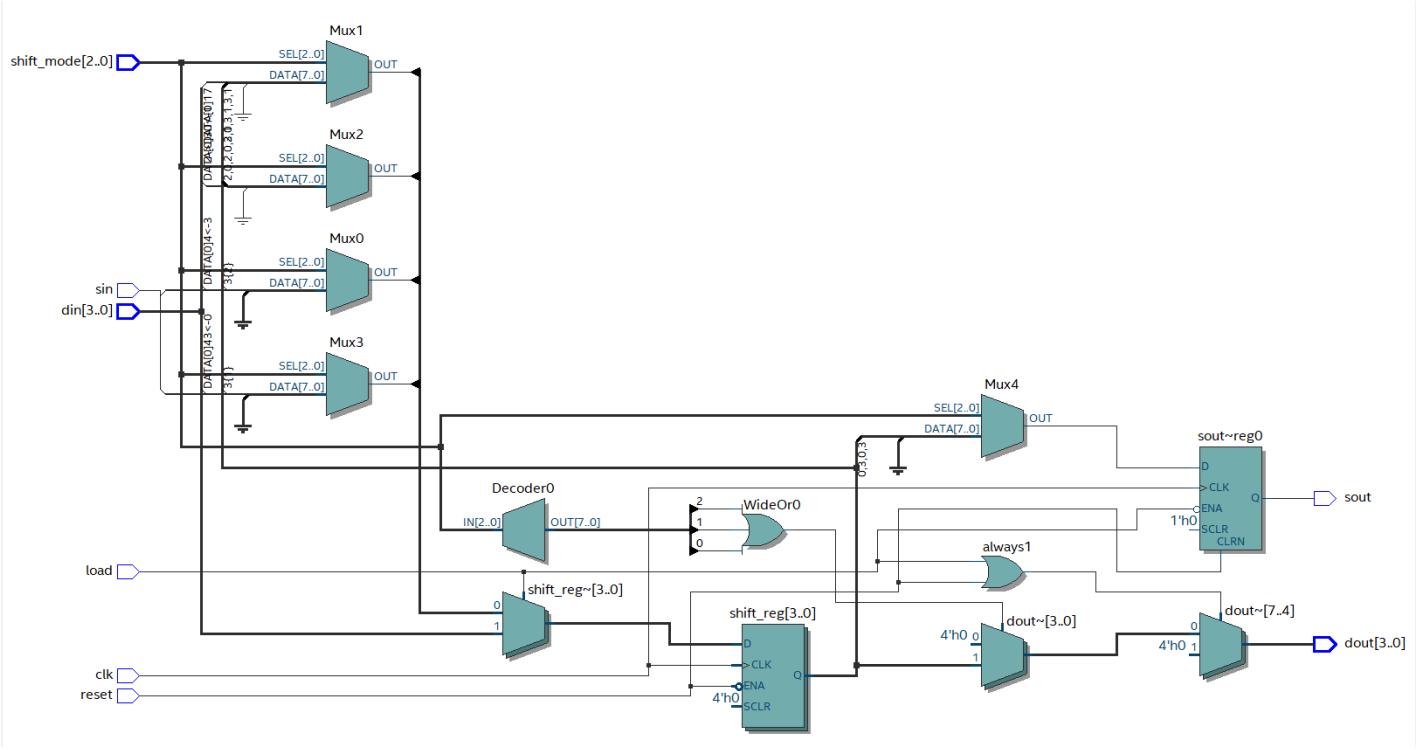
ECE 111 Winter 2022
HW3
Hao Le A15547504

Johnson Counter

Code

```
1   // Johnson Counter RTL Model
2   module johnson_counter (
3     input logic clk, clear, preset,
4     input logic[3:0] load_cnt,
5     output logic[3:0] count
6   );
7    always@(posedge clk or negedge clear) begin
8
9      if(!clear)
10        count <= 4'b0000;
11
12     else if(!preset)
13        count <= load_cnt;
14
15     else begin
16        count[3] <= !count[0];
17        count[2] <= count[3];
18        count[1] <= count[2];
19        count[0] <= count[1];
20     end
21
22   end
23  endmodule: johnson_counter
24
```

