

DIGITAL LOGIC DESIGN

Course Code: 328356(28)



Mr. Manjeet Singh Sonwani
Assistant Professor
Department of Electronics & Telecomm.
Government Engineering College Raipur



UNIT-III COMBINATIONAL CIRCUITS

- Introduction : Adder & Subtractor: Half adder, Full adder, Half subtractor, Full subtractor;
- Binary Parallel Adder; The Look Ahead Carry Adder; Serial Adder; BCD Adder;
- Code Converters; Parity Bit Generators/ Checkers; Comparators;
- Decoders: 3-Line to 8-Line Decoder, 8-4-2-1 BCD to Decimal Decoder, BCD to Seven Segment Decoder; Encoders: Octal to Binary and Decimal to BCD Encoder; Multiplexers: 2-Input Multiplexer, 4-Input Multiplexer, 16-Input Multiplexer;
- Demultiplexers: 1-Line to 4-Line & 1-Line to 8- Line Demux
- Applications of Multiplexers.

Combinational Logic

- Logic circuits for digital systems may be combinational or sequential.
- A combinational circuit consists of input variables, logic gates, and output variables.

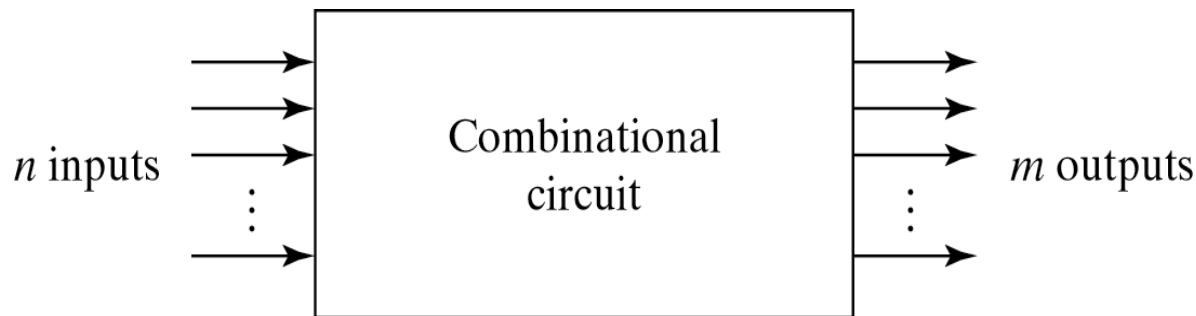


Fig. 4-1 Block Diagram of Combinational Circuit



4-2. Analysis procedure

- To obtain the output Boolean functions from a logic diagram, proceed as follows:
 1. Label all gate outputs that are a function of input variables with arbitrary symbols. Determine the Boolean functions for each gate output.
 2. Label the gates that are a function of input variables and previously labeled gates with other arbitrary symbols. Find the Boolean functions for these gates.



Analysis procedure

3. Repeat the process outlined in step 2 until the outputs of the circuit are obtained.
4. By repeated substitution of previously defined functions, obtain the output Boolean functions in terms of input variables.

Example

$$F_2 = AB + AC + BC; \quad T_1 = A + B + C; \quad T_2 = ABC; \quad T_3 = F_2' T_1;$$

$$F_1 = T_3 + T_2$$

$$F_1 = T_3 + T_2 = F_2' T_1 + ABC = A'BC' + A'B'C + AB'C' + ABC$$

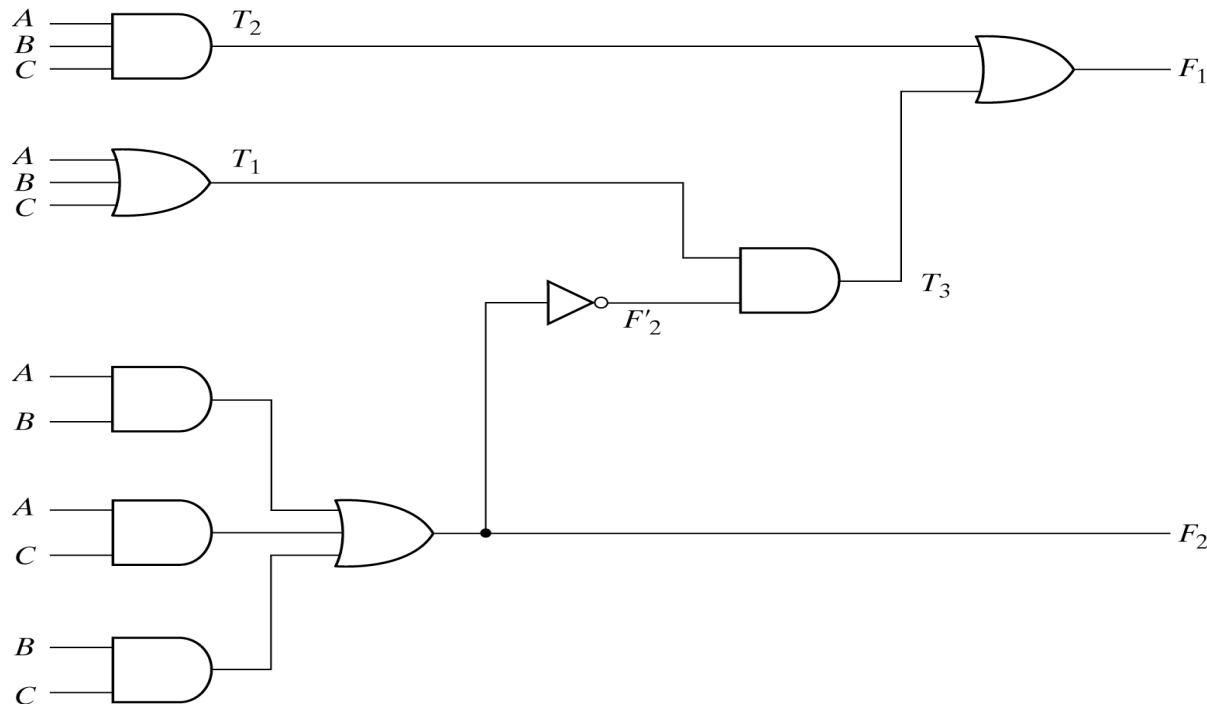


Fig. 4-2 Logic Diagram for Analysis Example

Derive truth table from logic diagram

- We can derive the truth table in Table 4-1 by using the circuit of Fig.4-2.

Table 4-1
Truth Table for the Logic Diagram of Fig. 4-2

<i>A</i>	<i>B</i>	<i>C</i>	<i>F</i> ₂	<i>F</i> ₂	<i>T</i> ₁	<i>T</i> ₂	<i>T</i> ₃	<i>F</i> 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

1. Table 4-2 is a Code-Conversion example, first, we can list the relation of the BCD and Excess-3 codes in 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

Karnaugh map

2. For each symbol of the Excess-3 code, we use 1's to draw the map for simplifying Boolean function.

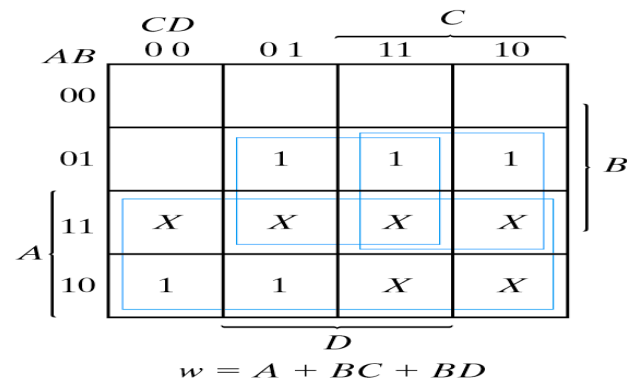
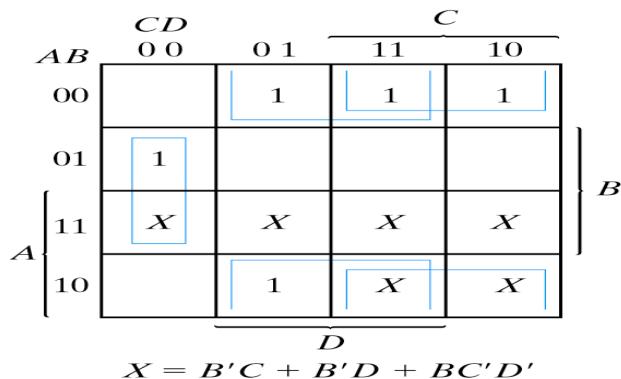
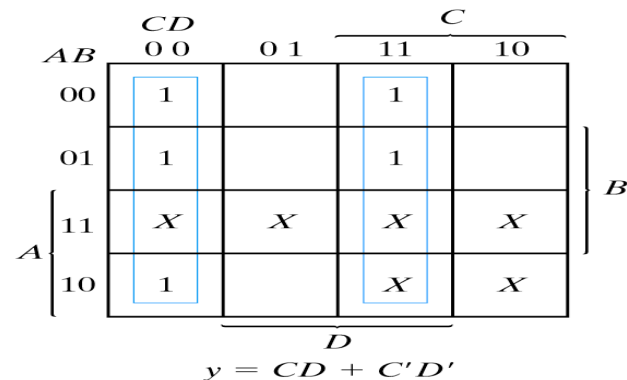
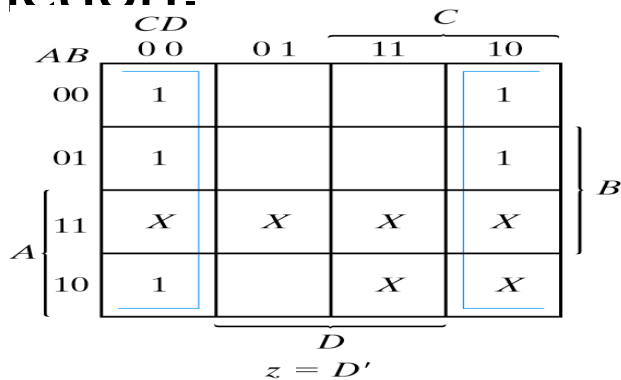


Fig. 4-3 Maps for BCD to Excess-3 Code Converter

Circuit implementation

$$\begin{aligned}z &= D'; & y &= CD + C'D' = CD + (C + D)' \\x &= B'C + B'D + BC'D' = B'(C + D) + B(C + D)' \\w &= A + BC + BD = A + B(C + D)\end{aligned}$$

