Digital Design with HDL-Homework-1

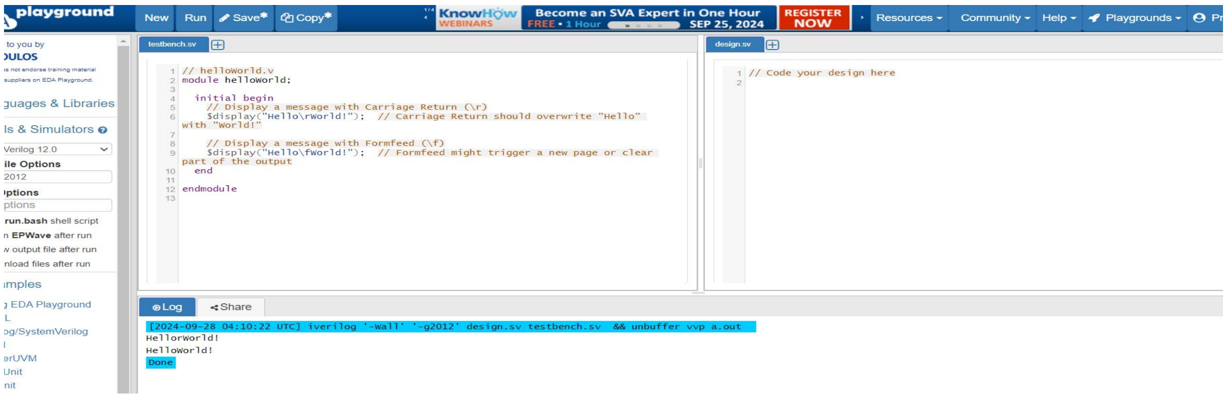
Name- Aye Myat Thiri Khin

ID-20293

Exercise-1

Q1: Write the program to verify whether white space “Carriage Return” and “Formfeed” in “helloWorld.v” work or not, and show the result.

Ans:



testbench.sv

```
// helloWorld.v
module helloWorld;

  initial begin
    // Display a message with Carriage Return (\r)
    $display("Hello\rWorld!");  // Carriage Return should overwrite "Hello" with "World!"

    // Display a message with Formfeed (\f)
    $display("Hello\fWorld!");  // Formfeed might trigger a new page or clear part of the output
  end

endmodule
```

design.sv

```
// Code your design here
```

```
[2024-09-28 04:10:22 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
HellorWorld!
HelloWorld!
Done
```

// helloWorld.v

module helloWorld;

initial begin

    // Display a message with Carriage Return (\r)

     $display("Hello\rWorld!"); // Carriage Return should overwrite "Hello" with "World!"

     // Display a message with Formfeed (\f)

    $display("Hello\fWorld!"); // Formfeed might trigger a new page or clear part of the output

 end

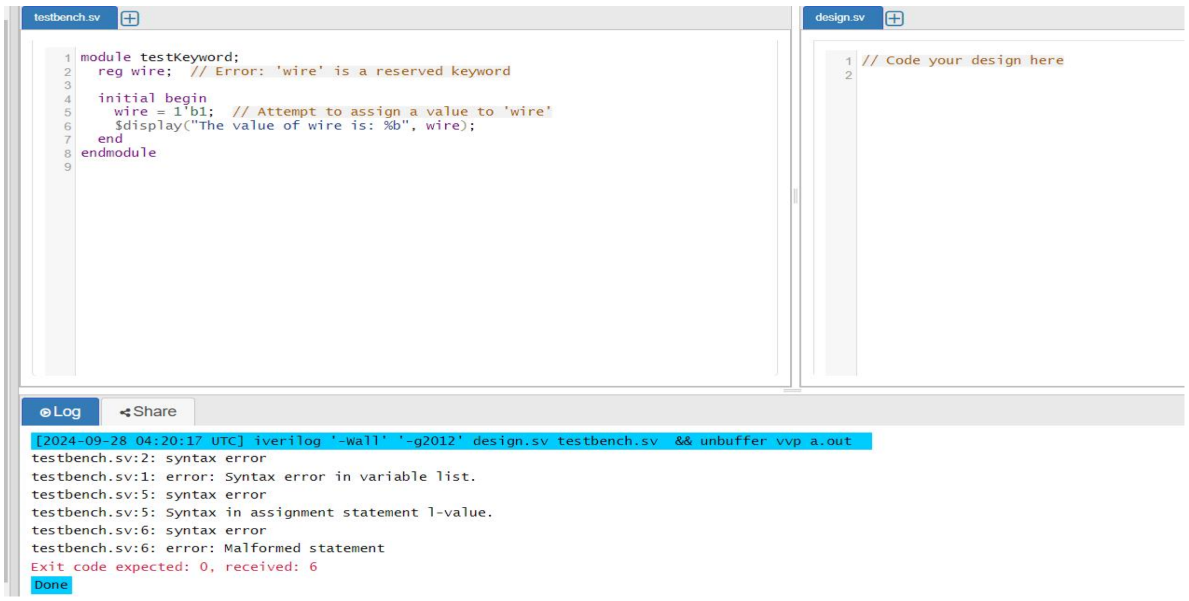
endmodule

Result Output :

```
[2024-09-28 04:10:22 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv && unbuffer vvp a.out
HellorWorld!
Hello World!
Done
```

Q2: If a variable name is defined using a keyword, write a code snippet to look at what will happen and show error/warning information.

