CDA4203L.001
Computer System Design Lab
Spring 2015

Chris Frazier
John Gangemi
Bassam Saed

Lab 5 Report

Introduction

A Finite State Machine and Datapath (FSMD) is best described as a custom single-purpose processor in which there exists a controller and datapath element sharing a number of communication paths.

In Lab 5 the construct of a FSMD is utilized to compute the greatest common denominator of two 4-bit numbers and output the result on LEDs 3-0 of the Atlys FPGA board. As provided in the lecture, two design possibilities can be used to achieve the desired results. One design is a direct one-to-one mapping of the Euclidean algorithm pseudo-code whereas the second design is an optimized version of the first using a fewer number of states.

The top-module should consist of two components, a controller and datapath, where the controller must be a *pure* FSM and the *structural* datapath should function solely based on signals generated from the controller.

Design

It was decided that the final design used for computing the greatest common denominator will be the one which implements a direct mapping of the Euclidean algorithm pseudo-code. No particular reason for choosing this design over the optimized design, it was purely preference (more states == more fun!!!).

While the *normal* design was provided, the final design implemented was altered minimally to satisfy the given design constraints specified in the lab handout. All design deviations/altercations are listed below:

1) external controller input *go_i* changed to *start*
2) datapath output *d_o* changed to *gcd_out*
3) controller output *done* added
    - asserted when the GCD has been computed and held until reset
4) controller/datapath input *reset* added

When creating the structural datapath in Verilog a number of arithmetic modules, sequential register modules, and selection modules were included. The type of module and its function can be shown in a table.

Module Type	Module Function	Number of Instances
Selection	2:1 Multiplexer	2
Sequential	Register	3

Arithmetic	Not Equal	1
Arithmetic	Less Than	1
Arithmetic	Subtraction	2

When creating the FSM controller in Verilog a generic always-block for current state output and always-block for next state assignment was utilized. A single state register *state* held both the current state and next state binary encoding. The initial FSM design placed all output assignments in the always-block for current state output, however this caused an issue that will be discussed later in the *Testing* portion of this report. Suffice to say the final FSM design assigns values to *done* register in a non-blocking fashion in the always-block for next state assignment.

A Verilog wrapper module was created to instantiate instances of the controller and datapath modules, it also includes a module for debouncing the *start* button. The debounce module was sourced from a MIT Verilog resources web-page, likewise group ISIS claims no credit and/or affiliation with MIT.

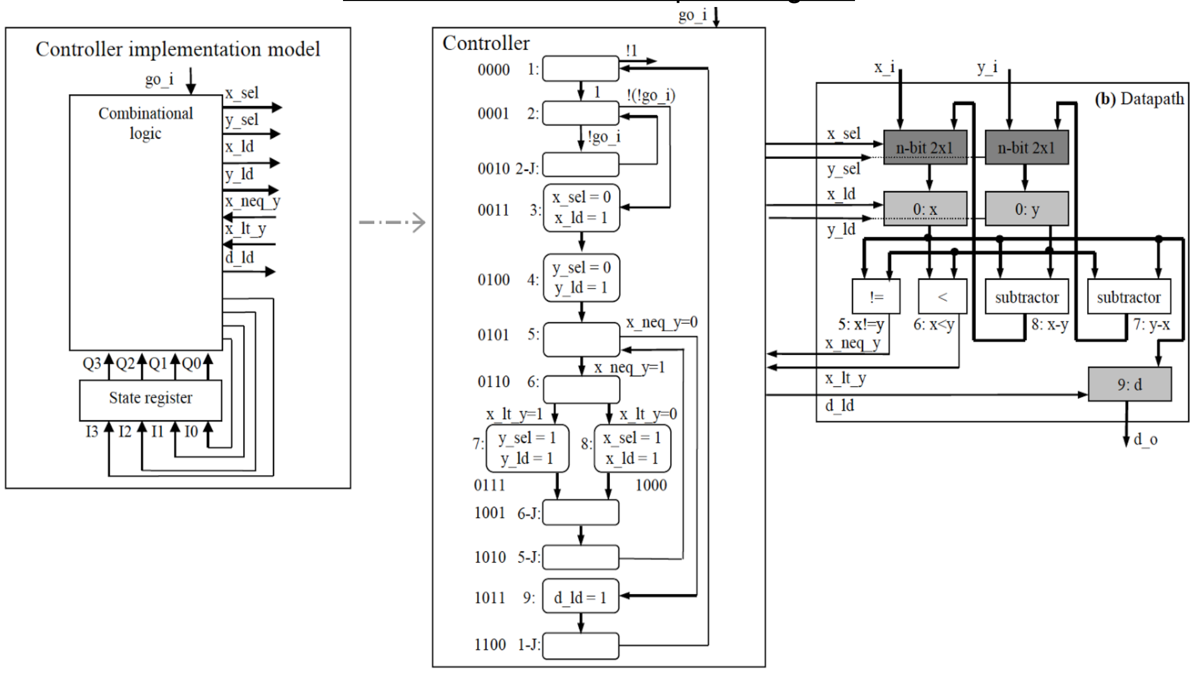
All FSMD design resources provided by Dr. Srinivas Katkoori for the computation of a greatest common denominator of two numbers:

<u>Euclidean Algorithm Pseudo-Code</u>

```
0: int  x, y;
1: while (1) {
2:   while (!go_i);
3:   x = x_i;
4:   y = y_i;
5:   while  (x != y)  {
6:       if  (x < y)
7:           y = y - x;
         else
8:           x = x - y;
       }
9:    d_o = x;
   }
```

Embedded Systems Design: A Unified Hardware/Software Introduction, (c) 2000 Vahid/Givargis

Controller FSM and Datapath Diagram



Embedded Systems Design: A Unified Hardware/Software Introduction, (c) 2000 Vahid/Givargis

Testing

Simulating the FSMD proved to be difficult such that the number of states processed to produce output was not constant amongst a set of differing inputs. In essence figuring out the necessary runtime for two numbers had to be calculated mathematically and was the most tedious part of the lab assignment. Thus, to simplify testing the “$finish” statement was excluded from the testbench and simulations were all run well beyond the total time needed to find the GCD of two numbers.

As mentioned earlier in the *Design* section there were issues with the placement of *done* register assignments in the always-block dedicated to current state output, such that designated LED 7 would not “function” properly upon initial physical testing. Believing there was an issue with onboard LED 7 the pin mapping was quickly altered to change the mapping of *done* output to LED 6. This proved to be successful but why?

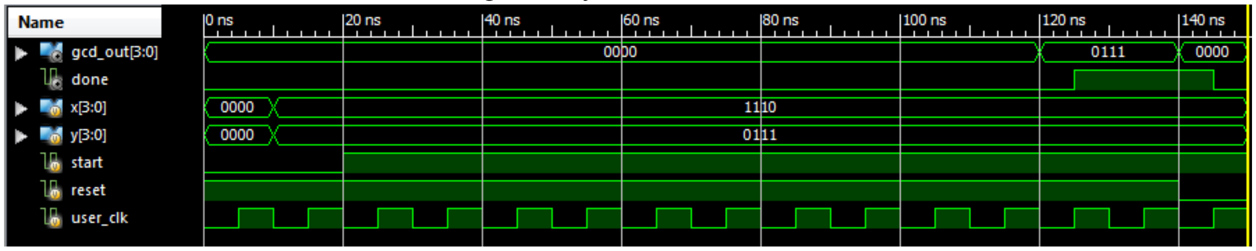
After having informed instructor Chris Bell to a possible dead LED on the Atlys board a number of diagnostic tests were performed in an attempt to find a culprit. Checking design synthesis showed a couple of hidden warnings outlining a “latch-up” condition for the *done* register in the FSM module.

The solution to the “latch-up” issue was to place all *done* register assignments in the always-block for next state assignment. This simple alteration eliminated all warnings and allowed LED 7 to function properly. Paraphrasing from instructor Chris Bell, a possible reason for LED 7 not behaving as expected with the initial FSM design could be due it sharing connections to other components on the board and therefore it’s not as robust as the other dedicated LEDs (expect LED 5). While it would be convenient to fully attribute the issue with

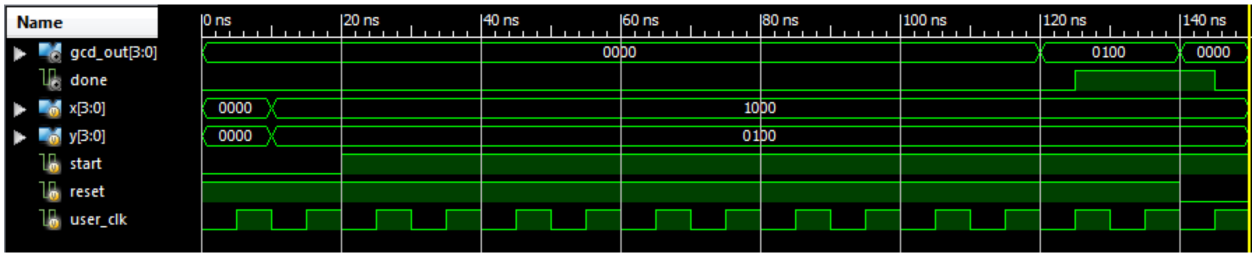
LED 7 to its lack of "robustness" we feel that the issue had more to do with in-efficient FSM practices than physical hardware limitations.

