COMPUTER ORGANIZATION
EECS 2021

LAB “K” REPORT
BASIC VERILOG PROGRAMMING

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.

X ____________________

Camillo John (CJ) D’Alimonte
212754396
Lab Section 2
11/04/14
Professor Aboelaze

Abstract:

This laboratory/experiment introduced me to the basics of Verilog programming. I will learn how to declare and initialize registers and how to format output. I will learn how to iterate through my code and how to test if the circuits are working correctly. I will be able to design Verilog code from a given circuit diagram. By the end of the lab, I will have learned some of the simple programming concepts of the Verilog language.

Equipment Used:

This lab was done through the UNIX environment at the Lassonde Lab. I used the pico editor in UNIX to write my code and I used the built-in Verilog compiler to run and test my programs.

Methods/Procedures:

Pseudo Code is provided for each of the 9 experiments. It combines the code provided to us in the lab manual as well as code that I wrote myself.

Pseudo Code LabK1	reg [31:0] x;
	reg [0:0] one;
	reg [1:0] two;
	reg [2:0] three;
	begin
	    $display(x);
	    x → 32'hffff0000;
	    $display(x);
	    x → x + 2;
	    $display(x);
	    One→ &x;
	    two → x[1:0];
	    three → {one, two};
	    $display(one, two, three);
	end

Pseudo Code LabK2	reg [31:0] x, y, z;
	begin
	    #10 x → 5;
	    $display (x, y, z);
	    #10 y → x + 1;
	    $display x, y, z);
	    #10 z → y + 1;
	    $display (x, y, z);
	    #1 $finish;
	end

	always
	    #7 x → x + 1; // changes for each delay
	initial
	    $monitor (x, y, z);

Pseudo Code LabK3	reg a, b;
	wire notOutput, lowerInput, tmp, z;
	not my_not(notOutput, b);
	and my_and(z, a, lowerInput);
	assign lowerInput → notOutput;
	initial
	begin
	    a → 1; b → 1;
	    #1 $display (a, b, z);
	    $finish;
	end

Pseudo Code LabK4	reg a, b;
	wire tmp, z;
	integer i, j;
	not my_not(tmp, b);
	and my_and(z, a, tmp);
	initial
	begin
	    for (i → 0; i < 2; i → i + 1) //loop1
	    begin
	        for (j → 0; j < 2; j → j + 1) //loop2
	        begin
	            a → i; b → j;
	            #1 $display (a, b, z);
	        end
	    end
	    $finish;
	end

Pseudo Code LabK5	reg a, b, expect;
	wire tmp, z;
	integer i, j;
	not my_not(tmp, b);
	and my_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; // wait for z
            expect → a & ~b;
            if (expect → z)
                #1 $display ("PASS: a, b, z);
            else
                #1 $display ("FAIL: a, b, z);
        end
    end
    $finish;
end
```

Pseudo Code LabK6

```
reg a, b, c, expect;
wire not_c, a_and_not_c, c_and_b, z, not_c_out, c_and_b_out,
a_and_not_c_out;
integer i, j, k;
not (not_c, c);
and (a_and_not_c, a, not_c_out);
and (c_and_b, c, b);
or (z, a_and_not_c_out, c_and_b_out);
assign not_c_out → not_c;
assign c_and_b_out → c_and_b;
assign a_and_not_c_out → a_and_not_c;
```

Pseudo Code LabK7

```
reg a, b, c, flag, expect;
wire not_c, a_and_not_c, c_and_b, z;
not (not_c, c);
and (a_and_not_c, a, not_c);
and (c_and_b, c, b);
or (z, a_and_not_c, c_and_b);
initial
begin
    flag → $value$plusargs("a=%b", a);
    flag → flag & $value$plusargs("b=%b", b);
    flag → flag & $value$plusargs("c=%b", c);
    if (flag → 0)
        $display ("Argument missing!");
    #1; // wait for z
    expect → (~c & a) | (c & b);
    if (expect → z)
        #1 $display ("PASS: a, b, c, z);
        else
                #1 $display ("FAIL: a, b, c, z);
        $finish;
end
```

Pseudo Code LabK8

