

CSC258H1: Pre-lab Exercise for Lab 5

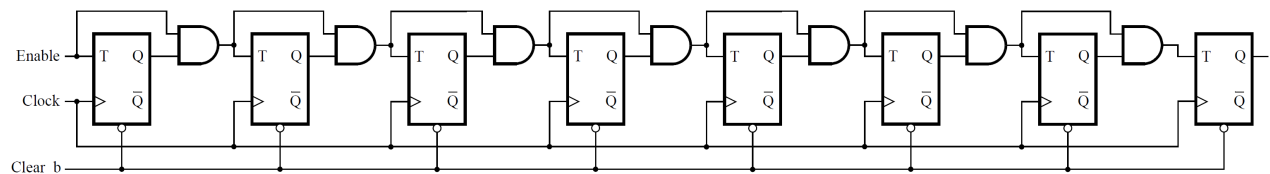
Wang, Weiqing

Due: February 11th, 2018 before 12 a.m.

1 Part I

1. Draw the schematic for an 8-bit counter using the same structure as shown in Figure 1.

Solution.



2. Annotate all Q outputs of your schematic with the bit of the counter ($Q_7Q_6Q_5Q_4Q_3Q_2Q_1Q_0$) that they correspond to.

Solution.

The output Q of the T-type Flip-Flops from right to left are respectively Q_{7-0} .

3. Write the Verilog corresponding to your schematic. Your code should use a module for the flip-flop that is instantiated 8 times to create the counter. Note that you should not name your module TFF as this is a Quartus primitive and thus Quartus will ignore that entity after issuing a warning. (e.g. use a name such as MyTFF).

Solution.

Here is the Verilog for my 8-bit counter.

```

1 module eightBitCounter(KEY, SW, HEX0, HEX1, LEDR);
2     input [0:0] KEY;    //Clock.
3     input [1:0] SW;     //SW[0] for Clear_b, SW[1] for Enable.
4     output [6:0] HEX0;
5     output [6:0] HEX1;
6     output [7:0] LEDR;
7     wire [6:0] T;
8
9     assign T[0] = SW[1] & LEDR[0];
10    assign T[1] = T[0] & LEDR[1];
11    assign T[2] = T[1] & LEDR[2];
12    assign T[3] = T[2] & LEDR[3];
13    assign T[4] = T[3] & LEDR[4];
14    assign T[5] = T[4] & LEDR[5];
15    assign T[6] = T[5] & LEDR[6];
16
17    tTypeFlipFlop TFF0(.Clk(KEY[0]), .T(SW[0]), .Q(LED0[0]), .Clear_n(SW[0]));
18    tTypeFlipFlop TFF1(.Clk(KEY[0]), .T(T[0]), .Q(LED0[1]), .Clear_n(SW[0]));
19    tTypeFlipFlop TFF2(.Clk(KEY[0]), .T(T[1]), .Q(LED0[2]), .Clear_n(SW[0]));
20    tTypeFlipFlop TFF3(.Clk(KEY[0]), .T(T[2]), .Q(LED0[3]), .Clear_n(SW[0]));
21    tTypeFlipFlop TFF4(.Clk(KEY[0]), .T(T[3]), .Q(LED0[4]), .Clear_n(SW[0]));
22    tTypeFlipFlop TFF5(.Clk(KEY[0]), .T(T[4]), .Q(LED0[5]), .Clear_n(SW[0]));
23    tTypeFlipFlop TFF6(.Clk(KEY[0]), .T(T[5]), .Q(LED0[6]), .Clear_n(SW[0]));
24    tTypeFlipFlop TFF7(.Clk(KEY[0]), .T(T[6]), .Q(LED0[7]), .Clear_n(SW[0]));
25
26    hexDecoder hex0(.SW(LED0[3:0]), .HEX(HEX0[6:0]));

