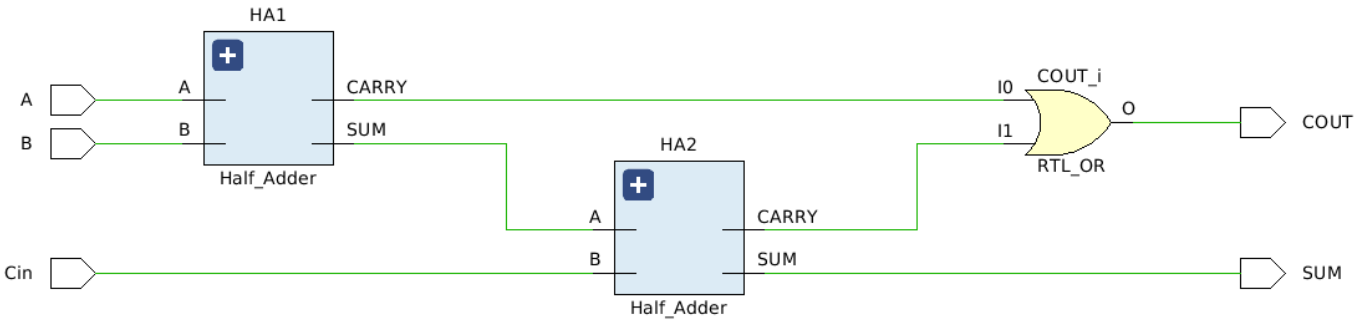
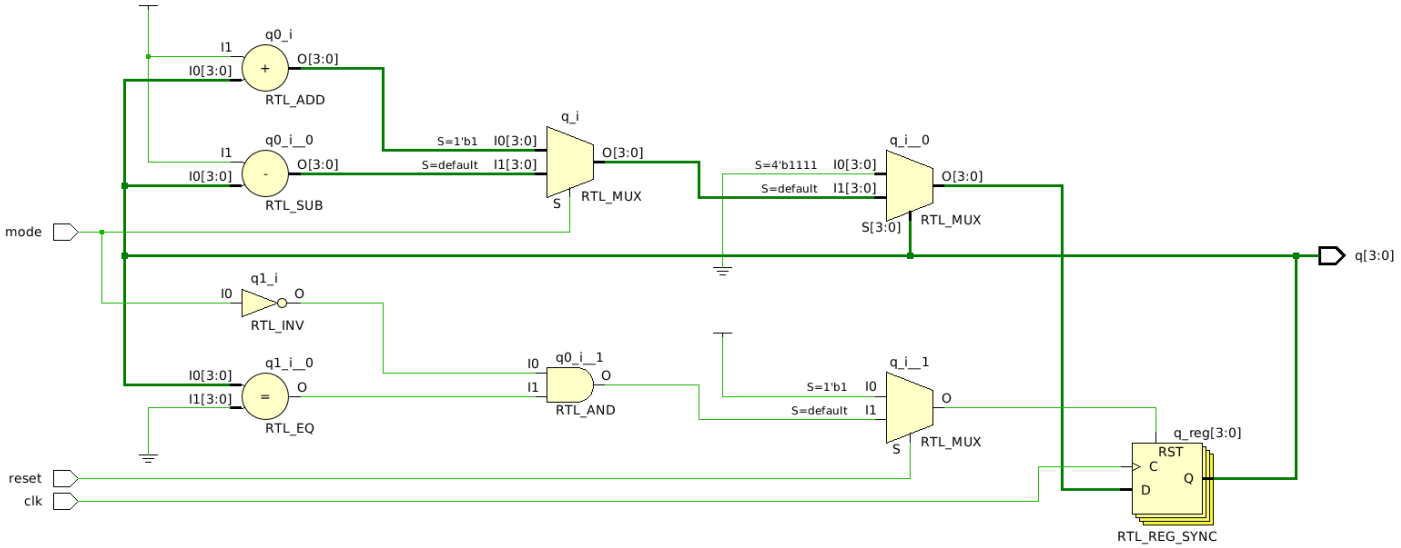
ITI Summer Training

