EC-1002 DIGITAL LOGIC & DESIGN

L-T-P-C:3-0-0-3

Module II: Combinational Logic Circuits

SHIVANG TRIPATHI

Combinational circuits

- A combinational circuit consists of logic gates whose **outputs** at any time **depends only on the present input** values at that time.
- Combinational circuit has logic gates with **no** feedback paths or memory elements.

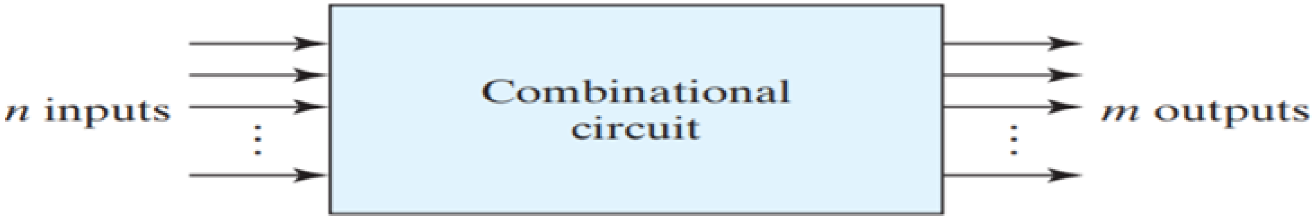


Fig. 2.1 Block Diagram of Combinational circuit

➢ In a sequential circuit, the output at any time depends on the present values as well as the past output values.

- A combinational circuit also can be described by ***m*** Boolean functions, one for each output variable.
- Each output function is expressed in terms of the n input variables.

Design Procedure:

The procedure involves the following steps:

- **Identify** the **inputs** and **outputs** and draw a block diagram.
- Derive the **truth table** that defines the **relationship** between inputs and outputs.
- Write logical expression in SOP or POS
- Minimize the logical expression
- Implement the logic circuit

Combinational circuits example:

- Design a staircase light system that is controlled by two switches, one is at the top of the stairs and the other at the bottom of the stairs.

Let us consider the switches S_1 and S_2, the light is ON when one of the switch is ON.

Adders:

- An adder is a digital logic circuit in electronics that implements addition of numbers.

- Digital system can perform various arithmetic operations over the binary data.

Let us first take a look at the addition of single bits.

0+0 = 0

0+1 = 1

1+0 = 1

1+1 =10 (Sum= 0; Carry = 1)

Adders are classified into two types:

1. Half adder. 2. Full adder.

Half Adder:

Half Adder is a combinational circuit that performs the addition of two bits, this circuit needs two binary inputs and two binary outputs.

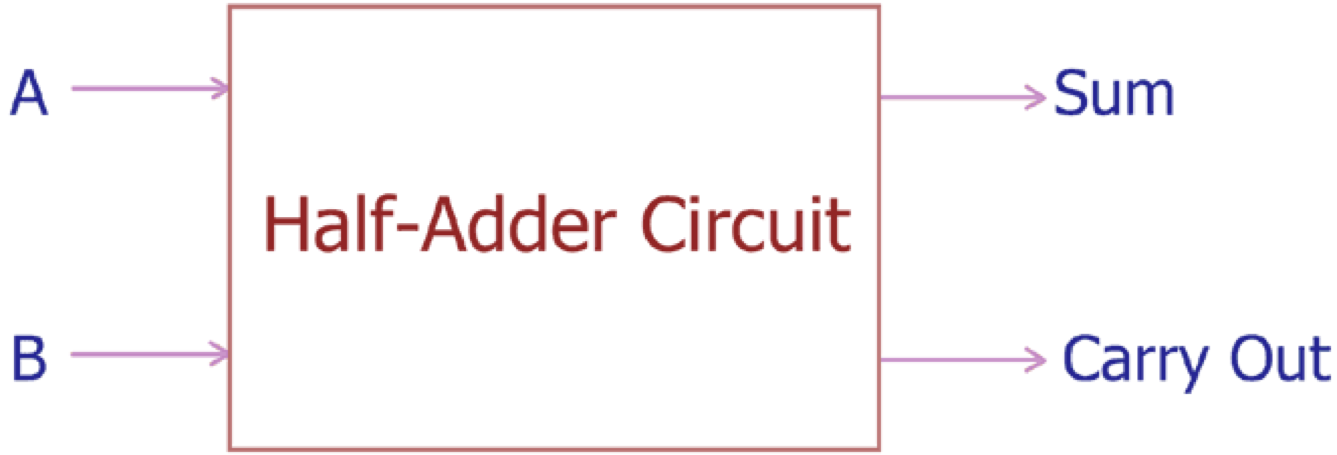


Fig. 2.2 Block diagram of Half adder

Half Adder: Truth table and expressions

Truth Table:

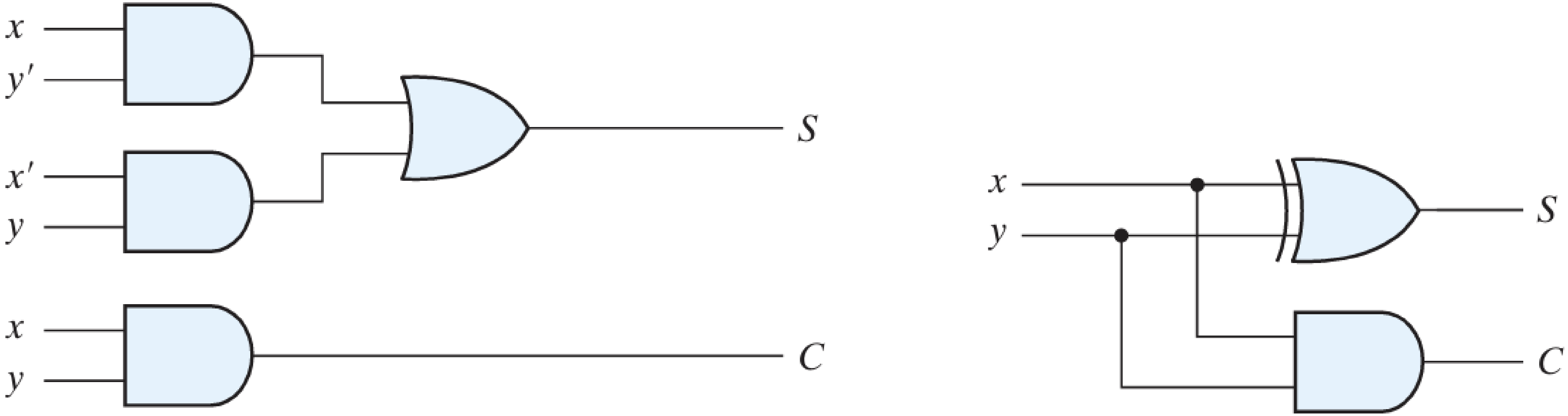
Inputs		Outputs	
Augend A	Addend B	Sum S	Carry C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

The sum (S) bit and carry (C) bit are given by:

$$\mathrm{S} = \bar{A}.B + A.\bar{B} = \mathrm{A} \oplus B$$

$$\mathrm{C} = A.B$$

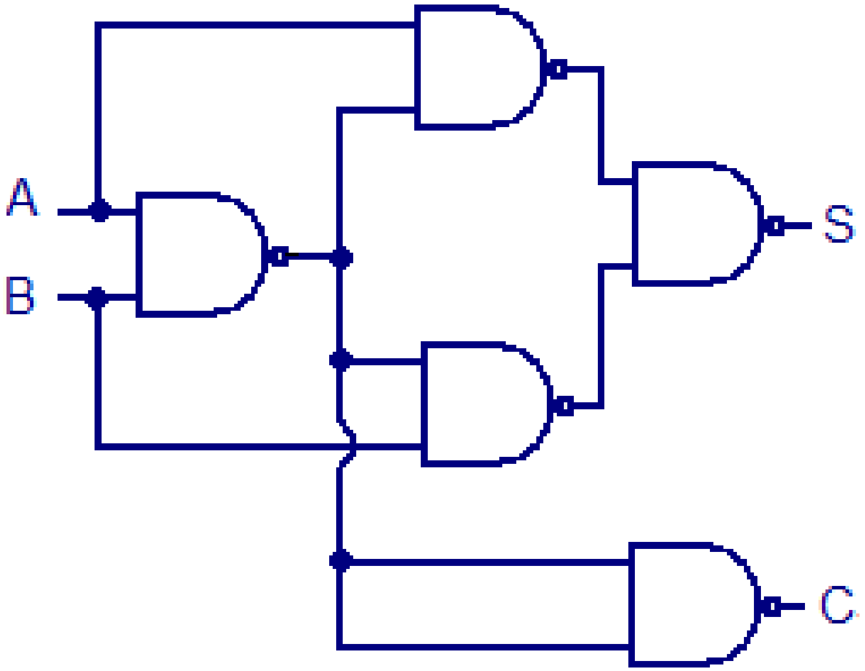
Half adder realization



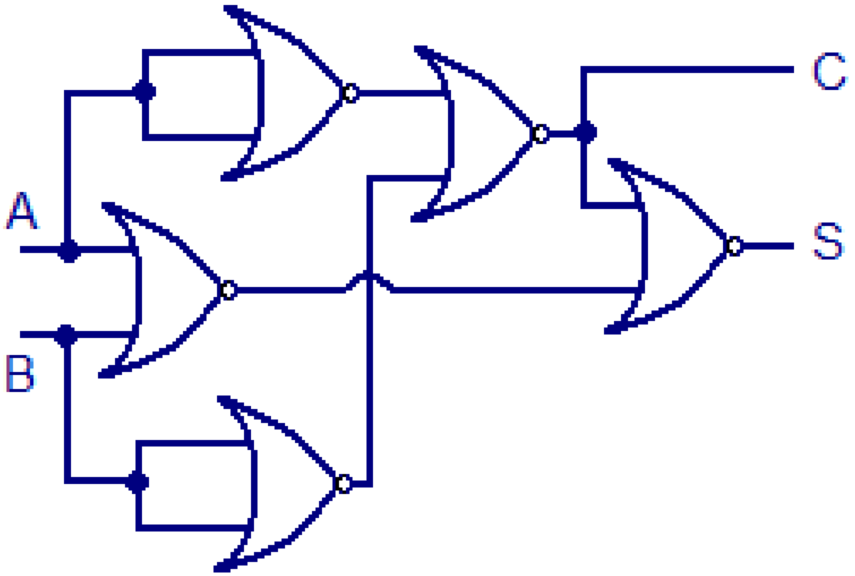
(a) $S = xy' + x'y$
$C = xy$

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

Half adder realization: Using Universal Gates



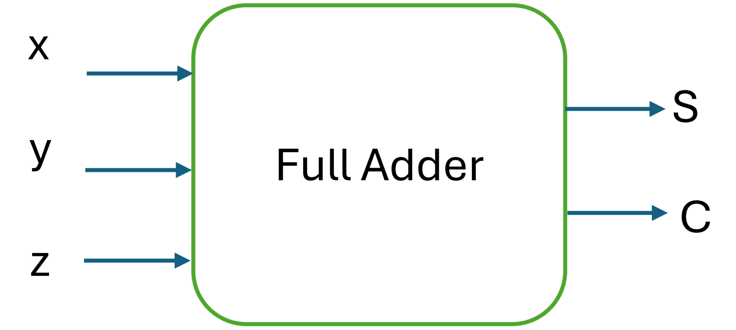
Half adder using NAND logic



Half adder using NOR logic

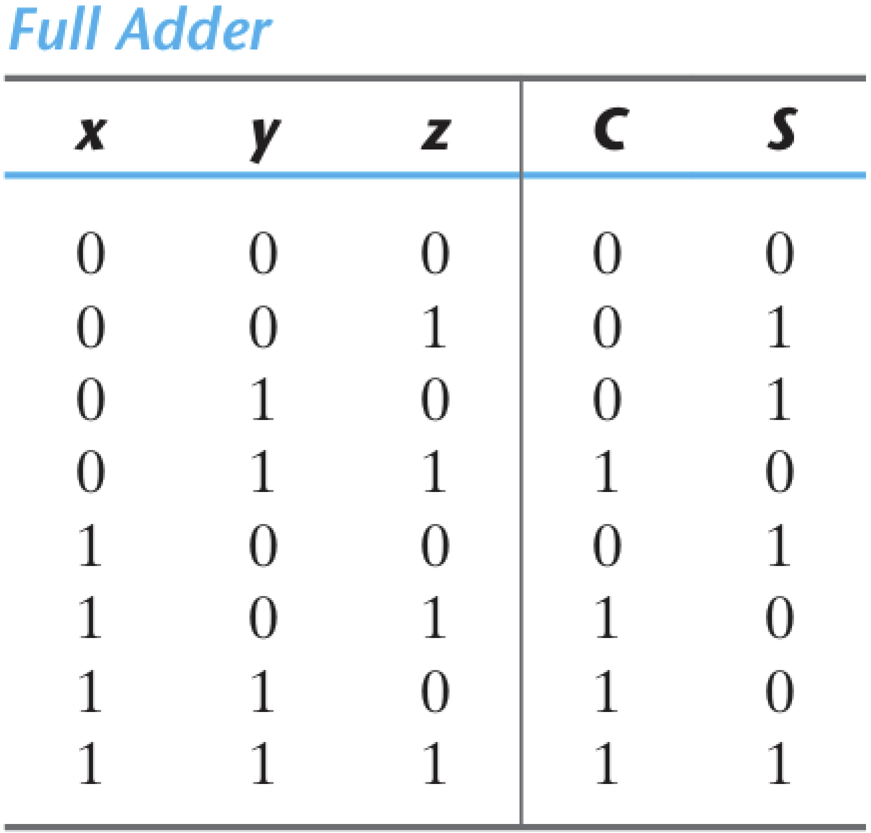
Full adder:

