



END SEMESTER ASSESSMENT (ESA)
B.TECH. (CSE)
III SEMESTER

UE18CS206 – DIGITAL DESIGN & COMPUTER
ORGANIZATION LABORATORY

PROJECT REPORT

ON

**“Design & implement shift-left, shift-
right & rotate operations for the 16-
bit ALU
using multiplexers.”**

SUBMITTED BY

NAME

SRN

1) Serena A. Gomez	PES2UG19CS372
2) Shabrinath K.	PES2UG19CS373
3) Shamanth N. Chitturi	PES2UG19CS374
4) Shanoo Raghav	PES2UG19CS375

AUGUST – DECEMBER 2020

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

**ELECTRONIC CITY CAMPUS,
BENGALURU – 560100, KARNATAKA, INDIA**

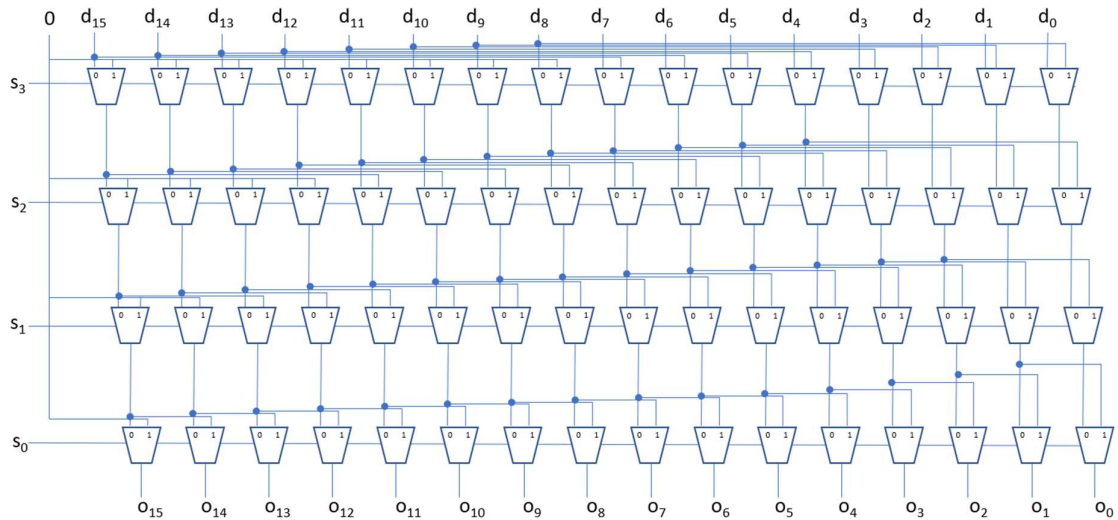
TABLE OF CONTENTS		
Sl.No	TOPIC	PAGE No
1.	ABSTRACT OF THE PROJECT	1
2.	CIRCUIT DIAGRAM	2
3.	MAIN VERILOG CODE	4
4.	TEST BENCH FILE	5
5.	SCREEN SHOTS OF THE OUTPUT	6

ABSTRACT OF THE PROJECT:

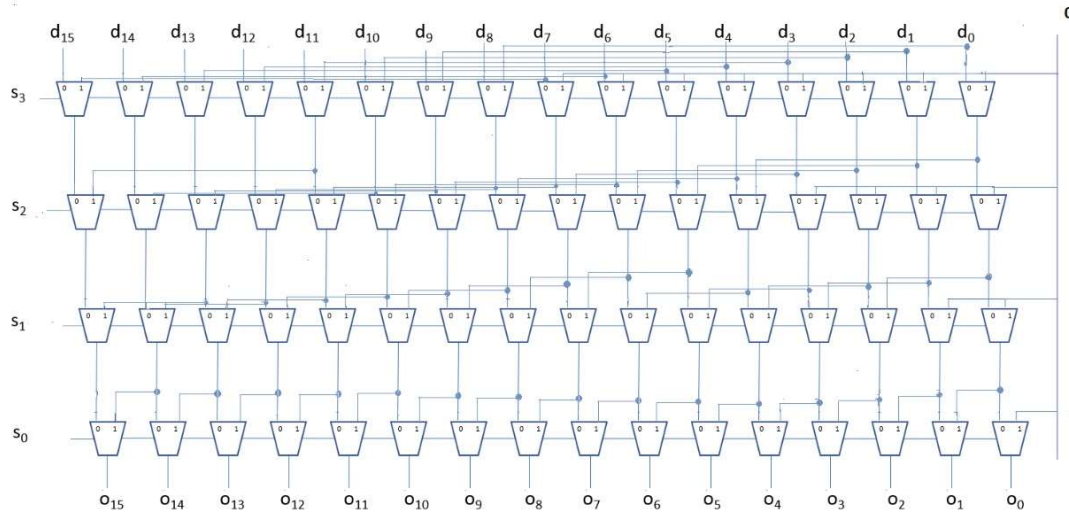
As the name implies, a **shifter** shifts a binary number left or right by a specified number of positions. And a **rotator** rotates number in a circle such that empty spots are filled with bits shifted off the other end. Shifting and rotating data is required in several applications including arithmetic operations, variable-length coding, and bit-indexing. These are commonly found in both digital signal processors and general-purpose processors. Here we design and implement shift right logic, shift left logic, rotate right and rotate left using multiplexers. The 16- bit shifters and rotators, which uses four stages with 8-bit, 4- bit, 2- bit, and 1- bit shifts. These designs are optimized to share hardware for different operations.

CIRCUIT DIAGRAM:

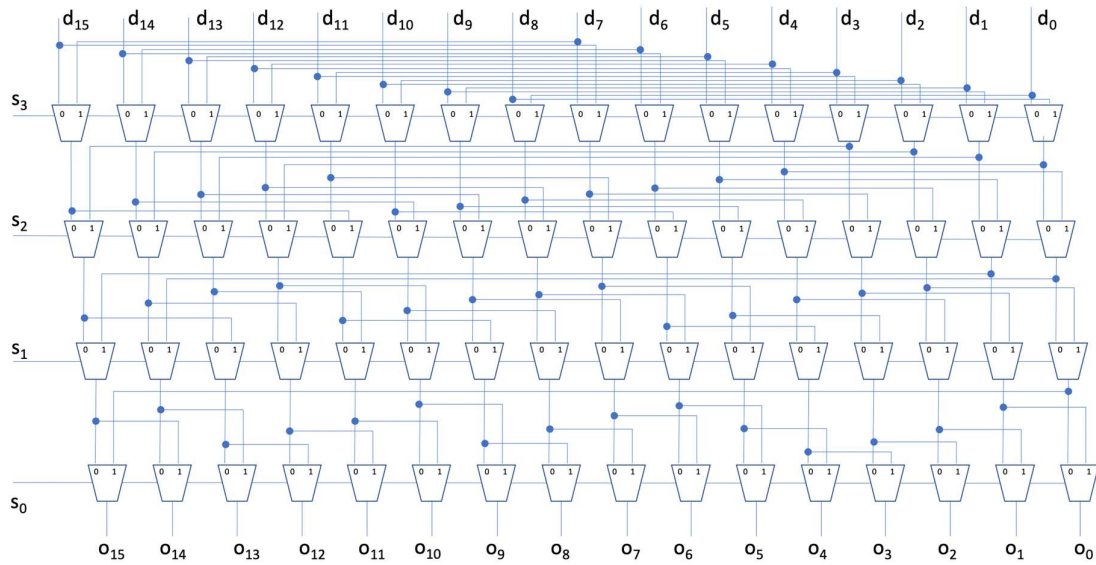
16-bit Shift Right Operation using Multiplexers:



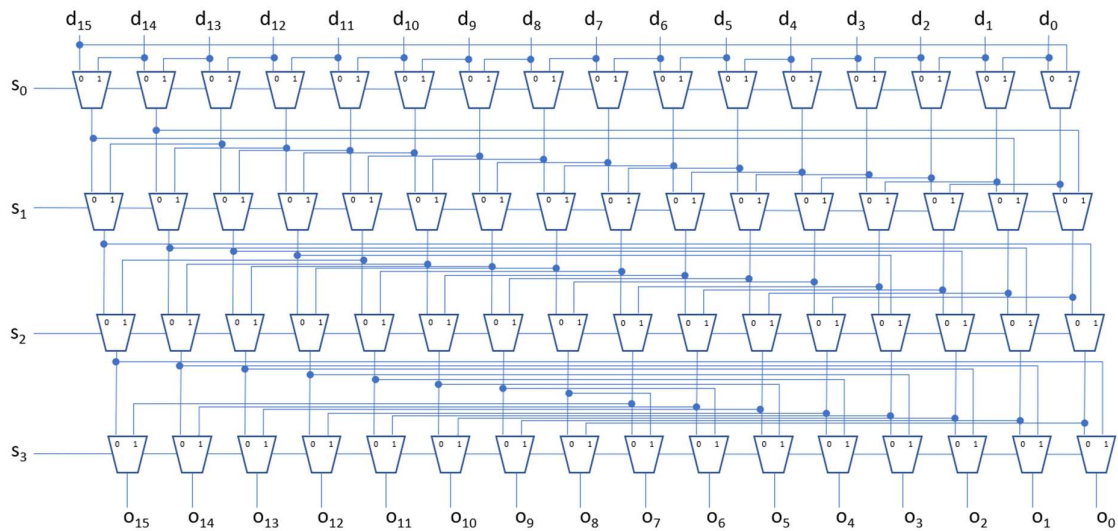
16-bit Shift Left Operation using Multiplexers:



16-bit Right Rotate Operation using Multiplexers:



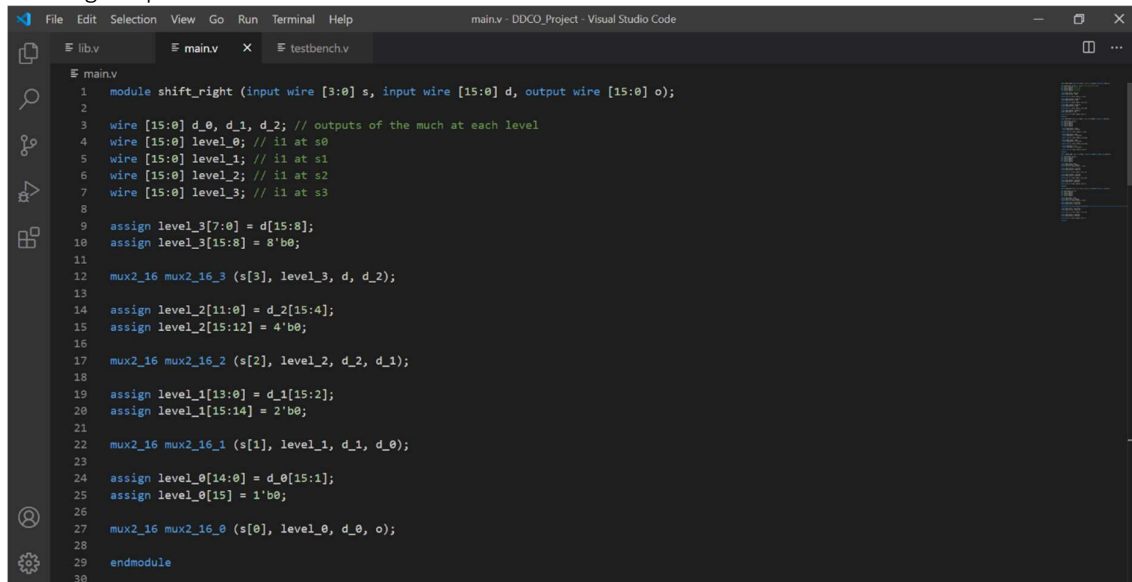
16-bit Left Rotate Operation using Multiplexers:



MAIN VERILOG CODE:

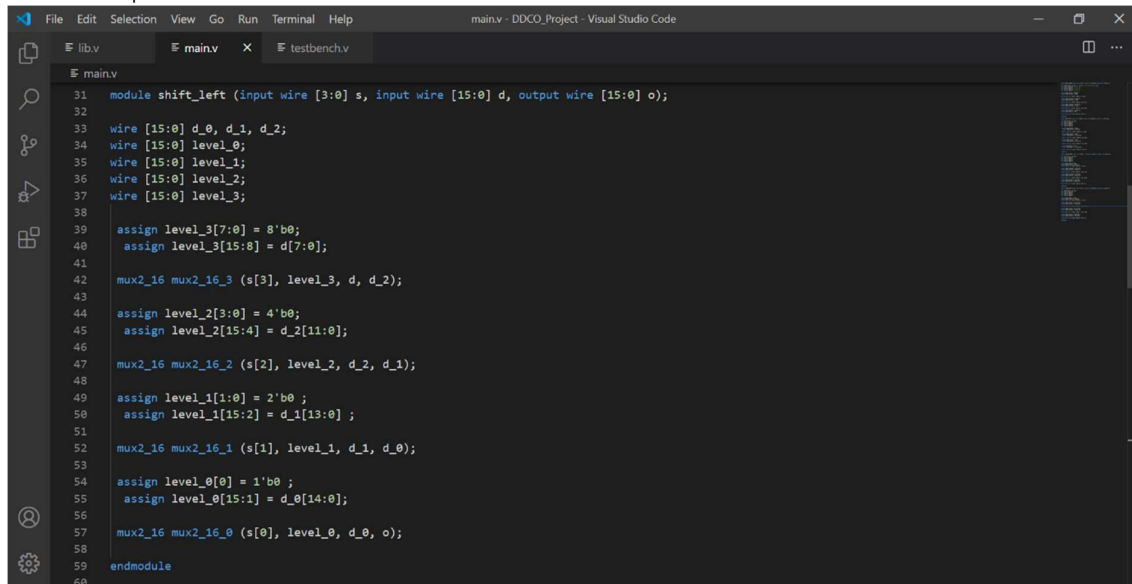
main.v

Shift Right Operation



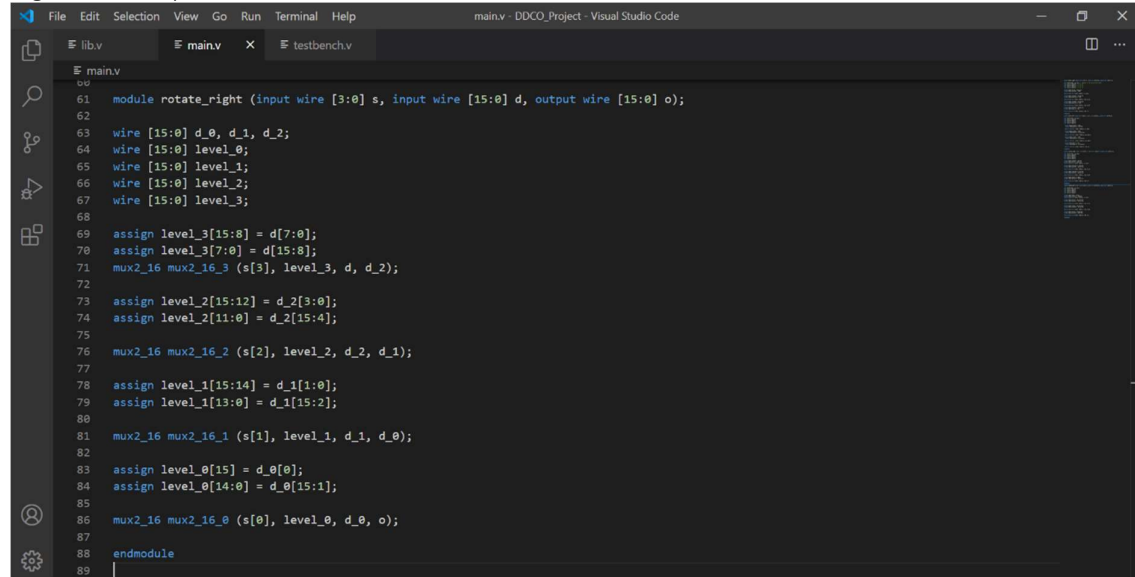
```
1 module shift_right (input wire [3:0] s, input wire [15:0] d, output wire [15:0] o);
2
3   wire [15:0] d_0, d_1, d_2; // outputs of the mux at each level
4   wire [15:0] level_0; // 11 at s0
5   wire [15:0] level_1; // 11 at s1
6   wire [15:0] level_2; // 11 at s2
7   wire [15:0] level_3; // 11 at s3
8
9   assign level_3[7:0] = d[15:8];
10  assign level_3[15:8] = 8'b0;
11
12  mux2_16 mux2_16_3 (s[3], level_3, d, d_2);
13
14  assign level_2[11:0] = d_2[15:4];
15  assign level_2[15:12] = 4'b0;
16
17  mux2_16 mux2_16_2 (s[2], level_2, d_2, d_1);
18
19  assign level_1[13:0] = d_1[15:2];
20  assign level_1[15:14] = 2'b0;
21
22  mux2_16 mux2_16_1 (s[1], level_1, d_1, d_0);
23
24  assign level_0[14:0] = d_0[15:1];
25  assign level_0[15] = 1'b0;
26
27  mux2_16 mux2_16_0 (s[0], level_0, d_0, o);
28
29 endmodule
30
```

