VLSI LAB Experiment-2

Implementation of Barrel shifter

Harshavardhan Alimi

18EC10021

1.Barrel shifter - left shift or left rotate

Aim :-

The aim of this experiment is to design a barrel shifter that performs logical left shift or left rotate operations based on the control input. The 2:1 MUX should be used as the building blocks for the design.

Schematic diagrams and labels :-

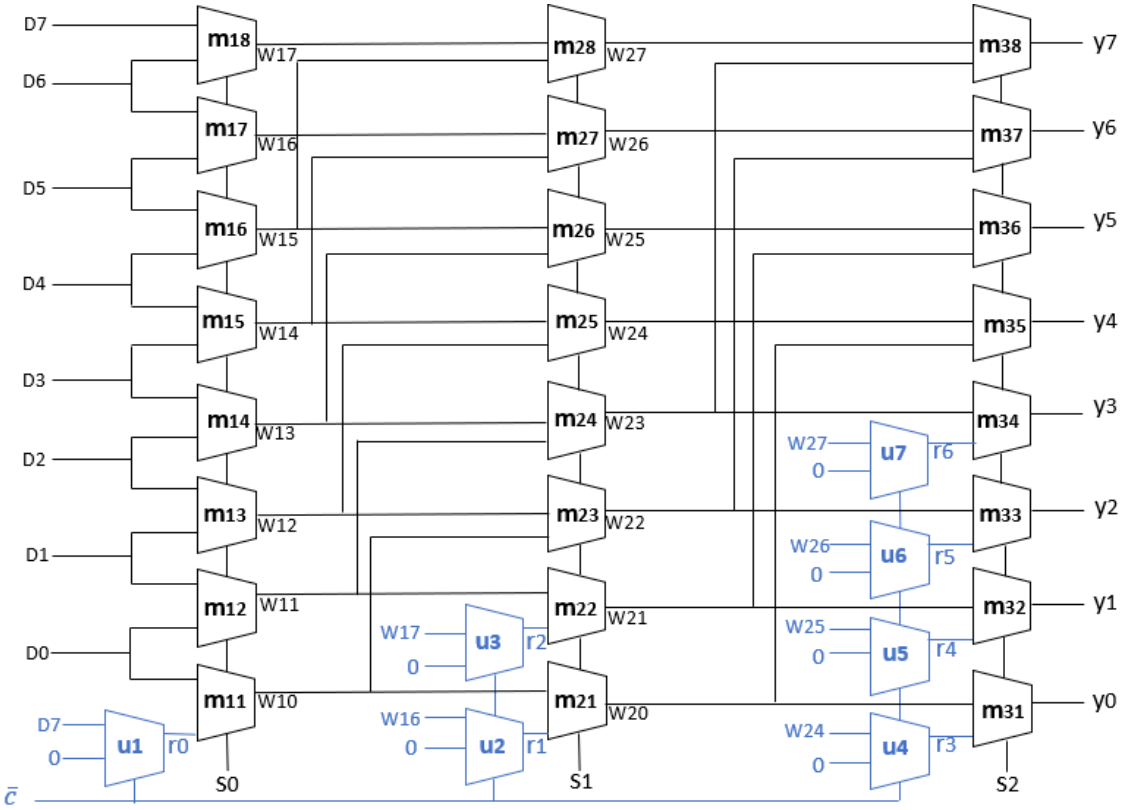


Fig.1:- implementation of barrel shifter using only 2:1 multiplexers, which performs left shift or left rotate operation based on control input

Description :-

- Number of multiplexers used:31
- Below is the code for implementation of barrel shifter, In the code:-
- 8 - bit Input : x
- Control input : cntrl
- 3-bit Select line : sel (to decide how much shifting or rotation takes place).
- Output displayed : z
- With control bit=0 the code will perform left shift operation and with control bit=1 the code will perform left rotate operation.
- 3-bit select(sel) input will decide how much shifting or rotation should take place.i.e., if sel=000 means no shifting or rotation takes place,if sel=010 means the 8-bit input x gets shifted or rotated by 2 steps(operations will take place 2 times).
- u1,u2,u3,u4,u5,u6,u7 named multiplexers in the above figure are used to determine the correct value(zeros for padding while shifting and specific values while for rotation) for the selected input at the control line.
- 3 layers of 8 2:1 multiplexers were used,with only 1st layer, maximum 1 shifting(or rotation) take place(as it consider single bit) and similarly with only 2nd and 1st layer,maximum 3 shiftings(or rotations) take place(as it consider 2 bits) , similarly maximum 7 shiftings(or rotations) with all the 3 layers(as it consider 3 bits).

