Lab Assignment #2

Guideline

This lab is to be done individually. Each person does his/her own assignment and turns it in. Optional starter code for Problem 3 is provided on Canvas.

Objective

To learn designing basic sequential circuits in Verilog and implementing them on an FPGA.

Problem 1: Excess-3 code converter design

In this problem, you will be designing an FSM using three different styles of Verilog coding: behavioral, dataflow, and structural. The following is the problem for which you will be designing the FSM:

A sequential circuit has one input (*X*), a clock input (CLK), and two outputs (*S* and *V*). X, S and V are all one-bit signals. *X* represents a 4-bit binary number *N*, which is input least significant bit first. *S* represents a 4-bit binary number equal to *N* + 3, which is output least significant bit first. At the time the fourth input occurs, *V* = 1 if *N* + 3 is too large to be represented by 4 bits; otherwise, *V* = 0. The value of *S* should be the proper value, not a don't care, in both cases. The circuit always resets after the fourth bit of *X* is received. Assume the sequential circuit is implemented with the following state table. The outputs are (S,V). All state changes occur on the falling edge of the clock pulse.

Present State	Next State		Output	
	X=0	X=1	X=0	X=1
S0	S1	S2	1,0	0,0
S1	S3	S4	1,0	0,0
S2	S4	S4	0,0	1,0
S3	S5	S5	0,0	1,0
S4	S5	S6	1,0	0,0
S5	S0	S0	0,0	1,0
S6	S0	S0	1,0	0,1

a. Write a *behavioral Verilog description* using the state table shown above. Compile and simulate your code using the following test sequence:

X = 1011 1100 1101 ß

The first input bit is at the far right. This is the LSB of the first 4-bit value. Therefore, you will be adding 3 to 13, then to 12, and then to 11. While simulating, keep the period of the CLK to be 10ns. Change X 1/4 clock period after the rising edge of the clock.

b. Write a *data flow Verilog description* using the next state and output equations to describe the state machine. You can use Logic Aid to derive the logic equations. Assume the following state assignment:

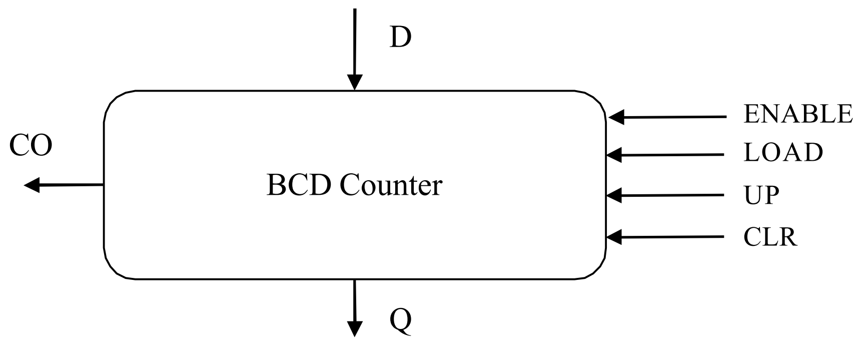
S0 = 000, S1 = 010, S2 = 001, S3 = 101, S4 = 011, S5 = 100, S6=111

Compile and simulate your code using the same test sequence and timing as (a).

c. Write a *structural model* of the state machine in Verilog that contains the interconnection of gates and D flip-flops. Note that the D flip-flop's Verilog implementation should also be structural i.e. implemented using gates and not through behavioral Verilog. Compile and simulate your code using the same test sequence and timing as (a).

Problem 2: BCD Counter Design

Implement a 1 digit BCD (binary coded decimal) counter. It should be a synchronous (4-bit) up/down decade counter with output *Q* that works as follows: All state changes occur on the rising edge of the *CLK* input, except the asynchronous clear (*CLR*). When *CLR* = 0, the counter is reset regardless of the values of the other inputs. You can keep the time period of the CLK signal to 10ns for simulating your design.



On the rising clock edge, the following should take place if the counter is not in reset:

If the *LOAD* = ENABLE = 1, the data input *D* is loaded into the counter.
If *LOAD* = 0 and *ENABLE* = *UP* = 1, the counter is incremented.
If *LOAD* = 0, *ENABLE* = 1, and *UP* = 0, the counter is decremented.

Furthermore, carry out (*CO*) should be implemented using combinational logic using the following criteria:

If *ENABLE* = 1 and *UP* = 1, the carry output (*CO*) = 1 when the counter's value is 9.
If *ENABLE* = 1 and *UP* = 0, the carry output (*CO*) = 1 when the counter's value is 0.
Otherwise, the carry output (*CO*) = 0.

a. Write a Verilog description of the counter. You may implement your design in any style you wish. It will be easier to use a behavioral description which can be either written in the algorithmic way (eg. Count <= Count + 1 – Figure 2.44 in the text) or a state machine way (eg. State <= Next_State – Figure 2.52/2.54 in the text). You may also use dataflow or structural descriptions, although that will be more work. Use the following simulation for your waveforms:

   1. Load counter with 6 (in BCD)
   2. Increment counter four times. You should get 9 and then 0.
   3. Decrement counter once. You should get 9.
   4. Clear the counter.

b. Write a Verilog description of a decimal counter that uses two of the above counters to form a two-decade decimal up/down counter that counts up from 00 to 99 or down from 99 to 00. In other words, instantiate two single digit counters in a top module (the two-digit counter). You may need some extra logic in the top module too other than these instantiations. The top module will have these inputs and outputs: CLR, CLK, ENABLE, LOAD, UP, D1, D2, Q1, D2, CO.

