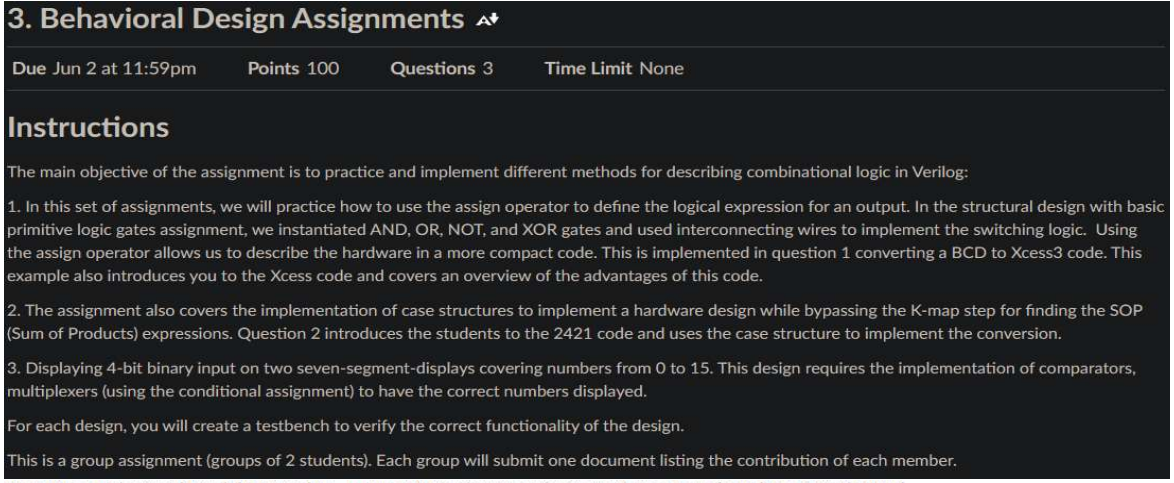


3. Behavioral Design Assignments

Due Jun 2 at 11:59pm **Points** 100 **Questions** 3 **Time Limit** None

Instructions

The main objective of the assignment is to practice and implement different methods for describing combinational logic in Verilog:

1. In this set of assignments, we will practice how to use the assign operator to define the logical expression for an output. In the structural design with basic primitive logic gates assignment, we instantiated AND, OR, NOT, and XOR gates and used interconnecting wires to implement the switching logic. Using the assign operator allows us to describe the hardware in a more compact code. This is implemented in question 1 converting a BCD to Xcess3 code. This example also introduces you to the Xcess code and covers an overview of the advantages of this code.

2. The assignment also covers the implementation of case structures to implement a hardware design while bypassing the K-map step for finding the SOP (Sum of Products) expressions. Question 2 introduces the students to the 2421 code and uses the case structure to implement the conversion.

3. Displaying 4-bit binary input on two seven-segment-displays covering numbers from 0 to 15. This design requires the implementation of comparators, multiplexers (using the conditional assignment) to have the correct numbers displayed.

For each design, you will create a testbench to verify the correct functionality of the design.

This is a group assignment (groups of 2 students). Each group will submit one document listing the contribution of each member.

Objective: Behavioral Models in Verilog - Designing combinational circuits that can perform binary-to-decimal number conversion.

Part (1): Seven Segment Display (Book example page 170 & 171)

We wish to display on 7-segment display the values set by 4 input switches. Your circuit should be able to display the digits from 0 to 9 and should treat the values 1010 to 1111 as BLANK display (all OFF). The LEDs for the 7-segment display are all active low (common anode), which means that the LED would turn ON when its assigned bit value is a logic '0'.

