Name- Moitrish Maity Roll-48 Dept-CSE 4th Sem

EXPERIMENT-11

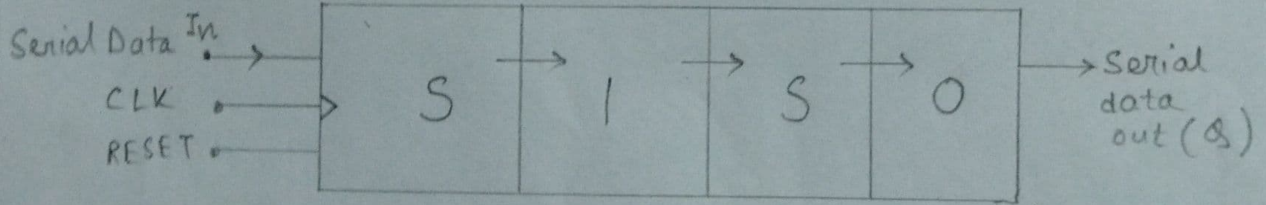
- Title: Shift registers
- Objective: Implementation of shift registers and GPR.
- Course Outcome: CO3
- Bloom's Level: Analysis
- Problem Statement: Write a verilog program to design, test, and simulate these following registers:

A) SISO B) SIPO C) PISO D) PIPO

(A) SISO

- Theory: The shift register, which allows serial input and produces a serial output is SISO shift register. Since there is only one output, the data left at the register at a time in serial pattern. The circuit consists of four D ffs which are connected in a serial manner. All these ffs are synchronized with each other since same clock signal is applied to each flipflop.

- Logic Diagram:



▫ Function Table:

No. of clock Pulse	Serial Input	Serial Output
T	D	Q_0
0	H	L
1	L	L
2	H	L
3	L	L
4	-	H
5	-	L
6	-	H
7	-	L

(Initially, all ffs are in 'Reset' mode)

↓↓

Q_3 Q_2 Q_1 Q_0

↓↓

L L L L

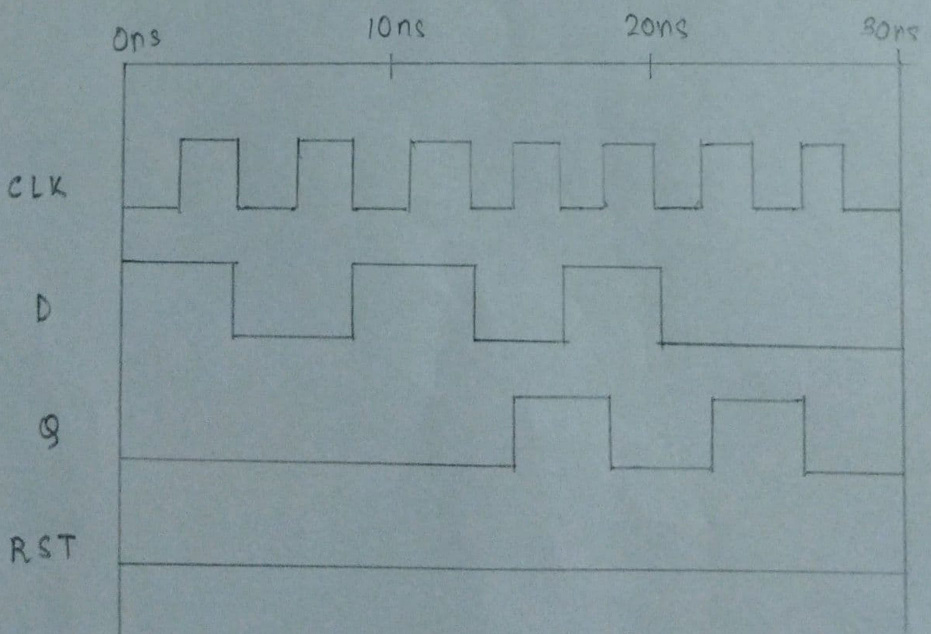
▫ Design Code :

```
module d-ff (input d, input clk, input rst, output q, output qbar);
    reg q=0;
    assign qbar = ~q;
    always @ (posedge clk)
      begin
          if (d==1 || d==0) begin
              q<=d; end
          end
    always @ (posedge rst)
      begin
          q<= 0;
      end
endmodule
module siso( input d, input clk, input rst, output [3:0]s, output q,
                                                output [3:0] qbar);
    d-ff f4 (d, clk, rst, s[3], qbar[3]);
    d-ff f3 (s[3], clk, rst, s[2], qbar[2]);
    d-ff f2 (s[2], clk, rst, s[1], qbar[1]);
    d-ff f1 (s[1], clk, rst, s[0], qbar[0]);
    assign q = s[0];
endmodule
```

