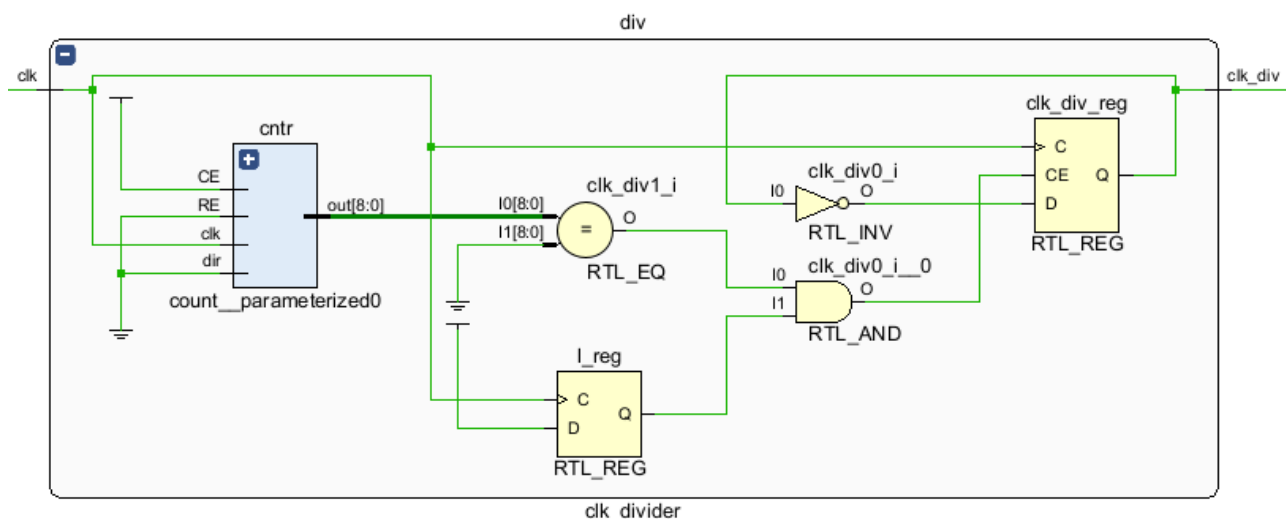
Следующий модуль делителя частоты. Название модуля – «clk\_divider». Модуль имеет параметр «div», отвечающий за значение, на которое будет изменяться частота синхросигнала, по умолчанию равный 2. Модуль обладает входным портом «clk» - синхросигнал; и выходной регистр «clk\_div».

Далее создаётся экземпляр счётчика «cnt» с параметрами «STEP» со значением единицы и «MODULE» со значением «DIV/2». После происходит присваивание портам соответствующих значений. На вход «reset» подаётся 0, «direction» подаётся 0, так как направлению счёта у нас постоянное на суммирование, на порт «enable» подаётся 1. С помощью оператора «initial» происходит инициализация значения «clk\_div» значением 0. Далее с помощью блока «always» отслеживается поступление переднего фронта «clk». При равенстве значение «cnt», полученного с выхода счётчика, значению 0, будет произведена инвертация значения «clk\_div».

Реализация модуля представлена в Листинге 1.2. RTL-схема представлена на Рисунке 1.2.

*Листинг 1.2 – Реализация модуля параметризованного делителя частоты*

```
`timescale 1ns / 1ps
module clk_divider#(div = 2) (
    input clk,
    output reg clk_div
);
    reg l;
    wire [8:0] out;
    count #(.step(1), .mod(div/2)) cnter(
        .clk(clk),
        .RE(1'b0),
        .CE(1'b1),
        .dir(1'b0),
        .out(out)
    );
    initial clk_div = 0;
    always@(posedge clk)begin
        if (out ==0 && l==1)
            clk_div = ~clk_div;
            l<=1;
    end
endmodule
```



