|  |
| --- |
|  |
| 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  ИВБО-22-23 | Столба А. С. |
| 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 — 36.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc199324186)

[1 ХОД РАБОТЫ 6](#_Toc199324187)

[1.1 Постановка задачи 6](#_Toc199324188)

[1.2 Системная модель 6](#_Toc199324189)

[1.2.1 Алгоритм автомата 6](#_Toc199324190)

[1.2.2 Тестовое покрытие системной модели 6](#_Toc199324191)

[1.3 Описание автомата на Verilog HDL 8](#_Toc199324192)

[1.3.1 Этап тестирования 10](#_Toc199324193)

[1.4 Реализация основных модулей 11](#_Toc199324194)

[1.4.1 Реализация синхронизатора 11](#_Toc199324195)

[1.4.2 Реализация счётчика 11](#_Toc199324196)

[1.4.3 Реализация триггера 12](#_Toc199324197)

[1.4.4 Реализация делителя частоты 12](#_Toc199324198)

[1.4.5 Реализация модуля управления семисегментными индикаторами 13](#_Toc199324199)

[1.4.6 Реализация модуля фильтра дребезга контактов 14](#_Toc199324200)

[1.4.7 Реализация модуля дешифратора для PS/2 15](#_Toc199324201)

[1.4.8 Реализация модуля PS/2 16](#_Toc199324202)

[1.4.9 Реализация модуля управления для PS/2 18](#_Toc199324203)

[1.4.10 Реализация модуля верхнего уровня 19](#_Toc199324204)

[1.5 Создание и верификация тестового модуля верхнего уровня 23](#_Toc199324205)

[1.6 Создание файла проектных ограничений и загрузка проекта на отладочную плату nexys a7 31](#_Toc199324206)

[ЗАКЛЮЧЕНИЕ 35](#_Toc199324207)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 36](#_Toc199324208)

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 решения СЛАУ методом Крамера 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: Решение СЛАУ методом Крамера.[3]

## 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 элементы матрицы 3х3 и вектора В.
2. Вычислить определитель матрицы 3х3.
3. Если определитель = 0, то вывести код ошибки.
4. Вычислить вектор решения.
5. Вывести результат.

**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

Введите коэффициенты матрицы A (3x3):

A[0][0]: 1

A[0][1]: -1

A[0][2]: 1

A[1][0]: -1

Продолжение листинга 1.1

A[1][1]: 3

A[1][2]: -1

A[2][0]: 1

A[2][1]: -1

A[2][2]: 2

Введите свободные члены (вектор B):

B[0]: 1

B[1]: 1

B[2]: 2

Решение системы:

x = 1

y = 1

z = 1

Вектор решения: (1, 1, 1)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Введите коэффициенты матрицы A (3x3):

A[0][0]: 2

A[0][1]: 2

A[0][2]: 1

A[1][0]: 4

A[1][1]: 1

A[1][2]: 0

A[2][0]: 0

A[2][1]: 2

A[2][2]: 1

Введите свободные члены (вектор B):

B[0]: 1

B[1]: 0

B[2]: 1

Решение системы:

x = 0

y = 0

z = 1

Вектор решения: (0, 0, 1)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Введите коэффициенты матрицы A (3x3):

A[0][0]: 1

A[0][1]: 2

A[0][2]: 3

A[1][0]: 2

A[1][1]: 4

A[1][2]: 6

A[2][0]: 3

A[2][1]: 6

A[2][2]: 9

Введите свободные члены (вектор B):

B[0]: 1

B[1]: 1

B[2]: 1

Код ошибки 00, определитель равен 0.

## 1.3 Описание автомата на Verilog HDL

Исходный код представлен в Листинге 1.2.

