SWE223: Digital Electronics
Fall 2015

Lecture 4
Tanjila Farah (TF)

25/10/2015

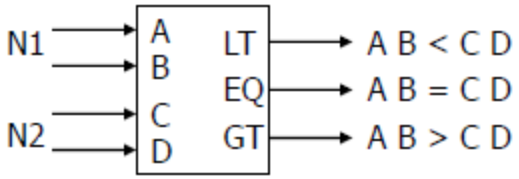
Textbooks

- M. Moris Mano “Digital Logic and Computer Design”, Prentice Hall.

Overview

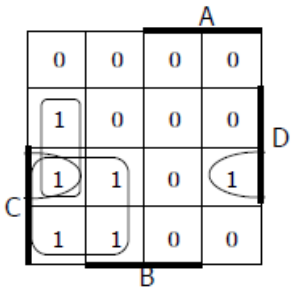
- Comparator

Design example: two-bit comparator

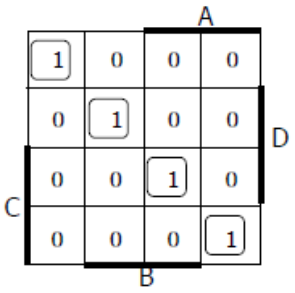


block diagram and truth table

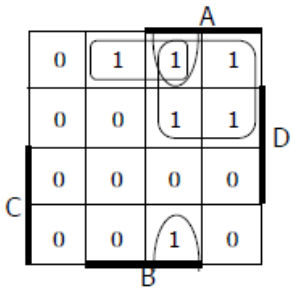
A	B	C	D	LT	EQ	GT
0	0	0	0	0	1	0
		0	1	1	0	0
		1	0	1	0	0
		1	1	1	0	0
0	1	0	0	0	0	1
		0	1	0	1	0
		1	0	1	0	0
		1	1	1	0	0
1	0	0	0	0	0	1
		0	1	0	0	1
		1	0	0	1	0
		1	1	1	0	0
1	1	0	0	0	0	1
		0	1	0	0	1
		1	0	0	0	1
		1	1	0	1	0



K-map for LT



K-map for EQ



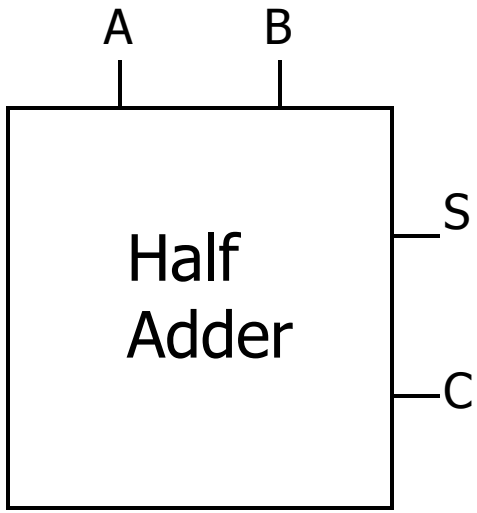
K-map for GT

LT = A' B' D + A' C + B' C D

EQ = A' B' C' D' + A' B C' D + A B C D + A B' C D'= (A xnor C) • (B xnor D)

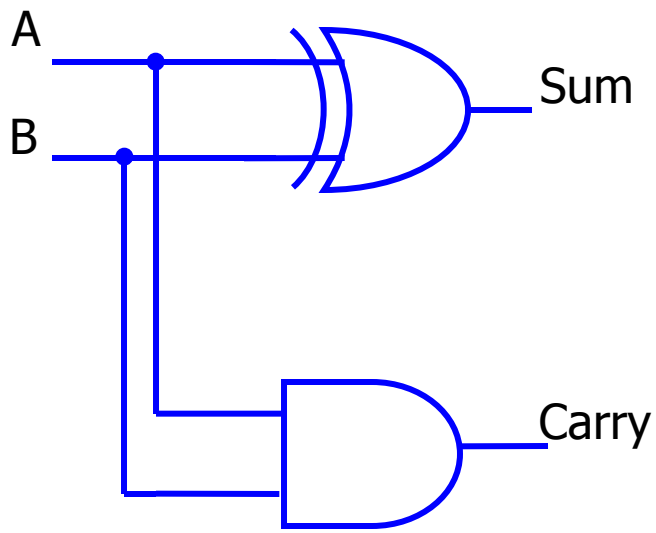
GT = B C' D' + A C' + A B D'

Half Adder (1-bit)



A	B	S(um)	C(arry)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Half Adder (1-bit)

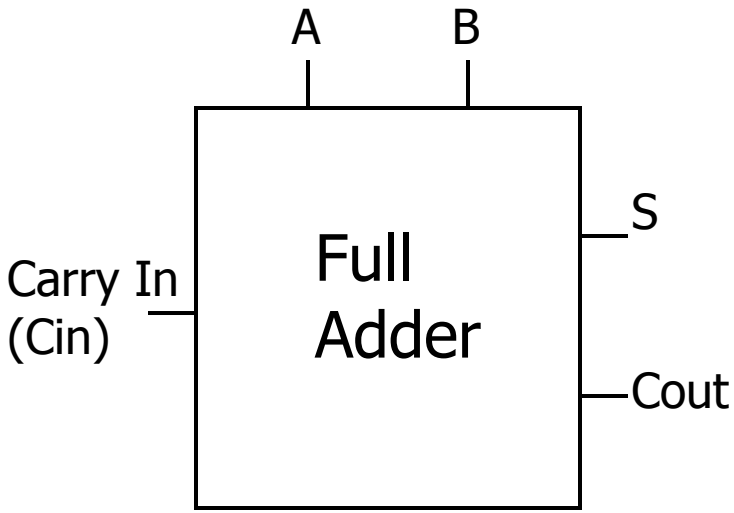


A	B	S(um)	C(arry)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$S = \overline{A}B + A\overline{B} = A \oplus B$$

$$C = AB$$

Full Adder



Cin	A	B	S(um)	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Full Adder

Cin	A	B	S(um)	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Cin \ AB	00	01	11	10
0	0	1	0	1
1	1	0	1	0

$$S = Cin\,\overline{A}\,\overline{B} + \overline{Cin}\,\overline{A}B + CinAB + \overline{Cin}A\overline{B}$$

$$= Cin(\overline{A}\,\overline{B} + AB) + \overline{Cin}(\overline{A}B + A\overline{B})$$

$$= Cin(\overline{A \oplus B}) + \overline{Cin}(A \oplus B)$$

$$= Cin \oplus A \oplus B$$

Cin \ AB	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$$Cout = CinB + CinA + AB$$

Or

Cin \ AB	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$$Cout = AB + Cin(\overline{A}B + A\overline{B}) = AB + Cin(A \oplus B)$$

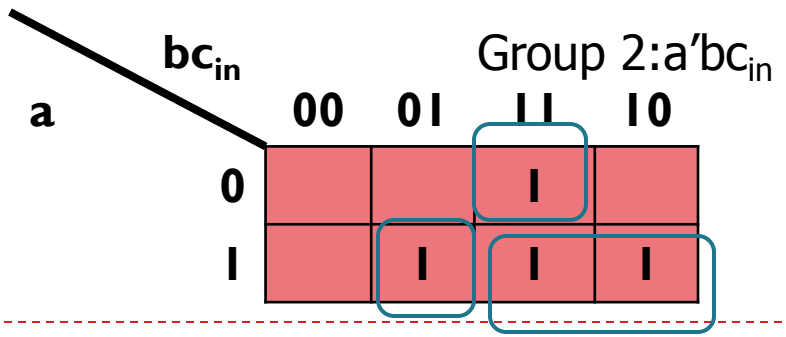
Full adder alternative circuit

Constructing a Full Adder using Half Adders (cont.)

- Sum Boolean expression of a full adder:

$$sum = a'b'c_{in} + a'bc_{in}' + ab'c_{in}' + abc_{in}$$
$$sum = c_{in}(a'b' + ab) + c_{in}' (a'b + ab')$$
$$sum = c_{in}(a \oplus b)' + c_{in}' (a \oplus b)$$
$$sum = (a \oplus b) \oplus c$$

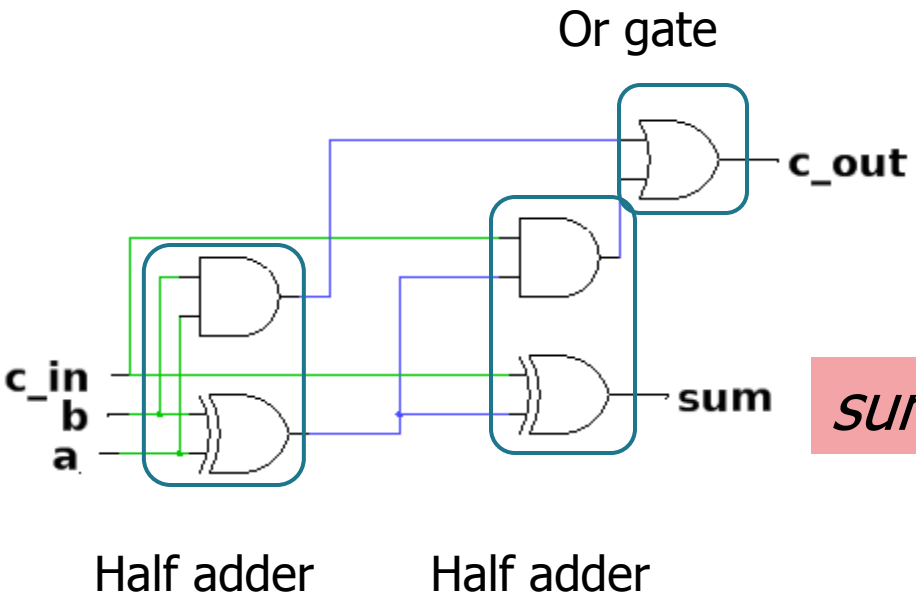
- C_{out} Boolean expression of a full adder:



$$c_{out} = a'bc_{in} + ab'c_{in} + abc_{in}' + abc_{in}$$
$$c_{out} = ab + a'bc_{in} + ab'c_{in}$$
$$c_{out} = ab + c_{in}(a'b + ab')$$
$$c_{out} = ab + c_{in}(a \oplus b)$$

Constructing a Full Adder using Half Adders (cont.)

- Circuit diagram of a full adder (which is made up of 2 half adders and 1 OR gate):



