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 № 1

«Управление семисегментными индикаторами»

по дисциплине

«Схемотехника устройств компьютерных систем»

Выполнил студент группы ИВБО-08-22 — Стецюк В.В.

Принял преподаватель кафедры ВТ — Дуксина И.И.

Лабораторная работа выполнена «__»_____2024 г.

«Зачтено» «__»_____2024 г.

Москва 2024

АННОТАЦИЯ

Данная работа включает в себя 8 рисунков и 8 листингов. Количество страниц в работе — 21.

СОДЕРЖАНИЕ

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 буду использованы модули счётчика, делителя частоты, синхронизатора и фильтра дребезга контактов. Рассмотрим каждый модуль.

Начнём с модуля счётчика. Название модуля – «counter». 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 параметризированного универсального реверсивного счётчика

```
module counter #(STEP = 1, MODULE = 2)(
    input clk, reset, enable, direction,
    output reg[$clog2(MODULE)-1:0] cnt
);

initial cnt = 0;

always@(posedge clk)
begin
    if (reset)
        cnt <= 0;
    else if (enable)
        cnt <= direction ? (MODULE + cnt - STEP) % MODULE : (cnt + STEP) %
MODULE;
end

endmodule
```

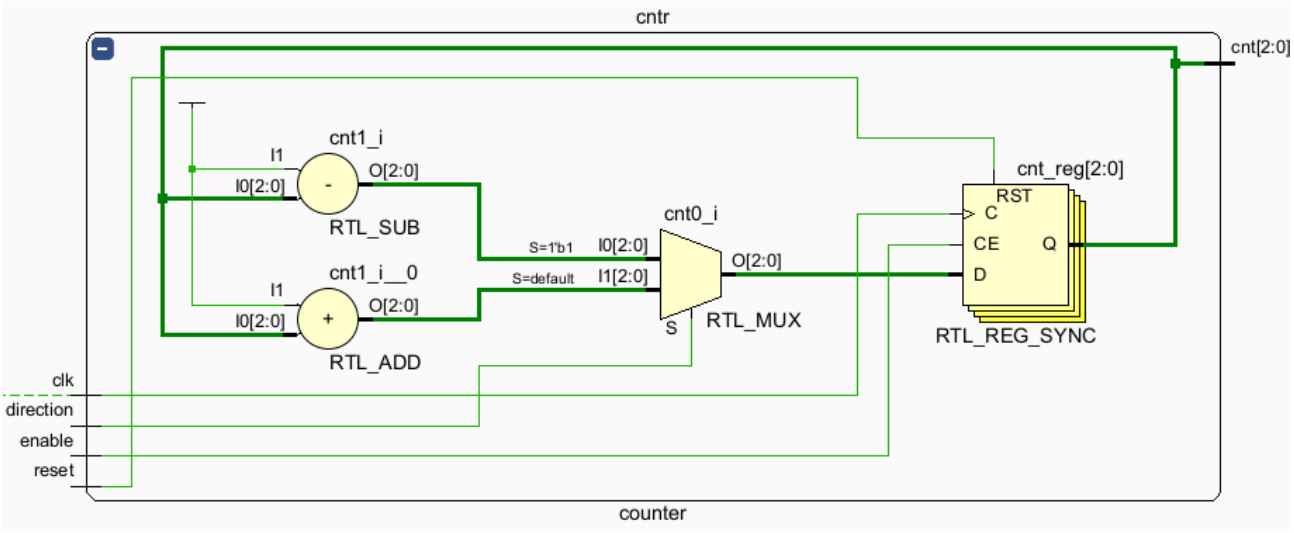

Рисунок 1.1 - RTL-схема модуля универсального параметризированного реверсивного счётчика

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 «cntr» 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 параметризированного делителя частоты

```
`timescale 1ns / 1ps

module clk_divider #(DIV = 2) (
    input clk,
    output reg clk_div);

wire [$clog2(DIV/2)-1:0] cnt;
```

Продолжение Листинга 1.2

```
counter #(.STEP(1), .MODULE(DIV/2)) cntr(
    .clk(clk),
    .reset(1'b0),
    .enable(1'b1),
    .direction(1'b0),
    .cnt(cnt)
);

initial clk_div = 0;

always@(posedge clk)
    if (cnt == 0)
        clk_div = ~clk_div;

endmodule
```

