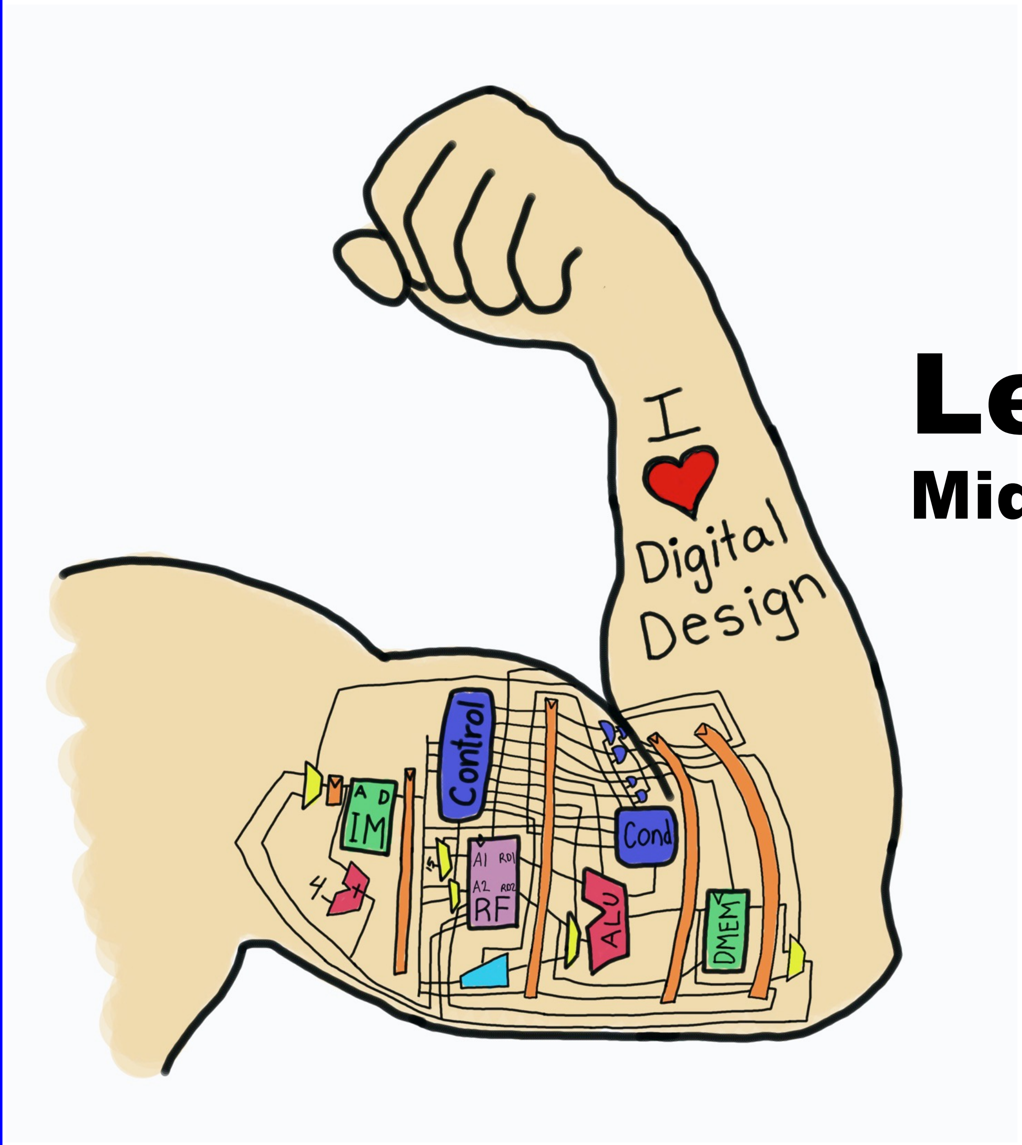
E85 Digital Design & Computer Engineering



Lecture 01011:

Midterm Review

HARVEY MUDD COLLEGE

Midterm Review

- Logic Levels
- Number Systems
- CMOS Transistors
- Power Consumption
- Combinational Logic Design
- Finite State Machines
- Timing
- Verilog
- Arithmetic Circuits

Logic Levels

- Assign V_{IH}, V_{IL}, V_{OH}, V_{OL} to maximize noise margins $|V_{OH} – V_{IH}|$, $|V_{OL} – V_{IL}|$
- Normally at the unity gain points
- If the curve has many bends, pick the ones to maximize noise margins

Logic Levels: Example

- What is the logic function?

