C

3C7 DIGITAL SYSTEMS DESIGN LABORATORY

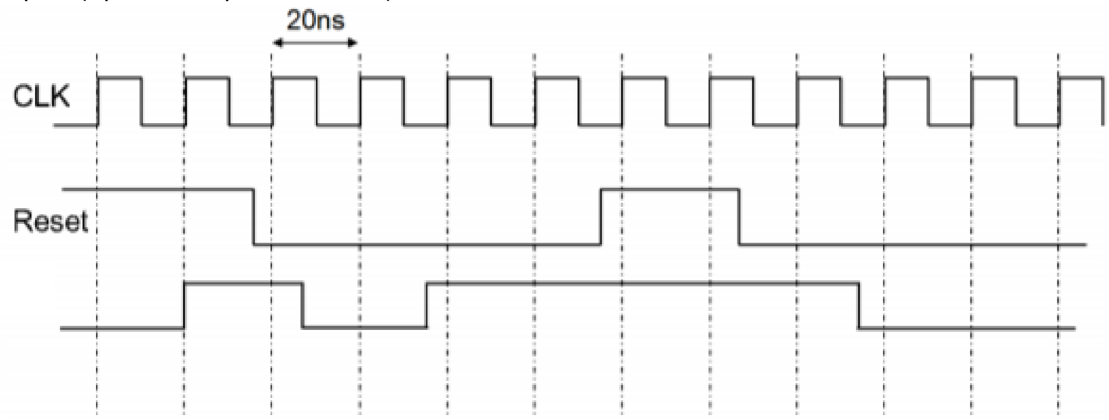
Lab F

Department of Electronic and Electrical Engineering

(e-Report submission)

Description:

A. LFSR Implementation for given LFSR setup (XOR 13-bit)
B. Advance part (optional implementation)



Submitted by:

Rohan Taneja
19323238
Submission Date: 31/03/2020

Part A. LFSR: Implementation

1. Task:

Linear Feedback Shift Register implemented is done as explained in Lecture 6. The task was to implement LFSR with feedback as XOR for N=13 bits. Implementation of LFSR required input clock and asynchronous active-high reset.

2. Block Diagram:

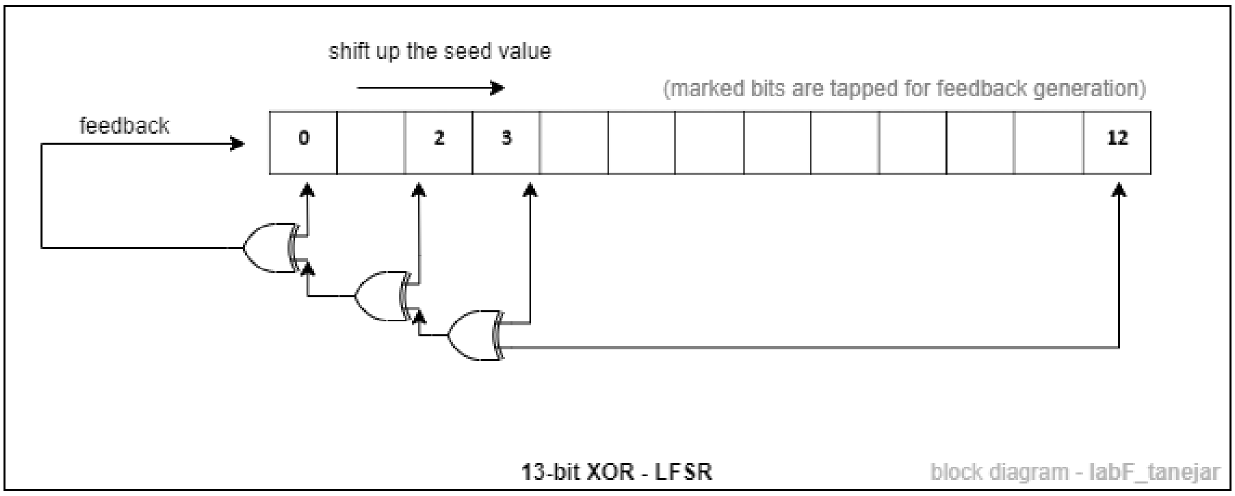

shift up the seed value
(marked bits are tapped for feedback generation)
feedback
0
2
3
12
13-bit XOR - LFSR
block diagram - labF_tanejar

