# MODULE – 2

## THE BASIC GATES & THE COMBINATIONAL LOGIC CIRCUITS

## THE BASIC GATES

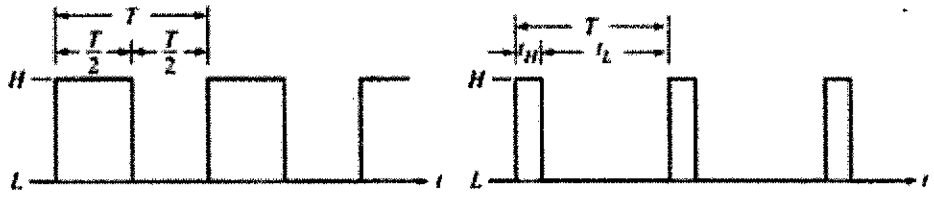
**PREREQUISITES:**

Electronic circuits and systems can be divided into two broad categories – *analog* and *digital*. Analog circuits are designed for use with small signals and are used in a linear fashion. Digital circuits are generally used with large signals and are considered nonlinear. Any quantity that changes with time can be represented as an analog signal or it can be treated as digital signal.

Digital electronics involves circuits that have exactly two possible states. A system having only two states is said to be *binary*. The binary number system is widely used in digital electronics.

| Hexa-Decimal | Decimal | Binary | | Hexa-Decimal | Decimal | Binary |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 0 0 0 | | 8 | 8 | 1 0 0 0 |
| 1 | 1 | 0 0 0 1 | | 9 | 9 | 1 0 0 1 |
| 2 | 2 | 0 0 1 0 | | A | 10 | 1 0 1 0 |
| 3 | 3 | 0 0 1 1 | | B | 11 | 1 0 1 1 |
| 4 | 4 | 0 1 0 0 | | C | 12 | 1 1 0 0 |
| 5 | 5 | 0 1 0 1 | | D | 13 | 1 1 0 1 |
| 6 | 6 | 0 1 1 0 | | E | 14 | 1 1 1 0 |
| 7 | 7 | 0 1 1 1 | | F | 15 | 1 1 1 1 |

The operation of electronic circuits can be described in terms of its voltage levels – *high* (H) level and *low* (L) level. This could be related to the binary number system by assigning L = 0 = F (false) and H = 1 = T (true).



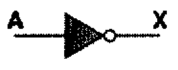
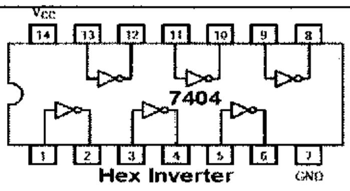
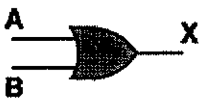
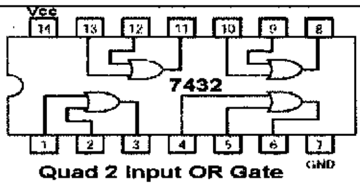
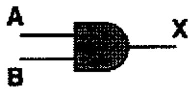
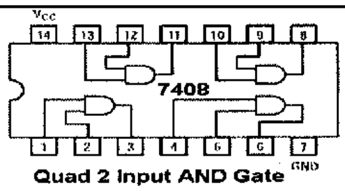
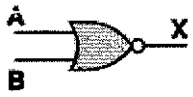
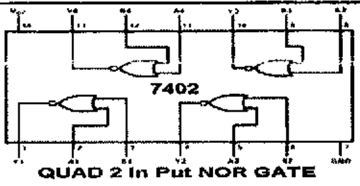
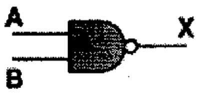
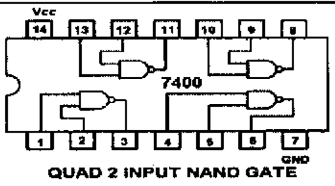
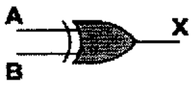
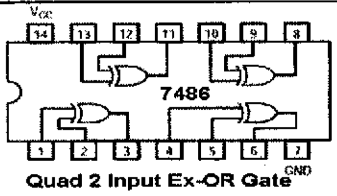
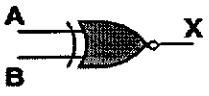
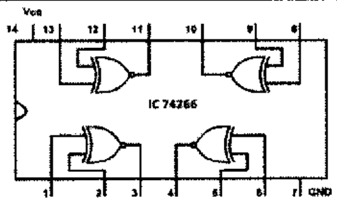
Symmetrical Signal & Asymmetrical Signal

The frequency is defined as, f = 1 / T where, T is the period of the signal.

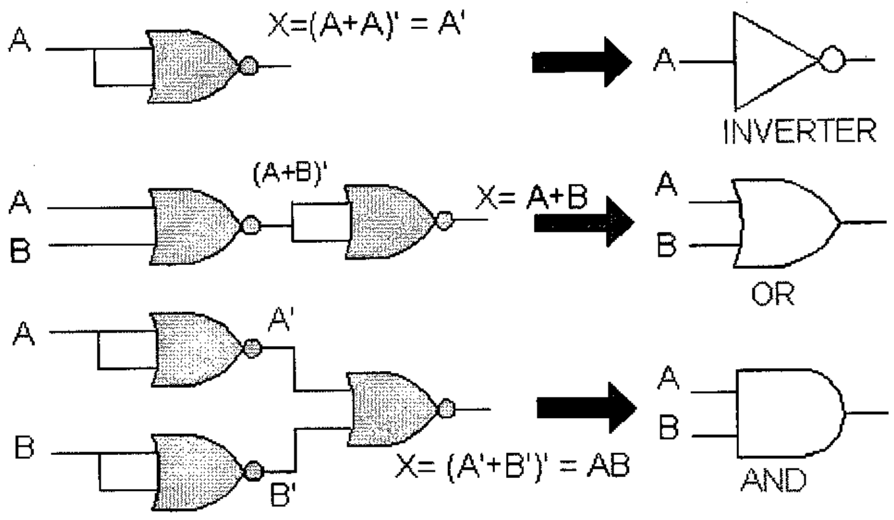
Duty Cycle is a convenient measure of how symmetrical or how unsymmetrical a waveform is.

$$Duty\ Cycle = \frac{T_{on}}{T_{on}+T_{off}} \qquad Duty\ Cycle, H = \frac{T_{on}}{T_{on}+T_{off}} \qquad Duty\ Cycle, L = \frac{T_{off}}{T_{on}+T_{off}}$$

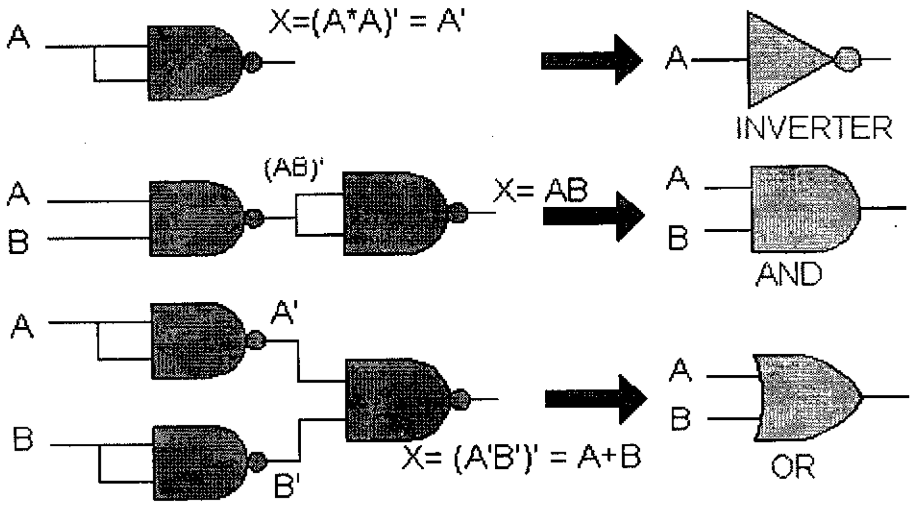
## REVIEW OF LOGIC GATES:

| Circuit Symbol | Truth Table | | | Verilog | IC Details |
|---|---|---|---|---|---|
| | A | B | X | | |
| NOT Gate:  $X = \bar{A}$ | 0 | - | 1 | X = ~A not (X, A) |  |
| | 1 | - | 0 | | |
| OR Gate:  | 0 | 0 | 0 | X = A \| B or (X, A, B) |  |
| | 0 | 1 | 1 | | |
| | 1 | 0 | 1 | | |
| $X = A + B$ | 1 | 1 | 1 | | |
| AND Gate:  | 0 | 0 | 0 | X = A & B and (X, A, B) |  |
| | 0 | 1 | 0 | | |
| | 1 | 0 | 0 | | |
| $X = A.B$ | 1 | 1 | 1 | | |
| NOR Gate:  | 0 | 0 | 1 | X = ~(A \| B) nor (X, A, B) |  |
| | 0 | 1 | 0 | | |
| | 1 | 0 | 0 | | |
| $X = \overline{A + B}$ | 1 | 1 | 0 | | |
| NAND Gate:  | 0 | 0 | 1 | X = ~(A & B) nand (X, A, B) |  |
| | 0 | 1 | 1 | | |
| | 1 | 0 | 1 | | |
| $X = \overline{A \cdot B}$ | 1 | 1 | 0 | | |
| XOR Gate:  | 0 | 0 | 0 | X = A ^ B xor (X, A, B) |  |
| | 0 | 1 | 1 | | |
| | 1 | 0 | 1 | | |
| $X = A \oplus B = \bar{A}B + A\bar{B}$ | 1 | 1 | 0 | | |
| XNOR Gate:  | 0 | 0 | 1 | X = ~(A ^ B) xnor (X, A, B) |  |
| | 0 | 1 | 0 | | |
| | 1 | 0 | 0 | | |
| $X = A \odot B = \bar{A}\bar{B} + AB$ | 1 | 1 | 1 | | |

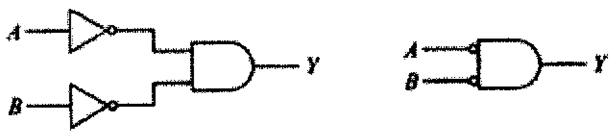
**Universality of NOR Gate:**



**Universality of NAND Gate:**



**Bubbled AND Gate:**



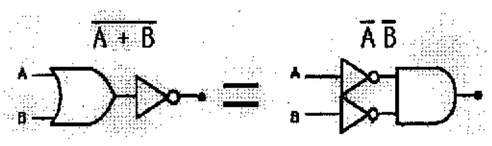
Bubbled AND gate and NOR gate are equivalent

**De Morgan's First Theorem:**

The complement of a sum equals the product of the complements. $\overline{A+B} = \bar{A}.\bar{B}$

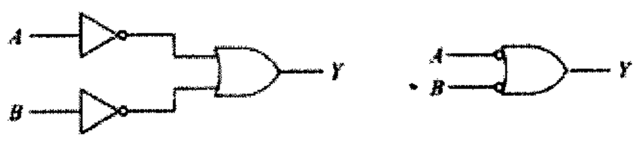
**Proof:**

| A | B | A+B | $\overline{A+B}$ | $\bar{A}$ | $\bar{B}$ | $\bar{A}.\bar{B}$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 |



NOR Gate Bubbled AND Gate

**Bubbled OR Gate:**



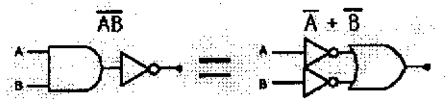
Bubbled OR gate and NAND gate are equivalent

**De Morgan's Second Theorem:**

The complement of a ~~sum~~ product equals the ~~product~~ sum of the complements. $\overline{AB} = \bar{A} + \bar{B}$

**Proof:**

| A | B | AB | $\overline{AB}$ | $\bar{A}$ | $\bar{B}$ | $\bar{A}+\bar{B}$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 |



NAND Gate Bubbled OR Gate

**Duality Theorem:** *Starting with a Boolean relation, you can derive another Boolean relation by –*

1. *Changing each OR sign to an AND sign*
2. *Changing each AND sign to an OR sign*
3. *Complementing any 0 or 1appearing in the expression.*

Example: 1. We say that, A+0 = A; the dual is, A.1 = A

2. Consider, A(B+C) = AB + AC

By changing the OR and AND operation, we get the dual relation:

A + BC = (A+B)(A+C)

**Laws of Boolean Algebra:**

- The following laws are of immense use in the simplification of Boolean expressions.

- Note that, if A is a variable, then either A = 0 or A = 1. Also, when A = 0, A ≠ 1; and when A = 1, A ≠ 0.

*De Morgan's First Theorem:-*

The complement of sum is equal to the product of the complements.

(A + B)' = A' . B'    i.e., a bubbled AND gate & a NOR gate are equivalent.

*De Morgan's Second Theorem:-*

The complement of a product is equal to the sum of the compliments.

(A . B)' = A' + B'    i.e., a bubbled OR gate & a NAND gate are equivalent.

1) *Commutative Law:-*

   A + B = B + A    and    A . B = B . A

2) *Associative Law:-*

   A + (B + C) = (A + B) + C    and    A . (BC) = (AB) . C

3) *Distributive Law:-*

   A(B + C) = AB + AC

4) *In relation to OR operation, the following laws hold good:-*

   A + 0 = A

   A + A = A

   A + 1 = 1 and

   A + A' = 1

5) *In relation to AND operation, the following laws hold good:-*

   A . 1 = A

   A . A = A

   A . 0 = 0

   A . A' = 0

   A'' = A

6) *Some more useful Boolean relations:-*

   A + AB = A

   A + A'B = A + B

   A (A + B) = A

   A (A' + B) = AB

   A + (B . C) = (A + B) (A + C)

*Simplification of Boolean Expressions:-*

- The following **hints** are found to be of use, in reducing complex Boolean expressions –

1. If there are parentheses present in the given expression, they are removed first; since, multiplication should precede addition.

   E.g.:- AB + C (A + B) = AB + AC + BC

2. If there are several identical terms, all except one can be removed.

   E.g.:- A + B + C + A . 1 = A + B + C + A = A + B + C

3. If a variable repeats in a term, only one variable may be retained.

   E.g.:- A . A = A

   B .B . C = BC

4. If in any term, both a variable & its complement are present, that term may be removed; since, AA' = 0.

   E.g.:- XX'Y = 0 . Y = 0

5. Identify pairs of terms which contains same variables. If in a pair, a variable is absent in one term, it can be removed.

   E.g.:- ABCD + ABC = ABC (D + 1)

   = ABC . 1 since, 1 + D = 1

   = ABC

6. If, in a pair of terms, several variables are common, and another variable is present in one term & its complement is present in another term, this variable & its complement can be removed.

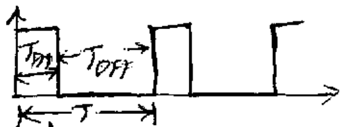
   E.g.:- ABC + A'BC = BC (A' + A)

   = BC . 1 since, A' + A = 1

   = BC

***Problem:*** *A signal waveform has a frequency of of 5 MHz, and the width of the positive pulse is 0.05 μs. What is the high duty cycle?*

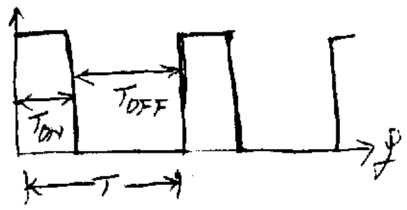
**Solution:** Given: $f = 5\ MHz$ & $T_{ON} = 0.05\mu s$.



The period of the waveform is: $T = \frac{1}{f} = \frac{1}{5\times10^6} = \underline{0.2\mu s}$.