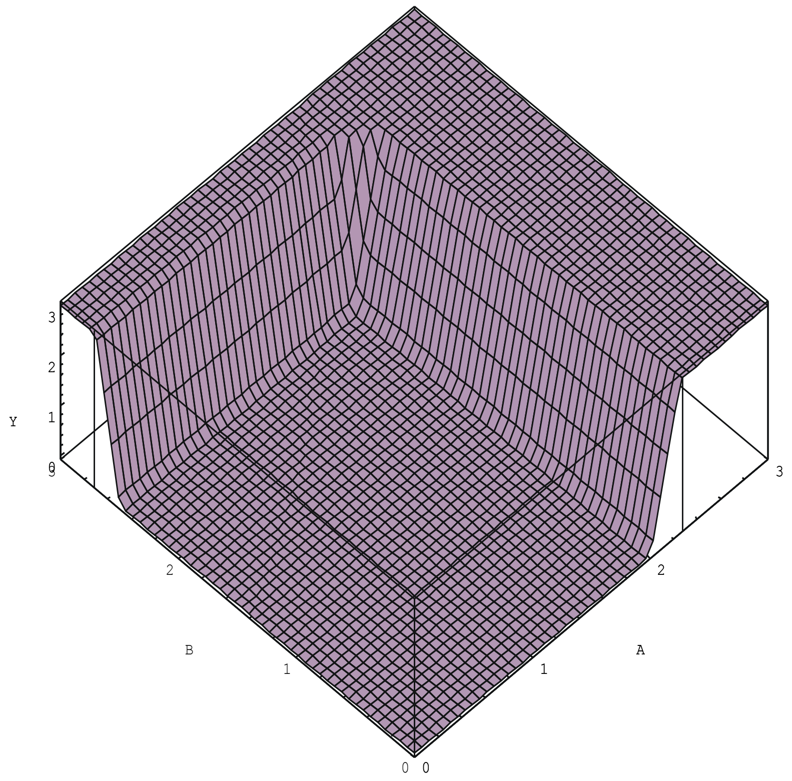
- What are the logic levels?

  V_{IL} = V_{IH} =

  V_{OL} = V_{OH} =

- What are the noise margins?

  NM_L = NM_H =



Logic Levels: Compatibility

- Consider two logic families

  A: $V_{IL} = 1$, $V_{IH} = 2.3$, $V_{OL} = 0.4$, $V_{OH} = 2.9$

  B: $V_{IL} = 2$, $V_{IH} = 3$, $V_{OL} = 1.1$, $V_{OH} = 3.2$

Can A drive itself?

Can B drive itself?

Can A drive B?

Can B drive A?

Number Systems: Signed and Unsigned

Find decimal value of 101_2 interpreted as:

Unsigned:

Sign/Magnitude:

Two's Complement:

Number Systems: Negative Numbers

Write 19 as a 6-bit binary number:

19 =

Write -19 as a 6-bit binary number

Two’s complement

Invert the bits and add 1

Sign/Magnitude

[illegible]

Number Systems: Bases

Write 37_{10} in other bases

Hexadecimal:

Binary:

CMOS Transistors

- Design nMOS pull-down network
  - Series for AND, parallel for OR
- pMOS pull-up network is complement
- CMOS gates are inherently inverting
- Add another stage to get non-inverting

CMOS Transistors: OR3

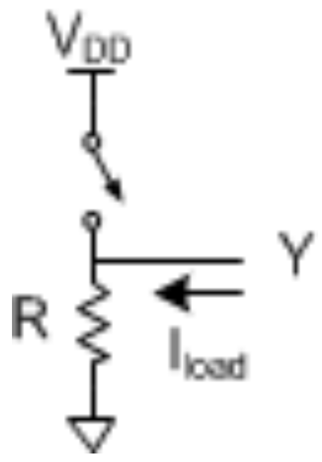
- Sketch a 3-input OR gate
  - Use NOR3 + inverter
  - NOR3: nMOS in parallel, hence pMOS in series