```

```

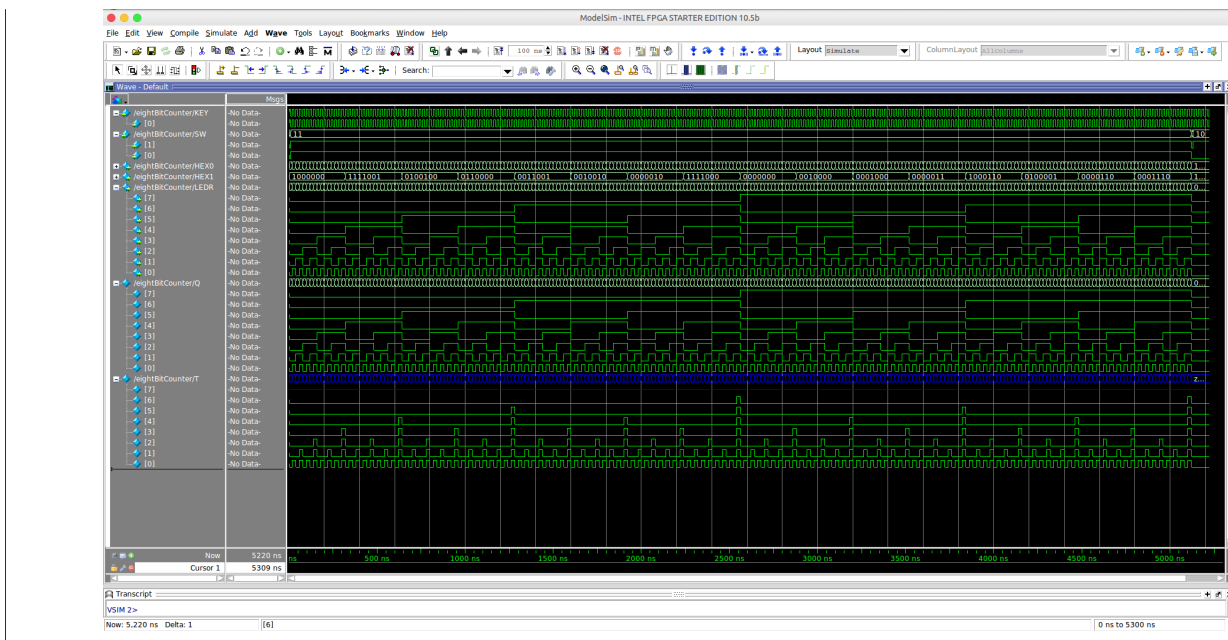
27     hexDecoder hex1(.SW(LED1[7:4]), .HEX(HEX1[6:0]));
28 endmodule
29
30 module tTypeFlipFlop(Clk, T, Clear_n, Q);
31     input Clk;
32     input T;
33     input Clear_n;
34     output reg Q;
35
36     always @(posedge Clk, negedge Clear_n)
37     begin
38         if (Clear_n == 1'b0)
39             Q <= 1'b0;
40         else if (T == 1'b1)
41             Q <= !Q;
42     end
43 endmodule
44
45 module hexDecoder (SW, HEX);
46     input [3:0] SW;
47     reg [6:0] result;
48     output reg [6:0] HEX;
49
50     always @(*)
51     begin
52         case (SW[3:0])
53             4'b0000: HEX[6:0] = 7'b1000000;
54             4'b0001: HEX[6:0] = 7'b1111001;
55             4'b0010: HEX[6:0] = 7'b0100100;
56             4'b0011: HEX[6:0] = 7'b0110000;
57             4'b0100: HEX[6:0] = 7'b0011001;
58             4'b0101: HEX[6:0] = 7'b0010010;
59             4'b0110: HEX[6:0] = 7'b0000010;
60             4'b0111: HEX[6:0] = 7'b1111000;
61             4'b1000: HEX[6:0] = 7'b0000000;
62             4'b1001: HEX[6:0] = 7'b0010000;
63             4'b1010: HEX[6:0] = 7'b0001000;
64             4'b1011: HEX[6:0] = 7'b0000011;
65             4'b1100: HEX[6:0] = 7'b1000110;
66             4'b1101: HEX[6:0] = 7'b0100001;
67             4'b1110: HEX[6:0] = 7'b0000110;
68             4'b1111: HEX[6:0] = 7'b0001110;
69             default: HEX[6:0] = 7'b1000000;
70         endcase
71     end
72 endmodule

