3, 5)

module lab5p1(SW, KEY, HEX0, HEX1);

input [1:0] SW;

input [0:0] KEY;

wire [7:0] q;

output [6:0] HEX0, HEX1;

bitcounter b0(.enable(SW[1]), .clock(KEY[0]), .clear\_b(SW[0]), .q(q));

segment h1(.out(q[7:4]), .in(HEX1));

segment h0(.out(q[3:0]), .in(HEX0));

endmodule

module bitcounter(enable, clock, clear\_b, q);

input enable, clock, clear\_b;

output [7:0] q;

tflfl t1(.enable(enable),

.clk(clock),

.clear\_b(clear\_b),

.q(q[0]));

tflfl t2(.enable(q[0] && enable),

.clk(clock),

.clear\_b(clear\_b),

.q(q[1]));

tflfl t3(.enable(q[1] && q[0] && enable),

.clk(clock),

.clear\_b(clear\_b),

.q(q[2]));

tflfl t4(.enable(q[2] && q[1] && q[0] && enable),

.clk(clock),

.clear\_b(clear\_b),

.q(q[3]));

tflfl t5(.enable(q[3] && q[2] && q[1] && q[0] && enable),

.clk(clock),

.clear\_b(clear\_b),

.q(q[4]));

tflfl t6(.enable(q[4] && q[3] && q[2] && q[1] && q[0] && enable),

.clk(clock),

.clear\_b(clear\_b),

.q(q[5]));

tflfl t7(.enable(q[5] && q[4] && q[3] && q[2] && q[1] && q[0] && enable),

.clk(clock),

.clear\_b(clear\_b),

.q(q[6]));

tflfl t8(.enable(q[6] && q[5] && q[4] && q[3] && q[2] && q[1] && q[0] && enable),

.clk(clock),

.clear\_b(clear\_b),

.q(q[7]));

endmodule

module tflfl (enable, clk, clear\_b, q);

input enable, clk, clear\_b ;

output q;

reg q;

always @ (posedge clk, negedge clear\_b)

if (clear\_b == 1'b0) begin

q <= 1'b0;

end else if (enable) begin

q <= !q;

end

endmodule

module segment(out, in);

input [3:0] out;

output [6:0] in;

assign in[0]=((!out[3])&(!out[2])&(!out[1])&out[0])|((!out[3])&out[2]&(!out[1])&(!out[0]))|(out[3]&(!out[2])&out[1]&out[0])|(out[3]&out[2]&(!out[1])&out[0]);

assign in[1]=(out[3]&out[2]&(!out[0]))|((!out[3])&out[2]&(!out[1])&out[0])|(out[3]&out[1]&out[0])|(out[2]&out[1]&(!out[0]));

assign in[2]=((!out[3])&(!out[2])&out[1]&(!out[0]))|(out[3]&out[2]&out[1])|(out[3]&out[2]&(!out[0]));

assign in[3]=((!out[3])&(!out[2])&(!out[1])&out[0])|((!out[3])&out[2]&(!out[1])&(!out[0]))|(out[3]&(!out[2])&out[1]&(!out[0]))|(out[2]&out[1]&out[0]);

assign in[4]=((!out[3])&out[2]&(!out[1]))|((!out[2])&(!out[1])&out[0])|((!out[3])&out[0]);

assign in[5]=((!out[3])&(!out[2])&out[0])|((!out[3])&(!out[2])&out[1])|(out[3]&out[2]&(!out[1])&out[0])|((!out[3])&out[1]&out[0]);

assign in[6]=((!out[3])&(!out[2])&(!out[1]))|(out[3]&out[2]&(!out[1])&(!out[0]))|((!out[3])&out[2]&out[1]&out[0]);

