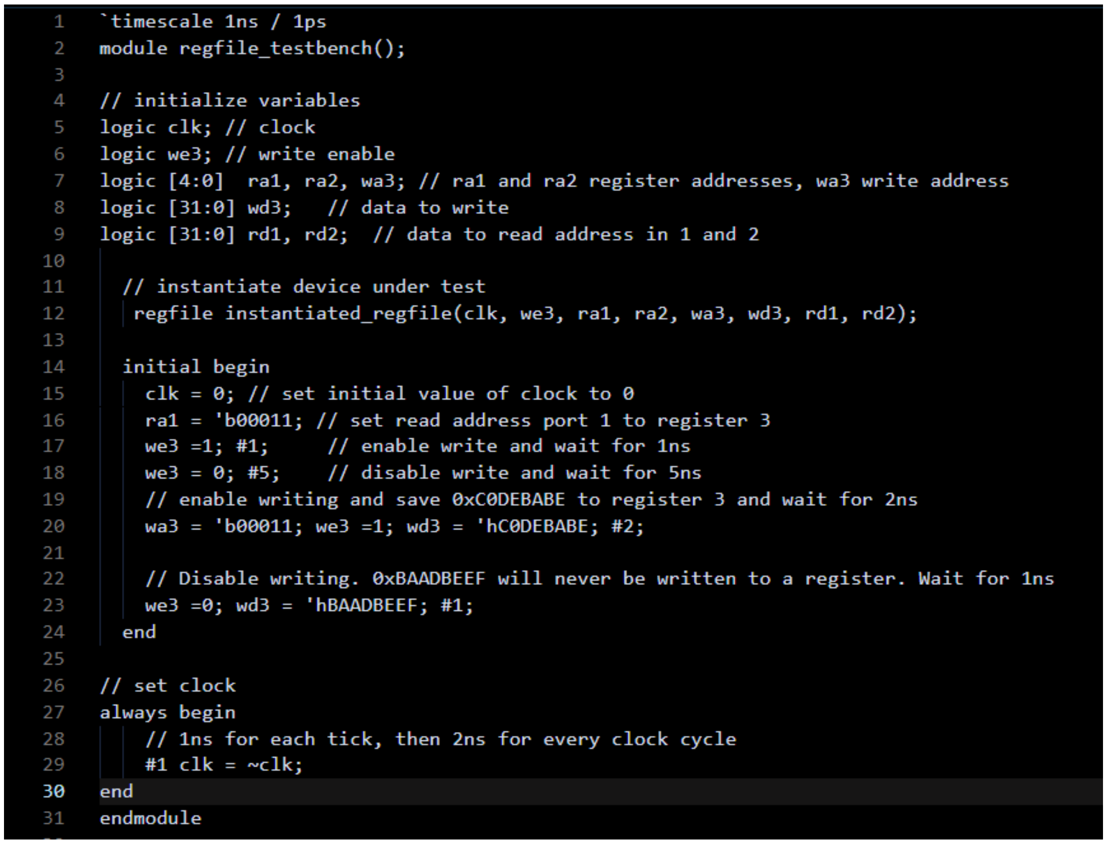
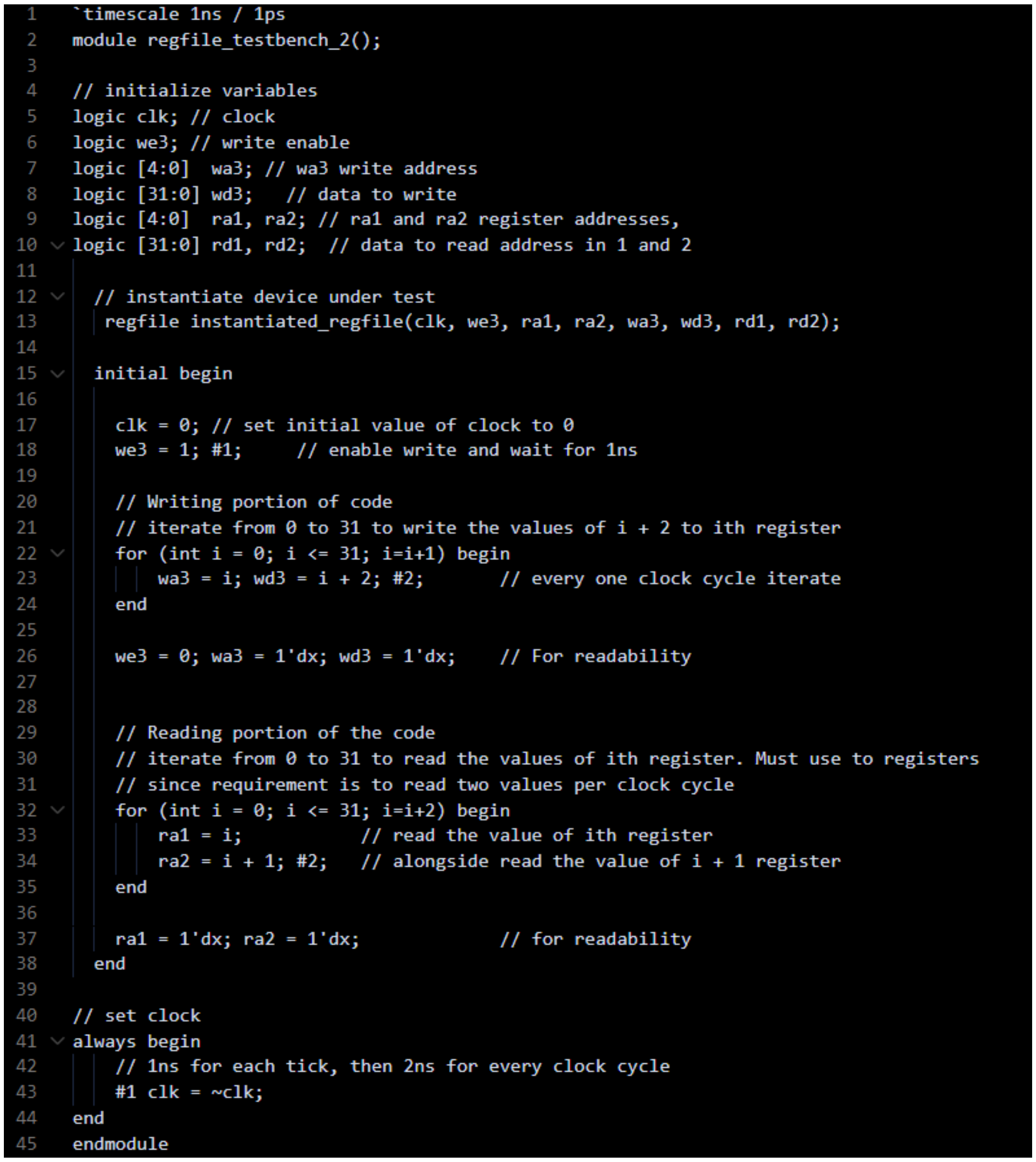
REGISTER FILE

1.