Листинг 1.2 — Программная реализация автомата на Verilog HDL

`timescale 1ns / 1ps

module crimer(

input clk,

input reset,

input input\_en,

input signed [3:0] DataIn,

output reg signed [3:0] X1, X2, X3,

output reg valid\_cramer,

output reg failed,valid\_in,

output reg [2:0] state

);

reg signed [3:0] A[0:2][0:2];

reg signed [3:0] b[0:2];

reg [3:0] counter = 0;

reg signed [3:0] detA, detA1, detA2, detA3;

parameter wait\_start = 0;

parameter read\_data = 1;

parameter calculate = 2;

parameter data\_out = 3 initial

begin

state = wait\_start;

failed<=0;

end

always #5 if(valid\_in) valid\_in = ~valid\_in;

always @(posedge clk) begin

if(reset)

begin

failed <= 0;

valid\_cramer <= 0;

X1 <= 0;

X2 <= 0;

X3 <= 0;

b[0] <= 0;

b[1] <= 0;

b[2] <= 0;

A[0][0] <= 0;

A[0][1] <= 0;

A[0][2] <= 0;

A[1][0] <= 0;

A[1][1] <= 0;

A[1][2] <= 0;

A[2][0] <= 0;

A[2][1] <= 0;

A[2][2] <= 0;

detA<=0;

detA1<=0;

detA2<=0;

detA3<=0;

Продолжение листинга 1.2

counter <= 0;

end

if (input\_en) begin

if (counter < 9) begin

A[counter % 3][counter / 3] <= DataIn;

end

else if (counter < 12) begin

b[counter - 9] <= DataIn;

end

counter <= counter + 1;

valid\_in =1'd1;

end

if (counter == 12) begin

detA = (A[0][0] \* (A[1][1] \* A[2][2] - A[1][2] \* A[2][1]))

- (A[0][1] \* (A[1][0] \* A[2][2] - A[1][2] \* A[2][0]))

+ (A[0][2] \* (A[1][0] \* A[2][1] - A[1][1] \* A[2][0]));

detA1 = (b[0] \* (A[1][1] \* A[2][2] - A[1][2] \* A[2][1]))

- (A[0][1] \* (b[1] \* A[2][2] - A[1][2] \* b[2]))

+ (A[0][2] \* (b[1] \* A[2][1] - A[1][1] \* b[2]));

detA2 = (A[0][0] \* (b[1] \* A[2][2] - A[1][2] \* b[2]))

- (b[0] \* (A[1][0] \* A[2][2] - A[1][2] \* A[2][0]))

+ (A[0][2] \* (A[1][0] \* b[2] - b[1] \* A[2][0]));

detA3 = (A[0][0] \* (A[1][1] \* b[2] - b[1] \* A[2][1]))

- (A[0][1] \* (A[1][0] \* b[2] - b[1] \* A[2][0]))

+ (b[0] \* (A[1][0] \* A[2][1] - A[1][1] \* A[2][0]));for (i = 0; if (detA == 0) begin

failed <= 1;

valid\_cramer <= 0;

end

else begin

X1 <= detA1 / detA;

X2 <= detA2 / detA;

X3 <= detA3 / detA;

valid\_cramer <= 1;

failed <= 0;

end

end

case(state)

wait\_start:

if(input\_en)

state<=read\_data;

else

state<=state;

read\_data:

if(counter == 12)

state <= calculate;

else

state<=state;

calculate:

if(valid\_cramer)

state <= data\_out;

else

state <= state;

Продолжение листинга 1.2

data\_out:

if(reset)

state <= wait\_start;

else

state <= state;

default:

state <= wait\_start;

endcase

end

edmodule

**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 1.4. 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]

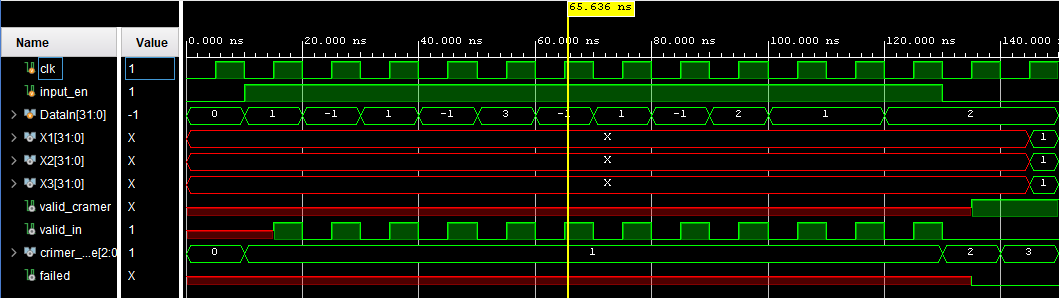


Рисунок 1.1 — Временная диаграмма работы устройства

## 1.4 Реализация основных модулей

**1.4.1 Реализация синхронизатора**

Опишем синхронизатор в модулеs 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 модулеs 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 модулеs 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 модулеs 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 модулеs 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,

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] MINUS\_CODE = 8'h4E;

parameter NUMBER\_F = 0, ENTER\_F = 1, BACKSPACE\_F = 2, RESET\_F = 3, MINUS\_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)

// ?????