Ans:



```
module testKeyword;
  reg wire;  // Error: 'wire' is a reserved keyword

  initial begin
    wire = 1'b1;  // Attempt to assign a value to 'wire'
    $display("The value of wire is: %b", wire);
  end
endmodule
```

```
// Code your design here
```

```
[2024-09-28 04:20:17 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
testbench.sv:2: syntax error
testbench.sv:1: error: Syntax error in variable list.
testbench.sv:5: syntax error
testbench.sv:5: Syntax in assignment statement l-value.
testbench.sv:6: syntax error
testbench.sv:6: error: Malformed statement
Exit code expected: 0, received: 6
Done
```

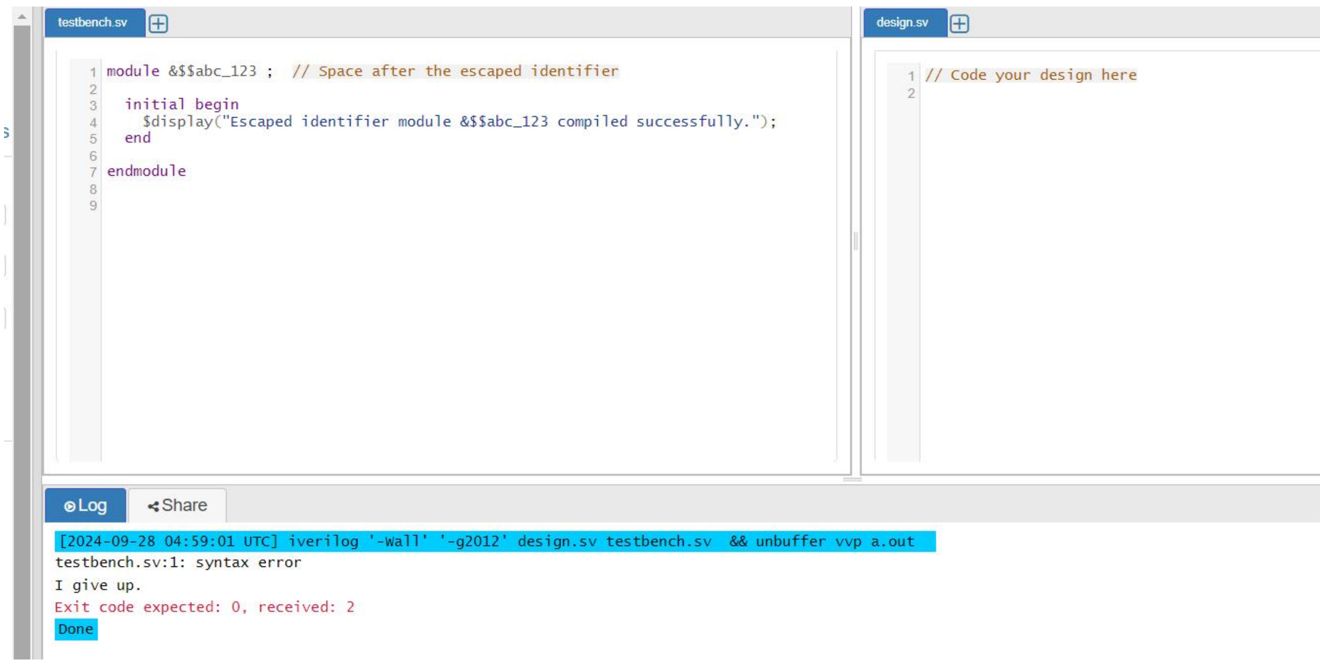
```
module testKeyword;

reg wire;  // Error: 'wire' is a reserved keyword

initial begin

    wire = 1'b1;  // Attempt to assign a value to 'wire'

    $display("The value of wire is: %b", wire);

  end

endmodule
```

Error Result :

```
[2024-09-28 04:20:17 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv && unbuffer vvp a.out
testbench.sv:2: syntax error
testbench.sv:1: error: Syntax error in variable list.
testbench.sv:5: syntax error
testbench.sv:5: Syntax in assignment statement l-value.
testbench.sv:6: syntax error
testbench.sv:6: error: Malformed statement
Exit code expected: 0, received: 6
Done
```

Q3: When a module name is &$$abc_123, how to make it pass compilation by writing a program to verify?

Ans:




```
module &$$abc_123 ;  // Space after the escaped identifier

  initial begin
    $display("Escaped identifier module &$$abc_123 compiled successfully.");
  end

endmodule
```

```
// Code your design here
```

```
[2024-09-28 04:59:01 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
testbench.sv:1: syntax error
I give up.
Exit code expected: 0, received: 2
Done
```

```
module \&$$abc_123 ;  // Space after the escaped identifier

initial begin
    $display("Escaped identifier module &$$abc_123 compiled successfully.");
  end

endmodule
```

Result Output :

```
[2024-09-28 04:39:38 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv && unbuffer vvp a.out
Escaped identifier module &$$abc_123 compiled successfully.
Done
```

