Introduction to Digital Logic

Lecture 17:
Latches
Flip-Flops
Problem w/ Bistables

- Output should have been 0 at end of sequence
- Problem: Glitch was remembered
- Need some way to ignore inputs until they are stable and valid
Latches

- Latches are bistables that include a new **clock input (sometimes called the ‘gate’ or ‘enable’)**
- The clock input will tell the latch when to ignore the inputs (when $C=0$) and when to respond to them (when $C=1$)
Latches

RS Latch (C=0)

C=0 causes S=R=0 and thus Q and Q’ remain unchanged

RS Latch (C=1)

C=1 allows S,R to pass and thus Q and Q’ are set, reset or remain unchanged based on those inputs
Clock Signals

- A clock signal is an alternating sequence of 1’s and 0’s.
- It can be used to help ignore the inputs of a bistable when there might be glitches or other invalid values.
- Idea:
  - When clock is 0, ignore inputs.
  - When clock is 1, respond to inputs.

Sample Clock Signal

\[ f = \frac{1}{T} = 1 \text{ kHz} \]

\begin{align*}
0 & \quad 1 & \quad 0 & \quad 1 & \quad 0 & \quad 1 & \quad 0 & \quad 1 & \quad 0 & \quad 1 \\
0 \text{ ms} & \quad 1 \text{ ms} & \quad 2 \text{ ms} & \quad 3 \text{ ms} & \quad 4 \text{ ms} & \quad 5 \text{ ms}
\end{align*}
Solution with Latches

- $C = 0$ when inputs change
  - In fact, in a real digital system, it is $C$’s transition to 0 that triggers the inputs to change
  - Glitches occur during this time and are filtered
- When $C = 1$, inputs are stable and no glitches will occur

Glitch gets filtered in latch because $C=0$
Latches

• Rule
  – When clock = 0, inputs don’t matter, outputs remain the same
  – When clock = 1, inputs pass to the inner bistable and the outputs change based on those inputs
SR-Latch

- When $C = 0$, $Q$ holds (remembers) its value
- When $C = 1$, $Q$ responds as a normal SR-bistable

<table>
<thead>
<tr>
<th>CLK</th>
<th>S</th>
<th>R</th>
<th>Q</th>
<th>Q'</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x</td>
<td>x</td>
<td>$Q_0$</td>
<td>$Q_0'$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$Q_0$</td>
<td>$Q_0'$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</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>1</td>
<td>illegal</td>
<td></td>
</tr>
</tbody>
</table>
SR-Latch

<table>
<thead>
<tr>
<th>CLK</th>
<th>S</th>
<th>R</th>
<th>Q</th>
<th>Q'</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x</td>
<td>x</td>
<td>Q₀</td>
<td>Q₀'</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Q₀</td>
<td>Q₀'</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</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>1</td>
<td>illegal</td>
<td></td>
</tr>
</tbody>
</table>

When C=0, Q holds its value

S=1, R=0 causes Q=1

S=0, R=1 causes Q=0

S=1, R=0 causes Q=1
RS (SR) Latches

When $C=0$, outputs don’t change no matter what the inputs do.

When $C=1$, outputs change based on inputs.

<table>
<thead>
<tr>
<th>$C$</th>
<th>$S$</th>
<th>$R$</th>
<th>$Q$</th>
<th>$Q'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x</td>
<td>x</td>
<td>$Q_0$</td>
<td>$Q'_0$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$Q_0$</td>
<td>$Q'_0$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</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>1</td>
<td>illegal</td>
<td>illegal</td>
</tr>
</tbody>
</table>
RS (SR) Latches

When $C=0$, ignore inputs

When $C=1$, outputs change based on inputs

<table>
<thead>
<tr>
<th>$C$</th>
<th>$S$</th>
<th>$R$</th>
<th>$Q$</th>
<th>$Q'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x</td>
<td>x</td>
<td>$Q_0$</td>
<td>$Q'_0$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$Q_0$</td>
<td>$Q'_0$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</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>1</td>
<td>illegal</td>
<td>illegal</td>
</tr>
</tbody>
</table>
Adding a Sequence of Numbers

• Back to our example of adding a sequence of numbers
  – RS latches require 2 inputs (S,R) per output bit Q
  – In this scenario, we only have 1-bit of input per output
  – We’ll modify an SR latch to become a latch that can remember 1 input bit

\[ 9, \ 3, \ 2 \]

Just remember initial sum of 2 until 3 arrives.
The data can still loop around and add up again (2+2=4) but if we just remember our output = 2 then the feedback loop will be broken

This logic should remember (i.e. sequential logic) the sum and only update it when the next number arrives
D-Latches

- D-Latches (Data latches) store data when the clock is low and pass data when the clock is high.
- D-Latch is just an SR Latch with the D-input run into the S-input and inverted into the R-input.
D-Latches

