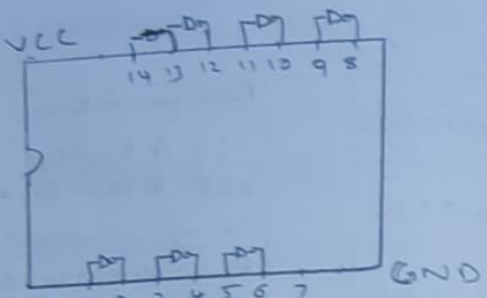
INDEX

EXP NO:	DATE	CO	EXPERIMENT NAME	MARKS	FACULTY SIGNATURE
1	02-07-25		Familiarization of Digital IC's & boolean Expression	25	16/7/25
2	11-07-25		SOP & POS simplification Using universal gates	25	
3	11-07-25		Familiarization of IC's & Boolean Expressions	25	
4	16/07/25		Familiarisation of simulation software & waveform	25	23/7/25
5	16-07-25		Realization of Boolean function using logic gates	25	
6	23-07-25		Familiarisation of ~~Boolean~~ verilog HDL modeling	25	28/7/25
8	28-07-25		Magnitude comparators using IC 7485	25	
7	28-07-25		Parity Generation & checker using IC 74180	25	
9	28-07-25		Multiplexer	25	6/8/25
10	06-08-25		Familiarisation of Flip flop	25	10/9/25
11	06-08-25		Asynchronous ~~Mod 3 to bit~~ updown counter	24	
12	10-09-25		Asynchronous MOD 10 Counter	25	17/9/25
13	10-09-25		Asynchronous 3 bit up counter	25	
14	17-09-25		Asynchronous 3 bit down counter	25	8/10/25
15	17-09-25		shift Registers	25	
16	08-10-25		Behavioural Model for flip flop	25	15/10/25
17	08-10-25		Behavioural model for Synchronous Counter	25	
18	15-10-25		MUX & DEMUX using Assignment	25	17/10/25

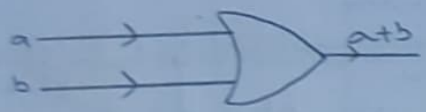
Completed

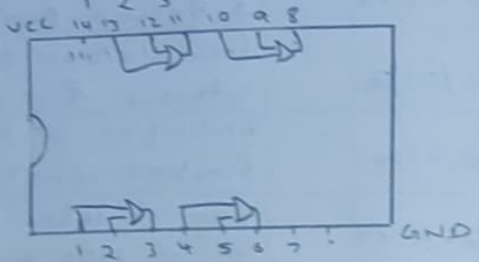
PROGRAM CODE / CIRCUIT DIAGRAMS / SIMPLIFICATIONS

NOT Gate (7404)

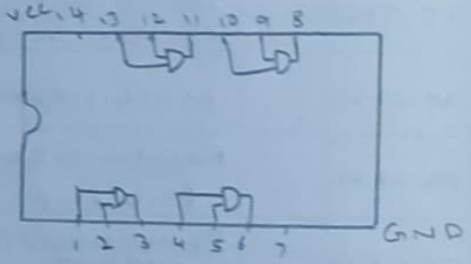


OR (7432)

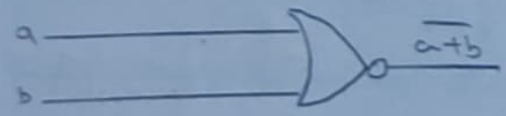


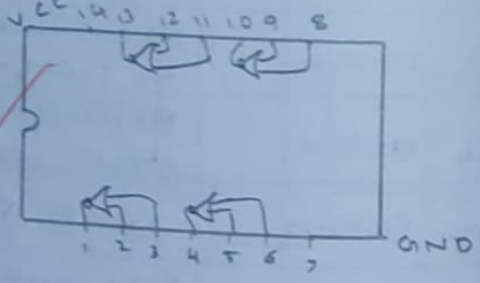


AND (7408)

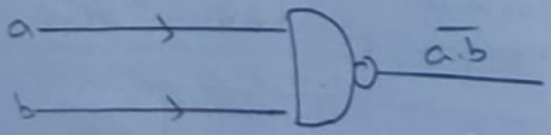


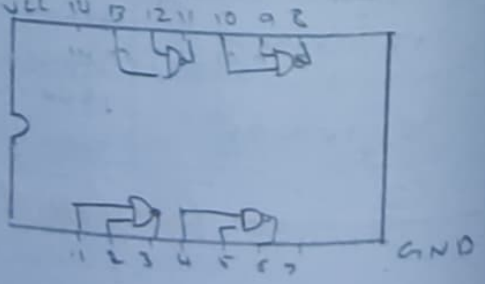
NOR (7402)





NAND (7400)





Experiment No. : 1
Date : 02-07-25

FAMILIARIZATION OF DIGITAL ICs AND BOOLEAN THEOREM VERIFICATION

➢ AIM

To study digital ICs and perform the verification of Boolean theorems using digital logic gates.

➢ COMPONENTS REQUIRED

Digital trainer kit, IC-7404, 7432, 7402, 7408, 7400, 7486

➢ PROCEDURE

1. Study the pin diagrams of the ICs
2. Insert the ICs into the digital trainer kit.
3. Make the necessary connections and connect +5V supply to VCC pin and ground to GND pin of each IC.
4. Verify the truth table for each gate

➢ TRUTH TABLES

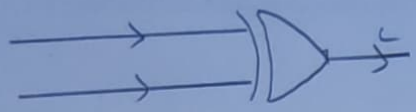
OR GATE

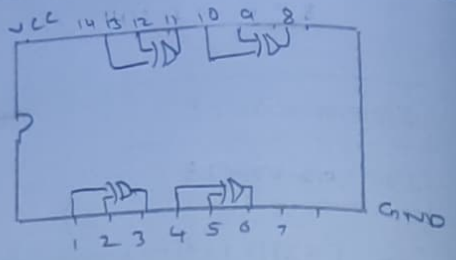
a	b	a+b
0	0	0
0	1	1
1	0	1
1	1	1

NOT GATE

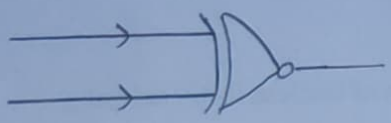
a	$\bar{a}$
1	0
0	1

X-OR



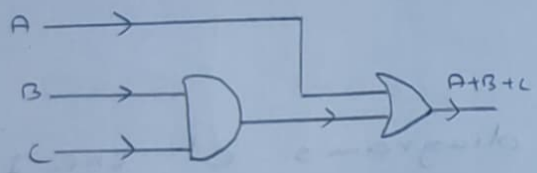


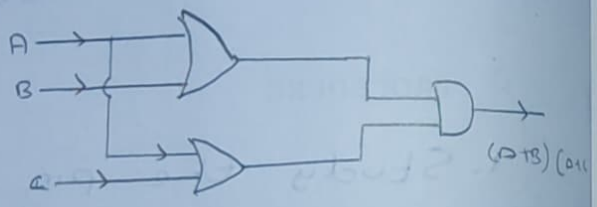
X-NOR



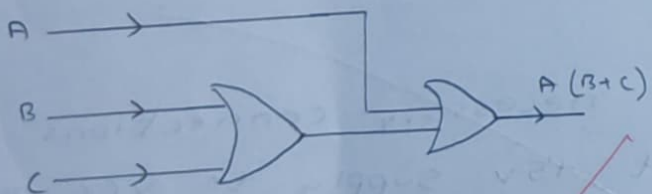
1. Distributive Law

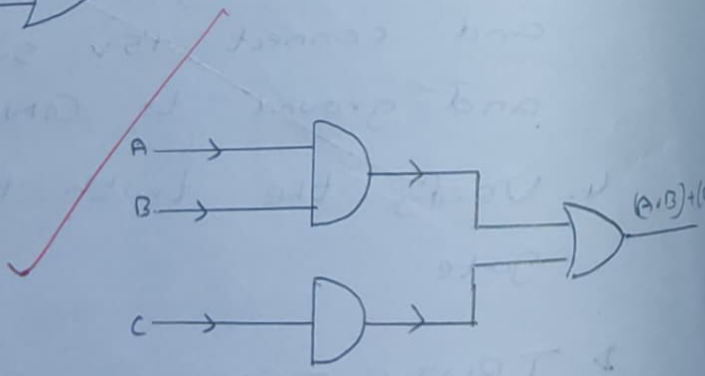
a)





b)

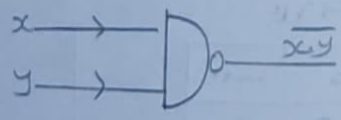


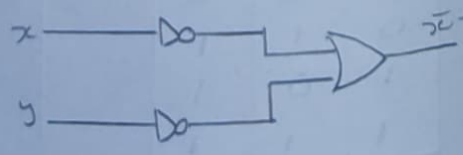


2. De Morgan's Law

a)

b)





AND

a	b	a.b
0	0	0
0	1	0
1	0	0
1	1	1

NOR

a	b	$\overline{a+b}$
0	0	1
0	1	0
1	0	0
1	1	0

NAND

a	b	$\overline{a.b}$
0	0	1
0	1	1
1	0	1
1	1	0

XOR

a	b	c
0	0	0
0	1	1
1	0	1
1	1	0

X-NOR

a	b	c
0	0	1
0	1	0
1	0	0
1	1	1

Verification Of Boolean Expression

1. Distributive Law: The binary operation OR, AND is said to be distributive when

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

   ii) $A.(B+C) = AB + AC$

1 a) Truth Table

A	B	C	BC	A+BC	A+B	A+C	(A+B).(A+C)
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0
0	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1
1	0	1	0	1	1	1	1
1	1	0	0	1	1	1	1
1	1	1	1	1	1	1	1

PERFORMANCE EVALUATION

No	Performance criteria	Excellent (5)	Good (3)	Satisfactory (2)	Poor (1)	Total
1	**Circuit**	Circuit is correctly designed, neatly built, and fully functional without errors	Circuit is mostly correct with minor wiring or logic issues	Circuit has noticeable design or connection errors but shows partial functionality	Circuit is incorrectly designed, poorly connected, or non-functional	5
2	**Output Obtained**	Output is fully correct and matches expected results in all test case	Output is mostly correct with minor mismatches	Output is partially correct with several errors or missing cases	Output is incorrect, missing, or not generated	5
3	**Timetaken**	Output was obtained successfully within 30 minutes	Output was obtained successfully within 45 minutes	Output was obtained successfully within 60 minutes	Output was obtained successfully after 1 hour	5
4	**Record**	Experiment is recorded clearly, completely, and in correct format	Experiment is mostly complete with minor errors or omissions	Experiment is partially recorded with several details missing or unclear	Experiment is poorly documented or largely incomplete	5
					TOTAL MARKS	

CRITERIA	MARKS AWARDED
Performance (20)	20
Viva (5)	5
Total (25)	25

1 b) Truth Table

A	B	C	BC	A(B+C)	AB	AC	(A.B)+(A.C)
0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	1	0	0	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	0	1	0	1	1
1	1	0	0	1	1	0	1
1	1	1	1	1	1	1	1

2. De Morgan's law: It is defined as:

a) $\overline{x+y} = \bar{x}.\bar{y}$ b) $\overline{xy} = \bar{x}+\bar{y}$

x	y	$\bar{x}$	$\bar{y}$	$\overline{x+y}$	$\bar{x}.\bar{y}$
0	0	1	1	1	1
0	1	1	0	0	0
1	0	0	1	0	0
1	1	0	0	0	0

x	y	$\bar{x}$	$\bar{y}$	$\overline{xy}$	$\bar{x}+\bar{y}$
0	0	1	1	1	1
0	1	1	0	1	1
1	0	0	1	1	1
1	1	0	0	0	0

RESULT

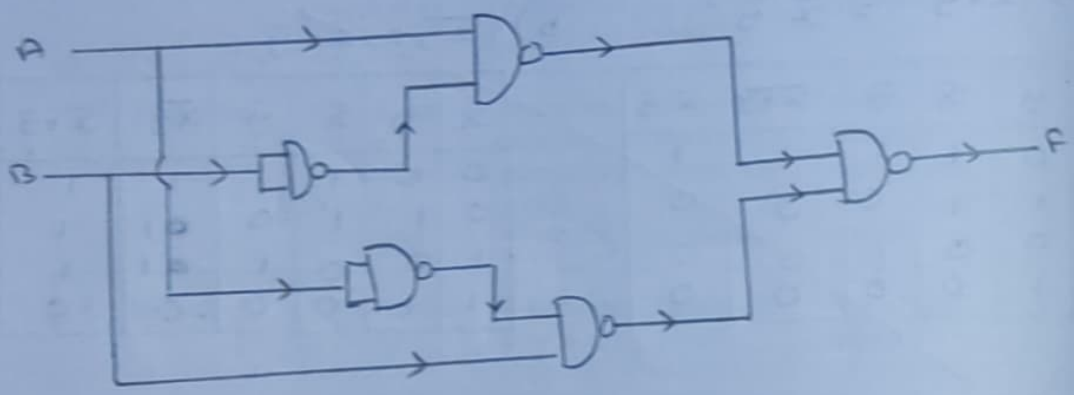
14/07/25 Familiarised different ICs and verified boolean theorem successfully

SOP Expression

$F = A\bar{B} + \bar{A}B$

$\bar{\bar{F}} = \overline{\overline{A\bar{B}} + \overline{\bar{A}B}}$

Equation used : $F = \overline{\overline{A\bar{B}} \cdot \overline{\bar{A}B}}$



Truth Table

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

Experiment No. : 2
Date : 11-07-25

SOP, POS SIMPLIFICATIONS & REALIZATION USING UNIVERSAL GATES

➢ AIM

Realization of SOP and its corresponding POS expression using Universal Gates.

➢ COMPONENTS REQUIRED

Digital trainer kit IC-7400, 7402

➢ PROCEDURE

1. Write the given boolean expression
2. Obtain the minimised SOP & POS expression
3. Convert the SOP expression into NAND realization
4. Convert the POS expression into NOR realization
5. Connect the required ICs on the digital trainer kit.
6. Apply all the input combinations & verify output with truth table.

