```
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,
```

```
70    input logic [3:0] Money, Price,
71    output logic [3:0] change,
72    output logic [5:0] availableCoins);
73
74    assign change = (en) ? Money - Price : 0;
75    assign availableCoins[5:4] = Pentagons;
76    assign availableCoins[3:2] = Triangles;
77    assign availableCoins[1:0] = Circles;
78
79  endmodule: FindChange
80
81
82  //Calculates the first coin that can be produced from the change leftover
83  module CalcFirstCoin
84    (input logic [3:0] change,
85    input logic [5:0] availableCoins,
86    output logic [2:0] first,
87    output logic [3:0] restOfChange,
88    output logic [5:0] coinsRemain);
89
90
91    FindFirst coin(.change, .availableCoins, .first, .restOfChange);
92    RemoveCoin remove(.coinType(first), .availableCoins, .coinsRemain);
93
94
95  endmodule: CalcFirstCoin
96
97
98  //Submodule that finds the first coin based on the change
99  module FindFirst
100   (input logic [3:0] change,
101   input logic [5:0] availableCoins,
102   output logic [2:0] first,
103   output logic [3:0] restOfChange);
104
105   always_comb begin
106     if (availableCoins[5:4] > 2'b00 && change >= 3'b101)
107       first = 3'b101;
108     else if (availableCoins[3:2] > 2'b00 && change >= 3'b011)
109       first = 3'b011;
110     else if (availableCoins[1:0] > 2'b00 && change >= 3'b001)
111       first = 3'b001;
112     else
113       first = 3'b000;
114
115     restOfChange = change - first;
116   end
117
118 endmodule: FindFirst
119
120
121 /*Removes the coin taken when calculating the first coin from the group of
122 coins that are available in the coinbox*/
123 module RemoveCoin
124   (input logic [2:0] coinType,
125   input logic [5:0] availableCoins,
126   output logic [5:0] coinsRemain);
127
128   always_comb begin
129     unique case (coinType)
130       3'b000: coinsRemain = availableCoins;
131       3'b001: begin
132               coinsRemain[5:4] = availableCoins[5:4];
133               coinsRemain[3:2] = availableCoins[3:2];
134               coinsRemain[1:0] = availableCoins[1:0] - 2'b01;
135               end
136       3'b011: begin
137               coinsRemain[5:4] = availableCoins[5:4];
138               coinsRemain[3:2] = availableCoins[3:2] - 2'b01;
139               coinsRemain[1:0] = availableCoins[1:0];
140               end
```

```
141       3'b101: begin
142               coinsRemain[5:4] = availableCoins[5:4] - 2'b01;
143               coinsRemain[3:2] = availableCoins[3:2];
144               coinsRemain[1:0] = availableCoins[1:0];
145               end
146     endcase
147   end
148
149 endmodule: RemoveCoin
150
151
152 //Calculates the second coin that can be produced from the change leftover
153 module CalcSecondCoin
154   (input logic [3:0] leftoverChange,
155   input logic [5:0] coinsRemain,
156   output logic [2:0] second,
157   output logic [3:0] remaining,
158   output logic NotEnoughChange);
159
160
161   FindFirst coin2(.change(leftoverChange), .availableCoins(coinsRemain),
162                   .first(second), .restOfChange(remaining));
163
164   assign NotEnoughChange = (remaining > 0) ? 1 : 0;
165
166 endmodule: CalcSecondCoin
167
168
169 //Contains all of the modules for convience when using them in the top module
170 module ChangeMachine
171   (input logic [3:0] Cost, Paid,
