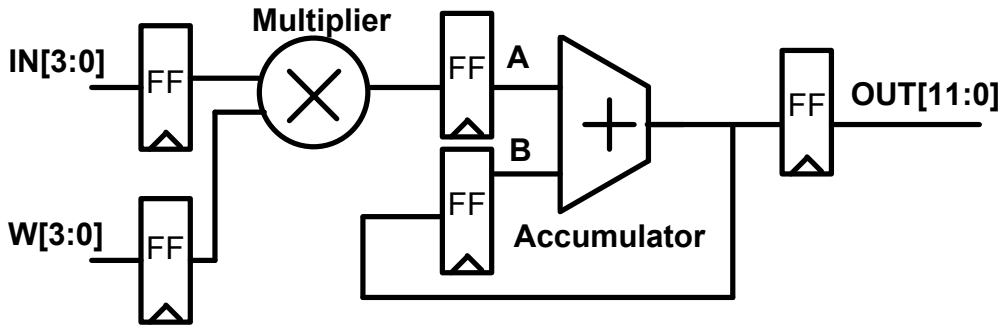
Assignment #5
Comp_Eng 303: Advanced Digital Logic Design

Problem 1. (60 Points) Verilog and Logic Design

1. **Background**: Multiplication-Accumulation (MAC) module is the central computing element for the emerging on-chip neural network accelerator. In this lab, you will design a MAC module with pipelined operation.
2. **Description**:
    a. Inputs: IN (signed 4 bits), W (signed 4 bits, weights for neural network), clk (clock), rstb (resetb, reset at 0);
    b. Output: OUT (signed 12 bits);
    c. At every clock cycle, your MAC will perform IN*WEIGHT and accumulate with its internal stored value from previous cycle to generate the OUT value.
    d. The accumulator should start from 0 value (for example, A and B are set to 0) at the beginning and perform accumulation for 9 cycles (the reason for 9 cycles is that we usually have 3x3 image filter as weight for deep neural network); After that, B starts from 0 again while A takes the new input from Multiplier. Essentially, the accumulator accumulates every 9 clock cycles and restart the accumulation from the next cycle. The OUT will generate final MAC output results every 9 clock cycles (after a few clock cycles' delay). You need to decide how many bits A and B signals should have to accommodate the value range of the total.
    e. The MAC has a pipelined structure as below (FSM or counter is not shown).



    f. The MAC design runs at 1GHz clock;
    g. All inputs (except clk, resetb) need to satisfy a max input delay of 0.5ns and a minimum input delay of -0.2ns from external modules (The negative minimum delay is used to add additional hold margin for the input signal paths);
    h. All output needs to satisfy a max output delay of 0.5ns and a minimum output delay of -0.2ns (The negative minimum delay is used to add additional hold margin for the output signal paths);
    i. Upon reset (resetb =0), all internal flip-flops are reset to 0;

3. Deliverables:

(1). Following the lectures, create all required design files including RTL of MAC, testbench, timing constraint file, and other needed files.

(2). Use Xcelium to simulate your RTL using your testbench. Show the snapshots of the simulated waveforms (show all inputs and outputs signals) for two rounds of operations, e.g. 20 clock cycles. The inputs should change every clock cycle (It is your choices of what IN and W to give, but cannot be all zeros). Show the outputs are calculated correctly.

(3). Perform synthesis following Genus tutorial. Report your area, cell counts, and timing slack. Attach snapshot of report_area and report_timing output.

(4). Repeat the step (2) by simulating your generated gate level netlist. Correct functions should be shown.

(5). Please attach all supporting files in your submission including RTL, testbench, SDC file. They can be pasted into the same file of your report or be attached to your submission as separate files.

(6). Hint: For setting input delay and output delay of the ports, an example of commands to use in your SDC file is below. You may need to replace the port and clock names with the names in your design.

