Sequential Logic

- Sequential Circuits
  - Simple circuits with feedback
  - Latches
  - Edge-triggered flip-flops
- Timing Methodologies
  - Cascading flip-flops for proper operation
  - Clock skew
- Asynchronous Inputs
  - metastability and synchronization
- Basic Registers
  - Shift registers

Circuits with Feedback

- How to control feedback?
  - What stops values from cycling around endlessly

Simplest Circuits with Feedback

- 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

Memory with Cross-coupled Gates

- Cross-coupled NOR gates
  - Similar to inverter pair, with capability to force output to 0 (reset=1 or 1 (set=0)
- Cross-coupled NAND gates
  - Similar to inverter pair, with capability to force output to 0 (reset=0) or 1 (set=0)

Timing Behavior
**State Behavior of R-S latch**

- 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>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Theoretical R-S Latch Behavior**

- State Diagram
  - States: possible values
  - Transitions: changes based on inputs
  - 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
- Ambiguously returns to state 0-1 or 1-0
- A so-called “race condition”
- Or non-deterministic transition

**R-S Latch Analysis**

- Break feedback path

**Gated R-S Latch**

- Control when R and S inputs matter
- Otherwise, the slightest glitch in R or S while enable is low can cause change in value stored

**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)
Clocks (cont’d)

- Controlling an R-S latch with a clock
  - Can’t let R and S change while clock is active (allowing R and S to pass)
  - Only have half of clock period for signal changes to propagate
  - Signals must be stable for the other half of clock period

![Clock Diagram](Image)

Cascading Latches

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

![Cascading Latches Diagram](Image)

Master-Slave Structure

- Break flow by alternating clocks (like an on-off)
  - 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 the 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

![Master-Slave Structure Diagram](Image)

The Is Catching Problem

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

![The Is Catching Problem Diagram](Image)

D Flip-Flop

- Make S and R complements of each other
  - Eliminates Is catching problem
  - Can’t just hold previous value (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

![D Flip-Flop Diagram](Image)

Edge-Triggered Flip-Flops

- More efficient solution: only 6 gates
  - sensitive to inputs only near edge of clock signal (set while high)
  - master stage
  - slave stage

![Edge-Triggered Flip-Flops Diagram](Image)
**Edge-Triggered Flip-Flops (cont’d)**

- Step-by-step analysis

![Diagram of edge-triggered flip-flops](image)

- 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

**Timing Methodologies**

- Rules for interconnecting components and clocks
  - Guarantee proper operation of system when strictly followed
- Approach depends on building blocks used for memory elements
  - Flip-flops in systems with edge-triggered flip-flops
  - Found in programs like logic design
- Many custom integrated circuits focus on level-sensitive latches
- Basic rules for correct timing:
  1. Correct inputs, with respect to time, are provided to the flip-flops
  2. No flip-flop changes state more than once per clocking event

**Comparison of Latches and Flip-Flops**

- **Transparent latch**
  - Propagation delay from input changes
- **Clearing edge-trigger flip-flop**
  - Propagation delay from input changes

**Timing Methodologies (cont’d)**

- Definition of terms
  - Clock event: causes state of memory element to change; can be rising or falling edge, or high or low level
- Setup time: minimum time before the clocking event by which the input must be stable (T_{su})
- Hold time: minimum time after the clocking event until which the input must remain stable (T_{th})

**Comparison of Latches and Flip-Flops (cont’d)**

<table>
<thead>
<tr>
<th>Type</th>
<th>When input is sampled</th>
<th>When output is valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transparent latch</td>
<td>a low</td>
<td>Propagation delay from input change</td>
</tr>
<tr>
<td>Level-sensitive</td>
<td>a low</td>
<td>Propagation delay from input change</td>
</tr>
<tr>
<td>edge-trigger flip-flop</td>
<td>a low</td>
<td>Propagation delay from falling edge of clock</td>
</tr>
<tr>
<td>Negative edge-trigger flip-flop</td>
<td>a low</td>
<td>Propagation delay from falling edge of clock</td>
</tr>
</tbody>
</table>
Typical Timing Specifications

- Positive edge-triggered D flip-flop
- Setup and hold times
- Minimum clock width
- Propagation delays (low to high, high to low, max and typical)

Cascading Edge-triggered Flip-Flops

- Shift register
  - New value goes into first stage
  - While previous value of first stage goes into second stage
  - Consider setup/hold or propagation delays (prop must > hold)

Cascading Edge-triggered Flip-Flops (cont'd)

- Why this works
  - Propagation delays exceed hold times
  - Clock width constraint exceeds setup time
  - Clock width constraint is satisfied if the system is designed properly

Clock Skew

- The problem
  - Correct behavior assumes next state of all storage elements determined by all storage elements at the same time
  - This is difficult in high-performance systems because timing for clock to arrive at flip-flops is comparable to delays through logic

Summary of Latches and Flip-Flops

- Development of D-FF
  - Level-sensitive used in custom integrated circuits
  - Edge-triggered used in programmable logic devices
  - Good choice for data storage register

