3- Variable Boolean Expression

$F(A,B,C) = \Sigma m(0,2,4,5,6,7)$

$y = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + A\bar{B}\bar{C} + A\bar{B}C + AB\bar{C} + ABC$

K-Map

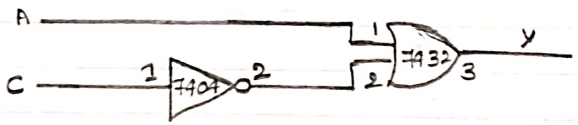
A \ BC	$\bar{B}\bar{C}$ (00)	$\bar{B}C$ (01)	BC (11)	$B\bar{C}$ (10)
$\bar{A}$ (0)	1 (0)	0 (1)	0 (3)	1 (2)
A (1)	1 (4)	1 (5)	1 (7)	1 (6)

$y = A + \bar{C}$

Truth Table

A	C	y
0	0	1
0	1	0
1	0	1
1	1	1

Circuit diagram



4- Variable Boolean Expression

$F(A,B,C,D) = \Sigma m(0,2,5,7,8,10,13,15)$

$y = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}C\bar{D} + \bar{A}B\bar{C}D + \bar{A}B\bar{C}D + A\bar{B}\bar{C}D + A\bar{B}\bar{C}\bar{D} + AB\bar{C}D + ABCD$

Date :	Exp. Title Realize 3-variable and 4-Variable Boolean expressions, Simplify using K-map and implementing Basic gates.	Page No.
Exp. No. 01 [A]		1

Aim: Realize 3-variable and 4-variable Boolean expressions, Simplify using K-map and implementing Basic Gates.

Apparatus Required

SL.NO.	Component	Specification
1	AND GATE	IC 7408
2	OR GATE	IC 7432
3	NOT GATE	IC 7404
4	IC TRAINER GATE	-
5	PATCH CORDS	-

Theory: Karnaugh maps are used to simplify real-world logic requirements so that they can be implemented using a minium number of logic gates. A sum of products expression (SOP) can always be implemented using AND gates feeding into an OR gate, and product-of sums expression (POS) leads to OR gates feeding and AND gate.

The pos expression gives a Complement of the function (if F is the function so its complement will be F). Karnaugh maps can also be used to simplify logic expressions in software design. Boolean conditions as used for example in Conditional statements, can get very complicated, which makes the codedifficult to read and to maintain. Once minimized, canonical sum-of-products and product-of-sums expressions can be implemented directly using AND and OR logic operators.

K-Map

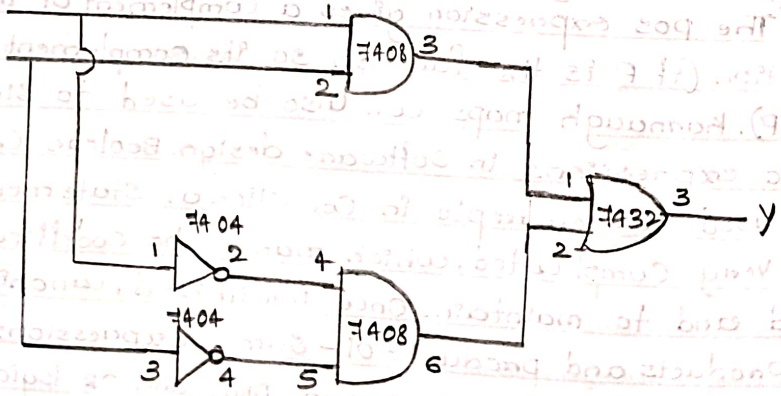
AB \ cd	$\bar{C}\bar{D}$ (00)	$\bar{C}D$ (01)	CD (11)	$C\bar{D}$ (10)
$\bar{A}\bar{B}$ (00)	1 (0)	0 (1)	0 (3)	1 (2)
$\bar{A}B$ (01)	(4)	1 (5)	1 (7)	(6)
AB (11)	0 (12)	1 (13)	1 (15)	0 (14)
$A\bar{B}$ (10)	1 (8)	0 (9)	0 (11)	1 (10)

$$Y = \bar{B}\bar{D} + BD$$

Truth table

Inputs						output
B	D	B'	D'	B'D'	BD	B'D'+BD
0	0	1	1	1	0	1
0	1	1	0	0	0	0
1	0	0	1	0	0	0
1	1	0	0	0	1	1

Circuit diagram



Applications of Karnaugh's Map

* They are used in design and implementation of Digital circuits.
* K-maps are useful for detecting and eliminating race condition.

