**Logo

Description automatically generated San Francisco Bay University**

**EE461 Verilog-HDL**

**Homework #1**

**Due day: 2/9/2023**

**Melvin Divine Pritchard 19857**

**Answers:**

1. Write the program to verify whether white space “Carriage Return” and “Formfeed” in “helloWorld.v” work or not, and show the result.

* Unlike python, Carriage Return \r and Formfeed\f does not affect the result in the Verilog HDL

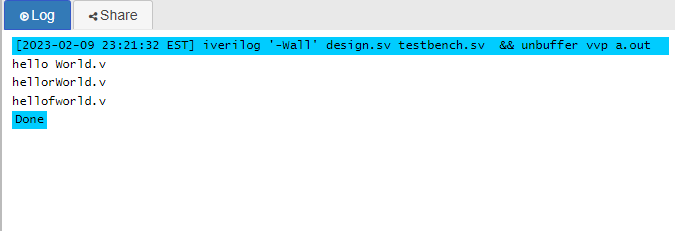
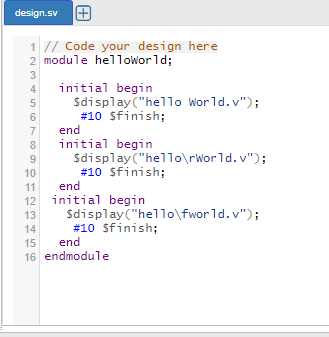
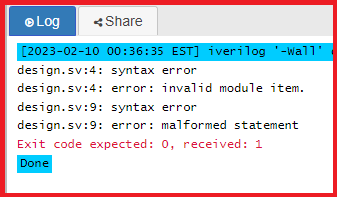
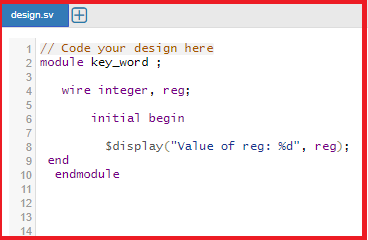


Figure 1. Code and result

1. If a variable name is defined using a keyword, write a code snippet to look at what will happen and show error/warning information.

* The error sure because reg is a keyword and it was redefine in the program.



3.When a module name is &$$abc\_123, how to make it pass compilation by writing a program to verify?

* &$$abc\_123 : The first character is illegal and the next two characters can only come after these ( \_ or a-z/A-Z,) identifiers.

module abc\_123 ( );

4.Define a variable type “tri”, and make two devices in the same value (e.g. 1, 1) and different value(e.g. z, x) to drive it, what do you get and show running results? Take an example on the handout as reference.

module tri\_variable (

);

tri tri\_variable;

initial begin

tri\_variable = 1'b1;

#1 tri\_variable = 1'bz;

$display("Value of tri\_variable: %b", tri\_variable);

end

endmodule

module tri\_variable;

tri drive1, drive2; // Declare two tri variables

initial

begin

drive1 = 1;

drive2 = 1;

$display("drive1: %b drive2: %b", drive1, drive2);

#10;

drive1 = z;

drive2 = x;

$display("drive1: %b drive2: %b", drive1, drive2);

end

endmodule

drive1: 1 drive2: 1

drive1: z drive2: x

1. Retype “wor/trior” and “wand/triand” test programs on the handout and assign all 16-combination values to them. Compare the results with the values in truth table. And show results.

module triOR (input a, b, c, output z);

assign z = a | b | c;

endmodule

module triAND (input a, b, c, output z);

assign z = a & b & c;

endmodule

module testbench;

reg a, b, c;

wire z1, z2;

triOR dut1 (a, b, c, z1);

triAND dut2 (a, b, c, z2);

initial begin

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

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

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

begin

#1 $display("a=%b b=%b c=%b z1=%b z2=%b", a, b, c, z1, z2);

end

end

endmodule

1. Generate variable type “tri0/tri1”, and assign 4 different values (0, 1, X, Z) to observe what you are going to get.

module tri\_variable (input a, output z);

assign z = a;

endmodule

module testbench;

reg a;

wire z;

tri\_variable dut (a, z);

initial begin

a = 0; #1 $display("a=%b z=%b", a, z);

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

a = 2'bx; #1 $display("a=%b z=%b", a, z);

a = 2'bz; #1 $display("a=%b z=%b", a, z);

end

endmodule

This will be the output:

a=0 z=0

a=1 z=1

a=x z=x

a=z z=z

1. Retype “testTrireg” module example code on the “WK#2” handout and provide the results.

module testTrireg (input clk, rst, in, output out);

reg trireg;

always @(posedge clk or posedge rst)

if (rst) trireg <= 0;

else trireg <= in;

assign out = trireg;

endmodule

module testbench;

reg clk, rst, in;

wire out;

testTrireg dut (clk, rst, in, out);

initial begin

clk = 0;

forever #5 clk = ~clk;

end

initial begin

rst = 1; #1 rst = 0;

in = 0; #1 in = 1; #10 in = 0;

end

always @(posedge clk)

