Jake Liao, Mark Zakharov

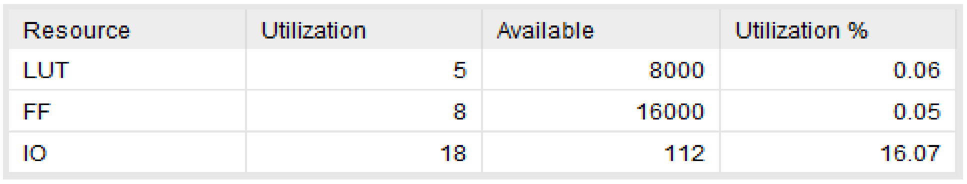
Lab 1

4-15-20

PART 1.

CLK	res_n	data_in	output
0	0	data_in	data_in
0	1	data_in	no change
1	0	data_in	data_in
1	1	data_in	shift

The Linear Feedback Shift Register (LFSR) is a series connection of flip flops, some of which are interconnected with XOR gates. The output of each flip flop is read serially to provide a pseudorandom binary integer. This pseudo-randomness comes from the XOR gates changing some of the bits of what would otherwise be a rotating byte. Liao designed the structural module of the LFSR, using eight flip flops alongside an 8-bit 2 to 1 multiplexor to control whether each flip flop would receive its upcoming signal from the previous flip flop or from user inputted data. The multiplexor "busmux_8b_2_1" is a modified version of a 2 to 1 multiplexor from CE100, modified to handle two 8 bit buses. The multiplexor selector "sel" was an active low reset signal, an asynchronous input from the user. FDCE flip-flops are used to conserve power. Clear and clock enable are set to 1'b0 and 1'b1 since they are not necessary in this implementation of the LFSR. Two wire buses "d_shift" and "flop_in" are used as the multiplexor's inputs. "flop_in" handles the default input when the LFSR is reset. "d_shift" handles the combination logic. With all input and output signals constrained to a delay of 2 ns and a clock set to 100 MHz (10 ns period), the design was synthesized to check the possible slack. The worst negative slack shown in Vivado actually represents greatest possible speed up of the design; this means the maximum frequency Fmax could be increased without risk of delay. Since the greatest delay (WNS) for this design came out to be 2.716 ns and the set clock period was 10 ns, Fmax can be calculated with (1/(T-WNS))=(1/(10-2.716))= 137 MHz. The Resource Utilization report of the Vivado Synthesis states the design used 5 LUTs and 8 FFs, which is expected using a structural design where 8 FFs were individually instantiated.



Resource	Utilization	Available	Utilization %
LUT	5	8000	0.06
FF	8	16000	0.05
IO	18	112	16.07

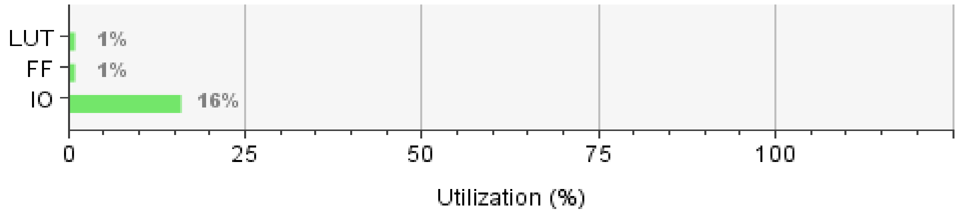
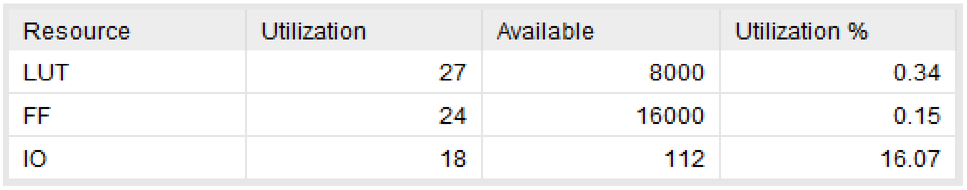