▫ Testbench Code:

```
module siso_tb();
  reg D;
  reg CLK, RST;
  wire Q;
  SISO uut (.d(D), .clk(CLK), .rst(RST), .q(Q));
  initial begin
      $dumpfile("dump.vcd"); $dumpvars(1);
      CLK=0;
      RST=0;
      D=1; #4;
      D=0; #4;
      D=1; #4;
      D=0; #4;
      D=1; #4;
      D=0; #12;
      $finish;
  end
  always #2 CLK=~CLK;
endmodule
```

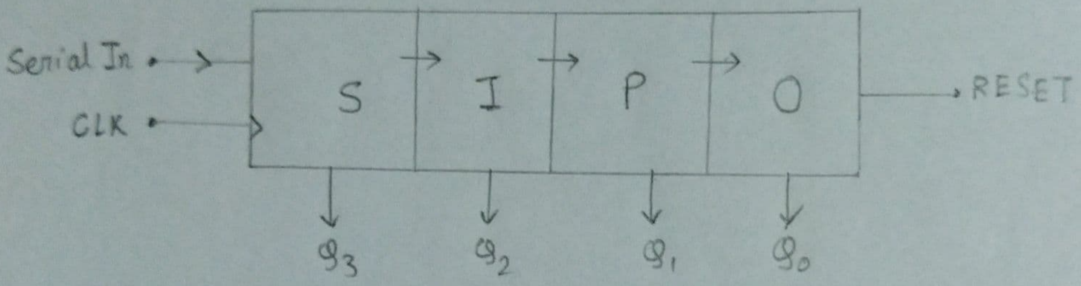
▫ Timing Diagram:



Ⓑ SIPO

▫ **Theory:** The shift register, which allows serial inputand produces a parallel output is known as SIPO register. Circuit consists of 4 D ffs which are connected through a serial manner. All ffs are synchronous with each other since same clock signal is applied to each ffs. From each ffs we get four separate output lines as register output.

▫ **Logic Diagram:**



▫ **Function Table:**

NO. OF CLK	SERIAL INPUT	PARALLEL OUTPUT			
T	D	Q_3	Q_2	Q_1	Q_0
0	–	L	L	L	L
1	H	H	L	L	L
2	L	L	H	L	L
3	H	H	L	H	L
4	L	L	H	L	H

(Initially all ffs are in 'Reset' mode)

▫ **Design Code:**

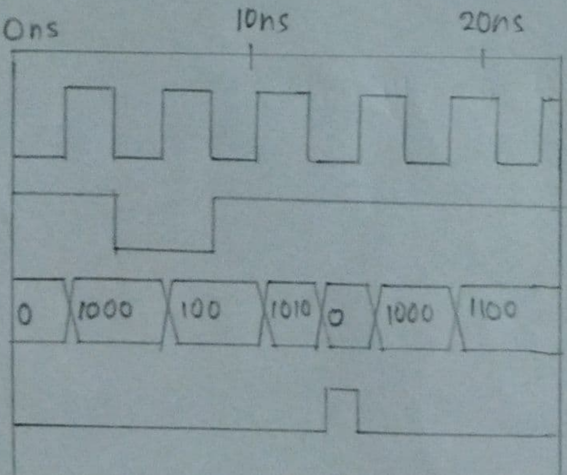
```
module SIPO( input d , input clk , input rst, output [3:0] q,
                                          output [3:0] qbar);
   d-ff f4 (d, clk , rst, q[3], qbar[3]);
   d-ff f3 ( q[3], clk , rst, q[2], qbar[2]);
```

```
    d-ff f2(q[2], clk, rst, q[1], qbar[1]);
    d-ff f1 (q[1], clk, rst, q[0], qbar[0]);
endmodule
```

