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 № 4

«Проектирование аппаратного драйвера для приёма данных по протоколу PS/2.»

по дисциплине

«Схемотехника устройств компьютерных систем»

Выполнил студент группы Туктаров Т.А.

ИВБО-11-23

Принял ассистент кафедры ВТ Дуксина И.И.

Лабораторная работа выполнена «__»_____2025 г.

«Зачтено» «_»_____2025 г.

Москва 2025

АННОТАЦИЯ

Данная работа включает в себя 9 рисунков, 13 листингов. Количество страниц в работе — 33.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

До текущего момента все разрабатываемые в рамках лабораторных работ устройства были основаны на автоматной логике, а вопрос взаимодействия с внешней средой на ввод информации решался за счёт использования кнопок и движковых переключателей, которые непосредственно размещены на отладочной плате; вывод информации осуществлялся при помощи простейших средств индикации, а именно светодиодов и семисегментных индикаторов.

Даже в случае декомпозиции устройства на ряд автоматов любые потоки передачи информации не выходили за рамки отладочной платы.

При разработке более сложных комплексов могут возникать задачи, связанные с передачей данных между устройствами, соединёнными при помощи проводных и беспроводных технологий, находящихся в одном помещении или же разнесёнными на внушительное расстояние. Если в случае разработки устройства, чей масштаб производства ограничивается единичным экземпляром, формат данных, определяемый для их передачи, может выбираться произвольно, то переходя к серийному производству и, самое важное, сопряжению с другими устройствами различных производителей, на первое место выходит вопросы унификации и стандартизации процессов приёма и передачи информации.

Среди множества протоколов передачи данных в рамках лабораторных работ будут рассмотрены вопросы построения аппаратных драйверов для тех протоколов, которые могут быть реализованы без использования знаний в дополнительных областях. К таким протоколам может быть отнесён протокол PS/2.

1 ХОД РАБОТЫ

1.1 Постановка задачи

Для устройства субквадратичной сортировки массива обеспечить возможность ввода данных с использованием клавиатуры, работающей по протоколу PS/2. Провести моделирование и верификацию работы устройства согласно выданному варианту. Реализовать тестовый модуль, позволяющий проверить корректность работы проектируемого устройства на различных входных данных. Проанализировать роль верификации на этапах проектирования цифровых систем. Вариант: Субквадратичная сортировка массива

1.2 Системная модель

1.2.1 Алгоритм автомата

Для данной задачи алгоритм с точки зрения достаточно высокого уровня абстракции будет представлен ниже:

1. Ввести размер массива
2. Если размер больше 8, то вывести код ошибки
3. Отсортировать массив сортировкой слияния
4. Вывести массив

1.2.2 Тестовое покрытие системной модели

Результаты тестирования представлены в листинге 1.1. Таким образом, были сформированы эталонные наборы значений.

Листинг 1.1 — Результаты работы программной реализации

```
n: = 3 | arr: [7, 3, 1] | sorted: [1, 3, 7] | error code:  0
n: = 5 | arr: [7, 3, 1, 2] | sorted: [1, 2, 3, 7] | error code:  0
n: = 9 | arr: [] | sorted: [] | error code:  1
```

1.3 Описание автомата на Verilog HDL

Исходный код представлен в Листинге 1.2.

Листинг 1.2 — Программная реализация автомата на Verilog HDL

```
module fsm#(max_size = 16, bit_size = 16)
(
    input [bit_size-1:0] n,
    input [bit_size-1:0] dataIn,
    input R_I,
    input reset,
    input clk,
    output reg [bit_size * max_size - 1: 0] dataOut,
    output reg R_O
    );

localparam S0 = 4'b0000, S1 = 4'b0001, S2 = 4'b0010, S3 = 4'b0011, S4 = 4'b0100,
S5 = 4'b0101, S6 = 4'b0110, S7 = 4'b0111, S8 = 4'b1000, S9 = 4'b1001;

reg [bit_size-1:0] arr [0:max_size-1]; // Массив

reg [bit_size * max_size - 1:0] flattered_arr; // Норм массив блэать.

reg [3:0] state; // состояния

reg [bit_size - 1:0] counter; // счетчик индекса массива

integer i;
integer j;
integer k;
integer g;

reg [bit_size - 1:0] it1;
reg [bit_size - 1:0] it2;

reg [bit_size - 1:0] result [0:max_size - 1]; // массив для хранения временных
отсортированных данных.

reg [1:0] z;

reg [bit_size * max_size - 1: 0] packed;

reg loop_flag;

reg [bit_size - 1:0] left;
reg [bit_size - 1:0] mid;
reg [bit_size - 1:0] right;
initial
begin
    loop_flag <= 0;
    state <= S0;
    // n <= 0;
    it1 <= 0;
    it2 <= 0;
    counter <= 0;
    packed <= 0;
    R_O <= 0;

    left <= 0;
    mid <= 0;
    right <= 0;
```

Продолжение листинга 1.2

```
    for(i = 0; i < max_size; i = i + 1)begin
        arr[i] <= 0;
        result[i] <= 0;
    end
end

always@(posedge clk)
begin
    if(reset)begin
        state <= S0;
    end
    else
    case(state)
        S0:
        begin
            // n <= 0;
            it1 <= 0;
            it2 <= 0;
            counter <= 0;
            R_O <= 0;
            dataOut <= 0;
            loop_flag <= 0;
            for(i = 0; i < max_size; i = i + 1)begin
                arr[i] = 0;
                result[i] = 0;
            end
            state <= S2;
        end
        S2: // ввод массива
        begin
            if(R_I) begin
                arr[counter] <= dataIn;
                counter <= counter + 1;
            end
            if(counter == n)begin
                state <= S3;
                i <= 1;
            end
        end // сортировка

        S3: begin
            if(i < n)begin
                j <= 0;
                state <= S4;
            end else state <= S9;
        end

        S4:begin
            if(j < n - i)begin
                left <= j;
                mid <= j + i;
                if(j + 2 * i < n)
                    right <= j + 2 * i;
                else
                    right <= n;
                it1 <= 0;
                it2 <= 0;
                $display(" i = %0h, j = %0h", i, j);
                j <= j + 2 * i;
                state <= S5;
            end else begin
                state <= S3;
```