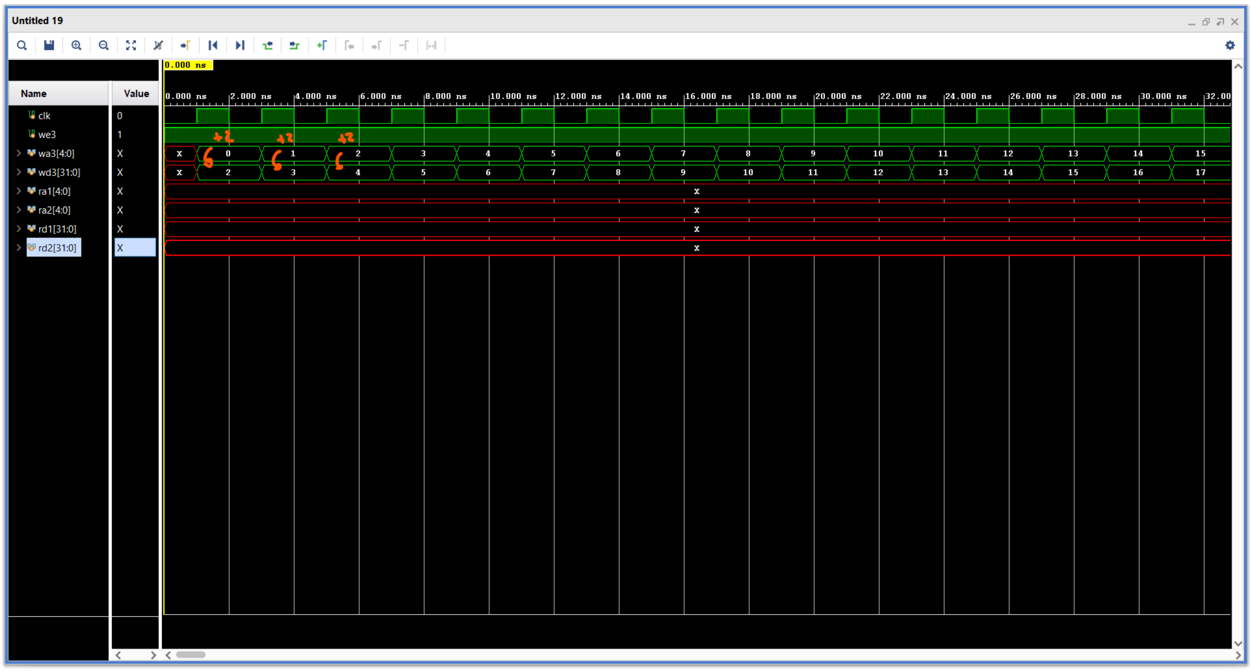
```
`timescale 1ns / 1ps
module regfile_testbench();

// initialize variables
logic clk; // clock
logic we3; // write enable
logic [4:0]  ra1, ra2, wa3; // ra1 and ra2 register addresses, wa3 write address
logic [31:0] wd3;   // data to write
logic [31:0] rd1, rd2;  // data to read address in 1 and 2

  // instantiate device under test
   regfile instantiated_regfile(clk, we3, ra1, ra2, wa3, wd3, rd1, rd2);

  initial begin
    clk = 0; // set initial value of clock to 0
    ra1 = 'b00011; // set read address port 1 to register 3
    we3 =1; #1;     // enable write and wait for 1ns
    we3 = 0; #5;    // disable write and wait for 5ns
    // enable writing and save 0xC0DEBABE to register 3 and wait for 2ns
    wa3 = 'b00011; we3 =1; wd3 = 'hC0DEBABE; #2;

    // Disable writing. 0xBAADBEEF will never be written to a register. Wait for 1ns
    we3 =0; wd3 = 'hBAADBEEF; #1;
  end

// set clock
always begin
    // 1ns for each tick, then 2ns for every clock cycle
    #1 clk = ~clk;
end
endmodule
```

