Memory Elements

• Combinational logic cannot remember
  ➢ Output logic values are function of inputs only
  ➢ Feedback is needed to be able to remember a logic value

• Memory elements are needed in most digital logic circuits to hold (remember) logic values

• 2 basic types of memory elements
  ➢ Latches
    • *Level-sensitive* to inputs
  ➢ Flip-flops
    • *Edge-triggered* on active edge of clock
Reset-Set (RS) Latch (NOR)

- The simplest memory element
  - Aka set-reset (SR) latch
- Cross-coupled NOR gates
  - Level sensitive
  - Active high inputs
    - R (reset)
    - S (set)
    - Only one input can be active
      - To avoid undefined state
  - Outputs: Q and Q’
    - Q = current state of latch

<table>
<thead>
<tr>
<th>R</th>
<th>S</th>
<th>Q</th>
<th>Q’</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Q</td>
<td>Q’</td>
<td>Storage</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Set</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Reset</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0-?</td>
<td>0-?</td>
<td>Undefined</td>
</tr>
</tbody>
</table>
Reset-Set (RS) Latch (NAND)

- Dual of NOR RS latch
- Cross-coupled NAND gates
  - Level sensitive
  - Active low inputs
    - R (reset)
    - S (set)
    - Only one input can be active
      ✓ To avoid undefined state
  - Outputs: Q and Q’
    - Q = current state of latch

<table>
<thead>
<tr>
<th>R</th>
<th>S</th>
<th>Q</th>
<th>Q’</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1-?</td>
<td>1-?</td>
<td>Undefined</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Reset</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Set</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Q</td>
<td>Q’</td>
<td>Storage</td>
</tr>
</tbody>
</table>
Enabled Reset-Set (RS) Latch (NOR)

• Aka gated RS latch
  ➢ When enable E is inactive, RS latch is forced into storage state
    • R and S can do nothing

• AND gates plus NOR RS latch
  ➢ Level sensitive
  ➢ Active high inputs
    • E (enable)
    • R (reset)
    • S (set)
    • R and S cannot both be active when E is active
      ✓ To avoid undefined state
  ➢ Outputs: Q and Q’
    • Q = current state of latch

<table>
<thead>
<tr>
<th>E</th>
<th>R</th>
<th>S</th>
<th>Q</th>
<th>Q’</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>Q</td>
<td>Q’</td>
<td>Storage</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Q</td>
<td>Q’</td>
<td>Storage</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Set</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Reset</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0-?</td>
<td>0-?</td>
<td>Undefined</td>
</tr>
</tbody>
</table>
Enabled Reset-Set (RS) Latch (NAND)

- Aka gated RS latch
  - When enable E is inactive, RS latch is forced into storage state
    - R and S can do nothing

- OR gates plus NAND RS latch
  - Level sensitive
  - Active low inputs
    - E (enable)
    - R (reset)
    - S (set)
    - R and S cannot both be active when E is active
      ✓ To avoid undefined state
  - Outputs: Q and Q’
    - Q = current state of latch

<table>
<thead>
<tr>
<th>E</th>
<th>R</th>
<th>S</th>
<th>Q</th>
<th>Q’</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>Q</td>
<td>Q’</td>
<td>Storage</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1-?</td>
<td>1-?</td>
<td>Undefined</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Reset</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Set</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Q</td>
<td>Q’</td>
<td>Storage</td>
</tr>
</tbody>
</table>

C. E. Stroud  Latches & Flip-flops (10/12)
Enabled Data or Delay (D) Latch

• Aka transparent D latch
  ➢ Overcomes undefined state
    • R & S never active at same time
• Inverter plus enabled RS latch
  ➢ Level sensitive
    • Active high enable for NOR latch
    • Active low enable for NAND latch

### Logic Table

<table>
<thead>
<tr>
<th>E</th>
<th>D</th>
<th>Q</th>
<th>Q'</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>Q</td>
<td>Q'</td>
<td>Storage</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Transparent</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Transparent</td>
</tr>
</tbody>
</table>

### Logic Symbols

- **E**: Enable
- **D**: Data Input
- **Q**: Output
- **Q'**: Complement of Output

- **D active high**: OR gate
- **D active low**: NAND gate
- **E active low**: Inverter
- **E active high**: Inverter
D Flip-Flop

