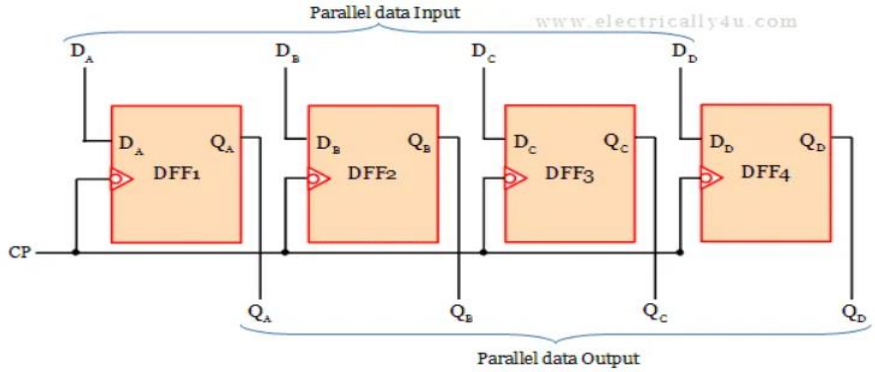
PROBLEM STATEMENT:

To design and construct a parallel in parallel out and a parallel in serial out shift register

THEORY:

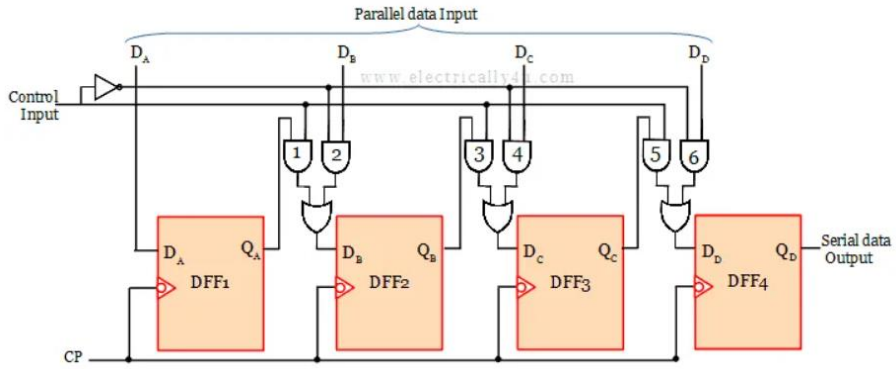
A shift register is a type of sequential logic circuit whose output is primarily determined by its input and previous output. This register contains a collection of Flip Flops that are linked in a cascade fashion, which means that one FF output is simply connected to the input of another FF. This register is used to store and shift a group of binary data. The number of FFs available within the shift register is primarily determined by the number of binary bits stored within the register. For example, if we want to store 4-bit binary data, we need four flip-flops.

Parallel PIPO shift registers are storage devices that perform both data loading and retrieval in parallel. D flip-flops are used to construct the Parallel-in Parallel-out shift register. The four-bit data is received by four flip-flops. When a clock pulse is applied, the loaded data is shifted to the flip-flop's output.



The data is inserted or loaded into each register in parallel in the parallel In Serial Out Shift register, and the inserted data is received serially at the output. Each flip-flop's input is to be loaded with data. At the same time as the clock pulse is applied, the data at each flip-flop's output is moved to the input of the next flip-flop. In order to avoid input conflicts between the loaded and shifted data, the control input is added to this shift register.

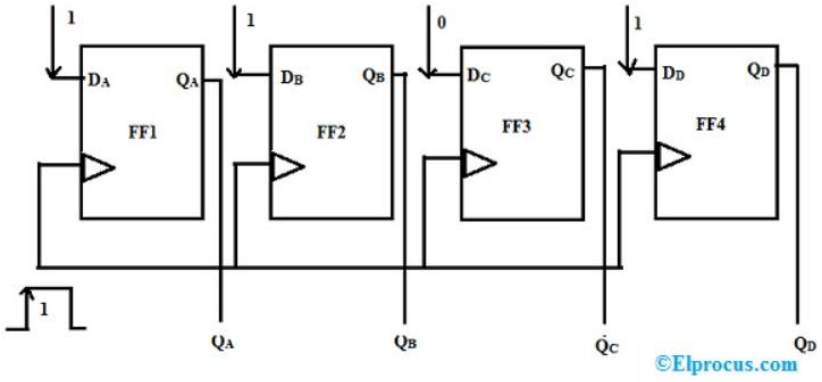
If the control input is 0 then AND gates 1, 3, and 5 are disabled and gates 2, 4, and 6 are enabled. It will cause the parallel data at the inputs to be stored in the respective flip-flops. A clock pulse is not needed to load the parallel input data. When the control input is 1, the AND gates 1, 3, and 5 are enabled, while the gates 2, 4, and 6 are disabled. When a clock pulse is applied during this time, the data bit at each flip-flop input is shifted to its output.



