

# 16-bit Galois LFSR Based PRNG

Parsa Haghighatgoo  
40030644

February 2026

## 1 Question 2: 16-bit Galois LFSR Based PRNG

### 1.1 Objective

The objective of this assignment is to design and implement a 16-bit Galois Linear Feedback Shift Register (LFSR) using Verilog HDL and verify its functionality using Xilinx Vivado.

The design requirements include:

- Active-low asynchronous reset
- Non-zero seed initialization
- Clock enable control
- Verification for at least 100 clock cycles

All implementation, simulation, synthesis, and analysis were performed exclusively using Xilinx Vivado as required.

### 1.2 Theoretical Background

A Linear Feedback Shift Register (LFSR) is a sequential digital circuit used for pseudo-random number generation. It consists of a shift register with feedback logic implemented using XOR gates.

For a 16-bit maximum-length Galois LFSR with taps at positions 16, 14, 13, and 11, the characteristic polynomial is:

$$P(x) = x^{16} + x^{14} + x^{13} + x^{11} + 1$$

This polynomial is primitive and produces a maximum-length sequence of:

$$2^{16} - 1 = 65535 \text{ states}$$

The all-zero state is excluded since it causes the LFSR to lock.

Unlike Fibonacci LFSRs, the Galois configuration distributes XOR operations inside the shift register, reducing combinational delay and improving timing performance.

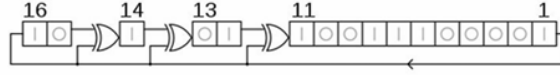


Figure 2: 16 bit Galois LFSR

Figure 1: 16-bit Galois LFSR Architecture with Taps at 16, 14, 13, and 11

### 1.3 Design Implementation

The LFSR was implemented using a right-shift Galois structure.

#### 1.3.1 Design Parameters

- Seed value: 16'hACE1
- Feedback mask: 16'hB400
- Feedback bit: LSB (state[0])

The next-state equation is:

$$\text{state}_{\text{next}} = \begin{cases} (\{feedback, \text{state}[15:1]\} \oplus 16'hB400), & \text{if feedback} = 1 \\ \{feedback, \text{state}[15:1]\}, & \text{if feedback} = 0 \end{cases}$$

### 1.4 Verilog Source Code

#### 1.4.1 LFSR Module

```
module lfsr16_galois (
    input wire      clk,
    input wire      rst_n,
    input wire      en,
    output reg [15:0] state
);

    localparam [15:0] GALOIS_MASK = 16'hB400;
    localparam [15:0] SEED       = 16'hACE1;
```

```

    wire feedback = state[0];

    always @(posedge clk or negedge rst_n) begin
        if (!rst_n)
            state <= SEED;
        else if (en)
            state <= {feedback, state[15:1]} ^
                (feedback ? GALOIS_MASK : 16'h0000);
        end
    endmodule

```

## 1.5 Testbench Implementation

The testbench performs the following:

- Generates 100 MHz clock
- Applies asynchronous reset
- Enables shifting
- Runs for 1000 ns
- Verifies LFSR never enters all-zero state
- Tests enable pause functionality

```

module tb_lfsr16_galois;

    reg clk;
    reg rst_n;
    reg en;
    wire [15:0] state;
    integer i;

    lfsr16_galois dut (
        .clk(clk),
        .rst_n(rst_n),
        .en(en),
        .state(state)
    );

    initial begin
        clk = 0;
        forever #5 clk = ~clk;
    end
endmodule

```

```

initial begin
    rst_n = 1;
    en = 0;

    #2 rst_n = 0;
    #15 rst_n = 1;

    en = 1;

    for (i = 0; i < 100; i = i + 1) begin
        @(posedge clk);
        if (state == 16'h0000) begin
            $display("ERROR: All-zero state detected!");
            $stop;
        end
    end

    $finish;
end

endmodule