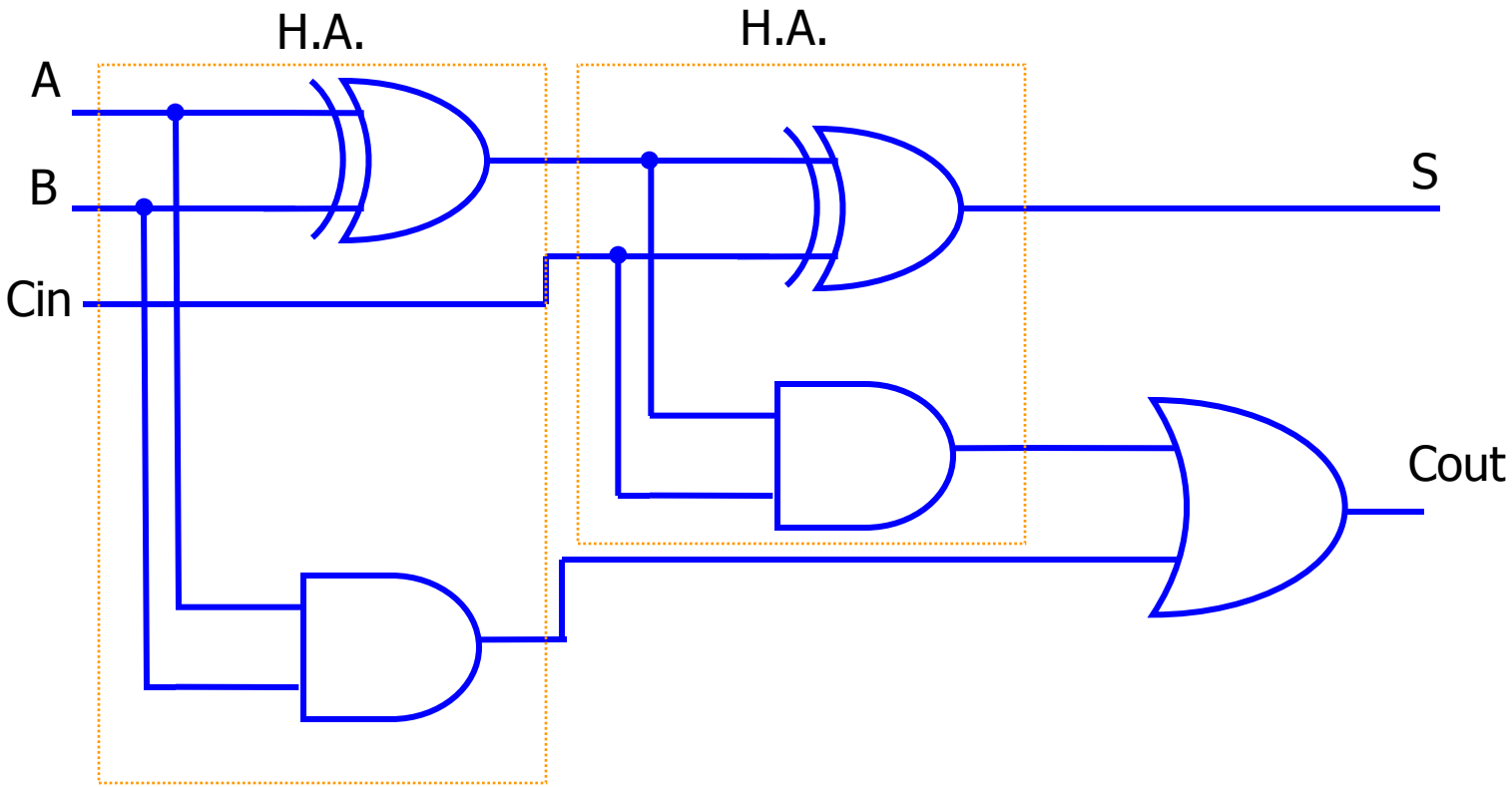
$c_{out} = ab + c_{in}(a \oplus b)$

$sum = (a \oplus b) \oplus c$

Full Adder

$$S = Cin \oplus A \oplus B$$

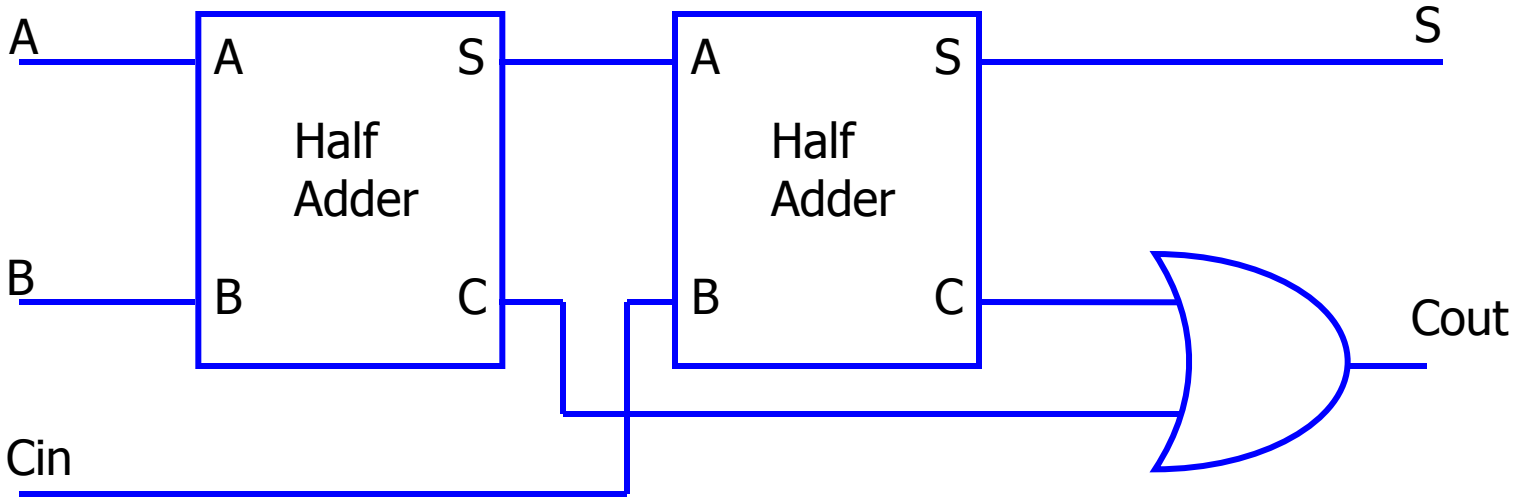
$$Cout = AB + Cin(A \oplus B)$$



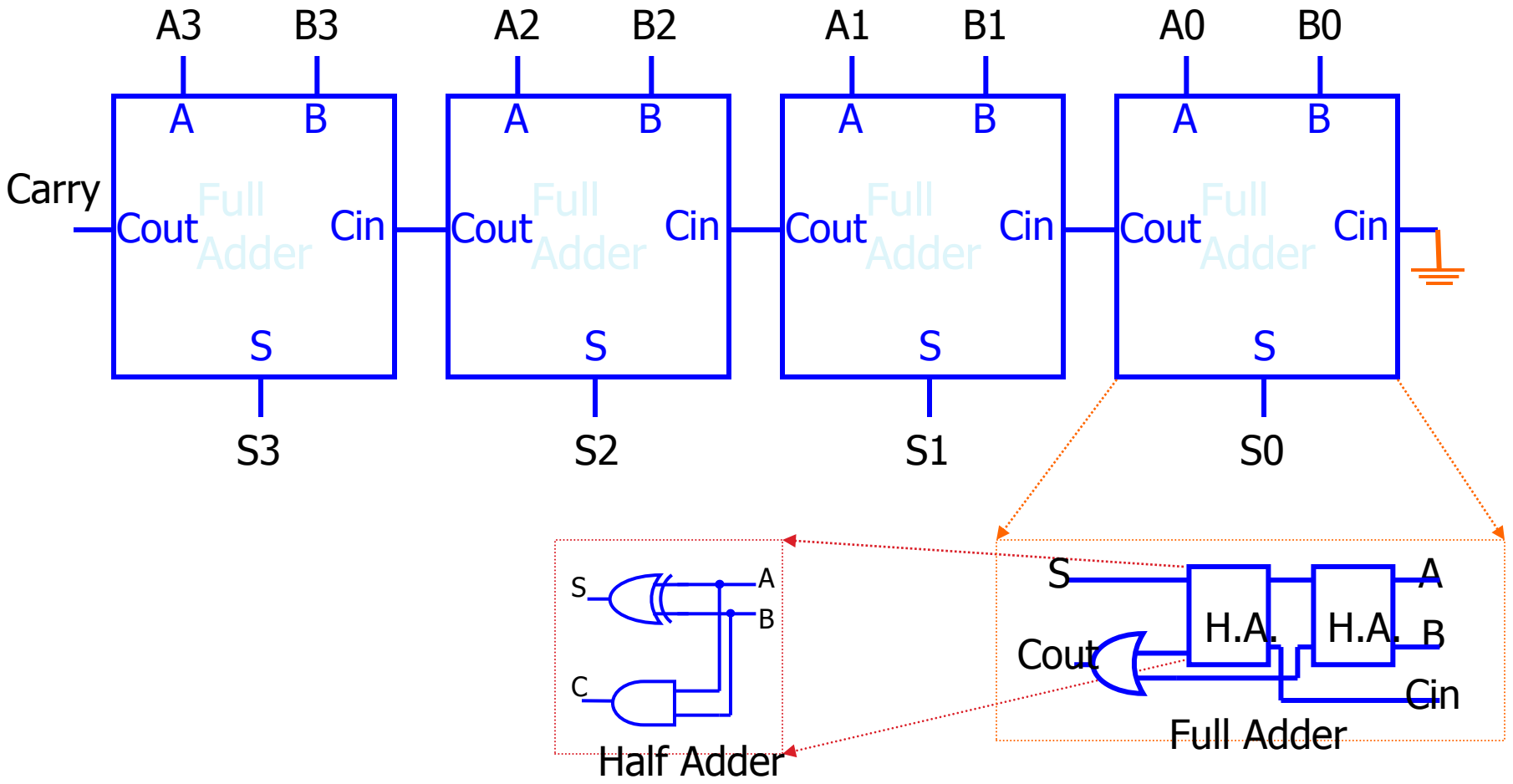
Full Adder

$$S = Cin \oplus A \oplus B$$

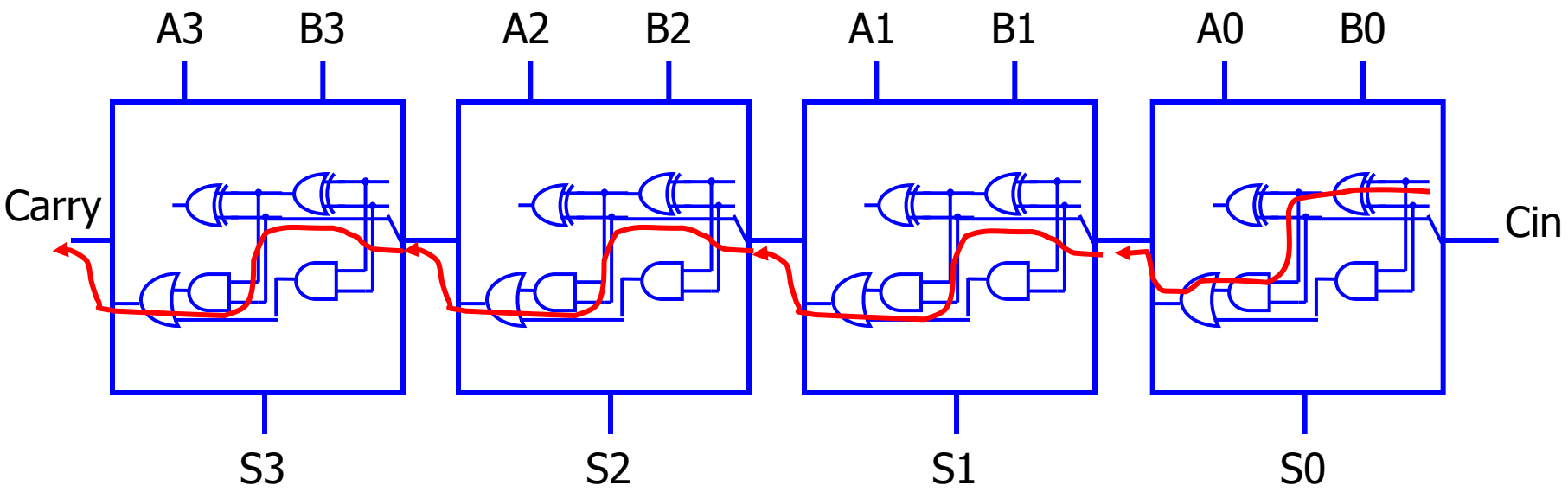
$$Cout = AB + Cin(A \oplus B)$$



4-bit Ripple Adder using Full Adder



Issue of 4-bit Ripple Adder



Issue of Ripple Adder

- Carry propagation is the main issue in an N-bit ripple adder
- A faster adder needs to address the serial propagation of the carry bit
- Let's re-examine the equation for full adders

