



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.

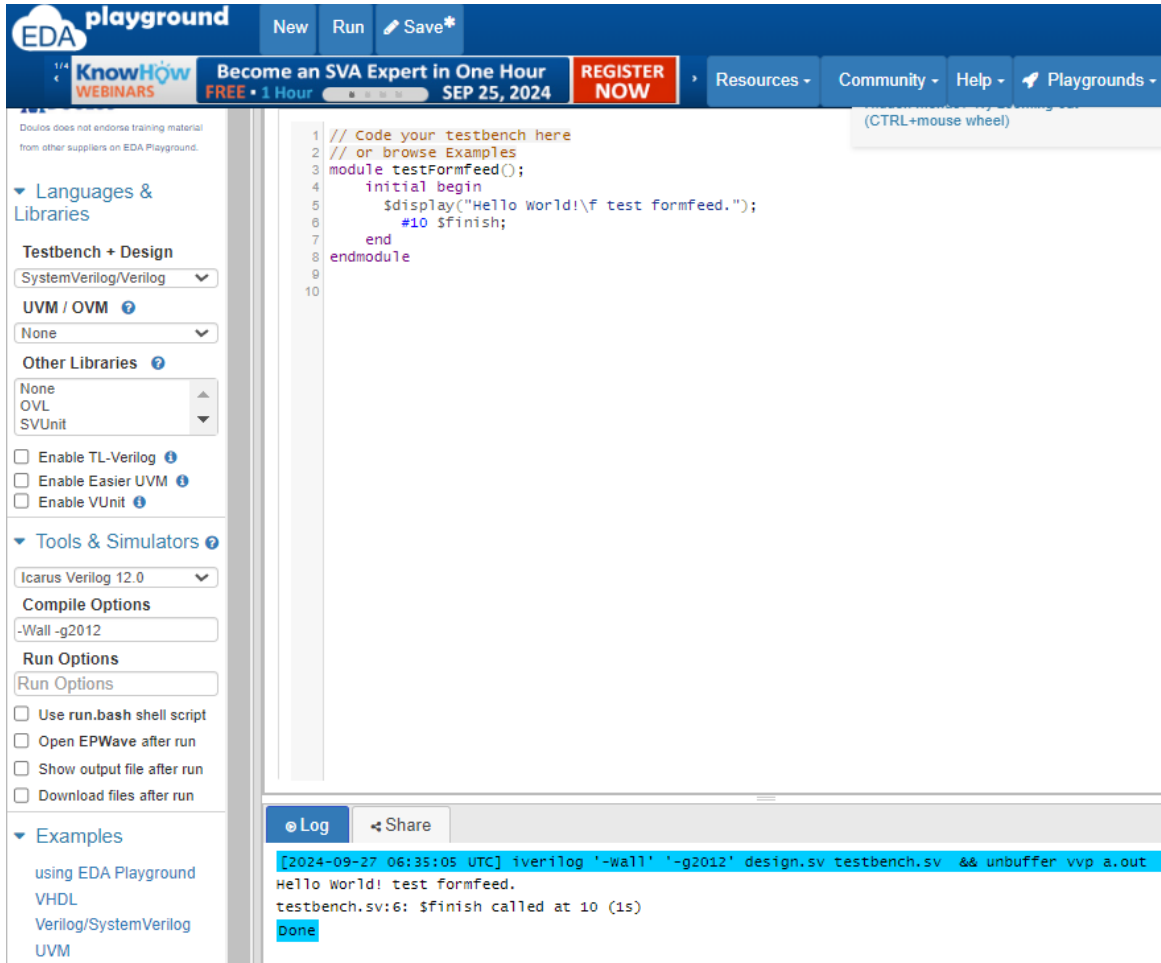
Testing Carriage Return

The screenshot shows the EDA Playground web interface. The top navigation bar includes 'New', 'Run', and 'Save*' buttons. A banner for 'Deep Learning with FPGAs' is visible. The left sidebar contains a 'Languages & Libraries' section with 'Testbench + Design' and 'Tools & Simulators' subsections. The main editor area displays Verilog code for a testbench named 'helloworld'. The code includes a module definition, an initial block with a \$display statement, and a \$finish statement. The bottom panel shows the simulation log, which includes the timestamp '[2024-09-27 06:32:47 UTC]', the command 'iverilog', and the output 'Hello World\rHey' followed by 'testbench.sv:6: \$finish called at 10 (1s)' and 'Done'.

```
1 // Code your testbench here
2 // or browse Examples
3 module helloworld ();
4     initial begin
5         $display("Hello world!\rHey");
6         #10 $finish;
7     end
8 endmodule
9
10
```

Log

```
[2024-09-27 06:32:47 UTC] iverilog '-wall' '-g2012' design.sv testbench.sv && unbuffer vvp a.out
Hello World\rHey
testbench.sv:6: $finish called at 10 (1s)
Done
```



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.

