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

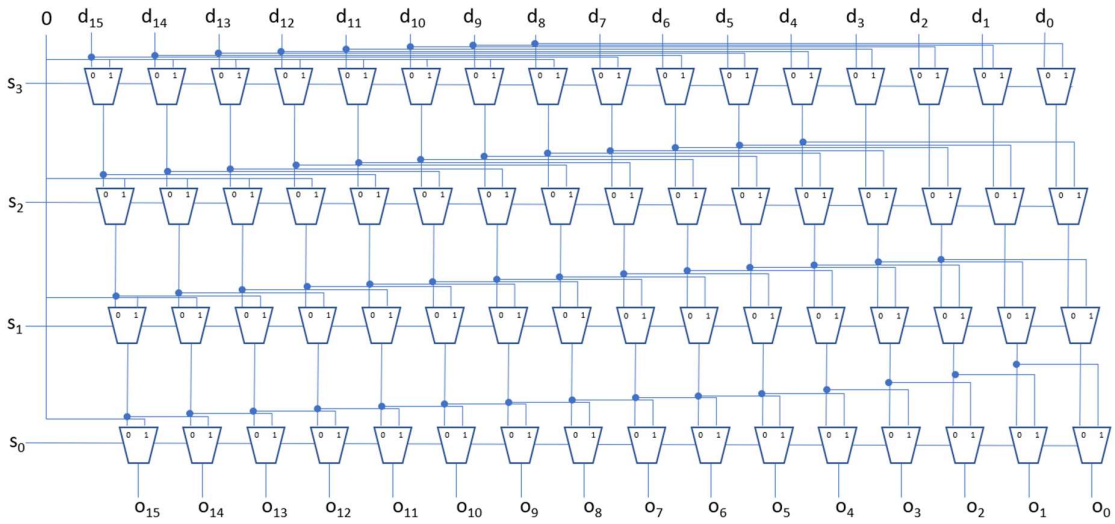
TABLE OF CONTENTS

ABSTRACT OF THE PROJECT:

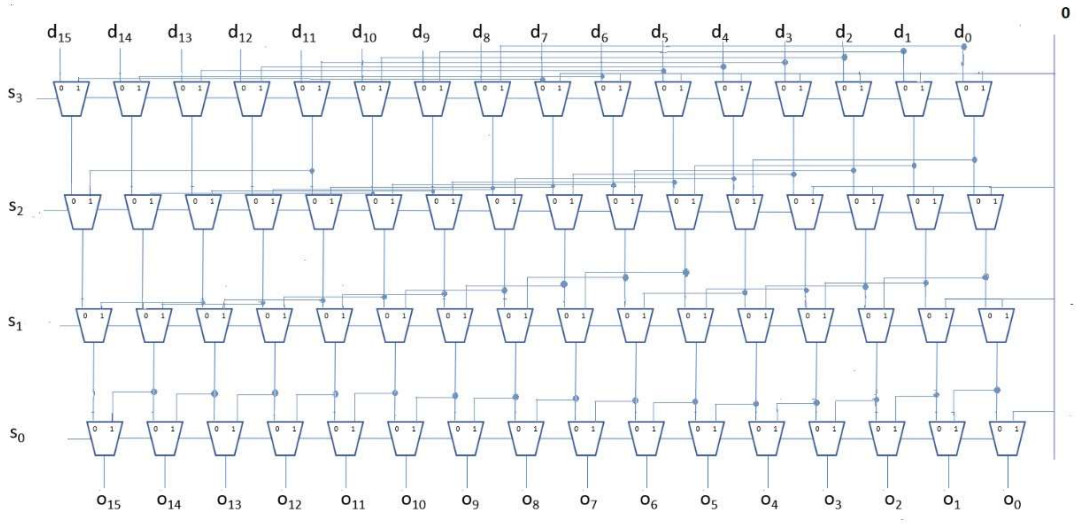
As the name implies, a **shifter** shifts a binary number left or right by a specified number of positions. And a **rotator** rotates number in a circle such that empty spots are filled with bits shifted off the other end. Shifting and rotating data is required in several applications including arithmetic 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:

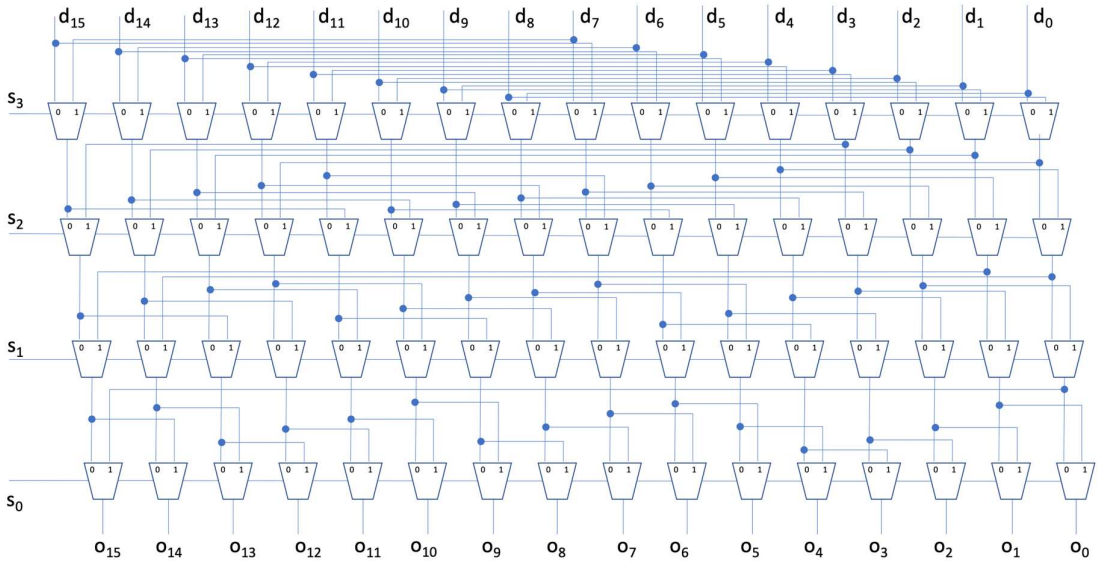
16-bit Shift Right Operation using Multiplexers:



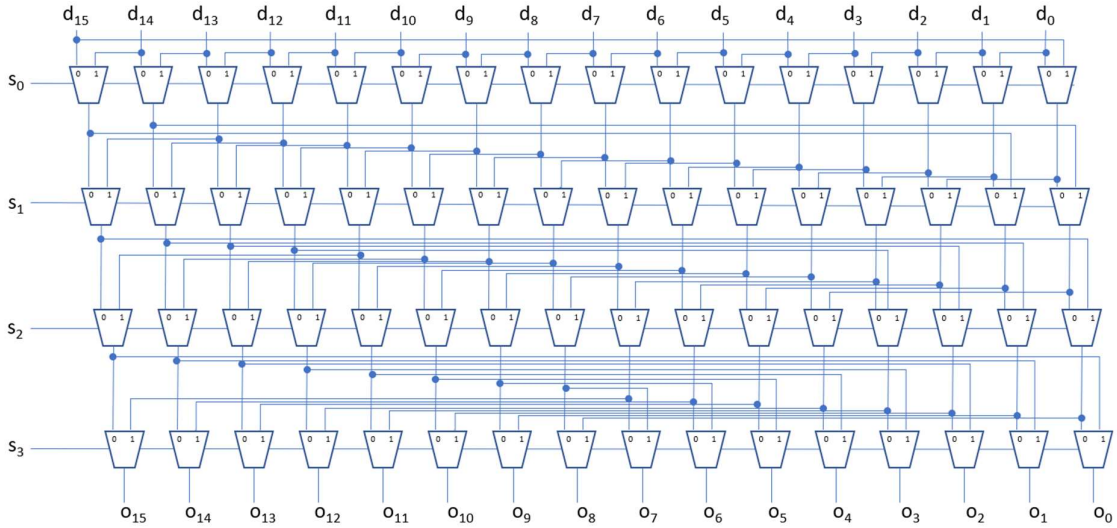
16-bit Shift Left Operation using Multiplexers:



16-bit Right Rotate Operation using Multiplexers:



16-bit Left Rotate Operation using Multiplexers:



MAIN VERILOG CODE:

main.v

Shift Right Operation

```verilog
module shift_right (input wire [3:0] s, input wire [15:0] d, output wire [15:0] o);

wire [15:0] d_0, d_1, d_2; // outputs of the much at each level
wire [15:0] level_0; // i1 at s0
wire [15:0] level_1; // i1 at s1
wire [15:0] level_2; // i1 at s2
wire [15:0] level_3; // i1 at s3

assign level_3[7:0] = d[15:8];
assign level_3[15:8] = 8'b0;

mux2_16 mux2_16_3 (s[3], level_3, d, d_2);

assign level_2[11:0] = d_2[15:4];
assign level_2[15:12] = 4'b0;

mux2_16 mux2_16_2 (s[2], level_2, d_2, d_1);

assign level_1[13:0] = d_1[15:2];
assign level_1[15:14] = 2'b0;

mux2_16 mux2_16_1 (s[1], level_1, d_1, d_0);

assign level_0[14:0] = d_0[15:1];
assign level_0[15] = 1'b0;

mux2_16 mux2_16_0 (s[0], level_0, d_0, o);

endmodule
```

