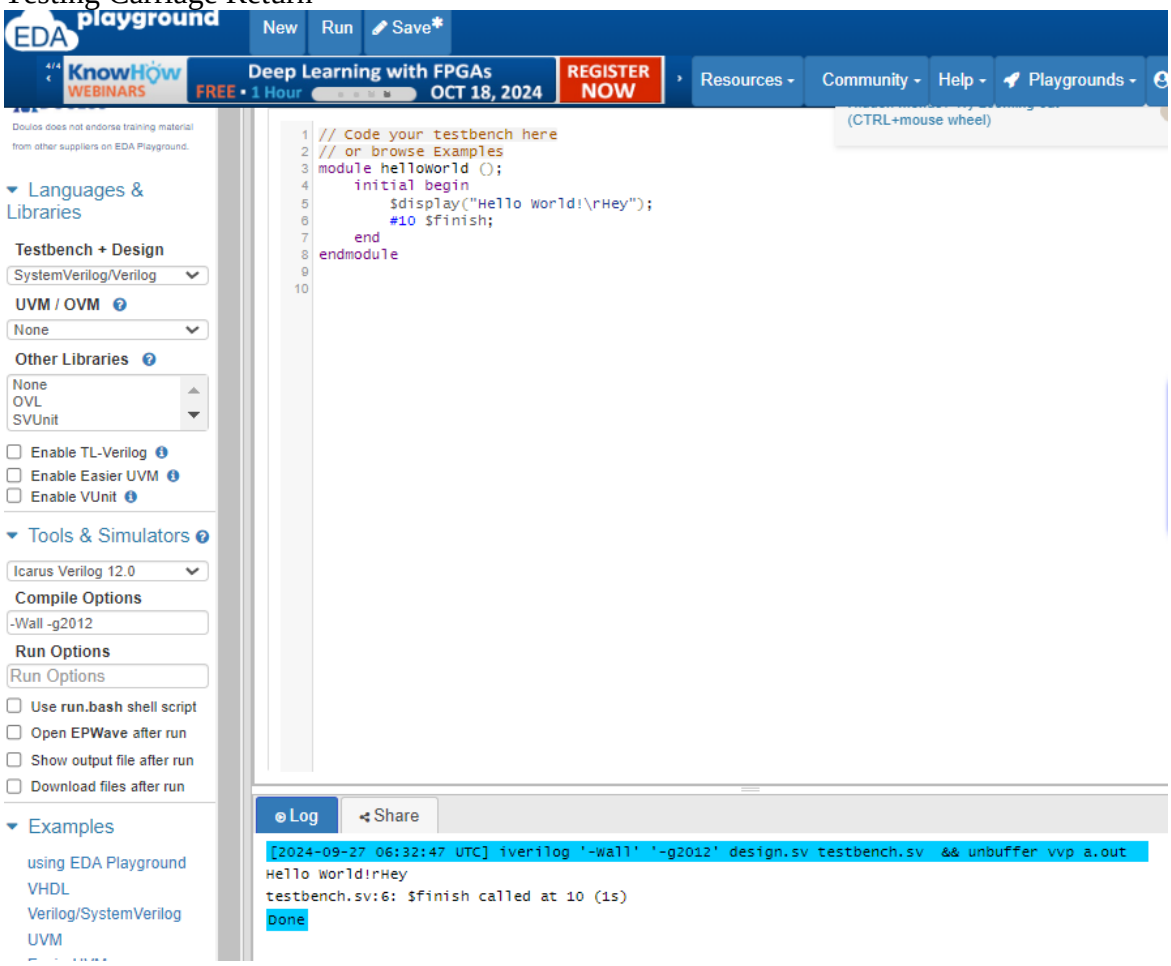
San Francisco Bay University

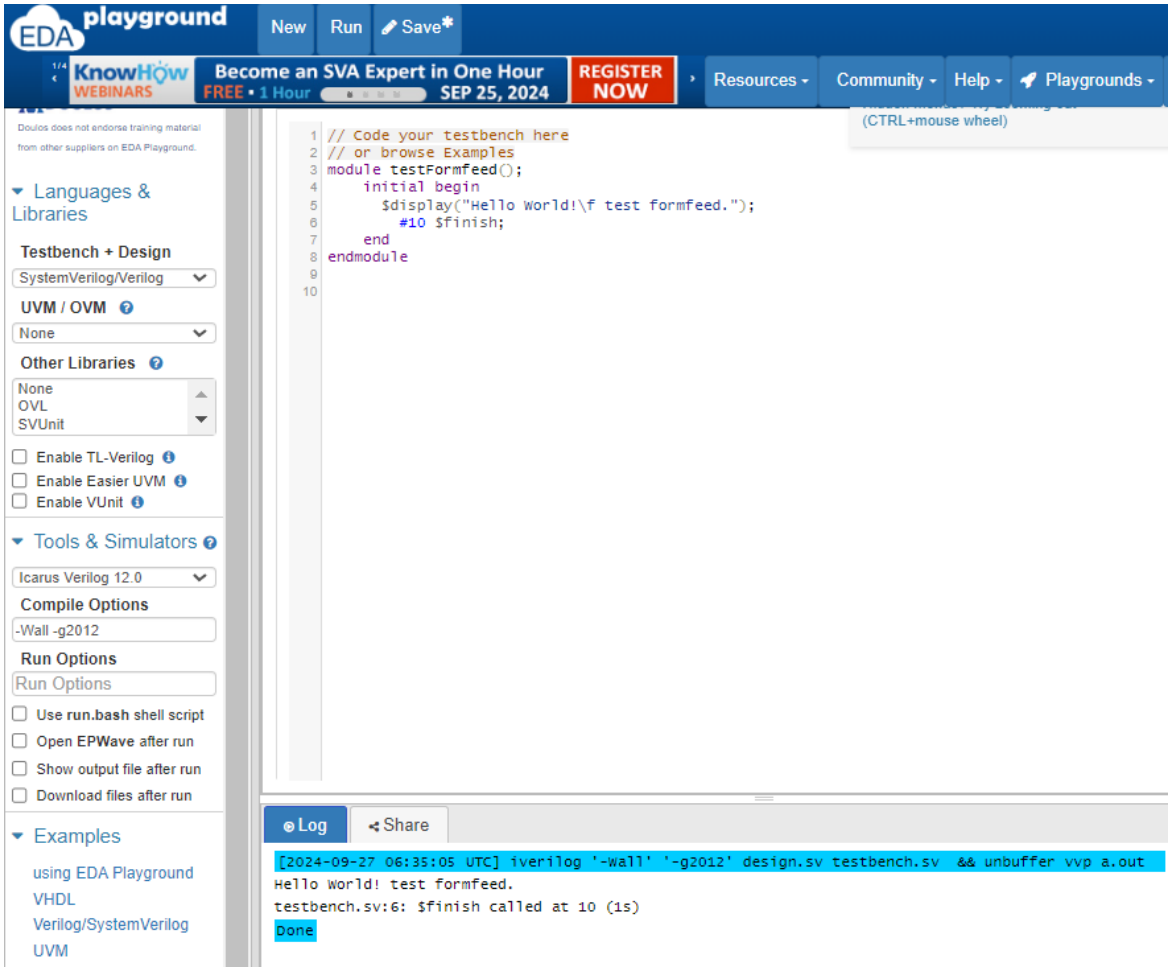
EE461 Verilog-HDL
Homework #1

