Sequential Logic

- **Sequential Circuits**
  - Simple circuits with feedback
  - Latches
  - Edge-triggered flip-flops
- **Timing Methodologies**
  - Cascading flip-flops for proper operation
  - Clock skew
- **Basic Registers**
  - Shift registers
  - Counters

Circuits with Feedback

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

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=1)

Sequential Circuits

- **Circuits with Feedback**
  - Outputs = f(inputs, past inputs, past outputs)
  - Basis for building "memory" into logic circuits
  - Door combination lock is an example of a sequential circuit
  - State is memory
  - State is an "output" and an "input" to combinational logic
  - Combination storage elements are also memory

Simplest Circuits with Feedback

- **Two inverters form a static memory cell**
  - Will hold value as long as it has power applied
    - "1" -- "stored value"
    - "0" -- "remember"

- **How to get a new value into the memory cell?**
  - Selectively break feedback path
  - Load new value into cell

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

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

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

Theoretical R-S Latch Behavior

- State Diagram
- States: possible values
- Transitions: changes based on inputs
- possible oscillation between states 00 and 11

R-S Latch Analysis

- Break feedback path

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

**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

**Master-Slave Structure**

- Break flow by alternating clocks (like 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 "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
  - 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 \( D = S + R' \)

**Edge-Triggered Flip-Flops**

- More efficient solution: only 6 gates
  - sensitive to inputs only near edge of clock signal (not while high)
**Edge-Triggered Flip-Flops (cont’d)**

- **Step-by-step analysis**
  - When clock goes high-to-low, data is latched.
  - New data is held when clock is low.

**Edge-Triggered Flip-Flops (cont’d)**

- **D = 1, Clk High**
  - Act as inverters
  - Hold state

**Edge-Triggered Flip-Flops (cont’d)**

- **D = 0, Clk LOW**
  - Act as inverters

**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.

```verilog
module d_ff (q, q_bar, data, clk);
    input  data, clk;
    output q, q_bar;
    reg    q;

    assign q_bar = ~q;
    always @(negedge clk)
        begin
            q <= data;
        end
endmodule
```

**Negative Edge Trigger FF in Verilog**
Announcements

- Cancel Lab Section Tu 11-2 PM starting next week
- Young assigned to Th 5-8 PM lab
- Allen assigned to W 5-8 PM lab (3 TAs!)

Homework #2 Bug
- Problem 7(b) revised and posted to the web
- Starting Thursday, lecture meets in 159 Mulford

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
  - Focus on systems with edge-triggered flip-flops
  - Found in programmable logic devices
  - 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 (cont'd)

- Definition of terms
  - clock: periodic 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 (Ts)
  - hold time: minimum time after the clocking event until which the input must remain stable (Th)

<table>
<thead>
<tr>
<th>Type</th>
<th>When inputs are sampled</th>
<th>When output is valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>unclocked latch</td>
<td>always</td>
<td>propagation delay from input change</td>
</tr>
<tr>
<td>level-sensitive</td>
<td>clock high (Ts/Th around falling edge of clock)</td>
<td>propagation delay from input change or clock edge (whichever is later)</td>
</tr>
<tr>
<td>master-slave flip-flop</td>
<td>clock high (Ts/Th around falling edge of clock)</td>
<td>propagation delay from falling edge of clock</td>
</tr>
<tr>
<td>negative edge-triggered flip-flop</td>
<td>clock hi-to-lo transition (Ts/Th around falling edge of clock)</td>
<td>propagation delay from falling edge of clock</td>
</tr>
</tbody>
</table>

Comparison of Latches and Flip-Flops

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

Typical Timing Specifications
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/propagation delays (prop must be > hold)

```
IN  D  Q0  D  Q1  OUT
CLK
```

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

- **Why this works**
  - Propagation delays exceed hold times
  - Clock width constraint exceeds setup time
  - This guarantees following stage will latch current value before it changes to new value

```
IN  Q0  Q1  CLK
100
```

**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/propagation delays (prop must be > hold)

```
IN  D  Q0  D  Q1  OUT
CLK
```

**Clock Skew**

- **The problem**
  - Correct behavior assumes next state of all storage elements determined by all storage elements at the same time
  - Difficult in high-performance systems because time for clock to arrive at flip-flop is comparable to delays through logic (and will soon become greater than logic delay)
  - Effect of skew on cascaded flip-flops:

```
CLK1 is a delayed version of CLK0
```