- Full Adder is a combinational circuit that performs the addition of three bits (two significant bits and previous carry).
- It consists of three inputs and two outputs, two inputs are the bits to be added, the third input represents the **carry** form the **previous position**



Z is Carry in from previous calculation

C is Carry-out for next stage



Full Adder

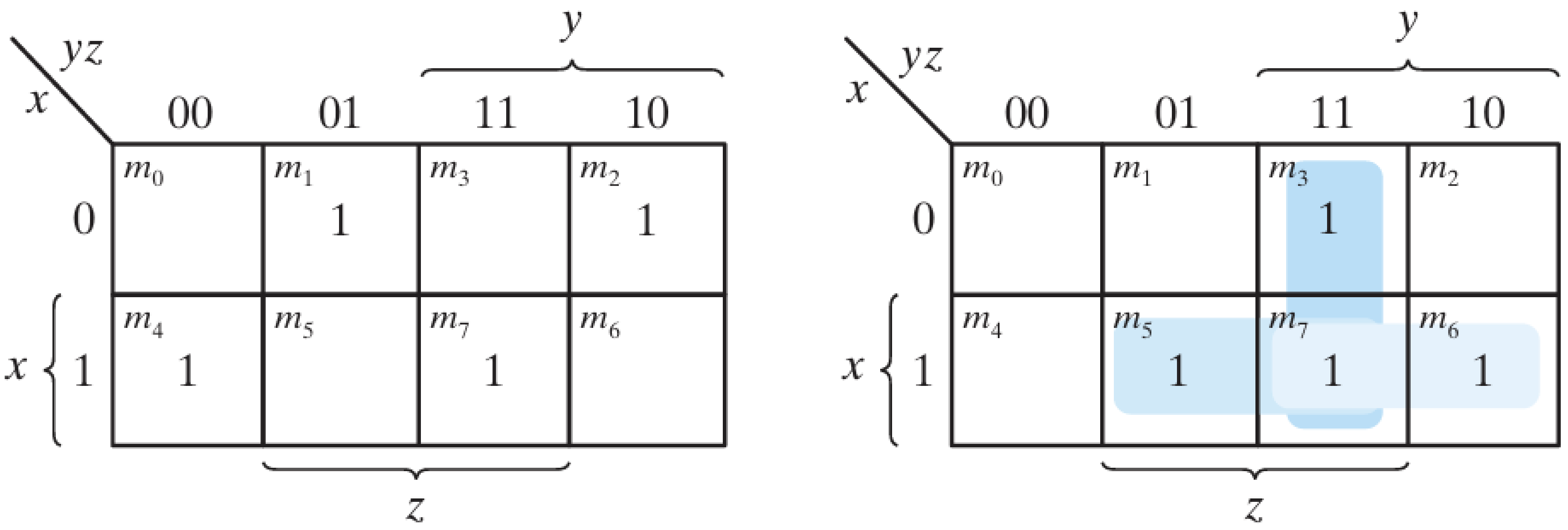
x	*y*	*z*	*C*	*S*
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

SUM : $S = \sum m(1,2,4,7)$

Carry : $C = \sum m(3,5,6,7)$

SUM : $S = \sum m(1,2,4,7)$

Carry : $C = \sum m(3,5,6,7)$



x \ yz	00	01	11	10
0	m_0	m_1 1	m_3	m_2 1
1	m_4 1	m_5	m_7 1	m_6

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

x \ yz	00	01	11	10
0	m_0	m_1	m_3 1	m_2
1	m_4	m_5 1	m_7 1	m_6 1

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

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

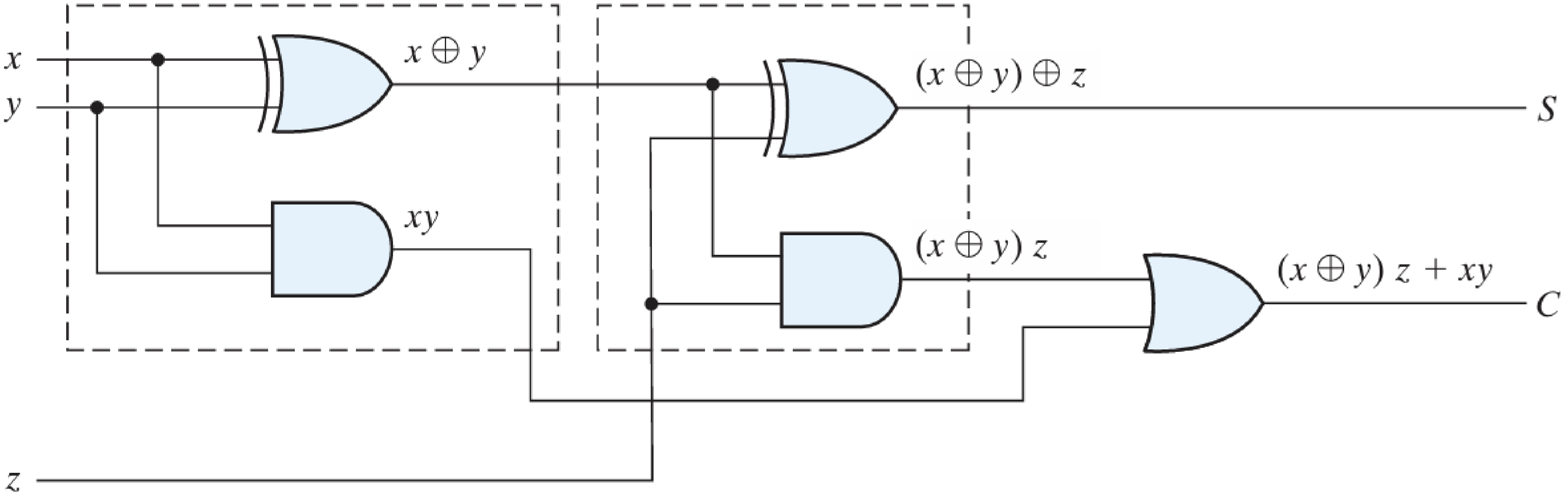
The expression for Carry: $C_{out} = \sum m(3, 5, 6, 7)$

$$C_{out} = \bar{x}yz + x\bar{y}z + xy\bar{z} + xyz$$

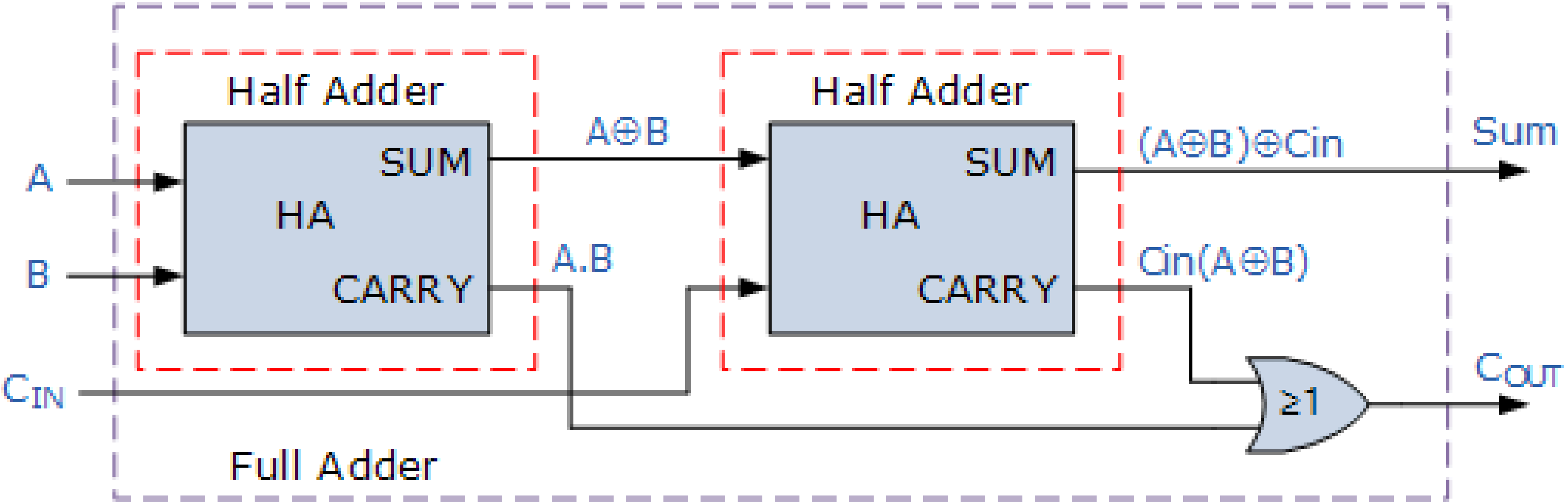
$$C_{out} = \bar{x}yz + x\bar{y}z + xy(\bar{z} + z)$$

