University of California at Berkeley
College of Engineering
Department of Electrical Engineering and Computer Sciences

CS 150
Fall 1996

J. Wawrzynek
N. Weaver

Lab 5
Shift Registers and Counters

1 Objective

Counters and shift registers are commonly-used finite state machines. In this lab, you will build some and use the Xilinx board to observe their operation.

2 Prelab

Read through this, enter the circuit described in Section 4, and answer the checkoff sheet questions.

3 Linear Feedback Shift Registers

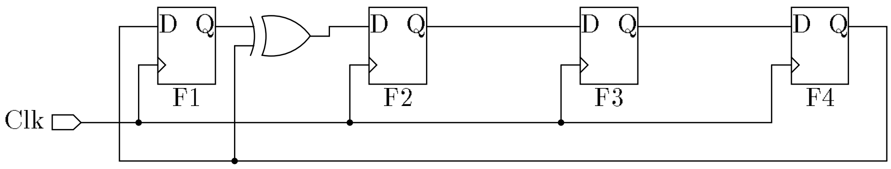

Figure 1: A four-bit linear feedback shift register

Linear feedback shift registers, such as the one in Figure 1, are n-bit counters exhibiting pseudo-random behavior using little circuitry. They have many applications, including computer graphics and pseudorandom number generation. You will use an LFSR in your project to generate a random timeout and to perform error checking.

Consider the operation of the LFSR shown in Figure 1. When the latches are all zero, it does nothing. Every Q output is zero, the feedback path is zero, and the output of the XOR is zero, so the LFSR stays in this state.

When a 1 is introduced, the circuit counts through $2^4 - 1 = 15$ different non-zero bit patterns (at right).

You can build a similar circuit with any number of flip-flops, although you may need more XORs. The table of primitive polynomials on Page 5 can be used to design these circuits. Section 7 discusses *why* these circuits do what they do.

Q1	Q2	Q3	Q4
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
0	1	1	1
1	1	1	1
1	0	1	1
1	0	0	1
1	0	0	0

3.1 Using the Table to Build an LFSR

To build a k-bit LFSR, number the flip-flops starting with 1 on the left. The feedback path comes from the Q output of the rightmost flip-flop k.

Find the primitive polynomial of the form $x^k + \cdots + 1$ in the table on Page 5. Its $x^0 = 1$ term corresponds to connecting the feedback directly to the D input of flip-flop 1. Each term of the form x^n corresponds to connecting an XOR between flip-flop n and $n + 1$.

To build the LFSR of Figure 1, I used the primitive polynomial $x^4 + x + 1$:

$$\underbrace{x^4}_{\text{F4's Q Output}} \quad + \quad \underbrace{x}_{\text{XOR between F1 and F2}} \quad + \quad \underbrace{1}_{\text{F1's D input}}$$

To build an eight-bit LFSR, use the primitive polynomial $x^8 + x^4 + x^3 + x^2 + 1$ and connect XORs between F2 and F3, F3 and F4, and F4 and F5.

4 What to Build

Build a configurable linear-feedback shift register on the Xilinx Design Demonstration Board. The DIP switches will control where the XORs are placed, the eight LEDs will display either the contents of the shift register or an eight-bit binary counter, the RESET button will reset the counter, and the SPARE button will switch between the two display modes.

We must clock the shift register slowly to see its operation. The 1 MHz clock from the XChecker cable is much too fast, so we will divide it by $2^{15} = 32768$ to obtain a 30.5 Hz clock. We will use this 30.5 Hz signal as a clock enable input to the other flip-flops in the design, rather than using a separate clock signal. A single-clock design is more robust than one with multiple clocks.

Our shift register will consist of eight flip-flops. Between each flip-flop will be a small circuit: an XOR enabled by a switch. In one position, the circuit will emulate an XOR connected as in Figure 1. In the other position, the circuit will emulate a wire.

Since flip-flops in the Xilinx start in the reset state, and LFSRs do nothing in this all-zero state, building Figure 1 exactly will not work. The easiest way around this is to invert both the input and output of a flip flop. This will make the flip flop behave normally, but when reset, the inverters will insure that it will output a 1 instead of a 0.

We will visually compare the operation of the shift register to an eight-bit binary counter clearable with the RESET button. In one display mode, seven of the eight LEDs will display the most significant bits of this counter, and the eighth LED will indicate the flip-flops are in the all-zeros state.

In the other display mode, the LEDs will display the contents of the shift register. Use two-input multiplexers controlled by the SPARE button to steer one of the two modes to the LEDs.

As with the lock from Lab 3, the clock will be on Pin 8, the RESET button will be on Pin 56, and the SPARE button will be on Pin 68. Remember to use a PULLUP with the SPARE button, as in the Lab 3 schematic. The DEBOUNCE circuit is unnecessary. To use the XChecker debugging facilities, include the READBACK circuit.

