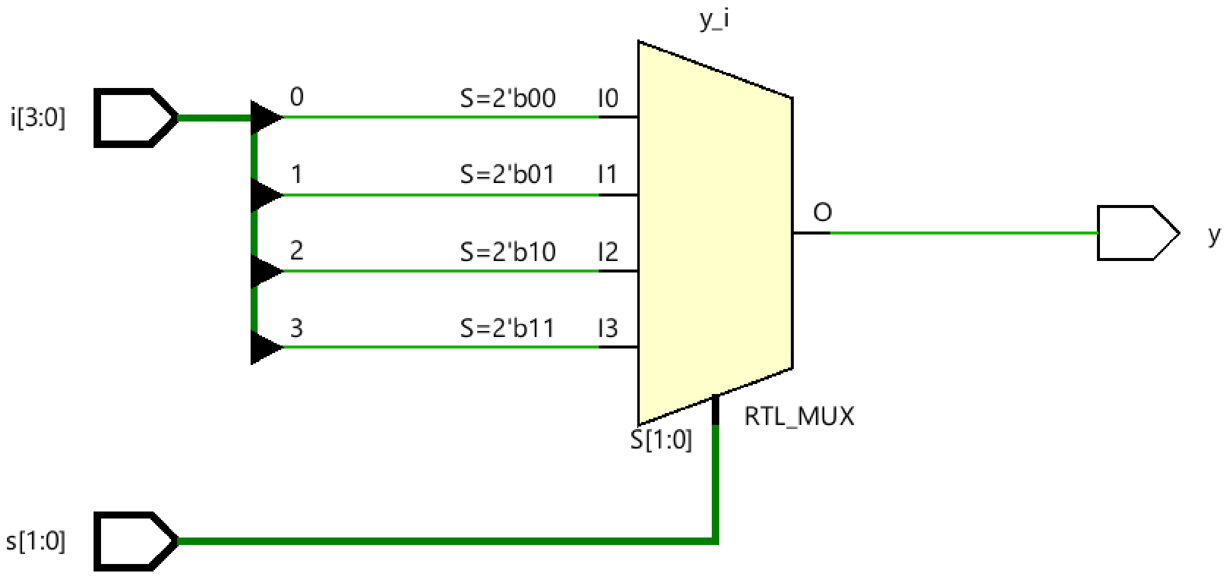
A Sanath Hoysala - PES1UG21EE002 - Section A

1 4x1 Multiplexer

```
module MUX4x1(y,i,s);
output reg y;
input [3:0]i; input [1:0]s;
always@(i,s)
case(s)
2'b00 : y<=i[0];
2'b01 : y<=i[1];
2'b10 : y<=i[2];
2'b11 : y<=i[3];
endcase
endmodule
```

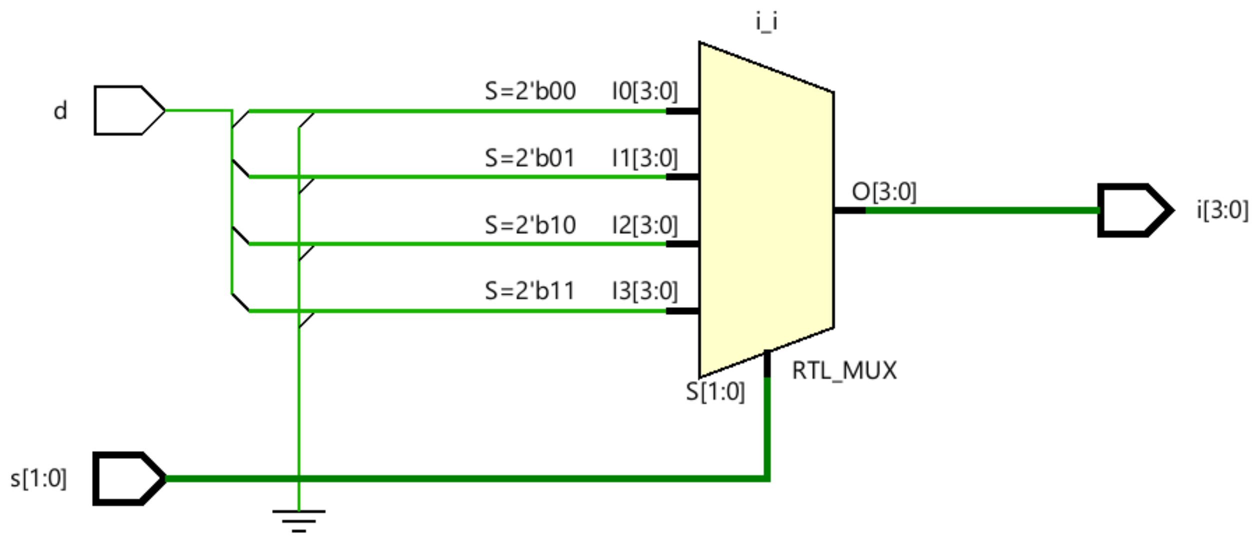
RTL Analysis :-



2 1x4 Demultiplexer

```
module DEMUX1x4(i,d,s);
output reg [3:0]i;
input d; input [1:0]s;
always@(d,s)
case(s)
    2'b00: begin i[0]<=d; i[1]<=0; i[2]<=0; i[3]<=0; end
    2'b01: begin i[0]<=0; i[1]<=d; i[2]<=0; i[3]<=0; end
    2'b10: begin i[0]<=0; i[1]<=0; i[2]<=d; i[3]<=0; end
    2'b11: begin i[0]<=0; i[1]<=0; i[2]<=0; i[3]<=d; end
endcase
endmodule
```