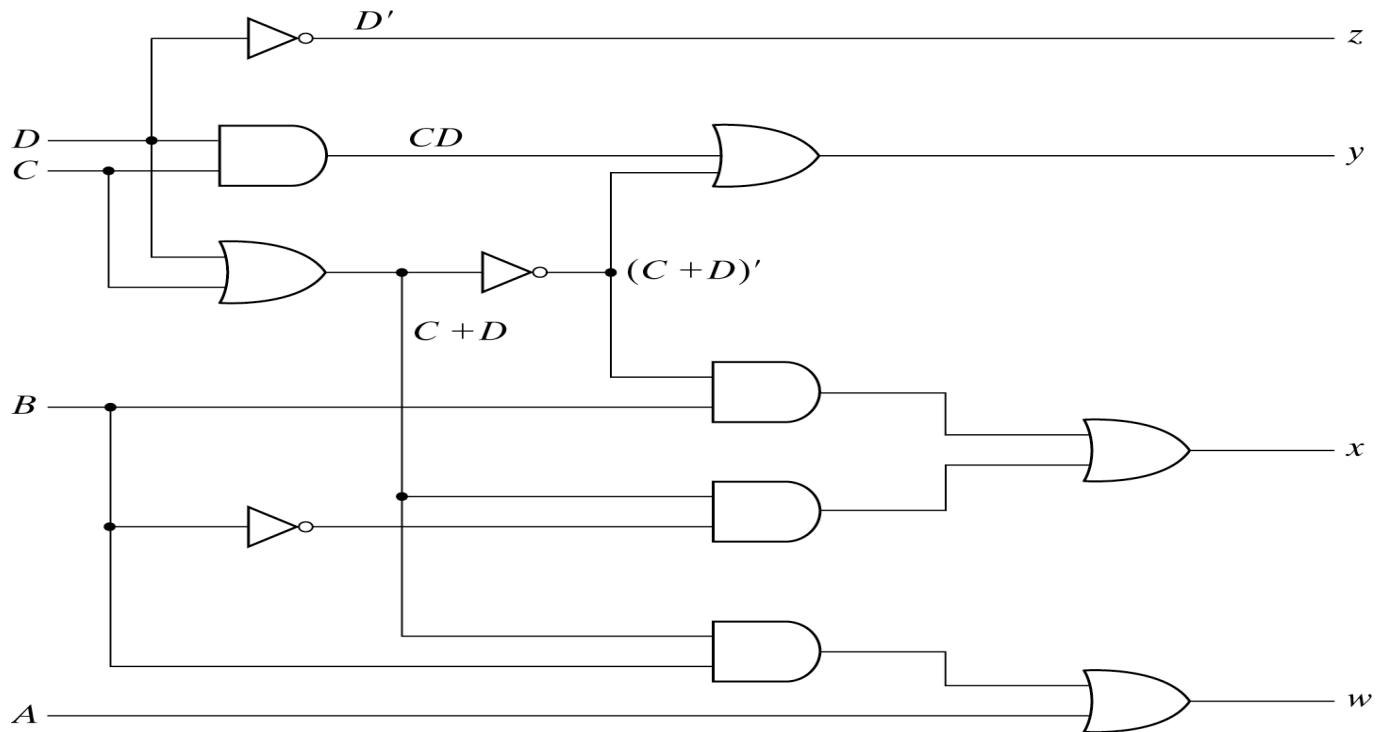


Fig. 4-4 Logic Diagram for BCD to Excess-3 Code Converter

Binary Adder-Subtractor

- A combinational circuit that performs the addition of two bits is called a **half adder**.
- The truth table for the half adder is listed below:

Table 4-3
Half Adder

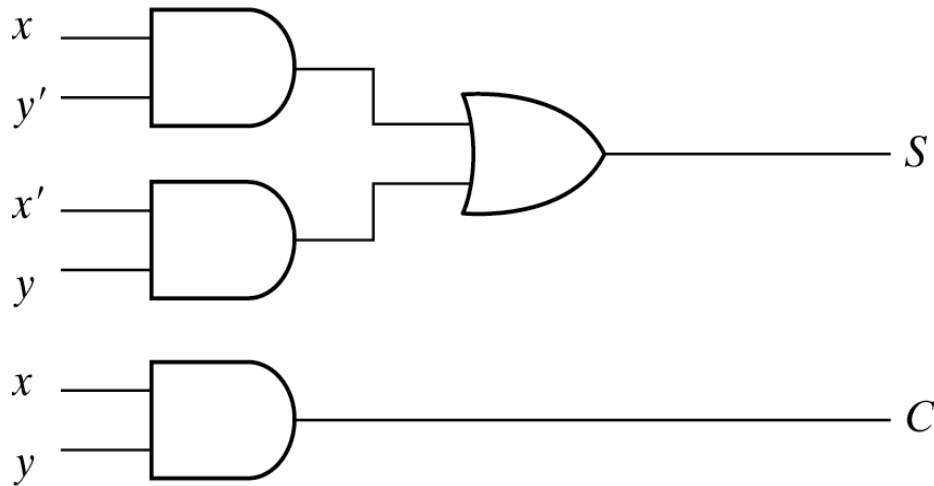
<i>x</i>	<i>y</i>	<i>C</i>	<i>S</i>
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

S: Sum
C: Carry

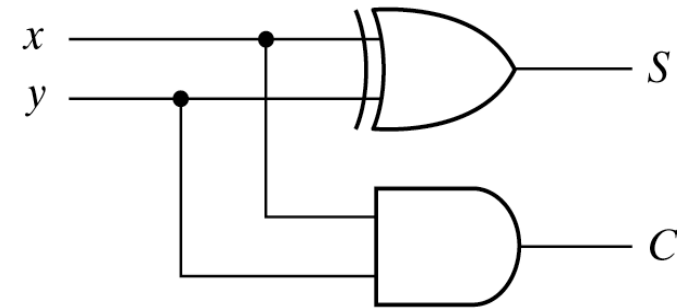
$$S = x'y + xy'$$

$$C = xy$$

Implementation of Half-Adder



$$(a) \begin{aligned} S &= xy' + x'y \\ C &= xy \end{aligned}$$



$$(b) \begin{aligned} S &= x \oplus y \\ C &= xy \end{aligned}$$

Fig. 4-5 Implementation of Half-Adder

Full-Adder

- One that performs the addition of three bits (two significant bits and a previous carry) is a **full adder**.

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

Simplified Expressions

		yz		y	
		00	01	11	10
x	0		1		1
x	1	1		1	
		z			

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

		yz		y	
		00	01	11	10
x	0			1	
x	1		1	1	1
		z			

$$\begin{aligned} C &= xy + xz + yz \\ &= xy + xy'z + x'yz \end{aligned}$$

Fig. 4-6 Maps for Full Adder

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

$$C = xy + xz + yz$$

Full adder implemented in SOP

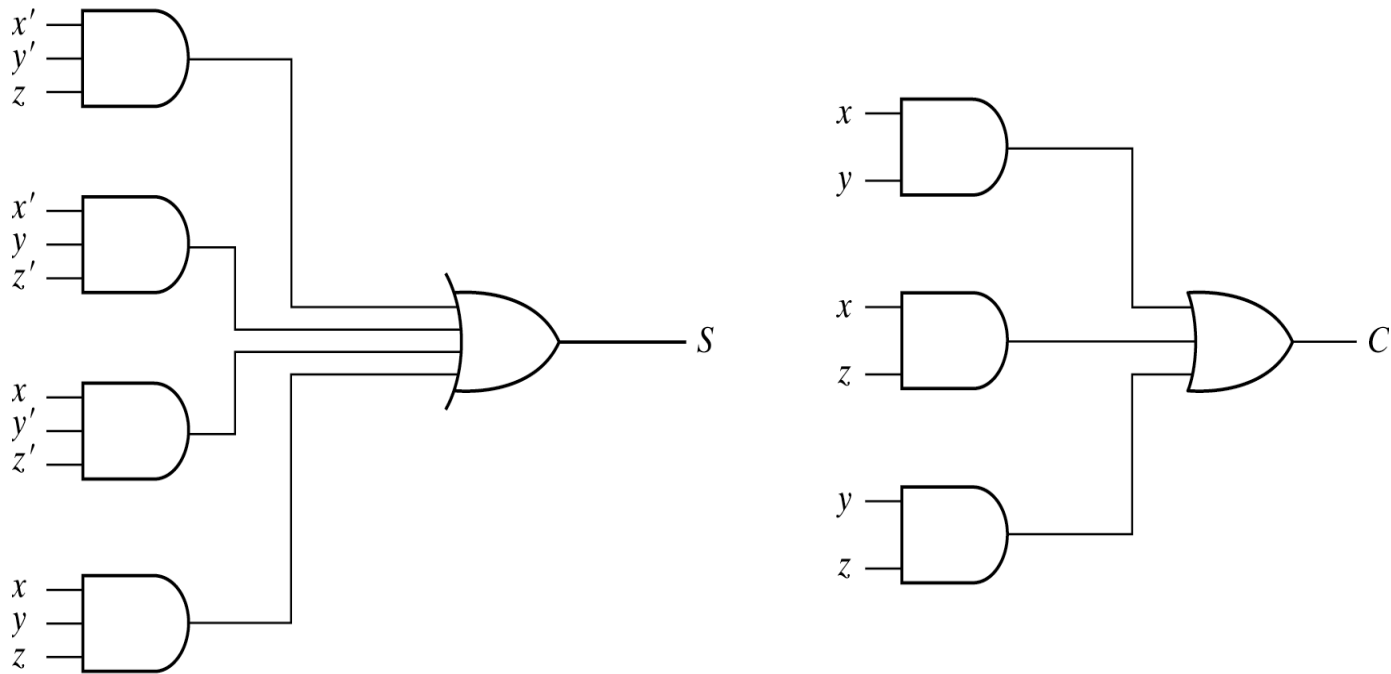


Fig. 4-7 Implementation of Full Adder in Sum of Products

Another implementation

- Full-adder can also implemented with **two half adders and one OR gate** (Carry Look-Ahead adder).

$$S = z \oplus (x \oplus y)$$

$$= z'(xy' + x'y) + z(xy' + x'y)$$

$$= xy'z' + x'yz' + xyz + x'y'z$$

$$C = z(xy' + x'y) + xy = xy'z + x'yz + xy$$

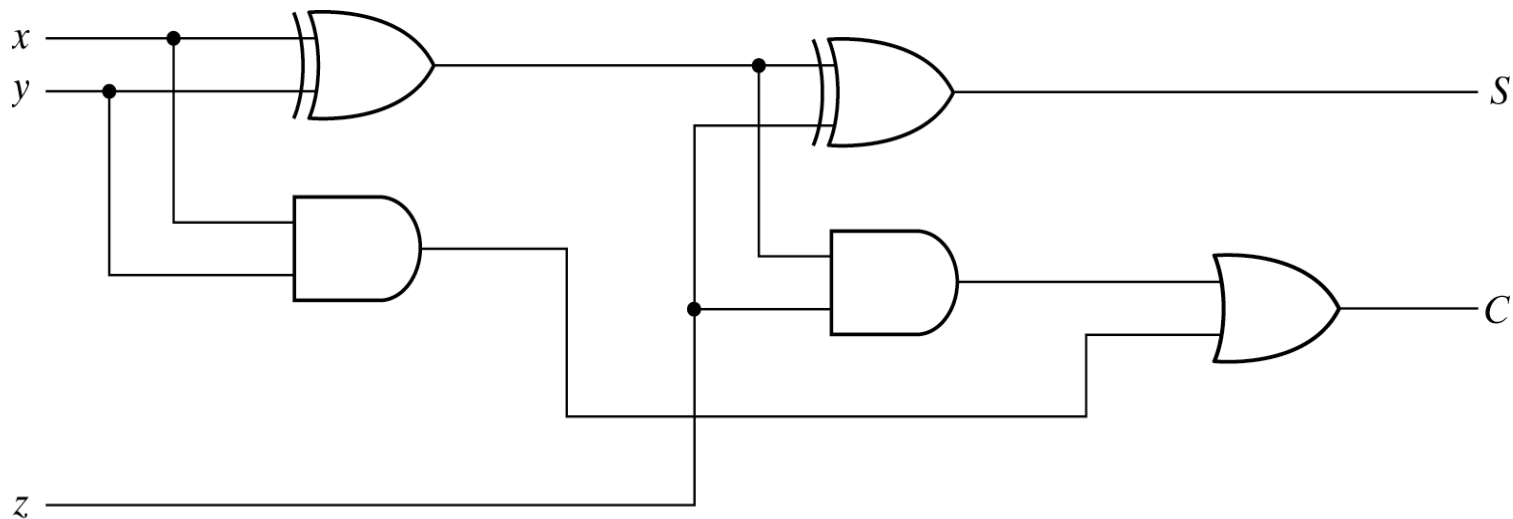


Fig. 4-8 Implementation of Full Adder with Two Half Adders and an OR Gate

Binary adder

- This is also called **Ripple Carry Adder**, because of the construction with full adders are connected in cascade.

