ECE 440

LFSR

Lab #4 Report

Thomas Schmidt

March 2nd, 2025

Contents

1.1 Introduction . . . 2
1.2 Implementation . . . 2
1.2.1 LFSR Module . . . 2
1.2.2 BRAM Interface Module . . . 2
1.3 Testing and Verification . . . 3
1.3.1 Behaivoral Simulation . . . 5
1.3.2 Console Output . . . 6
1.3.3 Post-synthesis . . . 7
1.4 Conclusion . . . 8

1.1 Introduction

This lab report documents the design, implementation, and testing of a 4-bit Linear Feedback Shift Register (LFSR) interfaced with a Block RAM (BRAM) module. The goal was to generate a pseudo-random sequence using an LFSR and store it in an 8-entry, 4-bit wide BRAM, operating with a 125 MHz clock for the BRAM logic and a 10 MHz clock for the LFSR and write operations.

The LFSR is a key component in digital systems for tasks like random number generation, while the BRAM interface showcases memory management and clock domain crossing.

1.2 Implementation

The design consists of two main modules: `LFSR` and `bram_interface`, described below.

1.2.1 LFSR Module

The `LFSR` module generates a 4-bit pseudo-random sequence with a period of 15 states, using an XOR feedback from bits 3 and 1. It operates on a clock (10 MHz in the testbench) with an asynchronous reset initializing the output to `4'b1001`.

```
`timescale 1ns / 1ps

module LFSR(
    input clk,
    input reset,
    output reg [3:0] lfsr_out
    );
    always @(posedge clk or posedge reset) begin
        if (reset)
            lfsr_out <= 4'b1001;    //initial set to 9
        else begin
            lfsr_out[0] <= lfsr_out[3] ^ lfsr_out[1]; //XOR bit 0 of bit 1&3
            lfsr_out[1] <= lfsr_out[0]; //bit0 -> bit1
            lfsr_out[2] <= lfsr_out[1]; //bit1 -> bit2
            lfsr_out[3] <= lfsr_out[2]; //bit2 -> bit3
        end
    end
endmodule
```

Listing 1.1: LFSR Module

1.2.2 BRAM Interface Module

The `bram_interface` module manages an 8-entry, 4-bit BRAM, storing LFSR data. It uses a 125 MHz clock for logic and a 10 MHz clock for write pulses, implementing a circular buffer with modulo-8 addressing.

```
module bram_interface (
    input clk_125MHz,   //main clk
    input clk_10MHz,    //write pulse clk
    input reset,
    input [3:0] lfsr_data,
    input bram_write_enable,
    output [3:0] bram_out
);
```

```
    (* keep = "true" *) reg [3:0] bram [7:0];   //4bit bram
    (* keep = "true" *) reg [2:0] write_addr;   //3bit addr
    reg prev_clk_10MHz;

    wire write_pulse;
    always @(posedge clk_125MHz or posedge reset) begin
        if (reset)
            prev_clk_10MHz <= 0;    //clear old clk state
        else
            prev_clk_10MHz <= clk_10MHz;    //curr clk vlaue
    end
    assign write_pulse = (clk_10MHz && !prev_clk_10MHz) && bram_write_enable;   //on clk
    rising edge, start write with bram_write_enable

    always @(posedge clk_125MHz or posedge reset) begin
        if (reset) begin
            bram[0] <= 4'b1001;  // Cycle 0 at reset
            write_addr <= 3'b001;  //addr 1, next write
        end else if (write_pulse) begin //rise edge
            bram[write_addr] <= lfsr_data;  //write LFSR to curr addr
            write_addr <= (write_addr + 1) % 8; //wrap arround
        end
    end

    assign bram_out = bram[(write_addr - 1) % 8];   //wrap arround
endmodule
```

