# ПРИЛОЖЕНИЯ

## Приложение 1

…\Control-Unit.sv

module Control\_Unit

(

input logic button,

input logic reset,

input logic clock,

input logic switch,

output logic [6:0] ssegment0,

output logic [6:0] ssegment1,

output logic [6:0] ssegment2,

output logic [6:0] ssegment4,

output logic [6:0] ssegment5,

output logic [6:0] ssegment6

);

logic [6:0] counter\_settings;

logic work\_button;

logic [6:0] memory;

antidrebezg button0

(

.clock (clock),

.reset (reset),

.button (~button),

.pulse (work\_button)

);

always\_ff @(posedge clock)

begin

if (reset == 1) counter\_settings <= 0;

else

begin

if (work\_button == 1)

begin

if (counter\_settings < 16) counter\_settings <= counter\_settings + 1;

else counter\_settings <= 0;

end

else counter\_settings <= counter\_settings;

end

end

SEG7counter display

(

.clock (clock),

.reset (reset),

.counter\_settings (counter\_settings),

.memory (memory),

.ssegment1 (ssegment0),

.ssegment2 (ssegment1),

.ssegment3 (ssegment2),

.ssegment5 (ssegment4),

.ssegment6 (ssegment5),

.ssegment7 (ssegment6)

);

logic v1, v2, v3, v4, v5, v6, v7, v8, v9, v10, v11, v12, v13, v14, v15, v16, v17, v18, v19, v20, v21, v22, v23, v24, v25, v26, v27, v28, v29, v30, v31, v32, v33, v34, v35, v36, v37, v38, v39;

logic [4:0] MK\_Adress;

logic [7:0] Reg\_1;

logic [7:0] Reg\_2;

logic [15:0] Reg\_SUM;

logic [7:0] Reg\_A;

logic [7:0] Reg\_B;

logic [1:0] Reg\_d;

logic Reg\_sd;

logic Tzn;

logic [38:0] MK;

logic [3:0]counter;

logic [3:0]counter\_cikl;

logic [7:0] Reg\_Com;

logic [7:0] Reg\_ReadMem;

logic [7:0] Reg\_AdressMem;

logic condition1, condition2, condition3, condition4, condition5, condition6;

logic [7:0] mem [16] = '{

{8'b01110111},

{8'b01100101},

{8'b01010110},

{8'b01000100},

{8'b00000010},

{8'b00001010},

{8'b01100100},

{8'b01111011},

{8'b00000010},

{8'b00000000},

{8'b00000111},

{8'b00000000},

{8'b00011010},

{8'b00000000},

{8'b00100111},

{8'b00000000}};

logic [7:0] Reg\_buf;

logic [38:0] mem\_MK [24] = '{

{39'b100000000000000100010010000000001000000},

{39'b011000000000000000000000000000000000000},

{39'b000010100000000000000000000000000000000},

{39'b001000000000000000000000000000000000000},

{39'b000001001000000000000000000000000000000},

{39'b001000000000000000000000000000010000000},

{39'b000000010000100000000000000001000100000},

{39'b000000000001000000000000000000000001100},

{39'b000000000000000000000000001000000000000},

{39'b000000000100000000000000000000000000000},

{39'b000000000000010000000000000000000000000},

{39'b000000000000000000000000000000000000001},

{39'b000000000000000000000001000000000000000},

{39'b000000000000000000000001000000000000000},

{39'b000000000010000000000000000000000000000},

{39'b000000000000000000000100000000000000000},

{39'b000000000000000000000001000000000000000},

{39'b000000000000000011000000100000000000000},

{39'b000000000000000000100000000000000010000},

{39'b000000000000000000000000000000000000010},

{39'b000000000000000000000000000100000000000},

{39'b000100000000000000001000000000000000000},

{39'b000000000000000000000000010000000000000},

{39'b000100000000000000000000000000000000000}};

logic [4:0] mem\_next\_MK\_Adress [24] = '{

{5'b00001},

{5'b00010},

{5'b00011},

{5'b00100},

{5'b00101},

{5'b00110},

{5'b00111},

{5'b01111},

{5'b01011},

{5'b01010},

{5'b01011},

{5'b01100},

{5'b10001},

{5'b01110},

{5'b10001},

{5'b10000},

{5'b10001},

{5'b10011},

{5'b00111},

{5'b10100},

{5'b10101},

{5'b10110},

{5'b10111},

{5'b00000}};

logic start;

logic work\_en;

always\_comb

begin

MK = mem\_MK[MK\_Adress];