```
reg a, b, c, flag, expect;
wire not_c, a_and_not_c, c_and_b, z;
integer i, j, k;
not (not_c, c);
and (a_and_not_c, a, not_c);
and (c_and_b, c, b);
or (z, a_and_not_c, c_and_b);
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
                                a → i; b → j; c → k;
                                #1; // wait for z
                                expect → c ? b : a;
                                if (expect → z)
                                        #1 $display ("PASS:  a, b, c, z);
                                else
                                        #1 $display ("FAIL: a, b, c, z);
                        end
                end
        end
        $finish;
end
```

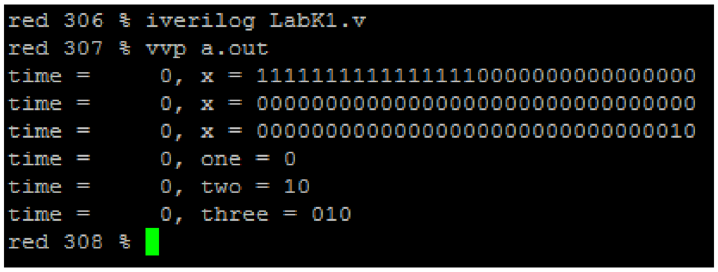
Pseudo Code LabK9

```
reg a, b, cin, flag;
reg[1:0] expect;
wire z, cout;
wire a_xor_b, a_and_b, axb_and_cin;
integer i, j, k;
xor (a_xor_b, a, b);
and (a_and_b, a, b);
and (axb_and_cin, a_xor_b, cin);
or (cout, axb_and_cin, a_and_b);
xor (z, cin, a_xor_b);
```

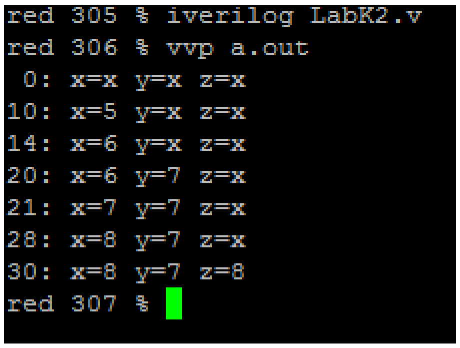
	initial
	begin
	    for (i → 0; i < 2; i → i + 1)
	    begin
	        for (j → 0; j < 2; j → j + 1)
	        begin
	            for (k → 0; k < 2; k → k + 1)
	            begin
	                a → i; b → j; cin → k;
	                #1;
	                expect → a + b + cin;
	                if (expect[0] →z && expect[1] → cout)
	                    $display ("PASS: a, b, cin, z, cout);
	            end
	        end
	    end
	    $finish;