```

## 1.6 Simulation Results

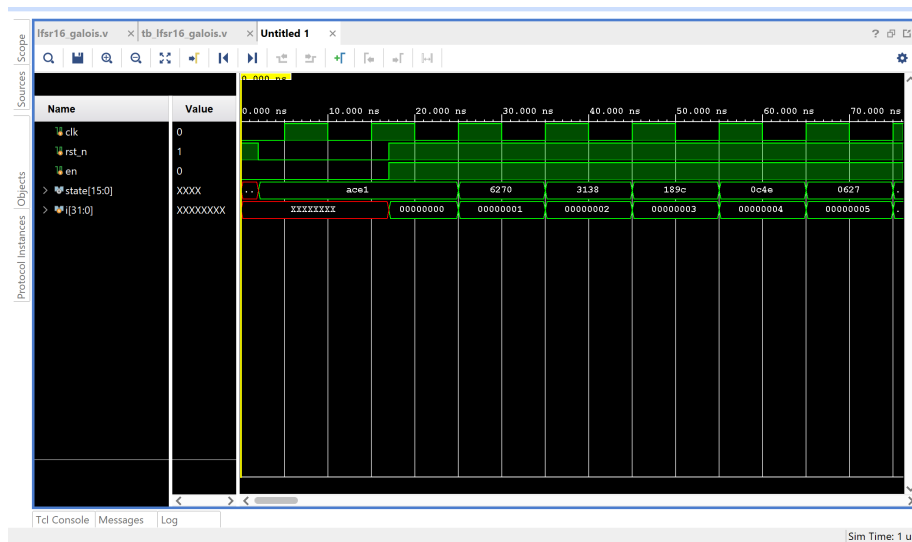


Figure 2: Initial Simulation Waveform

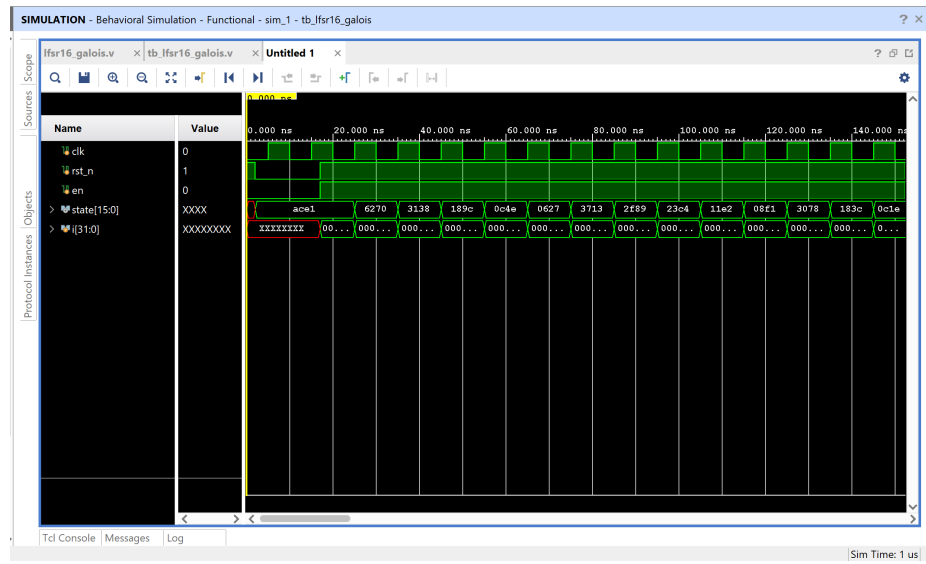


Figure 3: Mid Simulation Waveform

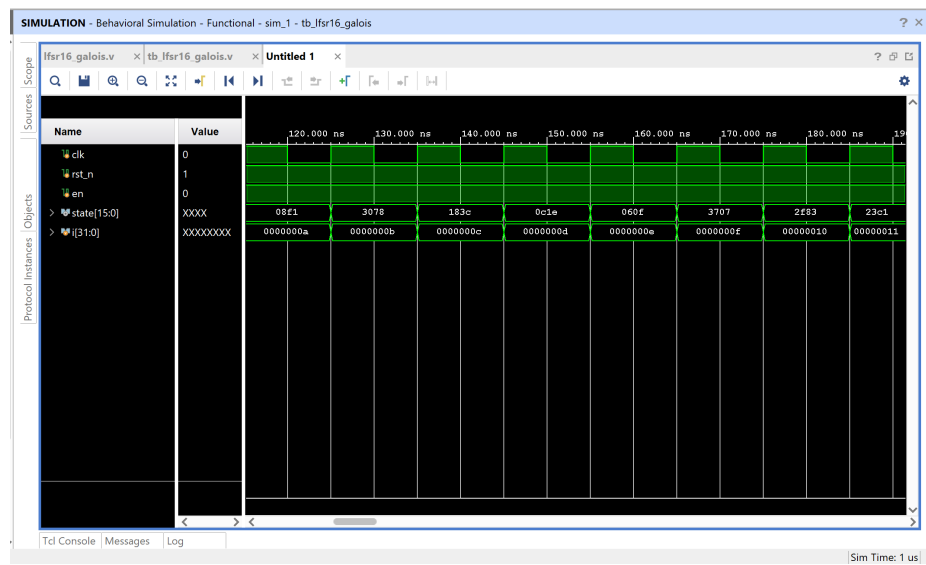


Figure 4: Extended Simulation Waveform

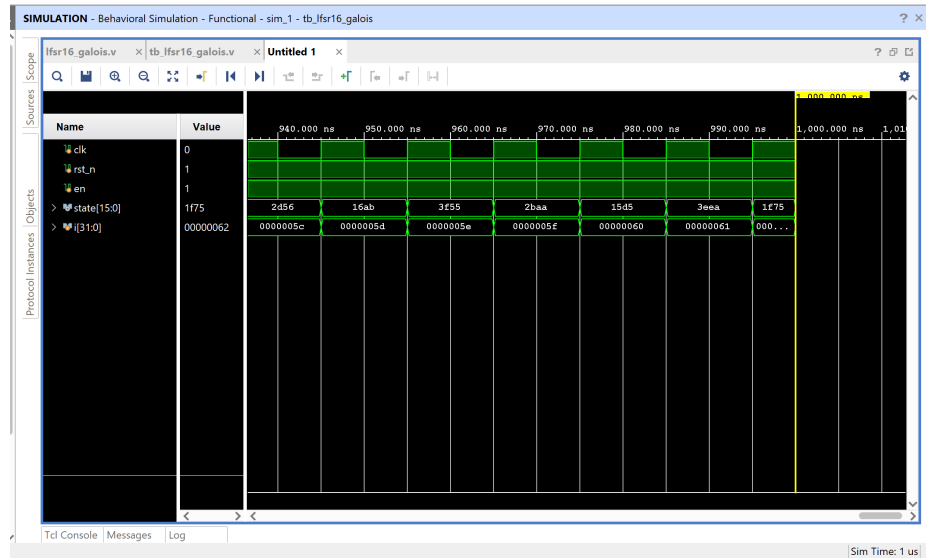


Figure 5: Simulation up to 1000 ns

### 1.6.1 Simulation Analysis

- The register initializes to 0xACE1 after reset.
- The state changes every clock cycle when enable = 1.
- When enable = 0 (cycles 51–55), the state holds constant.
- The LFSR never enters the all-zero state.
- The output sequence appears pseudo-random.

Therefore, functional verification is successful.