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

Testing Carriage Return



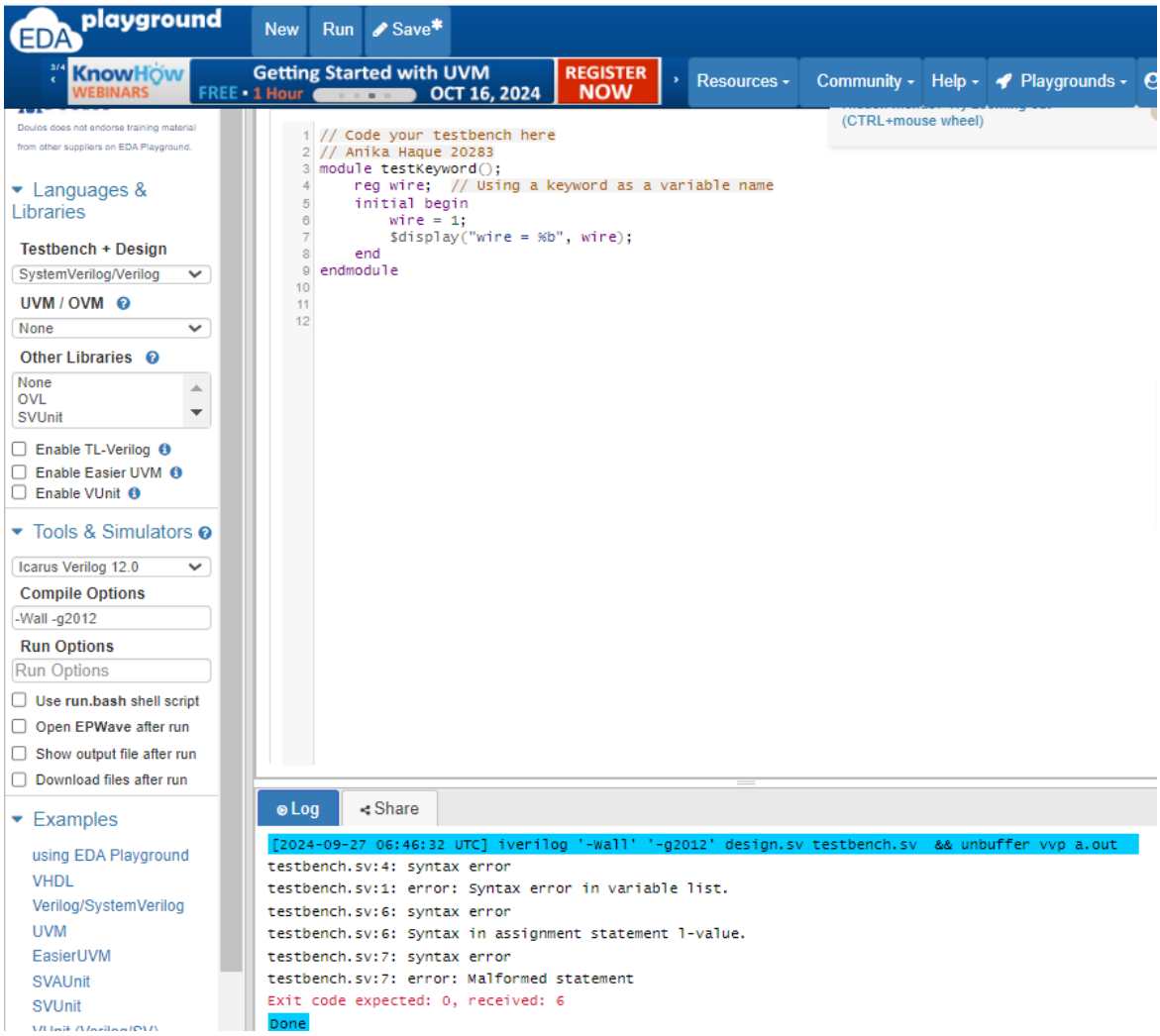
Testing Formfeed



From the 2 screenshots we can see that Verilog does not process Whitespace and treats both Carriage Return and Formfeed as part of the actual string displayed.