Hold Mode

<table>
<thead>
<tr>
<th>C</th>
<th>D</th>
<th>Q</th>
<th>Q'</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x</td>
<td>Q₀</td>
<td>Q₀'</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

When C=0, outputs don’t change no matter what the inputs do

When C=1, outputs change based on inputs

Transparent Mode
D-Latches

• When $C = 0$, $Q = Q_0$
  – Hold mode => $Q$ stays the same

• When $C = 1$, $Q = D$
  – Transparent mode => $Q$ follows $D$
Bistables vs. Latches

Bistables
• No clock input
  – outputs can change anytime the inputs change (including glitches)

Latches
• Clock/Gate/Enable input
  – outputs can only change during clock high/low time
  – Active-hi clock/gate/enable
    • When C=1, outputs respond to inputs
    • When C=0, outputs hold their value
Notation

• To show that Q remembers its value we can put it in the past tense:
  – \( Q = Q_0 \) (Current Value of Q = Old Value of Q)

• OR put it in the future tense
  – \( Q^* = Q \) (Next Value of Q = Current Value of Q)

Indicates “next-value” of Q

<table>
<thead>
<tr>
<th>C</th>
<th>D</th>
<th>Q</th>
<th>Q’</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x</td>
<td>( Q_0 )</td>
<td>( Q_0’ )</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Current Value = Old Value

<table>
<thead>
<tr>
<th>C</th>
<th>D</th>
<th>Q*</th>
<th>Q’*</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x</td>
<td>Q</td>
<td>Q’</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Next Value = Current Value
Adding a Sequence of Numbers

- Suppose we have a sequence of numbers that comes in over time that we want to sum up.
- Possible solution: Route the outputs back to the inputs so we can add the current sum to the input X.
- Problem 1: No way to initialize sum.
- Problem 2: Outputs can race around to inputs and be added more than once per input number.

Outputs feedback to inputs and update them sum more than once per input.
Adding a Sequence of Numbers

Time: 0

• Suppose somehow we can initialize outputs to 0
• Input X = 2
• 2+0 = 2
Adding a Sequence of Numbers

Time: 1

- Output of 2 feeds back around to the inputs
- The $X = 2$ input hasn’t changed
- Adder adds $2+2 = 4$
- It doesn’t wait until $X$ changes to the next number
Adding a Sequence of Numbers

Problem: We have an uncontrolled feedback loop

Time: 2

• Output of 4 feeds back around to the inputs again
• The X = 2 input still hasn’t changed
• Adder adds 2+4 = 6
Adding a Sequence of Numbers

- Add logic at outputs to just capture and remember the new sum until we’re ready to input the next number in the sequence.

This logic should remember (i.e. sequential logic) the sum and only update it when the next number arrives.

Just remember initial sum of 2 until 3 arrives.

The data can still loop around and add up again (2+2=4) but if we just remember our output = 2 then the feedback loop will be broken.
Adding a Sequence of Numbers

• What if we put D-Latches at the outputs
Adding a Sequence of Numbers

- We’ll change X on every clock period

When C=0 => Q* = Q
When C=1 => Q* = D
Adding a Sequence of Numbers

• Since the clock starts off low, the outputs of the latches can’t change and just hold at 0

When C=0 => Q* = Q
When C=1 => Q* = D
Adding a Sequence of Numbers

- When the clock goes high the D goes through to Q and is free to loop back around

When C=0 => Q* = Q
When C=1 => Q* = D
Adding a Sequence of Numbers

• Once it loops back around it will be added again, change the Y value and go through to Z and loop back around again

When C=0 => Q* = Q
When C=1 => Q* = D
Adding a Sequence of Numbers

- This feedback loop continues until the clock goes low again

When C=0 => Q* = Q
When C=1 => Q* = D
Adding a Sequence of Numbers

- When the clock goes low again, the outputs will hold at their current value 8 until the clock goes high.

```
When \( C=0 \) => \( Q^* = Q \)
When \( C=1 \) => \( Q^* = D \)
```
Adding a Sequence of Numbers

• When the clock goes high, the outputs will be free to change and we will get the feedback problem

When \( C = 0 \) => \( Q^* = Q \)
When \( C = 1 \) => \( Q^* = D \)
Adding a Sequence of Numbers

- Latches clearly don’t work
- The goal should be to get **one change of the outputs per clock period**

When C=0 => $Q^* = Q$
When C=1 => $Q^* = D$
Flip-Flops

- New class of sequential devices
- Outputs only change once per clock period
  - Outputs change on either the *positive edges* of the clock or the *negative edges*
Flip-Flops vs. Latches

Flip-Flops

- Edge-Sensitive (Edge-Triggered)
  - Outputs can change only at the edge

Latches

- Level Sensitive
  - Outputs can change whenever clock is at a particular level (1 or 0)