Listing 1.2: BRAM Interface Module

1.3 Testing and Verification

The testbench (`bram_tb`) simulates the system, driving the LFSR with a 10 MHz clock (100 ns period) and the BRAM with a 125 MHz clock (8 ns period). It verifies BRAM storage and overwriting behavior over 2 µs.

```
`timescale 1ns / 1ps

module bram_tb;
    reg clk_125MHz;
    reg clk_10MHz;
    reg reset;
    reg bram_write_enable;
    wire [3:0] lfsr_out;
    wire [3:0] bram_out;

    LFSR lfsr_inst (.clk(clk_10MHz), .reset(reset), .lfsr_out(lfsr_out));
    bram_interface uut (.clk_125MHz(clk_125MHz), .clk_10MHz(clk_10MHz), .reset(reset),
                        .lfsr_data(lfsr_out), .bram_write_enable(bram_write_enable), .
    bram_out(bram_out));

    always #4 clk_125MHz = ~clk_125MHz;
    always #50 clk_10MHz = ~clk_10MHz;

    //counter
    reg [3:0] lfsr_cycle_count = 0;
    always @(posedge clk_10MHz or posedge reset) begin
```

```
        if (reset)
            lfsr_cycle_count <= 0;  //reset to 0
        else
            lfsr_cycle_count <= lfsr_cycle_count + 1;   //increment
    end

    integer i;
    initial begin
        clk_125MHz = 0; //start clk low
        clk_10MHz = 0;  // "    "    "
        reset = 1;
        bram_write_enable = 1;

        #50; //50ns
        reset = 0;

        wait (lfsr_cycle_count == 7);    //wait 8 total LFSR cycles
        #50;  //800ns
        $display("Time=%t: BRAM is full", $time);   //print console msg when BRAM is full
        for (i = 0; i < 8; i = i + 1) $display("bram[%d] = %b", i, uut.bram[i]);    //loop
    through 8 entries

        wait (lfsr_cycle_count == 10);  //wait 11 total LFSR cycles after start, 3 more
        #50;    //50ns
        $display("Time=%t: 3 BRAM values replaced", $time); //print console msg when BRAM is
     full
        for (i = 0; i < 8; i = i + 1) $display("bram[%d] = %b", i, uut.bram[i]);    //loop
    through 8 entries

        #850;  //2us
        $finish;
    end

    initial begin
        $monitor("Time=%t, Reset=%b, LFSR=%b, BRAM Out=%b, Cycle Count=%d",
                 $time, reset, lfsr_out, bram_out, lfsr_cycle_count);
    end
endmodule
```

Listing 1.3: Testbench

The testbench checks:

- At 800 ns (8 cycles), BRAM fills with the LFSR sequence starting from `4'b1001`.
- At 1100 ns (11 cycles), 3 entries are overwritten, verifying the circular buffer.

1.3.1 Behaivoral Simulation

Simulation was conducted using Vivado's behavioral simulator, with results captured in Figure 1.1.