Продолжение листинга 1.2

```
                i <= i * 2;
            end
        end

        S5: begin
            if((left + it1 < mid) && (mid + it2 < right))begin
                if(arr[left + it1] < arr[mid + it2])begin
                    result[it1 + it2] = arr[left + it1];
                    it1 = it1 + 1;
                end
                else begin
                    result[it1 + it2] = arr[mid + it2];
                    it2 = it2 + 1;
                end
                state <= S5;
            end else
                state <= S6;
        end

        S6:begin
            if(left + it1 < mid)begin
                result[it1 + it2] = arr[left + it1];
                it1 = it1 + 1;
                state <= S6;
            end else state <= S7;
        end

        S7:begin
            if(mid + it2 < right)begin
                result[it1 + it2] = arr[mid + it2];
                it2 = it2 + 1;
                state <= S7;
            end else state <= S8;
        end

        S8:begin
            for(g = 0; g < it1 + it2; g = g + 1)begin
                arr[left + g] = result[g];
            end
            state <= S4;
        end
        S9: begin
            R_O <= 1'b1;
            for (i = 0; i < max_size; i = i + 1) begin
                packed[i*bit_size +: bit_size] = arr[n - 1 - i];  // "+:" -
part-select
            end
            dataOut <= packed;
        end
    endcase
end
endmodule
```

1.3.1 Этап тестирования

На данном этапе основной целью является получение значений, идентичных эталонным, а также проверка различных особенностей работы

самой схемы. Пример тестового модуля представлен в Листинге 1.3. Результат симуляции представлен на Рисунке 1.3. В ходе тестирования были получены ответы, идентичные эталонным, а сброс автомата в действительности привёл автомат в начальное положение. [4]

Рисунок 1.1 — Временная диаграмма работы устройства

1.4 Реализация основных модулей

1.4.1 Реализация синхронизатора

Опишем синхронизатор в модулес synchro. Реализация представлена в Листинге 1.3.

Листинг 1.3 — Модуль synchro.v

```
module synchro(
 input in_signal,clk,
 output q
 );
 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
```

1.4.2 Реализация счётчика

Опишем счётчик в модулес count. Реализация представлена в Листинге 1.4.

Листинг 1.4— Модуль count.v

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

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

endmodule
```

1.4.3 Реализация триггера

Опишем d-trigger в модулес dtrigger. Реализация представлена в Листинге 1.5.

Листинг 1.5 — Модуль dtrigger.v

```
`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
```

1.4.4 Реализация делителя частоты

Опишем параметризированный делитель частоты в модулес clk_divider. Реализация представлена в Листинге 1.6.

Листинг 1.6 — Модуль clk_divider.v

```
`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)) cntr(
 .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
```

1.4.5 Реализация модуля управления семисегментными индикаторами

Опишем модуль управления семисегментными индикаторами в модуле SevenSegmentLED. Реализация представлена в Листинге 1.7.

Листинг 1.7 — Модуль SevenSegmentLED.v

```
module SevenSegmentLED(
 input [7:0] AN_MASK,
 input [31:0] NUMBER,
 input clk,
 output [7:0] AN,
 output reg[7:0] SEG);
wire[2:0] counter_res;
count #(.mod(8), .step(1)) cntr(
 .clk(clk),
 .RE(1'b0),
 .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
 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
endmodule
```

1.4.6 Реализация модуля фильтра дребезга контактов

Опишем фильтр дребезга контактов в модулес filtercon. Реализация представлена в Листинге 1.8.

Листинг 1.8 — Модуль filtercon.v

```
`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_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;
endmodule
```

1.4.7 Реализация модуля дешифратора для PS/2

Опишем модуль дешифратора PS/2, реализующие коды клавиш. Реализация представлена в Листинге 1.9.

Листинг 1.9 — Модуль PS2_DC.v

```
module PS2_DC(
    input [7:0] keycode,
    input double_mode,
    output reg [3:0] out,
    output reg [4:0] flags
);

reg [7:0] NUMBERS [0:15];
parameter [7:0] ENTER_CODE = 8'h5A;
parameter [7:0] BACKSPACE_CODE = 8'h66;
parameter [7:0] RESET_CODE = 8'h2D;
parameter [7:0] DELETE_CODE = 8'h71;
parameter NUMBER_F = 0, ENTER_F = 1, BACKSPACE_F = 2, RESET_F = 3, DELETE_F = 4;

initial
begin
    NUMBERS[0] = 8'h45;
    NUMBERS[1] = 8'h16;
    NUMBERS[2] = 8'h1E;
    NUMBERS[3] = 8'h26;
    NUMBERS[4] = 8'h25;
    NUMBERS[5] = 8'h2E;
    NUMBERS[6] = 8'h36;
    NUMBERS[7] = 8'h3D;
    NUMBERS[8] = 8'h3E;
    NUMBERS[9] = 8'h46;
    NUMBERS[10] = 8'h1C;
    NUMBERS[11] = 8'h32;
    NUMBERS[12] = 8'h21;
    NUMBERS[13] = 8'h23;
    NUMBERS[14] = 8'h24;
    NUMBERS[15] = 8'h2B;

    out = 0;
    flags = 0;
end

always@(keycode)
begin
     case(keycode)
            // Цифры
            NUMBERS[0] : out = 4'h0;
            NUMBERS[1] : out = 4'h1;
            NUMBERS[2] : out = 4'h2;
            NUMBERS[3] : out = 4'h3;
            NUMBERS[4] : out = 4'h4;
            NUMBERS[5] : out = 4'h5;
            NUMBERS[6] : out = 4'h6;
            NUMBERS[7] : out = 4'h7;
            NUMBERS[8] : out = 4'h8;
            NUMBERS[9] : out = 4'h9;
            NUMBERS[10]: out = 4'hA;
            NUMBERS[11]: out = 4'hB;
```

Продолжение листинга 1.9

