Microprocessor and VLSI Lab

Lab Manual

ECPC-304

HDL Lab

Bachelor of Technology

in

Electronics & Communication Engineering

Department of Electronics & Communication Engineering

National Institute of Technology

Kurukshetra-136119

Website: www.nitkkr.ac.in

Vision

To impart state-of-the-art Electronics and Communication Engineering Education and Research responsive to global challenges.

Mission

- **M1:** To prepare students with strong theoretical and practical knowledge by imparting quality education.
- **M2:** To produce comprehensively trained and innovative graduates in Electronics and Communication Engineering through hands on practice and research to encourage them for entrepreneurship.
- **M3:** To inculcate team work spirit and professional ethics in students.

Program Educational Objectives (PEOs)

PEO – 1: Have a lead and successful role in their professional career.

PEO – 2: Be able to analyze real life problems and design socially accepted and economically viable solutions in Electronics and Communication Engineering area.

PEO –3: Be capable of lifelong learning and professional development by pursuing higher education and participation in research and development activities.

PEO –4: Have appropriate human and technical communication skills to be a good team-member/leaders and responsible human being

Program Outcomes (POs)

A graduate of the Electronics and Communication Engineering Program will:

PO1: **Engineering knowledge**: Possess knowledge of mathematics, science, engineering fundamentals, and Electronics and Communication Engineering specialization to solve the problems in Electronics and Telecommunication Systems.

PO2: **Problem analysis**: Be able to analyze complex problems in Communication systems, Analog &Digital Electronic Systems, & DSP based systems using first principles of mathematics, science, and engineering sciences to reach substantiated conclusions.

PO3: **Design/development of solutions**: Be able to design solutions for complex Electronics and communication engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: **Conduct investigations of complex problems**: Be able to use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: **Modern tool usage**: Be able to create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6: **The engineer and society**: Be able to apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional Electronics and Communication Engineering practice.

PO7: **Environment and sustainability**: Be able to understand the impact of Electronics and communication engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: **Ethics**: Be able to apply ethical principles, commit to professional ethics in context to Electronics and communication engineering practice.

PO9: **Individual and team work**: Be able to function effectively as an individual, as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: **Communication**: Be able to communicate effectively on complex Electronics and communication engineering activities with the engineering community and with society at large.

PO11: **Life-long learning**: Be able to recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PO12: **Project management and finance:** Have knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member/ leader in a team, to manage projects in multidisciplinary environments.

Program Specific Outcomes (PSOs)

At the end of the program, the student will:

PSO1: Clearly understand the fundamental concepts of Electronics and Communication Engineering.

PSO2: Formulate the real life problems and develop solutions in the area of semiconductor technology, signal processing and communication systems.

PSO3: Posses the skills to communicate effectively in both oral and written forms, demonstrating the practice of professional ethics, and responsive to societal and environmental needs.

INTRODUCTION TO VERILOG

Introduction to Verilog :

Verilog is a HARDWARE DESCRIPTION LANGUAGE (HDL). It is a language used for describing a digital system like a network switch or a microprocessor or a memory or a flip−flop. It means, by using a HDL we can describe any digital hardware at any level. Designs, which are described in HDL are independent of technology, very easy for designing and debugging, and are normally more useful than schematics, particularly for large circuits.

Verilog supports a design at many levels of abstraction.

Different abstraction Levels of Modeling.

1. Behavioral Level
2. Data flow level
3. Structural Level
4. Gate level

Capabilities of Verilog:

Verilog is a hardware description language (HDL) that is used to model electronic systems. It has a variety of capabilities, including:

- Design description

  Verilog can describe design behavior, data flow, structure composition, and delay and waveform generation.

- Module design

  Verilog can design modules using different coding styles, and the internals of each module can be defined at four levels of abstraction. The module will behave similarly to the external environment regardless of the internal abstraction level.

- Operators

  Verilog supports a variety of operators, including arithmetic, relational, logical, and bitwise operators. These operators allow designers to perform calculations, comparisons, and data manipulations.

- Procedural and continuous assignments

  These assignments are used to describe the behavior of digital systems and model the relationship between signals and combinational logic.

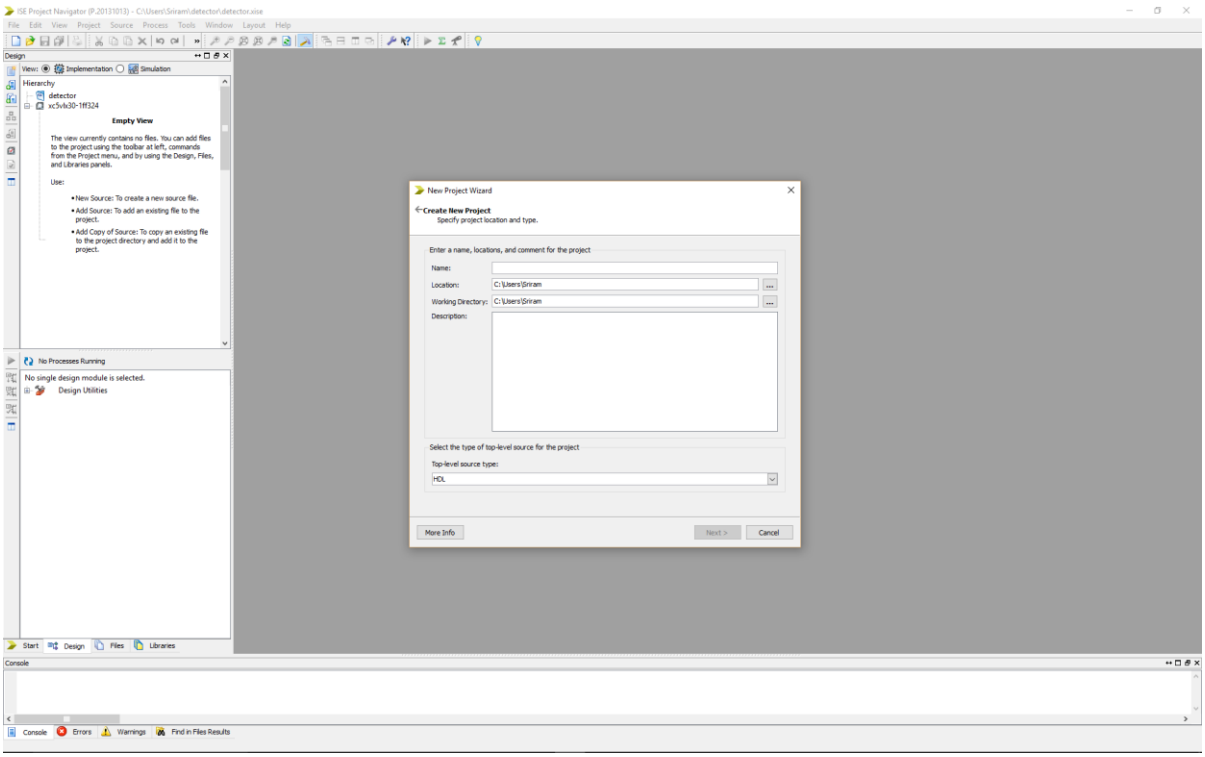
- Parallelism

  Verilog code is inherently parallel, so two or more instantiations of a module can result in multiple pieces of hardware that can operate simultaneously.

Verilog is often used to design and verify digital circuits at the register-transfer level, but it can also be used to verify analog and mixed-signal circuits, and to design genetic circuits.

Steps to implement the design

Step 1: Start the Xilinx project navigator by Stat->programs->Xilinx ISE->Project Navigator
Step 2: In the project navigator window click on new project->give file name->next.



Step 3: In the projector window right click on project name-> new source->Verilog module->give file name->define ports->finish.
Step 4: Write the Verilog code for any gate or circuit.
Step 5: Check Syntax and remove error if present.
Step 6: Simulate design using Modelsim/ISIM.
Step 7: In the project navigator window click on simulation->click on simulate behavioral model. Step 8: Give inputs by right click on any input->force constant
Step 9: Run simulation
Step 10: Analyze the waveform.

EXPERIMENT-1

Objective: Write a program to implement 3:8 decoder

Resources Required:

Hardware Requirement: Computer

Software Requirement: XILINX 14.7 Software,Isim (verilog)

Theory:

A binary decoder is a combinational logic circuit that converts a binary integer value to an associated pattern of output bits. They are used in a wide variety of applications, including data de- multiplexing, seven segment display, and memory address decoding. Decoder is with multiple data inputs and multiple outputs that converts every unique combination of data input states into a specific combination of output states.

Example: Imagine you are a mall security guard. In your office is a very important and unique public announcement (PA) phone. The phone has three dialing buttons (A, B, C) and is connected to eight different speakers, as shown in Table 1. Consequently, you get to choose which section of the mall hears your announcement based on the set of buttons you press. For example, if you press A and B and start speaking into the phone (ABC = 110), the Food Court (D6) is the only place that can hear you. However, if you press A and C (ABC = 101) then the Lady's Room (D5) is the only place that can hear you.

Such a public announcement phone (or PA system) is an example of a 3-to-8 decoder. Since the phone has three buttons each of which can either be in one of two possible states — pressed (=1) or not pressed (= 0) — then the phone can dial eight possible different numbers ($2^3 = 2*2*2 = 8$) as shown below

A	B	C		MALL AREA
0	0	0		Security lunch room (D0)
0	0	1		Men's Room (D1)
0	1	0		Footwear Stores (D2)
0	1	1		Jewelry Dealers (D3)
1	0	0		Appliance Stores (D4)
1	0	1		Lady's Room (D5)
1	1	0		Food Court (D6)
1	1	1		Bookstores (D7)

Application:The Decoders were used in analog to digital conversion in analog decoders, Used in electronic circuits to convert instructions into CPU control signals, also used in logical circuits, data transfer.

Truth Table:

Inputs				Outputs							
EN	A	B	C	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	x	x	x	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

Boolean Expression

Y0 = A'B'C'

Y1 = A'B'C

Y2 = A'BC'

Y3 = A'BC

Y4 = AB'C'

Y5 = AB'C

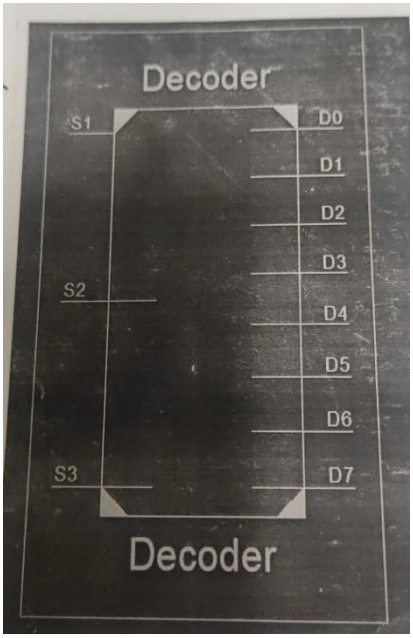
Y6 = ABC'

Y7 = ABC

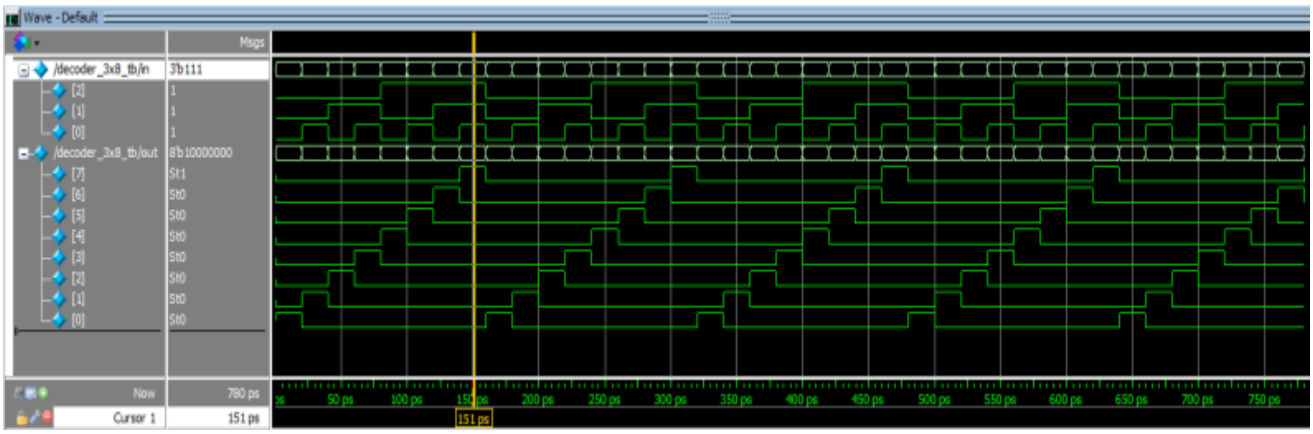
Verilog Code:

```
module Decoder(S1,S2,S3,D0,D1,D2,D3,D4,D5,D6,D7);
input S1,S2,S3;
output D0,D1,D2,D3,D4,D5,D6,D7;

and G0(D0,!S3,!S2,!S1);
and G1(D1,!S3,!S2,S1);
and G2(D2,!S3,S2,!S1);
and G3(D3,!S3,S2,S1);
and G4(D4,S3,!S2,!S1);
and G5(D5,S3,!S2,S1);
and G6(D6,S3,S2,!S1);
and G7(D7,S3,S2,S1);

endmodule
```

RTL Design:



Output Simulation:



Results: Verilog codes of 3:8 decoder is simulated & synthesized

Question:

1. How many 3-line-to-8-line decoders are required for a 1-of-32 decoder?
2. How many data select lines are required for selecting eight inputs?
3. How many 1-of-16 decoders are required for decoding a 7-bit binary number?

EXPERIMENT -2 (a)

Objective: Write a program to implement **8:1 multiplexer**.

Resources Required:

Hardware Requirement: Computer

Software Requirement: XILINX 14.7 Software

Theory:

Multiplexer is simply a data selector. It has multiple inputs and one output. Any one of the input line is transferred to output depending on the control signal. This type of operation is usually referred as multiplexing. In 8:1 multiplexer, there are 8 inputs. Any of these inputs are transferring to output, which depends on the control signal. For 8 inputs we need 3 bit wide control signal If control signal is "000", then the first input is transferring to output line. If control signal is "111", then the last input is transferring to output. Similarly, for all values of control signals.

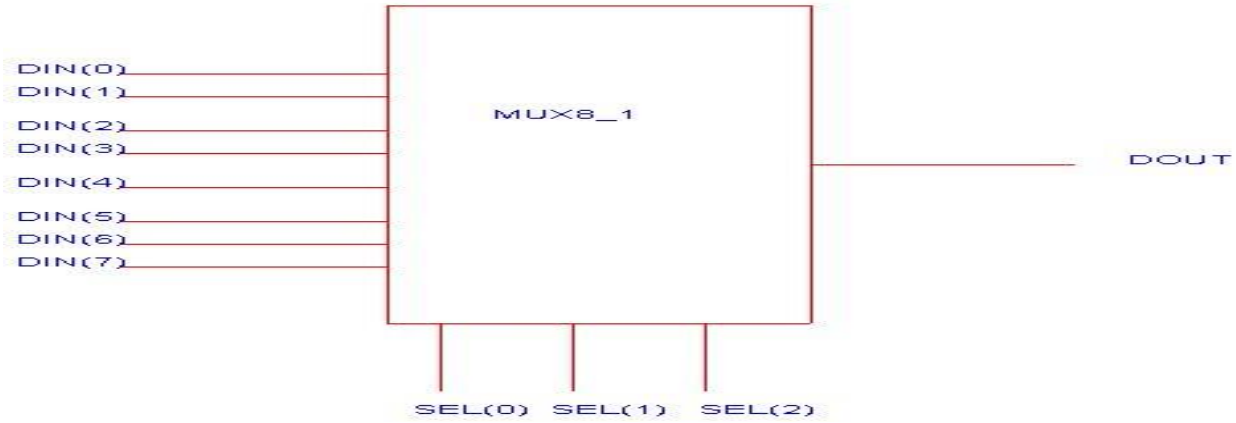


Figure 2.2 : Logical Diagram of 8:1 Mux

Truth Table:

Data Input	Select Inputs			Outputs							
D	S_2	S_1	S_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
D	0	0	0	0	0	0	0	0	0	0	D
D	0	0	1	0	0	0	0	0	0	D	0
D	0	1	0	0	0	0	0	0	D	0	0
D	0	1	1	0	0	0	0	D	0	0	0
D	1	0	0	0	0	0	D	0	0	0	0
D	1	0	1	0	0	D	0	0	0	0	0
D	1	1	0	0	D	0	0	0	0	0	0
D	1	1	1	D	0	0	0	0	0	0	0

Simplified as:

Select Data Inputs			Output
S_2	S_1	S_0	Y
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	0	0	D_4
1	0	1	D_5
1	1	0	D_6
1	1	1	D_7

Boolean Expression

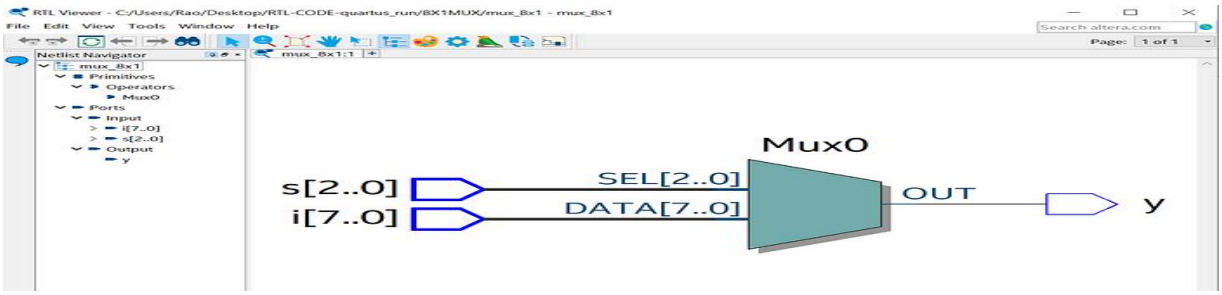
$$Y = D0\,\overline{S2}\,\overline{S1}\,\overline{S0} + D1\,\overline{S2}\,\overline{S1}\,S0 + D2\,\overline{S2}\,S1\,\overline{S0} + D3\,\overline{S2}\,S1\,S0 + D4\,S2\,\overline{S1}\,\overline{S0} + D5\,S2\,\overline{S1}\,S0 + D6\,S2\,S1\,\overline{S0} + D7\,S2\,S1\,S0$$

Application: Communication systems for modulation purpose, telephone networks, parallel to serial convertor.

