Lab Code [10 points]
Filename: changeMachine.sv
AndrewID: tbeasley

```
`default_nettype none


/*Compares Paid and Cost to see if Paid is (<, >, or =) Cost and checks if
they are both 0*/
module ModifiedMagComparator
  (input logic [3:0] A, B,
  output logic [3:0] options);

  assign options[3] = (A == B) ? 1 : 0;
  assign options[2] = (A < B) ? 1 : 0;
  assign options[1] = (A > B) ? 1 : 0;
  assign options[0] = (A == 0 && B == 0) ? 1 : 0;

endmodule: ModifiedMagComparator


/*Checks which path we want to take since there are only 4 possible cases
that could happen based on the comparison that happens in the MagComparator*/
module DecidePath
  (input logic [2:0] comp,
  input logic checkZero,
  output logic [1:0] path);

  always_comb begin
    if (comp[2] && checkZero)
      path = 2'b00;
    else if (comp[2])
      path = 2'b10;
    else if (comp[1])
      path = 2'b01;
    else
      path = 2'b00;
  end

endmodule: DecidePath


//Calculates the amount of change that is leftover after the Zorgian pays
module CalcChange
  (input logic [1:0] Pentagons, Triangles, Circles, path,
  input logic [3:0] Money, Price,
  output logic [3:0] change,
  output logic [5:0] availableCoins);

  logic en;

  Skip decide(.path, .enable(en));
  FindChange amountleft(.en, .Pentagons, .Triangles, .Circles,
                        .Money, .Price, .availableCoins, .change);

endmodule: CalcChange


/*Gives the option to skip all of the calculations based on which path we are
on*/
module Skip
  (input logic [1:0] path,
  output logic enable);

  assign enable = (path == 2'b00) ? 1 : 0;

endmodule: Skip


//Submodule that calculates the change
module FindChange
  (input logic en,
  input logic [1:0] Pentagons, Triangles, Circles,
  input logic [3:0] Money, Price,
  output logic [3:0] change,
  output logic [5:0] availableCoins);

  assign change = (en) ? Money - Price : 0;
  assign availableCoins[5:4] = Pentagons;
  assign availableCoins[3:2] = Triangles;
  assign availableCoins[1:0] = Circles;

endmodule: FindChange


//Calculates the first coin that can be produced from the change leftover
module CalcFirstCoin
  (input logic [3:0] change,
  input logic [5:0] availableCoins,
  output logic [2:0] first,
  output logic [3:0] restOfChange,
  output logic [5:0] coinsRemain);


  FindFirst coin(.change, .availableCoins, .first, .restOfChange);
  RemoveCoin remove(.coinType(first), .availableCoins, .coinsRemain);


endmodule: CalcFirstCoin


//Submodule that finds the first coin based on the change
module FindFirst
  (input logic [3:0] change,
  input logic [5:0] availableCoins,
  output logic [2:0] first,
  output logic [3:0] restOfChange);

  always_comb begin
    if (availableCoins[5:4] > 2'b00 && change >= 3'b101)
      first = 3'b101;
    else if (availableCoins[3:2] > 2'b00 && change >= 3'b011)
      first = 3'b011;
    else if (availableCoins[1:0] > 2'b00 && change >= 3'b001)
      first = 3'b001;
    else
      first = 3'b000;

    restOfChange = change - first;
  end

endmodule: FindFirst


/*Removes the coin taken when calculating the first coin from the group of
coins that are available in the coinbox*/
module RemoveCoin
  (input logic [2:0] coinType,
