VLSI Engineering Lab (Digital)

Experiment 2 – Barrel Shifters and Rotators

Utkarsh Patel (18EC30048)

---

1 Objective

- Implementing 8-bit barrel shifter that performs logical shift left or rotate left operations based on the control input
- Implementing 8-bit barrel shifter that performs logical shift left or logical shift right based on the control input

2 Circuit Diagram

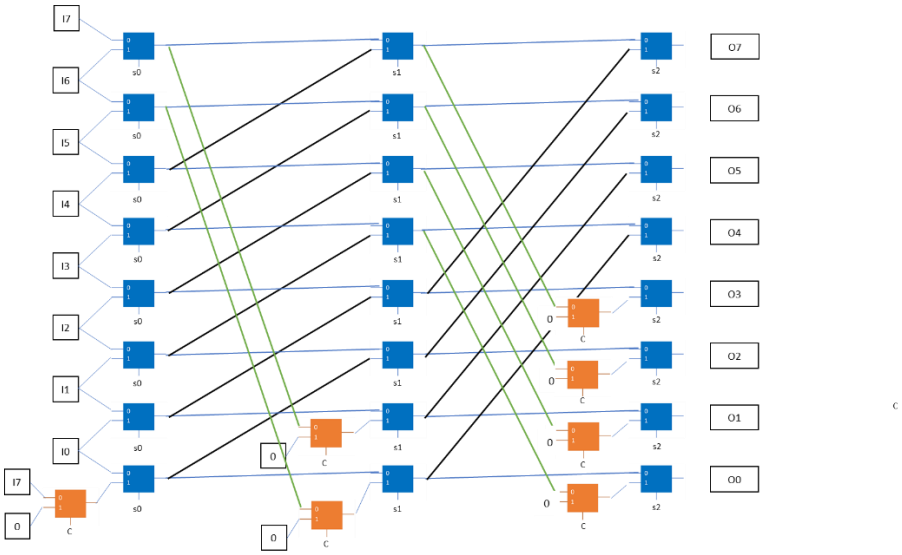

Fig. 1. ***Barrel shifter supporting logic shift left and rotate left operations:*** The circuits takes as input 8-bit integer `I[7:0]`, 3-bit select line `s[2:0]` (that control amount of shift / rotation), 1-bit control line `c` (that controls whether the output will be left shifted or left rotated), and produces the desired output `O[7:0]`. Number of 2:1 MUX used is 24 + 7 = 31.

