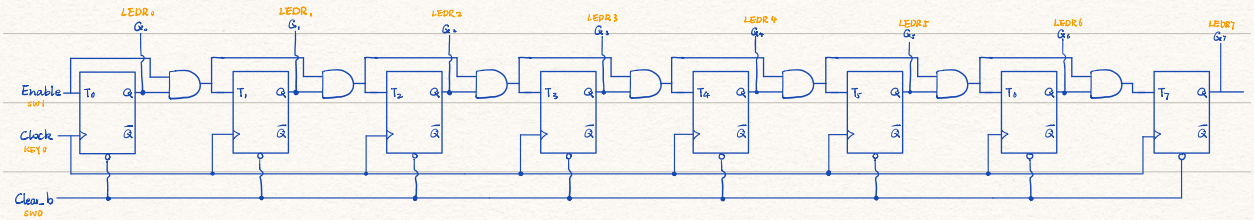
Part 1

4.1 Draw schematic for a 8-bit counter.

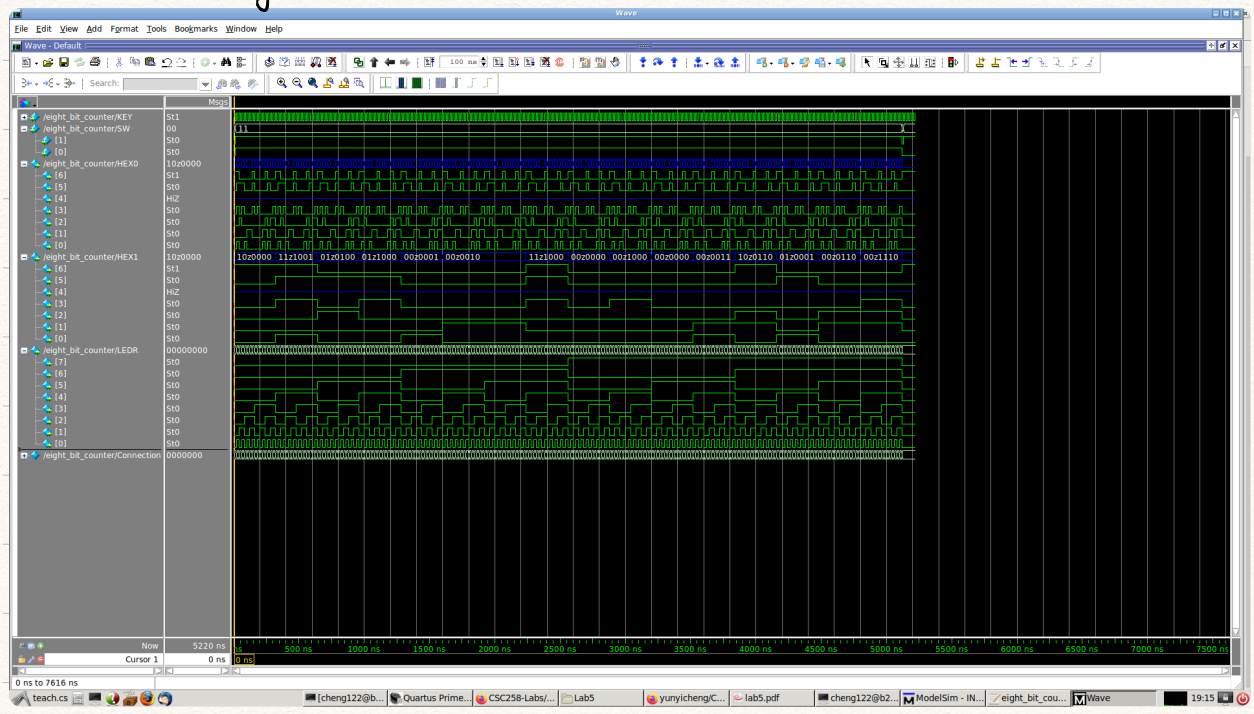
4.2 Annotate all Q outputs of schematic with the bit of the counter ($Q_7 Q_6 Q_5 Q_4 Q_3 Q_2 Q_1 Q_0$) that they correspond to.