DESIGN:

4bit pipo:

Diagram:

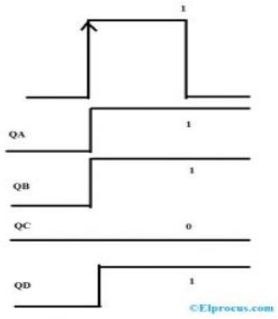


Truth table:

CLK Pulse	QA	QB	QC	QD
0	0	0	0	0
1	1	1	0	1

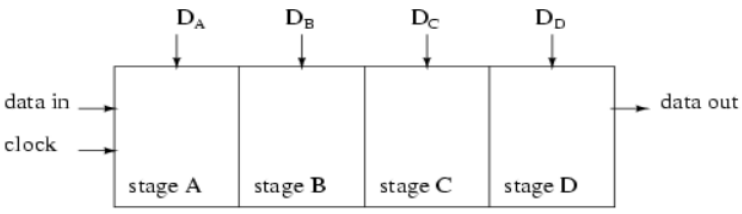
Timing diagram:

Data shits on the positive edge clk.

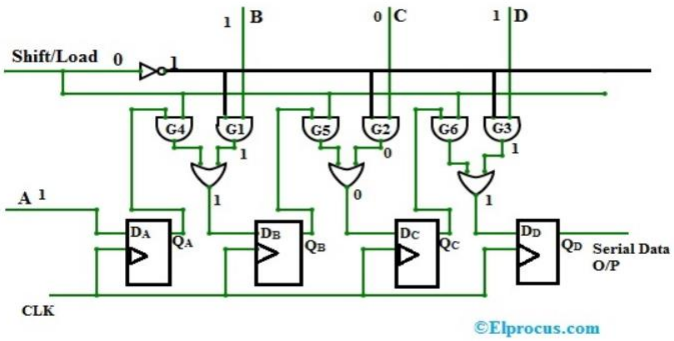


Timing Diagram

4-bit Piso:



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



Truth table:

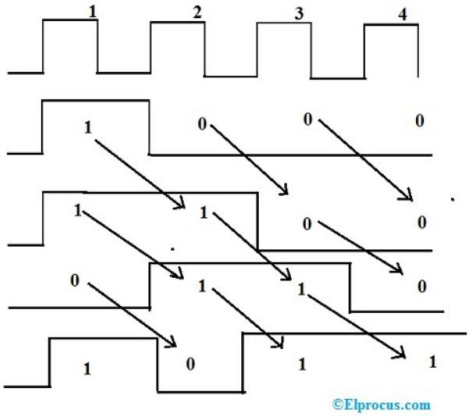
When the signal is load the data is taken as input

D_A	D_B	D_C	D_D
1	1	0	1

When the control signal is shift and clk=1 we get the output.

CLK Pulse	Q_A	Q_B	Q_c	Q_D (Data Output)
0	0	0	0	0
1	1	1	0	1
2	0	1	1	0
3	0	0	1	1
4	0	0	0	1

Timing diagram:



HDL CODE:

FOR PIPO:

```
module pipo(
   input logic [3:0] din,
   input logic clk,
   input logic ret,
   output logic [3:0] dout
   );
   always_ff @(posedge clk or negedge ret)
   if(!ret)   begin
   dout <= 4'b0;
    end
   else
   begin
    dout <= din;
    end
    endmodule
```

FOR PISO:

```
module piso(
   input logic clk,
   input logic clr,
   input logic sel,
   input logic [3:0]d,
   output logic sout );
```

```
    logic [3:0]q;
    always @(posedge clk,posedge clr)
    if (clr==1)    q<=4'b0000;
    else if(sel==0)    q<=d;
    else
    begin
    sout<=q[0];
    q<=q>>1;
    end
endmodule
```

