Amrita Vishwa Vidyapeetham

Amrita School of Engineering, Coimbatore

Department of Computer Science and Engineering

Practice Question

Sub Code: 19CS211	**Sub Title: COA**
Roll No: CB.EN.U4CSE19453	**Name: R.ABHINAV**
Practice Lab	**Date: 08-03-2021**

---

1. Design an Up-Counter using the half-adder circuit

- **Verilog Code:**

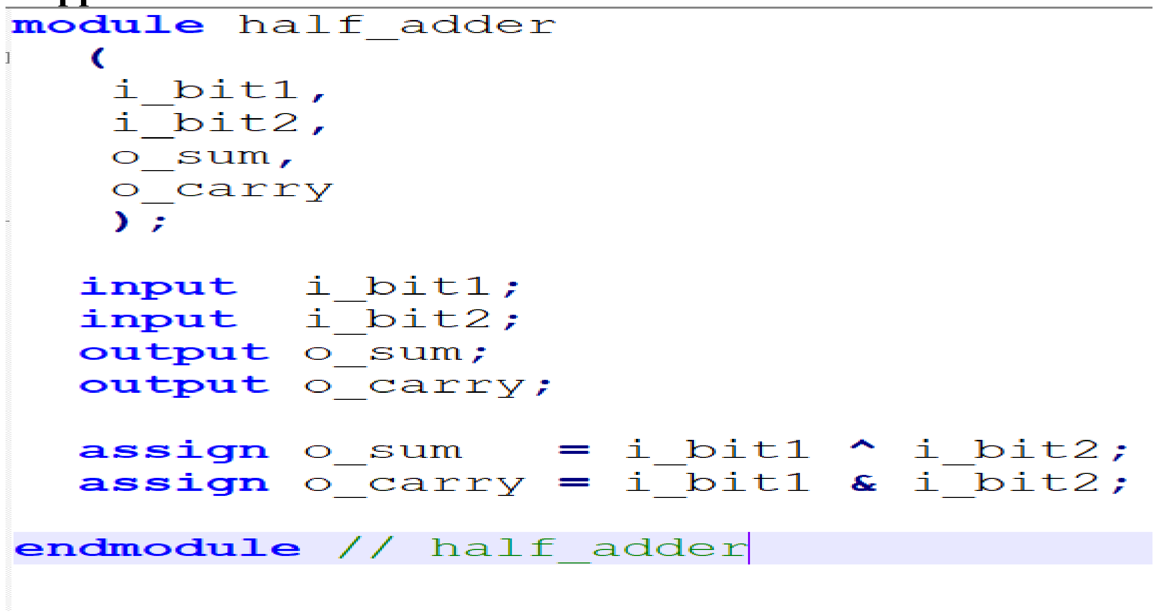
```
module half_adder
  (
   i_bit1,
   i_bit2,
   o_sum,
   o_carry
   );

  input  i_bit1;
  input  i_bit2;
  output o_sum;
  output o_carry;

  assign o_sum   = i_bit1 ^ i_bit2;  // bitwise xor
  assign o_carry = i_bit1 & i_bit2;  // bitwise and

endmodule // half_adder
```

