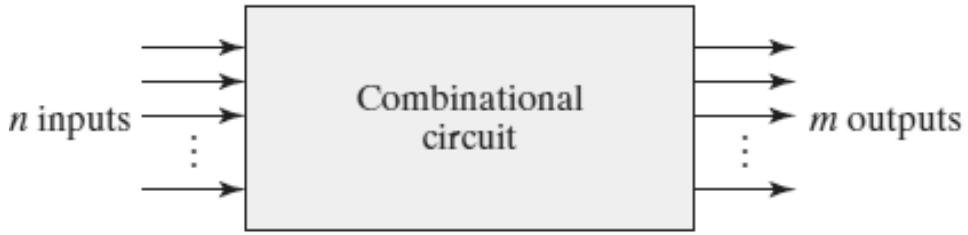
Combinational Circuits

Logic Circuits

- Logic circuits for digital systems may be
  - combinational
    - A combinational circuit consists of logic gates whose outputs at any time are determined from only the present combination of inputs.
  - Sequential
    - sequential circuits employ storage elements in addition to logic gates.
    - Their outputs are a function of the inputs and the state of the storage elements.
    - The outputs of a sequential circuit depend not only on present values of inputs, but also on past inputs,
    - The circuit behavior must be specified by a time sequence of inputs and internal states.

Combinational Circuit

- A combinational circuit consists of an interconnection of logic gates.
- Combinational logic gates react to the values of the signals at their inputs and produce the value of the output signal, transforming binary information from the given input data to a required output data.



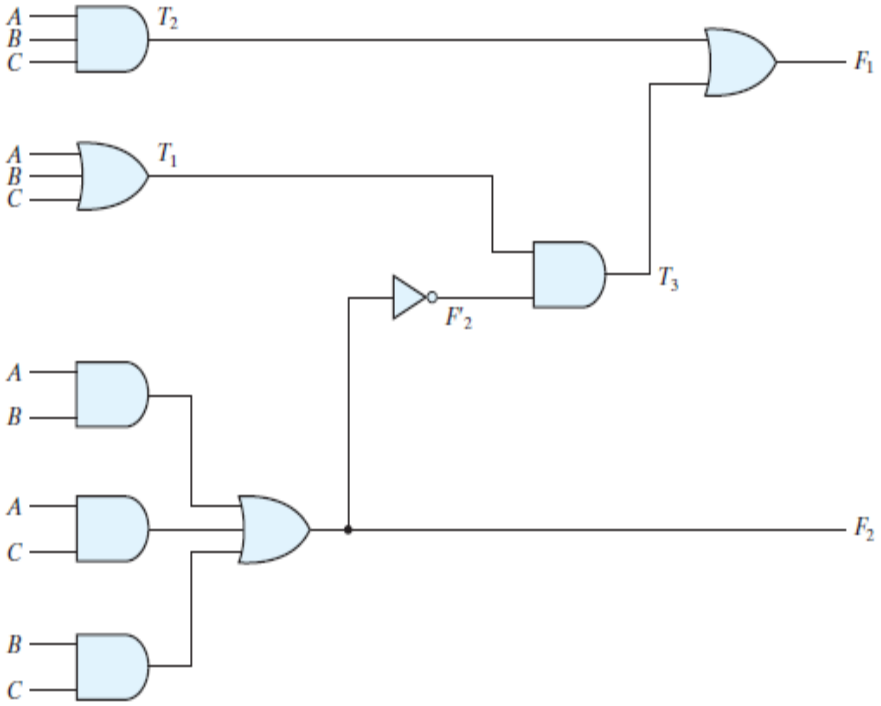
  - The *n input* binary variables come from an external source;
  - the *m output variables are produced by* the internal combinational logic circuit and go to an external destination.
- For *n input variables, there are* 2^n *possible combinations of the binary inputs.*
- A combinational circuit can be specified with a truth table that lists the output values for each combination of input variables.
- 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.*

Analysis

- **Goal :** The analysis of a combinational circuit requires the determination of the function that the circuit implements.
- The first step in the analysis is to make sure that the given circuit is combinational and not sequential.
  - The diagram of a combinational circuit has logic gates with no feedback paths(connection from the output of one gate to the input of a second gate whose output forms part of the input to the first gate) or memory elements .
- 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— but with meaningful names. 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.

Analysis

- The circuit has
  - three binary inputs— *A , B , C*
  - *two binary* outputs— *F1 ,F2.*
  - *The outputs of various gates are labeled with intermediate symbols.*
- The outputs of gates that are a function only of input variables are *T1,T2 &F2*



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

- outputs of gates that are a function of already defined symbols:

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

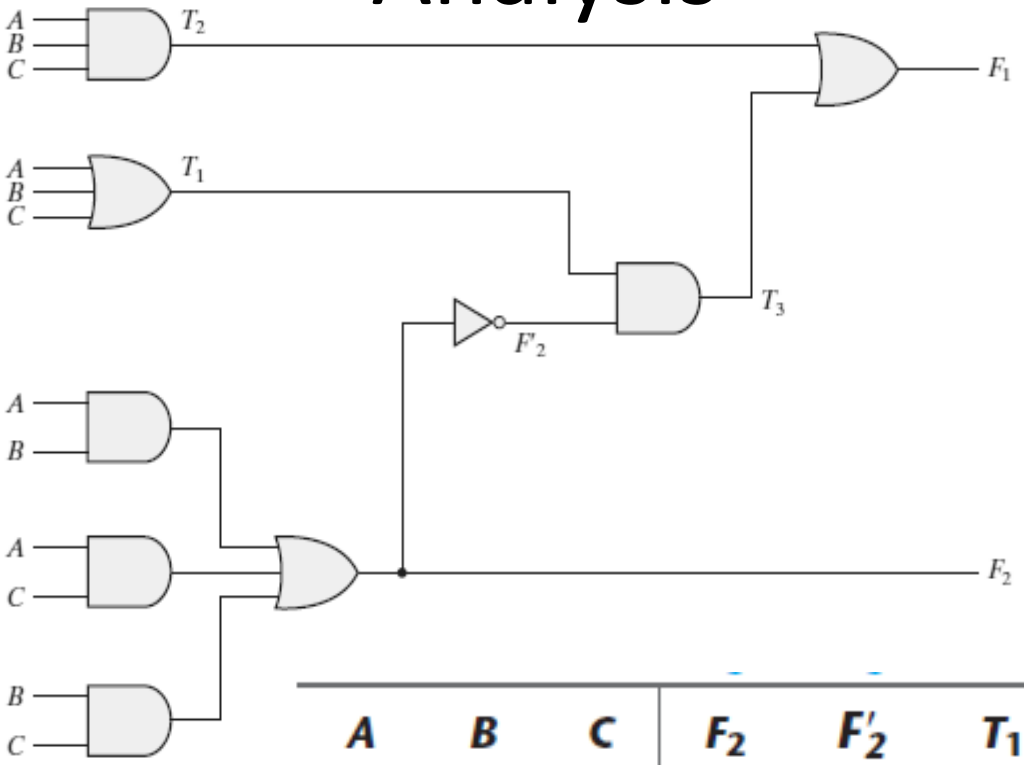
•To obtain F1 as a function of A , B , and C , we form a series of substitutions as follows:

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

Analysis

- The derivation of the truth table for a circuit is a straightforward process once the output Boolean functions are known.
- To obtain the truth table directly from the 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 which are a function of the input variables only.
  - 4. Proceed to obtain the truth table for the outputs of those gates which are a function of previously defined values until the columns for all outputs are determined.

Analysis



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

DESIGN PROCEDURE

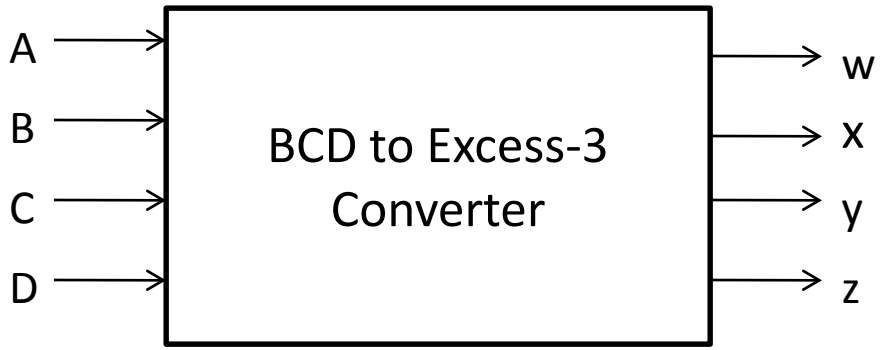
- The design of combinational circuits starts from the specification of the design objective and culminates in a logic circuit diagram or a set of Boolean functions from which the logic diagram can be obtained.
- The procedure involves the following steps:
  - 1. From the specifications of the circuit, determine the required number of inputs and outputs and assign a symbol to each(draw a **block diagram**).
  - 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

- The availability of a large variety of codes for the same discrete elements of information results in the use of different codes by different digital systems.
- It is sometimes necessary to use the output of one system as the input to another.
  - A conversion circuit must be inserted between the two systems if each uses different codes for the same information.
  - Thus, a code converter is a circuit that makes the two systems compatible even though each uses a different binary code.
- To convert from binary code A to binary code B, the input lines must supply the bit combination of elements as specified by code A and the output lines must generate the corresponding bit combination of code B.
- A combinational circuit performs this transformation by means of logic gates

