EC 8392 – DIGITAL ELECTRONICS

UNIT – II : COMBINATIONAL CIRCUIT DESIGN

INTRODUCTION:

The digital system consists of two types of circuits, namely

(i). Combinational circuits

(ii). Sequential circuits

Combinational circuit consists of logic gates whose output at any time is determined from the present combination of inputs. The logic gate is the most basic building block of combinational logic. The logical function performed by a combinational circuit is fully defined by a set of Boolean expressions.

Sequential logic circuit comprises both logic gates and the state of storage elements such as flip-flops. As a consequence, the output of a sequential circuit depends not only on present value of inputs but also on the past state of inputs.

In the previous chapter, we have discussed binary numbers, codes, Boolean algebra and simplification of Boolean function and logic gates. In this chapter, formulation and analysis of various systematic designs of combinational circuits will be discussed.

A combinational circuit consists of input variables, logic gates, and output variables. The logic gates accept signals from inputs and output signals are generated according to the logic circuits employed in it. Binary information from the given data transforms to desired output data in this process. Both input and output are obviously the binary signals, *i.e.*, both the input and output signals are of two possible states, logic 1 and logic 0.

For *n* number of input variables to a combinational circuit, 2^n possible combinations of binary input states are possible. For each possible combination, there is one and only one possible output combination. A combinational logic circuit can be described by *m* Boolean functions and each output can be expressed in terms of *n* input variables.

<u>DESIGN PROCEDURE:</u>

Any combinational circuit can be designed by the following steps of design procedure.

1. The problem is stated.
2. Identify the input and output variables.
3. The input and output variables are assigned letter symbols.
4. Construction of a truth table to meet input -output requirements.
5. Writing Boolean expressions for various output variables in terms of input variables.
6. The simplified Boolean expression is obtained by any method of minimization—algebraic method, Karnaugh map method, or tabulation method.
7. A logic diagram is realized from the simplified boolean expression using logic gates.

The following guidelines should be followed while choosing the preferred form for hardware implementation:

1. The implementation should have the minimum number of gates, with the gates used having the minimum number of inputs.
2. There should be a minimum number of interconnections. 3.Limitation on the driving capability of the gates should not be ignored.

ARITHMETIC CIRCUITS – BASIC BUILDING BLOCKS

In this section, we will discuss those combinational logic building blocks that can be used to perform addition and subtraction operations on binary numbers. Addition and subtraction are the two most commonly used arithmetic operations, as the other two, namely multiplication and division, are respectively the processes of repeated addition and repeated subtraction.

The basic building blocks that form the basis of all hardware used to perform the arithmetic operations on binary numbers are half-adder, full adder, half-subtractor, full- subtractor.

Half-Adder:

A half-adder is a combinational circuit that can be used to add two binary bits. It has two inputs that represent the two bits to be added and two outputs, with one producing the SUM output and the other producing the CARRY.

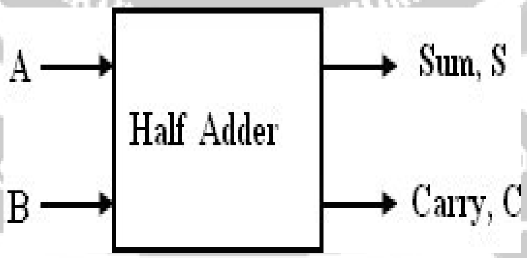


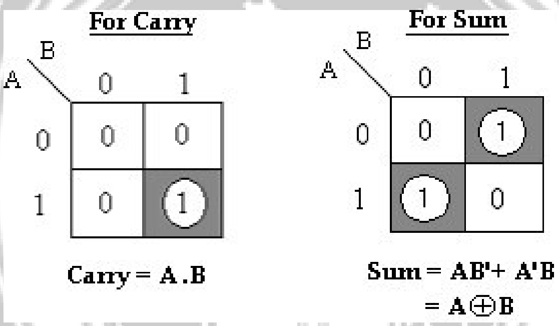
Fig 2.1 - Block schematic of half-adder

The truth table of a half-adder, showing all possible input combinations and the corresponding outputs are shown below.