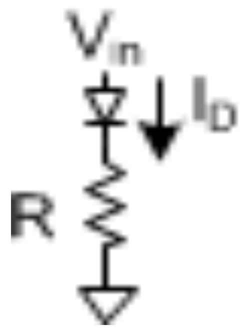
CMOS Transistors: AOI

- Sketch an AND-OR-INVERT gate Y = ~(AB+C)
  - nMOS network
    - A and B in series. This stack in parallel with C
  - pMOS network is complement
    - A and B in parallel. This stack in series with C

Switches and LEDs

- Switch:
  - Choose R big enough to limit power, small enough to keep a good logic level if I_{load} is leakage current.
  - $P = V_{DD}^2/R$
  - $V_{out} = I_{load} * R < V_{IL}$
- Light Emitting Diode
  - Choose R small enough to make the LED bright, large enough to not overstress I_{OH} of the gate driving V_{in}.
  - $I_D \sim (V_{in}-2) / R$
  - 5 mA is visible in room lighting and near max I_{OH} of many gates





Power Consumption

- $P = P_{dynamic} + P_{static} = \alpha C V_{DD}^2 f + I_{static} V_{DD}$
  - α = activity factor:
    - 1 for clocks rising and falling each cycle
    - 0.5 for data signal switching once per cycle
    - 0.5p for data signal switching with probability p
- Know your units
  - $K = 10^3$, $M = 10^6$, $G = 10^9$, $T=10^{12}$
  - $m = 10^{-3}$, $\mu = 10^{-6}$, $n = 10^{-9}$, $p = 10^{-12}$, $f=10^{-15}$

Power Consumption: Example

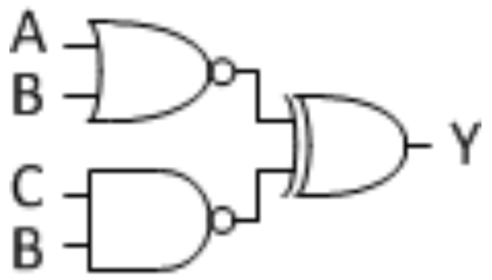
- V_{DD} = 0.707 V
- 1000 flip-flops clocked at 1 GHz. For each:
  - 100 nA leakage
  - 5 fF of clock capacitance
  - 20 fF capacitance on Q
  - 10% of inputs change on any given cycle
- Idle power = P_{static} =
- Running power = $P_{static} + P_{dynamic}$ = 70.7 µW +

Combinational Logic Design

- Output depends on current inputs
- Write truth table
- Circle 1's to find sum of products
- Simplify with Boolean algebra or inspection

Combinational Logic: Example

- Write a truth table & eqn for



A	B	C	Y
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

$Y =$

Combinational Logic: K-Map

- Write inputs in Gray code order 00 01 11 10
- Populate grid
- Circle 1's in boxes 1, 2, or 4 on a side
- Optionally circle Xs if it simplifies

		AB			
		00	01	11	10
CD	00	0	1	X	1
	01	0	1	1	0
	11	0	0	0	0
	10	X	0	0	1

$Y =$

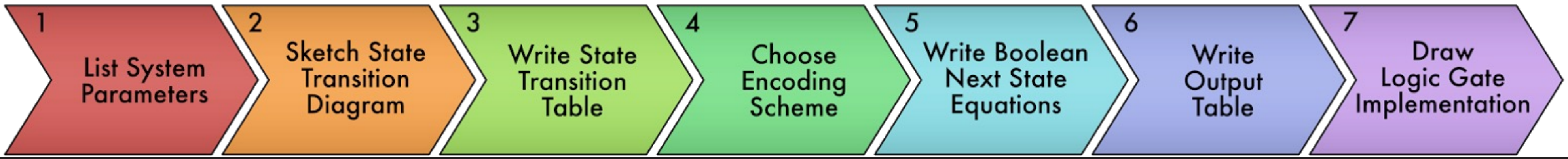
Sequential Circuits

- Sequential circuits: output depends on previous as well as current inputs
- Flip-flops
  - On the rising edge of CLK, Q gets D.
  - Enables
  - Reset: synchronous or asynchronous
- Synchronous sequential design: every element is combinational or a flip-flop, and all flops share the same clock. Easy to analyze.