  input logic [5:0] availableCoins,
  output logic [5:0] coinsRemain);

  always_comb begin
    unique case (coinType)
      3'b000: coinsRemain = availableCoins;
      3'b001: begin
              coinsRemain[5:4] = availableCoins[5:4];
              coinsRemain[3:2] = availableCoins[3:2];
              coinsRemain[1:0] = availableCoins[1:0] - 2'b01;
              end
      3'b011: begin
              coinsRemain[5:4] = availableCoins[5:4];
              coinsRemain[3:2] = availableCoins[3:2] - 2'b01;
              coinsRemain[1:0] = availableCoins[1:0];
              end
      3'b101: begin
              coinsRemain[5:4] = availableCoins[5:4] - 2'b01;
              coinsRemain[3:2] = availableCoins[3:2];
              coinsRemain[1:0] = availableCoins[1:0];
              end
    endcase
  end

endmodule: RemoveCoin


//Calculates the second coin that can be produced from the change leftover
module CalcSecondCoin
  (input logic [3:0] leftoverChange,
  input logic [5:0] coinsRemain,
  output logic [2:0] second,
  output logic [3:0] remaining,
  output logic NotEnoughChange);


  FindFirst coin2(.change(leftoverChange), .availableCoins(coinsRemain),
                  .first(second), .restOfChange(remaining));

  assign NotEnoughChange = (remaining > 0) ? 1 : 0;

endmodule: CalcSecondCoin


//Contains all of the modules for convience when using them in the top module
module ChangeMachine
  (input logic [3:0] Cost, Paid,
  input logic [1:0] Pentagons, Triangles, Circles,
  output logic [2:0] FirstCoin, SecondCoin,
  output logic ExactAmount, NotEnoughChange, CoughUpMore,
  output logic [3:0] Remaining);

  logic [3:0] options;
  logic [1:0] wire_a;
  logic [3:0] wire_b;
  logic [5:0] wire_c;
  logic [3:0] wire_d;
  logic [5:0] wire_e;



  ModifiedMagComparator compare(.A(Paid), .B(Cost), .options(options));
  DecidePath findpath(.comp(options[3:1]), .checkZero(options[0]),
                      .path(wire_a));
  CalcChange calculate1(.Pentagons, .Triangles, .Circles, .path(wire_a),
                        .Money(Paid), .Price(Cost), .change(wire_b),
                        .availableCoins(wire_c));
  CalcFirstCoin calculate2(.change(wire_b),
                           .availableCoins(wire_c),
                           .first(FirstCoin), .restOfChange(wire_d),
                           .coinsRemain(wire_e));
  CalcSecondCoin calculate3(.leftoverChange(wire_d),
                            .coinsRemain(wire_e),
                            .second(SecondCoin), .remaining(Remaining),
                            .NotEnoughChange(NotEnoughChange));
  assign ExactAmount = wire_a[1];
  assign CoughUpMore = wire_a[0];

endmodule: ChangeMachine


//Test the top module that has all modules in it
module ChangeMachine_test;

logic [3:0] Cost, Paid;
logic [1:0] Pentagons, Triangles, Circles;
logic [2:0] FirstCoin, SecondCoin;
logic ExactAmount, NotEnoughChange, CoughUpMore;
logic [3:0] Remaining;

ChangeMachine DUT(.*);

initial begin

  $monitor($time, {" Cost = %b Paid = %b\n\t\t Pentagons = %b Triangles = %b",
          " Circles = %b\n\t\t First = %b Second = %b\n\t\t Exact = %b ",
          "Enough = %b More = %b\n\t\t Remain = %b"},
          Cost, Paid, Pentagons, Triangles, Circles, FirstCoin, SecondCoin,