Procedure

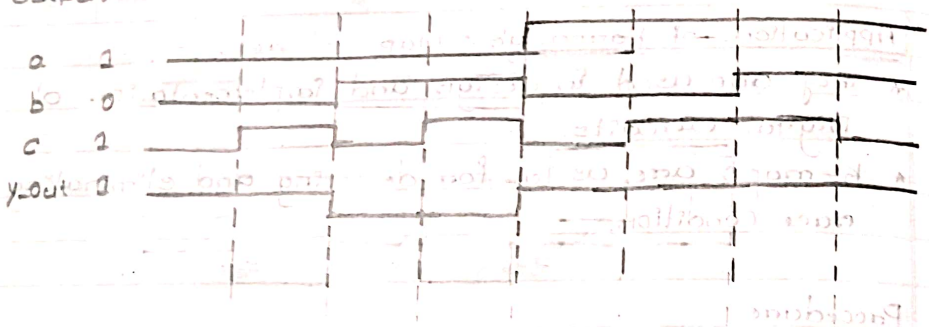
1. Make sure that the given components are working properly before connecting to the trainer circuit.
2. Place the given Ic's in the sockets provided on the trainer circuit.
3. Make the connections as per the circuit diagram. Now on the power supply.
4. For the different combinations of the inputs given in the truth table, and verify the outputs in trainer kit.
5. Switch off the power supply and remove the connections.

Result :

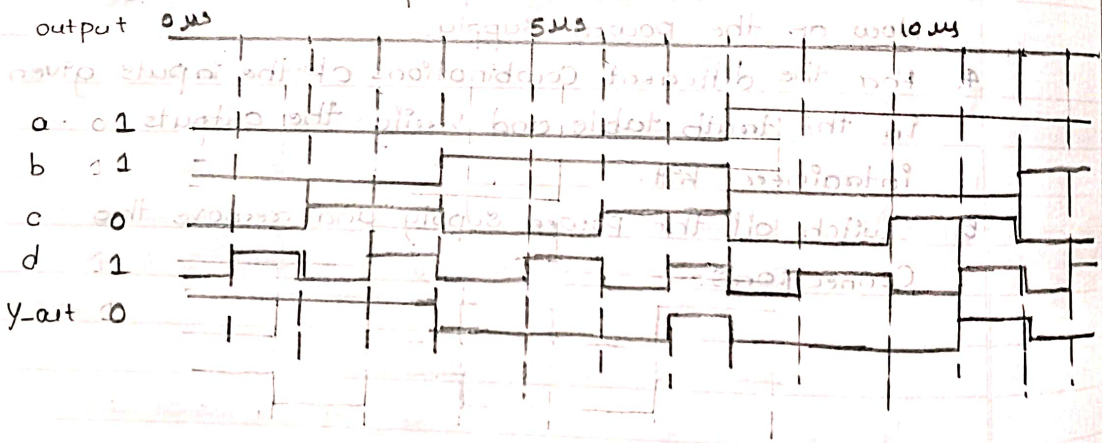
3 variable and 4-variable Boolean expressions are simplified and verified using Basic gates

3- Variable Boolean Expression

Output:



4 - Variable Boolean Expression



3- Variable Boolean Expression

$Y = A + \overline{C}$

```
entity three_variable_booleanExp
  isport(a,b,c : in STD_LOGIC; y_out:out STD_LOGIC);
end three_variable_booleanExp;
architecture Behavioral of three_variable_booleanExp is
begin
    y_out <=(a or (not(c)));
end Behavioral;
```

4- Variabe Boolean Expression

$Y = \overline{B}\,\overline{D} + BD$

```
entity Four_variableBool_Exp is
  Port (a,b,c,d : in STD_LOGIC; y_Out: out STD_LOGIC);
end Four_variableBool_Exp;
architecture Behavioral of Four_variableBool_Exp is begin
  y_out<=((((not(b)) and (not(d))) or (b and d));
    end Behavioral;
```