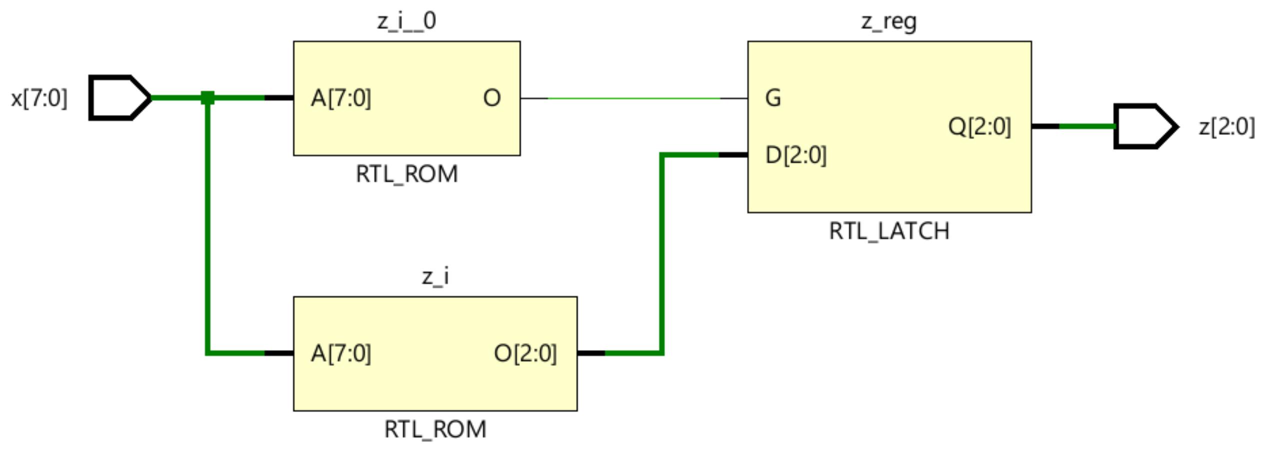
RTL Analysis :-



3 Encoder 8x3

```
module ENCD8x3(z,x);
output reg [2:0]z;
input [7:0]x;
always@(z,x)
begin
    case(x)
        8'b00000001: begin z[2]<=0; z[1]<=0; z[0]<=0; end
        8'b00000010: begin z[2]<=0; z[1]<=0; z[0]<=1; end
        8'b00000100: begin z[2]<=0; z[1]<=1; z[0]<=0; end
        8'b00001000: begin z[2]<=0; z[1]<=1; z[0]<=1; end
        8'b00010000: begin z[2]<=1; z[1]<=0; z[0]<=0; end
        8'b00100000: begin z[2]<=1; z[1]<=0; z[0]<=1; end
        8'b01000000: begin z[2]<=1; z[1]<=1; z[0]<=0; end
        8'b10000000: begin z[2]<=1; z[1]<=1; z[0]<=1; end
    endcase
end
endmodule
```

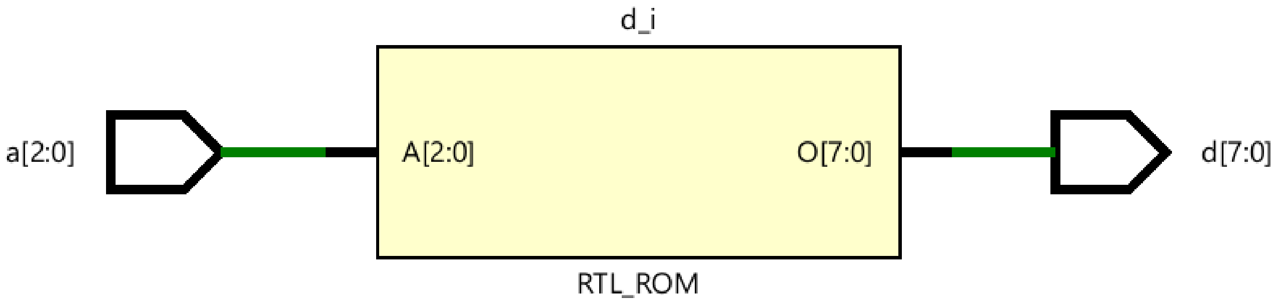
RTL Analysis :-



4 Decoder 3x8

```
module DCDR3x8(d,a);
output reg [7:0]d;
input [2:0]a;
always@(a)
begin
case(a)
3'b000: begin d[0]<=1 ;d[1]<=0 ;d[2]<=0 ;d[3]<=0 ;d[4]<=0 ;d[5]<=0 ;
        d[6]<=0 ;d[7]<=0 ;end
3'b001: begin d[0]<=0 ;d[1]<=1 ;d[2]<=0 ;d[3]<=0 ;d[4]<=0 ;d[5]<=0 ;
        d[6]<=0 ;d[7]<=0 ;end
3'b010: begin d[0]<=0 ;d[1]<=0 ;d[2]<=1 ;d[3]<=0 ;d[4]<=0 ;d[5]<=0 ;
        d[6]<=0 ;d[7]<=0 ;end
3'b011: begin d[0]<=0 ;d[1]<=0 ;d[2]<=0 ;d[3]<=1 ;d[4]<=0 ;d[5]<=0 ;
        d[6]<=0 ;d[7]<=0 ;end
3'b100: begin d[0]<=0 ;d[1]<=0 ;d[2]<=0 ;d[3]<=0 ;d[4]<=1 ;d[5]<=0 ;
        d[6]<=0 ;d[7]<=0 ;end
3'b101: begin d[0]<=0 ;d[1]<=0 ;d[2]<=0 ;d[3]<=0 ;d[4]<=0 ;d[5]<=1 ;
        d[6]<=0 ;d[7]<=0 ;end
3'b110: begin d[0]<=0 ;d[1]<=0 ;d[2]<=0 ;d[3]<=0 ;d[4]<=0 ;d[5]<=0 ;
        d[6]<=1 ;d[7]<=0 ;end
3'b111: begin d[0]<=0 ;d[1]<=0 ;d[2]<=0 ;d[3]<=0 ;d[4]<=0 ;d[5]<=0 ;
        d[6]<=0 ;d[7]<=1 ;end
    endcase
end
endmodule
```

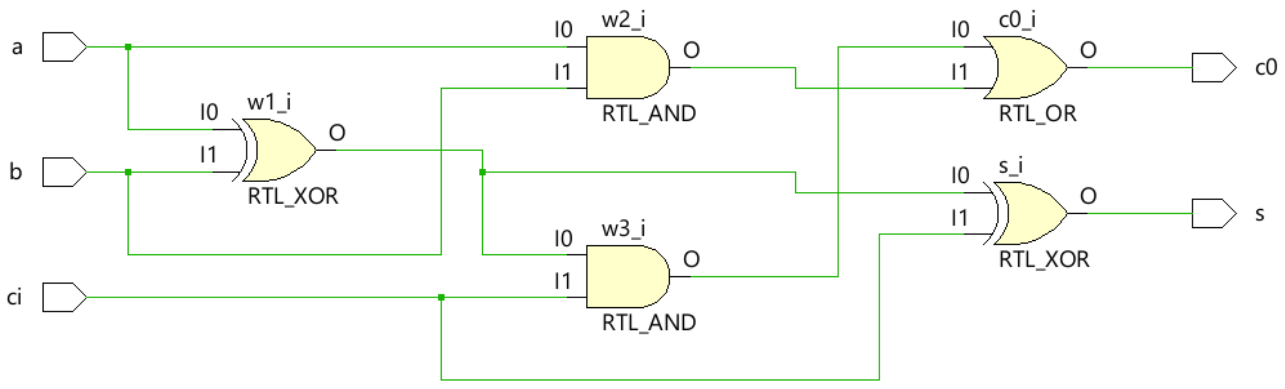
RTL Analysis :-



5 Full Adder using structural modelling

```
module FA_Struct(s,c0,a,b,ci);
output s,c0;
input a,b,ci;
wire w1,w2,w3;
    xor(w1,a,b);
    and(w2,a,b);
    xor(s,w1,ci);
    and(w3,w1,ci);
    or(c0,w3,w2);
endmodule
```

