EEA051 - Digital Logic
數位邏輯

Chapter 4
Combinational Logic

吳俊興
國立高雄大學 資訊工程學系

November 2005

Chapter 4 Combinational Logic

Logic Circuits

Combinational Circuits

- Consist of *logic gates* whose outputs at any time are determined from the present combination of inputs
  - input variables, logic gates, and output variables
- The logic gates accept signals from the inputs and generate signals to the output

Sequential Circuits

- Consist of *memory storage elements* and *logic gates*
  - Their outputs are a function of the inputs and the state of the storage elements
  - The state of storage elements is a function of previous inputs
- The outputs of a a sequential circuit also depend on the past inputs

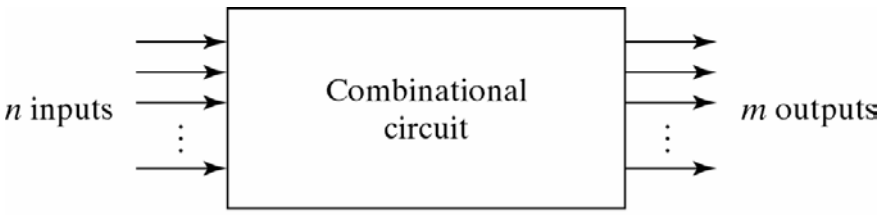


Fig. 4-1 Block Diagram of Combinational Circuit

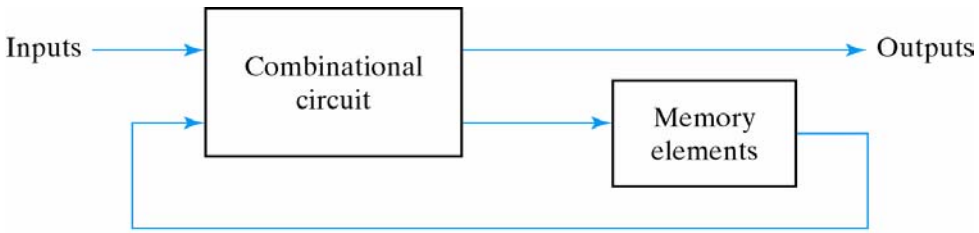


Fig. 5-1 Block Diagram of Sequential Circuit

Combinational Circuits

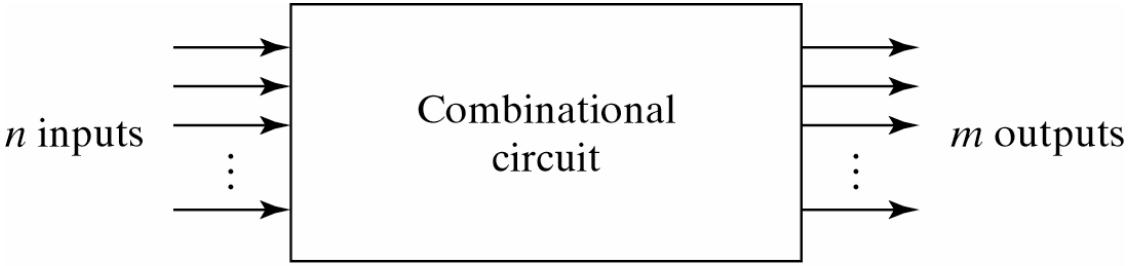


Fig. 4-1 Block Diagram of Combinational Circuit

Transform binary information from the given input data to a required output data

- n input variables
  - 2^n possible binary input combinations
  - $(2^n)^2=2^{2n}$ possible Boolean functions
- m output variables
  - each output function is expressed in terms of the n input variables
  - described by m Boolean functions

Standard combinational circuits

- available in MSI, standard cells in complex VLSI circuits
- i.e. adders, subtractors, comparators, decoders, encoders, multiplexers

4-2 Analysis Procedure

Start with a given logic diagram and culminate with a set of Boolean functions, a truth table, or a possible explanation of the circuit operation

- Make sure the given circuit is combinational and not sequential
  - no feedback paths or memory elements
    - A feedback path is a connection from the output of one gate to the input of a second gate that forms part of the input to the first gate
- Obtain the output Boolean functions or the truth table

Obtaining Boolean Functions

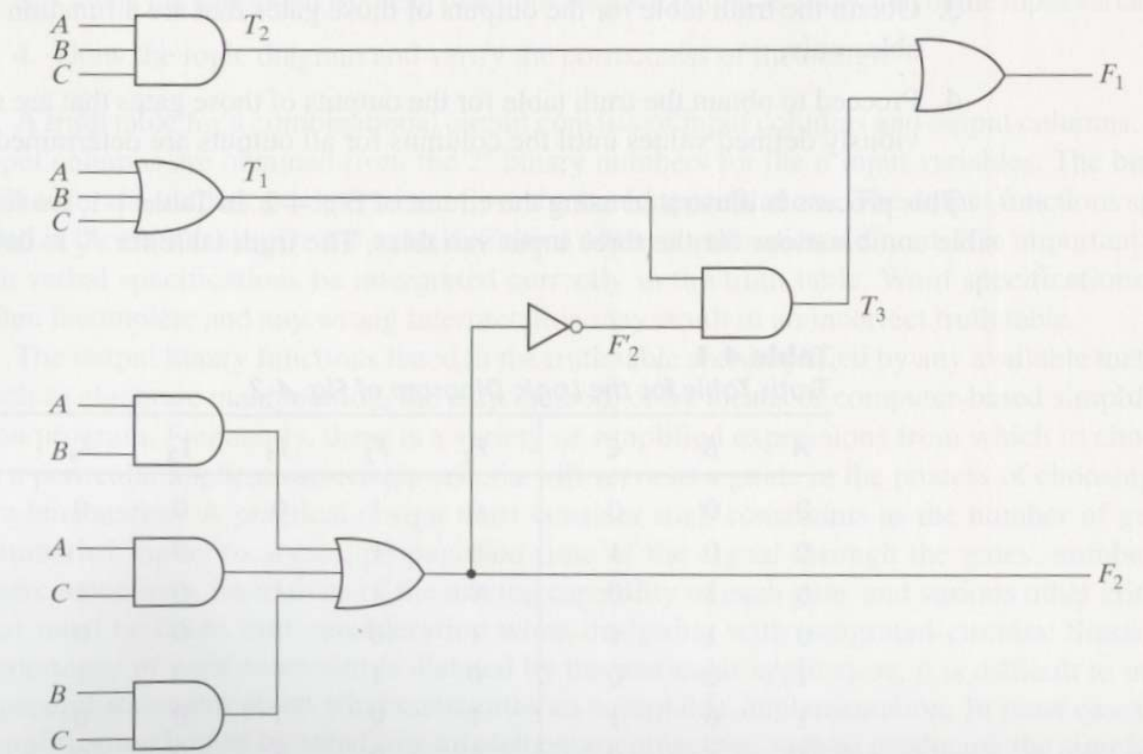


FIGURE 4-2
Logic Diagram for Analysis Example

First, obtain functions of input variables

$$F_2 = AB + AC + BC$$
$$T_1 = A + B + C$$
$$T_2 = ABC$$

Next, we consider outputs of gates that are a function of already defined symbols:

$$T_3 = F'_2 T_1$$
$$F_1 = T_3 + T_2$$

To obtain F_1 as a function of A, B, and C, form a series of substitutions as follows:

$$\begin{aligned} F_1 &= T_3 + T_2 = F'_2 T_1 + ABC = (AB + AC + BC)'(A + B + C) + ABC \\ &= (A' + B')(A' + C')(B' + C')(A + B + C) + ABC \\ &= (A' + B'C')(AB' + AC' + BC' + B'C) + ABC \\ &= A'BC' + A'B'C + AB'C' + ABC \end{aligned}$$

Obtaining Truth Table

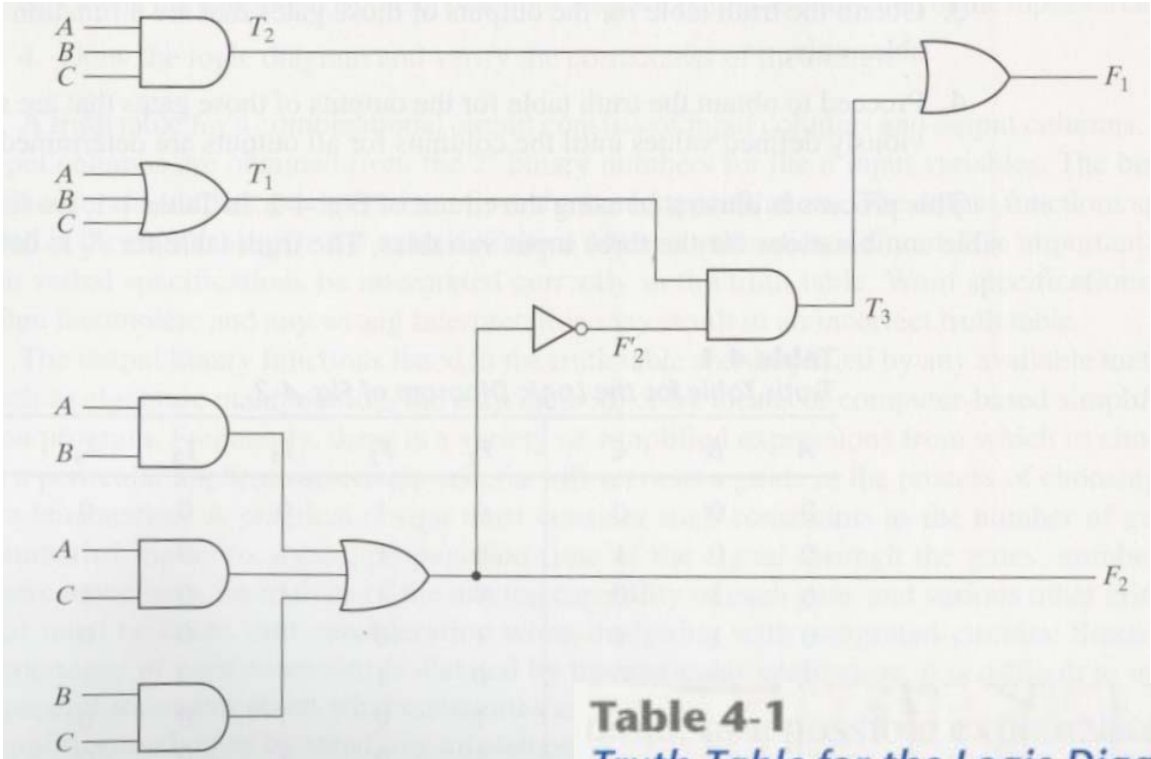


FIGURE 4-2
Logic Diagram for Analysis Example

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

A 3-bit full-adder
F1: the sum
F2: the carry

Obtaining Boolean Functions and Truth Table from a Logic Diagram

Obtain output Boolean functions from a logic diagram:

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.
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.

Obtain the truth table directly from a logic diagram:

1. Determine the number of input variables in the circuit. For n inputs, form the 2^n possible input combinations and list the binary numbers from 0 to $2^n - 1$ in a table.
2. Label the outputs of selected gates with arbitrary symbols.
3. Obtain the truth table for the outputs of those gates that are a function of the input variables only.
4. Proceed to obtain the truth table for the outputs of those gates that are a function of previously defined values until the columns for all outputs are determined.

Problem 4-2

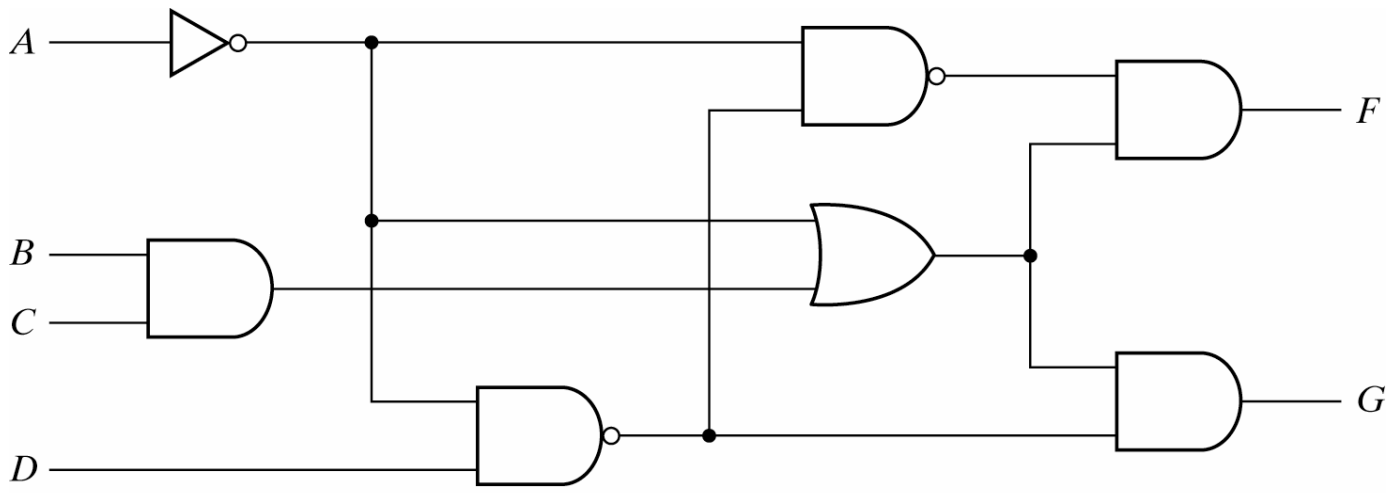


Fig. P4-2

$F = (A+D)(A'+BC) = A'D + ABC + BCD = A'D + ABC$

$G = (A+D')(A'+BC) = A'D' + ABC + BCD' = A'D' + ABC$

AB \ CD	00	01	11	10
00		1	1	
01		1	1	
11			1	1
10				

$A'D + ABC + BCD = A'D + ABC$

AB \ CD	00	01	11	10
00	1			1
01	1			1
11			1	1
10				

$A'D' + ABC + BCD' = A'D' + ABC$

4-3 Design Procedure

The design of combinational circuits:

- start from the specification of the problem and
- culminates in a logic circuit diagram
  or a set of Boolean functions

1. From the specifications of the circuit, determine the required number of inputs and outputs and assign a symbol to each.
3. Derive the truth table that defines the required relationship between inputs and outputs.
3. Obtain the simplified Boolean functions for each output as a function of the input variables.
4. Draw the logic diagram and verify the correctness of the design.

Constraints in a practical design:

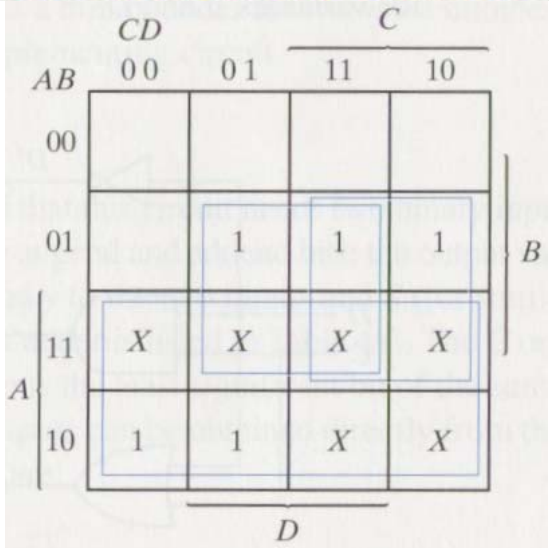
- number of gates
- number of inputs to a gate
- propagation time of the signal through the gates
- number of interconnections
- limitations of the driving capability of each gate
- etc.