$$C_{out} = (\bar{x}y + x\bar{y})z \ + xy$$

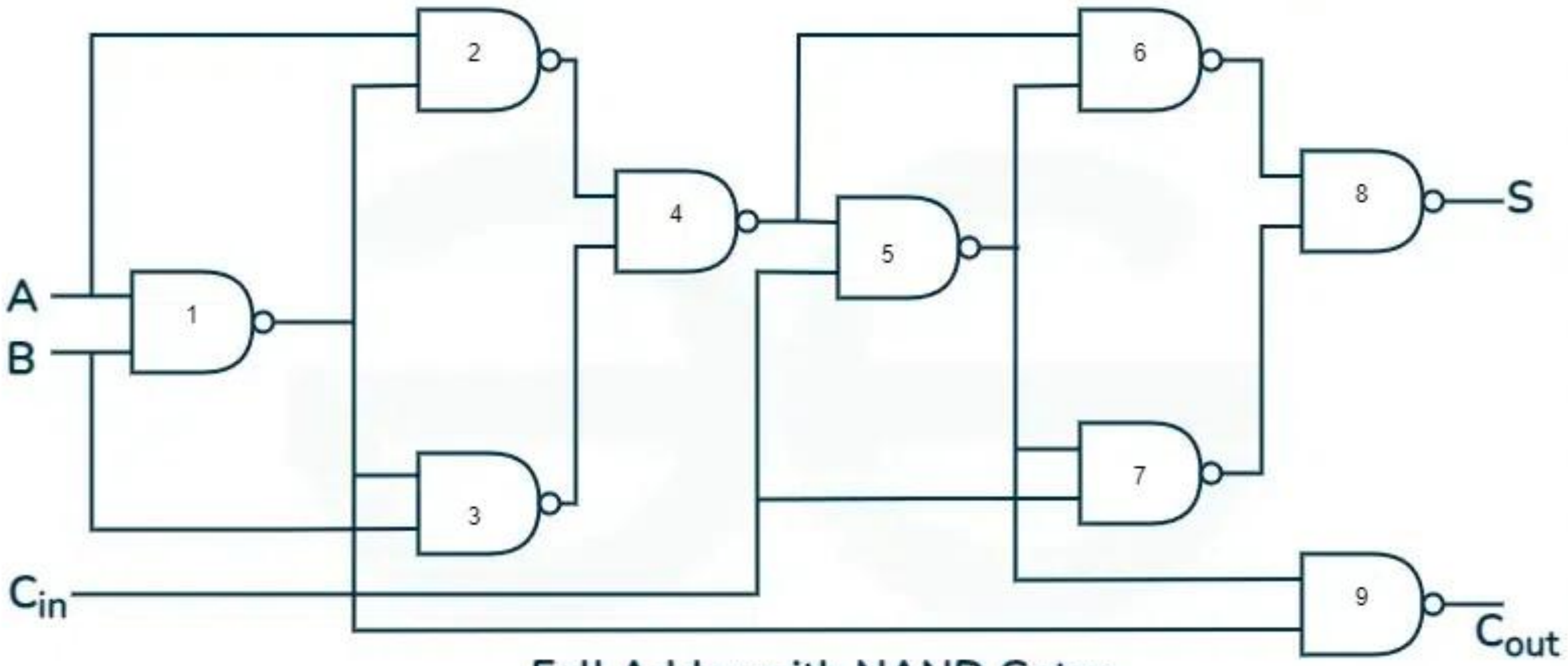
$$C_{out} = (x \oplus y)z \ + xy$$



Full adder: using 2 H.A. and 1 OR

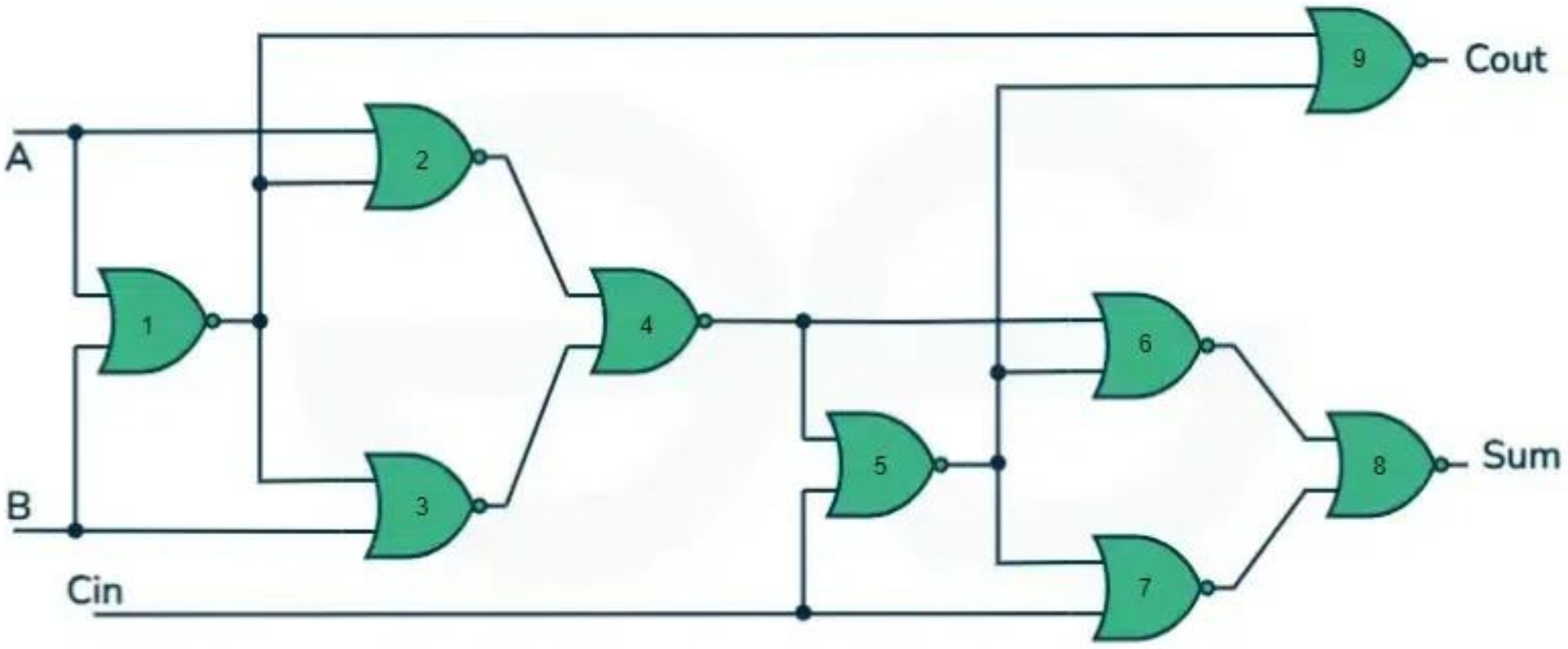


Full adder: using NAND



Full Adder with NAND Gates

Full adder: using NOR



Subtractor

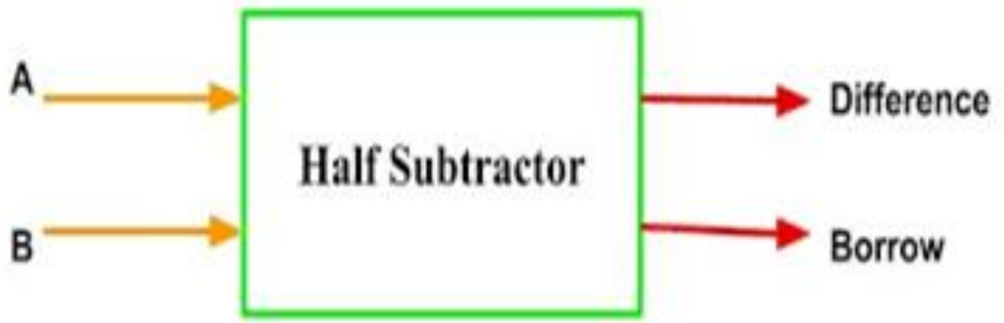
In subtraction, each subtrahend bit of the number is subtracted from its corresponding significant minuend bit to form a difference bit.

There is two type of subtractor:

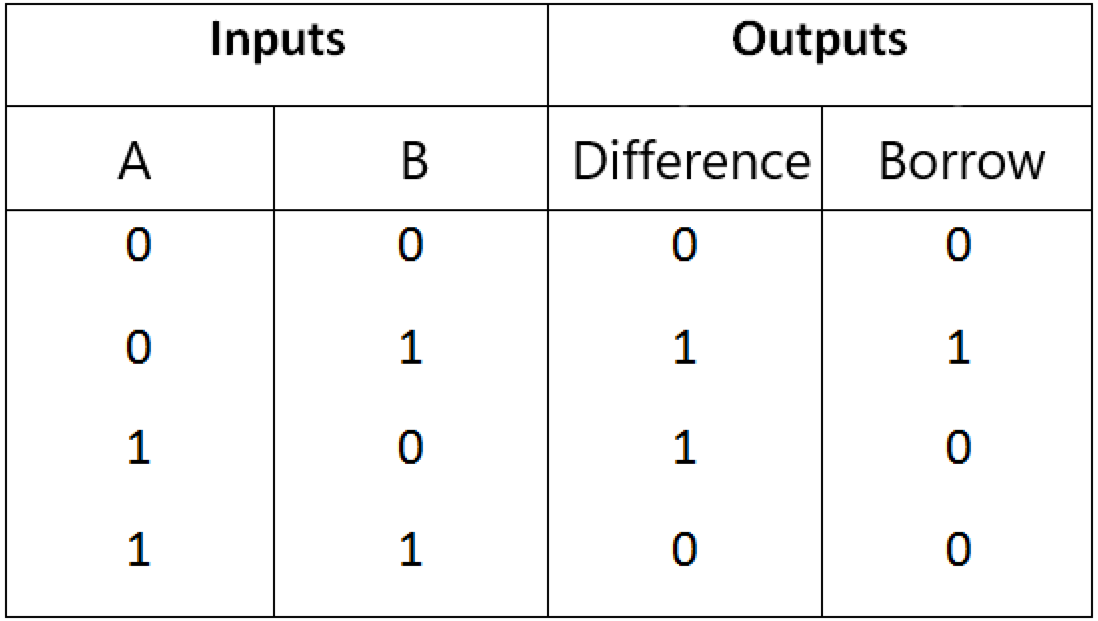
1. Half Subtractor
2. Full Subtractor

Half Subtractor

- A Half Subtractor is a combination circuit that subtract one bit from the other bit and produces the difference.
- It consist of two inputs A and B and two outputs d and b indicates the difference and borrow out respectively.



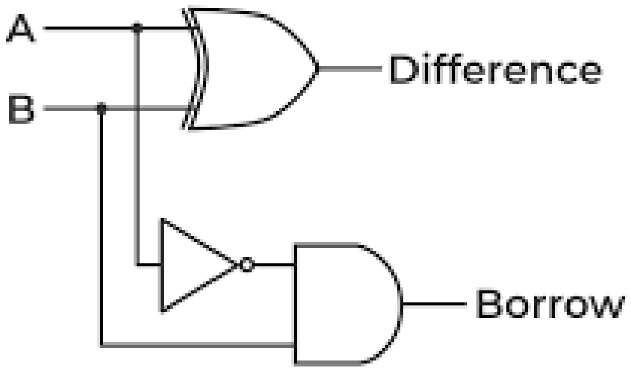
Half Subtractor: Truth Table



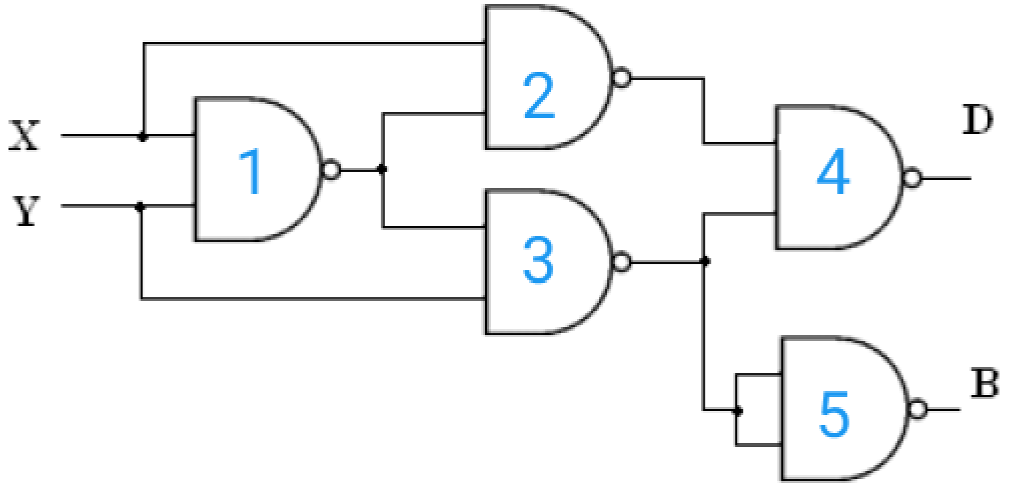
Inputs		Outputs	
A	B	Difference	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

$$D = \overline{A}B + A\overline{B} = A \oplus B$$

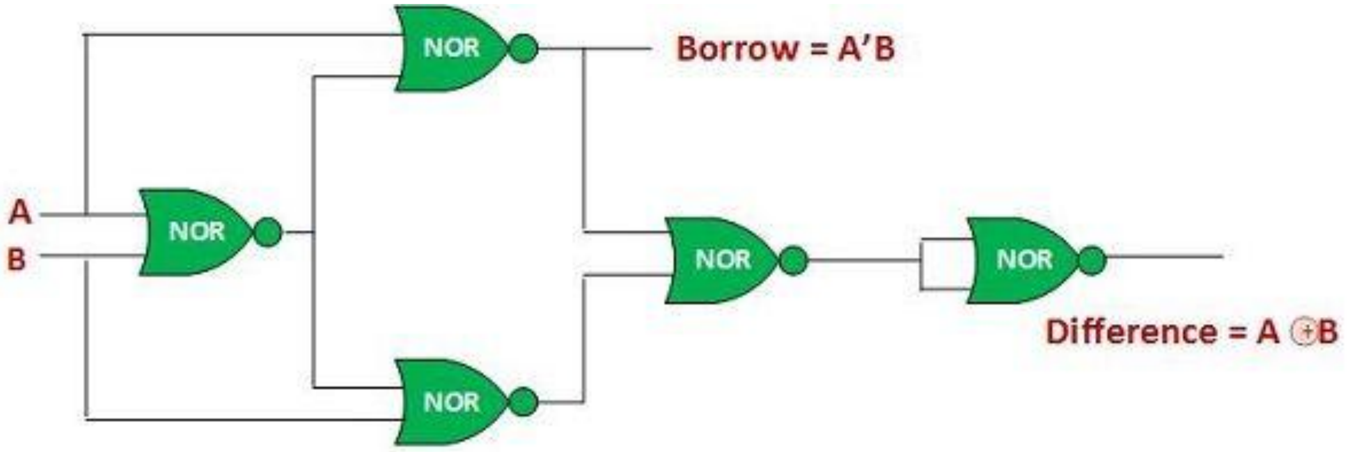
$$C = \overline{A}B$$



Half Subtractor: Using Universal Gates



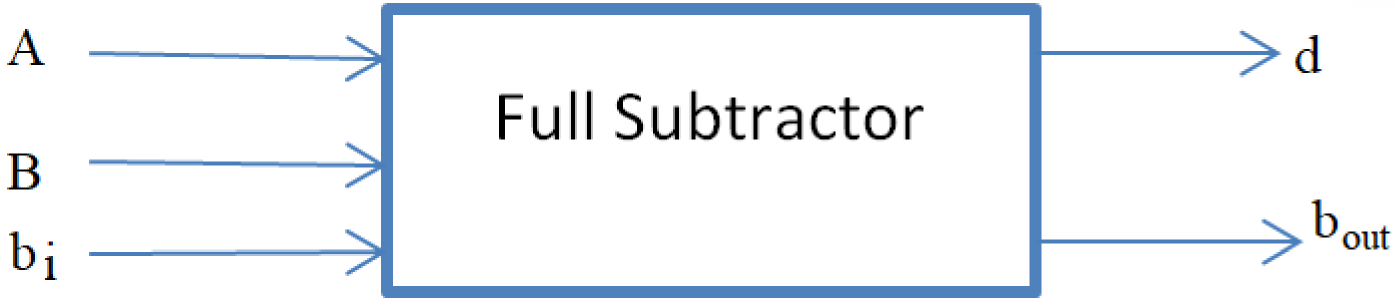
Half-Subtractor using NAND gates



Half Subtractor using NOR Gate

Full Subtractor

- The Half subtractor can be used only for LSB Subtraction. If there is a borrow during the subtraction of the LSBs, it affects the subtraction in the next higher column.
- The subtrahend bit is subtracted from the minuend bit, considering the borrow from that column used for the subtraction in the preceding column.