**Рисунок 1.2 - RTL-схема модуля параметризованного делителя частоты**

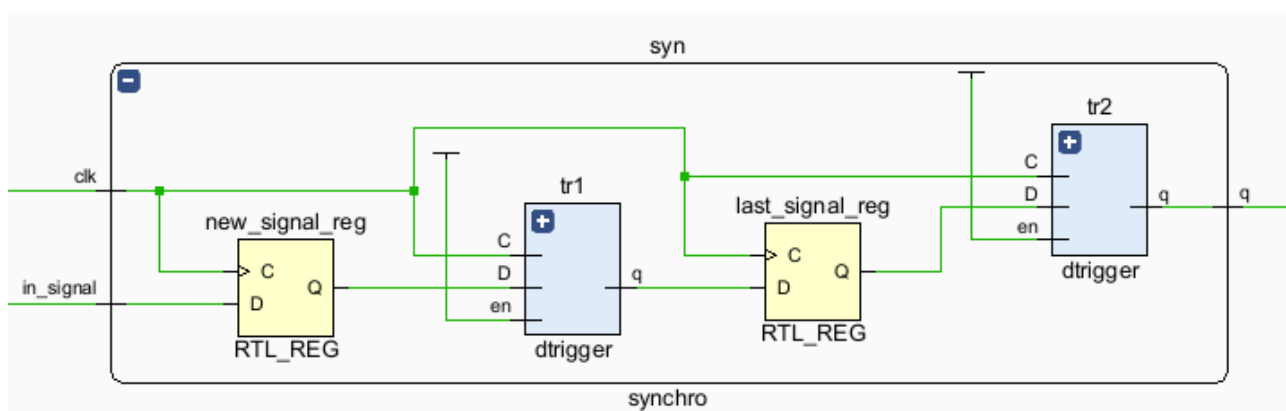
Следующий модуль синхронизатора. Название модуля – «synchro». Модуль обладает следующими входными портами: «clk» - синхросигнал, «in» - сигнал с кнопки; и выходом «out». Объявляются два регистра – «a» и «b». В блоке «always», работающем по переднему фронту синхросигнала, регистру «b» присваивается значение на регистре «a», а регистру «a» – значение на входном порте «in». Далее с помощью оператора непрерывного присваивания «assign» выходному порту присваивается значение на регистре «b».

Реализация модуля представлена в Листинге 1.3. RTL-схема представлена на Рисунке 1.3.



*Листинг 1.3 – Реализация модуля синхронизатора*

```
`timescale 1ns / 1ps
module synchro(
    input in_signal,
    input clk,
    output q,
    output notq
);
    reg new_signal = 1'bx;
    reg last_signal = 1'bx;
    wire lq;
    dtrigger tr1(
        .C(clk),
        .D(new_signal),
        .en(1'b1),
        .q(lq)
    );
    dtrigger tr2(
        .C(clk),
        .D(last_signal),
        .en(1'b1),
        .q(q)
    );
    always @(posedge clk) begin
        new_signal=in_signal;
        last_signal=lq;
    end
endmodule
```



**Рисунок 1.3 - RTL-схема модуля синхронизатора**

Следующий модуль фильтра дребезга контактов. Название модуля – «filtercon». Модуль обладает параметром «MODULE», по умолчанию равный 8. Модуль обладает следующими входными портами: «clk» - синхросигнал, «in\_signal» - сигнал с кнопки; и выходными регистрами: «out\_signal» - значение кнопки без дребезга, «out\_signal\_enable» - сигнал о нажатии кнопки.

Создается экземпляр модуля «synchronizer» с названием «sync». К порту «in» подключается цепь «in\_signal», к порту «clk» – синхросигнал «clk», к выходному порту «out» подключается цепь «sync\_signal». Далее создается экземпляр модуля «counter» с названием «cntr», в параметр «MODULE» подается

«MODULE» фильтра, а в параметр «STEP» подается 1. К порту «clk» подключается синхросигнал «clk», в порт «reset» подается результат выражения «sync\_signal~^out\_signal», к порту «enable» подключается параметр «CLOCK\_ENABLE», в порт «direction» подается 0, а к выходному порту «cnt» подключается цепь «counter\_res».

В блоке «always», работающему по переднему фронту, регистру «out\_signal» присваивается значение на цепи «sync\_signal», если выражение «&(counter\_res) & CLOCK\_ENABLE» равняется 1, также «out\_signal\_enable» присваивается результат выражения «&(counter\_res) & sync\_signal & CLOCK\_ENABLE».

Реализация модуля представлена в Листинге 1.4. RTL-схема представлена на Рисунке 1.4.

*Листинг 1.4 – Реализация модуля устранения дребезга контактов*

```
`timescale 1ns / 1ps
module filtercon #(mode = 2) (
    input in_signal,
    input clock_enable,
    input clk,
    output wire out_signal,
    output out_signal_enable,
    output wire [1:0] q_count
);

    wire out_sync;
    wire out_xnor;
    wire out_first_and;
    wire out_second_and;
    wire out_third_and;
    synchro syn(
        .in_signal(in_signal),
        .clk(clk),
        .q(out_sync)
    );

    count cs(
        .clk(clk),
        .RE(out_sync~^out_signal),
        .dir(0),
        .CE(clock_enable),
        .out(q_count)
    );

    dtrigger dt1(
        .C(clk),
        .D(out_sync),
        .en(out_second_and),
        .q(out_signal)
    );
    dtrigger dt2(
        .C(clk),
        .D(out_third_and),
        .en(1'b1),
        .q(out_signal_enable)
    );

    assign out_first_and = &q_count;
    assign out_second_and = out_first_and & clock_enable;
    assign out_third_and = out_second_and & out_sync;

endmoduleend

endmodule
```



