**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

(СПбГУТ)

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ **(ИТПИ)**

Кафедра программной инженерии и вычислительной техники **(ПИиВТ)**

Дисциплина: «Архитектура распределенных вычислительных систем»

Лабораторная работа №6.

Вариант 7.

**Тема: «Программное проектирование конечного автомата»**

Выполнили:

Студенты группы ИКПИ-23

Даненко Д.А.

Луценко Д.А.

Харлова А.А

Подпись \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Принял:

ст. преподаватель кафедры ПИиВТ

Неелова О.Л.

Подпись \_\_\_\_\_\_\_\_\_\_\_\_\_\_

2024 г.

# **Цель работы:**

# Изучение методов программного проектирования конечных автоматов по выданному заданию на проектирование (Автомат для прохода в метро). Определение количества состояний автомата, условий перехода и действий внутри состояния.

# **Ход работы:**

# Автомат для прохода в метро имеет следующие состояния:

**S0 – начальное состояние.**

**S1 – ожидание карты.**

Индикация «\_».

Приложенная карта включает счетчик тактов.

Сигнал, имитирующий это действие, получаем с тумблера Т1.

Через 2 такта автомат переходит в состояние S2 – чтение состояния карты.

**S2 – чтение карты.**

Время удержания 2 такта.

Состояние карты имитируется положением тумблеров, определяющих баланс карты.

Состояние выхода автомата «\_\_».

Переход в S3.

**S3 – индикация состояния карты.**

Автомат выводит в течение 3 тактов состояние остатка на балансе при вычете тарифа.

После этого автомат переходит в состояние S4 или S1(если баланс меньше тарифа).

**S4 – вход разрешён.**

Тумблер Т1 выключить.

Состояние выхода GO (турникет открыт).

Время удержания 4 такта. Переход в S1.

Диаграмма состояний конечного автомата:

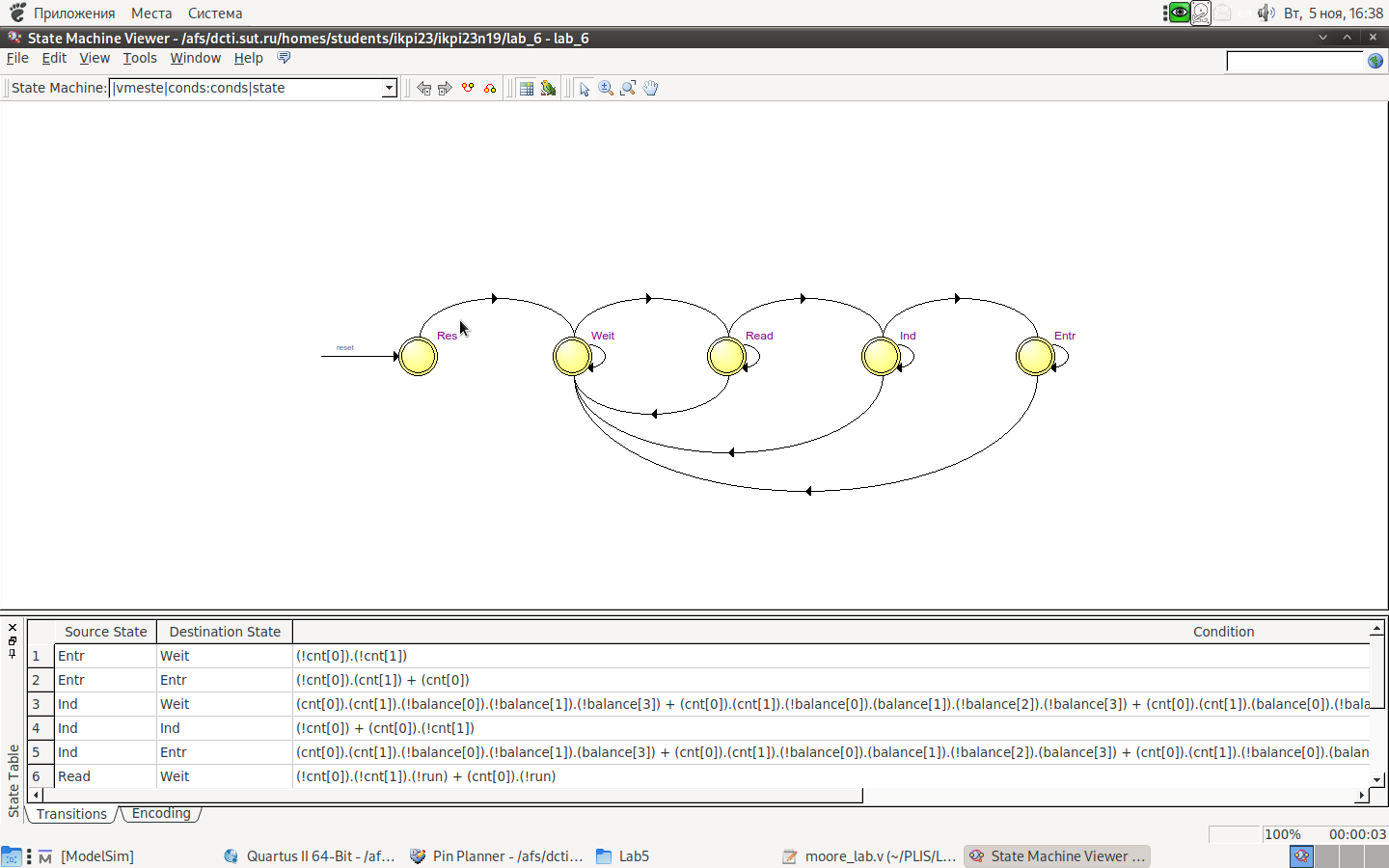
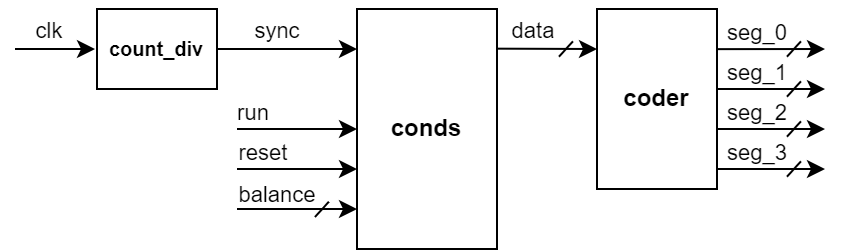