Shift Left Operation



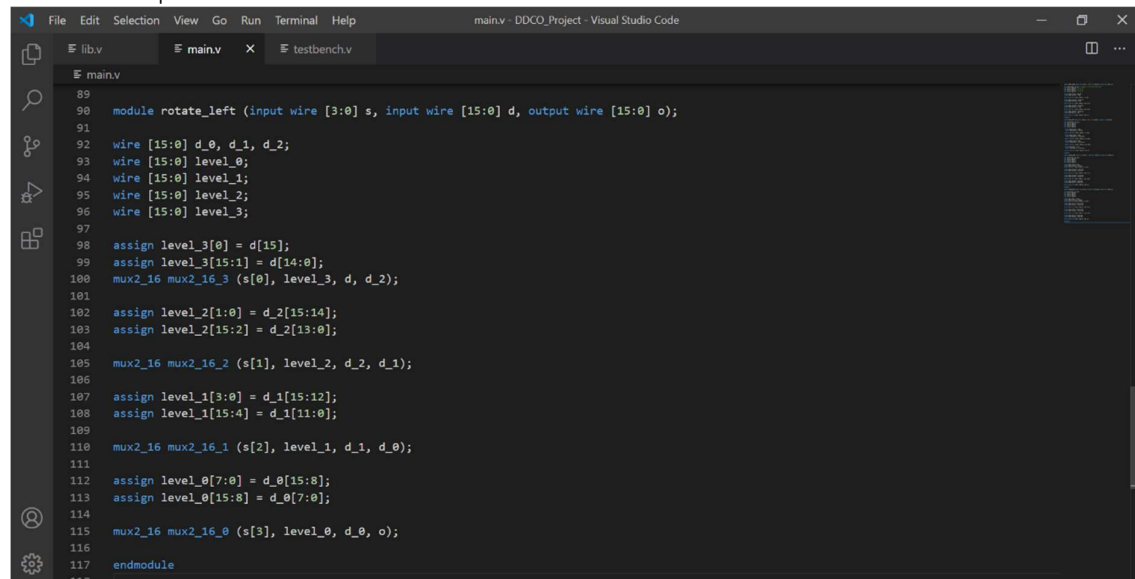
```
31 module shift_left (input wire [3:0] s, input wire [15:0] d, output wire [15:0] o);
32
33   wire [15:0] d_0, d_1, d_2;
34   wire [15:0] level_0;
35   wire [15:0] level_1;
36   wire [15:0] level_2;
37   wire [15:0] level_3;
38
39   assign level_3[7:0] = 8'b0;
40   assign level_3[15:8] = d[7:0];
41
42   mux2_16 mux2_16_3 (s[3], level_3, d, d_2);
43
44   assign level_2[3:0] = 4'b0;
45   assign level_2[15:4] = d_2[11:0];
46
47   mux2_16 mux2_16_2 (s[2], level_2, d_2, d_1);
48
49   assign level_1[1:0] = 2'b0 ;
50   assign level_1[15:2] = d_1[13:0] ;
51
52   mux2_16 mux2_16_1 (s[1], level_1, d_1, d_0);
53
54   assign level_0[0] = 1'b0 ;
55   assign level_0[15:1] = d_0[14:0];
56
57   mux2_16 mux2_16_0 (s[0], level_0, d_0, o);
58
59 endmodule
60
```

