Learning Outcomes

- I understand how a bistable works
  - I understand how a bistable holds, sets, and resets
- I understand the issues that glitches pose to bistables and the need for latches
- I understand the difference between level-sensitive and edge-sensitive
- I understand how to create an edge-triggered FF from 2 latches

Sequential Logic

- Suppose we have a sequence of input numbers on X[3:0] that are entered over time that we want to sum up
- Possible solution: Route the outputs ________________ so we can add the current sum to the input X

BISTABLES, LATCHES, AND FLIP-FLOPS

\[ X[3:0] \quad 9, \ 3, \ 2 \quad \rightarrow \quad 14,5,2 \quad Z[3:0] \]
Sequential Logic

• Suppose we have a sequence of input numbers on X[3:0] that are entered 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 __________________
• Problem 2: Outputs can _______________ to inputs and be added more than once per input number

Possible Solution
Outputs can feedback to inputs and update them sum more than once per input

Sequence Adder

• If X changes __________ then Z should also change once per cycle
• That is why we will use a ________________ to ensure the outputs can only update once per cycle

Sequence Adder

• The 0 on Clear will cause Z to be initialized to 0, but then Z can’t change until the next positive edge
• That means we will just keep adding 0 + 2 = 2
Sequence Adder

- At the edge the flip-flops will sample the D inputs and then remember 2 until the next positive edge.
- That means we will just keep adding $3 + 2 = 5$.

Finally, at the positive edge the flip-flops will sample the D inputs and then remember 14.

Sequential Logic

- But how do flip-flops work?
- Our first goal will be to design a circuit that can remember one bit of information.
- Easiest approach...

But how do you change the input?
- A signal should only have one driver.

SET/RESET BISTABLES
RS (or SR) Bistable

- **Terminology**
  - \[\text{______} = \text{Force output to 1}\]
  - \[\text{______} = \text{Force output to 0}\]

- **Set/Reset Bistable Circuit**
  - A circuit that can set or reset its output...
  - ...but then can remember its current output value once the inputs are removed

- Cross-Connected NOR gates (outputs feed back to inputs)
- When Set = 1, Q should be forced to ____
- When Reset = 1, Q should be forced to ____
- When neither are 1, Q should __________ at its present value

---

Always start your analysis from the output Q and cycle it around the loop
13.17 RS (SR) Bistable

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

• 1,1 combination violates the Q, Q' relationship

13.18 RS (SR) Bistable

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

• 1,1 combination violates the Q, Q' relationship

It cannot be "remembered"...meaning as soon as R or S goes to 0 then it will set and reset; if R and S goto 0 at the same instant, then we will have unpredictable behavior

RS (SR) Bistable

13.19 RS (SR) Bistable

13.20 Waveform Examples

• Waveform for an SR bistable with active-hi inputs (cross-connected NOR gates)

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

0 feeds back

• 1,1 combination violates the Q, Q' relationship

Waveform Examples
A problem with bistables

**MOTIVATION FOR LATCHES**

Glitches

- __________, __________ output values due to differing ______________ of gate inputs
- Complete the waveform below to see an example

![Waveform Diagram]

Glitches

- Bistables will remember input values whether we
  ________________

Clock Signals

- A clock signal is an alternating sequence of __________
- It can be used to help __________ 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
Latches

- Latches are bistables that include a new **input**
- The clock input will tell the latch when to ignore the inputs (when C=0) and when to respond to them (when C=1)

\[ \text{RS Bistable} \]

\[ \text{RS Latch} \]

\[ \text{C=0 causes S=R=0 and thus Q and Q}' \text{ remain unchanged} \]
\[ \text{C=1 allows S,R to pass and thus Q and Q}' \text{ are set, reset or remain unchanged based on those inputs} \]

**Rule for all latches:**
- 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

\[ \text{SR-Latch} \]

\[ \text{CLK} \]

\[ \text{S} \]

\[ \text{R} \]

\[ \text{Q} \]

\[ \text{S}=1, \text{R}=0 \text{ causes } Q=1 \]
\[ \text{S}=0, \text{R}=1 \text{ causes } Q=0 \]
\[ \text{S}=1, \text{R}=0 \text{ causes } Q=1 \]

\[ \text{When C}=0, \text{ Q holds its value} \]

