SIMPLE PROCESSOR

LAB REPORT 5 FOR ECE327

DIGITAL SYSTEMS DESIGN

SUBMITTED BY

JULIAN COY

UNDERGRADUATE OF ELECTRICAL & COMPUTER ENGINEERING

CLEMSON UNIVERSITY

APRIL 28, 2014

Abstract

A system that can perform different operations based on numerical operations considered a processor. Processors are the basic control units of all computing. The task of this lab was to implement a simple processor system. The first part of the lab was to build a processor that contained a full adder that was also capable of subtraction. Located in the processor were 8 16-bit registers, as well as other registers required by the system (in this lab, two were needed). The second part of the lab was to create a memory module that would feed the processor instructions by reading them sequentially from a ROM module. The final portion of the lab was to identify critical paths in the processor circuit and then postulate methods to obtain the maximum viable frequency.

Note: All clocks generated for simulation were built using Morten Zilmers clock gen package [1].

INTRODUCTION

The purpose of this lab is to simulate a simple 16-bit processor. A diagram of the system can be seen in Figure 1.1. The full adder/subtracter and multiplexer will operate instantaneously and not need to be operated by a clock. The code for each module is provided in the code appendices at the end of the report.

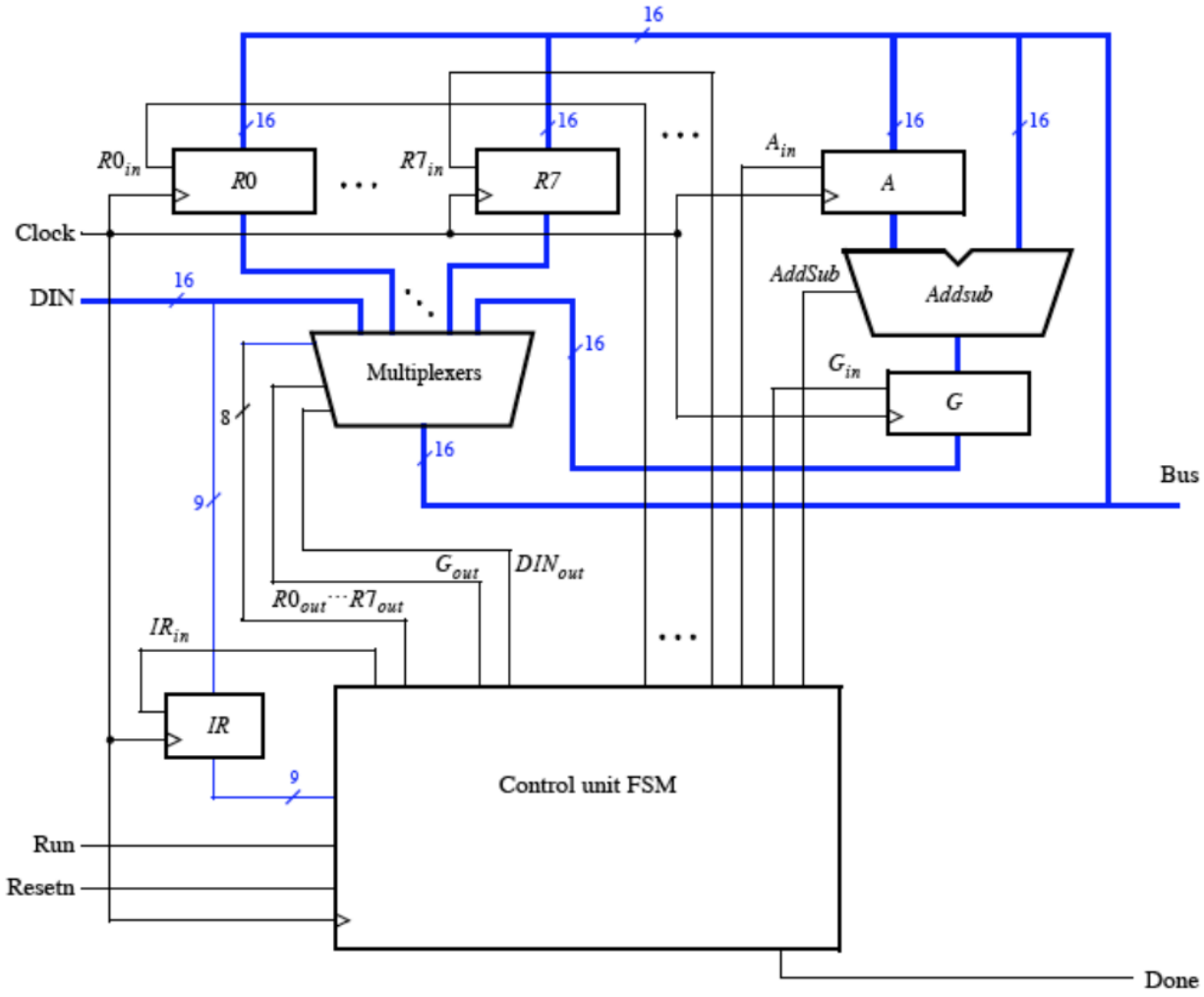

Figure 1.1: Simple Processor

There are four main components to this design: the control state module, the registers, the multiplexer, and the full adder/subtracter. The IR register is similar to the other registers, but only stores a 9 bit value. The multiplexer logic is contained inside the control FSM due to it's simplicity.

1.1 LAB 4 SYSTEM DESIGN

1.1.1 REGISTER DESIGN

There are only two register prototypes in this design, and both are incredibly similar. Registers 0-7, A, and G are all 16 bit flip flops. They store a value on the rising edge of the clock only when their select line is driven high. The instruction register (IR) is the exact same as the others, but only utilizes 9 bit buses and storage.

The only real variation in the registers is their data line connections. Registers 0-7 feed from a 16 bit data bus and output to the multiplexer. Register A feeds from the data bus as well, but instead of the multiplexer, it outputs to the full adder/subtracter (adder). Register G's input is from the output of the adder unit and outputs back into the multiplexer. The most different of the connections is again for the IR. The IR captures the 9 MSBs of the data in (DIN) line. It also outputs its data directly to the control unit.

1.1.2 TESTING OF THE REGISTERS