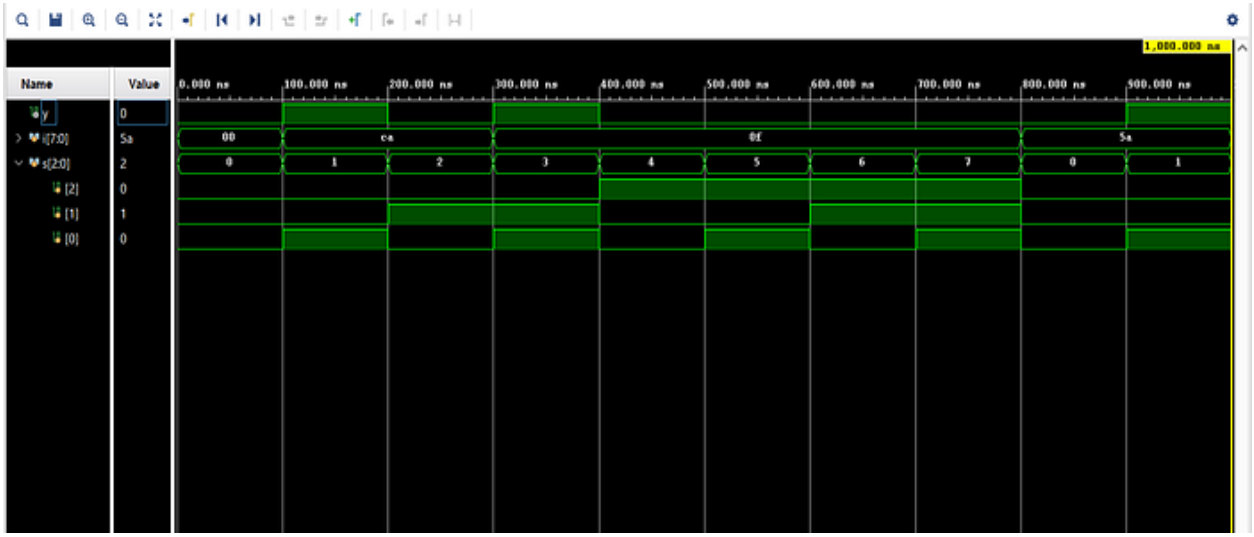
Verilog code:

```
module MUX8_1(i0,i1,i2,i3,i4,i5,i6,i7,s0,s1,s2,o);
input i0,i1,i2,i3,i4,i5,i6,i7,s0,s1,s2;
output o;

assign o=(i0 && !s0 && !s1 && !s2) ||
         (i1 && !s0 && !s1 && s2) ||
         (i2 && !s0 && s1 && !s2) ||
         (i3 && !s0 && s1 && s2) ||
         (i4 && s0 && !s1 && !s2) ||
         (i5 && s0 && !s1 && s2) ||
         (i6 && s0 && s1 && !s2) ||
         (i7 && s0 && s1 && s2);

endmodule
```

RTL Design:



Output Simulation:



Results: Verilog codes of 8:1 Multiplexer is simulated & synthesized.

Questions:

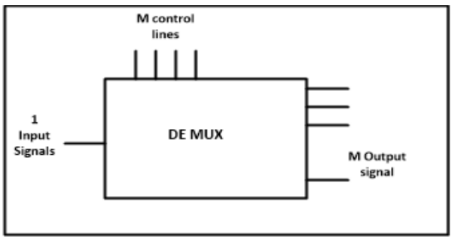
1. How many inputs in case of 4 to 1 MUX?
2. One multiplexer can take the place of which type of circuit and why?
3. What is data routing in a multiplexer?

EXPERIMENT-2 (b)

Objective: Write a program to implement a **1:8 Demultiplexer**.

Tools used: Xilinx-ISE 14.7.

Theory: Demultiplexer means one to many. A demultiplexer is a circuit with one input and many output. By applying control signal, we can steer any input to the output. Few types of demultiplexer are 1-to 2, 1-to-4, 1-to-8 and 1-to 16 demultiplexer. Following figure illustrate the general idea of a demultiplexer with 1 input signal, m control signals, and n output signals.



Demultiplexer Pin Diagram

Applications: The main application area of demultiplexer is communication system,ALU,serial to parallel converter.

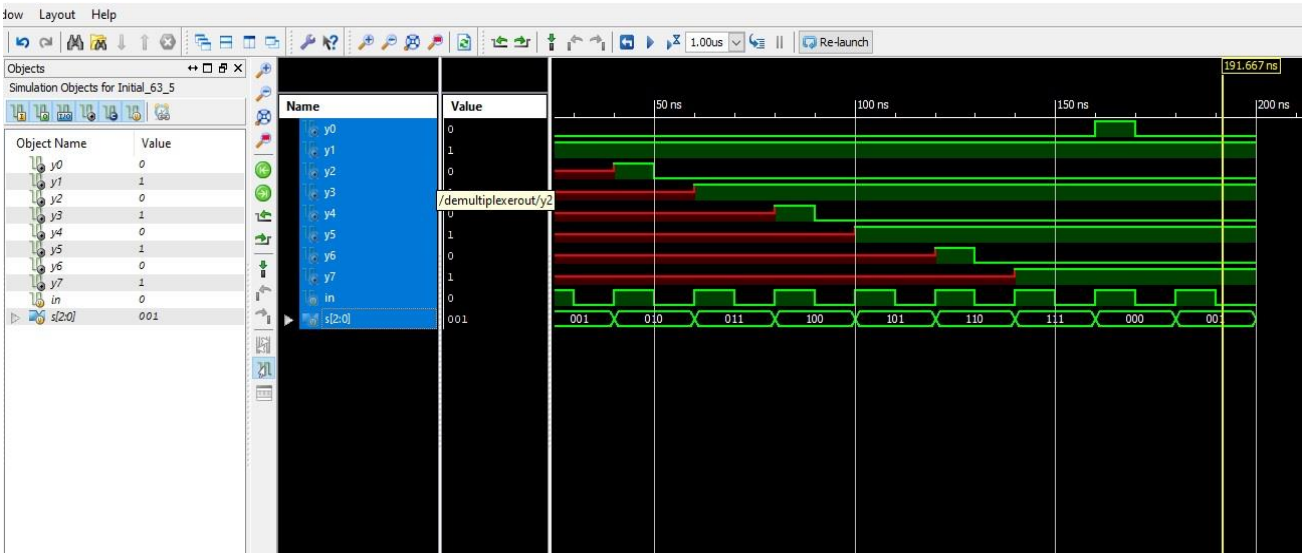
Verilog Code:

```
module demultiplexer (I, S2, S1, S0, y0,y1,y2,y3,y4, y5,y6,y7);
input I,S2,S1,S0;
output y0,y1,y2,y3, y4,y5,y6,y7;
assign y0=(!S2) &&(!S1) && (!S0) &&I;
assign y1=(!S2) &&(!S1) && (S0) &&I;
assign y2=(!S2) && (S1) && (!S0) &&I;
assign y3=(!S2) && (S1) && (S0) &&I;
assign y4=(S2) &&(!S1)&&(!S0) &&I;
assign y5=(S2) && (!S1) && (S0) &&I;
assign y6=(S2) && (S1) && (!S0) &&I;
assign y7=(S2) && (S1) && (S0) &&I;

endmodule
```

RTL Schematic:

Output:

1 to 8 DEMUX Response



Results: Verilog codes of 1:8 Demultiplexer is simulated & synthesized.

EXPERIMENT-3

Objective: Write a program to implement 4 bit addition/subtraction

Tools Used: Xilinx-ISE 14.7,Isim(vhdl/verilog)

Theory:

A full adder is a combinational circuit that forms the arithmetic sum of three bits. It consists of three inputs and two outputs. Two of the inputs variables represent the two significant bits to be added. The third input represents the carry from the previous lower significant position. The two outputs are designated by the symbols S for sum and C for carry. The binary variable S gives the value of the least significant bit of the sum. The binary variable C gives the output carry. The output variables are determined from the arithmetic sum of the input bits. A binary adder is a digital circuit that produces the arithmetic sum of two binary numbers. It can be constructed with full adder connected in cascade, the output carry from each full adder connected to the input carry of the next full adder in the chain. Figure 4.1 shows the interconnection of four full adder (FA) circuits to provide a 4-bit binary ripple carry adder. The augend bits of A and addend bits of B are designated by subscript numbers from right to left, with subscript 0denoting the least significant bit. The carries are connected in the chain through the full adders. The input carry to the adder is C0 and it ripples through the full adder to the output carry C4.The S output generate the required sum bits.

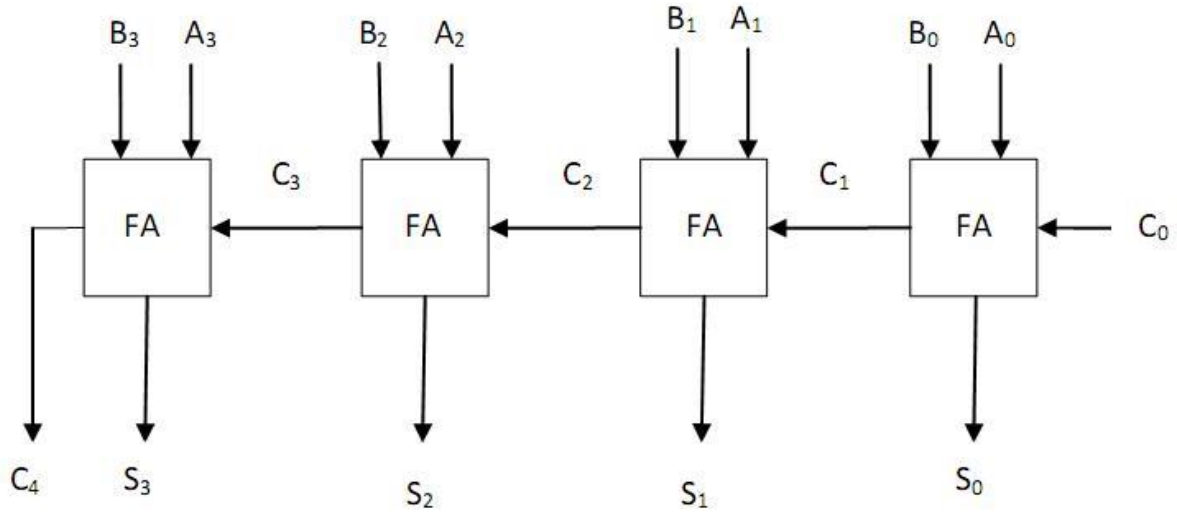


Figure 4.14-Bit Adder

The 4bit adder is a typical example of a standard component. It can be used in many applications involving arithmetic operations. Observe that the design of this circuit by the classical method would require a truth table with $2^9 = 512$ entries, since there are nine inputs to the circuit. By using an iterative method of cascading a standard function, it is possible to obtain a simple and straightforward implementation.

BinarySubtractor

The subtraction of unsigned binary numbers can be done most conveniently by means of complement. Subtraction A–B can be done by tacking the 2's complement of B and adding it to A. The 2's complement can be obtained by taking the 1's complement and adding one to the least significant pair of bits. The 1's complement can be implemented with the inverters and a one can be added to the sum through the input carry as shown in figure4.2.

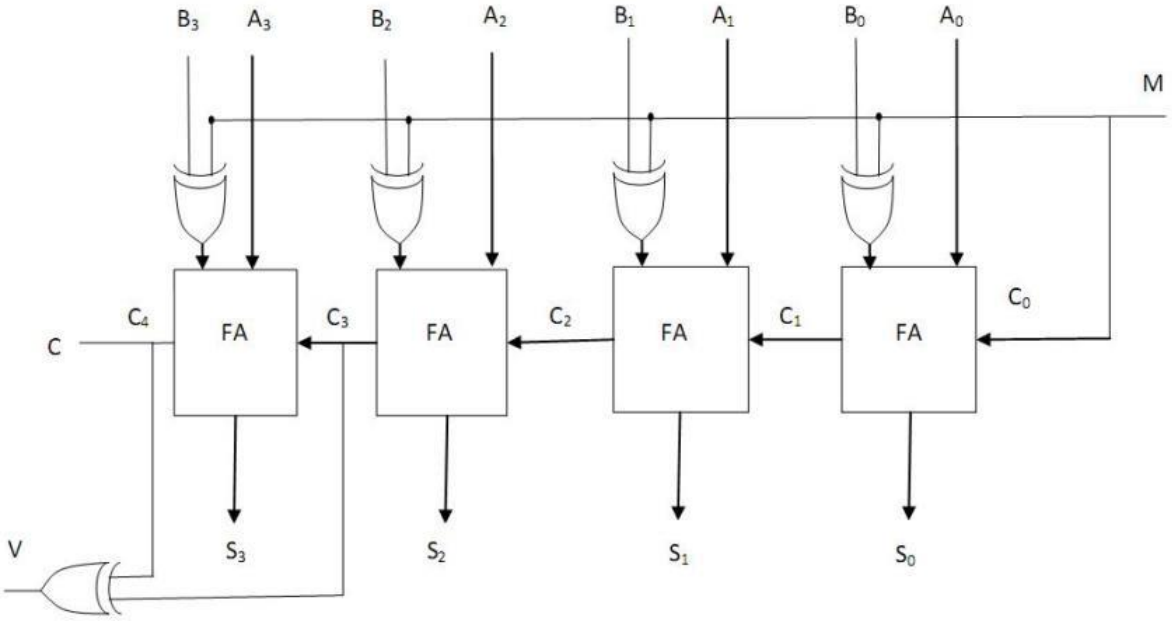


Figure 4.2 (4-Bit Adder Subtractor)

Application:

1) The ALU (arithmetic logic circuitry) of a computer uses half adder to compute the binary addition operation on two bits.

2) Half adder is used to make full adder as a full adder requires 3 inputs, the third input being an input carry i.e. we will be able to cascade the carry bit from one adder to the other.

3) Ripple carry adder is possible to create a logical circuit using multiple full adders to add N-bit numbers. Each full adder inputs a C(in), which is the C(out) of the previous adder. This kind of adder is called RIPPLE CARRY ADDER, since each carry bit "ripples" to the next full adder. Note that the first full adder (and only the first) may be replaced by a half adder.

Truth table for 4 bit adder:

Cin	A				B				Sum				Carry
	A3	A2	A1	A0	B3	B2	B1	B0	S3	S2	S1	S0	Cout
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	1	0	0	1	0	0
0	0	0	1	0	0	0	1	0	0	1	0	0	0
0	0	0	1	1	0	0	1	1	0	1	1	0	0
0	0	1	0	0	0	1	0	0	1	0	0	0	0
0	0	1	0	1	0	1	0	1	1	0	1	0	0
0	0	1	1	0	0	1	1	0	1	1	0	0	0
0	0	1	1	1	0	1	1	1	1	1	1	0	0
0	1	0	0	0	1	0	0	0	0	0	0	0	1
0	1	0	0	1	1	0	0	1	0	0	1	0	1
0	1	0	1	0	1	0	1	0	0	1	0	0	1
0	1	0	1	1	1	0	1	1	0	1	1	0	1
0	1	1	0	0	1	1	0	0	1	0	0	0	1
0	1	1	0	1	1	1	0	1	1	0	1	0	1
0	1	1	1	0	1	1	1	0	1	1	0	0	1
0	1	1	1	1	1	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1

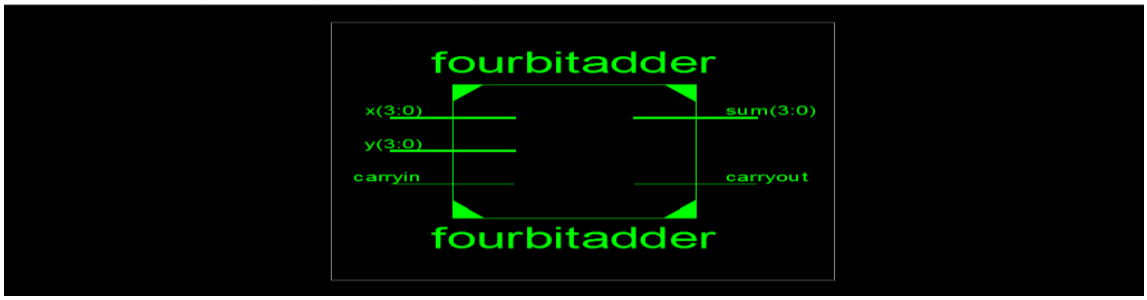
Verilog Code:

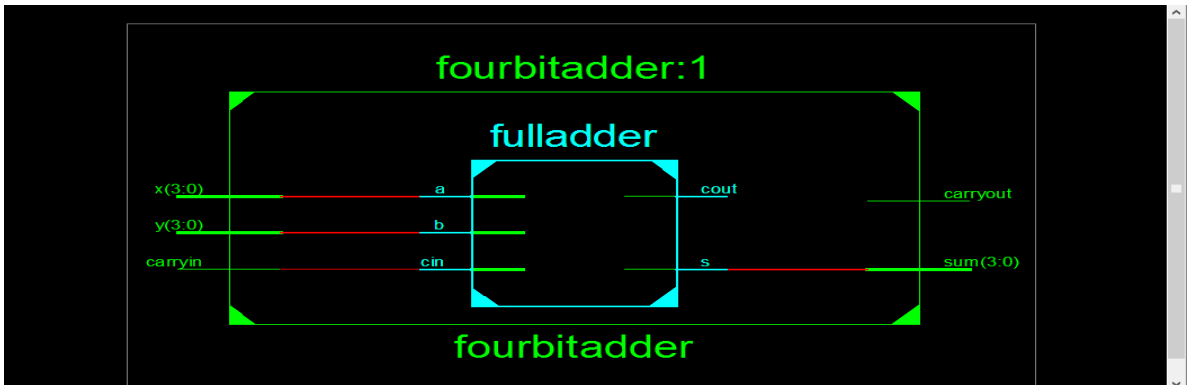
Adder

```
module fa (
    input a,b,cin,
    output wire s,cout
);
    assign s=a ^ b ^ cin;
    assign cout= a&b | b&cin) | cin&a;



endmodule
```