Inputs		Outputs	
A	B	Carry (C)	Sum (S)
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Truth table of half-adder

K-map simplification for carry and sum:



The Boolean expressions for the SUM and CARRY outputs are given by the equations,

Sum, S = A’B+ AB’= A B

Carry, C = A . B

The first one representing the SUM output is that of an EX-OR gate, the second one representing the CARRY output is that of an AND gate. The logic diagram of the half adder is,

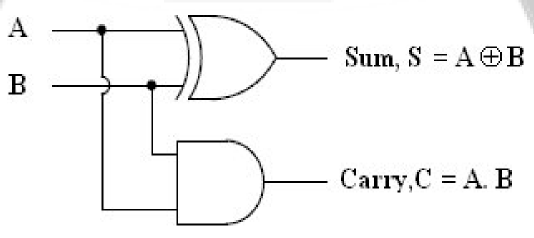


Fig 2.2 : Logic Implementation of Half-adder

Full-Adder:

A full adder is a combinational circuit that forms the arithmetic sum of three input bits. It consists of 3 inputs and 2 outputs.

Two of the input variables, represent the significant bits to be added. The third input represents the carry from previous lower significant position. The block diagram of full adder is given by,

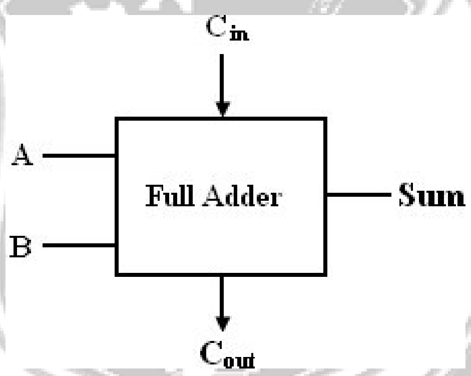


Fig 2.3 - Block schematic of full-adder

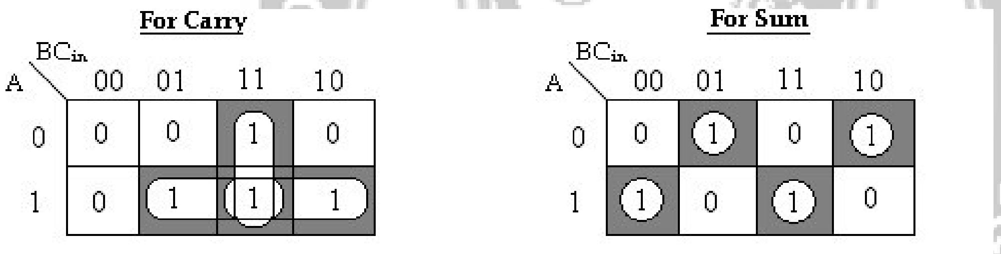
The full adder circuit overcomes the limitation of the half-adder, which can be used to add two bits only. As there are three input variables, eight different input combinations are possible. The truth table is shown below,

Truth Table:

Inputs			Outputs	
A	**B**	**Cin**	**Sum (S)**	**Carry (C_{out})**
0	0	0	0	0

0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

To derive the simplified Boolean expression from the truth table, the Karnaugh map method is adopted as,



Carry, C_{out} = AB+ AC_{in} + BC_{in} Sum, S = A′B′C_{in}+ A′BC′$_{in}$+ AB′C′$_{in}$+ ABC_{in}

The Boolean expressions for the SUM and CARRY outputs are given by the equations,

Sum, S = A’B’Cin+ A’BC’in + AB’C’in + ABCin Carry, Cout =

AB+ ACin + BCin .