$\therefore$ Duty cycle, $H = \frac{T_{ON}}{T} = \frac{0.05\mu s}{0.2\mu s} = \underline{\underline{0.25 \text{ or } 25\%}}$.

***Problem:*** *An asymmetrical signal waveform is high for 2 ms and low for 5 ms. Find the frequency and duty cycle L of the waveform.*



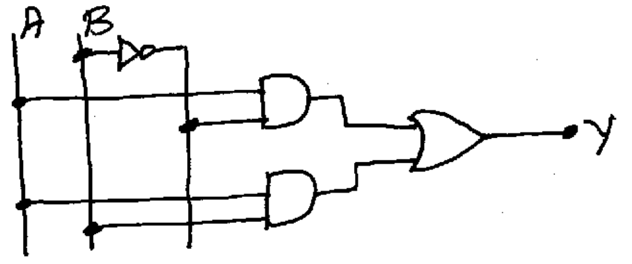
**Solution:** Given: $T_{ON} = 2ms$ & $T_{OFF} = 5ms$.

$$\text{Frequency, } f = \frac{1}{T} = \frac{1}{T_{ON}+T_{OFF}} = \frac{1}{(2+5)\times10^{-3}} = \frac{1}{7\times10^{-3}} = 142.8$$

$$\text{Duty cycle, } L = \frac{T_{OFF}}{T} = \frac{5\times10^{-3}}{7\times10^{-3}} = 0.714 = 71.4\%$$

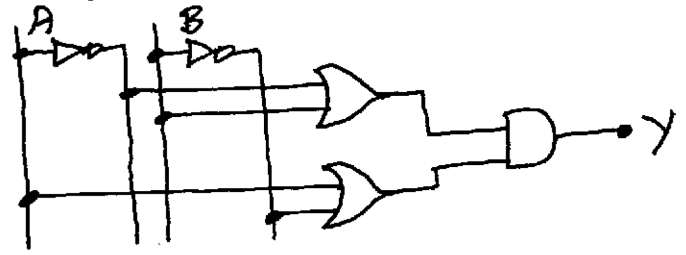
***Problem:*** *Show the logic circuit for;* $Y = A\bar{B} + AB$

***Solution:***



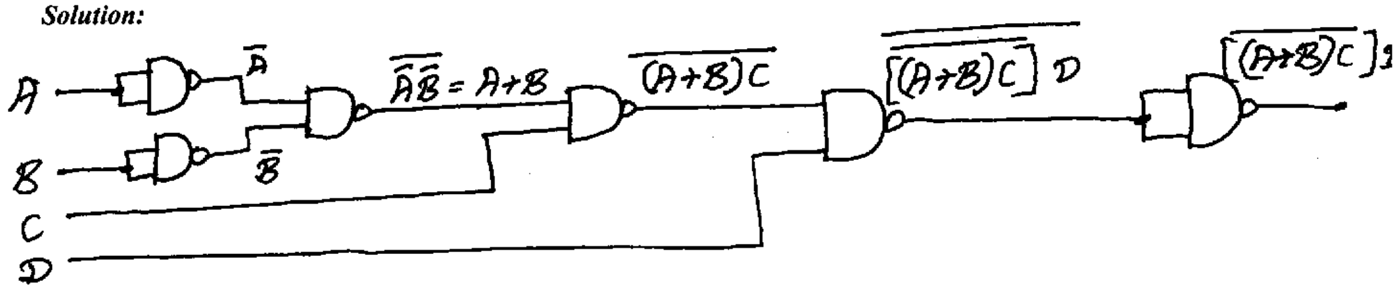
***Problem:*** *Show the logic circuit for;* $Y = (\bar{A} + B)(A + \bar{B})$ *and simplify.*

***Solution:***



***Problem:*** *Implement the following function using only NAND gates:* $\overline{((A + B)C))}D$.

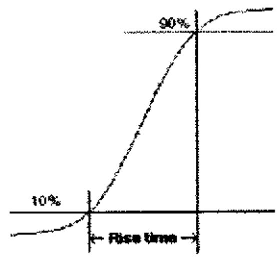
***Solution:***


A
B
C
D
$\bar{A}$
$\bar{B}$
$\overline{\bar{A}\bar{B}} = A+B$
$\overline{(A+B)C}$
$\overline{\overline{[(A+B)C]}D}$
$\overline{[(A+B)C]}$

**NOTE:** The signal changes between the logic levels are not instantaneous but take an amount of time.

The time taken for the signal voltage to rise from low-level to a high-level is called *rise time, $t_r$*.

The time for the signal voltage to fall from a high-level to a low-level is called *fall time, $t_f$*.



## POSITIVE AND NEGATIVE LOGIC:

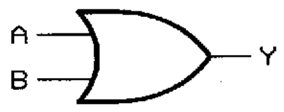
If we use a binary 0 for low voltage and binary 1 for high voltage, then it is called *positive logic*. Choosing H = 1 = T and L = 0 = F is called positive logic. If we use a binary 0 for high voltage and binary 1 for low voltage, then it is called *negative logic*. Choosing H = 0 = F and L = 1 = T is called negative logic.

**Positive and Negative Gates:**

In a positive logic system, binary 0 stands for low and binary 1 for high. Consider the following Table. Note that, Y is 1 if either A or B is 1. This is OR gate; and it is because, we are using positive logic.

| A | B | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Positive OR



Negative AND

| A | B | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

| A | B | Y |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

In a negative logic system, binary 1 stands for low and binary 0 for high. With this code we can convert the 1st Table to 3rd Table. Here, the output Y is 1, only when both A and B are 1. This is AND gate; and it is because, we are using negative logic. Hence, gates are defined by the way they process the binary 0s and 1s.

In the similar way, we can find the following equivalences between the positive and negative logic:

| | | |
|---|---|---|
| Positive OR | ↔ | Negative AND |
| Positive AND | ↔ | Negative OR |
| Positive NOR | ↔ | Negative NAND |
| Positive NAND | ↔ | Negative NOR |

**Assertion-Level Logic:**

Many designers draw logic circuits with bubbles on all pins with active-low signals or omit bubbles on all pins with active-high signals. This use of bubbles with active-low signals is called *assertion-level logic*. It means that you draw chips with the kind of input that causes something to happen, or with the kind of output that indicates something has happened. If a low input turns on a chip, you show a bubble on that input. If a low output is a sign of chip action, you draw a bubble on that output. You can equate the word assert with activate.

E.g.: The 74150 Multiplexer has an active low input STROBE; this input turns on the chip only when it is low. This is an active-low signal; which causes something to happen when it is low, rather than high.

## INTRODUCTION TO HDL:

Hardware Description Language (HDL) – a textual description of a digital circuit – a language which is more crisp, and machine readable.

*Advantages:*

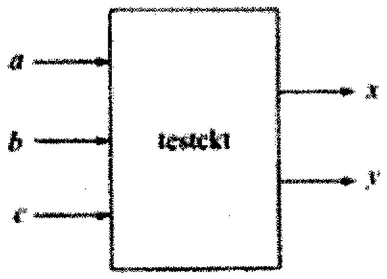
1. To describe a large complex design requiring hundreds of logic gates in a convenient manner, in a smaller space
2. To use software test bench to detect functional error, if any and correct it (called *simulation*)
3. To get hardware implementation details (called *synthesis*)

There are two widely used HDLs – Verilog and Very high speed integrated circuit Hardware Description Language (VHDL). Verilog is considered simple of the two and is more popular.

**Verilog HDL:**

Verilog, introduced in 1980, as a simulation and verification tool by *Gateway Design Associations*, later acquired by *Cadence Data Systems*. Put to public domain in 1990, and is now controlled by a group of companies and universities, called *Open Verilog International.*

**Describing Input Output:** In any digital circuit, there will be a set inputs and a set of outputs, often termed as *ports*. The relationship between these inputs and outputs are explained within the digital circuit. To design any circuit, that has (say) three inputs a, b, c and two outputs x, y as shown in the following Fig; the corresponding Verilog code can be written as follows:



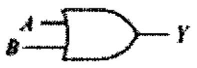
```
module testckt(x,y,a,b,c); //module name with port list

input  a,b,c; //defines input ports
output x,y; //defines output ports

//module body begins next describing logic relation
.
.
//module body ends
endmodule
```

Note that, *module* and *endmodule* are keywords for Verilog. A *module* describes a design entity with a name or identifier selected by user (here it is *testckt*) followed by input output port list. The symbol "//" is used to put comments and improve readability for a human. The module body describes the logic within the black box which acts on the inputs a, b, c and generates output x, y. Observe, where semicolon ";" is used and where not to end the statement.

**Writing Module Body:** There are three different models of writing module body in Verilog HDL – Structural, Data flow, and Behavioral.

**Structural Modeling:**

```
module or_gate(A,B,Y);
input A,B;     // defines two input port
output Y;      // defines one output port
or g1(Y,A,B);   /*Gate declaration with predefined keyword or representing
                  logic OR, g1 is optional user defined gate identifier */
endmodule
```

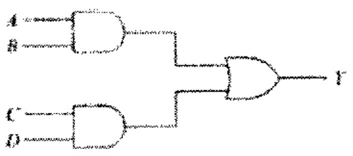


Verilog supports predefined gate level primitives such as and, or, not, nor, nand, xor, xnor, etc. The syntax followed above can be extended to other gates.

| | |
|---|---|
| For NOT Gate: | not (output, input) |
| For 2 input OR Gate | or (output, input1, input2) |
| For 4 input OR Gate | or (output, input1, input2, input3, input4) |

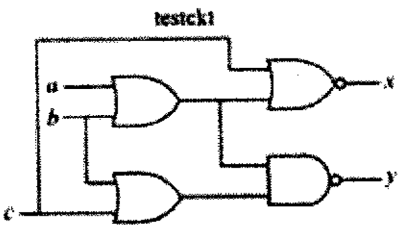
Note that, Verilog can take up to 12 inputs for logic gates. Comments when extended to the next line is written within /* . . . . */. Identifiers in Verilog are case sensitive, begin with a letter or underscore and can be of any length. Observe the following:

```
module fig2_24a(A,B,C,D,Y);
  input A,B,C,D;
  output Y;
  wire and_op1, and_op2;  // internal connections
  and g1(and_op1,A.B); // g1 represents upper AND gate
  and g2(and_op2,C,D); // g2 represents lower AND gate
  or g3(Y,and_op1,and_op2); // g3 represents the OR gate
endmodule
```



Note that, we define two intermediate variables; and_op1 and and_op2, representing two AND gate outputs through keyword *wire*. Wire represents a physical wire in a circuit.

Now, write the Verilog code for the testckt (shown below):



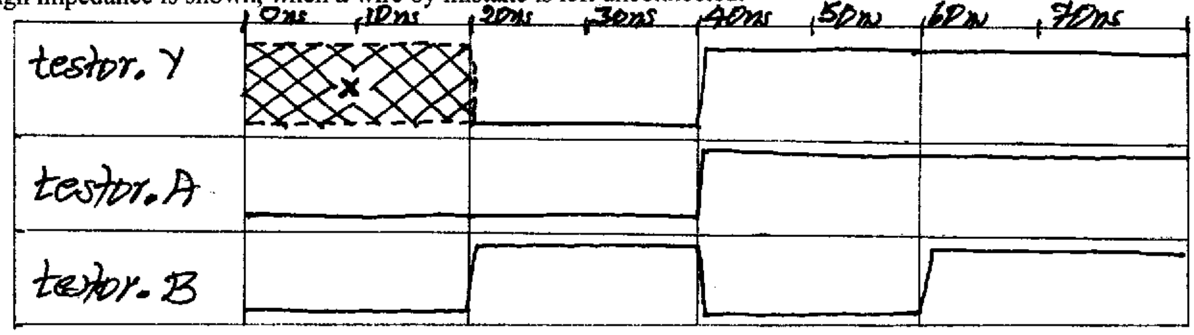
```
module testckt(a,b,c,x,y);
  input a,b,c;
  output x,y;
  wire or_op1, or_op2; /* internal
    connections, outputs of upper and
    lower OR gates respectively */
  or g1(or_op1,a,b); // g1 represents upper OR gate
  or g2(or_op2,b,c); // g2 represents lower OR gate
  nor g3(x,c,or_op1); // g3 represents the NOR gate
  nand g4(y,or_op1,or_op2); // g4 represents the NAND gate
endmodule
```

**Preparation of Test Bench:** A test bench in Verilog is used to simulate a digital circuit. Consider the example of simulating a simple OR gate, for which Verilog code is described. The test bench creates the input in the form of a timing waveform and passes this to OR gate module through a function or procedural call. To generate timing waveform, we use the time delay available in Verilog in the form of *#n* where *n* denotes a number in decimal that gives delay in nanosecond; E.g.: #20. (NOTE: All practical logic circuit comes with finite gate delay, i.e., output changes according to input after certain time).

```
module testor;              // Simulation module given a name testor
reg A, B;                   // Storage of data for passing it to module OR_Gate
wire X;
OR_Gate org (A, B, Y);      // Circuit is instantiated with name OR_Gate
initial                     // Start simulation
      begin                 /* Input is generated to test the circuit through following
statements, simulation begins */
            A = 1'b0; B = 1'b0;     // 1'b0 signifies on binary digit with value 0, AB = 00
            # 20
            A = 1'b0; B = 1'b1;     // After 20 ns, AB = 01
            # 20
            A = 1'b1; B = 1'b0;     // After 20 ns, AB = 10
            # 20
            A = 1'b0; B = 1'b1;     // After 20 ns, AB = 11
            # 20
      end
endmodule
```

```
module OR_Gate (A, B, Y);     // OR gate used as procedure in simulation
        input A, B;           // Define two input ports (for two input OR gate)
        output Y;             // Define one output port (OR gate output)
        or # (20) g1(Y, A, B); /* Gate declaration with a gate delay of 20 ns; output will be
effected after 20 ns */
endmodule
```

The input AB given by testor is taking values 00, 01, 10, 11 and retain them for 20 ns. Output of OR gate changes according to input but after a delay of 20 ns. For first 20 ns, OR gate output is unknown, as it needs 20 ns (gate delay) to respond. Note that, Verilog, in general offers four logic values in simulation: *0, 1, x* (unknown), and *z* (high impedance). Unknown value is exhibited, when the input is ambiguous and high impedance is shown, when a wire by mistake is left unconnected.

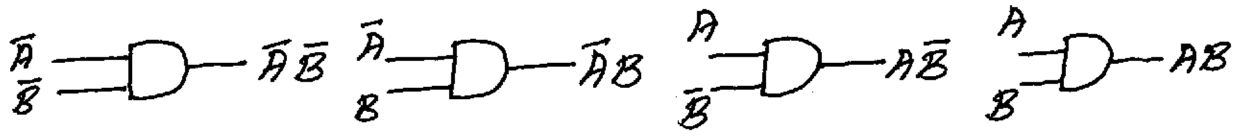


Verilog Simulation of 2 input OR gate with 20 ns given Delay

## COMBINATIONAL LOGIC CIRCUITS

### SUM-OF-PRODUCTS (SOP) METHOD:

The following Fig shows four possible ways to AND two input signals that are in complemented and un-complemented form. These outputs are called *fundamental products*.