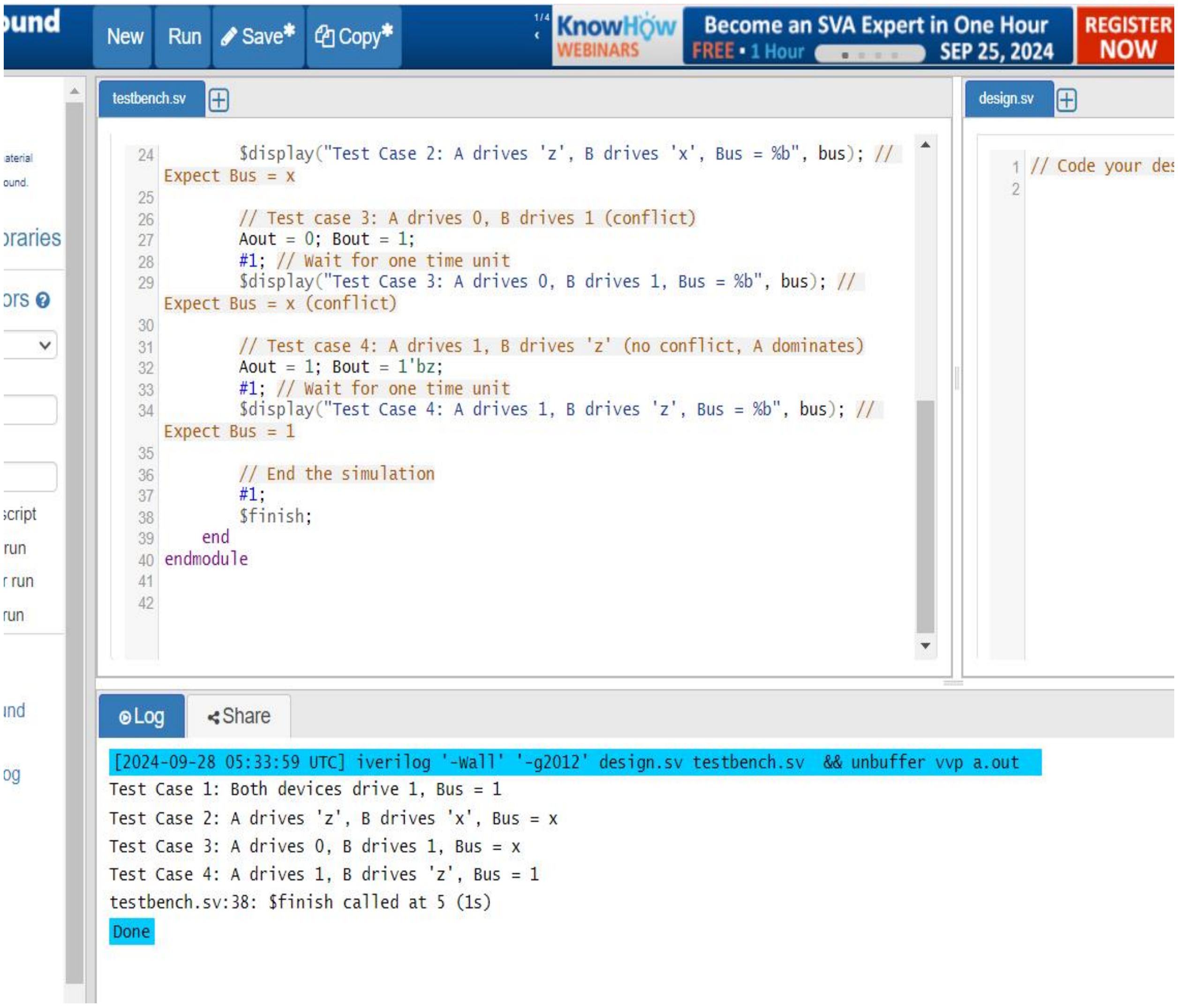
· **Invalid Module Name:** Writing module &$$abc_123; without escaping will result in a syntax error because special characters like & and $ are not allowed in module names.

· **Valid Module Name:** To successfully compile module names with special characters, use an escaped identifier by prefixing the name with a backslash (\). For example, module \&$$abc_123;.

· **Compiling Successfully:** With the escaped identifier, the module will compile without errors, allowing you to use special characters in the name.

Q4: 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.

Ans:



```
        $display("Test Case 2: A drives 'z', B drives 'x', Bus = %b", bus); // Expect Bus = x

        // Test case 3: A drives 0, B drives 1 (conflict)
        Aout = 0; Bout = 1;
        #1; // Wait for one time unit
        $display("Test Case 3: A drives 0, B drives 1, Bus = %b", bus); // Expect Bus = x (conflict)

        // Test case 4: A drives 1, B drives 'z' (no conflict, A dominates)
        Aout = 1; Bout = 1'bz;
        #1; // Wait for one time unit
        $display("Test Case 4: A drives 1, B drives 'z', Bus = %b", bus); // Expect Bus = 1

        // End the simulation
        #1;
        $finish;
    end
endmodule
```

```
// Code your des
```

```
[2024-09-28 05:33:59 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
Test Case 1: Both devices drive 1, Bus = 1
Test Case 2: A drives 'z', B drives 'x', Bus = x
Test Case 3: A drives 0, B drives 1, Bus = x
Test Case 4: A drives 1, B drives 'z', Bus = 1
testbench.sv:38: $finish called at 5 (1s)
Done
```

```
module tri_state_example;

    reg Aout, Bout;              // Values driven by two devices
    reg EnableA, EnableB;        // Enable signals for each device
    tri bus;                     // Define a 'tri' type variable

    // Bus driven by Aout if EnableA is high, else high-impedance
    assign bus = EnableA ? Aout : 1'bz;

    // Bus driven by Bout if EnableB is high, else high-impedance
```

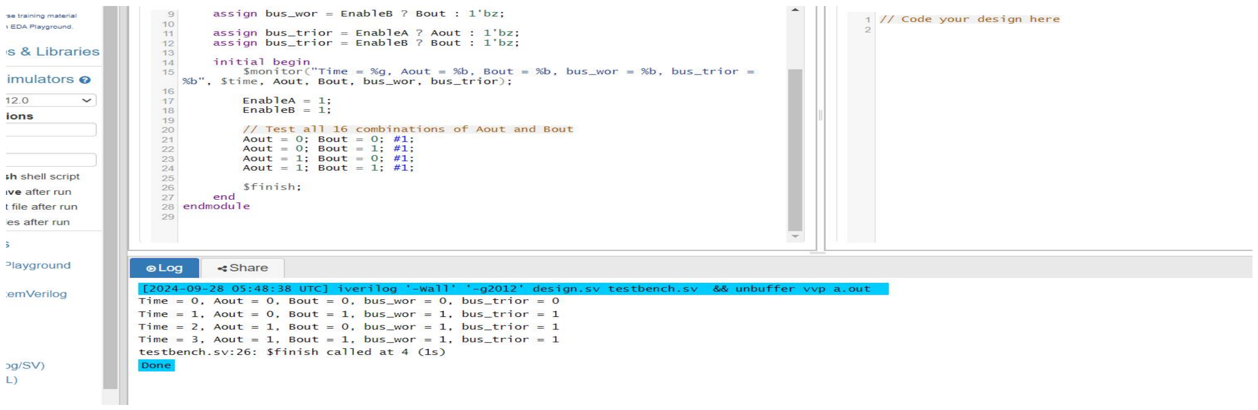
```
  assign bus = EnableB ? Bout : bus; // Maintain the previous value of bus when EnableB is low

  initial begin
    // Initialize enable signals
    EnableA = 0;
    EnableB = 0;

    // Test case 1: Both devices enabled driving the same value (1)
    EnableA = 1; EnableB = 1;
    Aout = 1; Bout = 1;
    #1; // Wait for one time unit
    $display("Test Case 1: Both devices drive 1, Bus = %b", bus); // Expect Bus = 1

    // Test case 2: A drives 'z', B drives '1'
    Aout = 1'bz; Bout = 1;
    #1; // Wait for one time unit
    $display("Test Case 2: A drives 'z', B drives 1, Bus = %b", bus); // Expect Bus = 1

    // Test case 3: A drives 0, B drives 1 (conflict)
    Aout = 0; Bout = 1;
    #1; // Wait for one time unit
    $display("Test Case 3: A drives 0, B drives 1, Bus = %b", bus); // Expect Bus = x (conflict)

    // Test case 4: A drives 1, B drives 'z' (no conflict)
    Aout = 1; Bout = 1'bz;
    #1; // Wait for one time unit
    $display("Test Case 4: A drives 1, B drives 'z', Bus = %b", bus); // Expect Bus = 1

    // End the simulation
    #1;
    $finish;

  end
endmodule
```

