DIGITAL DESIGN USING HDL

MODULE 2

MODULE 1 SYLLABUS

Gate-level minimization and Introduction to Verilog HDL

Introduction to Boolean Functions,
K Map-Method: 2-Variable, 3-Variable, 4-Variable, Five Variable map,
POS Simplification,
Don't Care Conditions,
NAND NOR Implementation,
Other Two Level Implementations,
Quine-Mc Cluskey Minimization Methods;

HDL Flow, Module Declaration,
Gate Delays,
Boolean Expression Assignment,
User Defined Primitives

MODULE 2 SYLLABUS

Combinational Logic Circuit

Design Procedure,
Binary Adder-Subtractor,
Decimal Adder,
Binary Multiplier,
Magnitude Comparator,
Decoders and Encoders,
Multiplexers and Demultiplexers;

Verilog Models of CLC: Gate Level Modeling,
Three State Gates,
Data Flow modeling,
Behavioral Modeling,
Test Bench

Design Procedure

1. From the **specifications of the circuit,** determine the required number of inputs and outputs and assign a symbol to each.

2. **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** (manually or by simulation).

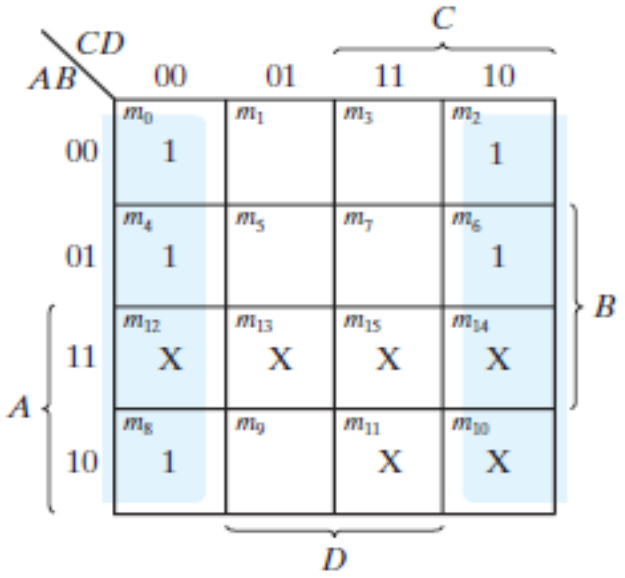
Code Conversion Example

Truth Table

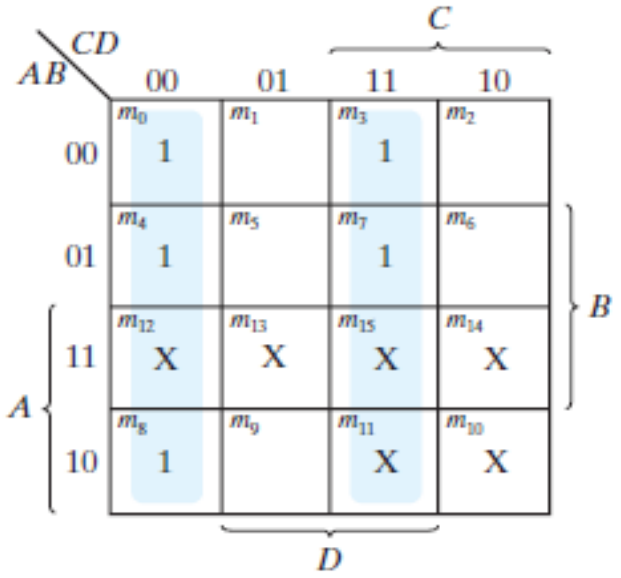
Truth Table for Code Conversion Example

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

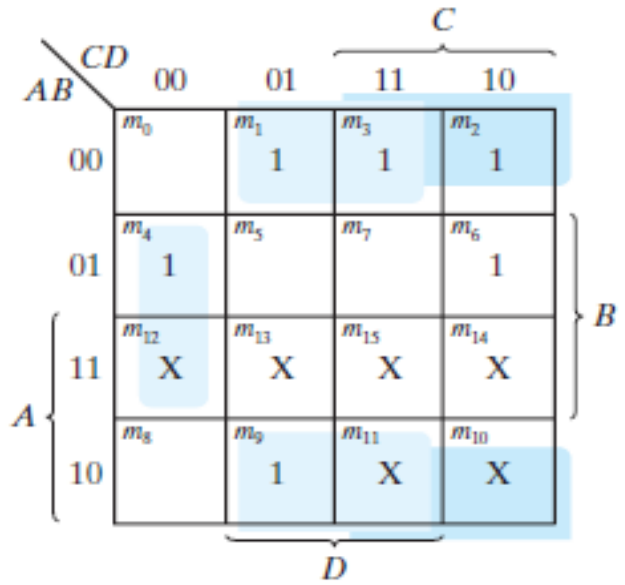
BCD – EXCESS3 CODE CONVERSION K MAPS



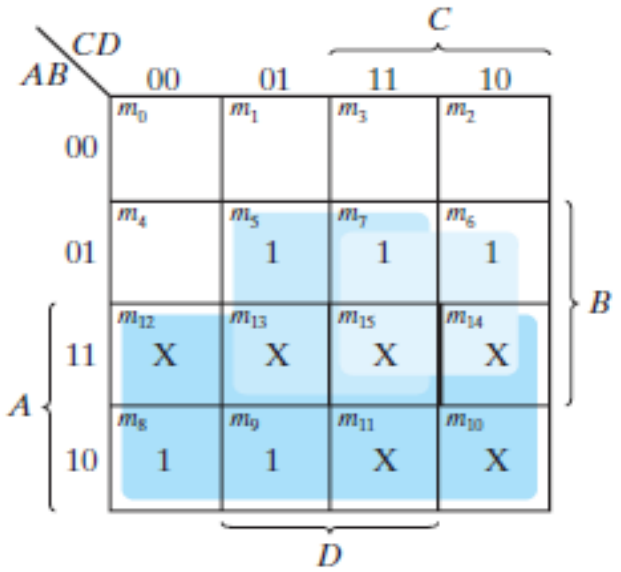
$z = D'$



$y = CD + C'D'$



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

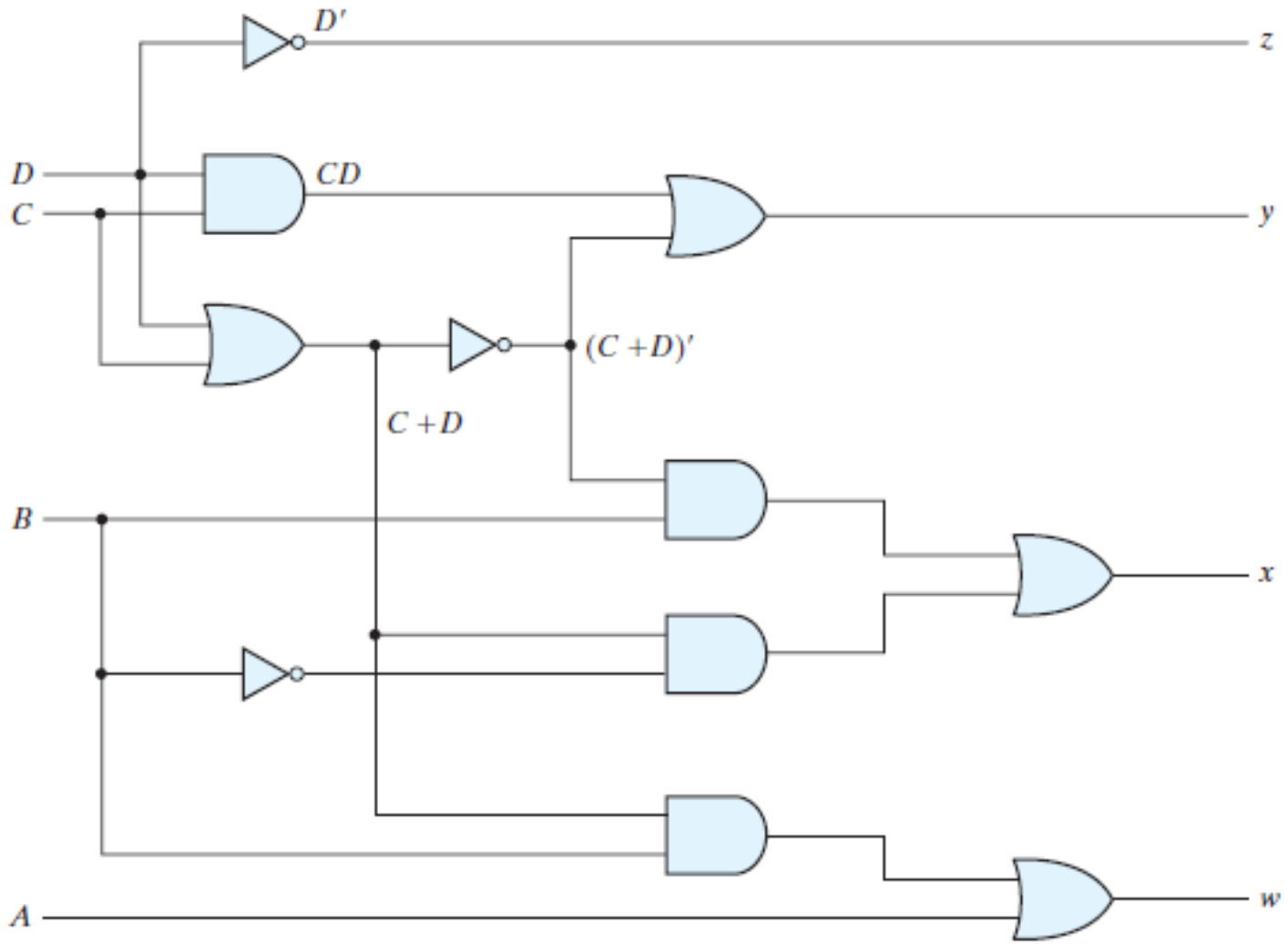


$w = A + BC + BD$

Boolean Expression

$$z = D'$$
$$y = CD + C'D' = CD + (C + D)'$$
$$x = B'C + B'D + BC'D' = B'(C + D) + BC'D'$$
$$= B'(C + D) + B(C + D)'$$
$$w = A + BC + BD = A + B(C + D)$$