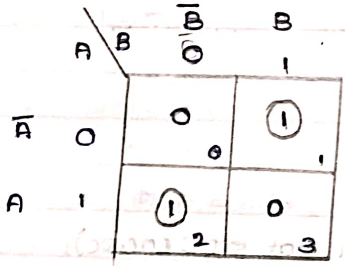
Half Adder

Truth Table

Cell NO.	Input		Output	
	A	B	Sum	Carry
0	0	0	0	0
1	0	1	1	0
2	1	0	1	0
3	1	1	0	1

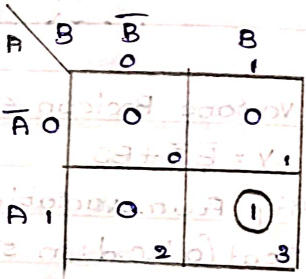
K-Map

Sum



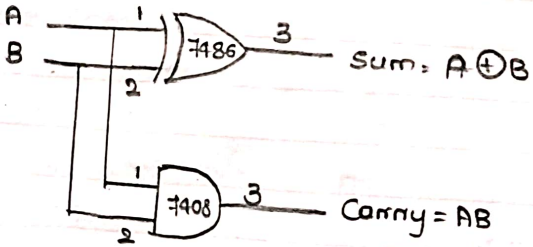
$Sum = \overline{A}B + A\overline{B}$

Carry



$Carry = AB$

Circuit diagram



Date :	Exp. Title Design and Implement Half Adder and Full Adder using Basic Gates	Page No.
Exp. No. 02 [A]		4

Aim: Design and Implement Half Adder and Full Adder using Basic Gates.

Apparatus Required

SL.NO.	Component	Specification
1	AND GATE	IC 7408
2	OR GATE	IC 7432
3	NOT GATE	IC 7404
4	IC TRAINER KIT	
5	PATCH CORDS	

Theory

Half Adder A half adder has two inputs, for the two bits to be added and two outputs one from the Sum 'S' and other from the carry 'C' into the higher adder position. Above circuit is called as a Carry signal from the addition of the less significant bits sum from the XOR Gate the Carry out from the AND gate.

Full Adder A full adder is a combinational circuit that froms the arithmetic sum of inputs it consists of three inputs and two outputs. A full adder is useful to add three bits at a time but a half adder cannot do so. In full adder sum output will be taken from XOR Gate, Carry output will be taken from OR Gate.

Applications

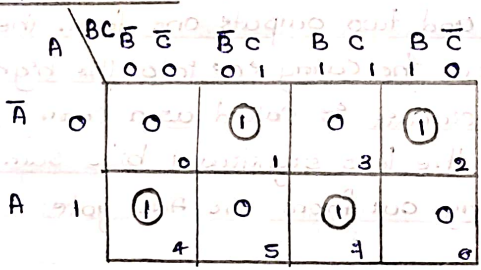
* They are used computers, Calculators and various digital measuring instruments.
* They are used in digital processing devices.

Full Adder

Truth Table

Cell No.	Inputs			Outputs	
	A	B	C	Sum	Carry
0	0	0	0	0	0
1	0	0	1	1	0
2	0	1	0	1	0
3	0	1	1	0	1
4	1	0	0	1	0
5	1	0	1	0	1
6	1	1	0	0	1
7	1	1	1	1	1

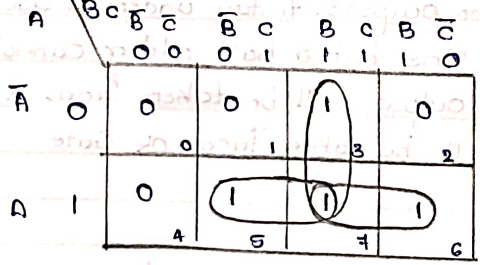
K-Map for Sum:



$Sum = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC$

$Sum: \bar{A}\bar{B}C + B\bar{C} + A(BC + \bar{B}\bar{C})$

K-Map for Carry:



$Carry = BC + AC + AB$

Procedure

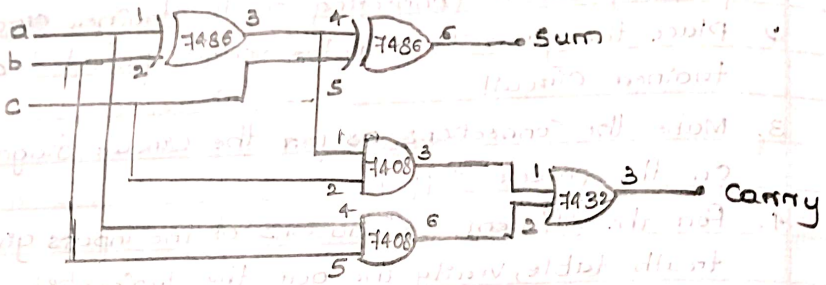
1. Make sure that the given components are working properly before connecting to the trainer circuit.
2. Place the given IC's in the sockets provided on the trainer circuit.
3. Make the connections as per the circuit diagram. Now on the power supply.
4. For the different combinations of the inputs given in the truth table, verify the out the trainerkit

Switch off the power supply and remove the connections.

