LFSR

- **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
);
//student to add implementation for LFSR code
reg [N:1] r_LFSR;// changed N-1 to N because out of range error in N==4 part of code
reg xor_data;
always@(posedge clk, negedge reset)begin //at end
if(!reset)begin
r_LFSR = 1'b0;
//code
 end
else if(load_seed)
r_LFSR <= seed_data;
else
//TA suggestion on how to shift the bits
r_LFSR <= {r_LFSR[N-1:1], xor_data};
//code
end
always@(posedge clk, negedge reset)begin
//else begin
if (N==2)begin
xor_data = r_LFSR[2] ^ r_LFSR[1];
end
else if (N==3)begin
xor_data = r_LFSR[3] ^ r_LFSR[2];
end
 else if (N==4)begin
xor_data = r_LFSR[4] ^ r_LFSR[3];
end
else if (N==5)begin
xor_data = r_LFSR[5] ^ r_LFSR[3];
end
else if (N==6)begin
xor_data = r_LFSR[6] ^ r_LFSR[5];
end
```

```
else if (N==7)begin
xor_data = r_LFSR[7] ^ r_LFSR[6];
end
else if (N==8)begin
xor_data = r_LFSR[8] ^ r_LFSR[6] ^ r_LFSR[5] ^ r_LFSR[4];
end
end
//load the xor data into LSB
assign lfsr_data = r_LFSR;
//lfsr done
assign lfsr_done = ((reset != 0) && (lfsr_data[N-1:0] != 1'b0) && (lfsr_data[N-1:0] == seed_data))
? 1'b1 : 1'b0;
endmodule: lfsr
```

- **Testbench**