Verilog codes :-

```
module design_2_1_MUX(
      input a0,
      input a1,
      input s,
      output z
      );
  wire T1,T2,sbar;
  not(sbar,s);
```

```
   and (T1,a1,s) , (T2,a0,sbar);
   or (z,T1,T2);
endmodule

module layer_1_8_8_MUX(
       input [7:0] x,
       input  sel,
       input cntrl,
       output [7:0] z
       );
   wire ip1;
   design_2_1_MUX a1(1'b0,x[7],cntrl,ip1);
   design_2_1_MUX b1(x[0],ip1,sel,z[0]);
   design_2_1_MUX b2(x[1],x[0],sel,z[1]);
   design_2_1_MUX b3(x[2],x[1],sel,z[2]);
   design_2_1_MUX b4(x[3],x[2],sel,z[3]);
   design_2_1_MUX b5(x[4],x[3],sel,z[4]);
   design_2_1_MUX b6(x[5],x[4],sel,z[5]);
   design_2_1_MUX b7(x[6],x[5],sel,z[6]);
   design_2_1_MUX b8(x[7],x[6],sel,z[7]);
endmodule

module layer_2_8_8_MUX(
       input [7:0] x,
       input  sel,
       input cntrl,
       output [7:0] z
```

```
      );
   wire ip1,ip2;
   design_2_1_MUX a1(1'b0,x[6],cntrl,ip1);
   design_2_1_MUX a2(1'b0,x[7],cntrl,ip2);
   design_2_1_MUX b1(x[0],ip1,sel,z[0]);
   design_2_1_MUX b2(x[1],ip2,sel,z[1]);
   design_2_1_MUX b3(x[2],x[0],sel,z[2]);
   design_2_1_MUX b4(x[3],x[1],sel,z[3]);
   design_2_1_MUX b5(x[4],x[2],sel,z[4]);
   design_2_1_MUX b6(x[5],x[3],sel,z[5]);
   design_2_1_MUX b7(x[6],x[4],sel,z[6]);
   design_2_1_MUX b8(x[7],x[5],sel,z[7]);
endmodule


module layer_3_8_8_MUX(
      input [7:0] x,
      input  sel,
      input cntrl,
      output [7:0] z
      );
   wire ip1,ip2,ip3,ip4;
   design_2_1_MUX a1(1'b0,x[4],cntrl,ip1);
   design_2_1_MUX a2(1'b0,x[5],cntrl,ip2);
   design_2_1_MUX a3(1'b0,x[6],cntrl,ip3);
   design_2_1_MUX a4(1'b0,x[7],cntrl,ip4);
   design_2_1_MUX b1(x[0],ip1,sel,z[0]);
   design_2_1_MUX b2(x[1],ip2,sel,z[1]);
```

```
   design_2_1_MUX b3(x[2],ip3,sel,z[2]);
   design_2_1_MUX b4(x[3],ip4,sel,z[3]);
   design_2_1_MUX b5(x[4],x[0],sel,z[4]);
   design_2_1_MUX b6(x[5],x[1],sel,z[5]);
   design_2_1_MUX b7(x[6],x[2],sel,z[6]);
   design_2_1_MUX b8(x[7],x[3],sel,z[7]);
endmodule

module structural_rotational(
       input [7:0] x,
       input [2:0] sel,
       input cntrl,
       output [7:0] z
       );
   wire [7:0]z1;
   wire [7:0]z2;
   layer_1_8_8_MUX l1(x,sel[0],cntrl,z1);
   layer_2_8_8_MUX l2(z1,sel[1],cntrl,z2);
   layer_3_8_8_MUX l3(z2,sel[2],cntrl,z);
endmodule

module structural_rotational_tb;
   // Inputs
   reg [7:0] x;
   reg  [2:0]sel;
   reg cntrl;
   // Outputs
```

```
wire [7:0]z;
// Instantiate the Unit Under Test (UUT)
structural_rotational uut (
    .x(x),
    .sel(sel),
    .cntrl(cntrl),
    .z(z)
);
initial begin
    $dumpfile("test.vcd");
    $dumpvars(0,structural_rotational_tb);
end

initial begin
// Initialize Inputs
#0  x = 8'b11001100;sel=3'b000 ;cntrl=1'b0;
#10 sel=3'b001 ;cntrl=1'b0;
#10 sel=3'b010 ;cntrl=1'b0;
#10 sel=3'b011 ;cntrl=1'b0;
#10 sel=3'b100 ;cntrl=1'b0;
#10 sel=3'b101 ;cntrl=1'b0;
#10 sel=3'b110 ;cntrl=1'b0;
#10 sel=3'b111 ;cntrl=1'b0;
#10 sel=3'b000 ;cntrl=1'b1;
#10 sel=3'b001 ;cntrl=1'b1;
#10 sel=3'b010 ;cntrl=1'b1;
#10 sel=3'b011 ;cntrl=1'b1;
```

```
    #10 sel=3'b100 ;cntrl=1'b1;
    #10 sel=3'b101 ;cntrl=1'b1;
    #10 sel=3'b110 ;cntrl=1'b1;
    #10 sel=3'b111 ;cntrl=1'b1;
    #10;
    end

    initial begin
        $monitor("t=%3d x=%b,sel=%b,z=%b,cntrl=%b \n",$time,x,sel,z,cntrl);
    end
endmodule
```