Figure 1: Structural LFSR resources

Zakharov designed the behavioral module of the LFSR, using a positive edge clock driven always block. The logic contained was an if/else statement, where if the reset signal was set low then the flip flops would be set to the inputted data in parallel, otherwise the flip flops (which are the output registers) swap values with the flip flop before them, except for three who XOR their signal and the last flip flops signal. Just like in the structural module, the in/out delay was set to 2 ns and the clock period was 10 ns in the constraints. The WNS came out to be 2.092, less than the structural model indicating a more complex (and less efficient) synthesized model. The Fmax came out to be 126 Mhz, 11 MHz less than the structural model. The Resource report shows that 27 LUTs and 24 FFs were generated in the design, which may explain why this design ran slower than its structural counterpart. In the always block the 8 output registers were mentioned 3 times, once when loading data_in and twice to swap each registers value with the previous one. This implementation may have created a holding and receiving register for each "middle" register, thereby having 3 per held value and 24 total. Nevertheless, both modules came out to the same end result, the first 10 values of which are: 0xA5, 0x57, 0xAE, 0x41, 0x82, 0x19, 0x32, 0x64, 0xC8, 0x8D.



Resource	Utilization	Available	Utilization %
LUT	27	8000	0.34
FF	24	16000	0.15
IO	18	112	16.07

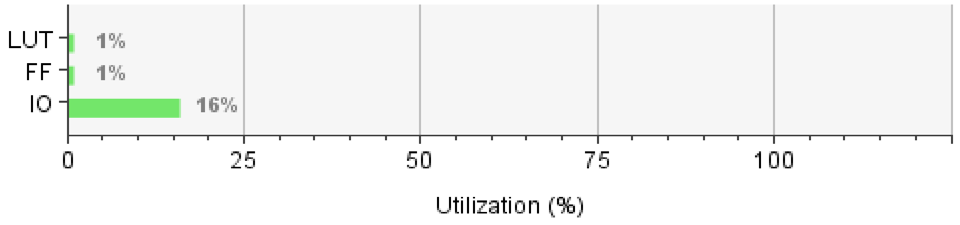
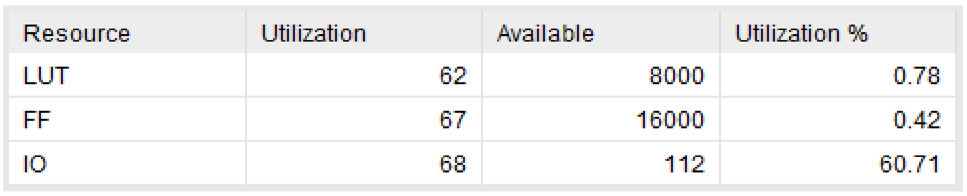


Figure 2: Behavioral LFSR resources

PART 2.

The simplest way to build a multiplier is through an accumulator, a hardware block that incrementally adds to its stored value. By setting argument 1 of the multiplicative operation to a counter that makes the accumulator add argument 2 each time the counter counts simulated multiplication. Such a design was set up by Liao in the form of a Moore state machine by using case statement. The case default initializes the PRODUCT, COUNT, and DONE to 0 and transitions to STATE 00. In STATE 00, PRODUCT, COUNT, and DONE remains at 0 and remains in STATE 00 until START is high, which transitions STATE to 01. In STATE 01, the accumulator increments PRODUCT by ARG1 and increments count by 1 until count equals to ARG2, where STATE transitions to 10. In STATE 10, DONE is held at high until res_n transitions the STATE back to 01. Only 3 states are used, if for any reason STATE enters 11, STATE will immediately return to 00.

The simulation ran on the multiplied module shows that it took it 455 ns to reach the desired result of 67*43, approximately 45 clock cycles of the constrained 10 ns clock period. This is correct as 43 was used as the counter value and 1 clock cycle was used to initialize the FFs and 1 was used to recognize the counter had completed. Using 2 ns delay for each in/out the WNS came out to be 2.716 ns which results in a 137 MHz Fmax. The resources used list 62 LUTs and 67 FFs, this may be a result of the product output being a 32 bit register and being added to itself using the accumulator required two of them, taking up 64 FFs with the other 3 FFs being the done signal and the 2-bit state machine.



