LE/EECS 2021 4.00   Computer Organization

LAB K REPORT

Michael Williams 211087798

Section E

November 3rd 2014

The work in this report is my own. I have read and understood York University academic dishonesty policy and I did not violate the senate dishonesty policy in writing this report.

ABSTRACT

In this lab, we were taught the basics and fundamentals of Verilog programming. This included printing out statements, formatting output, and reading inputs. This also meant learning how the registers and operands work with Verilog, as well as how to manipulate them at specific points in time. Finally, the lab introduced us to the many uses of the logical operands, being And, Or, Not, etc and how to use them, in conjunction with regular Verilog coding, to create specifically designed circuits and run them, both correctly, and efficiently, while using as little coding as possible and still maintaining the ability to understand the coding easily. The purpose was to teach us the basics behind Verilog as mentioned above, in order to create a foundation of understanding that would become essential for upcoming labs, and understanding how to program in Verilog in general. For the most part, the code was rather simple, as it was mostly just following instructions. There were a few problems, when needing to figure out how to properly create and use certain aspects of the program, that I felt the lab did not properly explain. However, with the exception of those few instances, the lab itself was quite simple. In conclusion, we were shown and given the means to practise Verilog to a point of proper understanding so that we might be able to apply to future labs, and future understandings of more difficult concepts to be discussed, regarding Verilog.

EQUIPMENT

-laptop

-java eclipse

-putty

-xming

-gedit

METHODS/PROCEDURES

For the most part, the lab was a simple. I simply followed the given instructions, and made little alterations to what was provided. The only places I found myself needing to test out a few options was when actually creating a register of specific sizes in the proper format, and in program 3, when we were to use the assign function. As for programs 6 and 9, where we were to create a Verilog circuit of the diagram provided, I also faced little to no problem. Program 6 and 9 followed the same pseudocode, as program 6 was essentially a less complicated version of program 9. For both of these programs, I simply started by creating the input and output registers, and then doing the logical operations, being And, Or, etc. as I followed the circuit and used these logical operations, I continuously added more wires to accommodate for the resultants, and to signify the actual wiring of the visual circuit given. At the end I simply assigned the outputs to the values of the wires attached, and then ran exhaustive testing to ensure the outputs were as they were supposed to be.

RESULTS

The results shown below are comprised of both the example inputs given for programs, as well as my own inputs for instances where we were not provided an input. Input is only provided for programs that specifically needed input or were given input

LABK1

input: no input. A register was created and the register's value was changed during the program.

output:

time = 0, x = 11111111111111110000000000000000

time = 0, x = 00000000000000000000000000000000

time = 0, x = 00000000000000000000000000000010

one = 0, two = 10, three = 010

LABK2

input: x = 5

Other registers were created and the registers' values were changed during the program.

output:

0: x=x y=x z=x

10: x=5 y=x z=x

14: x=6 y=x z=x

20: x=6 y=7 z=x

21: x=7 y=7 z=x

28: x=8 y=7 z=x

30: x=8 y=7 z=8

LABK3

input: a = 1

b = 2

output:

a=1 b=1 z=0

LABK4

input: No specified input. For this program, exhaustive testing was done

output:

a=0 b=0 z=0

a=0 b=1 z=0

a=1 b=0 z=1

a=1 b=1 z=0

LABK5

input: No specified input. For this program, exhaustive testing was done

output:

PASS: a=0 b=0 z=0

PASS: a=0 b=1 z=0

PASS: a=1 b=0 z=1

PASS: a=1 b=1 z=0

LABK6

input: a = 1

b = 0

c = 0

output:

a=1 b=0 c=0 z=1

LABK7

input: a = 0

b = 0

c = 0

output:

a=0 b=0 c=0 z=0

LABK8

input: a = 1

b = 1

c = 1

output:

PASS: a=0 b=0 c=0 z=0

LABK9

input: no specific input. Exhaustive testing was used in this program.

output:

PASS: a=0 b=0 Cin=0 Cout=0 z=0

PASS: a=0 b=0 Cin=1 Cout=0 z=1

PASS: a=0 b=1 Cin=0 Cout=0 z=1

PASS: a=0 b=1 Cin=1 Cout=1 z=0

PASS: a=1 b=0 Cin=0 Cout=0 z=1

PASS: a=1 b=0 Cin=1 Cout=1 z=0

PASS: a=1 b=1 Cin=0 Cout=1 z=0

PASS: a=1 b=1 Cin=1 Cout=1 z=1

DISCUSSION

The programs in this lab were rather easy. This was expected, as it was simply an introductory lab to Verilog and its many processes. However, there was one problem I faced, that I still do not have a perfect grasp on. That was using the assign key. The explanation given in the lab, I felt, was insufficient in providing a good understanding of how to properly use it. I am still not entirely certain as to when and why we use it. I do understand that we use it for the sake of proficiency and better readability and understanding. However, I do not understand the proper format to use it in, or the wires that should be assigned, or when said variables should be assigned. The only other slight problem I encountered was when I was creating registers in program 1, or more specifically, the proper format to create single bit registers. However, I quickly figured out a way that worked. In addition to this, my guess was reaffirmed in program 2 when it was properly defined as to how to create a single bit register.

LABK1

The output indicates that all statements were executed at the same time, time 0. Statements in an initial block are executed one after the other but each still takes 0 time to execute.

LABK2

Again, because of the statements in the result, and because there is no time delay specified at any point at the beginning of the lab, the beginning code all takes 0 time to execute and, as explained in lab K1 of the lab itself, are sequentially executed. The reason for the delay of #5 causing the program to revert to its original output is because of the delays itself. Because all the variables are defined by the time execution time 15 has been reached, no change is made. Also, because time 15 executes two operations at once, being to change the x value, and to assign the z variable, the x value does not take precedence in the programs computation, but rather, both variations are executed, independently. Had x been executed at an earlier time, or z at a later time, the value of z would have been different. When changing the delay to #25 we get the following resultants

10: x=5 y=x z=x

20: x=5 y=6 z=x

30: x=6 y=6 z=7

When changing the delay to #35, we get the following result.

10: x=5 y=x z=x

20: x=5 y=6 z=x

30: x=5 y=6 z=7

LABK3

The reason the program was unable to capture the output was because of the lack of a time delay. Because there was no time delay, all instructions are being run at the exact same time. Therefore, because no piece of code takes precedence, the output value is not assigned a variable before the output is printed out. This is easily corrected by simply causing a time delay in the display itself, so it will print out time x after the rest of the code has processed. Upon doing this, the output should be displayed properly and as expected.

LABK4

Please refer to the labK4 appendix section. This verifies that all possible combinations used in the testing did output properly and pass all tests. We wouldn’t be able to use a and b directly as loop counters. This is because a and b are registers of a fixed size, being 1. This means that they are only capable of returning a value of 1 or 0. In other words, they cannot even reach 2, or any other value necessary to end the loop, which would be, for obvious reasons, problematic. Thus, the only option is to assign actual integer values for the counters.

LABK5

The testing works as it is supposed to. This is because of the oracle defined in the coding. Defining an oracle seems to not only reduce the amount of code needed, but ensure whether or not the coding outputs as its supposed to, which is detrimental to making the program itself work.

LABK6

As shown in this circuit, if the value of c is 0, then the value of b becomes arbitrarily unneeded and useless. This is a result of what the input c does in the circuit. As seen in the circuit, the bottom AND statement between b and c will always result in 0, whenever c has an input of 0. Therefore whatever value b is is irrelevant. More so, because the upper AND statement turns c into a NOT, this means that c will always enter this AND statement as a 1 instead of a 0. Thus, because the connecting operation is the or operation, all result lies on input a. If it is 0, the output becomes 0, and the OR statement outputs a 0 and likewise, if the input is 1, the output of the AND statement becomes a 1, and the OR statement results in a 1.

