# Lab Assignment No. 1:
## 4-Bit Binary Full Adder & Subtractor

**Half Adder:**

```
module half_adder(sum,carry,a,b);
input a,b;
output sum,carry;
xor g1(sum,a,b);
and  g2(carry,a,b);
endmodule
```

**Full Adder:**

```
module full_adder(sum,carry,a,b,c);
input a,b,c;
output sum,carry;
wire w1,w2,w3;
half_adder m1(w1,w2,a,b);
half_adder m2(sum,w3,w1,c);
or   g1(carry,w2,w3);
endmodule
```

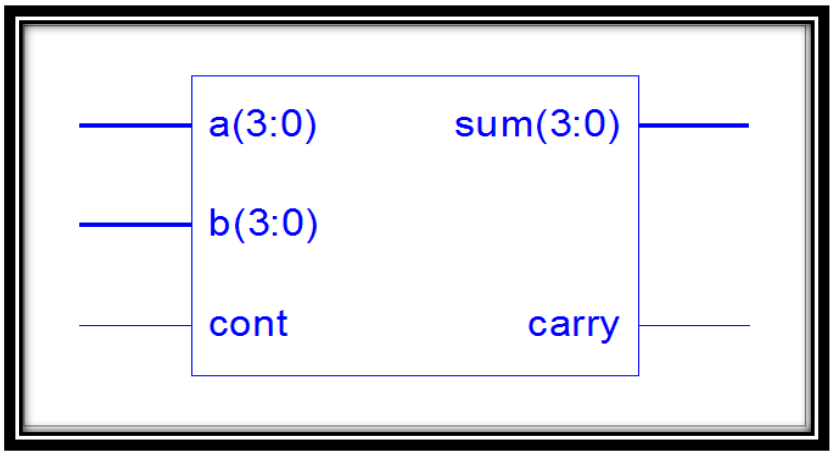
**4-Bit Adder:**

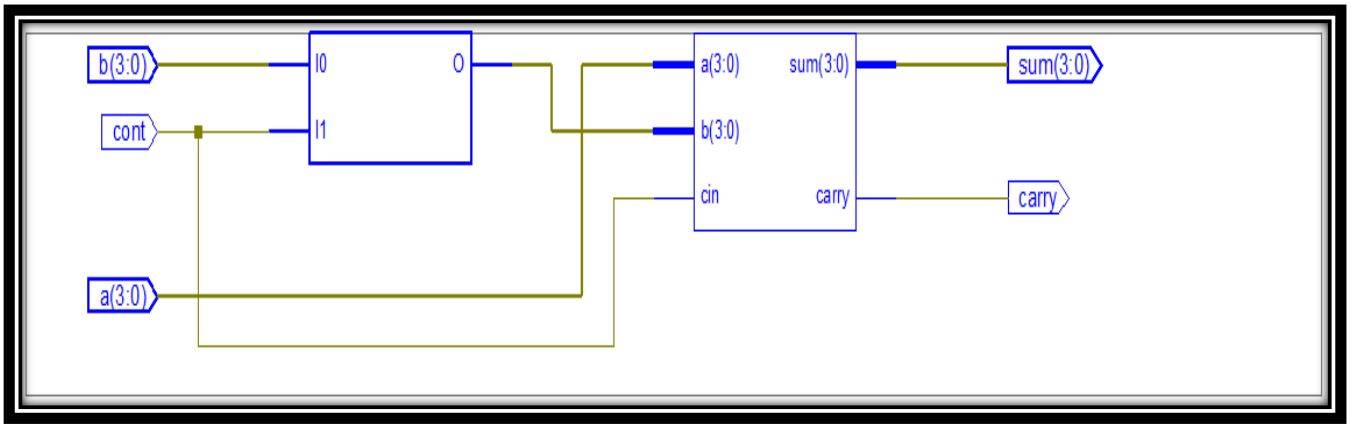
```
module binary_adder(sum,carry,a,b,cin);
input  [3:0] a,b;
input  cin;
output [3:0] sum;
output carry;
wire   w1,w2,w3;
full_adder M1(sum[0],w1,a[0],b[0],cin);
full_adder M2(sum[1],w2,a[1],b[1],w1);
full_adder M3(sum[2],w3,a[2],b[2],w2);
full_adder M4(sum[3],carry,a[3],b[3],w3);
endmodule
```

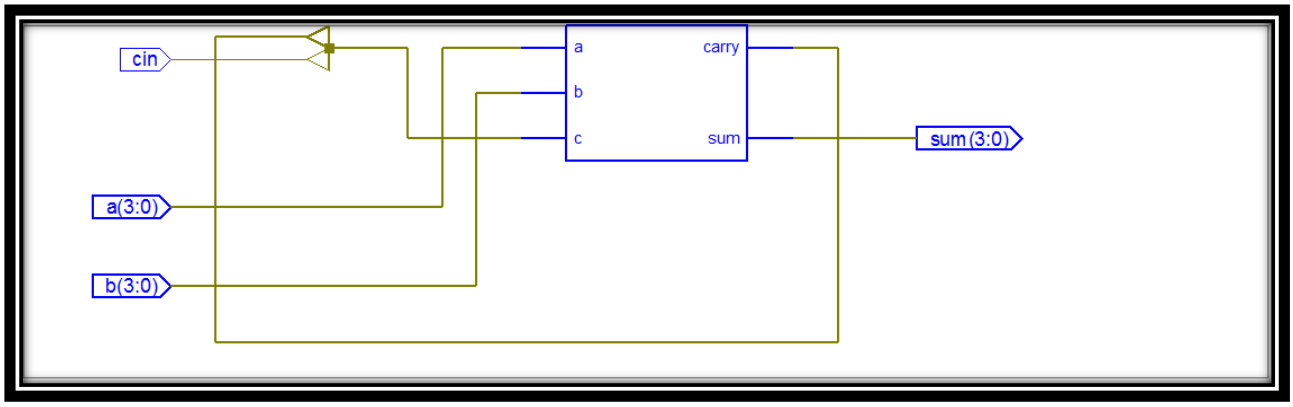
**4-Bit Adder/ Subtractor:**

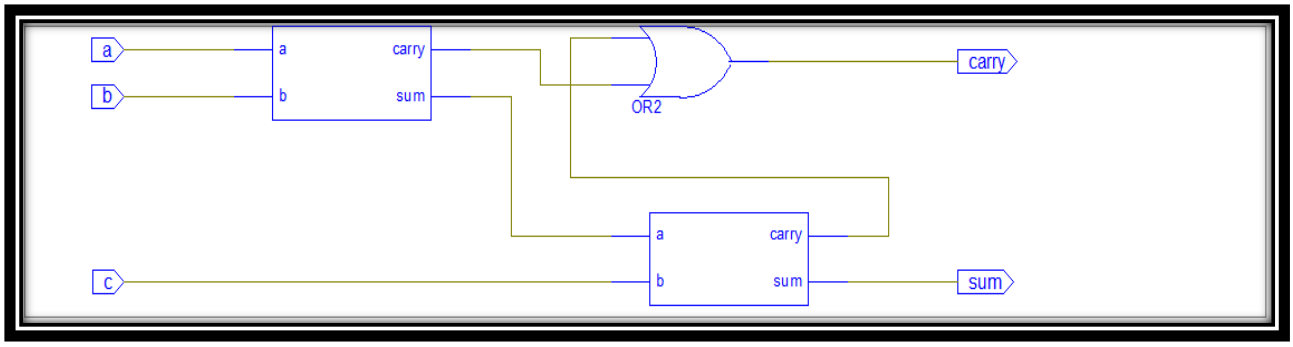
```
module binary_add_sub(sum,carry,a,b,cont);
input [3:0] a,b;
input cont;
output [3:0] sum;
output carry;
wire [3:0] w1;
binary_adder m1(sum,carry,a,w1,cont);
xor g1(w1[0],cont,b[0]);
xor g2(w1[1],cont,b[1]);
xor g3(w1[2],cont,b[2]);
xor g4(w1[3],cont,b[3]);
endmodule
```

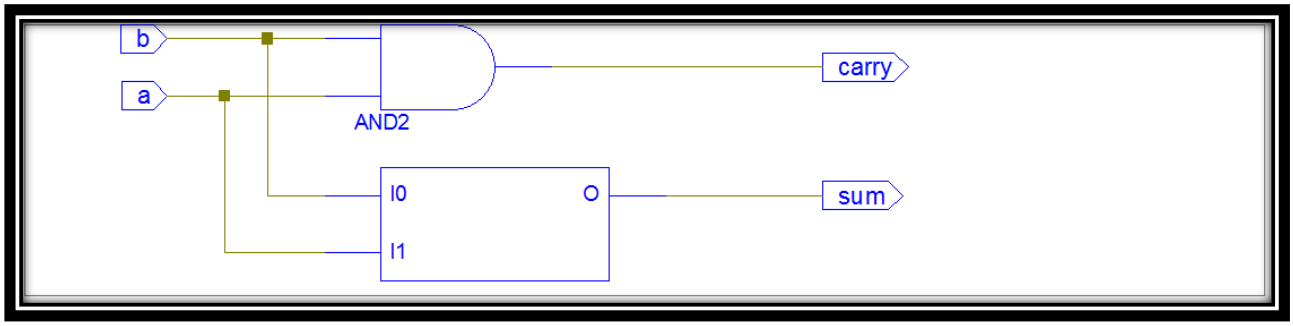
**RTL Schematics:**




cin
a(3:0)
b(3:0)
a
b
c
carry
sum
sum(3:0)


a
b
a
b
carry
sum
OR2
carry
c
a
b
carry
sum
sum


b
a
AND2
carry
I0
I1
O
sum

# Lab Assignment No. 2:
## Common Bus Architecture Structural Modeling

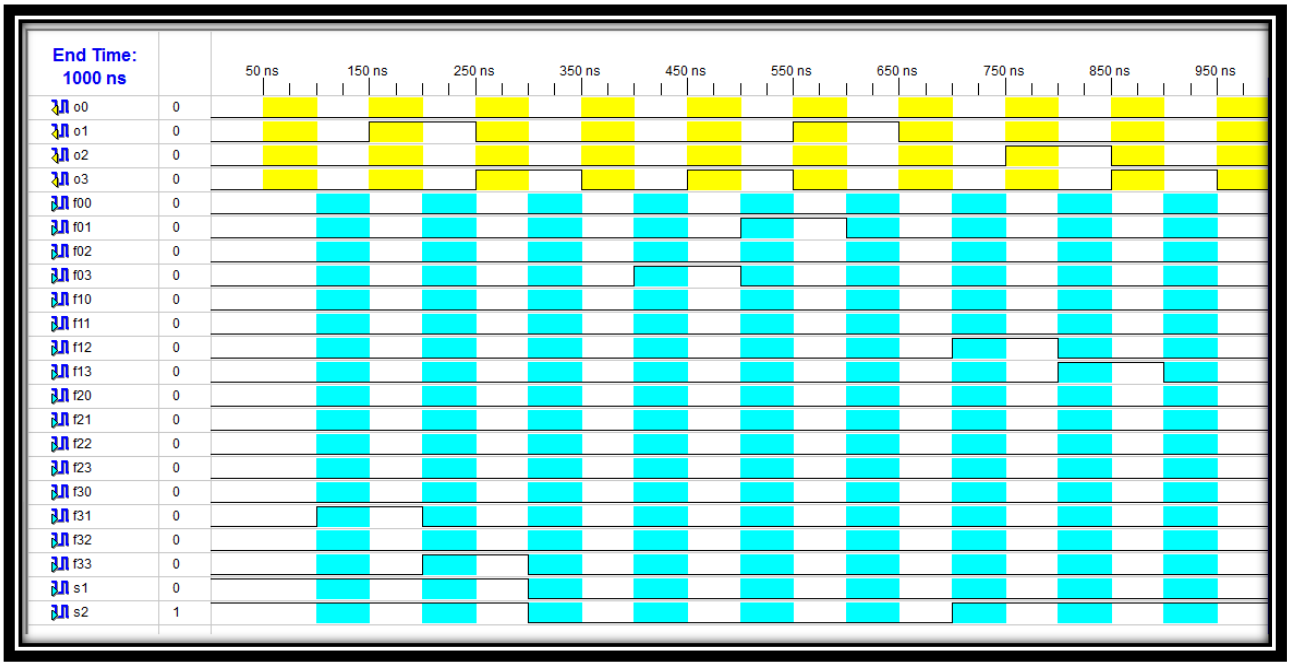
### Multiplexer:

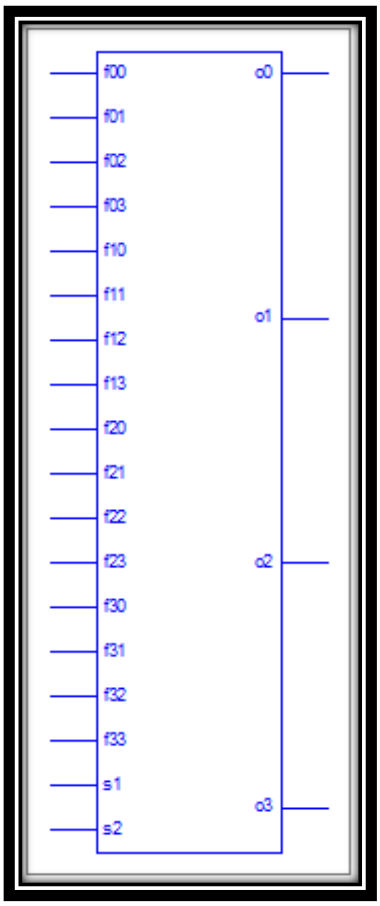
```
module mux_4(o,i1,i2,i3,i4,s1,s2);
input i1,i2,i3,i4,s1,s2;
output o;
wire w1,w2,w3,w4,w5,w6;
not g1(w1,s1);
not g2 (w2,s2);
and g3(w3,i1,w1,w2);
and g4 (w4,i2,w1,s2);
and g5(w5,i3,s1,w2);
and g6(w6,i4,s1,s2);
or  g7(o,w3,w4,w5,w6);
endmodule
```

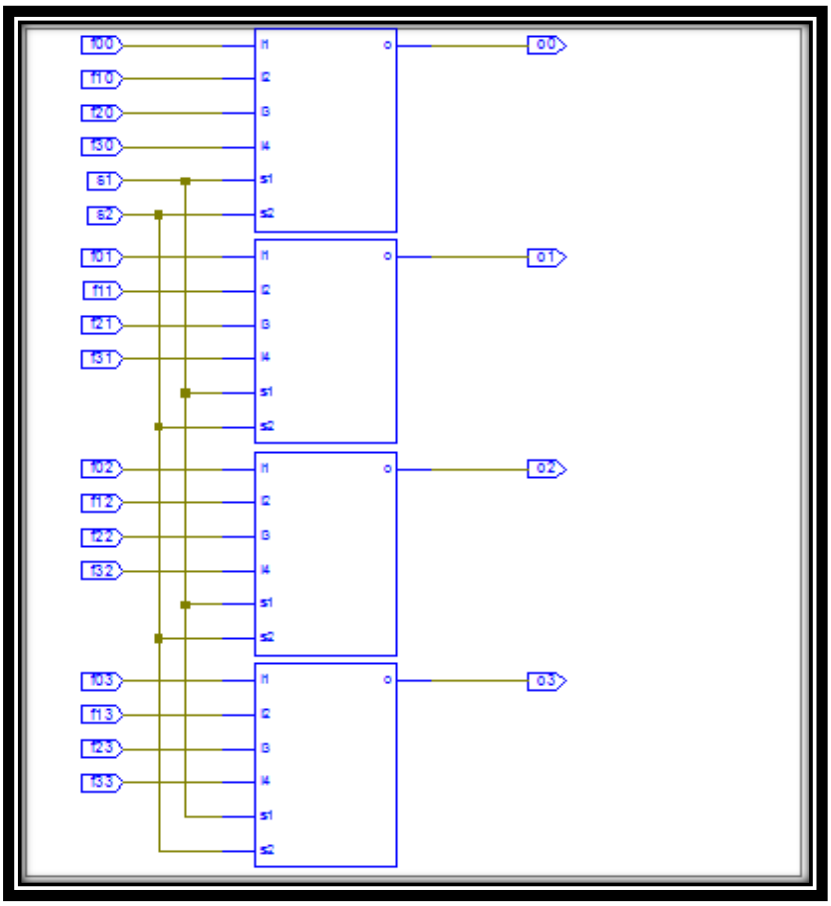
### Common Bus Multiplexer:

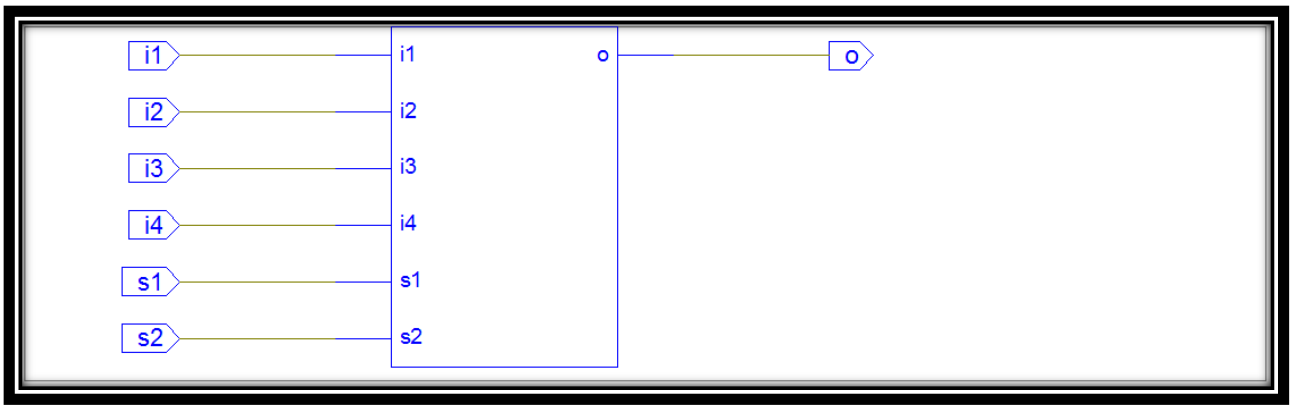
```
module com_mux(o0,o1,o2,o3,f00,f01,f02,f03,f10,f11,f12,f13,f20,f21,f22,f23,
f30,f31,f32,f33,s1,s2);
input f00,f01,f02,f03,f10,f11,f12,f13,f20,f21,f22,f23,f30,f31,f32,f33,s1,s2;
output o0,o1,o2,o3;
mux_4 m1(o0,f00,f10,f20,f30,s1,s2);
mux_4 m2(o1,f01,f11,f21,f31,s1,s2);
mux_4 m3(o2,f02,f12,f22,f32,s1,s2);
mux_4 m4(o3,f03,f13,f23,f33,s1,s2);
endmodule
```

