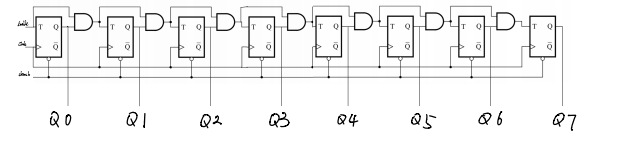
**Lab5 Pre-Lab Report**

Ziyao Chen

1002453409

**Part1**







module eightbitcounter(enable, clk, clear\_b, Q);

input enable, clk, clear\_b;

output [7:0] Q;

wire [6:0] connection;

tflipflop t0(

.enable(enable),

.clk(clk),

.clear\_b(clear\_b),

.Q(Q[0])

);

assign connection[0] = Q[0] && enable;

tflipflop t1(

.enable(connection[0]),

.clk(clk),

.clear\_b(clear\_b),

.Q(Q[1])

);

assign connection[1] = Q[1] && connection[0];

tflipflop t2(

.enable(connection[1]),

.clk(clk),

.clear\_b(clear\_b),

.Q(Q[2])

);

assign connection[2] = Q[2] && connection[1];

tflipflop t3(

.enable(connection[2]),

.clk(clk),

.clear\_b(clear\_b),

.Q(Q[3])

);

assign connection[3] = Q[3] && connection[2];

tflipflop t4(

.enable(connection[3]),

.clk(clk),

.clear\_b(clear\_b),

.Q(Q[4])

);

assign connection[4] = Q[4] && connection[3];

tflipflop t5(

.enable(connection[4]),

.clk(clk),

.clear\_b(clear\_b),

.Q(Q[5])

);

assign connection[5] = Q[5] && connection[4];

tflipflop t6(

.enable(connection[5]),

.clk(clk),

.clear\_b(clear\_b),

.Q(Q[6])

);

assign connection[6] = Q[6] && connection[5];

tflipflop t7(

.enable(connection[6]),

.clk(clk),

.clear\_b(clear\_b),

.Q(Q[7])

);

endmodule

module tflipflop(enable, clk, clear\_b, Q);

input enable, clk, clear\_b;

output Q;

reg Q;

always @(posedge clk, negedge clear\_b)

begin

if (clear\_b == 1'b0)

Q <= 0;

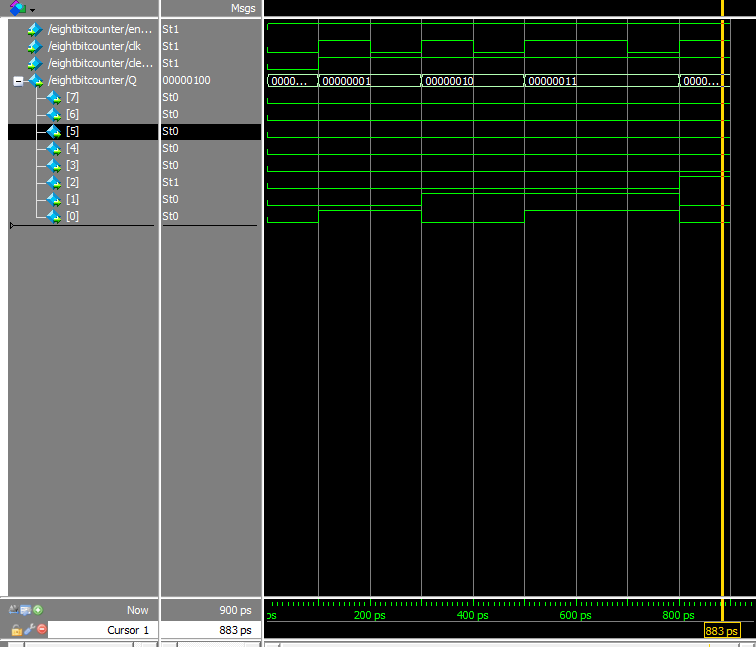
else

Q <= enable ^ Q;

end

endmodule

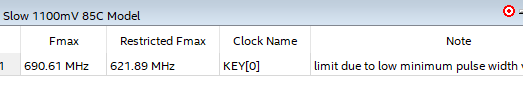






![C:\Users\chenz\AppData\Roaming\Tencent\Users\498992664\QQ\WinTemp\RichOle\(E4{BGMT5@C](7B~HXY1Z)N.png](data:image/png;base64,)

