Code for the 4-bit counter implementation on DE-10 Lite FPGA board.

Hello, sir hope this message finds you well this code we have generated and not copied from anywhere after many hits and trials and then made actual working code. We wanted to add more functions to it but it was getting more complex so we stopped there and finished this project.

Work distribution of the team was done like

1. **Harsh Shivdarshan Wanwale (50%) [2020BEC127]**

Logic Building For This Code, Variable Register Defining, Case Statement, Ppt Making

2. **Sanskruti Vynkatrao Shewale(25%) [2020BEC037]**

Input Output Pin Assignments On the Software, Some Help in Code, Creating Work Distribution

3. **Devesh Gadhave (25%)[2020BECA25]**

Implementing Created Logic On Software and Correcting the Errors, Ideas for Further Functionality in Counter

Working code:

```
//firstly defined the module, inputs, and outputs
module counter (
  input clk,
  input wire rst,
  output [6:0] hour1,
  output [6:0] hour2,
  output [6:0] min1,
  output [6:0] min2,
  output [6:0] sec1,
  output [6:0] sec2
);
// This registers used here are like variable because we cant assign multiple values to contant drivers
  reg [26:0] counter = 0;
  reg [3:0] count1 = 0;
  reg [3:0] count2 = 0;
```

```
 reg [3:0] count3 = 0;
 reg [3:0] count4 = 0;
 reg [3:0] count5 = 0;
 reg [3:0] count6 = 0;
 reg [6:0] h1 = 0;
 reg [6:0] h2 = 0;
 reg [6:0] m1 = 0;
 reg [6:0] m2 = 0;
 reg [6:0] s1 = 0;
 reg [6:0] s2 = 0;
```

//This is the main loop-like structure similar to C++ for counting the numbers

At every positive edge of the clock, this loop begins and execute the if else conditions

```
always @(posedge clk) begin
 if (rst) begin
  if (counter == 50_000_00) begin
   counter <= 0;
                count1 <= count1 + 1;
   if (count1 == 9) begin
                 count2 <= count2 + 1;
    count1 <= 0;
                 if (count2 == 0) begin
                  count2 <= 0;
                        count3 <= count3 + 1;
                        if (count3 == 9) begin
                   count3 <= 0;
                              count4 <= count4 + 1;
                         if (count4 == 5) begin
                    count4 <= 0;
                                 count5 <= count5 + 1;
                                 if (count5 == 9) begin
                      count5 <= 0;
```

```
                                    count6 <= count6 + 1;
                                    if (count6 == 5) begin
                        count6 <= 0;
                              end
                         end
                        end
                       end
                    end
                end
        end
    else begin
       counter <= counter + 1;
    end
    end
          else begin
           count1 = 0;
```

// this else statement to reset the values of each counter by which we can reset all values to 000

```
     count2 = 0;
     count3 = 0;
     count4 = 0;
     count5 = 0;
     count6 = 0;
                  end
 end
```

