CS2022: 數位系統設計

Combinational Logic

Outline

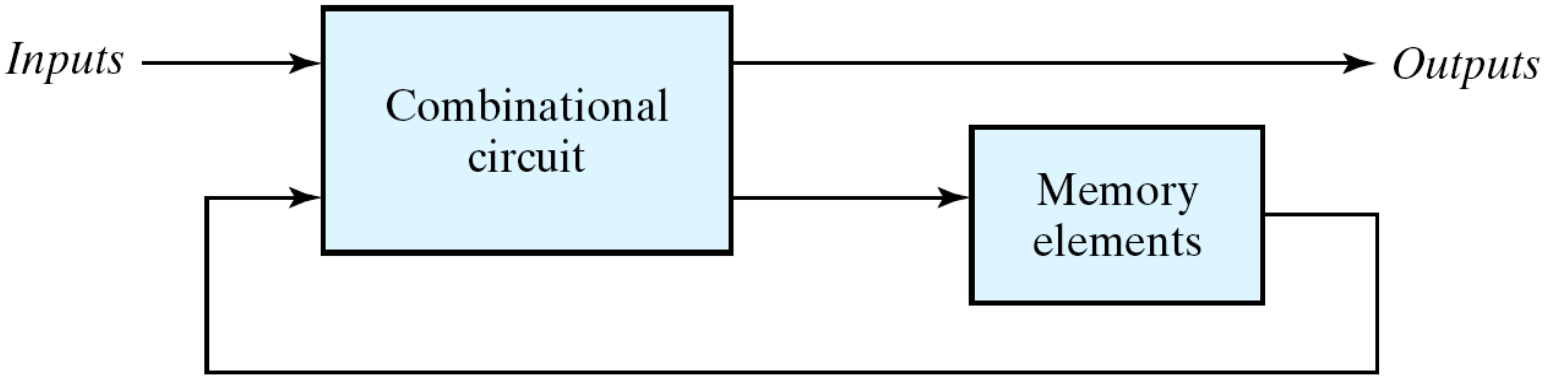
- Introduction
- Combinational Circuits
- Analysis Procedure
- Design Procedure
- Binary Adder-Subtractor
- Decimal Adder
- Binary Multiplier
- Magnitude Comparator
- Decoders
- Encoders
- Multiplexers

Introduction

- **Logic circuits for digital systems may be combinational or sequential**
- **A combinational circuit consists of logic gates whose outputs at any time are determined from only the present combination of inputs**

Combinational Circuits

- **Logic circuits for digital system**
  - **Sequential circuits**
    - **Contain memory elements**
    - **The outputs are a function of the current inputs and the state of the memory elements**
    - **The outputs also depend on past inputs**



Combinational Circuits

- **A combinational circuits**
  - **2^n possible combinations of input values**

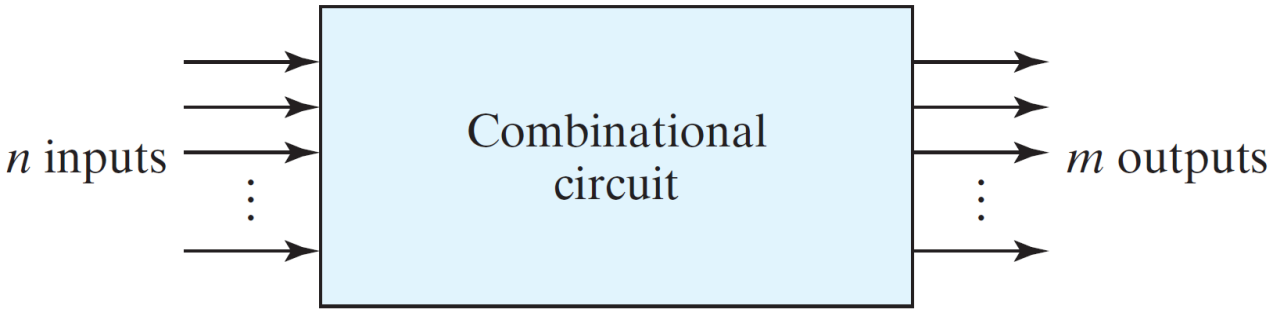


Figure 1 Block diagram of combinational circuit

  - **Specific functions**
    - » **Adders, subtractors, comparators, decoders, encoders, and multiplexers**
    - » **MSI circuits or standard cells**

Analysis Procedure

- **Analysis procedure for combinational circuit**
  - **Make sure that it is combinational not sequential**
    - **No feedback path**
  - **Derive its Boolean functions (truth tables)**
  - **Design verification**
  - **A verbal explanation of its function**

A Straight-forward Procedure

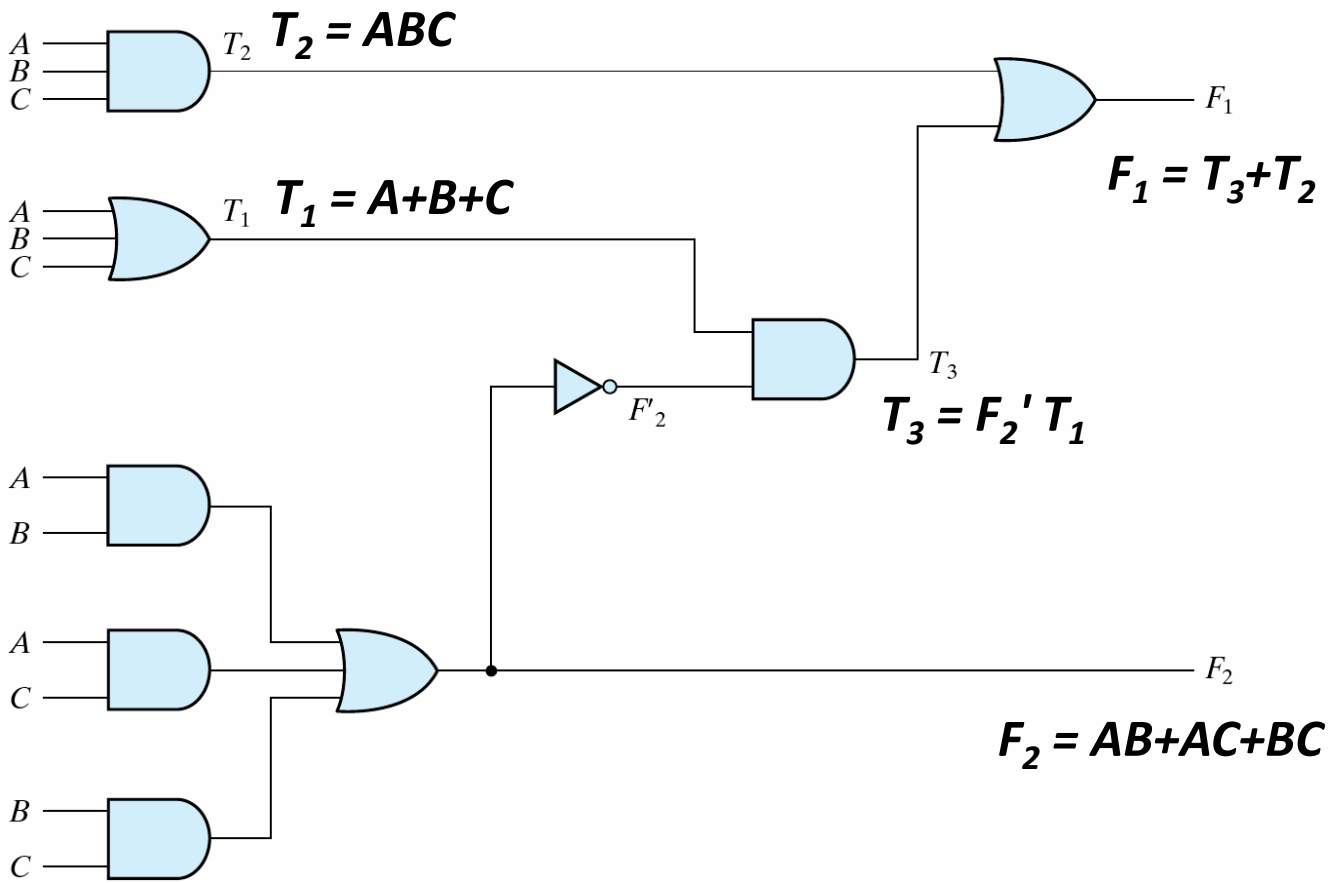
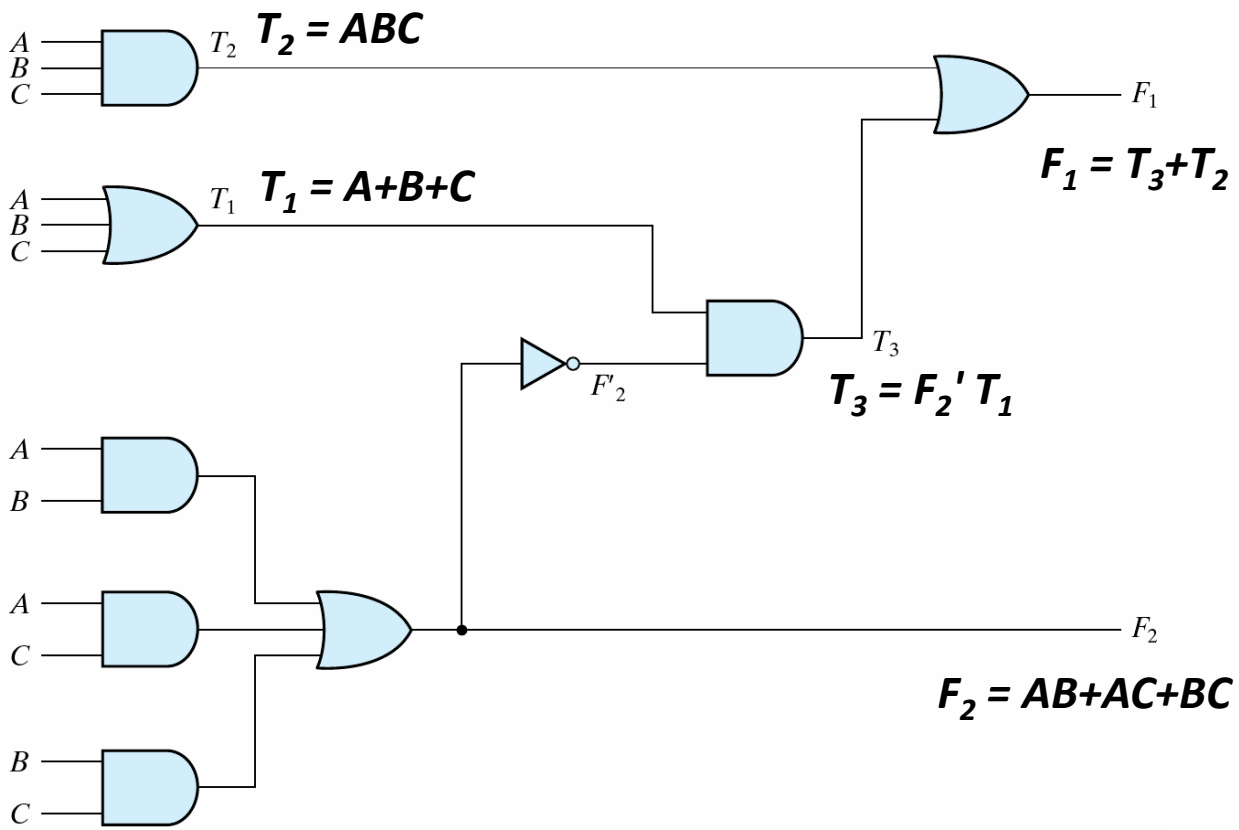


Figure 2 Logic Diagram for Analysis Example

A Straight-forward Procedure

- $F_1 = T_3+T_2 = F_2'T_1+ABC = (AB + AC + BC)'(A + B + C) + ABC = (A' + B')(A' + C')(B' + C')(A + B + C) + ABC = (A' + B'C')(AB' + AC' + BC' + B'C) + ABC = A'BC' + A'B'C + AB'C' + ABC$



Truth Table for Logic Diagram

- **Enumerate input combinations for the truth table**

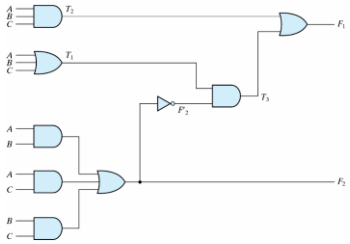


Table 4.1
Truth Table for the Logic Diagram of Fig. 4.2

A	B	C	F_2	F'_2	T_1	T_2	T_3	F_1
0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	0	1	1
0	1	0	0	1	1	0	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	1	1	0	1	1
1	0	1	1	0	1	0	0	0
1	1	0	1	0	1	0	0	0
1	1	1	1	0	1	1	0	1

Design Procedure

- **The design procedure of combinational circuits**
  - **State the problem (system specification (spec.))**
  - **Determine the inputs and outputs**
  - **The input and output variables are assigned symbols**
  - **Derive the truth table**
  - **Derive the simplified Boolean functions**
  - **Draw the logic diagram and verify the correctness**

Design Procedure

- **Functional description**
  - **Boolean function**
  - **HDL (Hardware description language)**
    - **Verilog HDL**
    - **VHDL**
  - **Schematic entry**
- **Design objectives and constraints**
  - **Number of gates**
  - **Number of inputs to a gate**
  - **Propagation delay**
  - **Number of interconnection**
  - **Limitations of the driving capabilities**

Code Conversion Example

- **BCD to excess-3 code**
  - **The truth table**

Table 4.2
Truth Table for Code-Conversion Example

Input BCD				Output Excess-3 Code			
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

BCD Maps

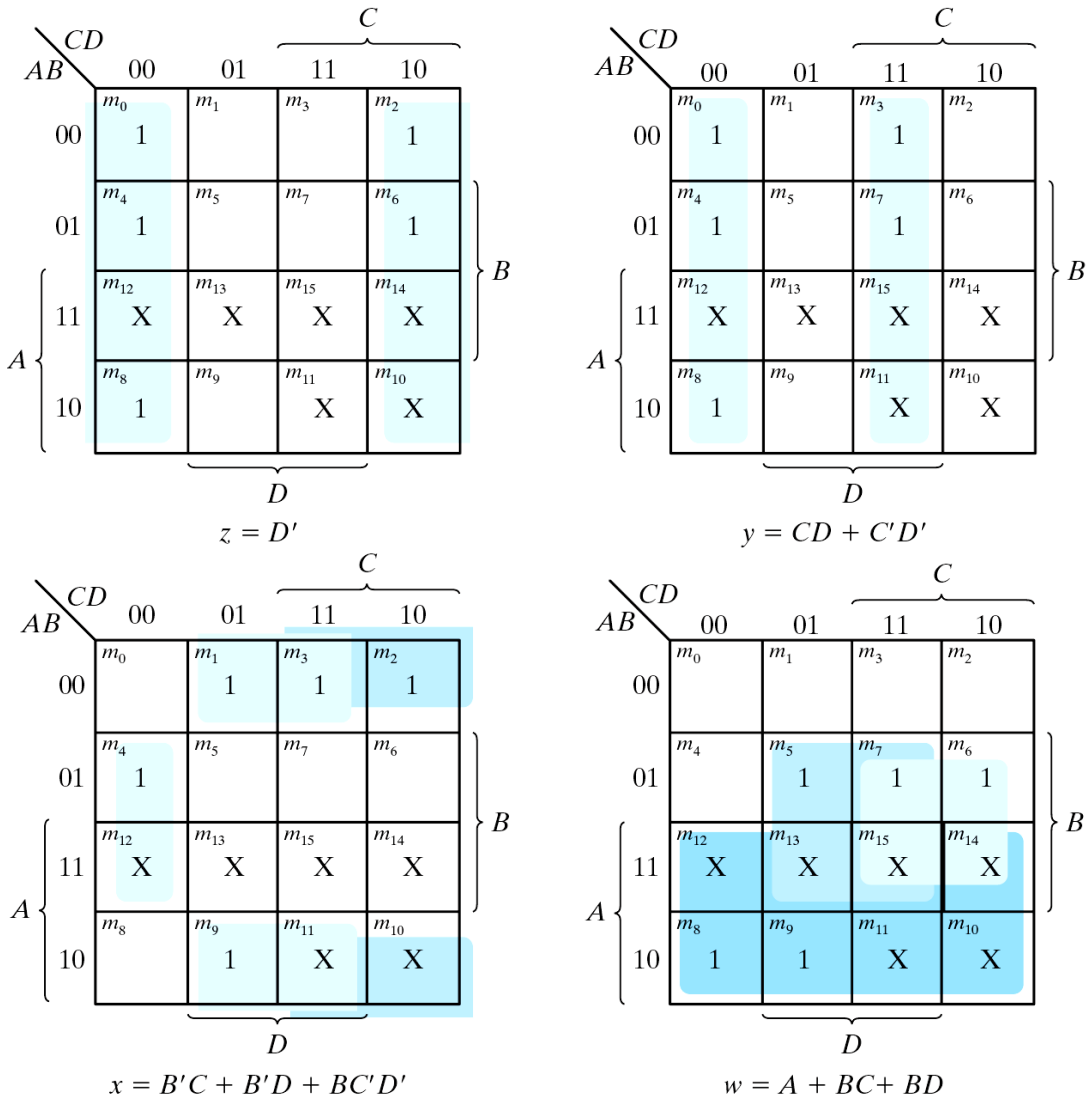
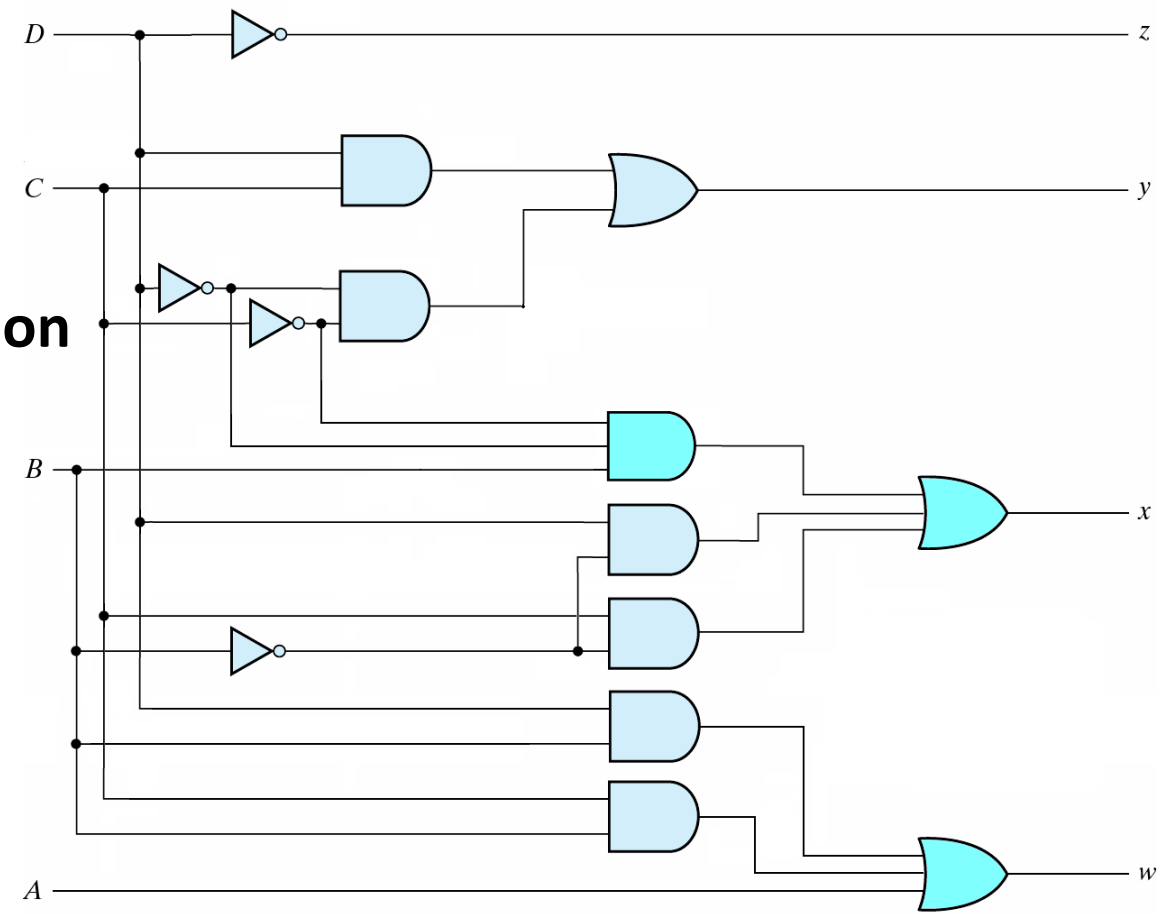