```
            NUMBERS[12]: out = 4'hC;
            NUMBERS[13]: out = 4'hD;
            NUMBERS[14]: out = 4'hE;
            NUMBERS[15]: out = 4'hF;

            // Клавиша "Enter"
            ENTER_CODE: out = 0;
            BACKSPACE_CODE: out = 0;
            RESET_CODE: out = 0;
            DELETE_CODE: begin
                if(double_mode)
                    out = 0;
            end
            default: out = 0;
     endcase
end

always@(keycode)
begin
    case(keycode)
        NUMBERS[0],  NUMBERS[1],  NUMBERS[2],  NUMBERS[3],
        NUMBERS[4],  NUMBERS[5],  NUMBERS[6],  NUMBERS[7],
        NUMBERS[8],  NUMBERS[9],  NUMBERS[10], NUMBERS[11],
        NUMBERS[12], NUMBERS[13], NUMBERS[14], NUMBERS[15]:
            flags <= 1 << NUMBER_F;
        ENTER_CODE:
            flags <= 1 << ENTER_F;
        BACKSPACE_CODE:
            flags <= 1 << BACKSPACE_F;
         RESET_CODE:
              flags <= 1 << RESET_F;
         DELETE_CODE: begin
                if(double_mode)
                    flags <= 1 << DELETE_F;
            end
        default:
            flags <= 0;
    endcase
end

endmodule
```

1.4.8 Реализация модуля PS/2

Опишем модуль PS/2. Реализация представлена в Листинге 1.10.

Листинг 1.10 — Модуль PS2.v

```
 module PS2(
    input clk,
    input PS2_clk,
    input PS2_dat,
    output [7:0] out,
    output reg R_O,
    output reg ERROR
);

parameter WAIT_START_BIT = 0,
          IDLE = 1,
```

Продолжение листинга 1.10

```
          WRITE = 2,
          PARITY_BIT = 3,
          STOP_BIT = 4;
reg [2:0] state;
reg [3:0] cnt;
reg [7:0] PS2_buf;
reg [1:0] PS2_clk_sync, PS2_dat_sync;

assign out = PS2_buf;

initial
begin
    cnt = 0;
    R_O = 0;
    ERROR = 0;
    state = WAIT_START_BIT;
    PS2_buf = 0;
    PS2_clk_sync = 2'b11;
    PS2_dat_sync = 2'b11;
end


always@(posedge clk)
begin
    PS2_clk_sync <= {PS2_clk_sync[0], PS2_clk};
    PS2_dat_sync <= {PS2_dat_sync[0], PS2_dat};
end


// Блок управления буфером для записи пакета
always@(negedge PS2_clk_sync[1])
begin
    case(state)

        // Ожидание стартового бита
        WAIT_START_BIT:
        begin
            R_O <= 0;
            ERROR <= 0;
            state <= ~PS2_dat_sync[1] ? WRITE : IDLE;
        end

        // Ожидание конца пакета
        IDLE:
            if (cnt == 4'd10)
            begin
                ERROR <= 1;
                R_O <= 1;
                state <= WAIT_START_BIT;
            end
        // Обработка битов данных
        WRITE: begin
            if (cnt == 4'd8)
                state <= PARITY_BIT;
            PS2_buf <= {PS2_dat_sync[1], PS2_buf[7:1]};
        end

        // Обработка бита чётности
        PARITY_BIT: begin
            if ((~^PS2_buf) == PS2_dat_sync[1])
                state <= STOP_BIT;
            else
```

Продолжение листинга 1.10

```
                state <= IDLE;
        end

        // Обработка стоп-бита
        STOP_BIT: begin
            if (!PS2_dat_sync[1])
                ERROR <= 1;
            R_O <= 1;
            state <= WAIT_START_BIT;
        end
    endcase
end

always@(negedge PS2_clk_sync[1])
begin
    cnt <= cnt + 1;
    if (cnt == 4'd10)
        cnt <= 0;
end

endmodule
```

1.4.9 Реализация модуля управления для PS/2

Опишем модуль PS/2_Manager, реализующий логику флагов. Реализация представлена в Листинге 1.11.

Листинг 1.11 — Модуль PS2_Manager.v

```
module PS2_Manager (
    input clk,
    input PS2_dat,
    input PS2_clk,

    output reg R_O,
    output [3:0] out,
    output [4:0] flags
);
// Состояние ожидания установки сигнала готовности пакета данных
parameter WAIT_ONE = 0;
// Состояние ожидания сброса сигнала готовности пакета данных
parameter WAIT_ZERO = 1;
reg state;
// Выходы модуля приёма одного пакета PS2
wire PS2_R_O, PS2_ERROR;
wire [7:0] PS2_out;
// Регистр флага отжатия клавиши
reg release_flag;
reg double_mode;
initial begin
    state = 0; R_O = 0;
    release_flag = 0;
    double_mode = 0;
end