LAB 1

Q1


HA1
A
B
A
B
CARRY
SUM
Half_Adder
HA2
A
B
CARRY
SUM
Half_Adder
Cin
I0
I1
COUT_i
O
RTL_OR
COUT
SUM

Q2


q0_i
I1
I0[3:0]
O[3:0]
RTL_ADD
q0_i__0
I1
I0[3:0]
O[3:0]
RTL_SUB
q_i
S=1'b1 I0[3:0]
S=default I1[3:0]
O[3:0]
S
RTL_MUX
q_i__0
S=4'b1111 I0[3:0]
S=default I1[3:0]
O[3:0]
S[3:0]
RTL_MUX
mode
q[3:0]
q1_i
I0
O
RTL_INV
q1_i__0
I0[3:0]
I1[3:0]
O
RTL_EQ
q0_i__1
I0
I1
O
RTL_AND
q_i__1
S=1'b1 I0
S=default I1
O
S
RTL_MUX
q_reg[3:0]
RST
C
D
Q
RTL_REG_SYNC
reset
clk

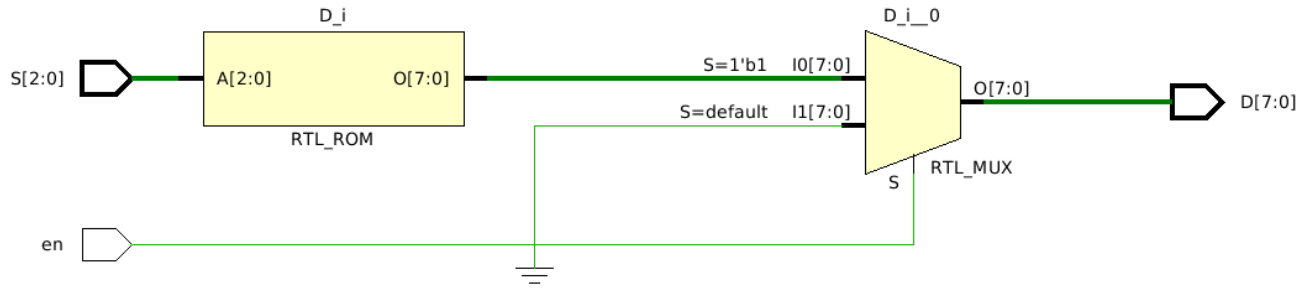

Msgs
/Counter_TB/q
/Counter_TB/clk
/Counter_TB/mode
/Counter_TB/reset
/Counter_TB/COUNT/clk
/Counter_TB/COUNT/reset
/Counter_TB/COUNT/mode
/Counter_TB/COUNT/q
xxxxxxxx
0
x
1
St0
St1
StX
xxxxxxxx
00000000
00000001
00000010
00000011
00000100
00000101
00000100
00000011
00000010
00000001
00000000

```
module Counter ( clk,reset,mode,q);

parameter Width = 4;

input clk,reset,mode;
output reg [Width-1:0]  q;

always @(posedge clk) begin

  if(reset)
            q<=0;

  else begin

      if(mode)   q<=q+1;

      else      q<=q-1;


      if(q==2**Width-1) q<=0;

      if(!mode && q==0) q<=0;

  end

end


endmodule
```

```
module Counter_TB();

wire[7:0] q;
reg clk,mode,reset;




Counter #(.Width(8))  COUNT  (.clk(clk),.mode(mode),.reset(reset),.q(q));


initial begin


  clk = 0;
forever begin

 #5 clk = ~clk;

end

end


initial begin

reset = 1;

#50


reset = 0;
```

```
71
72
73      mode = 1;
74
75      #50
76
77      mode = 0;
78
79
80
81      end
82
83
84      endmodule
85
```

