About the Lab Re-enroll

- Both my lab sections are full
- If you have requested to re-enroll my lab sections, I will approve it as well. But very possibly it will be denied.
- If this applied to your case, you should now resort to other sections, we have enough (4 in total)

Mid-term Review

- Data conversion
- FPGA design
- Combinational circuit
- Sequential Circuit
- Encoder/Decoder
- Verilog

Mid-term Review

- Data conversion
  - Binary to decimal
  - Signed and unsigned
  - One's and Two's complement
  - Hexadecimal to and from Octal

Mid-term Review

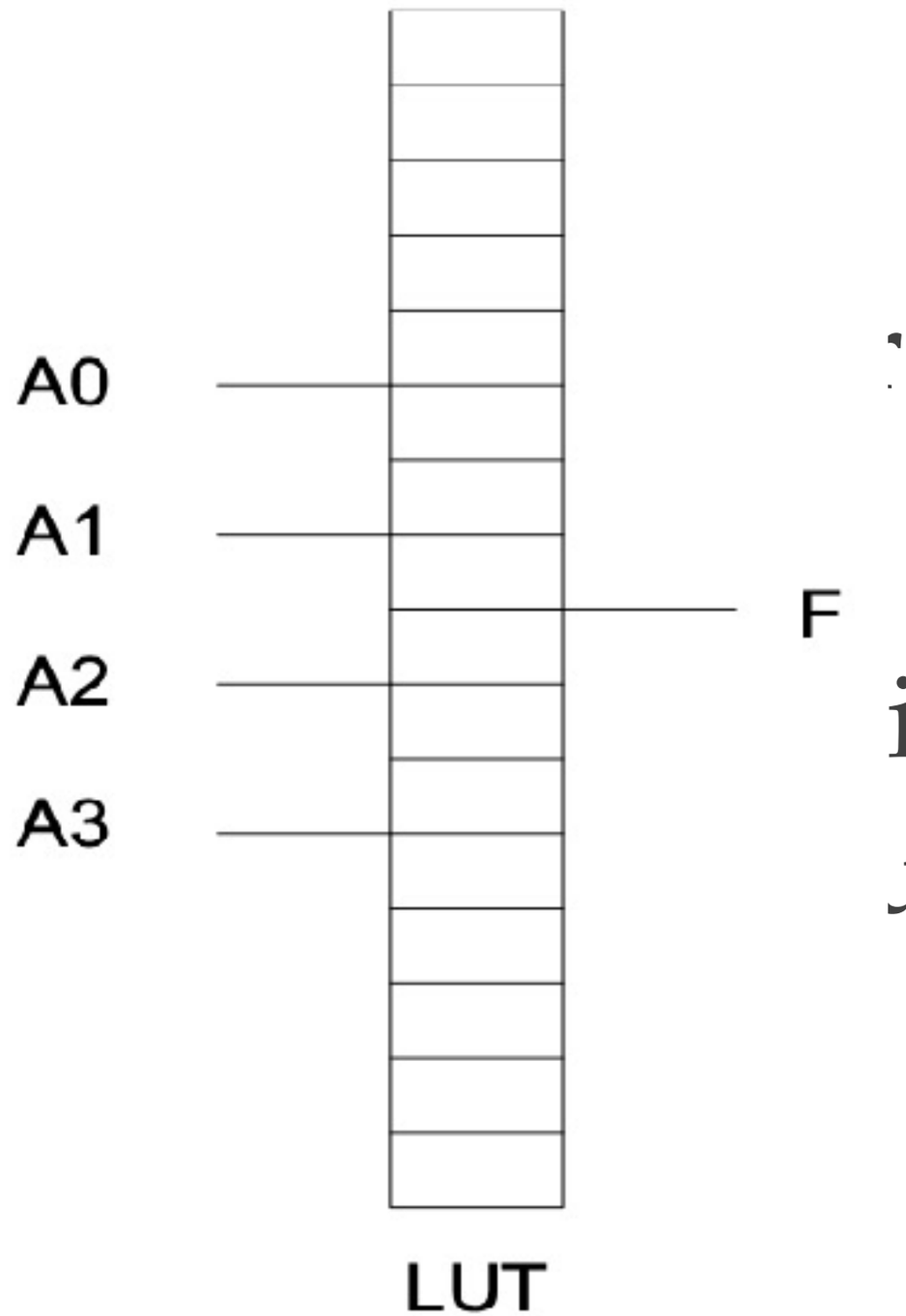
- FPGA design
-

LUT: Look-Up Table

- Look-up tables are how your logic actually gets implemented. Users can program what the output should be for every single possible input
- A LUT consists of a block of RAM that is indexed by the LUT's inputs. The output of the LUT is whatever value is in the indexed location in its RAM cell

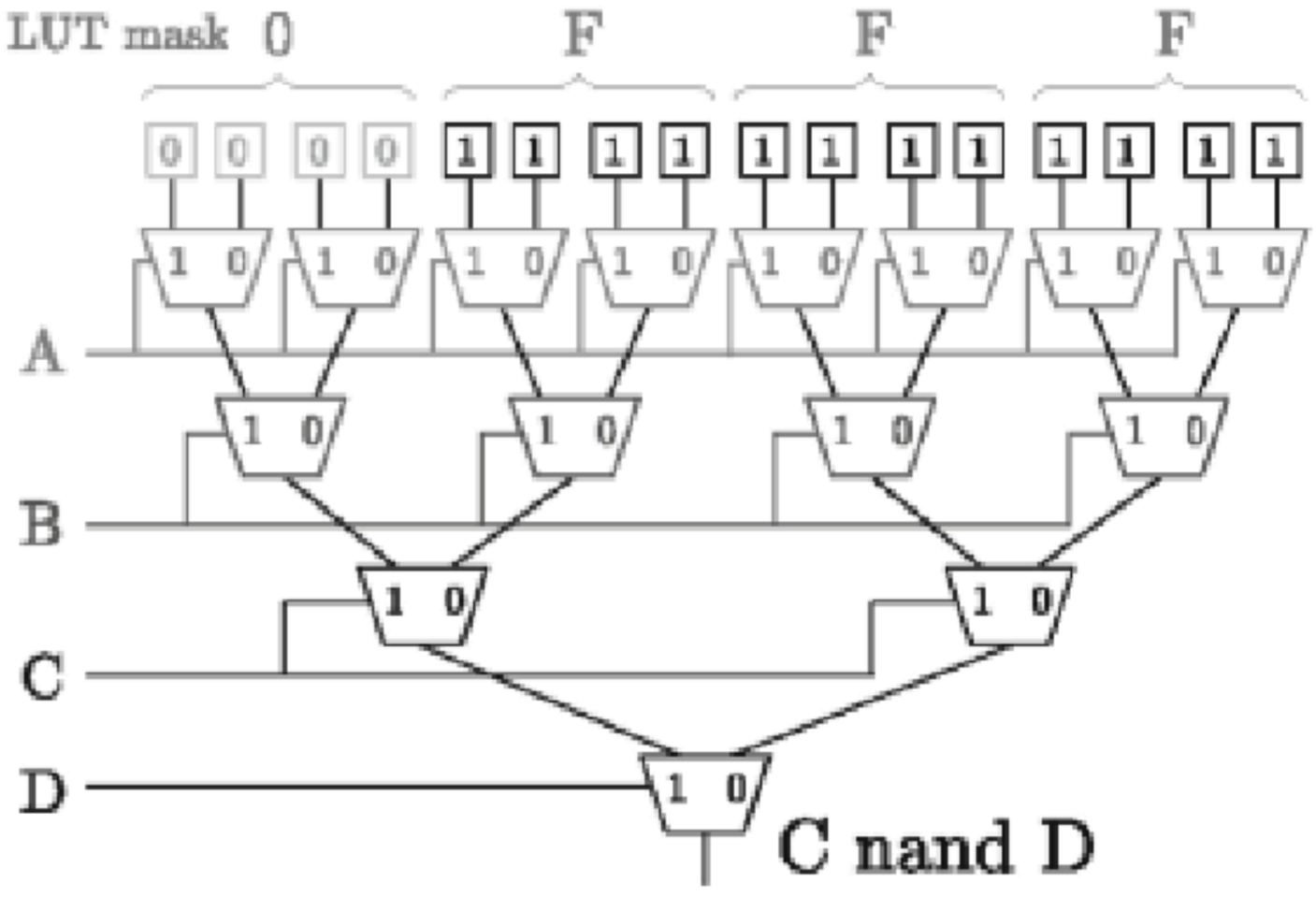
LUT: Look-Up Table

- Look-up tables are ho
  actually gets impleme
  program what the ou
  every single possible
- A LUT consists of a bl
  indexed by the LUT's
  of the LUT is whateve
  indexed location in its



FPGA Design Example

- Lets reverse the function from LUT



FPGA Design Example

- A Boolean Function of four input variables A, B, C and D using a 4-input LUT.
- Here, let the output become high only when any of the two input variables are one.
- What is the hardware realization?

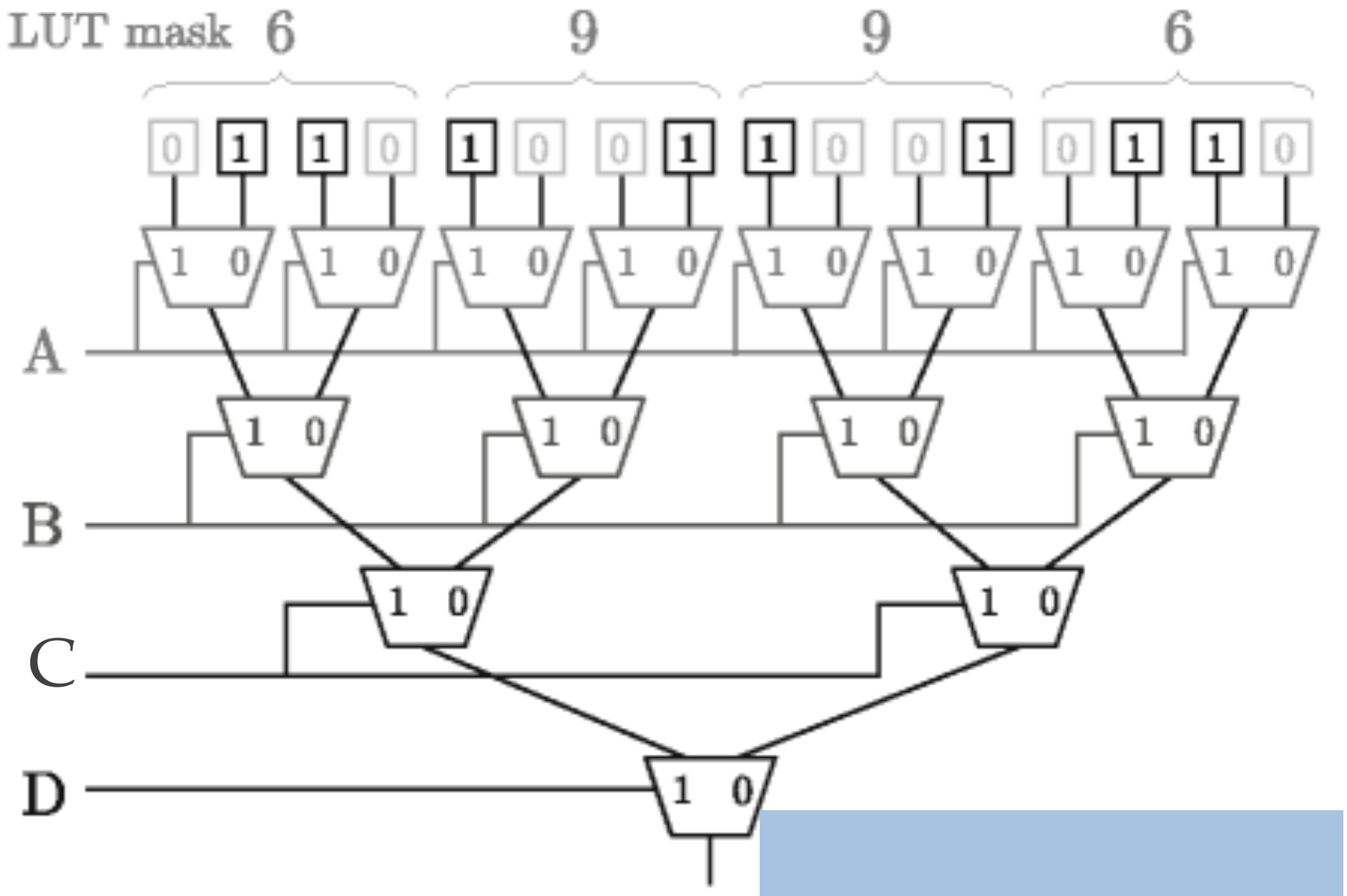
FPGA Design Example

- A Boolean Function of four inp C and D using a 4-input LUT.
- Here, let the output become hig of the two input variables are one.
- What is the hardware realizatic