- **Testbench :**

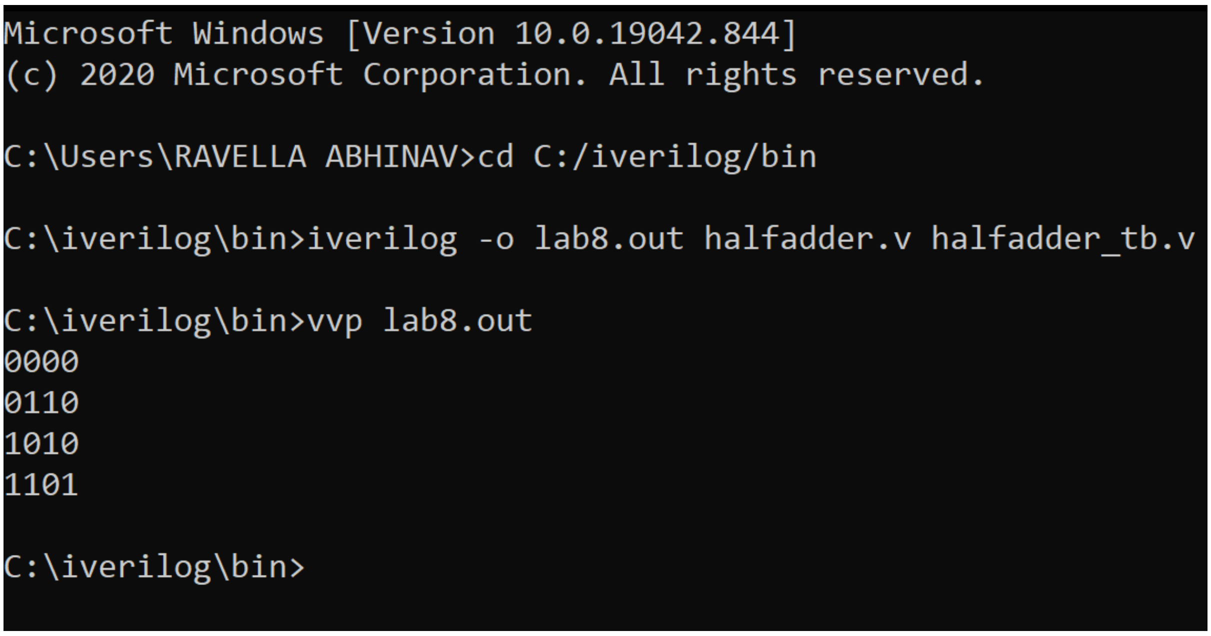
```
module half_adder_tb;

  reg r_BIT1 = 0;
  reg r_BIT2 = 0;
  wire w_SUM;
  wire w_CARRY;
```

```
half_adder half_adder_inst
  (
   .i_bit1(r_BIT1),
   .i_bit2(r_BIT2),
   .o_sum(w_SUM),
   .o_carry(w_CARRY)
   );

initial
  begin
    $monitor(r_BIT1, r_BIT2, w_SUM, w_CARRY);
    r_BIT1 = 1'b0;
    r_BIT2 = 1'b0;
    #10;
    r_BIT1 = 1'b0;
    r_BIT2 = 1'b1;
    #10;
    r_BIT1 = 1'b1;
    r_BIT2 = 1'b0;
    #10;
    r_BIT1 = 1'b1;
    r_BIT2 = 1'b1;
    #10;
  end

endmodule
```

- **Snippets :**



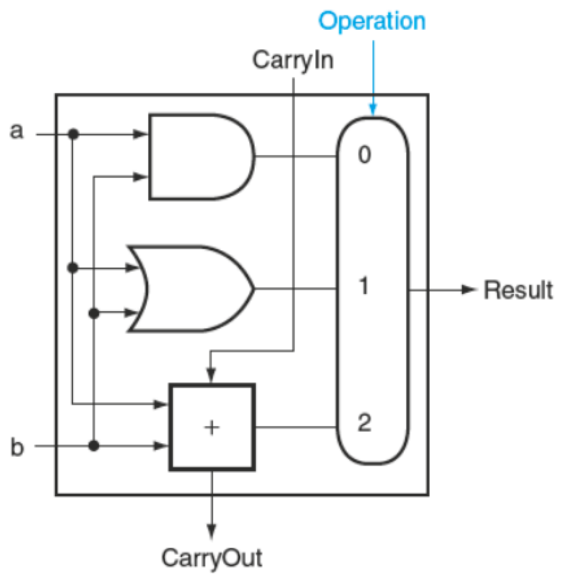
```
module half_adder
  (
   i_bit1,
   i_bit2,
   o_sum,
   o_carry
   );

  input  i_bit1;
  input  i_bit2;
  output o_sum;
  output o_carry;

  assign o_sum   = i_bit1 ^ i_bit2;
  assign o_carry = i_bit1 & i_bit2;

endmodule // half_adder
```