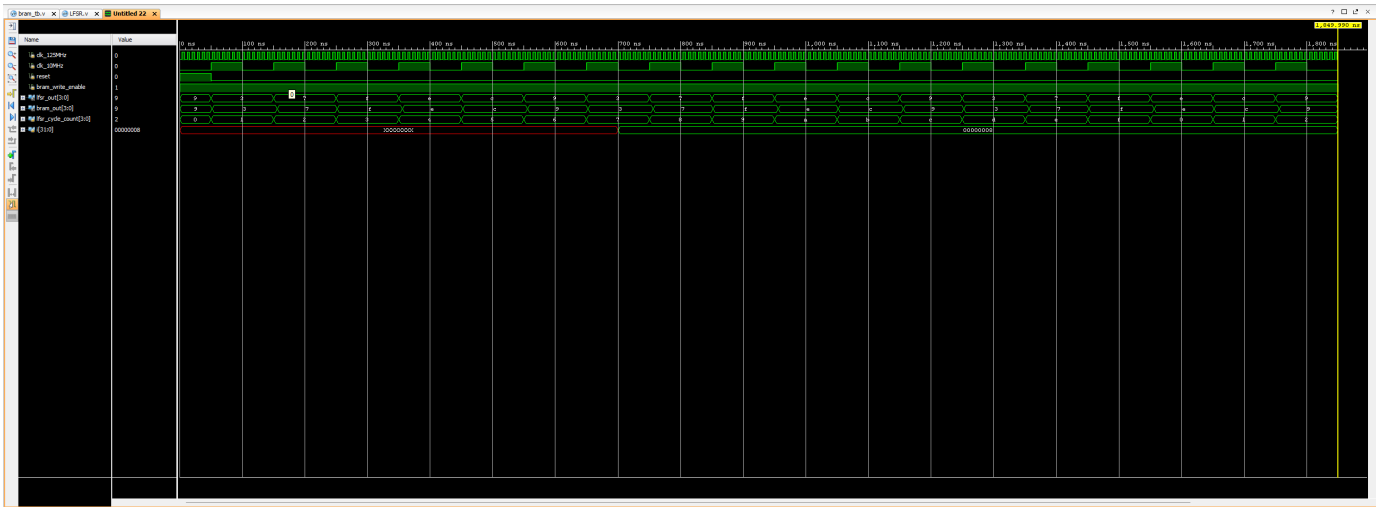

Figure 1.1: Behavioral Simulation Waveform

1.3.2 Console Output

The console output, shown in Figure 1.2, displays the LFSR sequence, BRAM contents, and cycle counts, verifying correct operation.

```
# run 2us
Time=                  0, Reset=1, LFSR=1001, BRAM Out=1001, Cycle Count= 0
Time=              50000, Reset=0, LFSR=0011, BRAM Out=1001, Cycle Count= 1
Time=              52000, Reset=0, LFSR=0011, BRAM Out=0011, Cycle Count= 1
Time=             150000, Reset=0, LFSR=0111, BRAM Out=0011, Cycle Count= 2
Time=             156000, Reset=0, LFSR=0111, BRAM Out=0111, Cycle Count= 2
Time=             250000, Reset=0, LFSR=1111, BRAM Out=0111, Cycle Count= 3
Time=             252000, Reset=0, LFSR=1111, BRAM Out=1111, Cycle Count= 3
Time=             350000, Reset=0, LFSR=1110, BRAM Out=1111, Cycle Count= 4
Time=             356000, Reset=0, LFSR=1110, BRAM Out=1110, Cycle Count= 4
Time=             450000, Reset=0, LFSR=1100, BRAM Out=1110, Cycle Count= 5
Time=             452000, Reset=0, LFSR=1100, BRAM Out=1100, Cycle Count= 5
Time=             550000, Reset=0, LFSR=1001, BRAM Out=1100, Cycle Count= 6
Time=             556000, Reset=0, LFSR=1001, BRAM Out=1001, Cycle Count= 6
Time=             650000, Reset=0, LFSR=0011, BRAM Out=1001, Cycle Count= 7
Time=             652000, Reset=0, LFSR=0011, BRAM Out=0011, Cycle Count= 7
Time=             700000: BRAM is full
bram[          0] = 1001
bram[          1] = 0011
bram[          2] = 0111
bram[          3] = 1111
bram[          4] = 1110
bram[          5] = 1100
bram[          6] = 1001
bram[          7] = 0011
Time=             750000, Reset=0, LFSR=0111, BRAM Out=0011, Cycle Count= 8
Time=             756000, Reset=0, LFSR=0111, BRAM Out=0111, Cycle Count= 8
Time=             850000, Reset=0, LFSR=1111, BRAM Out=0111, Cycle Count= 9
Time=             852000, Reset=0, LFSR=1111, BRAM Out=1111, Cycle Count= 9
Time=             950000, Reset=0, LFSR=1110, BRAM Out=1111, Cycle Count=10
Time=             956000, Reset=0, LFSR=1110, BRAM Out=1110, Cycle Count=10
Time=            1000000: 3 BRAM values replaced
bram[          0] = 0111
bram[          1] = 1111
bram[          2] = 1110
bram[          3] = 1111
bram[          4] = 1110
bram[          5] = 1100
bram[          6] = 1001
bram[          7] = 0011
Time=            1050000, Reset=0, LFSR=1100, BRAM Out=1110, Cycle Count=11
Time=            1052000, Reset=0, LFSR=1100, BRAM Out=1100, Cycle Count=11
Time=            1150000, Reset=0, LFSR=1001, BRAM Out=1100, Cycle Count=12
Time=            1156000, Reset=0, LFSR=1001, BRAM Out=1001, Cycle Count=12
Time=            1250000, Reset=0, LFSR=0011, BRAM Out=1001, Cycle Count=13
Time=            1252000, Reset=0, LFSR=0011, BRAM Out=0011, Cycle Count=13
Time=            1350000, Reset=0, LFSR=0111, BRAM Out=0011, Cycle Count=14
Time=            1356000, Reset=0, LFSR=0111, BRAM Out=0111, Cycle Count=14
Time=            1450000, Reset=0, LFSR=1111, BRAM Out=0111, Cycle Count=15
Time=            1452000, Reset=0, LFSR=1111, BRAM Out=1111, Cycle Count=15
Time=            1550000, Reset=0, LFSR=1110, BRAM Out=1111, Cycle Count= 0
Time=            1556000, Reset=0, LFSR=1110, BRAM Out=1110, Cycle Count= 0
Time=            1650000, Reset=0, LFSR=1100, BRAM Out=1110, Cycle Count= 1
Time=            1652000, Reset=0, LFSR=1100, BRAM Out=1100, Cycle Count= 1
Time=            1750000, Reset=0, LFSR=1001, BRAM Out=1100, Cycle Count= 2
Time=            1756000, Reset=0, LFSR=1001, BRAM Out=1001, Cycle Count= 2
$finish called at time : 1850 ns : File "E:/School/Spring 2025/ECE 440/project_4/project_4.srcs/sim_1/new/bram_tb.v" Line 46
```