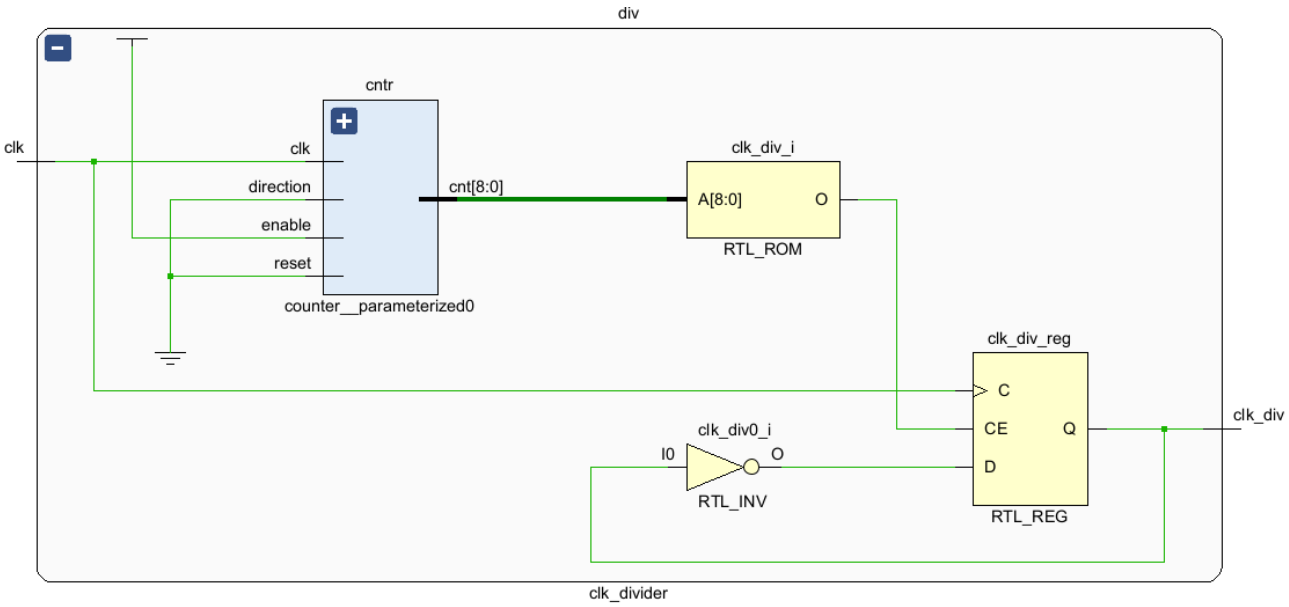

Рисунок 1.2 - RTL-схема модуля параметризированного делителя частоты

Следующий модуль синхронизатора. Название модуля – «synchronizer». 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

```
module synchronizer(
    input in, clk,
    output out);

reg a, b;

always@(posedge clk)
begin
    b <= a;
    a <= in;
end

assign out = b;

endmodule
```