The screenshot shows the EDA Playground interface. The code editor contains the following Verilog code:

```
1 // Code your testbench here
2 // Anika Haque 20283
3 module testKeyword();
4     reg wire; // Using a keyword as a variable name
5     initial begin
6         wire = 1;
7         $display("wire = %b", wire);
8     end
9 endmodule
```

The left sidebar shows the configuration for the testbench, including the simulator (Icarus Verilog 12.0) and various options. The bottom panel shows the output of the simulation, which includes several syntax errors:

```
[2024-09-27 06:46:32 UTC] iverilog '-wall' '-g2012' design.sv testbench.sv && unbuffer vvp a.out
testbench.sv:4: syntax error
testbench.sv:1: error: Syntax error in variable list.
testbench.sv:6: syntax error
testbench.sv:6: Syntax in assignment statement 1-value.
testbench.sv:7: syntax error
testbench.sv:7: error: Malformed statement
Exit code expected: 0, received: 6
Done
```

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?

The screenshot shows the EDA Playground interface. The top navigation bar includes 'New', 'Run', and 'Save*' buttons. A banner for 'KnowHow WEBINARS' promotes a free 1-hour session on 'Become an SVA Expert in One Hour' on September 25, 2024, with a 'REGISTER NOW' button. The left sidebar contains settings for 'Languages & Libraries' (SystemVerilog/Verilog, UVM/OVM, Other Libraries), 'Tools & Simulators' (Icarus Verilog 12.0, Compile Options, Run Options), and 'Examples'. The main editor displays a Verilog testbench for a module named &\$\$abc_123. The code is as follows:

```
1 // Code your testbench here
2 // Anika Haque 20283
3
4 module &$$abc_123();
5     initial begin
6         $display("Escaped identifier works!");
7         #10 $finish;
8     end
9 endmodule
10
11
12
13
```

The output window at the bottom shows the command executed: `iverilog '-wall' '-g2012' design.sv testbench.sv && unbuffer vvp a.out`. The output indicates a syntax error at line 3 of testbench.sv: `testbench.sv:3: syntax error`. The message continues: `I give up. Exit code expected: 0, received: 2`. The output window also has 'Log' and 'Share' buttons.

&\$\$abc_123 gives syntax error.

The screenshot shows the EDA Playground web interface. On the left, there's a sidebar with navigation options like 'Languages & Libraries', 'Testbench + Design', 'Tools & Simulators', and 'Examples'. The main area displays a Verilog testbench file named 'testbench.v'. The code defines a module with an escaped identifier and a display statement. The bottom panel shows the execution log, confirming that the escaped identifier works and the display statement was executed at 10 (1s).

```

1 // Code your testbench here
2 // or browse Examples
3 module `$$abc_123 ();
4
5     initial begin
6
7         $display("Escaped identifier works!");
8
9         #10 $finish;
10
11     end
12
13 endmodule

```

Log:

```

[2024-09-30 10:55:54 UTC] iverilog '-wall' '-g2012' design.v testbench.v && unbuffer vvp a.out
Escaped identifier works!
testbench.v:9: $finish called at 10 (1s)
Done

```

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.

- 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

```

1 module triState();
2   tri [7:0] bus;           // Declare a tri-state 8-bit bus
3   reg [7:0] Aout, Bout, Cout; // Drivers for the bus
4   reg EnableA, EnableB, EnableC; // Enable signals for drivers
5
6   // Assign drivers to the bus using enable signals
7   assign bus = EnableA ? Aout : 8'hzz; // A drives bus if EnableA is 1
8   assign bus = EnableB ? Bout : 8'hzz; // B drives bus if EnableB is 1
9   assign bus = EnableC ? Cout : 8'hzz; // C drives bus if EnableC is 1
10
11   initial begin
12     // Scenario: EnableA = 1, EnableB = 1, EnableC = 0
13     EnableA = 1; Aout = 8'hFF; // A drives 'FF'
14     EnableB = 1; Bout = 8'h00; // B drives '00'
15     EnableC = 0; Cout = 8'hZZ; // C is inactive (Hi-Z)
16
17     #10 $display("At time %0t: bus = %h", $time, bus);
18     #10 $finish;
19   end
20 endmodule
21

```

Log: [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

- 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.

