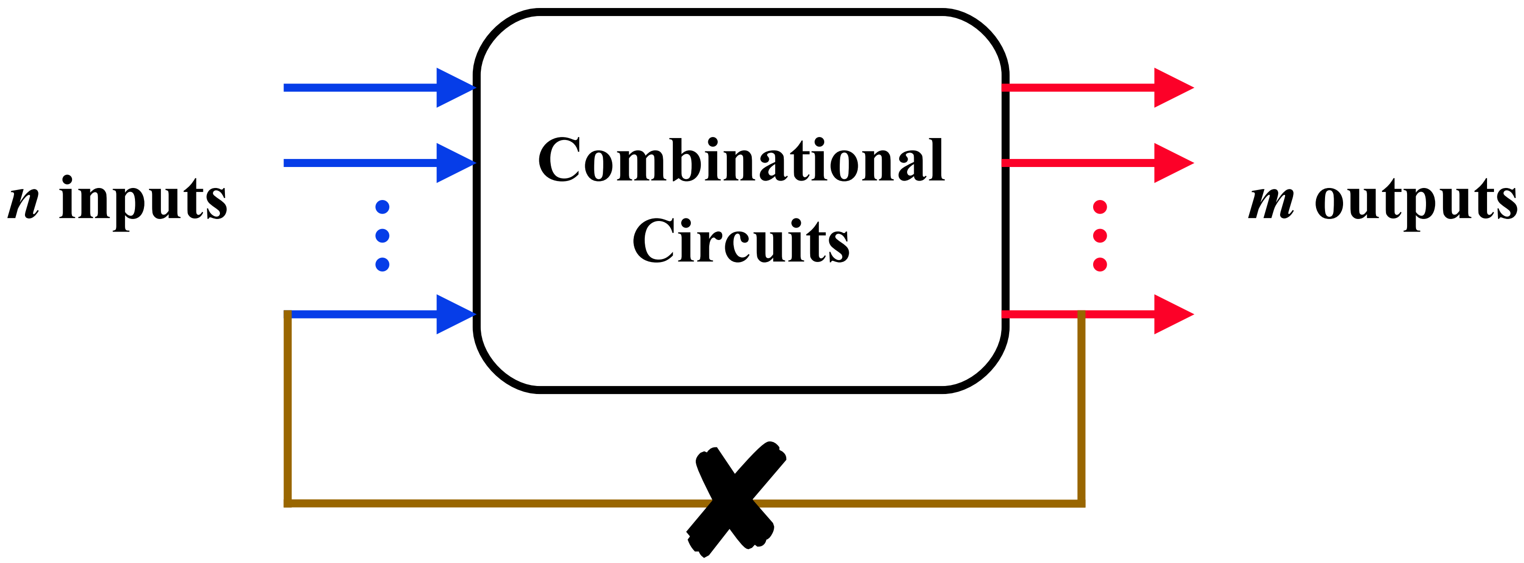
Combinational Circuit Design

Chapter 4

Combinational Circuits

★ **Output is function of input only**

i.e. no feedback

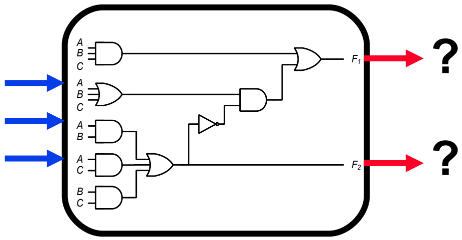


When input changes, output may change (after a delay)

Combinational Circuits

★ **Analysis**

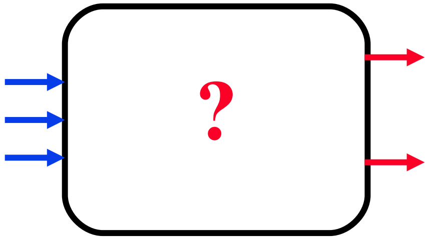
- **Given a circuit, find out its *function***



- **Function may be expressed as:**
  - ♦ **Boolean function**
  - ♦ **Truth table**

★ **Design**

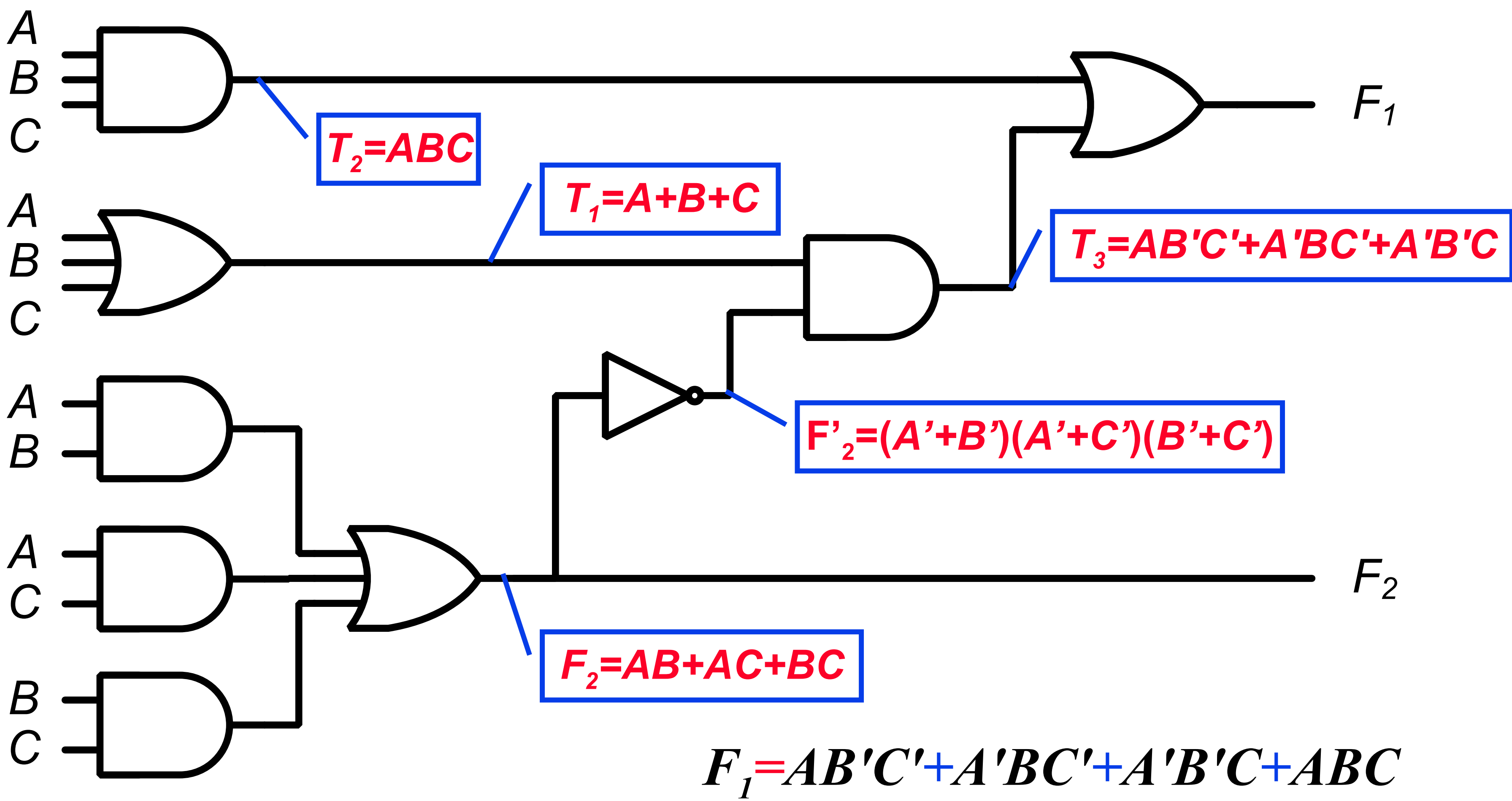
- **Given a desired function, determine its *circuit***



- **Function may be expressed as:**
  - ♦ **Boolean function**
  - ♦ **Truth table**

