DATA-TYPES Assignment

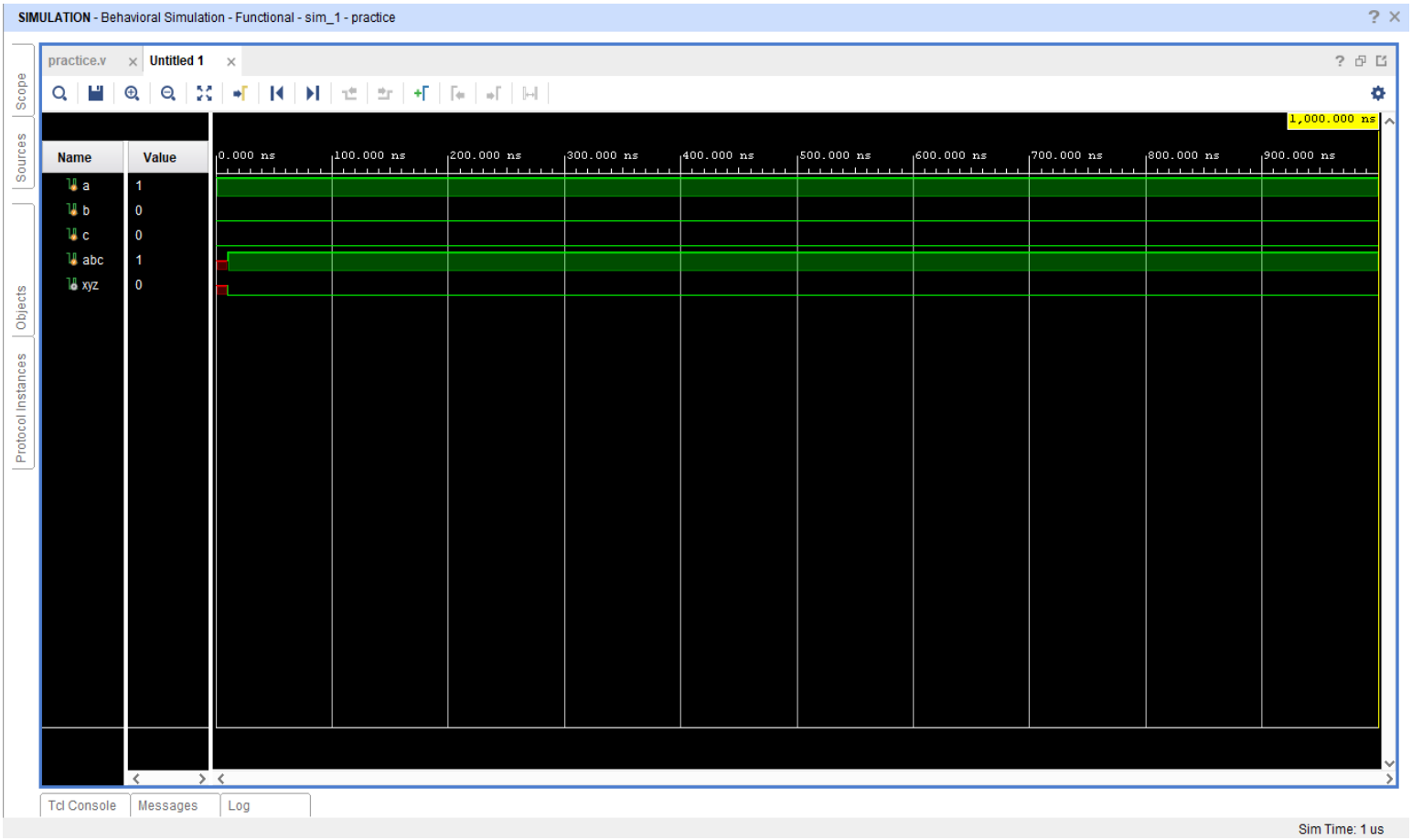
Snippet #1:-

(RTL/Testbench)

```
module practice;
reg a,b,c;
reg abc;
wire xyz;

initial begin
a = 1'b1;
b = 1'b0;
c = 1'b0;
#10;
abc = a|b|c;
end
assign xyz = ~abc;
initial
  $monitor("Value of abc = %0d, xyz = %0d",abc, xyz);
endmodule
```

Output Snippet #1:-

```
# run 1000ns
Value of abc = x, xyz = x
Value of abc = 1, xyz = 0
INFO: [USF-XSim-96] XSim completed. Design snapshot 'practice_behav' loaded.
```



---------##---------

Snippet #2:-

(RTL/Testbench)

```
module practice;

integer i, i_r;

real r, r_i;

time t;


initial begin

i = 8'b0100_1000;  // d = 72

r = 13.563;

#10;

t = $time;

i_r = r;  //real assigned to integer

r_i = i; //integer assigned to real

$display("Value of i = %0d", i);

$display("Value of r = %0d", r);
```

$display("Value of i_r = %0d", i_r);

$display("Value of r_i = %0d", r_i);

end

endmodule

Output Snippet #2:-

run 1000ns

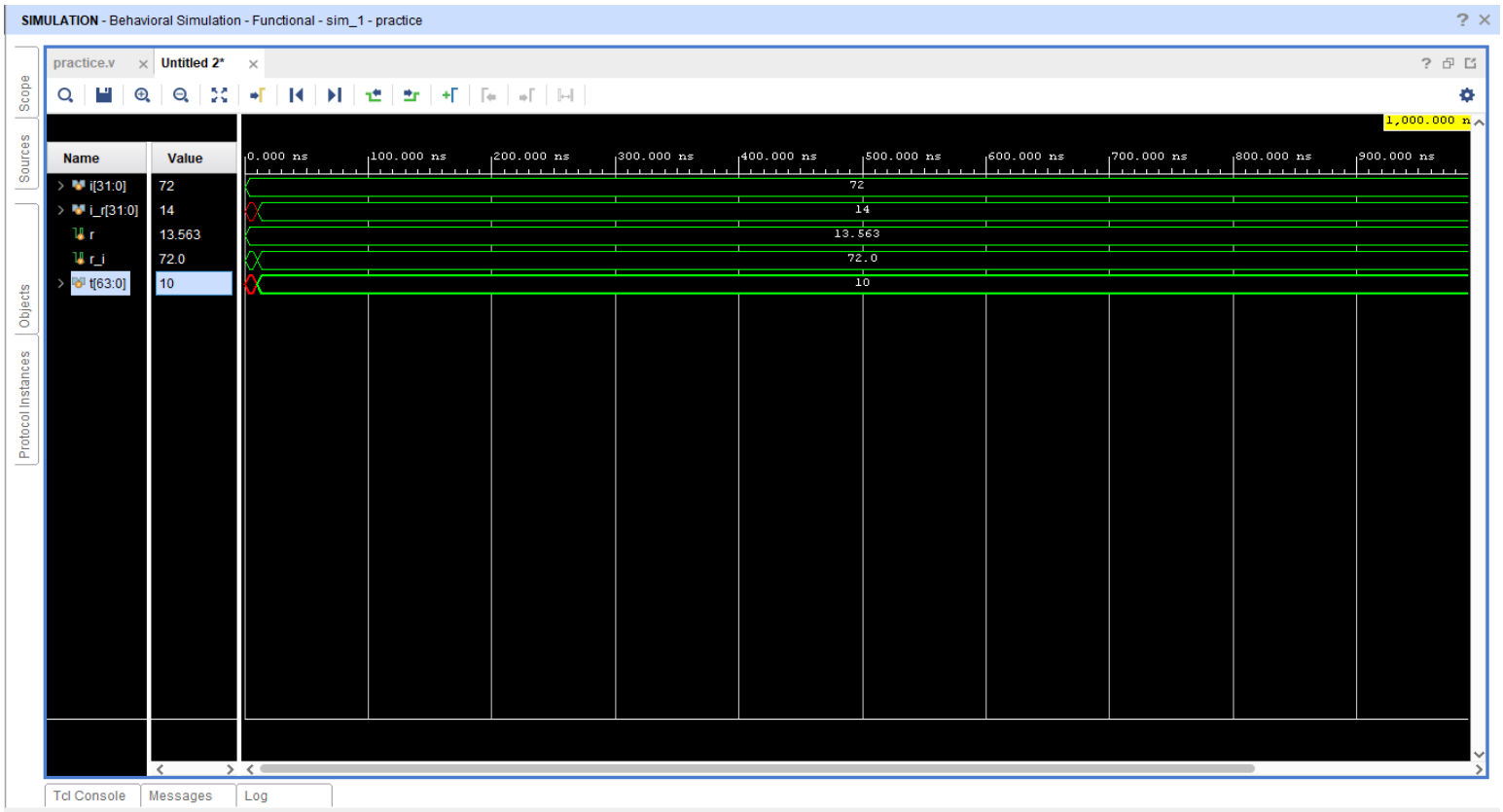
Value of i = 72

Value of r = 14

Value of i_r = 14

Value of r_i = 72

INFO: [USF-XSim-96] XSim completed. Design snapshot 'practice_behav' loaded.



----------##----------

Snippet #3:-

```
module testbench;

  reg [8*20:1] str1; //20 bytes, each byte for a character in a string
  reg [8*5:1]  str2; //5 bytes
  reg [8*24:1] str3; //24 bytes
  initial begin
    str1 = "Verilog Practice Session";
    str2 = "Verilog Practice Session";
    str3 = "Verilog Practice Session";

    $display ("str1 = %s", str1);
    $display ("str2 = %s", str2);
    $display ("str3 = %s", str3);
  end
endmodule
```

Output Snippet #3:-

```
# run 1000ns
str1 = log Practice Session
str2 = ssion
str3 = Verilog Practice Session
INFO: [USF-XSim-96] XSim completed. Design snapshot 'testbench_behav' loaded.
----------##########################################################################################----------
```

Snippet #4:-

```
module test;
  reg signed [3:0]a;
  reg [3:0]b;
  initial begin
    a = -24/3;
    b = -24/3;
    #10;
```

```
    $display(a);
    $display(b);
  end
endmodule
```

Output Snippet #4:-

```
# run 1000ns
 -8
 8
INFO: [USF-XSim-96] XSim completed. Design snapshot 'test_behav' loaded.
```

(since ‘a’ variable is declared as a signed reg variable, it displays the signed value of the division ie. -8. On the other hand, the variable ‘b’ is an unsigned reg variable, thus, it stores and displays the 2’s complemented form of the final output of the RHS ie. 2’s complement of -8 is 8.)