Carry Generate & Propagate

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

$$g_i = A_i B_i \qquad \text{(generate)}$$

$$p_i = A_i + B_i \qquad \text{(propagate)}$$

$$C_{i+1} = g_i + p_i C_i$$

$$C_1 = g_0 + p_0 C_0$$

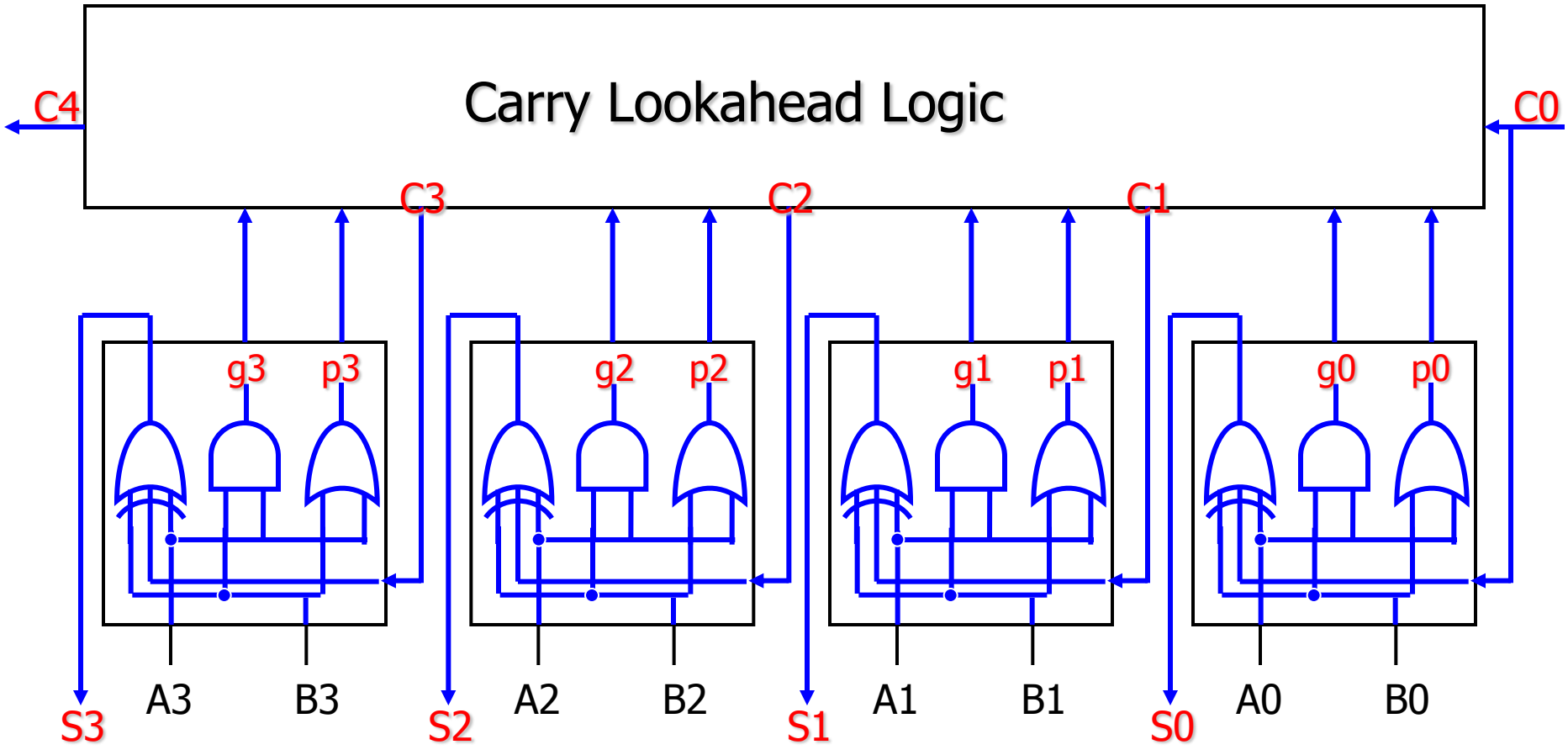
$$C_2 = g_1 + p_1 C_1 = g_1 + p_1 g_0 + p_1 p_0 C_0$$

$$C_3 = g_2 + p_2 C_2 = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 C_0$$

$$C_4 = g_3 + p_3 C_3 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 C_0$$

Note that all the carry's are only dependent on input A and B and C

4-bit Carry-Lookahead Adder (CLA)

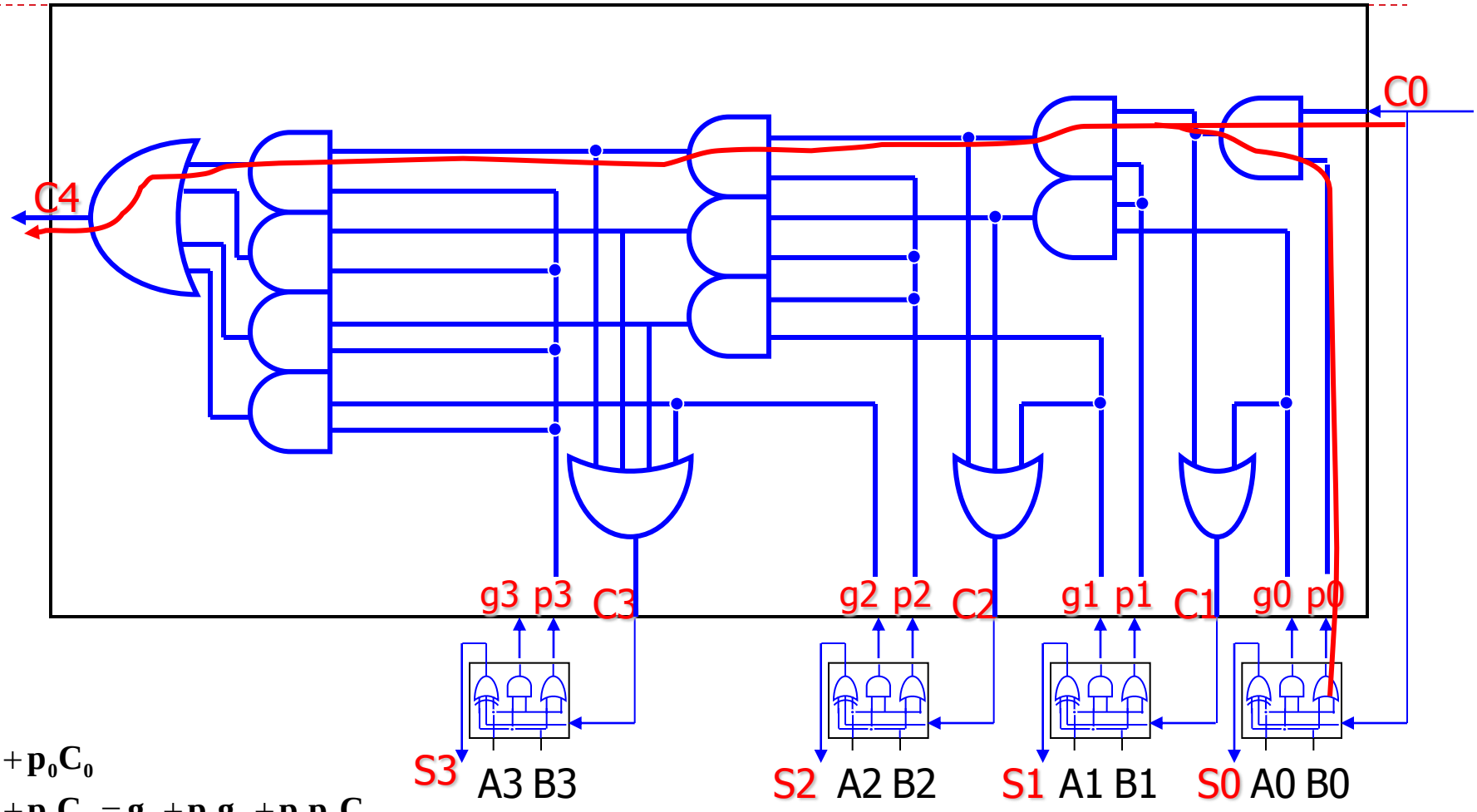


$$S_i = C_i \oplus A_i \oplus B_i$$

$$g_i = A_i B_i \quad \text{(generate)}$$

$$p_i = A_i + B_i \quad \text{(propagate)}$$

Inefficient Implementation of Carry Lookahead Logic



$C_1 = g_0 + p_0 C_0$

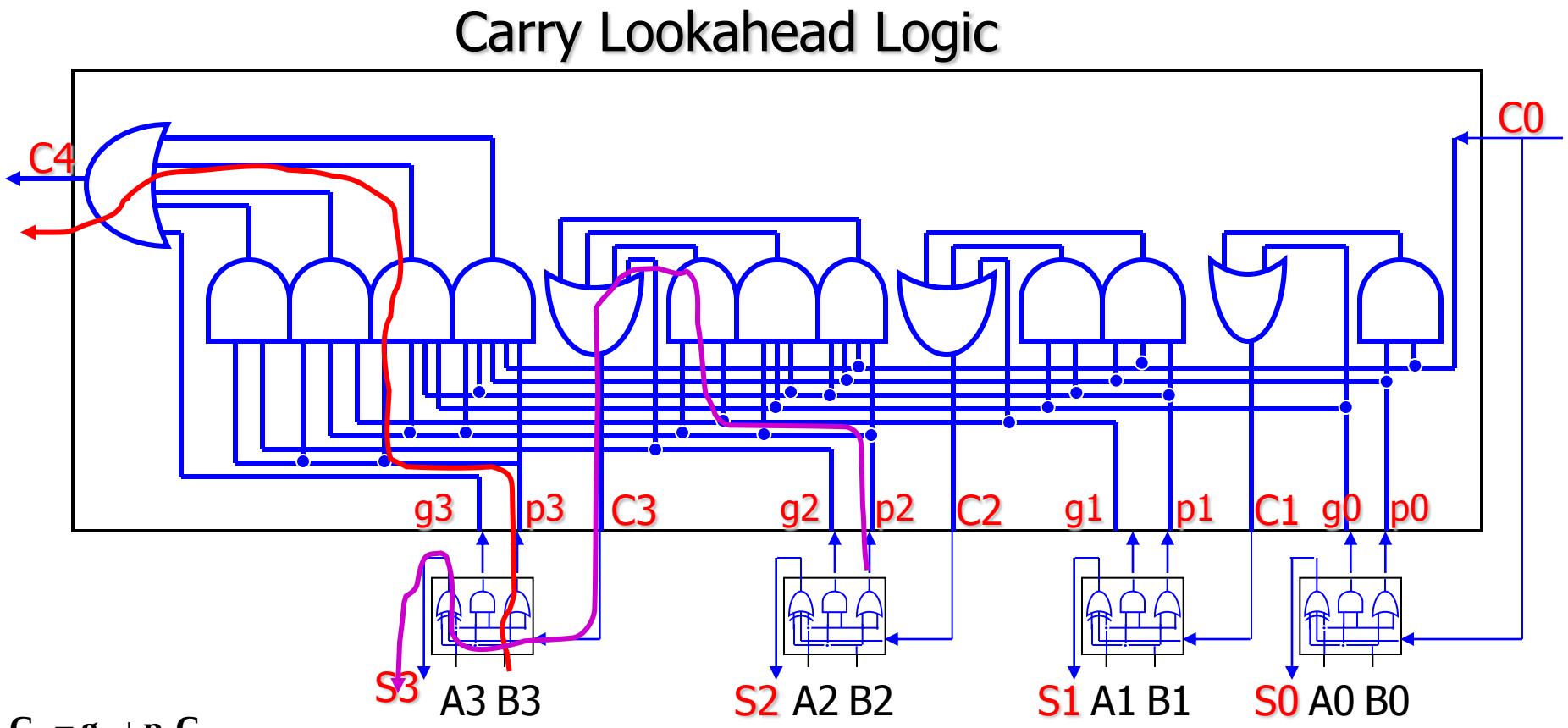
$C_2 = g_1 + p_1 C_1 = g_1 + p_1 g_0 + p_1 p_0 C_0$

$C_3 = g_2$

$C_4 = g_3$

Reuse some gate output results →
Little Improvement
Carry Delay is $4*D_{AND} + 2*D_{OR}$ for Carry C_4

Implementation of Carry Lookahead Logic



$$C_1 = g_0 + p_0 C_0$$

$$C_2 = g_1 + p_1 C_1 = g_1 + p_1 g_0 + p_1 p_0 C_0$$

$$C_3 = g_2 + p_2 C_2 = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 C_0$$