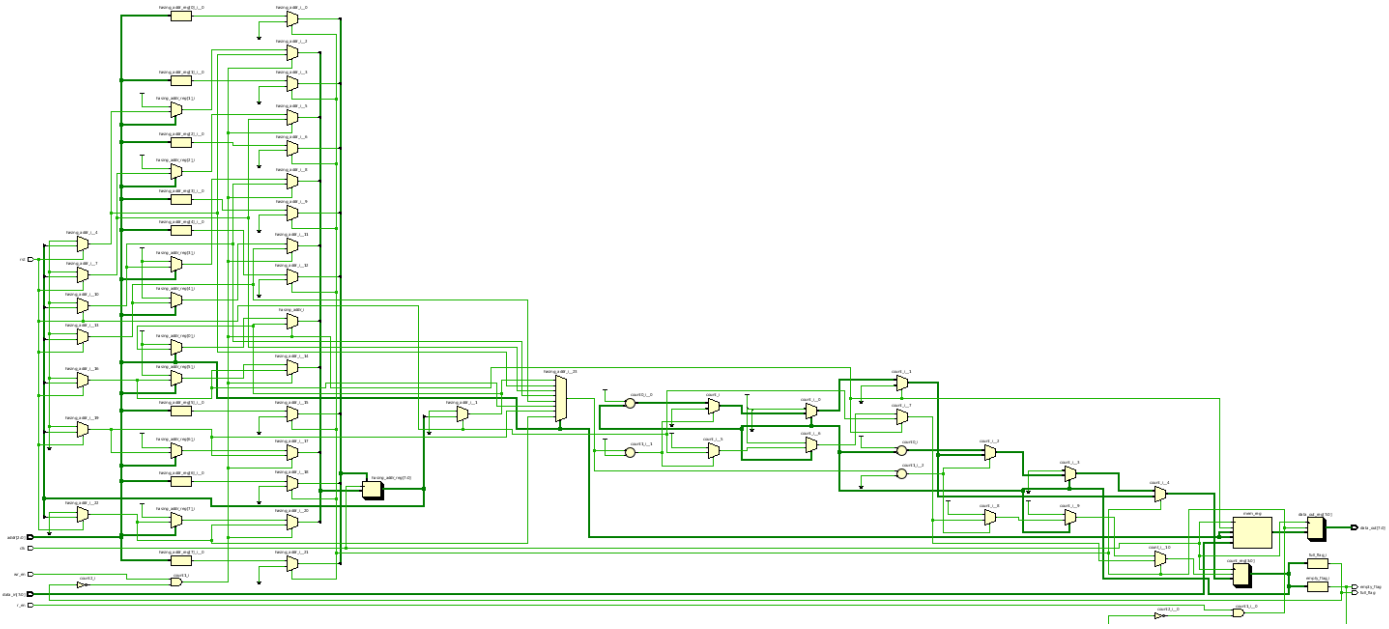
Q3

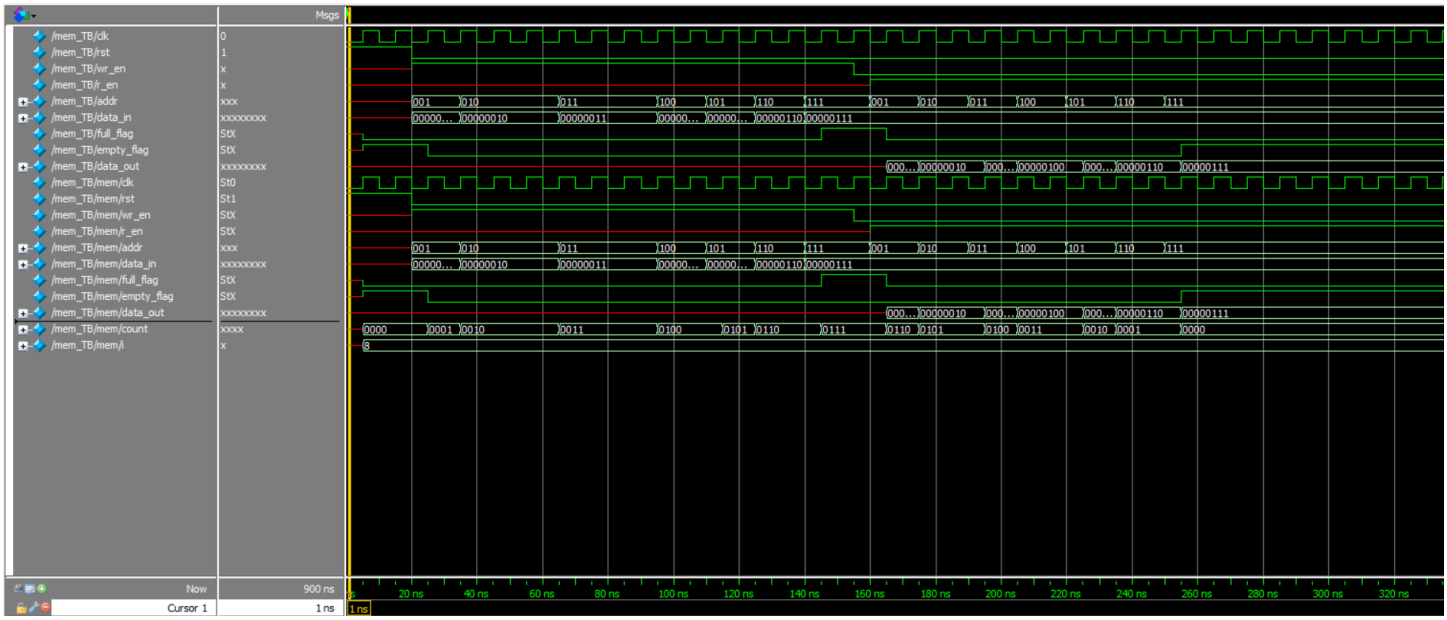


```
module decoder(input[2:0]S ,input en, output reg [7:0] D);


always@*

begin

if(en) begin
 case(S)

   3'b000: D=8'b0000_0001;
   3'b001: D=8'b0000_0010;
   3'b010: D=8'b0000_0100;
   3'b011: D=8'b0000_1000;
   3'b100: D=8'b0001_0000;
   3'b101: D=8'b0010_0000;
   3'b110: D=8'b0100_0000;
   3'b111: D=8'b1000_0000;

endcase

end


else D = 0;



end


endmodule

```

```
37    module decoder_TB();
38
39    wire[7:0] D;
40
41    reg[2:0] S;
42
43    reg en;
44
45    decoder dec(.S(S),.en(en),.D(D));
46
47    initial begin
48
49
50    en =0;
51
52    #20
53
54    en = 1;
55    #10
56
57     S = 0;
58     #10
59
60     S = 1;
61     #10
62
63      S = 2;
64     #10
65
66      S = 3;
67     #10
68
69      S = 4;
70     #10
71
```