Fig A. (i) – Block diagram made using draw.io

3. Testbench:

Simulation takes place under 20 ns clock period and rest high set till 10 cycles. The testbench used is as follows:

```
module lfsr_tb;
    reg clk, reset;
    wire max_tick;
    wire lfsr_out;

    lfsr uut(
        .clk(clk),
        .reset(reset),
        .lfsr_out(lfsr_out),
        .max_tick(max_tick)
    );
    // oscillate clock 20 ns period
    always
    begin
        clk = 1'b1; //high
        #10;
        clk = 1'b0; //low
        #10;
    end

    // reset for first 2 clock cycles
    initial
    begin
        reset = 1'b1;
        #200; // 10 clock cycles high reset
        reset = 1'b0;
    end
endmodule
```

Fig A. (ii) - Testbench for LFSR XOR 13-bit simulation

4. Simulation:

The simulation of my LFSR XOR feedback 13-bit module is as follows.

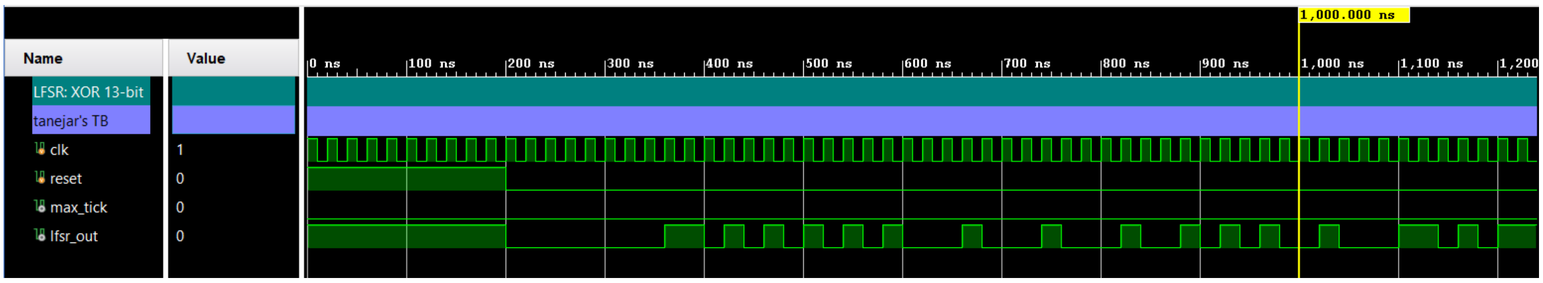


Fig A. (iii) - LFSR simulation for XOR 13-bit – reset after 10th cycle

The image below shows that max_tick signal implementation works well for my 13-bit LFSR implementation as the seed value matches itself after $2^N – 1$ cycles.

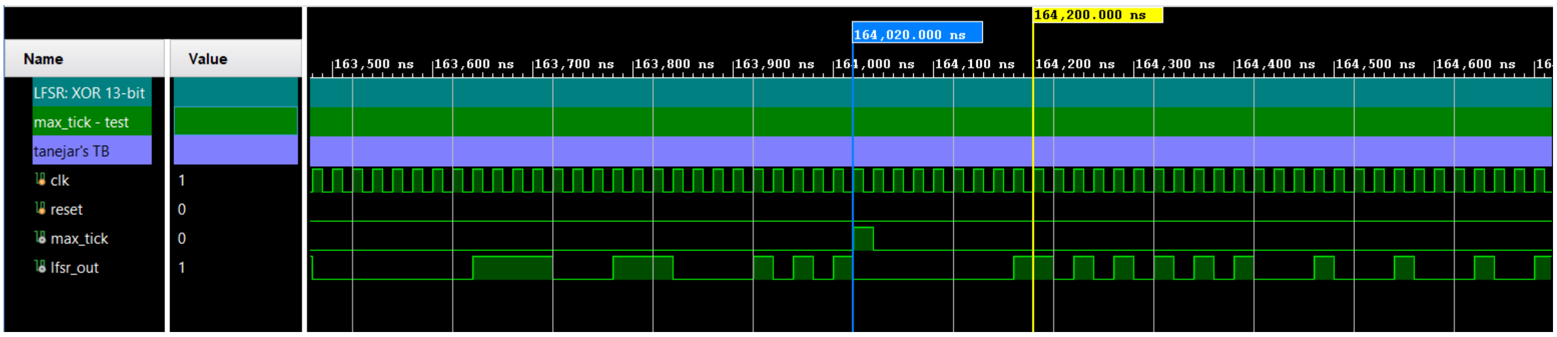


Fig A. (iv) - LFSR simulation for XOR 13-bit – max_tick signal after 8201th (8191 + 10th reset) cycle