The following Table lists each fundamental product next to the input conditions producing a high output.

| A | B | Fundamental Products |
|---|---|---|
| 0 | 0 | $\bar{A}\bar{B}$ |
| 0 | 1 | $\bar{A}B$ |
| 1 | 0 | $A\bar{B}$ |
| 1 | 1 | $AB$ |

| A | B | C | Fundamental Products |
|---|---|---|---|
| 0 | 0 | 0 | $\bar{A}\bar{B}\bar{C}$ |
| 0 | 0 | 1 | $\bar{A}\bar{B}C$ |
| 0 | 1 | 0 | $\bar{A}B\bar{C}$ |
| 0 | 1 | 1 | $\bar{A}BC$ |

| | | | | | |
|---|---|---|---|---|---|
| | | 1 | 0 | 0 | $A\bar{B}\bar{C}$ |
| | | 1 | 0 | 1 | $A\bar{B}C$ |
| | | 1 | 1 | 0 | $AB\bar{C}$ |
| | | 1 | 1 | 1 | $ABC$ |

The fundamental products are also called *minterms*. Products are represented by m0, m1, m2, and m3 respectively.

**Sum-of-Products Equation:** Given a truth table, to get the sum-of-products solution –

1. Locate each output 1 in the truth table and write down the fundamental products
2. Identify all the fundamental products
3. OR the fundamental product.

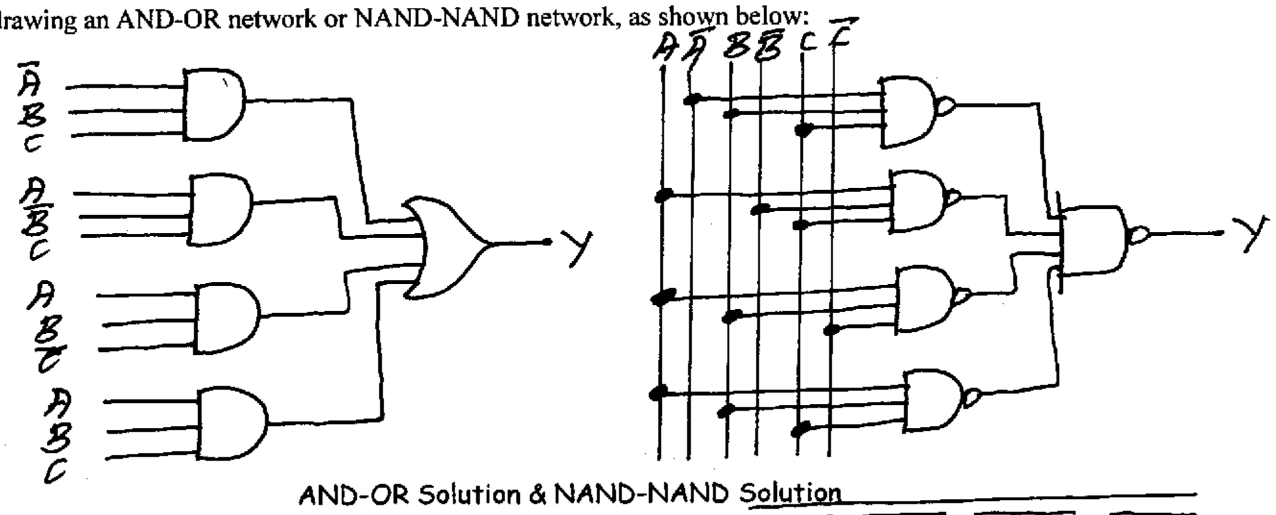
***Problem:*** *For the following truth table, get the sum-of-products solution.*

| A | B | C | Y |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

***Solution:***

| A | B | C | Y |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 $\longrightarrow \bar{A}BC$ |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 $\longrightarrow A\bar{B}C$ |
| 1 | 1 | 0 | 1 $\longrightarrow AB\bar{C}$ |
| 1 | 1 | 1 | 1 $\longrightarrow ABC$ |

**Logic Circuit:** After getting a sum-of-products equation, derive the corresponding logic circuit by drawing an AND-OR network or NAND-NAND network, as shown below: $\longrightarrow$



AND-OR Solution & NAND-NAND Solution

$$Y = \overline{\overline{\bar{A}BC}\;\overline{A\bar{B}C}\;\overline{AB\bar{C}}\;\overline{ABC}}$$

$$= \bar{A}BC + A\bar{B}C + \text{[illegible]} + ABC.$$

**Problem:** *Simplify the Boolean equation:* $Y = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + A\bar{B}\bar{C} + AB\bar{C}$

***Solution:***

$$
\begin{aligned}
Y &= \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + A\bar{B}\bar{C} + AB\bar{C} \\
&= \bar{C}\left[\bar{A}\bar{B} + \bar{A}B + A\bar{B} + AB\right] \\
&= \bar{C}\left[\bar{A}(\bar{B}+B) + A(\bar{B}+B)\right] \\
&= \bar{C}\left[\bar{A}\cdot 1 + A\cdot 1\right] = \bar{C}\cdot 1 = \bar{C}.
\end{aligned}
$$

| Y | $\bar{C}$ | $C$ |
|---|---|---|
| $\bar{A}\bar{B}$ | 1 | 0 |
| $\bar{A}B$ | 1 | 0 |
| $AB$ | 1 | 0 |
| $A\bar{B}$ | 1 | 0 |

$$\underline{\underline{Y = \bar{C}}}$$

**TRUTH TABLE TO KARNAUGH MAP (K-MAP):**

A *Karnaugh Map* is a visual display of the fundamental products needed for a sum-of-products solution.

**Two-Variable Map:**

| A | B | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

| Y | $\bar{B}$ | $B$ |
|---|---|---|
| $\bar{A}$ | 0 | 0 |
| $A$ | 1 | 1 |

**Three-Variable Map:**

| A | B | C | Y |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

| Y | $\bar{C}$ | $C$ |
|---|---|---|
| $\bar{A}\bar{B}$ | 0 | 0 |
| $\bar{A}B$ | 1 | 0 |
| $AB$ | 1 | 1 |
| $A\bar{B}$ | 0 | 0 |

**Four-Variable Map:**

| A | B | C | D | Y | A | B | C | D | Y |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |

| Y | $\bar{C}\bar{D}$ | $\bar{C}D$ | $CD$ | $C\bar{D}$ |
|---|---|---|---|---|
| $\bar{A}\bar{B}$ | 0 | 1 | 0 | 0 |
| $\bar{A}B$ | 0 | 0 | 1 | 1 |
| $AB$ | 0 | 0 | 0 | 1 |
| $A\bar{B}$ | 0 | 0 | 0 | 0 |

## IRS, QUADS, AND OCTETS:

**Pairs:** The following K-map contains a pair of 1s that are horizontally adjacent. Two adjacent 1s, such as these are called a *pair. A pair eliminates one variable and its complement.*

| $Y$ | $\bar{C}\bar{D}$ | $\bar{C}D$ | $CD$ | $C\bar{D}$ |
|---|---|---|---|---|
| $\bar{A}\bar{B}$ | 0 | 0 | 0 | 0 |
| $\bar{A}B$ | 0 | 0 | 0 | 0 |
| $AB$ | 0 | 0 | 1 | 1 |
| $A\bar{B}$ | 0 | 0 | 0 | 0 |

The sum of products eqn is:

$$Y = ABCD + ABC\bar{D}$$
$$= ABC(D+\bar{D})$$
$$= ABC.$$

**Quad:** A *quad* is a group of four 1s that are horizontally or vertically adjacent. *A quad eliminates two variables and their complements.*

| $Y$ | $\bar{C}\bar{D}$ | $\bar{C}D$ | $CD$ | $C\bar{D}$ |
|---|---|---|---|---|
| $\bar{A}\bar{B}$ | 0 | 0 | 0 | 0 |
| $\bar{A}B$ | 0 | 0 | 0 | 0 |
| $AB$ | 1 | 1 | 1 | 1 |
| $A\bar{B}$ | 0 | 0 | 0 | 0 |

$$Y = AB\bar{C} + ABC$$
$$= AB(\bar{C}+C)$$
$$= AB.$$

**The Octet:** The *octet* is a group of eight 1s, as shown in the following Fig. *An octet eliminates three variables and their complements.*

| $Y$ | $\bar{C}\bar{D}$ | $\bar{C}D$ | $CD$ | $C\bar{D}$ |
|---|---|---|---|---|
| $\bar{A}\bar{B}$ | 0 | 0 | 0 | 0 |
| $\bar{A}B$ | 0 | 0 | 0 | 0 |
| $AB$ | 1 | 1 | 1 | 1 |
| $A\bar{B}$ | 1 | 1 | 1 | 1 |

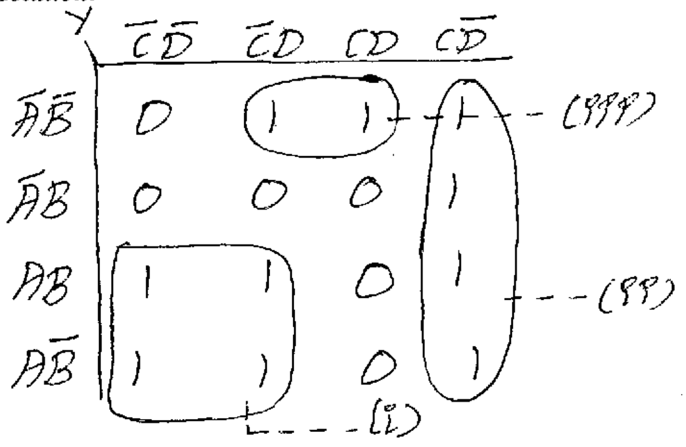
$$Y = AB + A\bar{B}$$
$$= A(B+\bar{B})$$
$$= A$$

## KARNAUGH SIMPLIFICATIONS:

*A pair eliminates one variable and its complement. A quad eliminates two variables and their complements. An octet eliminates three variables and their complements.* Because of this, after drawing the K-map, first encircle the octets, then the quads, and finally the pairs. Hence, the greatest simplification results.

***Problem:*** *Using K-map, simplify;* $Y = \sum m\,(1, 2, 3, 6, 8, 9, 10, 12, 13, 14)$.

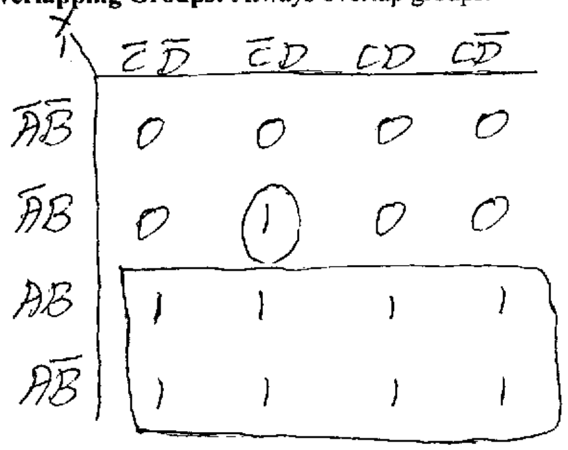
***Solution:***



| $Y$ | $\bar{C}\bar{D}$ | $\bar{C}D$ | $CD$ | $C\bar{D}$ |
|---|---|---|---|---|
| $\bar{A}\bar{B}$ | 0 | 1 | 1 | 1 |
| $\bar{A}B$ | 0 | 0 | 0 | 1 |
| $AB$ | 1 | 1 | 0 | 1 |
| $A\bar{B}$ | 1 | 1 | 0 | 1 |

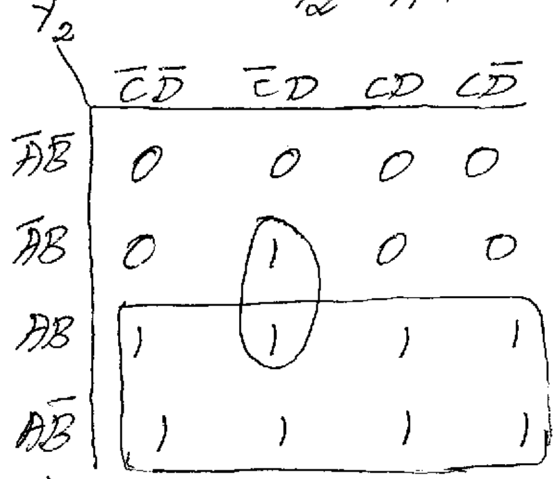
$Y = A\bar{C} + C\bar{D} + \bar{A}\bar{B}D.$

**Overlapping Groups:** Always overlap groups.



| $Y_1$ | $\bar{C}\bar{D}$ | $\bar{C}D$ | $CD$ | $C\bar{D}$ |
|---|---|---|---|---|
| $\bar{A}\bar{B}$ | 0 | 0 | 0 | 0 |
| $\bar{A}B$ | 0 | 1 | 0 | 0 |
| $AB$ | 1 | 1 | 1 | 1 |
| $A\bar{B}$ | 1 | 1 | 1 | 1 |

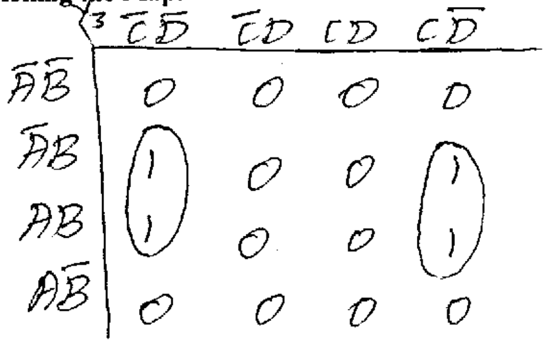
$Y_1 = A + \bar{A}B\bar{C}D$



| $Y_2$ | $\bar{C}\bar{D}$ | $\bar{C}D$ | $CD$ | $C\bar{D}$ |
|---|---|---|---|---|
| $\bar{A}\bar{B}$ | 0 | 0 | 0 | 0 |
| $\bar{A}B$ | 0 | 1 | 0 | 0 |
| $AB$ | 1 | 1 | 1 | 1 |
| $A\bar{B}$ | 1 | 1 | 1 | 1 |

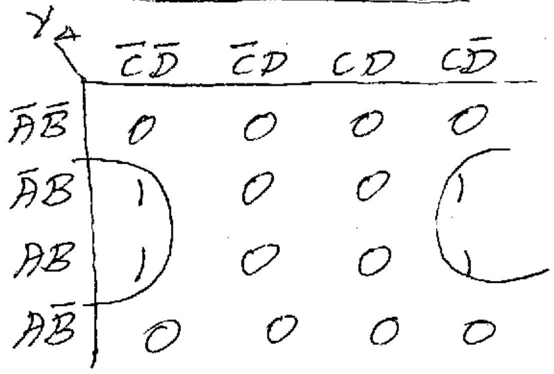
$Y_2 = A + B\bar{C}D.$

**Rolling the Map:**



| $Y_3$ | $\bar{C}\bar{D}$ | $\bar{C}D$ | $CD$ | $C\bar{D}$ |
|---|---|---|---|---|
| $\bar{A}\bar{B}$ | 0 | 0 | 0 | 0 |
| $\bar{A}B$ | 1 | 0 | 0 | 1 |
| $AB$ | 1 | 0 | 0 | 1 |
| $A\bar{B}$ | 0 | 0 | 0 | 0 |

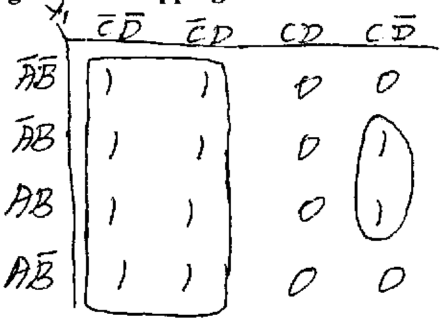
$Y_3 = B\bar{C}\bar{D} + BC\bar{D}$



| $Y_4$ | $\bar{C}\bar{D}$ | $\bar{C}D$ | $CD$ | $C\bar{D}$ |
|---|---|---|---|---|
| $\bar{A}\bar{B}$ | 0 | 0 | 0 | 0 |
| $\bar{A}B$ | 1 | 0 | 0 | 1 |
| $AB$ | 1 | 0 | 0 | 1 |
| $A\bar{B}$ | 0 | 0 | 0 | 0 |