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

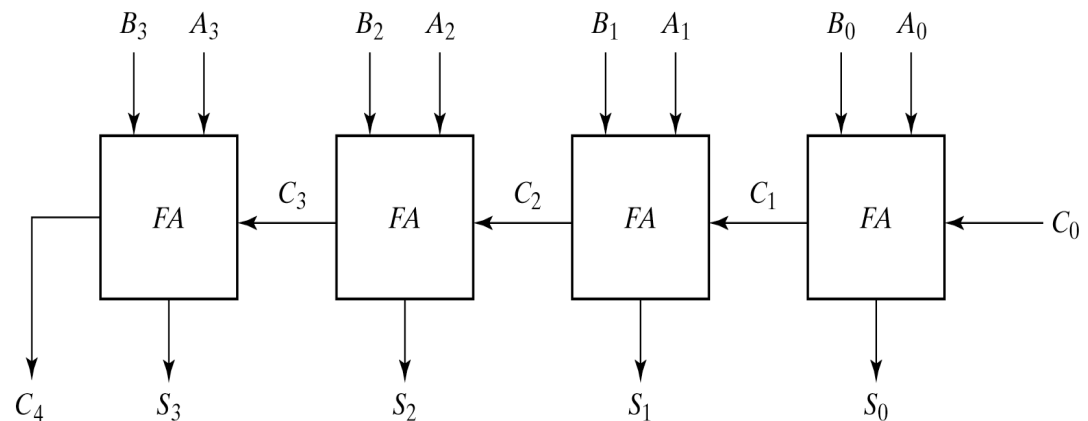


Fig. 4-9 4-Bit Adder



Carry Propagation

- Fig.4-9 causes a **unstable** factor on **carry bit**, and produces a **longest propagation delay**.
- The signal from C_i to the output carry C_{i+1} , **propagates through an AND and OR gates**, so, for an n-bit RCA, there are **$2n$** gate levels for the carry to propagate from input to output.

Carry Propagation

- Because the propagation delay will affect the output signals on different time, so the signals are **given enough time to get the precise and stable outputs**.
- The most widely used technique employs the principle of **carry look-ahead** to **improve the speed of the algorithm**.

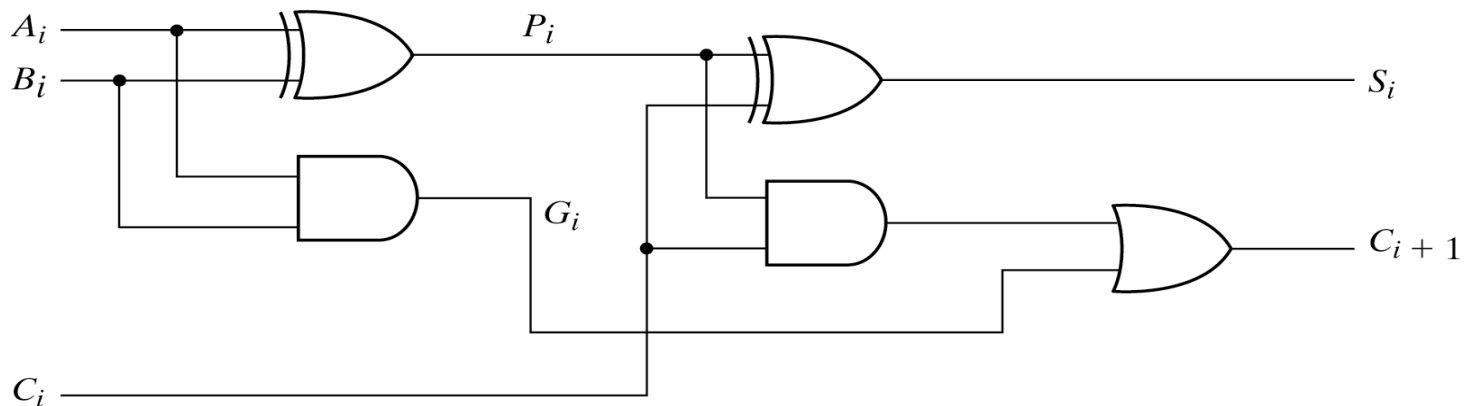


Fig. 4-10 Full Adder with P and G Shown



Boolean functions

$$P_i = A_i \oplus B_i \quad \text{steady state value}$$

$$G_i = A_i B_i \quad \text{steady state value}$$

Output sum and carry

$$S_i = P_i \oplus C_i$$

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

G_i : carry generate P_i : carry propagate

C_0 = input carry

$$C_1 = G_0 + P_0 C_0$$

$$C_2 = G_1 + P_1 C_1 = G_1 + P_1 G_0 + P_1 P_0 C_0$$

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

- C_3 does not have to wait for C_2 and C_1 to propagate.

Logic diagram of carry look-ahead generator

- C_3 is propagated at the same time as C_2 and C_1 .

