Combinational Logic

Chapter 4

- Agenda
- Combinational Logic
  - Design Procedure
  - Adders, Subtractors
  - Analysis Procedure
  - Multilevel Logic Circuit
- Reading
- Mano: Ch 4
- Project #1

- Objectives:
- Understand the nature

of Combinational Logic

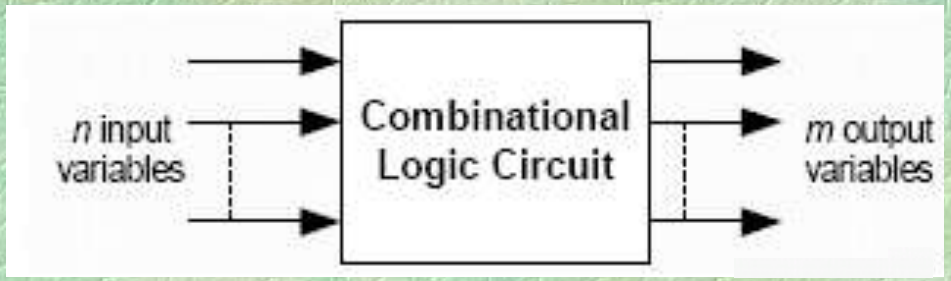
- Understand and able

to execute the combinational logic

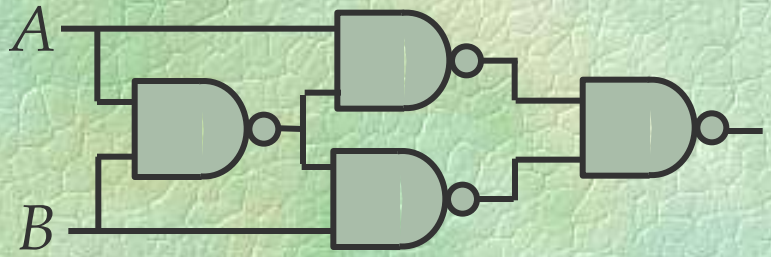
- design procedure

Combinational Logic : Definition

- Combinational Logic is a logical circuit consists of logic gates whose outputs at any time are determined directly from the present combination of inputs without regard to previous inputs



Combinational Circuit



Analysis Procedure:

F2=AB+AC+BC
T1=A+B+C
T2=ABC
and
T3=F2' T1
F1=T3+T2
Therefore,
F1=T3+T2= F2' T1 + ABC
=(AB+AC+BC)'(A+B+C)
+ ABC
=(A'+B')(A'+C')(B'+C')(A
+B+C)+ABC
=
=A'BC'+A'B'C+AB'C'+ABC

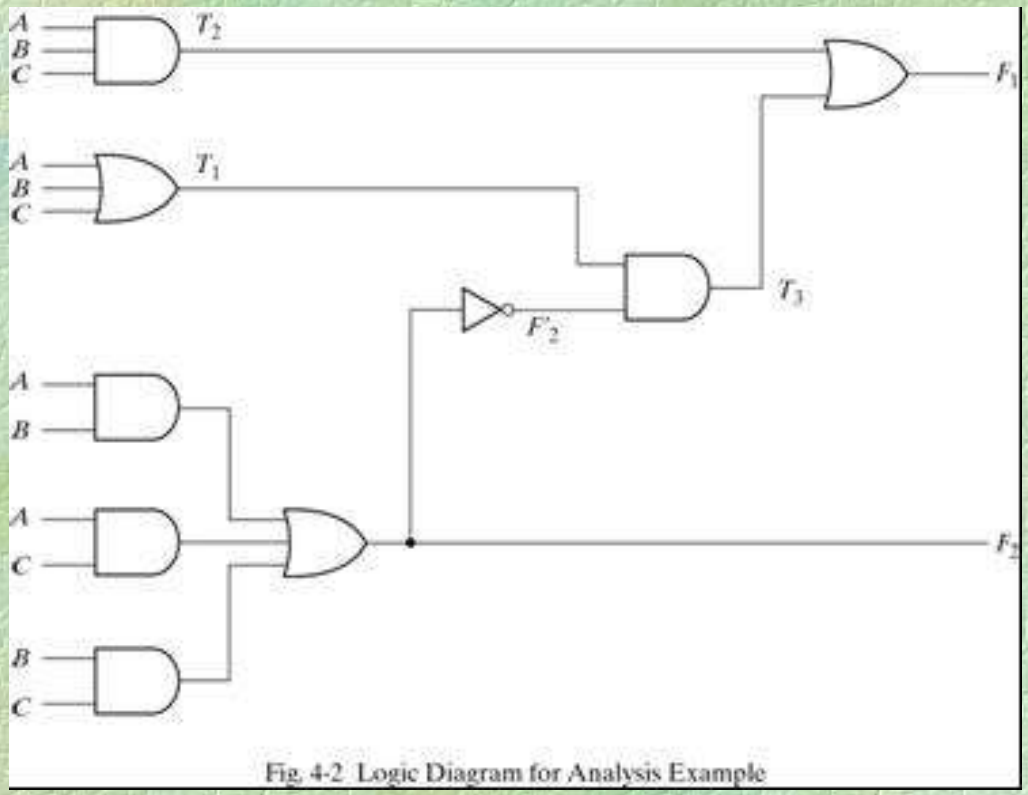


Fig. 4-2 Logic Diagram for Analysis Example

Design Procedure:

1. **Define the problem**
2. **Define the input/output variables**
3. **Truth table of the relationships**
4. **Simplify Boolean functions**
5. **Draw the logic diagram**

***Constraints*:**

1. Min No. of gates
2. Minimum number of inputs to a gate
3. Min Propagation time
4. Min no. of interconnections

Example: Code Conversion (BCD to Excess-3)

Input BCD					Output Excess-3			
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