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;

NUMBERS[12]: out = 4'hC;

NUMBERS[13]: out = 4'hD;

Продолжение листинга 1.9

NUMBERS[14]: out = 4'hE;

NUMBERS[15]: out = 4'hF;

// ??????? "Enter"

ENTER\_CODE: out = 0;

BACKSPACE\_CODE: out = 0;

RESET\_CODE: out = 0;

MINUS\_CODE: out = 0;

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;

MINUS\_CODE: flags <= 1 << MINUS\_F;

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,

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;

Продолжение листинга 1.10

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

state <= IDLE;

end

// Обработка стоп-бита

STOP\_BIT: begin

if (!PS2\_dat\_sync[1])

ERROR <= 1;

R\_O <= 1;

state <= WAIT\_START\_BIT;

Продолжение листинга 1.10

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

);

parameter WAIT\_ONE = 0;

parameter WAIT\_ZERO = 1;

reg state;

wire PS2\_R\_O, PS2\_ERROR;

wire [7:0] PS2\_out;

reg release\_flag;

initial begin

state = 0; R\_O = 0;

release\_flag = 0;

end

always@(posedge clk)

begin

case(state)

WAIT\_ONE: begin

if (PS2\_R\_O) begin

if (!PS2\_ERROR) begin

if (PS2\_out == 8'hF0)

release\_flag <= 1;

else if (release\_flag)

begin

R\_O <= 1;

release\_flag <= 0;

end

end

state <= WAIT\_ZERO;

end

end

WAIT\_ZERO: begin

R\_O <= 0;

Продолжение листинга 1.11

if (!PS2\_R\_O)

state <= WAIT\_ONE;

end

endcase

end

PS2 ps2(

.clk(clk),

.PS2\_clk(PS2\_clk), .PS2\_dat(PS2\_dat),

.out(PS2\_out), .R\_O(PS2\_R\_O),

.ERROR(PS2\_ERROR));

PS2\_DC dc(.keycode(PS2\_out), .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,button\_minus,

output [7:0] AN,

output [6:0] SEG,

output reg is\_minus\_last,

output wire is\_failed,

input clk,

input PS2\_clk,

input PS2\_dat

//output wire reset\_signal,

//output reg signed [3:0] Data\_in,

//output wire button\_signal\_en,

// output wire reset\_signal\_en,

//output wire button\_minus\_en,

//output wire minus\_signal

//output reg [31:0] NUMBER,

// output wire button\_signal

);

wire button\_signal, button\_signal\_en, reset\_signal\_en, reset\_signal;

reg[7:0] AN\_MASK = 8'b11111111;

reg [31:0] NUMBER;

wire minus\_signal;

wire button\_minus\_en;

reg signed [3:0] Data\_in;

reg is\_data\_last;

reg is\_minus = 0;

wire [3:0] X1, X2, X3;

wire valid\_cramer;

parameter minus = -1;

reg input\_en = 0;

reg [2:0] n = 0;

reg [3:0] n1 = 0;

reg [3:0] data\_last = 0;

reg reset = 0;

Продолжение листинга 1.12

wire [3:0] out;

wire [4:0] flags;

wire R\_O;

initial begin

NUMBER <= 0;

end

// filtercon #(128) dbnc\_minus(

// .clk(clk),

// .in\_signal(button\_minus),

// .clock\_enable(1'b1),

// .out\_signal(minus\_singal),

// .out\_signal\_enable(button\_minus\_en));

// 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)

// );

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)

);

crimer aut(

.clk(clk),

.reset(reset),

.input\_en(input\_en),

.DataIn(Data\_in),

.X1(X1),

.X2(X2),

Продолжение листинга 1.12

.X3(X3),

.valid\_cramer(valid\_cramer),

.failed(is\_failed),

.valid\_in(),

.state());

reg [3:0] current\_number;

always@(posedge clk)

begin

if(reset)

reset = ~reset;

if(is\_failed)

begin

NUMBER = 0;