COMBINIG BOTH:

```
module parall(
    input logic clk,
    input logic reset,
    input logic load,
    input logic [3:0] din,
    output logic [3:0] dout,
    output logic sout
    );
      /* logic clk_div=0;
       logic [27:0]count=0;
       always@ (posedge clk)
       begin
       count<=count+1;
       if(count[27]==1)
       begin
       clk_div<=~(clk_div);
       count<=0;
       end
       end*/
       logic [3:0]temp;
       always_ff @(posedge clk,posedge reset)
```

```
   if(reset) dout<=0;
    else if(load)
    begin
    dout<=din;
    temp<=din;
    end
   else
    begin
    sout<=temp[3];
   temp<={temp[2:0],1'b0};
    end
endmodule
```

TESTBENCH:

FOR PIPO:

For pipo:

```
module tb();
logic clk,ret;
logic [3:0]din,dout;
pipo dut(din,clk,ret,dout);
initial begin
ret=1'b0;
clk=1'b0;
end
initial begin
forever #40 clk=~clk;
end
initial begin
din=4'b0010;
#50 ret=1'b1;
#25 din=4'b0101;
#95 din=4'b110;
#5000 $finish;
end
```

```
endmodule
```

FOR PISO:

```
module tb( );
 logic clk,clr,sel,sout;
logic [3:0] d;
piso dut(clk,clr,sel,d,sout);
initial begin
clk=0;
forever #1 clk=~clk;
end
initial begin
clr=1;
#2 clr=0;
sel=0;
d=4'b0100;
#4 sel=1;
#10 sel=0;
d=4'b1110;
#2 sel=1;
#100 $finish;
end
  endmodule
```

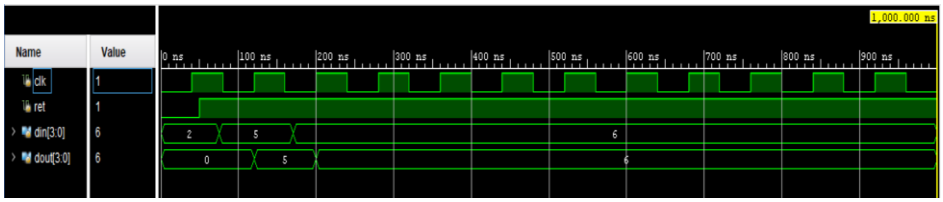
COMBINIG BOTH:

```
module tb();
    logic clk,reset, load,sout;
 logic [3:0] din,dout;
  parall dut( clk,reset, load,din,dout,sout);
  initial begin
  clk=1'b0;
  end
  initial begin
  forever #40 clk=~clk;
```

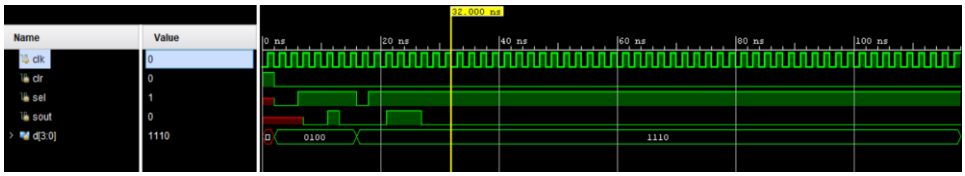
```
 end
 initial begin
 reset=1;load=1;din=4'b1000;
 #40 reset=0;
 #200 load=0;
 #200 load=1;
  end
endmodule
```

SIMULATION RESULT:

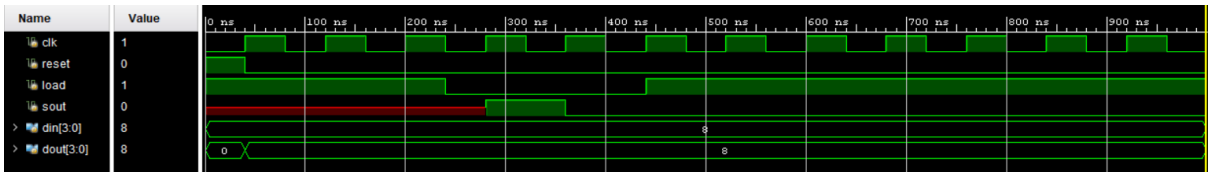
PIPO:



