

**CpE358/CS381**

**Switching Theory and  
Logical Design**

**Class 5**

# Today

- Fundamental concepts of digital systems (Mano Chapter 1)
- Binary codes, number systems, and arithmetic (Ch 1)
- Boolean algebra (Ch 2)
- Simplification of switching equations (Ch 3)
- Digital device characteristics (e.g., TTL, CMOS)/design considerations (Ch 10)
- **Combinatoric logical design including LSI implementation (Chapter 4)**
- Hazards, Races, and time related issues in digital design (Ch 9)
- Flip-flops and state memory elements (Ch 5)
- Sequential logic analysis and design (Ch 5)
- Synchronous vs. asynchronous design (Ch 9)
- Counters, shift register circuits (Ch 6)
- Memory and Programmable logic (Ch 7)
- Minimization of sequential systems
- Introduction to Finite Automata

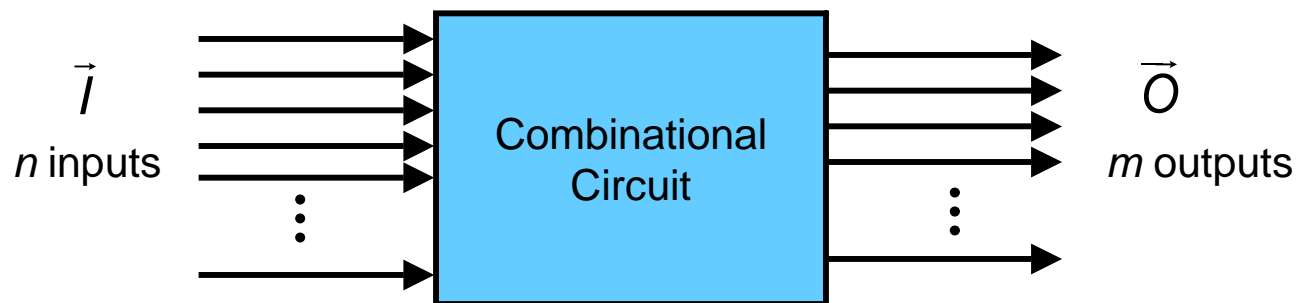
# Standard Combinational Circuits in TTL

- All part number prefixed by 74, 74S, 74LS, etc.

'00, '01, '03, '10, '12, '13, '18, '20, '22, '26, '30, '37, '38, '39, '40	NAND	'70-'79	Flip-Flops
'02, '23, '24, '25, '27, '28, '33	NOR	'80, '82, '83,	Adders
'04, '05, '06, '14, '16	NOT	'81, '84, '89	RAM
'07, '17	Buffer	'85	Magnitude comparator
'08, '09, '11, '15, '21	AND	'86	XOR
'32	OR	'87	True/complement
'41, '42, '43, '44	4-line to 10-line decoder	'88	ROM
'45	BCD-to-decimal	'90-'116 '160-'179, '190-'199	Counters, S/R, Latches
'46, '47, '48, '49	BCD-to-7-segment	'138, '139, '148-'159, '251-258, '348-'359	Decoder, MUX, Encoders, Selectors, DeMUX
'35, '50, '51, '53, '54, '55, '56	AND-OR-Invert		
'52	AND-OR		
'60, '61	AND expander		
'62	AND-OR expander		

# Generalized Combinational Circuits

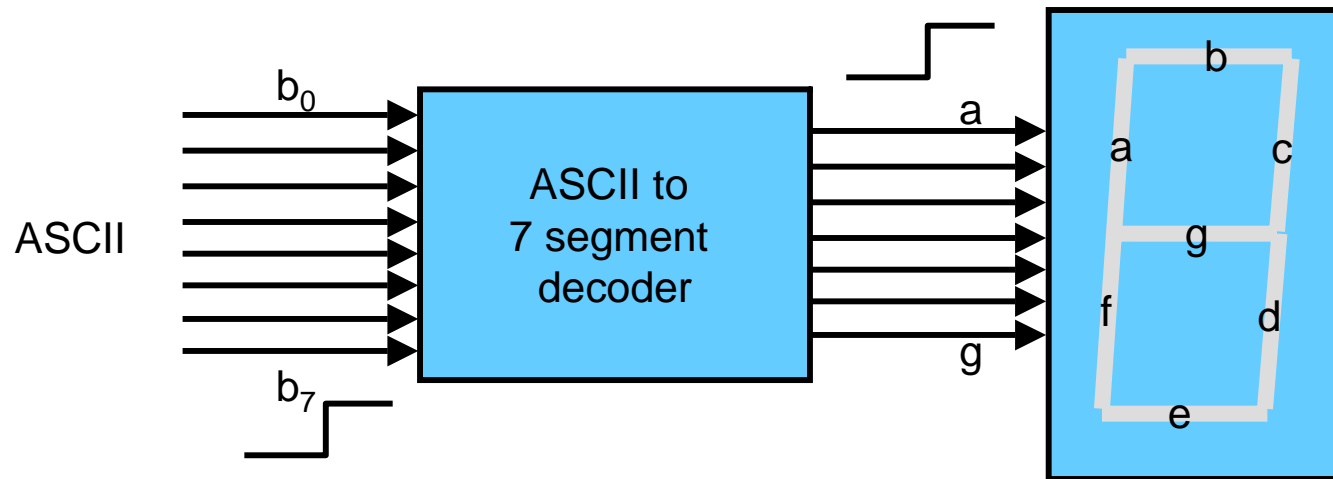
- In general,  $F( )$  will represent some high-level function needed for a system



$$\vec{O} = (o_0, o_1, o_2, \dots, o_{m-1}) = F(\vec{I}) = F(i_0, i_1, i_2, \dots, i_{n-1})$$

# Generalized Combinational Circuits

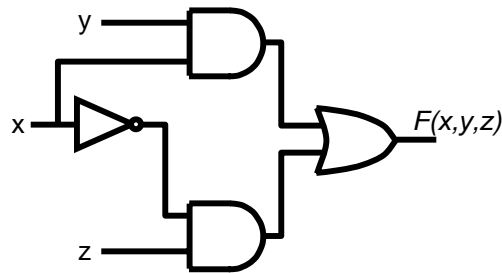
- E.g., ASCII to 7-segment decoder needed for a display system



$$\vec{O} = (a, b, c, d, e, f, g) = F(\vec{B}) = F(b_0, b_1, b_2, \dots, b_7)$$

# Analysis vs. Design

- Analysis procedure:
  - Given a logic diagram, find  $F()$

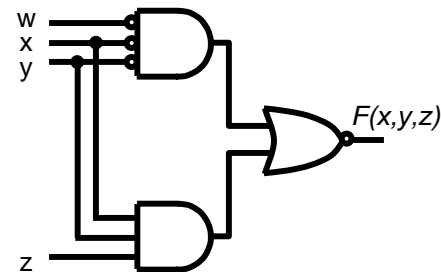


$$F(x, y, z) = x \cdot y + x' \cdot z$$

- Design procedure:
  - Given  $F()$ , design a logic circuit that implements  $F()$  with minimum number of gates

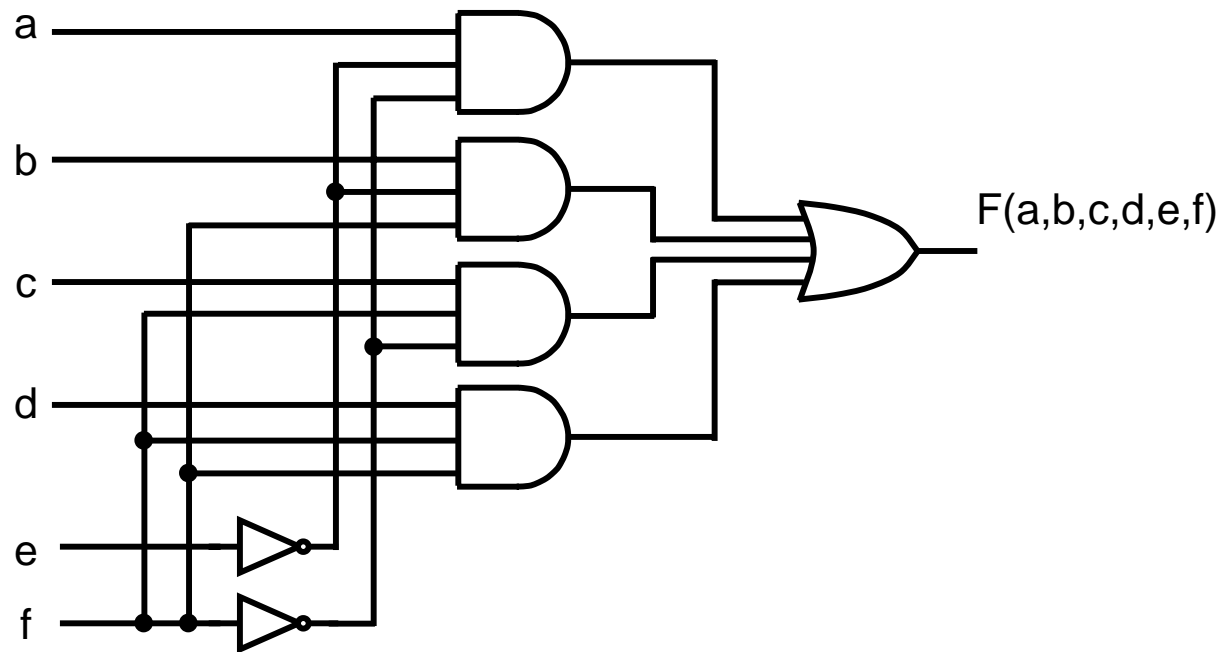
$$F(x, y, z) = xy + xz' + xy' + w = (w' x' y' + xyz)'$$

wxyz	00	01	11	10
00	0	0	1	1
01	1	1	0	1
11	X	X	X	X
10	1	1	X	X



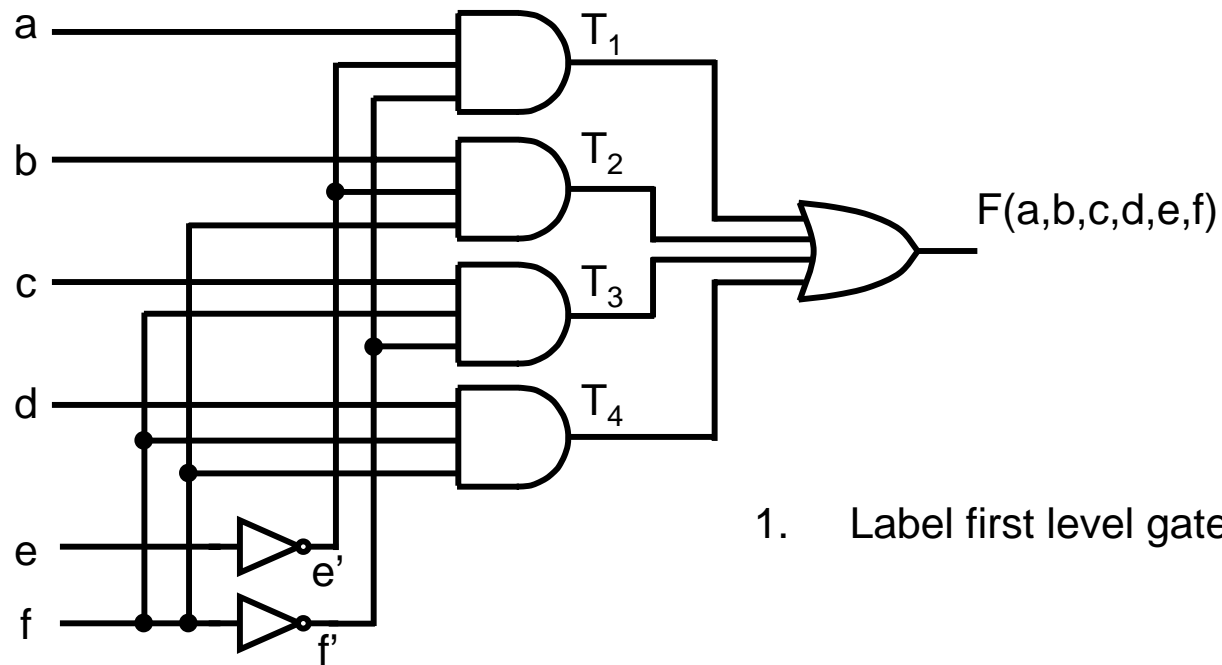
# Analysis

- Consider this logic diagram – what function does it perform?:



# Analysis

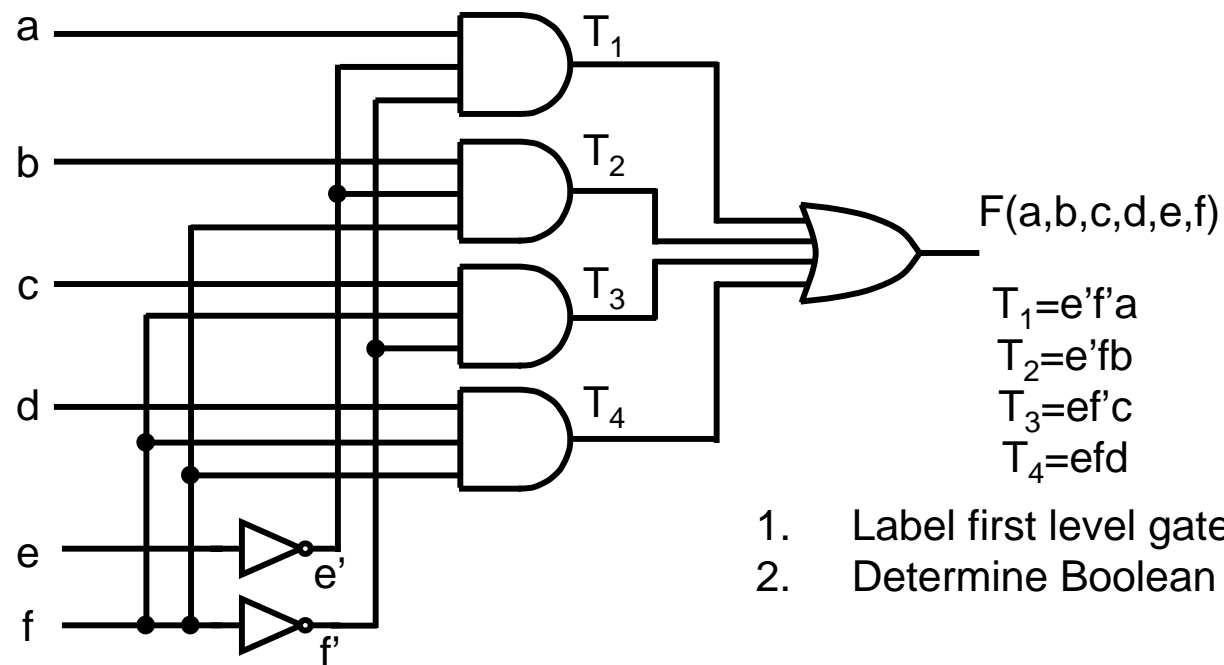
- Consider this logic diagram – what function does it perform?:



1. Label first level gate outputs

# Analysis

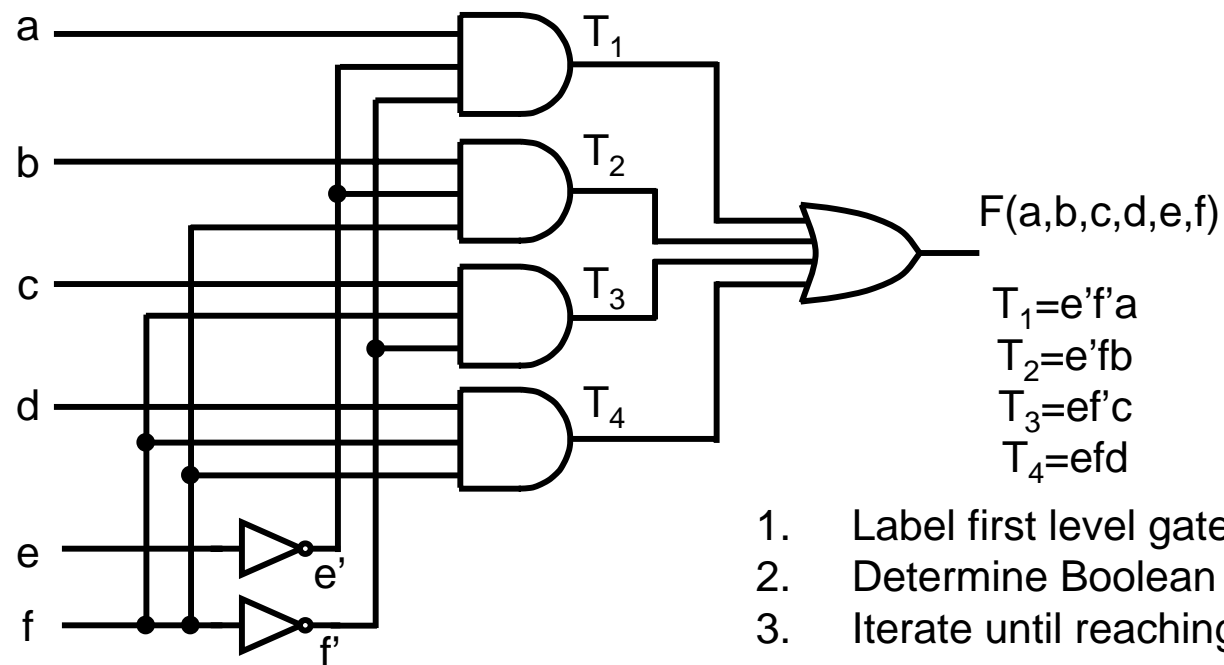
- Consider this logic diagram – what function does it perform?:



1. Label first level gate outputs
2. Determine Boolean functions

# Analysis

- Consider this logic diagram – what function does it perform?:



$F(a,b,c,d,e,f)$

$$\begin{aligned} T_1 &= e'f'a \\ T_2 &= e'fb \\ T_3 &= ef'c \\ T_4 &= efd \end{aligned}$$

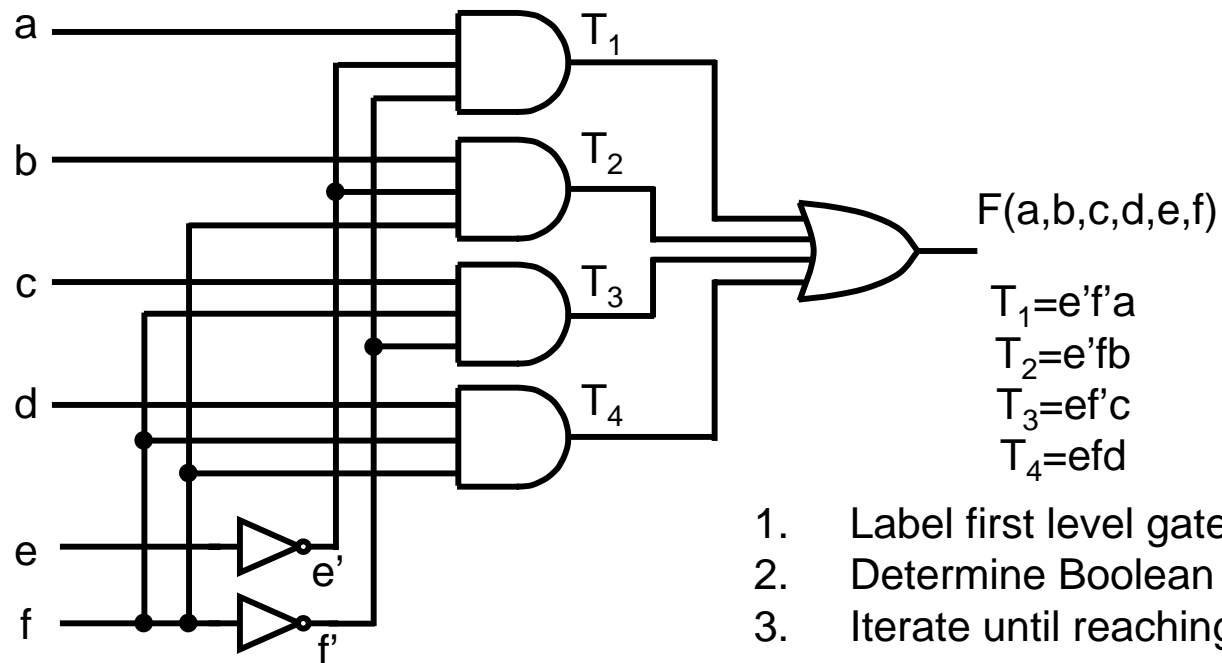
1. Label first level gate outputs
2. Determine Boolean functions
3. Iterate until reaching output

$$F(a,b,c,d,e,f) = T_1 + T_2 + T_3 + T_4$$

$$F(a,b,c,d,e,f) = e'f'a + e'fb + ef'c + efd$$

# Analysis

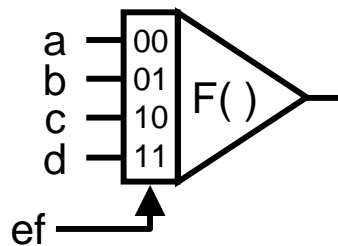
- Consider this logic diagram – what function does it perform?:



1. Label first level gate outputs
2. Determine Boolean functions
3. Iterate until reaching output

$$F(a,b,c,d,e,f) = T_1 + T_2 + T_3 + T_4$$

$$F(a,b,c,d,e,f) = e'f'a + e'fb + ef'c + efd$$



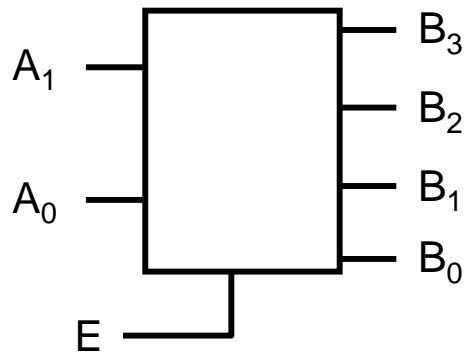
$F()$  is a 4 input MUX

# Design

- Design a 2-line to 4-line decoder with enable
1. Create a high-level function definition

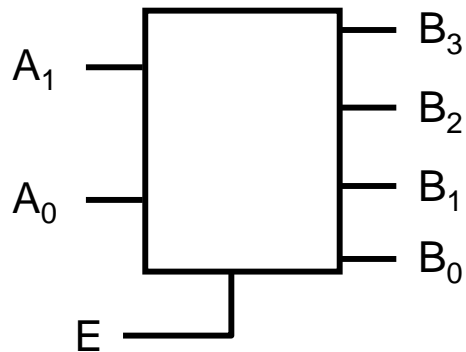
# Design

- Design a 2-line to 4-line decoder with enable
1. Create a high-level function definition, determine I/O requirements



# Design

- Design a 2-line to 4-line decoder with enable

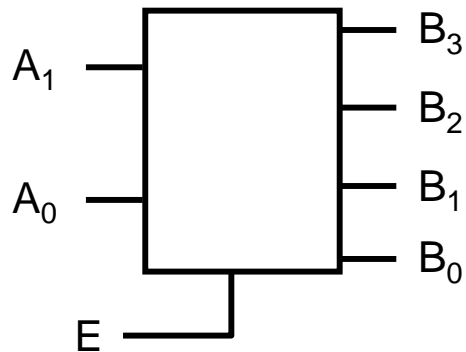


1. Create a high-level function definition, determine I/O requirements
2. Define truth table

E	$A_1$	$A_0$	$B_0$	$B_1$	$B_2$	$B_3$
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

