Ryan Hunter
Kendall Won

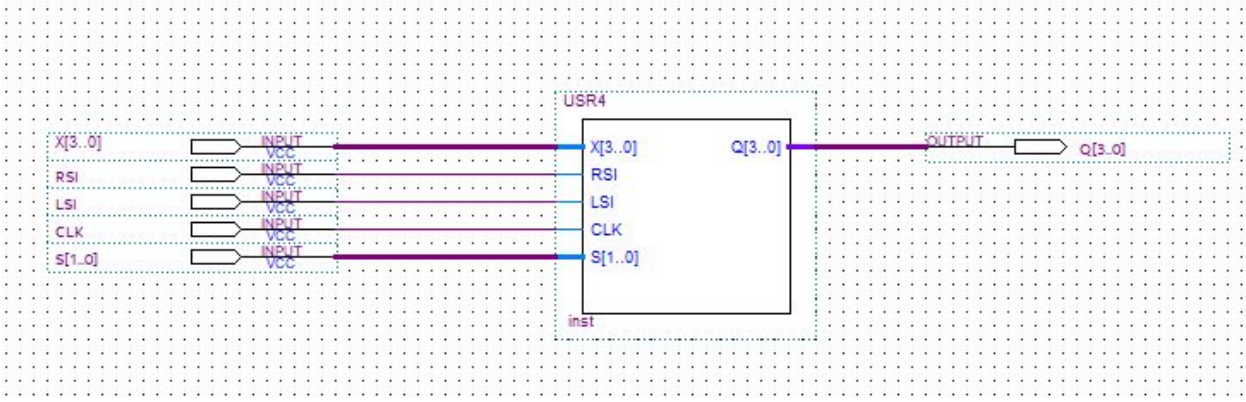
**COEN 21 Lab 8**

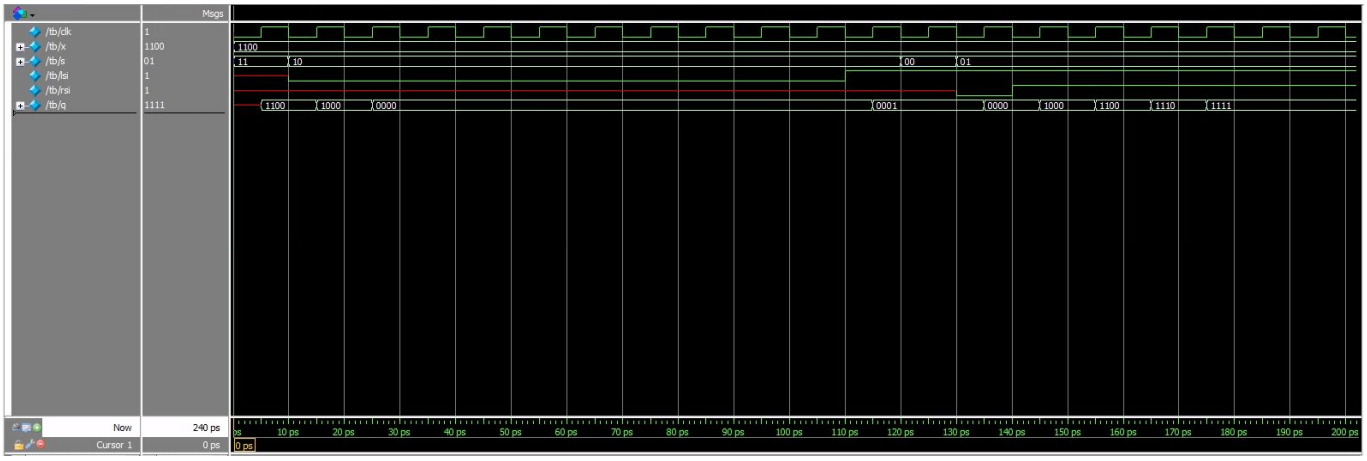
## Part 1

```
1
2   module USR4 (X,RSI,LSI,CLK,S,Q);
3       input [3:0] X;
4       input RSI, LSI, CLK;
5       input [1:0]S;
6       output reg [3:0] Q;
7       always @ (posedge CLK)
8       begin
9           case(S)
10          2'b00: Q<=Q;
11          2'b01: Q<={RSI, Q[3:1]};
12          2'b10: Q<={Q[2:0], LSI};
13          2'b11: Q<=X;
14          endcase
15      end
16  endmodule
```