2. Its 13/32070 (< 1%)



1. It’s almost the same but the size if double of figure1, and it has HEX0 and HEX1 to display the values.

**Part2**

Because in binary arithmetic, 1111 + 1 = 10000, since q is declared as a 4-bit value, it takes the right most four digits of the result, which will be ‘0000’. Thus the check for the maximum value is not necessary.

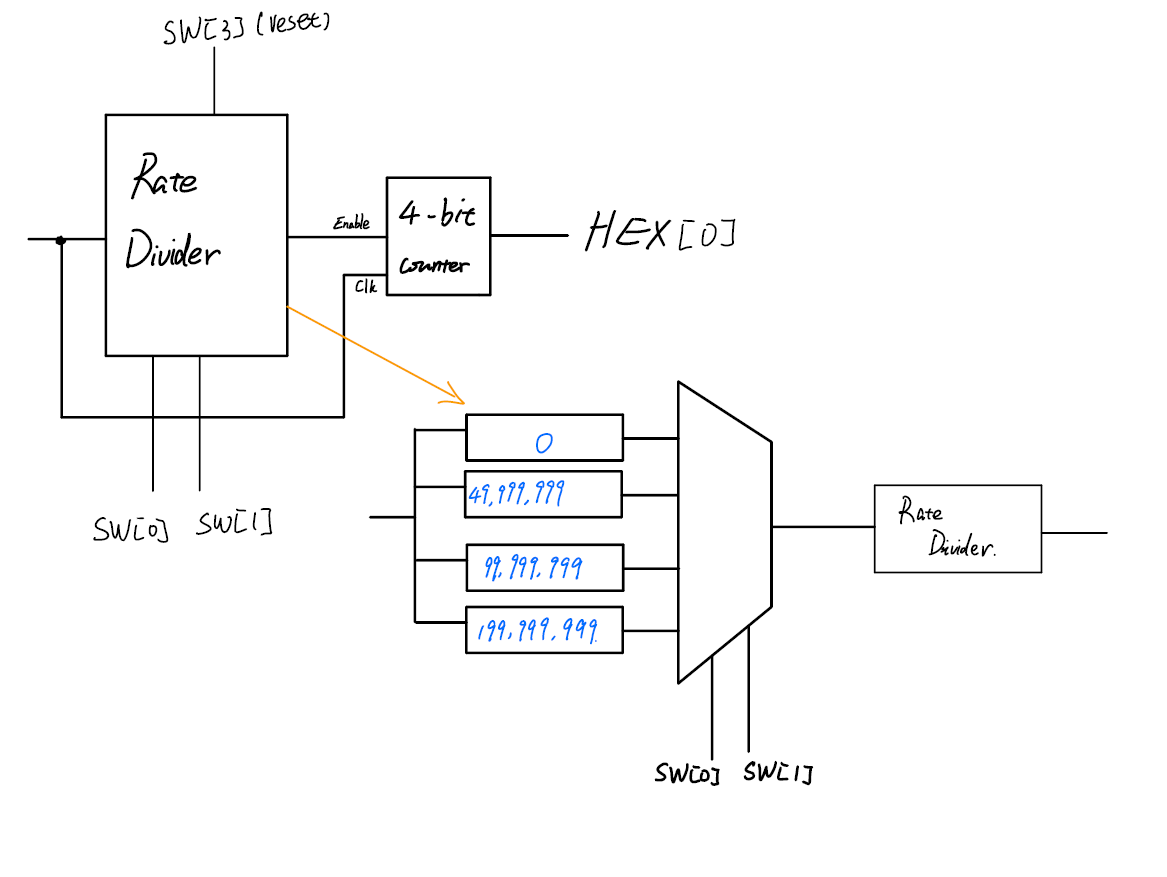
Change the line of code which checks the maximum value of q to be:

if (q == 4’b1001)

The frequency of each change(on to off or off to on) is 1s/50MHz which is equals to 20 nano second. Thus we expect to see all the segments lighting on, because the Speed is Full which is too fast even that human eyes can hardly notice the on and off action of any particular segments.

