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;
```

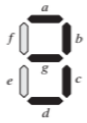


FIGURE 5-14 A seven-segment LED display.

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

```
1   // Name:  Ron Kalin, Date: 5-31-24, Design: Lesson 3A3P1: 7-segment display
2   // Group: Ron Kalin/Lamin Jammeh _____
3   // 7-segment display
4   module Seven_Seg_Display ( output reg [6:0] Display, //output is the display abc_defg
5                              input  [3:0] BCD); //input BCD
6   //                             abc_defg
7   parameter       BLANK     = 7'b111_1111;  //blank
8   parameter       ZERO      = 7'b000_0001;  //h01 hexadecimal 1st 3-digits = 0 = 000
9   parameter       ONE       = 7'b100_1111;  //h4F hexadecimal 2nd 4-digits = F = 1111
10  parameter       TWO       = 7'b001_0010;  //h12
11  parameter       THREE     = 7'b000_0110;  //h06
12  parameter       FOUR      = 7'b100_1100;  //h4c
13  parameter       FIVE      = 7'b010_0100;  //h24
14  parameter       SIX       = 7'b010_0000;  //h20
15  parameter       SEVEN     = 7'b000_1111;  //h0f
16  parameter       EIGHT     = 7'b000_0000;  //h00
17  parameter       NINE      = 7'b000_0100;  //h04
18  always @ (BCD)
19    case (BCD) //BCD is
20      0:          Display = ZERO;
21      1:          Display = ONE;
22      2:          Display = TWO;
23      3:          Display = THREE;
24      4:          Display = FOUR;
25      5:          Display = FIVE;
26      6:          Display = SIX;
27      7:          Display = SEVEN;
28      8:          Display = EIGHT;
29      9:          Display = NINE;
30      default:    Display = BLANK;
31    endcase
32  endmodule
33
```

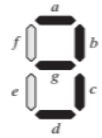
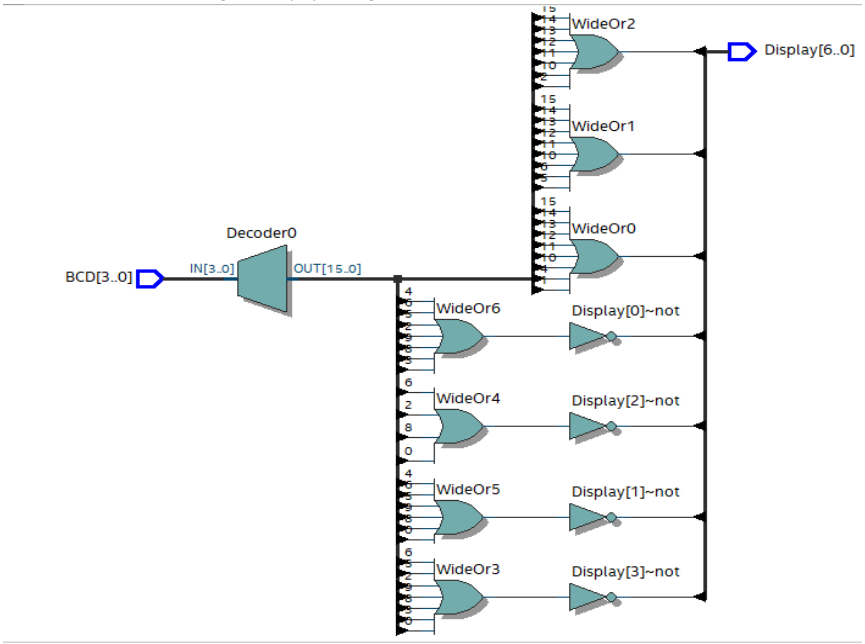


FIGURE 5-14 A seven-segment LED display.

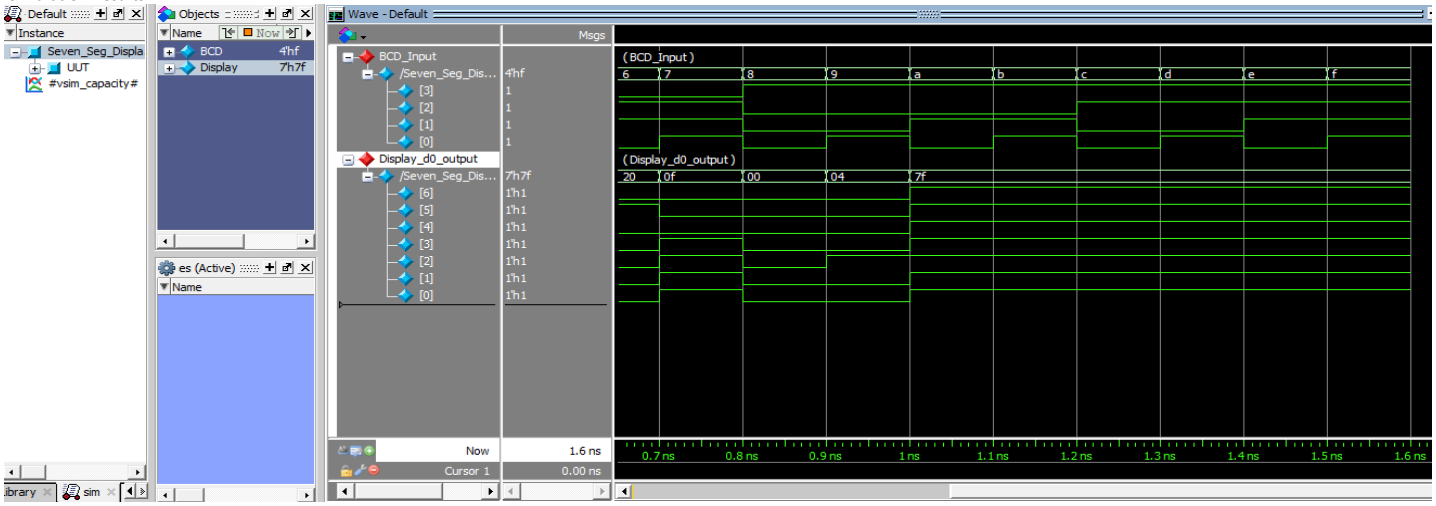
Netlist RTL Viewer 7-segment display one digit



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