----------##----------

Snippet #5:-

```
module practice;
reg [3:0]array1[0:3][0:1];
reg [3:0]array2[0:7];
integer i,j;

initial begin
  for(i=0;i<=7;i=i+1) begin
       array2[i] = i;
       #10;
     $display("ARRAY2 values: %0d",array2[i]);
    end
    for(i = 0; i < 4; i=i+1) begin
      for(j = 0; j < 2; j=j+1) begin
       array1[i][j] = i + j;
       $display("ARRAY1[%0d][%0d] = %0d", i, j, array1[i][j]);
      end
    end
end
endmodule
```

Output Snippet #5:-

```
Time resolution is 1 ps
ARRAY2 values: 0
ARRAY2 values: 1
ARRAY2 values: 2
ARRAY2 values: 3
ARRAY2 values: 4
ARRAY2 values: 5
ARRAY2 values: 6
ARRAY2 values: 7
ARRAY1[0][0] = 0
ARRAY1[0][1] = 1
```

ARRAY1[1][0] = 1

ARRAY1[1][1] = 2

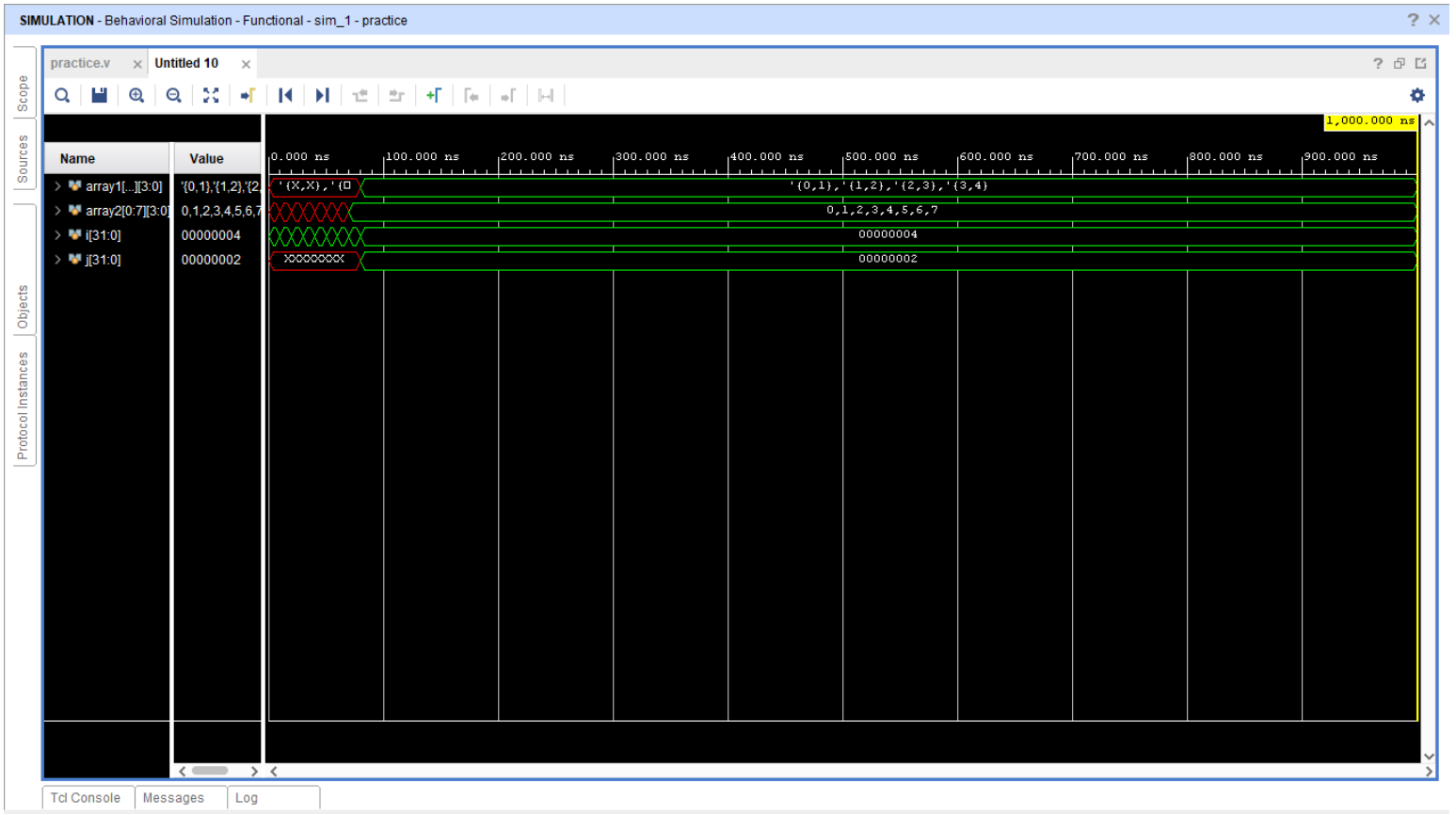
ARRAY1[2][0] = 2

ARRAY1[2][1] = 3

ARRAY1[3][0] = 3

ARRAY1[3][1] = 4

relaunch_xsim_kernel: Time (s): cpu = 00:00:05 ; elapsed = 00:00:05 . Memory (MB): peak = 4228.543 ; gain = 0.



----------##----------

Snippet #6:-

module practice;

parameter array_width = 4;

parameter array_depth = 16;

reg [array_width-1:0]array_1[0:array_depth-1];

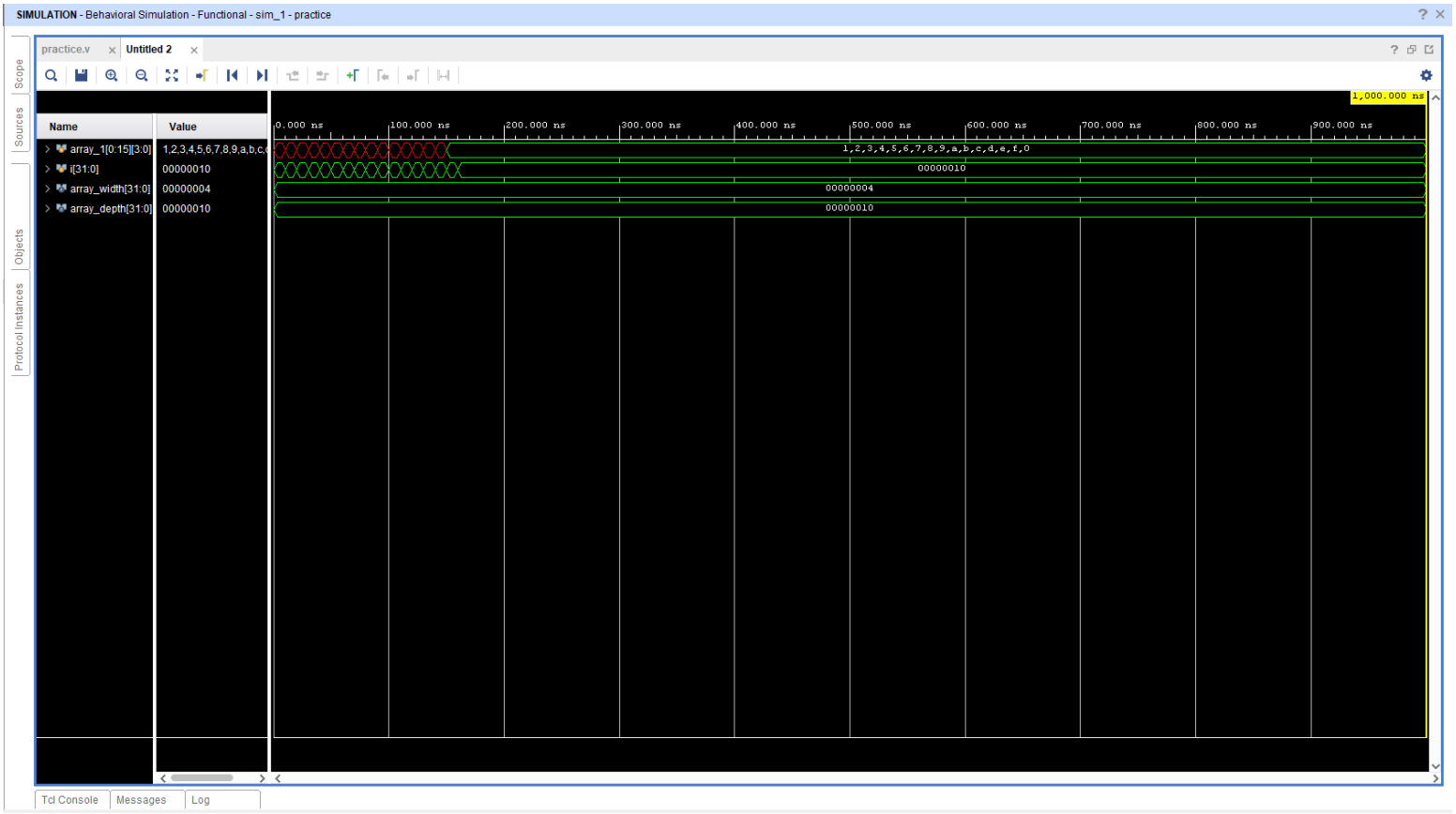
integer i;

initial begin

```
for(i=0;i<array_depth;i=i+1)
begin
array_1[i] = i +1;
#10;
$display("Array_1[%0d]=%0d",i,array_1[i]);
end
end

endmodule
```

Output Snippet #6:-

```
Time resolution is 1 ps
Array_1[0]=1
Array_1[1]=2
Array_1[2]=3
Array_1[3]=4
Array_1[4]=5
Array_1[5]=6
Array_1[6]=7
Array_1[7]=8
Array_1[8]=9
Array_1[9]=10
Array_1[10]=11
Array_1[11]=12
Array_1[12]=13
Array_1[13]=14
Array_1[14]=15
Array_1[15]=0
relaunch_sim: Time (s): cpu = 00:00:01 ; elapsed = 00:00:06 . Memory (MB): peak = 885.012 ; gain = 0.355
```



OPERATORS Assignment

Snippet #1:- (Arithmetic Operator)

(RTL/Testbench)

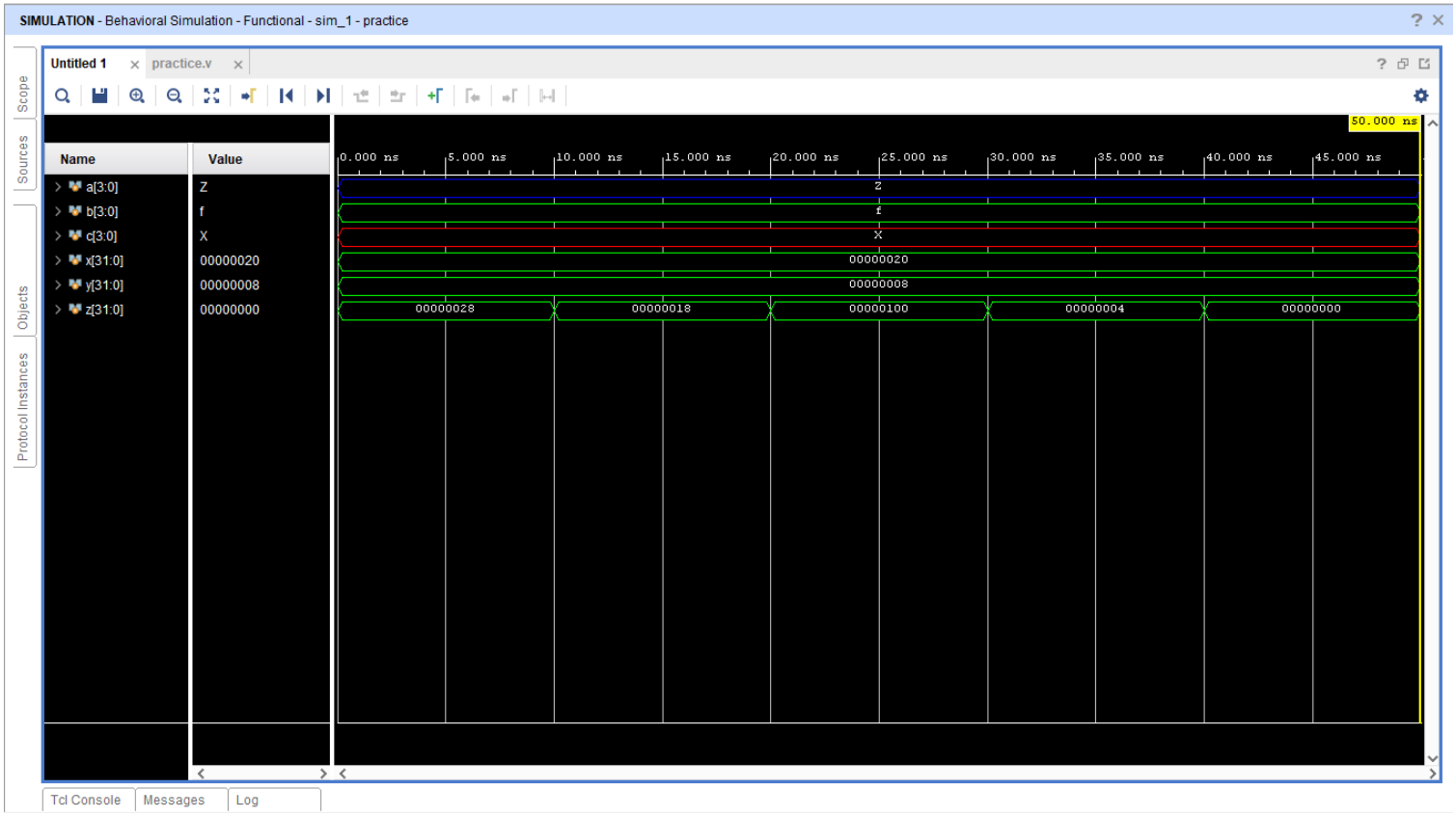
```
module practice;

reg [3:0]a, b,c;

integer x,y,z;


initial begin

a = 4'b1z01;

b = 4'b1111;

x = 32;

y = 8;


c = a + b;

$display("Value of c is %0b",c);

c = a*b;

$display("Value of c is %0b",c);

c = a/b;
```

```
$display("Value of c is %0b",c);
c = a%b;
$display("Value of c is %0b",c);

$display("_____");

z = x + y;
$display("Value of z is %0d",z);
#10;
z = x - y;
$display("Value of z is %0d",z);
#10;
z = x * y;
$display("Value of z is %0d",z);
#10;
z = x / y;
$display("Value of z is %0d",z);
#10;
z = x % y;
$display("Value of z is %0d",z);
#10 $finish;
end
endmodule
```