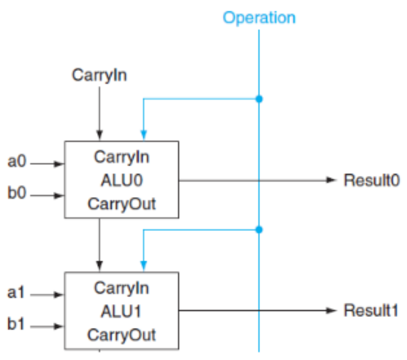


```
Microsoft Windows [Version 10.0.19042.844]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\RAVELLA ABHINAV>cd C:/iverilog/bin

C:\iverilog\bin>iverilog -o lab8.out halfadder.v halfadder_tb.v

C:\iverilog\bin>vvp lab8.out
0000
0110
1010
1101

C:\iverilog\bin>
```

2. Design an Down-Counter using the half-subtractor circuit

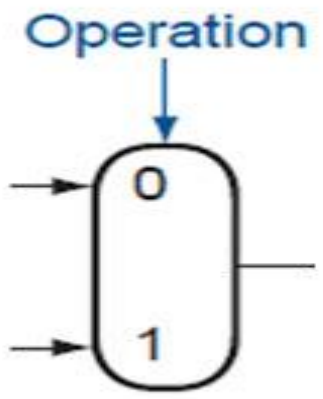
3. Design an 2-bit ALU using the following circuit and logic





Based on the diagram above,

- the multiplexer on the right side select either a AND b or a OR b depending on whether the value of operation is 0 or 1.
- While the line in color control the multiplexer and the line is to distinguish from the lines containing data.



- CarryOut=(b.CarryIn)+ (a.CarryIn)+(a.b)+(a.b.CarryIn)
- Simplified :

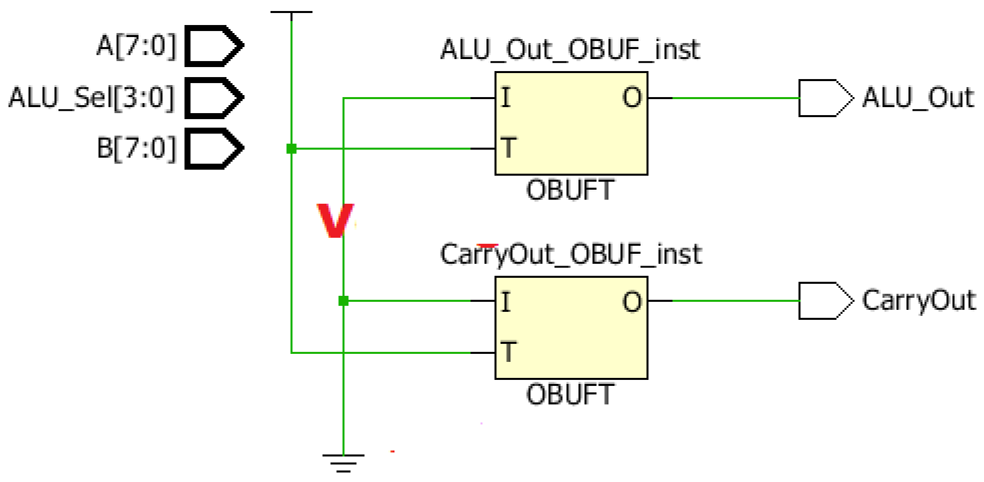
  CarryOut=(b.CarryIn)+(a.CarryIn)+(a.b)

- If (a.b.CarryIn) is true, all other three terms must be true and so the last term can be leaving out corresponding to the 4th line of the table)
-

