Elements of sequential logic

- One can view combinational logic as the means to produce outputs in a feed-forward fashion from a given set of inputs. What could we do by introducing feedback in a digital circuit?

- It turns out, a LOT of things.
  - Including the ability to “store” previous values, i.e. making future outputs dependent on stored values, and hence on previous inputs.
  - Basic sequential logic components:
    - Latches
    - Edge-triggered flip-flops
The nature of memory

- Two strategies to implement “memory”
  - **Retention**
    - Exploits capacitance, it is passive, and eventually degrades (leakage currents).
  - **Reinforcement**
    - Exploits feedback, it is active, lasts indefinitely (as long as current is supplied).
  - **Combinations of the two**
    - E.g., capacitance used but “refreshed” occasionally.

Inverter pair as a memory cell

- Two inverters form a static memory cell
  - Will hold value as long as it has power applied

- How to get a new value into the memory cell?
  - Selectively break feedback path
  - Load new value into cell
NOR gate as a controlled inverter

\( (\text{Control}+A)' \)

- If Control=0
  - Output is complement of A.
- If Control=1
  - Output is 0.

Controlling the double inversion

- Cross-coupled NOR gates
  - similar to inverter pair, with capability to force output to 0 \((R=1)\) or 1 \((S=1)\)

- Cross-coupled NAND gates
  - similar to inverter pair, with capability to force output to 0 \((R=0)\) or 1 \((S=0)\)
Timing behavior

R-S latch states

Truth table of R-S latch behavior

<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>hold</td>
<td>0 1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>unstable</td>
<td>0 0</td>
</tr>
</tbody>
</table>

\[ Q' = 1 \]
Theoretical R-S latch behavior

- State diagram
  - states: possible values
  - transitions: changes based on inputs

  ![State diagram](image)

  Note: possible oscillation between states 00 and 11

Observed R-S latch behavior

- Very difficult to observe R-S latch in the 1-1 state
  - one of R or S usually changes first, even if by a small time difference
- Ambiguously returns to state 0-1 or 1-0
  - a so-called “race condition” or non-deterministic transition

  ![Observed state diagram](image)
**R-S latch analysis**

- **Breaking the feedback path**

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>Q(t)</th>
<th>Q(t+Δ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>X</td>
</tr>
</tbody>
</table>

- **Characteristic equation**

\[ Q(t+Δ) = S + R' Q(t) \]

- **Gated R-S latch**

- **Control when R and S inputs “matter”**
- Otherwise, the slightest glitch on R or S while enable is low could cause change in value stored

**Set**

**Reset**

**Enable’**

**Q’**

**Q**

**S’**

**R’**
Clocks

- Used to keep time
  - Wait long enough for inputs (R' and S') to settle
  - Then allow to have effect on value stored

- Clocks are regular periodic signals
  - Period (time between ticks)
  - Duty-cycle (time clock is high between ticks - expressed as % of period)

\[ \text{Clocks (cont'd)} \]

- Controlling an R-S latch with a clock
  - R and S should not change while clock is active (R and S pass through)
  - Only have half of clock period for signal changes to propagate
  - Signals must be stable for the other half of clock period
Cascading latches

- Connect output of one latch to input of another
- How to stop changes from racing through chain?
  - need to be able to control flow of data from one latch to the next
  - move one latch per clock period
  - have to worry about logic between latches (arrows) that is too fast

Master-slave structure

- Break flow by alternating clocks (analogous to an air-lock)
  - use positive clock to latch inputs into one R-S latch
  - use negative clock to change outputs with another R-S latch
- View pair as one basic unit
  - master-slave flip-flop
  - twice as much logic
  - output changes a few gate delays after the falling edge of clock but does not affect any cascaded flip-flops
The 1s catching problem

- In first R-S stage of master-slave FF
  - 0-1-0 glitch on R or S while clock is high is "caught" by master stage
  - leads to constraints on logic to be hazard-free

D flip-flop

- Make S and R complements of each other
  - eliminates 1s catching problem
  - cannot hold the previous value (note: we lost the SR=00).
    hence, it must have new value ready every clock period
  - value of D just before clock goes low is what is stored in flip-flop
  - can make R-S flip-flop by adding logic to make \( D = S + R' \ Q \)
Edge-triggered flip-flops

- More efficient solution (6 gates)
- Sensitive to input values “around” the edge of the clock signal (not “while” high).
- Must respect setup and hold time constraints to successfully capture input.

Negative edge-triggered D flip-flop (D-FF)

Characteristic equation: \( Q(t+1) = D \)

Edge-triggered flip-flops (cont’d)

- When clock goes high-to-low, data is latched.
- When clock is low, data is held and Dnew ≠ D.
Edge-triggered flip-flops (cont’d)

- Positive edge-triggered
  - inputs sampled on rising edge; outputs change after rising edge
- Negative edge-triggered flip-flops
  - inputs sampled on falling edge; outputs change after falling edge

Comparison of latches and flip-flops

behavior is the same unless input changes while the clock is high
References