LABK7

One issue that arises is if the user does not insert all of the required input variables. If the user even forgets 1 required field, the resultant will be undefined. This is because the input for the given variable whose value is not specified can still exist in verilog. However, when this input is needed to produce and output, that about will also not result in a defined value. Normally, in a programming language like java, not defining the variable would result in a run time error. However, Verilog does not force a variable to be defined when making the program. As an example, this is the output, should we not assign the variable a.

a=x b=1 c=0 z=x

As you can see, the variable a is not assigned to any value, but the computation still goes on. However, because the output is also reliant on the input of a, it too is also not given a proper output, as the value cannot be defined. It is possible to detect whether or not each value is properly defined. This is by using a few if statements. In each if statement, we simply check to see whether the user has passed a value, and more importantly, if the value has been assigned to either a 1 or a 0. If there exists a value that has not been defined, then we simply send out a message stating that the user did not provide valid input for the program.

LABK8

The program outputs as its supposed to. For this program, i used exhaustive testing to ensure that each and every output produces the correct output. Instead of using the oracle, i simply referred to the basis of the input c, as described above under LABK6. First, i checked to see what the value of c was. If the value of c at input was 0, i simply made the resultant the value of the input a. If this was not the case, i just used a set of if statements to replicate the circuit given, and produce the same output as provided by the oracle.

CONCLUSION

In conclusion, the lab was very straight forward, and did not provide much challenge in terms of understanding, or implementation. I achieved everything I sought to learn with this lab, and easily overcame all but 1 issue that arose while making the programs, all while learning about the basics of Verilog, the many functions and uses it provides, and its many similarities to java programming. I also learned how to implement circuits, and do a variety of different methods to perform tests, giving me knowledge and a basis for understanding the future labs regarding Verilog and its many processes.

APPENDIX

module labK1;

//reg [31:0] x; //a 32 bit register

reg [31:0] x = 32'hffff0000;

reg one;

reg [1:0] two;

reg [2:0] three;

initial

begin

$display("time = %5d, x = %b", $time, x);

x = 0;

$display("time = %5d, x = %b", $time, x);

x = x + 2;

$display("time = %5d, x = %b", $time, x);

one = &x; //and reduction

two = x[1:0]; //part-select

three = {one, two}; //concatenate

$display("one = %b, two = %b, three = %b", one, two, three);

$finish;

end

endmodule

module labK2;

reg [31:0] x, y, z;

initial

begin

#10 x = 5;

//$display("%2d: x=%1d y=%1d z=%1d", $time, x, y, z);

#10 y = x + 1;

//$display("%2d: x=%1d y=%1d z=%1d", $time, x, y, z);

#10 z = y + 1;

//$display("%2d: x=%1d y=%1d z=%1d", $time, x, y, z);

#1 $finish;

end

initial

$monitor("%2d: x=%1d y=%1d z=%1d", $time, x, y, z);

/\*initial

begin

repeat (4)

#7 x = x + 1;

end

\*/

always

#7 x = x + 1;

endmodule

module labK3;

reg a, b; //register without size means 1 ibt

wire notOutput, lowerInput, tmp, z;

not (notOutput, b);

and (z, a, lowerInput);

assign lowerInput = notOutput;

initial

begin

a = 1; b = 1;

#5 $display("a=%b b=%b z=%b", a, b, z);

$finish;

end

endmodule

module labK4;

reg a, b; //register without size means 1 ibt

wire z, tmp;

integer i, j;

not (tmp, b);

and (z, a, tmp);

initial

begin

for(i = 0; i < 2; i = i + 1)

begin

for(j = 0; j < 2; j = j + 1)

begin

a = i; b = j;

#1 $display("a=%b b=%b z=%b", a, b, z);

end

end

$finish;

end

endmodule

module labK5;

reg a, b, expect; //register without size means 1 ibt

wire z, tmp;

integer i, j;

not (tmp, b);

and (z, a, tmp);

initial

begin

for(i = 0; i < 2; i = i + 1)

begin

for(j = 0; j < 2; j = j + 1)

begin

a = i; b = j;

expect = i & ~b;