Finite State Machines

- State transition diagram
- State encodings
- Next state and output tables
- Derive and simplify Boolean equations
- Sketch circuit

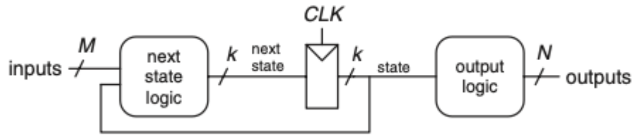
- Inverse problems: derive diagram from circuit

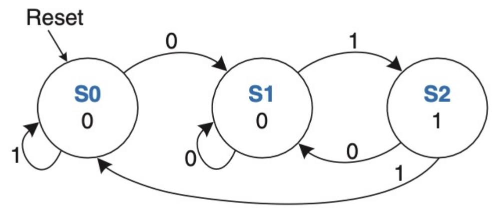


1 List System Parameters
2 Sketch State Transition Diagram
3 Write State Transition Table
4 Choose Encoding Scheme
5 Write Boolean Next State Equations
6 Write Output Table
7 Draw Logic Gate Implementation

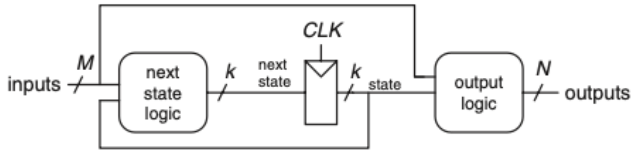
Moore Vs. Mealy Machines

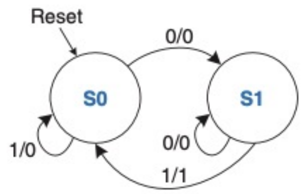
- Moore:
  - output depends only on state.
  - labeled in bubbles





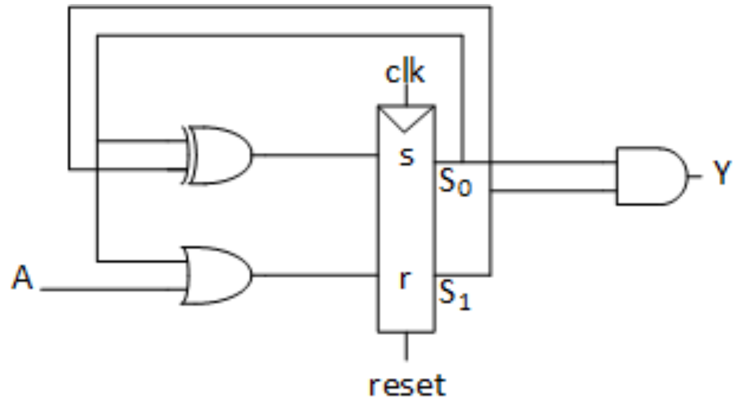
- Mealy:
  - output depends on state and inputs
  - labeled on arcs