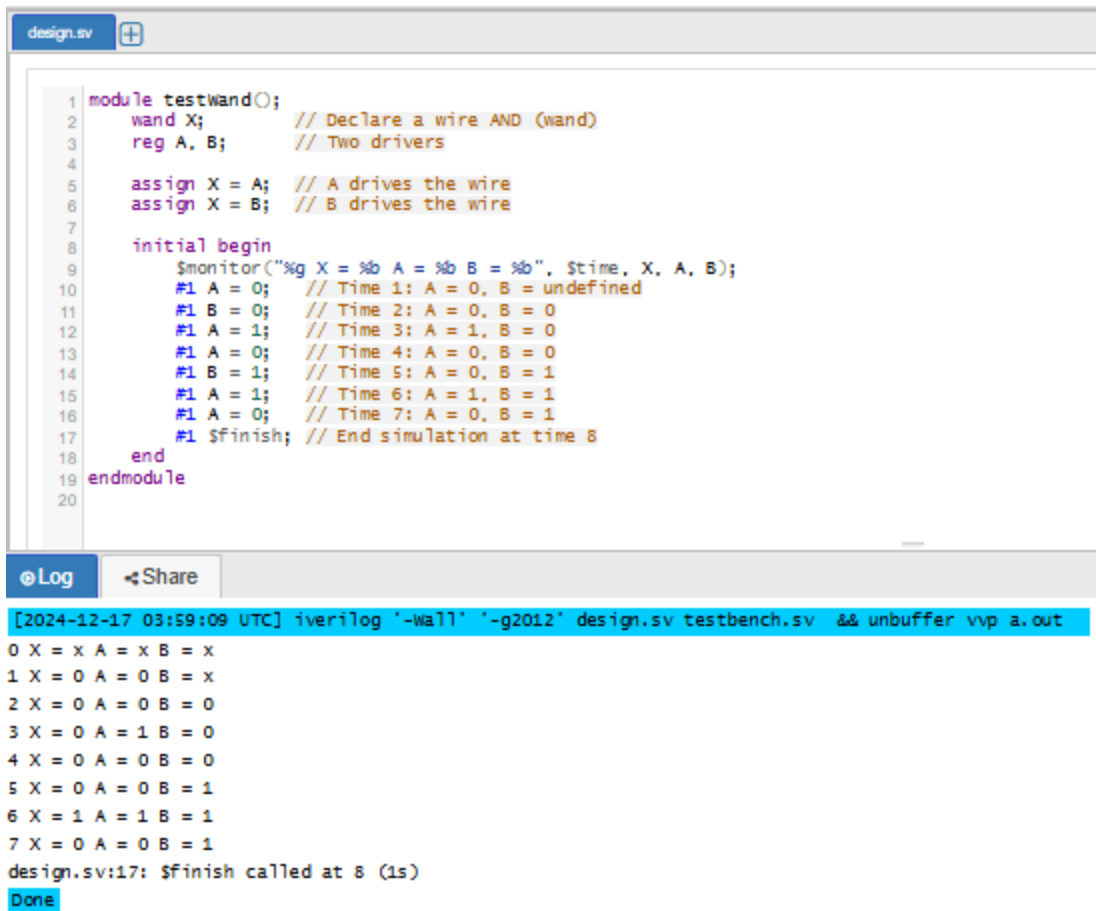
Wor/trior

```
1 module TestTrior;
2   trior bus;          // Declare a wire with OR behavior (trior)
3   reg Aout, Bout;     // Two data drivers
4   reg EnableA, EnableB; // Enable signals for drivers
5
6   // Drive 'bus' based on enable signals
7   assign bus = EnableA ? Aout : 8'hz; // Aout drives bus when EnableA = 1
8   assign bus = EnableB ? Bout : 8'hz; // Bout drives bus when EnableB = 1
9
10  initial begin
11    $monitor("time=%0g, bus=%b", $time, bus);
12    EnableA = 1; EnableB = 1; Aout = 0; Bout = 0; // Both enabled, drivers set to 0
13    #1 Aout = 1;           // Change Aout to 1
14    #1 Bout = 1;          // Change Bout to 1
15    #2 $finish;           // End simulation
16  end
17 endmodule
18
```