end

always\_comb

begin

v1 = MK[0];

v2 = MK[1];

v3 = MK[2];

v4 = MK[3];

v5 = MK[4];

v6 = MK[5];

v7 = MK[6];

v8 = MK[7];

v9 = MK[8];

v10 = MK[9];

v11 = MK[10];

v12 = MK[11];

v13 = MK[12];

v14 = MK[13];

v15 = MK[14];

v16 = MK[15];

v17 = MK[16];

v18 = MK[17];

v19 = MK[18];

v20 = MK[19];

v21 = MK[20];

v22 = MK[21];

v23 = MK[22];

v24 = MK[23];

v25 = MK[24];

v26 = MK[25];

v27 = MK[26];

v28 = MK[27];

v29 = MK[28];

v30 = MK[29];

v31 = MK[30];

v32 = MK[31];

v33 = MK[32];

v34 = MK[33];

v35 = MK[34];

v36 = MK[35];

v37 = MK[36];

v38 = MK[37];

v39 = MK[38];

end

always\_ff @(posedge clock)

begin

if (reset == 1) memory <= 0;

else memory <= mem[counter\_settings][6:0];

end

always\_ff @(posedge clock)

begin

if (reset == 1) start <= 0;

else

begin

if (switch == 1) start <= 1;

else start <= 0;

end

end

always\_ff @(posedge clock)

begin

if (reset == 1) work\_en <=0;

else

begin

if (counter < 5) work\_en <= 1;

else work\_en <= 0;

end

end

always\_ff @(posedge ~clock)

begin

if (reset == 1)

begin

Reg\_AdressMem <= 0;

end

else

begin

if (start == 1)

begin

if (work\_en == 1)

begin

if (v39 == 1) Reg\_AdressMem = counter;

if (v34 == 1) Reg\_AdressMem = Reg\_Com[6:3];

if (v19 == 1) Reg\_AdressMem = Reg\_buf;

if (v33 == 1) Reg\_AdressMem = Reg\_Com[2:0];

end

end

end

end

always\_ff @(posedge clock)

begin

if (reset == 1) counter = 0;

else

begin

if (start == 1)

begin

if (work\_en == 1)

begin

if (v38 == 1) counter = counter + 1;

end

end

end

end

always\_ff @(posedge clock)

begin

if (reset == 1) Reg\_ReadMem = 0;

else

begin

if (start == 1)

begin

if (work\_en == 1)

begin

if (v37 == 1) Reg\_ReadMem = mem[Reg\_AdressMem][7:0];

if (v14 == 1) Reg\_ReadMem = Reg\_SUM[7:0];

if (v12 == 1) Reg\_ReadMem = Reg\_SUM[15:8];

end

end

end

end

always\_ff @(posedge clock)

begin

if (start == 1)

begin

if (work\_en == 1)

begin

if (v36 == 1) mem[Reg\_AdressMem][7:0] = Reg\_ReadMem;

end

end

end

always\_ff @(posedge clock)

begin

if (reset == 1) Reg\_Com = 0;

else

begin

if (start == 1)

begin

if (work\_en == 1)

begin

if (v35 == 1) Reg\_Com = Reg\_ReadMem;

end

end

end

end

always\_ff @(posedge clock)

begin

if (reset == 1) Reg\_2 = 0;

else

begin

if (start == 1)

begin

if (work\_en == 1)

begin

if (v31 == 1) Reg\_2 = Reg\_ReadMem;

if (v21 == 1) Reg\_2 = Reg\_2 >> 1;

if (v3 == 1) Reg\_2[7] = 0;

end

end

end

end

always\_ff @(posedge clock)

begin

if (reset == 1) Reg\_B = 0;

else

begin

if (start == 1)

begin

if (work\_en == 1)

begin

if (v30 == 1) Reg\_B[7] = 1;

if (v28 == 1)

begin

Reg\_B[7] = ~Reg\_1[7];

Reg\_B[6] = ~Reg\_1[6];

Reg\_B[5] = ~Reg\_1[5];

Reg\_B[4] = ~Reg\_1[4];

Reg\_B[3] = ~Reg\_1[3];

Reg\_B[2] = ~Reg\_1[2];

Reg\_B[1] = ~Reg\_1[1];