Output Snippet #1:-

```
Time resolution is 1 ps
Value of c is xxxx
Value of c is xxxx
Value of c is xxxx
Value of c is xxxx

_____

Value of z is 40
Value of z is 24
Value of z is 256
Value of z is 4
Value of z is 0
```

$finish called at time : 50 ns



Snippet #2:- (Logical Operators)

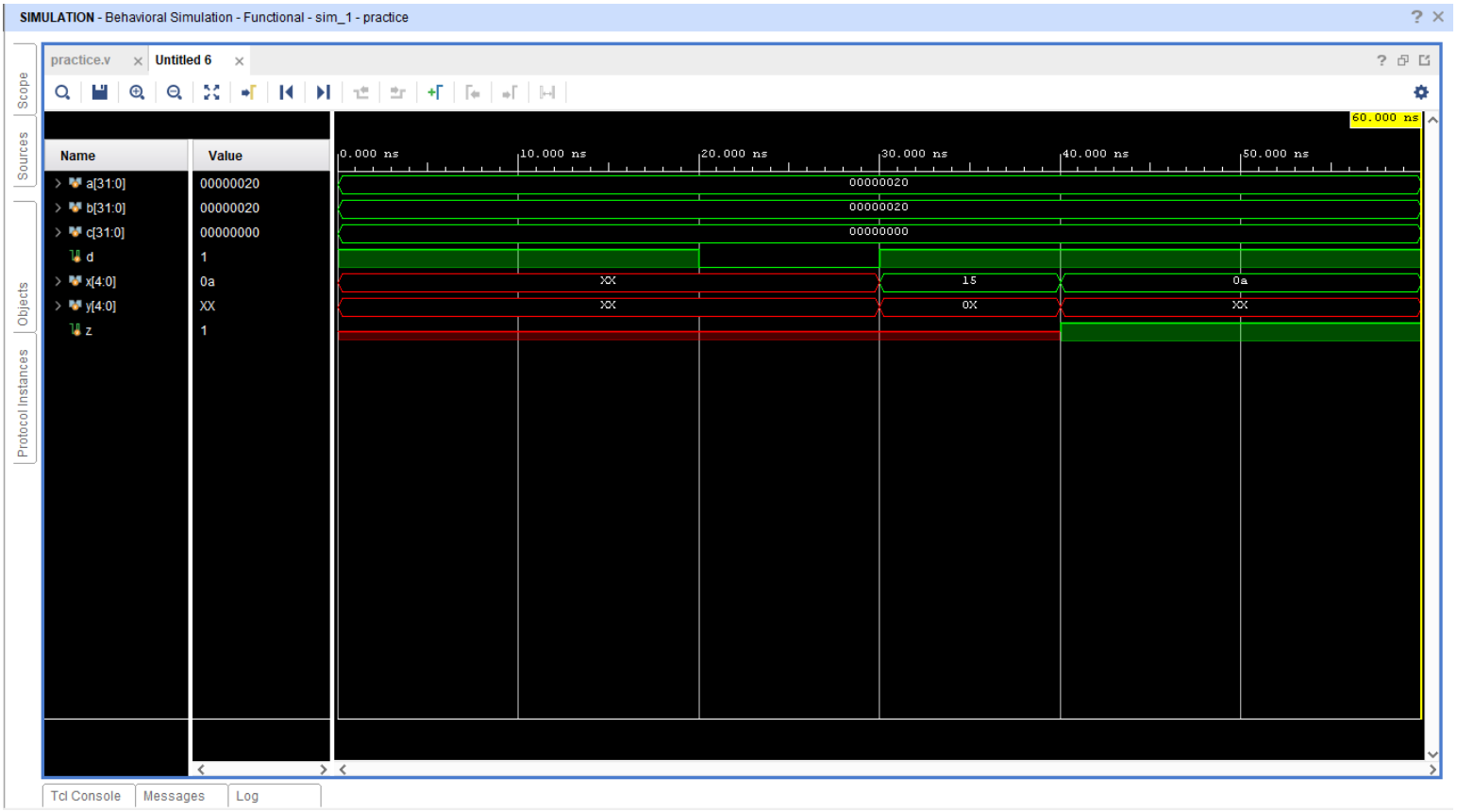
(RTL/Testbench)

```
module practice;

integer a,b,c;

reg d;

reg [4:0]x,y;

reg z;


initial begin

a = 32;

b = 32;

c = 0;


d = a&&b;   //logical AND for this case will give "True(1)" as the output

$display("Value of d is %0b",d);

#10;

d =  a||b;   //logical OR for this case will give "True(1)" as the output

$display("Value of d is %0b",d);
```

```
#10;
d = a&&c;   //logical AND for this case will give "False(0)" as the output
$display("Value of d is %0b",d);
#10;
d = !c;
$display("Value of d is %0b",d);

$display("-----------------------------------------");

x = 5'b10101;
y = 5'b000x0;
z = x&&y;   //logical AND for this case will give "Unknown" as the output since the bits in variable 'y' are '0' and 'x'
$display("Value of z is %0b",z);
#10;
x = 5'b01010;
y = 5'bx0x1x;

z = x&&y;   //logical AND for this case will give "True" as the output since the bits in variable 'y' are contain at least one bit as '1'
$display("Value of z is %0b",z);
#20 $finish;
end

endmodule
```

Output Snippet #2:-

```
Time resolution is 1 ps
Value of d is 1
Value of d is 1
Value of d is 0
Value of d is 1
-----------------------------------------
Value of z is x
Value of z is 1
$finish called at time : 60 ns
```



---------##---------

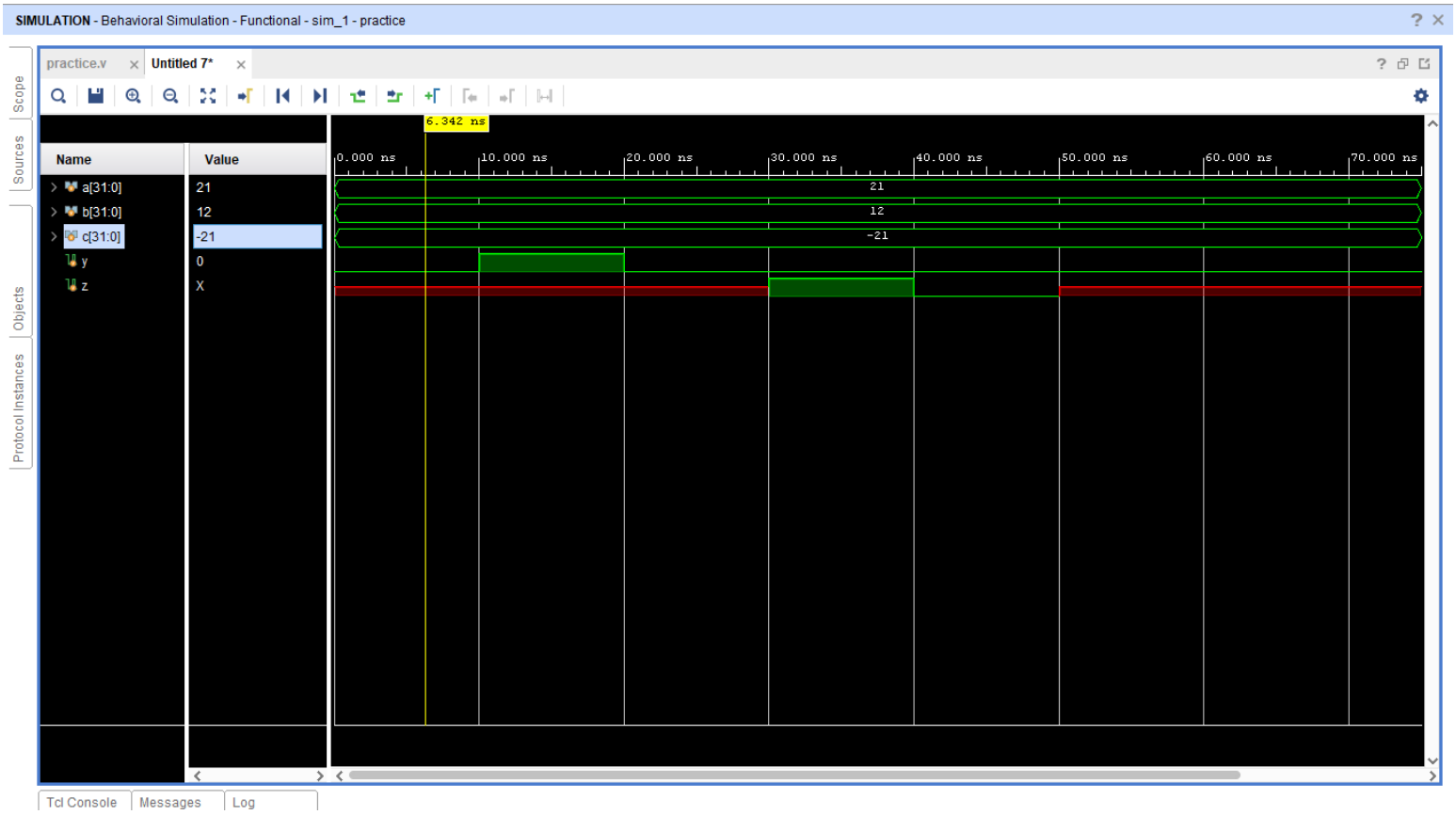
Snippet #3:- (Relational Operators)

```
module practice;

integer a,b,c;

reg y;

reg [3:0]a1, b1,c1;

reg z;


initial begin

a = 21;

b = 12;

c = -21;


a1 = 4'b1010; b1 = 4'b1101; c1 = 4'b1xxx;


y = a<=b;

$display("a<=b relation is logic:%0b", y);

#10;
```

```
y = a>=b;
$display("a>=b relation is logic:%0b", y);
#10;
y = a<c;
$display("a<=c relation is logic:%0b", y);
#10;
$display("-----------------------------------------------------");
z = a1<=b1;
$display("a1<=b1 relation is logic:%0b", z);
#10;
z = a1>=b1;
$display("a1>=b1 relation is logic:%0b", z);
#10;
z = a1<c1;
$display("a1<=c1 relation is logic:%0b", z);
#25 $finish;
end
endmodule
```

Output Snippet #3:-

```
Time resolution is 1 ps
a<=b relation is logic:0
a>=b relation is logic:1
a<=c relation is logic:0
-----------------------------------------------------
a1<=b1 relation is logic:1
a1>=b1 relation is logic:0
a1<=c1 relation is logic:x
$finish called at time : 75 ns
```



