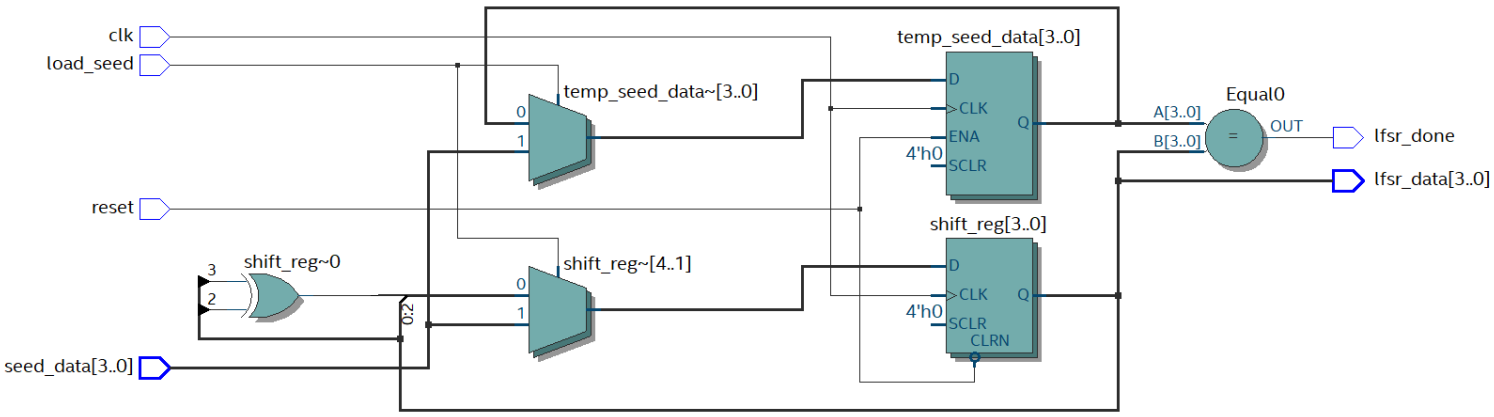
ECE 111 Winter 2022
HW4
Hao Le A15547504

Barrel Shifter

Code

```
//RTL Model for Linear Feedback Shift Register
module lfsr
#(parameter N = 4) // Number of bits for LFSR
(
   input logic clk, reset, load_seed,
   input logic[N-1:0] seed_data,
   output logic lfsr_done,
   output logic[N-1:0] lfsr_data
);

logic[N-1:0] shift_reg;
logic[N-1:0] temp_seed_data;



always_ff@(posedge clk, negedge reset)
begin
   if(reset == 0) begin
      shift_reg <= 0;
   end

   else if(load_seed == 1) begin
      shift_reg <= seed_data;
      temp_seed_data <= seed_data;
   end

   else begin

      case(N)
         2 : begin
               shift_reg[0] <= shift_reg[1] ^ shift_reg[0];
             end
         3 : begin
               shift_reg[0] <= shift_reg[2] ^ shift_reg[1];
             end
         4 : begin
               shift_reg[0] <= shift_reg[3] ^ shift_reg[2];
             end
         5 : begin
               shift_reg[0] <= shift_reg[4] ^ shift_reg[2];
             end
         6 : begin
               shift_reg[0] <= shift_reg[5] ^ shift_reg[4];
             end
         7 : begin
               shift_reg[0] <= shift_reg[6] ^ shift_reg[5];
             end
         8 : begin
               shift_reg[0] <= shift_reg[7] ^ shift_reg[5] ^ shift_reg[4] ^ shift_reg[3];
             end
         default : shift_reg[0] <= 1'b0;
      endcase


      for (int i=0; i<N-1; i++) begin
         shift_reg[i+1] <= shift_reg[i];
      end
   end
end

always_comb begin
   lfsr_data <= shift_reg;

   if (shift_reg === temp_seed_data) begin
      lfsr_done <= 1'b1;
   end else begin
      lfsr_done <= 1'b0;
   end
end

//student to add implementation for LFSR code

endmodule: lfsr
```