	end

Results:

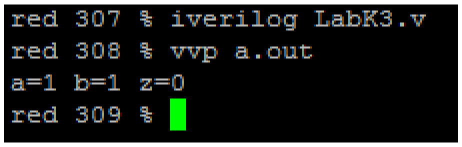
The following terminal screen-shots provide a view of the different results I had during each of the 9 experiments.



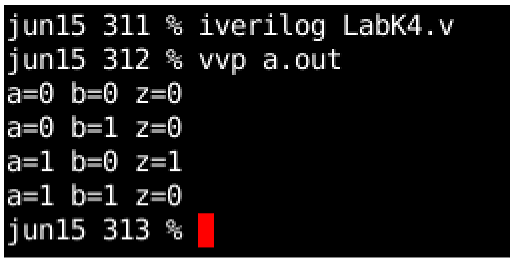
```
red 306 % iverilog LabK1.v
red 307 % vvp a.out
time =      0, x = 11111111111111110000000000000000
time =      0, x = 00000000000000000000000000000000
time =      0, x = 00000000000000000000000000000010
time =      0, one = 0
time =      0, two = 10
time =      0, three = 010
red 308 %
```



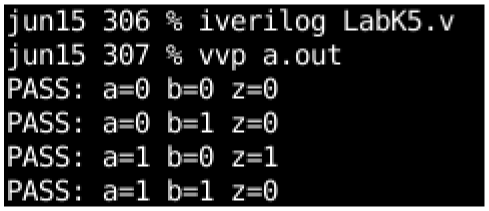
```
red 305 % iverilog LabK2.v
red 306 % vvp a.out
 0: x=x y=x z=x
10: x=5 y=x z=x
14: x=6 y=x z=x
20: x=6 y=7 z=x
21: x=7 y=7 z=x
28: x=8 y=7 z=x
30: x=8 y=7 z=8
red 307 %
```



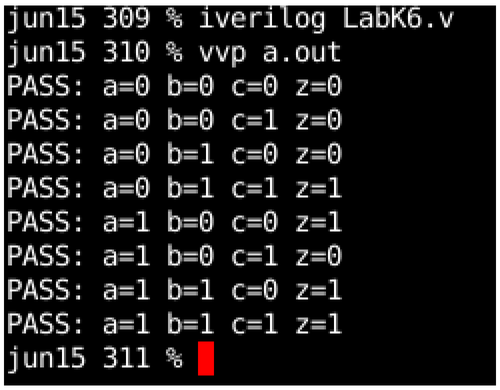
```
red 307 % iverilog LabK3.v
red 308 % vvp a.out
a=1 b=1 z=0
red 309 %
```



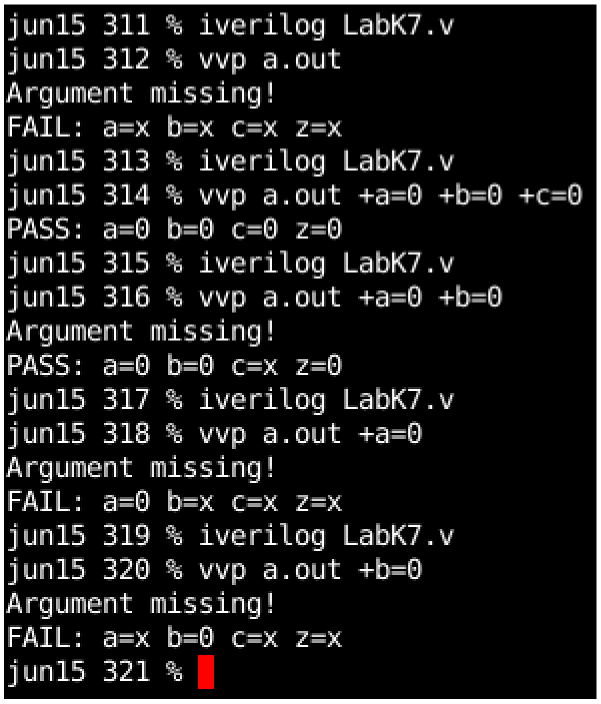
```
jun15 311 % iverilog LabK4.v
jun15 312 % vvp a.out
a=0 b=0 z=0
a=0 b=1 z=0
a=1 b=0 z=1
a=1 b=1 z=0
jun15 313 %
```



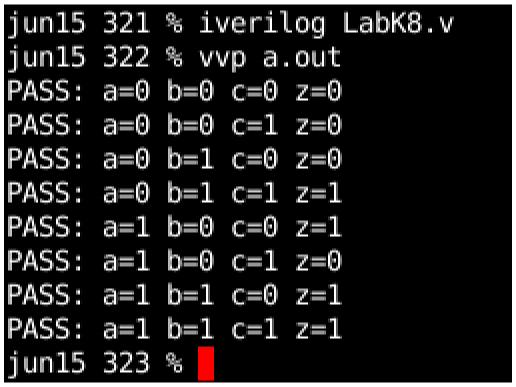
```
jun15 306 % iverilog LabK5.v
jun15 307 % vvp a.out
PASS: a=0 b=0 z=0
PASS: a=0 b=1 z=0
PASS: a=1 b=0 z=1
PASS: a=1 b=1 z=0
```



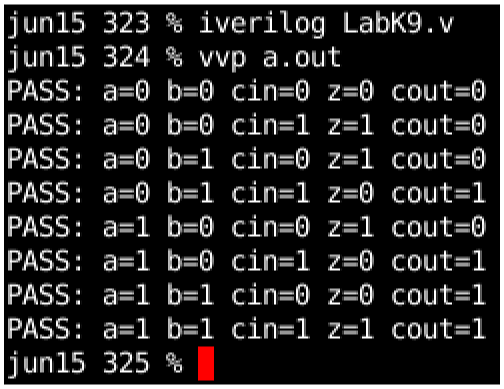
```
jun15 309 % iverilog LabK6.v
jun15 310 % vvp a.out
PASS: a=0 b=0 c=0 z=0
PASS: a=0 b=0 c=1 z=0
PASS: a=0 b=1 c=0 z=0
PASS: a=0 b=1 c=1 z=1
PASS: a=1 b=0 c=0 z=1
PASS: a=1 b=0 c=1 z=0
PASS: a=1 b=1 c=0 z=1
PASS: a=1 b=1 c=1 z=1
jun15 311 %
```