---------##---------

Snippet #4:- (Equality Operators)

```
module practice;

integer a,b;

reg c;

reg [3:0]w,x,y,z,m,n;

reg f;


initial begin

a = 21;

b = 12;

w = 4'b1010; x = 4'b1001; y = 4'b1xxx; z = 4'b1xxz;

m = 4'b1xxx; n = 4'b1xxz;
```

```
c = a==b;
$display("a==b relation is logic:%0b", c);
#10;
c = a!=b;
$display("a!=b relation is logic:%0b", c);
#10;
$display("-----------------------------------------------------");
f = w==x;
$display("w==x relation is logic:%0b", f);
#10;
f = w==y;
$display("w==y relation is logic:%0b", f);
#10;
f = w===x;
$display("w===x relation is logic:%0b", f);
#10;
f = w===y;
$display("w===y relation is logic:%0b", f);

f = m===y;
$display("m===y relation is logic:%0b", f);

f = n===z;
$display("n===z relation is logic:%0b", f);
f = m!==n;
$display("m!==n relation is logic:%0b", f);
#25 $finish;
end
endmodule
```

Output Snippet #4:-

```
Time resolution is 1 ps
a==b relation is logic:0
a!=b relation is logic:1
-----------------------------------------------------
```

w==x relation is logic:0

w==y relation is logic:x

w===x relation is logic:0

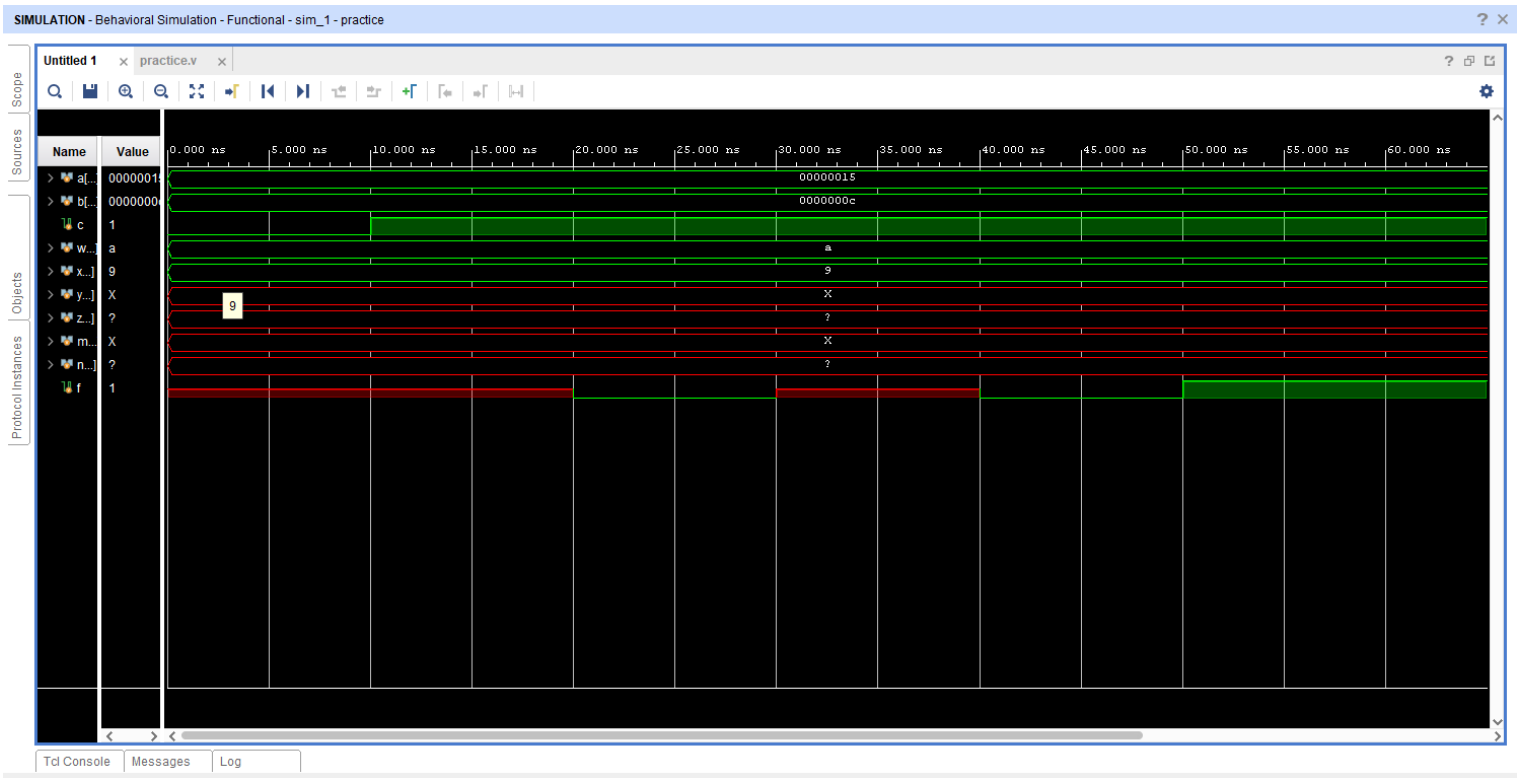
w===y relation is logic:0

m===y relation is logic:1

n===z relation is logic:1

m!==n relation is logic:1

$finish called at time : 75 ns



----------##----------

Snippet #5:- (Bitwise Operators)

module practice;

reg[3:0]a,b,c,d;

reg [3:0]f;

initial begin

a = 4'b0101;

b = 4'b1010;

c = 4'b010x;

d = 4'b010z;

```
f = a&b;
$display("Value of a&b is %b", f); //4'b0000
#10;

f = a|b;
$display("Value of a|b is %b", f); //4'b1111
#10;

f = a^b;
$display("Value of a^b is %b", f); //4'b1111
#10;

f = a^~b;
$display("Value of a^`b is %b", f); //4'b0000
#10;

f = ~a;
$display("Value of ~a is %b", f); //4'b1010
#10;

f = a&c;
$display("Value of a&c is %b", f); //4'b010x
#10;
f = a&d;
$display("Value of a&d is %b", f); //4'b010x
#10;
f = c&d;
$display("Value of c&d is %b", f); //4'b010x
#10 $finish;
end
endmodule
```

Output Snippet #5:-

```
Time resolution is 1 ps
Value of a&b is 0000
```

Value of a|b is 1111

Value of a^b is 1111

Value of a^`b is 0000

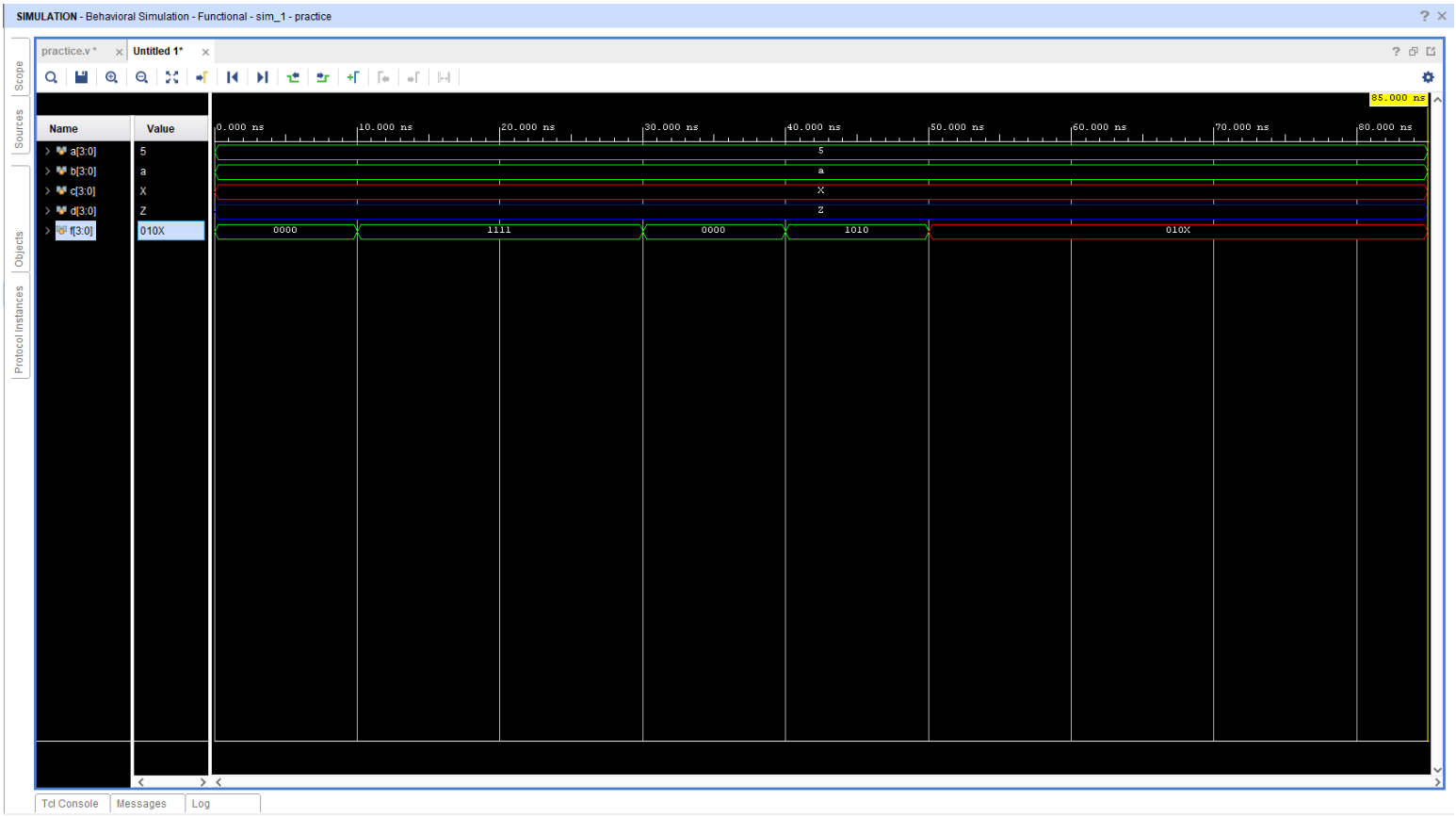
Value of ~a is 1010

Value of a&c is 010x

Value of a&d is 010x

Value of c&d is 010x

$finish called at time : 85 ns



----------##----------

Snippet #6:- (Reduction Operators)

module practice;

reg[3:0]a,b,c,d;

reg f;

initial begin

a = 4'b0101;

c = 4'bxx01;

d = 4'b010z;

f = &a;

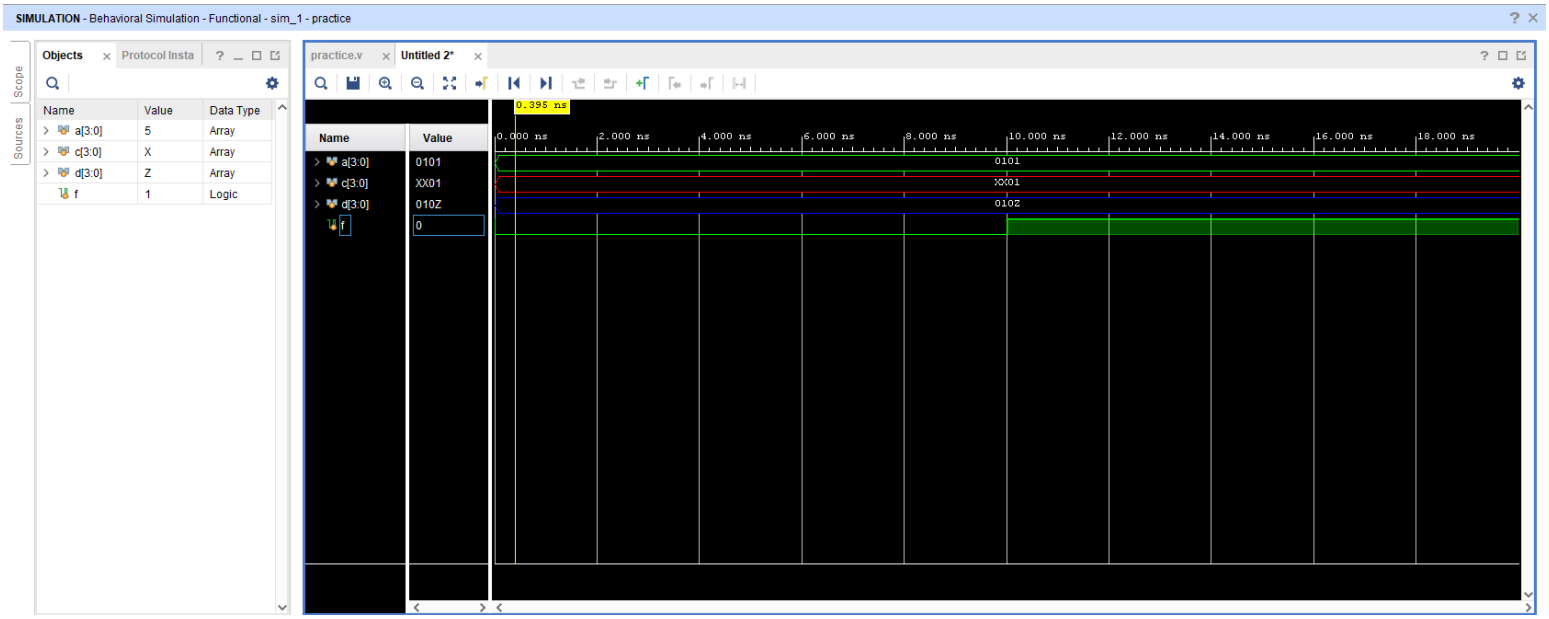
$display("Value of &a is %b",f);

