Latches and Flip-Flops

Type	Inputs	EN/CLK	Q(next)	Comment
SR	S,R	–	00:Q, 01:0, 10:1, 11:?	Set/Reset
D	D	EN	EN=1:Q=D, EN=0:Q	Data/Delay
JK	J,K	–	00:Q, 01:0, 10:1, 11:$\overline{Q}$	Toggle
T	T	EN	EN=1:Q$\oplus$T, EN=0:Q	Toggle

```
// SR Latch (active high, asynchronous)
always_comb begin
  case ({S, R})
    2'b00: Q_next = Q;
    2'b01: Q_next = 0;
    2'b10: Q_next = 1;
    2'b11: Q_next = 1'bx; // Invalid
  endcase
end

// D Latch (level-sensitive)
always_comb
  if (EN) Q = D;

// JK Latch (level-sensitive)
always_comb begin
  case ({J, K})
    2'b00: Q_next = Q;
    2'b01: Q_next = 0;
    2'b10: Q_next = 1;
    2'b11: Q_next = ~Q;
  endcase
end

// T Latch (level-sensitive)
always_comb
  if (EN) Q = Q ^ T;
```

D Flip-Flop w/ Async Reset:

```
always_ff @(posedge clk, posedge reset)
  if (reset) Q <= 0;
  else Q <= D;
```

Timing Constraints

- **Setup time** t_{setup}: Data stable before clock edge.
- **Hold time** t_{hold}: Data stable after clock edge.
- **Propagation delay** t_{pcq}: Clock to stable output.
- **Contamination delay** t_{ccq}: Clock to output may change.
- **Clock skew** t_{skew}: Difference in clock arrival times.

$$T_{clk} \geq t_{pcq} + t_{pd} + t_{setup} + t_{skew} \quad \text{(Setup)}$$
$$t_{hold} < t_{ccq} + t_{cd} - t_{skew} \quad \text{(Hold)}$$
$$f_{clk,\,max} = \frac{1}{t_{pcq} + t_{pd} + t_{setup} + t_{skew}}$$

Finite State Machines

Mealy: Output = f(state, input). **Moore:** Output = f(state).

Signed Numbers

2's Complement:

$$-b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i$$

Manual: Invert all bits, then add 1.
Overflow: $C_{in} \neq C_{out}$ for addition.
Sign Extension: Replicate MSB.

Adder Circuits

Half Adder: $S = A \oplus B$, $C_{out} = A \cdot B$
Full Adder: $S = A \oplus B \oplus C_{in}$, $C_{out} = AB + C_{in}(A \oplus B)$

```
// Half Adder
assign S = A ^ B;
assign C_out = A & B;

// Full Adder
assign S = A ^ B ^ C_in;
assign C_out = (A & B) | (C_in & (A ^ B));
```

Carry Look-Ahead:
$G_i = A_i B_i$, $P_i = A_i + B_i$
$C_{i+1} = G_i + P_i C_i$

Adder Comparison:

- **Ripple Carry Adder:** Simple, area-efficient. Each bit must wait for carry from previous stage. Delay $O(n)$.
- **Carry-Lookahead Adder:** More complex, uses generate/propagate logic to compute carries in parallel. Delay $O(\log n)$.

BCD Adder

Add 6 if sum > 9. Use two 4-bit adders:

1. Add inputs (binary).
2. If sum > 9, add 6 (0110).

```
logic [3:0] A, B, SUM, SUM_CORR;
logic C_OUT, C_CORR;

assign {C_OUT, SUM} = A + B;
assign {C_CORR, SUM_CORR} = (SUM > 9) ? (SUM + 4'd6) : SUM;
```

Mod-N Counter:

```
always_comb
  count_next = (count == N-1) ? 0 : count + 1;
```

Multipliers

Array Multiplier: AND gates for partial products, adders for summation.
Booth's Algorithm: Encodes multiplier to reduce operations.

Clocking Issues

Metastability: Caused by setup/hold violations.
Synchronizer: Two flip-flops in series.
MTBF $\propto \frac{e^{t/\tau}}{fT_0}$