



Kammavari Sangham (R)-1952

K.S.GROUP OF INSTITUTIONS

K.S. INSTITUTE OF TECHNOLOGY

(Approved by AICTE & Affiliated to VTU)

KANAKAPURA ROAD, BANGALORE - 560 062

A P J ABDUL KALAM

"Tell me and I'll forget;
Show me and I may
remember; Involve me
and I'll understand"

GRAHAM BELL

MARIE CURIE

SRINIVASA
RAMANUJAN

J R D TATA

THOMAS ALVA EDISON

DENNIS RITCHIE

PRACTICAL RECORD

NAME : RAMYA.M.S

SEM/ BRANCH: V ECE 'B'

SUBJECT & CODE: HDL LAB

USN :

1	K	S	1	6	5	0	2	1
---	---	---	---	---	---	---	---	---

Vision:

“To strive continuously to impart quality technical education with ethical values, employable skills and research to achieve excellence”

Mission:

- ◆ **To attract and retain highly qualified, experienced and committed Faculty.**
- ◆ **To create relevant infrastructure.**
- ◆ **Network with industry and premier institutions to encourage emergence of new ideas by providing Research and Development Facilities to achieve excellence.**
- ◆ **To inculcate the professional and ethical values among students with employable skills and Knowledge acquired in transforming society.**



KSIT
K S INSTITUTE OF TECHNOLOGY

Kammavari Sangham (R)-1952

K.S.GROUP OF INSTITUTIONS

K.S. INSTITUTE OF TECHNOLOGY

(Approved by AICTE & Affiliated to VTU)

KANAKAPURA ROAD, BANGALORE - 560 062

Laboratory Certificate

This is to certify that Mr./Ms. RAMYA.M.S
has satisfactorily completed the course of experiments in
Verilog HDL laboratory, Code 15ECL58
prescribed by Visvesvaraya Technological University, Belgaum for
the V Semester B.E. ECE Branch
in this College during the academic year 20..18... - 20...19..

Name of the Candidate : RAMYA.M.S

USN : 1K816 ECL071 Subject (with code) Verilog HDL Lab (15ECL58)

Internal assessment marks awarded :

<u>10</u>	+	<u>6</u>	=	<u>16</u>
<u>15</u>		<u>10</u>		<u>25</u>
<u>R</u>		<u>T</u>		

[Signature]

Signature of Staff Incharge

Date:

29/11/18

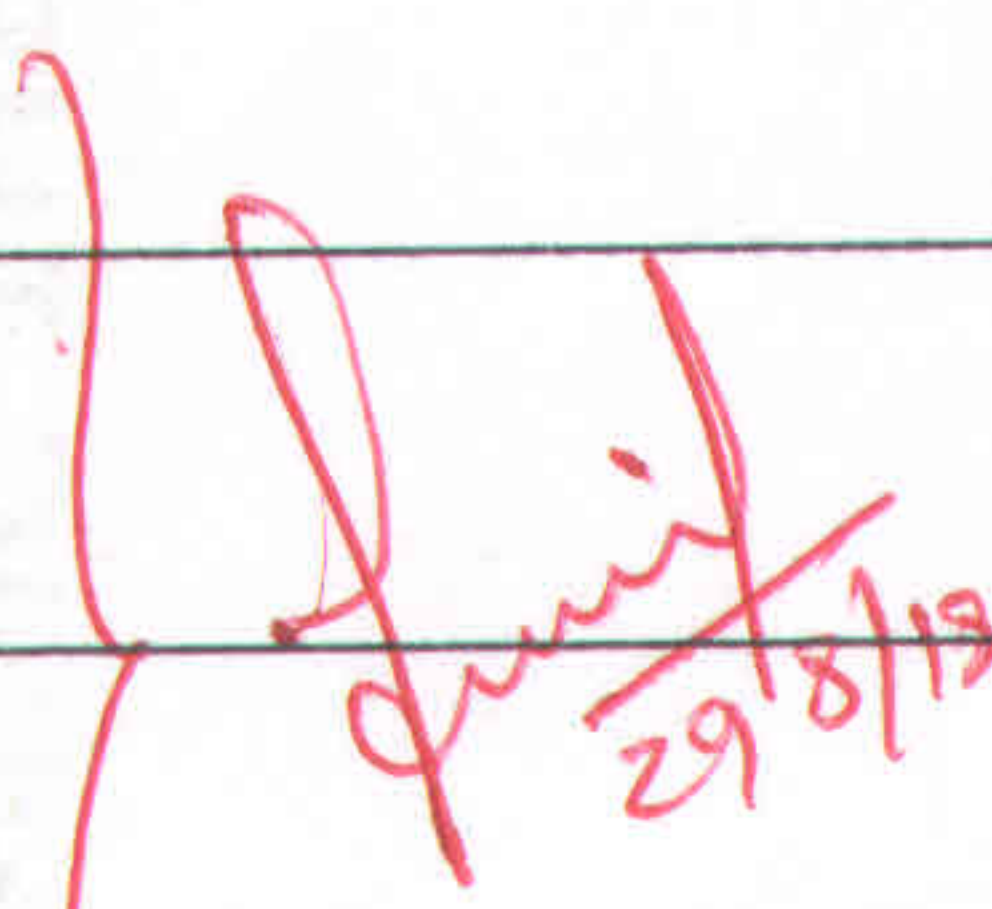
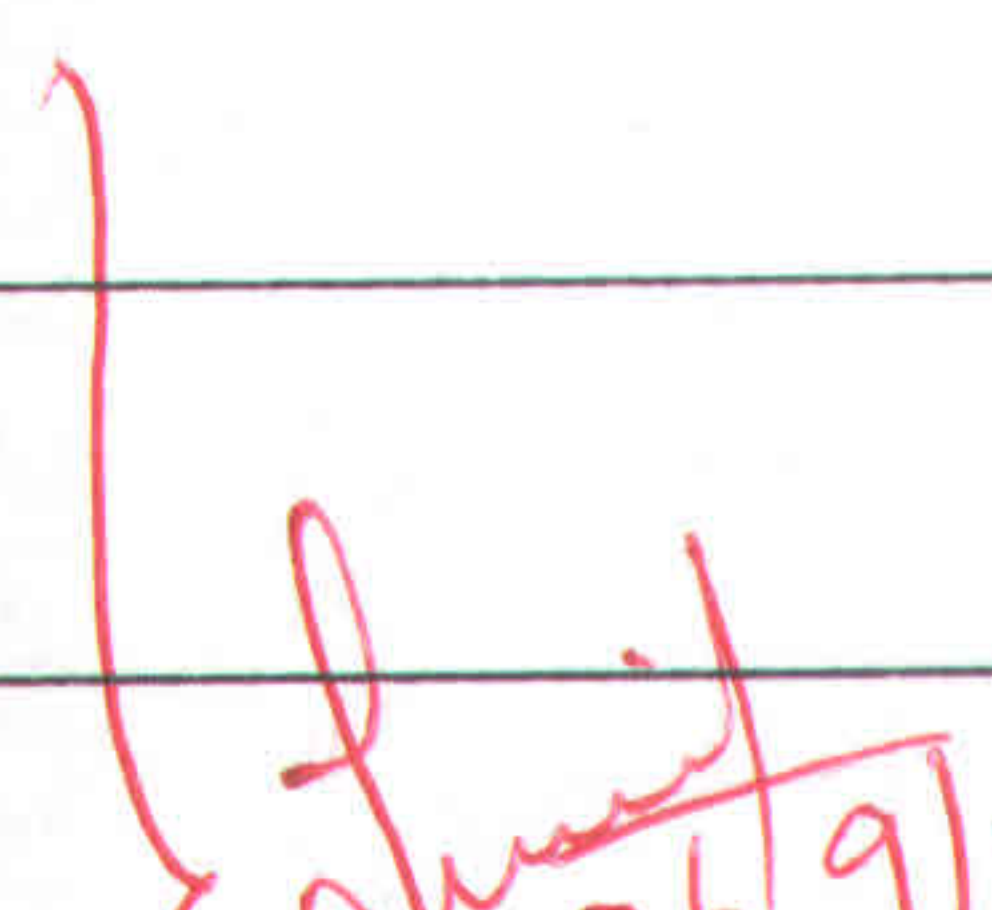
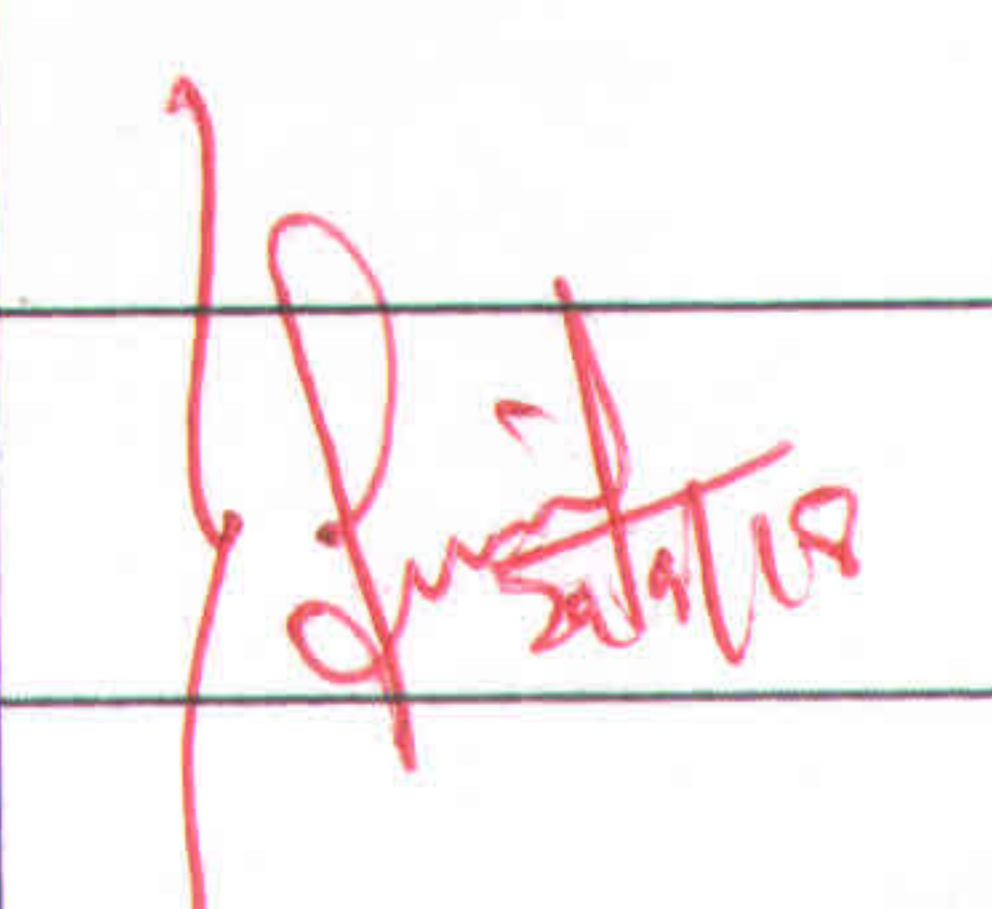
[Signature]

Signature of Head of
Department

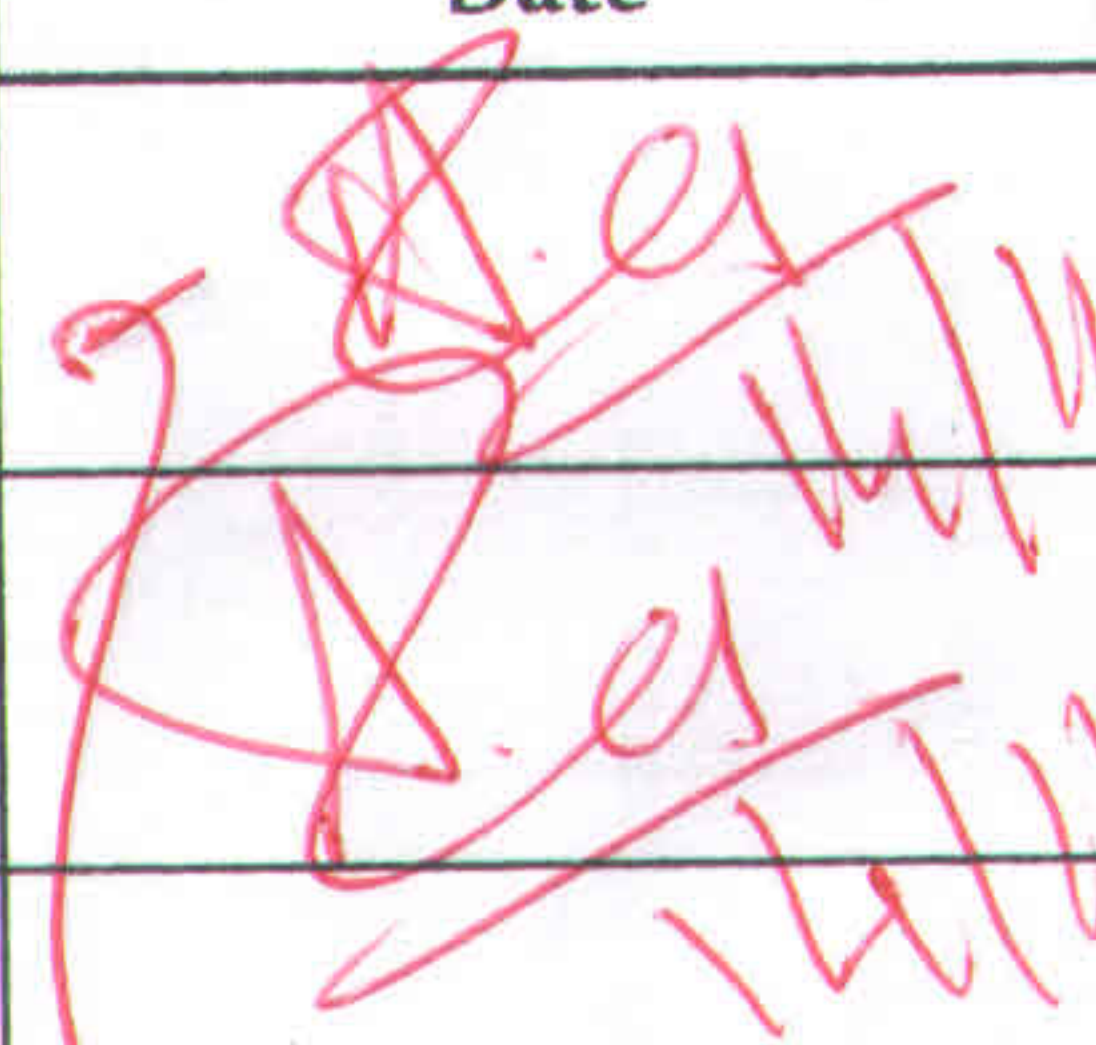
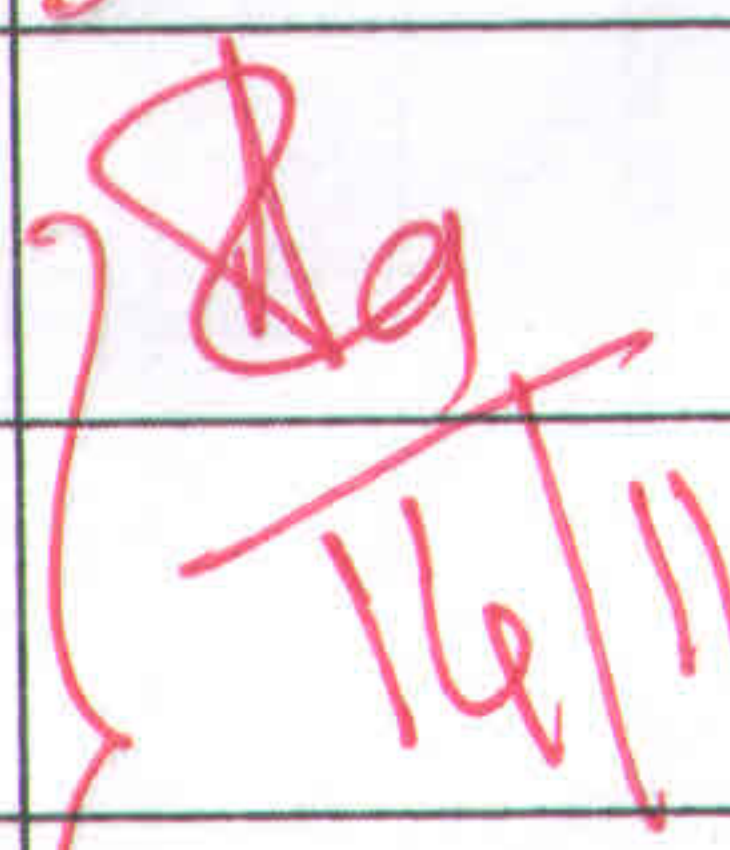
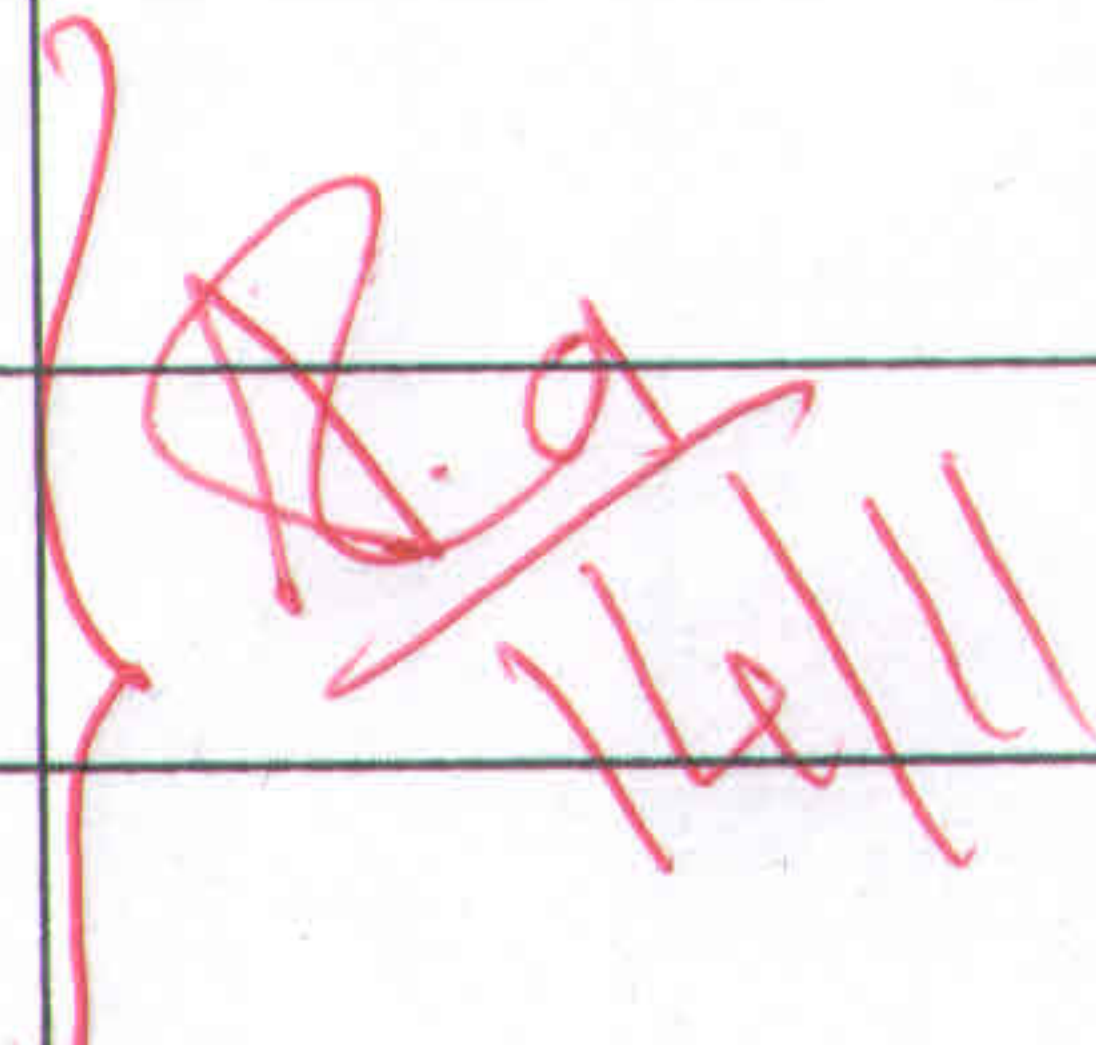
INSTRUCTIONS FOR MAINTAINING THE PRACTICAL RECORD

1. Record should be written neatly in ink on the right side page only, left side pages being reserved for diagrams and graphs in pencil.
2. Record should contain
 - Number and name of the Experiment
 - The date
 - Principle
 - Procedure
 - Observation calculation (to be entered on the left hand side pages in neat tabular forms wherever applicable)
 - Results

CONTENTS AND EVALUATION

Sl.No.	Date of Conducting Experiment	Date of Submission	Page No.	Title of the Experiment	Assessment Marks			Signature of Staff with Date
					Experiment 10 Marks	Record 10 Marks	Total	
1.	31/8/18	29/8/18	1.	INTRODUCTION.	10	10	20	 29/8/18
2.	18/8/18	29/8/18	3	BASIC GATES	10	10	20	
3.	18/8/18	29/8/18	5	2:4 DECODER	10	10	20	
4.	18/8/18	29/8/18	7	BINARY TO GRAY	10	10	20	
5.	18/8/18	29/8/18	9.	8:1 MUX .	10	10	20	
6.	29/8/18	5/9/18	11	8:3 priority encoder	10	10	20	 26/9/18
7.	29/8/18	5/9/18	13	1:8 demux	10	10	20	
8.	29/8/18	5/9/18	15	2 bit comparator .	10	10	20	
9.	29/8/18	19/9/18	17	Full adder .	10	10	20	
10	19/9/18	20/9/18	21	4 bit ALU	10	10	20	
11.	19/9/18	29/9/18	23	SR Flip flop	10	10	20	 26/9/18
12.	19/9/18	29/9/18	25	JK flip flop	10	10	20	
13	19/9/18	29/9/18	27	T- Flip flop	10	10	20	

CONTENTS AND EVALUATION

Sl.No.	Date of Conducting Experiment	Date of Submission	Page No.	Title of the Experiment	Assessment Marks			Signature of Staff with Date
					Experiment 10 Marks	Record 10 Marks	Total	
14	19/9/18	29/9/18	29	D - flip flop	10	10	20	 14/11
15	29/9/18	3/10/18	31	4 bit binary Synchronous Counter	10	10	20	
16	29/9/18	3/10/18	33	4 bit binary asynchronous counter	10	10	20	
17	29/9/18	3/10/18	35	4 bit BCD Synchronous counter	10	10	20	
18	29/9/18	3/10/18	37	4 bit BCD asynchronous counter	10	10	20	 14/11
19	31/10/18	3/11/18	39	STEPPER MOTOR	10	10	20	
20	31/10/18	3/11/18	40	DC MOTOR	10	10	20	
21	31/10/18	3/11/18	41	ADC.	10	10	20	 14/11
22	3/11/18	14/11/18	43	DAC (Square, triangular, sine, sawtooths)	10	10	20	
23	3/11/18	14/11/18	44	LCD	10	10	20	
24	3/11/18	14/11/18	53	HEX KEYPAD	10	10	20	

Tool used - Xilinx 13.1 version.

Software procedure - ISE design tool

Family - Spartan 6

device - XC653LX4

↳ Xilinx Computer

Package - TQ644

↳ Thin Quad

(144 programmable I/O pins are available for design)

Speed - (-3)

Top level source - HDL

Synthesis tool - XST

Simulator - ISIM

preferred language - Verilog

⇒ Neat

⇒ finish

Project → New Source → Verilog Module → file name [Module name]
⇒ Neat

input ☐

Input ☐

output ☐

⇒ Neat ⇒ finish

→ Code

→ for O/P i) Implementation → Synthesize XST (check Syntax)

(ii) Simulation → ISIM → Simulate → behavioural Model

→ Right click → force constants → give inputs → apply → OK
waveform obtained } → fit to the window.

Hand wave procedure
open the code

Synthesize XST → user constraint → I/O planning port
↓

Synthesize → Yes → Cancel
Select pins according to inputs and outputs } save
→ click on other side to assign } → minimize
(Bank0)

My Computer (right click)

↳ Manage → device manager → V8F6 SPATR N 6
↓
[COM-]

Synthesis → implement → general programming file

Start → all programs → V8F6 Spatn 6 → V8F6 → Select
↓
Configure ← Load file ← file

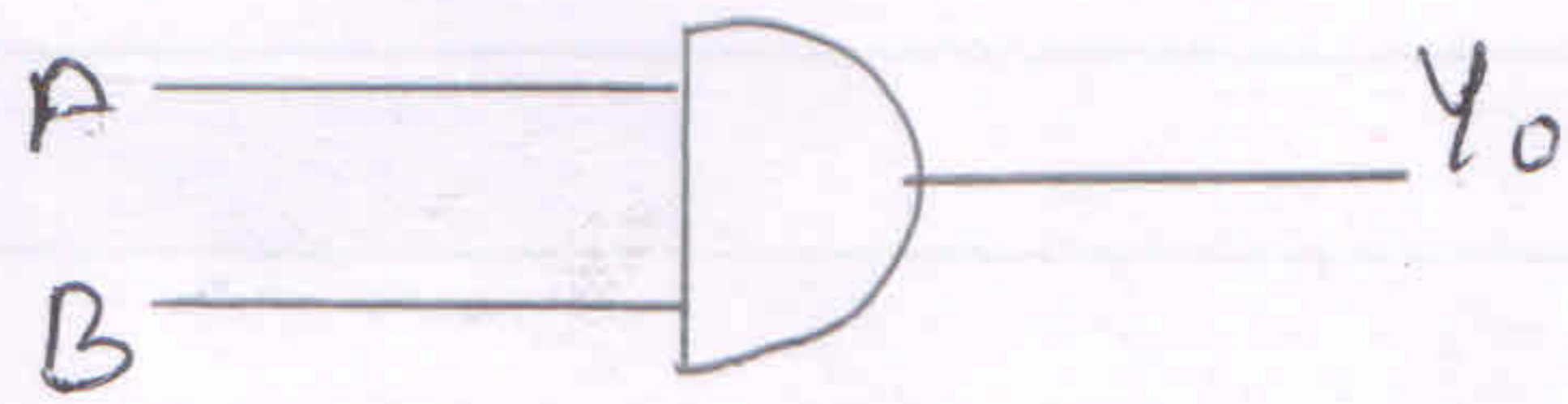
→ observe pin configuration.

R-10. Record
O-10 Observation
T-20. Total

DESIGN:

Logic diagram:

i) AND gate:

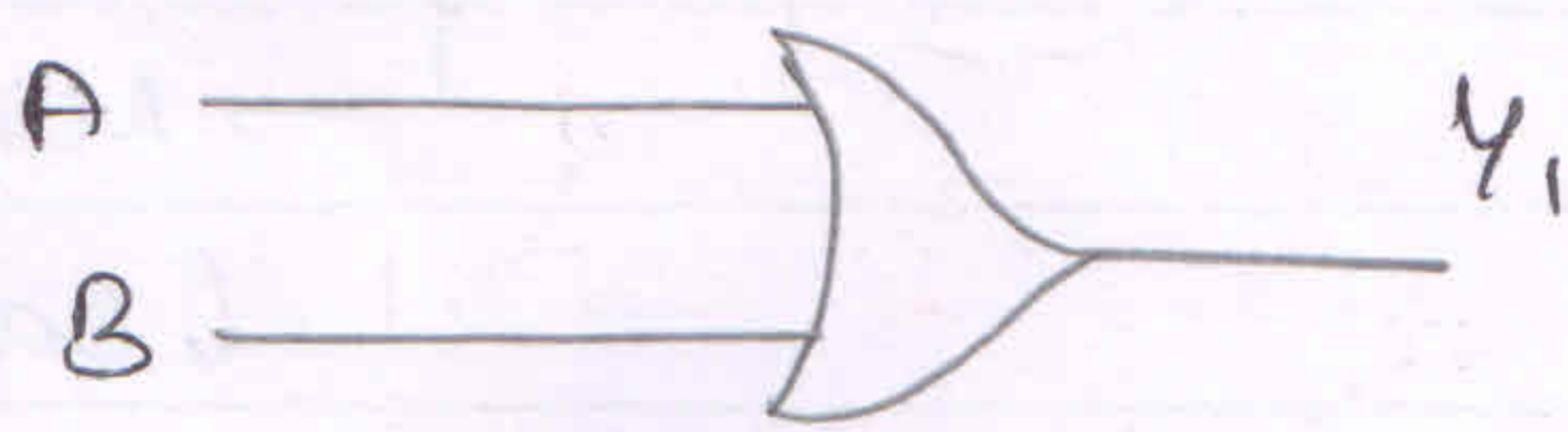


$$Y_0 = A \cdot B$$

Truth table:

I/P		O/P
A	B	$Y_0 = A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

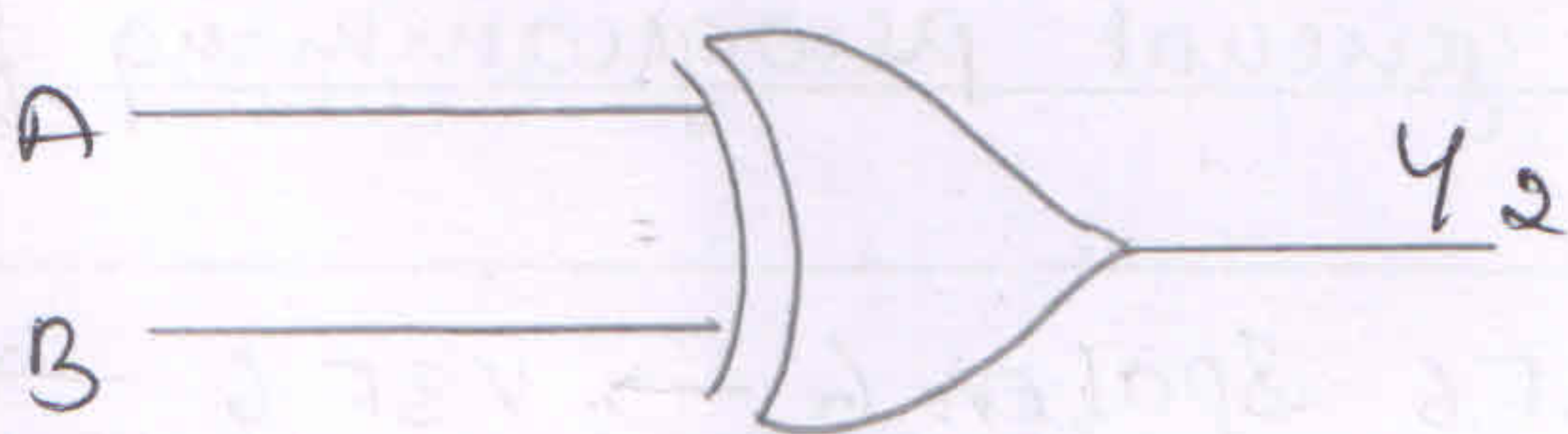
ii) OR gate:



$$Y_1 = A + B$$

I/P		O/P
A	B	$Y_1 = A + B$
0	0	0
0	1	1
1	0	1
1	1	1

iii) EXOR gate:

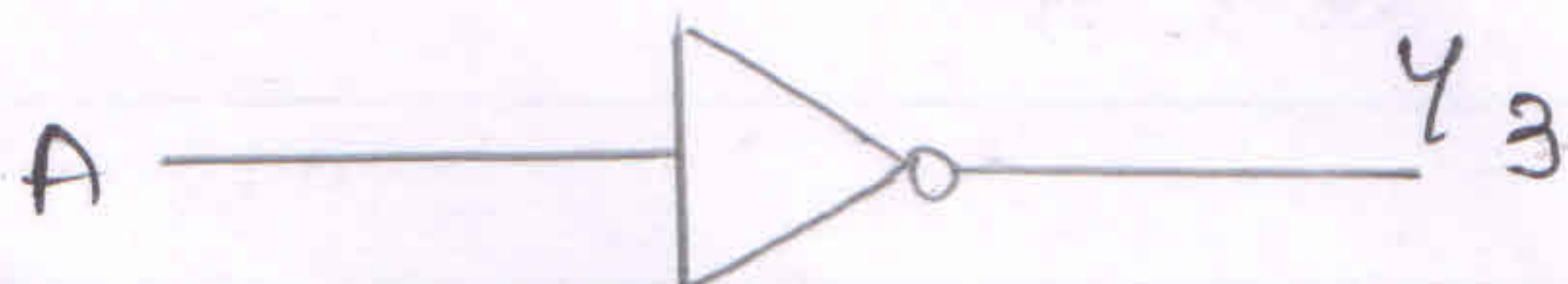


$$Y_2 = \bar{A}B + A\bar{B}$$

$$Y_2 = A \oplus B$$

I/P		O/P
A	B	$Y_2 = A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

(iv) NOT gate:



$$Y_3 = \bar{A}$$

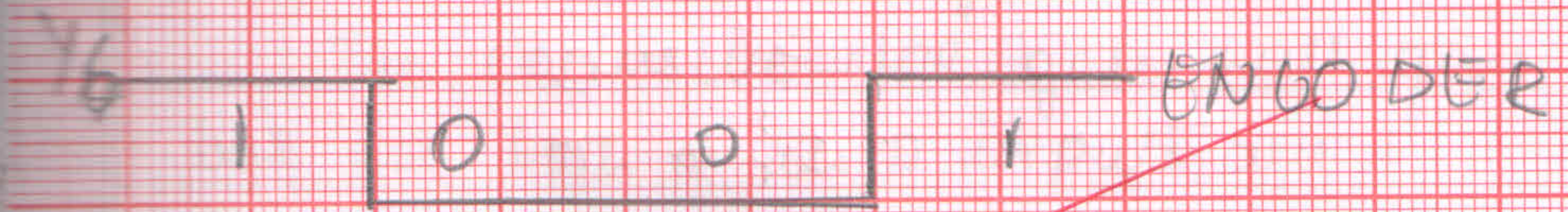
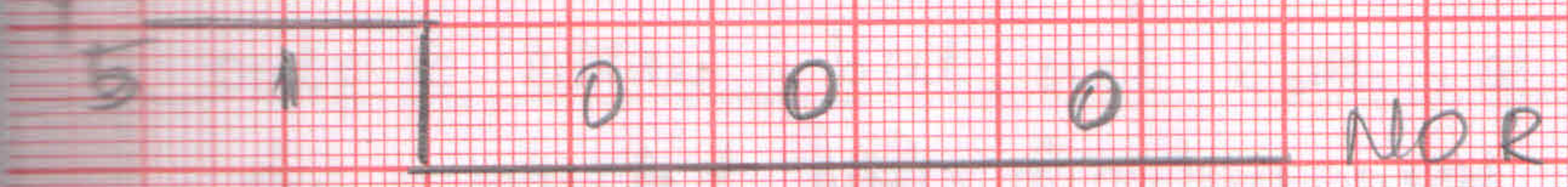
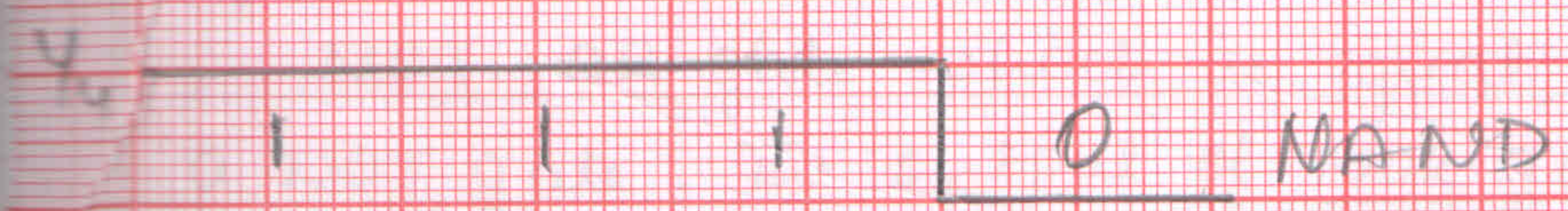
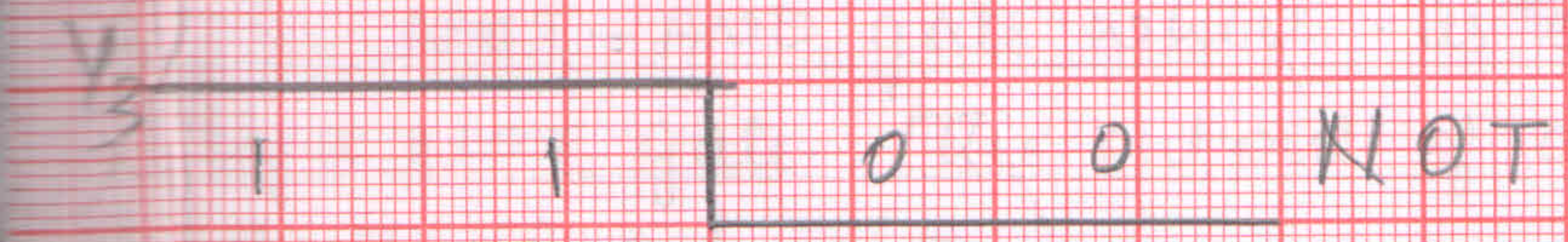
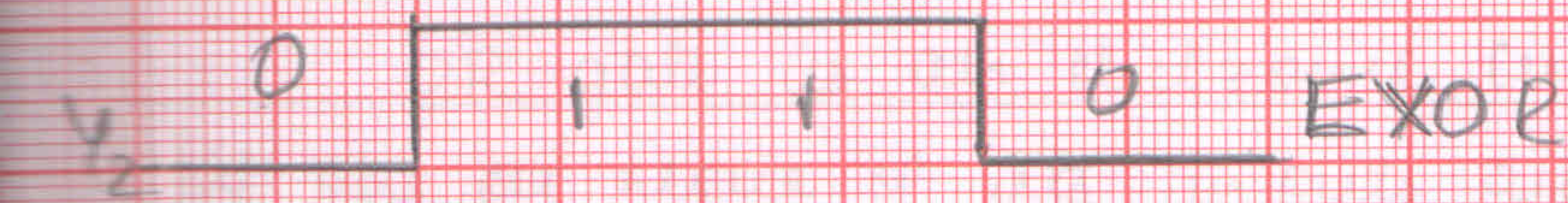
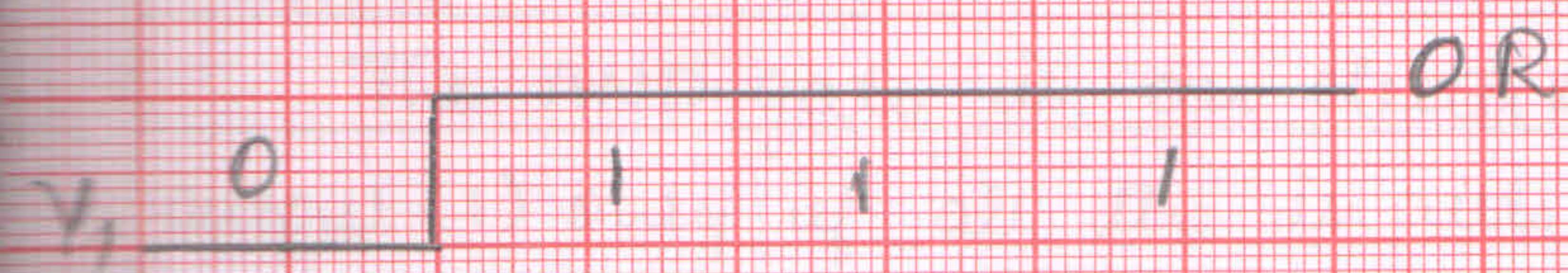
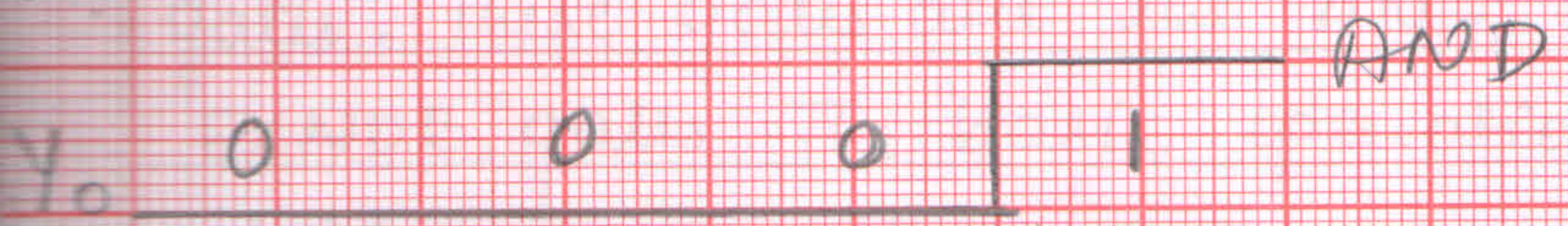
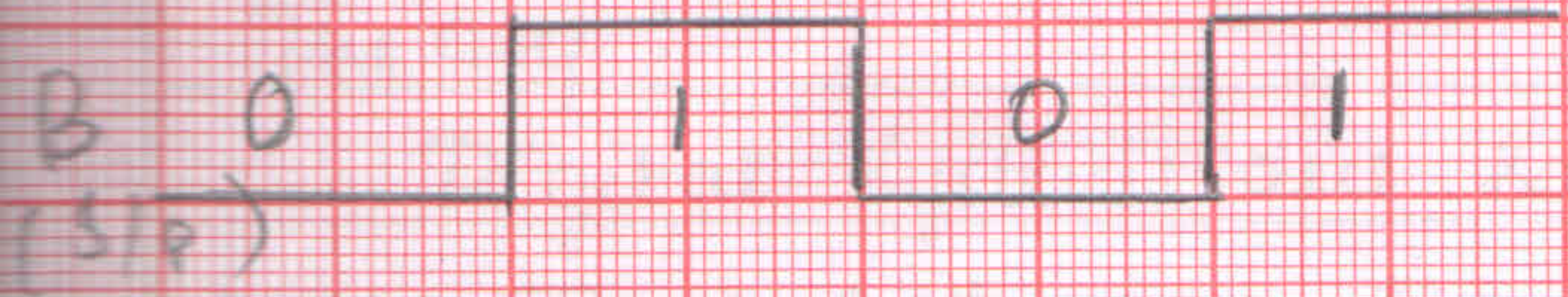
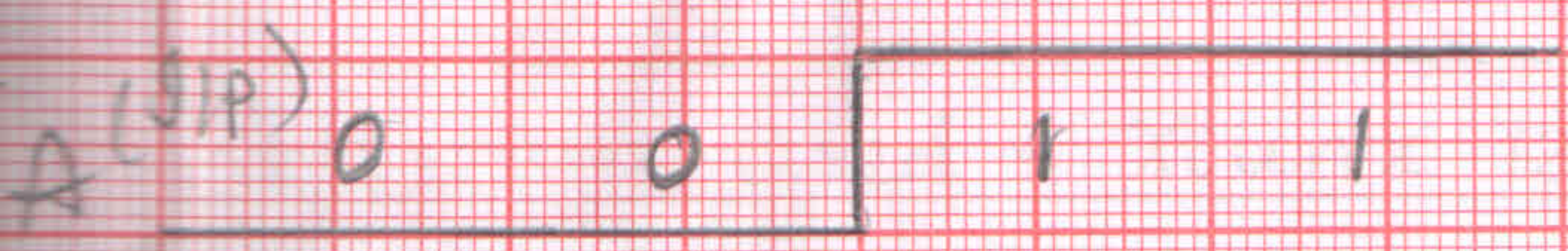
I/P	O/P
A	$Y = \bar{A}$
0	1
1	0

BASIC GATES

Scale —

1 cm = 1 μ sec

1 cm = 1 Unit



1 μ s 1 μ s 2 μ s 3 μ s 2 μ s

AIM: To verify write a verilog code to verify functionality of all basic gates.

Software used - Xilinx 13.1, Spartan 6

Theory:

Digital Systems are said to be constructed using logic gates. These gates are the AND, OR, NOT, NAND, NOR, EXOR and EXNOR gates. The basic operations are described using truth table.

The AND gate is an electronic circuit that gives a high output (1) only if all the inputs are high. A dot ($\cdot$) is used to show the AND operation i.e. $A \cdot B$.

OR gate is an electronic circuit that gives a high output if one or more of its inputs are high. The (+) is used to show the OR operation.

NOT-AND gate which is equal to an AND gate followed by a NOT gate. The output of all NAND gates are high if any of the input is 0.

NOT-OR gate which is equal to an OR gate followed by a NOT gate. The output of NOR gate are low if any i/p is 1.

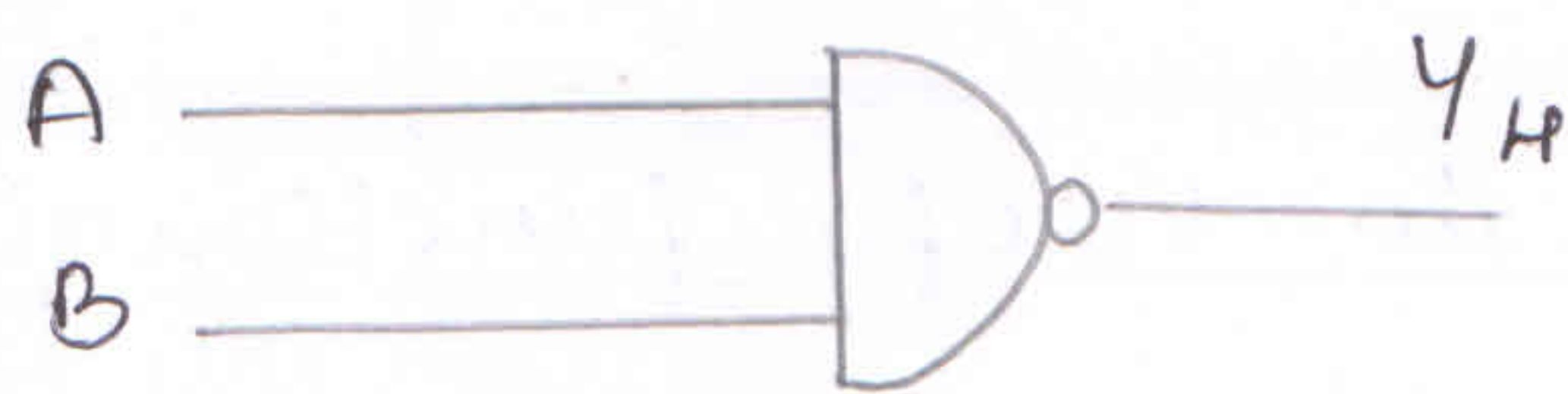
The exclusive-OR gate is a circuit which will give a high output either, but not both, of its two inputs are high.

An encircled plus sign is used to show the EXOR operation.

The exclusive-NOR gate will give a low output if either but not both of its 2 inputs are 1.

The NOT gate is an electronic circuit that produces an inverted version of the input at its output. It is also known as inverter.

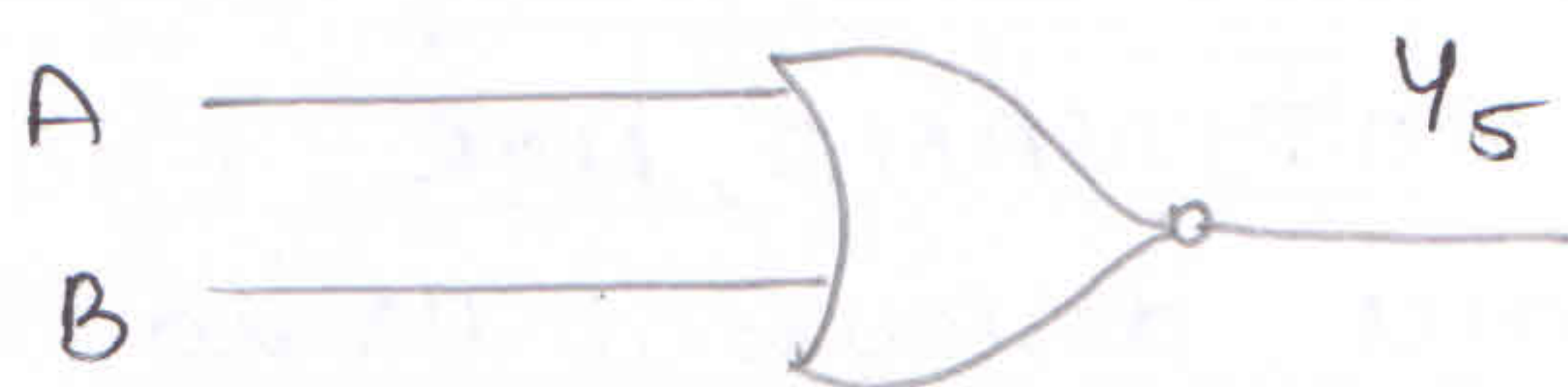
v) NAND gate :



$$Y_4 = \overline{A \cdot B}$$

I/P		O/P
A	B	$Y_4 = \overline{AB}$
0	0	1
0	1	1
1	0	1
1	1	0

vi) NOR gate :



$$Y_5 = \overline{A + B}$$

I/P		O/P
A	B	$Y_5 = \overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

vii) Ex-NOR gate :



$$Y_6 = \overline{AB} + AB$$

$$Y_6 = \overline{A \oplus B}$$

I/P		O/P
A	B	$Y_6 = \overline{A \oplus B}$
0	0	1
0	1	0
1	0	0
1	1	1

PIN CONFIGURATION :

Inputs :

A - 121

B - 123

Outputs :

Y₀ - P120

Y₂ - P118

Y₄ = P116

Y₆ = P114.

Y₁ - P119

Y₃ - P117

Y₅ = P115

CODE:

```
Module gates (A, B, Y0, Y1, Y2, Y3, Y4, Y5, Y6);
```

```
inputs A, B;
```

```
outputs Y0, Y1, Y2, Y3, Y4, Y5, Y6;
```

```
assign Y0 = A & B;
```

```
assign Y1 = A | B;
```

```
assign Y2 = A ^ B;
```

```
assign Y3 = ~A;
```

```
assign Y4 = ~(A & B);
```

```
assign Y5 = ~(A | B);
```

```
assign Y6 = ~(A ^ B);
```

```
end Module.
```

RESULT:

Hence Verified the functionality of all basic gates using verilog code.

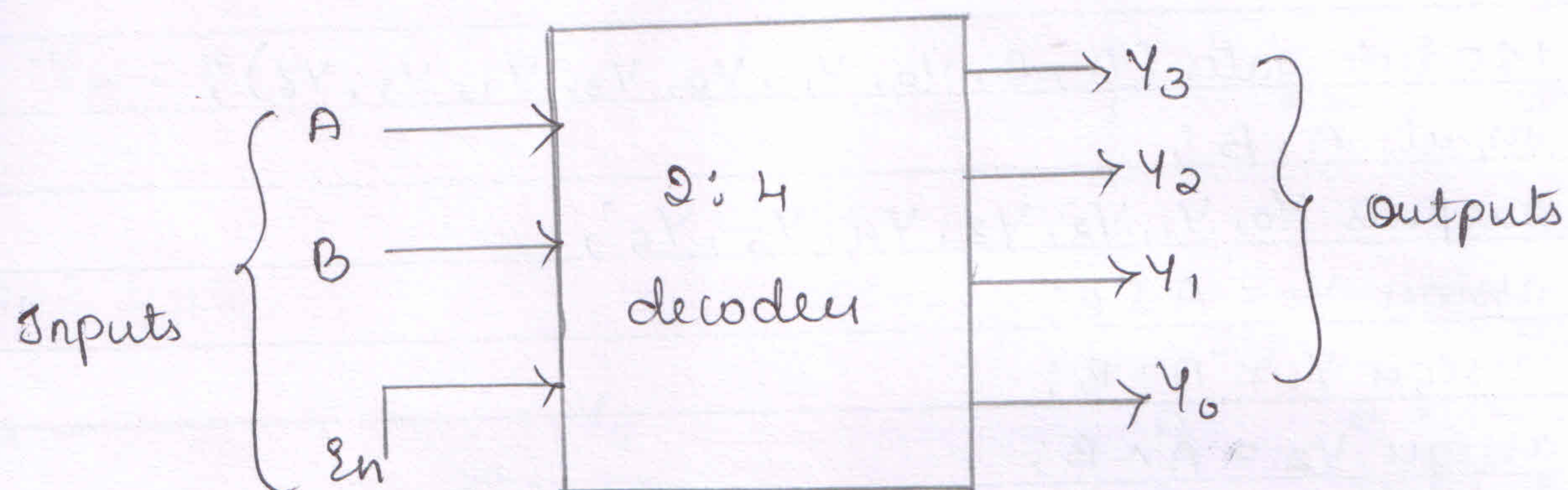
Record - 10

Observation - 10

Total - 20

Junit

BLOCK DIAGRAM



Truth table:

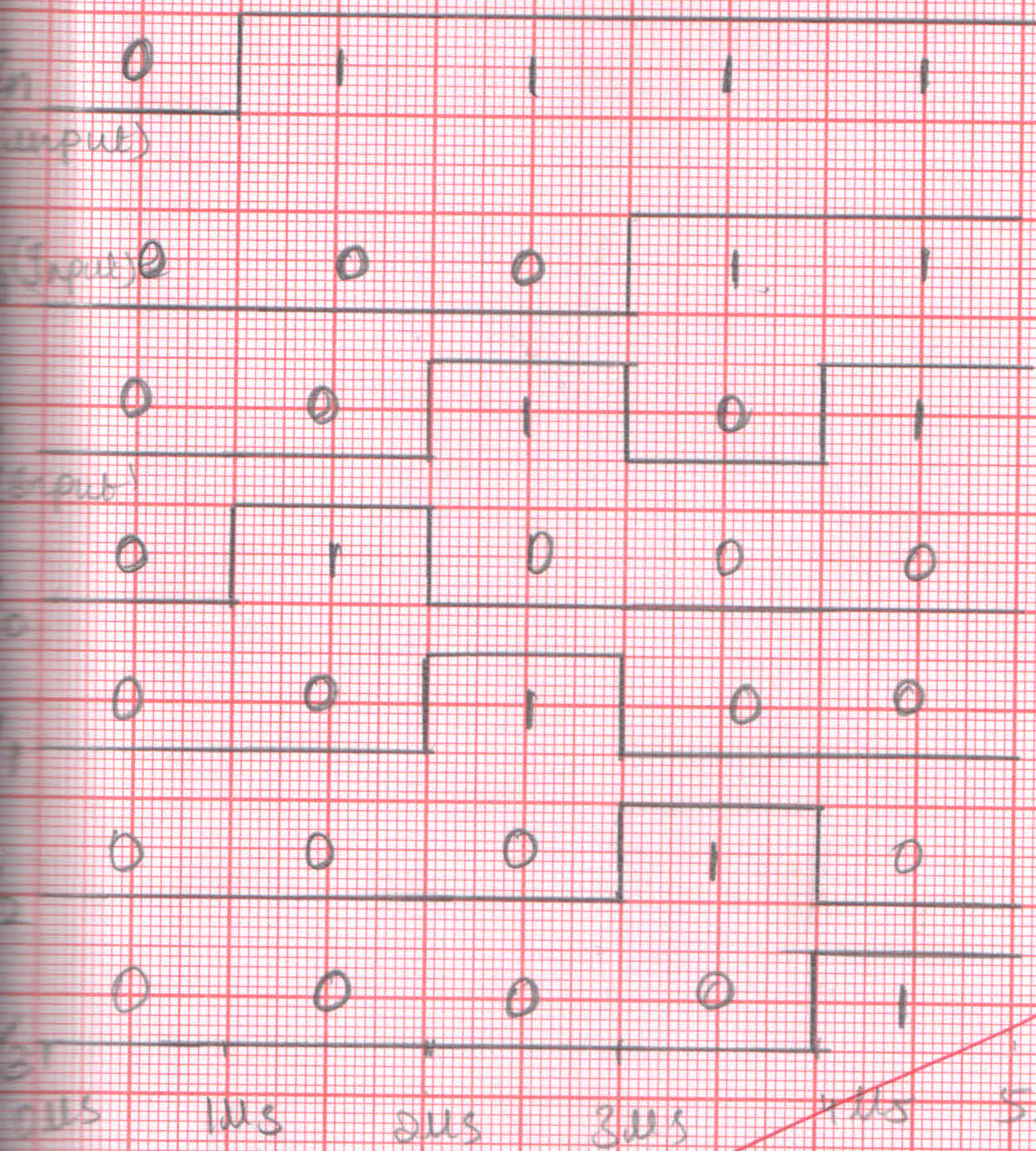
En	A	B	Y_3	Y_2	Y_1	Y_0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

$$Y_0 = \text{En} \bar{A} \bar{B}$$

$$Y_1 = \text{En} \bar{A} B$$

$$Y_2 = \text{En} A \bar{B}$$

$$Y_3 = \text{En} A B$$



Scale —

x axis 2cm = 1ns

y axis 1cm = 1 unit

2:4 DECODER

AIM: To write a verilog code to verify the functionality of 2:4 decoder.

SOFTWARE USED: Xilinx 13.1 Spartan 6.

THEORY:

A decoder is a circuit that changes a code into a set of signals. It is called as decoder because it does the reverse of encoding, but we will begin our study of encoders and decoders with decoders because they are simple to design.

A common type of decoder is the line decoder which takes an n -digit binary number and decodes it into 2^n data lines.

The simplest is 1 to 2 decoder.

2:4 line decoder only slightly more complex is the 2:4 decoder.

Larger line decoders can be designed in a similar fashion, but just like with the binary adder there is a way to make larger decoders using basic decoders.

Decoder has 2 inputs namely A and B along with an enable signal which is an active high signal and the 4 outputs denoted as Y_3, Y_2, Y_1, Y_0 . This circuit can be implemented using AND gates also.

CODE:

```
Module decoder(En, A, B, Y0, Y1, Y2, Y3);
    inputs En, A, B;
```


Pin Configuration:

Inputs:

Σ_n — P124

a — P121

b — P123

Outputs:

Y_3 — P117

Y_2 — P118

Y_1 — P119

Y_0 — P120.


```
outputs Y0, Y1, Y2, Y3;  
assign Y0 = En & (~A) & (~B);  
assign Y1 = En & (~A) & B;  
assign Y2 = En & A & (~B);  
assign Y3 = En & A & B;  
end Module.
```

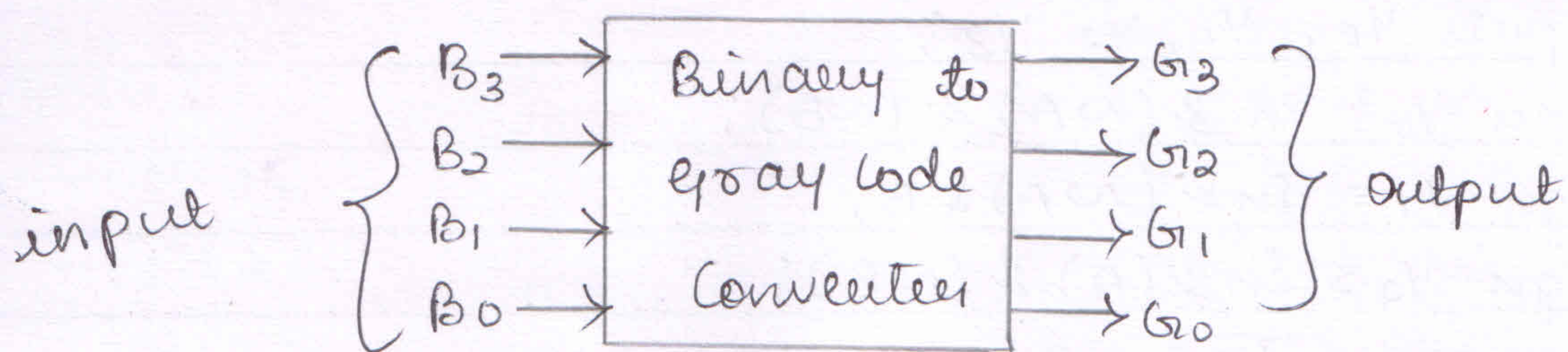
RESULT:

Hence verified the functionality of 2:4 decoder using verilog code.

Record — 10
Observation — 10
Total — 20

Signature

BLOCK DIAGRAM:



TRUTH TABLE:

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

Input

83

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

82

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

81

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

80

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

79

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

78

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

77

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

76

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

0ms

1ms

2ms

3ms

4ms

5ms

6ms

7ms

Scale —

x axis 1cm = 1ms

y axis 1cm = 1unit

BINARY TO GRAY

AIM : To write a verilog code to verify the code conversion from binary to gray.

SOFTWARE USED: Xilinx 13.1, Spartan 6.

Theory:

The reflected binary code, also known as Gray Code after Frank Gray, is a binary numeral system where 2 successive values differ in only one bit. The reflected binary code was originally designed to present spurious output from electromechanical switches. Today Gray codes are widely used to facilitate error correction in digital communication such as digital terrestrial receivers and some cable systems.

The Gray Code is Unweighted and is not an arithmetic code, that is there are no specific weights to the bit positions. The important feature of Gray Code is that it exhibits only a single bit change from one code word to the next in sequence. This property is important in many applications, such as shaft position encoders, where error susceptibility increases with the number of bits changes.

CODE:

```
Module binary_to_gray(B3, B2, B1, B0, G3, G2, G1, G0);
input B3, B2, B1, B0;
```


Equations:

$$G_3 = B_3$$

$$G_2 = B_2 \oplus B_3$$

$$G_1 = B_1 \oplus B_2$$

$$G_0 = B_0 \oplus B_1$$

Pin Configuration:

Inputs

B_0 - P121

B_1 - P123

B_2 - P124

B_3 - P126

Outputs

G_0 - P120

G_1 - P119

G_2 - P118

G_3 - P117

outputs $G_3, G_2, G_1, G_0;$

assign $G_3 = B_3;$

assign $G_2 = B_2 \wedge B_3;$

assign $G_1 = B_1 \wedge B_2;$

assign $G_0 = B_0 \wedge B_1;$

end Module.