Runtime log :-

VCD info: dumpfile test.vcd opened for output.

t= 0 x=11001100,sel=000,z=11001100,cntrl=0

t= 10 x=11001100,sel=001,z=10011000,cntrl=0

t= 20 x=11001100,sel=010,z=00110000,cntrl=0

t= 30 x=11001100,sel=011,z=01100000,cntrl=0

t= 40 x=11001100,sel=100,z=11000000,cntrl=0

t= 50 x=11001100,sel=101,z=10000000,cntrl=0

t= 60 x=11001100,sel=110,z=00000000,cntrl=0

t= 70 x=11001100,sel=111,z=00000000,cntrl=0

t= 80 x=11001100,sel=000,z=11001100,cntrl=1

t= 90 x=11001100,sel=001,z=10011001,cntrl=1

t=100 x=11001100,sel=010,z=00110011,cntrl=1

t=110 x=11001100,sel=011,z=01100110,cntrl=1

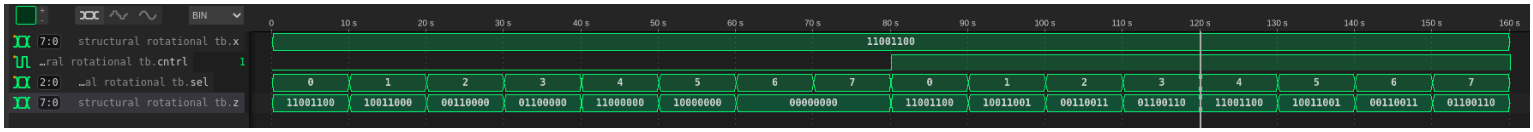
t=120 x=11001100,sel=100,z=11001100,cntrl=1

t=130 x=11001100,sel=101,z=10011001,cntrl=1

t=140 x=11001100,sel=110,z=00110011,cntrl=1

t=150 x=11001100,sel=111,z=01100110,cntrl=1

Results :-



Discussion :-

- Structural implementation of the 2:1 MUX, AND, OR, NOT modules have been used to design it.
- The code has been divided into three main layer functions which have been called sequentially to perform the rotation or shifting as per the select line and control input.
- The output wave forms were generated using GTKWAVE and it has been observed for all 8 values of the select line and 2 values of control (so 16 in total) to show the working of the barrel shifter.
- When control is 0, and select is 001 for example, the input 11001100 shifted to left with zero padding to give - 1001100110
- When control is 1 and select is 001 for example, the input 11001100 rotated towards left to give - 1001100111
- The test bench was written in the same le which generates the corresponding test.vcd file.

2.Barrel shifter - left shift or right shift

Aim :-

The aim of this experiment is to design a barrel shifter that performs logical left shift or right shift operations based on the control input. The 2:1 MUX should be used as the building blocks for the design(and to minimise the number of multiplexers used).