Create a design that assigns the correct output bits to the 7-segments based on the 4- bit input value. Use parameters for the 10 different decimal digits (0 to 9) and the BLANK case.

```
//                          abc_defg
parameter        BLANK     = 7'b111_1111;
parameter        ZERO      = 7'b000_0001;
```

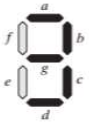


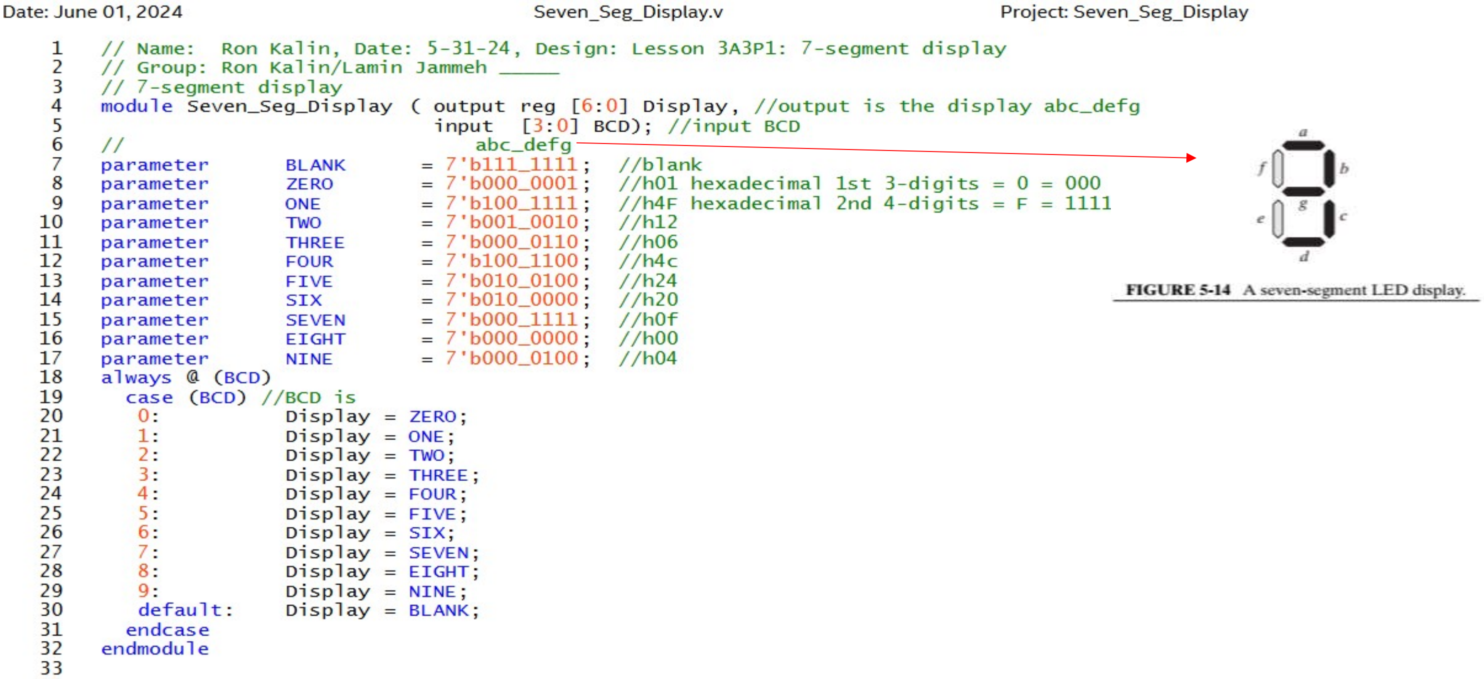
FIGURE 5-14 A seven-segment LED display.

Division of Responsibility

Assignment	Map/ Equations/ Code	TestBench
1	Kalin	Jammeh
2	Kalin	Jammeh
3 Part 1	Kalin	Jammeh
3 Part 2	Kalin	Jammeh

Module code

// Name: Ron Kalin, Date: 5-31-24, Design: Lesson 3A3P1: 7-segment display
// Group: Ron Kalin/Lamin Jammeh _____
// 7-segment display