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



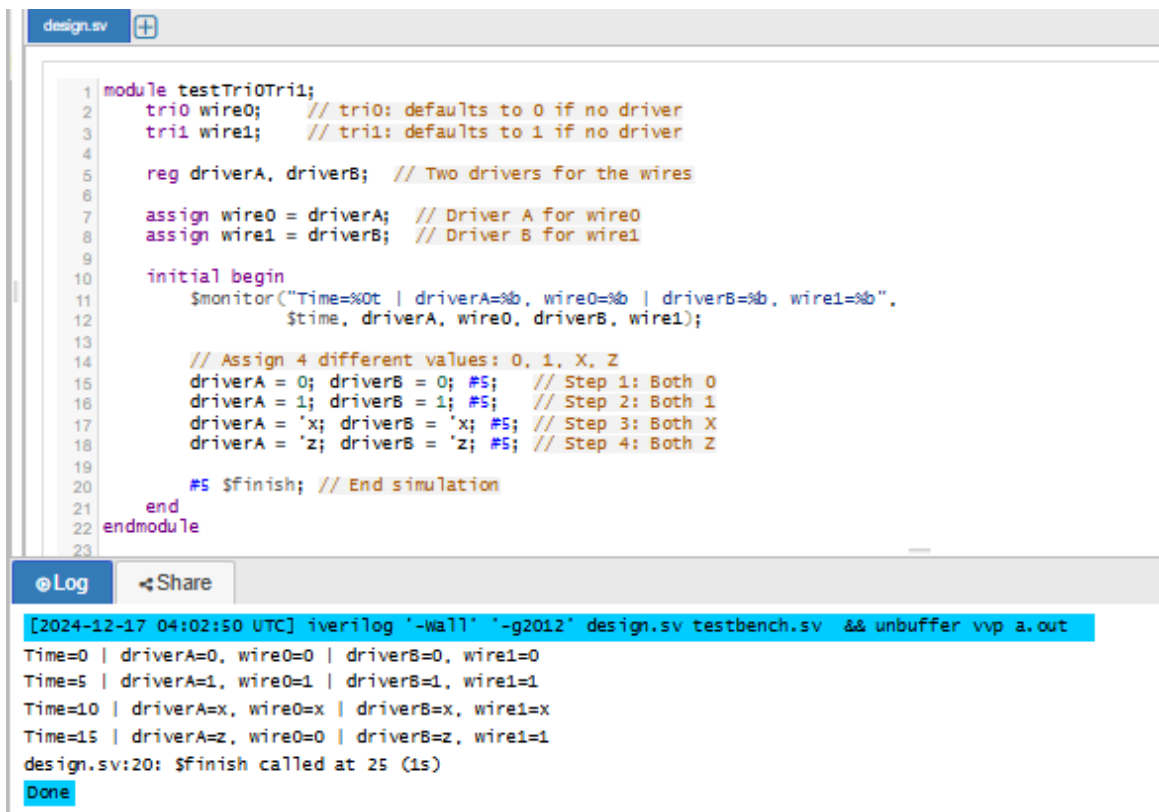
```
1 module testwand();
2     wire X; // Declare a wire AND (wand)
3     reg A, B; // Two drivers
4
5     assign X = A; // A drives the wire
6     assign X = B; // B drives the wire
7
8     initial begin
9         $monitor("%g X = %b A = %b B = %b", $time, X, A, B);
10        #1 A = 0; // Time 1: A = 0, B = undefined
11        #1 B = 0; // Time 2: A = 0, B = 0
12        #1 A = 1; // Time 3: A = 1, B = 0
13        #1 A = 0; // Time 4: A = 0, B = 0
14        #1 B = 1; // Time 5: A = 0, B = 1
15        #1 A = 1; // Time 6: A = 1, B = 1
16        #1 A = 0; // Time 7: A = 0, B = 1
17        #1 $finish; // End simulation at time 8
18    end
19 endmodule
20
```

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.



```
1 module testTri0Tri1;
2   tri0 wire0;    // tri0: defaults to 0 if no driver
3   tri1 wire1;    // tri1: defaults to 1 if no driver
4
5   reg driverA, driverB; // Two drivers for the wires
6
7   assign wire0 = driverA; // Driver A for wire0
8   assign wire1 = driverB; // Driver B for wire1
9
10  initial begin
11    $monitor("Time=%0t | driverA=%b, wire0=%b | driverB=%b, wire1=%b",
12            $time, driverA, wire0, driverB, wire1);
13
14    // Assign 4 different values: 0, 1, X, Z
15    driverA = 0; driverB = 0; #5; // Step 1: Both 0
16    driverA = 1; driverB = 1; #5; // Step 2: Both 1
17    driverA = 'x; driverB = 'x; #5; // Step 3: Both X
18    driverA = 'z; driverB = 'z; #5; // Step 4: Both Z
19
20    #5 $finish; // End simulation
21  end
22 endmodule
23
```