```
`timescale 1ns/1ns
//LFSR Testbench Code
module lfsr_testbench;
 parameter N = 4;
 logic clock;
 logic [N-1:0] lfsr_data, seed_data;
 logic lfsr_done;
 logic reset, load_seed;
 lfsr #(.N(N)) design_inst(
 .clk(clock),
 .reset(reset),
 .load_seed(load_seed),
 .seed_data(seed_data),
 .lfsr_data(lfsr_data),
 .lfsr_done(lfsr_done)
 );
 initial begin
 // Initialize Inputs
 reset = 0;
 load_seed = 0;
 clock = 0;
 seed_data = 4'b0000;
 // Wait 10 ns for global reset to finish and start counter
 #10;
 reset = 1;
 #10;
 load_seed = 1;
 seed_data = 4'b1111;
 #20;
```

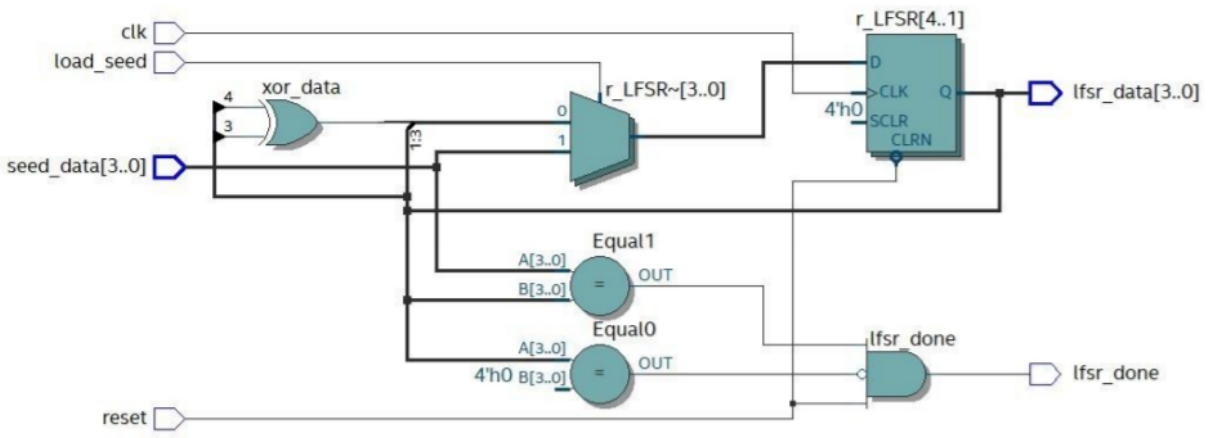
```
 load_seed = 0;
 #200;
 // terminate simulation
 $finish();
 end
 // Clock generator logic
 always@(clock) begin
 #10ns clock <= !clock;
 end
 // Print input and output signals
 initial begin
 $monitor(" time=%0t, reset=%b clk=%b load_seed=%b count=%d", $time, reset, clock,
load_seed, lfsr_data);
 end
endmodule
```

- **Resource Usage**

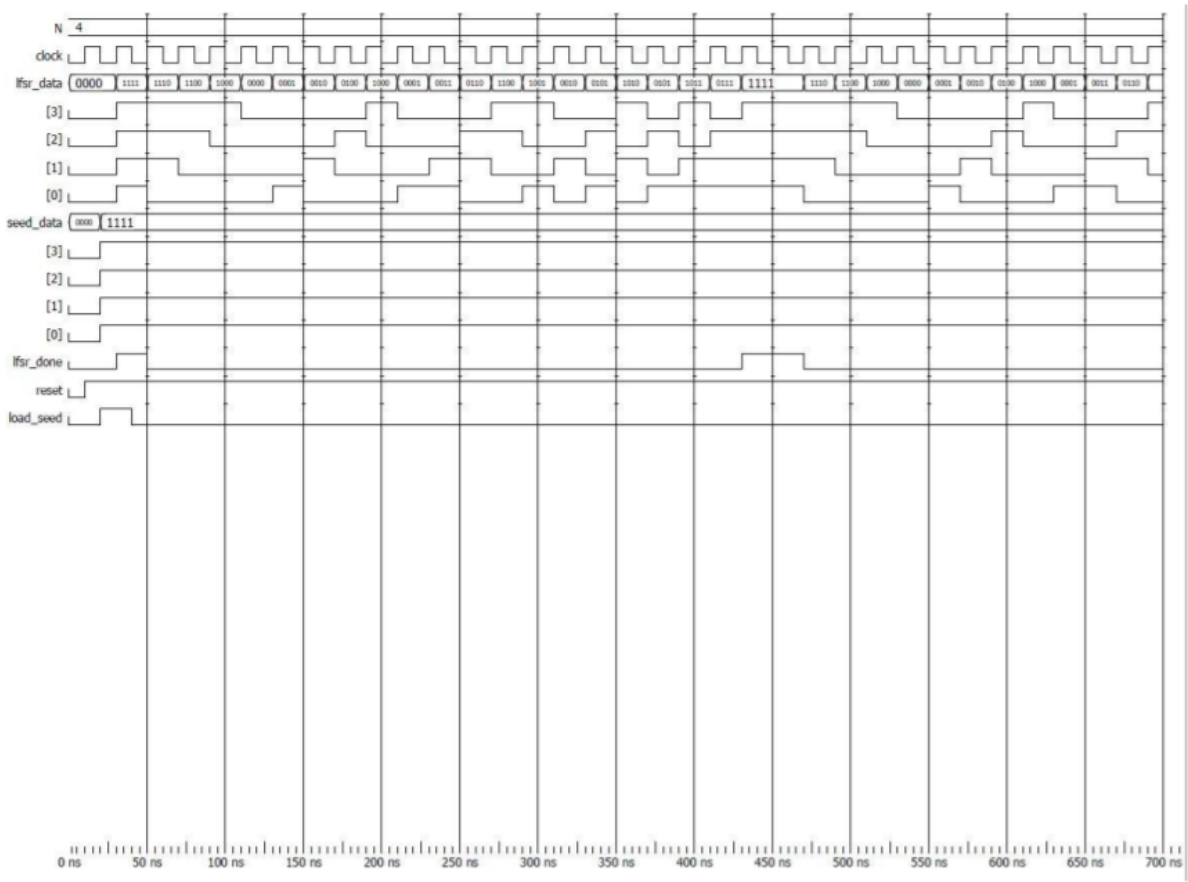
	Resource	Usage
1	Estimated ALUTs Used	6
1	-- Combinational ALUTs	6
2	-- Memory ALUTs	0
3	-- LUT_REGs	0
2	Dedicated logic registers	4
3		
4	Estimated ALUTs Unavailable	2
1	-- Due to unpartnered combinational logic	2
2	-- Due to Memory ALUTs	0
5		
6	Total combinational functions	6
7	Combinational ALUT usage by number of inputs	
1	-- 7 input functions	0
2	-- 6 input functions	2
3	-- 5 input functions	0
4	-- 4 input functions	1
5	-- <=3 input functions	3
8		
9	Combinational ALUTs by mode	
1	-- normal mode	6
2	-- extended LUT mode	0
3	-- arithmetic mode	0
4	-- shared arithmetic mode	0
10		
11	Estimated ALUT/register pairs used	8
12		
13	Total registers	4
1	-- Dedicated logic registers	4
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		
20	Maximum fan-out node	reset~input
21	Maximum fan-out	5
22	Total fan-out	54

	Resource	Usage
23	Average fan-out	1.59

- **RTL**


clk
load_seed
seed_data[3..0]
xor_data
4
3
r_LFSR~[3..0]
0
1
13
r_LFSR[4..1]
D
>CLK
Q
4'h0
SCLR
CLRN
lfsr_data[3..0]
Equal1
A[3..0]
B[3..0]
=
OUT
Equal0
A[3..0]
4'h0 B[3..0]
=
OUT
lfsr_done
lfsr_done
reset

- **Waveform**


N 4
clock
lfsr_data 0000 1111
[3]
[2]
[1]
[0]
seed_data 0000 1111
[3]
[2]
[1]
[0]
lfsr_done
reset
load_seed
0 ns
50 ns
100 ns
150 ns
200 ns
250 ns
300 ns
350 ns
400 ns
450 ns
500 ns
550 ns
600 ns
650 ns
700 ns