Daniel Celnik
ECE 527L
12/02/2025

Project 2 Report

In this project, we explored linear feedback shift registers (LFSRs) and implemented them to create target code and a testbench that compares the outputs to a file containing expected values. We then made a module with the same inputs and outputs that increments like a binary counter instead of employing LFSR feedback, and a corresponding testbench that iterates thousands of times to fulfill an exhaustive functional testing of all 8-11 bits in the output register. Finally, we ran synthesis on both designs to compare their area and timing analyses and provide insight on different implementations and their cost regarding hardware resources.

The LFSR target code is implemented using a "mode" input that selects between 8, 9, 10, and 11-bit output values. A "start" input initializes the LFSR with a starting seed of 'b1. A "stop" input halts the state progression. The design additionally includes an active low asynchronous reset, a clock, and an 11-bit output. The starting and stopping functionality is implemented using a register called "run" that is set to 1 when the module is started, not stopped, and not reset. The LFSR changes state so long as the stop and reset signals are not asserted.

The LFSR testbench is implemented via $readmemb to import the expected data values. A register called "index" is used to keep track of the current index of the expected data values. If the current output of the target code does not equal the value of the expected data value memory, the terminal outputs a Mismatch message alerting the developer.

The counter target code's bit selection functionality is implemented in a similar fashion to the LFSR, aside from the incrementation when determining the next state. The counter testbench code makes thousands of iterations to ensure that all 8 bits of the mode "00" counter are used, that all 9 bits of the mode "01" counter are used, etc. Unlike the LFSR, there are no expected values to import and compare against.

Regarding the area results, the LFSR netlist has a total area of 754.25, while the counter netlist has a total area 1440.79. This discrepancy occurs since a LFSR is implemented using exclusive OR gates as taps while the counter requires many more registers and adder modules to allow for the conditional incrementation of a variable 8 to 11-bit binary number.

For the timing results, the LFSR netlist has a slack of -4.04 while the counter has a slack of -0.9. In the LFSR target code, the "mode" input is used in a mux to determine the size of the LFSR, which is used in another case statement with a function to determine the tap feedback. There are multiple levels of combinational logic including adders, XOR trees, and MUXs wired in series to implement the LFSR, whereas the counter design is implemented solely using adders and registers. Regardless, both designs violated their respective slack requirements.

In this project, we learned how to implement LFSRs, how to design slightly complex control signal systems, and how to use a scoreboard for testbench expected results checking. All of these skills are paramount for an ASIC developer looking to contribute their skills.