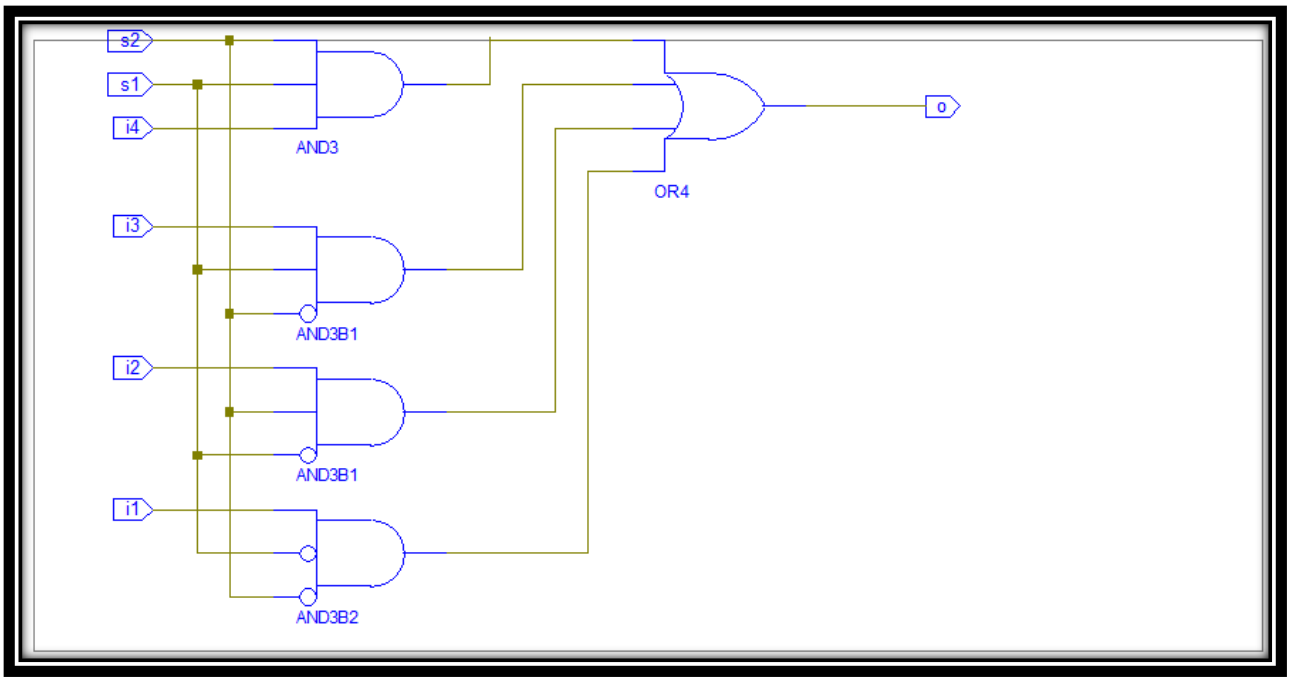
## RTL Schematics:




i1
i2
i3
i4
s1
s2
i1
i2
i3
i4
s1
s2
o
o


s2
s1
i4
AND3
i3
AND3B1
i2
AND3B1
i1
AND3B2
OR4
o

# Lab Assignment No. 3:
## Arithmetic Logic Unit (ALU) Structural Modeling

### 4-Bit AND Module:

```
module AND(f1,a,b);
output [3:0] f1;
input [3:0] a,b;
and g1(f1[0],a[0],b[0]);
and g2(f1[1],a[1],b[1]);
and g3(f1[2],a[2],b[2]);
and g4(f1[3],a[3],b[3]);
endmodule
```

### 4-Bit OR Module:

```
module OR(f2,a,b);
output [3:0] f2;
input [3:0] a,b;
or g1(f2[0],a[0],b[0]);
or g2(f2[1],a[1],b[1]);
or g3(f2[2],a[2],b[2]);
or g4(f2[3],a[3],b[3]);
endmodule
```

### 4-Bit XOR Module:

```
module XOR(f3,a,b);
output [3:0] f3;
input [3:0] a,b;
xor g1(f3[0],a[0],b[0]);
xor g2(f3[1],a[1],b[1]);
xor g3(f3[2],a[2],b[2]);
xor g4(f3[3],a[3],b[3]);
endmodule
```

**4-Bit NOT Module:**

```
module NOT(f4,a);
output [3:0] f4;
input [3:0] a;
not g1(f4[0],a[0]);
not g2(f4[1],a[1]);
not g3(f4[2],a[2]);
not g4(f4[3],a[3]);
endmodule
```

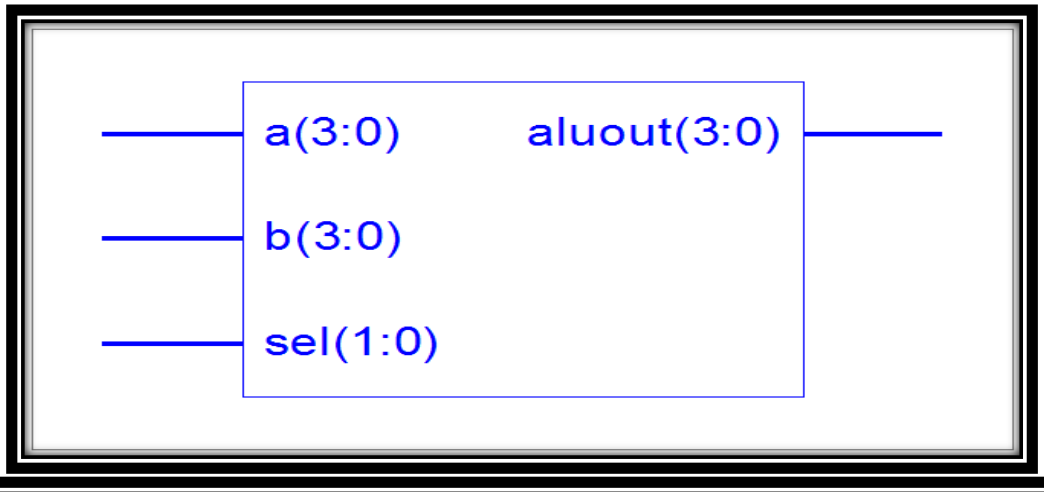
**Common Bus Architecture:**

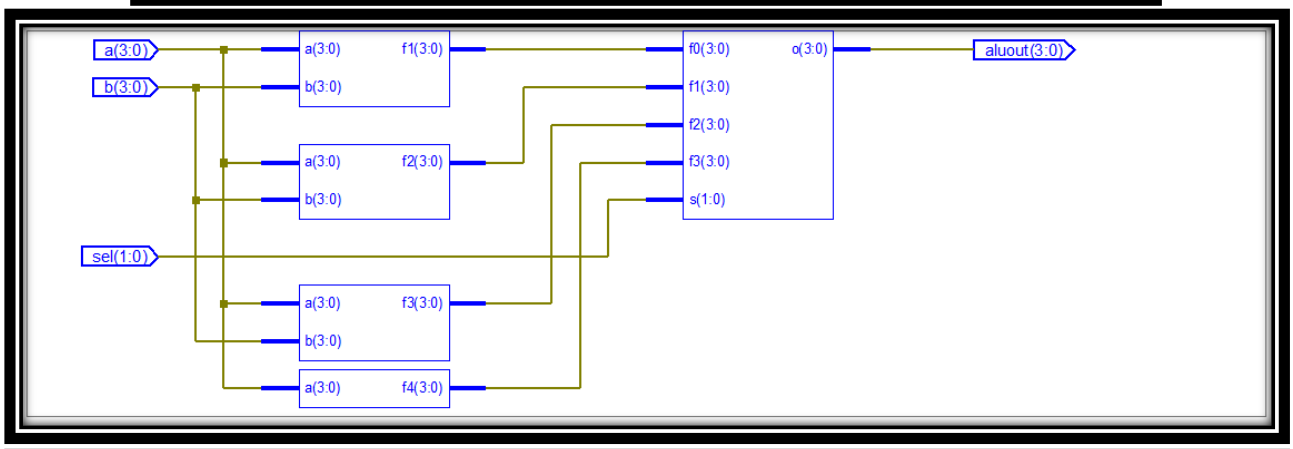
```
module COMBUS(o,f0,f1,f2,f3,s);
input[3:0] f0,f1,f2,f3;
input [1:0] s;
output [3:0] o;
MUX m1(o[0],f0[0],f1[0],f2[0],f3[0],s[0],s[1]);
MUX m2(o[1],f0[1],f1[1],f2[1],f3[1],s[0],s[1]);
MUX m3(o[2],f0[2],f1[2],f2[2],f3[2],s[0],s[1]);
MUX m4(o[3],f0[3],f1[3],f2[3],f3[3],s[0],s[1]);
Endmodule
```

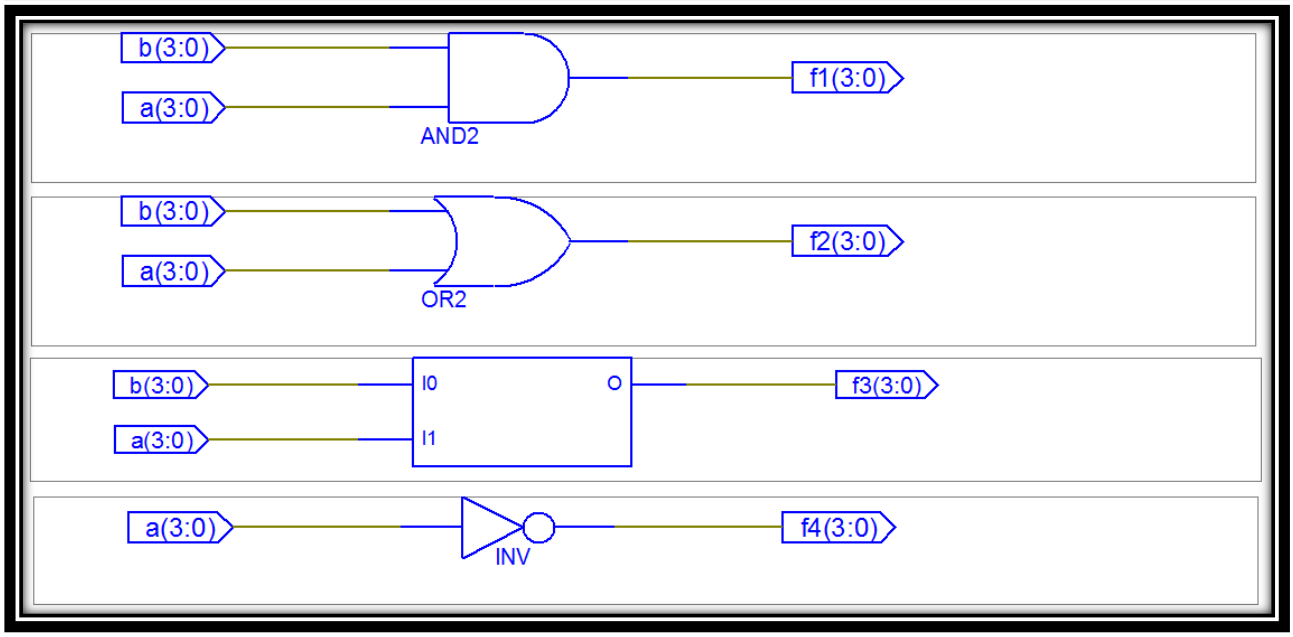
**Arithmetic Control Unit:**

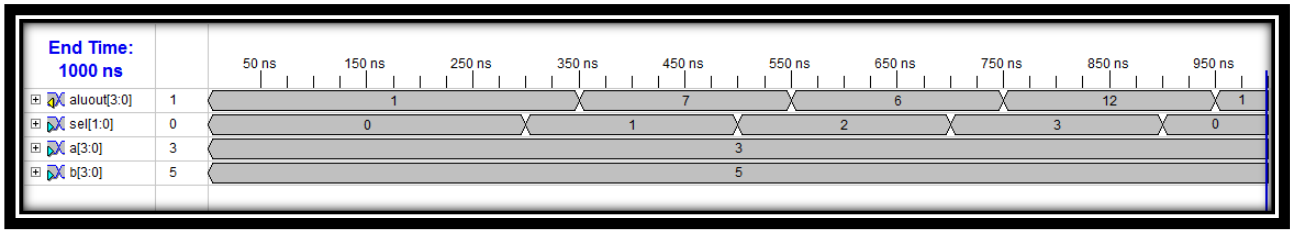
```
module ALU(aluout,sel,a,b);
output [3:0] aluout;
input [3:0] a,b;
input [1:0] sel;
wire [3:0] w1,w2,w3,w4;
COMBUS M(aluout,w1,w2,w3,w4,sel);
AND A(w1,a,b);
OR O(w2,a,b);
XOR X(w3,a,b);
NOT N(w4,a);
Endmodule
```

## RTL Schematics:



# Lab Assignment No. 4:
## Universal Shift Register Structural Modeling

**D Flipflop:**

```
module DFF(out,d,clk,reset);
output out;
input d,clk,reset;
assign out = reset?0:clk?d:out;
endmodule
```

**Multiplexer:**

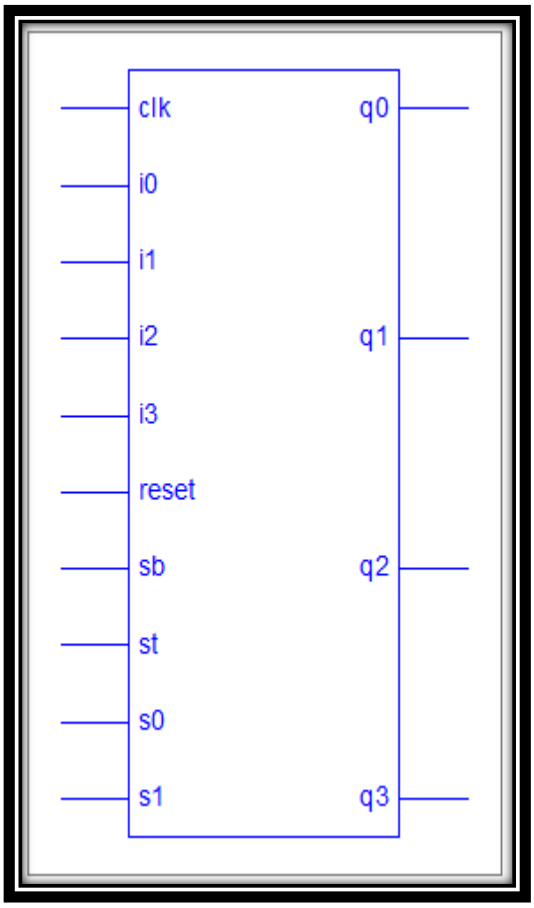
```
module MUX_4_1(out,i0,i1,i2,i3,s0,s1);
output out;
input i0,i1,i2,i3,s0,s1;
wire w1,w2,w3,w4,w5,w6;
not g1(w1,s0);
not g2(w2,s1);
and g3(w3,i0,w1,w2);
and g4(w4,i1,s0,w2);
and g5(w5,i2,w1,s1);
and g6(w6,i3,s1,s0);
or g7(out,w3,w4,w5,w6);
endmodule
```

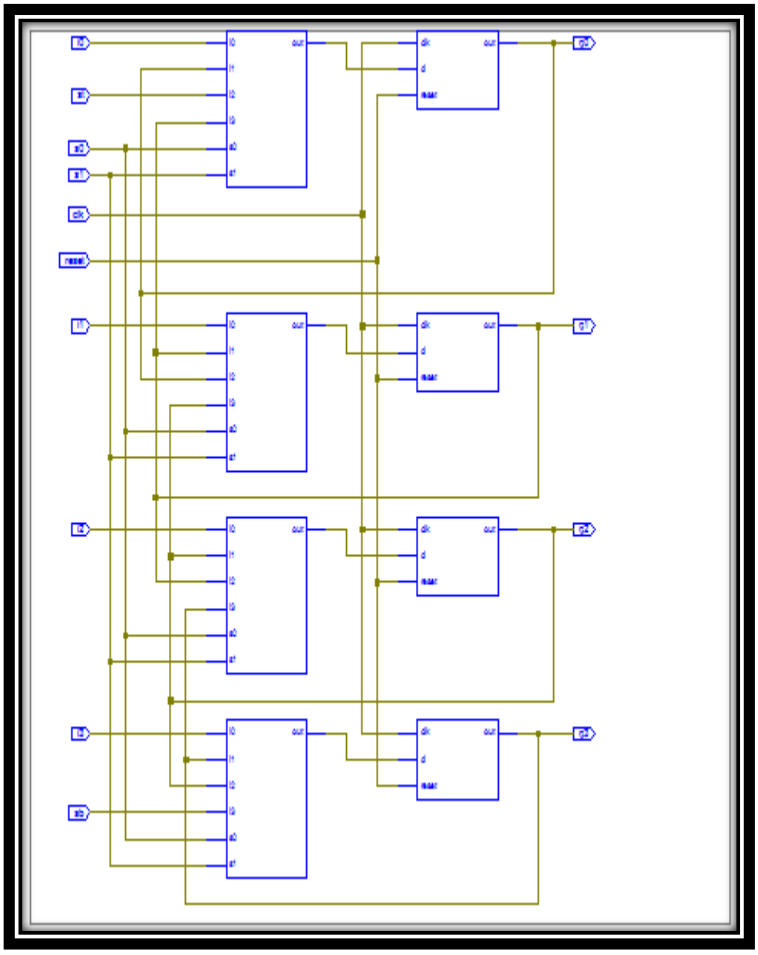
**Main Module USR:**

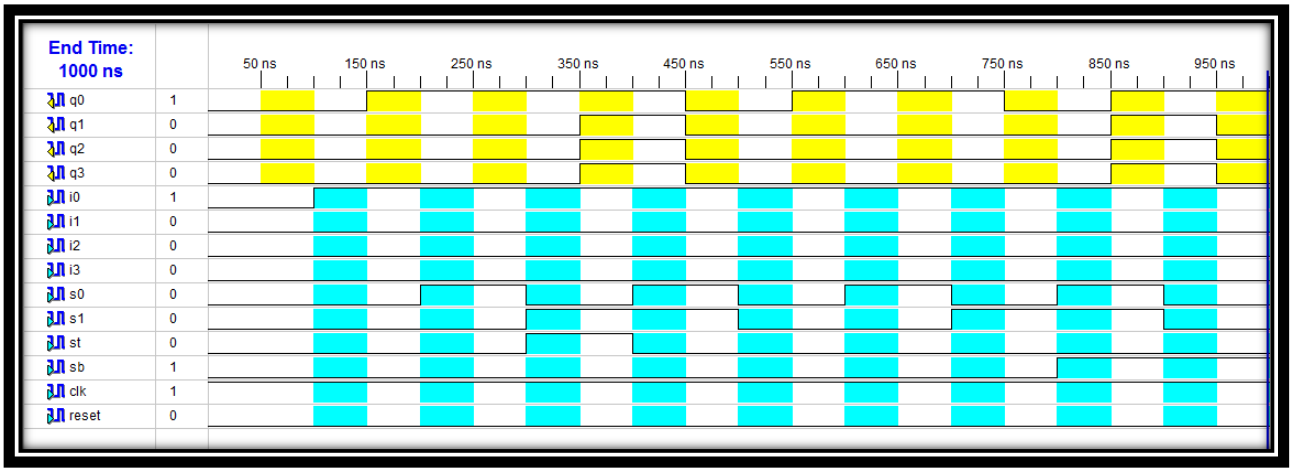
```
module U_S_R(q0,q1,q2,q3,i0,i1,i2,i3,s0,s1,st,sb,clk,reset);
output q0,q1,q2,q3;
input i0,i1,i2,i3,s0,s1,st,sb,clk,reset;
wire w1,w2,w3,w4;

MUX_4_1 M1(w1,i0,q0,st,q1,s0,s1);
MUX_4_1 M2(w2,i1,q1,q0,q2,s0,s1);
MUX_4_1 M3(w3,i2,q2,q1,q3,s0,s1);
MUX_4_1 M4(w4,i3,q3,q2,sb,s0,s1);
```

```
DFF D1(q0,w1,clk,reset);
DFF D2(q1,w2,clk,reset);
DFF D3(q2,w3,clk,reset);
DFF D4(q3,w4,clk,reset);
Endmodule
```