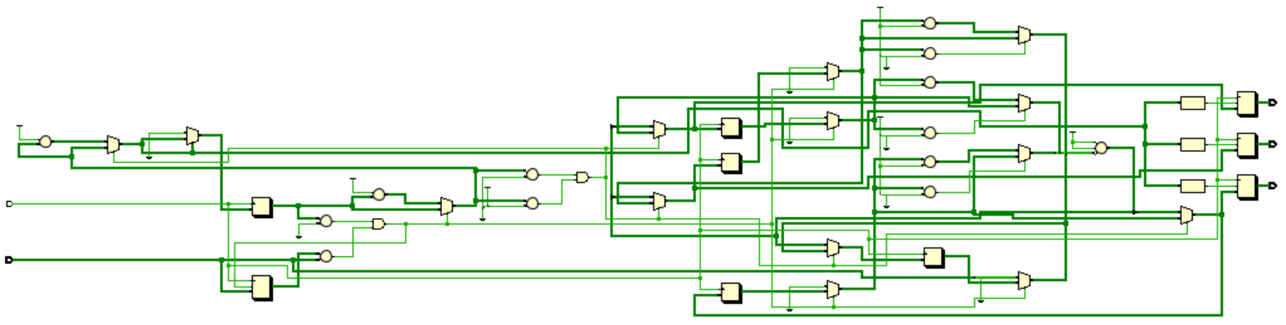
RTL Analysis :-



6 Binary to Decimal Conversion

```
module bin_8bit_to_bcd(BCD_h,BCD_t,BCD_o,binary8,clk);
output reg [3:0]BCD_h;
output reg [3:0]BCD_t;
output reg [3:0]BCD_o;
input [7:0]binary8;
input clk;
reg [3:0]temp_BCDh;
reg [3:0]temp_BCDt;
reg  [3:0]temp_BCDo;
reg  [3:0] i=0;
reg  [19:0]s_reg;
reg  [7:0]old_binary8=0;
always @(posedge clk)
begin
if(i==0&(old_binary8!=binary8))
begin
s_reg=20'd0;
old_binary8=binary8;
s_reg[7:0]=binary8;
temp_BCDh=s_reg[19:16];
temp_BCDt=s_reg[15:12];
temp_BCDo=s_reg[11:8];
i=i+1;
end
if(i>0&i<9)
begin
if(temp_BCDh>4)temp_BCDh=temp_BCDh+3;
if(temp_BCDt>4)temp_BCDt=temp_BCDt+3;
if(temp_BCDo>4)temp_BCDo=temp_BCDo+3;
s_reg[19:8]={temp_BCDh,temp_BCDt,temp_BCDo};
s_reg=s_reg<<1;
temp_BCDh=s_reg[19:16];
temp_BCDt=s_reg[15:12];
temp_BCDo=s_reg[11:8];
i=i+1;
end
if(i==9)
begin
i=0;
BCD_h=temp_BCDh;
BCD_t=temp_BCDt;
BCD_o=temp_BCDo;
end
end
endmodule
```

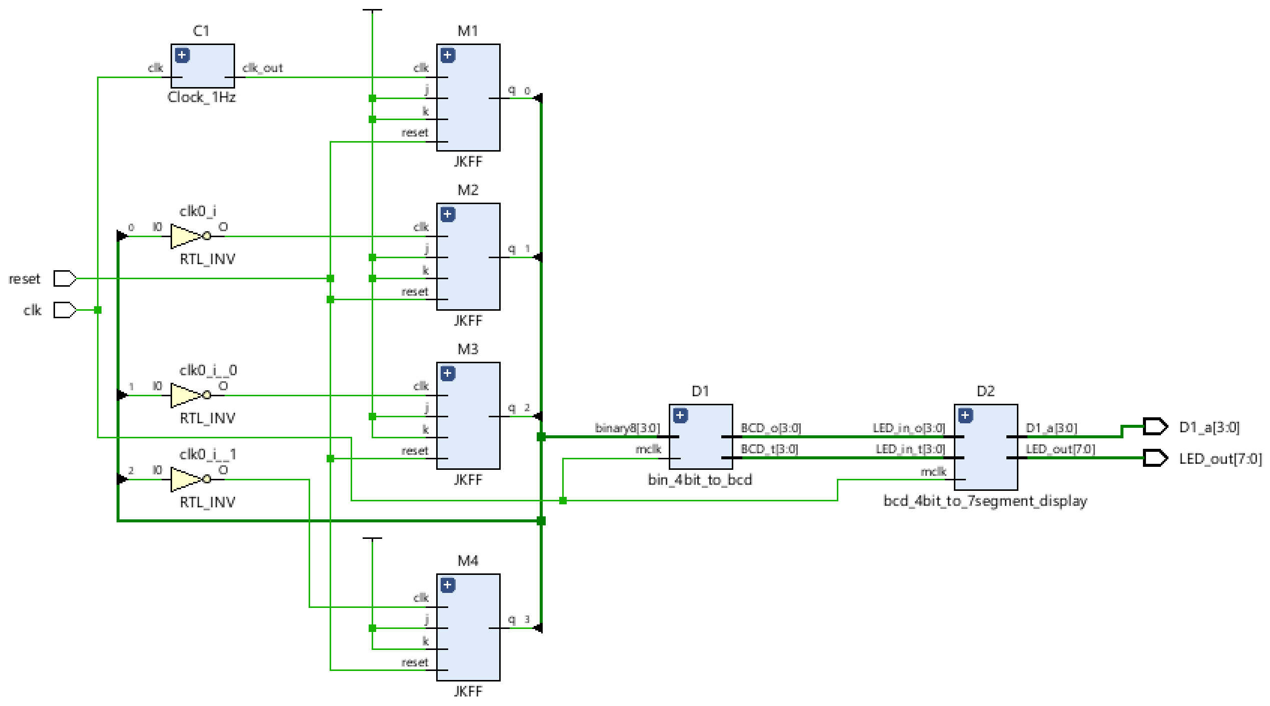
RTL Analysis :-



7 4-bit Asynchronous Down Counter

```
module async_down_count(D1_a,LED_out,clk,reset);
wire [3:0]q;
output [3:0]D1_a;
output [7:0]LED_out;
input clk,reset;
wire [3:0]BCD_t;
wire [3:0]BCD_o;
wire mclk;
Clock_1Hz C1(mclk,clk);          //Reduces clk to 1Hz
JKFF M1(q[0],1,1,mclk,reset);    //JKFF(q,j,k,clk,reset)
JKFF M2(q[1],1,1,~q[0],reset);
JKFF M3(q[2],1,1,~q[1],reset);
JKFF M4(q[3],1,1,~q[2],reset);
bin_4bit_to_bcd D1(BCD_t,BCD_o,q,clk);
bcd_4bit_to_7segment_display D2(D1_a,LED_out,BCD_t,BCD_o,clk);
endmodule
```