- Historically, J-K FF was popular but now never used
  - Similar to R-S but with J=1 being used to toggle output (complimentary state)
  - Good in days of TTL/SS7 (more complex input function: d = Qd' + Qd)
  - Not a good choice for PALs/PALs as it requires 2 inputs
  - Can always be implemented using D-FF

- Preset and clear inputs are highly desirable on flip-flops
  - Used at startup or to reset system to default state

Metastability and Asynchronous inputs

- Clocked synchronous circuits
  - Inputs, state, and outputs sampled or changed in relation to a common reference signal (called the clock)
  - E.g., master/slave, edge-triggered

- Asynchronous circuits
  - Inputs, state, and outputs sampled or changed independently of a common reference signal (glitches/hazards a major concern)
  - E.g., R-S latch

- Asynchronous inputs to synchronous circuits
  - Inputs can change at any time without reset setup/hold times
  - Dangerous, synchronous inputs are greatly preferred
  - Cannot be avoided (e.g., reset signal, memory write, user input)
Synchronization Failure

- Occurs when FF input changes close to clock edge
- FF may enter a metastable state - neither a logic 0 nor 1
- May stay in this state an indefinite amount of time
- Is not likely in practice but has some probability

Dealing with Synchronization Failure

- Probability of failure can never be reduced to 0, but it can be reduced
  1. Use 2 fast-timing clock (a clock slower than the system clock)
  2. Use a higher clock (this makes for a very sharp 'peak' upon which to balance)

Handling Asynchronous Inputs

- Never allow asynchronous inputs to fan-out more than one flip-flop
- Synchronize as soon as possible and then treat as synchronous signal

Handling Asynchronous Inputs (cont'd)

What can go wrong?
- Input changes too close to clock edge (violating setup time constraint)

Flip-Flop Features

- Reset (set state to 0) - R
  1. Synchronous: D = R' . B . 0 (when next clock edge arrives)
  2. Asynchronous: doesn't wait for clock, quick but dangerous

- Preset (set state to 1) - P (or sometimes S)
  1. Synchronous: D = B . P . 0 (when next clock edge arrives)
  2. Asynchronous: doesn't wait for clock, quick but dangerous

- Both reset and preset
  1. D = R' . B . 0 + P (set-dominant)
  2. D = P . B . R' S (reset-dominant)

- Selective input capability (input enable/lock) - LD or EN
- Multiple or at input: D = B . (LD + EN) . B
- Lead may not override reset/set (usually reset/set have priority)
- Complementary outputs - Q and Q'

Registers

- Collections of flip-flops with similar controls and logic
- Stored values somewhat related (e.g., 1's complement)
- Share clock, reset, and set lines
- Similar logic at each stage

Examples
- Shift registers
- Counters
Shift Register
- Holds 4 values
- Serial or parallel inputs
- Serial or parallel outputs
- Permits shift left or right
- Shift in new values from left or right

Universal Shift Register
- Holds 4 values
- Serial or parallel inputs
- Serial or parallel outputs
- Permits shift left or right
- Shift in new values from left or right

Design of Universal Shift Register
- Consider one of the four flip-flops
- New value at next clock cycle:

Pattern Recognizer
- Combinational function of input samples
- In this case, recognizing the pattern 001 on the single input signal

Counter
- Sequences through a fixed set of patterns
- In this case, 1000, 0000, 0110, 0001
- If one of the patterns is its initial state (by loading or set/reset)

Möbius (or Johnson) counter
- In this case, 1000, 1010, 1110, 0111, 0011, 0001, 0000
**Binary Counter**

- Logic between registers (not just multiplexer)
- XOR decides when bit should be toggled
- Always for lower-order bit only when first bit is true for second bit, and so on

**Four-bit Binary Synchronous Up-Counter**

- Standard component with many applications
- Positive edge-triggered FFs w/sync load and clear inputs
- Parallel load data from D, C, B, A
- Enable inputs must be asserted to enable counting
- RCO ripple-carry out used for cascading counters
- High when counter is in highest state
- Implemented using an AND gate

---

**Offset Counters**

- Starting offset counters - use of synchronous load
  - e.g., 0000, 0001, 0010, 0011, ..., 1000, 1001, 1010, 1100, 1110

- Ending offset counter - comparator for ending value
  - e.g., 0000, 0000, 0000, ..., 1100, 1110, 1111

- Combinations of the above (start and stop value)

---

**Sequential Logic Summary**

- Fundamental building block of circuits with state
  - Latch and flip-flop
  - R-S latch, R-S master-slave, D master-slave, edge-triggered D FF

- Timing methodologies
  - Use of clocks
  - Cascaded FFs work because prop delays exceed hold times
  - Beware of clock skew

- Asynchronous inputs and their dangers
  - Synchronizer failure: what it is and how to minimize its impact

- Basic registers
  - Shift registers
  - Pattern detectors
  - Counters