Figure 3 Maps for BCD to Excess-3 Code Converter

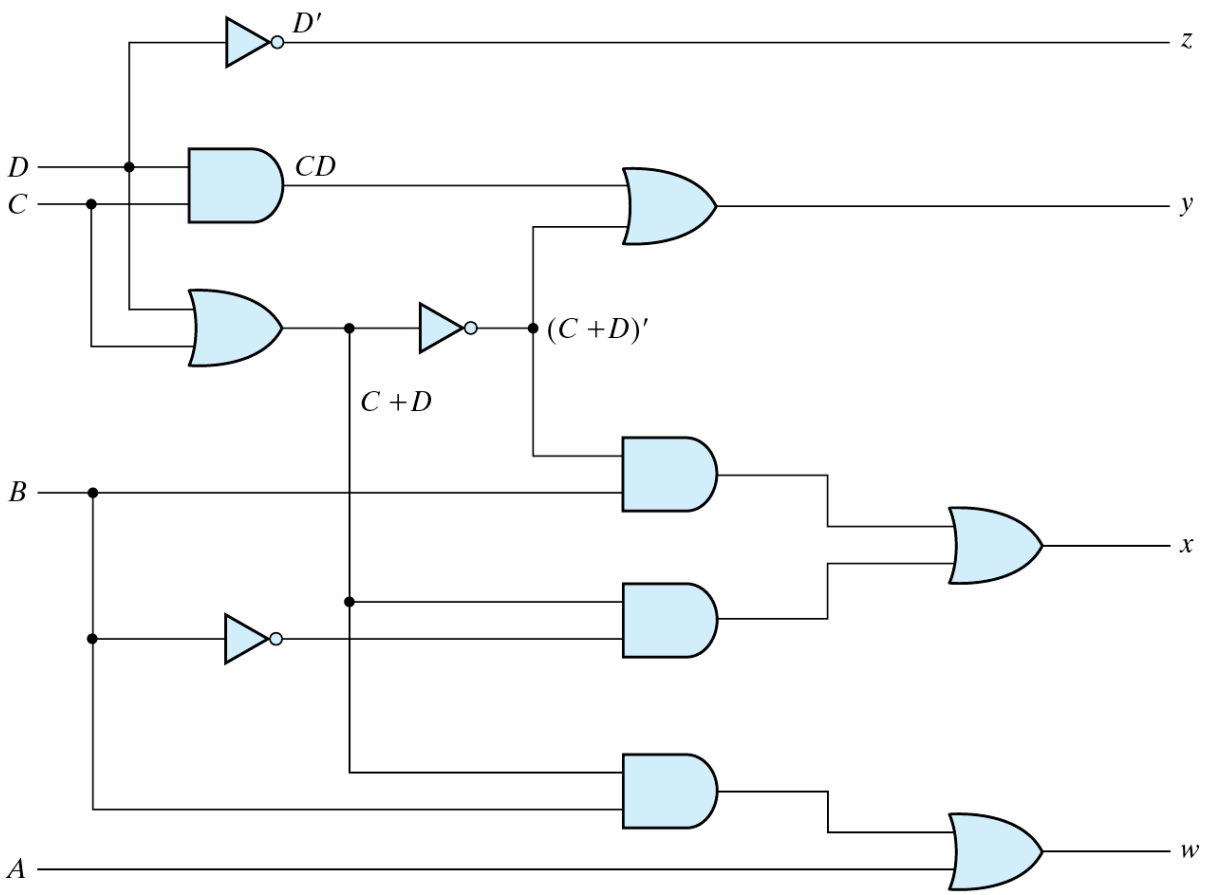
BCD to Excess-3 Functions

- **The simplified functions of two-level implementation**
  - $z = D'$
  - $y = CD + C'D'$
  - $x = B'C + B'D + BC'D'$
  - $w = A + BC + BD$
- **Another implementation**
  - $z = D'$
  - $y = CD + (C + D)'$
  - $x = B'(C + D) + B(C + D)'$
  - $w = A + B(C + D)$



Yet Another BCD to Excess-3

- **The logic diagram of multi-level implementation**



$z = D'$

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

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

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

Fig. 4 Logic Diagram for BCD to Excess-3 Code Converter

Binary Adder-Subtractor

- **Half adder**
  - **0 + 0 = 0 ; 0 + 1 = 1 ; 1 + 0 = 1 ; 1 + 1 = 10**
  - **Two input variables: *x*, *y***
  - **Two output variables: *C* (carry), *S* (sum)**
  - **Truth table**

Table 4.3
Half Adder

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

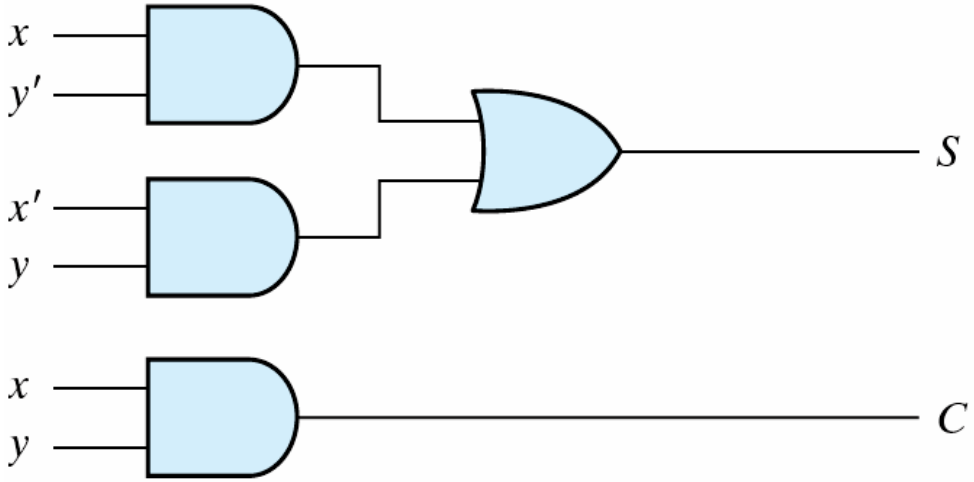
Half Adder

- **Simplified sum-of-products**
  - $S = x'y + xy'$
  - $C = xy$
- **The flexibility for implementation**
  - $S = x \oplus y$
  - $S = (x + y)(x' + y')$
  - $S' = xy + x'y'$
  - $S = (C + x'y')'$
  - $C = xy = (x' + y')'$

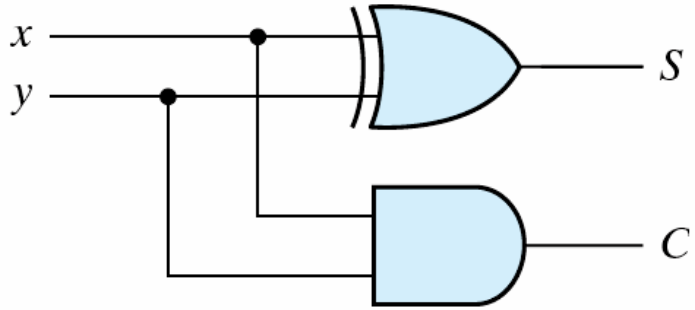
Table 4.3
Half Adder

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

Implementation of Half-Adder



(a) $S = xy' + x'y$
$C = xy$



(b) $S = x \oplus y$
$C = xy$

Figure 5 Implementation of Half-Adder

Full-Adder

- **Full-Adder**
  - **The arithmetic sum of three input bits**
  - **Three input bits**
    - ***x*, *y*: two significant bits**
    - ***z*: the carry bit from the previous lower significant bit**
  - **Two output bits: *C*, *S***

Table 4.4
Full Adder

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

Full-Adder

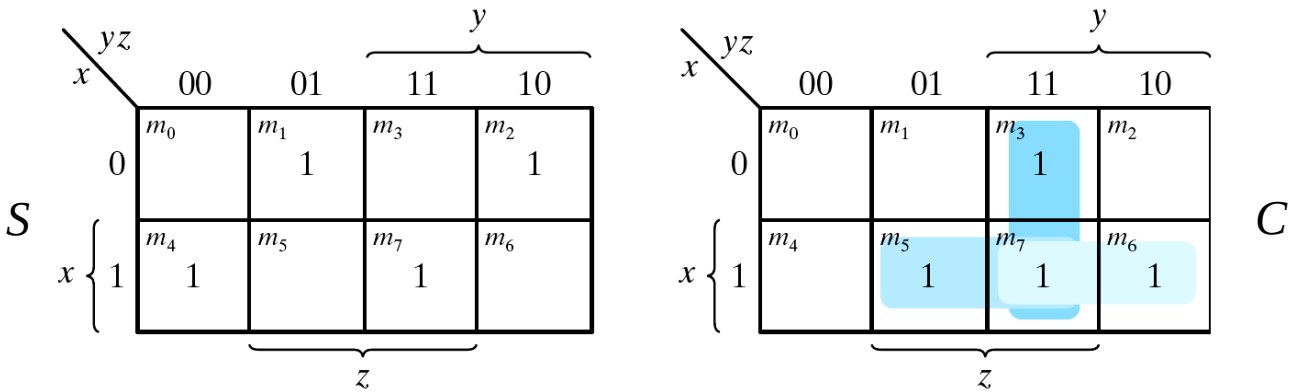


Fig. 6 Map for Full Adder

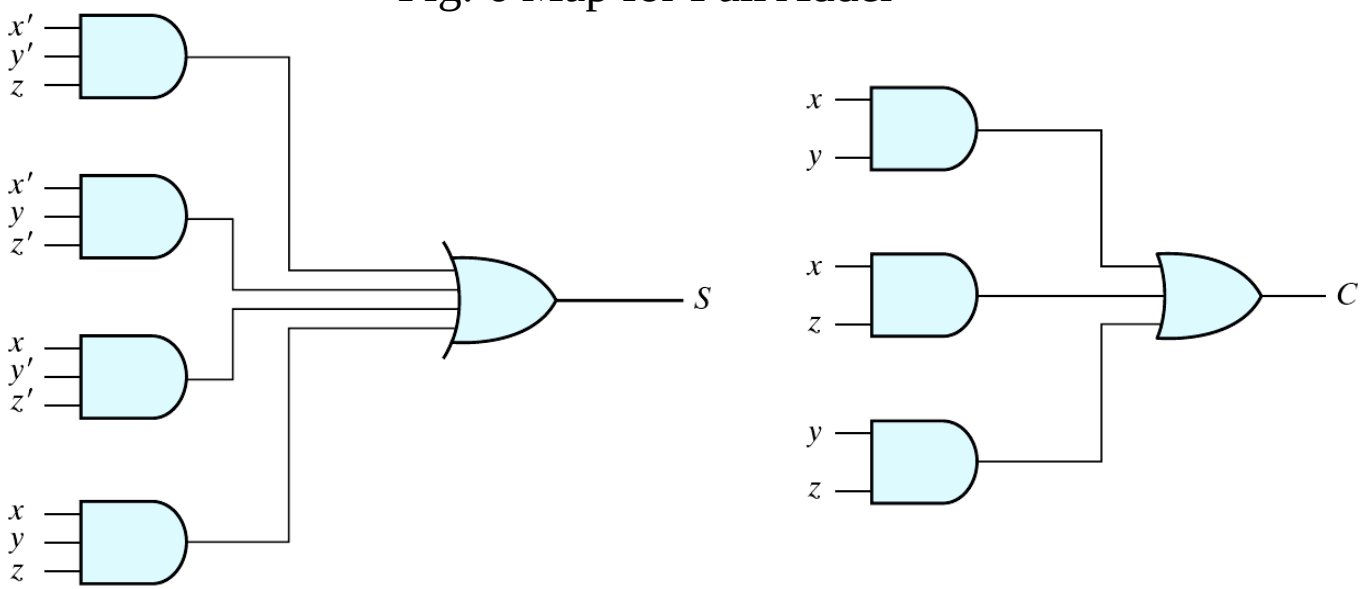


Fig. 7 Implementation of Full Adder in Sum of Products

Full-Adder

- **Simplified sum-of-products**
  - $S = x'y'z + x'yz' + xy'z' + xyz$
  - $C = xy + xz + yz$
- **Full-adder by using 2 half-adders**
  - $S = z \oplus (x \oplus y) = z'(xy' + x'y) + z(xy' + x'y)' = z'xy' + z'x'y + z((x' + y)(x + y')) = xy'z' + x'yz' + xyz + x'y'z$
  - $C = z(xy' + x'y) + xy = xy'z + x'yz + xy$

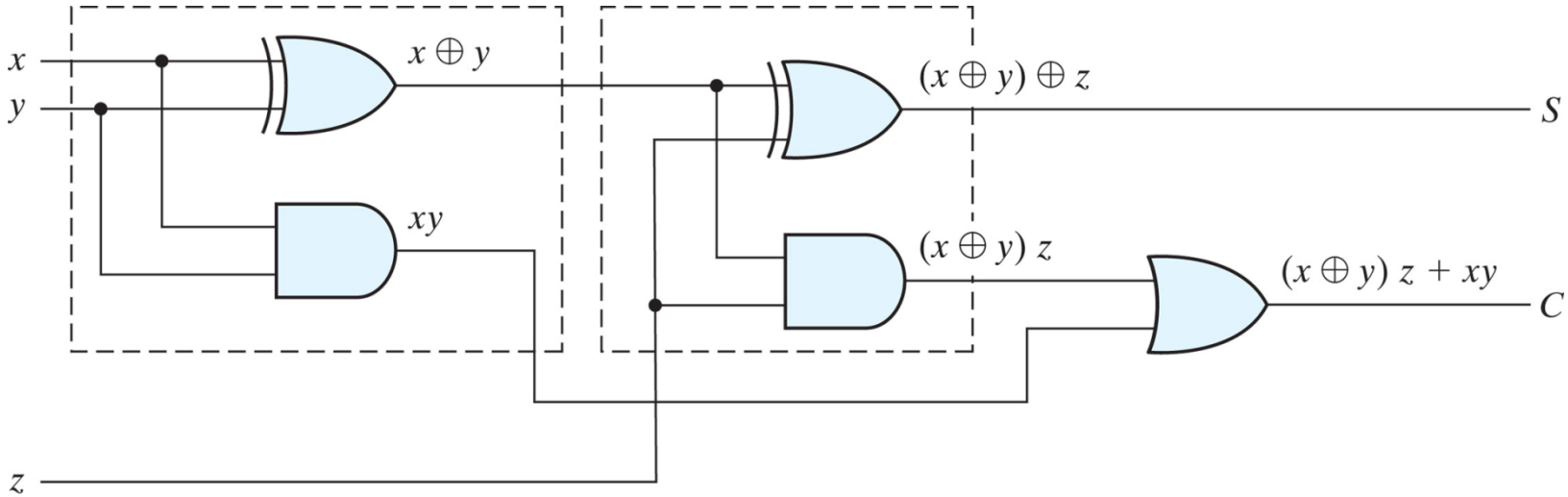


Fig. 8 Implementation of Full Adder with Two Half Adders and an OR Gate

Binary (Ripple-Carry) Adder

Subscript i:	3	2	1	0	
Input carry				0	C_i
Augend	1	0	1	1	A_i
Addend	0	0	1	1	B_i
Sum					S_i
Output carry					C_{i+1}