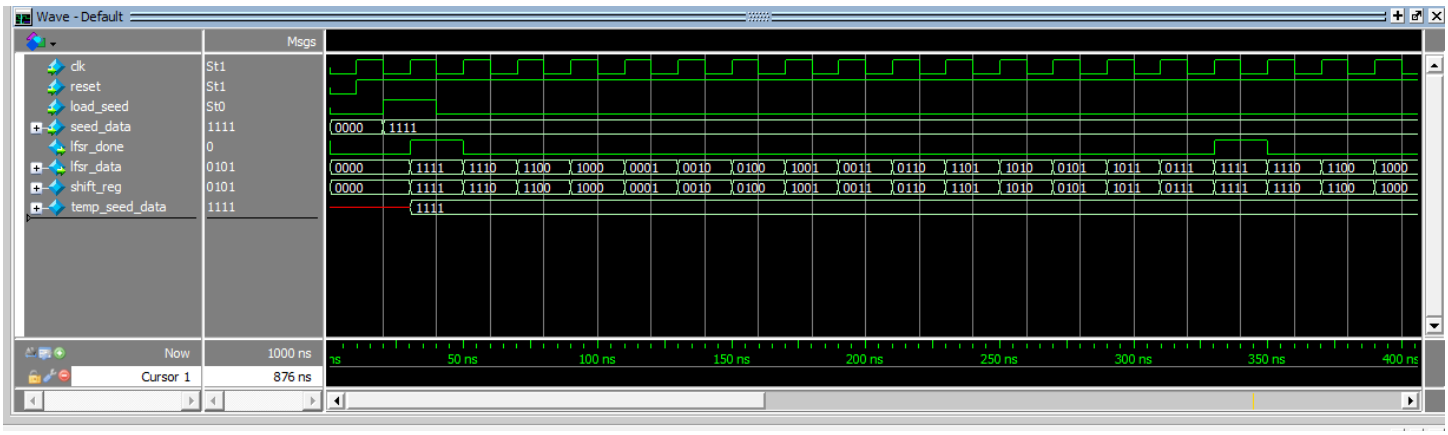
RTL netlist


clk
load_seed
reset
seed_data[3..0]
temp_seed_data~[3..0]
shift_reg~0
shift_reg~[4..1]
temp_seed_data[3..0]
shift_reg[3..0]
D
CLK
ENA
SCLR
Q
4'h0
CLRN
Equal0
A[3..0]
B[3..0]
OUT
lfsr_done
lfsr_data[3..0]

Resource usage

	Resource	Usage
1	Estimated ALUTs Used	7
1	-- Combinational ALUTs	7
2	-- Memory ALUTs	0
3	-- LUT_REGs	0
2	Dedicated logic registers	8
3		
4	Estimated ALUTs Unavailable	0
1	-- Due to unpartnered combinational logic	0
2	-- Due to Memory ALUTs	0
5		
6	Total combinational functions	7
7	Combinational ALUT usage by number of inputs	
1	-- 7 input functions	0
2	-- 6 input functions	0
3	-- 5 input functions	1
4	-- 4 input functions	2
5	-- <=3 input functions	4
8		
9	Combinational ALUTs by mode	
1	-- normal mode	7
2	-- extended LUT mode	0
3	-- arithmetic mode	0
4	-- shared arithmetic mode	0
10		
11	Estimated ALUT/register pairs used	10
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	12
17		
18	DSP block 18-bit elements	0
19		