• Aka Master-Slave flip-flop
• Two transparent D latches
  ➢ Sensitive to opposite levels of Clock
    • One is always in storage and the other transparent
• Edge-triggered
  ➢ Data moves through on Clock transition
  ➢ Active-low latch followed by active-high
    • Rising edge-triggered
      ✓ aka leading edge-triggered
  ➢ Active-high latch followed by active-low
    • Falling edge-triggered
      ✓ aka trailing edge-triggered
D Flip-Flop

- Gate-level implementation
  - No need for inverter in slave latch since master has $Q$ & $Q'$

Rising edge-triggered D flip-flop

Falling edge-triggered D flip-flop
Timing Considerations

• Set-up time ($t_{su}$) = minimum time data (D) must be valid at input to flip-flop prior to the active edge of the clock

• Hold time ($t_h$) = minimum time data (D) must remain valid at input to flip-flop after the active edge of the clock

• Clock-to-output delay ($t_{co}$) = maximum time before output data (Q) is valid after the active edge of the clock
Timing Considerations

- Set-up & hold time violations in a real circuit result in metastability
  - Flip-flop goes to intermediate logic levels \((Q = Q')\)
  - Eventually resolves to an unknown state
- Set-up & hold time violations in a vector set for simulation referred to as clock-data-races
  - Leads to invalid simulation results & manufacturing testing problems
What is the Clock?

- Typically a periodic signal (a sequence of pulses) used to:
  - sample data, and
  - store the sampled data in memory elements

- **Clock frequency** = $1/\text{period}$
  - $f_{clk} = 1/T_p$
  - $T_p \geq t_{co} + P_{del} + t_{su}$
  - $P_{del} \leq T_p - t_{co} - t_{su}$
Serial Shift Register Example

- A series of D flip-flops whose outputs are connected to the input of the next flip-flop
  - \textit{serial-in, serial-out} = data in on Din, data out on Qc
  - \textit{serial-in, parallel-out} = data in on Din, data out on Qa, Qb, and Qc
Another Shift Register Example

- A series of multiplexers and D flip-flops whose outputs are connected to the input of the next flip-flop

  - *parallel-in, parallel-out* = data in on Da, Db, and Dc; data out on Qa, Qb, and Qc (Shift/Load = 0)
  
  - *parallel-in, serial-out* = data in on Da, Db, and Dc; data out on Qc (Shift/Load = 0, then Shift/Load = 1)
  
  - *Serial-in, serial-out* = data in on Din, data out on Qc (Shift/Load = 1)
  
  - *Serial-in, parallel-out* = data in on Din, data out on Qa, Qb, and Qc (Shift/Load = 1)
PSIM Architecture

**Sequential Logic:**
- Program Memory (MEM)
- Program Counter (PC)
- Address Register (AR)
- Data Register (DR)
- Input Register (IN)
- Output Register (OR)
- Accumulator (AC)
- ALU Carry Register (C)
- Instruction Register (IR)
- Timing Counter (TC)

**Combinational Logic:**
- Control Logic
- Arithmetic/Logic Unit (ALU)
- Multiplexers 1&2 (MUX)

C. E. Stroud
Latches & Flip-flops (10/12)
Another Register Example

• A series of multiplexers and D flip-flops whose outputs are connected to the input of the MUX

➢ Register with active high Load
  • Load = 1 & rising edge of clock: parallel-in, parallel-out = data in on Da, Db, and Dc; data out on Qa, Qb, and Qc
  • Otherwise: Holds data; data out remains on Qa, Qb, and Qc

➢ Basic register design used in PSIM for:
  • AR, DR, OR, IN (all 8-bits) and IR (4-bits)
Accumulator Register Example

• Accumulator in PSIM
  ➢ Functions controlled by combinational logic design
    • Including holding data when no operations are specified
      ✓ Via feedback of AC_i
  ➢ Only need a flip-flop at output of MUX
    • AC register (8-bits)
    • C register (1-bit)
      ✓ Similar to AC_i design shown here
Random Access Memory (RAM)

- Assuming MEM from PSIM
  - 8-bit address => 256 words
    - MADD
  - 8-bit words
    - Input data = 8-bits
      ✓ From DR
    - Output data = 8-bits
      ✓ From MEM
  - Active high write enable
    - WR-MEM
      ✓ When WR-MEM = 1, data from DR is written into address location specified by MADD
RAM continued

• RAM consists of:
  ➢ Address decoder with enable
    • Produces active high enables to registers
  ➢ Registers with parallel load
    • Stores data associated with specified address
  ➢ Read MUX
    • Reads specified address
RAM continued

- **Word Registers with parallel load**
  - 8 D-latches with active high enable

![Diagram of D-latches with active high enable](image)
RAM continued

- **Read MUX**
  - 8 256-to-1 MUXs
  - Functional equivalent

- **Address decoder**
  - 256 9-input AND gates
  - 8 inverters

![Diagram of RAM components](image-url)
What is Sequential Logic?