#1; //wait for z

if (expect == z)

$display("PASS: a=%b b=%b z=%b", a, b, z);

else

$display("FAIL: a=%b b=%b, z=%b", a, b, z);

end

end

$display(tmp);

$finish;

end

endmodule

module labK6;

reg a, b, c;

wire AandNotC, BandC, z, temp, expect,temp2, cNot;

integer i, j, k;

not (temp, c);

and(AandNotC, a, temp);

and(BandC, b, c);

or(z, AandNotC, BandC);

//assign AandNotC = temp;

//and(temp, b, c);

//assign BandC = temp;

//or(temp, AandC, BandC);

//assign z = temp;

initial

begin

a = 1; b = 0; c = 0;

//expect = ...

//wait for z

//if (expect == z)

#1 $display(" a=%b b=%b c=%b z=%b", a, b, c, z);

//else

//$display("FAIL: a=%b b=%b, z=%b", a, b, z)

$finish;

end

endmodule

module labK7;

reg a, b, c;

wire AandNotC, BandC, z, temp, expect,temp2, cNot;

integer i, j, k;

reg flag;

not (temp, c);

and(AandNotC, a, temp);

and(BandC, b, c);

or(z, AandNotC, BandC);

//assign AandNotC = temp;

//and(temp, b, c);

//assign BandC = temp;

//or(temp, AandC, BandC);

//assign z = temp;

initial

begin

flag = $value$plusargs("a=%b", a);

flag = $value$plusargs("b=%b", b);

flag = $value$plusargs("c=%b", c);

//look up how to use flag to detec if an argument is missing

//expect = ...

//wait for z

//if (expect == z)

#1 $display(" a=%b b=%b c=%b z=%b", a, b, c, z);

//else

//$display("FAIL: a=%b b=%b, z=%b", a, b, z)

$finish;

end

endmodule

module labK8;

reg a, b, c, expect, expect2;

wire AandNotC, BandC, z, temp, temp2, cNot;

integer i, j, k;

reg flag;

not (temp, c);

and(AandNotC, a, temp);

and(BandC, b, c);

or(z, AandNotC, BandC);

//assign AandNotC = temp;

//and(temp, b, c);

//assign BandC = temp;

//or(temp, AandC, BandC);

//assign z = temp;

initial

begin

flag = $value$plusargs("a=%b", a);

flag = $value$plusargs("b=%b", b);

flag = $value$plusargs("c=%b", c);

//look up how to use flag to detec if an argument is missing

#1 expect = (a & ~c);

#1 expect2 = (b & c);

#5 //wait for z

if (z == expect || z == expect2)

$display("PASS: a=%b b=%b c=%b z=%b", a, b, c, z);

else

$display("FAIL: a=%b b=%b, z=%b", a, b, z);

$finish;

end

endmodule

module labK9;

reg a, b, cIN;

reg[1:0] expect;

wire z, cOUT, aXORb, aANDb, temp;

integer i, j, k;

xor(aXORb, a, b);

xor(z, cIN, aXORb);

and(aANDb, a, b);

and(temp, cIN, aXORb);

or(cOUT, temp, aANDb);

initial

begin

for(i = 0; i < 2; i = i + 1)

begin

for(j = 0; j < 2; j = j + 1)

begin

for(k = 0; k < 2; k = k + 1)

begin

#1 a = i; b = j; cIN = k;

#1 expect = a + b + cIN;

#1; //wait for z

if(expect[0] === z && expect[1] === cOUT)

#5 $display("PASS: a=%b b=%b Cin=%b Cout=%b z=%b", a, b, cIN, cOUT, z);

else

#5 $display("FAIL: a=%b b=%b Cin=%b Cout=%b z=%b", a, b, cIN, cOUT, z);

end

end

end

$finish;

end

endmodule