Right Rotate Operation



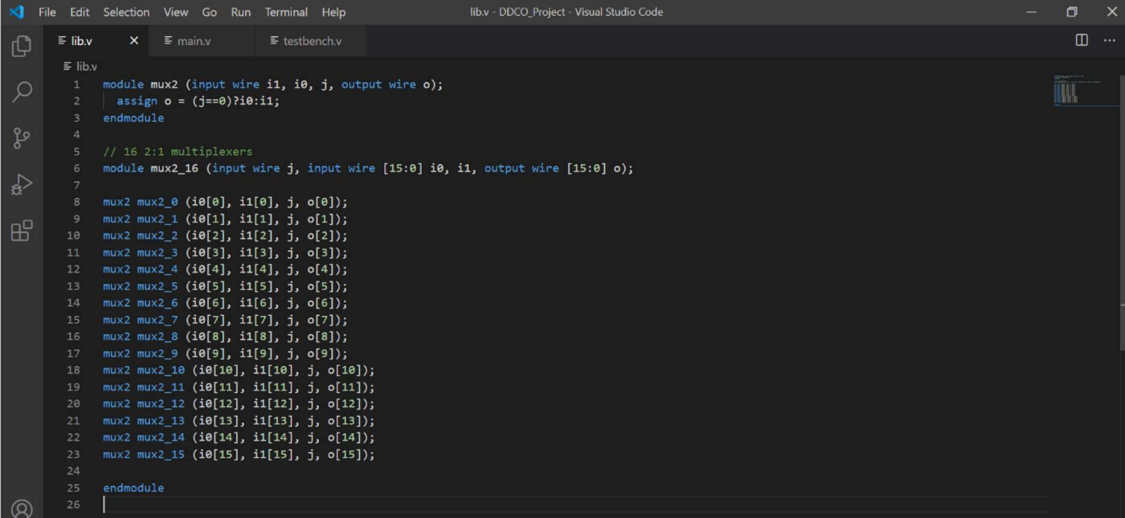
```
61 module rotate_right (input wire [3:0] s, input wire [15:0] d, output wire [15:0] o);
62
63 wire [15:0] d_0, d_1, d_2;
64 wire [15:0] level_0;
65 wire [15:0] level_1;
66 wire [15:0] level_2;
67 wire [15:0] level_3;
68
69 assign level_3[15:8] = d[7:0];
70 assign level_3[7:0] = d[15:8];
71 mux2_16 mux2_16_3 (s[3], level_3, d, d_2);
72
73 assign level_2[15:12] = d_2[3:0];
74 assign level_2[11:0] = d_2[15:4];
75
76 mux2_16 mux2_16_2 (s[2], level_2, d_2, d_1);
77
78 assign level_1[15:14] = d_1[1:0];
79 assign level_1[13:0] = d_1[15:2];
80
81 mux2_16 mux2_16_1 (s[1], level_1, d_1, d_0);
82
83 assign level_0[15] = d_0[0];
84 assign level_0[14:0] = d_0[15:1];
85
86 mux2_16 mux2_16_0 (s[0], level_0, d_0, o);
87
88 endmodule
89
```

Left Rotate Operation



```
89
90 module rotate_left (input wire [3:0] s, input wire [15:0] d, output wire [15:0] o);
91
92 wire [15:0] d_0, d_1, d_2;
93 wire [15:0] level_0;
94 wire [15:0] level_1;
95 wire [15:0] level_2;
96 wire [15:0] level_3;
97
98 assign level_3[0] = d[15];
99 assign level_3[15:1] = d[14:0];
100 mux2_16 mux2_16_3 (s[0], level_3, d, d_2);
101
102 assign level_2[1:0] = d_2[15:14];
103 assign level_2[15:2] = d_2[13:0];
104
105 mux2_16 mux2_16_2 (s[1], level_2, d_2, d_1);
106
107 assign level_1[3:0] = d_1[15:12];
108 assign level_1[15:4] = d_1[11:0];
109
110 mux2_16 mux2_16_1 (s[2], level_1, d_1, d_0);
111
112 assign level_0[7:0] = d_0[15:8];
113 assign level_0[15:8] = d_0[7:0];
114
115 mux2_16 mux2_16_0 (s[3], level_0, d_0, o);
116
117 endmodule
118
```