<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>
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

MOTIVATION FOR D-LATCHES

D-Latches

- D-Latches (Data latches) ________ data when the clock is ______ and __________ data when the clock is ______
- D-Latch is just an SR Latch with the D-input run into the S-input and ________ into the R-input

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

D-Latches

- D-Latches (Data latches) ________ data when the clock is ______ and __________ data when the clock is ______
- D-Latch is just an SR Latch with the D-input run into the S-input and ________ into the R-input
D-Latches

Hold Mode

- **D-Latch 7475**
- As clock is LOW, don’t look at the D input
- **Triggering Rule**: The Q output follow the D input (i.e., Q=1) when the clock or gate input is high (i.e., the latch is enabled). When the latch is disabled (Clock = LOW) the output remains put.

- **Complete waveform for Q**

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

D-Latches

Hold Mode

- **Q** = **Q** (Next Value of Q = Current Value of Q)
- When C=0, outputs don’t change no matter what the inputs do
- When C=1, outputs change based on inputs

Notation

- To show that Q remembers its value we can put it in the past tense:
  - Q = Q₀ (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>Q₀</th>
<th>Q₀*</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>

<table>
<thead>
<tr>
<th>C</th>
<th>D</th>
<th>Q₀</th>
<th>Q₀*</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>

Current Value = Old Value

Next Value = Current Value
Adding a Sequence of Numbers

• What if we put D-Latches at the outputs

![Diagram of D-Latches and Adder Circuit]

Adding a Sequence of Numbers

• We’ll change X on every clock period

![Diagram showing change in X at different clock periods]

Adding a Sequence of Numbers

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

![Diagram showing clock starting low]

Adding a Sequence of Numbers

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

![Diagram showing clock going high and D looping back]

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.

Adding a Sequence of Numbers

• This feedback loop continues until the clock goes low again.

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.

Adding a Sequence of Numbers

• When the clock goes high, the outputs will be free to change and we will get the feedback problem.
Adding a Sequence of Numbers

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

There is a diagram showing the timing and circuitry involved in adding a sequence of numbers. The diagram includes inputs and outputs, along with a clock signal to illustrate the timing of the changes.

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

There is a diagram showing the circuitry for bistables, latches, and flip-flops. The flip-flops include labels for their inputs and outputs, as well as a triangle at the clock input to indicate edge-sensitive flip-flops.

Flip-Flops

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

There is a diagram showing the conversion of latches to flip-flops. The circuitry is illustrated with labels for each component, and a triangle at the clock input to indicate edge-sensitive flip-flops.
Flip-Flops

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

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

D FF Example

- Assume positive edge-triggered FF
Shift Register

- A shift register is a device that acts as a ‘queue’ or ‘FIFO’ (First-in, First-Out).
- It can store n bits and each bit moves one step forward each clock cycle
  - One bit comes in the overall input per clock
  - One bit ‘falls out’ the output per clock

BUILDING A FLIP FLOP

When we want to ensure an output updates only ONCE per clock, we need to use flip-flops (not latches or bistables)!
Master-Slave D-FF

- To build an edge-triggered D-FF we can use two D-Latches

\[ \text{These 2 latches form a flip-flop} \]

Complete the Waveform

Master-Slave D-FF

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

\[ \text{Negative-Edge Triggered} \]

\[ \text{Positive-Edge Triggered} \]

INITIALIZING OUTPUTS
Initialize Outputs

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

When CLEAR = active
- Q*=_____
When SET = active
- Q*=____
When NEITHER = active
- _______ FF operation

Note: CLR and SET have _______ over normal FF inputs

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
- Once RESET goes to 0, the FF’s look at the D inputs

To help us initialize our FF’s use a RESET signal
- Generally produced for us and given along with CLK
- It starts at Active (1) when power _________ and then goes to Inactive (0) for the ____________
- 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