Shift Left Operation

```verilog
module shift_left (input wire [3:0] s, input wire [15:0] d, output wire [15:0] o);

wire [15:0] d_0, d_1, d_2;
wire [15:0] level_0;
wire [15:0] level_1;
wire [15:0] level_2;
wire [15:0] level_3;

 assign level_3[7:0] = 8'b0;
  assign level_3[15:8] = d[7:0];

 mux2_16 mux2_16_3 (s[3], level_3, d, d_2);

 assign level_2[3:0] = 4'b0;
  assign level_2[15:4] = d_2[11:0];

 mux2_16 mux2_16_2 (s[2], level_2, d_2, d_1);

 assign level_1[1:0] = 2'b0 ;
  assign level_1[15:2] = d_1[13:0] ;

 mux2_16 mux2_16_1 (s[1], level_1, d_1, d_0);

 assign level_0[0] = 1'b0 ;
  assign level_0[15:1] = d_0[14:0];

 mux2_16 mux2_16_0 (s[0], level_0, d_0, o);

endmodule
```

Right Rotate Operation

```
module rotate_right (input wire [3:0] s, input wire [15:0] d, output wire [15:0] o);

wire [15:0] d_0, d_1, d_2;
wire [15:0] level_0;
wire [15:0] level_1;
wire [15:0] level_2;
wire [15:0] level_3;

assign level_3[15:8] = d[7:0];
assign level_3[7:0] = d[15:8];
mux2_16 mux2_16_3 (s[3], level_3, d, d_2);

assign level_2[15:12] = d_2[3:0];
assign level_2[11:0] = d_2[15:4];

mux2_16 mux2_16_2 (s[2], level_2, d_2, d_1);

assign level_1[15:14] = d_1[1:0];
assign level_1[13:0] = d_1[15:2];

mux2_16 mux2_16_1 (s[1], level_1, d_1, d_0);

assign level_0[15] = d_0[0];
assign level_0[14:0] = d_0[15:1];

mux2_16 mux2_16_0 (s[0], level_0, d_0, o);

endmodule
```

Left Rotate Operation

```
module rotate_left (input wire [3:0] s, input wire [15:0] d, output wire [15:0] o);

wire [15:0] d_0, d_1, d_2;
wire [15:0] level_0;
wire [15:0] level_1;
wire [15:0] level_2;
wire [15:0] level_3;

assign level_3[0] = d[15];
assign level_3[15:1] = d[14:0];
mux2_16 mux2_16_3 (s[0], level_3, d, d_2);

assign level_2[1:0] = d_2[15:14];
assign level_2[15:2] = d_2[13:0];

mux2_16 mux2_16_2 (s[1], level_2, d_2, d_1);

assign level_1[3:0] = d_1[15:12];
assign level_1[15:4] = d_1[11:0];

mux2_16 mux2_16_1 (s[2], level_1, d_1, d_0);

assign level_0[7:0] = d_0[15:8];
assign level_0[15:8] = d_0[7:0];

mux2_16 mux2_16_0 (s[3], level_0, d_0, o);

endmodule
```