INPUTS		
a	b	CarryIn
0	1	1
1	0	1
1	1	0
1	1	1

-

4. Write Verilog code for ALU. Write testbench Verilog code for the ALU for simulation and testing.



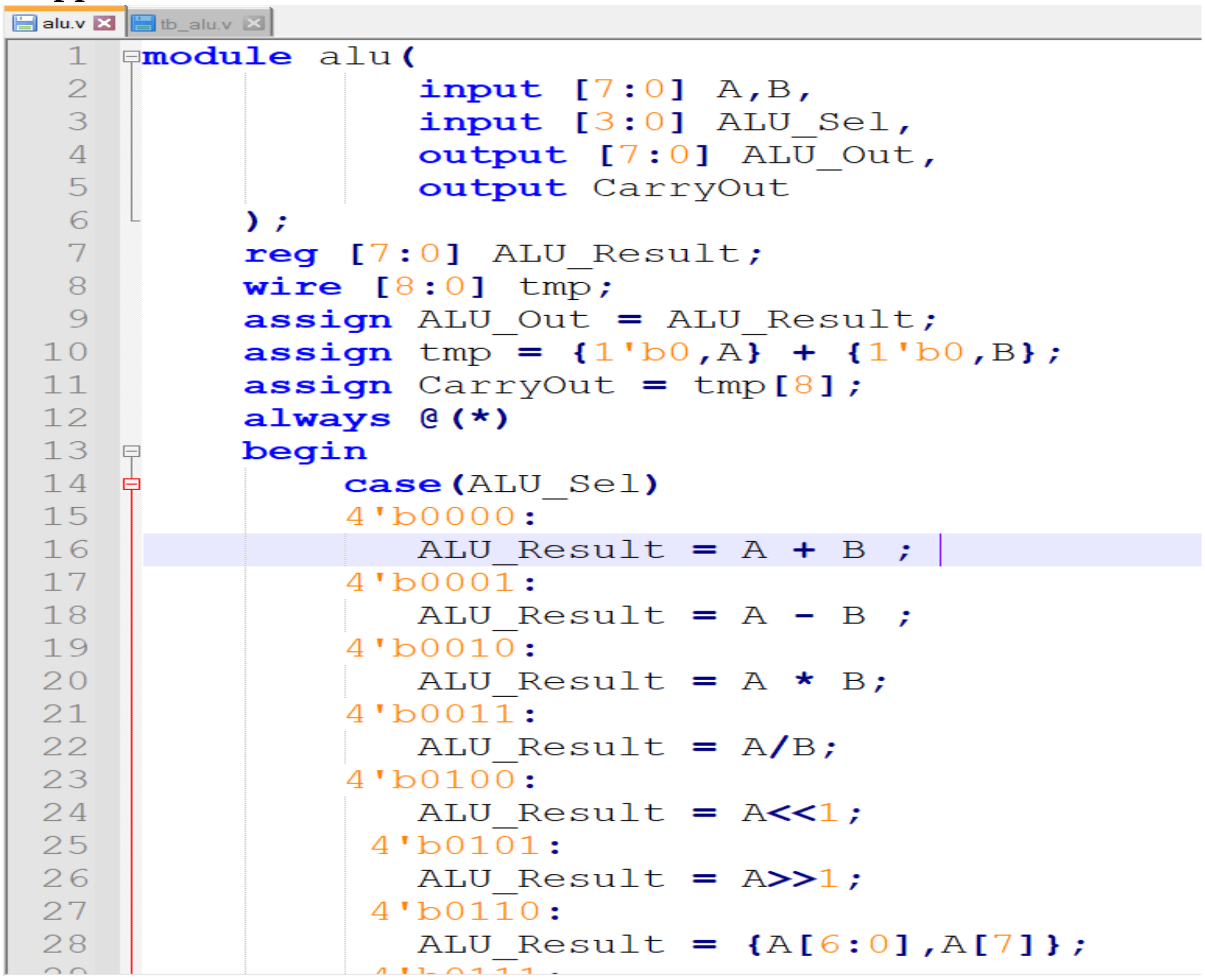
Verilog Code :

```
        module alu(
                input [7:0] A,B,
                input [3:0] ALU_Sel,
                output [7:0] ALU_Out,
                output CarryOut
           );
           reg [7:0] ALU_Result;
           wire [8:0] tmp;
           assign ALU_Out = ALU_Result;
           assign tmp = {1'b0,A} + {1'b0,B};
           assign CarryOut = tmp[8];
           always @(*)
           begin
              case(ALU_Sel)
              4'b0000:
                 ALU_Result = A + B ;
              4'b0001:
                 ALU_Result = A - B ;
              4'b0010:
                 ALU_Result = A * B;
              4'b0011:
                 ALU_Result = A/B;
              4'b0100:
                 ALU_Result = A<<1;
                4'b0101:
                  ALU_Result = A>>1;
                4'b0110:
                  ALU_Result = {A[6:0],A[7]};
                4'b0111:
                  ALU_Result = {A[0],A[7:1]};
                 4'b1000:
                  ALU_Result = A & B;
                 4'b1001:
                  ALU_Result = A | B;
                 4'b1010:
                  ALU_Result = A ^ B;
                 4'b1011:
                  ALU_Result = ~(A | B);
                 4'b1100:
                  ALU_Result = ~(A & B);
                 4'b1101:
                  ALU_Result = ~(A ^ B);
                 4'b1110:
                  ALU_Result = (A>B)?8'd1:8'd0 ;
                 4'b1111:
                   ALU_Result = (A==B)?8'd1:8'd0 ;
                 default: ALU_Result = A + B ;
               endcase
           end

endmodule
```