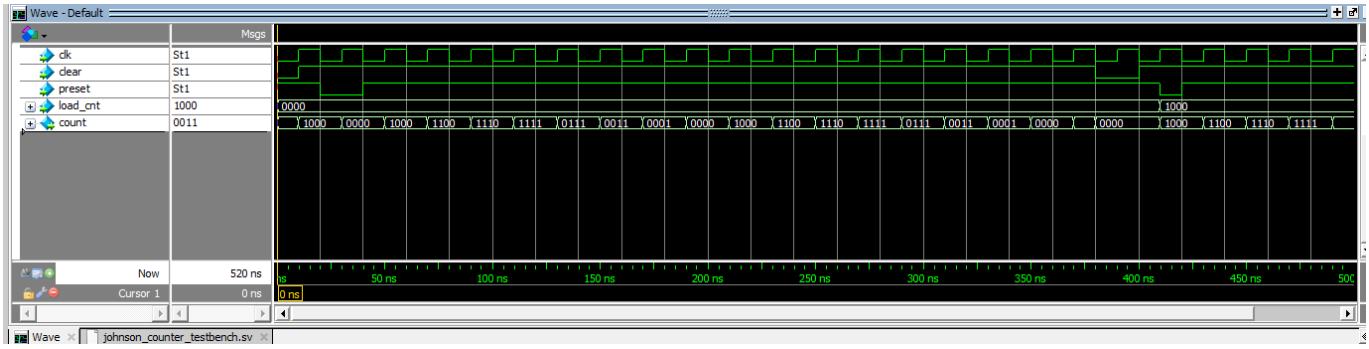
RTL netlist


Mux1
Mux2
Mux0
Mux3
Mux4
SEL[2..0]
DATA[7..0]
OUT
shift_mode[2..0]
sin
din[3..0]
load
clk
reset
Decoder0
IN[2..0]
OUT[7..0]
WideOr0
always1
shift_reg~[3..0]
shift_reg[3..0]
dout~[3..0]
dout~[7..4]
dout[3..0]
sout~reg0
sout
4'h0
1'h0
D
CLK
ENA
SCLR
CLRN
Q

Resource usage

	Resource	Usage
1	Estimated ALUTs Used	113
1	-- Combinational ALUTs	113
2	-- Memory ALUTs	0
3	-- LUT_REGs	0
2	Dedicated logic registers	8
3		
4	Estimated ALUTs Unavailable	9
1	-- Due to unpartnered combinational logic	9
2	-- Due to Memory ALUTs	0
5		
6	Total combinational functions	113
7	Combinational ALUT usage by number of inputs	
1	-- 7 input functions	3
2	-- 6 input functions	6
3	-- 5 input functions	21
4	-- 4 input functions	31
5	-- <=3 input functions	52
8		
9	Combinational ALUTs by mode	
1	-- normal mode	54
2	-- extended LUT mode	3
3	-- arithmetic mode	33
4	-- shared arithmetic mode	23
10		
11	Estimated ALUT/register pairs used	122
12		
13	Total registers	8
1	-- Dedicated logic registers	8
2	-- I/O registers	0
3	-- LUT_REGs	0
14		
15		
16	I/O pins	22
17		
18	DSP block 18-bit elements	0
19		