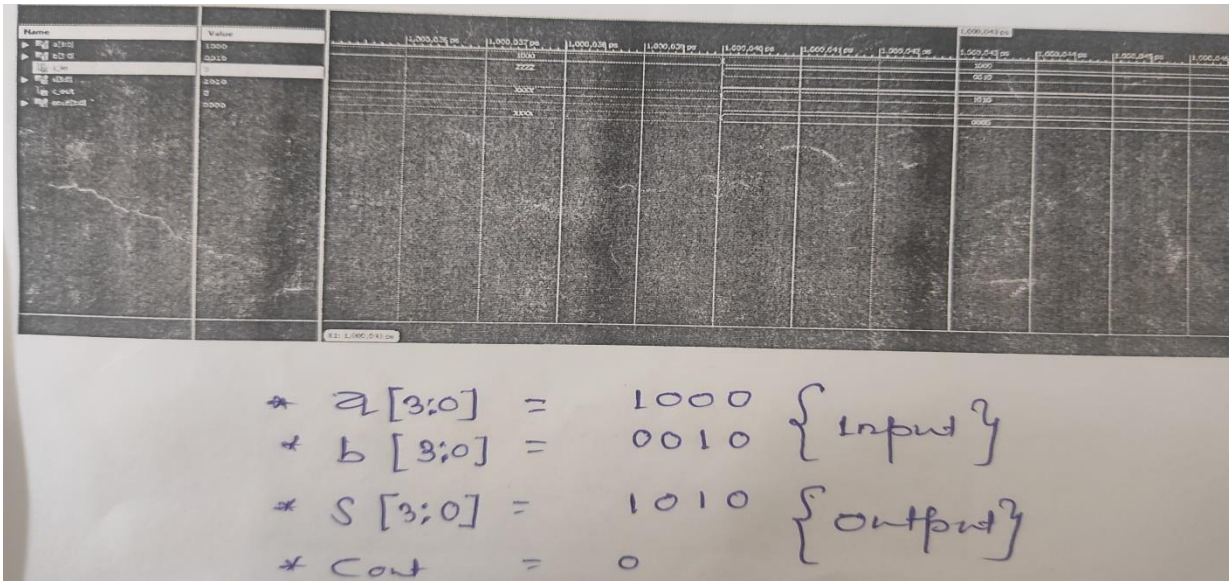
```
module fulladder(input wire[3:0] a,b,input c_in,output wire [3:0] s,output c_out);
    wire[3:0] cout;
    fa f1(a[0],b[0],c_in,s[0],cout[0]);
    fa f2(a[1],b[1],cout[0],s[1],cout[1]);
    fa f3(a[2],b[2],cout[1],s[2],cout[2]);
    fa f4(a[3],b[3],cout[2],s[3],cout[3]);
    assign c_out=cout[3];

endmodule
```

RTL views





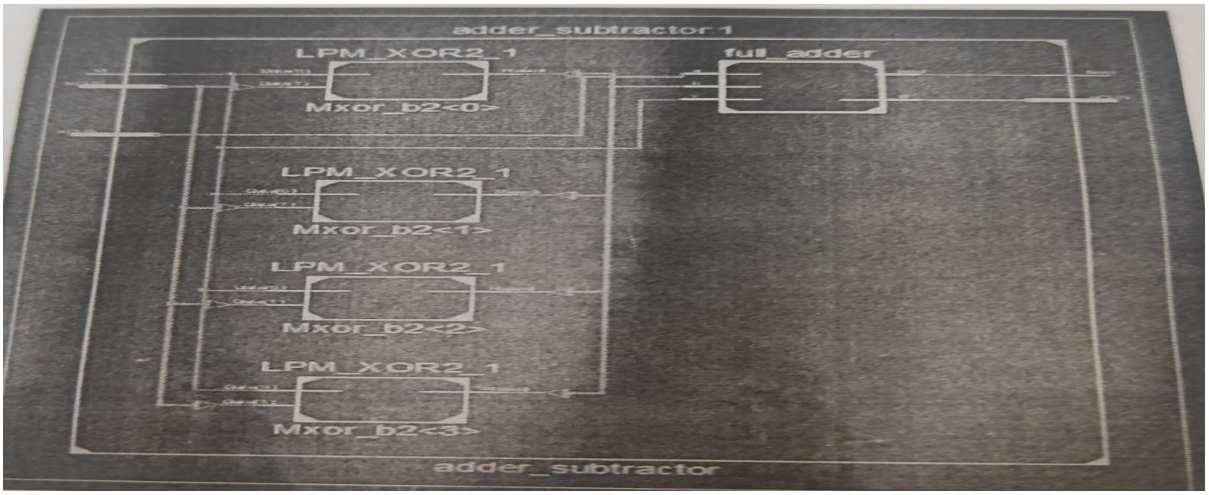
Output Simulation:



Verilog code for 4 bit Adder/Subtractor:

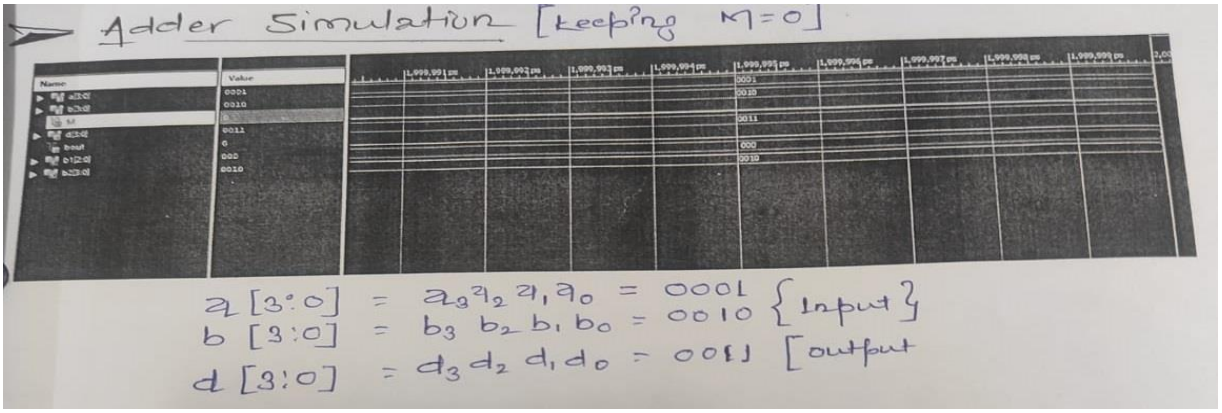
```
module adder_subtractor(a,b,M,d,bout);
 input [3:0] a;
 input [3:0] b;
 input wire M;
 output [3:0] d;
 output bout;
 wire [2:0] bl;
 wire [3:0] b2;
 assign b2[0]=b[0]^M;
 assign b2[1]=b[1]^M;
 assign b2[2]=b[2]^M;
 assign b2[3]=b[3]^M;
       full_adder f1(a[0],b2[0],M,d[0],bl[0]);
       full_adder f2(a[1],b2[1],bl[0],d[1],bl[1]);
       full_adder f3(a[2],b2[2],bl[1],d[2],bl[2]);
       full_adder f4(a[3],b2[3],bl[2],d[3],bout);
 endmodule
 module full_adder (a,b,c,d,bout);
 input a;
 input b;
 input c;
 output d;
 output bout;
 assign d=a^b^c;
 assign bout=(a&b)|(b&c)|(c&a);
 endmodule
```

RTL view:

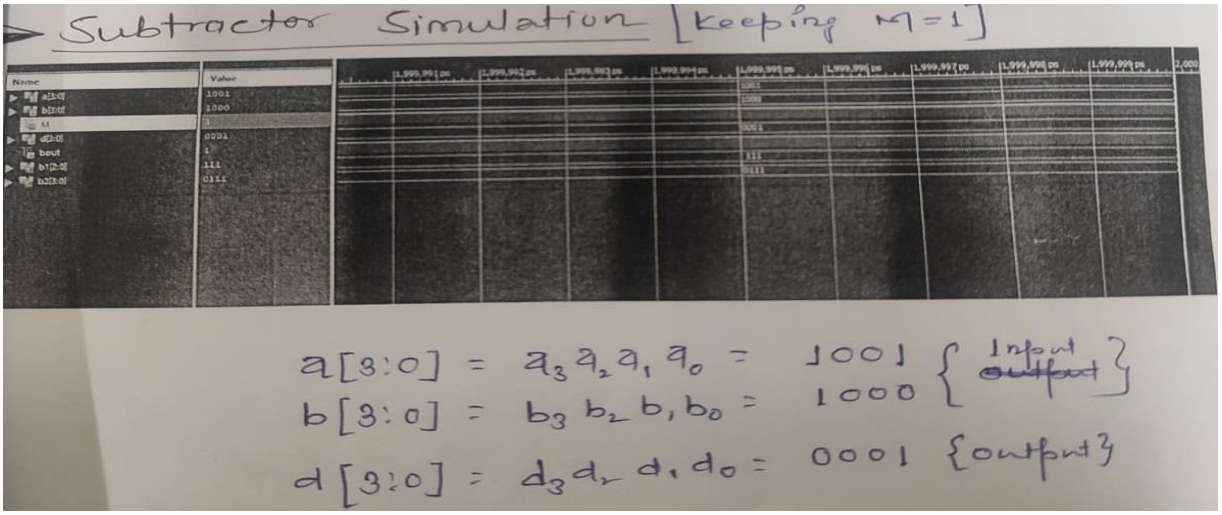


Output Simulation:

Adder simulation:



Subtractor simulation:



Results: Verilog codes of implement 4 bit addition/subtraction is simulated &verified.

EXPERIMENT–4

Objective: Write a program to implement 4- bit Comparator

Resources Required:

Hardware Requirement: Computer

Software Requirement: XILINX 14.7 Software

Theory:

A **digital comparator** is a hardware electronic device that takes two numbers as input in binary form and determines whether one number is greater than, less than or equal to the other number.

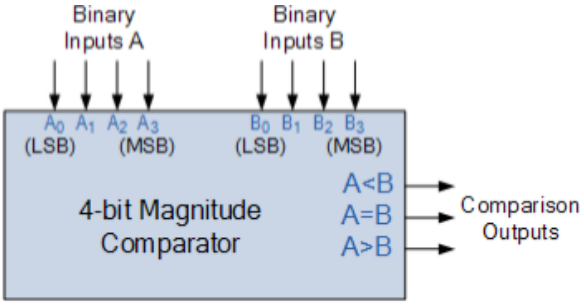


Figure 5: Logical Diagram of 4 bit Comparator

Truth Table:

Inputs				Outputs		
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	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

It can be used to compare two four-bit words. The two 4-bit numbers are A = A3 A2 A1 A0 and B3 B2 B1 B0 where A3 and B3 are the most significant bits.

It compares each of these bits in one number with bits in that of other number and produces one of the following outputs as A = B, A < B and A>B. The output logic statements of this converter are

Boolean Expression

- If A3 = 1 and B3 = 0, then A is greater than B (A>B). Or
- If A3 and B3 are equal, and if A2 = 1 and B2 = 0, then A > B. Or
- If A3 and B3 are equal & A2 and B2 are equal, and if A1 = 1, and B1 = 0, then A>B. Or

- If A3 and B3 are equal, A2 and B2 are equal and A1 and B1 are equal, and if A0 = 1 and B0 = 0, then A > B.
- the output A > B logic expression can be written as

$G = A3\,\overline{B3} + (A3 \text{ Ex-NOR } B3)\,A2\,\overline{B2} + (A3 \text{ Ex-NOR } B3)\,(A2 \text{ Ex-NOR } B2)\,A1\,\overline{B1} + (A3 \text{ Ex-NOR } B3)\,(A2 \text{ Ex-NOR } B2)\,(A1 \text{ Ex-NOR } B1)\,A0\,\overline{B0}$

Similarly the logic expression for the L or A<B output can be expressed as

$L = \overline{A3}\,B3 + (A3 \text{ Ex-NOR } B3)\,\overline{A2}\,B2 + (A3 \text{ Ex-NOR } B3)\,(A2 \text{ Ex-NOR } B2)\,\overline{A1}\,B1 + (A3 \text{ Ex-NOR } B3)\,(A2 \text{ Ex-NOR } B2)\,(A1 \text{ Ex-NOR } B1)\,\overline{A0}\,B0$

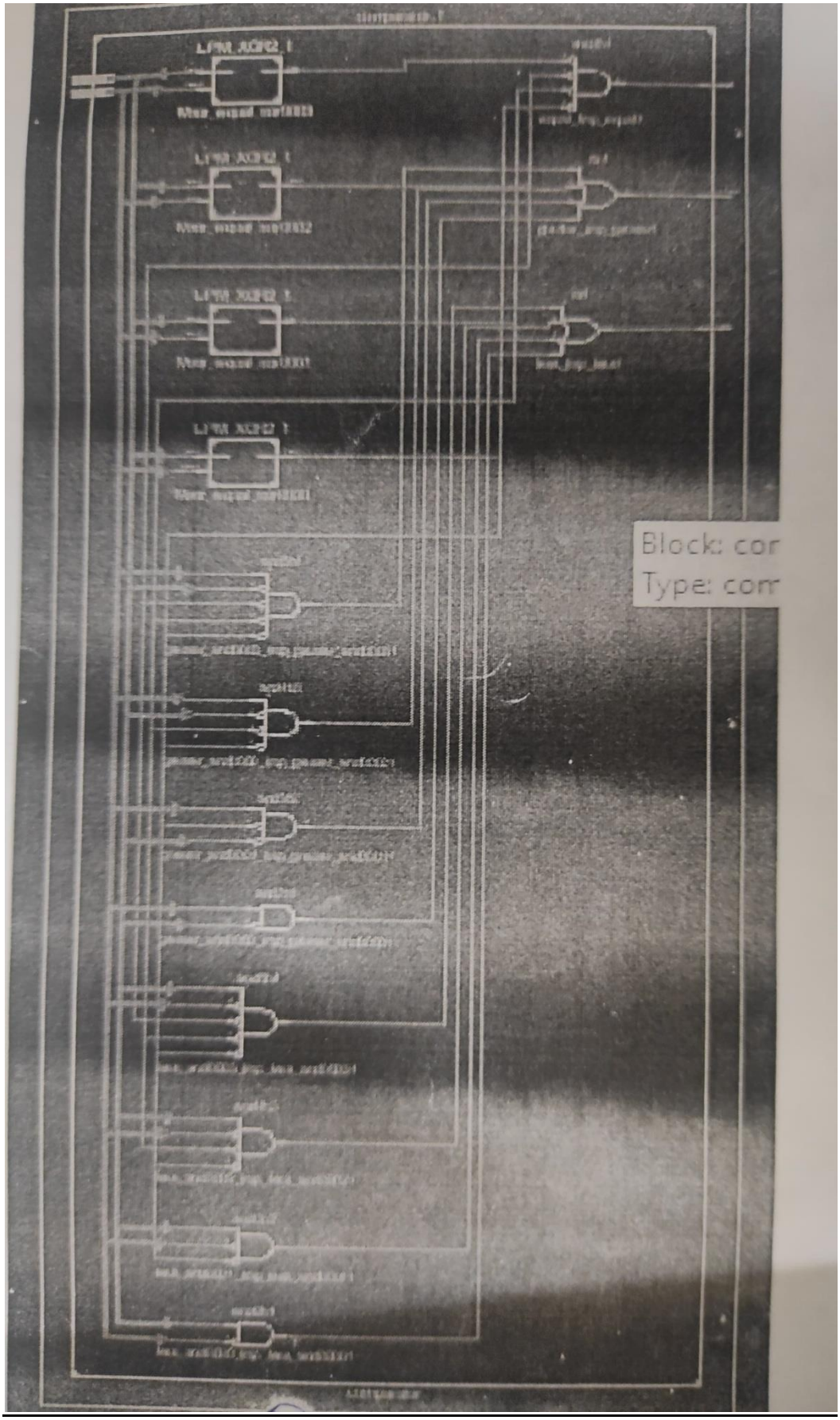
- The equal output is produced when all the individual bits of one number are exactly coincides with corresponding bits of another number. Then the logical expression for A=B output can be written as
- E = (A3 Ex-NOR B3) (A2 Ex-NOR B2) (A1 Ex-NOR B1) (A0 Ex-NOR B0)

Application: Comparators are used in central processing unit s (CPUs) and microcontrollers (MCUs). Examples of digital comparator include the CMOS 4063 and 4585 and the TTL 7485 and 74682-'89.