▫ Testbench Code:

```
module SIPO_tb();
    reg D;
    reg CLK, RST;
    wire [3:0]Q;
    SIPO uut (.d(D), .clk(CLK), .rst(RST), .q(Q));
    initial begin
        $dumpfile("dump.vcd"); $dumpvars(1);
        CLK=0;  RST=0;
        D=1; #4;
        D=0; #4;
        D=1; #4;
        RST = 1; #1;
        RST=0;
        D=1; #4;
        D=1; #4;
        $finish;
    end
    always #2 CLK=~CLK;
endmodule
```

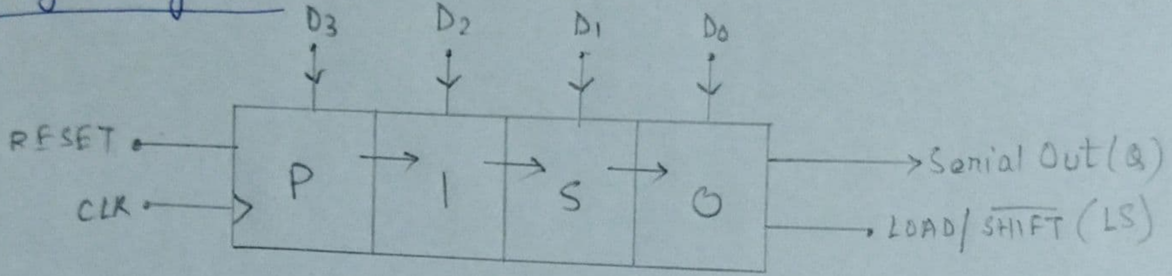
▫ Timing Diagram:



© PISO

□ Theory: In PISO register, parallel loading is done by applying four parallel input bits. Then for each positive clock edge, data shifts one stage to the right. So, we can get serial output from rightmost ff. A load/ $\overline{\text{shift}}$ control signal is used to perform load and shift operation. (for loading → 'L' and for shift → 'H')

□ Logic Diagram:



□ Function Table:

NO. OF CLK	~~SERIAL INPUT~~ LOAD/ SHIFT	~~PAR~~ PARALLEL INPUTS				SERIAL OUTPUT
(T)	(LS)	D3	D2	D1	D0	(Q)
0	L	H	L	H	H	L
1	L	H	L	H	H	H
2	H	L	H	L	H	H
3	H	L	L	H	L	L
4	H	L	L	L	H	H
5	H	L	L	L	L	L

(Initially all ffs are in 'Reset' mode)

□ Design Code:

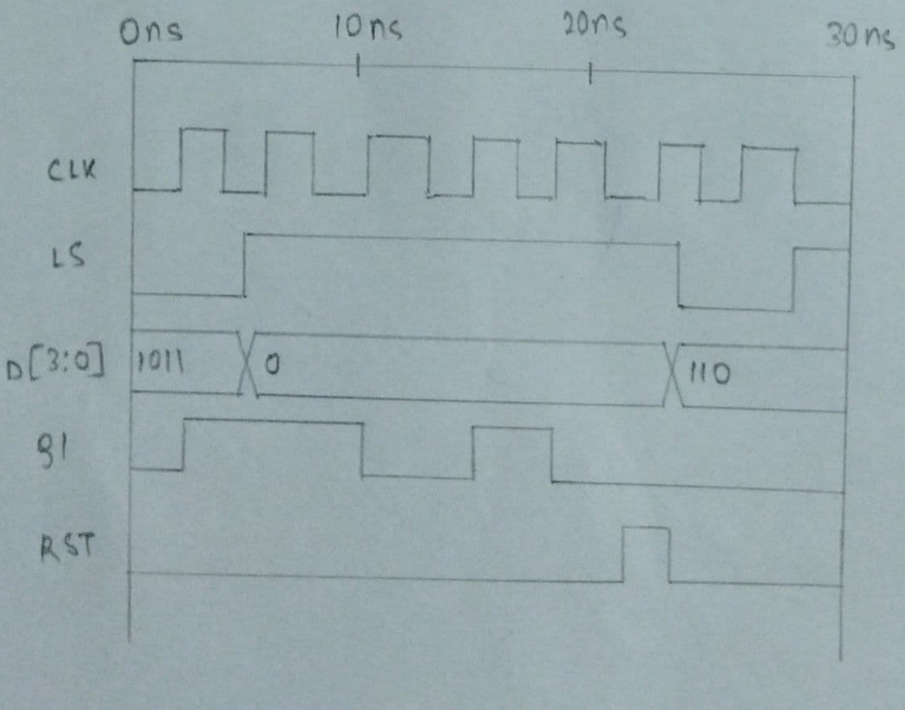
```
module PISO (input [3:0]d, input clk, input rst, input ls, output
                                  [3:0]s, output [3:0]q, output [3:0]qbar);

    d-ff f4 (d[3], clk, rst, s[3], qbar[3]);
    d-ff f3 (((ls & s[3])|(~ls & d[2])), clk, rst, s[2], qbar[2]);
    d-ff f2 (((ls & s[2])|(~ls & d[1])), clk, rst, s[1], qbar[1]);
    d-ff f1 (((ls & s[1])|(~ls & d[0])), clk, rst, s[0], qbar[0]);
    assign q = s[0];
endmodule
```

▫ Testbench Code:

```
module PISO_tb();
   reg [3:0] D;
   reg CLK, RST, LS;
   wire q1;
   PISO uut (.d(D), .clk(CLK), .rst(RST), .ls(LS), .q(q1));
   initial begin
         $dumpfile("dump.vcd"); $dumpvars(1);
         CLK=0; RST=0; LS=0;
         D=4'b1011; #5;
         D=4'b0000;
         LS=1 ; #16;
         RST=1;  #2 ; RST=0;
         LS=0 ;
         D=4'b0110; #5 ; LS=1; #20;
         $finish;
      end
   always #2 CLK=~CLK;
endmodule
```

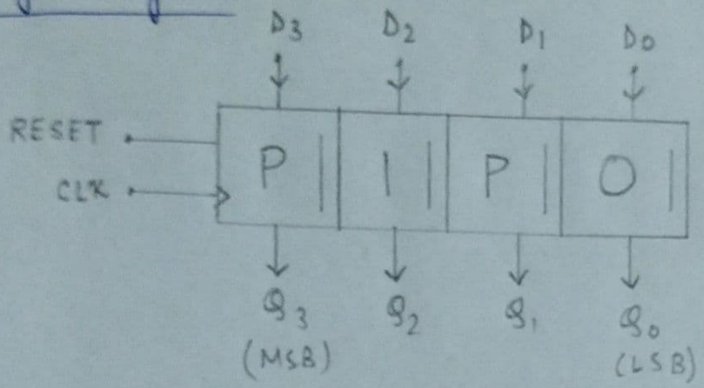
▫ Timing Diagram:



Ⓓ PIPO

▫ **Theory:** In PIPO register, parallel loading is done by applying four parallel input bits at a time. From these four flipflops four parallel output bits are obtained by applying a single clock pulse to all flipflops. All ff's are synchronized with a same clock signal.