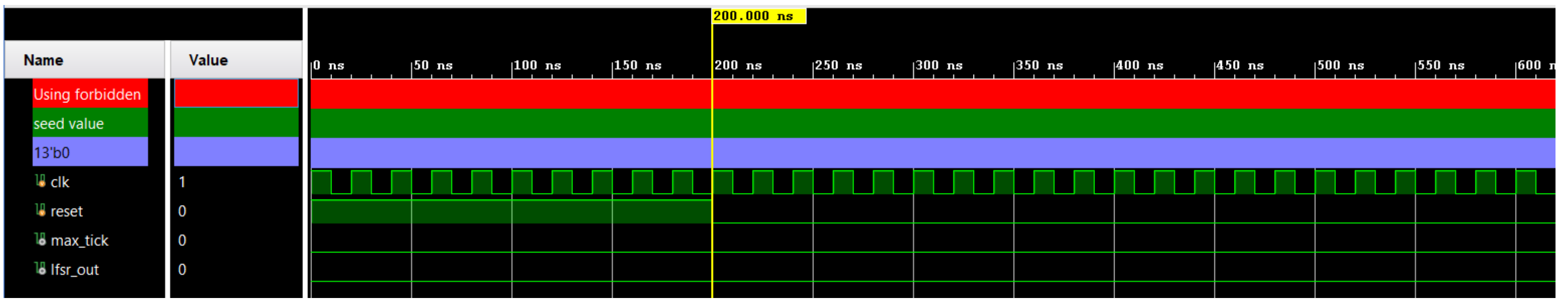
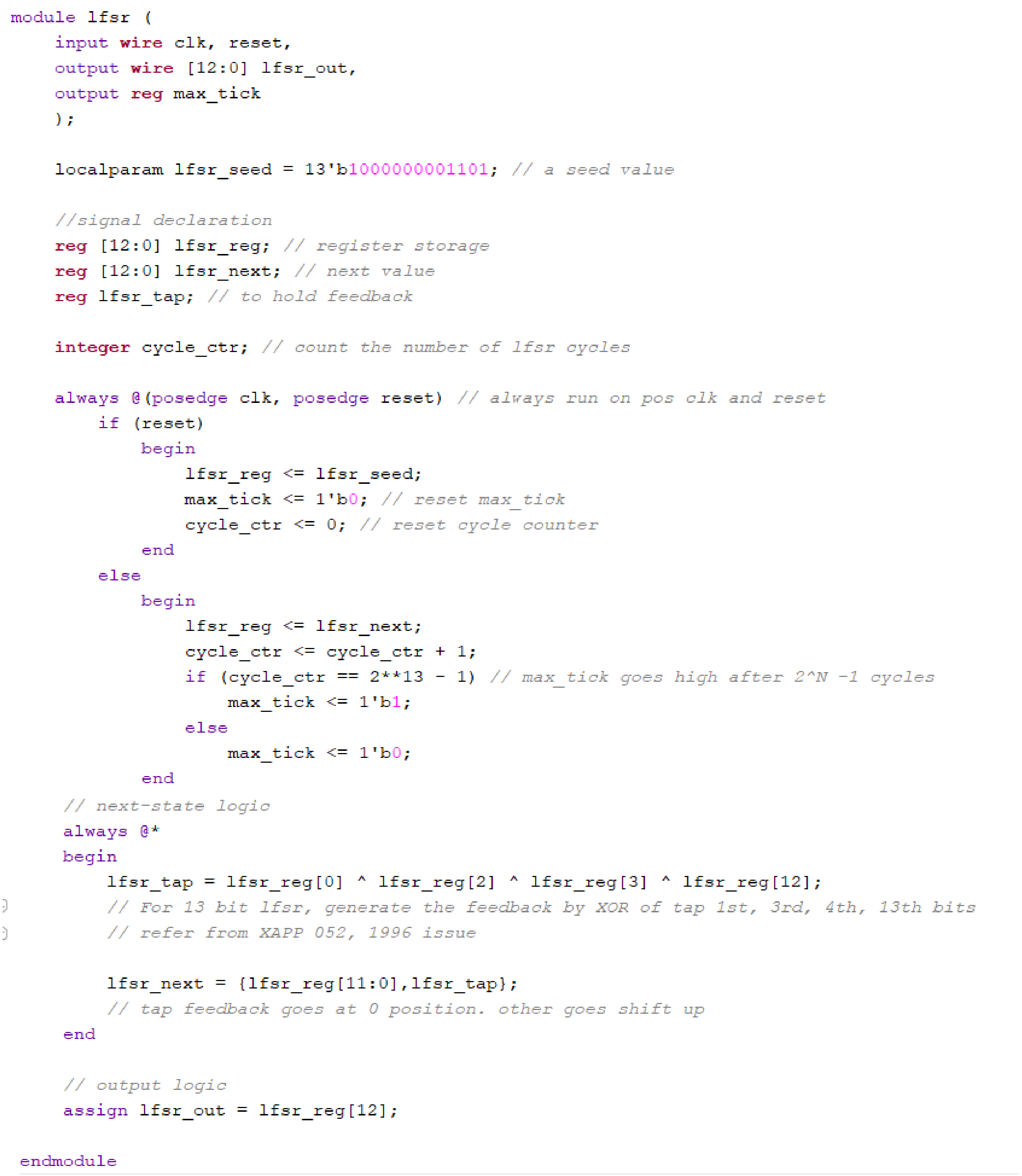