Code Conversion Example – BCD to Excess-3 Code

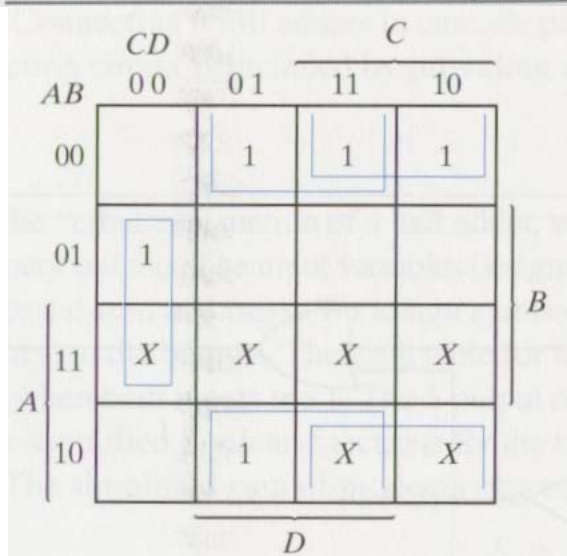
Table 4-2
Truth Table for Code-Conversion Example

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

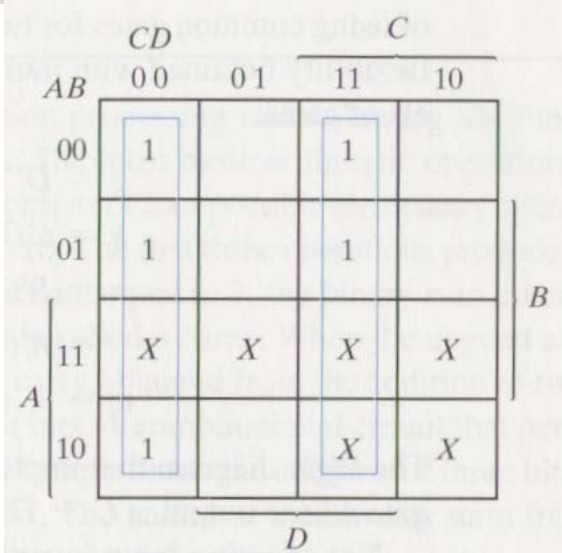
FIGURE 4-3
Maps for BCD to Excess-3 Code Converter



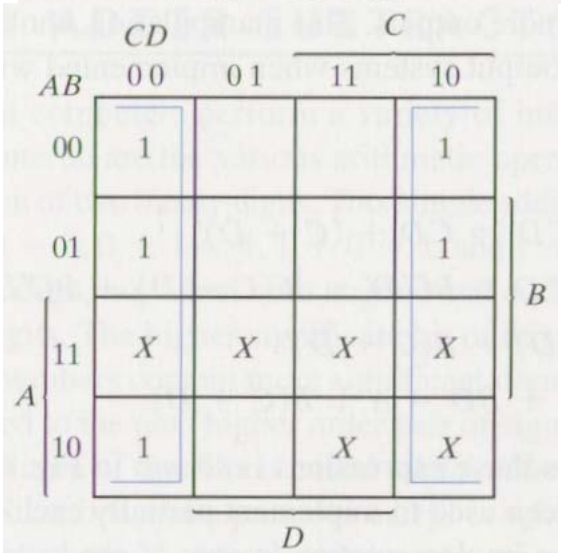
$w = A + BC + BD$



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



$y = CD + C'D'$



$z = D'$

Code Conversion Example – BCD to Excess-3 Code *(cont.)*

$$z = D'$$

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

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

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

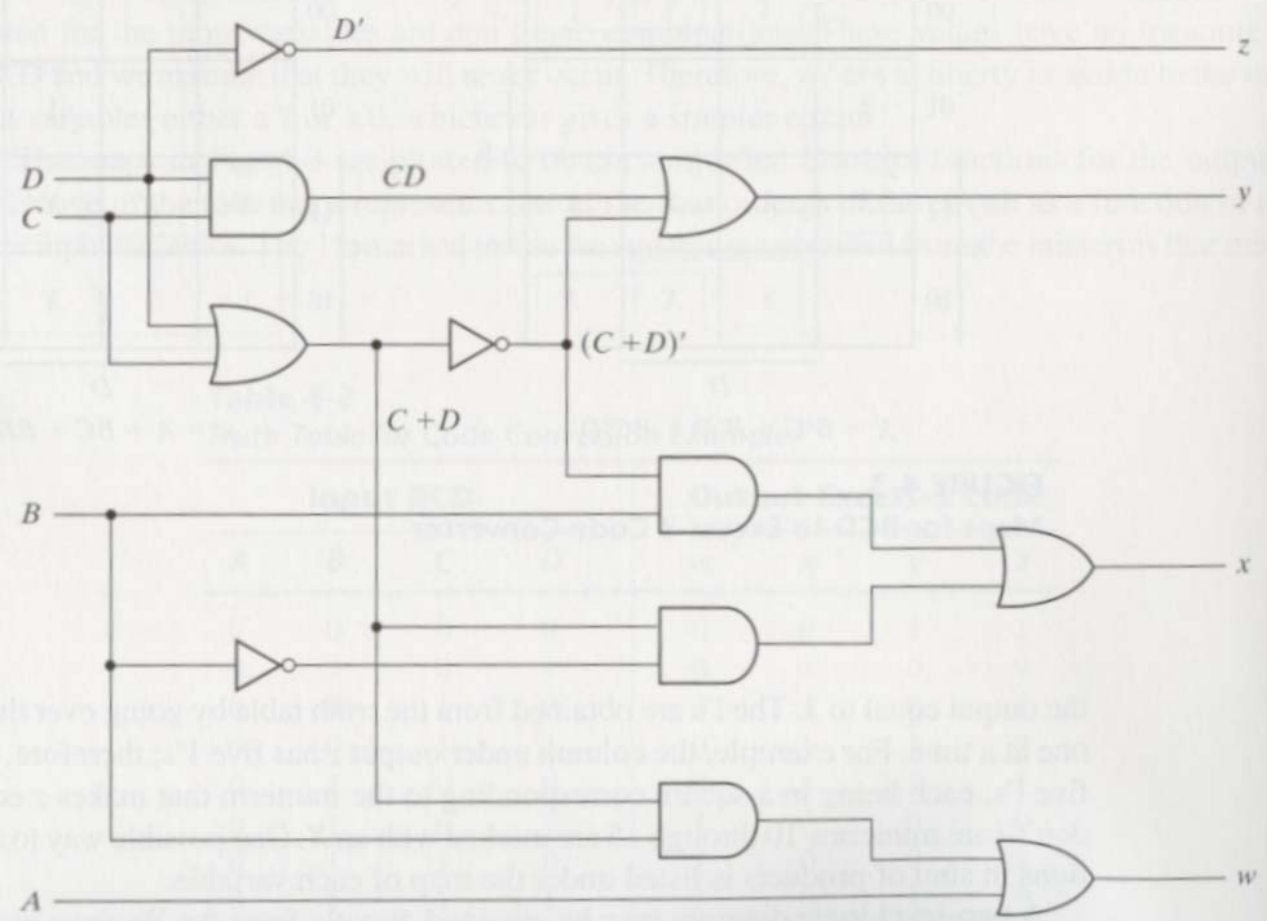


FIGURE 4-4
Logic Diagram for BCD to Excess-3 Code Converter

Two-level implementation (Figure 4-3)
- 7 AND gates
- 3 OR gates

Multiple-level two-input implementation (Figure 4-4)
- 4 AND gates
- 4 OR gates

Problem 4-6 Majority Circuit

A majority circuit is a combinational circuit whose output is equal to 1 if the input variables have more 1's than 0's. The output is 0 otherwise. Design a 3-input majority circuit.

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

x \ yz	00	01	11	10
0			1	
1		1	1	1

$$F = xy + xz + yz$$

4-4 Binary Adder-Subtractor

Addition of two binary digits

- The most basic arithmetic operation
- 4 possible elementary operations
  - one-digit sum: 0 + 0 = 0, 0 + 1 = 1, 1 + 0 = 1, and
  - two-digit sum: 1 + 1 = 10 (augend, addend, carry)

- Half adder
  - A combinational circuit that performs the addition of two bits
- Full adder
  - A combinational circuit that performs the addition of three bits (two significant bits and a previous carry)
  - can be implemented by two half adders
  - Connecting n full addres in cascade produces a binary adder for two n-bit numbers
- Binary adder-subtractor
  - A combinational circuit that performs the arithmetic operations of addition and subtraction with binary numbers

Half Adder - Addition of Two Bits

Table 4-3
Half Adder

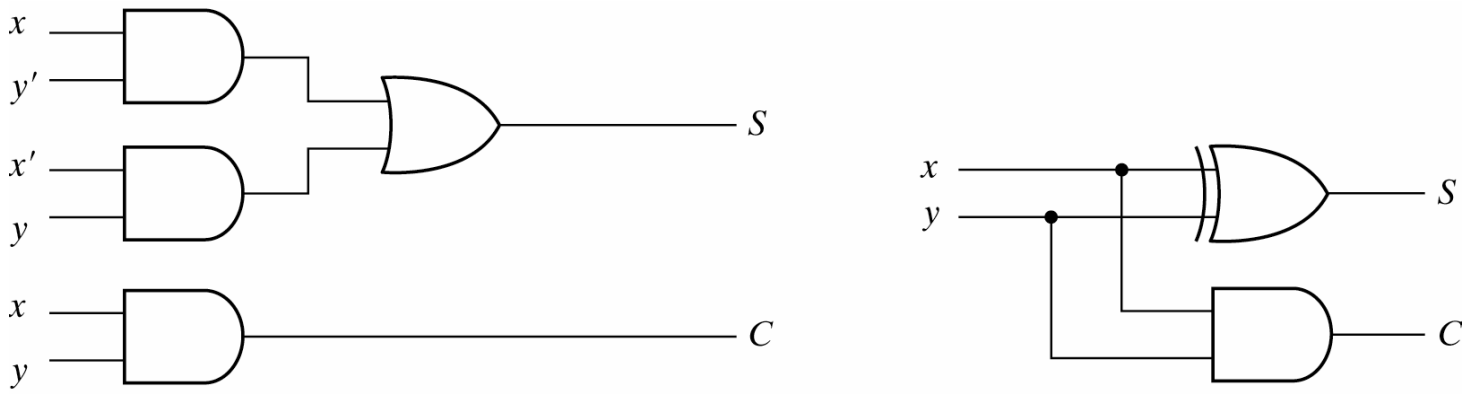
x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Two inputs: x and y

Two outputs:

- Sum S = x'y + xy'
- Carry C = xy

It can also be implemented with an exclusive-OR and an AND gate



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

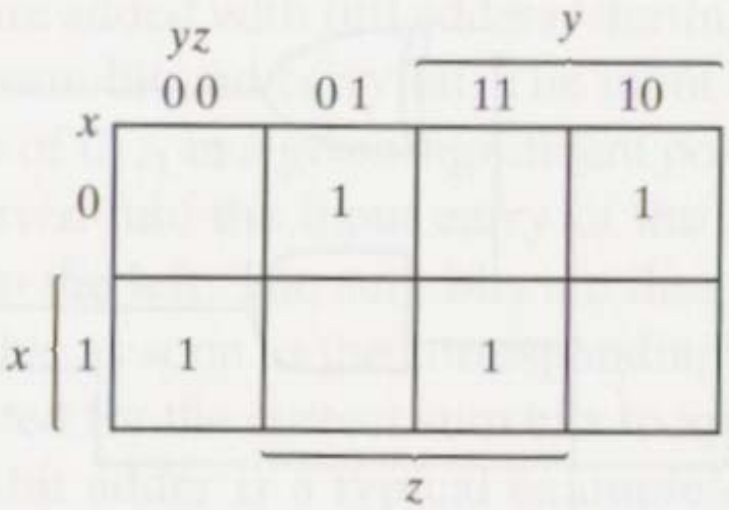
(b) $S = x \oplus y$
$C = xy$

Fig. 4-5 Implementation of Half-Adder

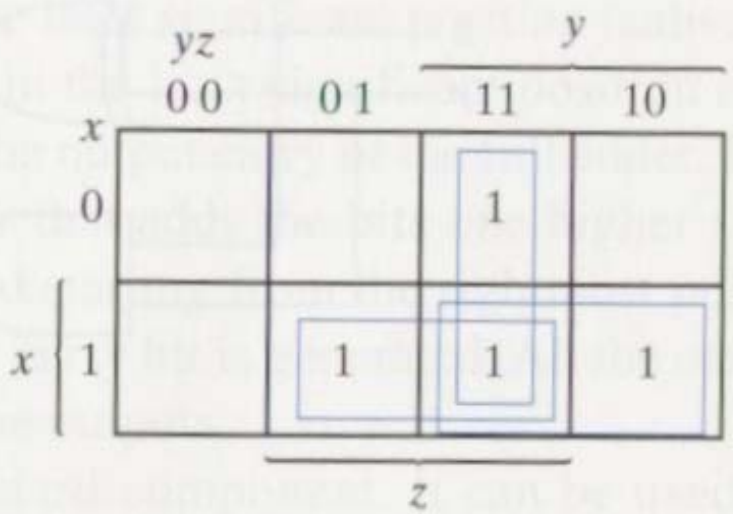
Full-Adder – Sum of Three Bits

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



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



C $S = xy + xz + yz$
$= xy + xy'z + x'yz$

FIGURE 4-6 Maps for Full Adder

Implementations of Full-Adder

Two-level AND-OR Implementation

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

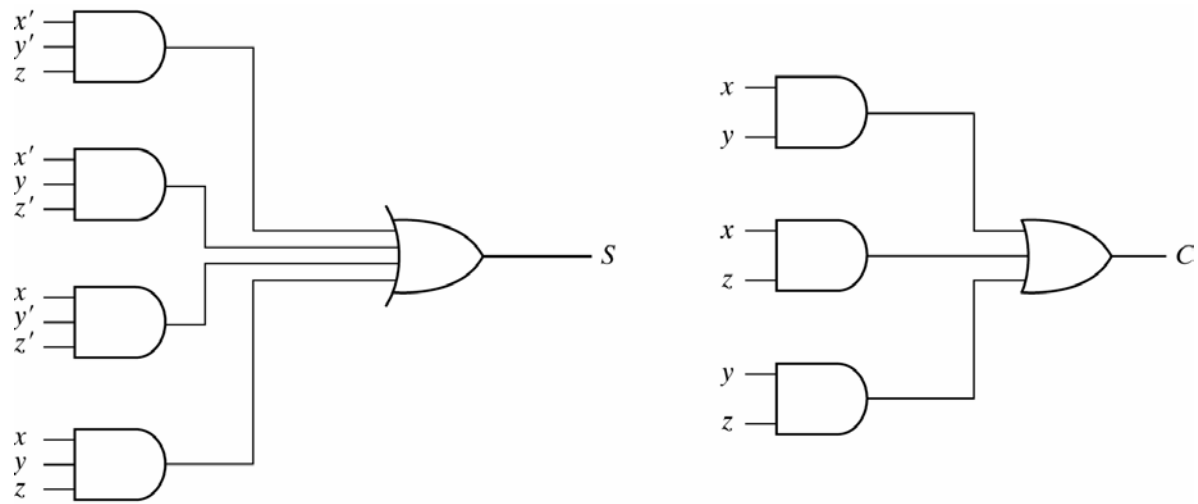


Fig. 4-7 Implementation of Full Adder in Sum of Products

Implemented with two half adders and one OR gate

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

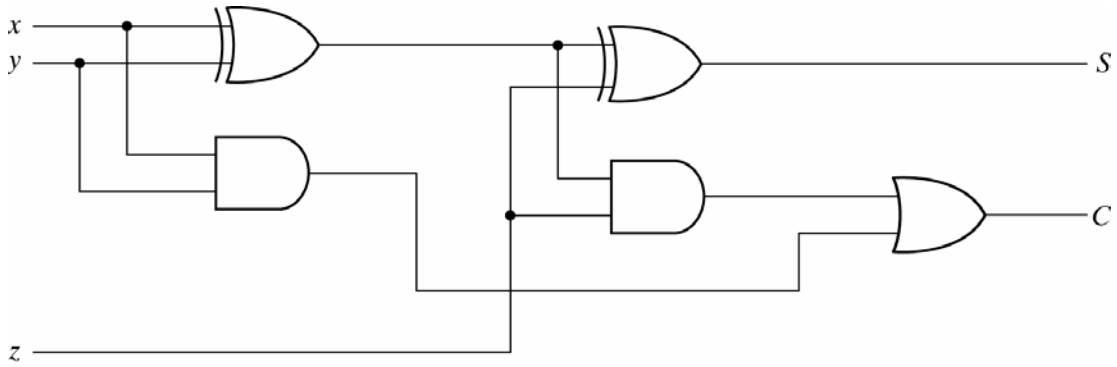


Fig. 4-8 Implementation of Full Adder with Two Half Adders and an OR Gate

Binary Adder – Sum of Two Binary Numbers

- An n-bit adder requires n full adders with each output carry connected to the input carry of the next higher-order full adder
- 4-bit adder: Interconnection of four full adder (FA) circuits