▫ **Logic Diagram:**



▫ **Function Table:**

NO. OF CLK PULSE	PARALLEL INPUT				PARALLEL OUTPUT			
	D3	D2	D1	D0	Q3	Q2	Q1	Q0
↑	L	L	H	H	L	L	H	H
↑	L	H	H	L	L	H	H	L
↑	L	L	H	L	L	L	H	L
↑	H	L	H	L	H	L	H	L
↑	H	H	L	L	H	H	L	L
↑	H	L	L	H	H	L	L	H

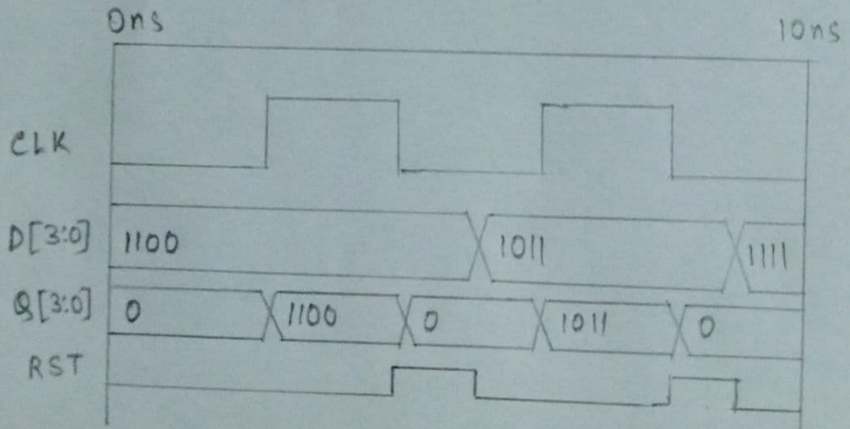
▫ **Design Code:**

```
module PIPO(input [3:0]d, input clk, input rst, output [3:0]q, output
                                                          [3:0]qbar);
    d-ff f1(d[0], clk, rst, q[0], qbar[0]);
    d-ff f2(d[1], clk, rst, q[1], qbar[1]);
    d-ff f3(d[2], clk, rst, q[2], qbar[2]);
    d-ff f4(d[3], clk, rst, q[3], qbar[3]);
endmodule
```

□ Testbench Code:

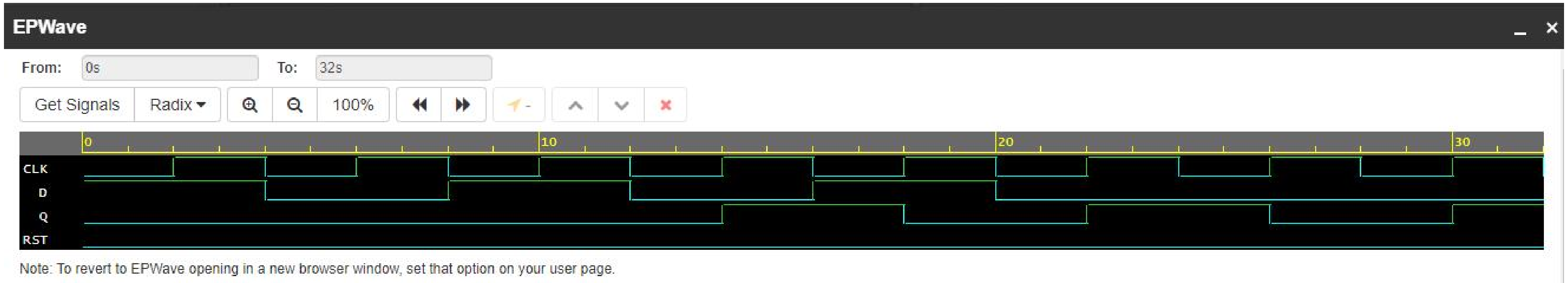
```
module PIPO_tb();
   reg [3:0]D;
   reg CLK, RST;
   wire [3:0]Q;
   PIPO uut (.d(D),.clk(CLK),.rst(RST),.q(Q));
   initial begin
      $dumpfile("dump.vcd"); $dumpvars(1);
      CLK=0; RST=0;
      D=4'b1100; #4;
      RST=1; #1; RST=0;
      D=4'b1011; #3;
      RST=1; #1; RST=0;
      D=4'b1111; #4;
      $finish;
    end
  always #2 CLK=~CLK;
endmodule
```