1. Build a divide-by-32768 counter with a `CC16CE`. AND together the least significant fifteen bits with four `AND4`s and an `AND3` to produce the clock enable signal for the other flip-flops in the design.

2. Use a `CC8CE` eight-bit counter with its `CE` input connected to the clock enable signal from the `CC16CE`'s outputs, its `CLR` connected to the RESET button. Think about the polarity of this signal.

3. Build the shift register using eight `FDCE`s with their `CE` inputs connected to the clock enable signal. Put an XOR with a multiplexer, and control the multiplexer with the DIP switches, whose pins are listed in Table 1, to select between the output of the previous flip flop (or ground for the left most FF) and the previous flip flop XOR the feedback.

4. Use eight two-input multiplexers (`M2_1`s) controlled by the SPARE button to select the two display modes for the LEDs. Think about what you want the default (SPARE released) to be. The LED pin numbers are listed in Table 1.

   - In one mode, display the Q outputs of the eight shift register flip-flops.
   - In the other mode, display the seven most significant bits of the `CC8CE` and have the last display be a circuit which is high only when the rightmost flip flop is the ONLY flip-flop high.

	left							right
switch	19	20	23	24	25	26	27	28
LED	61	62	65	66	57	58	59	60

Table 1: DIP switch and LED pins

5 What to Do

1. Make sure the eight-bit counter is working. Make the LEDs display the counter and verify the counter's LEDs are flashing at half the speed of their neighbor. Try resetting the counter with the RESET button.

2. Observe the shift register contents on the LEDs. Set the switches so that all bits become zero (all switches in the same direction). Flip the leftmost switch. The display should change to a sweeping all-on to all-off pattern.

3. Use the table on Page 5 to set the switches to make a seven-bit, period 127 LFSR. Verify this period by waiting for the eighth LED to flash briefly, pressing the RESET button, and noting that the next flash happens in about the same time it takes the second-slowest counter bit to pulse on and off. Show a TA.

4. Use the table to set the switches to make an eight-bit, period 255 LFSR. Verify the period as before; use instead the slowest counter bit. Show a TA.

5. Change one of the other switches so that you still have an eight-bit LFSR, but with a different period. Time it as before. Show a TA.

6 Windows 95 transition

Unfortunatly, although Windows NT is a more robust environment, not all software we need to do the project work properly in the NT environment. So we are going to slowly change the lab machines over to Windows 95. This will result in a few changes. Firstly, you can no longer change file permissions, but we will keep a couple machine running NT for this purpose. The xilinx tools (xmake, xchecker) should be run under the MSDOS prompt, available from the start menu. And you no longer need to install the Viewlogic software. And to log out, you chose shutdown from the start menu.

7 Galois Fields

This circuit behaves as it does because it is performing multiplication on a *field*, a set with two operations defined on it: "addition" and "multiplication." The set is closed under these operations, the associative and distributive laws hold, there is both an additive and multiplicative identity element, there is an additive inverse for every element, and a multiplicative inverse for every non-zero element.

Familiar infinite fields include the set of rational numbers, the set of real numbers, and the set of complex numbers. The set of integers is *not* a field, since most integers do not have a integer multiplicative inverse.

Finite fields are called *Galois* (gal-WAH) fields. Binary numbers form a simple Galois field with two elements called $GF(2)$. The XOR function is "addition," and the AND function is "multiplication."

Consider polynomials whose coefficients come from $GF(2)$, that is, each term of the form x^n is either present or absent. For example, 0, 1, x, x^2, and $x^7 + x^6 + 1$ are all $GF(2)$-coefficient polynomials.

Adding these polynomials is easy. "Add" (XOR) each element individually—there is no carry:

$$\begin{array}{rl} & x^4 + x^3 + x + 1 \\ + & x^4 + x^2 + x \\ \hline & x^3 + x^2 + 1 \end{array}$$

Multiplying these polynomials is also easy. Multiplying by a monomial of the form x^n is like shifting to the left:

$$\begin{array}{rr} & x^2 + x + 1 \\ \times & x + 1 \\ \hline & x^2 + x + 1 \\ & x^3 + x^2 + x \\ \hline & x^3 + 1 \end{array}$$

If we use this addition and take the results of this multiplication modulo[1] a prime polynomial[2] $p(x)$, these polynomials form a Galois field. For any degree, there is at least one prime polynomial. By using it, we can form $GF(2^n)$, the Galois field with 2^n elements, for any positive integer n.

[1] This ensures the leading coefficient of the result is smaller than the leading coefficient of $p(x)$. Do this by subtracting a (polynomial) multiple of $p(x)$ from the result. Often this multiple is 1, corresponding to XORing the result with $p(x)$.