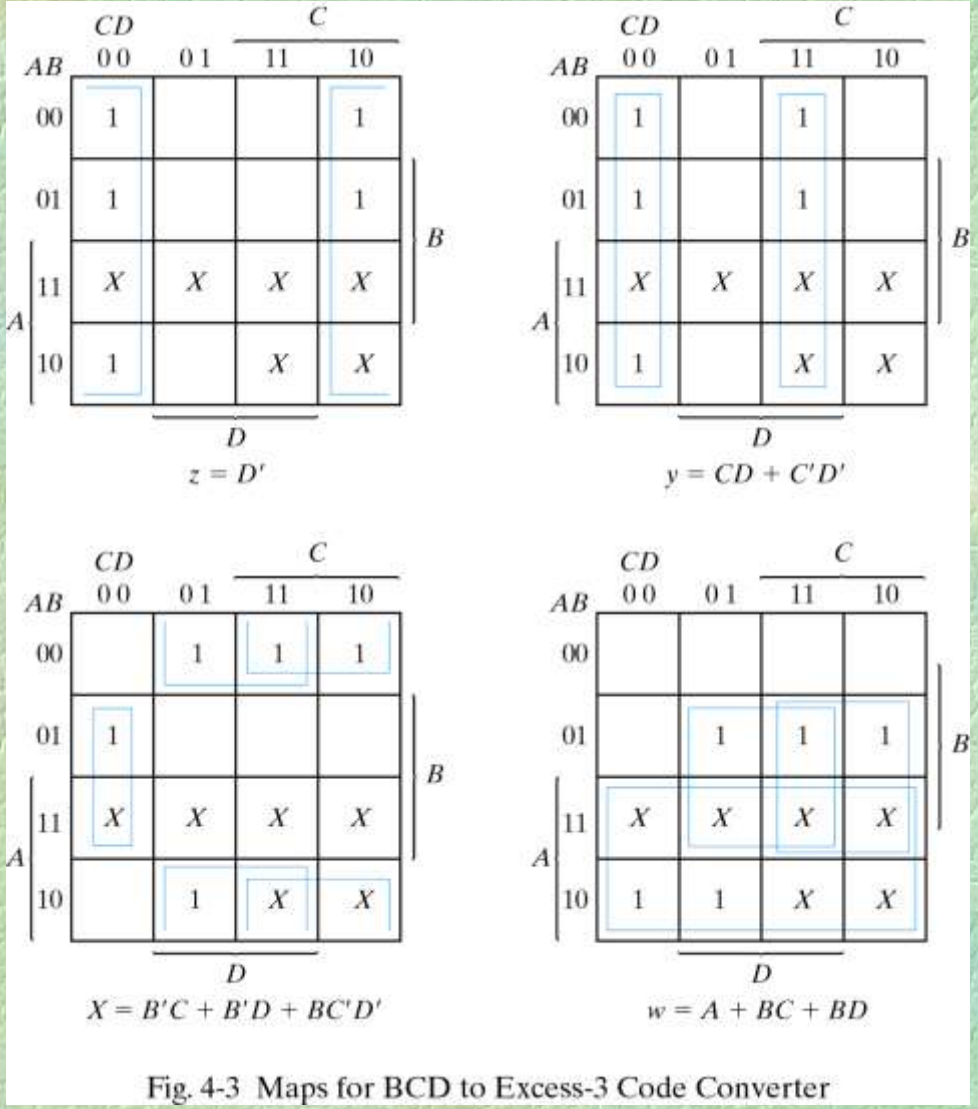


Fig. 4-3 Maps for BCD to Excess-3 Code Converter

Example: Code Conversion (BCD to Excess-3)

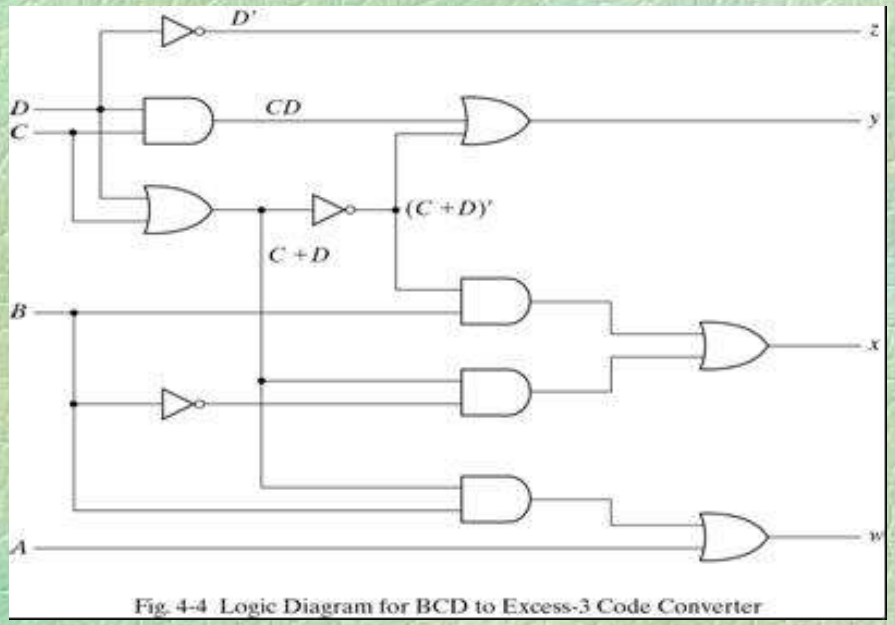


Fig. 4-4 Logic Diagram for BCD to Excess-3 Code Converter

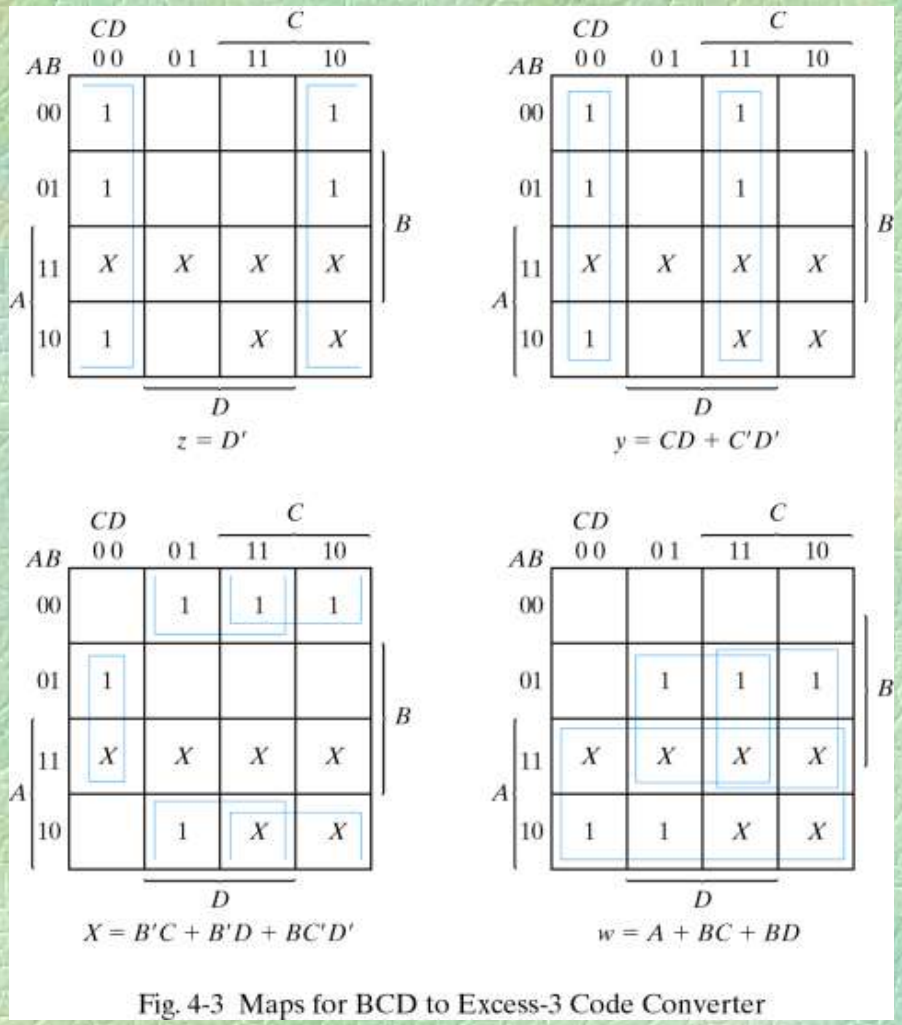


Fig. 4-3 Maps for BCD to Excess-3 Code Converter

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)

Common Combinational Logic

Binary Adders

- Half-Adders
- Full-Adders

Binary Substractors

- Half-Substractors
- Full-Substractors
- Decoders/Encoders
- Multiplexers

Binary Adders

- One of the basic arithmetic process in computer system
- One that performs the addition of 2 bits is Half-Adder
- One that performs the addition in 3 bits (2 significant bits and a previous carry) is called Full-Adder