Binary Coded Decimal (BCD) to Excess-3 Code

- Block Diagram

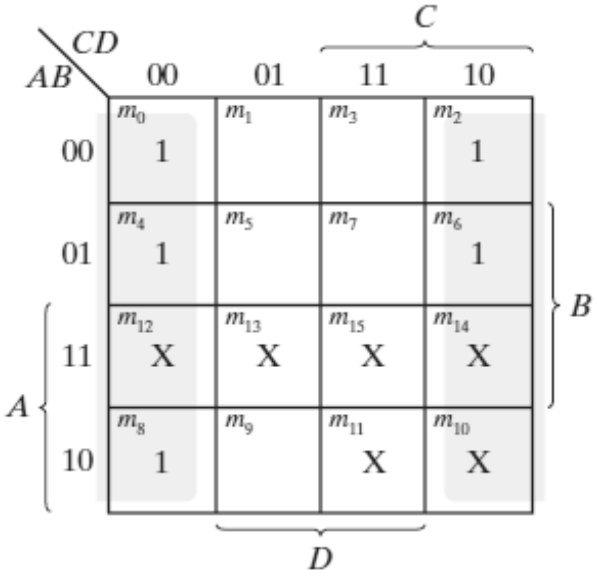


- Truth Table

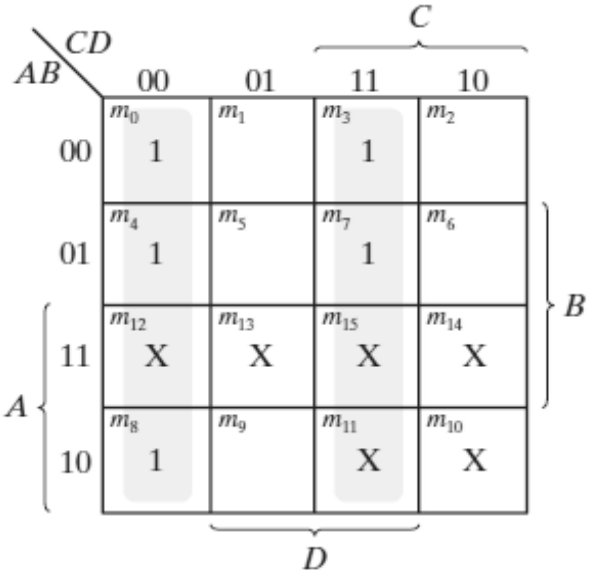
BCD Code	Excess-3 Code
A B C D	w x y z
0 0 0 0	0 0 1 1
0 0 0 1	0 1 0 0
0 0 1 0	0 1 0 1
0 0 1 1	0 1 1 0
0 1 0 0	0 1 1 1
0 1 0 1	1 0 0 0
0 1 1 0	1 0 0 1
0 1 1 1	1 0 1 0
1 0 0 0	1 0 1 1
1 0 0 1	1 1 0 0
1 0 1 0	X X X X
1 0 1 1	X X X X
1 1 0 0	X X X X
1 1 0 1	X X X X
1 1 1 0	X X X X
1 1 1 1	X X X X

Eg: Binary Coded Decimal (BCD) to Excess-3 Code

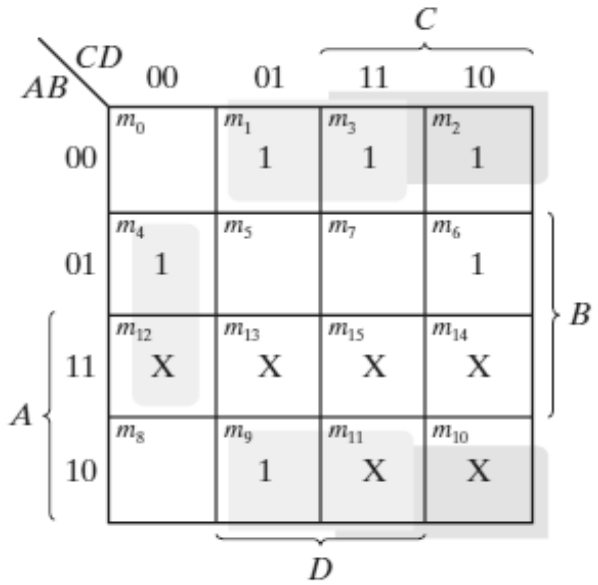
- Simplification:



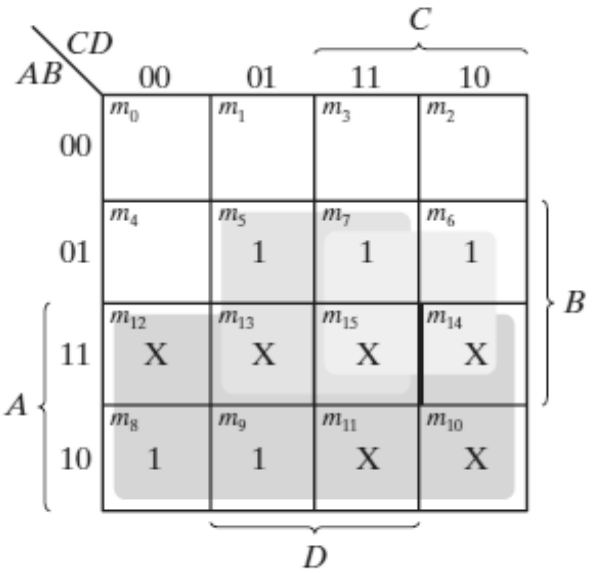
$z = D'$



$y = CD + C'D'$



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



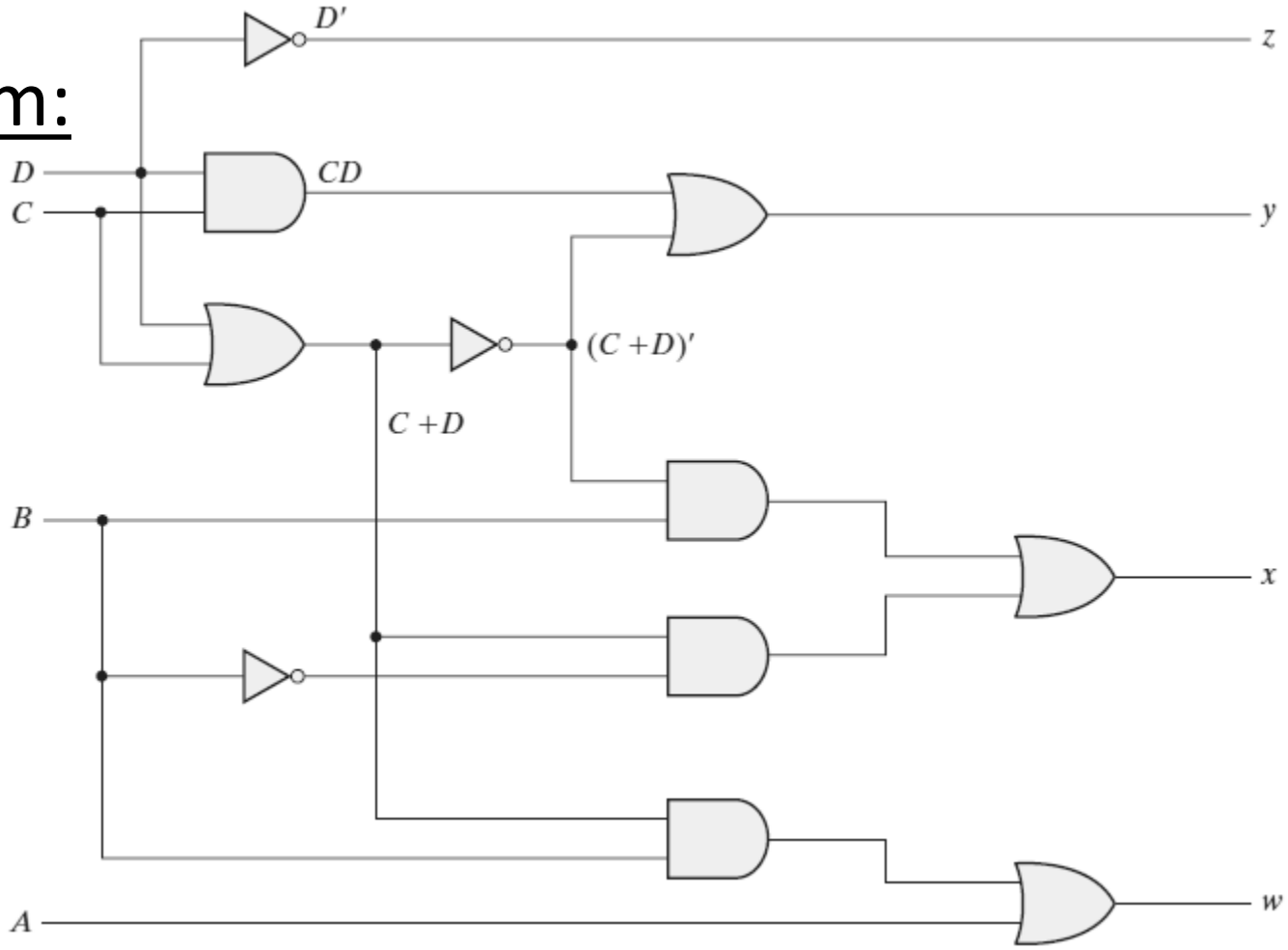
$w = A + BC + BD$

Eg: Binary Coded Decimal (BCD) to Excess-3 Code

- Simplified Boolean Functions:

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

- Circuit Diagram:

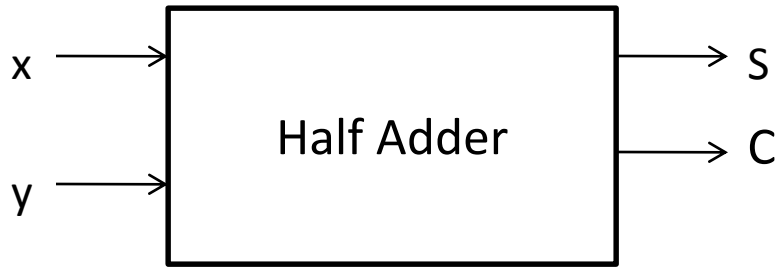


Standard Combinational Circuits

- There are several combinational circuits that are employed extensively in the design
- of digital systems.
- These circuits are available in integrated circuits and are classified as standard components.
- They perform specific digital functions commonly needed in the design of digital systems.
- The most important standard combinational circuits:
  - Adders
  - Subtractors
  - Comparators
  - Decoders
  - Encoders
  - Multiplexers
  - Demultiplexer
- These components are available in integrated circuits as medium-scale integration (MSI) circuits.
- They are also used as *standard cells in complex very large-scale* integrated (VLSI) circuits such as application-specific integrated circuits (ASICs).