## RTL Schematics:





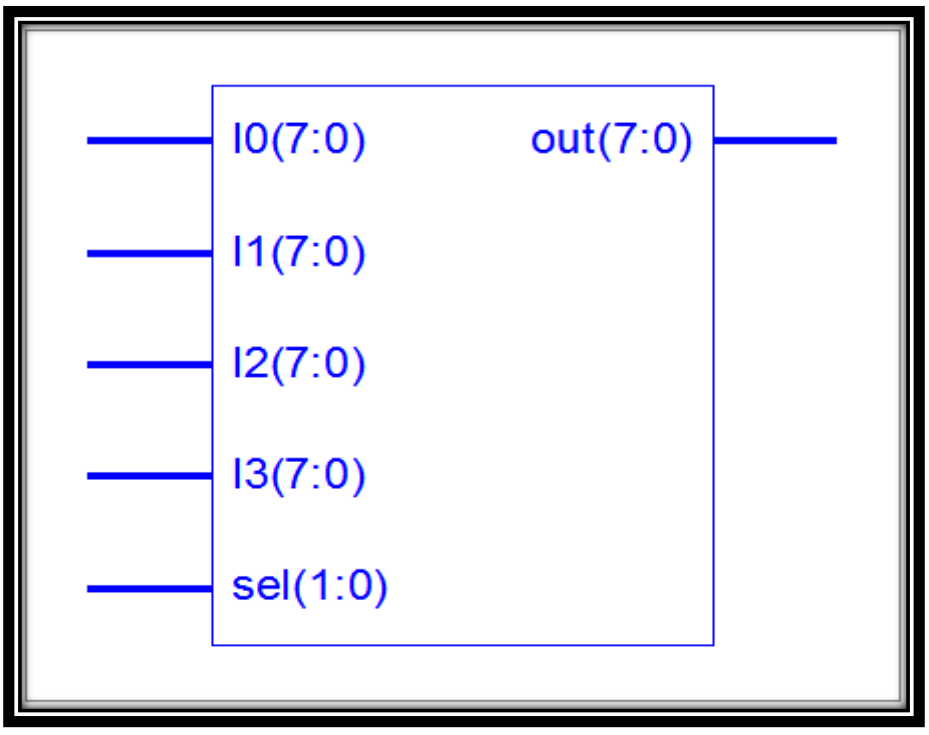
# Lab Assignment No. 5:

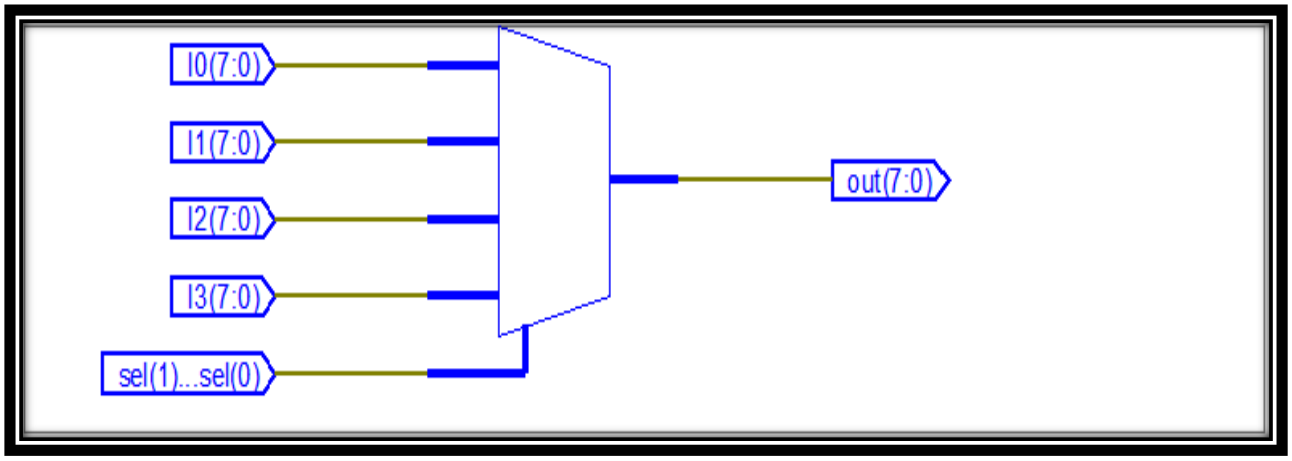
## Behavioral Modeling
## Common Bus Architecture , Decoder , Encoder

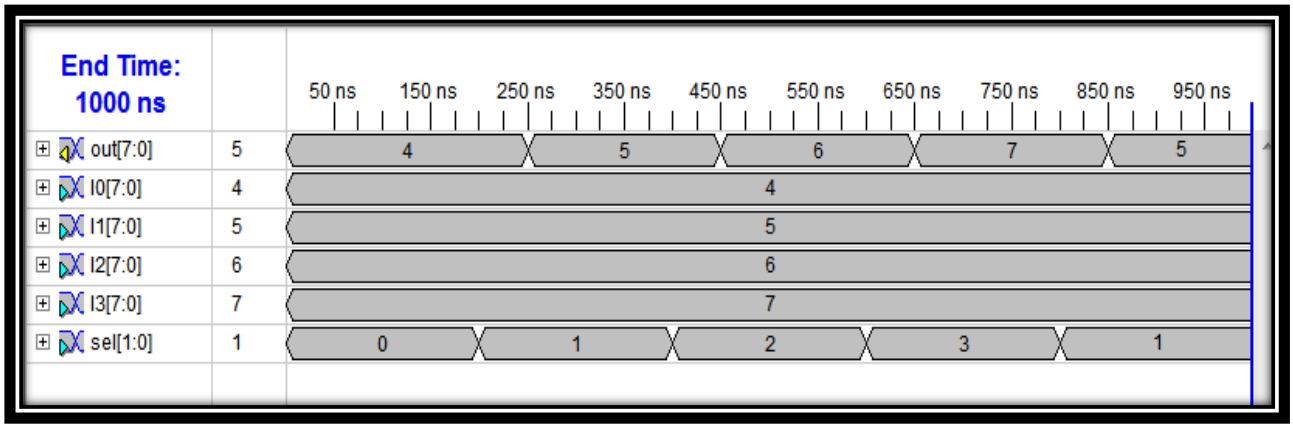
### Common Bus Architecture:

```
module com_bus(out,I0,I1,I2,I3,sel);
parameter bit = 8;
output [bit-1:0] out;
input [bit-1:0] I0,I1,I2,I3;
input [1:0] sel;
reg [bit-1:0] out;
always @ (I0,I1,I2,I3,sel)
        case (sel)
        0:      out = I0;
        1:      out = I1;
        2:      out = I2;
        3:      out = I3;
        default: out = 8'bz;
        endcase
endmodule
```

### RTL Schematics:



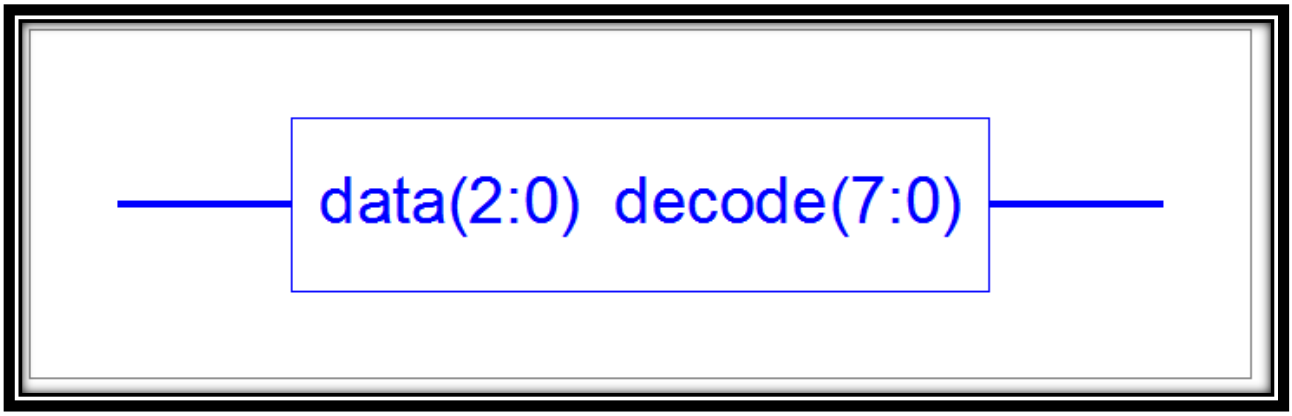

I0(7:0)
I1(7:0)
I2(7:0)
I3(7:0)
sel(1)...sel(0)
out(7:0)

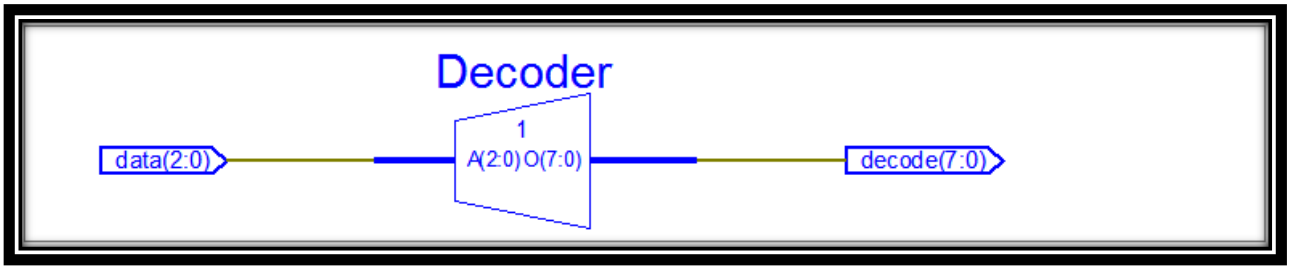

End Time:
1000 ns
50 ns
150 ns
250 ns
350 ns
450 ns
550 ns
650 ns
750 ns
850 ns
950 ns
out[7:0] 5 4 5 6 7 5
I0[7:0] 4 4
I1[7:0] 5 5
I2[7:0] 6 6
I3[7:0] 7 7
sel[1:0] 1 0 1 2 3 1

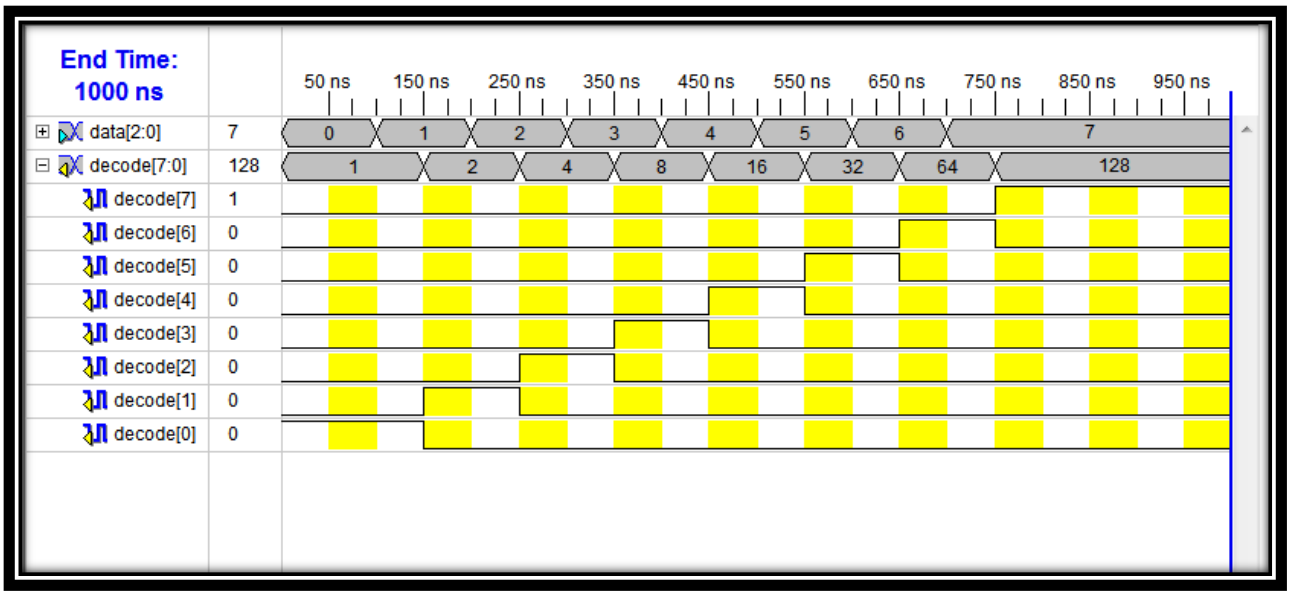
**Decoder:**

```
module decoder(data,decode);
output[7:0] decode;
input[2:0] data;
reg [7:0] decode;
always @ (data)
        case (data)
        0:      decode = 8'b00000001;
        1:      decode = 8'b00000010;
        2:      decode = 8'b00000100;
        3:      decode = 8'b00001000;
        4:      decode = 8'b00010000;
        5:      decode = 8'b00100000;
        6:      decode = 8'b01000000;
        7:      decode = 8'b10000000;
        endcase
endmodule
```

**RTL Schematics:**



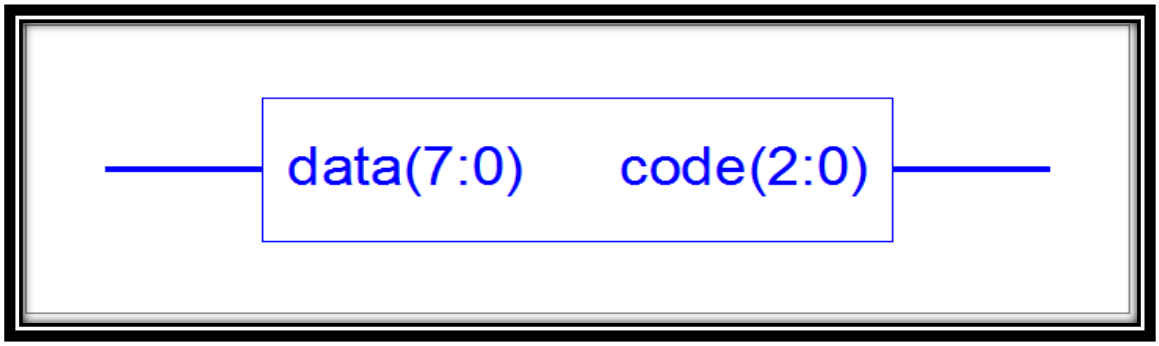



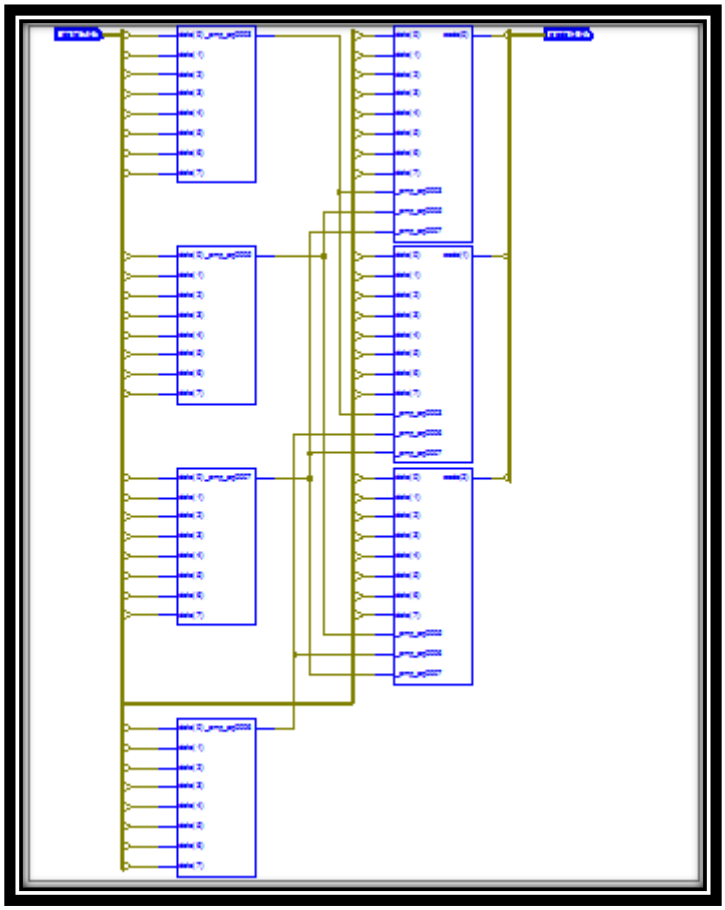


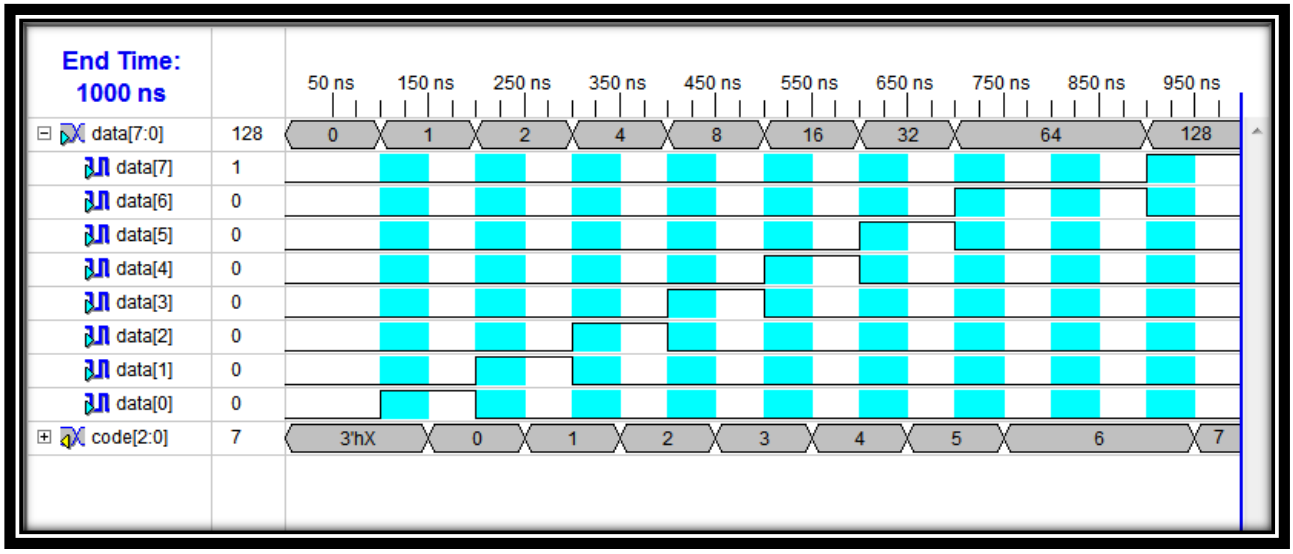

**Encoder:**

```
module encoder(code,data);
output[2:0] code;
input[7:0] data;
reg [2:0] code;
always @ (data)
        case (data)
        8'b00000001:        code = 0;
        8'b00000010:        code = 1;
        8'b00000100:        code = 2;
        8'b00001000:        code = 3;
        8'b00010000:        code = 4;
        8'b00100000:        code = 5;
        8'b01000000:        code = 6;
        8'b10000000:        code = 7;
        default: code = 3'bx;
        endcase
endmodule
```

RTL Schematics:





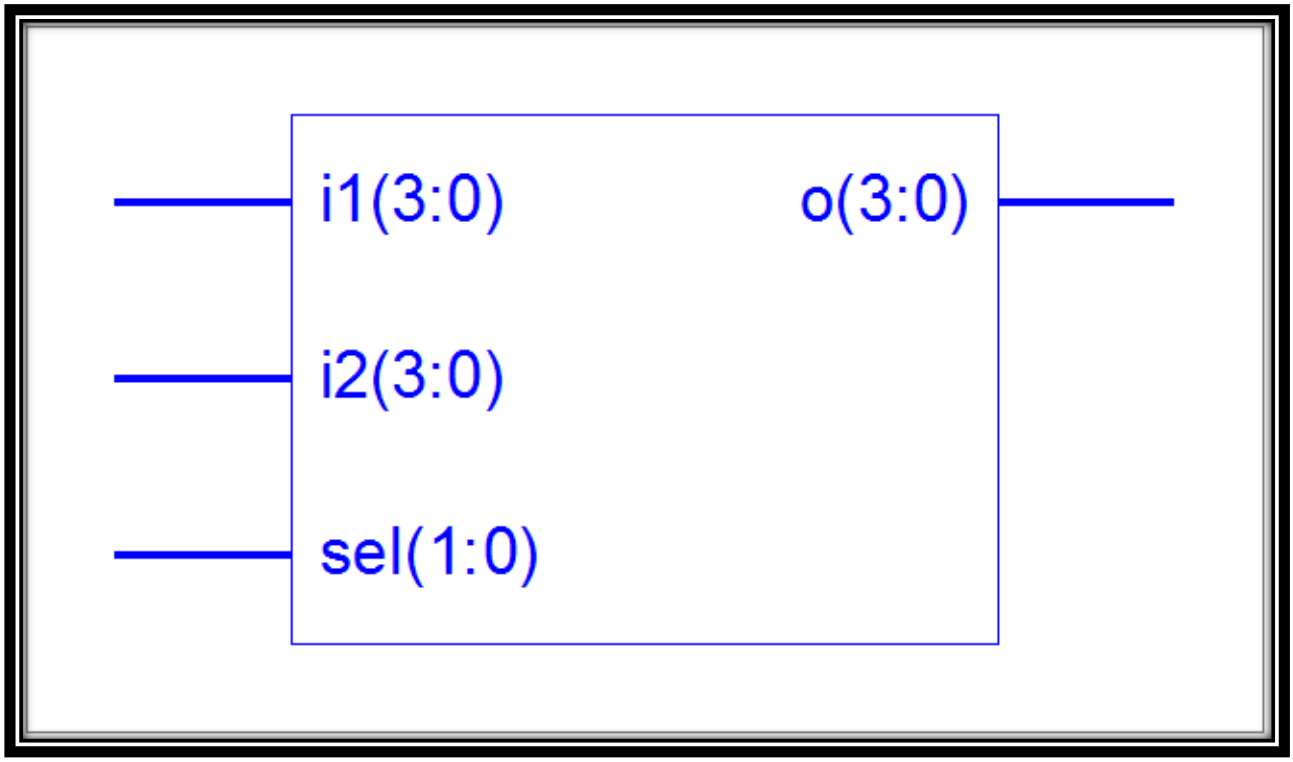
# Lab Assignment No. 6:

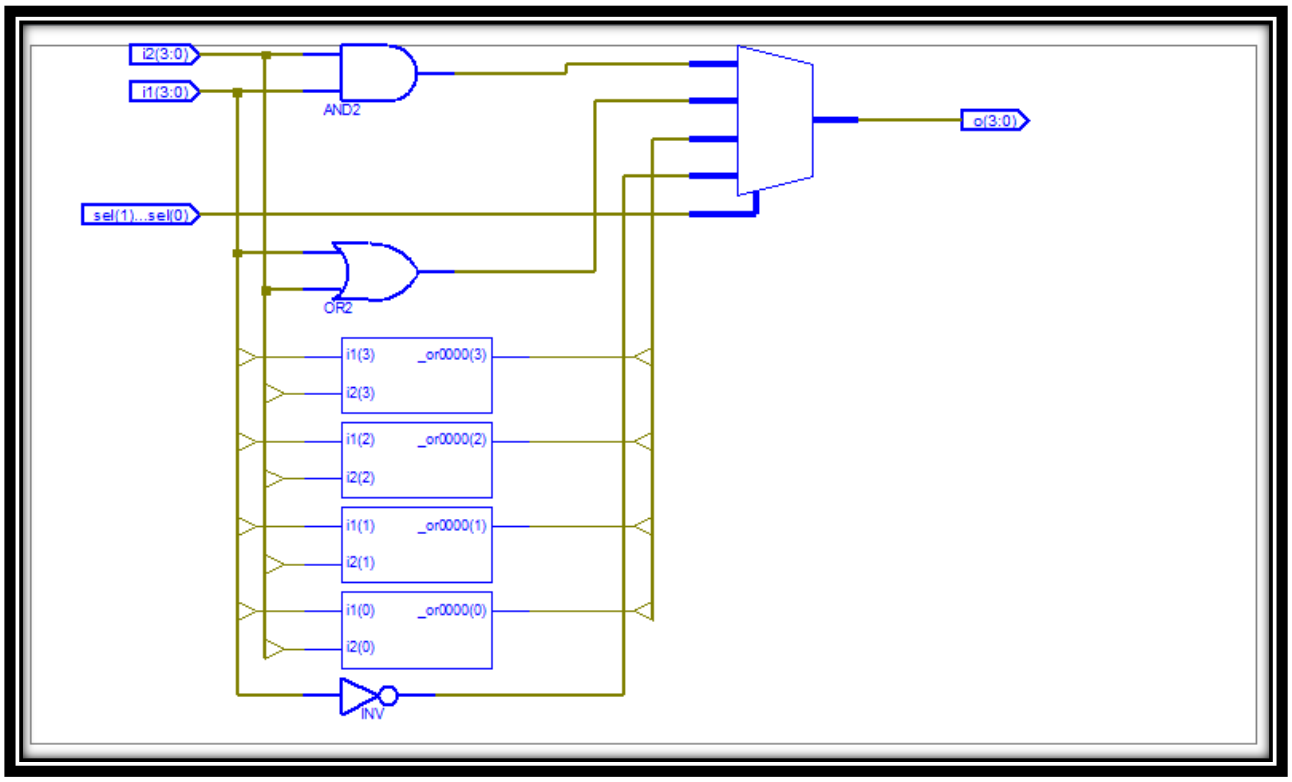
## Behavioral Modeling
## ALU , BCD-7SEG, Accumulator Register

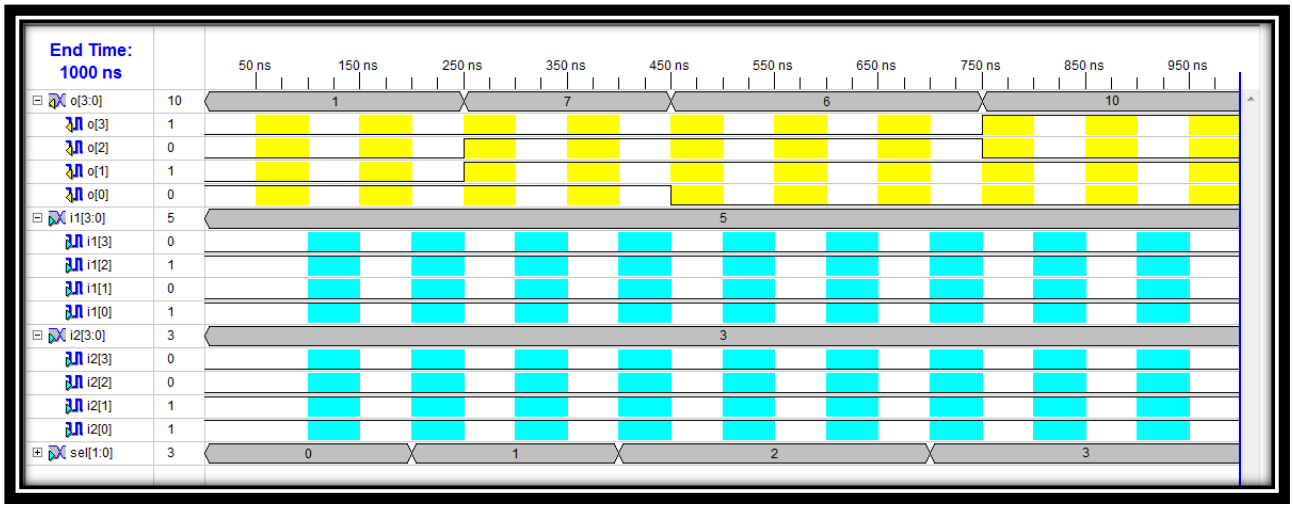
### Arithmetic Control Unit:

```
module logic_unit(o,i1,i2,sel);
output[3:0] o;
input [3:0] i1,i2;
input [1:0] sel;
reg [3:0] o;
always @ (i1 or i2 or sel)
        case (sel)
        0: o = i1 & i2;
        1: o = i2 | i1;
        2:      o = (~i1 & i2)|(~i2 & i1);
        3: o = ~i1;
        endcase
endmodule
```

### RTL Schematics:



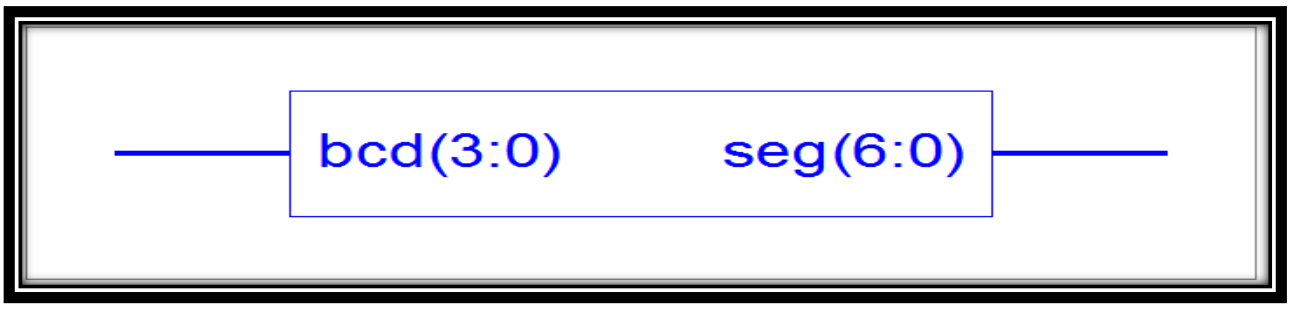

i2(3:0)
i1(3:0)
AND2
o(3:0)
sel(1)..sel(0)
OR2
i1(3)
_or0000(3)
i2(3)
i1(2)
_or0000(2)
i2(2)
i1(1)
_or0000(1)
i2(1)
i1(0)
_or0000(0)
i2(0)
INV

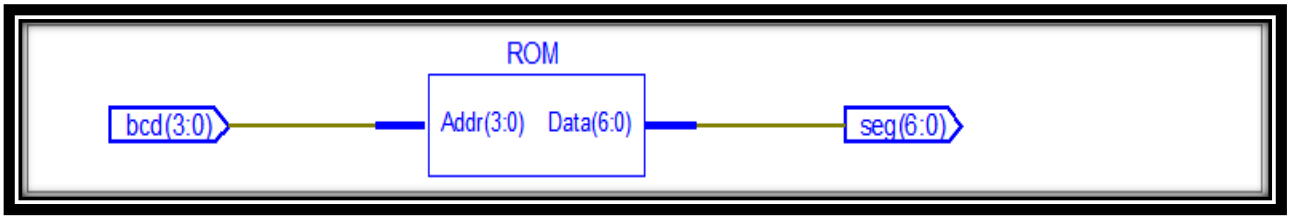

End Time:
1000 ns
50 ns
150 ns
250 ns
350 ns
450 ns
550 ns
650 ns
750 ns
850 ns
950 ns
o[3:0] 10
o[3] 1
o[2] 0
o[1] 1
o[0] 0
i1[3:0] 5
i1[3] 0
i1[2] 1
i1[1] 0
i1[0] 1
i2[3:0] 3
i2[3] 0
i2[2] 0
i2[1] 1
i2[0] 1
sel[1:0] 3
1
7
6
10
5
3
0
1
2
3

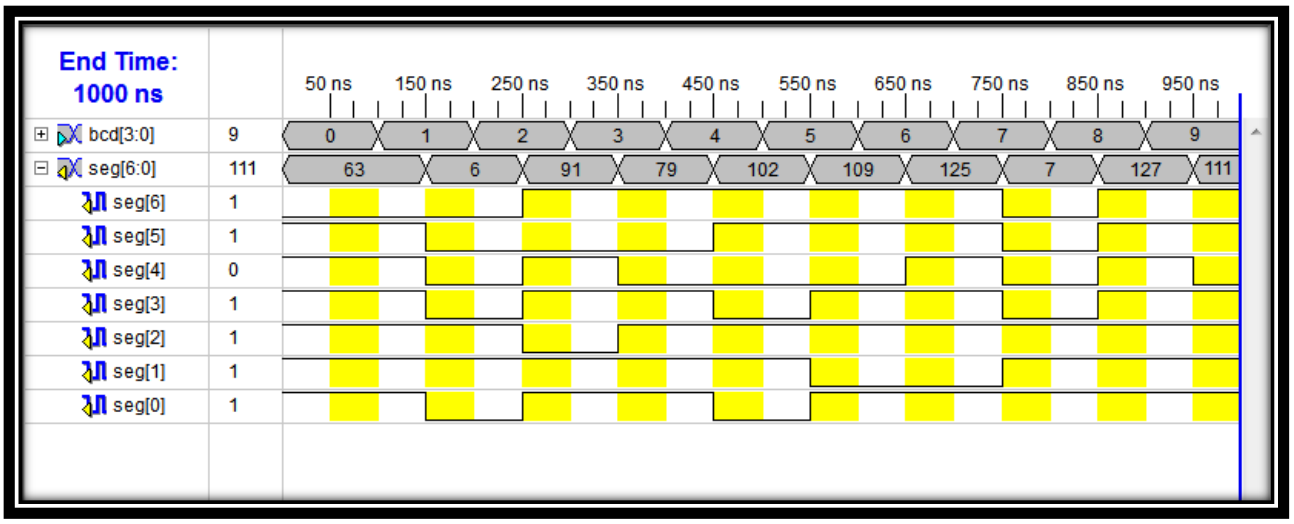
**BCD To 7Seg Display:**

```
module bcd_7seg(seg,bcd);
output[6:0] seg;
input [3:0] bcd;
reg [6:0] seg;
always @ (bcd)
        case (bcd)
        4'b0000: seg =  7'b0111111;
        4'b0001: seg =  7'b0000110;
        4'b0010: seg =  7'b1011011;
        4'b0011: seg =  7'b1001111;
        4'b0100: seg =  7'b1100110;
        4'b0101: seg =  7'b1101101;
        4'b0110: seg =  7'b1111101;
        4'b0111: seg =  7'b0000111;
        4'b1000: seg =  7'b1111111;
        4'b1001: seg =  7'b1101111;
        default: seg =  7'bx;
        endcase
endmodule
```