Testbench simulation waveform

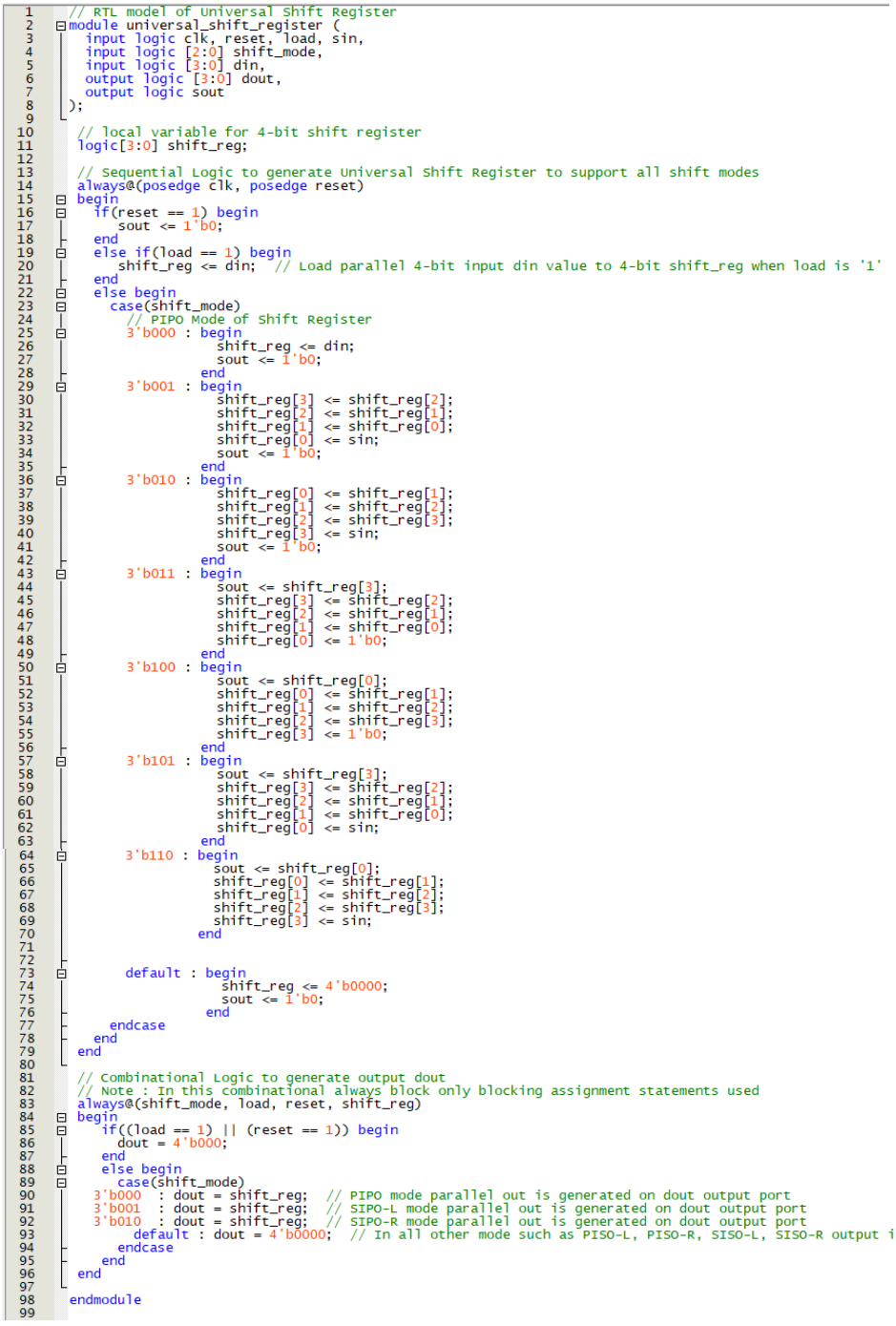


Upon review of the simulation waveform, the module works as a Johnson Counter by observations:

- At 45ns, when present goes low, this sets the shift register values to load_cnt, which is 4'b0000.
- Then on every positive edge of clk, we see 1's start to shift left from the right as a result of the initial 0's being negated and wrapped around
- After the shift register is filled with 1's, we see 0's start to shift left from the right by the same logic
- In total, there are 8 different states count can have, which is 2N where N = the 4 bits of the shift register
- We also see that upon clear going low, count goes and stays at 4'b0000 even at the positive edge of clk.
- When present goes low again, this time with load_cnt at a different value of 4'b1000, that get's loaded into the shift register