endmodule

3) Q0, Q1, Q2, Q3, Q4, Q5, Q6, Q7

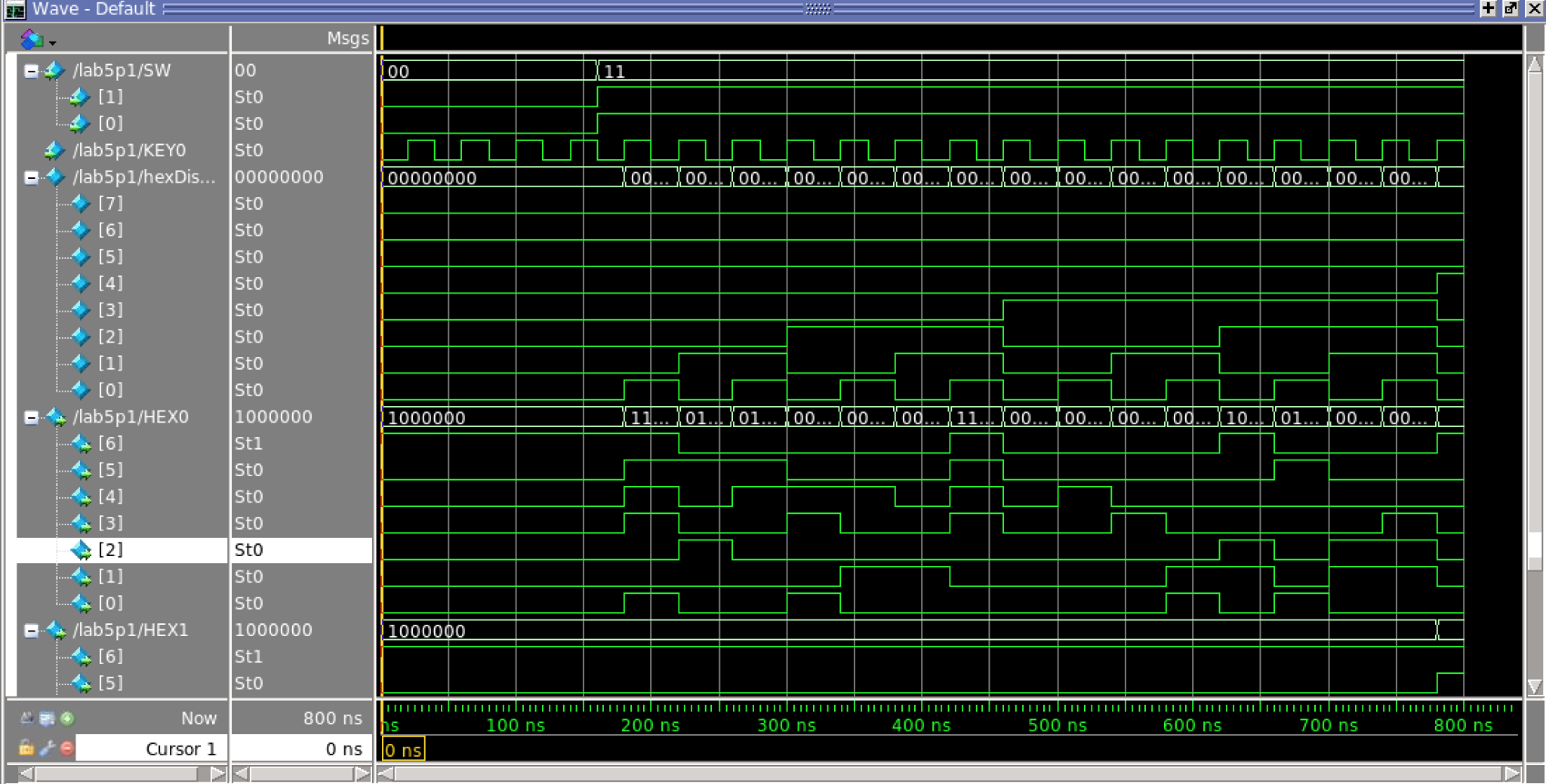
4)

6)

(13) ALMs, less then 1%

641.03 MHz (Restricted 621.89 MHz)

7) It is almost the same



2)

module lab5p2(SW, HEX0, CLOCK\_50);

input [4:0] SW;

input CLOCK\_50;

wire [28:0] control;

wire [28:0] q;

wire [3:0] counteroutput;

output [6:0] HEX0;

speed s(.switch(SW[1:0]),

.d(control));

ratedivider r(.d(control),

.clock(CLOCK\_50),

.reset\_n(SW[2]),

.par\_load(SW[3]),

.enable(SW[4]),

.q(q));

displaycounter d(.d(q),

.clock(CLOCK\_50),

.reset\_n(SW[2]),

.par\_load(SW[3]),

.q(counteroutput));

segment s0(.out(counteroutput),

.in(HEX0));

endmodule

module displaycounter(d, clock, reset\_n, par\_load, q);

input wire [28:0] d;

input wire clock;

input wire reset\_n, par\_load ;

wire enable;

assign enable = ( d == 0 ) ? 1 : 0 ;

output reg [3:0] q;

always @(posedge clock)

begin

if ( reset\_n == 1'b0 )

q <= 0 ;

