EE 331 Digital Systems With VHDL
Oregon Tech, Wilsonville, Winter 2014

Lab Assignment 8, due Finals Week, March 18

In this lab, like the last one, you will be displaying prime numbers. The difference is that in this lab you will be calculating the prime numbers. Your program needs to be able to display all the prime numbers less than 65536, starting from 2. The Sieve of Eratosthenes will be used to discover the prime numbers.

We will revisit prime number generation in an entirely different way (but perhaps with the same algorithm) in EE432 next term.

You will need your working Lab 7 assignment as a starting point for this one. The ROM will need to be replaced with a RAM, and there will need to be a new register which drives the address lines of the RAM. The existing counters (BCD for display and binary for calculations) remain. You will also need to modify the state machine.

The Sieve algorithm requires that we first set all memory locations to 0, indicating prime numbers. This can be done by having the RAM initialize itself to all 0's. We start operation with the counters advanced to 1, then we find the next prime, 2. Every time a prime number is found, all locations in the RAM at addresses that are a multiple of the prime number are marked as not-prime by setting those locations to 1. We stop and display the prime number until the next button press to advance.

To implement the design, we need the following components (modules, if you wish):

- A 16-bit counter which contains the value we are testing. I called this *test_counter*. It needs a control input for incrementing, *incrTC*. We assert incrTC high at the same time we increment the BCD counter for the display.
- A 16-bit memory address register, *memory_address*, which drives the RAM's address inputs. The register can be loaded from two sources, either test_counter (control signal named *loadTCtoMA)* or from the sum of *memory_address* and *test_counter* (control signal named *addTCtoMA*) Adding the test counter to an existing memory address value is useful for advancing through multiples of a discovered prime.
- A 17-bit adder which adds the contents of *memory_address* and *test_counter*. The extra bit is needed to know if the sum has overflowed beyond 65535. The lower 16 bits of the sum are used for the *addTCtoMA* operation in the memory address register and the upper bit is used as an overflow signal input to the state machine, *adderov*, so the state machine knows when the last prime multiple has been exceeded.
- The generated 65,536x1 RAM will have a write enable signal, *wea*, that will cause data to be written to the RAM on the next active clock edge. This signal is

generated by the state machine. We only write a 1 to the RAM, so the data input is fixed at '1'. The data out of the RAM, *douta*, is an input to the state machine, just like in the last lab assignment.
- A signal *memCounterffff* which is 1 when the memory address counter is 0xffff (all 1's) and 0 otherwise. This signal goes to the state machine and is used to determine when the end of memory has been reached. At this point operation stops.

Segment 2 — The State Machine
With all the calculation modules in place, you should be able to build a state machine that will find the prime numbers. You can, of course, add additional circuits if you find my suggestion to be incomplete or undesirable. I do know that it is sufficient to solve the problem.

Your lab report should contain the following data:

- VHDL source for all modules.
- The RTL schematic of the synthesized VHDL code (can be printed from Xilinx ISE).
- Design Summary (can be printed from Xilinx ISE)
- Any simulation output you have
- If you have a camera or video recorder, include photographs of operation or submit a video recording with your report.