We will need the ceiling of logbase 2 of 50 million, which is 26 enables. Thus we would need 26 bits to represent such a value.







module part2(SW, CLOCK\_50, HEX0);

input [2:0] SW;

input CLOCK\_50;

output [6:0] HEX0;

wire [3:0] connection;

whole w1(

.clk(CLOCK\_50),

.reset\_r(SW[2]),

.reset\_d(SW[2]),

.s(SW[1:0]),

.out(connection)

);

SevenSegmentDecoder s0(

.c(connection),

.ss\_out(HEX0)

);

endmodule

module whole(clk, reset\_r, reset\_d, s, out);

input [1:0] s; //selecters

input clk;

input reset\_r; //reset for ratedivider

input reset\_d; //reset for displaycounter

output [3:0] out;

reg [27:0] max;

wire connection1;

always @(\*)

begin

case(s[1:0])

2'b00: max = 0;

2'b01: max = 28'b0010111110101111000001111111;

2'b10: max = 28'b0101111101011110000011111111;

2'b11: max = 28'b1011111010111100000111111111;

endcase

end

RateDivider r1(

.max(max),

.clock(clk),

.reset\_n(reset\_r),

.enable(connection1)

);

DisplayCounter d1(

.enable(connection1),

.clk(clk),

.reset(reset\_d),

.Q(out)

);

endmodule

module DisplayCounter(enable, clk, reset, Q);

input enable, clk;

input reset;

output [3:0] Q;

reg [3:0] Q;

always @(posedge clk)

begin

if (reset == 1'b0)

Q <= 0;

else if (enable == 1'b1)

Q <= Q + 1'b1;

end

endmodule

module RateDivider(max, clock, reset\_n, enable);

input [27:0] max;

input clock, reset\_n;

output enable;

reg [27:0] q;

always @(posedge clock)

begin

if ((reset\_n) == 1'b0)

q <= 0;

else

if (q == 0)

q <= max;

else

q <= q - 1'b1;

end

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

endmodule

module SevenSegmentDecoder(c,ss\_out);

input [3:0] c; //the bcd input

output [6:0] ss\_out; //the seven segements

assign ss\_out[0] = ~c[3] & ~c[2] & ~c[1] & c[0] | ~c[3] & c[2] & ~c[1] & ~c[0] | c[3] & ~c[2] & c[1] & c[0] | c[3] & c[2] & ~c[1] & c[0];

assign ss\_out[1] = c[3] & c[1] & c[0] | c[2] & c[1] & ~c[0] | ~c[3] & c[2] & ~c[1] & c[0] | c[3] & c[2] & ~c[0];

assign ss\_out[2] = ~c[3] & ~c[2] & c[1] & ~c[0] | c[3] & c[2] & ~c[0] | c[3] & c[2] & c[1];

assign ss\_out[3] = ~c[3] & ~c[2] & ~c[1]& c[0] | ~c[3] & c[2] & ~c[1] & ~c[0] | c[2] & c[1] & c[0] | c[3] & ~c[2] & c[1] & ~c[0];