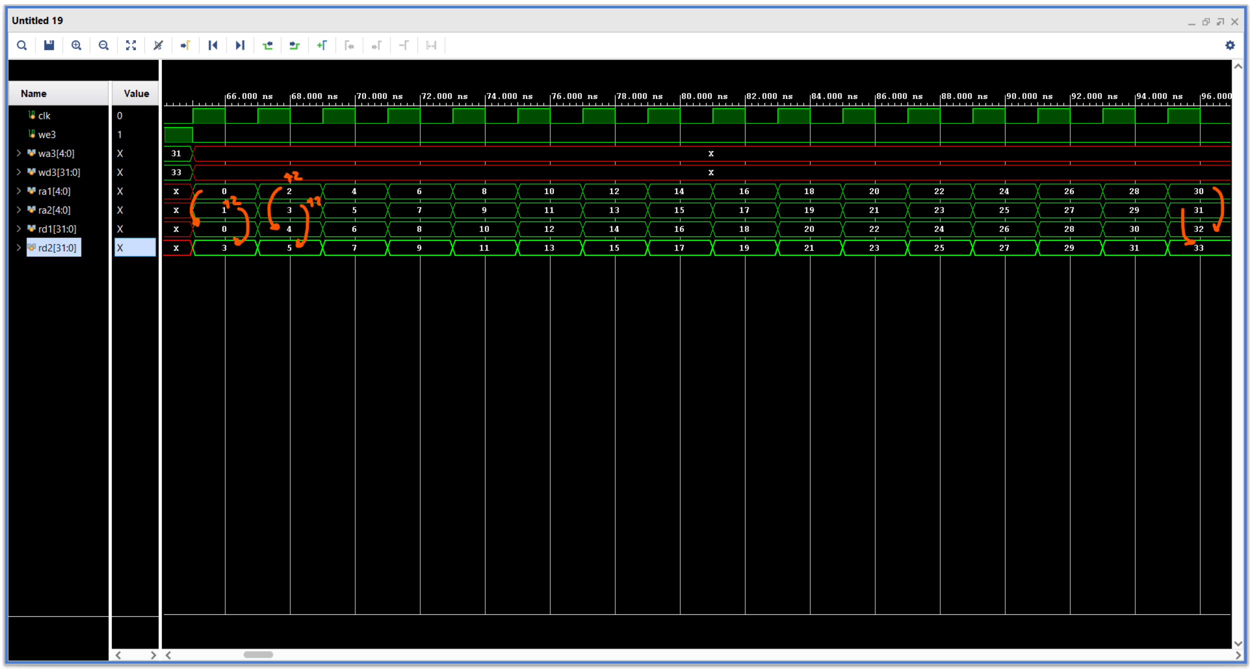
2.



```
`timescale 1ns / 1ps
module regfile_testbench_2();

// initialize variables
logic clk; // clock
logic we3; // write enable
logic [4:0]  wa3; // wa3 write address
logic [31:0] wd3;   // data to write
logic [4:0]  ra1, ra2; // ra1 and ra2 register addresses,
logic [31:0] rd1, rd2;  // data to read address in 1 and 2

  // instantiate device under test
   regfile instantiated_regfile(clk, we3, ra1, ra2, wa3, wd3, rd1, rd2);

  initial begin

    clk = 0; // set initial value of clock to 0
    we3 = 1; #1;     // enable write and wait for 1ns

    // Writing portion of code
    // iterate from 0 to 31 to write the values of i + 2 to ith register
    for (int i = 0; i <= 31; i=i+1) begin
        wa3 = i; wd3 = i + 2; #2;       // every one clock cycle iterate
    end

    we3 = 0; wa3 = 1'dx; wd3 = 1'dx;    // For readability


    // Reading portion of the code
    // iterate from 0 to 31 to read the values of ith register. Must use to registers
    // since requirement is to read two values per clock cycle
    for (int i = 0; i <= 31; i=i+2) begin
        ra1 = i;           // read the value of ith register
        ra2 = i + 1; #2;   // alongside read the value of i + 1 register
    end

    ra1 = 1'dx; ra2 = 1'dx;             // for readability
  end

// set clock
always begin
    // 1ns for each tick, then 2ns for every clock cycle
    #1 clk = ~clk;
end
endmodule
```

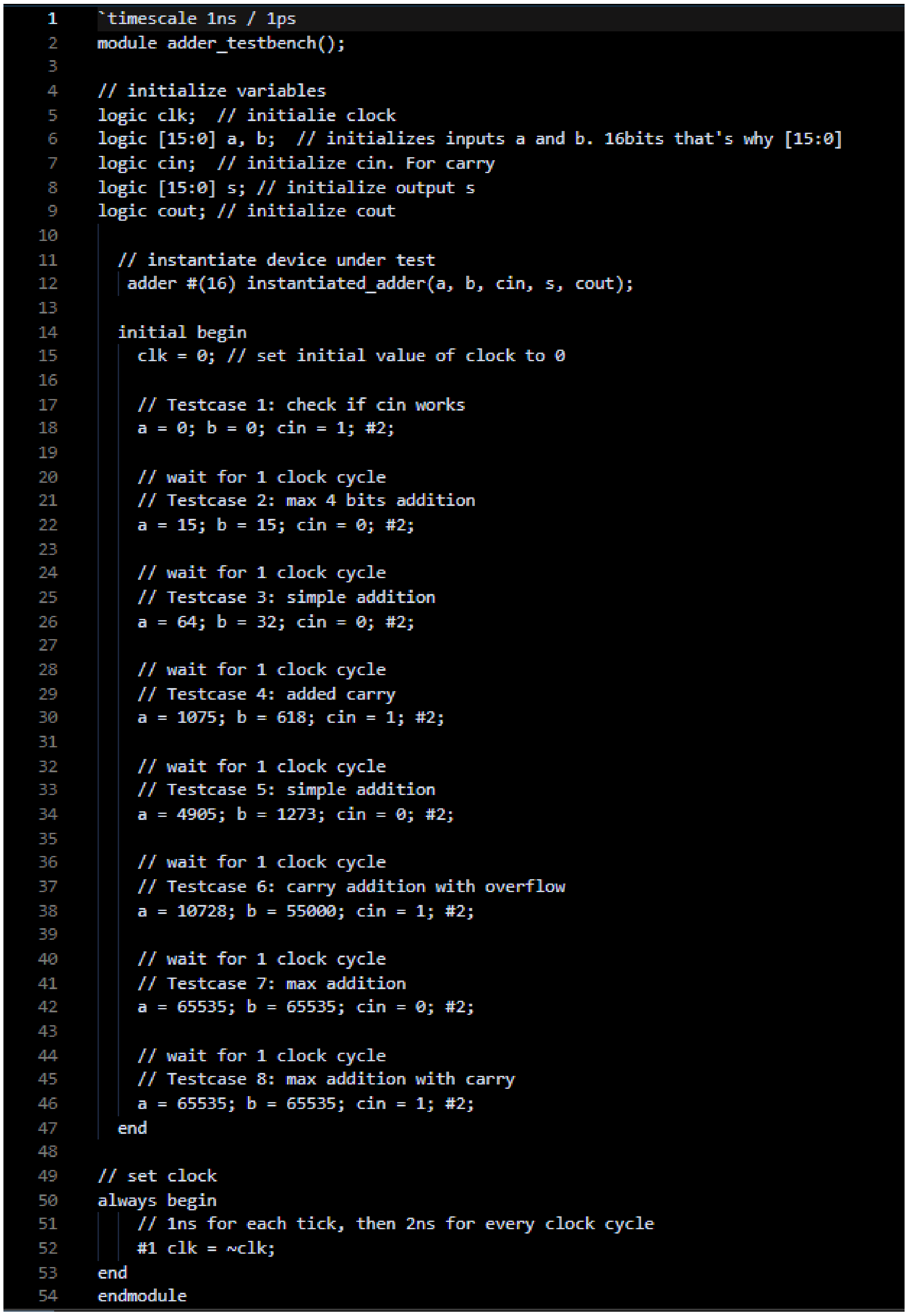
3.


