Lecture 8: Sequential Logic

• Last lecture discussed how we can use digital electronics to do combinatorial logic
  – we designed circuits that gave an immediate output when presented with a given set of inputs
• In many cases, however, we want to do “sequential logic”
  – that means that the output of the circuit should change only when certain other conditions are met
• An example: memory in a computer
  – we want our circuit to store the value of a variable in binary form, and keep that value until we decide to change it
• To achieve this sequential logic, we combine two of the concepts we’ve already talked about:
  – logic gates and feedback
• A simple example is the “Set-Reset (S-R) Latch”:

  ![Set-Reset (S-R) Latch Diagram]

  – Has two outputs, but really only one bit of information
• Let’s set the inputs to $S = 1$, $R = 0$
  – then $\overline{Q} = 0$ and $Q = 1$
• Now we set $S = 0$ (and keep $R = 0$)
  – we then have:
    
    $$Q' = \overline{S + Q} = \overline{0 + 1} = 0$$
    $$Q' = \overline{R + \overline{Q}} = \overline{0 + 0} = 1$$

  • In other words, the output doesn’t change

• Now set $R = 1$ (and keep $S = 0$):
    
    $$Q' = \overline{R + \overline{Q}} = \overline{1 + 0} = 0$$
    $$\overline{Q'} = \overline{S + Q'} = \overline{0 + 0} = 1$$

• Finally, set $R = 0$ (and keep $S = 0$):
    
    $$\overline{Q'} = \overline{S + Q} = \overline{0 + 0} = 1$$
    $$Q' = \overline{R + \overline{Q}} = \overline{0 + 1} = 0$$

  – once again, output doesn’t change
• Note that for two of the four possible input states, the output is uniquely defined:

\[ S = 1, R = 0 \rightarrow \bar{Q} = 0, Q = 1 \]
\[ S = 0, R = 1 \rightarrow \bar{Q} = 1, Q = 0 \]

• But the output for \( S = R = 0 \) depends on what happened earlier
  – that’s what we mean by “sequential logic”
  – the circuit “remembers” whether \( S \) or \( R \) was set to 1 most recently

• I didn’t mention the last possible input state, \( S = 1, R = 1 \)
  – that turns out not to be useful
Gated S-R Latch

- The S-R latch is able to store information, but is still an asynchronous device
  - that is, its outputs respond immediately when either S or R is set to 1
- For better control, we want to be able to set S and R, and then at some later time tell the latch to set its final state
- So, we do a simple modification to come up with the “gated S-R latch”:
- With this circuit, nothing happens until the “enable” line is set to 1
- In other words, the truth table for this device is:

<table>
<thead>
<tr>
<th>E</th>
<th>S</th>
<th>R</th>
<th>Q</th>
<th>Q̅</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Q</td>
<td>Q̅</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Q</td>
<td>Q̅</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Q</td>
<td>Q̅</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Q</td>
<td>Q̅</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Q</td>
<td>Q̅</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Means output values don’t change

Not used
**Edge-triggered flip flops**

- With the gated latch, we have the first example of a flip-flop
  - that’s a circuit whose output changes how we want and when we want, and remains constant otherwise
- In many applications, a clock signal is used on the enable line
  - these clock signals can be fanned out to a large number of flip-flops, all of which then change synchronously
- Rather than enabling on a value of the clock signal, we’d often prefer to enable on a transition
  - either 0 $\rightarrow$ 1 or 1 $\rightarrow$ 0
- Devices that do so are called *edge-triggered* flip-flops
• One way to achieve the edge triggering is with the following gates on the enable line:

- At first glance, looks like output will always be 0
- But the inverter takes some time to do the inversion
  - hence the output looks like

Flip-flop can only change value during this small interval
• The previous circuit would only allow the enable line to go to 1 when the clock makes a $0 \rightarrow 1$ ("rising edge") transition.

• Simple modification allows enable on either transition:

• Symbol for this type of flip-flop:

Rising-edge trigger

Falling-edge trigger
**J-K flip-flops**

- The $S-R$ flip-flop still has the problem that one of the possible input states is useless
  - in fact, the circuit would behave unpredictably for $S = R = 1$
- We can avoid this problem by adding a couple of AND gates at the input:

  - With $J=K=0$, the AND outputs are always 0
  - so $R$ and $S$ are 0, and the flip-flop is in the “memory” state (outputs don’t change)
• If we’re in the state with $Q = 1$, and we input $J = 0, K = 1$, then $R = 1, S = 0$, is passed to the $S-R$ latch
  – so it resets $Q$ to 0, just as before
• If we’re in the state with $Q = 0$, and we input $J = 1, K = 0$, then $R = 0, S = 1$, is passed to the $S-R$ latch
  – so it sets $Q$ to 1, just as before
• In fact, the $J-K$ flop behaves differently only when $J$ and $K$ are both 1
  – If $Q$ is 1, then $R = 1, S = 0$, is passed to the $S-R$ latch
    • so $Q$ is reset to 0
  – If $Q$ is 0, then $R = 0, S = 1$, is passed to the $S-R$ latch
    • so $Q$ is set to 1
• Thus the input $J = K = 1$ is now useful
  – it “toggles” the flip-flop to the opposite output
Variants on the $J$-$K$ flip flop

- We can make useful flip-flops with a single input, starting from an edge-triggered $J$-$K$ flip-flop:

- **D-type:**
  - shifts $D$ to $Q$ at each active clock edge

- **T-type:**
  - toggles at active clock edge if $T = 0$, does nothing otherwise
Designing circuits with flip-flops

- Some parameters and effects to consider:
  - Must make sure the input data arrives before the active clock edge (the time difference between the two is known as the setup time). Otherwise these signals may race against each other, giving unpredictable output.
  - Must make sure data maintains its value long enough for result to be propagated to output. This is called the hold time.

- Signal timing should look something like this:

  
  ![Signal timing diagram]

  Setup time is typically ~10ns
  Propagation delay typically ~30ns
Some flip-flop applications

• Consider the following circuit, based on T type flip-flops:

Output changes at every falling clock edge

Output changes whenever first flip-flop goes from 1 to 0

Output changes whenever second flip-flop goes from 1 to 0
So if we look at all three outputs as a function of time, we find:

This circuit counts the number of clock cycles (in binary)
We could keep adding flip-flops to count to higher numbers (more bits)
• Counters like this are called “ripple counters”
  – since input has to “ripple” through a series of flip-flops to get the correct result
• Problem:
  – what if we add more flip-flops so that the sum of the propagation delays is greater than the clock period?
  – The circuit will give incorrect results (i.e., by the time the MSB changes, more clock cycles will have arrived at the input, and some of the lower-bit flip-flops will have already toggled)
Synchronous Counter

- We can avoid those problems by sending the same clock signal to all the flip-flops.

- Second flip-flop toggles only when output of first one is high.

- Third only toggles when both 1\textsuperscript{st} and 2\textsuperscript{nd} are high.
  - note that we’d need a three-input AND gate to add another bit to the counter.