Analysis Procedure

★ Boolean Expression Approach

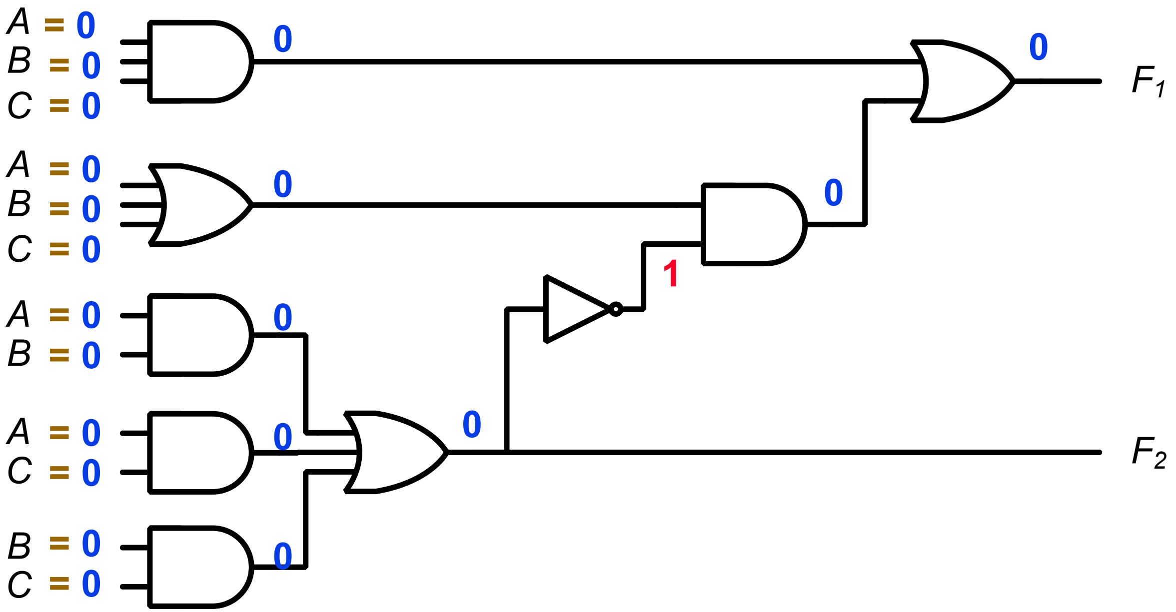


$$F_1 = AB'C' + A'BC' + A'B'C + ABC$$

$$F_2 = AB + AC + BC$$

Analysis Procedure

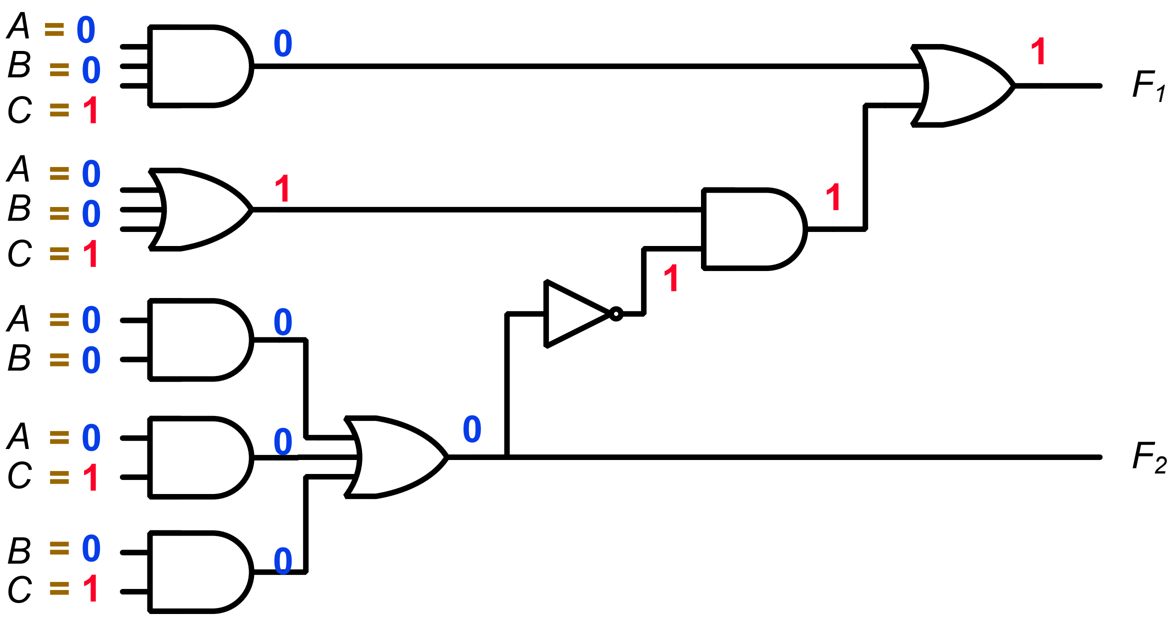
★ Truth Table Approach



A B C	F_1	F_2
0 0 0	0	0

Analysis Procedure

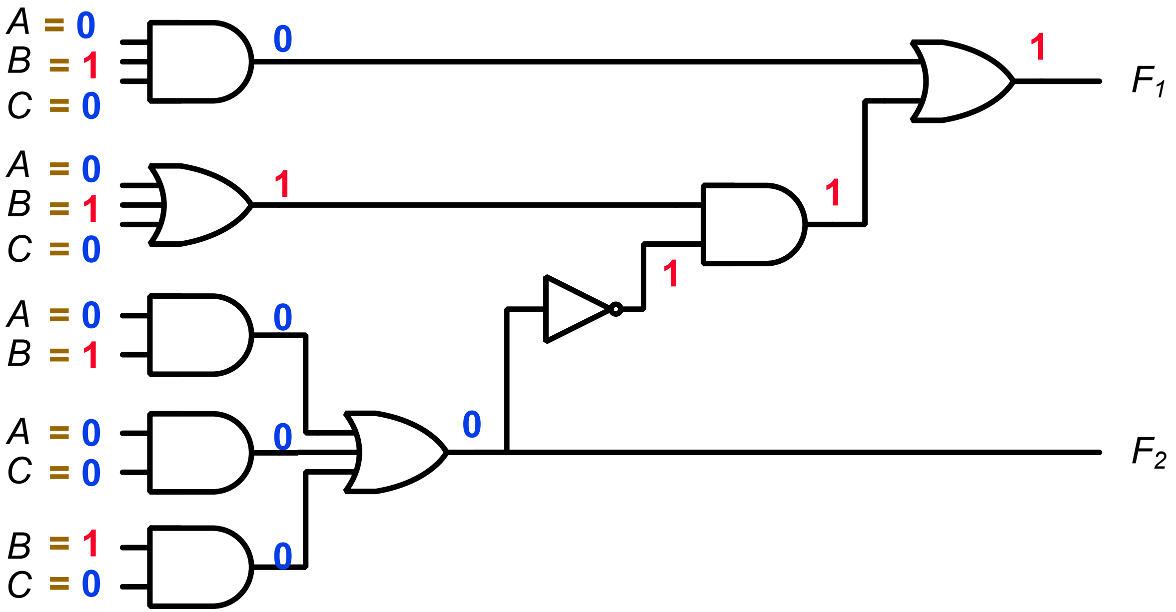
★ Truth Table Approach



A B C	F_1	F_2
0 0 0	0	0
0 0 1	1	0

Analysis Procedure

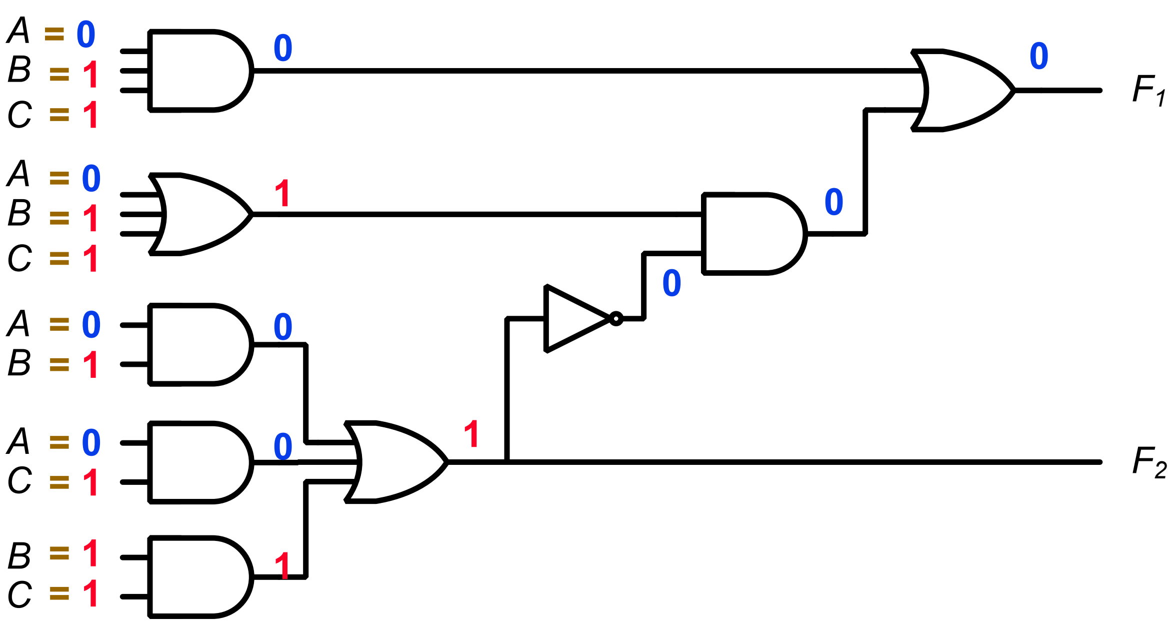
★ Truth Table Approach



A	B	C	F_1	F_2
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0

Analysis Procedure

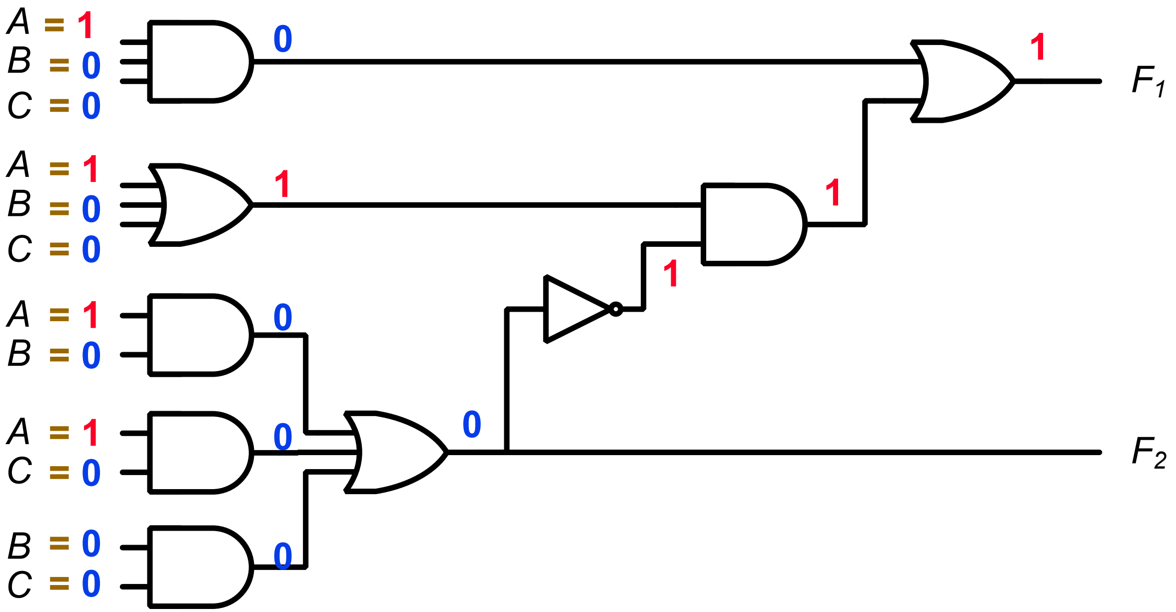
★ Truth Table Approach



A	*B*	*C*	F_1	F_2
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1

Analysis Procedure

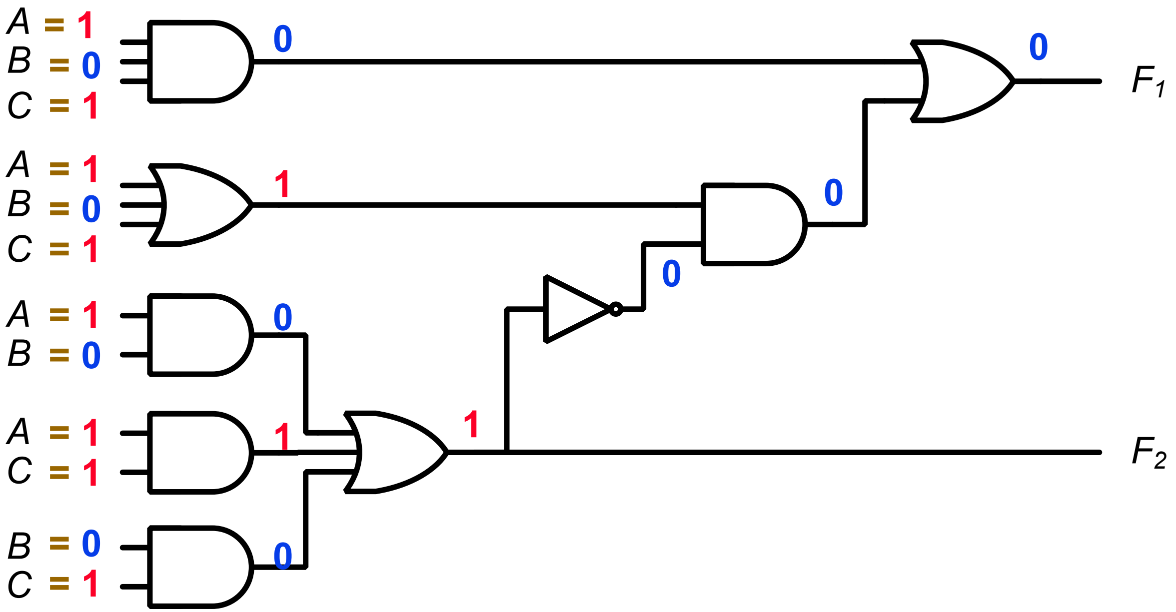
★ Truth Table Approach



A B C	F_1	F_2
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

Analysis Procedure

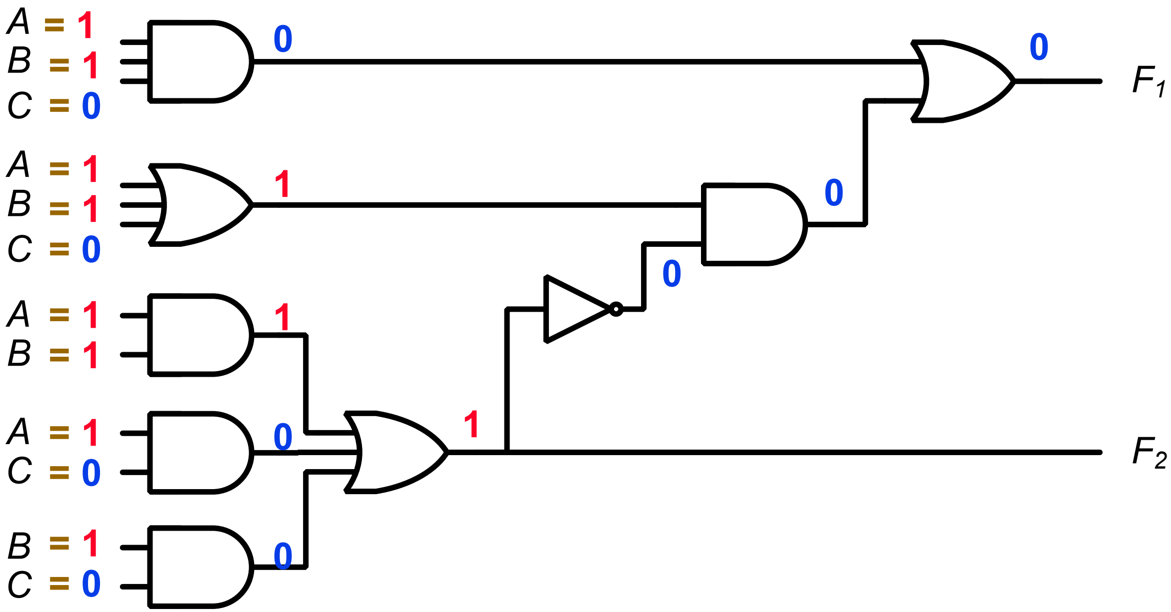
★ Truth Table Approach



A B C	F_1	F_2
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

Analysis Procedure

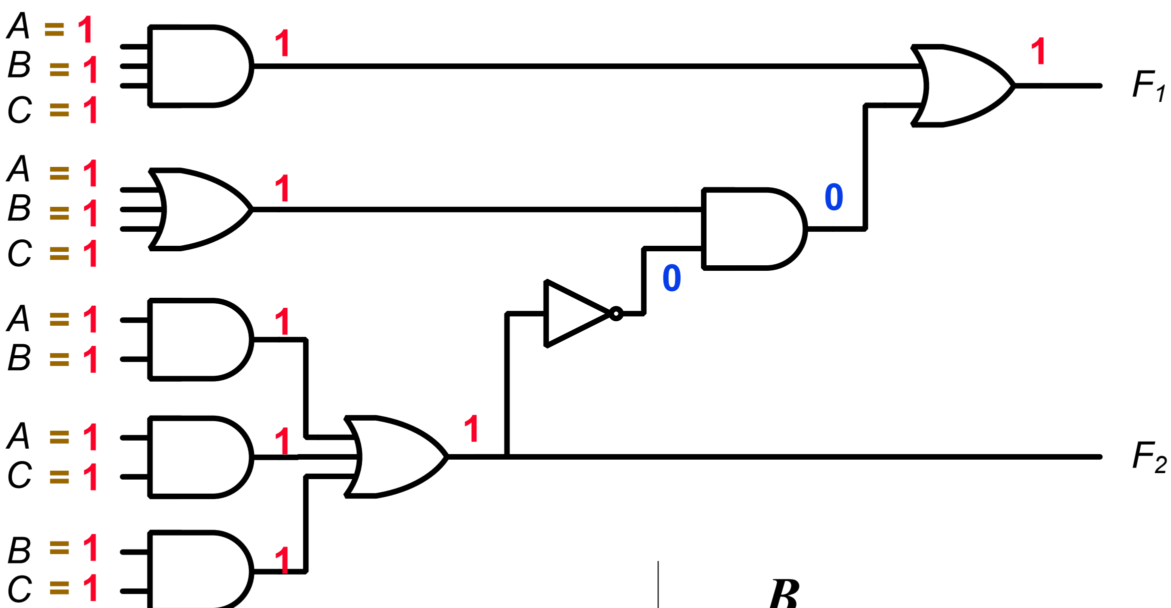
★ Truth Table Approach



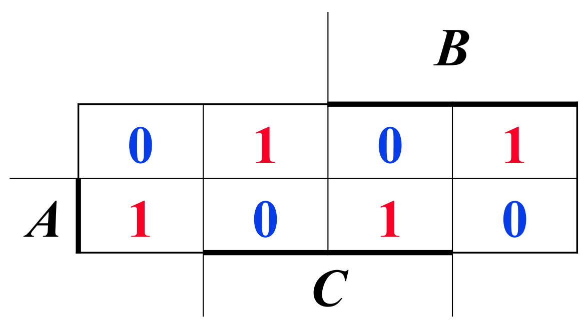
A B C	F_1	F_2
0 0 0	0	0
0 0 1	1	0
0 1 0	1	0
0 1 1	0	1
1 0 0	1	0
1 0 1	0	1
1 1 0	0	1

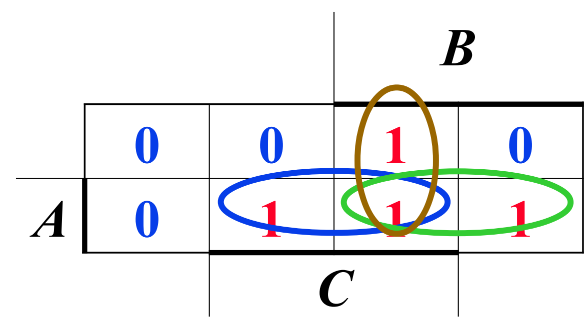
Analysis Procedure

★ Truth Table Approach



A B C	F_1	F_2
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





$F_1 = AB'C' + A'BC' + A'B'C + ABC$ $\qquad$ $F_2 = AB + AC + BC$

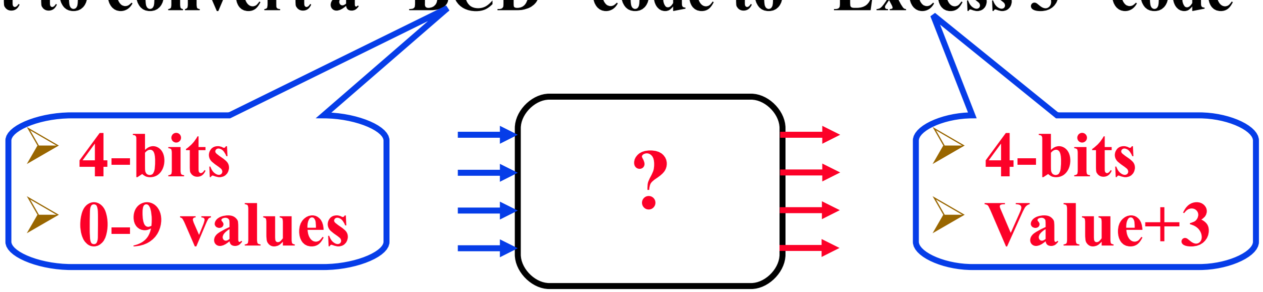
Design Procedure

★ **Given a problem statement:**

- **Determine the number of *inputs* and *outputs***
- **Derive the truth table**
- **Simplify the Boolean expression for each output**
- **Produce the required circuit**

Example:

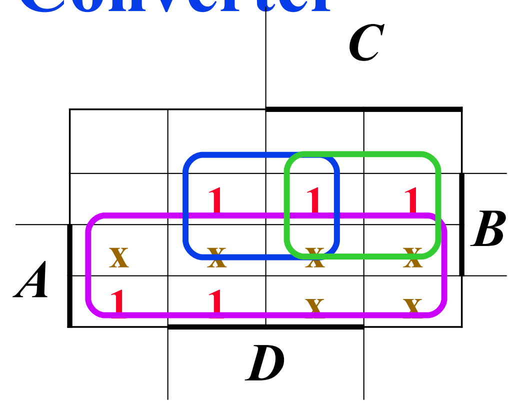
Design a circuit to convert a “BCD” code to “Excess 3” code



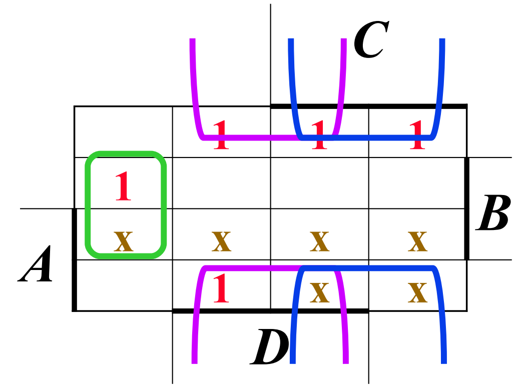
Design Procedure

★ BCD-to-Excess 3 Converter

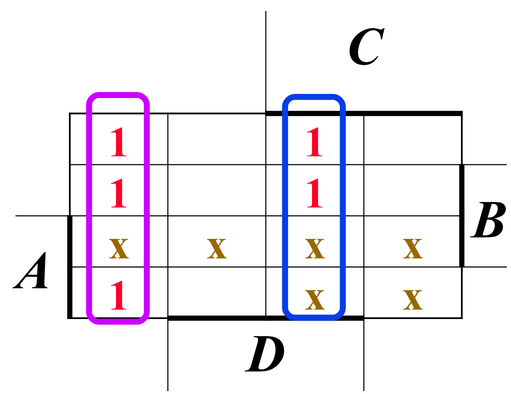
A	B	C	D	w	x	y	z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	x	x	x	x
1	0	1	1	x	x	x	x
1	1	0	0	x	x	x	x
1	1	0	1	x	x	x	x
1	1	1	0	x	x	x	x
1	1	1	1	x	x	x	x



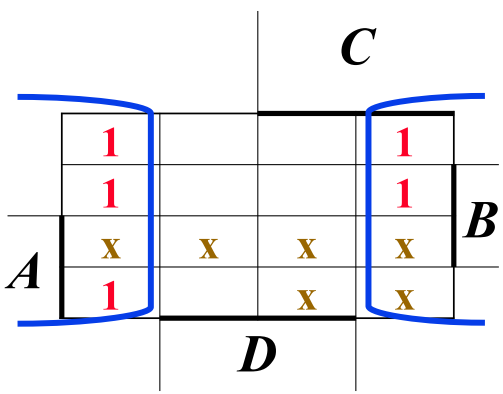
$w = A+BC+BD$



$x = B'C+B'D+BC'D'$



$y = C'D'+CD$

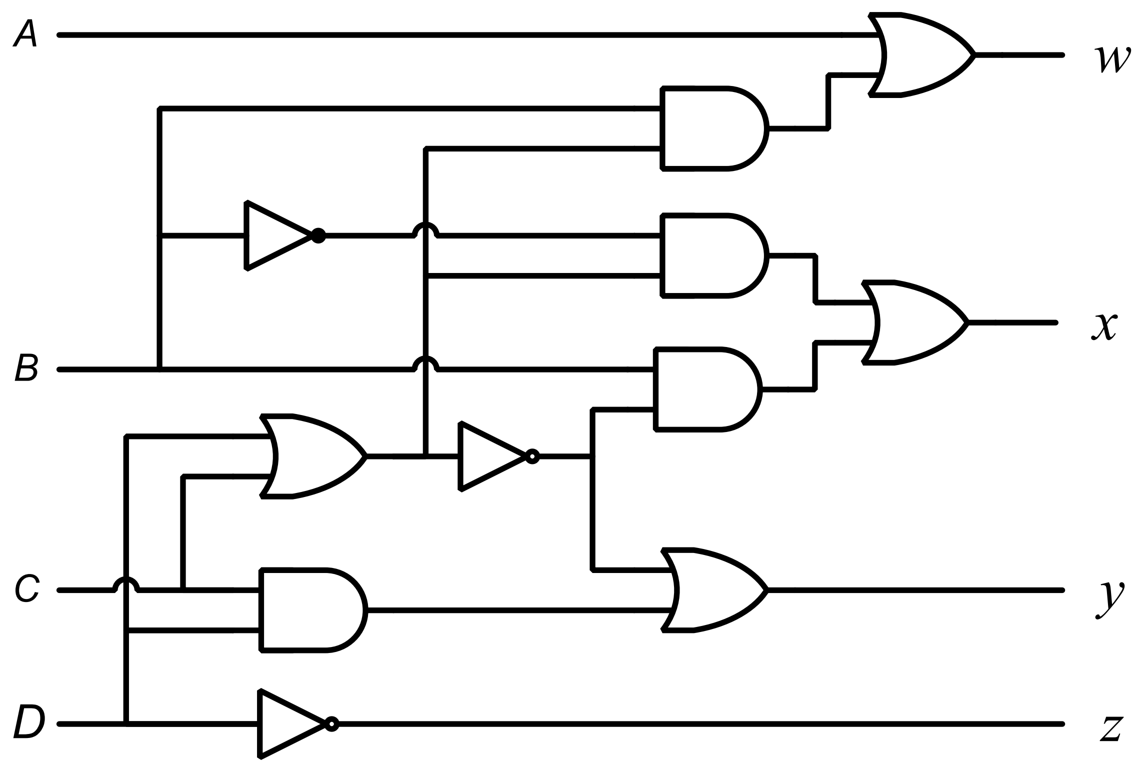


$z = D'$

Design Procedure

★ BCD-to-Excess 3 Converter

A	B	C	D	w	x	y	z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	x	x	x	x
1	0	1	1	x	x	x	x
1	1	0	0	x	x	x	x
1	1	0	1	x	x	x	x
1	1	1	0	x	x	x	x
1	1	1	1	x	x	x	x



$w = A + B(C+D)$

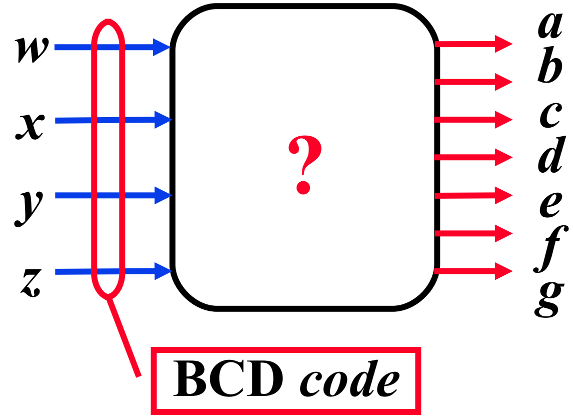
$x = B'(C+D) + B(C+D)'$

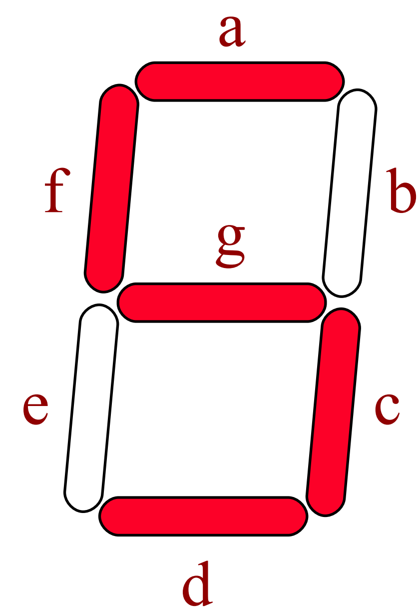
$y = (C+D)' + CD$

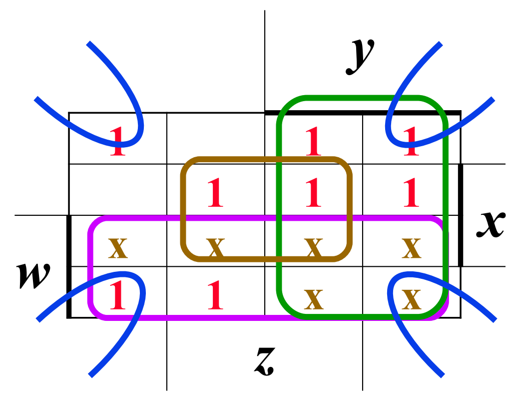
$z = D'$

Seven-Segment Decoder

w x y z	a b c d e f g
0 0 0 0	1 1 1 1 1 1 0
0 0 0 1	0 1 1 0 0 0 0
0 0 1 0	1 1 0 1 1 0 1
0 0 1 1	1 1 1 1 0 0 1
0 1 0 0	0 1 1 0 0 1 1
0 1 0 1	1 0 1 1 0 1 1
0 1 1 0	1 0 1 1 1 1 1
0 1 1 1	1 1 1 0 0 0 0
1 0 0 0	1 1 1 1 1 1 1
1 0 0 1	1 1 1 1 0 1 1
1 0 1 0	x x x x x x x
1 0 1 1	x x x x x x x
1 1 0 0	x x x x x x x
1 1 0 1	x x x x x x x
1 1 1 0	x x x x x x x
1 1 1 1	x x x x x x x