Log

[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=z | 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.

```
1 module testWireRetain();
2   reg [1:0] flag; // Control flag
3   reg [7:0] last_data; // Register to "retain" previous value
4   wire [7:0] data; // Output wire to simulate triereg behavior
5
6   // Procedurally assign the data
7   assign data = (flag == 1) ? 10 :
8                 (flag == 3) ? 30 :
9                 (flag == 2) ? 255 : last_data;
10
11   initial begin
12     $monitor("Time=%0t, flag=%d, data=%d", $time, flag, data);
13
14     last_data = 8'bz; // Initialize last_data
15     flag = 1; #200; // flag = 1 → data = 10
16     last_data = data;
17
18     flag = 0; #200; // flag = 0 → data retains last value
19     last_data = data;
20
21     flag = 3; #200; // flag = 3 → data = 30
22     last_data = data;
23
24     flag = 0; #200; // flag = 0 → data retains last value
25     last_data = data;
26
27     flag = 2; #200; // flag = 2 → data = 255
28     last_data = data;
29
30     flag = 0; #10; // flag = 0 → data retains last value
31     $finish;
32   end
33 endmodule
34
```

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.

```

1 module testInteger;
2   wire pwrGood, pwrOn, pwrStable; // Wire declarations for signals
3
4   // Explicit declarations
5   integer i; // 32-bit signed integer
6   time t; // 64-bit unsigned, behaves like a 64-bit reg
7   real r; // Real data type for floating-point numbers
8
9   // Assign statements for wire signals
10  assign pwrStable = 1'b1; // Constant 1 assigned to pwrStable
11  assign pwrOn = 1; // Equivalent to 1'b1
12  assign pwrGood = pwrOn & pwrStable; // Logical AND operation
13
14  initial begin
15    // Assign values to integer, time, and real types
16    i = 123; // Integer value (fractional part ignored)
17    r = 123456e3; // Real value: Scientific notation (123456 * 10^3)
18    t = 123456e3; // Time value, rounded to the nearest integer (time is in simulation ticks)
19
20    // Display values of i, t, and r
21    $display("i=%0d, t=%0d, r=%f", i, t, r);
22
23    // Display additional signal values and simulation time
24    #2 $display("TIME=%0d, ON=%b, STABLE=%b, GOOD=%b", $time, pwrOn, pwrStable, pwrGood);
25
26    $finish; // End the simulation
27  end
28 endmodule
29

```

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?

```

1 module testTime;
2   time t_var; // Declare a 'time' variable
3   real r_var; // Declare a 'real' variable for $realtime
4
5   initial begin
6     $monitor("Time = %0t, t_var = %0d, r_var = %0f", $time, t_var, r_var);
7
8     #0 t_var = $stime; // Assign the simulation time using $stime
9     #5 r_var = $realtime; // Assign the simulation time using $realtime
10    #10 t_var = $stime; // Update t_var with $stime
11    #15 r_var = $realtime; // Update r_var with $realtime
12
13    #20 $finish; // End simulation
14  end
15 endmodule
16
17

```

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.

```
1 `timescale 10ns/100ps
2
3 module testTimescale;
4     real real_delay; // Declare a real variable for calculated delay
5     integer time_units; // Integer value for equivalent time units
6
7     initial begin
8         $display("Time at Start = %0t", $time);
9
10        // Calculate delay in integer time units (rounded)
11        real_delay = 2.71828 * 10; // 10ns per time unit
12        time_units = real_delay + 0.5; // Round to the nearest integer
13
14        #time_units; // Apply the rounded delay
15        $display("Time after Delay = %0t", $time);
16
17        // Display calculated real delay for comparison
18        $display("Calculated Real Delay = %0.4f ns", real_delay);
19        $display("Rounded Delay Applied = %0d time units", time_units);
20
21        $finish;
22    end
23 endmodule
24
```

[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.

```

1 module signedNumber;
2   reg [31:0] a; // 32-bit register to hold hexadecimal values
3
4   initial begin
5     a = 14'h1234; // Assign a 14-bit value
6     $display("Current Value of a = %h", a);
7
8     a = 14'h1234; // Reassign the same value
9     $display("Current Value of a = %h", a);
10
11    a = 32'hDEAD_BEEF; // Assign a 32-bit value
12    $display("Current Value of a = %h", a);
13
14    a = 32'hDEAD_BEEF; // Reassign the same value
15    $display("Current Value of a = %h", a);
16
17    #10 $finish; // End simulation
18  end
19 endmodule
20