Half Adder

- A combinational circuit that performs the addition of two bits is called a *half adder.*
- This circuit needs two binary inputs and two binary outputs.
  - The input variables designate the augend and addend bits
  - the output variables produce the sum and carry
- Block Diagram:



- Truth Table:

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

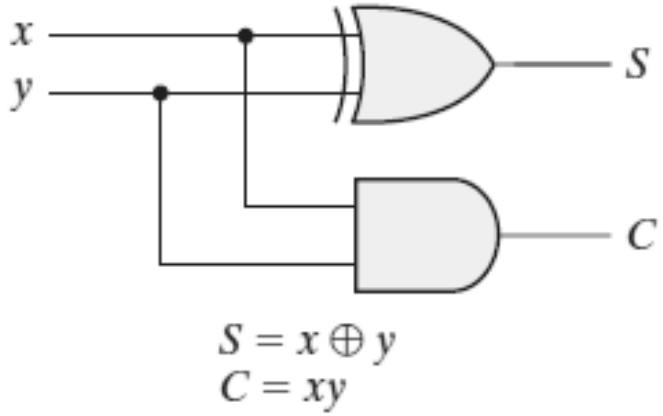
Half Adder

- Simplified Boolean Functions:

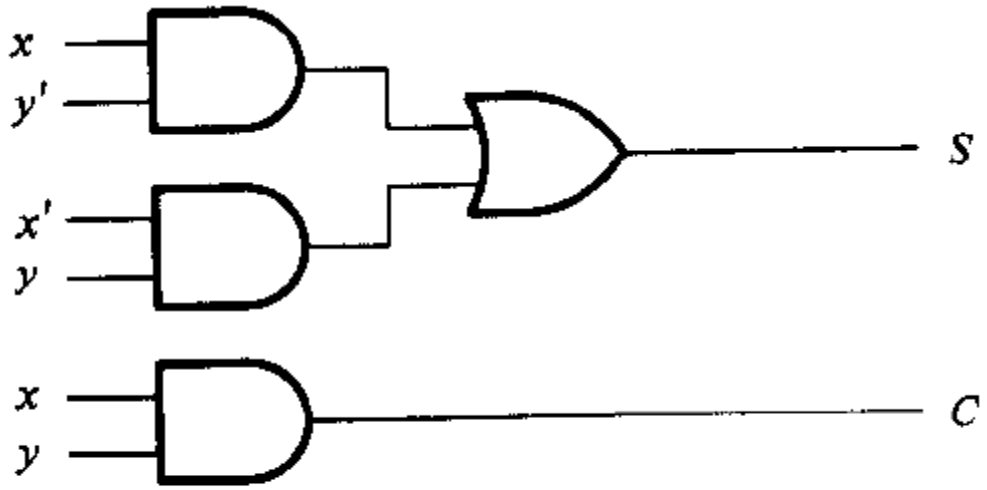
  $S = x'y + xy' = x \oplus y$

  $C = xy$

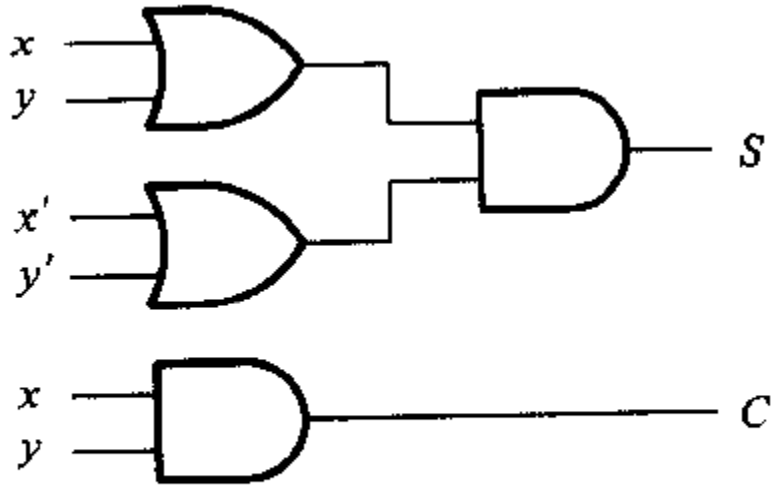
- Circuit Diagram:



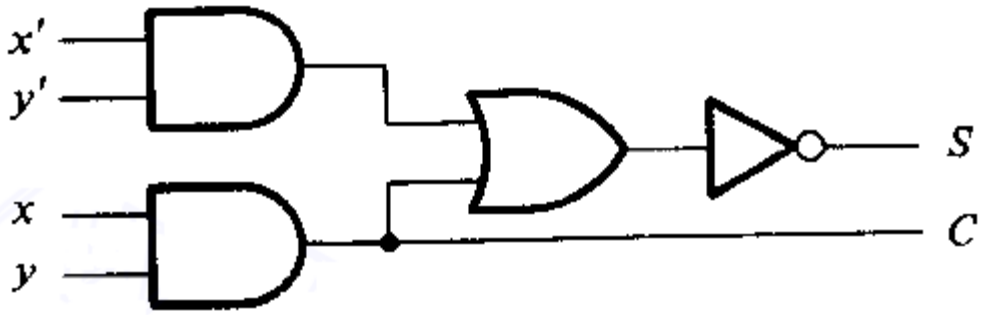
Various Implementations of Half Adder



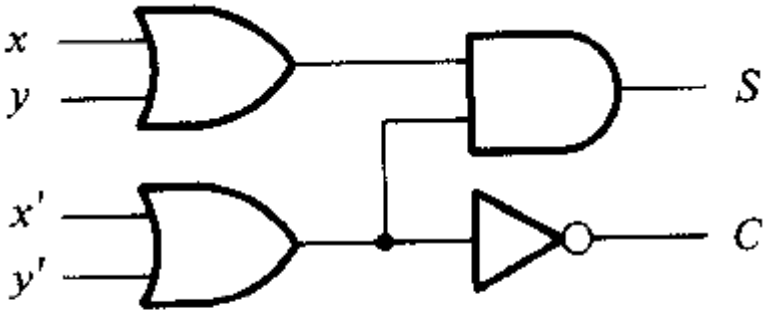
$S = xy' + x'y$
$C = xy$



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



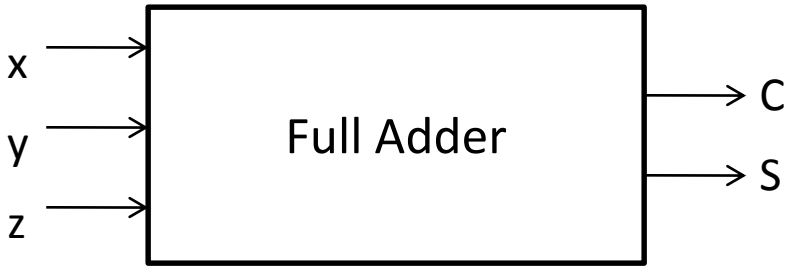
$S = (C + x'y')'$
$C = xy$



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

Full Adder

- A full adder is a combinational circuit that forms the arithmetic sum of three bits.
- It consists of three inputs and two outputs.
  - Two of the input variables designate the augend and addend bits
  - The third input *represents the carry from* the previous lower significant position.
  - The output variables produce the sum and carry
- Block Diagram:

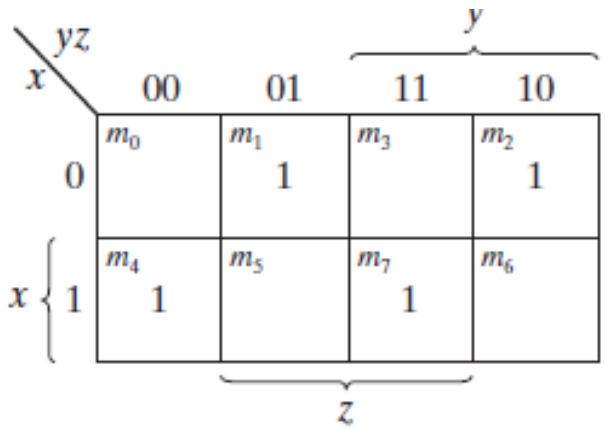


- Truth Table:

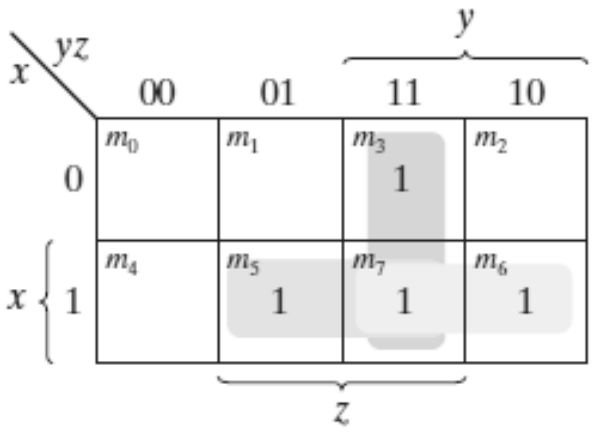
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Full Adder

- Simplification



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



$C = xy + xz + yz$

(or)

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

= x'(y'z + yz') + x(y'z' + yz)