**RTL Schematics:**



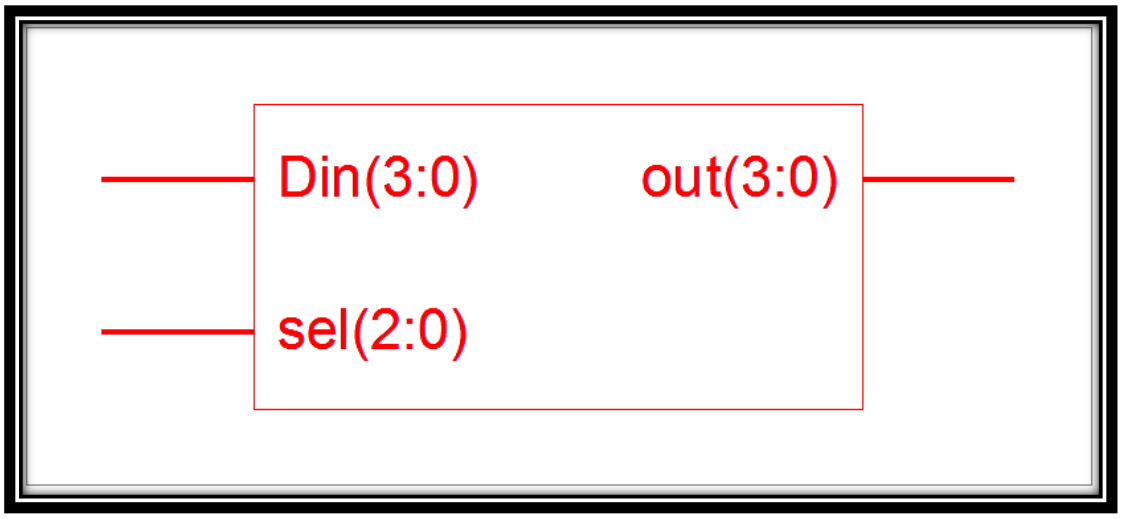



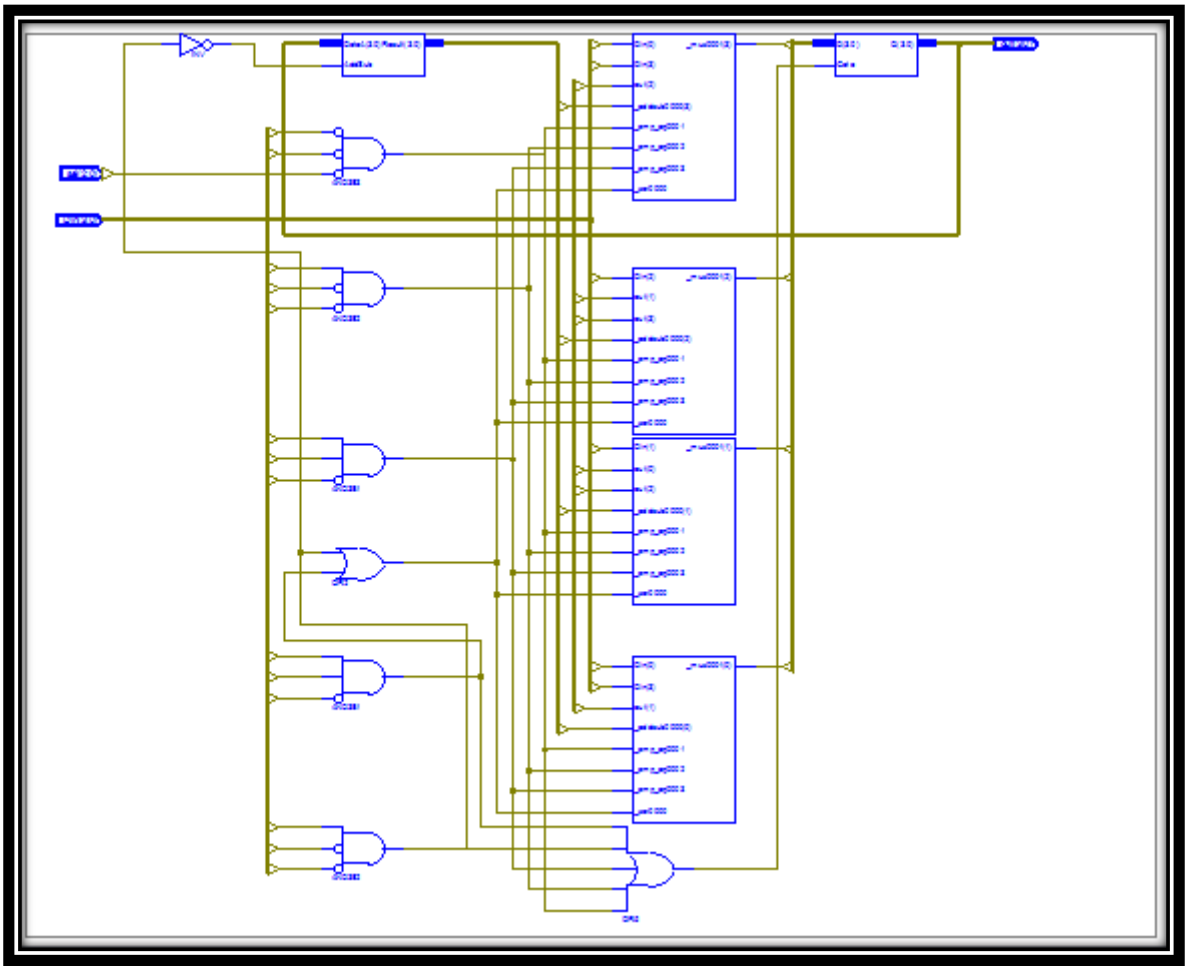


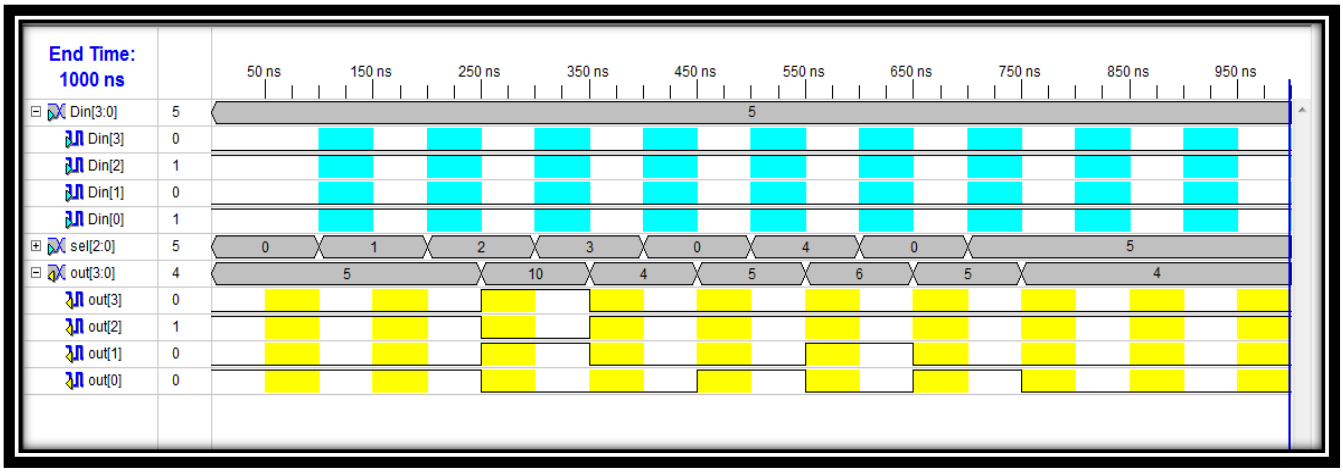

**Accumulator Register:**

```
module acc_reg(out,Din,sel);
output [3:0] out;
input [3:0] Din;
input [2:0] sel;
reg[3:0] out;
always @ (sel)
        case (sel)
        0:      out = Din;
        1:      out = out;
        2: out = {Din[0],out[3:1]};
        3: out = {out[2:0],Din[3]};
        4: out = out + 1;
        5: out = out - 1;
        endcase
endmodule
```

RTL Schematics:




End Time:
1000 ns
50 ns
150 ns
250 ns
350 ns
450 ns
550 ns
650 ns
750 ns
850 ns
950 ns
Din[3:0] 5
Din[3] 0
Din[2] 1
Din[1] 0
Din[0] 1
sel[2:0] 5
out[3:0] 4
out[3] 0
out[2] 1
out[1] 0
out[0] 0
5
0 1 2 3 0 4 0 5
5 10 4 5 6 5 4

# Lab Assignment No. 7:
## Data-Path of RISC Architecture

**ALU:**

```
module logic_unit(zf,o,i1,i2,sel);
output[7:0] o;
output zf;
input [7:0] i1,i2;
input [3:0] sel;
reg [7:0] o;
assign zf = ~|o;
always @ (i1 or i2 or sel)
        case (sel)
        0: o = i1 + i2;
        1: o = i2 - i1;
        2:o = i1 & i2;
        3: o = ~i2;
        default o = 8'b0;
        endcase
endmodule
```

**5-1Multiplexer:**

```
module mul_5(o,i0,i1,i2,i3,i4,sel);
parameter bit_size = 8;
output[bit_size-1:0] o;
input [2:0] sel;
input [bit_size-1:0] i0,i1,i2,i3,i4;
reg [bit_size-1:0] o;
always @ (sel or i0,i1,i2,i3,i4)
        case(sel)
        0: o = i0;
        1:      o = i1;
        2: o = i2;
        3: o = i3;
        4: o = i4;
```

```
        default : o = 8'bz;
        endcase
endmodule
```

### 3-1 Multiplexer

```
module mux_3(o,i0,i1,i2,sel);
output[7:0] o;
input[7:0] i0,i1,i2;
input [1:0] sel;
reg [7:0] o;
always @ (sel or i0 or i1 or i2)
        case (sel)
        0: o = i0;
        1: o = i1;
        2: o = i2;
        default : o = 8'bz;
        endcase
endmodule
```

### 8-Bit Register:

```
module regs(o,i,sel,clk,rst);
output[7:0] o;
input [7:0] i;
input sel,clk,rst;
reg [7:0] o;
always @ (posedge clk , negedge rst)
assign o = !rst? 0:     sel ? i:8'bz;
endmodule
```

**D-Flipflop:**

```
module regz(o,i,sel,clk,rst);
output o;
input  i;
input sel,clk,rst;
reg  o;
always @ (posedge clk , negedge rst)
assign o = !rst? 0:     sel ? i:1'bz;
endmodule
```

**8-Bit Program Counter:**

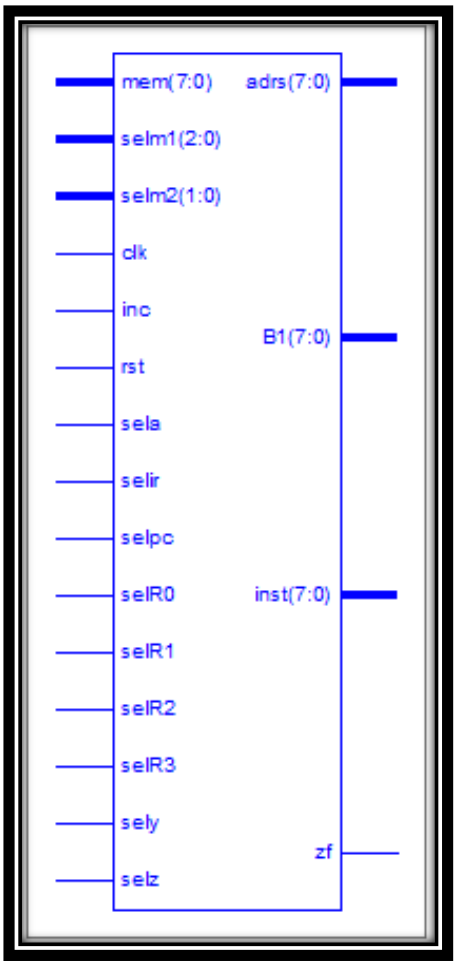
```
module p_count(o,in,inc,sel,clk,rst);
output [7:0] o;
input [7:0] in;
input sel,inc,clk,rst;
reg[7:0] o;
always @ (posedge clk, negedge rst)
        if (rst == 0)
        o = 8'b0;
        else if (sel == 1)
        o = in;
        else if (inc)
        o = o + 1;
endmodule
```

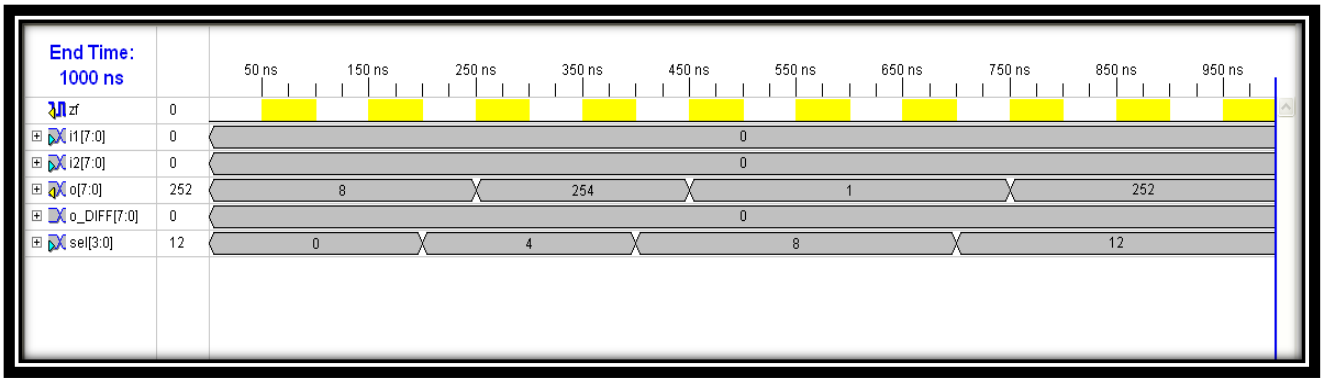
**Main Module:**

```
module process(inst,zf,adrs,selR0,selR1,selR2,selR3,selpc,inc,selir,sela,sely,selz,
selm1,selm2,B1,mem,clk,rst);
output [7:0] inst,adrs,B1;
output zf;
input [7:0] mem;
input [2:0] selm1;
input [1:0] selm2;
input selR0,selR1,selR2,selR3,selpc,inc,selir,sela,sely,selz,clk,rst;
wire [7:0] B2,w0,w1,w2,w3,w4,alu1,mo;
```

```
wire [3:0]      opcode = inst[7:4];
wire zfin;
regs R0(w0,B2,selR0,clk,rst);
regs R1(w1,B2,selR1,clk,rst);
regs R2(w2,B2,selR2,clk,rst);
regs R3(w3,B2,selR3,clk,rst);
regs IR(inst,B2,selir,clk,rst);
regz RZ(zf,zfin,selz,clk,rst);
p_count PC(w4,B2,inc,selpc,clk,rst);
mul_5  MUX1(B1,w0,w1,w2,w3,w4,selm1);
regs RY(alu1,B2,sely,clk,rst);
logic_unit ALU(zfin,m0,alu1,B1,opcode);
mux_3 MUX2(B2,mo,B1,mem,selm2);
regs RA(adrs,B2,sela,clk,rst);
endmodule
```

**RTL Schematics:**




End Time:
1000 ns
50 ns
150 ns
250 ns
350 ns
450 ns
550 ns
650 ns
750 ns
850 ns
950 ns
zf
0
i1[7:0]
0
0
i2[7:0]
0
0
o[7:0]
252
8
254
1
252
o_DIFF[7:0]
0
0
sel[3:0]
12
0
4
8
12

# Lab Assignment No. 8:

## Control Unit Of Risc

### Control Unit:

```
module c_unit(R0,R1,R2,R3,Load_pc,in_pc,sel_mux1,sel_mux2,L_ir,L_add,Load_y,Load_z,
wr,ins,zero,clk,rst);

parameter a=0,b=1,c=2,d=3,e=4,f=5,g=6,h=7,i=8,j=9,k=10,l=11;
parameter nop=0,add=1,sub=2,AND=3,NOT=4;
parameter RD=5,WR=6,BR=7,BRZ=8;

output R0,R1,R2,R3;
output Load_pc,in_pc;
output [c:0] sel_mux1;
output L_ir,L_add,Load_y,Load_z;
output [b:0] sel_mux2;
output wr;

input [h:0] ins;
input zero,clk,rst;

reg [d:0] st,nst;
reg R0,R1,R2,R3,Load_pc,in_pc;
reg L_ir,L_add,Load_y;
reg sel_alu, sel_b1,sel_mem;
reg sel_r0,sel_r1,sel_r2,sel_r3,sel_pc;
reg Load_z,wr;
reg err_flag;

wire [e-1:0] opcode=ins[i-1:i-e];
wire [c-1:0] scr=ins[c+c-1:c];
wire [c-1:0] dest=ins[c-1:0];

assign sel_mux1=sel_r0?0:sel_r1?1:sel_r2?2:sel_r3?3:sel_pc?4:3'bx;
assign sel_mux2=sel_alu?0:sel_b1?1:sel_mem?2:2'bx;
```

```
always@(posedge clk or negedge rst)begin: state_transitions

if(rst==0)st<=a;else st<=nst;end
always@(st or opcode or scr or dest or zero)begin: output_next_state
sel_r0=0;sel_r1=0;sel_r2=0;sel_r3=0;sel_pc=0;
R0=0;R1=0;R2=0;R3=0;Load_pc=0;
L_ir=0;L_add=0;Load_y=0;Load_z=0;
in_pc=0;sel_alu=0; sel_b1=0;sel_mem=0;wr=0;err_flag=0;nst=st;
case(st)
a: nst=b;
b: begin
  nst=c;
        sel_pc=1;
        sel_b1=1;
        L_add=1;end
c: begin
  nst=d;
        sel_mem=1;
        in_pc=1;
        L_ir=1;
        end
d: case(opcode)
  nop:nst=c;
        add,sub,AND:begin
        nst=e;
        sel_b1=1;
        Load_y=1;
        case(scr)
        a: sel_r0=1;
        b: sel_r1=1;
        c: sel_r2=1;
        d: sel_r3=1;
        default: err_flag=1;
  endcase
        end
```

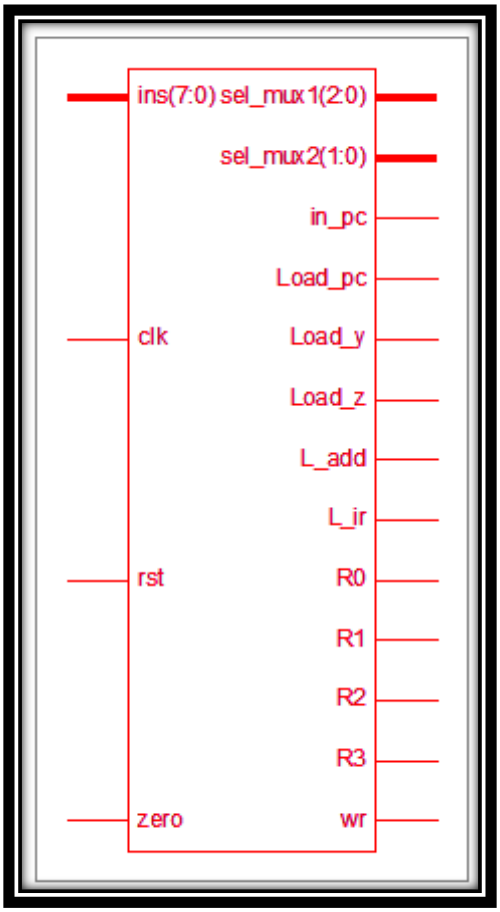
```
        NOT:begin
        nst=b;
        Load_z=1;
        sel_b1=1;
        sel_alu=1;
        case(scr)
        a: sel_r0=1;
        b: sel_r1=1;
        c: sel_r2=1;
        d: sel_r3=1;
        default: err_flag=1;
  endcase
        case(dest)
        a: R0=1;
        b: R1=1;
        c: R2=1;
        d: R3=1;
        default: err_flag=1;
  endcase
        end
        RD:begin
        nst=f;
        sel_pc=1;
        sel_b1=1;
        L_add=1;
        end
        WR:begin
        nst=h;
        sel_pc=1;
        sel_b1=1;
        L_add=1;
        end
```

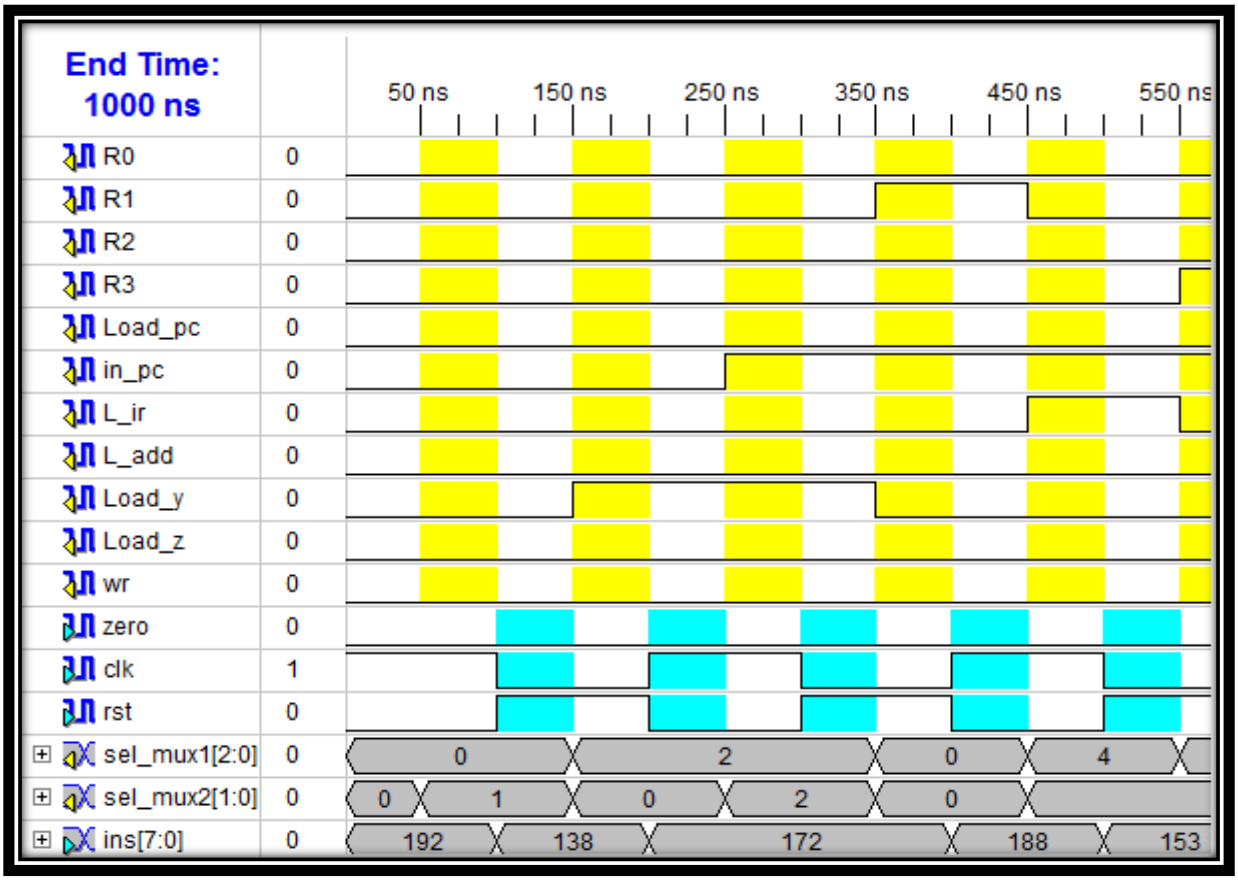
```
        BR:begin
        nst=j;
        sel_pc=1;
        sel_b1=1;
        L_add=1;
        end
        BRZ:if(zero==1)begin
        nst=j;
        sel_pc=1;
        sel_b1=1;
        L_add=1;
        end
        else
        begin
        nst=b;
        in_pc=1;
        end
        default:nst=1;
        endcase
e: begin
  nst=b;
        Load_z=1;
        sel_alu=1;
        case(dest)
        a: begin sel_r0=1;
          R0=1;end
        b: begin sel_r1=1;
          R1=1;end
        c: begin sel_r2=1;
          R2=1;end
        d: begin sel_r3=1;
          R3=1;end
        default: err_flag=1;
  endcase
        end
```

```
f: begin
  nst=g;
        sel_mem=1;
        L_add=1;
        in_pc=1;
        end
h: begin
  nst=i;
        sel_mem=1;
        L_add=1;
        in_pc=1;
        end
g: begin
  nst=b;
        sel_mem=1;
        case(dest)
        a: R0=1;
        b: R1=1;
        c: R2=1;
        d: R3=1;
        default: err_flag=1;
  endcase
        end
i: begin
  nst=b;
        wr=1;
        case(scr)
        a: sel_r0=1;
        b: sel_r1=1;
        c: sel_r2=1;
        d: sel_r3=1;
        default: err_flag=1;
  endcase
        end
```

```
j: begin
   nst=k;
        sel_mem=1;
        L_add=1;
        end
k: begin
   nst=b;
        sel_mem=1;
        L_add=1;
        end
l: nst=l;
default: nst=a;
endcase
end
endmodule
```