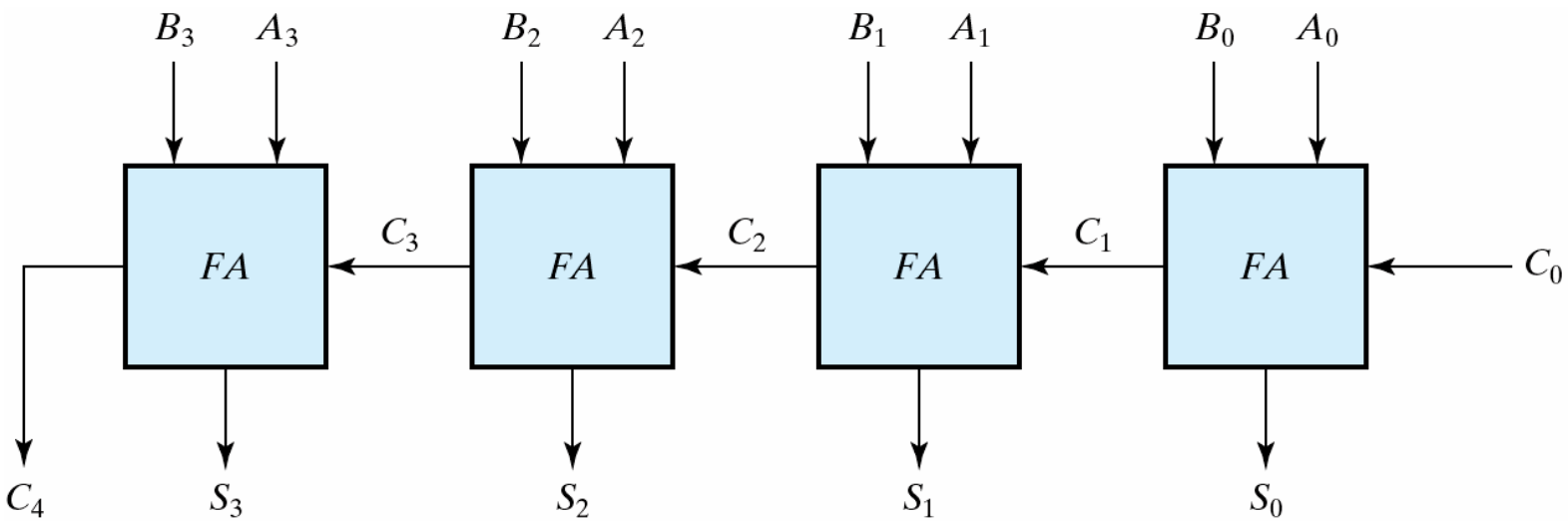


Figure 9 Four-bit adder

Carry Propagation

- **Carry propagation**
  - **When the correct outputs are available?**
  - **The critical path counts (the worst case)**
  - **(A_0, B_0, C_0) → C_1 → C_2 → C_3 → (C_4, S_3)**
  - **4-bit adder → 8 gate levels (*n*-bit adder: 2*n* gate levels)**

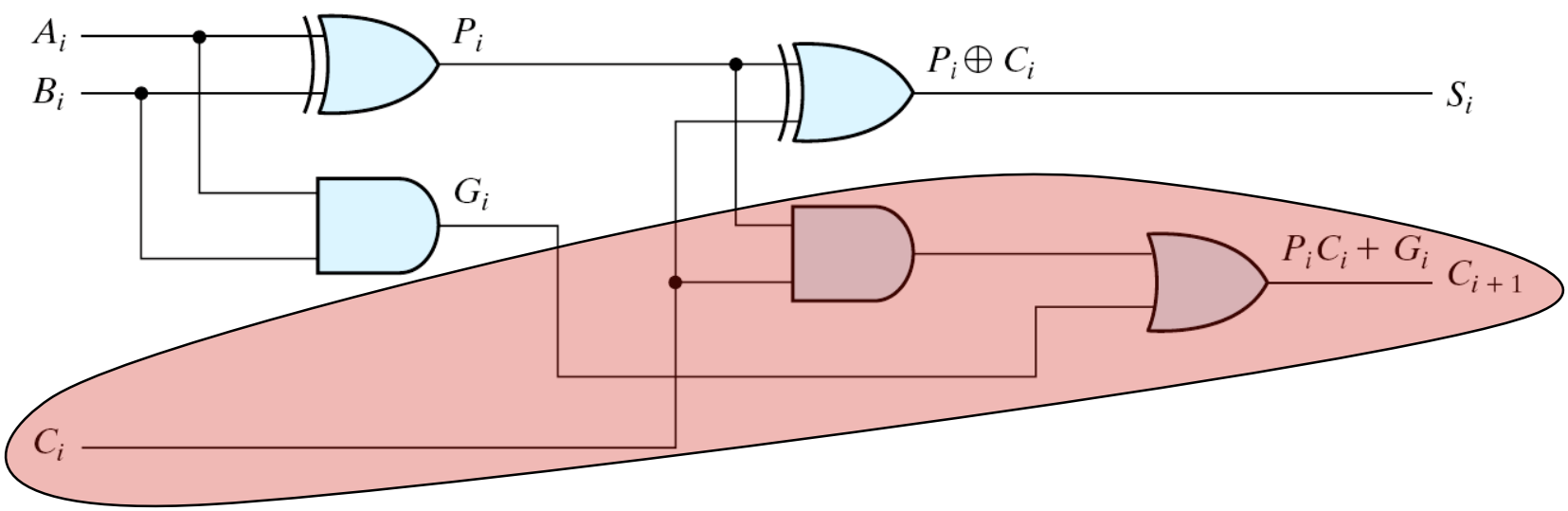
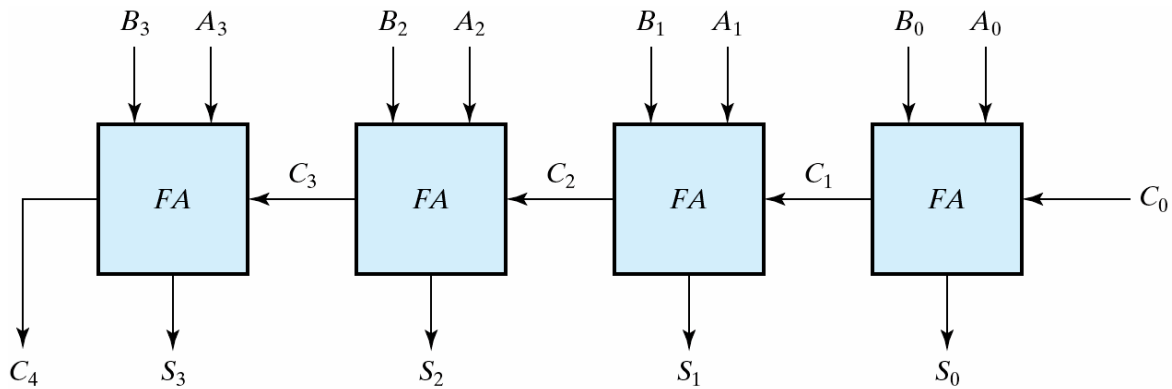
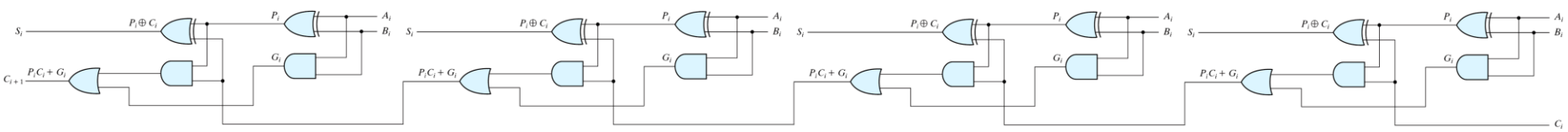


Figure 10 Full Adder with *P* and *G* Shown

Carry Propagation

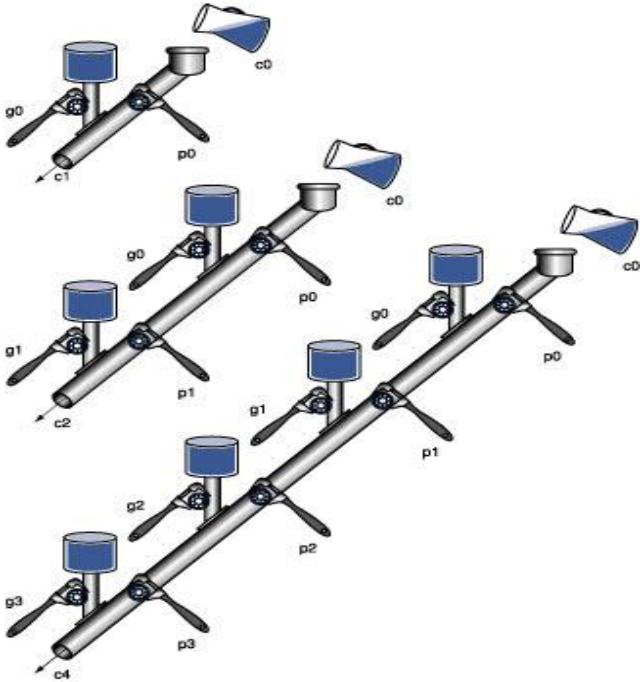
- **Carry propagation**
  - **When the correct outputs are available?**
  - **The critical path counts (the worst case)**
  - **$(A_0, B_0, C_0) \rightarrow C_1 \rightarrow C_2 \rightarrow C_3 \rightarrow (C_4, S_3)$**
  - **4-bit adder → 8 gate levels (*n*-bit adder: 2*n* gate levels)**





Parallel Adders

- **Reduce the carry propagation delay**
  - **Using faster gates**
  - **Parallel adders (e.g., carry look-ahead adder)**
- **Carry look-ahead adder (CLA)**
  - **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**
  - $C_1 = G_0 + P_0 C_0$
  - $C_2 = G_1 + P_1 C_1 = G_1 + P_1(G_0 + P_0 C_0) = G_1 + P_1 G_0 + P_1 P_0 C_0$
  - $C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$
  - $C_4 = G_3 + P_3 C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_0$



Carry Look-ahead Adder (1/2)

- **Logic diagram of carry look-ahead generator**

Carry propagate: $P_i = A_i \oplus B_i$
Carry generate: $G_i = A_i B_i$

C_0 = input carry
$C_1 = G_0 + P_0 C_0$
$C_2 = G_1 + P_1 C_1 = G_1 + P_1 G_0 + P_1 P_0 C_0$
$C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$

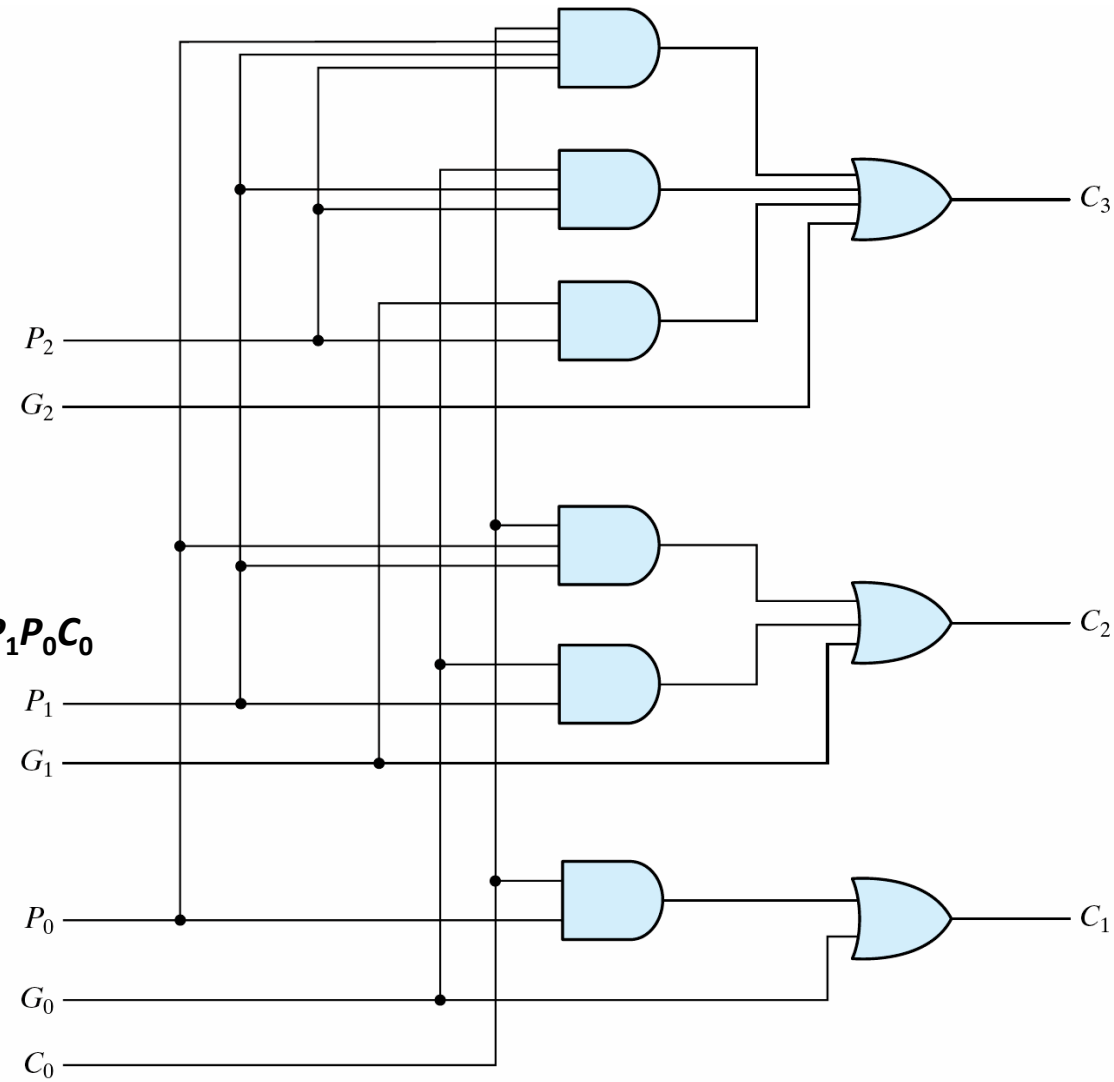


Fig. 11 Logic Diagram of Carry Look-ahead Generator

Carry Look-ahead Adder (2/2)

- **4-bit carry-look ahead adder**
  - **Propagation delay of C_3, C_2 and C_1 are equal**
  - **Propagation delay of S_3, S_2 and S_1 are equal**

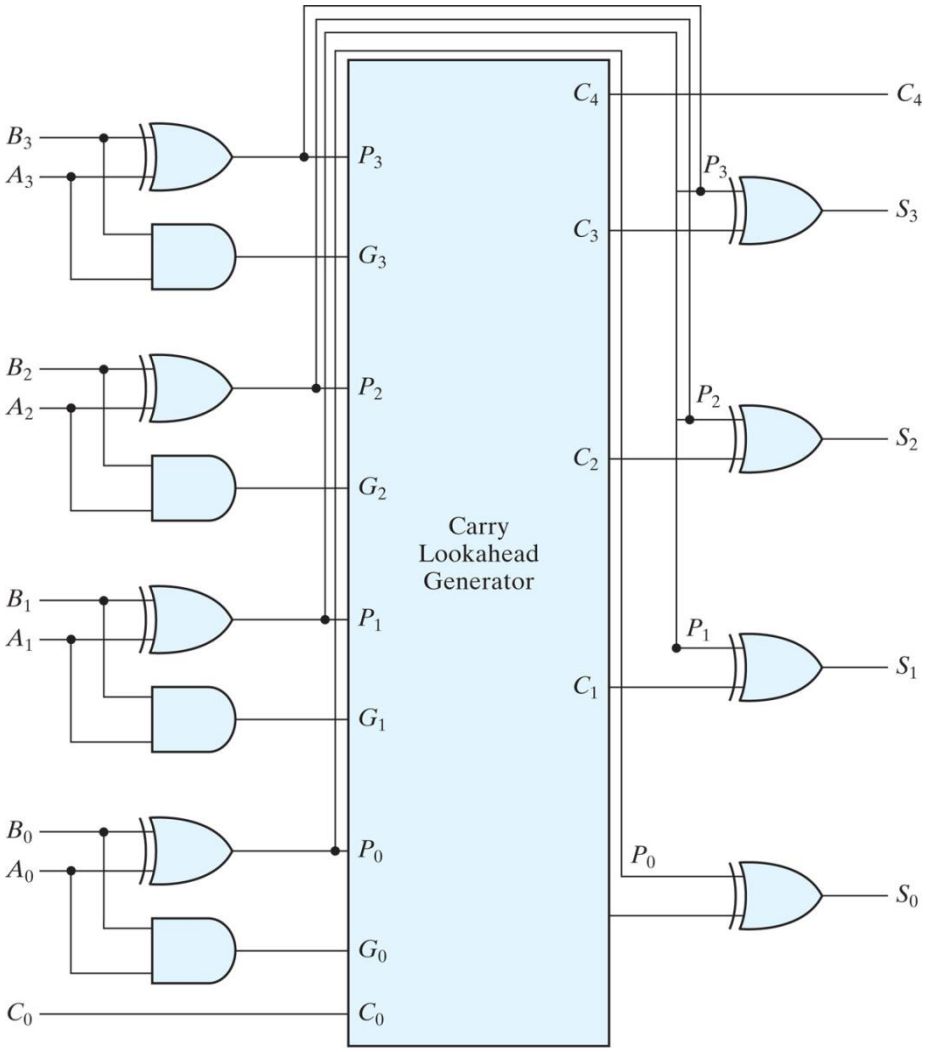
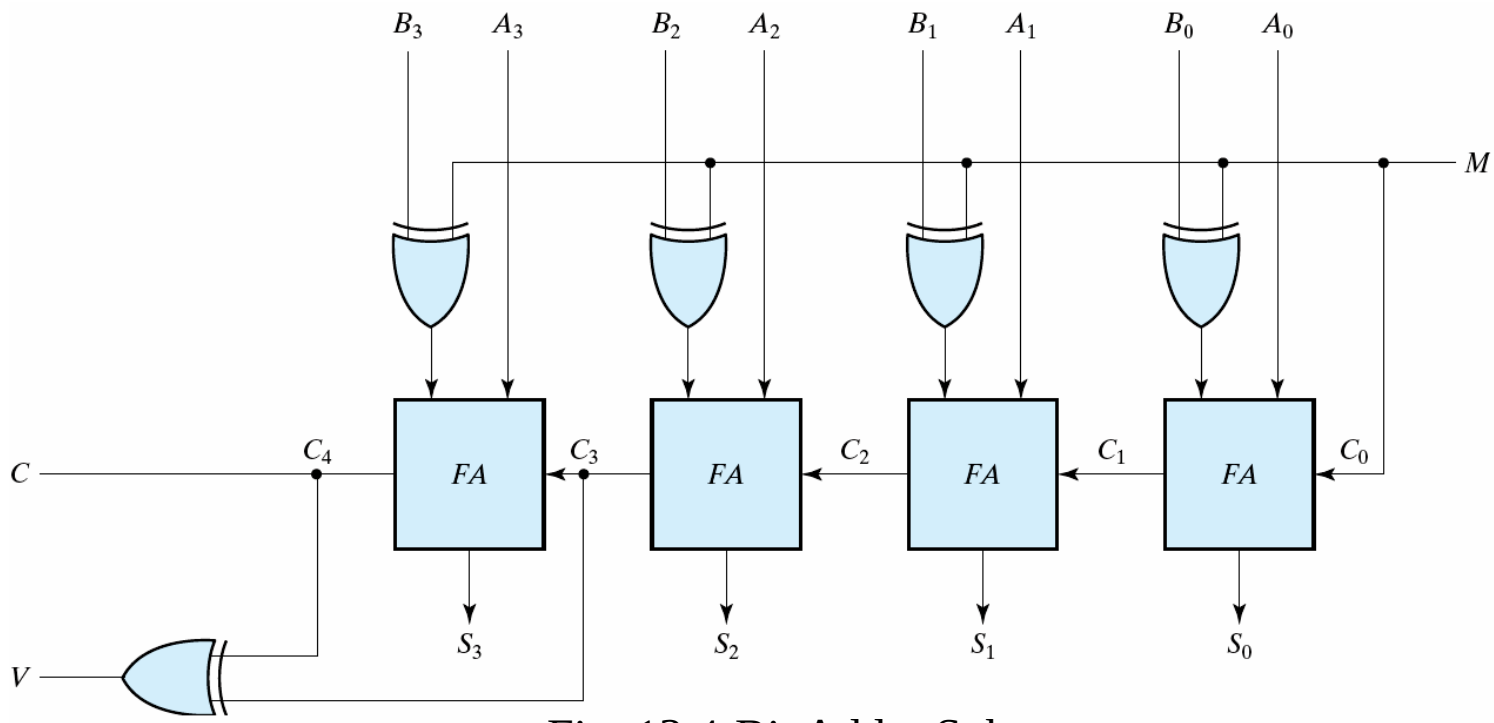