**Summary of Latches and Flip-Flops**

- **Development of D-FF**
  - Level-sensitive used in custom integrated circuits
  - More complex input function: D = Q' + Q'
  - Good choice for data storage register
  - Historically J-K FF was popular but now never used
  - Similar to R-S but with 1-1 being used to toggle output (complement state)
  - Good in days of TTL/SSI (more complex input function: D = JQ + KQ)
  - Not a good choice for PLAs as it requires two inputs
  - Can always be implemented using D-FF

- **Preset and clear inputs are highly desirable on flip-flops**
  - Used at start-up or to reset system to a known state

**Flip-Flop Features**

- **Reset (set state to Q): R**
  - Synchronous: D_{Reset} = R \cdot D_{Input} (when next clock edge arrives)
  - Asynchronous: doesn’t wait for clock, quick but dangerous

- **Preset or set (set state to Q): S**
  - Synchronous: D_{Reset} = D_{Input} + S (when next clock edge arrives)
  - Asynchronous: doesn’t wait for clock, quick but dangerous

- **Both reset and preset**
  - D_{Reset} = R \cdot D_{Input} + S (set-dominant)
  - D_{Reset} = R' \cdot D_{Input} + S (reset-dominant)

- **Selective input capability (input enable/load): LD or EN**
  - Multiplexer at input: D_{Reset} = LD \cdot D_{Input} + EN \cdot D_{Input}
  - Load may/may not override reset/set (usually R/S have priority)

- **Complementary outputs: Q and Q’**
Registets
- Collections of flip-flops with similar controls and logic
- Stored values somehow related (e.g., form binary value)
- Share clock, reset, and set lines
- Similar logic at each stage

Examples
- Shift registers
- Counters

Shift Register
- Holds samples of input
- Store last 4 input values in sequence
- 4-bit shift register

Shift Register Verilog
```verilog
module shift_reg (out4, out3, out2, out1, in, clk);
    output out4, out3, out2, out1;
    input in, clk;
    reg out4, out3, out2, out1;

    always @(posedge clk)
    begin
        out4 <= out3;
        out3 <= out2;
        out2 <= out1;
        out1 <= in;
    end
endmodule
```

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

Universal Shift Register Verilog
```verilog
module shift_reg (out, in, clk);
    output [4:1] out;
    input in, clk;
    reg [4:1] out;

    always @(posedge clk)
    begin
        out <= out[3:1], in;
    end
endmodule
```
Universal Shift Register Verilog

```
module univ_shift (out, lo, ro, in, li, ri, s, clr, clk);
output [3:0] out;
output lo, reg;
input [3:0] in;
input [3:0] s;
input li, ri, clr, clk;
reg [3:0] out;
assign lo = out[3];
assign ro = out[0];
always @(posedge clk or clr)
begin
  if (clr) out <= 0;
  else
    case (s)
      3: out <= in;
      2: out <= {out[2:0], ri};
      1: out <= {li, out[3:1]};
      0: out <= out;
    endcase
  end
endmodule
```

Shift Register Application

- **Parallel-to-serial conversion for serial transmission**

![Parallel-to-serial conversion circuit](image)

Pattern Recognizer

- **Combinational function of input samples**
  - In this case, recognizing the pattern 1001 on the single input signal

![Pattern Recognizer circuit](image)

Counters

- **Sequences through a fixed set of patterns**
  - In this case, 1000, 0100, 0010, 0001
  - If one of the patterns is its initial state (by loading or set/reset)

![Counter circuit](image)

- **Mobius (or Johnson) counter**
  - In this case, 1000, 1100, 1110, 1111, 0111, 0011, 0001, 0000

Binary Counter

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

![Binary Counter circuit](image)

Binary Counter Verilog

```
module shift_reg (out4, out3, out2, out1, clk);
  output out4, out3, out2, out1;
  input in, clk;
  reg out4, out3, out2, out1;
  always @(posedge clk)
  begin
    out4 <= (out1 & out2 & out3) ^ out4;
    out3 <= (out1 & out2) ^ out3;
    out2 <= out1 ^ out2;
    out1 <= out1 ^ 1'b1;
  end
endmodule
```
Binary Counter Verilog

module shift_reg (out4, out3, out2, out1, clk);
output [4:1] out;
input in, clk;
reg [4:1] out;
always @(posedge clk)
    out <= out + 1;
endmodule

Offset Counters

- Starting offset counters - use of synchronous load
  e.g., 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1111, 0110, ...
- Ending offset counter - comparator for ending value
  e.g., 0000, 0001, 0010, ..., 1100, 1101, 0000
- 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
- Basic registers
  - Shift registers
  - Pattern detectors
  - Counters

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 its highest state IIII
  - implemented using an AND gate

- 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
- Basic registers
  - Shift registers
  - Pattern detectors
  - Counters