Simulation

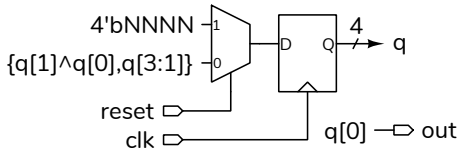
Revision 3: Initial value should be last non-zero *digit instead of just the last digit. Also fixed block diagram.*

Introduction

In this lab you will design a 4-bit linear-feedback shift register circuit that generates a maximal-length pseudo-random bit sequence. LFSRs are used to generate pseudo-random signals and as low-complexity/high-speed counters. You will test your design by simulating it.

Requirements

The schematic of a 4-bit LFSR is:



Where NNNN is the last (rightmost) non-zero digit of your BCIT ID in binary. For example, if your BCIT ID were A01234560 then the register would be reset to **`4'b0110`**.

The circuit has 1-bit **`clk`** (clock) and **`reset`** inputs and a 1-bit **`out`** output. This output is the rightmost (least-significant) bit of the shift register.

Your test vectors must have two values: the input value of **`reset`** and the expected output value of **`out`**.

There should be 18 test vectors. Only the first should assert **`reset`**. Each test vector except the last should include the expected value of **`out`** (the least-significant bit of the shift register). The last one should include the logical complement of the expected value so that your testbench will detect an error.

Procedure

Write a module that implements the LFSR module described above and a testbench that applies test vectors read from a file. You should be able to use the example in the (revised) lecture notes with some small changes. Remember to follow the course coding guidelines, including adding the appropriate comment at the beginning of each file.

Your testbench should display the value of the LFSR register (although it is *not* an output of your module), the value of **`out`**, and the expected value of **`out`** for each test vector. In addition, it should print an error message if there are mismatches between the expected and actual outputs. See below for an example of the expected output.

You can compute the test vectors by hand or use a spreadsheet to compute the values.

Follow the procedure in Software Installation and Use document and the video on the course web site to create a simulation project, add the file(s) with your modules to the project and compile them. Add the **`reset`**, **`clock`**, and **`out`** signals to the Wave window. Run the simulation. The Transcript window should show any messages generated by the testbench and the Wave window should show the signal waveforms.

Report

Submit a PDF file to the appropriate Assignment folder that includes the following:

- listing(s) of your DUT and testbench modules.
- a screen capture of the simulation waveforms similar to that in Figure 1 or Figure 2.
- a screen capture of the Transcript window showing the messages generated by running the simulation. For example:

```
# run -all
# DUT q = 0110
# DUT q = 1011
# DUT q = 0101
# DUT q = 1010
# DUT q = 1101
# DUT q = 1110
# DUT q = 1111
# DUT q = 0111
# DUT q = 0011
# DUT q = 0001
# DUT q = 1000
```

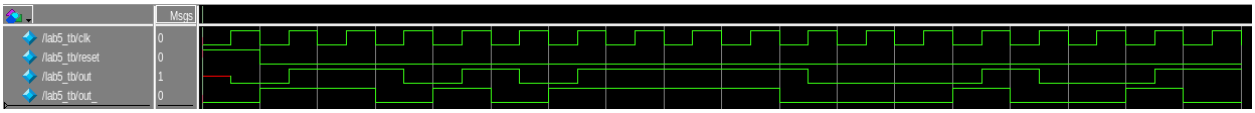


Figure 1: Simulation waveforms (Modelsim screen capture).

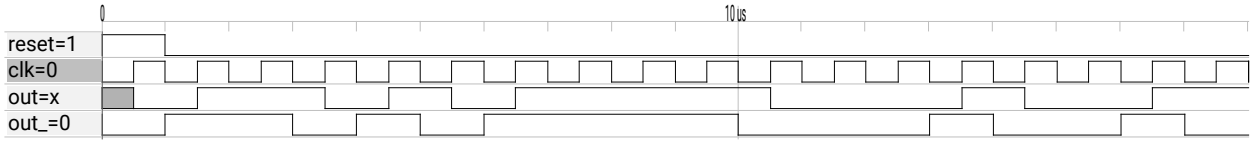


Figure 2: Simulation waveforms (GTKWave).

```
# DUT q = 0100
# DUT q = 0010
# DUT q = 1001
# DUT q = 1100
# DUT q = 0110
# DUT q = 1011
# DUT q = 0101
# Error: out=1 but expecting out_=0
# ** Note: $stop    : lab5_tb.sv(26)
#    Time: 18 us  Iteration: 2  Instance: /lab5_tb
# Break in Module lab5_tb at lab5_tb.sv line 26
# Stopped at lab5_tb.sv line 26
#  exit
# End time: 15:44:39 on Oct 24,2023, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
```