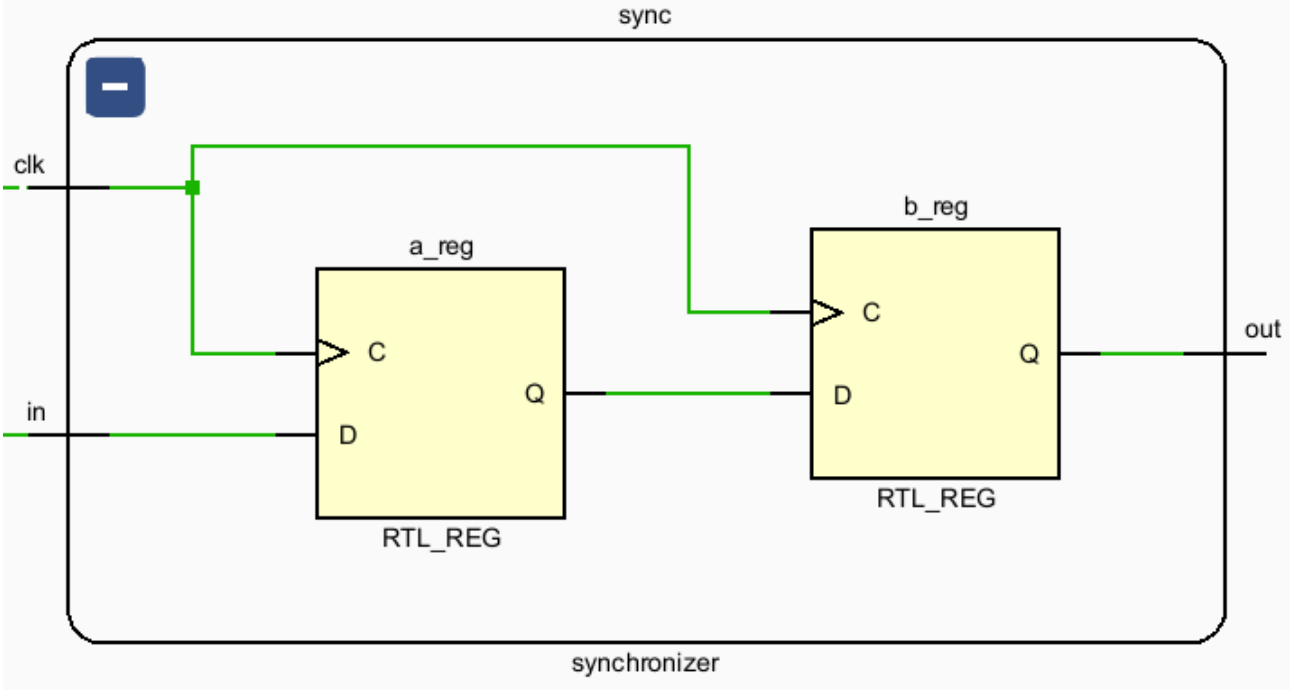

Рисунок 1.3 - RTL-схема модуля синхронизатора

Следующий модуль фильтра дребезга контактов. Название модуля – «debouncer». 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

```
module debouncer #(MODULE = 8) (
    input clk, in_signal, CLOCK_ENABLE,
    output reg out_signal, reg out_signal_enable
 );

wire sync_signal;
wire [$clog2(MODULE)-1:0] counter_res;

synchronizer sync(.in(in_signal), .clk(clk), .out(sync_signal));

counter #(.MODULE(MODULE), .STEP(1)) cntr(
    .clk(clk),
    .reset(sync_signal~^out_signal),
    .enable(CLOCK_ENABLE),
    .direction(1'b0),
    .cnt(counter_res)
);

always@(posedge clk)
begin
    if (&(counter_res) & CLOCK_ENABLE)
        out_signal <= sync_signal;
    out_signal_enable <= &(counter_res) & sync_signal & CLOCK_ENABLE;
end


endmodule
```

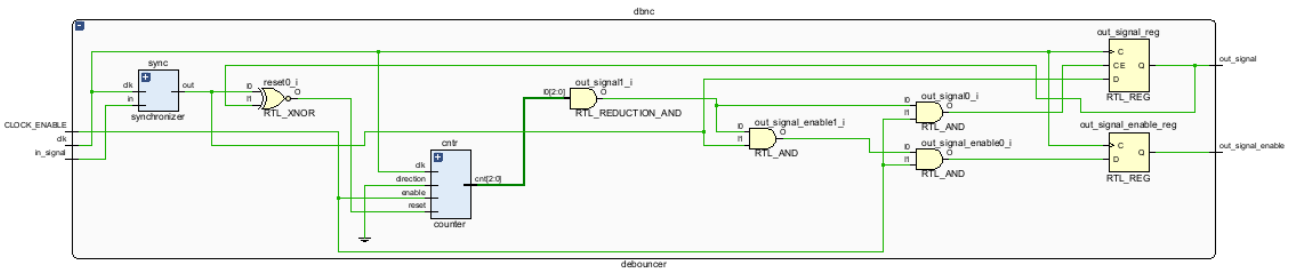

Рисунок 1.4 - RTL-схема модуля синхронизатора

1.2 Создание модуля управления семисегментными индикаторами

Название модуля – «SevenSegmentLED». Модуль имеет следующие порты: входная восьмибитная шина «AN_MASK» – маска для отключения части семисегментных индикаторов, входная 32-битная шина «NUMBER» - значение, выводимое на дисплей, входной порт «RESET» - сброс, входной порт «clk» – синхросигнал, выходная восьмибитная шина «AN» – значения анодов для всех индикаторов, выходной восьмибитный регистр «SEG» - значения катодов.

Объявляется и инициализируется нулем восьмибитный регистр «AN_REG» для управления анодами семисегментного индикатора. Объявляется трехбитный регистр «counter_res» для определения позиции горящего символа на дисплее. Объявляется четырехбитная шина «NUMBER_SPLITTER» из восьмибитных проводов. Далее с помощью ключевого слова «genvar» объявляется переменная «i», которая используется только в блоке «generate». В блоке «generate» находится цикл «for», с помощью которого шина «NUMBER» разделяется на 8 частей, объединенных в шине «NUMBER_SPLITTER».

С помощью оператора непрерывного присваивания «assign» к выходному порту «AN» подключается результат выражения «AN_REG | AN_MASK».

Создаётся модуль счётчика cntr 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

```
`timescale 1ns / 1ps

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;

counter #(.MODULE(8), .STEP(1)) cntr(
    .clk(clk),
    .reset(RESET),
    .enable(1'b1),
    .direction(1'b0),
    .cnt(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
```