POS Expression

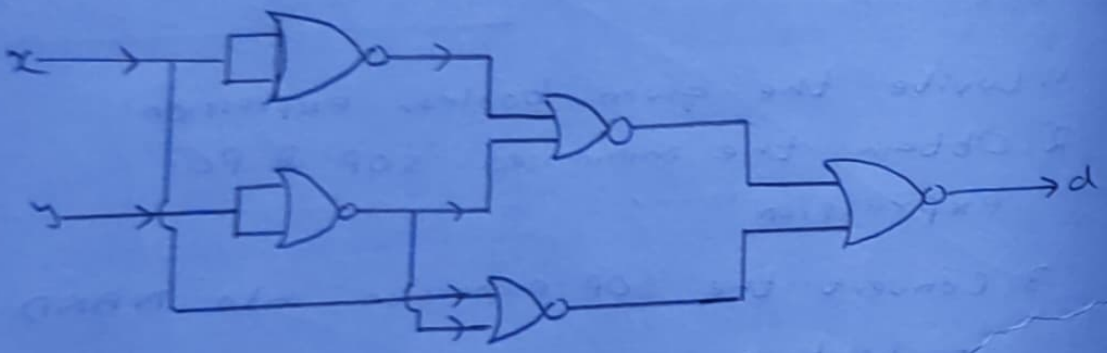
$d = (\bar{x}y)(x+\bar{y})$

$\bar{d} = \overline{\overline{(\bar{x}y)}\,\overline{(x+\bar{y})}}$

$= \overline{\overline{\bar{x}y} + \overline{x+\bar{y}}}$

$= \overline{(\bar{x}+\bar{y}) + (x+\bar{y})}$

Equation used, $d = \overline{\overline{\bar{x}+\bar{y}} + \overline{x+\bar{y}}}$



Truth Table

x	y	d
0	0	0
0	1	1
1	0	1
1	1	0

PERFORMANCE EVALUATION

No	Performance criteria	Excellent (5)	Good (3)	Satisfactory (2)	Poor (1)	Total
1	Circuit	Circuit is correctly designed, neatly built, and fully functional without errors	Circuit is mostly correct with minor wiring or logic issues	Circuit has noticeable design or connection errors but shows partial functionality	Circuit is incorrectly designed, poorly connected, or non-functional	5
2	Output Obtained	Output is fully correct and matches expected results in all test case	Output is mostly correct with minor mismatches	Output is partially correct with several errors or missing cases	Output is incorrect, missing, or not generated	5
3	Timetaken	Output was obtained successfully within 30 minutes	Output was obtained successfully within 45 minutes	Output was obtained successfully within 60 minutes	Output was obtained successfully after 1 hour	5
4	Record	Experiment is recorded clearly, completely, and in correct format	Experiment is mostly complete with minor errors or omissions	Experiment is partially recorded with several details missing or unclear	Experiment is poorly documented or largely incomplete	5
					TOTAL MARKS	20

CRITERIA	MARKS AWARDED
Performance (20)	20
Viva (5)	5
Total (25)	25

RESULT

Realised SOP & POS expression using universal gates

PROGRAM CODE / CIRCUIT DIAGRAMS / SIMPLIFICATIONS

Binary to Gray code Converter

Binary Input				Gray Code Output			
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	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	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
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

Experiment No. : 3
Date : 11-07-25

FAMILIARIZATION OF DIGITAL ICS AND BOOLEAN THEOREM VERIFICATION

➢ AIM

To Design and implement the following:

a) Binary to Grey code converter

b) Grey to Binary code converter

c) Adder Circuits

d) Subtractor Circuits

➢ COMPONENTS REQUIRED

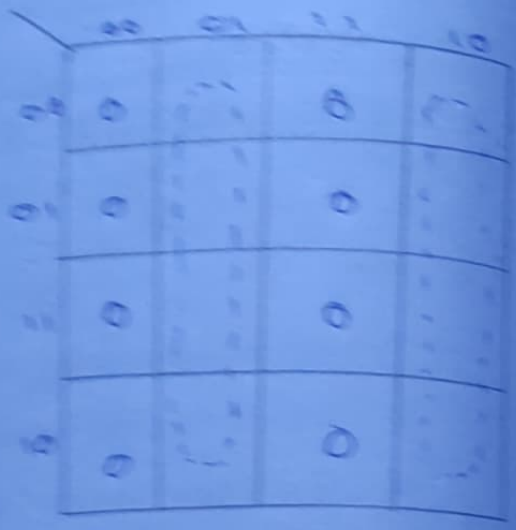
➢ PROCEDURE

For Binary to Grey Code converter:

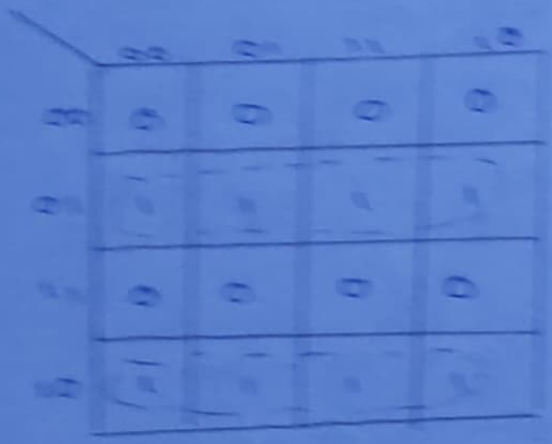
1. Test all the components & Ic packages
2. Set up the circuit of binary code to gray code converter
3. Connect the output in sequence to LED's
4. Feed various 4 bit binary inputs to the circuit through toggle switches on the trainer kit.
5. Observe & verify the corresponding gray code outputs
6. For gray to Binary code converter

Realization using k-maps :

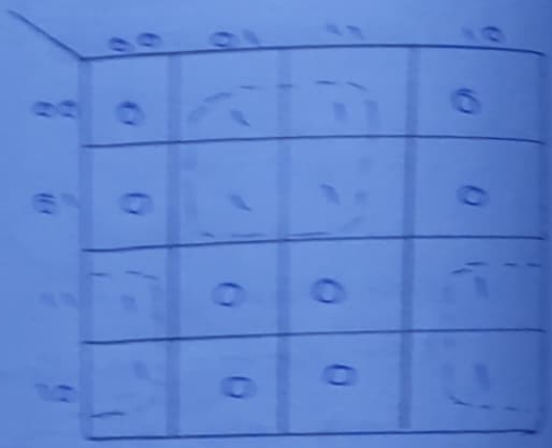
$G_0 = B_0$



$G_{1s} = B_2 \oplus B_3$



$G_{1a} = B_1 \oplus B_2$



$G_2 = B_2 \oplus B_3$

Theory:

Binary Code To Gray Code Converter

To convert a binary number to corresponding Gray Code the following rules are applied:

1. The MSB in the Gray Code is the same as the corresponding bit in a binary number
2. Going from left to right, add each adjacent pair of binary digits to get the next gray code digit
   Disregard carriers

As the first step to design a binary to Gray code converter, set up a truth table with binary numbers $B_3\ B_2\ B_1\ B_0$ & corresponding Gray code number $G_3\ G_2\ G_1 G_0$ set up a circuit realizing the simplified logic expressions obtained using k-maps for G's as the functions of B's

Procedure:

For Gray to Binary Code converter

1. Modify the circuit to set up a gray code to binary code converter.
2. Feed gray code to the inputs of the circuit
3. Observe & verify the corresponding binary code outputs

Gray To Binary Code Converter

Gray Code Input				Binary Output			
G_3	G_2	G_1	G_0	B_3	B_2	B_1	B_0
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	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	1	1	1	1	0	1
1	0	0	1	1	1	1	0
1	0	0	0	1	1	1	1

Theory

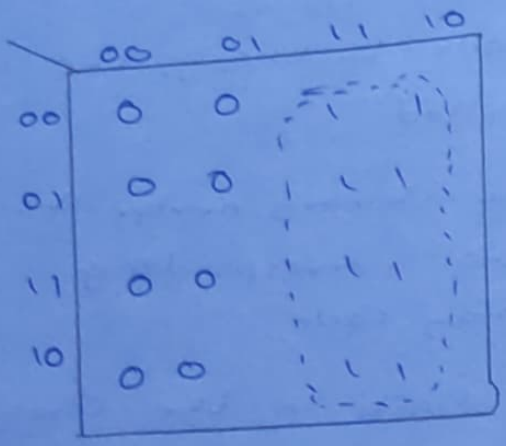
Gray code to Binary Code converter

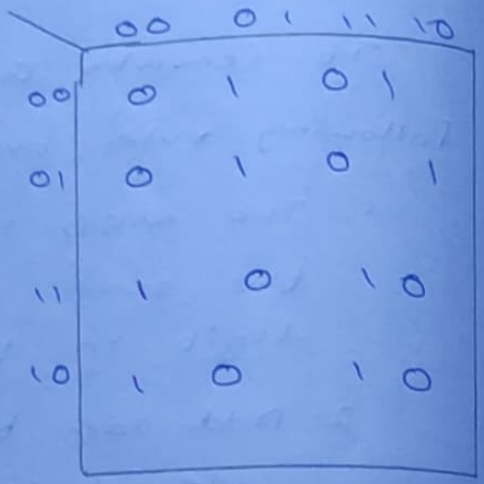
To Convert Gray Code to binary, the following rules are applied

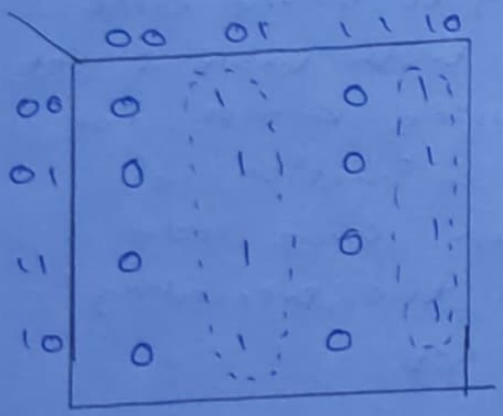
1. The MSD in the binary number is the same as the corresponding digit in the Gray code
2. Add each binary digit in the Gray Code digit in the next adjacent position. Disregard carry

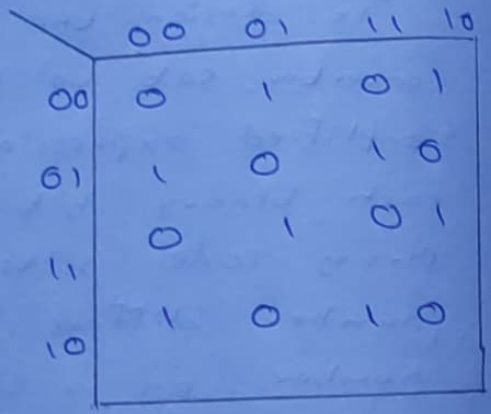
To design the Gray to binary code converter, set up the truth table & get simplified expression using K-maps for each binary bit as a function of Gray code bits Each Gray code number differs from the preceding number by a single bit