Fig. 12 4-Bit Adder with Carry Look-ahead

Signed Binary Subtractor

- Signed binary subtraction is performed by adding the minuend to the 2's complement of the subtrahend
- 4-bit adder-subtractor
  - M=0: *A + B; A + B + 0*
  - M=1: *A – B; A + B′ + 1*



Overflow detection

Fig. 13 4-Bit Adder Subtractor

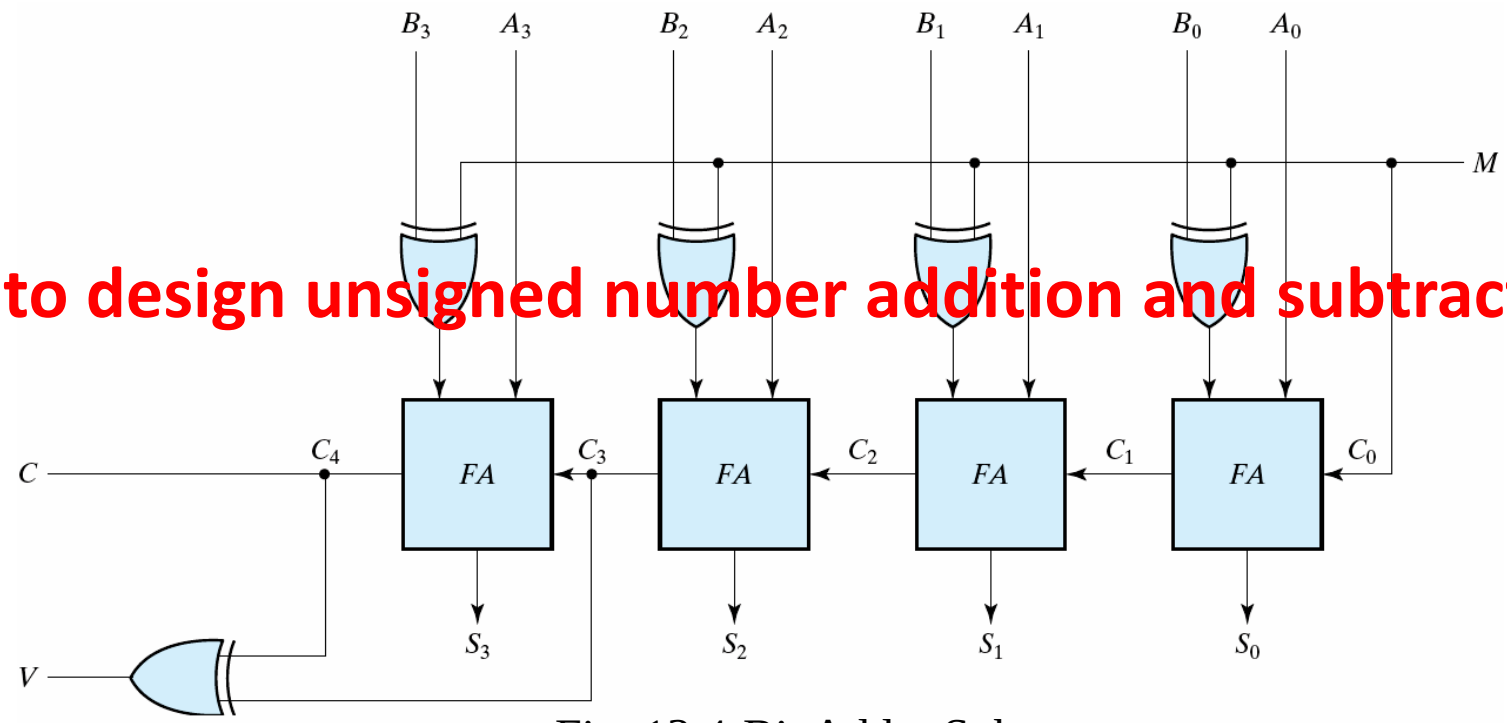
Overflow

- **Since the number of bits is limited, *overflow* occurs when the resulting value of an operation is out of the range of valid values. That is, the resulting value is greater than max or less than min.**
  - **Add two positive numbers and obtain a negative number**
  - **Add two negative numbers and obtain a positive number**
  - ***V* = 0, no overflow; *V* = 1, overflow**
- **Example:**

carries:	0 1		carries:	1 0	
+70	0	1000110	−70	1	0111010
+80	0	1010000	−80	1	0110000
+150	1	0010110	−150	0	1101010

Unsigned Binary Subtractor

- Unsigned binary subtraction can also be performed by adding the minuend to the 2's complement of the subtrahend
- 4-bit adder-subtractor C_4
  - M=0: *A + B; A + B + 0*
  - M=1: *A – B; A + B′ + 1*



How to design unsigned number addition and subtraction?

Overflow detection

Fig. 13 4-Bit Adder Subtractor

Decimal Adder

- **Add two BCD's**
  - **9 inputs: two BCD's and one carry-in**
  - **5 outputs: one BCD and one carry-out**
- **Design approaches**
  - **A truth table with 2^9 entries (many entries, 312 exactly, are don't cares)**
  - **Use 4-bit binary adder**
    - **The maximum sum ← 9 + 9 + 1 = 19**
    - **Binary to BCD**

BCD Adder (1/3)

- **BCD Adder: binary sum to BCD sum**

Table 4.5
Derivation of BCD Adder

Binary Sum					BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

BCD Adder (2/3)

- **Modifications are needed if the sum > 9**
  - **If C = 1, then sum > 9**
    - K = 1, or
    - Z_8Z_4 = 1 (11××), or
    - Z_8Z_2 = 1 (1×1×).
  - **Modification: $-(10)_d$ or + 6**

$$C = K + Z_8Z_4 + Z_8Z_2$$

BCD Adder (3/3)

- **Block diagram**

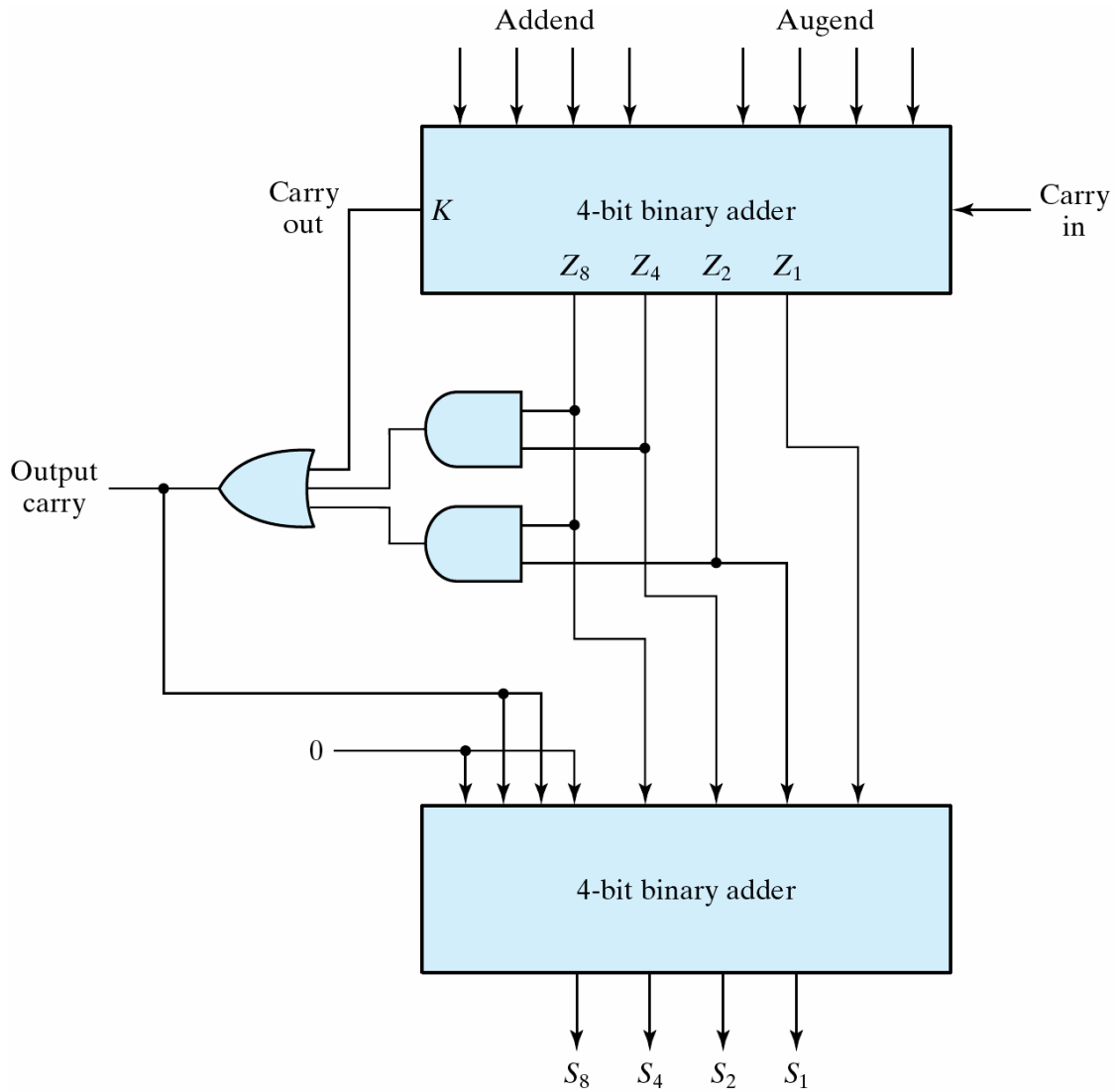


Fig. 14 Block Diagram of a BCD Adder

Binary Multiplier (1/3)

- **Multiplication consists of**
  - **Generation of partial products**
  - **Accumulation of shifted partial products**

						y_5	y_4	y_3	y_2	y_1	y_0	Multiplicand
						x_5	x_4	x_3	x_2	x_1	x_0	Multiplier
						x_0y_5	x_0y_4	x_0y_3	x_0y_2	x_0y_1	x_0y_0	
					x_1y_5	x_1y_4	x_1y_3	x_1y_2	x_1y_1	x_1y_0		
				x_2y_5	x_2y_4	x_2y_3	x_2y_2	x_2y_1	x_2y_0			Partial Products
			x_3y_5	x_3y_4	x_3y_3	x_3y_2	x_3y_1	x_3y_0				
		x_4y_5	x_4y_4	x_4y_3	x_4y_2	x_4y_1	x_4y_0					
	x_5y_5	x_5y_4	x_5y_3	x_5y_2	x_5y_1	x_5y_0						
p_{11}	p_{10}	p_9	p_8	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0	Product

Binary Multiplier (2/3)

- **Partial products**
  - **AND operations**

$$
\begin{array}{cccc}
 & & B_1 & B_0 \\
 & & A_1 & A_0 \\
\hline
 & & A_0B_1 & A_0B_0 \\
 & A_1B_1 & A_1B_0 & \\
\hline
C_3 & C_2 & C_1 & C_0
\end{array}
$$

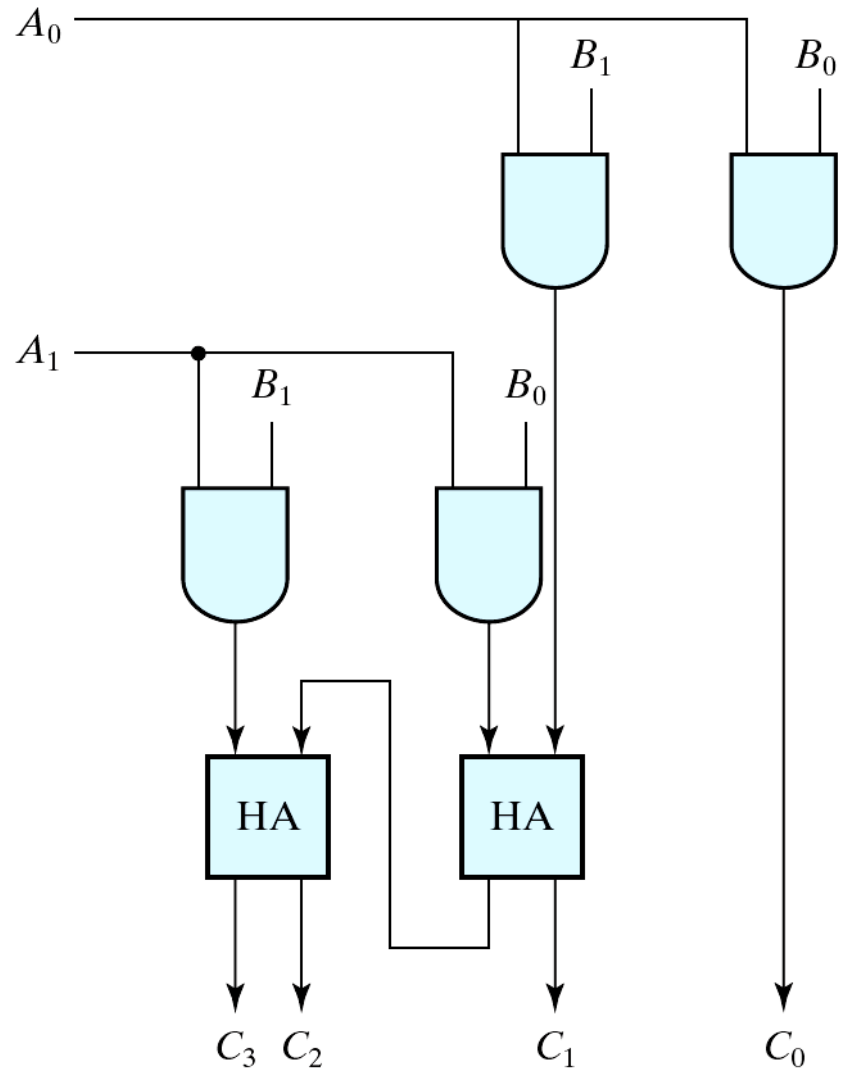


Fig. 15 Two-bit by two-bit binary multiplier

Binary Multiplier (3/3)

- **4-bit by 3-bit binary multiplier**

			B_3	B_2	B_1	B_0
				A_2	A_1	A_0
			A_0B_3	A_0B_2	A_0B_1	A_0B_0
		A_1B_3	A_1B_2	A_1B_1	A_1B_0	
	A_2B_3	A_2B_2	A_2B_1	A_2B_0		
C_6	C_5	C_4	C_3	C_2	C_1	C_0