```
/*----------------------------------------------------------------------------
Name Lamin Jammeh
CLass: EE417 Summer 2024
Lesson 03 HW Question 3 Part1
Group: Ron Kalin/ Lamin Jammeh
Main design was done by Ron Kalin
TestBench was done by Lamin Jammeh
----------------------------------------------------------------------------*/

//Step1 define a name for the test-bench
module Seven_Seg_Display_tb;

//Step2 define the inputs as registers and outputs as wires
reg [3:0] BCD;
wire [6:0] Display;

//Step3 define the Unit under test UUT with all inputs and outputs
Seven_Seg_Display UUT (
                                .Display(Display),
                                .BCD(BCD)
                                );

//Step4 open an initial block and define all the possible input combination for the BCD
from 0-15 using 4bits
initial
   begin
          BCD = 4'b_0000;  //0
    #100 BCD = 4'b_0001;  //1
    #100 BCD = 4'b_0010;  //2
    #100 BCD = 4'b_0011;  //3
    #100 BCD = 4'b_0100;  //4
    #100 BCD = 4'b_0101;  //5
    #100 BCD = 4'b_0110;  //6
    #100 BCD = 4'b_0111;  //7
    #100 BCD = 4'b_1000;  //8
    #100 BCD = 4'b_1001;  //9
    //Needs a second circuit to show the tenth digits
    #100 BCD = 4'b_1010;  //10
    #100 BCD = 4'b_1011;  //11
    #100 BCD = 4'b_1100;  //12
    #100 BCD = 4'b_1101;  //13
    #100 BCD = 4'b_1110;  //14
    #100 BCD = 4'b_1111;  //15
    #100;
      end
//Display the results
initial begin
        $monitor("BCD = %b:    Decimal_Input = %d:      Display = %b:   Hex_Output =%h:", BCD,
BCD, Display, Display);
    end
endmodule
```

Simulation Results




