```
1   //Main Game Control
2
3   module maingamecontrol
4         (
5            CLOCK_50,                        // On Board 50 MHz
6            KEY,                             // On Board Keys
7            GPIO_0,
8            GPIO_1,
9            VGA_CLK,                         // VGA Clock
10           VGA_HS,                          // VGA H_SYNC
11           VGA_VS,                          // VGA V_SYNC
12           VGA_BLANK_N,                        // VGA BLANK
13           VGA_SYNC_N,                      // VGA SYNC
14           VGA_R,                           // VGA Red[9:0]
15           VGA_G,                           // VGA Green[9:0]
16           VGA_B                         // VGA Blue[9:0]
17        );
18
19        input            CLOCK_50;              // 50 MHz
20        input  [3:0]     KEY;
21        input  [5:0]     GPIO_0;
22        output [20:0]    GPIO_1;
23        output           VGA_CLK;               // VGA Clock
24        output           VGA_HS;                // VGA
    H_SYNC
25        output           VGA_VS;                // VGA
    V_SYNC
26        output           VGA_BLANK_N;           // VGA BLANK
27        output           VGA_SYNC_N;            // VGA SYNC
28        output [7:0]     VGA_R;                 // VGA
    Red[7:0] Changed from 10 to 8-bit DAC
29        output [7:0]     VGA_G;                 // VGA
    Green[7:0]
30        output [7:0]     VGA_B;                 // VGA
    Blue[7:0]
31
32
33
34        //declaring wires for external inputs
35        wire startbutton;
36        wire resetn;
37        wire closed;
38        wire opened;
39
40        //assigning external inputs
41        assign resetn = KEY[0];
42        assign startbutton = ~GPIO_0[0];
```

```
43          assign closed = ~GPIO_0[1];
44          assign opened = ~GPIO_0[2];
45
46          //declaring wires for external outputs
47          wire [3:0] motorpins;
48
49          //assigning external outputs
50          assign GPIO_1[3:0] = motorpins[3:0];
51
52          //declaring wires for FSM
53          wire close, open, rng, prestart, win, ready;
54          wire rngcountdone;
55          wire [3:0] current_state;
56          wire hand_out;
57
58          //declaring wires for countdown controls
59          wire countdone_quarter, countdone_half,
    countdone_one, countdone_seven, countdone_two,
    countdone_four;
60          wire enable_quarter, enable_half, enable_seven,
     enable_one, enable_two, enable_four;
61
62          //declaring wires for ramdon number look-up list
63          wire [3:0] rngnum;
64          wire [32:0] rngnumout;
65          wire [4:0] score1, score2, score3;
66
67          //declaring wires for motorcontrol
68          wire jaw_is_open, jaw_is_closed;
69          wire [3:0] current;
70          wire draw;
71
72          //declaring wires for VGA control
73          wire [7:0] x;
74          wire [6:0] y;
75          wire [23:0] colour;
76          wire [3:0] currentVGA;
77
78          //Declaring Video output module
79          vga_adapter VGA(.resetn(resetn),
80                          .clock(CLOCK_50),
81                          .colour(colour),
82                          .x(x),
83                          .y(y),
84                          .plot(1),
85                          /* Signals for the DAC to
    drive the monitor. */
```

```
 86                              .VGA_R(VGA_R),
 87                              .VGA_G(VGA_G),
 88                              .VGA_B(VGA_B),
 89                              .VGA_HS(VGA_HS),
 90                              .VGA_VS(VGA_VS),
 91                              .VGA_BLANK(VGA_BLANK_N),
 92                              .VGA_SYNC(VGA_SYNC_N),
 93                              .VGA_CLK(VGA_CLK));
 94          defparam VGA.RESOLUTION = "160x120";
 95          defparam VGA.MONOCHROME = "FALSE";
 96          defparam VGA.BITS_PER_COLOUR_CHANNEL  = 8;
 97          defparam VGA.BACKGROUND_IMAGE =
     "StartingBackground.mif" ;
 98
 99
100          //declaring motorFSM module
101          motorFSM m0(.clock(CLOCK_50),
102                      .resetn(resetn),
103                      .open_fsm(open),
104                      .close_fsm(close),
105                      .limit_jawOpen(opened),
106                      .limit_jawClose(closed),
107                      .jaw_is_open(jaw_is_open),
108                      .jaw_is_closed(jaw_is_closed),
109                      .pin(motorpins),
110                      .current(current)
111                     );
112
113          //declaring main game FSM module
114          controller c0(.clk(CLOCK_50),
115                        .resetn(resetn),
116                        .startbutton(startbutton),
117                        .rngcountdone(rngcountdone),
118                        .hand_out(hand_out),
119                        .closed(closed),
120                        .prestart(prestart),
121                        .close(close),
122                        .open(open),
123                        .rng(rng),
124                        .win(win),
125                        .ready(ready),