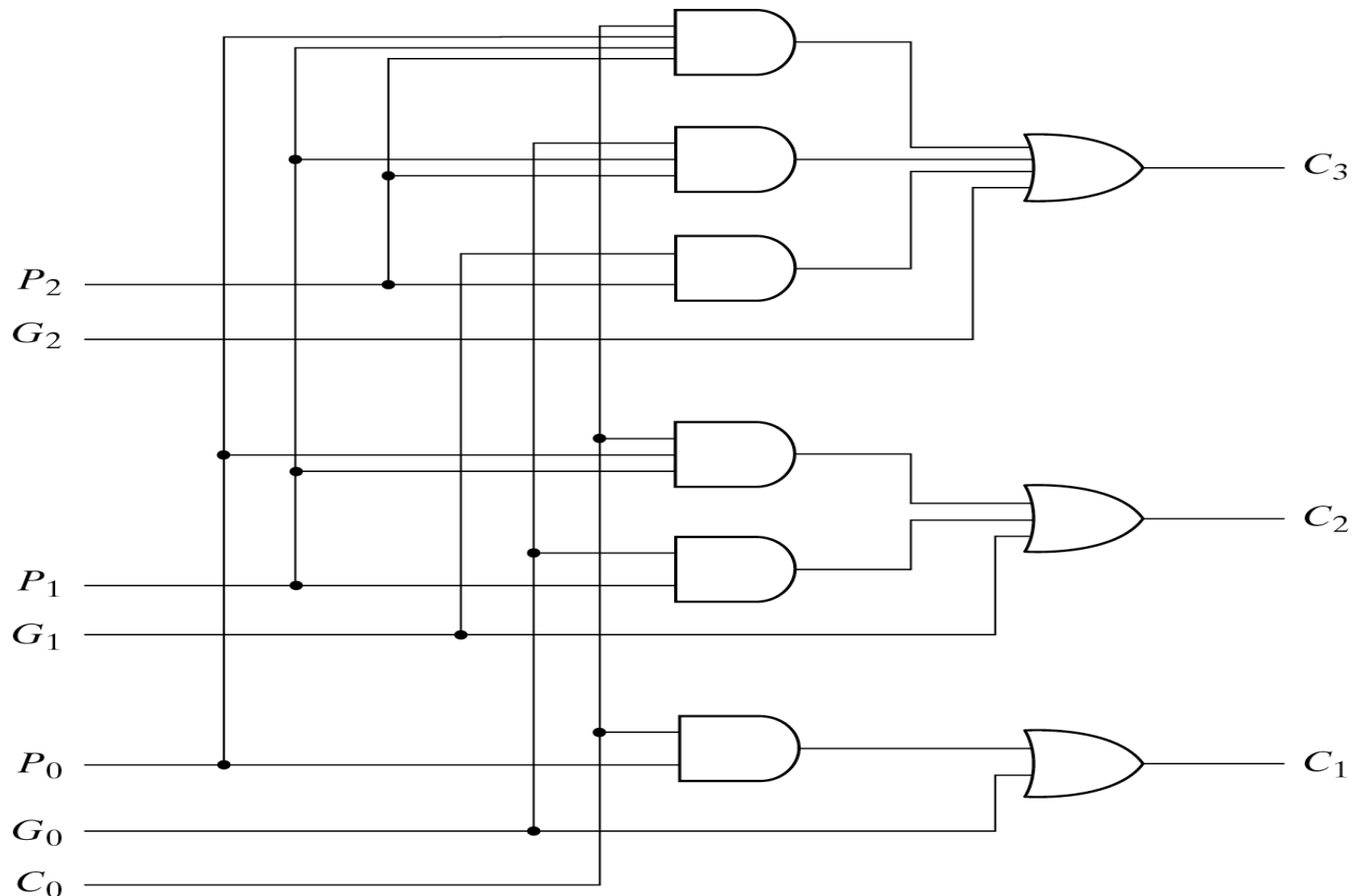


Fig. 4-11 Logic Diagram of Carry Lookahead Generator

4-bit adder with carry lookahead

- Delay time of n-bit CLAA = XOR + (AND + OR) + XOR

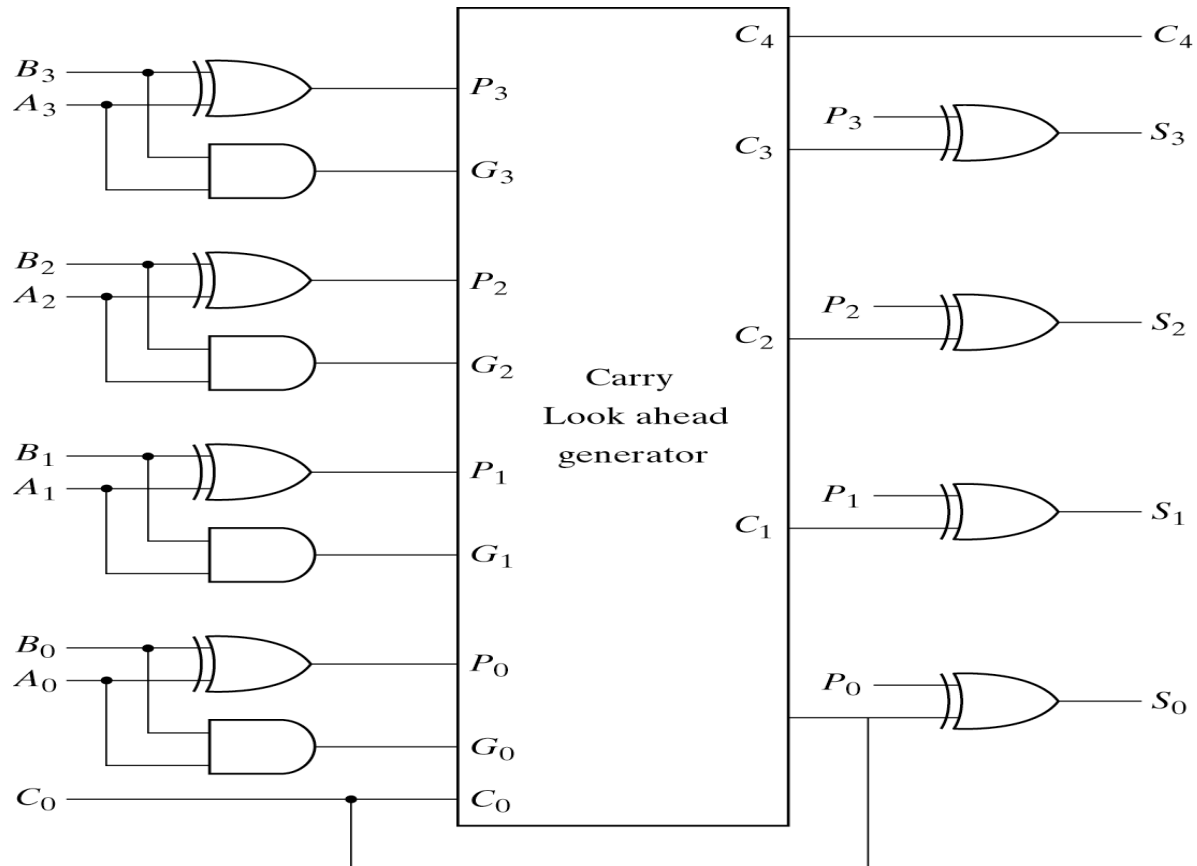


Fig. 4-12 4-Bit Adder with Carry Lookahead

Binary subtractor

$M = 1 \rightarrow \text{subtractor}$; $M = 0 \rightarrow \text{adder}$

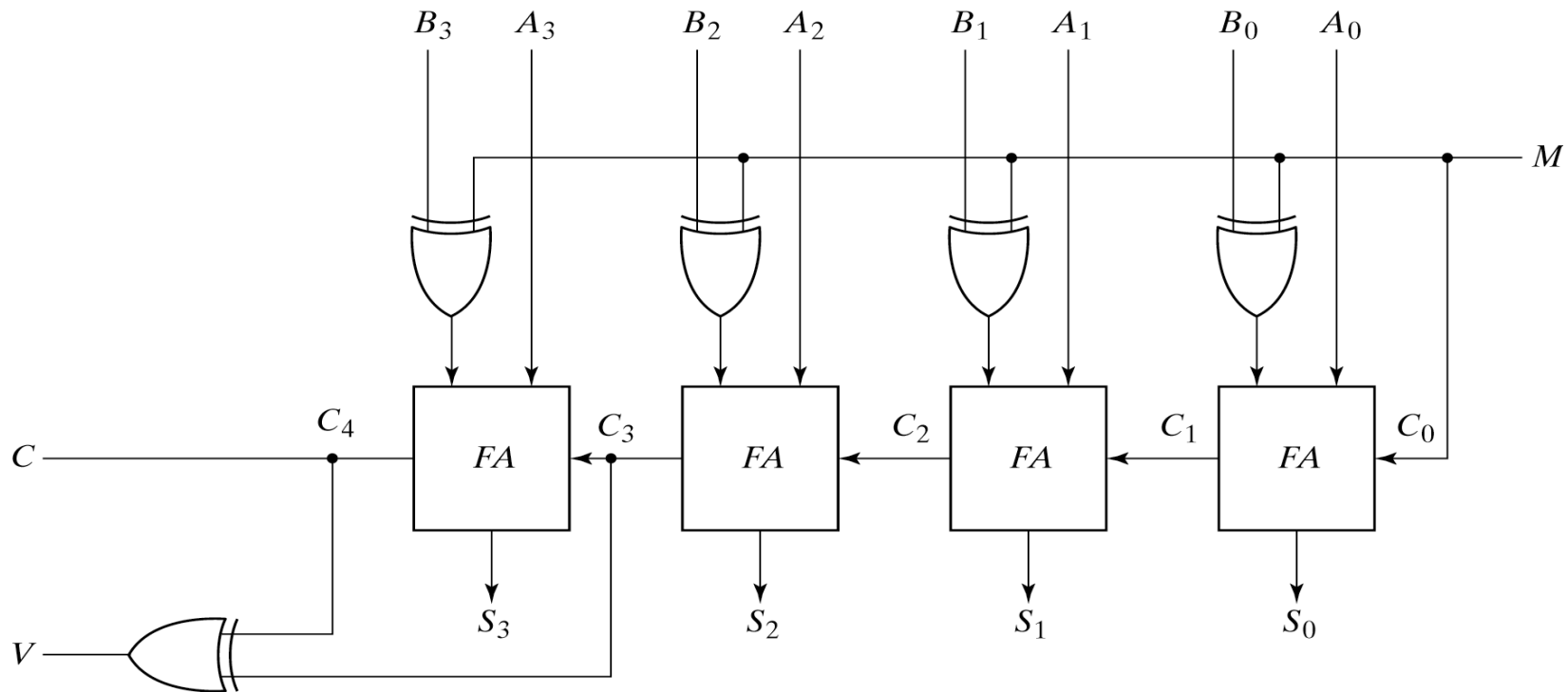


Fig. 4-13 4-Bit Adder Subtractor



Overflow

- It is **worth** noting Fig.4-13 that binary numbers in the **signed-complement system are added and subtracted** by the same basic addition and subtraction rules **as unsigned numbers**.
- Overflow is a problem in digital computers because the number of bits that hold the number is finite and a result that contains $n+1$ bits cannot be accommodated.



Overflow on signed and unsigned

- When two **unsigned** numbers are added, an overflow is detected from the **end carry out of the MSB position**.
- When two **signed** numbers are added, the sign bit is treated as part of the number and the end carry does not indicate an overflow.
- An **overflow can't occur** after an addition if one number is **positive** and the other is **negative**.
- An overflow may occur if the two numbers added are both positive or both negative.

4-5 Decimal adder

BCD adder can't exceed 9 on each input digit. K is the carry.

Table 4-5
Derivation of BCD Adder

K	Binary Sum				BCD Sum					Decimal
	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



Rules of BCD adder

- When the binary sum is greater than 1001, we obtain a non-valid BCD representation.
- The addition of binary 6(0110) to the binary sum converts it to the correct BCD representation and also produces an output carry as required.
- To distinguish them from binary 1000 and 1001, which also have a 1 in position Z_8 , we specify further that either Z_4 or Z_2 must have a 1.

$$C = K + Z_8Z_4 + Z_8Z_2$$

Implementation of BCD adder

- A decimal parallel adder that adds n decimal digits needs n BCD adder stages.
- The output carry from one stage must be connected to the input carry of the next higher-order stage.

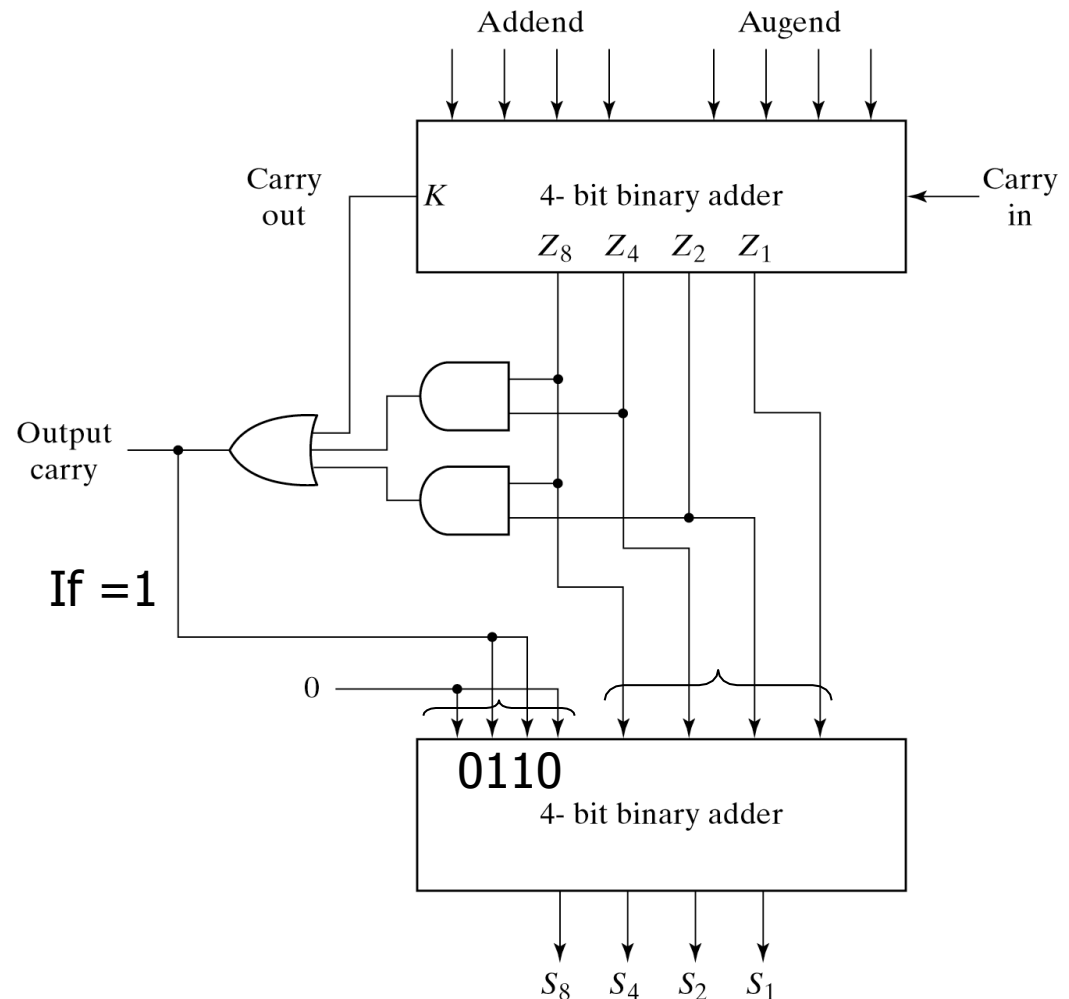


Fig. 4-14 Block Diagram of a BCD Adder

4-6. Binary multiplier

- Usually there are **more bits** in the partial products and it is necessary to use **full adders** to produce the sum of the partial products.

