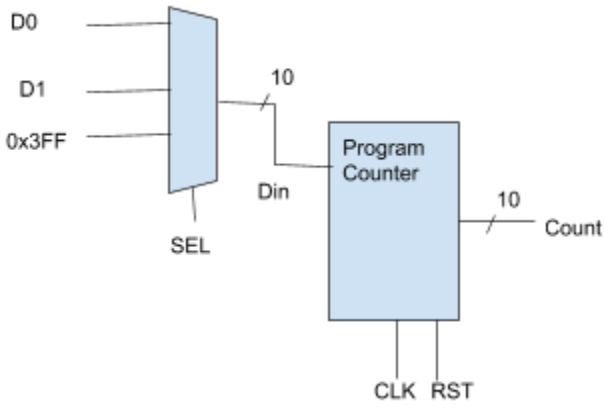
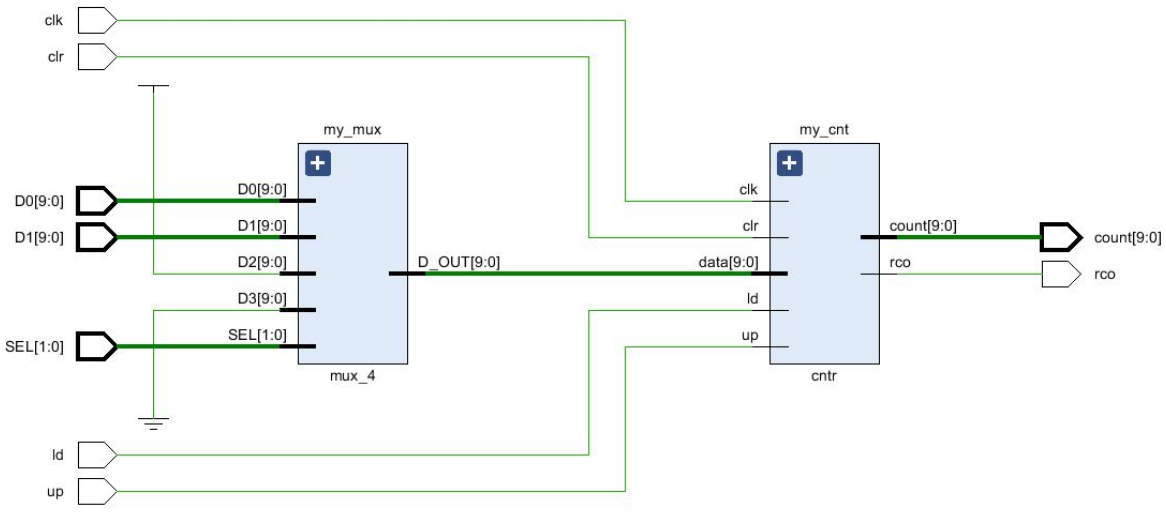
Black Box Diagram :

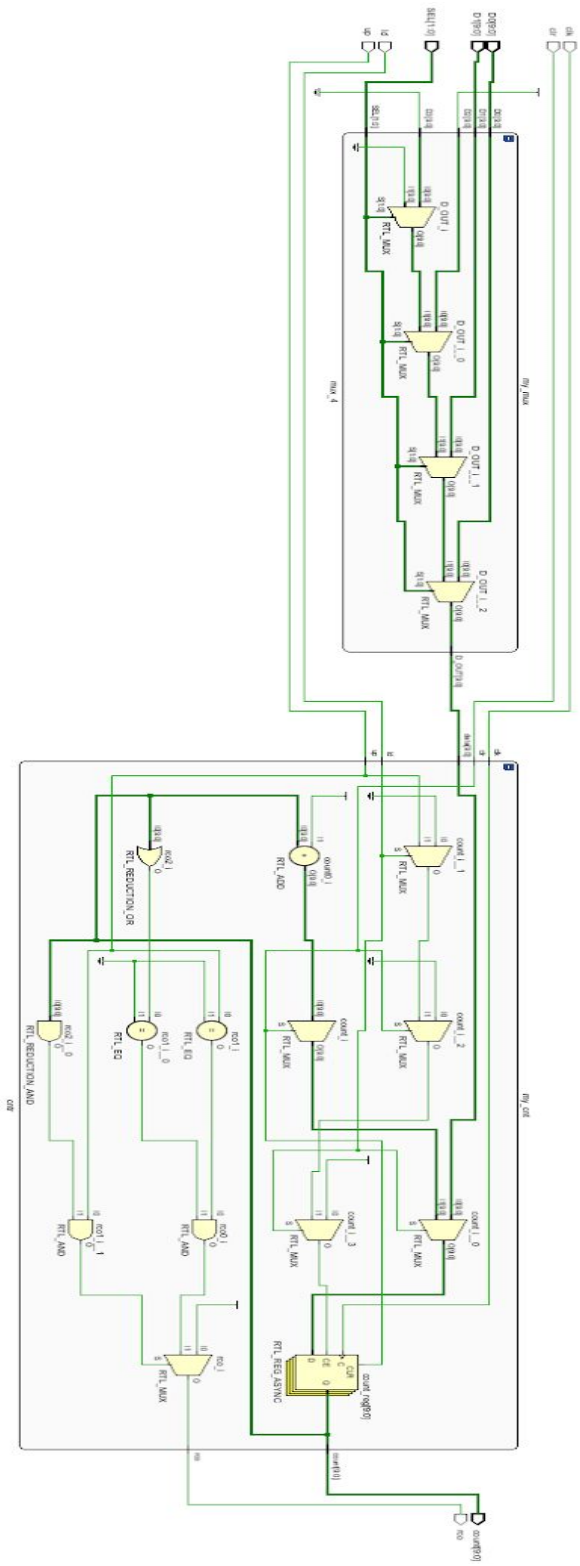


Behavior Design :

In this RAT assignment, we designed a program counter with a multiplex input. We coded this through modules in system verilog. The multiplexor decided on the input from either the stack, immediate input, or an interrupt.