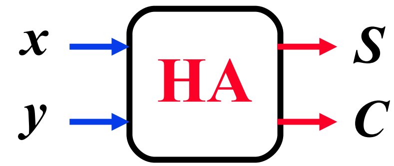
$a = w + y + xz + x'z'$

$b = \ldots$
$c = \ldots$
$d = \ldots$

Binary Adder

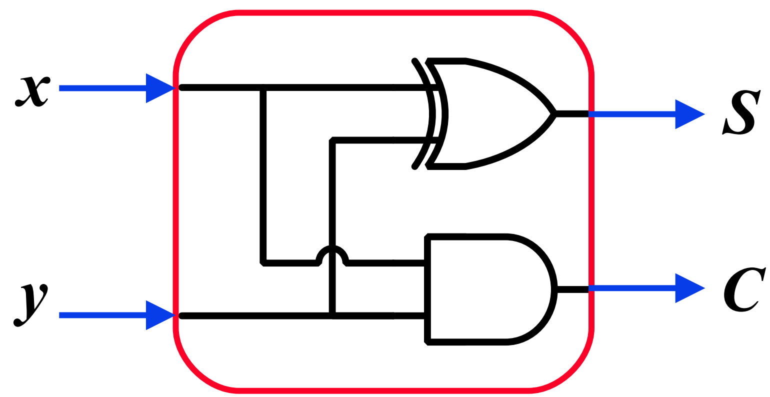
★ Half Adder

- **Adds 1-bit plus 1-bit**
- **Produces Sum and Carry**



$$\begin{array}{rr} & x \\ + & y \\ \hline C & S \end{array}$$

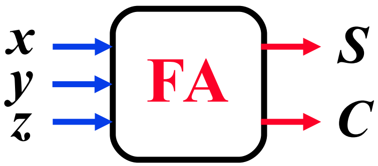
x y	C S
0 0	0 0
0 1	0 1
1 0	0 1
1 1	1 0



Binary Adder

★ Full Adder

- Adds 1-bit plus 1-bit plus 1-bit
- Produces Sum and Carry



$$\begin{array}{r} x \\ +\ y \\ +\ z \\ \hline C\ S \end{array}$$

x	y	z	C	S
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

			y	y
	0	1	0	1
x	1	0	1	0
		z	z	

$$S = xy'z' + x'yz' + x'y'z + xyz = x \oplus y \oplus z$$

			y	y
	0	0	1	0
x	0	1	1	1
		z	z	

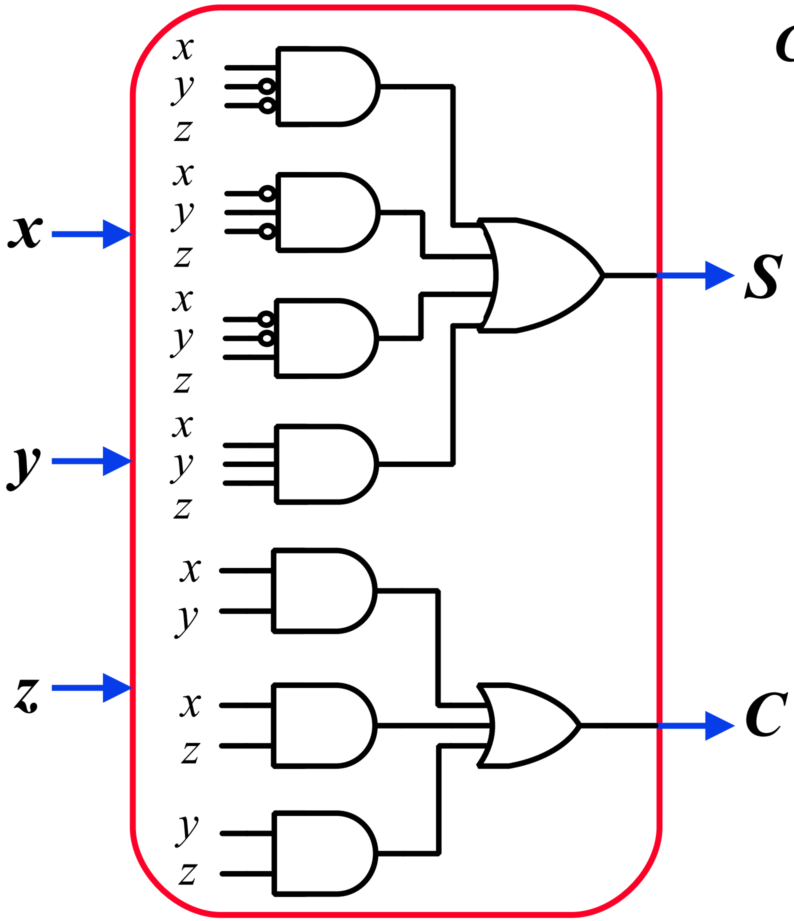
$$C = xy + xz + yz$$

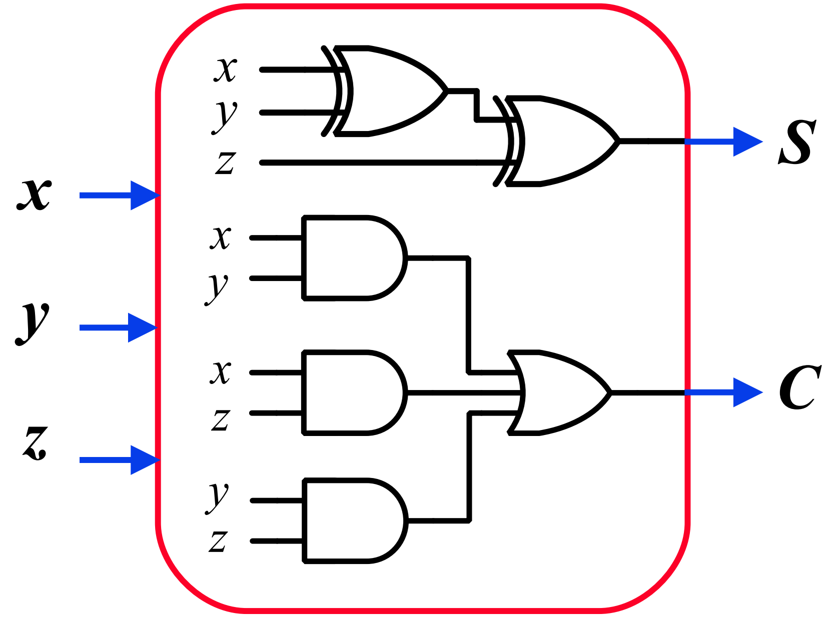
Binary Adder

★ Full Adder

$$S = xy'z' + x'yz' + x'y'z + xyz = x \oplus y \oplus z$$

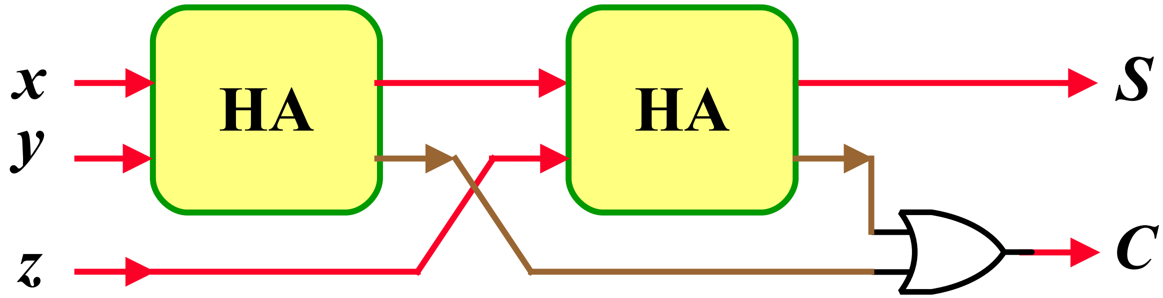
$$C = xy + xz + yz$$

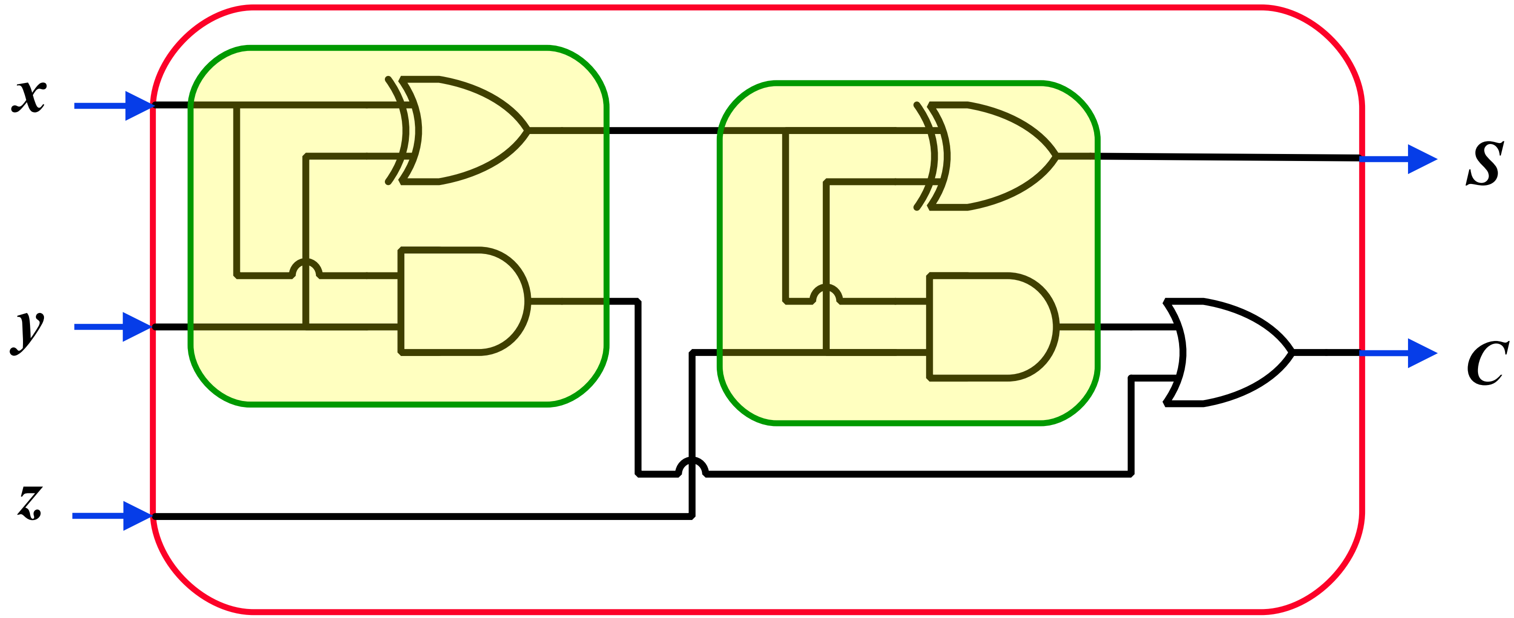




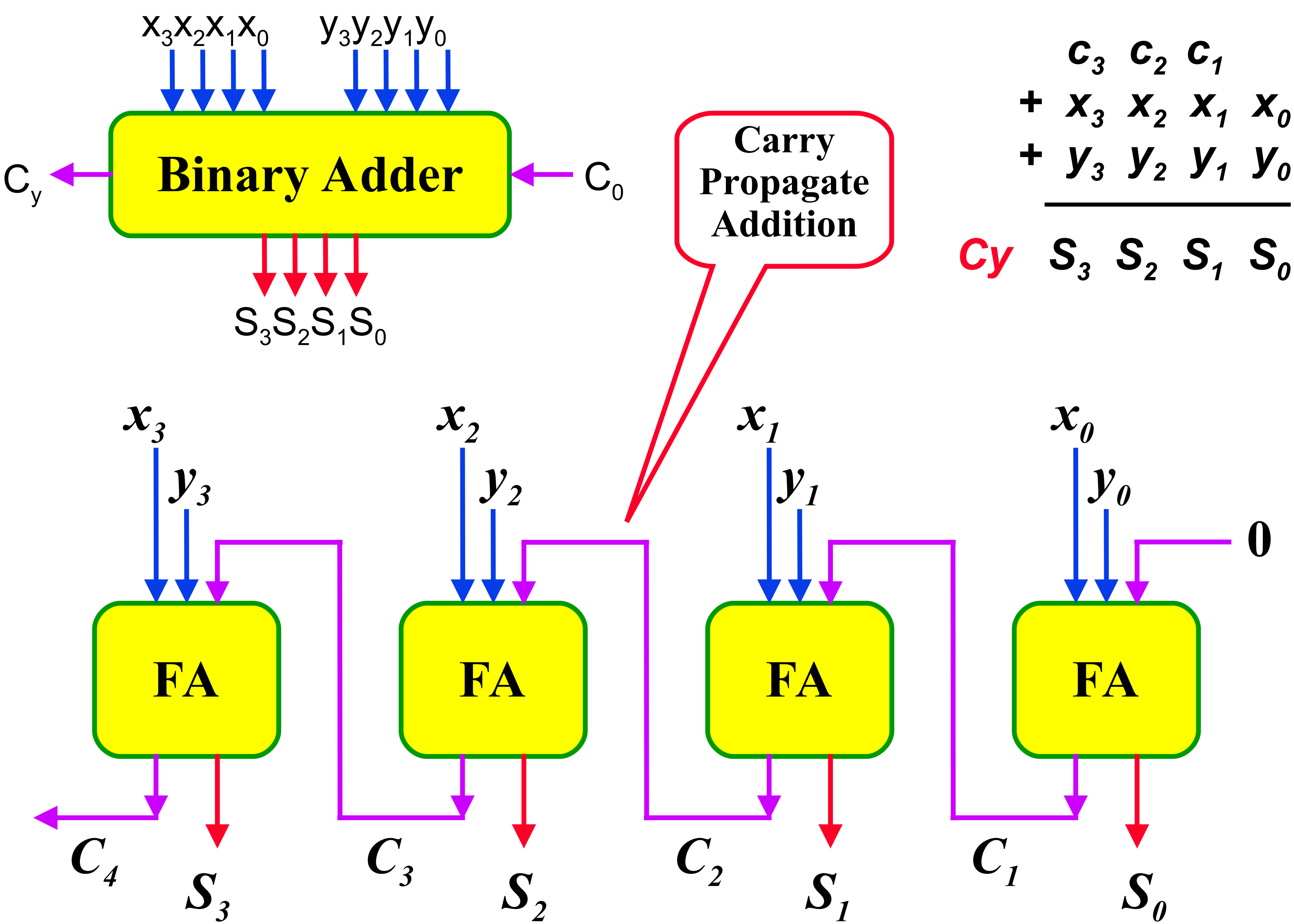
Binary Adder

★ Full Adder



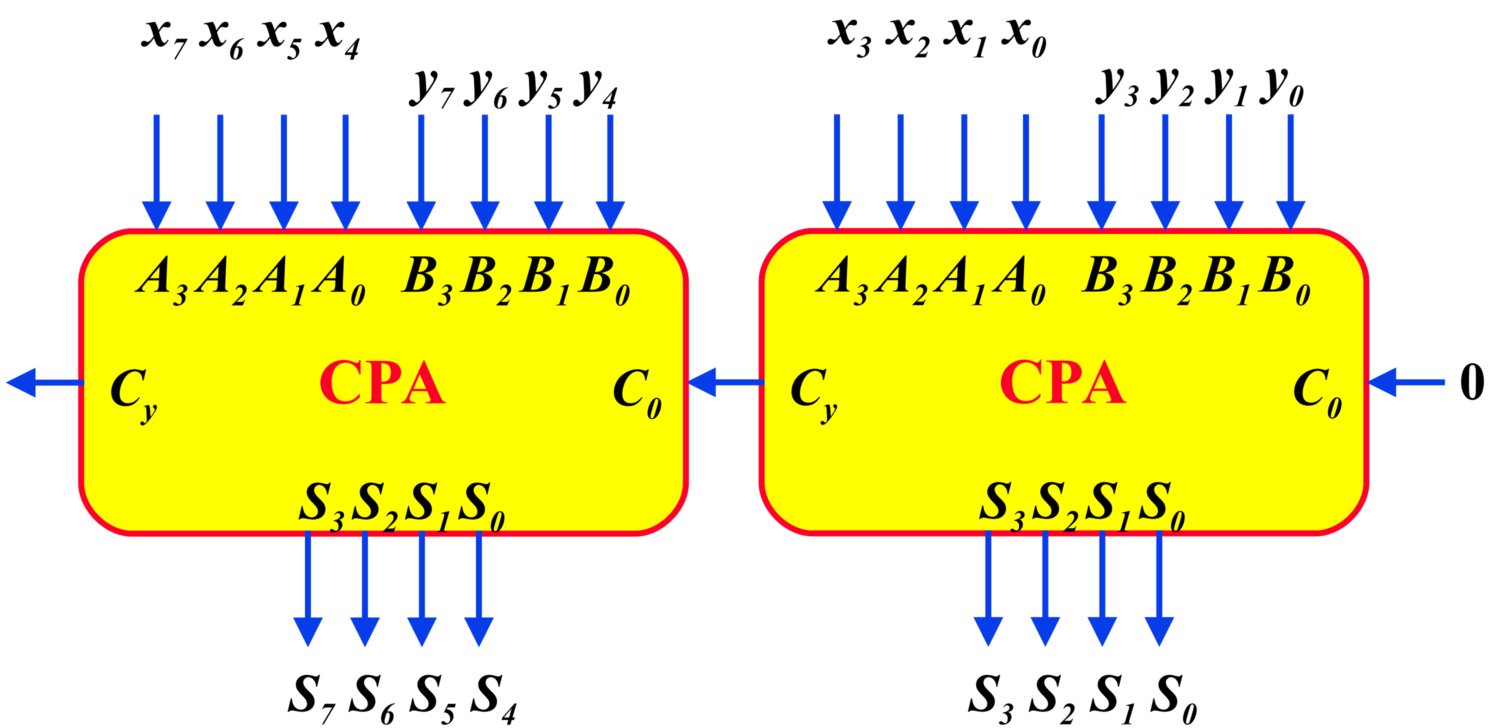


Binary Adder



Binary Adder

★ **Carry Propagate Adder**



★ Carry propagation

- **When the correct outputs are available**
- **The critical path counts (the worst case)**
- $(A_1, B_1, C_1) \rightarrow C_2 \rightarrow C_3 \rightarrow C_4 \rightarrow (C_5, S_4)$
- **When 4-bits full-adder → 8 gate levels (*n*-bits: 2*n* gate levels)**

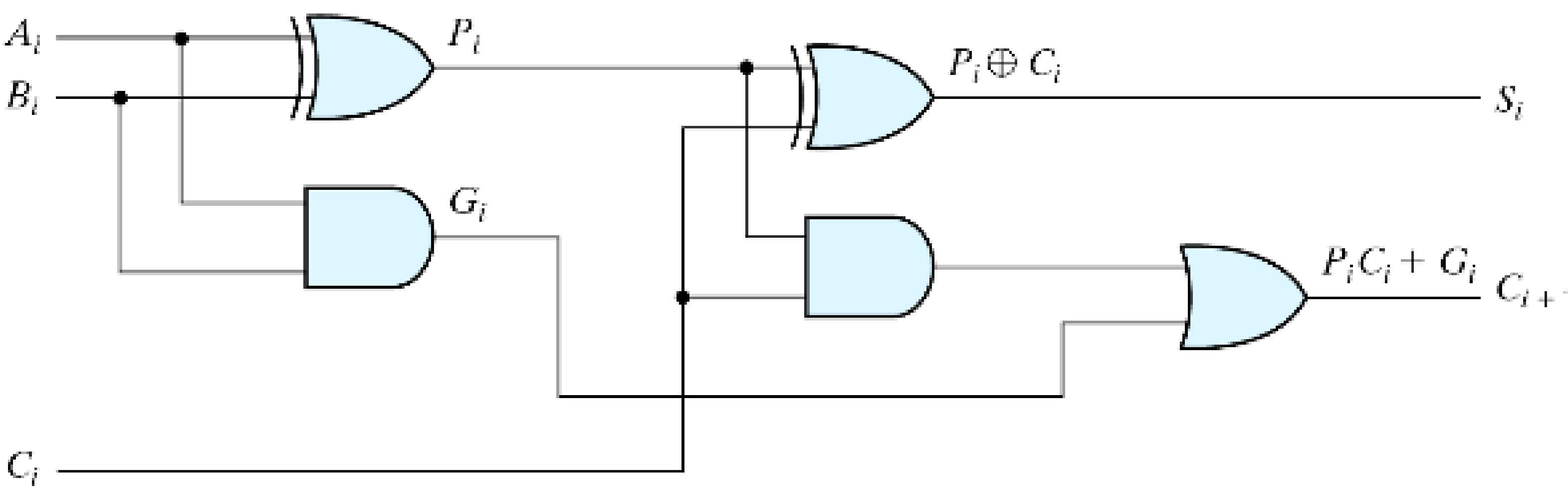
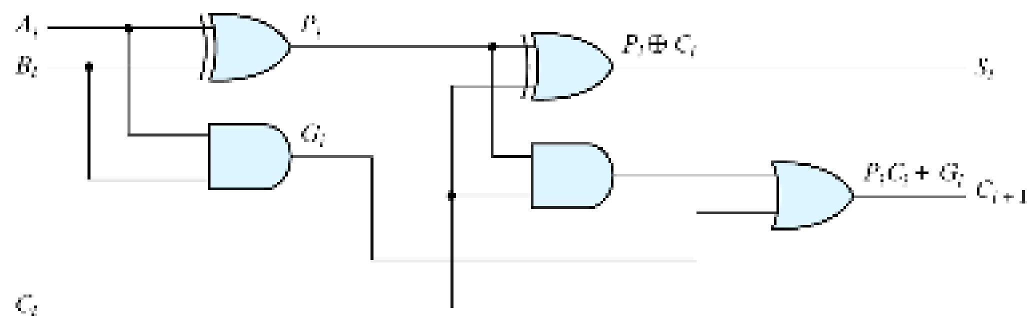


Figure 4.10 Full Adder with *P* and *G* Shown

Parallel Adders

★ **Reduce the carry propagation delay**

- **Employ faster gates**
- **Look-ahead carry (more complex mechanism, yet faster)**
- **Carry propagate:** $P_i = A_i \oplus B_i$
- **Carry generate:** $G_i = A_i B_i$
- **Sum:** $S_i = P_i \oplus C_i$
- **Carry:** $C_{i+1} = G_i + P_i C_i$
- C_0 **= Input carry**
- $C_1 = G_0 + P_0 C_0$
- $C_2 = G_1 + P_1 C_1 = G_1 + P_1(G_0 + P_0 C_0) = G_1 + P_1 G_0 + P_1 P_0 C_0$
- $C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$



Carry Look-ahead Adder (1/2)

★ **Logic diagram**

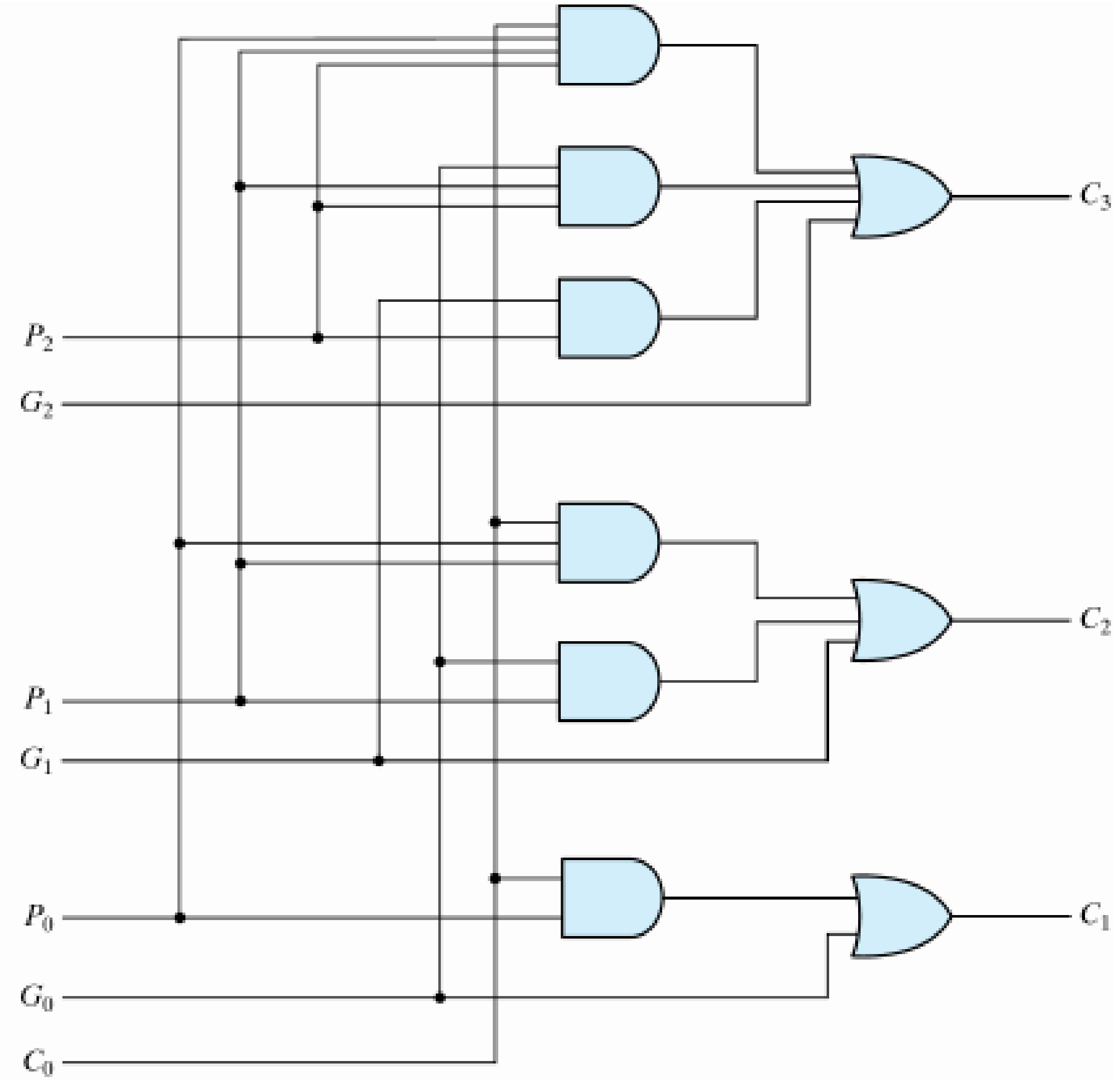


Fig. 4.11 Logic Diagram of Carry Look-ahead Generator

Carry Look-ahead Adder (2/2)

- **4-bit carry-look ahead adder**
  - **Propagation delay of C_3, C_2 and C_1 are equal.**

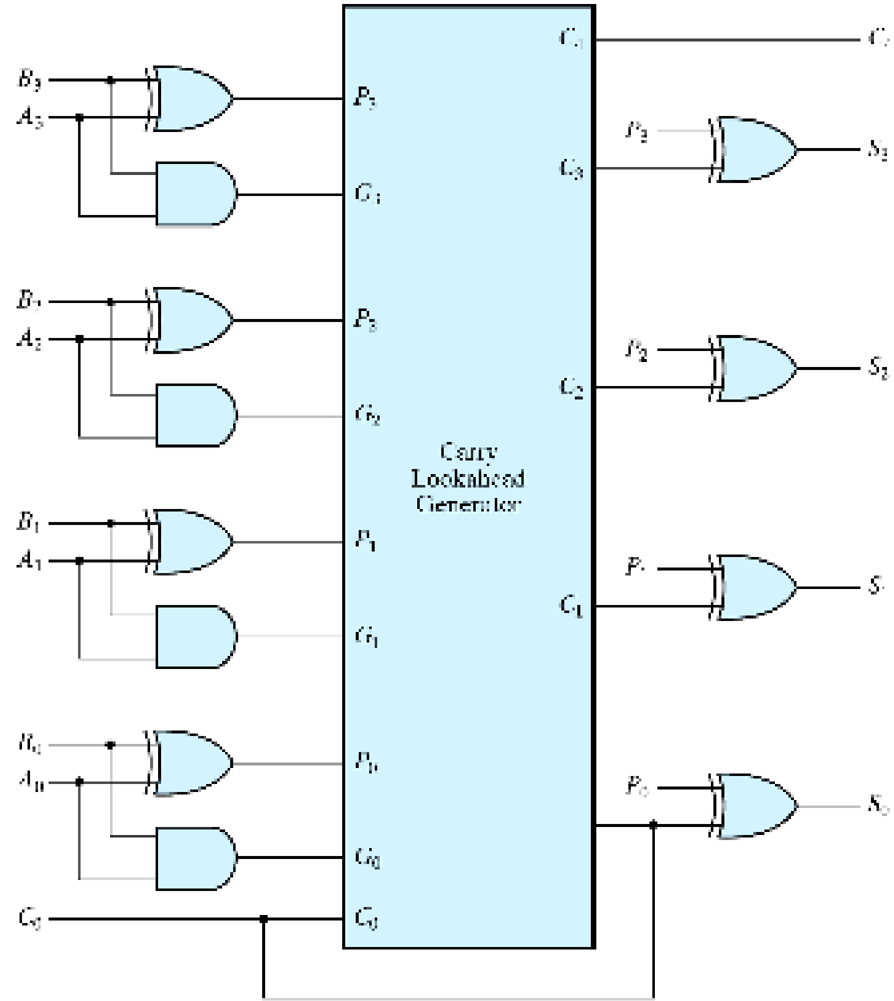


Fig. 4.12 4-Bit Adder with Carry Look-ahead

BCD Adder

- 4-bits plus 4-bits
- Operands and Result: 0 to 9

$$
\begin{array}{rcccc}
+ & x_3 & x_2 & x_1 & x_0 \\
+ & y_3 & y_2 & y_1 & y_0 \\
\hline
Cy & S_3 & S_2 & S_1 & S_0
\end{array}
$$