= x' (y⊕ z) + x(y ⊕z)'

= x ⊕y ⊕z

C = xy'z + xyz' + xyz +x'yz

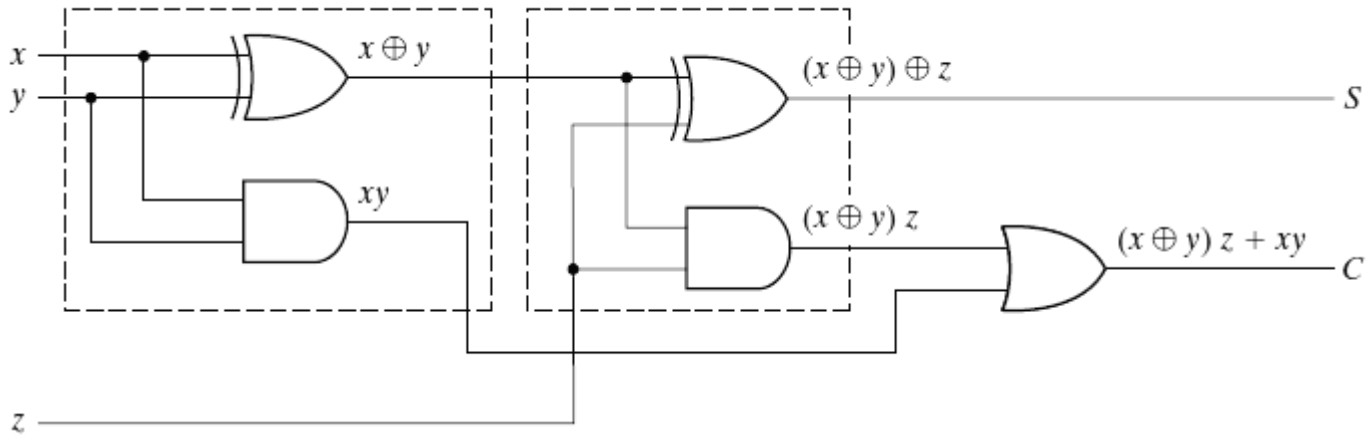
= x (y'z + yz') + yz(x+x')

= x (y ⊕z) + yz

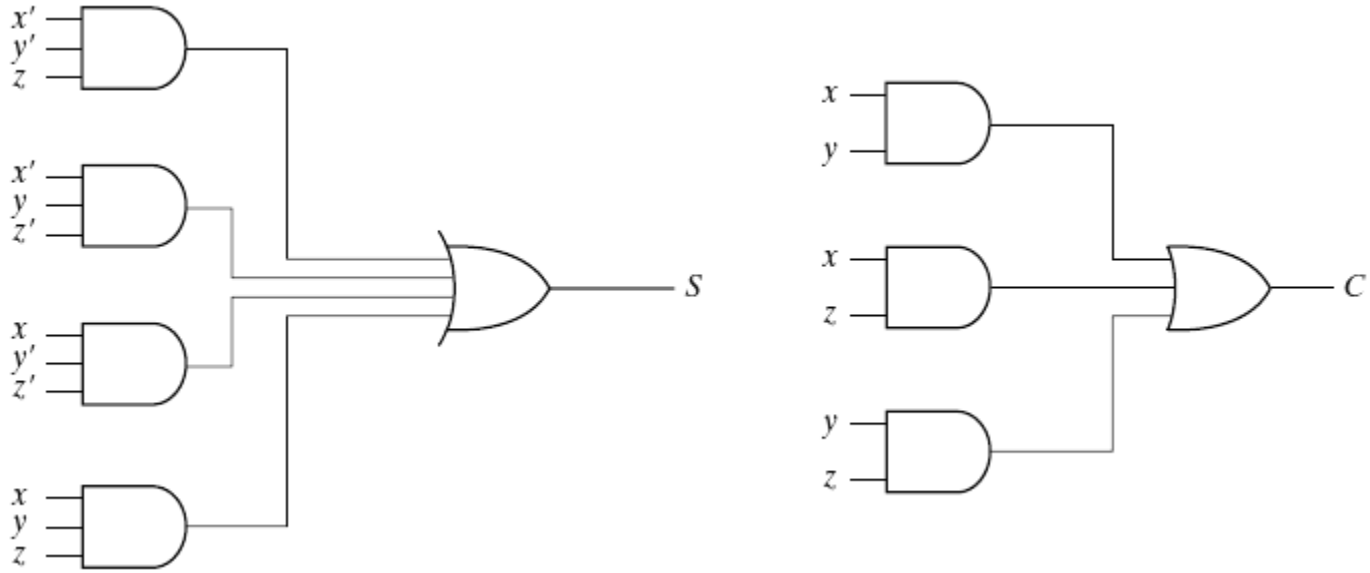
Full Adder

- Circuit Diagram:

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



Implementation of full adder in sum-of-products form



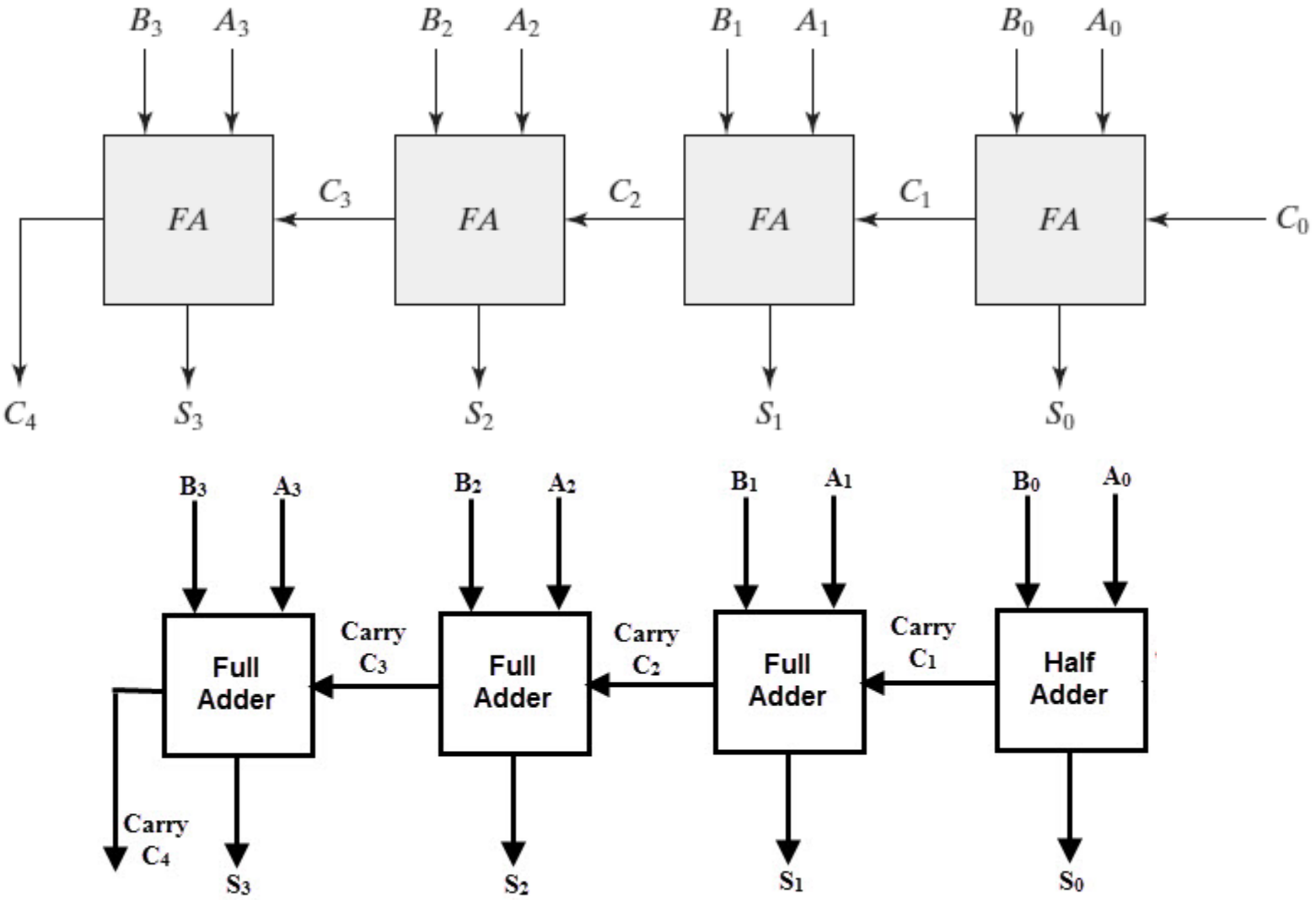
Full Adder

- Addition of *n-bit numbers requires*
  - *a chain of n full-adders*
    - *the input carry to the least significant position* is fixed at 0.

      (or)
  - *a chain of one half-adder and n-1 full-adders.*
- Consider the two binary numbers
  - *A = 1011*
  - *B = 0011.*
  - *Their sum S = 1110 is formed with the four-bit adder as follows:*
    - The bits are added with full adders, starting from the least significant position (bit position 0), to form the sum bit and carry bit.