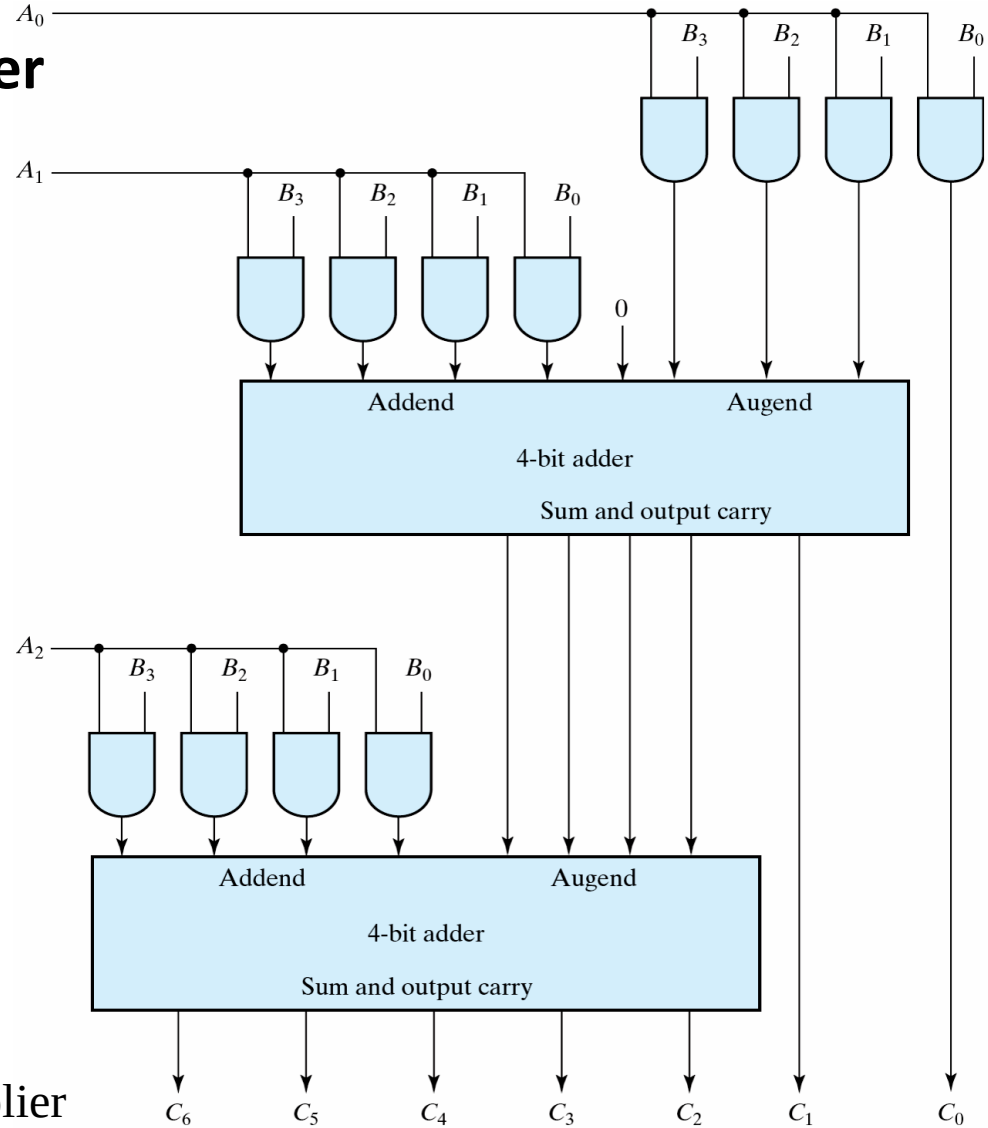
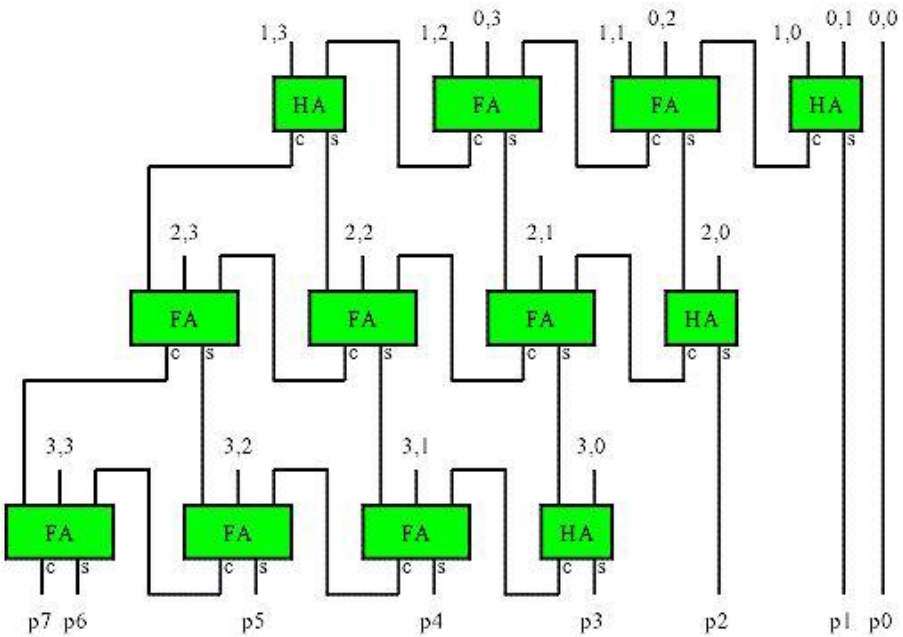
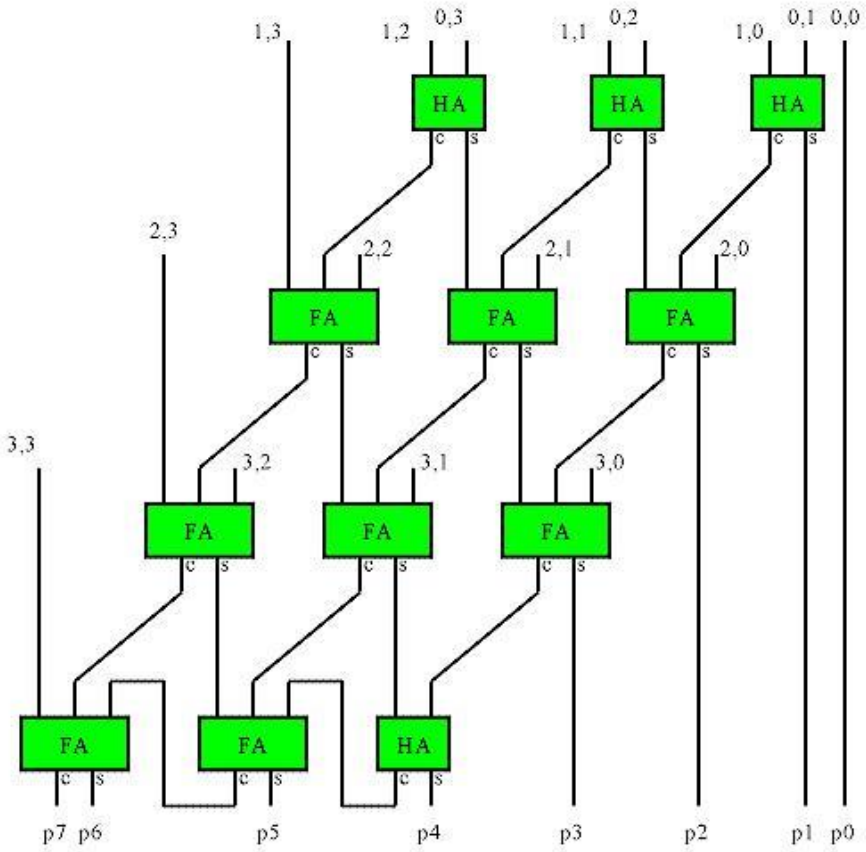


Fig. 16 Four-bit by three-bit binary multiplier

Various Multipliers





Magnitude Comparator (1/3)

- **The comparison of two unsigned numbers**
  - **Outputs: *A>B*, *A=B*, *A<B***
- **Design Approaches**
  - **The truth table of 2*n*-bit comparator**
    - » **2^{2n} entries - too cumbersome for large *n***
  - **Use inherent regularity of the problem**
    - » **Reduce design efforts**
    - » **Reduce human errors**

Magnitude Comparator (2/3)

- **Algorithm → logic**
  - $A = A_3A_2A_1A_0$; $B = B_3B_2B_1B_0$
  - $A=B$
    - $A_3=B_3$, $A_2=B_2$, $A_1=B_1$, and $A_1=B_1$
      - Equality: $x_i = A_iB_i + A_i'B_i'$ (equivalence)
      - $(A=B) = x_3x_2x_1x_0=1$
  - $(A>B)$
    - $A_3B_3' + x_3A_2B_2' + x_3x_2A_1B_1' + x_3x_2x_1 A_0B_0'$
  - $(A<B)$
    - $A_3'B_3 + x_3A_2'B_2 + x_3x_2A_1'B_1 + x_3x_2x_1 A_0'B_0$
- **Implementation**
  - $x_i = (A_iB_i' + A_i'B_i)'$

Magnitude Comparator (3/3)

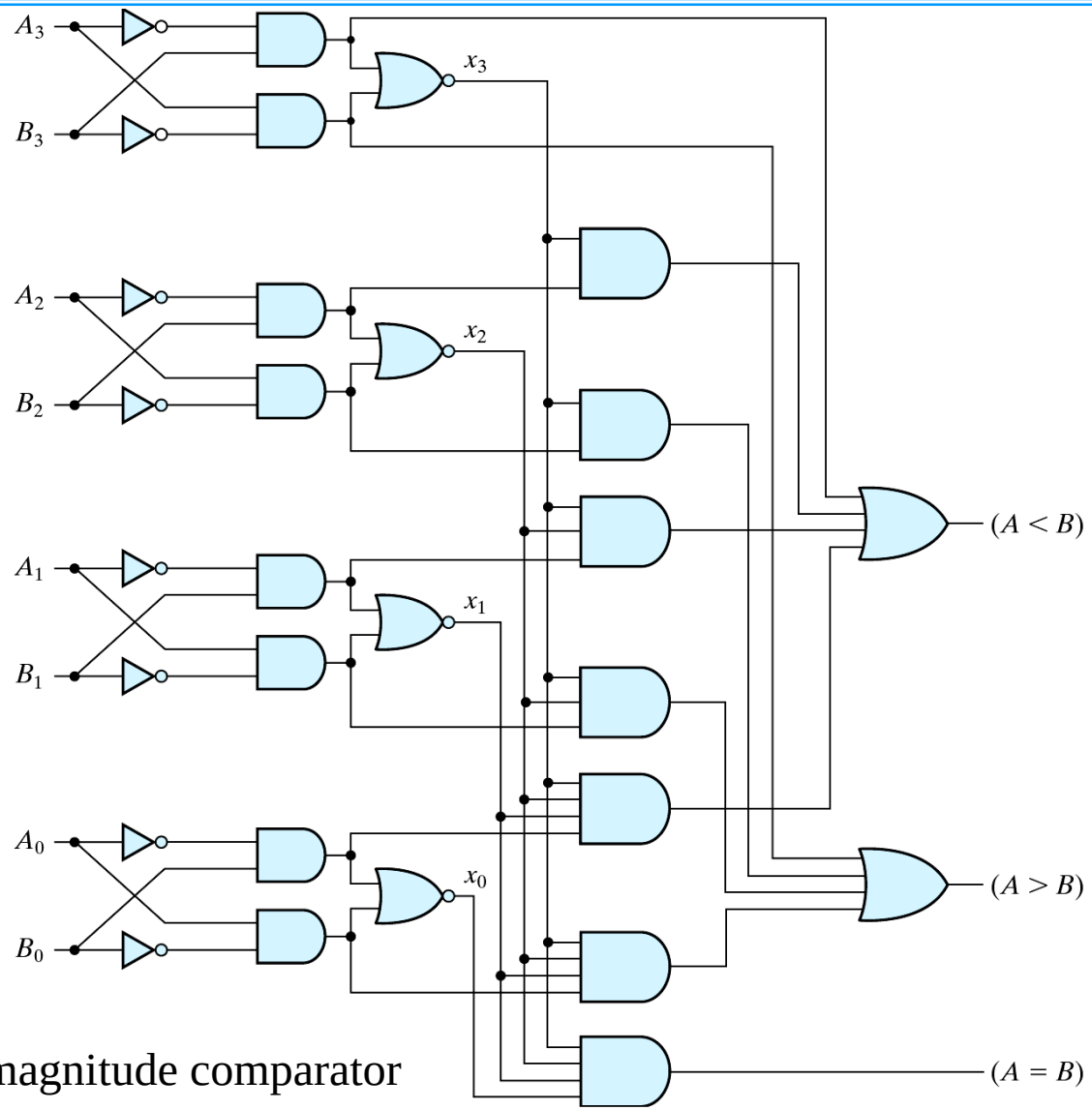


Fig. 17 Four-bit magnitude comparator

Decoder (1/2)

- A *n*-to-*m* decoder (active high)
  - A binary code of *n* bits = 2^n distinct information
  - *n* input variables; up to 2^n output lines
  - Only one output can be active (high) at any time

Table 4.6
Truth Table of a Three-to-Eight-Line Decoder

Inputs			Outputs							
x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Decoder (2/2)

- **An implementation**

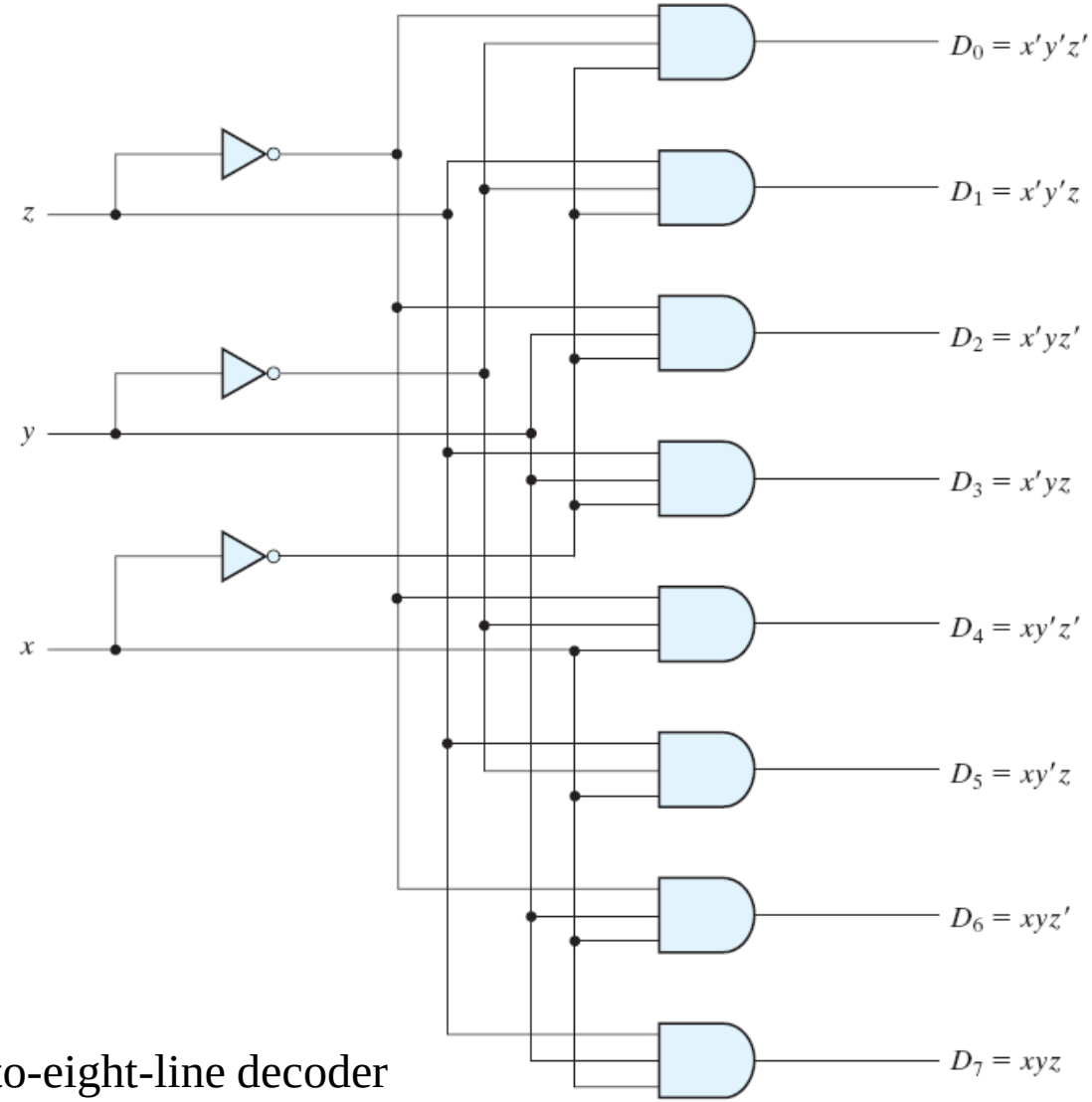


Fig. 18 Three-to-eight-line decoder

Decoder with Enable Input

- **Decoder with enable (active low)**
  - **A decoder with an enable input (using controlling value)**
  - **Receive information on a single line and transmits it on one of 2^n possible output lines**
  - **Only one output can be active (low) when enable (active-low)**

E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

Input don't care *X*: for all cases (0 and 1)

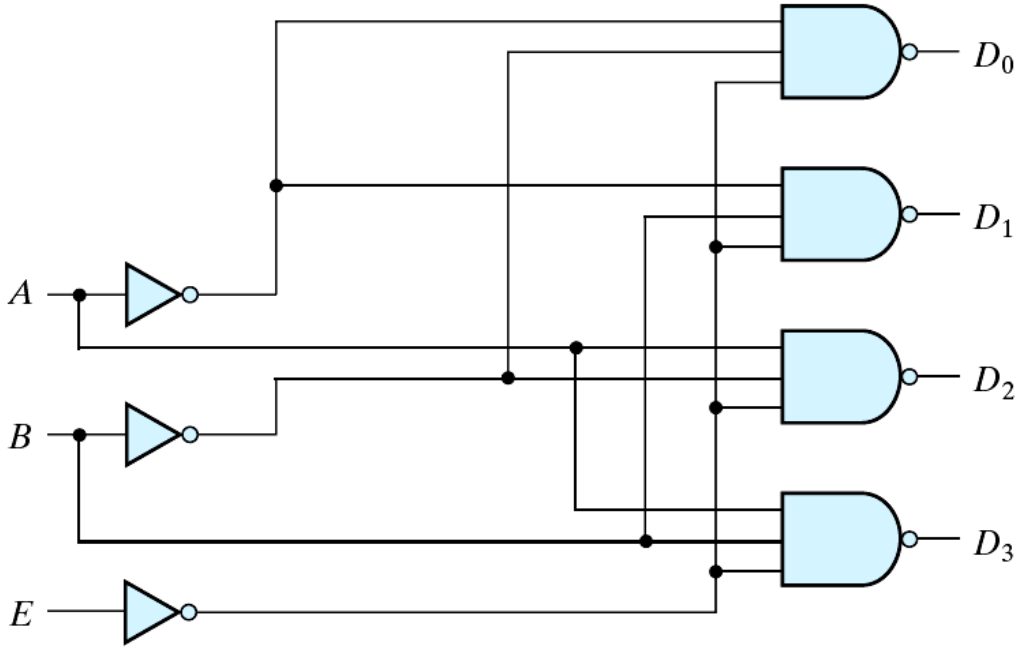


Fig. 19 Two-to-four-line decoder with enable input (NAND implementation)

