The Tank Game

ECE241 Final Report

Prepared by: Harvey Chen, Gligor Djogo

December 1, 2014

I. INTRODUCTION

The final project for Digital Systems (ECE241) was a comprehensive test of our knowledge and skills in designing hardware with Verilog. Since we were permitted to set our own goals for the three week timeline, we decided on simple statements:

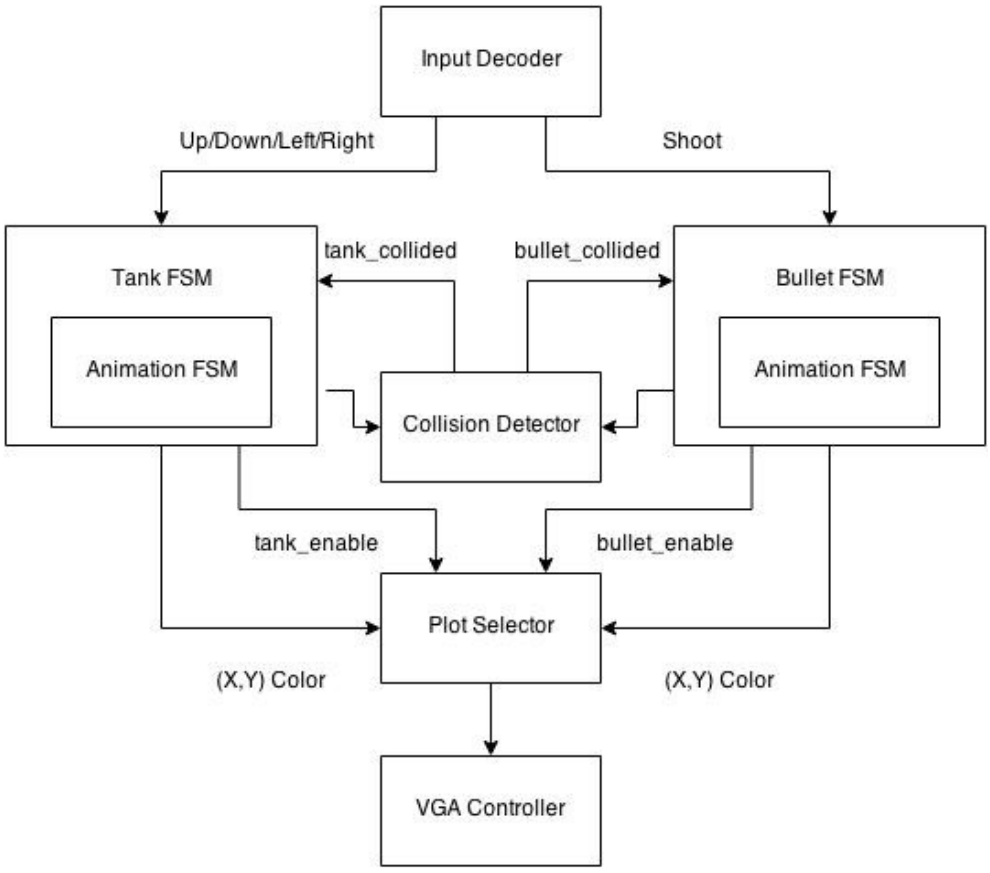
- Incorporate much of what was learned of hardware design from lectures and labs.
- Build on the hardware fundamentals by adding on external components, such as RAM, VGA controllers, or GPIO interface.
- Create a working multiplayer tank game, including shooting bullets, hitting other players, and free movement in a play area.

The underlying motivation was to create an interactive and enjoyable project for the user, something to grab their attention. The theme of a 'game' was inspired by previous years' examples. Also, equally important to us was to demonstrate the power of hardware in an unusual framework. That is to say, when someone thinks of computer games, there is more of an association to software than hardware, and building a game with circuits intrigued us conceptually.

II. THE DESIGN

Our core game design can be broken down into six major components, summarized below:

1. Tank FSM
2. Bullet FSM
3. Animation FSM
4. Collision detector
5. Plot selector
6. VGA controller



In essence, the user inputs commands through a controller, which the game interprets; directional movement shifts the tank a pixel unit in any direction, while the shoot button creates up to four independent bullets at a time, travelling in straight paths. Tanks are stopped from moving if they try to enter the surrounding boundary or other tanks, and bullets are 'destroyed' if they hit a wall or another tank. Users lose a life with every hit, as displayed in the margins of the playing field, and their tanks are reset to their initial position. Game is over when any player acquires three deaths.