Schematic diagrams and labels :-

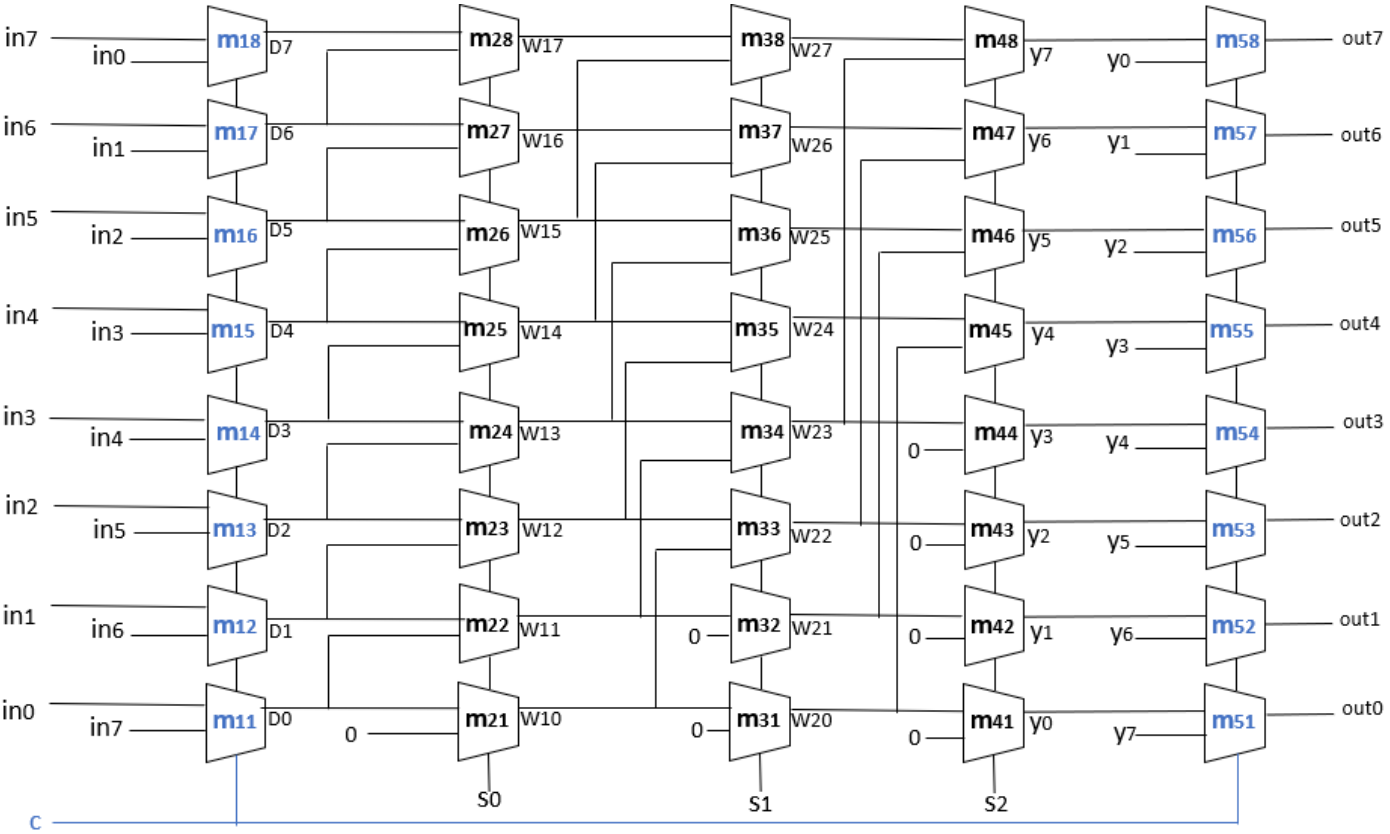


Fig2:- implementation of barrel shifter using only 2:1 multiplexers, which performs left shift or left rotate operation based on control input.

Description :-

- Number of multiplexers used:40
- Below is the code for implementation of barrel shifter, In the code:-
- 8 - bit Input : x
- Control input : cntrl
- 3-bit Select line : sel (to decide the number of shifting operations to take place).
- Output displayed : z
- With control bit=0 the code will perform left shift operation and with control bit=1 the code will perform right shift operation.
- 3-bit select(sel) input will decide how much shifting should take place.i.e., if sel=000 means no shifting takes place,if sel=010 means the 8-bit input x gets shifted or rotated by 2 steps(operations will take place 2 times).
- 5 layers of 8 2:1 multiplexers were used,1st and last layer(both layers implementation is same) will make mirror of the bit based on the control input with control input=0 no mirroring of bits at 1st and last layer and with control bit=1 mirroring of bits at first and last layer,for example if 10101010 is input to any one of the layer then output through that layer will 10101010, 01010101 when control bit=0 ,1 respectively.