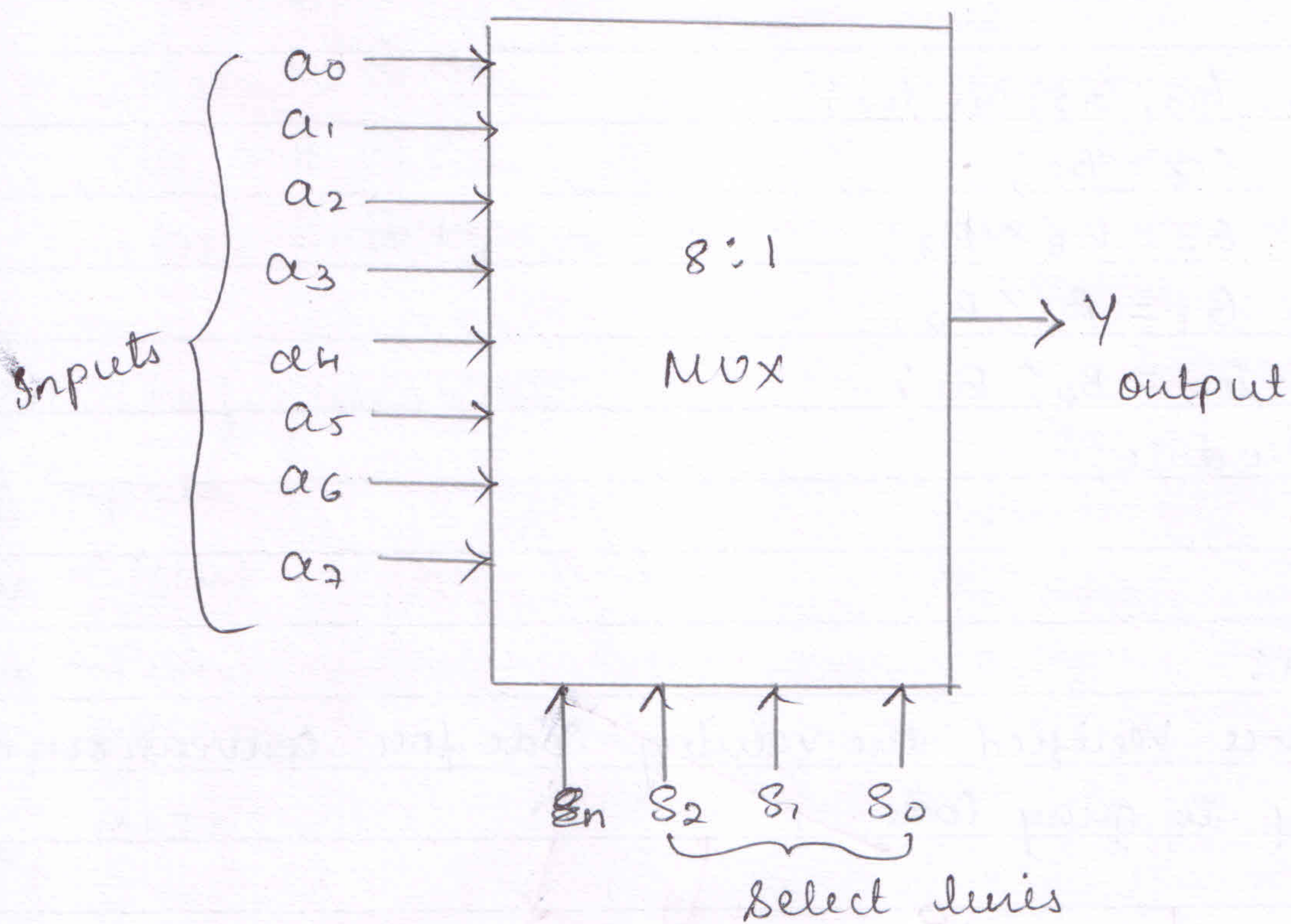
RESULT:

Hence Verified the verilog code for conversion of binary to gray code.

Record — 10
Observation — 10
Total — 20

Sum

BLOCK DIAGRAM:



TRUTH TABLE:

Inputs				Output
S_n	S_2	S_1	S_0	Y
0	X	X	X	Z
1	0	0	0	a_0
1	0	0	1	a_1
1	0	1	0	a_2
1	0	1	1	a_3
1	1	0	0	a_4
1	1	0	1	a_5
1	1	1	0	a_6
1	1	1	1	a_7

DAT

EXF

0 1 1 1 1 1 1 1

MSB

0 0 0 0 1 1 1 1

000 001 010 011 100 101 110 111

1 1 1 1 0 0 0 0

1 1 1 1 1 1 1 1

8:1 MUX

Scale -

X axis 1 cm = 1 μ sec

Y axis 1 cm = 1 unit

AIM: To write a Verilog code to verify functionality of 8:1 MUX.

SOFTWARE USED: Xilinx 13.1 Spartan 6

Theory:

There are 8 input lines, 1 output line and 3 select lines available in 8:1 MUX. Based on the values of selection lines one input line is selected to the output port. A Multiplexer or a data selector or a input selector is a combinational circuit device that select one of N inputs and provides it on its output.

Multiplexers are useful in any application in which data must be chosen from multiple source to a single destination. Multiplexers are also heavily used in I/O operations, data buses and register files, additional multiplexers are also found their way to various other circuits such as address.

It is qualified for automatic applications wide operating voltage range of 2V to 5V

Outputs can drive upto 10 LSTTL loads.

± 6 mV output drive at 5V.

low input current of 1 μ A

It can also perform as

- Boolean function generator
- parallel to serial converters.
- data source selectors.

Pin Configuration

Inputs :

$a_0 - 121$

$a_1 - 123$

$a_2 - 124$

$a_3 - 126$

$a_4 - 127$

$a_5 - 131$

$a_6 - 132$

$a_7 - 133$

$z_n - 134$

$s_2 - 137$

$s_1 - 138$

$s_0 - 139$

Output

$y - 120$

Code:

```
Module eight-to-one-mux (y, a, s, e);  
output y;  
input [7:0] a, [2:0] s, e;  
reg y;  
always @ (a, s, e) &  
begin  
if (e == 1)  
case (s)  
3'b000: y = a[0];  
3'b001: y = a[1];  
3'b010: y = a[2];  
3'b011: y = a[3];  
3'b100: y = a[4];  
3'b101: y = a[5];  
3'b110: y = a[6];  
3'b111: y = a[7];  
default: y = 1'bZ;  
end case  
else y = 1'bZ;  
end  
end Module
```

RESULT:

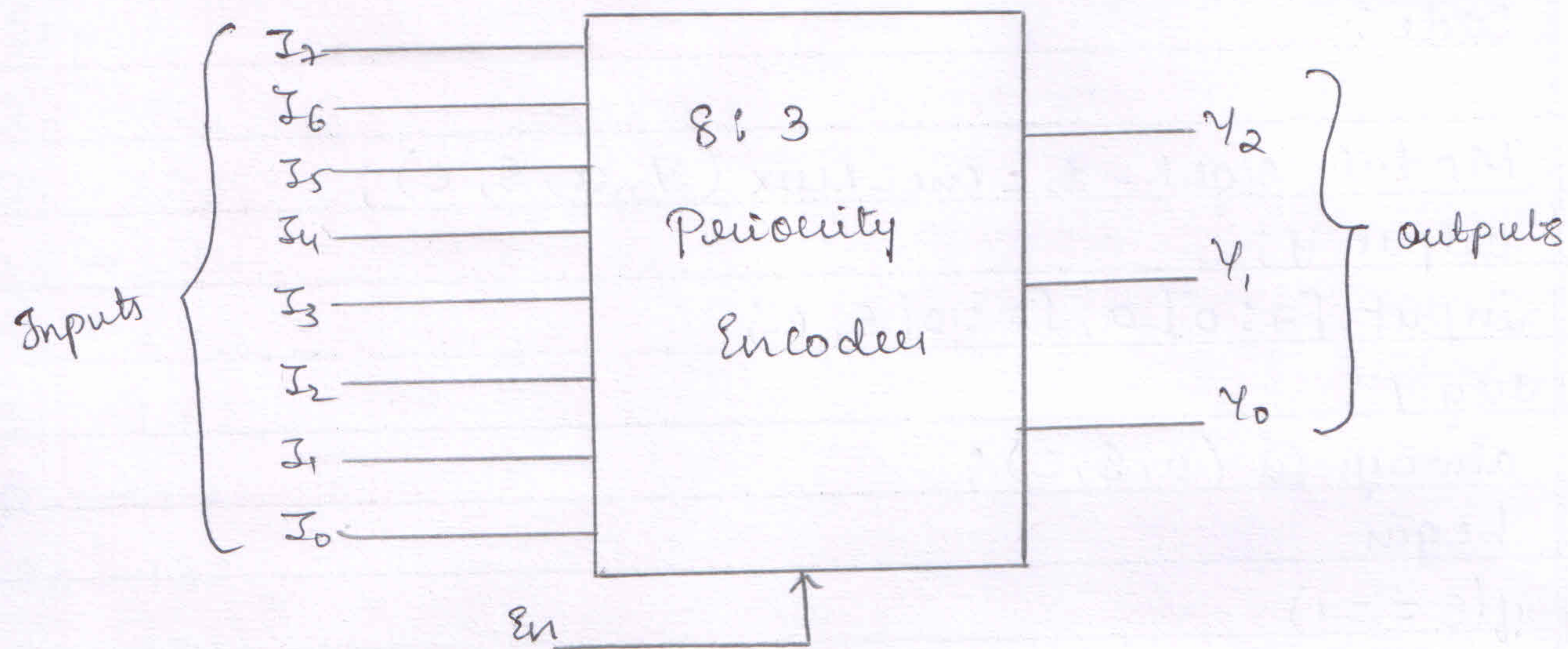
Hence verified the functionality of the 8:1 Mux using Verilog code.

Record - 10
Observation - 10
Total - 20

Amal
29/8/18

DESIGN :

BLOCK DIAGRAM :



En	I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0	Y_2	Y_1	Y_0
0	X	X	X	X	X	X	X	X	0	0	0
1	1	X	X	X	X	X	X	X	0	1	1
1	0	1	X	X	X	X	X	X	1	1	0
1	0	0	1	X	X	X	X	X	1	0	1
1	0	0	0	1	X	X	X	X	1	0	0
1	0	0	0	0	1	X	X	X	0	1	1
1	0	0	0	0	0	1	X	X	0	1	0
1	0	0	0	0	0	0	1	X	0	0	1
1	0	0	0	0	0	0	0	1	0	0	0

DATE ..29/8/18....

EXPT TITLE:

8:3 PRIORITY ENCODER

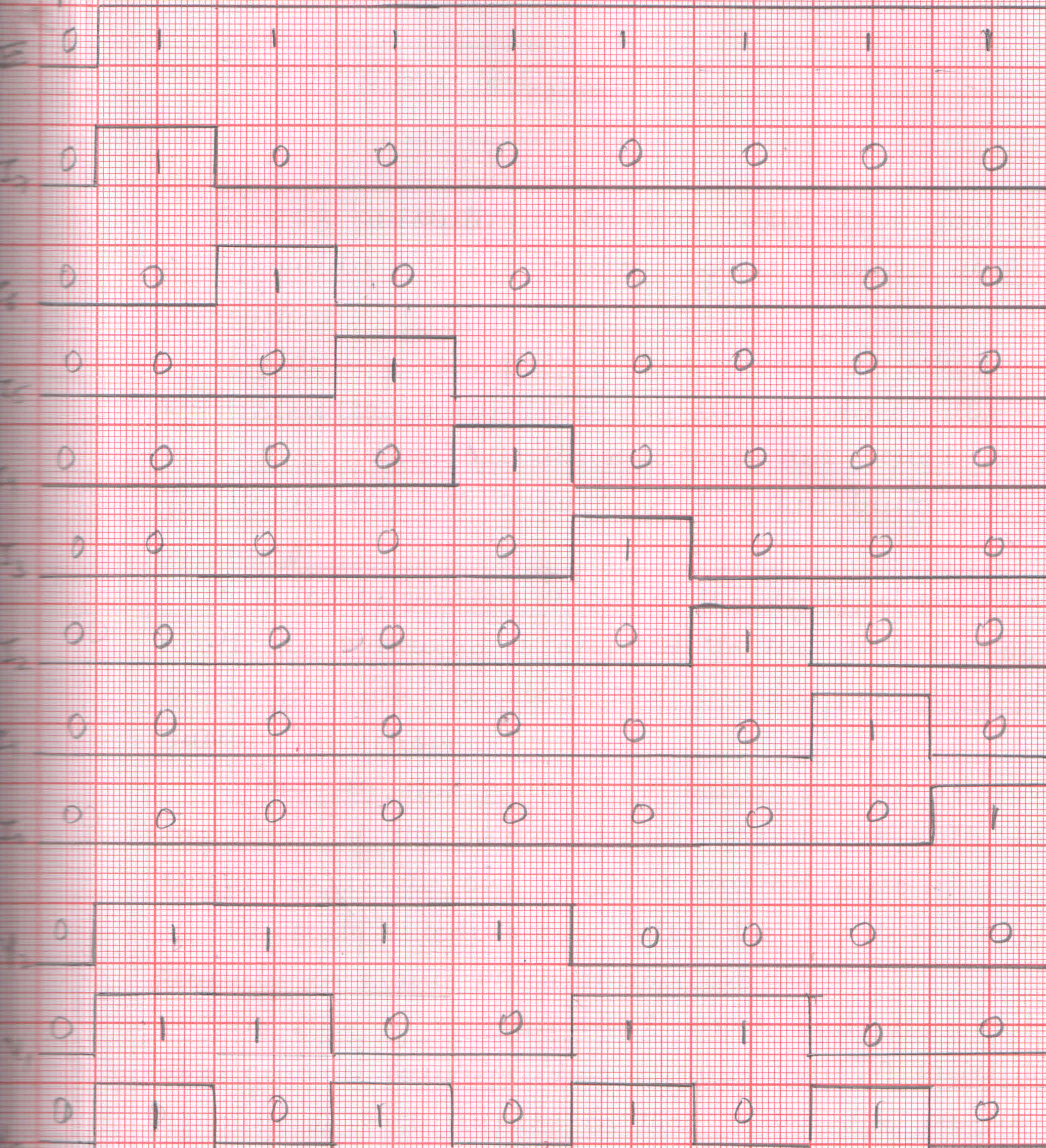
EXP. NO.5.....

PAGE NO.

11

AIM: To write the verilog code to verify the functionality

Inputs



0ns 1ns 2ns 3ns 4ns 5ns 6ns 7ns 8ns 9ns

Scale = x axis 1cm = 1ns

→ time scale

y axis 1cm = 1bit

AIM: To write the verilog code to verify the functionality of 8:3 priority encoder.

TOOL USED: Xilinx 13.1, Spartan 6.

Theory:

The 8:3 priority encoder has eight active low inputs and provides a 3 bit code of highest ranked input at its output.

Considering case of 8:3 line encoder in which, if E_i are 1 simultaneously, the encoder produces the output as 11 that means all three outputs are equal to logic 1. This value does not correspond to either binary 2 or binary 3.

A priority encoder is one of the types of encoders in which an ordering is imposed to the inputs that means compared with the standard encoder, it includes the priority function.

However, this priority is based on the relative magnitudes of the input. Hence, the input with larger magnitude is the one that is encoded first.

Priority encoders can select the inputs with highest priority in many practical applications.

This type of encoder has 8 inputs and three outputs that generate corresponding binary code. A priority is assigned to each input so that where two or more inputs are 1 at a time, the input with highest priority will be the output.

Pin Configuration:

Inputs:

$en \rightarrow P121$

$I_7 \rightarrow P123$

$I_6 \rightarrow P124$

$I_5 \rightarrow P126$

$I_4 \rightarrow P127$

$I_3 \rightarrow P131$

$I_2 \rightarrow P132$

$I_1 \rightarrow P133$

$I_0 \rightarrow P134$

Output:

$Y_0 \Rightarrow 120$

$Y_1 \rightarrow 119$

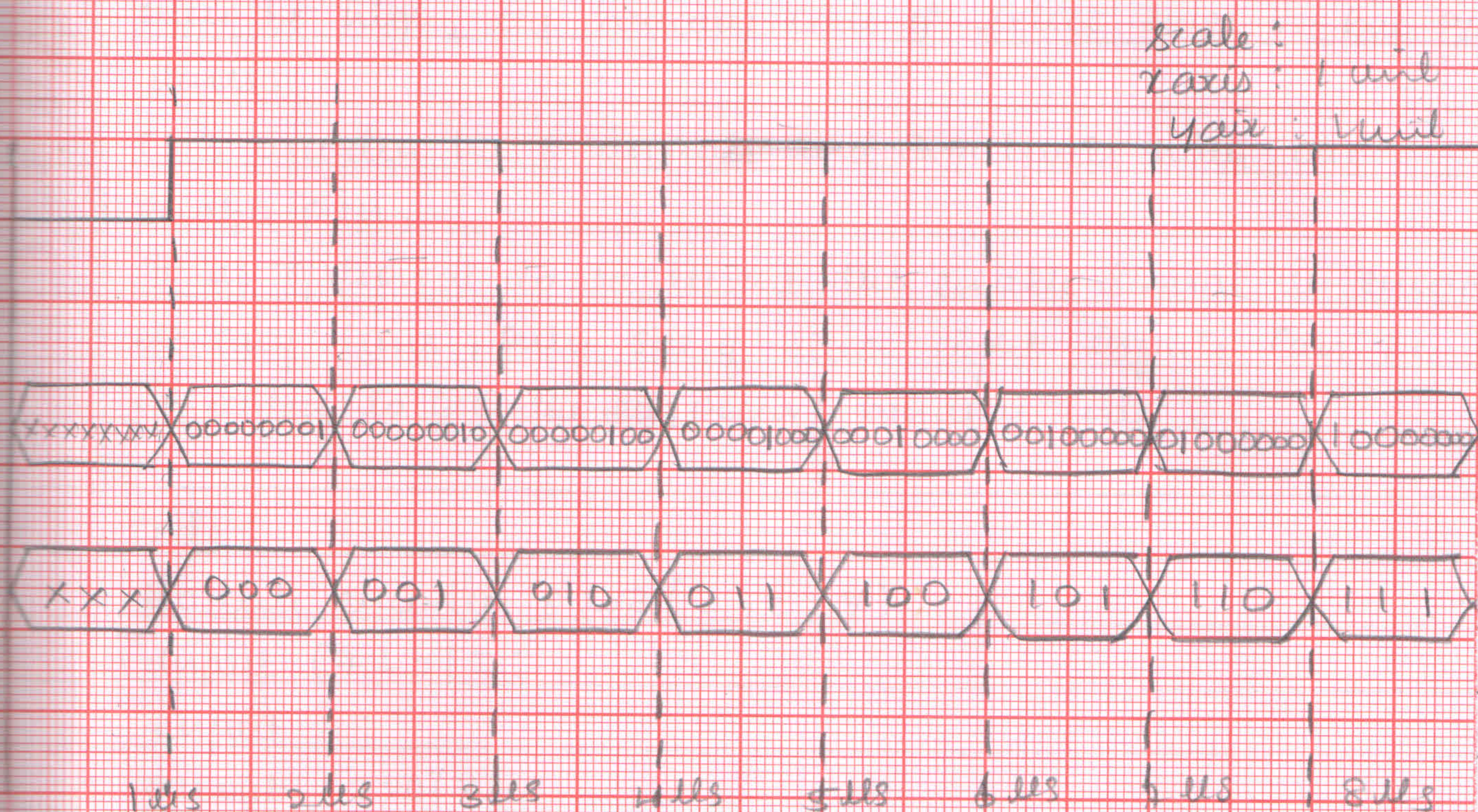
$Y_2 \rightarrow 118$

AIM: To write a Verilog Code for realizing 8 to 3 encoder without priority.

TOOLS USED: Xilinx 13.1, FPGA KIT Spartan 6.

Theory:

2^n input lines and atmost only one of them will



```
always @ (I, E)
```

```
begin
```

```
if (E)
```

```
begin
```

```
if (I == 8'b00000001) Y = 3'b000;
```

```
else if (I == 8'b00000010) Y = 3'b001;
```

```
else if (I == 8'b00000100) Y = 3'b010;
```


AIM: To write a Verilog Code for realizing 8 to 3 encoder without priority.

TOOLS USED: Xilinx 13.1, FPGA KIT Spartan 6.

Theory:

2^n input lines and atmost only one of them will ever be high, the binary code of select lines n -bit o/p lines.

8:3 encoder has 8 inputs and 3 outputs with one enable pin.

It is optional to represent the enable signal in encoders. At any time, only one of these eight inputs can be '1' in order to get the respective binary code.

Code:

```
Module encoder 8 to 3 (I, E, Y);  
input [7:0] I;  
input E;  
output [2:0] Y;  
reg [2:0] Y;  
always @ (I, E)  
begin  
if (E)  
begin  
if (I == 8'b00000001) Y = 3'b000;  
else if (I == 8'b00000010) Y = 3'b001;  
else if (I == 8'b00000100) Y = 3'b010;
```


Truth table:

Σn	I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0	Y_2	Y_1	Y_0
0	x	x	x	x	x	x	x	x	x	x	x
1	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	1	0	0	0	1
1	0	0	0	0	0	1	0	0	0	1	0
1	0	0	0	0	1	0	0	0	0	1	1
1	0	0	0	1	0	0	0	0	1	0	0
1	0	0	1	0	0	0	0	0	1	0	1
1	0	1	0	0	0	0	0	0	1	1	0
1	1	0	0	0	0	0	0	0	1	1	1
1	1	1	0	0	0	0	0	0	x	x	x

Pin Configuration:

i/p:

$\Sigma n \rightarrow P_{121}$

$I_1 \rightarrow P_{134}$

$I_2 \rightarrow P_{133}$

$I_3 \rightarrow P_{132}$

$I_4 \rightarrow P_{127}$

$I_5 \rightarrow P_{126}$

$I_6 \rightarrow P_{124}$

$I_7 \rightarrow P_{123}$

o/p

$Y_0 \rightarrow P_{119}$

$Y_1 \rightarrow P_{118}$

$Y_2 \rightarrow P_{117}$


```
else if (I == 8'b00001000) Y = 3'b011;
```

```
else if (I == 8'b00010000) Y = 3'b100;
```

```
else if (I == 8'b00100000) Y = 3'b101;
```

```
else if (I == 8'b01000000) Y = 3'b110;
```

```
else if (I == 8'b10000000) Y = 3'b111;
```

```
else Y = 3'bxxx;
```

```
end
```

```
else
```

```
Y = 3'bxxx;
```

```
end
```

```
end module.
```

RESULT:

verified the functionality of 8:3 encoders without
penalty

Received - 10
Observation - 10
Total - 20

Dr. J. S. J.

Code :

Module encoder - lines (E, I, Y);

input E;

input [7:0] I;

output [2:0] Y;

reg [2:0] Y;

always @ (E, I)

begin

if (E == 1)

begin

if (I[7] == 1) Y = 3'b111;

else if (I[6] == 1) Y = 3'b110;

else if (I[5] == 1) Y = 3'b101;

else if (I[4] == 1) Y = 3'b100;

else if (I[3] == 1) Y = 3'b011;

else if (I[2] == 1) Y = 3'b010;

else if (I[1] == 1) Y = 3'b001;

else if (I[0] == 1) Y = 3'b000;

end

else Y = 3'b000;

end

end Module .

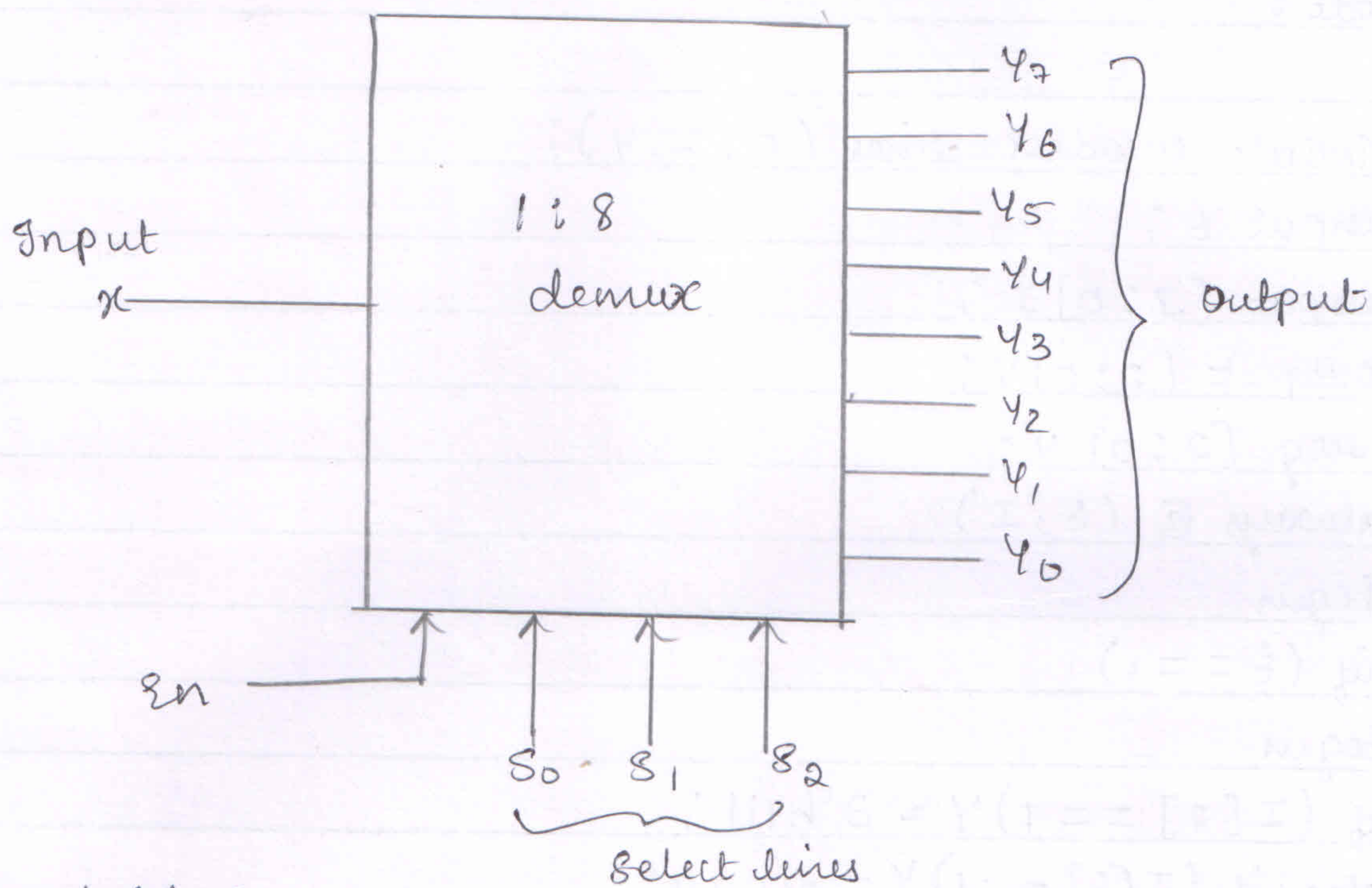
RESULT:

Hence verified the functionality of 8:3 priority encoder using verilog code.

Record
Observed
Total
10
10
20
Final

Design:

Block diagram:



Truth table:

En	x	s ₂	s ₁	s ₀	y ₇	y ₆	y ₅	y ₄	y ₃	y ₂	y ₁	y ₀
0	x	x	x	x	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0
1	0	1	0	1	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0
1	0	1	1	1	0	0	0	0	0	0	0	0

AIM: To write a verilog code to verify the functionality of 1:8 demultiplexer.

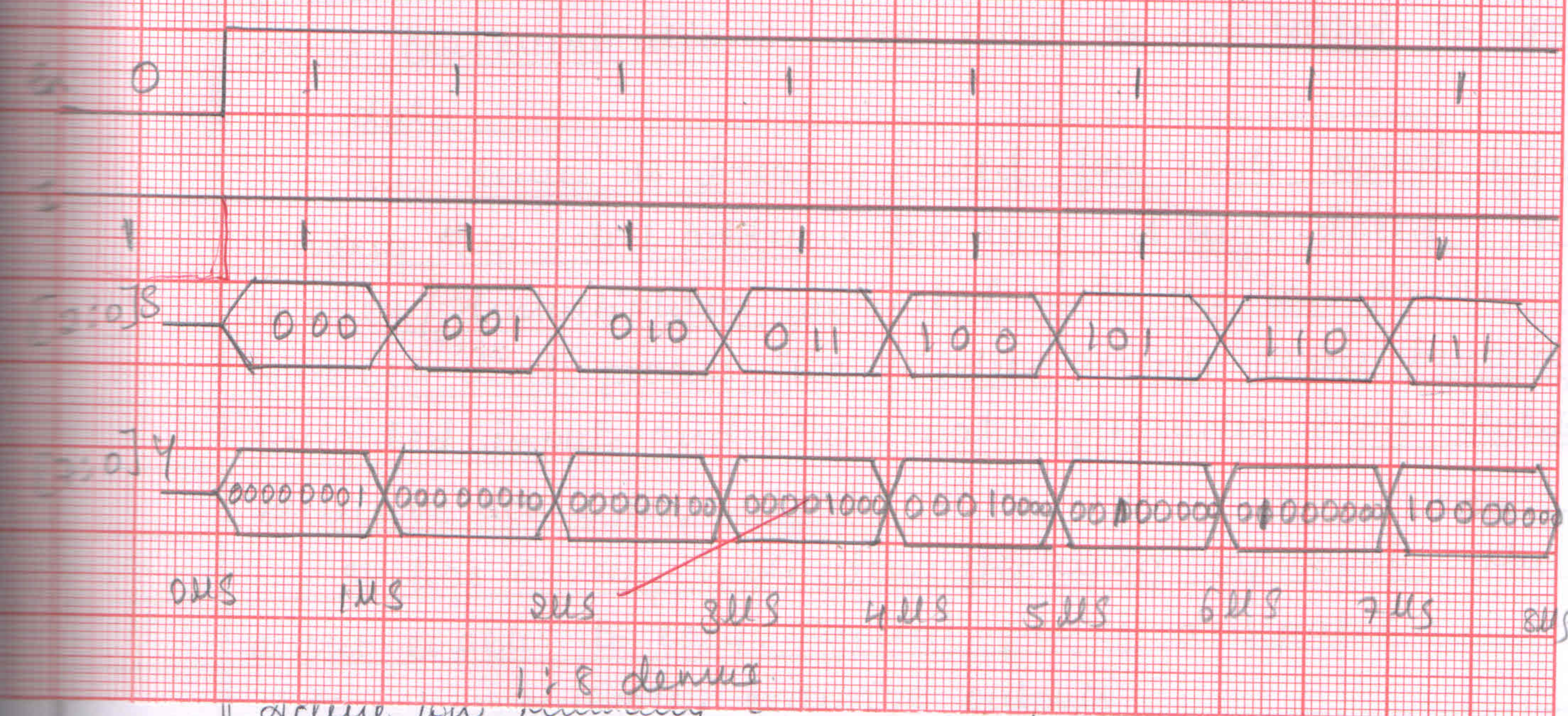
TOOL USED: Xilinx 13.1, Spartan 6.

THEORY:

A 1:8 demultiplexer circuit has 1 input line with enable signal and 8 output lines starting

Scale: X axis: 2cm = 1 μ s

Y axis: 1cm = 1 bit



CODE:

```
Module demux (y, i, en, s);
    output [7:0] y;
    input [2:0] s, i, en;
    reg [7:0] y;
    always @ (i, en, s)
    begin
```


AIM: To write a Verilog code to verify the functionality of 1:8 demultiplexer.

TOOL USED: Xilinx 13.1, Spartan 6.

THEORY:

A 1:8 demultiplexer circuit has 1 input line with a active high enable signal and 8 output lines starting from O_7 to O_0 . This circuit were designed for light wave communications, and were demonstrated to operate at data.

This device is ideally suited for high speed bipolar memory chip select address decoding. A high speed 1:8 decoder/demultiplexer fabricated with the low power barrier diode process. The decoder accepts three binary weighted inputs and when enable provides eight mutually exclusive active low outputs.

It has a very high demultiplexing capability. Multiple input enables to easy. Typical power dissipation. Active low mutually exclusive outputs.