126                        .current_state(current_state),
127                        .countdone_quarter(
     countdone_quarter),
128                        .countdone_half(countdone_half),
129                        .countdone_one(countdone_one),
130                        .countdone_two(countdone_two),
```

```
131                         .countdone_four(countdone_four),
132                         .countdone_seven(countdone_seven),
133                         .enable_quarter(enable_quarter),
134                         .enable_half(enable_half),
135                         .enable_one(enable_one),
136                         .enable_two(enable_two),
137                         .enable_four(enable_four),
138                         .enable_seven(enable_seven),
139                         .opened(opened)
140                         );
141
142         //declaring main game control datapath
143         datapath d0(.clk(CLOCK_50),
144                     .prestart(prestart),
145                     .win(win),
146                     .closed(closed),
147                     .close(close),
148                     .check(check),
149                     .score1(score1),
150                     .score2(score2),
151                     .score3(score3),
152                     .hand_out(hand_out),
153                     .ready(ready)
154                     );
155
156
157         //declaring Video output FSM module
158         VGAcontrol VGA0(.clk(CLOCK_50),
159                     .resetn(resetn),
160                     .start(rng),
161                     .score1(score1),
162                     .score2(score2),
163                     .score3(score3),
164                     .x_out(x),
165                     .y_out(y),
166                     .colour_out(colour),
167                     .draw(draw),
168                     .currentmain(current_state)
169                     );
170
171         //declaring random number cycling module
172         rng r0(.clk(CLOCK_50),
173                .reset_n(resetn),
174                .rngnum(rngnum)
175               );
176
177         //declaring random number lookup table module
```

```
178         randomnumberlookup  r1(.rngnum(rngnum),
179                                .rngnumout(rngnumout)
180                                );
181
182         //declaring countdown from ramdom number module
183         rngcountdown  c1(.clk(CLOCK_50),
184                         .loadEnable(rng),
185                         .load(rngnumout),
186                         .countDone(rngcountdone)
187                         );
188
189         //declaring a variety of countdown modules
190         countdown_half  c2(.clk(CLOCK_50),
191                         .loadEnable(enable_half),
192                         .countDone(countdone_half)
193                         );
194
195         countdown_quarter  c3(.clk(CLOCK_50),
196                               .loadEnable(enable_quarter),
197                               .countDone(
    countdone_quarter)
198                               );
199
200         countdown_seven  c4(.clk(CLOCK_50),
201                             .loadEnable(enable_seven),
202                             .countDone(countdone_seven)
203                             );
204
205         countdown_four  c6(.clk(CLOCK_50),
206                             .loadEnable(enable_four),
207                             .countDone(countdone_four)
208                             );
209
210         countdown_two  c7(.clk(CLOCK_50),
211                             .loadEnable(enable_two),
212                             .countDone(countdone_two)
213                             );
214
215         countdown_one  c8(.clk(CLOCK_50),
216                             .loadEnable(enable_one),
217                             .countDone(countdone_one)
218                             );
219
220     endmodule
221
222
223     //This module is an FSM which acts as the brain of
```

```
    the game
224 //it determines when the game starts, when the jaw
    closes
225 //and if the player won or lost
226 module controller(
227                 input clk,
228                 input resetn,
229                 input startbutton,
230                 input rngcountdone, hand_out,
231                 input closed, opened,
232                 input countdone_quarter,
    countdone_half, countdone_one, countdone_two,
    countdone_four, countdone_seven,
233                 output reg prestart, close, open, rng,
     win, ready, gameover,
234                 output reg enable_quarter, enable_half
    , enable_one, enable_two, enable_four, enable_seven,
235                 output reg [3:0] current_state
236                 );
237
238    reg [3:0] next_state;
239
240    //declares states
241    localparam  PRESTART    = 4'd0,
242                WAIT        = 4'd1,
243                CLOSE       = 4'd2,
244                CHECK       = 4'd3,
245                WIN         = 4'd4,
246                OPEN        = 4'd5,
247                AFTERWIN    = 4'd6,
248                READY       = 4'd7,