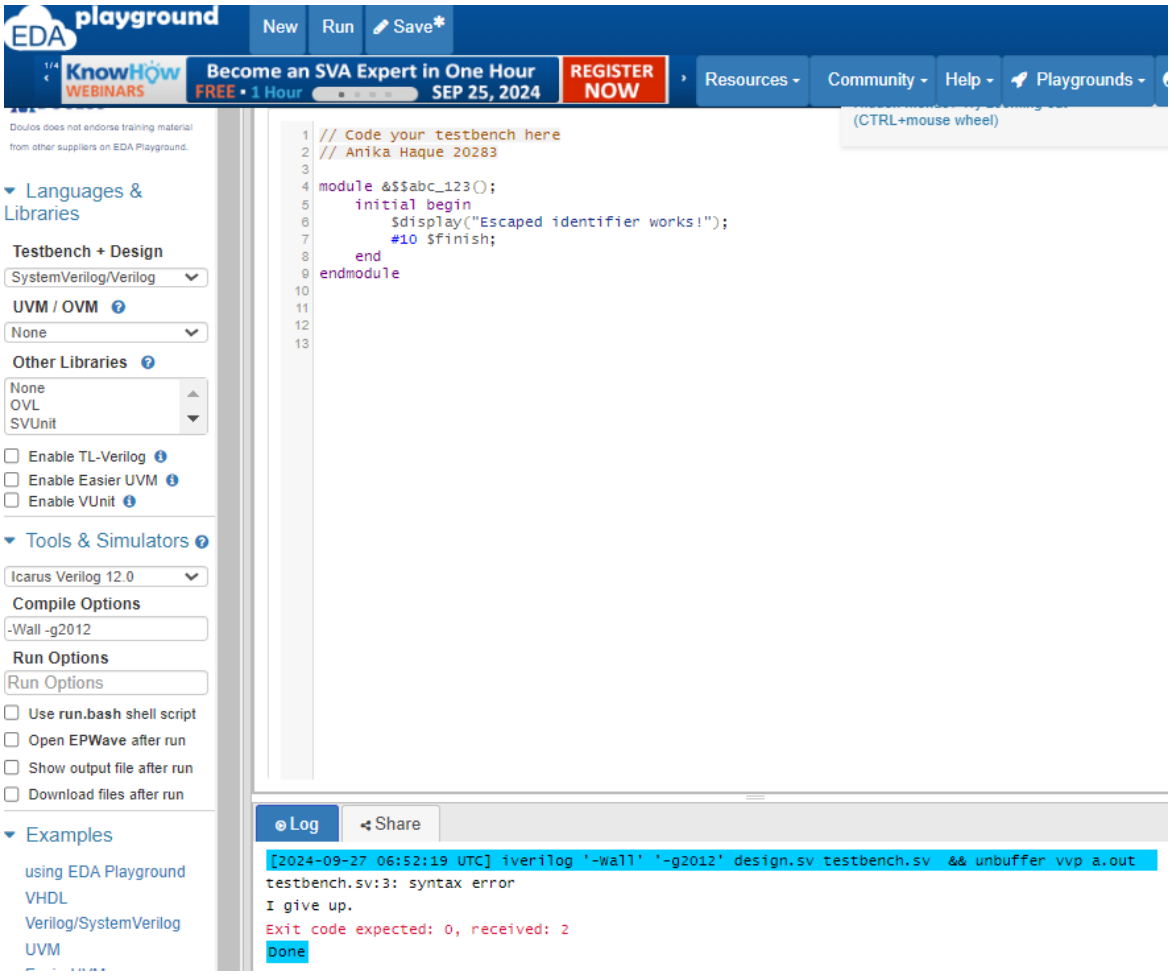
2. If a variable name is defined using a keyword, write a code snippet to look at what will happen and show error/warning information.

Code snippet used to test whether a keyword (like `wire`) can be used as a variable name.