PISO:

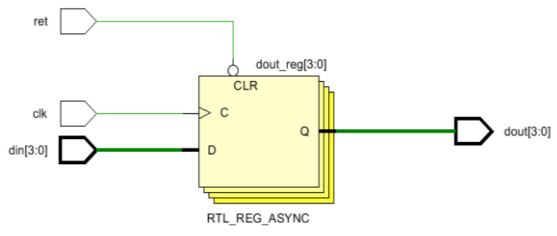


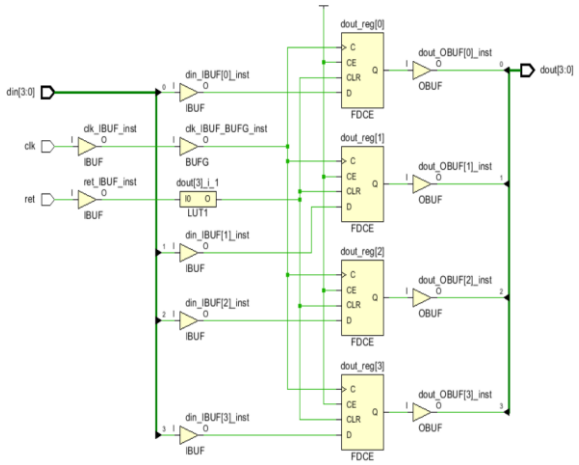
Combining both:



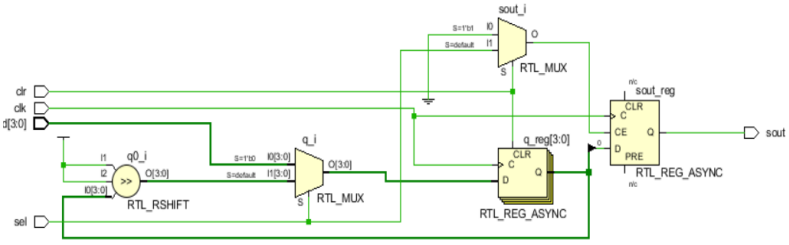
SYNTHESISED SCHEMATIC:

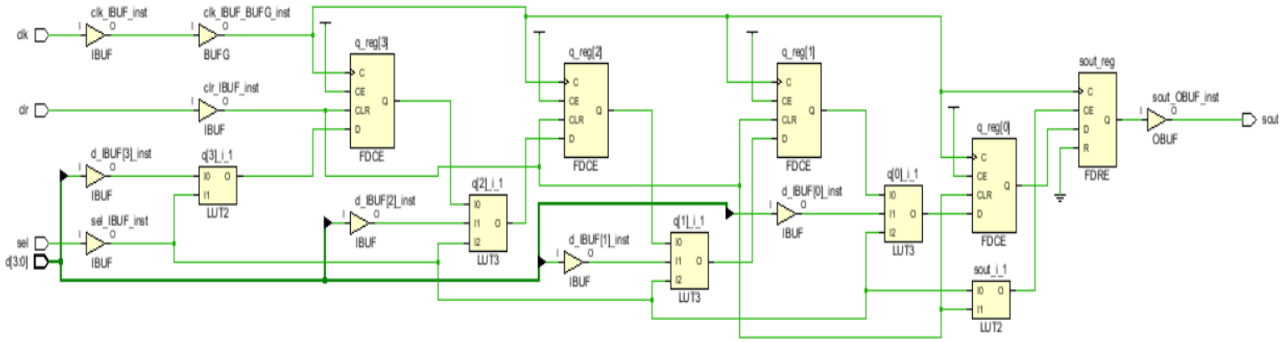
For PIPO:



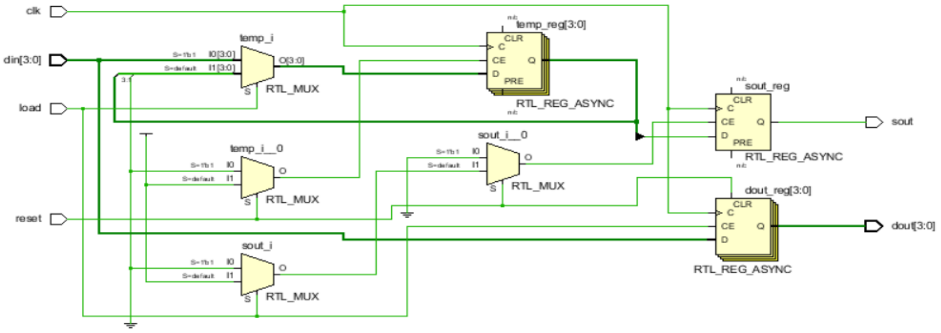


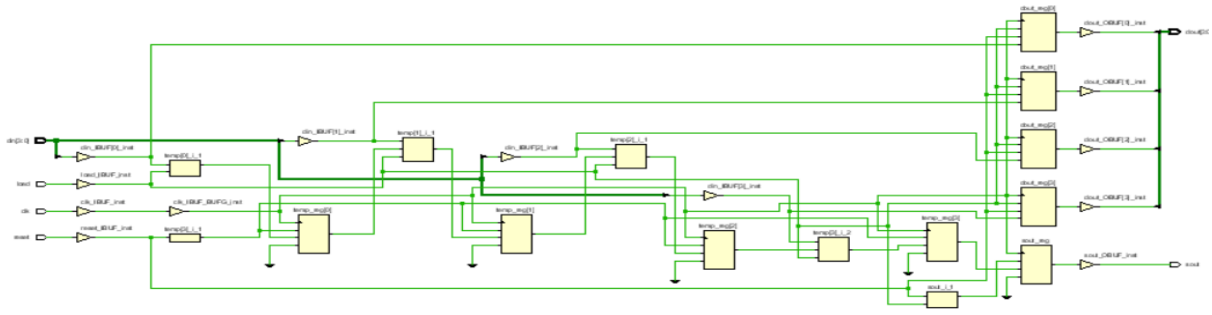
For PISO:





Combining the circuits:





HARDWARE IMPLEMENTATION:

The constrain file is as follows:

All ports (12)											
din (4)	IN		✓	34	LVCMOS33*	3.300				NONE	NONE
din[3]	IN	R2	✓	34	LVCMOS33*	3.300				NONE	NONE
din[2]	IN	T1	✓	34	LVCMOS33*	3.300				NONE	NONE
din[1]	IN	U1	✓	34	LVCMOS33*	3.300				NONE	NONE
din[0]	IN	W2	✓	34	LVCMOS33*	3.300				NONE	NONE
dout (4)	OUT		✓	35	LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50	
dout[3]	OUT	L1	✓	35	LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50	
dout[2]	OUT	P1	✓	35	LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50	
dout[1]	OUT	N3	✓	35	LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50	
dout[0]	OUT	P3	✓	35	LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50	
Scalar ports (4)											
clk	IN	W5	✓	34	LVCMOS33*	3.300				NONE	NONE
load	IN	T2	✓	34	LVCMOS33*	3.300				NONE	NONE
reset	IN	T3	✓	34	LVCMOS33*	3.300				NONE	NONE
sout	OUT	U3	✓	34	LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50	

set_property IOSTANDARD LVCMOS33 [get_ports {din[0]}]

set_property IOSTANDARD LVCMOS33 [get_ports {din[1]}]

set_property IOSTANDARD LVCMOS33 [get_ports {din[2]}]

set_property IOSTANDARD LVCMOS33 [get_ports {din[3]}]

set_property IOSTANDARD LVCMOS33 [get_ports {dout[0]}]

set_property IOSTANDARD LVCMOS33 [get_ports {dout[1]}]

set_property IOSTANDARD LVCMOS33 [get_ports {dout[2]}]

set_property IOSTANDARD LVCMOS33 [get_ports {dout[3]}]

set_property PACKAGE_PIN R2 [get_ports {din[3]}]

set_property PACKAGE_PIN T1 [get_ports {din[2]}]

set_property PACKAGE_PIN U1 [get_ports {din[1]}]

set_property PACKAGE_PIN W2 [get_ports {din[0]}]

set_property PACKAGE_PIN L1 [get_ports {dout[3]}]

set_property PACKAGE_PIN P1 [get_ports {dout[2]}]

set_property PACKAGE_PIN N3 [get_ports {dout[1]}]

set_property PACKAGE_PIN P3 [get_ports {dout[0]}]

set_property IOSTANDARD LVCMOS33 [get_ports clk]

set_property PACKAGE_PIN W5 [get_ports clk]

set_property IOSTANDARD LVCMOS33 [get_ports load]

set_property PACKAGE_PIN T2 [get_ports load]

set_property IOSTANDARD LVCMOS33 [get_ports reset]