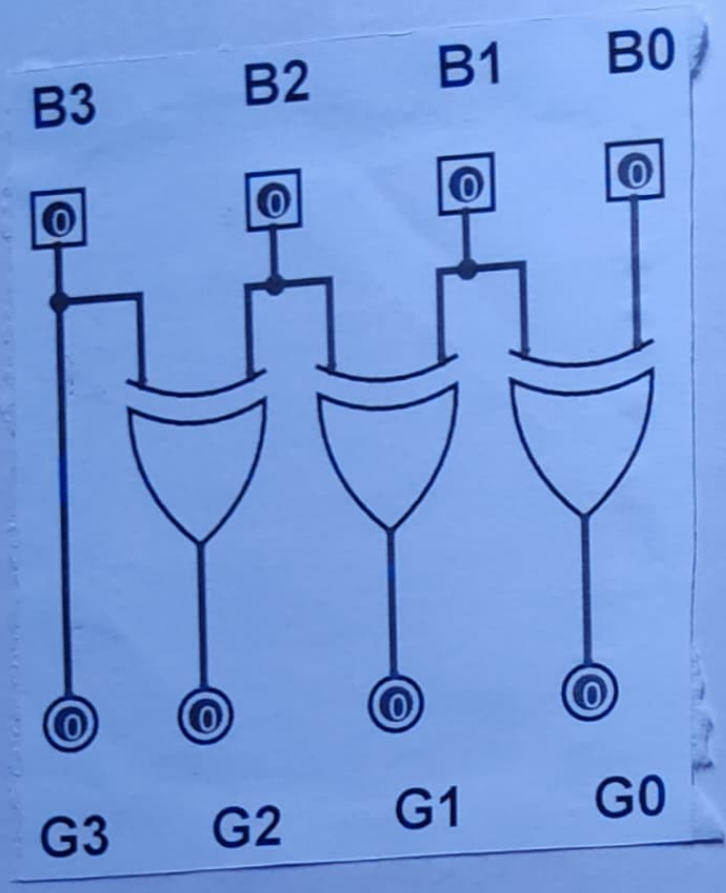
Realizabpon Usng kmap

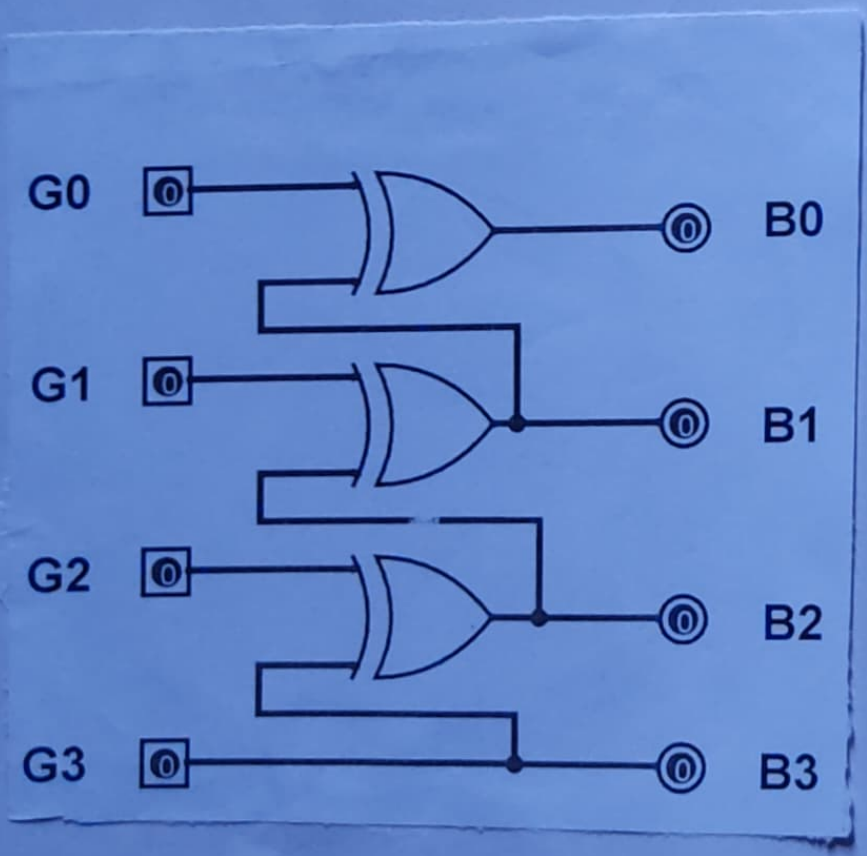


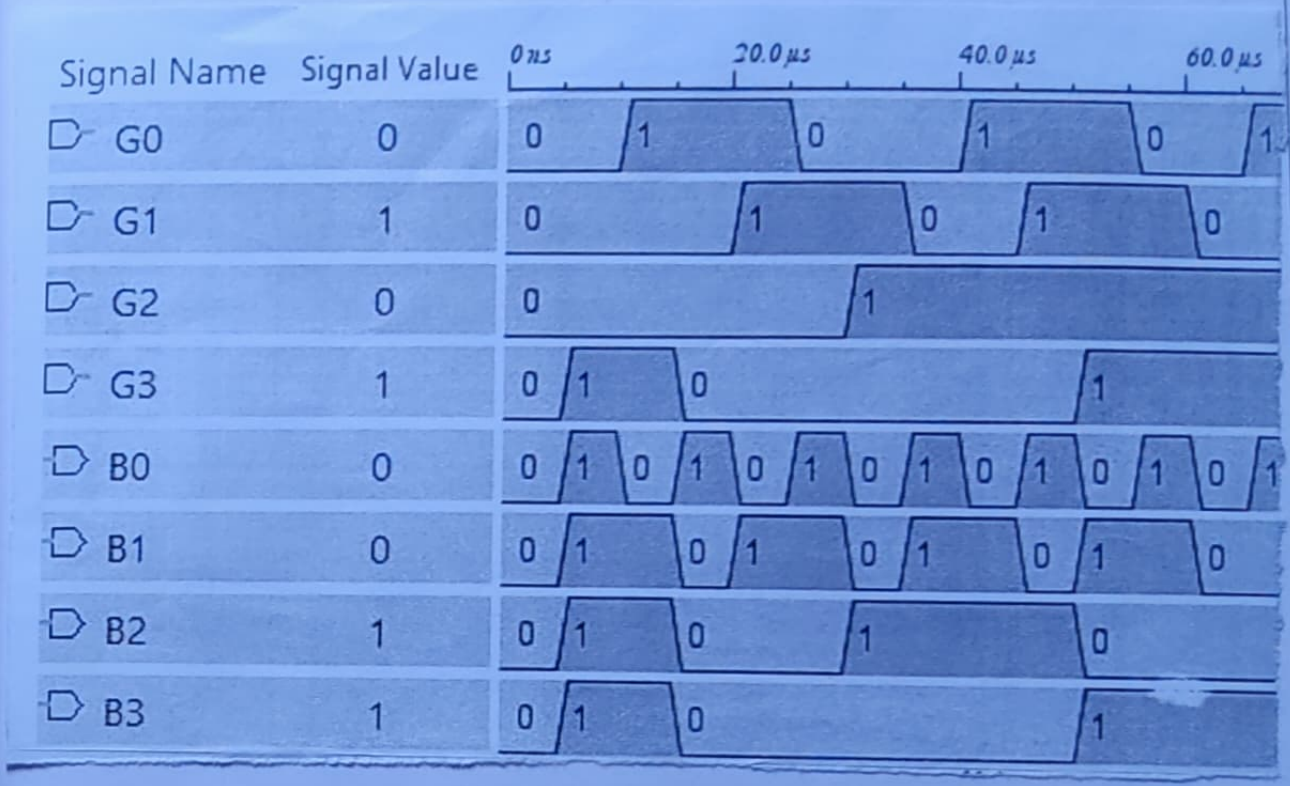


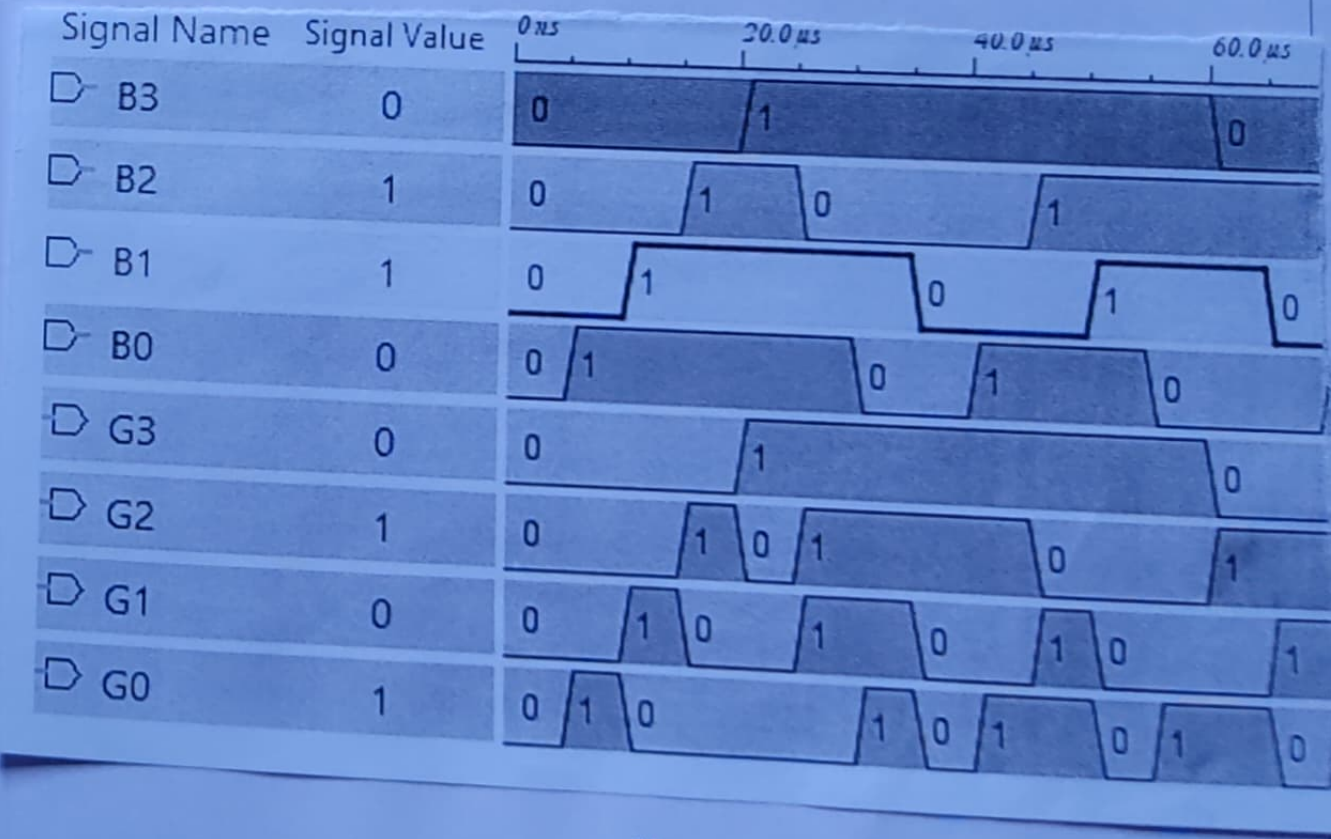





B3
B2
B1
B0
G3
G2
G1
G0


G0
B0
G1
B1
G2
B2
G3
B3


Signal Name
Signal Value
0 ns
20.0 µs
40.0 µs
60.0 µs
G0 0
G1 1
G2 0
G3 1
B0 0
B1 0
B2 1
B3 1


Signal Name
Signal Value
0 ns
20.0 µs
40.0 µs
60.0 µs
B3 0
B2 1
B1 1
B0 0
G3 0
G2 1
G1 0
G0 1

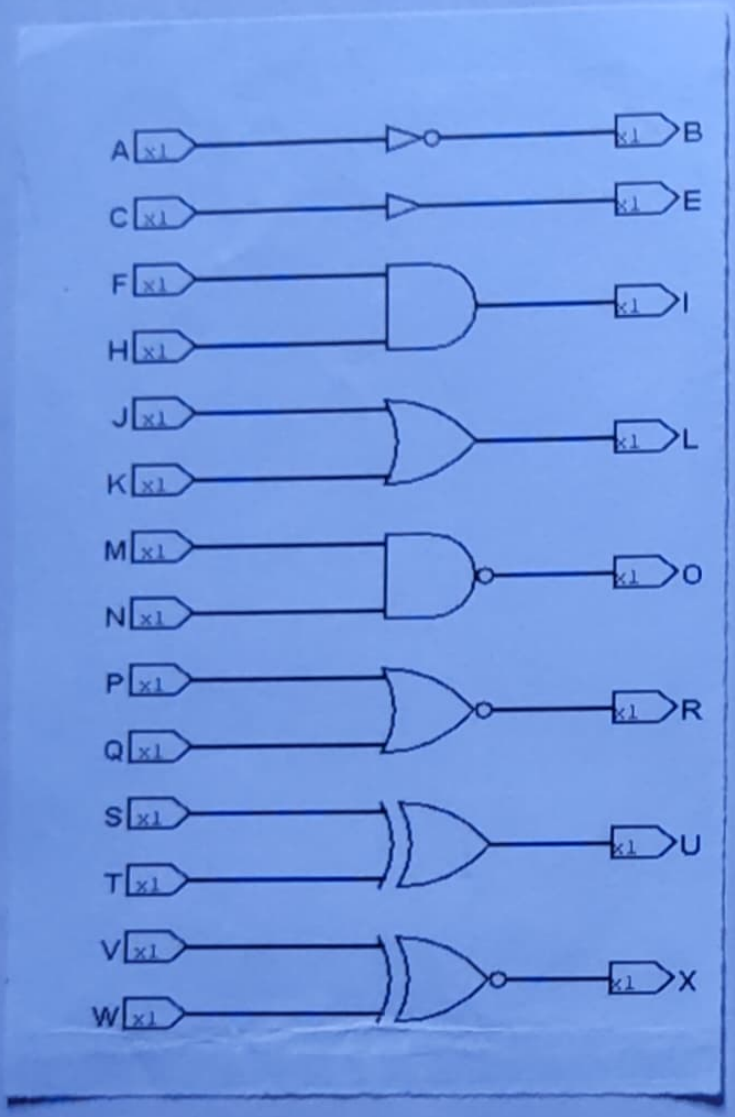
PERFORMANCE EVALUATION

No	Performance criteria	Excellent (5)	Good (3)	Satisfactory (2)	Poor (1)	Total
1	**Circuit**	Circuit is correctly designed, neatly built, and fully functional without errors	Circuit is mostly correct with minor wiring or logic issues	Circuit has noticeable design or connection errors but shows partial functionality	Circuit is incorrectly designed, poorly connected, or non-functional	5
2	**Output Obtained**	Output is fully correct and matches expected results in all test case	Output is mostly correct with minor mismatches	Output is partially correct with several errors or missing cases	Output is incorrect, missing, or not generated	5
3	**Timetaken**	Output was obtained successfully within 30 minutes	Output was obtained successfully within 45 minutes	Output was obtained successfully within 60 minutes	Output was obtained successfully after 1 hour	5
4	**Record**	Experiment is recorded clearly, completely, and in correct format	Experiment is mostly complete with minor errors or omissions	Experiment is partially recorded with several details missing or unclear	Experiment is poorly documented or largely incomplete	5
					TOTAL MARKS	20

CRITERIA	MARKS AWARDED
Performance (20)	20
Viva (5)	5
Total (25)	25

RESULT

Studied & verified of binary to Gray & Gray to Binary code converted


A
C
F
H
J
K
M
N
P
Q
S
T
V
W
B
E
I
L
O
R
U
X

Experiment No. : 4
Date : 16-07-25

FAMILIARIZATION OF SIMULATION SOFTWARE, REALIZATION AND ANALYSIS OF BASIC LOGIC GATES AND ITS WAVEFORMS

➢ AIM

To familiarize the working of circuit simulation software and to realize the basic logic gates. Also analyze their waveforms.

➢ COMPONENTS REQUIRED

Logism evolution software tool.

➢ PROCEDURE

1. Open Software tool
2. Create new file
3. Implement logic gate
4. Give input as per truth table
5. Generate.

TRUTH TABLE

NOT GATE

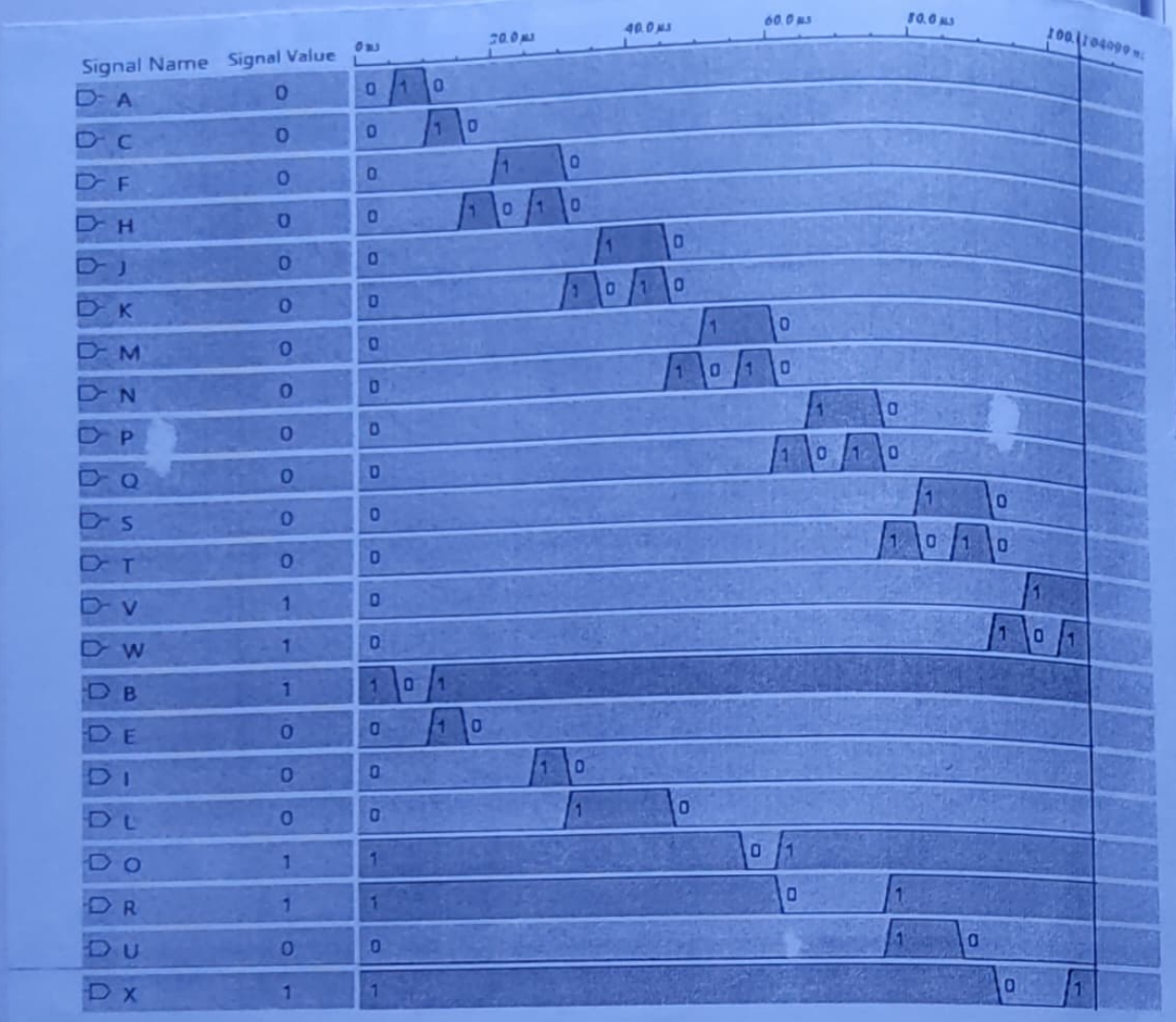
A	B
0	1
1	0

OR GATE

P	G	H
0	0	0
0	1	1
1	0	1
1	1	1

AND GATE

C	D	E
0	0	0
0	1	0
1	0	0
1	1	1


Signal Name
Signal Value
A 0
C 0
F 0
H 0
J 0
K 0
M 0
N 0
P 0
Q 0
S 0
T 0
V 1
W 1
B 1
E 0
I 0
L 0
O 1
R 1
U 0
X 1

NOR GATE

L	M	N
0	0	1
0	1	0
1	0	0
1	1	0

NAND

I	J	K
0	0	1
0	1	1
1	0	1
1	1	0

XNOR GATE

R	S	T
0	0	1
0	1	0
1	0	0
1	1	1

X-OR Gate

O	R	Q
0	0	0
0	1	1
1	0	1
1	1	0

PERFORMANCE EVALUATION

No	Performance criteria	Excellent (5)	Good (3)	Satisfactory (2)	Poor (1)	Total
1	Code	Code is correct, efficient, well-structured, and fully meets design requirements	Code works with minor issues or inefficiencies and meets most design goals	Code has multiple errors or lacks clarity but shows basic functionality	Code is mostly incorrect, incomplete, or fails to meet design requirements	5
2	Output Obtained	Output is fully correct and matches expected results in all test case	Output is mostly correct with minor mismatches	Output is partially correct with several errors or missing cases	Output is incorrect, missing, or not generated	5
3	Timetaken	Output was obtained successfully within 30 minutes	Output was obtained successfully within 45 minutes	Output was obtained successfully within 60 minutes	Output was obtained successfully after 1 hour	5
4	Record	Experiment is recorded clearly, completely, and in correct format	Experiment is mostly complete with minor errors or omissions	Experiment is partially recorded with several details missing or unclear	Experiment is poorly documented or largely incomplete	5
					TOTAL MARKS	

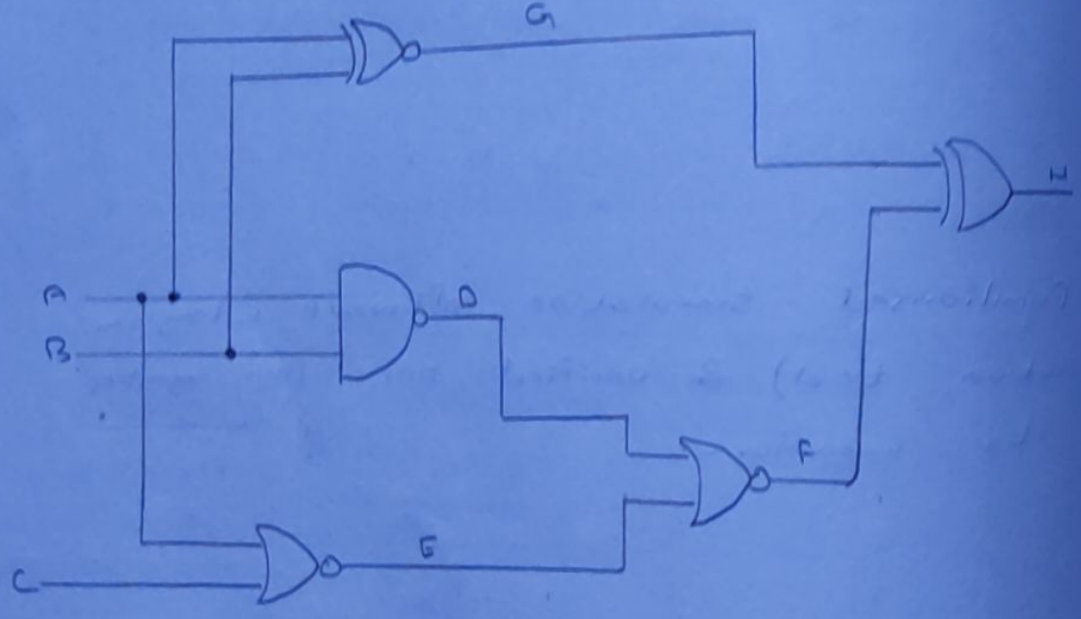
CRITERIA	MARKS AWARDED
Performance (20)	20
Viva (5)	5
Total (25)	25

RESULT

Familiarised simulation software (logism Evolution tool) & verified basic logic gates & its waveforms

PROGRAM CODE / CIRCUIT DIAGRAMS / SIMPLIFICATIONS

A	B	C	$(AB)'$	$(A+C)'$	$A \ominus B$	$(AB)' + (A+C)'$	$(A+B) \odot (A+C)$
0	0	0	1	1	1	1	0
0	0	1	1	0	1	1	0
0	1	0	1	1	0	1	1
0	1	1	1	0	0	1	1
1	0	0	1	0	0	1	1
1	0	1	1	0	0	1	1
1	1	0	0	0	1	0	1
1	1	1	0	0	1	0	1



Experiment No. : 5
Date : 16-07-25

REALIZATION OF BOOLEAN FUNCTION USING BASIC LOGIC GATES AND ANALYZING ITS WAVEFORMS

➢ AIM

To realize the given Boolean function using basic gates and to verify its truth table. Analyze its waveforms.