249                LOSE        = 4'd8,
250                RESETGAME   = 4'd9,
251                AFTERLOSE   = 4'd10,
252                GAMEOVER    = 4'd11;
253
254     //this section dictates the order of states and
    the
255     //requirements for switching between states
256     always@(*)
257     begin: state_table
258     case(current_state)
259          PRESTART: begin
260             if(startbutton) next_state = WAIT;
    //checks if startbutton pressed if so goes to WAIT
    state
261          end
```

```
262             WAIT: begin
263                 if(rngcountdone)  next_state = CLOSE;
    //checks if the random countdown is done if so jaw
    closes
264             end
265             CLOSE: begin
266                 if(countdone_one) next_state = CHECK;
    //waits one second as the jaw closes then checks if
    the jaw has closed
267             end
268             CHECK: begin
269                 if(countdone_quarter) begin   //checks
    if jaws caught hand
270                     if(closed || hand_out > 0 ) next_state
     = WIN;       //closed represents the limit switch if
    the limit switch is closed the player wins
271                     else if(!closed) next_state = LOSE;
        //If the limit switch is open the player loses
272                 end
273             end
274             LOSE: begin
275                 next_state = RESETGAME;  //continues
    directly to resetgame
276             end
277             AFTERLOSE: begin
278                 if(countdone_two) next_state = GAMEOVER;
      //waits 2 seconds then contines to GAMEOVER this
    is to allow for the You Lose screen to show
279             end
280             GAMEOVER: begin
281                 if(countdone_four) next_state = PRESTART;
     //waits 4 second while showing the GAME OVER screen
    the goes back to the PRESTART state
282             end
283             RESETGAME: begin
284                 if(opened || countdone_four)  //waits
    for the jaw to hit the open limit switch or for 4
    seconds to protect the motor
285                 next_state = AFTERLOSE;  //moves to
    afterlose state
286             end
287             WIN: begin
288                 next_state = OPEN;  //continues directly
    to open state
289             end
290             OPEN: begin
291                 if(opened || countdone_one)   next_state
```

```
= AFTERWIN;  //Waits For Jaws to open or for 1 second
292             end
293             AFTERWIN: begin
294                 if(countdone_two) next_state = READY;
    //waits 2 seconds while Escaped screen shown
295             end
296             READY: begin
297                 if(startbutton) next_state = WAIT;
    //checks if startbutton pressed if so goes to WAIT
    state
298                 else if(countdone_seven) next_state =
    PRESTART;  //Waits 7 seconds then goes to PRESTART
299             end
300             default: next_state = PRESTART;
301         endcase
302         end
303
304         //This section dictates what will happen in each
    state
305         //as well as starting countdowns
306         always @(*)
307         begin: enable_signals
308
309           prestart        <= 0;
310           rng             <= 0;
311           close           <= 0;
312           open            <= 0;
313           win             <= 0;
314           ready           <= 0;
315           enable_half     <= 0;
316           enable_quarter  <= 0;
317           enable_one      <= 0;
318           enable_two      <= 0;
319           enable_four     <= 0;
320           enable_seven    <= 0;
321
322             case (current_state)
323                 PRESTART: begin   //the prestart state
    ensures that all game values are in their starting
    state
324                     prestart        <= 1;
325                     rng             <= 0;
326                     close           <= 0;
327                     open            <= 0;
328                     win             <= 0;
329                     ready           <= 0;
330                     enable_half     <= 0;
```

```
331                     enable_quarter  <= 0;
332                     enable_two      <= 0;
333                     enable_four     <= 0;
334                     enable_seven    <= 0;
335                 end
336                 WAIT: begin         //wait begins the
    ramdon countdown
337                     prestart        <= 0;
338                     rng             <= 1;
339                     close           <= 0;
340                     open            <= 0;
341                     win             <= 0;
342                     ready           <= 0;
343                     enable_half     <= 0;
344                     enable_quarter  <= 0;
345                     enable_two      <= 0;
346                     enable_four     <= 0;
347                     enable_seven    <= 0;
348                 end
349                 CLOSE: begin        //close closes the jaw
350                     prestart        <= 0;
351                     rng             <= 0;
352                     close           <= 1;