Full subtractor: Truth Table

Inputs			Difference	Borrow
A	B	b_i	d	b_{out}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

D = ?

B = ??

- Write SOP or POS
- Minimize
- And find out the expression(s) for the D & B

Full subtractor : Logic Diagram

The expressions for:

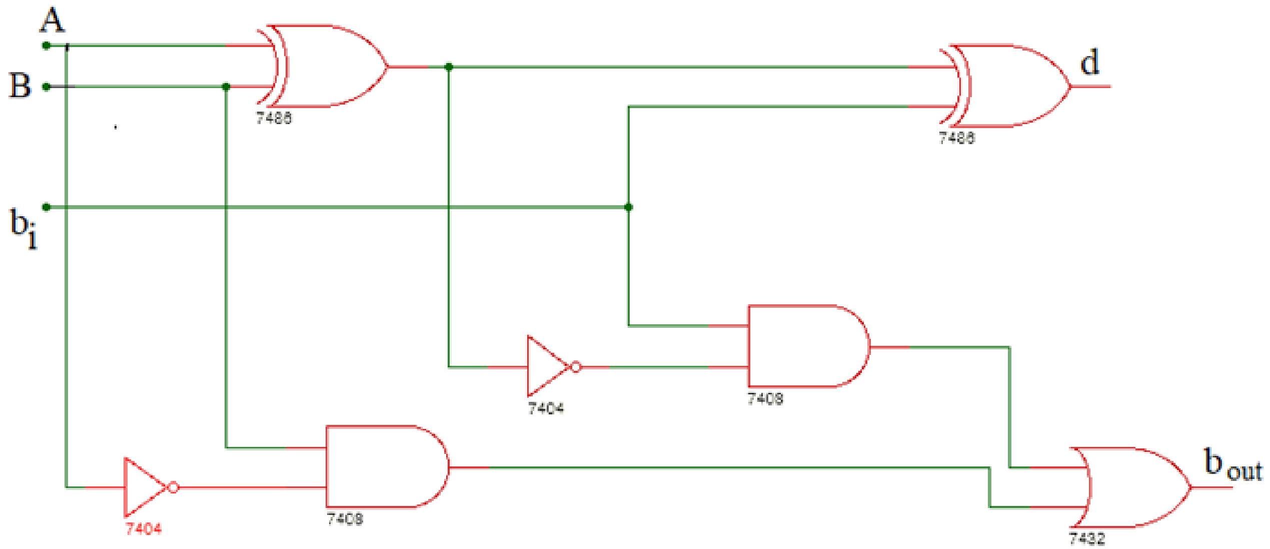
- ➢difference:

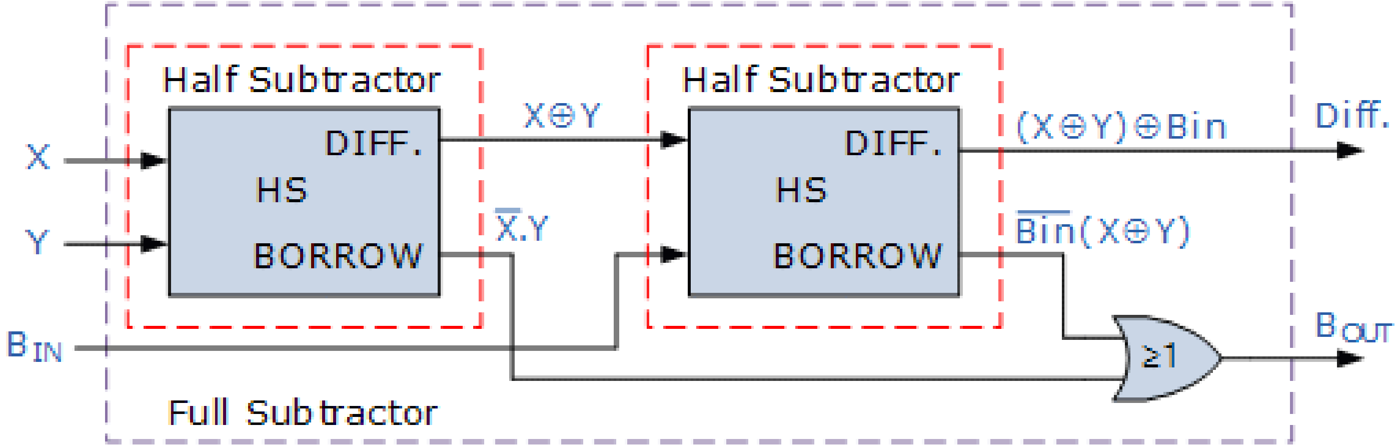
✓$d = A \oplus B \oplus b_i$

- ➢borrow:

✓ $b_{out} = b_i\left(\overline{A \oplus B}\right) + \overline{A}B$

✓$b_{out} = b_i(A \odot B) + \overline{A}B$





Full Subtractor Implementation using NAND gates:

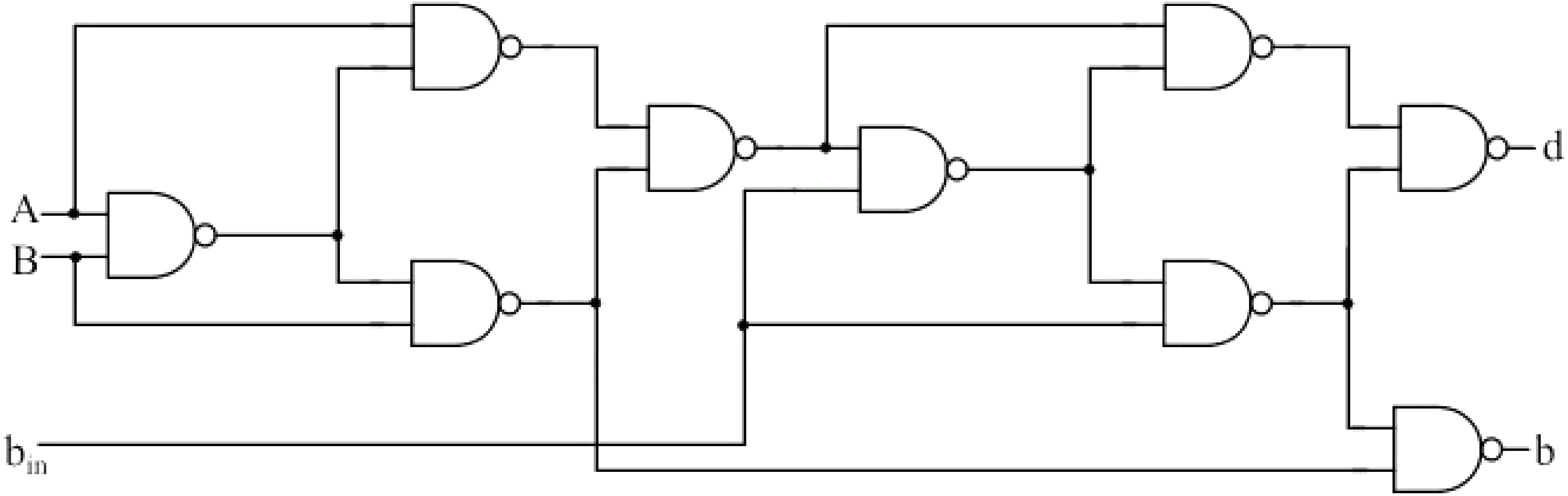
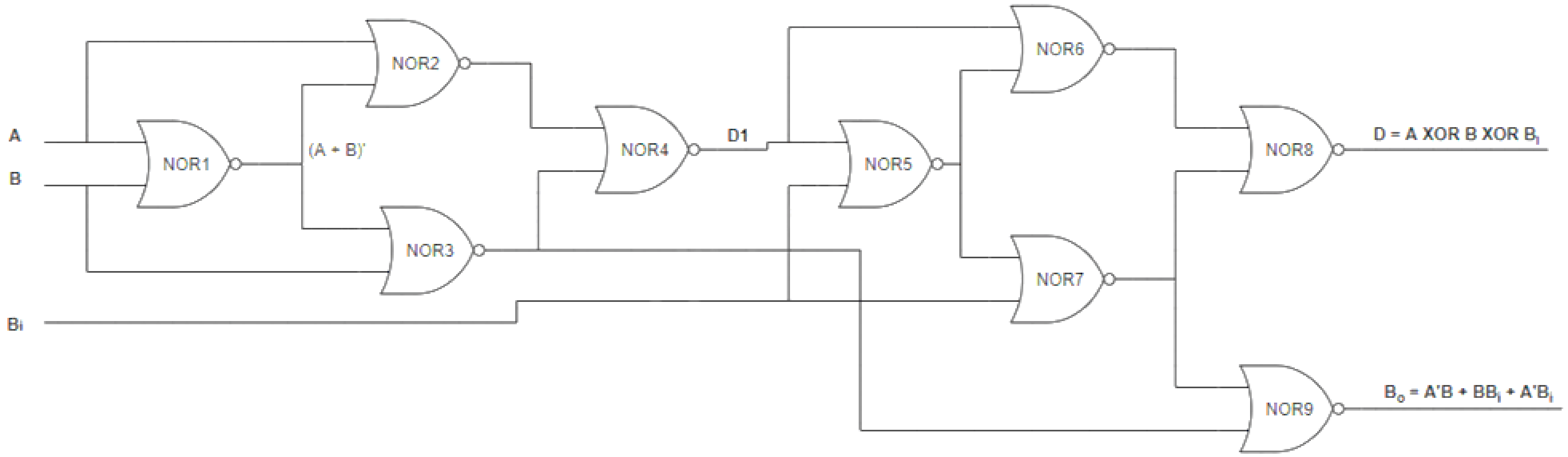


Figure 2 - Full Subtractor with NAND Gates

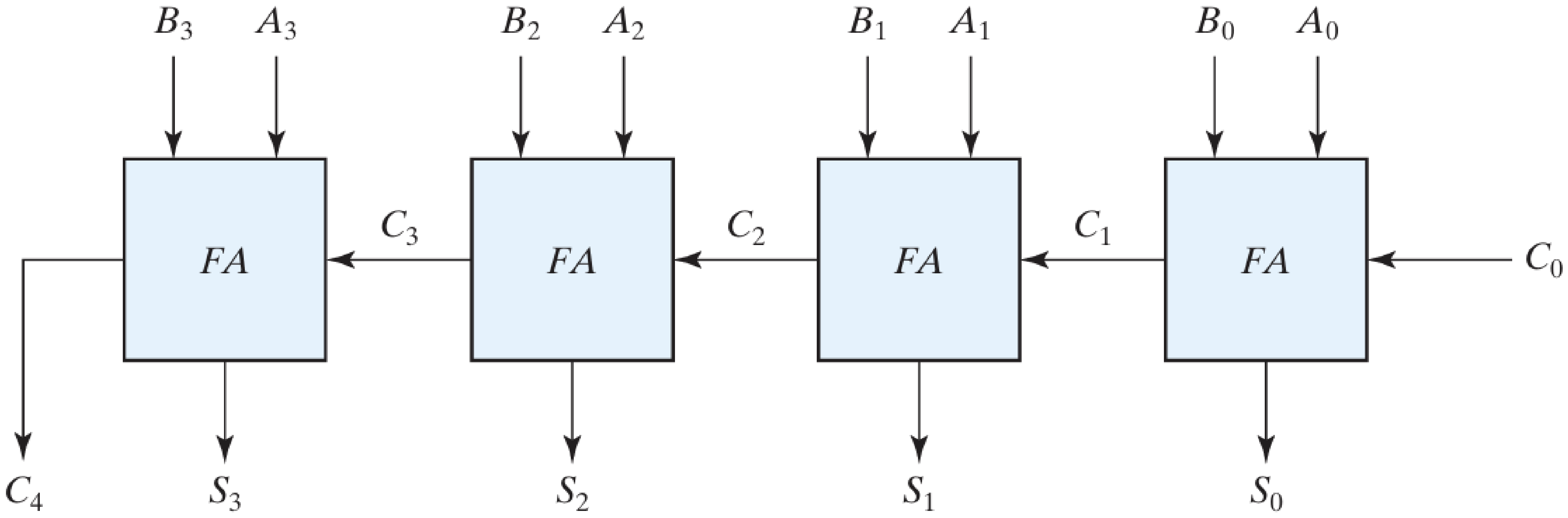
Full Subtractor Implementation using NOR gates:



Full Subtractor using NOR Gates

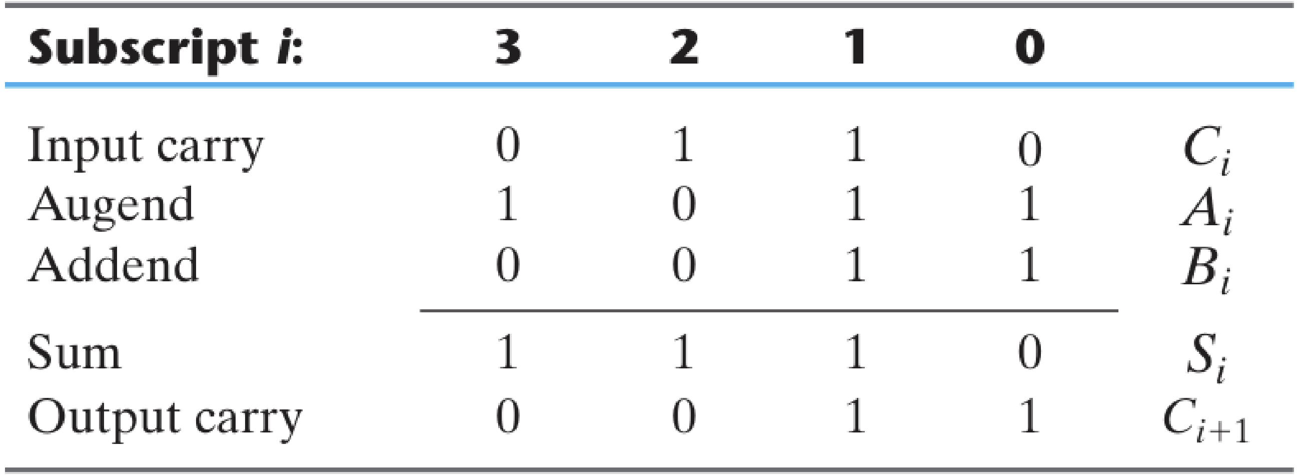
4-bit Parallel Binary Adder:

- 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 Parallel Binary Adder:

- The carries are connected in a chain through the full adders.
- The input carry to the adder is C_0, and it **ripples** through the full adders to the output carry C_4.
- The S outputs generate the required sum bits.
- An n -bit adder requires **n full adders**, with each output carry connected to the input carry of the next higher order full 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}

- The propagation delay (t_p) of a full adder is the time difference between the instant at which the inputs(A_i, B_i and C_i) are applied and the instant at which its outputs (S_i and C_{i+1}) are generated.

