Lab 2 – Boolean Arithmetic

Nathan Hirsh

Csc345 – Berry College

September 6, 2016

Project Summary and Procedure:

Throughout the course of this project, my understanding of binary numbers has expanded immensely. This lab has shown me how binary numbers are used and how to add and combine them. Instead of having a base 10 like the normal number system, binary has a base 2. This means any binary number can be 2 to any power: 2^(n). In the lab, I started with the half-adder, and similar to the last project, built others on top. The half adder simply starts by adding two bits with 2 outputs being the sum and carry. The sum was what the two numbers added to equal, and the carry is used as overflow, just like in basic arithmetic. Next, the Full-Adder adds 3 bits. Likewise, the Add16 adds a 16 bit number. These additions of true and false Boolean values are what “sum” up the chapter of Boolean Arithmetic (no pun intended). Each of these chips are very generic and hold true for all computers.

All of these adder chips, in addition to Inc16 (used to add 1 to a given number), are used to create the Arithmetic Logic Unit chip (ALU). This chip is used as the main piece of a computer platform called *Hack*. The ALU can produce up to 2^(6) functions (64), because it has 6 control bits. The ALU has 2 16bit inputs and 6 other inputs. It outputs a 16bit out and 2 separate Boolean outputs. (Page 36, Figure 2.5)

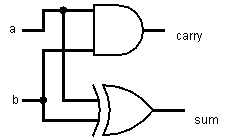
The project provided the .cmp and .tst files, and had the .hdl files for us to fill in with the necessary code. Like the last project, we used the hardware simulator to test whether our code would pass. The textbook, and the names of the chips, helped us to understand what order to complete them in. The importance of order has been made much more apparent as each chip builds on the previous.

Results:

Our method of completing this project worked very well in general. Instead of trying to split up the work as much, we found it much faster and more helpful to all meet and work on it together. When one person got stuck, 2 others would be there to put our heads together to figure out the problem. Sometimes the mistake was an incorrect approach, other times, and more commonly, it was something as simple as a missing semicolon. This made the project much smoother and much more enjoyable, at least as enjoyable as school gets.

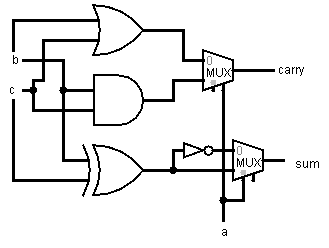
We used the textbook and the provided files side-by-side in our approach. We first looked up the rudimentary properties and implementation of each chip. Often we read further into the details of how it should be used. Once we felt we had a decent idea, we would start writing code with a little trial and error. If we were still struggling, we usually would diagram the chip to help us better understand it.

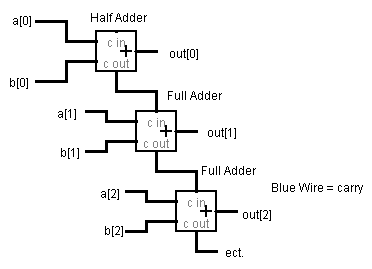
With the HalfAdder we used a Xor and And gate. The And gate gave the simple Sum and the Xor provided the Carry. This resulted in the intended addition of 2 bits.



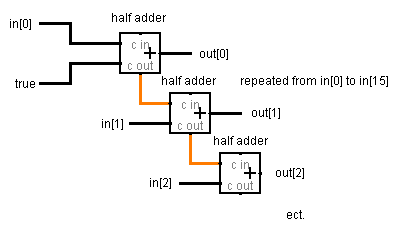
With the FullAdder, we should have used two HalfAdders, but sometimes we decide to make things more complicated than we have to, unintentionally of course. At the time, we did’nt recognize that we could use two HalfAdders.

Instead we used And, Or, Not, and Xor as well as 2 Mux gates. We used the And and Or gates to give values to plug into the first Mux. This resulted in out Carry. Likewise, we used the Not and Xor gates to provide the second Mux to result in our sum. Both Mux gates used the same sel=a.



In our Add16 chip, we decided to be a little smarter and use the HalfAdder and FullAdder in our code to make it simpler. We did this by beginning with a HalfAdder and using the resulting Carry as our c for the FullAdder. Likewise, each FullAdder used the Carry from the previous as its c value. This results in bits sum values and 16 carry values. 

The Inc16 chip was composed of 16 HalfAdders. In a similar way to Add16, each half adder passed its carry to the b of the next HalfAdder. This results in 16 sum values with 16 Carry values.



For the ALU, the code was most likely more complicated than it needed to be. Our approach was affective, but took a long time. Dr. Hamid advised that we try to figure just one part of the ALU out at a time instead of attempting to complete it all at once. We saw this as valuable advice so approached the chip in a way that tackled one step at a time.

See Code Below:

*PARTS:*

*Mux16(a=x, b=false, sel=zx, out=zxO); // If zx is set itll select 16 bits of 0 (false), x otherwise*

*Not16(in=zxO, out=notl);//Negate x using a Not16.*

*Mux16(a=zxO, b=notl, sel=nx, out=m); //Use Mux16 to select between negated x or zxOut.*

*Mux16(a=y, b=false, sel=zy, out=n); //see line 48*

*Not16(in=n, out=notn); //see line 50*

*Mux16(a=n, b=notn, sel=ny, out=o); //see line 51*

*And16(a=m, b=o, out=I);*

*Add16(a=m, b=o, out=l);*

*Mux16(a=I, b=l, sel=f, out=d);*

*Not16(in=d, out=notd);*

*Mux16(a=d, b=notd, sel=no, out[15]=s, out[0..14]=t);*

*Mux16(a=d, b=notd, sel=no, out[0..7]=aa, out[8..15]=bb);*

*Or8Way(in=aa, out=xx);*

*Or8Way(in=bb, out=zz);*

*Or(a=xx, b=zz, out=yy);*

*Not(in=yy, out=zr);*

*Mux16(a=d, b=notd, sel=no, out=out);*

*Mux(a=false, b=true, sel=s, out=ng);*

The code above shows the route we took, even though it was most likely more complicated than it had to be. But it worked! We took one step at a time to achieve the next.

Self-Evaluation:

I believe our group deserves a 20 because every file was completed while passing all of the test cases.

Conclusion:

We moved pretty fast throughout most of these files, but definitely got held up on the ALU file. I think one thing we struggled with was attempting to complete one step at a time while still using the previous adding chips to assist in the next. I think we often got a sort of tunnel vision and did not recognize alternate ways to complete the files. Also, it is often very difficult to locate where an error is coming from. I have used a few different coding programs and languages, and with each one comes a different way to check and communicate errors. The tester in the Hardware Simulator is great, but often the error it gives is very useless to us. It often gives very generic problems like “error line 16”. However, we did get better at understanding what the hardware simulator errors were referring to as we become more experienced.

In the end, just like most of my coding class of any kind, I quite enjoyed it. Often the process is full of errors and annoying red letters, and coding still seems to take me at least 1.357 years to start to grasp the concepts, but it is rewarding. I enjoy learning and this project is helping me learn what may be very useful in my future career. So in summary, this coding stinks and I really enjoy it and look forward to the next project.