          ExactAmount, NotEnoughChange, CoughUpMore,
          Remaining);

  //Normal Cases
  Cost = 4'b1111;
  Paid = 4'b0001;
  Pentagons = 2'b10;
  Triangles = 2'b10;
  Circles = 2'b10;
  #10 Cost = 4'b1111;
      Paid = 4'b1111;
      Pentagons = 2'b00;
      Triangles = 2'b10;
      Circles = 3'b010;
  #10 Cost = 4'b0001;
      Paid = 4'b1111;
      Pentagons = 2'b10;
      Triangles = 2'b10;
      Circles = 2'b10;
  #10 Cost = 4'b0001;
      Paid = 4'b1111;
      Pentagons = 2'b01;
      Triangles = 2'b01;
      Circles = 2'b10;
  #10 Cost = 4'b0001;
      Paid = 4'b1111;
      Pentagons = 2'b00;
      Triangles = 2'b00;
      Circles = 2'b00;
  #10 Cost = 4'b1100;
      Paid = 4'b0000;
      Pentagons = 2'b10;
      Triangles = 2'b10;
      Circles = 2'b01;
  #10 Cost = 4'b1001;
      Paid = 4'b1111;
      Pentagons = 2'b10;
      Triangles = 2'b10;
      Circles = 2'b01;

  //Edge Cases
  #10 Cost = 4'b0000;
      Paid = 4'b0000;
      Pentagons = 2'b00;
      Triangles = 2'b00;
      Circles = 2'b10;
  #10 Cost = 4'b0000;
      Paid = 4'b0011;
      Pentagons = 2'b00;
      Triangles = 2'b01;
      Circles = 2'b10;
  #10 Cost = 4'b1111;
      Paid = 4'b1111;
      Pentagons = 2'b01;
      Triangles = 2'b10;
      Circles = 2'b00;
  #10 $finish;

end

```

```systemverilog
endmodule: ChangeMachine_test


//Tests the MagComparator
module ModifiedMagComparator_test;

logic [3:0] optionA, optionB;
logic [3:0] chooses;

ModifiedMagComparator DUT(.A(optionA), .B(optionB), .options(chooses));

initial begin
$monitor($time, "A = %b B = %b chooses = %b", optionA, optionB, chooses);

optionA = 4'b0001;
optionB = 4'b1000;
#10 optionA = 4'b0100;
    optionB = 4'b0010;
#10 optionA = 4'b0000;
    optionB = 4'b0000;
#10 optionA = 4'b0101;
    optionB = 4'b0101;
#10 $finish;

end

endmodule: ModifiedMagComparator_test


//Tests the module that decides the path we take
module decidePath_test;

logic [2:0] comp;
logic checkZero;
logic [1:0] path;

DecidePath DUT(.*);

initial begin
$monitor($time, "combination = %b checkZero = %b path = %b", comp, checkZero,
        path);

comp = 3'b100;
checkZero = 1;
#10 comp = 3'b100;
    checkZero = 0;
#10 comp = 3'b010;
    checkZero = 1;
#10 comp = 3'b010;
    checkZero = 0;
#10 comp = 3'b001;
    checkZero = 1;
#10 comp = 3'b001;
    checkZero = 0;
#10 $finish;
end

endmodule: decidePath_test


//Tests the module that calculates the amount of change
module CalcChange_test;

logic [1:0] Pentagons, Triangles, Circles, path;
logic [3:0] Money, Price;
logic [3:0] change;
logic [5:0] availableCoins;

CalcChange DUT(.*);

initial begin
```

```
$monitor($time,{" Pentagons = %b Triangles = %b Circles = %b\n\t\t\t path = %b"
        ," Money = %b Price = %b\n\t\t\t change = %b availableCoins = %b"},
        Pentagons,Triangles, Circles, path, Money, Price, change,
        availableCoins);

Pentagons = 2'b10;
Triangles = 2'b10;
Circles = 2'b10;
path = 2'b10;
Money =  4'b1010;
Price = 4'b0101;
#10 Pentagons = 2'b10;
    Triangles = 2'b10;
    Circles = 2'b10;
    path = 2'b11;
    Money =  4'b1010;
    Price = 4'b0101;
#10 Pentagons = 2'b10;
    Triangles = 2'b10;
    Circles = 2'b10;
    path = 2'b01;