#10;

```
f = ~&a;
$display("Value of ~&a is %b",f);

f = &c;
$display("Value of &c is %b",f);

f = ~&d;
$display("Value of ~&d is %b",f);

f = ^c;
$display("Value of ^c is %b",f);

f = ~^a;
$display("Value of ~^a is %b",f);

#10 $finish;
end
endmodule
```

Output Snippet #6:-

```
Time resolution is 1 ps
Value of &a is 0
Value of ~&a is 1
Value of &c is 0
Value of ~&d is 1
Value of ^c is x
Value of ~^a is 1
$finish called at time : 20 ns
```



----------##---------

Snippet #7:- (Concatenation Operator, Replication Operator, Conditional Operator)

```
module practice;

reg[3:0]a;

reg [3:0]f;

reg [7:0]c;


reg[1:0]s;

reg[3:0]i;

reg y;


initial begin

a = 4'b0101;

$display("------------SHIFT OPERATORS------------");

f = a<<1;

$display("Left shift by 1 yields: %b",f);

#10;

f = a>>1;

$display("Right shift by 1 yields: %b",f);

#10;

f = a<<2;

$display("Left shift by 2 yields: %b",f);
```

```
#10;

$display("---------------------------------------");
$display("--------CONCATENATION OPERATOR--------");
c = {f,a};
$display("Concatenation result is %b",c);
#10;

$display("-------------------------------------");
$display("--------REPLICATION OPERATOR--------");
c = {2{a<<2}};
$display("Replication result is %b",c);
#10;

$display("-------------------------------------");
$display("--------CONDITIONAL OPERATOR--------");
s[0] = 1'b1;
s[1] = 1'b0;
i = 4'b0101;

y = s[1] ? (s[0] ? i[3]:i[2]):(s[0] ? i[1]:i[0]);  //Conditional Operator used for a 4:1 MUX
$display("For the input [i]: %b, output y is %b",i,y);

#10 $finish;
end
endmodule
```

Output Snippet #7:-

```
Time resolution is 1 ps
------------SHIFT OPERATORS------------
Left shift by 1 yields: 1010
Right shift by 1 yields: 0010
Left shift by 2 yields: 0100
---------------------------------------
--------CONCATENATION OPERATOR--------
Concatenation result is 01000101
```

--------REPLICATION OPERATOR--------

Replication result is 01000100

--------CONDITIONAL OPERATOR--------

For the input [i]: 0101, output y is 0

$finish called at time : 60 ns

----------##----------

Snippet #8:-

Q1] There are two inputs and one output. One input is of 3 bits wide and the other is 1 bit wide.

You need to perform multiplication by 2 and division by 2 operation on both the inputs by using shift operators only.

Solution:

RTL: -

```
module practice(input [2:0]number_1, input number_2, mul_ctrl, div_ctrl, output reg[3:0]product, division);
reg[3:0]mul_number_1, div_number_1;
reg[2:0]mul_number_2, div_number_2;

  always@(*) begin
//multiplication operation is performed by left shifting the binary number by one bit.
mul_number_1 = number_1<<1;
mul_number_2= number_2<<1;
//division operation is performed by right shifting the binary number by one bit.
div_number_1= number_1>>1;
div_number_2= number_2>>1;
product = mul_ctrl ? mul_number_1:mul_number_2;
division = div_ctrl ? div_number_1:div_number_2;
end
endmodule
```

TESTBENCH: -

```
module practice_tb;
reg [2:0]number_1;
```

```
reg number_2, mul_ctrl, div_ctrl;
wire [3:0]product, division;
integer i;

practice DUV(number_1, number_2, mul_ctrl, div_ctrl, product, division);

initial begin

  for(i =0; i<8; i = i+1) begin
number_1 =i;//3'b100;
number_2 =i;// 1'b1;
mul_ctrl = 1;
div_ctrl = 1;
#10;
end
#20;
  $display("-------------------------------");

for(i =0; i<8; i = i+1) begin
number_1 =i;//3'b100;
number_2 =i;// 1'b1;
mul_ctrl = 0;
div_ctrl = 0;
#10;
end
#110 $finish;
end
initial
  $monitor("product is %b, division is %b", product, division);

endmodule
```

OUTPUT:-

```
# run 1000ns
product is 0000, division is 0000
product is 0010, division is 0000
```

product is 0100, division is 0001

product is 0110, division is 0001

product is 1000, division is 0010

product is 1010, division is 0010

product is 1100, division is 0011

product is 1110, division is 0011

---

product is 0000, division is 0000

product is 0010, division is 0000

product is 0000, division is 0000

product is 0010, division is 0000

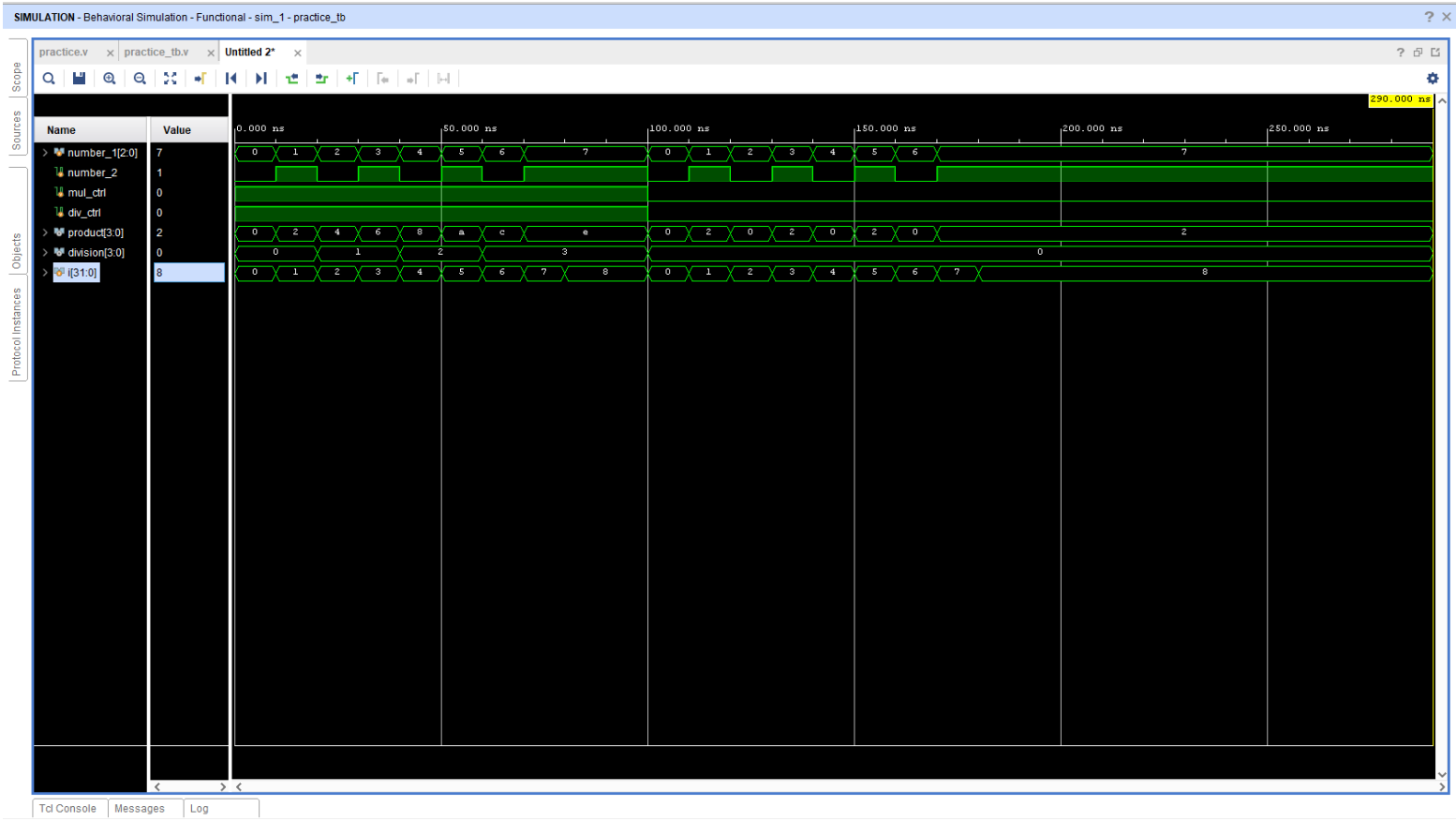
product is 0000, division is 0000

product is 0010, division is 0000

product is 0000, division is 0000

product is 0010, division is 0000

$finish called at time : 290 ns



PROCESSES & STRUCTURED PROCESSES Assignment

Snippet #1:- (CONTINUOUS ASSIGNMENT with and without DELAY)

```
module practice;

 wire [3:0]result_and;
 wire[3:0]result_or;
 wire[3:0]result_xor;
 reg[3:0]result_xnor;
 reg [3:0]number_1, number_2;
 reg ctrl;

 initial begin
 number_1 = 4'b1010;
 number_2 = 4'b1001;
 ctrl = 1'b0;
 result_xnor = ctrl ? (number_1 ~^ number_2):(result_or ~^ result_and);
 #5;
 ctrl = 1'b1;
 result_xnor = ctrl ? (number_1 ~^ number_2):(result_or ~^ result_and);
 #100;
 ctrl = 1'b0;
  $monitor("Result for XNOR is :%b",result_xnor);
 end

 assign result_and = number_1 & number_2;
 assign #10 result_or = number_1 | number_2;
 assign #30 result_xor = ctrl ? (number_1^number_2):(result_or ^ result_and);
 initial begin
 $monitor("Result for AND is :%b",result_and);
 $monitor("Result for OR is :%b",result_or);
 $monitor("Result for XOR is :%b",result_xor);
 //$monitor("Result for XNOR is :%b",result_xnor);
```