$Y_4 = B\bar{D}.$

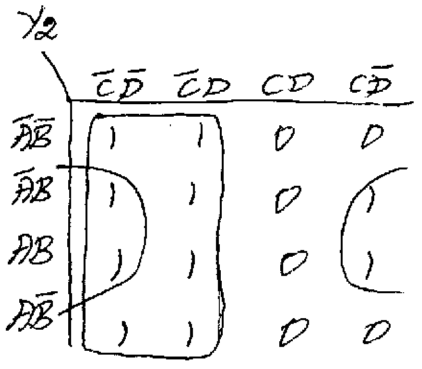
**Rolling and Overlapping:**



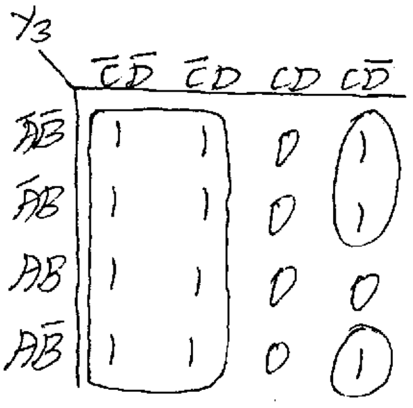
| $Y_1$ | $\bar{C}\bar{D}$ | $\bar{C}D$ | $CD$ | $C\bar{D}$ |
|---|---|---|---|---|
| $\bar{A}\bar{B}$ | 1 | 1 | 0 | 0 |
| $\bar{A}B$ | 1 | 1 | 0 | 1 |
| $AB$ | 1 | 1 | 0 | 1 |
| $A\bar{B}$ | 1 | 1 | 0 | 0 |

$Y_1 = \bar{C} + BC\bar{D}$

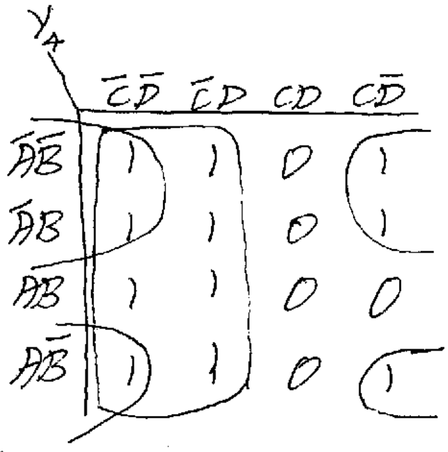
$Y_2 = \bar{C} + B\bar{D}$



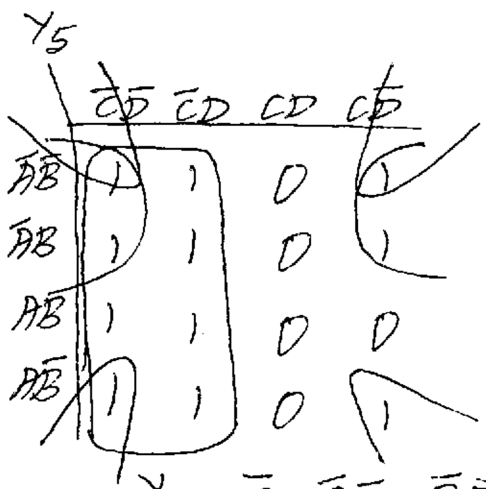
| $Y_2$ | $\bar{C}\bar{D}$ | $\bar{C}D$ | $CD$ | $C\bar{D}$ |
|---|---|---|---|---|
| $\bar{A}\bar{B}$ | 1 | 1 | 0 | 0 |
| $\bar{A}B$ | 1 | 1 | 0 | 1 |
| $AB$ | 1 | 1 | 0 | 1 |
| $A\bar{B}$ | 1 | 1 | 0 | 0 |



| $Y_3$ | $\bar{C}\bar{D}$ | $\bar{C}D$ | $CD$ | $C\bar{D}$ |
|---|---|---|---|---|
| $\bar{A}\bar{B}$ | 1 | 1 | 0 | 1 |
| $\bar{A}B$ | 1 | 1 | 0 | 1 |
| $AB$ | 1 | 1 | 0 | 0 |
| $A\bar{B}$ | 1 | 1 | 0 | 1 |



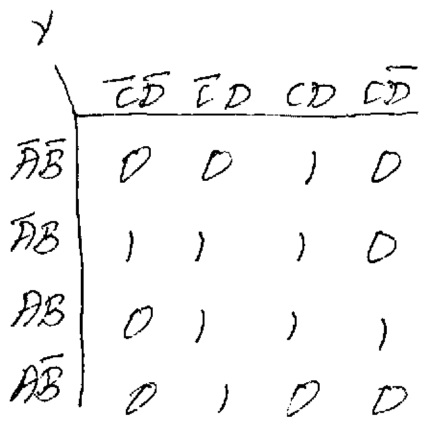
| $Y_4$ | $\bar{C}\bar{D}$ | $\bar{C}D$ | $CD$ | $C\bar{D}$ |
|---|---|---|---|---|
| $\bar{A}\bar{B}$ | 1 | 1 | 0 | 1 |
| $\bar{A}B$ | 1 | 1 | 0 | 1 |
| $AB$ | 1 | 1 | 0 | 0 |
| $A\bar{B}$ | 1 | 1 | 0 | 1 |



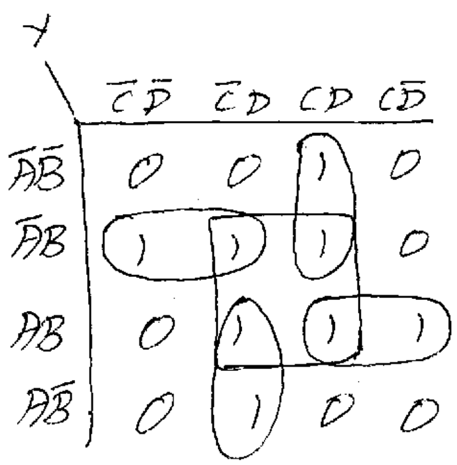
| $Y_5$ | $\bar{C}\bar{D}$ | $\bar{C}D$ | $CD$ | $C\bar{D}$ |
|---|---|---|---|---|
| $\bar{A}\bar{B}$ | 1 | 1 | 0 | 1 |
| $\bar{A}B$ | 1 | 1 | 0 | 1 |
| $AB$ | 1 | 1 | 0 | 0 |
| $A\bar{B}$ | 1 | 1 | 0 | 1 |

$Y_3 = \bar{C} + \bar{A}C\bar{D} + A\bar{B}C\bar{D}$ $\quad Y_4 = \bar{C} + \bar{A}\bar{D} + A\bar{B}\bar{D}$ $\quad Y_5 = \bar{C} + \bar{A}\bar{D} + \bar{B}\bar{D}$

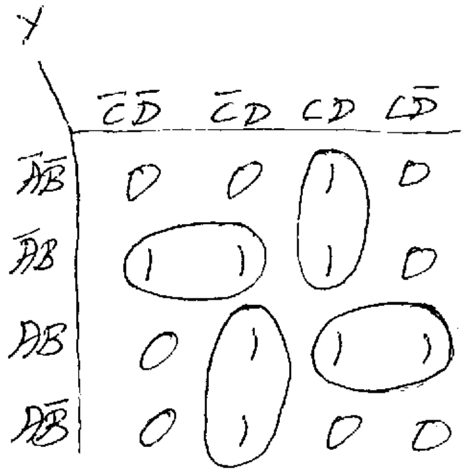
**Eliminating Redundant Groups:** After encircling groups, eliminate any *redundant groups*. This is a group whose 1s are already used by other groups.



| $Y$ | $\bar{C}\bar{D}$ | $\bar{C}D$ | $CD$ | $C\bar{D}$ |
|---|---|---|---|---|
| $\bar{A}\bar{B}$ | 0 | 0 | 1 | 0 |
| $\bar{A}B$ | 1 | 1 | 1 | 0 |
| $AB$ | 0 | 1 | 1 | 1 |
| $A\bar{B}$ | 0 | 1 | 0 | 0 |



| $Y$ | $\bar{C}\bar{D}$ | $\bar{C}D$ | $CD$ | $C\bar{D}$ |
|---|---|---|---|---|
| $\bar{A}\bar{B}$ | 0 | 0 | 1 | 0 |
| $\bar{A}B$ | 1 | 1 | 1 | 0 |
| $AB$ | 0 | 1 | 1 | 1 |
| $A\bar{B}$ | 0 | 1 | 0 | 0 |



| $Y$ | $\bar{C}\bar{D}$ | $\bar{C}D$ | $CD$ | $C\bar{D}$ |
|---|---|---|---|---|
| $\bar{A}\bar{B}$ | 0 | 0 | 1 | 0 |
| $\bar{A}B$ | 1 | 1 | 1 | 0 |
| $AB$ | 0 | 1 | 1 | 1 |
| $A\bar{B}$ | 0 | 1 | 0 | 0 |

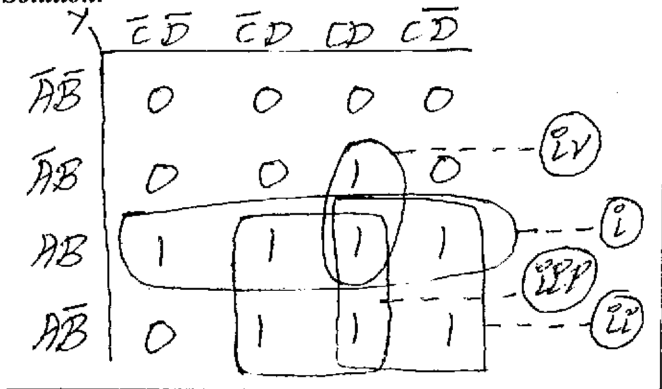
**Conclusion:** The summary of the K-map method for simplifying Boolean equation:

1. Enter a 1 on the K-map for each fundamental product that produces a 1 output in the truth table. Enter 0s elsewhere.
2. Encircle the octets, quads, and pairs. Remember to roll and overlap to get the largest groups possible.
3. If any isolated 1s remain, encircle each.
4. Eliminate any redundant group.
5. Write the Boolean equation by ORing the products corresponding to the encircled groups.

***Problem:*** *What is the simplified Boolean equation for the following equation expressed by minterms?*

$Y = F(A, B, C, D) = \sum M(7, 9, 10, 11, 12, 13, 14, 15)$. $Y = AB + AC + AD + BCD$.

***Solution:***



| Y | $\bar{C}\bar{D}$ | $\bar{C}D$ | $CD$ | $C\bar{D}$ |
|---|---|---|---|---|
| $\bar{A}\bar{B}$ | 0 | 0 | 0 | 0 |
| $\bar{A}B$ | 0 | 0 | 1 | 0 |
| $AB$ | 1 | 1 | 1 | 1 |
| $A\bar{B}$ | 0 | 1 | 1 | 1 |

Let $Y = \sum m(2, 6, 7)$.

| A B C | Y |
|---|---|
| 0 0 0 | 0 |
| 0 0 1 | 0 |
| 0 1 0 | 1 |
| 0 1 1 | 0 |
| 1 0 0 | 0 |
| 1 0 1 | 0 |
| 1 1 0 | 1 |
| 1 1 1 | 1 |

| A B | $Y_C$ |
|---|---|
| 0 0 | 0 |
| 0 1 | $\bar{C}$ |
| 1 0 | 0 |
| 1 1 | 1 |

| $Y_C$ | $\bar{B}$ | $B$ |
|---|---|---|
| $\bar{A}$ | 0 | $\bar{C}$ |
| $A$ | 0 | 1 |

| B C | $Y_A$ |
|---|---|
| 0 0 | 0 |
| 0 1 | 0 |
| 1 0 | 1 |
| 1 1 | A |

| $Y_A$ | $\bar{C}$ | $C$ |
|---|---|---|
| $\bar{B}$ | 0 | 0 |
| $B$ | 1 | A |

**Entered Variable Map (EVM):** In EVM, one of the input variables is placed inside K-map. This reduces the K-map size by one degree; i.e. a three variable problem that requires $2^3 = 8$ locations in K-map will require $2^{(3-1)} = 4$ locations in EVM. This technique is particularly useful for mapping problems with more than four variables.

**Simplification of EVM:** This is similar to K-map method. In Fig (a), C' is grouped with 1 to get a larger group as 1 can be written ac 1 = 1 + C'. Similarly, A is grouped with 1 in Fig (b).

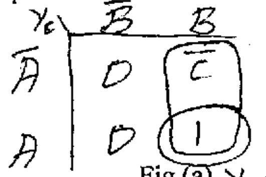


Fig (a) $Y_C = B\bar{C} + AB$

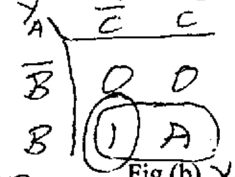


Fig (b) $Y_A = AB + B\bar{C}$

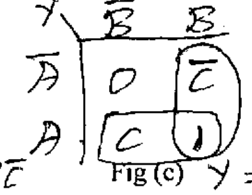


Fig (c) $Y = AB + B\bar{C}$

Now, the product term representing each group is obtained by including map entered variable (MEV) in the group as an additional ANDed term.

Hence, for Fig (a): $Y = B\bar{C} + AB$. For Fig (b): $Y = B\bar{C} + AB$.

Consider the EBM shown in Fig (c). This has only two product terms; and doesn't need a separate coverage for 1. This is because, one can write 1 = C + C', and C is included in one group and C' is included in other group.

***blem:** Simplify Y (A, B, C) = ∑m (2, 6, 7) by using entered variable map method by –*

*a) "A" as map entered variable*

*b) "C" as map entered variable.*

***Solution:***

**DON'T CARE CONDITIONS:**

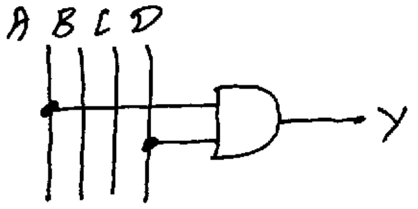
In some digital systems, certain input conditions never occur during normal operation; therefore, the corresponding output never appears. Since the output never appears, it is indicated by an X in the truth table. The X is called a *don't-care condition.*

Example: Consider the following truth table with don't care conditions for all the inputs from 1010 to 1111.

| ABCD | Y |
|---|---|
| 0000 | 0 |
| 0001 | 0 |
| 0010 | 0 |
| 0011 | 0 |
| 0100 | 0 |
| 0101 | 0 |
| 0110 | 0 |
| 0111 | 0 |
| 1000 | 0 |
| 1001 | 1 |
| 1010 | X |
| ⋮ | ⋮ |
| 1111 | X |

$$
Y \quad
\begin{array}{c|cccc}
 & \bar{C}\bar{D} & \bar{C}D & CD & C\bar{D} \\
\hline
\bar{A}\bar{B} & 0 & 0 & 0 & 0 \\
\bar{A}B & 0 & 0 & 0 & 0 \\
AB & \times & \times & \times & \times \\
A\bar{B} & 0 & 1 & \times & \times
\end{array}
$$

$Y = AD$



member these points about don't-care conditions:

1. Given the truth table, draw the K-map and transfer 0s, 1s, and don't-care terms.
2. Encircle the actual 1s on the K-map in the largest groups you can find treating don't cares as 1s.
3. After the actual1s have been included in the groups, disregard the remaining don't cares by visualizing them as 0s.