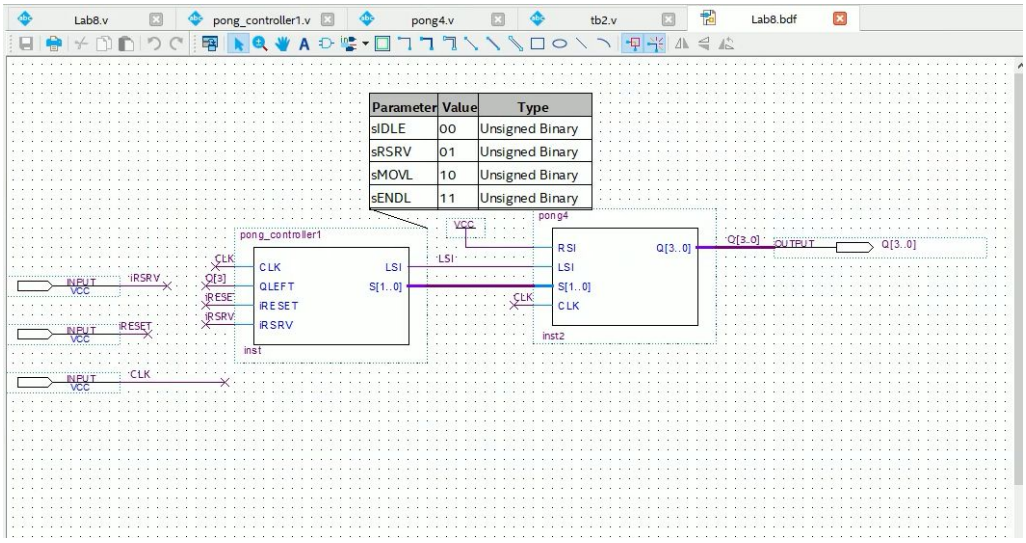
```
1
2   module tb();
3
4     reg clk;
5     reg [3:0]x;
6     reg [1:0]s;
7     reg lsi, rsi;
8     wire [3:0]q;
9
10    Lab81 dut(.CLK(clk), .X(x), .S(s), .RSI(rsi), .LSI(lsi), .Q(q));
11
12
13    initial begin
14      clk = 0;
15
16    forever #5 clk = ~clk;
17    end
18
19  initial begin
20
21
22  x = 12;
23  s = 3;
24
25  #10;
26
27  s = 2;
28  lsi = 0;
29
30
31  #100;
32
33  lsi = 1;
34
35
36  #10;
37
38  s = 0;
39
40  #10
41
42  s = 1;
43  rsi = 0;
44
45
46  #10
47
48  rsi = 1;
49
50  #100
51  $finish;
52  end
53  endmodule
54
```

```
19
20  module Lab8(
21      RSI,
22      LSI,
23      CLK,
24      S,
25      X,
26      Q
27  );
28
29
30    input wire  RSI;
31    input wire  LSI;
32    input wire  CLK;
33    input wire  [1:0] S;
34    input wire  [3:0] X;
35    output wire [3:0] Q;
36
37
38
39
40
41
42  USR4  b2v_inst(
43      .RSI(RSI),
44      .LSI(LSI),
45      .CLK(CLK),
46      .S(S),
47      .X(X),
48      .Q(Q));
49
50
51    endmodule
52
```



## Part 2




pong_controller1.v | pong4.v | tb2.v | Lab8.v

```
1  module pong_controller1(input CLK, QLEFT, iRESET, iRSRV, output reg LSI, output reg [1:0] S);
2     reg [1:0] currState, nextState;
3     parameter sIDLE = 2'b00, sRSRV = 2'b01, sMOVL = 2'b10, sENDL = 2'b11;
4
5     always @(posedge CLK) currState <= nextState;
6
7     always @(*) begin
8        nextState = sIDLE;
9        case(currState)
10          sIDLE: begin
11                   if(iRSRV) nextState = sRSRV;
12                   {LSI, S} = 3'b011;
13             end
14          sRSRV: begin
15                   if(!iRESET) nextState = sMOVL;
16                   {LSI, S} = 3'b110;
17
18             end
19          sMOVL: begin
20                   if(iRESET) nextState = sIDLE;
21                   else if(QLEFT) nextState = sENDL;
22                   else nextState = sMOVL;
23                   {LSI, S} = 3'b010;
24             end
25          sENDL: begin
26                   nextState = sIDLE;
27                   {LSI, S} = 3'b000;
28             end
29          default: begin
30                   nextState = sIDLE;
31                   {LSI, S} = 3'b000;
32             end
33          endcase
34       end
35 endmodule
36
```