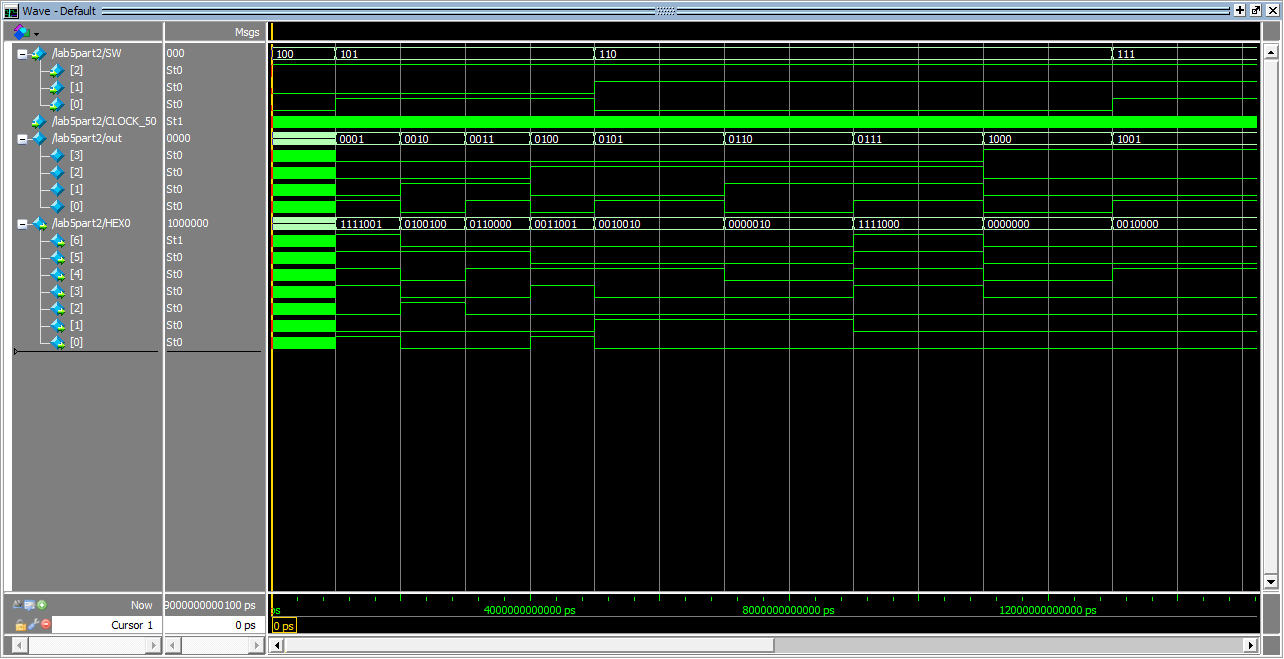
assign ss\_out[4] = ~c[3] & c[0] | ~c[3] & c[2] & ~c[1] | ~c[2] & ~c[1] & c[0];