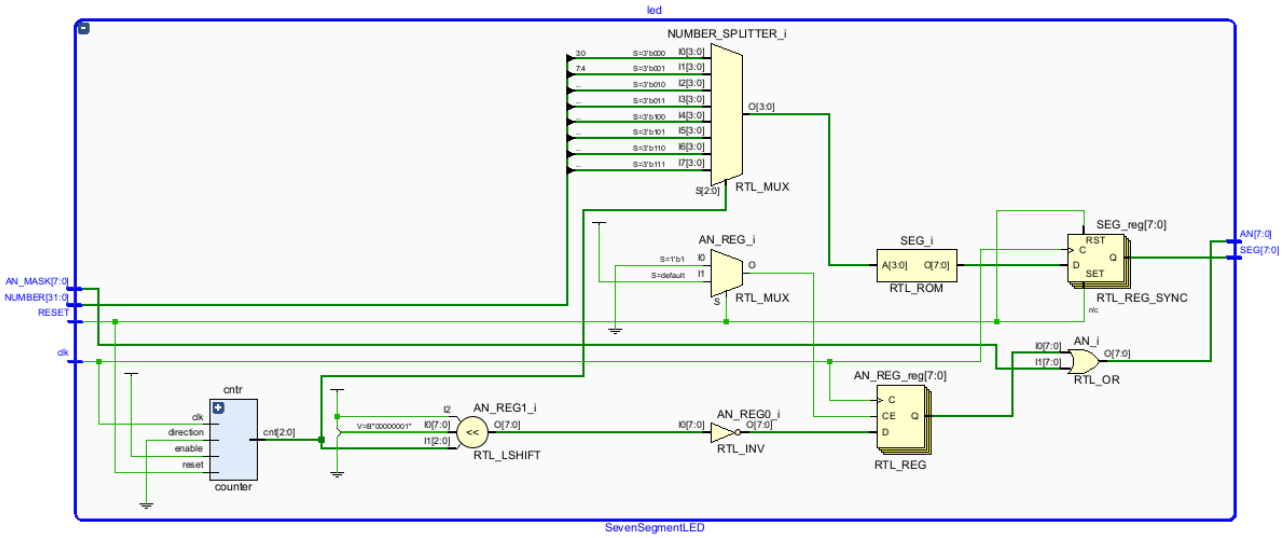

Рисунок 1.5 - RTL-схема модуля управления семисегментными индикаторами

1.3 Создание модуля верхнего уровня

Модуль верхнего уровня имеет название «Controller». 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, использующаяся 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», «1'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», «1'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 Controller(
    input [3:0] SWITCHES,
    input button_in, clk, button_reset_in,
    output [7:0] AN,
    output [7:0] SEG);

wire button_signal, button_signal_en, reset_signal_en, reset_signal, clk_div;
reg[7:0] AN_MASK = 8'b11111111;
reg[31:0] NUMBER = 0;

debouncer #(128) dbnc(
    .clk(clk),
    .in_signal(button_in),
    .CLOCK_ENABLE(1'b1),
    .out_signal(button_signal),
    .out_signal_enable(button_signal_en));

debouncer #(128) dbnc_reset(
    .clk(clk),
```

Продолжение Листинга 1.6

```
    .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));

always@(posedge clk)
begin
     NUMBER <= {NUMBER[31:4], SWITCHES};
     AN_MASK <= {AN_MASK[7:1], 1'b0};
     if (reset_signal)
     begin
          NUMBER <= 0;
          AN_MASK <= 8'b11111111;
     end
     else
          if (button_signal_en)
          begin
               NUMBER <= NUMBER << 4;
               AN_MASK <= AN_MASK << 1;
          end
end

endmodule
```

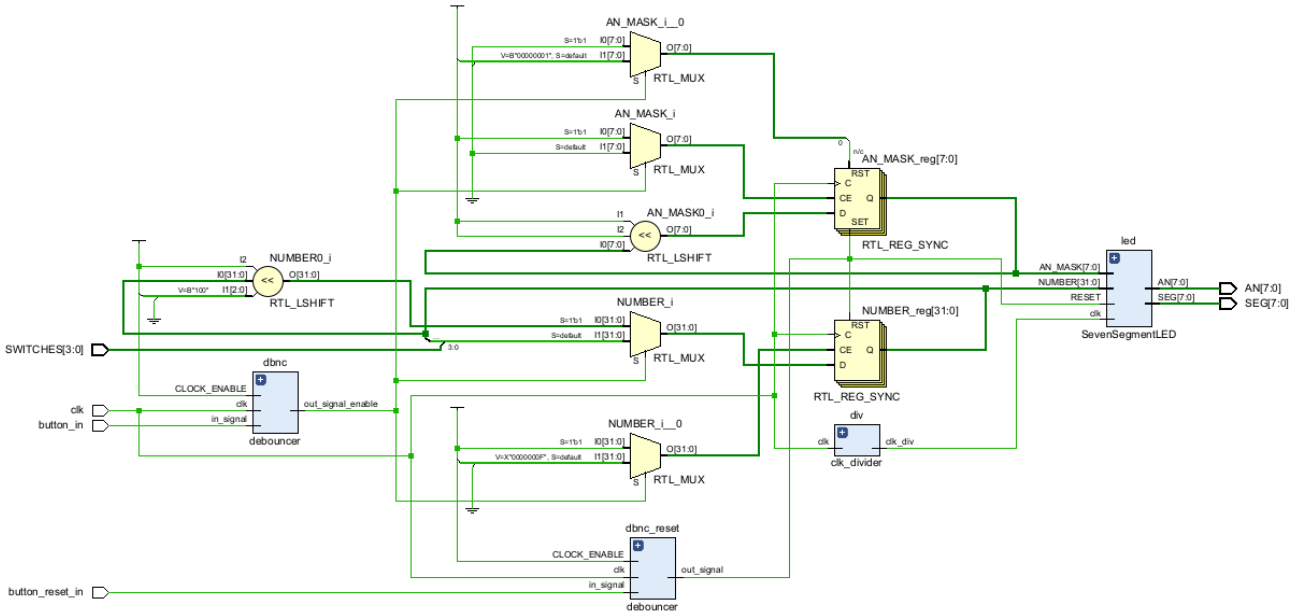

Рисунок 1.6 - RTL-схема модуля верхнего уровня

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 сымитировать 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[7:0] SEG;


Controller cntlr(
    .SWITCHES(SWITCHES),
    .button_in(button),
```