4.3 Write Verilog code according to the schematic.

See eight_bit_counter.v

4.4 Simulate your code in ModelSim



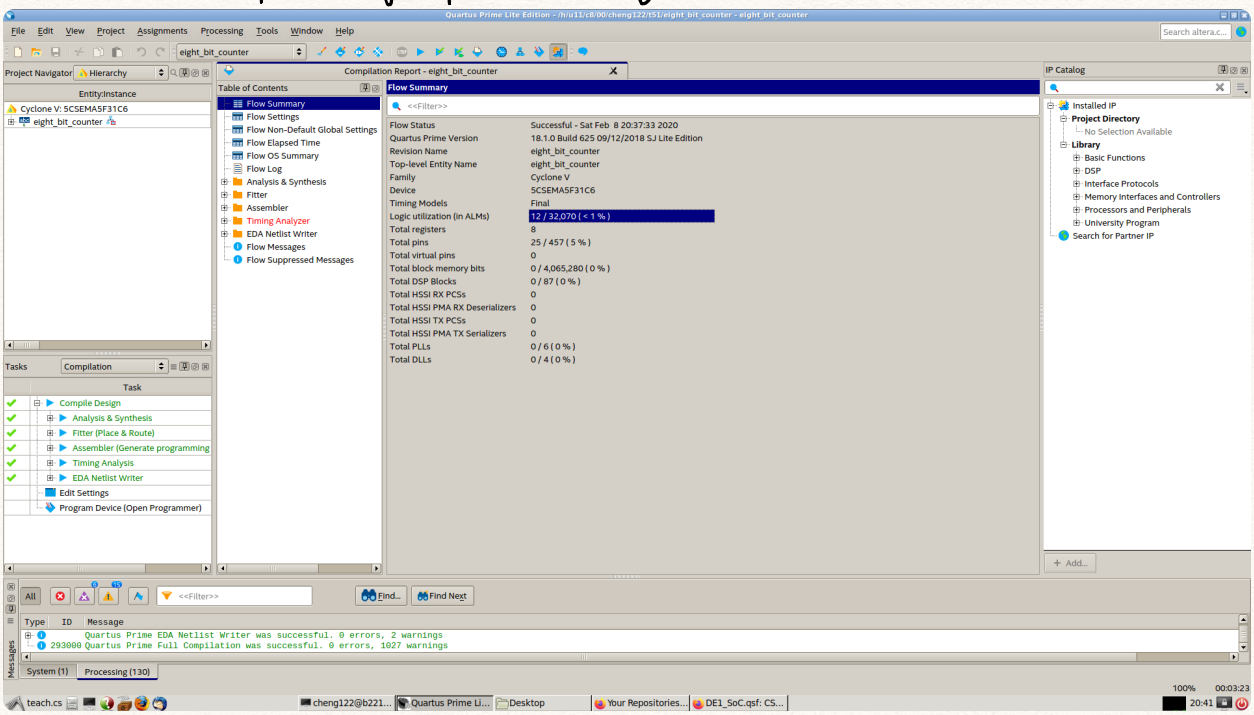
4.5 Augment your code and test it again

the augmented version is eight_bit_counter.v