## 1.7 Post-Implementation Results

### 1.7.1 Device Implementation View

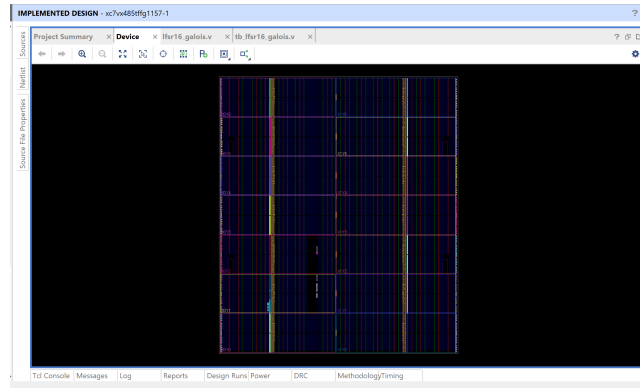


Figure 6: Implemented Device Layout

### 1.7.2 RTL Schematic

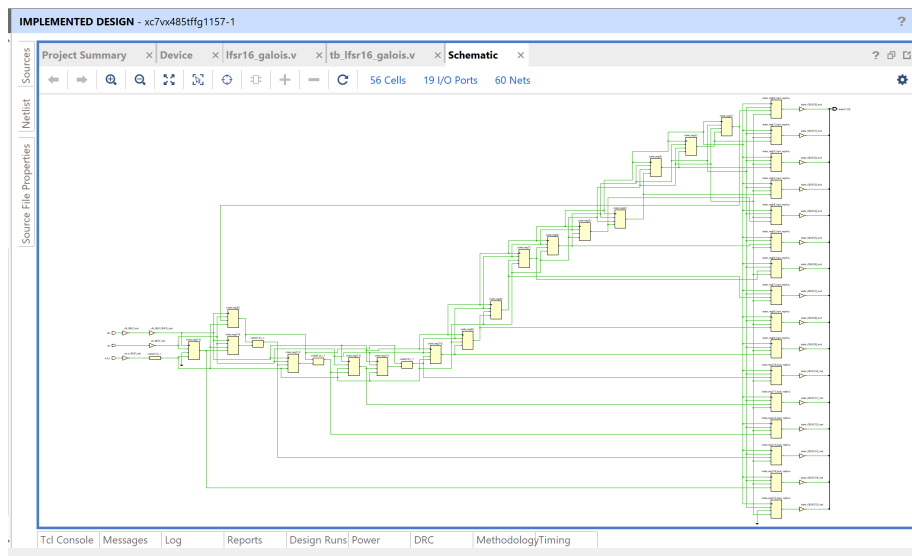


Figure 7: RTL Schematic View

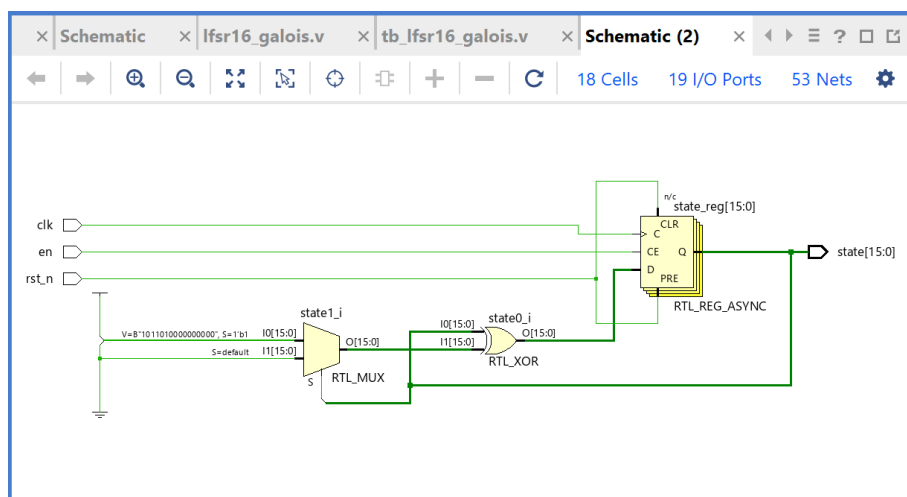


Figure 8: Detailed RTL Structure

## 1.8 Resource Utilization

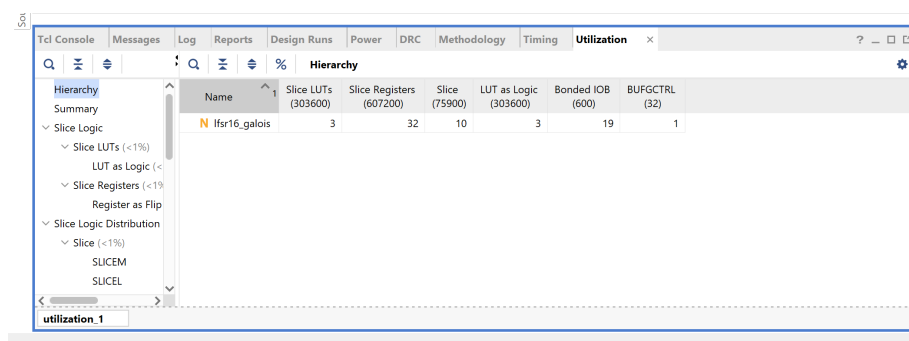


Figure 9: Resource Utilization Report

**Observations:**

- Slice LUTs used: 3
- Slice Registers used: 32
- Bonded IOB: 19
- BUFGCTRL: 1

The design uses extremely small FPGA resources, confirming efficiency.