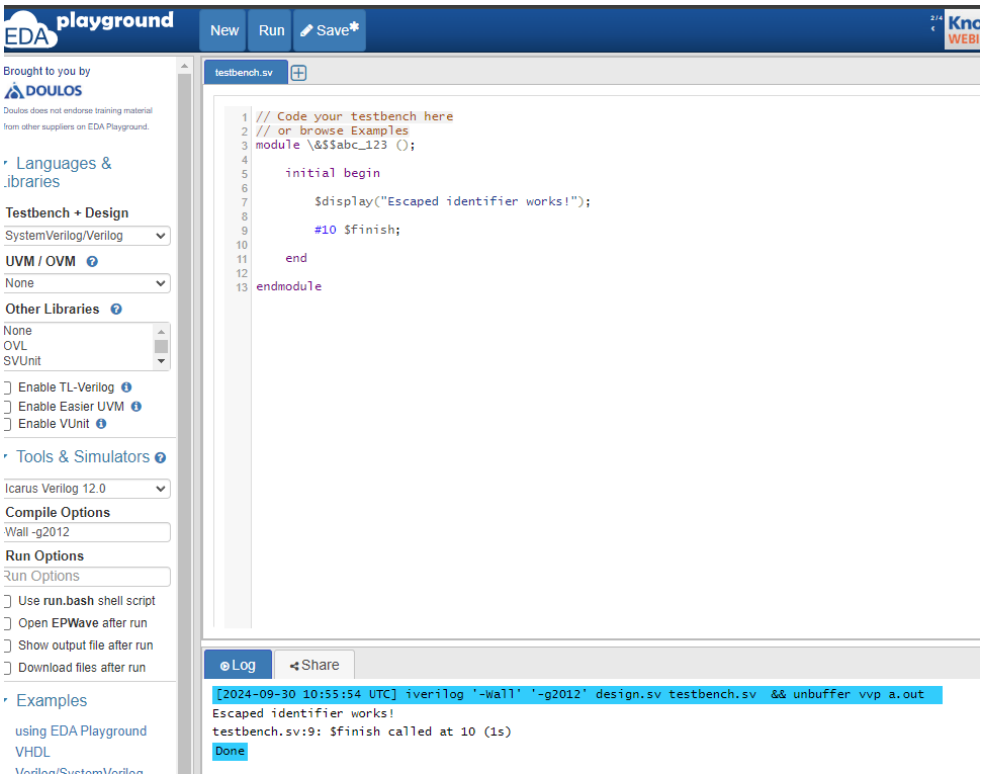


From the screenshot above we can see that using a keyword as a variable name does not work and gives a syntax error in Verilog.

3. When a module name is &$$abc_123, how to make it pass compilation by writing a program to verify?



&$$abc_123 gives syntax error.



As the above screenshot shows, Verilog allows special characters in identifiers when escaped with a backslash (`\`), followed by a space at the end of the name of the module.

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.

A tri wire allows only one active driver at a time. If 2 or more drivers attempt to drive conflicting values onto the same tri wire, the value of the wire becomes unknown (x) because Verilog cannot resolve the conflict. This is shown in screenshot below

```
module triState();
    tri [7:0] bus;            // Declare a tri-state 8-bit bus
    reg [7:0] Aout, Bout, Cout;  // Drivers for the bus
    reg EnableA, EnableB, EnableC; // Enable signals for drivers

    // Assign drivers to the bus using enable signals
    assign bus = EnableA ? Aout : 8'hzz;  // A drives bus if EnableA is 1
    assign bus = EnableB ? Bout : 8'hzz;  // B drives bus if EnableB is 1
    assign bus = EnableC ? Cout : 8'hzz;  // C drives bus if EnableC is 1

    initial begin
        // Scenario: EnableA = 1, EnableB = 1, EnableC = 0
        EnableA = 1; Aout = 8'hFF;  // A drives 'FF'
        EnableB = 1; Bout = 8'h00;  // B drives '00'
        EnableC = 0; Cout = 8'hZZ;  // C is inactive (Hi-Z)

        #10 $display("At time %0t: bus = %h", $time, bus);
        #10 $finish;
    end
endmodule
```

```
[2024-12-17 03:35:35 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
At time 10: bus = xx
design.sv:18: $finish called at 20 (1s)
Done
```

5. 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.

Wor/trior

```
module TestTrior;
    trior bus;            // Declare a wire with OR behavior (trior)
    reg Aout, Bout;       // Two data drivers
    reg EnableA, EnableB; // Enable signals for drivers

    // Drive 'bus' based on enable signals
    assign bus = EnableA ? Aout : 8'hz;  // Aout drives bus when EnableA = 1
    assign bus = EnableB ? Bout : 8'hz;  // Bout drives bus when EnableB = 1

    initial begin
        $monitor("time=%0g, bus=%b", $time, bus);
        EnableA = 1; EnableB = 1; Aout = 0; Bout = 0; // Both enabled, drivers set to 0
        #1 Aout = 1;          // Change Aout to 1
        #1 Bout = 1;          // Change Bout to 1
        #2 $finish;           // End simulation
    end
endmodule
```

Log Share

```
[2024-12-17 03:54:54 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
time=0, bus=0
time=1, bus=1
design.sv:15: $finish called at 4 (1s)
Done
```

Wand

design.sv

```
module testWand();
    wand X;         // Declare a wire AND (wand)
    reg A, B;       // Two drivers

    assign X = A;   // A drives the wire
    assign X = B;   // B drives the wire

    initial begin
        $monitor("%g X = %b A = %b B = %b", $time, X, A, B);
        #1 A = 0;    // Time 1: A = 0, B = undefined
        #1 B = 0;    // Time 2: A = 0, B = 0
        #1 A = 1;    // Time 3: A = 1, B = 0
        #1 A = 0;    // Time 4: A = 0, B = 0
        #1 B = 1;    // Time 5: A = 0, B = 1
        #1 A = 1;    // Time 6: A = 1, B = 1
        #1 A = 0;    // Time 7: A = 0, B = 1
        #1 $finish;  // End simulation at time 8
    end
endmodule
```