Verilog codes :-

```
module design_2_1_MUX(
      input a0,
      input a1,
      input s,
      output z
      );
  wire T1,T2,sbar;
  not(sbar,s);
  and (T1,a1,s) , (T2,a0,sbar);
  or (z,T1,T2);
```

```
endmodule


module layer_0_8_8_MUX(
       input [7:0] x,
       input cntrl,
       output [7:0] z
       );
   design_2_1_MUX b1(x[0],x[7],cntrl,z[0]);
   design_2_1_MUX b2(x[1],x[6],cntrl,z[1]);
   design_2_1_MUX b3(x[2],x[5],cntrl,z[2]);
   design_2_1_MUX b4(x[3],x[4],cntrl,z[3]);
   design_2_1_MUX b5(x[4],x[3],cntrl,z[4]);
   design_2_1_MUX b6(x[5],x[2],cntrl,z[5]);
   design_2_1_MUX b7(x[6],x[1],cntrl,z[6]);
   design_2_1_MUX b8(x[7],x[0],cntrl,z[7]);
endmodule


module layer_1_8_8_MUX(
       input [7:0] x,
       input  sel,
       output [7:0] z
       );
   design_2_1_MUX b1(x[0],1'b0,sel,z[0]);
   design_2_1_MUX b2(x[1],x[0],sel,z[1]);
   design_2_1_MUX b3(x[2],x[1],sel,z[2]);
   design_2_1_MUX b4(x[3],x[2],sel,z[3]);
   design_2_1_MUX b5(x[4],x[3],sel,z[4]);
```

```
   design_2_1_MUX b6(x[5],x[4],sel,z[5]);
   design_2_1_MUX b7(x[6],x[5],sel,z[6]);
   design_2_1_MUX b8(x[7],x[6],sel,z[7]);
endmodule


module layer_2_8_8_MUX(
       input [7:0] x,
       input  sel,
       output [7:0] z
       );
   design_2_1_MUX b1(x[0],1'b0,sel,z[0]);
   design_2_1_MUX b2(x[1],1'b0,sel,z[1]);
   design_2_1_MUX b3(x[2],x[0],sel,z[2]);
   design_2_1_MUX b4(x[3],x[1],sel,z[3]);
   design_2_1_MUX b5(x[4],x[2],sel,z[4]);
   design_2_1_MUX b6(x[5],x[3],sel,z[5]);
   design_2_1_MUX b7(x[6],x[4],sel,z[6]);
   design_2_1_MUX b8(x[7],x[5],sel,z[7]);
endmodule


module layer_3_8_8_MUX(
       input [7:0] x,
       input  sel,
       output [7:0] z
       );
   design_2_1_MUX b1(x[0],1'b0,sel,z[0]);
   design_2_1_MUX b2(x[1],1'b0,sel,z[1]);
```

```
   design_2_1_MUX b3(x[2],1'b0,sel,z[2]);
   design_2_1_MUX b4(x[3],1'b0,sel,z[3]);
   design_2_1_MUX b5(x[4],x[0],sel,z[4]);
   design_2_1_MUX b6(x[5],x[1],sel,z[5]);
   design_2_1_MUX b7(x[6],x[2],sel,z[6]);
   design_2_1_MUX b8(x[7],x[3],sel,z[7]);
endmodule

module barrel_shifter(
       input [7:0] x,
       input [2:0] sel,
       input cntrl,
       output [7:0] z
       );
   wire [7:0]z0;
   wire [7:0]z1;
   wire [7:0]z2;
   wire [7:0]z3;
   layer_0_8_8_MUX l0(x,cntrl,z0);
   layer_1_8_8_MUX l1(z0,sel[0],z1);
   layer_2_8_8_MUX l2(z1,sel[1],z2);
   layer_3_8_8_MUX l3(z2,sel[2],z3);
   layer_0_8_8_MUX l4(z3,cntrl,z);
endmodule

module barrel_shifter_tb;
   // Inputs
```

```
reg [7:0] x;
reg  [2:0]sel;
reg cntrl;
// Outputs
wire [7:0]z;
// Instantiate the Unit Under Test (UUT)
barrel_shifter uut (
    .x(x),
    .sel(sel),
    .cntrl(cntrl),
    .z(z)
);
initial begin
    $dumpfile("test.vcd");
    $dumpvars(0,barrel_shifter_tb);
end

initial begin
// Initialize Inputs
#0  x = 8'b10101101;sel=3'b000 ;cntrl=1'b0;
#10 sel=3'b001 ;cntrl=1'b0;
#10 sel=3'b010 ;cntrl=1'b0;
#10 sel=3'b011 ;cntrl=1'b0;
#10 sel=3'b100 ;cntrl=1'b0;
#10 sel=3'b101 ;cntrl=1'b0;
#10 sel=3'b110 ;cntrl=1'b0;
#10 sel=3'b111 ;cntrl=1'b0;
```

```
   #10 sel=3'b000 ;cntrl=1'b1;
   #10 sel=3'b001 ;cntrl=1'b1;
   #10 sel=3'b010 ;cntrl=1'b1;
   #10 sel=3'b011 ;cntrl=1'b1;
   #10 sel=3'b100 ;cntrl=1'b1;
   #10 sel=3'b101 ;cntrl=1'b1;
   #10 sel=3'b110 ;cntrl=1'b1;
   #10 sel=3'b111 ;cntrl=1'b1;
   #10;
   end

   initial begin
       $monitor("t=%3d x=%b,sel=%b,z=%b,cntrl=%b \n",$time,x,sel,z,cntrl);
   end
endmodule
```