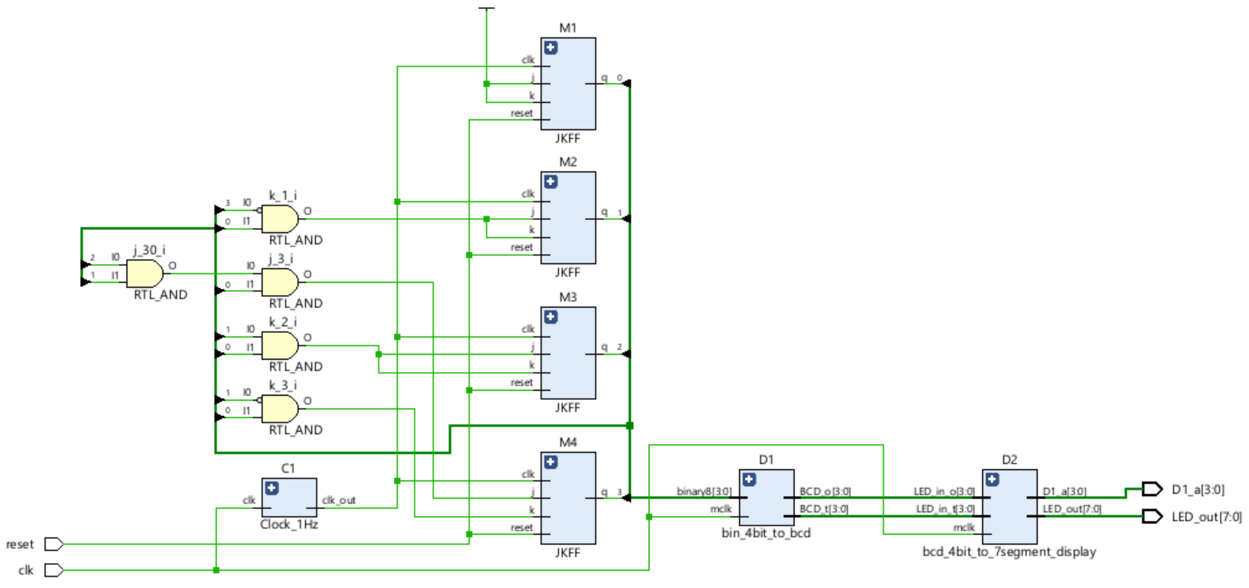
RTL Analysis :-



8 Mod10 Synchronous Up-Down Counter

```
module mod10_sync_count_updown(D1_a,LED_out,clk,reset);
wire [3:0]q;
input clk,reset;
output [3:0]D1_a;
output [7:0]LED_out;
wire [3:0]BCD_t;
wire [3:0]BCD_o;
wire [3:0]j,k;
wire mclk;
assign j[0]=1'b1 , k[0]=1'b1;
assign j[1]=~q[3]&q[0] , k[1]=~q[3]&q[0];
assign j[2]=q[1]&q[0] , k[2]=q[1]&q[0];
assign j[3]=q[2]&q[1]&q[0] , k[3]=~q[1]&q[0];
Clock_1Hz C1(mclk,clk);
JKFF M1(q[0],j[0],k[0],mclk,reset);
JKFF M2(q[1],j[1],k[1],mclk,reset);
JKFF M3(q[2],j[2],k[2],mclk,reset);
JKFF M4(q[3],j[3],k[3],mclk,reset);
bin_4bit_to_bcd D1(BCD_t,BCD_o,q,clk);
bcd_4bit_to_7segment_display D2(D1_a,LED_out,BCD_t,BCD_o,clk);
endmodule
```

RTL Analysis :-

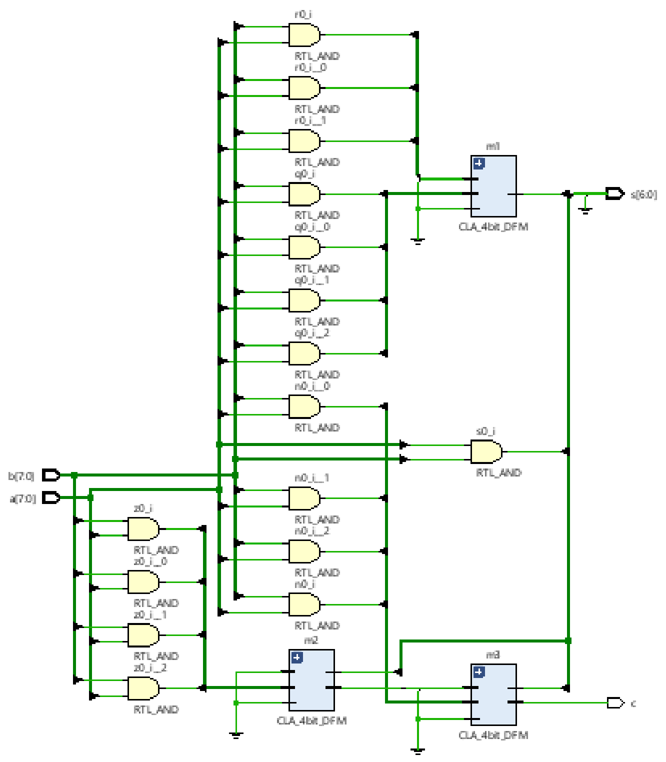


9 4x4 Multiplier

```
module multiplier4x4(s,c,a,b);
output [6:0]s;
output c;
input [7:0]a,b;
wire [4:0]q,r,p,z,w,n;
```

```
assign s[0]=a[0]&&b[0];
assign r[0]=b[0]&&a[1];
assign r[1]=b[0]&&a[2];
assign r[2]=b[0]&&a[3];
assign r[3]=1'b0;
assign q[0]=b[1]&&a[0];
assign q[1]=b[1]&&a[1];
assign q[2]=b[1]&&a[2];
assign q[3]=b[1]&&a[3];
CLA_4bit_DFM m1(q[4],p[3:0],r,q,1'b0);
assign s[1]=p[0];
assign z[0]=b[2]&&a[0];
assign z[1]=b[2]&&a[1];
assign z[2]=b[2]&&a[2];
assign z[3]=b[2]&&a[3];
CLA_4bit_DFM m2(w[4],w[3:0],p[4:1],z,1'b0);
assign s[2]=w[0];
assign n[0]=b[3]&&a[0];
assign n[1]=b[3]&&a[1];
assign n[2]=b[3]&&a[2];
assign n[3]=b[3]&&a[3];
CLA_4bit_DFM m3(c,s[6:3],w[4:1],n,1'b0);
endmodule
```