LOGIC DIAGRAM

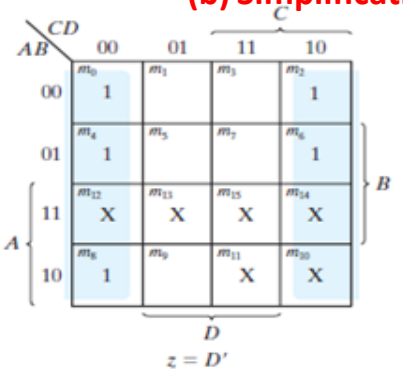


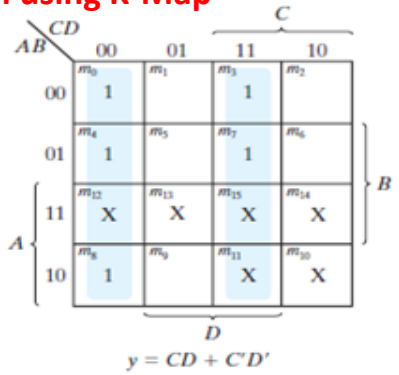
(a) TRUTH TABLE

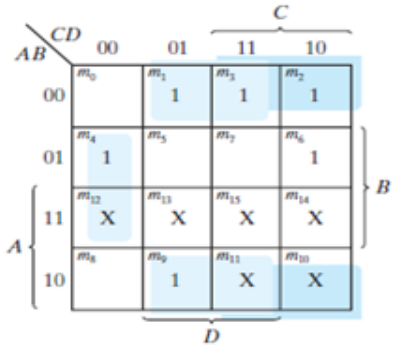
Truth Table for Code Conversion Example

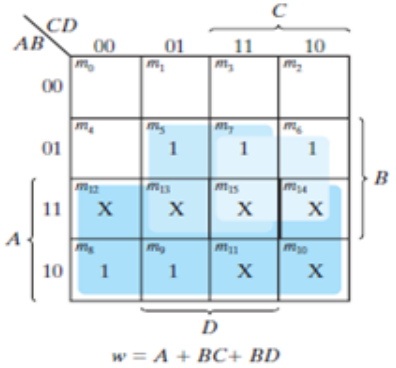
Input BCD				Output Excess-3 Code			
A	B	C	D	w	x	y	z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