```

4. Simulate your circuit in ModelSim to verify its correctness. You will need to reset (clear) all your flip-flops early in your simulation, so your circuit is in a known state. Include screenshots of simulation output in your prelab.

Solution.

Here is the screenshot for my ModelSim simulation:



5. Augment your Verilog code to use the push button KEY 0 as the Clock input, switches SW_1 and SW_0 as *Enable* and *Clear_b* inputs, and 7-segment displays *HEX0* and *HEX1* to display the hexadecimal value of your counter as your circuit operates. Simulate your circuit to ensure that you have done this correctly (include at least one screenshot).

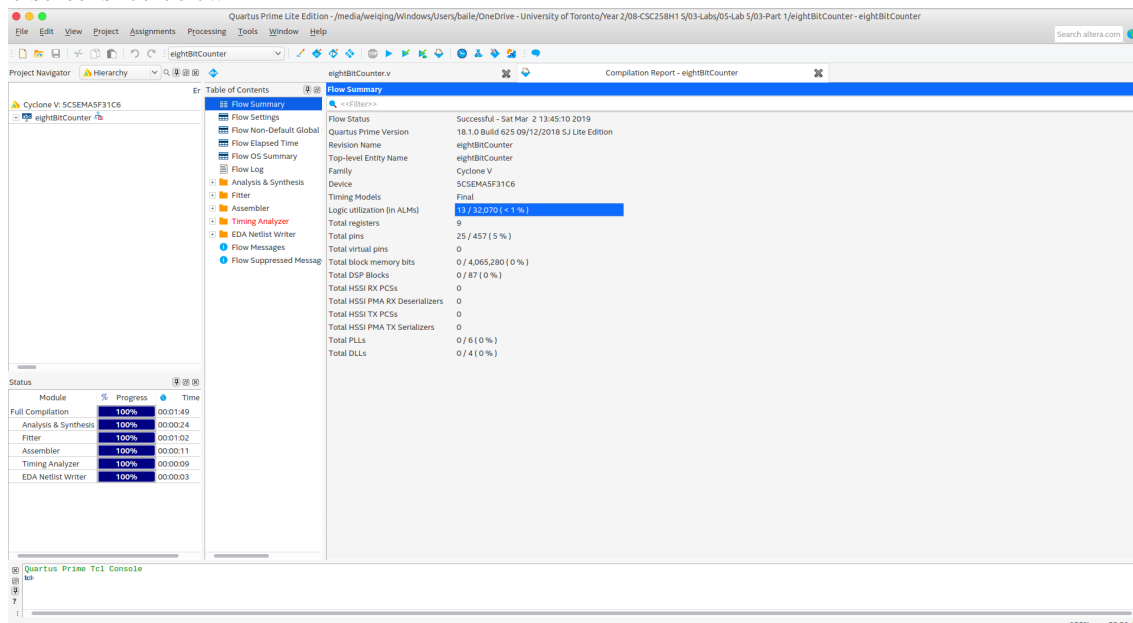
Solution.

See my solution to question 4. The simulation in Question 4 is enough to ensure I have done correctly.

6. Create a new Quartus Prime project for your circuit. Make sure it is stored in your W: drive. Do not forget to select the correct FPGA device (5CSEMA5F31C6) and import the pin assignments. Compile the circuit in Quartus and answer the following questions:
- (a) What percentage of FPGA logic resources are used to implement your circuit?

Solution.

See the screenshot below:



- (b) What is the maximum clock frequency, F_{max} , at which your circuit can be operated?