Decoder Expansion

- **Decoder expansion**
  - **Two 3-to-8 decoder with enable: a 4-to-16 decoder**

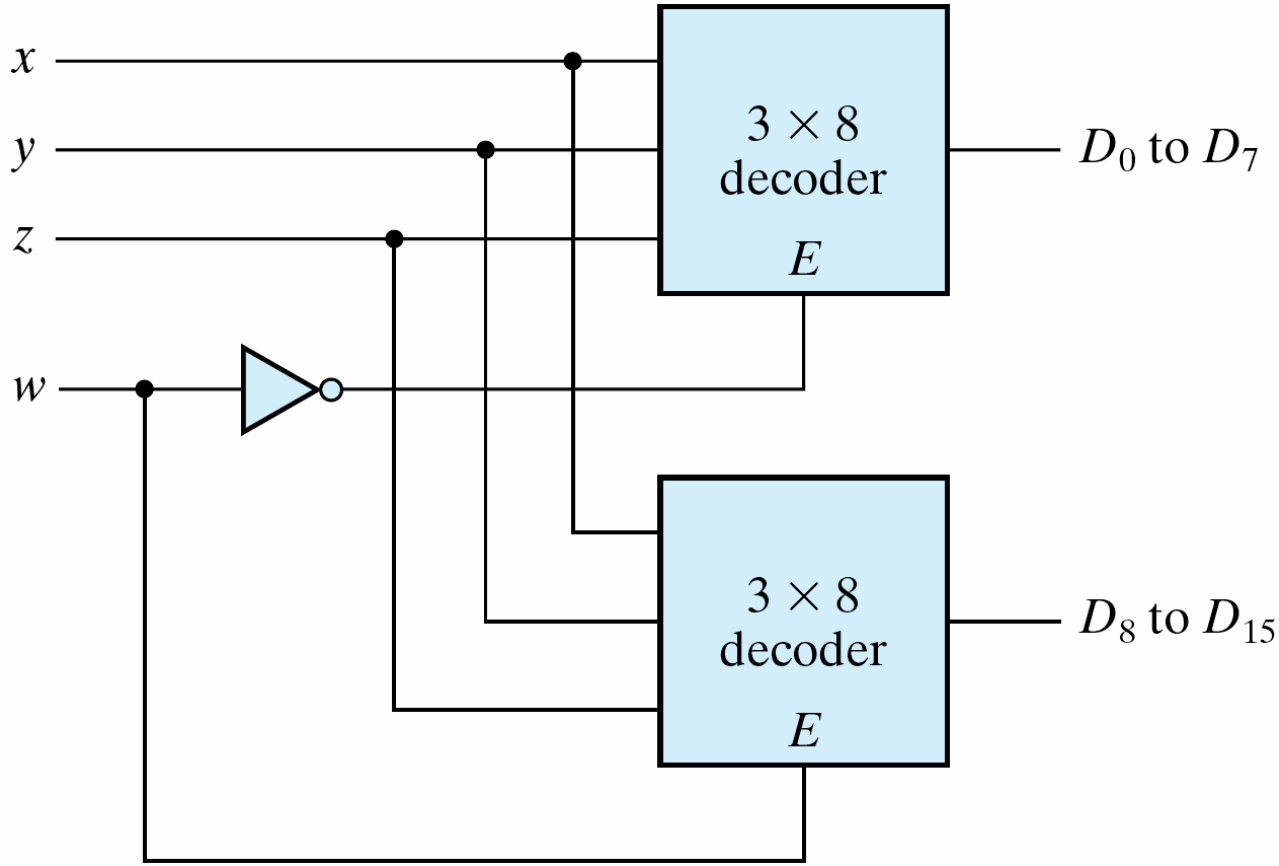


Fig. 20 4 × 16 decoder constructed with two 3 × 8 decoders

SOP Implementation (1/2)

- **Combinational logic implementation**
  - **Each output = a minterm**
  - **Use a decoder and an external OR gate to implement any Boolean function of *n* input variables**
  - **A full-adder**
    - » $S(x, y, z) = \Sigma(1, 2, 4, 7)$
    - » $C(x, y, z) = \Sigma(3, 5, 6, 7)$

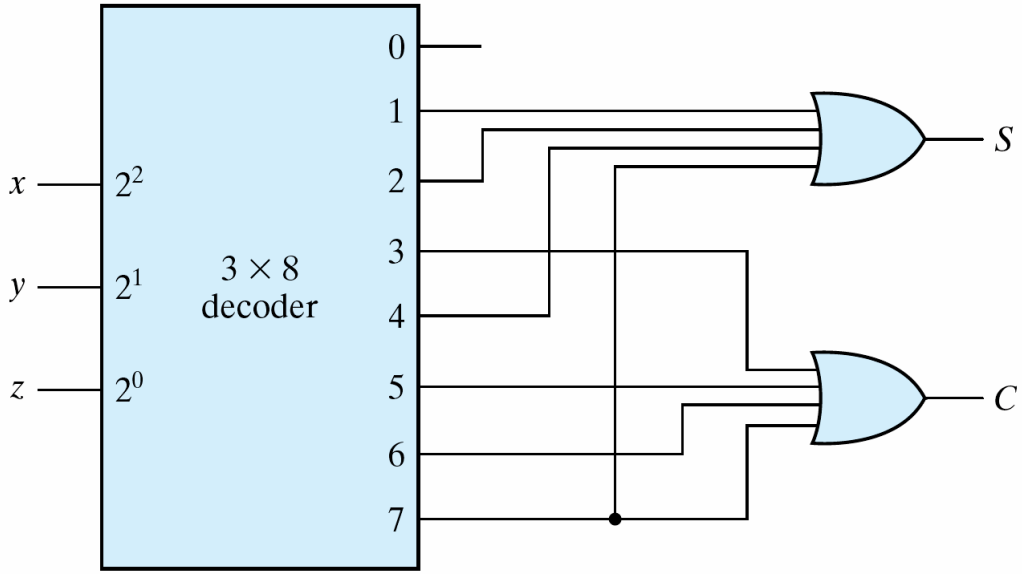


Fig. 21 Implementation of a full adder with a decoder

SOP Implementation (2/2)

- **Two possible approaches using decoder**
  - **OR (minterms of *F*): *k* inputs (*k* minterms)**
  - **NOR (minterms of *F'*): $2^n - k$ inputs**
- **In general, it is not a practical implementation for a large design**
  - **Too many minterms**

Encoders (1/2)

- **The inverse function of a decoder**

Table 4.7
Truth Table of an Octal-to-Binary Encoder

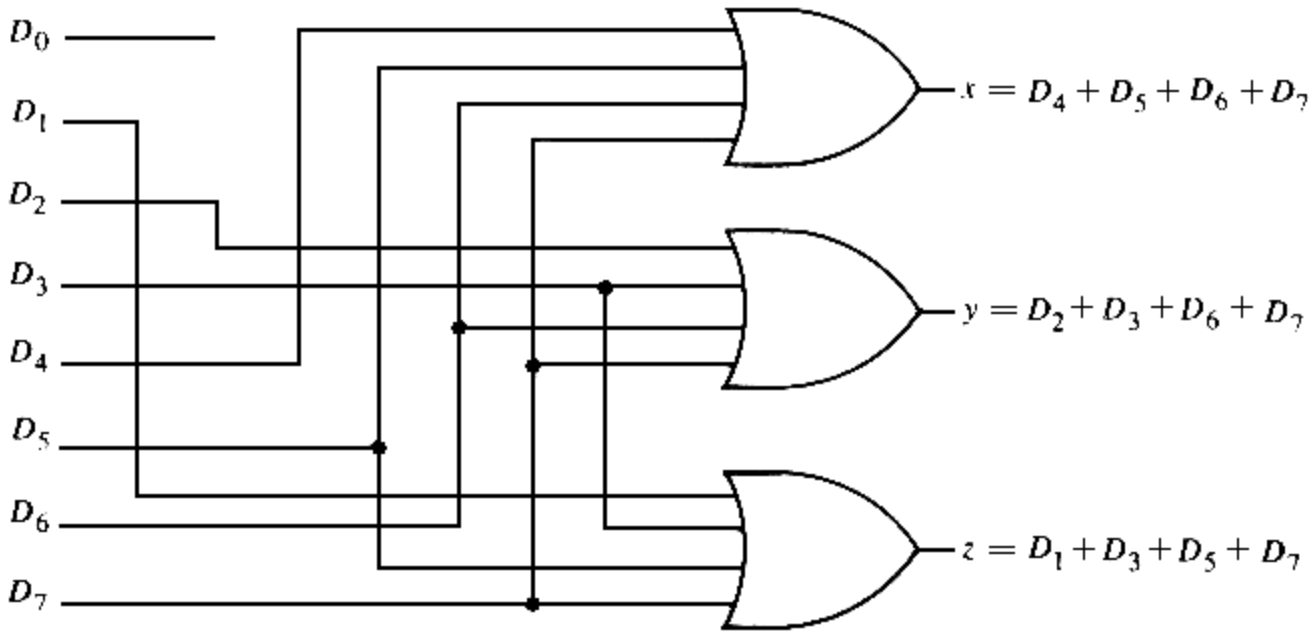
Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

$$z = D_1 + D_3 + D_5 + D_7$$
$$y = D_2 + D_3 + D_6 + D_7$$
$$x = D_4 + D_5 + D_6 + D_7$$

The encoder can be implemented with three OR gates

Encoders (2/2)

- **Three OR gates implementation**



- **Limitation of illegal input patterns**
  - **Illegal input: D_3=D_6=1**
    - **The output = 111 (D_7=1)**
  - **Illegal input: D_0=D_1=D_2=D_3=D_4=D_5=D_6=D_7=0**
    - **The output = 000 (D_0=1)**

Priority Encoder (1/3)

- Resolve the ambiguity of illegal inputs
- Only one of the input is encoded

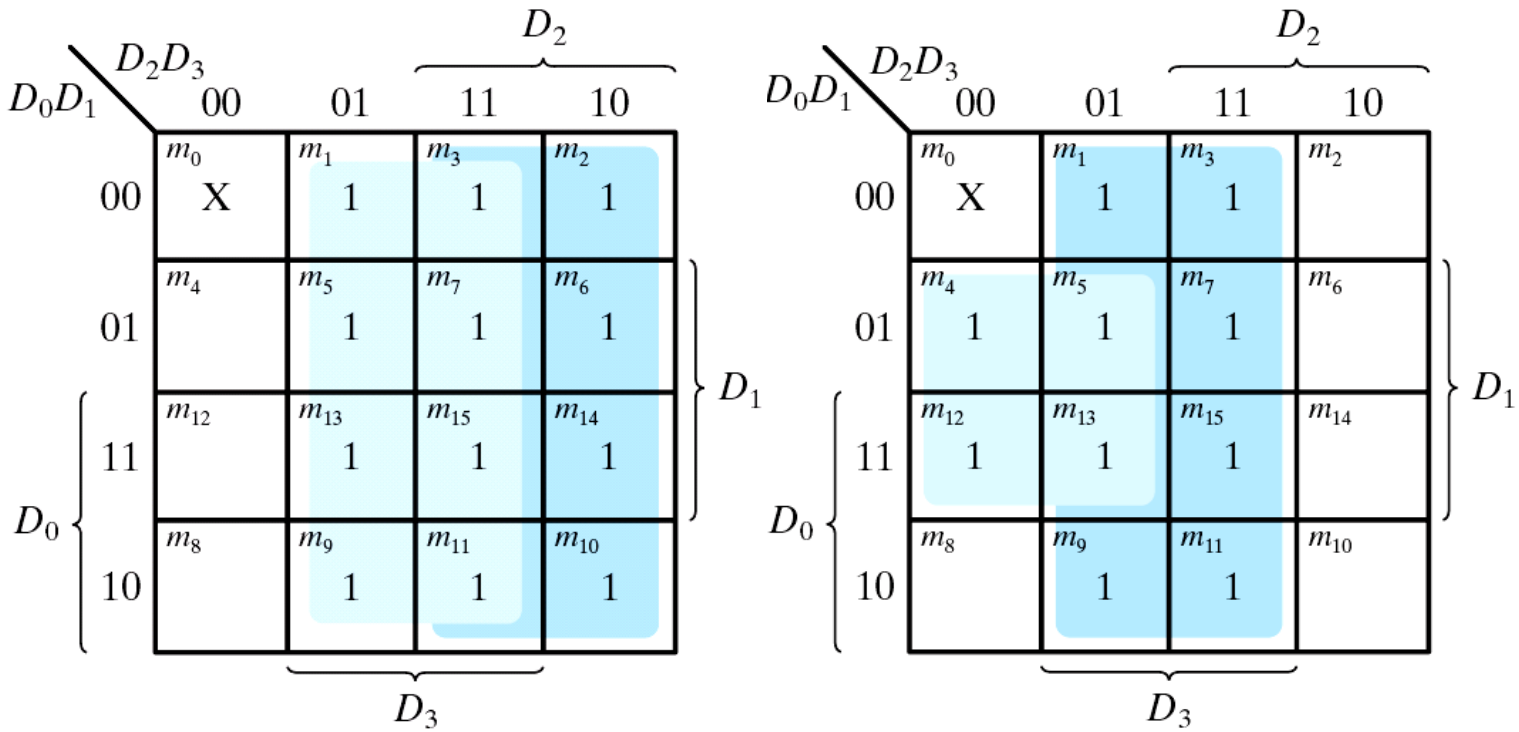
Table 4.8
Truth Table of a Priority Encoder

Inputs				Outputs		
D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

- D_3 has the highest priority
- D_0 has the lowest priority
- X: don't-care conditions
  - NOTICE: input and output don't-cares are different
- V: valid output indicator
  - Avoid $D_0=D_1=D_2=D_3=0$

Priority Encoder (2/3)

- **The maps for simplifying outputs *x* and *y***



$x = D_2 + D_3$ $\qquad$ $y = D_3 + D_1D'_2$

Fig. 22 Maps for a priority encoder

Priority Encoder (3/3)