Result Output :

```
[2024-09-28 05:33:59 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv && unbuffer vvp
a.out
Test Case 1: Both devices drive 1, Bus = 1
Test Case 2: A drives 'z', B drives 'x', Bus = x
Test Case 3: A drives 0, B drives 1, Bus = x
Test Case 4: A drives 1, B drives 'z', Bus = 1
testbench.sv:38: $finish called at 5 (1s)
Done
```

Q5: 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.
Ans:

Test Program for wor and trior



```
    assign bus_wor = EnableB ? Bout : 1'bz;

    assign bus_trior = EnableA ? Aout : 1'bz;
    assign bus_trior = EnableB ? Bout : 1'bz;

    initial begin
        $monitor("Time = %g, Aout = %b, Bout = %b, bus_wor = %b, bus_trior =
%b", $time, Aout, Bout, bus_wor, bus_trior);

        EnableA = 1;
        EnableB = 1;

        // Test all 16 combinations of Aout and Bout
        Aout = 0; Bout = 0; #1;
        Aout = 0; Bout = 1; #1;
        Aout = 1; Bout = 0; #1;
        Aout = 1; Bout = 1; #1;

        $finish;
    end
endmodule
```

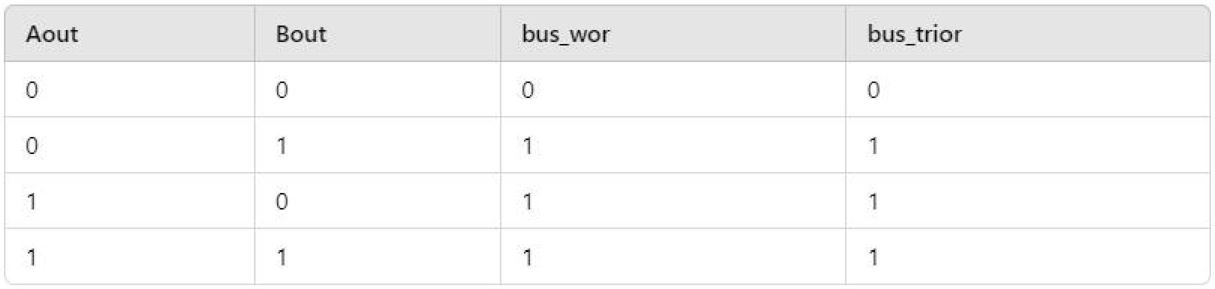
```
[2024-09-28 05:48:38 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
Time = 0, Aout = 0, Bout = 0, bus_wor = 0, bus_trior = 0
Time = 1, Aout = 0, Bout = 1, bus_wor = 1, bus_trior = 1
Time = 2, Aout = 1, Bout = 0, bus_wor = 1, bus_trior = 1
Time = 3, Aout = 1, Bout = 1, bus_wor = 1, bus_trior = 1
testbench.sv:26: $finish called at 4 (1s)
Done
```

```
module TestWorTrior;
    wor bus_wor;      // Wor declaration
    trior bus_trior; // Trior declaration
    reg Aout, Bout;  // Input signals
    reg EnableA, EnableB;

    // Assign values to bus_wor and bus_trior based on enable signals
    assign bus_wor = EnableA ? Aout : 1'bz;
    assign bus_wor = EnableB ? Bout : 1'bz;

    assign bus_trior = EnableA ? Aout : 1'bz;
    assign bus_trior = EnableB ? Bout : 1'bz;

    initial begin
        $monitor("Time = %g, Aout = %b, Bout = %b, bus_wor = %b, bus_trior = %b", $time,
Aout, Bout, bus_wor, bus_trior);

        EnableA = 1;
        EnableB = 1;

        // Test all 16 combinations of Aout and Bout
        Aout = 0; Bout = 0; #1;
        Aout = 0; Bout = 1; #1;
        Aout = 1; Bout = 0; #1;
        Aout = 1; Bout = 1; #1;

        $finish;
    end
endmodule
```

Result Output :