# Design

- Design a 2-line to 4-line decoder with enable



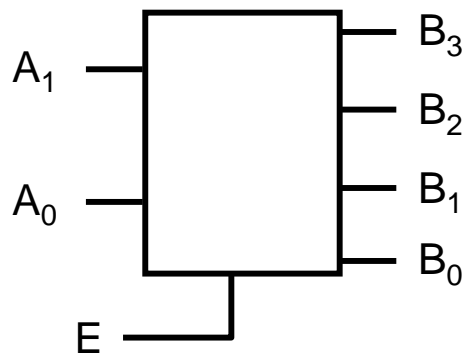
1. Create a high-level function definition, determine I/O requirements
2. Define truth table
3. Derive Boolean functions for each output

E	A <sub>1</sub>	A <sub>0</sub>	B <sub>0</sub>	B <sub>1</sub>	B <sub>2</sub>	B <sub>3</sub>
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

$$\begin{aligned} B_0 &= EA_1'A_0' \\ B_1 &= EA_1'A_0 \\ B_2 &= EA_1A_0' \\ B_3 &= EA_1A_0 \end{aligned}$$

# Design

- Design a 2-line to 4-line decoder with enable



E	A <sub>1</sub>	A <sub>0</sub>	B <sub>0</sub>	B <sub>1</sub>	B <sub>2</sub>	B <sub>3</sub>
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

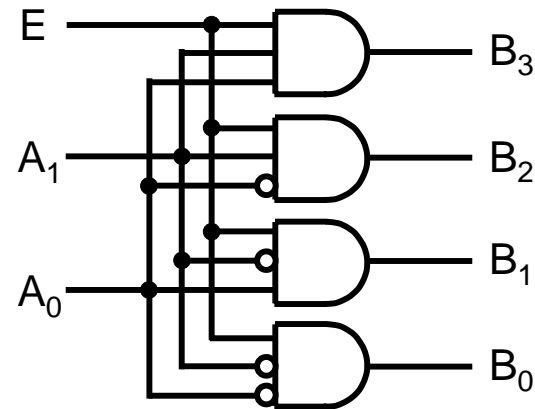
1. Create a high-level function definition, determine I/O requirements
2. Define truth table
3. Derive Boolean functions for each output
4. Draw the logic diagram and verify correctness

$$B_0 = EA_1'A_0'$$

$$B_1 = EA_1'A_0$$

$$B_2 = EA_1A_0'$$

$$B_3 = EA_1A_0$$



# Binary Addition Review

- 1-bit addition:

+	0	1
0	0	1
1	1	0*

\* carry

# Binary Addition Review

- 1-bit addition:

+	0	1
0	0	1
1	1	0*

\* carry

- As two Boolean functions:

+	0	1
0	0	1
1	1	0

sum

+	0	1
0	0	0
1	0	1

carry

# Binary Addition Review

- 1-bit addition:

+	0	1
0	0	1
1	1	0*

\* carry

- As two Boolean functions:

+	0	1
0	0	1
1	1	0

sum

+	0	1
0	0	0
1	0	1

carry

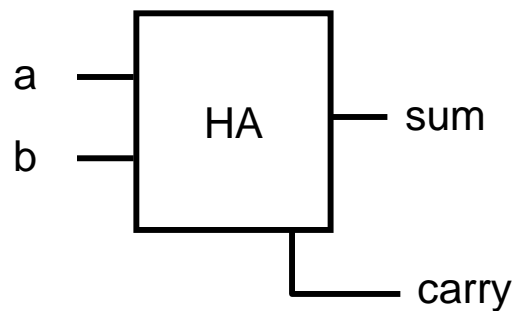
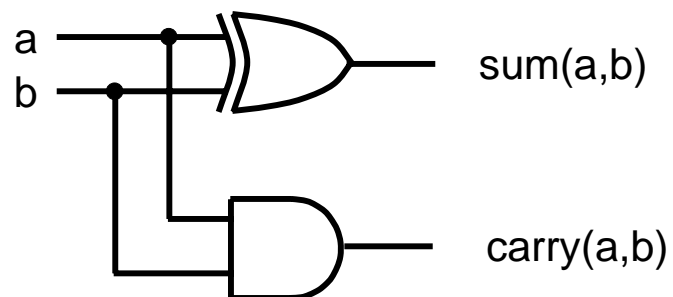
$$\text{sum}(a, b) = a' b + a b' = a \oplus b$$

$$\text{carry}(a, b) = ab$$

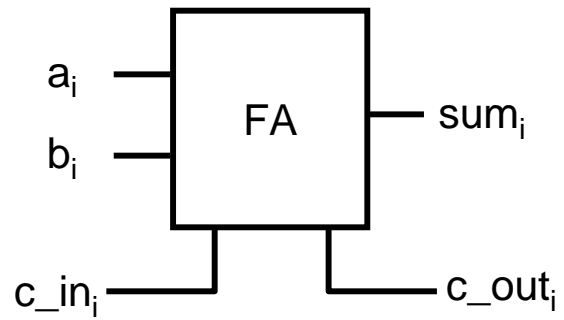
# Half Adder

$$\text{sum}(a,b) = a'b + ab' = a \oplus b$$

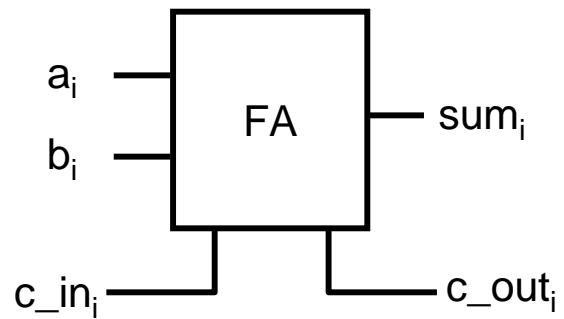
$$\text{carry}(a,b) = ab$$



# Full Adder

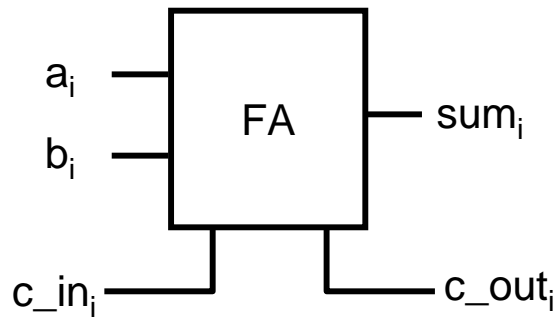


# Full Adder



a	b	$c_{in}$	$c_{out}$	sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

# Full Adder

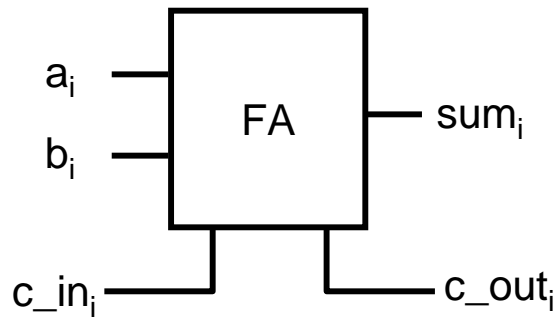


a	b	$c_{in}$	$c_{out}$	sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

		ab		a		
c		00	01	11	10	
0		0	1	0	1	sum
1		1	0	1	0	
		b				

		ab		a		
c		00	01	11	10	
0		0	0	1	0	carry
1		0	1	1	1	
		b				

# Full Adder



a	b	$c_{in}$	$c_{out}$	sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

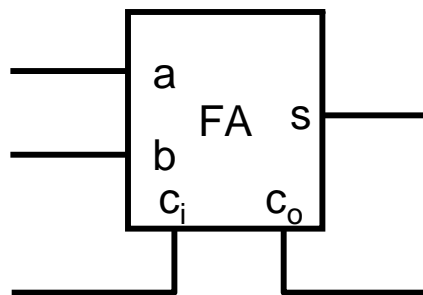
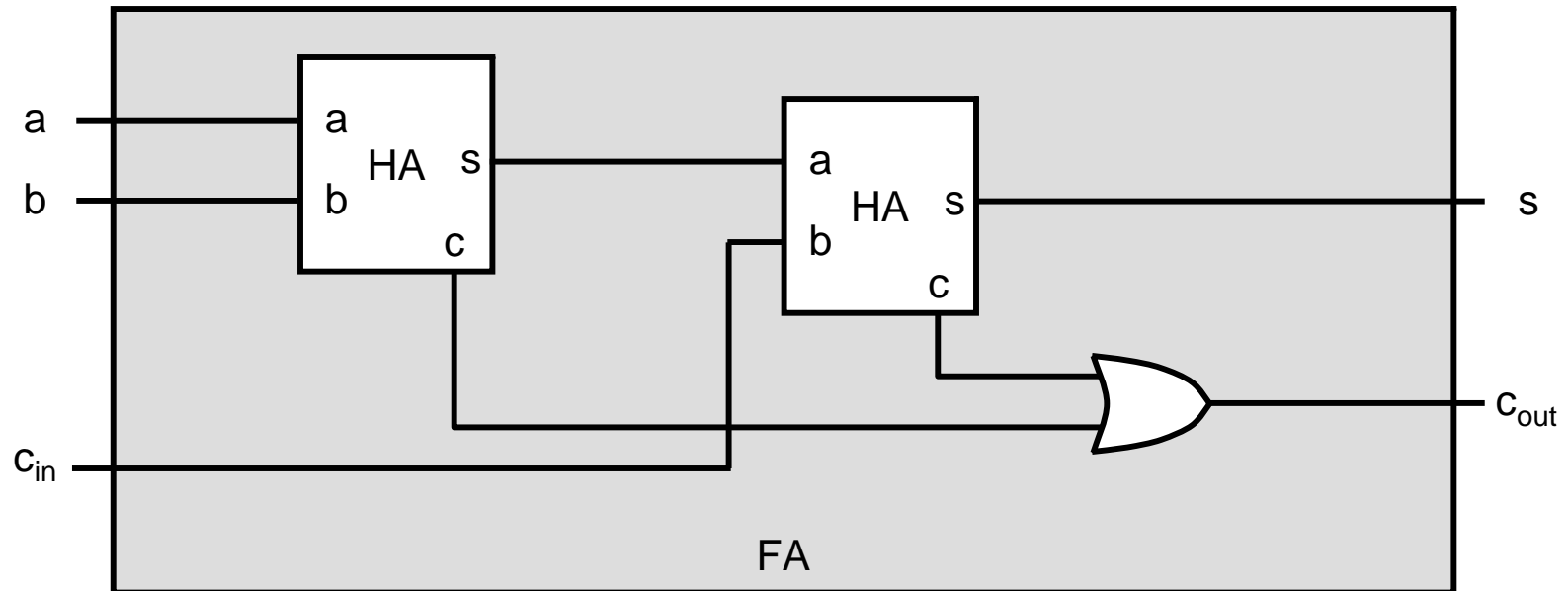
		ab		a		
c		00	01	11	10	
0	c	0	1	0	1	sum
1	c	1	0	1	0	