    Money =  4'b1010;
    Price = 4'b0101;
#10 Pentagons = 2'b10;
    Triangles = 2'b10;
    Circles = 2'b10;
    path = 2'b00;
    Money =  4'b1010;
    Price = 4'b0101;
#10 $finish;
end

endmodule: CalcChange_test


/*Tests the submodule that allows the calculations to be skipped depending on
the path*/
module Skip_test;

logic [1:0] path;
logic enable;

Skip s1(.*);

initial begin

  $monitor($time, " path = %b enable = %b", path, enable);

  path = 2'b00;
  #10 path = 2'b01;
  #10 path = 2'b10;
  #10 $finish;
  end

endmodule: Skip_test


//Tests the submodule that finds the change
module FindChange_test;


logic en;
logic [1:0] Pentagons, Triangles, Circles;
logic [3:0] Money, Price;
logic [3:0] change;
logic [5:0] availableCoins;

FindChange find(.*);

initial begin

```

```
$monitor($time, {" en = %b\n\t\t Pentagons = %b Triangles = %b Circles = %b\n",
        "\t\tMoney = %b Price = %b\n\t\t Change = %b AvailaibleCoins = %b"},
        en, Pentagons, Triangles, Circles, Money, Price, change,
        availableCoins);

en = 1;
Pentagons = 2'b10;
Triangles = 2'b10;
Circles = 2'b10;
Money = 4'b0100;
Price = 4'b0010;
#10 en = 0;
    Pentagons = 2'b10;
    Triangles = 2'b10;
    Circles = 2'b10;
    Money = 4'b0001;
    Price = 4'b0010;
#10 en = 1;
    Pentagons = 2'b10;
    Triangles = 2'b10;
    Circles = 2'b10;
    Money = 4'b1100;
    Price = 4'b0100;
#10 en = 1;
    Pentagons = 2'b01;
    Triangles = 2'b10;
    Circles = 2'b00;
    Money = 4'b1111;
    Price = 4'b0101;
#10 en = 1;
    Pentagons = 2'b01;
    Triangles = 2'b10;
    Circles = 2'b00;
    Money = 4'b0110;
    Price = 4'b0110;
#10 en = 0;
    Pentagons = 2'b01;
    Triangles = 2'b10;
    Circles = 2'b00;
    Money = 4'b0110;
    Price = 4'b0110;
#10 $finish;
end

endmodule: FindChange_test


//Tests the module that calculates the first coin
module CalcFirstCoin_test;
  logic [3:0] change;
  logic [5:0] availableCoins;
  logic [2:0] first;
  logic [3:0] restOfChange;
  logic [5:0] coinsRemain;

  CalcFirstCoin DUT(.*);

  initial begin

  $monitor($time, {" change = %b availableCoins = %b first = %b\n\t\t",
          " restOfChange = %b coinsRemain = %b"}, change, availableCoins, first,
          restOfChange, coinsRemain);

  change = 4'b0101;
  availableCoins = 6'b10_1010;
  #10 change = 4'b0101;
      availableCoins = 6'b00_1010;
  #10 change = 4'b0101;
      availableCoins = 6'b00_0010;
  #10 change = 4'b0101;
      availableCoins = 6'b00_0000;
  #10 $finish;
  end


endmodule: CalcFirstCoin_test


//Tests the submodule that finds the first coin
module FindFirst_test;
  logic [3:0] change;
  logic [5:0] availableCoins;
  logic [2:0] first;
  logic [3:0] restOfChange;

  FindFirst DUT(.*);

  initial begin

  $monitor($time, {" change = %b availableCoins = %b first = %b\n\t\t",
          " restOfChange = %b"}, change, availableCoins, first, restOfChange);

  change = 4'b0001;
  availableCoins = 6'b10_1010;
  #10 change = 4'b1100;
      availableCoins = 6'b00_1010;
  #10 change = 4'b0111;
      availableCoins = 6'b00_0010;
  #10 change = 4'b1011;
      availableCoins = 6'b00_0000;
  #10 change = 4'b1010;
      availableCoins = 6'b01_0001;
  #10 change = 4'b1001;
      availableCoins = 6'b00_0101;