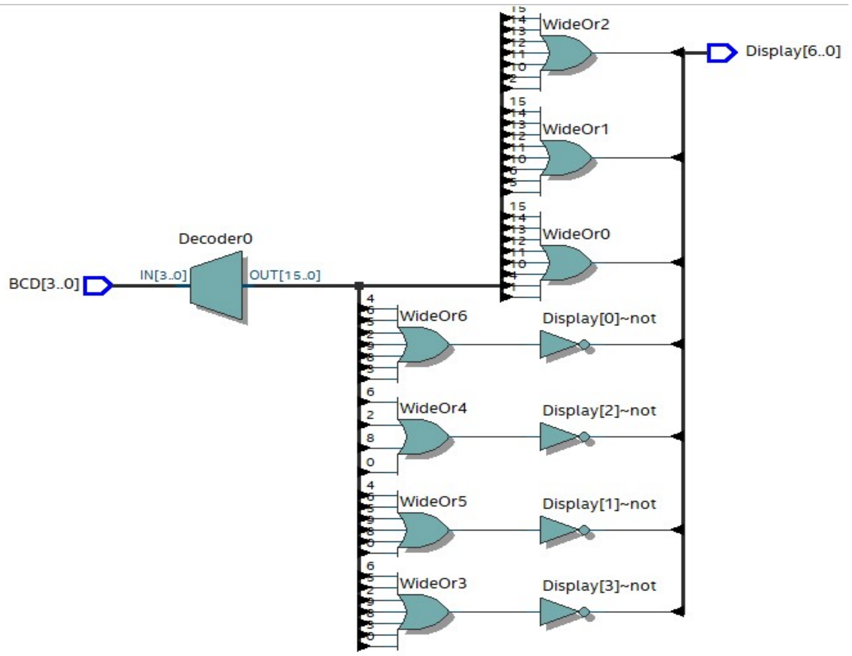
Date: June 01, 2024 Seven_Seg_Display.v Project: Seven_Seg_Display

```
// Name:  Ron Kalin, Date: 5-31-24, Design: Lesson 3A3P1: 7-segment display
// Group: Ron Kalin/Lamin Jammeh _____
// 7-segment display
module Seven_Seg_Display ( output reg [6:0] Display, //output is the display abc_defg
                           input   [3:0] BCD); //input BCD
//                              abc_defg
parameter      BLANK       = 7'b111_1111;   //blank
parameter      ZERO        = 7'b000_0001;   //h01 hexadecimal 1st 3-digits = 0 = 000
parameter      ONE         = 7'b100_1111;   //h4F hexadecimal 2nd 4-digits = F = 1111
parameter      TWO         = 7'b001_0010;   //h12
parameter      THREE       = 7'b000_0110;   //h06
parameter      FOUR        = 7'b100_1100;   //h4c
parameter      FIVE        = 7'b010_0100;   //h24
parameter      SIX         = 7'b010_0000;   //h20
parameter      SEVEN       = 7'b000_1111;   //h0f
parameter      EIGHT       = 7'b000_0000;   //h00
parameter      NINE        = 7'b000_0100;   //h04
always @ (BCD)
  case (BCD) //BCD is
    0:          Display = ZERO;
    1:          Display = ONE;
    2:          Display = TWO;
    3:          Display = THREE;
    4:          Display = FOUR;
    5:          Display = FIVE;
    6:          Display = SIX;
    7:          Display = SEVEN;
    8:          Display = EIGHT;
    9:          Display = NINE;
    default:    Display = BLANK;
  endcase
endmodule
```

FIGURE 5-14 A seven-segment LED display.

endmodule

Netlist RTL Viewer 7-segment display one digit



```
//  5/31/24 testbench 7-segment display
```

Simulation Results

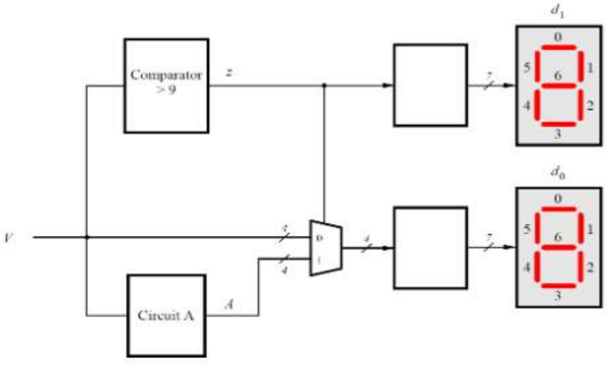
Part 2: 7-segment display, 4-bit input word and output 2-decimal equivalent

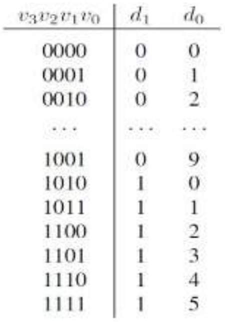
Part (2): Binary-Coded Decimal

You are to design a circuit that converts a four-bit binary number **V** = **$v_3 v_2 v_1 v_0$** into its two-digit decimal equivalent **D** = **$d_1 d_0$**. Table 1 shows the required output values.