• A collection of logic gates and flip-flops
  ➢ The logic values stored in the flip-flops establish the *current state* of the sequential logic circuit
  ➢ The logic values at the inputs in conjunction with the current state determines the next state of the sequential logic circuit after the active edge of the clock
Flip-Flop Information for Sequential Logic Design

• Types of flip-flops
  ➢ D (data)
  ➢ T (toggle)
  ➢ SR (set-reset)
    • Also known as RS (reset-set)
  ➢ JK (Jack Kilby)

We will consider only edge-triggered flip-flops

• Each type has associated:
  ➢ Characteristic equation
  ➢ Characteristic table
    • sometimes called state table
  ➢ State diagram
  ➢ Excitation table

All provide same basic information but in slightly different forms
State Diagrams & State Tables

• Describe complete operation of sequential logic circuit
  ➢ Vertices (nodes) represent states
  ➢ Edges represent state transitions on active edge of clock based on primary input logic values

• State diagram & state tables provide exact same information
  ➢ Diagram is graphical representation of same info as in state table

• Given current state and primary input values we can determine the next state after active edge of clock
D Flip-Flop Specification

state diagram

characteristic table

<table>
<thead>
<tr>
<th>D</th>
<th>Q⁺</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

characteristic equation

Q⁺ = D

excitation table

<table>
<thead>
<tr>
<th>Q</th>
<th>Q⁺</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

logic diagram
T Flip-Flop Specification

state diagram

characteristic equation

\[ Q^+ = TQ' + T'Q \]
\[ = T \oplus Q \]

characteristic table

<table>
<thead>
<tr>
<th>T</th>
<th>Q+</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Q</td>
<td>Storage</td>
</tr>
<tr>
<td>1</td>
<td>Q'</td>
<td>Toggle</td>
</tr>
</tbody>
</table>

excitation table

<table>
<thead>
<tr>
<th>Q</th>
<th>Q+</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
RS Flip-Flop Specification

**state diagram**

![State Diagram](image)

**characteristic equation**

\[ Q^+ = S + R'Q \]

**input ordering = SR**

**characteristic table**

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>Q+</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Q</td>
<td>Storage</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Reset</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Set</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>?</td>
<td>Indeterminant</td>
</tr>
</tbody>
</table>

**logic diagram**

![Logic Diagram](image)

**excitation table**

<table>
<thead>
<tr>
<th>Q</th>
<th>Q+</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0X</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>01</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>X0</td>
</tr>
</tbody>
</table>
JK Flip-Flop Specification

characteristic equation

\[ Q^+ = JQ' + K'Q \]

input ordering = JK

state diagram

characteristic table

<table>
<thead>
<tr>
<th>J</th>
<th>K</th>
<th>Q+</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Q</td>
<td>Storage</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Reset</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Set</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Q'</td>
<td>Toggle</td>
</tr>
</tbody>
</table>

excitation table

<table>
<thead>
<tr>
<th>Q</th>
<th>Q+</th>
<th>JK</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0X</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1X</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>X1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>X0</td>
</tr>
</tbody>
</table>

logic diagram

C. E. Stroud
Latches & Flip-flops (10/12)
Flip-Flop Initialization

- Preset (aka set) => \( Q^+ = 1 \)
- Clear (aka reset) => \( Q^+ = 0 \)
- Some flip-flops have:
  - Both preset and clear (set and reset)
  - A preset or a clear
  - Neither (JK & SR flops have set/reset functions)
- Preset and/or clear can be
  - Active high or active low
  - Synchronous => with respect to active edge of clock
  - Asynchronous => independent of clock edges
- Initialization important for:
  - logic simulation to remove undefined logic values (2, 3, U, etc.)
  - system operation to put system in a known state
Synchronous vs. Asynchronous

- Synchronous => states of memory elements change only with respect to active edge of clock
- Asynchronous => states of memory elements can change without an active edge of clock
  ➢ Asynchronous designs often have timing problems

Example: assume sync preset and async clear