Note: CLR and SET have _______ over normal FF inputs
Presets / Clear Example

- Complete the waveform

Using muxes to control when register save data

REGISTER WITH ENABLES

Register Resets/Clears

- When the power turns on the bit stored in a flip-flop will initialize to a random value
- Better to initialize it to a known value (usually 0's)
- Use a special signal called "reset" to force the flip-flops to 0's

<table>
<thead>
<tr>
<th>CLK</th>
<th>RST</th>
<th>D</th>
<th>Q*</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>X</td>
<td>Q_i</td>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>↑</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>↑</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Register Problem

- Whatever the D value is at the clock edge is sampled and passed to the Q output until the next clock edge
- Problem: Register will save data on EVERY edge
  - Often we want the ability to save on one edge and keep that value for many more cycles
Solution

- Registers (D-FF's) will sample the D bit every clock edge and pass it to Q.
- Sometimes we may want to hold the value of Q and ignore D even at a clock edge.
- We can add an enable input and some logic in front of the D-FF to accomplish this.

<table>
<thead>
<tr>
<th>CLK</th>
<th>RST</th>
<th>EN</th>
<th>Dᵢ</th>
<th>Qᵢ</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>Qᵢ</td>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>↑</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>Qᵢ</td>
</tr>
<tr>
<td>↑</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>↑</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

FF with Data Enable (Always clocks, but selectively chooses old value, Q, or new value D)

Registers w/ Enables

- When EN=0, Q value is passed back to the input and thus Q will maintain its value at the next clock edge.
- When EN=1, D value is passed to the input and thus Q will change at the edge based on D.

<table>
<thead>
<tr>
<th>CLK</th>
<th>RST</th>
<th>EN</th>
<th>D[3:0]</th>
<th>Q[3:0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0011</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
</tr>
</tbody>
</table>

4-bit Register w/ Data (Load) Enable

- Registers (D-FF's) will sample the D bit every clock edge and pass it to Q.
- Sometimes we may want to hold the value of Q and ignore D even at a clock edge.
- We can add an enable input and some logic in front of the D-FF to accomplish this.

<table>
<thead>
<tr>
<th>CLK</th>
<th>RST</th>
<th>EN</th>
<th>Dᵢ</th>
<th>Qᵢ</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>Qᵢ</td>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>↑</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>Qᵢ</td>
</tr>
<tr>
<td>↑</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>↑</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

4-bit register with 4-bit wide 2-to-1 mux in front of the D inputs

Registers w/ Enables

- The D value is sampled at the clock edge only if the enable is active.
- Otherwise the current Q value is maintained.
### Counters

- Count (Add 1 to Q) at each clock edge
  - Up Counter: \( Q^* = Q + 1 \)
  - Can also build a down counter as well (\( Q^* = Q - 1 \))
- Standard counter components include other features
  - Resets: Reset count to 0
  - Enables: Will not count at edge if \( EN=0 \)
  - Parallel Load Inputs: Can initialize count to a value \( P \) (i.e. \( Q^* = P \) rather than \( Q+1 \))

### Sample 4-bit Counter

- 4-bit Up Counter
  - RST: a synchronous reset input
  - PE and P, inputs: loads Q with P when PE is active
  - CE: Count Enable
    - Must be active for the counter to count up
  - TC (Terminal Count) output
    - Active when \( Q=1111 \) AND counter is enabled
    - \( TC = EN \cdot Q_3 \cdot Q_2 \cdot Q_1 \cdot Q_0 \)
    - Mealy output
    - Indicates that on the next edge it will roll over to 0000

<table>
<thead>
<tr>
<th>CLK</th>
<th>RST</th>
<th>PE</th>
<th>CE</th>
<th>( Q^* )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>Q</td>
</tr>
<tr>
<td>↑</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>↑</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>P</td>
</tr>
<tr>
<td>↑</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Q+1</td>
</tr>
<tr>
<td>↑</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Q+1</td>
</tr>
</tbody>
</table>
Counter Exercise

CLK
RST
PE
CE
P[3:0]
Q[3:0]

0011
1101
1001

Counter Design

• Sketch the design of the 4-bit counter presented on the previous slides

CLK
D[3:0]
Q[3:0]

01
PE
RST
CLK

TC
Reg
CLR
Q[3:0]