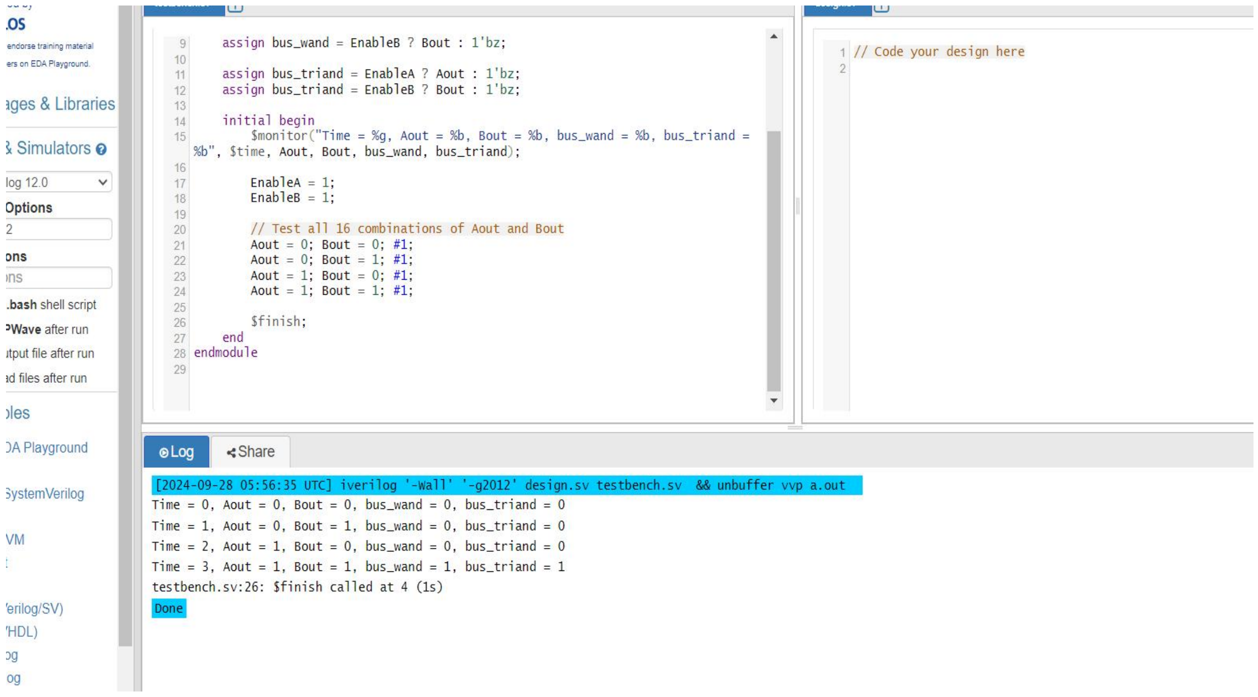
```
[2024-09-28 05:48:38 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv && unbuffer vvp a.out
Time = 0, Aout = 0, Bout = 0, bus_wor = 0, bus_trior = 0
Time = 1, Aout = 0, Bout = 1, bus_wor = 1, bus_trior = 1
Time = 2, Aout = 1, Bout = 0, bus_wor = 1, bus_trior = 1
Time = 3, Aout = 1, Bout = 1, bus_wor = 1, bus_trior = 1
testbench.sv:26: $finish called at 4 (1s)
Done
```

Truth Table for wor & trior



Aout	Bout	bus_wor	bus_trior
0	0	0	0
0	1	1	1
1	0	1	1
1	1	1	1

Test Program for wand and triand



```
    assign bus_wand = EnableB ? Bout : 1'bz;

    assign bus_triand = EnableA ? Aout : 1'bz;
    assign bus_triand = EnableB ? Bout : 1'bz;

    initial begin
        $monitor("Time = %g, Aout = %b, Bout = %b, bus_wand = %b, bus_triand = %b", $time, Aout, Bout, bus_wand, bus_triand);

        EnableA = 1;
        EnableB = 1;

        // Test all 16 combinations of Aout and Bout
        Aout = 0; Bout = 0; #1;
        Aout = 0; Bout = 1; #1;
        Aout = 1; Bout = 0; #1;
        Aout = 1; Bout = 1; #1;

        $finish;
    end
endmodule
```

```
// Code your design here
```

```
[2024-09-28 05:56:35 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
Time = 0, Aout = 0, Bout = 0, bus_wand = 0, bus_triand = 0
Time = 1, Aout = 0, Bout = 1, bus_wand = 0, bus_triand = 0
Time = 2, Aout = 1, Bout = 0, bus_wand = 0, bus_triand = 0
Time = 3, Aout = 1, Bout = 1, bus_wand = 1, bus_triand = 1
testbench.sv:26: $finish called at 4 (1s)
Done
```

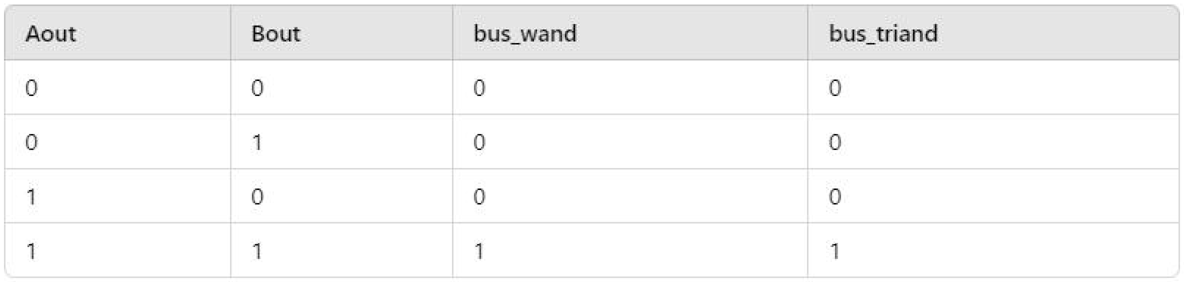
module TestWandTriand;

```
    wand bus_wand;     // Wand declaration
    triand bus_triand; // Triand declaration
    reg Aout, Bout;    // Input signals
    reg EnableA, EnableB;

    // Assign values to bus_wand and bus_triand based on enable signals
    assign bus_wand = EnableA ? Aout : 1'bz;
    assign bus_wand = EnableB ? Bout : 1'bz;

    assign bus_triand = EnableA ? Aout : 1'bz;
    assign bus_triand = EnableB ? Bout : 1'bz;

    initial begin
        $monitor("Time = %g, Aout = %b, Bout = %b, bus_wand = %b, bus_triand = %b", $time,
Aout, Bout, bus_wand, bus_triand);

        EnableA = 1;
        EnableB = 1;

        // Test all 16 combinations of Aout and Bout
        Aout = 0; Bout = 0; #1;
        Aout = 0; Bout = 1; #1;
        Aout = 1; Bout = 0; #1;
        Aout = 1; Bout = 1; #1;
        $finish;
    end
endmodule
```

Result Output :

```
[2024-09-28 05:56:35 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv && unbuffer vvp a.out
Time = 0, Aout = 0, Bout = 0, bus_wand = 0, bus_triand = 0
Time = 1, Aout = 0, Bout = 1, bus_wand = 0, bus_triand = 0
Time = 2, Aout = 1, Bout = 0, bus_wand = 0, bus_triand = 0
Time = 3, Aout = 1, Bout = 1, bus_wand = 1, bus_triand = 1
testbench.sv:26: $finish called at 4 (1s)
Done
```