□ Timing Diagram:



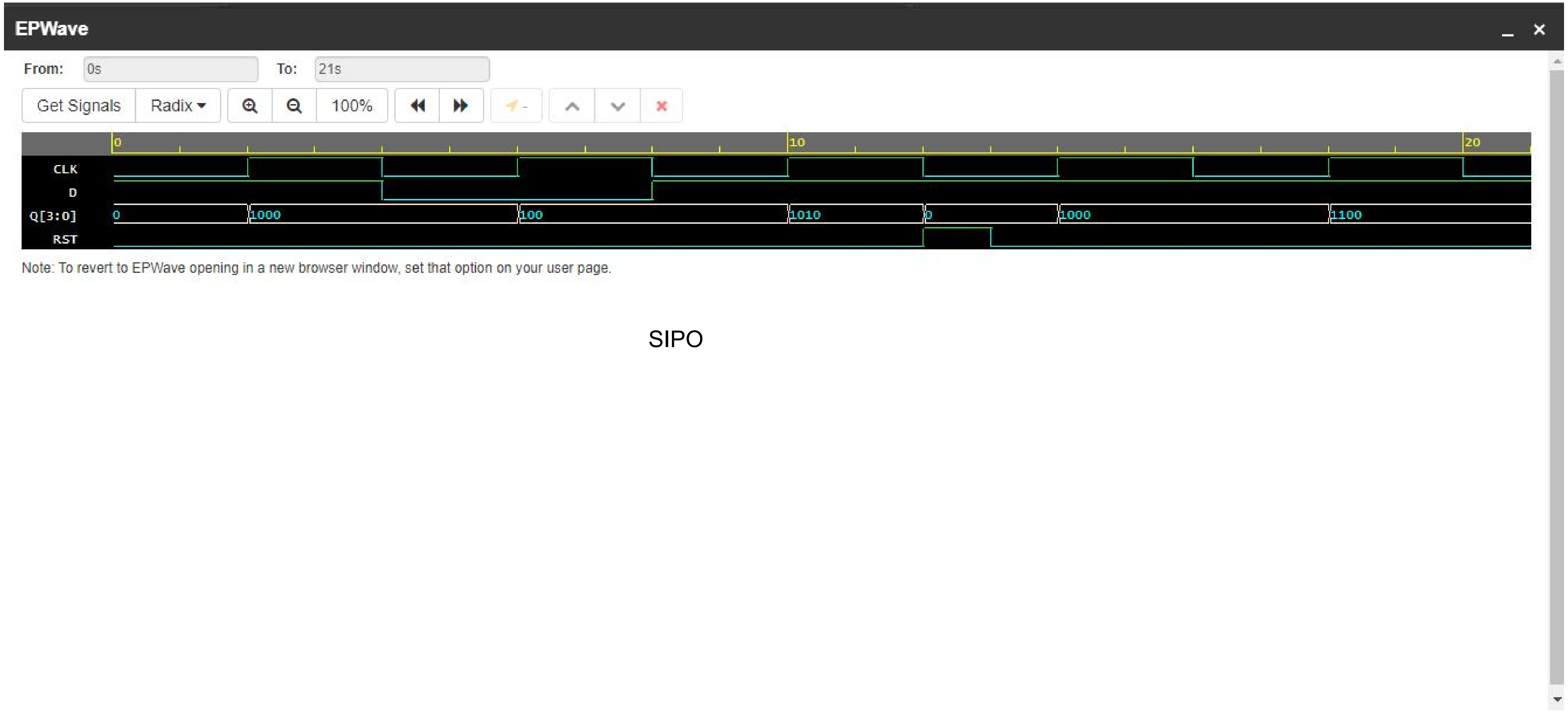
□ Discussion: In this experiment, we have implemented various shift registers like SISO, SIPO, PISO & PIPO using flipflop modules. We have learned various HDL terms also.