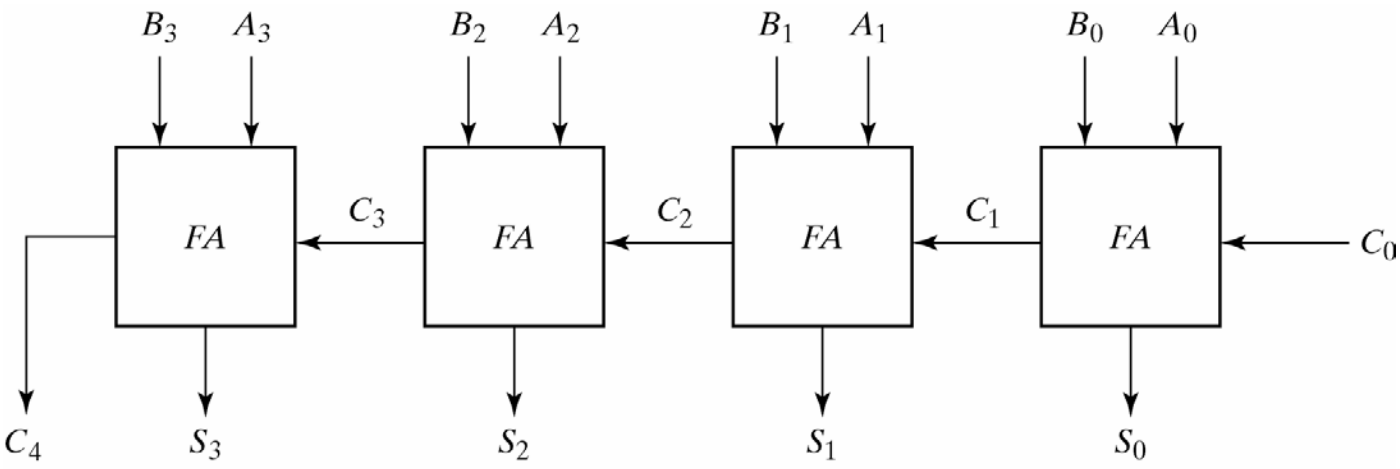


Fig. 4-9 4-Bit Adder

Example: A=1011
and B=0011

Subscript 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}

Carry Propagation

- Total propagation time
  = propagation delay of a typical gate
  * number of gate levels in the circuit
- The longest propagation delay time in an adder is the time it takes the carry to propagate through the full adders
  - Inputs A_i and B_i are available as soon as input signals are applied to the adder
  - The value of S_i in any given stage will be in its steady state final value only after the input carry to that stage has been propagated
  - C_3 has to wait for C_2, C_2 has to wait for C_1 and so on down to C_0
- A limiting factor on the speed with which two numbers are added

Carry Propagation of a Full Adder

- The signal from the input carry C_i to the output carry C_{i+1} propagates through an AND gate and an OR gate, which constitute two gate levels
  - For an n-bit adder, there are 2n gate levels for the carry to propagate from input to output
- 4-bit adder
  - Carry propagation: 2 * 4 = 8 gate levels from C_0 to C_4
  - Critical path: 9 gate levels (3 for C_0)

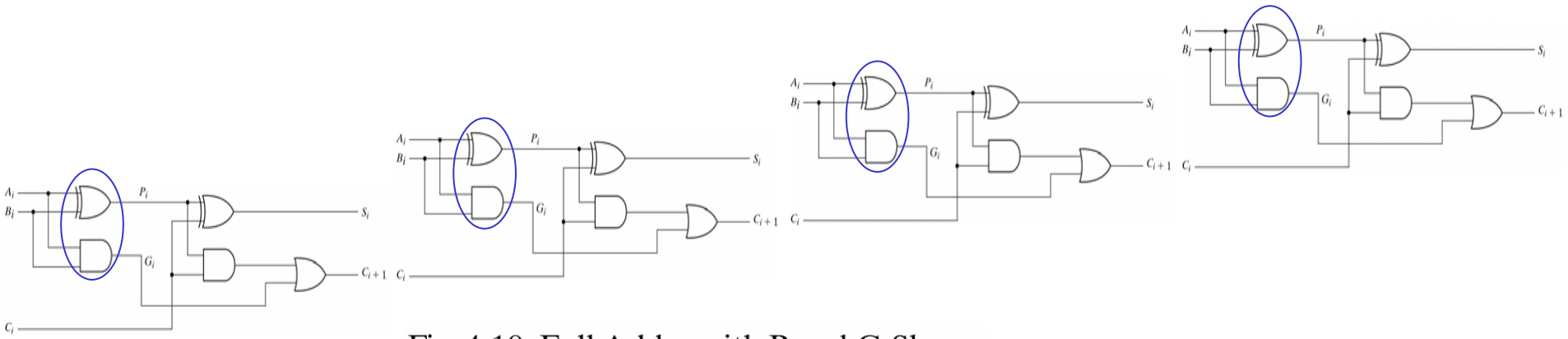


Fig. 4-10 Full Adder with P and G Shown

Reducing Carry Propagation Delay

1. Employ faster gates with reduced delays
   - Physical circuits have a limit to their capability
2. Complicated techniques for parallel adders
   - Principle of ***carry lookahead***

Definitions of two new variables

- carry generate G_i: produces a carry of 1 when both A_i and Bi are 1, regardless of the input carry C_i
- carry propagate P_i: the term associated with the propagation of the carry from C_i to C_{i+1}

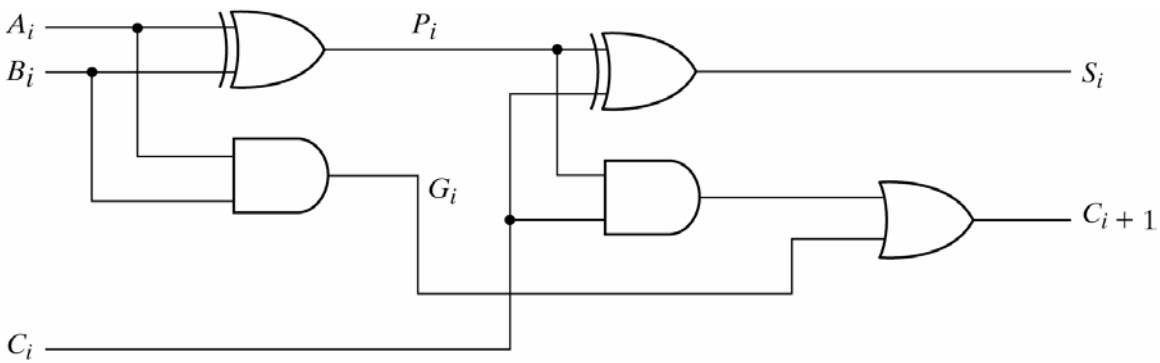


Fig. 4-10 Full Adder with P and G Shown

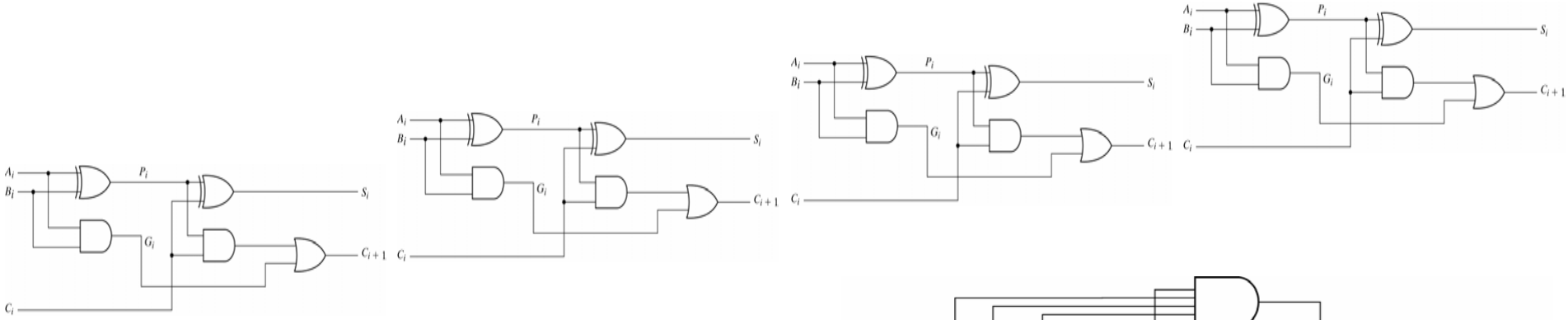
$P_i = A_i \oplus B_i$

$G_i = A_i B_i$

sum $S_i = P_i \oplus C_i$

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

Carry Lookahead Generator



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

C_0 = input carry

$C_1 = G_0 + P_0 C_0$

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

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

P_i, G_i, and C_i do not have to wait for carries from previous stages

$P_i = A_i \oplus B_i$

$G_i = A_i B_i$

Implemented by two-level AND-OR

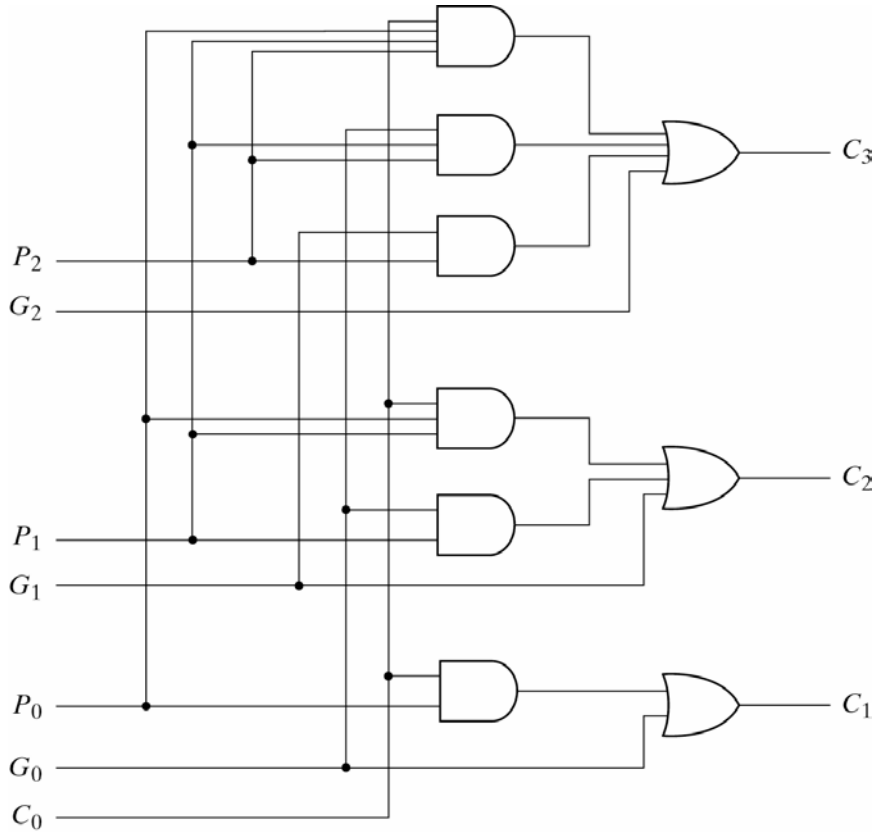


Fig. 4-11 Logic Diagram of Carry Lookahead Generator

4-Bit Adder with Carry Lookahead

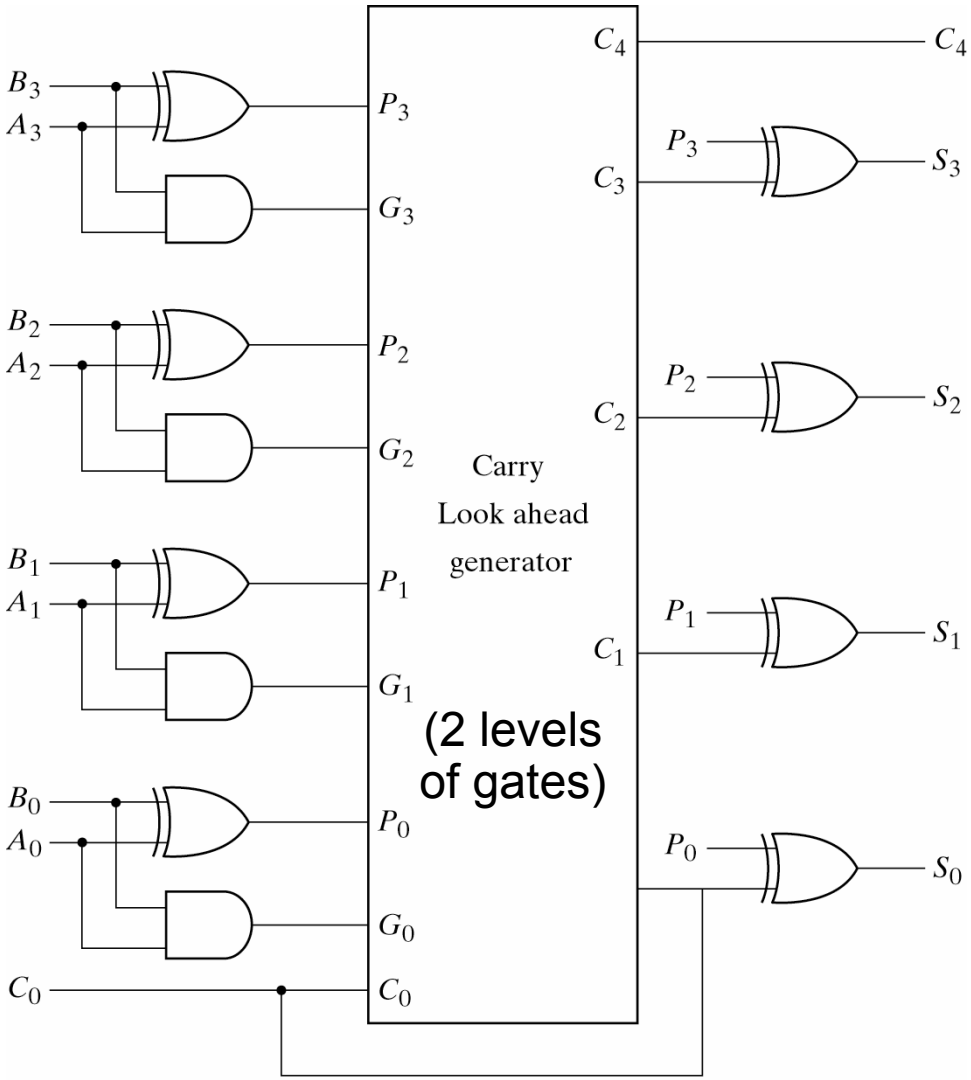


Fig. 4-12 4-Bit Adder with Carry Lookahead

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

All output carries are generated concurrently by the carry lookahead generator after a delay through two levels of gates

Binary Subtractor

Subtracting A – B

= A + (1's complement of B) + 1

$= A + ((r^n-1) - B)) - r^n + 1$

= A + (2's complement of B)