always@(posedge clk)
begin
    case(state)
```

Продолжение листинга 1.11

```
        WAIT_ONE: begin
            if (PS2_R_O) begin
                if (!PS2_ERROR) begin
                    if (PS2_out == 8'hE0) begin
                        double_mode<=1;
                    end else
                    if (PS2_out == 8'hF0)
                        release_flag <= 1;
                    else if (release_flag)
                    begin
                        R_O <= 1;
                        release_flag <= 0;
                        double_mode<=0;
                    end
                end
                state <= WAIT_ZERO;
            end
        end
        WAIT_ZERO: begin
            R_O <= 0;
            if (!PS2_R_O)
                state <= WAIT_ONE;
        end
   endcase
end
// Приём одного пакета
PS2 ps2(
    .clk(clk),
    .PS2_clk(PS2_clk), .PS2_dat(PS2_dat),
    .out(PS2_out), .R_O(PS2_R_O),
    .ERROR(PS2_ERROR)
);
// Дешифратор пакета
PS2_DC dc(.keycode(PS2_out), .double_mode(double_mode), .out(out),
.flags(flags));
endmodule
```

1.4.10 Реализация модуля верхнего уровня

Опишем модуль верхнего уровня в модуле main. Реализация представлена в Листинге 1.12.

Листинг 1.12 — Модуль main.v

```
    `timescale 1ns / 1ps
    module main(
    //input [3:0] SWITCHES,
    //input button_in, clk, button_reset_in,
    output reg error_led,
    //output reg non_error_led,
    output [7:0] AN,
    output [6:0] SEG,
    input clk,
    input PS2_clk,
    input PS2_dat,
    output reg [7:0] LEDS
    );
    reg non_error_led;
```

Продолжение листинга 1.12

```
    wire button_signal_en, reset_signal_en,button_signal,reset_signal;
    wire clk_div;
    reg[7:0] AN_MASK = 8'b11111111;
    reg flag = 0;
    reg enable = 0;
    reg is_data_last = 0;
    reg [31:0] NUMBER;
    reg [4*8-1:0] data_in = 0;
    wire [4*8-1:0] data_out;
    reg [3:0] count_in = 0;
    reg [3:0] data_last = 0;
    reg [31:0] error_code = 0;
    wire data_out_enable;
    reg [1:0] flag1 = 0;
    reg error_enable;
    reg [3:0] n = 0;
    wire R_O;
    wire [4:0] flags;
    reg [3:0] current_number;
    wire [3:0] out;
    reg reset;



    initial begin
    NUMBER <= 0;
    reset<=0;
    end

    PS2_Manager ps2_m(
    .clk(clk),
    .PS2_dat(PS2_dat),
    .PS2_clk(PS2_clk),
    .R_O(R_O),
    .out(out),
    .flags(flags)
    );

    clk_divider #(1024) div(
    .clk(clk),
    .clk_div(clk_div));
    SevenSegmentLED led(
    .AN_MASK(AN_MASK),
    .NUMBER(NUMBER),
    .clk(clk),
    .ce(clk_div),
    .RESET(reset),
    .AN(AN),
    .SEG(SEG));
    batcher_sort_fsm #(8,4)uut(
        .data_count(n),
        .data_in(data_in),
        .reset(reset),
        .enable(enable),
        .clk(clk),
        .data_out(data_out),
        .data_out_enable(data_out_enable)
    );
    integer i;
    always@(posedge clk)
        begin
         if(reset)
```

Продолжение листинга 1.12

```
            reset = ~reset;
        if(R_O) begin
            if (flags==16)
                begin
                    reset = 1;
                    NUMBER <= 0;
                    AN_MASK <= 8'b11111111;
                    error_led =0;
                    n<=4'd0;
                    data_last<=0;
                    is_data_last<=0;
                    flag<=0;enable<=0;
                    LEDS<=8'b00000000;
                    count_in<=0;
                    data_in<= 0;
                    flag1<=0;
                    non_error_led=0;
                    current_number<=0;
                    error_enable<=0;
                end
            else
                if(error_enable) begin
                        NUMBER <= error_code;
                        AN_MASK<=~8'b11111111;
                        error_led<=1;
                end else begin
                if (flags==1)
                begin
                     current_number <= out;
                end
                else
                if (flags==8)
                begin
                    current_number<=4'b1010;
                end
                else
                if (flags==4)
                begin
                    current_number<=0;
                end
                else
                if (flags==2)
                    begin
                        if(flag == 0)
                            begin
                                flag<=1;
                                n <= current_number;
                                if(current_number==0) begin
                                error_code<=32'h77777777;
                                error_enable<=1;
                            end
                            else if(current_number>8) begin
                                error_code<=32'h88888888;
                                error_enable<=1;
                            end
                            end
                            else
                            begin
                                if(count_in == n) begin
                                    if(n <8) begin
                                        data_in = pad_data(data_in, n);
                                    end
```

Продолжение листинга 1.12

```
                                    enable <= 1'b1;
                                end
                                else
                                    begin
                                        data_in <= {data_in[27:0],
current_number};
                                        count_in <= count_in + 1;
                                        LEDS <= {LEDS[6:0], 1'b1};
                                    end
                            end
                           current_number<=0;
                    end
                    end
                    end
                        if(~error_enable) begin
                        if(data_out_enable) begin
                            non_error_led<=1;
                            NUMBER <= data_out;
                            if (flag1 == 0) begin
                                AN_MASK <= 8'b11111111;
                                flag1 = flag1 +1;
                            end
                            else if(flag1 == 1) begin
                                AN_MASK<= AN_MASK << n;
                                flag1 = 2;
                            end
                        end
                        else begin
                            NUMBER<=current_number;
                            AN_MASK <= ~8'b00000001;
                        end
                        end
                end
    function [31:0] pad_data;
        input [31:0] raw_data;
        input [3:0] n;
        reg [31:0] padding;
        begin
            case (n)
                0: padding = {8{4'b1111}};
                1: padding = {7{4'b1111}};
                2: padding = {6{4'b1111}};
                3: padding = {5{4'b1111}};
                4: padding = {4{4'b1111}};
                5: padding = {3{4'b1111}};
                6: padding = {2{4'b1111}};
                7: padding = {1{4'b1111}};
                8: padding = 32'b0;
                default: padding = 32'b0;
            endcase
            pad_data = (raw_data << ((8 - n) * 4)) | padding;
        end
    endfunction
        initial begin
            current_number<=0;
            LEDS <= 0;
            error_led<=0;
            non_error_led<=0;
            error_enable<=0;
            error_code<=0;
        end
    endmodule
```

1.5 Создание и верификация тестового модуля верхнего уровня

Для тестирования был выбран набор тестов аналогичный тестам конечного автомата. Код тестового модуля представлен в Листинге 1.13.

Листинг 1.13 — Модуль testbench.v