Untitled 19
Name
Value
clk
we3
wa3[4:0]
wd3[31:0]
ra1[4:0]
ra2[4:0]
rd1[31:0]
rd2[31:0]
+2
+2
+2


Untitled 19
Name
Value
clk
we3
wa3[4:0]
wd3[31:0]
ra1[4:0]
ra2[4:0]
rd1[31:0]
rd2[31:0]
+2
+2
+2

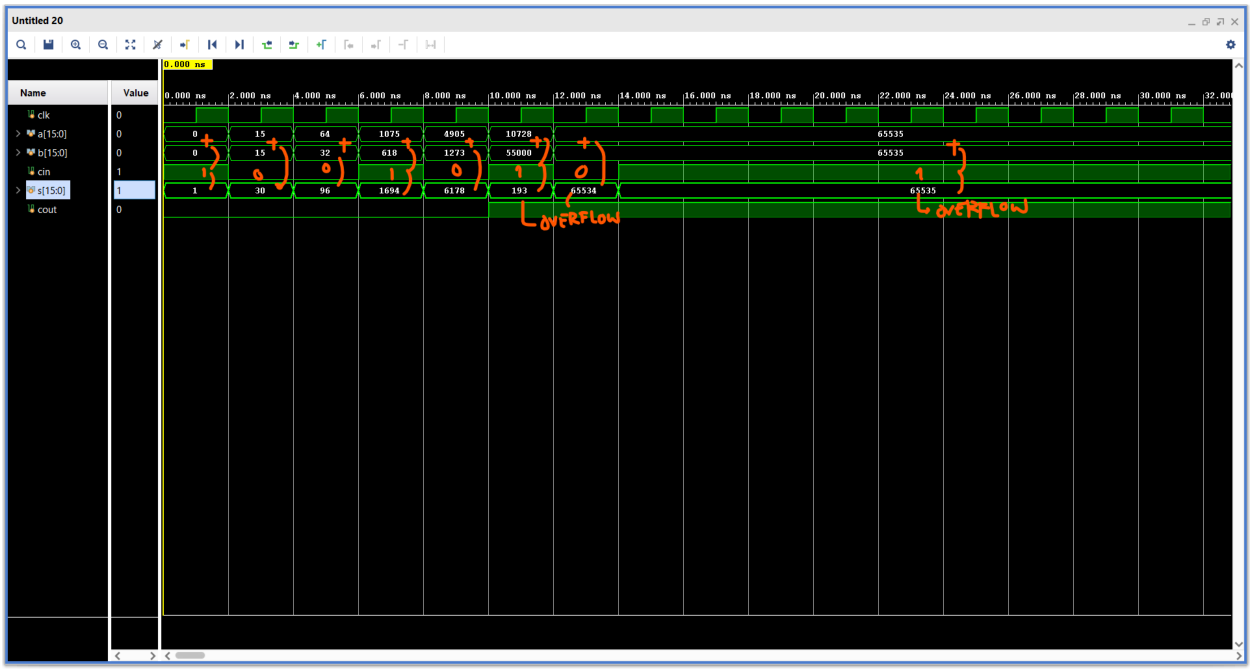
ADDER

1.



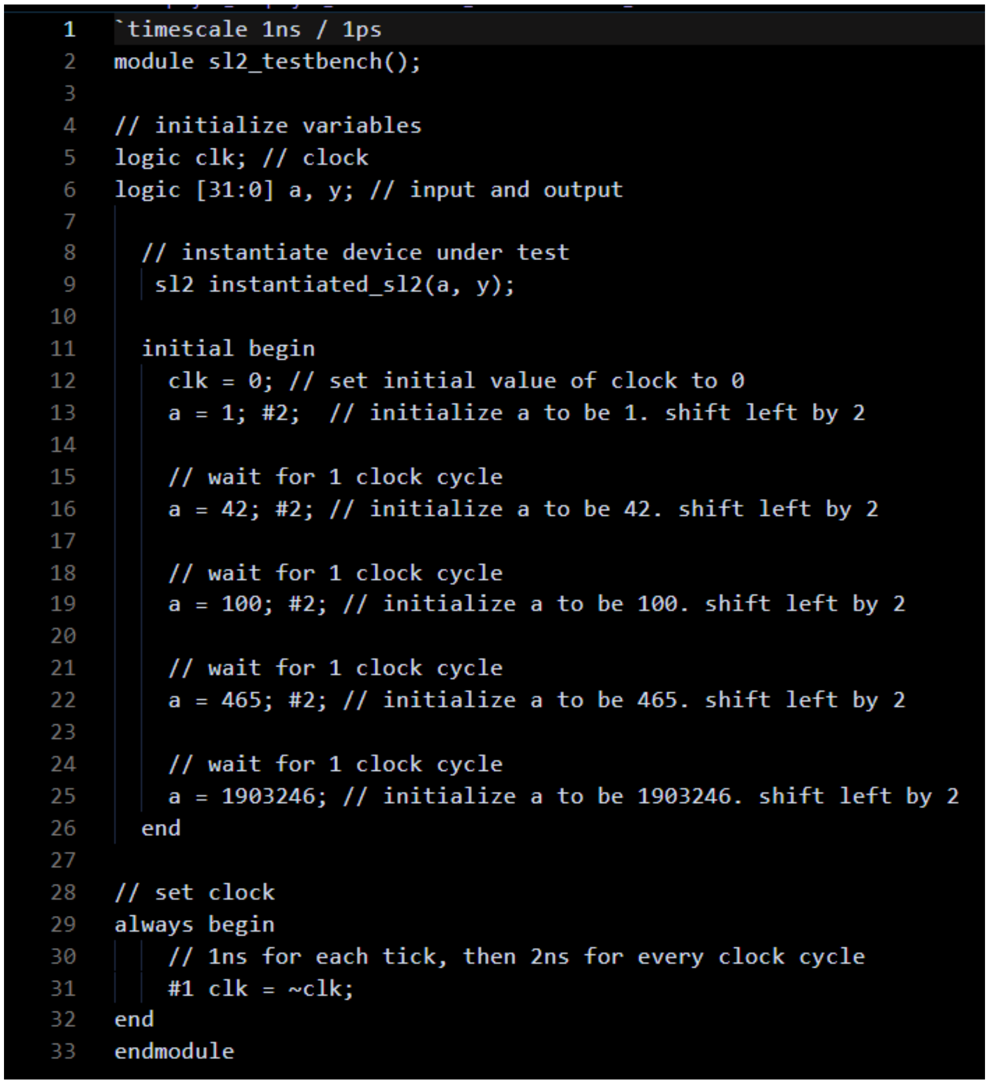
```
`timescale 1ns / 1ps
module adder_testbench();

// initialize variables
logic clk;  // initialie clock
logic [15:0] a, b;  // initializes inputs a and b. 16bits that's why [15:0]
logic cin;  // initialize cin. For carry
logic [15:0] s; // initialize output s
logic cout; // initialize cout

  // instantiate device under test
   adder #(16) instantiated_adder(a, b, cin, s, cout);

  initial begin
    clk = 0; // set initial value of clock to 0

    // Testcase 1: check if cin works
    a = 0; b = 0; cin = 1; #2;

    // wait for 1 clock cycle
    // Testcase 2: max 4 bits addition
    a = 15; b = 15; cin = 0; #2;

    // wait for 1 clock cycle
    // Testcase 3: simple addition
    a = 64; b = 32; cin = 0; #2;

    // wait for 1 clock cycle
    // Testcase 4: added carry
    a = 1075; b = 618; cin = 1; #2;

    // wait for 1 clock cycle
    // Testcase 5: simple addition
    a = 4905; b = 1273; cin = 0; #2;

    // wait for 1 clock cycle
    // Testcase 6: carry addition with overflow
    a = 10728; b = 55000; cin = 1; #2;

    // wait for 1 clock cycle
    // Testcase 7: max addition
    a = 65535; b = 65535; cin = 0; #2;

    // wait for 1 clock cycle
    // Testcase 8: max addition with carry
    a = 65535; b = 65535; cin = 1; #2;
  end

// set clock
always begin
    // 1ns for each tick, then 2ns for every clock cycle
    #1 clk = ~clk;
end
endmodule
```