Runtime log :-

VCD info: dumpfile test.vcd opened for output.

t= 0 x=10101101,sel=000,z=10101101,cntrl=0

t= 10 x=10101101,sel=001,z=01011010,cntrl=0

t= 20 x=10101101,sel=010,z=10110100,cntrl=0

t= 30 x=10101101,sel=011,z=01101000,cntrl=0

t= 40 x=10101101,sel=100,z=11010000,cntrl=0

t= 50 x=10101101,sel=101,z=10100000,cntrl=0

t= 60 x=10101101,sel=110,z=01000000,cntrl=0

t= 70 x=10101101,sel=111,z=10000000,cntrl=0

t= 80 x=10101101,sel=000,z=10101101,cntrl=1

t= 90 x=10101101,sel=001,z=01010110,cntrl=1

t=100 x=10101101,sel=010,z=00101011,cntrl=1

t=110 x=10101101,sel=011,z=00010101,cntrl=1

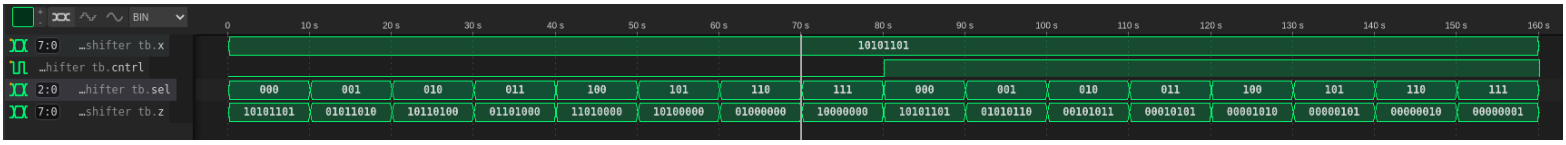
t=120 x=10101101,sel=100,z=00001010,cntrl=1

t=130 x=10101101,sel=101,z=00000101,cntrl=1

t=140 x=10101101,sel=110,z=00000010,cntrl=1

t=150 x=10101101,sel=111,z=00000001,cntrl=1

Results :-



Discussion :-

- Structural implementation of the 2:1 MUX, AND, OR, NOT modules have been used to design it.
- The code has been divided into five main layer functions which have been called sequentially to perform the rotation or shifting as per the select line and control input.
- The output wave forms were generated using GTKWAVE and it has been observed for all 8 values of the select line and 2 values of control (so 16 in total) to show the working of the barrel shifter.
- When control is 0, and select is 001 for example, the input 10101010 shifted to the left with zero padding to give - '01010100.
- When control is 1 and select is 001 for example, the input 10101011 shifted to right with zero padding to give - **0**1010101.
- The test bench was written in the same le which generates the corresponding test.vcd file.