   Use the following criteria for CO:

   If *ENABLE* = 1 and *UP* = 1, the carry output (*CO*) = 1 when the counter's value is 99.
   If *ENABLE* = 1 and *UP* = 0, the carry output (*CO*) = 1 when the counter's value is 00.
   Otherwise, the carry output (*CO*) = 0.

Use the following simulation for your waveforms:

1. Load counter with 97 (in BCD)
2. Increment counter five times.
3. Do nothing for 2 clock periods
3. Decrement counter four times.
4. Clear the counter.

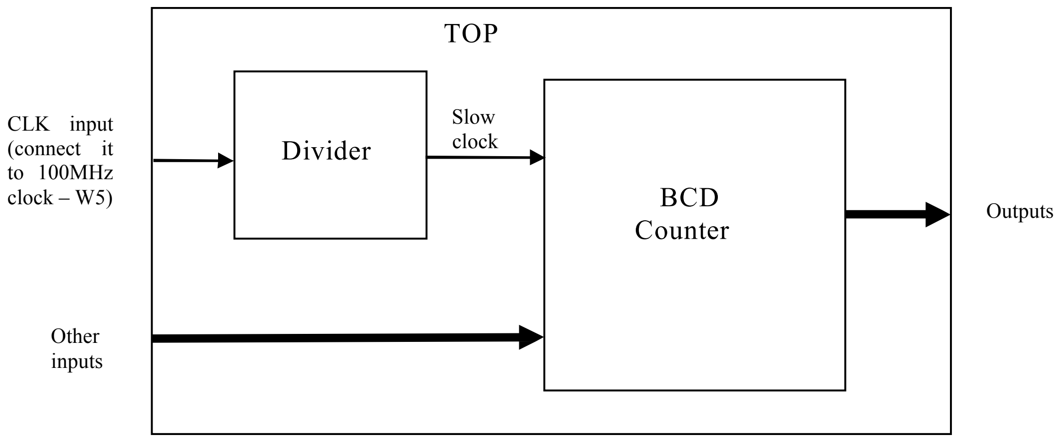
Problem 3: Synthesizing and implementing the BCD counter on the FPGA

Use the code for the single digit BCD counter that you wrote in Problem 2a. Before you synthesize it and implement it on the board, you will have to modify your code a little bit. This is because the CLK signal available on the board is a high frequency signal (100 MHz). If you use this high frequency for your circuit, you will not be able to give proper inputs or see proper outputs to your design.

So, you need to add a clock divider to your Verilog description. Create two more entities in your design. Call one as *top* and another as *divider.* Make connections as shown in the following figure. Look at optional starter code to see how clock dividers are implemented.

To look for latches in your synthesized design, open the utilization report generated by Vivado. From the "Reports" tab in the bottom window, expand "Synth Design" and open "Utilization Report." In the synthesis report, look for "Slice Logic" and see if any latches are being shown.

Also, after adding the counter/clock divider block to your design, simulate the top module in Modelsim before directly synthesizing using Vivado to ensure that the counter/divider works. And while simulating, reduce the large values (like 5000000) in the counter to small values (say 50), so that simulation takes less time and the waveforms are legible. Don't forget to switch to the correct (large) value before synthesizing.



Synthesize the top module (which includes the divider and the 1-digit bcd counter) and use the following pin assignments. Download the design onto the board and make sure it works as expected.

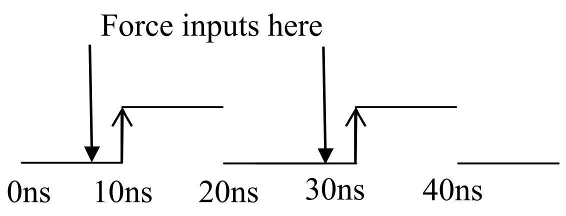
LOAD	BTNU
D	SW[3:0]
ENABLE	SW4
UP	SW5

CLK	W5
COUNT	LED[3:0]
CO	LED4
CLR	SW6

Useful Information

1. Don't limit your testing to the input sequences mentioned with the problem statement. During the checkouts, the TAs will apply several input combinations to test your design. So, make sure to do a thorough testing of your design using sufficient number of inputs.

2. While simulating your design, it is always a good idea to stagger your inputs with respect to the active clock edge. For example, if your active clock edge is occurring at 10ns, apply your inputs sometime before 10ns, say at 8ns. This ensures that when your design was clocked, the input was successfully read. If your active edge occurs at 10ns and your input also changes at 10ns, then it becomes hard to see whether the input was successfully captured by the clock edge or not.



3. A state machine can be designed using either a single always statement (like Figure 2.54 in the text) or using two always statements (like Figure 2.52 in the text). Both ways are correct. However, it is easier to design it using a single always statement. Generally, the single always statement partakes less debugging effort. This is good guideline to observe during the entire semester.

Submission Details

All parts of this lab are to be submitted on Canvas. No hard-copy submission is needed. Please zip all your files into a single folder with the following naming scheme: ***Lastname_Lab#.zip***

Problem	Submission Requirements
1	• Verilog file(s) • Do-file
2	• Verilog file(s) • Do-file
3	• Verilog file(s) • Bit-file and XDC File

Checkout Details

During your checkout you will be expected to demonstrate each of the problems in the assignment and answer verbal questions about the assignment.