Testbench simulation waveform



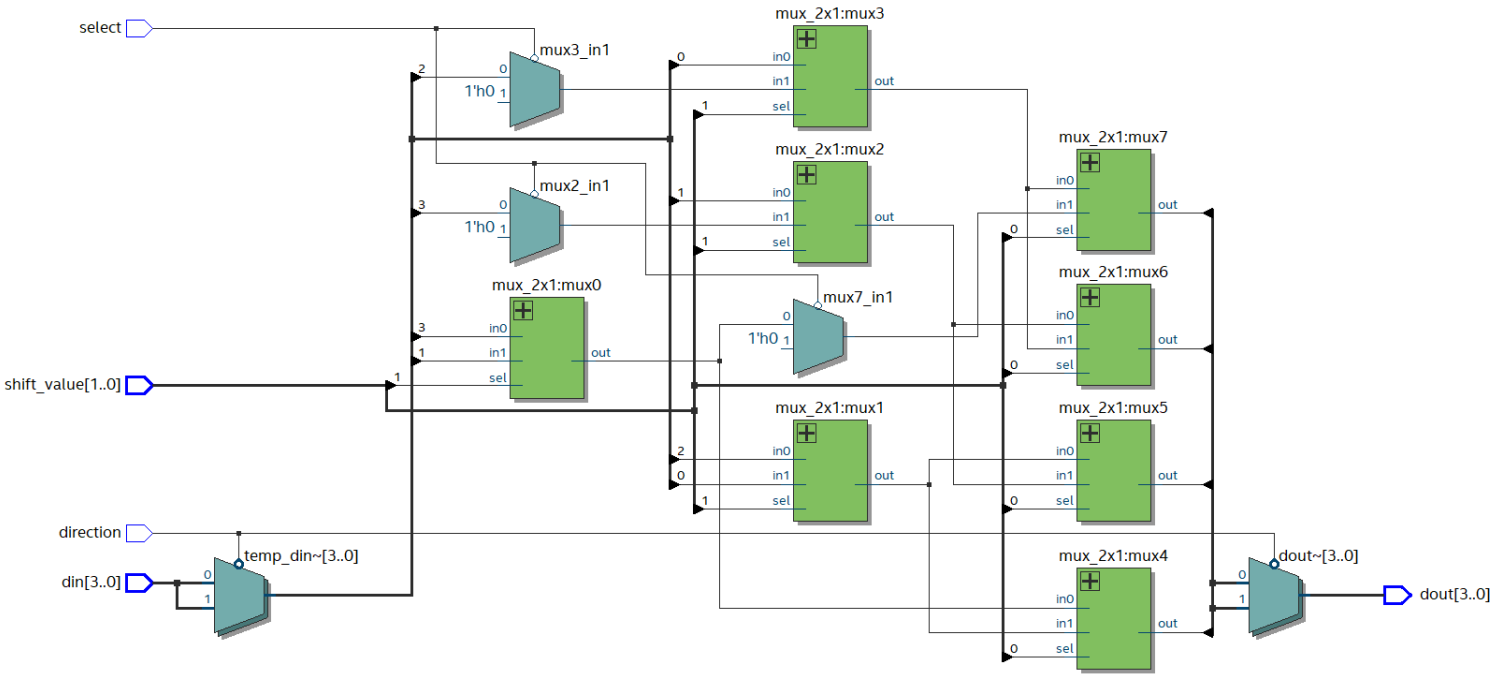
- The testbench in this case was for a 4-bit LFSR, but will work widths 2-8. We see that the seed is 1111, and upon the positive edge of clock when load_seed was high, the output data of the LFSR takes on 1111, with lfsr_done going high to signify the start of the pattern. Then, for every positive edge of the clock after, lfsr_data takes on all the other 14 state patterns in a pseudorandom manner before going back to 1111, upon which lfsr_done goes high for one clock period to signify the repetition of the pattern. This is the expected and correct behavior

Barrel Shifter

Code

```
module barrel_shifter (
   input logic select,  // select=0 shift operation, select=1 rotate operation
   input logic direction, // direction=0 right move, direction=1 left move
   input logic[1:0] shift_value, // number of bits to be shifted (0, 1, 2 or 3)
   input logic[3:0] din,
   output logic[3:0] dout
);

  // Students to add code for barrel shifter

  logic[3:0] temp_din, temp_dout;


  logic mux0_out, mux1_out, mux2_out, mux3_out;

  logic mux2_in1, mux3_in1, mux7_in1;


mux_2x1 mux0( .in0(temp_din[0]),
              .in1(temp_din[2]),
              .sel(shift_value[1]),
              .out(mux0_out));

mux_2x1 mux1( .in0(temp_din[1]),
              .in1(temp_din[3]),
              .sel(shift_value[1]),
              .out(mux1_out));

mux_2x1 mux2( .in0(temp_din[2]),
              .in1(mux2_in1),
              .sel(shift_value[1]),
              .out(mux2_out));

mux_2x1 mux3( .in0(temp_din[3]),
              .in1(mux3_in1),
              .sel(shift_value[1]),
              .out(mux3_out));

mux_2x1 mux4( .in0(mux0_out),
              .in1(mux1_out),
              .sel(shift_value[0]),
              .out(temp_dout[0]));

mux_2x1 mux5( .in0(mux1_out),
              .in1(mux2_out),
              .sel(shift_value[0]),
              .out(temp_dout[1]));

mux_2x1 mux6( .in0(mux2_out),
              .in1(mux3_out),
              .sel(shift_value[0]),
              .out(temp_dout[2]));

mux_2x1 mux7( .in0(mux3_out),
              .in1(mux7_in1),
              .sel(shift_value[0]),
              .out(temp_dout[3]));


always_comb begin

   if(select == 1'b0) begin //shift operation
      mux2_in1 <= 1'b0;
      mux3_in1 <= 1'b0;
      mux7_in1 <= 1'b0;
   end
   else begin //rotate operation
      mux2_in1 <= temp_din[0];
      mux3_in1 <= temp_din[1];
      mux7_in1 <= mux0_out;
   end

   if(direction == 1'b0) begin //right direction
      temp_din <= din;
      dout <= temp_dout;
   end
   else begin
      temp_din[0] <= din[3];
      temp_din[1] <= din[2];
      temp_din[2] <= din[1];
      temp_din[3] <= din[0];
      dout[0] <= temp_dout[3];
      dout[1] <= temp_dout[2];
      dout[2] <= temp_dout[1];
      dout[3] <= temp_dout[0];
   end

end


endmodule: barrel_shifter
```