Structural Design:

It includes a comparator that checks when the value of V is greater than 9 and uses the output of this comparator in the control of the 7-segment displays. You can use the conditional and comparison operators





$v_3v_2v_1v_0$	d_1	d_0
0000	0	0
0001	0	1
0010	0	2
...	...	...
1001	0	9
1010	1	0
1011	1	1
1100	1	2
1101	1	3
1110	1	4
1111	1	5

Table 1: Binary-to-decimal conversion values.

Figure 1: Partial design of the binary-to-decimal conversion circuit.

Notice that the circuit in Figure 1 includes a 4-bit wide 2-to-1 multiplexer. The purpose of this multiplexer is to drive digit d_0 with the value of V when z = 0, and the value of A when z = 1. To design circuit A consider the following. For the input values V < 9, the circuit A does not matter, because the multiplexer in Figure 1 just selects V in these cases. But for the input values V > 9, the multiplexer will select A. Thus, A has to provide output values that properly implement Table 1 when V > 9.

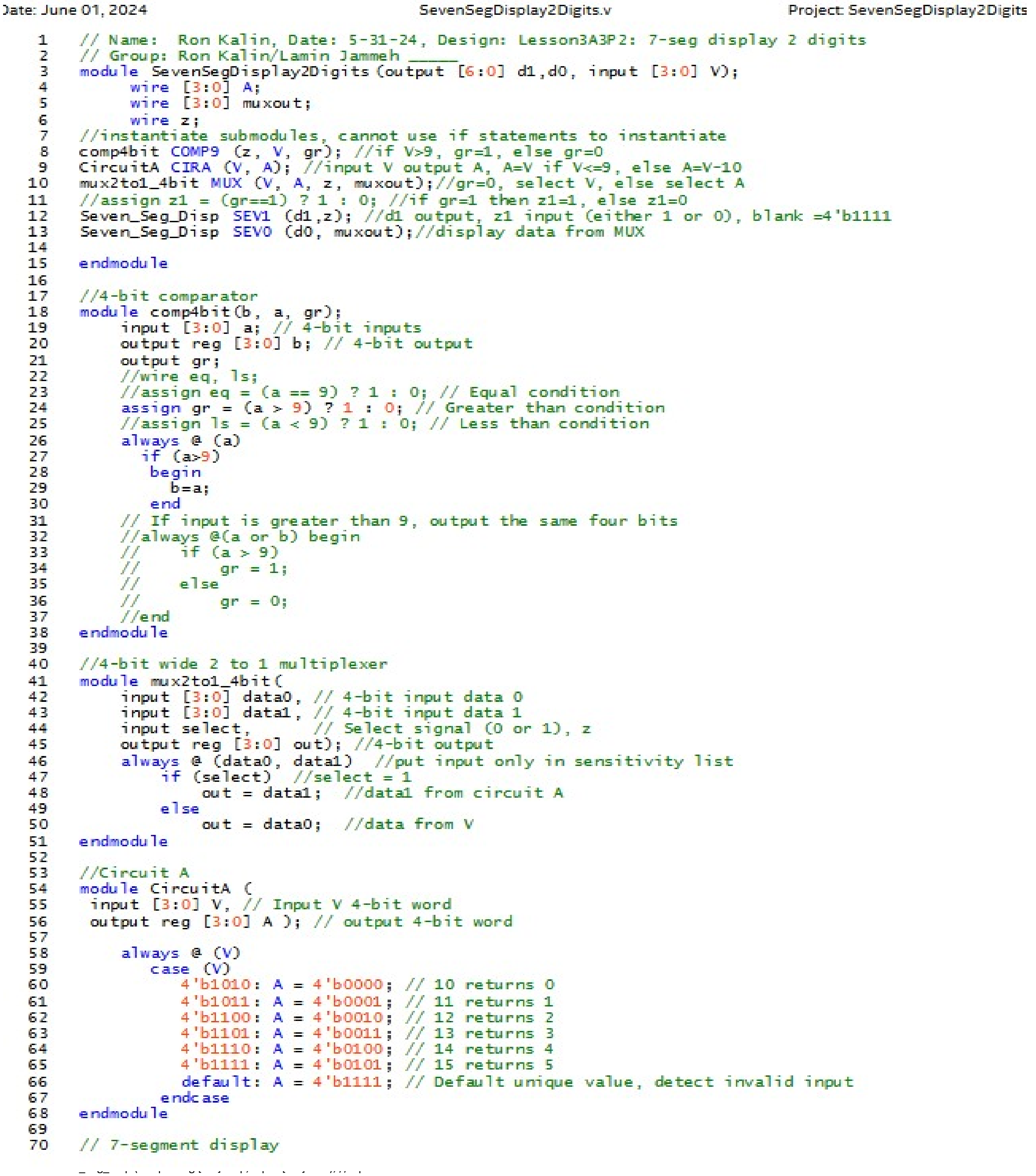
You need to design circuit A so that the input V = 1010 gives an output A = 0000, the input V = 1011 gives the output A = 0001, and the input V = 1111 gives the output A = 0101. Design circuit A using a case structure.

