Digital Circuit Lab 1 Report

林軒霆, 蔡丞彥, 侯奕安

September 22, 2025

1 Introduction

In Lab 1, we designed a random number generator using the LFSR technique. We also implemented two additional features: the ability to pause during generation, and a fast mode that doubles the speed of generation.

2 File Structure

All of our verilog code is written in `Top.sv` file, and no `.sv` files were added in this lab. However, we modified `DE2_115.sv` in order to add additional features to two other buttoms.

3 System Architecture

The inputs of our Top module are `i_clk`, `i_rst_n`, `i_start`, `i_key2`, and `i_key3`. The only submodule we used is the LFSR module to generate pseudo random number. It has three input ports, including `clk` and `rst_n`, which are direclty assigned to the external `clk` and `rst_n` signals, and `pause`, which depends on the `state_pause_r` register in the Top module. Finally, there are two output ports: `o_random_out`, which is the random number displayed on the monitor, and `o_sel_led`, which will make a LED on FPGA glow to indicate that the current mode operating is the fast mode.

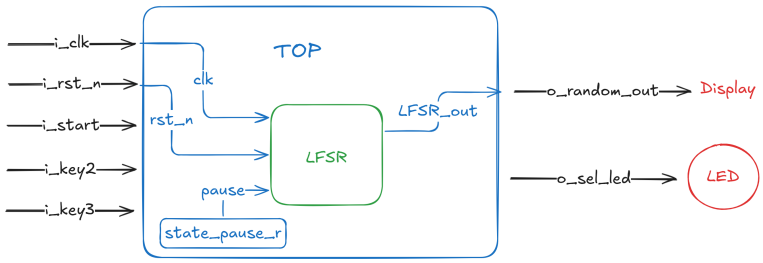

Figure 1: Architecture of Our Design

4 Hardware Scheduling

4.1 Algorithm Workflow

4.1.1 Random Number Generation

We use a 16-bit LFSR `state[15:0]` to generate random numbers. At each clock, we XOR bits 0, 2, 3, and 5 and place the result into bit 15, while the remaining 15 bits are shifted right by one position. We select `state[3:0]` as the LFSR output. After running it 100,000 times in Python, we found that the probabilities of outputs 0—15 are approximately 1/16, and the sequence shows no discernible pattern. Therefore, it can be considered a good random number generator.

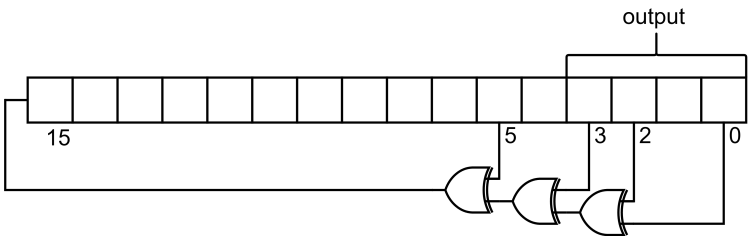

Figure 2: LSFR

After pressing key1 (reset), the LFSR generates a number at each clock. When key0 (start) is pressed, the sequence begins from the number currently being generated. Since the timing of pressing key0 is unpredictable, we can

ensure that each sequence is different.

4.1.2 Speed Control

We defined three parameters: `MAX`, `STOP`, and `INCRE`. When key0 is pressed and the system enters the `S_RUN` state, `counter_r` starts running, incrementing by 1 on each clock cycle. Whenever `counter_r` reaches `max_r`, `counter_r` is reset to zero, and the LFSR is sampled to obtain a random number.

Each time the counter reaches `max_r`, the value of `max_r` is updated to `max_r` + `INCRE`, and `counter_r` is reset to zero to start counting again. When `counter_r` reaches the new `max_r`, the process repeats. The initial value of `max_r` is set to `MAX`, and once `max_r` becomes greater than or equal to `STOP`, the system enters the `S_STOP` state, where the displayed number remains fixed at the last updated value and does not change further, waiting for the next press of key0 to return to the `S_RUN` state. By increasing `max_r` each time, the display duration of each number gradually becomes longer, thereby achieving the effect of reducing the display speed.

4.2 Finite State Machine

We designed a finite state machine with three states: `S_IDLE`, `S_RUN`, and `S_STOP`. `S_IDLE` will switch to `RUN` when the start signal turns to 1, and `S_RUN` will eventually switch to `S_STOP` when there's no current pause signal and `max_r` reaches the `STOP` threshold.