Half-Adder

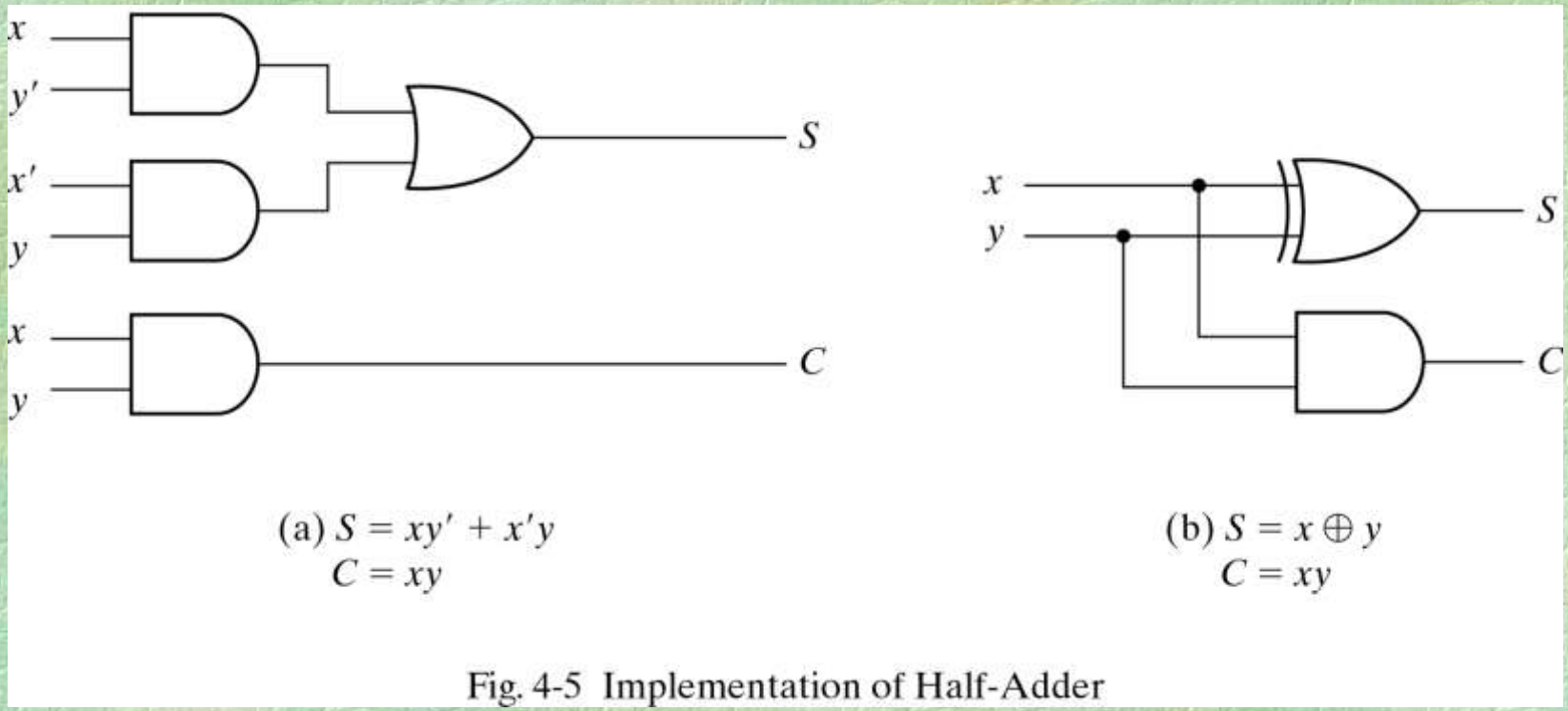
2 Input & 2 output

- The truth table
- Thus the Boolean Function is
- S = x'y+ xy';
- C = xy
- The function cannot be further simplified

Truth Table

X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Half-Adder Implementations



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

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

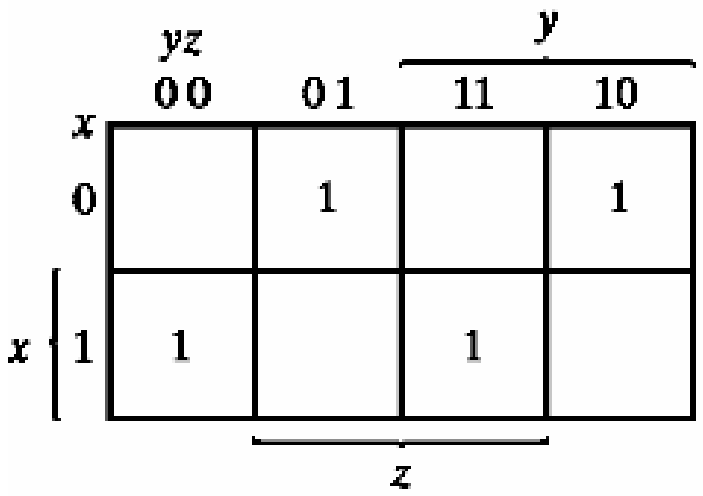
Fig. 4-5 Implementation of Half-Adder

Full-Adder Truth Table

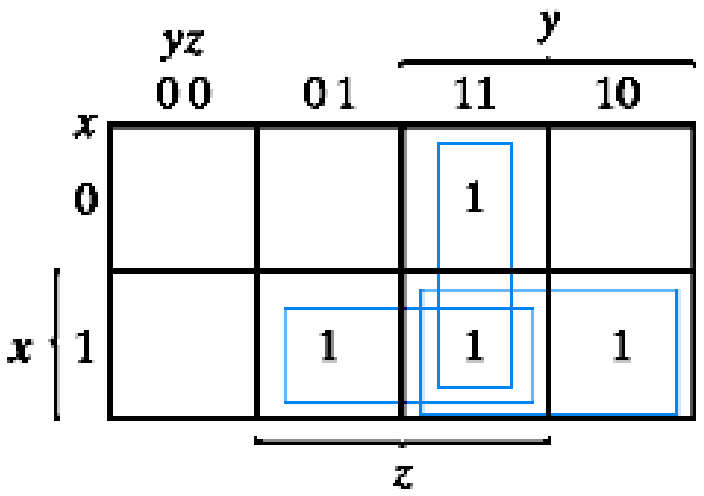
- 3 Input (x & y as the input and z as the previous carry), & 2 output (s, c)
- The truth table is :