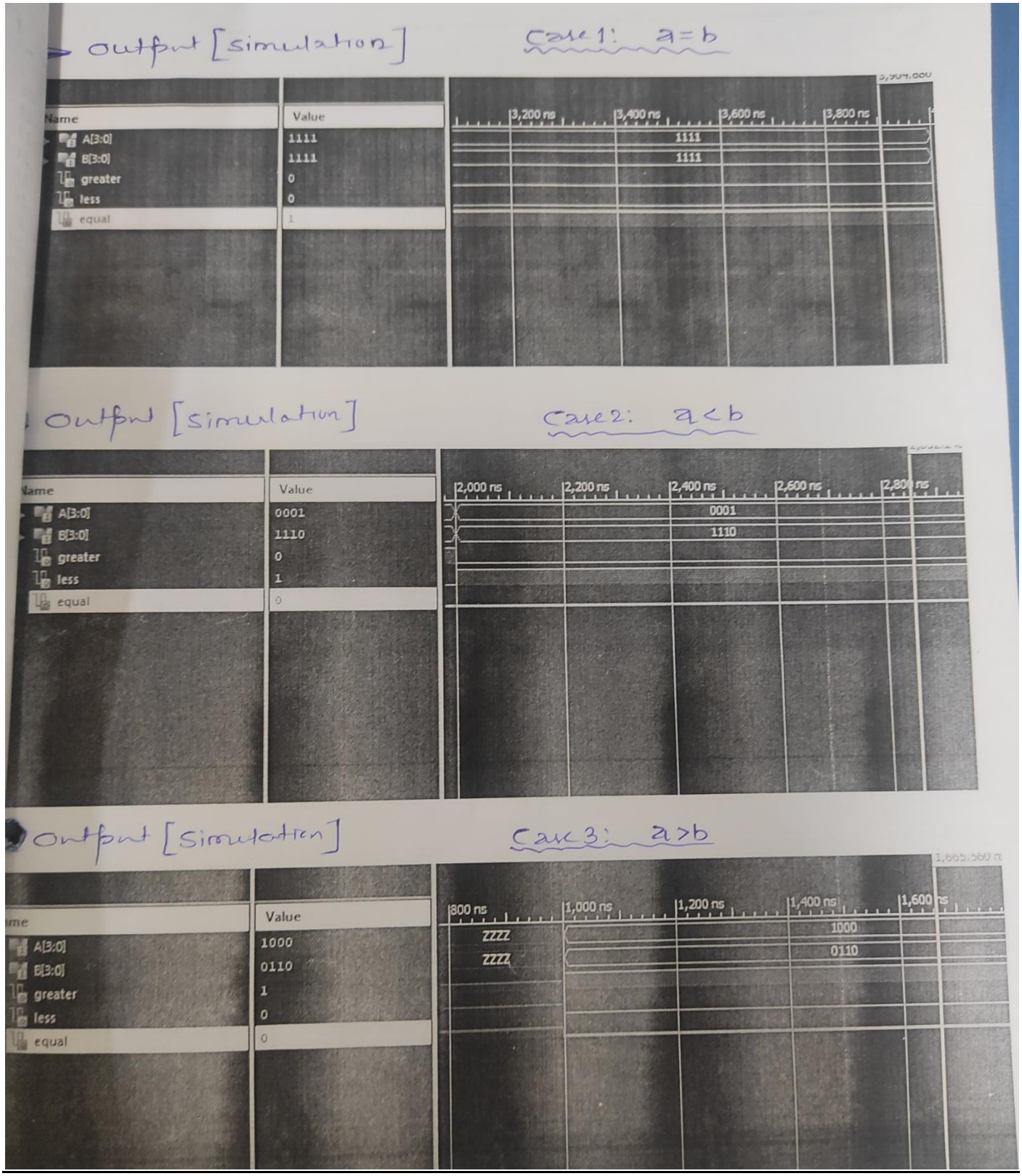
Verilog Code:

```
module comparator(
input [3:0] A,
input [3:0] B,
output greater,
output less,
output equal
);
assign equal=(A[3]~^B[3])&(A[2]~^B[2])&(A[1]~^B[1])&(A[0]~^B[0]);

assign greater=(A[3]&~B[3]) |
((A[3]~^B[3])&(A[2]&~B[2])) |
(((A[3]~^B[3])&(A[2]~^B[2]))&(A[1]&~B[1])) |
(((A[3]~^B[3]) & (A[2]~^B[2])) & ((A[1]~^B[1])&(A[0]&~B[0])));

assign less=(~A[3]&B[3]) |
((A[3]~^B[3])&(~A[2]&B[2])) |
(((A[3]~^B[3])&(A[2]~^B[2]))&(~A[1]&B[1])) |
(((A[3]~^B[3])&(A[2]~^B[2])) & ((A[1]~^B[1])&(~A[0]&B[0])));

endmodule
```

RTL Design:



Output simulation:



Result: Verilog codes of implement 4 bit comparator is simulated &verified.

EXPERIMENT-5

Objective: Write Verilog Code for SR, D, JK, and T Flip flops.

Resource Required:

Hardware Requirement: Computer

Software Requirement: XILINX 14.7 Software

Theory:

A flip-flop is a bi-stable electronic circuit. Flip-flop has 2 stable output states,0 and 1.The stable state of a flip-flop can be changed only by changing the set of its inputs.

S-R Flip-flop:

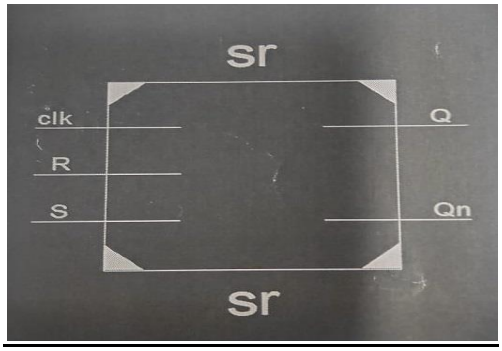
is basically a device that has 2 inputs along with the clock and 2 outputs, one output being the complement of the other. 2 inputs are called Set and Reset.The clocked RS flip flop has additional input called clock which control the action of flip-flop.When clock goes low, we get a stable output. Circuit will operate as an RS flip-flop only when clock goes high.

Truth table of S-R Flip Flop:

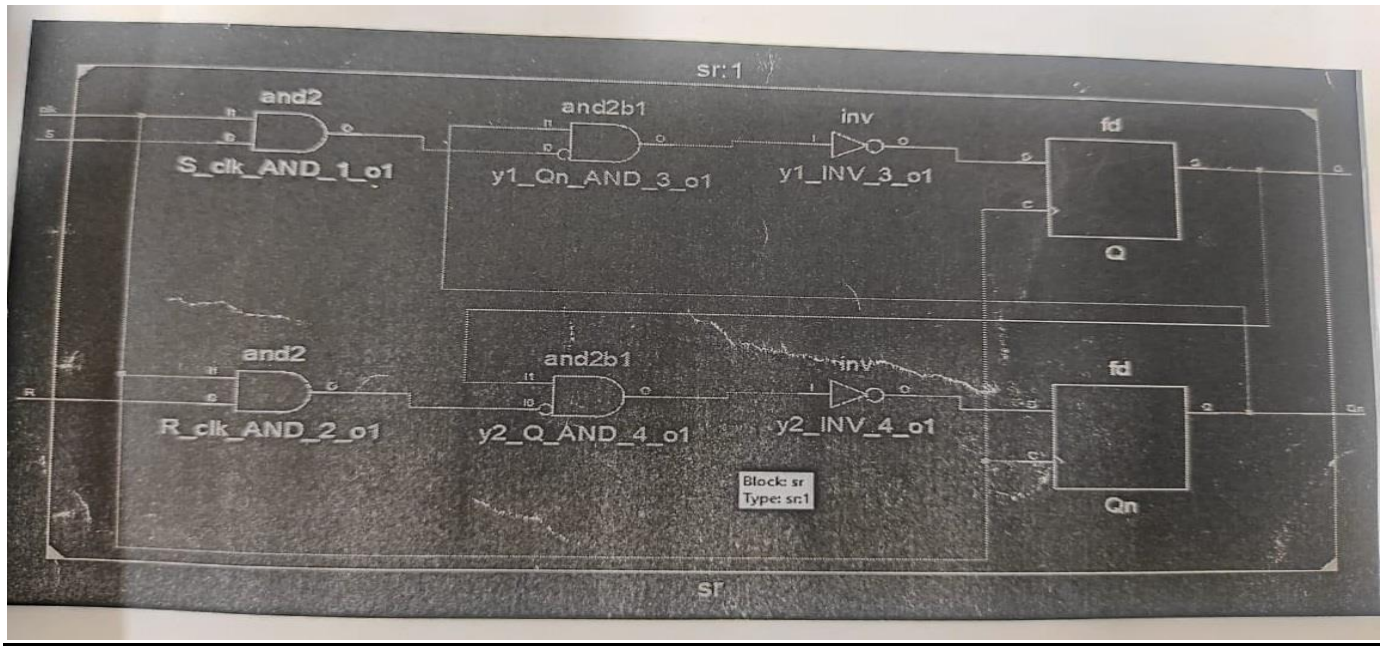
CLK	S	R	Q_{N+1}	$Q_{N+1\ BAR}$
0	×	×	Q_N	$Q_{N\ BAR}$
1	0	0	Q_N	$Q_{N\ BAR}$
1	0	1	0	1
1	1	0	1	0
1	1	1	×	×

Verilog code for SR flip-flop:

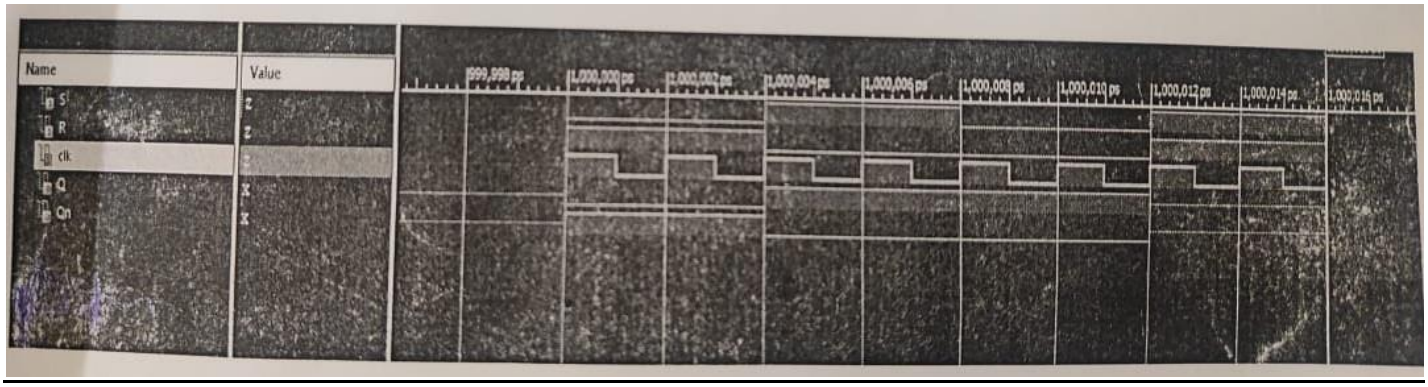
```
module sr(input wire S,R,clk,output reg Q,Qn
);
always@ (posedge clk)
begin
case ({S,R})
2'd0:;
2'd1:
begin
Q<=1'b0;
Qn<=1'b1;
end
2'd2:
begin
Q<=1'b1;
Qn<=1'b0;
end
2'd3:
begin
Q<=1'bx;
Qn<=1'bx;
end
endcase
end

endmodule
```



RTL schematic:



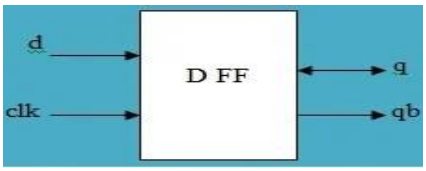
Output Simulation:



D Flip Flop:-

The D Flip-Flop is very useful when a signal bit is to be stored. It can be developed from RS flip flop using a signal inverter.
The flip flop has a1 i/p & a clock.Case1: When D=0 S=0, R=1Q=0. The flip-flop is in Reset state.Case 2: When D=1 S=1, R=0So Q=1 hence flip-flop is in Set state.Q will always follow D input after some time delay. Hence it is also called as Delay flip-flop

D Flipflop Symbol : Following is the symbol and truth table of **D flipflop**.

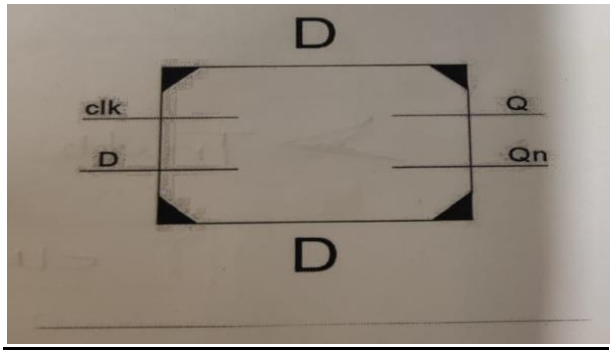


D Flipflop Truth Table

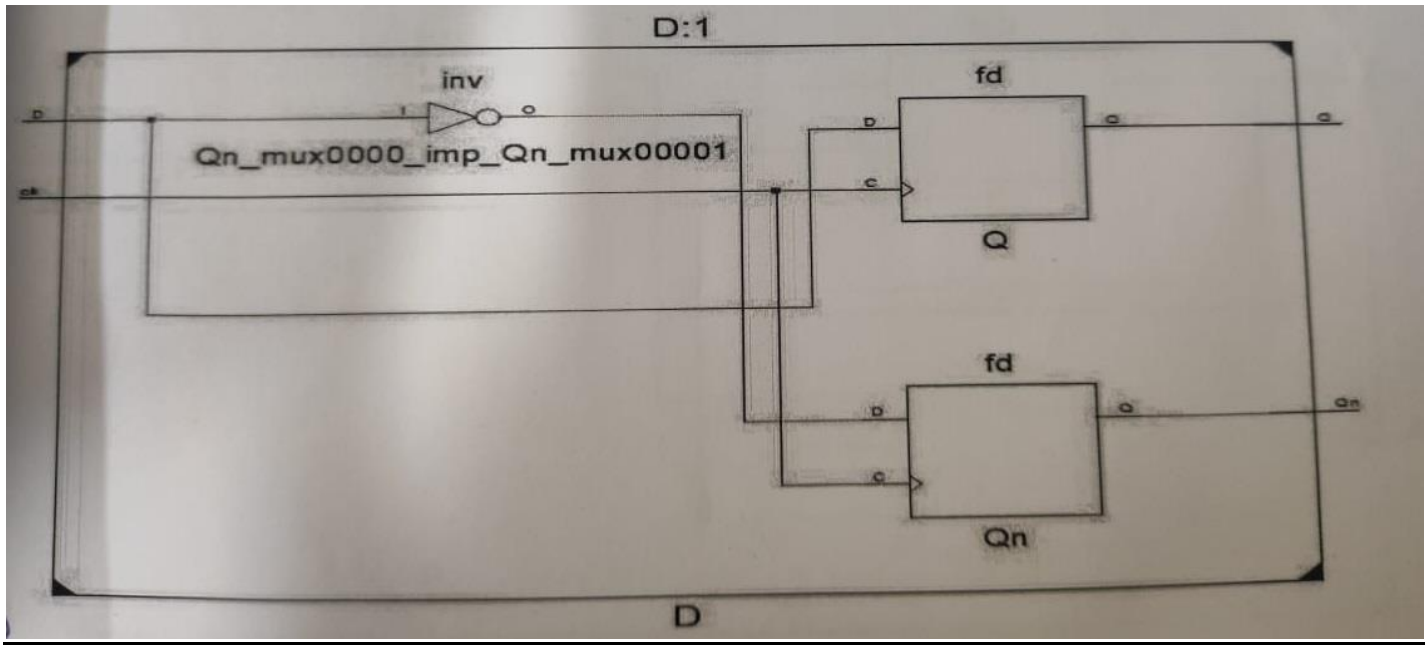
Clk	D	Q	Qb
X	1	1	0
1	1	1	0
1	0	0	1

D Flipflop Verilog Code:

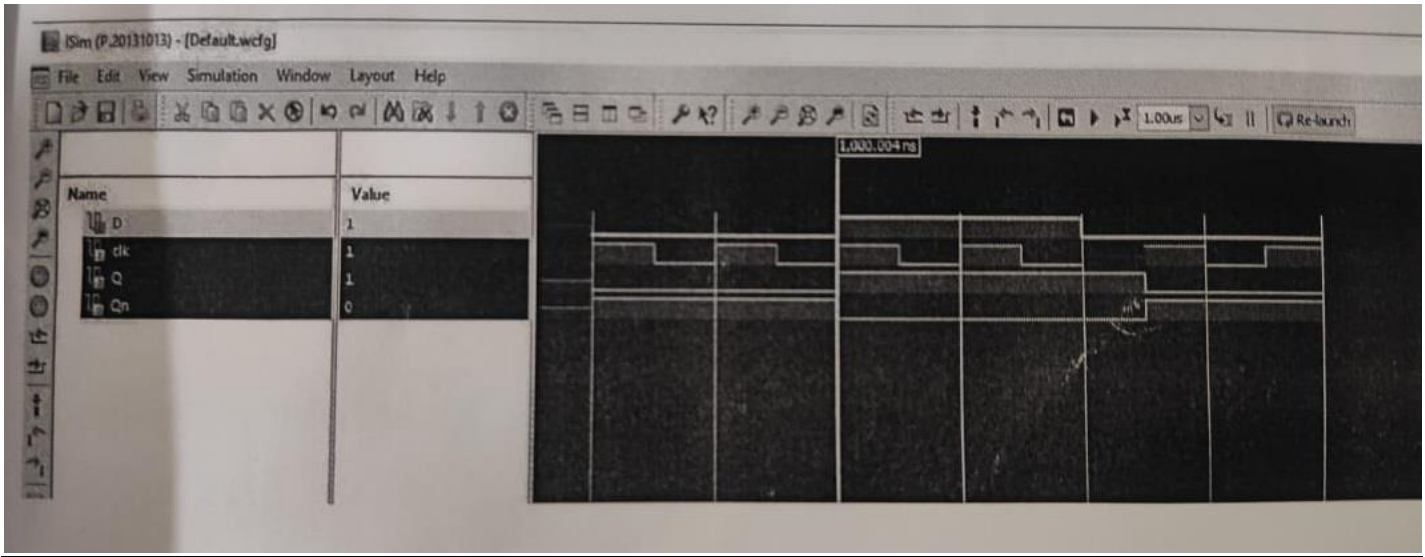
```
module D(input wire D,clk, output reg Q,Qn

);
always@ (posedge clk)
begin
case (D)
1'b0:
begin
Q<=0;
Qn<=1;
end
1'b1:
begin
Q<=1;
Qn<=0;
end
endcase
end
endmodule
```



RTL schematic:



Output simulation:

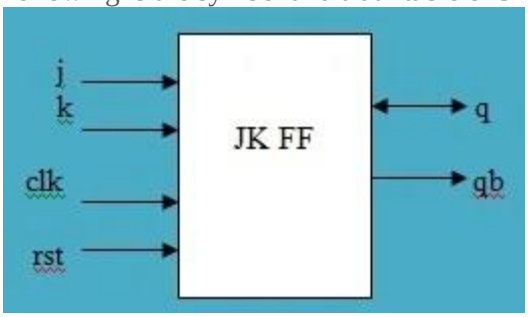


J-K Flip-flop:-

In JK flip-flop, output Q is fed as an additional input to gate A and Q is fed as an additional input togate B apart from JK inputs and clock.Case 1: When J & K both are low both the NAND gate are disabled, c/k pulse have no effect. The o/pof Q retains its last value i.e. No change in the outputs
Case 2: If 'J' is low &'K 'is high the upper gate is disabled.
So flip-flop is in Reset condition.When J is high & K is low,Q=1 and Q-0 i.e. flip-flop is in Set state.Case 4: When J & K is high, Q and Q are inverted. This is called as toggling mode. That means the flipflop will be toggle.