3.Barrel shifter - right shift with 2's complement input

Aim :-

The aim of this experiment is to design a barrel shifter that performs a logical right shift if the input is in 2's complement.

Schematic diagrams and labels :-

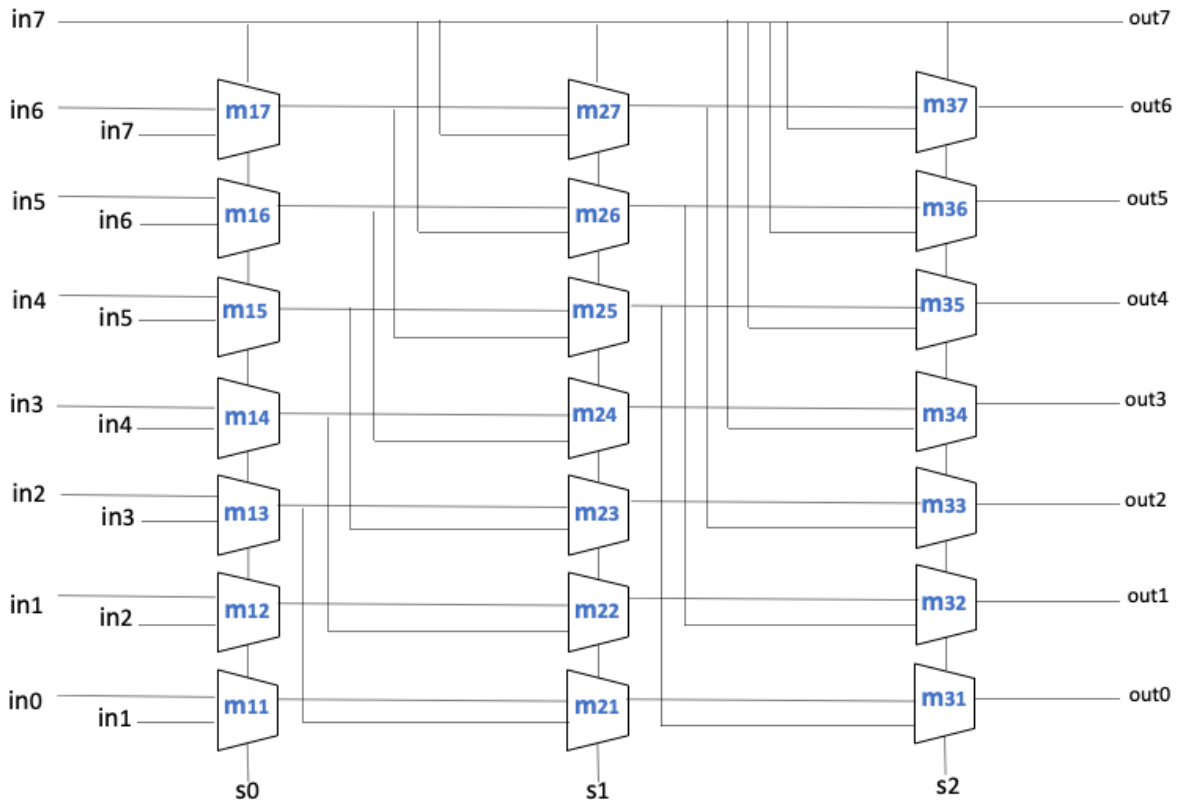


Fig.3:- implementation of Barrel shifter that performs logical right shift when input is 2's complement

Description :-

- Number of 2:1 MUXs used: 21
- 8-bit input: in7 in6 in5 in4 in3 in2 in1 in0
- 3-bit select line: S2 S1 S0
- Output: out7 out6 out5 out4 out3 out2 out 1

Conclusion :-

- In case of a 2's complement input, the MSB indicates the sign of the decimal. For example consider -8. Its 2s complement = 1111 1000.
- Shifting by right indicates division by two. So the expected output should be -4 whose 2s complement is 1111 1100.
- This indicates that for 8-bit numbers with MSB 1, after right shifting it should be padded with one and for 8-bit numbers with MSB 0, it should be padded with 0.
- In the above implementation (Fig3), inp7 is directly sent as out7 indicating that MSB remains constant and the 2:1 MUXs are in such a way they shift each bit to left depending on the 3-bit sel line.
- 00000000 and 11111111 will be the nal outputs for positive and negative numbers respectively. 11111111 in 2s complement in -1. (this is because oor(-1/2) = -1)