Q4






```
1   module mem_buff(input clk ,rst,wr_en,r_en,input[2:0]addr,input[7:0] data_in,output full_flag,empty_flag,output reg [7:0] data_out);
2
3   reg[7:0]mem[0:7];
4   reg[3:0] count;
5   reg hasing_addr [0:7];
6   integer i;
7
8
9
10  assign full_flag = (count==7) ? 1 : 0;
11  assign empty_flag = (count==0) ? 1 : 0;
12
13  always@ (posedge clk)
14  begin
15
16  if(rst) begin
17      count <= 0;
18      for(i=0;i<8;i=i+1) hasing_addr[i] = 0;
19  end
20
21
22  if(wr_en && !full_flag) begin
23
24   if(hasing_addr[addr]!=1)
25         count <= count +1;
26
27   mem[addr] <= data_in;
28   hasing_addr[addr] <= 1;
29
30  if(count==7)begin
31    count <= 7;
32
33    end
34
35
36   end
```

```
37
38
39
40      if(r_en && !empty_flag)  begin
41         data_out <= mem[addr];
42
43         if(hasing_addr[addr]!=0)
44           count <= count -1;
45
46         hasing_addr[addr]<=0;
47
48       if(count==0)begin
49         count <= 0;
50
51        end
52
53
54      end
55
56
57
58
59      end
60
61
62
63      endmodule
64
65
```

```
68    module mem_TB;
69
70    reg clk ,rst,wr_en,r_en;
71    reg[2:0]addr;
72    reg[7:0] data_in;
73    wire full_flag,empty_flag;
74    wire [7:0] data_out;
75
76    mem_buff mem(.clk(clk),.rst(rst),.wr_en(wr_en),.addr(addr),.data_in(data_in),.r_en(r_en),.full_flag(full_flag),.empty_flag(empty_flag),.data_out(data_out));
77
78    initial begin
79
80
81      clk = 0;
82    forever begin
83
84     #5 clk = ~clk;
85
86    end
87
88    end
89
90
91
92    initial begin
93
94
95    rst =1;
96    #20
97    rst = 0;
98
99    data_in = 1;
100   wr_en= 1;
101   addr = 1;
102   #15
```

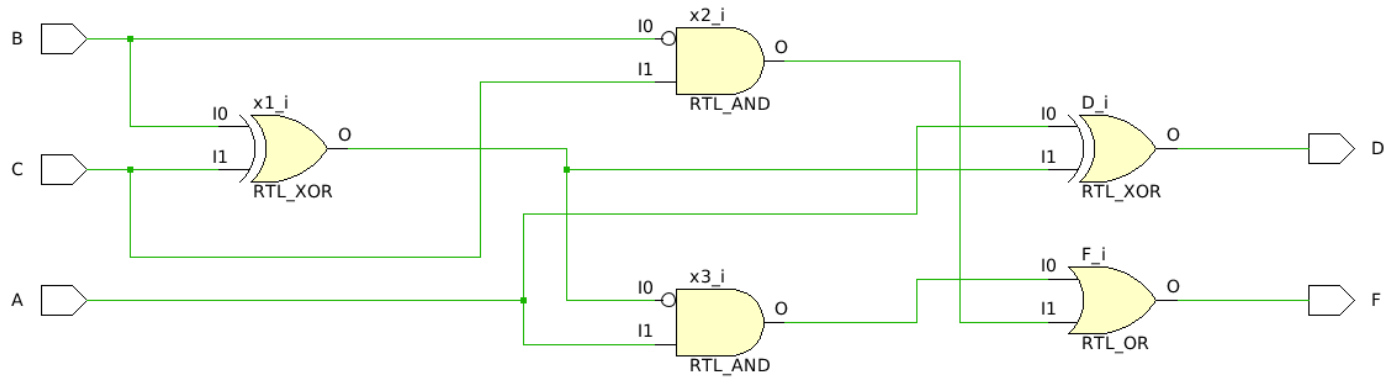
```
110     data_in = 2;
111     wr_en= 1;
112     addr = 2;
113     #15
114
115
116     data_in = 3;
117     wr_en= 1;
118     addr = 3;
119     #15
120
121     data_in = 3;
122     wr_en= 1;
123     addr = 3;
124     #15
125
126     data_in = 4;
127     wr_en= 1;
128     addr = 4;
129     #15
130
131     data_in = 5;
132     wr_en= 1;
133     addr = 5;
134     #15
135
136     data_in = 6;
137     wr_en= 1;
138     addr = 6;
139     #15
140
141     data_in= 7;
142     wr_en= 1;
143     addr = 7;
144     #15
145
```