***Problem:*** *What is the simplest logic circuit for –*

*a)* $Y = F(A, B, C, D) = \sum m(0) + \sum d(8, 9, 10, 11, 14, 15)$

*b)* $Y = F(A, B, C, D) = \sum m(0) + \sum d(12, 13, 14, 15)$

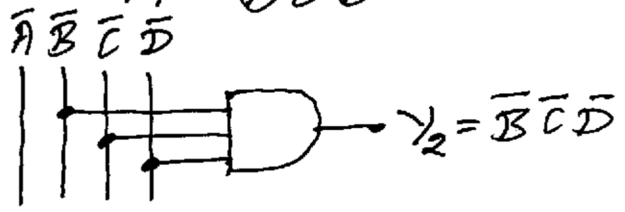
*c)* $Y = F(A, B, C, D) = \sum m(7) + \sum d(10, 11, 12, 13, 14, 15)$.

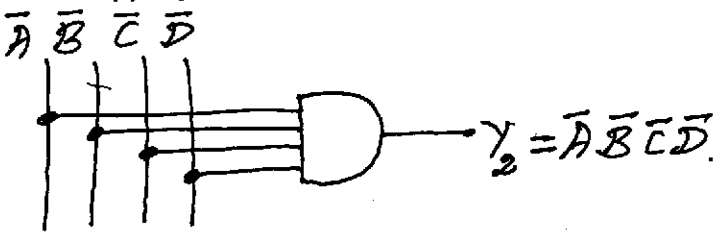
***Solution:***

ⓐ $Y = \sum m(0) + \sum d(8, 9, 10, 11, 14, 15)$

| $Y_1$ | $\bar{C}\bar{D}$ | $\bar{C}D$ | $CD$ | $C\bar{D}$ |
|---|---|---|---|---|
| $\bar{A}\bar{B}$ | 1 | 0 | 0 | 0 |
| $\bar{A}B$ | 0 | 0 | 0 | 0 |
| $AB$ | 0 | 0 | × | × |
| $A\bar{B}$ | × | × | × | × |

$Y_1 = \bar{B}\bar{C}\bar{D}$


$\bar{A}\ \bar{B}\ \bar{C}\ \bar{D}$
$Y_2 = \bar{B}\bar{C}\bar{D}$


$\bar{A}\ \bar{B}\ \bar{C}\ \bar{D}$
$Y_2 = \bar{A}\bar{B}\bar{C}\bar{D}$.

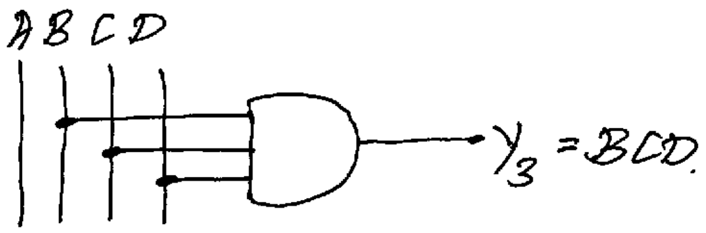
ⓑ $Y_2 = \sum(0) + \sum d(12, 13, 14, 15)$

| $Y_2$ | $\bar{C}\bar{D}$ | $\bar{C}D$ | $CD$ | $C\bar{D}$ |
|---|---|---|---|---|
| $\bar{A}\bar{B}$ | 1 | 0 | 0 | 0 |
| $\bar{A}B$ | 0 | 0 | 0 | 0 |
| $AB$ | × | × | × | × |
| $A\bar{B}$ | 0 | 0 | 0 | 0 |

$Y_2 = \bar{A}\bar{B}\bar{C}\bar{D}$.

ⓒ $Y_3 = \sum m(7) + \sum d(10, 11, 12, 13, 14, 15)$

| $Y_3$ | $\bar{C}\bar{D}$ | $\bar{C}D$ | $CD$ | $C\bar{D}$ |
|---|---|---|---|---|
| $\bar{A}\bar{B}$ | 0 | 0 | 0 | 0 |
| $\bar{A}B$ | 0 | 0 | 1 | 0 |
| $AB$ | × | × | × | × |
| $A\bar{B}$ | 0 | 0 | × | × |



$Y_3 = BCD$.

## PRODUCT-OF-SUMS (POS) METHOD:

With SOP method –

- A *fundamental product* produces an output *1* for the corresponding input condition.

With POS method –

- A *fundamental sum* produces and output *0* for the corresponding input condition.

**Product-of-Sums Equation:** In the following Table, the first output 0 appears for A = 0, B = 0, and C = 0. The fundamental sum for these inputs is A + B + C; because, this produces an output zero for the corresponding input condition: Y = A + B + C = 0 + 0 + 0 = 0.

| A | B | C | Y – Fundamental Sum | Max-term |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 – A + B + C | M0 |
| 0 | 0 | 1 | 1 | M1 |
| 0 | 1 | 0 | 1 | M2 |
| 0 | 1 | 1 | 0 – $A+\bar{B}+\bar{C}$ | M3 |
| 1 | 0 | 0 | 1 | M4 |
| 1 | 0 | 1 | 1 | M5 |
| 1 | 1 | 0 | 0 – $\bar{A}+\bar{B}+C$ | M6 |
| 1 | 1 | 1 | 1 | M7 |

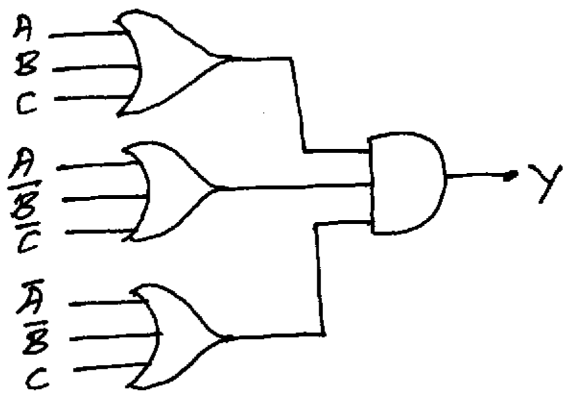
The second output 0 appears for the input condition A = 0, B = 1, and C = 1. The fundamental sum for this is A + B' + C'. Notice that, B and C are complemented because; this is the only way to get a logical sum of 0 for the given input condition: $Y = A+\bar{B}+\bar{C} = 0+\bar{1}+\bar{1} = 0+0+0 = 0$.

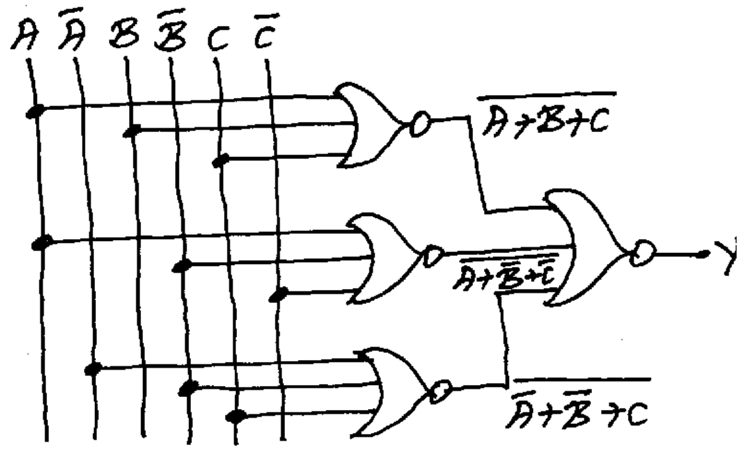
Similarly, the third output 0 occurs for A = 1, B = 1, and C = 0; hence, its fundamental sum is A' + B' + C: Y = $\bar{A}+\bar{B}+C = \bar{1}+\bar{1}+0 = 0+0+0 = 0$.

To get the POS equation, AND the fundamental sums:

$Y = (A+B+C)(A+\bar{B}+\bar{C})(\bar{A}+\bar{B}+C)$ OR Y = F (A, B, C) = ΠM (0, 3, 6)

**Logic Circuit:**





A $\bar{A}$ B $\bar{B}$ C $\bar{C}$
$\overline{A+B+C}$
$\overline{A+\bar{B}+\bar{C}}$
$\overline{\bar{A}+\bar{B}+C}$
Y

OR-AND Network & NOR-NOR Network

$$Y = \overline{\overline{A+B+C} + \overline{A+\bar{B}+\bar{C}} + \overline{\bar{A}+\bar{B}+C}}$$
$$= (A+B+C)(A+\bar{B}+\bar{C})(\bar{A}+\bar{B}+C)$$

In SOP method –

6. Given truth table
7. Identify 1s
8. Write the fundamental products
9. OR the fundamental products
10. AND-OR network or NAND-NAND network.

In POS method –

1. Given truth table
2. Identify 0s
3. Write the fundamental sums
4. AND the fundamental sums
5. OR-AND network or NOR-NOR network.

***Problem:*** *Write the POS and SOP representations for the following truth tables:*

| *A* | *B* | *C* | *Y* |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

***Solution:***

$$Y_{POS} = \Pi M(0,1,2,4)$$
$$= (A+B+C)(A+B+\bar{C})(A+\bar{B}+C)(\bar{A}+B+C)$$
$$Y_{SOP} = \Sigma m(3,5,6,7)$$
$$= \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC.$$

| *A* | *B* | *C* | *Y* |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

***Solution:***

$$Y_{POS} = \Pi M(0,3,6)$$
$$= (A+B+C)(A+\bar{B}+\bar{C})(\bar{A}+\bar{B}+C).$$
$$Y_{SOP} = \Sigma m(1,2,4,5,7)$$
$$= \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + A\bar{B}C + ABC.$$

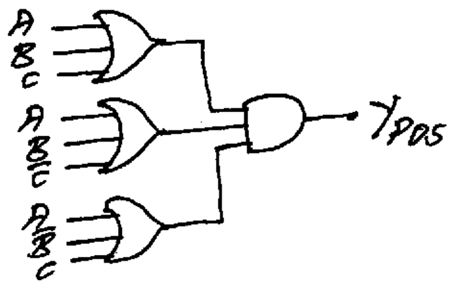
***Problem:*** *Suppose a truth table has a low output for the first three input conditions: 000, 001, and 010. If all other outputs are high, write POS and SOP circuits?*

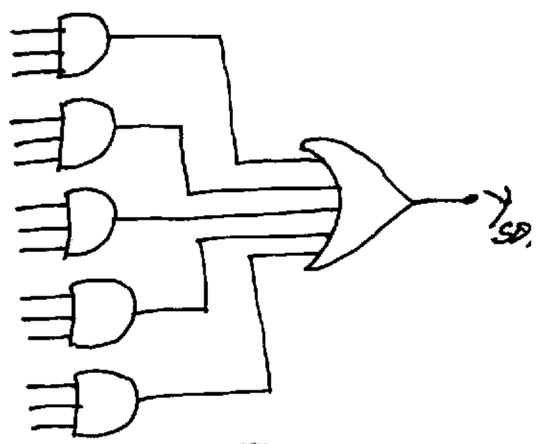
***Solution:***

| ABC | Y |
|---|---|
| 000 | 0 |
| 001 | 0 |
| 010 | 0 |
| 011 | 1 |
| 100 | 1 |
| 101 | 1 |
| 110 | 1 |
| 111 | 1 |

$$Y_{POS} = \Pi M(0,1,2)$$
$$= (A+B+C)(A+B+\bar{C})(A+\bar{B}+C).$$

$$Y_{SOP} = \Sigma m(3,4,5,6,7)$$
$$= \bar{A}BC + A\bar{B}\bar{C} + A\bar{B}C + AB\bar{C} + ABC.$$





## PRODUCT-OF-SUMS SIMPLIFICATION:

**Method 1:** After writing the POS equation, one can simplify by using Boolean algebra.

**Method 2:** After writing the POS equation, simplification can be done on the K-map.

- Forming largest group of zeros
- Replace each group by a sum term
- The variable going in the formation of sum term is inverted if it remains constant with a value of 1 in the group and it is not inverted if that value is 0
- Finally, all the sum terms are ANDed to get simplest POS form.

***Problem:*** *By grouping zeros, give the POS form of –*

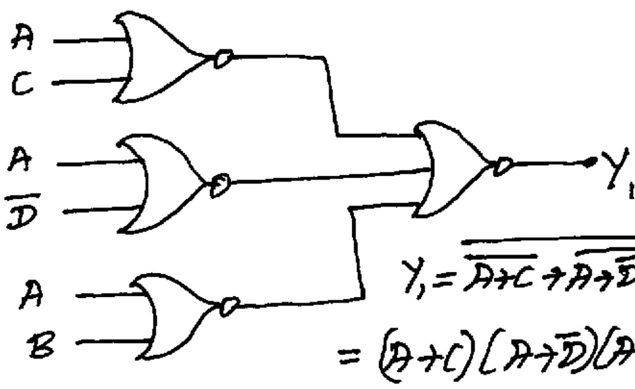
*a)* $Y_1 = F(A, B, C, D) = \Pi M(0, 1, 2, 3, 4, 5, 7)$

*b)* $Y_2 = F(A, B, C, D) = \Pi M(0, 1, 2, 4, 5, 10) + d(8, 9, 11, 12, 13, 15)$

***Solution:***

$Y_1 = \Pi M(0, 1, 2, 3, 4, 5, 7)$  $\therefore Y_1 = (A+C)(A+\bar{D})(A+B)$.

| $Y_1$ | $\bar{C}\bar{D}$ | $\bar{C}D$ | $CD$ | $C\bar{D}$ | |
|---|---|---|---|---|---|
| $\bar{A}\bar{B}$ | 0 | 0 | 0 | 0 | (iii) |
| $\bar{A}B$ | 0 | 0 | 0 | 1 | (i) (ii) |
| $AB$ | 1 | 1 | 1 | 1 | |
| $A\bar{B}$ | 1 | 1 | 1 | 1 | |



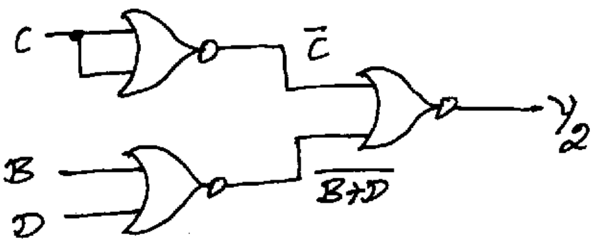
$Y_1 = \overline{\overline{A+C} + \overline{A+\bar{D}} + \overline{A+B}}$

$= (A+C)(A+\bar{D})(A+B)$

$Y_2 = \Pi M(0, 1, 2, 4, 5, 10) + d(8, 9, 11, 12, 13, 15)$.

$\therefore Y_2 = (C)(B+D)$.

| $Y_2$ | $\bar{C}\bar{D}$ | $\bar{C}D$ | $CD$ | $C\bar{D}$ | |
|---|---|---|---|---|---|
| $\bar{A}\bar{B}$ | 0 | 0 | 1 | 0 | |
| $\bar{A}B$ | 0 | 0 | 1 | 1 | (i) |
| $AB$ | × | × | × | 1 | |
| $A\bar{B}$ | × | × | × | 0 | (ii) |



$Y_2 = \overline{\bar{C} + \overline{B+D}}$

$= C(B+D)$.

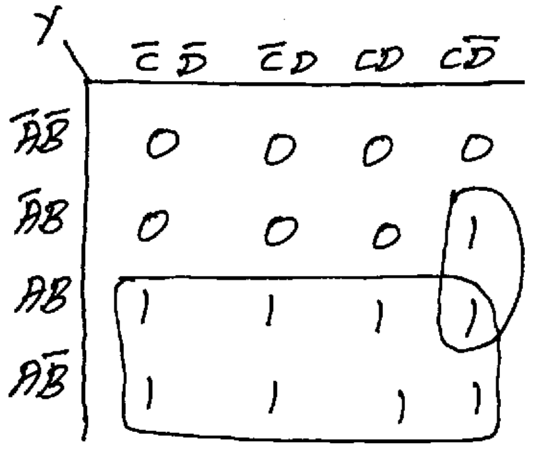
**Method 3:** Self Study.