172   input logic [1:0] Pentagons, Triangles, Circles,
173   output logic [2:0] FirstCoin, SecondCoin,
174   output logic ExactAmount, NotEnoughChange, CoughUpMore,
175   output logic [3:0] Remaining);
176
177   logic [3:0] options;
178   logic [1:0] wire_a;
179   logic [3:0] wire_b;
180   logic [5:0] wire_c;
181   logic [3:0] wire_d;
182   logic [5:0] wire_e;
183
184
185
186   ModifiedMagComparator compare(.A(Paid), .B(Cost), .options(options));
187   DecidePath findpath(.comp(options[3:1]), .checkZero(options[0]),
188                       .path(wire_a));
189   CalcChange calculate1(.Pentagons, .Triangles, .Circles, .path(wire_a),
190                         .Money(Paid), .Price(Cost), .change(wire_b),
191                         .availableCoins(wire_c));
192   CalcFirstCoin calculate2(.change(wire_b),
193                            .availableCoins(wire_c),
194                            .first(FirstCoin), .restOfChange(wire_d),
195                            .coinsRemain(wire_e));
196   CalcSecondCoin calculate3(.leftoverChange(wire_d),
197                             .coinsRemain(wire_e),
198                             .second(SecondCoin), .remaining(Remaining),
199                             .NotEnoughChange(NotEnoughChange));
200   assign ExactAmount = wire_a[1];
201   assign CoughUpMore = wire_a[0];
202
203 endmodule: ChangeMachine
204
205
206 //Test the top module that has all modules in it
207 module ChangeMachine_test;
208
209 logic [3:0] Cost, Paid;
210 logic [1:0] Pentagons, Triangles, Circles;
211 logic [2:0] FirstCoin, SecondCoin;
```

```
212 logic ExactAmount, NotEnoughChange, CoughUpMore;
213 logic [3:0] Remaining;
214
215 ChangeMachine DUT(.*);
216
217 initial begin
218
219   $monitor($time, {" Cost = %b Paid = %b\n\t\t Pentagons = %b Triangles = %b",
220           " Circles = %b\n\t\t First = %b Second = %b\n\t\t Exact = %b ",
221           "Enough = %b More = %b\n\t\t Remain = %b"},
222           Cost, Paid, Pentagons, Triangles, Circles, FirstCoin, SecondCoin,
223           ExactAmount, NotEnoughChange, CoughUpMore,
224           Remaining);
225
226   //Normal Cases
227   Cost = 4'b1111;
228   Paid = 4'b0001;
229   Pentagons = 2'b10;
230   Triangles = 2'b10;
231   Circles = 2'b10;
232   #10 Cost = 4'b1111;
233       Paid = 4'b1111;
234       Pentagons = 2'b00;
235       Triangles = 2'b10;
236       Circles = 3'b010;
237   #10 Cost = 4'b0001;
238       Paid = 4'b1111;
239       Pentagons = 2'b10;
240       Triangles = 2'b10;
241       Circles = 2'b10;
242   #10 Cost = 4'b0001;
243       Paid = 4'b1111;
244       Pentagons = 2'b01;
245       Triangles = 2'b01;
246       Circles = 2'b10;
247   #10 Cost = 4'b0001;
248       Paid = 4'b1111;
249       Pentagons = 2'b00;
250       Triangles = 2'b00;
251       Circles = 2'b00;
252   #10 Cost = 4'b1100;
253       Paid = 4'b0000;
254       Pentagons = 2'b10;
255       Triangles = 2'b10;
256       Circles = 2'b01;
257   #10 Cost = 4'b1001;
258       Paid = 4'b1111;
259       Pentagons = 2'b10;
260       Triangles = 2'b10;
261       Circles = 2'b01;
262
263   //Edge Cases
264   #10 Cost = 4'b0000;
265       Paid = 4'b0000;
266       Pentagons = 2'b00;
267       Triangles = 2'b00;
268       Circles = 2'b10;
269   #10 Cost = 4'b0000;
270       Paid = 4'b0011;
271       Pentagons = 2'b00;
272       Triangles = 2'b01;
273       Circles = 2'b10;
274   #10 Cost = 4'b1111;