$= A + (r^n - B) - r^n$

V: to detect an overflow for signed numbers

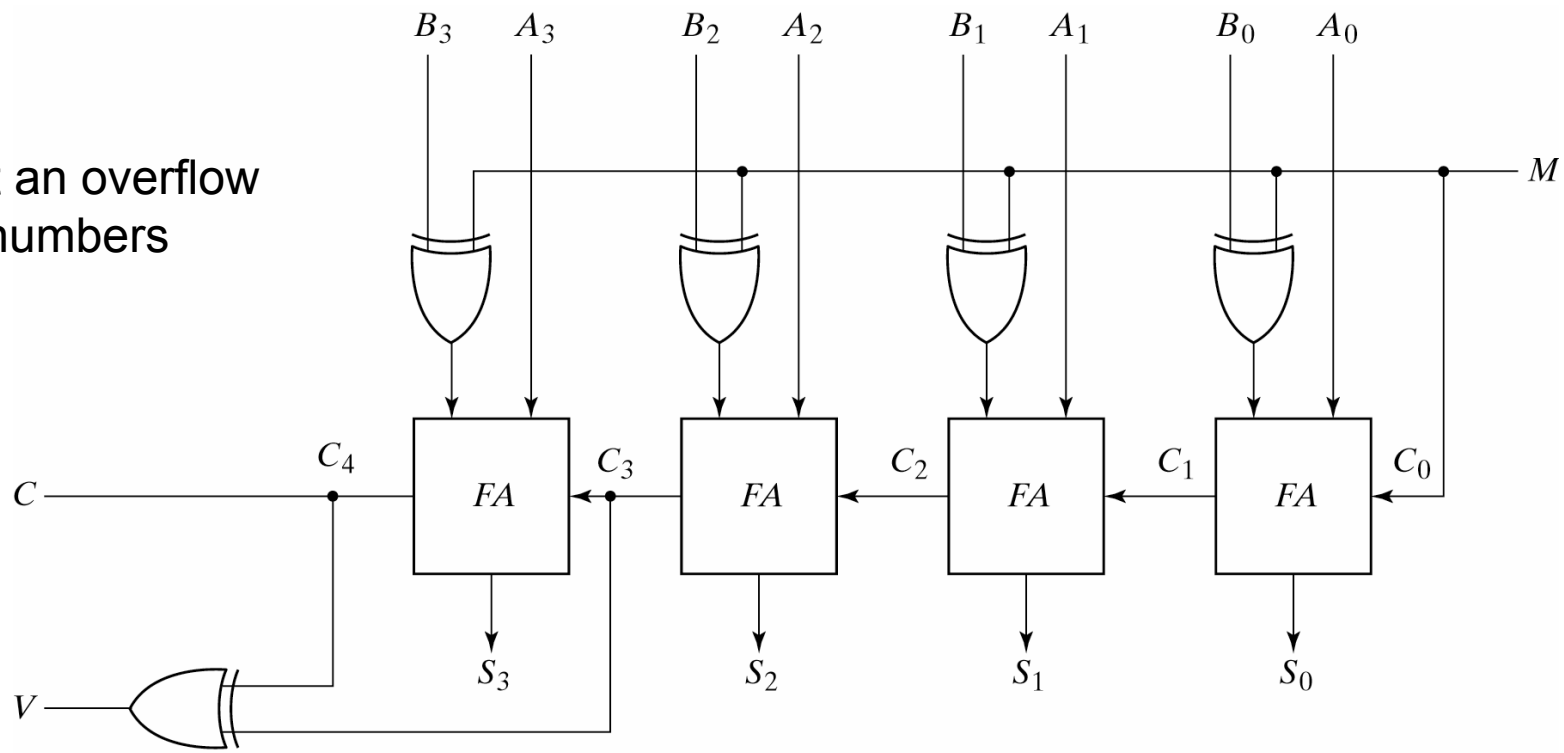


Fig. 4-13 4-Bit Adder Subtractor

4-Bit Adder Subtractor

M=0, the circuit is an adder ($B \oplus 0 = B$)

M=1, the circuit is a subtractor ($B \oplus 1 = B'$, $C_0=1$)

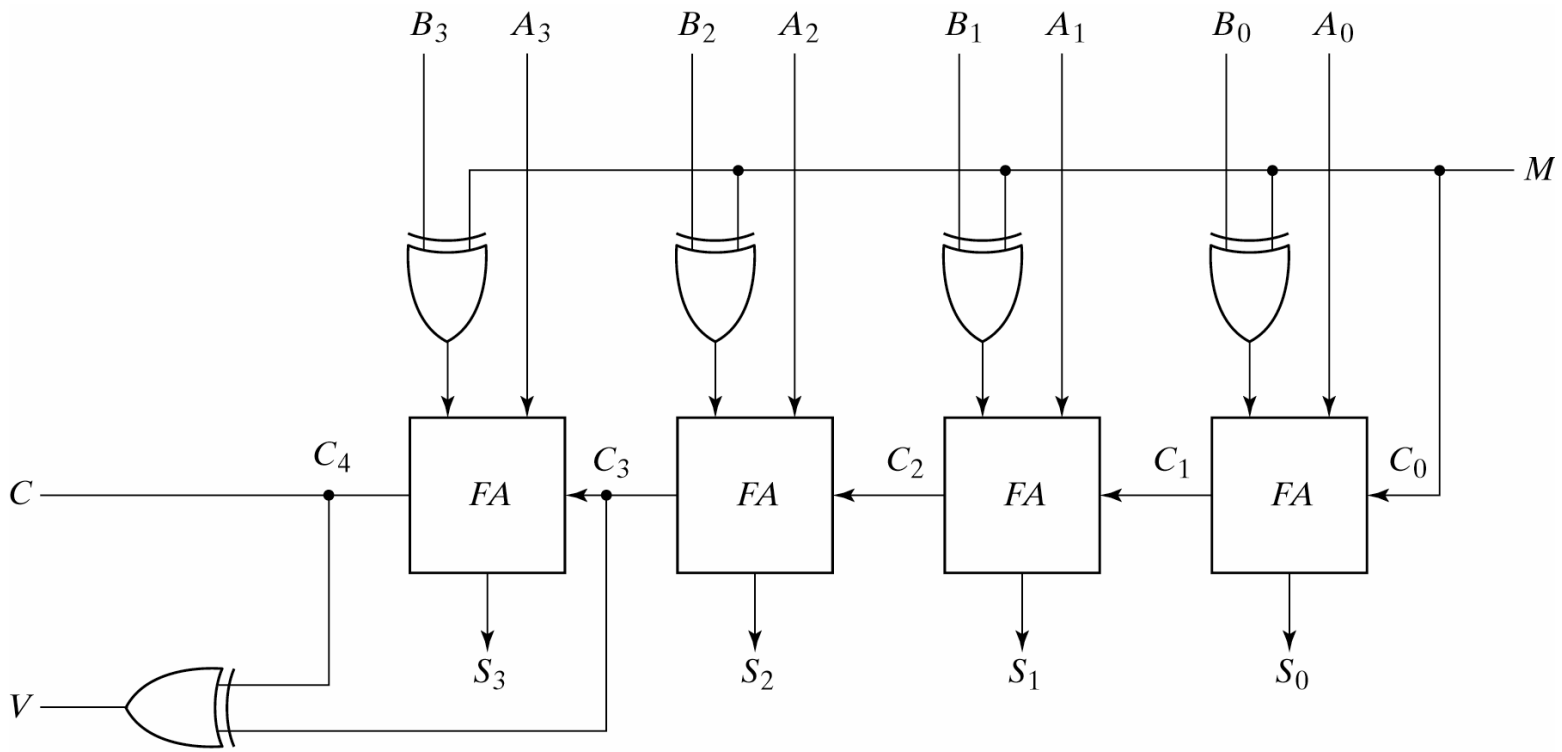


Fig. 4-13 4-Bit Adder Subtractor

Binary numbers in the signed-complement system are added and subtracted by the same basic addition and subtraction rules as unsigned numbers

Overflow

When two numbers of n digits each are added and the sum occupies n+1 digits, an overflow occurs

- Addition of two unsigned numbers: detected from the end carry of the most significant position
- Addition of two signed numbers: May occur if the two numbers are both positive or both negative

•Signed numbers

- the leftmost bit always represents the sign and negative numbers are in 2's complement form
- Addition: the sign bit is treated as part of the number and the end carry does not indicate an overflow

Overflow for Signed Numbers

Examples: two signed binary numbers, +70 and +80, stored in two 8-bit register (+127 to -128)

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

Detecting overflow condition by observing

- the carry into the sign bit position
- the carry out of the sign bit position

An overflow has occurred if two carries are not equal (V bit)

Overflow Detection

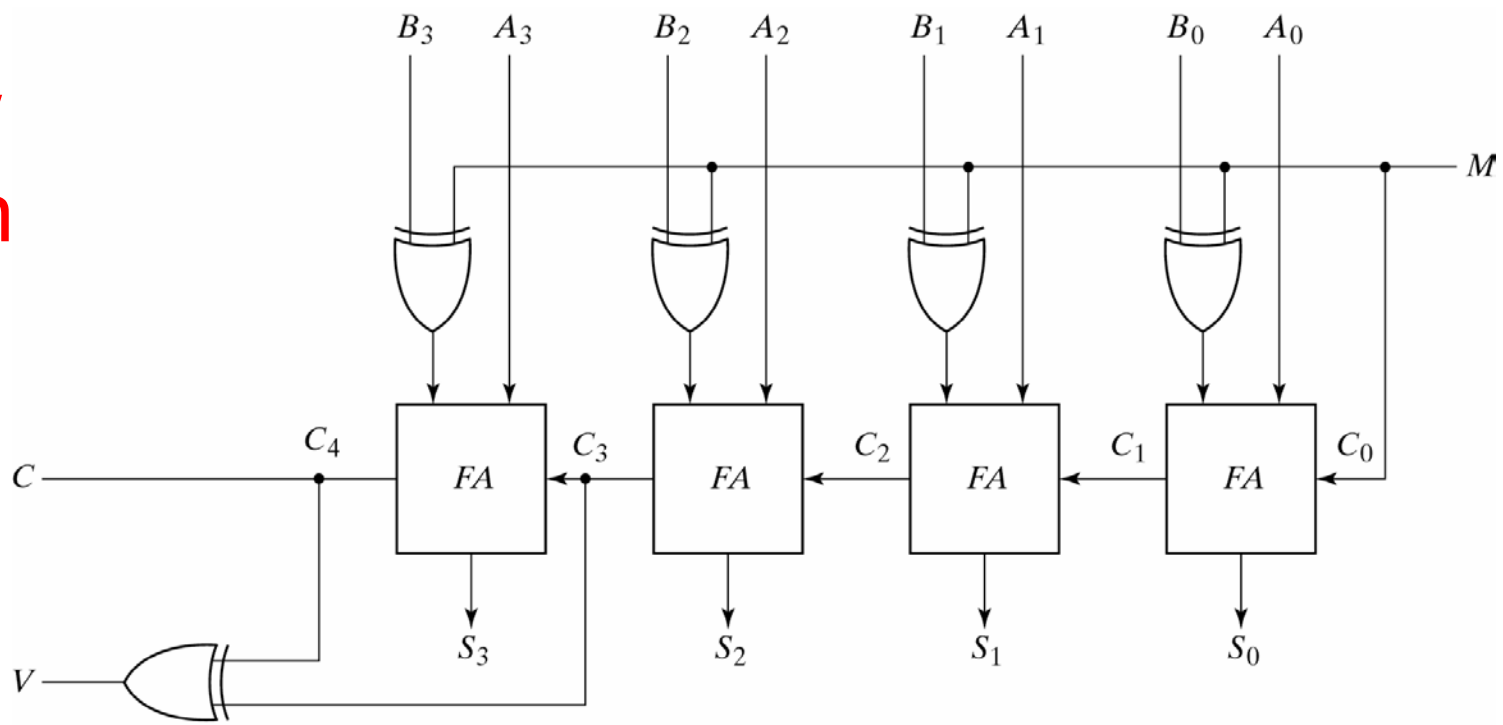


Fig. 4-13 4-Bit Adder Subtractor

- Overflow for unsigned numbers

  C bit detects a carry after addition or a borrow after subtraction

- Overflow for signed numbers

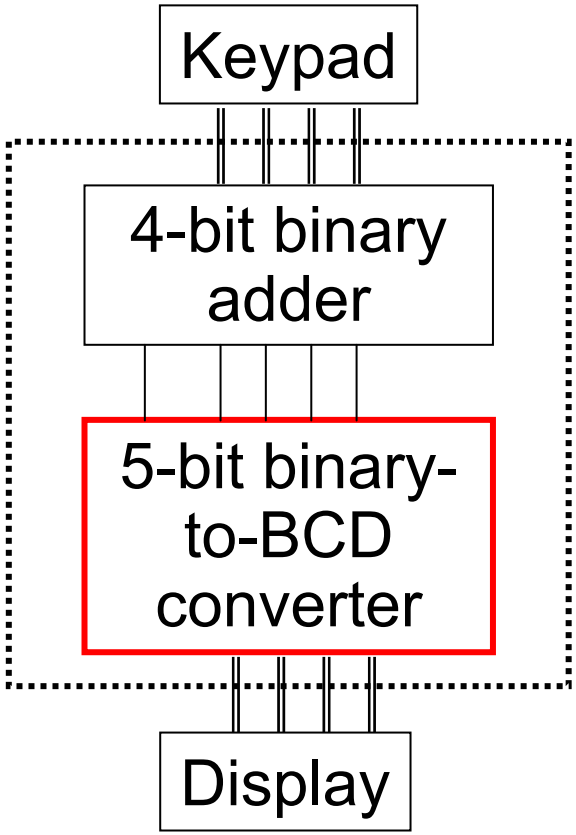
  V=0 after an addition or subtraction: indicate no overflow

  V=1: the result of the operation contains n+1 bits

  - An overflow has occurred
  - The (n+1)th bit is the actual sign and has been shifted out of position

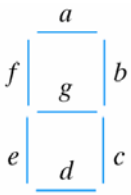
4-5 Decimal Adder

- Binary adder: $(1000)_2 + (1001)_2 = (1\ 0001)_2$
- BCD adder: $(1000)_{BCD} + (1001)_{BCD} = (1\ 0111)_{BCD}$
- Decimal adder: 8 + 9 = 17



BCD / Decimal Adder

0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001



(a) Segment designation

(b) Numerical designation for display

BCD Adder

Table 4-5
Derivation of BCD Adder

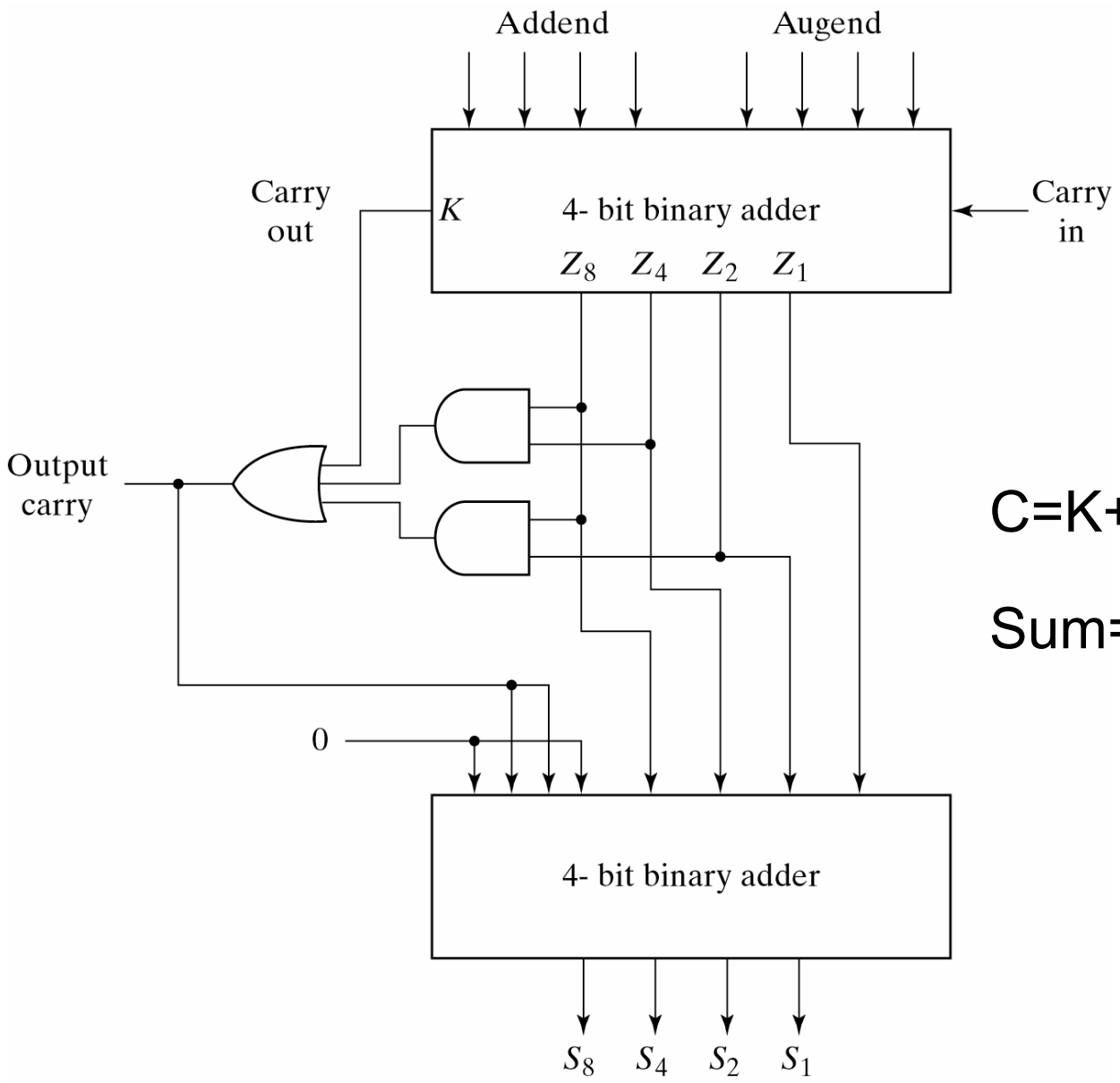
Binary Sum						BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1		C	S_8	S_4	S_2	S_1	
0	0	0	0	0		0	0	0	0	0	0
0	0	0	0	1		0	0	0	0	1	1
0	0	0	1	0		0	0	0	1	0	2
0	0	0	1	1	=	0	0	0	1	1	3
0	0	1	0	0		0	0	1	0	0	4
0	0	1	0	1	+ 0	0	0	1	0	1	5
0	0	1	1	0	C=0	0	0	1	1	0	6
0	0	1	1	1		0	0	1	1	1	7
0	1	0	0	0		0	1	0	0	0	8
0	1	0	0	1		0	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	+0110	1	0	1	0	0	14
0	1	1	1	1	C=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