***Problem:** Give the SOP and POS circuits for –*

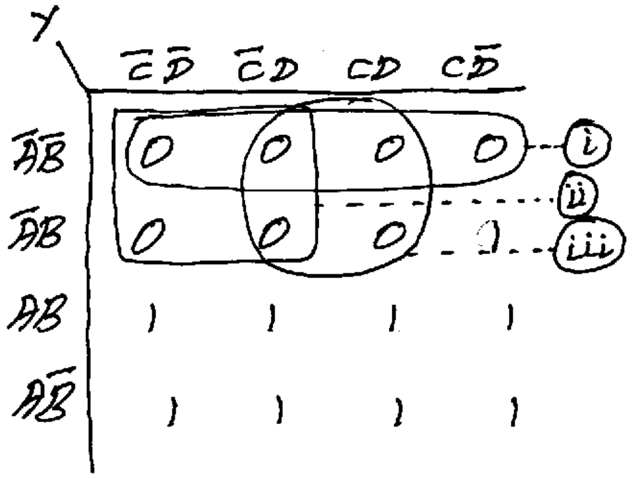
*$Y = F(A, B, C, D) = \sum m(6, 8, 9, 10, 11, 12, 13, 14, 15)$.*

***Solution:***

$Y = \sum m(6, 8, 9, 10, 11, 12, 13, 14, 15)$.



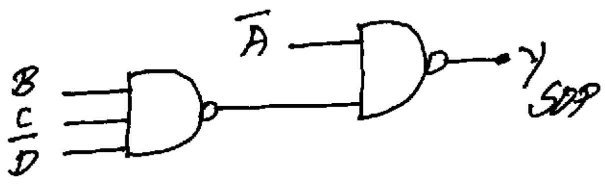
| $Y$ | $\bar{C}\bar{D}$ | $\bar{C}D$ | $CD$ | $C\bar{D}$ |
|---|---|---|---|---|
| $\bar{A}\bar{B}$ | 0 | 0 | 0 | 0 |
| $\bar{A}B$ | 0 | 0 | 0 | 1 |
| $AB$ | 1 | 1 | 1 | 1 |
| $A\bar{B}$ | 1 | 1 | 1 | 1 |

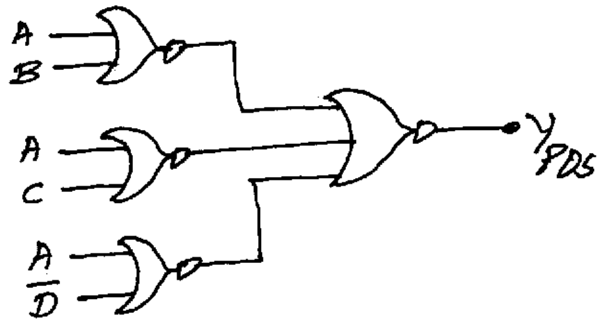


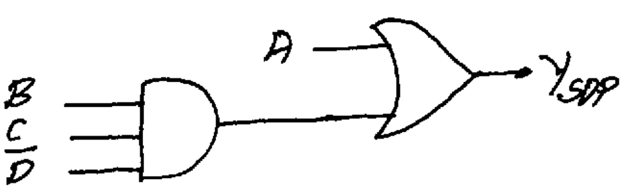
| $Y$ | $\bar{C}\bar{D}$ | $\bar{C}D$ | $CD$ | $C\bar{D}$ |
|---|---|---|---|---|
| $\bar{A}\bar{B}$ | 0 | 0 | 0 | 0 |
| $\bar{A}B$ | 0 | 0 | 0 | 1 |
| $AB$ | 1 | 1 | 1 | 1 |
| $A\bar{B}$ | 1 | 1 | 1 | 1 |

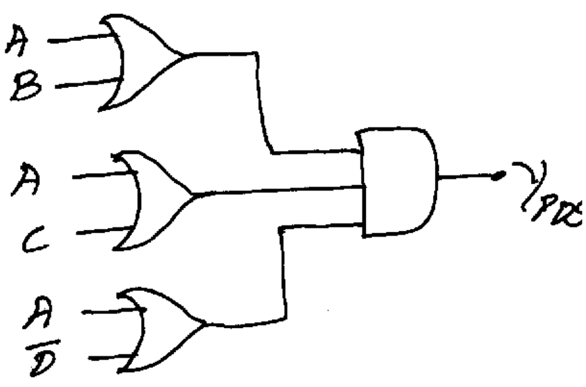
$Y_{SOP} = A + BC\bar{D}$.

$Y_{POS} = (A+B)(A+C)(A+\bar{D})$.









## SIMPLIFICATION BY QUINE-McCLUSKY (QM) METHOD:

*Reduction of logic equation by K-map method is very simple, but has some limitations:*

1. *It depends on the user's ability to identify patterns that gives largest size*
2. *The method becomes difficult to adapt for simplification of 5 or more variables.*

Quine-McClusky method is a systematic approach for logic simplification that does not have these limitations and also can easily be implemented in a digital computer.

**Determination of Prime Implicants:** QM method involves preparation of two tables – one determines *prime implicants* and the other selects *essential prime implicants* to get minimal expression. *Prime implicants* are expressions with least number of literals that represents all the terms given in a truth table. Prime implicants are examined to get essential prime implicants for a particular expression that avoids any type of duplication.

Consider the min-term expression Y = $\sum m$ (0, 1, 2, 3, 10, 11, 12, 13, 14, 15). The following truth table can be written based on the given min-term expression:

| A | B | C | D | Y |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

| Stage 1 | Stage 2 | Stage 3 |
|---|---|---|
| ABCD | ABCD | ABCD |
| 0000 (0) ✓ | 000_ (0,1) ✓ | 00_ _ (0,1,2,3) ● |
| 0001 (1) ✓ | 00_0 (0,2) ✓ | 00_ _ (0,2,1,3) |
| 0010 (2) ✓ | 00_1 (1,3) ✓ | _01_ (2,3,10,11) ● |
| 0011 (3) ✓ | 001_ (2,3) ✓ | _01_ (2,10,3,11) |
| 1010 (10) ✓ | _010 (2,10) ✓ | 1_1_ (10,11,14,15) ● |
| 1100 (12) ✓ | _011 (3,11) ✓ | 1_1_ (10,14,11,15) |
| 1011 (11) ✓ | 101_ (10,11) ✓ | 11_ _ (12,13,14,15) ● |
| 1101 (13) ✓ | 1_10 (10,14) ✓ | 11_ _ (12,14,13,15) |
| 1110 (14) ✓ | 110_ (12,13) ✓ | |
| 1111 (15) ✓ | 11_0 (12,14) ✓ | |
| | 1_11 (11,15) ✓ | |
| | 11_1 (13,15) ✓ | |
| | 111_ (14,15) ✓ | |

In Stage 1 of the process, we find out all the terms that gives output 1 from truth table. Put them in different groups depending on how many 1 input variable combinations (ABCD) have. For example, first group has no 1 in input combination, second group has only one 1, third two 1s, and fourth four 1s. We also write decimal equivalent of each combination to their right for convenience.

In Stage 2, we first try to combine first and second group of Stage 1, on a member to member basis. The rule is to see if only one binary digit is differing between two members and we mark that position by "–". This means corresponding variable is not required to represent those members. Thus (0) of first group combine with (1) of second group to form (0, 1) in Stage 2 and can be represented by A'B'C' (000–). The logic of this representation comes from the fact that, min-term A'B'C'D' (0000) and A'B'C'D (0001) can be combined as A'B'C'(D'+D) = A'B'C'. We proceed in the same manner to find rest of the combinations in successive groups of Stage 1 and table them. All the members of particular stage, which finds itself in at least one combination of next stage, are tick marked.

In Stage 3, we combine members of different groups of Stage 2 in a similar way. Now, it will have two "–" elements in each combination. This means each combination requires two literals to represent it. For example, (0, 1, 2, 3) is represented by A'B'(0 0 – –). There are three other groups in Stage 3: (2, 10, 3, 11) represented by B'C, (10, 14, 11, 15) by AC and (12, 13, 14, 15) by AB. Note that, (0, 2, 1, 3), (10, 11, 14, 15) and (12, 14, 13, 15) get represented by A'B, AC and AB respectively and do not give any new term.

There is no Stage 4 for this problem, as no two members of Stage 3 has only one digit changing among them. This completes the process of determination of prime implicants. The rule is all the terms that are not ticked at any stage is treated as prime implicants for that problem.

**Selection of Prime Implicants:** Now, we try to select essential prime implicants and remove redundancy or duplication among them. For this, we prepare a Table as shown below:

| | 0 | 1 | 2 | 3 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|
| $\bar{A}\bar{B}$ (0,1,2,3) | ✓ | ✓ | ✓ | ✓ | | | | | | |
| $\bar{B}C$ (2,3,10,11) | | | ✓ | ✓ | ✓ | ✓ | | | | |
| AC (10,11,14,15) | | | | | ✓ | ✓ | | | ✓ | ✓ |
| AB (12,13,14,15) | | | | | | | ✓ | ✓ | ✓ | ✓ |

Here, row lists all the prime implicants and columns lists all min-terms. The cross point of a row and column is ticked if the term is covered by corresponding prime implicants.

Now, find minimum number of prime implicants that covers all the min-terms. We find A'B' and AB cover terms that are not covered by others and they are essential prime implicants. B'C and AC among themselves cover 10, 11 which are not covered by others. So, one of them has to be included in the list of essential prime implicants.

Hence, we get; $Y = \bar{A}\bar{B} + \bar{B}C + AB$ or $Y = \bar{A}\bar{B} + AC + AB$

***Homework:*** *Solve the above problem by using K-map method.*

***Solution:***

***Homework:** Give simplified logic equation of $Y = \sum m\,(2, 6, 7)$ by Quine-McClusky method.*

***Solution:***

| A B C | Y |
|---|---|
| 0 0 0 | 0 |
| 0 0 1 | 0 |
| 0 1 0 | 1 |
| 0 1 1 | 0 |
| 1 0 0 | 0 |
| 1 0 1 | 0 |
| 1 1 0 | 1 |
| 1 1 1 | 1 |

| Stage 1<br>A B C | Stage 2<br>A B C |
|---|---|
| 0 1 0 (2) ✓ | – 1 0 (2, 6) |
| 1 1 0 (6) ✓ | 1 1 – (6, 7) |
| 1 1 1 (7) | |

| – | 2 | 6 | 7 |
|---|---|---|---|
| $B\bar{C}$ (2, 6) | ✓ | ✓ | |
| AB (6, 7) | | ✓ | ✓ |

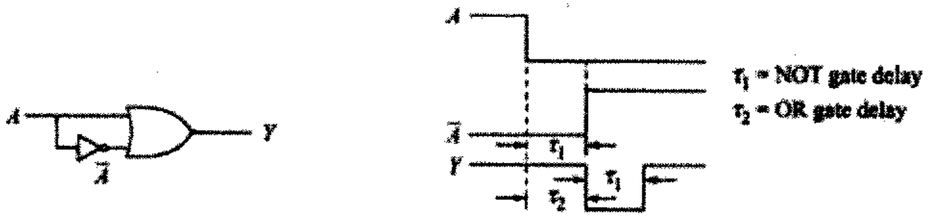
$\therefore Y = AB + B\bar{C}$.

**HAZARDS AND HAZARD COVERS:**

We have discussed various simplification techniques that give minimal expression for a logic equation, which in turn requires minimum hardware for realization. But, due to some practical problems, in certain cases, we may prefer to include more terms than given by simplification techniques. The discussion so far considered gates generating outputs instantaneously. But, practical circuits always offer finite propagation delay, though very small, in nanoseconds order.

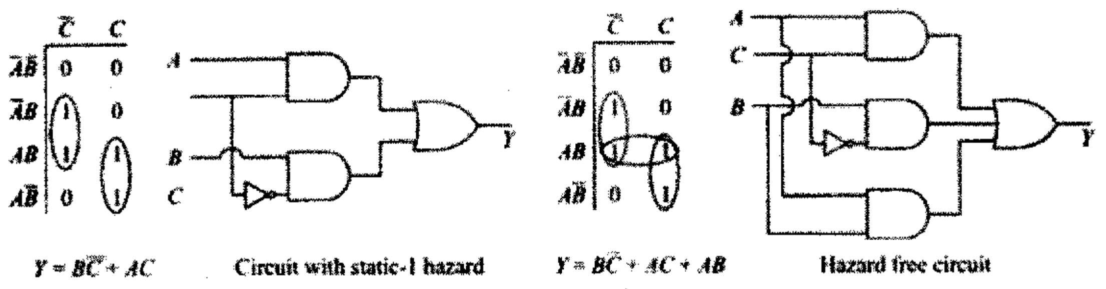
**Sattic-1 Hazard:**

This type of hazard occurs when Y = A + A' type of situation appear for a logic circuit and when A makes a transition 1 → 0. An A + A' condition should always generate 1 at the output (static-1). But, the NOT gate output (as shown in the following Fig) takes finite time to become 1 following 1 → 0 transition of A. Thus for the OR gate there are two zeros appearing at its input for a small duration, resulting a 0 at its output. The width of this zero is in nanoseconds and is called a *glitch*.



Static-1 Hazard

**To Cover Static-1 Hazard:** Refer to the K-map shown in the following Fig; represented by Y = BC' + AC. The corresponding circuit is also shown in the following Fig. For this circuit, if input B = 1, A = 1 and C makes a transition 1 → 0; the output shows glitch.



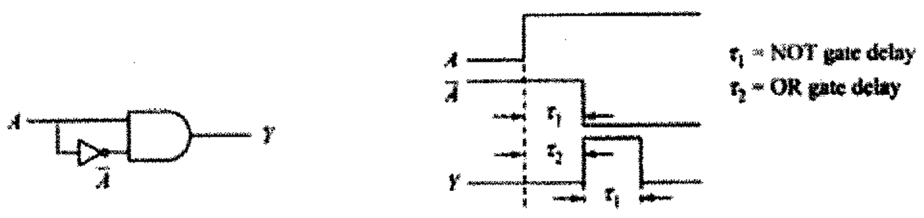
Static-1 Hazard & Its Cover

Consider another grouping for the same K-map (as shown in the above Fig). This includes one additional AND term, and now, output Y = BC' + AC + AB. The corresponding circuit diagram is also given. This circuit requires more hardware, but it is hazard free. The additional term AB ensures Y = 1 for B = 1, A = 1 and C makes a transition 1 → 0, does not affect output.

**NOTE:** A NAND gate with A and A' connected at its input for certain input combination will give static-1 hazard when A makes a transition 0 → 1 and requires hazard cover.