Схема результирующего устройства (модуль vmeste.v):



Описание использованных блоков:

**count\_div** - модуль счётчика понижения частоты.

Входы: clk - вход высокой частоты; выходы: sync - выход низкой частоты.

**conds** - модуль, реализующий конечный автомат.

Входы: clk - тактовый сигнал, run - флаг ожидания ввода, reset - вход сброса, balance - ввод баланса; Выходы: out - информация для вывода на индикаторах.

**coder** - модуль кодировки информационных сообщений.

Входы: data - код сообщения; выходы: seg\_0, seg\_1, seg\_2, seg\_3 - выходы подключения семисегментных индикаторов, формирующих надпись и числа.

Коды использованных модулей на языке Verilog:

**Модуль 'count\_div.v':**

module count\_div

#(parameter N=29,M=29'd100000000)

//#(parameter N=2,M=2'd2)

(input wire clk,

output reg sync=0);

reg [N-1:0]cnt=0;

wire [N-1:0]cnt\_next;

assign cnt\_next=cnt+1'b1;

always@(posedge clk)

begin

cnt<=cnt\_next;

sync<=sync;

if (cnt==M-1)

begin

cnt<=0;

sync<=~sync;

end

end

endmodule

**Модуль 'conds.v':**

module conds

(input clk,

input reset,

input run,

input [3:0] balance,

output reg [5:0] out);

reg [2:0] state;

reg [1:0] cnt;

parameter Res = 0,

Weit = 1,

Read = 2,

Ind = 3,

Entr = 4,

Fin = 5;

always @ (posedge clk or posedge reset)

begin

if (reset) state <= Res;

else

begin case (state)

Res:

state <= Weit;

Weit:

if (cnt==2'd2) state <= Read; //- \* 2

Read: begin

if (!run) state <= Weit;

if (cnt==2'd2) state <= Ind; // Read \* 2

end

Ind:

if (cnt==2'd3) // (balance - 4) \* 3

if (balance > 3'd4) state <= Entr;

else state <= Weit;

Entr:

if (cnt==2'd4) state <= Weit; // Go \* 4

default:

state <= Res;

endcase

end

end

always @ (posedge clk)

begin

case (state)

Res:

begin

cnt <= 2'd0;

out <= 3'b000;

end

Weit:

begin

if (cnt==2'd2) cnt <= 2'd0;

else

cnt <= cnt + 2'd1;

out <= 6'b010000; // \_

end

Read:

begin

if (cnt==2'd2) cnt <= 2'd0;

else

cnt <= cnt + 2'd1;

out <= 6'b100000; // read

end

Ind:

begin

if (cnt==2'd3) cnt <= 2'd0;

else

cnt <= cnt + 2'd1;

if (balance > 3'd4) begin

out <= 6'd0;

out <= balance - 3'd4; // balance - 4

end

else out <= 6'd0;

end

Entr:

begin

if (cnt==2'd4) cnt <= 2'd0;

else

cnt <= cnt + 2'd1;

out <= 6'b110000; // Go \* 4

end

endcase

end

Endmodule

**Модуль 'coder.v':**

module coder

(input wire [5:0] data,

output wire [6:0] seg\_0,

output wire [6:0] seg\_1,

output wire [6:0] seg\_2,

output wire [6:0] seg\_3);

reg [6:0] out\_0,

out\_1,

out\_2,

out\_3;

assign seg\_0=out\_0;

assign seg\_1=out\_1;

assign seg\_2=out\_2;

assign seg\_3=out\_3;

always @\*

case(data)