Создаётся модуль счётчика cntр с параметрами «MODULE», равным 8, и «STEP», равным 1. К порту «clk» подключен «clk», к порту «reset» - подключен «RESET», на порт «enable» подана 1, на порт «direction» подан 0, к порту «cnt» подключена трёхбитная шина «counter\_res»

В блоке «always», работающему по переднему фронту «clk», «SEG», присваивается значение «8'b11111111», если «RESET» равно 1, иначе в блоке «case» по значению «NUMBER\_SPLITTER[counter\_res]» выбирается значение для выходного порта «SEG», с помощью логического сдвига на значение «counter\_res» выбирается значение для регистра «AN\_REG».

Реализация модуля представлена в Листинге 1.5. RTL-схема представлена на Рисунке 1.5.

*Листинг 1.5 – Реализация модуля управления семисегментными индикаторами*

```
module SevenSegmentLED(
    input [7:0] AN_MASK,
    input [31:0] NUMBER,
    input clk,
    input RESET,
    output [7:0] AN,
    output reg[7:0] SEG
);

wire[2:0] counter_res;

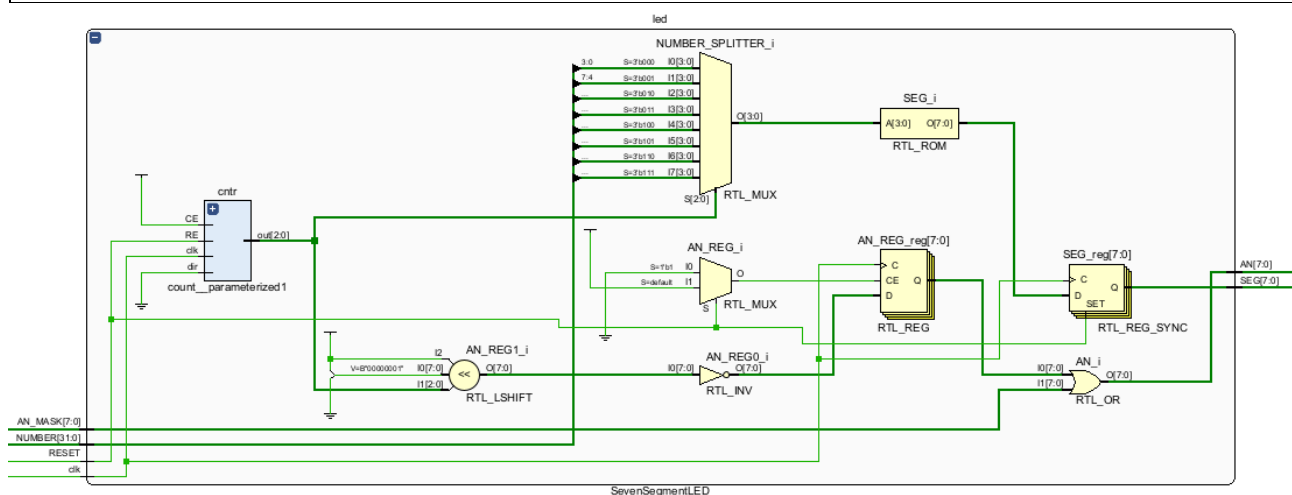
count #(.mod(8), .step(1)) cntр(
    .clk(clk),
    .RE(RESET),
    .CE(1'b1),
    .dir(1'b0),
    .out(counter_res)
);

reg [7:0] AN_REG = 0;
assign AN = AN_REG | AN_MASK;
wire [3:0] NUMBER_SPLITTER[0:7];
genvar i;
generate
    for (i = 0; i < 8; i = i + 1)
    begin
        assign NUMBER_SPLITTER[i] = NUMBER[((i+1)*4-1)-:4];
    end
endgenerate
always @(posedge clk)
begin
    if (RESET)
        SEG <= 8'b11111111;
    else
    begin
        case (NUMBER_SPLITTER[counter_res])
            4'h0: SEG <= 8'b11000000;
            4'h1: SEG <= 8'b11111001;
```

```

        4'h2: SEG <= 8'b10100100;
        4'h3: SEG <= 8'b10110000;
        4'h4: SEG <= 8'b10011001;
        4'h5: SEG <= 8'b10010010;
        4'h6: SEG <= 8'b10000010;
        4'h7: SEG <= 8'b11111000;
        4'h8: SEG <= 8'b10000000;
        4'h9: SEG <= 8'b10010000;
        4'ha: SEG <= 8'b10001000;
        4'hb: SEG <= 8'b10000011;
        4'hc: SEG <= 8'b11000110;
        4'hd: SEG <= 8'b10100001;
        4'he: SEG <= 8'b10000110;
        4'hf: SEG <= 8'b10001110;
        default: SEG <= 8'b11111111;
    endcase
    AN_REG = ~(8'b1 << counter_res);
end
end
endmodule