275       Paid = 4'b1111;
276       Pentagons = 2'b01;
277       Triangles = 2'b10;
278       Circles = 2'b00;
279   #10 $finish;
280
281 end
282
```

```
283 endmodule: ChangeMachine_test
284
285
286 //Tests the MagComparator
287 module ModifiedMagComparator_test;
288
289 logic [3:0] optionA, optionB;
290 logic [3:0] chooses;
291
292 ModifiedMagComparator DUT(.A(optionA), .B(optionB), .options(chooses));
293
294 initial begin
295 $monitor($time, "A = %b B = %b chooses = %b", optionA, optionB, chooses);
296
297 optionA = 4'b0001;
298 optionB = 4'b1000;
299 #10 optionA = 4'b0100;
300     optionB = 4'b0010;
301 #10 optionA = 4'b0000;
302     optionB = 4'b0000;
303 #10 optionA = 4'b0101;
304     optionB = 4'b0101;
305 #10 $finish;
306
307 end
308
309 endmodule: ModifiedMagComparator_test
310
311
312 //Tests the module that decides the path we take
313 module decidePath_test;
314
315 logic [2:0] comp;
316 logic checkZero;
317 logic [1:0] path;
318
319 DecidePath DUT(.*);
320
321 initial begin
322 $monitor($time, "combination = %b checkZero = %b path = %b", comp, checkZero,
323         path);
324
325 comp = 3'b100;
326 checkZero = 1;
327 #10 comp = 3'b100;
328     checkZero = 0;
329 #10 comp = 3'b010;
330     checkZero = 1;
331 #10 comp = 3'b010;
332     checkZero = 0;
333 #10 comp = 3'b001;
334     checkZero = 1;
335 #10 comp = 3'b001;
336     checkZero = 0;
337 #10 $finish;
338 end
339
340 endmodule: decidePath_test
341
342
343 //Tests the module that calculates the amount of change
344 module CalcChange_test;
345
346 logic [1:0] Pentagons, Triangles, Circles, path;
347 logic [3:0] Money, Price;
348 logic [3:0] change;
349 logic [5:0] availableCoins;
350
351 CalcChange DUT(.*);
352
353 initial begin
```

```
354 $monitor($time,{" Pentagons = %b Triangles = %b Circles = %b\n\t\t\t path = %b"
355         ," Money = %b Price = %b\n\t\t\t change = %b availableCoins = %b"},
356         Pentagons,Triangles, Circles, path, Money, Price, change,
357         availableCoins);
358
359 Pentagons = 2'b10;
360 Triangles = 2'b10;
361 Circles = 2'b10;
362 path = 2'b10;
363 Money =  4'b1010;
364 Price = 4'b0101;
365 #10 Pentagons = 2'b10;
366     Triangles = 2'b10;
367     Circles = 2'b10;
368     path = 2'b11;
369     Money =  4'b1010;
370     Price = 4'b0101;
371 #10 Pentagons = 2'b10;
372     Triangles = 2'b10;
373     Circles = 2'b10;
374     path = 2'b01;
375     Money =  4'b1010;
376     Price = 4'b0101;
377 #10 Pentagons = 2'b10;
378     Triangles = 2'b10;
379     Circles = 2'b10;
380     path = 2'b00;
381     Money =  4'b1010;
382     Price = 4'b0101;
383 #10 $finish;
384 end
385
386 endmodule: CalcChange_test
387
388
389 /*Tests the submodule that allows the calculations to be skipped depending on
390 the path*/
391 module Skip_test;
392
393 logic [1:0] path;
394 logic enable;
395
396 Skip s1(.*);
397
398 initial begin
399
400   $monitor($time, " path = %b enable = %b", path, enable);
401
402   path = 2'b00;
403   #10 path = 2'b01;
404   #10 path = 2'b10;
405   #10 $finish;
406   end
407
408 endmodule: Skip_test
409
410
411 //Tests the submodule that finds the change
412 module FindChange_test;
413
414
415 logic en;
416 logic [1:0] Pentagons, Triangles, Circles;
417 logic [3:0] Money, Price;
418 logic [3:0] change;
419 logic [5:0] availableCoins;
420
421 FindChange find(.*);
422
423 initial begin
424