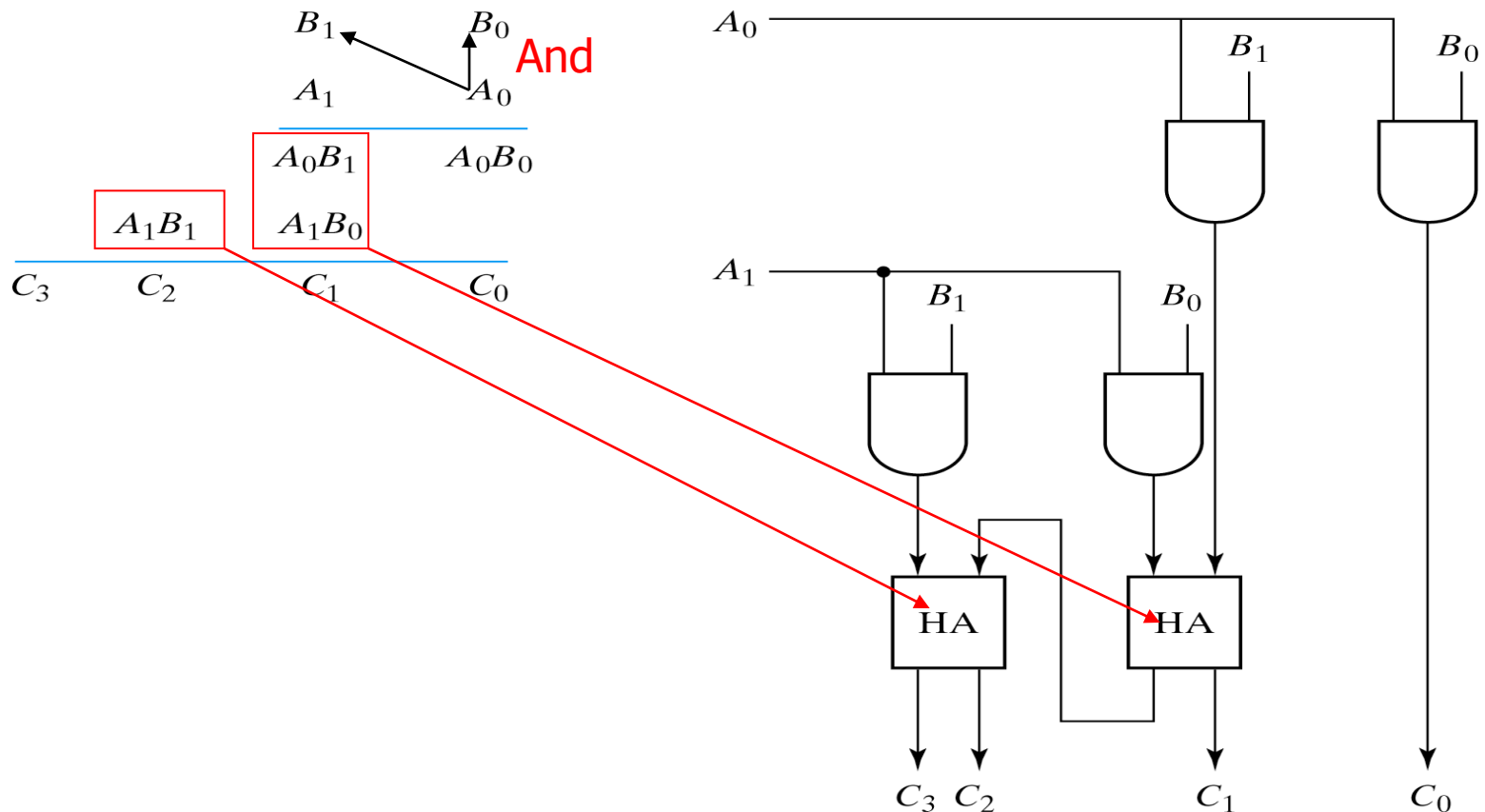


Fig. 4-15 2-Bit by 2-Bit Binary Multiplier

4-bit by 3-bit binary multiplier

- For J multiplier bits and K multiplicand bits we need $(J \times K)$ AND gates and $(J - 1)$ K -bit adders to produce a product of $J+K$ bits.
- $K=4$ and $J=3$, we need 12 AND gates and two 4-bit adders.

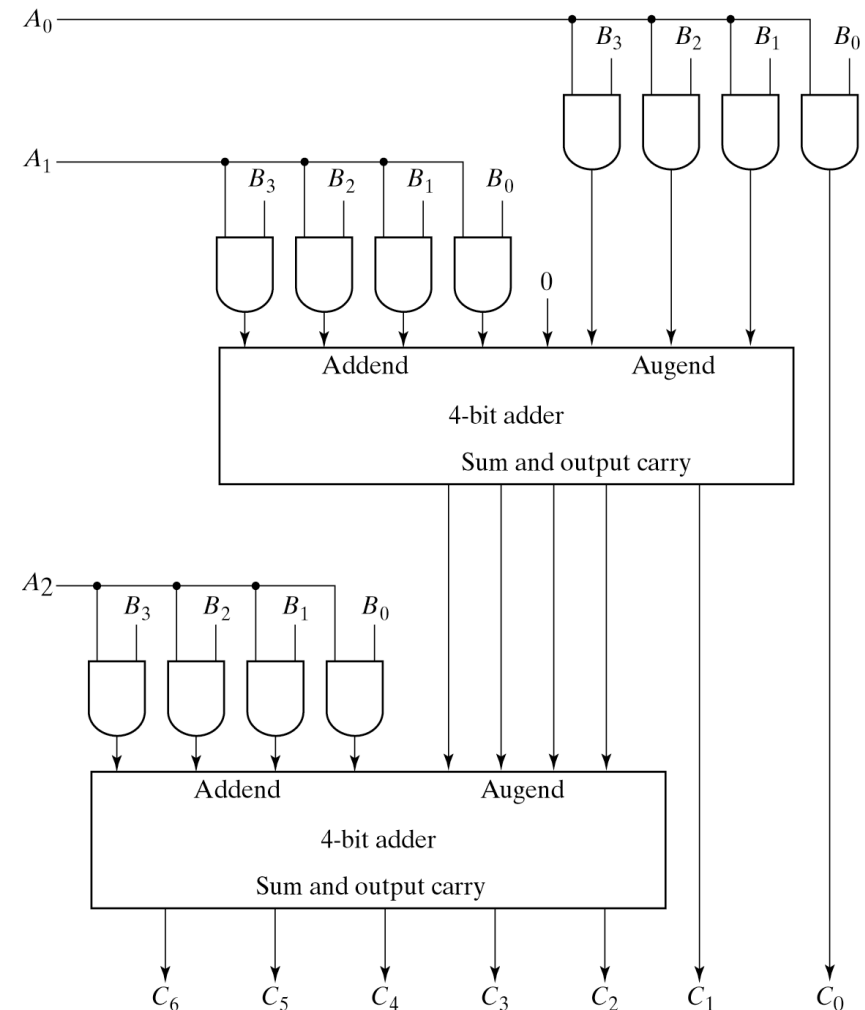


Fig. 4-16 4-Bit by 3-Bit Binary Multiplier

4-7. Magnitude comparator

- The equality relation of each pair of bits can be expressed logically with an exclusive-NOR function as:

$$A = A_3A_2A_1A_0 ; B = B_3B_2B_1B_0$$

$$x_i = A_i B_i + A_i' B_i' \quad \text{for } i = 0, 1, 2, 3$$

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

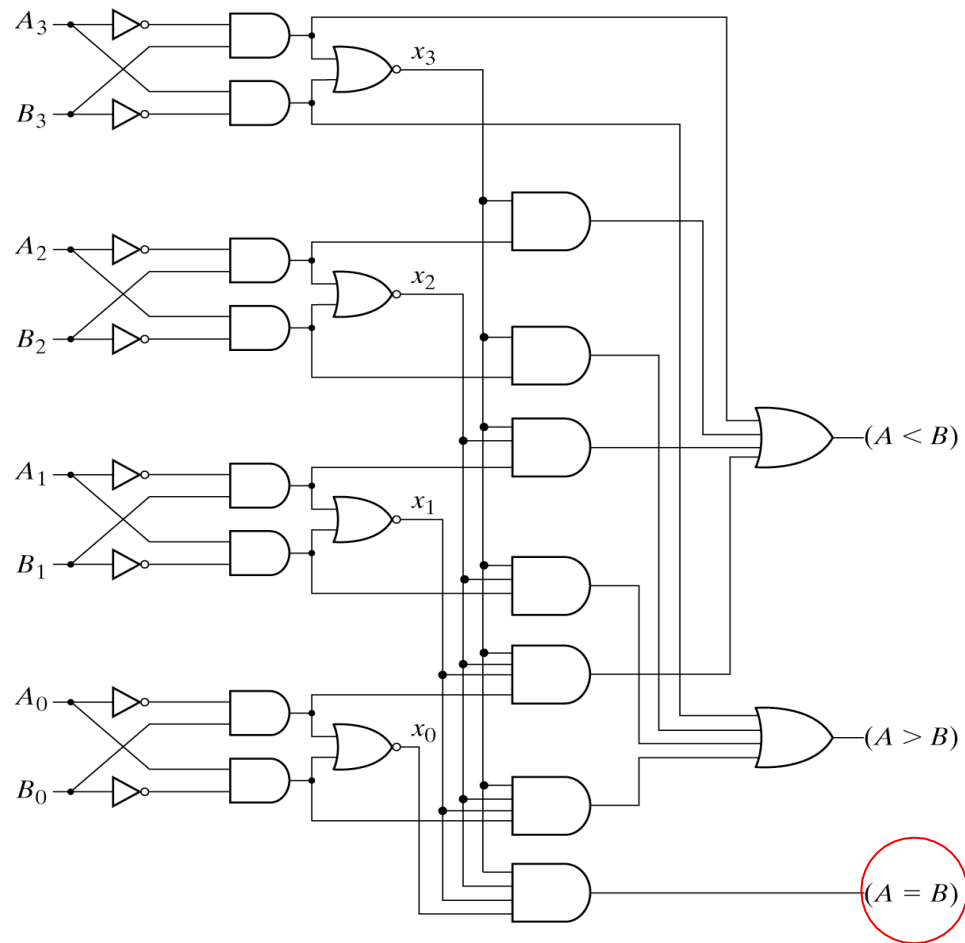


Fig. 4-17 4-Bit Magnitude Comparator

Magnitude comparator

- We inspect the relative magnitudes of pairs of MSB. If equal, we compare the next lower significant pair of digits until a pair of unequal digits is reached.
- If the corresponding digit of A is 1 and that of B is 0, we conclude that $A > B$.