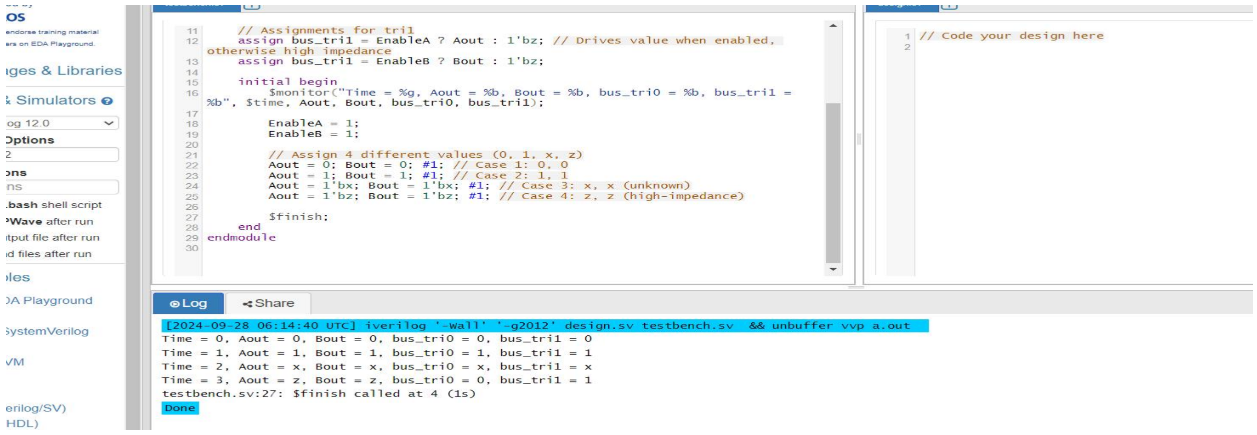
Truth Table for wand & triand



Aout	Bout	bus_wand	bus_triand
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

Q6: Generate variable type “tri0/tri1”, and assign 4 different values (0, 1, X, Z) to observe what you are going to get.

Ans:



```
    // Assignments for tri1
    assign bus_tri1 = EnableA ? Aout : 1'bz; // Drives value when enabled,
otherwise high impedance
    assign bus_tri1 = EnableB ? Bout : 1'bz;

    initial begin
        $monitor("Time = %g, Aout = %b, Bout = %b, bus_tri0 = %b, bus_tri1 =
%b", $time, Aout, Bout, bus_tri0, bus_tri1);

        EnableA = 1;
        EnableB = 1;

        // Assign 4 different values (0, 1, x, z)
        Aout = 0; Bout = 0; #1; // Case 1: 0, 0
        Aout = 1; Bout = 1; #1; // Case 2: 1, 1
        Aout = 1'bx; Bout = 1'bx; #1; // Case 3: x, x (unknown)
        Aout = 1'bz; Bout = 1'bz; #1; // Case 4: z, z (high-impedance)

        $finish;
    end
endmodule
```

```
// Code your design here
```

```
[2024-09-28 06:14:40 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
Time = 0, Aout = 0, Bout = 0, bus_tri0 = 0, bus_tri1 = 0
Time = 1, Aout = 1, Bout = 1, bus_tri0 = 1, bus_tri1 = 1
Time = 2, Aout = x, Bout = x, bus_tri0 = x, bus_tri1 = x
Time = 3, Aout = z, Bout = z, bus_tri0 = 0, bus_tri1 = 1
testbench.sv:27: $finish called at 4 (1s)
Done
```

```
module TestTri0Tri1;

  tri0 bus_tri0;  // Tri-state net that defaults to 0

  tri1 bus_tri1;  // Tri-state net that defaults to 1
  reg Aout, Bout; // Input signals
  reg EnableA, EnableB;

  // Assignments for tri0
  assign bus_tri0 = EnableA ? Aout : 1'bz; // Drives value when enabled, otherwise high
impedance
  assign bus_tri0 = EnableB ? Bout : 1'bz;

  // Assignments for tri1
  assign bus_tri1 = EnableA ? Aout : 1'bz; // Drives value when enabled, otherwise high
impedance
  assign bus_tri1 = EnableB ? Bout : 1'bz;

  initial begin
    $monitor("Time = %g, Aout = %b, Bout = %b, bus_tri0 = %b, bus_tri1 = %b", $time, Aout,
Bout, bus_tri0, bus_tri1);

    EnableA = 1;
    EnableB = 1;

    // Assign 4 different values (0, 1, x, z)
    Aout = 0; Bout = 0; #1; // Case 1: 0, 0
    Aout = 1; Bout = 1; #1; // Case 2: 1, 1
    Aout = 1'bx; Bout = 1'bx; #1; // Case 3: x, x (unknown)
    Aout = 1'bz; Bout = 1'bz; #1; // Case 4: z, z (high-impedance)

    $finish;
  end
endmodule
```

Result Output :

```
[2024-09-28 06:14:40 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv && unbuffer vvp a.out
Time = 0, Aout = 0, Bout = 0, bus_tri0 = 0, bus_tri1 = 0
Time = 1, Aout = 1, Bout = 1, bus_tri0 = 1, bus_tri1 = 1
Time = 2, Aout = x, Bout = x, bus_tri0 = x, bus_tri1 = x
Time = 3, Aout = z, Bout = z, bus_tri0 = 0, bus_tri1 = 1
testbench.sv:27: $finish called at 4 (1s)
Done
```

Q7:Retype "testTrireg" module example code on the "WK#2" handout and provide the results.