```

```
425 $monitor($time, {" en = %b\n\t\t Pentagons = %b Triangles = %b Circles = %b\n",
426         "\t\tMoney = %b Price = %b\n\t\t Change = %b AvailaibleCoins = %b"},
427         en, Pentagons, Triangles, Circles, Money, Price, change,
428         availableCoins);
429 
430 en = 1;
431 Pentagons = 2'b10;
432 Triangles = 2'b10;
433 Circles = 2'b10;
434 Money = 4'b0100;
435 Price = 4'b0010;
436 #10 en = 0;
437     Pentagons = 2'b10;
438     Triangles = 2'b10;
439     Circles = 2'b10;
440     Money = 4'b0001;
441     Price = 4'b0010;
442 #10 en = 1;
443     Pentagons = 2'b10;
444     Triangles = 2'b10;
445     Circles = 2'b10;
446     Money = 4'b1100;
447     Price = 4'b0100;
448 #10 en = 1;
449     Pentagons = 2'b01;
450     Triangles = 2'b10;
451     Circles = 2'b00;
452     Money = 4'b1111;
453     Price = 4'b0101;
454 #10 en = 1;
455     Pentagons = 2'b01;
456     Triangles = 2'b10;
457     Circles = 2'b00;
458     Money = 4'b0110;
459     Price = 4'b0110;
460 #10 en = 0;
461     Pentagons = 2'b01;
462     Triangles = 2'b10;
463     Circles = 2'b00;
464     Money = 4'b0110;
465     Price = 4'b0110;
466 #10 $finish;
467 end
468 
469 endmodule: FindChange_test
470 
471 
472 //Tests the module that calculates the first coin
473 module CalcFirstCoin_test;
474   logic [3:0] change;
475   logic [5:0] availableCoins;
476   logic [2:0] first;
477   logic [3:0] restOfChange;
478   logic [5:0] coinsRemain;
479 
480   CalcFirstCoin DUT(.*);
481 
482   initial begin
483 
484   $monitor($time, {" change = %b availableCoins = %b first = %b\n\t\t",
485           " restOfChange = %b coinsRemain = %b"}, change, availableCoins, first,
486           restOfChange, coinsRemain);
487 
488   change = 4'b0101;
489   availableCoins = 6'b10_1010;
490   #10 change = 4'b0101;
491       availableCoins = 6'b00_1010;
492   #10 change = 4'b0101;
493       availableCoins = 6'b00_0010;
494   #10 change = 4'b0101;
495       availableCoins = 6'b00_0000;
```

```
496   #10 $finish;
497   end
498
499
500 endmodule: CalcFirstCoin_test
501
502
503 //Tests the submodule that finds the first coin
504 module FindFirst_test;
505   logic [3:0] change;
506   logic [5:0] availableCoins;
507   logic [2:0] first;
508   logic [3:0] restOfChange;
509
510   FindFirst DUT(.*);
511
512   initial begin
513
514   $monitor($time, {" change = %b availableCoins = %b first = %b\n\t\t",
515           " restOfChange = %b"}, change, availableCoins, first, restOfChange);
516
517   change = 4'b0001;
518   availableCoins = 6'b10_1010;
519   #10 change = 4'b1100;
520       availableCoins = 6'b00_1010;
521   #10 change = 4'b0111;
522       availableCoins = 6'b00_0010;
523   #10 change = 4'b1011;
524       availableCoins = 6'b00_0000;
525   #10 change = 4'b1010;
526       availableCoins = 6'b01_0001;
527   #10 change = 4'b1001;
528       availableCoins = 6'b00_0101;
529
530   #10 $finish;
531   end
532
533
534 endmodule: FindFirst_test
535
536
537 //Tests the submodule that removes the coin from the available ones
538 module RemoveCoin_test;
539
540   logic [2:0] coinType;
541   logic [5:0] availableCoins;
542   logic [5:0] coinsRemain;
543
544   RemoveCoin DUT(.*);
545
546
547   initial begin
548
549   $monitor($time, " Coin Type = %b Available Coins = %b Remaining = %b\n",
550           coinType, availableCoins, coinsRemain);
551
552   coinType = 3'b101;