Log Share

```
[2024-12-17 03:59:09 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
0 X = x A = x B = x
1 X = 0 A = 0 B = x
2 X = 0 A = 0 B = 0
3 X = 0 A = 1 B = 0
4 X = 0 A = 0 B = 0
5 X = 0 A = 0 B = 1
6 X = 1 A = 1 B = 1
7 X = 0 A = 0 B = 1
design.sv:17: $finish called at 8 (1s)
Done
```

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

design.sv

```
module testTri0Tri1;
    tri0 wire0;     // tri0: defaults to 0 if no driver
    tri1 wire1;     // tri1: defaults to 1 if no driver

    reg driverA, driverB;  // Two drivers for the wires

    assign wire0 = driverA;  // Driver A for wire0
    assign wire1 = driverB;  // Driver B for wire1

    initial begin
        $monitor("Time=%0t | driverA=%b, wire0=%b | driverB=%b, wire1=%b",
                 $time, driverA, wire0, driverB, wire1);

        // Assign 4 different values: 0, 1, X, Z
        driverA = 0; driverB = 0; #5;   // Step 1: Both 0
        driverA = 1; driverB = 1; #5;   // Step 2: Both 1
        driverA = 'x; driverB = 'x; #5; // Step 3: Both X
        driverA = 'z; driverB = 'z; #5; // Step 4: Both Z

        #5 $finish; // End simulation
    end
endmodule
```

Log Share

```
[2024-12-17 04:02:50 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
Time=0 | driverA=0, wire0=0 | driverB=0, wire1=0
Time=5 | driverA=1, wire0=1 | driverB=1, wire1=1
Time=10 | driverA=x, wire0=x | driverB=x, wire1=x
Time=15 | driverA=z, wire0=0 | driverB=z, wire1=1
design.sv:20: $finish called at 25 (1s)
Done
```

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

```
module testWireRetain();
    reg [1:0] flag;        // Control flag
    reg [7:0] last_data;   // Register to "retain" previous value
    wire [7:0] data;       // Output wire to simulate trireg behavior

    // Procedurally assign the data
    assign data = (flag == 1) ? 10 :
                  (flag == 3) ? 30 :
                  (flag == 2) ? 255 : last_data;

    initial begin
        $monitor("Time=%0t, flag=%d, data=%d", $time, flag, data);

        last_data = 8'bz; // Initialize last_data
        flag = 1;  #200;   // flag = 1 → data = 10
        last_data = data;

        flag = 0;  #200;   // flag = 0 → data retains last value
        last_data = data;

        flag = 3;  #200;   // flag = 3 → data = 30
        last_data = data;

        flag = 0;  #200;   // flag = 0 → data retains last value
        last_data = data;

        flag = 2;  #200;   // flag = 2 → data = 255
        last_data = data;

        flag = 0;  #10;    // flag = 0 → data retains last value
        $finish;
    end
endmodule
```

Log Share

```
[2024-12-17 04:11:55 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
Time=0, flag=1, data= 10
Time=200, flag=0, data= 10
Time=400, flag=3, data= 30
Time=600, flag=0, data= 30
Time=800, flag=2, data=255
Time=1000, flag=0, data=255
design.sv:31: $finish called at 1010 (1s)
Done
```

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

```
module testInteger;
    wire pwrGood, pwrOn, pwrStable; // Wire declarations for signals

    // Explicit declarations
    integer i;        // 32-bit signed integer
    time t;           // 64-bit unsigned, behaves like a 64-bit reg
    real r;           // Real data type for floating-point numbers

    // Assign statements for wire signals
    assign pwrStable = 1'b1; // Constant 1 assigned to pwrStable
    assign pwrOn = 1;        // Equivalent to 1'b1
    assign pwrGood = pwrOn & pwrStable; // Logical AND operation

    initial begin
        // Assign values to integer, time, and real types
        i = 123;             // Integer value (fractional part ignored)
        r = 123456e3;        // Real value: Scientific notation (123456 * 10^3)
        t = 123456e3;        // Time value, rounded to the nearest integer (time is in simulation ticks)

        // Display values of i, t, and r
        $display("i=%0d, t=%0d, r=%f", i, t, r);

        // Display additional signal values and simulation time
        #2 $display("TIME=%0d, ON=%b, STABLE=%b, GOOD=%b", $time, pwrOn, pwrStable, pwrGood);

        $finish; // End the simulation
    end
endmodule
```