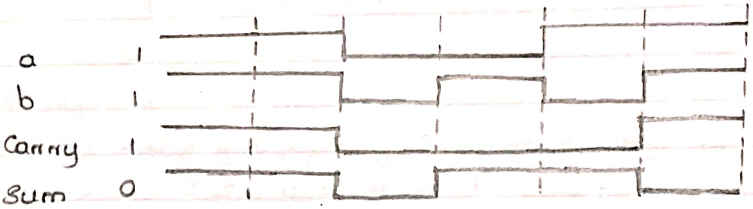
Result

Half adder and full adder truth tables are verified.

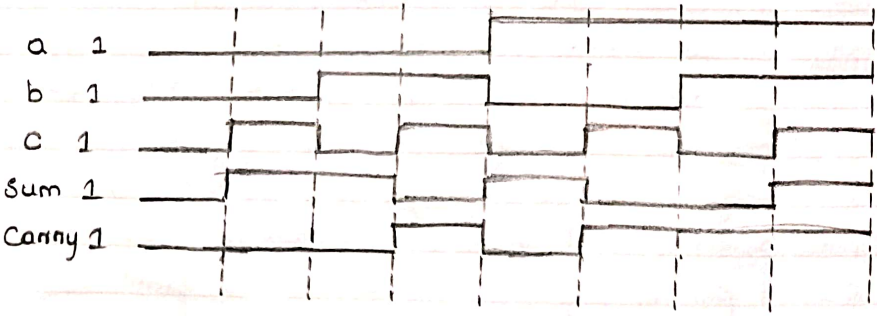
Circuit diagram For Full adder



Half Adder



Full Adder



Date :

Exp. No 02 [B]

Exp. Title Simulate and Verify the working of above expressions using VHDL.

Half Adder

```
entity half_Adder is
    Port (a, b: in STD_LOGIC; Carry, Sum: out STD_LOGIC);
end half_Adder;
architecture Behavioural of half_Adder is begin
    Sum <= (not (a) and b) or (a and (not(b)));
    Carry <= a and b;
end Behavioural;
```

Full Adder

```
entity FullAdder is
    Port (a, b, c : in STD_LOGIC ; Sum, Carry: out STD_LOGIC);
end Full Adder;
architecture Behavioural of Full Adder is begin
    Sum <= (not(c) and (a xor b)) or (c and (a xnor b));
    Carry <= (a and b) or (a and c) or (b and c);
end Behavioral;
```

Truth Table

	A	B	C	D	y	Implementation Table		
0	0	0	0	0	1	1	D_0	4
	0	0	0	1	1			
1	0	0	1	0	0	0	D_1	3
	0	0	1	1	0			
2	0	1	0	0	1	$\bar{D}$	D_2	2
	0	1	0	1	0			
3	0	1	1	0	1	$\bar{D}$	D_3	1
	0	1	1	1	0			
4	1	0	0	0	0	D	D_4	15
	1	0	0	1	1			
5	1	0	1	0	1	$\bar{D}$	D_5	14
	1	0	1	1	0			
6	1	1	0	0	1	$\bar{D}$	D_6	13
	1	1	0	1	0			
7	1	1	1	0	1	$\bar{D}$	D_7	12
	1	1	1	1	0			

Date :

Exp. No. 03 [A]

Exp. Title A 4-Variable logic expression, Simplify it using Entered Variable Map and realize the simplified logic exp using 8:1 MUX.

Aim : Given a 4-Variable logic expression, Simplify it using Entered Variable Map and realize the Simplified logic expression using 8:1 multiplexer.

Apparatus Required

Sl.No	Component	Specification
1	8:1 MUX	IC 74151
2	NOT GATE	IC 7404
3	IC TRAINER KIT	-
4	PATCH CORDS	-

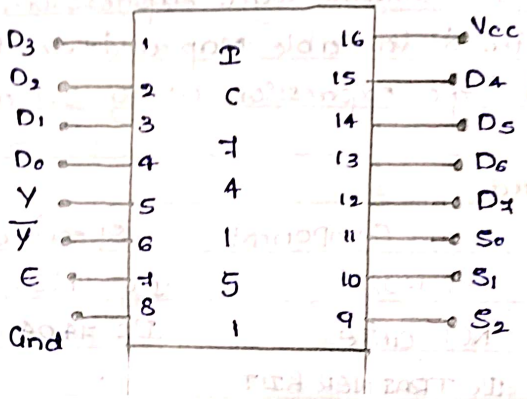
Theory

Entered Variable Map K-Map is the best manual technique to solve Boolean equations, but it becomes difficult to manage when number of variables exceed 5 or 6. So, a technique Called Variable Entrant Map (VEM) is used to increase the effective size of K-map. It allows a smaller map to handle large number of variables. This is done by writing output in terms of input.