- The output in LSB stage is generated only after t_p seconds.

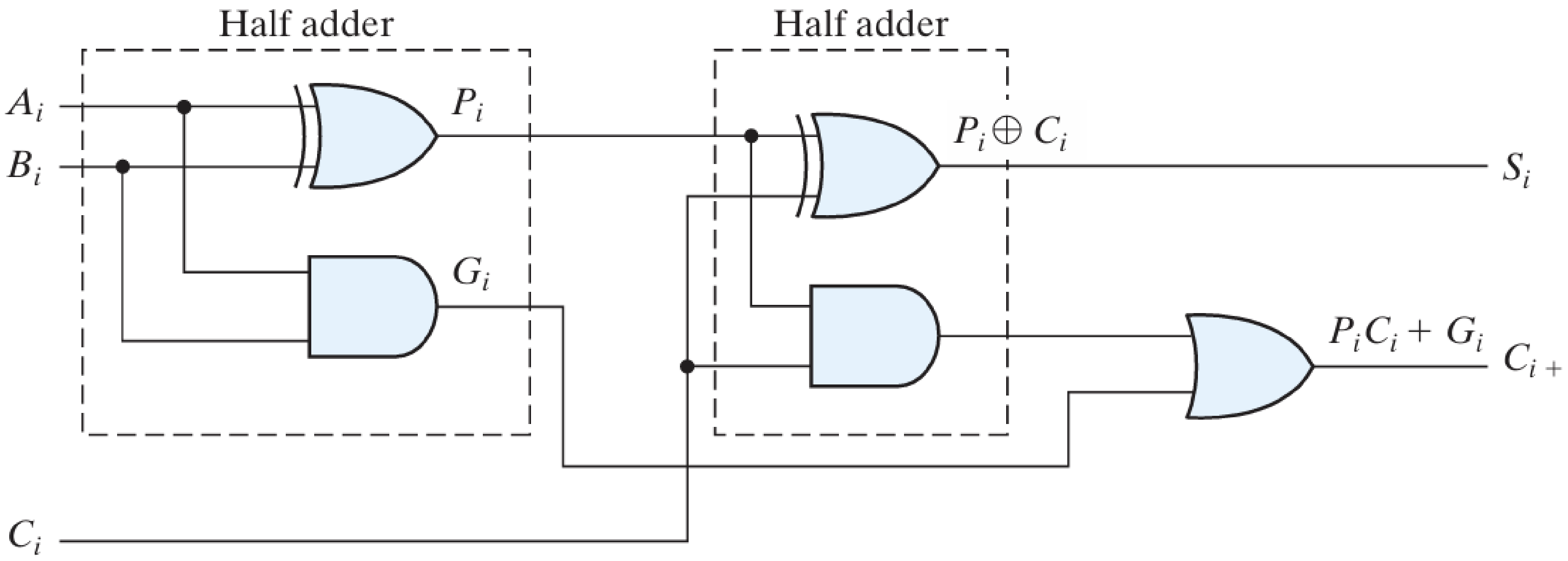
- The output in the second stage will be generated after $2t_p$ seconds from the time the inputs are applied.

- The third stage will generate outputs only after $3t_p$ seconds and the fourth stage will generate outputs only after $4t_p$ seconds.

- **Carry propagation delay affects adder speed** –
  - The time required for carry signals to propagate through the circuit limits the speed of addition, which is crucial since all arithmetic operations rely on addition.
- **Solutions to reduce carry delay** –
  - Using **faster gates** can help but has physical limitations;
  - alternatively, **advanced techniques** like **carry lookahead logic** can significantly reduce propagation time in parallel adders.

Carry LookAhead Adder (CLA):

Consider the circuit of the full adder shown



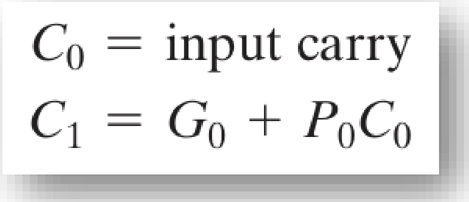
If we define two new binary variables

$$P_i = A_i \oplus B_i$$
$$G_i = A_iB_i$$

the output sum and carry can respectively be expressed as

$$S_i = P_i \oplus C_i$$
$$C_{i+1} = G_i + P_iC_i$$

- G_i is called a carry generate , and it produces a carry of 1 when both A_i and B_i are 1, regardless of the input carry C_i.
- P_i is called a carry propagate , because it determines whether a carry into stage i will propagate into stage $i + 1$
- the Boolean functions for the carry outputs of each stage and substitute the value of each C_i from the previous equations:



$$C_0 = \text{input carry}$$
$$C_1 = G_0 + P_0C_0$$

$$\boldsymbol{C_2 = G_1 + P_1C_1 = G_1 + P_1\,(G_0+P_0C_0) = G_1 + P_1G_0 + P_1P_0C_0}$$

$$\boldsymbol{C_3 = G_2 + P_2C_2 = G_2 + P_2\,G_1 + P_2P_1G_0 + P_2P_1P_0C_0}$$

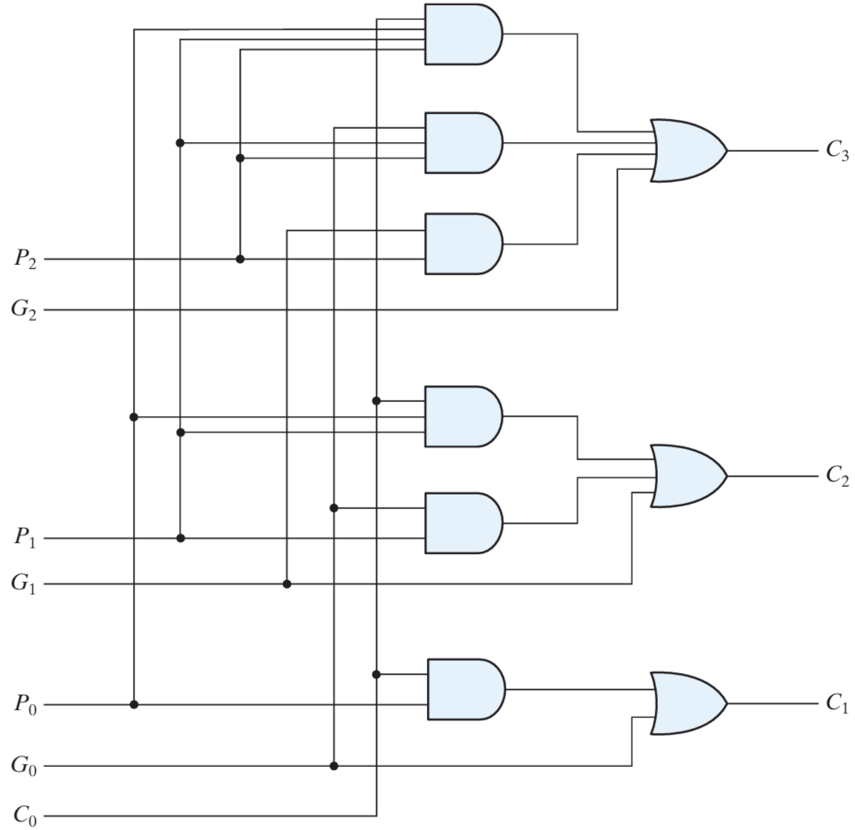


FIGURE 4.11
Logic diagram of carry lookahead generator

4-bit adder with a carry lookahead scheme

- Each sum output requires two exclusive-OR gates.
- The output of the first exclusive-OR gate generates the P_i variable, and the AND gate generates the G_i variable.
- The carries are propagated through the carry lookahead generator and applied as inputs to the second exclusive-OR gate.
- All output carries are generated after a delay through two levels of gates.
- Thus, outputs S1 through S3 have equal propagation delay times.

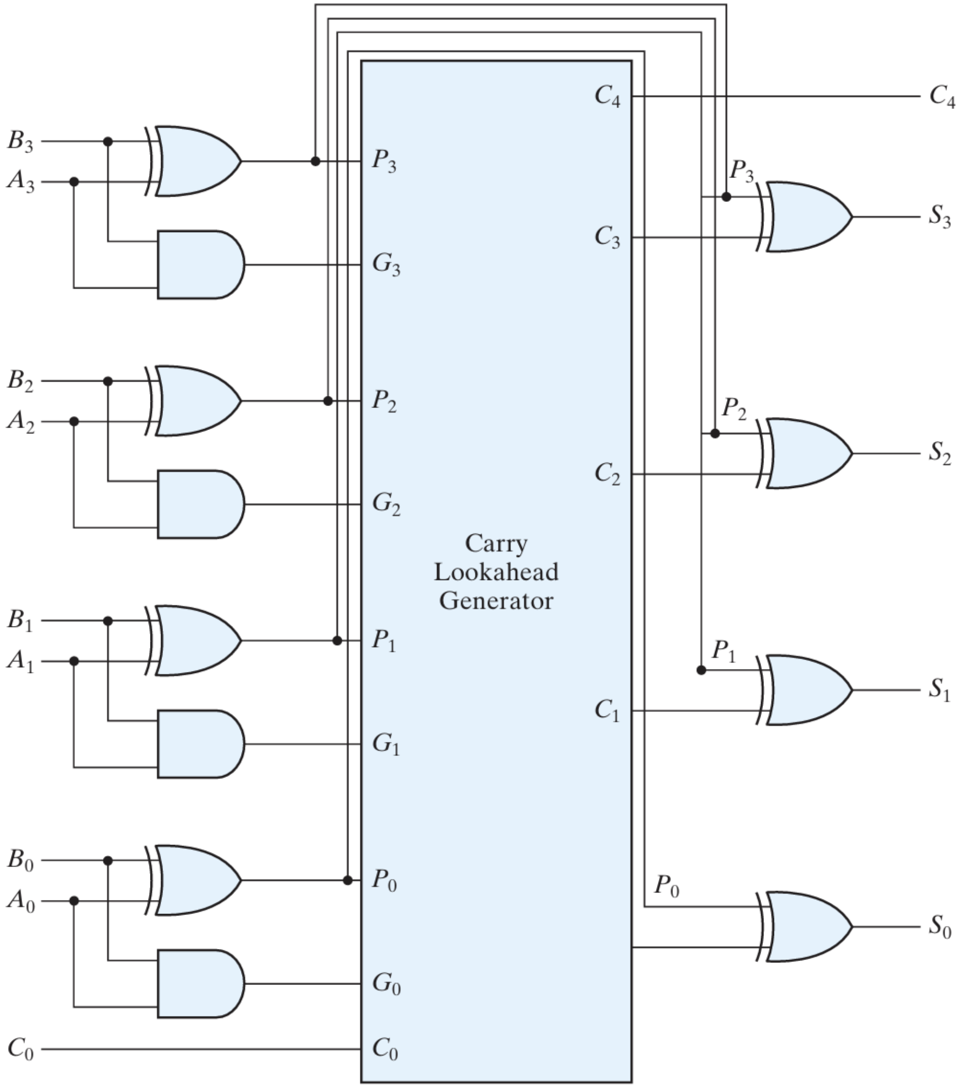
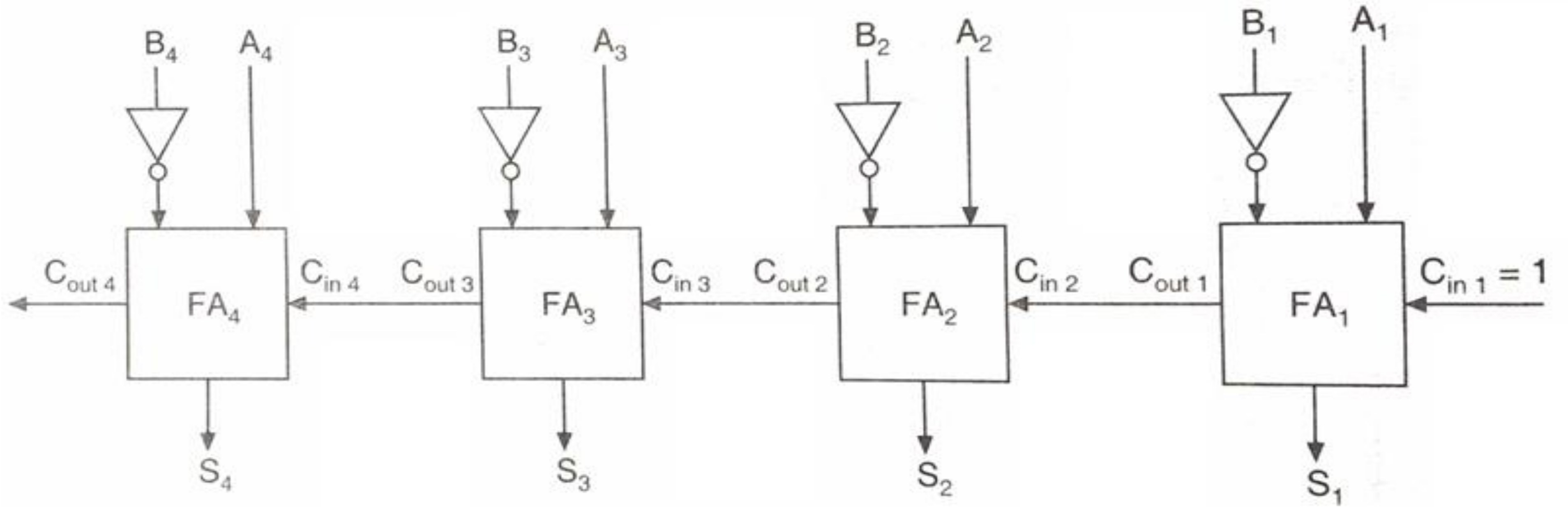