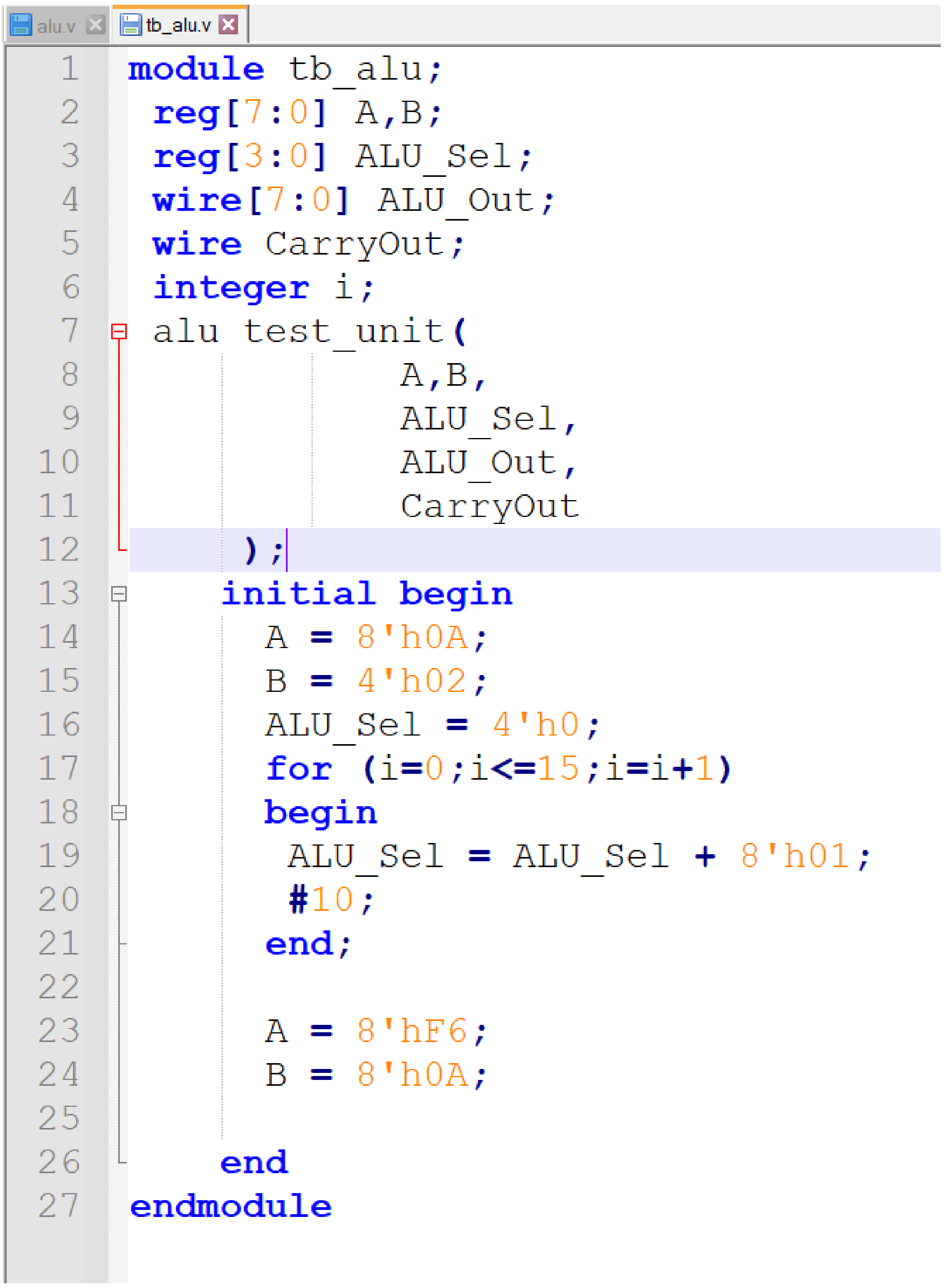
Test Bench:

```
          module tb_alu;
           reg[7:0] A,B;
           reg[3:0] ALU_Sel;
           wire[7:0] ALU_Out;
           wire CarryOut;
           integer i;
           alu test_unit(
                      A,B,
                      ALU_Sel,
                      ALU_Out,
          CarryOut
      );
     initial begin
      A = 8'h0A;
      B = 4'h02;
      ALU_Sel = 4'h0;
      for (i=0;i<=15;i=i+1)
      begin
       ALU_Sel = ALU_Sel + 8'h01;
       #10;
      end;

      A = 8'hF6;
      B = 8'h0A;

     end
endmodule
```