$(A > B) =$

$$A_3B'_3 + x_3A_2B'_2 + x_3x_2A_1B'_1 + x_3x_2x_1A_0B'_0$$

$(A < B) =$

$$A'_3B_3 + x_3A'_2B_2 + x_3x_2A'_1B_1 + x_3x_2x_1A'_0B_0$$

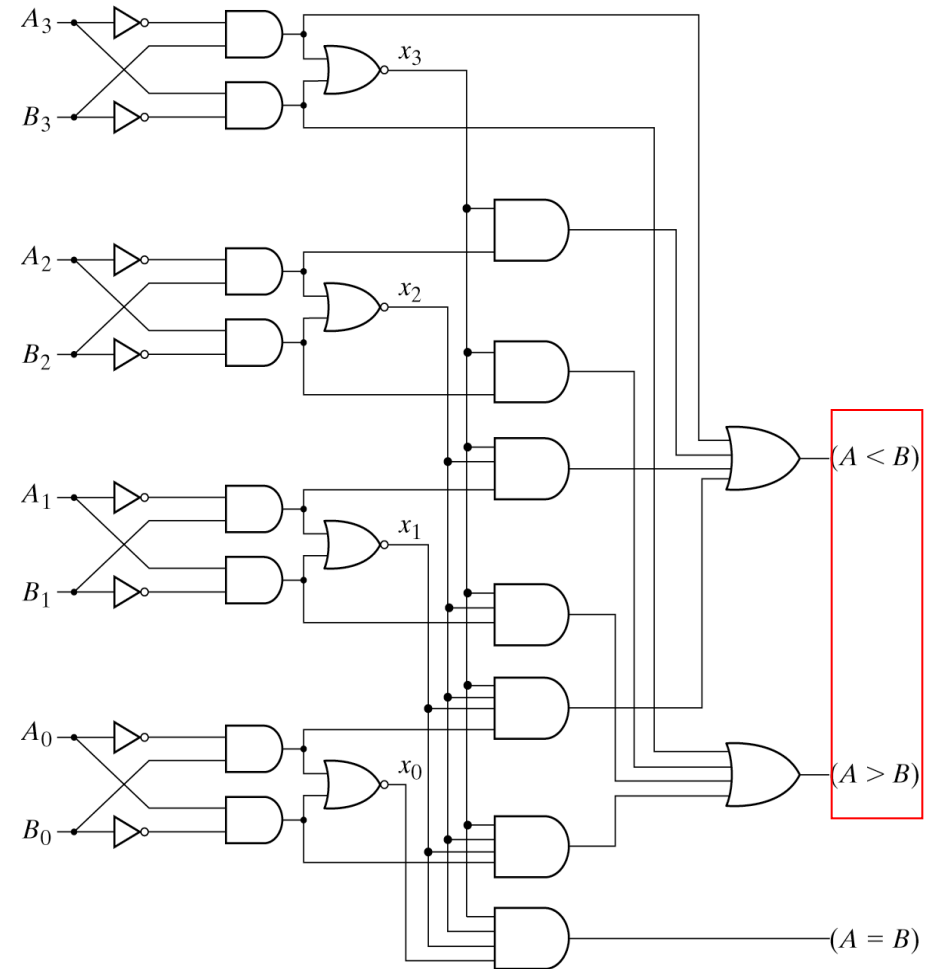


Fig. 4-17 4-Bit Magnitude Comparator



4-8. Decoders

- The decoder is called n-to-m-line decoder, where $m \leq 2^n$.
- the decoder is also used in conjunction with other code converters such as a BCD-to-seven_segment decoder.
- 3-to-8 line decoder: For each possible input combination, there are seven outputs that are equal to 0 and only one that is equal to 1.

Implementation and truth table

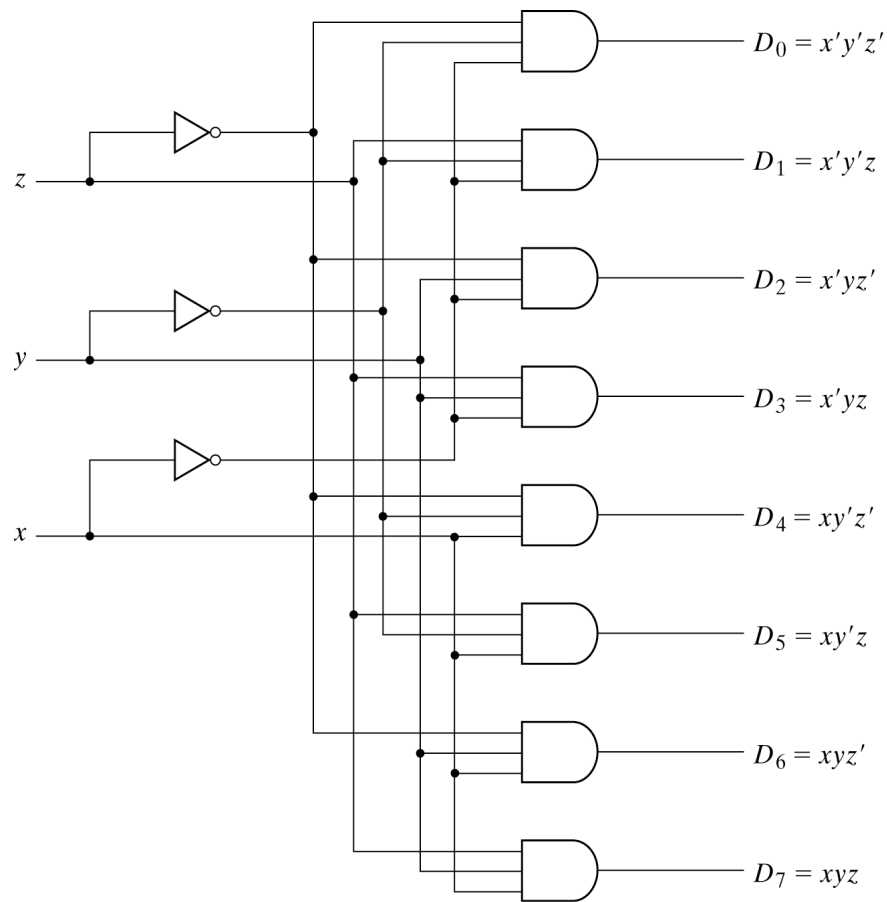


Fig. 4-18 3-to-8-Line Decoder

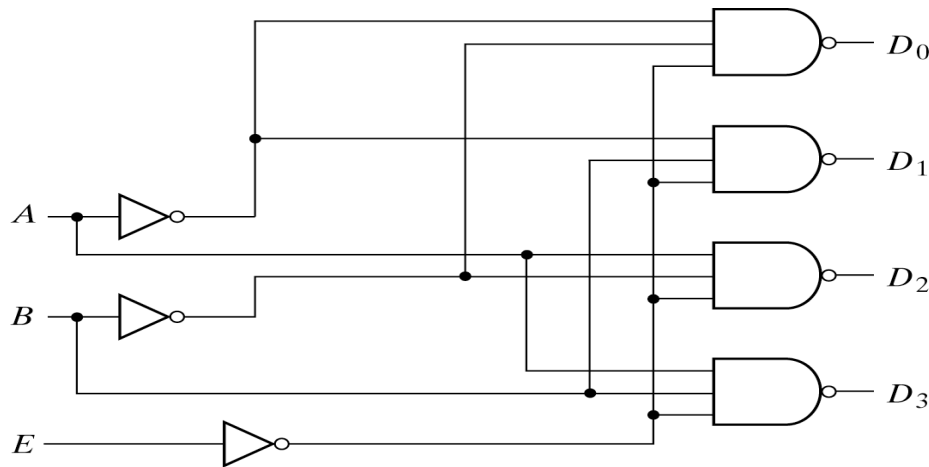
Table 4-6

Truth Table of a 3-to-8-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 with enable input

- Some decoders are constructed with NAND gates, it becomes more economical to generate the decoder minterms in their complemented form.
- As indicated by the truth table, only one output can be equal to 0 at any given time, all other outputs are equal to 1.



(a) Logic diagram

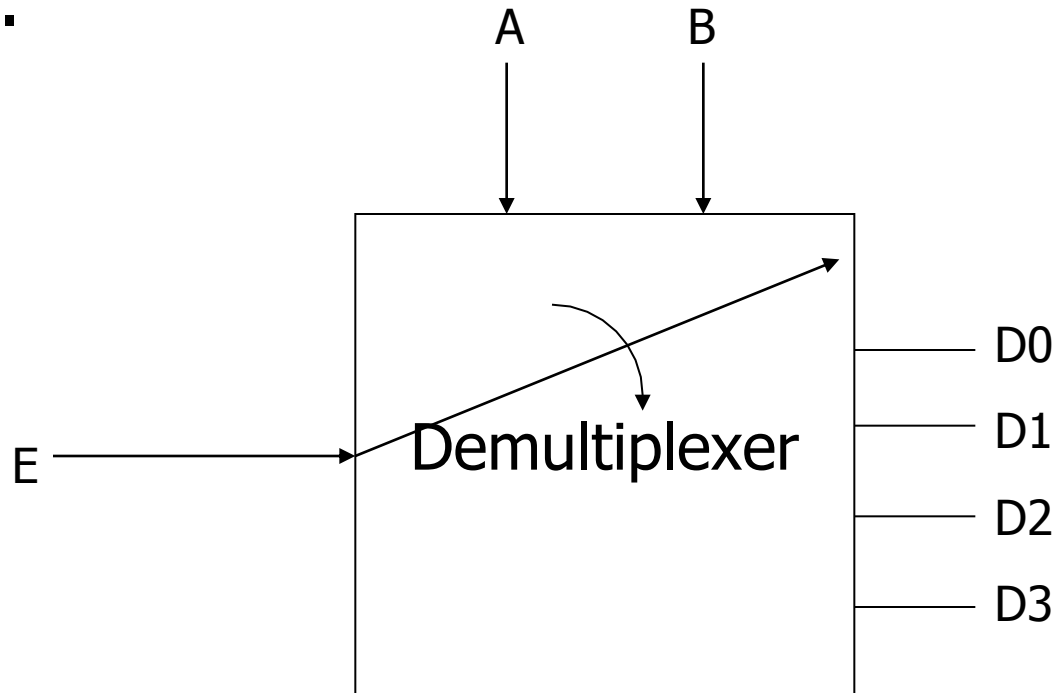
E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

(b) Truth table

Fig. 4-19 2-to-4-Line Decoder with Enable Input

Demultiplexer

- A decoder with an enable input is referred to as a decoder/demultiplexer.
- The truth table of demultiplexer is the same with decoder.