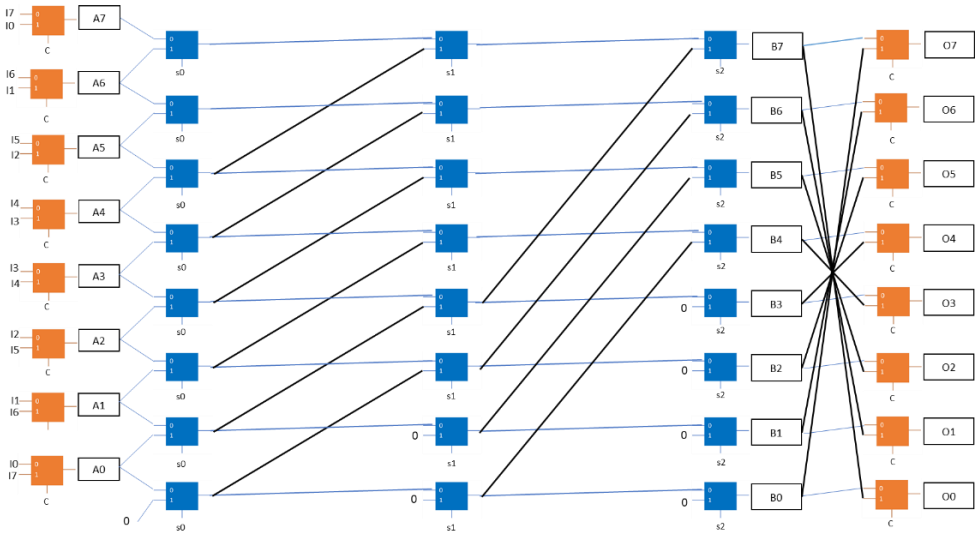

Fig. 2. ***Barrel shifter supporting logic shift left and logic shift right operations:*** The circuits takes as input 8-bit integer `I[7:0]`, 3-bit select line `s[2:0]` (that control amount of shift), 1-bit control line `c` (that controls whether the output will be left shifted or right shifted), and produces the desired output `O[7:0]`. Number of 2:1 MUX used is 24 + 8 + 8 = 40.

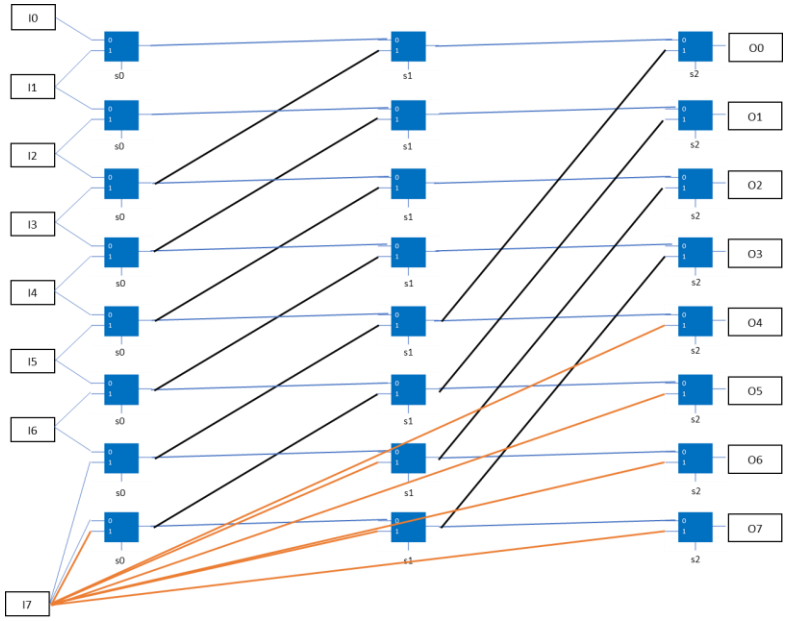

Fig. 3. ***Barrel shifter supporting logic shift right operation for 2's complement input:*** The circuits takes as input 8-bit integer in 2's complement representation `I[7:0]`, 3-bit select line `s[2:0]` (that control amount of shift) and produces the desired output `O[7:0]`. Number of 2:1 MUX used is 24. The idea is if the MSB of the input is '0', then we insert '0' while right-shifting. However, if MSB of input is '1', we insert '1' while right-shifting.

3 Theory

3.1 Logic shift left

Given an 8-bit input sequence `I[7:0]`, we have to left-shift this input sequence by amount given by 3-bit select line `s[2:0]`. For example, if `I[7:0]` = `11001101` and `s[2:0]` = `101`, then the desired output will be `O[7:0]` = `10100000`. This type of circuit can be easily implemented using 24 2:1 MUX. In general, if the input sequence has n bits, and select line has m bits, then optimal number of 2:1 MUX required is nm.

3.2 Logic shift right

Given an 8-bit input sequence `I[7:0]`, we have to right-shift this input sequence by amount given by 3-bit select line `s[2:0]`. For example, if `I[7:0]` = `11001101` and `s[2:0]` = `101`, then the desired output will be `O[7:0]` = `00000110`. This type of circuit can be easily implemented using 24 2:1 MUX. In general, if the input sequence has n bits, and select line has m bits, then optimal number of 2:1 MUX required is nm.