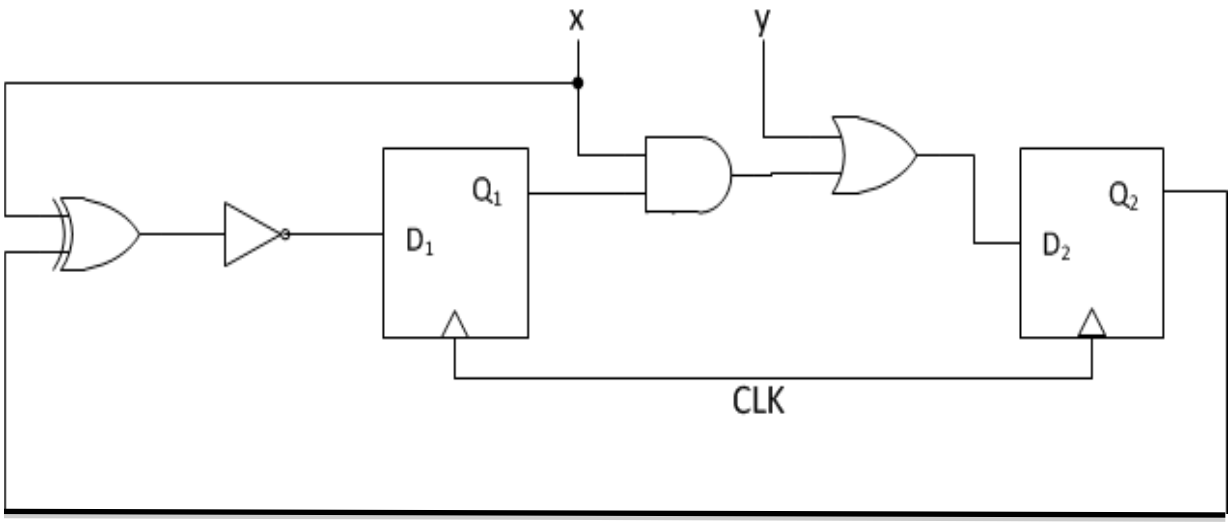
```
set_input_delay -max 0.5 -clock clk [get_ports {IN W}]
set_input_delay -min -0.2 -clock clk [get_ports {IN W}]
set_output_delay -max 0.5 -clock clk [get_ports {OUT}]
set_output_delay -min -0.2 -clock clk [get_ports {OUT}]
```

Problem 2. (10 Points) Sequential Circuits

Use the circuit diagram in Figure below with the following parameters:

- **t_{pcq}= 10 ps**
- **5 ps < $t_{p,\ AND}$ < 10 ps**
- **5 ps < $t_{p,\ OR}$ < 10 ps**
- **10 ps < $t_{p,\ XOR}$ < 20 ps**
- **2 ps < $t_{p,\ INV}$ < 5 ps**

Both flip-flops are identical rising-edge DFFs. Assume the values of x and y are always stable from one rising clock edge to the next rising clock edge.



(a) What is the maximum setup time of the flip-flops in order to safely use a 50ps clock?

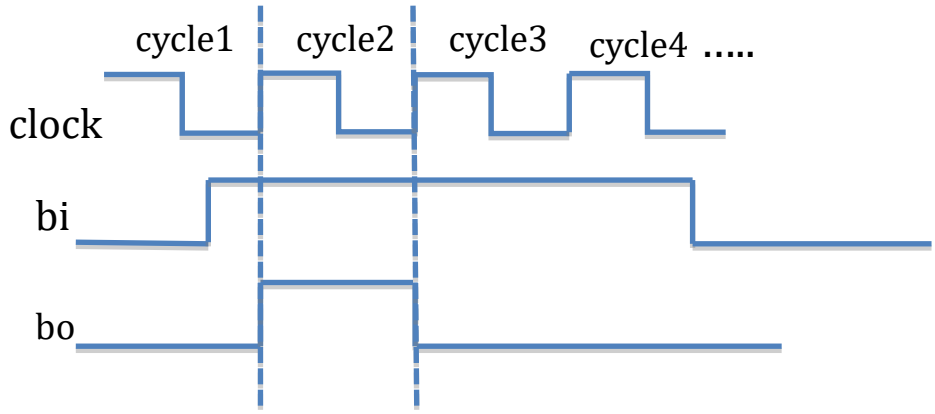
(b) What is the maximum hold time of the flip-flops in order to avoid hold violation?

Problem 3. (30 Points) FSM Design

Design a Finite State Machine for a Button Press Synchronizer. This circuit should synchronize a button press to a clock signal, such that when the button is pressed, the output of this circuit is a signal that is '1' exactly for one clock cycle. Such a synchronized signal is useful to prevent a single button press that lasts multiple cycles from being interpreted as multiple button presses.

The circuit's input will be a signal b_i and the output a signal b_o. When b_i becomes 1, representing the button being pressed, the system should set b_o to 1 for exactly one cycle. The system waits for b_i to return to 0 again, and then waits for b_i to become 1 again, which would represent the next pressing of the button.

A representative timing diagram of this behavior is shown below:



Part a. Derive the FSM diagram. Refer to the lecture slides and the reading material (FSM Tutorial.pdf) posted on Canvas if you need pointers to how to convert a verbal description into a state diagram.
Part b. Derive the transition table and assign binary codes of your choice to states.
Part c. From this table derive the next state and output logic expression and draw the resulting circuit diagram of the sequential circuit.
Part d. Implement the same FSM in Verilog using always statements and verify the behavior shown in the timing diagram above is matching your Verilog simulation.