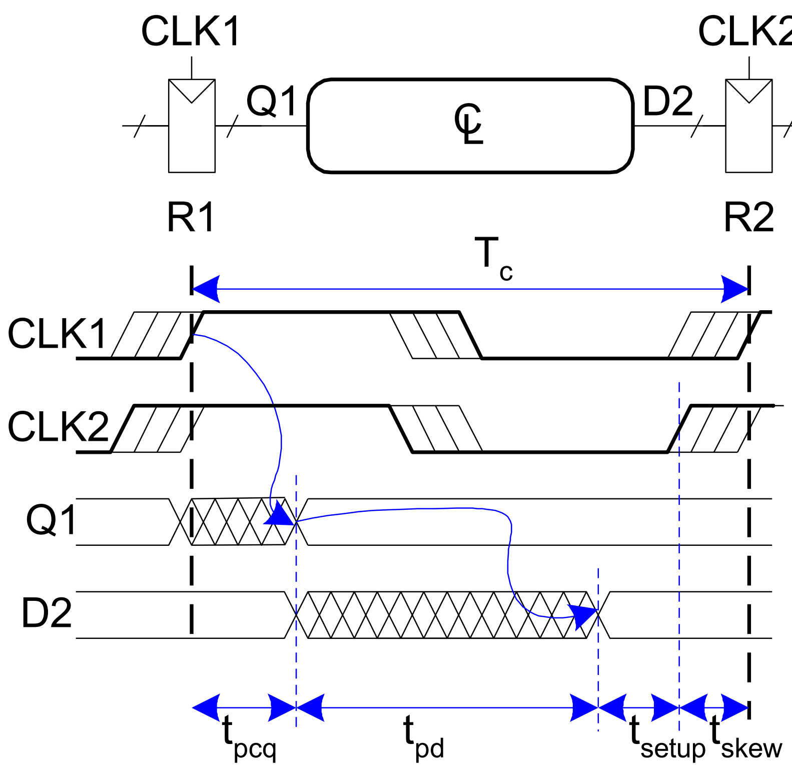


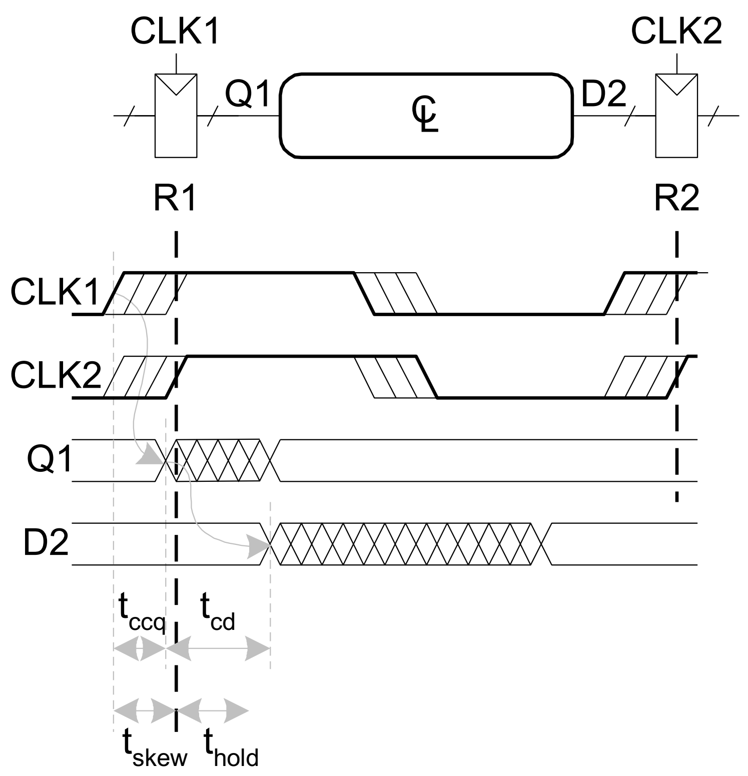
Inverse FSM Problems

- Derive FSM state transition diagram from circuit
- Trace FSM
  - Start in reset state
  - For each state being explored
    - Determine next state for each input pattern
    - Determine output



Timing





$T_c \geq$

Metastability

For each sample, probability of failure is:

$$\mathbf{P(failure)} = (T_0/T_c)\, e^{-(T_c - t_{setup})/\tau}$$

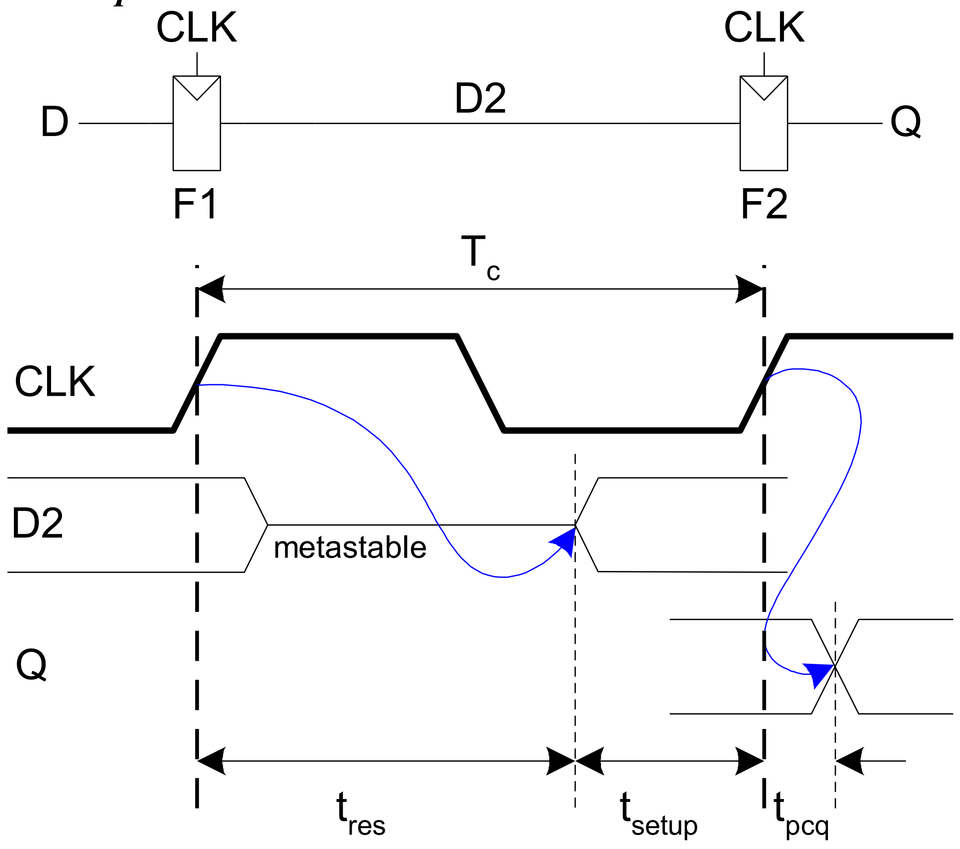
P(failure)/second =

$$(NT_0/T_c)\, e^{-(T_c - t_{setup})/\tau}$$

MTBF =

$$1/[P(\text{failure})/\text{second}] =$$

$$(T_c/NT_0)\, e^{(T_c - t_{setup})/\tau}$$



Verilog

- Think of the logic you want first
- Use Verilog as shorthand for logic
- Pick the appropriate idiom for each element

Verilog Idioms: Combinational Logic

Combinational Logic with Boolean Eqns.

```
assign y = (a & b) ^ (c | ~d);
```

Multiplexers

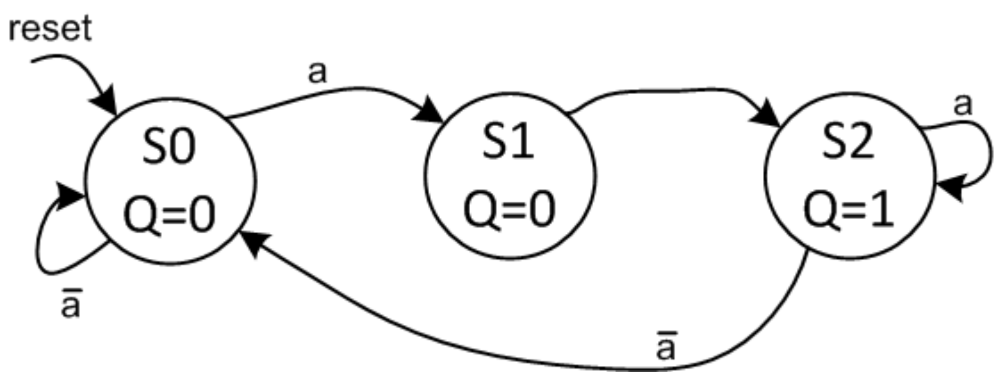
```
assign y = s ? d1 : d0;
```

Comb logic with truth tables

```
always_comb
     casez(in)
          3'b1xx: y <= 2'b11;
          3'b01x: y <= 2'b10;
          3'b001: y <= 2'b01;
          default: y <= 2'b00;
     endcase
```

Verilog Idioms: FSMs

```
module fsmWithInputs(input  logic clk,
                     input  logic reset,
                     input  logic a,
                     output logic q);

   typedef enum logic [1:0] {S0, S1, S2} statetype;
   statetype state, nextstate;

   // state register
   always_ff @(posedge clk, posedge reset)
      if (reset) state <= S0;
      else       state <= nextstate;

   // next state logic
   always_comb
      case (state)
         S0:      if (a) nextstate = S1;
                  else   nextstate = S0;
         S1:             nextstate = S2;
         S2:      if (a) nextstate = S2;
                  else   nextstate = S0;
         default:        nextstate = S0;
      endcase

   // output logic
   assign q = (state == S2);
endmodule
```