```

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.

```

1 module helloWorld;
2
3   initial begin
4     // Print characters and their ASCII values
5     $display("Printing Characters and their ASCII Code Numbers:");
6     $display("Double Quote: \"%c\" ASCII Code: %0d", 34, 34);
7     $display("Percent Character: \"%c\" ASCII Code: %0d", 37, 37);
8     $display("At Symbol: \"%c\" ASCII Code: %0d", 64, 64);
9
10    // Alternative representation using character literals
11    $display("Using Character Literals:");
12    $display("Double Quote: \"\\\"\" ASCII Code: %0d", 34);
13    $display("Percent Character: \"%\" ASCII Code: %0d", 37);
14    $display("At Symbol: \"%@\" ASCII Code: %0d", 64);
15
16    #10 $finish; // End simulation
17  end
18 endmodule
19
20

```

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.

```

1 module FormatExample;
2   // Variable declarations
3   reg [7:0] binary_val; // 8-bit binary value
4   reg [7:0] char_val; // Character value (ASCII)
5   integer dec_val; // Decimal integer
6   real exp_val; // Real number for scientific notation
7   real float_val; // Floating-point number
8   reg [15:0] hex_val; // 16-bit hexadecimal value
9   reg [7:0] oct_val; // 8-bit octal value
10  reg [7:0] str_val [0:5]; // String array
11  time sim_time; // Simulation time variable
12
13  initial begin
14    // Initialize values
15    binary_val = 8'b10110101; // Binary
16    char_val = 8'd65; // ASCII 'A'
17    dec_val = -12345; // Decimal value
18    exp_val = 12345.678; // Exponential notation
19    float_val = 3.14159265; // Floating-point number
20    hex_val = 16'hDEAD; // Hexadecimal
21    oct_val = 8'o77; // Octal value
22    str_val[0] = "H";
23    str_val[1] = "e";
24    str_val[2] = "l";
25    str_val[3] = "l";
26    str_val[4] = "o";
27    str_val[5] = "!";
28    sim_time = 1234; // Simulation time
29
30    // Display values using different formats
31    $display("=== Demonstration of Different Format Specifiers ===");
32    $display("Binary Value (%b): %b", binary_val);
33    $display("Character Value (%c): %c", char_val);
34    $display("Decimal Value (%d): %d", dec_val);
35    $display("Decimal Value (%0d): %0d", dec_val);
36    $display("Exponential Notation (%e): %e", exp_val);
37    $display("Floating Point Value (%f): %f", float_val);
38    $display("Formatted Float (%6.2f): %6.2f", float_val);
39    $display("General Notation (%g): %g", float_val);
40    $display("Hexadecimal Value (%h): %h", hex_val);
41    $display("Octal Value (%o): %o", oct_val);
42    $display("String (%s): %s%s%s%s%s", str_val[0], str_val[1], str_val[2], str_val[3], str_val[4], str_val[5]);
43    $display("Simulation Time (%t): %t", sim_time);
44    $display("Zero-Padded Time (%00t): %00t", sim_time);
45
46    #10 $finish; // End simulation
47  end
48 endmodule
49

```

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.

```

1 module compareDisplaywriteMonitor;
2   reg [3:0] a, b;
3
4   initial begin
5     // Demonstration of $display
6     $display("=== Demonstration of $display ===");
7     $display("At Time = %0t, a = %0d, b = %0d", $time, a, b);
8
9     // Demonstration of $write
10    $display("\n=== Demonstration of $write ===");
11    $write("At Time = %0t, a = %0d", $time, a);
12    $write(", b = %0d", b); // Note: No newline until explicitly added
13    $display("\nAdding a newline manually after $write");
14
15    // Demonstration of $monitor
16    $display("\n=== Demonstration of $monitor ===");
17    $monitor("At Time = %0t, a = %0d, b = %0d", $time, a, b);
18
19    // Test Values: Change a and b over time
20    a = 0; b = 0; #5; // Time = 5
21    a = 1; b = 2; #5; // Time = 10
22    a = 2; b = 3; #5; // Time = 15
23    a = 3; b = 4; #5; // Time = 20
24    a = 4; b = 5; #5; // Time = 25
25
26    #10 $finish; // End simulation
27  end
28 endmodule
29

```

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

```