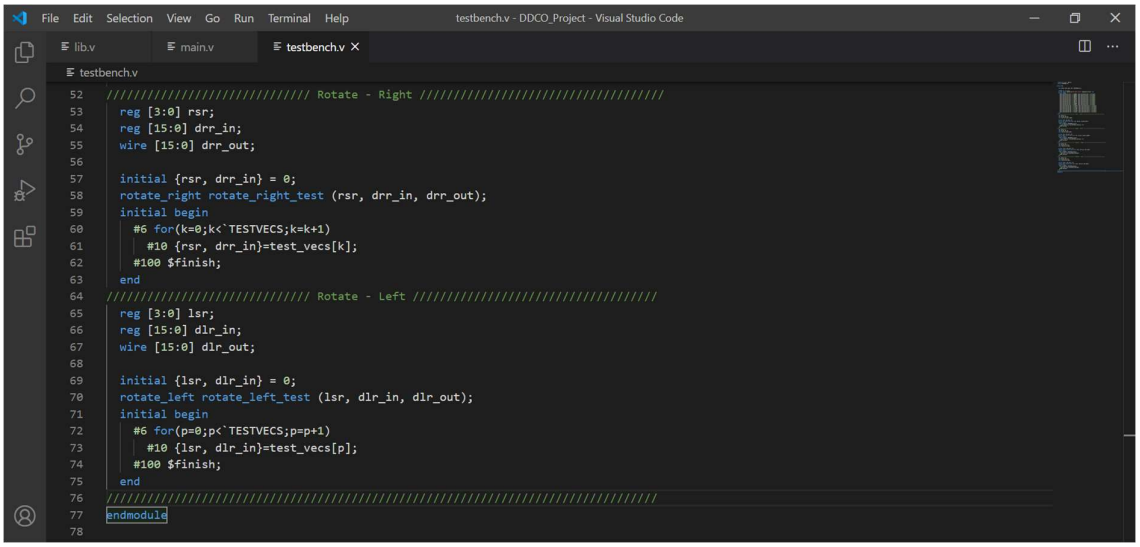
lib.v

```
module mux2 (input wire i1, i0, j, output wire o);
  assign o = (j==0)?i0:i1;
endmodule

// 16 2:1 multiplexers
module mux2_16 (input wire j, input wire [15:0] i0, i1, output wire [15:0] o);

mux2 mux2_0 (i0[0], i1[0], j, o[0]);
mux2 mux2_1 (i0[1], i1[1], j, o[1]);
mux2 mux2_2 (i0[2], i1[2], j, o[2]);
mux2 mux2_3 (i0[3], i1[3], j, o[3]);
mux2 mux2_4 (i0[4], i1[4], j, o[4]);
mux2 mux2_5 (i0[5], i1[5], j, o[5]);
mux2 mux2_6 (i0[6], i1[6], j, o[6]);
mux2 mux2_7 (i0[7], i1[7], j, o[7]);
mux2 mux2_8 (i0[8], i1[8], j, o[8]);
mux2 mux2_9 (i0[9], i1[9], j, o[9]);
mux2 mux2_10 (i0[10], i1[10], j, o[10]);
mux2 mux2_11 (i0[11], i1[11], j, o[11]);
mux2 mux2_12 (i0[12], i1[12], j, o[12]);
mux2 mux2_13 (i0[13], i1[13], j, o[13]);
mux2 mux2_14 (i0[14], i1[14], j, o[14]);
mux2 mux2_15 (i0[15], i1[15], j, o[15]);

endmodule
```

TEST BENCH FILE:

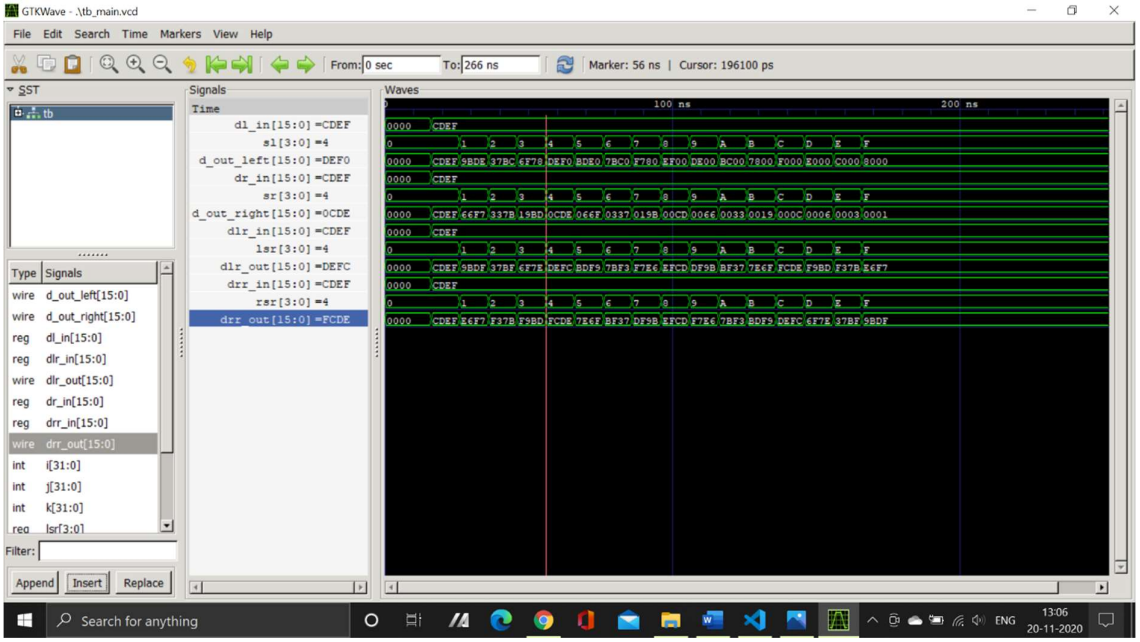
```verilog
`timescale 1 ns / 100 ps
`define TESTVECS 16

module tb;

  reg [19:0] test_vecs [0:(`TESTVECS-1)];

  integer i, j, k, p;
  initial begin $dumpfile("tb_main.vcd"); $dumpvars(0,tb); end
  initial begin
    test_vecs[0][19:16] = 4'b0000; test_vecs[0][15:0] = 16'hcdef;
    test_vecs[1][19:16] = 4'b0001; test_vecs[1][15:0] = 16'hcdef;
    test_vecs[2][19:16] = 4'b0010; test_vecs[2][15:0] = 16'hcdef;
    test_vecs[3][19:16] = 4'b0011; test_vecs[3][15:0] = 16'hcdef;
    test_vecs[4][19:16] = 4'b0100; test_vecs[4][15:0] = 16'hcdef;
    test_vecs[5][19:16] = 4'b0101; test_vecs[5][15:0] = 16'hcdef;
    test_vecs[6][19:16] = 4'b0110; test_vecs[6][15:0] = 16'hcdef;
    test_vecs[7][19:16] = 4'b0111; test_vecs[7][15:0] = 16'hcdef;
    test_vecs[8][19:16] = 4'b1000; test_vecs[8][15:0] = 16'hcdef;
    test_vecs[9][19:16] = 4'b1001; test_vecs[9][15:0] = 16'hcdef;
    test_vecs[10][19:16] = 4'b1010; test_vecs[10][15:0] = 16'hcdef;
    test_vecs[11][19:16] = 4'b1011; test_vecs[11][15:0] = 16'hcdef;
    test_vecs[12][19:16] = 4'b1100; test_vecs[12][15:0] = 16'hcdef;
    test_vecs[13][19:16] = 4'b1101; test_vecs[13][15:0] = 16'hcdef;
    test_vecs[14][19:16] = 4'b1110; test_vecs[14][15:0] = 16'hcdef;
    test_vecs[15][19:16] = 4'b1111; test_vecs[15][15:0] = 16'hcdef;
  end
```

```verilog
  end
////////////////////////////// Left - Shift ////////////////////////////////////
  reg [3:0] sl;
  reg [15:0] dl_in;
  wire [15:0] d_out_right;

  initial {sl, dl_in} = 0;
  shift_left shift_left_test (sl, dl_in, d_out_left);
  initial begin
    #6 for(i=0;i<`TESTVECS;i=i+1)
      begin #10 {sl, dl_in}=test_vecs[i]; end
    #100 $finish;
  end
////////////////////////////// Right - Shift ////////////////////////////////////
  reg [3:0] sr;
  reg [15:0] dr_in;
  wire [15:0] d_out_left;

  initial {sr, dr_in} = 0;
  shift_right shift_right_test (sr, dr_in, d_out_right);
  initial begin
    #6 for(j=0;j<`TESTVECS;j=j+1)
      begin #10 {sr, dr_in}=test_vecs[j]; end
    #100 $finish;
  end
```



```
////////////////////////////// Rotate - Right ////////////////////////////////////
  reg [3:0] rsr;
  reg [15:0] drr_in;
  wire [15:0] drr_out;

  initial {rsr, drr_in} = 0;
  rotate_right rotate_right_test (rsr, drr_in, drr_out);
  initial begin
    #6 for(k=0;k<`TESTVECS;k=k+1)
      #10 {rsr, drr_in}=test_vecs[k];
    #100 $finish;
  end
////////////////////////////// Rotate - Left ////////////////////////////////////
  reg [3:0] lsr;
  reg [15:0] dlr_in;
  wire [15:0] dlr_out;

  initial {lsr, dlr_in} = 0;
  rotate_left rotate_left_test (lsr, dlr_in, dlr_out);
  initial begin
    #6 for(p=0;p<`TESTVECS;p=p+1)
      #10 {lsr, dlr_in}=test_vecs[p];
    #100 $finish;
  end
////////////////////////////////////////////////////////////////////////////////
endmodule
```

SCREEN SHOT OF THE OUTPUT:



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