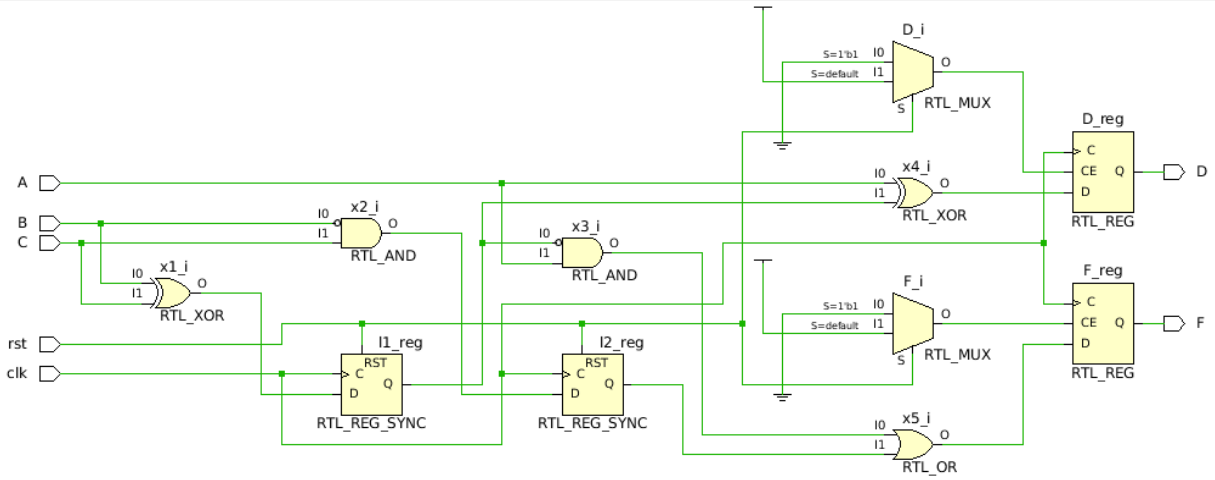
```
146     wr_en=0;
147     #5
148
149     r_en= 1;
150     addr = 1;
151     #15
152
153     r_en= 1;
154     addr = 2;
155     #15
156
157
158
159     r_en= 1;
160     addr = 3;
161     #15
162
163
164     r_en= 1;
165     addr = 4;
166     #15
167
168
169     r_en= 1;
170     addr = 5;
171     #15
172
173
174     r_en= 1;
175     addr = 6;
176     #15
```