$X + Y$	$x_3\ x_2\ x_1\ x_0$	$y_3\ y_2\ y_1\ y_0$	Sum	Cy	$S_3\ S_2\ S_1\ S_0$
0 + 0	0 0 0 0	0 0 0 0	= 0	0	0 0 0 0
0 + 1	0 0 0 0	0 0 0 1	= 1	0	0 0 0 1
0 + 2	0 0 0 0	0 0 1 0	= 2	0	0 0 1 0
0 + 9	0 0 0 0	1 0 0 1	= 9	0	1 0 0 1
1 + 0	0 0 0 1	0 0 0 0	= 1	0	0 0 0 1
1 + 1	0 0 0 1	0 0 0 1	= 2	0	0 0 1 0
1 + 8	0 0 0 1	1 0 0 0	= 9	0	1 0 0 1
1 + 9	0 0 0 1	1 0 0 1	= A	0	1 0 1 0
2 + 0	0 0 1 0	0 0 0 0	= 2	0	0 0 1 0
9 + 9	1 0 0 1	1 0 0 1	= 12	1	0 0 1 0

Invalid Code

Wrong BCD Value

0001 1000

BCD Adder

X + Y	$x_3\, x_2\, x_1\, x_0$	$y_3\, y_2\, y_1\, y_0$	*Sum*	*Cy*	$S_3\, S_2\, S_1\, S_0$	*Required BCD Output*	*Value*	
9 + 0	1 0 0 1	0 0 0 0	= 9	0	1 0 0 1	0 0 0 0 1 0 0 1	= 9	
9 + 1	1 0 0 1	0 0 0 1	= 10	0	1 0 1 0	0 0 0 1 0 0 0 0	= 16	✘
9 + 2	1 0 0 1	0 0 1 0	= 11	0	1 0 1 1	0 0 0 1 0 0 0 1	= 17	✘
9 + 3	1 0 0 1	0 0 1 1	= 12	0	1 1 0 0	0 0 0 1 0 0 1 0	= 18	✘
9 + 4	1 0 0 1	0 1 0 0	= 13	0	1 1 0 1	0 0 0 1 0 0 1 1	= 19	✘
9 + 5	1 0 0 1	0 1 0 1	= 14	0	1 1 1 0	0 0 0 1 0 1 0 0	= 20	✘
9 + 6	1 0 0 1	0 1 1 0	= 15	0	1 1 1 1	0 0 0 1 0 1 0 1	= 21	✘
9 + 7	1 0 0 1	0 1 1 1	= 16	1	0 0 0 0	0 0 0 1 0 1 1 0	= 22	✘
9 + 8	1 0 0 1	1 0 0 0	= 17	1	0 0 0 1	0 0 0 1 0 1 1 1	= 23	✘
9 + 9	1 0 0 1	1 0 0 1	= 18	1	0 0 1 0	0 0 0 1 1 0 0 0	= 24	✘

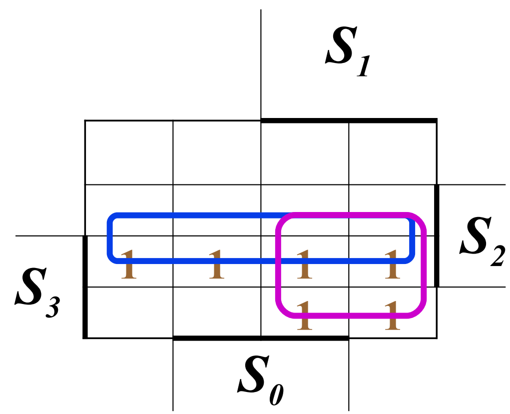
+ 6

BCD Adder

★ **Correct Binary Adder's Output (+6)**

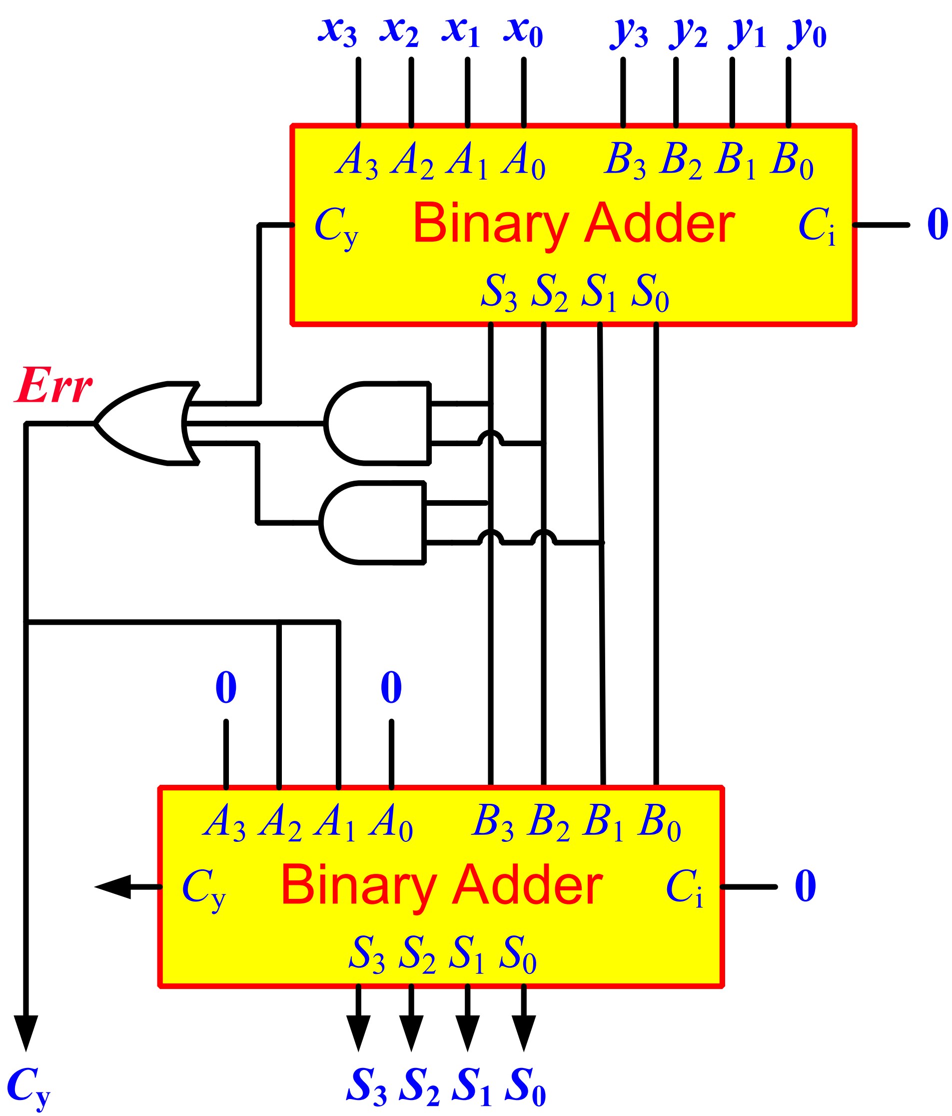
- **If the result is between 'A' and 'F'**
- **If Cy = 1**

$S_3\ S_2\ S_1\ S_0$	Err
0 0 0 0	0
1 0 0 0	0
1 0 0 1	0
1 0 1 0	1
1 0 1 1	1
1 1 0 0	1
1 1 0 1	1
1 1 1 0	1
1 1 1 1	1



$$Err = S_3 S_2 + S_3 S_1$$

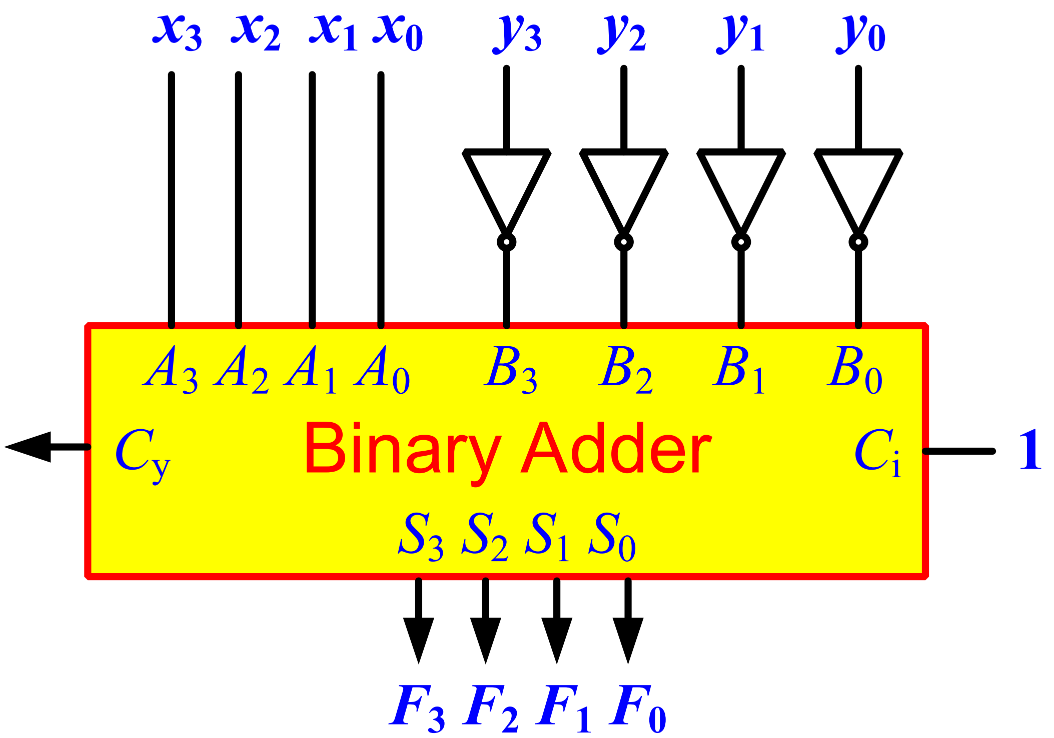
BCD Adder



x_3 x_2 x_1 x_0 y_3 y_2 y_1 y_0
A_3 A_2 A_1 A_0 B_3 B_2 B_1 B_0
C_y Binary Adder C_i
0
S_3 S_2 S_1 S_0
Err
0 0
A_3 A_2 A_1 A_0 B_3 B_2 B_1 B_0
C_y Binary Adder C_i
0
S_3 S_2 S_1 S_0
C_y
S_3 S_2 S_1 S_0

Binary Subtractor

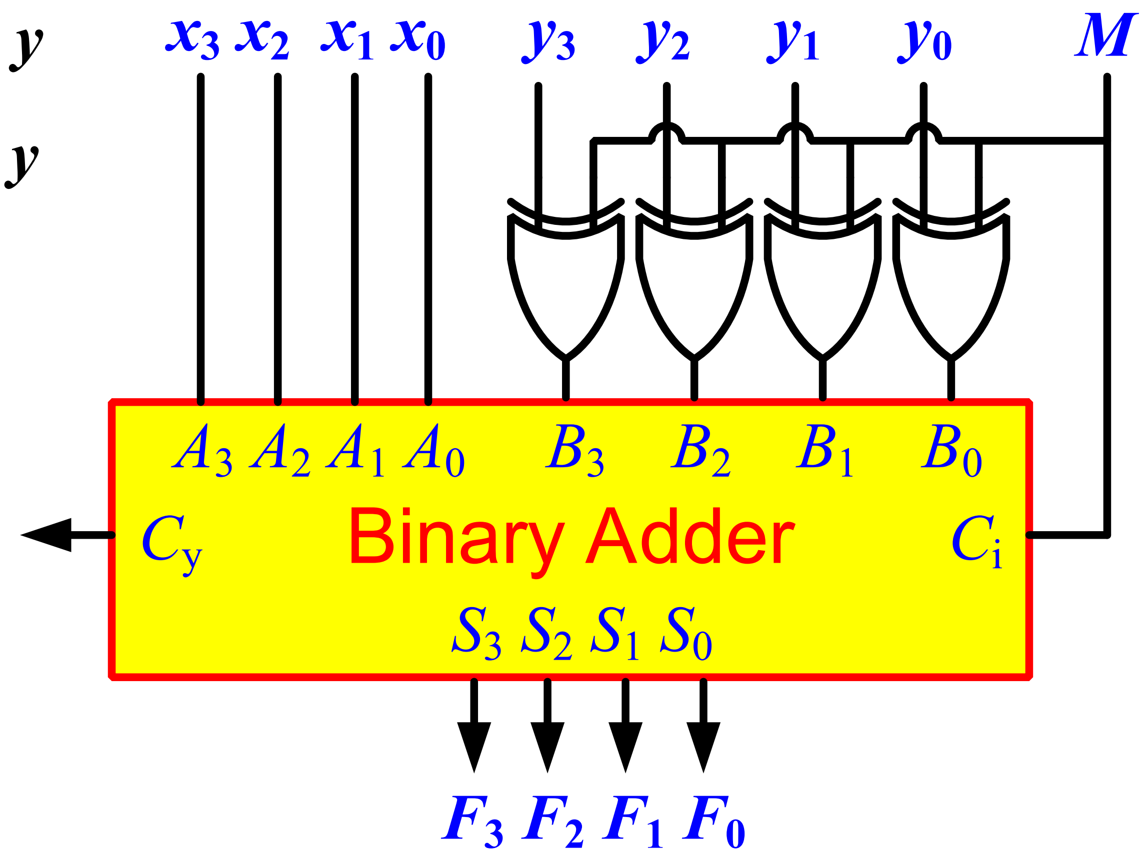
- **Use 2's complement with binary adder**
  - $x - y = x + (-y) = x + y' + 1$



Binary Adder/Subtractor

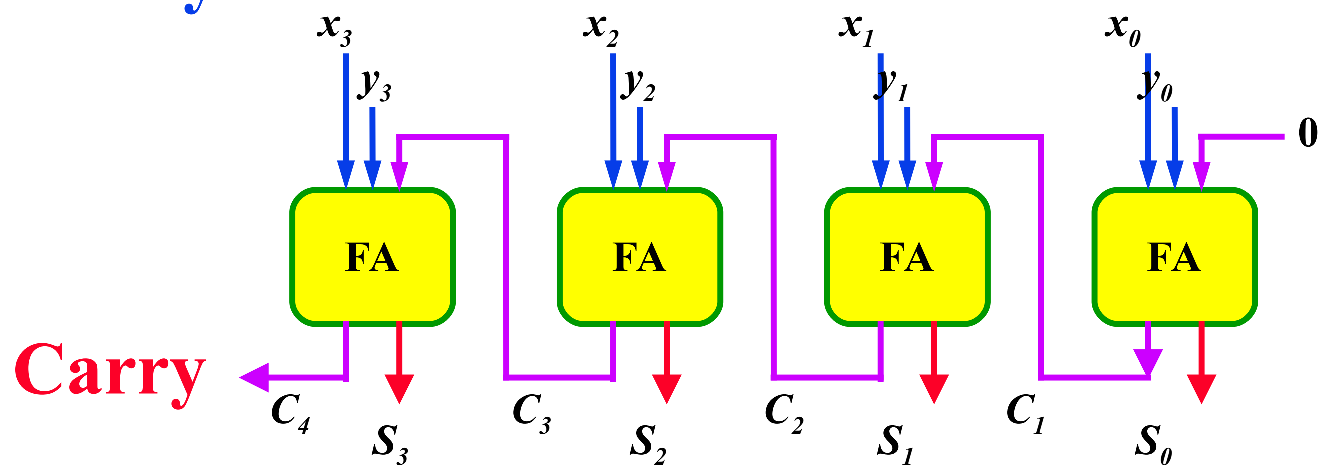
★ *M*: Control Signal (Mode)

- $M=0 \Rightarrow F = x + y$
- $M=1 \Rightarrow F = x - y$

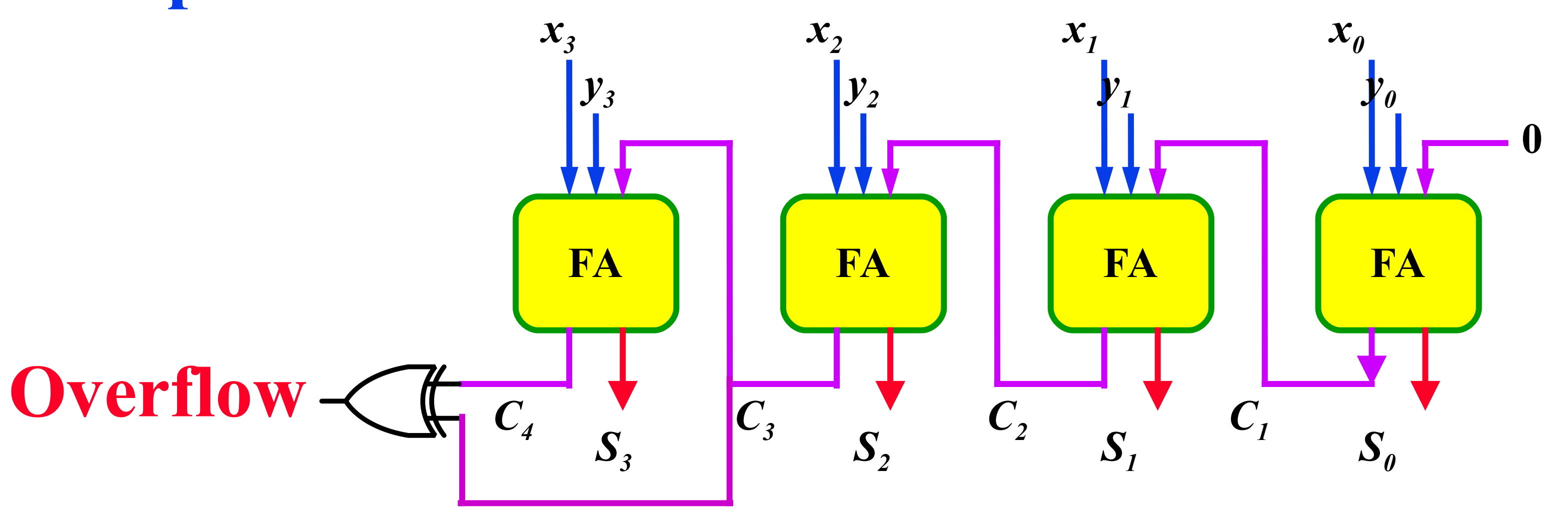


Overflow

★ **Unsigned Binary Numbers**



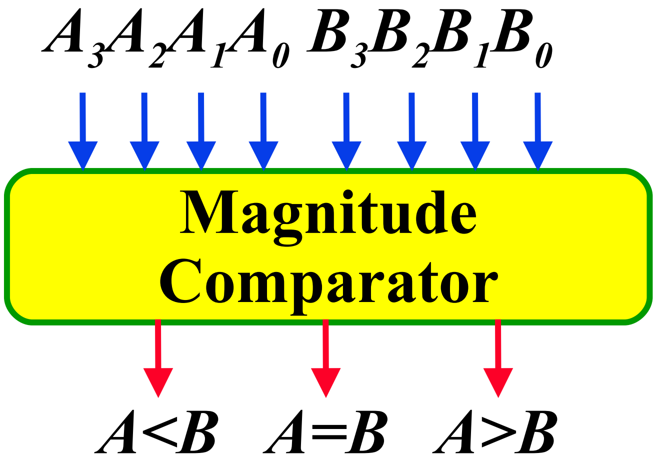
★ **2's Complement Numbers**



Magnitude Comparator

★ Compare 4-bit number to 4-bit number

- **3 Outputs: < , = , >**
- **Expandable to more number of bits**



$$x_3 = \overline{A}_3\,\overline{B}_3 + A_3\,B_3$$

$$x_2 = \overline{A}_2\,\overline{B}_2 + A_2\,B_2$$

$$x_1 = \overline{A}_1\,\overline{B}_1 + A_1\,B_1$$

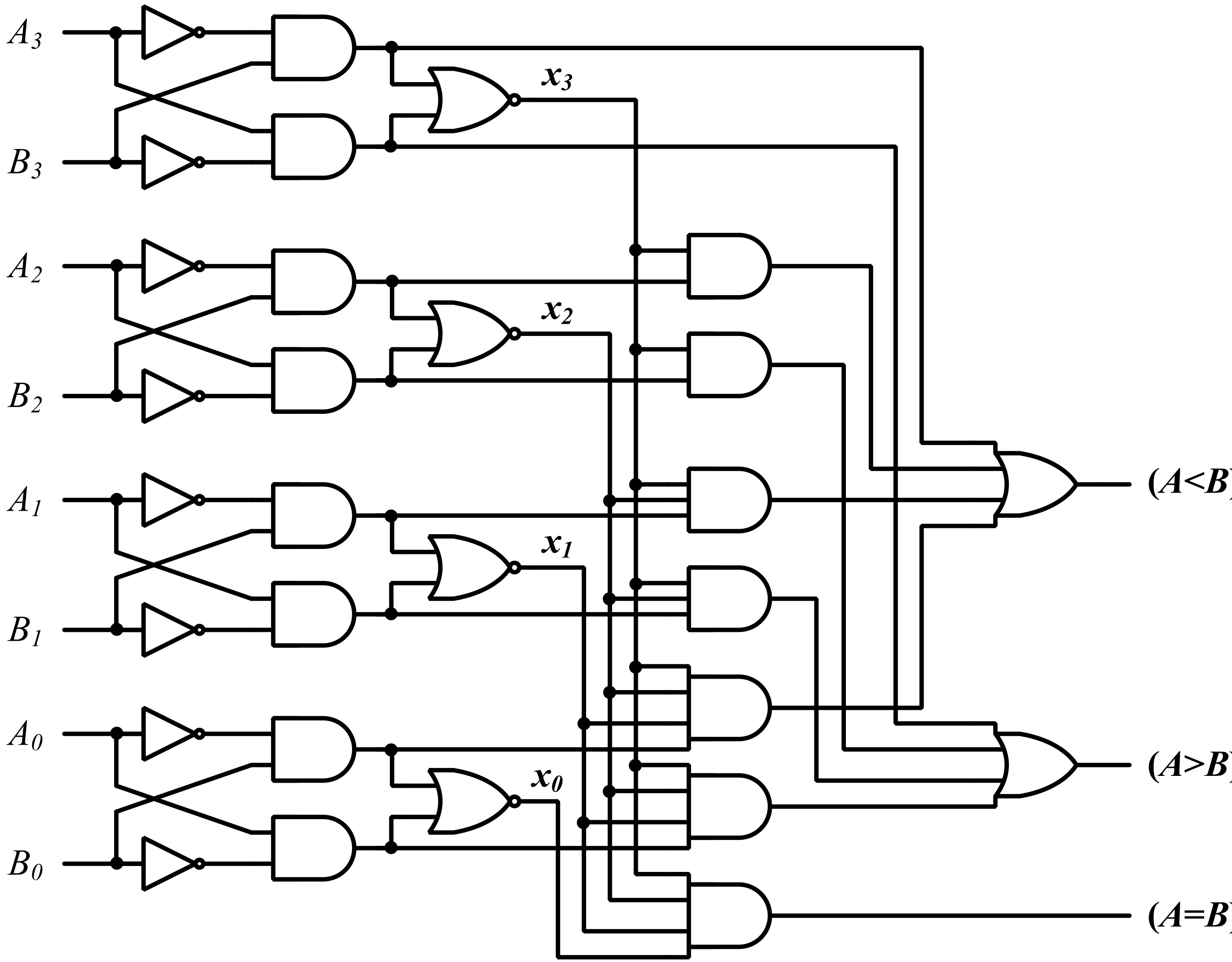
$$x_0 = \overline{A}_0\,\overline{B}_0 + A_0\,B_0$$