else if ( par\_load == 1'b1 )

q <= 0 ;

else if ( enable == 1'b1 )

begin

if ( q == 4'b1111 )

q <= 0 ;

else

q <= q + 1'b1 ;

end

end

endmodule

module ratedivider(d, clock, reset\_n, par\_load, enable, q);

input wire [28:0] d;

input wire clock;

input wire reset\_n ;

input wire par\_load, enable;

output reg [28:0] q;

always @(posedge clock)

begin

if ( reset\_n == 1'b0 )

q <= 0 ;

else if ( par\_load == 1'b1 )

q <= d ;

else if ( enable == 1'b1 )

begin

if ( q == 0 )

q <= d ;

else

q <= q - 1'd1 ;

end

end

endmodule

module speed(switch, d);

input [1:0] switch;

output reg [28:0] d;

always @(\*)

begin

case (switch[1:0])

2'b00: d = 1;

2'b01: d = 49\_999\_999;

2'b10: d = 99\_999\_999;

2'b11: d = 199\_999\_999;

default: ;

endcase

end

endmodule

module segment(out, in);

input [3:0] out;

output [6:0] in;

assign in[0]=((!out[3])&(!out[2])&(!out[1])&out[0])|((!out[3])&out[2]&(!out[1])&(!out[0]))|(out[3]&(!out[2])&out[1]&out[0])|(out[3]&out[2]&(!out[1])&out[0]);

assign in[1]=(out[3]&out[2]&(!out[0]))|((!out[3])&out[2]&(!out[1])&out[0])|(out[3]&out[1]&out[0])|(out[2]&out[1]&(!out[0]));

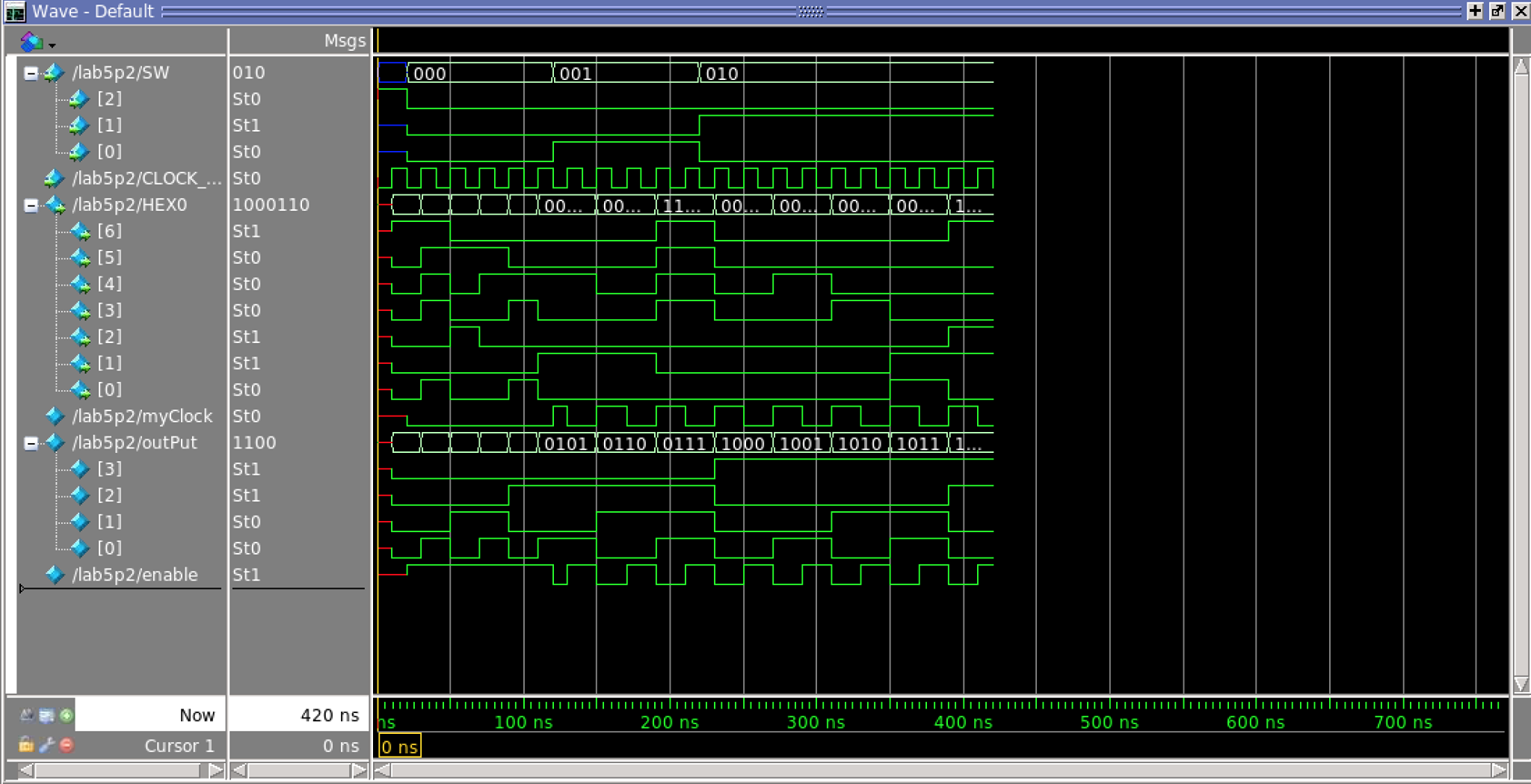
assign in[2]=((!out[3])&(!out[2])&out[1]&(!out[0]))|(out[3]&out[2]&out[1])|(out[3]&out[2]&(!out[0]));