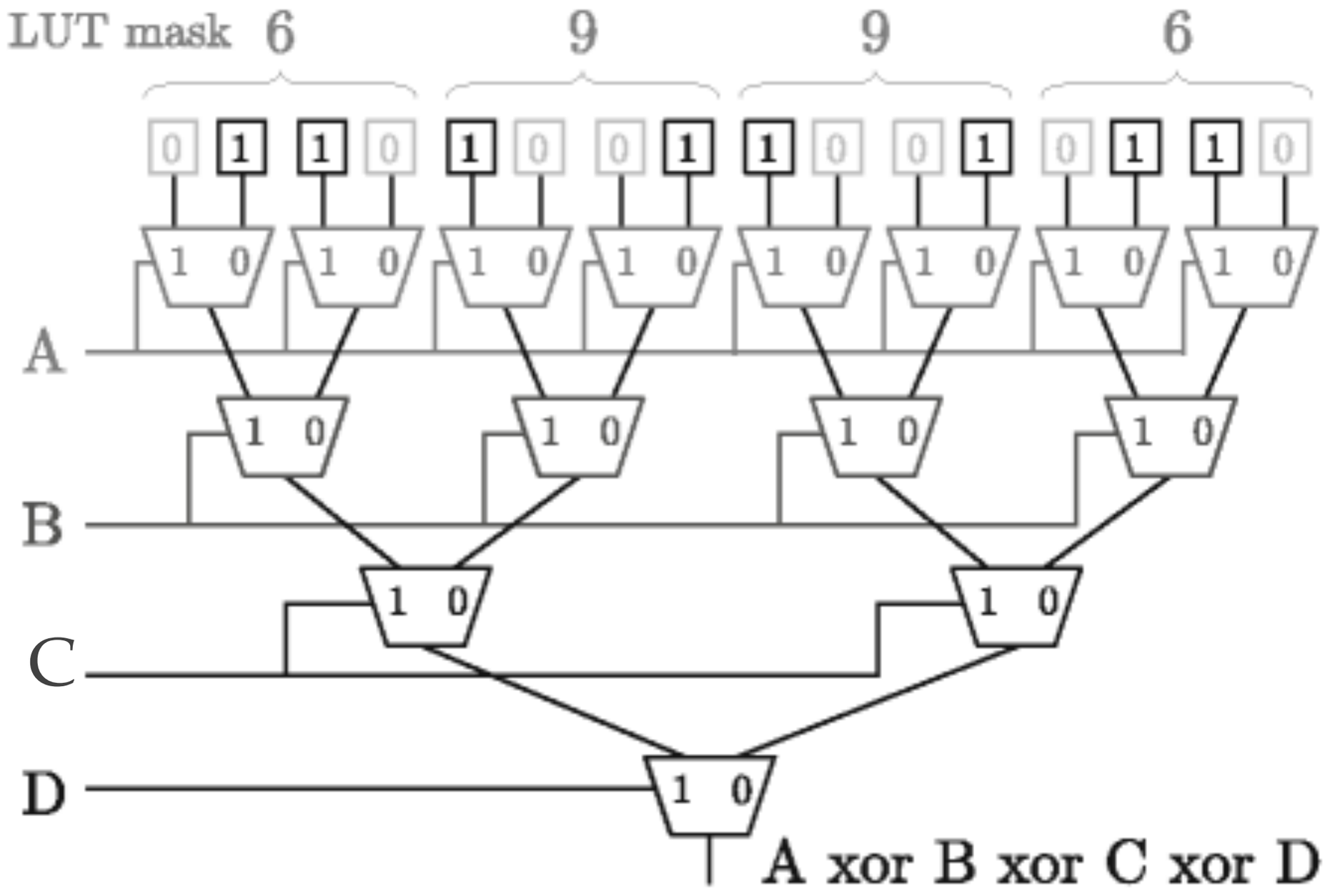
Truth Table

Inputs				Output
A	B	C	D	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

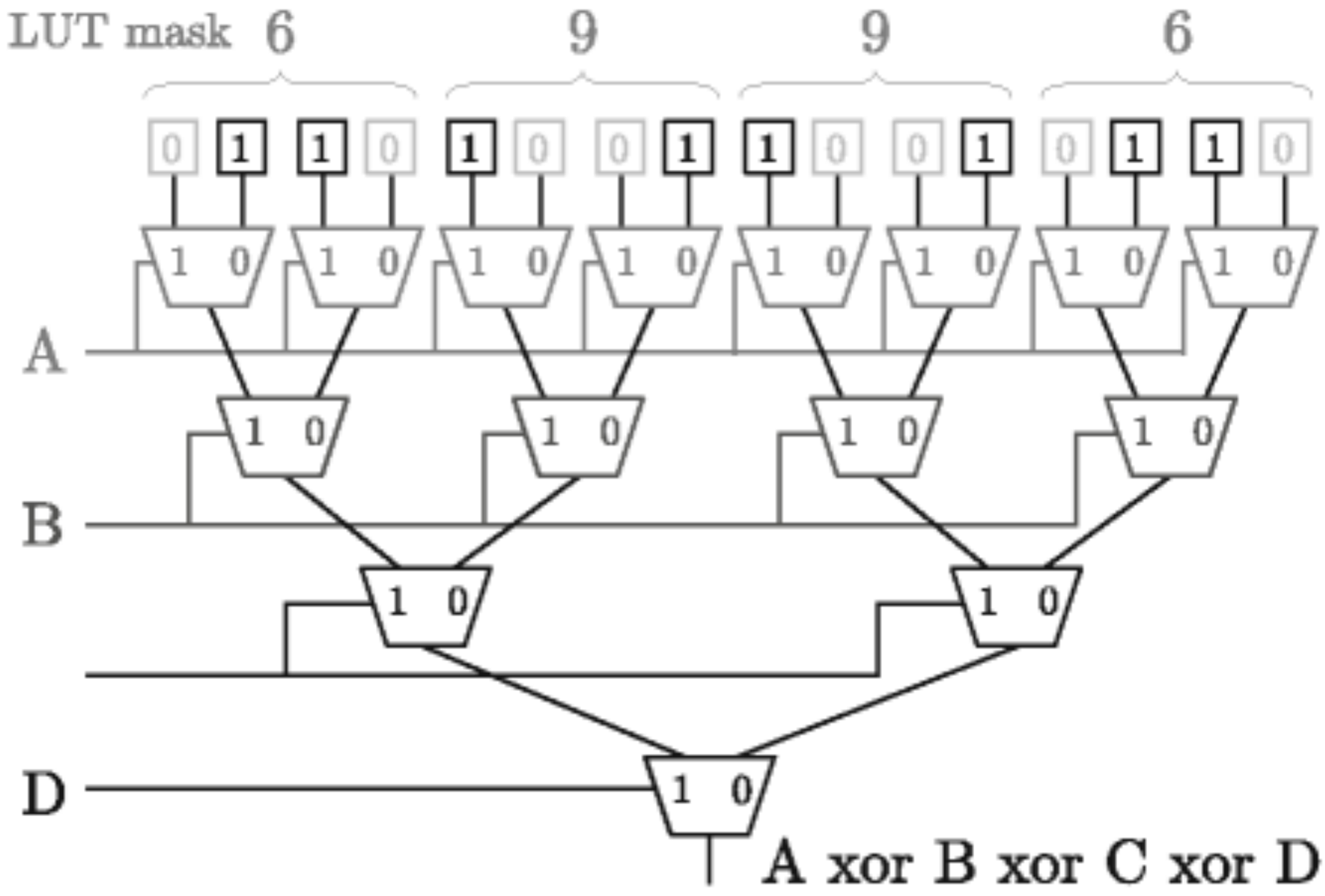
Practice


LUT mask
6
9
9
6
0 1 1 0 1 0 0 1 1 0 0 1 0 1 1 0
1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
A
1 0 1 0 1 0 1 0
B
1 0 1 0
C
1 0
D

Practice



Practice

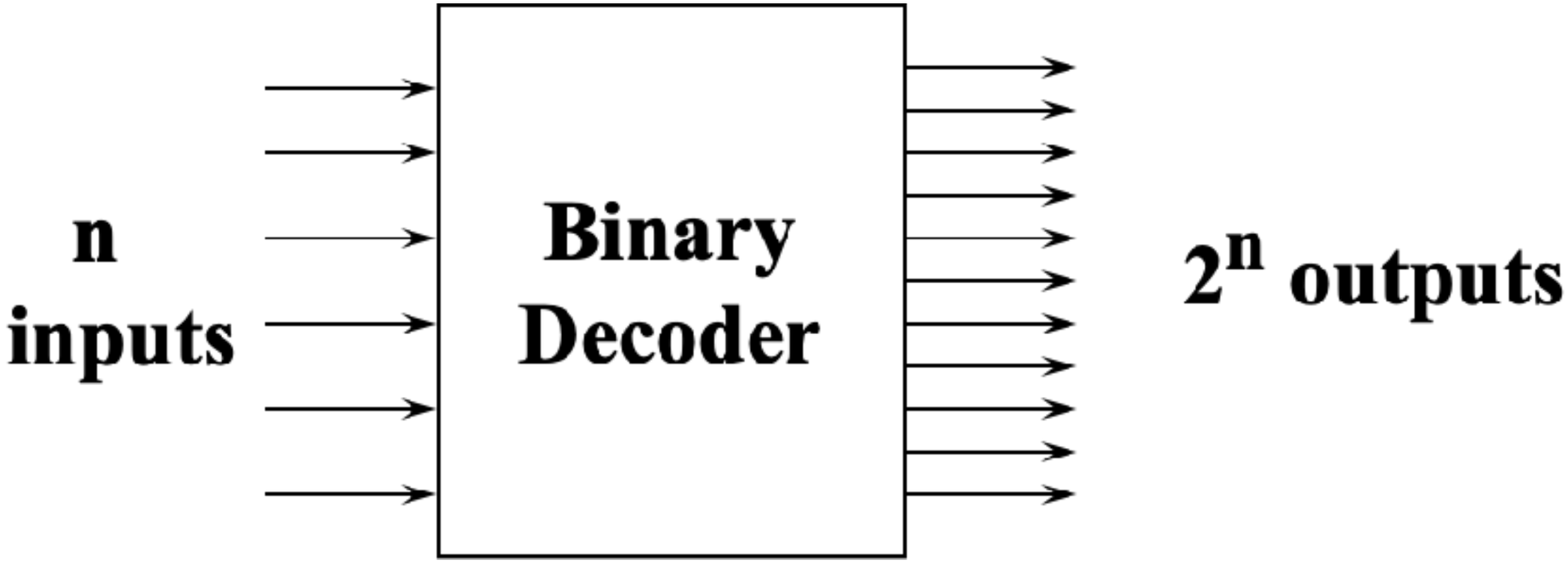


Mid-term Review

- Combinational circuit
- Logic gates
- Encoder
- Decoder

Decoder

- Logic with n input lines and 2n output lines
- Only one output is a 1 for any given input
-



Decoder Example

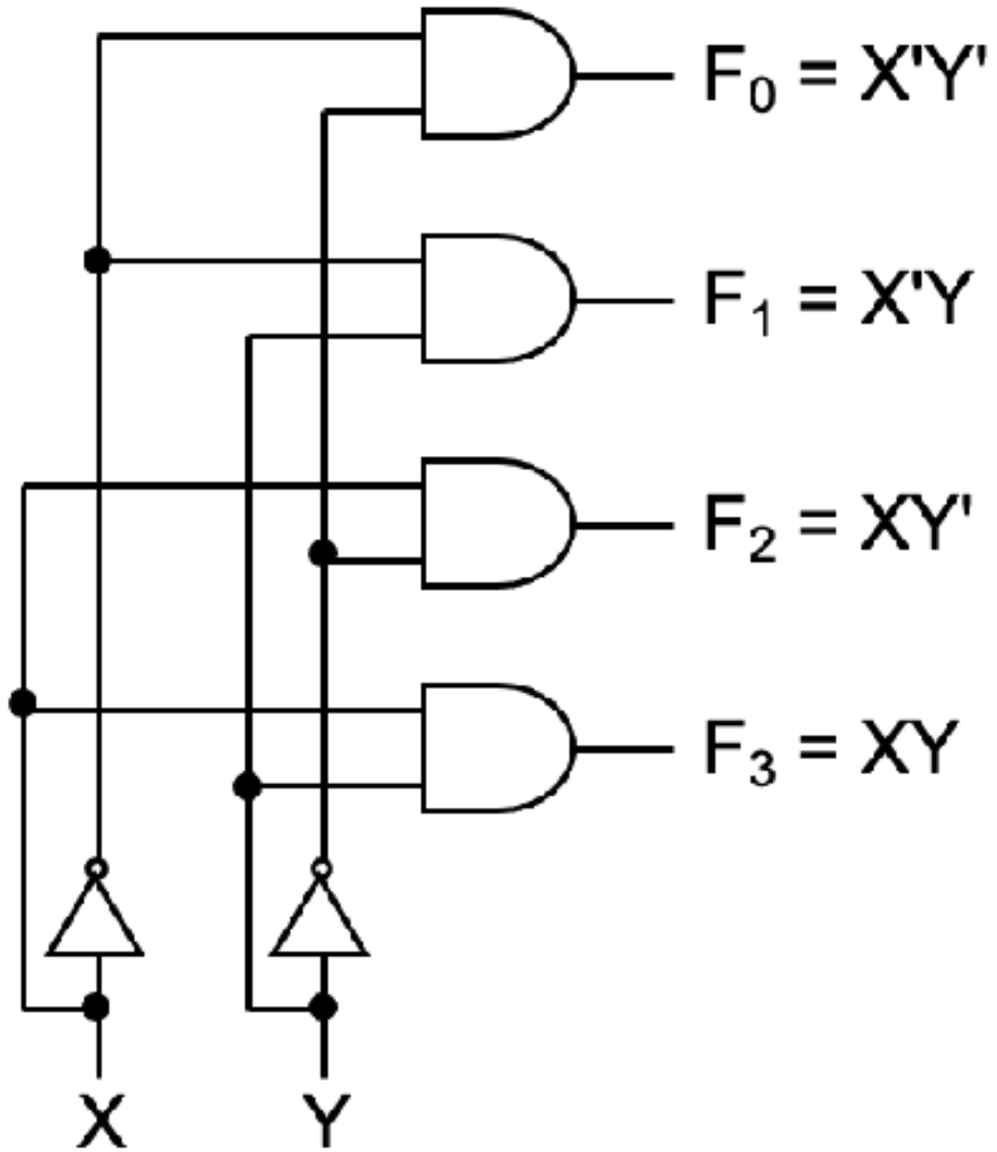
- 2-to-4 decoder

X	Y	F_0	F_1	F_2	F_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Decoder Example

- 2-to-4 decoder

X	Y	F_0	F_1	F_2	F_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

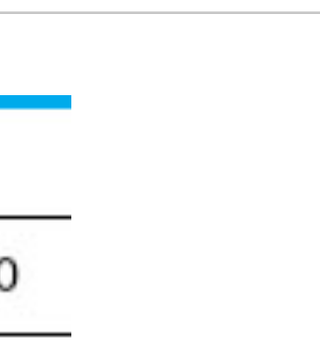


Binary Decoder with ENable

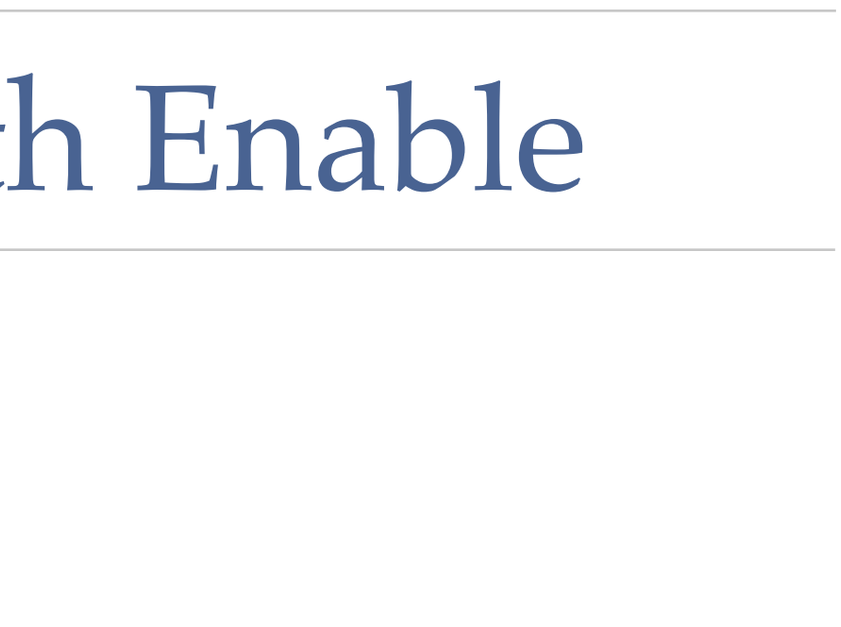
Inputs			*Outputs*			
EN	A1	A0	Y3	Y2	Y1	Y0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

Binary Decoder with Enable

Inputs			*Outputs*			
EN	A1	A0	Y3	Y2	Y1	Y0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0



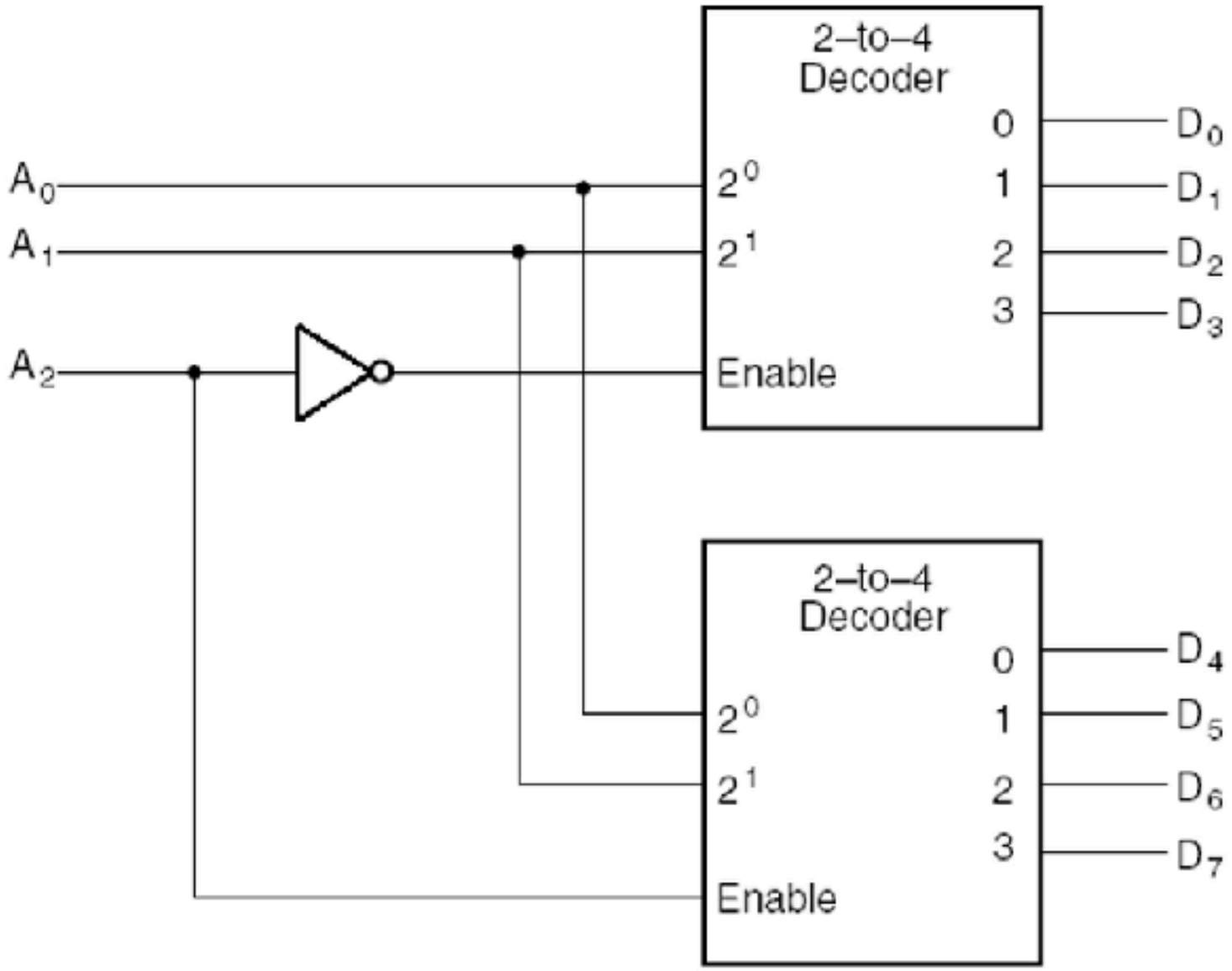
(a)



(b)