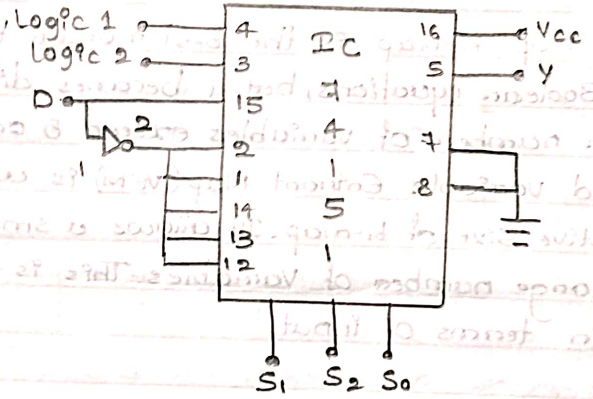
Multiplexers It is a Combinational Circuit which have many data inputs and single output depending on Control or select inputs. For N inputs lines, log n (base 2) Selection lines, or we Can say that for 2n input lines, n Selection lines are required. Multiplexers are also known as "Data n selector, Parallel to Serial Convertor, many to one Circuit, universal logic circuit". Multiplexers are mainly used to increase amount of the data that Can be sent over the network within certain amount of time and bandwidth.

4-variable function using IC 74151

8:1 Multiplexer using IC 74151



$f(a,b,c,d) = m(0,1,4,6,9,10,12,14)$



Date :	Exp. Title	Page No.
Exp. No.		08

Application

Multiplexers

* Communication system
* Telephone Network
* Computer Memory

Entered Variable Map

* It is Commonly used to solve problems involving multiplexers.

Procedure

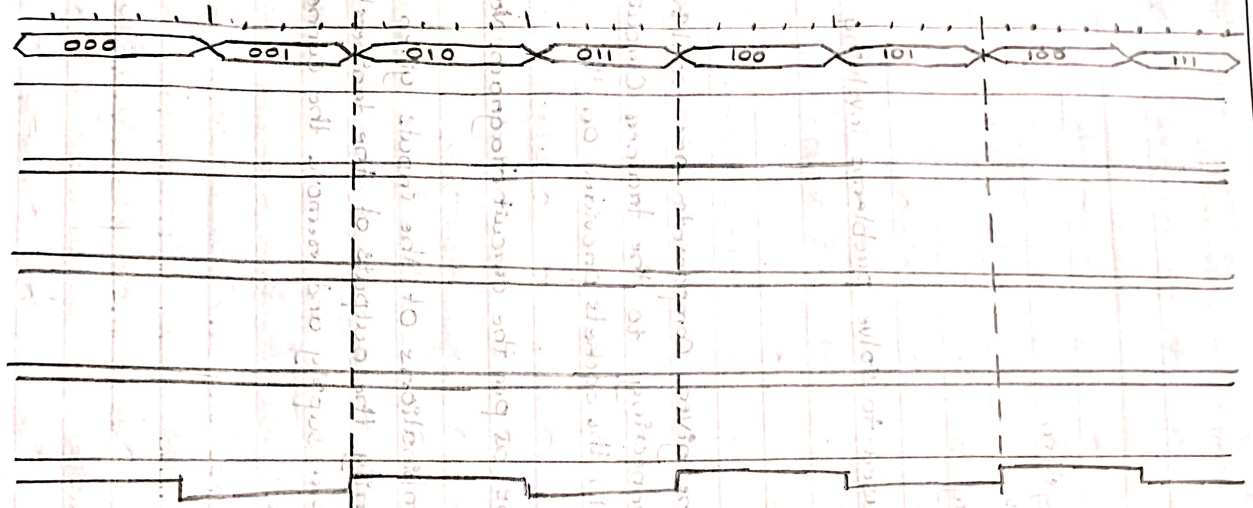
1. Make sure that the given components are working properly before connecting to the trainer Circuit.
2. Place the given IC's in the sockets provided on the trainer circuit.
3. Make the connections as per the circuit diagram. Now on the power supply.
4. For the different combinations of the inputs given in the truth table, Verify the outputs of the trainer kit. Switch off the power supply and remove the connections

Result

Given function is verified using truth table.

