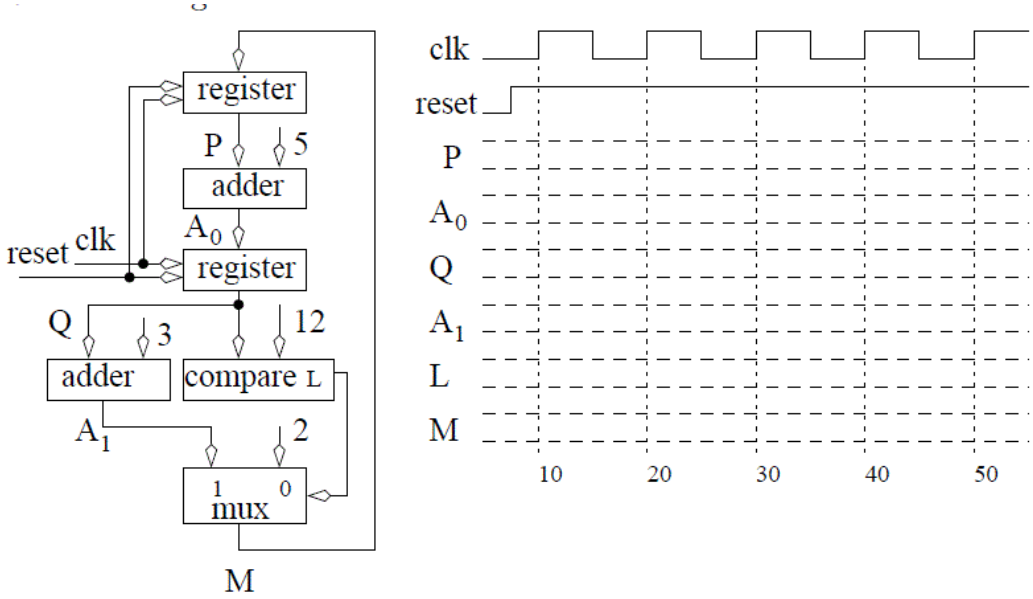
Name: **Section:**

Homework Assignment: submit via gradescope

1. Complete the timing diagram below assuming that the register is positive edge trigger and has an asynchronous active low reset.



2. If the circuit above has the following timings, what is the fastest clock frequency that can be used? (Register T_{CQ} = 3 ns, T_{SU} = 10 ns, T_{HOLD} = 1 ns; Adder T_{PD} = 3 ns; Comparator T_{PD} = 5 ns; MUX T_{PD} = 4 ns)

3. In an ideal world when the button on the FPGA board is pressed, the output go solidly from 0 Volts to 5 volts and back to 0 volts when released. Unfortunately, push-buttons are mechanical devices with contacts that literally bounce when contact is made and broken. This bouncing will cause the voltage to oscillate several times in quick succession.

   Switch bouncing is a major problem in digital circuits because the circuit sees the signal change several times and may take actions appropriate for each of these bounce values when the user only intended a single actions to take place. Your task in this problem is to develop a statemachine based debouncer (with a timing delay), in which every time a button is pressed and release, the output will increment a counter only one time using the "action" signal. It will follow this mini-C algorithm:

```
while(1) {              // infinite loop
        action = 0;                          // reset the action for the button event
        while (button == 0);                 // wait for button press
        for (i=0; i < 20_msec_count; i++);   // delay 20 msec for bouncing to stop
        while (button == 1);                 // wait for button release
```

```
for (i=0; i < 20_msec_count; i++);   // delay 20 msec for bouncing to stop
action = 1;                          // do the action for the button event
}
```

Note1: 20 milli-seconds is usually a sufficient delay for most switch bouncing to end. The world record for the fastest button presses by a human was 16 presses in one second, which corresponds to 62.5 msec, so delaying 20 msec will not miss a button press.

Design a FSM component called **button_debounce** based on the above mini-C code using the methods taught in lesson 10 and the state machine code style taught in lesson 9. Use the lesson 10 test bench HW07_tb.vhdl and lec10.vhdl (the counter to be incremented by your **button_debounce** component, and the button_debounce.vhdl template code.

The entity for your FSM is:

```
entity button_debounce is
        Port(   clk: in  STD_LOGIC;
                reset : in  STD_LOGIC;
                button: in STD_LOGIC;
                action: out STD_LOGIC);
end button_debounce;
```

The given test bench simulates two button push and releases, with oscillations after each push and release. Therefore, if your code works properly, the count should be 2 at the end.

Note2: In the testbench, I slowed the clock from your FPGA's 100 MHz clock down to a 100 KHz clock to greatly speed up the simulation time. However, this obviously impacts the "count" required to reach 20 msec. For your simulation, you can use either a 100 MHz clock (needed for debouncing your future labs) or the 100 KHz clock (to run a faster simulation).

Turn in:

a. The state diagram for your FSM and its datapath, and the FSM output table (hint: edit HW7_design_template.pptx and see button_debounce.vhdl).

b. Upload the vhdl code to GitHub, with an appropriate header and comments.

c. Use the testbench to test your component and include the simulation plot(s). The plots should include a zoomed out plot of the overall simulation, and possibly zoomed in plots of key points (I should be able to see the time that the "action" occurred; each time the test-bench-level counter is counted up; and the timing of the 20 msec delays [can usually see this as a time from a certain state transition to the time the action occurs]). Show the following signals - remove all other signals from the plot(s).
    - clk
    - reset
    - FSM state
    - Button
    - Action
    - Count value

Documentation Statement: For all assignments in this course, you may work with any faculty members or students **currently** enrolled in ECE383 unless otherwise indicated. We expect all graded work, to include software programs, wired circuits, lab notebooks, and written reports, to be your own work. If they aren't, you've copied and will receive **no academic credit** even if the copying is documented. Further, copying without attribution is dishonorable and will be dealt with as a suspected honor code violation. As in all courses, cadets must document any assistance received in the execution of graded work. If you receive no assistance on an assignment, the use of the **Documentation: None** statement is mandatory. If no documentation statement exists, the assignment will be returned for correction and the work will be considered at least one day late.