RTL Schematics:



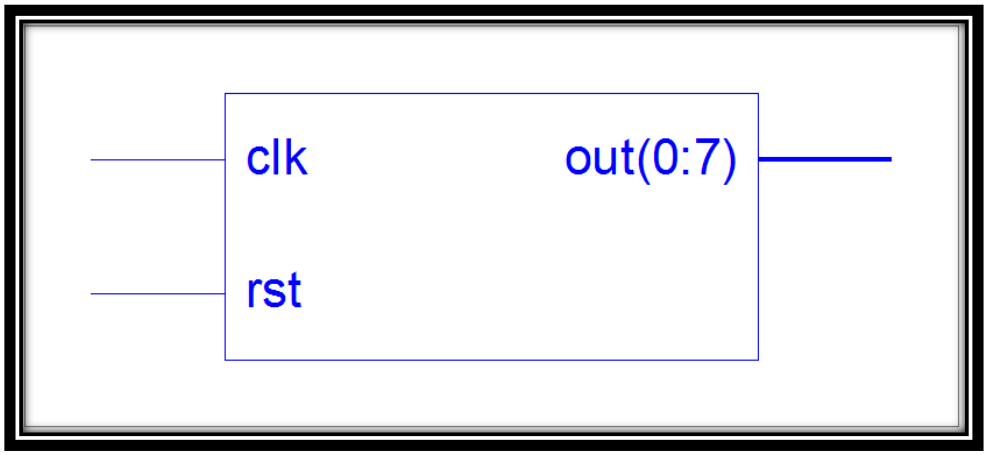



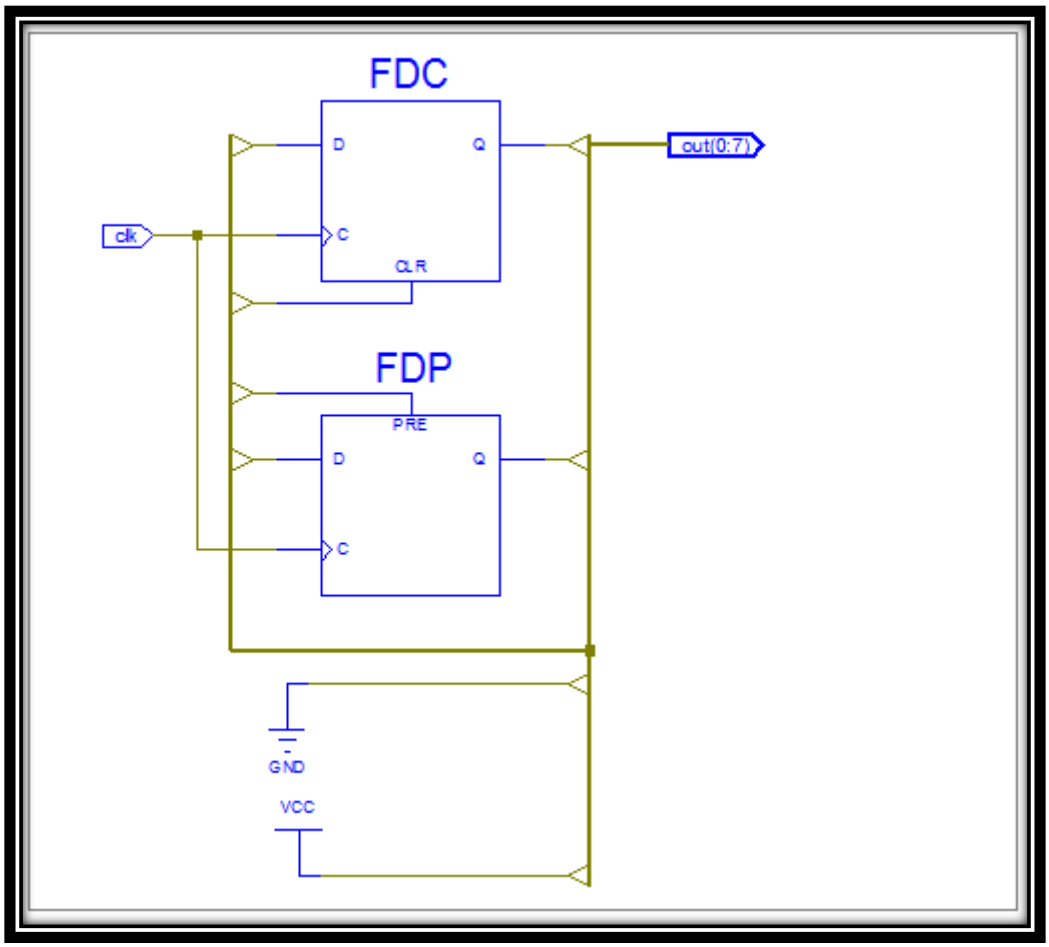
# Lab Assignment No. 9:
## Linear Feedback Shift Register

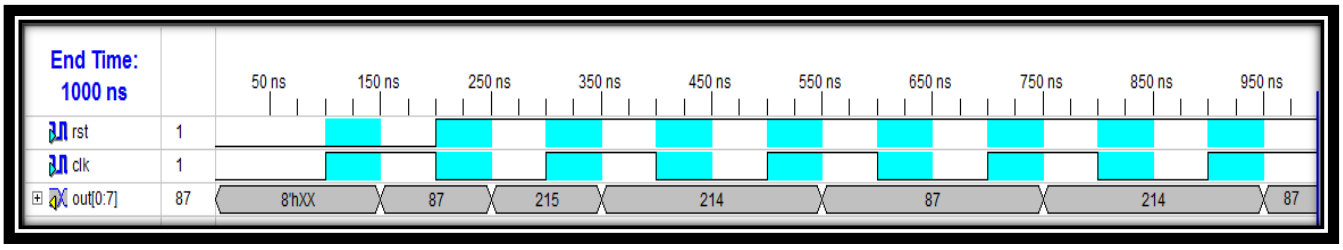
**Linear Feedback Shift Register:**

```
module lfs_reg(out,rst,clk);
parameter L = 7;
parameter [L:0]tap_co = 8'b0111_0111;
parameter [0:L]in_state = 8'b0101_0111;
output[0:L] out;
input rst,clk;
reg[0:L] out;
reg [2:0]k;
always @ (posedge clk or posedge rst)
        if (rst)
        begin
        assign out[0] = out[L];
        for(k = 1; k < L ; k = k + 1)
                if (tap_co[L - k])
                        out[k] = out[k-1]^out[L];
                else
                        out[k] = out[k-1];
        end
        else
        out = 8'b0101_0111;
endmodule
```

**RTL Schematics:**




FDC
D
Q
out(0:7)
clk
C
CLR
FDP
PRE
D
Q
C
GND
VCC


End Time:
1000 ns
50 ns
150 ns
250 ns
350 ns
450 ns
550 ns
650 ns
750 ns
850 ns
950 ns
rst
1
clk
1
out[0:7]
87
8'hXX
87
215
214
87
214
87

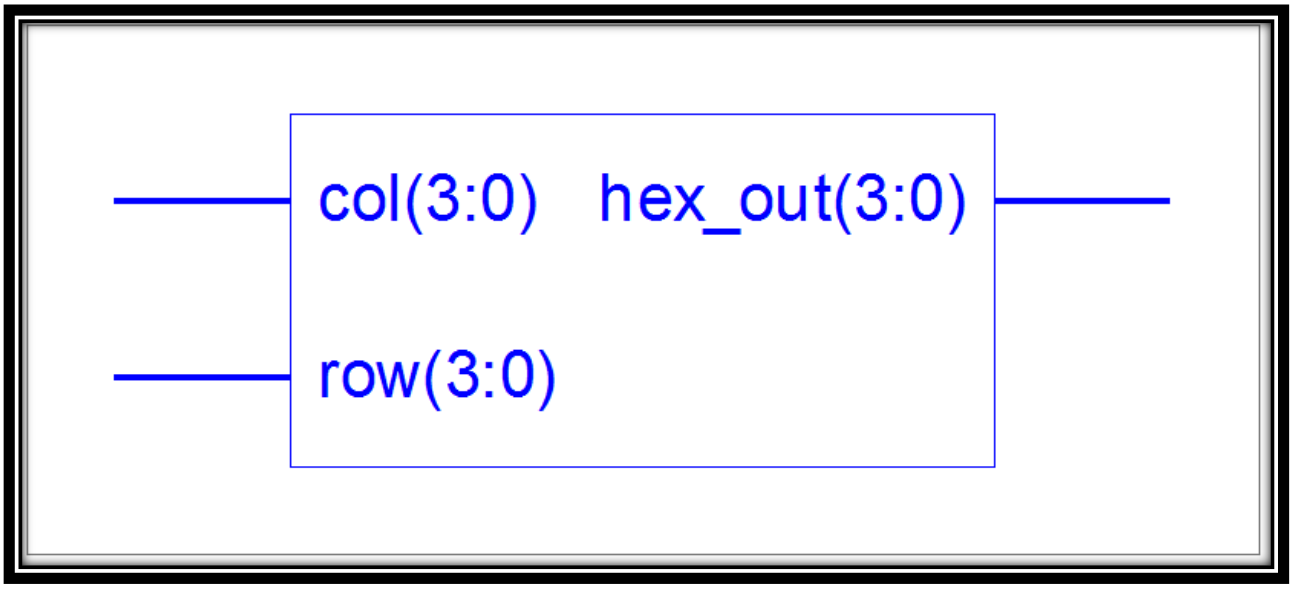
# Lab Assignment No. 10:

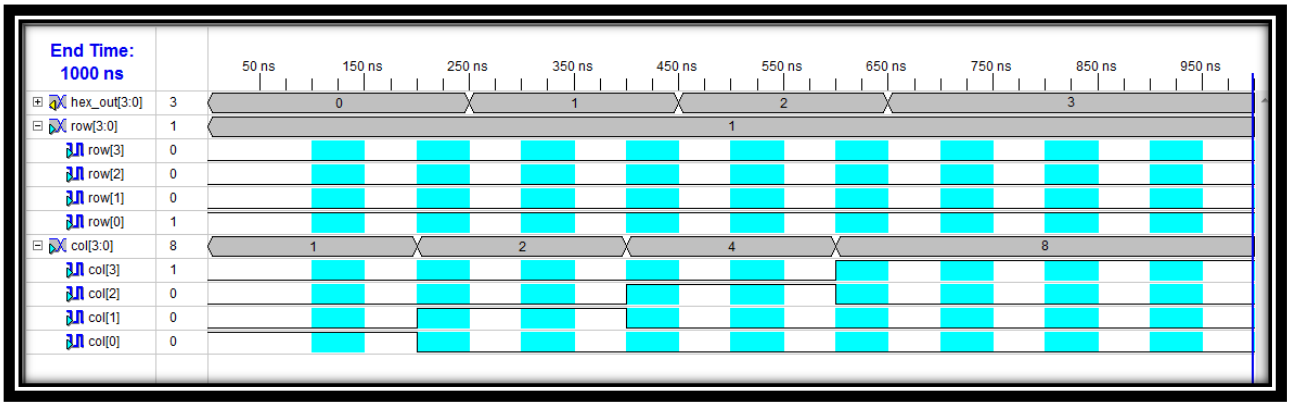
## Hex Keyboard To 4Bit Binary Code

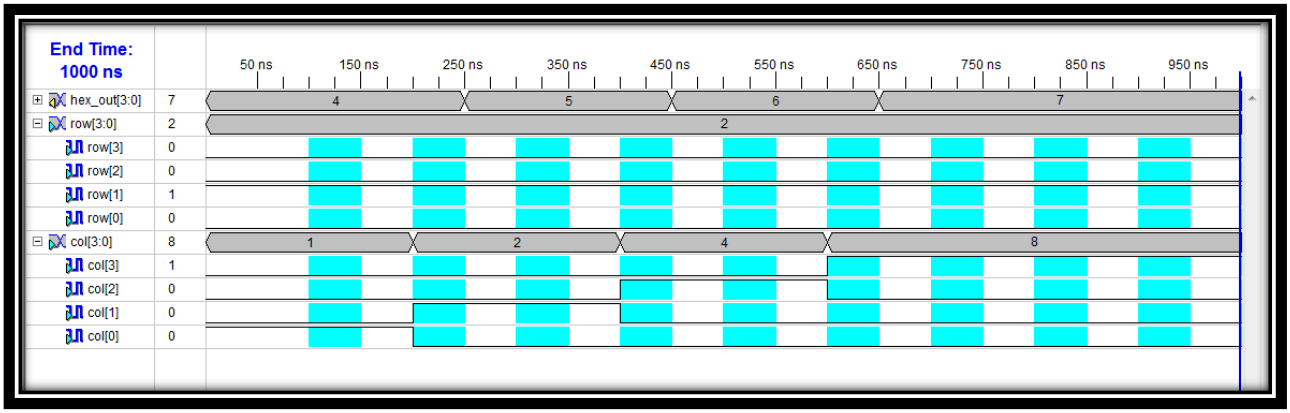
**Hex Keyboard:**

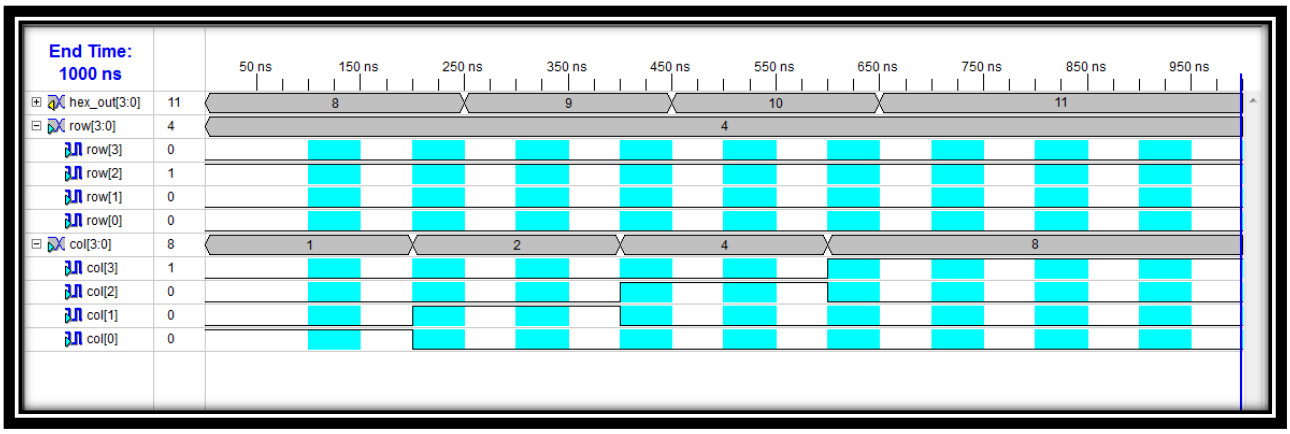
```
module hex_keyboard(hex_out,row,col);
output [3:0] hex_out;
input [3:0]  row,col;
reg[3:0] hex_out;
always @(row,col)
        case ({row,col})
        8'b0001_0001: hex_out = 0;
        8'b0001_0010: hex_out = 1;
        8'b0001_0100: hex_out = 2;
        8'b0001_1000: hex_out = 3;
        8'b0010_0001: hex_out = 4;
        8'b0010_0010: hex_out = 5;
        8'b0010_0100: hex_out = 6;
        8'b0010_1000: hex_out = 7;
        8'b0100_0001: hex_out = 8;
        8'b0100_0010: hex_out = 9;
        8'b0100_0100: hex_out = 4'hA;
        8'b0100_1000: hex_out = 4'hB;
        8'b1000_0001: hex_out = 4'hC;
        8'b1000_0010: hex_out = 4'hD;
        8'b1000_0100: hex_out = 4'hE;
        8'b1000_1000: hex_out = 4'hF;
        default: hex_out = 4'bx;
        endcase
endmodule
```

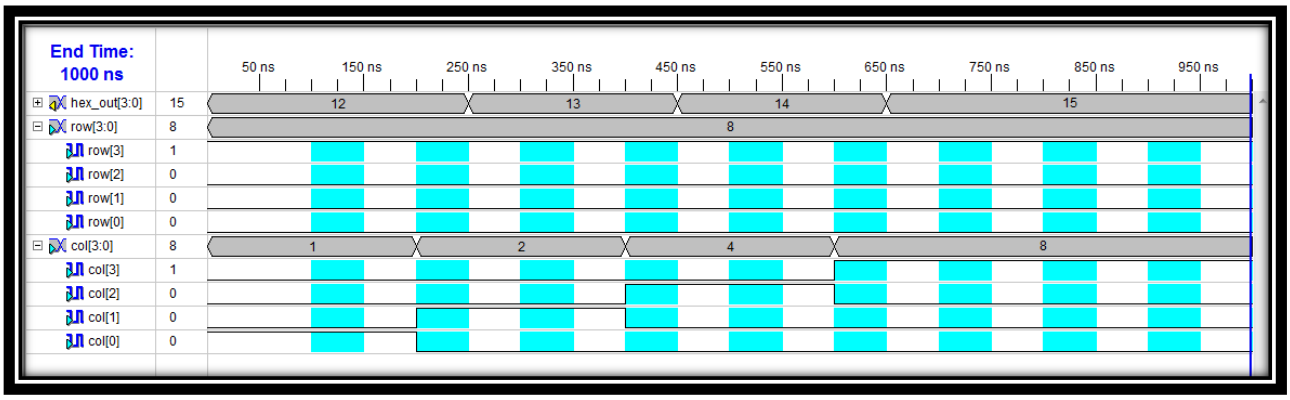
## RTL Schematics:




End Time:
1000 ns
50 ns
150 ns
250 ns
350 ns
450 ns
550 ns
650 ns
750 ns
850 ns
950 ns
hex_out[3:0] 11
8
9
10
11
row[3:0] 4
4
row[3] 0
row[2] 1
row[1] 0
row[0] 0
col[3:0] 8
1
2
4
8
col[3] 1
col[2] 0
col[1] 0
col[0] 0