Fig A. (v) - LFSR simulation for XOR 13-bit – forbidden seed value – *13'b0000000000000*

5. Module:

The module LFSR is implemented as follows in Verilog. The local parameter seed value is set in the module. For the forbidden seed value case, value = 13’b0 was used.



```
module lfsr (
    input wire clk, reset,
    output wire [12:0] lfsr_out,
    output reg max_tick
    );

    localparam lfsr_seed = 13'b1000000001101; // a seed value

    //signal declaration
    reg [12:0] lfsr_reg; // register storage
    reg [12:0] lfsr_next; // next value
    reg lfsr_tap; // to hold feedback

    integer cycle_ctr; // count the number of lfsr cycles

    always @(posedge clk, posedge reset) // always run on pos clk and reset
        if (reset)
            begin
                lfsr_reg <= lfsr_seed;
                max_tick <= 1'b0; // reset max_tick
                cycle_ctr <= 0; // reset cycle counter
            end
        else
            begin
                lfsr_reg <= lfsr_next;
                cycle_ctr <= cycle_ctr + 1;
                if (cycle_ctr == 2**13 - 1) // max_tick goes high after 2^N -1 cycles
                    max_tick <= 1'b1;
                else
                    max_tick <= 1'b0;
            end
     // next-state logic
     always @*
     begin
         lfsr_tap = lfsr_reg[0] ^ lfsr_reg[2] ^ lfsr_reg[3] ^ lfsr_reg[12];
         // For 13 bit lfsr, generate the feedback by XOR of tap 1st, 3rd, 4th, 13th bits
         // refer from XAPP 052, 1996 issue

         lfsr_next = {lfsr_reg[11:0],lfsr_tap};
         // tap feedback goes at 0 position. other goes shift up
     end

     // output logic
     assign lfsr_out = lfsr_reg[12];

 endmodule
```

Fig A. (vi) – module lfsr.v – used for the implementation

Part B. Advanced Implementation

1. Task:

To write an extra module which works as counter for MSB (or the feedback tap) generated from LFSR operation and count the number of 0s and 1s feed generated on runtime.