lib.v



```
1 module mux2 (input wire i1, i0, j, output wire o);
2     assign o = (j==0)?i0:i1;
3 endmodule
4
5 // 16 2:1 multiplexers
6 module mux2_16 (input wire j, input wire [15:0] i0, i1, output wire [15:0] o);
7
8     mux2 mux2_0 (i0[0], i1[0], j, o[0]);
9     mux2 mux2_1 (i0[1], i1[1], j, o[1]);
10    mux2 mux2_2 (i0[2], i1[2], j, o[2]);
11    mux2 mux2_3 (i0[3], i1[3], j, o[3]);
12    mux2 mux2_4 (i0[4], i1[4], j, o[4]);
13    mux2 mux2_5 (i0[5], i1[5], j, o[5]);
14    mux2 mux2_6 (i0[6], i1[6], j, o[6]);
15    mux2 mux2_7 (i0[7], i1[7], j, o[7]);
16    mux2 mux2_8 (i0[8], i1[8], j, o[8]);
17    mux2 mux2_9 (i0[9], i1[9], j, o[9]);
18    mux2 mux2_10 (i0[10], i1[10], j, o[10]);
19    mux2 mux2_11 (i0[11], i1[11], j, o[11]);
20    mux2 mux2_12 (i0[12], i1[12], j, o[12]);
21    mux2 mux2_13 (i0[13], i1[13], j, o[13]);
22    mux2 mux2_14 (i0[14], i1[14], j, o[14]);
23    mux2 mux2_15 (i0[15], i1[15], j, o[15]);
24
25 endmodule
26
```