353                     open            <= 0;
354                     win             <= 0;
355                     ready           <= 0;
356                     enable_half     <= 0;
357                     enable_quarter  <= 0;
358                     enable_one      <= 1;
359                     enable_two      <= 0;
360                     enable_four     <= 0;
361                     enable_seven    <= 0;
362                 end
363                 OPEN: begin         //open opens the jaw
364                     prestart        <= 0;
365                     rng             <= 0;
366                     close           <= 0;
367                     open            <= 1;
368                     win             <= 0;
369                     ready           <= 0;
370                     enable_half     <= 0;
371                     enable_quarter  <= 0;
372                     enable_one      <= 1;
373                     enable_two      <= 0;
374                     enable_four     <= 0;
375                     enable_seven    <= 0;
376                 end
```

```
377                 CHECK: begin        //check starts a
    counter after which the FSM checks if the player won
    or lost
378                     prestart        <= 0;
379                     rng             <= 0;
380                     close           <= 0;
381                     open            <= 0;
382                     win             <= 0;
383                     ready           <= 0;
384                     enable_half     <= 0;
385                     enable_quarter  <= 1;
386                     enable_one      <= 0;
387                     enable_two      <= 0;
388                     enable_four     <= 0;
389                     enable_seven    <= 0;
390                 end
391                 WIN: begin          //sends the win signal
    to the datapath incrementing the score
392                     prestart        <= 0;
393                     rng             <= 0;
394                     close           <= 0;
395                     open            <= 0;
396                     win             <= 1;
397                     ready           <= 0;
398                     enable_half     <= 0;
399                     enable_quarter  <= 0;
400                     enable_two      <= 0;
401                     enable_four     <= 0;
402                     enable_seven    <= 0;
403                 end
404                 AFTERWIN: begin     //starts 2 second
    counter for video purposes
405                     prestart        <= 0;
406                     rng             <= 0;
407                     close           <= 0;
408                     open            <= 0;
409                     win             <= 0;
410                     ready           <= 0;
411                     enable_half     <= 0;
412                     enable_quarter  <= 0;
413                     enable_two      <= 1;
414                     enable_four     <= 0;
415                     enable_seven    <= 0;
416                 end
417                 LOSE: begin
418                     prestart        <= 0;
419                     rng             <= 0;
```

```
420                     close           <= 0;
421                     open            <= 0;
422                     win             <= 0;
423                     ready           <= 0;
424                     enable_half     <= 0;
425                     enable_quarter  <= 0;
426                     enable_two      <= 0;
427                     enable_four     <= 0;
428                     enable_seven    <= 0;
429                 end
430                 AFTERLOSE: begin  //starts the 2 second
    counter for video purposes
431                     prestart        <= 0;
432                     rng             <= 0;
433                     close           <= 0;
434                     open            <= 0;
435                     win             <= 0;
436                     ready           <= 0;
437                     enable_half     <= 0;
438                     enable_quarter  <= 0;
439                     enable_two      <= 1;
440                     enable_four     <= 0;
441                     enable_seven    <= 0;
442                 end
443                 GAMEOVER: begin   //starts the 4 second
    timer for video purposes
444                     prestart        <= 0;
445                     rng             <= 0;
446                     close           <= 0;
447                     open            <= 0;
448                     win             <= 0;
449                     ready           <= 0;
450                     enable_half     <= 0;
451                     enable_quarter  <= 0;
452                     enable_two      <= 0;
453                     enable_four     <= 1;
454                     enable_seven    <= 0;
455                 end
456                 RESETGAME: begin  //opens the jaw and
    satrts 4 second timer
457                     prestart        <= 0;
458                     rng             <= 0;
459                     close           <= 0;
460                     open            <= 1;