Log Share

```
[2024-12-17 04:15:27 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
i=123, t=123456000, r=123456000.000000
TIME=2, ON=1, STABLE=1, GOOD=1
design.sv:26: $finish called at 2 (1s)
Done
```

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

```
module testTime;
    time t_var;         // Declare a 'time' variable
    real r_var;         // Declare a 'real' variable for $realtime

    initial begin
        $monitor("Time = %0t, t_var = %0d, r_var = %0f", $time, t_var, r_var);

        #0 t_var = $stime;  // Assign the simulation time using $stime
        #5 r_var = $realtime;  // Assign the simulation time using $realtime
        #10 t_var = $stime;  // Update t_var with $stime
        #15 r_var = $realtime; // Update r_var with $realtime

        #20 $finish; // End simulation
    end
endmodule
```

Log Share

```
[2024-12-17 04:18:24 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
Time = 0, t_var = 0, r_var = 0.000000
Time = 5, t_var = 0, r_var = 5.000000
Time = 15, t_var = 15, r_var = 5.000000
Time = 30, t_var = 15, r_var = 30.000000
design.sv:13: $finish called at 50 (1s)
Done
```

10. 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 testTimescale;
    real real_delay; // Declare a real variable for calculated delay
    integer time_units; // Integer value for equivalent time units

    initial begin
        $display("Time at Start = %0t", $time);

        // Calculate delay in integer time units (rounded)
        real_delay = 2.71828 * 10; // 10ns per time unit
        time_units = real_delay + 0.5; // Round to the nearest integer

        #time_units; // Apply the rounded delay
        $display("Time after Delay = %0t", $time);

        // Display calculated real delay for comparison
        $display("Calculated Real Delay = %0.4f ns", real_delay);
        $display("Rounded Delay Applied = %0d time units", time_units);

        $finish;
    end
endmodule
```

Log Share

```
[2024-12-17 04:23:28 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
Time at Start = 0
Time after Delay = 2800
Calculated Real Delay = 27.1828 ns
Rounded Delay Applied = 28 time units
design.sv:21: $finish called at 2800 (100ps)
Done
```

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


```
module signedNumber;
    reg [31:0] a; // 32-bit register to hold hexadecimal values

    initial begin
        a = 14'h1234; // Assign a 14-bit value
        $display("Current Value of a = %h", a);

        a = 14'h1234; // Reassign the same value
        $display("Current Value of a = %h", a);

        a = 32'hDEAD_BEEF; // Assign a 32-bit value
        $display("Current Value of a = %h", a);

        a = 32'hDEAD_BEEF; // Reassign the same value
        $display("Current Value of a = %h", a);

        #10 $finish; // End simulation
    end
endmodule
```

Log Share

```
[2024-12-17 04:27:53 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
Current Value of a = 00001234
Current Value of a = 00001234
Current Value of a = deadbeef
Current Value of a = deadbeef
design.sv:17: $finish called at 10 (1s)
Done
```

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

```
module helloWorld;

    initial begin
        // Print characters and their ASCII values
        $display("Printing Characters and their ASCII Code Numbers:");
        $display("Double Quote: \"%c\" ASCII Code: %0d", 34, 34);
        $display("Percent Character: \"%c\" ASCII Code: %0d", 37, 37);
        $display("At Symbol: \"%c\" ASCII Code: %0d", 64, 64);

        // Alternative representation using character literals
        $display("Using Character Literals:");
        $display("Double Quote: \"\\\"\" ASCII Code: %0d", 34);
        $display("Percent Character: \"%%\" ASCII Code: %0d", 37);
        $display("At Symbol: \"@\" ASCII Code: %0d", 64);