```
`timescale 1ns / 1ns

module test();

parameter ENTER_CODE = 8'h5A;
parameter STOP_CODE = 8'hF0;
parameter BACKSPACE_CODE = 8'h66;
parameter RESET_CODE = 8'h2D;
parameter clk_period = 5;
parameter PS2_clk_period = 40;
parameter code_space_period = 60;
wire [3:0] key_value;
reg clk, PS2_clk, PS2_dat;
reg [7:0] key_code;
reg [3:0] i;
wire [7:0] AN;
wire [6:0] SEG;
main uut (
    .clk(clk),
    .PS2_clk(PS2_clk),
    .PS2_dat(PS2_dat),
    .AN(AN),
    //.out(key_value),

    .SEG(SEG)
);

always #(clk_period) clk <= ~clk;

initial
begin
    PS2_clk <= 1;
    PS2_dat <= 1;
    key_code <= 0;

    clk <= 0;

    @(posedge clk);
    @(posedge clk);

    #(2*clk_period) key_code = HEX_CD(4'h4);
    PS2_press_and_release_key(key_code);
    #5000;
    #(2*clk_period) key_code = ENTER_CODE;
    PS2_press_and_release_key(key_code);

    #(2*clk_period) key_code = HEX_CD(4'h7);
    PS2_press_and_release_key(key_code);
    #5000;
    #(2*clk_period) key_code = ENTER_CODE;
    PS2_press_and_release_key(key_code);
```

Продолжение листинга 1.13

```
    #(2*clk_period) key_code = HEX_CD(4'h3);
    PS2_press_and_release_key(key_code);
    #5000;
    #(2*clk_period) key_code = ENTER_CODE;
    PS2_press_and_release_key(key_code);

    #(2*clk_period) key_code = HEX_CD(4'h1);
    PS2_press_and_release_key(key_code);
    #5000;
    #(2*clk_period) key_code = ENTER_CODE;
    PS2_press_and_release_key(key_code);

    #(2*clk_period) key_code = HEX_CD(4'h2);
    PS2_press_and_release_key(key_code);
    #5000;
    #(2*clk_period) key_code = ENTER_CODE;
    PS2_press_and_release_key(key_code);

    #5000;
    $finish;

    #(2*clk_period)
    $finish;
end


// Нажатие и отпускание клавиши с расширенным кодом (E0 префикс)
task automatic PS2_press_and_release_extended_key(
    input [7:0] code
);
    begin
        fork
            PS2_gen_byte_clk();
            PS2_code_input(8'hE0); // Префикс E0
        join
        fork
            PS2_gen_byte_clk();
            PS2_code_input(code); // Код клавиши
        join
        #code_space_period;

        // Отпускание
        fork
            PS2_gen_byte_clk();
            PS2_code_input(8'hE0); // Префикс E0
        join
        fork
            PS2_gen_byte_clk();
            PS2_code_input(8'hF0); // Stop code
        join
        fork
            PS2_gen_byte_clk();
            PS2_code_input(code); // Код клавиши
        join
        #code_space_period;
    end
endtask



// Нажатие и отжатие клавиши
```

Продолжение листинга 1.13

```
task automatic PS2_press_and_release_key(
    input [7:0] code
);
    begin
        fork
            PS2_gen_byte_clk();
            PS2_code_input(code);
        join
        #code_space_period;
        fork
            PS2_gen_byte_clk();
            PS2_code_input(STOP_CODE);
        join
        #code_space_period;
        fork
            PS2_gen_byte_clk();
            PS2_code_input(code);
        join
    end
endtask

// Генерация пакета данных
task automatic PS2_code_input(
    input [7:0] code
);
    begin
        PS2_dat <= 0;
        for (i = 0; i < 8; i = i + 1)
        begin
            @(posedge PS2_clk)
            PS2_dat <= code[i];
        end
        @(posedge PS2_clk)
        PS2_dat <= ~^(code);

        @(posedge PS2_clk)
        PS2_dat <= 1;
    end
endtask

// Генерация синхросигнала для передачи пакета
task automatic PS2_gen_byte_clk;
    begin
        #(clk_period);
        repeat(22)
        begin
            PS2_clk <= ~PS2_clk;
            #(PS2_clk_period);
        end
        PS2_clk <= 1;
    end
endtask

function [7:0] HEX_CD;
    input [3:0] number_in;
    begin
        case(number_in)
            4'h0: HEX_CD = 8'h45;
            4'h1: HEX_CD = 8'h16;
            4'h2: HEX_CD = 8'h1E;
            4'h3: HEX_CD = 8'h26;
            4'h4: HEX_CD = 8'h25;
```

Продолжение листинга 1.13