CODE:

```
Module demux (y, i, en, s);  
    output [7:0] y;  
    input [2:0] s, i, en;  
    reg [7:0] y;  
    always @ (i, en, s)  
    begin
```


Pin Configuration:

Input:

en $\rightarrow$ P121

S[2] $\rightarrow$ P123

S[1] $\rightarrow$ P124

S[0] $\rightarrow$ P126

X $\rightarrow$ P127

Output:

Y[7] $\rightarrow$ P120

Y[6] $\rightarrow$ P119

Y[5] $\rightarrow$ P118

Y[4] $\rightarrow$ P117

Y[3] $\rightarrow$ P116

Y[2] $\rightarrow$ P115

Y[1] $\rightarrow$ P114

Y[0] $\rightarrow$ P112


```
Y = 8'b00000000;
```

```
if (en == 1)
```

```
case(8)
```

```
3'b000: Y[0] = i;
```

```
3'b001: Y[1] = i;
```

```
3'b010: Y[2] = i;
```

```
3'b011: Y[3] = i;
```

```
3'b100: Y[4] = i;
```

```
3'b101: Y[5] = i;
```

```
3'b110: Y[6] = i;
```

```
3'b111: Y[7] = i;
```

```
default: Y = 8'b00000000;
```

```
endcase
```

```
else Y = 8'b00000000;
```

```
end
```

```
endmodule.
```

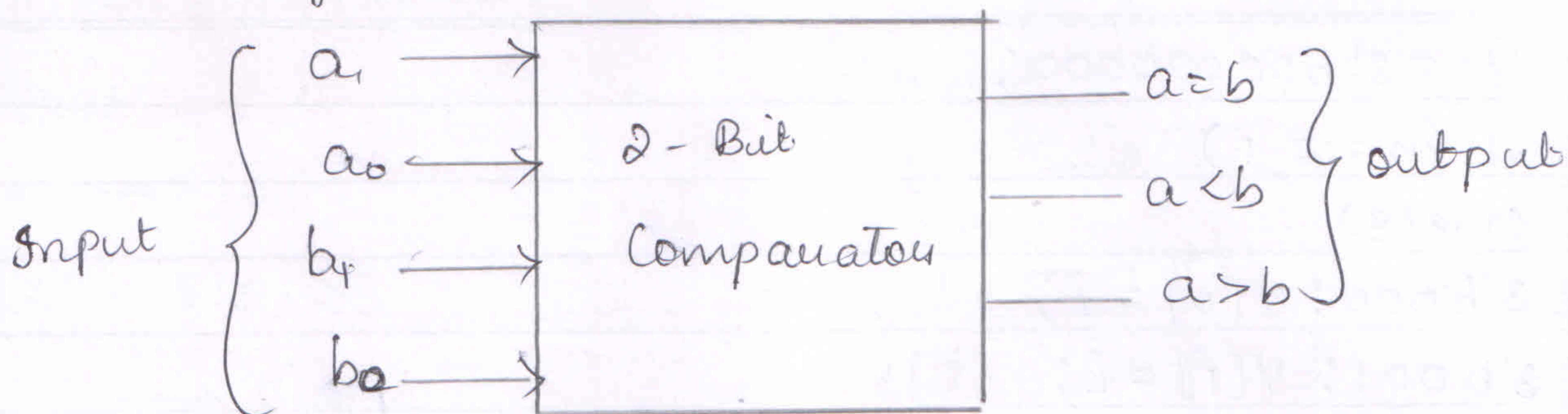
RESULT:

Hence verified the functioning of 1:8 demux using verilog code.

Record - 10
Observation - 10
Total - 20

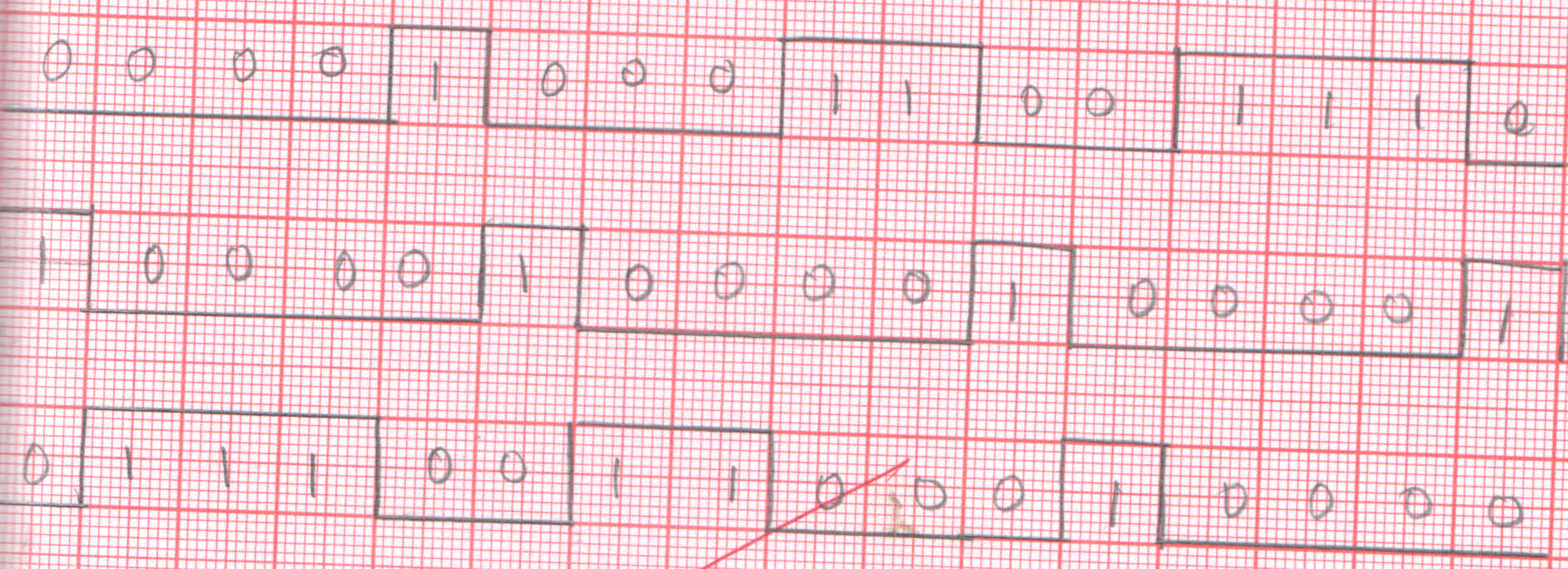
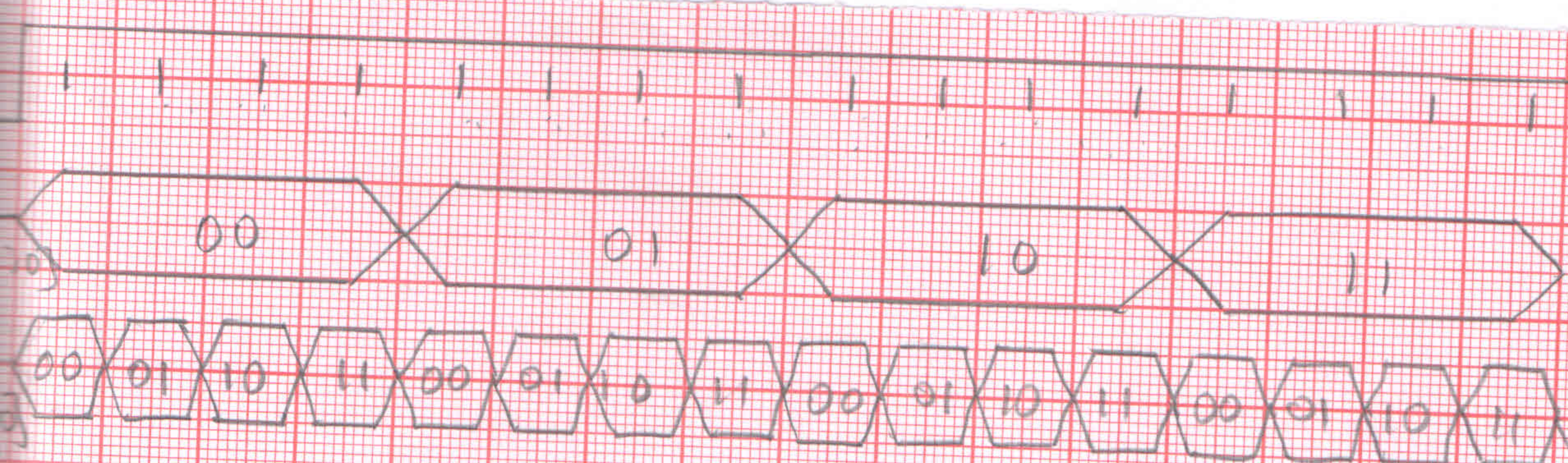
Signature

Block diagram:



Truth table:

en	a ₁	a ₀	b ₁	b ₀	(a=b) e _a	(a<b) l	(a>b) g
1	0	0	0	0	1	0	0
1	0	0	0	1	0	1	0
1	0	0	1	0	0	1	0
1	0	0	1	1	0	1	0
1	0	1	0	0	0	0	1
1	0	1	0	1	1	0	0
1	0	1	1	0	0	1	0
1	0	1	1	1	0	1	0
1	1	0	0	0	0	0	1
1	1	0	0	1	0	0	1
1	1	0	1	0	1	0	0
1	1	0	1	1	0	1	0
1	1	1	0	0	0	0	1
1	1	1	0	1	0	0	1
1	1	1	1	0	0	0	1
1	1	1	1	1	1	0	0
0	x	x	x	x	0	0	0



Scale : 1 unit = 1 μ sec
 Y-axis 1 unit = 1 cm

Obit Compacator

AIM: Write a verilog code and check the functionality of a 2 bit Comparator.

TOOL USED: Xilinx ISE 13.1, Spartan 6.

THEORY:

Data comparison is needed in digital systems while performing arithmetic or logical operations. This comparison determines whether one number is greater than, equal, or less than the other number. A digital comparator is widely used in combinational system and is specially designed to compare the relative magnitude of binary numbers.

These are also available in IC form with different bit comparing configuration such as 4-bit, 8-bit etc.

More than one comparator can also be connected in cascade arrangement to perform comparison of numbers of longer lengths. Whenever we want to compare the two binary numbers, first we have to compare the most significant bits.

If these MSB's are equal, then only we need to compare the next significant bits. But if the MSB's are not equal then, it would be clear that either A is greater than or less than B and the process of comparison ends.

CODE:

```
Module Compare (a, b, en, g, e, l);  
input en, [1:0]a, [1:0]b;
```


Pin Configuration:

Input:

en $\rightarrow$ P121

a[1] $\rightarrow$ P123

a[0] $\rightarrow$ P124

b[0] $\rightarrow$ P126

b[0] $\rightarrow$ P127

Output:

c $\rightarrow$ P120

d $\rightarrow$ P119

g $\rightarrow$ P118


```
output g, e, l;  
reg g, e, l;  
always @ (en, a, b)  
begin  
  if (en == 1)  
  begin  
    if (a > b) begin  
      g = 1; e = 0; l = 0; end  
    else if (a == b)  
    begin  
      e = 1; g = 0; l = 0; end  
    else  
    begin e = 0, g = 0, l = 1; end  
  end.  
  else  
  begin  
    g = 1'bZ; e = 1'bZ; l = 1'bZ;  
  end  
end  
end module
```

RESULT:

∴ The Verification of functionality of 2-bit Comparator is done Successfully.

Record — 10
Observation — 10
Total — 20

Just

Block diagram:



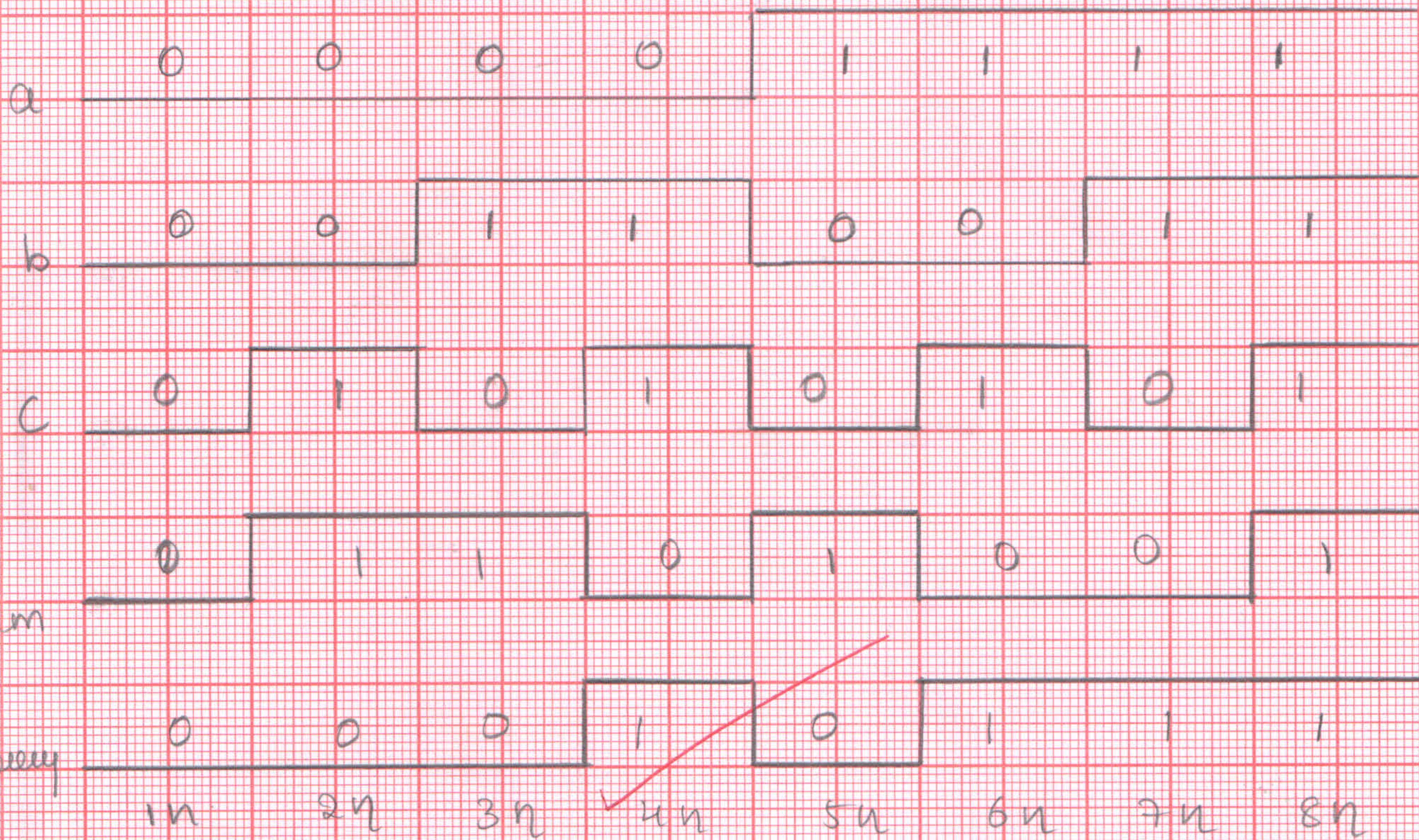
Truth table:

a	b	cin	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Full adder

Time State

2 unit = 1 ns



Aim: write a Verilog code to verify the functionality of full adder in (i) data flow style (ii) Behavioural style (iii) Structural (or) gate level.

Tool used: Xilinx 13.1, Spartan 6.

Theory:

A full adder is a logical circuit that performs an addition operation on three one-bit binary numbers. The full adder produces the sum of the three inputs and carry value. It can be combined with another full adder or work on its own.

A full adder can be constructed from two half adders by connecting A and B to the input of one half adder, connecting sum from that to an input to the second adder, connecting cin to the other input and OR the two carry outputs equivalently. S would be the three bits XOR of A, B, cin and Co would be the three bit majority function.

Code:

(i) Data flow style:

```
Module full adder (a, b, cin, sum, carry);
```

```
  i/p a, b, cin;
```

```
  o/p sum, carry;
```

```
  assign sum = a ^ b ^ c;
```

```
  assign carry = (a & b) | (b & c) | (c & a);
```

```
end Module
```


(ii) Behavioural Style:

```
Module full adder (a, b, cin, sum, carry);
```

```
  i/p a, b, cin;
```

```
  output sum, carry;
```

```
  reg sum, carry;
```

```
  always @ (a, b, cin);
```

```
  begin
```

```
    if (a == 0 && b == 0 && cin == 0)
```

```
    begin
```

```
      sum = 0; carry = 0;
```

```
    end
```

```
  else if (a == 0 && b == 0 && cin == 1)
```

```
  begin
```

```
    sum = 1;
```

```
    carry = 0;
```

```
  end
```

```
  else if (a == 0 && b == 1 && cin == 0)
```

```
  begin
```

```
    sum = 1;
```

```
    carry = 0;
```

```
  end
```

```
  else if (a == 0 && b == 1 && cin == 1)
```

```
  begin
```

```
    sum = 0;
```

```
    carry = 1;
```

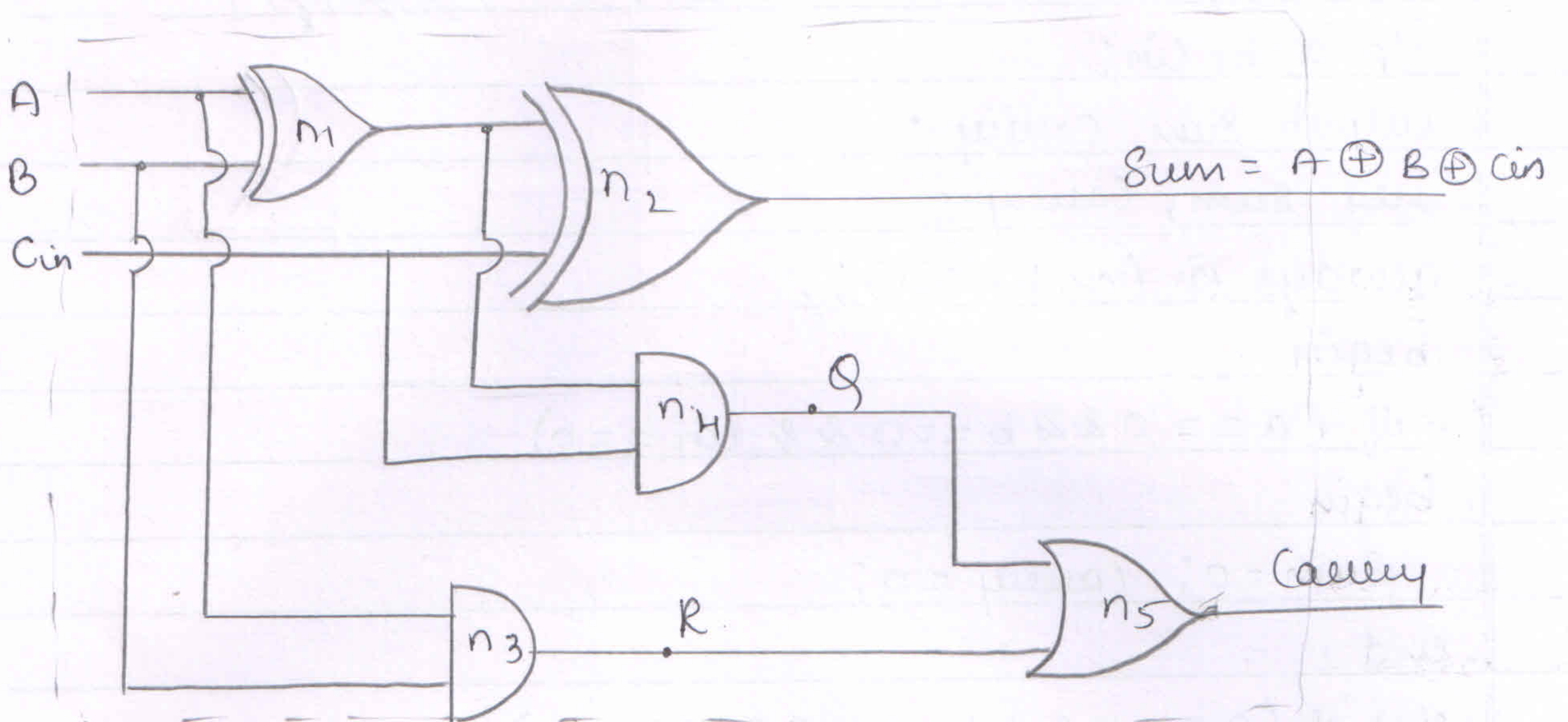
```
  else if (a == 1 && b == 0 && cin == 0)
```

```
  begin
```

```
    sum = 1;
```

```
    carry = 0;
```

```
  end
```

Pin Configuration:

Inputs:

P121 $\rightarrow$ a

P123 $\rightarrow$ b

P124 $\rightarrow$ Cin

output :

P120 $\rightarrow$ Sum

P119 $\rightarrow$ Carry


```
else if (a == 1 && b == 0 && cin == 1)
```

```
begin
```

```
    Sum = 0;
```

```
    Carry = 1;
```

```
end
```

```
else if (a == 1 && b == 1 && cin == 0) ;
```

```
begin
```

```
    Sum = 0;
```

```
    Carry = 1;
```

```
end
```

```
else if (a == 1 && b == 1 && cin == 1)
```

```
begin
```

```
    Sum = 1;
```

```
    Carry = 1;
```

```
end
```

```
end
```

```
end Module.
```

(iii) Gate level

```
Module full adder (a, b, cin, Sum, Carry);
```

```
input a, b, cin;
```

```
output Sum, Carry;
```

```
wire R, S, Q;
```

```
XOR(Y, a, b, c);
```

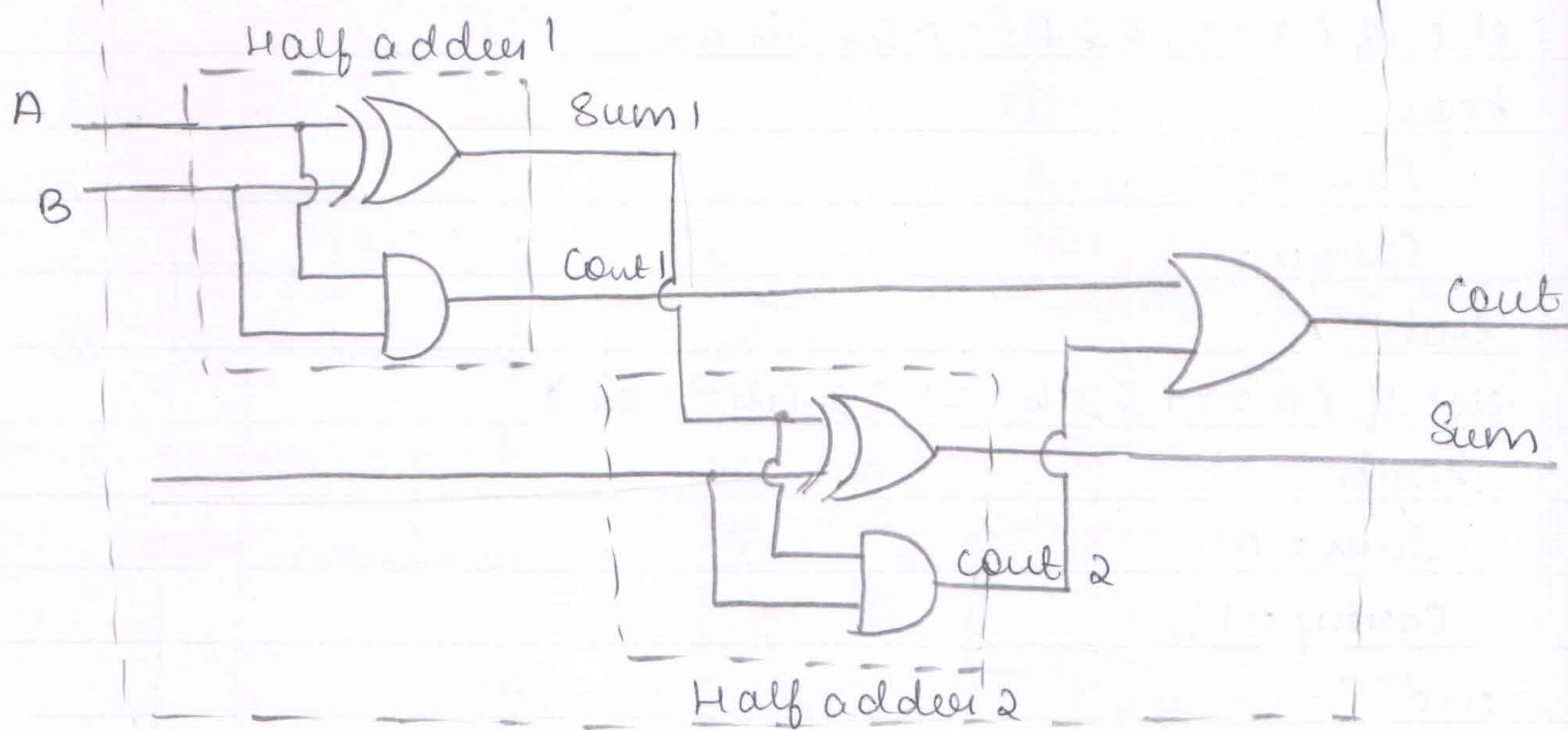
```
AND(R, a, b);
```

```
AND(S, b, c);
```

```
AND(Q, a, c);
```

```
OR(Y2, R, S, Q);
```

```
end Module
```

(iii)

Gate level:

Module HA(s, c, a, b);

I/P a, b;

O/P s, c;

assign s = a ^ b;

assign c = a & b;

end module

Module OR-2(y, x, z);

Input x, z;

output y;

assign y = x | z;

end module

Module fA(Sum, Cout, L, M, N);

Input L, M, N;

output Sum, Cout;

wire Sum1, Cout1, Cout2;

HA I1(Sum1, Cout1, L, M);

HA I2(Sum, Cout2, N, Sum1);

OR-2 I3(Cout, Cout1, Cout2);

end module.

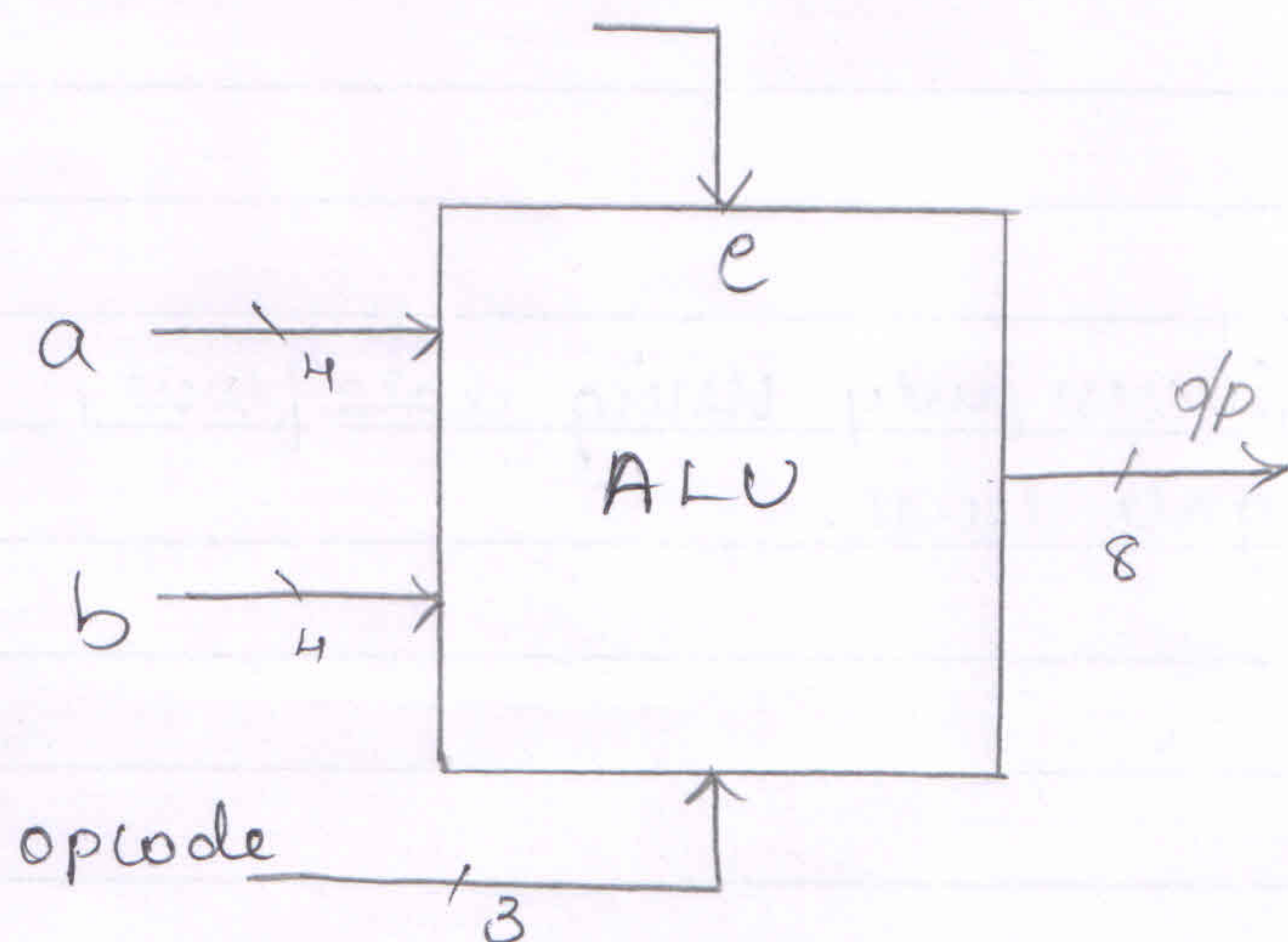
RESULT:

∴ The verification of functionality of full adder is done successfully

Recd - 10
Observation - 10
Total - 20

Final

BLOCK DIAGRAM!



Truth table:

Op code	Operation
0 0 0	Addition
0 0 1	Subtraction
0 1 0	Complement of 'a'
0 1 1	Multiplication
1 0 0	logical and
1 0 1	logical OR
1 1 0	logical NAND
1 1 1	logical XOR

AIM: To write a Verilog code to implement 4 bit ALU

TOOL USED: Xilinx ISE 13.1, Spartan 6.

Theory:

An arithmetic logic unit is a Combinational digital electronic circuit that performs arithmetic and bitwise operations on integer binary numbers.

This is in contrast to a floating-point unit (FPU), which operates on floating point numbers. An ALU is a fundamental building block of many types of computing circuits, including the central processing unit (CPU) of computers, FPUs, and graphical processing units (GPUs). A single CPU, FPU or GPU may contain multiple ALUs.

The inputs to an ALU are the data to be operated on, called operands, and a code indicating the operation to be performed, the ALU's output is the result of the performed operation. In many designs, the ALU also has status inputs or outputs, or both, which convey information about a previous operation or the current operation, respectively, between the ALU and external status registers.