553   availableCoins = 6'b10_1010;
554   #10 coinType = 3'b011;
555       availableCoins = 6'b00_1010;
556   #10 coinType = 3'b001;
557       availableCoins = 6'b00_0010;
558   #10 coinType = 3'b001;
559       availableCoins = 6'b00_0001;
560   #10 coinType = 3'b101;
561       availableCoins = 6'b01_0001;
562   #10 coinType = 3'b011;
563       availableCoins = 6'b00_0101;
564
565   #10 $finish;
566   end
```

```
567
568
569 endmodule: RemoveCoin_test
570
571
572 //Tests the module that finds the second coin
573 module CalcSecondCoin_test;
574   logic [3:0] leftoverChange;
575   logic [5:0] coinsRemain;
576   logic [2:0] second;
577   logic [3:0] remaining;
578   logic NotEnoughChange;
579
580   CalcSecondCoin DUT(.*);
581
582   initial begin
583
584   $monitor($time, {" change = %b availableCoins = %b first = %b\n\t\t",
585           " remaining = %b Enough = %b"}, leftoverChange, coinsRemain,
586           second, remaining, NotEnoughChange);
587
588   leftoverChange = 4'b0011;
589   coinsRemain = 6'b01_1010;
590   #10 leftoverChange = 4'b0000;
591       coinsRemain = 6'b01_1010;
592   #10 leftoverChange = 4'b0101;
593       coinsRemain = 6'b00_0110;
594   #10 leftoverChange = 4'b0001;
595       coinsRemain = 6'b00_0110;
596   #10 $finish;
597   end
598
599
600 endmodule: CalcSecondCoin_test
601
602
603 //Figures out which display should be on and off on the FPGA board
604 module ApplyBlanks
605   (input logic [3:0] remaining,
606   output logic [7:0] blanks,
607   output logic [3:0] newRemain,
608   output logic [3:0] newRemain2);
609
610   always_comb begin
611     if (remaining[3] == 0 || remaining == 8 || remaining == 9) begin
612       blanks = 8'b0010_1111;
613     end
614     else begin
615       blanks = 8'b0000_1111;
616     end
617
618     if (remaining <= 4'd9) begin
619       newRemain = remaining;
620       newRemain2 = 0;
621     end
622     else begin
623       newRemain = remaining - 4'b1010;
624       newRemain2 = 1;
625     end
626
627   end
628
629 endmodule: ApplyBlanks
630
631
632 /*Adjusts the width of the first coin, second coin, and remaining so they
633 so they can fit in the input of the BCD they are assign to*/
634 module ChangeWidth
635   (input logic [2:0] first_coin, second_coin,
636   input logic [3:0] remaining,
637   output logic [3:0] correctFirst, correctSecond);
```

```
638
639   assign correctFirst[3] = 0;
640   assign correctFirst[2:0] = first_coin;
641   assign correctSecond[3] = 0;
642   assign correctSecond[2:0] = second_coin;
643
644
645 endmodule: ChangeWidth
646
647
648 /*The top module that holds the overall coin machine and every module that does
649 the calculations for it inside*/
650 module ChipInterface
651   (output logic [6:0] HEX7, HEX6, HEX5, HEX4,
652   output logic [17:0] LEDR,
653   input logic [17:0] SW);
654
655   logic [2:0] first_coin, second_coin;
656   logic [3:0] remaining;
657   logic [7:0] blanks;
658   logic [3:0] first_coin_out, second_coin_out, remaining_out1, remaining_out2;
659
660
661   ChangeMachine machine(.Cost(SW[17:14]), .Pentagons(SW[13:12]),
662                         .Triangles(SW[11:10]), .Circles(SW[9:8]),
663                         .Paid(SW[3:0]), .ExactAmount(LEDR[17]),
664                         .NotEnoughChange(LEDR[16]), .CoughUpMore(LEDR[15]),
665                         .FirstCoin(first_coin), .SecondCoin(second_coin),