3-to-8 decoder with enable implement the 4-to-16 decoder

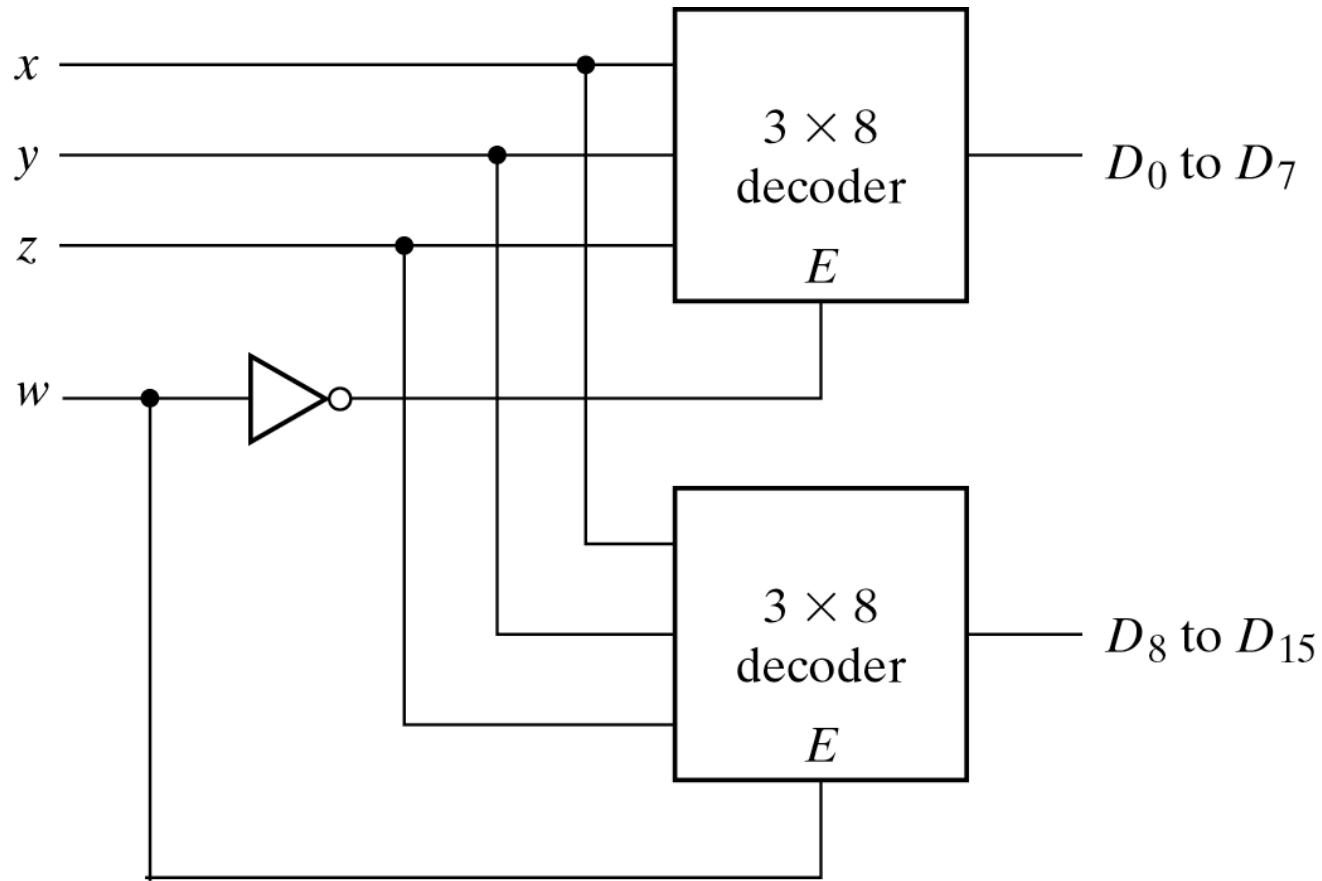


Fig. 4-20 4×16 Decoder Constructed with Two 3×8 Decoders

Implementation of a Full Adder with a Decoder

- From table 4-4, we obtain the functions for the combinational circuit in sum of minterms:

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

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

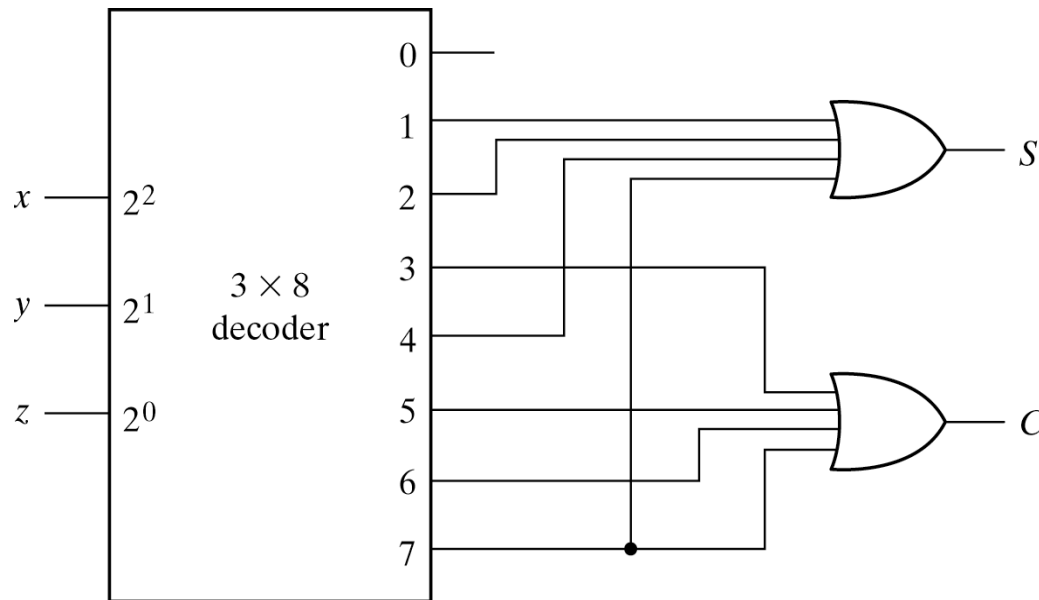


Fig. 4-21 Implementation of a Full Adder with a Decoder

4-9. Encoders

- An **encoder** is the **inverse operation of a decoder**.
- We can derive the Boolean functions by table 4-7

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

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

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

Table 4-7
Truth Table of 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



Priority encoder

- If two **inputs** are **active simultaneously**, the **output** produces an **undefined combination**. We can establish an input **priority** to ensure that only one input is encoded.
- **Another ambiguity** in the octal-to-binary encoder is that an **output with all 0's** is generated when **all the inputs are 0**; the output is the same as when D_0 is equal to 1.
- The discrepancy tables on Table 4-7 and Table 4-8 can **resolve aforesaid condition by providing one more output** to indicate that at least one input is equal to 1.

Priority encoder

$V=0 \rightarrow$ no valid inputs

$V=1 \rightarrow$ valid inputs

X's in output columns represent
don't-care conditions

X's in the input columns are
useful for representing a truth
table in condensed form.

Instead of listing all 16
minterms of four variables.