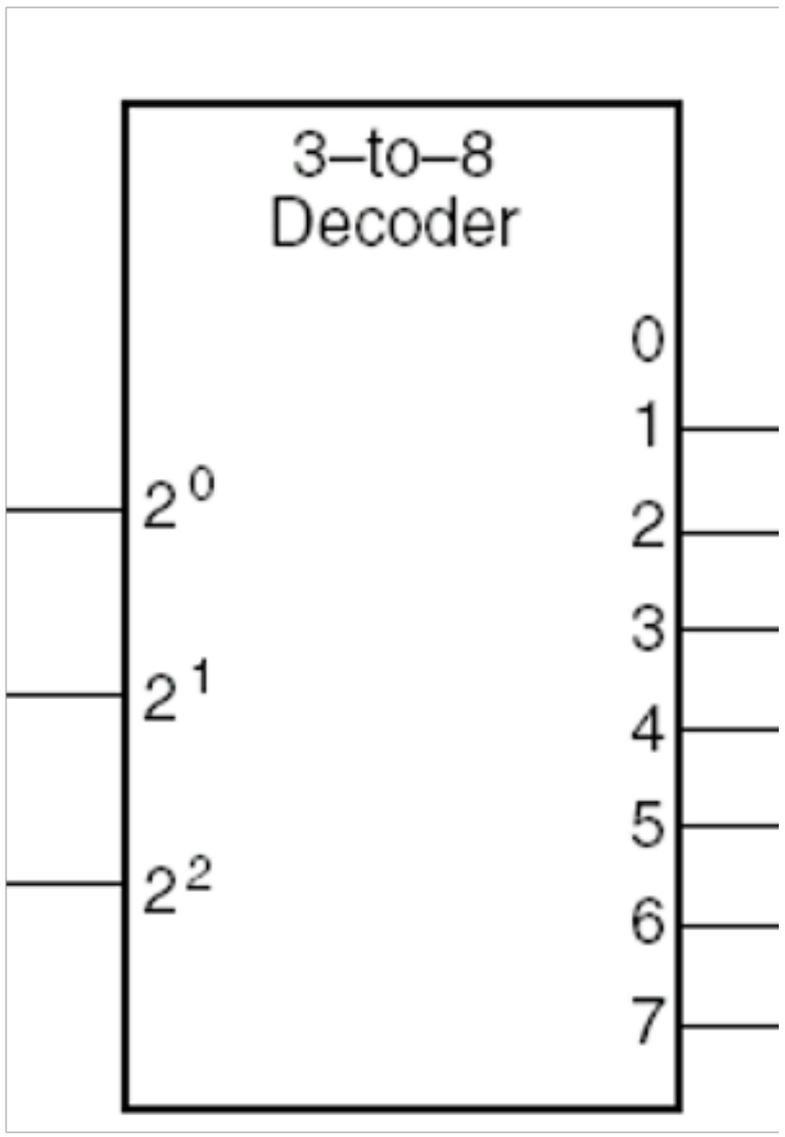
Building Larger Decoder

- Build larger decoders out of two or more smaller decoders
-



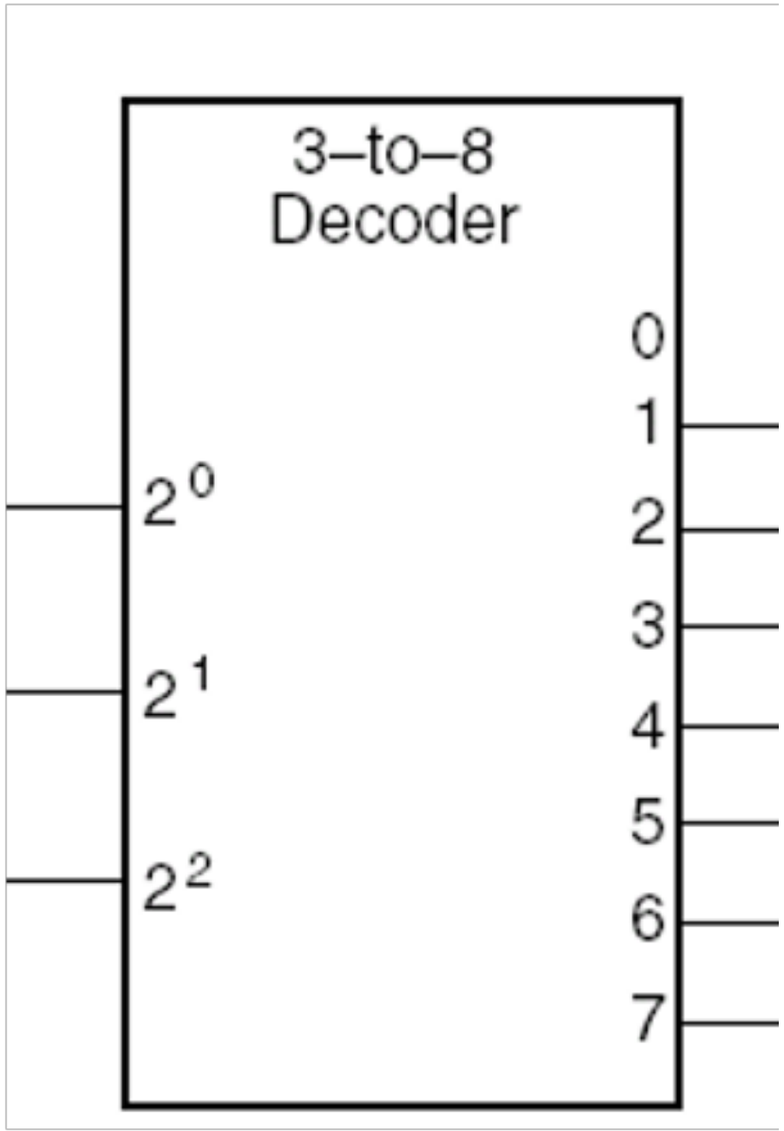
Decoder Example

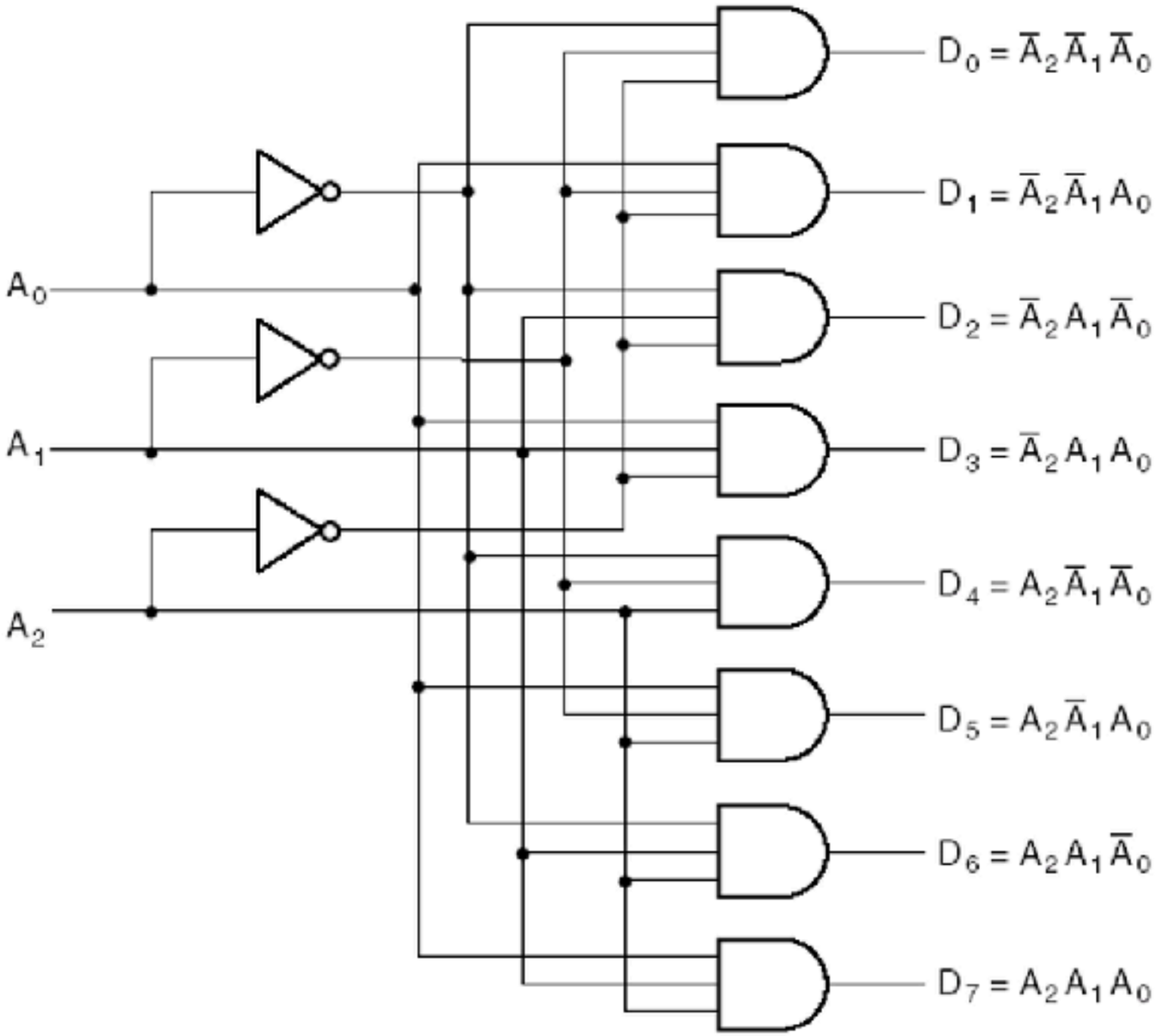
- 3-to-8 decoder



Decoder Example

- 3-to-8 decoder





5-To-32 Decoder

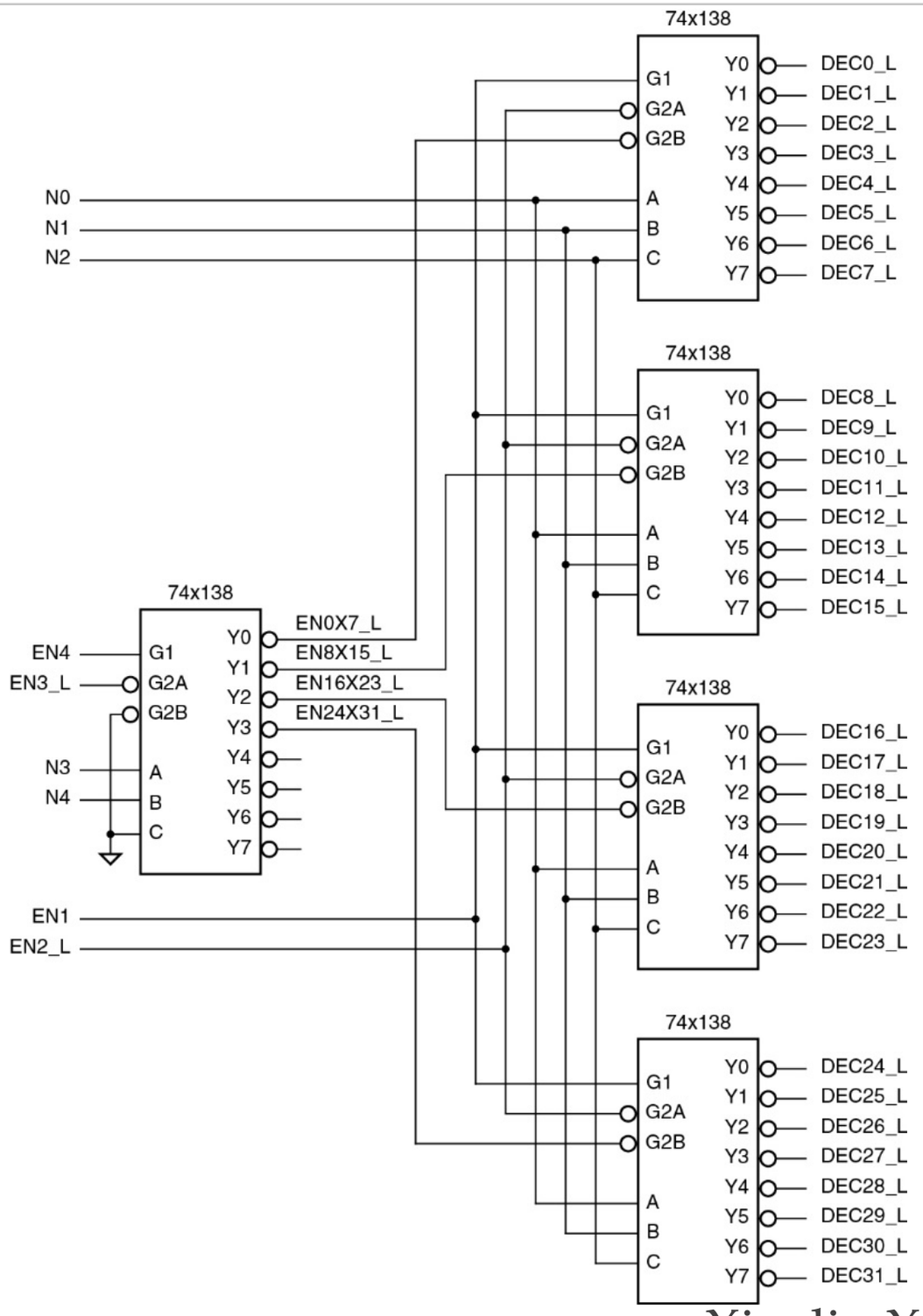
- How?

5-To-32 Decoder

- How?
- Cascading 3-To-8 decoders

5-To-32 Decoder

- How?
- Cascading 3-To-8 decoders



Encoder Example

TABLE 3-5
Truth Table for Octal–to–Binary Encoder

Inputs								Outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

4-to-2 Binary Encoder in Verilog

```
module encoder (Y, W);
   input [3:0] W;
   output [1:0] Y;
   reg [1:0] Y;
   always@ (W)
      case (W)
         4'b1000: Y = 2'b11;
         4'b0100: Y = 2'b10;
         4'b0010: Y = 2'b01;
         4'b0001: Y = 2'b00;
         default: Y = 2'bxx;  //don't care case
      endcase
endmodule
```

Priority Encoder

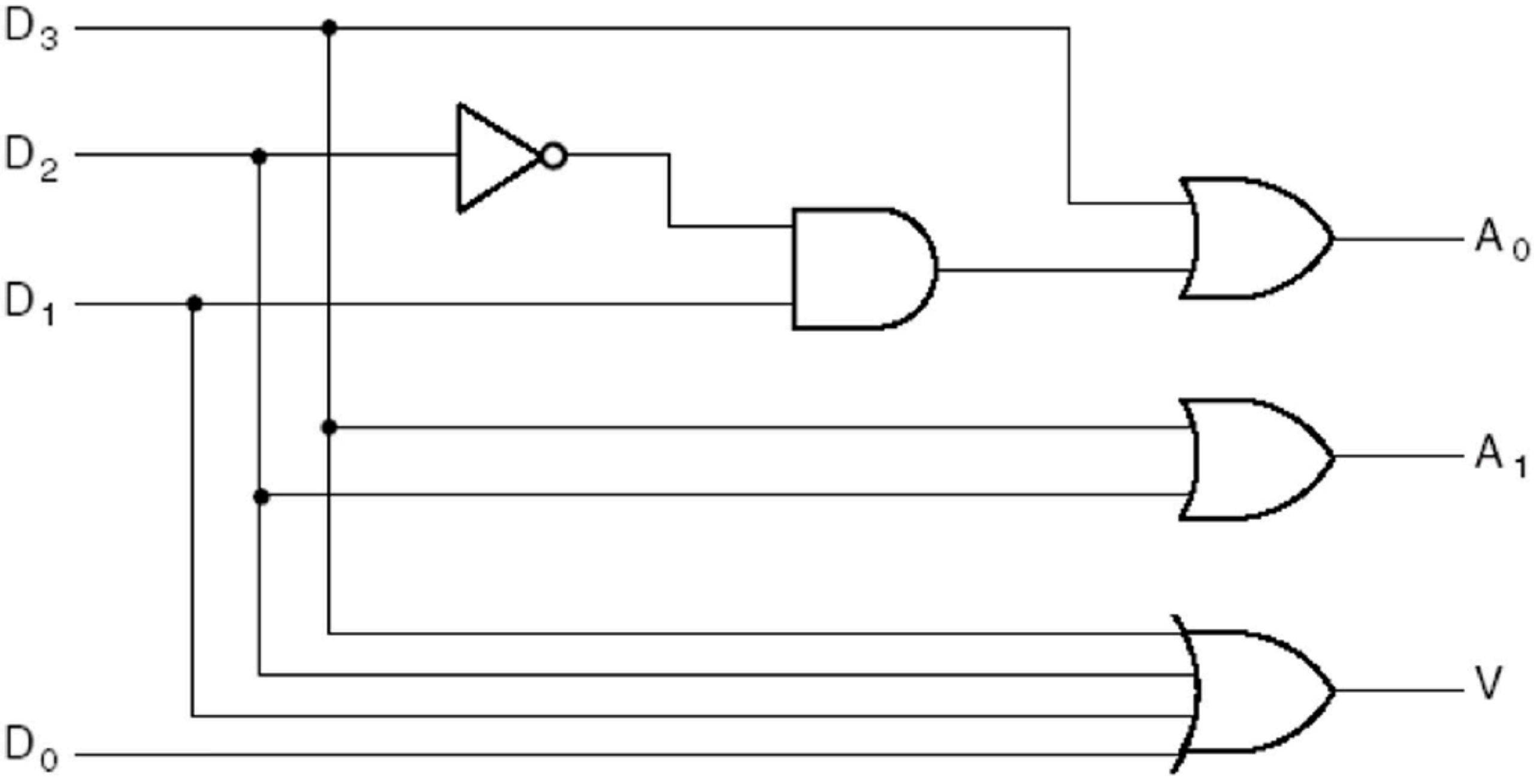
- If two or more inputs are equal to 1 at the same time, the input with highest priority takes precedence.
- Truth Table of 4-Input Priority Encoder