```
            4'h5: HEX_CD = 8'h2E;
            4'h6: HEX_CD = 8'h36;
            4'h7: HEX_CD = 8'h3D;
            4'h8: HEX_CD = 8'h3E;
            4'h9: HEX_CD = 8'h46;
            4'hA: HEX_CD = 8'h1C;
            4'hB: HEX_CD = 8'h32;
            4'hC: HEX_CD = 8'h21;
            4'hD: HEX_CD = 8'h23;
            4'hE: HEX_CD = 8'h24;
            4'hF: HEX_CD = 8'h2B;
         default: HEX_CD = 0;
      endcase
    end
endfunction
endmodule
```

Результат каждого теста можно проверить по временной диаграмме, рассматривая значение «NUMBER», так как именно оно будет записано в семисигментные индикаторы. [5]

На Рисунке 1.2 представлена временная диаграмма для первого теста.

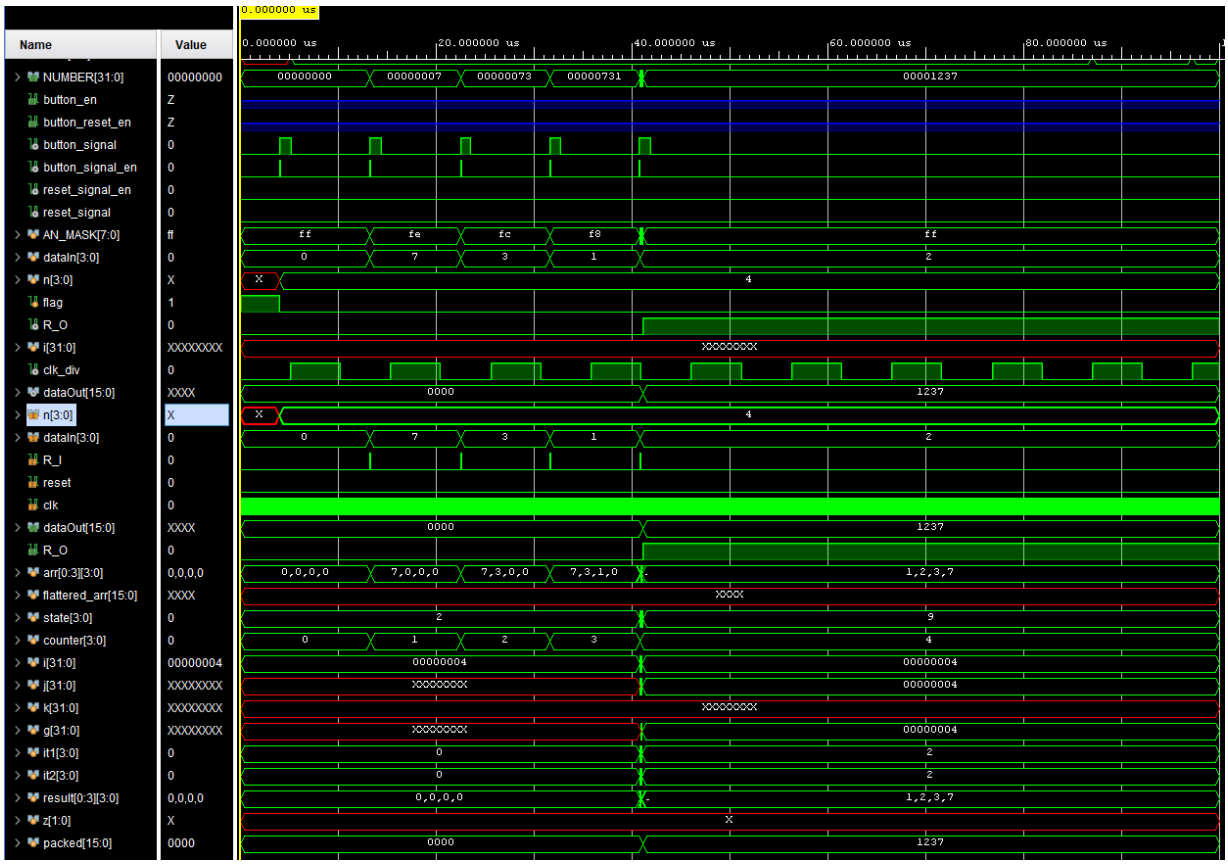

Рисунок 1.2 — Временная диаграмма первого теста

На Рисунке 1.3 представлена временная диаграмма для второго теста.

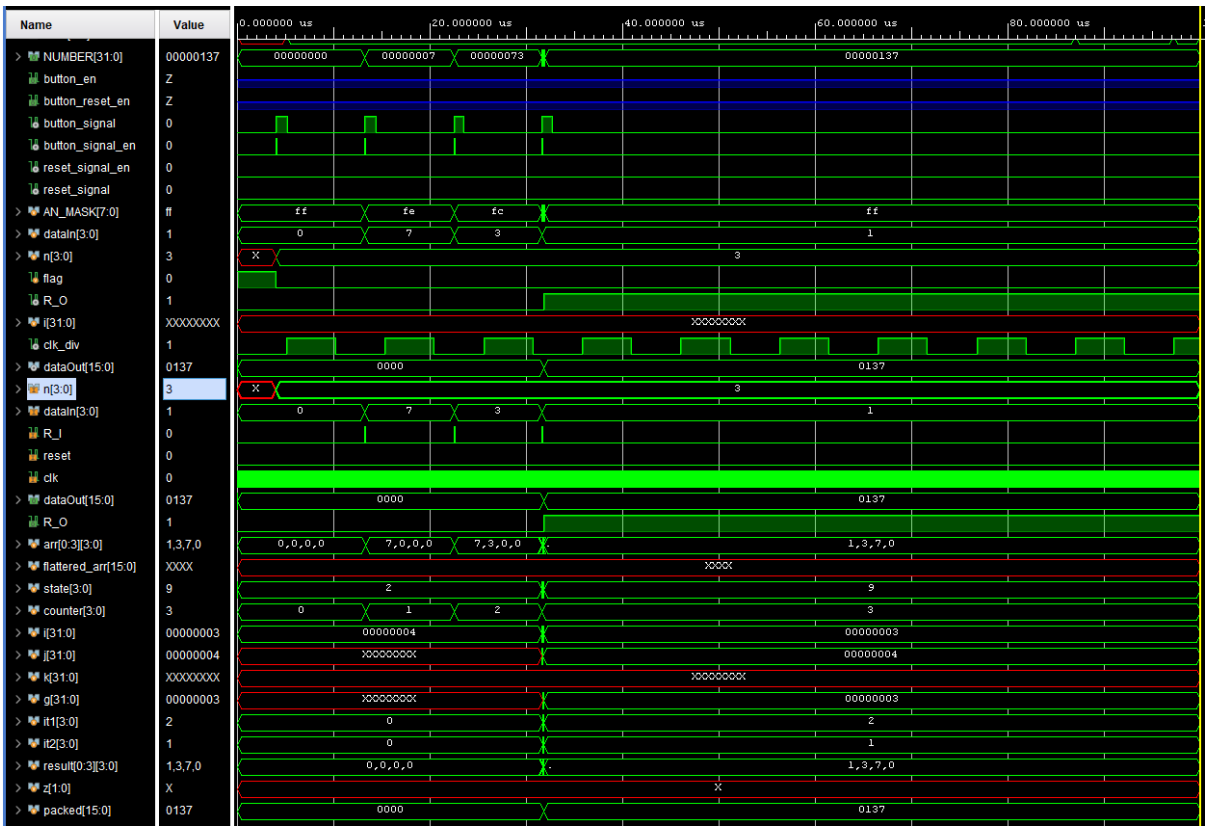

Рисунок 1.3 — Временная диаграмма второго теста