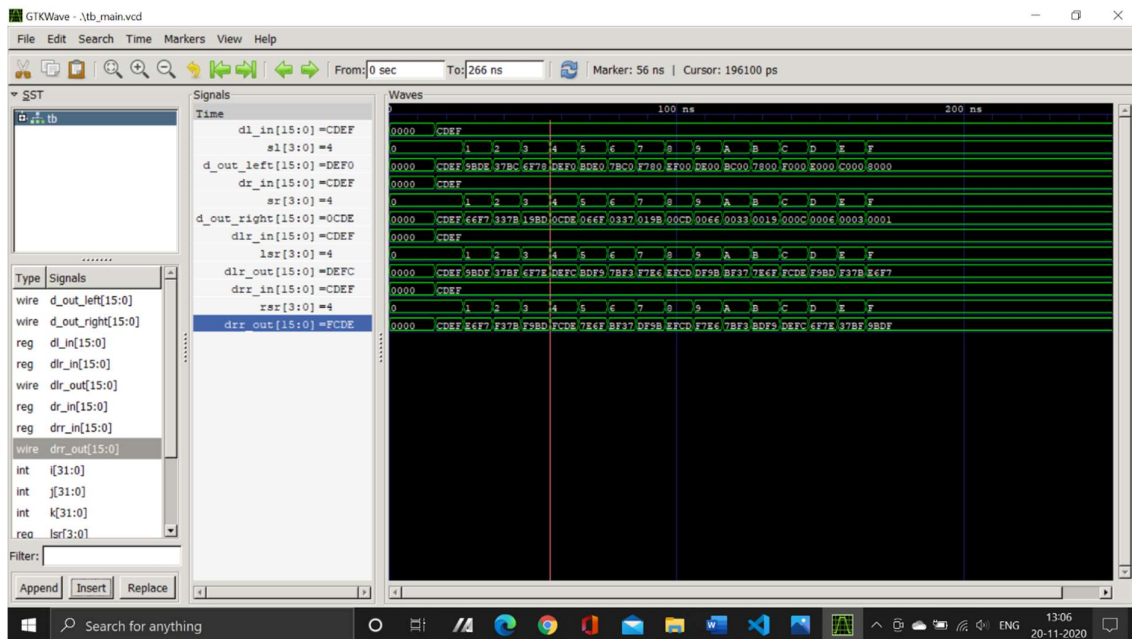

TEST BENCH FILE:

```
File Edit Selection View Go Run Terminal Help testbench.v - DDCO_Project - Visual Studio Code
lib.v main.v testbench.v X
testbench.v
1 `timescale 1 ns / 100 ps
2 `define TESTVECS 16
3
4 module tb;
5
6     reg [19:0] test_vecs [0:(`TESTVECS-1)];
7
8     integer i, j, k, p;
9     initial begin $dumpfile("tb_main.vcd"); $dumpvars(0,tb); end
10    initial begin
11        test_vecs[0][19:16] = 4'b0000; test_vecs[0][15:0] = 16'hcdef;
12        test_vecs[1][19:16] = 4'b0001; test_vecs[1][15:0] = 16'hcdef;
13        test_vecs[2][19:16] = 4'b0010; test_vecs[2][15:0] = 16'hcdef;
14        test_vecs[3][19:16] = 4'b0011; test_vecs[3][15:0] = 16'hcdef;
15        test_vecs[4][19:16] = 4'b0100; test_vecs[4][15:0] = 16'hcdef;
16        test_vecs[5][19:16] = 4'b0101; test_vecs[5][15:0] = 16'hcdef;
17        test_vecs[6][19:16] = 4'b0110; test_vecs[6][15:0] = 16'hcdef;
18        test_vecs[7][19:16] = 4'b0111; test_vecs[7][15:0] = 16'hcdef;
19        test_vecs[8][19:16] = 4'b1000; test_vecs[8][15:0] = 16'hcdef;
20        test_vecs[9][19:16] = 4'b1001; test_vecs[9][15:0] = 16'hcdef;
21        test_vecs[10][19:16] = 4'b1010; test_vecs[10][15:0] = 16'hcdef;
22        test_vecs[11][19:16] = 4'b1011; test_vecs[11][15:0] = 16'hcdef;
23        test_vecs[12][19:16] = 4'b1100; test_vecs[12][15:0] = 16'hcdef;
24        test_vecs[13][19:16] = 4'b1101; test_vecs[13][15:0] = 16'hcdef;
25        test_vecs[14][19:16] = 4'b1110; test_vecs[14][15:0] = 16'hcdef;
26        test_vecs[15][19:16] = 4'b1111; test_vecs[15][15:0] = 16'hcdef;
27    end
```

```
File Edit Selection View Go Run Terminal Help testbench.v - DDCO_Project - Visual Studio Code
lib.v main.v testbench.v X
testbench.v
28    end
29    ////////////////////////////////// Left - Shift //////////////////////////////////
30    reg [3:0] sl;
31    reg [15:0] dl_in;
32    wire [15:0] d_out_right;
33
34    initial {sl, dl_in} = 0;
35    shift_left shift_left_test (sl, dl_in, d_out_left);
36    initial begin
37        #6 for(i=0;i<`TESTVECS;i=i+1)
38            begin #10 {sl, dl_in}=test_vecs[i]; end
39        #100 $finish;
40    end
41    ////////////////////////////////// Right - Shift //////////////////////////////////
42    reg [3:0] sr;
43    reg [15:0] dr_in;
44    wire [15:0] d_out_left;
45
46    initial {sr, dr_in} = 0;
47    shift_right shift_right_test (sr, dr_in, d_out_right);
48    initial begin
49        #6 for(j=0;j<`TESTVECS;j=j+1)
50            begin #10 {sr, dr_in}=test_vecs[j]; end
51        #100 $finish;
52    end
```

```
testbench.v - DDCO_Project - Visual Studio Code
lib.v main.v testbench.v X
testbench.v
52 ////////////////////////////////////////////////// Rotate - Right ///////////////////////////////////
53 reg [3:0] rsr;
54 reg [15:0] drr_in;
55 wire [15:0] drr_out;
56
57 initial {rsr, drr_in} = 0;
58 rotate_right rotate_right_test (rsr, drr_in, drr_out);
59 initial begin
60     #6 for(k=0;k<'TESTVECS;k=k+1)
61         #10 {rsr, drr_in}=test_vecs[k];
62     #100 $finish;
63 end
64 ////////////////////////////////////////////////// Rotate - Left ///////////////////////////////////
65 reg [3:0] lsr;
66 reg [15:0] dlr_in;
67 wire [15:0] dlr_out;
68
69 initial {lsr, dlr_in} = 0;
70 rotate_left rotate_left_test (lsr, dlr_in, dlr_out);
71 initial begin
72     #6 for(p=0;p<'TESTVECS;p=p+1)
73         #10 {lsr, dlr_in}=test_vecs[p];
74     #100 $finish;
75 end
76 //////////////////////////////////////////////////
77 endmodule
78
```

SCREEN SHOT OF THE OUTPUT:



The cursur shows the 4-bit left shift, right shift, left rotate and right rotate of cdef.