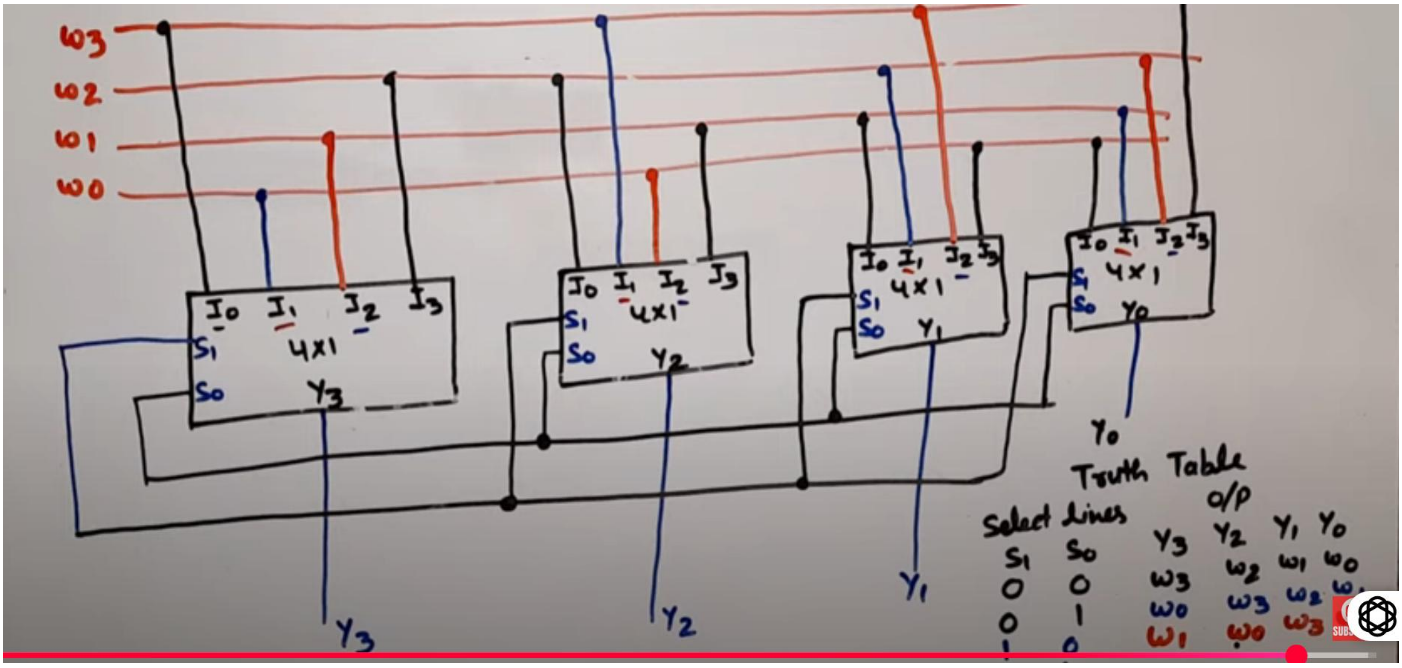
11. 4-bit BARREL SHIFTER (only left rotate)

- they are digital circuits (not sequential i.e. no flip flops used)
- we use MUX (combinational)
- to shift/rotate bits
- both left/right



CODE:

```
`timescale 1ns / 1ps

module barrel_shifter(
   input [3:0] w,
   input [1:0] s,
   output [3:0] y
   );
mux4x1 m0(w[3],w[0],w[1],w[2], s, y[3]);
mux4x1 m1(w[2],w[3],w[0],w[1], s, y[2]);
mux4x1 m2(w[1],w[2],w[3],w[0], s, y[1]);
mux4x1 m3(w[0],w[1],w[2],w[3], s, y[0]);

endmodule

module mux4x1(
input i0,i1,i2,i3,
input [1:0]s,
output reg y
);
always@(*)begin
```

```
    case(s)
        2'b00: y = i0;
        2'b01: y = i1;
        2'b10: y = i2;
        2'b11: y = i3;
        default: y = 4'b0;
    endcase
end
endmodule
```