Code:

```
Module ALU bit (a, b, opcode, e, y);  
  input [3:0] a, b;  
  input e;  
  output [7:0] y;  
  input [3:0] opcode;
```


Manual Calculations:

1. addition:

$$\begin{array}{r} 0101 \\ + 1010 \\ \hline 1111 \end{array}$$

2. Subtraction

$$\begin{array}{r} 0101 \\ - 1010 \\ \hline 1011 \end{array}$$

3. Complement

$$\begin{array}{r} \sim 0101 \\ \hline 1010 \end{array}$$

4. Multiplication:

$$\begin{array}{r} 0101 \times 1010 \\ \hline 0000 \\ 0101x \\ 0000xx \\ 0101xxx \\ \hline 0110010 \end{array}$$

5. AND

$$\begin{array}{r} 0101 \\ 1010 \\ \hline 0000 \end{array}$$

6. OR

$$\begin{array}{r} 0101 \\ 1010 \\ \hline 1111 \end{array}$$

7. NAND

$$\begin{array}{r} 0101 \\ 1010 \\ \hline 1111 \end{array}$$

8. XOR

$$\begin{array}{r} 0101 \\ 1010 \\ \hline 1111 \end{array}$$

Pin Configuration:

Inputs

a → P121

b → P123

c → P124

opcode → P126

Output:

P120 → Y[0]

P119 → Y[1]

P118 → Y[2]

P117 → Y[3]

P116 → Y[4]

P115 → Y[5]

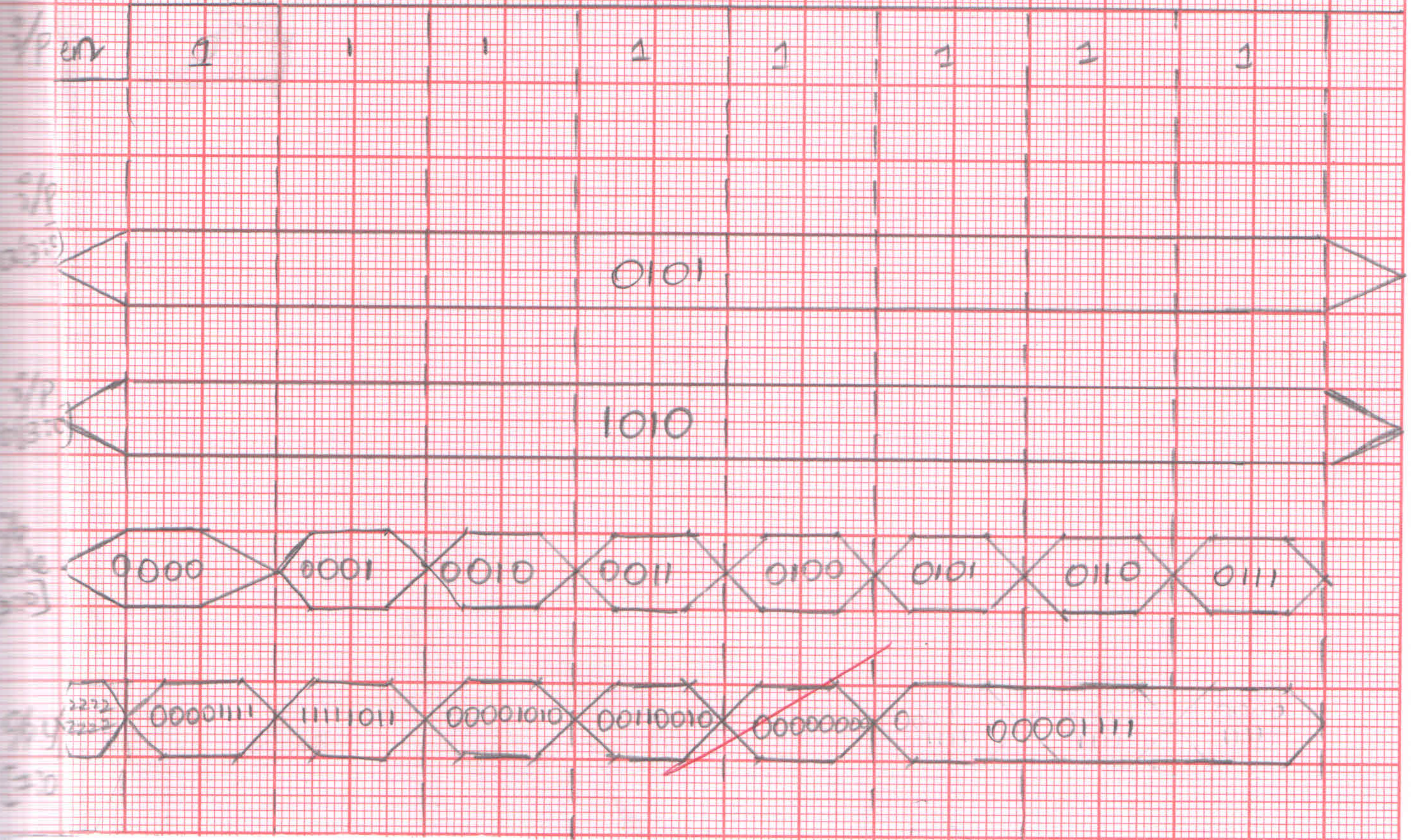
P114 → Y[6]

P112 → Y[7]

DAT

EXP

State
x axis : 2 unit = 16°E




```
reg [7:0] y;  
always @ (a, b, opcode, c)  
begin  
  if (c == 1)  
  begin case (opcode)  
    3'b000: y = a + b;  
    3'b001: y = a - b;  
    3'b010: y = {4'b0000, ~a};  
    3'b011: y = a * b;  
    3'b100: y = a & b;  
    3'b101: y = a | b;  
    3'b110: y = {4'b0000, ~(a & b)};  
    3'b111: y = a ^ b;  
    default: y = 8'b00000000;  
  end case  
end else  
  end  
begin y = 8'bzzzzzzzz;  
end  
end  
end Module.
```

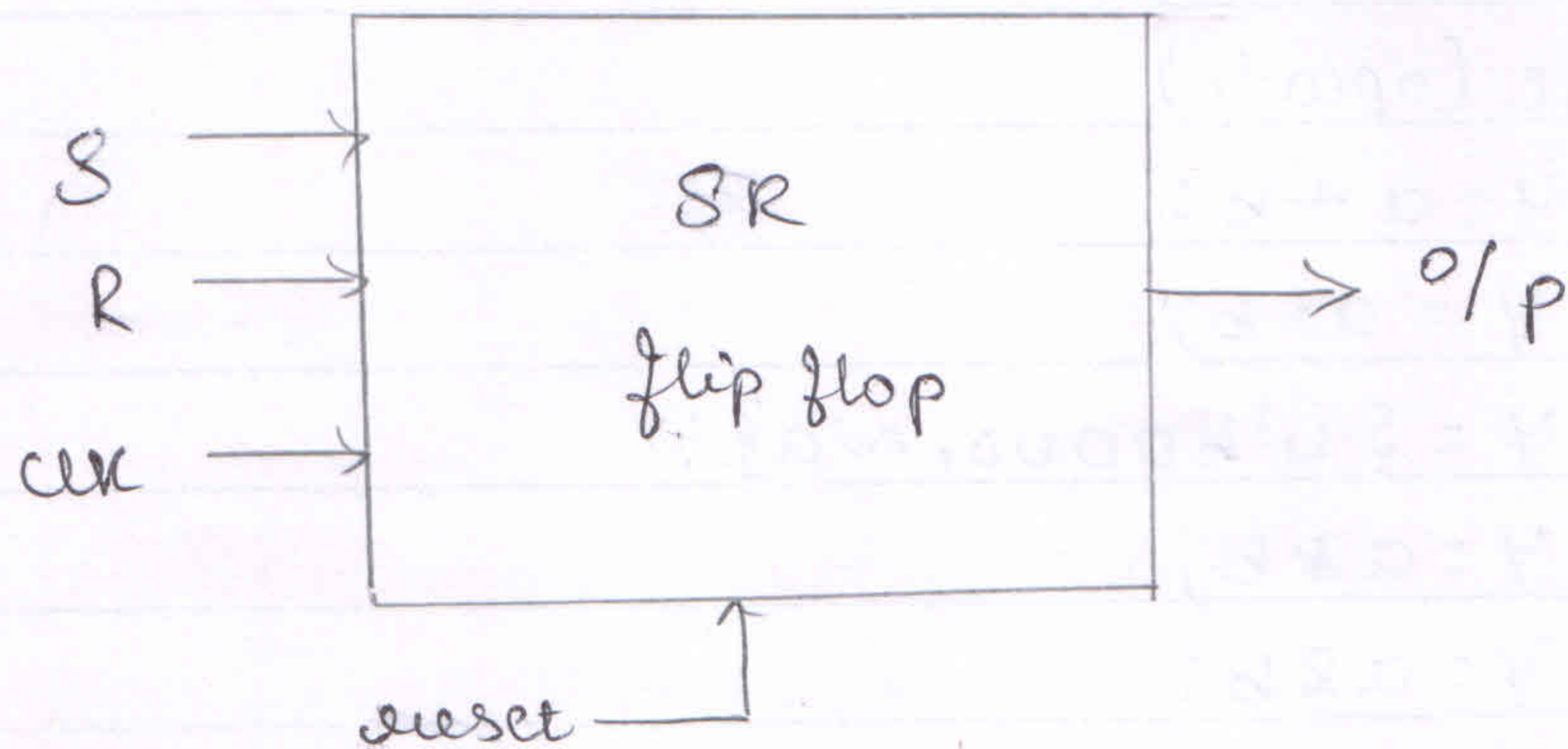
RESULT

∴ The verilog Code to implement 4 bit ALU is executed Successfully.

Record — 10
Observations — 10
Total — 20

26/9/18

Block diagram:



Truth table:

reset	clk	S	R	Q	Qb
1	0	x	x	0	1
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	invalid	

DATE ...19/9/18...

EXPT TITLE:

SR Flip flop

PAGE NO.

23

EXP. NO.

AIM: To write a verilog code to verify the function of SR flip flop and download it to Spartan 6.

TOOL USED: Xilinx 13.1, Spartan-6.

Theory:

The SR flip flop also known as SR Latch can be considered as one of the most basic sequential logic circuit possible. This simple flip flop is basically a one-bit memory bistable device that has two inputs, one which will set that device meaning the output = 1 and is labelled S, and one which will reset the device meaning the output = 0, labelled R. Then SR description stands for Set-Reset.

Code:

```
Module SRff ( S, R, clk, rst, q, qb );
```

```
  i/p  S, R, clk, rst
```

```
  o/p  q, qb;
```

```
  reg q = 0, qb;
```

```
  reg [1:0] sr;
```

```
  reg [31:0] clkdiv = 32'd0;
```

```
  always @ (posedge clk)
```

```
    clkdiv = clkdiv + 1;
```

```
  always @ (posedge clk)
```

```
  begin
```

```
    sr = { S, R };
```

```
    if (rst == 1)
```

```
      q = 0;
```

```
  else
```


Pin configuration :

P121 $\rightarrow$ γ s t

P123 $\rightarrow$ s

P124 $\rightarrow$ γ

P56 $\rightarrow$ clk

P119 $\rightarrow$ q

P120 $\rightarrow$ qb

1 0 1 0 1 0 1 0 1 0

Time Scale:

2 units = 1 η Sec

X 0 0 1 1

X 0 1 0 1

1 0 0 0 0

0 0 0 1 X

1 1 1 0 X

00 01 10 11

1 η Sec 2 η Sec 3 η Sec 4 η Sec 5 η Sec

SR FLIP FLOP

begin

case (8x)

2'b00 : q = 0;

2'b01 : q = 0;

2'b10 : q = 1;

default : q = q;

end case

end

qb = ~q;

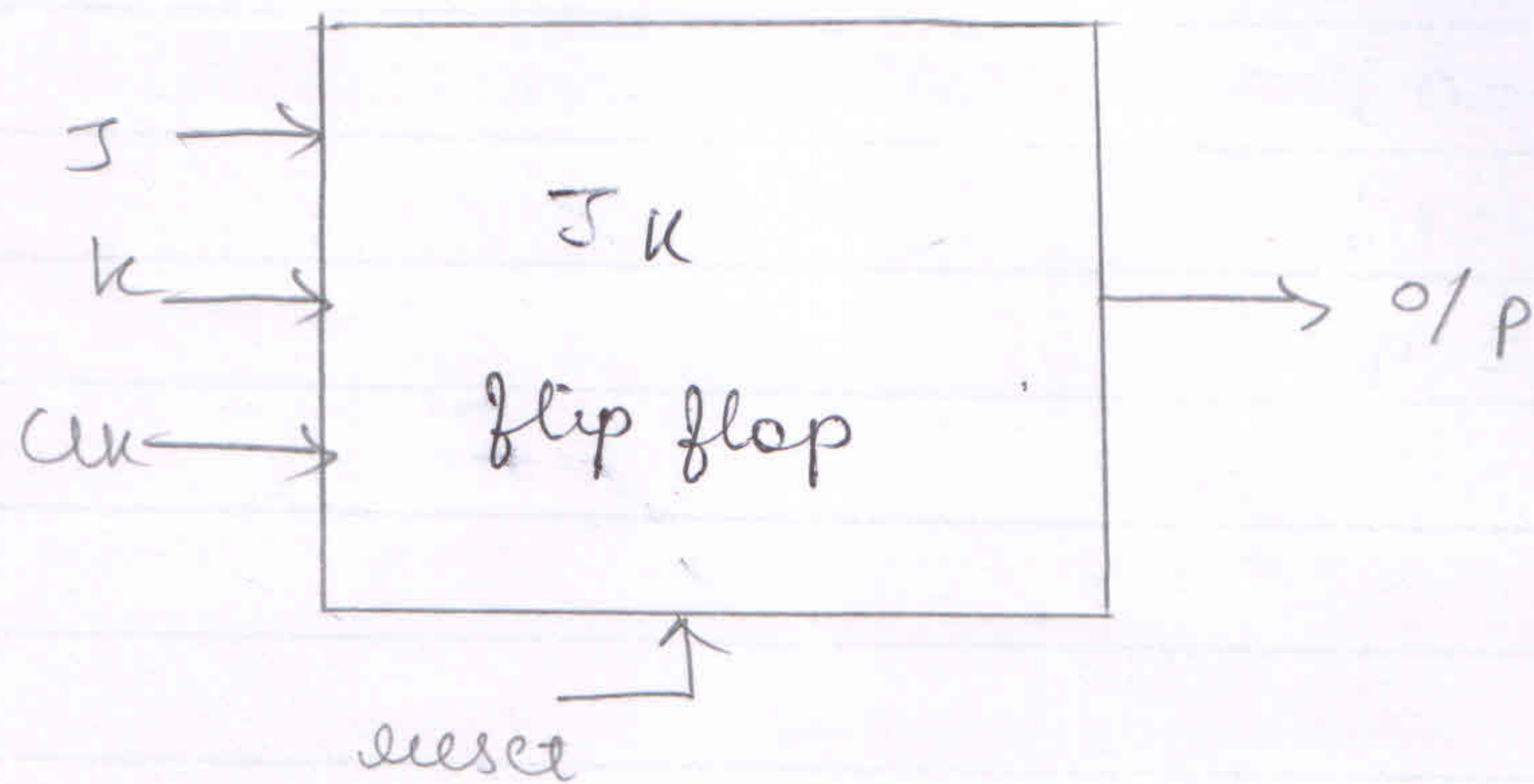
end

end module

Result :

SR flip flop is verified ~~Successfully~~ using Verilog code.

Block diagram:



Truth table:

reset	clk	J	K	q	qb
1	0	x	x	0	1
0	0	0	0	q	qb
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	Toggle.	

AIM: To write a verilog code to verify the function of JK flip flop and download it to the kit.

Tool used: Xilinx 13.1, Spartan-6

Theory:

The JK flip flop is most versatile of the basic flip flop. It has the input following characters of clocked D-flip flop but has two inputs. Traditionally labelled J and K. If J and K are different then the output Q takes the value of J at the next clock edge. The inputs are labelled J and K in honor of the inventor of the device Jack Kilby. If J and K are low then no change occurs. If J and K is high then output will toggle.

Code:

```
Module JKff (J, K, reset, Q, Qb, clk);  
input J, K, reset, clk;  
output Q, Qb;  
reg Q = 0, Qb;  
reg [1:0] JK;  
reg [31:0] clkdiv = 32'd0;  
always @ (posedge clk)  
    clkdiv = clkdiv + 1;  
always @ (posedge clkdiv[24])  
begin  
    JK = {J, K};  
    if (reset == 1)
```


Pin configuration:

J → P121

P120 → Gb

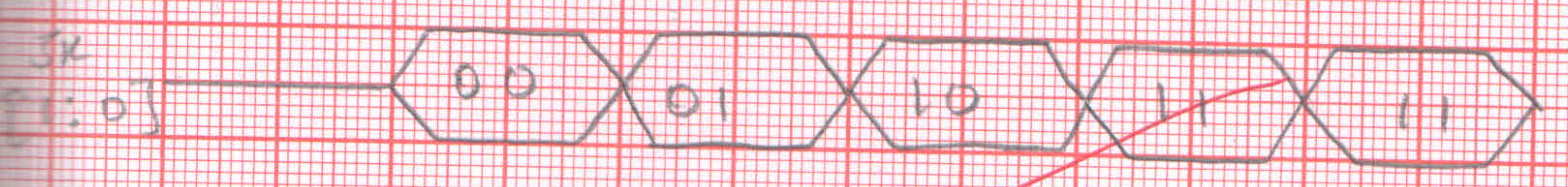
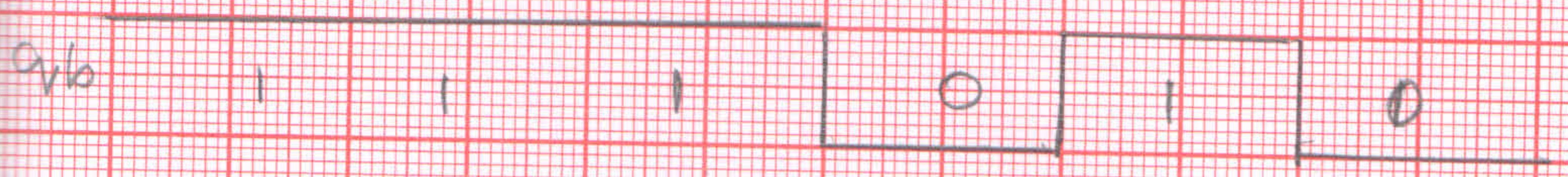
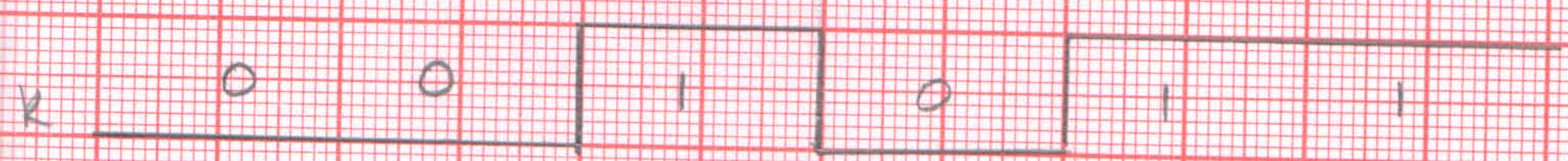
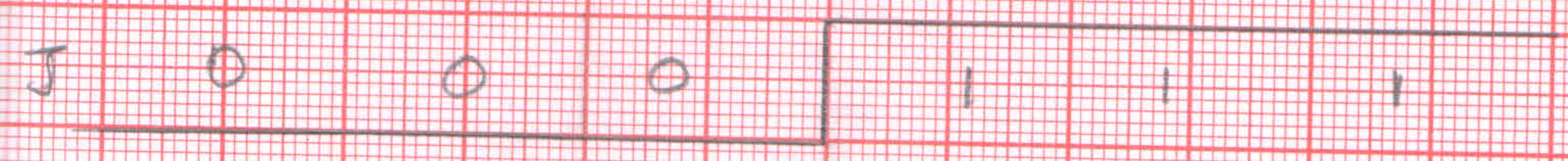
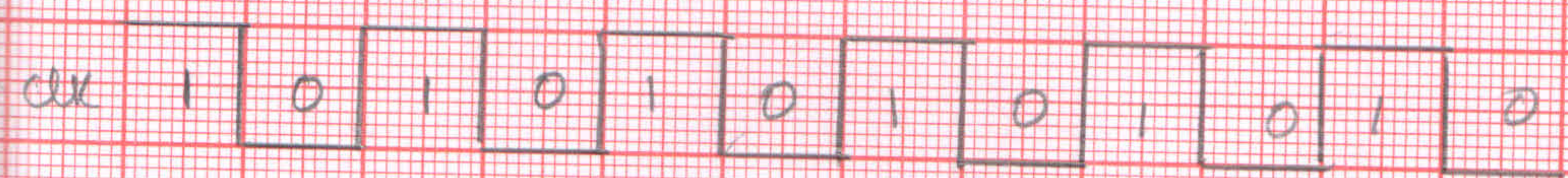
P56 → CLK.

K → P123

P119 → G

88E → P124

Scale
2 units = 1 μ Sec



JK flip flop


```
q = 0;
```

```
else
```

```
begin
```

```
case (JK)
```

```
2'b00 : q = q;
```

```
2'b01 : q = 0;
```

```
2'b10 : q = 1;
```

```
2'b11 : q = ~q;
```

```
default : q = q;
```

```
end case
```

```
end
```

```
qb = ~q;
```

```
end
```

```
end Module
```

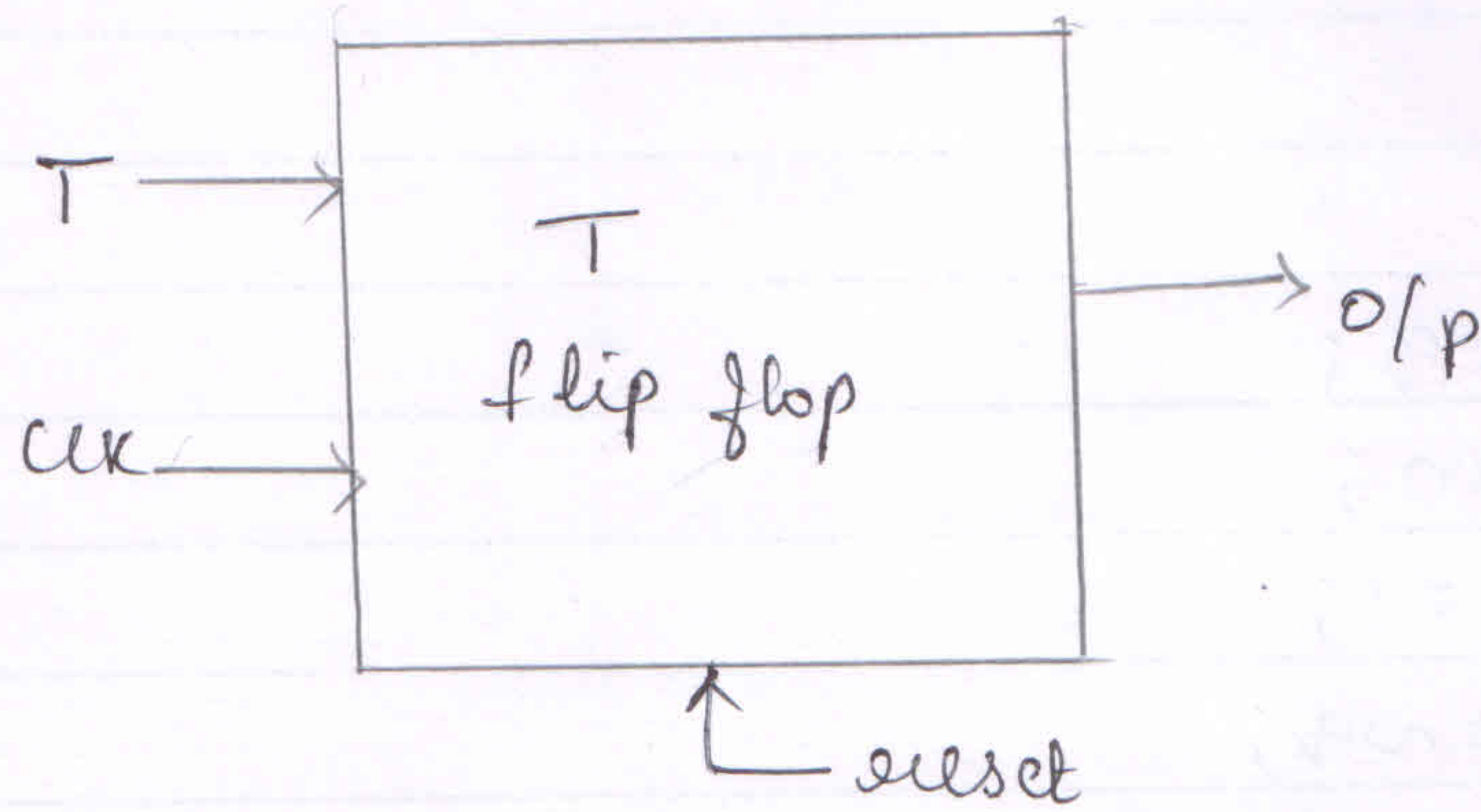
RESULT:

JK flipflop is verified ~~successfully~~ using Verilog Code.

Record — 10
Observations — 10
Total — 20

[Signature]

Block diagram:



Truth table:

reset	clk	T	q	q'
1	0	X	q	q'
0	0	0	q	q'
0	0	1	q'	q

AIM: To write a verilog code to verify the function of T-flip flop and download to the kit.

TOOL USED: Xilinx 13.1, Spartan 6

Theory:

The T has 'toggle' flip flop changes its output on each clock edge, giving an output which is half the frequency of the signal to the T input. It is useful for constructing binary counters, frequency dividers, and generate binary addition device. It can be made from J-K flip flop by tying both of the inputs high. According to the table, based on the input the output changes its states.

Code:

```
Module Tff(T, clk, rst, q, qb);  
input T, clk, rst;  
output q, qb;  
reg q = 0, qb;  
reg [31:0] clkdiv = 32'd0;  
always @ (posedge clk)  
    clkdiv = clkdiv + 1;  
always @ (posedge clkdiv[24])  
begin  
    if (rst == 1)  
        q = 0;  
    else if (t == 1)
```


Pin Configuration :

P121 $\rightarrow$ rst

P119 $\rightarrow$ q

P123 $\rightarrow$ t

P120 $\rightarrow$ qb

P56 $\rightarrow$ clk

DATE

EXP. NO.

T Flip flop

ax 1 0 1 0 1 0 1 0

Scale:

2 units = 1 nSec

enst 1 0 0 0

t 0 0 1 1

q 0 0 1 0

q/b 1 1 0 1

1 nSec 2 nSec 3 nSec

$q = \sim q;$

else

$q = q;$

$qb = \sim q;$

end Module

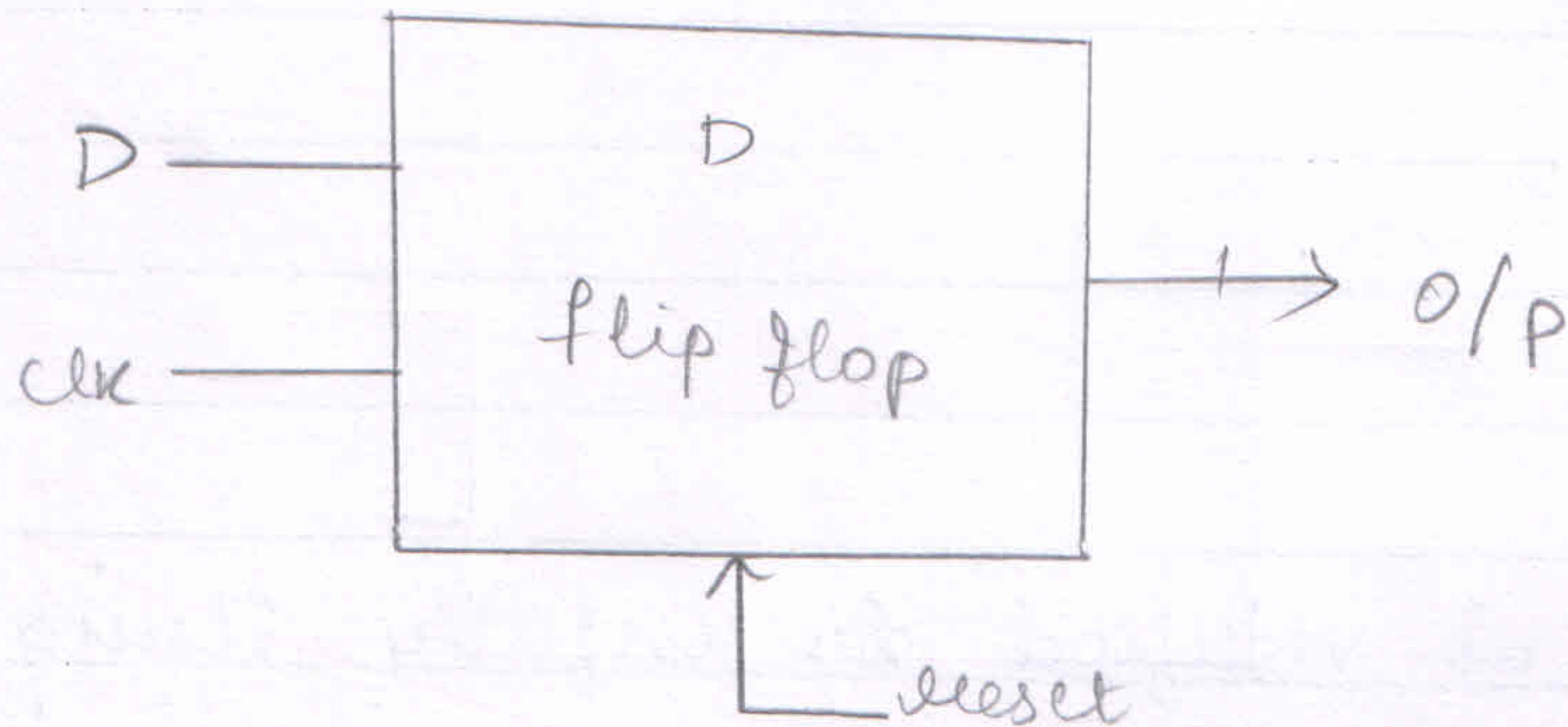
RESULT:

T flip flop is verified successfully using Verilog Code.

Revised - 10
observation - 10
Total - 20

[Signature]

Block diagram:



Truth table:

reset	clk	D	q	qb
1	0	X	0	1
0	0	0	0	1
0	1	1	1	0

AIM: To write a verilog code to verify the functionality of D-flip flop and download to Spartan 6 kit

TOOL USED: Xilinx 13.1, Spartan 6

Theory:

The D-flip flop track the input, making transitions with match those of the input D. The D stands for 'data'. This flip flop stores the value that is on the data line. It can be thought of a 1 and 0 memory cell. A D flip flop can be made from SR flip flop by tying the Set to the reset through an inverter. The result may be clocked, the output it will the input D is long as the flip flop remains enabled. The D-flip flop can be connected from a latch.

Code:

```
Module Dff (D, clk, rst, q, qb);
```

```
input D, clk, rst;
```

```
output q, qb;
```

```
reg q = 0, qb;
```

```
reg [31:0] clkdiv = 32'd0;
```

```
always @ (posedge clk)
```

```
clkdiv = clkdiv + 1;
```

```
always @ (posedge clk)
```

```
begin
```

```
if (rst == 1)
```

```
q = 0;
```


Pin Configuration :

P121 $\rightarrow$ reset

P56 $\rightarrow$ clk.