$$(A = B) = x_3\,x_2\,x_1\,x_0$$

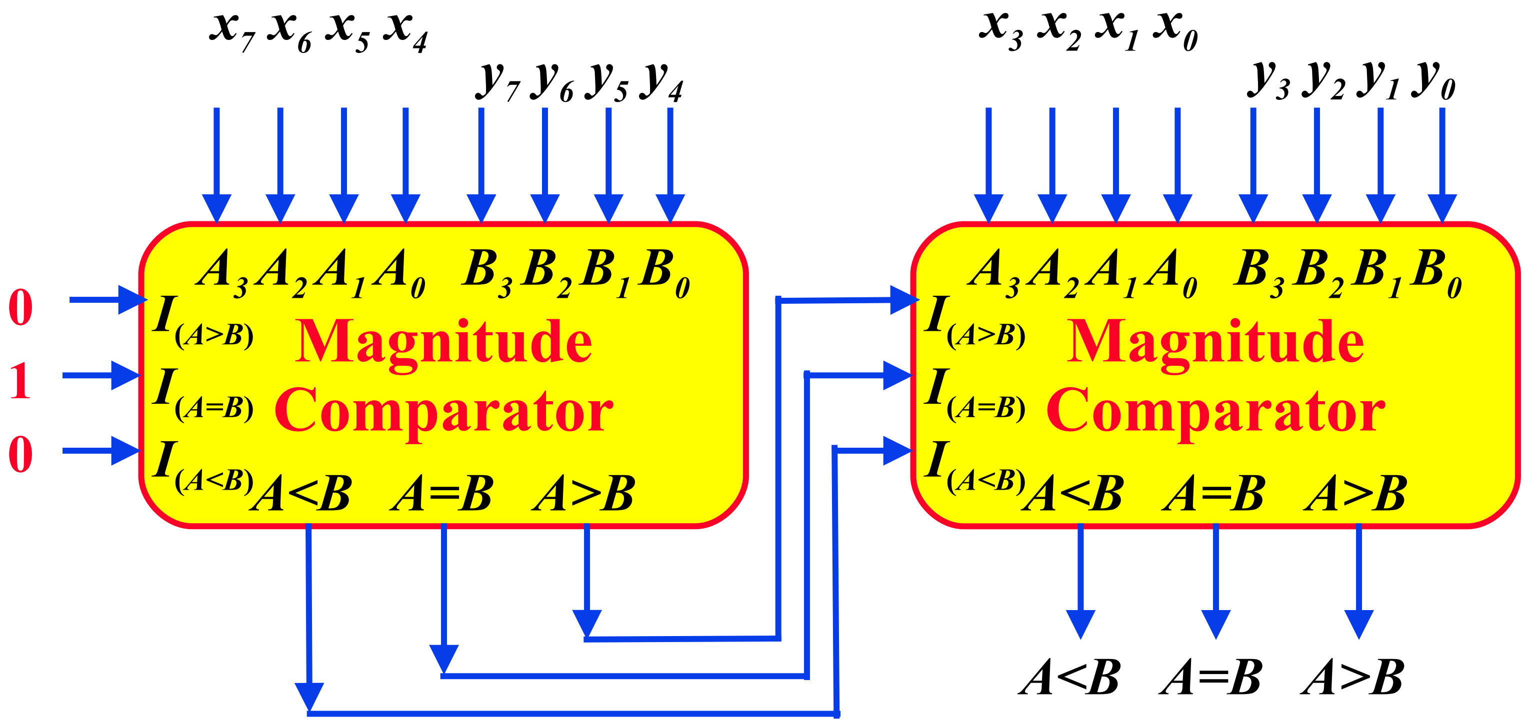
$$(A > B) = A_3\,\overline{B}_3 + x_3\,A_2\,\overline{B}_2 + x_3\,x_2\,A_1\,\overline{B}_1 + x_3\,x_2\,x_1\,A_0\,\overline{B}_0$$

$$(A < B) = \overline{A}_3\,B_3 + x_3\,\overline{A}_2\,B_2 + x_3\,x_2\,\overline{A}_1\,B_1 + x_3\,x_2\,x_1\,\overline{A}_0\,B_0$$

Magnitude Comparator



Magnitude Comparator

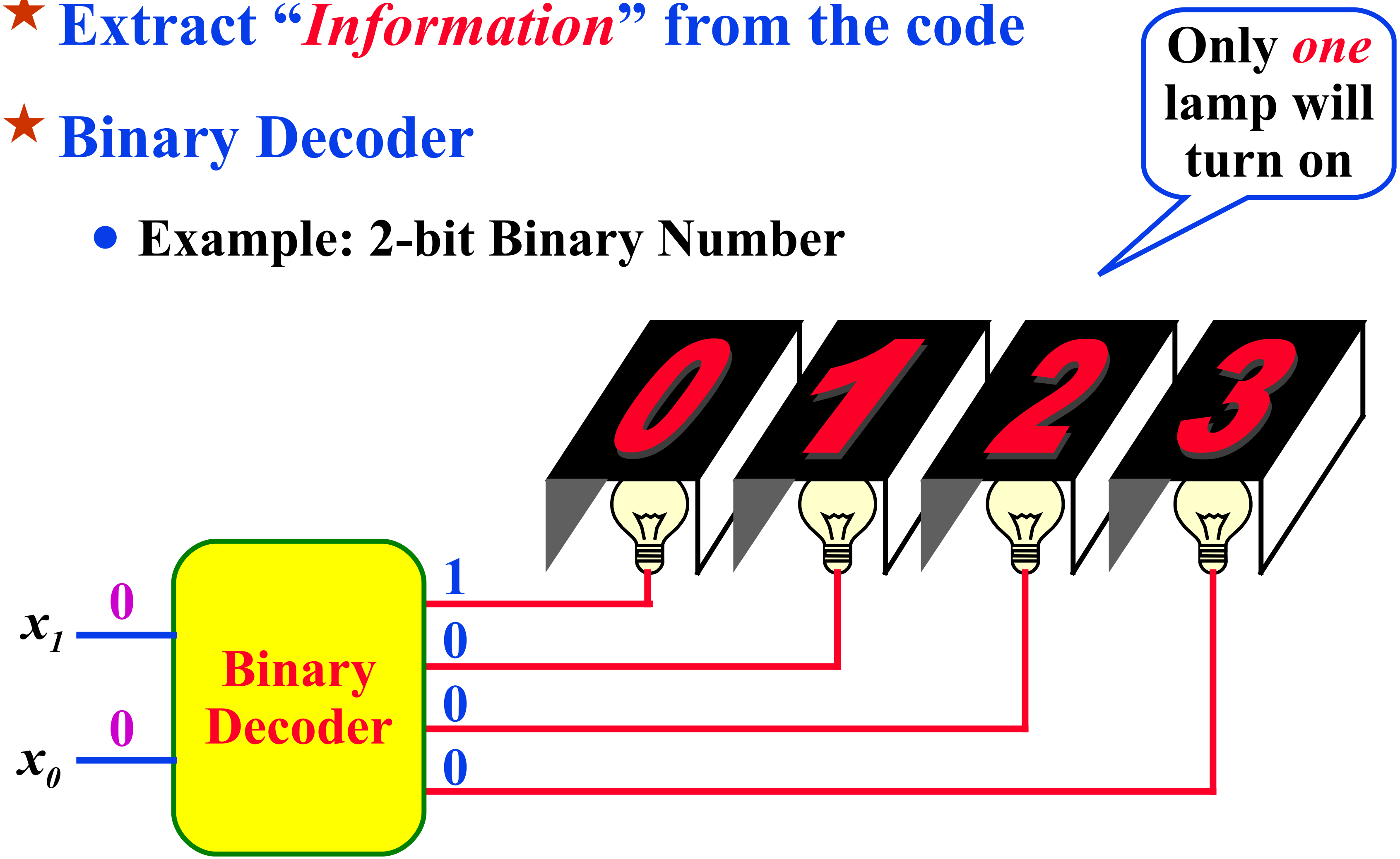


Decoders

★ **Extract "*Information*" from the code**

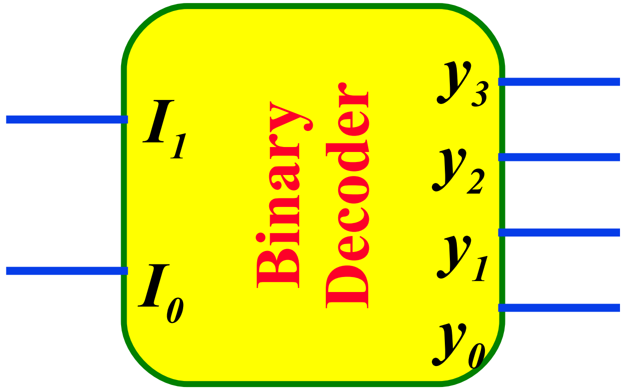
★ **Binary Decoder**

- **Example: 2-bit Binary Number**

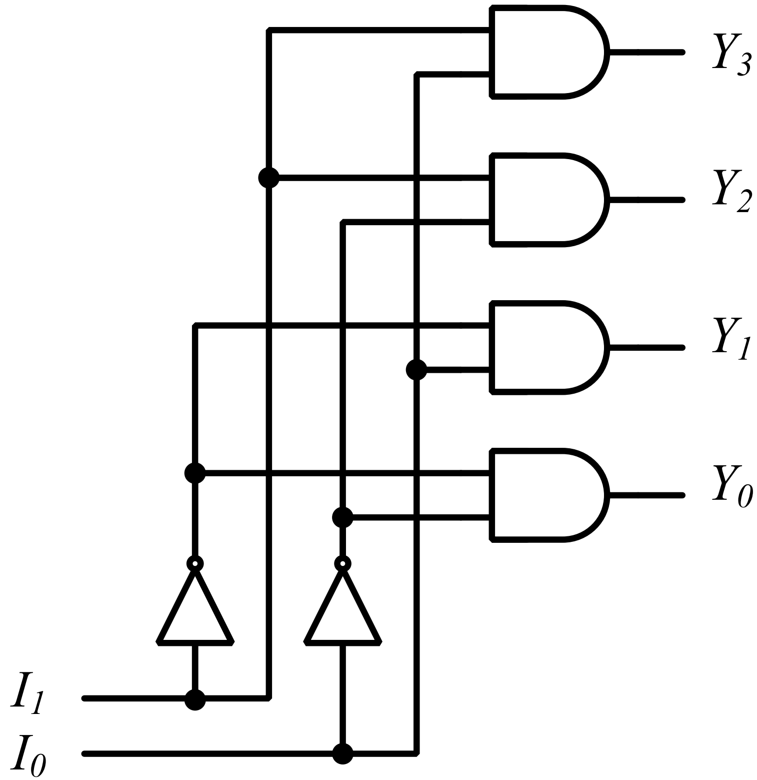


Decoders

★ **2-to-4 Line Decoder**



I_1 I_0	Y_3 Y_2 Y_1 Y_0
0 0	0 0 0 1
0 1	0 0 1 0
1 0	0 1 0 0
1 1	1 0 0 0

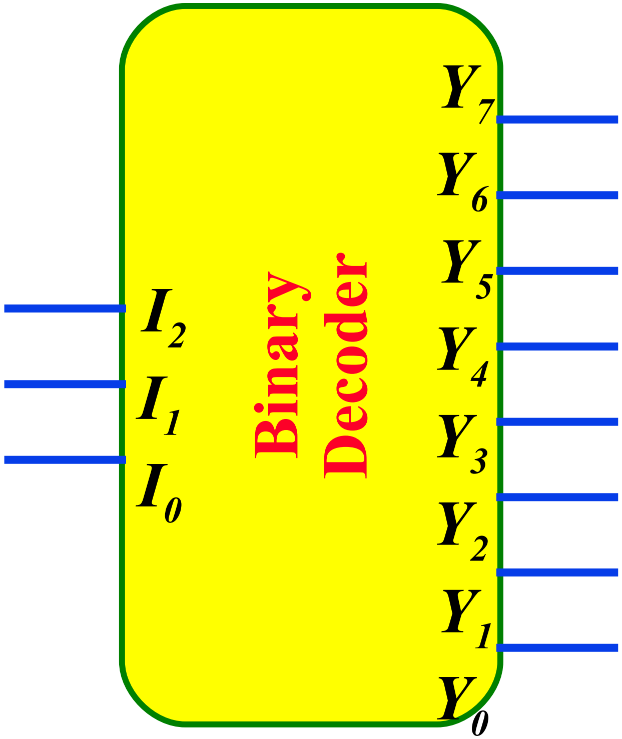


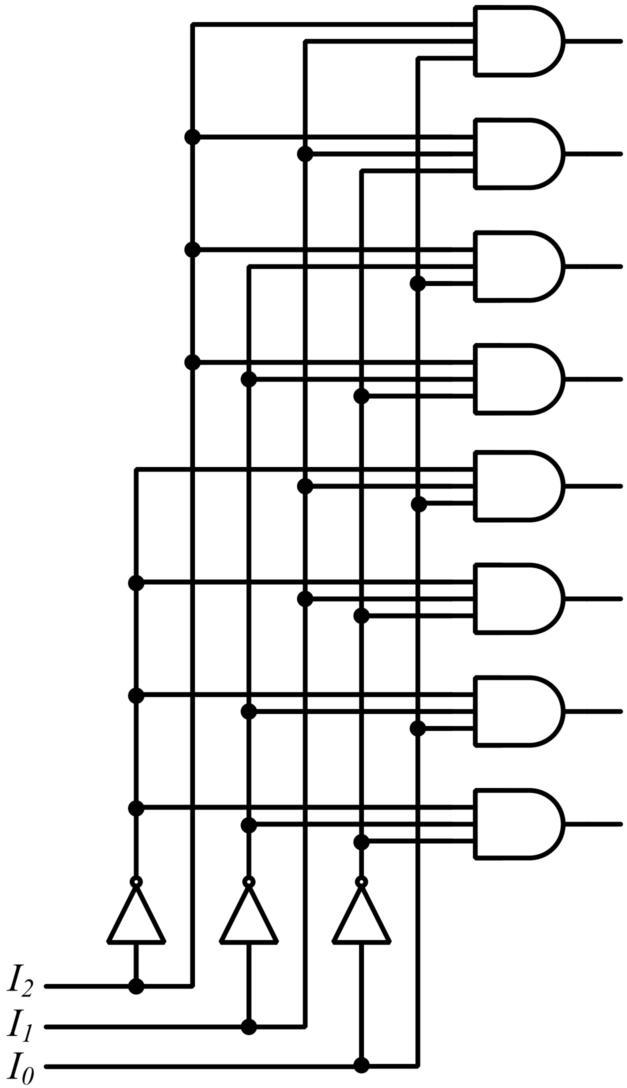
$Y_3 = I_1 I_0$ $\quad$ $Y_2 = I_1 \bar{I}_0$

$Y_1 = \bar{I}_1 I_0$ $\quad$ $Y_0 = \bar{I}_1 \bar{I}_0$

Decoders

★ **3-to-8 Line Decoder**

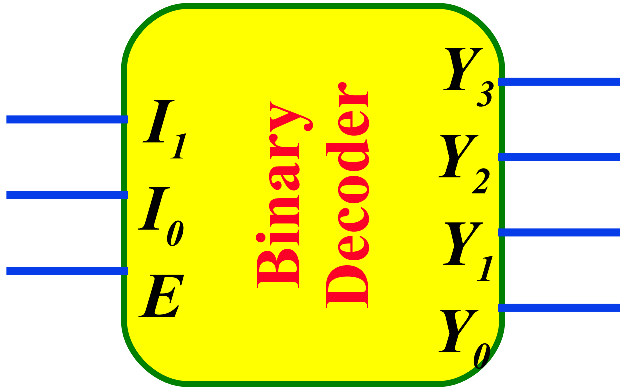




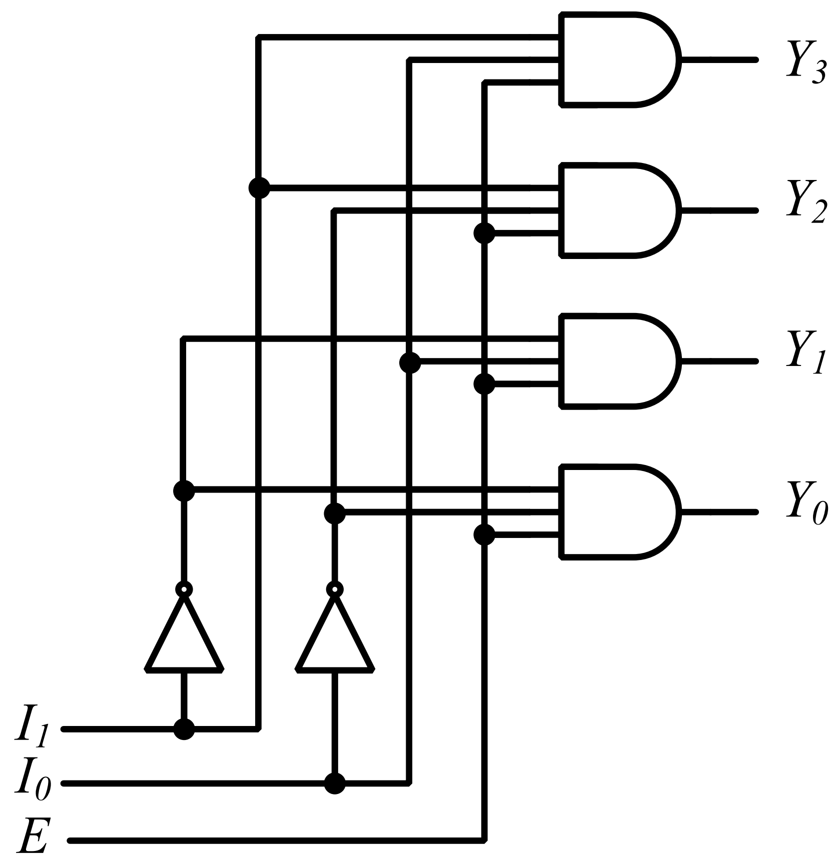
$$Y_7 = I_2 I_1 I_0$$
$$Y_6 = I_2 I_1 \bar{I}_0$$
$$Y_5 = I_2 \bar{I}_1 I_0$$
$$Y_4 = I_2 \bar{I}_1 \bar{I}_0$$
$$Y_3 = \bar{I}_2 I_1 I_0$$
$$Y_2 = \bar{I}_2 I_1 \bar{I}_0$$
$$Y_1 = \bar{I}_2 \bar{I}_1 I_0$$
$$Y_0 = \bar{I}_2 \bar{I}_1 \bar{I}_0$$

Decoders

★ "*Enable*" Control



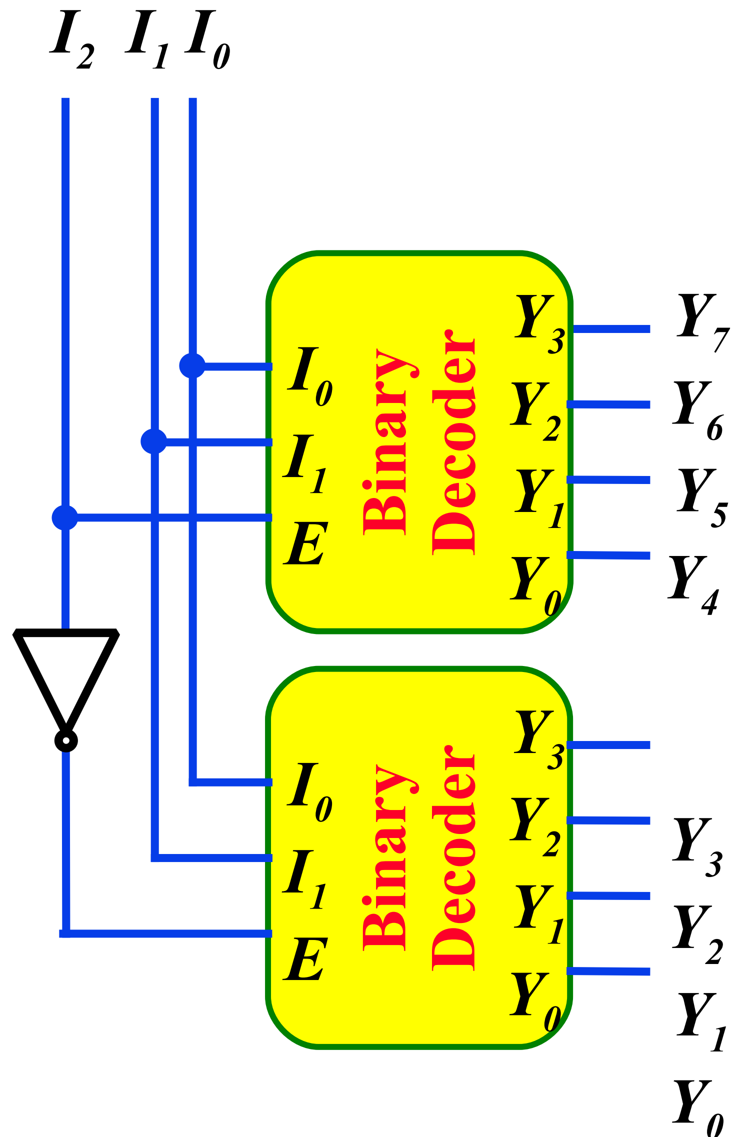
E	I_1 I_0	Y_3 Y_2 Y_1 Y_0
0	x x	0 0 0 0
1	0 0	0 0 0 1
1	0 1	0 0 1 0
1	1 0	0 1 0 0
1	1 1	1 0 0 0