```
# run -all
# BCD = 0000:  Decimal_Input =  0:  Display = 0000001:  Hex_Output =01:
# BCD = 0001:  Decimal_Input =  1:  Display = 1001111:  Hex_Output =4f:
# BCD = 0010:  Decimal_Input =  2:  Display = 0010010:  Hex_Output =12:
# BCD = 0011:  Decimal_Input =  3:  Display = 0000110:  Hex_Output =06:
# BCD = 0100:  Decimal_Input =  4:  Display = 1001100:  Hex_Output =4c:
# BCD = 0101:  Decimal_Input =  5:  Display = 0100100:  Hex_Output =24:
# BCD = 0110:  Decimal_Input =  6:  Display = 0100000:  Hex_Output =20:
# BCD = 0111:  Decimal_Input =  7:  Display = 0001111:  Hex_Output =0f:
# BCD = 1000:  Decimal_Input =  8:  Display = 0000000:  Hex_Output =00:
# BCD = 1001:  Decimal_Input =  9:  Display = 0000100:  Hex_Output =04:
# BCD = 1010:  Decimal_Input = 10:  Display = 1111111:  Hex_Output =7f:
# BCD = 1011:  Decimal_Input = 11:  Display = 1111111:  Hex_Output =7f:
# BCD = 1100:  Decimal_Input = 12:  Display = 1111111:  Hex_Output =7f:
# BCD = 1101:  Decimal_Input = 13:  Display = 1111111:  Hex_Output =7f:
# BCD = 1110:  Decimal_Input = 14:  Display = 1111111:  Hex_Output =7f:
# BCD = 1111:  Decimal_Input = 15:  Display = 1111111:  Hex_Output =7f:
```

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

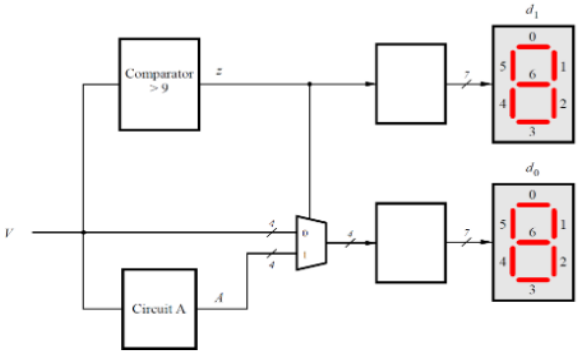


Figure 1: Partial design of the binary-to-decimal conversion circuit.

$v_3 v_2 v_1 v_0$	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.

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

Date: June 02, 2024 SevenSegDisplay2Digits.v Project: SevenSegDisplay2Digits

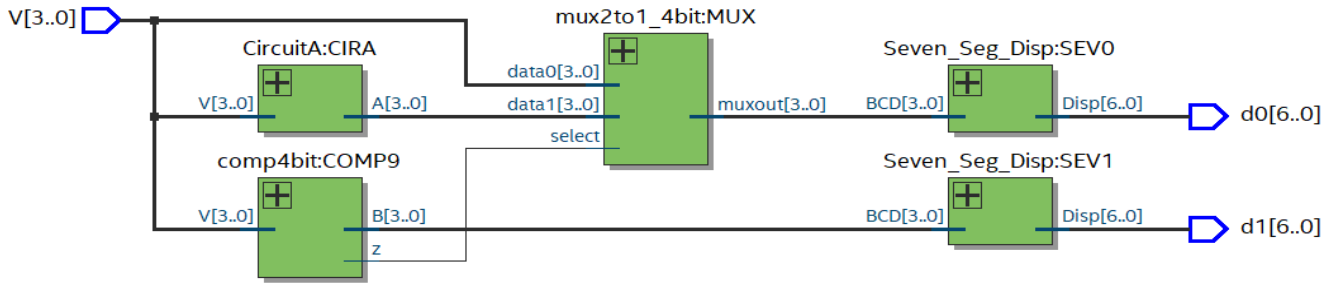
```
1   // Name:  Ron Kalin, Date: 5-31-24, Design: Lesson3A3P2: 7-seg display 2 digits
2   // Group: Ron Kalin/Lamin Jammeh _____
3   module SevenSegDisplay2Digits (output [6:0] d1,d0, input [3:0] V);
4        wire [3:0] A;
5        wire [3:0] muxout;
6        wire [3:0] z;
7        wire gr;
8   //instantiate submodules, cannot use if statements to instantiate
9   comp4bit COMP9 (z, V, gr); //if V>9, gr=1, else gr=0
10  CircuitA CIRA (V, A); //input V output A, A=V if V<=9, else A=V-10
11  mux2to1_4bit MUX (V, A, gr, muxout);//z=0, select V, else select A
12  //assign z1 = (z==1) ? z : 4'b1111; //if gr=1 then z1=1, else z1=blank
13  Seven_Seg_Disp SEV1 (d1,z); //d1 output, z1 input (either 1 or 0), blank =4'b1111
14  Seven_Seg_Disp SEV0 (d0, muxout);//display data from MUX
15
16  endmodule
17
18  //4-bit comparator
19  module comp4bit(B, V, z);
20      input [3:0] V; // 4-bit inputs
21      output reg [3:0] B; // 4-bit output
22      output z;
23      //wire eq, ls;
24      //assign eq = (a == 9) ? 1 : 0; // Equal condition
25      assign z = (V > 9) ? 1 : 0; // Greater than condition
26      //assign ls = (a < 9) ? 1 : 0; // Less than condition
27      always @ (V)
28        if (V>9) begin
29          //V>9 so display 1
30           B=4'b0001;
31           //b=V;
32           end
33        else begin //V<9 so blank
34           B=4'b1111;
35           end
36  endmodule
37
38  //4-bit wide 2 to 1 multiplexer
39  module mux2to1_4bit(
40      input [3:0] data0, // 4-bit input data 0
41      input [3:0] data1, // 4-bit input data 1
42      input select,      // Select signal (0 or 1), z
43      output reg [3:0] muxout); //4-bit output
44      always @ (data0, data1)  //put input only in sensitivity list
45          if (select)  //select = 1
46              muxout = data1;  //data1 from circuit A
47          else
48              muxout = data0;  //data from V
49  endmodule
50
51  //Circuit A
52  module CircuitA (
53   input [3:0] V, // Input V 4-bit word
54   output reg [3:0] A ); // output 4-bit word
55
56      always @ (V)
57         case (V)
58            4'b1010: A = 4'b0000; // 10 returns 0
59            4'b1011: A = 4'b0001; // 11 returns 1
60            4'b1100: A = 4'b0010; // 12 returns 2
61            4'b1101: A = 4'b0011; // 13 returns 3
62            4'b1110: A = 4'b0100; // 14 returns 4
63            4'b1111: A = 4'b0101; // 15 returns 5
64            default: A = 4'b1111; // Default unique value, detect invalid input
65          endcase
66  endmodule
```