Q5

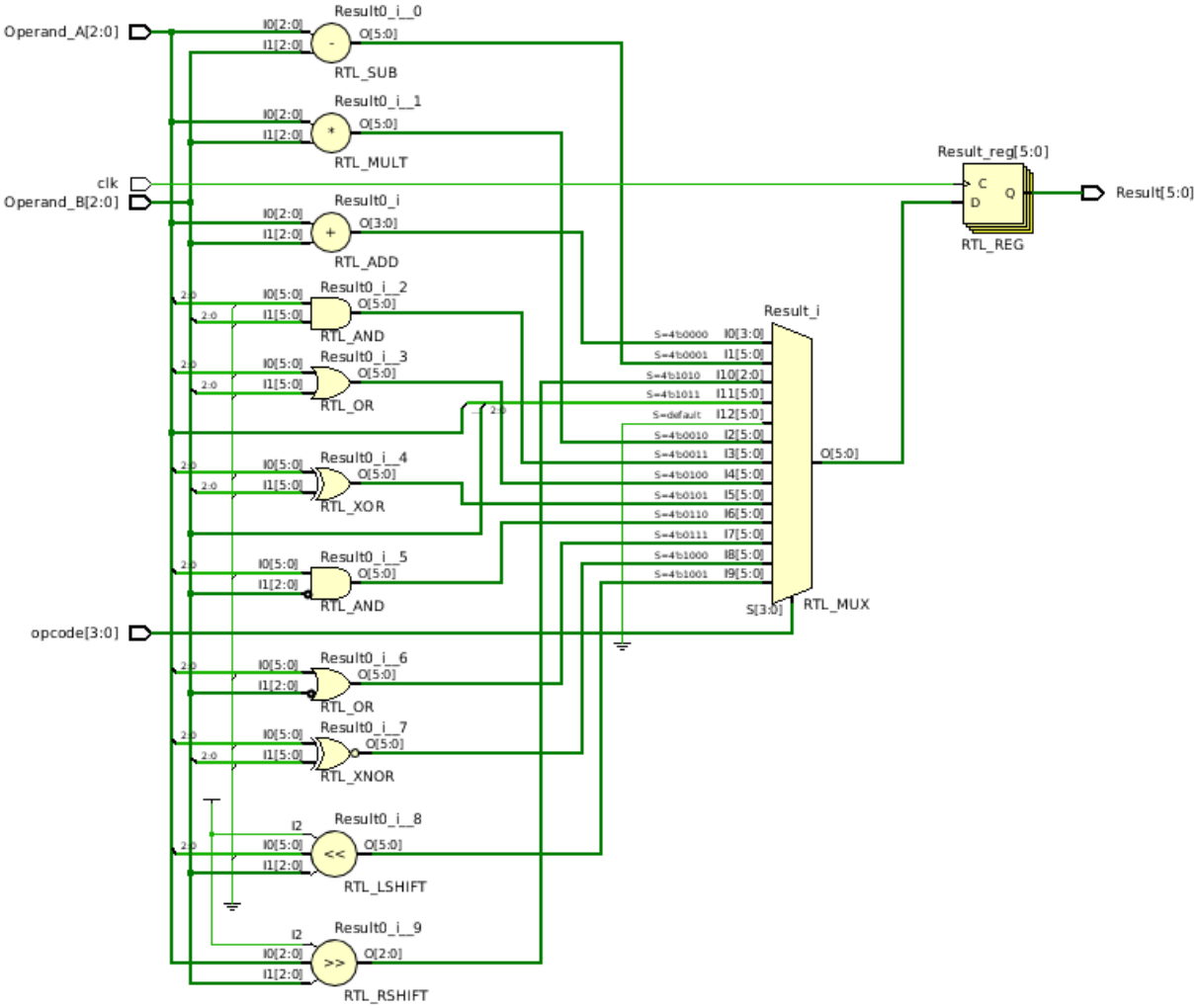
```
4     wire[3:0] Seq1 = 4'b1011;
5     wire[3:0] Seq2 = 4'b0011;
6     wire[3:0] Seq3 = 4'b1010;
7     wire[1:0] temp = Seq2[1:0] & Seq1[3:2];
8     result = {2{Seq3[3]},Seq3,temp};
```