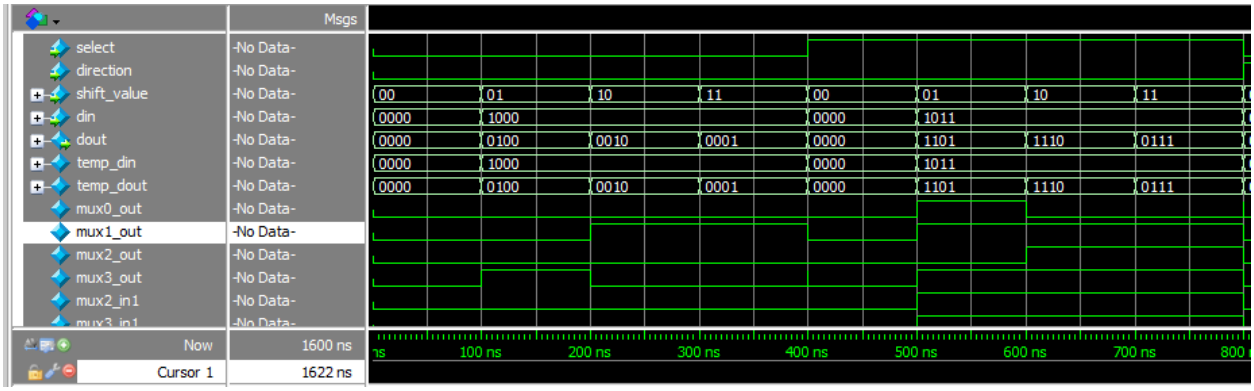
RTL netlist


select
mux3_in1
1'h0
mux_2x1:mux3
mux2_in1
1'h0
mux_2x1:mux2
mux_2x1:mux7
mux_2x1:mux0
mux7_in1
1'h0
mux_2x1:mux6
shift_value[1..0]
mux_2x1:mux1
mux_2x1:mux5
direction
temp_din~[3..0]
din[3..0]
mux_2x1:mux4
dout~[3..0]
dout[3..0]
in0
in1
sel
out

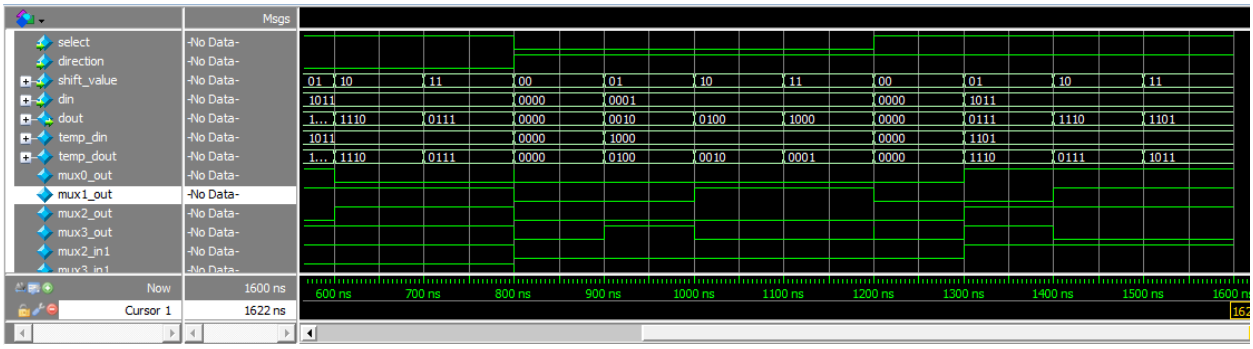
Resource usage

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

Testbench simulation waveform



- The first half of the simulation is in right direction mode since direction input is low. From 0 to 400 ns, select is low meaning it is in shift mode. 1000 is loaded in, and shift value is incremented from 1 to 3. We subsequently see the loaded in data shifted to the right by the specified shift value, with a 0 being appended to the trailing end at dout e.g. 1000 to 0100. This is the correct behavior of right shift mode. After 400ns, select goes high, and now the module is in right rotate mode, so now the trailing end is appended with the overflow bit. We see this is the behavior observed e.g. 1101 loaded in becomes 1101 after being rotated to the right by 1.



- The second half of the simulation is in left direction mode since direction goes high. Thus we can observe the correct behaviors of left shift and rotation by 1, 2, and 3 bits.