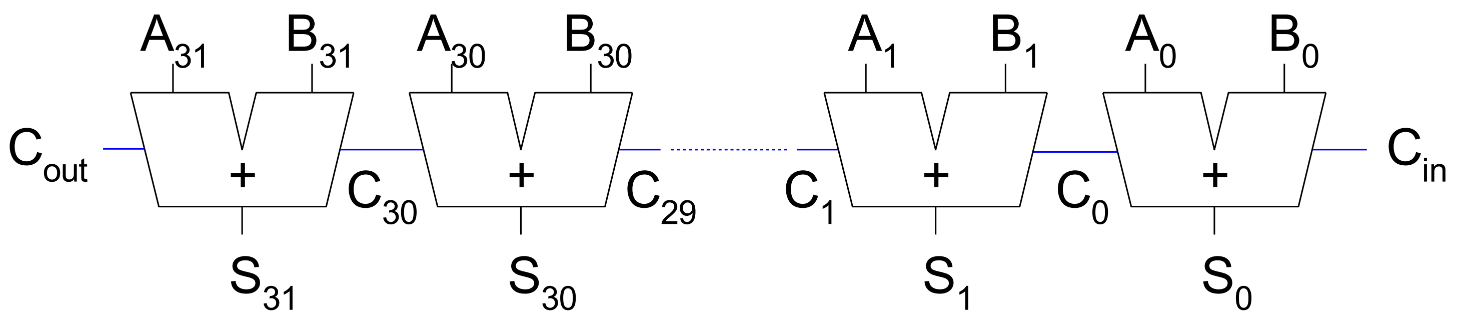


Verilog Idioms: Structural

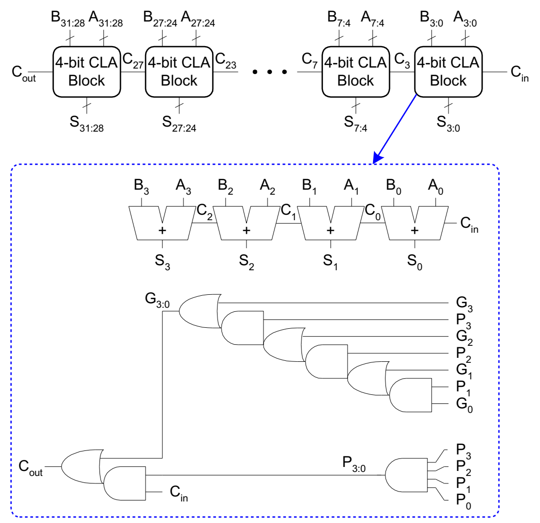
```
module mux2(input  logic [3:0] d0, d1,
            input  logic       s,
            output logic [3:0] y);

  assign y = s ? d1 : d0;
endmodule

module mux2_8(input  logic [7:0] d0, d1,
              input  logic       s,
              output logic [7:0] y);

  mux2 lsbmux(d0[3:0], d1[3:0], s, y[3:0]);
  mux2 msbmux(d0[7:4], d1[7:4], s, y[7:4]);
endmodule
```

Adders

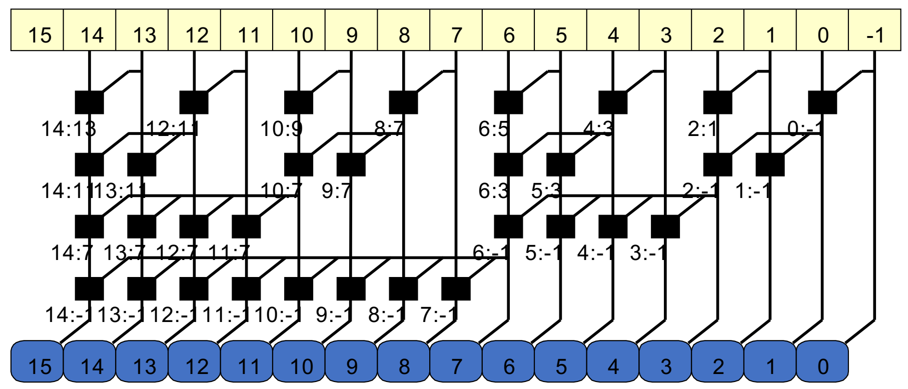
Ripple Carry



Carry Lookahead



Parallel Prefix



15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 -1
14:13 12:11 10:9 8:7 6:5 4:3 2:1 0:-1
14:11 13:11 10:7 9:7 6:3 5:3 2:-1 1:-1
14:7 13:7 12:7 11:7 6:-1 5:-1 4:-1 3:-1
14:-1 13:-1 12:-1 11:-1 10:-1 9:-1 8:-1 7:-1
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0