(b) Simplification using K-Map

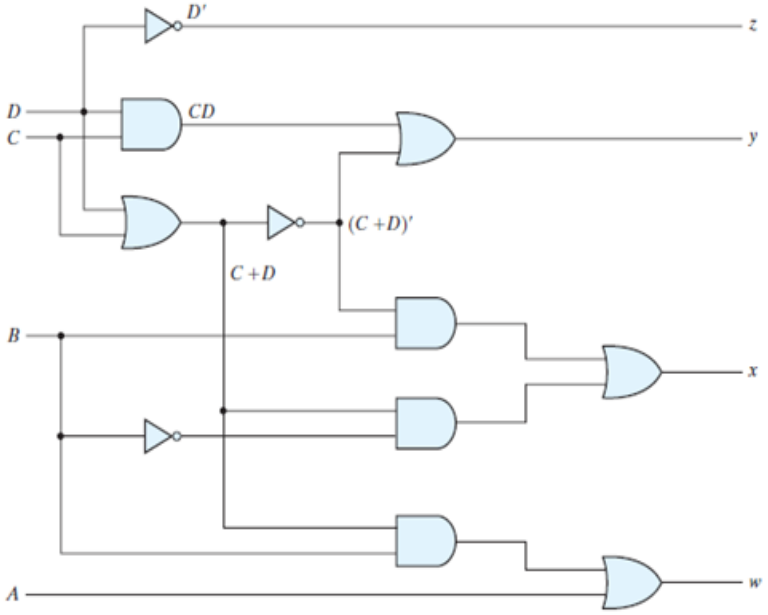








(d) Logic Diagram



(C) Boolean Expressions

$$z = D'$$

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

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

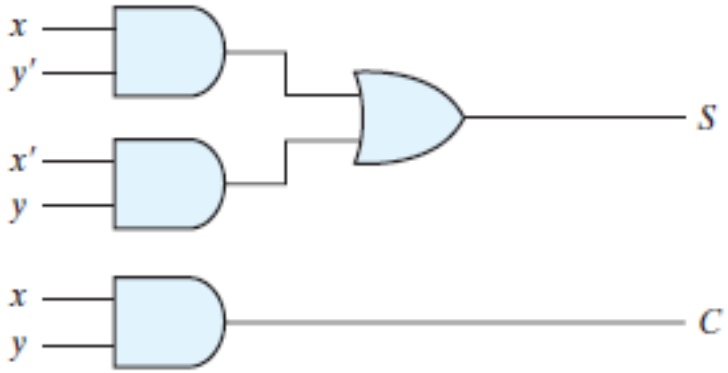
$$w = A + BC + BD = A + B(C + D)$$

BINARY ADDER - SUBTRACTOR

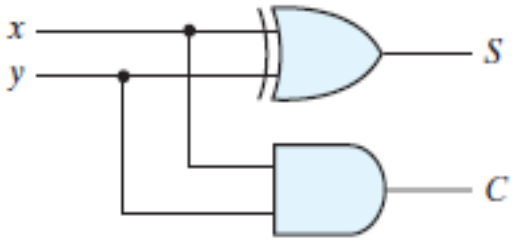
HALF ADDER

Half Adder

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



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



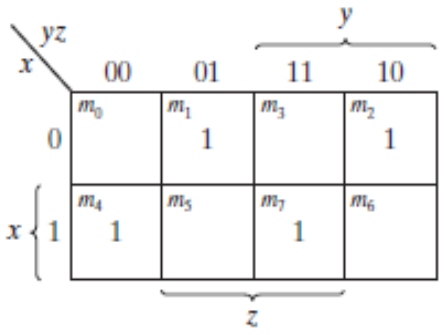
(b) $S = x \oplus y$
$C = xy$

BINARY ADDER - SUBTRACTOR

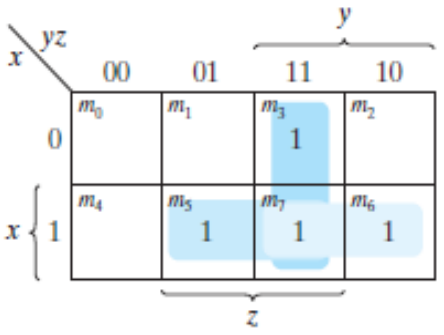
FULL ADDER

Full Adder

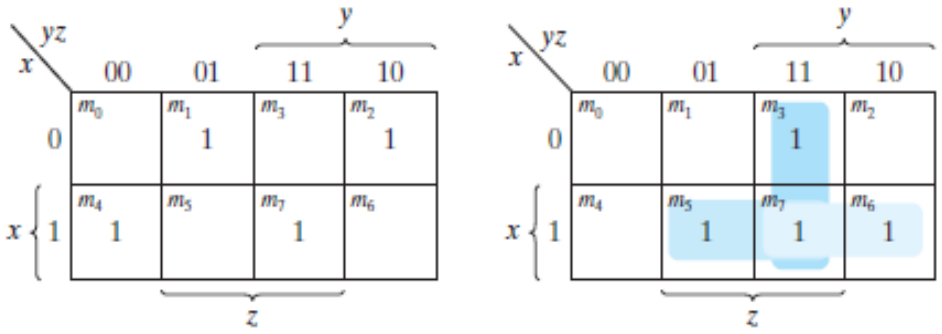
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