AN\_MASK <= 8'b00000000;

end

else

begin

if(R\_O) begin

if (flags==1)

begin

current\_number <= out;

end

else if(flags == 5'b10000)

is\_minus <= ~is\_minus;

else if (flags==8)

begin

reset <= 1;

Data\_in <= 0;

NUMBER <= 0;

n<=0;

is\_data\_last <= 0;

AN\_MASK <= 8'b11111111;

input\_en<=0;

end

else if (flags==2)

begin

if(is\_minus) begin

Data\_in <= minus \* current\_number;

is\_minus\_last <= 1;

is\_data\_last <= 1;

data\_last<=current\_number;

end

else begin

Data\_in <= current\_number;

is\_minus\_last <= 0;

is\_data\_last <= 1;

data\_last<=current\_number;

end

input\_en<=1;

if(valid\_cramer) begin

is\_data\_last <= 0;

n <= NEXT(n);

end

end

end

if(input\_en)

input\_en<=~input\_en;

if(is\_data\_last == 1) begin

AN\_MASK<=~8'b00000001;

NUMBER <= data\_last;

end

Продолжение листинга 1.12

else if(valid\_cramer)

begin

case(n)

0: begin NUMBER<=0; NUMBER<=0; AN\_MASK<=~8'b00000000;end

1: begin NUMBER<=0; NUMBER<=X1; AN\_MASK<=~8'b00000001;n1=0;end

2: begin NUMBER<=0; NUMBER<=X2; AN\_MASK<=~8'b00000001;end

3: begin NUMBER<=0; NUMBER<=X3; AN\_MASK<=~8'b00000001;end

4: begin

if(n1==0) begin

NUMBER<=0;

n1 <= n1+1; end

else if(n1<6)

n1 <= n1+1;

else if(n1)

AN\_MASK<=~8'b00010101;

case(n1)

1: NUMBER <= {NUMBER[27:0], X1};

2: NUMBER <= {NUMBER[27:0], 4'b0000};

3: NUMBER <= {NUMBER[27:0], X2};

4: NUMBER <= {NUMBER[27:0], 4'b0000};

5: NUMBER <= {NUMBER[27:0], X3};

endcase

end

endcase

end

end

end

function [2:0] NEXT ( input [2:0] num);

begin

if(num == 4)

NEXT = 1;

else

NEXT = num + 1;

end

endfunction

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 testPs();

parameter ENTER\_CODE = 8'h5A;

parameter STOP\_CODE = 8'hF0;

parameter BACKSPACE\_CODE = 8'h66;

parameter RESET\_CODE = 8'h2D;

parameter MINUS\_CODE = 8'h4E;

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'h1);

PS2\_press\_and\_release\_key(key\_code);

#(2\*clk\_period) key\_code = ENTER\_CODE;

PS2\_press\_and\_release\_key(key\_code);

#5000;

#(2\*clk\_period) key\_code = MINUS\_CODE;

PS2\_press\_and\_release\_key(key\_code);

#(2\*clk\_period) key\_code = HEX\_CD(4'h1);

PS2\_press\_and\_release\_key(key\_code);

Продолжение листинга 1.13

#(2\*clk\_period) key\_code = ENTER\_CODE;

PS2\_press\_and\_release\_key(key\_code);

#5000;

#(2\*clk\_period) key\_code = MINUS\_CODE;

PS2\_press\_and\_release\_key(key\_code);

#(2\*clk\_period) key\_code = HEX\_CD(4'h1);

PS2\_press\_and\_release\_key(key\_code);

#(2\*clk\_period) key\_code = ENTER\_CODE;

PS2\_press\_and\_release\_key(key\_code);

#5000;

#(2\*clk\_period) key\_code = MINUS\_CODE;

PS2\_press\_and\_release\_key(key\_code);

#(2\*clk\_period) key\_code = HEX\_CD(4'h1);

PS2\_press\_and\_release\_key(key\_code);

#(2\*clk\_period) key\_code = ENTER\_CODE;

PS2\_press\_and\_release\_key(key\_code);

#5000;