Priority Encoder

- If two or more inputs are equal to 1 at the same time, the input with highest priority takes precedence.
- Truth Table of 4-Input Priority Encoder

Inputs				Outputs		
D_3	D_2	D_1	D_0	A_1	A_0	V
0	0	0	0	X	X	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

Priority Encoder: Circuit Implementation



4-to-2 Priority Encoder in Verilog

```
module priority (Y, V, W);
  input [3:0] W;
  output [1:0] Y;
  output V;
  reg [1:0] Y;
  reg V;
 always@ (W)  begin
      V = 1; // assume valid output
    casex (W)
      4'b1xxx: Y = 2'b11;
      4'b01xx: Y = 2'b10;
      4'b001x: Y = 2'b01;
      4'b0001: Y = 2'b00;
      default: begin
       V = 0;
       Y = 2'bxx;  //don't care case
        end
    endcase
  end
endmodule
```

Mid-term Review

- Sequential Circuit
- Latch
- Flip-flop

Difference Between Combinational and Sequential Circuits

- Combinational Circuit
  - Time independent circuits
  - Do not depend on previous inputs to generate any output
  -

Figure: Combinational Circuits

Difference Between Combinational and Sequential Circuits

- Sequential Circuit
  - Dependent on
    - Clock cycles
    - Present inputs
    - Past inputs
  - to generate any

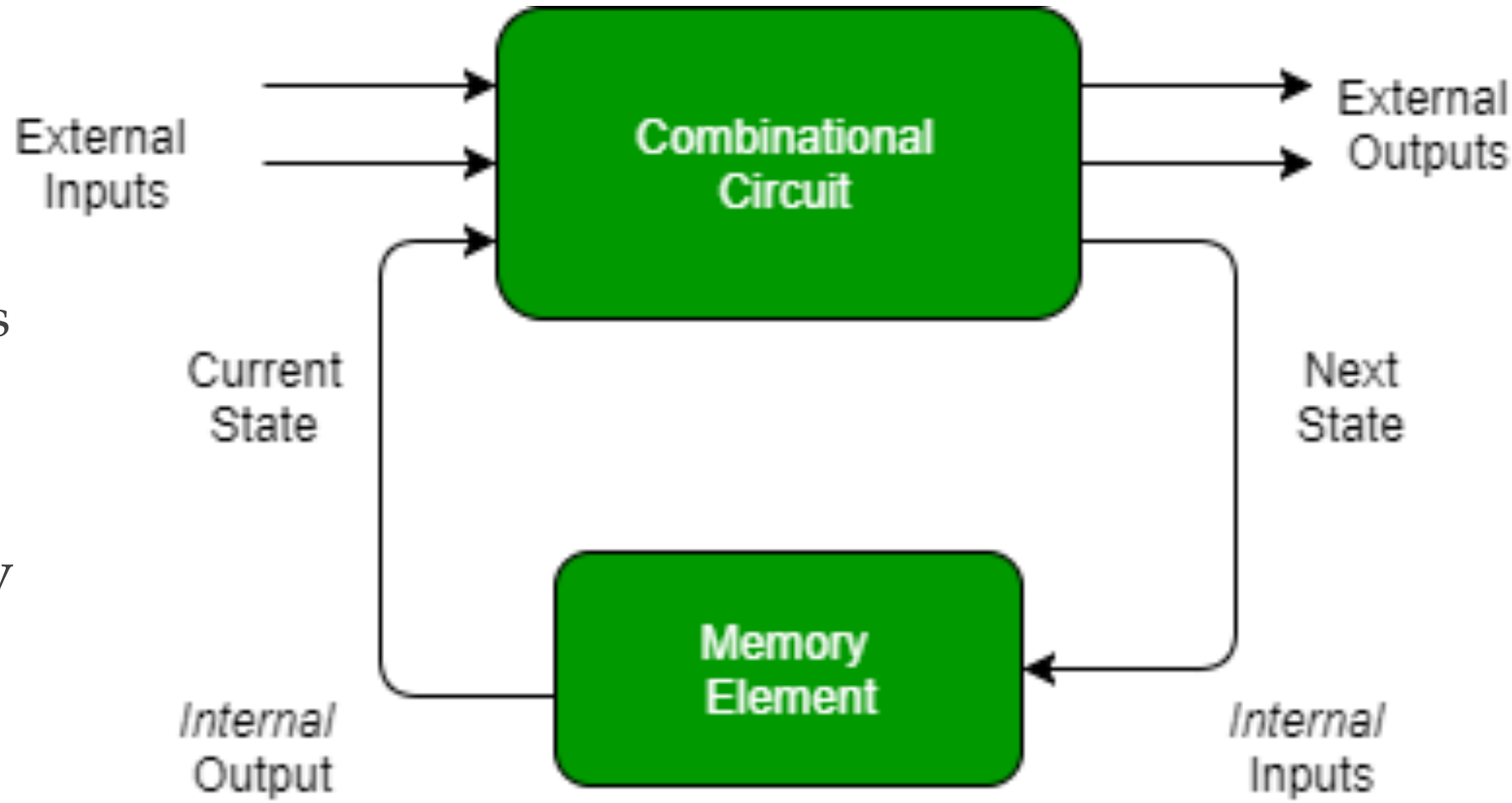
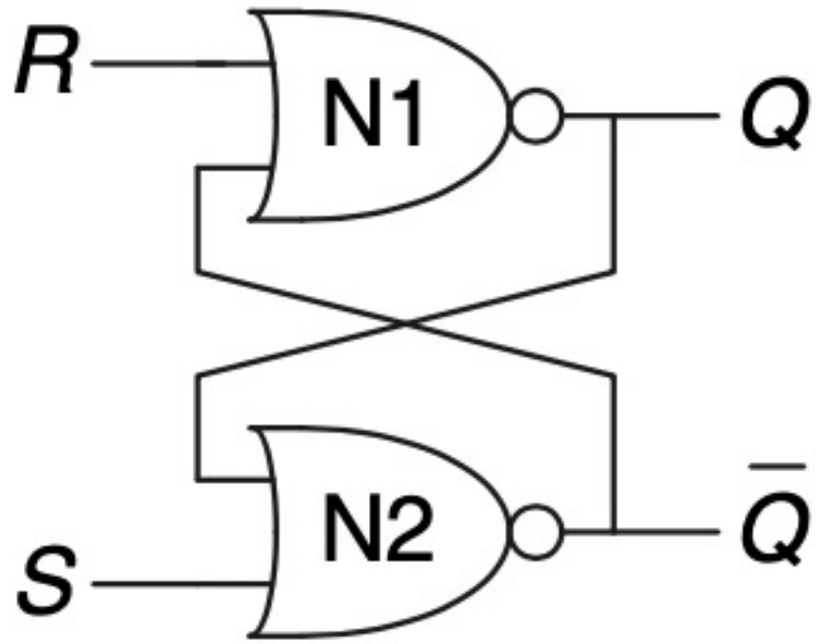


Figure: Sequential Circuit

Sequential Logic (2): SR Latch

- SR latch
  - S: Set
  - R: Reset

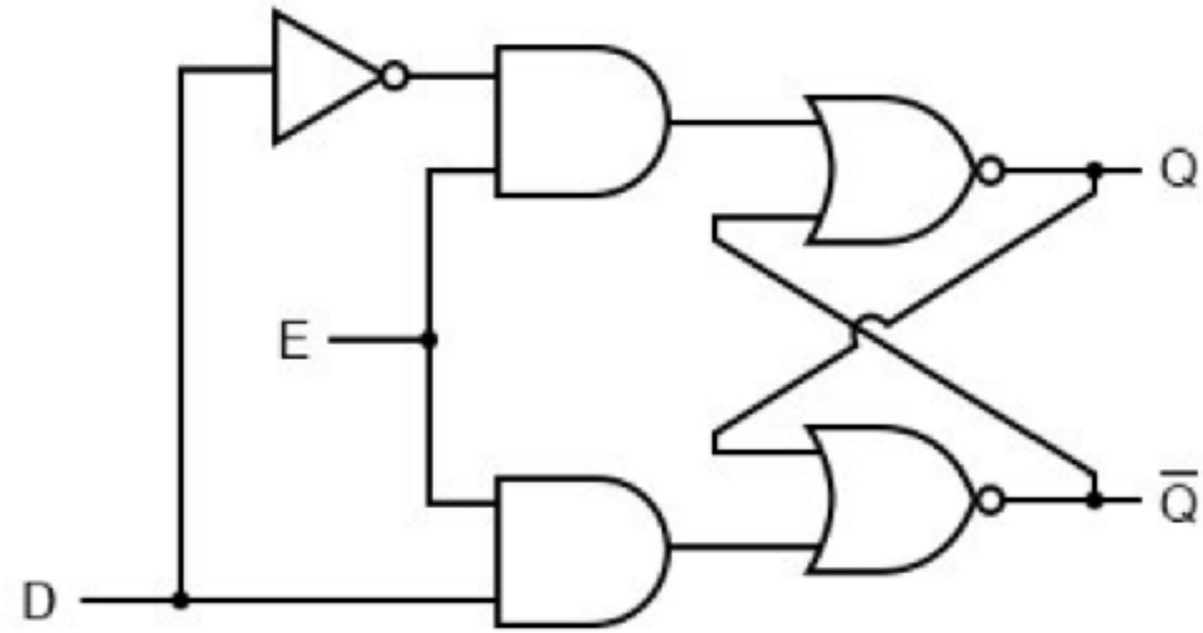


- Truth table of SR latch

R	S	Q	~Q
1	0		
0	1		
1	1		
0	0		

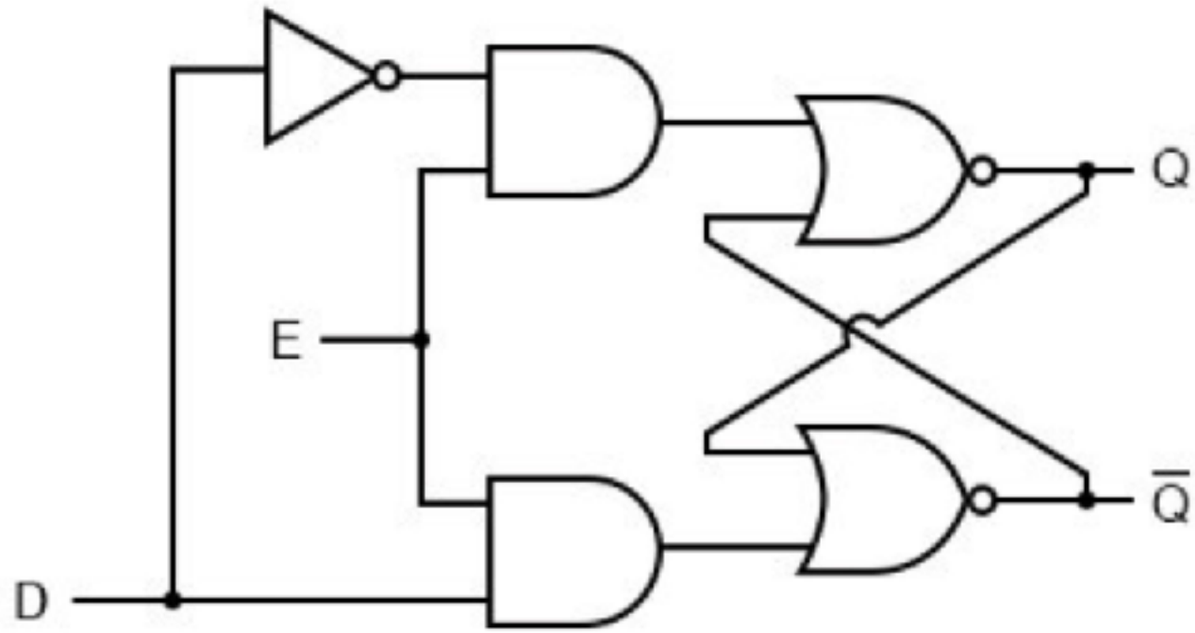
Sequential Logic (2): D Latch

- ***The current design CANNOT latch the D value***
- How to solve this issue?
  - Adding an ***Enable***



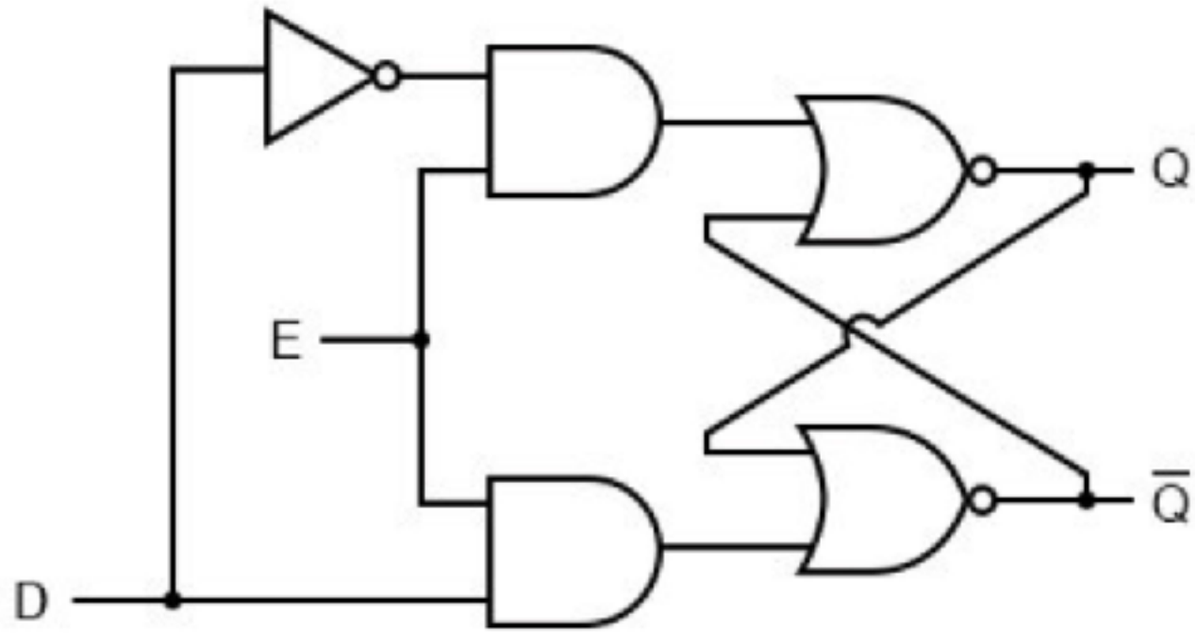
Sequential Logic (2): D Latch

- ***The current design CANNOT latch the D value***
- How to solve this issue?
  - Adding an ***Enable***
- Truth table