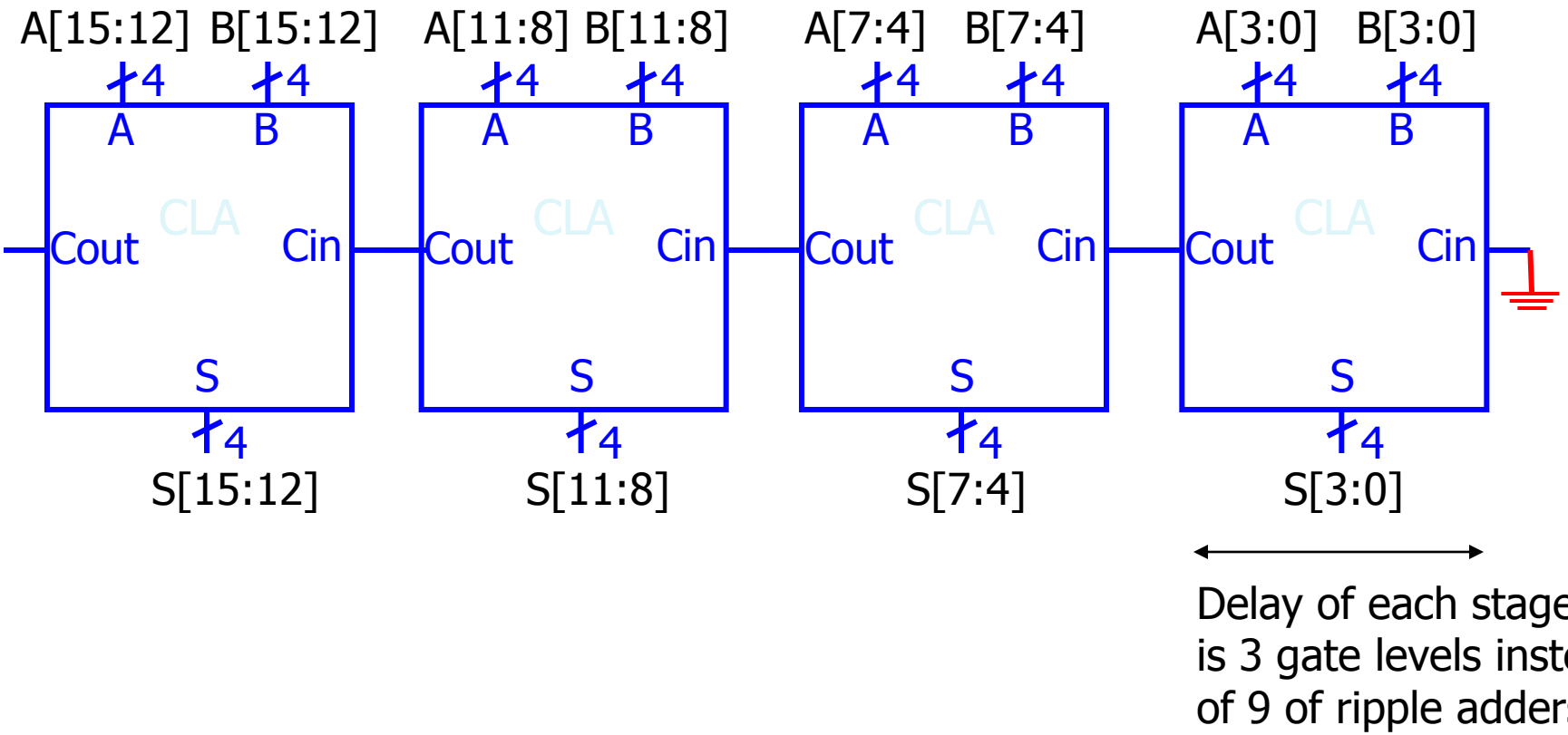
$$C_4 = g_3 + p_3 C_3 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 C_0$$

Only 3 Gate Delay for each Carry C_i
$= D_{AND} + 2*D_{OR}$

4 Gate Delay for each Sum S_i
$= D_{AND} + 2*D_{OR} + D_{XOR}$

Cascading CLA

- Similar to ripple adder, but different latency





Unsigned Binary Multiply

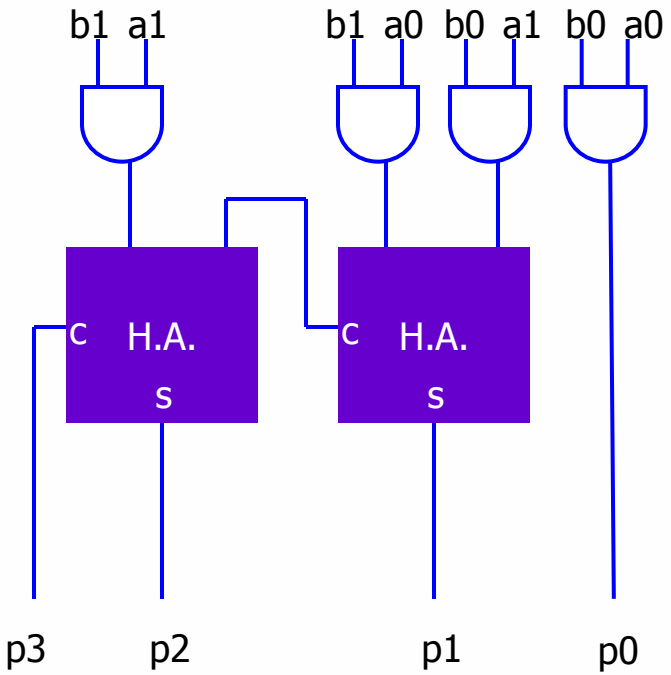
```
    101  (5)
x   111  (7)
-----------
    101
   101
  101
-----------
100011 (35)
```

Unsigned Integer Multiplier (2-bit)

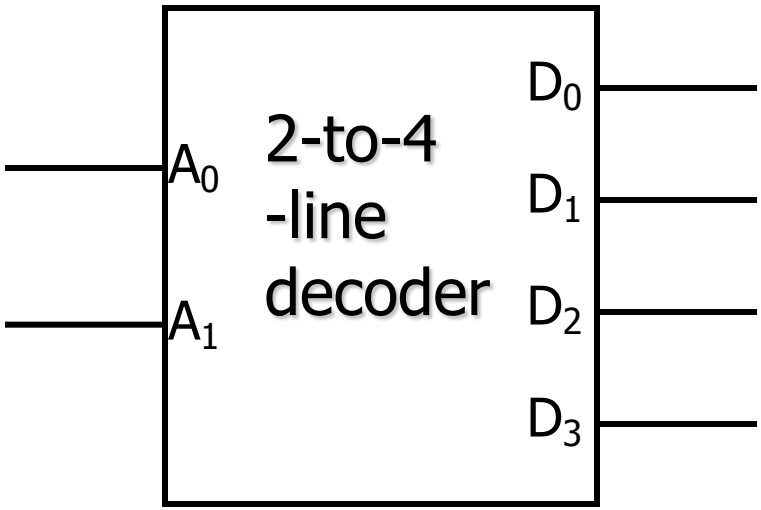
2-bit by 2-bit

	a_1	a_0
x	b_1	b_0
	a_1b_0	a_0b_0
a_1b_1	a_0b_1	
a_1b_1	$a_1b_0 + a_0b_1$	a_0b_0

carry out carry

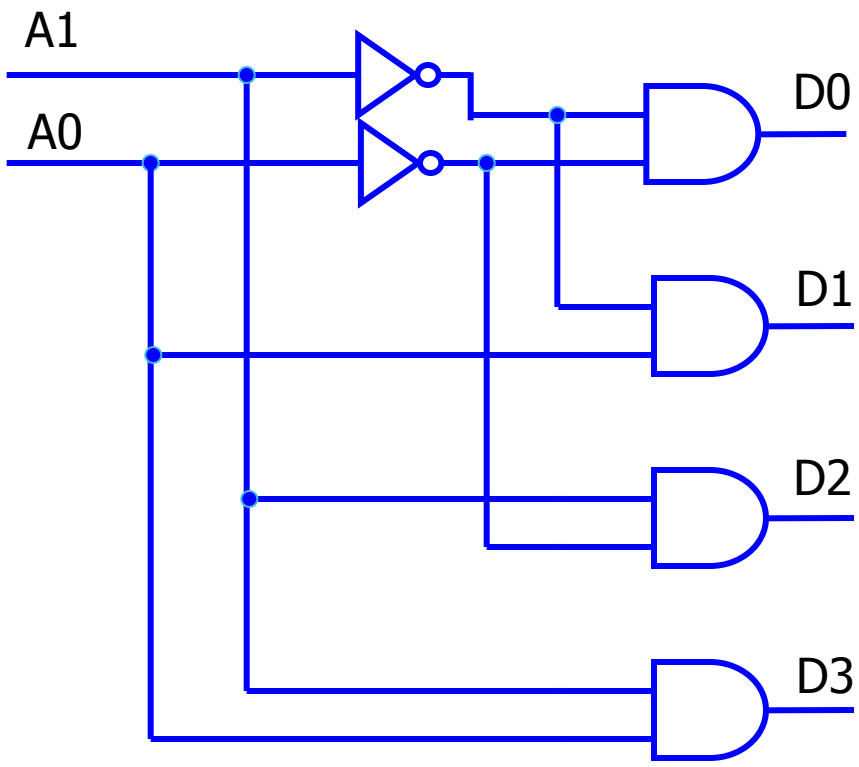


N-to-M-Line Decoder ($2^N \geq M$)



A1	A0	D3	D2	D1	D0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

2-to-4-Line Decoder



A1	A0	D3	D2	D1	D0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

$$D_0 = \overline{A_1}\,\overline{A_0}$$

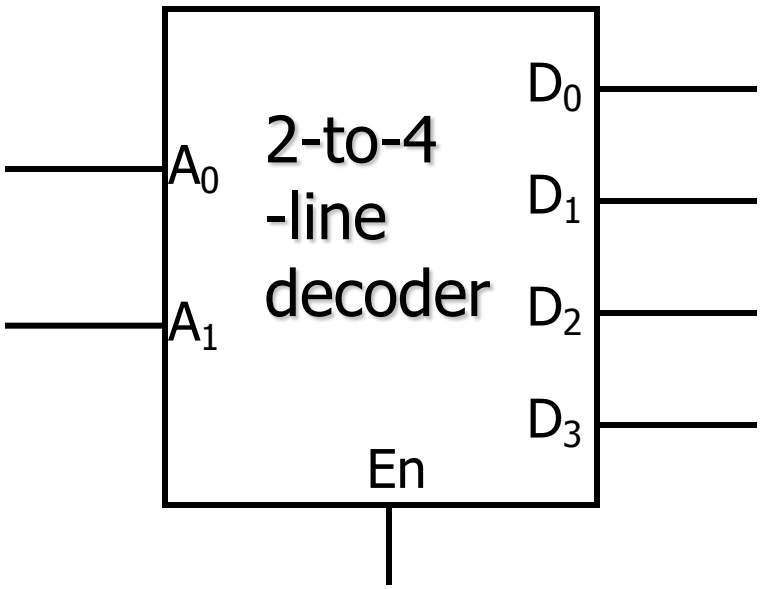
$$D_1 = \overline{A_1} A_0$$

$$D_2 = A_1 \overline{A_0}$$

$$D_3 = A_1 A_0$$

How about if no one should be enabled ?