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 Map



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



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

Fig. 4-6 Maps for Full Adder

Full-Adder Simple Implementation

- Logic Expression

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

C = xy + xz + yz

- **Implementation**

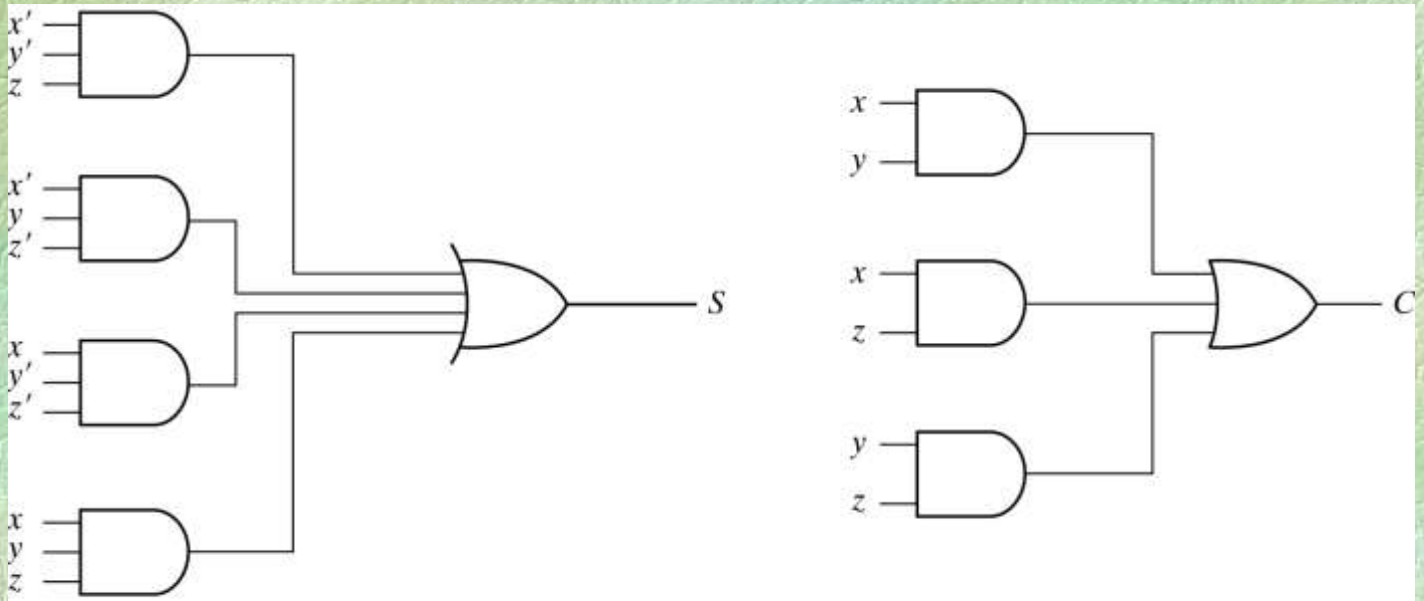


Fig. 4-7 Implementation of Full Adder in Sum of Products

Full-Adder Implementation

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

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

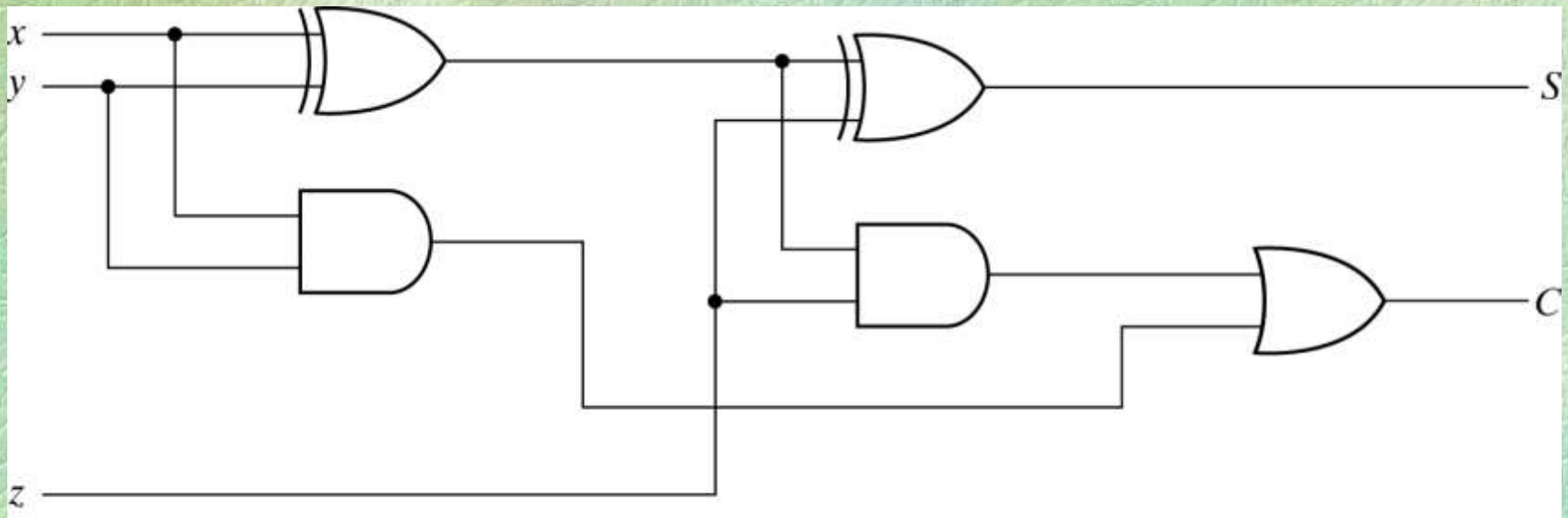


Fig. 4-8 Implementation of Full Adder with Two Half Adders and an OR Gate

Full-Adder Application

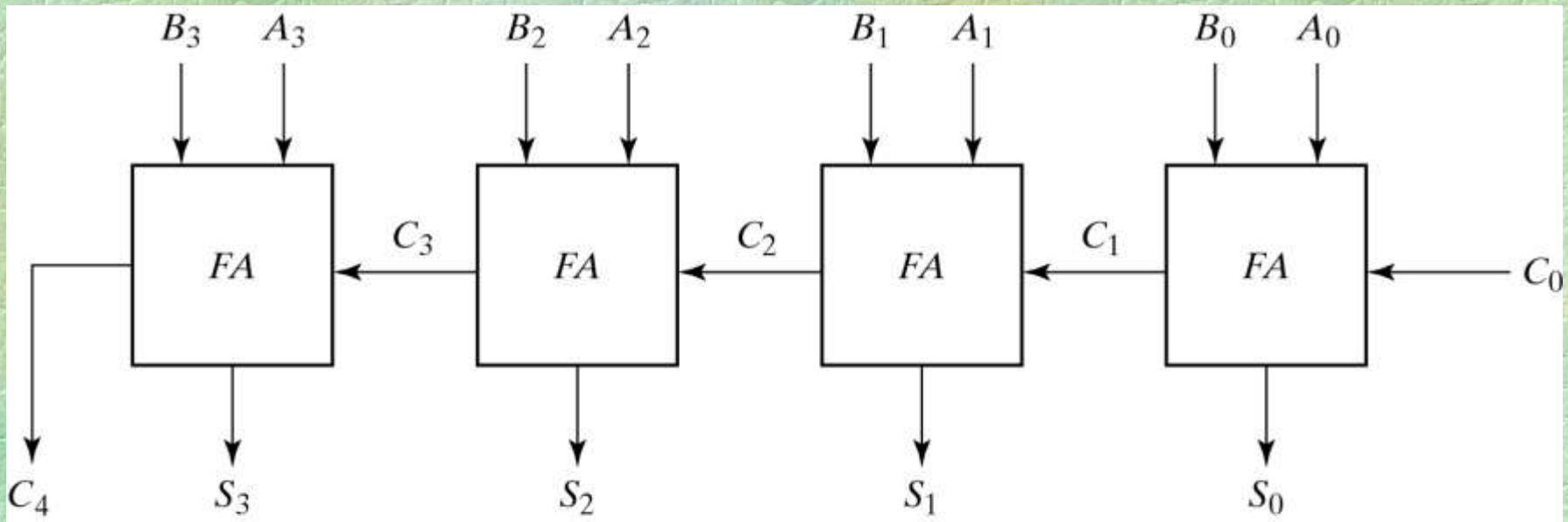


Fig. 4-9 4-Bit Adder

Example:

A=1011, B=0011 then S= 1110

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}

Full-Adder Other Implementation

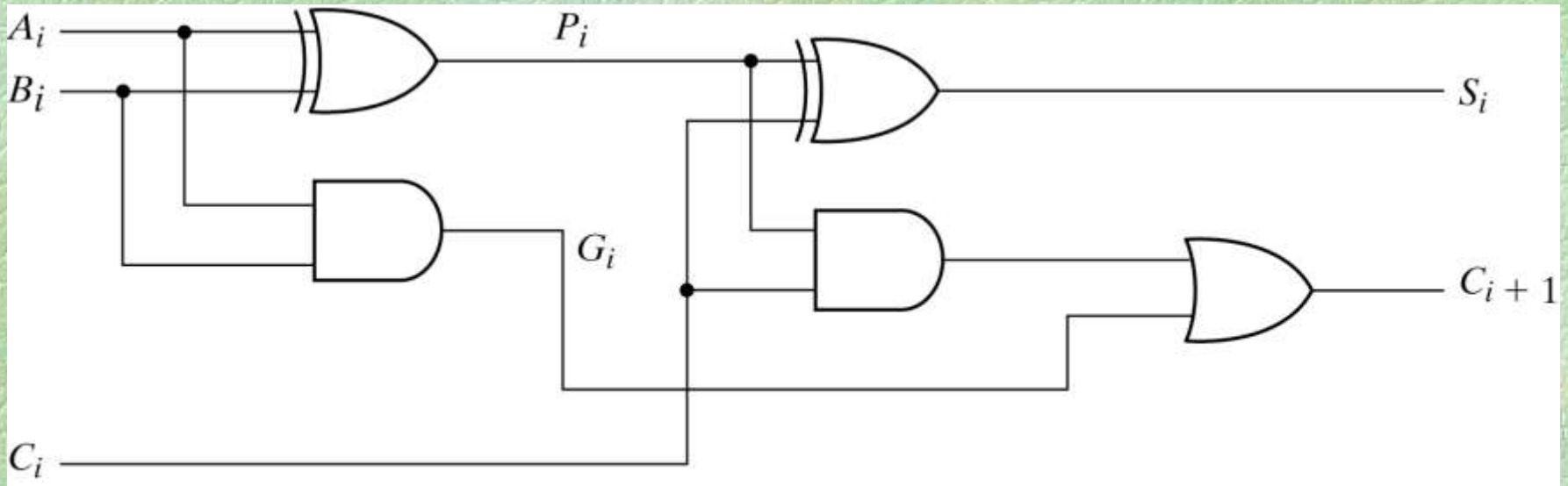


Fig. 4-10 Full Adder with P and G Shown

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

$G_i = A_i B_i$ $\quad$ $C_{i+1} = G_i + P_i C_i$

G_i: Carry generate

P_i: Carry propagate

Carry Propagation:

- *The total propagation time is equal to the propagation delay of a typical gate times the number of gate levels in the circuit.*