3.3 Rotate left

Given an 8-bit input sequence `I[7:0]`, we have to rotate this input sequence in left direction by amount given by 3-bit select line `s[2:0]`. For example, if `I[7:0]` = `11001101` and `s[2:0]` = `101`, then the desired output will be `O[7:0]` = `10111001`. This type of circuit can be easily implemented using 24 2:1 MUX. In general, if the input sequence has n bits, and select line has m bits, then optimal number of 2:1 MUX required is nm.

3.4 Implementing logic shift left and rotate left in the same circuit optimally

Though the individual circuit of logic shift left and rotate left require 24 2:1 MUX for 8-bit input and 3-bit select lines, it is possible to implement these two circuits simultaneously using just 24 + 7 = 31 2:1 MUX. The design is shown in Fig. 1.

3.5 Implementing logic shift left and logic shift right in the same circuit optimally

Though the individual circuit of logic shift left and rotate left require 24 2:1 MUX for 8-bit input and 3-bit select lines, it is possible to implement these two circuits simultaneously using just 24 + 8 + 8 = 40 2:1 MUX. The design is shown in Fig. 2. The idea is used to use logic shift left circuit, and if we are required to perform right logic shift, the input sequence is reversed by using additional 2:1 MUX. The output sequence has to be reversed in this case as well.

4 Verilog codes

4.1 Barrel shifter supporting logic shift left and rotate left

```
module mux2(input in0,in1 ,input sel,output out);
  wire w1,w2,sel_bar;
  not N1(sel_bar,sel);
  and a1(w1,sel,in1);
  and a2(w2,sel_bar,in0);
  or g1(out,w1,w2);
endmodule
module barrel_shifter(input [7:0]in,output [7:0]out,input [2:0]s,input control);
  wire ground;
  assign ground=0;
  wire [7:0]k;
  wire [7:0]k2;
  wire [7:0]k3;

  mux2 temp1(ground,in[7],control,k3[0]);

  mux2 m1(in[0],k3[0],s[0],k[0]);
  mux2 m2(in[1],in[0],s[0],k[1]);
  mux2 m3(in[2],in[1],s[0],k[2]);
  mux2 m4(in[3],in[2],s[0],k[3]);
  mux2 m5(in[4],in[3],s[0],k[4]);
  mux2 m6(in[5],in[4],s[0],k[5]);
  mux2 m7(in[6],in[5],s[0],k[6]);
  mux2 m8(in[7],in[6],s[0],k[7]);

  mux2 temp2(ground,k[6],control,k3[1]);
  mux2 temp3(ground,k[7],control,k3[2]);

  mux2 m9(k[0],k3[1],s[1],k2[0]);
  mux2 m10(k[1],k3[2],s[1],k2[1]);
  mux2 m11(k[2],k[0],s[1],k2[2]);
  mux2 m12(k[3],k[1],s[1],k2[3]);
  mux2 m13(k[4],k[2],s[1],k2[4]);
  mux2 m14(k[5],k[3],s[1],k2[5]);
  mux2 m15(k[6],k[4],s[1],k2[6]);
  mux2 m16(k[7],k[5],s[1],k2[7]);

  mux2 temp4(ground,k2[4],control,k3[3]);
```

```
  mux2 temp5(ground,k2[5],control,k3[4]);
  mux2 temp6(ground,k2[6],control,k3[5]);
  mux2 temp7(ground,k2[7],control,k3[6]);

  mux2 m17(k2[0],k3[3],s[2],out[0]);
  mux2 m18(k2[1],k3[4],s[2],out[1]);
  mux2 m19(k2[2],k3[5],s[2],out[2]);
  mux2 m20(k2[3],k3[6],s[2],out[3]);
  mux2 m21(k2[4],k2[0],s[2],out[4]);
  mux2 m22(k2[5],k2[1],s[2],out[5]);
  mux2 m23(k2[6],k2[2],s[2],out[6]);
  mux2 m24(k2[7],k2[3],s[2],out[7]);
endmodule

module barrel_shifter_tb;
  reg [7:0]in;
  reg [2:0]s;
  wire [7:0]out;
  reg control;
  barrel_shifter instant2(in,out,s,control);
  initial begin
    $monitor("time=%2d input=%b output=%b s=%b control=%b",$time,in,out,s,control);
    #0;in=8'b10111010;control=1'b0;s=3'b000;
    #2;s=3'b001;
    #2;s=3'b010;
    #2;s=3'b011;
    #2;s=3'b100;
    #2;s=3'b101;
    #2;s=3'b110;
    #2;s=3'b111;

    #2;s=3'b000;control=1'b1;
    #2;s=3'b001;
    #2;s=3'b010;
    #2;s=3'b011;
    #2;s=3'b100;
    #2;s=3'b101;
    #2;s=3'b110;
    #2;s=3'b111;
    #2;$finish;
  end

  initial begin
    $dumpfile("barrel_shifter.vcd");
    $dumpvars;
  end
endmodule
```