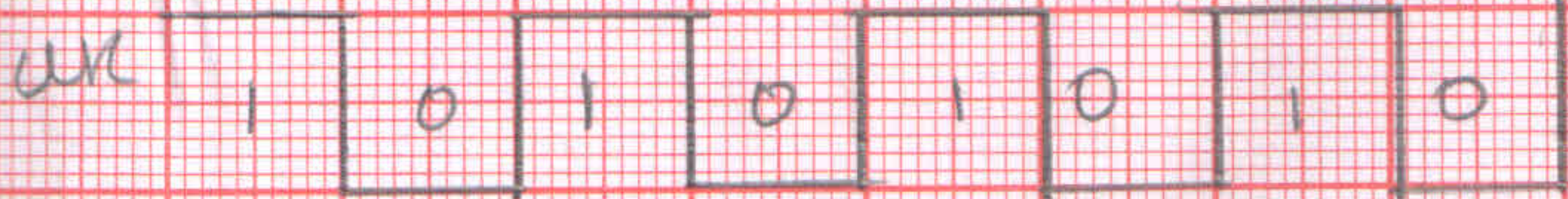
P123 $\rightarrow$ d

P119 $\rightarrow$ q

P120 $\rightarrow$ qb

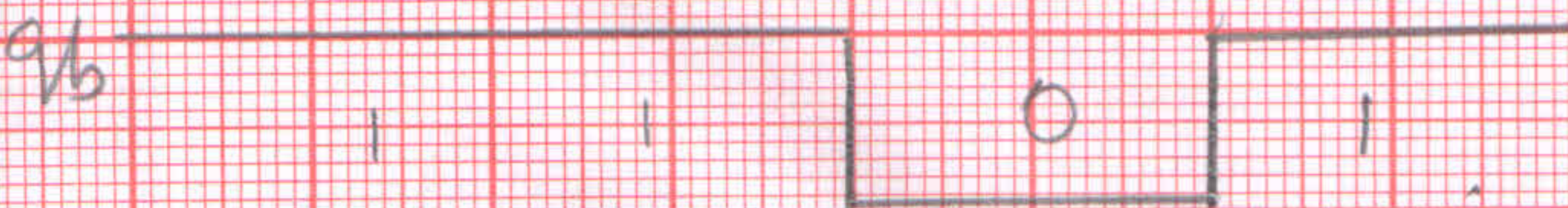
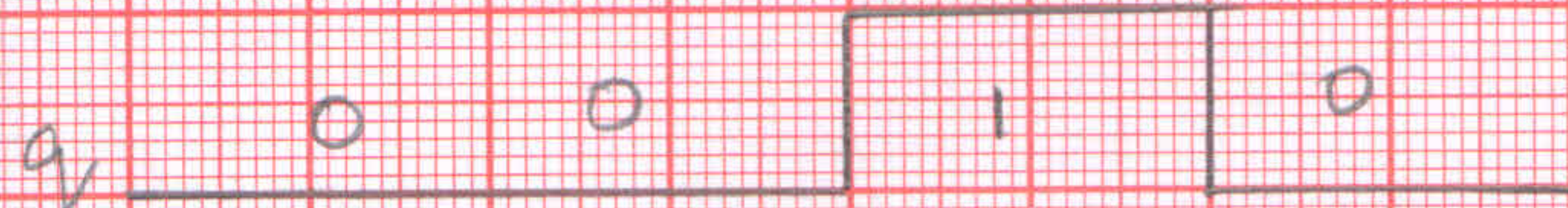
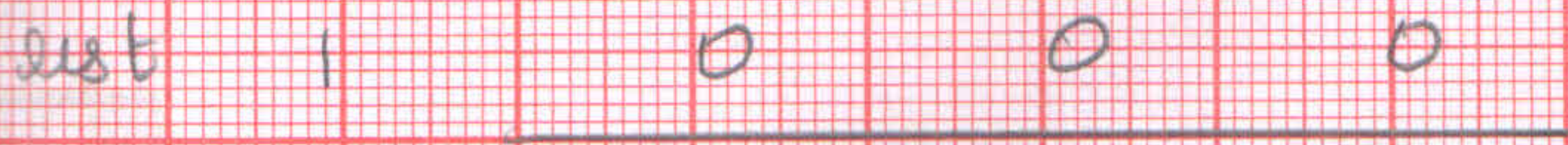
DATE .

EXP. NO



Scale:

2 units = 1 μ sec



1 μ sec

2 μ sec

3 μ sec

4 μ sec

D Flip flop

else

$q = d;$

end

$qb = \sim q;$

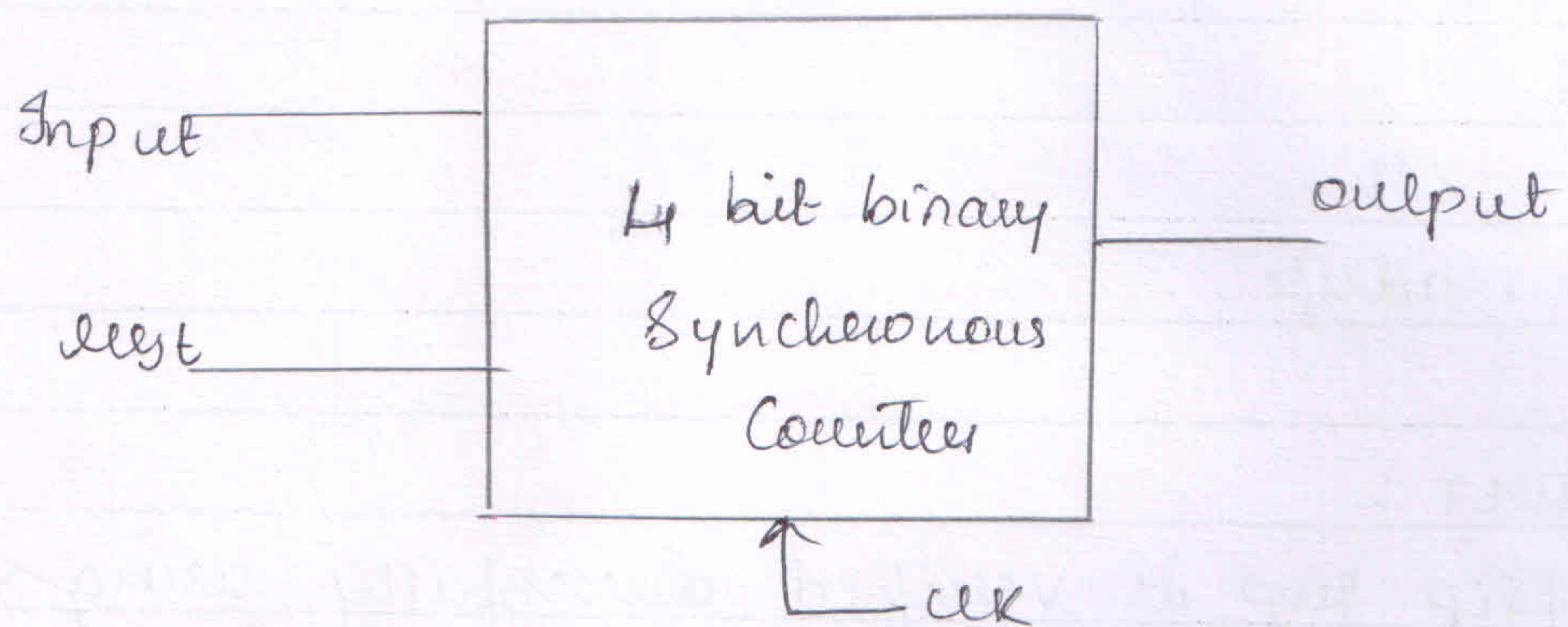
end module.

RESULT :

D flip flop is verified successfully using verilog
Code.

Record - 10
Observation - 10
Total - 20

Praveen
29/9/18



Truth table :

i/p	reset	q ₀	q ₁	q ₂	q ₃
x	1	0	0	0	0
↑	0	0	0	0	0
↑	0	0	0	0	1
↑	0	0	0	1	0
↑	0	0	0	1	1
↑	0	0	1	0	0
↑	0	0	1	0	1
↑	0	0	1	1	0
↑	0	0	1	1	1
↑	0	1	0	0	0
↑	0	1	0	0	1
↑	0	1	0	1	0
↑	0	1	0	1	1
↑	0	1	1	0	0
↑	0	1	1	0	1
↑	0	1	1	1	0
↑	0	1	1	1	1

AIM: To write a Verilog code to verify the functionality of 4 bit binary Synchronous Counter.

TOOL USED: Xilinx 13.1, Spartan 6.

Theory:

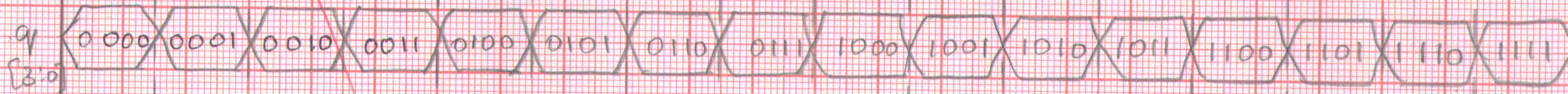
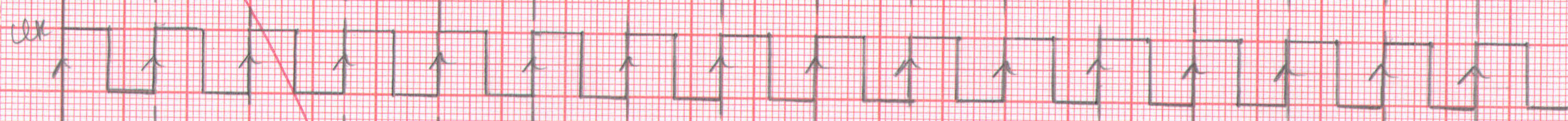
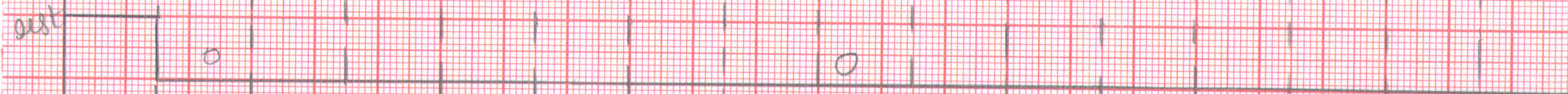
Synchronous Counters can be made from toggle or D-type flip flops. Synchronous Counters are easier to design than asynchronous Counters. They are called Synchronous Counters because the clock input of the flip flops are connected all clocked together at the same time with the same clock signal. Due to this common clock pulse all output states switch or change simultaneously. With all clocks input wired together there is no inherent propagation delay.

Synchronous Counters are sometimes called parallel Counters as the clock is fed in parallel to all flip-flops.

The inherent memory circuit keeps track of the Counter's present state. The Count Sequence is controlled using logic gates. Overall faster operation may be achieved compared to asynchronous Counters.

Code:

```
Module Counter (clk, en, q);  
output [3:0] q;  
input clk, en;  
reg [3:0] q = 4'd0;  
reg [31:0] clkdiv = 32'd0;  
always @(posedge clk)  
    clkdiv = clkdiv + 1;
```

4 bit Synchronous Counter

always @ (posedge clk)

begin

if (en == 1)

q = 4'd0;

else

q = q + 1;

end

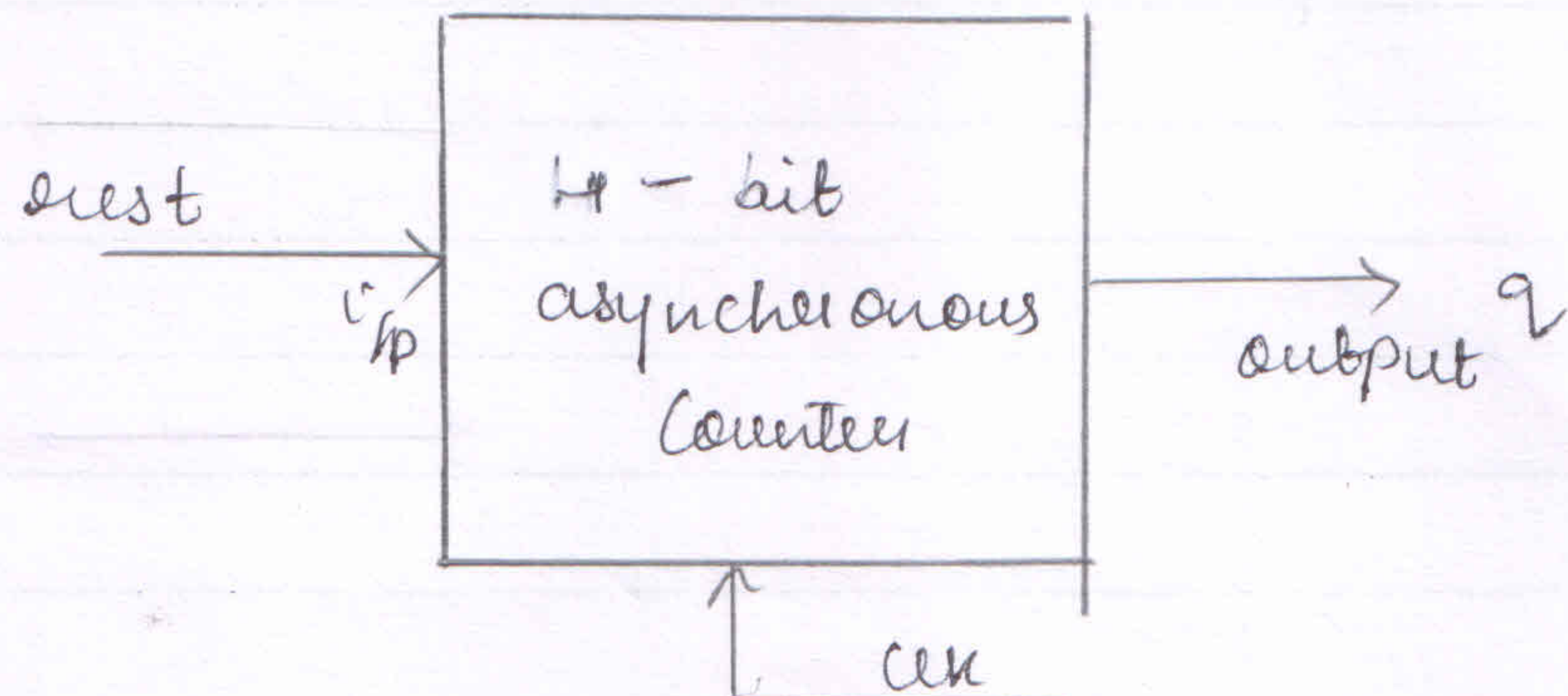
end module

RESULT:

The verification of 4 bit binary synchronous counter is completed.

~~80~~
14/11

Record - 10
Observation - 10
Total - 20



clock	reset	q ₀	q ₁	q ₂	q ₃
x	0	0	0	0	0
↑	1	0	0	0	0
↑	1	0	0	0	1
↑	1	0	0	1	0
↑	1	0	0	1	1
↑	1	0	1	0	0
↑	1	0	1	0	1
↑	1	0	1	1	0
↑	1	0	1	1	1
↑	1	1	0	0	0
↑	1	1	0	0	1
↑	1	1	0	1	0
↑	1	1	0	1	1
↑	1	1	1	0	0
↑	1	1	1	0	1
↑	1	1	1	1	0
↑	1	1	1	1	1

AIM: To write a verilog code to check the functionality of asynchronous counter

TOOL USED: Xilinx 13.1, Spartan 6.

THEORY:

Counters, consisting of a number of flip flops, count a stream of pulses applied to the counter's clock input.

The output is a binary value whose value is equal to the numbers of pulses received at the clk input.

Each output represents one bit of the output word, and the size of the output word depends on the number of flip-flops that make up the counter.

Asynchronous counters are those whose output is free from the clock signal. Because the flip flop in asynchronous counters are supplied with different clock signals, there may be delayed in producing output.

Code:

```
Module Counter (clk, en, q);
```

```
output [3:0] q;
```

```
input clk, en;
```

```
reg [3:0] q = 4'd0;
```

```
reg [31:0] clkdiv = 32'd0;
```

```
always @ (posedge clk)
```

```
    clkdiv <= clkdiv + 1;
```

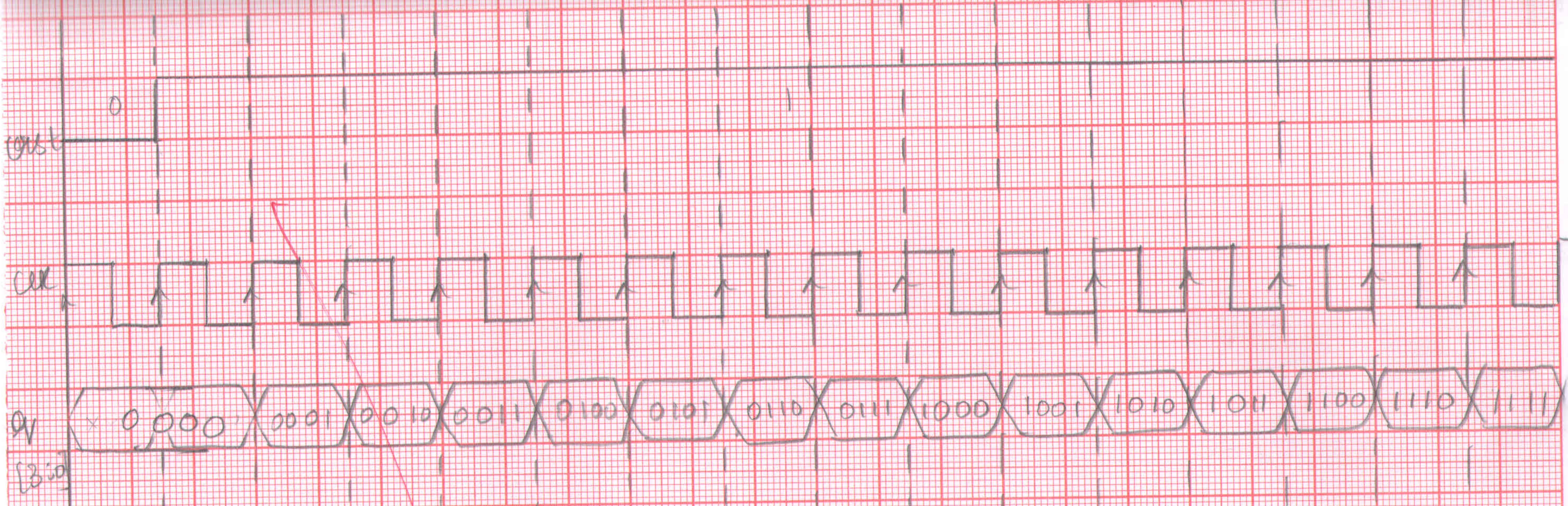
```
always @ (posedge clk, posedge en)
```

```
begin
```

```
    if (en == 1)
```


Scale: x axis: 1.5 cm = 1 μ s
 y axis: 1 cm = 1V

DATE



4-bit asynchronous
 binary counter

$q = q + 1;$

else

$q = H'do;$

end

end Module.

RESULT :

Verification of 4 bit binary asynchronous counter is completed.

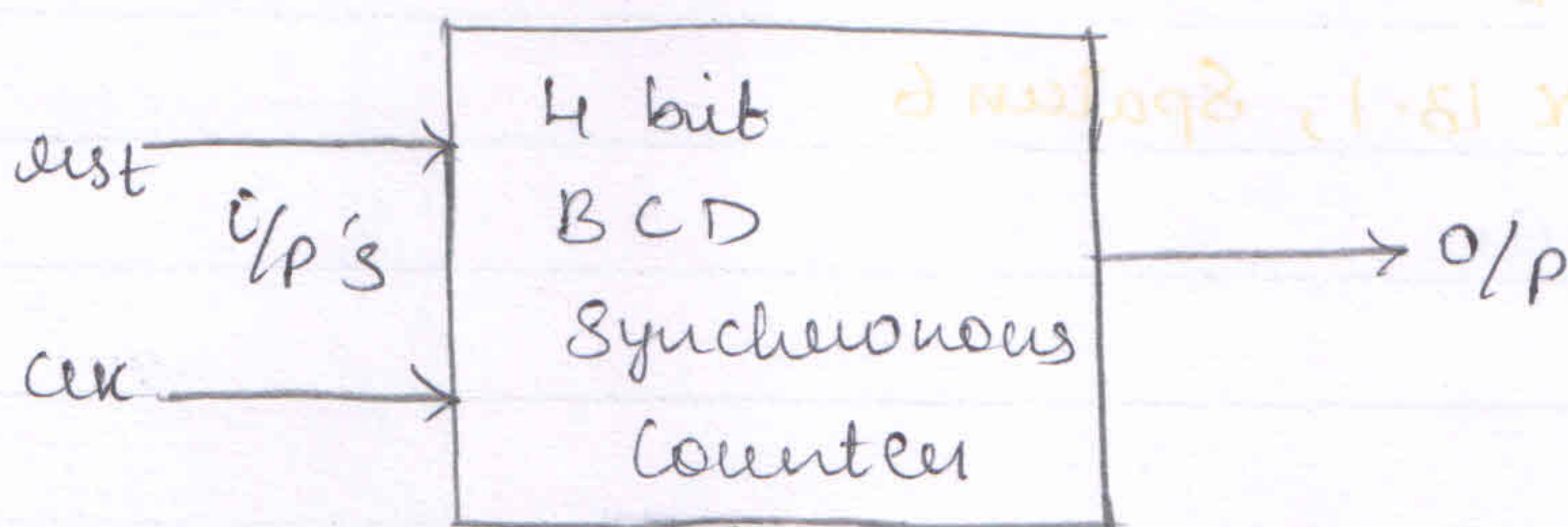
Pa
14/11

Record - 10
Observation - 10
Total - 20

4 bit Synchronous BCD Counter

Aim: To write a Verilog code to check the functionality of 4 bit Synchronous BCD Counter.

Block diagram:



Truth table

i/p reset	q ₀	q ₁	q ₂	q ₃
x 1	0	0	0	0
0	0	0	0	0
0	0	0	0	1
0	0	0	1	0
0	0	0	1	1
0	0	1	0	0
0	0	1	0	1
0	0	1	1	0
0	0	1	1	1
0	0	0	0	0
0	1	0	0	1

AIM: To write a Verilog Code to check the functionality of 4 bit binary asynchronous BCD Counter.

TOOL USED:

Xilinx 13.1, Spartan 6.

Theory:

A decade Counter counts in a sequence of ten and then returns back to zero after the count of nine. Obviously to count up to a binary value of nine, the counter must have at least of four flip flops within its chain to represent each decimal digit.

Then a decade counter has 4 flip flops and 16 potential states, of which only 10 are used and the total number of counts that a counter can count too is called its modulus. A counter that returns to zero after n counts is called a Modulo- n -Counter.

A counter which resets after ten counts with a divide by 10 counts sequence from binary 0000 (decimal "0") through to 1001 (decimal "9") is called a "binary coded decimal counter". A BCD counter for short and a MOD-10 counter can be constructed using a minimum of four toggle flip flops.

Code:

```
Module Counter (clk, en, q);
```

```
Output [3:0] q;
```

```
input clk, en;
```

```
reg [3:0] q = 4'd0;
```

```
reg [31:0] clkdiv = 32'd0;
```


x -axis:

clk



5.0]9, 0000, 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001


```
always @(posedge clk)
    clkdiv = clkdiv + 1;
always @(posedge clk)
begin
    if (en == 1)
        q = 4'd0;
    else
begin
        if (q != 4'd0) q = q + 1;
        else
            q = 4'd0;
        end
    end
end
end module.
```

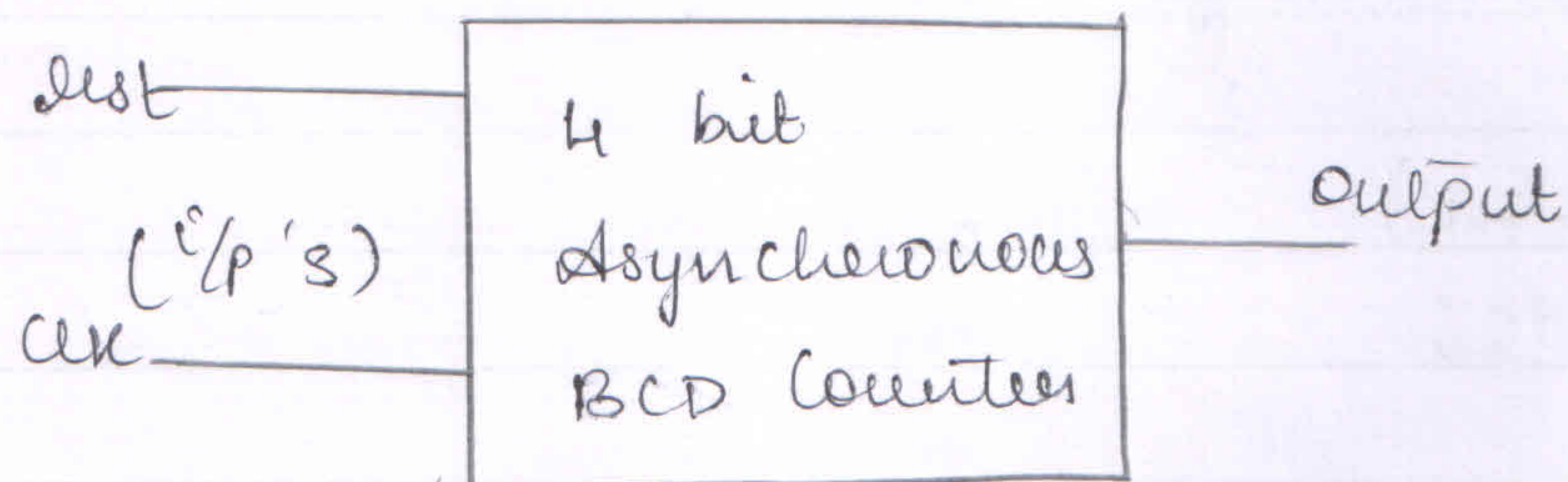
Result :

The Verification of the 4 bit BCD Synchronous Counter is completed

80
14/11

Record - 10
Observation - 10
Total - 20

Block diagram:



Truth table:

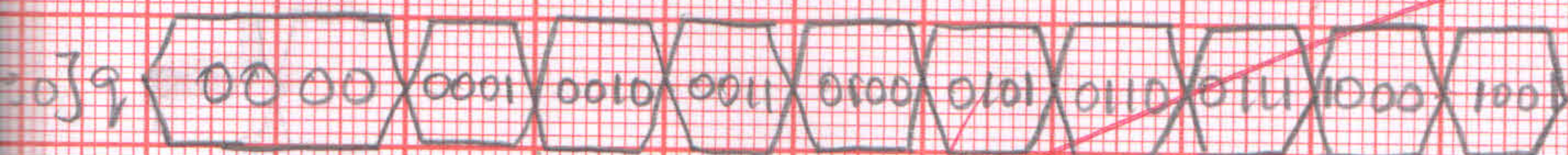
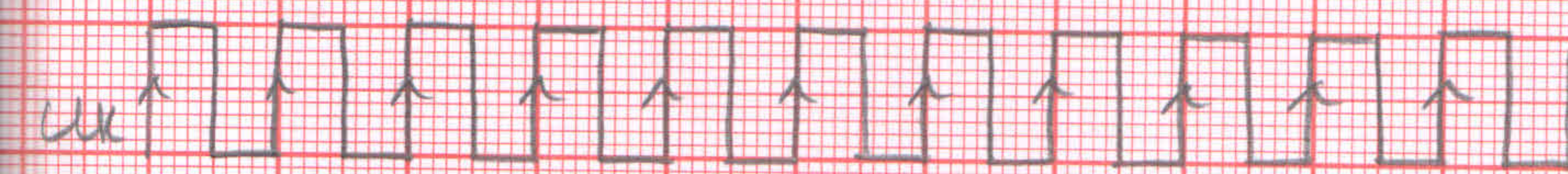
clk	en	Q ₃	Q ₂	Q ₁	Q ₀
X	0	0	0	0	0
↓	0	0	0	0	0
↓	0	0	0	0	1
↓	0	0	0	1	0
↓	0	0	0	1	1
↓	0	0	1	0	0
↓	0	0	1	0	1
↓	0	0	1	1	0
↓	0	0	1	1	1
↓	0	1	0	0	0
↓	0	1	0	0	1

en 1 0 0 0 0 0 0 0 0 0 0

Scale:

X-axis:

1 Unit: 1 bit



Program - 17

AIM: To write a Verilog code to verify the function of 4-bit BCD asynchronous Counter.

TOOLS USED: Xilinx 13.1
SPATRN 6

Theory:

The output of the asynchronous counters do not depend on the clock signal.

Asynchronous counters are also called as ripple counters. The output changes with respect to the enable signal.

The BCD asynchronous counter has 10 valid output states from 0000 to 1001.

Code:

```
Module BCD_async_Counter (clk, en, q);  
    output [3:0] q;  
    input clk, en;  
    reg [3:0] q = 4'b0;  
    reg [31:0] clkdiv = 32'd0;  
    always @ (posedge clk)  
        clkdiv = clkdiv + 1;  
    always @ (posedge clk, posedge en) en  
    begin  
        if (~rst || q == 4'b1001)
```


DATE

EXPT TITLE:

EXP. NO.

PAGE NO.

38

 $q = 4'd0$

else

 $q = q + 1$;

end

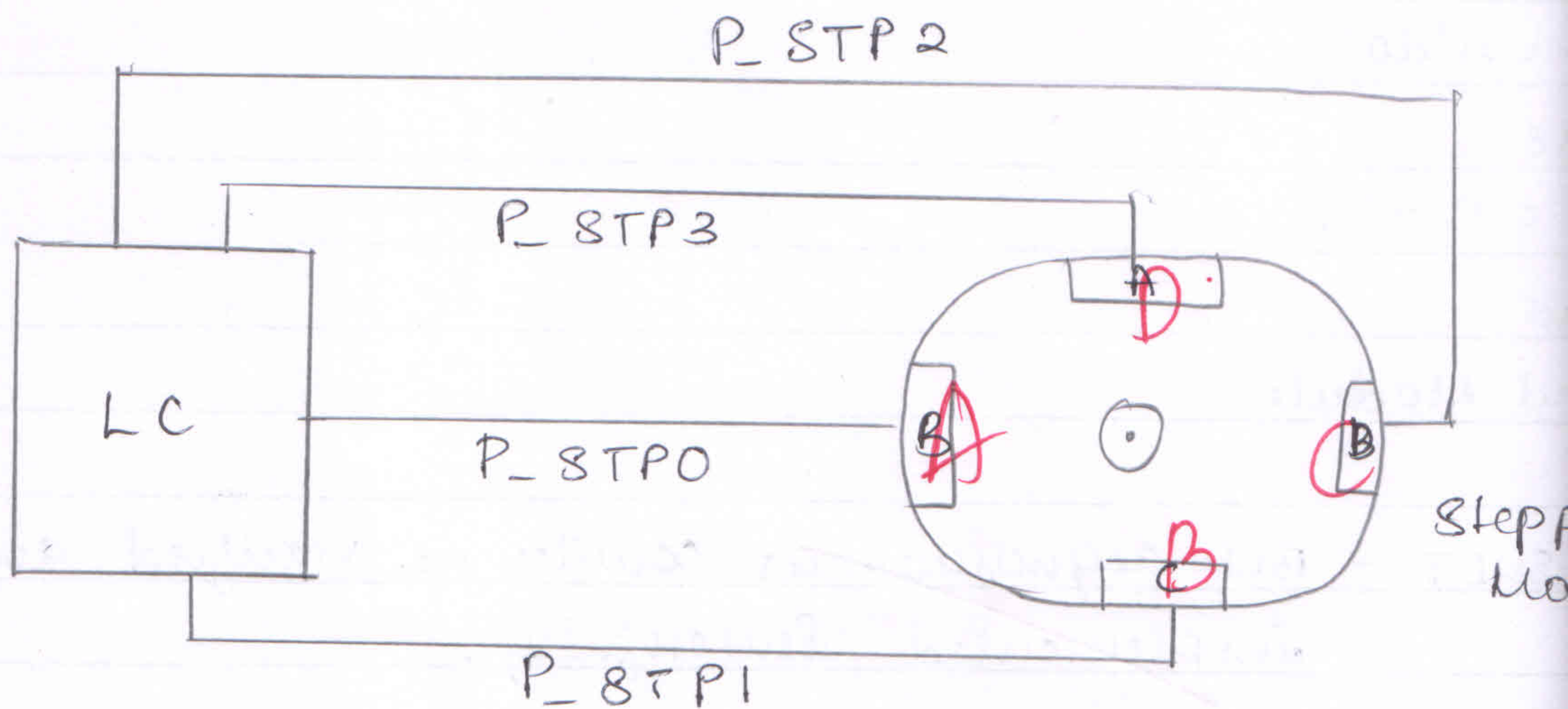
end module

RESULT - BCD asynchronous counter is verified and implemented successfully.

89
14/4

Record - 10
Observation - 10
Total - 20.

BLOCK DIAGRAM:



STEPPER MOTOR.

NET "P-STP(0)" LOC = P29;

NET "P-STP(1)" LOC = P30;

NET "P-STP(2)" LOC = P32;

NET "P-STP(3)" LOC = P33;

Main Clock Line

NET "P_Clk" LOC = P56;

NET "dclk" LOC = P121;

DATE ..31/10/18
EXP. NO.18

EXPT TITLE: STEPPER MOTOR

PAGE NO.