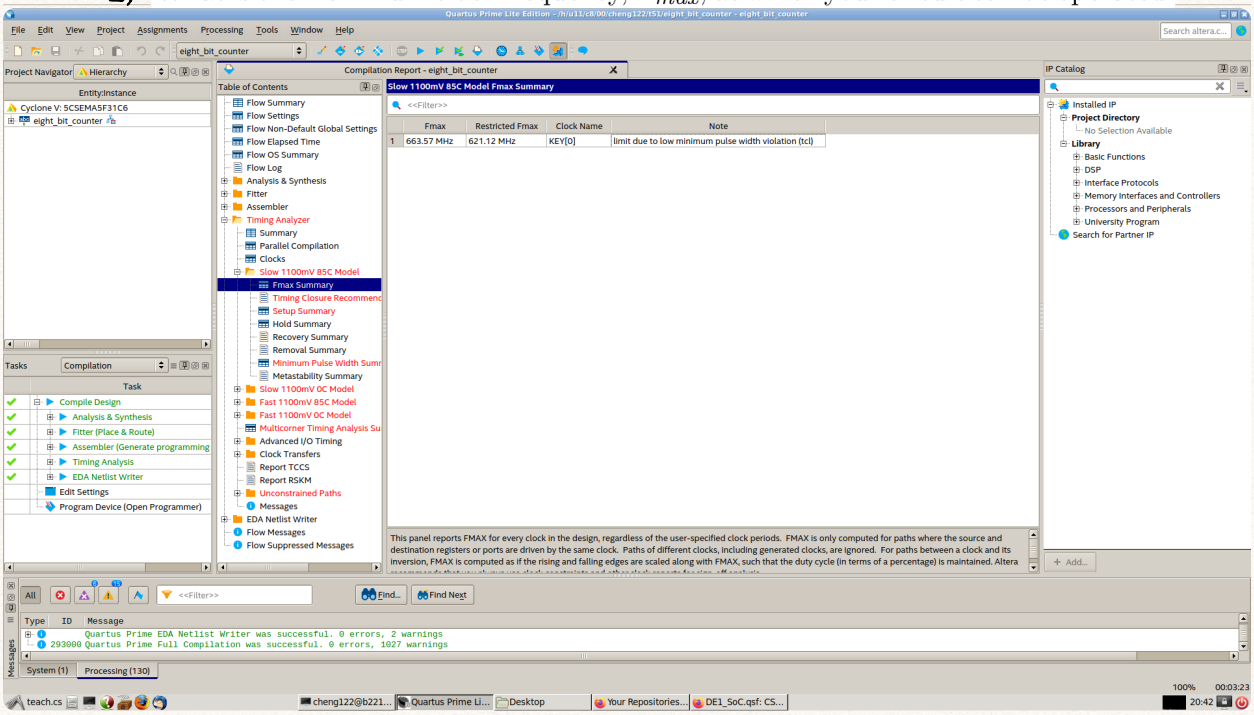
simulation is already done in 4.4

4.6 Create a new Quartus Prime project, compile and answer

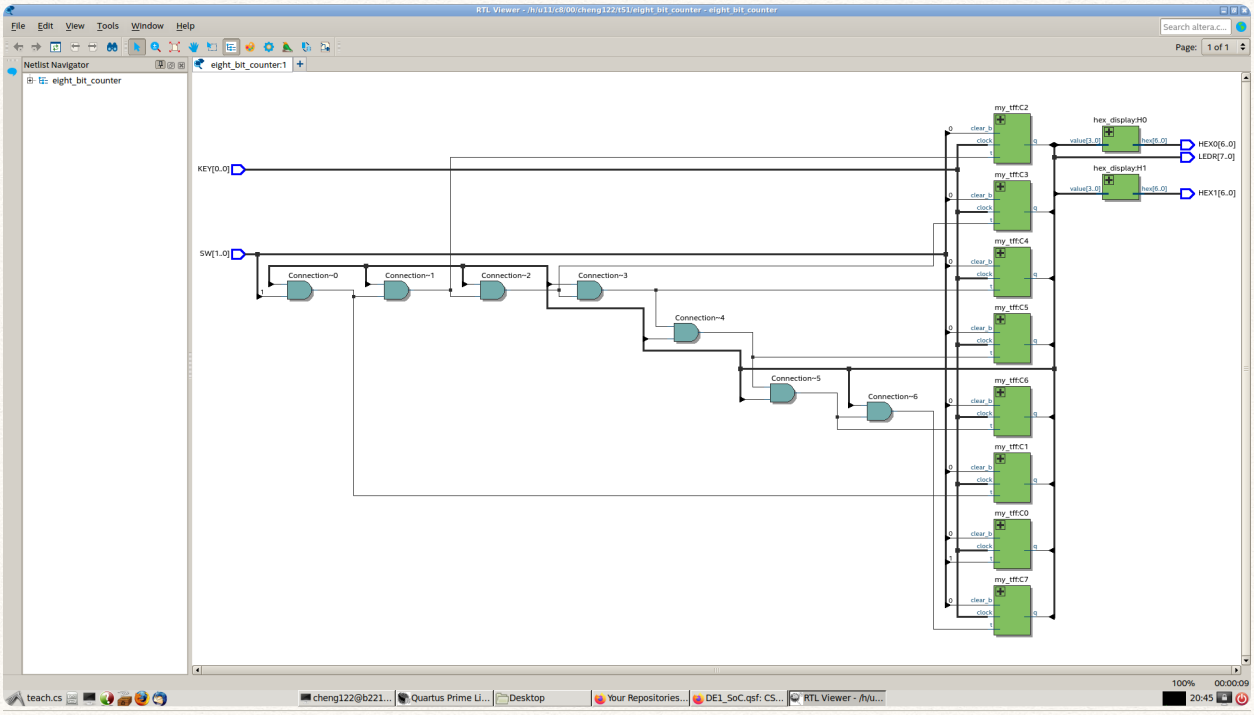
1) What percentage of FPGA logic are used ?