Продолжение Листинга 2.1

```
    .clk(clk),
    .button_reset_in(button_reset),
    .AN(AN),
    .SEG(SEG));

always #5 clk = ~clk;

initial
begin
    #50
    $srandom(35000);
    SWITCHES = 4'b1111;
    repeat($urandom_range(50,0))
    begin
        button = $random;
        #3;
    end
    button = 1;
    #100;

    repeat($urandom_range(50,0))
    begin
        button = $random;
        #3;
    end
    button = 0;
    #100;

    SWITCHES = 4'b0110;
    repeat($urandom_range(50,0))
    begin
        button = $random;
        #3;
    end
    button = 1;
    #100;

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

    repeat($urandom_range(50,0))
    begin
        button_reset = $random;
        #3;
    end
    button_reset = 1;
    #100;

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

endmodule
```

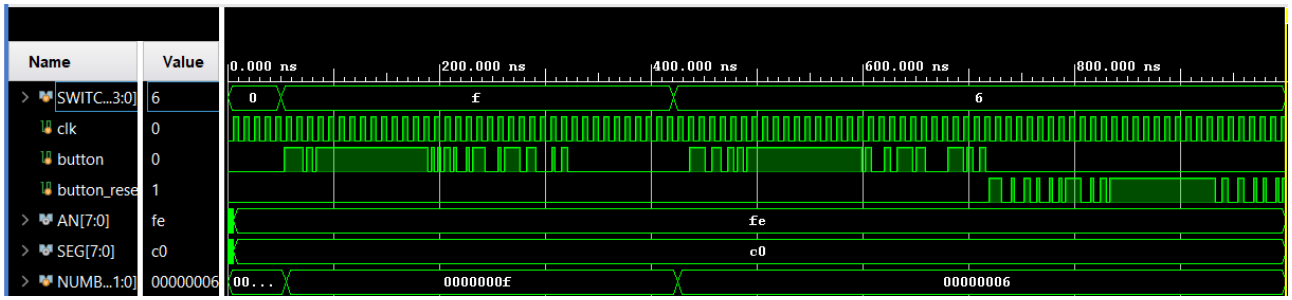

Рисунок 2.1 – Результат верификации тестового модуля

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 -add -name sys_clk -period 10.00 -waveform {0 5} [ get_ports {
clk }]
set_property IOSTANDARD LVCMOS33 [ get_ports { clk } ]
set_property PACKAGE_PIN E3 [ get_ports { clk } ]

set_property IOSTANDARD LVCMOS33 [ get_ports { button_in } ]
set_property PACKAGE_PIN N17 [ get_ports { button_in } ]

set_property IOSTANDARD LVCMOS33 [ get_ports { button_reset_in } ]
set_property PACKAGE_PIN P18 [ get_ports { button_reset_in } ]

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] } ]

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] } ]

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] } ]
```

Продолжение Листинга 3.1

```
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] } ]