2. Schematic:

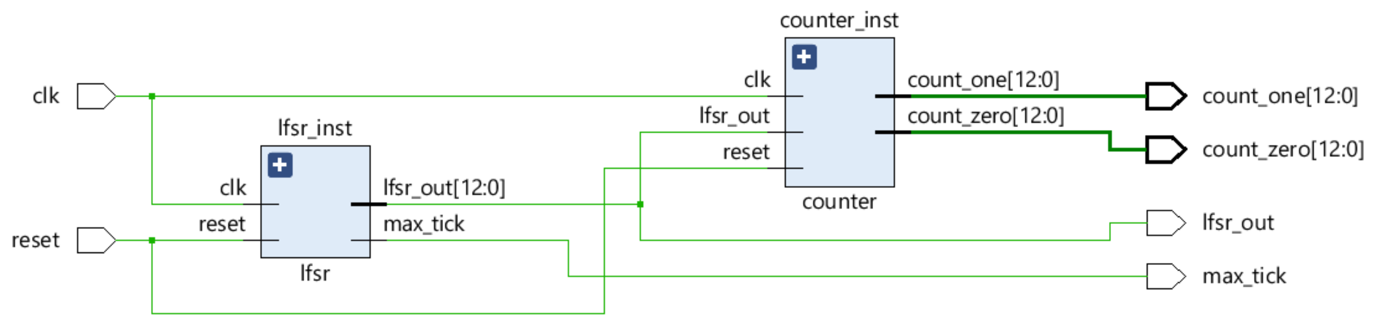


Fig B. (i) – Schematic for Advance Implementation

3. Testbench:

Simulation takes place under 20 ns clock period and rest high set till 10 cycles. The testbench used is as follows:

```
module advance_tb;
    reg clk, reset;
    wire max_tick;
    wire lfsr_out;
    wire [12:0] count_zero;
    wire [12:0] count_one;

    advance uut ( // Instantiate the Advance module
        .clk(clk),
        .reset(reset),
        .count_zero(count_zero),
        .count_one(count_one),
        .main_max_tick(max_tick),
        .main_lfsr_out(lfsr_out)
        );

    always
    // oscillate clock 20 ns period
    begin
        clk = 1'b1; //high
        #10;
        clk = 1'b0; //low
        #10;
    end
    initial
    begin
        reset = 1'b1;
        #200; // 10 clock cycles high reset
        reset = 1'b0;
    end
endmodule
```

Fig B. (ii) - Testbench for Advanced part simulation

4. Simulation:

The simulation of my advanced part for LFSR XOR feedback 13-bit module is as follows:

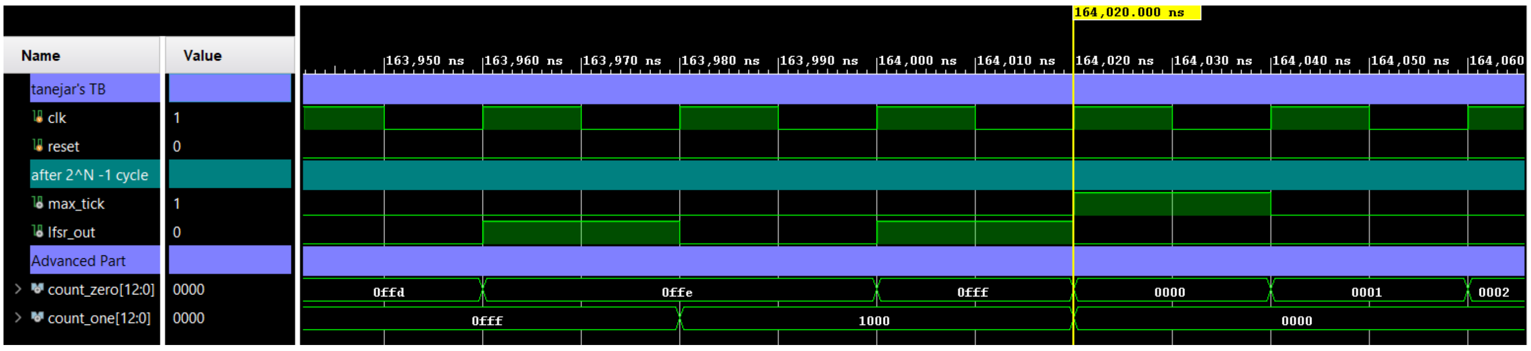


Fig B. (iii) – Advanced part – total 0s (0fff) and 1s (1000) counts in full run.