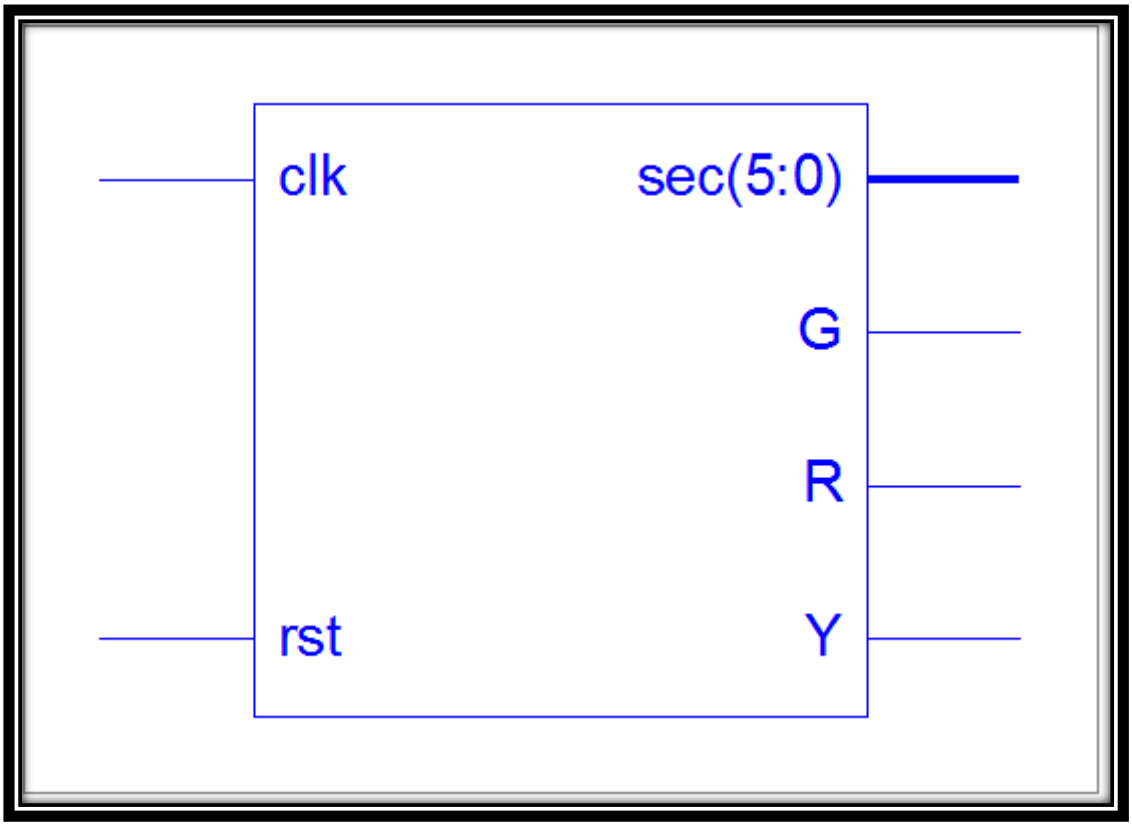

End Time:
1000 ns
50 ns
150 ns
250 ns
350 ns
450 ns
550 ns
650 ns
750 ns
850 ns
950 ns
hex_out[3:0] 15
12
13
14
15
row[3:0] 8
8
row[3] 1
row[2] 0
row[1] 0
row[0] 0
col[3:0] 8
1
2
4
8
col[3] 1
col[2] 0
col[1] 0
col[0] 0

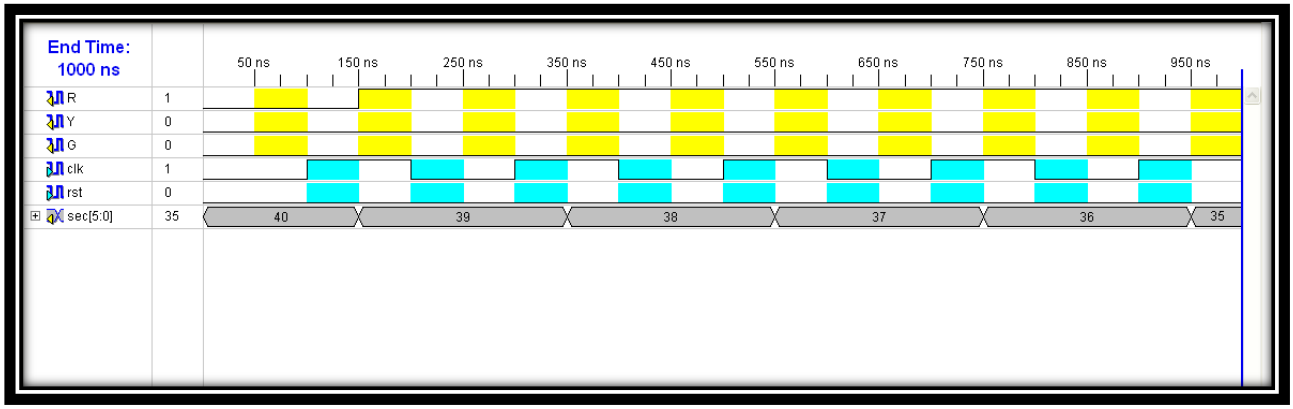
# Lab Assignment No. 11:
## One-way Traffic Light Controller

```
module traffic(R,Y,G,sec,clk,rst);
output R,Y,G;
output [5:0] sec;
input clk,rst;

reg [5:0] sec = 0;
reg R = 0;
reg Y = 0;
reg G = 0;
reg [5:0] cnt;

always @ (clk)
begin
        if(rst)
        begin
                sec <= 0;
                R <= 0;
                Y <= 0;
                G <= 0;
        end
        if (rst ==0)
        begin
                cnt = cnt + 1;
                if (cnt < 41)
                begin
                        sec <= 40;
                        R <= 1;
                        Y <= 0;
                        G <= 0;
                        sec <= sec - 1;
                end
```

```
            if (cnt < 43 & cnt > 40)
            begin
                  sec <= 0;
                  R <= 0;
                  Y <= 1;
                  G <= 0;
            end
            if (cnt > 42)
            begin
                  sec <= 8;
                  R <= 0;
                  Y <= 0;
                  G <= 1;
                  sec <= sec - 1;
            end
            if (cnt >51)
            begin
                  cnt = 0;
            end
      end
end
endmodule
```

RTL Schematics:



# Lab Assignment No. 12:

## 4-way Traffic Light Controller

**Counter:**

```
module counter(count,clk,enable,rst);
input clk,enable,rst;
output[7:0] count;
reg[7:0] count;
always @ (posedge clk or negedge rst)
if (rst==0) count = 40;
else
        if (enable ==1) count = count - 1;
        else count = count;
endmodule
```

**Light Controller:**

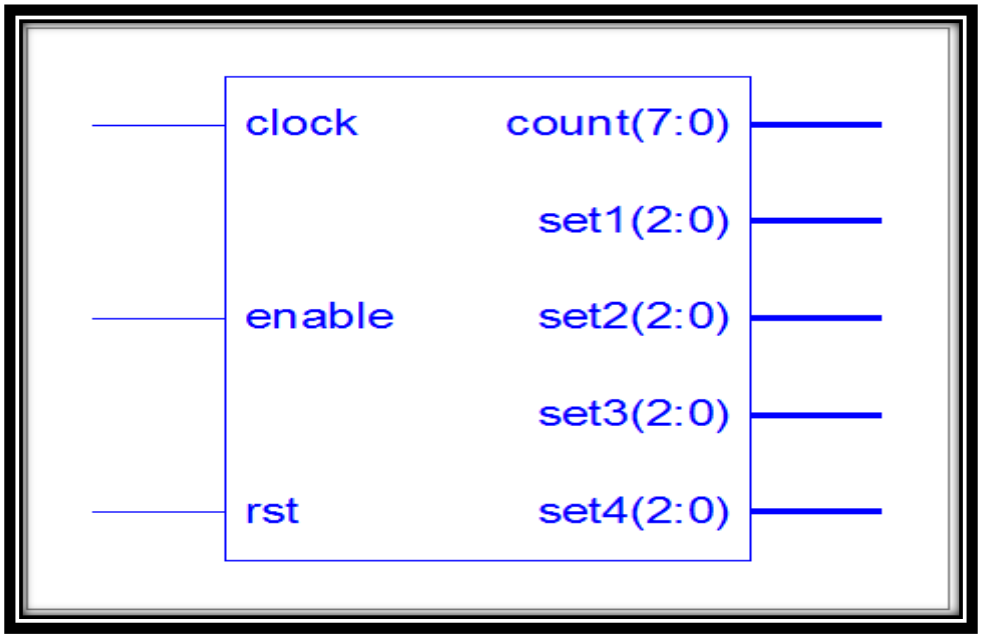
```
module light_controller(set1,set2,set3,set4,count);
input [7:0] count;
output [2:0] set1,set2,set3,set4; //each set={red,yellow,green}
reg [2:0] set1,set2,set3,set4;
always @ (count)
case (count)
40,39,38,37,36,35,34,33,32,31 :
begin
set1=3'b100;
set2=3'b100;
set3=3'b110;
set4=3'b001;
end
30,29,28,27,26,25,24,23,22,21 :
begin
set1=3'b100;
set2=3'b110;
set3=3'b001;
set4=3'b011;
end
```

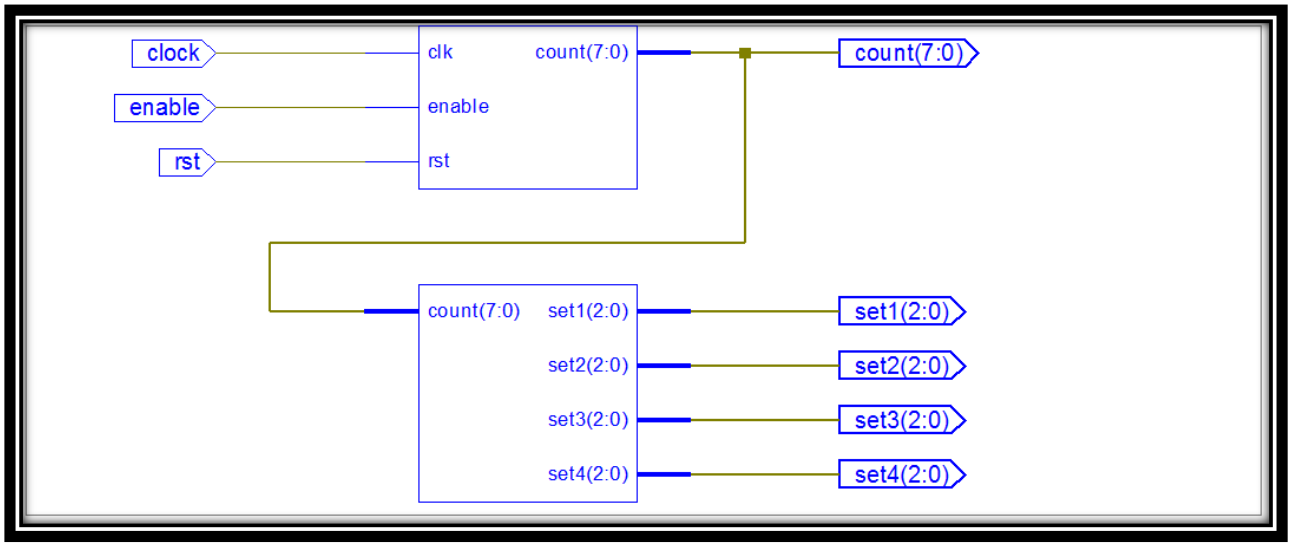
```
20,19,18,17,16,15,14,13,12,11 :
begin
set1=3'b110;
set2=3'b001;
set3=3'b011;
set4=3'b100;
end
10,9,8,7,6,5,4,3,2,1,0 :
begin
set1=3'b001;
set2=3'b011;
set3=3'b100;
set4=3'b110;
end
endcase
endmodule
```

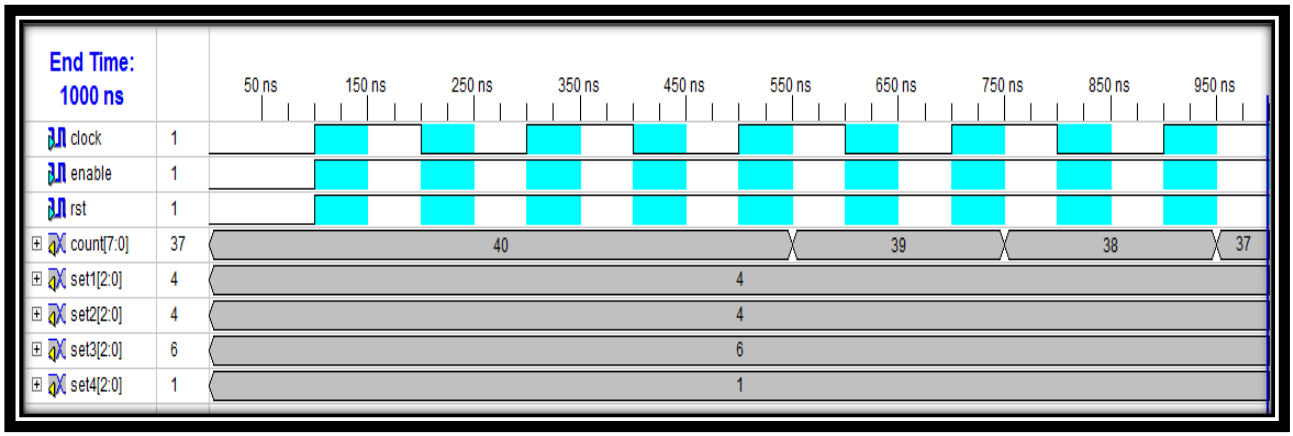
**Main Module:**

```
module signal_4way(count,set1,set2,set3,set4,clock,enable,rst);
input clock,enable,rst;
output [2:0] set1,set2,set3,set4;
output[7:0] count;
wire[7:0] w1;
assign count = w1;
counter m1(w1,clock,enable,rst);
light_controller m2(set1,set2,set3,set4,w1);
endmodule
```

RTL Schematics:



# Lab Assignment No. 13:
# Introduction to Digilent Nexys 2 Board FPGA Spartan 3E-500 FG320

## APPARATUS

Digilent Nexys 2 Board Spartan 3E

## THEORY

### Introduction:

The Nexys-2 is a powerful digital system design platform built around a Xilinx Spartan 3E FPGA. With 16Mbytes of fast SDRAM and 16Mbytes of Flash ROM, the Nexys-2 is ideally suited to embedded processors like Xilinx's 32-bit RISC Microblaze™. The on-board high-speed USB2 port, together with a collection of I/O devices, data ports, and expansion connectors, allow a wide range of designs to be completed without the need for any additional components. The Nexys2 circuit board is a complete, ready-to-use circuit development platform based on a Xilinx Spartan 3E FPGA. Its on-board high-speed USB2 port, 16Mbytes of RAM and ROM, and several I/O devices and ports make it an ideal platform for digital systems of all kinds, including embedded processor systems based on Xilinx's MicroBlaze. The USB2 port provides board power and a programming interface, so the Nexys2 board can be used with a notebook computer to create a truly portable design station.

The Nexys2 brings leading technologies to a platform that anyone can use to gain digital design experience. It can host countless FPGA-based digital systems, and designs can easily grow beyond the board using any or all of the five expansion connectors. Four 12-pin Peripheral Module (Pmod) connectors can accommodate up to eight low-cost Pmods to add features like motor control, A/D and D/A conversion, audio circuits, and a host of sensor and actuator interfaces. All user-accessible signals on the Nexys2 board are ESD and short-circuit protected, ensuring a long operating life in any environment. The Nexys2 board is fully compatible with all versions of the Xilinx ISE tools, including the free Web-Pack.

Now anyone can build real digital systems for less than the price of a textbook. The input power bus drives a 3.3V voltage regulator that supplies all required board current. Some devices require 2.5V, 1.8V, and 1.2V supplies in addition to the main 3.3V supply, and these additional supplies are created by regulators that take their input from the main 3.3V supply. The primary supplies are generated by highly efficient switching

regulators from Linear Technology. These regulators not only use USB power efficiently, they also allow the Nexys2 to run from battery packs for extended periods.

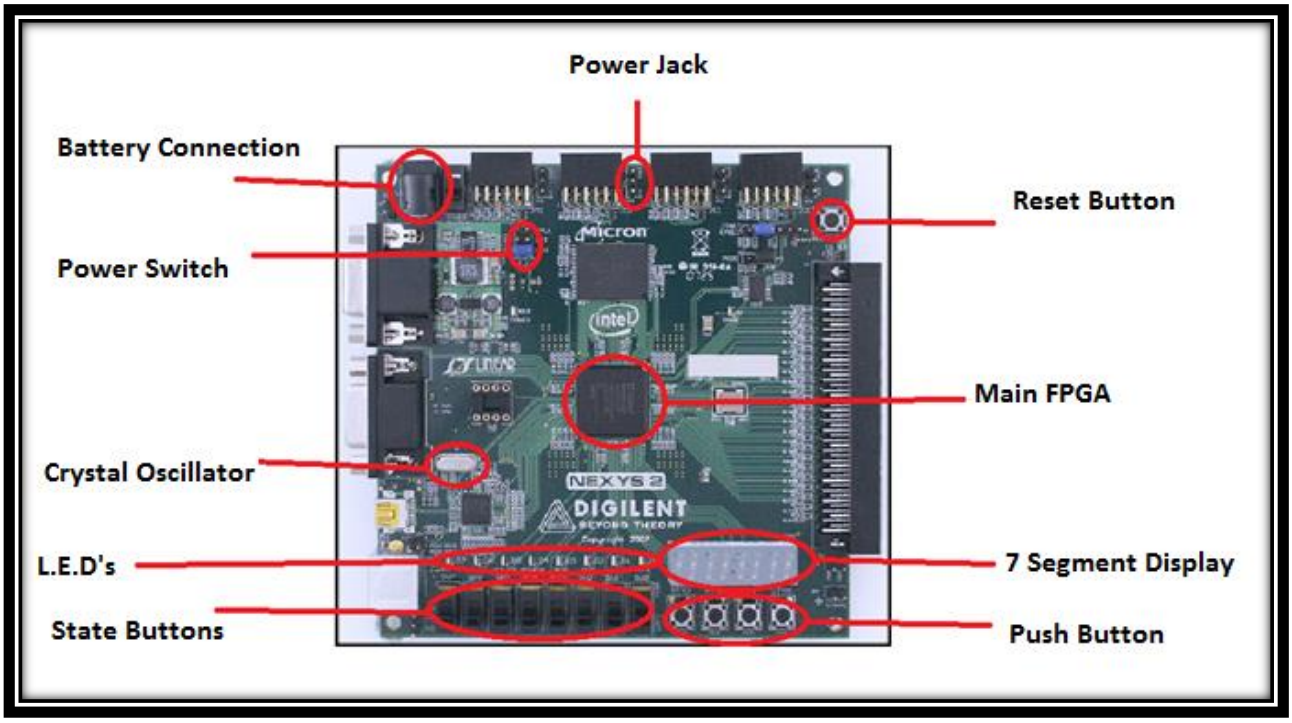
**Characteristics:**

Xilinx Spartan-3E FPGA, 500K or 1200K gate
USB2 port providing board power, device configuration, and high-speed data transfers
Works with ISE/Webpack and EDK
16MB fast Micron PSDRAM
16MB Intel StrataFlash Flash R
Xilinx Platform Flash ROM

High-efficiency switching power supplies (good for battery-powered applications
50MHz oscillator, plus a socket for a second oscillator
75 FPGA I/O's routed to expansion connectors (one high-speed Hirose FX2 connector with 43 signals and four 2x6 Pmod connectors)
All I/O signals are ESD and short-circuit protected, ensuring a long operating life in any environment.

On-board I/O includes eight LEDs, four-digit seven-segment display, four pushbuttons, eight slide switches
Ships in a DVD case with a high-speed USB2 cable

**Circuit Diagram:**



## Clock:

The Nexys2 board includes a 50MHz oscillator and a socket for a second oscillator. Clock signals from the oscillators connect to global clock input pins on the FPGA so they can drive the clock synthesizer blocks available in FPGA. The clock synthesizers (called DLLs, or delay locked loops) provide clock management capabilities that include doubling or quadrupling the input frequency, dividing the input frequency by any integer multiple, and defining precise phase and delay relationships between various clock signals.