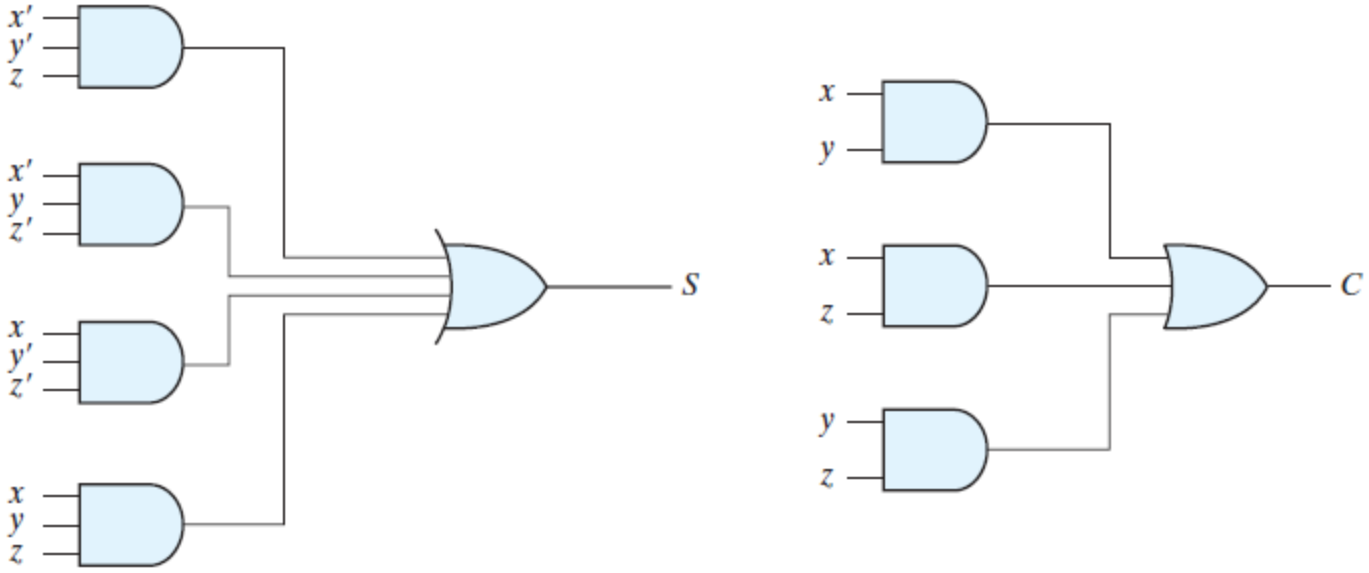
(a) $S = x'y'z + x'yz' + xy'z' + xyz$



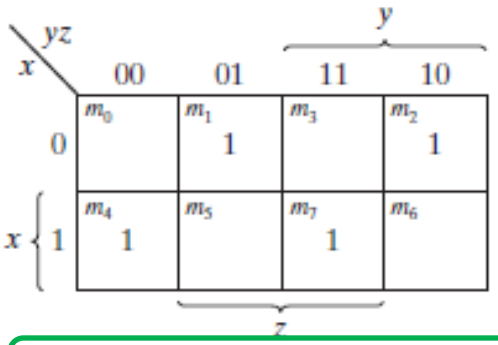
(b) $C = xy + xz + yz$



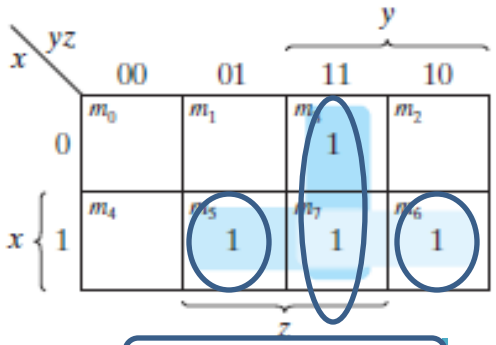
(a) $S = x'y'z + x'yz' + xy'z' + xyz$ (b) $C = xy + xz + yz$