```
jun15 311 % iverilog LabK7.v
jun15 312 % vvp a.out
Argument missing!
FAIL: a=x b=x c=x z=x
jun15 313 % iverilog LabK7.v
jun15 314 % vvp a.out +a=0 +b=0 +c=0
PASS: a=0 b=0 c=0 z=0
jun15 315 % iverilog LabK7.v
jun15 316 % vvp a.out +a=0 +b=0
Argument missing!
PASS: a=0 b=0 c=x z=0
jun15 317 % iverilog LabK7.v
jun15 318 % vvp a.out +a=0
Argument missing!
FAIL: a=0 b=x c=x z=x
jun15 319 % iverilog LabK7.v
jun15 320 % vvp a.out +b=0
Argument missing!
FAIL: a=x b=0 c=x z=x
jun15 321 %
```



```
jun15 321 % iverilog LabK8.v
jun15 322 % vvp a.out
PASS: a=0 b=0 c=0 z=0
PASS: a=0 b=0 c=1 z=0
PASS: a=0 b=1 c=0 z=0
PASS: a=0 b=1 c=1 z=1
PASS: a=1 b=0 c=0 z=1
PASS: a=1 b=0 c=1 z=0
PASS: a=1 b=1 c=0 z=1
PASS: a=1 b=1 c=1 z=1
jun15 323 %
```