Convert 5 bits into 2 BCD digits

$C=K+Z_8Z_4+Z_8Z_2$

$Sum=KZ_8Z_4Z_2Z_1+00CC0$

Figure 4-14 Block diagram of a BCD Adder



$C=K+Z_8Z_4+Z_8Z_2$

$Sum=KZ_8Z_4Z_2Z_1+00CC0$

Fig. 4-14 Block Diagram of a BCD Adder

4-6 Binary Multiplier

Multiplication of binary numbers is performed in the same way as in decimal numbers

– partial product: the multiplicand is multiplied by each bit of the multiplier starting from the least significant bit

```
        2     8
*       1     3
-------------------
        6 + 2 4
2       8
-------------------
3       6     4
```

```
                1 1 0
    *         1 1 0 1
-----------------------
                1 1 0
              0 0 0
          1 1 0
      1 1 0
-----------------------
1 0 0 1 1 1 0
```

```
        6
*     1 3
-----------
      1 8
    6
-----------
    7 8
```

Multiplication of two bits = A * B (AND)

0 * 0 = 0 0 * 1 = 0 1 * 0 = 0 1 * 1 = 1

2-bit by 2-bit binary multiplier

		B_1	B_0
		A_1	A_0
		A_0B_1	A_0B_0
	A_1B_1	A_1B_0	
C_3	C_2	C_1	C_0

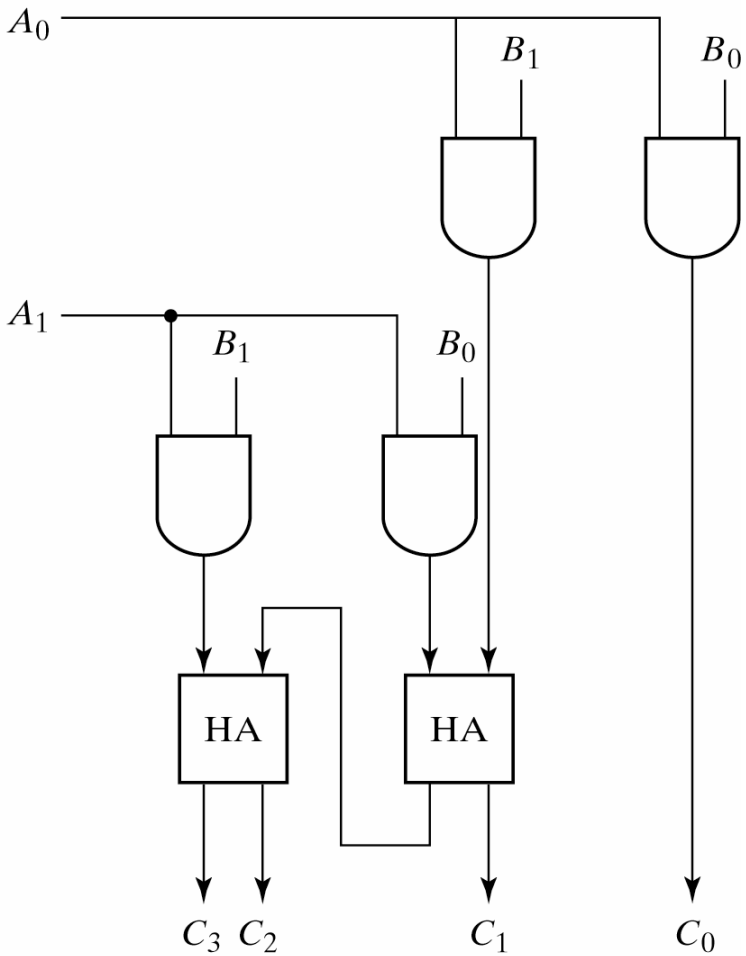


Fig. 4-15 2-Bit by 2-Bit Binary Multiplier

4-bit by 3-bit binary multiplier

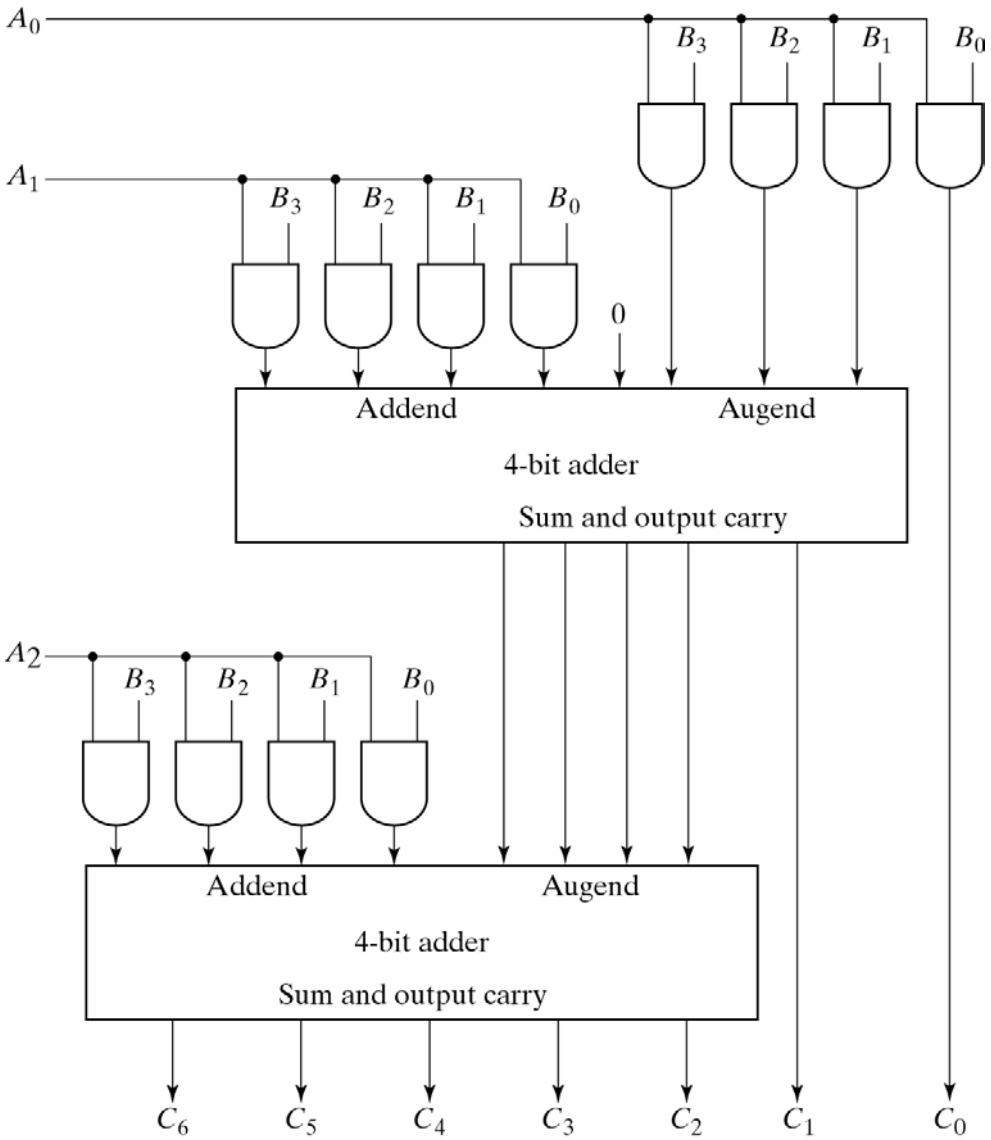


Fig. 4-16 4-Bit by 3-Bit Binary Multiplier

			B_3	B_2	B_1	B_0				
			1	1	0	1			1	3
	*			1	1	0		*		6
			0	0	0	0	A_0		1	8
+		1	1	0	1		A_1		6	
		1	1	0	1	0			7	8
+	1	1	0	1			A_2			
1	0	0	1	1	1	0				

4-7 Magnitude Comparator

A combinational circuit that compares two numbers, A and B, and determines their relative magnitudes: A>B, A=B, or A<B

4-bit comparator: $A=A_3A_2A_1A_0$, $B=B_3B_2B_1B_0$

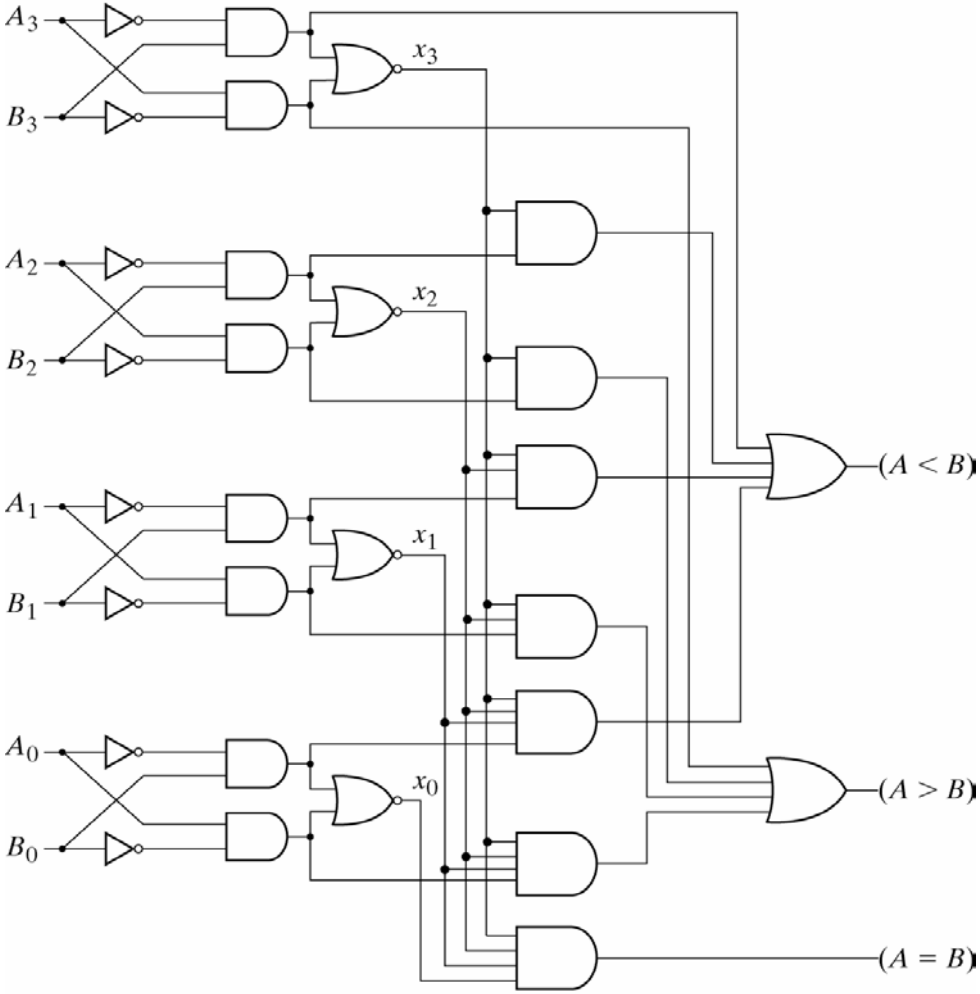


Fig. 4-17 4-Bit Magnitude Comparator

(A=B): $A_3=B_3$, $A_2=B_2$, $A_1=B_1$, $A_0=B_0$

$x_i = A_iB_i + A_i'B_i'$

XOR-Invert $= (A_iB_i'+A_i'B_i)'$

$= (A_i'+B_i)(A_i+B_i')$

$= A_i'A_i + A_i'B_i' + A_iB_i + B_iB_i'$

$= A_iB_i + A_i'B_i'$

(A>B) $= A_3B_3' + x_3A_2B_2' + x_3x_2A_1B_1' + x_3x_2x_1A_0B_0'$

(A<B) $= A_3'B_3 + x_3A_2'B_2 + x_3x_2A_1'B_1 + x_3x_2x_1A_0'B_0$

Using a binary subtractor:
A-B>0, A-B=0, or A-B<0

4-8 Decoders

A combinational circuit that converts binary information from n input lines to a maximum of 2^n unique output lines

