Outline for Today's Lecture

• Recap Latch vs Flip-Flop
• Shift Register
• Counter
D Latch

• Two inputs: \( CLK, D \)
  – \( CLK \): controls when the output changes
  – \( D \) (the data input): controls what the output changes to

• Function
  – When \( CLK = 1 \),
    \( D \) passes through to \( Q \) (\textit{transparent})
  – When \( CLK = 0 \),
    \( Q \) holds its previous value (\textit{opaque})

• Avoids invalid case when
  \( Q \neq \text{NOT} \bar{Q} \)
D Flip-Flop

- **Inputs:** $CLK$, $D$
- **Function**
  - Samples $D$ on rising edge of $CLK$
    - When $CLK$ rises from 0 to 1, $D$ passes through to $Q$
    - Otherwise, $Q$ holds its previous value
  - $Q$ changes only on rising edge of $CLK$
- **Called** *edge-triggered*
- **Activated on the clock edge**

Verilog module flop (input clk, d, output reg q);
always @(posedge clk)
q <= d;
endmodule
D Latch vs. D Flip-Flop

D Latch

D Flip-Flop

CLK

D

Q

Q

CLK

D

Q

Q
Shift Registers

- Shift a new bit in on each clock edge
- Shift a bit out on each clock edge
- *Serial-to-parallel converter*: converts serial input \((S_{in})\) to parallel output \((Q_{0:N-1})\)

**Symbol:**

**Implementation:**
Shift Register with Parallel Load

- When $Load = 1$, acts as a normal $N$-bit register
- When $Load = 0$, acts as a shift register
- Now can act as a *serial-to-parallel converter* ($S_{in}$ to $Q_{0:N-1}$) or a *parallel-to-serial converter* ($D_{0:N-1}$ to $S_{out}$)

Verilog
module counter #(parameter N=8)
    (input clk, reset, load, sin,
     input [N-1:0] d,
     output reg [N-1:0] q,
     output sout);
always @ (posedge clk or posedge reset)
    if (reset) q <= 0;
    else if (load) q <= d;
    else q <= {q[N-2:0], sin};
assign sout = q[N-1];
endmodule
Shift Register Example
Counters

- Increments on each clock edge
- Used to cycle through numbers. For example,
  - 000, 001, 010, 011, 100, 101, 110, 111, 000, 001…
- Example uses:
  - Digital clock displays
  - Program counter: keeps track of current instruction executing

Symbol | Implementation
--- | ---
CLK | ![Counter Circuit Diagram]
Q | ![Counter Circuit Diagram]
Reset | ![Counter Circuit Diagram]

Verilog

```verilog
module counter #(parameter N=8)
  (input clk, reset,
   output reg [N-1:0] q);
always @ (posedge clk or posedge reset)
  if (reset) q <= 0;
  else q <= q + 1;
endmodule
```
Binary Counter Example
Binary Counter with Parallel Load
Counter Design Example

• Design a counter that counts the sequence and repeats
Counter Design Example

• Create Next State Table

<table>
<thead>
<tr>
<th>Present State</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q2 Q1 Q0</td>
<td>D2 D1 D0</td>
</tr>
<tr>
<td>0 0 0</td>
<td>1 0 0</td>
</tr>
<tr>
<td>0 0 1</td>
<td>X X X</td>
</tr>
<tr>
<td>0 1 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>0 1 1</td>
<td>X X X</td>
</tr>
<tr>
<td>1 0 0</td>
<td>1 1 0</td>
</tr>
<tr>
<td>1 0 1</td>
<td>X X X</td>
</tr>
<tr>
<td>1 1 0</td>
<td>1 1 1</td>
</tr>
<tr>
<td>1 1 1</td>
<td>0 1 0</td>
</tr>
</tbody>
</table>
Counter Design Example

• Determine the Next State Logic

\[
\begin{align*}
D2 &= Q1' + Q2Q0' \\
D1 &= Q2 \\
D0 &= Q2Q1Q0'
\end{align*}
\]
Counter Design Example

• Verify it is self-starting where all next states point to a state in sequence.

<table>
<thead>
<tr>
<th>Present State</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q2  Q1  Q0</td>
<td>D2  D1  D0</td>
</tr>
<tr>
<td>0  0  0</td>
<td>1  0  0</td>
</tr>
<tr>
<td>0  0  1</td>
<td>1  0  0</td>
</tr>
<tr>
<td>0  1  0</td>
<td>0  0  0</td>
</tr>
<tr>
<td>0  1  1</td>
<td>0  0  0</td>
</tr>
<tr>
<td>1  0  0</td>
<td>1  1  0</td>
</tr>
<tr>
<td>1  0  1</td>
<td>1  1  0</td>
</tr>
<tr>
<td>1  1  0</td>
<td>1  1  1</td>
</tr>
<tr>
<td>1  1  1</td>
<td>0  1  0</td>
</tr>
</tbody>
</table>

Self-starting
Counter Design Example

• Resulting Logic Design

\[
\begin{align*}
\text{D0} &= \text{Q}_2 \text{Q}_1 \text{Q}_0' \\
\text{D1} &= \text{Q}_2 \\
\text{D2} &= \text{Q}_1' + \text{Q}_2 \text{Q}_0' \\
\end{align*}
\]

Output \(Q_2, Q_1, Q_0\)