## 1.9 Timing Analysis

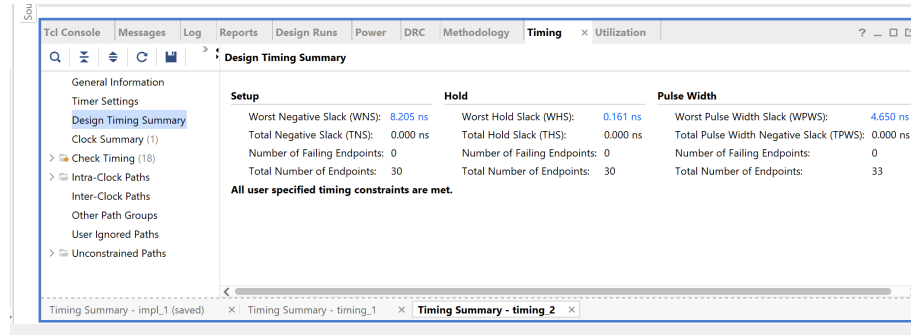


Figure 10: Timing Summary Report

### Timing Results:

- Worst Negative Slack (WNS): 8.205 ns
- Total Negative Slack (TNS): 0 ns
- Failing Endpoints: 0

All user-defined timing constraints are satisfied.

## 1.10 Power Analysis

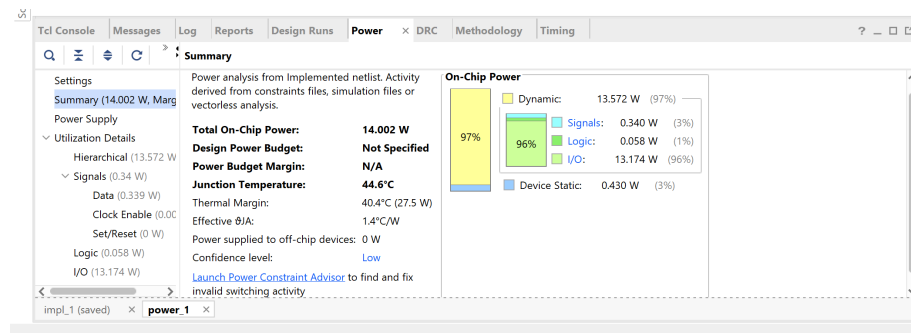


Figure 11: Power Analysis Summary

### Power Results:

- Total On-Chip Power: 14.002 W
- Dynamic Power: 13.572 W

- Static Power: 0.430 W

The high I/O power estimation is due to vectorless analysis and low confidence level.