assign in[3]=((!out[3])&(!out[2])&(!out[1])&out[0])|((!out[3])&out[2]&(!out[1])&(!out[0]))|(out[3]&(!out[2])&out[1]&(!out[0]))|(out[2]&out[1]&out[0]);

assign in[4]=((!out[3])&out[2]&(!out[1]))|((!out[2])&(!out[1])&out[0])|((!out[3])&out[0]);

assign in[5]=((!out[3])&(!out[2])&out[0])|((!out[3])&(!out[2])&out[1])|(out[3]&out[2]&(!out[1])&out[0])|((!out[3])&out[1]&out[0]);

assign in[6]=((!out[3])&(!out[2])&(!out[1]))|(out[3]&out[2]&(!out[1])&(!out[0]))|((!out[3])&out[2]&out[1]&out[0]);]

endmodule

3)

3)

module part3(CLOCK\_50, SW, LEDR, KEY);

input CLOCK\_50;

input [2:0] SW;

input [1:0] KEY;

output [1:0] LEDR;

wire divider;

TwentyFiveMilCounter tfmc(.clk(CLOCK\_50),

.reset(KEY[0]|divider),

.out(divider) );

MorseEncode morse(.clk(divider),

.load(KEY[1]),

.reset(KEY[0]),

.letter(SW[2:0]),

.out(LEDR[0]));

endmodule

module TwentyFiveMilCounter(clk,reset, out);

input clk, reset;

output out;

reg [24:0] curr;

assign out = (curr==25'b1011111010111100000111111)? 1:0;

always@(posedge clk, posedge reset)

begin

if(reset)

curr<= 25'b0000000000000000000000000;

else

curr <= curr+1;

end

endmodule

module MorseEncode(clk, load, reset, letter, out);

input clk, load, reset;

input [2:0] letter;

output out;

wire [13:0] decoded;

LookUpTable lut(.letter3bit(letter),

.morseOut(decoded));

ShiftRegister shftrg(.enable(clk),

.load\_val(decoded),

.reset(reset),

.set(load),

.out(out));

endmodule

module LookUpTable(letter3bit, morseOut);

input [2:0] letter3bit;

output [13:0] morseOut;

reg morseOut;

always@(\*)

case(letter3bit)

3'b000: morseOut<=14'b10101000000000; //S ...

3'b001: morseOut<=14'b11100000000000;//T \_

3'b010: morseOut<=14'b10101110000000;//U ..\_

3'b011: morseOut<=14'b10101011100000;//V ...\_

3'b100: morseOut<=14'b10111011100000;//W .\_ \_

3'b101: morseOut<=14'b11101010111000;//X \_ .. \_

3'b110: morseOut<=14'b11101011101110;//Y \_ . \_ \_

3'b111: morseOut<=14'b11101110101000; // Z \_ \_ ..

default:morseOut<=14'b00000000000000;

endcase

endmodule

module ShiftRegister(enable, load\_val, reset, set, out);

input enable, reset, set;

input [13:0] load\_val;

reg [13:0] array;

assign out=array[0];

output out;

always@(posedge enable, posedge set, posedge reset)

begin

if(set)

array <= load\_val;

else if(reset)

array <= {14{1'b0}};

else

array <= array>>1'b1;

end

endmodule