Resource	Utilization	Available	Utilization %
LUT	62	8000	0.78
FF	67	16000	0.42
IO	68	112	60.71

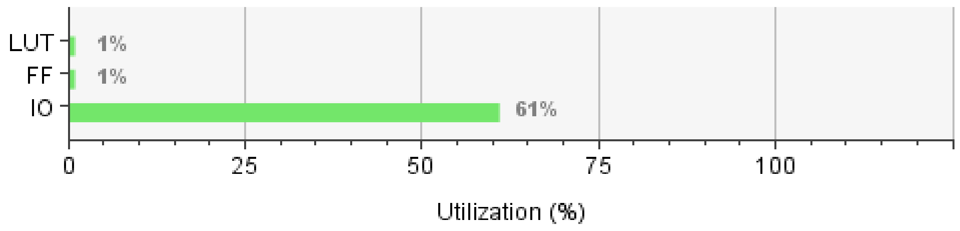
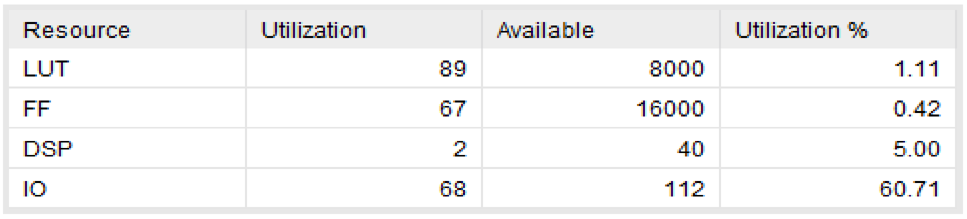


Figure 3: Simple Multiplicative Accumulator resources

The quadruple accumulator module was built by Zakharov, taking the skeleton of the simple multiplier model and modifying the conditionals of the second state. This implementation does four accumulations per clock cycle, so the COUNT was incremented by 4 each time the state was reentered if the difference between the COUNT and ARG2 is greater than or equal to 4 so as to not over-add. Given this, ARG1 was added to PRODUCT in the form of ARG1*4 to represent 4 accumulations. If COUNT was found to be less than 4 but not equal to ARG2, then ARG1*(ARG2-COUNT) would be added to product so as to not over-accumulate. It took the simulation to output the desired result 145 ns, which is less than 4 times as fast as it took the simple accumulator; this is due to needing extra clock cycles to switch between states and initialize values. Running the same 10 ns clock and 2 ns in/out constraints led to a 1.160 ns WNS, which means Fmax is 113 MHz, a frequency considerably slower than the simple model. The resources used by this module had the same amount of FFs as the uses of the PRODUCT output registers did not change from the simple model. There are now 89 LUTs from the previous 62, these were added due to the increased complexity of the second state which handled COUNT differently and also required an addition based on the difference between COUNT and ARG2, for which 2 DSPs were added to the design.



Resource	Utilization	Available	Utilization %
LUT	89	8000	1.11
FF	67	16000	0.42
DSP	2	40	5.00
IO	68	112	60.71

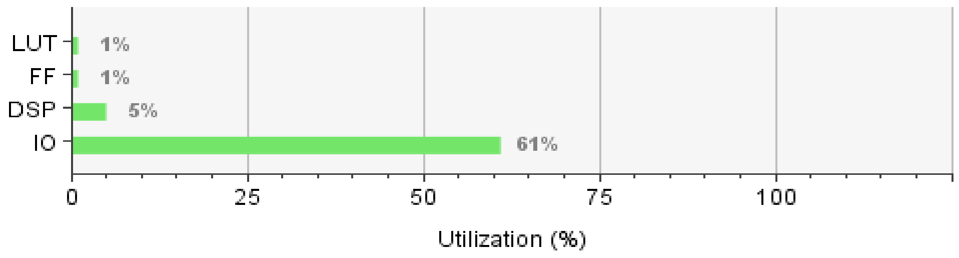


Figure 4: Quadruple Accumulator Multiplier