$$b \quad sum(a, b, c_i) = a \oplus b \oplus c_i$$

		ab		a		
c		00	01	11	10	
0	c	0	0	1	0	carry
1	c	0	1	1	1	

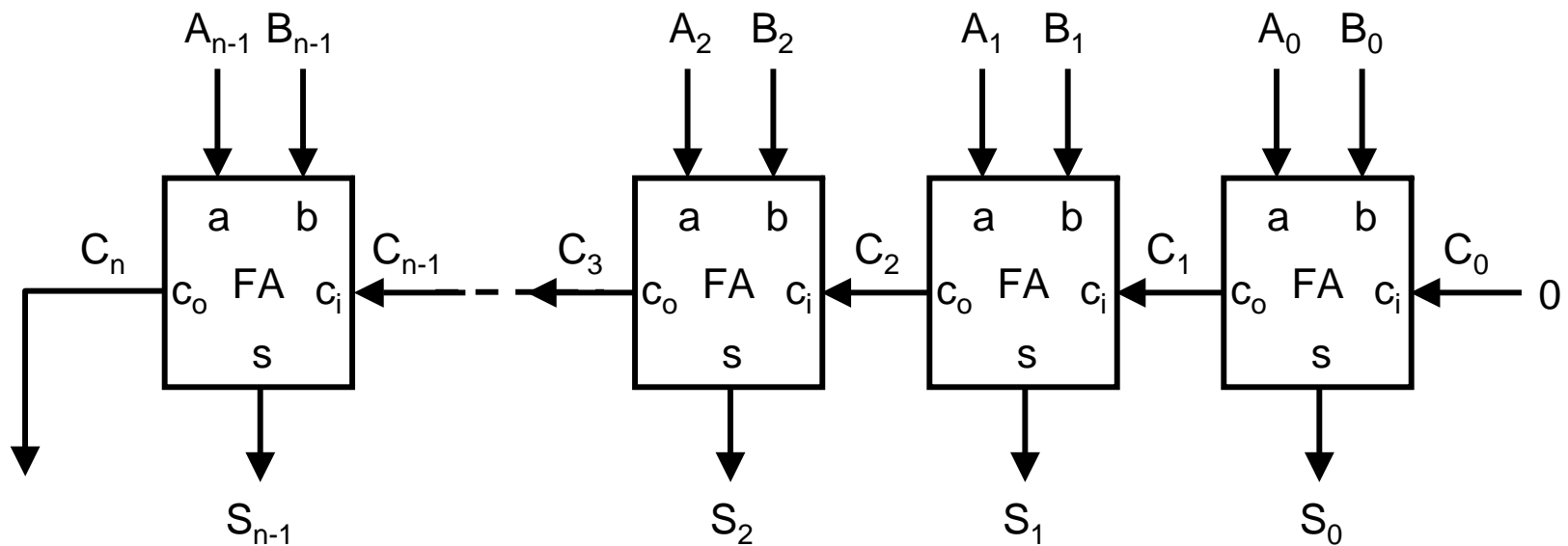
$$b \quad c_o(a, b, c_i) = ac_i + bc_i + ab$$

# Full Adder Implementation



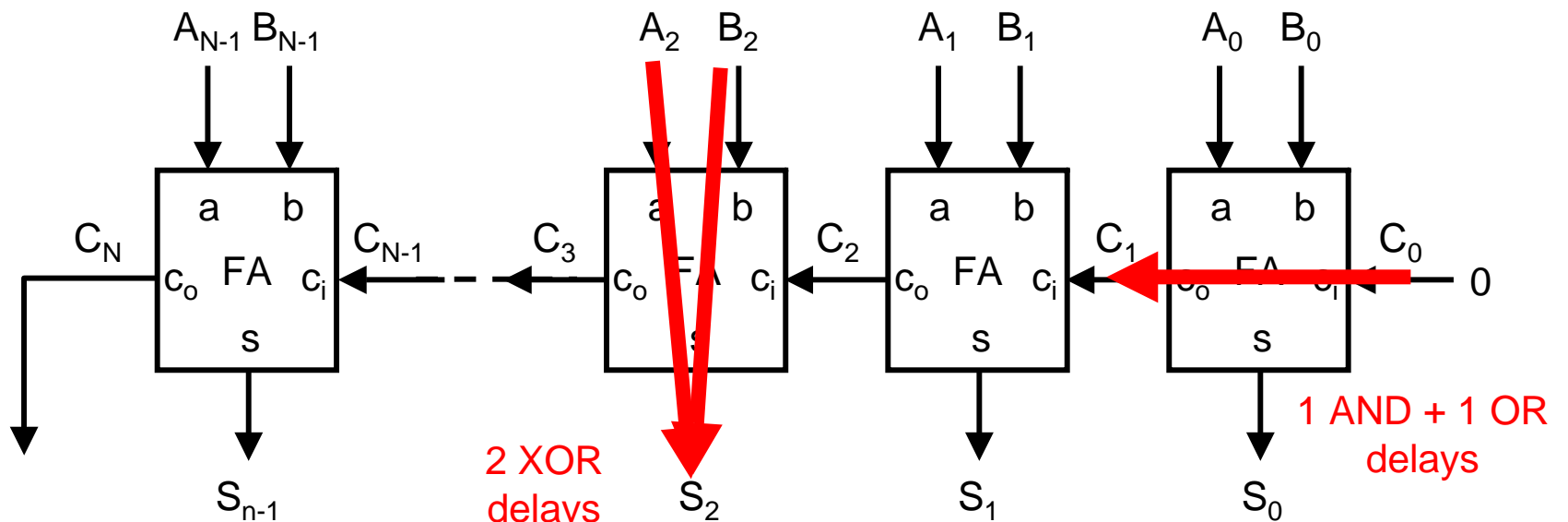
# Binary Adder

- Multiple Full Adders can be cascaded to create an arbitrary precision adder



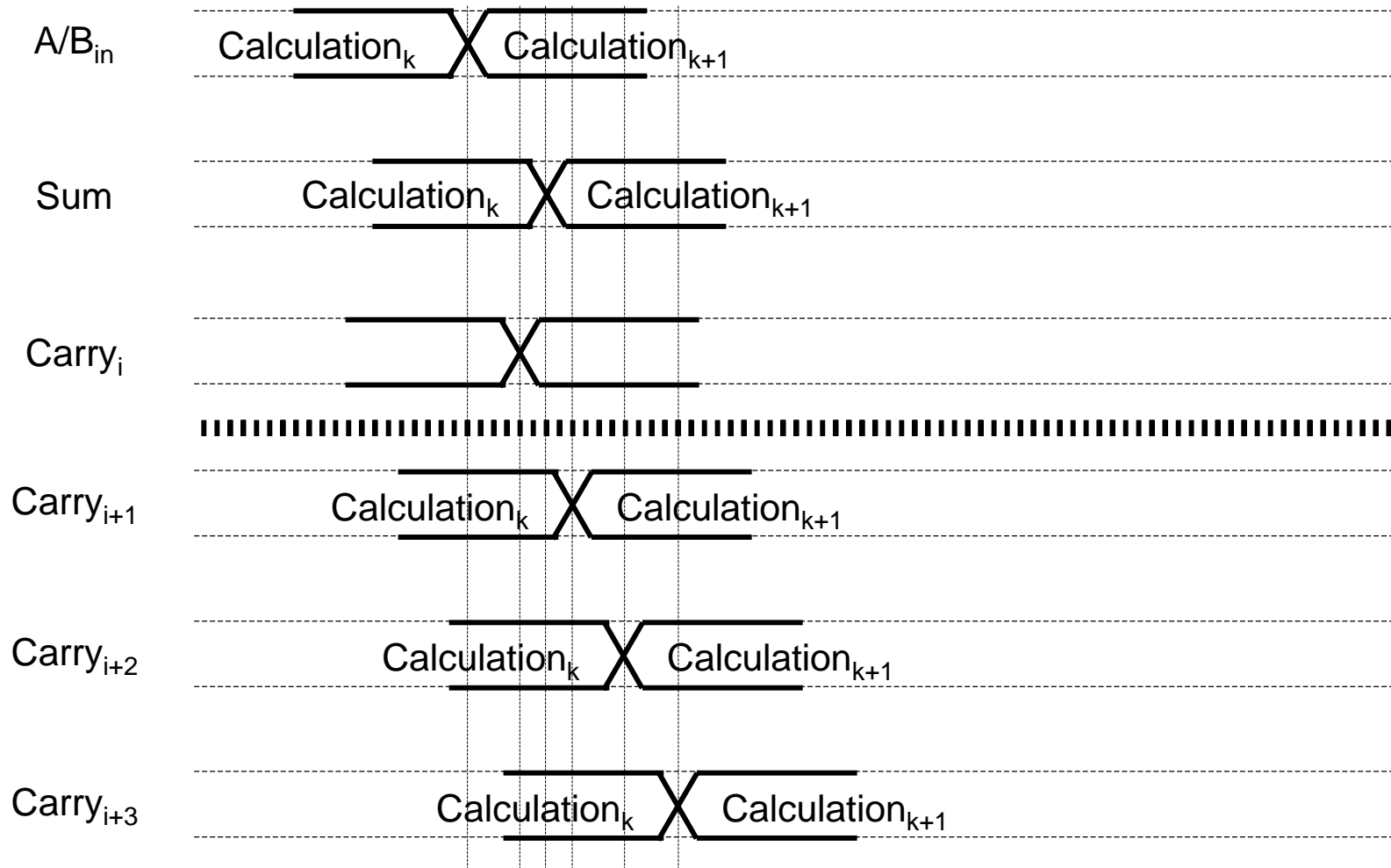
# Binary Adder

- Multiple Full Adders can be cascaded to create an arbitrary precision adder
- But there is an accumulation of delay through the carry stages



(1 AND + 1 OR  
Delays) x (N-1) stages

# Binary Adder Carry Propagation Delay



- 1 XOR delay ~3 gate delays
- 1 AND delay ~2 gate delays
- 1 OR delay ~2 gate delays

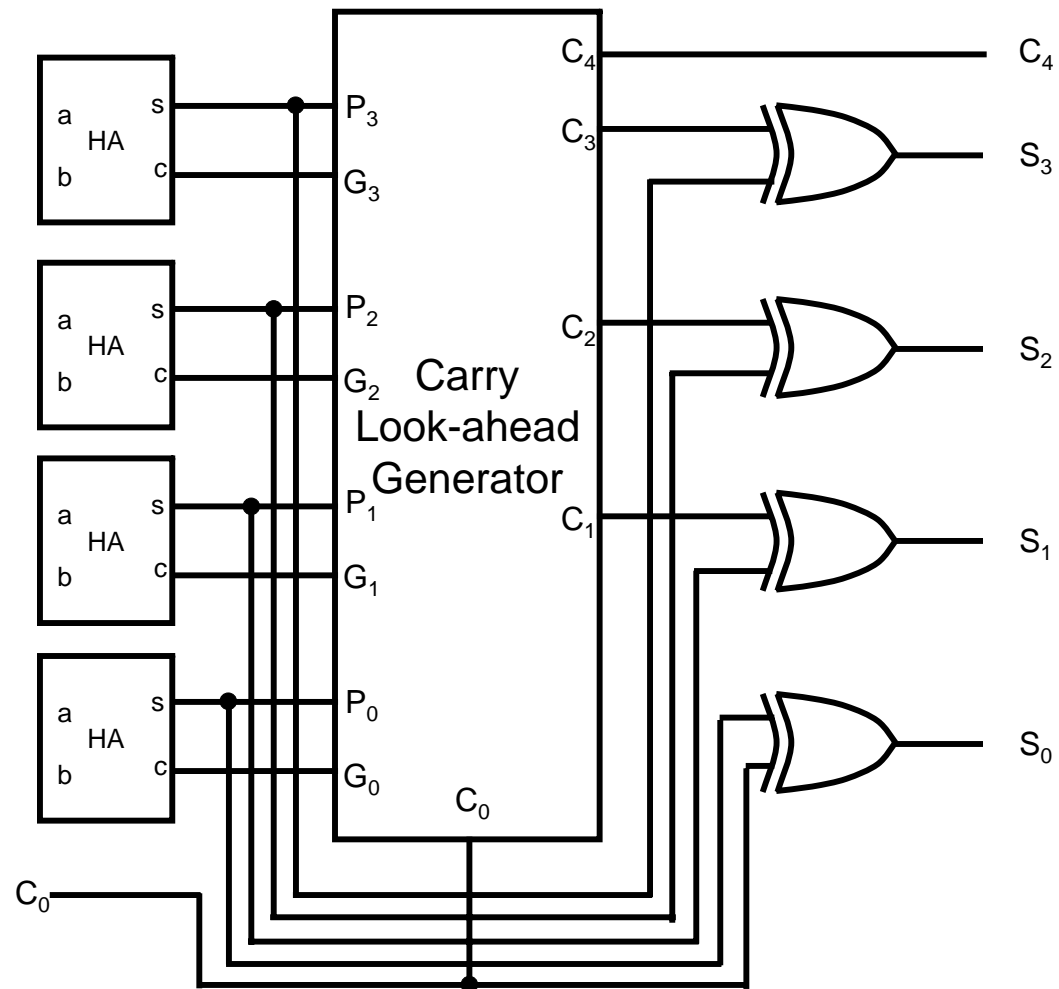
# Carries in Adders

- Consider the 8-bit sum:

$$\begin{array}{r} 01111111 \\ +00000001 \\ \hline 10000000 \end{array}$$

- There is a carry at each stage
- If the necessary carries could be scanned once in advance of the addition, incremental delays could be avoided

# Carries in Adders



# Review of Binary Subtraction

Define:  $\bar{x} = r - x$

$$-B \triangleq \bar{B} + 1$$

$$\begin{array}{r} 01001001 \\ -00110101 \\ \hline \end{array} \longrightarrow 11001010 + 1 \longrightarrow \begin{array}{r} 01001001 \\ +11001011 \\ \hline 100010100 \end{array} \quad \begin{array}{r} 73 \\ -53 \\ \hline 20 \end{array}$$

- Is there an easy way to do the 2's complement in one step?