<u>Total 0s counted</u> = 4095

<u>Total 1s counted</u> = 4096

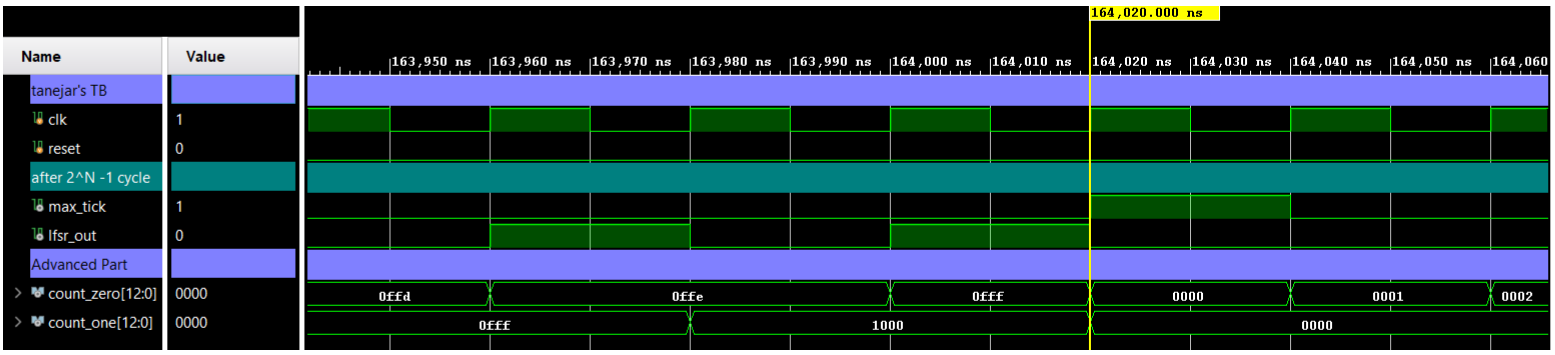


Fig B. (iv) – Advanced part – reset counter - max_tick signal after 8201th (8191 + 10th reset) cycle