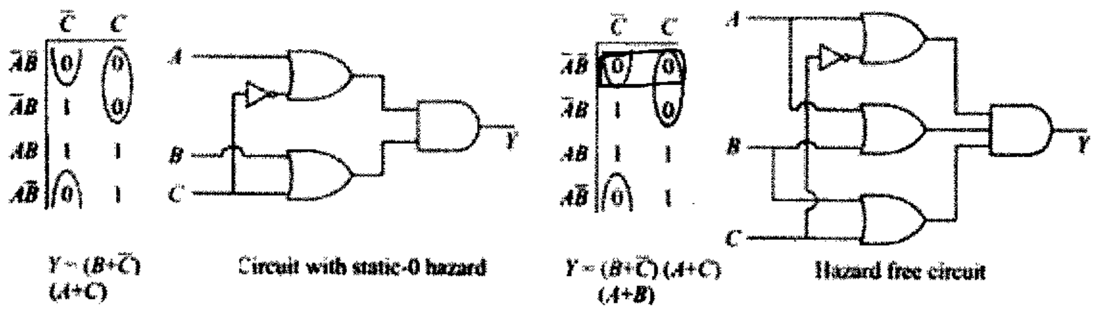
**Sattic-0 Hazard:**

This type of hazard occurs when Y = A.A' type of situation appear for a logic circuit and when A makes a transition 0 → 1. An A.A' condition should always generate 0 at the output (static-0). But, the NOT gate output (as shown in the following Fig) takes finite time to become 0 following 0 → 1 transition of A. Thus for the AND gate there are two ones appearing at its input for a small duration, resulting a 1 at its output.



Static-0 Hazard

**To Cover Static-0 Hazard:** Refer to the K-map shown in the following Fig; POS is given by Y = (B +C) (A+C'). The corresponding circuit is also shown in the following Fig. For this circuit, if input B = 0, A = 0 and C makes a transition 0 → 1; there will be static-0 hazard occurring at output.



Static-0 Hazard & Its Cover

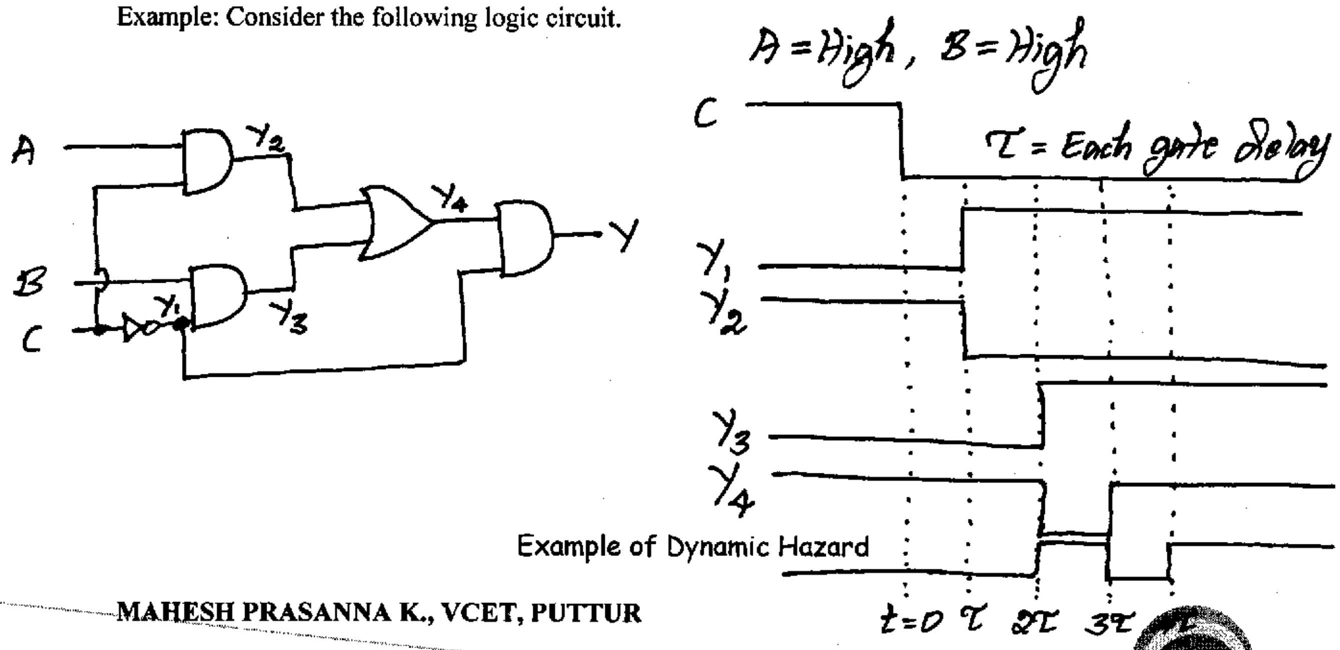
Consider another grouping for the same K-map (as shown in the above Fig). This includes one additional OR term, and now, output Y = (B + C) (A + C') (A + B). The corresponding circuit diagram is also given. This circuit requires more hardware, but it is hazard free. The additional term A + B ensures Y = 0 for B = 0, A = 0 and C makes a transition 0 → 1, does not affect output.

**NOTE:** A NOR gate with A and A' connected at its input for certain input combination will give static-0 hazard when A makes a transition 1 → 0 and requires hazard cover.

**Dynamic Hazard:**

Dynamic hazard occurs when circuit output makes multiple transitions before it settles to a final value, while the logic equation asks for only one transition. For example, an output transition designed as 1 → 0, may give 1 → 0 → 1 → 0 when such hazard occurs and a 0 → 1, can behave like 0 → 1 → 0 → 1.

Example: Consider the following logic circuit.



Example of Dynamic Hazard

This logic circuit can be written in the equation form as Y = (AC + BC')C. This shows dynamic hazard; for AB = 11, and C makes a transition 1 → 0 (as shown in the waveform). The hazard can be prevented by using an additional two input AND gate fed by input A and B and replacing the two input OR gate by a three input OR gate.

## HDL IMPLEMENTATION MODELS:

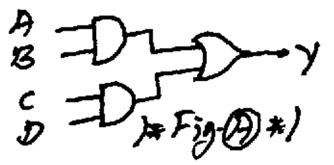
Structural modeling, though convenient, consumes more space in describing a circuit, and is unsuitable for large, complex design.

### Dataflow Modeling:

Verilog provides a keyword ***assign*** and a set of operators (given in the following Table) to describe a circuit through its function. All ***assign*** statements are concurrent and continuous.

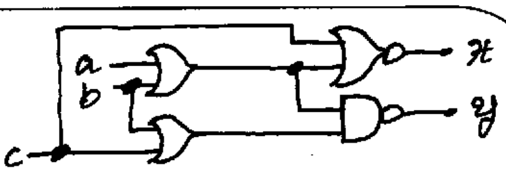
| **Relational Operation** | **Symbol** | **Bit-wise Operation** | **Symbol** |
|---|---|---|---|
| Less than | < | Bit-wise NOT | ~ |
| Less than or equal to | <= | Bit-wise OR | \| |
| Greater than | > | Bit-wise AND | & |
| Greater than or equal to | >= | Bit-wise Ex-OR | ^ |
| Not equal to | != | **Arithmetic Operation** | **Symbol** |
| **Logical Operation** | **Symbol** | Binary addition | + |
| Logical NOT | ! | Binary subtraction | – |
| Logical OR | \|\| | Binary multiplication | * |
| Logical AND | && | Binary division | / |

Data flow model resembles a logic equation and thus gives a more crisp representation.



/* Fig (A) */

```
module FigA (A,B,C,D,Y);
input A,B,C,D;
output Y;
   assign Y = (A&B) | (C&D);
endmodule
```



```
module testckt (a,b,c,x,y);
input a,b,c;
output x,y;
   assign x = ~((a|b) | c); // NOR by NOT-OR
   assign y = ~((a|b)&(b|c)); // NAND by NOT-OR
endmodule
```

**Behavioral Modeling:**

In a behavioral model, statements are executed sequentially following algorithmic description. It always uses ***always*** keyword followed by a sensitivity list. The procedural statements following ***always*** are executed only if any variable within sensitivity list changes its value. Procedure assignment or output variables within ***always*** must be register type, defined by ***reg***.

```
module FigA (A,B,C,D,Y);
input A,B,C,D; output Y;
reg Y;
always @ (A or B or C or D) // Sensitivity list
  if (A==1) && (B==1) // If A=1 & B=1; Y=1.
      Y=1;
  else if (C==1) && (D==1) // If C=1 & D=1; Y=1.
      Y=1;
  else Y=D; // For all other combination of A,B,C,D.
endmodule
```

***Problem:*** *Realize* $Y = \sum m\,(0, 1, 2, 4, 5, 6, 8, 9, 10, 12, 13)$ *using dataflow model.*

***Solution:***

| Y | $\bar{C}\bar{D}$ | $\bar{C}D$ | $CD$ | $C\bar{D}$ |
|---|---|---|---|---|
| $\bar{A}\bar{B}$ | 1 | 1 | 0 | 1 |
| $\bar{A}B$ | 1 | 1 | 0 | 1 |
| $AB$ | 1 | 1 | 0 | 0 |
| $A\bar{B}$ | 1 | 1 | 0 | 1 |

$$Y = \bar{C} + \bar{A}\bar{D} + \bar{B}\bar{D}.$$

**NOTE:**

Structural ↔ Gates / Circuits

Dataflow ↔ ***assign*** / Expression

Behavioral ↔ ***always*** / Truth table

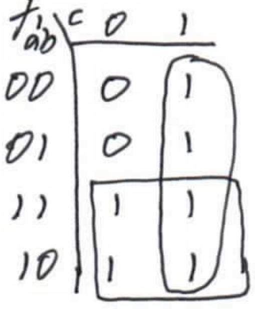
```
module K_Map (A,B,C,D,Y);
input A,B,C,D; output Y;
  assign Y = ~C | (~A & ~D) | (~B & ~D);
endmodule
```

**Problem:** *Find the minimal sums for –*

a) *f1 (a, b, c) = ∑ (1, 3, 4, 5, 6, 7)*

b) *f2 (a, b, c) = Π (2, 4, 7)*

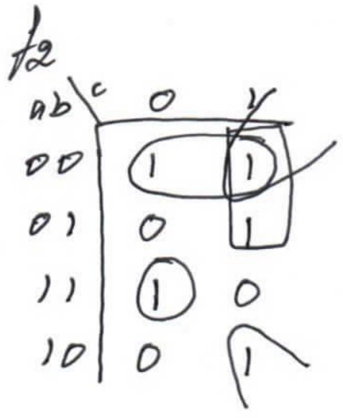
**Solution:**



| $f_1$ ab \ c | 0 | 1 |
|---|---|---|
| 00 | 0 | 1 |
| 01 | 0 | 1 |
| 11 | 1 | 1 |
| 10 | 1 | 1 |

$\therefore f_1 = c + a.$

$\therefore f_2 = \bar{a}\bar{b} + \bar{a}c + \bar{b}c + ab\bar{c}.$



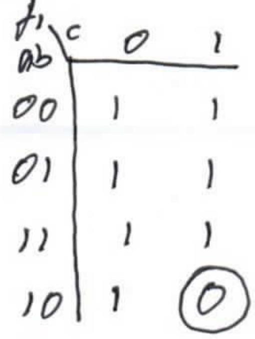
| $f_2$ ab \ c | 0 | 1 |
|---|---|---|
| 00 | 1 | 1 |
| 01 | 0 | 1 |
| 11 | 1 | 0 |
| 10 | 0 | 1 |

**Problem:** *Find the minimal products for –*

a) *f1 (a, b, c) = ∑ (0, 1, 2,3, 4, 6, 7)*

b) *f2 (a, b, c) = Π (1, 4, 5)*

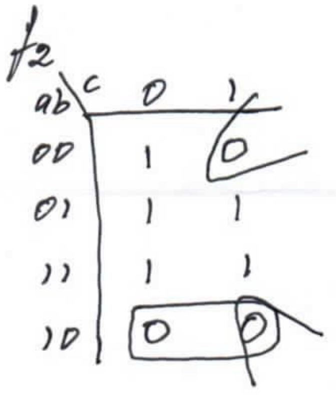
**Solution:**



| $f_1$ ab \ c | 0 | 1 |
|---|---|---|
| 00 | 1 | 1 |
| 01 | 1 | 1 |
| 11 | 1 | 1 |
| 10 | 1 | 0 |

$\therefore f_1 = \bar{a} + b + \bar{c}.$

$\therefore f_2 = (\bar{a} + b)(a + \bar{c}).$



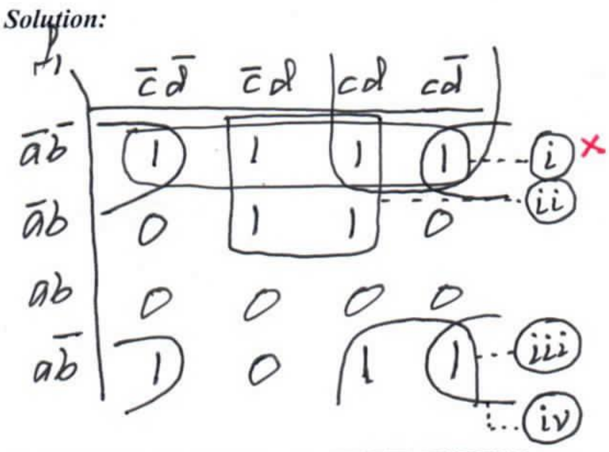
| $f_2$ ab \ c | 0 | 1 |
|---|---|---|
| 00 | 1 | 0 |
| 01 | 1 | 1 |
| 11 | 1 | 1 |
| 10 | 0 | 0 |

**Problem:** *Solve for the simplified Boolean expression using K-map:*

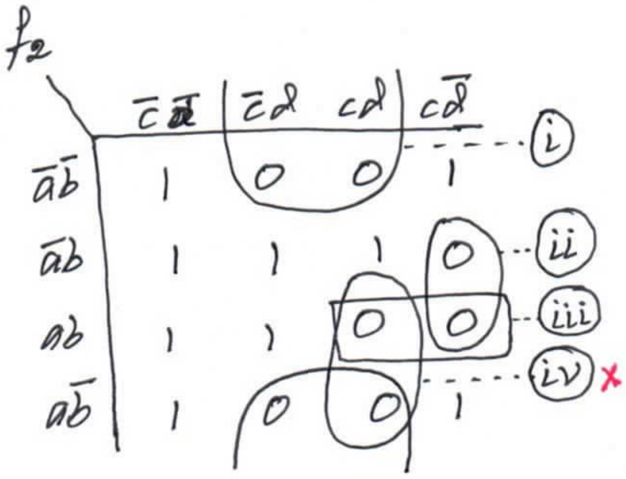
a) $f1\,(a, b, c, d) = \bar{a}\bar{c}d + \bar{a}cd + \bar{b}\bar{c}\bar{d} + a\bar{b}c + \bar{a}\bar{b}c\bar{d}$

b) $f2\,(a, b, c, d) = (a + b + \bar{d})(\bar{a} + b + \bar{d})(a + \bar{b} + \bar{c} + d)(\bar{a} + \bar{b} + \bar{c} + d)(\bar{a} + \bar{b} + \bar{c} + d)$

**Solution:**



| $f_1$ | $\bar{c}\bar{d}$ | $\bar{c}d$ | $cd$ | $c\bar{d}$ |
|---|---|---|---|---|
| $\bar{a}\bar{b}$ | 1 | 1 | 1 | 1 |
| $\bar{a}b$ | 0 | 1 | 1 | 0 |
| $ab$ | 0 | 0 | 0 | 0 |
| $a\bar{b}$ | 1 | 0 | 1 | 1 |



| $f_2$ | $\bar{c}\bar{d}$ | $\bar{c}d$ | $cd$ | $c\bar{d}$ |
|---|---|---|---|---|
| $\bar{a}\bar{b}$ | 1 | 0 | 0 | 1 |
| $\bar{a}b$ | 1 | 1 | 1 | 0 |
| $ab$ | 1 | 1 | 0 | 0 |
| $a\bar{b}$ | 1 | 0 | 0 | 1 |

$\therefore f_1 = \bar{a}d + \bar{b}c + \bar{b}\bar{d}.$

$\therefore f_2 = (b + \bar{d})(\bar{b} + \bar{c} + d)(\bar{a} + \bar{b} + \bar{c}).$

***Problem:*** *Design a 3-input, 1-output, minimal two-level gate combinational circuit; which has an output equal to 1 when majority of its inputs are at logic 1, and has output 0 when majority of inputs are at logic 0.*

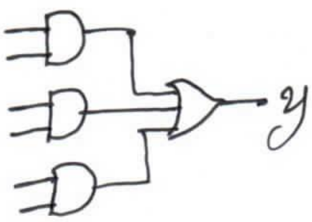
***Solution:***

Construct the truth table:

| a | b | c | y |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

| $y$ | $\bar{c}$ | $c$ |
|---|---|---|
| $\bar{a}\bar{b}$ | 0 | 0 |
| $\bar{a}b$ | 0 | 1 |
| $ab$ | 1 | 1 |
| $a\bar{b}$ | 0 | 1 |

Grouping for logic 1s; $y = ab + bc + ac$



If no. of gate inputs is taken as cost; the cost of implementation is 9.