#(2\*clk\_period) key\_code = MINUS\_CODE;

PS2\_press\_and\_release\_key(key\_code);

#(2\*clk\_period) key\_code = HEX\_CD(4'h3);

PS2\_press\_and\_release\_key(key\_code);

#(2\*clk\_period) key\_code = ENTER\_CODE;

PS2\_press\_and\_release\_key(key\_code);

#5000;

#(2\*clk\_period) key\_code = MINUS\_CODE;

PS2\_press\_and\_release\_key(key\_code);

#(2\*clk\_period) key\_code = HEX\_CD(4'h1);

PS2\_press\_and\_release\_key(key\_code);

#(2\*clk\_period) key\_code = ENTER\_CODE;

PS2\_press\_and\_release\_key(key\_code);

#5000;

#(2\*clk\_period) key\_code = MINUS\_CODE;

PS2\_press\_and\_release\_key(key\_code);

#(2\*clk\_period) key\_code = HEX\_CD(4'h1);

PS2\_press\_and\_release\_key(key\_code);

#(2\*clk\_period) key\_code = ENTER\_CODE;

PS2\_press\_and\_release\_key(key\_code);

#5000;

#(2\*clk\_period) key\_code = MINUS\_CODE;

PS2\_press\_and\_release\_key(key\_code);

#(2\*clk\_period) key\_code = HEX\_CD(4'h1);

PS2\_press\_and\_release\_key(key\_code);

#(2\*clk\_period) key\_code = ENTER\_CODE;

PS2\_press\_and\_release\_key(key\_code);

#5000;

#(2\*clk\_period) key\_code = MINUS\_CODE;

PS2\_press\_and\_release\_key(key\_code);

#(2\*clk\_period) key\_code = HEX\_CD(4'h2);

PS2\_press\_and\_release\_key(key\_code);

#(2\*clk\_period) key\_code = ENTER\_CODE;

PS2\_press\_and\_release\_key(key\_code);

#5000;

#(2\*clk\_period) key\_code = HEX\_CD(4'h1);

PS2\_press\_and\_release\_key(key\_code);

#(2\*clk\_period) key\_code = ENTER\_CODE;

Продолжение листинга 1.13

PS2\_press\_and\_release\_key(key\_code);

#5000;

#(2\*clk\_period) key\_code = HEX\_CD(4'h1);

PS2\_press\_and\_release\_key(key\_code);

#(2\*clk\_period) key\_code = ENTER\_CODE;

PS2\_press\_and\_release\_key(key\_code);

#5000;

#(2\*clk\_period) key\_code = HEX\_CD(4'h2);

PS2\_press\_and\_release\_key(key\_code);

#(2\*clk\_period) key\_code = ENTER\_CODE;

PS2\_press\_and\_release\_key(key\_code);

#5000;

#90000;

#(2\*clk\_period) key\_code = RESET\_CODE;

PS2\_press\_and\_release\_key(key\_code);

#5000;

PS2\_press\_and\_release\_extended\_key(8'h71);

#5000;

// #(2\*clk\_period) key\_code = HEX\_CD(4'hF);

// PS2\_press\_and\_release\_key(key\_code);

// #(2\*clk\_period) key\_code = ENTER\_CODE;

// PS2\_press\_and\_release\_key(key\_code);

// #(2\*clk\_period) key\_code = BACKSPACE\_CODE;

// PS2\_press\_and\_release\_key(key\_code);

// #(2\*clk\_period) key\_code = RESET\_CODE;

// PS2\_press\_and\_release\_key(key\_code);

// #(2\*clk\_period) key\_code = HEX\_CD(1);

// PS2\_press\_and\_release\_key(key\_code);

// #(2\*clk\_period) key\_code = HEX\_CD(4'hF);

// PS2\_press\_and\_release\_key(key\_code);

// #(2\*clk\_period) key\_code = ENTER\_CODE;

// PS2\_press\_and\_release\_key(key\_code);

#(2\*clk\_period)

$finish;

end

task automatic PS2\_press\_and\_release\_extended\_key(

input [7:0] code

);

begin

fork

PS2\_gen\_byte\_clk();

PS2\_code\_input(8'hE0);

join

fork

PS2\_gen\_byte\_clk();