➢ COMPONENTS REQUIRED

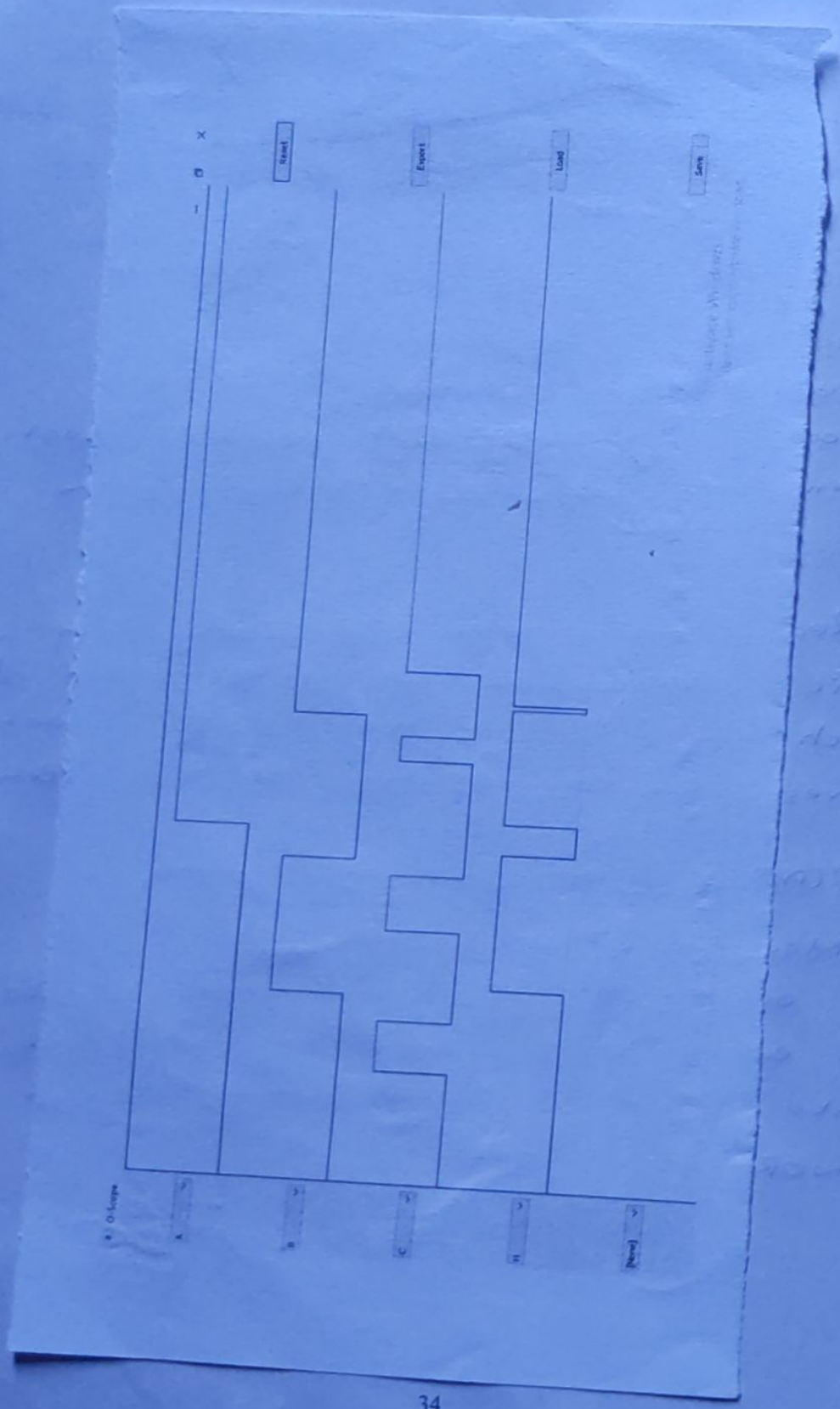
- Digital Logic simulator
- Computer or laptop with simulation tool
- Virtual switches clock & logic problem

➢ PROCEDURE

Realize the given boolean function using basic gates write the truth table of each term & find the output of each corresponding row. Analysis it's waveform

$F(A,B,C) = [(AB)' + (A+C)'] \oplus (A \oplus B)$

Here AB = AND gate $(AB)'$ Negate the output A+C = OR gate. $(A+B)'$ Negate the output Then $(AB)' + (A+C)'$. Here OR gate is used Then finally XOR & XNOR Gates are used.


Reset
Export
Load
Save
A
B
C
D
[New]

PERFORMANCE EVALUATION

No	Performance criteria	Excellent (5)	Good (3)	Satisfactory (2)	Poor (1)	Total
1	Code	Code is correct, efficient, well-structured, and fully meets design requirements	Code works with minor issues or inefficiencies and meets most design goals	Code has multiple errors or lacks clarity but shows basic functionality	Code is mostly incorrect, incomplete, or fails to meet design requirements	5
2	Output Obtained	Output is fully correct and matches expected results in all test case	Output is mostly correct with minor mismatches	Output is partially correct with several errors or missing cases	Output is incorrect, missing, or not generated	5
3	Timetaken	Output was obtained successfully within 30 minutes	Output was obtained successfully within 45 minutes	Output was obtained successfully within 60 minutes	Output was obtained successfully after 1 hour	5
4	Record	Experiment is recorded clearly, completely, and in correct format	Experiment is mostly complete with minor errors or omissions	Experiment is partially recorded with several details missing or unclear	Experiment is poorly documented or largely incomplete	5
					TOTAL MARKS	20

CRITERIA	MARKS AWARDED
Performance (20)	20
Viva (5)	5
Total (25)	25

RESULT

Boolean function using basic gates where simulates successfully the observed output & waveform matches the expected behaviour

```
// Gate-level AND Gate
module and_gate_gatelevel (input A, B, output Y);
and (Y, A, B);
endmodule
// Dataflow
module and_gate_dataflow (input A, B, output Y);
assign Y = A & B;
endmodule
// Behavioral
module and_gate_behavioral (input A, B, output reg Y);
always @ (A or B)
if (A & B) Y = 1; else Y = 0;
endmodule
//OR Gate
module or_gate_gatelevel (input A, B, output Y);
or (Y, A, B);
endmodule
module or_gate_dataflow (input A, B, output Y);
assign Y = A | B;
endmodule
module or_gate_behavioral (input A, B, output reg Y);
always @ (A or B)
if (A | B) Y = 1; else Y = 0;
endmodule
//NOT Gate
module not_gate_gatelevel (input A, output Y);
not (Y, A);
endmodule
module not_gate_dataflow (input A, output Y);
assign Y = ~A;
endmodule
module not_gate_behavioral (input A, output reg Y);
always @ (A)
if (~A) Y = 1; else Y = 0;
endmodule
// XOR Gate
module xor_gate_gatelevel (input A, B, output Y);
xor (Y, A, B);
endmodule
module xor_gate_dataflow (input A, B, output Y);
```

Experiment No. : 6
Date : 23-07-25

FAMILIARIZATION OF VERILOG HDL MODELING OF THE BASIC GATES

➢ AIM

To Familiarize Verilog HDL-Modeling of the basic gates using:

i. Gate level modeling

ii. Behavioral modeling

iii. Structural modeling

iv. Data Flow modeling

➢ COMPONENTS REQUIRED

- EDA playground / Cavus verilog
- GTK wave for waveform visualization
- Any text/code editor (for offline use

➢ PROCEDURE

1. ~~Procedure~~ Write each node version in the verilog simulator
2. Compile & simulate.
3. View waveform using GTK wave
4. Validate against truth tables

```
assign Y = A ^ B;

endmodule

module xor_gate_behavioral (input A, B, output reg Y);

always @ (A or B)

if (A == B) Y = 1; else Y = 0;

endmodule

//Structural

module not_gate (input A, output Y);

assign Y = ~A;

endmodule

module and_gate (input A, B, output Y);

assign Y = A & B;

endmodule

module or_gate (input A, B, output Y);

assign Y = A | B;

endmodule

module xor_structural (input A, B, output Y);

wire nA, nB, t1, t2;

not_gate U1 (.A(A), .Y(nA));

not_gate U2 (.A(B), .Y(nB));

and_gate U3 (.A(nA), .B(B), .Y(t1)); // nA

and_gate U4 (.A(A), .B(nB), .Y(t2)); // nB

or_gate U5 (.A(t1), .B(t2), .Y(Y)); // nA + nB

endmodule


//Testbench

module tb_basic_gates;

reg A, B;

wire Y_and_gate, Y_and_data, Y_and_behav;

wire Y_or_gate, Y_or_data, Y_or_behav;

wire Y_not_gate, Y_not_data, Y_not_behav;

wire Y_xor_gate, Y_xor_data, Y_xor_behav;

and_gate_gatelevel U1 (A, B, Y_and_gate);

and_gate_dataflow U2 (A, B, Y_and_data);

and_gate_behavioral U3 (A, B, Y_and_behav);

or_gate_gatelevel U4 (A, B, Y_or_gate);

or_gate_dataflow U5 (A, B, Y_or_data);

or_gate_behavioral U6 (A, B, Y_or_behav);

not_gate_gatelevel U7 (A, Y_not_gate);

not_gate_dataflow U8 (A, Y_not_data);
```

```
not_gate_behavioral u5(a, y_not_behav);

xor_gate_gatelevel u10(a, b, y_xor_gate);

xor_gate_dataflow u11(a, b, y_xor_data);

xor_gate_behavioral u12(a, b, y_xor_behav);

initial begin

$dumpfile("gates_wave.vcd");

$dumpvars(0, tb_basic_gates);

$display("A B | AND(G/D/B) | OR(G/D/B) | NOT(G/D/B) | XOR(G/D/B)");

$display("-----------------------------------");

for (integer i=0; i<4; i=i+1) begin

{a, b} = i[1:0];

$display("%0d %0d | %0d%0d%0d | %0d%0d%0d | %0d%0d%0d | %0d%0d%0d",

a, b,

y_and_gate, y_and_data, y_and_behav,

y_or_gate, y_or_data, y_or_behav,

y_not_gate, y_not_data, y_not_behav,

y_xor_gate, y_xor_data, y_xor_behav);

end

$finish;

end

endmodule

module tb_structural;

reg A, B;

wire F;

xor_structural UUT (.A(A), .B(B), .F(F));

initial begin

$dumpfile("xor_structural.vcd");

$dumpvars(0, tb_structural);

$display("A B | F");

$display("-----");

for (integer i=0; i<4; i=i+1) begin

{A, B} = i[1:0];

$display("%0d %0d | %0d", A, B, F);

end

$finish;

end

endmodule
```

PERFORMANCE EVALUATION

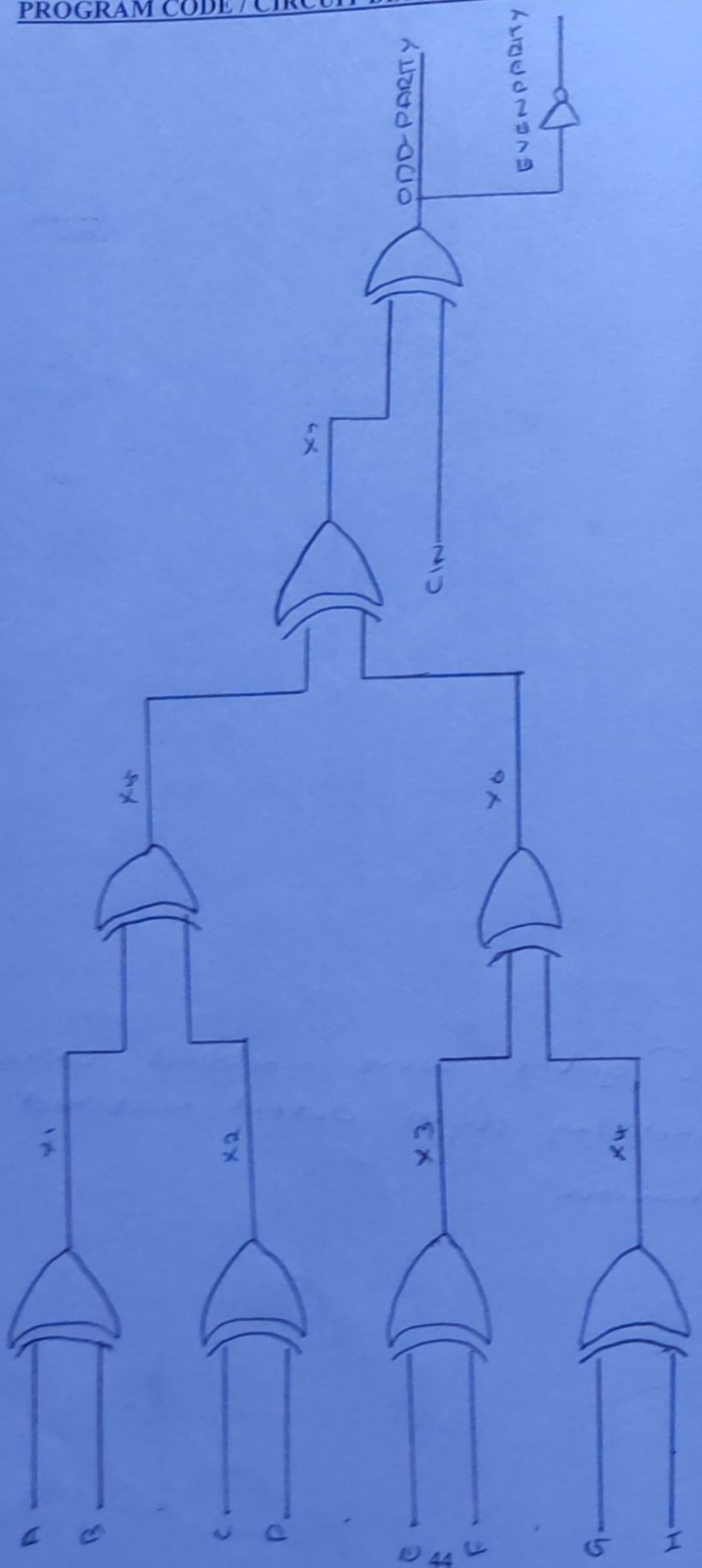
No	Performance criteria	Excellent (5)	Good (3)	Satisfactory (2)	Poor (1)	Total
1	Code	Code is correct, efficient, well-structured, and fully meets design requirements	Code works with minor issues or inefficiencies and meets most design goals	Code has multiple errors or lacks clarity but shows basic functionality	Code is mostly incorrect, incomplete, or fails to meet design requirements	5
2	Output Obtained	Output is fully correct and matches expected results in all test case	Output is mostly correct with minor mismatches	Output is partially correct with several errors or missing cases	Output is incorrect, missing, or not generated	5
3	Timetaken	Output was obtained successfully within 30 minutes	Output was obtained successfully within 45 minutes	Output was obtained successfully within 60 minutes	Output was obtained successfully after 1 hour	5
4	Record	Experiment is recorded clearly, completely, and in correct format	Experiment is mostly complete with minor errors or omissions	Experiment is partially recorded with several details missing or unclear	Experiment is poorly documented or largely incomplete	5
					TOTAL MARKS	20

CRITERIA	MARKS AWARDED
Performance (20)	20
Viva (5)	5
Total (25)	25

RESULT

Basic logic gates were modelled using all verilog modeling styles. Output matched expected behaviour.