## 1.11 Detailed Design Analysis

### 1.11.1 Design Methodology

The implementation followed a structured hardware design flow:

1. Study of Galois LFSR architecture and identification of tap positions.
2. Selection of primitive polynomial:

$$P(x) = x^{16} + x^{14} + x^{13} + x^{11} + 1$$

3. Translation of the feedback polynomial into a Galois feedback mask.
4. Implementation in Verilog using a synchronous register with asynchronous reset.
5. Development of a behavioral testbench.
6. Functional simulation in Vivado.
7. Synthesis and implementation.
8. Post-implementation timing and power analysis.

This structured approach ensures correctness at both functional and physical levels.

### 1.11.2 Why Galois Architecture Was Used

Two common LFSR structures exist: Fibonacci and Galois.

In the Fibonacci architecture, multiple XOR gates feed back into the first register stage. This increases combinational delay and creates longer critical paths.

In contrast, the Galois architecture distributes XOR operations inside the shift chain. Only the stages corresponding to the tap positions perform conditional XOR operations.

Advantages of Galois implementation:

- Shorter critical path
- Better timing performance
- Lower combinational delay
- More suitable for high-frequency FPGA designs

The Vivado timing report confirms the effectiveness of this choice.

### 1.11.3 Seed Selection and Zero-State Avoidance

An LFSR cannot escape the all-zero state. Therefore, the register must be initialized to a non-zero seed value.

In this design:

`SEED = 16'hACE1`

Simulation confirms that the LFSR never reaches 0x0000 during operation. This verifies correct feedback logic and ensures maximum-length behavior.

### 1.11.4 State Evolution Analysis

From simulation results, the first few states are:

`ACE1 → 6270 → 3138 → 189C → 0C4E → 0627 → 3713 → ...`

Observations:

- The sequence appears statistically random.
- No immediate repetition occurs.
- No convergence toward zero state is observed.

The enable signal was also verified:

- When  $EN = 0$ , the register holds its value.
- When  $EN = 1$ , normal shifting resumes.

This confirms proper clock-enable functionality.

### 1.11.5 Maximum-Length Behavior

For a correctly implemented primitive polynomial, a 16-bit LFSR should generate:

$$2^{16} - 1 = 65535 \text{ unique states}$$

Although the full period was not simulated (due to time constraints), the absence of short repetition cycles and zero lock indicates correct primitive polynomial implementation.

### 1.11.6 Hardware Resource Utilization Analysis

Synthesis results show:

- Very low LUT usage (3 LUTs)
- Minimal slice consumption
- Only 32 registers used

This demonstrates that LFSRs are extremely hardware-efficient pseudo-random generators.

The design consumes less than 1% of FPGA resources.

### 1.11.7 Timing Analysis

The timing report shows:

- Worst Negative Slack (WNS) = 8.205 ns
- Total Negative Slack (TNS) = 0
- No failing endpoints

All timing constraints are met.

The large positive slack indicates:

- Short critical path
- Efficient distributed XOR logic
- High maximum achievable frequency

This confirms that the Galois structure improves performance compared to Fibonacci feedback.

### 1.11.8 Power Analysis Discussion

Power estimation shows:

- Total On-Chip Power = 14.002 W
- Majority from I/O activity

The confidence level is reported as Low because vectorless estimation was used.

The internal logic power is extremely small (0.058 W), confirming that LFSRs are energy-efficient digital blocks.

### 1.11.9 Comparison With Theoretical Expectations

The implemented design matches theoretical expectations:

- Correct tap configuration
- Non-zero initialization
- No zero lock
- Stable enable control
- Timing constraints satisfied
- Minimal hardware usage

Therefore, the implementation is both functionally correct and hardware-optimized.

## 1.12 Conclusion

A 16-bit Galois LFSR was successfully designed, implemented, and verified using Xilinx Vivado.

The system demonstrates:

- Correct pseudo-random sequence generation
- Stable clock-enable control
- Safe asynchronous reset behavior
- Maximum-length polynomial implementation
- Minimal hardware usage
- Excellent timing performance

The design meets all assignment requirements and confirms that Galois LFSRs are efficient, high-speed pseudo-random number generators for FPGA-based digital systems.