Decoders

★ **Expansion**

I_2	I_1	I_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
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

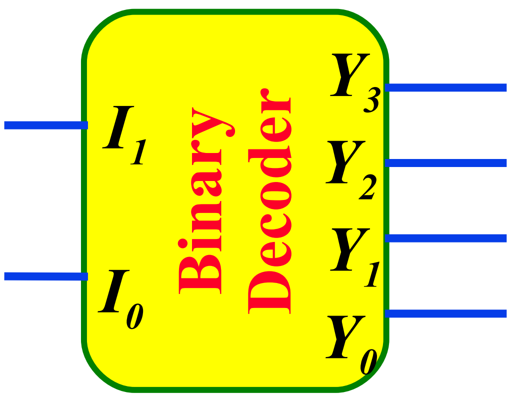


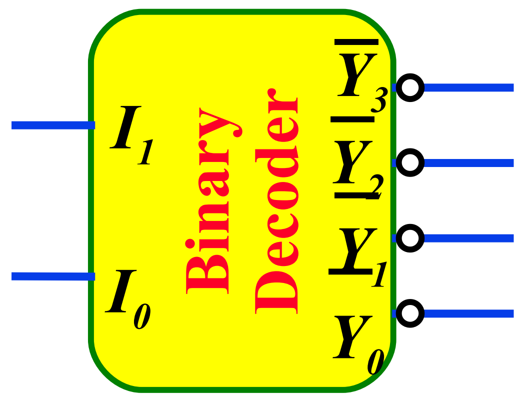
Decoders

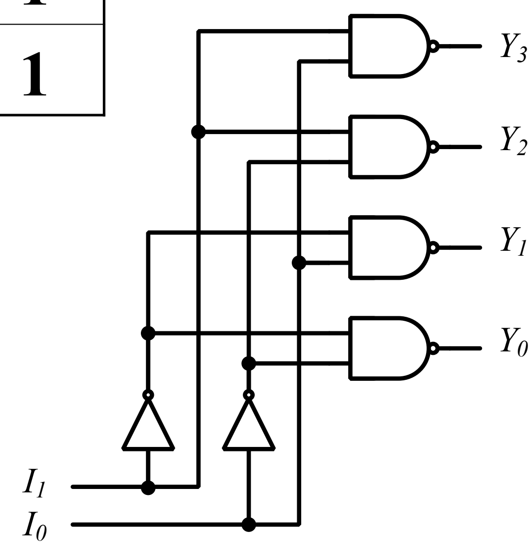
★ **Active-High / Active-Low**

I_1	I_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

I_1	I_0	Y_3	Y_2	Y_1	Y_0
0	0	1	1	1	0
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	1	1	1







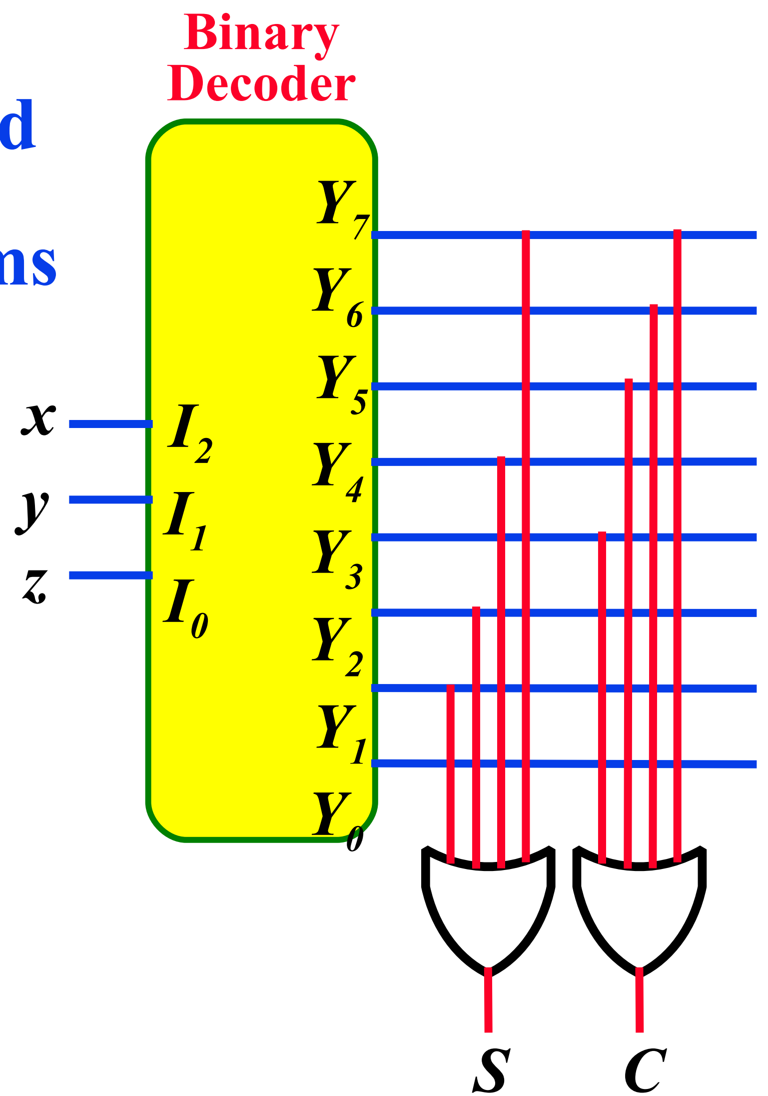
Implementation Using Decoders

- ★ Each output is a minterm
- ★ All minterms are produced
- ★ Sum the required minterms

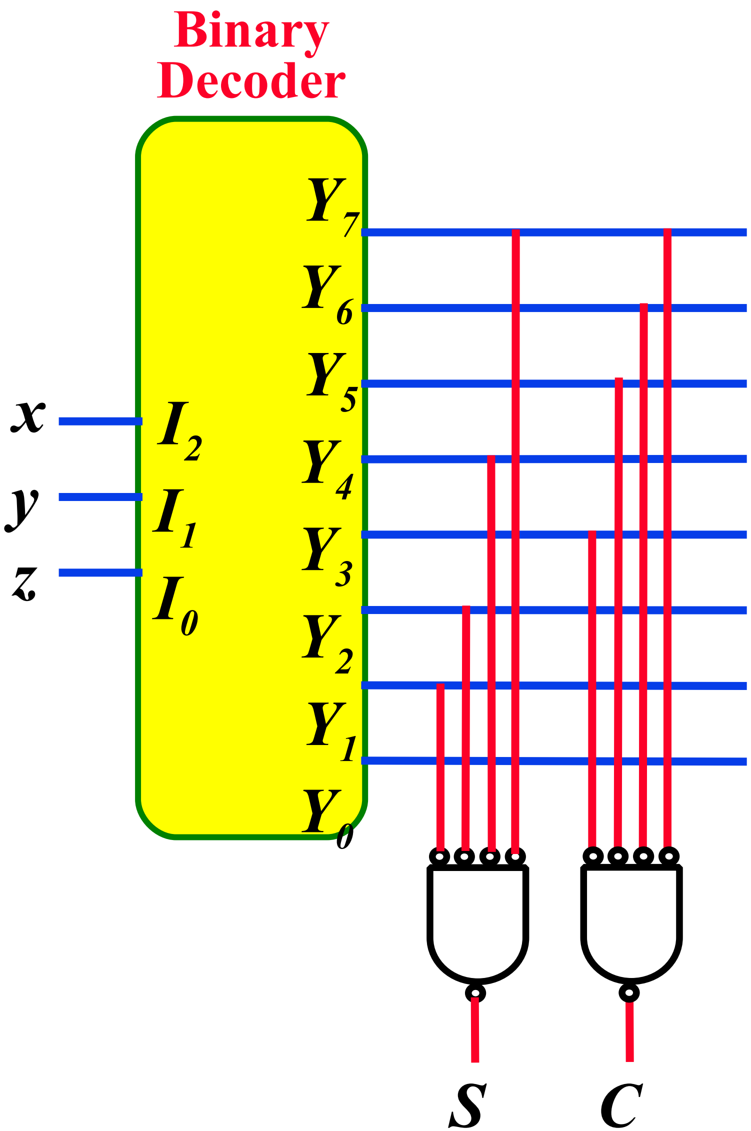
Example: **Full Adder**

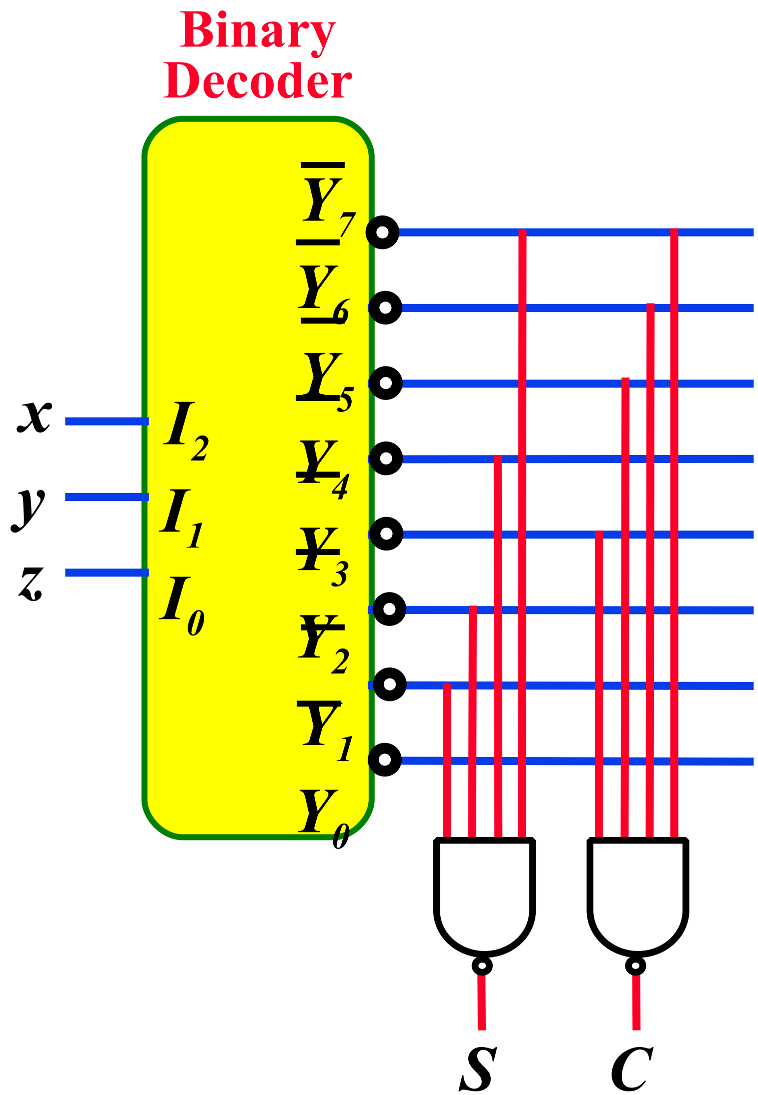
$$S(x, y, z) = \sum(1, 2, 4, 7)$$

$$C(x, y, z) = \sum(3, 5, 6, 7)$$



Implementation Using Decoders

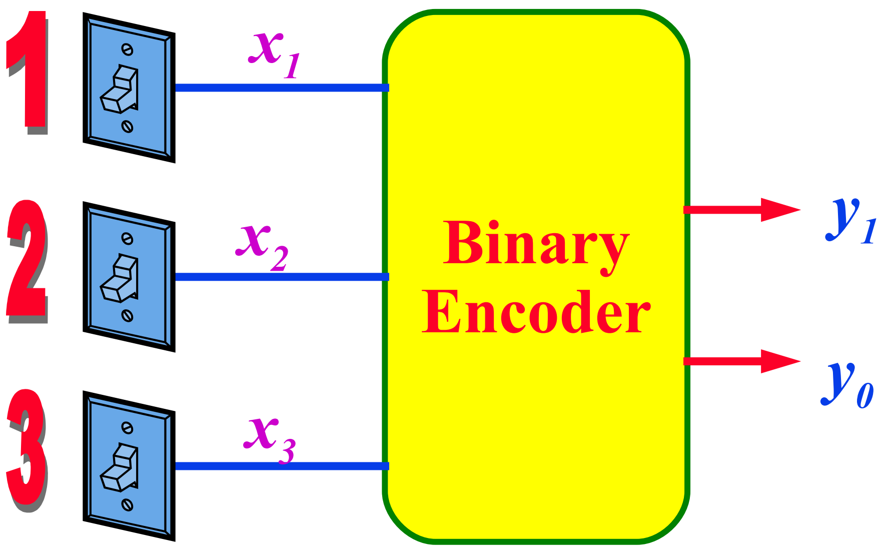




Encoders

- Put "*Information*" into code
- Binary Encoder
  - Example: 4-to-2 Binary Encoder

Only *one* switch should be activated at a time

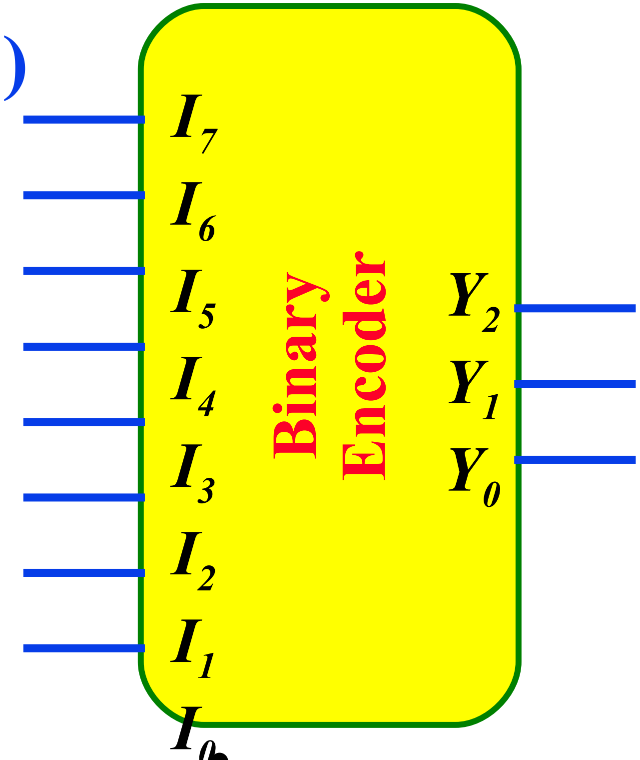


x_3	x_2	x_1	y_1	y_0
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
1	0	0	1	1

Encoders

★ **Octal-to-Binary Encoder (8-to-3)**

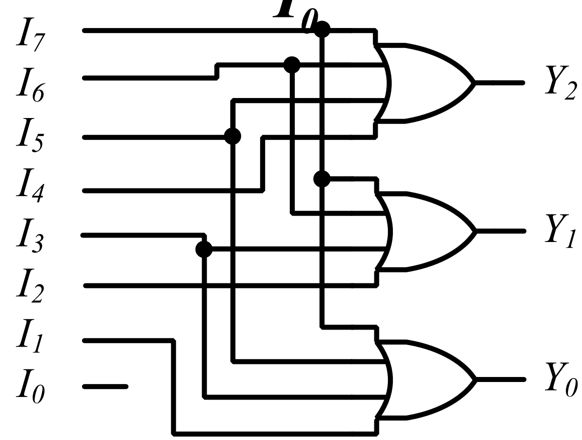
I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1



$$Y_2 = I_7 + I_6 + I_5 + I_4$$

$$Y_1 = I_7 + I_6 + I_3 + I_2$$

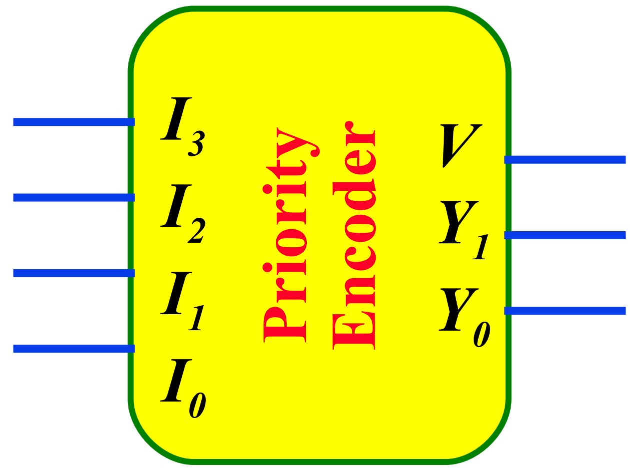
$$Y_0 = I_7 + I_5 + I_3 + I_1$$

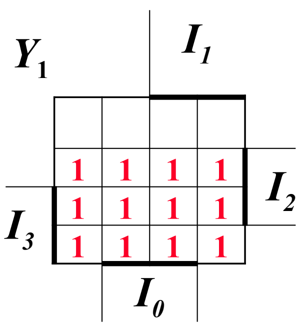


Priority Encoders

★ 4-Input Priority Encoder

I_3	I_2	I_1	I_0	Y_1	Y_0	V
0	0	0	0	0	0	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