- **Implementation of priority encoder**

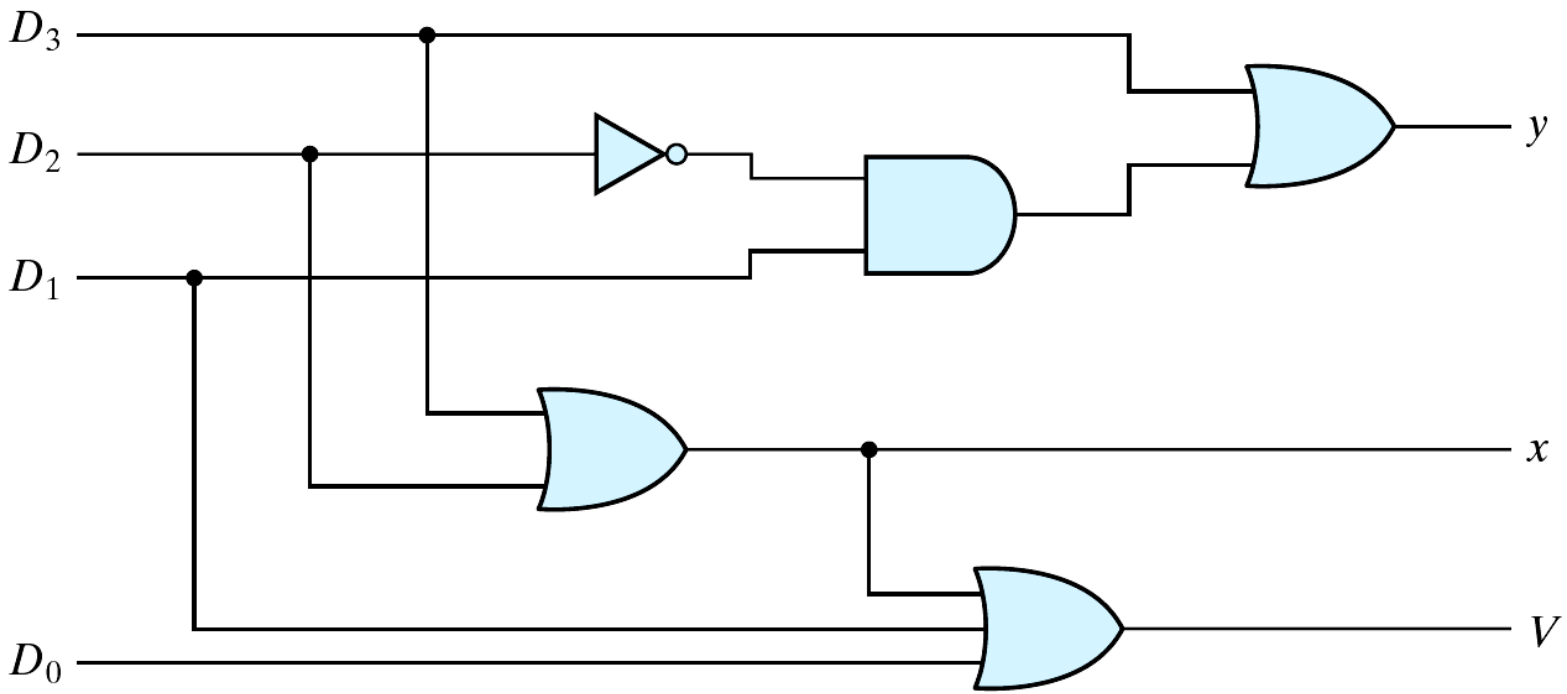


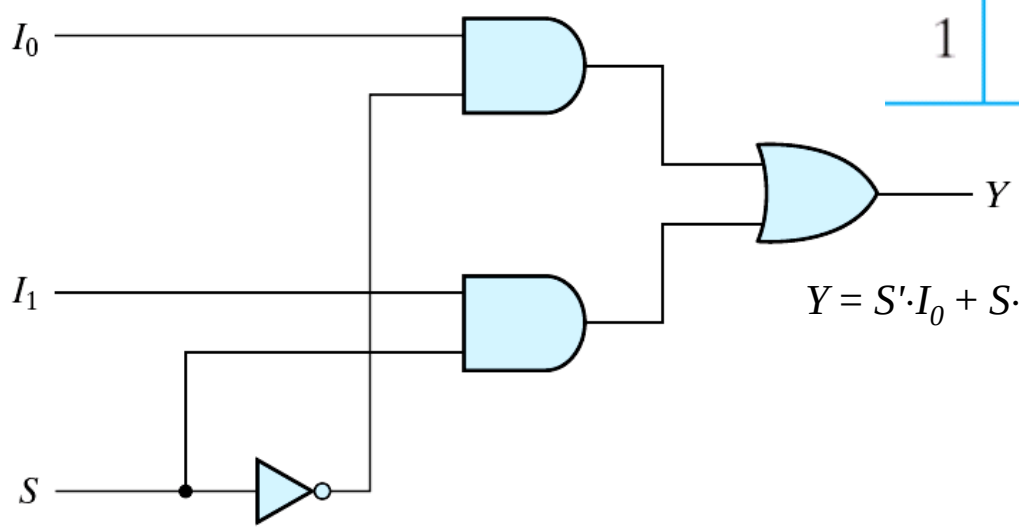
Fig. 23 Four-input priority encoder

$$x = D_2 + D_3$$
$$y = D_3 + D_1 D_2'$$
$$V = D_0 + D_1 + D_2 + D_3$$

Multiplexers

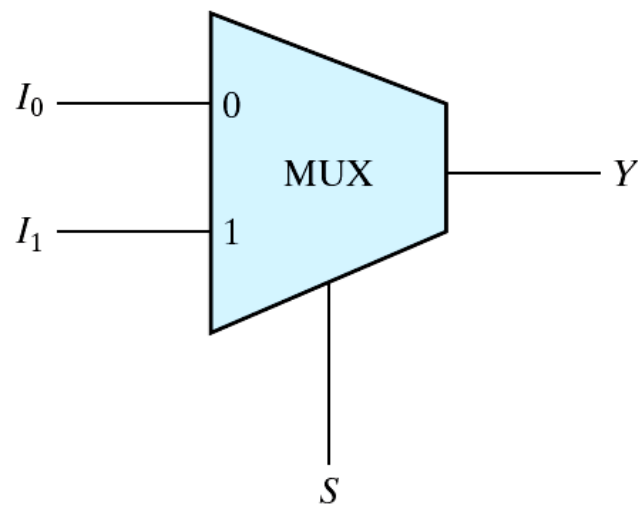
- Select binary information from one of many input lines and direct it to a single output line
- 2^n input lines, *n* selection lines, and one output line
- e.g.: 2-to-1-line multiplexer

S	Y
0	I_0
1	I_1



$Y = S' \cdot I_0 + S \cdot I_1$

(a) Logic diagram



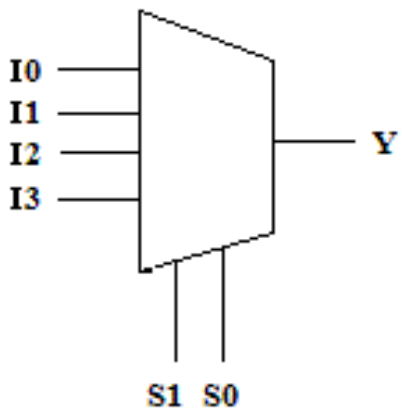
(b) Block diagram

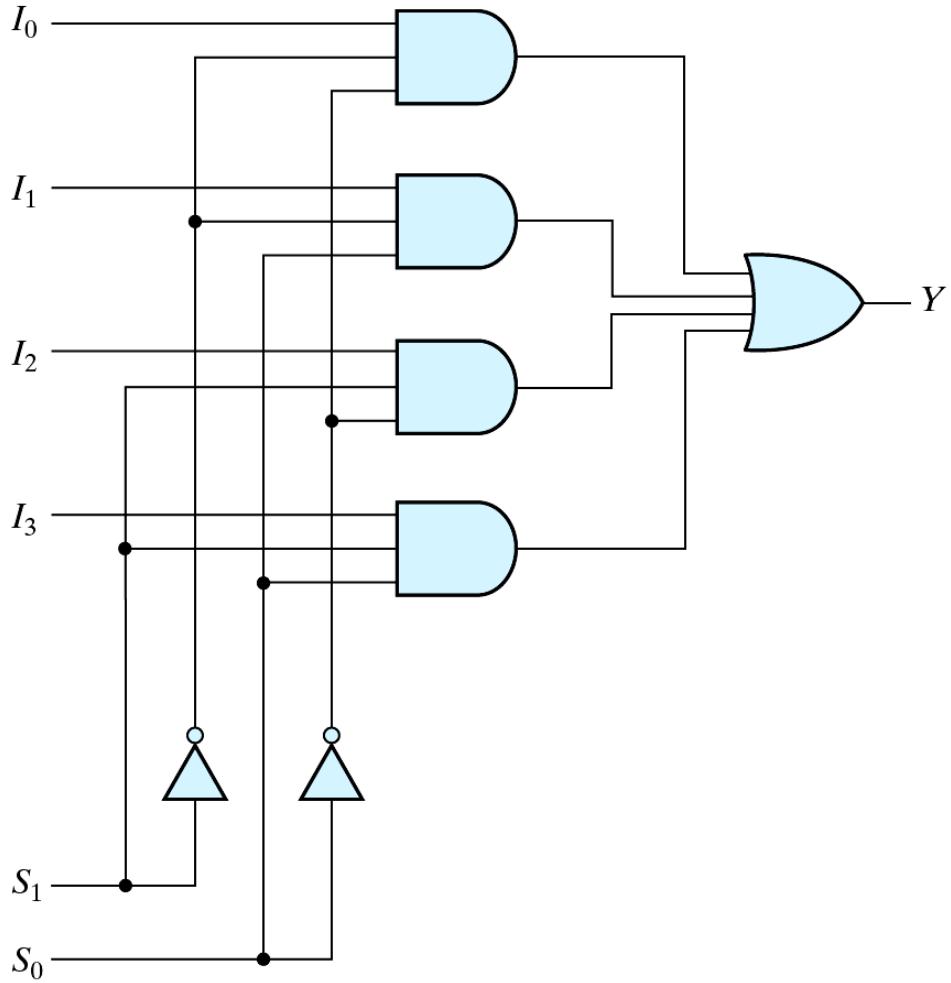
Fig. 24 Two-to-one-line multiplexer

Four-to-one-line Multiplexer

- **4-to-1 line multiplexer**

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





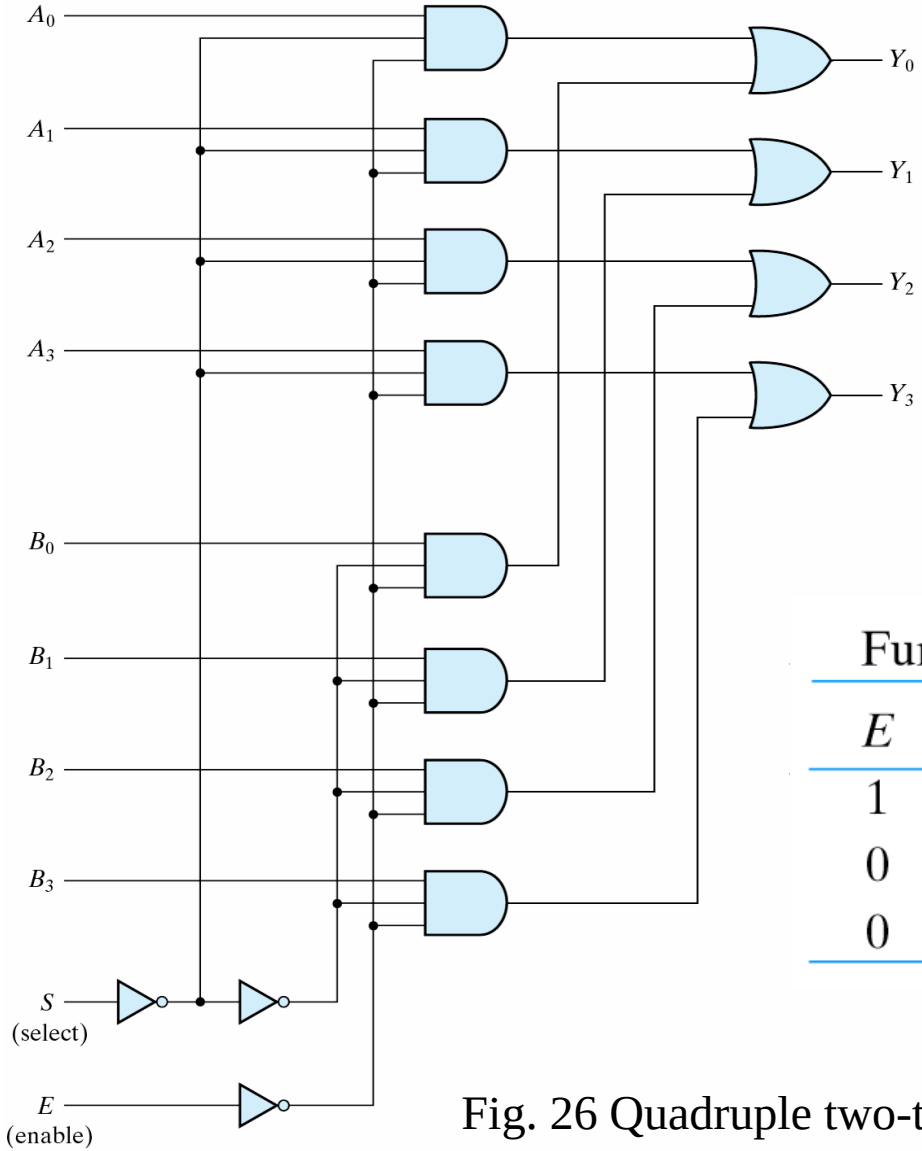
$$Y = S_1'\cdot S_0'\cdot I_0 + S_1'\cdot S_0\cdot I_1 + S_1\cdot S_0'\cdot I_2 + S_1\cdot S_0\cdot I_3$$

Fig. 25 Four-to-one-line multiplexer

2^n-to-1 Multiplexer

- **Note: 2^n-to-1 multiplexer**
  - ***n*-to-2^n decoder**
  - **Add the 2^n input lines to each AND gate**
  - **OR (all AND gates)**
  - ***n* selection lines**
  - **An enable input (optional)**

Quadruple 2-to-1-line MUX with Enable



Function table

E	S	Output Y
1	X	all 0's
0	0	select A
0	1	select B

Fig. 26 Quadruple two-to-one-line multiplexer

Boolean Function Implementation (1/2)

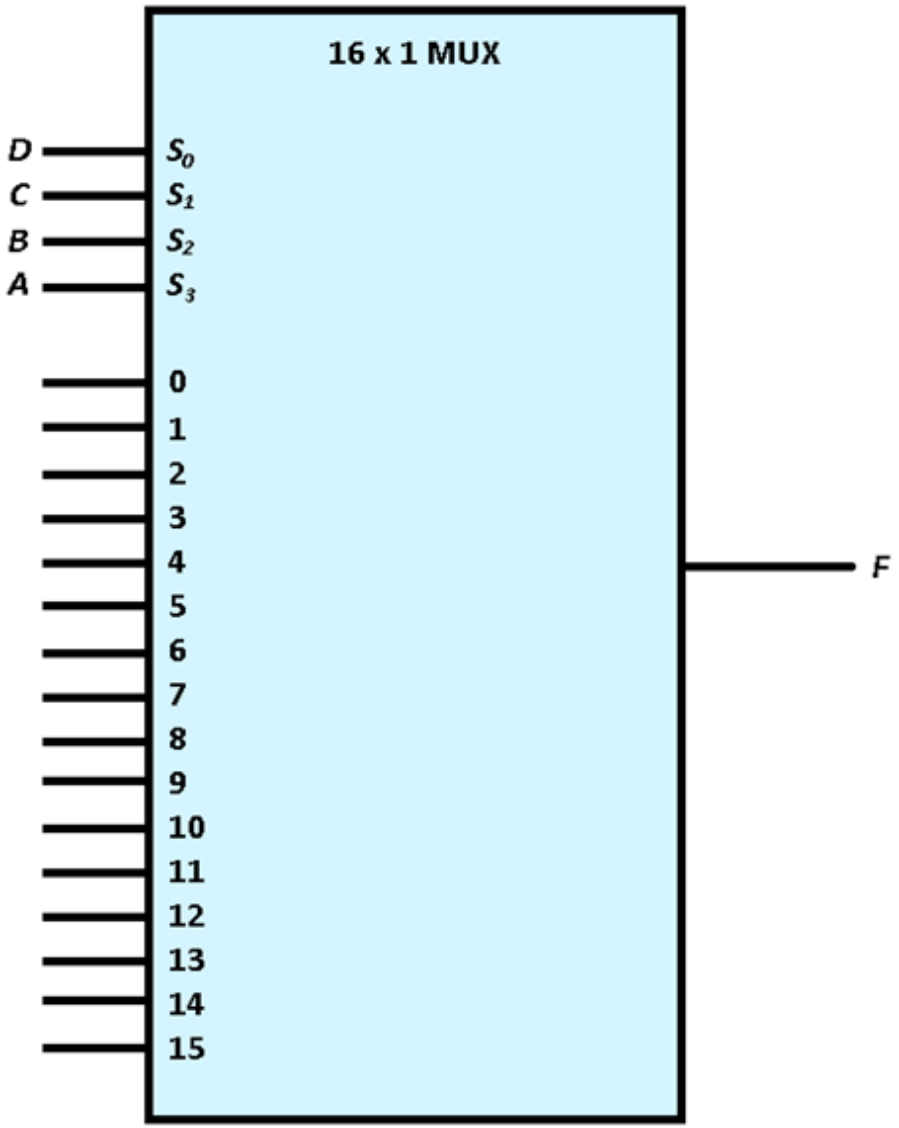
- **MUX: a decoder + an OR gate**
- ***n* inputs Boolean function implementation**
  - **2^n-to-1 MUX: trivial**
    - **Each minterm value (0 or 1) is assigned to corresponding MUX input line**
  - **2^{n-1}-to-1 MUX**
    - ***n-1* of these variables: MUX selection lines**
    - **The last variable: MUX input lines**

Boolean Function Implementation (2/2)

- **Procedure:**
  - **Assign an ordering sequence of the input variable**
  - **The rightmost variable will be used for the input lines**
  - **Assign the remaining n-1 variables to the selection lines w.r.t. their corresponding sequence**
  - **Construct the truth table**
  - **Consider a pair of consecutive minterms starting from m_0**
  - **Determine the input lines**

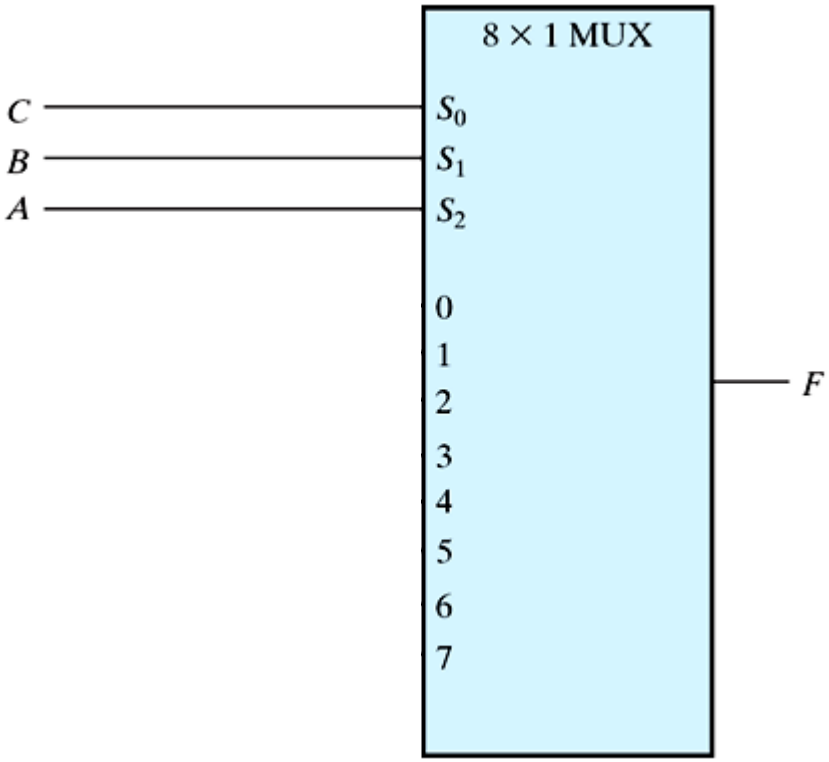
16-to-1 MUX Implementation (1/4)

A	B	C	D	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
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	0
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