Each unit was tested in groups. This is due to the reliance of one unit upon signals from other units. It may have been simpler to test units individually, but the method chosen proved to provide quicker results (although, debugging was more difficult). Below are the simulation results for the different unit tests.

The test shown in Figure **??** shows the manipulations that are performed on the multiplicand via register A. Notice that as the load value goes high, the first code is produced: 100. This code, and the following codes shown, align perfectly with the expectations for a bit-pair coding scheme.

1.1.3 TESTING OF REGISTERS B AND D

Registers B and D were tested at the same time to show that the counter will stop when the maximum number of shifts have occurred. This was not necessary to do, but it aids the user in understanding why the counter is needed, as the controller has no other way of keeping track of the amount of shifts performed.

Notice that the shifts are spaced slightly. This occurs when an add operation is warranted. Later on in the register C test, you will be able to see the add signal being used. The disparity between clock cycles used per operation is the reason for the counter. Without it there would be know way to know preemptively how many clock cycles would be needed for a given multiplication.

1.1.4 TESTING OF REGISTER C

Register C was fairly simple to implement, once the other units were designed. In the test, you will notice that the registers values change based upon the signal changes that occur upon the signals shift, add, and load. Register C performed the same operations as B except it did not send off the lowest three bytes for coding purposes.

1.2 CONCLUSIONS

There are definite changes that could be made to this project to increase the efficiency and decrease the overall runtime of the algorithm. Mainly making the system asynchronous. Do do that, more signals would need to be added to the control unit. In particular, bus read signals would be needed to make sure register C doesn't stomp out its own value through the adder loop.

Overall, this lab shows that complex algorithms can be implemented in VHDL by creating small modules, one piece at a time. If anything else can be taken away, it is that testing is best done on INDIVIDUAL units, not groups. By testing entities individually, debugging is greatly simplified and overall development time and stress are reduced.

WORKS CITED

[1] M. Zilmer, "Clock Generator." Internet: `http://stackoverflow.com/questions/17904514/vhdl-how-should-i-create-a-clock-in-a-testbench`, 2013.

APPENDIX A: LAB 4 CODE

```
-- something
```

```
-- something
```

```
-- something
```

```
-- something
```

```
-- something
```

```
-- something
```

```
-- something
```

```
-- something
```