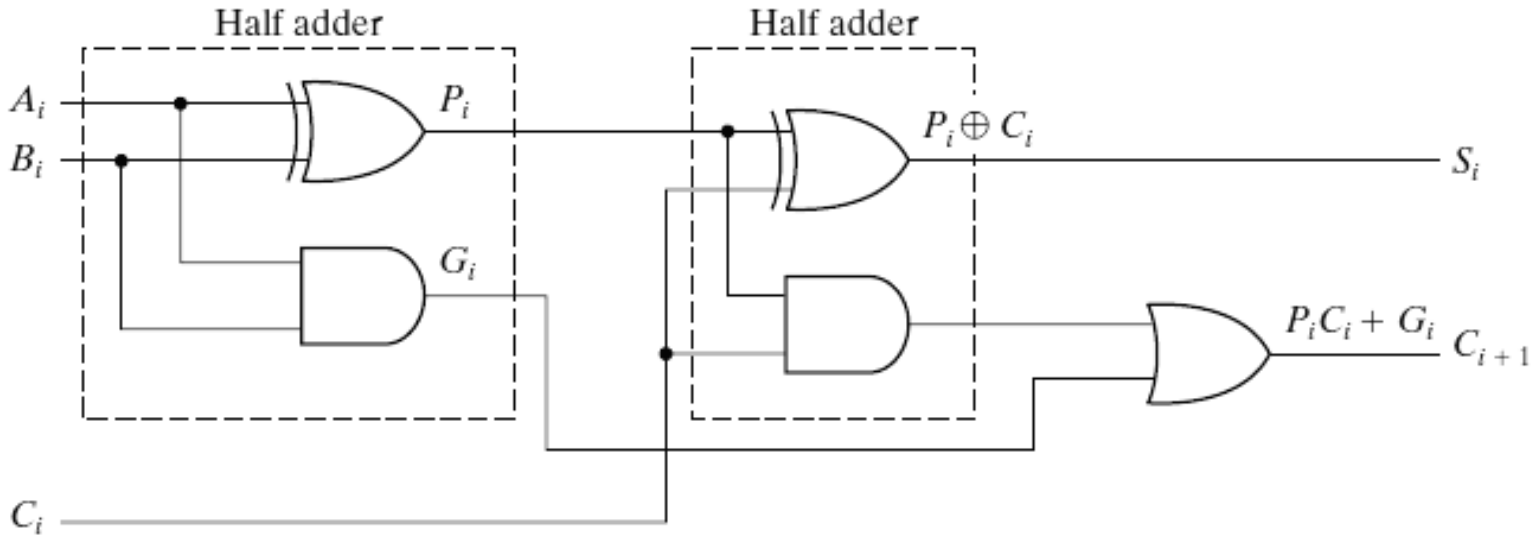
Bit Position	**3**	**2**	**1**	**0**	
Input carry	0	1	1	0	C_i
Augend	1	0	1	1	A_i
Addend	0	0	1	1	B_i
Sum	1	1	1	0	S_i
Output carry	0	0	1	1	C_{i+1}

Four-bit adder

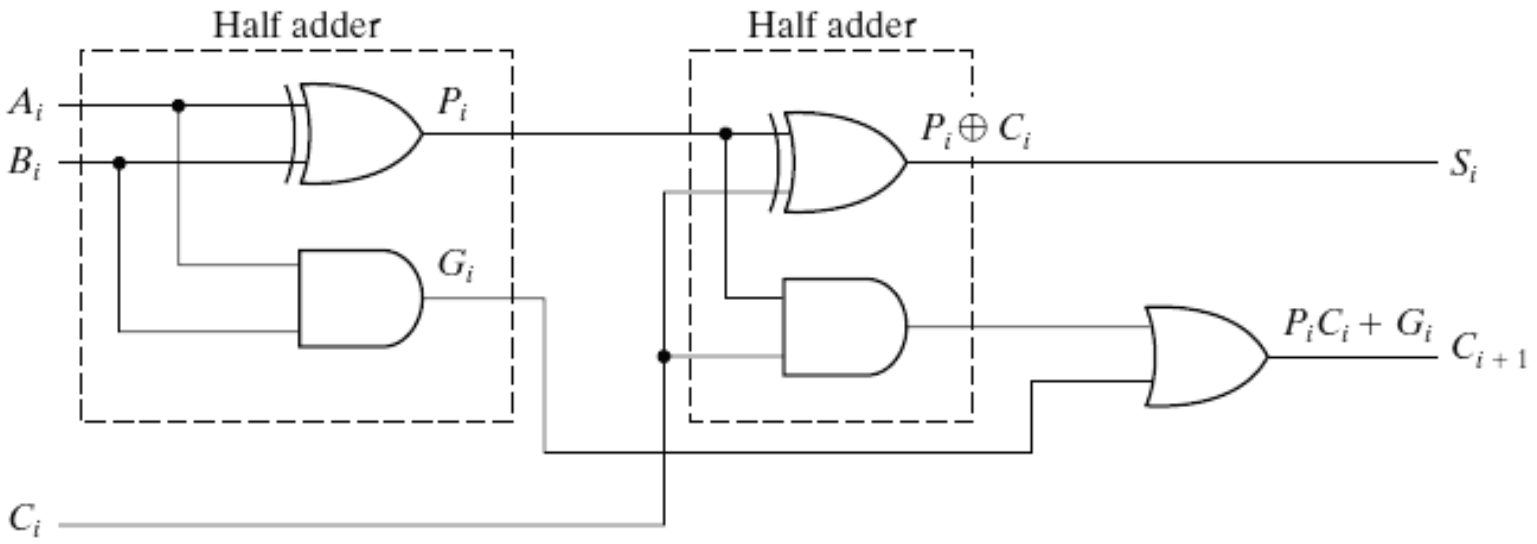


Carry Propagation

- Since each bit of the sum output depends on the value of the input carry, the value of S_i 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 because it limits the speed with which two numbers are added.
- Although the adder will always have some value at its output terminals, the outputs will not be correct unless the signals are given enough time to propagate through the gates connected from the inputs to the outputs.
- Since all other arithmetic operations are implemented by successive additions, the time consumed during the addition process is critical.
- 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 Propagation



- Consider the full adder circuit:
  - Let

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

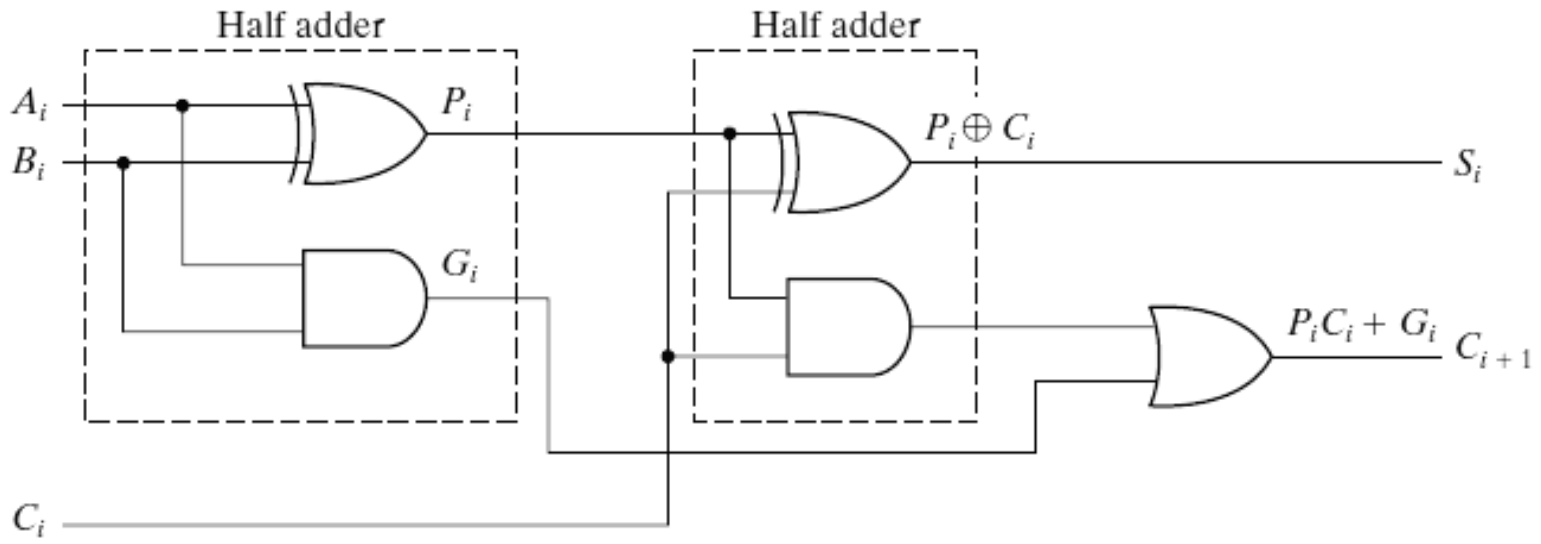
  - The output and carry are

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

G_i (carry generate) - produces a carry of 1 when both A_i and B_i are 1, regardless of the input carry *C_i*.