PS2\_code\_input(code);

join

#code\_space\_period;

// ??????????

fork

PS2\_gen\_byte\_clk();

PS2\_code\_input(8'hE0);

join

fork

PS2\_gen\_byte\_clk();

PS2\_code\_input(8'hF0);

join

fork

PS2

Продолжение листинга 1.13

PS2\_press\_and\_release\_key(key\_code);

#5000;

#(2\*clk\_period) key\_code = HEX\_CD(4'h1);

PS2\_press\_and\_release\_key(key\_code);

#(2\*clk\_period) key\_code = ENTER\_CODE;

PS2\_press\_and\_release\_key(key\_code);

#5000;

#(2\*clk\_period) key\_code = HEX\_CD(4'h2);

PS2\_press\_and\_release\_key(key\_code);

#(2\*clk\_period) key\_code = ENTER\_CODE;

PS2\_press\_and\_release\_key(key\_code);

#5000;

#90000;

#(2\*clk\_period) key\_code = RESET\_CODE;

PS2\_press\_and\_release\_key(key\_code);

#5000;

PS2\_press\_and\_release\_extended\_key(8'h71);

#5000;

// #(2\*clk\_period) key\_code = HEX\_CD(4'hF);

// PS2\_press\_and\_release\_key(key\_code);

// #(2\*clk\_period) key\_code = ENTER\_CODE;

// PS2\_press\_and\_release\_key(key\_code);

// #(2\*clk\_period) key\_code = BACKSPACE\_CODE;

// PS2\_press\_and\_release\_key(key\_code);

// #(2\*clk\_period) key\_code = RESET\_CODE;

// PS2\_press\_and\_release\_key(key\_code);

// #(2\*clk\_period) key\_code = HEX\_CD(1);

// PS2\_press\_and\_release\_key(key\_code);

// #(2\*clk\_period) key\_code = HEX\_CD(4'hF);

// PS2\_press\_and\_release\_key(key\_code);

// #(2\*clk\_period) key\_code = ENTER\_CODE;

// PS2\_press\_and\_release\_key(key\_code);

#(2\*clk\_period)

$finish;

end

task automatic PS2\_press\_and\_release\_extended\_key(

input [7:0] code

);

begin

fork

PS2\_gen\_byte\_clk();

PS2\_code\_input(8'hE0);

join

fork

PS2\_gen\_byte\_clk();

PS2\_code\_input(code);

join

#code\_space\_period;

fork

PS2\_gen\_byte\_clk();

PS2\_code\_input(8'hE0);

join

fork

PS2\_gen\_byte\_clk();

PS2\_code\_input(8'hF0);

join

fork

Продолжение листинга 1.13

PS2\_gen\_byte\_clk();

PS2\_code\_input(code);

join

#code\_space\_period;

end

endtask

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

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

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;

Продолжение листинга 1.13

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;

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 и второго тестов

На Рисунке 1.3 представлена временная диаграмма для второго теста.

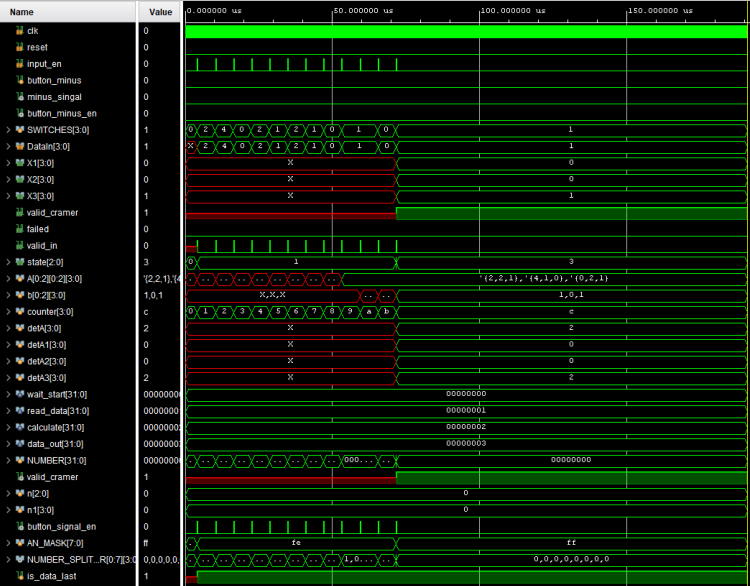


Рисунок 1.3 — Временная диаграмма второго

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]}]