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

set_property IOSTANDARD LVCMOS33 [ get_ports { SEG[7] } ]
set_property PACKAGE_PIN H15 [ get_ports { SEG[7] } ]
```

Файл с расширением .bit сгенерирован и загружен на плату. Результат представлен на Рисунке 3.1.

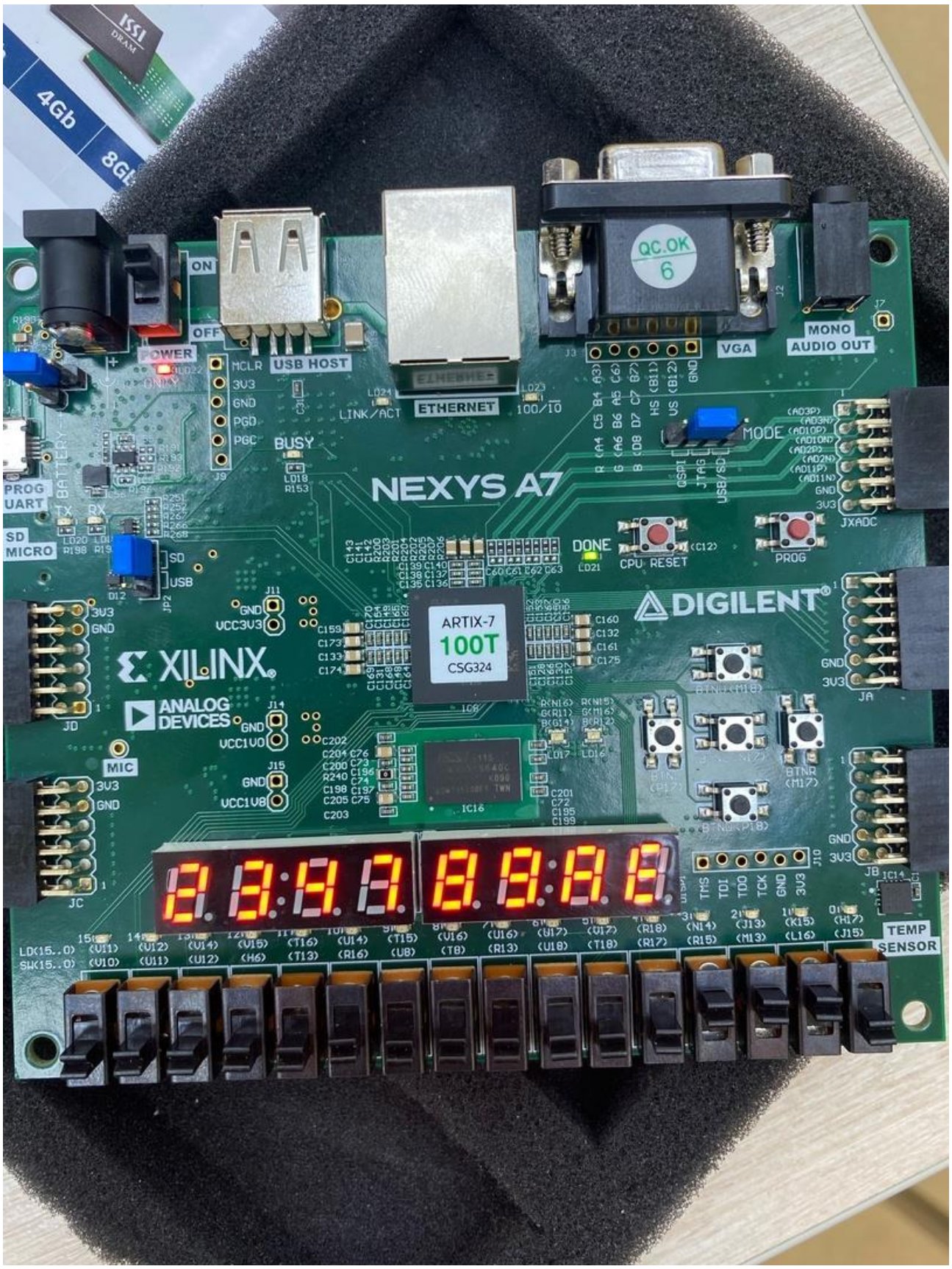