C_0=input carry

$C_1 = G_0 + P_0C_0$

$C_2 = G_1 + P_1C_1 = G_1 + P_1(G_0 + P_0C_0) = G_1 + P_1G_0 + P_1P_0C_0$

$C_3 = G_2 + P_2C_2 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$

Carry lockahead Generator

C_0=input carry

$C_1 = G_0 + P_0C_0$

$C_2 = G_1 + P_1C_1 = G_1 + P_1(G_0 + P_0C_0) = G_1 + P_1G_0 + P_1P_0C_0$

$C_3 = G_2 + P_2C_2 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$

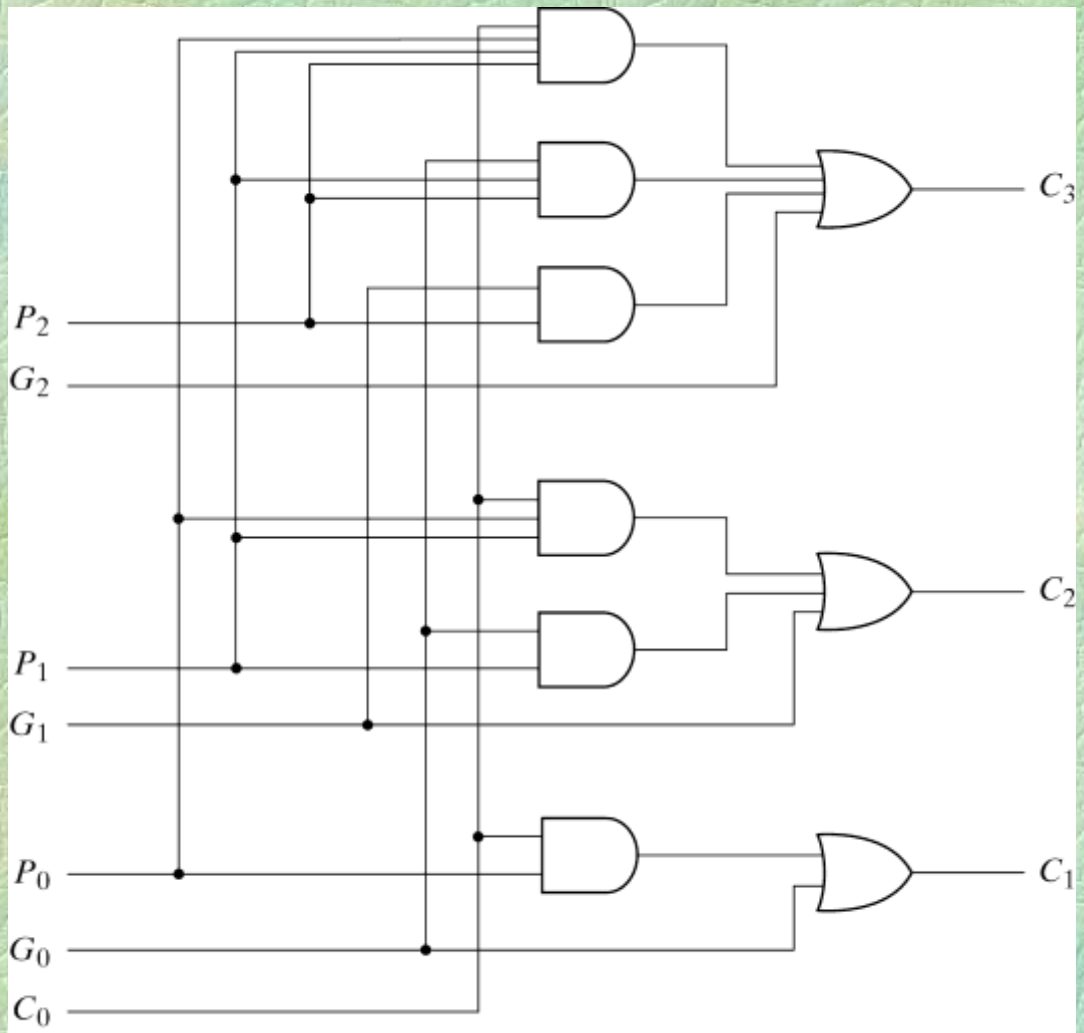


Fig. 4-11 Logic Diagram of Carry Lookahead Generator

Adder with carry lockahead Genertaor

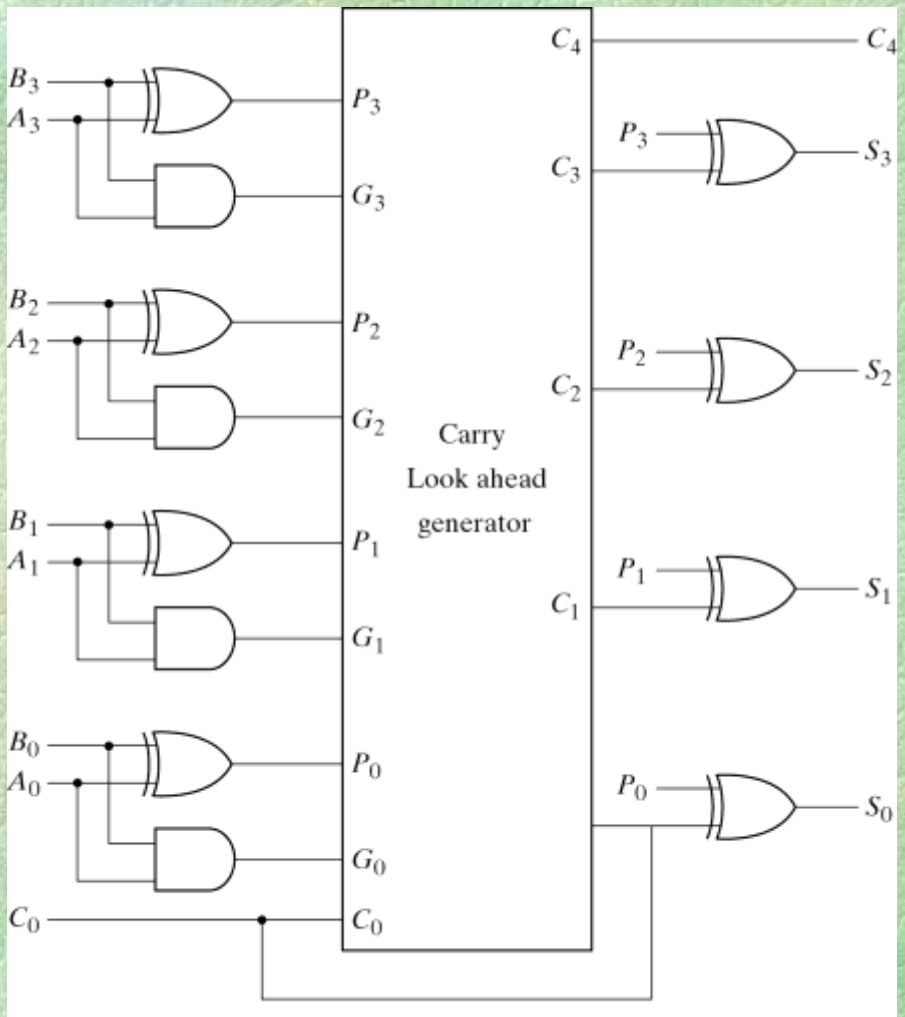


Fig. 4-12 4-Bit Adder with Carry Lookahead

Binary Substractors

- One of the basic arithmetic process in computer system
- One that subtracts 2 and produce their difference is Half-Substractor
- One that subtracts 2 and produce their difference while taking account that 1 have been borrowed by a lower significant stage.
- it is called : Full-Substractor

Half-Substractor

- 2 Input & 2 output (Borrow & Data)
- Boolean Function cannot be simplified
- D = x'y + xy'
- B = x'y

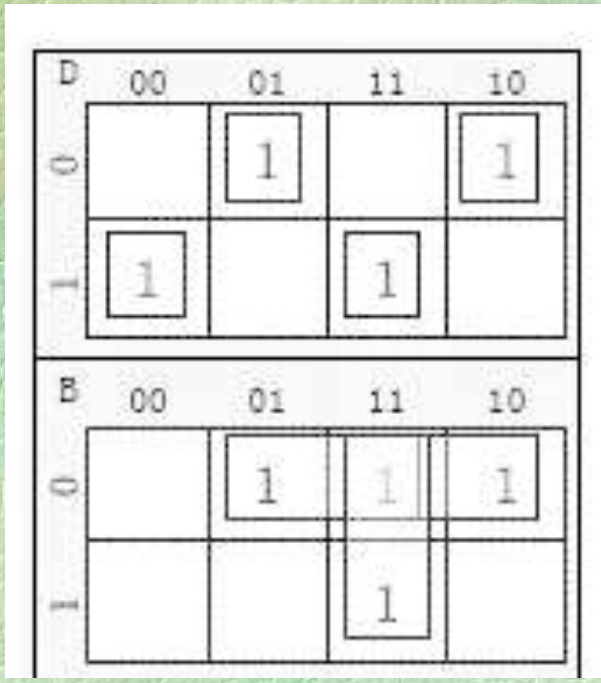
Truth table

X	Y	B	D
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

Full-Substractor

- You do it
- Truth table
- Simplify with K-Map
- Draw the Logic Gate

x	y	z	B	D
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1



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

$B = x'y + x'z + yz$

Adder-Substractor

- The operation A-B =A+ “1’s complement of B” +1
  = A+ “2’s complement of B.
- For unsigned numbers, this gives A-B if A>= B or the 2’s complement of (B-A) if A < B.
- For signed numbers the result is A-B provided that there is no overflow.
- The addition and subtraction operations can be combined into one circuit with one common binary adder and including an X-OR gate with each full adder.