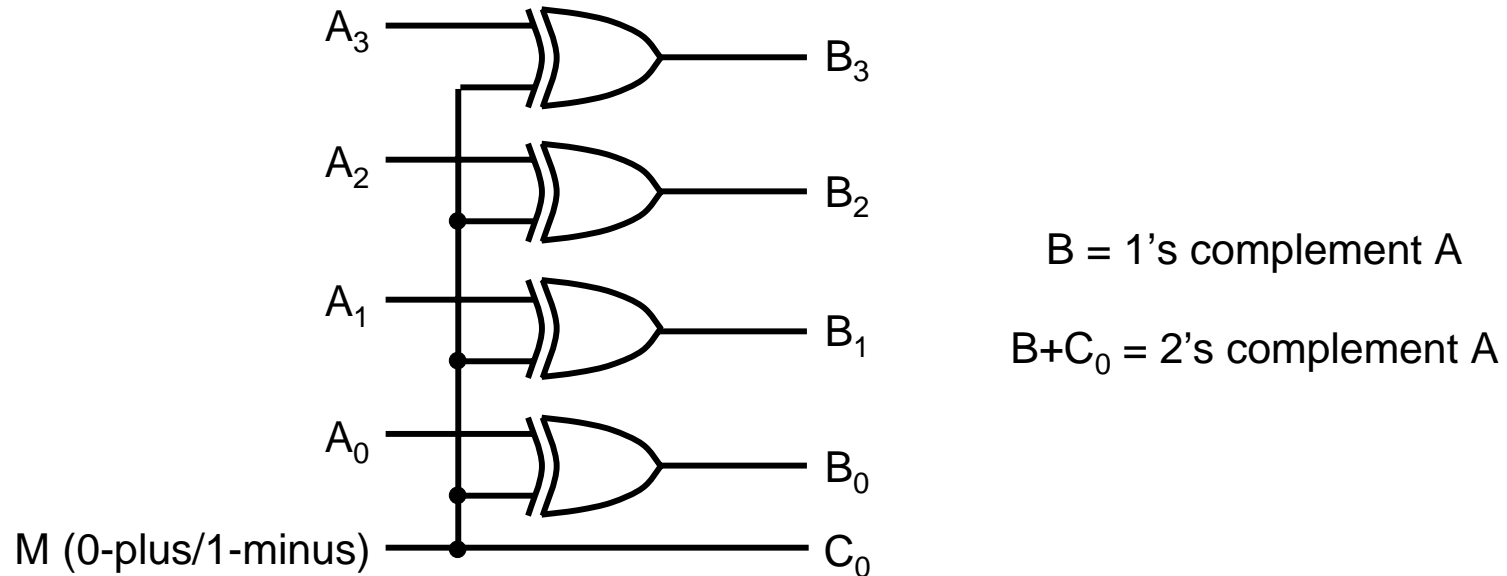
# Review of Binary Subtraction

Define:  $\bar{x} = r - x$

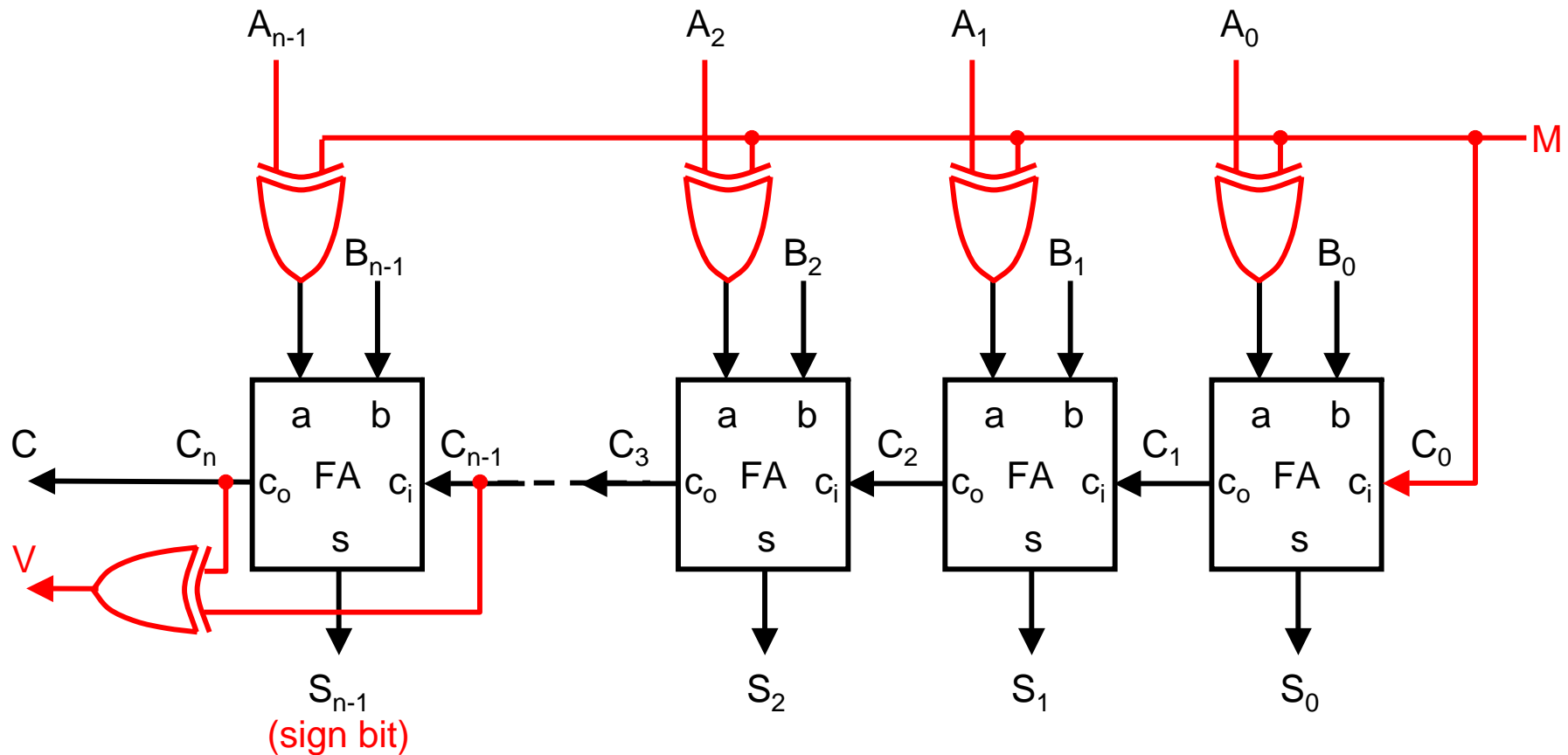
$-B \triangleq \bar{B} + 1$

$$\begin{array}{r}
 01001001 \\
 -00110101 \\
 \hline
 \end{array}
 \longrightarrow
 \begin{array}{r}
 11001010 + 1 \\
 \hline
 \end{array}
 \longrightarrow
 \begin{array}{r}
 01001001 \quad 73 \\
 +11001011 \quad -53 \\
 \hline
 100010100 \quad 20
 \end{array}$$

- Is there an easy way to do the 2's complement in one step?



# Adder/Subtractor



If  $V$  asserted, overflow has occurred

# Review of Multiplication

- Decimal

$$\begin{array}{r} 123 \\ \times 45 \\ \hline 615 \\ 492 \\ \hline 5535 \end{array} \quad \leftarrow \text{carries}$$

# Review of Multiplication

- Decimal

$$\begin{array}{r}
 123 \\
 \times 45 \\
 \hline
 615 \\
 492 \\
 \hline
 5535
 \end{array}$$

← carries

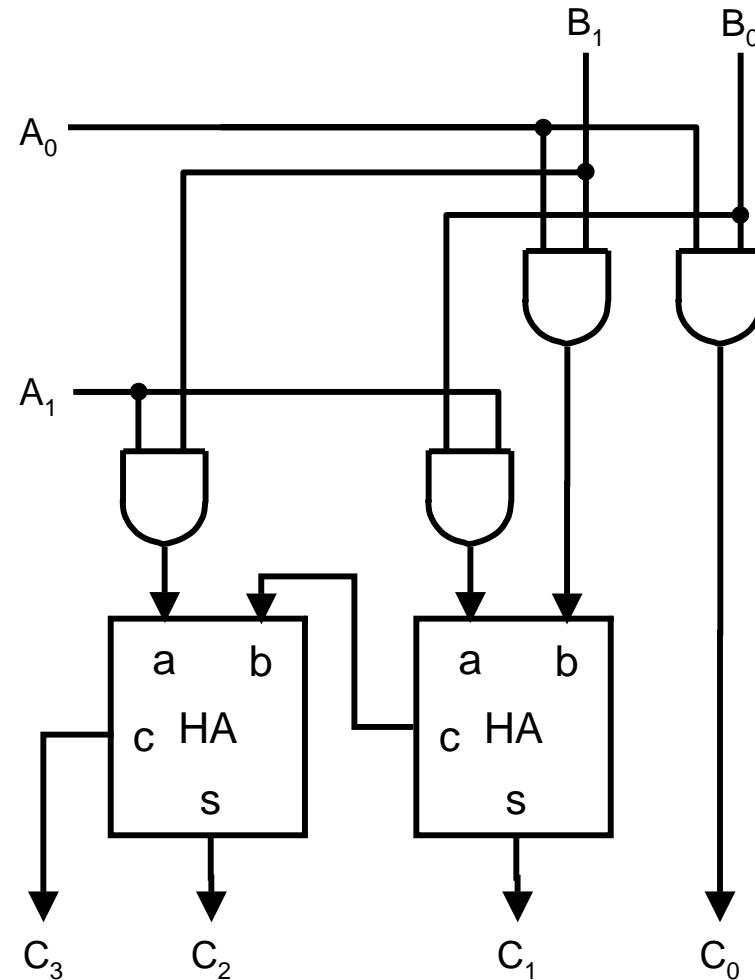
- Binary

no carries →  
(replicate  
and  
shift)