Output

d-0	1
d-1	0
d-2	1
d-3	0
d-4	1
d-5	0
d-6	1
d-7	0
mux_out	0



Date :

Exp. No. 03 [B]

Exp. Title Simulate and Verify the working of 8:1 multiplexer using VHDL.

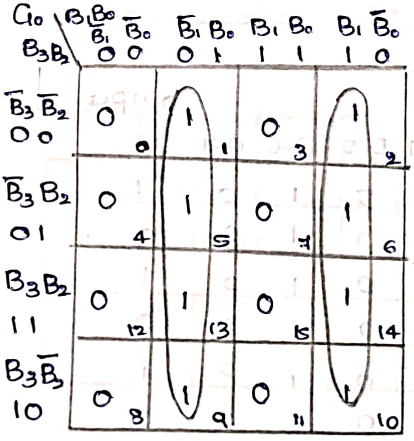

```
entity Multiplexer is
  Port(Sel: in STD_LOGIC_VECTOR (2 downto 0));
  d_0, d_1, d_2, d_3, d_4, d_5, d_6, d_7 : in STD_LOGIC;
  mux_out: out STD_LOGIC);
end Multiplexer;
architecture Behavioral of Multiplexer is begin
  Process(sel, d_0, d_1, d_2, d_3, d_4, d_5, d_6, d_7)
  begin
    Case sel is
      when "000" => mux_out <= d_0;
      when "001" => mux_out <= d_1;
      when "010" => mux_out <= d_2;
      when "011" => mux_out <= d_3;
      when "100" => mux_out <= d_4;
      when "101" => mux_out <= d_5;
      when "110" => mux_out <= d_6;
      when "111" => mux_out <= d_7;
      when others => null;
    end case;
  end process;
end Behavioral;
```

Select lines			Inputs								output
A	B	C	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	
0	0	0	1	0	1	0	1	0	1	0	1
0	0	1	1	0	1	0	1	0	1	0	0
0	1	0	1	0	1	0	1	0	1	0	1
0	1	1	1	0	1	0	1	0	1	0	0
1	0	0	1	0	1	0	1	0	1	0	1
1	0	1	1	0	1	0	1	0	1	0	0
1	1	0	1	0	1	0	1	0	1	0	1
1	1	1	1	0	1	0	1	0	1	0	0

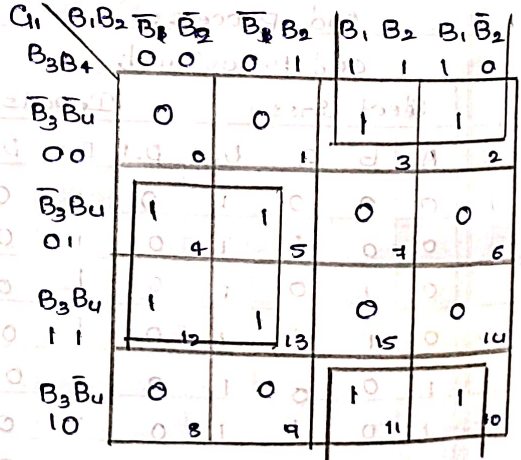
Truth Table

	Binary Code				Grey Code			
Cell No.	B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0
5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	1	1	1	1
11	1	0	1	1	1	1	1	0
12	1	1	0	0	1	0	1	0
13	1	1	0	1	1	0	1	1
14	1	1	1	0	1	0	0	1
15	1	1	1	1	1	0	0	0

K-Map



$G_0 = B_0\bar{B}_1 + \bar{B}_0B_1$



$G_1 = B_2\bar{B}_1 + \bar{B}_2B_1$

Date :	Exp. Title Design and implement the Binary to grey Code Converter using Basic gates.	Page No.
Exp. No. 04 [A]		10

Aim: Design and implement the Binary to grey code Converter using Basic gates.

Apparatus Required

SL. NO.	Component	Specification
1	AND GATE	IC 7408
2	OR GATE	IC 7432
3	NOT GATE	IC 7404
4	IC TRAINER KIT	
5	PATCH CORDS	

Theory : Binary Number is the default way to storre numbers but in many applications, binary numbers are difficult touse and a variety of binary numbers is needed. This is wherre Gray codes are verry useful.

Gray code has a property that two successive numbers differ in only one bit because of this property gray Code does the cycling through Varriocus states with minimal effort.