Implementation of full adder in sum-of-products form



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

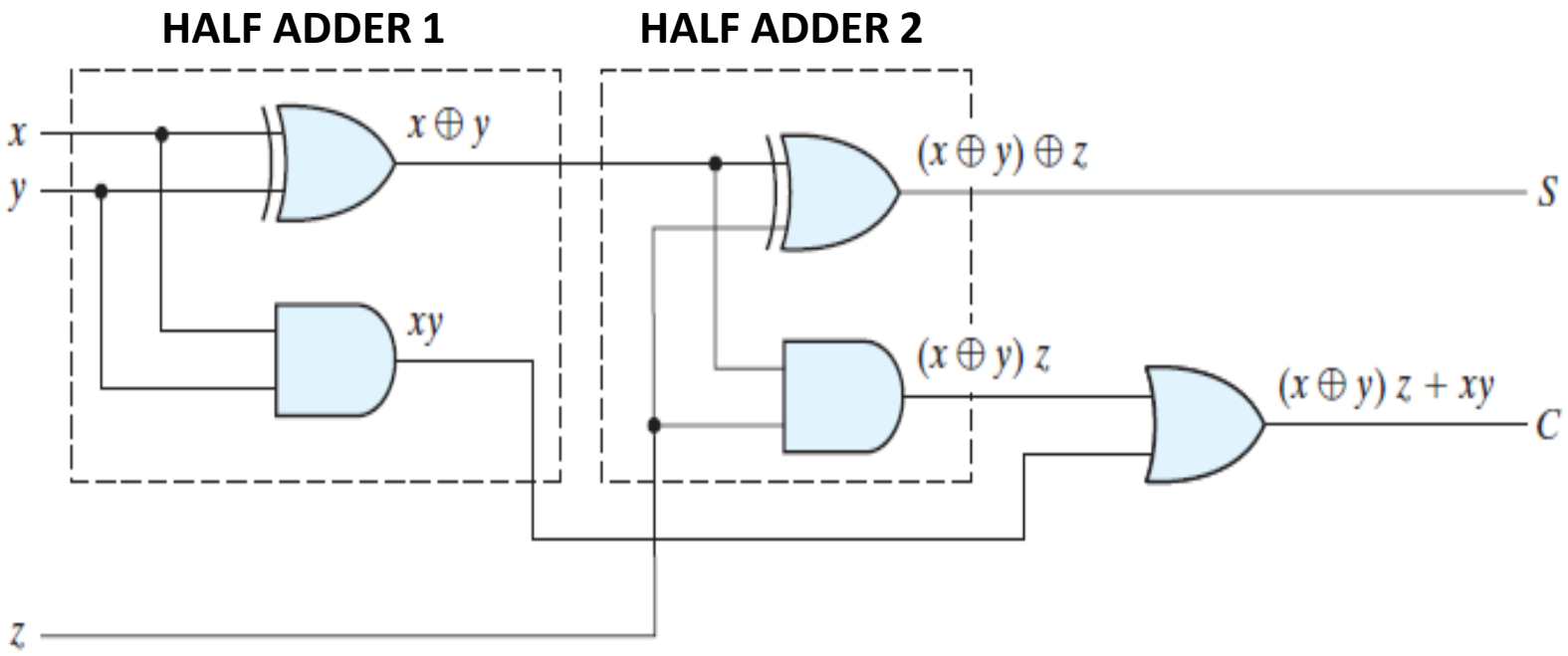


(b) $C = xy + xz + yz$

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

The carry output is

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

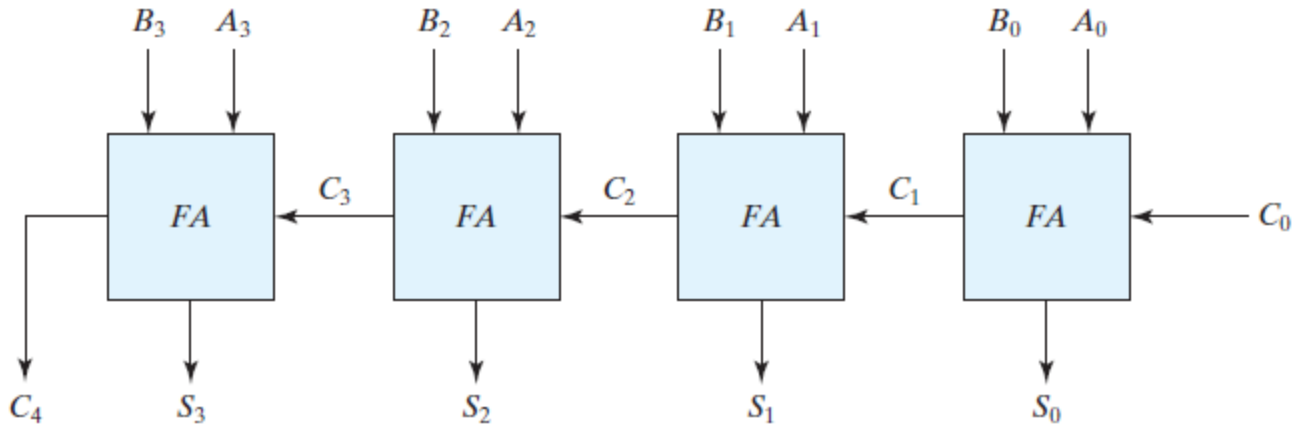


Implementation of full adder with two half adders and an OR gate

BINARY ADDER

A binary adder is a digital circuit that produces the arithmetic sum of two binary numbers.

It can be constructed with full adders connected in cascade, with the output carry from each full adder connected to the input carry of the next full adder in the chain.



4 BIT RIPPLE CARRY ADDER

4 BIT RIPPLE CARRY ADDER

To demonstrate with a specific example, consider the two binary numbers $A = 1011$ and $B = 0011$. Their sum $S = 1110$ is formed with the four-bit adder as follows:

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}

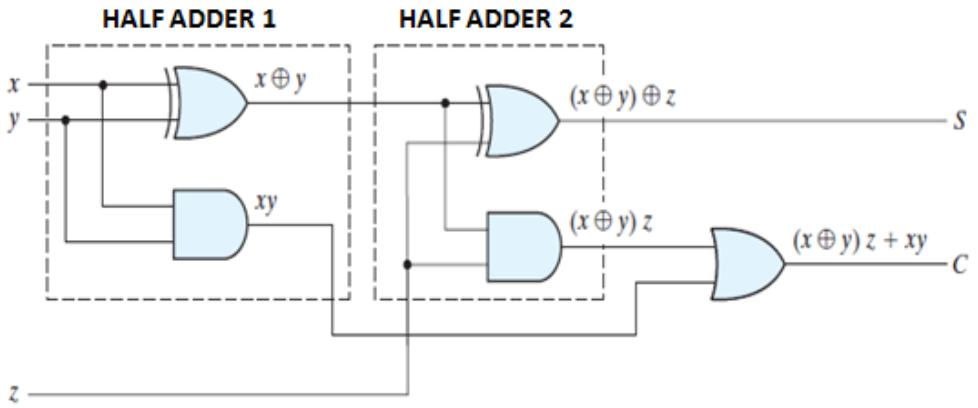
CARRY PROPAGATION

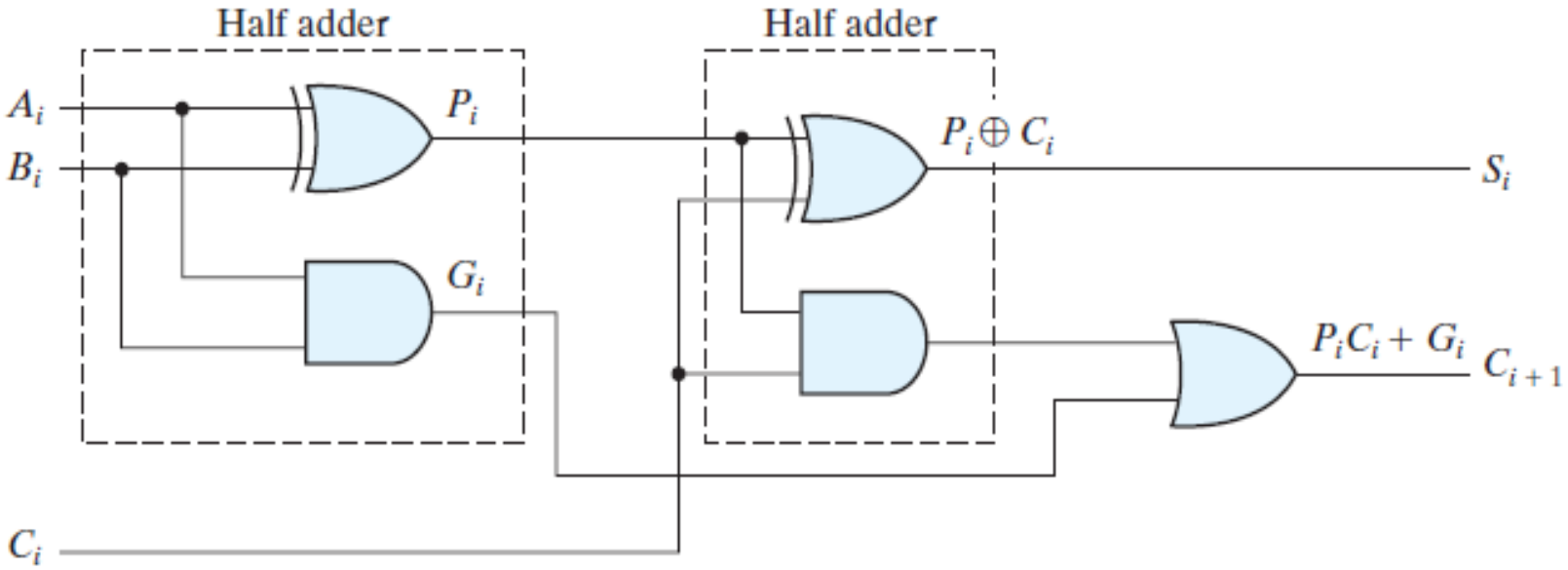
Since each bit of the sum output depends on the value of the input carry, the value of Si at any given stage in the adder will be in its steady-state final value only after the input carry to that stage has been propagated.

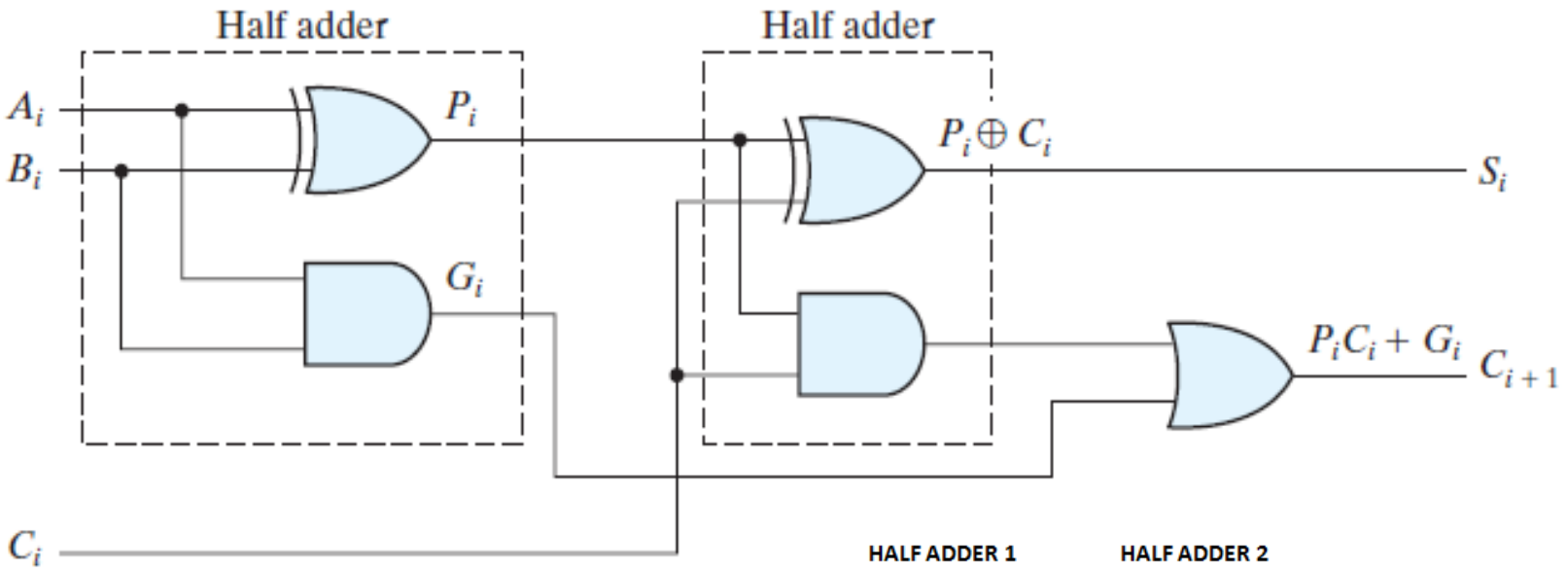
The carry propagation time is an important attribute of the adder because it limits the speed with which two numbers are added.