Structure Design :



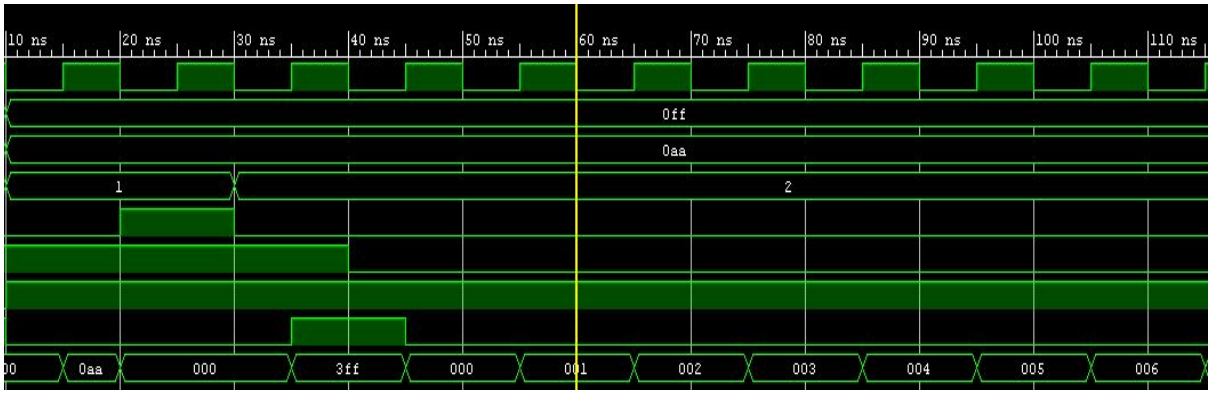


Verification :

Values:

Name	Value
CLK	0
D0[9:0]	0ff
D1[9:0]	0aa
SEL[1:0]	2
clr	0
ld	0
up	1
rco	0
count[9:0]	001

Test Bench Output



Values	Output
10ns-20ns **SEL = 2'b01;** **D0 = 'hFF;** **D1 = 'hAA;** **clr = 1'b0;** **ld = 1'b1;** **up = 1'b1;**	**0xAA loaded into counter from D1**
20ns-30ns **SEL = 2'b01;** **D0 = 'hFF;** **D1 = 'hAA;** **clr = 1'b1;** **ld = 1'b1;** **up = 1'b1;**	**It can be seen in the Figure above that the counter is cleared.**
30ns-110ns **SEL = 2'b10;** **D0 = 'hFF;** **D1 = 'hAA;** **clr = 1'b0;** **ld = 1'b1;** **up = 1'b1;**	**Here the interrupt is being loaded and can be seen counting up from 1 due to the fact that 0x3FF is the max address.**

The test cases shown above in the table were chosen to represent the individual inputs from the multiplexor. Through these test cases we could see that the functionality works. See the table above for more about each test case.

System Verilog Source Code :

```
module PC(D0, D1, clk, ld, up, SEL, clr, rco, count);

    input [9:0] D0, D1;
    input [1:0] SEL;
    input clk, clr, ld, up;

    output rco;
    output [9:0] count;

    wire [9:0] data;

    mux_4 my_mux(
        .SEL    (SEL),
        .D0     (D0),
        .D1     (D1),
        .D2     (10'b1111111111),
        .D3     (10'b0000000000),
        .D_OUT  (data)
    );

    cntr my_cnt(
        .clk    (clk),
        .clr    (clr),
        .up     (up),
        .ld     (ld),
        .data   (data),
        .count  (count),
        .rco    (rco)
    );


endmodule

module cntr(clk, clr, up, ld, data, count, rco);
    input  clk, clr, up, ld;
    input  [9:0] data;
    output   reg [9:0] count;
    output   reg rco;
```

```
    always_ff @(posedge clr, posedge clk)
    begin
        if (clr == 1'b1)        // asynch reset
           count <= 0;
        else if (ld == 1'b1)    // load new value
           count <= data;
        else if (up == 1'b1)    // count up (increment)
           //count <= count + 1'b1;
           count++;
    end


    //- handles the RCO, which is direction dependent
    always_ff @(count, up)
    begin
       if ( up == 1 && &count == 1'b1)
          rco = 1'b1;
       else if (up == 0 && |count == 1'b0)
          rco = 1'b1;
       else
          rco = 1'b0;
    end

endmodule

 module mux_4(SEL, D0, D1, D2, D3, D_OUT);
       input  [1:0] SEL;
       input  [9:0] D0, D1, D2, D3;
       output reg [9:0] D_OUT;

       always @(SEL, D0, D1, D2, D3)
       begin
          if       (SEL == 0)  D_OUT = D0;
          else if (SEL == 1)  D_OUT = D1;
          else if (SEL == 2)  D_OUT = D2;
          else if (SEL == 3)  D_OUT = D3;
          else                D_OUT = 0;
       end

endmodule
```