```
67
68  // 7-segment display
69  module Seven_Seg_Disp ( output reg [6:0] Disp, input [3:0] BCD); //input BCD
70  //               output Disp   abc_defg (seven segments, not including dec. pt)
```



```
71  parameter       BLANK       = 7'b111_1111;  //blank
72  parameter       ZERO        = 7'b000_0001;  //h01 hexadecimal 1st 3-digits = 0 = 000
73  parameter       ONE         = 7'b100_1111;  //h4F hexadecimal 2nd 4-digits = F = 1111
74  parameter       TWO         = 7'b001_0010;  //h12
75  parameter       THREE       = 7'b000_0110;  //h06
76  parameter       FOUR        = 7'b100_1100;  //h4c
77  parameter       FIVE        = 7'b010_0100;  //h24
78  parameter       SIX         = 7'b010_0000;  //h20
79  parameter       SEVEN       = 7'b000_1111;  //h0f
80  parameter       EIGHT       = 7'b000_0000;  //h00
81  parameter       NINE        = 7'b000_0100;  //h04
82  always @ (BCD)
83    case (BCD) //BCD is decimal value
84      0:          Disp = ZERO;
85      1:          Disp = ONE;
86      2:          Disp = TWO;
87      3:          Disp = THREE;
88      4:          Disp = FOUR;
89      5:          Disp = FIVE;
90      6:          Disp = SIX;
91      7:          Disp = SEVEN;
92      8:          Disp = EIGHT;
93      9:          Disp = NINE;
94      default:    Disp = BLANK;
95    endcase
96  endmodule
97
```

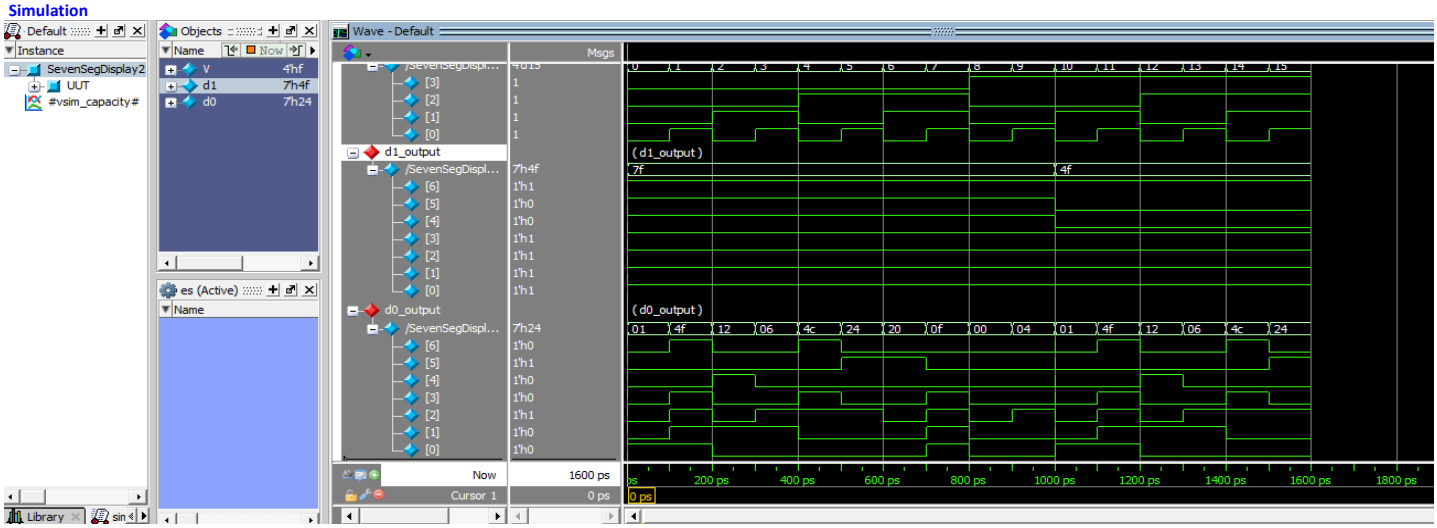
RTL Viewer


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

Testbench