Solution.

See the screenshot below:

Quartus Prime Lite Edition - /media/weiqing/Windows/Users/baile/OneDrive - University of Toronto/Year 2/08-CSC258H1 5/03-Labs/05-Lab 5/03-Part 1/eightBitCounter - eightBitCounter

Project Navigator: eightBitCounter.v

Table of Contents:

- Flow Summary
- Flow Settings
- Flow Non-Default Global Settings
- Flow Elapsed Time
- Flow OS Summary
- Flow Log
- Analysis & Synthesis
 - Filter
 - Assembler
 - Timing Analyzer
 - Summary
 - Parallel Compilation
 - Clocks
 - Slow 1100mV 85C Model
 - Fmax Summary
 - Timing Closure Reco
 - Setup Summary
 - Hold Summary
 - Recovery Summary
 - Removal Summary
 - Minimum Pulse Width
 - Metastability Summary
 - Slow 1100mV OC Model
 - Fast 1100mV 85C Model
 - Fast 1100mV OC Model
 - Multicorner Timing Anal
 - Advanced I/O Timing
 - Clock Transfers
 - Report TCES
 - Report RSKM
 - Unconstrained Paths
 - EDA Netlist Writer
 - Flow Messages
 - Flow Suppressed Messages

Status:

Module	% Progress	Time
Full Compilation	100%	00:01:49
Analysis & Synthesis	100%	00:00:24
Filter	100%	00:01:02
Assembler	100%	00:00:11
Timing Analyzer	100%	00:00:09
EDA Netlist Writer	100%	00:00:03