□ Justification of CO: In this experiment, we have designed various sequential circuits and used some sequential modules in those circuits, which fulfils the conditions of CO3. Hence, CO3 is justified.



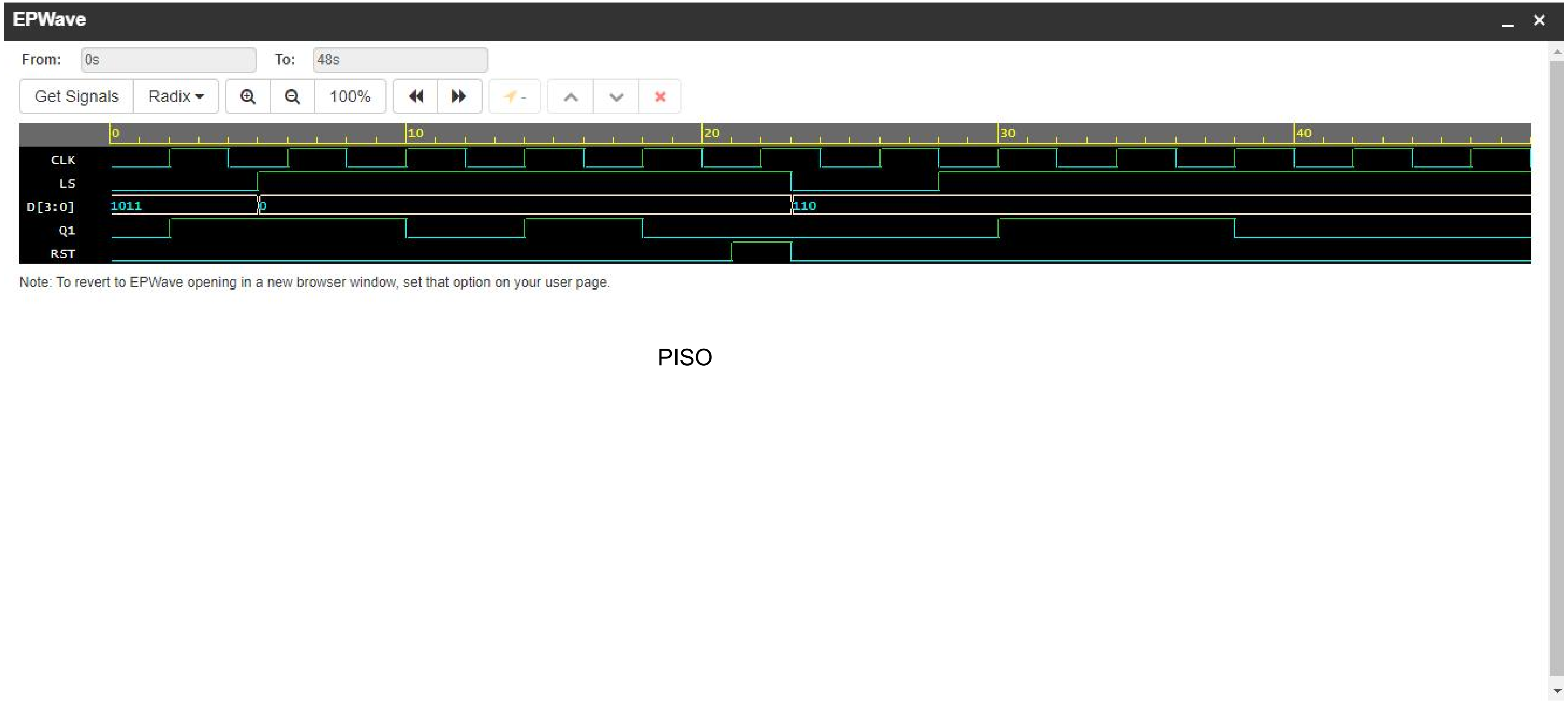
Note: To revert to EPWave opening in a new browser window, set that option on your user page.