Sequential Logic (2): D Latch

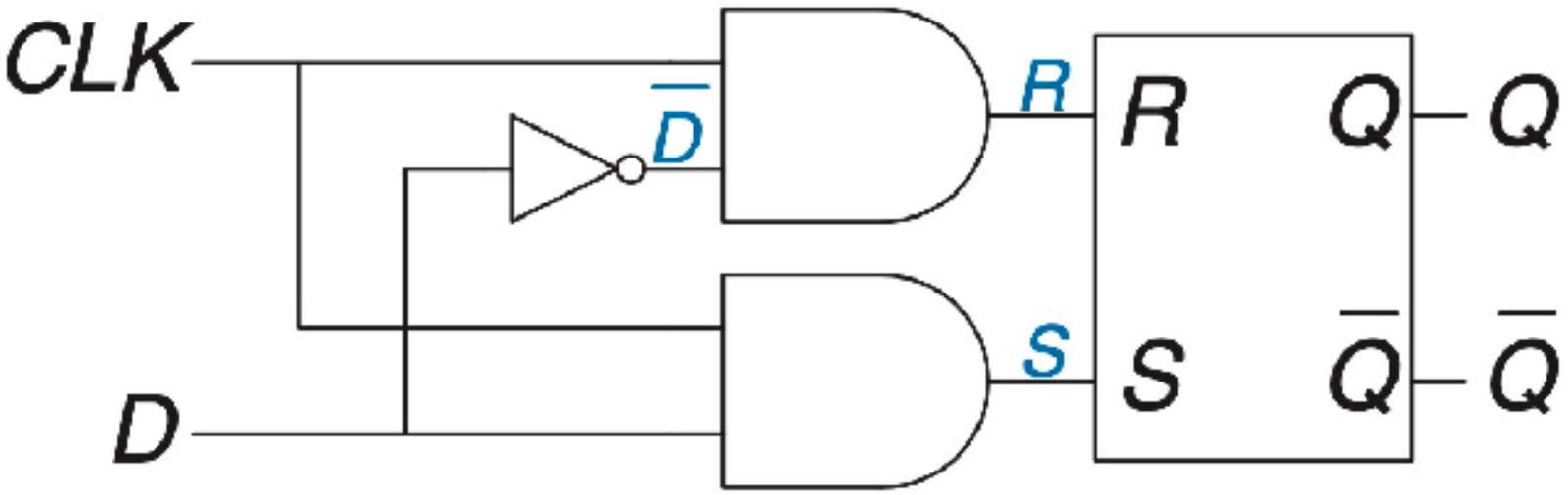
- ***The current design CANNOT latch the D value***
- How to solve this issue?
  - Adding an ***Enable***
- Truth table

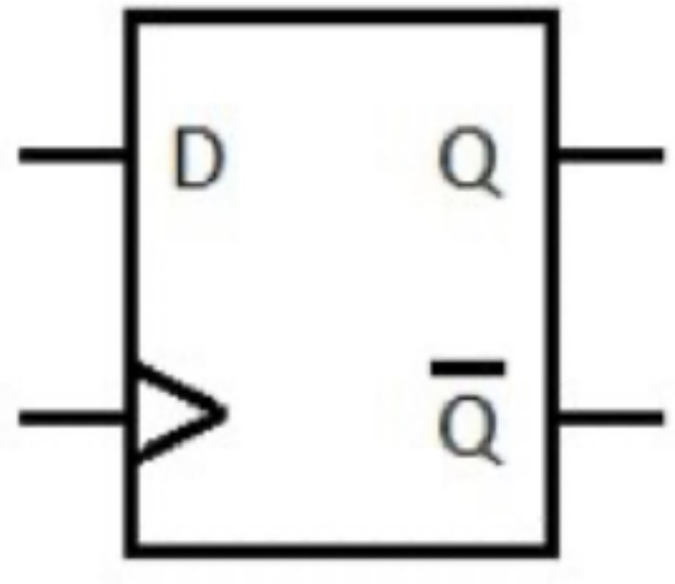


E	D	Q	~Q
1	0		
0	1		
0	1		
0	0		

Sequential Logic (2): D Latch

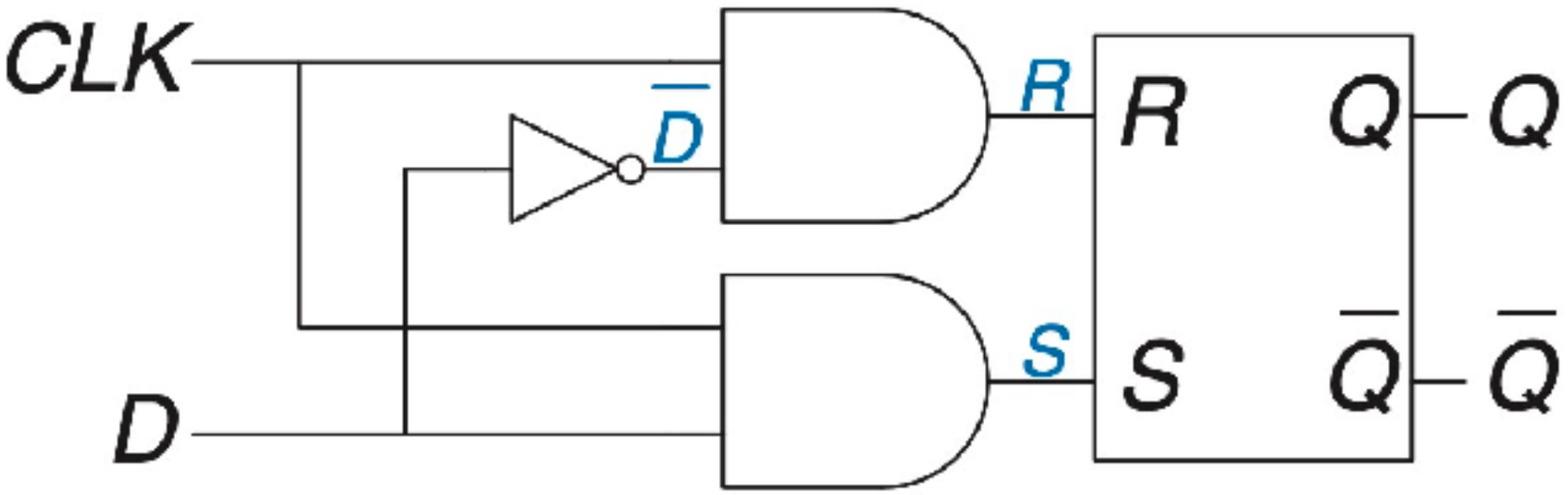
- Replacing **Enable** with **CLK** (clock)

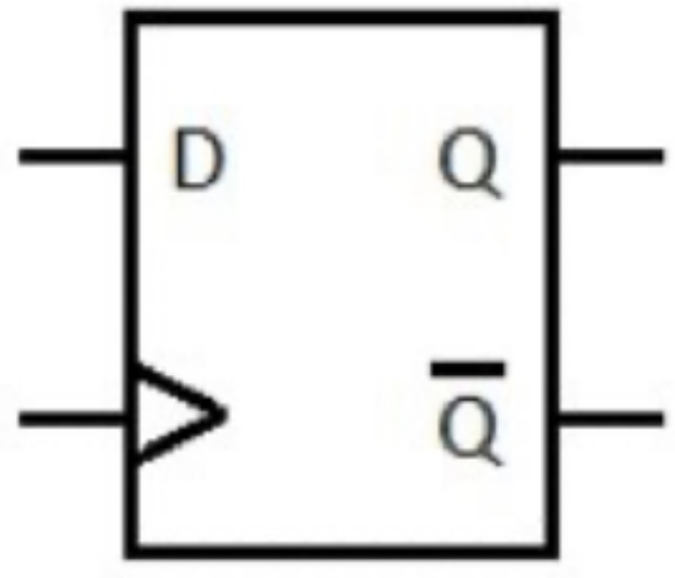




Sequential Logic (2): D Latch

- Replacing **Enable** with **CLK** (clock)



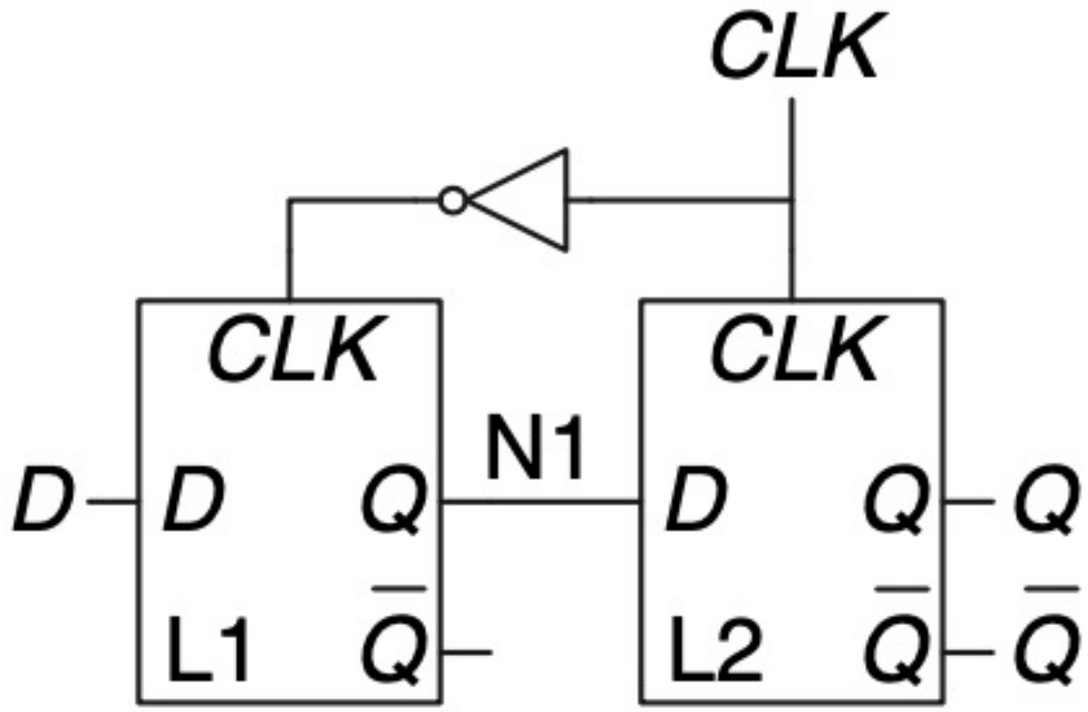


CLK	D	$\overline{D}$	S	R	Q	$\overline{Q}$
0	X	$\overline{X}$	0	0	Q_{prev}	$\overline{Q}_{prev}$
1	0	1	0	1	0	1
1	1	0	1	0	1	0

Sequential Logic (3): D Flip-flop

- Sequential circuit is edge-sensitive!
- Truth table
  - CLK = 0, first transparent
  - CLK = 1, second transparent

CLK	D	N1	Q	~Q
0				
1				
0				
1				



A D flip-flop copies D to Q on the rising edge of the clock, and remembers its state at all other times

Timing Diagram / Behavior of Latch and Flip-flop

Ripple-Carry Adder

- RCA

Ripple-Carry Adder

- RCA
  - The Cout of LSB is cascaded/used as the Cin of its LSB+1

Ripple-Carry Adder

- RCA
  - The Cout of LSB is cascaded/used as the Cin of its LSB+1
  - Pros

Ripple-Carry Adder

- RCA
  - The Cout of LSB is cascaded/used as the Cin of its LSB+1
  - Pros
    - Straightforward, easy topology

Ripple-Carry Adder

- RCA
  - The Cout of LSB is cascaded/used as the Cin of its LSB+1
  - Pros
    - Straightforward, easy topology
  - Cons

Ripple-Carry Adder

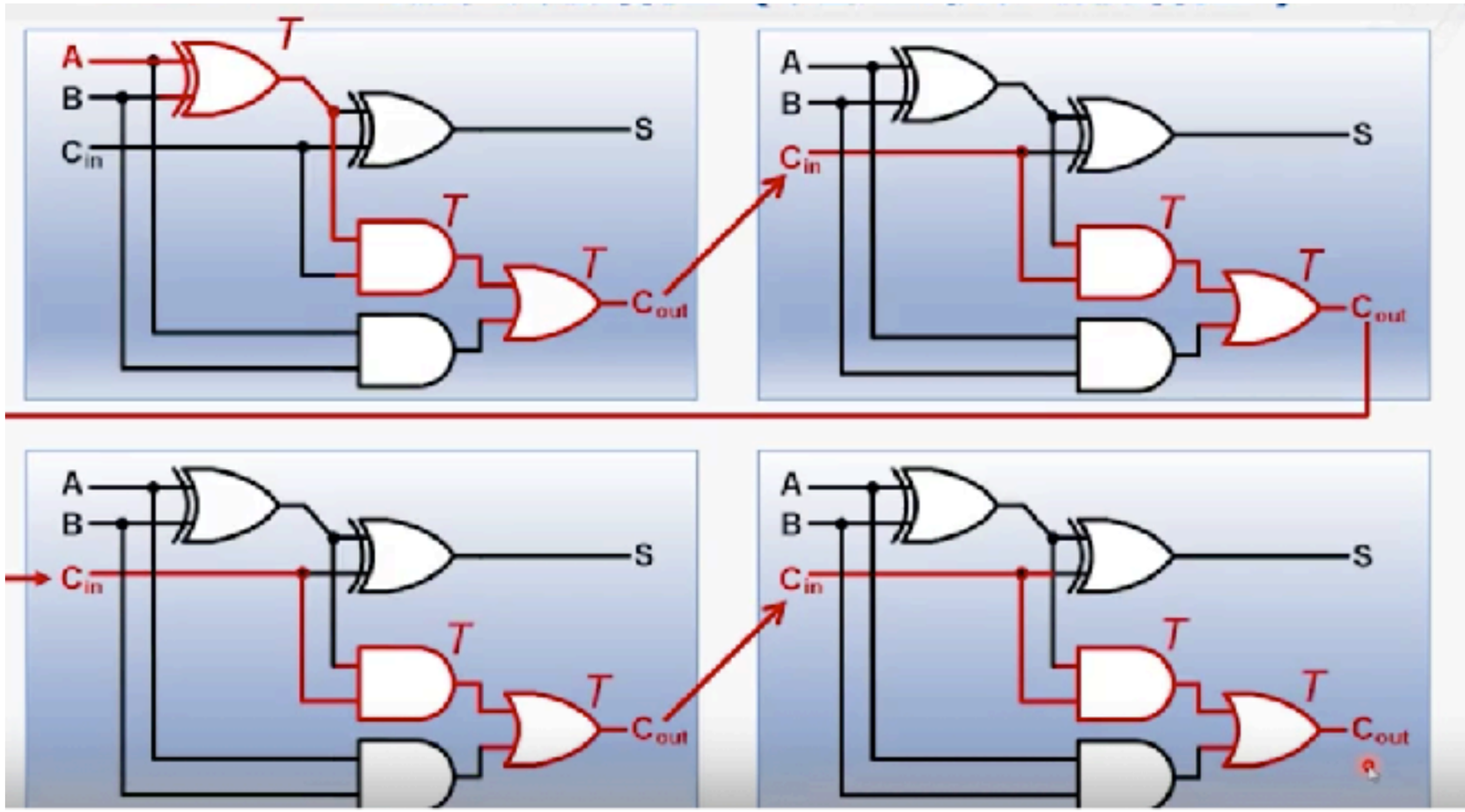
- RCA
  - The Cout of LSB is cascaded/used as the Cin of its LSB+1
  - Pros
    - Straightforward, easy topology
  - Cons
    - Long-latency

Ripple-Carry Adder

- RCA
  - The Cout of LSB is cascaded/used as the Cin of its LSB+1
  - Pros
    - Straightforward, easy topology
  - Cons
    - Long-latency
    - MSB will have to wait for the results from LSB!