Reg\_B[0] = ~Reg\_1[0];

end

if (v26 == 1) Reg\_B = Reg\_B + 1;

if (v25 == 1) Reg\_B = ~Reg\_2;

if (v13 == 1) Reg\_B = Reg\_1;

if (v9 == 1) Reg\_B = Reg\_2;

end

end

end

end

always\_ff @(posedge clock)

begin

if (reset == 1) Tzn = 0;

else

begin

if (start == 1)

begin

if (work\_en == 1)

begin

if (v27 == 1) Tzn = Reg\_1[7] ^ Reg\_2[7];

end

end

end

end

always\_ff @(posedge clock)

begin

if (reset == 1) Reg\_d = 0;

else

begin

if (start == 1)

begin

if (work\_en == 1)

begin

Reg\_d[1] = (Reg\_2[0] ^ Reg\_sd) & ~Reg\_d[0];

if (v24 == 1) Reg\_d = 0;

if (v23 == 1) Reg\_d = Reg\_d >> 1;

end

end

end

end

always\_ff @(posedge clock)

begin

if (reset == 1) Reg\_SUM = 0;

else

begin

if (start == 1)

begin

if (work\_en == 1)

begin

if (v17 == 1) Reg\_SUM = 0;

if (v16 == 1) Reg\_SUM = Reg\_SUM >> 1;

if (v2 == 1) Reg\_SUM[15] = Tzn;

if (v18 == 1) Reg\_SUM[15:8] = Reg\_A;

if (v29 == 1) Reg\_SUM[15] = 1;

if (v1 == 1) Reg\_SUM[15:8] = Reg\_A + Reg\_B;

end

end

end

end

always\_ff @(posedge clock)

begin

if (reset == 1) Reg\_A = 0;

else

begin

if (start == 1)

begin

if (work\_en == 1)

begin

if (v15 == 1) Reg\_A = Reg\_SUM[15:8];

if (v11 == 1) Reg\_A = Reg\_1;

if (v20 == 1) Reg\_A = 0;

end

end

end

end

always\_ff @(posedge clock)

begin

if (reset == 1) Reg\_buf = 0;

else

begin

if (start == 1)

begin

if (work\_en == 1)

begin

if (v10 == 1) Reg\_buf = Reg\_buf + 1;

if (v8 == 1) Reg\_buf = Reg\_AdressMem;

end

end

end

end

always\_ff @(posedge clock)

begin

if (reset == 1) Reg\_sd = 0;

else

begin

if (start == 1)

begin

if (work\_en == 1)

begin

if (v7 == 1) Reg\_sd = 0;

if (v22 == 1) Reg\_sd = Reg\_2[0];

end

end

end

end

always\_ff @(posedge clock)

begin

if (reset == 1) counter\_cikl = 0;

else

begin

if (start == 1)

begin

if (work\_en == 1)

begin

if (v6 == 1) counter\_cikl = 7;

if (v5 == 1) counter\_cikl = counter\_cikl - 1;

end

end

end

end

always\_ff @(posedge clock)

begin

if (reset == 1) Reg\_1 = 0;

else

begin

if (start == 1)

begin

if (work\_en == 1)

begin

if (v4 == 1) Reg\_1[7] = 0;

if (v32 == 1) Reg\_1 = Reg\_ReadMem;

end

end

end

end

always\_ff @(posedge ~clock)

begin

if (reset == 1) MK\_Adress = 0;

else

begin

if (start == 1)

begin

if (work\_en == 1)

begin

if (MK\_Adress == 17)

begin

if (condition1 == 1) MK\_Adress = 18;

else MK\_Adress = mem\_next\_MK\_Adress[MK\_Adress][4:0];

end

else if (MK\_Adress == 7)

begin

if (condition2 == 1)

begin

if (condition3 == 1) MK\_Adress = 9;

else MK\_Adress = 8;

end

else MK\_Adress = mem\_next\_MK\_Adress[MK\_Adress][4:0];

end

else if (MK\_Adress == 11)

begin

if (condition4 == 1)

begin

if (condition6 == 1) MK\_Adress = 13;

else MK\_Adress = 12;

end

else MK\_Adress = mem\_next\_MK\_Adress[MK\_Adress][4:0];

end

else if (MK\_Adress == 16)

begin

if (condition5 == 1) MK\_Adress = 14;

else MK\_Adress = mem\_next\_MK\_Adress[MK\_Adress][4:0];

end

else MK\_Adress = mem\_next\_MK\_Adress[MK\_Adress][4:0];

end

end

end

end

always\_ff @(posedge clock)

begin

if (counter\_cikl > 0) condition1 = 1;

else condition1 = 0;

condition2 = (Reg\_2[0] ^ Reg\_sd) & ~Reg\_d[0];

condition3 = Reg\_2[1] & Reg\_d[1];