На Рисунке 1.4 представлена временная диаграмма для третьего теста.

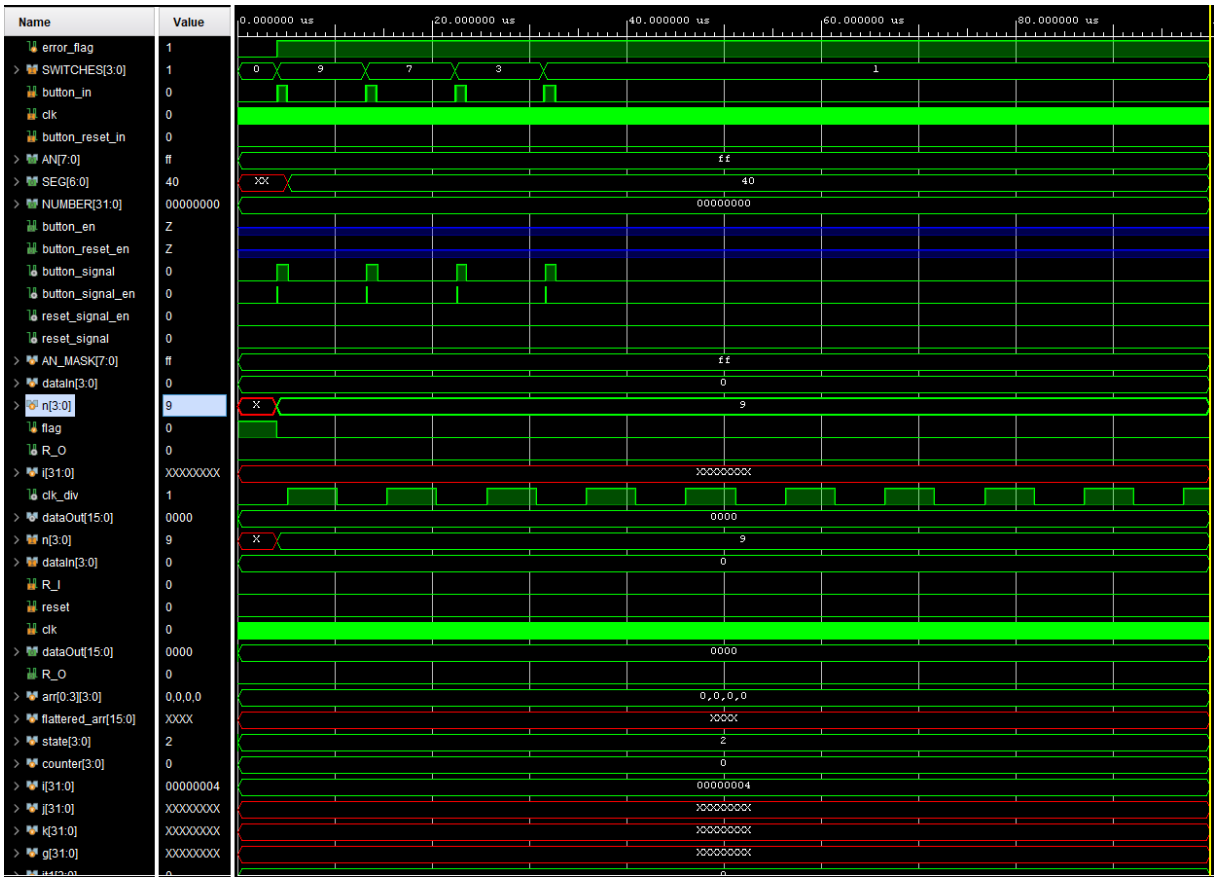

Рисунок 1.4 — Временная диаграмма третьего теста

1.6 Создание файла проектных ограничений и загрузка проекта на отладочную плату nexys a7

Содержание файла проектных ограничений представлено в Листинге 1.13.

Листинг 1.13 — Содержимое файла 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]

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]}]
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]}]

#LEDS
set_property IOSTANDARD LVCMOS33 [get_ports is_failed]
set_property PACKAGE_PIN N15 [get_ports is_failed]
set_property IOSTANDARD LVCMOS33 [get_ports is_minus_last]
set_property PACKAGE_PIN G14 [get_ports is_minus_last]

set_property IOSTANDARD LVCMOS33 [get_ports PS2_clk]
set_property PACKAGE_PIN  F4 [get_ports PS2_clk]
set_property IOSTANDARD LVCMOS33 [get_ports PS2_dat]
set_property PACKAGE_PIN  B2 [get_ports PS2_dat]
```