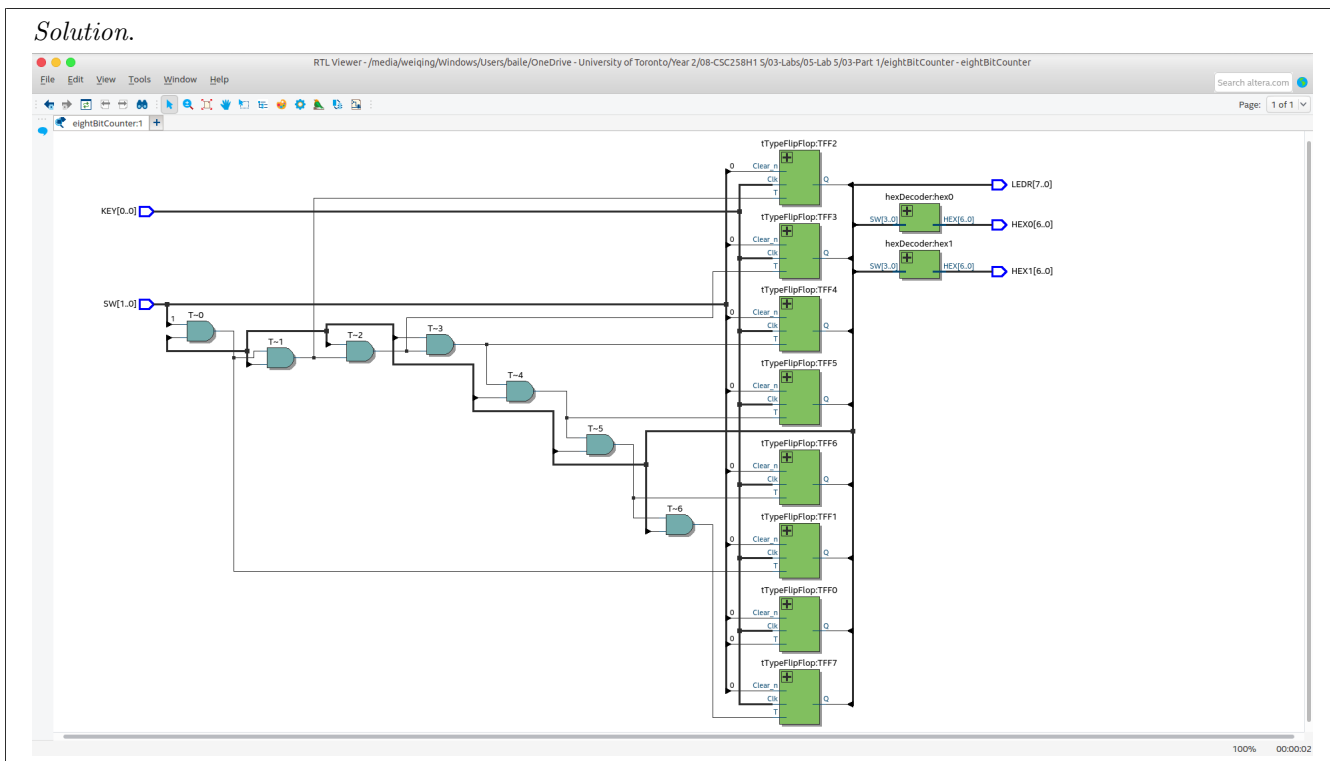
Slow 1100mV 85C Model Fmax Summary

Fmax	Restricted Fmax	Clock Name	Note
1	659.63 MHz	620.35 MHz	KEY[0] limit due to low minimum pulse width violation (tcl)

This panel reports FMAX for every clock in the design, regardless of the user-specified clock periods. FMAX is only computed for paths where the source and destination registers or ports are driven by the same clock. Paths of different clocks, including generated clocks, are ignored. For paths between a clock and its inversion, FMAX is computed as if the rising and falling edges are scaled along with FMAX, such that the duty cycle (in terms of a percentage) is maintained. Altera recommends that you always use clock constraints and other slack reports for sign-off analysis.

7. Use the Quartus Prime RTL Viewer to see how the Quartus Prime software synthesized your circuit. You can access the RTL viewer on Quartus via Tools -> Netlist Viewers -> RTL Viewer. You can zoom into the various building blocks of your circuit by double-clicking on them, to get more information about their implementation. What are the differences in comparison with Figure 1?

Solution.



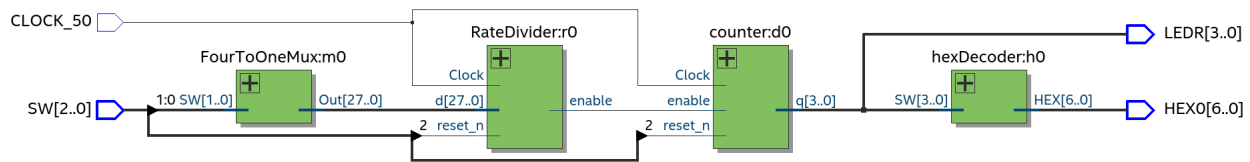
2 Part II

- I no longer need to check the maximum values, due to the limit in the number of digits.
- The following lines of code can be added to Verilog code below:

```
if (q == 4'b1111)
    q <= 1'b0
```
- I expect to see 8. Eagles may see something different.
- 50M requires 26-bit to represent in binary.

1. Draw a schematic of the circuit you wish to build. Work through the circuit manually to ensure that it will work according to your understanding.

Solution.



2. Write a Verilog module that realizes the behaviour described in your schematic. Your circuit should have a clock input and two switches inputs.