Grouping for 0s; $y = (a+b)(b+c)(a+c)$.

Here also, the cost is 9.

***Problem:*** *Design a minimal sum and minimal product combinational gate circuit to generate the odd parity bit for an 8421 BCD code.*

***Solution:*** Construct the truth table, as described in the question.

| A | B | C | D | Y |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | X |
| 1 | 0 | 1 | 1 | X |
| 1 | 1 | 0 | 0 | X |
| 1 | 1 | 0 | 1 | X |
| 1 | 1 | 1 | 0 | X |
| 1 | 1 | 1 | 1 | X |

$$\therefore Y = \sum m(0, 3, 5, 6, 9) + \sum d(10, 11, 12, 13, 14, 15)$$

| $Y$ | $\bar{C}\bar{D}$ | $\bar{C}D$ | $CD$ | $C\bar{D}$ |
|---|---|---|---|---|
| $\bar{A}\bar{B}$ | 1 | 0 | 1 | 0 |
| $\bar{A}B$ | 0 | 1 | 0 | 1 |
| $AB$ | X | X | X | X |
| $A\bar{B}$ | 0 | 1 | X | X |

$$Y_{SOP} = AD + B\bar{C}D + BC\bar{D} + \bar{B}CD + \bar{A}\bar{B}\bar{C}\bar{D}.$$

Cost = 20.

| $Y$ | $\bar{C}\bar{D}$ | $\bar{C}D$ | $CD$ | $C\bar{D}$ |
|---|---|---|---|---|
| $\bar{A}\bar{B}$ | 1 | 0 | 1 | 0 |
| $\bar{A}B$ | 0 | 1 | 0 | 1 |
| $AB$ | X | X | X | X |
| $A\bar{B}$ | 0 | 1 | X | X |

$$Y_{POS} = (\bar{A}+D)(B+\bar{C}+D)(\bar{B}+\bar{C}+\bar{D})(A+\bar{B}+C+D)(A+B+C+\bar{D}).$$

Cost = 21.

***blem:** Using Quine-McClusky method, simplify; f (a, b, c, d) = ∑ (3, 4, 5, 7, 10, 12, 14, 15) + ∑d (2).*

***Solution:*** Determination of prime implicants, considering don't care terms as 1:

| Sate 1 abcd | | Stage 2 abcd | |
|---|---|---|---|
| 0010 | (2) ✓ | 001– | (2,3) |
| 0100 | (4) ✓ | –010 | (2,10) |
| 0011 | (3) ✓ | 010– | (4,5) |
| 0101 | (5) ✓ | –100 | (4,12) |
| 1010 | (10) ✓ | 0–11 | (3,7) |
| 1100 | (12) ✓ | 01–1 | (5,7) |
| 0111 | (7) ✓ | 1–10 | (10,14) |
| 1110 | (14) ✓ | 11–0 | (12,14) |
| 1111 | (15) ✓ | –111 | (7,15) |
| | | 111– | (14,15) |

There is no stage 3 here,, as no two members of stage 2 has only one digit changing among them.

Selection of essential prime implicants: (Please note, don't care is not considered here)

| - | 3 | 4 | 5 | 7 | 10 | 12 | 14 | 15 | Row |
|---|---|---|---|---|---|---|---|---|---|
| $\bar{A}\bar{B}C$ (2,3) | ✓ | | | | | | | | Q |
| $\bar{B}C\bar{D}$ (2,10) | | | | | ✓ | | | | R |
| $\bar{A}B\bar{C}$ (4,5) | | ✓ | ✓ | | | | | | S |
| $B\bar{C}\bar{D}$ (4,12) | | ✓ | | | | ✓ | | | T |
| $\bar{A}CD$ (3,7) | ✓ | | | ✓ | | | | | U |
| $\bar{A}BD$ (5,7) | | | ✓ | ✓ | | | | | V |
| $AC\bar{D}$ (10,14) | | | | | ✓ | | ✓ | | W |
| $AB\bar{D}$ (12,14) | | | | | | ✓ | ✓ | | X |
| $BCD$ (7,15) | | | | ✓ | | | | ✓ | Y |
| $ABC$ (14,15) | | | | | | | ✓ | ✓ | Z |

- Row U dominates row Q
- Row W dominates row R
    - Columns with min-terms 3 & 10 and 7 & 14 have one tick (mark) each. The corresponding rows U & W are essential rows. Hence, minimal sum = U + W + . . . .

- Delete row U, row W, columns 3, 10, 7, & 14. Now, the table reduces to:

| - | 4 | 5 | 12 | 15 | Row |
|---|---|---|---|---|---|
| $\bar{A}B\bar{C}$ (4,5) | ✓ | ✓ | | | S |
| $B\bar{C}\bar{D}$ (4,12) | ✓ | | ✓ | | T |
| $\bar{A}BD$ (5,7) | | ✓ | | | V |
| $AB\bar{D}$ (12,14) | | | ✓ | | X |
| $BCD$ (7,15) | | | | ✓ | Y |
| $ABC$ (14,15) | | | | ✓ | Z |

- Row S dominates row V
- Row T dominates row X
- Rows Y and Z are equal. Hence, only one row out of Y and Z need to be considered for minimal sum
- Rows S, T, and Y are considered for minimal terms. Delete row V, row X, and row Z. Therefore, minimal sum = U + W + S + T + Y.

Hence, f = $\bar{A}CD + AC\bar{D} + \bar{A}B\bar{C} + B\bar{C}\bar{D} + BCD$.

***Problem:*** *Simplify f (a, b, c, d) = ∑ (2, 3, 4, 5, 13, 15) + ∑d (8, 9, 10, 11) using EVM techniques, taking –*

*a) The variable in the last significant position as MEV*

*b) The variables c and d as MEVs.*

***Solution:***

| a | b | c | d | f | MEV: d | MEVs: c & d |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | $c$ |
| 0 | 0 | 0 | 1 | 0 | | |
| 0 | 0 | 1 | 0 | 1 | 1 | |
| 0 | 0 | 1 | 1 | 1 | | |
| 0 | 1 | 0 | 0 | 1 | 1 | $\bar{c}$ |
| 0 | 1 | 0 | 1 | 1 | | |
| 0 | 1 | 1 | 0 | 0 | 0 | |
| 0 | 1 | 1 | 1 | 0 | | |
| 1 | 0 | 0 | 0 | × | × | × |
| 1 | 0 | 0 | 1 | × | | |
| 1 | 0 | 1 | 0 | × | × | |
| 1 | 0 | 1 | 1 | × | | |
| 1 | 1 | 0 | 0 | 0 | $d$ | $d$ |
| 1 | 1 | 0 | 1 | 1 | | |

| $f_a$ | $\bar{c}$ | $c$ |
|---|---|---|
| $\bar{a}\bar{b}$ | 0 | 1 |
| $\bar{a}b$ | 1 | 0 |
| $ab$ | $d$ | $d$ |
| $a\bar{b}$ | × | × |

$\therefore f_a = ad + \bar{b}c + \bar{a}b\bar{c}$.

| $f_b$ | $\bar{b}$ | $b$ |
|---|---|---|
| $\bar{a}$ | $c$ | $\bar{c}$ |
| $a$ | × | $d$ |

$\therefore f_b = bc + \bar{b}c + ab\bar{c}$.

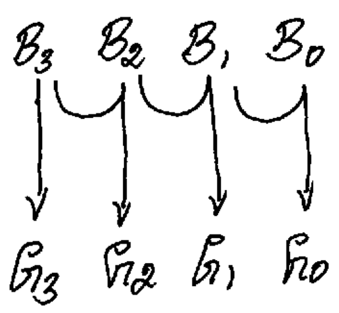
|  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|
| [illegible] | 1 | 1 | 0 | 0 | d |  |
| 1 | 1 | 1 | 1 | 1 |  |  |

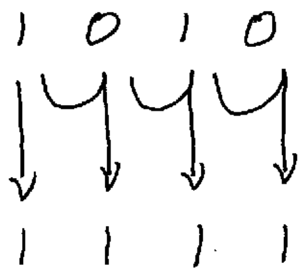
***Try these:*** *Design a minimal sum combinational circuit to –*

a) *Find the 9s complement of BCD numbers*
b) *Convert BCD to Excess-3*
c) *Multiply two 2-bit numbers*
d) *Output a 1 when an illegal BCD code occurs*
e) *Output the 2s complement of a 4-bit binary number.*

***Problem:*** *Design Binary-to-Gray Code Converter.*

***Solution:***





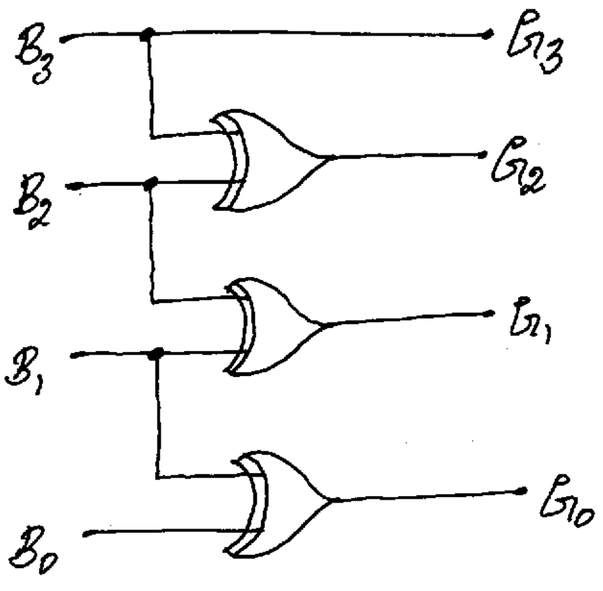
$G_3 = B_3$

$G_2 = B_3 \oplus B_2$

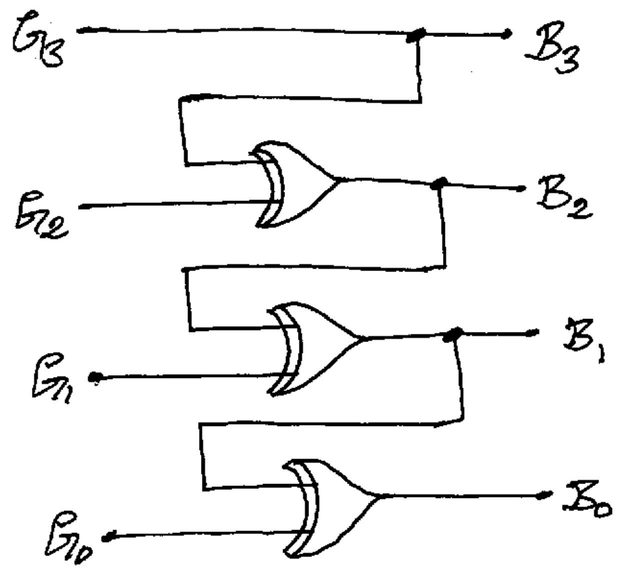
$G_1 = B_2 \oplus B_1$

$G_0 = B_1 \oplus B_0$

| Binary Code | | | | Gray Code | | | |
|---|---|---|---|---|---|---|---|
| B3 | B2 | B1 | B0 | G3 | G2 | G1 | G0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |

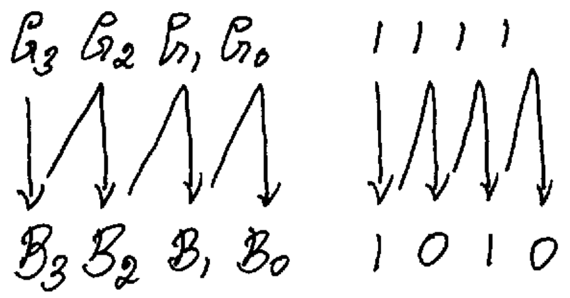


| | | | | | | | |
|---|---|---|---|---|---|---|---|
| [illegible] | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |



***Problem:*** *Design Gray-to-Binary Code Converter.*

***Solution:***



$B_3 = G_3$ $\qquad$ $B_1 = B_2 \oplus G_1$

$B_2 = B_3 \oplus G_2$ $\qquad$ $B_0 = B_1 \oplus G_0$

| Gray Code | | | | Binary Code | | | |
|---|---|---|---|---|---|---|---|
| G3 | G2 | G1 | G0 | B3 | B2 | B1 | B0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

| Gray Code | | | | Binary Code | | | |
|---|---|---|---|---|---|---|---|
| G3 | G2 | G1 | G0 | B3 | B2 | B1 | B0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 |

**Problem:** *Simplify, using K-map method Quine Mc-Clusky method: F = ∑m (0, 1, 2, 8, 10, 11, 14, 15).*

**Solution:**

*K-map method:*

| F | $\bar{C}\bar{D}$ | $\bar{C}D$ | $CD$ | $C\bar{D}$ |
|---|---|---|---|---|
| $\bar{A}\bar{B}$ | 1 | 1 | 0 | 1 |
| $\bar{A}B$ | 0 | 0 | 0 | 0 |
| $AB$ | 0 | 0 | 1 | 1 |
| $A\bar{B}$ | 1 | 0 | 1 | 1 |

$$\therefore F = AC + \bar{B}\bar{D} + \bar{A}\bar{B}\bar{C}.$$

*Quine Mc-Clusky method:*

| A | B | C | D | F |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

| Stage 1 | Stage 2 | Stage 3 |
|---|---|---|
| ABCD | ABCD | ABCD |
| 0000 (0)✓ | 000– (0,1) | –0–0 (0,2,8,10) |
| 0001 (1)✓ | 00–0 (0,2)✓ | –0–0 (0,8,2,10) |
| 0010 (2)✓ | –000 (0,8)✓ | |
| 1000 (8)✓ | –010 (2,10)✓ | |
| 1010 (10)✓ | 10–0 (8,10)✓ | 1–1– (10,11,14,15) |
| 1011 (11)✓ | 101– (10,11)✓ | 1–1– (10,14,11,15) |
| 1110 (14)✓ | 1–10 (10,14)✓ | |
| 1111 (15)✓ | 1–11 (11,15)✓ | |
| | 111– (14,15)✓ | |

| - | 0 | 1 | 2 | 8 | 10 | 11 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|
| $\bar{A}\bar{B}\bar{C}$ (0,1) | ✓ | ✓ | | | | | | |
| $\bar{B}\bar{D}$ (0,2,8,10) | ✓ | | ✓ | ✓ | ✓ | | | |
| $AC$ (10,11,14,15) | | | | | ✓ | ✓ | ✓ | ✓ |

Therefore, F = $AC + \bar{B}\bar{D} + \bar{A}\bar{B}\bar{C}$.

By: **MAHESH PRASANNA K.,**

**DEPT. OF CSE, VCET.**

*********

*********