Проект был загружен на отладочную плату NEXYS A7 и протестирован. На Рисунках 1.5–1.9.

Рисунок 1.5 — Запись размера массива

Рисунок 1.6 — Запись значений массива

Рисунок 1.7 — Результат теста 1

Рисунок 1.8 — Результат теста 2

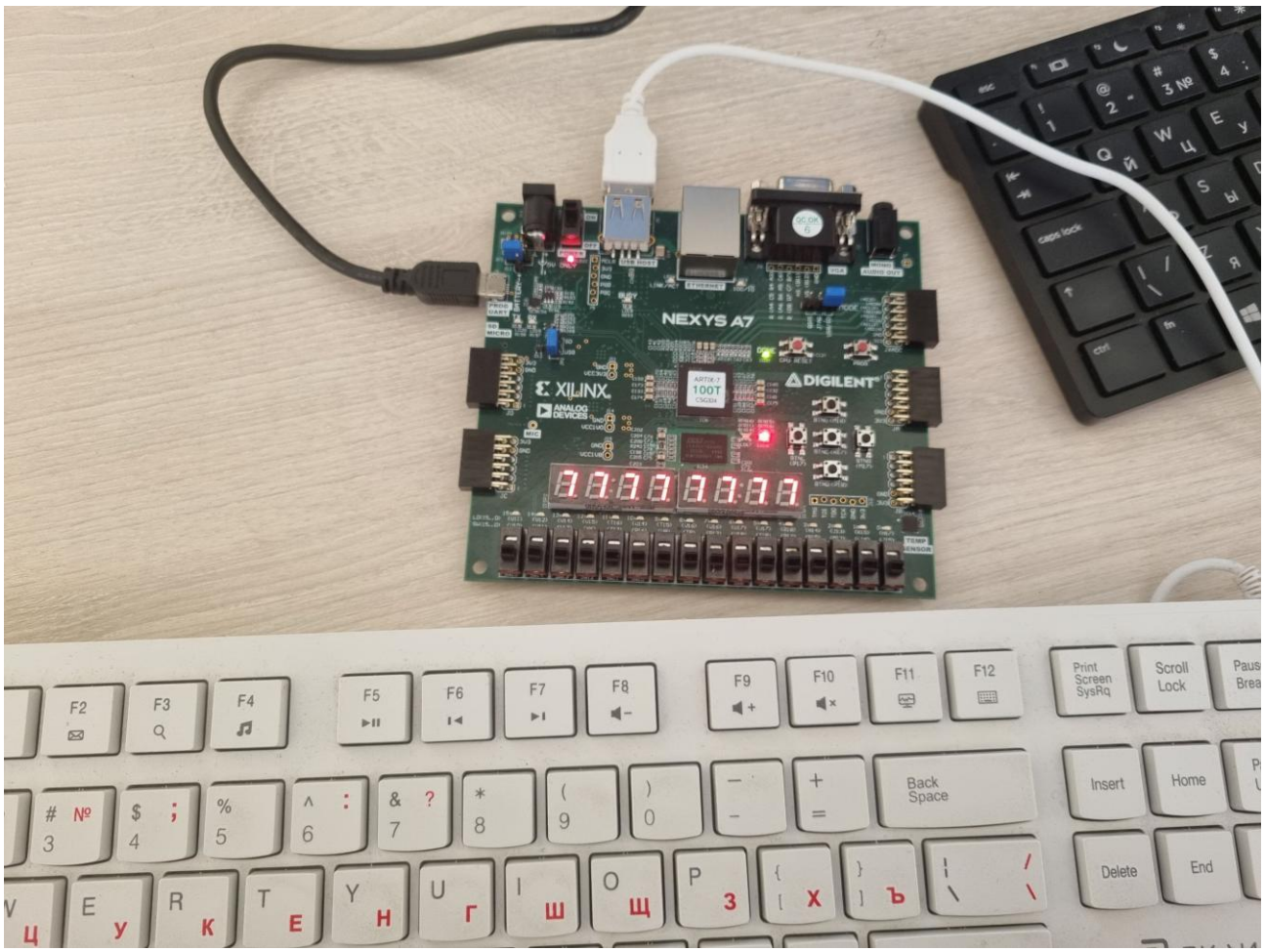

Рисунок 1.9 — Вывод ошибки как результат теста 3

ЗАКЛЮЧЕНИЕ

В результате выполнения лабораторной работы студентами был освоен маршрут проектирования компонентов аппаратного обеспечения, студенты овладели навыком проектирования и реализации конечных автоматов, а также работы с протоколом PS/2, кодировкой клавиш, работы с частотами, с однопакетным вводом и освоили механизм верификации проекта с использованием ПЛИС.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Дуксин, Н. А. Архитектура вычислительных машин и систем. Основы построения вычислительной техники: Практикум : учебное пособие / Н. А. Дуксин, Д. В. Люлява, И. Е. Тарасов. — Москва : РТУ МИРЭА, 2023. — 185 с.

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. Соловьев В. В. Основы языка проектирования цифровой аппаратуры Verilog. — М.: Горячая линия — Телеком, 2014. — 208 с.

4. Харрис Дэвид М., Харрис Сара Л. Цифровая схемотехника и архитектура компьютера. Издательство: ДМК-Пресс, 2018 г.

5. Максфилд К. Проектирование на ПЛИС. Курс молодого бойца. – М.: Издательский дом «Додэка-XXI», 2007. – 408 с.: илл. (Серия «Программируемые системы»)