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



4.7 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? **(PRELAB)**



Part 2

In this implementation, q is declared as a 4-bit value, which makes this a 4-bit counter. The check for the maximum value is not necessary in the example above. Why? **(PRELAB)**If you wanted this 4-bit counter to count from 0-9, what would you change? **(PRELAB)**

- Because the limit in digit naturally limits the maximum value
-

```
wire [3:0] d;                    // Declare d
wire clock;                      // Declare clock
wire reset_n;                    // Declare reset_n
wire par_load, enable;           // Declare par_load and enable
reg [3:0] q;                     // Declare q

always @(posedge clock)          // Triggered every time clock rises
begin
    if (reset_n == 1'b0)         // When reset_n is 0
        q <= 0;                  // Set q to 0
    else if (par_load == 1'b1)   // Check if parallel load
        q <= d;                  // Load d
    else if (enable == 1'b1)     // Increment q only when enable is 1
        begin
            if (q == 4'b1001)    // When q is the maximum value for the counter
                q <= 0;          // Reset q into 0
            else                 // When q is not the maximum value
                q <= q + 1'b1;   // Increment q
        end
end
```

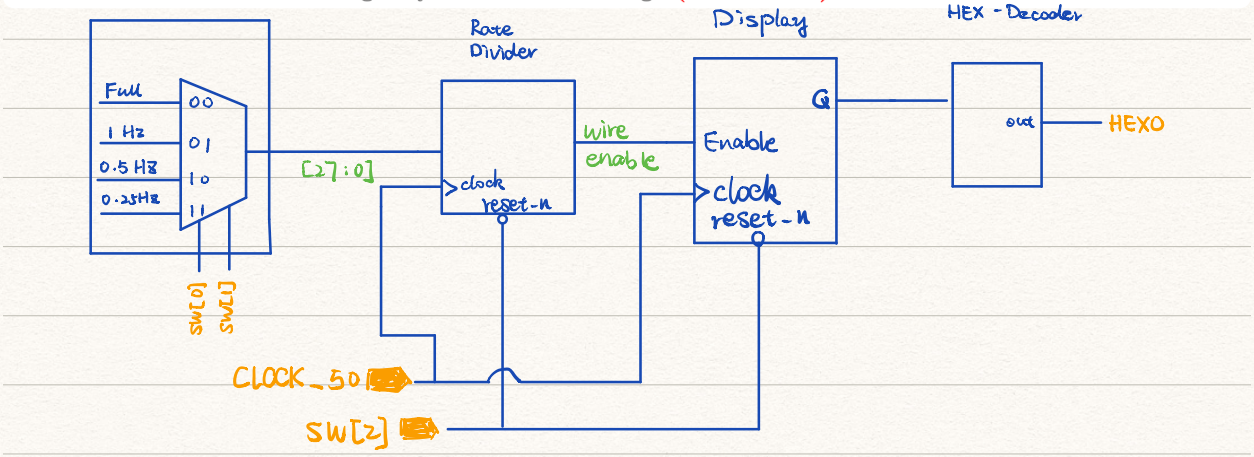
Full speed means that the display flashes at the rate of the 50 MHz clock provided on the DE1-SoC board. At this speed, what do you expect to see on the display? **(PRELAB)**(HINT: compute the period of that clock.)