4.2 Barrel shifter supporting left / right logical shift

```
module mux2(input in0,in1 ,input sel,output out);
  wire w1,w2,sel_bar;
  not N1(sel_bar,sel);
  and a1(w1,sel,in1);
  and a2(w2,sel_bar,in0);
  or g1(out,w1,w2);
endmodule

module reverse_if_rightshift(input [7:0]in,output [7:0]out,input s);
  mux2 m1(in[0],in[7],s,out[0]);
  mux2 m2(in[1],in[6],s,out[1]);
  mux2 m3(in[2],in[5],s,out[2]);
  mux2 m4(in[3],in[4],s,out[3]);
  mux2 m5(in[4],in[3],s,out[4]);
  mux2 m6(in[5],in[2],s,out[5]);
  mux2 m7(in[6],in[1],s,out[6]);
  mux2 m8(in[7],in[0],s,out[7]);
endmodule

module leftshift(input [7:0]in,output [7:0]out,input [2:0]s);
  wire ground;
  assign ground=0;
  wire [7:0]k;
  wire [7:0]k2;

  mux2 m1(in[0],ground,s[0],k[0]);
  mux2 m2(in[1],in[0],s[0],k[1]);
  mux2 m3(in[2],in[1],s[0],k[2]);
  mux2 m4(in[3],in[2],s[0],k[3]);
  mux2 m5(in[4],in[3],s[0],k[4]);
  mux2 m6(in[5],in[4],s[0],k[5]);
  mux2 m7(in[6],in[5],s[0],k[6]);
  mux2 m8(in[7],in[6],s[0],k[7]);

  mux2 m9(k[0],ground,s[1],k2[0]);
  mux2 m10(k[1],ground,s[1],k2[1]);
  mux2 m11(k[2],k[0],s[1],k2[2]);
  mux2 m12(k[3],k[1],s[1],k2[3]);
  mux2 m13(k[4],k[2],s[1],k2[4]);
  mux2 m14(k[5],k[3],s[1],k2[5]);
  mux2 m15(k[6],k[4],s[1],k2[6]);
  mux2 m16(k[7],k[5],s[1],k2[7]);

  mux2 m17(k2[0],ground,s[2],out[0]);
  mux2 m18(k2[1],ground,s[2],out[1]);
  mux2 m19(k2[2],ground,s[2],out[2]);
  mux2 m20(k2[3],ground,s[2],out[3]);
```

```
  mux2 m21(k2[4],k2[0],s[2],out[4]);
  mux2 m22(k2[5],k2[1],s[2],out[5]);
  mux2 m23(k2[6],k2[2],s[2],out[6]);
  mux2 m24(k2[7],k2[3],s[2],out[7]);
endmodule

module shifter_tb;
  reg [7:0]in;
  wire [7:0]in1;
  wire [7:0]in2;
  reg [2:0]s;
  wire [7:0]out;
  reg control;

  reverse_if_rightshift instant(in,in1,control);
  leftshift instant2(in1,in2,s);
  reverse_if_rightshift returns(in2,out,control);

  initial begin
    $monitor("time=%2d input=%b output=%b s=%b control=%b",$time,in,out,s,control);
    #0;in=8'b10111010;control=1'b0;s=3'b000;
    #2;s=3'b001;
    #2;s=3'b010;
    #2;s=3'b011;
    #2;s=3'b100;
    #2;s=3'b101;
    #2;s=3'b110;
    #2;s=3'b111;

    #2;s=3'b000;control=1'b1;
    #2;s=3'b001;
    #2;s=3'b010;
    #2;s=3'b011;
    #2;s=3'b100;
    #2;s=3'b101;
    #2;s=3'b110;
    #2;s=3'b111;
    #2;$finish;
  end

  initial begin
    $dumpfile("shifter.vcd");
    $dumpvars;
  end
endmodule
```