2.


Untitled 20
0.000 ns
Name
Value
clk 0
a[15:0] 0
b[15:0] 0
cin 1
s[15:0] 1
cout 0
0.000 ns
2.000 ns
4.000 ns
6.000 ns
8.000 ns
10.000 ns
12.000 ns
14.000 ns
16.000 ns
18.000 ns
20.000 ns
22.000 ns
24.000 ns
26.000 ns
28.000 ns
30.000 ns
0 15 64 1075 4905 10728 65535
0 15 32 618 1273 55000 65535
1 30 96 1694 6178 193 65534 65535
OVERFLOW
OVERFLOW

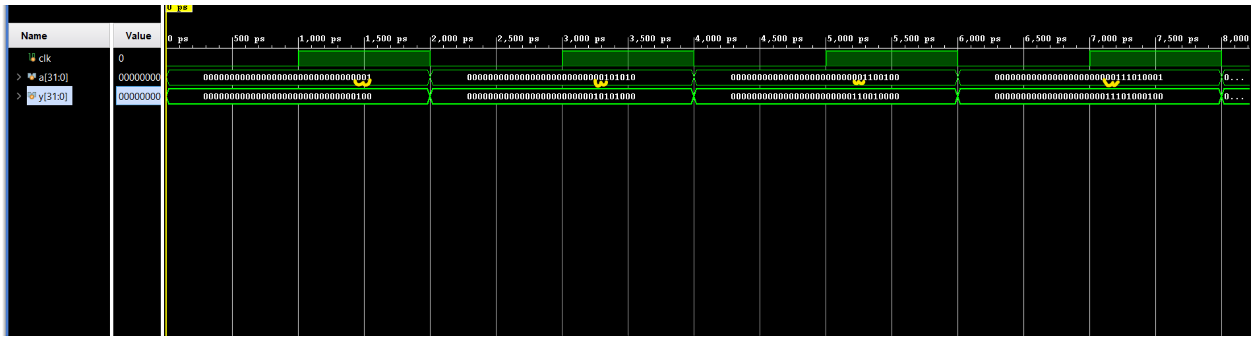
SHIFT-LEFT-BY-2

1.

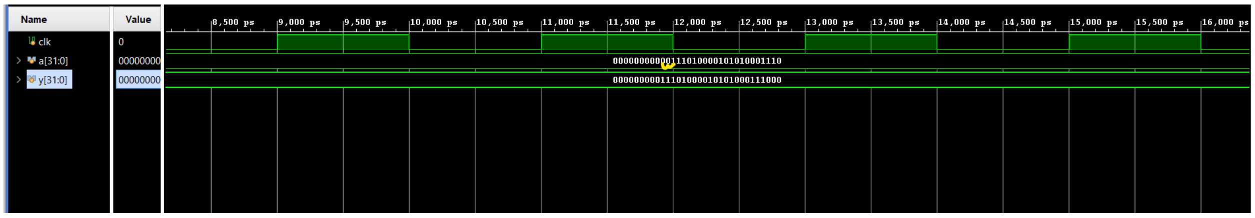


```
`timescale 1ns / 1ps
module sl2_testbench();

// initialize variables
logic clk; // clock
logic [31:0] a, y; // input and output

  // instantiate device under test
   sl2 instantiated_sl2(a, y);

  initial begin
    clk = 0; // set initial value of clock to 0
    a = 1; #2;  // initialize a to be 1. shift left by 2

    // wait for 1 clock cycle
    a = 42; #2; // initialize a to be 42. shift left by 2

    // wait for 1 clock cycle
    a = 100; #2; // initialize a to be 100. shift left by 2

    // wait for 1 clock cycle
    a = 465; #2; // initialize a to be 465. shift left by 2

    // wait for 1 clock cycle
    a = 1903246; // initialize a to be 1903246. shift left by 2
  end

// set clock
always begin
    // 1ns for each tick, then 2ns for every clock cycle
    #1 clk = ~clk;
end
endmodule
```

2.