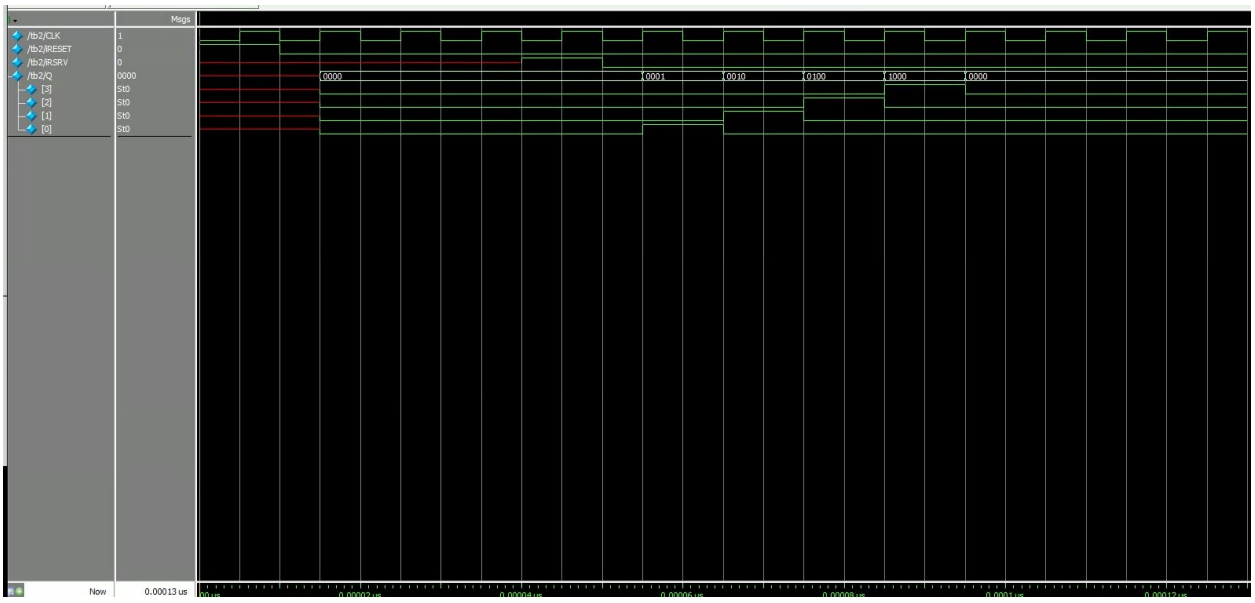
pong_controller1.v | pong4.v | tb2.v | Lab8.v

```
1  module pong4(RSI,LSI,S,CLK,Q);
2     input [1:0]S;
3     input RSI,LSI,CLK;
4     output reg [3:0]Q;
5
6     always@(posedge CLK)
7     begin
8        case(S)
9        2'b00: Q<=Q;
10       2'b01: Q<={RSI, Q[3:1]};
11       2'b10: Q<={Q[2:0], LSI};
12       2'b11: Q<=4'b0000;
13       endcase
14    end
15 endmodule
```

```v
module Lab8(
    CLK,
    iRESET,
    iRSRV,
    Q
);

input wire  CLK;
input wire  iRESET;
input wire  iRSRV;
output wire [3:0] Q;

wire  LSI;
wire  [3:0] Q_ALTERA_SYNTHESIZED;
wire  SYNTHESIZED_WIRE_0;
wire  [1:0] SYNTHESIZED_WIRE_1;

assign   SYNTHESIZED_WIRE_0 = 1;



pong4 b2v_inst(
    .RSI(SYNTHESIZED_WIRE_0),
    .LSI(LSI),
    .CLK(CLK),
    .S(SYNTHESIZED_WIRE_1),
    .Q(Q_ALTERA_SYNTHESIZED));

pong_controller1  b2v_inst2(
    .CLK(CLK),
    .QLEFT(Q_ALTERA_SYNTHESIZED[3]),
    .iRESET(iRESET),
    .iRSRV(iRSRV),
    .LSI(LSI),
    .S(SYNTHESIZED_WIRE_1));
    defparam b2v_inst2.sENDL = 2'b11;
    defparam b2v_inst2.sIDLE = 2'b00;
    defparam b2v_inst2.sMOVL = 2'b10;
    defparam b2v_inst2.sRSRV = 2'b01;


assign   Q = Q_ALTERA_SYNTHESIZED;

endmodule
```

pong_controller1.v | pong4.v | tb2.v | Lab8.v

```v
module tb2();
    reg CLK, iRESET, iRSRV;
    wire[3:0] Q;
    Lab8 dut(.CLK(CLK), .iRESET(iRESET), .iRSRV(iRSRV), .Q(Q));
    initial begin
        CLK = 0;
        forever #5 CLK = ~CLK;
    end
    initial begin
    iRESET = 1;
    #10
    iRESET = 0;
    #30
    iRSRV = 1;
    #10
    iRSRV = 0;
    #80
    $finish;
    end
endmodule
```




**2. Describe the most challenging part of this lab.**

The most challenging part of the lab was getting the right outputs in the simulation. We had problems with getting both of our test benches to work properly. One mistake that we had was that we forgot to change "top" in the test bench files to the name of the Verilog file that was created from our bdf file and not the USR4 Verilog.