The logic diagram for the above functions is shown as,

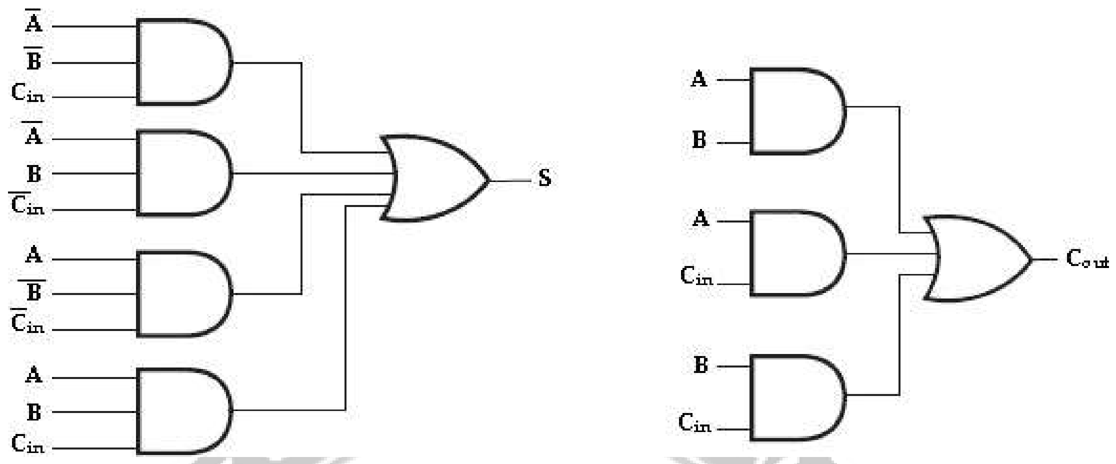


Fig : 2.4 -Implementation of full-adder in Sum of Products

The logic diagram of the full adder can also be implemented with two half-adders and one OR gate. The S output from the second half adder is the exclusive-OR of C_{in} and the output of the first half-adder, giving

Sum = $\mathbf{C_{in} \quad (A \quad B)}$ = $\qquad\qquad$ [$x \quad y = x'y+xy'$]

$Cin \quad (A'B+AB')$

$= C'in\ (A'B+AB') + Cin\ (A'B+AB')'$ $\qquad$ [$(x'y+xy')' = (xy+x'y')$]

$= C'in\ (A'B+AB') + Cin\ (AB+A'B')$

$= A'BC'in + AB'C'in + ABCin + A'B'Cin$.

and the carry output is,

Carry, $\mathbf{C_{out} = AB + C_{in}\,(A'B+AB')}$

$= AB + A'BCin + AB'Cin$

$= AB\,(C_{in}+1) + A'BC_{in} + AB'C_{in}$ $\qquad$ [$C_{in}+1 = 1$]

$= ABCin + AB + A'BCin + AB'Cin$

$= AB + ACin\,(B+B') + A'BCin$

$= AB + AC_{in} + A'BC_{in}$

$= AB(C_{in}+1) + AC_{in} + A'BC_{in}$ $[C_{in}+1 = 1]$

$= ABC_{in} + AB + AC_{in} + A'BC_{in}$

$= AB + AC_{in} + BC_{in}(A + A') =$

$AB + AC_{in} + BC_{in}.$

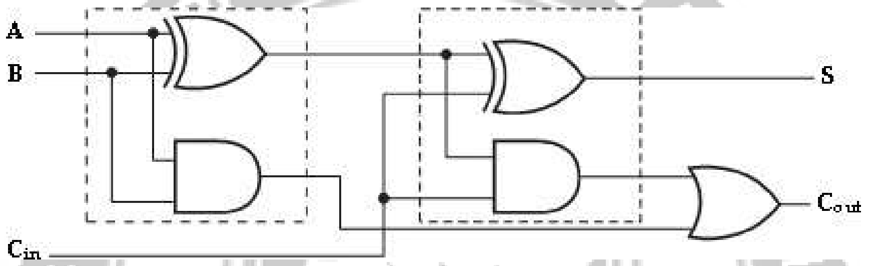


Fig : 2.5 - Implementation of full adder with two half-adders and an OR gate

Half -Subtractor:

A *half-subtractor* is a combinational circuit that can be used to subtract one binary digit from another to produce a DIFFERENCE output and a BORROW output. The BORROW output here specifies whether a ‗1' has been borrowed to perform the subtraction.

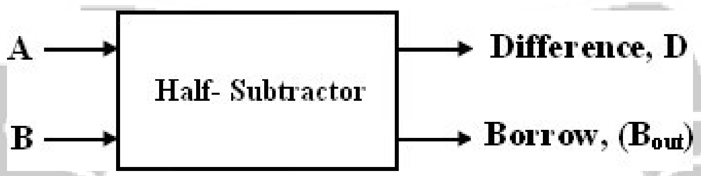
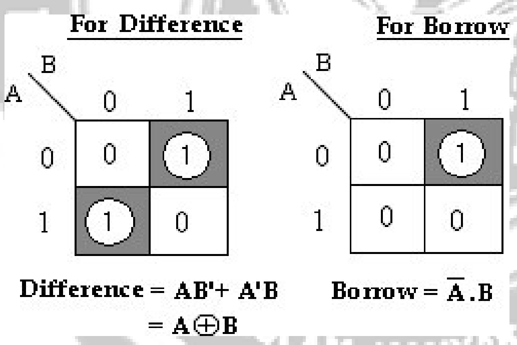


Fig : 2.6 - Block schematic of half-subtractor

The truth table of half-subtractor, showing all possible input combinations and the corresponding outputs are shown below.