Рисунок 3.1 – Полное заполнение на плате

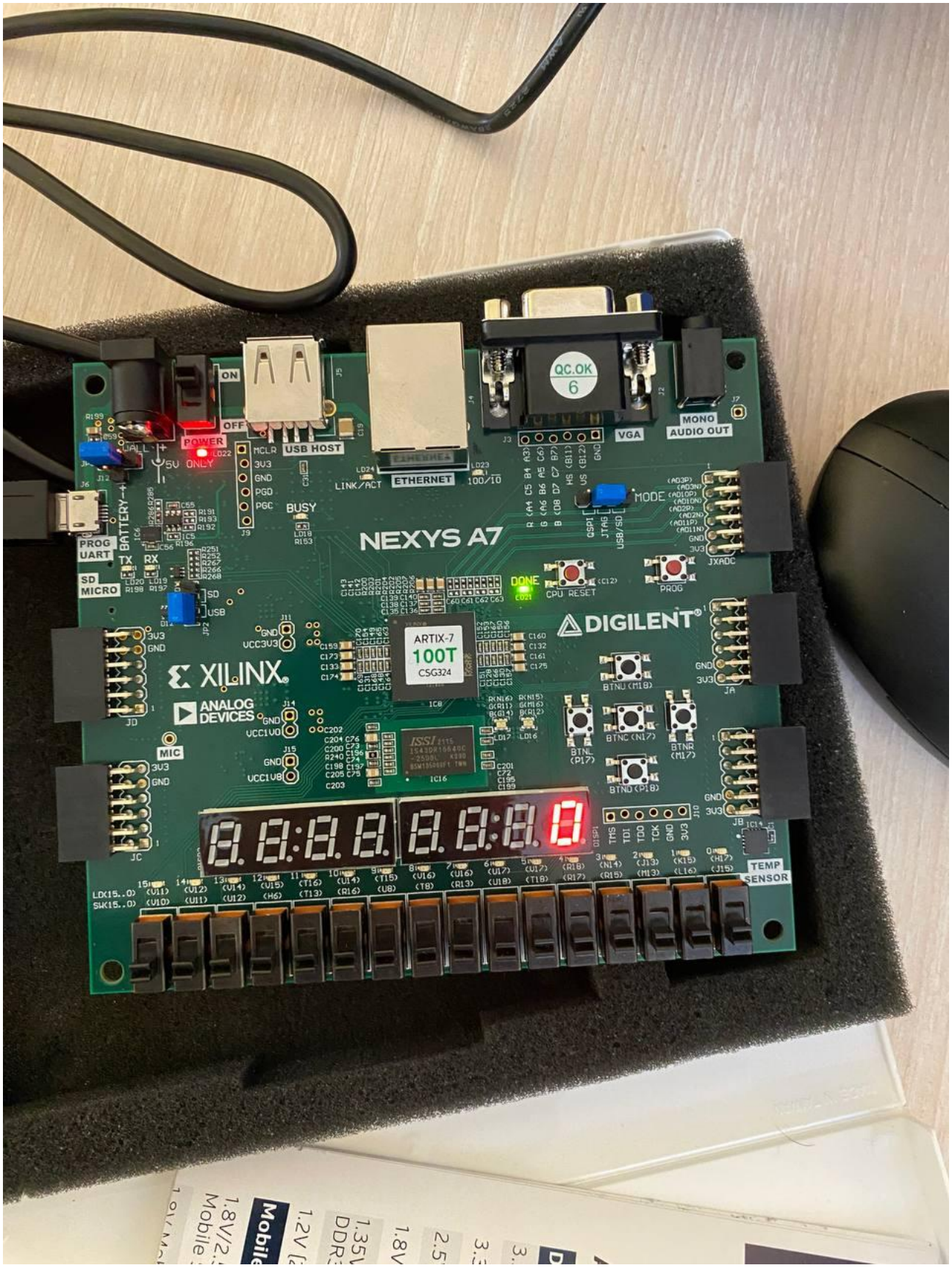

Рисунок 3.2 – Включение платы

На Рисунке 3.3 показывается выставление значения “4”.

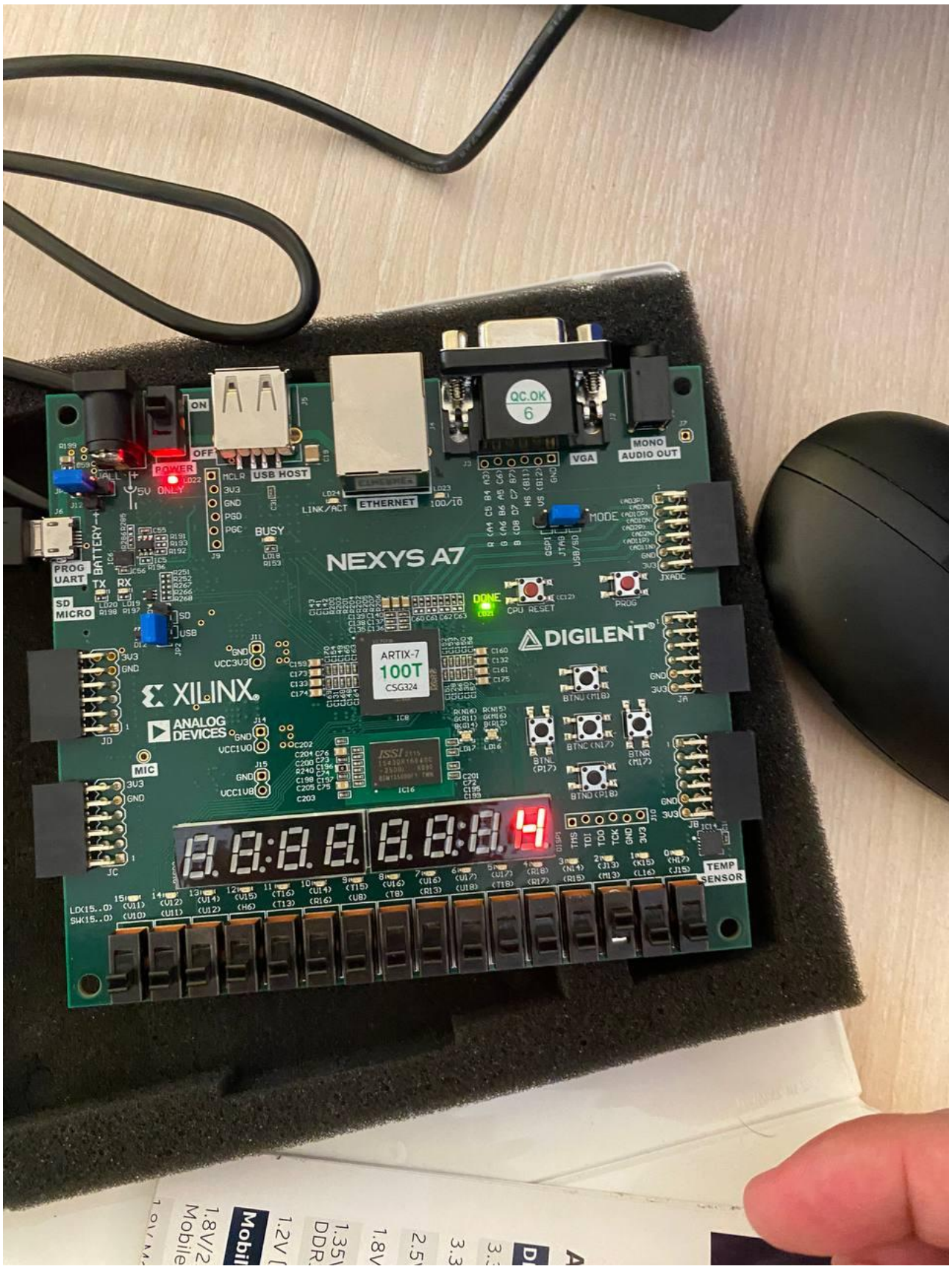

Рисунок 3.3 – Выставление значения на плате

На Рисунке 3.3 показывается запись значения “4” на плату при помощи