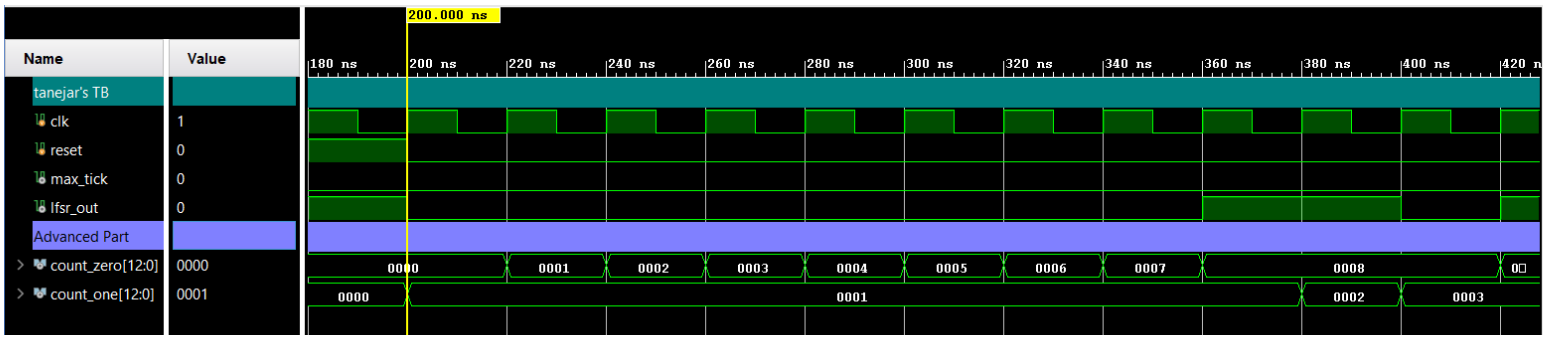


Fig B. (v) – Advanced part - the 0s and 1s counter start after reset

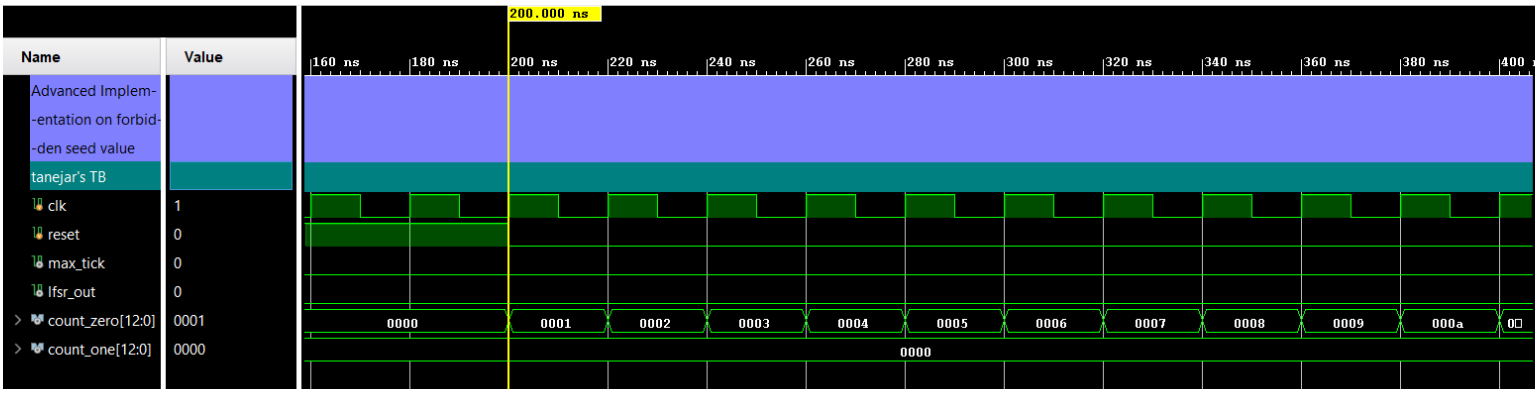
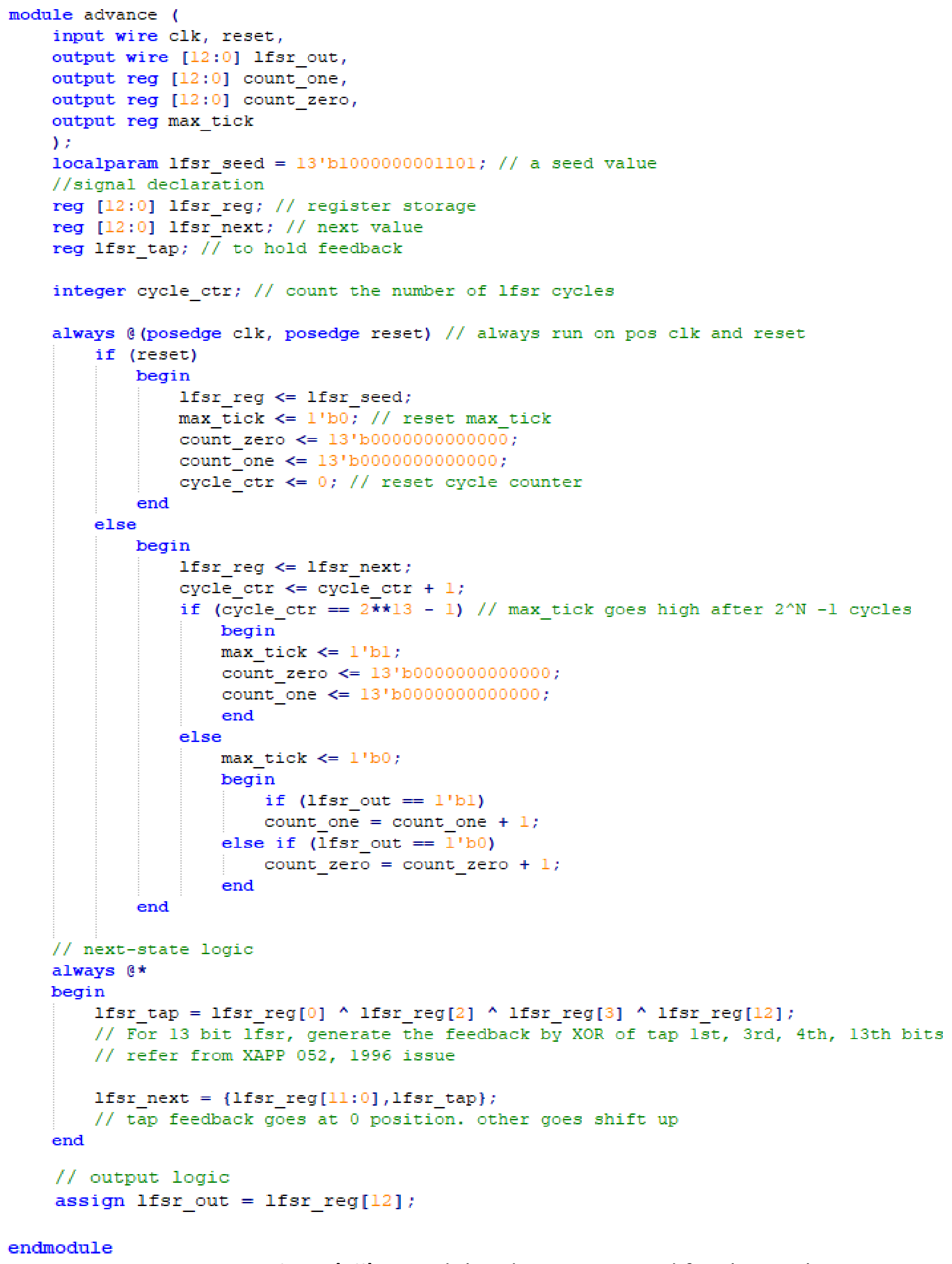