RTL Analysis :-

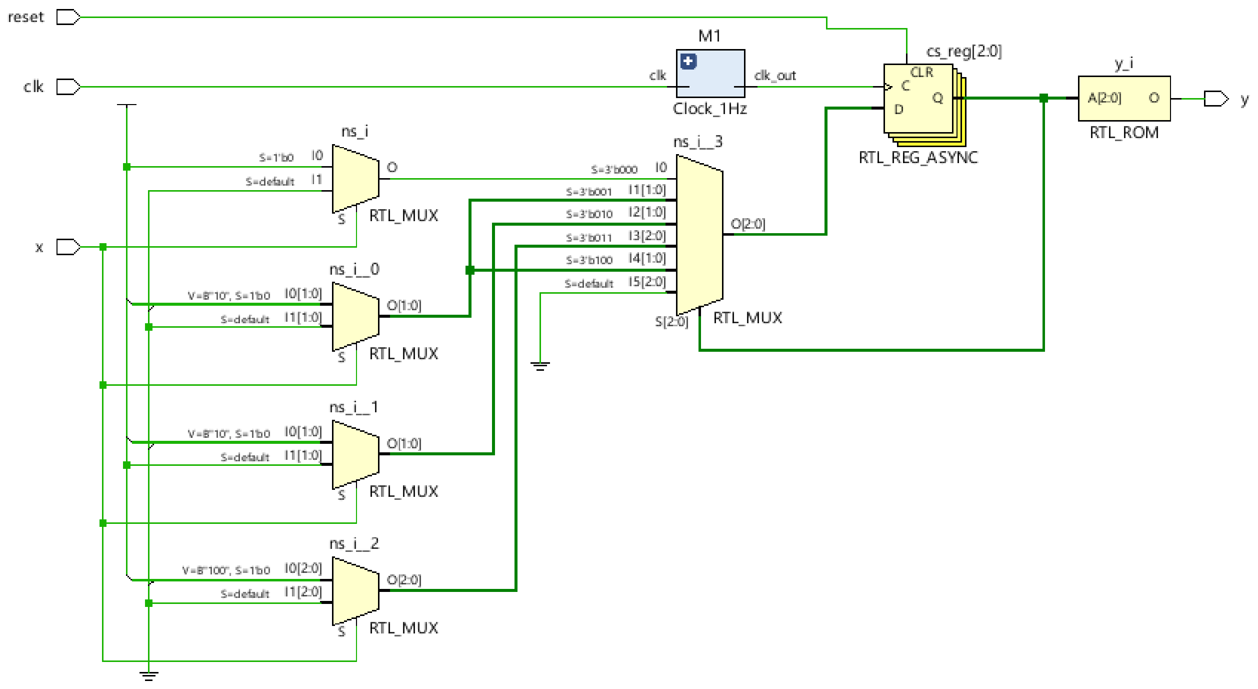


10 FSM 0010 Moore Machine

```
module FSM_moore_0010(y,x,clk,reset);
output reg y;
input x,clk,reset;
wire mclk;
reg[2:0] cs,ns;
parameter s0=3'b000,
          s1=3'b001,
          s2=3'b010,
          s3=3'b011,
          s4=3'b100;
Clock_1Hz M1(mclk,clk);
always @(posedge mclk, posedge reset)
begin
if(reset==1)
cs=s0;
else
cs=ns;
end
always @(cs,x)
begin
case(cs)
s0: begin
    if(x==0)
        ns=s1;
        else
        ns=s0;
    end
s1: begin
    if(x==0)
        ns=s2;
        else
        ns=s0;
    end
s2: begin
    if(x==0)
        ns=s2;
        else
        ns=s3;
    end
s3: begin
    if(x==0)
        ns=s4;
        else
        ns=s0;
    end
s4: begin
    if(x==0)
        ns=s2;
```

```
            else
            ns=s0 ;
        end
default :  ns=s0 ;
endcase
end
always @( cs )
begin
case ( cs )
s0 :  y=0;
s1 :  y=0;
s2 :  y=0;
s3 :  y=0;
s4 :  y=1;
default :  y=0;
endcase
end
endmodule
```

RTL Analysis :-



11 Memory Block :-

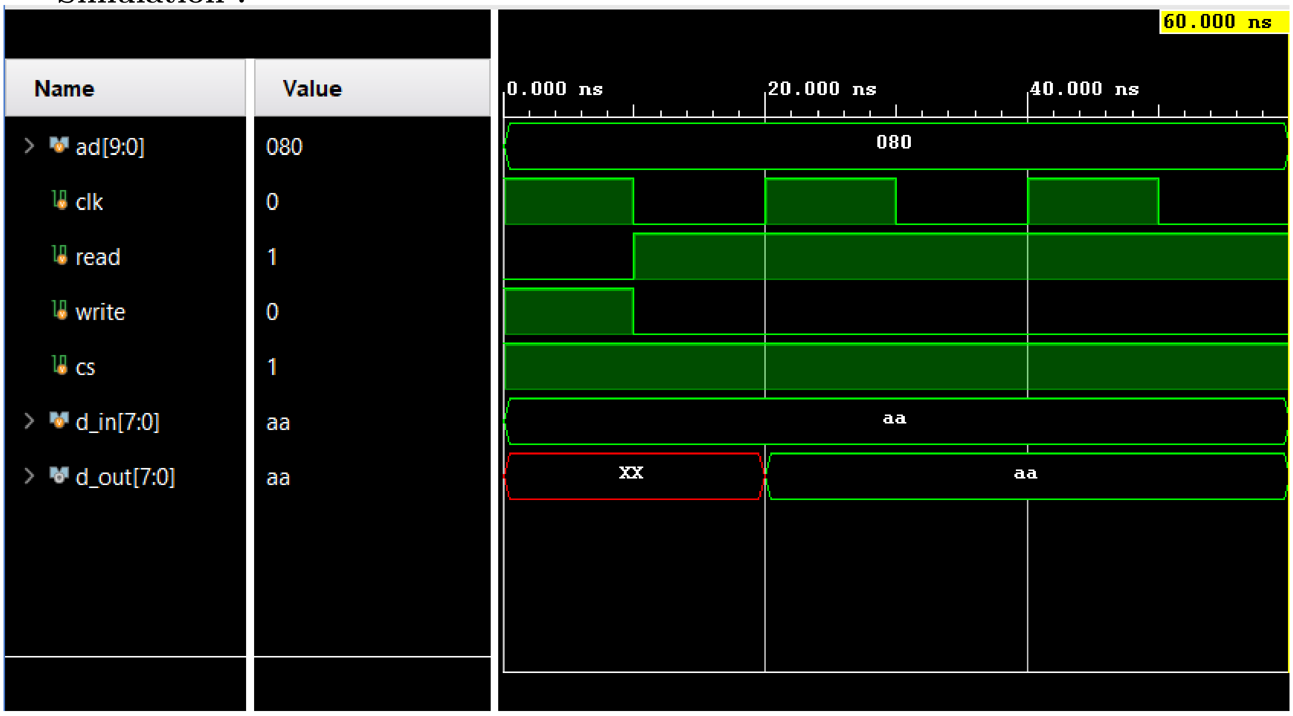
Code :-

```
module memory_read_wire_1024x8(addr,data_out,data_in,clk,rd,wr,cs);
input [9:0]addr;
input clk,rd,wr,cs;
input [7:0]data_in;
reg [7:0] mem[1023:0];
output reg [7:0]data_out;
assign data=(cs)?data_out:8'bz;
always@(posedge clk)
begin
if(cs&&wr&&!rd)
mem[addr]=data_in;
end
always@(posedge clk)
begin
if(cs&&rd&&!wr)
data_out=mem[addr];
end
endmodule
```

Test-bench Code :-

```
module memory_read_write_1024x8_tb();
reg [9:0]ad;
reg clk,read,write,cs;
reg [7:0]d_in;
wire [7:0]d_out;
memory_read_wire_1024x8 dut(.addr(ad),.data_out(d_out),.data_in(d_in),.clk(cl
initial begin
clk=1'b1;
forever #10 clk=~clk;
end
initial
begin
cs=1'b1;
ad=10'b0010000000;
read=0;
write=1;
d_in=8'b10101010;
#10
read=1;
write=0;
#50
$stop;
end
endmodule
```

Simulation :-



12 8-bit ALU

Operations:-

1. Pass A
2. Addition
3. Subtraction
4. 4x4 Multiplication
5. Comparator
6. 8-bit Synchronous Up Counter
7. 8-bit Synchronous Down Counter
8. FSM 0010 Moore Machine

The ALU converts the output from all the modules into 9-bit. Further converts it into 4-bit BCD and displays it on LED 7-segment display.

12.1 Addition and Subtraction

```
module ADD_SUB_8bit_CLA(s,c4,a,b,e);
output [7:0]s;
output c4;
input [7:0]a,b;
input e;
wire [7:0]r,p,q,g;
wire n,c8;
assign r[0]=e^b[0];
assign r[1]=e^b[1];
assign r[2]=e^b[2];
assign r[3]=e^b[3];
assign r[4]=e^b[4];
assign r[5]=e^b[5];
assign r[6]=e^b[6];
assign r[7]=e^b[7];
CLA_8bit_DFM M1(p,c4,a,r,e);
assign n=(~c4)&&e;
assign q[0]=p[0]^n;
assign q[1]=p[1]^n;
assign q[2]=p[2]^n;
assign q[3]=p[3]^n;
assign q[4]=p[4]^n;
assign q[5]=p[5]^n;
assign q[6]=p[6]^n;
assign q[7]=p[7]^n;
assign g=8'b00000000;
CLA_8bit_DFM M2(s,c8,q,g,n);
endmodule
```

12.2 4x4 Multiplication

Refer section 9.