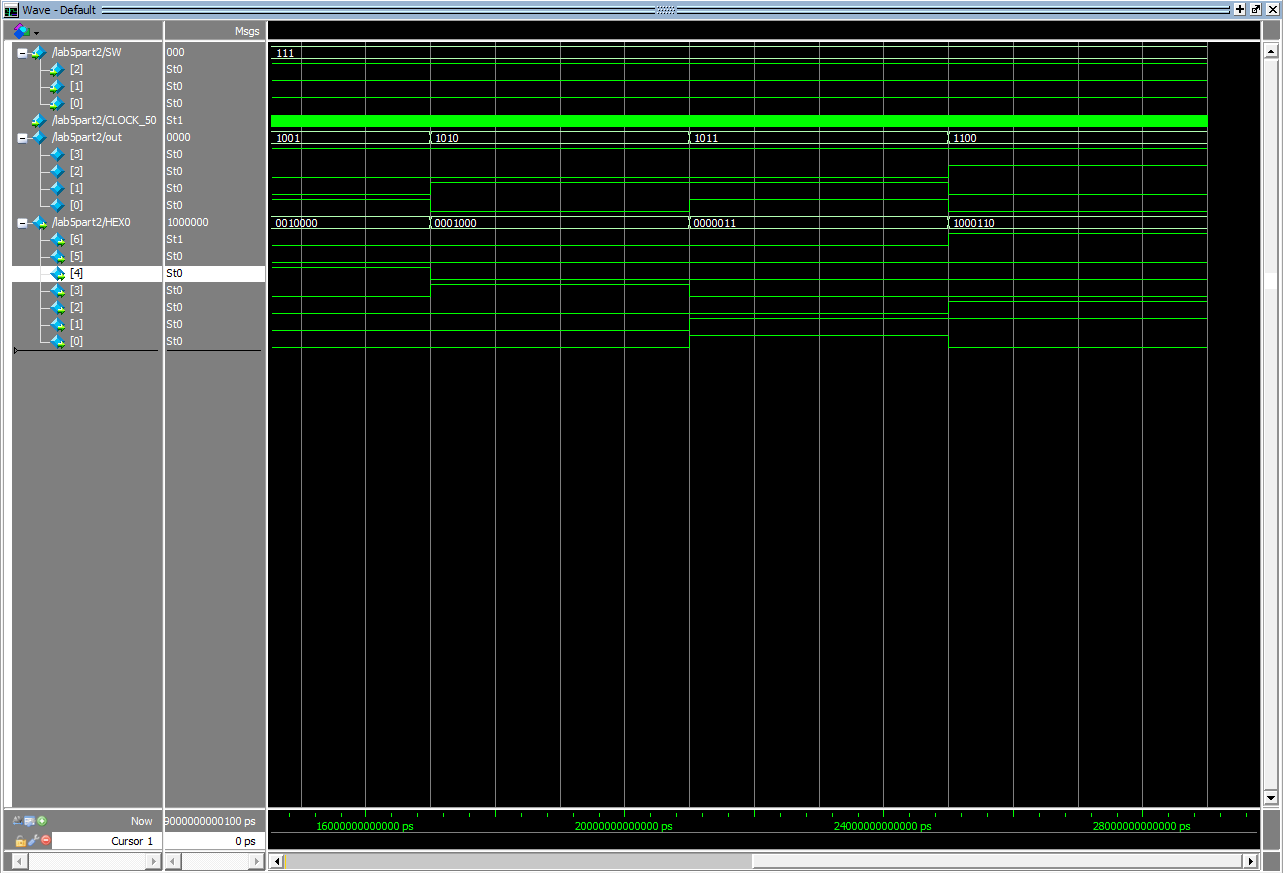
assign ss\_out[5] = ~c[3] & ~c[2] & c[0] | ~c[3] & ~c[2] & c[1] | ~c[3] & c[1] & c[0] | c[3] & c[2] & ~c[1] & c[0];

assign ss\_out[6] = ~c[3] & ~c[2] & ~c[1] | ~c[3] & c[2] & c[1] & c[0] | c[3] & c[2] & ~c[1] & ~c[0];