```



**Рисунок 1.5 - RTL-схема модуля управления семисегментными индикаторами**

Для реализации данных модулей использовался отдельный модуль, реализующий D–триггер, представленный в Листинге 1.6.

Листинг 1.6 – Реализация модуля D-триггера

```
timescale 1ns / 1ps
module dtrigger(
    input wire C,
    input wire D,
    input wire en,
    output reg q);
    initial begin
        q<=0;
    end

    always @(posedge C) begin
        if(en) begin
            q <= D;
        end
    end
endmodule
```

### 1.3 Создание модуля верхнего уровня

Модуль верхнего уровня имеет название «main». Он обладает следующими портами: четырехбитный входной порт «SWITHCES» - цифра, которую необходимо отобразить на дисплее, входной порт «button\_in» - кнопка для разрешения записи, синхросигнал «clk», входной порт «button\_reset\_in» для сброса значений маски и регистра, выходной порт «an» – шина разрешающих входов анодов для всех индикаторов, «seg» - шина катодов для одного индикатора.

Объявляется регистр «AN\_MASK» (маска, используемая для отключения части семисегментных индикаторов) и инициализируется значением «8'b11111111». Объявляется регистр «NUMBER» (используется для хранения введенных значений) и инициализируется значением 0. Объявляется цепь «clk\_div» для вывода результата работы делителя частоты. Объявляются 4 цепи «button\_signal», «button\_signal\_en», «reset\_signal\_en» и «reset\_signal» для вывода результатов работы первого и второго фильтра дребезга контактов соответственно.

Создается экземпляр модуля «debouncer» с названием «dbnc», в единственный параметр которого передается 128. К портам «clk», «in\_signal», «CLOCK\_ENABLE», «out\_signal» и «out\_signal\_enable» подключаются «clk», «button\_in», «l'b1», «button\_signal» и «button\_signal\_en» соответственно. Также создается экземпляр модуля «debouncer» с названием «dbnc\_reset», в единственный параметр которого передается 128. К портам «clk», «in\_signal», «CLOCK\_ENABLE», «out\_signal» и «out\_signal\_enable» подключаются «clk», «button\_reset\_in», «l'b1», «reset\_signal» и «reset\_signal\_en» соответственно.

Создается экземпляр модуля «clk\_div» с названием «div», в единственный параметр которого передается 1024. К портам «clk» и «clk\_div» подключаются «clk» и «clk\_div» соответственно.

Создается экземпляр модуля «SevenSegmentLED» с названием «led». К портам «AN\_MASK», «NUMBER», «RESET», «clk», «AN» и «SEG»

подключаются «mask», «number», «reset\_signal», «clk\_div», «AN» и «SEG» соответственно.

В блоке «always» обновляется последняя цифра регистра, регистрам «AN\_MASK» и «NUMBER» присваиваются их начальные значения, если на цепи «reset\_signal» значение 1. Если на цепи «button\_signal\_en» значение 1, регистр «AN\_MASK» сдвигается на 1 бит влево, а «NUMBER» сдвигается на 4 бита влево. Блок работает по переднему фронту синхросигнала.

Код модуля верхнего уровня представлен в Листинге 1.6, а его RTL-схема представлена на Рисунке 1.6.