PROGRAM CODE / CIRCUIT DIAGRAMS / SIMPLIFICATIONS



Experiment No. : 7
Date : 28-07-25

PARITY GENERATOR & CHECKER USING IC74180

➢ AIM

To design and implement Parity Generator / Checker using MSI Device IC74180.

➢ COMPONENTS REQUIRED

- Digital Logic Simulator
- Basic Logic gates
- Input wires & LEDs

➢ PROCEDURE

- Open cedar logic & start a new schematic place input pins & name them exactly ABCDEFGHCIN
- If the simulator let's you create a hierarchical block set these as the block inputs

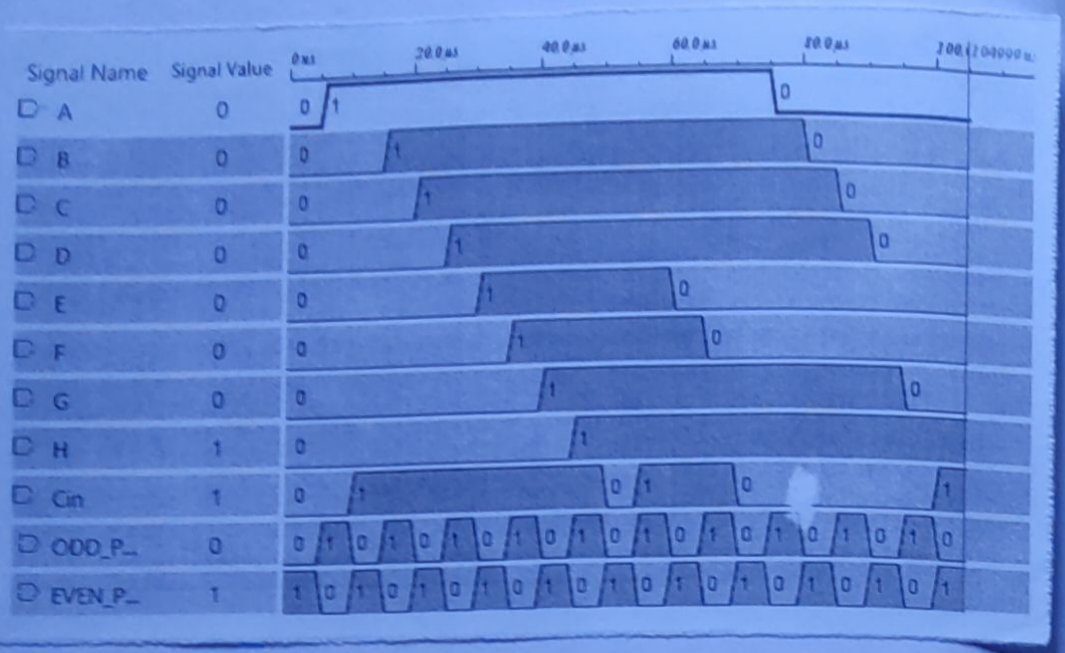
Build the XOR tree (option 1 or option 2)

Option 1: Place XOR gate & wire as described

Option 2: Place NAND gates to create XOR cells then wire them the same way.

Create ΣODD output: Connect output of final XOR ($x_7 \oplus C_{IN}$) to an output pin named SUM-ODD

Create ΣEVEN output: Place an inverter (NOT) or NAND inverter and connect it's input to SUM_ODD name the output pin SUM-EVEN


Signal Name
Signal Value
A
0
B
0
C
0
D
0
E
0
F
0
G
0
H
1
Cin
1
ODD_P...
0
EVEN_P...
1

Label pins & tidy wires:

- Make sure all inputs are labelled & output named as above.
- Optionally add power rails (VCC/GND) for convenience if your simulator requires it.

Group into a subcircuit / save as a custom component

- Use cedar Logic's 'Create subcircuit' (or equivalent) to turn the selected schematic into a reasonable IC named PAR8_EQ-74186
- Define it's interface pins in the same order you want them to appear in the component palette

Test the module:

- Create a small test bench that drives different combinations of A....H & CIN and display SUM_ODD & SUM_EVEN
- Quick checks:
  - All Zeroes input + CIN=0 → SUM_ODD=0, SUM_EVEN=1
  - Single 1 among A....H with CIN=0 → SUM_ODD=1, SUM_EVEN=0
  - Toggle CIN should invert SUM_ODD/SUM_EVEN accordingly

Place XOR gates & connect them in a balanced tree (pairwise) to reduce depth.

- X1 = A⊕B
- X2 = C⊕D

PERFORMANCE EVALUATION

No	Performance criteria	Excellent (5)	Good (3)	Satisfactory (2)	Poor (1)	Total
1	Code	Code is correct, efficient, well-structured, and fully meets design requirements	Code works with minor issues or inefficiencies and meets most design goals	Code has multiple errors or lacks clarity but shows basic functionality	Code is mostly incorrect, incomplete, or fails to meet design requirements	5
2	Output Obtained	Output is fully correct and matches expected results in all test case	Output is mostly correct with minor mismatches	Output is partially correct with several errors or missing cases	Output is incorrect, missing, or not generated	5
3	Timetaken	Output was obtained successfully within 30 minutes	Output was obtained successfully within 45 minutes	Output was obtained successfully within 60 minutes	Output was obtained successfully after 1 hour	5
4	Record	Experiment is recorded clearly, completely, and in correct format	Experiment is mostly complete with minor errors or omissions	Experiment is partially recorded with several details missing or unclear	Experiment is poorly documented or largely incomplete	5
					TOTAL MARKS	20

CRITERIA	MARKS AWARDED
Performance (20)	20
Viva (5)	5
Total (25)	25

- $x_3 = E \oplus F$
- $x_4 = G \oplus H$
- $x_5 = x_1 \oplus x_2$
- $x_6 = x_3 \oplus x_4$
- $x_7 = x_5 \oplus x_6$
- $SUM\_ODD = x_7 \oplus CIN$

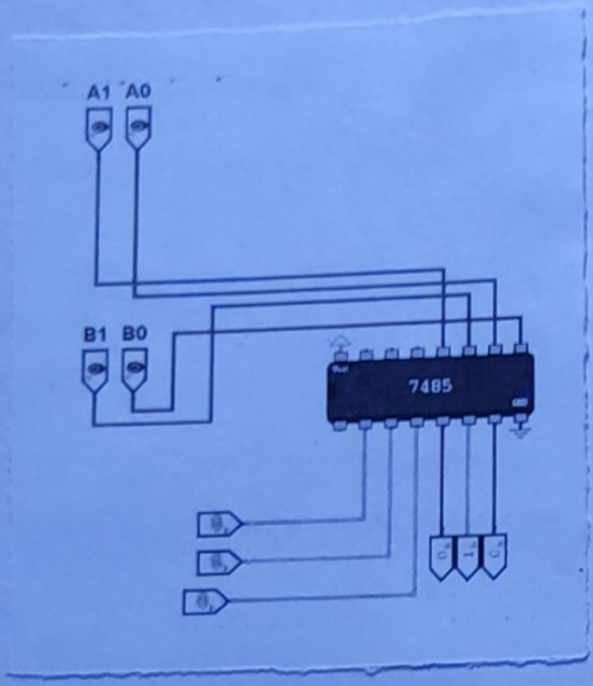
Invert for ΣEVEN:

- $SUM\_EVEN = NOT(SUM\_ODD)$

RESULT

Parity generator & checker circuit are successfully implemented using IC74186 & verified the truth table

PROGRAM CODE / CIRCUIT DIAGRAMS / SIMPLIFICATIONS



Experiment No. : 8
Date : 28-07-25

MAGNITUDE COMPARATOR USING MSI DEVICE IC7485

- **AIM**

To Design and implement Magnitude Comparator using MSI Device IC7485.

- **COMPONENTS REQUIRED**

• IC7485

- **PROCEDURE**

1. Connect the inputs for A & B in respective pin
2. Cascade the input using pins 2, 3 & 4
3. Connect output the pins 5, 6, & 7
4. connect V_{cc} & GND to Pin V_{cc} & GNS

Theory:

IC trainer kit:

A magnitude digital comparator is a combinational circuit mat compares two digital or binary number in order to find out whether one binary number is equal less than or greater than the other bing number. we logically design a circuit for where we will have two inputs one for A and the other for B & we have three output terminal onto for

Truth Table

A_1	A_0	B_1	B_0	A<B	A=B	A>B
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	1	0	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	0

A>B Condition one for A=B condition & one for A<B Condition

PERFORMANCE EVALUATION

No	Performance criteria	Excellent (5)	Good (3)	Satisfactory (2)	Poor (1)	Total
1	Code	Code is correct, efficient, well-structured, and fully meets design requirements	Code works with minor issues or inefficiencies and meets most design goals	Code has multiple errors or lacks clarity but shows basic functionality	Code is mostly incorrect, incomplete, or fails to meet design requirements	5
2	Output Obtained	Output is fully correct and matches expected results in all test case	Output is mostly correct with minor mismatches	Output is partially correct with several errors or missing cases	Output is incorrect, missing, or not generated	5
3	Timetaken	Output was obtained successfully within 30 minutes	Output was obtained successfully within 45 minutes	Output was obtained successfully within 60 minutes	Output was obtained successfully after 1 hour	5
4	Record	Experiment is recorded clearly, completely, and in correct format	Experiment is mostly complete with minor errors or omissions	Experiment is partially recorded with several details missing or unclear	Experiment is poorly documented or largely incomplete	5
					TOTAL MARKS	20

CRITERIA	MARKS AWARDED
Performance (20)	20
Viva (5)	5
Total (25)	25

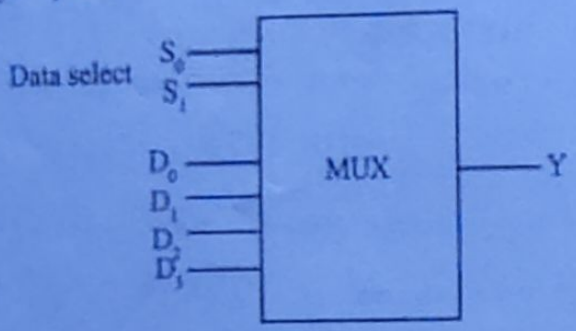
RESULT

A Magnitude comparator was designed using IC7485 and bit comparison done sucessfully

CIRCUIT DIAGRAM

4:1 Multiplexer

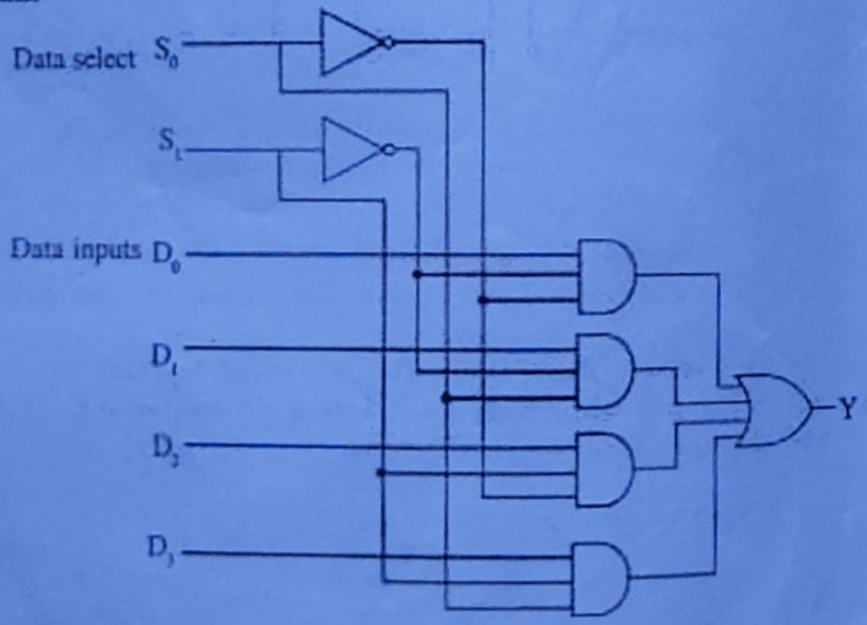
Logic symbol



Truth table

S_1	S_0	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

Circuit diagram



Experiment No. : 9
Date : 28-07-25

MULTIPLEXER

➢ AIM

To Design and implement Boolean function using MUX IC.

➢ COMPONENTS REQUIRED

IC Trainerkit , IC-7404 , IC-7432 , IC-7411

➢ PROCEDURE

Multiplexer is a combinational circuit which can select any one of the inputs & route it to the output A multiplexer has data input lines, data select lines & output. The logic symbol of a 4line to 1 line multiplexer is shown in figure.

According to the 2 bit binary code on the data select inputs, corresponding data input line will be selected & routed to the output for eg:- if S_1S_0 is 00, D_0 will be selected, if S_1S_0 is 01, D_1 will be selected & so on.

$$Y = D_0\overline{S_1}\overline{S_0} + D_1\overline{S_1}S_0 + D_2S_1\overline{S_0} + D_3S_1S_0$$

This boolean expression can be realized using gates

De Multiplexer

Demultiplexer does the reverse operation of multiplexer. The data on the line is distinguished to that output line whose binary code is given on the data select lines. ie, S_1S_0 Each o/p line has it's own unique binary code

For 4:1 Multiplexer

1) Test all the components & IC packages
2) Set up the circuit for multiplexer as per the diagram.
3) Give any random binary combinations at D_0 through D_3
4) Feed all 4 combinations at S_1S_0 & Observe the corresponding output.
5) Verify that the circuit functions as a multiplexer

For 1:4 Demultiplexer

1) Setup the circuit for demultiplexer for the diagram
2) Give logic 1 on data (0) input line.