  #10 $finish;
  end


endmodule: FindFirst_test


//Tests the submodule that removes the coin from the available ones
module RemoveCoin_test;

  logic [2:0] coinType;
  logic [5:0] availableCoins;
  logic [5:0] coinsRemain;

  RemoveCoin DUT(.*);


  initial begin

  $monitor($time, " Coin Type = %b Available Coins = %b Remaining = %b\n",
          coinType, availableCoins, coinsRemain);

  coinType = 3'b101;
  availableCoins = 6'b10_1010;
  #10 coinType = 3'b011;
      availableCoins = 6'b00_1010;
  #10 coinType = 3'b001;
      availableCoins = 6'b00_0010;
  #10 coinType = 3'b001;
      availableCoins = 6'b00_0001;
  #10 coinType = 3'b101;
      availableCoins = 6'b01_0001;
  #10 coinType = 3'b011;
      availableCoins = 6'b00_0101;

  #10 $finish;
  end
```

```
endmodule: RemoveCoin_test


//Tests the module that finds the second coin
module CalcSecondCoin_test;
  logic [3:0] leftoverChange;
  logic [5:0] coinsRemain;
  logic [2:0] second;
  logic [3:0] remaining;
  logic NotEnoughChange;

  CalcSecondCoin DUT(.*);

  initial begin

  $monitor($time, {" change = %b availableCoins = %b first = %b\n\t\t",
          " remaining = %b Enough = %b"}, leftoverChange, coinsRemain,
          second, remaining, NotEnoughChange);

  leftoverChange = 4'b0011;
  coinsRemain = 6'b01_1010;
  #10 leftoverChange = 4'b0000;
      coinsRemain = 6'b01_1010;
  #10 leftoverChange = 4'b0101;
      coinsRemain = 6'b00_0110;
  #10 leftoverChange = 4'b0001;
      coinsRemain = 6'b00_0110;
  #10 $finish;
  end


endmodule: CalcSecondCoin_test


//Figures out which display should be on and off on the FPGA board
module ApplyBlanks
  (input logic [3:0] remaining,
  output logic [7:0] blanks,
  output logic [3:0] newRemain,
  output logic [3:0] newRemain2);

  always_comb begin
    if (remaining[3] == 0 || remaining == 8 || remaining == 9) begin
      blanks = 8'b0010_1111;
    end
    else begin
      blanks = 8'b0000_1111;
    end

    if (remaining <= 4'd9) begin
      newRemain = remaining;
      newRemain2 = 0;
    end
    else begin
      newRemain = remaining - 4'b1010;
      newRemain2 = 1;
    end

  end

endmodule: ApplyBlanks


/*Adjusts the width of the first coin, second coin, and remaining so they
so they can fit in the input of the BCD they are assign to*/
module ChangeWidth
  (input logic [2:0] first_coin, second_coin,
  input logic [3:0] remaining,
  output logic [3:0] correctFirst, correctSecond);

  assign correctFirst[3] = 0;
  assign correctFirst[2:0] = first_coin;
  assign correctSecond[3] = 0;
  assign correctSecond[2:0] = second_coin;


endmodule: ChangeWidth


/*The top module that holds the overall coin machine and every module that does
the calculations for it inside*/
module ChipInterface
  (output logic [6:0] HEX7, HEX6, HEX5, HEX4,
  output logic [17:0] LEDR,
  input logic [17:0] SW);

  logic [2:0] first_coin, second_coin;
  logic [3:0] remaining;
  logic [7:0] blanks;
  logic [3:0] first_coin_out, second_coin_out, remaining_out1, remaining_out2;


  ChangeMachine machine(.Cost(SW[17:14]), .Pentagons(SW[13:12]),
                        .Triangles(SW[11:10]), .Circles(SW[9:8]),
                        .Paid(SW[3:0]), .ExactAmount(LEDR[17]),