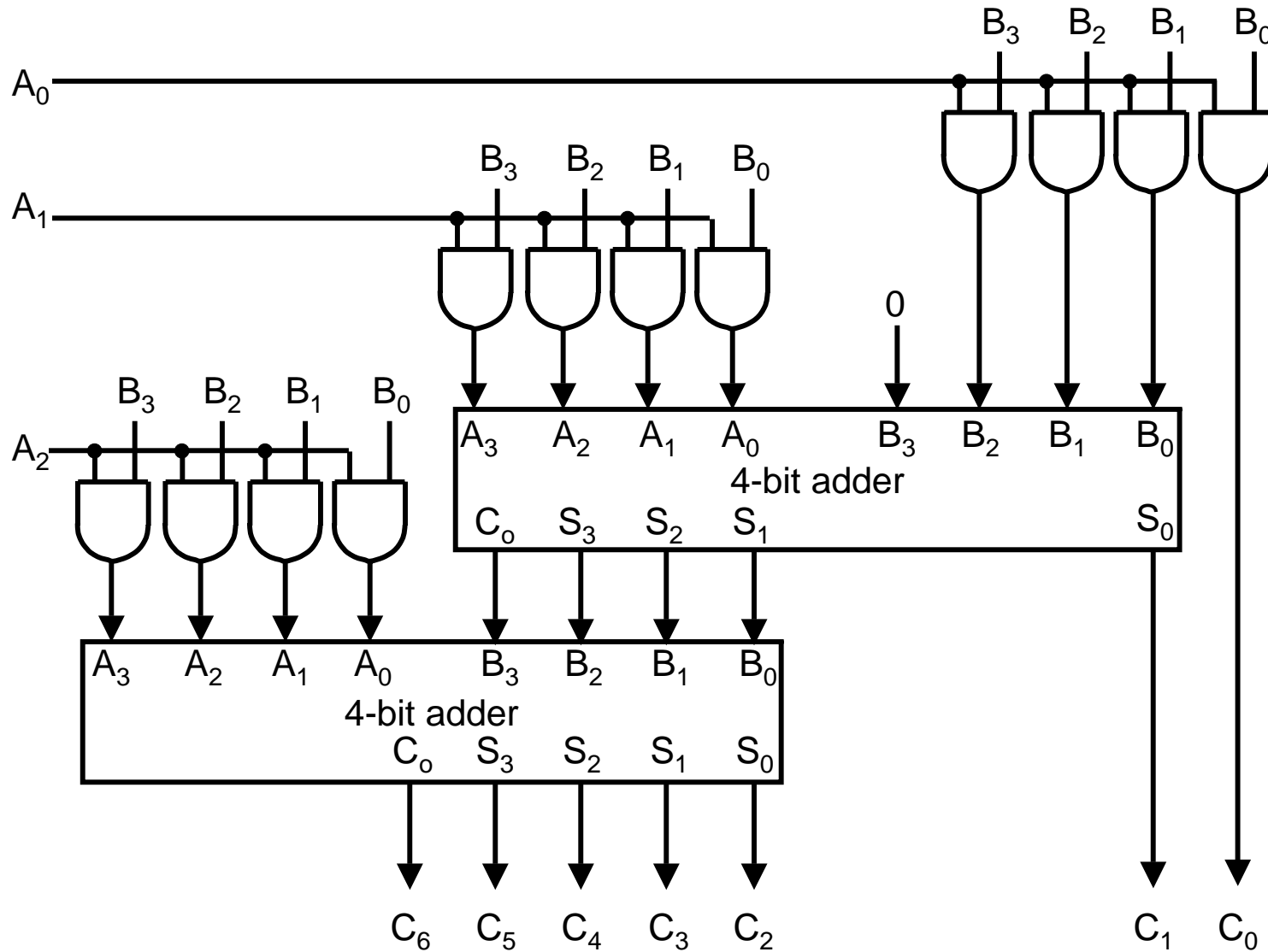
$$\begin{array}{r}
 101101 \\
 \times 11001 \\
 \hline
 101101 \\
 000000 \\
 000000 \\
 101101 \\
 \underline{101101} \\
 10001100101
 \end{array}$$

# 2-bit By 2-bit Binary Multiplication

$$\begin{array}{r}
 \phantom{x} \phantom{A_1} \phantom{A_0} B_1 \phantom{A_0} B_0 \\
 x \phantom{A_1} \phantom{A_0} \phantom{A_0} \phantom{A_0} \phantom{A_0} A_1 \phantom{A_0} A_0 \\
 \hline
 \phantom{A_1} \phantom{A_0} \phantom{A_0} \phantom{A_0} A_0 B_1 \phantom{A_0} A_0 B_0 \\
 A_1 B_1 \phantom{A_0} A_1 B_0 \\
 \hline
 C_3 \phantom{C_2} \phantom{C_1} \phantom{C_0} C_2 \phantom{C_1} C_1 \phantom{C_0} C_0
 \end{array}$$

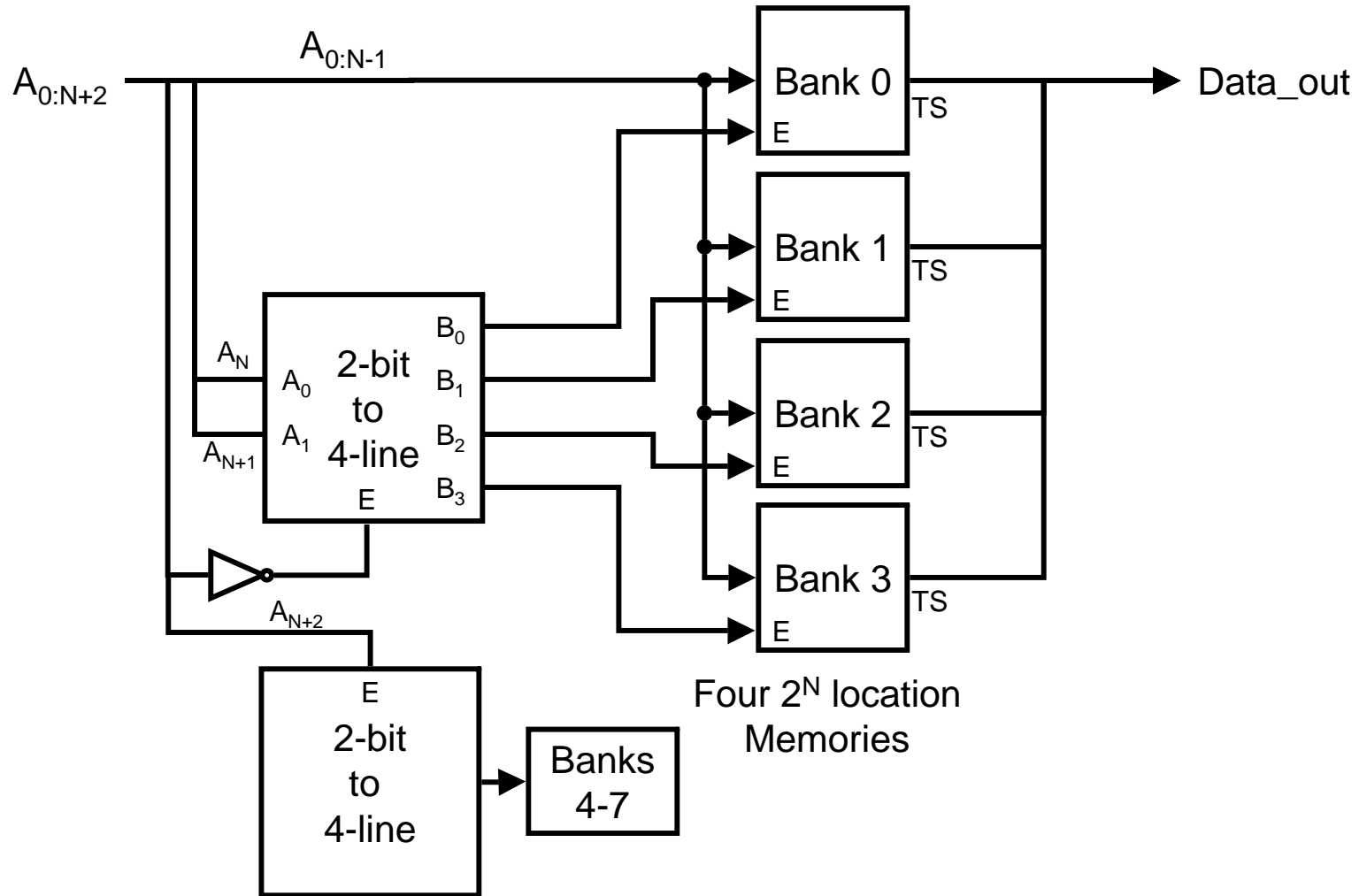


# 4-Bit By 3-Bit Binary Multiplier



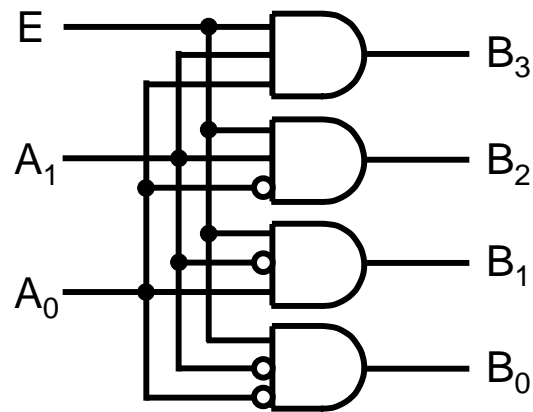
# Uses for Decoders

- Memory address expansion



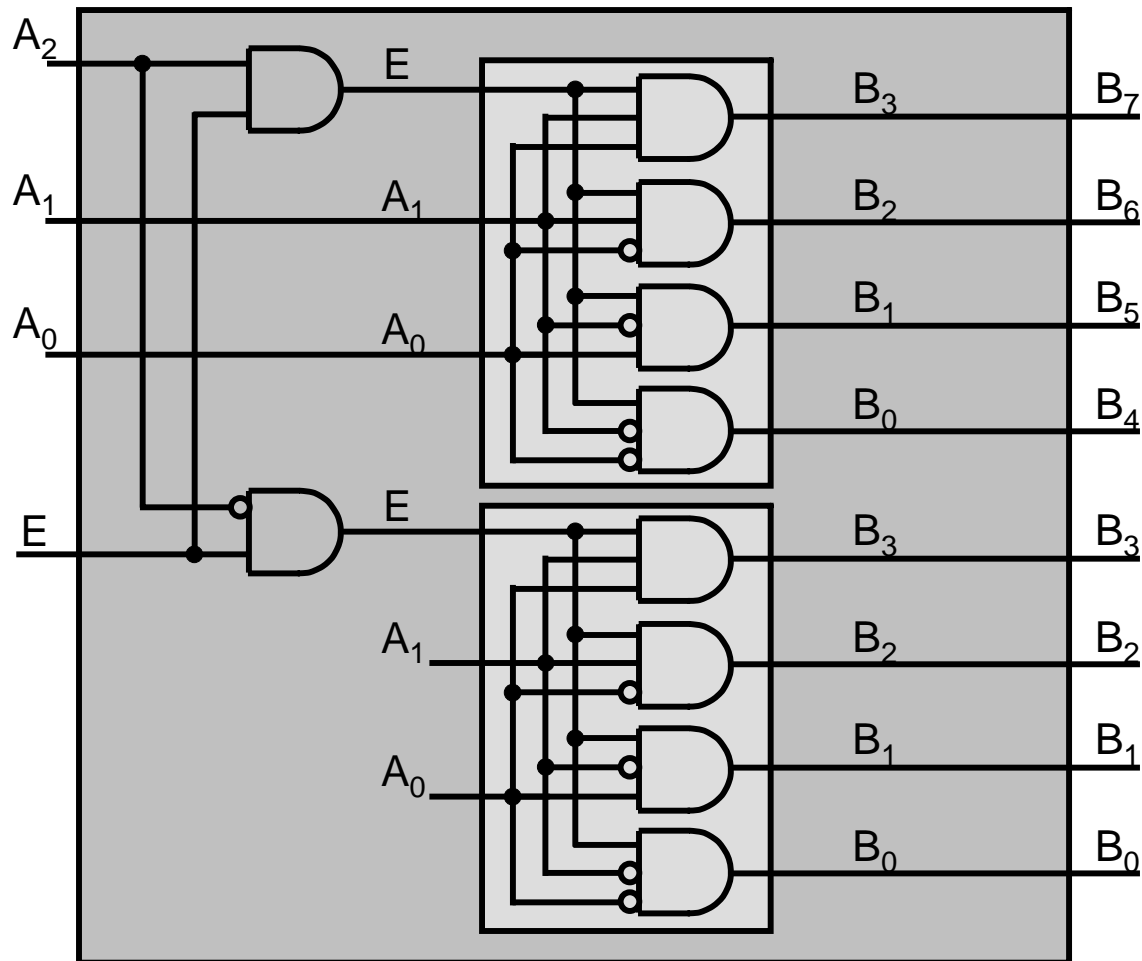
# 2-Line to 4-Line Decoder

E	A <sub>1</sub>	A <sub>0</sub>	B <sub>0</sub>	B <sub>1</sub>	B <sub>2</sub>	B <sub>3</sub>
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1