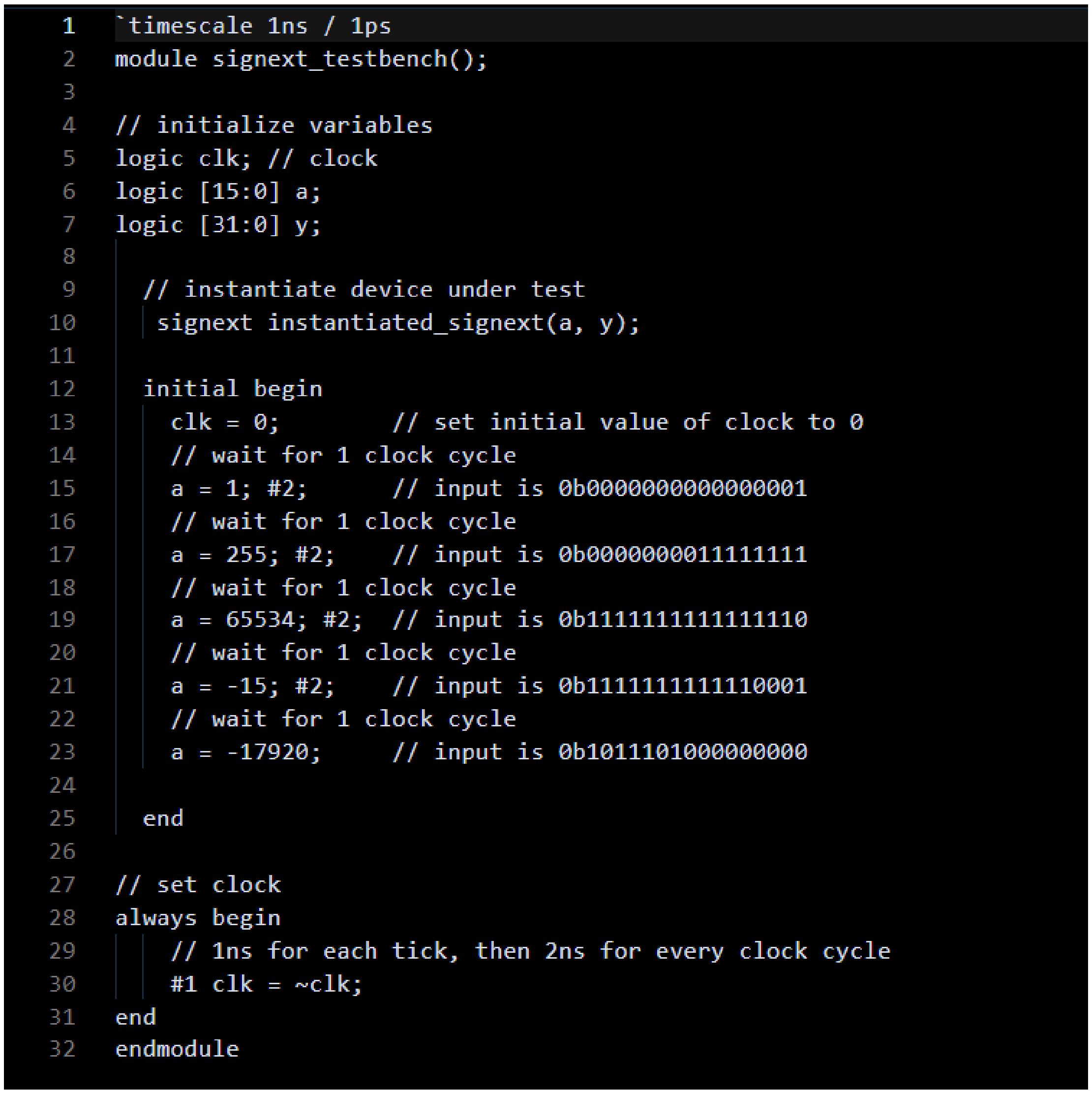
```
Name      Value
clk       0
a[31:0]   00000000
y[31:0]   00000000

0 ps  500 ps  1,000 ps  1,500 ps  2,000 ps  2,500 ps  3,000 ps  3,500 ps  4,000 ps  4,500 ps  5,000 ps  5,500 ps  6,000 ps  6,500 ps  7,000 ps  7,500 ps  8,000

a: 00000000000000000000000000000001  00000000000000000000000000101010  00000000000000000000000001100100  00000000000000000000000011101000 1  0...
y: 00000000000000000000000000000100  00000000000000000000000010101000  00000000000000000000000110010000  00000000000000000000001110100010 0  0...
```



```
Name      Value
clk       0
a[31:0]   00000000
y[31:0]   00000000

8,500 ps  9,000 ps  9,500 ps  10,000 ps  10,500 ps  11,000 ps  11,500 ps  12,000 ps  12,500 ps  13,000 ps  13,500 ps  14,000 ps  14,500 ps  15,000 ps  15,500 ps  16,000 ps

a: 00000000000111010000101010001110
y: 00000000011101000010101000111000
```

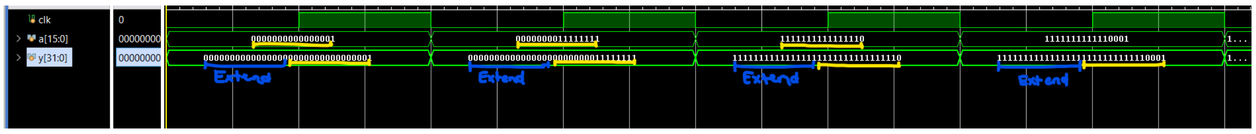
SIGN EXTENDER

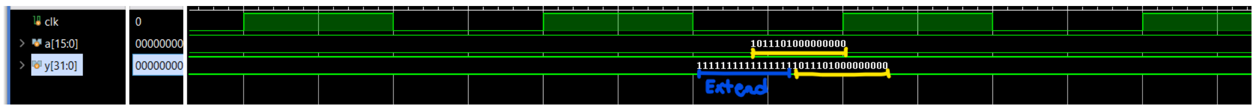
1.



```
`timescale 1ns / 1ps
module signext_testbench();

// initialize variables
logic clk; // clock
logic [15:0] a;
logic [31:0] y;

  // instantiate device under test
    signext instantiated_signext(a, y);

  initial begin
    clk = 0;         // set initial value of clock to 0
    // wait for 1 clock cycle
    a = 1; #2;       // input is 0b0000000000000001
    // wait for 1 clock cycle
    a = 255; #2;     // input is 0b0000000011111111
    // wait for 1 clock cycle
    a = 65534; #2;  // input is 0b1111111111111110
    // wait for 1 clock cycle
    a = -15; #2;     // input is 0b1111111111110001
    // wait for 1 clock cycle
    a = -17920;      // input is 0b1011101000000000

  end

// set clock
always begin
    // 1ns for each tick, then 2ns for every clock cycle
    #1 clk = ~clk;
end
endmodule
```