Results

Testing Evenly Divisible Numbers

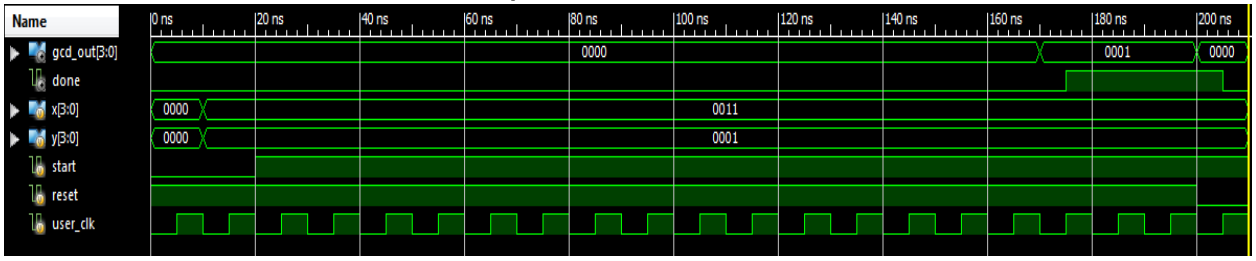


This image shows the GCD when x = 14 and y = 7. The GCD is computed around 120 ns.



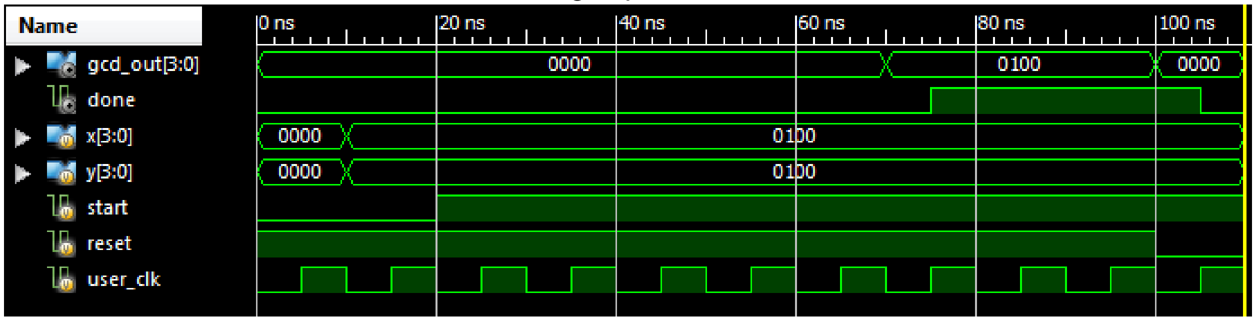
This image shows the GCD when x = 8 and y = 4 around 120 ns. The number of cycles required for GCD computation is equal to that of the first test.

Testing Lowest Possible GCD



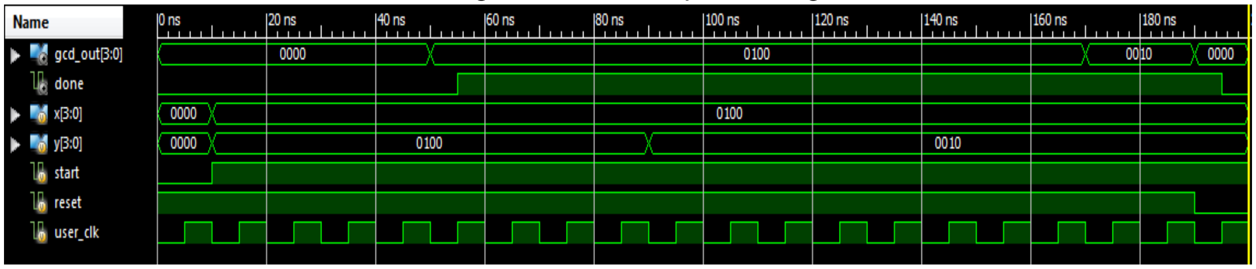
This image shows a test between two numbers that generates the lowest possible GCD of 1. The amount of time needed to find the GCD was far greater than two numbers that are evenly divisible.

Testing Equivalence



This image shows a test between two equivalent numbers and producing a GCD of the same value rather quickly around 70 ns.

Testing Continuous Input Changes



In this image the initial set of values for x and y were both 4, after some number of clock cycles a GCD was computed. Some cycles later the value of y was changed to 2 and as expected the GCD was computed a number of clock cycles later.

Conclusion

This lab successfully demonstrated the combined functionality of a datapath and a controller when dealing with finite state machines and sequential logic. The greatest common denominator example was fitting for an introduction to these concepts because the algorithm is fairly straightforward and easy to implement. In order to complete the task, we had to use the skills we learned from the previous labs, including structural design and state encoding, in addition to new concepts picked up from the lecture. In future labs, we will be able to look at an algorithm and derive a FSMD and implement a datapath and controller without hesitation.

Acknowledgements

*All work was distributed evenly amongst all team-members.
*debounce module is the original work of M.I.T.
http://web.mit.edu/6.111/www/f2005/code/jtag2mem_6111/debounce.v.html