P_i (carry propagate) – determines whether a carry into stage *i will propagate into stage i + 1 (i.e., whether an assertion of Ci will propagate to an assertion of C_{i+1}).*

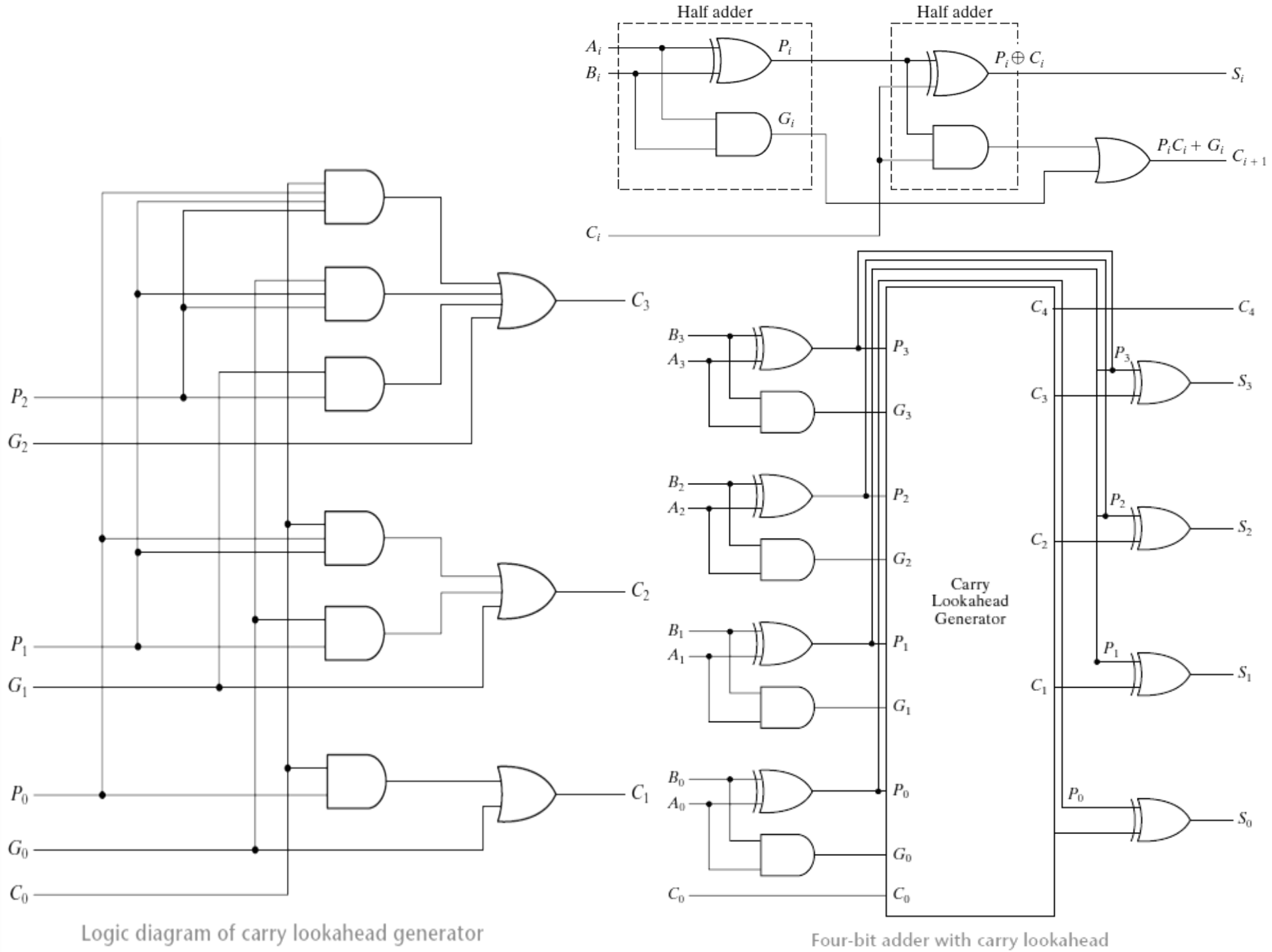
Carry Propagation



$$P_i = A_i \oplus B_i \qquad S_i = P_i \oplus C_i$$
$$G_i = A_iB_i \qquad C_{i+1} = G_i + P_iC_i$$

- The Boolean functions for the carry outputs of each stage and substitute the value of each C_i *from the previous equations:*

$$C_0 = \text{input carry}$$
$$C_1 = G_0 + P_0C_0$$
$$C_2 = G_1 + P_1C_1 = G_1 + P_1(G_0 + P_0C_0) = G_1 + P_1G_0 + P_1P_0C_0$$
$$C_3 = G_2 + P_2C_2 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$$

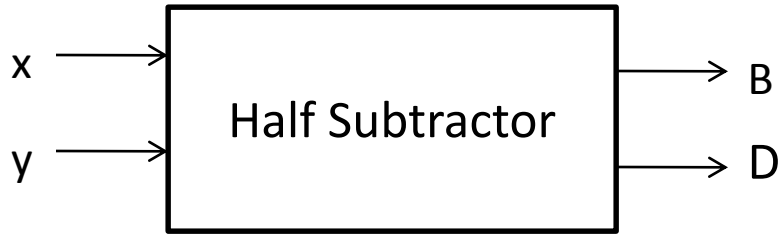


Logic diagram of carry lookahead generator

Four-bit adder with carry lookahead

Half Subtractor

- A combinational circuit that performs the subtraction of two bits is called a *half subtractor.*
- This circuit needs two binary inputs and two binary outputs.
  - The input variables designate the minuend and subtrahend bits
  - the output variables produce the difference and borrow
- Block Diagram:



- Truth Table:

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

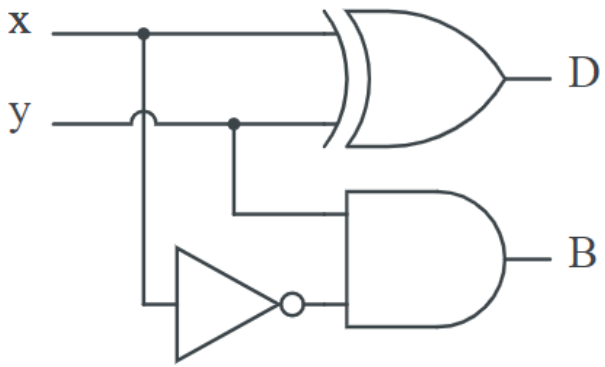
Half Subtractor

- Simplified Boolean Functions:

  $D = x'y + xy' = x \oplus y$

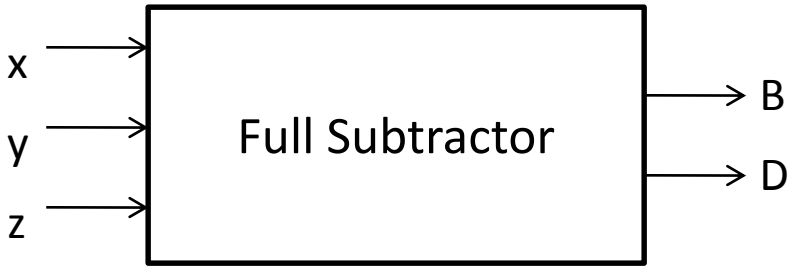
  $B = x'y$

- Circuit Diagram:



Full Subtractor

- A full subtractor is a combinational circuit that performs a subtraction between two bits taking into account that a 1 may have been borrowed by a lower significant stage.
- It consists of three inputs and two outputs.
  - Two of the input variables designate the minuend and subtrahend bits
  - The third input *represents* previous borrow.
  - The output variables represent difference and borrow.
- Block Diagram:

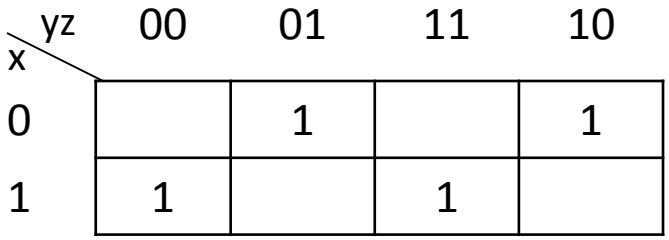


- Truth Table:

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

Full Subtractor

- Simplification



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

(or)

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

= x'(y'z + yz') + x(y'z' + yz)

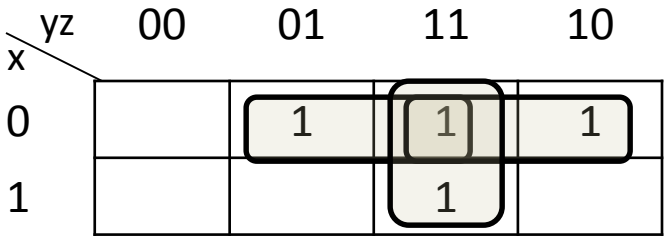
= x' (y⊕ z) + x(y ⊕z)'