Продолжение листинга 1.13

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 значения 1

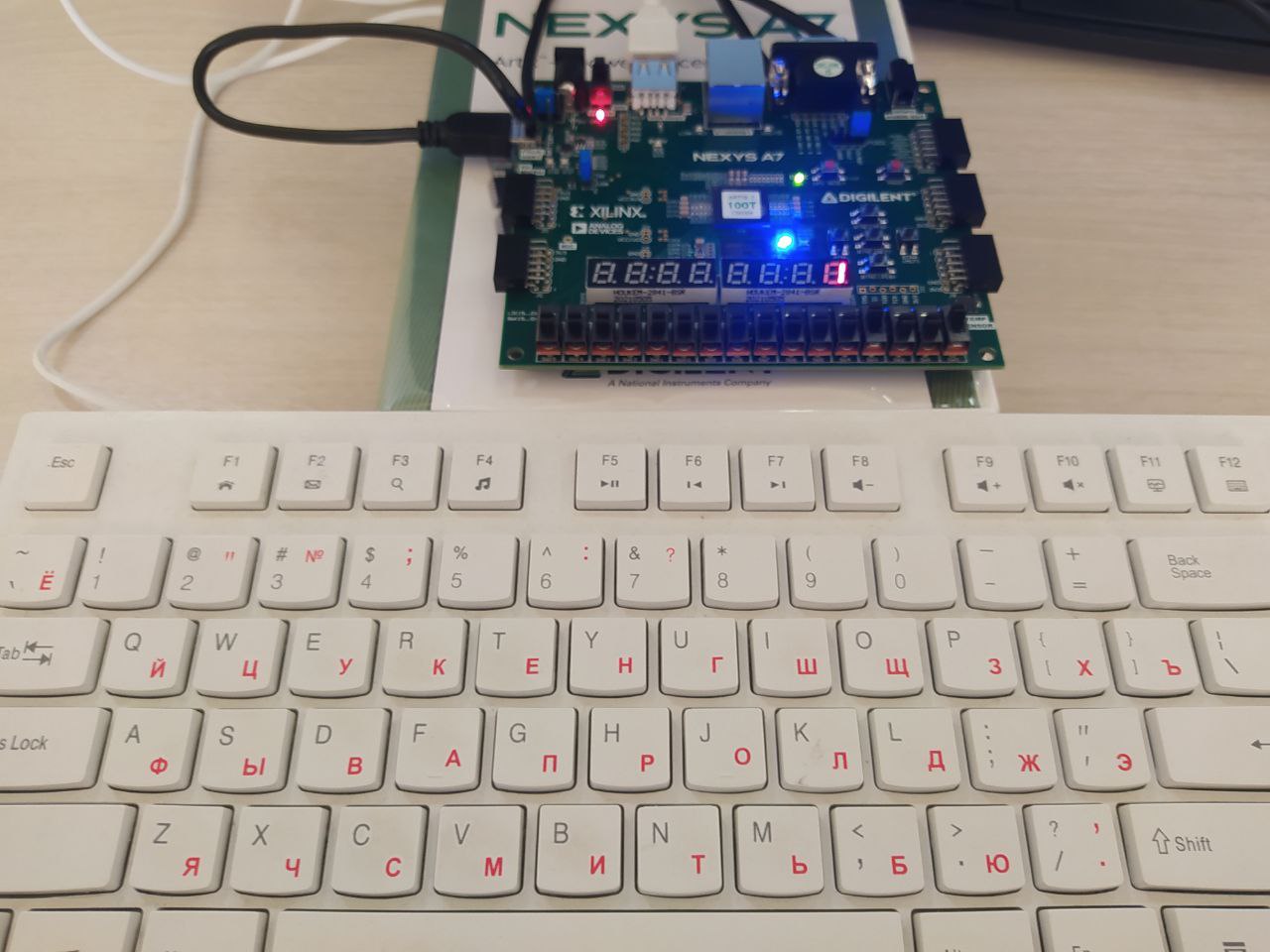


Рисунок 1.6 — Запись значения -1



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»)