n-to-m-line decoders: generate m (=2^n or fewer) minterms of n input variables

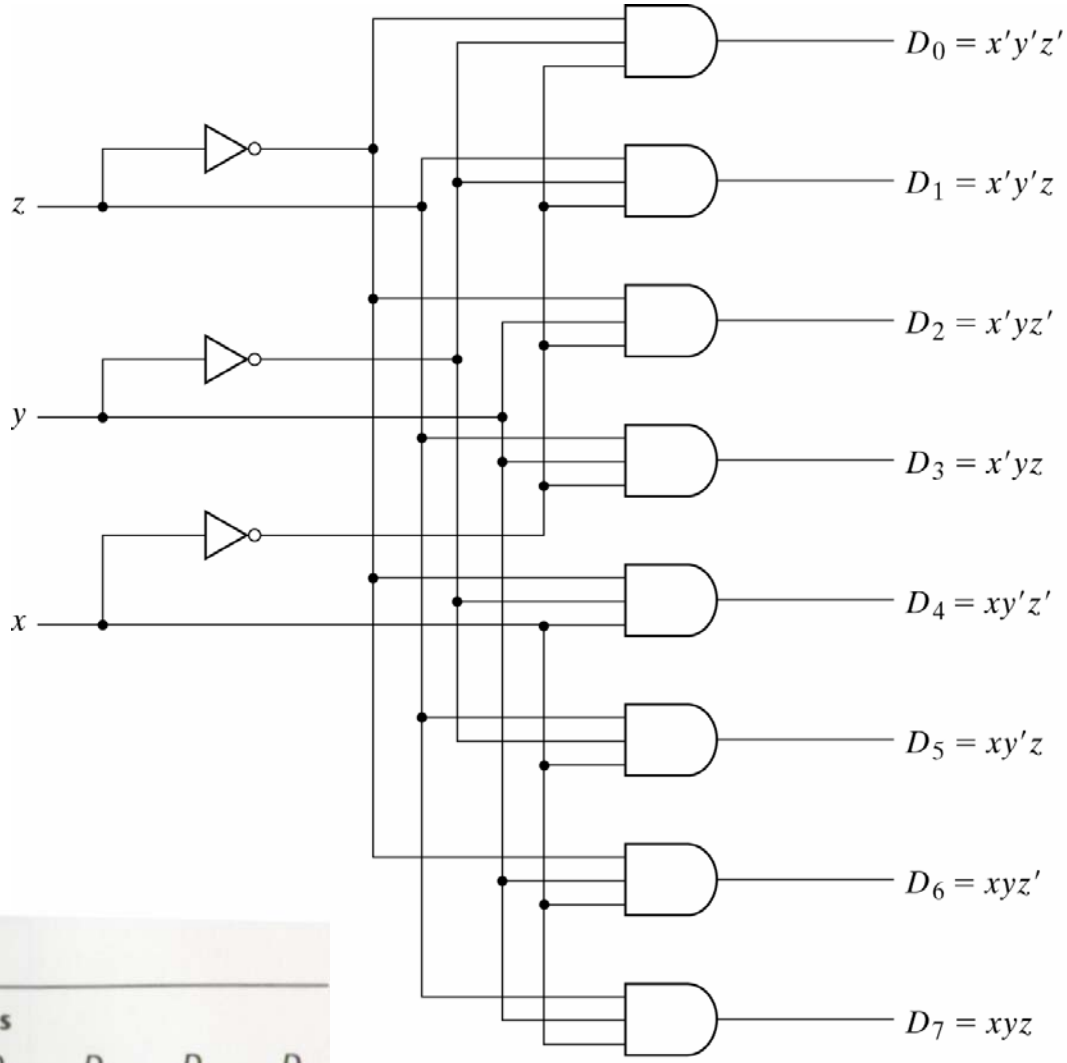


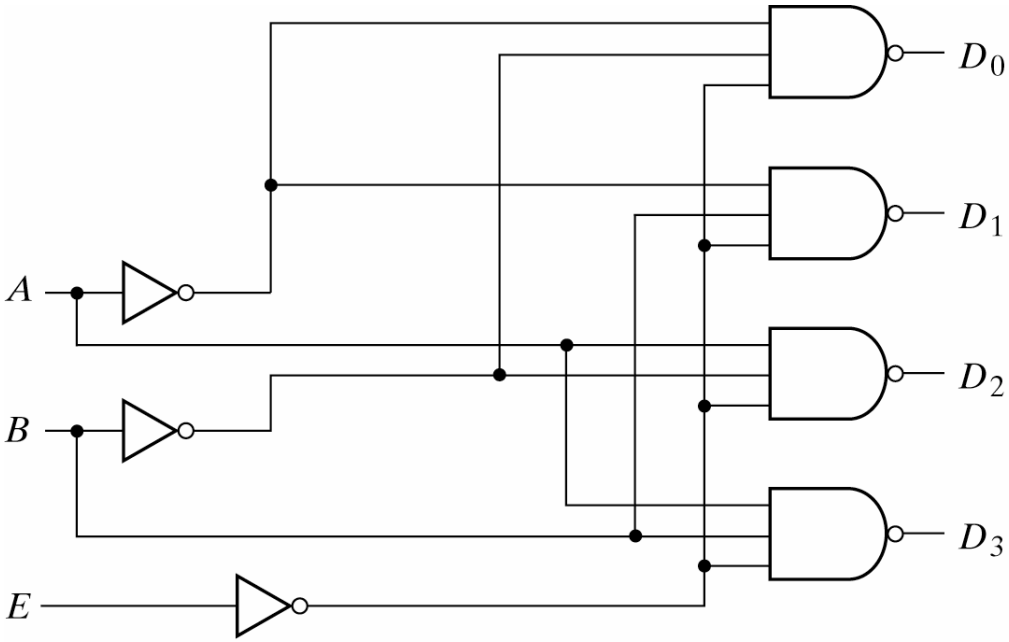
Fig. 4-18 3-to-8-Line Decoder

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 using NAND

- More economical to generate the decoder minterms in their complemented form using NAND gates
- Support one or more enable inputs to control the circuit operation
  - e.g. disabled when E is equal to 1 (no outputs)
  - same as a **Demultiplexer:** a circuit that receives information from a single line and directs it to one of 2^n possible output lines
    - Selection of a specific output is controlled by bit combination of n selection lines



(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

Connecting multiple decoders to form a larger one

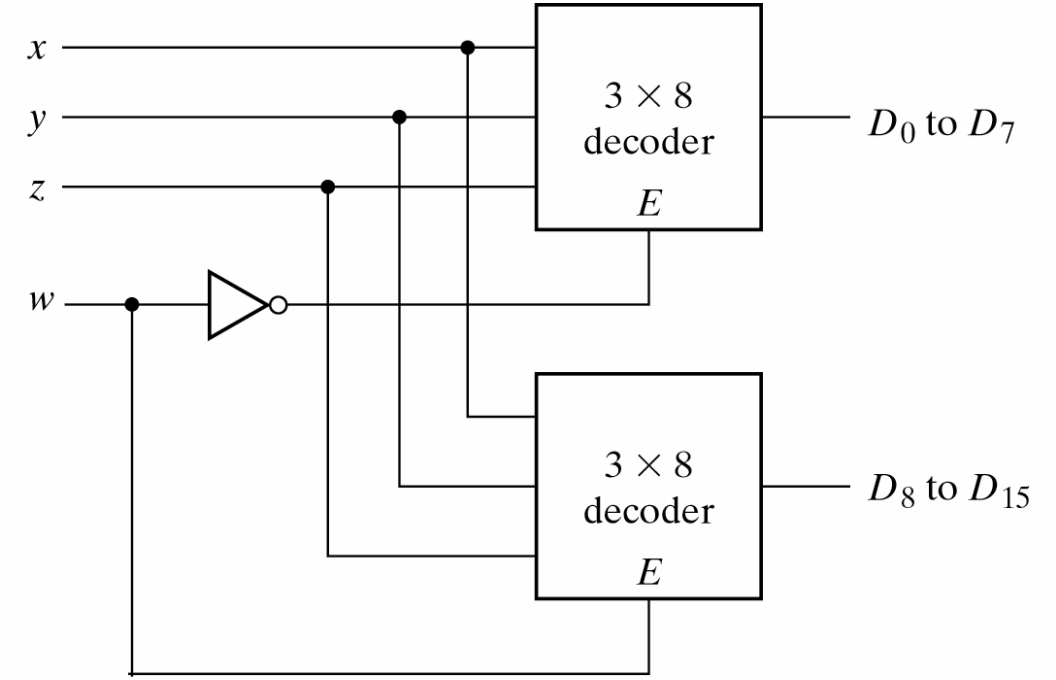


Fig. 4-20 4 × 16 Decoder Constructed with Two 3 × 8 Decoders

wxyz	D_0D_1*****D_7	D_8******D_{15}
0000	0111 1111	1111 1111
0001	1011 1111	1111 1111
0010	1101 1111	1111 1111
0011	1110 1111	1111 1111
0100	1111 0111	1111 1111
0101	1111 1011	1111 1111
0110	1111 1101	1111 1111
0111	1111 1110	1111 1111
1000	1111 1111	0111 1111
1001	1111 1111	1011 1111
1010	1111 1111	1101 1111
1011	1111 1111	1110 1111
1100	1111 1111	1111 0111
1101	1111 1111	1111 1011
1110	1111 1111	1111 1101
1111	1111 1111	1111 1110

w=0: the top decoder is enabled (0000 … 0111: D_0 to D_7)
w=1: the bottom decoder is enabled (1000 … 1111:D8 to D_{15})

- A decoder provides the 2^n minterm of n input variable
- Any combinational circuit with n inputs and m outputs can be implemented with an n-to-2^n-line decoder and m OR gates

Combinational Logic Implementation – Full Adder with a Decoder

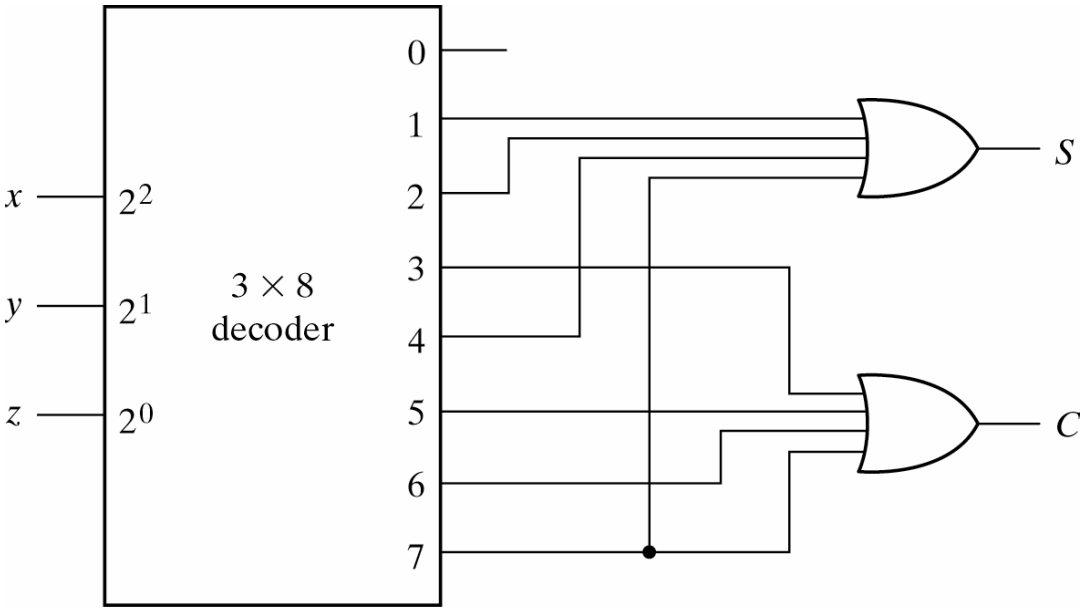


Fig. 4-21 Implementation of a Full Adder with a Decoder

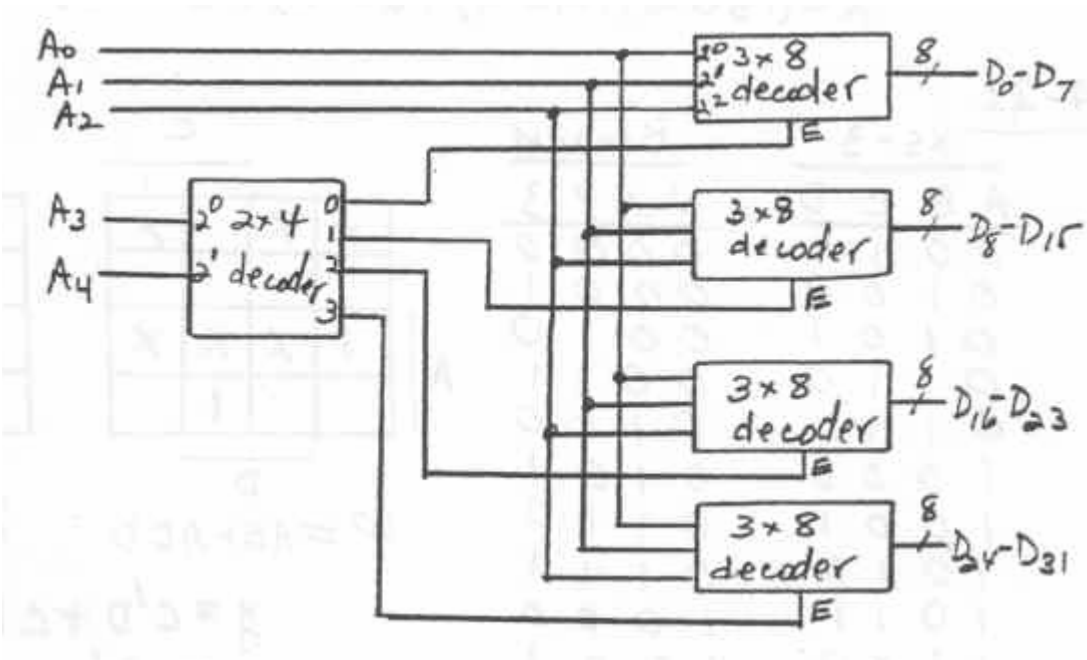
Table 4-4
Full Adder

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

$$S(x, y, z) = \Sigma(1, 2, 4, 7)$$
$$C(x, y, z) = \Sigma(3, 5, 6, 7)$$

Large input with NOR: if the number of minterms is greater than $2^n/2$, then F' can be expressed with fewer minterms

Problem 4-25 Construct a 5-to-32-line decoder with four 3-to-8-line decoders with enable and a 2-to-4-line decoder. Use block diagrams for the components



4-9 Encoders

An encoder has 2^n (or fewer) input lines and n output lines, which generate the binary code corresponding to the input value

Octal-to-binary encoder: 8 inputs (one for each of the octal digits) and three outputs generating the corresponding binary number

- multiple inputs: undefined => priority encoder
- Input with all 0's = D_0 is equal to 1

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

Table 4-7
Truth Table of Octal-to-Binary Encoder

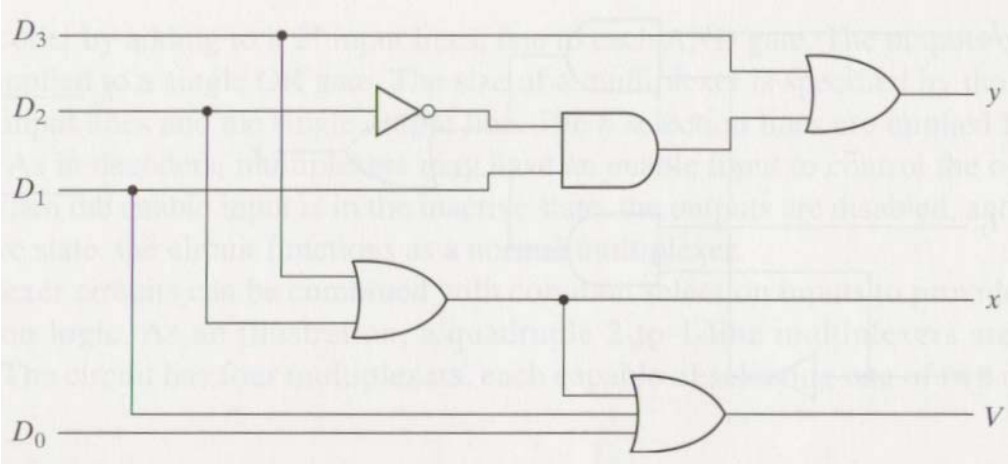
Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

Priority Encoder

An encoder circuit that includes the priority function: if two or more inputs are equal to 1 at the same time, the input having the highest priority will take precedence

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



X in inputs: condensed form, either 0 or 1
X in outputs: don't-care

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

v: valid bit indicator that is set to 1 when one or more inputs are equal to 1

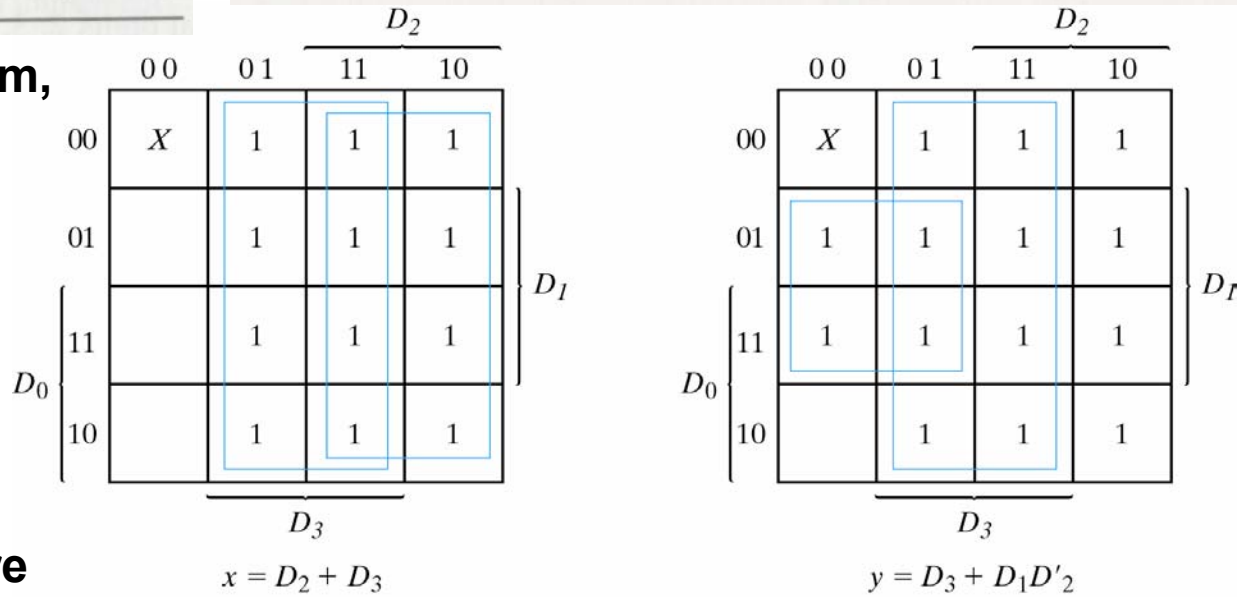


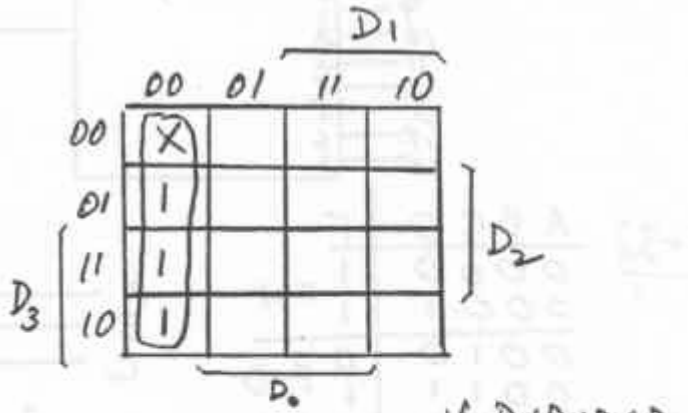
Fig. 4-22 Maps for a Priority Encoder

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

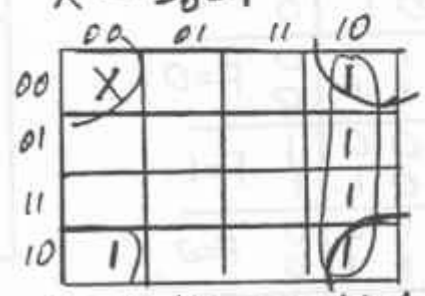
Problem 4-29 Design a 4-input priority encoder with inputs as in Table 4-8, but with inputs D_0 having the highest priority and input D_3 the lowest priority