461                     win             <= 0;
462                     ready           <= 0;
463                     enable_half     <= 0;
```

```
464                     enable_quarter  <= 0;
465                     enable_two      <= 0;
466                     enable_four     <= 1;
467                     enable_seven    <= 0;
468                 end
469                 READY: begin      //sends ready signal
    to datapath and starts 7 second timer
470                     prestart        <= 0;
471                     rng             <= 0;
472                     close           <= 0;
473                     open            <= 0;
474                     win             <= 0;
475                     ready           <= 1;
476                     enable_half     <= 0;
477                     enable_quarter  <= 0;
478                     enable_two      <= 0;
479                     enable_four     <= 0;
480                     enable_seven    <= 1;
481                 end
482             endcase
483         end // enable_signals
484
485         //sets state transition to clock edge
486         always@(posedge clk)
487         begin: state_FFs
488             if(!resetn)
489                 current_state  = PRESTART;
490             else
491                 current_state  = next_state;
492         end // state_FFS
493     endmodule
494
495     //the datapath 2 concerned with 2 functions
496     //1.incrementing the score
497     //2.ensuring that if the jaw fully closes then opens
    slightly the player will still win
498     module datapath(  clk,
499                       prestart,
500                       win,
501                       closed,
502                       close,
503                       check,
504                       ready,
505                       score1,
506                       score2,
507                       score3,
508                       hand_out
```

```
509                         );
510
511         input clk;
512         input prestart;
513         input win;
514         input closed;
515         input close;
516         input check;
517         input ready;
518         output reg [4:0] score1, score2, score3;
519         output reg hand_out;
520
521         //score incrementer
522         always@(posedge clk)
523         begin
524             //sets score to 0 if the game is in prestart
        state
525             if(prestart) begin
526                 score1 <= 4'b0;
527                 score2 <= 4'b0;
528                 score3 <= 4'b0;
529             end
530             //if the win value is true the score is
        incremented by one
531             //to make graphics display easier the score
        value is broken
532             //into 3 parts and each is kept between 0-9
533             else if(win) begin
534                 if(score1 < 9) score1 <= score1 + 1;
535                 else if(score1 == 9) begin
536                     score1 <= 0;
537                     if(score2 < 9) score2 <= score2 + 1;
538                     else if(score2 == 9) begin
539                         score2 <= 0;
540                         score3 <= score3 + 1;
541                     end
542                 end
543             end
544         end
545
546         //ensures that if the jaw fully closes then opens
        slightly the player will still win
547         always@(*)
548         begin
549             //sets handout to 0 after check states is over
        effectively
550             if(prestart || ready)
```

```
551             hand_out <= 0;
552         //sets hand_out to 1 if jaw closes during
    close or check state
553         else if(close || check) begin
554             if(closed && hand_out == 0)
555                 hand_out <= 1;
556         end
557     end
558 endmodule
559
560
561 //selects a number based on a 4 bit input
562 //in this circuit the 4 bit value in being fed in by
    a constantly cycling counter
563 module randomnumberlookup( input [3:0]rngnum, output
    reg[32:0]rngnumout);
564
565     always@(*)
566     begin
567     case(rngnum[3:0])
568         4'd0:    rngnumout = 'd50000000;
569         4'd1:    rngnumout = 'd106250000;
570         4'd2:    rngnumout = 'd134375000;
571         4'd3:    rngnumout = 'd162500000;
572         4'd4:    rngnumout = 'd190625000;
573         4'd5:    rngnumout = 'd218750000;
574         4'd6:    rngnumout = 'd246875000;
575         4'd7:    rngnumout = 'd275000000;
576         4'd8:    rngnumout = 'd303125000;
577         4'd9:    rngnumout = 'd331250000;
578         4'd10:   rngnumout = 'd359375000;
579         4'd11:   rngnumout = 'd387500000;
580         4'd12:   rngnumout = 'd415625000;
581         4'd13:   rngnumout = 'd443750000;
582         4'd14:   rngnumout = 'd471875000;
583         4'd15:   rngnumout = 'd500000000;
584         default: rngnumout = 'd50000000;
585     endcase
586     end
587 endmodule
588
589 //cycles 4-bit value at clock edge
590 module rng(input clk, reset_n, output reg [3:0]rngnum
    );
591      always @ (posedge clk) begin
592          rngnum <= rngnum + 1;
593      end
```

```