PERFORMANCE EVALUATION

No	Performance criteria	Excellent (5)	Good (3)	Satisfactory (2)	Poor (1)	Total
1	Code	Code is correct, efficient, well-structured, and fully meets design requirements	Code works with minor issues or inefficiencies and meets most design goals	Code has multiple errors or lacks clarity but shows basic functionality	Code is mostly incorrect, incomplete, or fails to meet design requirements	5
2	Output Obtained	Output is fully correct and matches expected results in all test case	Output is mostly correct with minor mismatches	Output is partially correct with several errors or missing cases	Output is incorrect, missing, or not generated	5
3	Timetaken	Output was obtained successfully within 30 minutes	Output was obtained successfully within 45 minutes	Output was obtained successfully within 60 minutes	Output was obtained successfully after 1 hour	5
4	Record	Experiment is recorded clearly, completely, and in correct format	Experiment is mostly complete with minor errors or omissions	Experiment is partially recorded with several details missing or unclear	Experiment is poorly documented or largely incomplete	5
					TOTAL MARKS	20

CRITERIA	MARKS AWARDED
Performance (20)	20
Viva (5)	5
Total (25)	25

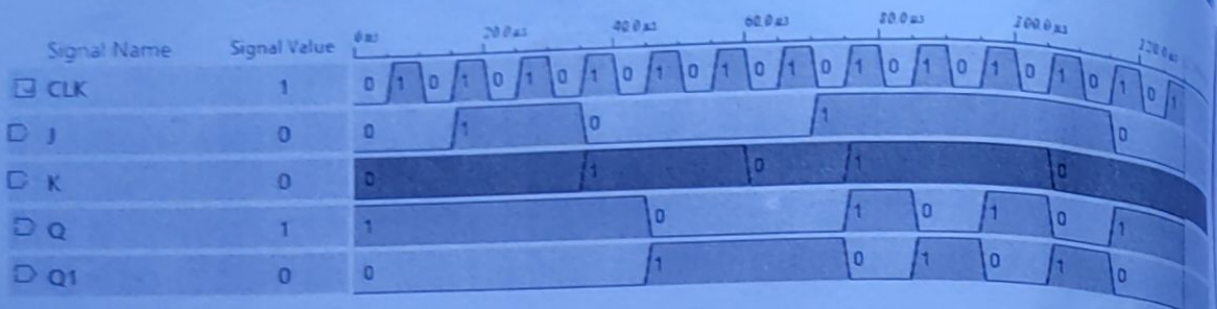
3) Feed all four combinations on by one at S1S0 & observe the data at the corresponding output line

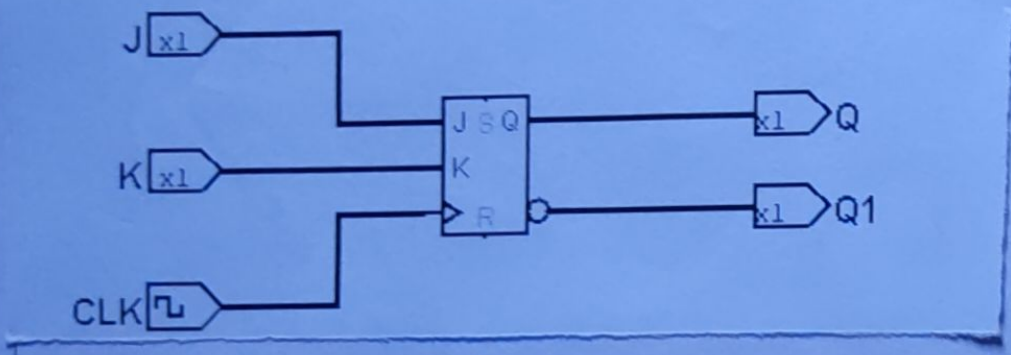
4) Verify that the circuit functions as a Demultiplexer

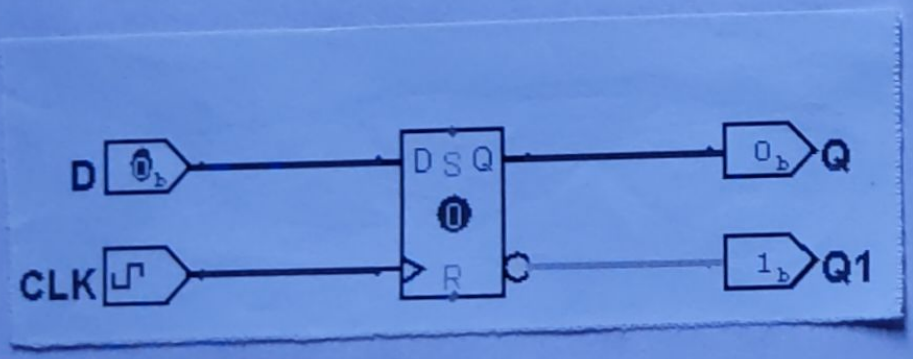
RESULT

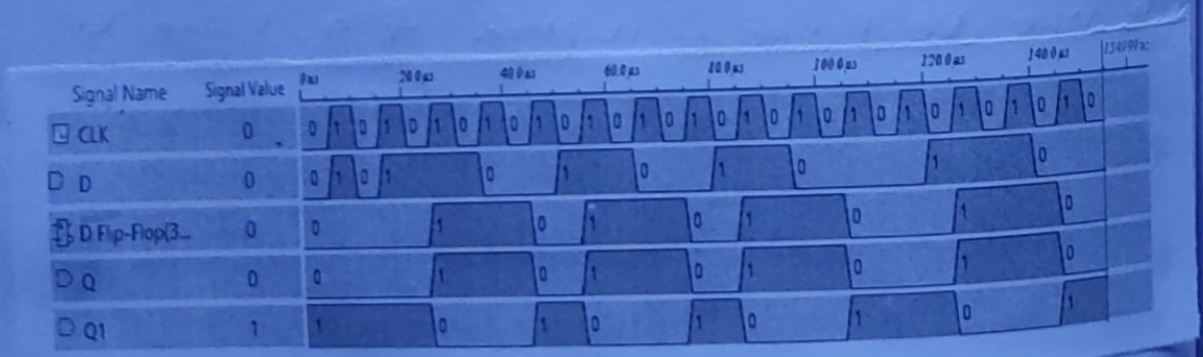
Studied & verified the working of a 4 line to 1 line multiplexer & a 1 line to 4 line demultiplexer using gates

PROGRAM CODE / CIRCUIT DIAGRAMS / SIMPLIFICATIONS









Experiment No. : 10
Date : 06-08-25

FAMILIARIZATION OF FLIPFLOPS

- **AIM**

To familiarize the working of J-K and D flipflops.

- **COMPONENTS REQUIRED**

- **PROCEDURE**

A flipflop is an electronic circuit with two stable states that can be used to store binary data. The stored data can be changed by applying varying inputs flip flops is probably known as the basic digital memory circuit. It has it's two states as logic 1 (SET) & logic 0 (RESET) states. A flipflop is a sequential circuit that consists of a single binary state of information or data flip flops are classified as tve edge triggered and -ve edge triggered flipflops

Flipflops & latches are fundamental building blocks of digital electronics systems used in computers, communications & many other types of systems.

D flip flop

it's a clocked flip flop with a single digital input D. Each time a D-flipflop is clocked, it's output follows the stated of D. The D flipfloo has 2 inputs, data & clock input which control the flipflop. When clock input is high, the data is transfered to the o/p of the flip flop. When the clock input is low, the o/p of the flipflop is held in it's previous state A D flipflop is created by modifying an SR flipflop. One major issue with SR flipflop is the rave-around condition, which is eliminated in the D flipflop due to the inverted inputs

JK flipflops

The JK flipflops is versatile & is widely used type of flip flop. The function of J-K flip flop is identical that of the S-R flip flop has no invalid state as does the S-R flip flop. When both J & K are 1, the i/p pulse is transmitted through one AND gate only; the one whose input is connected to the flopflop output that is presently equal to 1. Thus if Q=1, the output of the upper AND gate becons upon application of the clock pulse & the flip flop is cleared. if Q=1,

PERFORMANCE EVALUATION

No	Performance criteria	Excellent (5)	Good (3)	Satisfactory (2)	Poor (1)	Total
1	Code	Code is correct, efficient, well-structured, and fully meets design requirements	Code works with minor issues or inefficiencies and meets most design goals	Code has multiple errors or lacks clarity but shows basic functionality	Code is mostly incorrect, incomplete, or fails to meet design requirements	5
2	Output Obtained	Output is fully correct and matches expected results in all test case	Output is mostly correct with minor mismatches	Output is partially correct with several errors or missing cases	Output is incorrect, missing, or not generated	5
3	Timetaken	Output was obtained successfully within 30 minutes	Output was obtained successfully within 45 minutes	Output was obtained successfully within 60 minutes	Output was obtained successfully after 1 hour	5
4	Record	Experiment is recorded clearly, completely, and in correct format	Experiment is mostly complete with minor errors or omissions	Experiment is partially recorded with several details missing or unclear	Experiment is poorly documented or largely incomplete	5
					TOTAL MARKS	20

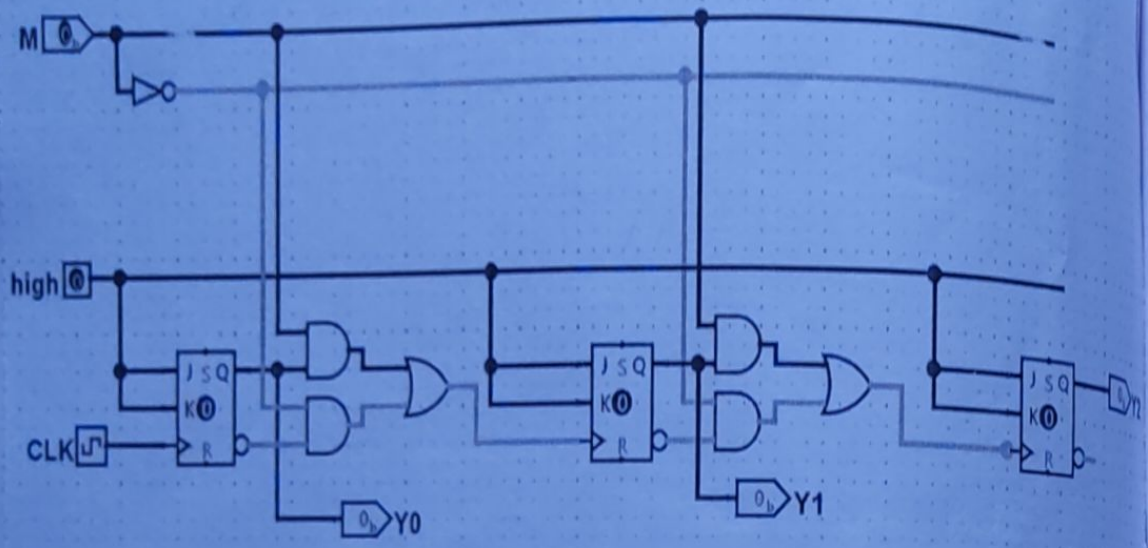
CRITERIA	MARKS AWARDED
Performance (20)	20
Viva (5)	5
Total (25)	25

the o/p of the lower of the lower AND gates becomes 1 & the flipflop is set. To avoid the undesirable operation, the clock pulse must have a time duration that is shorter than the propagation delay time of the flip flop

RESULT

The D flipflop & JK flipflop logic circuit were succesfully implemented using logism Evolution. The o/p of the simulation matched the expected resulted from the truth tables

PROGRAM CODE / CIRCUIT DIAGRAMS / SIMPLIFICATIONS



Experiment No. : 11
Date : 06-08-25

ASYNCHRONOUS 3-BIT UPDOWN COUNTER

➢ AIM

To Design and implement 3-bit Asynchronous updown counter with mod control.

➢ COMPONENTS REQUIRED

J-K flipflop

➢ PROCEDURE

1. Insevt 3 J-K flip flop
2. Make the JK inputs high
3. Connect a clock to the clock input of 1st flpp flop
4. Create an input pin fa mode change (having a normal & inverted output)
5. In between flpp flops invert 2 AND gates whose output Ps the input of an OR gate.
6. Connect the mode normal line & a output of flipflop to the 1st AND gate and the inverted mode line & an output to the 2nd AND gate.
7. OR the outputs of AND gates & input the value to clck of next flipflop

8. Repeat steps 5 to 7 for the next flipflop until LSM middle bit & MSB outpins are created

9. Mode 0 makes the circuit UP counter & MODE/ makes the circuit a down counter.

PERFORMANCE EVALUATION

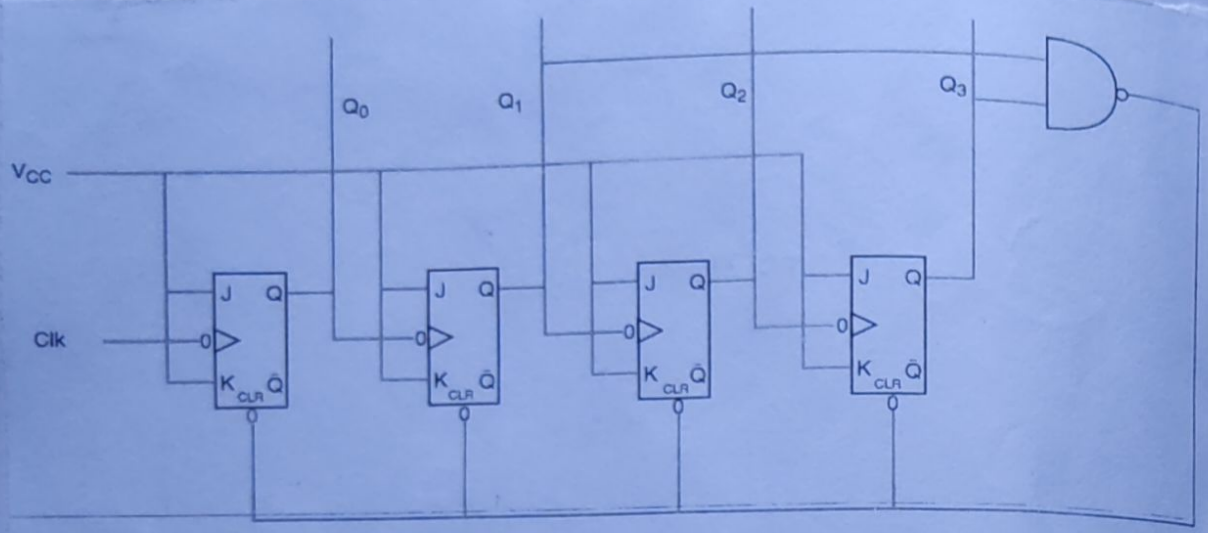
No	Performance criteria	Excellent (5)	Good (3)	Satisfactory (2)	Poor (1)	Total
1	Code	Code is correct, efficient, well-structured, and fully meets design requirements	Code works with minor issues or inefficiencies and meets most design goals	Code has multiple errors or lacks clarity but shows basic functionality	Code is mostly incorrect, incomplete, or fails to meet design requirements	5
2	Output Obtained	Output is fully correct and matches expected results in all test case	Output is mostly correct with minor mismatches	Output is partially correct with several errors or missing cases	Output is incorrect, missing, or not generated	5
3	Timetaken	Output was obtained successfully within 30 minutes	Output was obtained successfully within 45 minutes	Output was obtained successfully within 60 minutes	Output was obtained successfully after 1 hour	5
4	Record	Experiment is recorded clearly, completely, and in correct format	Experiment is mostly complete with minor errors or omissions	Experiment is partially recorded with several details missing or unclear	Experiment is poorly documented or largely incomplete	5
					TOTAL MARKS	20

CRITERIA	MARKS AWARDED
Performance (20)	20
Viva (5)	4
Total (25)	24

RESULT

The 3 bit asynchronous counters were designed & implemented

PROGRAM CODE / CIRCUIT DIAGRAMS / SIMPLIFICATIONS



Experiment No. : 12
Date : 10-09-25

ASYNCHRONOUS MOD-10 COUNTER

➢ AIM

Design and implement Asynchronous Mod-10 counter.

➢ COMPONENTS REQUIRED

➢ PROCEDURE

1. Test on the IC's using a digital IC tester & check the continuing of wires using a multiplexer or IC trainer.
2. Connect the circuit for a 4-bit ripple counter using 4 JK flip flops (eg: IC7476) Connect all PRESET Pins to +5v to disable preset.
3. Clear all flipp-flops initially by connecting the CLEAR pins to logic 0. After clearing, Connect them to logic 1
4. Apply clock pulses at the input & observe the binary counting sequence at the outputs (Q_0-Q_3).
5. Connect additional AND gates to detect the count 1010 (decimal 10) this o/p to reset all flipflops to zero, forming a mod-10 counter.