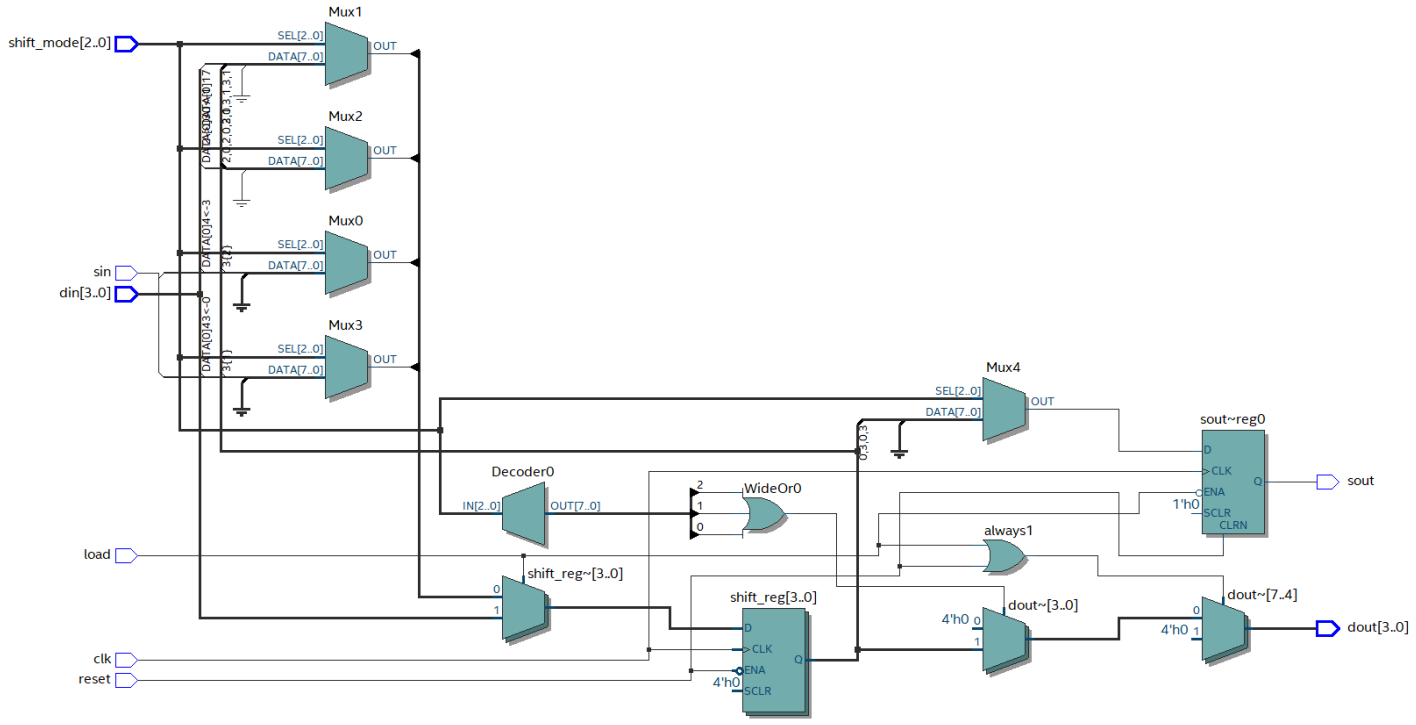
Universal Shift Register

Code



```
// RTL model of Universal Shift Register
module universal_shift_register (
  input logic clk, reset, load, sin,
  input logic [2:0] shift_mode,
  input logic [3:0] din,
  output logic [3:0] dout,
  output logic sout
);

  // local variable for 4-bit shift register
  logic[3:0] shift_reg;

  // Sequential Logic to generate Universal Shift Register to support all shift modes
  always@(posedge clk, posedge reset)
  begin
    if(reset == 1) begin
       sout <= 1'b0;
    end
    else if(load == 1) begin
       shift_reg <= din;  // Load parallel 4-bit input din value to 4-bit shift_reg when load is '1'
    end
    else begin
      case(shift_mode)
        // PIPO Mode of Shift Register
        3'b000 : begin
                   shift_reg <= din;
                   sout <= 1'b0;
                 end
        3'b001 : begin
                   shift_reg[3] <= shift_reg[2];
                   shift_reg[2] <= shift_reg[1];
                   shift_reg[1] <= shift_reg[0];
                   shift_reg[0] <= sin;
                   sout <= 1'b0;
                 end
        3'b010 : begin
                   shift_reg[0] <= shift_reg[1];
                   shift_reg[1] <= shift_reg[2];
                   shift_reg[2] <= shift_reg[3];
                   shift_reg[3] <= sin;
                   sout <= 1'b0;
                 end
        3'b011 : begin
                   sout <= shift_reg[3];
                   shift_reg[3] <= shift_reg[2];
                   shift_reg[2] <= shift_reg[1];
                   shift_reg[1] <= shift_reg[0];
                   shift_reg[0] <= 1'b0;
                 end
        3'b100 : begin
                   sout <= shift_reg[0];
                   shift_reg[0] <= shift_reg[1];
                   shift_reg[1] <= shift_reg[2];
                   shift_reg[2] <= shift_reg[3];
                   shift_reg[3] <= 1'b0;
                 end
        3'b101 : begin
                   sout <= shift_reg[3];
                   shift_reg[3] <= shift_reg[2];
                   shift_reg[2] <= shift_reg[1];
                   shift_reg[1] <= shift_reg[0];
                   shift_reg[0] <= sin;
                 end
        3'b110 : begin
                  sout <= shift_reg[0];
                  shift_reg[0] <= shift_reg[1];
                  shift_reg[1] <= shift_reg[2];
                  shift_reg[2] <= shift_reg[3];
                  shift_reg[3] <= sin;
                end

        default : begin
                    shift_reg <= 4'b0000;
                    sout <= 1'b0;
                  end
      endcase
    end
  end

  // Combinational Logic to generate output dout
  // Note : In this combinational always block only blocking assignment statements used
  always@(shift_mode, load, reset, shift_reg)
  begin
     if((load == 1) || (reset == 1)) begin
        dout = 4'b000;
     end
     else begin
        case(shift_mode)
     3'b000  : dout = shift_reg;  // PIPO mode parallel out is generated on dout output port
     3'b001  : dout = shift_reg;  // SIPO-L mode parallel out is generated on dout output port
     3'b010  : dout = shift_reg;  // SIPO-R mode parallel out is generated on dout output port
           default : dout = 4'b0000;  // In all other mode such as PISO-L, PISO-R, SISO-L, SISO-R output is
        endcase
     end
  end

endmodule
```