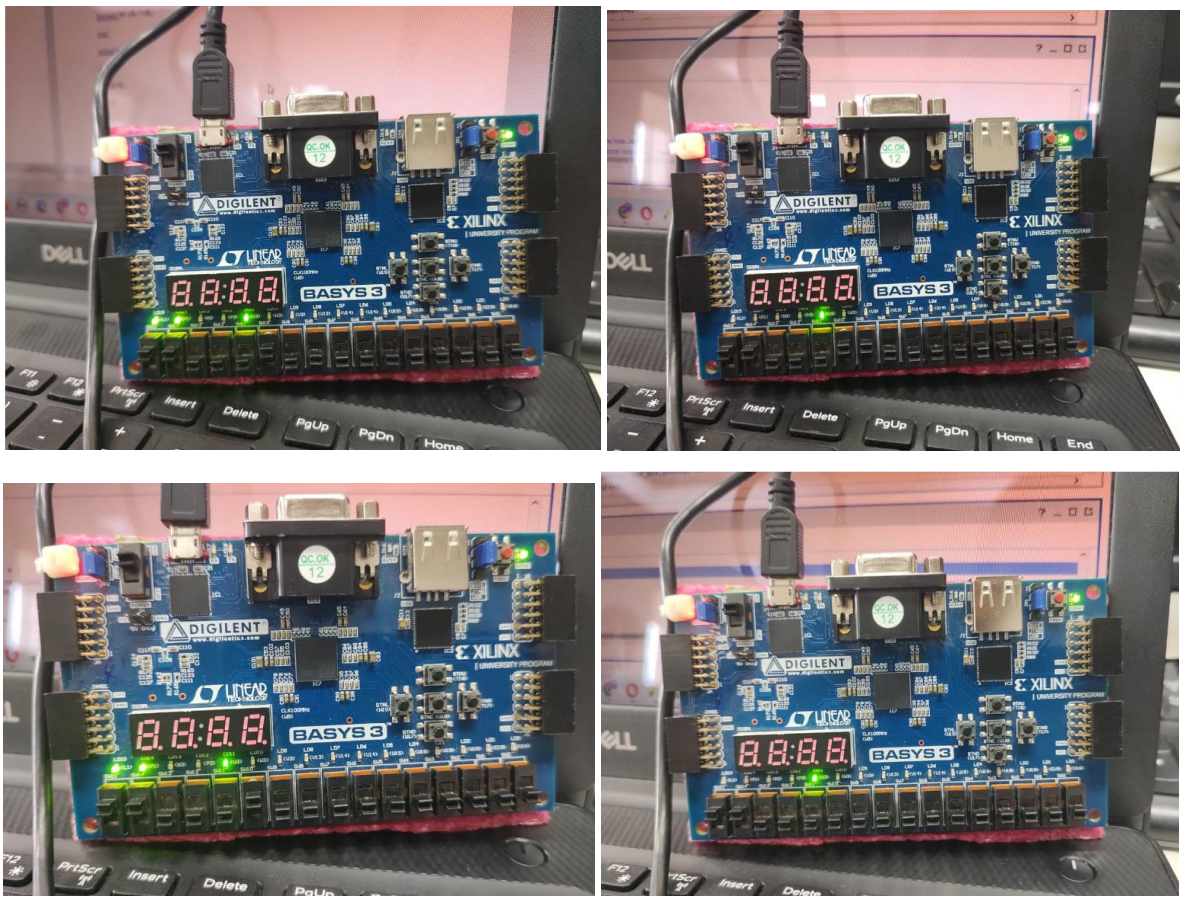
set_property PACKAGE_PIN T3 [get_ports reset]

set_property PACKAGE_PIN U3 [get_ports sout]

set_property IOSTANDARD LVCMOS33 [get_ports sout]

HARDWARE OUTPUT:

Output obtained for different conditions:



CONCLUSION:

PIPO and PISO circuits find their applications in many digital circuits (for example data storage, manipulation, and transmission). The purpose of the project was to design and build PIPO and PISO, which were served. There is still room for improvement which would take a bit more research.