2-to-4-Line Decoder w/ Enable



En	A1	A0	D 3	D 2	D 1	D 0
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

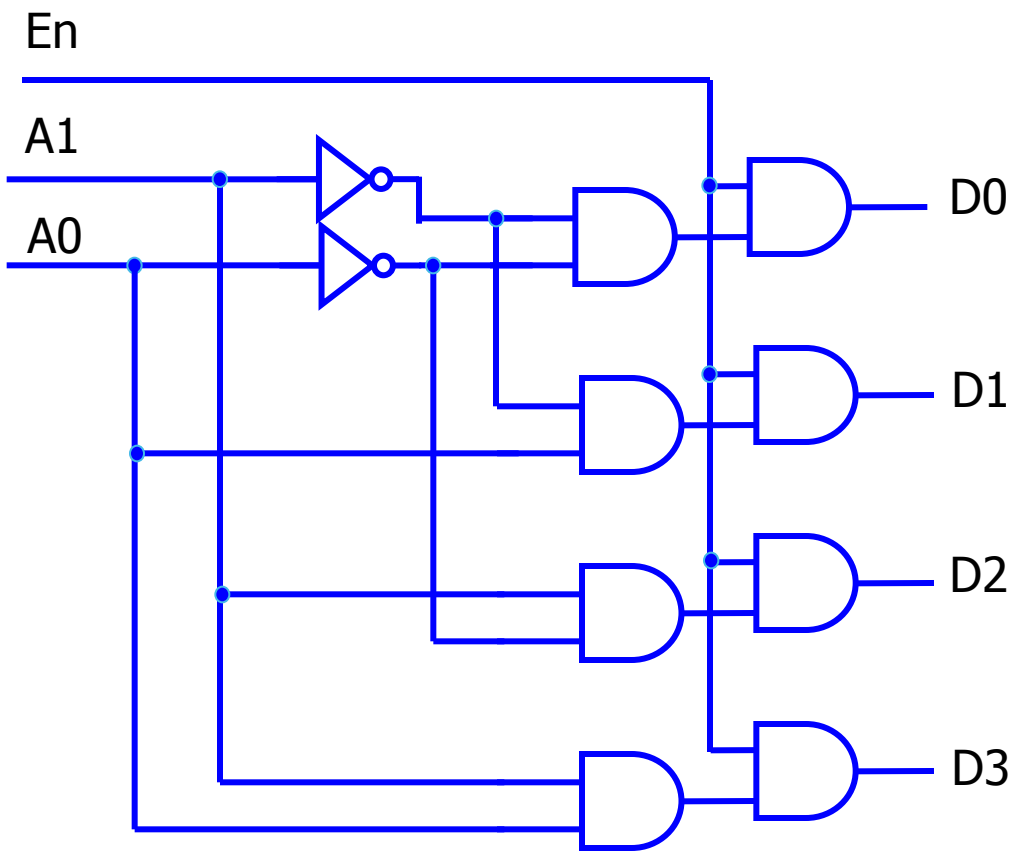
$$D_0 = En\overline{A_1}\,\overline{A_0}$$

$$D_1 = En\overline{A_1}A_0$$

$$D_2 = EnA_1\overline{A_0}$$

$$D_3 = EnA_1A_0$$

2-to-4-Line Decoder w/ Enable



En	A1	A0	D3	D2	D1	D0
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

$$D_0 = En\overline{A_1}\,\overline{A_0}$$

$$D_1 = En\overline{A_1}A_0$$

$$D_2 = EnA_1\overline{A_0}$$

$$D_3 = EnA_1A_0$$

3-to-8-Line Decoder

Truth Table

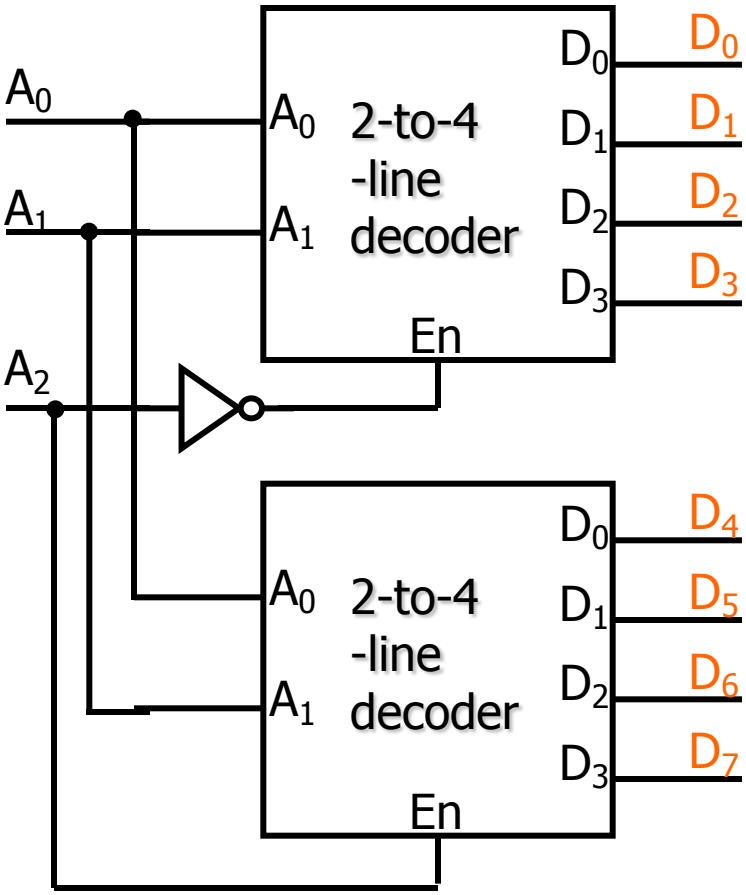
A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

3-to-8-Line Decoder

Truth Table

A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

3-to-8-Line Decoder

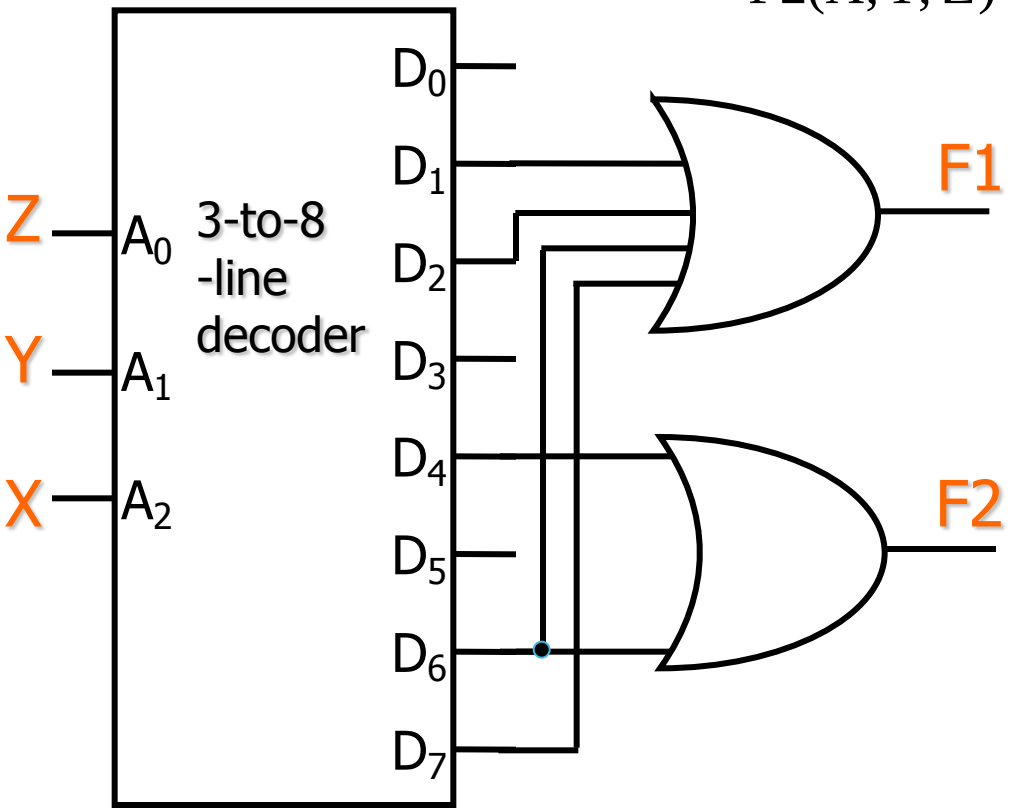


A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

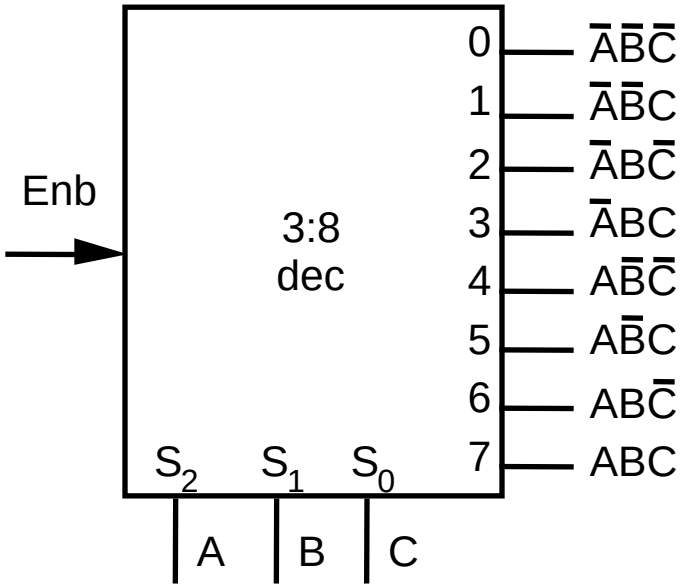
Implementing Logic w/ Decoder

$$F1(X, Y, Z) = \sum m(1,2,6,7)$$

$$F2(X, Y, Z) = \prod M(0,1,2,3,5,7)$$



Decoder as a Logic Building Block

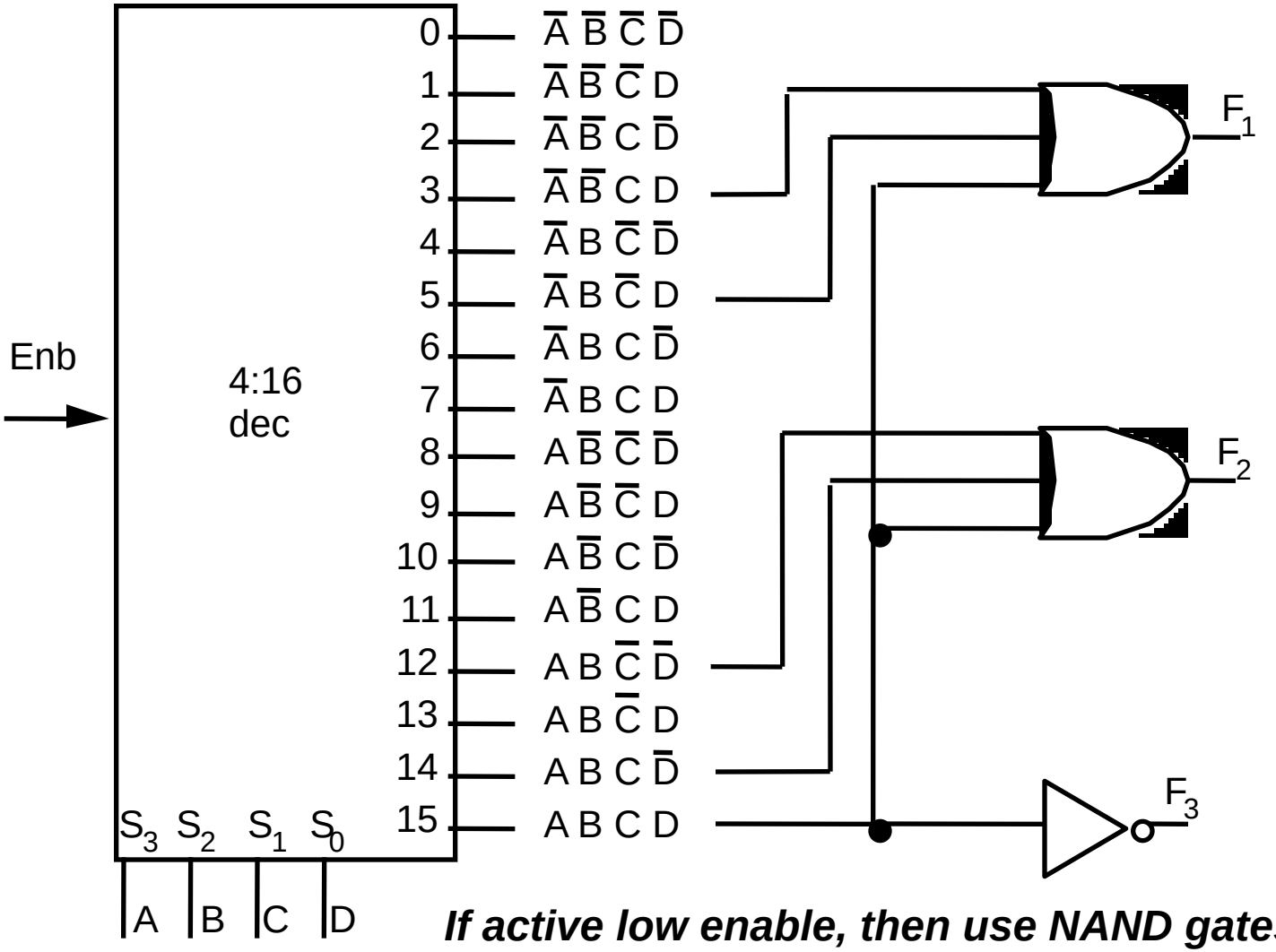


Decoder Generates Appropriate Minterm based on Control Signals

Example Function:

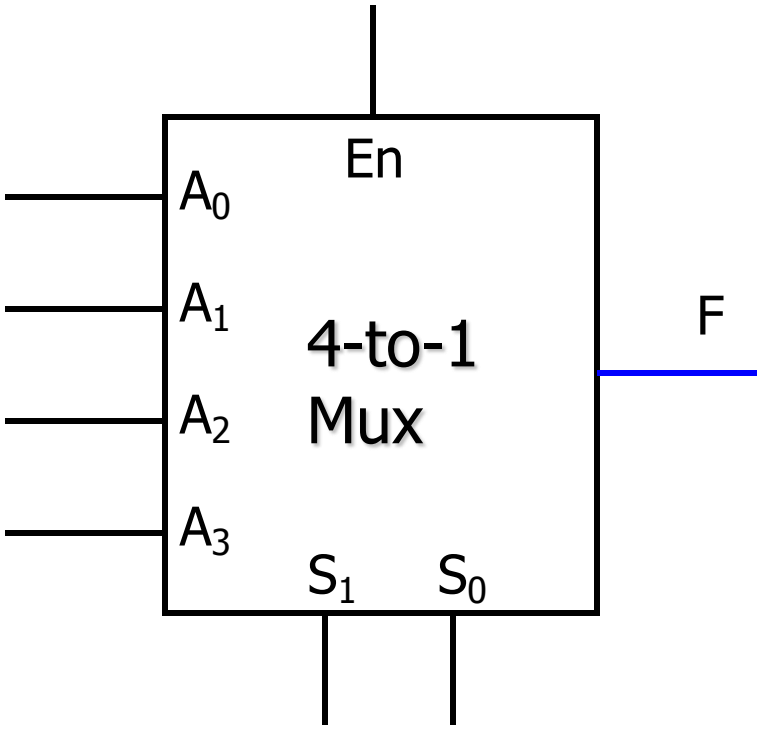
F1 = A' B C' D + A' B' C D + A B C D
F2 = A B C' D' + A B C
F3 = (A' + B' + C' + D')

Decoder as a Logic Building Block



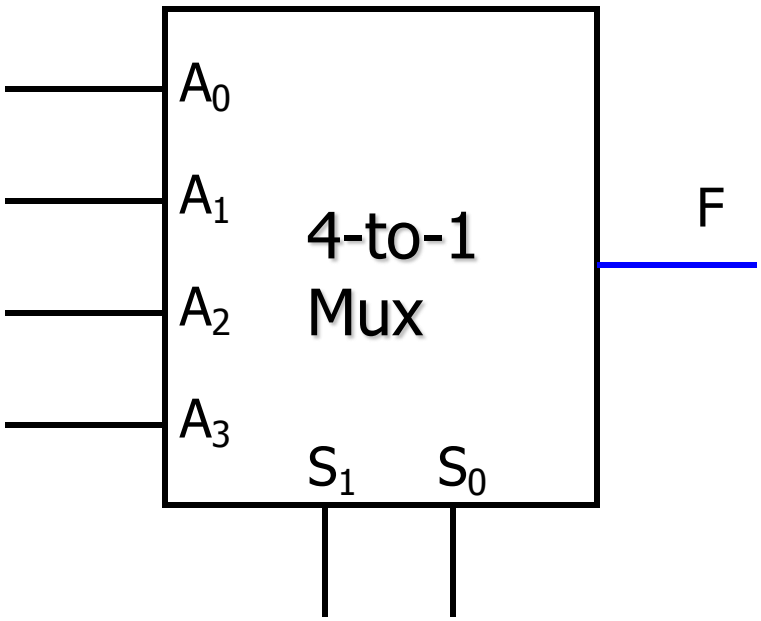
If active low enable, then use NAND gates!

Multiplexers (Mux)



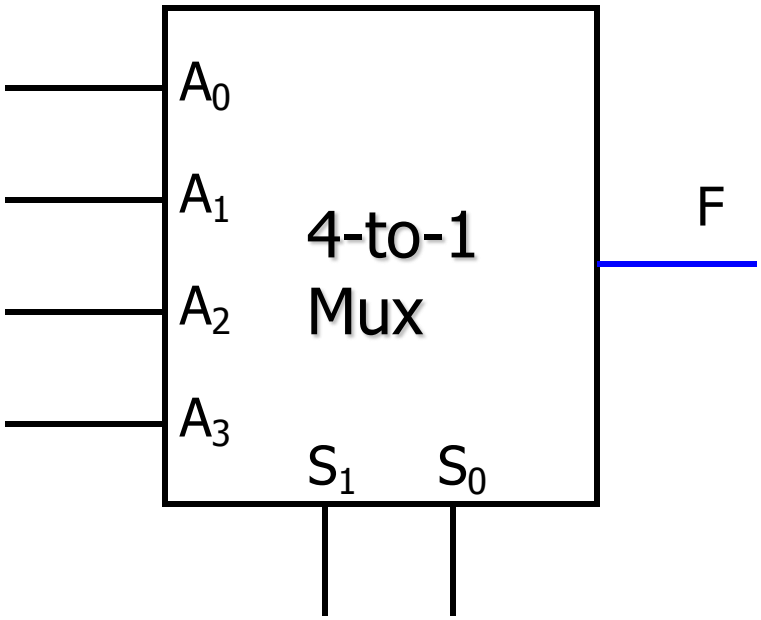
- Functionality: Selection of a particular input
- Route 1 of N inputs (A) to the output F
- Require $\log_2^N$ selection bits (S)
- En(able) bit can disable the route and set F to 0

Multiplexers (Mux) w/out Enable



S1	S0	A3	A2	A1	A0	F
0	0	X	X	X	0	0
0	1	X	X	0	X	0
1	0	X	0	X	X	0
1	1	0	X	X	X	0
0	0	X	X	X	1	1
0	1	X	X	1	X	1
1	0	X	1	X	X	1
1	1	1	X	X	X	1

Multiplexers (Mux) w/out Enable

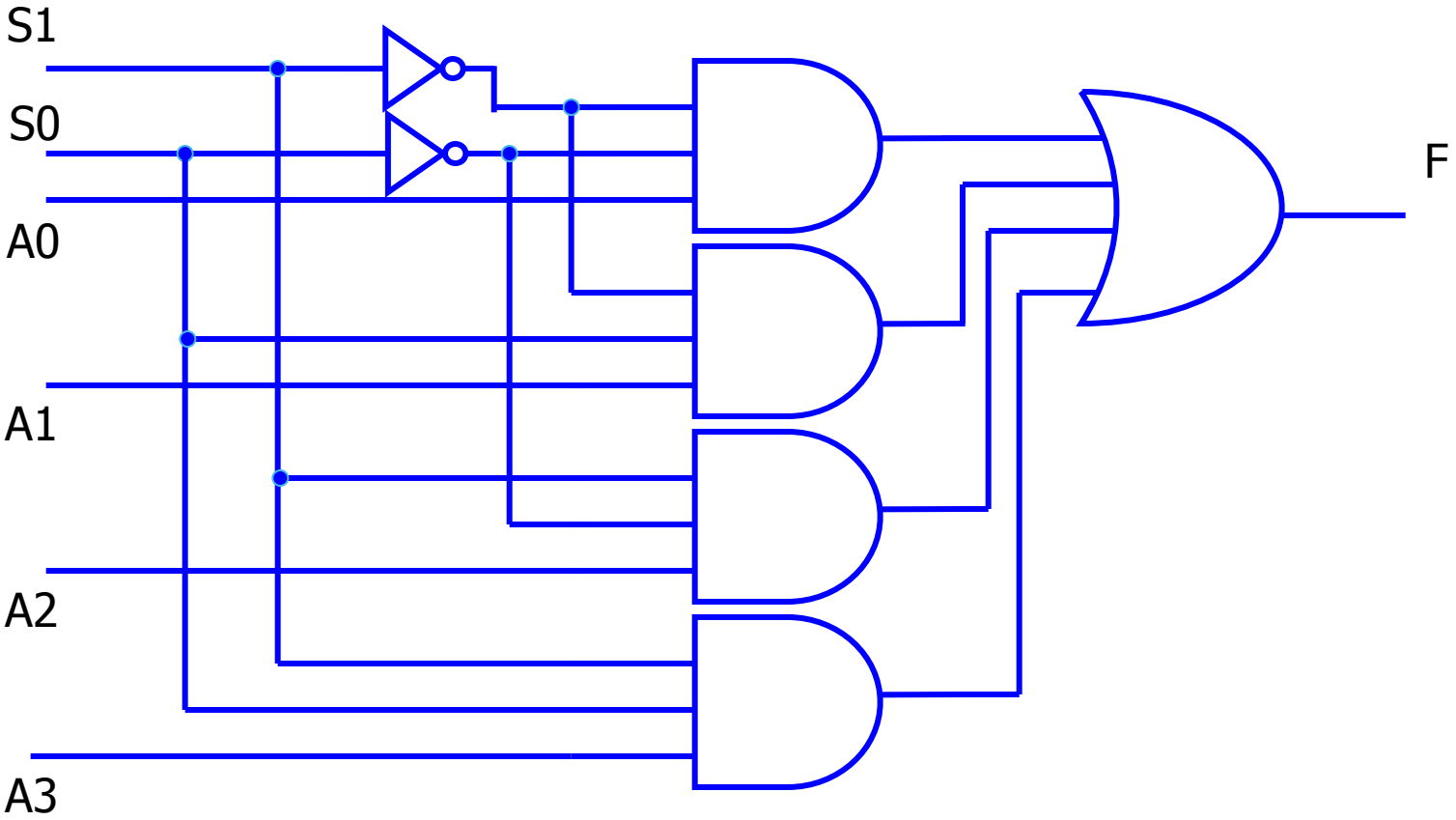


S1	S0	F
0	0	A0
0	1	A1
1	0	A2
1	1	A3

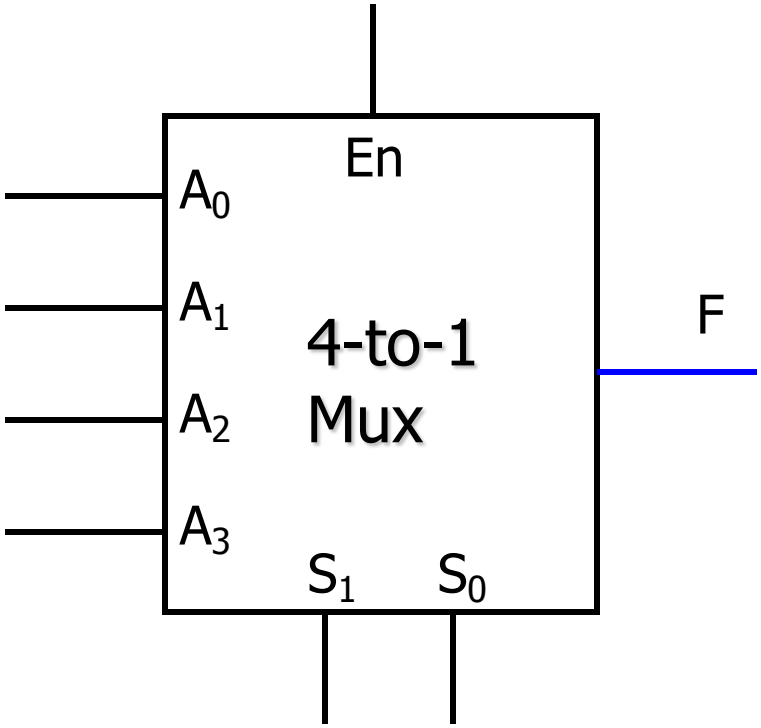
$$F = \overline{S_1}\,\overline{S_0}A_0 + \overline{S_1}S_0A_1 + S_1\overline{S0}A_2 + S_1S_0A_3$$

Logic Diagram of a 4-to-1 Mux

$$F = \overline{S_1}\,\overline{S_0}A_0 + \overline{S_1}S_0A_1 + S_1\overline{S0}A_2 + S_1S_0A_3$$