Ans:

```
module testTrireg ();

    trireg [7:0] data;   // Declare trireg for data

    reg [1:0] flag;      // Declare 2-bit register for flag



    // Assign data based on the value of flag

    assign data = (flag == 1) ? 10 :

                  (flag == 0) ? 8'bzz :

                  (flag == 3) ? 30 : 255;



    initial begin

        flag = 1;        // Initially, set flag to 1

        #200 flag = 0;   // After 200 time units, set flag to 0

        #200 flag = 3;   // After another 200 time units, set flag to 3

        #200 flag = 0;   // After another 200 time units, set flag to 0

        #200 flag = 2;   // After another 200 time units, set flag to 2

        #200 flag = 0;   // After another 200 time units, set flag to 0
        // Monitor changes in data and time

        $monitor("Time = %g, data = %d", $time, data);


        #10 $finish;     // Finish the simulation after 10 time units

    end

endmodule
```

Result Output :

```
QuestaSim-64 qrun 2023.1 Utility 2023.01 Jan 23 2023

Start time: 23:30:45 on Sep 25,2024

qrun -batch -access=rw+/. -timescale 1ns/1ns -mfcu design.sv testbench.sv -voptargs="+acc=npr" -do " run -all; exit"

Creating library 'qrun.out/work'.

QuestaSim-64 vlog 2023.1 Compiler 2023.01 Jan 23 2023

Start time: 23:30:45 on Sep 25,2024

vlog -timescale 1ns/1ns -mfcu design.sv testbench.sv -work qrun.out/work -csession=incr -writesessionid "+qrun.out/top_dus" -statslog qrun.out/stats_log

-- Compiling module testTrireg


Top level modules:

        testTrireg

End time: 23:30:45 on Sep 25,2024, Elapsed time: 0:00:00

Errors: 0, Warnings: 0

QuestaSim-64 vopt 2023.1 Compiler 2023.01 Jan 23 2023
```

```
** Warning: (vopt-10587) Some optimizations are turned off because the +acc switch is in effect. This will cause your simulation to run slowly. Please use -access/-debug to maintain needed visibility.

Start time: 23:30:45 on Sep 25,2024

vopt -access=rw+/. -timescale 1ns/1ns -mfcu "+acc=npr" -findtoplevels /home/runner/qrun.out/work+0+ -work qrun.out/work -statslog qrun.out/stats_log -o qrun_opt



Top level modules:

        testTrireg



Analyzing design...

-- Loading module testTrireg

Optimizing 1 design-unit (inlining 0/1 module instances):

-- Optimizing module testTrireg(fast)

Optimized design name is qrun_opt

End time: 23:30:45 on Sep 25,2024, Elapsed time: 0:00:00

Errors: 0, Warnings: 1

# vsim -batch -voptargs="+acc=npr" -lib qrun.out/work -do " run -all; exit" -statslog qrun.out/stats_log qrun_opt -appendlog -l qrun.log -csession=incr

# Start time: 23:30:45 on Sep 25,2024

# //  Questa Sim-64

# //  Version 2023.1 linux_x86_64 Jan 23 2023

# //

# //  Copyright 1991-2023 Mentor Graphics Corporation

# //  All Rights Reserved.
# //
# //  QuestaSim and its associated documentation contain trade
# //  secrets and commercial or financial information that are the property of
# //  Mentor Graphics Corporation and are privileged, confidential,
# //  and exempt from disclosure under the Freedom of Information Act,
# //  5 U.S.C. Section 552. Furthermore, this information
# //  is prohibited from disclosure under the Trade Secrets Act,
# //  18 U.S.C. Section 1905.
# //
# Loading sv_std.std
# Loading work.testTrireg(fast)
#
# run -all
# Time = 1000, data = 255
# ** Note: $finish    : design.sv(21)
#    Time: 1010 ns  Iteration: 0  Instance: /testTrireg
# End time: 23:30:46 on Sep 25,2024, Elapsed time: 0:00:01
# Errors: 0, Warnings: 0
# *** Summary *********************************************
#     qrun: Errors:   0, Warnings:   0
#     vlog: Errors:   0, Warnings:   0
#     vopt: Errors:   0, Warnings:   1
#     vsim: Errors:   0, Warnings:   0
#   Totals: Errors:   0, Warnings:   1
```

Done

Q8: Retype "testInteger" module example code on the "WK#2" handout and give the results.
Ans:

```
module testInteger;

   wire pwrGood, pwrOn, pwrStable; // Explicitly declare wires.

   integer i; // 32-bit, signed (2's complement).

   time t; // 64-bit, unsigned, behaves like a 64-bit reg.

   real r; // Real data type of implementation-defined size.


   // An assign statement continuously drives a wire:

   assign pwrStable = 1'b1; // Net type must be in "assign" block to be bound to a value

   assign pwrOn = 1; // Assign 1 to pwrOn (1 or 1'b1)

   assign pwrGood = pwrOn & pwrStable; // Logical AND between pwrOn and
pwrStable


   initial begin

      // integer/time/real must be in initial or always block to be assigned to a value

      i = 123; // Integer assignment, decimals are not allowed for 'integer'

      r = 123456e-3; // Real number assignment: 123.456

      t = 123456e-3; // Time assignment, will be rounded to integer: 123


      // Display formatted output using $display

      $display("i=%0d t=%6.2f r=%f", i, t, r); // Correct format specifiers


      // Wait for 2 time units
```

```
        #2  $display("TIME=%0t  ON=%b  STABLE=%b  GOOD=%b",  $time,  pwrOn,
pwrStable, pwrGood);


        // Finish the simulation

        $finish;

    end

endmodule
```

Result Output :

```
[2024-09-26 06:23:30 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer
vvp a.out

i=123 t=123.00 r=123.456000

TIME=2 ON=1 STABLE=1 GOOD=1

design.sv:25: $finish called at 2 (1s)

Done
```

Q9: Create a “time” variable type and assign it values from $stime and $realtime, what will you get?