12.3 Comparator

```
module com_8bit#(parameter size=8)(y,a,b);
input [7:0]a,b;
output reg [7:0]y;
always@(a,b)
begin
if(a>b)
y=8'b01100100;
else if(a<b)
y=8'b00000001;
else
y=8'b00001010;
end
endmodule
```

12.4 Clock 1Hz

```
module Clock_1Hz(clk_out ,clk);
output clk_out;
input clk;
reg [25:0]count=0;
reg clk_out;
always@(posedge clk)
begin
count<=count+1;
if(count==50_000_000)
begin
count<=0;
clk_out=~clk_out;
end
end
endmodule
```

12.5 8-bit Synchronous Up-Down Counter

```
module sync_8bit_up_down_counter(Q,clk ,reset ,e);
output reg [7:0]Q;
input clk,reset,e;
wire [7:0]j,k,q;
wire mclk;
assign j[0]=1 , k[0]=1;
assign j[1]=q[0];
assign k[1]=q[0];
assign j[2]=q[0]&&q[1];
assign k[2]=q[0]&&q[1];
assign j[3]=q[0]&&q[1]&&q[2];
assign k[3]=q[0]&&q[1]&&q[2];
assign j[4]=q[0]&&q[1]&&q[2]&&q[3];
assign k[4]=q[0]&&q[1]&&q[2]&&q[3];
assign j[5]=q[0]&&q[1]&&q[2]&&q[3]&&q[4];
assign k[5]=q[0]&&q[1]&&q[2]&&q[3]&&q[4];
assign j[6]=q[0]&&q[1]&&q[2]&&q[3]&&q[4]&&q[5];
assign k[6]=q[0]&&q[1]&&q[2]&&q[3]&&q[4]&&q[5];
assign j[7]=q[0]&&q[1]&&q[2]&&q[3]&&q[4]&&q[5]&&q[6];
assign k[7]=q[0]&&q[1]&&q[2]&&q[3]&&q[4]&&q[5]&&q[6];
Clock_1Hz M1(mclk,clk);
JKFF M2(q[0],j[0],k[0],mclk,reset);
JKFF M3(q[1],j[1],k[1],mclk,reset);
JKFF M4(q[2],j[2],k[2],mclk,reset);
JKFF M5(q[3],j[3],k[3],mclk,reset);
JKFF M6(q[4],j[4],k[4],mclk,reset);
JKFF M7(q[5],j[5],k[5],mclk,reset);
JKFF M8(q[6],j[6],k[6],mclk,reset);
JKFF M9(q[7],j[7],k[7],mclk,reset);
always@(posedge clk)
```

```
begin
Q[0]=e^q[0];
Q[1]=e^q[1];
Q[2]=e^q[2];
Q[3]=e^q[3];
Q[4]=e^q[4];
Q[5]=e^q[5];
Q[6]=e^q[6];
Q[7]=e^q[7];
end
endmodule
```

12.6 FSM 0010 Moore Machine

Refer section 10.

12.7 9-bit Binary to BCD

```
module bin_9bit_to_bcd(BCD_h,BCD_t,BCD_o,binary9,clk);
output reg [3:0]BCD_h;
output reg [3:0]BCD_t;
output reg [3:0]BCD_o;
input [8:0]binary9;
input clk;
reg [3:0]temp_BCDh;
reg [3:0]temp_BCDt;
reg [3:0]temp_BCDo;
reg [3:0] i=0;
reg [20:0]s_reg;
reg [7:0]old_binary9=0;
always @(posedge clk)
begin
if(i==0&(old_binary9!=binary9))
begin
s_reg=21'd0;
old_binary9=binary9;
s_reg[7:0]=binary9;
temp_BCDh=s_reg[20:17];
temp_BCDt=s_reg[16:13];
temp_BCDo=s_reg[12:9];
i=i+1;
end
if(i>0&i<10)
begin
if(temp_BCDh>4)temp_BCDh=temp_BCDh+3;
if(temp_BCDt>4)temp_BCDt=temp_BCDt+3;
if(temp_BCDo>4)temp_BCDo=temp_BCDo+3;
s_reg[20:9]={temp_BCDh,temp_BCDt,temp_BCDo};
s_reg=s_reg<<1;
temp_BCDh=s_reg[20:17];
```

```
temp_BCDt=s_reg[16:13];
temp_BCDo=s_reg[12:9];
i=i+1;
end
if(i==10)
begin
i=0;
BCD_h=temp_BCDh;
BCD_t=temp_BCDt;
BCD_o=temp_BCDo;
end
end
endmodule
```

12.8 BCD to 7-segment display

```
module bcd_8bit_to_7segment_display(D1_a,LED_out,LED_in_h,LED_in_t,
                                              LED_in_o,ds,mclk);
output reg [7:0]LED_out;
input [3:0]LED_in_o,LED_in_t,LED_in_h;
input mclk;
input ds;
output reg [3:0]D1_a;
reg [19:0] refresh_counter=20'd0;
wire [1:0] LED_activating_counter;
always @(posedge mclk)
    begin
    refresh_counter <= refresh_counter + 1;
    end
 assign LED_activating_counter = refresh_counter[19:18];
always @(posedge mclk)
    begin
      case(LED_activating_counter)
         2'b00: begin D1_a=4'b1110;
              case(LED_in_o)
              4'b0000: LED_out = 8'b11000000; // "0"
              4'b0001: LED_out = 8'b11111001; // "1"
              4'b0010: LED_out = 8'b10100100; // "2"
              4'b0011: LED_out = 8'b10110000; // "3"
              4'b0100: LED_out = 8'b10011001; // "4"
              4'b0101: LED_out = 8'b10010010; // "5"
              4'b0110: LED_out = 8'b10000010; // "6"
              4'b0111: LED_out = 8'b11111000; // "7"
              4'b1000: LED_out = 8'b10000000; // "8"
              4'b1001: LED_out = 8'b10010000; // "9"
                default: LED_out = 8'b11000000; // "0"
              endcase
             end
         2'b01: begin D1_a=4'b1101;
         case(LED_in_t)
```

```
                        4'b0000: LED_out = 8'b11000000; // "0"
                        4'b0001: LED_out = 8'b11111001; // "1"
                        4'b0010: LED_out = 8'b10100100; // "2"
                        4'b0011: LED_out = 8'b10110000; // "3"
                        4'b0100: LED_out = 8'b10011001; // "4"
                        4'b0101: LED_out = 8'b10010010; // "5"
                        4'b0110: LED_out = 8'b10000010; // "6"
                        4'b0111: LED_out = 8'b11111000; // "7"
                        4'b1000: LED_out = 8'b10000000; // "8"
                        4'b1001: LED_out = 8'b10010000; // "9"
                          default: LED_out = 8'b11000000; // "0"
                        endcase
                        end
            2'b10: begin D1_a=4'b1011;
            case(LED_in_h)
                        4'b0000: LED_out = 8'b11000000; // "0"
                        4'b0001: LED_out = 8'b11111001; // "1"
                        4'b0010: LED_out = 8'b10100100; // "2"
                        4'b0011: LED_out = 8'b10110000; // "3"
                        4'b0100: LED_out = 8'b10011001; // "4"
                        4'b0101: LED_out = 8'b10010010; // "5"
                        4'b0110: LED_out = 8'b10000010; // "6"
                        4'b0111: LED_out = 8'b11111000; // "7"
                        4'b1000: LED_out = 8'b10000000; // "8"
                        4'b1001: LED_out = 8'b10010000; // "9"
                          default: LED_out = 8'b11000000; // "0"
                        endcase
                        end   // "Blank"
            2'b11: begin D1_a=4'b0111;
            case(ds)
                  1'b0: LED_out = 8'b10111111; // "-"
                  1'b1: LED_out = 8'b11111111; // "0"
                        default: LED_out = 8'b11111111; // "0"
                        endcase
                        end // "Blank"
        endcase
      end
endmodule
```

