

## 1 Background

Implement the linear feedback shift register shown on [https://en.wikipedia.org/wiki/Linear\\_feedback\\_shift\\_register](https://en.wikipedia.org/wiki/Linear_feedback_shift_register).

The core 16-bit register must be implemented with a 16-bit signal. The the shift should be implemented using a concatenation operator to right-shift with the left-most bit loaded from the output of the XOR gates. Verify that linear-feedback shift-register produces all possible states of a 16-bit register excluding the zero state before repeating its sequence.

To do this, use a Verilog Test Bench to save the output to a file every clock cycle for at least two full cycles of the pattern. Write C/Python/Java code to verify that the pattern repeats identically at least twice and that no number is repeated or missed during one cycle of the pattern. You should submit all code and output files.

## 2 Implementation

The maximal linear shift register of 16 bits were computed, with the feedback polynomial  $x^{16} + x^{15} + x^{13} + x^4 + 1$  (1) to generate 65535 terms. The test bench allowed at least 2 cycles of shifting to fully complete until termination.

The module implementation along with its test bench can be found in the 'scripts' directory. A sample of the waveform generated is provided:

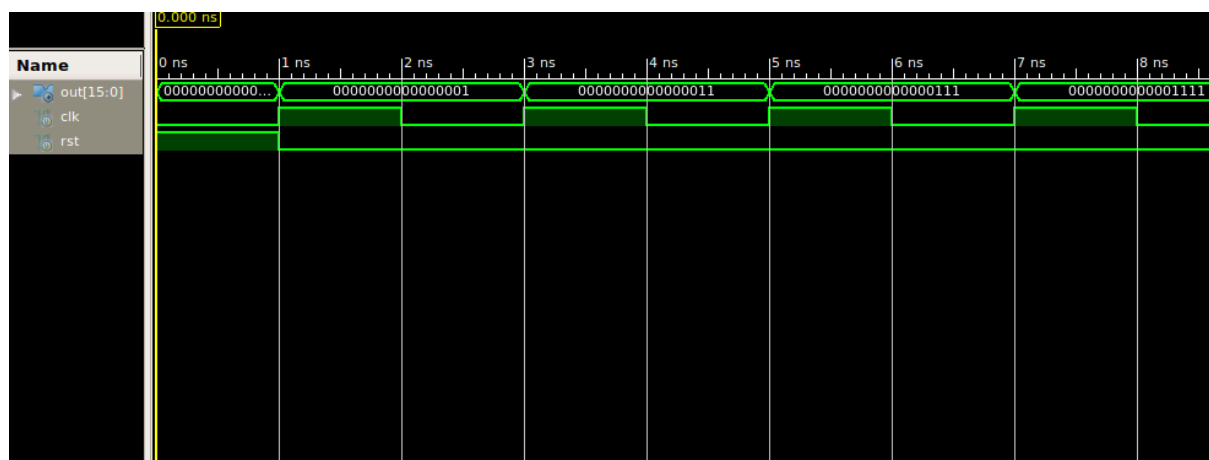


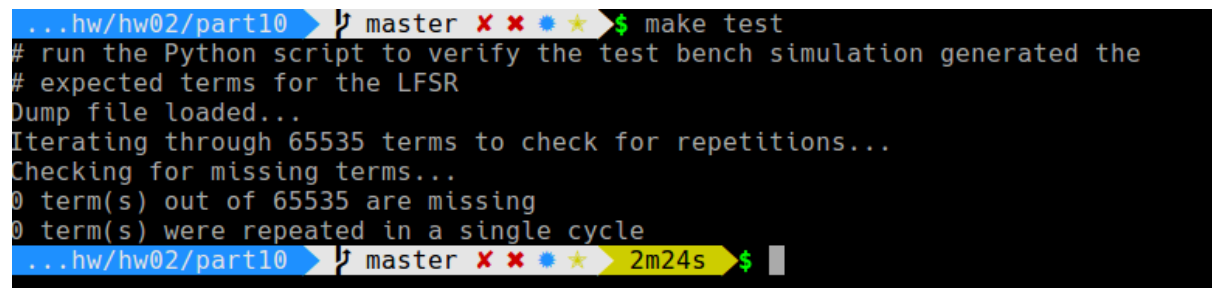
Figure 1: Waveform Generated from Part 10 Test Bench

### 3 Testing Validity

The terms generated by the linear shift register were dumped to a file for further analysis. A Python script have been provided along with a Makefile for convenience. The Makefile can be used for:

- dumping the outputs to the file with `make compile`
- testing with `make test`
- cleaning up with `make clean`
- handle generation of the LaTeX reports

The Python script reads the output file and iterates through each line to make sure there are no more repetitions of terms than expected. After that, it iterates through the binary representation of all decimal integers from 1 to  $2^{16}-1$ , and checks if they have been generated in each cycles.



```
...hw/hw02/part10 > master [X][X][*] $ make test
# run the Python script to verify the test bench simulation generated the
# expected terms for the LFSR
Dump file loaded...
Iterating through 65535 terms to check for repetitions...
Checking for missing terms...
0 term(s) out of 65535 are missing
0 term(s) were repeated in a single cycle
...hw/hw02/part10 > master [X][X][*] 2m24s $
```

Figure 2: Output of The Analysis of the Terms Generated by the LSFR

### References

- (1) <http://simplefpga.blogspot.com/2013/02/random-number-generator-in-verilog-fpga.html>