*Листинг 1.6 – Реализация модуля контроллера*

```
``timescale 1ns / 1ps
module main(
    input [3:0] SWITCHES,
    input button_in, clk, button_reset_in,
    output [7:0] AN,
    output [6:0] SEG
);
    wire button_signal, button_signal_en, reset_signal_en, reset_signal;
    reg[7:0] AN_MASK = 8'b11111111;
    reg [31:0] NUMBER;
    initial begin
        NUMBER <= 0;
    end
    filtercon #(128) dbnc(
        .clk(clk),
        .in_signal(button_in),
        .clock_enable(1'b1),
        .out_signal(button_signal),
        .out_signal_enable(button_signal_en)
    );

    filtercon #(128) dbnc_reset(
        .clk(clk),
        .in_signal(button_reset_in),
        .clock_enable(1'b1),
        .out_signal(reset_signal),
        .out_signal_enable(reset_signal_en)
    );

    clk_divider #(1024) div(
        .clk(clk),
        .clk_div(clk_div)
    );

    SevenSegmentLED led(
        .AN_MASK(AN_MASK),
        .NUMBER(NUMBER),
        .clk(clk_div),
        .RESET(reset_signal),
        .AN(AN),
        .SEG(SEG)
    );
endmodule
```





## 2 СОЗДАНИЕ ТЕСТОВОГО МОДУЛЯ И ЕГО ВЕРИФИКАЦИЯ

Название тестового модуля – «testbench». Объявляется четырехбитный регистр «SWITCHES» (ввод цифры с помощью движковых переключателей) и инициализируется значением 0. Объявляются две восьмибитные цепи «SEG» и «AN». Также объявляются регистры «clk», «button» и «button\_reset» и они инициализируются нулем.

Создается экземпляр модуля верхнего уровня «Controller» с названием «cntlr». К портам «SWITCHES», «button\_in», «clk», «button\_reset\_in», «AN» и «SEG» подключаются «SWITCHES», «button», «clk», «button\_reset», «AN» и «SEG» соответственно.

В блоке «always» каждые 5 наносекунд регистр «clk» меняет свое значение на противоположное.

В блоке «initial» симулируется работа с платой: два нажатия на кнопку, чтобы разрешить записи двух цифр. В конце работы происходит нажатие на кнопку, отвечающую за восстановление изначальных значений. Чтобы симитировать дребезг контактов, используется генерация псевдослучайных чисел.

Реализация тестового модуля представлена в Листинге 2.1. На Рисунке 2.1 представлена верификация тестового модуля.

*Листинг 2.1 – Реализация тестового модуля*

```
`timescale 1ns / 1ps
module testbench;
    reg[3:0] SWITCHES = 0;
    reg clk = 0;
    reg button = 0;
    reg button_reset = 0;
    wire[7:0] AN;
    wire[6:0] SEG;
    wire[31:0] NUMBER;
    wire button_signal;
    wire button_signal_en;
    wire reset_signal_en;
    wire reset_signal;
    main cntlr(
        .SWITCHES(SWITCHES),
        .button_in(button),
```

*Продолжение Листинга 2.1*

```
.clk(clk),
.button_reset_in(button_reset),
.AN(AN),
.NUMBER(NUMBER),
.SEG(SEG),
.button_signal(button_signal),
.button_signal_en(button_signal_en),
.reset_signal_en(reset_signal_en),
.reset_signal(reset_signal)
);
always #5 clk = ~clk;
initial
begin
    #4000
    $srandom(35000);
    SWITCHES = 4'b0000;
    repeat($urandom_range(50,0))
    begin
        button = $random;
        #3;
    end
    button = 1;
    #1000;

    repeat($urandom_range(50,0))
    begin
        button = $random;
        #3;
    end
    button = 0;
    #8000;

    SWITCHES = 4'b0001;
    repeat($urandom_range(50,0))
    begin
        button = $random;
        #3;
    end
    button = 1;
    #1000;

    repeat($urandom_range(50,0))
    begin
        button = $random;
        #3;
    end
    button = 0;
    #8000;

    #4000
    $srandom(35000);
    SWITCHES = 4'b0010;
    repeat($urandom_range(50,0))
    begin
        button = $random;
        #3;
    end
    button = 1;
    #1000;

    repeat($urandom_range(50,0))
    begin
```