39

AIM: To write the verilog code to verify the working of a Stepper Motor.

TOOL USED: Xilinx 13.1, Spartan-6, Stepper motor, power supply

PROGRAM:

```
Module MOTOR - STEPPER1 (
```

```
input P_Clk,
```

```
output reg [3:0] P_STP, // DAC out
```

```
input dir,
```

```
);
```

```
reg [20:0] Count = 21'd0;
```

```
reg [3:0] Stpval = 4'b0001;
```

```
always @(posedge P_Clk)
```

```
begin
```

```
Count = Count + 1;
```

```
if (Count == 21'd10000000) // change this value for Speed  
begin
```

```
Count = 0;
```

```
P_STP = Stpval;
```

```
if (dir == 0)
```

```
begin
```

```
Stpval = Stpval << 1;
```

```
if (Stpval == 4'b0000) Stpval = 4'b0001;
```

```
end
```

```
else
```

```
begin
```

```
Stpval = Stpval >> 1;
```

```
if (Stpval == 4'b0000) Stpval = 4'b1000;
```

```
end
```

```
end
```

```
end
```

```
endmodule
```

14/11

Record - 10
Observation - 10
Total - 20

#DCMOTOR

NET "P_DCM(0)" LOG = P82;

NET "P_DCM(1)" LOG = P83;

NET "P_DCM(2)" DCMEN LOG = P84;

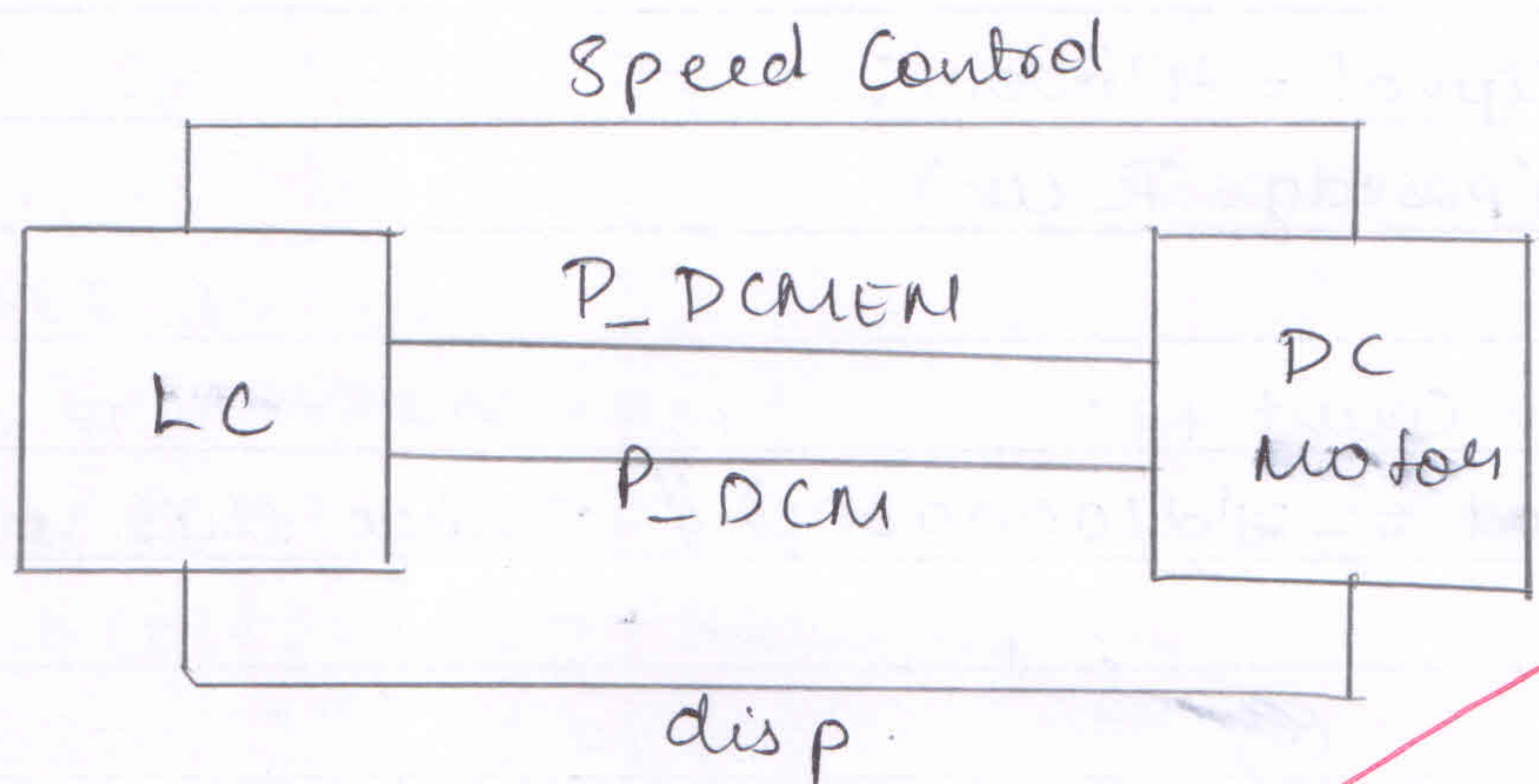
Main Clock line

NET "P_Clk" LOG = P56;

NET "dis" LOG = P121;

NET "Speed - Control" LOG = P123;

↓ BLOCK DIAGRAM:



DATE 03/10/18
EXP. NO. 19

EXPT TITLE:

DC MOTOR.

PAGE NO.

20

Aim : To write a verilog code to verify the interfacing of dc motor

TOOLS USED : Xilinx 13.1, Spartan-6, dc motor, power supply

code :

Module MOTOR_DC (

input P_Clk,

output [1:0] P_DCM, // DC Motor dir control.

output P_DCEN, // DC Motor enable PWM

input dir,

input Speed Control

);

reg [19:0] Count = 20'd0;

wire High-Speed, Low-Speed;

always @ (posedge P_Clk)

Count = Count + 1;

assign P_DCEN = Speed_Control ? High-Speed : Low-Speed;

assign High-Speed = (Count < 20'HFFFFF)? 1:0;

assign Low-Speed = (Count < 20'H3FFFF)? 1:0;

assign P_DCM = (dir) ? 2'b01 : 2'b10;

end module

THEORY

A DC motor is any of a class of rotary electrical machines that convert direct current electrical energy into mechanical energy. The most common types rely on the forces produced by magnetic fields.

RESULT : DC Motor Verified Successfully.

Record - 10
Observation - 10
Total - 10

Da
16/11

~~##~~ ADC

NET "P_ADC(0)" LOC=P12;
NET "P_ADC(1)" LOC=P14;
NET "P_ADC(2)" LOC=P15;
NET "P_ADC(3)" LOC=P16;
NET "P_ADC(4)" LOC=P17;
NET "P_ADC(5)" LOC=P21;
NET "P_ADC(6)" LOC=P22;
NET "P_ADC(7)" LOC=P23;
NET "ADC_WR" LOC=P24;
NET "ADC_INT" LOC=P26;

~~#~~ main clock line

NET "P_Clk" LOC=P56;

DATE ...31/10/18...

EXPT TITLE:

ADC POT :

PAGE NO.

41

AIM: To write a verilog code to verify the interfacing of ADC pot

TOOL USED: Xilinx 13.1, Spartan-6,

code:

```
Module ADC_POT(
```

```
    input P_Clk,
```

```
    input [7:0] P_ADC, // ADC Data
```

```
    input ADC_INT, // ADC INT
```

```
    output reg ADC_WR, // ADC WR
```

```
    output reg [7:0] P_LED
```

```
);
```

```
reg [4:0] count = 0;
```

```
reg await = 0;
```

```
always @ (posedge P_Clk)
```

```
begin
```

```
    if (count == 0)
```

```
    begin
```

```
        ADC_WR = 0;
```

```
        await = 1;
```

```
    end
```

```
    if (count == 5)
```

```
    begin
```

```
        ADC_WR = 1;
```

```
        await = 0;
```

```
    end
```

```
    if (await == 0)
```

```
    begin
```

```
        if (ADC_INT == 0)
```

```
        begin
```

```
            P_LED = P_ADC;
```

```
            await = 1;
```


end

end

if (await == 1) Count = Count + 1;

if (Count == 10) Count = 0;

end

end module.

THEORY:

An analog-to-Digital Converter (ADC, A/D or A-to-D) is a system that converts an analog signal, such as sound, etc. into a digital signal.

Typically, the digital output is a two's complement binary number that is proportional to the input, but there are other possibilities.

RESULT:

ADC POT is implemented and tested successfully.

89
14/11

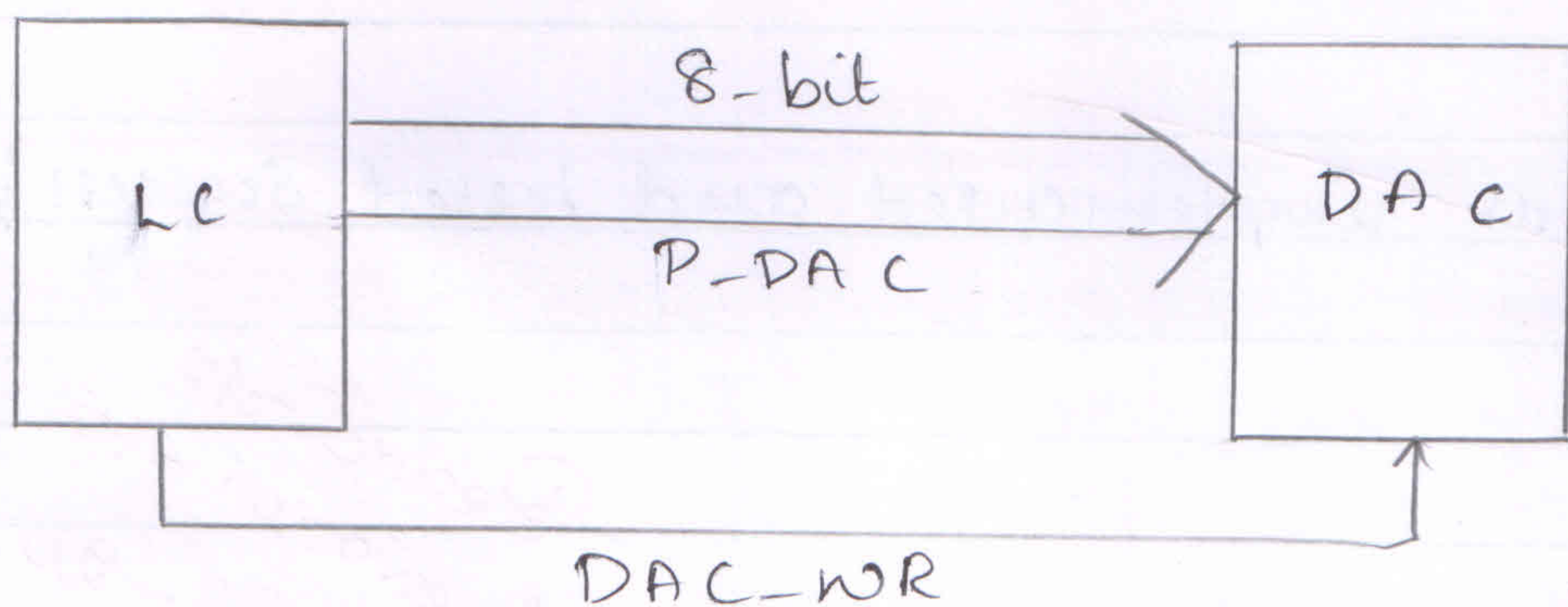
Record - 10
Observation - 10
Total - 20

#DAC

NET "P_DAC(0)" LOC = P66;
NET "P_DAC(1)" LOC = P67;
NET "P_DAC(2)" LOC = P65;
NET "P_DAC(3)" LOC = P74;
NET "P_DAC(4)" LOC = P75;
NET "P_DAC(5)" LOC = P78;
NET "P_DAC(6)" LOC = P79;
NET "P_DAC(7)" LOC = P80;
NET "DAC_WR" LOC = P81;

Main clock line

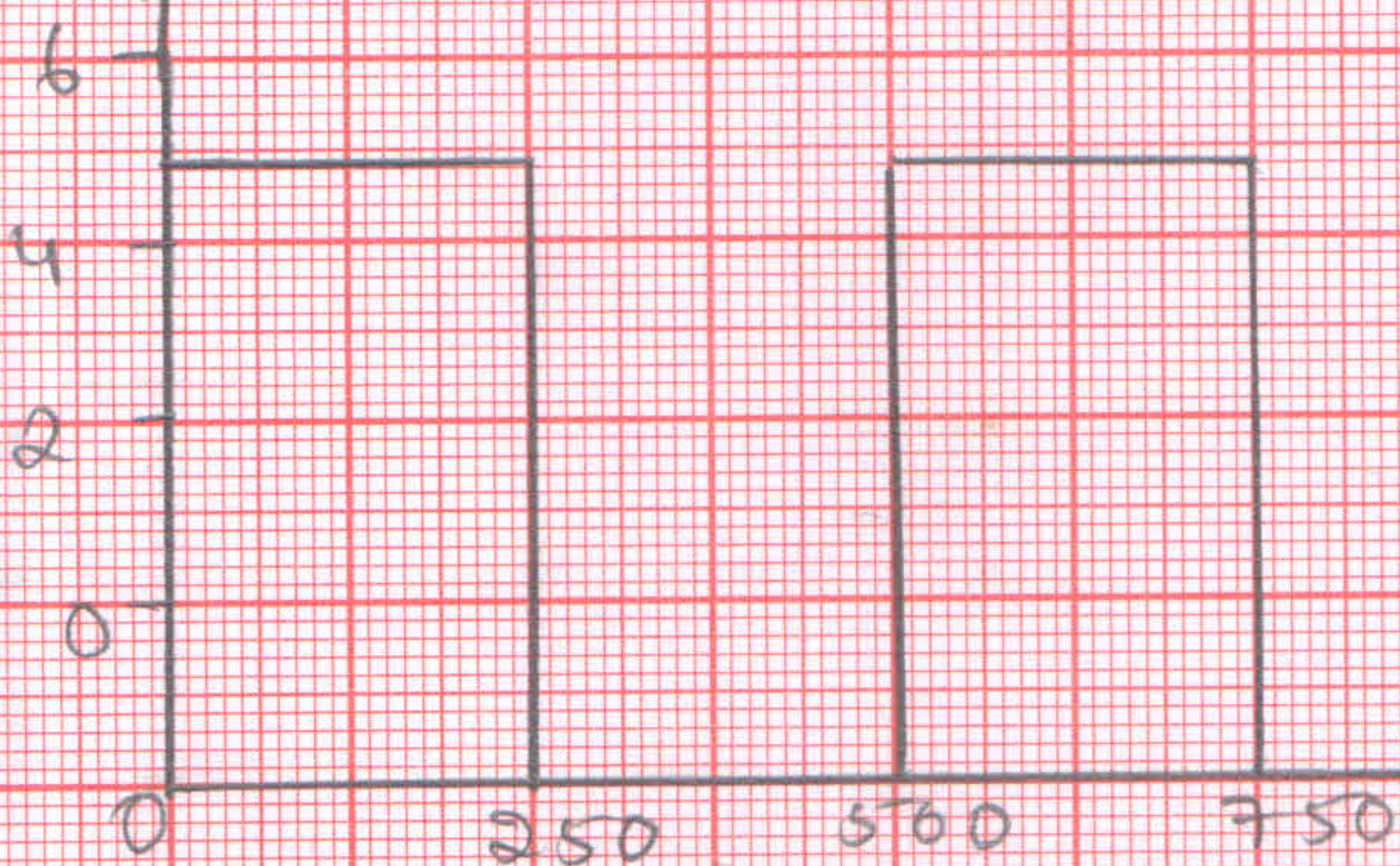
NET "P_Clk" LOC = P56;



DATE

EXP

V (volts)



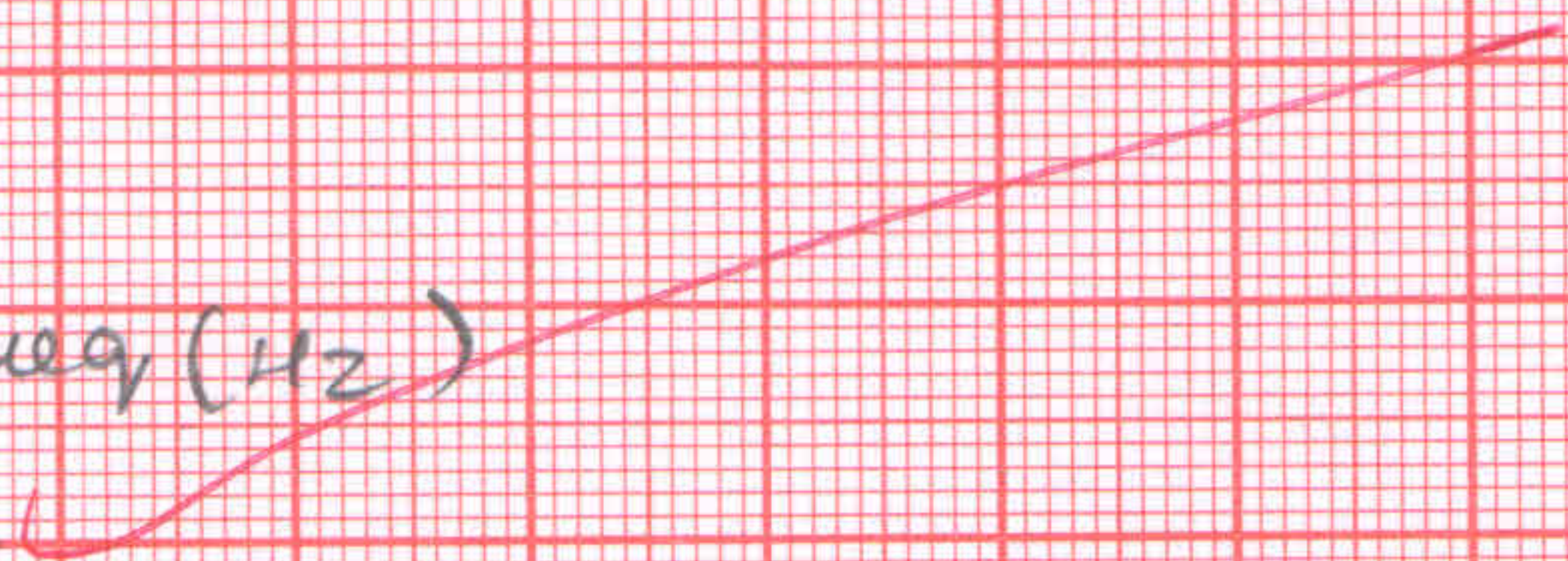
freq = 500 Hz

$V_{(p-p)}$
= 4.8V

Scale:

x axis: 2 units = 250 Hz

y axis: 1 unit = 2V



AIM : To implement & Test working of DAC

TOOL USED : Xilinx 13.1, Spartan-6, CRO.

Code :

Square waveform :

Module DAC_SQUARE(

input P_Clk,

output reg [7:0] P_DAC, // DAC out

output reg DAC_WR // DAC WR

);

reg [17:0] count = 0;

always @ (posedge P_Clk)

begin

if (count == 0) P_DAC = 0;

if (count == 2) DAC_WR = 1;

if (count == 10) DAC_WR = 0;

if (count == 50000) P_DAC = 'hFF;

if (count == 50002) P_DAC = WR = 1;

if (count == 50010) DAC_WR = 0;

count = count + 1;

if (count == 100000) count = 0;

end

end module

Theory :

A digital-to-analog converter (DAC, D/A, D2A or D to A) is a system that converts a digital signal to analog signal. DAC can degrade a signal, so a DAC should be specified that has insignificant errors in terms of application.

RESULT : DAC is implemented and tested working of square waveform

LCD

NET "LCD_RS" LOC = P7;

NET "LCD_EN" LOC = P8;

NET "~~LCD~~/P_LCD(0)" LOC = P97;

NET "P_LCD(1)" LOC = P98;

NET "P_LCD(2)" LOC = P94;

NET "P_LCD(3)" LOC = P93;

NET "P_LCD(4)" LOC = P92;

NET "P_LCD(5)" LOC = P88;

NET "P_LCD(6)" LOC = P87;

NET "P_LCD(7)" LOC = P85;

Main clock line

NET "P_Clk" LOC = P56;

AIM: To write a verilog code to Verify the working of LCD
TOOLS/VS

TOOLS USED: Xilinx 13.1, Spartan-6.

Code:

Module LCD-DEMO (

// Assuming 100MHz input clock. My need to adjust the Counter
// if any other input frequency is used. below.

input P-CLK,

// Register Select (rs), enable (en), and Data pins of LCD

output reg LCD_RS, LCD_EN,

output reg [7:0] P-LCD

);

Parameter length = 53; // message memory depth

integer counter;

// reg [32:0] delay = 32'h FFFF FFFF;

integer pointer = 0;

// Memory created to store the required data of 8 bits and
8 signals.

wire [8:0] memory [0: length - 1];

// function set : 8-bit, 2 line, 5x7 Dots

assign memory [0] = {1'b0, 8'h38};

// Display on cursor on

assign memory [1] = {1'b0, 8'h06};

// Entry Mode

assign memory [2] = {1'b0, 8'h0C};

// clear display (also clears DDRAM contents);

assign memory [3] = {1'b0, 8'h01};

// Memory location in middle are kept empty so that
the LCD is properly.

// initialized and is ready to accept data.

// character that should be displayed on the LCD

assign memory [20] = {1'b1, "*"};

assign memory [21] = {1'b1, "E"};

assign memory [22] = {1'b1, "C"};

assign memory [23] = {1'b1, "*"};

assign memory [24] = {1'b1, "D"};

assign memory [25] = {1'b1, "E"};

assign memory [26] = {1'b1, "P"};

assign memory [27] = {1'b1, "A"};

assign memory [28] = {1'b1, "R"};

assign memory [29] = {1'b1, "T"};

assign memory [30] = {1'b1, "M"};

assign memory [31] = {1'b1, "E"};

assign memory [32] = {1'b1, "A"};

assign memory [33] = {1'b1, "T"};

assign memory [34] = {1'b1, "*"};

assign memory [35] = {1'b1, "*"};

// Shift to second line of LCD

assign memory [36] = {1'b0, 8'hC0};

// character that should be displayed on the LCD

assign ~~mem~~ memory [37] = {1'b1, "*"};

assign memory [38] = {1'b1, "K"};

assign memory [39] = {1'b1, "S"};

assign memory [40] = {1'b1, "I"};

assign memory [41] = {1'b1, "T"};

assign memory [42] = {1'b1, "*"};

assign memory [43] = {1'b1, "B"};

assign memory [44] = {1'b1, "E"};


```
assign Memory [H5] = $1'b1, "N";
```

```
assign Memory [H6] = $1'b1, "G";
```

```
assign Memory [H7] = $1'b1, "A";
```

```
assign Memory [H8] = $1'b1, "L";
```

```
assign Memory [H9] = $1'b1, "U";
```

```
assign Memory [50] = $1'b1, "R";
```

```
assign Memory [51] = $1'b1, "V";
```

```
assign Memory [52] = $1'b1, "X";
```

```
// Scale down the clock such that it satisfy the timing  
characteristics of LCD
```

```
always @ (posedge P_Clk)
```

```
begin
```

```
Counter = Counter + 1;
```

```
// Check if all the Memory location are covered.
```

```
// Once its done disable the enable pin has no more data  
has to be send.
```

```
if (Pointer > Length)
```

```
LCD_EN = 'b0;
```

```
else
```

```
LCD_EN = Counter[15];
```

```
end
```

```
// Depending on the edge of the enable (en) signal Send the  
Command and data to LCD
```

```
always @ (negedge LCD_EN)
```

```
begin
```

```
// MSB bit of a Memory location is register select (rs) signal
```

```
LCD_RS = Memory [Pointer][8];
```

```
// Write data / Command to LCD
```

```
P_LCD = Memory [Pointer][7:0];
```


DATE

EXPT TITLE:

PAGE NO.

47

EXP. NO.

```
// once the data / command is send move to next location  
pointer = pointer + 1;  
end  
end Module
```

~~#LCD~~
~~NE-~~

THEORY:

A liquid-crystal display (LCD) is a flat-panel display or other electronically modulated optical device that uses the light modulating properties of liquid crystals. Liquid crystals does not emit light directly, instead using back light or reflector to produce images. in color or Monochrome.

RESULT: ∴ The verification of the Liquid crystal Display is done Successfully.

Record - 10
Observation - 10
Total - 20.

DA

EX



$$V_{p-p} = 4.8 \text{ V}$$

freq. (kHz)

Scale:

x-axis: 2 units = 9.8 kHz

y-axis: 1 unit = 1 V

* SINE WAVE:

Aim: To write a verilog code to implement and test working of DAC (Sine wave).

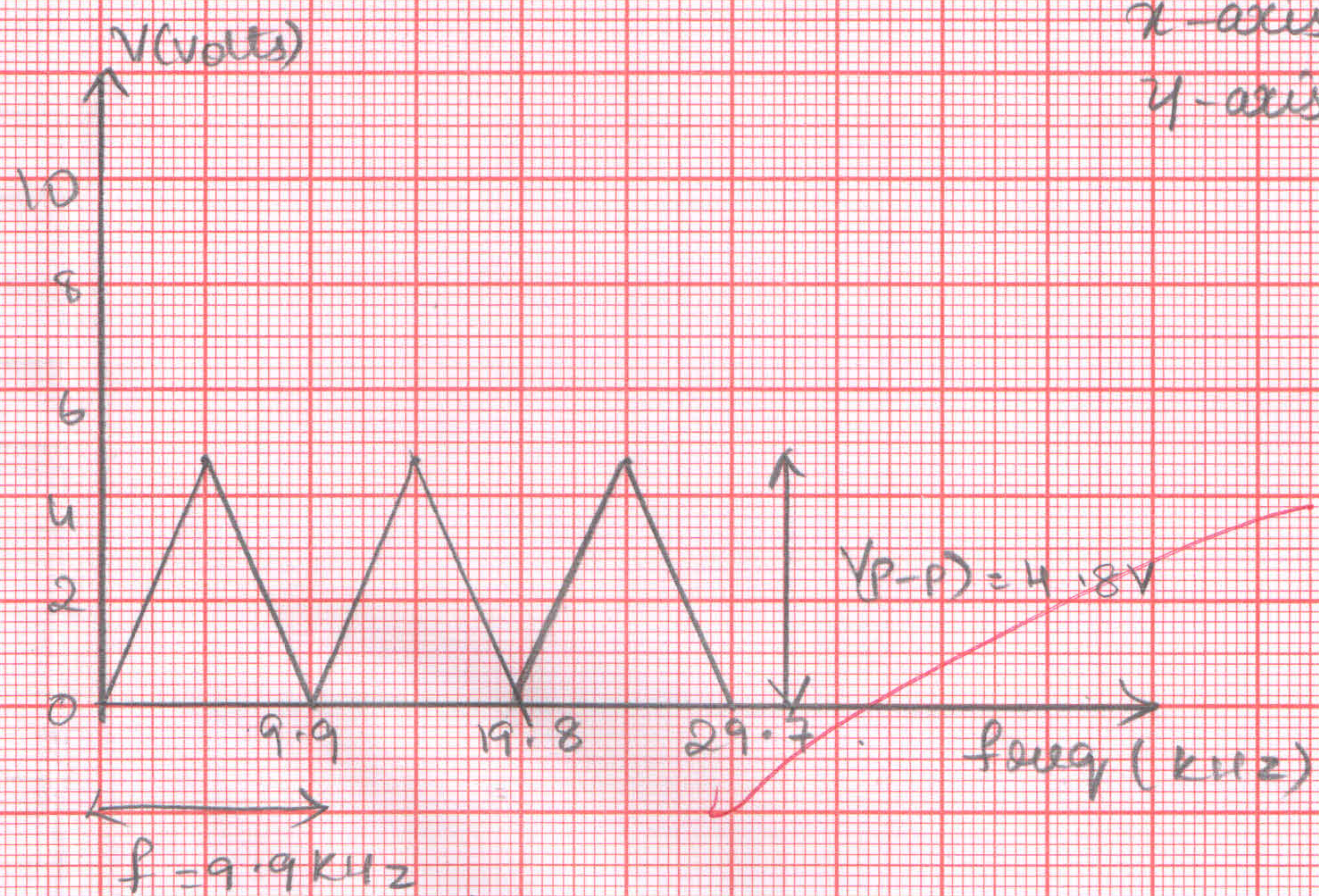
TOOL USED: Xilinx-13.1, Spartan-6, CRO.

Code:

```
Module DAC-SINE (  
    input P_Clk;  
    output reg [7:0] P_DAC, // DAC out  
    output reg DAC_WR // DAC WR  
);  
  
reg [4:0] count = 0;  
reg [8:0] i = 0;  
reg [7:0] sine [0:255];  
initial  
begin  
    $readmemb ("DAC-SINE.lst", sine);  
end  
  
always @ (posedge P_Clk)  
begin  
    if (count == 0)  
    begin  
        i = i + 1;  
        P_DAC = sine[i];  
        if (i == 255) i = 0;  
    end  
  
    if (count == 1) DAC_WR = 1;  
    if (count == 3) DAC_WR = 0;  
    count = count + 1;  
    if (count == 10) count = 0;  
end end Module
```


DATE ..