If positive-edge triggered, Q only looks at the input at this instant

Whenever clock is at level 1 the outputs respond to the inputs
Flip-Flops

- Change D Latches to D Flip-Flops
- Change SR Latches to SR Flip-Flops

Triangle at clock input indicates edge-sensitive FF
Flip-Flops

• To indicate negative-edge triggered use a bubble in front of the clock input

Positive-Edge Triggered D-FF

Negative-Edge Triggered D-FF

No bubble indicates positive-edge triggered

Bubble indicates negative-edge triggered
Positive-Edge Triggered SR-FF

- Q only looks at S and R values only on the positive-edge

<table>
<thead>
<tr>
<th>CLK</th>
<th>S</th>
<th>R</th>
<th>Q*</th>
<th>Q”*</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x</td>
<td>x</td>
<td>Q</td>
<td>Q’</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Q</td>
<td>Q’</td>
</tr>
<tr>
<td>↑</td>
<td>0</td>
<td>0</td>
<td>Q</td>
<td>Q’</td>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>↑</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>1</td>
<td>illegal</td>
<td></td>
</tr>
</tbody>
</table>
Positive-Edge Triggered D-FF

- Q looks at D only at the positive-edge

<table>
<thead>
<tr>
<th>CLK</th>
<th>D</th>
<th>Q*</th>
<th>Q’*</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x</td>
<td>Q</td>
<td>Q’</td>
</tr>
<tr>
<td>1</td>
<td>x</td>
<td>Q</td>
<td>Q’</td>
</tr>
<tr>
<td>↑</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Q only samples D at the positive edges and then holds that value until the next edge.
Negative-Edge Triggered D-FF

- Q looks at D only at the negative-edge

<table>
<thead>
<tr>
<th>CLK</th>
<th>D</th>
<th>Q*</th>
<th>Q’*</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x</td>
<td>Q</td>
<td>Q’</td>
</tr>
<tr>
<td>1</td>
<td>x</td>
<td>Q</td>
<td>Q’</td>
</tr>
<tr>
<td>↓</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>↓</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Q only samples D at the negative edges and then holds that value until the next edge.
Sequence Adder

• In our quest to build an adder that can add a sequence of numbers that come in over times, using flip-flops will produce a workable solution.

• Because FF only change on the edge, only 1 value will be added per clock (i.e. only 1 value can go around the loop per clock).
Sequence Adder

- We’ll still change X once per clock
Sequence Adder

- The 0 on Clear will cause Z to be initialized to 0, but then Z can’t change until the next negative edge.
- That means we will just keep adding $0 + 2 = 2$. 

### Diagram

- **X**
  - X0
  - X1
  - X2
  - X3
- **Y**
  - Y0
  - Y1
  - Y2
  - Y3
- **Z**
  - Z0
  - Z1
  - Z2
  - Z3
- **Clock**
  - Clock signal
- **Clear**
  - Clear signal

The timing diagram shows the sequence of events:

- **Time**
  - Events at specific time points:
    - $2$ at time 2
    - $3$ at time 3
    - $9$ at time 9

- **X**
  - Event at time 2
- **Y**
  - Event at time 2
- **Z**
  - Event at time 0
At the negative edge the flip-flops will sample the D inputs and then remember 2 until the next negative edge. That means we will just keep adding $3 + 2 = 5$.
• At the negative edge the flip-flops will sample the D inputs and then remember 5 until the next negative edge
• That means we will just keep adding $9 + 5 = 14$
Sequence Adder

- Finally, at the negative edge the flip-flops will sample the D inputs and then remember 14
Sequence Adder Summary

- Flip-Flops provide a solution that prevents any feedback problems because the outputs only look at the inputs once per clock.
JK Flip-Flop

• SR Flip-Flop suffers from the 1,1 illegal case
• We can change the SR-FF to do something useful when a 1,1 input occurs
• Resulting flip-flop is called a JK Flip-Flop
JK Flip-Flop

• JK-FF is exactly the same as an SR-FF except for the 1,1 case
• J = Set (S), K = Reset (R)
• So if J=1, K=1 that means we’re trying to Set Q=1 and Reset Q=0 at the same time
  – Instead, if Q=0 and we try to set and reset at the same time, only allow the set; since Q is already=0 disallow the reset…thus Q* = 1
  – Similarly, if Q=1 and we try to set and reset at the same time, only allow the reset; since Q is already=1 disallow the set…thus Q*=0
• Thus Q* = Q’
JK Flip-Flop

![JK Flip-Flop Diagram]

<table>
<thead>
<tr>
<th>CLK</th>
<th>S</th>
<th>R</th>
<th>Q*</th>
<th>Q''</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,1</td>
<td>x</td>
<td>x</td>
<td>Q</td>
<td>Q'</td>
</tr>
<tr>
<td>↑</td>
<td>0</td>
<td>0</td>
<td>Q</td>
<td>Q'</td>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>↑</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>1</td>
<td>Q'</td>
<td>Q</td>
</tr>
</tbody>
</table>

Same as SR-FF

Toggle (Invert) Mode
JK Flip-Flop

At this edge, J,K=1,1 so Q inverts