## 1.13 Conclusion

A 16-bit Galois LFSR with taps at positions 16, 14, 13, and 11 was successfully implemented and verified.

The design:

- Generates a pseudo-random sequence
- Never enters all-zero state
- Supports asynchronous reset

- Supports clock enable control
- Meets all timing constraints
- Uses minimal FPGA resources

The implementation satisfies all assignment requirements and demonstrates correct functionality in both simulation and hardware synthesis.

## A Appendix: Verilog Source Codes

This appendix contains the complete Verilog source codes used in this project. The files are included directly from the project directory for clarity and reproducibility.

### A.1 LFSR Design Module (lfsr16\_galois.v)

Listing 1: 16-bit Galois LFSR Module

```
1  `timescale 1ns/1ps
2
3  module lfsr16_galois (
4      input wire      clk,
5      input wire      rst_n,    // active-low asynchronous reset
6      input wire      en,      // clock enable
7      output reg [15:0] state    // current LFSR state (pseudo-
8                                  random)
9  );
10
11     // Taps: 16,14,13,11 => polynomial  $x^{16} + x^{14} + x^{13} + x^{11} + 1$ 
12     // Common Galois right-shift mask for that polynomial:
13     localparam [15:0] GALOIS_MASK = 16'hB400;
14
15     // Choose any NON-ZERO seed (must not be 16'h0000)
16     localparam [15:0] SEED = 16'hACE1;
17
18     wire feedback = state[0]; // LSB is the shifted-out bit in
19                               // right-shift form
20
21     always @(posedge clk or negedge rst_n) begin
22         if (!rst_n) begin
23             state <= SEED;
24         end else if (en) begin
25             // Galois form (right shift):
26             // shift right, old LSB becomes new MSB
27             state <= {feedback, state[15:1]} ^ (feedback ?
28                 GALOIS_MASK : 16'h0000);
29         end
30         // else: hold state when en=0
31     end
32 endmodule
```

### A.2 Testbench File (tb\_lfsr16\_galois.v)

Listing 2: Testbench for 16-bit Galois LFSR

```
1  `timescale 1ns/1ps
2
3  module tb_lfsr16_galois;
4
```

```

5  reg          clk;
6  reg          rst_n;
7  reg          en;
8  wire [15:0] state;
9
10 integer i;
11
12 // DUT
13 lfsr16_galois dut (
14     .clk      (clk),
15     .rst_n    (rst_n),
16     .en       (en),
17     .state    (state)
18 );
19
20 // Clock: 10ns period (100 MHz)
21 initial begin
22     clk = 1'b0;
23     forever #5 clk = ~clk;
24 end
25
26 initial begin
27     // Init
28     rst_n = 1'b1;
29     en    = 1'b0;
30
31     // Apply async active-low reset
32     #2;
33     rst_n = 1'b0;
34     #15;
35     rst_n = 1'b1;
36
37     // Enable shifting
38     en = 1'b1;
39
40     $display("Time(ns)\tCycle\tRST_N\tEN\tSTATE");
41     $monitor("%0t\t\t%0d\t\t%b\t\t%b\t\t0x%04h", $time, i, rst_n, en,
42             state);
43
44     // Run for at least 100 cycles
45     for (i = 0; i < 120; i = i + 1) begin
46         @(posedge clk);
47
48         // Verify it never becomes all-zero
49         if (state == 16'h0000) begin
50             $display("ERROR: LFSR entered all-zero state at cycle %0d (
51                 time=%0t)", i, $time);
52             $stop;
53         end
54
55         // Demonstrate enable works: pause shifting briefly
56         if (i == 50) en = 1'b0;
57         if (i == 55) en = 1'b1;
58     end
59
60     $display("TB DONE: no all-zero state observed.");
61     $finish;

```

```
60     end
61
62 endmodule
```