Multiplexers (Mux) w/ Enable



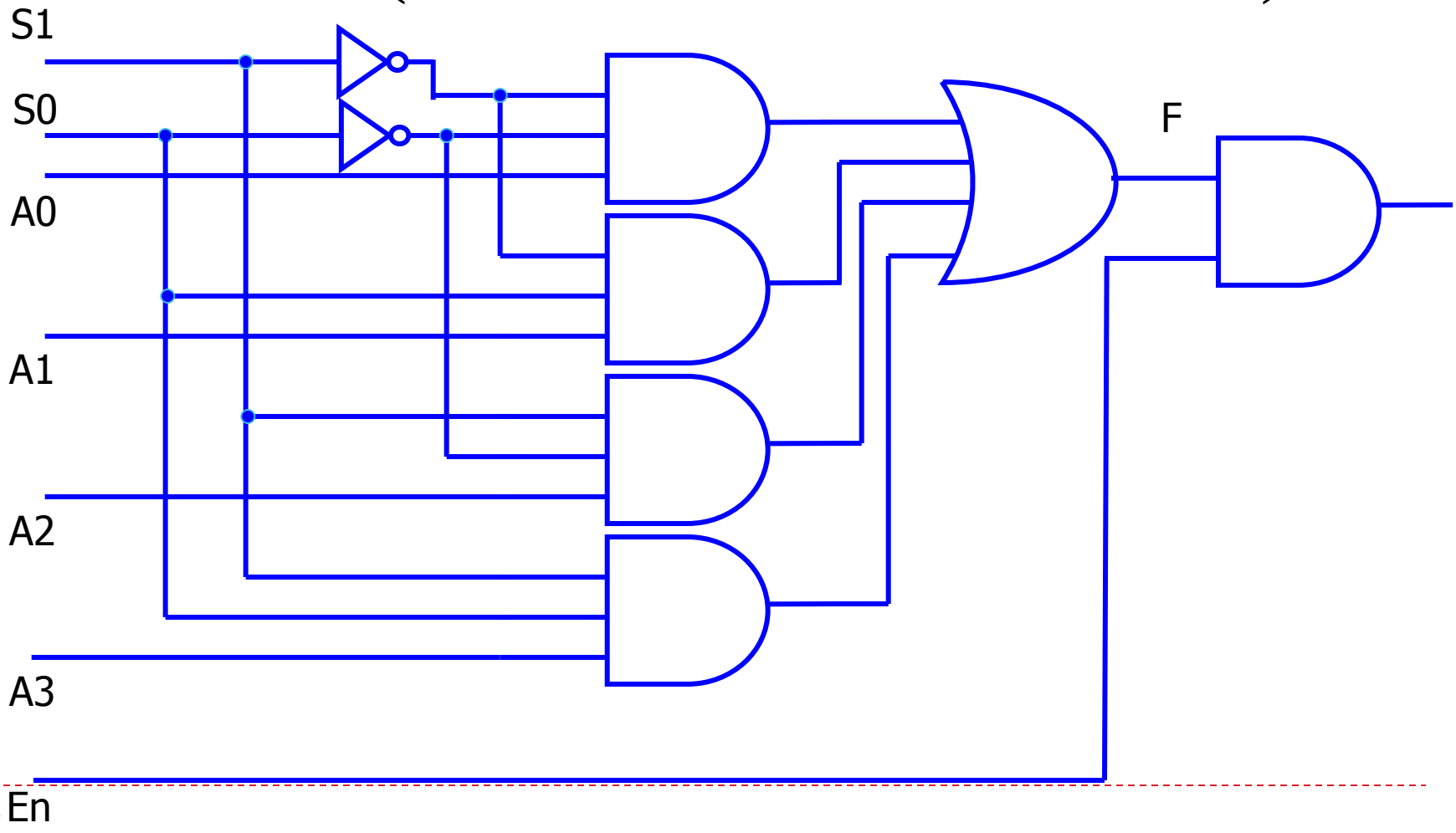
En	S1	S0	F
0	X	X	0
1	0	0	A0
1	0	1	A1
1	1	0	A2
1	1	1	A3

$$F = En \cdot (\overline{S_1}\,\overline{S_0}A_0 + \overline{S_1}S_0A_1 + S_1\overline{S0}A_2 + S_1S_0A_3)$$

$$= En\overline{S_1}\,\overline{S_0}A_0 + En\overline{S_1}S_0A_1 + EnS_1\overline{S0}A_2 + EnS_1S_0A_3$$

4-to-1 Mux w/ Enable Logic

$$F = En \cdot (\overline{S_1}\,\overline{S_0}A_0 + \overline{S_1}S_0A_1 + S_1\overline{S0}A_2 + S_1S_0A_3)$$

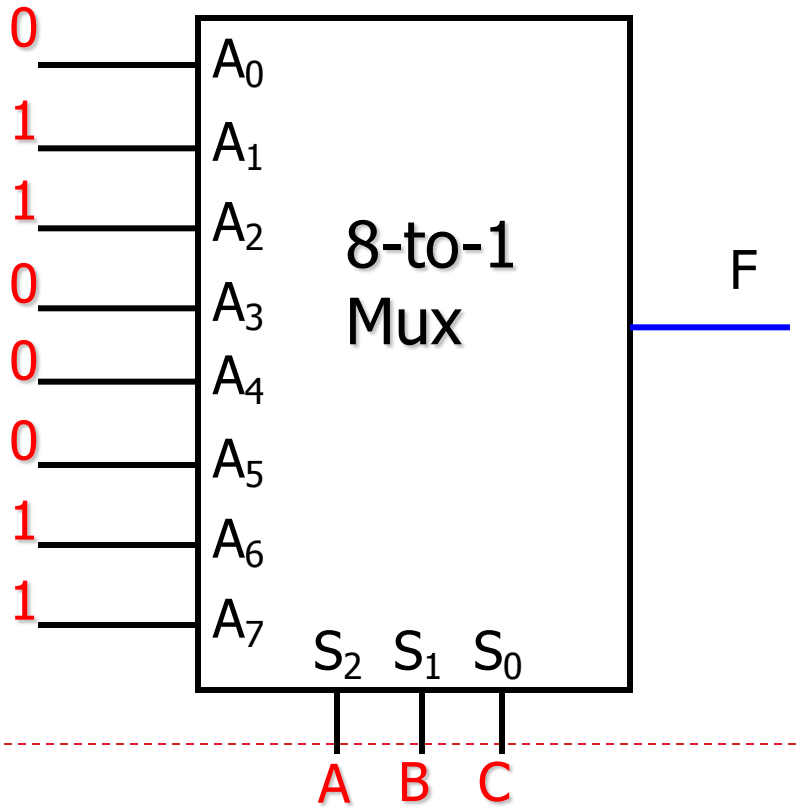


Design Canonical Form w/ MUX

$$F(A,B,C) = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC$$

$$F(A,B,C) = \sum m(1,2,6,7)$$

Each input in a MUX is a minterm



Design Canonical Form w/ MUX

$$F(A,B,C) = \sum m(1, 2, 6, 7)$$

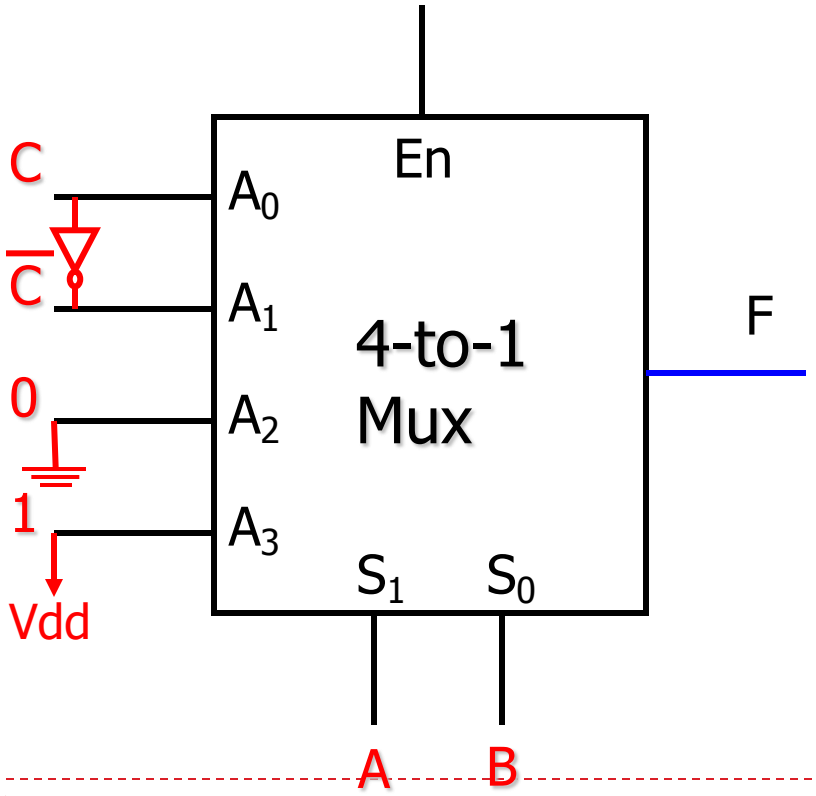
$$F = \overline{A}\,\overline{B}C + \overline{A}B\overline{C} + AB\overline{C} + ABC$$

A	B	F
0	0	
0	1	
1	0	
1	1	

Design Canonical Form w/ MUX

$$F = \sum m(1, 2, 6, 7)$$

$$F = \overline{A}\,\overline{B}C + \overline{A}B\overline{C} + AB\overline{C} + ABC$$

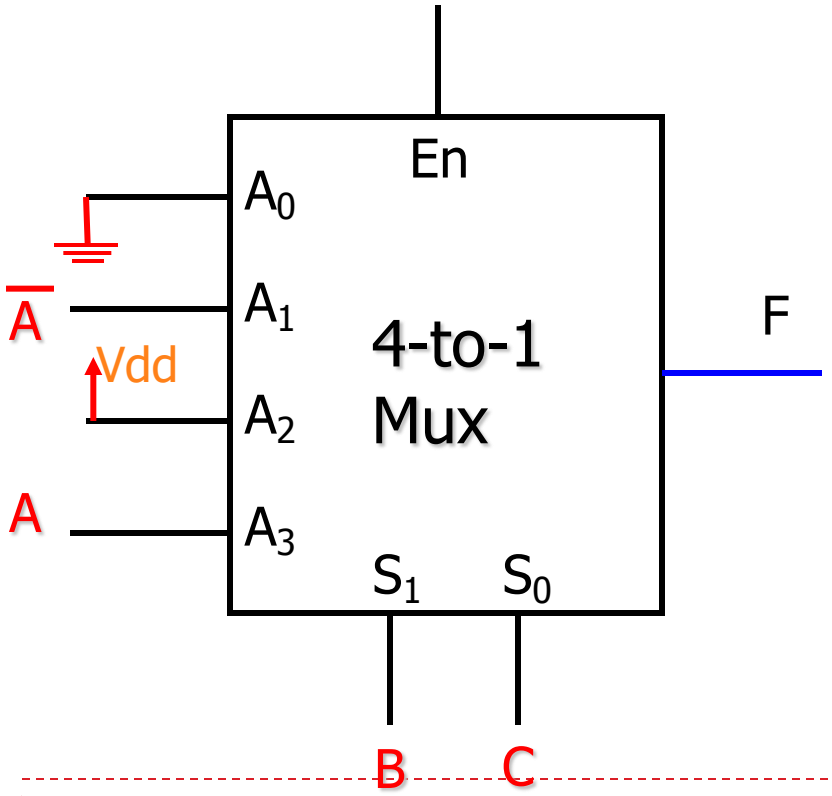


A	B	F
0	0	C
0	1	$\overline{C}$
1	0	0
1	1	1

Design Canonical Form w/ MUX

$$F = \sum m(1, 2, 6, 7)$$

$$F = \overline{A}\,\overline{B}C + \overline{A}B\overline{C} + AB\overline{C} + ABC$$