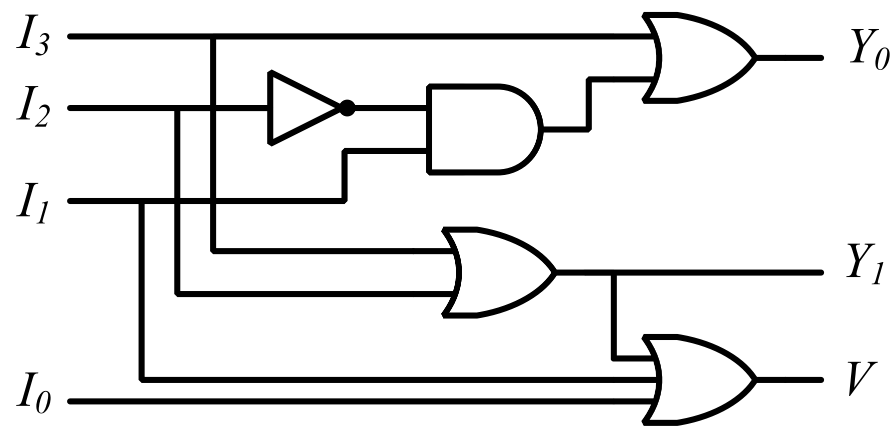




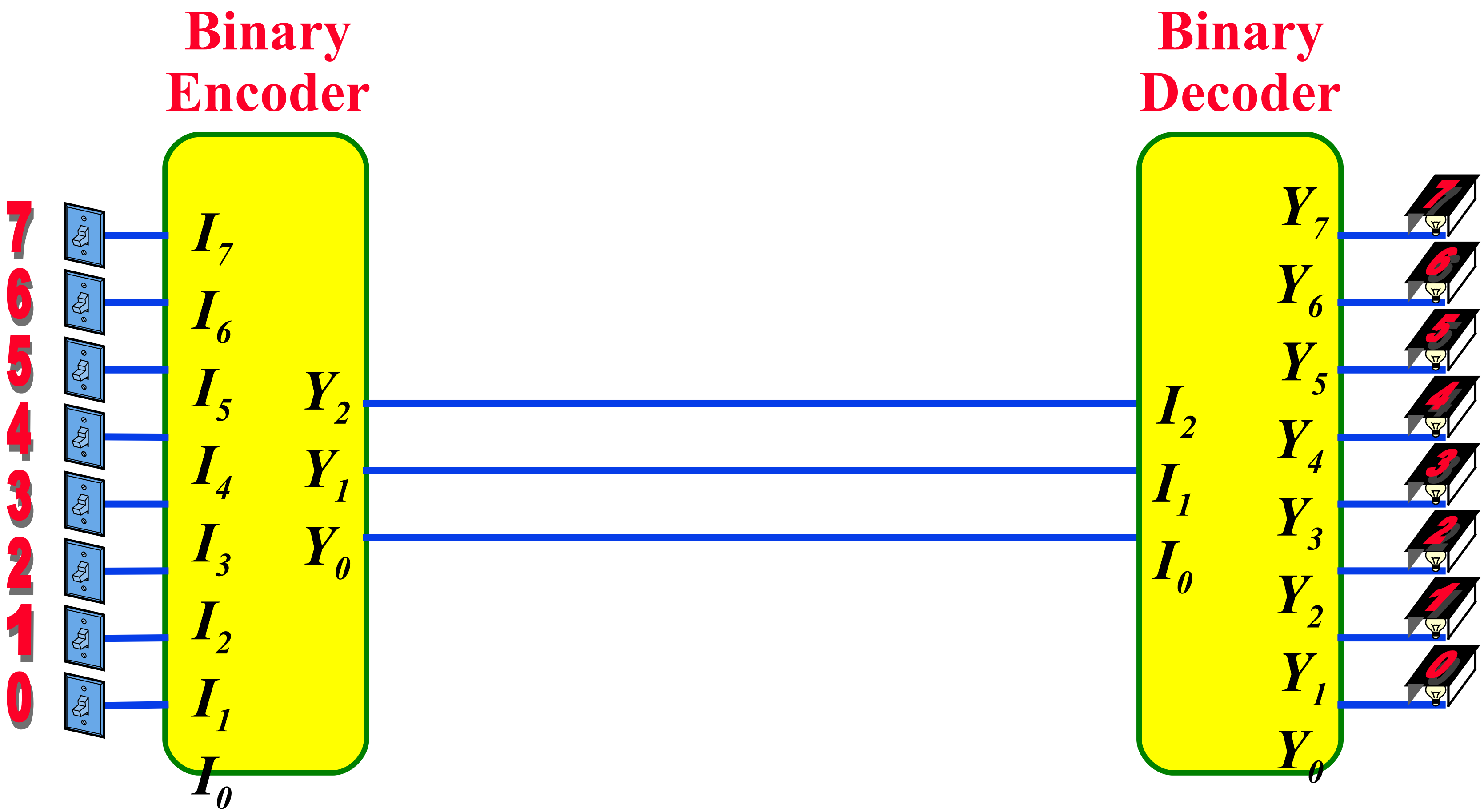
$$Y_1 = I_3 + I_2$$

$$Y_0 = I_3 + \bar{I}_2 I_1$$

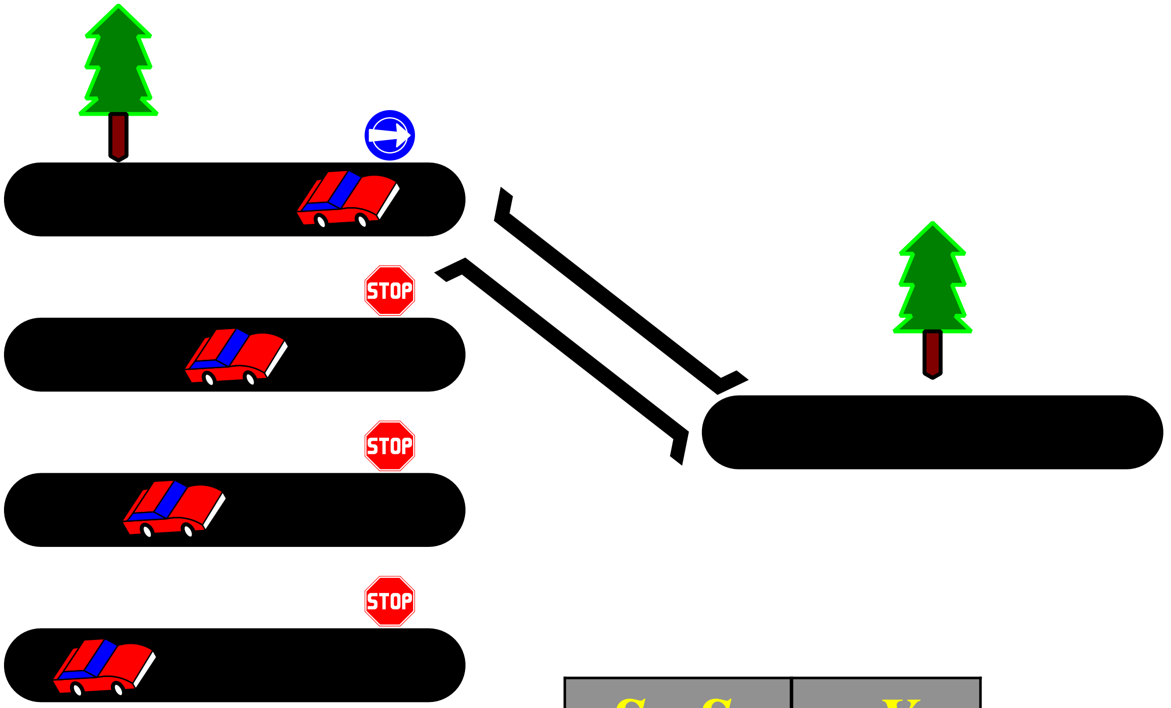
$$V = I_3 + I_2 + I_1 + I_0$$



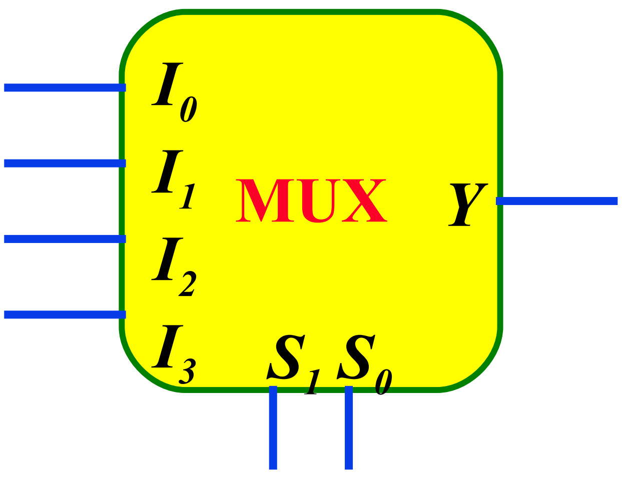
Encoder / Decoder Pairs



Multiplexers

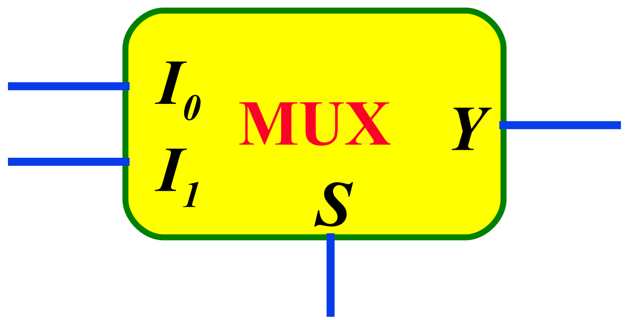


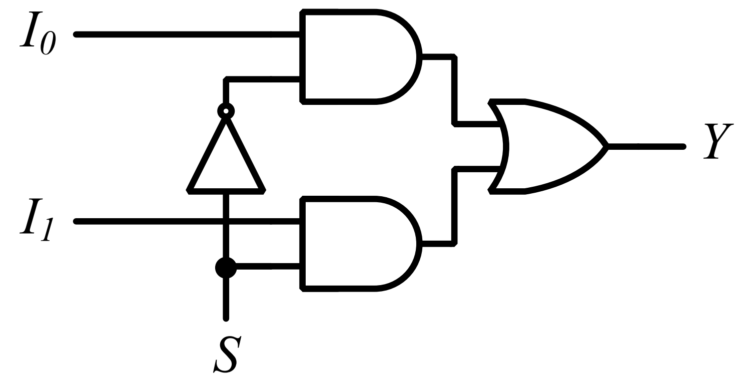
S_1 S_0	Y
0 0	I_0
0 1	I_1
1 0	I_2
1 1	I_3



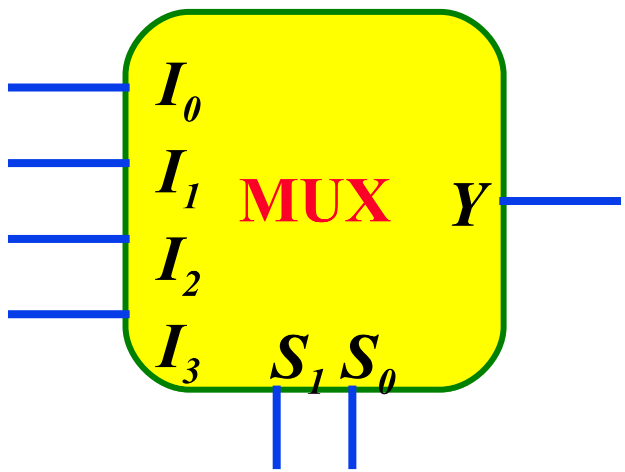
Multiplexers

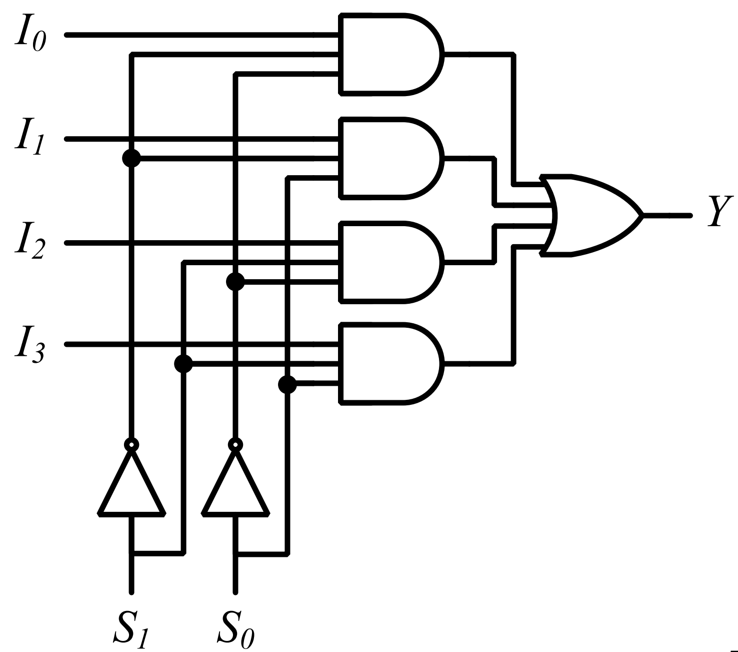
★ **2-to-1 MUX**





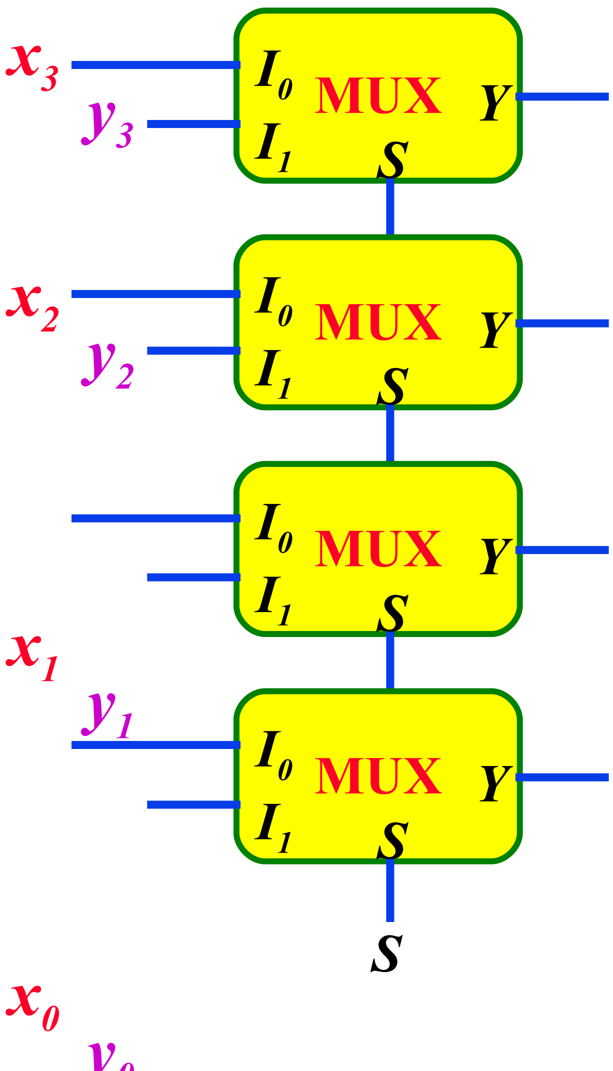
★ **4-to-1 MUX**

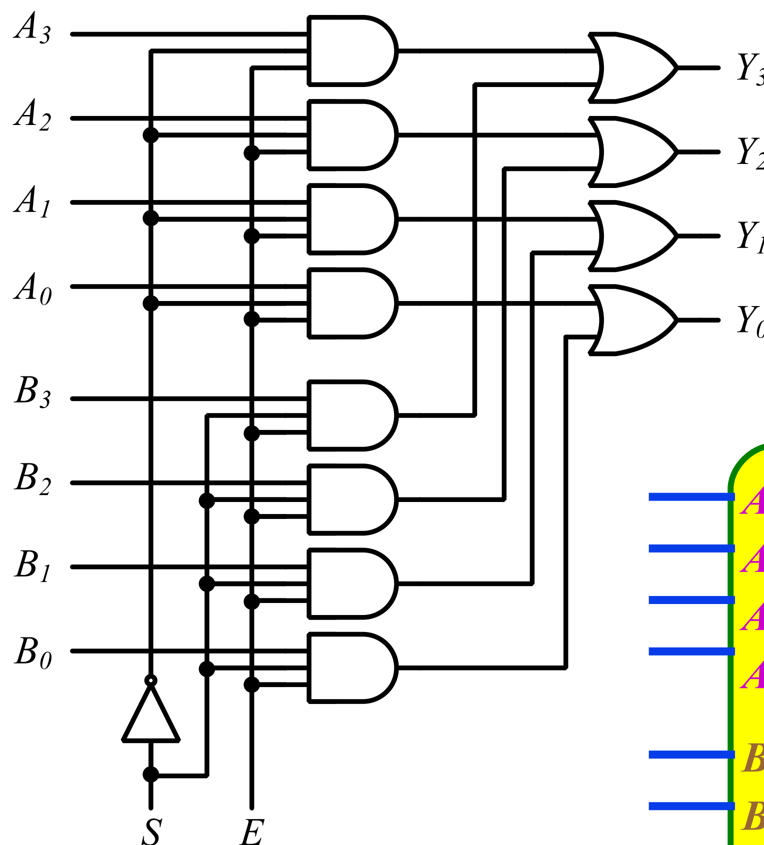


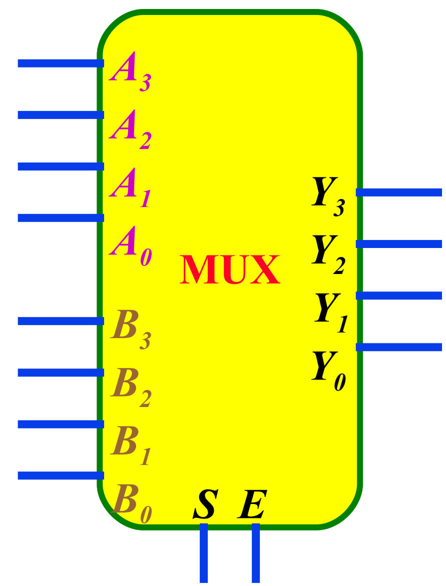


Multiplexers

★ Quad 2-to-1 MUX

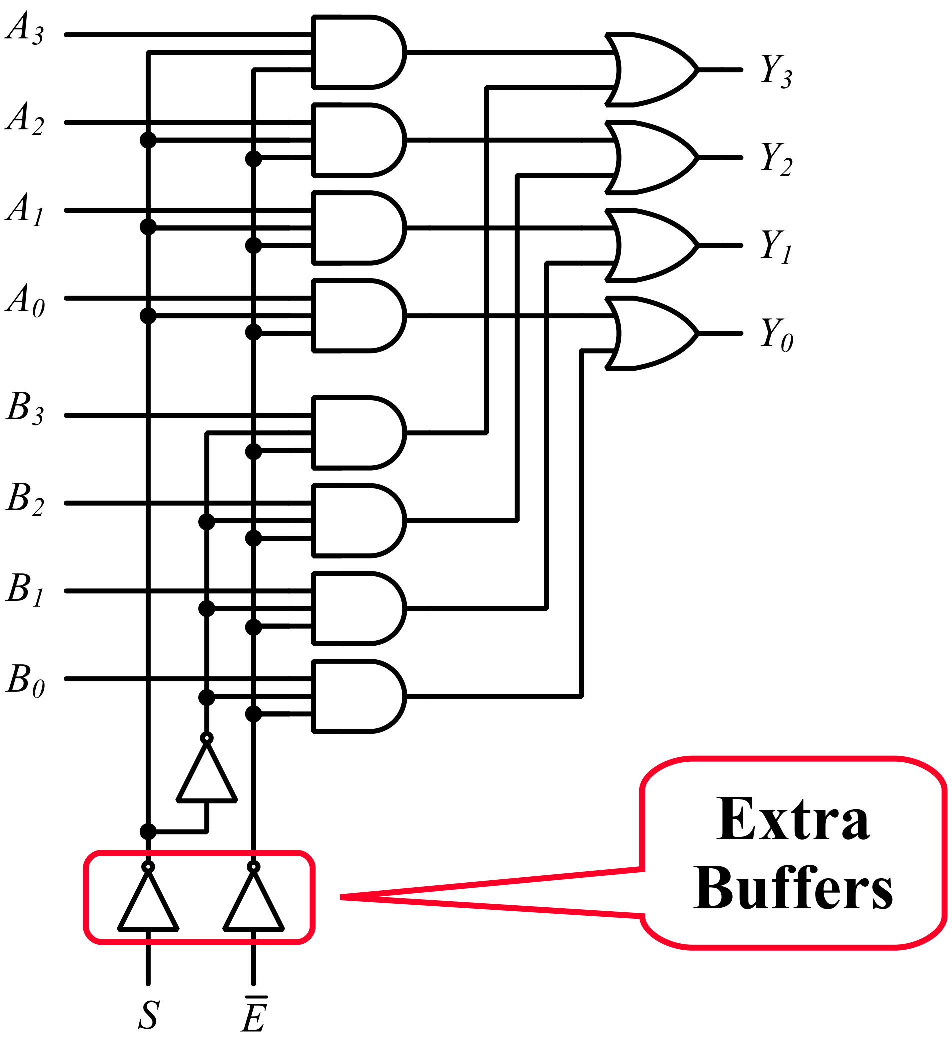


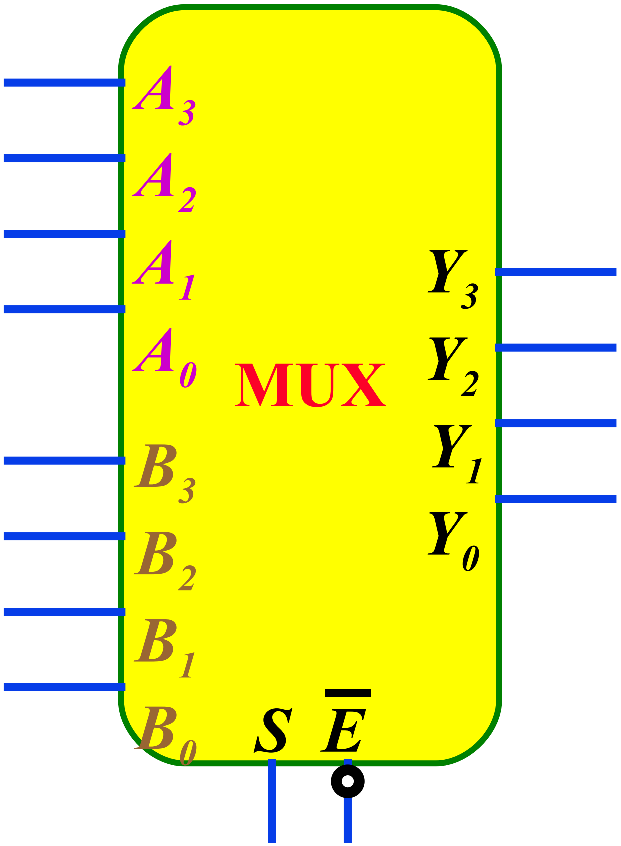




Multiplexers

★ Quad 2-to-1 MUX



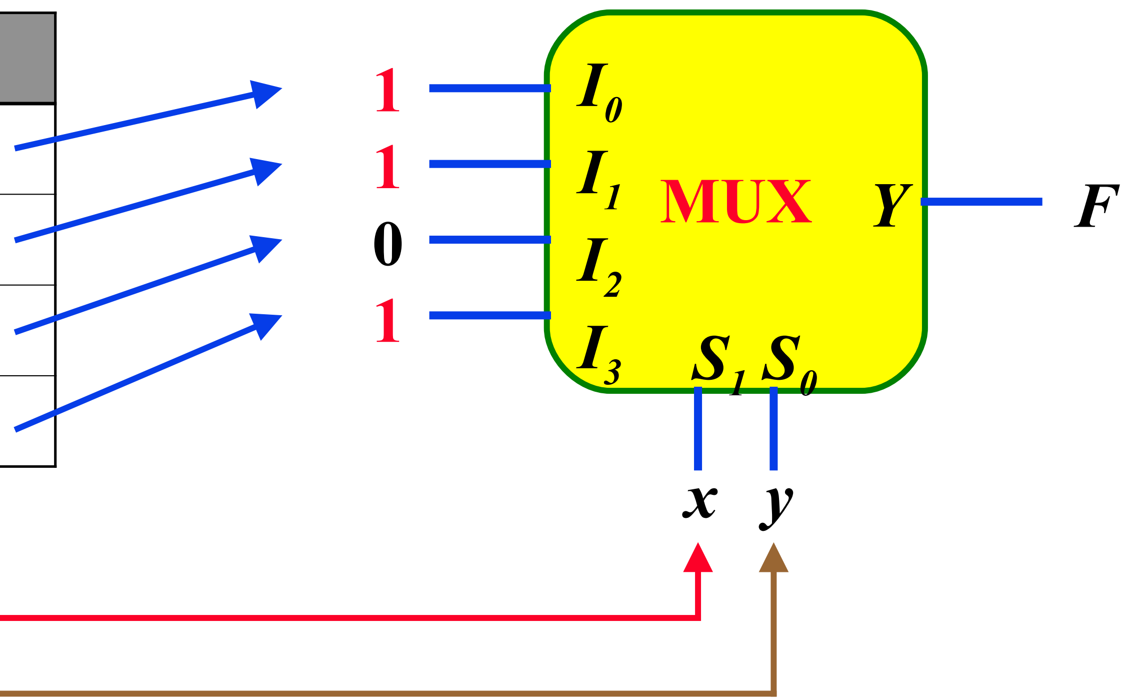


Implementation Using Multiplexers

★ **Example**

$F(x, y) = \sum(0, 1, 3)$

x y	F
0 0	1
0 1	1
1 0	0
1 1	1

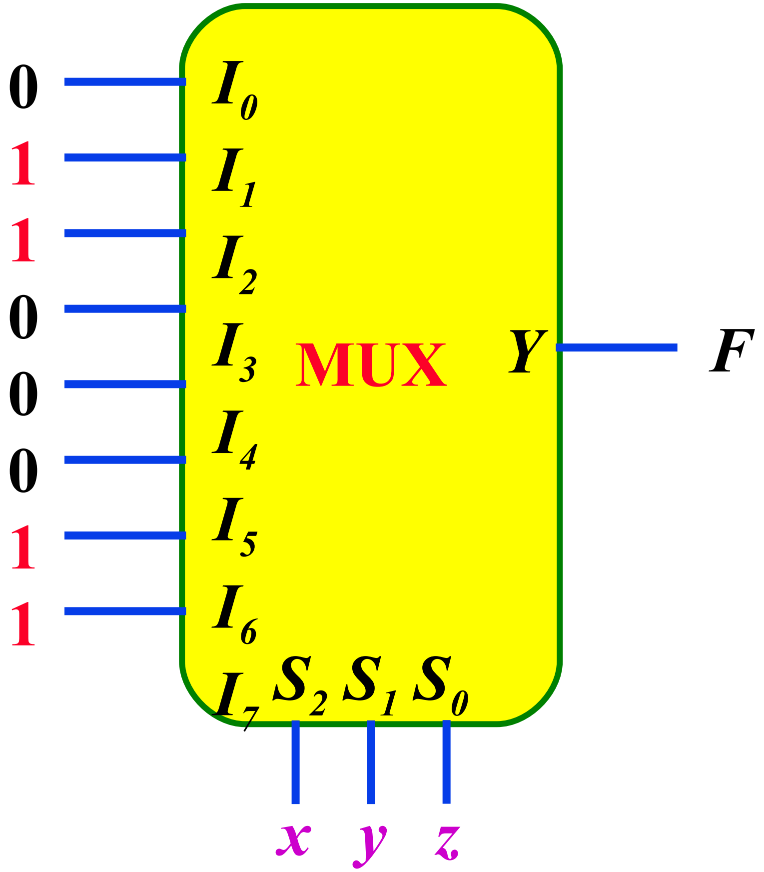


Implementation Using Multiplexers

★ **Example**

$F(x, y, z) = \sum(1, 2, 6, 7)$

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



Implementation Using Multiplexers

★ **Example**

$F(x, y, z) = \sum(1, 2, 6, 7)$

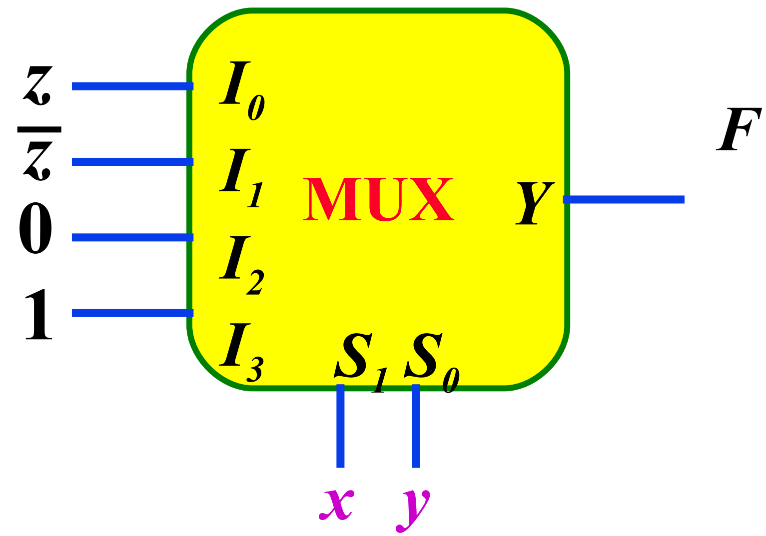
x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