Adder-Substractor

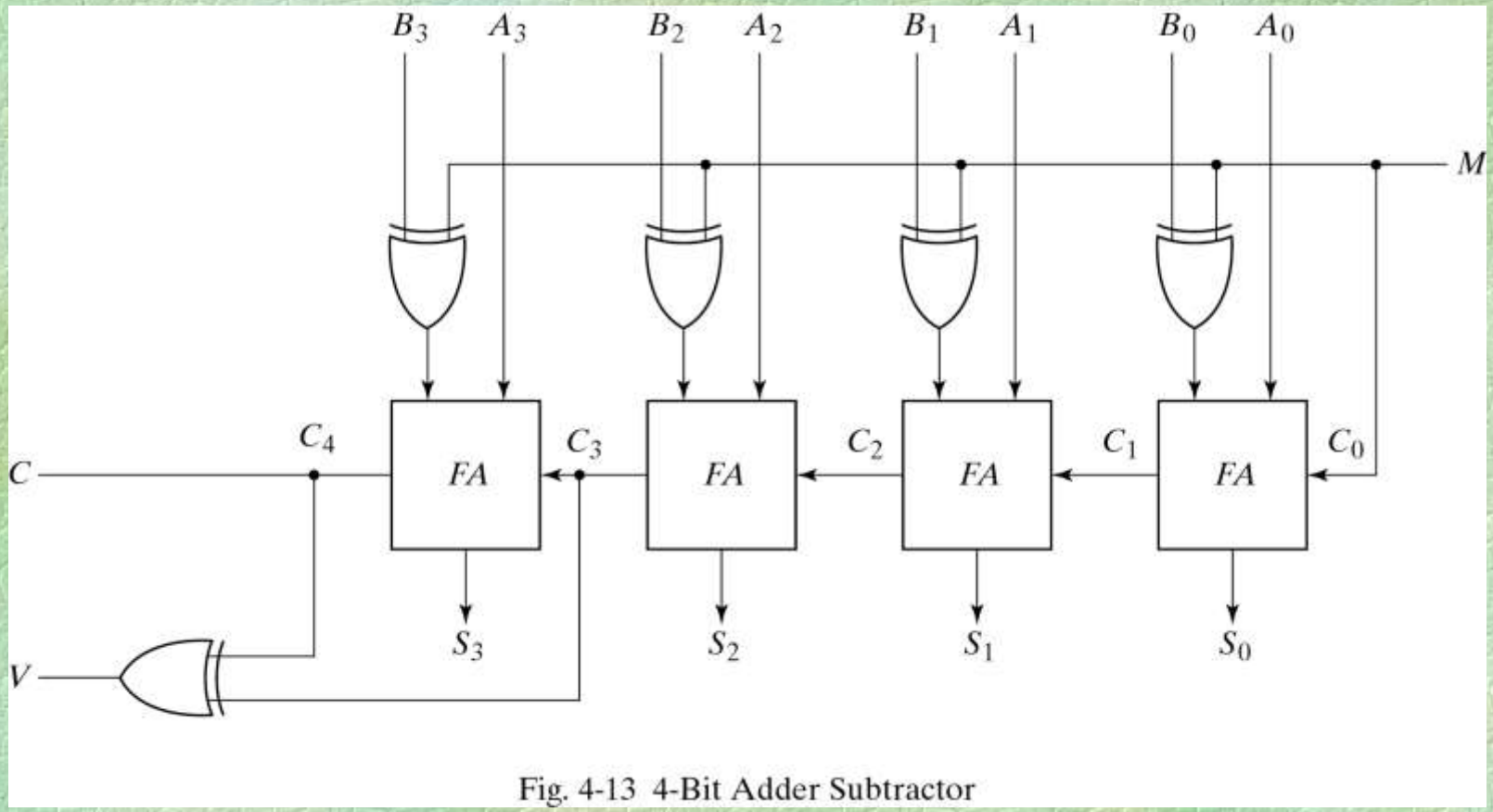


Fig. 4-13 4-Bit Adder Subtractor

When M=0, the circuit is adder (since $B \oplus 0 = B$), and when M=1, the circuit is subtractor (since $B \oplus 1 = B'$).