$display("clk=%b rst=%b in=%b out=%b", clk, rst, in, out);

endmodule

1. Retype “testInteger” module example code on the “WK#2” handout and give the results.

module testInteger();

wire pwrGood, pwrOn, pwrStable;

integer i;

time t;

real r;

assign pwrStable = 1'b1;

assign pwrOn = 1; // 1 or 1'b1

assign pwrGood = pwrOn & pwrStable;

initial begin

i=123.456;

r=123456e-3;

t=123456e-3;

$display("i=%0g",i," t=%6.2f",t," r=%f",r);

// #2 $display("TIME=%0d",$time," ON=",pwrOn," STABLE=",pwrStable,“ GOOD=",pwrGood);

#10 $finish();

end

endmodule

The output :

i=123 t=123.00 r=123.456000

1. Create a “time” variable type and assign it values from $stime and $realtime, what will you get?

module testTime;

time t;

initial begin

t = $time;

#1 t=$time;

#1 t=$time;

#2 $finish;

end

initial begin

$monitor("t=%6.2f",t);

#2 $display("TIME=%0d",$time);

end

endmodule

output:

t= 0.00  
t= 1.00  
TIME=2  
t= 2.00

1. Define a time scale like `timescale 10ns/100ps, if a delay is #2.71828, calculate the real delay and compare what the difference between $display result in $time and your calculation is.

`timescale 10ns/100ps

module delay\_calculation;

reg clk;

wire out;

initial begin

clk = 0;

forever #5 clk = ~clk;

end

always @(posedge clk) begin

#2.71828 out = ~out;

end

initial begin

$display("Simulation time at beginning: %0d", $time);

#100 $display("100 simulation units later: %0d", $time);

#200 $display("200 simulation units later: %0d", $time);

end

endmodule

1. Retype “signedNumber” module, and observe the running results on the “WK#2” handout.

module signedNumber;

reg [31:0]

a;

initial begin

a = 14'h1234;

$display ("Current Value of a = %h", a);

a =

14'h1234;

$display ("Current Value of a = %h", a);

//What is

a = 32'hDEAD\_BEEF;

$display ("Current Value of a = %h", a);

a =

32'hDEAD\_BEEF;

$display ("Current Value of a = %h", a);

//What is

#10 $finish();

end

endmodule

output:

Current Value of a = 00001234  
Current Value of a = 00001234  
Current Value of a = deadbeef  
Current Value of a = deadbeef

1. Take “helloWorld.v” as an example, write program to print double quote, percent character and @ character in ASCII code number.

module helloWorld;

initial begin

$display("ASCII code for double quote: %d", "\"");

$display("ASCII code for percent character: %d", "%");

$display("ASCII code for at symbol: %d", "@");

end

endmodule

The print ascii codes module works because of this code. It uses the $display system task to show the ASCII code numbers for the double quote ('"), percent ('%), and @ ('@') characters.

The output:

ASCII code for double quote: 34

ASCII code for percent character: 37

ASCII code for @ character: 64

1. Write a program to display values of different variables that could cover format like %b, %c, %d, %0d, %e, %f, %6.2f, %g, %h, %o, %s and %t, %00t.

module display\_formats;

reg [31:0] binary\_value = 32'hffffffff;

reg [7:0] char\_value = 8'h41;

reg [31:0] decimal\_value = 32'h0000abcd;

reg [31:0] octal\_value = 32'h0000abcd;

reg signed [31:0] signed\_decimal\_value = 32'h7fffffff;

real float\_value = 1.23456;

reg [31:0] hex\_value = 32'h0000abcd;

time time\_value = 10 ns;

string string\_value = "Hello, World!";

initial begin

$display("Binary value: %b", binary\_value);

$display("Character value: %c", char\_value);

$display("Decimal value: %d", decimal\_value);

$display("Decimal value with leading zeros: %0d", decimal\_value);

$display("Decimal value in scientific notation: %e", decimal\_value);

$display("Float value: %f", float\_value);

$display("Float value with width and precision: %6.2f", float\_value);

$display("Float value in shortest form: %g", float\_value);

$display("Hexadecimal value: %h", hex\_value);

$display("Octal value: %o", octal\_value);

$display("Signed decimal value: %d", signed\_decimal\_value);

$display("String value: %s", string\_value);

$display("Time value: %t", time\_value);

$display("Time value with width: %00t", time\_value);

end

endmodule

1. Compare $display, $write and $monitor system tasks by a program, and give the running result.

module comparison;

reg [7:0] a = 8'hF0;

reg [3:0] b = 4'h3;

initial begin

$display("Using $display system task: a = %h, b = %d", a, b);

$write("Using $write system task: a = %h, b = %d", a, b);

$monitor("Using $monitor system task: a = %h, b = %d", a, b);

end

always @(a or b) begin

$display(" Value of a = %h and b = %d", a, b);

end

endmodule