#300 $finish;

end

endmodule

Output Snippet #1:-

run 1000ns

Result for AND is :1000

Result for OR is :xxxx

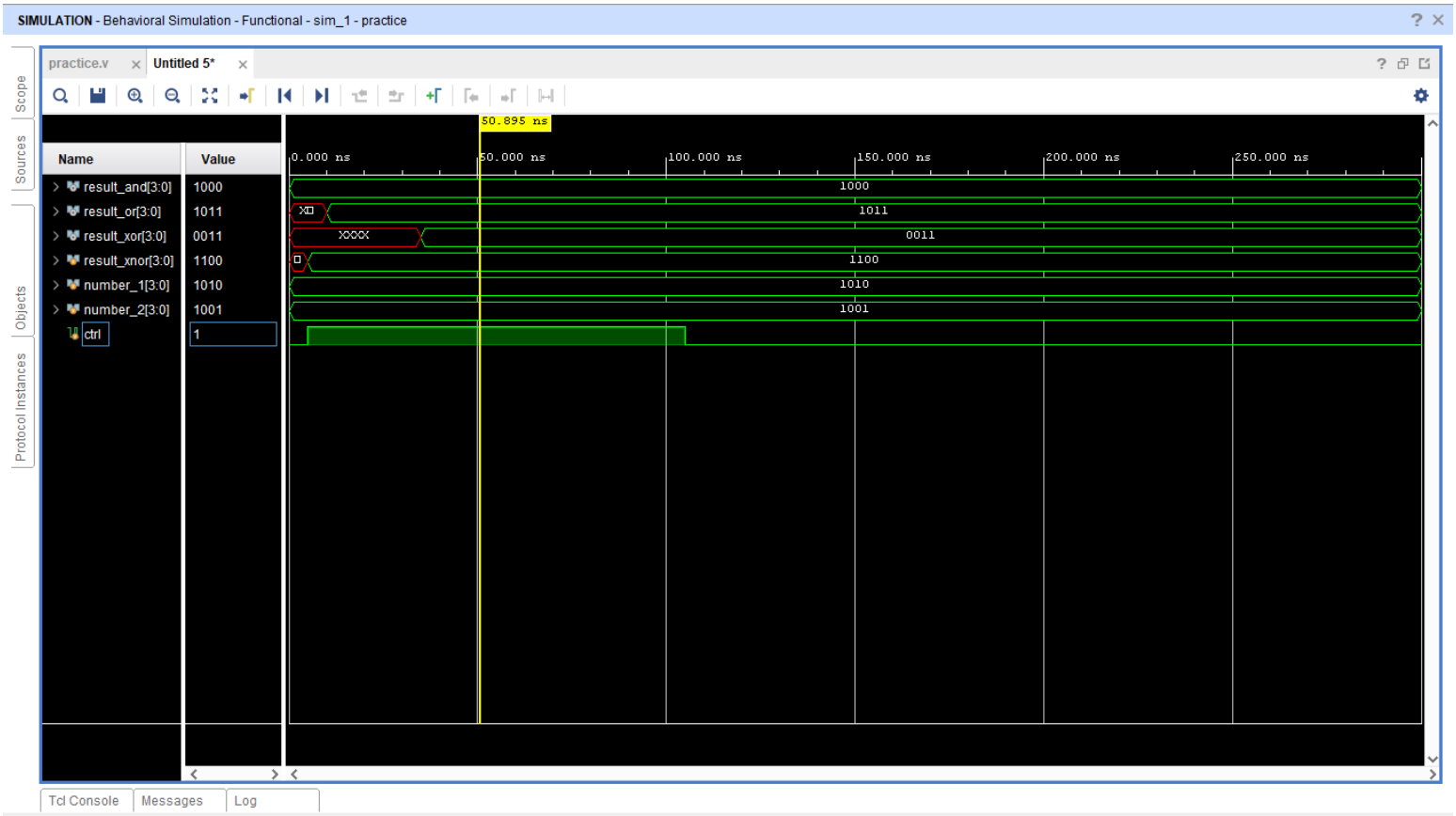
Result for XOR is :xxxx

Result for OR is :1011

Result for XOR is :0011

Result for XNOR is :1100

$finish called at time : 300 ns



---------##---------

Snippet #2:- (PROCEDURAL ASSIGNMENT)

RTL: -

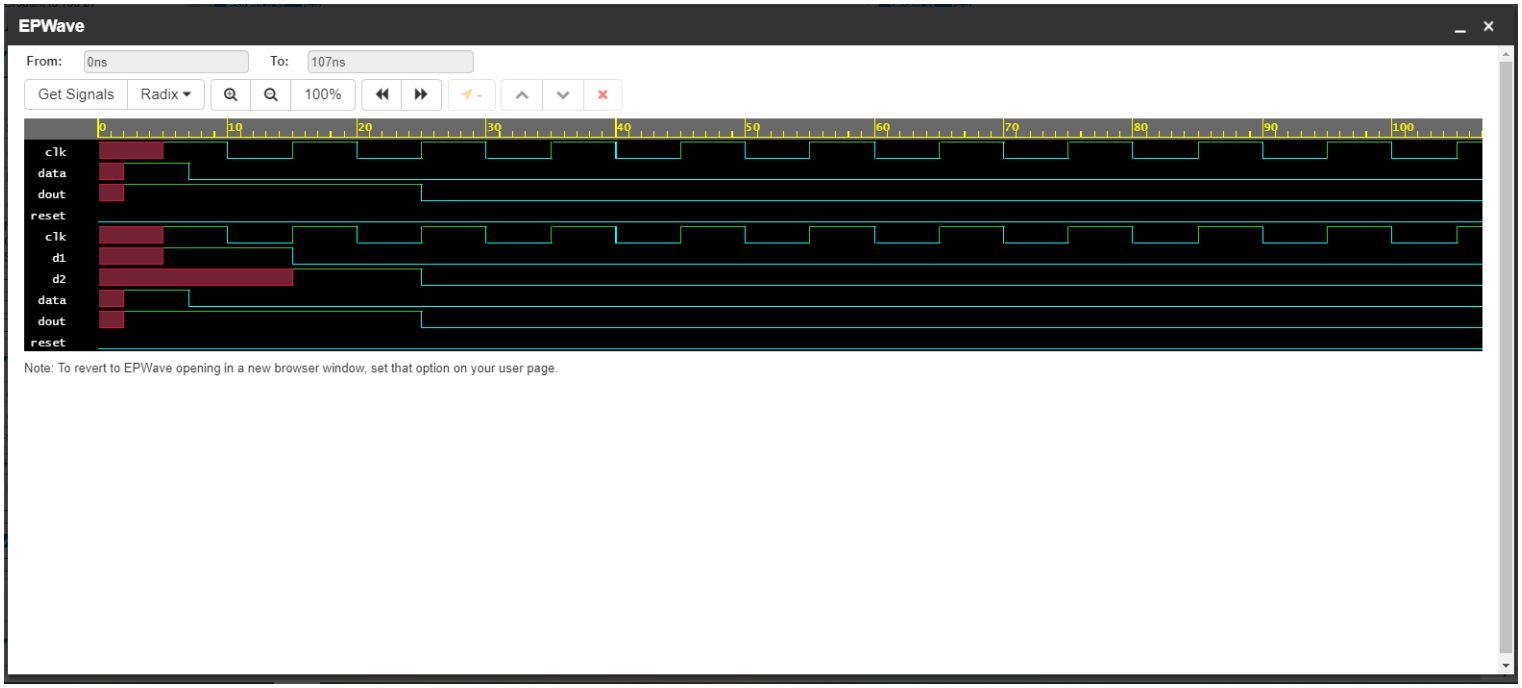
```
module practice(clk,reset,data, dout);
input clk, reset;
input data;
output dout;
reg d1,d2;
always@(posedge clk)
begin
if(reset)
begin
d1 <= 0;
d2 <= 0;
end

else
begin
d1<=data;
d2<=d1;
end
end
assign dout = data|d1|d2;
endmodule
```

TESTBENCH: -

```
module practice_tb;
reg clk,reset,data;
reg dout;
practice DUV(clk,reset,data,dout);

always begin
#5clk = 1'b1;
#5 clk = ~clk;
```

```
end

initial begin
reset = 1'b0;
#2
reset = 1'b0;
data = 1'b1;
#5;
data = 1'b0;
//#3;
//data = 1'b1;
#100 $finish;
end

initial begin
$monitor("DOUT value is: %b",dout);
$dumpfile("dump.vcd"); $dumpvars;
end
endmodule
```

Output Snippet #2:-



---------##---------

Snippet #3:-

RTL: -

```
module counter(clk, reset, load, din,count);
input clk, reset, load;
input [3:0] din;
output reg[3:0]count;

always@(posedge clk) begin
if(reset) begin
count<=0;
end

else if(load) begin
count <= din;
end

else
count <= count +1'd1;
end

endmodule
```

TESTBENCH: -

```
module counter_tb;
reg clk, reset, load;
reg [3:0] din;
wire [3:0]count;
counter DUV(clk, reset, load, din,count);

always begin
#5 clk = 1'b1;
```

```
#5 clk = ~clk;
end

initial begin

@(negedge clk)
reset = 1'b1;
@(negedge clk)
reset = 1'b0;

#5;
load =1;
din = 3;
#2
load =0;

#5
load =1;
din = 13;
#2
load =0;

end

initial begin
$monitor("Counter output is: %0d", count);
#200 $finish;
end

endmodule
```