FINAL CODE

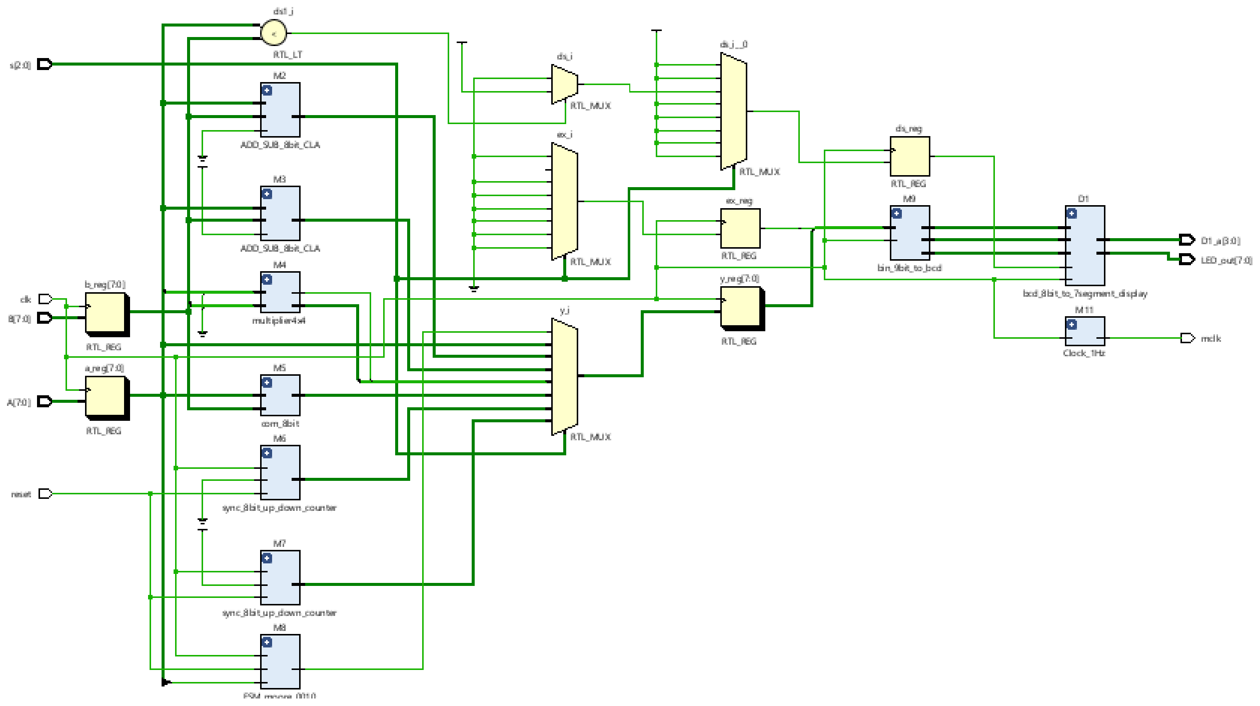
```
module ALU_8bit (D1_a,LED_out,A,B,s,clk,mclk,reset);
output [7:0]LED_out;
output [3:0]D1_a;
output mclk;
input [7:0]A,B;
input clk,reset;
input [2:0]s;
//---------------------------------------------------------------------//
wire [3:0]BCD_h,BCD_t,BCD_o;
wire [7:0]y1,y2,y4,y5,y6,y7,y8;
wire [6:0]y3;
wire c81,c82,c83,mclk;
reg [7:0]y,a,b;
reg ex,ds;
//---------------------------------------------------------------------//
ADD_SUB_8bit_CLA M2(y1,C81,a,b,1'b0);                          // Addition
ADD_SUB_8bit_CLA M3(y2,C82,a,b,1'b1);                          // Subtraction
multiplier4x4 M4(y3,c83,a[3:0],b[3:0]);                        // Multiplication
com_8bit M5(y4,a,b);                                           // Comparator
sync_8bit_up_down_counter M6(y5,clk,reset,1'b0);               // Up-Counter
sync_8bit_up_down_counter M7(y6,clk,reset,1'b1);               // Down-Counter
FSM_moore_0010 M8(y7,a[0],clk,reset);                          // 0010-FSM
//---------------------------------------------------------------------//
Clock_1Hz M11(mclk,clk);
//---------------------------------------------------------------------//
always@(posedge clk)
begin
a=A;
b=B;
end
//---------------------------------------------------------------------//
always@(posedge clk)
begin
case(s)
3'b000:  begin y=a; ds=1'b1 ;ex=0; end  // 'a' value pass
3'b001:  begin if(c81==1'b1) ex=1'b1; else ex=1'b0; ds=1'b1 ;y=y1 ; end
//Addition
3'b010:  begin if(a<b) ds=1'b0; else ds=1'b1; ex=1'b0 ;y=y2 ; end
// Subtraction
3'b011:  begin y={c83,y3}; ds=1'b1 ;ex=0; end
// Multiplication
3'b100:  begin y=y4; ds=1'b1 ;ex=0; end
// Comparator
3'b101:  begin y=y5; ds=1'b1 ;ex=0; end
// Up-Counter
3'b110:  begin y=y6; ds=1'b1 ;ex=0; end
// Down-Counter
3'b111:  begin y=y7; ds=1'b1 ;ex=0; end
```

```
// FSM - 0010 Sequence detector
endcase
end
//-----------------------------------------------------------------------------//
bin_9bit_to_bcd M9(BCD_h,BCD_t,BCD_o,{ex,y},clk);
bcd_8bit_to_7segment_display D1(D1_a,LED_out,BCD_h,BCD_t,BCD_o,ds,clk);
endmodule
```