Overflow

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

Binary Multiplier

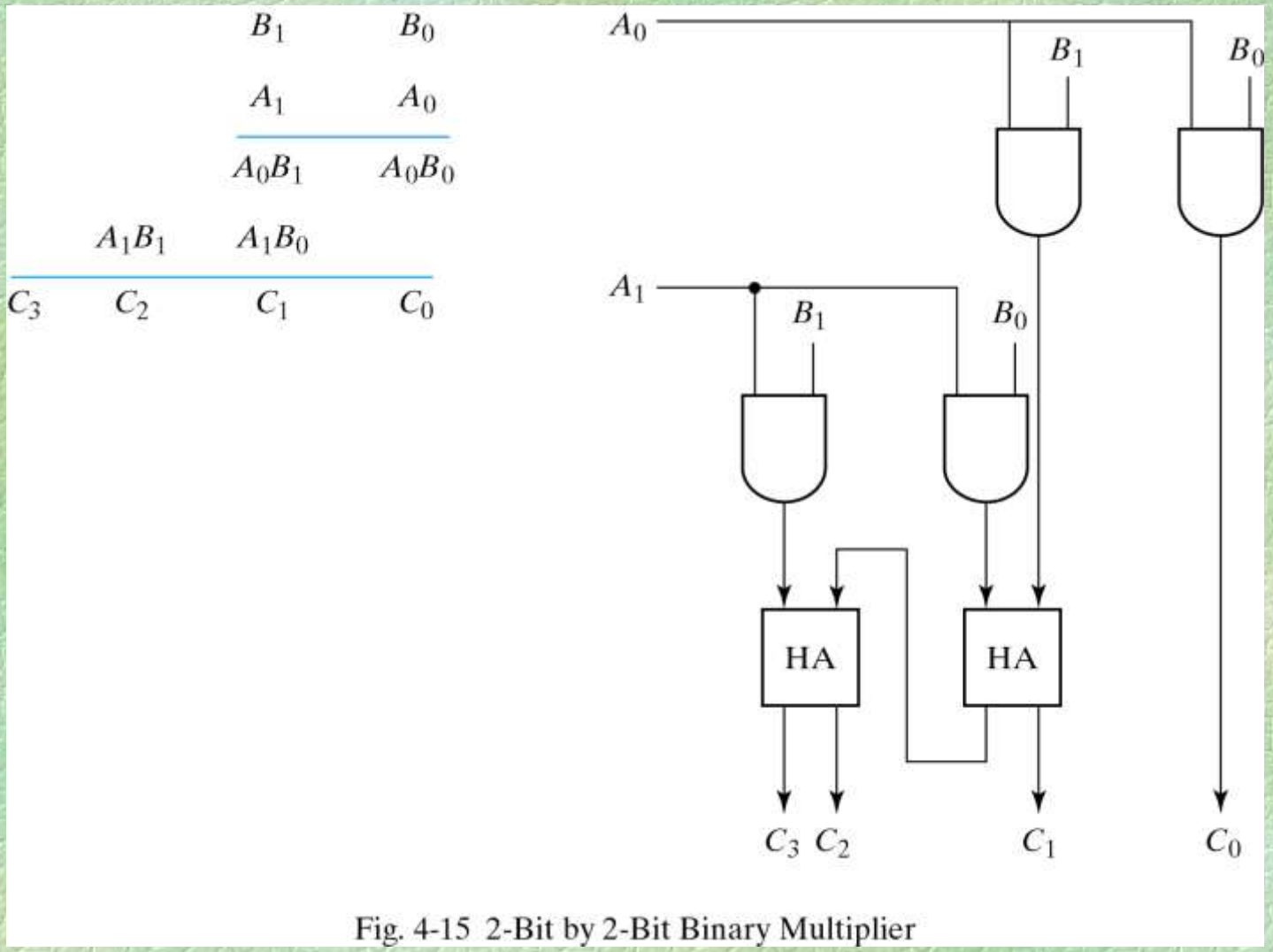


Fig. 4-15 2-Bit by 2-Bit Binary Multiplier

4 by 3 Binary Multiplier

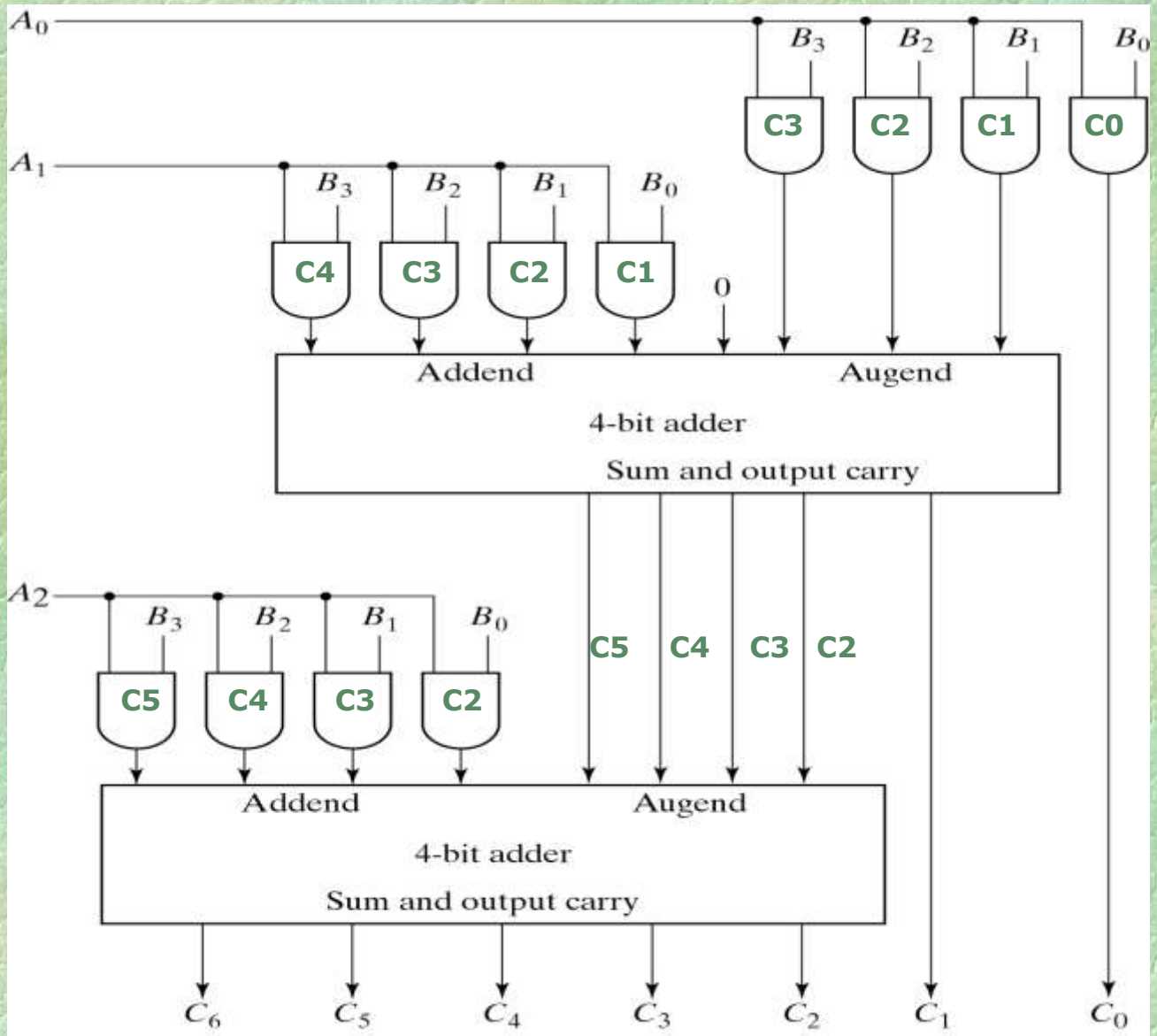


Fig. 4-16 4-Bit by 3-Bit Binary Multiplier

Decoder

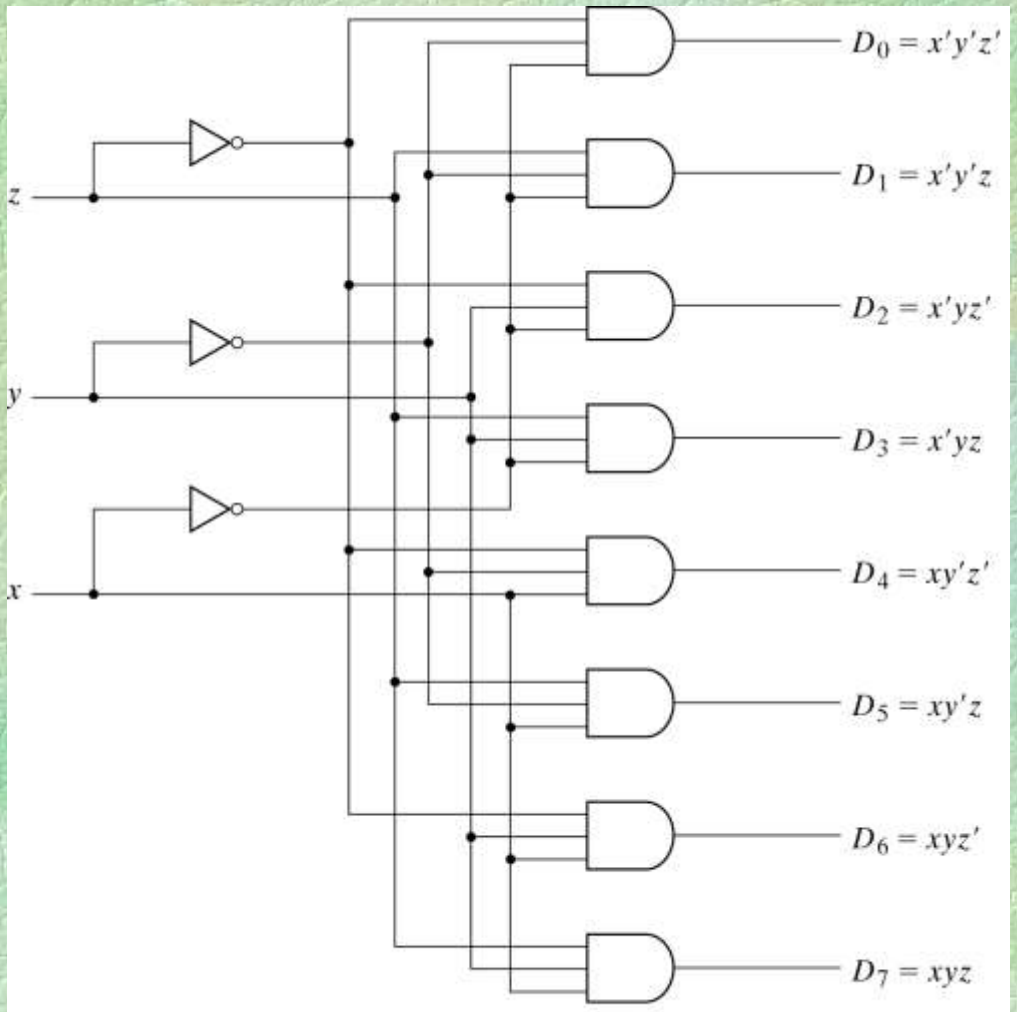
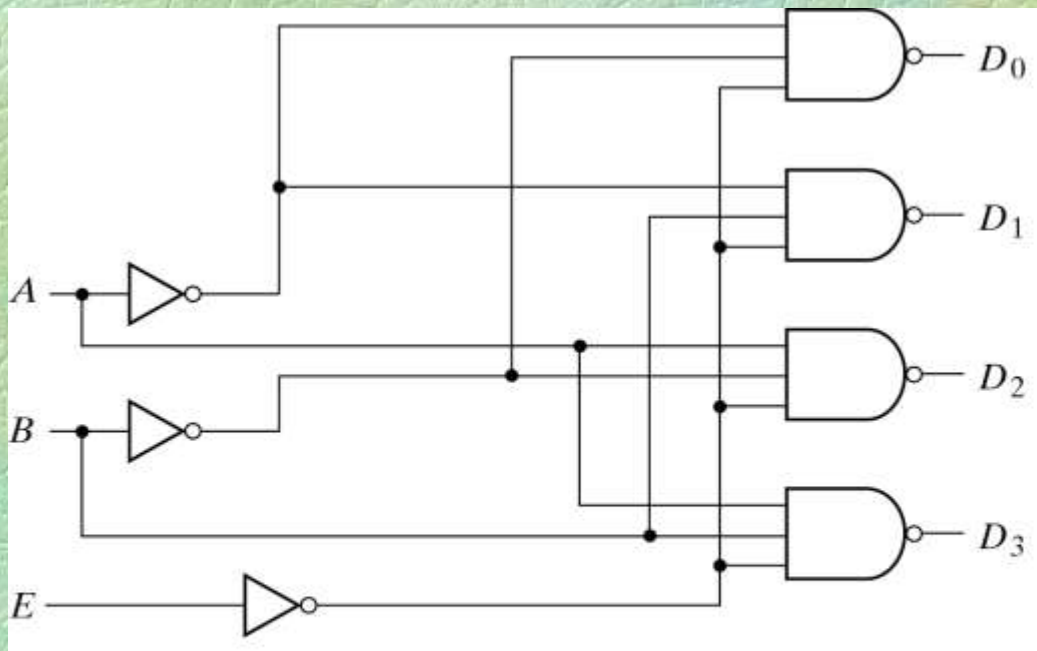


Fig. 4-18 3-to-8-Line Decoder

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

2-4 Line Decoder (1-4 line Demultiplexer)



(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