Inputs				outputs		
D_3	D_2	D_1	D_0	x	y	V
0	0	0	0	x	x	0
x	x	x	1	0	0	1
x	x	1	0	0	1	1
x	1	0	0	1	0	1
1	0	0	0	1	1	1

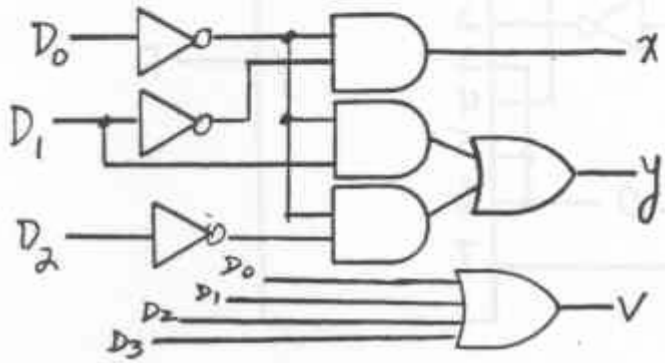


$V = D_0 + D_1 + D_2 + D_3$

$X = D_0' D_1'$



$Y = D_0' D_1 + D_0' D_2'$

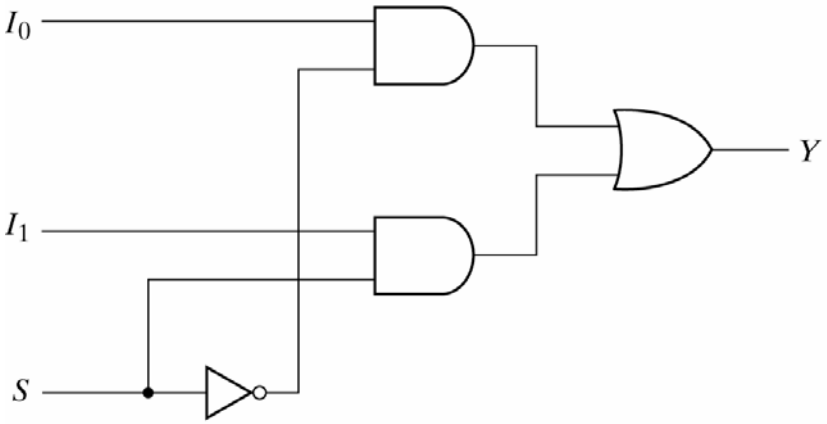


4-10 Multiplexers

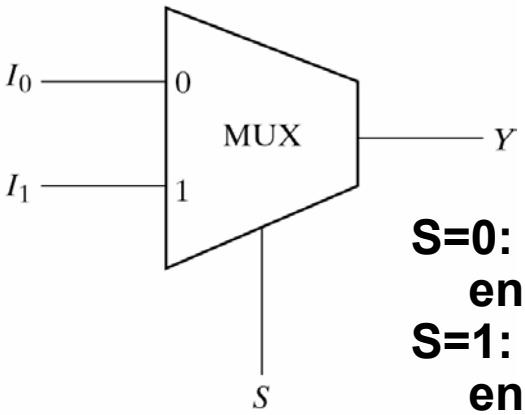
A combinational circuit that selects binary information from one of many input lines and directs it to a single output line

- Normally, there are 2^n input lines and n selection lines whose bit combinations determine which input is selected
- also called a ***data selector***

2-to-1-line multiplexer: two data input lines, one output line, and one selection line S



(a) Logic diagram

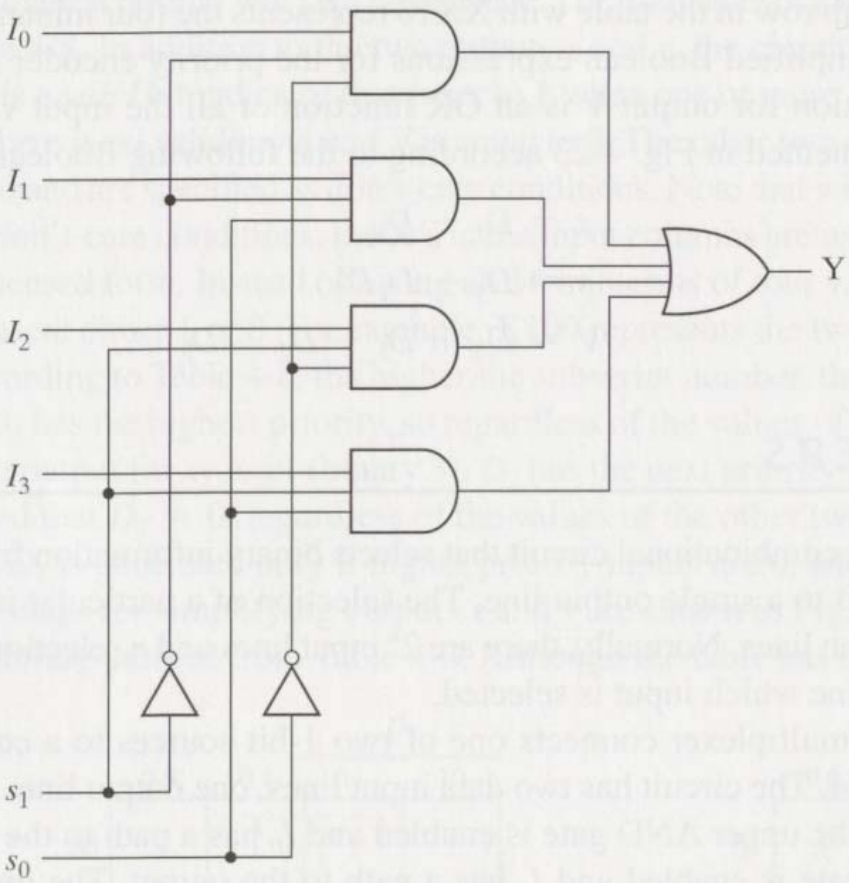


(b) Block diagram

S=0: the upper AND is enabled and Y = I_0
S=1: the lower AND is enabled and Y = I_1

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

4-to-1-line multiplexer



(a) Logic diagram

FIGURE 4-25
4-to-1-Line Multiplexer

Four inputs: I_0 through I_3
Selection lines S_1 and S_0

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

(b) Function table

Quadruple 2-to-1-line multiplexer with enable input

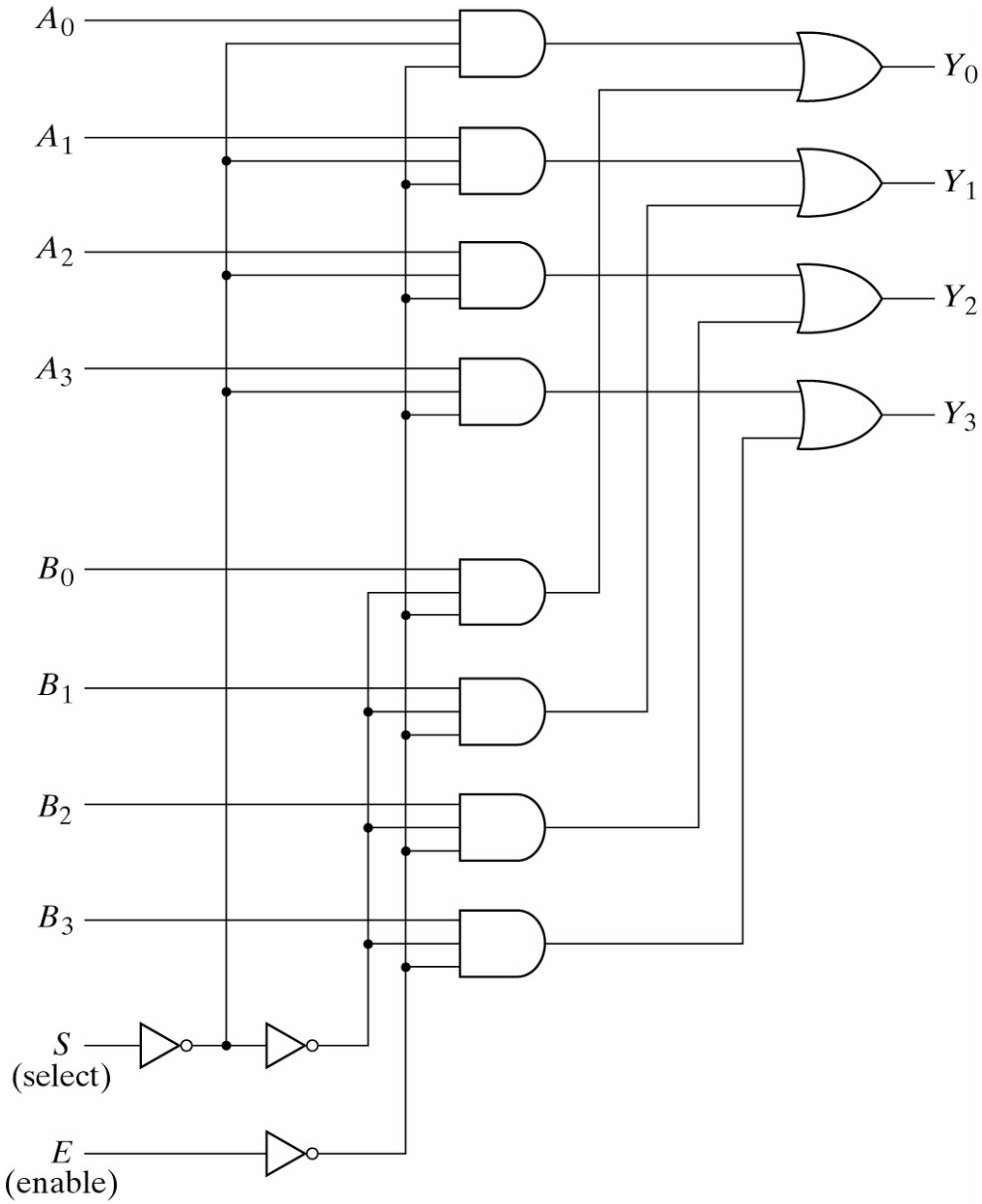


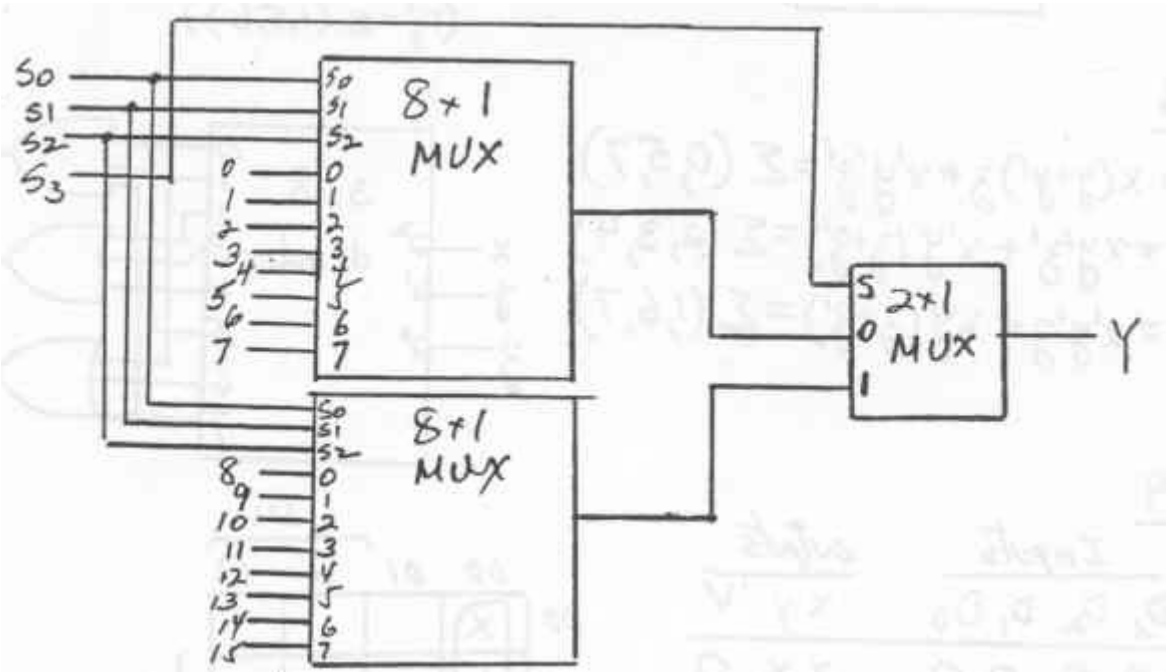
Fig. 4-26 Quadruple 2-to-1-Line Multiplexer

- Select one of two 4-bit sets of data lines
- 4 multiplexers, each capable of selecting one of two input lines
- Selection line S
- Enable line E

Function table

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

Problem 4-31 Construct a 16x1 multiplexer with two 8x1 and one 2x1 multiplexers. Use block diagrams



Boolean Function Implementation

Implementing a Boolean function of n variables with a multiplexer that has n-1 selection lines (2^{n-1} inputs)

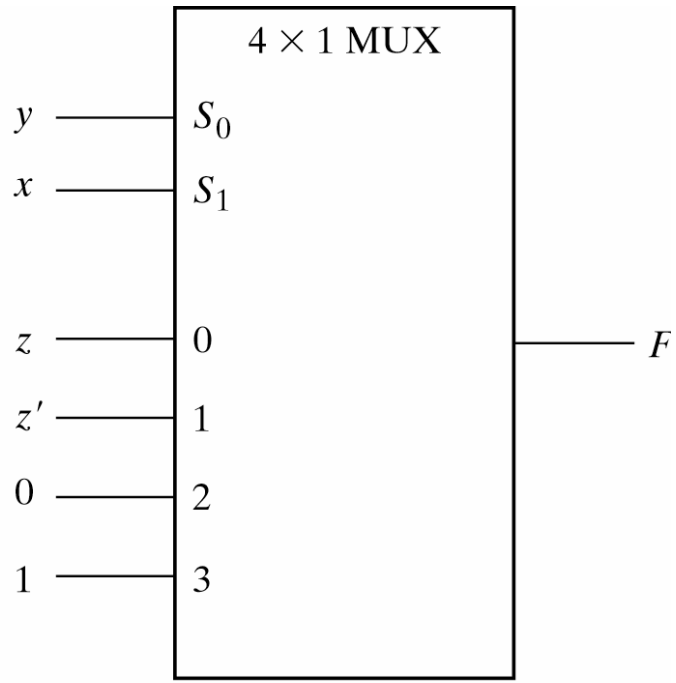
- the first n-1 variables are connected to the selection inputs
- the remaining single variable is used for the data input: z, z', 1, or 0

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

- x, y: selection inputs S_1 and S_0

x	y	z	F	
0	0	0	0	$F = z$
0	0	1	1	
0	1	0	1	$F = z'$
0	1	1	0	
1	0	0	0	$F = 0$
1	0	1	0	
1	1	0	1	$F = 1$
1	1	1	1	

(a) Truth table



(b) Multiplexer implementation

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

Implementing with a multiplexer of n selection lines

Implementing a 4-input function with a multiplexer

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

1. List the truth table of F
2. Evaluate the output as a function of the last variable: 0, 1, the variable, or the complement of the variable
3. Apply the first n-1 variables to the selection inputs
4. Connect 0, 1, the variable and the complement of the variable to the data inputs according to the results of step 2

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	$F = D'$
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	