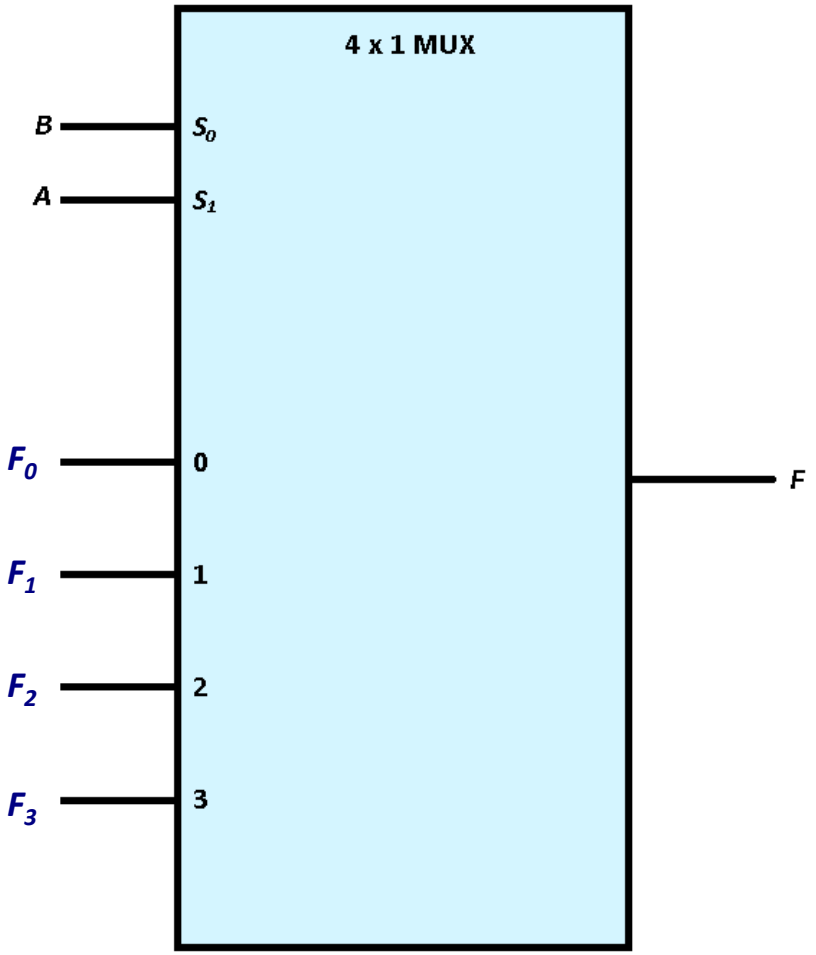
8-to-1 MUX Implementation (2/4)

A	B	C	D	F	
0	0	0	0	0	$F_0 = D$
0	0	0	1	1	
0	0	1	0	0	$F_1 = D$
0	0	1	1	1	
0	1	0	0	1	$F_2 = D'$
0	1	0	1	0	
0	1	1	0	0	$F_3 = 0$
0	1	1	1	0	
1	0	0	0	0	$F_4 = 0$
1	0	0	1	0	
1	0	1	0	0	$F_5 = D$
1	0	1	1	1	
1	1	0	0	1	$F_6 = 1$
1	1	0	1	1	
1	1	1	0	1	$F_7 = 1$
1	1	1	1	1	



4-to-1 MUX Implementation (3/4)

A	B	C	D	F	
0	0	0	0	0	
0	0	0	1	1	$F_0 = D$
0	0	1	0	0	
0	0	1	1	1	
0	1	0	0	1	
0	1	0	1	0	$F_1 = C'D'$
0	1	1	0	0	
0	1	1	1	0	
1	0	0	0	0	
1	0	0	1	0	$F_2 = CD$
1	0	1	0	0	
1	0	1	1	1	
1	1	0	0	1	
1	1	0	1	1	$F_3 = 1$
1	1	1	0	1	
1	1	1	1	1	

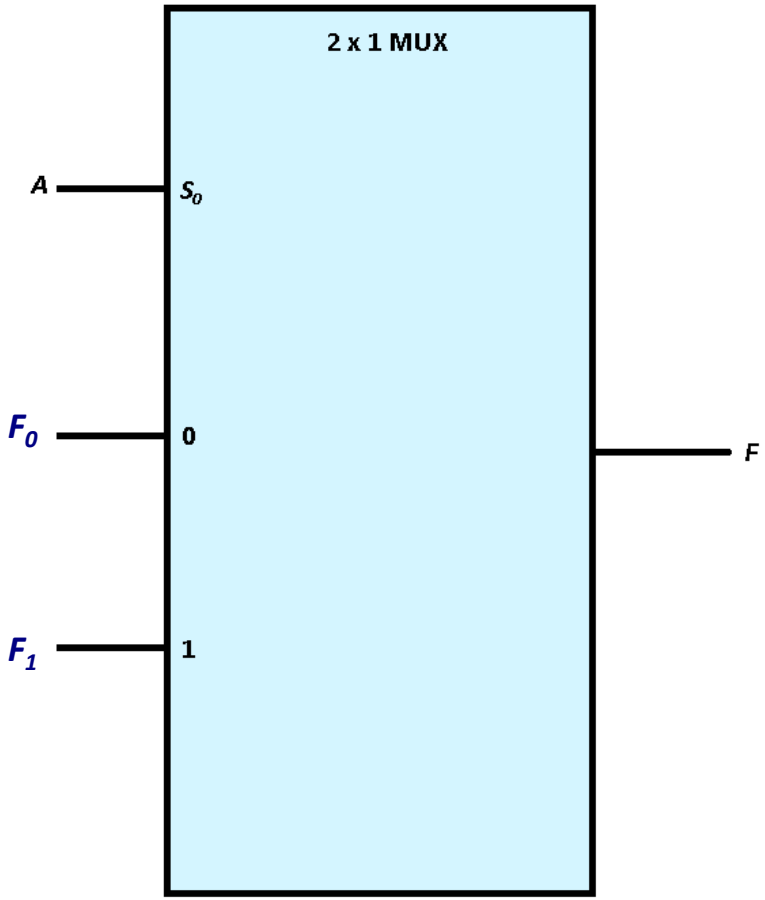


2-to-1 MUX Implementation (4/4)

A	B	C	D	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
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	0
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

$F_0 = B'D + BC'D'$

$F_1 = B'CD + B$



Three-state (Tri-state) Gates

- **A multiplexer can be constructed with three-state gates**
- **Output state: 0, 1, and high-impedance (open circuits)**

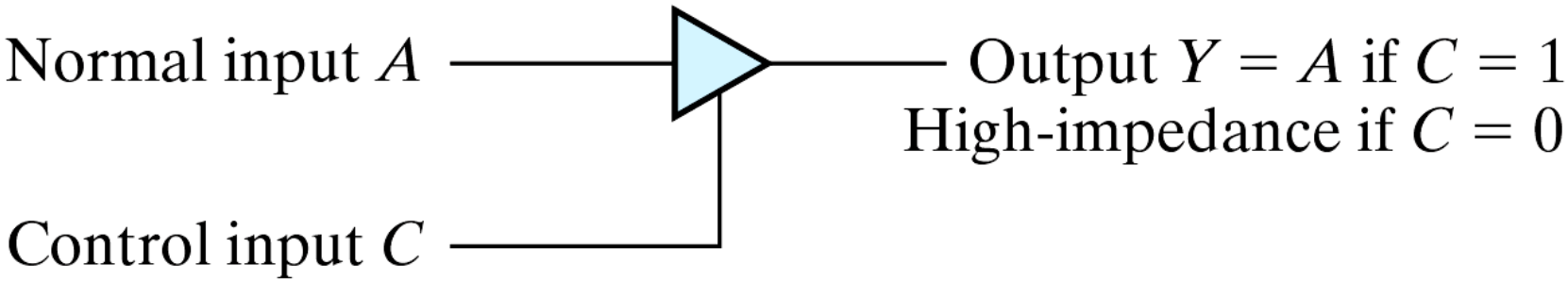
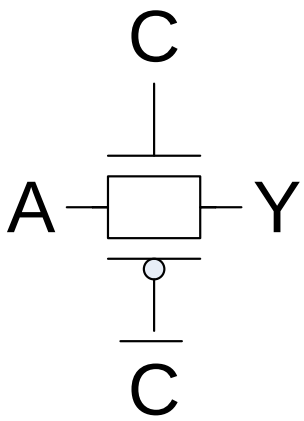


Fig. 29 Graphic symbol for a three-state buffer



C	A	Y
0	0	Z
0	1	Z
1	0	0
1	1	1

Multiplexer Examples

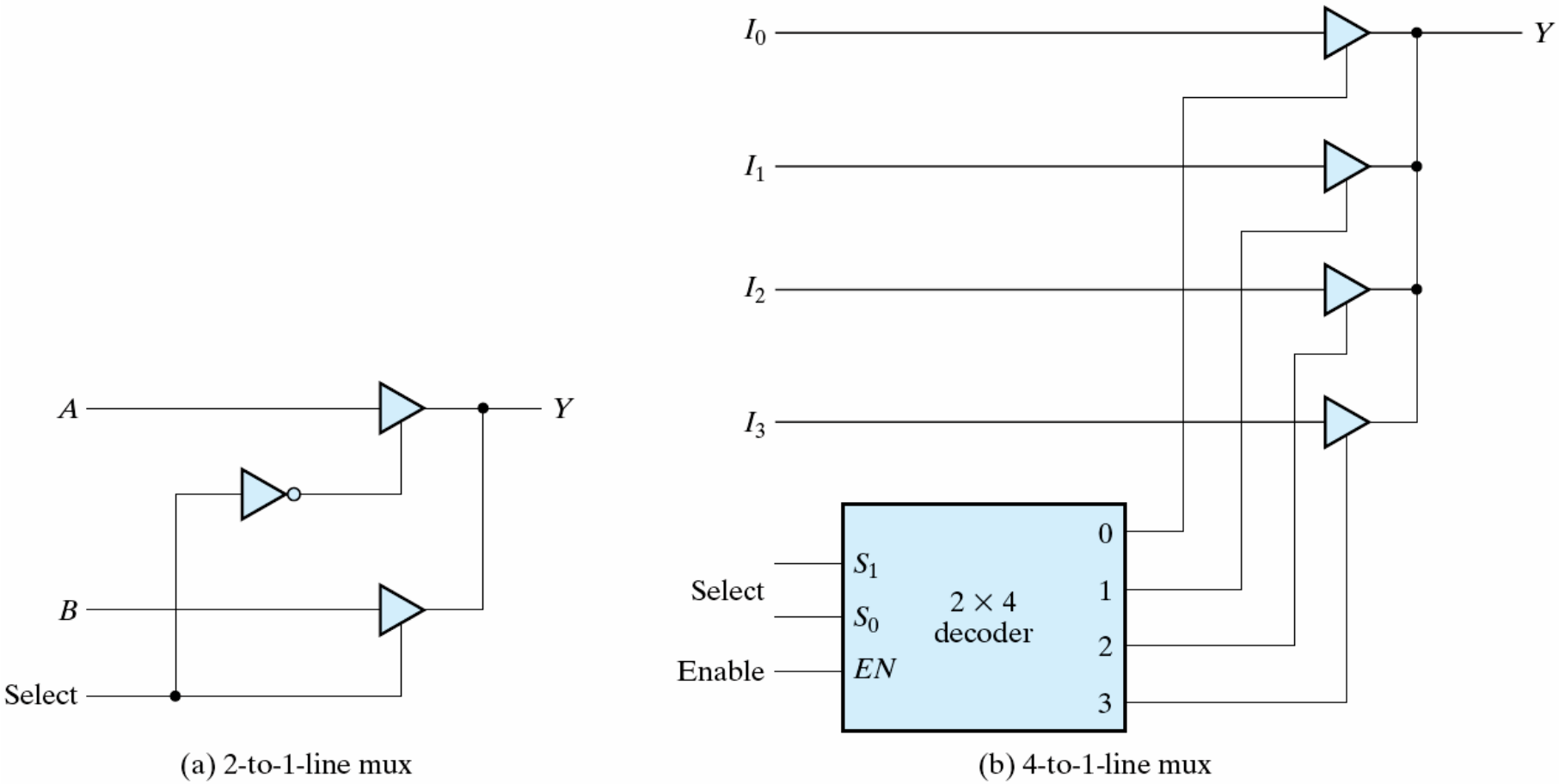
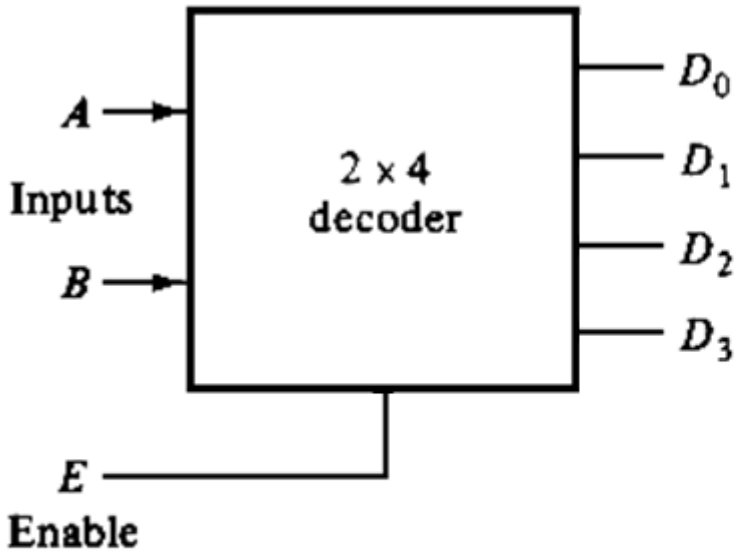


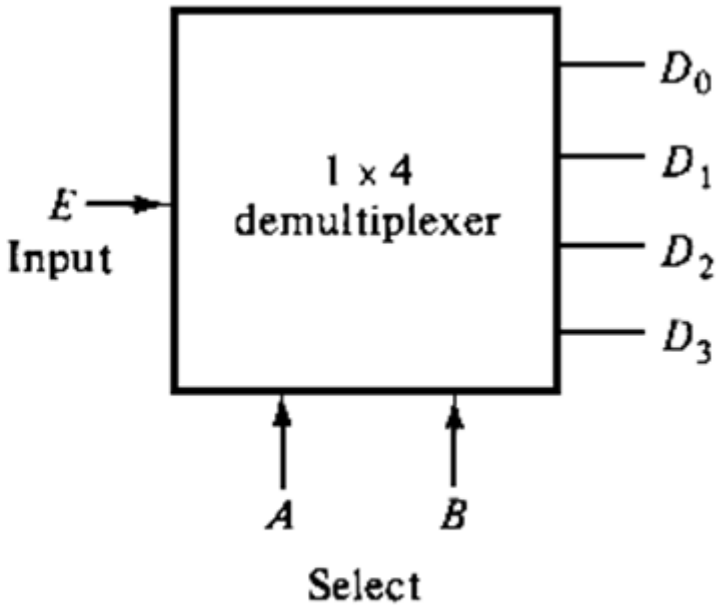
Fig. 30 Multiplexer with three-state gates

Decoder/Demultiplexer

- **Decoder/demultiplexer**



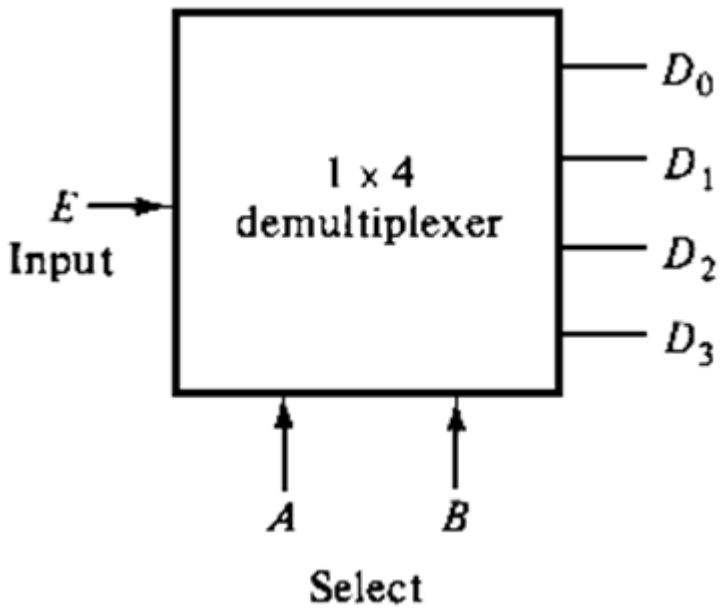
(a) Decoder with enable

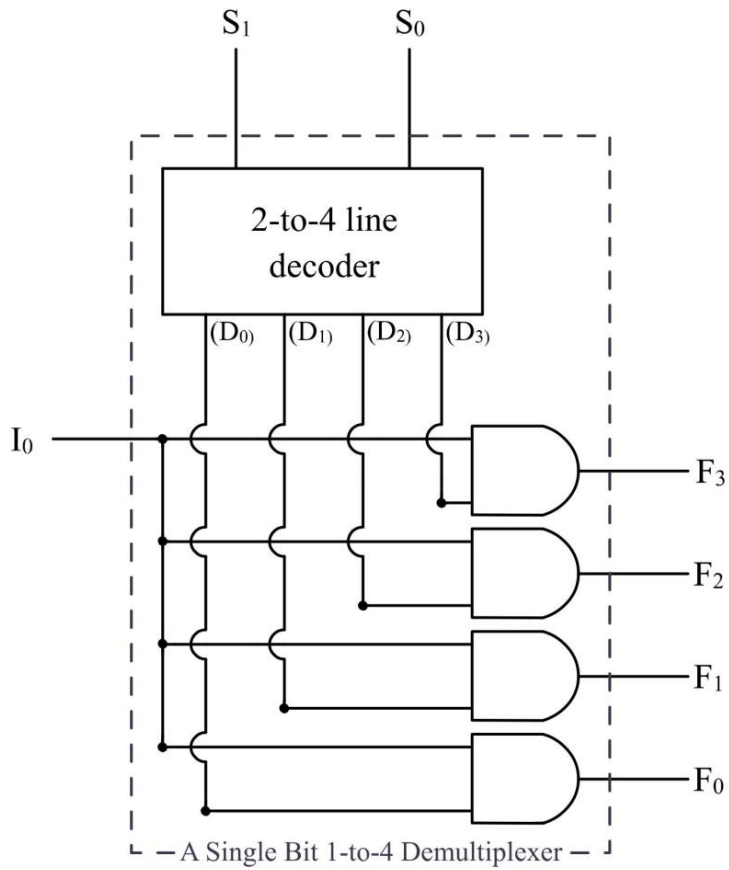


(b) Demultiplexer

Demultiplexer

- Forward the data input to one of the outputs
- 1 input lines, *n* selection lines, and 2^n input lines





Truth Table

S_1	S_0	I_0	F_3	F_2	F_1	F_0
0	0	0	0	0	0	0
		1	0	0	0	1
0	1	0	0	0	0	0
		1	0	0	1	0
1	0	0	0	0	0	0
		1	0	1	0	0
1	1	0	0	0	0	0
		1	1	0	0	0