6'b010000: begin // \_

out\_0 = 7'b1111111;

out\_1 = 7'b1111111;

out\_2 = 7'b1111111;

out\_3 = 7'b1110111;

end

6'b100000: begin // read

out\_0 = 7'b1001100;

out\_1 = 7'b0000110;

out\_2 = 7'b0001000;

out\_3 = 7'b0100001;

end

6'b110000: begin // go

out\_0 = 7'b1111111;

out\_1 = 7'b1111111;

out\_2 = 7'b1000010;

out\_3 = 7'b0100011;

end

6'b000001: begin // 1

out\_0 = 7'b1111111;

out\_1 = 7'b1111111;

out\_2 = 7'b1111111;

out\_3 = 7'b1001111;

end

6'b000010: begin // 2

out\_0 = 7'b1111111;

out\_1 = 7'b1111111;

out\_2 = 7'b1111111;

out\_3 = 7'b0010010;

end

6'b000011: begin // 3

out\_0 = 7'b1111111;

out\_1 = 7'b1111111;

out\_2 = 7'b1111111;

out\_3 = 7'b0000110;

end

6'b000100: begin // 4

out\_0 = 7'b1111111;

out\_1 = 7'b1111111;

out\_2 = 7'b1111111;

out\_3 = 7'b1001100;

end

6'b000101: begin // 5

out\_0 = 7'b1111111;

out\_1 = 7'b1111111;

out\_2 = 7'b1111111;

out\_3 = 7'b0100100;

end

6'b000110: begin // 6

out\_0 = 7'b1111111;

out\_1 = 7'b1111111;

out\_2 = 7'b1111111;

out\_3 = 7'b0000010;

end

6'b000111: begin // 7

out\_0 = 7'b1111111;

out\_1 = 7'b1111111;

out\_2 = 7'b1111111;

out\_3 = 7'b1111000;

end

6'b001000: begin // 8

out\_0 = 7'b1111111;

out\_1 = 7'b1111111;

out\_2 = 7'b1111111;

out\_3 = 7'b0000000;

end

6'b001001: begin // 9

out\_0 = 7'b1111111;

out\_1 = 7'b1111111;

out\_2 = 7'b1111111;

out\_3 = 7'b0010000;

end

6'b001010: begin // 10

out\_0 = 7'b1111111;

out\_1 = 7'b1111111;

out\_2 = 7'b1001111;

out\_3 = 7'b1000000;

end

6'b001011: begin // 11

out\_0 = 7'b1111111;

out\_1 = 7'b1111111;

out\_2 = 7'b1001111;

out\_3 = 7'b1001111;

end

6'b001100: begin // 12

out\_0 = 7'b1111111;

out\_1 = 7'b1111111;

out\_2 = 7'b1001111;

out\_3 = 7'b0010010;

end

6'b001101: begin // 13

out\_0 = 7'b1111111;

out\_1 = 7'b1111111;

out\_2 = 7'b1001111;

out\_3 = 7'b0000110;

end

6'b001110: begin // 14

out\_0 = 7'b1111111;

out\_1 = 7'b1111111;

out\_2 = 7'b1001111;

out\_3 = 7'b1001100;

end

6'b001111: begin // 15

out\_0 = 7'b1111111;

out\_1 = 7'b1111111;

out\_2 = 7'b1001111;

out\_3 = 7'b0100100;

end

endcase

endmodule

**Модуль 'vmeste.v':**

module vmeste

(input clk\_all, reset\_all, run\_all,

input [3:0] balance\_all,

output [6:0] seg\_0,

output [6:0] seg\_1,

output [6:0] seg\_2,

output [6:0] seg\_3);

wire clk\_div;

wire [5:0] out\_all;

count\_div count\_div (.clk(clk\_all),

.sync(clk\_div));

conds conds (.clk(clk\_div),

.run(run\_all),

.reset(reset\_all),

.balance(balance\_all),

.out(out\_all));

coder coder (.data(out\_all),

.seg\_0(seg\_0),

.seg\_1(seg\_1),

.seg\_2(seg\_2),

.seg\_3(seg\_3));

endmodule