Input		**Output**	
A	**B**	**Difference (D)**	**Borrow (B_{out})**
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

<u>K-map simplification for half subtractor</u>:



The Boolean expressions for the DIFFERENCE and BORROW outputs are given by the equations,

Difference, D = A'B+ AB'= A B Borrow, B_{out}

= A' .B

The first one representing the DIFFERENCE (**D**)output is that of an exclusive-OR gate, the expression for the BORROW output (**B_{out}**) is that of an AND gate with input A complemented before it is fed to the gate.

The logic diagram of the half adder is,

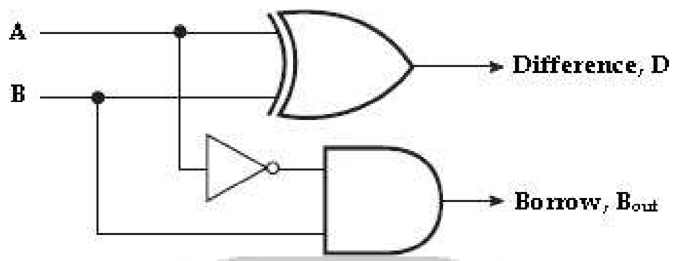


Fig : 2.7 - Logic Implementation of Half-Subtractor

Comparing a half-subtractor with a half-adder, we find that the expressions for the SUM and DIFFERENCE outputs are just the same. The expression for BORROW in the case of the half-subtractor is also similar to what we have for CARRY in the case of the half-adder. If the input A, ie., the minuend is complemented, an AND gate can be used to implement the BORROW output. **Full Subtractor:**

A *full subtractor* performs subtraction operation on two bits, a minuend and a subtrahend, and also takes into consideration whether a ‗1' has already been borrowed by the previous adjacent lower minuend bit or not.

As a result, there are three bits to be handled at the input of a full subtractor, namely the two bits to be subtracted and a borrow bit designated as B_{in}. There are two outputs, namely the DIFFERENCE output D and the BORROW output B_o. The

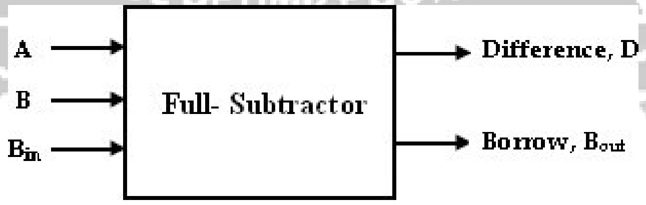


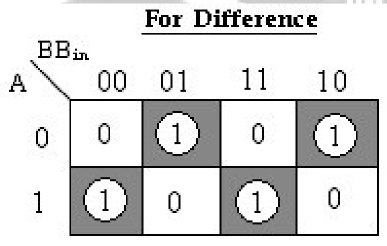
Fig : 2.8 - Block schematic of full-adder

BORROW output bit tells whether the minuend bit needs to borrow a ‗1' from the next possible higher minuend bit.

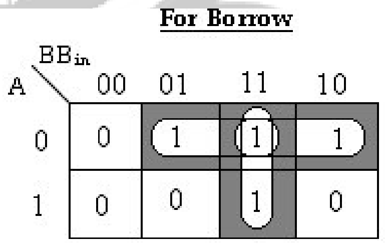
The truth table for full-subtractor is,

Inputs			Outputs	
A	**B**	**Bin**	**Difference(D)**	**Borrow(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

K-map simplification for full-subtractor:



Difference, $D = A'B'B_{in} + A'BB'_{in} + AB'B'_{in} + ABB_{in}$



Borrow, $B_{out} = A'B + A'B_{in} + BB_{in}$

The Boolean expressions for the DIFFERENCE and BORROW outputs are given by the equations,

Difference, D = A’B’Bin+ A’BB’in + AB’B’in + ABBin

Borrow, Bout = A’B+ A’Cin + BBin .