JK Flipflop Symbol

Following is the symbol and truth table of **JK flipflop**.

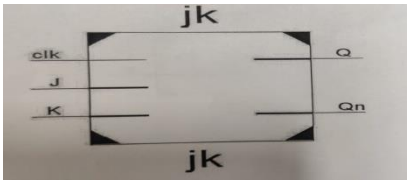


JK Flipflop Truth Table

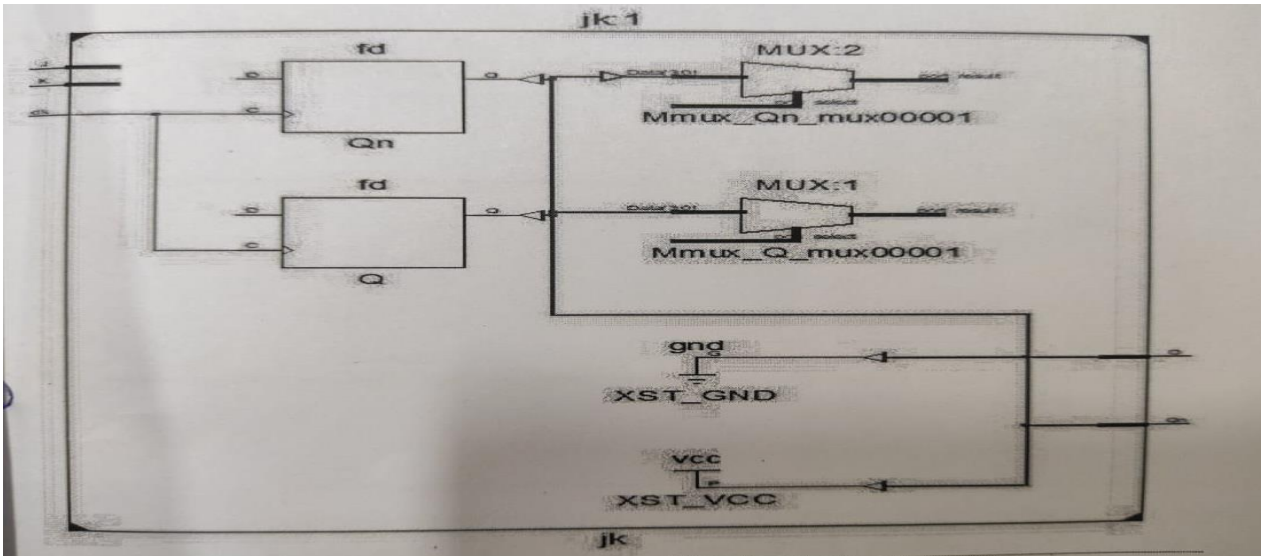
Reset	CLK	J	K	Q	Qn
1	1	0	0	Previous	state
1	1	0	1	0	1
1	1	1	0	1	0
1	1	1	1	Qb	Q
1	No +ve edge	-	-	Previous	state
0	-	-	-	0	1

JK Flipflop Verilog Code:

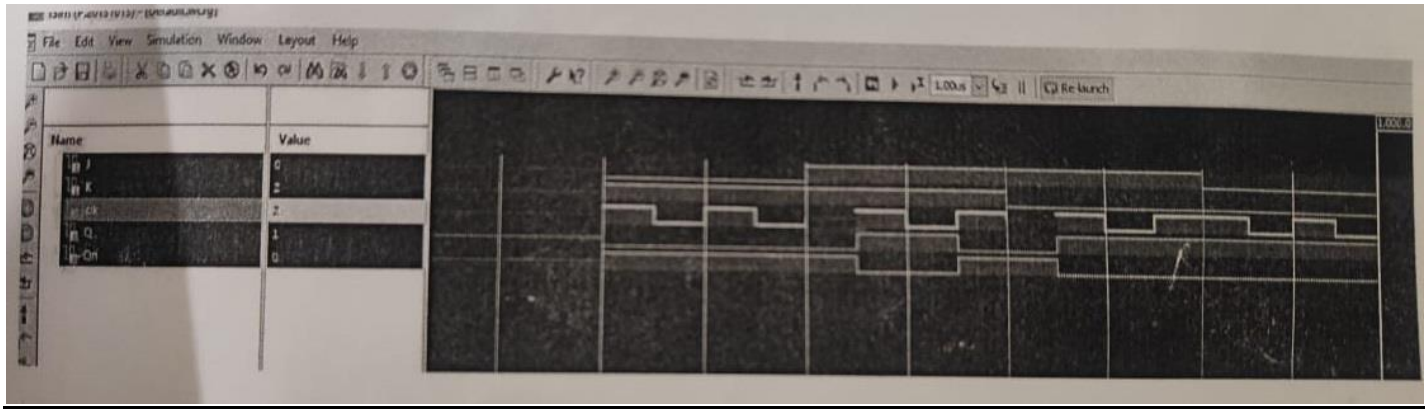
```
module jk(input wire J, K, clk, output reg Q, Qn
);
always@(posedge clk)
begin
case ({J,K})
2'd0:;
2'd1:
begin
Q<=0;
Qn<=1;
end
2'd2:
begin
Q<=1;
Qn<=0;
end
2'd3:
begin
Q<=Qn;
Qn<=Q;
end
endcase
end
endmodule
```



RTL schematic:



Output simulation:



T Flip flop:

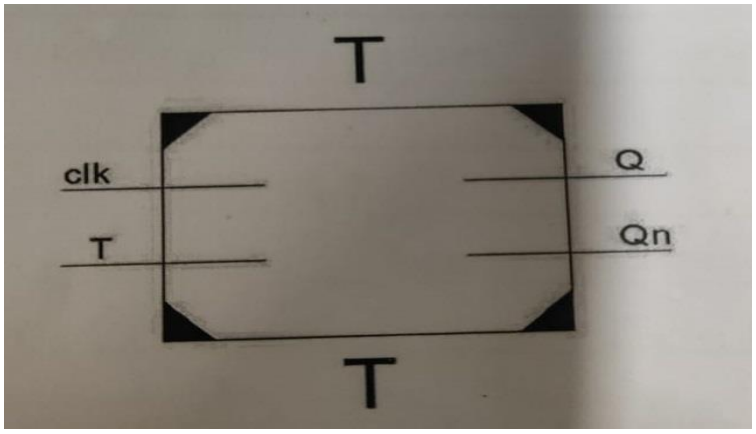
T Flip Flops used to create memory which are used to store data, when the power is turned off. Synchronous logic circuits: T flip-flops can be used to implement synchronous logic circuits, which are circuits that perform operations on binary data based on a clock signal.

T Flipflop Truth Table

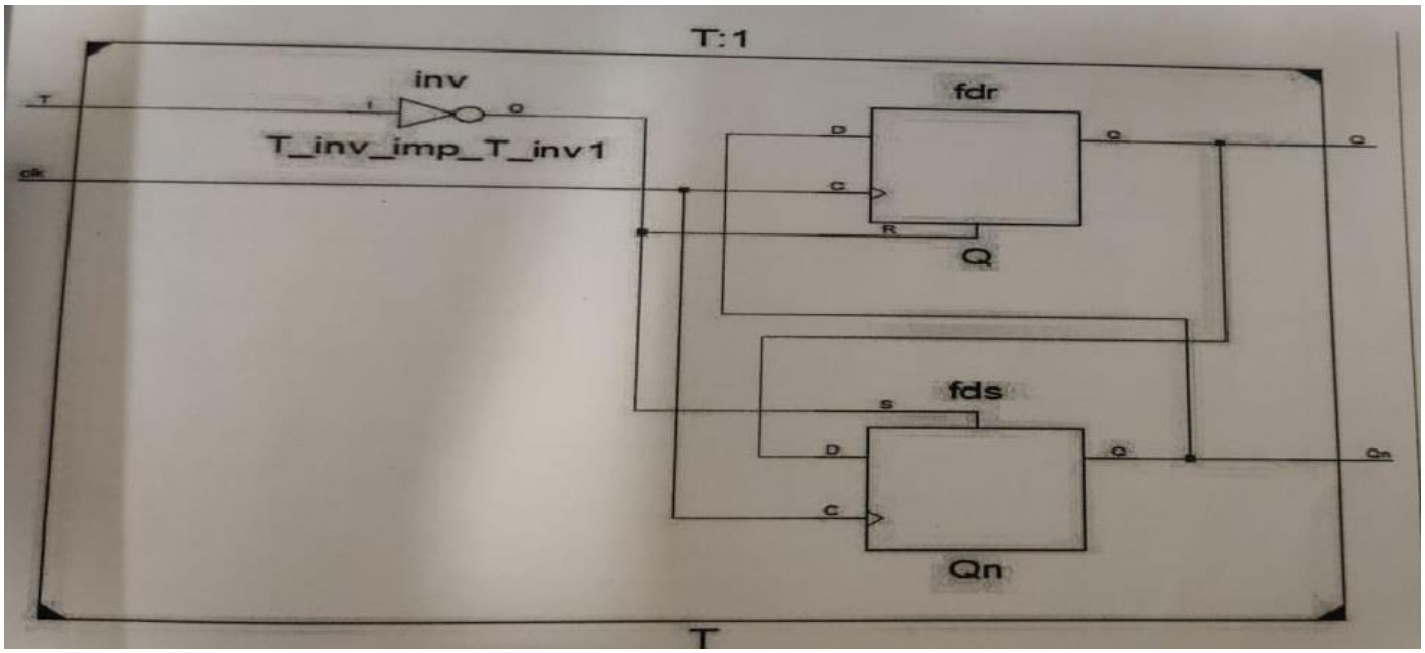
RST_N	T	CLK	Q
1	0	1	q
1	1	1	qb
1	X	No positive edge	Previous state
0	X	X	0

T Flipflop Verilog Code:

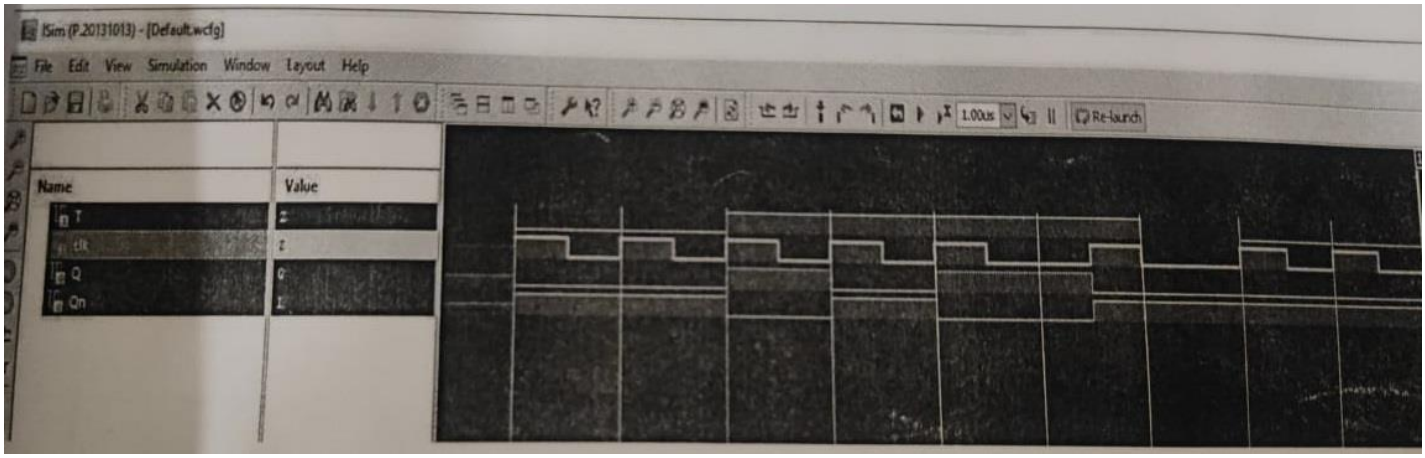
```
module T(input wire T,clk, output reg Q, Qn
);
always@ (posedge clk)
begin
Q<=1'b0;
Qn<=1'b1;
case (T)
1'b0: ;
1'b1:
begin
Q<=Qn;
Qn<=Q;
end
endcase
end
endmodule
```



RTL schematic:



Output simulation:



Result: Verilog codes for SR, D, JK and T flip flops are simulated &verified.

EXPERIMENT-6

Objective: Write a program to generate Mod-10 Up Counter

Resource Required: Xilinx ISE 14.7.

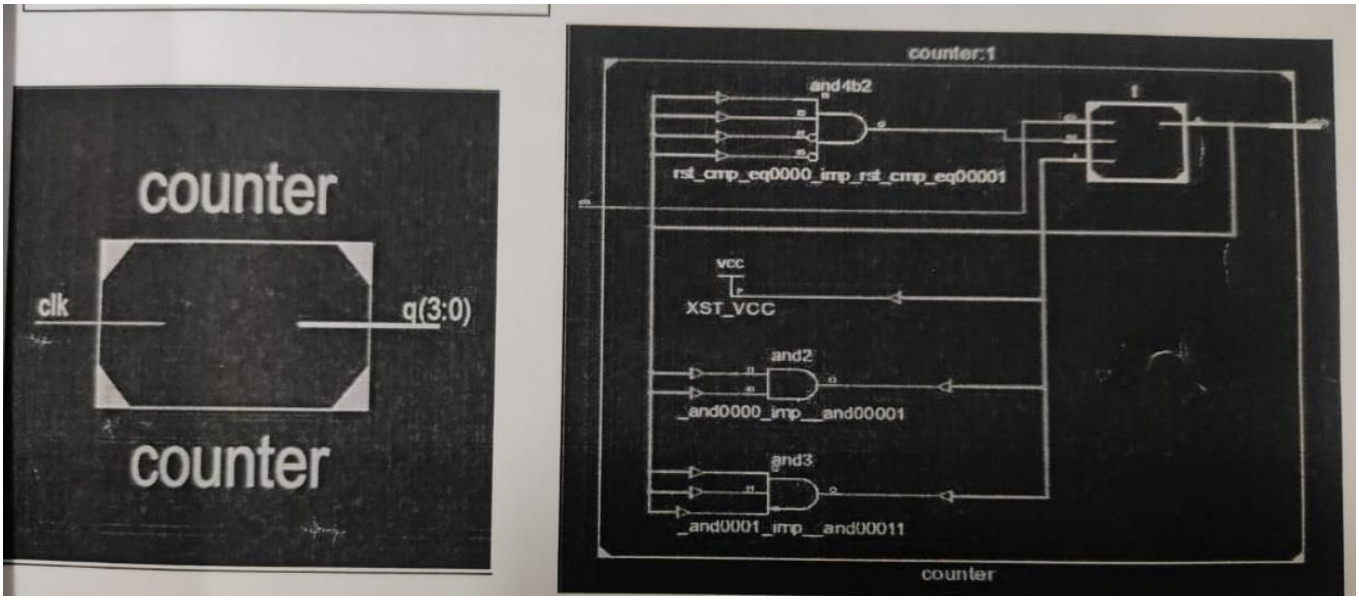
Theory: Counters are sequential logic devices that follow a predetermined sequence of counting states which are triggered by an external clock (CLK) signal. The number of states or counting sequences through which a particular counter advances before returning once again back to its original first state is called the **modulus** (MOD). In other words, the modulus (or just modulo) is the number of states the counter counts and is the dividing number of the counter. The decade counter has four outputs producing a 4-bit binary number, it receives an input clock pulse, one by one, and counts up from 0 to 9 repeatedly.

Application: It suitable for human interfacing where a digital display is required.

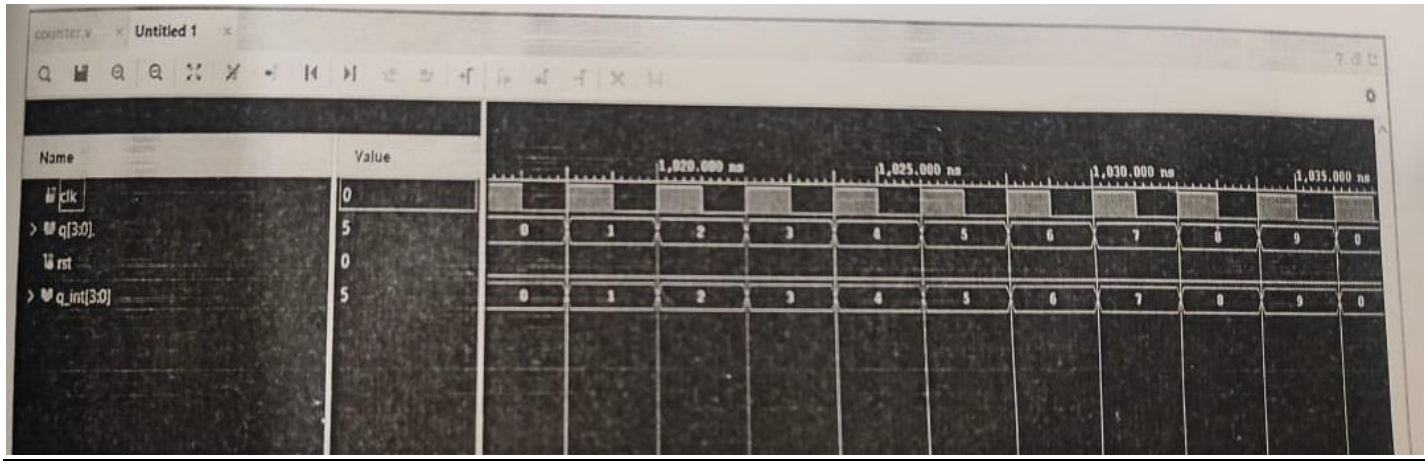
Verilog code:

```
module (input t,clk,rst, output reg q);
always @(posedge clk or posedge rst)
begin
if (rst)
q<= 1'b0;
else if (t)

q<=1’b0;
end
endmodule
module counter (input clk, output[3:0] q
);
wire rst;
wire[3:0] q_int;

t T1(.t(1'b1),.clk(clk),.rst(rst),.q(q_int[0]));
t T2(.t(q_int[0]),.clk(clk),.rst(rst),.q(q_int[1]));
T3(.t(q_int[1]&q_int[0]),.clk(clk),.rst(rst),.q(q_int[2]));
t T4(.t(q_int[2]&q_int[1]&q_int[0]),.clk(clk),.rst(rst),.q(q_int[3]));

assign rst=(q_int==4'b1010)?1'b1:1'b0;
assign q=q_int;

endmodule
```

RTL schematic:



OUTPUT:



Results: Verilog codes of Mod-10 up counter is simulated &verified.