666                         .Remaining(remaining));
667
668   ChangeWidth width(.first_coin(first_coin), .second_coin(second_coin),
669   .remaining(remaining), .correctFirst(first_coin_out),
670   .correctSecond(second_coin_out));
671
672   ApplyBlanks clear(.remaining(remaining), .blanks(blanks),
673                     .newRemain(remaining_out1), .newRemain2(remaining_out2));
674
675
676   SevenSegmentDisplay ssd(.BCD7(first_coin_out), .BCD6(second_coin_out),
677   .BCD5(remaining_out2), .BCD4(remaining_out1), .blank(blanks), .HEX7(HEX7),
678   .HEX6(HEX6), .HEX5(HEX5), .HEX4(HEX4));
679
680 endmodule : ChipInterface
681
682
683 //Helps to display the variables we defined onto the FPGA board BCDs
684 module SevenSegmentDisplay
685 (input logic [3:0] BCD7, BCD6, BCD5, BCD4, BCD3, BCD2, BCD1, BCD0,
686 input logic [7:0] blank,
687 output logic [6:0] HEX7, HEX6, HEX5, HEX4, HEX3, HEX2, HEX1, HEX0);
688
689 logic [6:0] preHEX7, preHEX6, preHEX5, preHEX4, preHEX3, preHEX2, preHEX1,
690             preHEX0;
691 logic [6:0] nonInvertedHEX7,nonInvertedHEX6,nonInvertedHEX5,nonInvertedHEX4,
692             nonInvertedHEX3,nonInvertedHEX2, nonInvertedHEX1, nonInvertedHEX0;
693
694 BCDtoSevenSegment d0(.bcd(BCD0), .segment(preHEX0));
695 BCDtoSevenSegment d1(.bcd(BCD1), .segment(preHEX1));
696 BCDtoSevenSegment d2(.bcd(BCD2), .segment(preHEX2));
697 BCDtoSevenSegment d3(.bcd(BCD3), .segment(preHEX3));
698 BCDtoSevenSegment d4(.bcd(BCD4), .segment(preHEX4));
699 BCDtoSevenSegment d5(.bcd(BCD5), .segment(preHEX5));
700 BCDtoSevenSegment d6(.bcd(BCD6), .segment(preHEX6));
701 BCDtoSevenSegment d7(.bcd(BCD7), .segment(preHEX7));
702
703 Mux2to1 m0(.I0(preHEX0), .I1(7'b0), .S(blank[0]), .Y(nonInvertedHEX0));
704 Mux2to1 m1(.I0(preHEX1), .I1(7'b0), .S(blank[1]), .Y(nonInvertedHEX1));
705 Mux2to1 m2(.I0(preHEX2), .I1(7'b0), .S(blank[2]), .Y(nonInvertedHEX2));
706 Mux2to1 m3(.I0(preHEX3), .I1(7'b0), .S(blank[3]), .Y(nonInvertedHEX3));
707 Mux2to1 m4(.I0(preHEX4), .I1(7'b0), .S(blank[4]), .Y(nonInvertedHEX4));
708 Mux2to1 m5(.I0(preHEX5), .I1(7'b0), .S(blank[5]), .Y(nonInvertedHEX5));
```

```
709 Mux2to1 m6(.I0(preHEX6), .I1(7'b0), .S(blank[6]), .Y(nonInvertedHEX6));
710 Mux2to1 m7(.I0(preHEX7), .I1(7'b0), .S(blank[7]), .Y(nonInvertedHEX7));
711
712 assign HEX0 = ~nonInvertedHEX0;
713 assign HEX1 = ~nonInvertedHEX1;
714 assign HEX2 = ~nonInvertedHEX2;
715 assign HEX3 = ~nonInvertedHEX3;
716 assign HEX4 = ~nonInvertedHEX4;
717 assign HEX5 = ~nonInvertedHEX5;
718 assign HEX6 = ~nonInvertedHEX6;
719 assign HEX7 = ~nonInvertedHEX7;
720
721 endmodule: SevenSegmentDisplay
722
723
724 //Converts the BCDs into the seven segments for the displays on the FPGA
725 module BCDtoSevenSegment
726   (input logic [3:0] bcd,
727   output logic [6:0] segment);
728
729   always_comb begin
730     unique case(bcd)
731       4'b0000: segment = 7'b011_1111;
732       4'b0001: segment = 7'b000_0110;
733       4'b0010: segment = 7'b101_1011;
734       4'b0011: segment = 7'b100_1111;
735       4'b0100: segment = 7'b110_0110;
736       4'b0101: segment = 7'b110_1101;
737       4'b0110: segment = 7'b111_1101;
738       4'b0111: segment = 7'b000_0111;
739       4'b1000: segment = 7'b111_1111;
740       4'b1001: segment = 7'b110_0111;
741       default: segment = 7'b000_0000;
742     endcase
743   end
744
745 endmodule: BCDtoSevenSegment
```