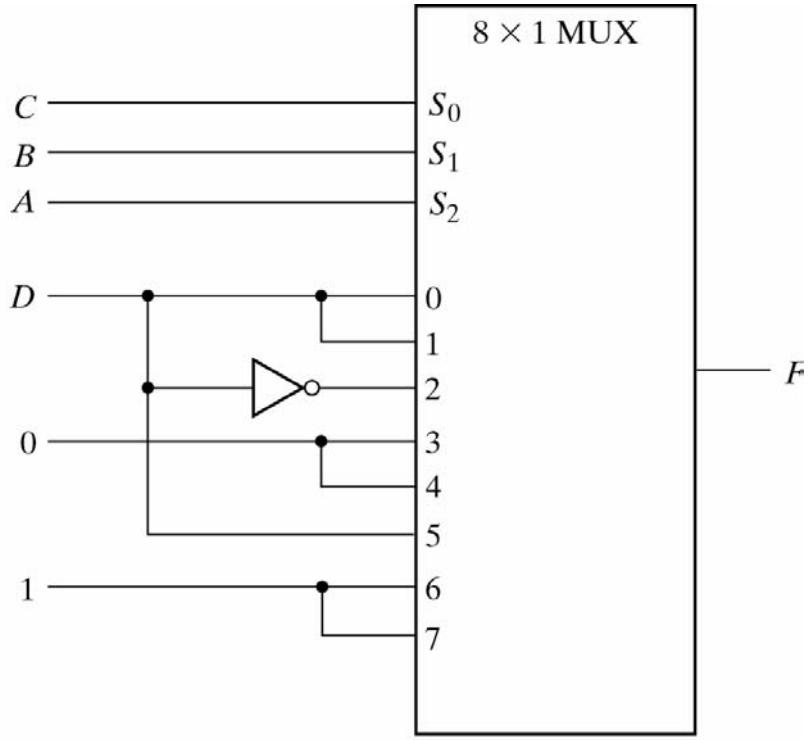
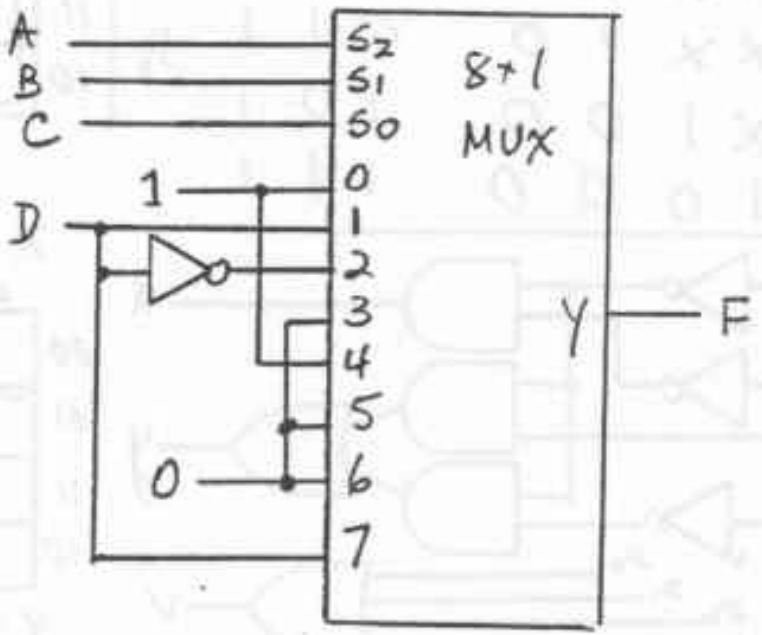


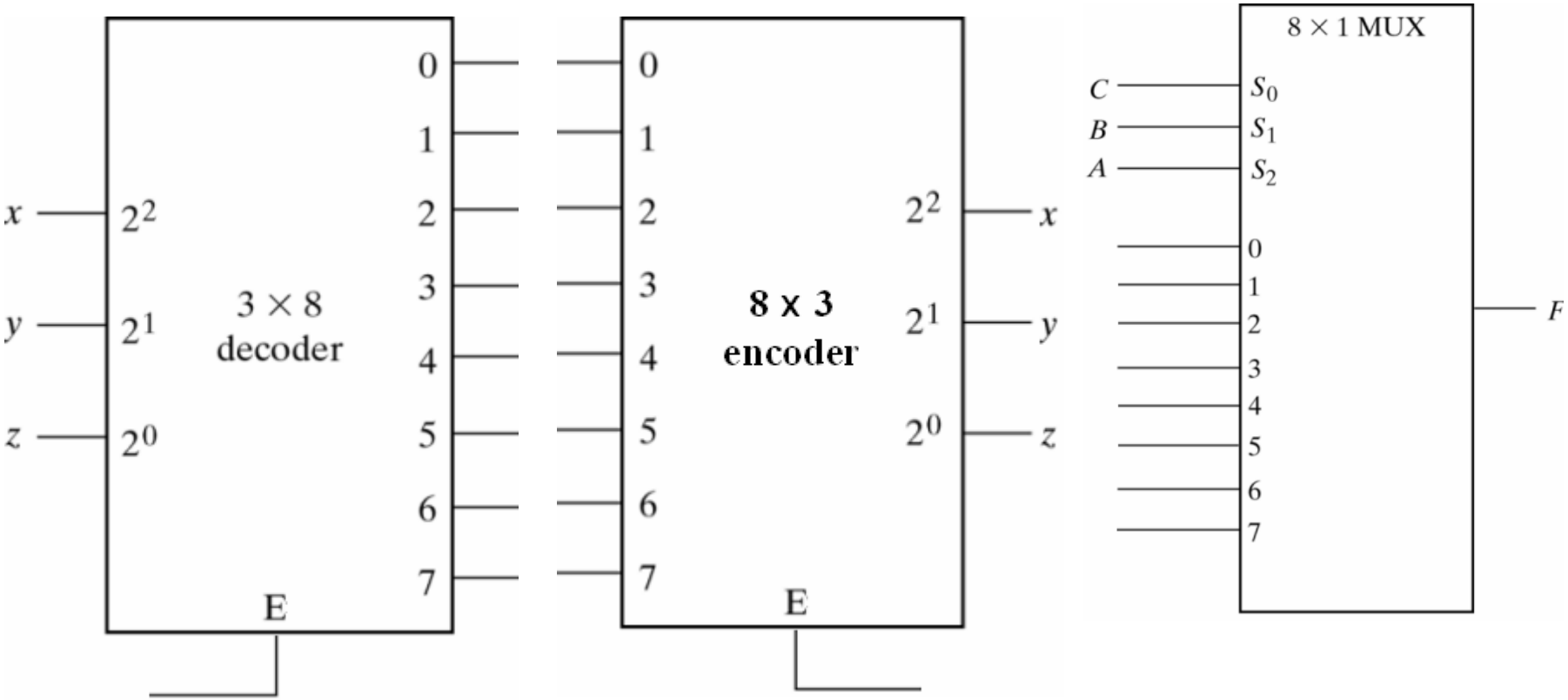
Fig. 4-28 Implementing a 4-Input Function with a Multiplexer

Problem 4-32 Implement the following Boolean function with a multiplexer: $F(A, B, C, D) = \Sigma(0, 1, 3, 4, 8, 9, 15)$

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



Decoder (Demultiplexer), Encoder and Multiplexer



- Enable input
- Priority encoder
- Construction of a large circuit with small ones
- Boolean function implementation

Decoder (Demultiplexer), Encoder and Multiplexer

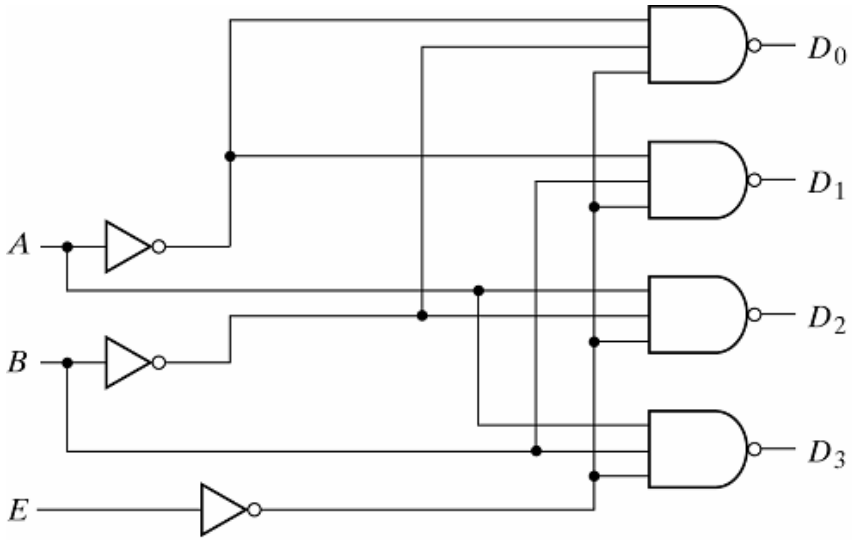


Fig. 4-19 2-to-4-Line Decoder with Enable Input

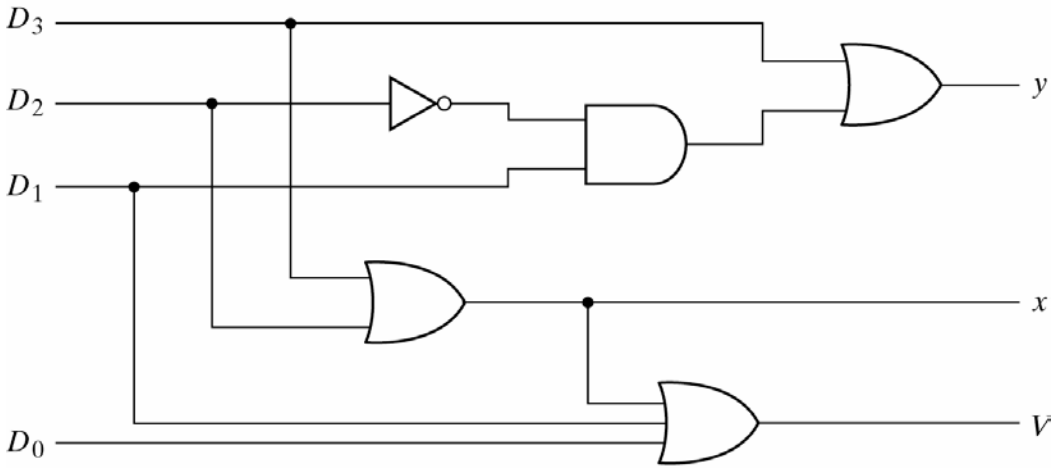


Fig. 4-23 4-Input Priority Encoder

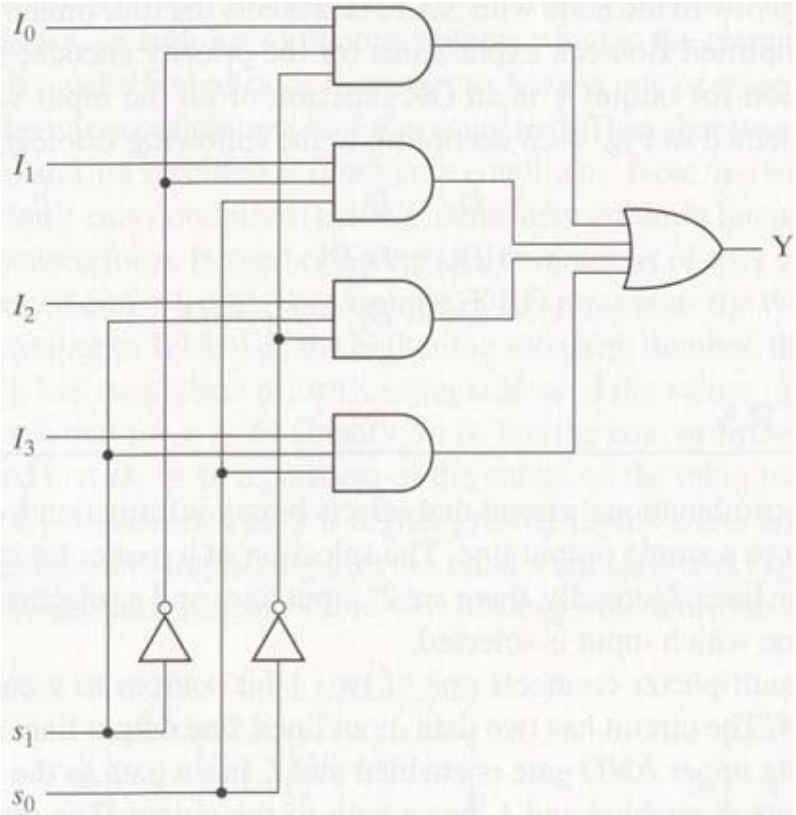


FIGURE 4-25 4-to-1-Line Multiplexer

Three-State Gates

- A digital circuit that exhibits three states
  - Two of the states are signal equivalent to logic 1 and 0
  - The third state is a high-impedance state, which behaves like a disconnected open circuit
  - May perform as AND, NAND, buffer, etc
- A multiplexer can be constructed with three-state gates

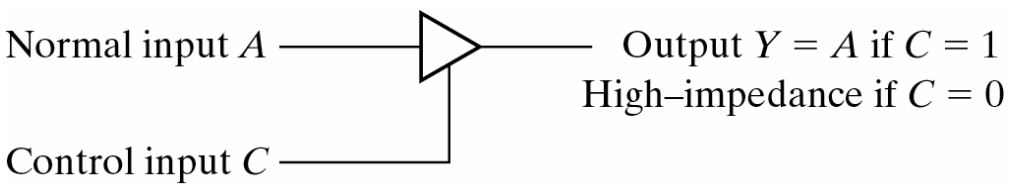
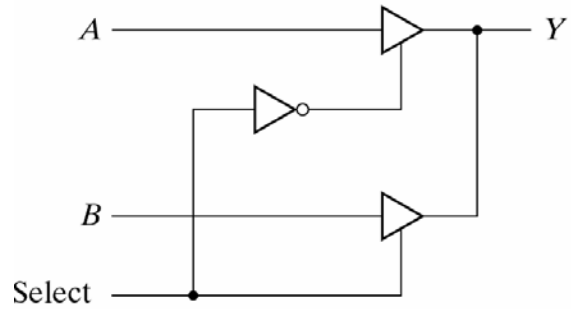
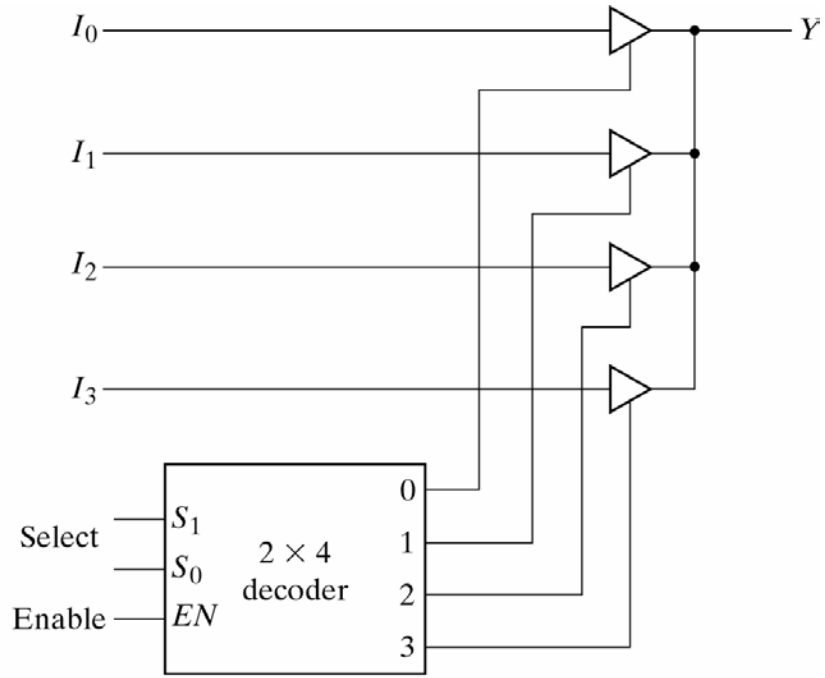


Fig. 4-29 Graphic Symbol for a Three-State Buffer



(a) 2-to-1- line mux



(b) 4 - to - 1 line mux

Fig. 4-30 Multiplexers with Three-State Gates

Summary

Chapter 4 Combinational Logic

4-1 Combinational Circuits

4-2 Analysis Procedure

4-3 Design Procedure

4-4 Binary Adder-Subtractor

- HA, FA, Binary Adder/subtractor
- propagation (carry lookahead generator), overflow

4-5 Decimal Adder

4-6 Binary Multiplier

4-7 Magnitude Comparator

4-8 Decoders

4-9 Encoders

4-10 Multiplexers

4-11 HDL For Combinational Circuits