*Продолжение листинга 2.1*

```
        button = $random;
        #3;
    end
    button = 0;
    #8000;

    #4000
    $srandom(35000);
    SWITCHES = 4'b0011;
    repeat($urandom_range(50,0))
    begin
        button = $random;
        #3;
    end
    button = 1;
    #1000;

    repeat($urandom_range(50,0))
    begin
        button = $random;
        #3;
    end
    button = 0;
    #8000;

    #4000
    $srandom(35000);
    SWITCHES = 4'b0100;
    repeat($urandom_range(50,0))
    begin
        button = $random;
        #3;
    end
    button = 1;
    #1000;

    repeat($urandom_range(50,0))
    begin
        button = $random;
        #3;
    end
    button = 0;
    #8000;

    #4000
    $srandom(35000);
    SWITCHES = 4'b0101;
    repeat($urandom_range(50,0))
    begin
        button = $random;
        #3;
    end
    button = 1;
    #1000;

    repeat($urandom_range(50,0))
    begin
        button = $random;
        #3;
    end
    button = 0;
    #8000;
```

```
#4000
$srandom(35000);
SWITCHES = 4'b0110;
repeat($urandom_range(50,0))
begin
    button = $random;
    #3;
end
button = 1;
#1000;

repeat($urandom_range(50,0))
begin
    button = $random;
    #3;
end
button = 0;
#8000;

#4000
$srandom(35000);
SWITCHES = 4'b0111;
repeat($urandom_range(50,0))
begin
    button = $random;
    #3;
end
button = 1;
#1000;

repeat($urandom_range(50,0))
begin
    button = $random;
    #3;
end
button = 0;
#8000;

#4000
$srandom(35000);
SWITCHES = 4'b1000;
repeat($urandom_range(50,0))
begin
    button = $random;
    #3;
end
button = 1;
#1000;

repeat($urandom_range(50,0))
begin
    button = $random;
    #3;
end
button = 0;
#8000;

#4000
$srandom(35000);
SWITCHES = 4'b1000;
repeat($urandom_range(50,0))
```



# 3 СОЗДАНИЕ                      ФАЙЛА                      ПРОЕКТНЫХ ОГРАНИЧЕНИЙ И ВЕРИФИКАЦИЯ НА ПЛАТЕ

Содержание файла проектных ограничений представлено в Листинге 3.1.

*Листинг 3.1 – Реализация проектных ограничений*

```
create_clock -period 10.000 -name sys_clk -waveform {0.000 5.000} -add
[get_ports clk]

set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property PACKAGE_PIN E3 [get_ports clk]

#кнопки
set_property PACKAGE_PIN N17 [get_ports button_in]
set_property IOSTANDARD LVCMOS33 [get_ports button_in]
set_property PACKAGE_PIN M18 [get_ports button_reset_in]
set_property IOSTANDARD LVCMOS33 [get_ports button_reset_in]

#свичи
set_property IOSTANDARD LVCMOS33 [get_ports {SWITCHES[0]}]
set_property PACKAGE_PIN J15 [get_ports {SWITCHES[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SWITCHES[1]}]
set_property PACKAGE_PIN L16 [get_ports {SWITCHES[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SWITCHES[2]}]
set_property PACKAGE_PIN M13 [get_ports {SWITCHES[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SWITCHES[3]}]
set_property PACKAGE_PIN R15 [get_ports {SWITCHES[3]}]

#аноды
set_property IOSTANDARD LVCMOS33 [get_ports {AN[0]}]
set_property PACKAGE_PIN J17 [get_ports {AN[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[1]}]
set_property PACKAGE_PIN J18 [get_ports {AN[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[2]}]
set_property PACKAGE_PIN T9 [get_ports {AN[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[3]}]
set_property PACKAGE_PIN J14 [get_ports {AN[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[4]}]
set_property PACKAGE_PIN P14 [get_ports {AN[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[5]}]
set_property PACKAGE_PIN T14 [get_ports {AN[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[6]}]
set_property PACKAGE_PIN K2 [get_ports {AN[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[7]}]
set_property PACKAGE_PIN U13 [get_ports {AN[7]}]

#катоды
set_property IOSTANDARD LVCMOS33 [get_ports {SEG[0]}]
set_property PACKAGE_PIN T10 [get_ports {SEG[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEG[1]}]
set_property PACKAGE_PIN R10 [get_ports {SEG[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEG[2]}]
set_property PACKAGE_PIN K16 [get_ports {SEG[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEG[3]}]
set_property PACKAGE_PIN K13 [get_ports {SEG[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEG[4]}]
set_property PACKAGE_PIN P15 [get_ports {SEG[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEG[5]}]
set_property PACKAGE_PIN T11 [get_ports {SEG[5]}]
```

### Продолжение Листинга 3.1

```
set_property IOSTANDARD LVCMOS33 [get_ports {SEG[6]}]  
set_property PACKAGE_PIN L18 [get_ports {SEG[6]}]
```

Файл с расширением .bit сгенерирован и загружен на плату. Результат представлен на Рисунках 3.1 – 3.5.



Рисунок 3.1 - .bit файл загружен на плату.