```
/*-------------------------------------------------------------------------
Name Lamin Jammeh
CLass: EE417 Summer 2024
Lesson 03 HW Question 3 Part1
Group: Ron Kalin/ Lamin Jammeh
Main design was done by Ron Kalin
TestBench was done by Lamin Jammeh
-------------------------------------------------------------------------*/

//Step1 define a name for the test-bench
module SevenSegDisplay2Digits_tb;

//Step2 define the inputs as registers and outputs as wires
reg [3:0] V;
wire [6:0] d1, d0;

//Step3 define the unit under test UUT with all inputs and outputs
SevenSegDisplay2Digits UUT (
                                .d1(d1),
                                .d0(d0),
                                .V(V)
                                );

//Step4 open an initial block and define all the possible input combination for the BCD from 0-15 using 4bits
initial
   begin
            V = 4'b0000;  //0
     #100 V = 4'b0001;  //1
     #100 V = 4'b0010;  //2
     #100 V = 4'b0011;  //3
     #100 V = 4'b0100;  //4
     #100 V = 4'b0101;  //5
     #100 V = 4'b0110;  //6
     #100 V = 4'b0111;  //7
     #100 V = 4'b1000;  //8
     #100 V = 4'b1001;  //9

     #100 V = 4'b1010;  //10
     #100 V = 4'b1011;  //11
     #100 V = 4'b1100;  //12
     #100 V = 4'b1101;  //13
     #100 V = 4'b1110;  //14
     #100 V = 4'b1111;  //15
     #100;
      end
//Display the results
initial begin
         $display("_____________________________output_didplay = --d1---_--d0---");
         $monitor("V = %b:  Decimal_Input = %d:     Output_Display = %b_%b ", V, V, d1, d0);
    end
endmodule
```

Simulation




```
#
# add wave *
# view structure
# .main_pane.structure.interior.cs.body.struct
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all
#                                output_didplay = --d1---_--d0---
# V = 0000:  Decimal_Input =  0:  Output_Display = 1111111_0000001
# V = 0001:  Decimal_Input =  1:  Output_Display = 1111111_1001111
# V = 0010:  Decimal_Input =  2:  Output_Display = 1111111_0010010
# V = 0011:  Decimal_Input =  3:  Output_Display = 1111111_0000110
# V = 0100:  Decimal_Input =  4:  Output_Display = 1111111_1001100
# V = 0101:  Decimal_Input =  5:  Output_Display = 1111111_0100100
# V = 0110:  Decimal_Input =  6:  Output_Display = 1111111_0100000
# V = 0111:  Decimal_Input =  7:  Output_Display = 1111111_0001111
# V = 1000:  Decimal_Input =  8:  Output_Display = 1111111_0000000
# V = 1001:  Decimal_Input =  9:  Output_Display = 1111111_0000100
# V = 1010:  Decimal_Input = 10:  Output_Display = 1001111_0000001
# V = 1011:  Decimal_Input = 11:  Output_Display = 1001111_1001111
# V = 1100:  Decimal_Input = 12:  Output_Display = 1001111_0010010
# V = 1101:  Decimal_Input = 13:  Output_Display = 1001111_0000110
# V = 1110:  Decimal_Input = 14:  Output_Display = 1001111_1001100
# V = 1111:  Decimal_Input = 15:  Output_Display = 1001111_0100100

VSIM 2>
```