Solution.

```
1 module flashes(SW, CLOCK_50, HEX0, LEDR);
2     input [2:0] SW;           //SW[1:0] for rate selection, SW[2] for reset_n.
3     input CLOCK_50;
4     output [6:0] HEX0;        //Display the result in base 10.
5     output [3:0] LEDR;        //Displaying the binary result.
6     wire enable;              //Whether to enable the counter.
7     reg [27:0] count;         //28 comes from log_2 <199_999_999> = 27.57.
8
9     always @(*)
10    begin
11        case (SW[1:0])
12            2'b00: count[27:0] = 28'd0;
13            2'b01: count[27:0] = 28'd49_999_999;
14            2'b10: count[27:0] = 28'd99_999_999;
15            2'b11: count[27:0] = 28'd199_999_999;
16            default: count[27:0] = 28'd0;
17        endcase
18    end
19
20    RateDivider r0(
21        .Clock(CLOCK_50),
22        .reset_n(SW[2]),
23        .enable(enable),
24        .d(count[27:0])
25    );
26    counter d0(
27        .Clock(CLOCK_50),
28        .enable(enable),
29        .reset_n(SW[2]),
30        .q(LED[3:0])
31    );
```

```

31 );
32 hexDecoder h0(
33     .SW(LED[3:0]),
34     .HEX(HEX0[6:0])
35 );
36 endmodule
37
38 module counter(Clock, enable, reset_n, q);
39     input Clock;
40     input enable;
41     input reset_n;
42     output [3:0] q;
43     reg [3:0] count;
44
45     always @(posedge Clock, negedge reset_n)
46     begin
47         if (reset_n == 1'b0)
48             count[3:0] <= 4'b0000;
49         else if (enable == 1'b1)
50             begin
51                 if (count[3:0] == 4'b1111)
52                     count[3:0] <= 4'b0000;
53                 else
54                     count[3:0] <= count[3:0] + 4'b0001;
55             end
56         end
57     assign q[3:0] = count[3:0];
58 endmodule
59
60 module RateDivider(Clock, reset_n, enable, d);
61     input [27:0] d;
62     input reset_n;
63     input Clock;
64     output enable;
65     reg [27:0] d_old;
66     reg [27:0] count;
67
68     always @(posedge Clock, negedge reset_n)
69     begin
70         if (reset_n == 1'b0)
71             count[27:0] <= d[27:0];
72         else if (count[27:0] == 28'd0)
73             count[27:0] <= d[27:0];
74         else if (d[27:0] != d_old[27:0])
75             begin
76                 count[27:0] <= d[27:0];
77                 d_old[27:0] <= d[27:0];
78             end
79         else
80             count[27:0] <= count[27:0] - 28'd1;
81     end
82     assign enable = (count[27:0] == 28'd0) ? 1'b1 : 1'b0;
83 endmodule
84
85 module hexDecoder (SW, HEX);