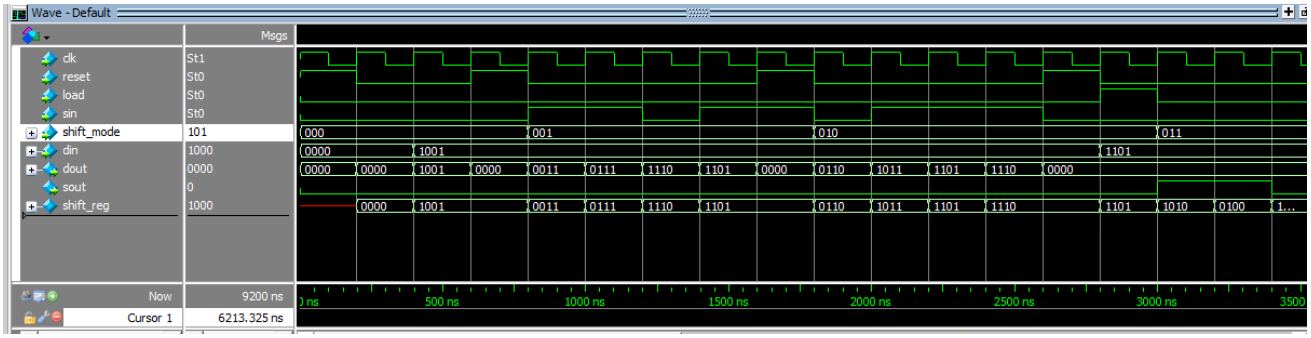
RTL netlist


shift_mode[2..0]
sin
din[3..0]
load
clk
reset
Mux1
Mux2
Mux0
Mux3
Mux4
SEL[2..0]
DATA[7..0]
OUT
Decoder0
IN[2..0]
OUT[7..0]
WideOr0
always1
shift_reg~[3..0]
shift_reg[3..0]
dout~[3..0]
dout~[7..4]
sout~reg0
D
CLK
ENA
SCLR
CLRN
Q
4'h0
1'h0
sout
dout[3..0]

