Intro to Digital Logic, Lab 5
Useful Components

Lab Objectives

Over the last 6 labs we've learned how to do most kinds of basic logic, but there are some standard elements that tend to come up over and over again. This lab will help you get some experience with them now, and add them to your toolkit in advance of the final project.

Design Problem – CyberWar

In the last lab we built a simple Tug of War game, and by now you've already crushed your room-mate into submission. Now it's the hardware's turn. Your goal is to develop a computer opponent to play against, as well as a scorekeeper that can show exactly how badly it beats you…

Counters

First off, take your lab #6 and replace the "winner" system with counters. Specifically, develop a 3-bit counter (holds values 0..7). It starts at 0, and whenever a "win" comes in to it, it increments its current value by 1. This is a simple FSM. Note that we assume once one player gets to 7 the game is over, so it doesn't matter what happens when a player with 7 points gets one more.

Now, alter your lab #6 so that there is a counter for each player, which drives a per-player 7-segment display with the current score for that player. Whenever someone wins, you increment the appropriate player's score, then restart the game (i.e. automatically reset the playfield). Resetting the entire game will reset the playfield and score, while winning only resets the playfield.

LFSRs

To build a cyber-player, we need to create a random number generator to simulate the button presses. In hardware, the simplest way to do this is generally an LFSR (linear feedback shift register). It consists of a set of N D-flip-flops ($DFF_1 \ldots DFF_N$), where the output of DFF_i is the input of DFF_{i+1}. The magic comes in on the input of DFF_1. It is the XNOR of 2 or more outputs of the DFF. By picking the bits to XNOR carefully, you get an FSM that goes through a fairly random pattern, but with very simple hardware (note that these LFSRs can never go into the state with all 1's, but reach all others). Two examples are given below.

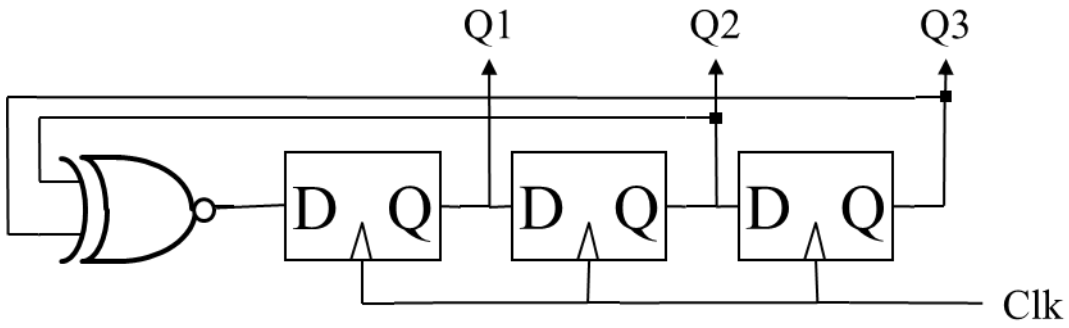

Figure 1. 3-bit LFSR

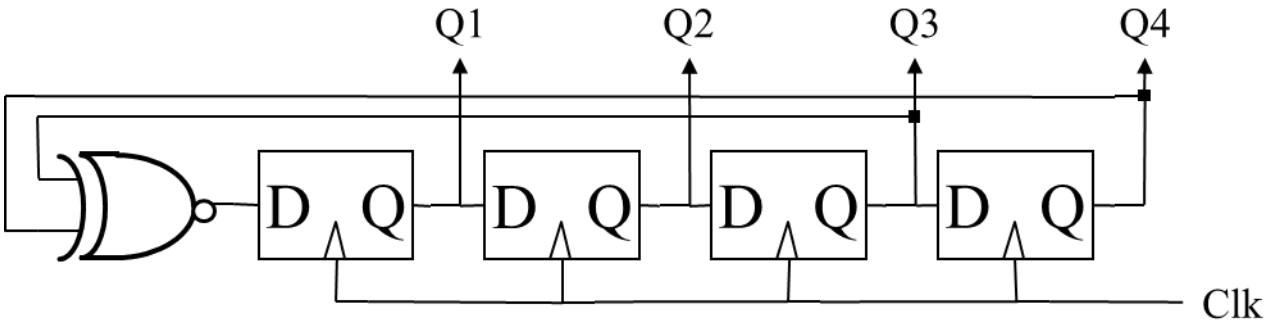

Figure 2. 4-bit LFSR

First, draw the state diagram for these two circuits. It will show every possible state for the machine (8 for the 3-bit, 16 for the 4-bit), with arrows showing the next state they enter after that state.

Next, create a 10-bit LFSR in Quartus II and simulate it. You can find the list of bits to XNOR together in the table at the end of this lab (do NOT make up your own – most choices don't work well, so the table shows the "best" connections to make).

> **ModelSim Tip:** now that you are working with multi-bit signals, it can be helpful in ModelSim to display signals as decimal or hex values. To do this, right-click on a multi-bit signal in ModelSim, and select "Radix". Hex and Unsigned are the most useful choices IMHO.

Comparator

Develop a 10-bit comparator. The unit takes in two unsigned 10-bit numbers, called A and B, and returns TRUE if A>B, FALSE otherwise. You can think about this as a subtraction problem, or just by considering the individual bits of the number themselves.

