

EE 331 Digital Systems With VHDL

Lab Assignment 5

Winter 2012

Objectives: Gain familiarity with the Xilinx tools by using Xilinx CORE Generator to add a RAM to a circuit design. Perform arithmetic operations and implement a much more complicated state machine. Test modules during development with the simulator and verify operation with the Nexys 2 FPGA board.

You will need your working Lab 4 assignment as a starting point for this one. Copy all the files into a new folder, rename the `xxxx.xise` and `xxxx.gise` files from `xxxx` (the old project name) to your new project name. Open the new project in ISE and synthesize to verify that it transferred correctly.

Introduction

When finished, this design will display all prime numbers less than 65,536, starting from 2. The circuit will first find the prime numbers using the *Sieve of Eratosthenes* which you should look up if you are unfamiliar with it. Wikipedia is a good resource.

From lab assignment 4 you will re-use the top level, bcd counter, display driver, enables and debounce-sync modules. This circuit uses a 65,536x1 RAM (instead of ROM) which will indicate values that are prime. Once the primes are found, states in the state machine will display successive prime numbers with each count button press, analogous to operation of the Lab Assignment 4.

The reset button will cause a recalculation of the prime numbers. Make sure this button works properly!

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

Segment 1 – Calculation Circuits

The Sieve algorithm requires that we first set all memory locations to 0, indicating prime numbers, except for locations 0 and 1 which we set to 1, indicating that they are not prime. Then the algorithm advances through memory finding the next prime number. 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.

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

- A 16-bit counter which contains the value we are testing. I called this `test_counter`. It needs control inputs for reset, `resetTC`, and incrementing, `incrTC`.

- A 16-bit memory address counter which drives the RAM's address inputs. Much like what we had in Lab Assignment 4, it needs to be able to reset and increment, and those operations are done at the same time the BCD Counter is reset and incremented. However, an additional operation is needed, one which adds the value in the test_counter to the memory address counter. I call this control input *addTMC*. If the memory address register is reset and then the *addTMC* operation is performed, the contents of the Test Counter get copied to the memory address register. 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 the memory address and test counters. 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 *addTMC* operation in the memory address counter 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. The signal for data to the RAM is named *dina* and is generated by the state machine. The data out of the RAM, *douta*, is an input to the state machine, just like in Lab Assignment 4.
- 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.

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. You might also want to add a comparison for the test counter being 255 since there is no need to test values .

My solution has 16 states. Note that since the state machine calculates the primes before it checks the push buttons it can be easily simulated at the top level and the process of finding the primes monitored. This will be very useful in debugging the design since with so many states it is unlikely to work correctly the first time.

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.