Snippets:



```
module alu(
                input [7:0] A,B,
                input [3:0] ALU_Sel,
                output [7:0] ALU_Out,
                output CarryOut
        );
        reg [7:0] ALU_Result;
        wire [8:0] tmp;
        assign ALU_Out = ALU_Result;
        assign tmp = {1'b0,A} + {1'b0,B};
        assign CarryOut = tmp[8];
        always @(*)
        begin
            case(ALU_Sel)
            4'b0000:
                ALU_Result = A + B ;
            4'b0001:
                ALU_Result = A - B ;
            4'b0010:
                ALU_Result = A * B;
            4'b0011:
                ALU_Result = A/B;
            4'b0100:
                ALU_Result = A<<1;
             4'b0101:
                ALU_Result = A>>1;
             4'b0110:
                ALU_Result = {A[6:0],A[7]};
```


alu.v tb_alu.v

```verilog
module tb_alu;
 reg[7:0] A,B;
 reg[3:0] ALU_Sel;
 wire[7:0] ALU_Out;
 wire CarryOut;
 integer i;
 alu test_unit(
           A,B,
           ALU_Sel,
           ALU_Out,
           CarryOut
     );
    initial begin
      A = 8'h0A;
      B = 4'h02;
      ALU_Sel = 4'h0;
      for (i=0;i<=15;i=i+1)
      begin
       ALU_Sel = ALU_Sel + 8'h01;
       #10;
      end;

      A = 8'hF6;
      B = 8'h0A;

    end
endmodule
```