// from this onwards these all the always block have case statement when there is increment in counts this case state ments increses their default value is 0 and after reaching to their maximum values they rest to 00 and again this loop starts

```
  always @(count6) begin
    case (count6)
     4'b0000: h1 <= 7'b1000000; // 0
     4'b0001: h1 <= 7'b1111001; // 1
     4'b0010: h1 <= 7'b0100100; // 2
```

```
    4'b0011: h1 <= 7'b0110000; // 3
    4'b0100: h1 <= 7'b0011001; // 4
    4'b0101: h1 <= 7'b0010010; // 5
    4'b0110: h1 <= 7'b0000010; // 6
    4'b0111: h1 <= 7'b1111000; // 7
    4'b1000: h1 <= 7'b0000000; // 8
    4'b1001: h1 <= 7'b0011000; // 9
    default: h1 <= 7'b1000000; // Display nothing (all segments off)
  endcase
end
always @(count5) begin
  case (count5)
    4'b0000: h2 <= 7'b1000000; // 0
    4'b0001: h2 <= 7'b1111001; // 1
    4'b0010: h2 <= 7'b0100100; // 2
    4'b0011: h2 <= 7'b0110000; // 3
    4'b0100: h2 <= 7'b0011001; // 4
    4'b0101: h2 <= 7'b0010010; // 5
    4'b0110: h2 <= 7'b0000010; // 6
    4'b0111: h2 <= 7'b1111000; // 7
    4'b1000: h2 <= 7'b0000000; // 8
    4'b1001: h2 <= 7'b0011000; // 9
    default: h2 <= 7'b1000000; // Display nothing (all segments off)
  endcase
end
always @(count4) begin
  case (count4)
    4'b0000: m1 <= 7'b1000000; // 0
    4'b0001: m1 <= 7'b1111001; // 1
    4'b0010: m1 <= 7'b0100100; // 2
    4'b0011: m1 <= 7'b0110000; // 3
    4'b0100: m1 <= 7'b0011001; // 4
```

```
    4'b0101: m1 <= 7'b0010010; // 5
    4'b0110: m1 <= 7'b0000010; // 6
    4'b0111: m1 <= 7'b1111000; // 7
    4'b1000: m1 <= 7'b0000000; // 8
    4'b1001: m1 <= 7'b0011000; // 9
    default: m1 <= 7'b1000000; // Display nothing (all segments off)
  endcase
end

 always @(count3) begin
  case (count3)
    4'b0000: m2 <= 7'b1000000; // 0
    4'b0001: m2 <= 7'b1111001; // 1
    4'b0010: m2 <= 7'b0100100; // 2
    4'b0011: m2 <= 7'b0110000; // 3
    4'b0100: m2 <= 7'b0011001; // 4
    4'b0101: m2 <= 7'b0010010; // 5
    4'b0110: m2 <= 7'b0000010; // 6
    4'b0111: m2 <= 7'b1111000; // 7
    4'b1000: m2 <= 7'b0000000; // 8
    4'b1001: m2 <= 7'b0011000; // 9
    default: m2 <= 7'b1000000; // Display nothing (all segments off)
  endcase
end

 always @(count2) begin
  case (count2)
    4'b0000: s1 <= 7'b1000000; // 0
    4'b0001: s1 <= 7'b1111001; // 1
    4'b0010: s1 <= 7'b0100100; // 2
    4'b0011: s1 <= 7'b0110000; // 3
    4'b0100: s1 <= 7'b0011001; // 4
```

```
    4'b0101: s1 <= 7'b0010010; // 5
    4'b0110: s1 <= 7'b0000010; // 6
    4'b0111: s1 <= 7'b1111000; // 7
    4'b1000: s1 <= 7'b0000000; // 8
    4'b1001: s1 <= 7'b0011000; // 9
    default: s1 <= 7'b1000000; // Display nothing (all segments off)
  endcase
 end
 always @(count1) begin
  case (count1)
    4'b0000: s2 <= 7'b1000000; // 0
    4'b0001: s2 <= 7'b1111001; // 1
    4'b0010: s2 <= 7'b0100100; // 2
    4'b0011: s2 <= 7'b0110000; // 3
    4'b0100: s2 <= 7'b0011001; // 4
    4'b0101: s2 <= 7'b0010010; // 5
    4'b0110: s2 <= 7'b0000010; // 6
    4'b0111: s2 <= 7'b1111000; // 7
    4'b1000: s2 <= 7'b0000000; // 8
    4'b1001: s2 <= 7'b0011000; // 9
    default: s2 <= 7'b1000000; // Display nothing (all segments off)
  endcase
 end
```

//Here are the assignments from which we assign the variable registers to the output .there is no direct output given because many meshes will be created hence when the results are ready they are fed to the final outputs

```
 assign hour1 = h1;
 assign hour2= h2;
 assign min1 = m1;
 assign min2 = m2;
 assign sec1 = s1;
 assign sec2 = s2;
```

```
endmodule
```