1) As we saw in class, ripple counters sometimes produces temporary out-of-order outputs because of the propagation delay of the flip-flops. Compile and run the ripple_counter_tb.v file from the Homework 8 zipfile on the course website.

   a. List the sequence of all outputs that are observed when incrementing the counter from 0 to 15, and wrapping back around to 0. Mark the out-of-order outputs with a star.

   b. Explain, based on the bits of the current count and the next count, how many temporary out-of-sequence outputs we will observe during the transition.

   c. Assume that the propagation delay of a T flip-flop is 250 ps. We will run the counter with a periodic clock with period \( x \) (i.e. positive clock edges are separated by a duration of \( x \)). If we want the output to settle into the correct state by a maximum duration of \((0.001)\times x\) after a positive clock edge, what is the minimum period at which we can run the clock? Show your work.

2) As we will see, synchronous counters can run into their own problems when the clock is too fast. Compile and run the sync_counter_tb.v file from the Homework 8 zipfile on the course website.

   a. Reduce the clock pulse width by changing the delay of the always statement near the bottom of the file. What happens to the phase relationship of Q and C as the clock frequency gets higher and higher?

   b. When the frequency gets high enough, the circuit stops working properly. What is the failure mode, and what is the exact minimum clock pulse width that the circuit can take before it stops working?

   c. Explain why the circuit fails the way does.
3) Below is the state diagram for a Moore state machine that controls a keypad entry for a door lock:

![State Diagram]

The keypad has 8 alphabetical buttons labeled a-g, and a “set” button (mounted next to the latch, so it is inaccessible when the door is locked). When a button is pressed, a single clock pulse is generated, along with the corresponding input to the state machine (see Verilog below). No clock pulses are generated unless a button is pressed.

The entire circuit can be reset by an asynchronous reset signal from a watchdog timer: when no buttons have been pressed for 15 seconds, the timer causes the circuit to reset to state S0 (but does not reset the combination, of course).

Normal operation of the lock is as follows: to set the combination from state S0, the user presses the “set” button, followed by the three alphabetical button combination. If the user presses “set” again before three alphabetical buttons have been pressed, the combination does not change.

To open the door lock from state S0, the user simply keys in three alphabetical buttons. If the three buttons correspond to the combination, then the circuit ends up in state S3 and the latch is released; otherwise, the circuit ends up in state S6. The only way to get out of state S3 or state S6 is to wait for the watchdog timer to reset the circuit.
a. What combinatorial module might be useful for converting the 8 alphabetical buttons to a 3-bit input to the circuit?

b. How would comparators be useful for determining certain state transitions in the circuit? Which state transitions would they be used to determine?

c. Explain why five 3-bit storage registers are needed in this circuit. *Hint: what happens if someone tries to set the combination, and then cancels?*

d. Start with the file `combolock.v` from the zipfile, and implement the state machine above using behavioral Verilog. You shouldn’t need to instantiate any other modules. The module is defined as

\[
\text{combolock}(\text{clk}, \text{rst}, \text{set}, \text{btn}, \text{latch}, S, C1, C2, C3)
\]

where `clk` is the clock input, `rst` is the watchdog timer (briefly set to 1 after periods of inactivity), `set` is set to 1 if the “set” button was pressed, `btn` is the 3-bit value of the alphabetical button if one was pressed (in order, from `a=0` to `g=7`; should be ignored when `set=1`), `latch` is the output indicating whether the latch should released (locked=0, unlocked=1), `S` is the current state of the circuit, and `C1`, `C2`, and `C3` are the 3-bit values of the current combination.

Note that the default combination is `g-a-g`. The test bench `combolock_success_tb.v` from the zipfile should demonstrate that your circuit correctly opens and ends up in state `S3`, given the default combination.

e. Create a test bench `combolock_foil_tb.v`, which demonstrates that your circuit ends up in state `S6` after an incorrect combination.

f. Create a test bench `combolock_set_open_tb.v`, which demonstrates setting the combination of the lock to `b-e-e` and subsequently opening the lock by entering that combination.

g. Create a test bench `combolock_combo_idle_tb.v`, which demonstrates that the circuit correctly resets to state `S0` after two correct inputs and a watchdog reset.

h. Create a test bench `combolock_set_cancel_tb.v`, which demonstrates that the circuit correctly resets to state `S0` after hitting “set”, pressing two alphabetical buttons, and hitting “set” again – *and* that the combination does not change during that interaction.