Figure 1.2: Behavioral Simulation Console Output

1.3.3 Post-synthesis

Figure 1.3 illustrates the post-synthesis schematic, though full synthesis was hindered by the `initial` block issue in the `bram_interface` module.

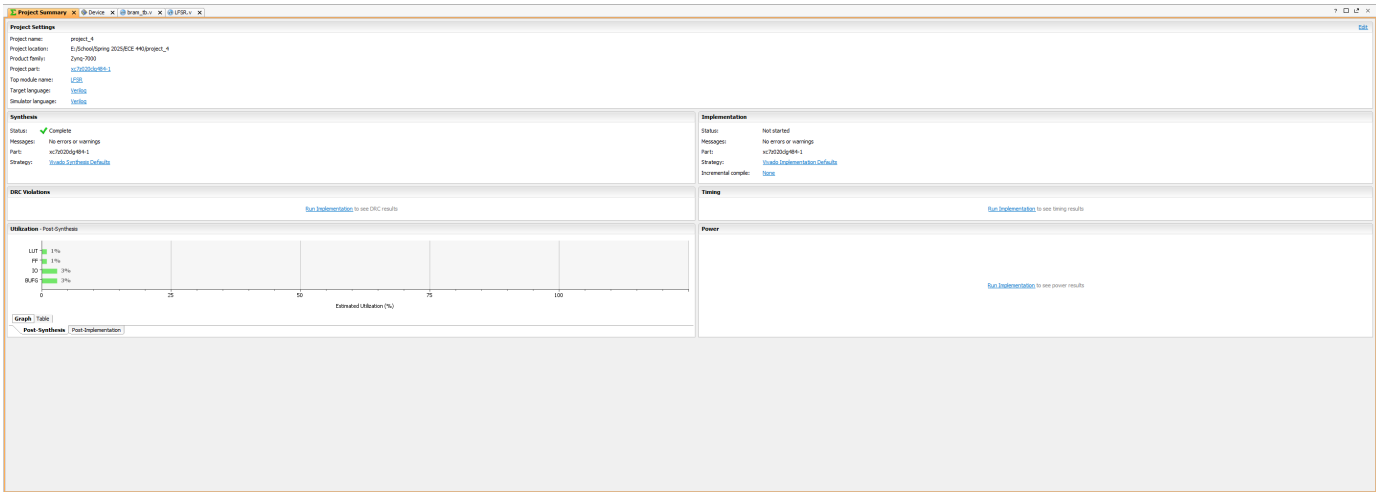

Figure 1.3: Post-Synthesis Schematic

1.4 Conclusion

This lab implemented a 4-bit LFSR and BRAM interface, successfully simulating a pseudo-random sequence stored in a circular buffer. The design highlighted multi-clock domain operation and memory management. However, a synthesis error in the `bram_interface` module due to an `initial` block revealed the need for synthesizable initialization logic, a key takeaway for FPGA design. The testbench validated functionality in simulation, providing a solid foundation for hardware adaptation.

The project enhanced my understanding of Verilog, clock synchronization, and the synthesis process, preparing me for future digital design challenges.