EXPERIMENT-7

Objective: Write a program to generate the 1010 sequence detector. The overlapping patterns are allowed
Resources Required: Xilinx ISE 14.7.

Theory:

The objective of this experiment is to introduce the use of **sequential logic**. The sequence is a sequential circuit. In sequential logic the output depends on the current input values and also the previous inputs. When describing the behavior of a sequential logic circuit we talk about the **state** of the circuit. The state of a sequential circuit is a result of all previous inputs and determines the circuit's output and future behavior. This is why sequential circuits are often referred to as *state machines*. Most sequential circuits (including our sequence detector) use a clock signal to control when the circuit changes states. The inputs of the circuit along with the circuit's current state provide the information to determine the circuit's *next state*. The clock signal then controls the passing of this information to the *state memory*. The output depends only on the circuit's state, this is known as a *Moore Machine*. Figure 7.1 shows the schematic of a Moore Machine.

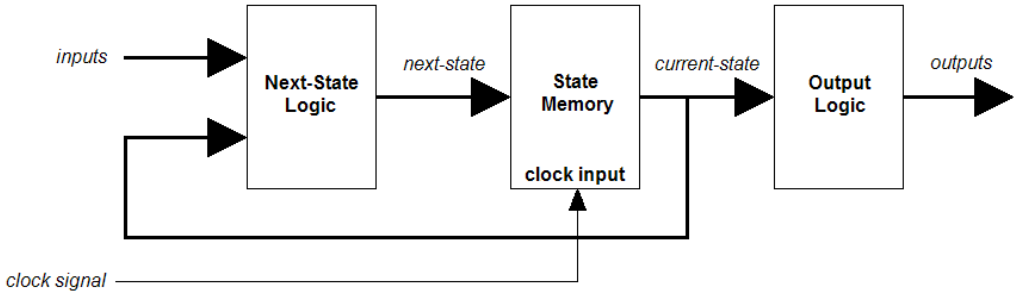


Figure 7.1 Schematic of a clocked synchronous state machine (Moore Machine).

A sequential circuit's behavior can be shown in a *state diagram*. The state diagram for our sequence detector is shown in figure 7.2. Each circle represents a state and the arrows show the transitions to the next state. Inside each circle are the state name and the value of the output. Along each arrow is the input value that corresponds to that transition.

State Diagram:

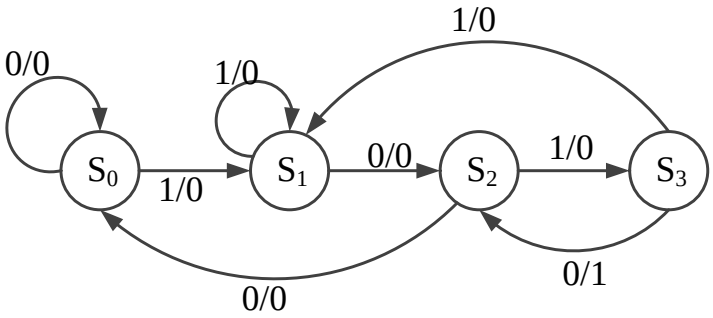


Figure 7.2 Sequence Detector state diagram.

A sequence detector accepts as input a string of bits: either 0 or 1. Its output goes to 1 when a target sequence has been detected. There are two basic types: overlap and non-overlap. In a sequence detector that allows overlap, the final bits of one sequence can be the start of another sequence.

Example:

11011 detector with overlap X 11011011011

Z 00001001**1**001

11011 detector with no overlap Z 00001000001

Applications:

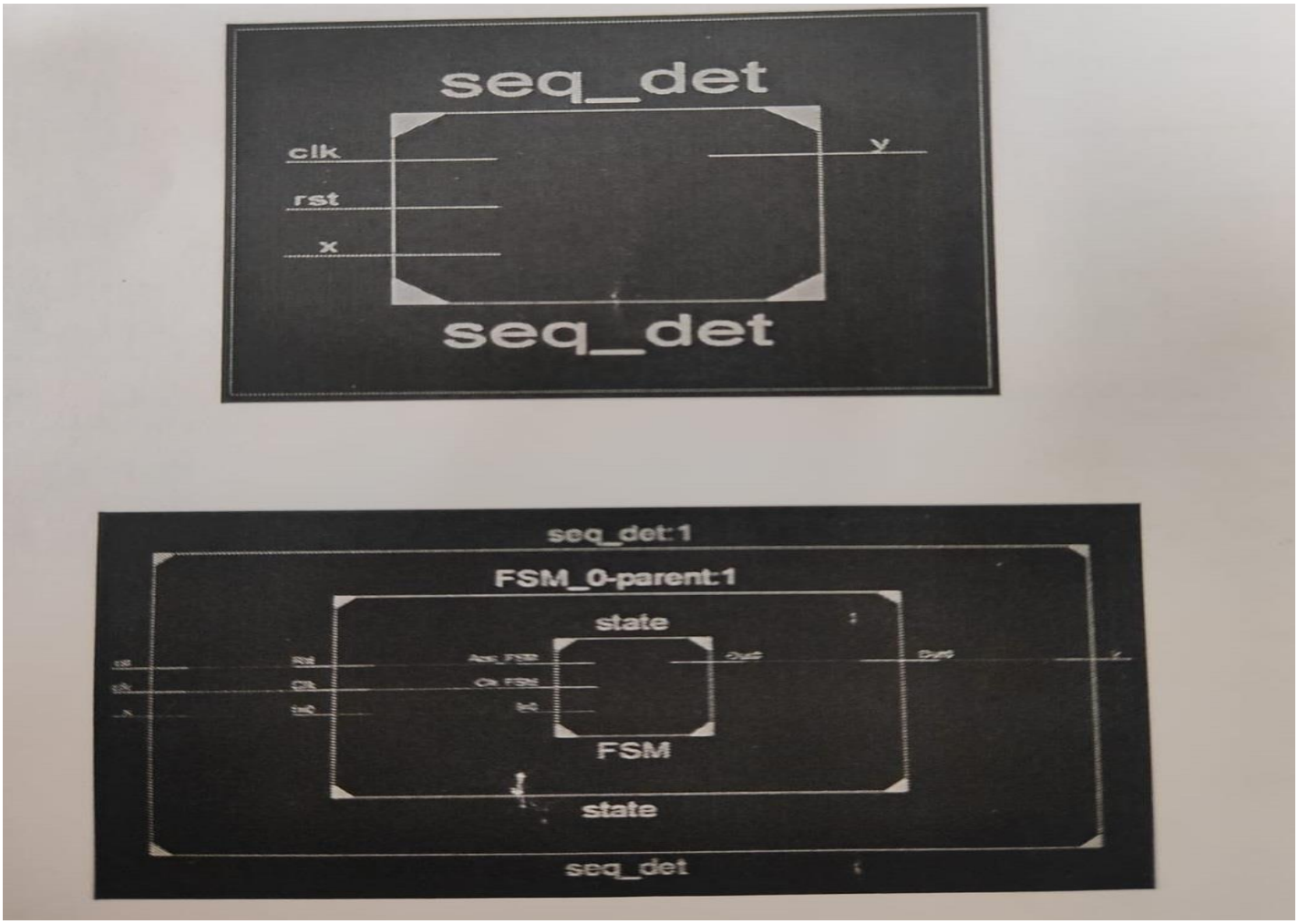
In a computer network like Ethernet, digital data is sent one bit at a time, at a very high rate. Such a movement of data is commonly called a *bit stream*. One characteristic is unfortunate, particularly that any one bit in a bit stream looks identical to many other bits. Clearly it is important that a receiver can identify important features in a bit stream. As an example, it is important to identify the beginning and ending of a message. This is the job of special bit sequences called *flags*. A flag is simply a bit sequence that serves as a marker in the bit stream. To detect a flag in a bit stream a *sequence detector* is used.

Verilog code:

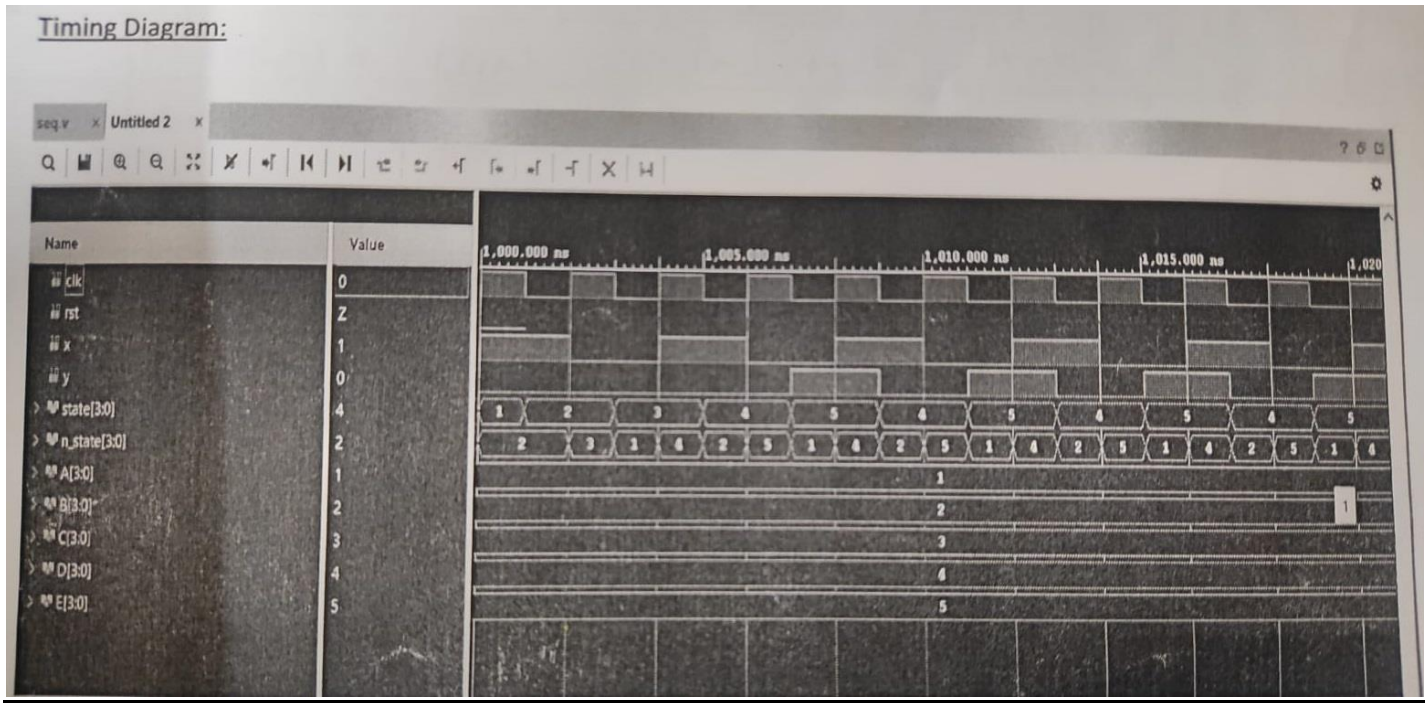
```
module seq(input clk, rst,x, output reg y);
// states defined using parameter
      parameter A=4'd1;
      parameter B=4'd2;
      parameter C=4'd3;
      parameter D=4'd4;
      parameter E=4'd5;
      reg [3:0] state ,n_state;
     always@ (negedge clk or negedge rst)
     begin
     if (!rst)
     state<=A;
     else
     state<=n_state;
     end
     //n_state comination logic
     always@(*)
     begin
     n_state=state;
     y<=0;
     case (state)
      A: begin
      if (x==1) n_state=B;
     else n_state=A;
     end
     B: begin
     if(x==1) n_state=B;
    else n_state=C;
    end
    C: begin
```

```
 if (x==1) n_state=D;
 else n_state=A;
 end
 D: begin
 if (x==1) n_state=B;
else n_state=E;
end
E: begin
if(x==1) n_state=D;
else n_state=A;
y<=1;
end
default:n_state=A;
endcase
end


endmodule
```

RTL schematic:



Output simulation:



Result: Thus, the Verilog code for the 1010 sequence detector circuit was simulated and verified.

EXPERIMENT –8 (a)

Objective: Write a program to perform **serial to parallel** of 4 bit binary number

Resources Required: Xilinx-ISE 14.7

Theory:

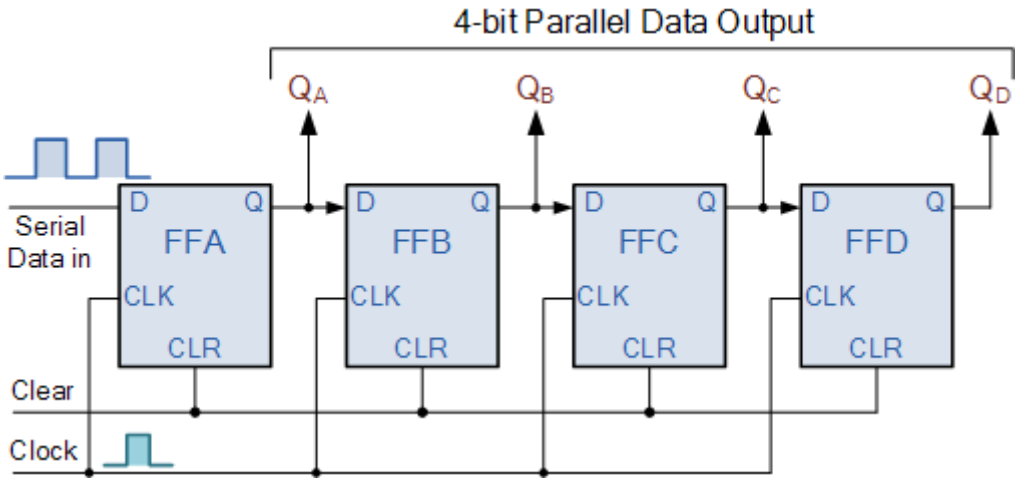


Figure 8.1 4-bit Serial-in to Parallel-out Shift Register

The operation is as follows. Let's assume that all the flip-flops (FFA to FFD) have just been RESET (CLEAR input) and that all the outputs Q_A to Q_D are at logic level "0" ie, no parallel data output.

If a logic "1" is connected to the DATA input pin of FFA then on the first clock pulse the output of FFA and therefore the resulting Q_A will be set HIGH to logic "1" with all the other outputs still remaining LOW at logic "0". Assume now that the DATA input pin of FFA has returned LOW again to logic "0" giving us one data pulse or 0-1-0.

The second clock pulse will change the output of FFA to logic "0" and the output of FFB and Q_B HIGH to logic "1" as its input D has the logic "1" level on it from Q_A. The logic "1" has now moved or been "shifted" one place along the register to the right as it is now at Q_A.

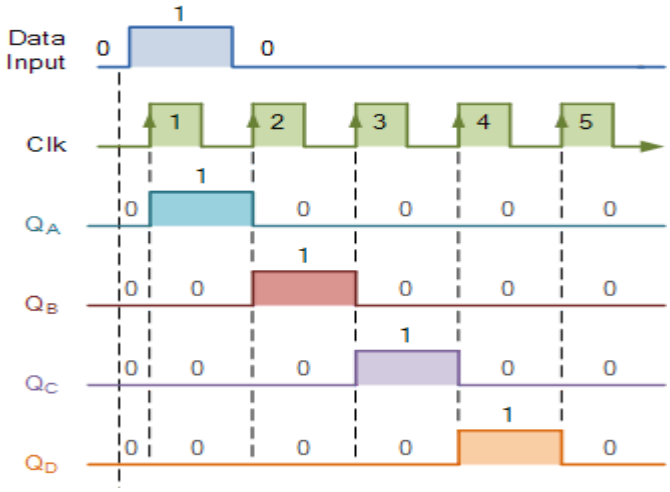
When the third clock pulse arrives this logic "1" value moves to the output of FFC (Q_C) and so on until the arrival of the fifth clock pulse which sets all the outputs Q_A to Q_D back again to logic level "0" because the input to FFA has remained constant at logic level "0".

The effect of each clock pulse is to shift the data contents of each stage one place to the right, and this is shown in the following table until the complete data value of 0-0-0-1is stored in the register. This data value can now be read directly from the outputs of $Q_A$to Q_D.

Then the data has been converted from a serial data input signal to a parallel data output. The truth table and following waveforms show the propagation of the logic "1" through the register from left to right as follows.