FIGURE 4.12
Four-bit adder with carry lookahead

4-bit Parallel Binary Subtractor:

- The subtraction of unsigned binary numbers can be done most conveniently by means of **complements**
- Remember that the subtraction A- B can be done by taking the 2’s complement of B and adding it to A .
- The 2’s complement can be obtained by taking the 1’s complement and adding 1 to the least significant pair of bits.
- The 1’s complement can be implemented with inverters, and a 1 can be added to the sum through the input carry.



4-bit Parallel Binary Adder /Subtractor:

- When **M = 0**, the circuit is an adder

- When **M = 1**, the circuit is a subtractor.

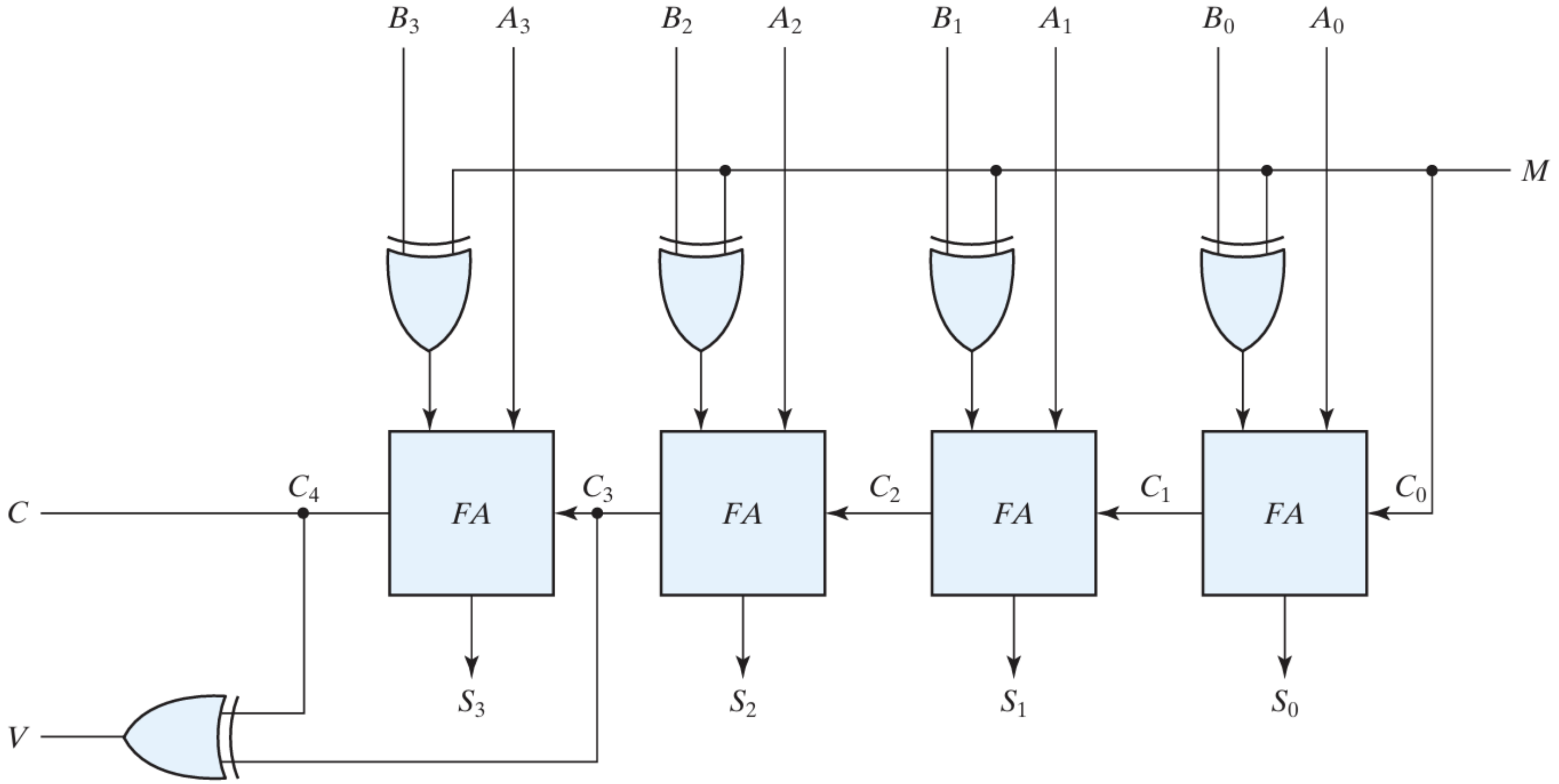


FIGURE 4.13
Four-bit adder–subtractor (with overflow detection)

The exclusive-OR with output V is for detecting an overflow.

Decimal Adder

- Computers or calculators that perform arithmetic operations directly in the **decimal number system** represent decimal numbers in **binary coded form**.
- An adder for such a computer must employ arithmetic circuits that accept coded decimal numbers and present results in the same code.
- A **minimum of 9 inputs** (4 bits per digit + carry) and **5 outputs** (4-bit result + carry) are needed for decimal addition.
- There is a wide variety of possible decimal adder circuits, depending upon the code used to represent the decimal digits. Here we examine a decimal adder for the BCD code.

Decimal Adder: BCD Adder

- A **BCD adder** is a circuit that adds two **Binary-Coded Decimal (BCD)** numbers and produces a result in BCD format.
- Since BCD uses **4-bit representation for each decimal digit (0-9)**, the sum must be **corrected** if it exceeds 9 (1001 in binary).
- Therefore, A BCD adder must include the correction logic in its internal construction.

Steps of BCD Addition

1️⃣ **Add the two BCD numbers** using a **4-bit binary adder**.

2️⃣ **Check if the sum is greater than 9 (1001 in binary) or if a carry is generated.**

3️⃣ **If sum > 9 or carry = 1, add 6 (0110 in binary)** to adjust the result to a valid BCD digit.

4️⃣ **If no correction is needed, the result remains unchanged.**

Decimal Adder: BCD Adder → Example

A (BCD)	B (BCD)	Binary Sum	Sum > 9 or Carry?	Correction (Add 6)	Carry Out	Final BCD Sum	Decimal Equivalent Sum
0000 (0)	0000 (0)	0000 (0)	No	No	0	0000 (0)	0
0001 (1)	0001 (1)	0010 (2)	No	No	0	0010 (2)	2
0100 (4)	0101 (5)	1001 (9)	No	No	0	1001 (9)	9
0101 (5)	0101 (5)	1010 (10)	Yes	Yes (Add 6)	1	0000 (0)	10
0110 (6)	0111 (7)	1101 (13)	Yes	Yes (Add 6)	1	0011 (3)	13
1000 (8)	1001 (9)	10001 (17)	Yes	Yes (Add 6)	1	0111 (7)	17

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 ??

Uncorrected Sum

Corrected Sum

BCD Adder

- When the binary sum is **=<1001**, the corresponding BCD number is identical, and therefore no conversion is needed.
- When the binary sum is **>1001**, we obtain an invalid 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.
- From table in previous slide, correction is needed when the binary sum has an output carry K=1.
- The other six combinations from 1010 through 1111 that need a correction have a 1 in position Z8. To distinguish them from binary 1000 and 1001, which also have a 1 in position Z8, we specify further that either Z4 or Z2 must have a 1.
- The condition for a correction and an output carry can be expressed by the Boolean function

$$C = K + Z_8Z_4 + Z_8Z_2$$

- When C=1, it is necessary to **add 0110** to the binary sum and provide an output carry for the next stage.

- The two decimal digits, together with the input carry, are first added in the top four-bit adder to produce the binary sum.
- When the output carry is equal to 0, nothing is added to the binary sum.
- When it is **equal to 1, binary 0110** is added to the binary sum through the bottom four-bit adder.
- The output carry generated from the bottom adder can be ignored, since it supplies information already available at the output carry terminal.