[2] That is, a polynomial that cannot be written as the product of two non-trivial polynomials $q(x)r(x)$.

Every Galois field has a primitive element: an element α such that every non-zero element can be expressed as a power of α. So if we know a primitive field element and can raise it to any power, we can obtain all non-zero elements of a field.

Certain choices of $p(x)$, the prime generating polynomial for a field (i.e., multiplication is taken modulo this polynomial), make the simple polynomial x a primitive element. It turns out there is such a $p(x)$, called a primitive polynomial, of every degree.

For example, the polynomial $x^4 + x + 1$ is primitive. So $\alpha = x$ is a primitive element, and successive powers of α will generate all non-zero elements of $GF(16)$:

$$
\begin{aligned}
\alpha^0 &= 1 \\
\alpha^1 &= x \\
\alpha^2 &= x^2 \\
\alpha^3 &= x^3 \\
\alpha^4 &= x + 1 \\
\alpha^5 &= x^2 + x \\
\alpha^6 &= x^3 + x^2 \\
\alpha^7 &= x^3 + x + 1 \\
\alpha^8 &= x^2 + 1 \\
\alpha^9 &= x^3 + x \\
\alpha^{10} &= x^2 + x + 1 \\
\alpha^{11} &= x^3 + x^2 + x \\
\alpha^{12} &= x^3 + x^2 + x + 1 \\
\alpha^{13} &= x^3 + x^2 + 1 \\
\alpha^{14} &= x^3 + 1 \\
\alpha^{15} &= 1
\end{aligned}
$$

This pattern of coefficients matches the bits in the table on Page 1.

Primitive Polynomials
$x^2 + x + 1$
$x^3 + x + 1$
$x^4 + x + 1$
$x^5 + x^2 + 1$
$x^6 + x + 1$
$x^7 + x^3 + 1$
$x^8 + x^4 + x^3 + x^2 + 1$
$x^9 + x^4 + 1$
$x^{10} + x^3 + 1$
$x^{11} + x^2 + 1$
$x^{12} + x^6 + x^4 + x + 1$
$x^{13} + x^4 + x^3 + x + 1$
$x^{14} + x^{10} + x^6 + x + 1$
$x^{15} + x + 1$
$x^{16} + x^{12} + x^3 + x + 1$
$x^{17} + x^3 + 1$
$x^{18} + x^7 + 1$
$x^{19} + x^5 + x^2 + x + 1$
$x^{20} + x^3 + 1$
$x^{21} + x^2 + 1$
$x^{22} + x + 1$
$x^{23} + x^5 + 1$
$x^{24} + x^7 + x^2 + x + 1$
$x^{25} + x^3 + 1$
$x^{26} + x^6 + x^2 + x + 1$
$x^{27} + x^5 + x^2 + x + 1$
$x^{28} + x^3 + 1$
$x^{29} + x + 1$
$x^{30} + x^6 + x^4 + x + 1$
$x^{31} + x^3 + 1$
$x^{32} + x^7 + x^6 + x^2 + 1$

In general, finding these primitive polynomials is difficult. Looking them up in a table, such as the one on the right, is easiest.

The equivalences between Galois fields and hardware are:

Galois Field		Hardware
Multiplication by x	$\Longleftrightarrow$	shift right
Taking the result mod $p(x)$	$\Longleftrightarrow$	XORing with the coefficients of $p(x)$ when the most significant coefficient is 1.
Obtaining all $2^n - 1$ non-zero elements by evaluating x^k for $k = 1, \ldots, 2^n - 1$	$\Longleftrightarrow$	Shifting and XORing $2^n - 1$ times

This is a small fraction of the power of Galois fields, which are used throughout the large, rich field of error control (error-correcting) codes.

Name: ______________________ Name: ______________________

Lab Section (Check one)

M: ☐11 ☐12 T: ☐13 W: ☐14 ☐15 Th: ☐16

8 Checkoffs

1. Use the table of primitive polynomials on Page 5 to draw a schematic for an LFSR with $2^9 - 1 = 511$ unique states. How many flip-flops do you need? How many XORs?

2. None of the counter and LFSR lights flash at exactly the same rate. Why are they not identical?

1. Divide-by-32768 and eight-bit counters working, displaying	TA:	______________	(30%)
2. Seven-bit LFSR with period 127	TA:	______________	(20%)
3. Eight-bit LFSR with period 255	TA:	______________	(20%)
4. Eight-bit LFSR with another period	TA:	______________	(20%)
5. **Extra Credit:** Make the SPARE button *toggle* between the two display modes	TA:	______________	(10%)
6. Turned in on time	TA:	______________	(10%)