- I expect to see [-] , since all segments in HEX are flashing too fast to capture whether they are on or off. To human eyes, all segments would seem to be lit up.

To derive the slower flashing rates you should use a special kind of counter, that we will call it **Rate-Divider**, which is running with the MHz clock. The output of RateDivider will be a slower alternating signal that can be used as the enable signal of another module. Every time RateDivider has counted the appropriate number of clock edges, a pulse will be generated for one clock cycle. Figure 3 shows a timing diagram for a 1 Hz Enable/pulse signal with respect to a 50 MHz clock. This pulse signal will be used to control the enable signal of your main 0 to F counter. How large a counter is required to count 50 million clock cycles? **(PRELAB)**How many bits would you need to represent such a value? **(PRELAB)**

- 26 bits
- But we need to count to million x 4 , so we need 28 bits number to count down.

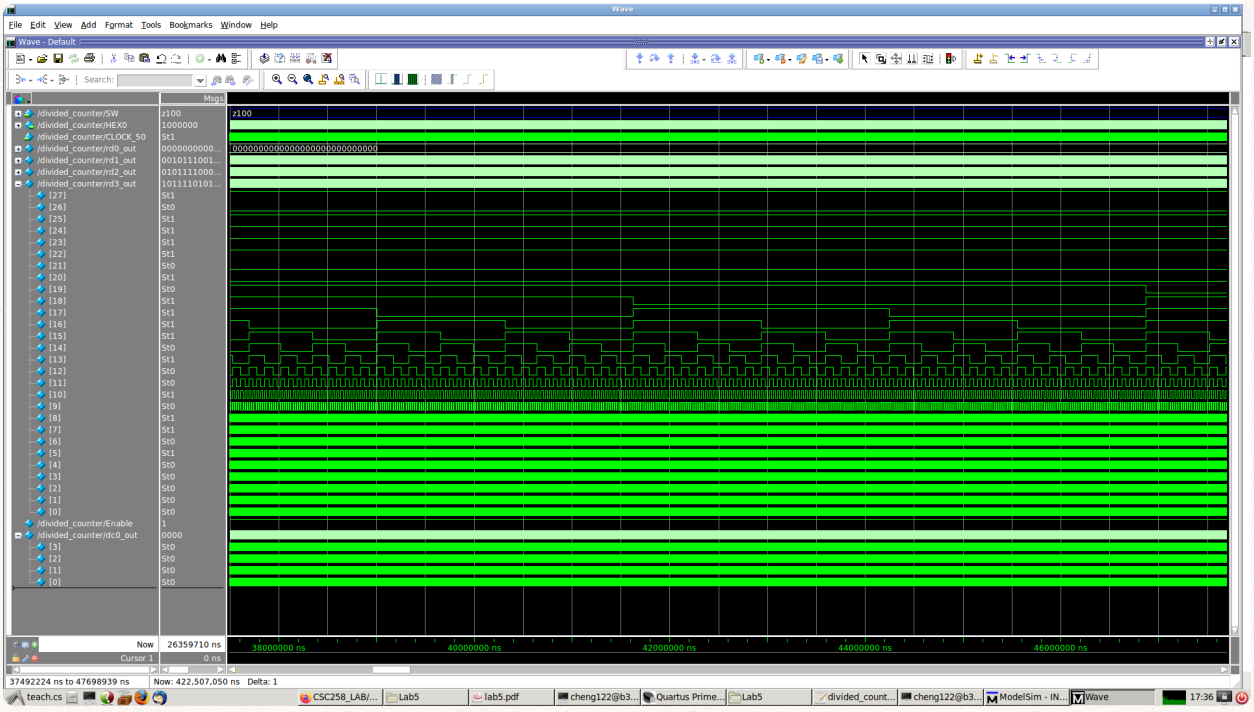
5.1. Draw a schematic of the circuit you wish to build. Work through the circuit manually to ensure that it will work according to your understanding. **(PRELAB)**