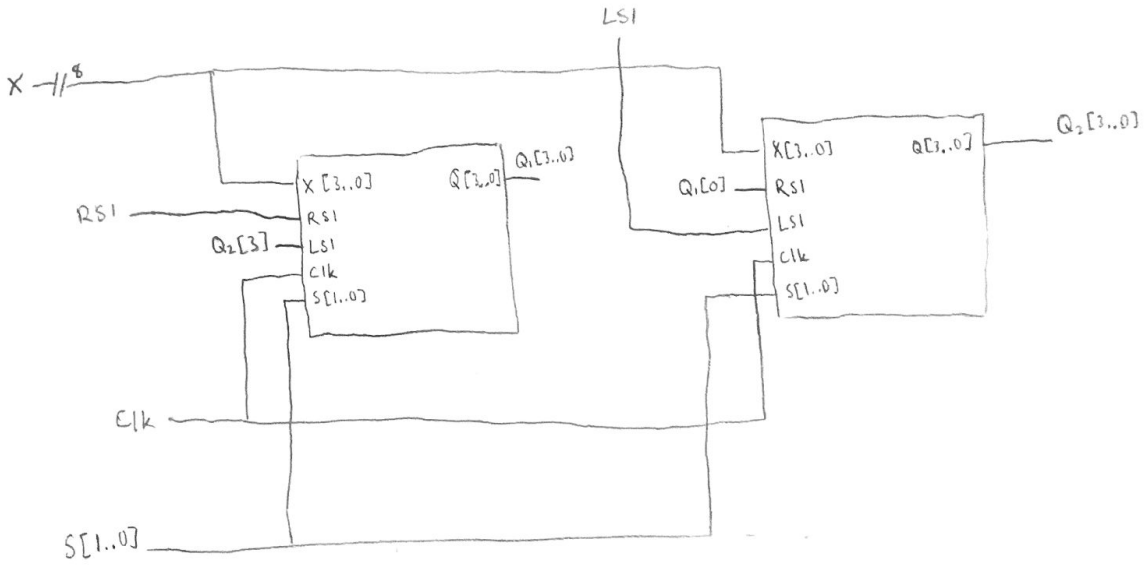
**3. For lab part 1, if you loaded the register with 1101, what value would that represent if you interpreted it as an unsigned integer? With RSI=0, if you shifted to the right what binary pattern would you see and how would it be interpreted as an unsigned number? With RSI=0, if you shifted to the right a second time what binary pattern would you see and how would it be interpreted as an unsigned number? Explain how right shifting is related to dividing by 2.**

It would represent 13 when interpreted as unsigned. The subsequent shift would have a value of 6 and if done again you would have 3. Each binary position represents a power of two. When you shift to the right, you are essentially dividing each bit's decimal representation by two. For 1101, for example, you get 8/2 + 4/2 = 4 + 2 which equals 6.

**4. In the previous question, for unsigned interpretation of the values, RSI was set to zero. To divide by 2 using a right shift for signed integer interpretation using 2's complement representation, what should RSI be? Why?**

RSI should be set to the value of the most significant bit of the binary number being shifted. If you take 1100 which is interpreted as -8 + 4 = -4, shifting to the right and replacing with 0 would result in a positive value of 6, whereas replacing with one gives you -8 + 6 = -2. If you always replaced it with 1, however, then positive values, such as 0110 would go from a value of 6 to a value of -5. By replacing with the value of the most significant bit 0, we would get 3, the correct result.

**5. How would two 4-bit universal shift registers be connected to form an 8-bit universal shift register? Show a schematic in which each 4-bit register is shown as a block component with inputs and outputs. Do not show the internal components and connections of the 4-bit universal shift register.**



**6. For lab part2, if the controller is in the idle state and the right serve input is high during an active clock edge, what is the next state? For the next 6 active clock edges, specify the state that the controller is in after each clock edge.**

According to the provided State Table the next states in order would be

1) sRightPlayerServes
2) sMoveLeft
3) sMoveLeft
4) sMoveLeft
5) sEndLeft
6) sIDLE

**7. If by mistake QLEFT were connected to Q[0] instead of Q[3], how would your answer to the previous question change?**

According to the provided State Table the next states in order would be

1) sIDLE to sRightPlayerServes
2) sMoveLeft
3) sEndLeft
4) sIDLE
5) sIDLE
6) sIDLE

**8. Show how you would modify your state transition diagram to allow both players to serve. How many additional states would you need? What additional inputs and outputs would be needed?**

There would be three additional states that we would need. The three states are sLeftPlayerServes, sMoveRight and sEndRight. The additional inputs would be QRIGHT and iLSRV, where iLSRV represents the left player hitting the ball and QRIGHT is connected to the Q[0] output. There would be one additional output which would be RSI, which indicates that the ball is in play for the right player to hit.