CyberPlayer

We now have most of the components to implement a tunable cyber-player. First, let's slow things down so you have a chance – run your entire Tug of War game off of the clock divider's divided_clocks[15] (about 768Hz). To generate the computer's button presses, compared the LFSR output (a value from 0…1023) to the value on SW[8:0] (a value from 0…511) – you can extend the SW[8:0] with a 0 at the top bit to make it a 10-bit unsigned value. If the SW value is greater than the LFSR value, consider this a computer button-press (i.e. the light should move one space toward the computer player's end, assuming the human doesn't make a move at the same instant). You can speed up or slow down the system by simply playing with the user switches, to see how fast you can go. If the clock is too fast, feel free to adjust to a different divided_clock output (for the ENTIRE design). Note: be sure that EVERYTHING that is clocked in your design (except for the clock_divider circuit) uses the same clock. If you use any other clock then strange things can happen.

> *USE CLOCK_50 during simulation! Use the divided_clock in your compiled design, but EVERYTHING must be clocked using that one clock.*

You will be graded 100 points on correctness, style, testing, testbenches, etc. Your bonus goal is developing the smallest circuit possible, measured in the same way as labs #5 & #6.

Lab Demonstration/Turn-In Requirements

A TA needs to "Check You Off" for each of the tasks listed below.

- Turn in your state diagrams derived from the 3-bit and 4-bit LFSRs.
- Turn in the Verilog for each of the modules developed in this lab, plus their testbenches.
- Demonstrate your complete system to the TA.
- Turn in a printout of the “Resource Utilization by Entity” page. Write on this the computed size for your design.
- Tell the TA how many hours (estimated) it took to complete this lab, including reading, planning, design, coding, debugging, testing, etc. Everything related to the lab (in total).

n	XNOR from	n	XNOR from	n	XNOR from	n	XNOR from
3	3,2	45	45,44,42,41	87	87,74	129	129,124
4	4,3	46	46,45,26,25	88	88,87,17,16	130	130,127
5	5,3	47	47,42	89	89,51	131	131,130,84,83
6	6,5	48	48,47,21,20	90	90,89,72,71	132	132,103
7	7,6	49	49,40	91	91,90,8,7	133	133,132,82,81
8	8,6,5,4	50	50,49,24,23	92	92,91,80,79	134	134,77
9	9,5	51	51,50,36,35	93	93,91	135	135,124
10	10,7	52	52,49	94	94,73	136	136,135,11,10
11	11,9	53	53,52,38,37	95	95,84	137	137,116
12	12,6,4,1	54	54,53,18,17	96	96,94,49,47	138	138,137,131,130
13	13,4,3,1	55	55,31	97	97,91	139	139,136,134,131
14	14,5,3,1	56	56,55,35,34	98	98,87	140	140,111
15	15,14	57	57,50	99	99,97,54,52	141	141,140,110,109
16	16,15,13,4	58	58,39	100	100,63	142	142,121
17	17,14	59	59,58,38,37	101	101,100,95,94	143	143,142,123,122
18	18,11	60	60,59	102	102,101,36,35	144	144,143,75,74
19	19,6,2,1	61	61,60,46,45	103	103,94	145	145,93
20	20,17	62	62,61,6,5	104	104,103,94,93	146	146,145,87,86
21	21,19	63	63,62	105	105,89	147	147,146,110,109
22	22,21	64	64,63,61,60	106	106,91	148	148,121
23	23,18	65	65,47	107	107,105,44,42	149	149,148,40,39
24	24,23,22,17	66	66,65,57,56	108	108,77	150	150,97
25	25,22	67	67,66,58,57	109	109,108,103,102	151	151,148
26	26,6,2,1	68	68,59	110	110,109,98,97	152	152,151,87,86
27	27,5,2,1	69	69,67,42,40	111	111,101	153	153,152
28	28,25	70	70,69,55,54	112	112,110,69,67	154	154,152,27,25
29	29,27	71	71,65	113	113,104	155	155,154,124,123
30	30,6,4,1	72	72,66,25,19	114	114,113,33,32	156	156,155,41,40
31	31,28	73	73,48	115	115,114,101,100	157	157,156,131,130
32	32,22,2,1	74	74,73,59,58	116	116,115,46,45	158	158,157,132,131
33	33,20	75	75,74,65,64	117	117,115,99,97	159	159,128
34	34,27,2,1	76	76,75,41,40	118	118,85	160	160,159,142,141
35	35,33	77	77,76,47,46	119	119,111	161	161,143
36	36,25	78	78,77,59,58	120	120,113,9,2	162	162,161,75,74
37	37,5,4,3,2,1	79	79,70	121	121,103	163	163,162,104,103
38	38,6,5,1	80	80,79,43,42	122	122,121,63,62	164	164,163,151,150
39	39,35	81	81,77	123	123,121	165	165,164,135,134
40	40,38,21,19	82	82,79,47,44	124	124,87	166	166,165,128,127
41	41,38	83	83,82,38,37	125	125,124,18,17	167	167,161
42	42,41,20,19	84	84,71	126	126,125,90,89	168	168,166,153,151
43	43,42,38,37	85	85,84,58,57	127	127,126		
44	44,43,18,17	86	86,85,74,73	128	128,126,101,99		

Figure 3. LFSR taps [XAPP 052 July 7, 1996 (Version 1.1), Peter Alfke, Xilinx Inc].