Basic Data Movement Through A Shift Register:

Clock Pulse No	Q_A	Q_B	Q_C	Q_D
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1
5	0	0	0	0

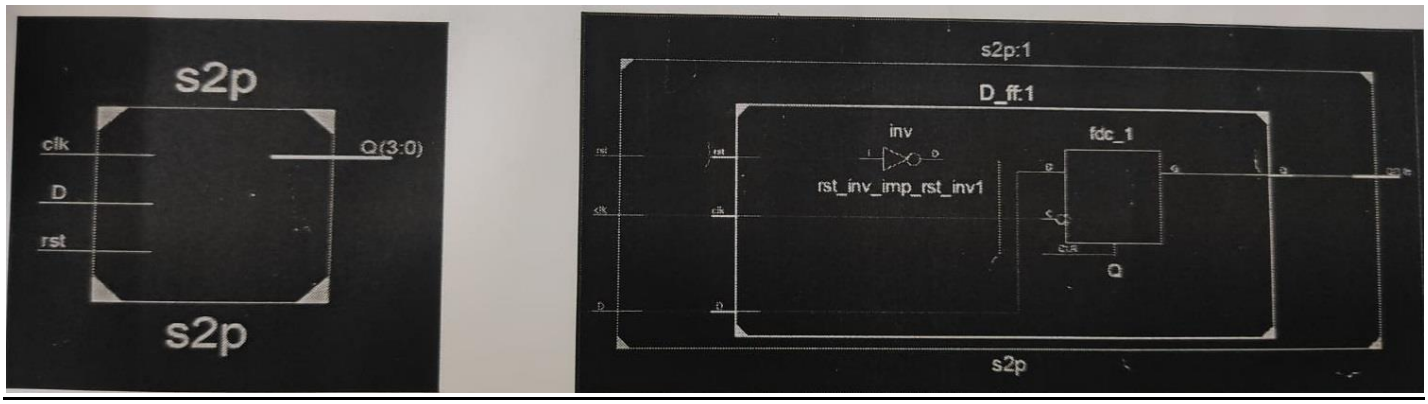


Note that after the fourth clock pulse has ended the 4-bits of data (0-0-0-1) are stored in the register and will remain there provided clocking of the register has stopped. In practice the input data to the register may consist of various combinations of logic "1" and "0". Commonly available SIPO IC's include the standard 8-bit 74LS164.

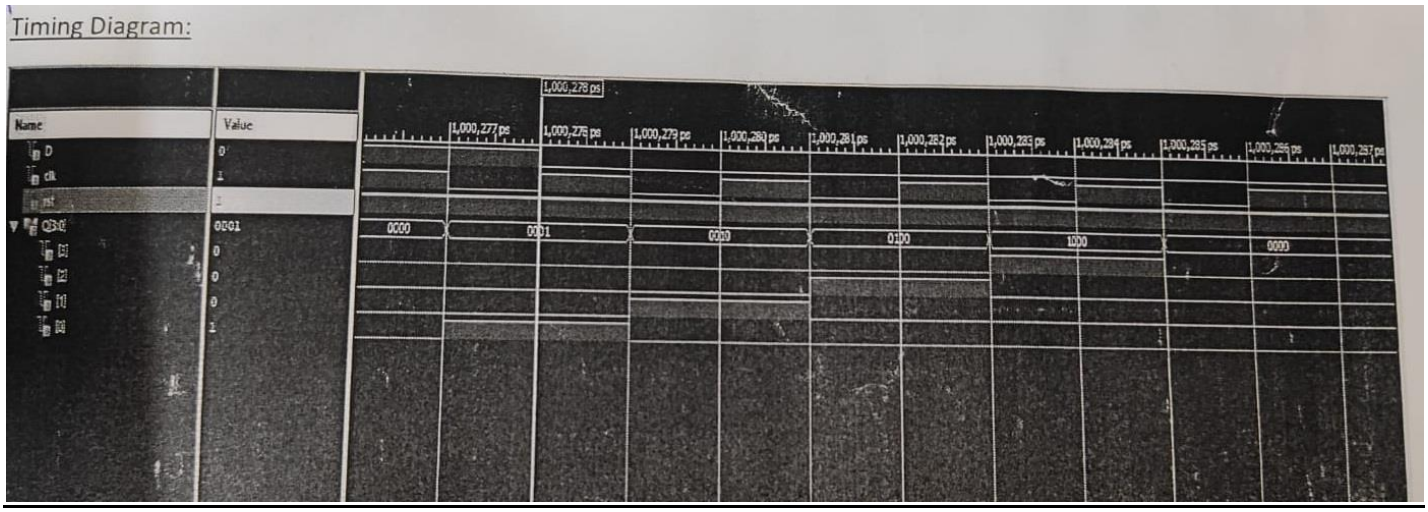
Verilog CODE:

```
module D_ff (input wire D,clk, rst, output reg Q);
always @(negedge clk or negedge rst)
begin
if (~rst)
Q<=1'b0;
else
Q<=D;
end
endmodule

module s2p (input wire D, clk,rst, output wire [3:0] Q
);
D_ff f1 (D,clk,rst, Q[0]);
D_ff f2 (Q[0],clk,rst,Q[1]);
D_ff f3 (Q[1],clk,rst,Q[2]);
D_ff f4 (Q[2],clk,rst,Q[3]);
endmodule
```

RTL schematic:



Output simulation:



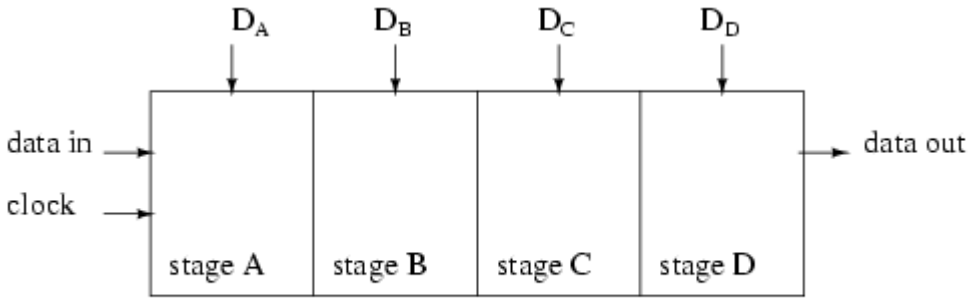
Result: Verilog code to implement serial to parallel of 4 bit binary number is simulated and verified.

EXPERIMENT-8(b)

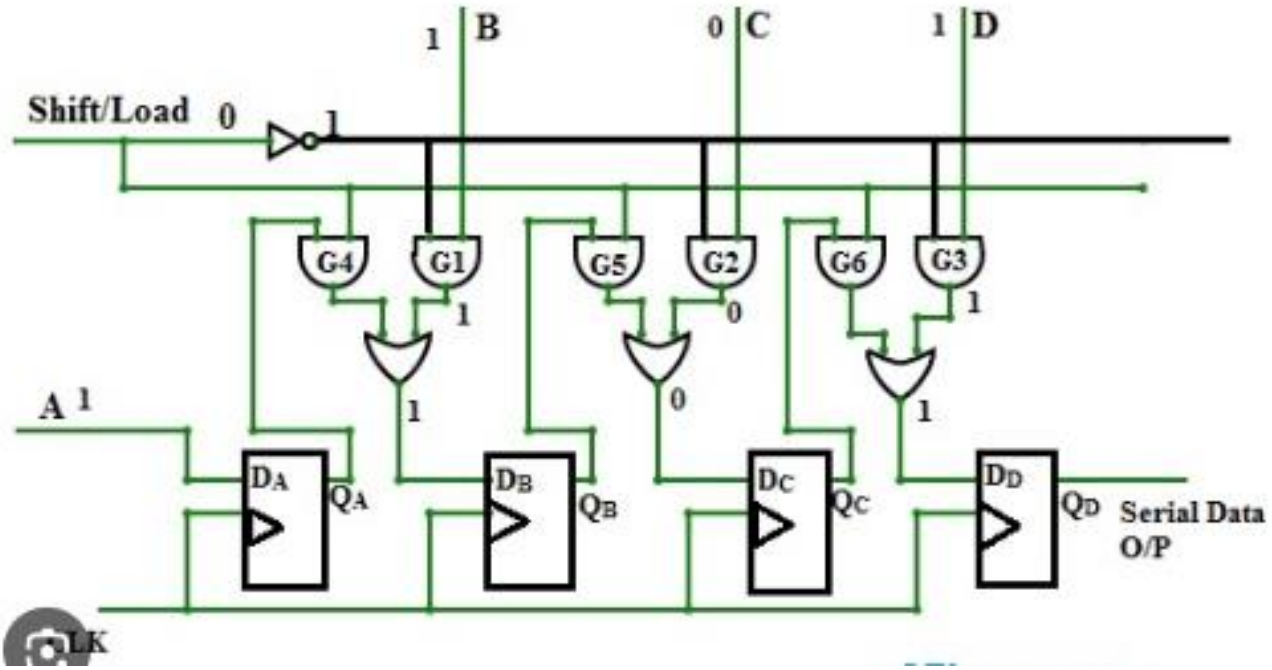
Objective: Write a program to perform **parallel to serial** transfer of 4 bit binary number
Resources Required: Xilinx-ISE 14.7

Theory:

Parallel-in/ serial-out shift registers do everything that the previous serial-in/ serial-out shift registers do plus input data to all stages simultaneously. The parallel-in/ serial-out shift register stores data, shifts it on a clock by clock basis, and delays it by the number of stages times the clock period. In addition, parallel-in/ serial-out really means that we can load data in parallel into all stages before any shifting ever begins. This is a way to convert data from a *parallel* format to a *serial* format. By parallel format we mean that the data bits are present simultaneously on individual wires, one for each data bit as shown below. By serial format we mean that the data bits are presented sequentially in time on a single wire or circuit as in the case of the "data out" on the block diagram below.



Parallel-in, serial-out shift register with 4-stages



Truth table:

CLK	SHIFT/LOAD	D0	D1	D2	D3	OUTPUT
0	1	0	0	0	0	0
1	1	0	0	0	0	0
1	0	1	0	1	1	0
1	1	0	1	0	1	1
1	1	0	0	1	0	1
1	1	0	0	0	1	0
1	1	0	0	0	0	1

Application:1.Bit serial operations can be performed quickly through device iteration

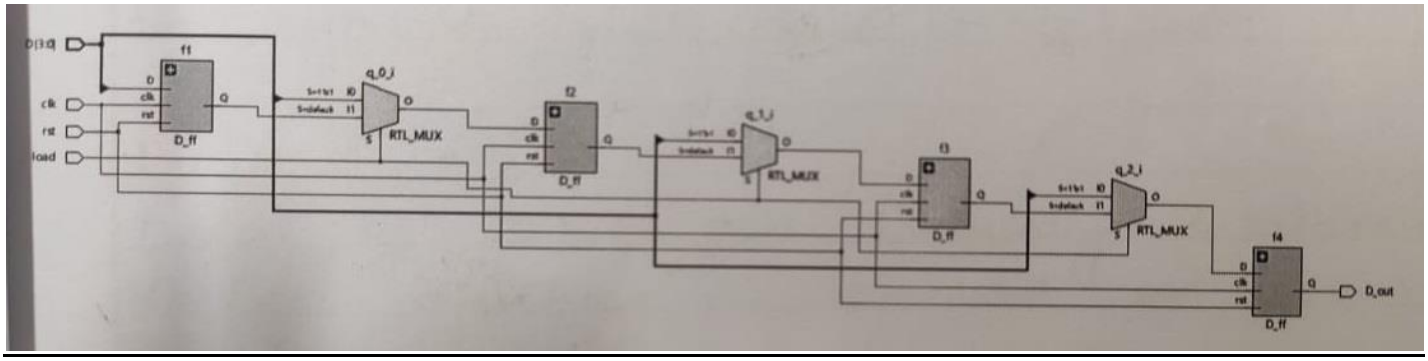
2. Iteration (a purely combinational approach) is expensive (in terms of # of transistors, chip area, power, etc).

3. A sequential approach allows the reuse of combinational functional units throughout the multi-cycle operation

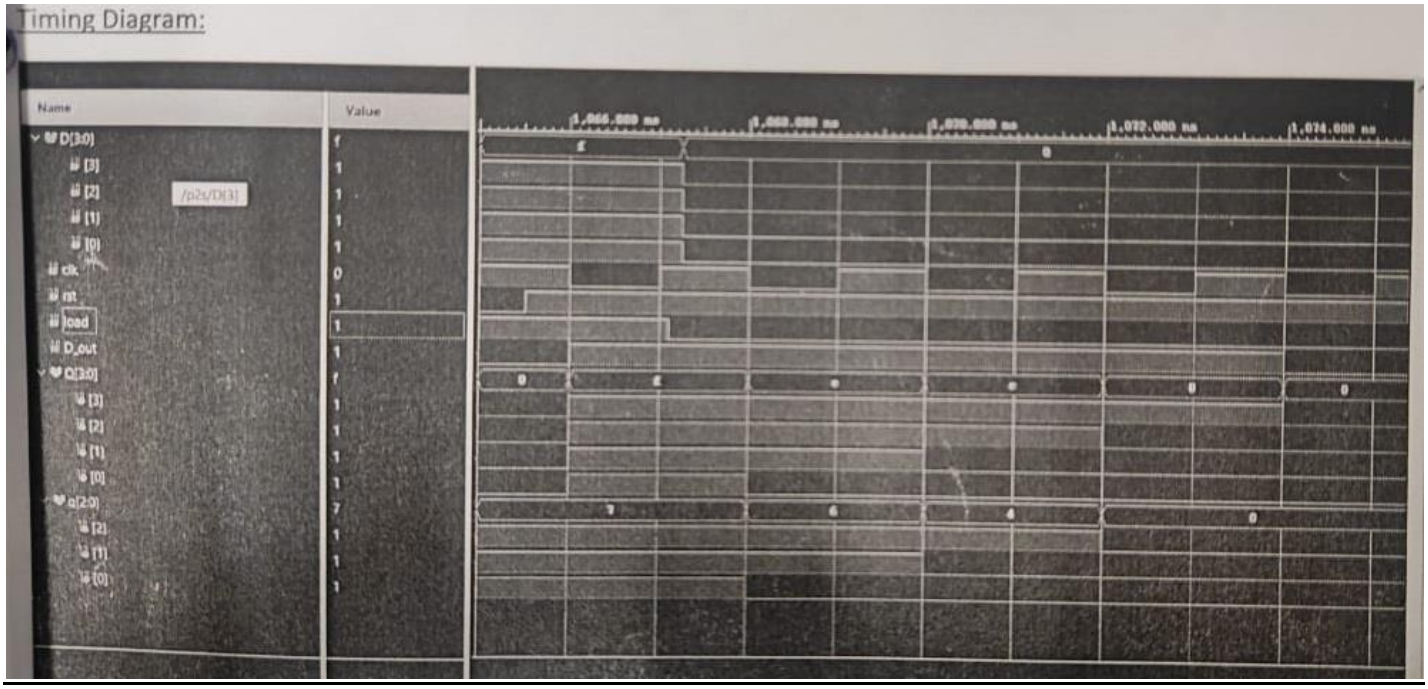
Verilog code:

```
module D_ff (input wire D, clk, rst, output reg Q);
always @(negedge clk or negedge rst)
begin
if (~rst)
Q<=1'b0;
else
Q<=D;
end
endmodule

module p2s (input wire [3:0]D, input wire clk, rst, load, output wire D_out

);

wire [3:0] Q;
wire [2:0] q;

//shift~load
assign q[0]= load ? D[1] : Q[0];
assign q[1]= load ? D[2] : Q[1];
assign q[2]= load ? D[3] : Q[2];

D_ff f1 (D[0],clk,rst,Q[0]);
D_ff f2 (q[0],clk,rst,Q[1]);
D_ff f3(q[1],clk,rst,Q[2]);
D_ff f4 (q[2],clk,rst,Q[3]);
assign D_out=Q[3];
endmodule
```

RTL schematic:



Output simulation:



Result: Verilog program to perform parallel to serial transfer of 4 bit binary number is simulated and verified.

EXPERIMENT–9

Objective: Write a program to design a 2 bit ALU containing 4 arithmetic & 4 logic operations

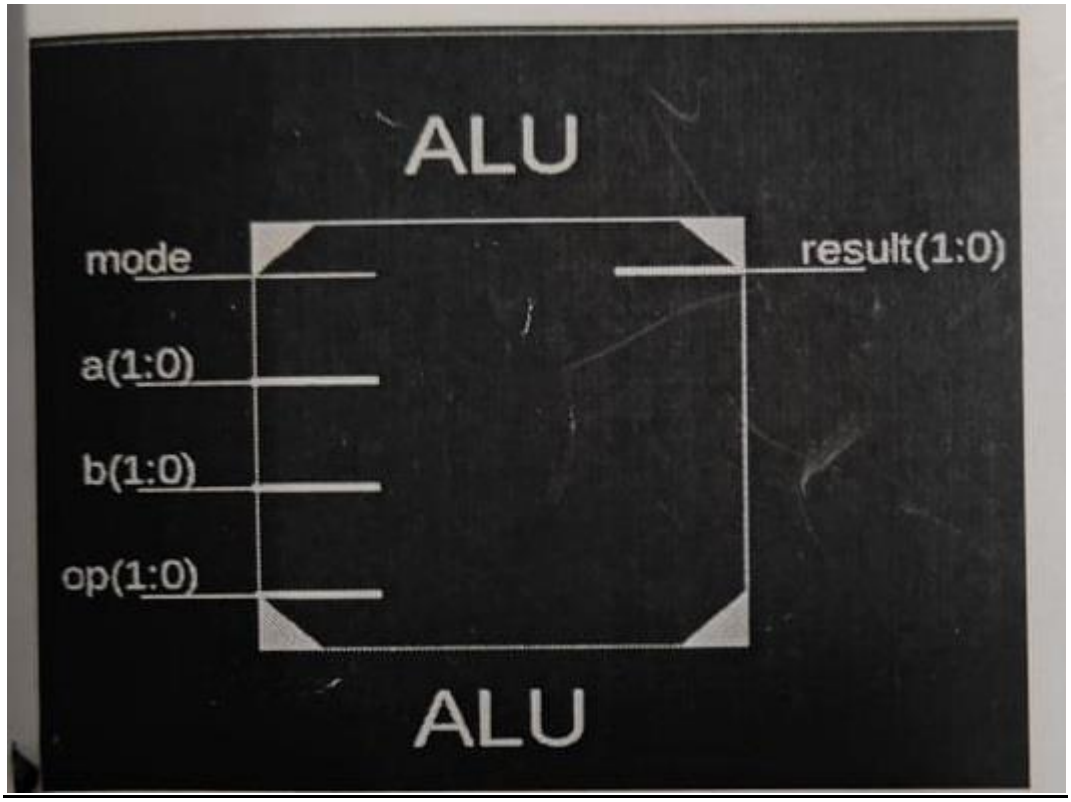
Resources Required: Xilinx-ISE 14.2

Theory: An ALU stands for arithmetic logic unit ALU is a digital circuit used to perform arithmetic and logic operations. It represents the fundamental building block of the central processing unit of CPU. Modern CPUs contains very powerful and complex ALUs. In some processors ALU is divided into two units an arithmetic unit and a logic unit.