Output Snippet #3:-

Counter output is: x

Counter output is: 0

Counter output is: 3

Counter output is: 4

Counter output is: 5

Counter output is: 6

Counter output is: 7

Counter output is: 8

Counter output is: 9

Counter output is: 10

Counter output is: 11

Counter output is: 12

Counter output is: 13

Counter output is: 14

Counter output is: 15

Counter output is: 0

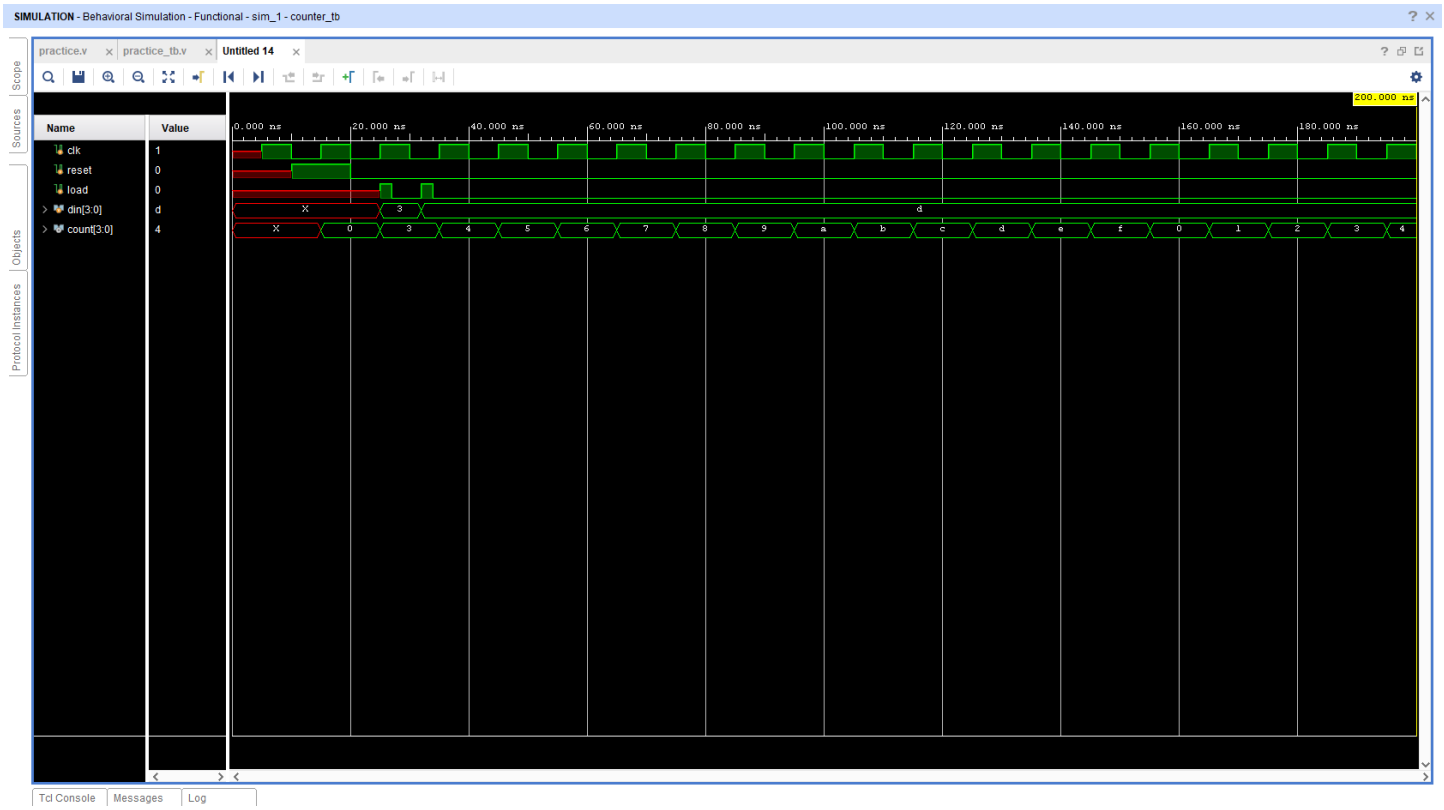
Counter output is: 1

Counter output is: 2

Counter output is: 3

Counter output is: 4

$finish called at time : 200 ns



Snippet #4:- (BLOCKING & NON_BLOCKING ASSIGNMENT)

```
module practice;
  reg [4:0] a, b, c, d, e;

  initial begin
    a = 5'd16;
    $display ("[%0t] a=%0d b=%0d c=%0d", $time, a, b, c);
    b = 5'd15;
    $display ("[%0t] a=%0d b=%0d c=%0d", $time, a, b, c);
    c = 5'd6;
    $display ("[%0t] a=%0d b=%0d c=%0d", $time, a, b, c);
  end

  initial begin
    d = 5'd19;
    $display ("[%0t] d=%0d e=%d", $time, d, e);
        e = 5'd12;
```

```
    $display ("[%0t] d=%0d e=%0d", $time, d, e);
  end

  initial begin //Previous values of a,b,c are retained for this initial block
    a <= 5'd9;
    $display ("[%0t] a=%0d b=%0d c=%0d", $time, a, b, c);
    b <= 5'd10;
    $display ("[%0t] a=%0d b=%0d c=%0d", $time, a, b, c);
    c <= 5'd17;
    $display ("[%0t] a=%0d b=%0d c=%0d", $time, a, b, c);
  end

  initial begin
     a <= 5'd9;
    $display ("[%0t] a=%0d b=%0d c=%0d", $time, a, b, c);
    #10 b <= 5'd10;
    $display ("[%0t] a=%0d b=%0d c=%0d", $time, a, b, c);
    #3 c <= 5'd17;
    $display ("[%0t] a=%0d b=%0d c=%0d", $time, a, b, c);
  end
endmodule
```

Output Snippet #4:-

```
Time resolution is 1 ps
[0] a=16 b=x c=x
[0] a=16 b=15 c=x
[0] a=16 b=15 c=6
[0] d=19 e= x
[0] d=19 e=12
[0] a=16 b=15 c=6
```

[0] a=16 b=15 c=6

[0] a=16 b=15 c=6

[0] a=16 b=15 c=6

[10000] a=9 b=10 c=17

[13000] a=9 b=10 c=17

relaunch_sim: Time (s): cpu = 00:00:00 ; elapsed = 00:00:06 . Memory (MB): peak = 1179.930 ; gain = 0.000

Snippet #5:- (BLOCKING & NON_BLOCKING ASSIGNMENT WITH DELAYS)

```
module practice;
  reg[4:0] a,b,c,d;

  initial begin
    a = 10;
       $display("%t,%d,%d,%d,%d",$time, a,b,c,d);
    #30;
    b <= 12;
       $display("%t,%d,%d,%d,%d",$time, a,b,c,d);
    #5 c = 15;
       $display("%t,%d,%d,%d,%d",$time, a,b,c,d);
    #7 d <= 18;
    $display("%t,%d,%d,%d,%d",$time, a,b,c,d);
    #30 a<=d;
    $display("%t,%d,%d,%d,%d",$time, a,b,c,d);
    #10 b=a;
    $display("%t,%d,%d,%d,%d",$time, a,b,c,d);
  end
endmodule
```

Output Snippet #5:-

Time resolution is 1 ps

0,10, x, x, x

30000,10, x, x, x

35000,10,12,15, x

42000,10,12,15, x

72000,10,12,15,18

82000,18,18,15,18

relaunch_sim: Time (s): cpu = 00:00:01 ; elapsed = 00:00:09 . Memory (MB): peak = 1179.930 ; gain = 0.000

---------##---------

Snippet #6:- (FULL-ADDER)

RTL: -

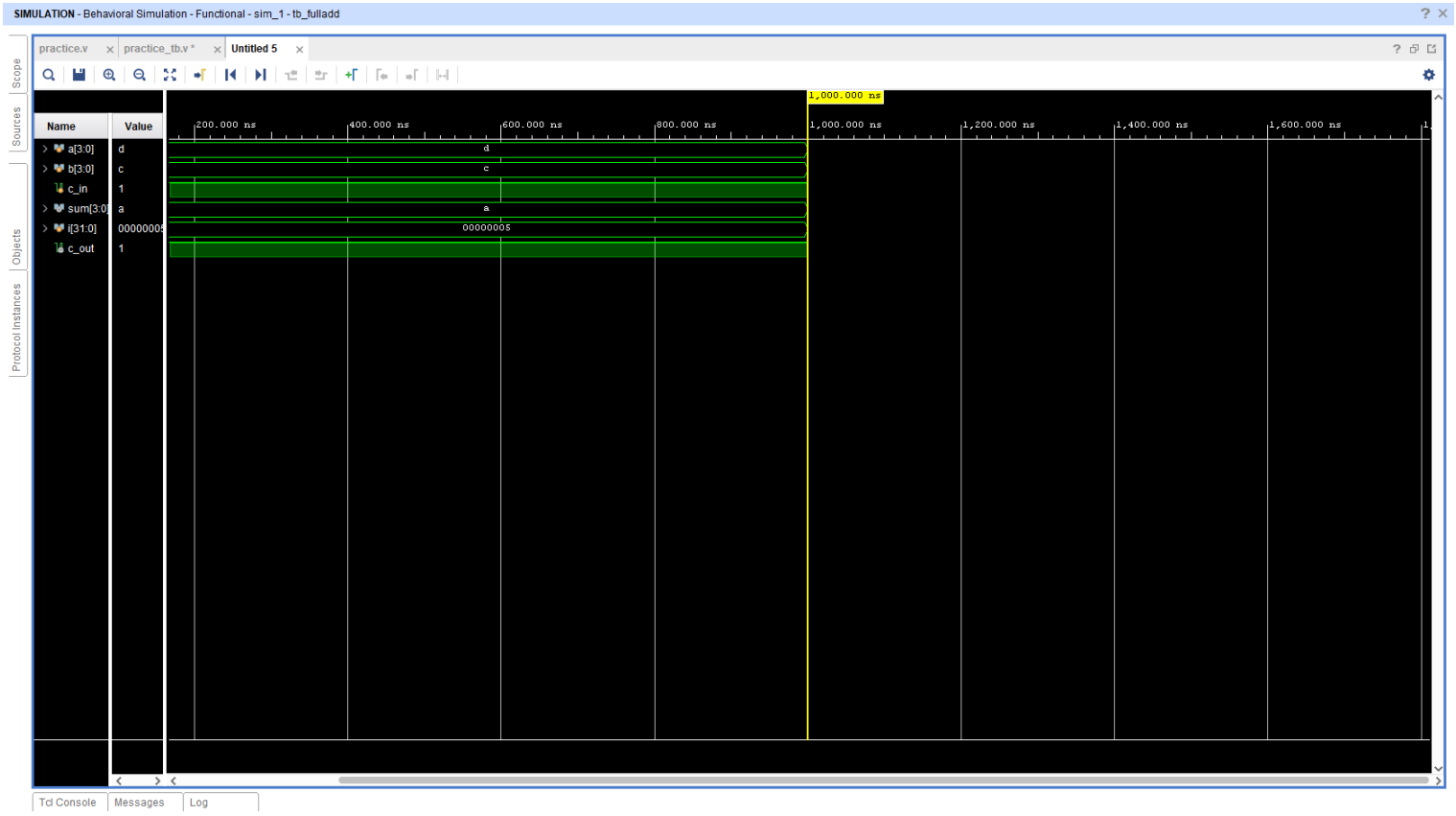
```
module fulladd (  input [3:0] a,
                  input [3:0] b,
                  input c_in,
                  output reg c_out,
                  output reg [3:0] sum);

        always @ (a or b or c_in) begin
        {c_out, sum} = a + b + c_in;
  end
endmodule
```

TESTBENCH: -

```
module tb_fulladd;
   reg [3:0] a;
   reg [3:0] b;
   reg c_in;
   wire [3:0] sum;
   integer i;

   fulladd  fa0 ( .a (a),
                  .b (b),
                  .c_in (c_in),
                  .c_out (c_out),
                  .sum (sum));

   initial begin
      a <= 0;
      b <= 0;
```

```
    c_in <= 0;

    for (i = 0; i < 5; i = i+1) begin
      #10 a <= $random;
          b <= $random;
                c_in <= $random;

    $monitor ("a=%0d, b=%0d, c_in=d%0d, sum=%0d, c_out=%0d", a, b, c_in, c_out, sum);

    end
  end
endmodule
```

Output Snippet #6:-



---------##---------

Snippet #7:- (JK FLIP-FLOP)

RTL: -

```
module jkff(input j,k,clk,rset, output reg q,qb,reg[5:0]out);
reg temp;
parameter  HOLD = 2'b00,
          TOGGLE = 2'b11,
          RESET = 2'b01,
               SET =2'b10;
always@(posedge clk or rset)
begin

if(rset)
begin
q<=1'b0;
qb<=1'b1;
end

else
begin
  case({j,k})
    HOLD:begin temp=temp; out={j,k};  end
    RESET:begin temp=1'b0; out={j,k}; end
    SET:begin temp=1'b1; out={j,k}; end
    TOGGLE:begin temp=~temp; out={j,k}; end
  endcase
  q=temp;
  qb=~temp;
end
   end
endmodule
```

TESTBENCH: -

```
module jkff_tb();


  reg j,k,clk,rset;
  wire[5:0] out;
  wire q,qb;
  integer i,z;
  parameter cycle=10;
  parameter  HOLD = 2'b00,
          TOGGLE = 2'b11,
          RESET = 2'b01,
               SET =2'b10;
  reg[7*8:0] strg="UNKNOWN";


jkff DUT(.j(j),.k(k),.clk(clk),.rset(rset),.q(q),.qb(qb),.out(out));


  always@(j,k)
  begin
  case({j,k})
   HOLD : strg="HOLD";
   TOGGLE : strg="TOGGLE";
   RESET : strg="RESET";
   SET : strg="SET";
   default:strg="UNKNOWN1";
  endcase
 end


  always
   begin
   #(cycle/2);
   clk=1'b0;
   #(cycle/2);
   clk=~clk;
   end
```

```
  task init(input u,v);
   begin
   @(negedge clk);
    j=u;
    k=v;
   end
  endtask

  task rst();
   begin
   @(negedge clk);
   rset=1'b0;
   @(negedge clk);
   rset=1'b1;
   end
  endtask

  initial
  begin
   rst;
     init(0,0);
     init(0,1);
     init(1,0);
     init(1,1);

     #20;
  end



initial
  $monitor("Values of J = %b, K = %b, Q = %b, ~Q = %b, Output = %0s",j,k,q,qb,strg);

initial
  begin
    $dumpfile("file.vcd");
```

```
  $dumpvars();
  #250 $finish;
 end

endmodule
```

Output Snippet #7:-

```
ISim> run
Simulator is doing circuit initialization process.
Finished circuit initialization process.
Values of J = x, K = x, Q = x, ~Q = x, Output = UNKNOWN
Values of J = x, K = x, Q = 0, ~Q = 1, Output = UNKNOWN
Values of J = 0, K = 0, Q = 0, ~Q = 1, Output = HOLD
Values of J = 0, K = 1, Q = 0, ~Q = 1, Output = RESET
Values of J = 1, K = 0, Q = 0, ~Q = 1, Output = SET
Values of J = 1, K = 1, Q = 0, ~Q = 1, Output = TOGGLE
```

Snippet #8:- (DUAL PORT SYNCHRONOUS RAM)