$F = z$

$F = \overline{z}$

$F = 0$

$F = 1$

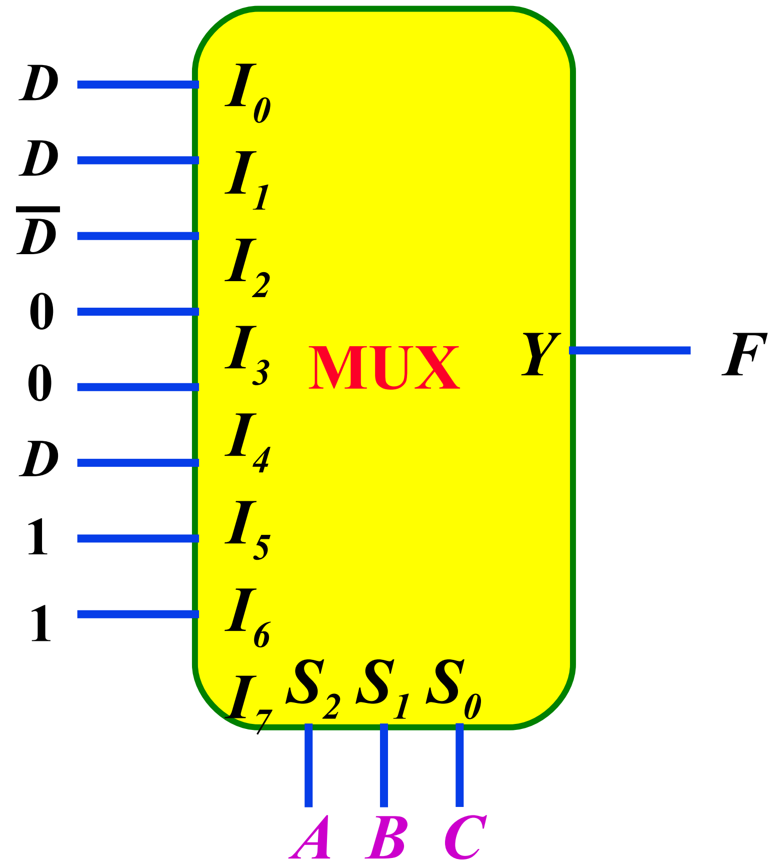


Implementation Using Multiplexers

★ **Example**

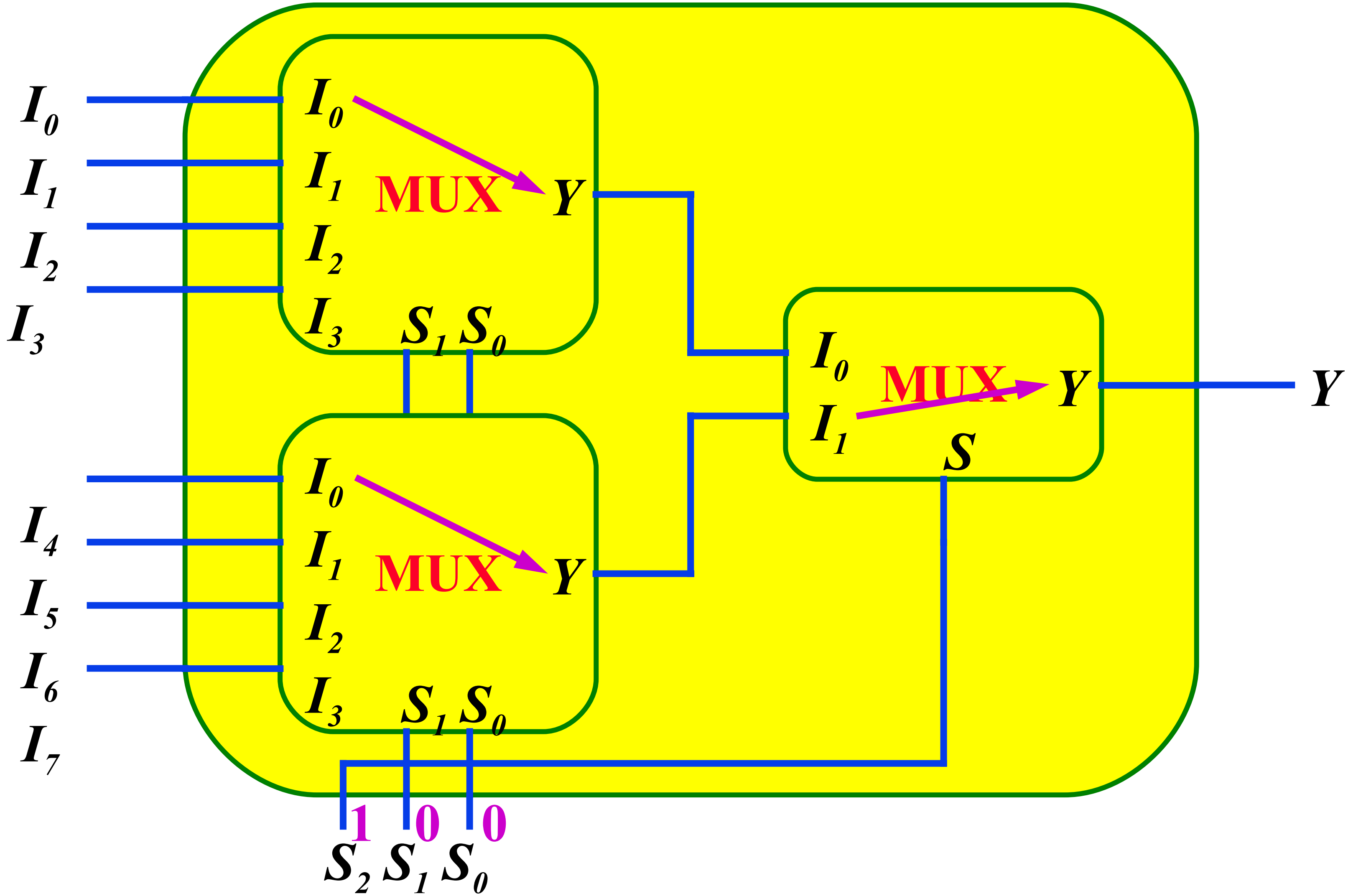
$F(A, B, C, D) = \sum(1, 3, 4, 11, 12, 13, 14, 15)$

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

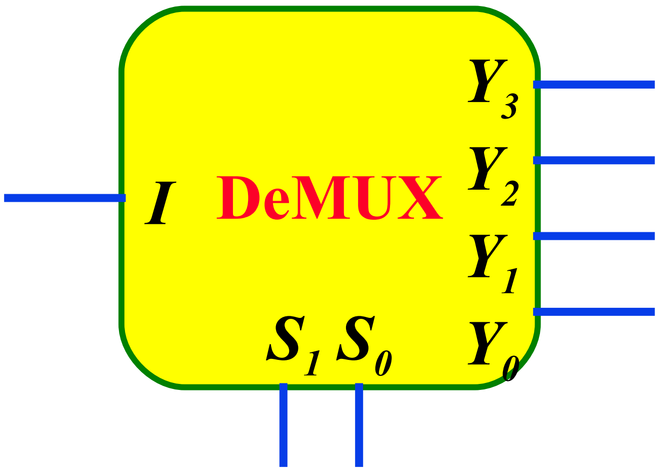


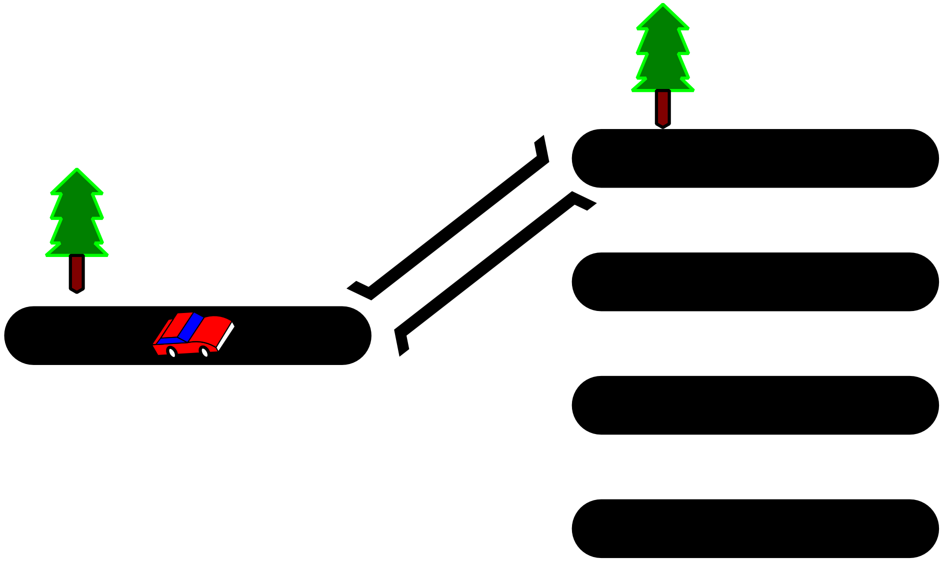
Multiplexer Expansion

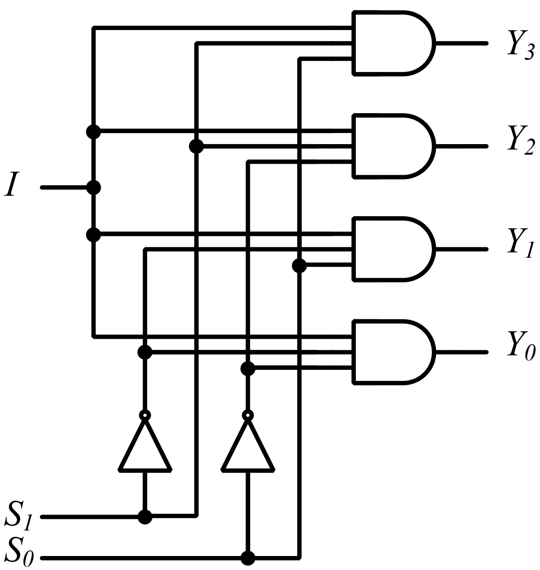
★ 8-to-1 MUX using Dual 4-to-1 MUX



DeMultiplexers

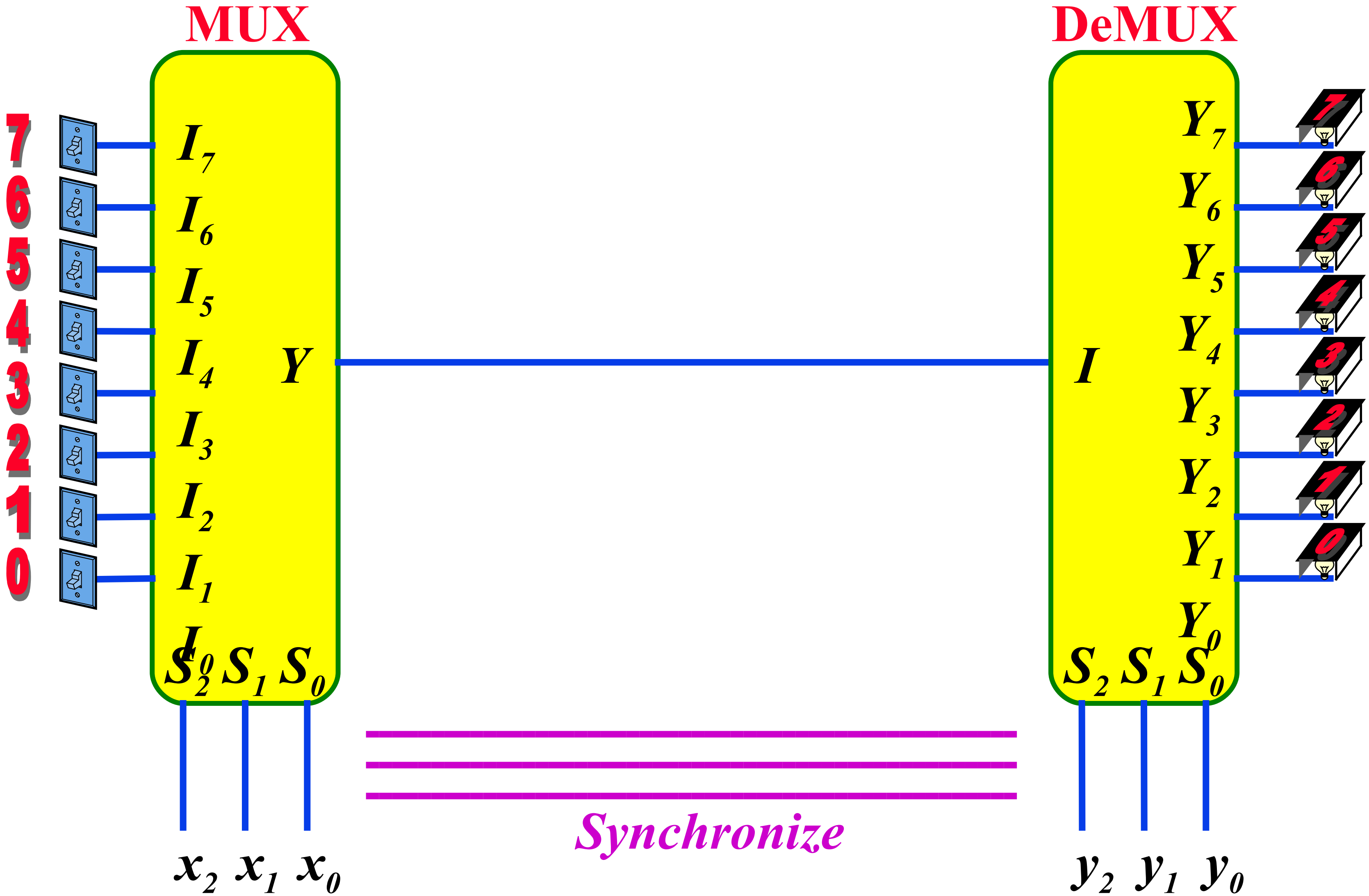




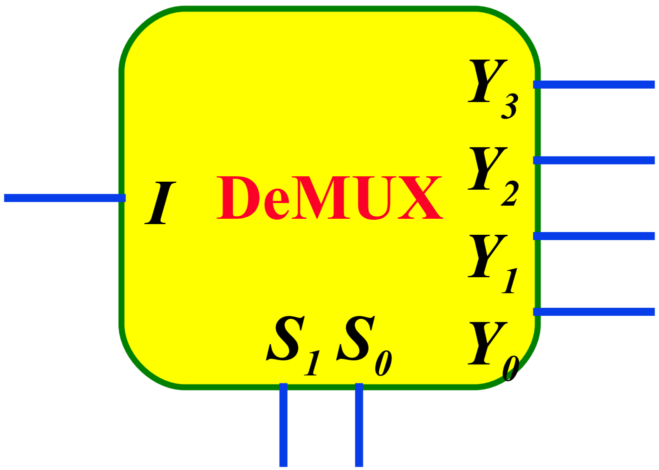


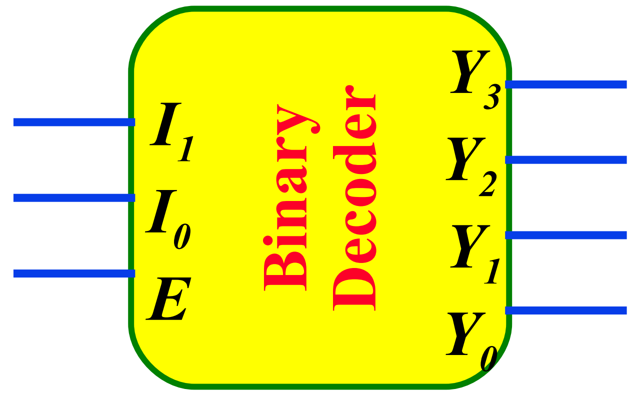
S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	I
0	1	0	0	I	0
1	0	0	I	0	0
1	1	I	0	0	0

Multiplexer / DeMultiplexer Pairs



DeMultiplexers / Decoders



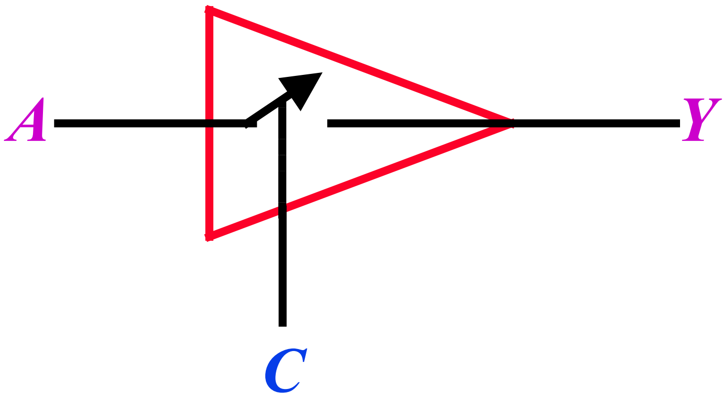


S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	I
0	1	0	0	I	0
1	0	0	I	0	0
1	1	I	0	0	0

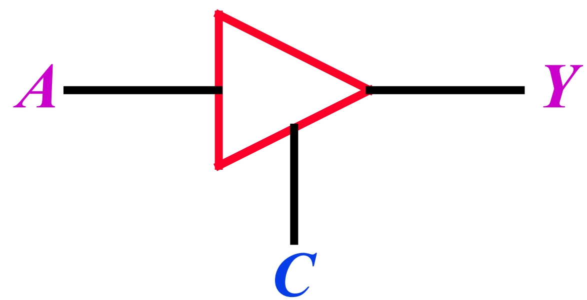
E	I_1	I_0	Y_3	Y_2	Y_1	Y_0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

Three-State Gates

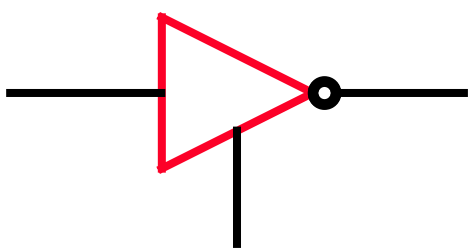
★ **Tri-State Buffer**



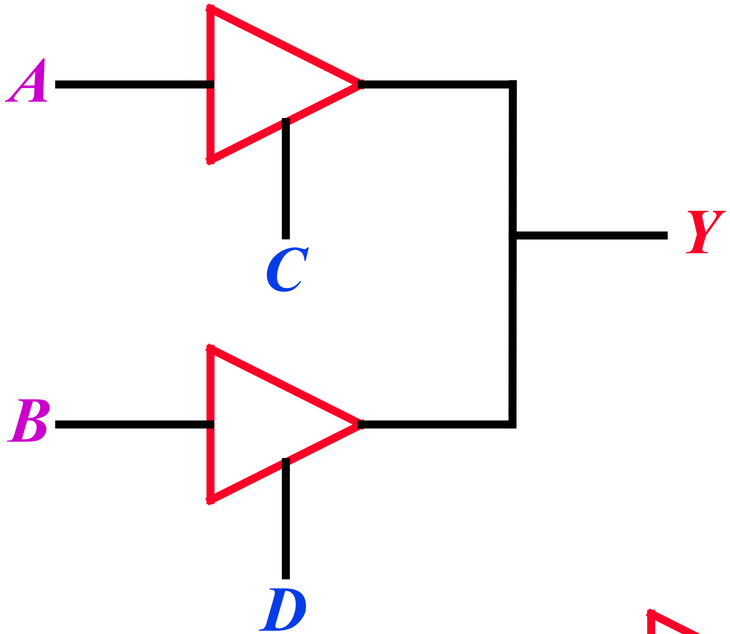
C	A	Y
0	x	Hi-Z
1	0	0
1	1	1



★ **Tri-State Inverter**

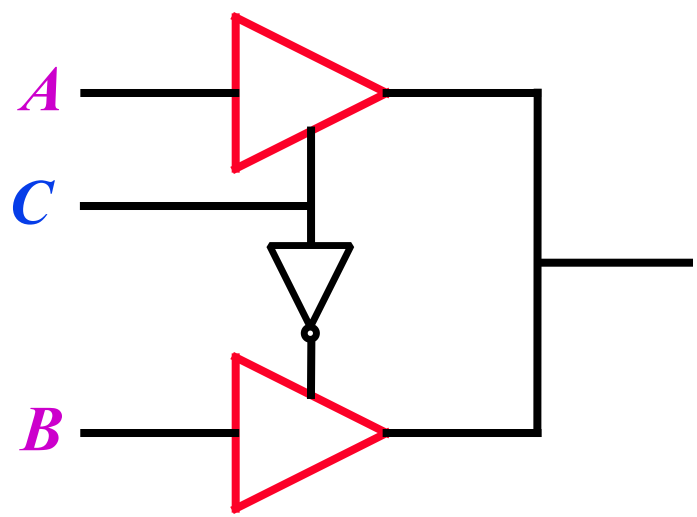


Three-State Gates



C *D*	*Y*
0 0	Hi-Z
0 1	B
1 0	A
1 1	?

Not Allowed



$$Y = \begin{cases} A & \text{if } C = 1 \\ B & \text{if } C = 0 \end{cases}$$

Three-State Gates