Circuit A

DecV	V	A	DecA
10	1010	0000	0
11	1011	0001	1
12	1100	0010	2
13	1101	0011	3
14	1110	0100	4
15	1111	0101	5

The top code module should instantiate the needed submodules with correct interconnections.

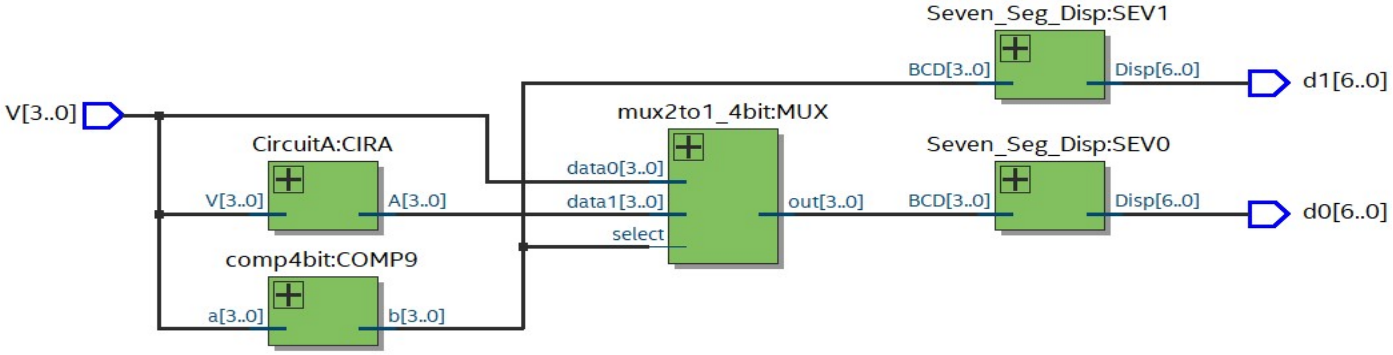
Code



```
// Name:  Ron Kalin, Date: 5-31-24, Design: Lesson3A3P2: 7-seg display 2 digits
// Group: Ron Kalin/Lamin Jammeh _____
module SevenSegDisplay2Digits(output [6:0] d1,d0, input [3:0] V);
     wire [3:0] A;
     wire [3:0] muxout;
     wire z;
//instantiate submodules, cannot use if statements to instantiate
comp4bit COMP9 (z, V, gr); //if V>9, gr=1, else gr=0
CircuitA CIRA (V, A); //input V output A, A=V if V<=9, else A=V-10
mux2to1_4bit MUX (V, A, z, muxout);//gr=0, select V, else select A
//assign z1 = (gr==1) ? 1 : 0; //if gr=1 then z1=1, else z1=0
Seven_Seg_Disp SEV1 (d1,z); //d1 output, z1 input (either 1 or 0), blank =4'b1111
Seven_Seg_Disp SEV0 (d0, muxout);//display data from MUX

endmodule

//4-bit comparator
module comp4bit(b, a, gr);
    input [3:0] a; // 4-bit inputs
    output reg [3:0] b; // 4-bit output
    output gr;
    //wire eq, ls;
    //assign eq = (a == 9) ? 1 : 0; // Equal condition
    assign gr = (a > 9) ? 1 : 0; // Greater than condition
    //assign ls = (a < 9) ? 1 : 0; // Less than condition
    always @ (a)
      if (a>9)
       begin
         b=a;
       end
    // If input is greater than 9, output the same four bits
    //always @(a or b) begin
    //    if (a > 9)
    //        gr = 1;
    //    else
    //        gr = 0;
    //end
endmodule

//4-bit wide 2 to 1 multiplexer
module mux2to1_4bit(
    input [3:0] data0, // 4-bit input data 0
    input [3:0] data1, // 4-bit input data 1
    input select,      // Select signal (0 or 1), z
    output reg [3:0] out); //4-bit output
    always @ (data0, data1)  //put input only in sensitivity list
        if (select)  //select = 1
            out = data1;  //data1 from circuit A
        else
            out = data0;  //data from V
endmodule