6. Verify that the counter from 0000(0) to (1001)(9) & resets back to 0000
7. Observe the o/p wave forms using a CRO, applying a suitable clock frequency to view the input ~~the~~ & output pulses

Theory:

An asynchronous counter (also called a ripple counter) is a type of counter in which the flipp-flop do not receieve the clock pulses simultaneously, The clock i/p is applied only to the first flipp-flop, & the output of each flipflop acts as the clock input for the next one, causing a propagation delay through the stages

A mod 10 counter is an asynchronous counter that counts from 0 to 9 & then reset back to 0. It can be designed using four JK-FF's connected in toggle mode (J=K=1) Since four flip-flops can count up to 16 state (2^4=16), external combinational logic is used to reset the counter when it reaches the 10^{th} state.

The TTL IC 74L590 is a parallel ~~exam~~ of a decode asynchronous counter. It produces a 4-bit binary output suitable for dining digital displays

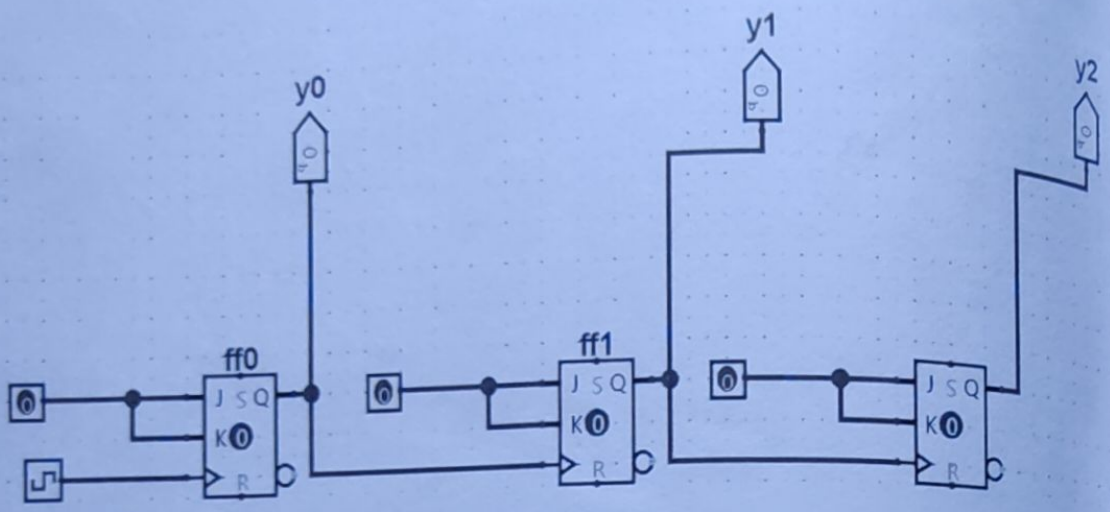
PERFORMANCE EVALUATION

No	Performance criteria	Excellent (5)	Good (3)	Satisfactory (2)	Poor (1)	Total
1	**Code**	Code is correct, efficient, well-structured, and fully meets design requirements	Code works with minor issues or inefficiencies and meets most design goals	Code has multiple errors or lacks clarity but shows basic functionality	Code is mostly incorrect, incomplete, or fails to meet design requirements	5
2	**Output Obtained**	Output is fully correct and matches expected results in all test case	Output is mostly correct with minor mismatches	Output is partially correct with several errors or missing cases	Output is incorrect, missing, or not generated	5
3	**Timetaken**	Output was obtained successfully within 30 minutes	Output was obtained successfully within 45 minutes	Output was obtained successfully within 60 minutes	Output was obtained successfully after 1 hour	5
4	**Record**	Experiment is recorded clearly, completely, and in correct format	Experiment is mostly complete with minor errors or omissions	Experiment is partially recorded with several details missing or unclear	Experiment is poorly documented or largely incomplete	5
					TOTAL MARKS	26

CRITERIA	MARKS AWARDED
Performance (20)	20
Viva (5)	5
Total (25)	25

RESULT

A mod-N counter (decade counter) was implemented & it's working studied.


y0
y1
y2
ff0
ff1
J S Q
K
R

Experiment No. : 13
Date : 10-09-25

SYNCHRONOUS 3-BIT UP COUNTER

➢ AIM

To design and implement Synchronous 3-bit up counter.

➢ COMPONENTS REQUIRED

J-K flip flop

➢ PROCEDURE

1. Insert 3 J-K flip flops
2. Make the JK inputs high
3. Connect a clock to the clock input of all 3 flip flops
4. Connects the Q output of 1st flip flop to the JK inputs of 2nd flip flop.
5. Connect the Q output of 1st & 2nd flip flops to the JK inputs of 3rd flip flop through an AND gate.
6. The Q outputs of 1st, 2nd & 3rd flip flops are connected to o/p pins for LSB, middle & MSB respectively
7. Enable the high of JK inputs & activate clock
8. Ensure count is ascending.

Theory

A 3 bit synchronous counter is a digital circuit that uses 3 flip flops to count from 0000 to 111 in binary. In a synchronous counter, all flip-flops receive the same common clock signal, allowing them to change states at the same time & eliminating the propagation delay found in asynchronous (ripple) counters.

PERFORMANCE EVALUATION

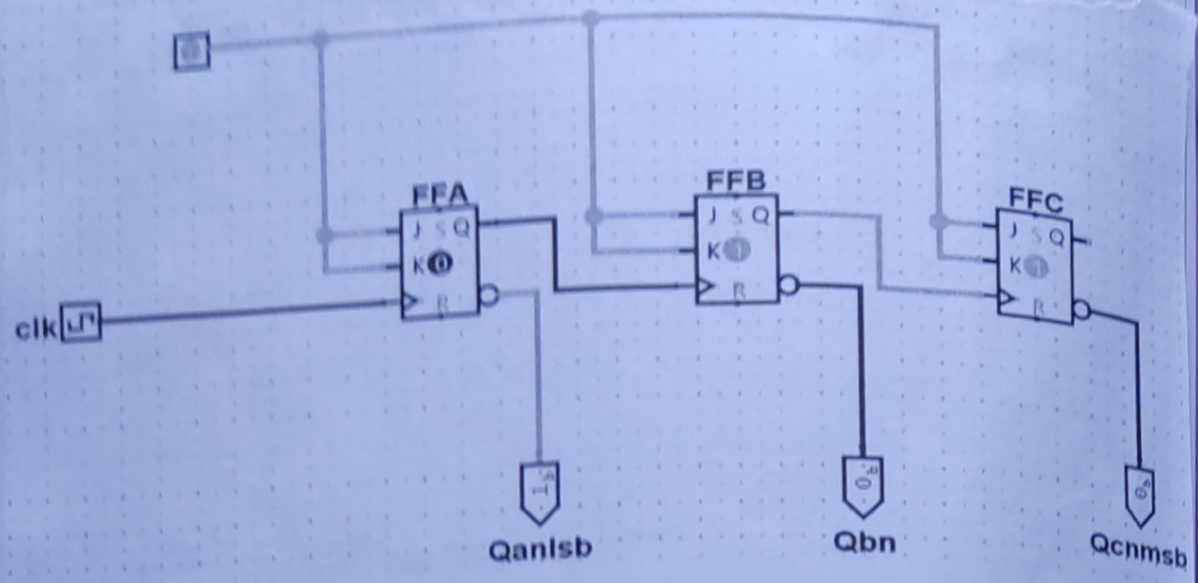
No	Performance criteria	Excellent (5)	Good (3)	Satisfactory (2)	Poor (1)	Total
1	**Code**	Code is correct, efficient, well-structured, and fully meets design requirements	Code works with minor issues or inefficiencies and meets most design goals	Code has multiple errors or lacks clarity but shows basic functionality	Code is mostly incorrect, incomplete, or fails to meet design requirements	5
2	**Output Obtained**	Output is fully correct and matches expected results in all test case	Output is mostly correct with minor mismatches	Output is partially correct with several errors or missing cases	Output is incorrect, missing, or not generated	5
3	**Timetaken**	Output was obtained successfully within 30 minutes	Output was obtained successfully within 45 minutes	Output was obtained successfully within 60 minutes	Output was obtained successfully after 1 hour	5
4	**Record**	Experiment is recorded clearly, completely, and in correct format	Experiment is mostly complete with minor errors or omissions	Experiment is partially recorded with several details missing or unclear	Experiment is poorly documented or largely incomplete	5
					TOTAL MARKS	20

CRITERIA	MARKS AWARDED
Performance (20)	20
Viva (5)	5
Total (25)	25

RESULT

The 3-bit Synchronous UP counter were designed & implemented.

PROGRAM CODE / CIRCUIT DIAGRAMS / SIMPLIFICATIONS



Experiment No. : 14
Date : 17-09-25

SYNCHRONOUS 3-BIT DOWN COUNTER

➢ AIM

To design and implement Synchronous 3-bit down counter.

➢ COMPONENTS REQUIRED

JK flip flop

➢ PROCEDURE

1) Insert 3 JK flip flops
2) Make the JK inputs high
3) Connect a clock to the clock input of all 3 flip flops
4) Connect a $\bar{Q}$n output of 1st flip flop to the JK inputs to 2nd flip flop
5) Connect the $\bar{Q}$n of 1st & 2nd flip flops to the JK inputs of 3rd flip flops through an AND gate.
6) The Q output of 1st, 2nd & 3rd flip flops are connected to o/p pins for LSB, middle & MSB respectively
7) Enable the high of JK inputs & activate clock.
8) Ensure count is descending

Theory

A 3 bit synchronous counter is a digital circuit that uses three flipflops to count from 000 to 111 in binary. In a synchronous counter, all flip flops receive the same common clock signal, allowing them to change the states at the same time & eliminating the propagation delay found in asynchronous (ripple) counters.

Theory

A 3 bit synchronous counter is a digital circuit that uses three flipflops to count from 000 to 111 in binary. In a synchronous counter, all flip flops receive the same common clock signal, allowing them to change the states at the same time & eliminating the propagation delay found in asynchronous (ripple) counters.

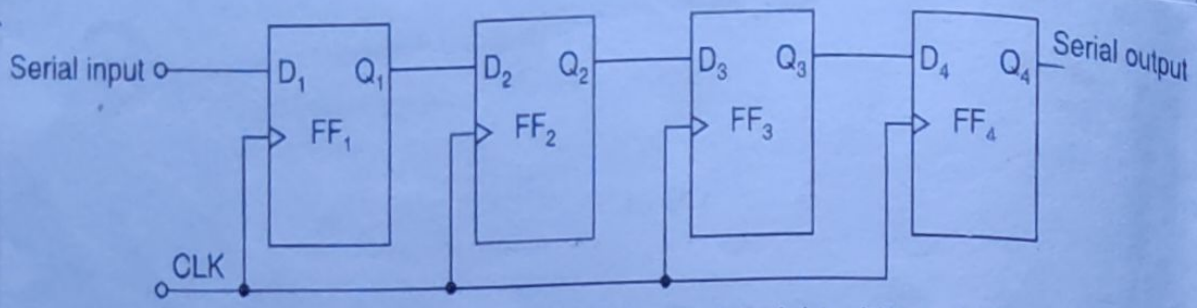
PERFORMANCE EVALUATION

No	Performance criteria	Excellent (5)	Good (3)	Satisfactory (2)	Poor (1)	Total
1	Code	Code is correct, efficient, well-structured, and fully meets design requirements	Code works with minor issues or inefficiencies and meets most design goals	Code has multiple errors or lacks clarity but shows basic functionality	Code is mostly incorrect, incomplete, or fails to meet design requirements	5
2	Output Obtained	Output is fully correct and matches expected results in all test case	Output is mostly correct with minor mismatches	Output is partially correct with several errors or missing cases	Output is incorrect, missing, or not generated	5
3	Timetaken	Output was obtained successfully within 30 minutes	Output was obtained successfully within 45 minutes	Output was obtained successfully within 60 minutes	Output was obtained successfully after 1 hour	5
4	Record	Experiment is recorded clearly, completely, and in correct format	Experiment is mostly complete with minor errors or omissions	Experiment is partially recorded with several details missing or unclear	Experiment is poorly documented or largely incomplete	5
					TOTAL MARKS	20

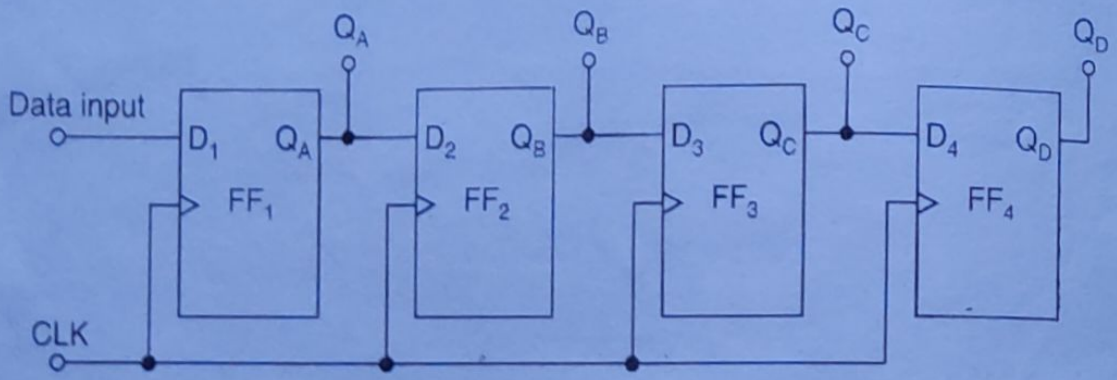
CRITERIA	MARKS AWARDED
Performance (20)	20
Viva (5)	5
Total (25)	25

RESULT

The 3 bit synchronous counters were designed & implemented.



4-bit serial-in, serial-out, shift-right, shift register.



Experiment No. : 15
Date : 17-09-25

SHIFT REGISTERS

- **AIM**

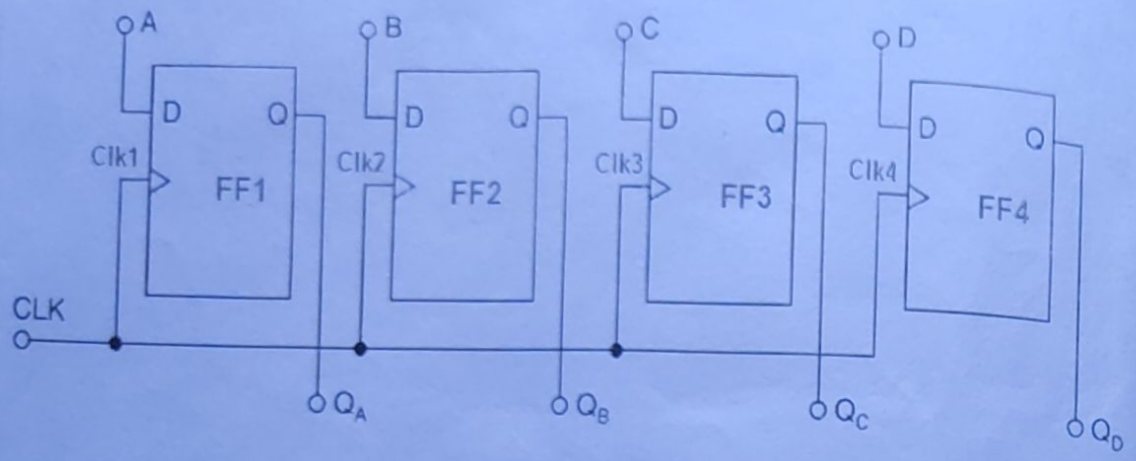
To implement shift registers using D-flip flops [SISO, SIPO, PISO, PIPO].