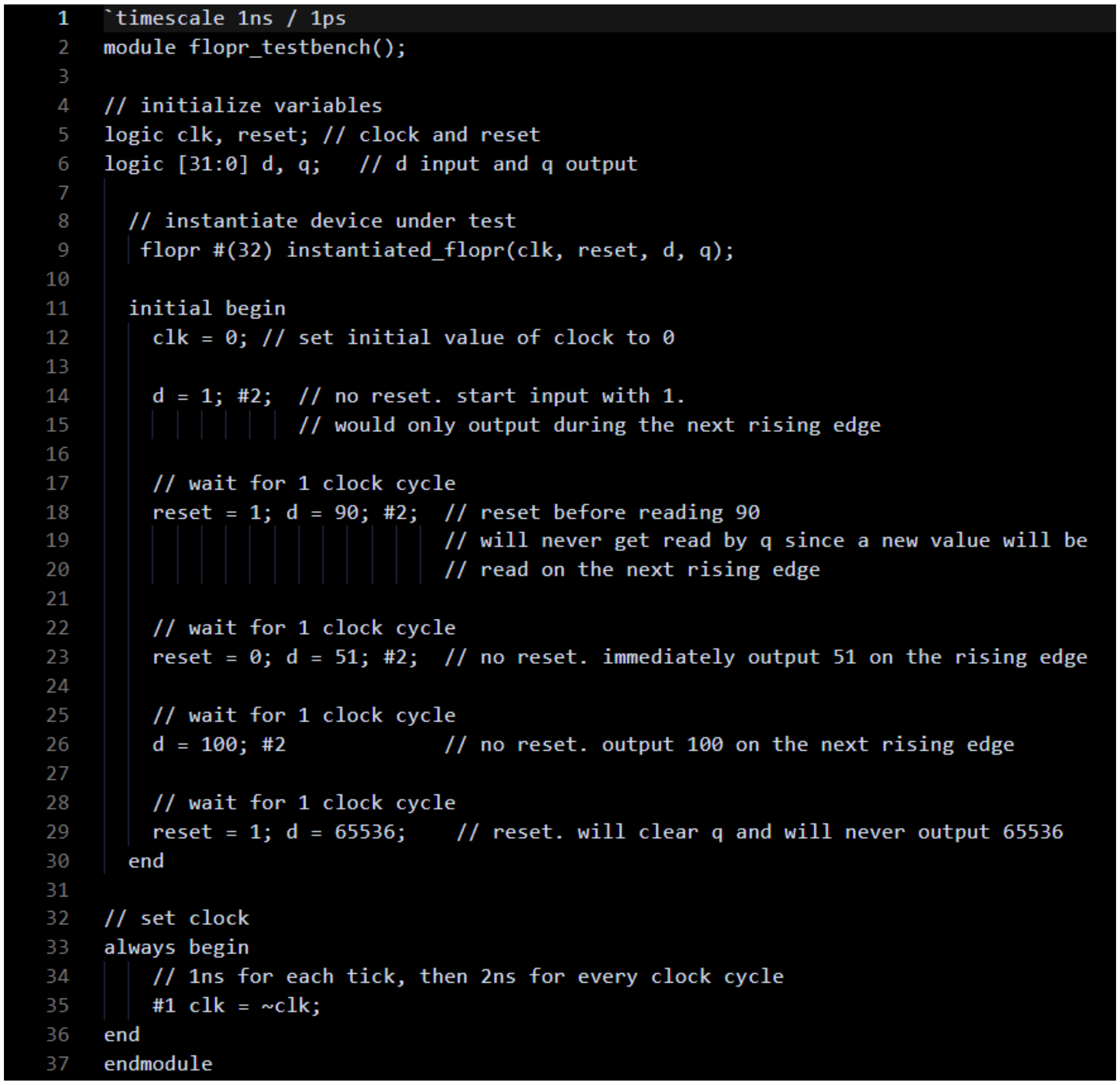
2.





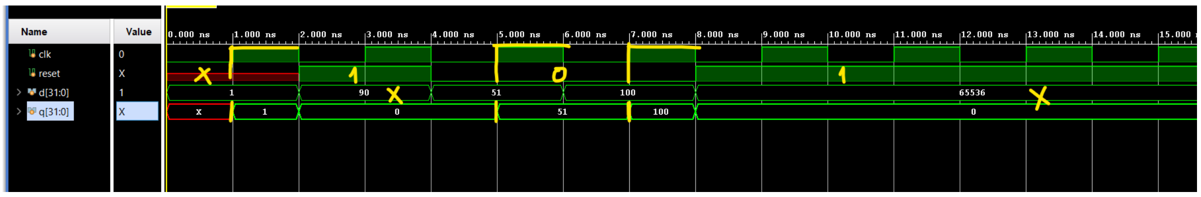
RESETTABLE FLIP FLOP

1.