кнопки (N 17), после чего значение смещается в лево на следующий семисегментный индикатор.

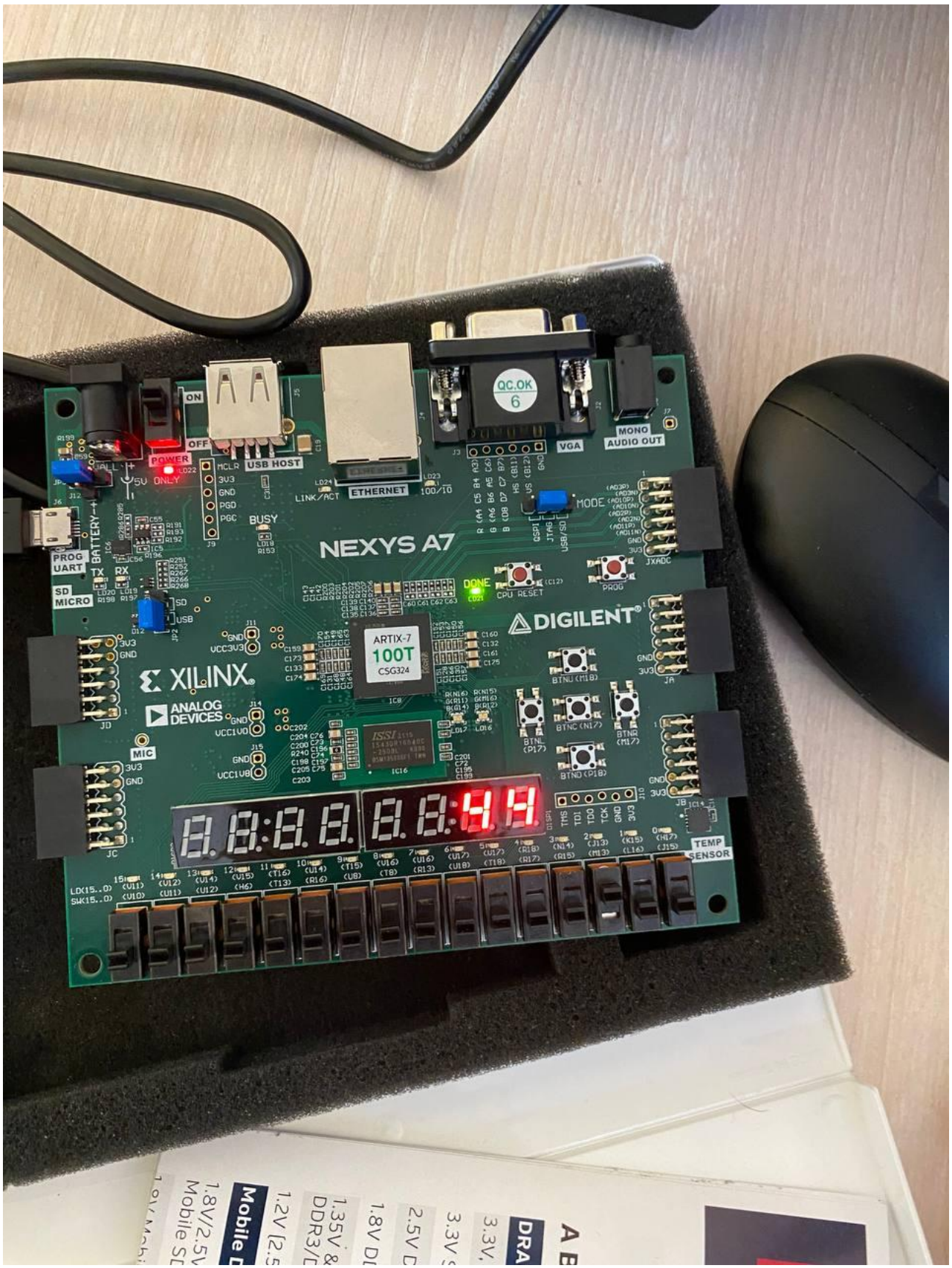

Рисунок 3.4 – Запись значения на плату

Также была добавлена возможность сброса записанных значений по средствам кнопки (P18).

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.