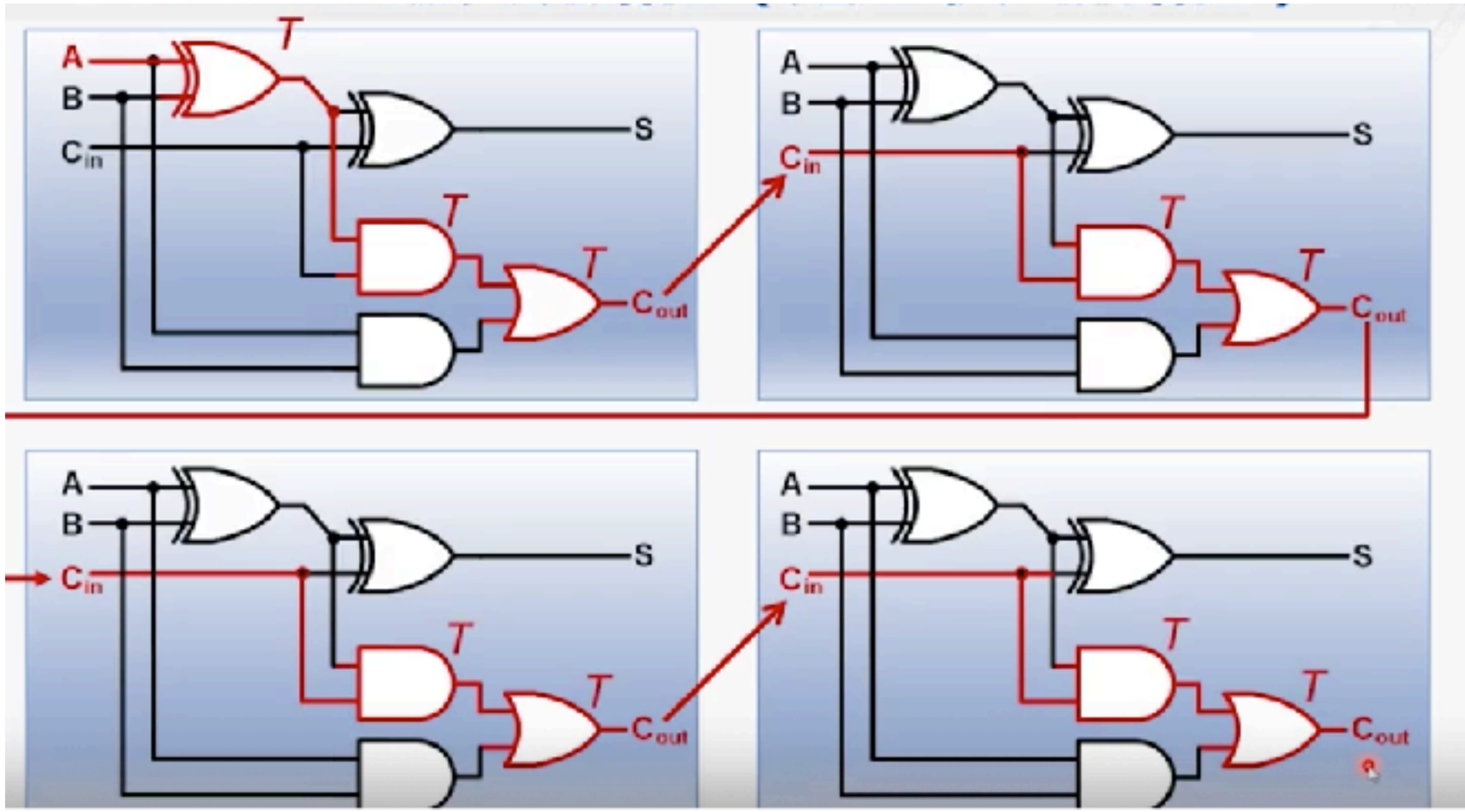
Delay Formulation

- Each gate has a delay of T (ideal)
- How much delay in total?
  - ***hand calculation***



Delay Formulation

- Each gate has a delay of T (ideal)
- How much delay in total?
  - ***hand calculation***
  - $2T^*N+T$

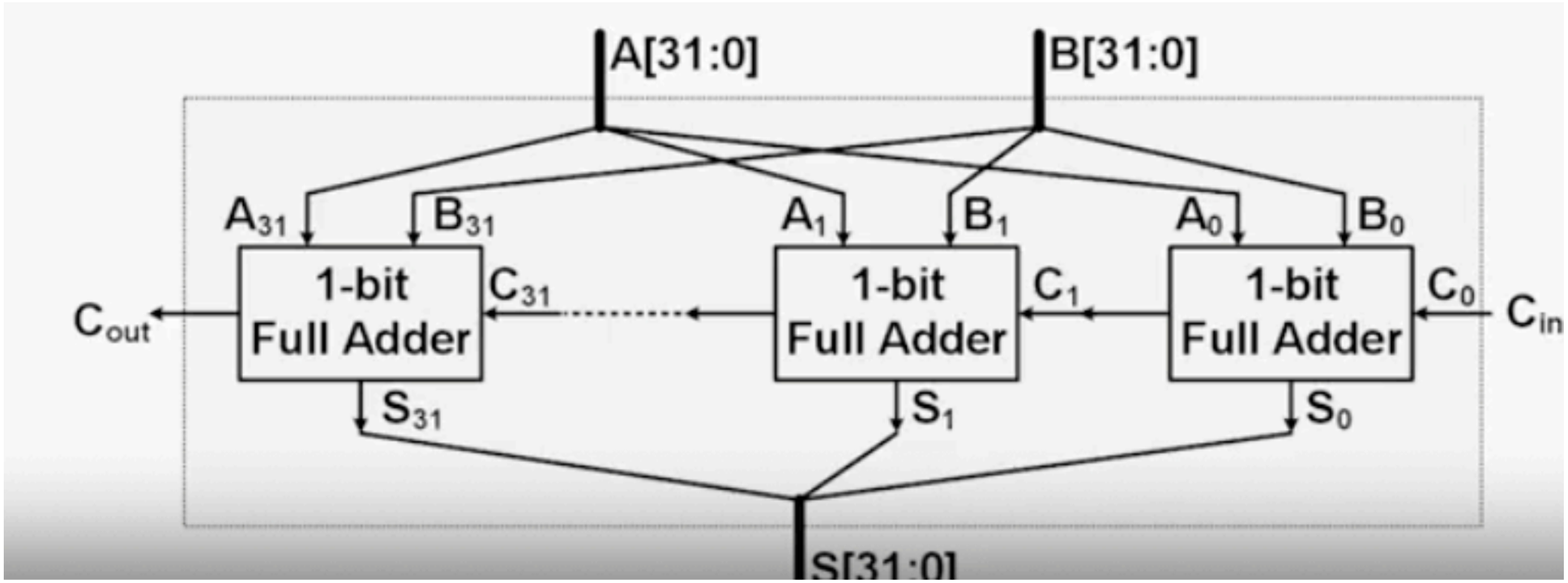


Delay Formulation

- How much delay for a 32-bit full adder?
- (T+T)*N+T where N=32 = 65T

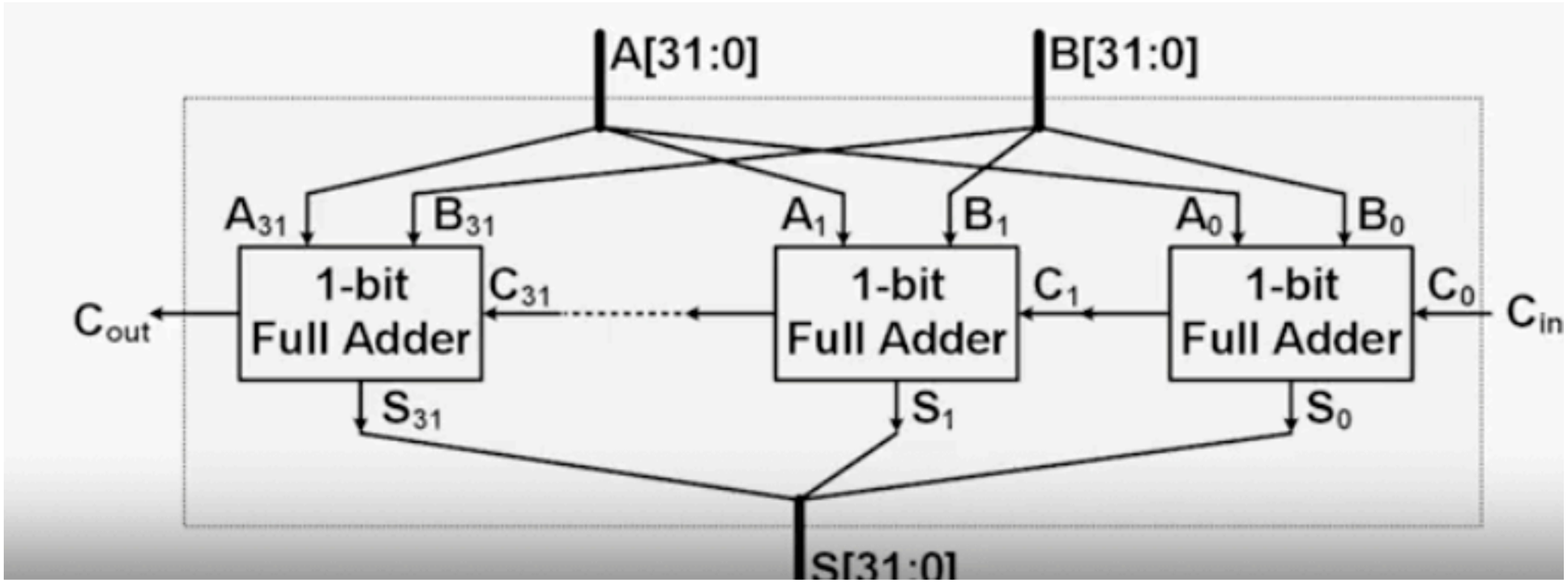
Delay Formulation

- How much delay for a 32-bit full adder?
- (T+T)*N+T where N=32 = 65T



Delay Formulation

- How much delay for a 32-bit full adder?
- (T+T)*N+T where N=32 = 65T
- Good? Bad? What this number means?



Carry-Lookahead Adder

- Proposed for addressing the low-performance of ripple-carry adder
- How?

$$C_{i+1} = (A_i \cdot B_i) + (A_i \cdot C_i) + (B_i \cdot C_i)$$
$$= (A_i \cdot B_i) + (A_i + B_i) \cdot C_i$$

- The i-th adder will **generate** a carry-out, if both A and B are 1, ***independent of carry_in!***
- The i-th adder will **propagate** a carry-out, if either A or B is 1, ***and there is a of carry_in!***

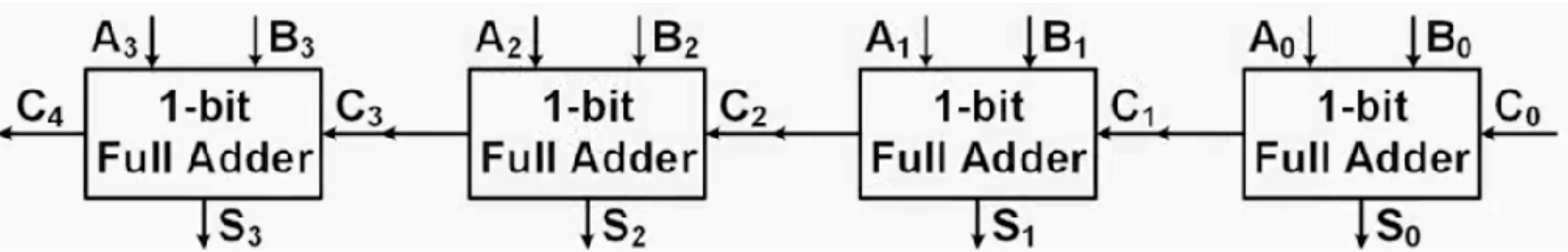
Analysis of Cout

- Define two new parameters:
  - Generate: Gi = Ai AND Bi
  - Propagate: Pi = Ai + Bi

$$C_{i+1} = G_i + P_i \cdot C_i$$

$$C_{i+1} = (A_i \cdot B_i) + (A_i \cdot C_i) + (B_i \cdot C_i)$$

$$= (A_i \cdot B_i) + (A_i + B_i) \cdot C_i$$



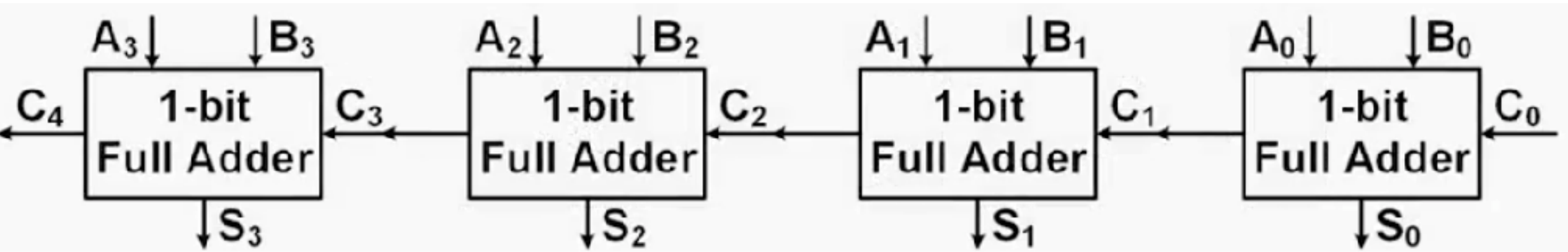
Analysis of Cout

- Define two new parameters:
  - Generate: Gi = Ai AND Bi
  - Propagate: Pi = Ai + Bi

$$C_{i+1} = G_i + P_i \cdot C_i$$

$$C_{i+1} = (A_i \cdot B_i) + (A_i \cdot C_i) + (B_i \cdot C_i)$$

$$= (A_i \cdot B_i) + (A_i + B_i) \cdot C_i$$



Know the Values in Advance

$$C_{i+1}=G_i+P_i \cdot C_i$$

Know the Values in Advance

$$C_1 = G_0 + P_0 \cdot C_0$$

$$C_{i+1} = G_i + P_i \cdot C_i$$

Know the Values in Advance

$$C_1 = G_0 + P_0 \cdot C_0$$

$$C_2 = G_1 + P_1 \cdot C_1$$

$$C_{i+1} = G_i + P_i \cdot C_i$$

Know the Values in Advance

$$C_1 = G_0 + P_0 \cdot C_0$$

$$C_2 = G_1 + P_1 \cdot C_1$$

$$= G_1 + P_1 \cdot (G_0 + P_0 \cdot C_0)$$

$$\boxed{C_{i+1} = G_i + P_i \cdot C_i}$$

Know the Values in Advance

$$C_1 = G_0 + P_0 \cdot C_0$$

$$\begin{aligned} C_2 &= G_1 + P_1 \cdot C_1 \\ &= G_1 + P_1 \cdot (G_0 + P_0 \cdot C_0) \\ &= G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0 \end{aligned}$$

$$C_{i+1} = G_i + P_i \cdot C_i$$

Know the Values in Advance

$$C_1 = G_0 + P_0 \cdot C_0$$

$$\begin{aligned} \mathbf{C_2} &= G_1 + P_1 \cdot C_1 \\ &= G_1 + P_1 \cdot (G_0 + P_0 \cdot C_0) \\ &= \mathbf{G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0} \end{aligned}$$

$$\begin{aligned} \mathbf{C_3} &= G_2 + P_2 \cdot C_2 \\ &= G_2 + P_2 \cdot (G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0) \\ &= \mathbf{G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_0} \end{aligned}$$

$$\boxed{C_{i+1} = G_i + P_i \cdot C_i}$$

Know the Values in Advance

$$C_1 = G_0 + P_0 \cdot C_0$$

$$\boxed{C_{i+1} = G_i + P_i \cdot C_i}$$

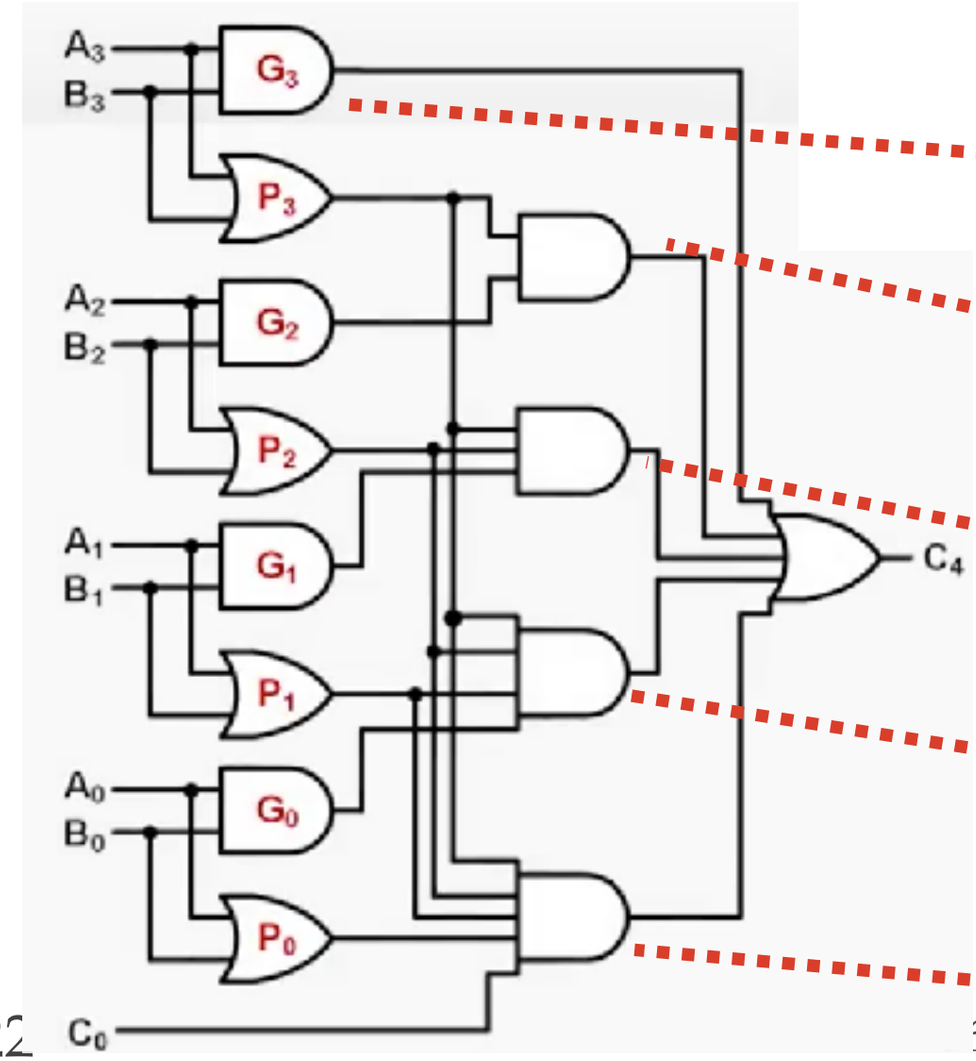
$$\begin{aligned} C_2 &= G_1 + P_1 \cdot C_1 \\ &= G_1 + P_1 \cdot (G_0 + P_0 \cdot C_0) \\ &= \mathbf{G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0} \end{aligned}$$

$$\begin{aligned} C_3 &= G_2 + P_2 \cdot C_2 \\ &= G_2 + P_2 \cdot (G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0) \\ &= \mathbf{G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_0} \end{aligned}$$

$$\begin{aligned} C_4 &= G_3 + P_3 \cdot C_3 \\ &= G_3 + P_3 \cdot (G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_0) \\ &= \mathbf{G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0} \end{aligned}$$