```

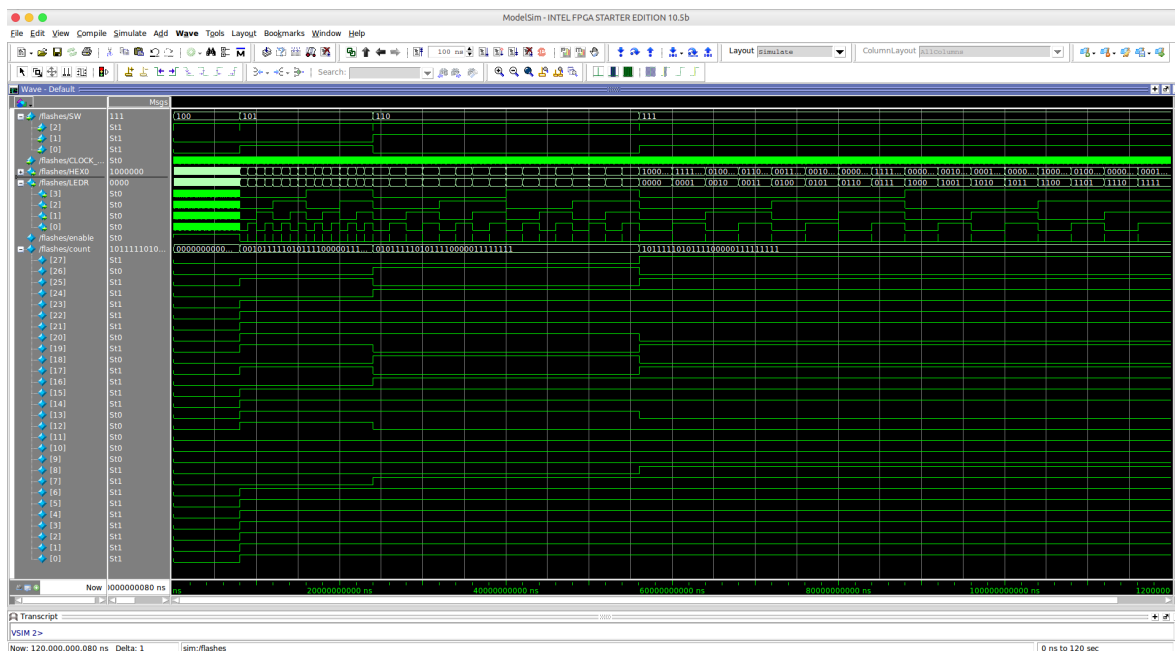
```

86  input [3:0] SW;
87  reg [6:0] result;
88  output reg [6:0] HEX;
89
90  always @(*)
91  begin
92      case (SW[3:0])
93          4'b0000: HEX[6:0] = 7'b1000000;
94          4'b0001: HEX[6:0] = 7'b11111001;
95          4'b0010: HEX[6:0] = 7'b0100100;
96          4'b0011: HEX[6:0] = 7'b0110000;
97          4'b0100: HEX[6:0] = 7'b00111001;
98          4'b0101: HEX[6:0] = 7'b0010010;
99          4'b0110: HEX[6:0] = 7'b0000010;
100         4'b0111: HEX[6:0] = 7'b11111000;
101         4'b1000: HEX[6:0] = 7'b0000000;
102         4'b1001: HEX[6:0] = 7'b0010000;
103         4'b1010: HEX[6:0] = 7'b0001000;
104         4'b1011: HEX[6:0] = 7'b0000011;
105         4'b1100: HEX[6:0] = 7'b1000110;
106         4'b1101: HEX[6:0] = 7'b0100001;
107         4'b1110: HEX[6:0] = 7'b0000110;
108         4'b1111: HEX[6:0] = 7'b0001110;
109         default: HEX[6:0] = 7'b1000000;
110     endcase
111 end
112 endmodule

```

3. Simulate your circuit with ModelSim for a variety of input settings, ensuring the output waveforms are correct. You must include screenshots of simulation output in the prelab. You will also need to think about how to simulate this kind of circuit. For example, how many 50 MHz clock pulses will you need to simulate to show that the RateDivider is properly outputting a 1 Hz pulse?

Solution.



3 Part III

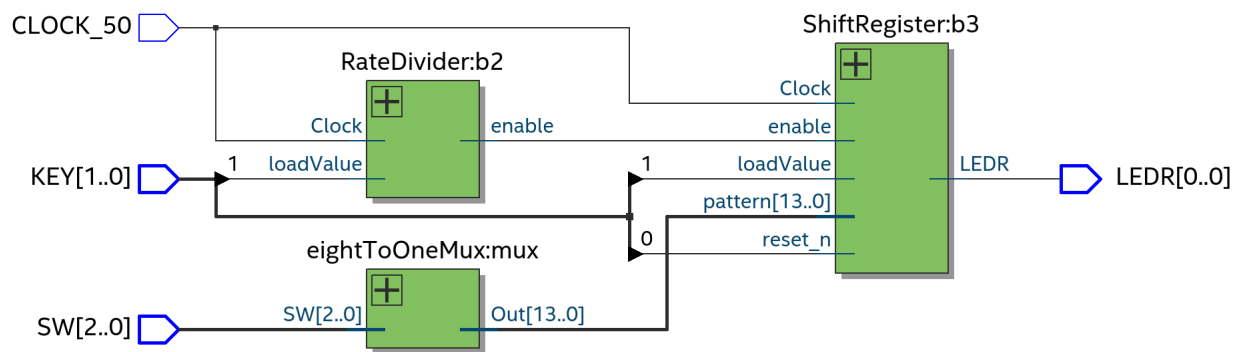
1. Use Table 1 to determine your codes and bit-width.

