HDL Code Solutions

# 1. Decoder Chip

CHIP Decoder<your group number> {  
 IN A, B, C, D;  
 OUT a, b, c, d, e, f, g;  
 PARTS:  
 // Decode each 4-bit input (A, B, C, D) to the corresponding 7-segment output  
 Mux4Way16(a, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, A, B, C, D);  
 Mux4Way16(b, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, A, B, C, D);  
 Mux4Way16(c, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, A, B, C, D);  
 Mux4Way16(d, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, A, B, C, D);  
 Mux4Way16(e, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, A, B, C, D);  
 Mux4Way16(f, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, A, B, C, D);  
 Mux4Way16(g, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, A, B, C, D);  
}

# 2. MultiDecoder Chip

CHIP MultiDecoder {  
 IN A[4], B[4]; // Two 4-bit inputs for the hexadecimal digits  
 OUT DigitA[7], DigitB[7]; // Two 7-segment outputs for the two digits  
 PARTS:  
 // Instantiate two Decoder chips for each input A[4] and B[4]  
 Decoder ADecoder {  
 IN A[0], A[1], A[2], A[3];  
 OUT DigitA[0], DigitA[1], DigitA[2], DigitA[3], DigitA[4], DigitA[5], DigitA[6];  
 }  
   
 Decoder BDecoder {  
 IN B[0], B[1], B[2], B[3];  
 OUT DigitB[0], DigitB[1], DigitB[2], DigitB[3], DigitB[4], DigitB[5], DigitB[6];  
 }  
}

# 3. Segment c Logic Implementation

## 1. Truth Table for Segment c

Hex A B C D c  
0 0 0 0 0 1  
1 0 0 0 1 1  
2 0 0 1 0 0  
3 0 0 1 1 1  
4 0 1 0 0 1  
5 0 1 0 1 1  
6 0 1 1 0 1  
7 0 1 1 1 1  
8 1 0 0 0 1  
9 1 0 0 1 1  
A 1 0 1 0 1  
B 1 0 1 1 0  
C 1 1 0 0 1  
D 1 1 0 1 1  
E 1 1 1 0 1  
F 1 1 1 1 0

## 2. Karnaugh Map for Segment c

We use a 4-variable K-map: A B on the rows and C D on the columns. Group 1s to minimize.  
K-map for c (1 = ON, 0 = OFF):  
  
 CD  
 00 01 11 10  
AB  
00 1 1 1 0  
01 1 1 1 1  
11 1 1 0 1  
10 1 1 0 1  
  
From this, we derive prime implicants and minimize.

## 3. Minimized Boolean Expression for Segment c

After grouping ones from the K-map, we derive the minimized Boolean expression:  
  
c = A'B' + A'BC'D' + A'BCD + AB'C' + AB'D + AB'C  
  
This equation is minimized from 16 input possibilities (0–F) and gives logic high (1) for segment c in all necessary hex digits.

## 4. Verilog HDL Implementation for Segment c

module segment\_c (  
 input wire A, B, C, D,  
 output wire c  
);  
  
assign c = (~A & ~B) | // A'B'  
 (~A & B & ~C & ~D) | // A'BC'D'  
 (~A & B & C & D) | // A'BCD  
 (A & ~B & ~C) | // AB'C'  
 (A & ~B & D) | // AB'D  
 (A & ~B & C); // AB'C  
  
endmodule  
  
// Testbench Snippet  
module test\_segment\_c;  
 reg A, B, C, D;  
 wire c;  
  
 segment\_c uut (.A(A), .B(B), .C(C), .D(D), .c(c));  
  
 initial begin  
 $display("A B C D | c");  
 for (integer i = 0; i < 16; i = i + 1) begin  
 {A, B, C, D} = i;  
 #10 $display("%b %b %b %b | %b", A, B, C, D, c);  
 end  
 end  
endmodule

HDL Code Solutions

# 1. Decoder Chip

CHIP Decoder<your group number> {  
 IN A, B, C, D;  
 OUT a, b, c, d, e, f, g;  
 PARTS:  
 // Decode each 4-bit input (A, B, C, D) to the corresponding 7-segment output  
 Mux4Way16(a, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, A, B, C, D);  
 Mux4Way16(b, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, A, B, C, D);  
 Mux4Way16(c, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, A, B, C, D);  
 Mux4Way16(d, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, A, B, C, D);  
 Mux4Way16(e, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, A, B, C, D);  
 Mux4Way16(f, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, A, B, C, D);  
 Mux4Way16(g, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, A, B, C, D);  
}

# 2. MultiDecoder Chip

CHIP MultiDecoder {  
 IN A[4], B[4]; // Two 4-bit inputs for the hexadecimal digits  
 OUT DigitA[7], DigitB[7]; // Two 7-segment outputs for the two digits  
 PARTS:  
 // Instantiate two Decoder chips for each input A[4] and B[4]  
 Decoder ADecoder {  
 IN A[0], A[1], A[2], A[3];  
 OUT DigitA[0], DigitA[1], DigitA[2], DigitA[3], DigitA[4], DigitA[5], DigitA[6];  
 }  
   
 Decoder BDecoder {  
 IN B[0], B[1], B[2], B[3];  
 OUT DigitB[0], DigitB[1], DigitB[2], DigitB[3], DigitB[4], DigitB[5], DigitB[6];  
 }  
}