594    endmodule
595
596    //counts down from loaded in random value
597    module rngcountdown(clk, load, loadEnable, countDone);
598       input clk, loadEnable;
599       input [32:0]load;
600       output reg countDone;
601
602       reg [32:0]countVal;
603
604       always @(posedge clk) begin
605          //if the loadEnable value is false random
       value is loaded in
606          //(a little confusing i know)
607          //and countDone set to zero
608          if (!loadEnable) begin
609             countVal <= load;
610             countDone <= 0;
611          end
612
613          //when the value is counted down the zero
       countDone is set to 1
614          else if (countVal == 'd0) begin
615             countDone <= 1;
616          end
617
618          //while the value is not zero it is
       incremented down each clock cycle
619          else if(countVal != 'd0) begin
620             countVal <= countVal - 1;
621             countDone <= 0;
622          end
623       end
624    endmodule
625
626    ////////////////////////////////////////////////////////
       ////////////////////////////////
627    ////////////Collection of counters below only first
       one will be commented////////////
628    ////////////////////////////////////////////////////////
       ////////////////////////////////
629
630    module countdown_one(clk, loadEnable, countDone);
631       input clk, loadEnable;
632       output reg countDone;
633
634       reg [32:0]countVal;
```

```
635
636         always @(posedge clk) begin
637         //while loadEnable = 0 countval is set to initail
        value and countDone is set to 0
638         if (!loadEnable) begin
639             countVal <= 'd50000000;
640             countDone <= 0;
641         end
642
643         //when countval is equal to 0 countDone is set to 1
644         else if (countVal == 'd0) begin
645             countDone <= 1;
646         end
647
648         //while countval is not zero it is incremented
        down each clock cycle
649         else if(countVal != 'd0) begin
650             countVal <= countVal - 1;
651             countDone <= 0;
652         end
653         end
654     endmodule
655
656
657     module countdown_half(clk, loadEnable, countDone);
658          input clk, loadEnable;
659          output reg countDone;
660
661          reg [32:0]countVal;
662
663          always @(posedge clk) begin
664              if (!loadEnable) begin
665             countVal <= 'd25000000;
666             countDone <= 0;
667         end
668
669         else if (countVal == 'd0) begin
670             countDone <= 1;
671         end
672
673         else if(countVal != 'd0) begin
674             countVal <= countVal - 1;
675             countDone <= 0;
676         end
677          end
678     endmodule
679
```

```
680   module countdown_quarter(clk, loadEnable, countDone);
681       input clk, loadEnable;
682       output reg countDone;
683
684       reg [32:0]countVal;
685
686      always @(posedge clk) begin
687           if (!loadEnable) begin
688         countVal <= 'd12500000;
689         countDone <= 0;
690      end
691
692           else if (countVal == 'd0) begin
693         countDone <= 1;
694      end
695
696      else if(countVal != 'd0) begin
697         countVal <= countVal - 1;
698         countDone <= 0;
699      end
700      end
701   endmodule
702
703   module countdown_two(clk, loadEnable, countDone);
704       input clk, loadEnable;
705       output reg countDone;
706
707       reg [32:0]countVal;
708
709      always @(posedge clk) begin
710           if (!loadEnable) begin
711         countVal <= 'd100000000;
712         countDone <= 0;
713      end
714
715      else if (countVal == 'd0) begin
716            countDone <= 1;
717      end
718
719      else if(countVal != 'd0) begin
720         countVal <= countVal - 1;
721         countDone <= 0;
722      end
723      end
724   endmodule
725
726
```

```
727   module countdown_four(clk, loadEnable, countDone);
728       input clk, loadEnable;
729       output reg countDone;
730   
731       reg [32:0]countVal;
732   
733      always @(posedge clk) begin
734           if (!loadEnable) begin
735         countVal <= 'd200000000;
736         countDone <= 0;
737      end
738   
739      else if (countVal == 'd0) begin
740            countDone <= 1;
741      end
742   
743      else if(countVal != 'd0) begin
744         countVal <= countVal - 1;
745         countDone <= 0;
746      end
747      end
748   endmodule
749   
750   
751   module countdown_seven(clk, loadEnable, countDone);
752       input clk, loadEnable;
753       output reg countDone;
754   
755       reg [32:0]countVal;
756   
757      always @(posedge clk) begin
758           if (!loadEnable) begin
759         countVal <= 'd750000000;
760         countDone <= 0;
761      end
762   
763      else if (countVal == 'd0) begin
764            countDone <= 1;
765      end
766   
767      else if(countVal != 'd0) begin
768         countVal <= countVal - 1;
769         countDone <= 0;
770      end
771      end
772   endmodule
773   
```