Solution.

Letter	Morse Code	Pattern Representation (Pattern is 14 its)
S	10101000000000
T	—	11100000000000
U	... —	10101110000000
V —	10101011100000
W	. — —	10111011100000
X	— . . —	11101010111000
Y	— . — —	11101011101110
Z	— — . .	11101110101000

2. Design your circuit by first drawing a schematic of the circuit. Think and work through your schematic to make sure that it will work according to your understanding. You can use Figure 4 as your starting point. You should include the schematic in your prelab.

Solution.



3. Write a Verilog module that realizes the behaviour described in your schematic. You should also include the Verilog code as part of your prelab.

Solution.

```

1 module Morse(LED, SW, KEY, CLOCK_50);
2     input [2:0] SW;
3     input [1:0] KEY;
4     input CLOCK_50;
5     output [0:0] LED;
6     reg [13:0] Pattern;
7     wire flash;
8
9     always @(*)
10    begin
11        case(SW[2:0])
12            3'b000: Pattern[13:0] = 14'b10101000000000;
13            3'b001: Pattern[13:0] = 14'b11100000000000;
14            3'b010: Pattern[13:0] = 14'b10101110000000;
15            3'b011: Pattern[13:0] = 14'b10101011100000;
16            3'b100: Pattern[13:0] = 14'b10111011100000;
17            3'b101: Pattern[13:0] = 14'b11101010111000;
18            3'b110: Pattern[13:0] = 14'b11101011101110;

```



```

19         3'b111: Pattern[13:0] = 14'b11101110101000;
20         default: Pattern[13:0] = 14'b000000000000000;
21     endcase
22 end
23
24 RateDivider b2(
25     .enable(flash),
26     .loadValue(KEY[1]),
27     .Clock(CLOCK_50)
28 );
29
30 ShiftRegister b3(
31     .LEDR(LED[0]),
32     .enable(flash),
33     .Clock(CLOCK_50),
34     .reset_n(KEY[0]),
35     .loadValue(KEY[1]),
36     .pattern(Pattern)
37 );
38 endmodule
39
40 module RateDivider(enable, loadValue, Clock);
41     input loadValue, Clock;
42     output enable;
43     reg [24:0] count;
44
45     always @(posedge Clock)
46     begin
47         if (loadValue == 1'b0)
48             count <= 25'd24_999_999;
49         else
50             begin
51                 if (count == 9'd0)
52                     count <= 25'd24_999_999;
53                 else
54                     count <= count - 25'd1;
55             end
56         end
57     assign enable = (count == 25'd0) ? 1 : 0;
58 endmodule
59
60 module ShiftRegister(LED, enable, Clock, reset_n, loadValue, pattern);
61     input enable, Clock, reset_n, loadValue;
62     input [13:0] pattern;
63     output reg LED;
64     reg [13:0] Pattern;
65
66     always @(posedge Clock, negedge reset_n)
67     begin
68         if (reset_n == 1'b0)
69             begin
70                 LED <= 1'b0;
71                 Pattern <= 14'b0;
72             end
73         else if (loadValue == 1'b0)

```

```

74         begin
75             LEDR <= 1'b0;
76             Pattern <= pattern;
77         end
78     else if (enable == 1)
79         begin
80             LEDR <= Pattern[13];
81             Pattern[13:0] = {Pattern[12:0], 1'b0};
82         end
83     end
84 endmodule

```

4. Simulate your circuit with ModelSim for a variety of input settings, ensuring the output waveforms are correct. If you cannot compile your design to simulate it, then you should at the very least (a) decide what are the inputs settings you want to model and (b) draw waveforms of what you expect your output signals to be for those inputs.

Solution.