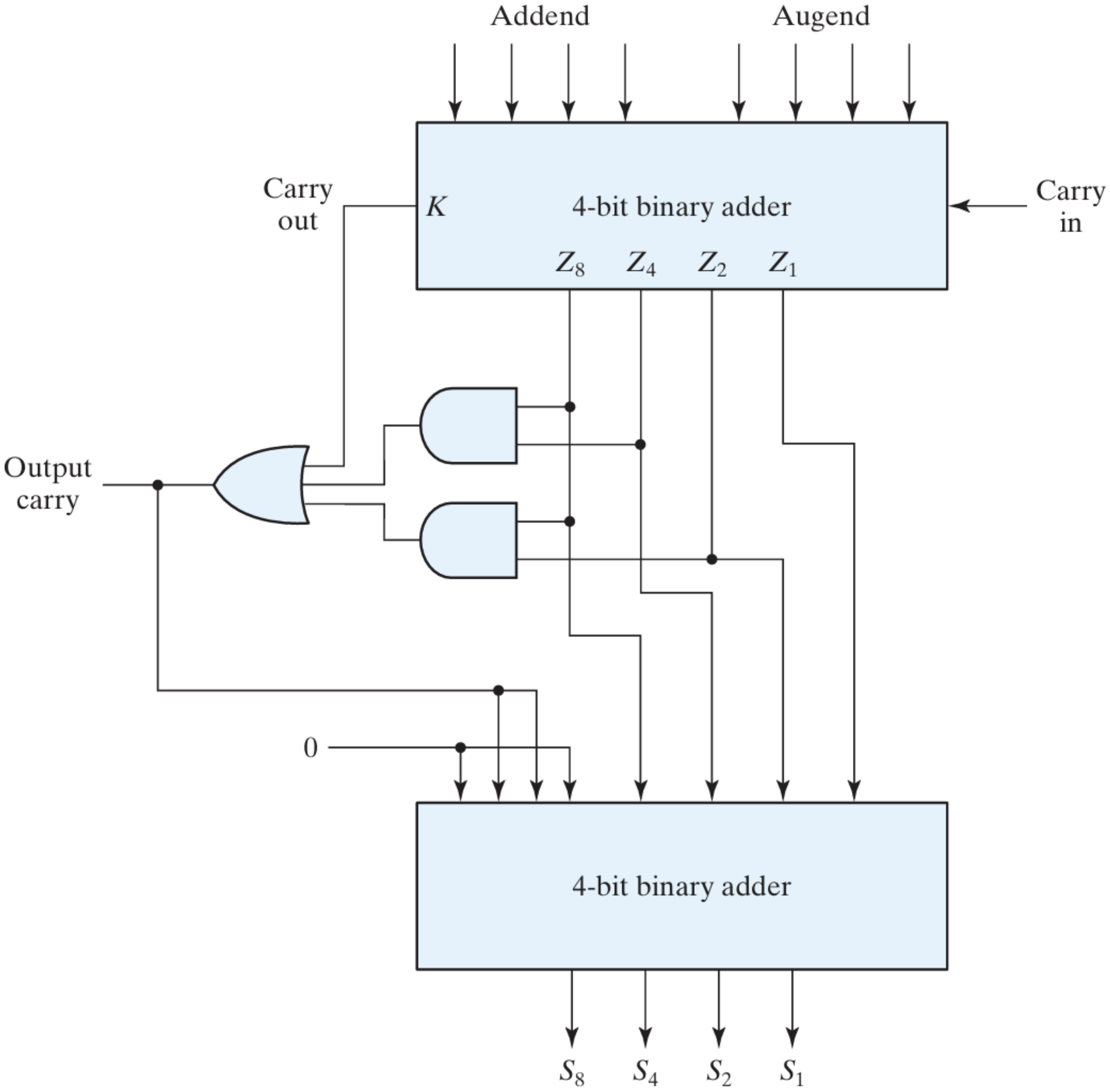


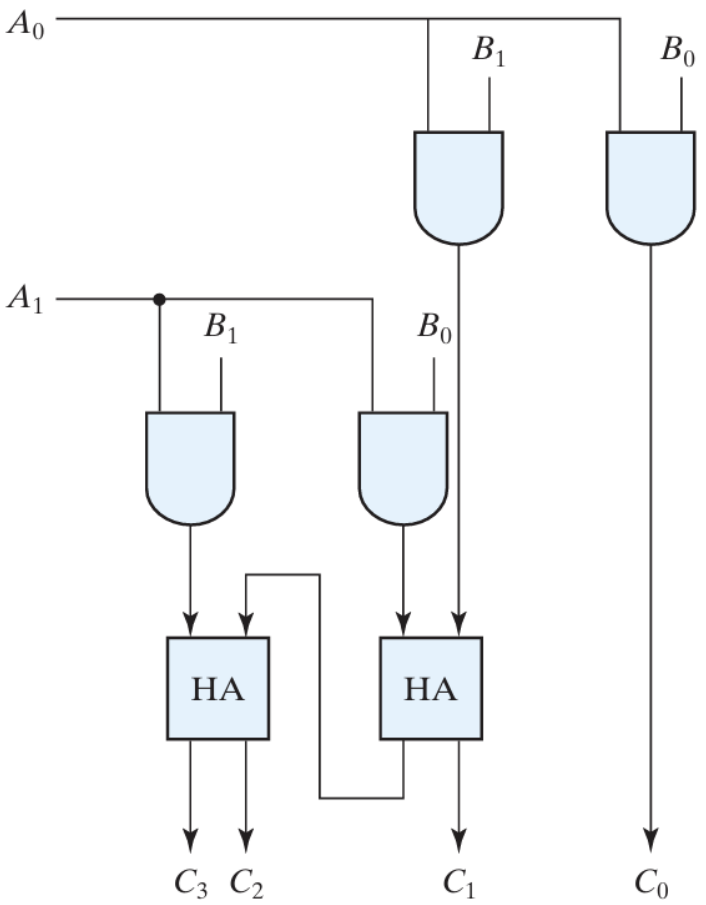
FIGURE 4.14
Block diagram of a BCD adder

Binary Multiplier

- Multiplication of binary numbers is performed in the same way as multiplication of decimal numbers.
- The multiplicand is multiplied by each bit of the multiplier, starting from the least significant bit.
- Each such multiplication forms a partial product.
- Successive partial products are shifted one position to the left.
- The final product is obtained from the sum of the partial products.

- The multiplication of two bits such as A_0 and B_0 produces a 1 if both bits are 1; otherwise, it produces a 0.
- This is identical to an AND operation.
- Therefore, the partial product can be implemented with AND gates
- The second partial product is formed by multiplying B_1B_0 by A_1 and shifting one position to the left.
- The two partial products are added with two half-adder (HA) circuits.

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



- A combinational circuit **binary multiplier** with **more bits** can be constructed in a similar fashion.
- A bit of the multiplier is **ANDed** with each bit of the multiplicand in as many levels as there are bits in the multiplier.
- The binary output in each level of AND gates is added with the partial product of the previous level to form a new partial product.
- The last level produces the product.
- 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.
- E.g.
- Let the multiplicand be represented by $B_3B_2B_1B_0$ and the multiplier by $A_2A_1A_0$.
- Since **K = 4** and **J = 3**,
    - we need **12 AND** gates and **two 4-bit adders** to produce a product of **seven bits**.

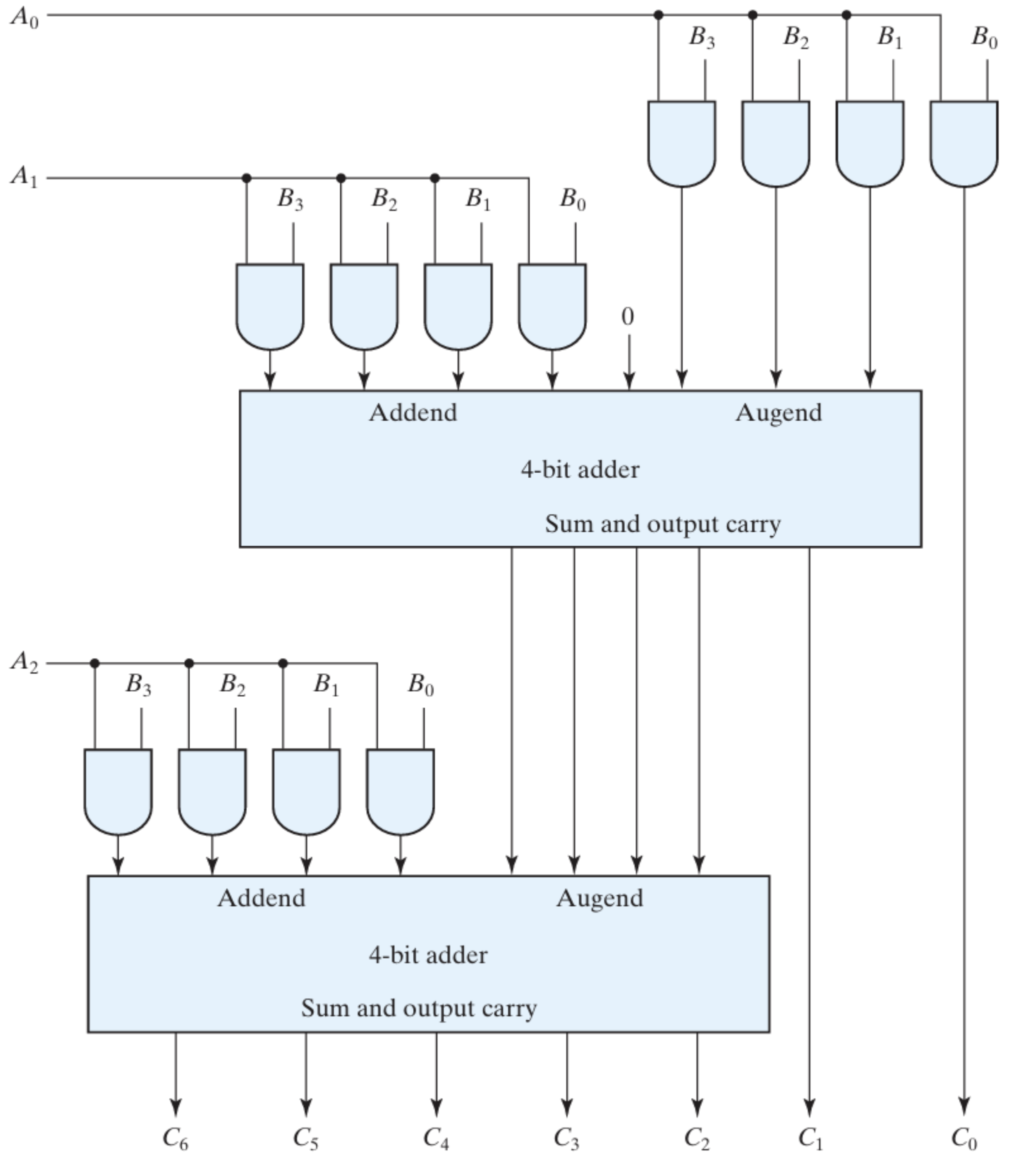
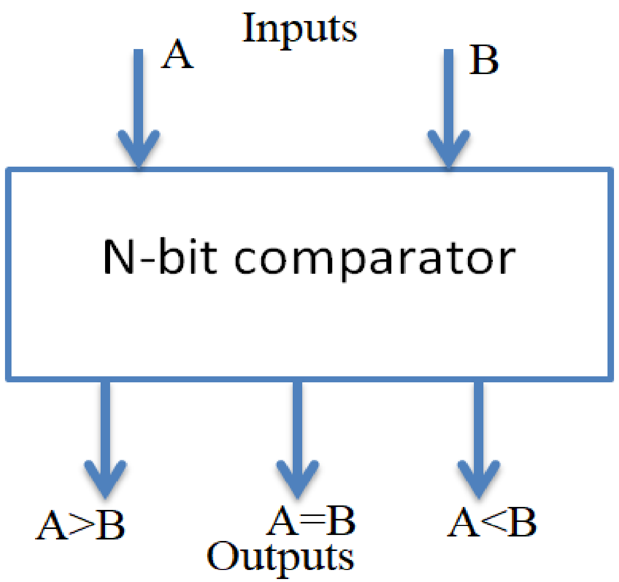


FIGURE 4.16
Four-bit by three-bit binary multiplier

Comparators

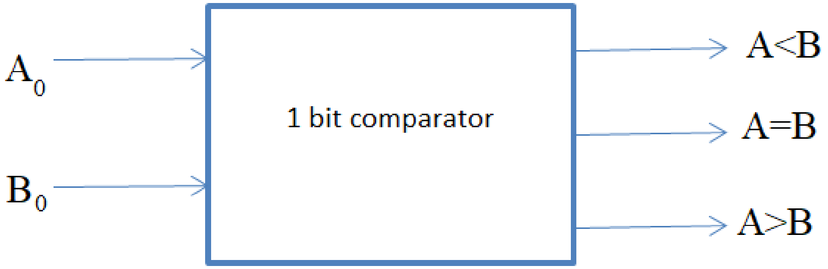
- Comparator is a combinational logic circuit that compares the magnitudes of two binary quantities(Numbers) to determine which one number has less, equal or greater magnitude.
- Three binary variables are used to indicate the outcome of the comparison as: $A > B$, $A < B$, or $A = B$.
- Block Diagram of Comparator:



1 bit Magnitude comparator

- One bit magnitude comparator used to compare two 1-bit numbers.
- Let the 1-bit number be A=A_0 and B=B_0

Block Diagram



A_0

B_0

1 bit comparator

A<B

A=B

A>B

Truth Table

Inputs		Outputs		
A_0	B_0	A<B L	A=B E	A>B G
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

Boolean Expression:

$$L = \bar{A}_0 B_0$$

$$E = A_0 B_0 + \bar{A}_0 \bar{B}_0 = A \odot B$$

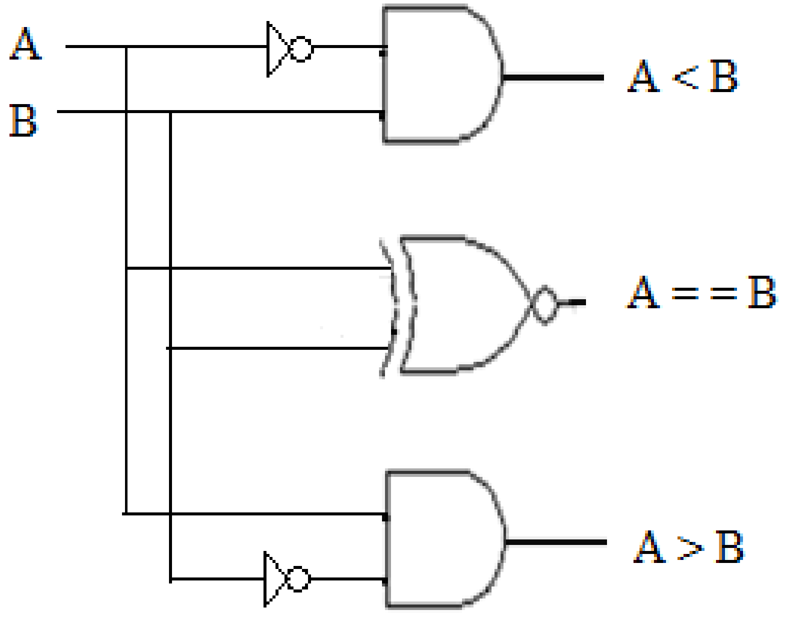
$$G = A_0 \bar{B}_0$$

1 bit Magnitude comparator

Boolean Expression:

$$L = \overline{A}_0 B_0$$
$$E = A_0 B_0 + \overline{A}_0 \overline{B}_0 = A \odot B$$
$$G = A_0 \overline{B}_0$$

Logic Diagram:

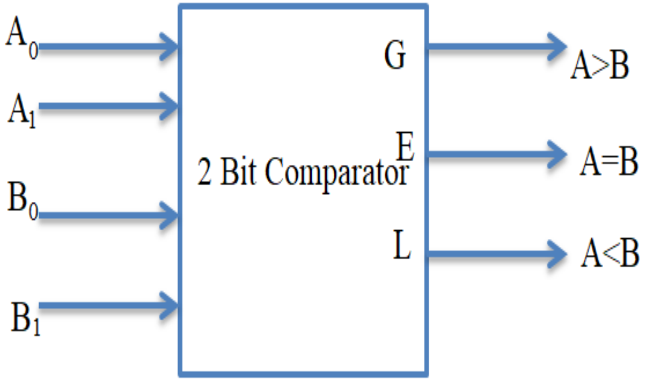


1-bit magnitude comparator

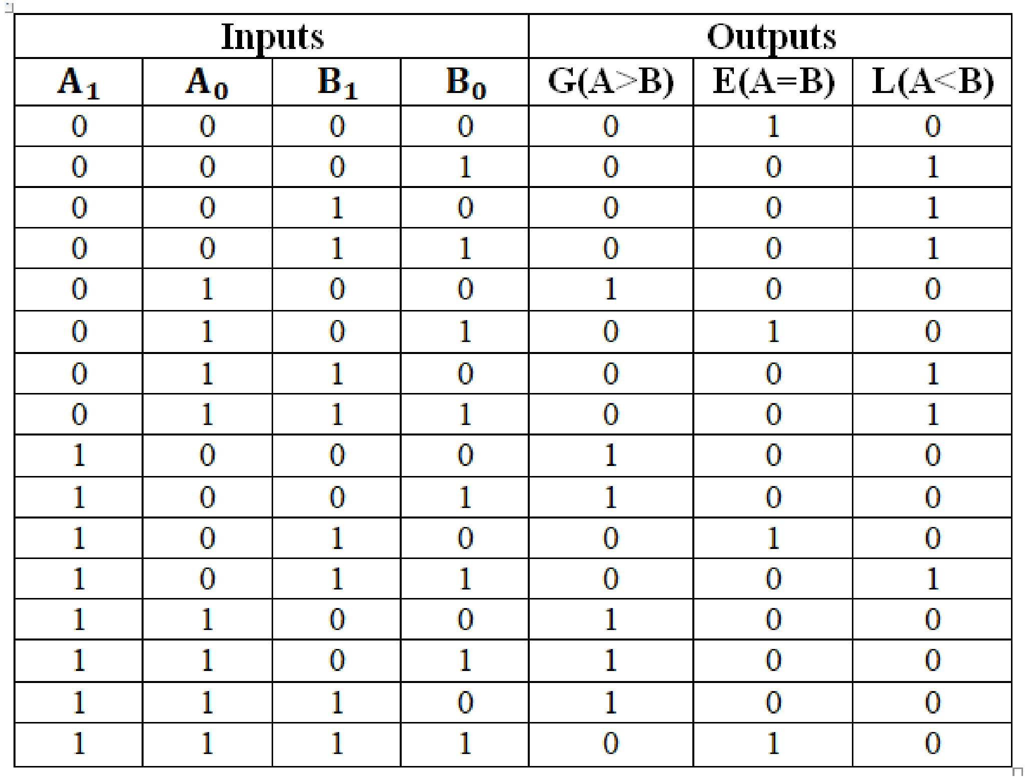
2- bit Comparator

In 2-bit comparator there are two 2-bit inputs and three outputs:

Block Diagram



Truth table



Inputs				Outputs		
A_1	A_0	B_1	B_0	G(A>B)	E(A=B)	L(A<B)
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

Boolean Expressions:

$G = \sum m(4,8,9,12,13,14)$,
$E = \sum m(0,5,10,15)$,
$L = \sum m(1,2,3,6,7,11)$,

2- bit Comparator

$$G = \sum m(4,8,9,12,13,14)\ ,\ E = \sum m(0,5,10,15)\ ,\ L = \sum m(1,2,3,6,7,11)\ ,$$

For:A>B

$A_1\ A_0$ B_1B_0	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

For: A=B

$A_1\ A_0$ B_1B_0	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

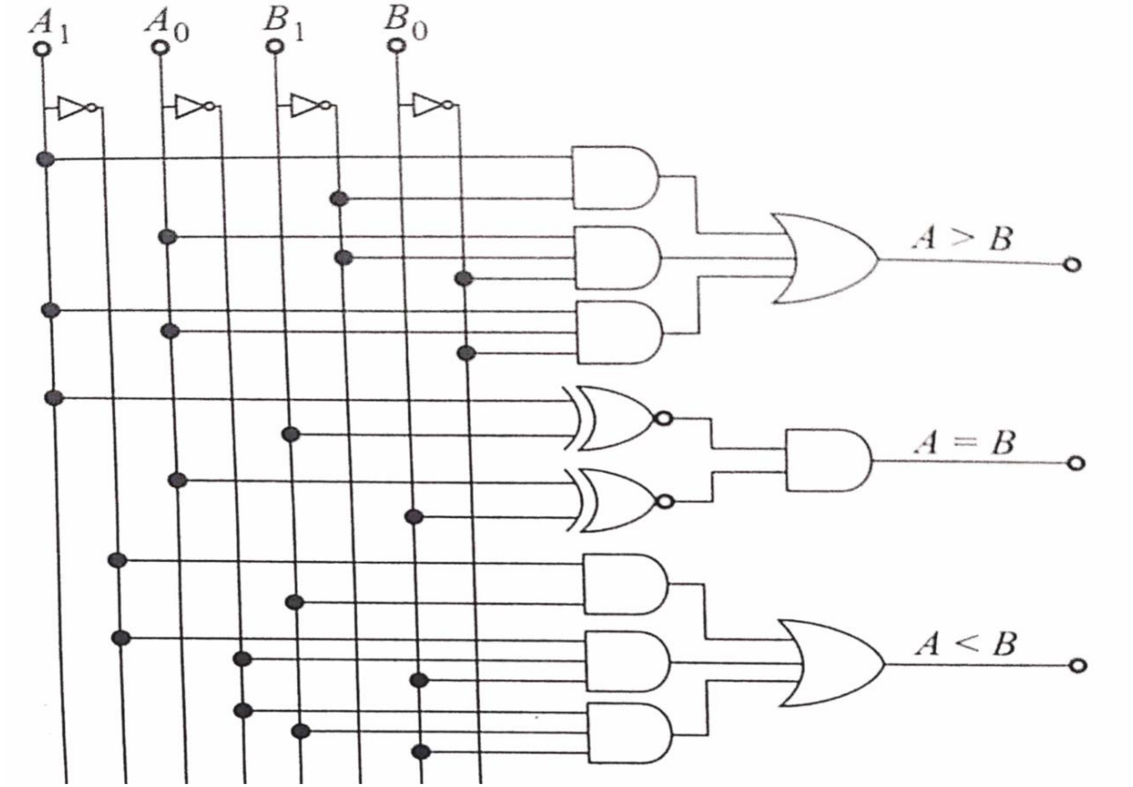
For: A<B

$A_1\ A_0$ B_1B_0	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

Boolean expressions:

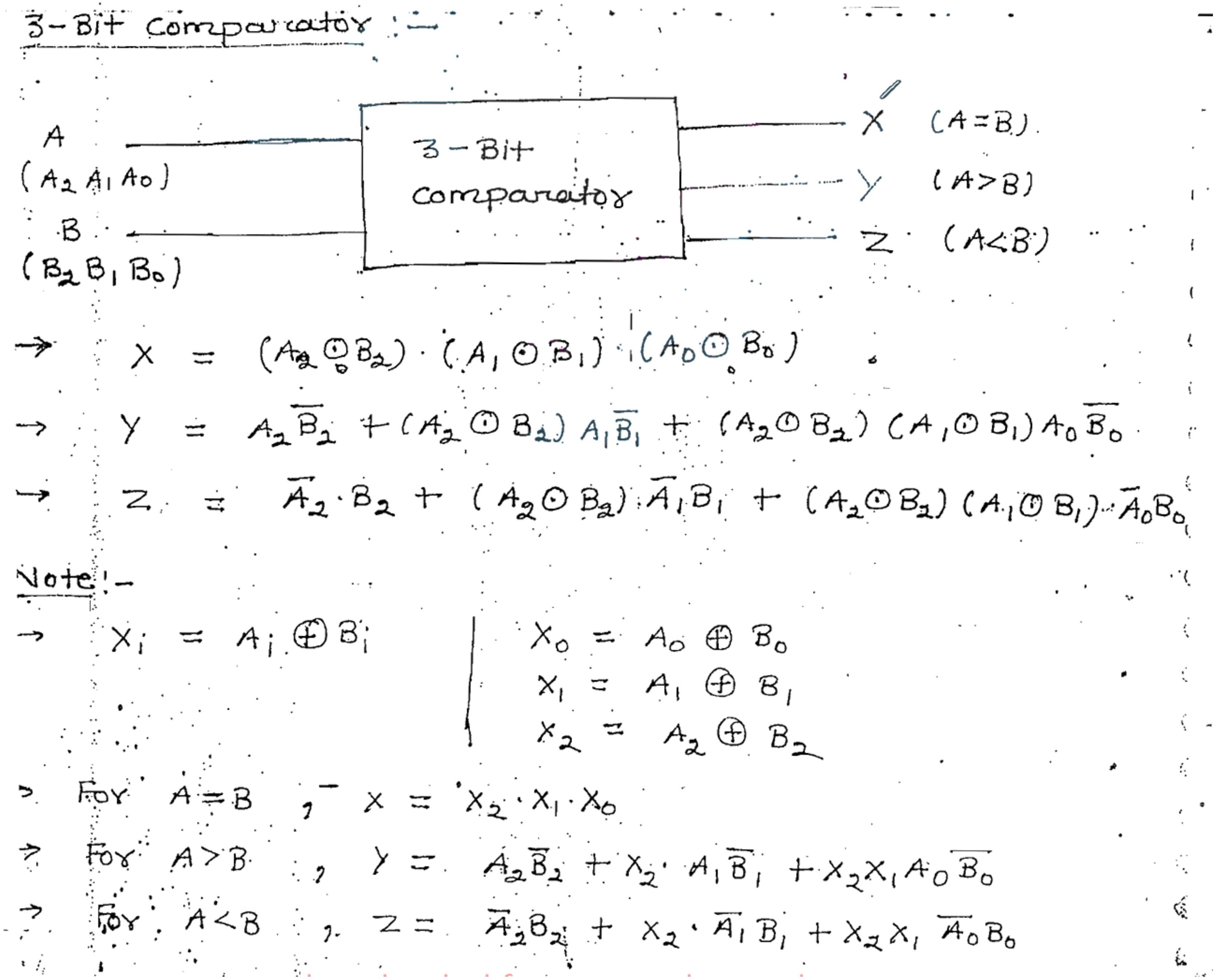
Logic Diagram:

$$G(A > B) = A_1\overline{B}_1 + A_0\overline{B}_1\overline{B}_0 + A_1A_0\overline{B}_0$$
$$E(A = B) = (A_0 \odot B_0)(A_1 \odot B_1)$$
$$L(A < B) = \overline{A}_1B_1 + \overline{A}_1\overline{A}_0B_0 + \overline{A}_0B_1B_0$$



- The circuit for comparing two n - bit numbers has 2^{2n} entries in the truth table and becomes too cumbersome, even with n = 3.

- Digital functions that possess an inherent well-defined regularity can usually be designed by means of an algorithm—a procedure which specifies a finite set of steps that, if followed, give the solution to a problem.



3-Bit Comparator :-

A $(A_2 A_1 A_0)$

B $(B_2 B_1 B_0)$

3-Bit comparator

X (A=B)

Y (A>B)

Z (A<B)

→ $X = (A_2 \odot B_2) \cdot (A_1 \odot B_1) \cdot (A_0 \odot B_0)$

→ $Y = A_2 \overline{B_2} + (A_2 \odot B_2) A_1 \overline{B_1} + (A_2 \odot B_2)(A_1 \odot B_1) A_0 \overline{B_0}$

→ $Z = \overline{A_2} \cdot B_2 + (A_2 \odot B_2) \overline{A_1} B_1 + (A_2 \odot B_2)(A_1 \odot B_1) \cdot \overline{A_0} B_0$

Note:-

→ $x_i = A_i \oplus B_i$

$x_0 = A_0 \oplus B_0$

$x_1 = A_1 \oplus B_1$

$x_2 = A_2 \oplus B_2$

→ For A = B, $X = x_2 \cdot x_1 \cdot x_0$

→ For A > B, $Y = A_2 \overline{B_2} + x_2 \cdot A_1 \overline{B_1} + x_2 x_1 A_0 \overline{B_0}$

→ For A < B, $Z = \overline{A_2} B_2 + x_2 \cdot \overline{A_1} B_1 + x_2 x_1 \overline{A_0} B_0$

Applications of Comparators

- These are used in control applications in which the binary numbers representing physical variables such as temperature, position, etc. are compared with a reference value.

- Process controllers (ON/OFF, self-tune, and manual tune)

- Servo-motor control
- We may have an IC 7485 which is a 4-bit comparator