Resource usage

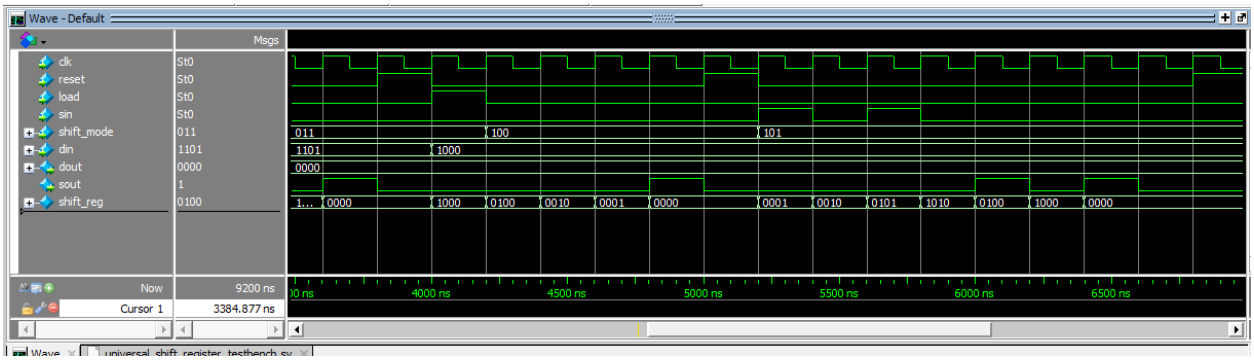
	Resource	Usage
1	Estimated ALUTs Used	11
1	-- Combinational ALUTs	11
2	-- Memory ALUTs	0
3	-- LUT_REGs	0
2	Dedicated logic registers	5
3		
4	Estimated ALUTs Unavailable	7
1	-- Due to unpartnered combinational logic	7
2	-- Due to Memory ALUTs	0
5		
6	Total combinational functions	11
7	Combinational ALUT usage by number of inputs	
1	-- 7 input functions	3
2	-- 6 input functions	4
3	-- 5 input functions	2
4	-- 4 input functions	1
5	-- <=3 input functions	1
8		
9	Combinational ALUTs by mode	
1	-- normal mode	8
2	-- extended LUT mode	3
3	-- arithmetic mode	0
4	-- shared arithmetic mode	0
10		
11	Estimated ALUT/register pairs used	18
12		
13	Total registers	5
1	-- Dedicated logic registers	5
2	-- I/O registers	0
3	-- LUT_REGs	0
14		
15		
16	I/O pins	16
17		
18	DSP block 18-bit elements	0

Testbench simulation waveform

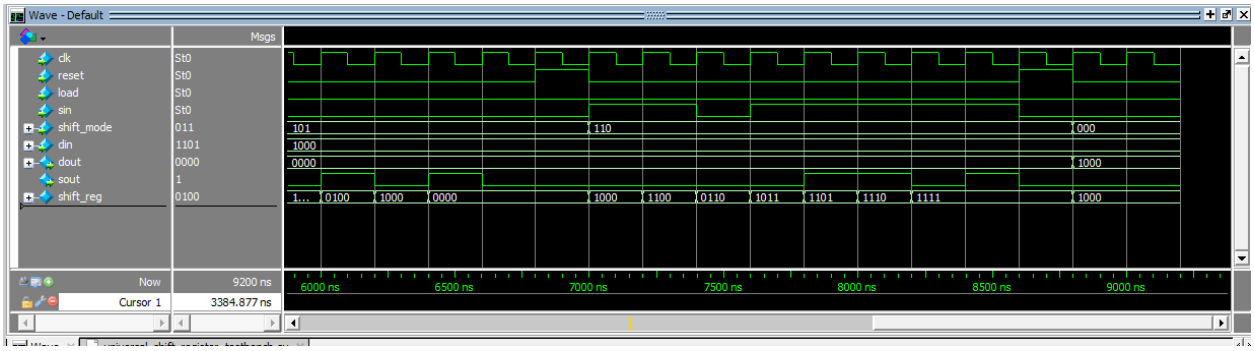


Checking shift modes:

- 0000 (PIPO) works because when din is 1001, so is dout.
- 0001 (SIPO-L) works because from an initial shift register state of 1001, upon positive edge of clk, and sin being high, we see a 1 inserted from the right, and shift_reg turns to 0011. Next shift also has sin as high, so another 1 is insert to make shift_reg 0111. However, next positive edge of clk has low sin, so now a 0 is inserted to make 1110. While these shifts are happening, dout reflects the values of shift_reg in parallel
- 0010 (SIPO-R) works similarly to 0001, but now new values from sin are inserted from the right.



- 0011 (PISO-L) works because upon load being high, din with 1000 is loaded into register. Then upon positive edge of clk, it is shifted left, and a 0 bit is inserted to the rightmost register and the leftmost bit is moved to sout which is both 1's for the next two clocks, but 0 for the third. Also, dout is 0 since this is a serial out shift mode.
- 0100 (PISO-R) works similar to 0011 but bits in shift_register are shifted to the right, the 0 bit is inserted to the leftmost register, and the rightmost bit goes to sout.
- 0101 (SISO-L) works because upon each positive edge of clk, the value of sin is loaded into the rightmost register and the leftmost register of shift_register is loaded into sout. Dout stays 0000 because this shift operation is serial.
-



- 0110 (SISO-R) works similar to 0110 but sin is loaded into the leftmost register, and the rightmost register's value is fed into sout.
- For all shifting operations, when reset is high, sout and dout is always low which is expected