endmodule



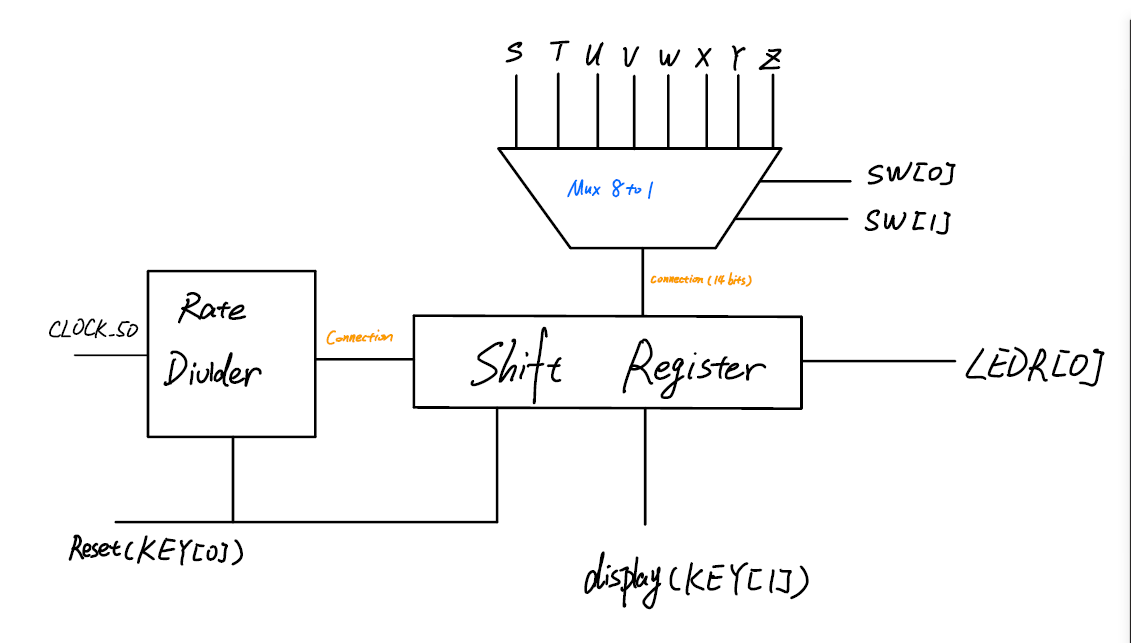




**Part3**

|  |  |
| --- | --- |
| **Letter** | **Pattern Representation (pattern length is 14 bits)** |
| S | 10101000000000 |
| T | 11100000000000 |
| U | 10101110000000 |
| V | 10101011100000 |
| W | 10111011100000 |
| X | 11101010111000 |
| Y | 11101011101110 |
| Z | 11101110101000 |

1. Bit width is 14 bits.



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

input CLOCK\_50;

input [1:0] KEY;

input [2:0] SW;

output [9:0] LEDR;

morsecode m1(

.clock\_50(CLOCK\_50),

.selecters(SW),

.display(KEY[1]),

.reset(KEY[0]),

.out\_light(LEDR[0])

);

endmodule

module morsecode(clock\_50, selecters, display, reset, out\_light);

input [2:0] selecters;

input display, reset, clock\_50;

output out\_light;

wire [13:0] LUT\_to\_Shift;

wire signal;

LUT L1(

.s(selecters),

.q(LUT\_to\_Shift)

);

ShiftRegister s1(

.display(display),

.bit(LUT\_to\_Shift),

.clock(signal),

.reset(reset),

.output\_seq(out\_light)

);

RateDivider r1(

.clock(clock\_50),

.reset\_n(reset),

.out(signal)

);

endmodule

module RateDivider(clock, reset\_n, out);

input clock, reset\_n;

output out;

wire [21:0] max = 22'b1001100010010110100000;

reg [21:0] q;

always @(posedge clock)

begin

if ((reset\_n) == 1'b0)

q <= 0;

else

if (q == 0)

q <= max;

else

q <= q - 1'b1;

end

assign out = (q == 0) ? 1 : 0;

endmodule

module LUT(s, q);

input [2:0] s; //selecters

output [13:0] q; // 14-bit representation of letters

reg [13:0] q;

always @(\*)

begin

case(s[2:0])

3'b000: q = 14'b10101000000000;

3'b001: q = 14'b11100000000000;

3'b010: q = 14'b10101110000000;

3'b011: q = 14'b10101011100000;

3'b100: q = 14'b10111011100000;

3'b101: q = 14'b11101010111000;

3'b110: q = 14'b11101011101110;

3'b111: q = 14'b11101110101000;

endcase

end

endmodule

module ShiftRegister(display, bit, clock, reset, output\_seq);

input display;

input [13:0] bit;

input clock;

input reset;

output output\_seq;

reg [13:0] q;

wire counter = 14;

always @(posedge clock, negedge reset, negedge display)

begin

if (reset == 0)

q = 14'b00000000000000;

else

if (display == 0)

q <= bit;

else

q = q << 1'b1;

end

assign output\_seq = q[13];

endmodule