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

Note that the $T$ flip-flops in this design have a simulated propagation delay of 250 ps. This test bench runs the clock at an initial period of 4 ns. The pulse width (duration of a 0 or 1 in the clock cycle) is half of the period, at 2000 ps, or 2 ns.

a. Using your waveform viewer, zoom in until you can closely inspect the counter transitioning from 7 to 8, around $t=30$ ns. What intermediate values of the count $Q$ do we see, and why?

b. How long must we wait after a positive clock edge before reading the value of $Q$, in the worst case?

c. What is the smallest you can make the pulse width on line 65 before the counter starts malfunctioning? You can determine a malfunction when the count $Q$ does not attain the correct value $C$ before the next positive edge of the clock. That is, the counter is broken when there exists some clock cycle during which $C$ is never equal to $Q$.

2. Now compile and run the sync_counter_tb.v file, which implements the synchronous counter we looked at in class. It also models $T$ flip-flops with a propagation delay of 250 ps, as well as AND gates with a delay of 50 ps.

a. At the initial pulse width of 2 ns, do you see any “spurious”, or intermediate values of the count $Q$ during transitions?

b. How long after a positive clock edge must we wait before reading $Q$, in the worst case? In the best case?

c. Is it possible to cause this counter to malfunction by running too fast? (As before, a malfunction is when the true count $C$ bears no overlap to the computed count $Q$.) If so, what is the minimum acceptable pulse width which can be used before it does malfunction?
3. We will design a 4-bit up/down counter which can either increment or decrement a count at a positive edge of the clock. In addition to a clock, your counter should have two inputs $U$ and $D$, which affect the current 4-bit count $Q$ in the following manner:

<table>
<thead>
<tr>
<th>$U$</th>
<th>$D$</th>
<th>$Q_{t+1}$</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>$Q_t$</td>
<td>No change</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>$Q_t - 1$</td>
<td>Decrement</td>
</tr>
<tr>
<td>1</td>
<td>$\times$</td>
<td>$Q_t + 1$</td>
<td>Increment</td>
</tr>
</tbody>
</table>

Note in the table above, the $Q_t + 1$ and $Q_t - 1$ expressions indicate binary arithmetic addition and subtraction (as opposed to Boolean algebraic operations).

a. When counting up (incrementing), we know the rules for toggling $Q_i$ are given as follows:
   
   - For $i = 0$ (LSB), always toggle when incrementing.
   - For $i > 0$ (non-LSB), $Q_i$ toggles when incrementing if $Q_j = 1$ for all $j < i$.

   Let $T_i^+$ be a variable that indicates when $Q_i$ should toggle on increment. Using the rules above, determine Boolean expressions for $T_0^+$ through $T_3^+$. Hint: you might want to look at the synchronous counter for inspiration here.

b. What are the rules for toggling $Q_i$ when decrementing (counting down)?

c. Similar to (a) above, write Boolean expressions for the corresponding set of variables $T_i^-$ based on your answer to (b).

d. Use your answers to (a) and (c) to design a logic diagram for a 4-bit up/down counter, using only four $T$ flip-flops, as well as any additional AND, OR, and NOT gates needed. Hint: the toggle input to the $i^{th}$ flip-flop should be given by

$$T_i = U T_i^+ + U' D T_i^-$$
4. Consider the following state machine with no inputs, and four bits of state $S_{3:0}$, which also constitute its outputs:

![State Machine Diagram]

a. In what ways is this similar to a counter? How is it similar to a shift register?
b. Write down the state transition function as a truth table with 16 rows.
c. Starting at state $S_{3:0} = 0001$, what sequence of states will the state machine pass through before returning to the initial state? Which is the only state among the 16 possible states that is not visited?
d. Why would it be particularly boring to reset this state machine to state $S = 0000$? How many different states would it visit before returning to the initial state?