```
jun15 323 % iverilog LabK9.v
jun15 324 % vvp a.out
PASS: a=0 b=0 cin=0 z=0 cout=0
PASS: a=0 b=0 cin=1 z=1 cout=0
PASS: a=0 b=1 cin=0 z=1 cout=0
PASS: a=0 b=1 cin=1 z=0 cout=1
PASS: a=1 b=0 cin=0 z=1 cout=0
PASS: a=1 b=0 cin=1 z=0 cout=1
PASS: a=1 b=1 cin=0 z=0 cout=1
PASS: a=1 b=1 cin=1 z=1 cout=1
jun15 325 %
```

Discussion:

My analysis of the results is separated for each experiment.

LabK1:

This lab introduced me to the concepts of making simple Verilog programs and learning how to run and compile Verilog code. It also introduced me to basic functions such as formatting output and declaring and initializing registers. Formatting output turns out to be very similar to the Java printf() function. I was introduced to three new functions during this lab. They were the reduction, part-select, and concatenate functions that work on binary strings. The reduction functions returns the logical AND of all the bits in the string. The part-select returns a specified bit inside the string while the concatenation functions combines two strings together (string1 and string2).

LabK2:

This lab introduced me to running multiple threads (initial blocks) in parallel. This was my first experience in running two programs in parallel at the same time together and seeing how to get one thread to wait while the other thread executed. The timing turned out to be the most important aspect of this lab. Syntactically, I learned that there are three ways to iterate in Verilog: repeat (n), for loop, and always. Multi-Line comments can be written like Java using the /* */. The $monitor command can be used to output the value of the variables every time a value has been changed. To answer the questions:

Q15: The output revered back to its original value because it was executed before the main thread. As a result, it's overwritten. Thus, the value = 5.

Q18: The value increments every 7 units. When it reaches 14, it increments to 6, therefore x = 6 and y = 7 at time 20. X increments two more times to time 30.

LabK3:

This lab introduced me to building my first circuit using a NOT and AND gate which allowed us to create two registers, two wires and two gates. I learned that there are two ways to represent wires in Verilog; a direct way and an explicit way. The direct way consists of: not(tmp, b); and(z, a, tmp); while the explicit way consists of: not(notOutput, b); and(z, a, lowerInput); assign lowerInput = notOutput. The direct way uses only one wire variable to connect two circuit elements, while the explicit way uses two wire variables to connect two different circuit elements. I needed a time delay to display the contents of the wires before the display functions is called, otherwise the value of the wire will be x. I assume this is because the current needs time to travel through the circuit before the output for the wires is determined.

LabK4:

This lab introduced me to the three different types of testing in Verilog: exhaustive, sampled, and random testing. Only exhaustive testing was used to test the LabK3 circuit. The circuit passed all the tests I ran. To answer question 30, it would be incorrect to use a and b directly because they are only one bit in size. When i and j exit their loops they hold the decimal value of 2, which cannot be stored by bit size one. If a and b were used, they would end up creating an infinite loop.

LabK5:

This lab introduced me to the concept of creating formulas for gates to find their expected values in the circuit. I would then be able to compare the expected value to the actual value in order to print out whether the circuit passes or fails the test(s).

LabK6:

This lab introduced me to converting a circuit diagram into its equivalent Verilog code from scratch. There were no issues experienced in this lab. My program ran correctly for the test case.

If the C wire has no current, then the bottom AND gate will always output no current (0) as well. The top AND gate will drive the output of the OR gate. The C wire travels through the NOT gate, and thus, its input to the AND gate will always have current (1). The current of the Z wire will always be the same as the A wire.

LabK7:

This lab introduced me to getting input from the user using flags. The flag will be set to 0 if the user neglects to input anything and it will be set to 1 if the user specifies an input. This strategy was used to determine if the user forgot to input any arguments. If the user forgets to input the value of a variable, the variable will be set to x. All in all, this lab was pretty straightforward. To answer the questions asked: the program will output “Missing Arguments” and set any undefined variables as x if there is a missing argument.

LabK8:

This lab asked me to test the LabK6 circuit. The circuit passed all the tests for both exhaustive oracles. The first oracle used a larger formula based approach with an if statement, whereas the second oracle used a smaller formula to decide the expected output.

LabK9:

This lab asked me to write the Verilog code for a given circuit. Question 51 was a little difficult but after figuring out the correct code to write, everything went well and the circuit passed all the tests I ran.

Conclusion:

By completing this lab, I was able to begin learning the basics behind Verilog programming. I now have a good basis of knowledge in Verilog and I now understand how to make and test different circuits, run code in parallel, get input from users, and use circuits to create code from scratch.

References:

No external references were used for this lab other than the lab material provided to us at the beginning of the lab.

Appendix:

The source code for all 9 experiments is included below.

LabK1:

```
module LabK;

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

reg [0:0] one;

reg [1:0] two;

reg [2:0] three;

initial

begin

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

        x = 32'hffff0000;

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

        x = x + 2;

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

        one = &x;

        two = x[1:0];

        three = {one, two};

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

        $finish;

end

endmodule
```

LabK2:

```
module LabK;

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

//integer i;

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
always
    #7 x = x + 1;
initial
    $monitor ("%2d: x=%1d y=%1d z=%1d", $time, x, y, z);
/*initial
begin
    //repeat (4)
    //      #7 x = x + 1;
    for (i = 0; i < 4; i = i + 1)
        #7 x = x + 1;
end*/
endmodule
```

<u>LabK3:</u>

```
/*module LabK;
reg a, b;       // reg without size means 1-bit
wire tmp, z;
```