Detailed Analysis of C4

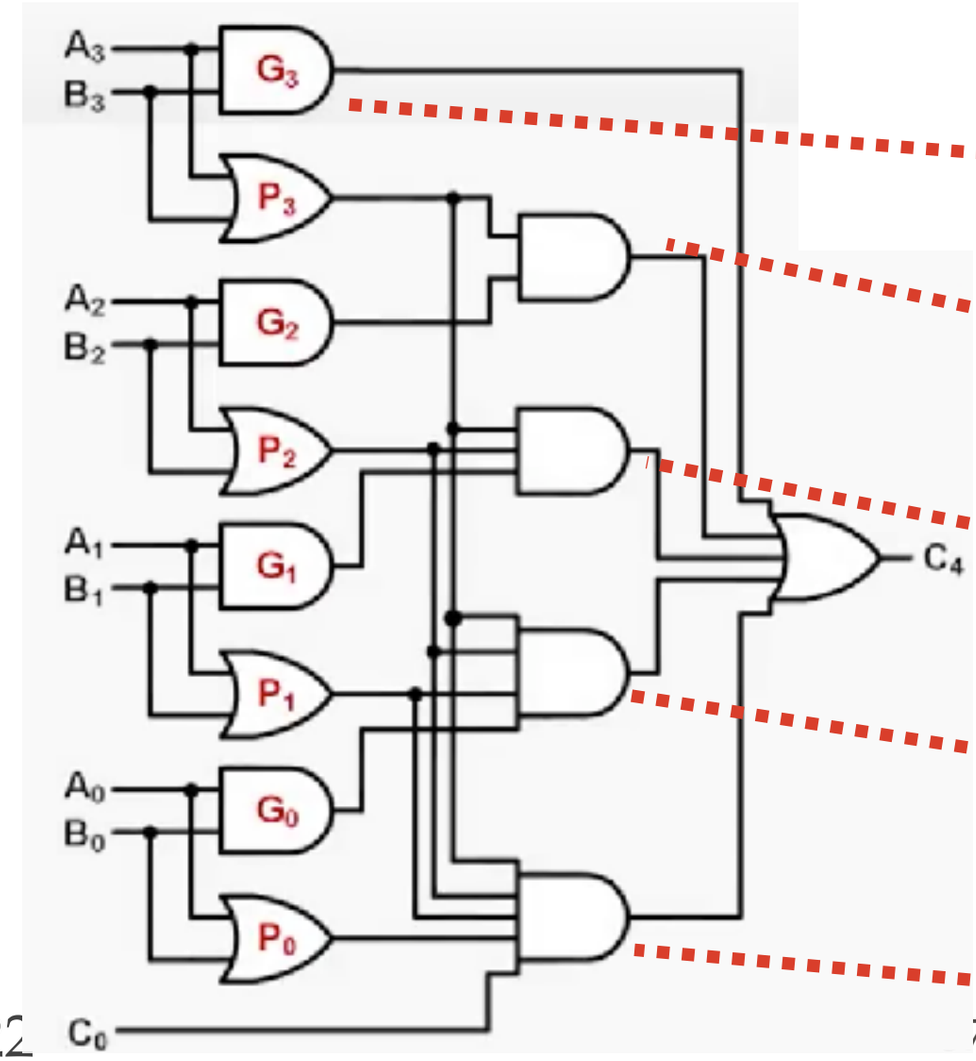
- Advantages?
  - The latency of Ci+1 is constant! Not dependent on N anymore



$$C_4 = G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0$$

Detailed Analysis of C4

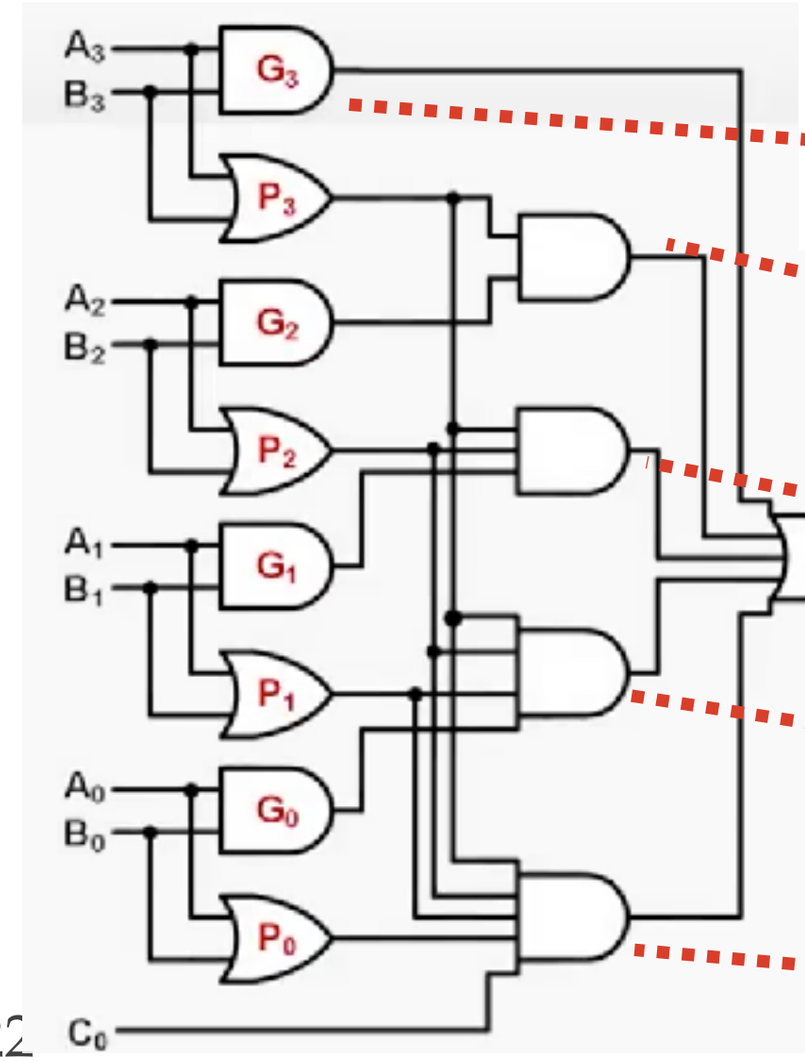
- Any disadvantages?



$$C_4 = G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0$$

Detailed Analysis of C4

- Any disadvantages?
  - With the adder becomes wider, the circuit will be very complicated!



$$C_4 = G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0$$

Further Optimization: Block of Adders

- Applying the concepts of "generate" and "propagate" to multiple-bit blocks/adders
  - A block ***generates*** a carry_out independent of the carry_in
  - A block ***propagates*** a carry_out if there is a carry_in

- Two new variables:
  - Gi:j and Pi:j

$$G_{3:0} = G_3 + P_3(G_2 + P_2(G_1 + P_1G_0))$$

$$P_{3:0} = P_3P_2P_1P_0$$

Further Optimization: Block of Adders

- Applying the concepts of "generate" and "propagate" to multiple-bit blocks/adders
  - A block ***generates*** a carry_out independent of the carry_in
  - A block ***propagates*** a carry_out if there is a carry_in