Ans:

```
module test_time;

  time t1, t2;  // Declare time variables


  initial begin

    // Delay for 5 time units

    #5;


    // Assign the current simulation time from $stime and $realtime
    t1 = $stime;

    t2 = $realtime;


    // Display the time values

    $display("Time using $stime: %0t", t1);

    $display("Time using $realtime: %0t (truncated to integer)", t2);

  end

endmodule
```

Result Output :

```
[2024-09-26 06:30:47 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  &&
unbuffer vvp a.out

Time using $stime: 5

Time using $realtime: 5 (truncated to integer)

Done
```

Q10: 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.
Ans:

```
`timescale 10ns/100ps



module test_delay;

  real actual_delay = 2.71828 * 10;  // Calculate the real delay in ns

  time start_time, end_time;         // Variables to track the time before and after the delay



  initial begin
```

```
    start_time = $time;  // Capture the simulation time before the delay
    #2.71828;             // Apply a delay of 2.71828 time units (interpreted as 10ns per time unit)
    end_time = $time;    // Capture the time after the delay


    $display("Start time: %0t", start_time);
    $display("End time: %0t", end_time);
    $display("Delay in simulation time: %0t", end_time - start_time);
    $display("Actual calculated delay (without truncation): %0f ns", actual_delay);
  end
endmodule
```

Result Output :

```
[2024-09-26 06:39:57 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
Start time: 0
End time: 300
Delay in simulation time: 300
Actual calculated delay (without truncation): 27.182800 ns
Done
```

Q11: Retype "signedNumber" module, and observe the running results on the "WK#2" handout.
Ans:

```
module signedNumber;
  reg [31:0] a;


  initial begin
```

```
    a = 14'h1234;                    // Assigning a positive value

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


    a = -14'h1234;                   // Assigning a negative value

    $display("Current Value of a = %h", a); // Result of this will be checked


    a = 32'hDEAD_BEEF;               // Assigning a positive value

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


    a = -32'hDEAD_BEEF;              // Assigning a negative value

    $display("Current Value of a = %h", a); // Result of this will be checked


    #10 $finish;

  end

endmodule
```

Result Output :

```
[2024-09-26 06:50:13 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer
vvp a.out

Current Value of a = 00001234

Current Value of a = ffffedcc

Current Value of a = deadbeef

Current Value of a = 21524111

design.sv:17: $finish called at 10 (1s)

Done
```

Q12: Take “helloWorld.v” as an example, write program to print double quote, percent character and @ character in ASCII code number.

Ans:

```
module printASCII();

   initial begin

      // Print ASCII values for characters

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

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

      $display("ASCII code for @ character = %d", "@");  // ASCII code for @



      // Finish the simulation after a delay

      #5 $finish();

   end

endmodule
```

Result Output :

```
[2024-09-26 06:43:48 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out

ASCII code for double quote (") =  34

ASCII code for percent character (%) =  37

ASCII code for @ character =  64

design.sv:9: $finish called at 5 (1s)

Done
```

Q13: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.

Ans:

```
module format_example();


  // Declare some variables of different types
  reg [7:0] bin_var = 8'b10101010;   // Binary variable
  reg [31:0] dec_var = 123456789;    // Decimal variable
  real float_var = 12345.6789;       // Floating-point variable
  reg [15:0] hex_var = 16'hABCD;     // Hexadecimal variable
  reg [7:0] char_var = 65;           // ASCII code for character 'A'
  time time_var;                     // Time variable for simulation
  reg [7:0] oct_var = 8'o77;         // Octal variable
  reg [15:0] short_var = 16'd25;     // Decimal variable for zero-padded output


  initial begin
    // Display binary format
    $display("Binary format (%%b): %b", bin_var);


    // Display ASCII character (%%c)
    $display("Character format (%%c): %c", char_var);


    // Display decimal format (%%d)
    $display("Decimal format (%%d): %d", dec_var);


    // Display decimal with zero padding (%%0d)
```

```
$display("Zero-padded decimal (%%0d): %0d", short_var);


// Display scientific notation (%%e)

$display("Scientific notation (%%e): %e", float_var);


// Display floating-point format (%%f)

$display("Floating-point format (%%f): %f", float_var);


// Display floating-point with field width and precision (%%6.2f)

$display("Formatted float (%%6.2f): %6.2f", float_var);


// Display general floating-point format (%%g)

$display("General floating-point (%%g): %g", float_var);


// Display hexadecimal format (%%h)

$display("Hexadecimal format (%%h): %h", hex_var);


// Display octal format (%%o)

$display("Octal format (%%o): %o", oct_var);


// Display string format (%%s)

$display("String format (%%s): %s", "Hello, Verilog!");


// Capture current simulation time and display using %%t and %%0t
```

```
    time_var = $time;

    $display("Simulation time (%%t): %t", time_var);

    $display("Zero-padded simulation time (%%0t): %0t", time_var);


    // End the simulation

    #5 $finish;

  end

endmodule
```

Result Output :

```
[2024-09-26 06:46:14 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  &&
unbuffer vvp a.out

Binary format (%b): 10101010

Character format (%c): A

Decimal format (%d):  123456789

Zero-padded decimal (%0d): 25

Scientific notation (%e): 1.234568e+04

Floating-point format (%f): 12345.678900

Formatted float (%6.2f): 12345.68

General floating-point (%g): 12345.7

Hexadecimal format (%h): abcd

Octal format (%o): 077

String format (%s): Hello, Verilog!

Simulation time (%t):                    0

Zero-padded simulation time (%0t): 0

design.sv:53: $finish called at 5 (1s)
```

Done

Q14:Compare $display, $write and $monitor system tasks by a program, and give the running result.

Ans:

```
module compare_display_write_monitor();


  reg [3:0] count; // A simple 4-bit register for counting


  initial begin

    count = 0;


    // Using $monitor to automatically display the changes in count

    $monitor("Count (using $monitor): %0d", count);


    // Initial display using $display

    $display("Initial Count (using $display): %0d", count);


    // Increment count with delays

    #5 count = count + 1;  // Increment after 5 time units

    $display("After 5 time units (using $display): %0d", count);


    #5 count = count + 1;  // Increment after another 5 time units

    $write("After another 5 time units (using $write): %0d\n", count); // $write does not
add a newline
```

```
    #5 count = count + 1;  // Increment again

    $display("After 15 time units (using $display): %0d", count);


    #5 count = count + 1;  // Increment again

    $display("Final Count (using $display): %0d", count);


    #5 $finish; // End the simulation

  end

endmodule
```

Result Output :

```
[2024-09-26 06:48:28 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer
vvp a.out

Initial Count (using $display): 0

Count (using $monitor): 0

After 5 time units (using $display): 1

Count (using $monitor): 1

After another 5 time units (using $write): 2

Count (using $monitor): 2

After 15 time units (using $display): 3

Count (using $monitor): 3

Final Count (using $display): 4

Count (using $monitor): 4

design.sv:27: $finish called at 25 (1s)

Done
```