Constraint File

```
set_property -dict {PACKAGE_PIN F14 IOSTANDARD LVCMOS33} [get_ports clk]
# Display Constraints
set_property -dict {PACKAGE_PIN H3 IOSTANDARD LVCMOS33} [get_ports
{D1_a[0]}]
set_property -dict {PACKAGE_PIN J4 IOSTANDARD LVCMOS33} [get_ports
{D1_a[1]}]
set_property -dict {PACKAGE_PIN F3 IOSTANDARD LVCMOS33} [get_ports
{D1_a[2]}]
set_property -dict {PACKAGE_PIN E4 IOSTANDARD LVCMOS33} [get_ports
{D1_a[3]}]
set_property -dict {PACKAGE_PIN F4 IOSTANDARD LVCMOS33} [get_ports
{LED_out[0]}]
set_property -dict {PACKAGE_PIN J3 IOSTANDARD LVCMOS33} [get_ports
{LED_out[1]}]
set_property -dict {PACKAGE_PIN D2 IOSTANDARD LVCMOS33} [get_ports
{LED_out[2]}]
set_property -dict {PACKAGE_PIN C2 IOSTANDARD LVCMOS33} [get_ports
{LED_out[3]}]
set_property -dict {PACKAGE_PIN B1 IOSTANDARD LVCMOS33} [get_ports
{LED_out[4]}]
set_property -dict {PACKAGE_PIN H4 IOSTANDARD LVCMOS33} [get_ports
{LED_out[5]}]
set_property -dict {PACKAGE_PIN D1 IOSTANDARD LVCMOS33} [get_ports
{LED_out[6]}]
set_property -dict {PACKAGE_PIN C1 IOSTANDARD LVCMOS33} [get_ports
{LED_out[7]}]
# [7:0]A,B;
# [2:0]s;
# clk,xin,reset
#Input Constraints
set_property -dict {PACKAGE_PIN K1 IOSTANDARD LVCMOS33} [get_ports {A[0]}]
set_property -dict {PACKAGE_PIN K2 IOSTANDARD LVCMOS33} [get_ports {A[1]}]
set_property -dict {PACKAGE_PIN L1 IOSTANDARD LVCMOS33} [get_ports {A[2]}]
set_property -dict {PACKAGE_PIN M1 IOSTANDARD LVCMOS33} [get_ports {A[3]}]
set_property -dict {PACKAGE_PIN M2 IOSTANDARD LVCMOS33} [get_ports {A[4]}]
set_property -dict {PACKAGE_PIN N1 IOSTANDARD LVCMOS33} [get_ports {A[5]}]
set_property -dict {PACKAGE_PIN N2 IOSTANDARD LVCMOS33} [get_ports {A[6]}]
set_property -dict {PACKAGE_PIN P1 IOSTANDARD LVCMOS33} [get_ports {A[7]}]
set_property -dict {PACKAGE_PIN P2 IOSTANDARD LVCMOS33} [get_ports {B[0]}]
set_property -dict {PACKAGE_PIN R1 IOSTANDARD LVCMOS33} [get_ports {B[1]}]
set_property -dict {PACKAGE_PIN R2 IOSTANDARD LVCMOS33} [get_ports {B[2]}]
set_property -dict {PACKAGE_PIN T1 IOSTANDARD LVCMOS33} [get_ports {B[3]}]
set_property -dict {PACKAGE_PIN T2 IOSTANDARD LVCMOS33} [get_ports {B[4]}]
set_property -dict {PACKAGE_PIN U1 IOSTANDARD LVCMOS33} [get_ports {B[5]}]
set_property -dict {PACKAGE_PIN U2 IOSTANDARD LVCMOS33} [get_ports {B[6]}]
set_property -dict {PACKAGE_PIN V2 IOSTANDARD LVCMOS33} [get_ports {B[7]}]
#s[2:0]
set_property -dict {PACKAGE_PIN J5 IOSTANDARD LVCMOS33} [get_ports {s[0]}]
set_property -dict {PACKAGE_PIN J1 IOSTANDARD LVCMOS33} [get_ports {s[1]}]
set_property -dict {PACKAGE_PIN H2 IOSTANDARD LVCMOS33} [get_ports {s[2]}]
# reset
set_property -dict {PACKAGE_PIN J2 IOSTANDARD LVCMOS33} [get_ports reset]
#To see the mclk
set_property -dict {PACKAGE_PIN A4 IOSTANDARD LVCMOS33} [get_ports mclk]
```

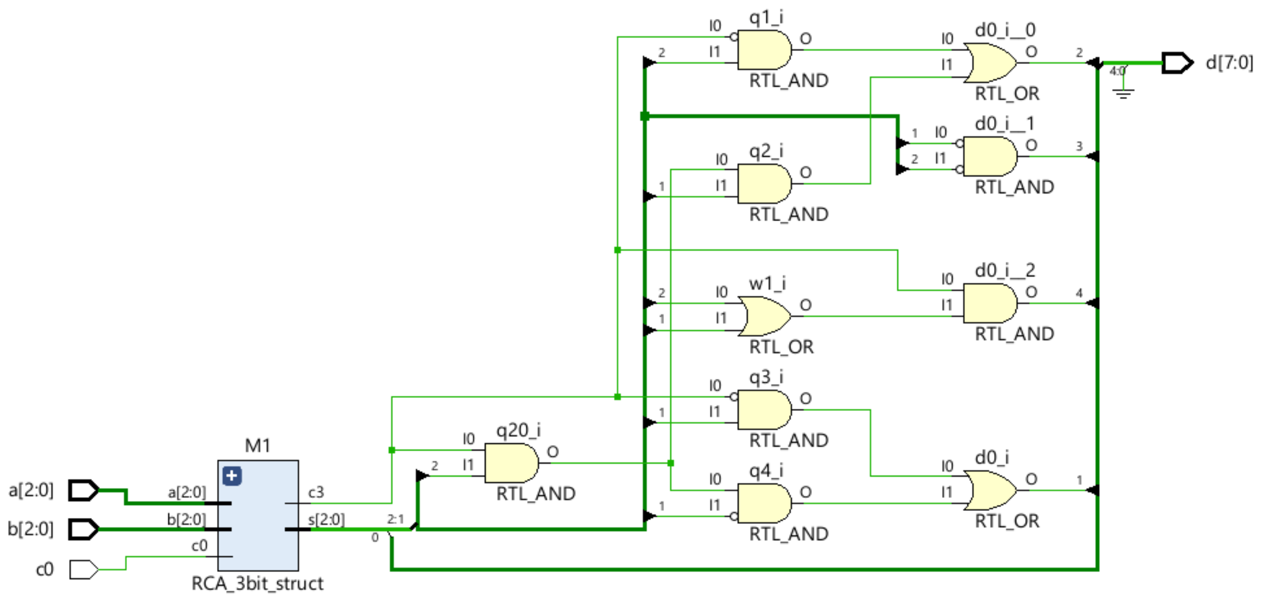
RTL Analysis :-



13 3-bit Binary to BCD

```
module bin_3bit_to_bcd(d,a,b,c0);
output [7:0]d;
input [2:0]a,b;
input c0;
wire [2:0]s;
wire c3;
wire w1,r1,s1,s2,q1,q2,q3,q4;
RCA_3bit_struct M1(s,c3,a,b,c0);
or(d[0],0,s[0]);
and(d[5],0,s[0]); //d7,d5,d6
and(d[6],0,s[0]);
and(d[7],0,s[0]);
or(w1,s[2],s[1]);
and(d[4],c3,w1);    //d4
and(d[3],s1,s2);    //d3
not(r1,c3);
not(s1,s[1]);
not(s2,s[2]);
and(q1,r1,s[2]);
and(q2,c3,s[2],s[1]);
and(q3,r1,s[1]);
and(q4,c3,s[2],s1);
or(d[2],q1,q2);    //d2
or(d[1],q3,q4);    //d1
endmodule
```

RTL Analysis :-



THE END