EXP. NO



Triangular wave

Aim: To verify the working of and implementation of DAC (triangular wave)

TOOLS USED: Xilinx 13.1, Spartan 6.

code:

```
Module DAC_TRIANGLE
```

```
input P_clk;
```

```
Output reg [7:0] P_DAC, // DAC out
```

```
Output reg DAC_WR // DAC WR
```

```
);
```

```
reg [3:0] count = 0;
```

```
reg [7:0] i = 0;
```

```
reg dir = 0;
```

```
always @ (posedge P_clk)
```

```
begin
```

```
    if (count == 0)
```

```
    begin
```

```
        if (dir == 0)
```

```
        begin
```

```
            i = i + 1;
```

```
            if (i == 255) dir = 1;
```

```
        end
```

```
    else
```

```
    begin
```

```
        i = i - 1;
```

```
        if (i == 0) dir = 0;
```

```
    end
```

```
    P_DAC = i;
```

```
end
```

```
if (count == 1) DAC_WR = 1;
```

```
if (count == 3) DAC_WR = 0;
```



```
Count = Count + 1;
```

```
if (Count == 10) Count = 0;
```

```
end
```

```
end Module
```

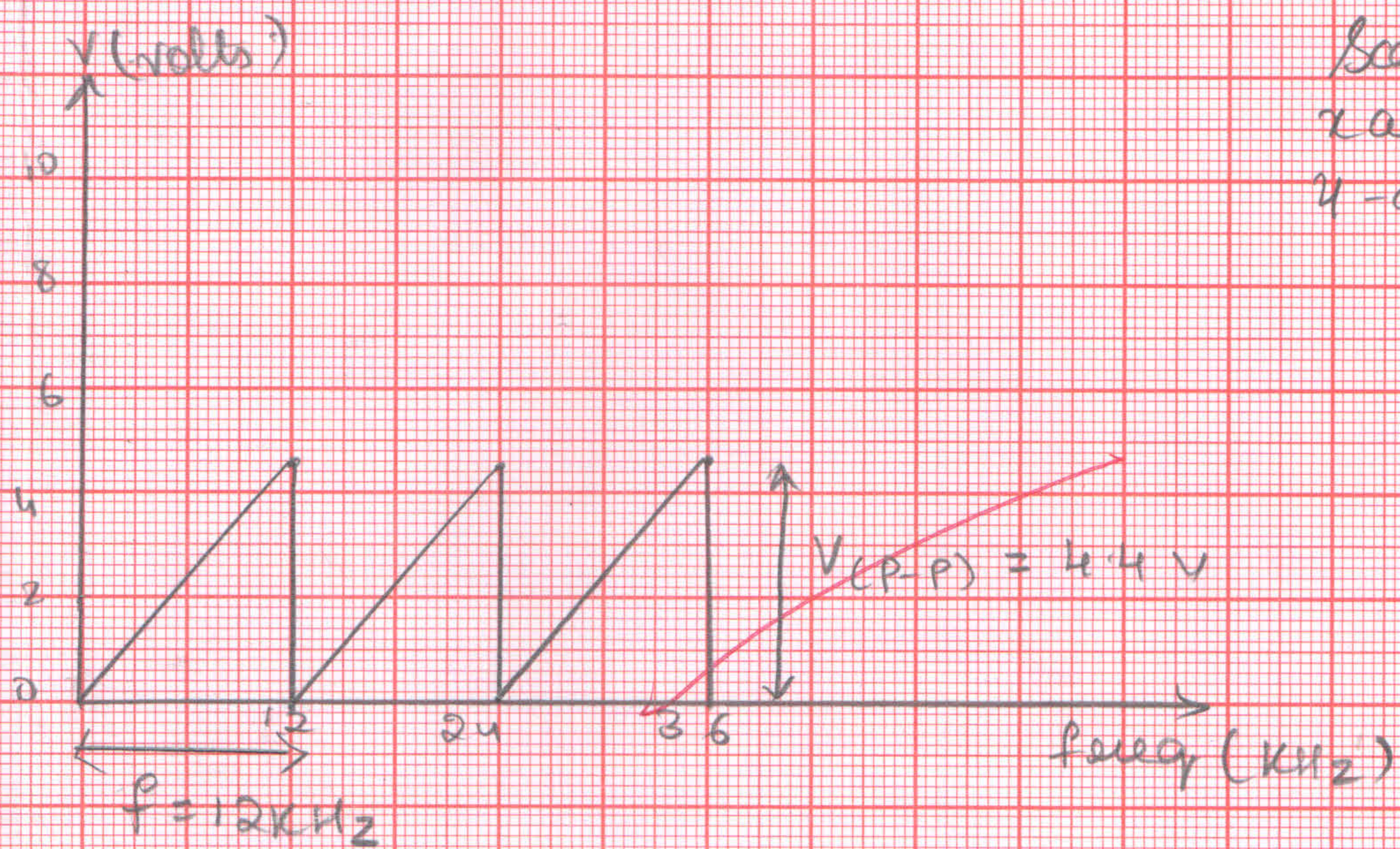
THEORY:

A digital-to-analog converter (DAC, D/A, D_2A or $D \rightarrow A$) is a system that converts a digital signal to analog signal. DAC can degrade a signal, so a DAC should be specified that has insignificant errors in terms of application.

RESULT: Therefore the DAC to generate triangular waveform is done successfully.

Record - 10
Observation - 10
Total - 20

DATE .
EXP. NO



Scale:
x-axis: 2 units = 12 kHz
y-axis: 1 unit = 2 V

AIM: To write a Verilog code to implement working of DAC (Sawtooth wave)

TOOLS USED: Xilinx 13.1, Spartan6

Code:

```
Module DAC_SAWTOOTH (
```

```
input P_Clk;
```

```
output [7:0] P_DAC, // DAC Out
```

```
output [reg] DAC_WR // DAC WR  
);
```

```
reg [3:0] Count = 0;
```

```
reg [7:0] dat = 0;
```

```
/* always @ (posedge P_Clk)
```

```
begin
```

```
    if (Count == 0)
```

```
    begin
```

```
        dat = dat + 1;
```

```
        if (dat == 255) dat = 0;
```

```
        P_DAC = dat;
```

```
    end
```

```
    if (Count == 1) DAC_WR = 1;
```

```
    if (Count == 3) DAC_WR = 0;
```

```
    Count = Count + 1;
```

```
    if (Count == 10) Count = 0;
```

```
end */
```

```
assign P_DAC = dat;
```

```
assign DAC_WR =
```

```
always @ (posedge P_Clk)
```

```
begin
```

```
    if (Count == 0)
```

```
        dat = dat + 1;
```



```
if (count == 1) DAC_WR = 1;  
else DAC_WR = 0;  
count = count + 1;  
end  
end module
```

RESULT:

DAC is implemented and tested the working of sawtooth waveform.


14/11

Record - 10
Observation - 10
Total - 20

~~##~~ Matrix Keypad IN

NET "MK_IN[0]" LOC = P62;

NET "MK_IN[1]" LOC = P61;

NET "MK_IN[2]" LOC = P58;

NET "MK_IN[3]" LOC = P57;

~~#~~

~~##~~ Matrix Keypad OUT

NET "MK_OUT[0]" LOC = P51;

NET "MK_OUT[1]" LOC = P50;

NET "MK_OUT[2]" LOC = P48;

NET "MK_OUT[3]" LOC = P46;

~~#~~ Main clock line

NET "P_Clk" LOC = P56;

AIM: To write a Verilog code to verify the working of HEX KEYPAD

TOOL USED: Xilinx 13.1, Spartan 6.

Code:

```
Module HEX_KEYPAD (
```

```
input P_Clk,
```

```
input [3:0] MK_IN, // Row in
```

```
output reg [3:0] MK_OUT, // Col out
```

```
output reg [3:0] P_dig, // anode signal of the 7 seg LED display.
```

```
output reg [7:0] P_7seg // Cathode signal of the 7 seg display.
```

```
);
```

```
reg [2:0] state = 0;
```

```
reg [7:0] count = 0;
```

```
reg clk = 0;
```

```
always @ (posedge P_Clk)
```

```
begin
```

```
if (count >= 50)
```

```
begin
```

```
clk = ~clk;
```

```
count = 0;
```

```
end
```

```
else count = count + 1;
```

```
end
```

```
always @ (posedge clk)
```

```
begin
```

```
P_dig = 'b0001;
```

```
case (state)
```

```
0: begin
```

```
P_7seg = 'b1111111; // null
```

```
MK_OUT = 'b1000;
```

```
state = 1;
```

```
end
```


1: begin

if (MK-IN == 'b1000) P_7seg = 'b10001000; // 0

if (MK-IN == 'b0100) P_7seg = 'b11101011; // 1

if (MK-IN == 'b0010) P_7seg = 'b01001100; // 2

if (MK-IN == 'b0001) P_7seg = 'b01001001; // 3

MK-OUT = 'b0100;

State = 2;

end

2: begin

if (MK-IN == 'b1000) P_7seg = 'b00001011; // 4

if (MK-IN == 'b0100) P_7seg = 'b00011001; // 5

if (MK-IN == 'b0010) P_7seg = 'b00011000; // 6

if (MK-IN == 'b0001) P_7seg = 'b1001011; // 7

MK-OUT = 'b0010;

State = 3;

end

3: begin

if (MK-IN == 'b1000) P_7seg = 'b00001000; // 8

if (MK-IN == 'b0100) P_7seg = 'b00001001; // 9

if (MK-IN == 'b0010) P_7seg = 'b00001010; // A

if (MK-IN == 'b0001) P_7seg = 'b00111000; // B

MK-OUT = 'b0001;

State = 4;

end

4: begin

if (MK-IN == 'b1000) P_7seg = 'b10011100; // C

if (MK-IN == 'b0100) P_7seg = 'b01101000; // D

if (MK-IN == 'b0010) P_7seg = 'b00011100; // E

if (MK-IN == 'b0001) P_7seg = 'b00011110; // F


```
State = 0;  
end  
end case  
end module.
```

THEORY:

The hex keypad is a peripheral that connects to the DE2 through JPI or JP2 via a 40-pin ribbon cable. It has 16 buttons in a 4 by 4 grid, labelled with the hexadecimal digits 0 to F.

Internally the structure of hex keypad is simple. There are 8 vertical columns and 4 horizontal rows.

These 8 lines are available externally, and will be connected to the lower 8 bits of the port.

Each key on the keypad is essentially a switch that connects a row line to a column line. When a key is pressed, it makes an electrical connection between the row and a column.

RESULT:

Therefore the verification of hex keypad is done successfully.

89
14/11

Revised - 10
Observation - 10
Total - 20



Kammavari Sangham (R)-1952
K.S.GROUP OF INSTITUTIONS
K.S. INSTITUTE OF TECHNOLOGY
ACCREDITED BY NAAC
(Approved by AICTE & Affiliated to VTU)
KANAKPURA ROAD, BENGALURU - 560 109

IA, PRACTICAL TEST /PROJECT VIVA- VOCE

Course/Branch: Electronics and Communication engineering.

1 K S I G E O 7 I
University Seat Number

Subject: VHDL LAB

Subject Code:

15 ECL 58

Total number of supplements tied together : 1 + _____ = _____

Date

10/10/2018.

D D M M Y Y Y Y

Signature of the Candidate

Entries to be made by the examiners

SCHEME OF AWARDING MARKS

Practical Examination					Project viva -voce				
a.	Procedure & write up	:		of Max. Marks	a.	Project Work	:	40% of Max .Marks	
b.	Conducting the practicals, calculations, graphs, results, etc.,	:		of Max. Marks	b.	Presentation	:	30% of Max .Marks	
c.	Viva-voce	:		of Max. Marks	c.	Viva-voce	:	30% of Max .Marks	
Question Number	Marks awarded				Title of the Project				
	a	b	c	Total					
1	11	31	03	45					
2									
3									
4									
					Total Max Marks	Marks awarded			
						a	b	c	Total
Total Maximum Marks : <u>80</u>					Total Marks obtained : <u>45</u>				
Marks awarded in words :					Marks awarded in words :				

Examiner 1

Examiner 2

Name

: Aruna Rao

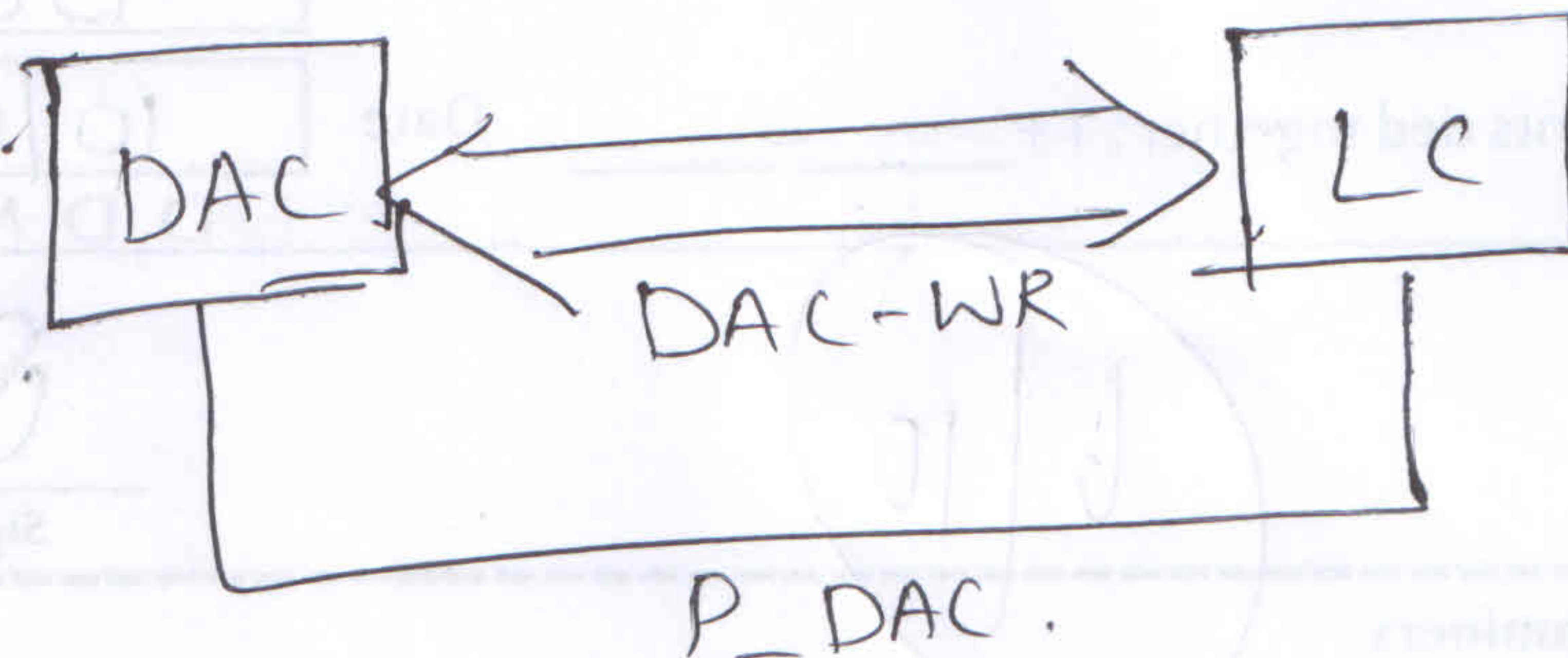
Signature with Date

: 10/10

Verify To write a Verilog Code to generate Sine and Saw tooth Wave.

Aim ; To write a Verilog Code to generate Sine and Saw tooth Wave.

Block diagram



Tools Used ; Xilinx 1.1, Spartan 6 and CRO ~~code~~

Code ;

SAW TOOTH :

Module DAC-SAWTOOTH (

Input P_clk,

Output reg [7:0] P_DAC,

output reg DAC-WR,

);

reg [3:0] count = 0;

reg [7:0] dat = 0;

assign P_DAC = dat;

always @ (posedge P_clk).

begin

if (count == 0).

dat = dat + 1;

if (count == 1) DAC-WR = 1;

else DAC-WR = 0;

end

b)

Sine Waveform

Module SINE_DAC (

input P_clk,

out reg [7:0] P_DAC;

out reg DAC_WR,

);

reg [4:0] count = 0;

reg [8:0] i = 0;

reg [7:0] sine [0:255]; initial.

always @ (posedge P_clk) begin.

begin.

if (count == 0)

begin

i = i + 1;

P_DAC = sine [i];

if (i == 255) i = 0;

end.

if (count == 1)

begin.

DAC_WR = 1;

end.

if (count == 3)

begin.

DAC_WR = 0;

end.

if else (count == 10) count = count + 1;

end.

end module.

begin.

\$readmem ("DAC-SINE.1st" sine);

end.



KSIT
K.S. INSTITUTE OF TECHNOLOGY

Kammavari Sangham (R)-1952
K.S.GROUP OF INSTITUTIONS
K.S. INSTITUTE OF TECHNOLOGY

ACCREDITED BY NAAC
(Approved by AICTE & Affiliated to VTU)
KANAKPURA ROAD, BENGALURU - 560 109

IA, PRACTICAL TEST /PROJECT VIVA- VOCE

Course/Branch: Electronics and Communication Engineering

1 K S I G E C O T I
University Seat Number

Subject: VHDL LAB.

Subject Code:

15ECL58

Total number of supplements tied together : 1 + _____ = _____

Date

29/11/2018.

D D M M Y Y Y Y

[Signature]

Signature of the Candidate

Entries to be made by the examiners

SCHEME OF AWARDING MARKS

Practical Examination

- a. Procedure & write up : of Max. Marks
- b. Conducting the practicals, calculations, graphs, results, etc., : of Max. Marks
- c. Viva-voce : of Max. Marks

Question Number	Marks awarded			
	a	b	c	Total
1	10	30	5	45
2				
3				
4				

Project viva -voce

- a. Project Work : 40% of Max .Marks
- b. Presentation : 30% of Max .Marks
- c. Viva-voce : 30% of Max .Marks

Title of the Project				
Total Max Marks	Marks awarded			
	a	b	c	Total

Total Maximum Marks :

80

Total Marks obtained :

45

Total Marks obtained :

Marks awarded in words :

Marks awarded in words :

Examiner 1

Examiner 2

Name :

Aruna Rao

Signature with Date :

[Signature]
29/11

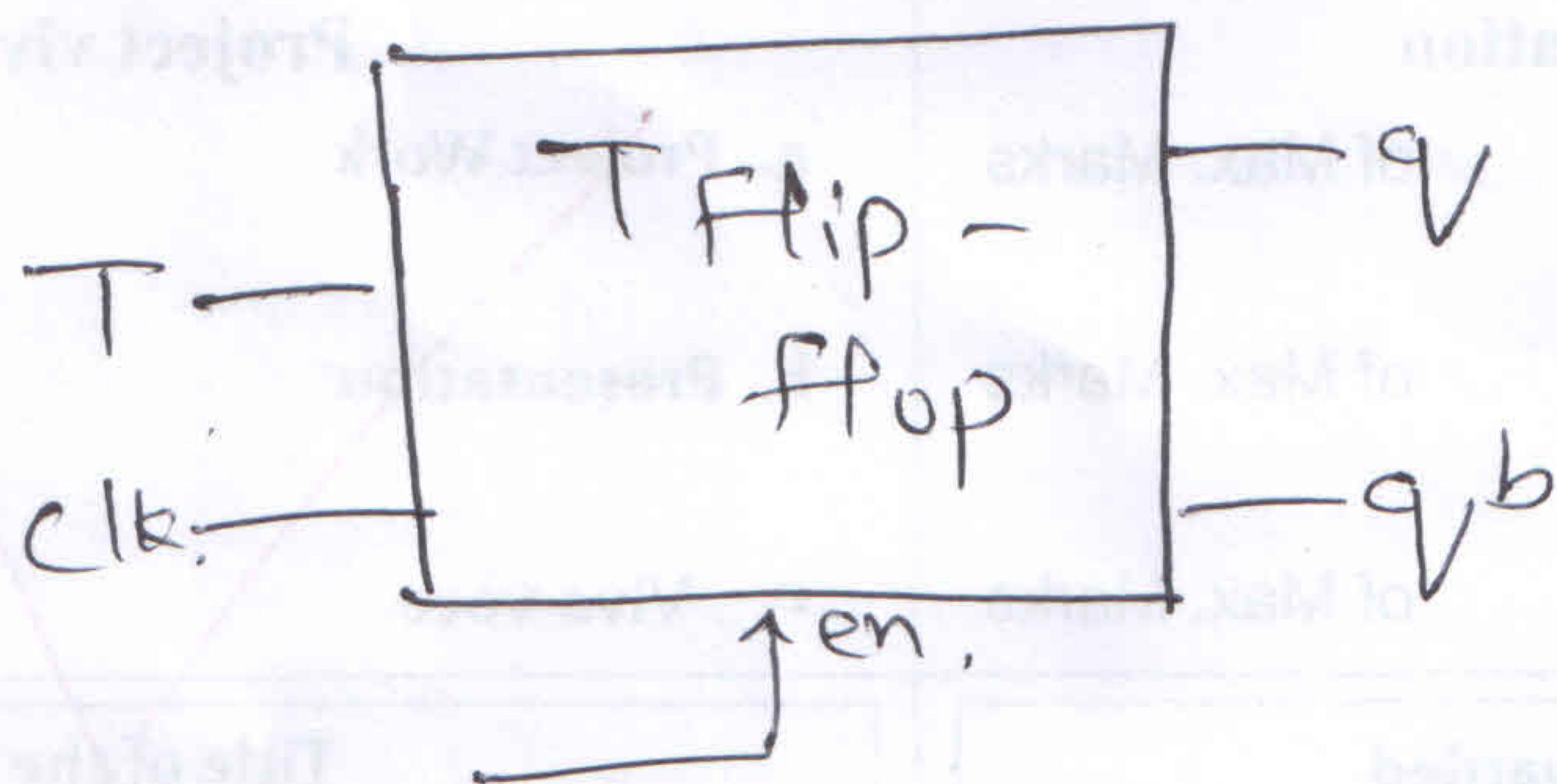
13. Write a Verilog Code for T-Flipflop. Simulate and Program to FPGA Kit.

23. Write a Verilog Code. to Control speed and directions of DC Motor.

13. Aim :- To verify the working of flipflop and to Simulate and program to FPGA kit.

Tool Used ; FPGA Kit (Spartan-6), Xilinx 1.

Block diagram ;



Truth Table :

in	clk	T	q	qb.
1		x	q	qb.
0		0	q	qb
0		1	$\overline{q}$	qb.

Code ;

```
Module T-Flipflop (en, clk, T, q, qb);
```

```
input en; clk, t;
```

```
output q, qb;
```

```
reg q = 0, qb;
```

```
reg [31:0] clk div = 32'd0
```



```
always @ (posedge clk),
```

```
clkdiv = clkdiv + 1;
```

```
always @ (posedge clk),
```

```
begin
```

```
if (en == 1)
```

```
q = 0;
```

```
else
```

```
if (t == 1)
```

```
q = q'b;
```

```
end;
```

```
else
```

```
q = q;
```

```
q'b = ~q;
```

```
end;
```

```
end module.
```

Result; $\therefore$ The verification of the T flip flop is done.
Successfully.

23: Aim : To write a Verilog Code to control speed and Direction of DC Motor.

Tool used : Spartan-6, Xilinx 1.1, DC Motor.

Code :

```
Module MOTOR-DC (
```

```
Input P_clk.
```

```
Output [1:0] P_DCM.
```

```
Output P_DCMEN,
```

```
Input div,
```

```
Input Speed Control,
```

```
) ;
```

```
reg [19:0] count = 20'd0.
```

```
Wire High-speed, low-speed;
```



```

always @ (posedge P_clk)
begin
assign P_DCMEN = Speed_Control ? High_Speed : Low_Speed;
assign High_Speed = (count < 20'HFFFFFF)? 1:0;
assign Low_Speed = (count < 20'H3FFFF)? 1:0;
assign P_DCM = (div)? 2'b01: 2'b10;
end
end module.

```

Result: $\therefore$ The Verification of the Speed Control and directions of DC Motor is done Successfully.





Branch : EC

Scheme : 2015

Semester : 5

Sl NO.	USN	15EC52	15EC53	15EC54	15EC553	15EC562	15EC563	15ECL57	15ECL58	15ES51	STUDENT SIGNATURE
1	1KS14EC068	13	16	16	13	-	14	17	14	18	<i>[Signature]</i>
2	1KS14EC101	12	14	12	16	-	13	15	16	15	<i>Sowjanya a.</i>
3	1KS14EC107	12	13	12	12	-	12	16	14	17	<i>[Signature]</i>
4	1KS14EC117	12	15	14	12	-	12	15	16	19	<i>[Signature]</i>
5	1KS15EC422	16	16	15	15	-	14	18	19	15	<i>[Signature]</i>
6	1KS16EC002	17	19	18	19	-	17	18	18	15	<i>[Signature]</i>
7	1KS16EC003	12	12	14	15	-	15	16	16	14	<i>[Signature]</i>
8	1KS16EC004	15	19	16	18	16	-	19	17	12	<i>Adithya DM</i>
9	1KS16EC006	14	13	14	15	-	14	14	15	15	
10	1KS16EC009	15	16	15	17	17	-	16	16	15	<i>Anurag S</i>
11	1KS16EC010	16	16	17	15	-	15	17	17	16	
12	1KS16EC011	19	18	17	14	-	17	19	19	15	
13	1KS16EC013	20	20	20	20	-	19	19	20	17	<i>Aswini</i>
14	1KS16EC014	15	15	14	14	-	14	15	12	13	<i>Ayush</i>
15	1KS16EC015	17	20	19	20	-	17	16	16	16	<i>Bhargya</i>
16	1KS16EC016	15	16	19	18	12	-	18	19	13	
17	1KS16EC017	20	20	20	20	-	19	19	18	18	
18	1KS16EC018	17	18	19	16	-	17	18	17	14	<i>Renuka</i>
19	1KS16EC019	20	18	18	20	-	20	17	19	16	<i>Chandana</i>
20	1KS16EC020	13	13	13	15	12	-	13	15	13	<i>Charan A</i>
21	1KS16EC022	19	18	18	19	15	-	18	19	14	
22	1KS16EC023	16	18	18	14	-	18	17	17	16	<i>[Signature]</i>
23	1KS16EC024	12	15	14	12	15	-	16	17	14	
24	1KS16EC025	13	17	14	15	-	14	15	16	15	
25	1KS16EC026	15	16	18	18	-	18	15	17	14	<i>[Signature]</i>
26	1KS16EC028	14	16	12	16	-	15	17	14	14	
27	1KS16EC030	10	12	12	12	-	12	13	12	12	
28	1KS16EC031	19	16	17	18	17	-	17	17	18	
29	1KS16EC032	18	20	20	20	-	18	18	20	18	<i>Jeonika</i>
30	1KS16EC034	19	20	20	20	-	20	18	20	20	<i>Sunitha</i>
31	1KS16EC035	15	15	19	16	16	-	15	20	14	<i>Karthik K.</i>
32	1KS16EC036	19	16	19	19	-	18	18	17	15	<i>Komal</i>
33	1KS16EC037	16	17	18	16	16	-	17	17	13	<i>Krishna E</i>
34	1KS16EC039	16	18	16	18	-	15	17	15	16	
35	1KS16EC040	20	19	20	20	-	19	19	20	19	
36	1KS16EC041	20	20	20	20	20	-	20	20	19	
37	1KS16EC042	17	17	19	18	18	-	18	17	14	<i>[Signature]</i>
38	1KS16EC043	17	14	15	15	14	-	15	17	16	<i>Sunayana</i>
39	1KS16EC044	14	16	14	16	14	-	17	17	14	<i>[Signature]</i>
40	1KS16EC045	16	14	15	16	13	-	18	16	12	
41	1KS16EC047	18	14	17	19	15	-	17	17	18	<i>[Signature]</i>
42	1KS16EC048	13	14	13	14	13	-	15	13	15	<i>Manjunatha</i>

Sl NO.	USN	15EC52	15EC53	15EC54	15EC553	15EC562	15EC563	15ECL57	15ECL58	15ES51	STUDENT SIGNATURE
43	1KS16EC049	20	20	20	20	-	18	20	20	18	Meenabha
44	1KS16EC050	16	19	18	19	19	-	17	19	18	Uthra
45	1KS16EC051	12	15	14	18	16	-	19	18	14	
46	1KS16EC052	19	19	20	18	17	-	20	20	17	Nam
47	1KS16EC053	19	20	20	20	-	19	20	20	19	
48	1KS16EC054	19	20	18	19	-	19	20	20	20	
49	1KS16EC055	19	18	19	19	-	19	19	20	18	
50	1KS16EC056	13	13	16	14	-	17	14	18	14	KSR
51	1KS16EC057	19	19	18	19	-	15	18	20	18	
52	1KS16EC058	19	20	19	18	-	18	18	20	18	Nela. N
53	1KS16EC059	19	19	20	19	-	18	20	20	17	Ant
54	1KS16EC060	17	16	17	12	17	-	18	14	18	Ant
55	1KS16EC061	19	20	20	20	20	-	20	19	20	Rupel
56	1KS16EC062	18	16	17	12	12	-	14	14	15	Ant
57	1KS16EC063	20	20	19	18	-	19	20	17	20	O. B. R. S.
58	1KS16EC064	19	20	20	20	-	19	20	20	20	Pratyusha
59	1KS16EC065	19	18	19	15	15	-	17	18	20	
60	1KS16EC067	18	19	19	15	-	16	19	15	16	Chitra
61	1KS16EC070	13	18	18	13	16	-	18	14	19	Krishna
62	1KS16EC071	17	20	19	16	16	-	18	16	20	Renuka
63	1KS16EC072	20	20	20	20	18	-	20	19	20	Rashmi
64	1KS16EC073	18	18	20	16	-	17	19	18	20	Ravinandan
65	1KS16EC074	14	16	17	12	13	-	15	17	16	
66	1KS16EC076	20	20	20	19	20	-	20	20	20	Ant
67	1KS16EC077	16	16	19	11	-	18	19	15	16	
68	1KS16EC078	12	20	18	13	-	15	18	19	19	Radhika
69	1KS16EC079	16	17	16	15	14	-	18	17	19	Radhika
70	1KS16EC080	20	20	20	18	17	-	19	16	20	
71	1KS16EC081	19	18	17	15	14	-	19	16	19	Saitan
72	1KS16EC082	14	14	14	08	12	-	16	17	16	Saravada
73	1KS16EC083	15	15	16	10	15	-	17	14	17	S. J. P.
74	1KS16EC084	16	18	15	15	14	-	19	16	19	Saravada
75	1KS16EC085	20	20	20	16	16	-	19	20	18	Shreya
76	1KS16EC087	14	17	16	15	13	-	17	15	19	Shreya
77	1KS16EC088	13	18	18	13	16	-	16	13	20	S. J. P.
78	1KS16EC089	15	17	14	14	-	13	16	15	16	S. J. P.
79	1KS16EC090	17	18	16	12	12	-	18	17	15	S. J. P.
80	1KS16EC091	17	18	16	14	12	-	18	18	19	S. J. P.
81	1KS16EC092	18	14	14	11	12	-	17	18	16	S. J. P.
82	1KS16EC094	19	19	20	19	19	-	19	19	20	Sonika
83	1KS16EC095	17	19	20	19	16	-	17	16	19	S. J. P.
84	1KS16EC096	18	20	18	17	17	-	19	20	18	S. J. P.
85	1KS16EC097	19	17	16	15	12	-	19	17	16	S. J. P.
86	1KS16EC098	14	15	16	13	16	-	15	17	17	S. J. P.
87	1KS16EC099	19	20	20	19	-	19	20	19	20	S. J. P.
88	1KS16EC102	19	20	20	19	18	-	20	20	20	S. J. P.
89	1KS16EC103	18	19	18	16	13	-	18	15	13	Tejas
90	1KS16EC105	18	17	16	14	19	-	20	18	20	Varshini
91	1KS16EC106	15	17	17	13	13	-	18	19	17	Varshini
92	1KS16EC108	19	20	20	18	18	-	20	20	18	Vidhya

Sl NO.	USN	15EC52	15EC53	15EC54	15EC553	15EC562	15EC563	15ECL57	15ECL58	15ES51	STUDENT SIGNATURE
93	1KS16EC109	20	20	20	20	17	-	20	20	20	Vidya.B
94	1KS16EC111	12	18	15	12	12	-	18	16	16	Vidya.B
95	1KS16EC112	18	18	20	16	13	-	19	19	17	Vidya.B
96	1KS16EC114	13	14	14	12	14	-	18	18	12	
97	1KS16EC115	17	17	17	16	-	15	20	16	19	Vidya.B
98	1KS16EC116	19	19	18	17	20	-	20	20	16	Vidya.B
99	1KS16EC401	12	13	13	12	-	13	16	17	15	Vidya.B
100	1KS16EC416	12	17	17	14	-	12	19	16	13	Vidya.B
101	1KS16EC419	12	14	17	16	-	14	15	16	12	
102	1KS16EC422	13	12	14	13	-	12	13	14	12	Vidya.B
103	1KS16EC430	12	14	15	16	-	12	14	17	13	Vidya.B
104	1KS17EC400	13	16	17	17	-	14	14	16	14	
105	1KS17EC402	11	12	13	14	-	12	14	15	13	
106	1KS17EC404	13	13	18	16	-	15	14	17	15	Chandrabhas
107	1KS17EC405	14	12	13	15	-	16	14	12	13	Chandrabhas
108	1KS17EC406	14	17	18	17	-	16	17	20	16	
109	1KS17EC407	15	19	14	14	-	18	18	19	17	Harsham
110	1KS17EC408	12	14	15	12	-	13	16	14	18	
111	1KS17EC409	17	18	20	17	-	16	19	19	19	
112	1KS17EC413	14	20	19	16	-	15	18	19	15	
--x--	Faculty Signature	-----XXXXXXXX-----									

* - values are either optional subjects or the faculty has not yet entered the marks

HOD

Seal and Signature

HEAD OF THE DEPARTMENT

Dept. of Electronics & Communication Engg

K.S. Institute of Technology

Bengaluru - 560 109

PRINCIPAL

Seal and Signature

PRINCIPAL

K.S. INSTITUTE OF TECHNOLOGY

BENGALURU - 560 109