condition4 = Reg\_2[7] | Reg\_2[6] | Reg\_2[5] | Reg\_2[4] | Reg\_2[3] | Reg\_2[2] | Reg\_2[1] | Reg\_2[0];

condition5 = Reg\_A[7];

condition6 = Reg\_A[7] | Reg\_B[7];

end

endmodule

…\antidrebezg.sv

module antidrebezg

(

input logic clock,

input logic reset,

input logic button,

output logic pulse

);

logic [7:0] count;

logic trigger\_button;

logic razresh;

always\_ff @(posedge clock)

begin

if(button == 0)

begin

count <= 0;

razresh <= 0;

end

else if (count == '1)

begin

count <= count;

razresh <= 1;

end

else

begin

count <= count + 1;

razresh <= 0;

end

end

always\_ff @(posedge clock)

begin

trigger\_button <= razresh;

end

always\_ff @(posedge clock)

begin

if((razresh == 1) && (trigger\_button == 0))

pulse <= 1;

else

pulse <= 0;

end

endmodule

…\SEG7.sv

module SEG7

(

input logic [3:0] i\_hundreds,

input logic [3:0] i\_tens,

input logic [3:0] i\_ones,

output logic [6:0] seg1,

output logic [6:0] seg2,

output logic [6:0] seg3

);

always\_comb

begin

case(i\_hundreds)

0: seg3 = 7'b1000000;

1: seg3 = 7'b1111001;

2: seg3 = 7'b0100100;

3: seg3 = 7'b0110000;

4: seg3 = 7'b0011001;

5: seg3 = 7'b0010010;

6: seg3 = 7'b0000010;

7: seg3 = 7'b1111000;

8: seg3 = 7'b0000000;

9: seg3 = 7'b0011000;

10: seg3 = 7'b0001000;

11: seg3 = 7'b0000011;

12: seg3 = 7'b1000110;

13: seg3 = 7'b0100001;

14: seg3 = 7'b0000110;

15: seg3 = 7'b0001110;

default: seg3 = 7'b1111111;

endcase

case(i\_tens)

0: seg2 = 7'b1000000;

1: seg2 = 7'b1111001;

2: seg2 = 7'b0100100;

3: seg2 = 7'b0110000;

4: seg2 = 7'b0011001;

5: seg2 = 7'b0010010;

6: seg2 = 7'b0000010;

7: seg2 = 7'b1111000;

8: seg2 = 7'b0000000;

9: seg2 = 7'b0011000;

10: seg2 = 7'b0001000;

11: seg2 = 7'b0000011;

12: seg2 = 7'b1000110;

13: seg2 = 7'b0100001;

14: seg2 = 7'b0000110;

15: seg2 = 7'b0001110;

default: seg2 = 7'b1111111;

endcase

case(i\_ones)

0: seg1 = 7'b1000000;

1: seg1 = 7'b1111001;

2: seg1 = 7'b0100100;

3: seg1 = 7'b0110000;

4: seg1 = 7'b0011001;

5: seg1 = 7'b0010010;

6: seg1 = 7'b0000010;

7: seg1 = 7'b1111000;

8: seg1 = 7'b0000000;

9: seg1 = 7'b0011000;

10: seg1 = 7'b0001000;

11: seg1 = 7'b0000011;

12: seg1 = 7'b1000110;

13: seg1 = 7'b0100001;

14: seg1 = 7'b0000110;

15: seg1 = 7'b0001110;

default: seg1 = 7'b1111111;

endcase

end

endmodule

…\SEG7counter.sv

module SEG7counter

(

input logic clock,

input logic reset,

input logic [4:0] counter\_settings,

input logic [6:0] memory,

output logic [6:0] ssegment1,

output logic [6:0] ssegment2,

output logic [6:0] ssegment3,

output logic [6:0] ssegment5,

output logic [6:0] ssegment6,

output logic [6:0] ssegment7

);

bin\_to\_bcd NUMBER

(

.preobr (memory),

.ssegment1 (ssegment1),

.ssegment2 (ssegment2),

.ssegment3 (ssegment3)

);

bin\_to\_bcd COUNT

(

.preobr (counter\_settings),

.ssegment1 (ssegment5),

.ssegment2 (ssegment6),

.ssegment3 (ssegment7)