5.2. Write a Verilog module that realizes the behaviour described in your schematic. Your circuit should have a clock input and two switches inputs. **(PRELAB)**

see divided_counter.v

5.3. 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? **(PRELAB)**



Part 3

6.1 Use the table to determine code and bit-width

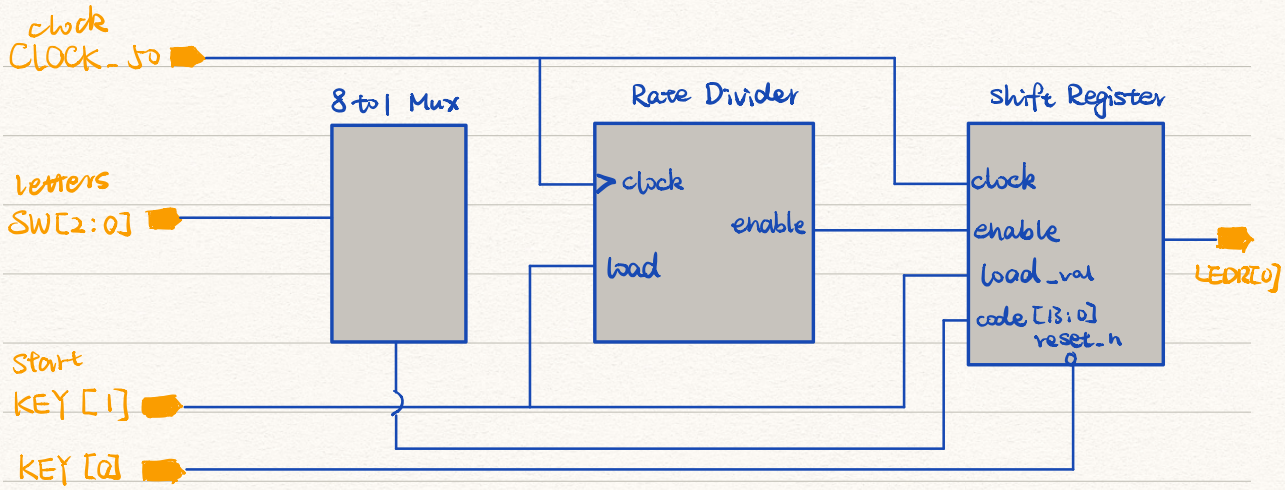
Fill in Table 1 below as part of your prelab. You will need to decide on the pattern length. Complete the pattern representation for letter X with as many zeros as needed based on the pattern length you chose. **(PRELAB)**

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

Table 1: Morse Pattern Representation with fixed bit-width **(PRELAB)**

6.2

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. **(PRELAB)**



6.3

Write a Verilog module that realizes the behaviour described in your schematic. You should also inlucde the Verilog code as part of your prelab. **(PRELAB)**

HINT: You should name your inputs and outputs in a way that makes it easy to interpret your simulations (*e.g.* use input/output names from your schematic). You can later enclose your module inside of another module that will connect inputs and outputs to *SW*, *KEY*, and *LEDR* so that Quartus can correctly assign them to appropriate FPGA pins.

see morse_code.v