5 Output Log

5.1 Barrel shifter supporting logic shift left and rotate left

```
C:\iverilog\DVLSI\exp2>iverilog left.v

C:\iverilog\DVLSI\exp2>vvp a.out
VCD info: dumpfile barrel_shifter.vcd opened for output.
time= 0 input=10111010 output=10111010 s=000 control=0
time= 2 input=10111010 output=01110100 s=001 control=0
time= 4 input=10111010 output=11101000 s=010 control=0
time= 6 input=10111010 output=11010000 s=011 control=0
time= 8 input=10111010 output=10100000 s=100 control=0
time=10 input=10111010 output=01000000 s=101 control=0
time=12 input=10111010 output=10000000 s=110 control=0
time=14 input=10111010 output=00000000 s=111 control=0
time=16 input=10111010 output=10111010 s=000 control=1
time=18 input=10111010 output=01110101 s=001 control=1
time=20 input=10111010 output=11101010 s=010 control=1
time=22 input=10111010 output=11010101 s=011 control=1
time=24 input=10111010 output=10101011 s=100 control=1
time=26 input=10111010 output=01010111 s=101 control=1
time=28 input=10111010 output=10101110 s=110 control=1
time=30 input=10111010 output=01011101 s=111 control=1
left.v:86: $finish called at 32 (1s)
```

5.2 Barrel shifter supporting left / right logical shift

```
C:\iverilog\DVLSI\exp2>iverilog left_right.v

C:\iverilog\DVLSI\exp2>vvp a.out
VCD info: dumpfile shifter.vcd opened for output.
time= 0 input=10111010 output=10111010 s=000 control=0
time= 2 input=10111010 output=01110100 s=001 control=0
time= 4 input=10111010 output=11101000 s=010 control=0
time= 6 input=10111010 output=11010000 s=011 control=0
time= 8 input=10111010 output=10100000 s=100 control=0
time=10 input=10111010 output=01000000 s=101 control=0
time=12 input=10111010 output=10000000 s=110 control=0
time=14 input=10111010 output=00000000 s=111 control=0
time=16 input=10111010 output=10111010 s=000 control=1
time=18 input=10111010 output=01011101 s=001 control=1
time=20 input=10111010 output=00101110 s=010 control=1
time=22 input=10111010 output=00010111 s=011 control=1
time=24 input=10111010 output=00001011 s=100 control=1
time=26 input=10111010 output=00000101 s=101 control=1
time=28 input=10111010 output=00000010 s=110 control=1
time=30 input=10111010 output=00000001 s=111 control=1
left_right.v:85: $finish called at 32 (1s)
```