SISO



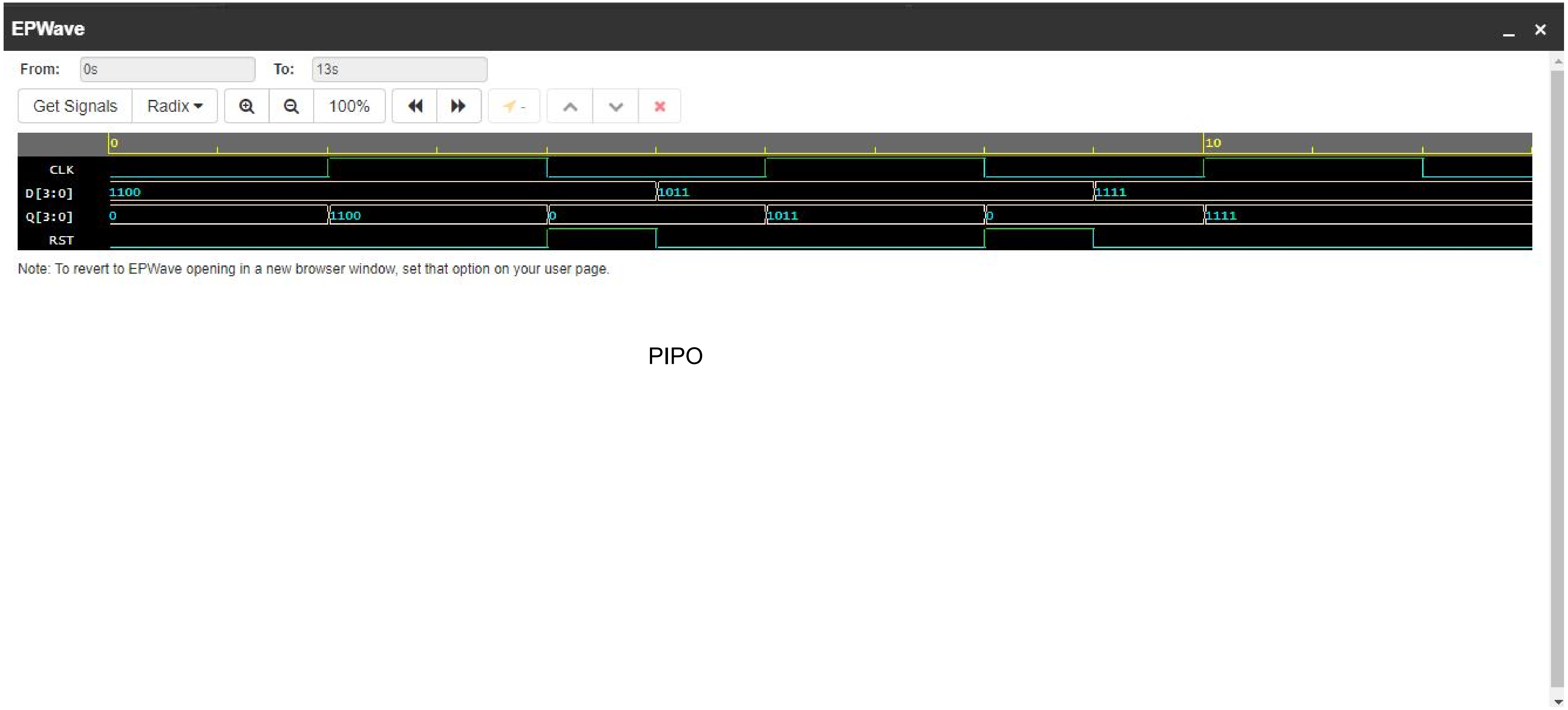
Note: To revert to EPWave opening in a new browser window, set that option on your user page.

SIPO



Note: To revert to EPWave opening in a new browser window, set that option on your user page.

PISO



Note: To revert to EPWave opening in a new browser window, set that option on your user page.

PIPO