## Memory:

The Nexys2 board has external RAM and ROM devices. The external RAM is a 128Mbit Micron M45W8MW16 Cellular RAM pseudo-static DRAM device organized as 8Mbytes x 16bits. It can operate as a typical asynchronous SRAM with read and write cycle times of 70ns, or as a synchronous memory with an 80MHz bus. When operated as an asynchronous SRAM, the Cellular RAM automatically refreshes its internal DRAM arrays, allowing for a simplified memory controller design (similar to any SRAM) in the FPGA.

When operated in synchronous mode, continuous transfers of up to 80MHz are possible. The external ROM is a 128Mbit Intel TE28F128J3D75-110 StrataFlash device organized as 8Mbytes x 16bits. Internally, it contains 128 blocks that can be individually erased, and it supports 110ns read cycle times, with 25ns page-mode reads within blocks. It has an internal 32-byte write buffer that can be written with 70ns cycle times, and the 32-byte buffer can be transferred to the Flash array in 218us (typical). Both devices share a common 16-bit data bus and 24-bit address bus. The Cellular RAM is byte addressable using the upper-byte and lower-byte signals (MT-UB and MT-LB), but the StrataFlash is configured for 16 byte operations only (it is not byte addressable). The output enable (OE) and write enable (WE) signals are shared by both devices, but each device has individual chip enable (CE) signals. Additionally, the Cellular RAM has clock (MT-CLK), wait (MT-WAIT), address valid (MT-ADV) and control register enable (MT_CRE) signals available to the FPGA for use with synchronous transfers, and the StrataFlash has Reset (RP#) and status (STS) signals routed to the FPGA.

## User I/O:

The Nexys2 board includes several input devices, output devices, and data ports, allowing many designs to be implemented without the need for any other components

**LED's:**

Eight LEDs are provided for circuit outputs. LED anodes are driven from the FPGA via 390-ohm resistors, so a logic '1' output will illuminate them with 3-4ma of drive current. A ninth LED is provided as a power-on LED, and a tenth LED indicates FPGA programming status. Note that LEDs 4-7 have different pin assignments due to pinout differences between the -500 and the -1200 die.

**Seven-Segment Display:**

The Nexys2 board contains a four-digit common anode seven-segment LED display. Each of the four digits is composed of seven segments arranged in a "figure 8" pattern, with an LED embedded in each segment. Segment LEDs can be individually illuminated, so any one of 128 patterns can be displayed on a digit by illuminating certain LED segments and leaving the others dark. Of these 128 possible patterns, the ten corresponding to the decimal digits are the most useful.

**Slide Switches and Pushbuttons:**

Four pushbuttons and eight slide switches are provided for circuit inputs. Pushbutton inputs are normally low, and they are driven high only when the pushbutton is pressed. Slide switches generate constant high or low inputs depending on their position. Pushbutton and slide switch inputs use a series resistor for protection against short circuits (a short circuit would occur if an FPGA pin assigned to a pushbutton or slide switch was inadvertently defined as an output).

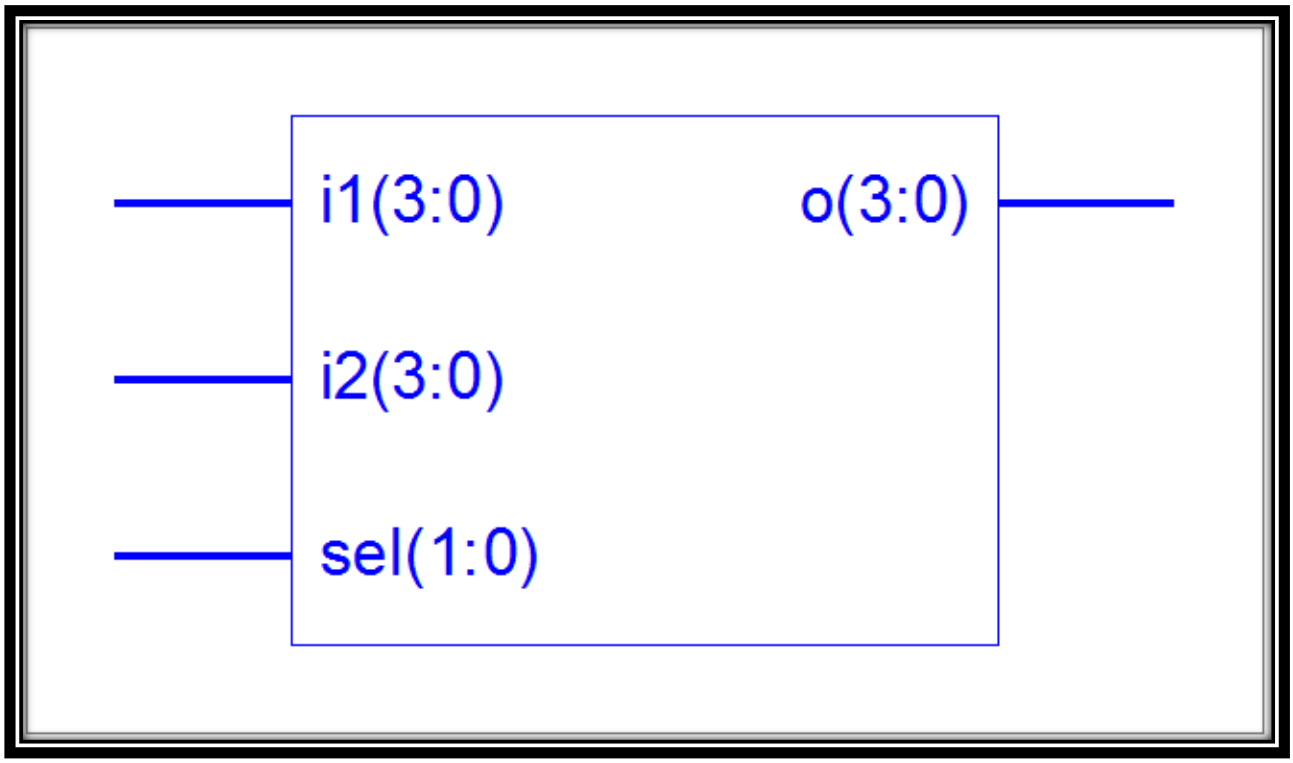
# Lab Assignment No. 14:
## Implementation of Project on FPGA Spartan 3E-500 FG320

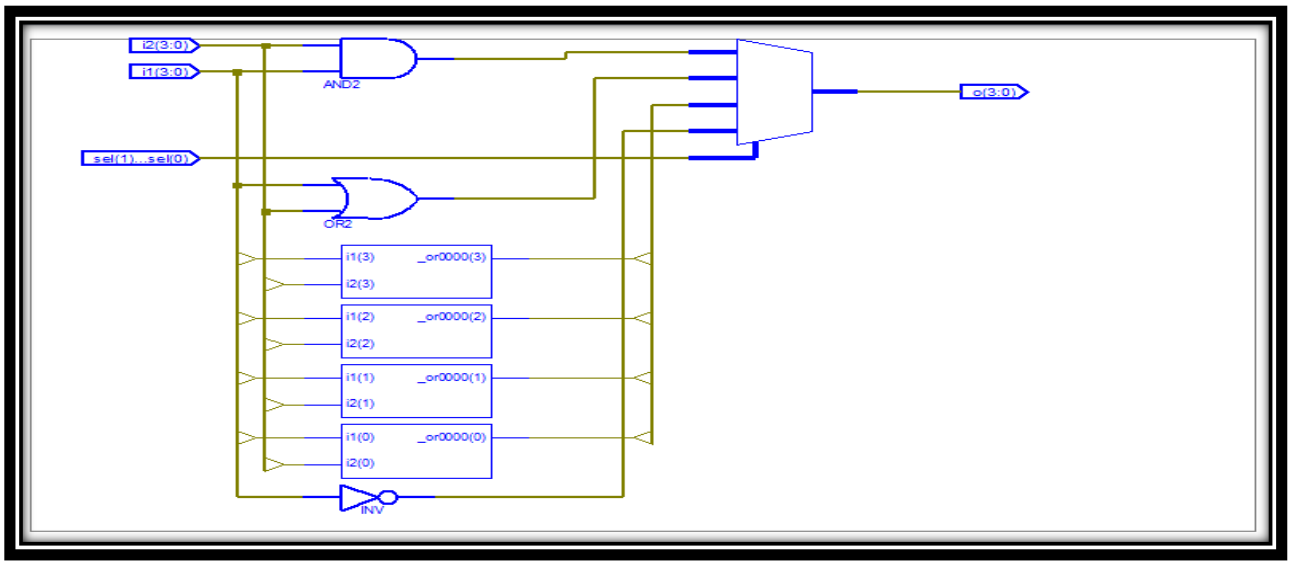
**Project: Arithmetic Control Unit:**

**ALU Module:**

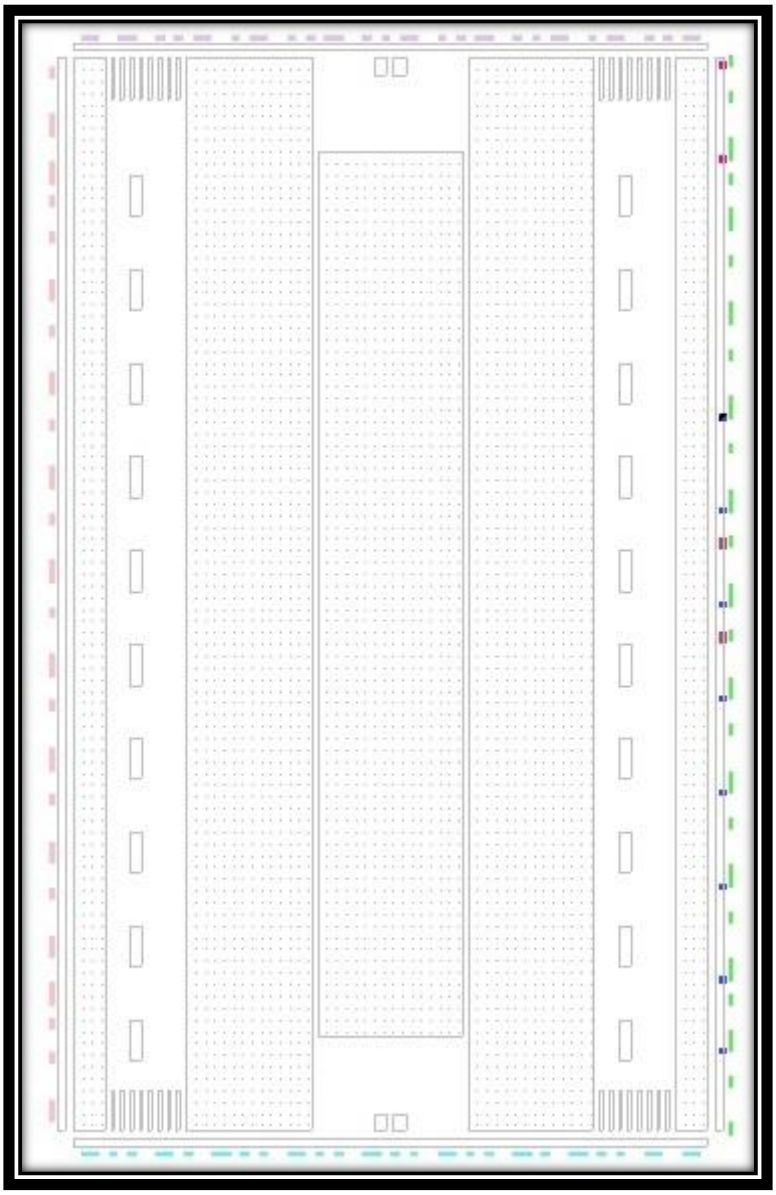
```
module logic_unit(o,i1,i2,sel);
output[3:0] o;
input [3:0] i1,i2;
input [1:0] sel;
reg [3:0] o;
always @ (i1 or i2 or sel)
        case (sel)
        0: o = i1 & i2;
        1: o = i2 | i1;
        2: o = (~i1 & i2)|(~i2 & i1);
        3: o = ~i1;
        endcase
endmodule
```

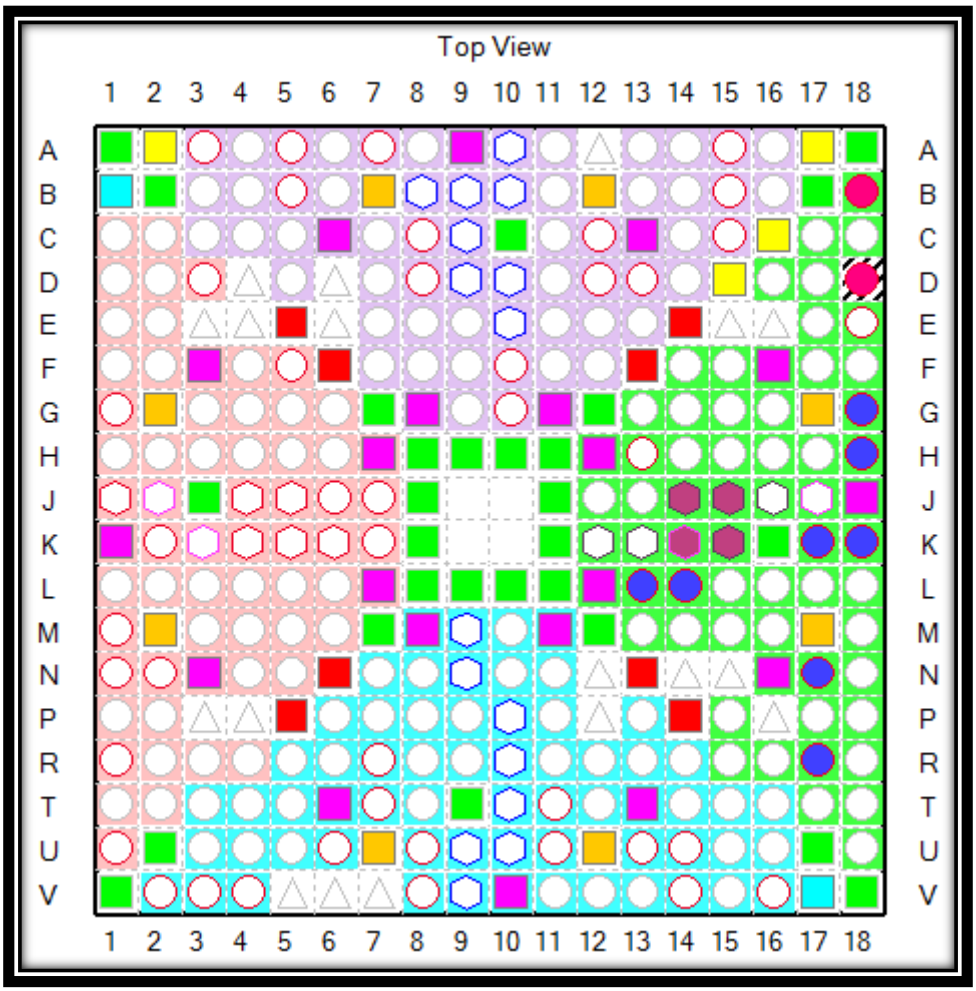
**RTL Schematics:**





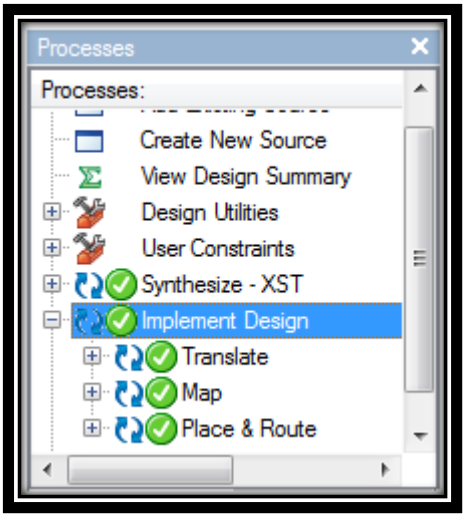

Pins Assign To FPGA:

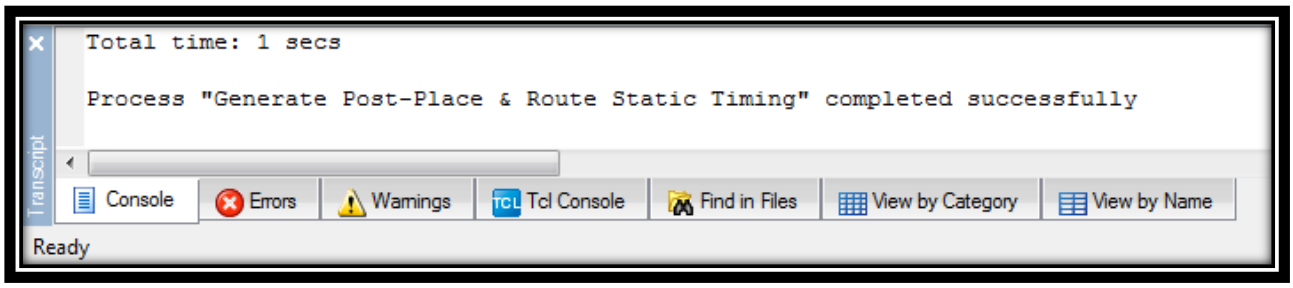





Design Object List - I/O Pins

| I/O Name | I/O Direction | Loc | Bank | I/O Std |
|---|---|---|---|---|
| in1[0] | Input | G18 | BANK | |
| in1[1] | Input | H18 | BANK | |
| in1[2] | Input | K18 | BANK | |
| in1[3] | Input | K17 | BANK | |
| in2[0] | Input | L14 | BANK | |
| in2[1] | Input | L13 | BANK | |
| in2[2] | Input | N17 | BANK | |
| in2[3] | Input | R17 | BANK | |
| out[0] | Output | J14 | BANK | |
| out[1] | Output | J15 | BANK | |
| out[2] | Output | K15 | BANK | |
| out[3] | Output | K14 | BANK | |
| sel[0] | Input | B18 | BANK | |
| sel[1] | Input | D18 | BANK | |






| IOB Name | IOB Type | Direction | IO Standard | Drive Strength | Slew Rate | Reg (s) | Resistor | IBUF/IFD Delay |
|---|---|---|---|---|---|---|---|---|
| in1<0> | IBUF | INPUT | LVCMOS25 | | | | | 0 / 0 |
| in1<1> | IBUF | INPUT | LVCMOS25 | | | | | 0 / 0 |
| in1<2> | IBUF | INPUT | LVCMOS25 | | | | | 0 / 0 |
| in1<3> | IBUF | INPUT | LVCMOS25 | | | | | 0 / 0 |
| in2<0> | IBUF | INPUT | LVCMOS25 | | | | | 0 / 0 |
| in2<1> | IBUF | INPUT | LVCMOS25 | | | | | 0 / 0 |
| in2<2> | IBUF | INPUT | LVCMOS25 | | | | | 0 / 0 |
| in2<3> | IBUF | INPUT | LVCMOS25 | | | | | 0 / 0 |
| out<0> | IOB | OUTPUT | LVCMOS25 | 12 | SLOW | | | 0 / 0 |
| out<1> | IOB | OUTPUT | LVCMOS25 | 12 | SLOW | | | 0 / 0 |
| out<2> | IOB | OUTPUT | LVCMOS25 | 12 | SLOW | | | 0 / 0 |
| out<3> | IOB | OUTPUT | LVCMOS25 | 12 | SLOW | | | 0 / 0 |
| sel<0> | IBUF | INPUT | LVCMOS25 | | | | | 0 / 0 |
| sel<1> | IBUF | INPUT | LVCMOS25 | | | | | 0 / 0 |