The logic diagram for the above functions is shown as,

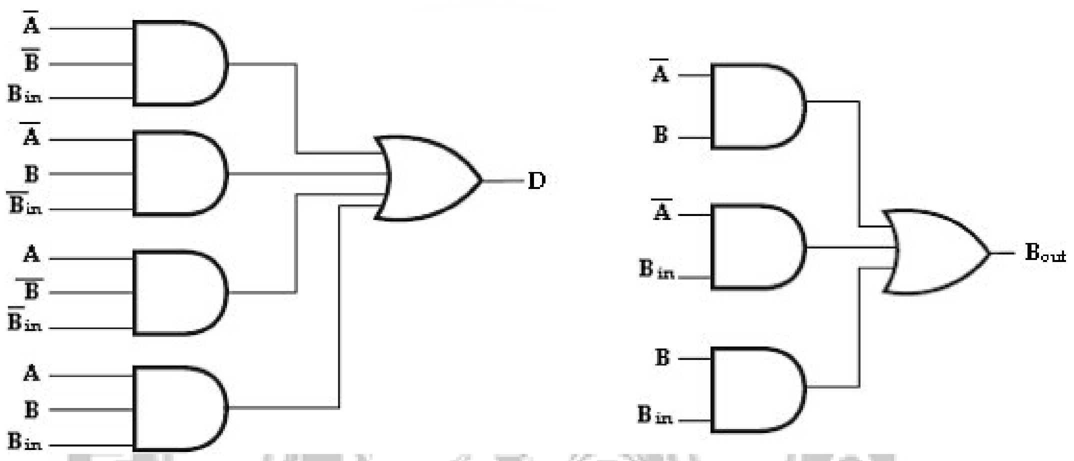


Fig : 2.9 - Implementation of full-adder in Sum of Products

The logic diagram of the full-subtractor can also be implemented with two half-subtractors and one OR gate. The difference,D output from the second half subtractor is the exclusive-OR of B_{in} and the output of the first half-subtractor, giving

Difference,D= **B_{in} (A B)** = [x y = x‘y+xy‘]

Bin (A‘B+AB‘)

= B‘in (A‘B+AB‘) + Bin (A‘B+AB‘)‘ [(x‘y+xy‘)‘= (xy+x‘y‘)]

= B‘in (A‘B+AB‘) + Bin (AB+A‘B‘)

= A‘BB‘in + AB‘B‘in + ABBin + A‘B‘Bin .

and the borrow output is,

Borrow, Bout = **A'B+ Bin (A'B+AB')'** [(x'y+xy')'=(xy+x'y')]

= A'B+ Bin (AB+A'B')

= A'B+ ABBin+ A'B'Bin

= A'B (Bin+1) + ABBin+ A'B'Bin [C_{in}+1= 1]

= A'BBin+ A'B+ ABBin+ A'B'Bin

= A'B+ BBin (A+A') + A'B'Bin [A+A'= 1]

= A'B+ BBin+ A'B'Bin

= A'B (Bin+1) + BBin+ A'B'Bin [C_{in}+1= 1]

= A'BBin+ A'B+ BBin+ A'B'Bin

= A'B+ BBin+ A'Bin (B +B')=

A'B+ BBin+ A'Bin.

Therefore, we can implement full-subtractor using two half-subtractors and OR gate as,

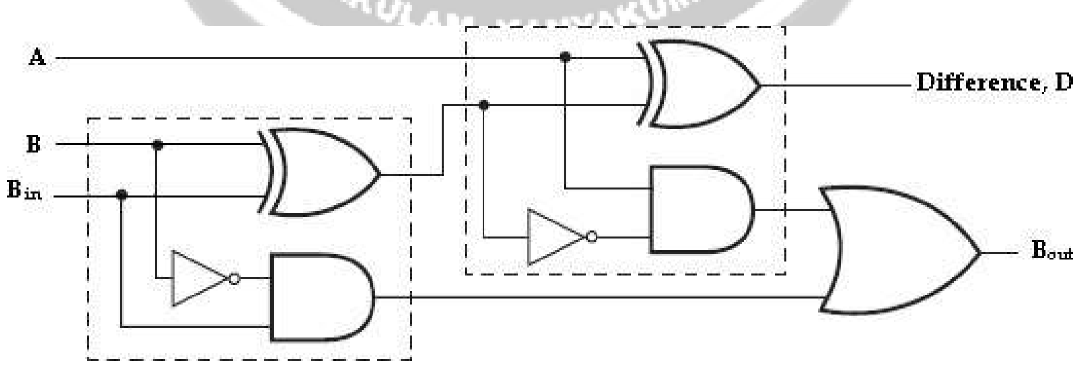


Fig : 2.10 - Implementation of full-subtractor with two half-subtractors and an OR gate