EECS/CSE 31L Midterm Project

Leo Leal, Eliseo Nunez, Donald Isbell, Manpreet Singh

10/30/2017

Introduction

The vision of this project was to complete the design of an 32-bits enhanced ALU (Arithmetic Logic Unit) to perform a variety of functions beyond that of standard addition or subtraction. Upon undertaking this task we took upon ourselves to set goals for ourselves in order to create the most efficient and relevant ALU as possible. The first of our goals was modualbility; a focus on this idea allowed for our code to be able to be expanded and contracted as needed, as well being able to divide the ALU into smaller modules allows for us to be able to expedite debugging and implement non-problematic optimization. Our second goal was efficiency; although not particularly problematic in this specific project, we strove to use a minimal amount of variables and through the use of deductive logic, minimize the amount of gates required thus decreasing the amount of time for the circuit to complete. Our last goal for the project was readability; which is slightly tied into our previous goals because at the end of the day in order for our code to be reusable it needed to be clear and concise so that if anyone picked up the code, even with a lack of comments they would be able to understand what would be happening in the circuit. Through our focus on these three goals we would be able to complete this project and design an enhanced ALU that would surpass expectations.

Design

Initialization

The overall design over our code includes 3 inputs and 5 outputs. The inputs include the two 32 bit numbers and the operation code which determines what answer will be provided in the output of the circuit. As well, we have the 4 flags as outputs with the carryout ,

overflow, zero, and sign flags. There are twelve functions that the circuit performs whenever numbers are present in the input. There is only and output however when there is the presence of an the operation code that indicates which function result is to be provided.

Function/Processing

After initialization, all twelve functions begin to process the inputs. All the functions work in their standard form, receiving the two numbered inputs as well as the indexed flag outputs. All of these modules are separated into their own source files and are called into the main ALU. All of the functions store their answers into an output that leads to our selection algorithm. Our selection algorithm goes through the operation code inputs from least significant bit to most significant combing through all of the 12 possible outcomes in 12 steps by using the assignment operators.

Output

There are a total of four Multiplexers that we use in this code, one for each of the flags. The multiplexer is effective for determining quickly which one of the indexed values we need for the flags. The overall output is determined through those 12 lines of code mentioned in the previous statement that saved our circuit from having to have 32 twelve to one multiplexers which would be an extreme amount of gates. Instead we were able to run all the values through strategically placed gates.

Testing

When it came to testing our code we ran it through many cases, focusing especially on extraneous cases that could possibly cause the most errors especially when it came to our increment in decrements circuits. We experimented to make sure nothing would happen if simply one number were forced, or if there would be any issues with extremely large or small numbers that would be represented in our machine. Overall very few bugs were ever found, and of those that were, we discovered that they had old code that had not been updated from the server yet which thus left to faulty testing. We threw out the old info and once the issue was corrected we began the process again. By the end of our efforts we were unable to determine any bugs in the code as far as we could see. It was efficient and accurate just as we had intended it to be. By the end of our investigation for issues over 50 tests had been performed on the code (some of which

you can view in the folders provided), all said and done we had not yet found any issues with the code.

Conclusion

At the beginning of this project we had our problem before us, developing an enhanced ALU capable of performing twelve of these specific functions on 32 bit numbers. In wake of this issue we decided to set for ourselves three goals: Moduability, Efficiency and Readability. From our tremendous amount of source files of which are all separated into modules of which could be used in other projects. We developed a code that would be able to grow and change into whatever we may need it to be in the future. With our efficient logic in determining a way in which we were able to cycle through much more information with less gates thus creating a machine that runs cooler and faster. Through viewing all of our .sv files it you can clearly see the path we took with our project and can understand all that is going on in this ALU even if you don't have an extensive knowledge of systemverilog. Lastly through developing a code that had no issues in our debugging phases we can firmly say without a doubt that this project has been a success. We planned what we wanted to put in and get out of the project, constructed a way to complete our goal and achieve our success.