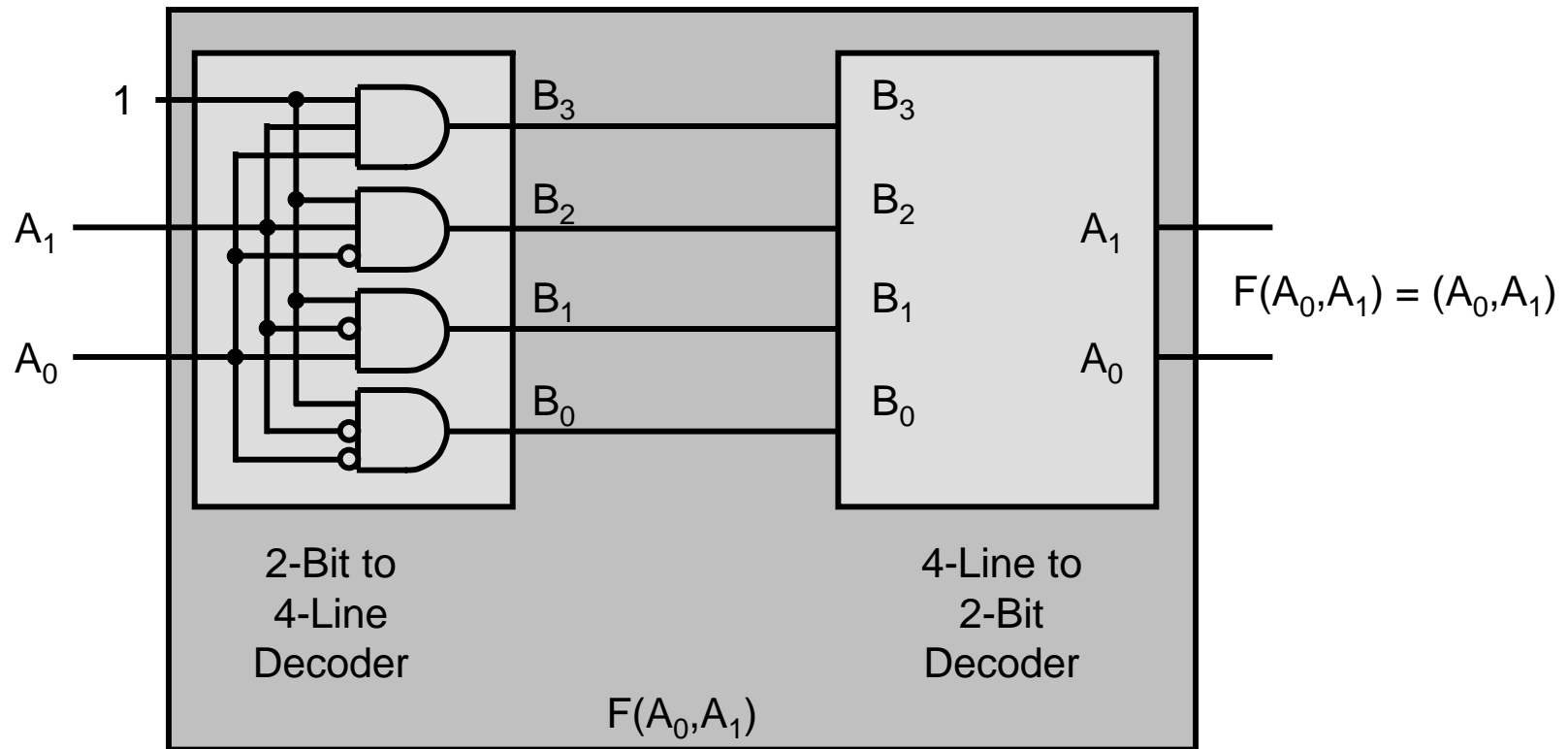
# 3-Line to 8-Line Decoder

- A  $2N$  to  $2^{2N}$  Decoder can be created from two  $N$  to  $2^N$  Decoders



# Encoder

- Encoder performs inverse operation of Decoder



# Encoder Truth Table

Inputs								Outputs		
D <sub>0</sub>	D <sub>1</sub>	D <sub>2</sub>	D <sub>3</sub>	D <sub>4</sub>	D <sub>5</sub>	D <sub>6</sub>	D <sub>7</sub>	x	y	z
<b>1</b>	0	0	0	0	0	0	0	0	0	0
0	<b>1</b>	0	0	0	0	0	0	0	0	1
0	0	<b>1</b>	0	0	0	0	0	0	1	0
0	0	0	<b>1</b>	0	0	0	0	0	1	1
0	0	0	0	<b>1</b>	0	0	0	1	0	0
0	0	0	0	0	<b>1</b>	0	0	1	0	1
0	0	0	0	0	0	<b>1</b>	0	1	1	0
0	0	0	0	0	0	0	<b>1</b>	1	1	1

$$x = D_4 + D_5 + D_6 + D_7$$

$$y = D_2 + D_3 + D_6 + D_7$$

$$z = D_1 + D_3 + D_5 + D_7$$

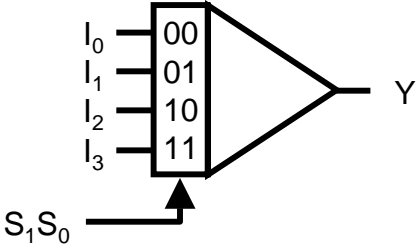
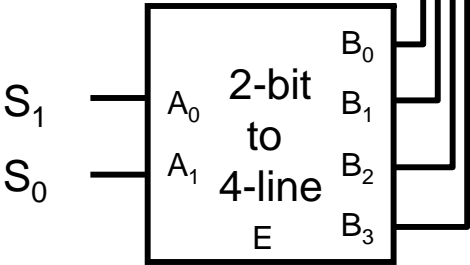
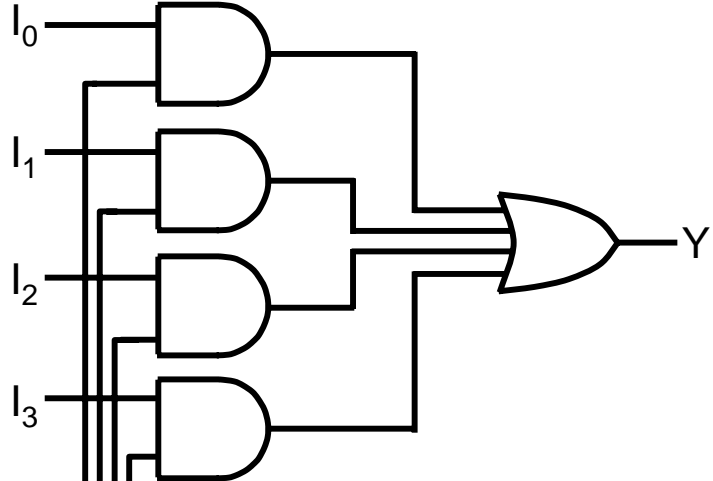
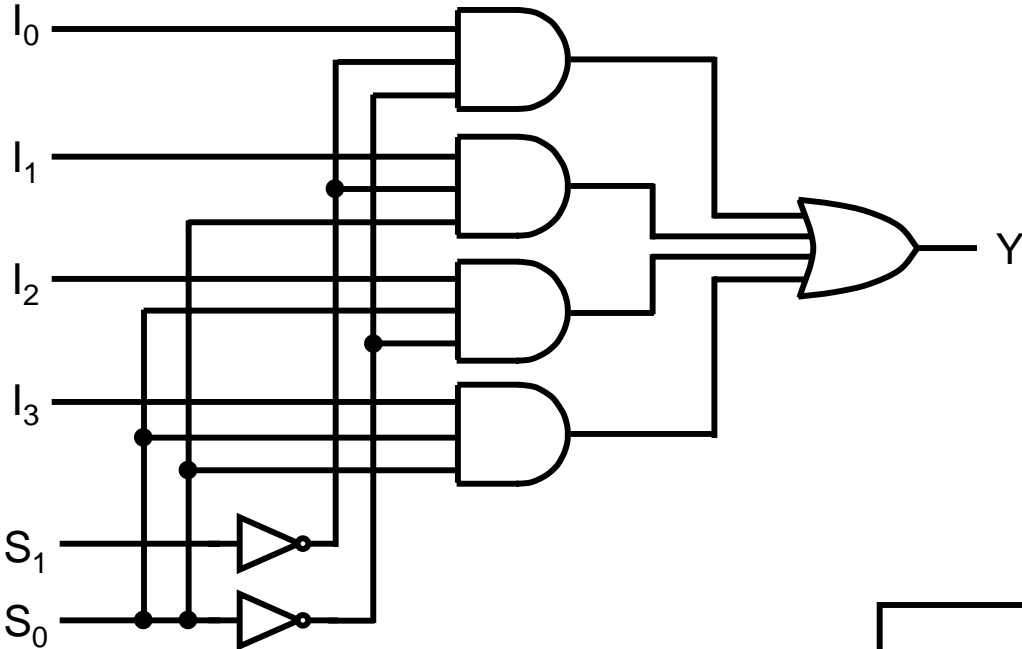
- What should output be for input (00000000)?
- What about (00100100)?