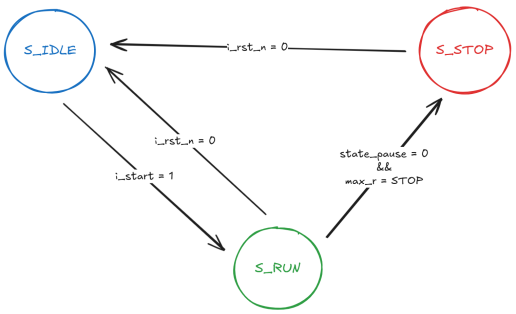

Figure 3: FSM in Our Design

5 Bonus

In `DE2_115.sv`, we applied the same method used for key0 to add key2 and key3, which correspond to `i_key2` and `i_key3` in Top.sv. We also added `LEDR[0]`, which corresponds to `o_sel_led` in Top.sv. These serve as the additional input and output for the BONUS part.

5.1 Pause Mode

We use the newly added key2 as a pause switch. In the `S_RUN` state, pressing key2 sets the pause signal to 1. When pause is 1, the LFSR module is halted—each cycle it simply reassigns its current value to itself—and the counter also stops. At this point, the display output remains fixed at the number shown before the pause was activated. When key2 is pressed again, the pause signal returns to 0, and both the counter and the LFSR module resume operation from the state they were in before the pause.

5.2 Speed Mode

We use key3 as the speed mode toggle button. When key3 is pressed, `state_speed` is updated as `state_speed ^ i_key3`. In other words, pressing it once switches to half-time mode, and pressing it again switches back to normal mode. At the moment of pressing, nothing happens immediately; the change only takes effect the next time key0 is pressed to enter the `S_RUN` state.

When key0 is pressed to enter the `S_RUN` state, if `state_speed` = 1, then `o_sel_led` is set to 1, causing LED0 on the FPGA board to light up, indicating that the system is in half-time mode. In this case, the initial value of `max_r` is set to `MAX_D`, and the subsequent behavior is the same as described in Section 4.1.2, except that `MAX`, `STOP`, and `INCRE` are replaced by `MAX_D`, `STOP_D`, and `INCRE_D`. The parameters with the subscript D are half the value of their non-subscripted counterparts. As a result, all timing intervals become half as long, achieving a double-speed effect.

6 Screen shot of Fitter Summary

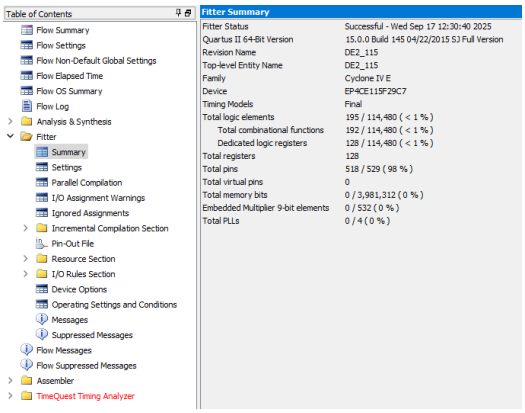

Figure 4: Screen shot of Fitter Summary

7 Screen shot of Timing Analyzer

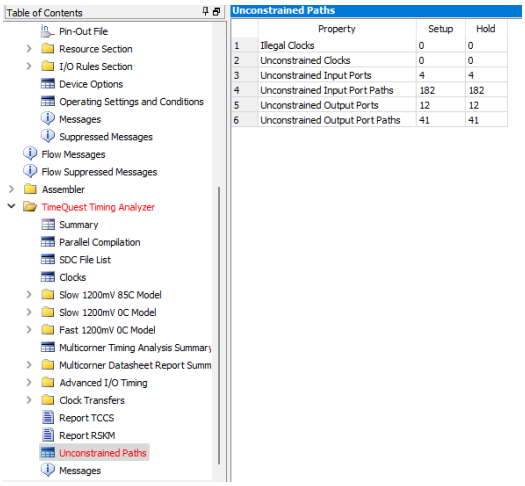

Figure 5: Screen shot of Timing Analyzer

8 Problems Encountered

Since this lab was relatively simple and straightforward, we didn't encounter many problems. However, the first version of our design could only

generate a fixed sequence of random numbers. In other words, after each reset, the subsequent outputs followed the same pattern, identical across all resets. After reviewing the program, we found that the issue was caused by not updating the random seed, which led to generating the same sequence of numbers each time. To address this problem, we decided to let the LFSR continue running even when no output was required. By doing so, the register values varied over time, resulting in genuinely random outputs.