```
not my_not(tmp, b);
and my_and(z, a, tmp);
initial
begin
       a = 1; b = 1;
       #1 $display ("a=%b b=%b z=%b", a, b, z);
       $finish;
end
endmodule
*/
module LabK;
reg a, b;       // reg without size means 1-bit
wire notOutput, lowerInput, tmp, z;
not my_not(notOutput, b);
and my_and(z, a, lowerInput);
assign lowerInput = notOutput;
initial
begin
       a = 1; b = 1;
       #1 $display ("a=%b b=%b z=%b", a, b, z);
       $finish;
end
endmodule
```

LabK4:

```
module LabK;
```

```
reg a, b;       // reg without size means 1-bit

wire tmp, z;

integer i, j;

not my_not(tmp, b);

and my_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
```

<u>LabK5:</u>

```
module LabK;

reg a, b, expect;

wire tmp, z;

integer i, j;

not my_not(tmp, b);

and my_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; // wait for z

                        expect = a & ~b;

                        if (expect == z)

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

                        else

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

                end

        end

        $finish;

end

endmodule
```

LabK6:

```
/*module LabK;

reg a, b, c, expect;

wire not_c, a_and_not_c, c_and_b, z;

integer i, j, k;

not (not_c, c);

and (a_and_not_c, a, not_c);
and (c_and_b, c, b);

or (z, a_and_not_c, c_and_b);

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

                a = i; b = j; c = k;

                #1; // wait for z

                expect = (~c & a) | (c & b);

                if (expect == z)

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

                else

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

            end

        end

    end

    $finish;

end

endmodule

*/

module LabK;
```

```
reg a, b, c, expect;

wire not_c, a_and_not_c, c_and_b, z, not_c_out, c_and_b_out, a_and_not_c_out;

integer i, j, k;

not (not_c, c);

and (a_and_not_c, a, not_c_out);

and (c_and_b, c, b);

or (z, a_and_not_c_out, c_and_b_out);

assign not_c_out = not_c;

assign c_and_b_out = c_and_b;

assign a_and_not_c_out = a_and_not_c;
```

LabK7:

```
module LabK;

reg a, b, c, flag, expect;

wire not_c, a_and_not_c, c_and_b, z;

not (not_c, c);

and (a_and_not_c, a, not_c);

and (c_and_b, c, b);

or (z, a_and_not_c, c_and_b);

initial

begin

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

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

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

    if (flag == 0)

        $display ("Argument missing!");
    #1; // wait for z
    expect = (~c & a) | (c & b);
    if (expect == z)
        #1 $display ("PASS: a=%b b=%b c=%b z=%b", a, b, c, z);
    else
        #1 $display ("FAIL: a=%b b=%b c=%b z=%b", a, b, c, z);
    $finish;
end
endmodule
```

<u>LabK8:</u>

```
module LabK;
reg a, b, c, flag, expect;
wire not_c, a_and_not_c, c_and_b, z;
integer i, j, k;
not (not_c, c);
and (a_and_not_c, a, not_c);
and (c_and_b, c, b);
or (z, a_and_not_c, c_and_b);
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
                        a = i; b = j; c = k;
                        #1; // wait for z
                        expect = c ? b : a;
                        if (expect == z)
                            #1 $display ("PASS: a=%b b=%b c=%b z=%b", a, b, c, z);
                        else
                            #1 $display ("FAIL: a=%b b=%b c=%b z=%b", a, b, c, z);
                    end
                end
            end
            $finish;
        end
endmodule
```

<u>LabK9:</u>

```
module LabK;
reg a, b, cin, flag;
reg[1:0] expect;
wire z, cout;
wire a_xor_b, a_and_b, axb_and_cin;
integer i, j, k;
xor (a_xor_b, a, b);
and (a_and_b, a, b);
and (axb_and_cin, a_xor_b, cin);
or (cout, axb_and_cin, a_and_b);
```

```
xor (z, cin, a_xor_b);

initial

begin

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

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

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

    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

                a = i; b = j; cin = k;

                #1;

                expect = a + b + cin;

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

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

            end

        end

    end

    $finish;

end

endmodule
```