        #10 $finish; // End simulation
    end

endmodule
```

Log Share

```
[2024-12-17 04:32:20 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
Printing Characters and their ASCII Code Numbers:
Double Quote: """ ASCII Code: 34
Percent Character: "%" ASCII Code: 37
At Symbol: "@" ASCII Code: 64
Using Character Literals:
Double Quote: "\"" ASCII Code: 34
Percent Character: "%" ASCII Code: 37
At Symbol: "@" ASCII Code: 64
design.sv:16: $finish called at 10 (1s)
Done
```

13. 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 formatExample;
    // Variable declarations
    reg [7:0] binary_val;  // 8-bit binary value
    reg [7:0] char_val;    // Character value (ASCII)
    integer dec_val;       // Decimal integer
    real exp_val;          // Real number for scientific notation
    real float_val;        // Floating-point number
    reg [15:0] hex_val;    // 16-bit hexadecimal value
    reg [7:0] oct_val;     // 8-bit octal value
    reg [7:0] str_val [0:5]; // String array
    time sim_time;         // Simulation time variable

    initial begin
        // Initialize values
        binary_val = 8'b10110101;  // Binary
        char_val = 8'd65;          // ASCII 'A'
        dec_val = -12345;          // Decimal value
        exp_val = 12345.678;       // Exponential notation
        float_val = 3.14159265;    // Floating-point number
        hex_val = 16'hDEAD;        // Hexadecimal
        oct_val = 8'o77;           // Octal value
        str_val[0] = "H";
        str_val[1] = "e";
        str_val[2] = "l";
        str_val[3] = "l";
        str_val[4] = "o";
        str_val[5] = "!";
        sim_time = 1234;           // Simulation time

        // Display values using different formats
        $display("=== Demonstration of Different Format Specifiers ===");
        $display("Binary Value (%%b): %b", binary_val);
        $display("Character Value (%%c): %c", char_val);
        $display("Decimal Value (%%d): %d", dec_val);
        $display("Decimal Value (%%0d): %0d", dec_val);
        $display("Exponential Notation (%%e): %e", exp_val);
        $display("Floating Point Value (%%f): %f", float_val);
        $display("Formatted Float (%%6.2f): %6.2f", float_val);
        $display("General Notation (%%g): %g", float_val);
        $display("Hexadecimal Value (%%h): %h", hex_val);
        $display("Octal Value (%%o): %o", oct_val);
        $display("String (%%s): %s%s%s%s%s%s", str_val[0], str_val[1], str_val[2], str_val[3], str_val[4], str_val[5]);
        $display("Simulation Time (%%t): %t", sim_time);
        $display("Zero-Padded Time (%%00t): %00t", sim_time);

        #10 $finish; // End simulation
    end
endmodule
```

Log Share

```
[2024-12-17 04:34:46 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
=== Demonstration of Different Format Specifiers ===
Binary Value (%b): 10110101
Character Value (%c): A
Decimal Value (%d):      -12345
Decimal Value (%0d): -12345
Exponential Notation (%e): 1.234568e+04
Floating Point Value (%f): 3.141593
Formatted Float (%6.2f):   3.14
General Notation (%g): 3.14159
Hexadecimal Value (%h): dead
Octal Value (%o): 077
String (%s): Hello!
Simulation Time (%t):                 1234
Zero-Padded Time (%00t): 1234
design.sv:46: $finish called at 10 (1s)
Done
```

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

```
module compareDisplayWriteMonitor;
    reg [3:0] a, b;

    initial begin
        // Demonstration of $display
        $display("=== Demonstration of $display ===");
        $display("At Time = %0t, a = %0d, b = %0d", $time, a, b);

        // Demonstration of $write
        $display("\n=== Demonstration of $write ===");
        $write("At Time = %0t, a = %0d", $time, a);
        $write(", b = %0d", b); // Note: No newline until explicitly added
        $display("\nAdding a newline manually after $write");

        // Demonstration of $monitor
        $display("\n=== Demonstration of $monitor ===");
        $monitor("At Time = %0t, a = %0d, b = %0d", $time, a, b);

        // Test Values: Change a and b over time
        a = 0; b = 0; #5;  // Time = 5
        a = 1; b = 2; #5;  // Time = 10
        a = 2; b = 3; #5;  // Time = 15
        a = 3; b = 4; #5;  // Time = 20
        a = 4; b = 5; #5;  // Time = 25

        #10 $finish; // End simulation
    end
endmodule
```

Log Share

```
[2024-12-17 04:37:10 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
=== Demonstration of $display ===
At Time = 0, a = x, b = x

=== Demonstration of $write ===
At Time = 0, a = x, b = x
Adding a newline manually after $write

=== Demonstration of $monitor ===
At Time = 0, a = 0, b = 0
At Time = 5, a = 1, b = 2
At Time = 10, a = 2, b = 3
At Time = 15, a = 3, b = 4
At Time = 20, a = 4, b = 5
design.sv:26: $finish called at 35 (1s)
Done
```