There are several techniques for reducing the carry propagation time in a parallel adder. The most widely used technique employs the principle of **carry lookahead logic.**

CARRY LOOKAHEAD GENERATION

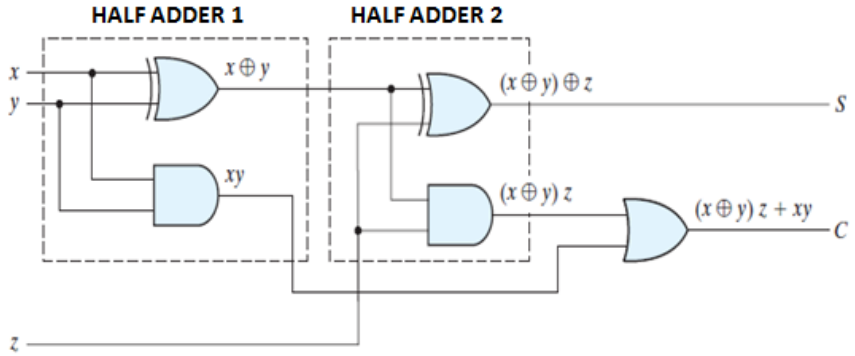






P_i is called a *carry propagate*

G_i is called a *carry generate*



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

$$G_i = A_iB_i \qquad C_{i+1} = G_i + P_iC_i$$

We now write the Boolean functions for the carry outputs of each stage and substitute the value of each Ci from the previous equations:

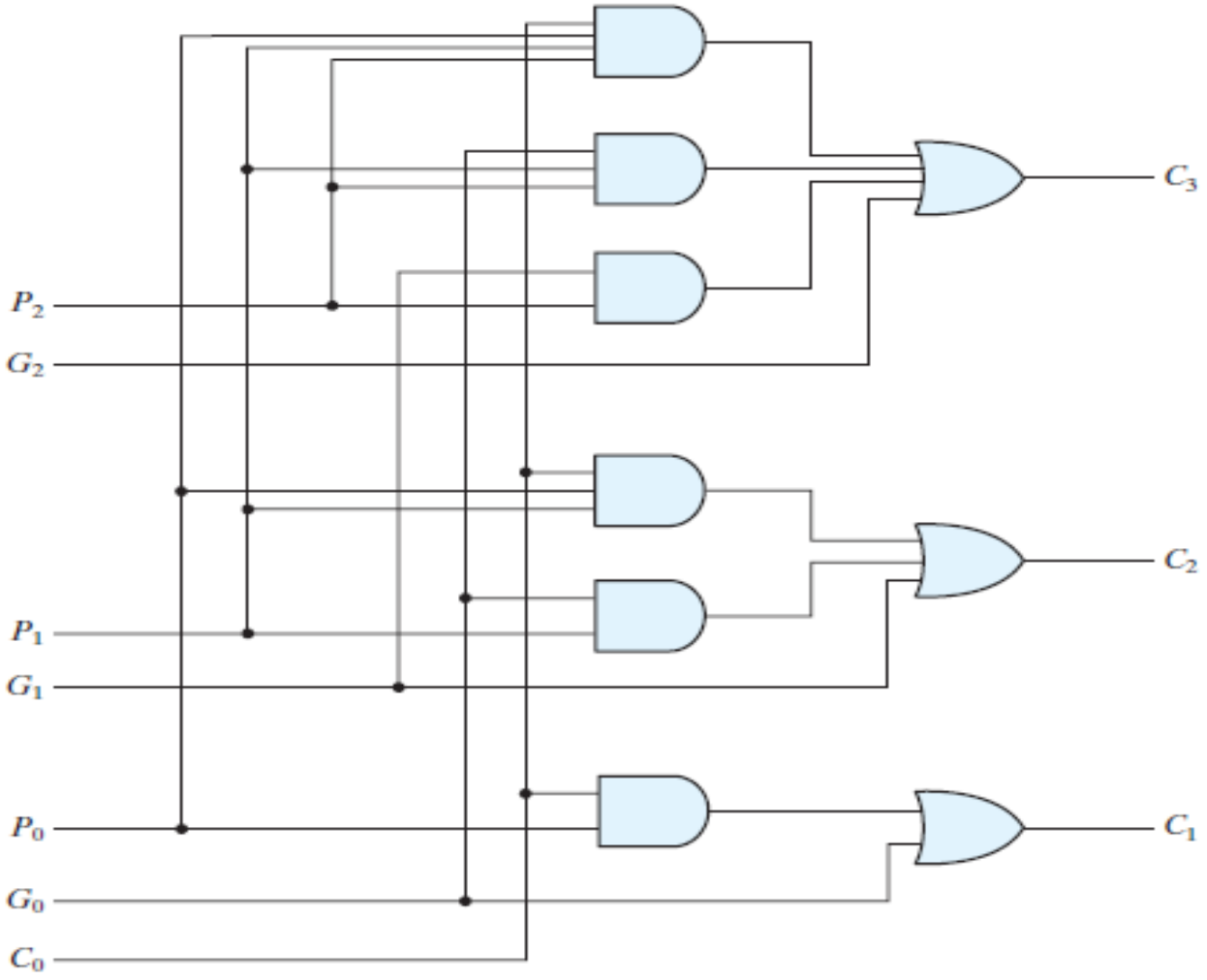
$$C_0 = \text{input carry}$$

$$C_1 = G_0 + P_0C_0$$

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

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

LOGIC DIAGRAM FOR CARRY LOOKAHEAD GENERATOR



FOUR BIT CARRY LOOKAHEAD ADDER