Fig B. (vi) - LFSR simulation for XOR 13-bit – forbidden seed value – *13'b00000000*

5. Module:



```verilog
module advance (
    input wire clk, reset,
    output wire [12:0] lfsr_out,
    output reg [12:0] count_one,
    output reg [12:0] count_zero,
    output reg max_tick
    );
    localparam lfsr_seed = 13'b1000000001101; // a seed value
    //signal declaration
    reg [12:0] lfsr_reg; // register storage
    reg [12:0] lfsr_next; // next value
    reg lfsr_tap; // to hold feedback

    integer cycle_ctr; // count the number of lfsr cycles

    always @(posedge clk, posedge reset) // always run on pos clk and reset
        if (reset)
            begin
                lfsr_reg <= lfsr_seed;
                max_tick <= 1'b0; // reset max_tick
                count_zero <= 13'b0000000000000;
                count_one <= 13'b0000000000000;
                cycle_ctr <= 0; // reset cycle counter
            end
        else
            begin
                lfsr_reg <= lfsr_next;
                cycle_ctr <= cycle_ctr + 1;
                if (cycle_ctr == 2**13 - 1) // max_tick goes high after 2^N -1 cycles
                    begin
                    max_tick <= 1'b1;
                    count_zero <= 13'b0000000000000;
                    count_one <= 13'b0000000000000;
                    end
                else
                    max_tick <= 1'b0;
                    begin
                        if (lfsr_out == 1'b1)
                        count_one = count_one + 1;
                    else if (lfsr_out == 1'b0)
                        count_zero = count_zero + 1;
                    end
            end

    // next-state logic
    always @*
    begin
        lfsr_tap = lfsr_reg[0] ^ lfsr_reg[2] ^ lfsr_reg[3] ^ lfsr_reg[12];
        // For 13 bit lfsr, generate the feedback by XOR of tap 1st, 3rd, 4th, 13th bits
        // refer from XAPP 052, 1996 issue

        lfsr_next = {lfsr_reg[11:0],lfsr_tap};
        // tap feedback goes at 0 position. other goes shift up
    end

    // output logic
    assign lfsr_out = lfsr_reg[12];

endmodule
```

Fig B. (vii) – module advance.v – used for the implementation

Comments:

The 0s and 1s generated combined are equal to $2^N - 1$ cycle. In my case for N=13 bit, we get 8191 cycles out of which 0s were 4095 and 1s were 4096 for the seed value = **b1000000001101**.

Appendix

The lab F assignment is uploaded along with lfsr.v, lfsr_tb.v, advance.v and advance_tb.v. The submission concludes with attached modules and their respective test-benches.

Also, I have attached the test-bench for sequential circuit made for d-flip flop.