# Priority Encoder

Inputs								Outputs			
D <sub>0</sub>	D <sub>1</sub>	D <sub>2</sub>	D <sub>3</sub>	D <sub>4</sub>	D <sub>5</sub>	D <sub>6</sub>	D <sub>7</sub>	x	y	z	V
0	0	0	0	0	0	0	0	X	X	X	0
<b>1</b>	0	0	0	0	0	0	0	0	0	0	1
X	<b>1</b>	0	0	0	0	0	0	0	0	1	1
X	X	<b>1</b>	0	0	0	0	0	0	1	0	1
X	X	X	<b>1</b>	0	0	0	0	0	1	1	1
X	X	X	X	<b>1</b>	0	0	0	1	0	0	1
X	X	X	X	X	<b>1</b>	0	0	1	0	1	1
X	X	X	X	X	X	<b>1</b>	0	1	1	0	1
X	X	X	X	X	X	X	<b>1</b>	1	1	1	1

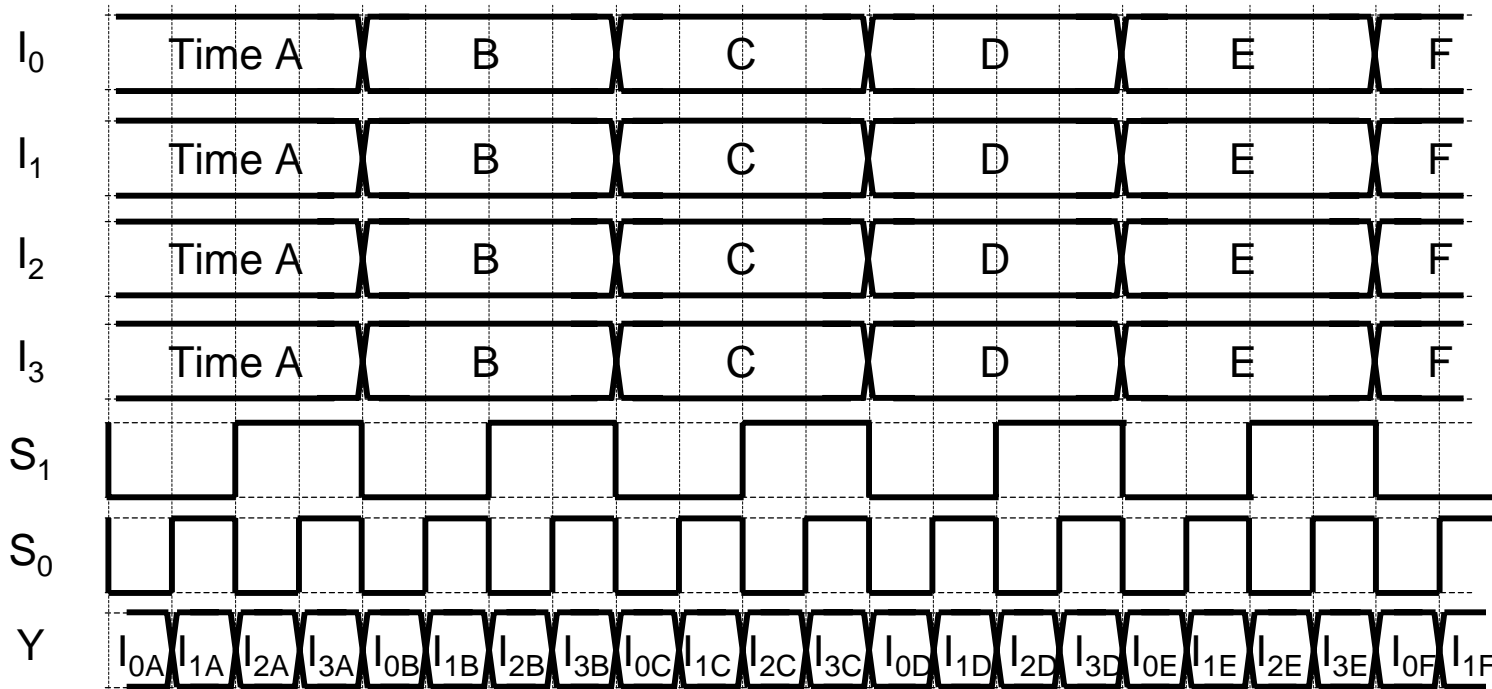
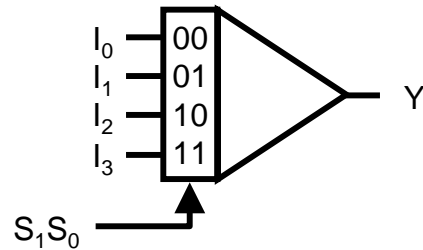
- Output encodes the largest (highest index) input that is 1.
- V indicates if there are any 1's in the input

# Multiplexer



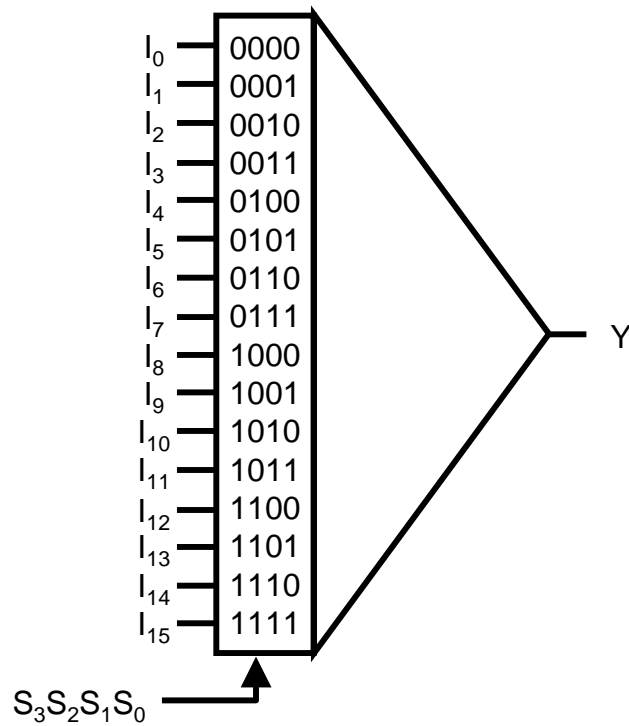
# Uses for MUXes

- Combining multiple information sources onto one channel



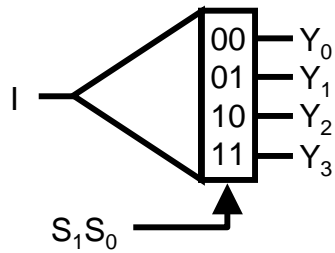
# Function Implementation with MUX

- With static values  $I[0:15]$ , this multiplexer implements the truth table shown

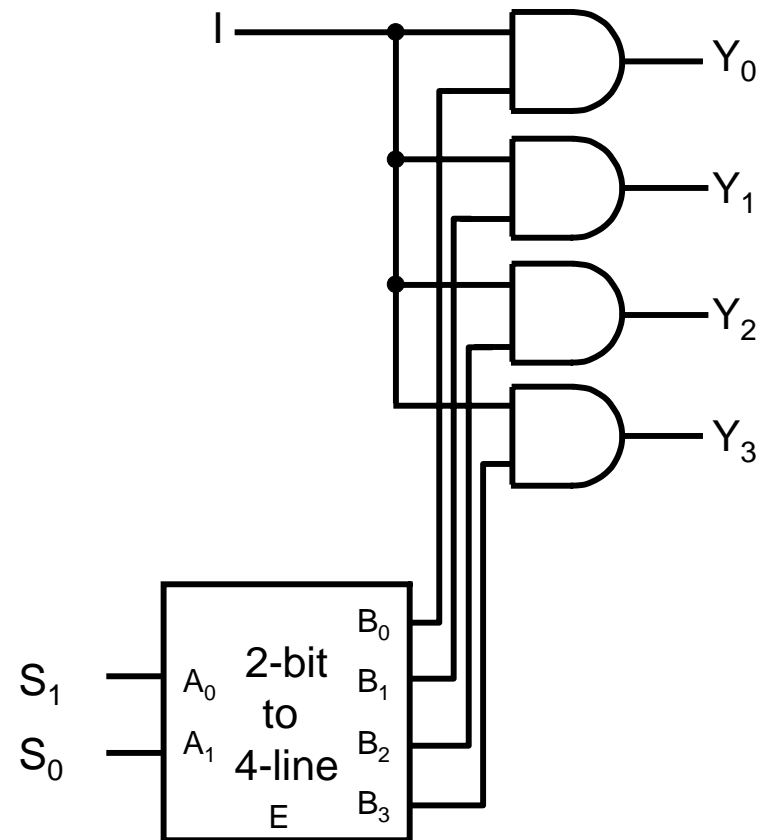


$S_3$	$S_2$	$S_1$	$S_0$	$Y$
0	0	0	0	$I_0$
0	0	0	1	$I_1$
0	0	1	0	$I_2$
0	0	1	1	$I_3$
0	1	0	0	$I_4$
0	1	0	1	$I_5$
0	1	1	0	$I_6$
0	1	1	1	$I_7$
1	0	0	0	$I_8$
1	0	0	1	$I_9$
1	0	1	0	$I_{10}$
1	0	1	1	$I_{11}$
1	1	0	0	$I_{12}$
1	1	0	1	$I_{13}$
1	1	1	0	$I_{14}$
1	1	1	1	$I_{15}$

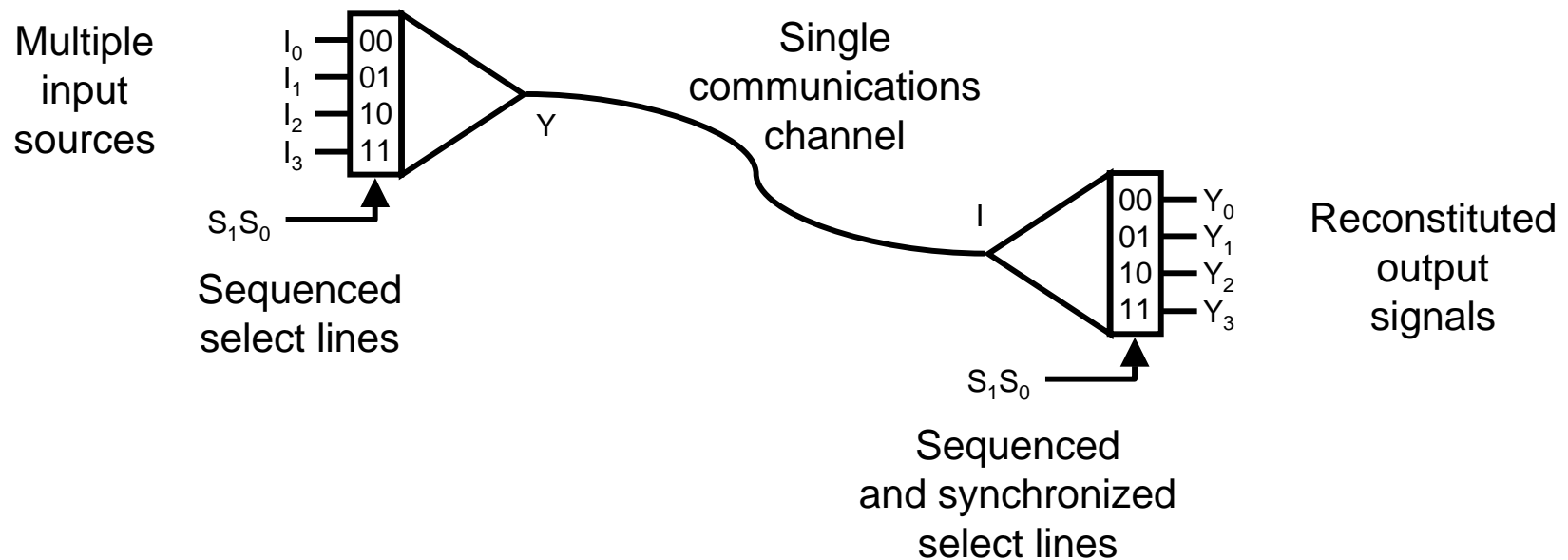
# DEMUX



- DEMUX performs inverse function of MUX



# Multiplexing and Demultiplexing Data Streams



- Most speech signals in the telephone plant is carried on T1 transmission facility:
  - 24 voice channels, each sampled at 8 kHz with 8 bits/channel + synchronization = 1.544 Mb/s
- Multiple T1's are combined to form a T3 line at ~45 Mb/s

# Summary

- Fundamental concepts of digital systems (Mano Chapter 1)
- Binary codes, number systems, and arithmetic (Ch 1)
- Boolean algebra (Ch 2)
- Simplification of switching equations (Ch 3)
- Digital device characteristics (e.g., TTL, CMOS)/design considerations (Ch 10)
- **Combinatoric logical design including LSI implementation (Chapter 4)**
- Hazards, Races, and time related issues in digital design (Ch 9)
- Flip-flops and state memory elements (Ch 5)
- Sequential logic analysis and design (Ch 5)
- Synchronous vs. asynchronous design (Ch 9)
- Counters, shift register circuits (Ch 6)
- Memory and Programmable logic (Ch 7)
- Minimization of sequential systems
- Introduction to Finite Automata

# Homework 5 – due in Class 7

- As always, show all work
- Problems 4-1, 4-5 ( $x=4, y=2, z=1$ ), ( $A=4, B=2, C=1$ ), 4-6, 4-7