6 Waveforms

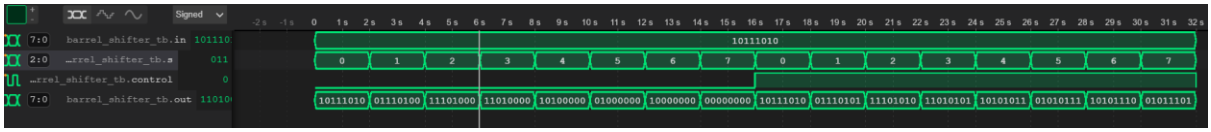

Fig. 4. ***Barrel shifter supporting logic shift left and rotate left operations:*** The circuits takes as input 8-bit integer `I[7:0]`, 3-bit select line `s[2:0]` (that control amount of shift / rotation), 1-bit control line `c` (that controls whether the output will be left shifted or left rotated), and produces the desired output `O[7:0]`

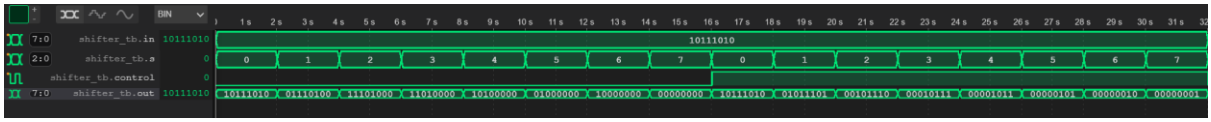

Fig. 5. ***Barrel shifter supporting logic shift left and logic shift right operations:*** The circuits takes as input 8-bit integer `I[7:0]`, 3-bit select line `s[2:0]` (that control amount of shift), 1-bit control line `c` (that controls whether the output will be left shifted or right shifted), and produces the desired output `O[7:0]`

7 Conclusion

- In this experiment, 8-bit barrel shifter supporting logic shift left and rotate left, 8-bit barrel shifter supporting logic shift left and logic shift right and barrel shifter supporting logic shift right for 8-bit 2's complement integer are designed. [See Fig. 1, 2 and 3]

- ***Logic shift left:*** Given an 8-bit input sequence `I[7:0]`, we have to left-shift this input sequence by amount given by 3-bit select line `s[2:0]`. For example, if `I[7:0]` = `11001101` and `s[2:0]` = `101`, then the desired output will be `O[7:0]` = `10100000`. This type of circuit can be easily implemented using 24 2:1 MUX.

- ***Logic shift right:*** Given an 8-bit input sequence `I[7:0]`, we have to right-shift this input sequence by amount given by 3-bit select line `s[2:0]`. For example, if `I[7:0]` = `11001101` and `s[2:0]` = `101`, then the desired output will be `O[7:0]` = `00000110`. This type of circuit can be easily implemented using 24 2:1 MUX.

- ***Rotate left:*** Given an 8-bit input sequence `I[7:0]`, we have to rotate this input sequence in left direction by amount given by 3-bit select line `s[2:0]`. For example, if `I[7:0]` = `11001101` and `s[2:0]` = `101`, then the desired output will be `O[7:0]` = `10111001`. This type of circuit can be easily implemented using 24 2:1 MUX.

- ***Logic shift right (for 2's complement input):*** Given an 8-bit input sequence in 2's complement representation `I[7:0]`, 3-bit select line `s[2:0]` (that control amount of shift) and produces the desired output `O[7:0]`. Number of 2:1 MUX used is 24. The idea is if the MSB of the input is '0', i.e., if the input is a non-negative integer, then we insert '0' while right-shifting. However, if MSB of input is '1', i.e., if the input is a negative integer, we insert '1' while right-shifting.

- Logic shift left and rotate left operations can be implemented in the same circuit using 24 + 7 = 31 2:1 MUX. The idea is to use logic shift left circuit, and add additional (7) 2:1 MUX to decide whether we insert '0' or the LSBs while left-shifting.

- Logic shift left and logic shift right operations can be implemented in the same circuit using 24 + 8 + 8 = 40 2:1 MUX. The idea is to use logic shift left circuit, and for logic shift right operation, we reverse the input sequence, pass it through logic shift left circuit, and again reverse the output of the circuit to obtain right-shifted version of the input.