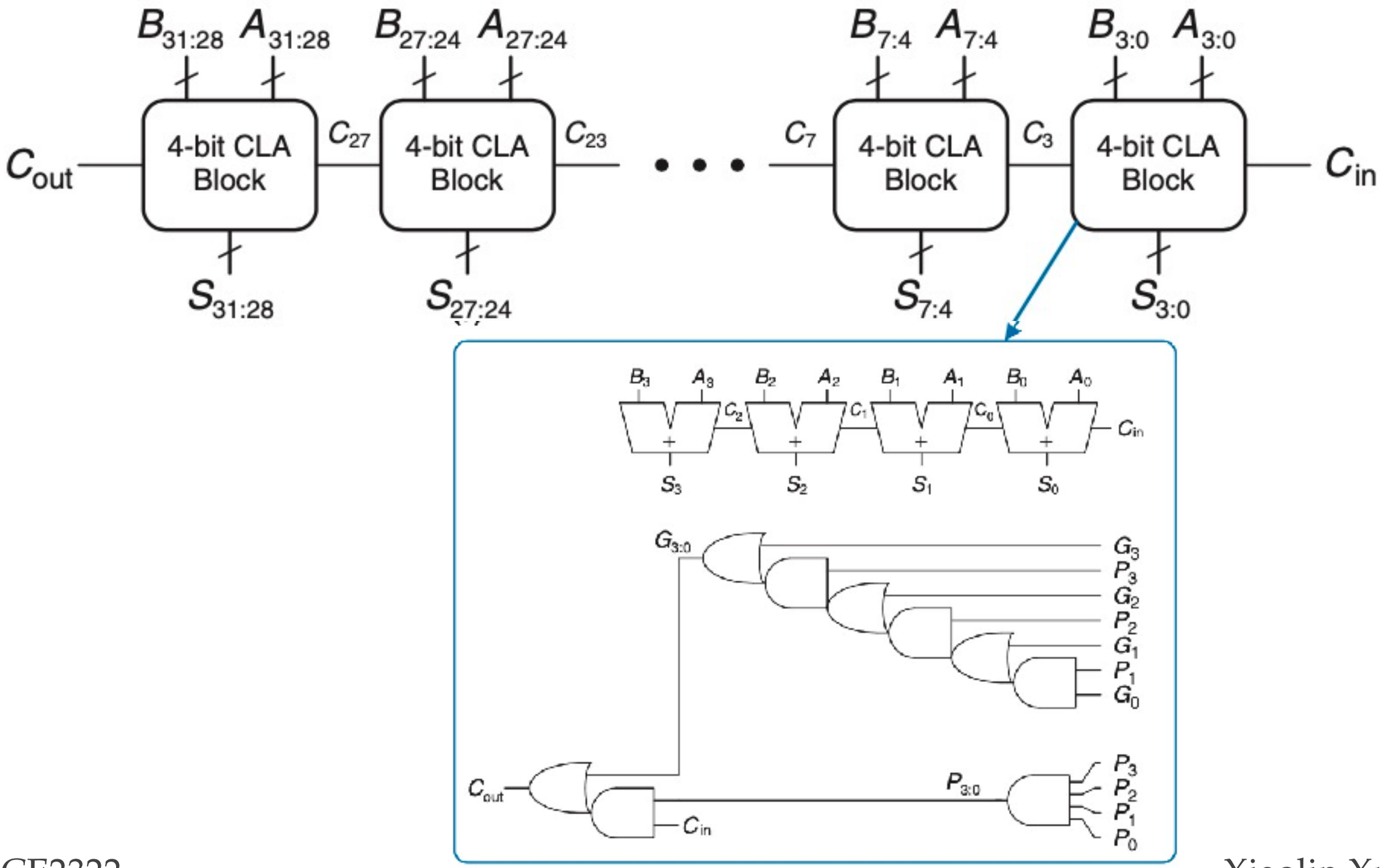
- Two new variables:
  - Gi:j and Pi:j

$$G_{3:0} = G_3 + P_3(G_2 + P_2(G_1 + P_1 G_0))$$

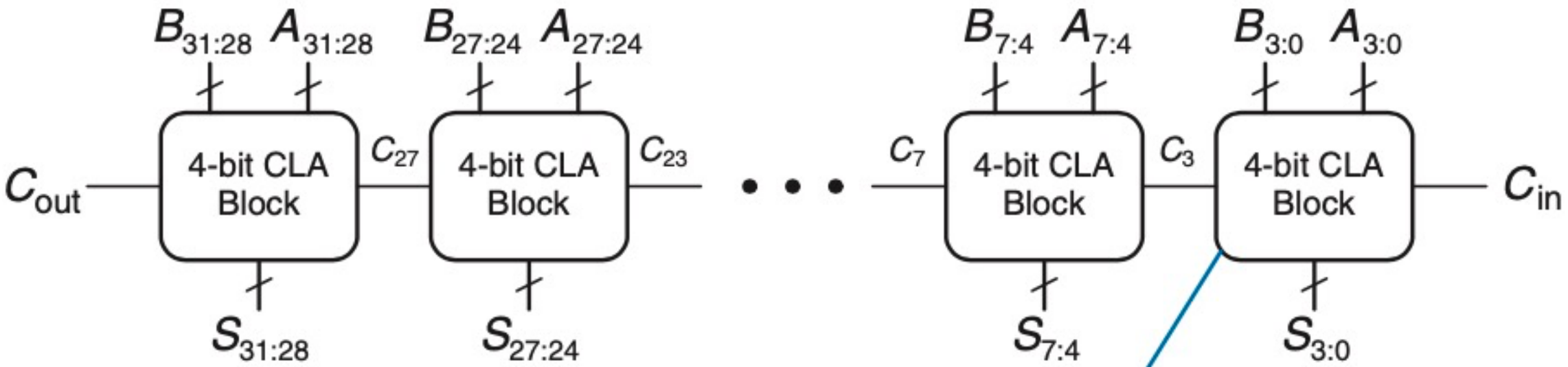
$$P_{3:0} = P_3 P_2 P_1 P_0$$

$$C_i = G_{i:j} + P_{i:j} C_j$$

32-bit CLA Example

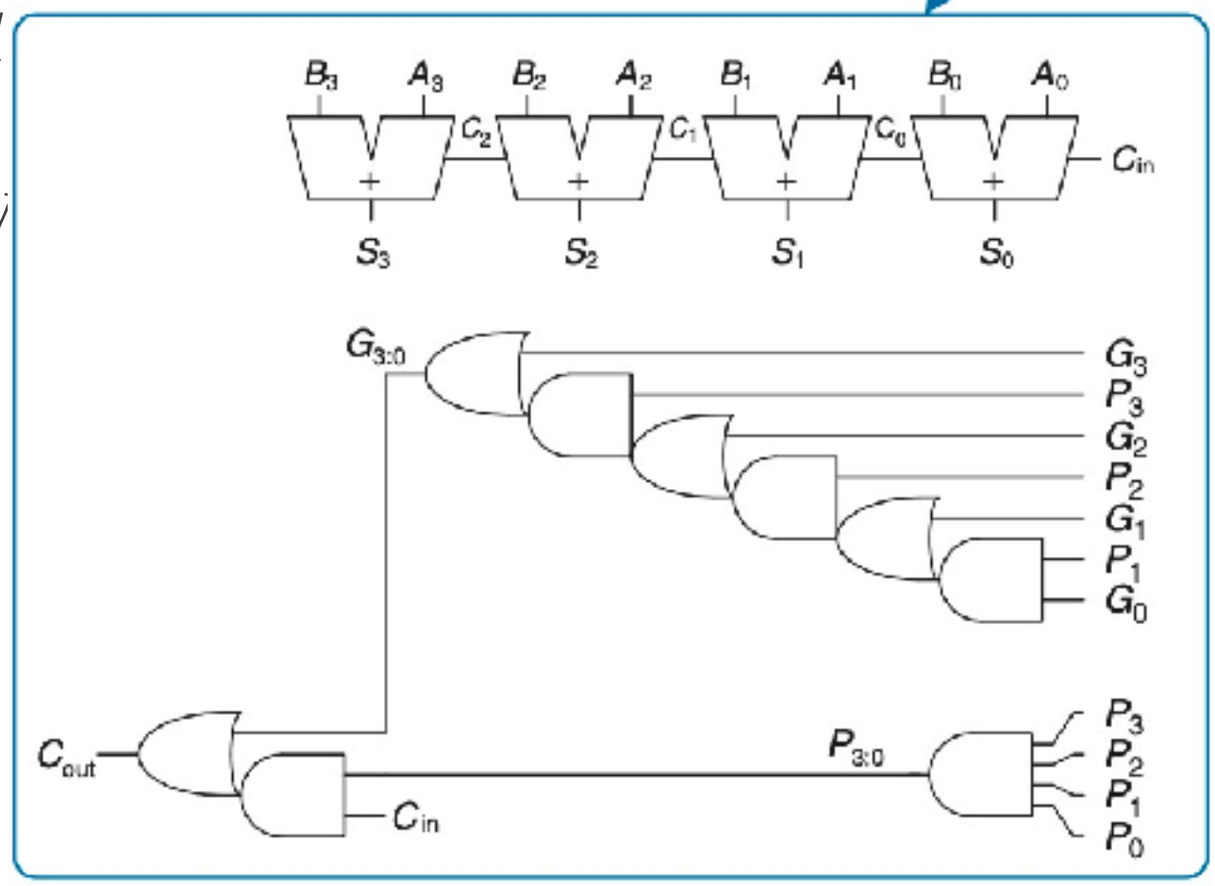


32-bit CLA Example

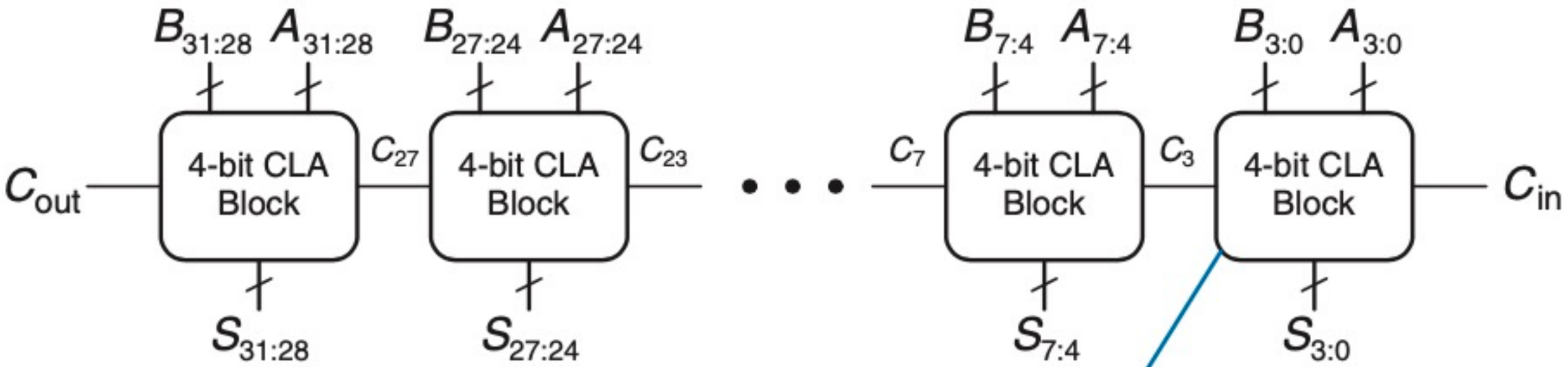


- generate/propagate *of a bl*

- t_{AND_OR} : Propagation delay

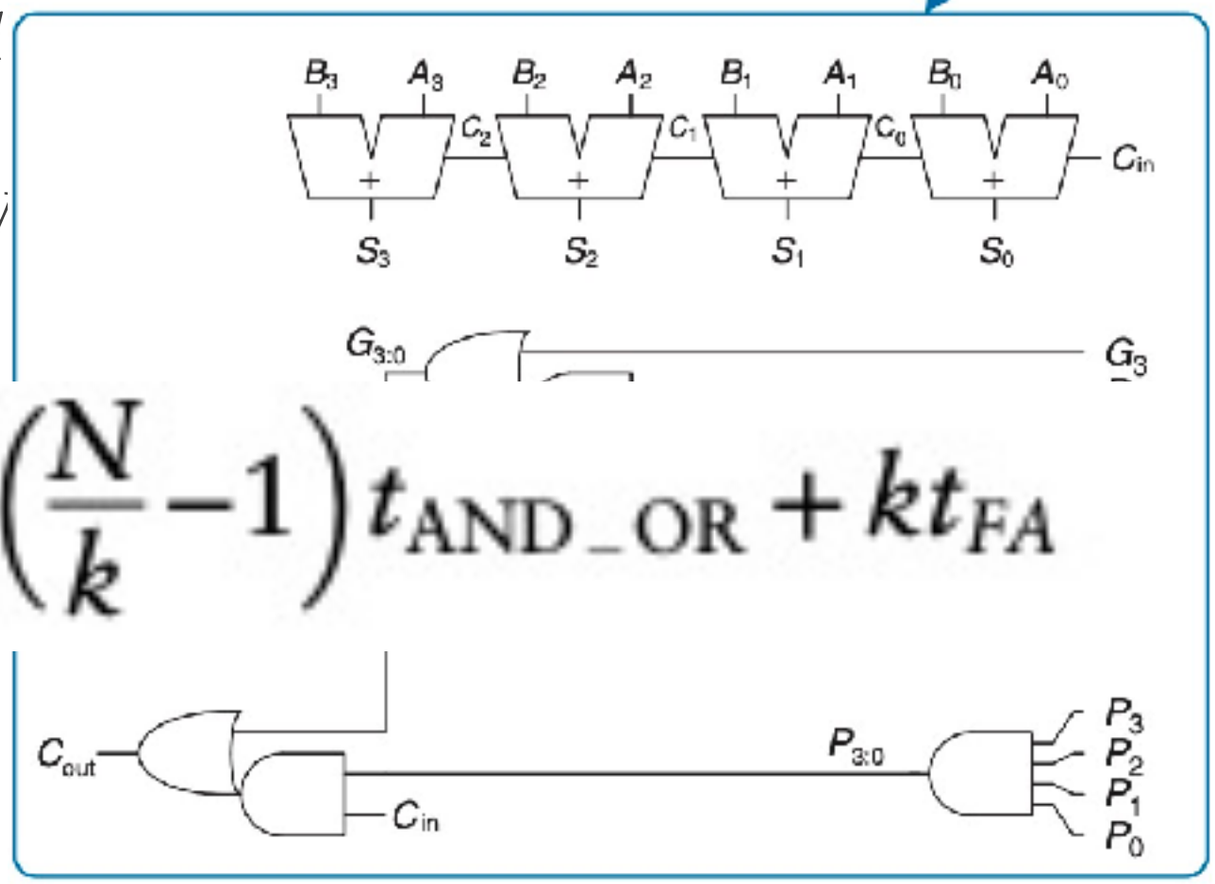


32-bit CLA Example



- generate/propagate *of a bl*

- t_{AND_OR} : Propagation delay



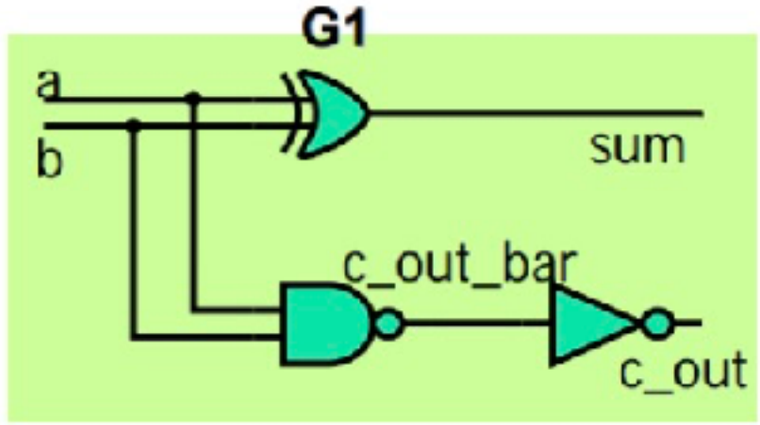
$$t_{CLA} = t_{pg} + t_{pg\_block} + \left(\frac{N}{k} - 1\right) t_{AND\_OR} + k t_{FA}$$

Mid-term Review

- Verilog
  - Write Verilog program for a given circuitry
  - Blocking and Non-blocking

Write the Code By Yourself

- A Multiplexer Using Basic Gates
  - Basic components?



Module name

Module ports

```
module Add_half ( input a, input b, output sum, output c_out );
wire c_out_bar;

    xor (sum, a, b);
    // xor G1(sum, a, b);
    nand (c_out_bar, a, b);
    not (c_out, c_out_bar);
endmodule
```

Declaration of internal signals

← the label G1 is optional for logic gate primitives

Instantiation of primitive gates

//Verilog keywords in blue

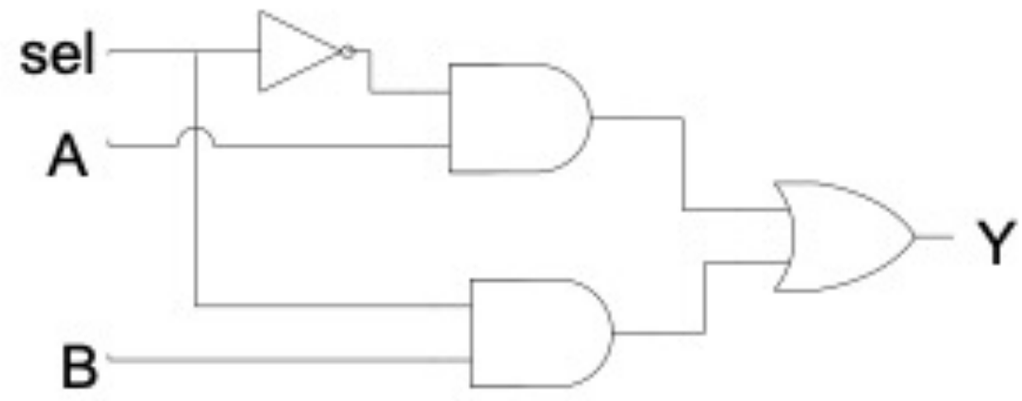
2-1 MUX

2-to-1 Multiplexer Behavior

when sel = 0 choose A to send to output Y
sel =1 choose B

$$Y = \overline{Sel} \bullet A + Sel \bullet B$$

sel	Y
0	A
1	B



Design Code of 2-1 MUX

- Structural Model: 2-1 mux
-

```
module mux2 (in0, in1, select, out);
    input in0,in1,select;
    output out;
    wire s0,w0,w1;

    not (s0, select);
    and (w0, s0, in0),
        (w1, select, in1);
    or  (out, w0, w1);

endmodule // mux2
```

Blocking and Non-blocking

- initial
- begin
- B <= A;
- C <= B;
- end

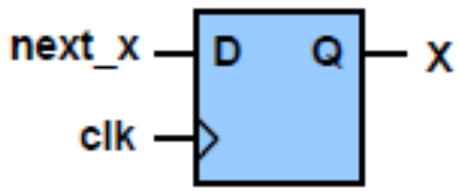
- Results?

- initial
- begin
- B = A;
- C = B;
- end

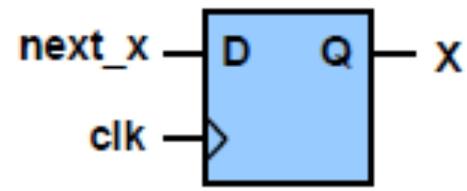
- Result?

Blocking and Nonblocking Statements

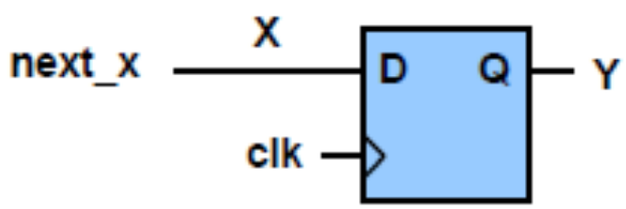
```
always @( posedge clk )
begin
  x = next_x;
end
```



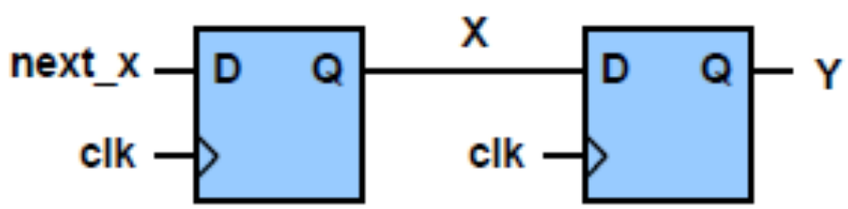

```
always @( posedge clk )
begin
  x <= next_x;
end
```




```
always @( posedge clk )
begin
  x = next_x;
  y = x;
end
```

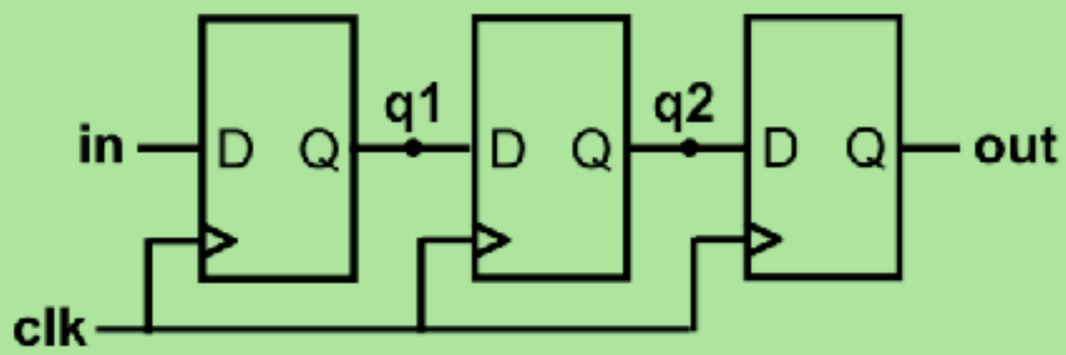



```
always @( posedge clk )
begin
  x <= next_x;
  y <= x;
end
```



Blocking and Nonblocking Statements

Flip-Flop Based Digital Delay Line



- **Will nonblocking and blocking assignments both produce the desired result?**

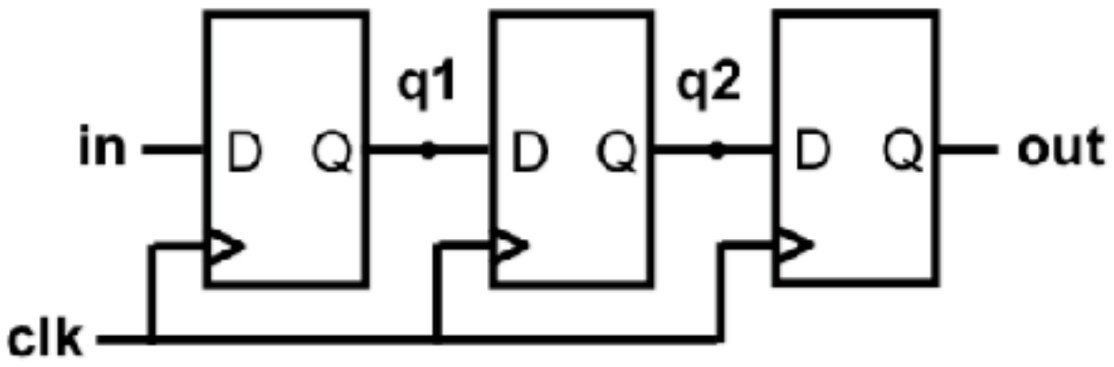
```
module nonblocking(in, clk, out);
  input in, clk;
  output out;
  reg q1, q2, out;

  always @ (posedge clk)
  begin
    q1 <= in;
    q2 <= q1;
    out <= q2;
  end

endmodule
```

```
module blocking(in, clk, out);
  input in, clk;
  output out;
  reg q1, q2, out;

  always @ (posedge clk)
  begin
    q1 = in;
    q2 = q1;
    out = q2;
  end

endmodule
```

Blocking and Nonblocking Statements

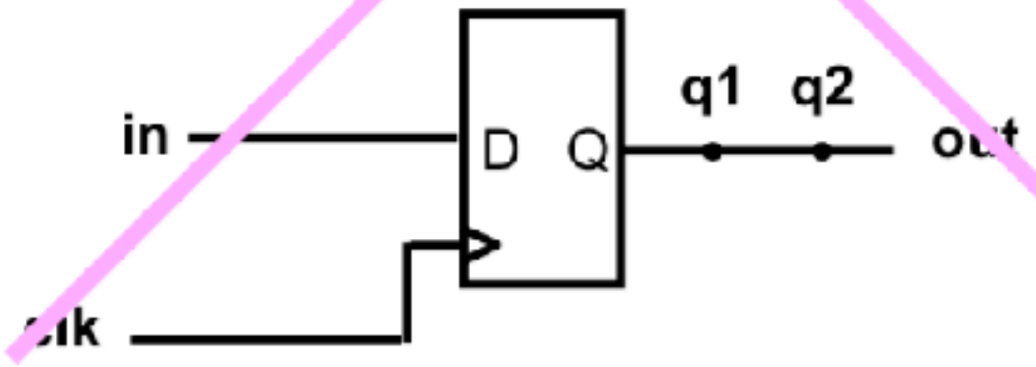
```
always @ (posedge clk)
begin
   q1 <= in;
   q2 <= q1;
   out <= q2;
end
```

"At each rising clock edge, *q1*, *q2*, and *out* simultaneously receive the old values of *in*, *q1*, and *q2*."




```
always @ (posedge clk)
begin
   q1 = in;
   q2 = q1;
   out = q2;
end
```

"At each rising clock edge, *q1* = *in*. After that, *q2* = *q1* = *in*; After that, *out* = *q2* = *q1* = *in*; Finally *out* = *in*."



Blocking and Non-blocking

- initial
- begin
- B <= A;
- C <= B;
- end

- Results?

- initial
- begin
- B = A;
- C = B;
- end

- Result?