Applications

* Analog to digital Converterrs.
* Used for errror Correction in digital Communications.
* Used in transmission of digital signals.

G_2

B_3B_2 \ B_1B_0	$\bar{B}_1\bar{B}_0$ (00)	$\bar{B}_1B_0$ (01)	B_1B_0 (11)	$B_1\bar{B}_0$ (10)
$\bar{B}_3\bar{B}_2$ (00)	0	1	3	2
$\bar{B}_3B_2$ (01)	1 (4)	1 (5)	1 (7)	1 (6)
B_3B_2 (11)	12	13	15	14
$B_3\bar{B}_2$ (10)	1 (8)	1 (9)	1 (11)	1 (10)

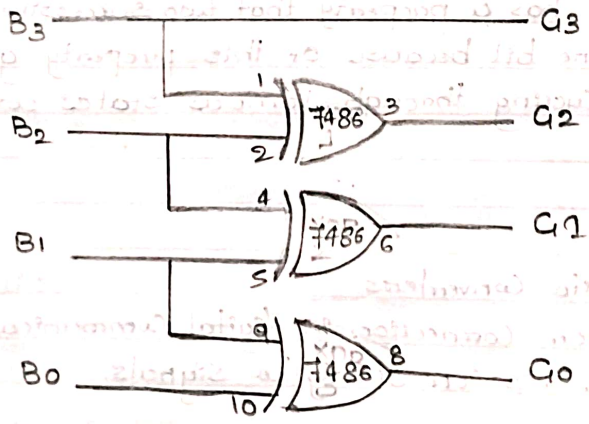
G_3

B_3B_2 \ B_1B_0	$\bar{B}_1\bar{B}_0$ (00)	$\bar{B}_1B_0$ (01)	B_1B_0 (11)	$B_1\bar{B}_0$ (10)
$\bar{B}_3\bar{B}_2$ (00)	0	1	3	2
$\bar{B}_3B_2$ (01)	4	5	7	6
B_3B_2 (11)	1 (12)	1 (13)	1 (15)	1 (14)
$B_3\bar{B}_2$ (10)	1 (8)	1 (9)	1 (11)	1 (10)

$$G_2 = B_2\bar{B}_3 + \bar{B}_2B_3$$

$$G_3 = B_3$$

Circuit diagram



Date :	Exp. Title	Page No.
Exp. No.		11

Procedure

1. Make sure that the given components are working properly before connecting to the trainer circuit.
2. Place the given IC's in the sockets provided on the trainer circuit.
3. Make the connections as per circuit diagram. Now on the power supply.
4. For the different combinations of the inputs given in the truth table, verify the outputs of trainer kit. Switch off the power supply and remove the connections.

Result

Binary to Grey code truth table is verified.

0 ns
5 ns
10 ns
15 ns
a[3:0]
b[3:0]

Date :	Exp. Title Simplify and verify the working of Binary to Grey Code Converter using VHDL.	Page No.
Exp. No. 04 [B]		12

```
entity BinaryToGrey is
        Port (b: in std_logic_vector (3 downto 0);
        g: out std_logic_vector (3 downto 0));
end BinaryToGrey;
architecture Behavioral of BinaryToGrey is
begin
        g(3)<=b(3);
        g(2)<=b(3) XOR b(2);
        g(1)<=b(2) XOR b(1);
        g(0)<=b(1) XOR b(0);
end Behavioral;
```

3-Bit Parity Generator

Truth Table

Cell NO.	A	B	C	Pae
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	0
6	1	1	0	0
7	1	1	1	1

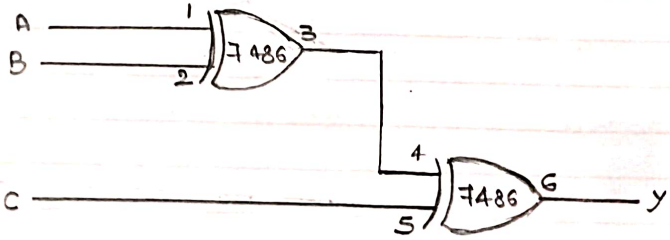
K-Map

A \ BC	$\bar{B}\bar{C}$ (00)	$\bar{B}C$ (01)	BC (11)	$B\bar{C}$ (10)
$\bar{A}$ (0)	0	(1) 1	0	(1) 1
A (1)	(1) 4	0 5	(1) 7	0 6

$$Pae = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC$$

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

Circuit diagram



Date :

Exp. No. 05

Exp. Title Design and Implement the truth table of 3-Bit Parity Generator and 4-bit Checker with an even parity bit.

