

EE 314 Experiment 4 - Project Preparation

Objectives

The purpose of this experiment is to practice all that you have been taught in this course up to this point and learn how to describe FSMs (Finite State Machines) in Verilog HDL. In this experiment, we will not supply you with any testbenches, and you will be responsible for creating all of your testbenches. Finally, in the experiment sessions, you will use your hardware design knowledge and implement the requested functionality, then verify it on the DE1-SoC Board, equipped with a field programmable gate array (FPGA) using several peripheral units, namely the switch inputs, button inputs, LEDs, 7-segment outputs, and the 50 Mhz clock.

Please take your time and familiarize yourself with everything present in this preliminary work, as you will require what is taught here in your term project.

Although the preliminary work grade is not included in the overall grade of the experiment,

- 1. You need to get at least 50% from the report.**

For this experiment, only the report will be checked, as the testbenches are your responsibility.

1 Preliminary Work

In this experiment, you will design and implement a counter, a clock divider, a shift register, and a linear pseudo-random number generator utilizing Linear Feedback Shift Register (LFSR). You will also describe a simple game FSM (a smaller version of the one you will need in your term project) using Verilog HDL. The concepts introduced here will be used in your term project, so take the time to master them thoroughly.

You will submit two assignments for the preliminary work: A .pdf report and Verilog codes inside a .zip file.

To fulfill the requirements of this laboratory work, the following tasks should be performed. **Note: If necessary, you should only include important parts of the code in the PDF report. The full codes should be submitted separately as a .zip file.**

Your report must include RTL schematic views of the specified components, along with screenshots showing the successful execution of the testbenches.

All designs must be written in Verilog HDL and included in your submission as a ".v" file wherever required.

1.1 Reading Assignment

The [Laboratory Manual](#), where the regulations and other useful information exist, is available on the ODTUClass course page. **Read the [Laboratory Manual](#) thoroughly.** Refer to all Verilog lectures and lecture notes published on ODTUClass **for this experiment.** We also assume you are familiar with Chapters 1-6 of the EE348 course, as this experiment is complementary to that.

If you are unfamiliar with Verilog, Quartus, or cocotb, please refer to the course materials on ODTUClass.

1.2 Design Question Part 1: Counter, Clock dividers, and LFSR (50% Credits)

1.2.1 Counter (10% Credits)

Using Verilog HDL, create a W -bit sequential counter that can increment, decrement, or hold its current value on each positive clock edge depending on the control configuration. Generate the RTL schematic and **add the screenshot of the RTL schematic to your report.(10% Credits)**

1.2.2 Clock Divider (15% Credits)

DE1 SOC FPGA's pre-declared internal clocks are at 50 MHz. Hence, at some point, you will have to divide such clocks in your Verilog codes to obtain frequencies that we can understand with our bare eyes if no exact frequency of operation is desired. Clock dividers are utilized to divide the clock's frequency if the desired hardware operation is slower than the clock signals.

Design and implement a clock divider module that divides a given master clock's frequency by a user-specified factor that will be given as a parameter to the module. You may assume that the maximum division factor is 2^{32} . Note that the duty cycle should remain at 50%. Generate the RTL schematic and **add the screenshot of the RTL schematic to your report.(15% Credits)**

Hint: You can utilize a counter to divide a clock.

1.2.3 LFSR (25% Credits)

A linear-feedback shift register (LFSR) is a shift register whose input bit is determined by a linear function of its current value. It has multiple uses, including implementing a Pseudo-Random Number Generator (PRNG). The linear function is an arrangement of XOR operations on some of its bits. The bits that are used in this operation are called taps. In each iteration shift register is shifted by 1, and the XOR result is given as the serial shift input bit. The first value in the shift register is called the seed. In this part:

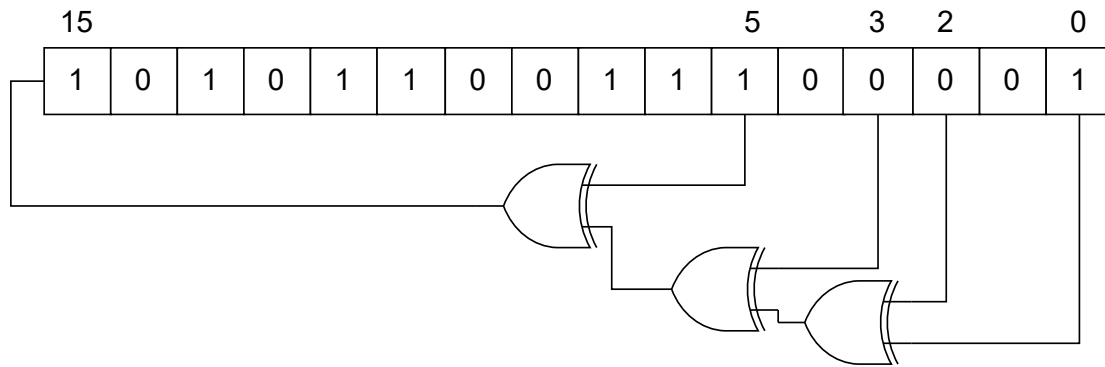


Figure 1: 16-bit LFSR Example

1. Using Verilog HDL, create a W -bit shift register module that can shift its contents left or right by one bit with serial input at each end or load new values in parallel. Generate the RTL schematic and **add the screenshot of the RTL schematic to your report.**(15% Credits)
2. Create a 16-bit LFSR module as shown in Figure 1 using your shift register module, which is a 16-bit LFSR with bits 0, 2, 3, and 5 tapped. The most significant bit is labeled as bit 15, and the shift direction is to the right. Generate the RTL schematic and **add the screenshot of the RTL schematic to your report.**(10% Credits)

1.3 Design Question Part 2: Simple Game Logic using FSMs (50% Credits)

For this part, you will implement an FSM that models a simple game character's movement. The player controlling the character has three inputs: left, right, and attack. You may **assume the player can only assert one of the inputs at any given time**. The FSM describing the character's movement is given in Figure 2 the don't care '*' in the transitions means at any value. Note that only the state transitions are shown on the FSM chart. FSM should also have 3 output flags depending on the state and the inputs, which are:

- Move flag: Should be 1 if the state is S_Left or S_Right, 0 otherwise.
- Directional attack flag: Should be 1 if the attack input is asserted and the state is S_Left or S_Right, 0 otherwise.
- Attack flag: Should be 1 if the state is S_Attack_start or S_Attack_active, 0 otherwise.

1. Generate the RTL schematic of the FSM module and **add the screenshot of the RTL schematic to your report.**(25% Credits)
2. Write a cocotb testbench for the FSM module. **Your testbench can be as you like, but beware that you will demonstrate the FSM on the FPGA board during the laboratory session.** You should check the [Experiment 2 Manual](#) for the cocotb references. The provided testbenches should also be a good reference point. You can include and utilize the "Log_Design" given in [Experiment2Materials](#) function for debugging. **Put your testbench code (without the function declarations for "Log_Design" and "to_hex" if used) in the appendix of your report.** (15% Credits)
3. Verify the module by running your own testbench and **add the screenshot of a successfully passed test to your report.** (10% Credits)

You should create a project ready to be uploaded into the FPGA board using the ping assignments given in Table 1, to save time during the laboratory sessions. You may use zero padding for the state variable if necessary. Also, we highly recommend soft-coding your state values using local parameters, as you will be asked to change them during the experiment sessions.

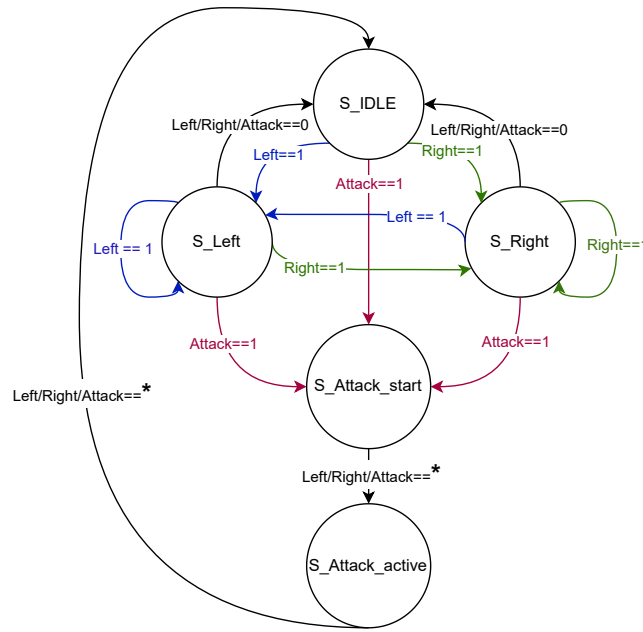


Figure 2: Simple Game Logic FSM

2 Experimental Work

In the experiment session, you will have two parts: a preliminary work demonstration and a design task.

2.1 Preliminary Work Demonstration (30% Credits)

Upload the simple game FSM you have designed to the FPGA and demonstrate the fully working design to your laboratory instructor. **We highly recommend soft-coding your state values using local parameters, as you will be asked to change them during the experiment sessions.**

Table 1: Pin assignment for the preliminary work part

Port	Pin
Left	\sim KEY[3]
Attack	\sim KEY[2]
Right	\sim KEY[1]
clock	KEY[0]
State	HEX0
Move flag	LEDR[0]
Direction attack flag	LEDR[1]
Attack flag	LEDR[2]

2.2 Experiment Session Design Task (70% Credits)

The experiment sessions will require implementations of various designs using FSM coding. The design utilizes, but is not limited to, all modules from the preliminary work. Details will only be given when the experiment sessions start.

A Parts List

DE1-SoC Board