Introduction to Digital Logic

Lecture 18:
Flip-Flops
Flip-Flops vs. Latches

**Bistables**
- Asynchronous
- No clock input

**Latches**
- Asynchronous
- Clock/Enable input
- Level Sensitive
  - Outputs can change anytime Clock = 1

**Flip-Flops**
- Synchronous
- Clock Input
- Edge-Sensitive
  - Outputs change only on the positive (negative) edges

---

**Flip-Flop**

[Diagram showing flip-flop with inputs S, Q, R, and Q']

**Latch**

[Diagram showing latch with inputs S, Q, C, and Q']

**Bistable**

[Diagram showing bistable with inputs S, Q, R, and Q']
Master-Slave D-FF

- One way to build a D-FF is to take 2 D-Latches and chain them together
- Let the clock go directly to one D-FF but be inverted into the other

![Diagram of Master-Slave D-FF]

The Master Latch gets the clock directly, the Slave gets the inverse of the clock

These are latches NOT FF’s…together they form a D-FF
Master-Slave D-FF

- First latch is referred to as master and second is referred to as slave because the input to the slave is the output of the master.
- The slave can only pass what the master “gives” it.

Q_{Master} = D_{Slave}
Master-Slave D-FF

• Important:
  – When Clock = 0 =>
    • Master = Hold mode, Slave = Transparent
  – When Clock = 1 =>
    • Master = Transparent, Slave = Hold Mode

C = 0: Hold Transp.
C = 1: Transp. Hold
Master-Slave D-FF

CLK = 0: Master Holds, Slave Passes $Q_{\text{Master}}$
Master-Slave D-FF

CLK = 1: Master Passes D, Slave remembers its last value, 0
Master-Slave D-FF

CLK = 0: Master remembers the last value, 1
Slave Passes $Q_{\text{Master}}$
Master-Slave D-FF

CLK = 1: Master Passes D, Slave remembers its last value, 1
Master-Slave D-FF

CLK = 0: Master remembers the last value, 0
Slave Passes Q_{Master}
Master-Slave D-FF

CLK = 1: Master Passes D, Slave remembers its last value, 0
Master-Slave D-FF

CLK = 0:  Master remembers the last value, 1
Slave Passes Q_{Master}
CLK = 1: Master Passes D, Slave remembers its last value, 1
CLK = 0: Master remembers the last value, 1 Slave Passes $Q_{\text{Master}}$
Master-Slave D-FF

- Notice that $Q_{\text{Slave}}$ changes on the negative-edge.
- This Master-Slave implements a negative-edge triggered D-FF.
Master-Slave D-FF

- To implement a positive edge-triggered D-FF change the clock inversion
Master-Slave SR-FF

- We can try to implement an SR-FF by using a Master-Slave configuration.
- This configuration will change on the negative edge.

\[ C = 0: \text{Hold} \]
\[ C = 1: \text{Enabled} \]

\[ C = 0: \text{Hold} \]
\[ C = 1: \text{Enabled} \]
Master-Slave SR-FF

- Problem: The master-slave configuration does not implement *truly* edge-triggered device
Problem w/ Master-Slave SR-FF

Master responds to S,R when C=1
Slave passes what's on Master when C=0
Problem: Master remembers the Set which is no longer present at the next negative edge
Problem w/ Master-Slave SR-FF

A truly edge-triggered SR-FF would never have seen the Set

Notice the difference between the truly edge-triggered device and the Master/Slave configuration...A Master/Slave configuration does work to implement an edge-triggered SR-FF
Initializing Outputs

• Need to be able to initialize Q to a known value (0 or 1)
• FF inputs are often connected to logic that will produce values after initialization
• Two extra inputs are often included: PRESET and CLEAR (usually active-low)

Logic

When CLEAR = active
Q* = 0
When PRESET = active
Q* = 1
When NEITHER = active
Normal FF operation

Note: PRE and CLR have priority over normal FF inputs
Initializing Outputs

• To help us initialize our FF’s use a RESET signal
• /RESET signal (active-low) is produced for us
• It starts at Active (0) when power turns on and then goes to Inactive (1) for the rest of time
• When it’s active use it to initialize the FF’s and then it will go inactive for the rest of time and the FF’s will work based on their inputs

/RESET

Active (0) at time=0

Inactive (1) for the rest of time
Initializing Outputs

- Need to be able to initialize Q to a known value (0 or 1)

When /RESET = 0, /CLR is activated and Q is forced to 0
Initializing Outputs

- Need to be able to initialize Q to a known value (0 or 1)

When /RESET = 1, /CLR is inactive and Q looks at D at each clock edge.

\[ Q^* = D \]
Implementing an Initial State

- When /RESET is activated Q’s initialize to 0 and then when it goes back to 1 the Q’s look at the D inputs.

/RESET

Q0

Q1

Forces Q’s to 0 because it’s connected to the CLR inputs.

Once /RESET goes to 1, the FF’s look at the D inputs.