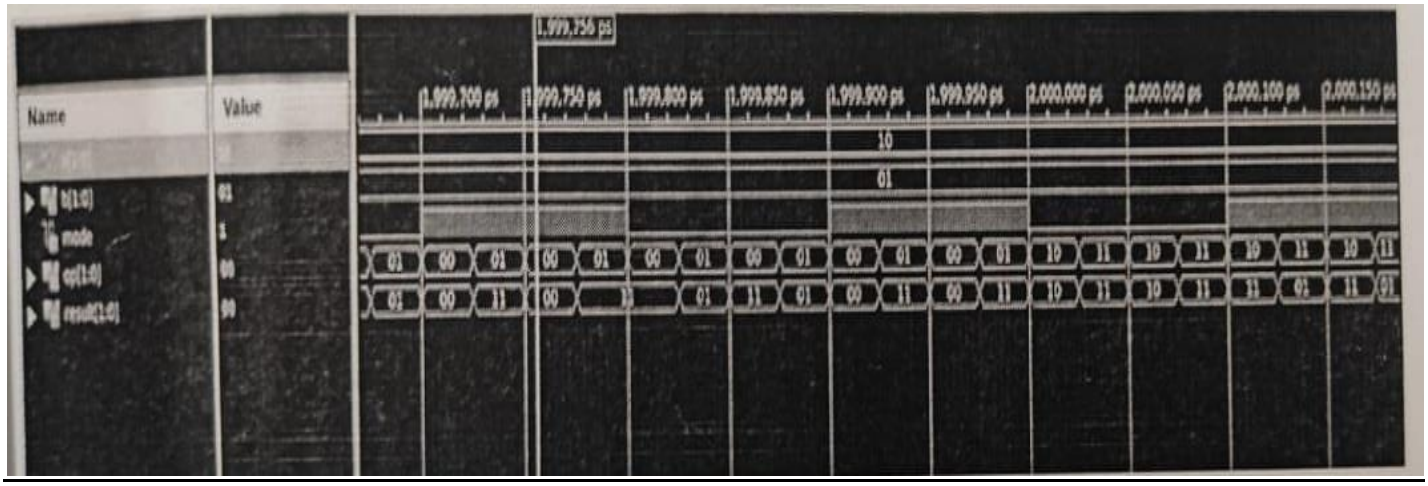
Verilog code:

```
module ALU (
   input [1:0]a,
   input [1:0]b,
   input mode,
   input [1:0]op,
   output reg [1:0] result
   );

   always@(*) begin
   if (mode==0) begin
   case (op)
   2'b00: result = a+b;
   2'b01: result = a-b;
   2'b10: result = a*b;
   2'b11: result = a+2'b01;
   endcase

end
else begin
case (op)
   2'b00: result = a&b;
   2'b01: result = a|b;
   2'b10: result = a^b;
   2'b11: result = ~a;
   endcase
end
end

endmodule
```

RTL schematic



Output simulation:



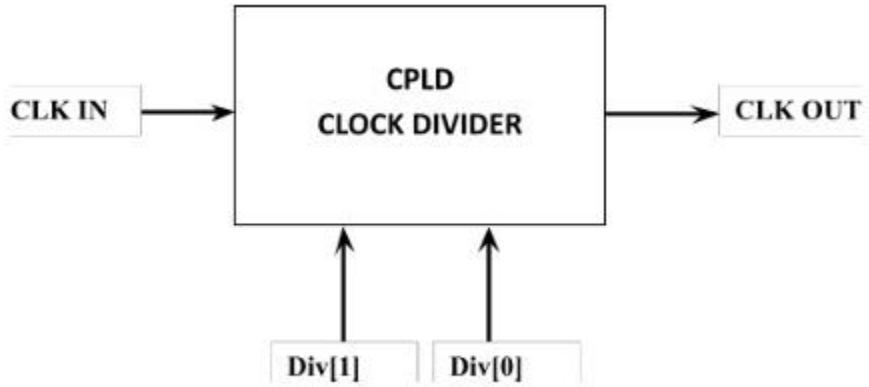
Result: Design of 2 bit ALU containing 4 arithmetic & 4 logic operations is simulated and verified.

EXPERIMENT–10

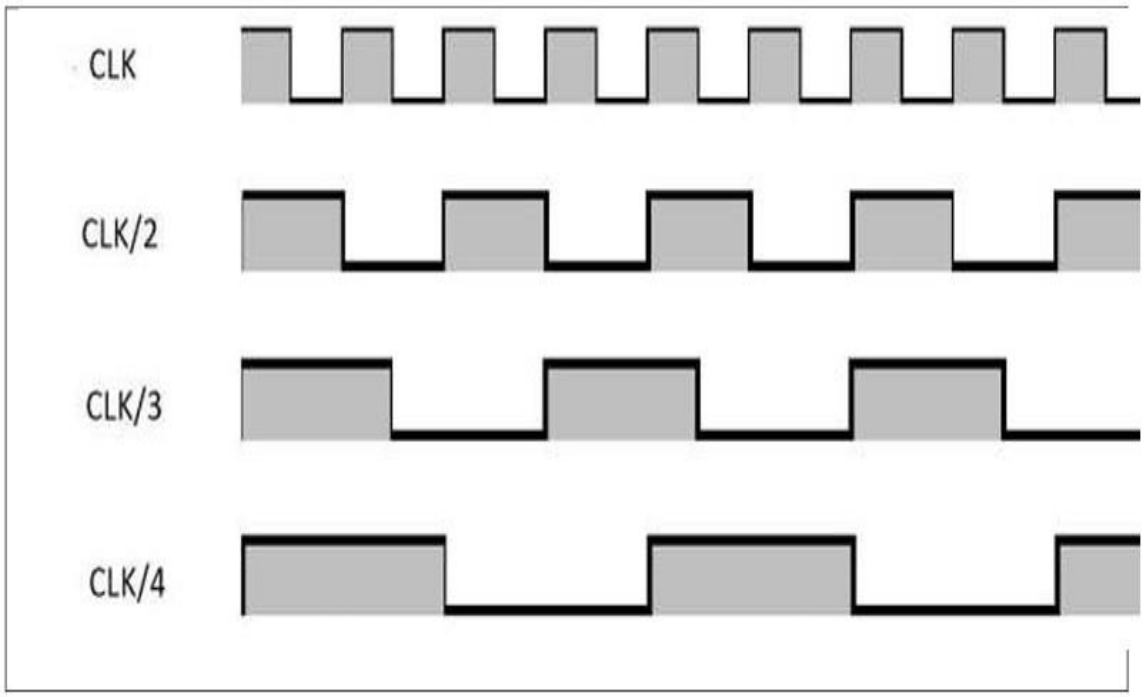
Objective: Write a Verilog Code to design a clock divider circuit that generates 1/2, 1/3rd and 1/4th clock from a given input clock. Port the design to FPGA and validate the functionality.

Resources Required: Xilinx-ISE 14.7i, FPGA board,DSO

Block Diagram:



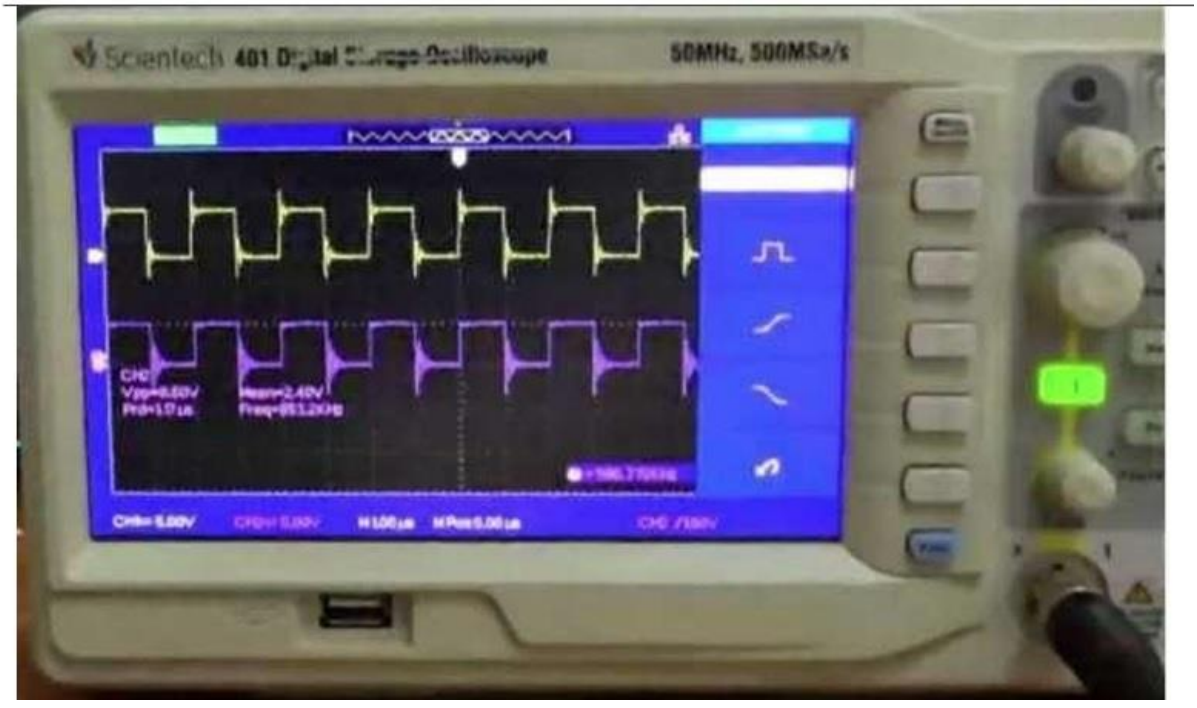
Waveform:



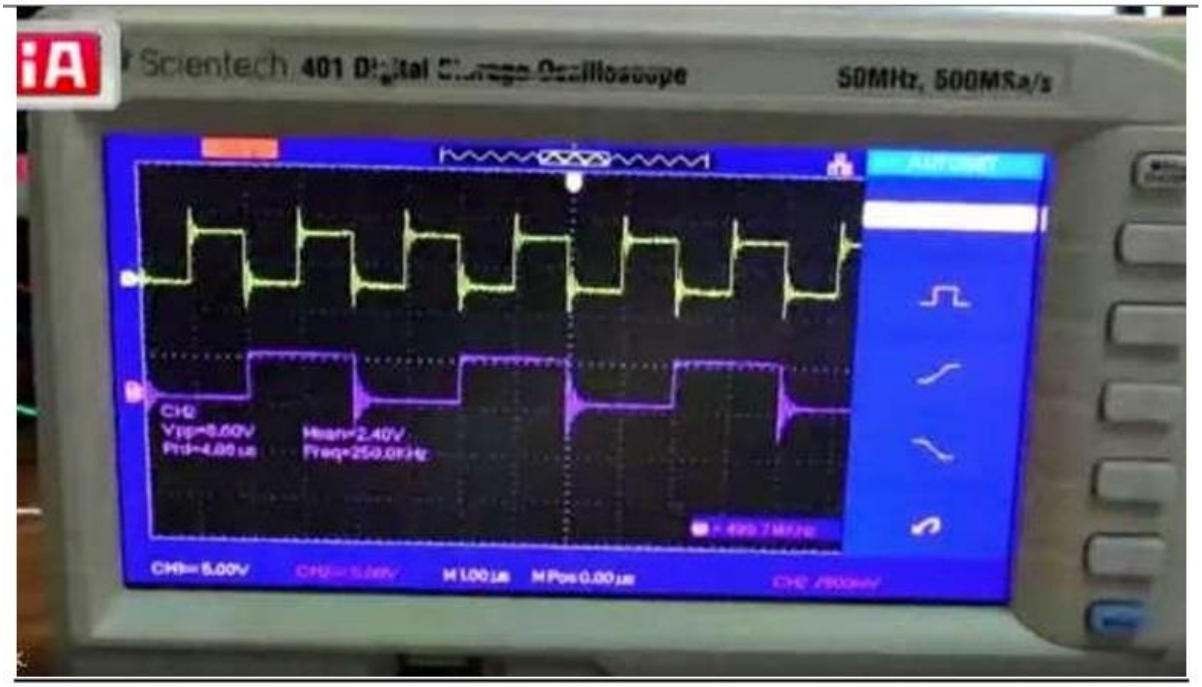
Verilog Program:

```
module CLK_DIVIDER(
 input CLK_IN,
 output reg CLK_OUT = 0,
 input [1:0] DIVISOR
 );
 reg [1:0] counter=0;

 always@(CLK_IN)
 begin
          if(counter != DIVISOR)
             counter = counter + 1;
          else
             begin
             CLK_OUT = ~CLK_OUT;
         counter = 0;
             end
          end
endmodule
```

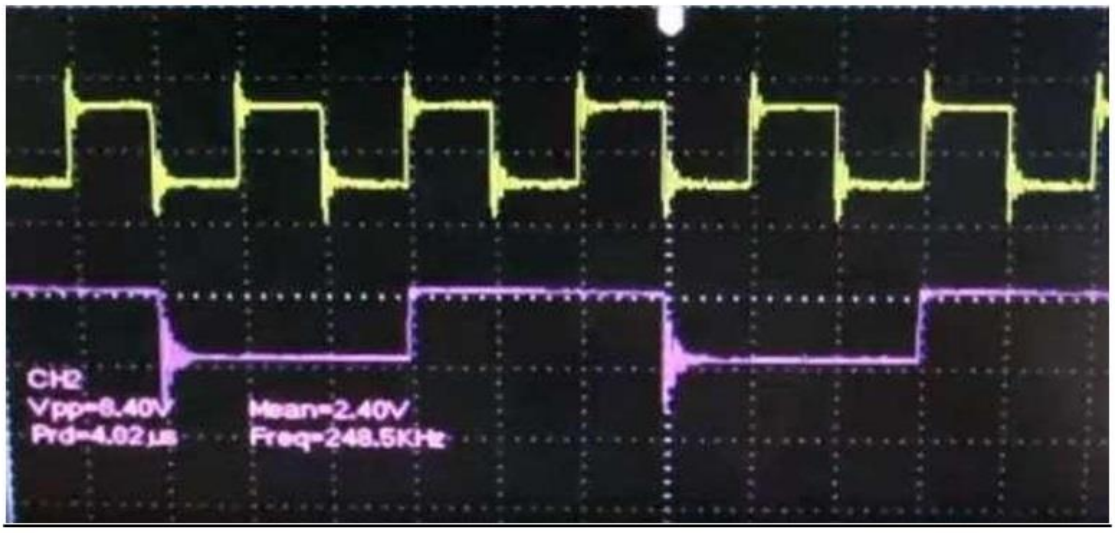
OUTPUT RESULTS:



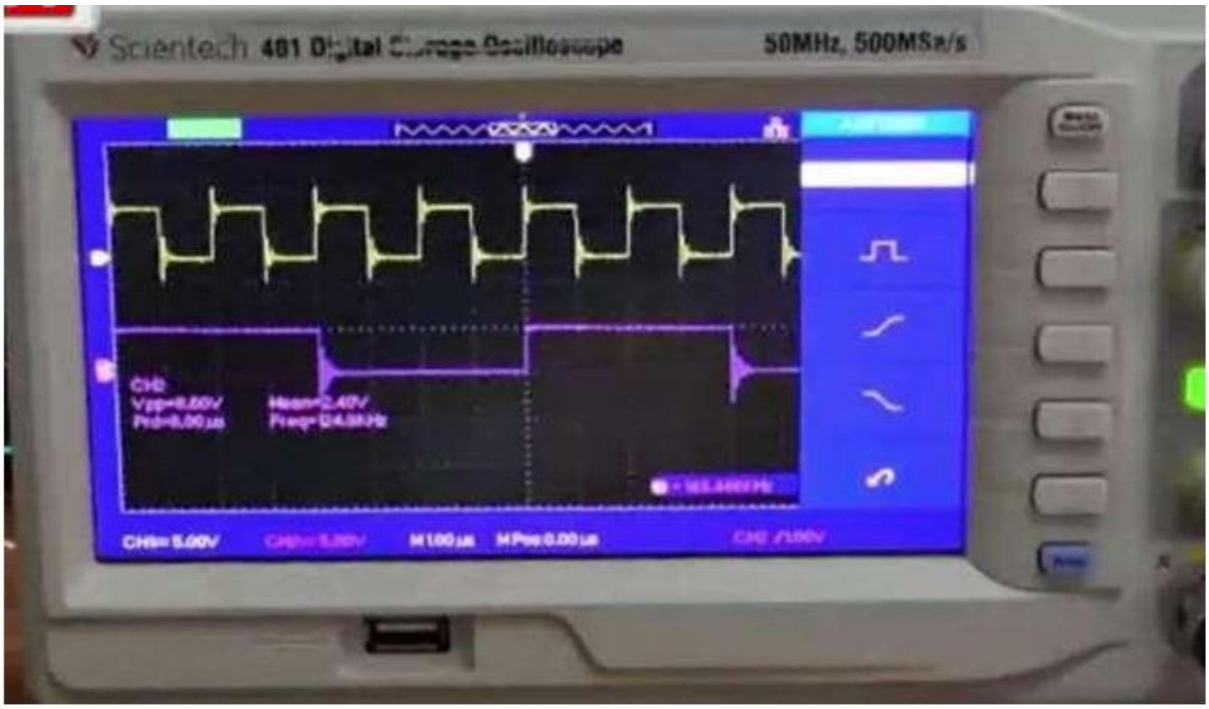
OUTPUT on oscilloscope for DIVISOR =00 (input clock divided by 1)



Output on Oscilloscope for DIVISOR=01, (input clock divide by 2)



Output on Oscilloscope for DIVISOR=10, (input clock divide by 3)



Output on Oscilloscope for DIVISOR=11, (input clock divide by 4)

Results: Verilog Code to design a clock divider circuit that generates 1/2, 1/3rd and 1/4th clock from a given input clock has been verified and Port the design to FPGA and validate the functionality successfully.