//Circuit A
module CircuitA (
 input [3:0] V, // Input V 4-bit word
 output reg [3:0] A ); // output 4-bit word

    always @ (V)
       case (V)
          4'b1010: A = 4'b0000; // 10 returns 0
          4'b1011: A = 4'b0001; // 11 returns 1
          4'b1100: A = 4'b0010; // 12 returns 2
          4'b1101: A = 4'b0011; // 13 returns 3
          4'b1110: A = 4'b0100; // 14 returns 4
          4'b1111: A = 4'b0101; // 15 returns 5
          default: A = 4'b1111; // Default unique value, detect invalid input
        endcase
endmodule

// 7-segment display
```

```
module Seven_Seg_Disp ( output reg [6:0] Disp, input [3:0] BCD); //input BCD
//               output Disp    abc_defg (seven segments, not including dec. pt)
parameter      BLANK       = 7'b111_1111;  //blank
parameter      ZERO        = 7'b000_0001;  //h01 hexadecimal 1st 3-digits = 0 = 000
parameter      ONE         = 7'b100_1111;  //h4F hexadecimal 2nd 4-digits = F = 1111
parameter      TWO         = 7'b001_0010;  //h12
parameter      THREE       = 7'b000_0110;  //h06
parameter      FOUR        = 7'b100_1100;  //h4c
parameter      FIVE        = 7'b010_0100;  //h24
parameter      SIX         = 7'b010_0000;  //h20
parameter      SEVEN       = 7'b000_1111;  //h0f
parameter      EIGHT       = 7'b000_0000;  //h00
parameter      NINE        = 7'b000_0100;  //h04
always @ (BCD)
  case (BCD) //BCD is decimal value
   0:           Disp = ZERO;
   1:           Disp = ONE;
   2:           Disp = TWO;
   3:           Disp = THREE;
   4:           Disp = FOUR;
   5:           Disp = FIVE;
   6:           Disp = SIX;
   7:           Disp = SEVEN;
   8:           Disp = EIGHT;
   9:           Disp = NINE;
   default:     Disp = BLANK;
  endcase
endmodule

```

RTL Viewer


Seven_Seg_Disp:SEV1
BCD[3..0]
Disp[6..0]
d1[6..0]
V[3..0]
mux2to1_4bit:MUX
CircuitA:CIRA
data0[3..0]
Seven_Seg_Disp:SEV0
V[3..0]
A[3..0]
data1[3..0]
out[3..0]
BCD[3..0]
Disp[6..0]
d0[6..0]
select
comp4bit:COMP9
a[3..0]
b[3..0]

Testbench

Simulation