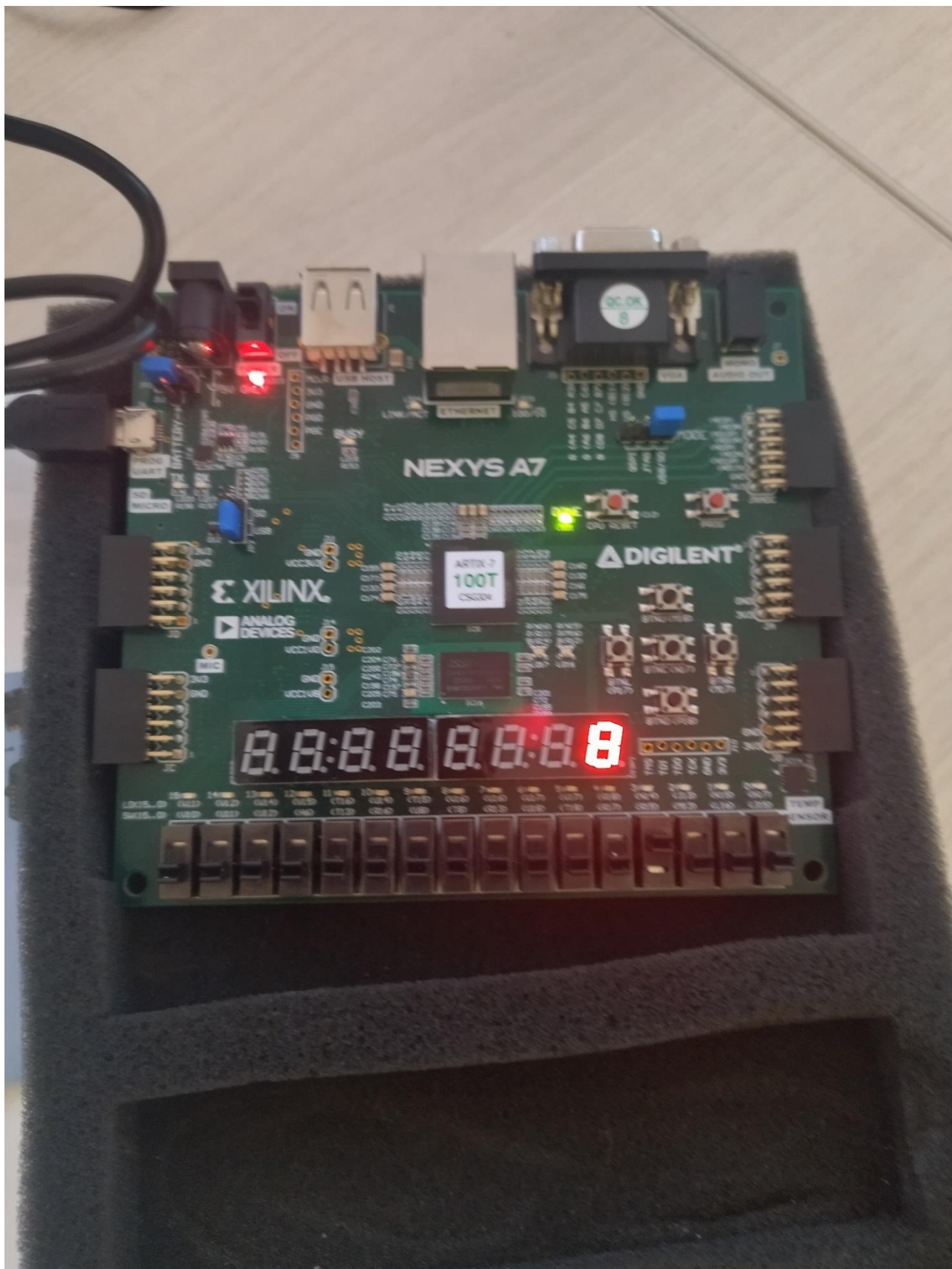


Рисунок 3.2 – Запись одной цифры



Рисунок 3.3 – Полное заполнение на плате





Рисунок 3.4 – Сдвиг значений на плате

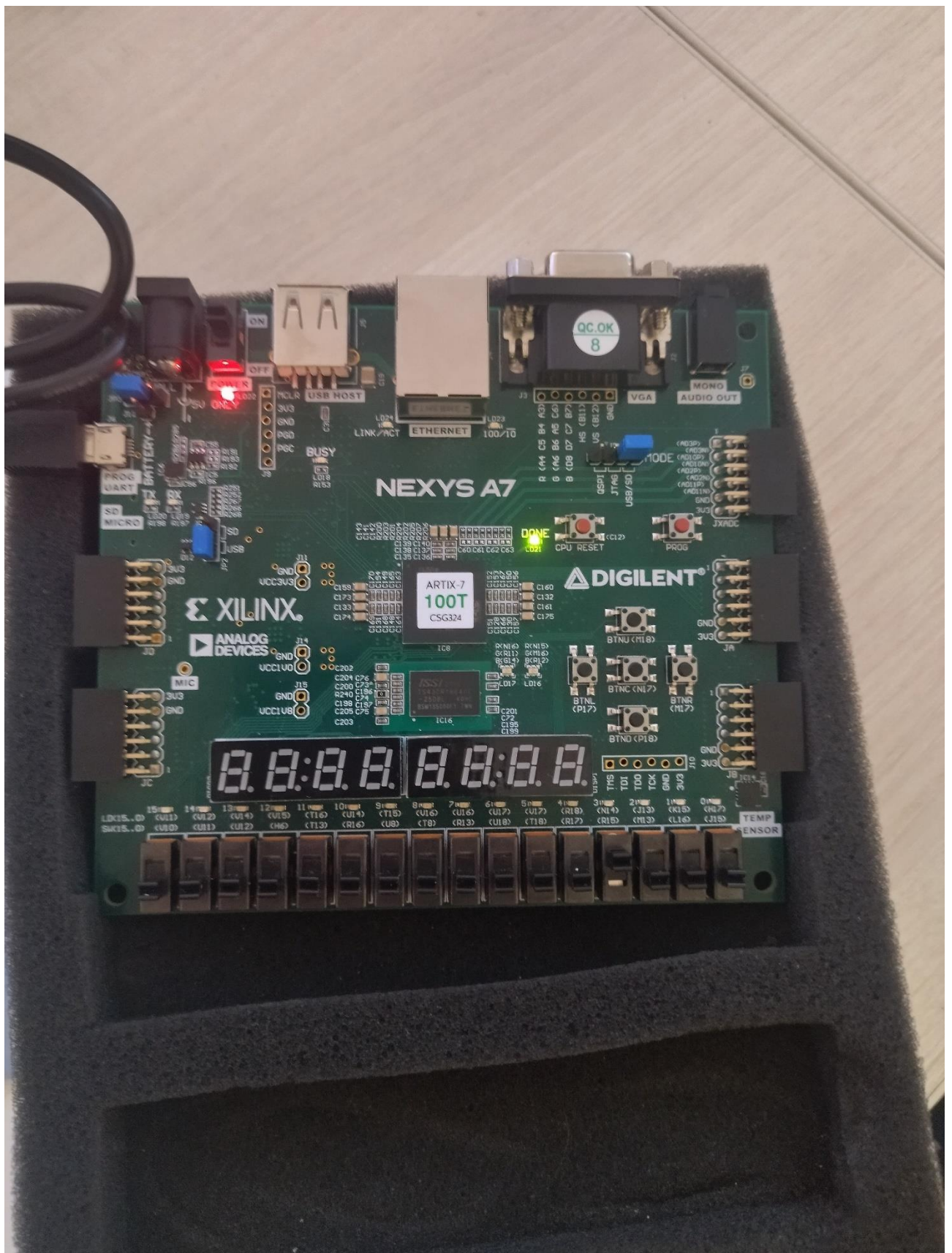


Рисунок 3.5 – Сброс значений.

## **ЗАКЛЮЧЕНИЕ**

В результате выполнения лабораторной работы приобретён навык построения управляющего устройства для визуального отображения информации при помощи набора семисегментных индикаторов, изучены основные особенности считывания сигнала с физического устройства ввода — кнопки.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Методические указания по ЛР № 1 — URL: <https://online-edu.mirea.ru/mod/resource/view.php?id=413209> (Дата обращения: 18.02.2024).
2. Смирнов С.С. Информатика [Электронный ресурс]: Методические указания по выполнению практических и лабораторных работ / С.С. Смирнов — М., МИРЭА — Российский технологический университет, 2018. — 1 электрон. опт. диск (CD-ROM).
3. Тарасов И.Е. ПЛИС Xilinx. Языки описания аппаратуры VHDL и Verilog, САПР, приемы проектирования. — М.: Горячая линия — Телеком, 2021. — 538 с.: ил.
4. Рабан, Жан.М., Чандракасан, А., Николич, Б. Цифровые интегральные схемы. Методология проектирования. 2-е изд.: Пер. с англ. — М.: ООО «И.Д. Вильямс», 2016. — 912 с.: ил. — Параллит. англ. ISBN 978-5-8459- 1116-2 (рус.).
5. Шафер Д., Фатрелл Р., Шафер Л. Управление программными проектами: достижение оптимального качества при минимуме затрат: Пер. с англ. — М.: Издательский дом «Вильямс», 2004. — 1136 с.: ил. — Парал.тит.англ.