RTL: -

```
module dual_ram(we_clk,re_clk,rst,din,we,re,we_addr,re_addr,dout);

input rst,we,re,we_clk,re_clk,we,re;
input [7:0] din;
input [3:0]we_addr,re_addr;
output reg[3:0]dout;

integer i,j;
parameter depth=16;
parameter width=8;

reg[(width-1):0]mem[(depth-1):0];
```

```
always@(posedge we_clk)
begin

if(rst)
begin
for(i=0;i<depth;i=i+1)
begin
mem[i]=0;
dout=0;
end
end
else
begin
mem[we_addr]=din;
end

end

always@(posedge re_clk)
begin

if(rst)
begin
for(j=0;j<depth;j=j+1)
begin
mem[j]=0;
dout=0;
end
end
else
begin
dout=mem[re_addr];
end
end
endmodule
```

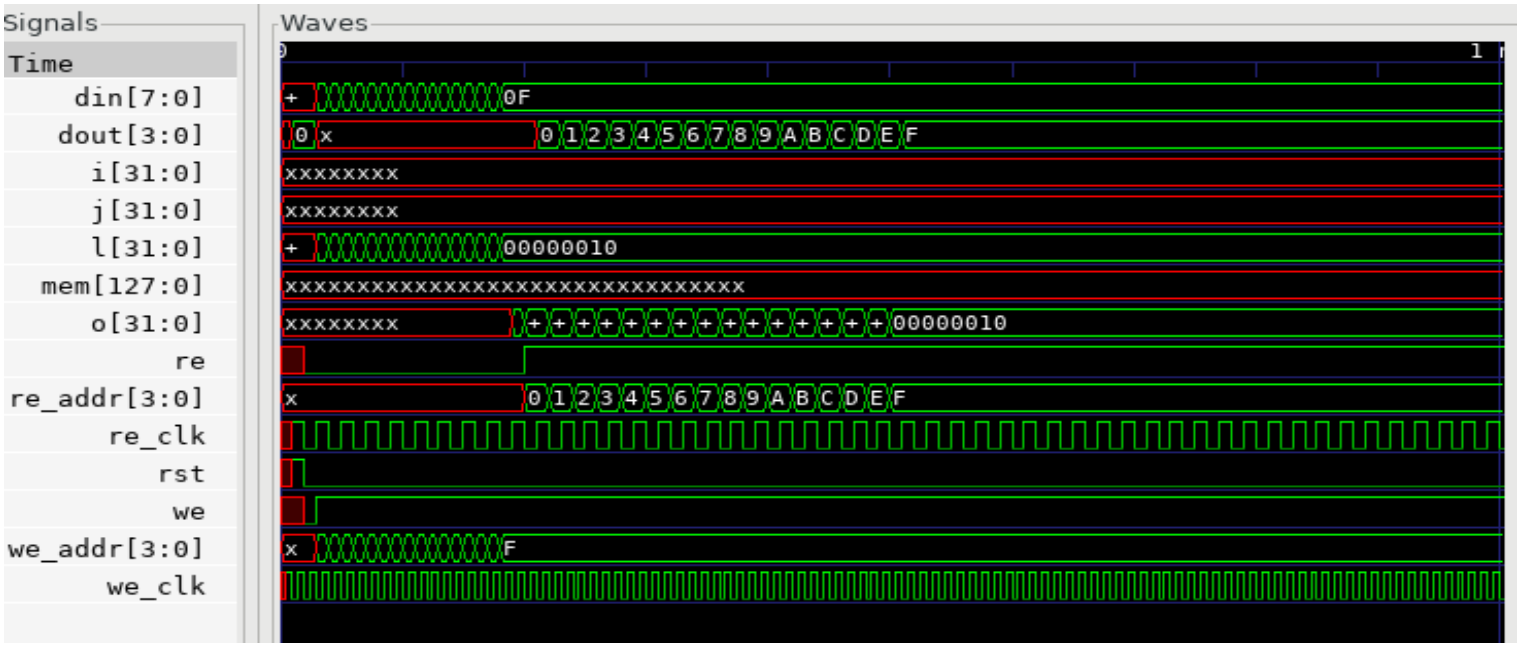
TESTBENCH: -

```
module dual_ram_tb();

reg rst,we,re,we_clk,re_clk,we,re;
reg [7:0] din;
reg [3:0]we_addr,re_addr;
wire [3:0]dout;

integer i,j,l,o;
parameter depth=16;
parameter width=8;
parameter c1=10;
reg[(width-1):0]mem[(depth-1):0];

dual_ram DUT(we_clk,re_clk,rst,din,we,re,we_addr,re_addr,dout);

always
begin
#(c1/2);
we_clk=1'd1;
#(c1/2);
we_clk=1'd0;
end
always
begin
#(c1);
re_clk=1'd1;
#(c1);
re_clk=1'd0;
end

task initialise;
begin
we=0;re=0;
```

```
end
endtask

task write(input [3:0]i,input[7:0]j);
begin
@(negedge we_clk)
we=1'b1;
we_addr=i;
din=j;
end
endtask

task read(input [3:0]y);
begin
@(negedge re_clk)
re=1'b1;
re_addr=y;
end
endtask

task reset;
begin
#c1
rst=1'd1;
#c1
rst=1'd0;
end
endtask



task delay;
begin
#10;
end
endtask
```

```
initial
begin
reset();
initialise();
delay();
for(l=0;l<16;l=l+1)
write(l,l);
delay();
for(o=0;o<16;o=o+1)
read(o);
delay();
end
initial
begin
$dumpfile("file.vcd");
$dumpvars();
#2000 $finish;
end
endmodule
```

Output Snippet #8:-



Snippet #9:- (COUNTER: SEQUENCE-> 1,2,1,4,1,8,1,16,1,32…)

RTL: -

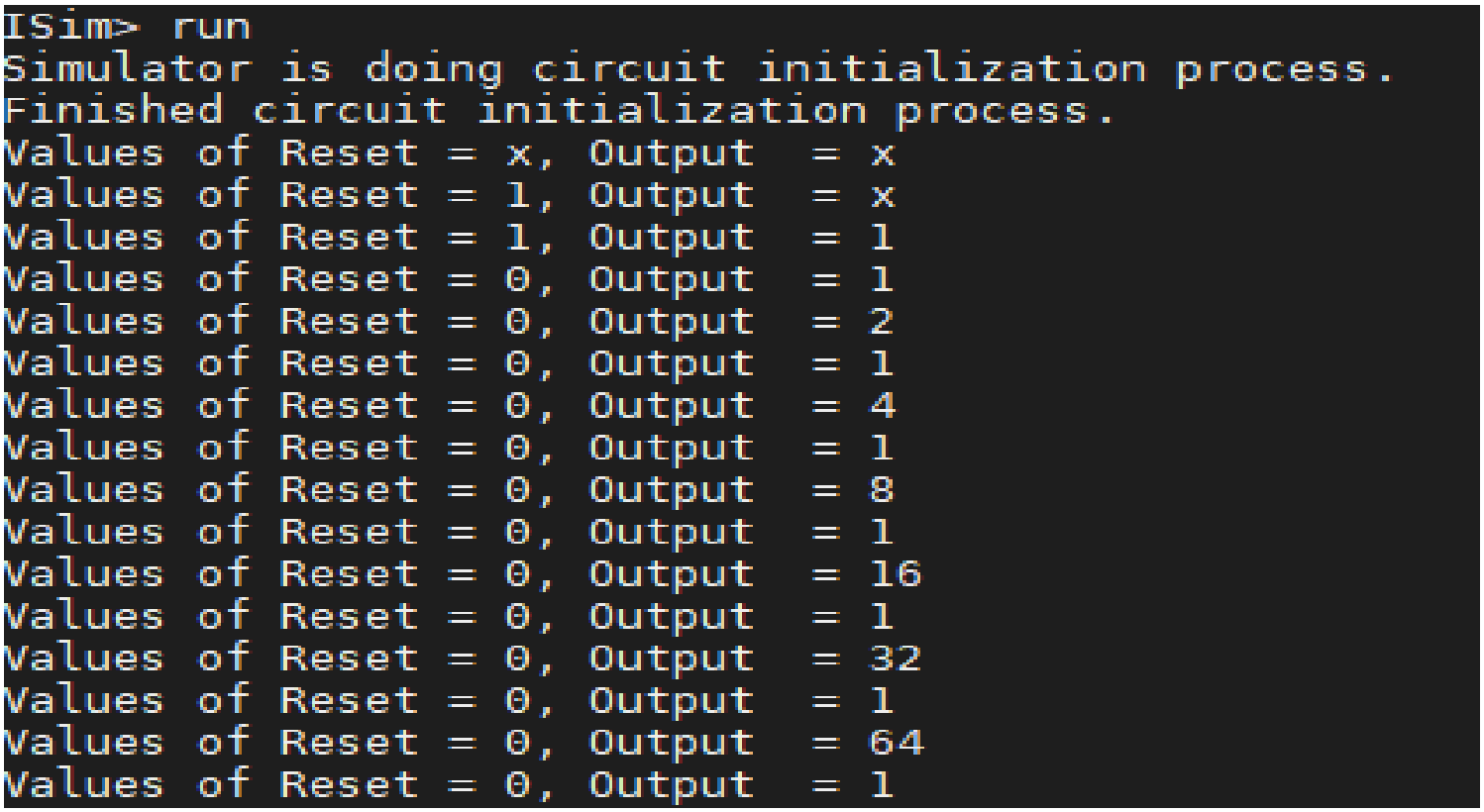
```
module counter1(clock,reset,dout);

input clock,reset;
output reg[7:0]dout;

reg [7:0]temp;

always@(posedge clock)
begin
if(reset)
begin
dout=1;
temp=1;
end

else
begin
if(dout==1)
begin
if(temp==0)
temp=1;
else
begin
dout=temp;
temp=temp*2;
end
end
else
dout=1;
end
end
endmodule
```

TESTBENCH: -

```
reg reset,clock;
wire [7:0]dout;

parameter cycle=10;

counter1 DUT(clock,reset,dout);

always
  begin
    #(cycle/2);
    clock<=1'b0;
    #(cycle/2);
    clock<=~clock;
  end

task rst();
  begin
    @(negedge clock)
    reset<=1'b1;
    @(negedge clock)
    reset<=1'b0;
  end
endtask


initial
  begin
    rst();
        #140;
            $finish;
  end

initial
```

```
$monitor("Values of Reset = %b, Output  = %0d ",reset,dout);


/*initial
begin
$dumpfile("counter1.vcd");
$dumpvars();
#5000 $finish;
end*/


endmodule
```

Output Snippet #9:-



Snippet #10:- (SHIFT REGISTER)

RTL: -

```
module shift_reg(clock,reset,mode,din,dout);


input clock,reset,mode;

input [3:0]din;

output [3:0]dout;


reg [3:0]temp_out;


always@(posedge clock)

begin

  if(reset)

    begin

          temp_out<=0;

    end


  else

    begin

          case(mode)

          1'b0:begin

               temp_out[3]<=din[0];

               temp_out[2]<=temp_out[3];

               temp_out[1]<=temp_out[2];

               temp_out[0]<=temp_out[1];

              end

          1'b1:begin

               temp_out[3]<=din[3];

               temp_out[2]<=din[2];

               temp_out[1]<=din[1];

               temp_out[0]<=din[0];

              end

          endcase

    end
```

```
end

assign dout=temp_out;

endmodule
```

TESTBENCH: -

```
module shift_reg_tb();
reg clock,reset,mode;
reg [3:0]din;
wire [3:0]dout;
shift_reg DUT(clock,reset,mode,din,dout);

always
  begin
  #5 clock=1'b1;
  #5 clock=~clock;
  end

task rst();
  begin
    @(negedge clock)
          reset=1;
    @(negedge clock)
          reset=0;
  end
endtask

task initialize_dut(input mode_dut,[3:0]din_dut);
  begin
    @(negedge clock)
          mode=mode_dut;
          din=din_dut;
  end
endtask
```

```
initial
  begin
    rst();
    initialize_dut(0,0);
    initialize_dut(0,1);
    initialize_dut(0,0);
    initialize_dut(0,1);      //Output-1010
    initialize_dut(1,4'b1011); //Output-1011
    #300;
  end
initial
$monitor("Reset=%b ,Input=%b ,Output=%b",reset,din,dout);
initial
  begin
    $dumpfile("shift_reg.vcd");
    $dumpvars();
    #200 $finish;
  end
endmodule
```

Output Snippet #10:-