= x ⊕y ⊕z

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

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

= z (x ⊕y)' + x'y

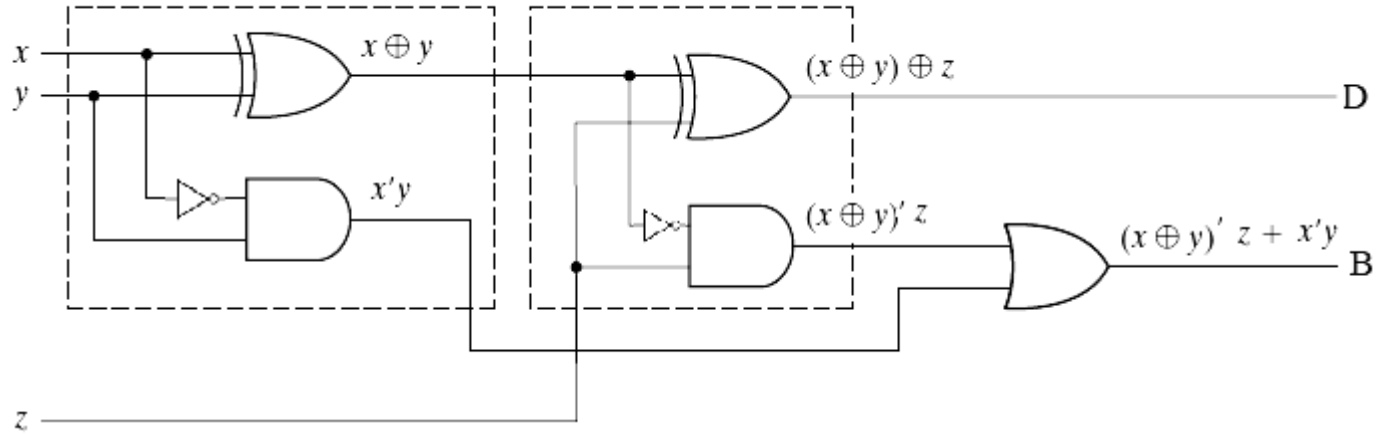


B = x'y + x'z + yz

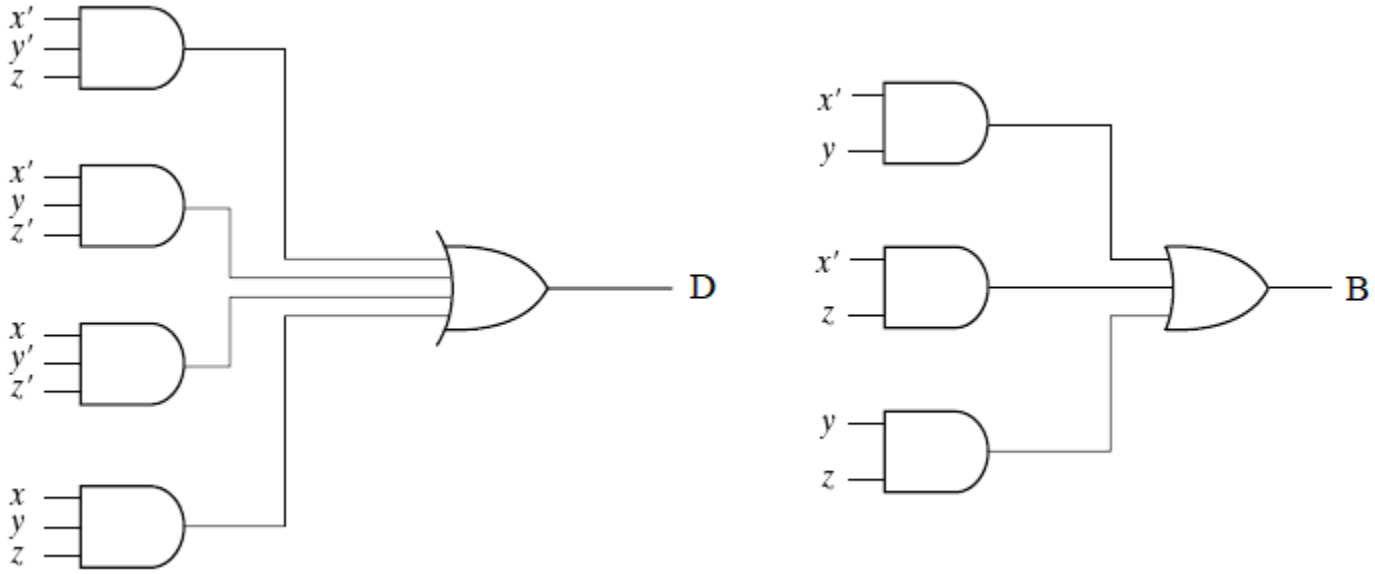
Full Subtractor

- Circuit Diagram:

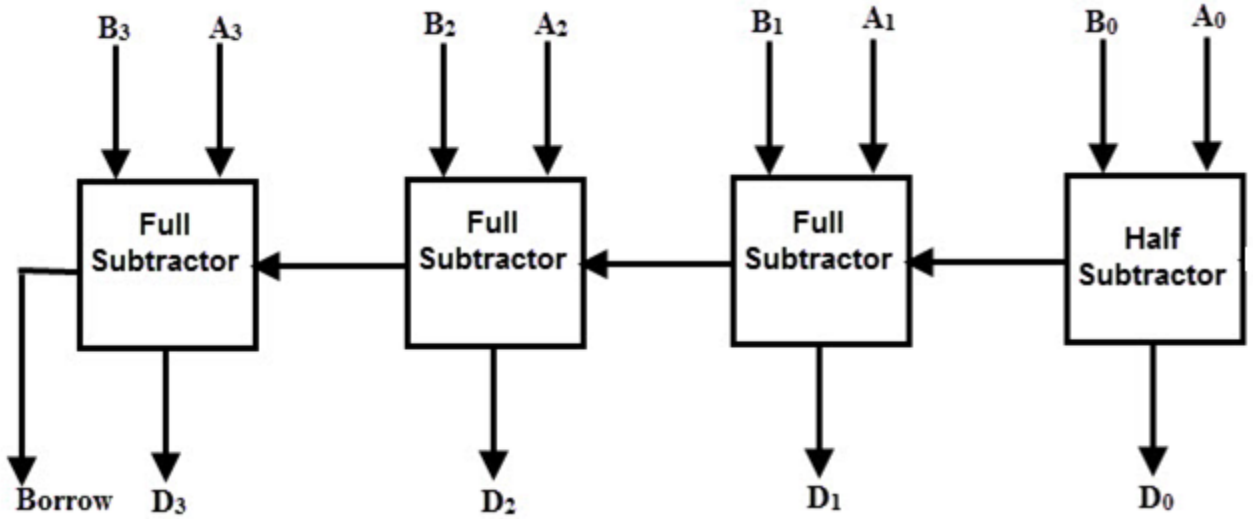
Implementation of full subtractor with two half subtractor and an OR gate

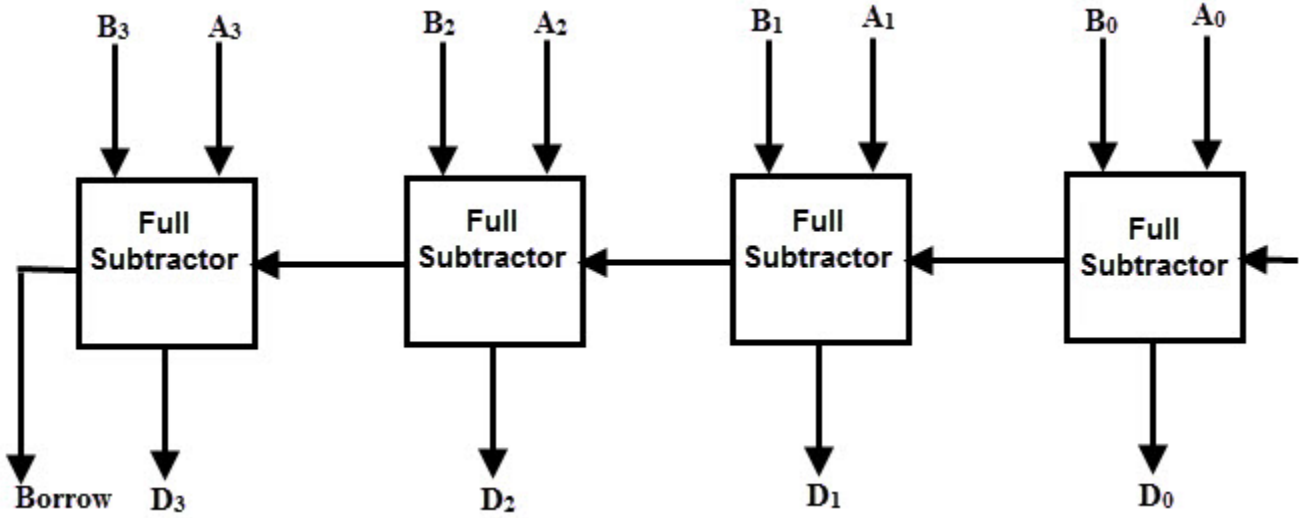


Implementation of full subtractor in sum-of-products form

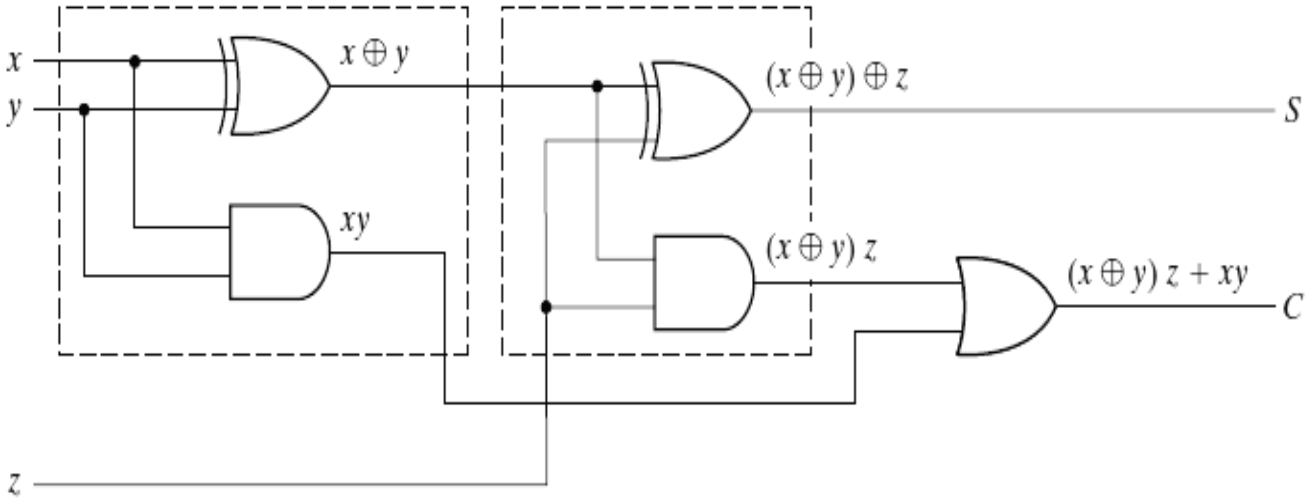


Four bit subtractor





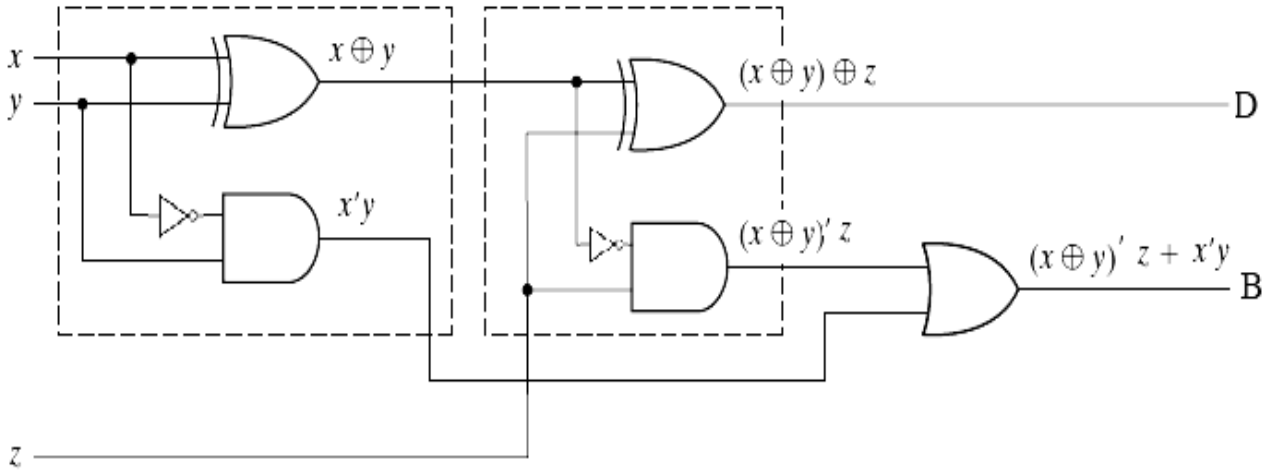
Full Adder:



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

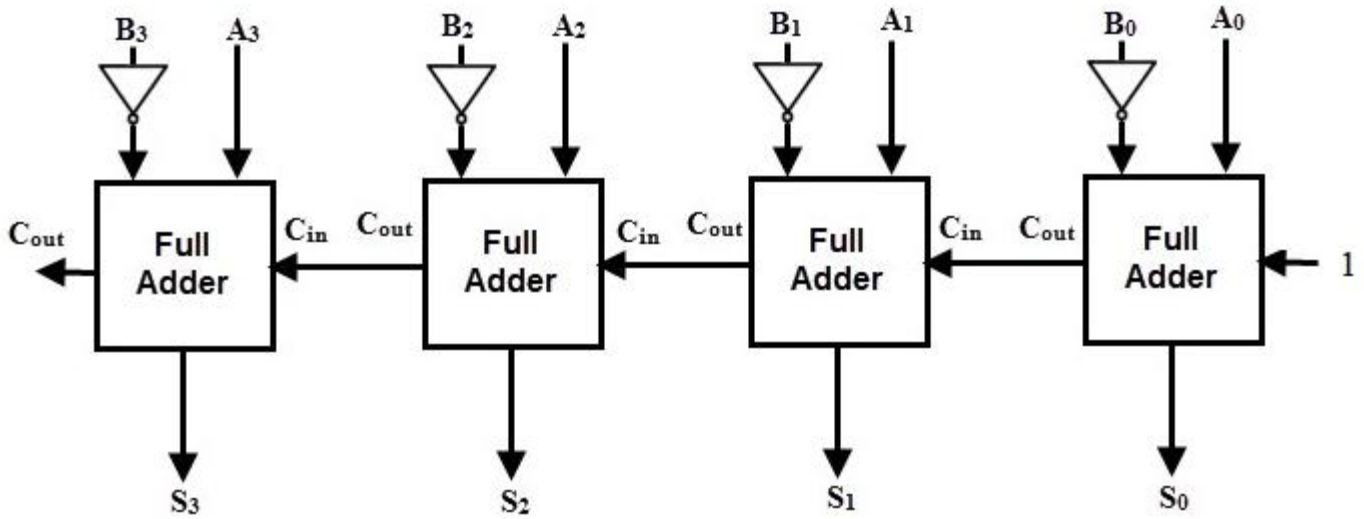
Full Subtractor:

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



Binary Subtractor – Another Implementation

- The subtraction of unsigned binary numbers can be done most conveniently by means of complements
  - 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.



Addition & Subtraction

Addition(x+y+z)

i ⟶	3	2	1	0
Carry(z_i)	0	1	0	0
Addend(x_i)	1	0	1	0
Augend(y_i)	0	0	1	1
Sum(S_i)	1	1	0	1
Carry(C_i)	0	0	1	0

Subtraction(x-y-z)

i ⟶	3	2	1	0
Borrow(z_i)	1	1	1	0
Minuend(x_i)	1	0	1	0
Subtrahend(y_i)	0	0	1	1
Difference(D_i)	0	1	1	1
Borrow(B_i)	0	1	1	1

Subtraction using Addition

i ⟶	3	2	1	0
Carry(z_i)	0	0	0	1
Addend(x_i)	1	0	1	0
Augend(y_i')	1	1	0	0
Sum(S_i)	0	1	1	1
Carry(C_i)	1	0	0	0

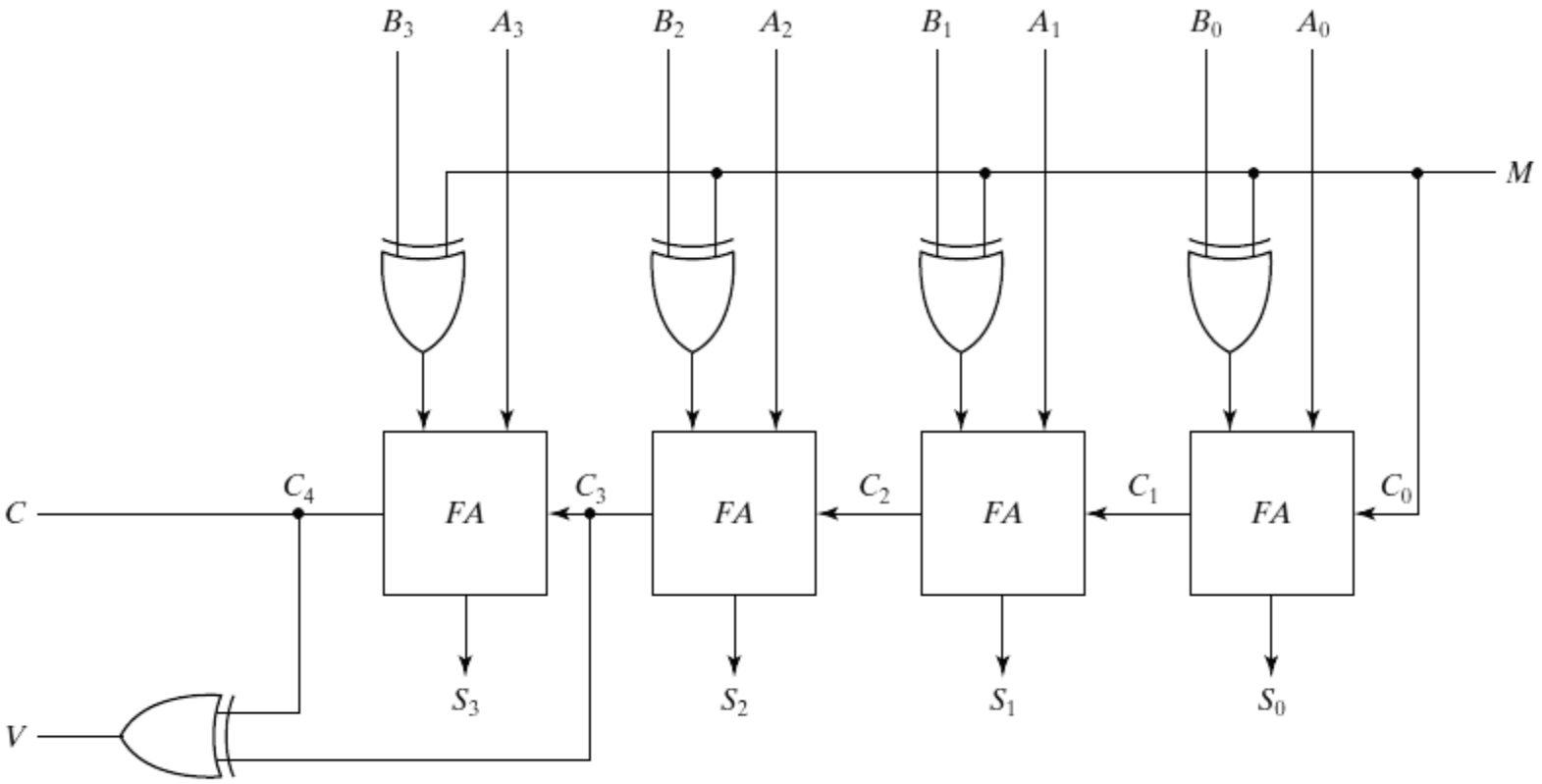
Carry bit starts with 1!

Complement of 0011

Adder – Subtractor Circuit

- The addition and subtraction operations can be combined into one circuit with one common binary adder by including an exclusive-OR gate with each full adder.
- A four-bit adder–subtractor circuit is shown:
  - The mode input *M controls the operation.*
    - When *M = 0, the circuit is an adder*
    - *When M = 1, the circuit becomes a subtractor.*
- Each exclusive-OR gate receives input *M and one of the inputs of B .*
  - *When M = 0, we* have *B ⊕ 0 = B.*
    - *The full adders receive the value of B , the input carry is 0, and the* circuit performs *A plus B .*
  - *When M = 1, we have B ⊕ 1 = B' and C_0 = 1.*
    - *The B inputs* are all complemented and a 1 is added through the input carry. The circuit performs the operation *A plus the 2's complement of B . (The exclusive-OR with output V is for* detecting an overflow.)
- Binary numbers in the signed-complement system are added and subtracted by the same basic addition and subtraction rules as are unsigned numbers.
  - Therefore, computers need only one common hardware circuit to handle both types of arithmetic.
  - The user or programmer must interpret the results of such addition or subtraction differently, depending on whether it is assumed that the numbers are signed or unsigned.

Adder – Subtractor Circuit



Four-bit adder–subtractor (with overflow detection)