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

4-input priority encoder

- Implementation of table 4-8

$$x = D_2 + D_3$$

$$y = D_3 + D_1 D'_2$$

$$V = D_0 + D_1 + D_2 + D_3$$

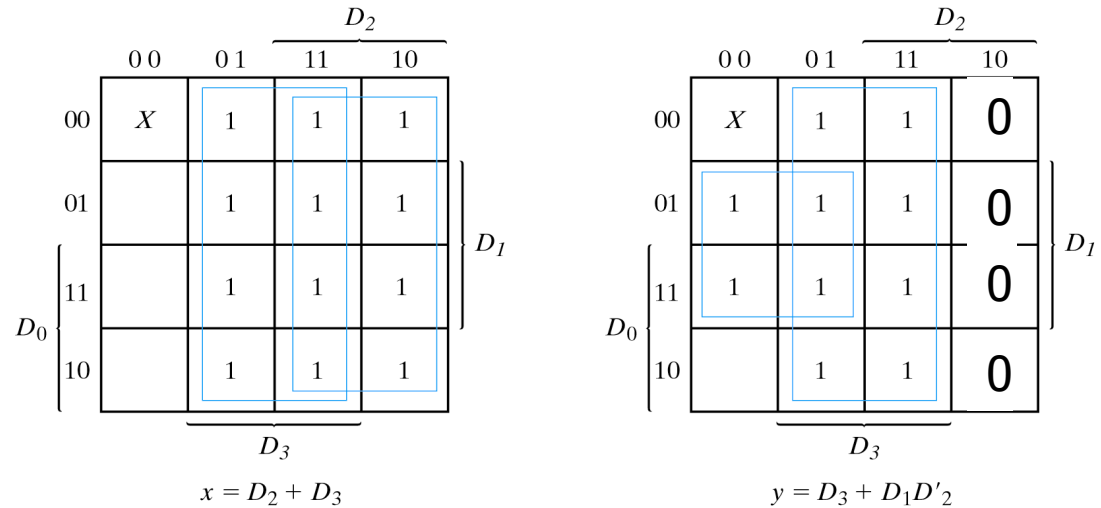


Fig. 4-22 Maps for a Priority Encoder

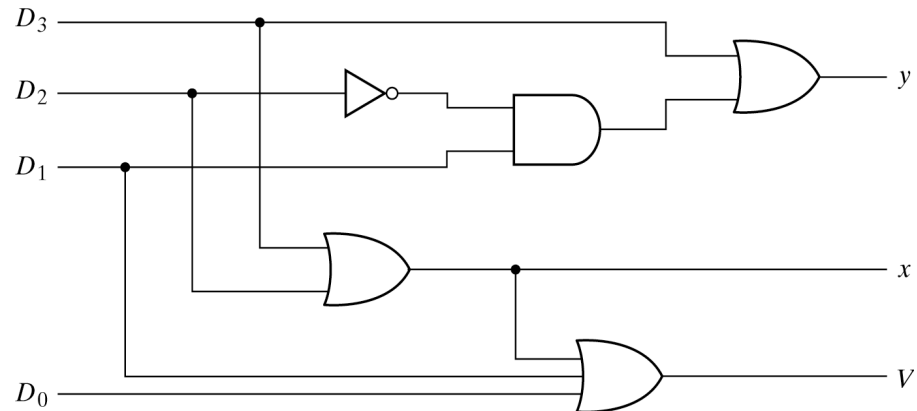


Fig. 4-23 4-Input Priority Encoder

4-10. Multiplexers

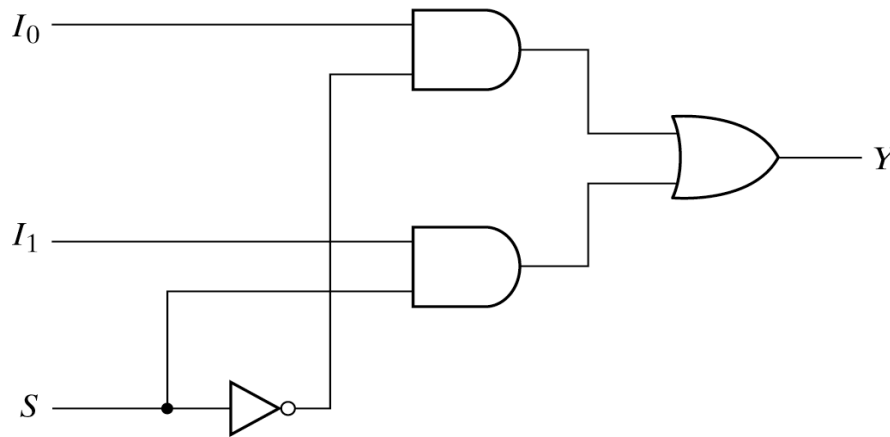
$S = 0, Y = I_0$

$S = 1, Y = I_1$

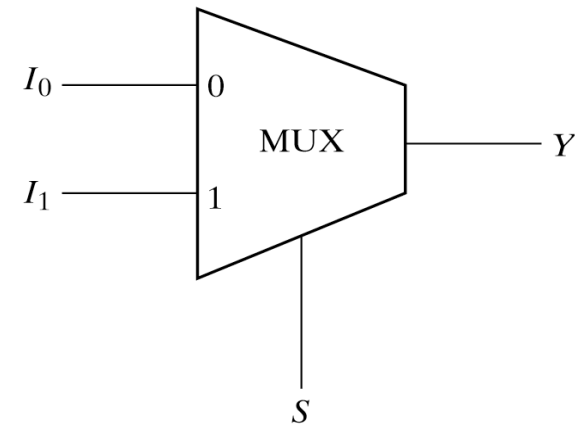
Truth Table \rightarrow

S	Y
0	I_0
1	I_1

$$Y = S'I_0 + SI_1$$



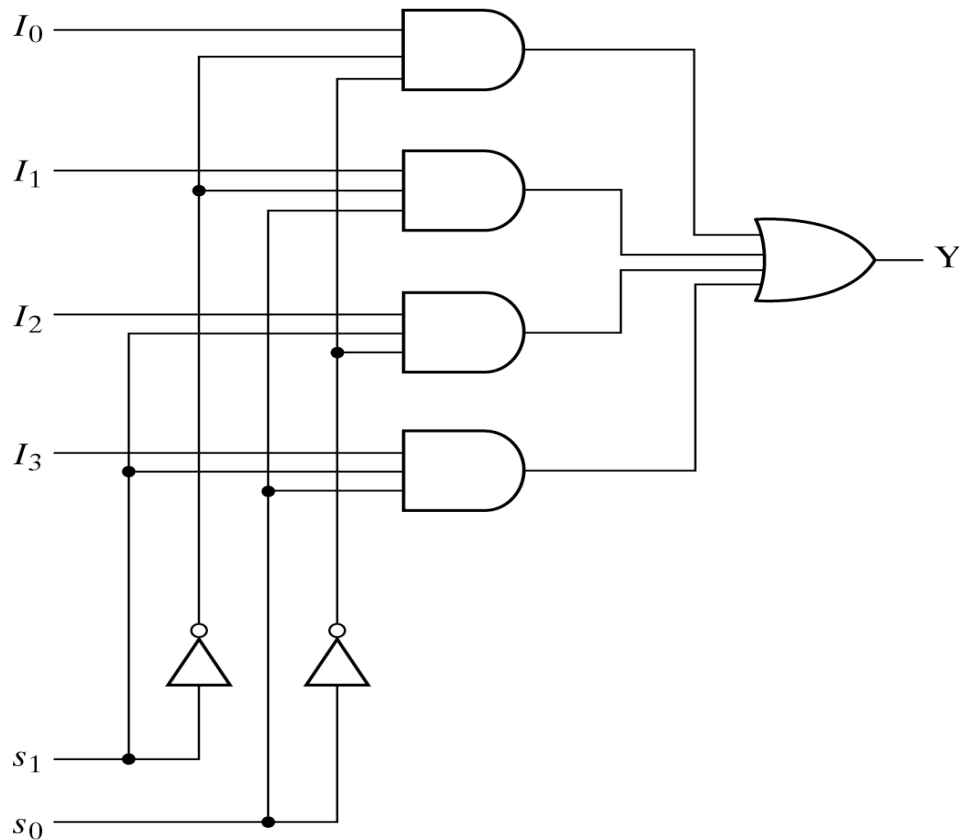
(a) Logic diagram



(b) Block diagram

Fig. 4-24 2-to-1-Line Multiplexer

4-to-1 Line Multiplexer



(a) Logic diagram

s_1	s_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

(b) Function table

Fig. 4-25 4-to-1-Line Multiplexer

Quadruple 2-to-1 Line Multiplexer

- Multiplexer circuits can be combined with common selection inputs to provide multiple-bit selection logic. Compare with Fig4-24.

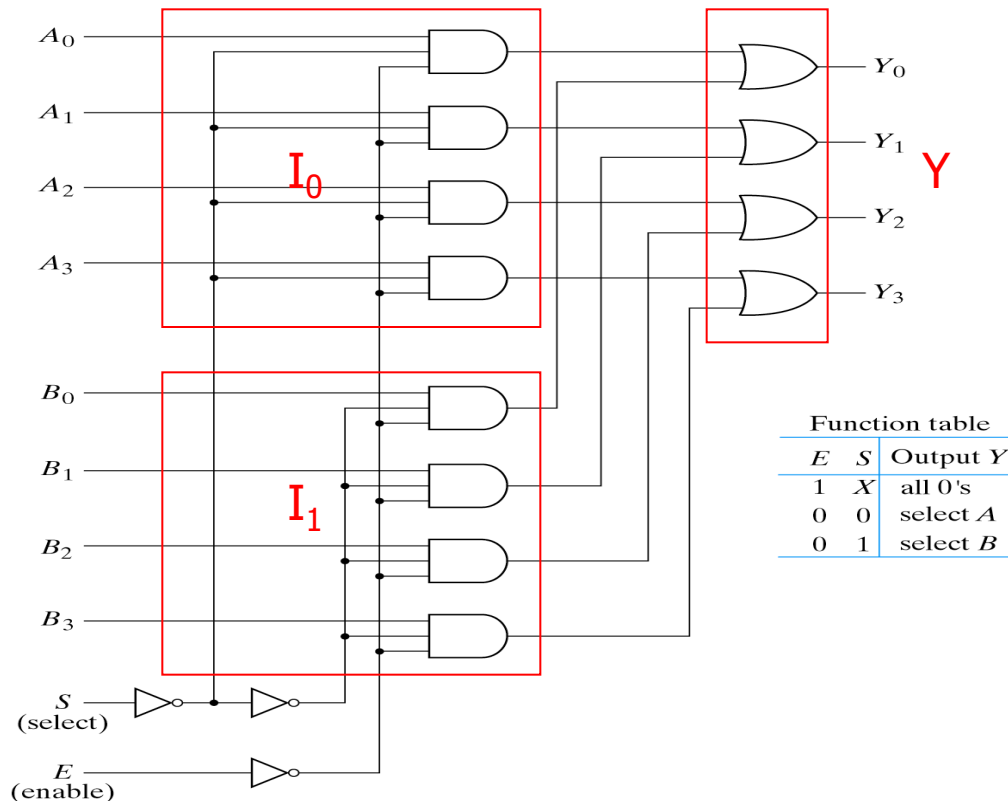


Fig. 4-26 Quadruple 2-to-1-Line Multiplexer

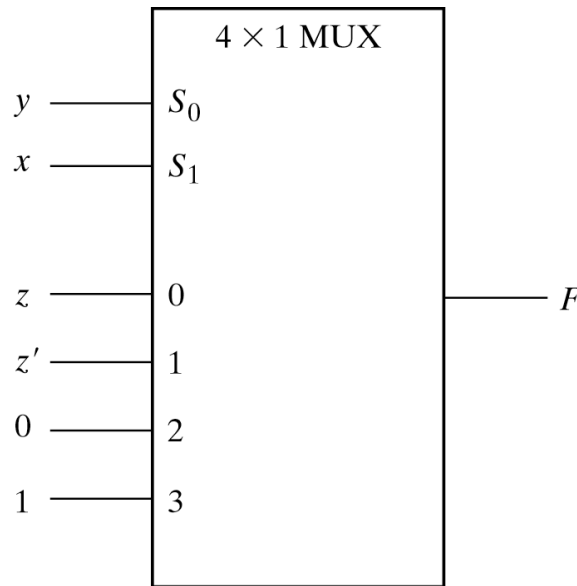
Boolean function implementation

- A more efficient method for implementing a Boolean function of n variables with a multiplexer that has $n-1$ selection inputs.

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

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

(a) Truth table



(b) Multiplexer implementation

Fig. 4-27 Implementing a Boolean Function with a Multiplexer

4-input function with a multiplexer

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

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>F</i>	
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	$F = D'$
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	

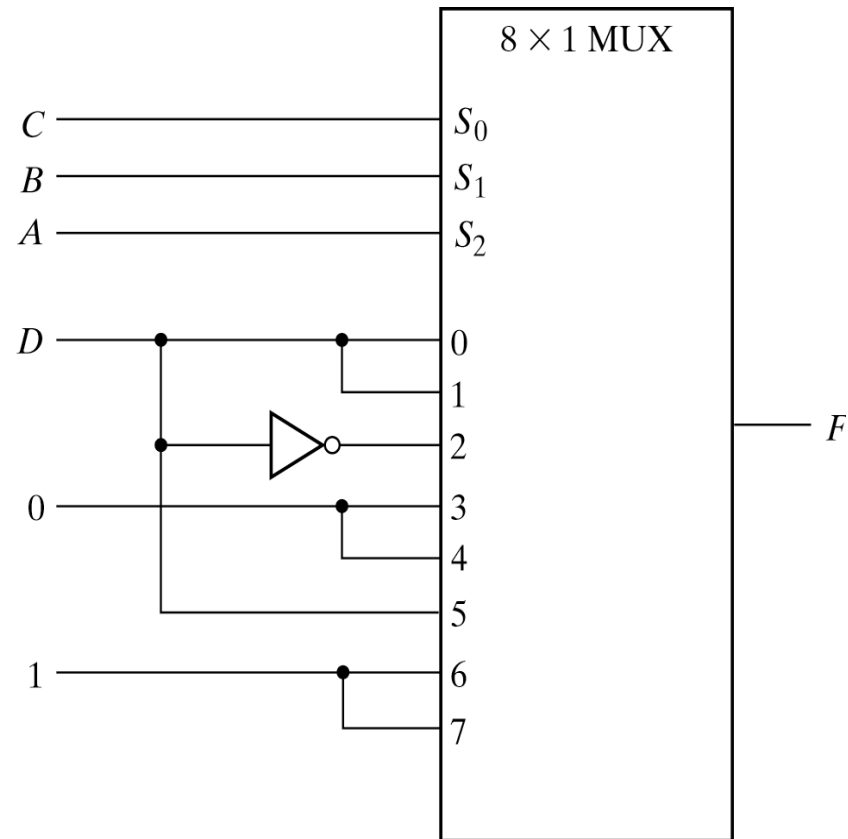


Fig. 4-28 Implementing a 4-Input Function with a Multiplexer