ECE459/559 Secure & Trustworthy Computer Hardware Design, Spring 2017
Lab 2

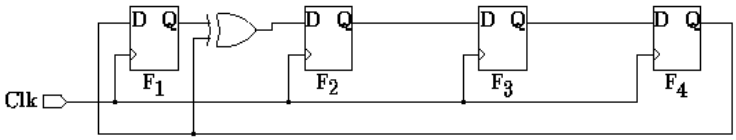
For all labs:

- Use a Digilent Nexys-4 board with Xilinx Vivado 2014.4
  NOTE: Vivado 2014.4 WebPACK can be installed for free on your own laptop or PC. Also, Vivado 2014.4 is installed on the PCs in the Digital Systems Laboratory (MK224).

- Follow the general design and implementation process using Xilinx Vivado WebPACK:
  1. *Capture* the design using VHDL code.
  2. *Simulation* is run using a VHDL test bench to analyze, verify and debug the functionality of VHDL design code.
  3. *Synthesis* maps the design onto the CLBs and other functional blocks that exist on the Artix-7 FPGA (the "FPGA architecture primitives") .
  4. *Constraints Wizard* (post-Synthesis) provides an interface to update the XDC constraints for mapping FPGA I/O pins to top-level ports in your design. These constraints can also be updated by manually editing the XDC constraints file (*.xdc).
  5. *Edit Timing Constraints* (post-Synthesis) provides an interface to update the XDC timing constraints, including constraining the clock period. These constraints can also be updated by manually editing the XDC constraints file (*.xdc).
  6. *Implementation* (place & route) figures out where on the FPGA each primitive block goes and how to connect them. Specifically, the CLBs to be used, what logic each must implement, and how these CLBs must be connected on the Artix-7 FPGA.
  7. *Generate Bitstream* converts the implemented design to a programming file (.bit) to be uploaded to the Nexys4 board.

- Write a short (2 page minimum) report detailing your result and **include signature from TA or professor showing successful demonstration of a working design**. Signature page can be separate individual page (usually part of a title page) of lab report allowing more space for the remainder of the write-up.

***Linear Feedback Shift Registers (LFSR)*:**



Linear feedback shift registers, such as that shown above, are n-bit counters exhibiting pseudorandom behavior using very little circuitry (low overhead). They have several applications in security, specifically due to their ability to act as pseudorandom number generators (PRNG).

Consider the operation of the LFSR above. When all of the flops are 0, nothing happens. Every Q output is 0, the feedback path from F_4 to F_1 is also 0, and the output of the XOR is 0, so the LFSR stays in the 0 state forever. When a logic 1 is introduced at F_1, the circuit counts through $2^4 - 1 = 15$ different non-zero bit patterns, as shown below:

LFSR Sequence
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
1 1 0 0
0 1 1 0
0 0 1 1
1 1 0 1
1 0 1 0
0 1 0 1
1 1 1 0
1 1 1 1
1 0 1 1
1 0 0 1
1 0 0 0

A similar circuit can be constructed with any number of flip-flops, though more XOR gates are also needed. A k-bit LFSR (k flip-flops) can be constructed by finding the corresponding primitive polynomial for the flop. This primitive polynomial is written in the form $x^k + \ldots + 1$, where each term x^n corresponds to a connection between flip-flop F_n and F_{n+1}. There are two special cases: $x^0 = 1$ and x^k, which simply represents a connection from the Q output of flip-flop F_k to the D input of flip-flop F_1. All other x^n connections correspond to an XOR between flip-flop F_n and F_{n+1}. For example, for the 4-bit LFSR shown above:

Primitive polynomial: $x^4 + x + 1$
This leads to 1) a feedback connection from F_4 to F_1 and 2) an XOR between F_1 and F_2

To build an 8-bit LFSR, the primitive polynomial is $x^8 + x^4 + x^3 + x^2 + 1$, indicating feedback from F_8 to F_1 and XOR gates between F_2 and F_3, F_3 and F_4, and F_4 and F_5. Other XOR input is F_8.

***Task 1*: Draw a logic circuit schematic for an 8-bit LFSR.** The schematic should look similar to that shown above for the 4-bit LFSR.

*** Deliverable: Clearly drawn 8-bit LFSR schematic to be included in report.**

***Task 2*:** Implement an 8-bit LFSR using VHDL. You should implement to the following specifications to successfully complete this task:

Use LEDs LD0 through LD7 to display the state of the LFSR flip-flops (F_1 to F_8)
Use the center push button 'BTNC' as the clock to sequence through the LFSR
Use the top push button 'BTNU' as a reset for the LFSR flip-flops
Use switches SW0 through SW7 as inputs to directly feed data into LFSR flip-flops
Use switch SW15 to control what drives the LFSR flops:
SW15 = 1 – D inputs of flops are driven by switches SW0 through SW7
SW15 = 0 – D inputs of flops are connected within LFSR according to design

For this lab, you will need to modify the constraints file "Nexys4_Master.xdc" to connect the ports from your VHDL code to the I/O pins on the FPGA. For example, to connect the physical switch SW0 to a signal switch0 from your VHDL, you would enter the following:

```
set_property PACKAGE_PIN U9 [get_ports switch0]
	set_property IOSTANDARD LVCMOS33 [get_ports switch0]
```

Since we are using a push button as an artificial clock, you may also need to enter the following line in your "Nexys4_Master.xdc" constraint file (assuming you call clock "clock"):

```
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clock]
```

You can use components from Lab1 for flip-flops and XOR gates or you can build your own from scratch.

*** Deliverable: VHDL code for your 8-bit LFSR to be included in report as an appendix.**

***Task 3*:** Using the Tutorial and Lab1 test benches as examples, construct your own VHDL simulation test bench to verify your LFSR design.

Simulate the design using your test bench, stepping through several sequences of the LFSR.

*** Deliverable: Take a snapshot of your simulated result, save it as an image and include in your lab report.**

***Task 4*:** Program your FPGA (follow steps in Vivado tutorial) to implement your complete LFSR design. Once programmed, initialize the LFSR using the switches then step through several sequences using the clock.

*** Deliverable: Write down at least 16 of the sequences you observe, starting from "0000 0001" and include in your report.**