Aim : Design and Implement the truth table of 3-Bit parity Generator and 4-bit checker with an Even Parity bit using basic gates.

Apparatus Required

Sl.No	Component	Specification
1	AND GATE	IC 7408
2	OR GATE	IC 7432
3	NOT GATE	IC 7404
4	IC TRAINER KIT	-
5	PATCH CORDS	-

Theory : Majority of modern communication is Digital in nature i.e., it is a combination of 1's and 0's. The digital data is transmitted either through wires (in case of wained communication) or wireless. Even in an advanced mode of communication, there will be errors while transmitting data (due to noise).

The simplest of errors is corruption of a bit i.e., a 1 may be transmitted as a 0 or vice-versa. To confirm whether the received data is the intended data or not, we should be able to detect errors at the receiver.

The parity generating technique is one of the most widely used error detection techniques for the data transmission. In digital systems, when binary data is transmitted and processed, data may be subjected to noise so that such noise can alter 0s (of data bits) to 1s and 1s to 0s.

Hence, a parity Bit is added to the word containing data in order to make number of 1s either even or odd.

4-Bit parity checker

Truth Table

Cell No.	A	B	C	D	Pch
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	0

The message containing the data bits along with parity bit is transmitted from transmitter to receiver.

At the receiving end, the number of 1s in the message is counted and if it doesn't match with the transmitted one, it means there is an error in the data. Thus, the parity Bit it is used to detect errors, during the transmission of binary data.

A parity Generator is a combinational logic circuit that generates the parity bit in the transmitter. On the other hand, a circuit that checks the parity in the receiver is called Parity checker. A combined circuit circuit or device of parity generators and parity checkers are commonly used in digital systems to detect the single bit errors in the transmitted data.

The sum of the data bits and parity bits can be even or odd. In even parity, the added parity bit will make the total number of 1s an even number, whereas in odd parity, the added parity bit will make the total number of 1s an odd number.

Applications

* Most widely used error detection technique for data transmission.
* Used in digital systems and many hardware applications
* Used peripheral Component Interconnect (PCI) to detect errors.

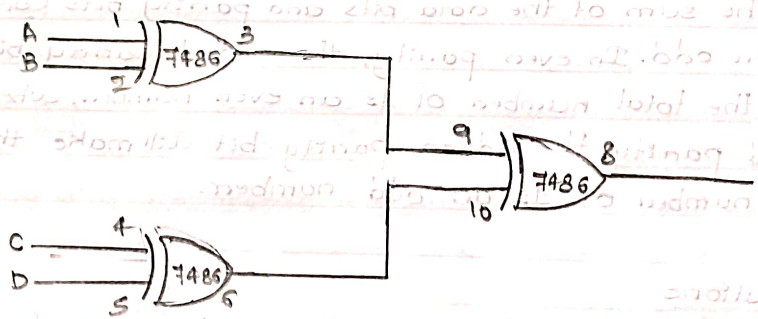
K-Map

AB \ CD	$\bar{C}\bar{D}$ (00)	$\bar{C}D$ (01)	CD (11)	$C\bar{D}$ (10)
$\bar{A}\bar{B}$ (00)	0 (0)	1 (1)	0 (3)	1 (2)
$\bar{A}B$ (01)	1 (4)	0 (5)	1 (7)	0 (6)
AB (11)	0 (12)	1 (13)	0 (14)	1 (15)
$A\bar{B}$ (10)	1 (8)	0 (9)	1 (11)	0 (10)

$P_{ch} = \bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}C\bar{D} + \bar{A}B\bar{C}\bar{D} + \bar{A}BCD + AB\bar{C}D + ABC\bar{D} + A\bar{B}\bar{C}\bar{D} + A\bar{B}CD$

$P_{ch} = \bar{A}\bar{B}(\bar{C}D + C\bar{D}) + \bar{A}B(\bar{C}D + \bar{C}\bar{D}) + AB(\bar{C}D + C\bar{D}) + A\bar{B}(\bar{C}\bar{D} + CD)$

Circuit diagram



Procedure

1. Make sure that the given components are working properly before connecting to the trainer circuit.
2. Place the given IC's in the sockets provided on the trainer circuit.
3. Make the connections as per circuit diagram. Now on the power supply.
4. For the different combinations of inputs given in the truth table, verify the outputs of the trainer kit.
5. Switch off the power supply and remove the connections.

Result

Parity generator and checker truth tables are verified.