//testbench

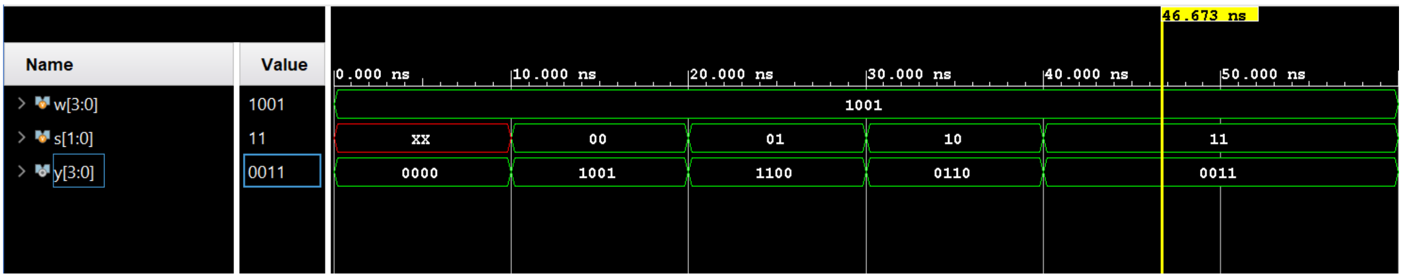
```
`timescale 1ns / 1ps

module tb_bs;
    reg [3:0] w;
    reg [1:0] s;
    wire [3:0] y;

    barrel_shifter bs1(w,s,y);

    initial begin
        w = 1001;
        #10
        s = 00;
        #10
        s = 01;
        #10
        s=10;
        #10
        s=11;
        #20
        $stop;
    end
endmodule
```

Simulation Result:



UNIVERSAL BARREL SHIFTER

This does both logical and rotational shifts using mode. Left/right is decided by dir.

We also select shift amount using s.

```
`timescale 1ns / 1ps

module barrel_shifter_universal(
   input  [3:0] w,     // data in
   input  [1:0] s,     // shift amount
   input        mode,  // 0 = logical, 1 = rotate
   input        dir,   // 0 = left, 1 = right
   output [3:0] y      // shifted output
);
   wire [3:0] in0, in1, in2, in3;

   // Generate mux inputs based on mode & direction
   assign in0 = (dir==0) ?
             { w[3], w[0], w[1], w[2] } :  // Rotate left by 1
             { w[1], w[2], w[3], w[0] };   // Rotate right by 1

   assign in1 = (dir==0) ?
             { w[2], w[3], w[0], w[1] } :  // Rotate left by 2
             { w[2], w[3], w[0], w[1] };   // Rotate right by 2 (same pattern for 4 bits)

   assign in2 = (dir==0) ?
             { w[1], w[2], w[3], w[0] } :  // Rotate left by 3
             { w[3], w[0], w[1], w[2] };   // Rotate right by 3

   // Select inputs for logical shift mode
   wire [3:0] log_shift [3:0];
   assign log_shift[0] = (dir==0) ? { w[2:0], 1'b0 } : { 1'b0, w[3:1] }; // shift by 1
   assign log_shift[1] = (dir==0) ? { w[1:0], 2'b00 } : { 2'b00, w[3:2] }; // shift by 2
   assign log_shift[2] = (dir==0) ? { w[0], 3'b000 } : { 3'b000, w[3] };   // shift by 3
```

```
    // Final selection based on mode
    wire [3:0] sel0 = (mode) ? in0 : log_shift[0];
    wire [3:0] sel1 = (mode) ? in1 : log_shift[1];
    wire [3:0] sel2 = (mode) ? in2 : log_shift[2];

    // Multiplex based on shift amount
    assign y = (s == 2'b00) ? w :
               (s == 2'b01) ? sel0 :
               (s == 2'b10) ? sel1 :
                              sel2;

endmodule

`timescale 1ns / 1ps

module tb_barrel_shifter_universal;

    reg  [3:0] w;
    reg  [1:0] s;
    reg        mode;  // 0 = logical, 1 = rotate
    reg        dir;   // 0 = left, 1 = right
    wire [3:0] y;

    // Instantiate DUT
    barrel_shifter_universal uut (
        .w(w),
        .s(s),
        .mode(mode),
        .dir(dir),
        .y(y)
    );

    integer i, j, k;
```

```
  initial begin
    // Test with an easy-to-see pattern
    w = 4'b1011;

    // Loop through all modes (0 = logical, 1 = rotate)
    for (i = 0; i < 2; i = i + 1) begin
      mode = i;

      // Loop through both directions (0 = left, 1 = right)
      for (j = 0; j < 2; j = j + 1) begin
        dir = j;

        // Loop through shift amounts (0 to 3)
        for (k = 0; k < 4; k = k + 1) begin
          s = k;
          #10; // wait 10ns between each case
        end
      end
    end

    $stop;
  end

endmodule
```