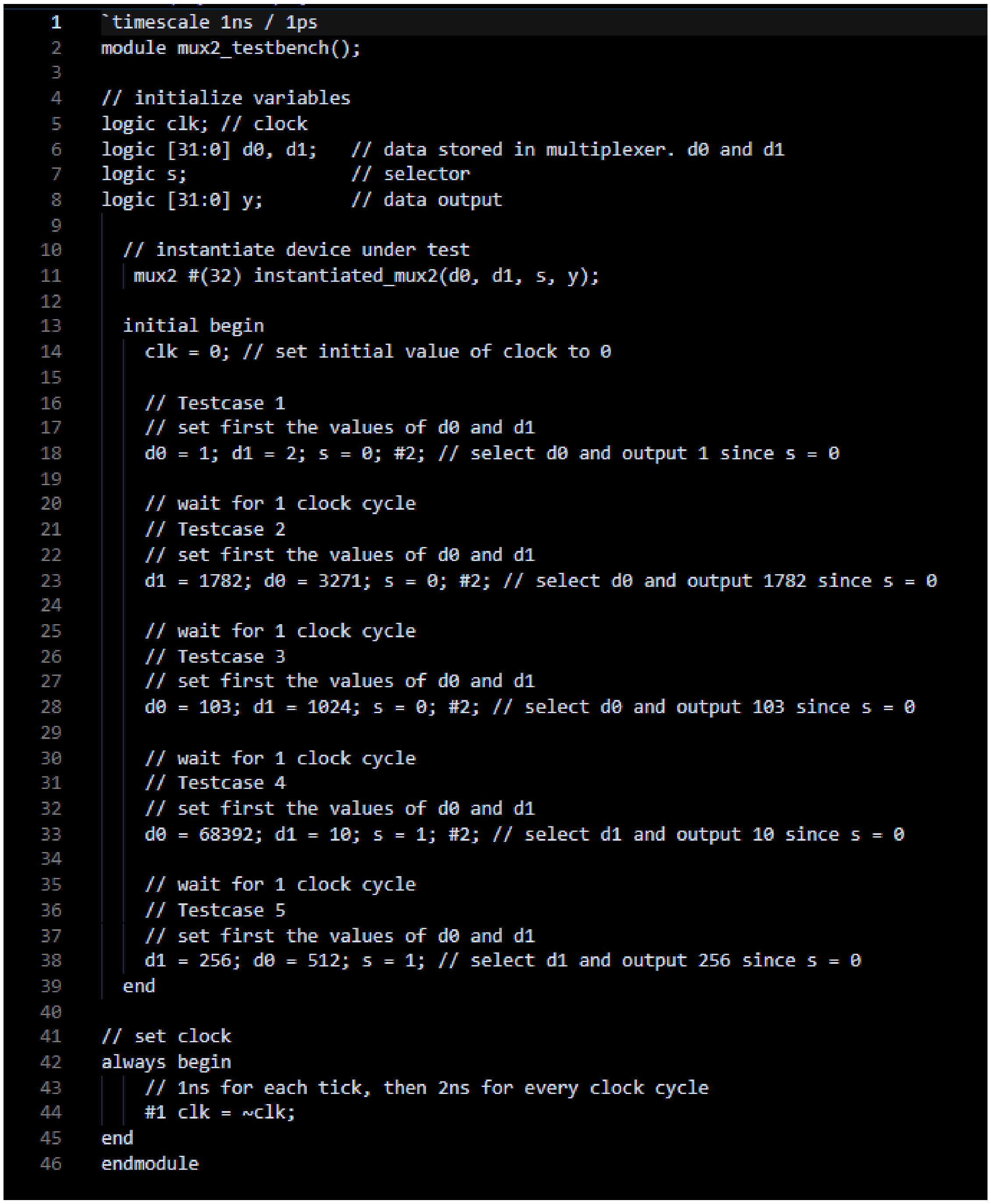
```
`timescale 1ns / 1ps
module flopr_testbench();

// initialize variables
logic clk, reset; // clock and reset
logic [31:0] d, q;   // d input and q output

  // instantiate device under test
   flopr #(32) instantiated_flopr(clk, reset, d, q);

  initial begin
    clk = 0; // set initial value of clock to 0

    d = 1; #2;  // no reset. start input with 1.
                // would only output during the next rising edge

    // wait for 1 clock cycle
    reset = 1; d = 90; #2;  // reset before reading 90
                            // will never get read by q since a new value will be
                            // read on the next rising edge

    // wait for 1 clock cycle
    reset = 0; d = 51; #2;  // no reset. immediately output 51 on the rising edge

    // wait for 1 clock cycle
    d = 100; #2             // no reset. output 100 on the next rising edge

    // wait for 1 clock cycle
    reset = 1; d = 65536;    // reset. will clear q and will never output 65536
  end

// set clock
always begin
    // 1ns for each tick, then 2ns for every clock cycle
    #1 clk = ~clk;
end
endmodule
```

2.



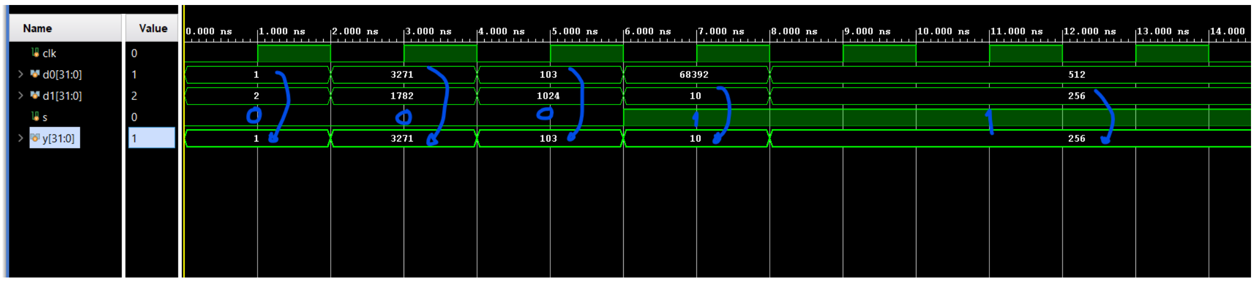
TWO-WAY MULTIPLEXER

1.



```
`timescale 1ns / 1ps
module mux2_testbench();

// initialize variables
logic clk; // clock
logic [31:0] d0, d1;   // data stored in multiplexer. d0 and d1
logic s;               // selector
logic [31:0] y;        // data output

  // instantiate device under test
   mux2 #(32) instantiated_mux2(d0, d1, s, y);

  initial begin
    clk = 0; // set initial value of clock to 0

    // Testcase 1
    // set first the values of d0 and d1
    d0 = 1; d1 = 2; s = 0; #2; // select d0 and output 1 since s = 0

    // wait for 1 clock cycle
    // Testcase 2
    // set first the values of d0 and d1
    d1 = 1782; d0 = 3271; s = 0; #2; // select d0 and output 1782 since s = 0

    // wait for 1 clock cycle
    // Testcase 3
    // set first the values of d0 and d1
    d0 = 103; d1 = 1024; s = 0; #2; // select d0 and output 103 since s = 0

    // wait for 1 clock cycle
    // Testcase 4
    // set first the values of d0 and d1
    d0 = 68392; d1 = 10; s = 1; #2; // select d1 and output 10 since s = 0

    // wait for 1 clock cycle
    // Testcase 5
    // set first the values of d0 and d1
    d1 = 256; d0 = 512; s = 1; // select d1 and output 256 since s = 0
  end

// set clock
always begin
    // 1ns for each tick, then 2ns for every clock cycle
    #1 clk = ~clk;
end
endmodule
```

2.


Name
Value
clk 0
d0[31:0] 1
d1[31:0] 2
s 0
y[31:0] 1
0.000 ns
1.000 ns
2.000 ns
3.000 ns
4.000 ns
5.000 ns
6.000 ns
7.000 ns
8.000 ns
9.000 ns
10.000 ns
11.000 ns
12.000 ns
13.000 ns
14.000
1
3271
103
68392
512
2
1782
1024
10
256
0
0
0
1
1
1
3271
103
10
256