B	C	F
0	0	0
0	1	$\overline{A}$
1	0	1
1	1	A

Implement a function using MUX

A combinational circuit is specified by the following Boolean function:

$$F(A,B,C,D) = \Sigma(0,2,6,7,8)$$

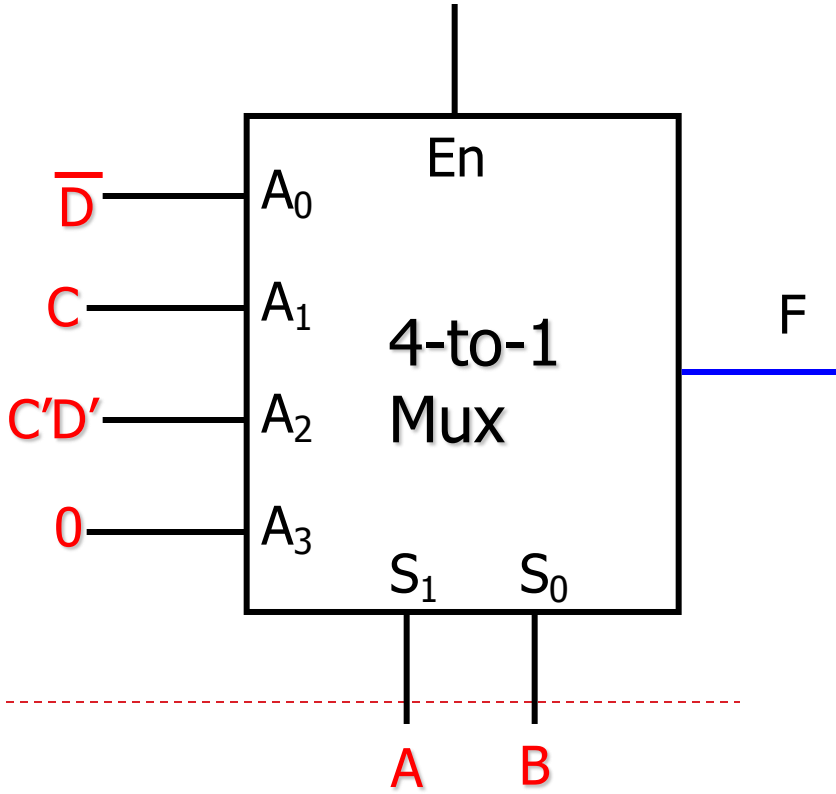
Implement the circuit using ONE 4:1 multiplexer and other necessary gates. Use A and B as MUX selection inputs. Only uncomplemented inputs are available.

Ans. F = A'B'C'D' + A'B'CD' + A'BCD' + A'BCD + AB'C'D'

F = A[B'C'D'] + A'[B'C'D' + B'CD' + BCD' + BCD]

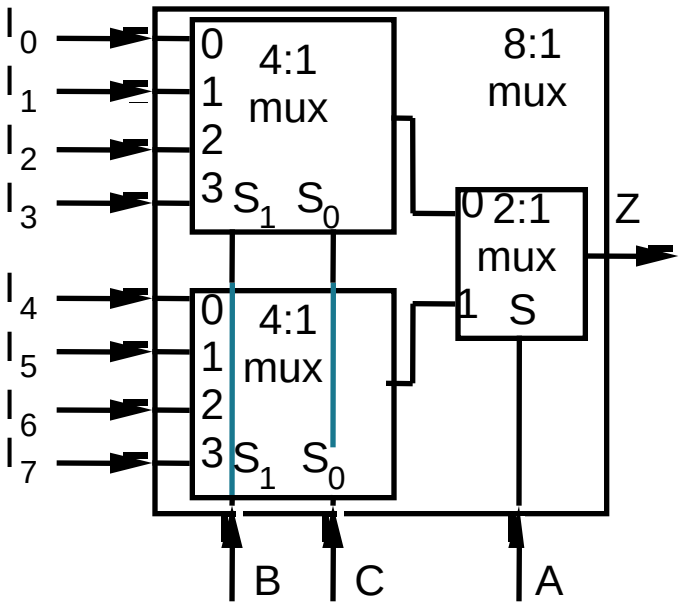
F = A[B(0) + B'(C'D')] + A'[B(CD' + CD) + B'(C'D' + CD')]

F = A[B(0) + B'(C'D')] + A'[B(C) + B'(D')]



Design of Large Multiplexers

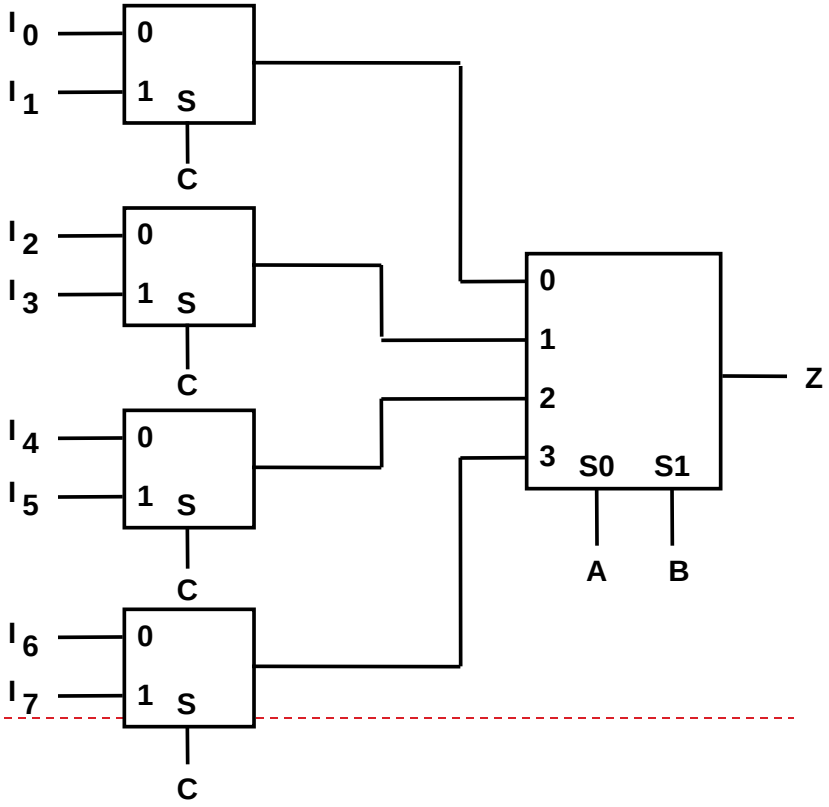
Large multiplexers can be implemented by cascaded smaller ones



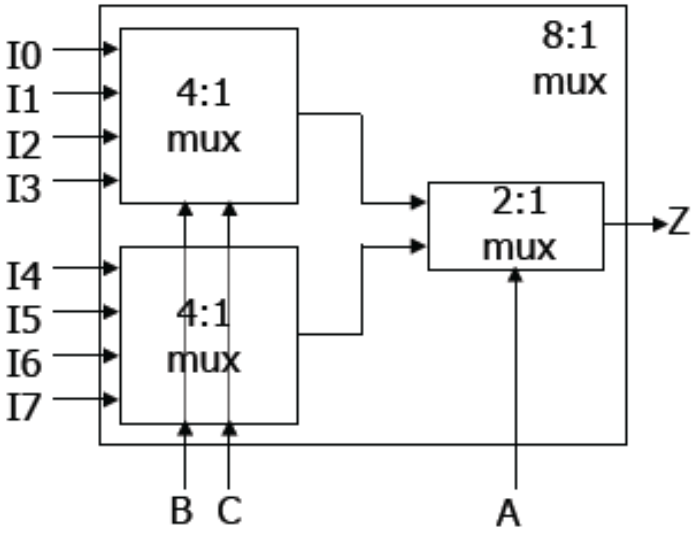
Alternative 8:1 Mux Implementation

Control signals B and C simultaneously choose one of I0-I3 and I4-I7

Control signal A chooses which of the upper or lower MUX's output to gate to Z



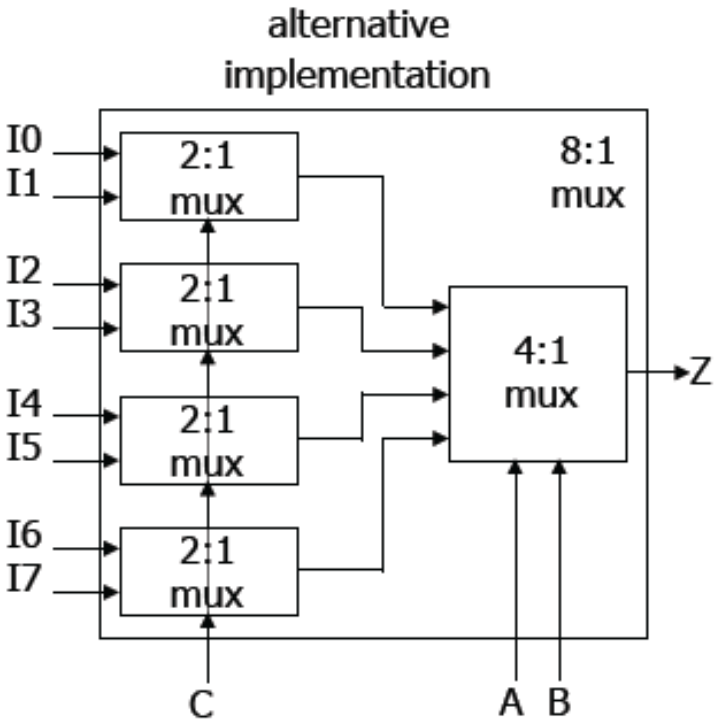
- Large multiplexers can be made by cascading smaller ones



control signals B and C simultaneously choose one of I0, I1, I2, I3 and one of I4, I5, I6, I7

control signal A chooses which of the upper or lower mux's output to gate to Z

alternative implementation



Multiplexers/Selectors as General Purpose Blocks

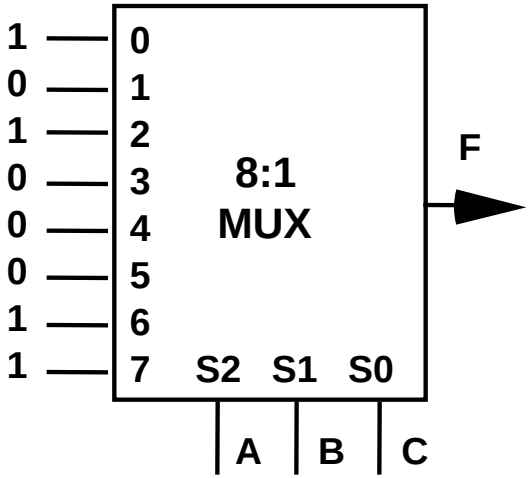
2^{n-1}:1 multiplexer can implement any function of n variables

n-1 control variables; remaining variable is a data input to the mux

Example: **F(A,B,C) = m0 + m2 + m6 + m7**

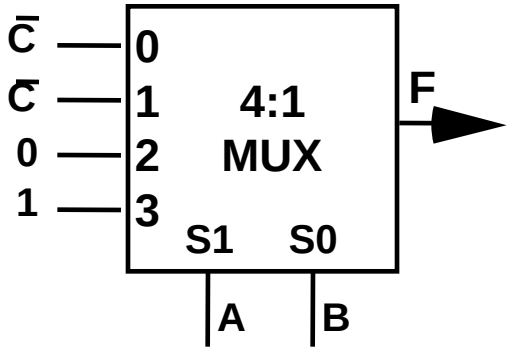
= A' B' C' + A' B C' + A B C' + A B C

= A' B' (C') + A' B (C') + A B' (0) + A B (1)



"Lookup Table"

A	B	C	F	
0	0	0	1	$\bar{C}$
0	0	1	0	
0	1	0	1	$\bar{C}$
0	1	1	0	
1	0	0	0	0
1	0	1	0	
1	1	0	1	1
1	1	1	1	



- Realize F = B'CD' + ABC' with a 4:1 multiplexer and a minimum of other gates:

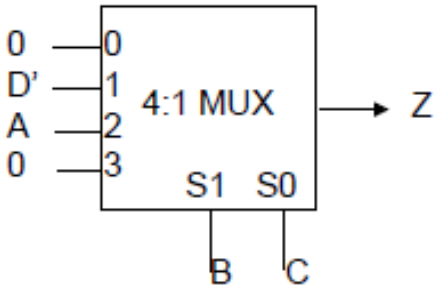
A	B	C	D	Z
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

0 when B′C′

D′ when B′C

A when BC′

0 when BC



$$Z = B'C'(0) + B'C(D') + BC'(A) + BC(0)$$