Sequential Logic Design

Introduction

- Sequential circuit
  - Output depends not just on present inputs (as in combinational circuit), but on past sequence of inputs
  - Stores bits, also known as having “state”
  - Simple example: a circuit that counts up in binary

- Flight attendant call button
  - Press call: light turns on
  - Press cancel: light turns off
  - Logic gate circuit to implement this?

  - Doesn’t work. Q=1 when Call=1, but doesn’t stay 1 when Call returns to 0
  - Need some form of “feedback” in the circuit
Sequential Logic Design
First attempt at Bit Storage

- We need some sort of feedback
  - Does circuit on the right do what we want?
    - No: Once Q becomes 1 (when S=1), Q stays 1 forever - no value of S can bring Q back to 0

Sequential Logic Design
Simple Example Using SR Latch for Bit Storage

- SR latch can serve as bit storage in previous example of flight-attendant call button
  - Call=1: sets Q to 1
    - Q stays 1 even after Call=0
  - Cancel=1: resets Q to 0

- But, there's a problem...

Sequential Logic Design
Bit Storage Using an SR Latch

- Does the circuit to the right, with cross-coupled NOR gates, do what we want?
  - Yes! How did someone come up with that circuit? Maybe just trial and error, a bit of insight...

Sequential Logic Design
SR Latch

- What value with the output Q have at the indicated time?
  1. 1
  2. 0
  3. Neither
  4. Either

25% 25% 25% 25%
Sequence Logic Design
Problem with SR Latch

Problem
- If \( S=1 \) and \( R=1 \) simultaneously, we don't know what value \( Q \) will take.

\[
\begin{array}{c|c|c}
S & R & Q \\
---&---&--- \\
0 & 0 & 0 \\
0 & 1 & 1 \\
1 & 0 & 0 \\
1 & 1 & \text{undefined} \\
\end{array}
\]

\( Q \) may oscillate. Then, because one path will be slightly longer than the other, \( Q \) will eventually settle to 1 or 0 – but we don’t know which.

Sequential Logic Design
Solution: Level-Sensitive SR Latch

- Add enable input “\( C \)” as shown.
- Only let \( S \) and \( R \) change when \( C=0 \).
- Ensure circuit in front of \( S \) never sets \( SR=11 \), except briefly due to path delays.
- Change \( C \) to 1 only after sufficient time for \( S \) and \( R \) to be stable.
- When \( C \) becomes 1, the stable \( S \) and \( R \) value passes through the two AND gates to the SR latch’s \( S_1 \), \( R_1 \) inputs.

Math.
Though \( SR=11 \) briefly...
...\( S_1R_1 \) never = 11

Sequential Logic Design
Clock Signals for a Latch

- How do we know when it’s safe to set \( C=1 \)?
  - Most common solution – make \( C \) pulse up/down
  - \( C=0 \): Safe to change \( X, Y \)
  - \( C=1 \): Must not change \( X, Y \)
  - We’ll see how to ensure that later
- \textbf{Clock signal} – Pulsing signal used to enable latches
  - Because it ticks like a clock
  - Sequential circuit whose storage components all use clock signals: \textit{synchronous} circuit
  - Most common type
  - Asynchronous circuits – important topic, but left for advanced course
Sequential Logic Design
Clock Signal Terminology

- Clock period
  - Time interval between pulses
    - Above signal: period = 20 ns
- Clock cycle
  - Oone such time interval
    - Above signal shows 3.5 clock cycles
- Clock frequency
  - \( 1/\text{period} \)
    - Above signal: frequency = \( 1 / 20 \text{ ns} = 50 \text{ MHz} \)
    - \( 1\text{ Hz} = 1/\text{s} \)

<table>
<thead>
<tr>
<th>Freq</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 GHz</td>
<td>0.01 ns</td>
</tr>
<tr>
<td>10 GHz</td>
<td>0.1 ns</td>
</tr>
<tr>
<td>1 GHz</td>
<td>1 ns</td>
</tr>
<tr>
<td>100 MHz</td>
<td>10 ns</td>
</tr>
<tr>
<td>10 MHz</td>
<td>100 ns</td>
</tr>
</tbody>
</table>

Sequential Logic Design
Level-Sensitive D Latch

- SR latch requires careful design to ensure SR=11 never occurs
- D latch relieves designer of that burden
  - Inserted inverter ensures R always opposite of S

- D latch still has problem (as does SR latch)
  - When C=1, through how many latches will a signal travel?
  - Depends on for how long C=1
    - \( \text{Clk}_A \) -- signal may travel through multiple latches
    - \( \text{Clk}_B \) -- signal may travel through fewer latches
  - Hard to pick C that is just the right length
    - Can we design bit storage that only stores a value on the rising edge of a clock signal?

- Flip-flop. Bit storage that stores on clock edge, not level
  - One design -- master-servant
    - Two latches, output of first goes to input of second, master latch has inverted clock signal
    - So master loaded when C=0, then servant when C=1
    - When C changes from 0 to 1, master disabled, servant loaded with value that was at D just before C changed -- i.e., value at D during rising edge of C

Sequential Logic Design
D Flip-Flop

Note: Hundreds of different flip-flop designs exist.
Sequential Logic Design
D Flip-Flop

- Symbol for rising-edge triggered D flip-flop
- Symbol for falling-edge triggered D flip-flop
- Internal design: Just invert servant clock rather than master
- The triangle means clock input, edge triggered

Sequential Logic Design
D Latch vs. D Flip-Flop

- Latch is level-sensitive: Stores D when C=1
- Flip-flop is edge triggered: Stores D when C changes from 0 to 1
- Saying “level-sensitive latch,” or “edge-triggered flip-flop,” is redundant
- Two types of flip-flops -- rising or falling edge triggered.
- Comparing behavior of latch and flip-flop:

Sequential Logic Design
Bit Storage Summary

- D flip-flop
- D latch
- Master
- Servant
- Feature: Only loads D value present at rising clock edge, so values can't propagate to other flip-flops during same clock cycle.
- Tradeoff: uses more gates internally than D latch, and requires more external gates than SR – but gate count is less of an issue today.

- Feature: Only loads D value present at rising clock edge, so values can't propagate to other flip-flops during same clock cycle.
- Tradeoff: uses more gates internally than D latch, and requires more external gates than SR – but gate count is less of an issue today.
Typically, we store multi-bit items:
- e.g., storing a 4-bit binary number

**Register**: multiple flip-flops sharing clock signal
- From this point, we’ll use registers for bit storage:
  - No need to think of latches or flip-flops
  - But now you know what’s inside a register