);

endmodule

…\bin\_to\_bcd.sv

module bin\_to\_bcd

(

input logic [6:0] preobr,

output logic [6:0] ssegment1,

output logic [6:0] ssegment2,

output logic [6:0] ssegment3

);

logic [3:0] hundreds;

logic [3:0] tens;

logic [3:0] ones;

integer i;

always\_comb

begin

hundreds = 4'd0;

tens = 4'd0;

ones = 4'd0;

for (i = 6; i >= 0; i = i - 1)

begin

if (hundreds >= 5) hundreds = hundreds + 3;

if (tens >= 5) tens = tens + 3;

if (ones >= 5) ones = ones + 3;

hundreds = hundreds << 1;

hundreds[0] = tens[3];

tens = tens << 1;

tens[0] = ones[3];

ones = ones << 1;

ones[0] = preobr[i];

end

end

SEG7 DISPLAY\_HEX

(

.i\_hundreds (hundreds),

.i\_tens (tens),

.i\_ones (ones),

.seg1 (ssegment1),

.seg2 (ssegment2),

.seg3 (ssegment3)

);

endmodule

## Приложение 2

…\TB\tb.sv

//Директива определения точности моделирования

`timescale 1ns/1ns

//Объявление модуля testbench - являющегося модулем сборки

//тестируемого устройства и программы, которая его тестирует,

//т.е. модуль организует интерфейс между ними

module tb ();

//Объявление самого интерфейса (проводов) между программой и модулем

logic clock;

logic reset;

logic button;

logic switch;

//Объявление экземпляра тестовой программы

test TEST

(

.clock (clock ),

.reset (reset ),

.button (button ),

.switch (switch )

);

//Объявление экземпляра устройства

Control\_Unit CN

(

.clock (clock ),

.reset (reset ),

.button (button ),

.switch (switch ),

.ssegment0 (),

.ssegment1 (),

.ssegment2 (),

.ssegment4 (),

.ssegment5 (),

.ssegment6 ()

);

endmodule

…\TB\test.sv

`timescale 1ns/1ns

//Описание работы тестовой программы

module test

(

// input logic [3:0] low\_sec\_dig,

// input logic [2:0] high\_sec\_dig,

//

// input logic second,

// input logic minute

output logic clock,

output logic reset,

output logic button,

output logic switch

);

// Объявление внутренних переменных

//Процедурный блок. Все операции процедурного блока между begin и end выполняются последовательно,

//т.е. как в обычном языке программирования (например С++)

initial begin

//Инициализация без затраты модельного времени

//Инициализация начальных значений выходных линий

clock = 0;

reset = 1;

//Инициализация начальных значений переменных

button = 0;

switch = 0;

//Процедурный блок fork ... join состоит из несккольких последовательных блоков, которые в свою очередь,

//выполняются параллельно относительно друг друга.

fork

//Первый последовательный блок

//Определение бесконечного цикла генерации сигнала синхрочастоты

forever #1 clock = ~clock; //#5 - означет задержку в модельном времени перед изменением сигнала

//относительно времени от предыдущего оператора. Единица измерения задержки

//первое значение в команде директиве timescale

//Второй последовательный блок

//Описание сигнала сброса

forever begin

#2 reset = ~reset;

#1000 button = ~button;

#1000 button = ~button;

#1000 button = ~button;

#1000 button = ~button;

#1000 button = ~button;

#1000 button = ~button;

#1000 button = ~button;

#1000 button = ~button;

#1000 button = ~button;

#1000 button = ~button;

#1000 button = ~button;

#1000 button = ~button;

#1000 button = ~button;

#1000 button = ~button;

#1000 button = ~button;

#1000 button = ~button;

#1000 button = ~button;

#1000 button = ~button;

#1000 button = ~button;

#1000 button = ~button;

#1000 button = ~button;

#1000 button = ~button;

#1000 button = ~button;

#1000 button = ~button;

#1000 button = ~button;

#1000 button = ~button;

#1000 button = ~button;

#1000 button = ~button;

#1000 button = ~button;

#1000 button = ~button;

#1000 button = ~button;

#1000 button = ~button;

#1000 button = ~button;

#1000 button = ~button;

#1000 switch = ~switch;

#10000 reset = ~reset;

end

begin

$display ("------------------------ATTENTION------------------------------");

$display ("Please add additional wave to waveform and run simulation");

$display ("Menu Simulate/Run/Run -all or write command run -all to console");

$display ("------------------------ATTENTION------------------------------");

$stop();

end

join // Окончание параллельного блока

end

endmodule

Дипломный проект выполнен мной самостоятельно. Использованные в работе материалы из опубликованной литературы и другие источники имеют ссылки в тексте.

Отпечатано в одном экземпляре, копия работы представлена на электронном носителе.

      2018г. (личная подпись ) (И.О.Ф)