                        .NotEnoughChange(LEDR[16]), .CoughUpMore(LEDR[15]),
                        .FirstCoin(first_coin), .SecondCoin(second_coin),
                        .Remaining(remaining));

  ChangeWidth width(.first_coin(first_coin), .second_coin(second_coin),
  .remaining(remaining), .correctFirst(first_coin_out),
  .correctSecond(second_coin_out));

  ApplyBlanks clear(.remaining(remaining), .blanks(blanks),
                    .newRemain(remaining_out1), .newRemain2(remaining_out2));


  SevenSegmentDisplay ssd(.BCD7(first_coin_out), .BCD6(second_coin_out),
  .BCD5(remaining_out2), .BCD4(remaining_out1), .blank(blanks), .HEX7(HEX7),
  .HEX6(HEX6), .HEX5(HEX5), .HEX4(HEX4));

endmodule : ChipInterface


//Helps to display the variables we defined onto the FPGA board BCDs
module SevenSegmentDisplay
(input logic [3:0] BCD7, BCD6, BCD5, BCD4, BCD3, BCD2, BCD1, BCD0,
input logic [7:0] blank,
output logic [6:0] HEX7, HEX6, HEX5, HEX4, HEX3, HEX2, HEX1, HEX0);

logic [6:0] preHEX7, preHEX6, preHEX5, preHEX4, preHEX3, preHEX2, preHEX1,
            preHEX0;
logic [6:0] nonInvertedHEX7,nonInvertedHEX6,nonInvertedHEX5,nonInvertedHEX4,
            nonInvertedHEX3,nonInvertedHEX2, nonInvertedHEX1, nonInvertedHEX0;

BCDtoSevenSegment d0(.bcd(BCD0), .segment(preHEX0));
BCDtoSevenSegment d1(.bcd(BCD1), .segment(preHEX1));
BCDtoSevenSegment d2(.bcd(BCD2), .segment(preHEX2));
BCDtoSevenSegment d3(.bcd(BCD3), .segment(preHEX3));
BCDtoSevenSegment d4(.bcd(BCD4), .segment(preHEX4));
BCDtoSevenSegment d5(.bcd(BCD5), .segment(preHEX5));
BCDtoSevenSegment d6(.bcd(BCD6), .segment(preHEX6));
BCDtoSevenSegment d7(.bcd(BCD7), .segment(preHEX7));

Mux2to1 m0(.I0(preHEX0), .I1(7'b0), .S(blank[0]), .Y(nonInvertedHEX0));
Mux2to1 m1(.I0(preHEX1), .I1(7'b0), .S(blank[1]), .Y(nonInvertedHEX1));
Mux2to1 m2(.I0(preHEX2), .I1(7'b0), .S(blank[2]), .Y(nonInvertedHEX2));
Mux2to1 m3(.I0(preHEX3), .I1(7'b0), .S(blank[3]), .Y(nonInvertedHEX3));
Mux2to1 m4(.I0(preHEX4), .I1(7'b0), .S(blank[4]), .Y(nonInvertedHEX4));
Mux2to1 m5(.I0(preHEX5), .I1(7'b0), .S(blank[5]), .Y(nonInvertedHEX5));
```

```
Mux2to1 m6(.I0(preHEX6), .I1(7'b0), .S(blank[6]), .Y(nonInvertedHEX6));
Mux2to1 m7(.I0(preHEX7), .I1(7'b0), .S(blank[7]), .Y(nonInvertedHEX7));

assign HEX0 = ~nonInvertedHEX0;
assign HEX1 = ~nonInvertedHEX1;
assign HEX2 = ~nonInvertedHEX2;
assign HEX3 = ~nonInvertedHEX3;
assign HEX4 = ~nonInvertedHEX4;
assign HEX5 = ~nonInvertedHEX5;
assign HEX6 = ~nonInvertedHEX6;
assign HEX7 = ~nonInvertedHEX7;

endmodule: SevenSegmentDisplay


//Converts the BCDs into the seven segments for the displays on the FPGA
module BCDtoSevenSegment
  (input logic [3:0] bcd,
  output logic [6:0] segment);

  always_comb begin
    unique case(bcd)
      4'b0000: segment = 7'b011_1111;
      4'b0001: segment = 7'b000_0110;
      4'b0010: segment = 7'b101_1011;
      4'b0011: segment = 7'b100_1111;
      4'b0100: segment = 7'b110_0110;
      4'b0101: segment = 7'b110_1101;
      4'b0110: segment = 7'b111_1101;
      4'b0111: segment = 7'b000_0111;
      4'b1000: segment = 7'b111_1111;
      4'b1001: segment = 7'b110_0111;
      default: segment = 7'b000_0000;
    endcase
  end

endmodule: BCDtoSevenSegment
```