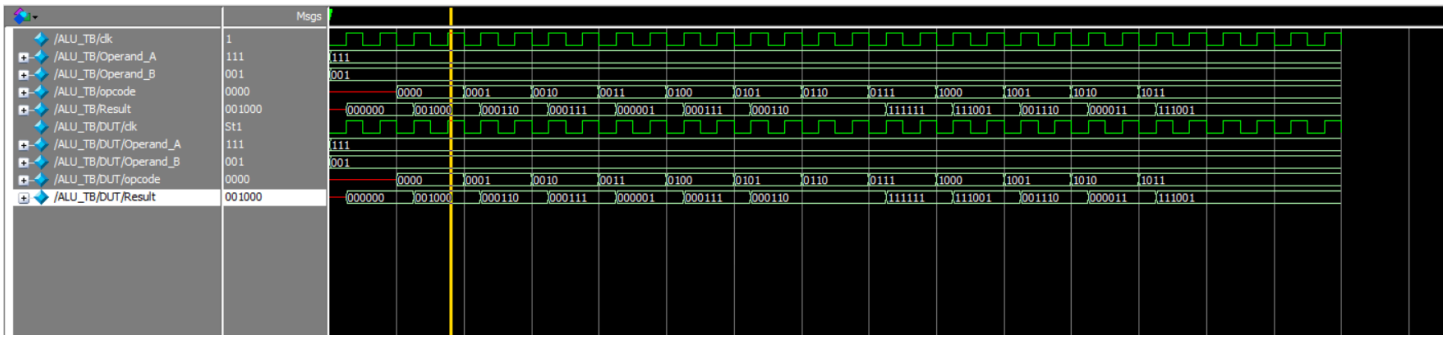
Q6


B
C
A
x2_i
I0
I1
O
RTL_AND
x1_i
RTL_XOR
x3_i
D_i
F_i
RTL_OR
D
F


A
B
C
rst
clk
x1_i
x2_i
x3_i
x4_i
x5_i
D_i
F_i
I1_reg
I2_reg
RTL_XOR
RTL_AND
RTL_REG_SYNC
RTL_MUX
RTL_OR
D_reg
F_reg
RTL_REG
D
F

Q7


Operand_A[2:0]
clk
Operand_B[2:0]
opcode[3:0]
Result0_i__0
RTL_SUB
Result0_i__1
RTL_MULT
Result0_i
RTL_ADD
Result0_i__2
RTL_AND
Result0_i__3
RTL_OR
Result0_i__4
RTL_XOR
Result0_i__5
RTL_AND
Result0_i__6
RTL_OR
Result0_i__7
RTL_XNOR
Result0_i__8
RTL_LSHIFT
Result0_i__9
RTL_RSHIFT
Result_i
RTL_MUX
Result_reg[5:0]
RTL_REG
Result[5:0]


Msgs
/ALU_TB/clk
/ALU_TB/Operand_A
/ALU_TB/Operand_B
/ALU_TB/opcode
/ALU_TB/Result
/ALU_TB/DUT/clk
/ALU_TB/DUT/Operand_A
/ALU_TB/DUT/Operand_B
/ALU_TB/DUT/opcode
/ALU_TB/DUT/Result

```
module ALU(input clk,input[2:0] Operand_A,Operand_B,input[3:0] opcode,output reg[5:0] Result);


always@(posedge clk) begin

  case(opcode)

  4'b0000:  Result <= Operand_A + Operand_B;
  4'b0001:  Result <= Operand_A - Operand_B;
  4'b0010:  Result <= Operand_A * Operand_B;
  4'b0011:  Result <= Operand_A & Operand_B;
  4'b0100:  Result <= Operand_A | Operand_B;
  4'b0101:  Result <= Operand_A ^ Operand_B;
  4'b0110:  Result <= Operand_A &~ Operand_B;
  4'b0111:  Result <= Operand_A |~ Operand_B;
  4'b1000:  Result <= Operand_A ^~ Operand_B;
  4'b1001:  Result <= Operand_A << Operand_B;
  4'b1010:  Result <= Operand_A >> Operand_B;
  4'b1011:  Result <= {Operand_A , Operand_B};
  default:  Result <=0;
  endcase


end


endmodule
```

```
module ALU_TB;

 reg clk;
 reg[2:0] Operand_A,Operand_B;
 reg[3:0] opcode;
 wire[5:0] Result;

 ALU DUT(.clk(clk),.Operand_A(Operand_A),.Operand_B(Operand_B),.opcode(opcode),.Result(Result));

 initial begin

  clk =0;

  forever begin
   clk = #5 ~clk;
  end

 end


 initial begin

 Operand_A = 8'b000_0111;
 Operand_B = 8'b0000_0001;
 #20

 opcode = 0 ;
```

```
  #20

 opcode = 1 ;
 #20

 opcode = 2 ;


  #20

 opcode = 3 ;

  #20

 opcode = 4 ;

  #20

 opcode = 5;

  #20

 opcode = 6 ;


  #20
```

```
opcode = 6 ;


 #20

opcode = 7 ;
 #20

opcode = 8;


 #20

opcode = 9 ;

 #20

opcode = 10 ;

 #20

opcode = 11 ;
end

endmodule
```