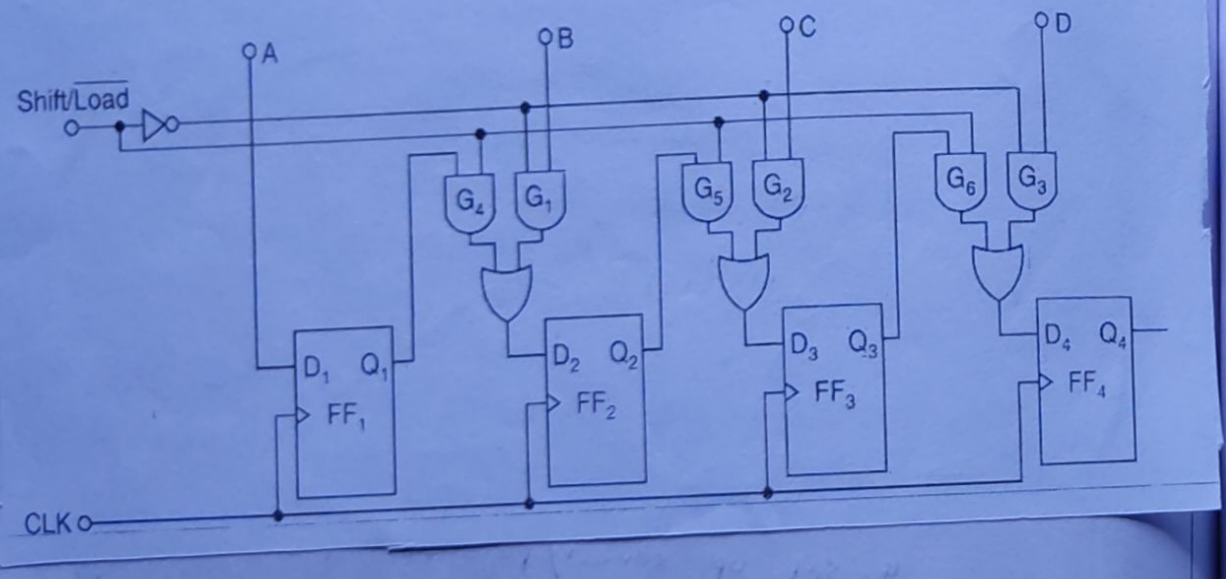
- **COMPONENTS REQUIRED**

ICs 7474, 7408, 7404, 7432 & Trainer kit

- **PROCEDURE**

1. Test all the components & IC packages using multimeter & digital IC tester
2. Set up serial input shift register using JKFF. Clear all the flip flops using CLEAR pins feed 1011 to the serial input starting from LSB using PRESET & CLEAR pins. A low input to CLEAR & PRESET will make Q output 0 & 1 respectively
3. Apply 1Hz clock & observe the bits are shifting right. Repeat the above step using DFF
4. Set up serial/parallel input shift register. Enter a data serially keeping load/shift = 1. Apply clock pulses & note the outputs after each clock pulse
5. Enter a data parallel with load/shift=0. Output can be obtained in parallel or in serial form. Repeat for various input combinations.


A
B
C
D
D
Q
Clk1
FF1
Clk2
FF2
Clk3
FF3
Clk4
FF4
CLK
Q_A
Q_B
Q_C
Q_D


A
B
C
D
Shift/Load
G_4
G_1
G_5
G_2
G_6
G_3
D_1 Q_1
FF_1
D_2 Q_2
FF_2
D_3 Q_3
FF_3
D_4 Q_4
FF_4
CLK

SERIAL-IN, SERIAL-OUT, SHIFT REGISTER

Serial input shift register as it's name suggests, serial input shift register allow the data to enter serially. Output can be taken serial or in parallel. Serial input is fed through input, A & serial output is taken from Q_4

SERIAL-IN, PARALLEL-OUT SHIFT REGISTER

In this type of register, the data bits are entered serially, but the data stored in the register is shifted out in parallel form. Once the data bits are stored, each bit appears on it's respective output line and all bits are available simultaneously

PARALLEL-IN, PARALLEL-OUT SHIFT REGISTER

In Parallel-In Parallel-out shift register the data is entered into the register in parallel form & also the data is taken out of the register in parallel form. Immediately following simultaneous entry of all data bits the bits appear on the parallel outputs. Figure shows a 4-bit PiPo shift register using DFF's Data is applied to the D input teminals of FF's when a clock pulse is applied at the +ve going edge of that pulse, the D inputs are shifted into the Q outputs of the FFs

PERFORMANCE EVALUATION

No	Performance criteria	Excellent (5)	Good (3)	Satisfactory (2)	Poor (1)	Total
1	Code	Code is correct, efficient, well-structured, and fully meets design requirements	Code works with minor issues or inefficiencies and meets most design goals	Code has multiple errors or lacks clarity but shows basic functionality	Code is mostly incorrect, incomplete, or fails to meet design requirements	5
2	Output Obtained	Output is fully correct and matches expected results in all test case	Output is mostly correct with minor mismatches	Output is partially correct with several errors or missing cases	Output is incorrect, missing, or not generated	5
3	Timetaken	Output was obtained successfully within 30 minutes	Output was obtained successfully within 45 minutes	Output was obtained successfully within 60 minutes	Output was obtained successfully after 1 hour	5
4	Record	Experiment is recorded clearly, completely, and in correct format	Experiment is mostly complete with minor errors or omissions	Experiment is partially recorded with several details missing or unclear	Experiment is poorly documented or largely incomplete	5
					TOTAL MARKS	20

CRITERIA	MARKS AWARDED
Performance (20)	20
Viva (5)	5
Total (25)	25

PARALLEL-IN SERIAL-OUT SHIFT REGISTER

A parallel-in, serial out shift register is a sequential logic circuit designed to convert parallel data into a serial stream. This is typically achieved using D-type flip flops arranged in cascaded manner. Data is loaded into the register in a single clock pulse, with each flip-flop receiving its corresponding bit from a seperate line. Once the data is loaded, a control signal switches the register to shift mode. In this mode, subsequent clock pulses cause the data bits to shift from one flip flop to the next with the most significant bit appearing first at the serial output. This process allows multiple bits of data to be transmitted over a single line. making it a fundamental component for communication & data bus system

RESULT

SISO, PISO, PIPO registers using D-flipflop are verified & set up

PROGRAM CODE / CIRCUIT DIAGRAMS / SIMPLIFICATIONS

Experiment No. : 16
Date : 08-10-25

BEHAVIORAL MODEL FOR FLIPFLOP

- **AIM**

To design and synthesize the behavioral model for a D-flip flop.

- **COMPONENTS REQUIRED**

D flip flop

EDA Playground

GTk wave

- **PROCEDURE**

1. Provide 4 bit inputs A & B
2. Simulate for all combinations
3. Observe comparision outputs

PERFORMANCE EVALUATION

No	Performance criteria	Excellent (5)	Good (3)	Satisfactory (2)	Poor (1)	Total
1	Code	Code is correct, efficient, well-structured, and fully meets design requirements	Code works with minor issues or inefficiencies and meets most design goals	Code has multiple errors or lacks clarity but shows basic functionality	Code is mostly incorrect, incomplete, or fails to meet design requirements	5
2	Output Obtained	Output is fully correct and matches expected results in all test case	Output is mostly correct with minor mismatches	Output is partially correct with several errors or missing cases	Output is incorrect, missing, or not generated	5
3	Timetaken	Output was obtained successfully within 30 minutes	Output was obtained successfully within 45 minutes	Output was obtained successfully within 60 minutes	Output was obtained successfully after 1 hour	5
4	Record	Experiment is recorded clearly, completely, and in correct format	Experiment is mostly complete with minor errors or omissions	Experiment is partially recorded with several details missing or unclear	Experiment is poorly documented or largely incomplete	5
					TOTAL MARKS	20

CRITERIA	MARKS AWARDED
Performance (20)	20
Viva (5)	5
Total (25)	25

RESULT

The 4bit Comparator was successfully implemented in verilog HDL

```
// File: sync_counter.v
module sync_counter (
  input wire clk,      // Clock input
  input wire rst,      // Synchronous reset (active high)
  output reg [3:0] count // 4-bit counter output
);

  always @(posedge clk) begin
    if (rst)
      count <= 4'b0000;
    else
      count <= count + 1;
  end

endmodule
// File: tb_sync_counter.v
`timescale 1ns/1ps

module tb_sync_counter;
  reg clk, rst;
  wire [3:0] count;

  // Instantiate the counter
  sync_counter uut (
    .clk(clk),
    .rst(rst),
    .count(count)
  );

  // Clock generation: 10ns period
  initial begin
```

Experiment No. : 17
Date : 08-10-25

BEHAVIORAL MODEL FOR SYNCHRONOUS COUNTER

➢ AIM

To design and synthesize the behavioral model for a Synchronous Counter.

➢ COMPONENTS REQUIRED

1. EDA Playground
2. GTK wave

➢ PROCEDURE

1. Apply clock & reset signals
2. Observe counter output after each clock pulse
3. Verify the counting sequence

0000 → 0001 → 0010 → 1111

```
    clk = 0;
    forever #5 clk = ~clk;
  end

  // VCD dump
  initial begin
    $dumpfile("sync_counter.vcd");
    $dumpvars(0, tb_sync_counter);
  end

  // Stimulus
  initial begin
    $display("Time\tclk rst | count");
    $monitor("%0dns\t%b %b | %b", $time, clk, rst, count);

    rst = 1; #10;  // Apply reset
    rst = 0; #100; // Count up for 10 clock cycles
    rst = 1; #10;  // Apply reset again
    rst = 0; #50;  // Count up for 5 more clock cycles
    $finish;
  end

endmodule
```

PERFORMANCE EVALUATION

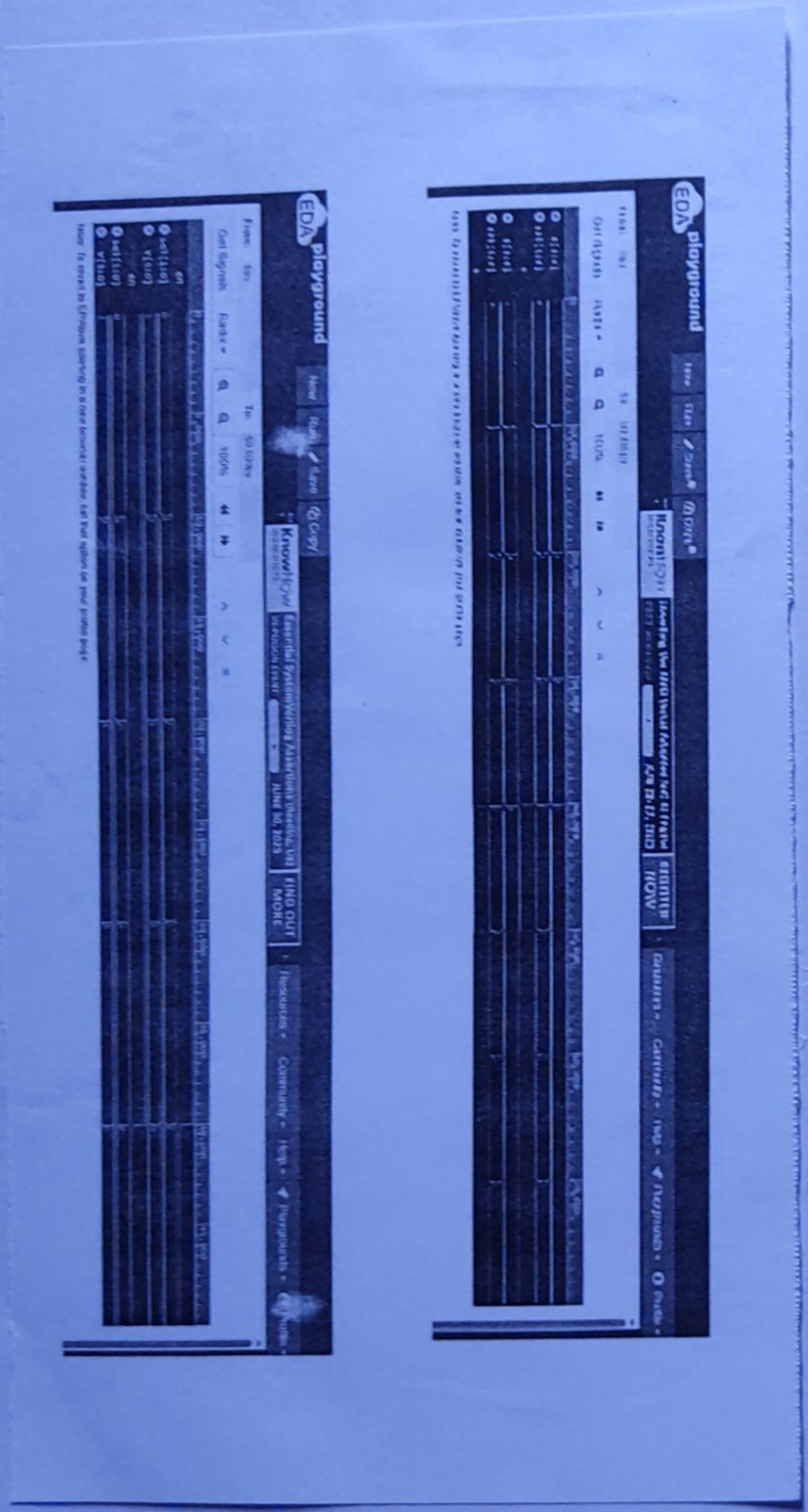
No	Performance criteria	Excellent (5)	Good (3)	Satisfactory (2)	Poor (1)	Total
1	**Code**	Code is correct, efficient, well-structured, and fully meets design requirements	Code works with minor issues or inefficiencies and meets most design goals	Code has multiple errors or lacks clarity but shows basic functionality	Code is mostly incorrect, incomplete, or fails to meet design requirements	5
2	**Output Obtained**	Output is fully correct and matches expected results in all test case	Output is mostly correct with minor mismatches	Output is partially correct with several errors or missing cases	Output is incorrect, missing, or not generated	5
3	**Timetaken**	Output was obtained successfully within 30 minutes	Output was obtained successfully within 45 minutes	Output was obtained successfully within 60 minutes	Output was obtained successfully after 1 hour	5
4	**Record**	Experiment is recorded clearly, completely, and in correct format	Experiment is mostly complete with minor errors or omissions	Experiment is partially recorded with several details missing or unclear	Experiment is poorly documented or largely incomplete	5
					TOTAL MARKS	20

CRITERIA	MARKS AWARDED
Performance (20)	20
Viva (5)	5
Total (25)	25

RESULT

4 bit synchronous counter was succesfully implimented & simulated using verilog HDL

PROGRAM CODE / CIRCUIT DIAGRAMS / SIMPLIFICATIONS



Experiment No. : 18
Date : 15-10-25

MUX AND DECODER USING CONTINUOUS ASSIGNMENT

- **AIM**

To model 4:1 Mux and 2:4 Decoder using continuous assignment with logical and conditional operators.

- **COMPONENTS REQUIRED**

  - EDA Playground
  - GTR wave

- **PROCEDURE**

For Mux

1. Declare input, select & output signal
2. Use a conditional (ternary) operator for selection.
3. Assign output using assign statement

For decoder

1. Declare input & output signals
2. Use logic expressions to generate each output
3. Assign each output using assign

PERFORMANCE EVALUATION

No	Performance criteria	Excellent (5)	Good (3)	Satisfactory (2)	Poor (1)	Total
1	**Code**	Code is correct, efficient, well-structured, and fully meets design requirements	Code works with minor issues or inefficiencies and meets most design goals	Code has multiple errors or lacks clarity but shows basic functionality	Code is mostly incorrect, incomplete, or fails to meet design requirements	5
2	**Output Obtained**	Output is fully correct and matches expected results in all test case	Output is mostly correct with minor mismatches	Output is partially correct with several errors or missing cases	Output is incorrect, missing, or not generated	5
3	**Timetaken**	Output was obtained successfully within 30 minutes	Output was obtained successfully within 45 minutes	Output was obtained successfully within 60 minutes	Output was obtained successfully after 1 hour	5
4	**Record**	Experiment is recorded clearly, completely, and in correct format	Experiment is mostly complete with minor errors or omissions	Experiment is partially recorded with several details missing or unclear	Experiment is poorly documented or largely incomplete	5
					TOTAL MARKS	20

CRITERIA	MARKS AWARDED
Performance (20)	20
Viva (5)	5
Total (25)	25

RESULT

Mode 4:MUX & 2:4 decoder using continous assignments with logic & conditional operators has been successfully executed.

IB Experiment Completed & Verified

Result