Tank FSM

The central module of the entire project controls the order of outputs to the VGA, and when each submodule is sent its enable signal. The module containing the Tank FSM has the bulk of these controls. It takes in user input through GPIO pins, and outputs a set of values for the VGA controller, while controlling the animation of tanks, bullets and hearts. The module starts by erasing both tanks, one after the other in their current positions, by enabling the tank draw/erase module(Animation_Tank). Then, incrementing tank positions based on user inputs while making sure there are no collisions, and sequentially redrawing two tanks. Next, the eight bullets are managed, one at a time: they are either created (if the shoot button is pressed) or translated and refreshed in their new position, all by calling the bullet FSM. Bullet creation has specific logic which will only fire bullets spaced out by a certain delay, and it will always create them by filling the first available empty bullet one to four. This module also internally receives feedback when a tank is hit, and will reset the tank to its original coordinates, enable the heart erasing module, followed by enable signal to clear the playing field. If all hearts are lost, a call is made to print the winner on the monitor. Thus, the core of the game is handled by one main FSM, to best synchronize all the different animation modules, which all send their VGA values to a main internal multiplexer before being output to the monitor. This FSM is also synchronized with the monitor's refresh resolution frequency by implementing a delay counter state after drawing.

Bullet FSM

In the game, bullets are given a starting position and direction (depending on the orientation and position of the tank), after which they travel independently until they hit a wall or a tank. Thus, the above FSM will input an enable (make bullet) signal to read in coordinates and a direction, then create a bullet by drawing it via the animation (bullet draw/erase) submodule. At this point, the process is delayed as it waits for a second enable signal, this time for a refresh of the bullet's position. Once received, it erases the current bullet, increments, checks collisions, and redraws. The cycle continues autonomously outputting VGA values until the bullet is no longer active, ie. it collided.

Animation FSM

All such ‘animation’ or ‘draw/erase’ machines have one role, to output the coordinates and colour of a specified picture, pixel by pixel. They only differ by size of parameters, but all draw a box of some dimension. When enabled, they run in a simple loop: draw, increment to the next pixel, draw, etc. Exit conditions are specified to the desired size of box. When colour data needs to be read from a RAM, a secondary address counter runs in parallel and is fed to the RAM module, which will find the desired colour in the preloaded picture file. All these values are sent to the FSM above.

Collision FSM

There are four types of collision in the game. Collisions between two tanks, between a tank and a wall, between a bullet and a tank, and between a bullet and a wall. First, for collisions between tanks, both tanks’ positions are kept track of and these coordinates are compared when the user wants to move the tank. If the coordinates are too close to each other, such that the tanks would overlap, the tank is restricted from translating any further in that direction by skipping the increment command. Second, for collisions between tanks and walls, similarly compare the position of the tank with the boundaries. Third, to detect a collision between a bullet and a tank, the tank’s positions are input into the other player’s bullet FSM for comparison. If their positions overlap, a signal is sent from inside the bullet FSM to the tank FSM, where a hit counter is decremented (resulting in a lost heart). Lastly, collisions between bullets and walls is logically identical as collisions between tanks and walls. Instead of a 22 by 22 pixels object, a bullet is a 2 by 2 pixels object. Collisions checks are implemented within bigger modules, rather than a separate entity, for the sake of efficiency and compactness.

Plot Selector

The Plot Selector module takes all the x, y coordinates, color, plot and enable signals of all the tanks and bullets, and puts them through a 10 to 1 multiplexer. In order to plot each element of the game sequentially, different signals are turned on at different states to allow the correct coordinates and color to be sent out to the VGA controller. The multiplexer is an important component as it allows multiple entities to draw on the screen, in a controlled manner, since only one VGA module can be instantiated.

VGA Controller

The VGA controller module was taken from the seventh lab assignment. The module receives x, y coordinates, the colour to plot to the monitor, and a plot enable and converts it to analog signals for the monitor. It also preloads the screen with the game’s background.

III. THE SUCCESSES

Overall, we succeeded in reaching all milestones in time and ended up with the desired result: a two player tank game, with tank, bullet and wall collision detection, and a life counter to determine when a player has died. As a result, we were able to add one additional feature to the game: erasing a player's heart when they get hit by the opponent's bullet.

However, big problems were certainly encountered, troubling us for multiple days. The first of two main problems encountered is that the bullet's position would increment twice the amount expected, through only a single cycle of the FSM. After multiple attempts to fix the problem by debugging with QSim, LEDs, super slow clocking, or rewriting the code, the problem persisted. In the end, after much time lost, a work-around was found by subtracting the extra increment before the erase occurs. The behaviour is yet to be explained, but its suspected there may be underlying timing issues between the bullet FSM and the VGA controller.

The second problem would appear when moving tanks starting from a stationary position. When a key is pressed down, the keys around it would flicker once very quickly and cause the tank to jitter. We had no idea as to what could be wrong, this was a hardware problem of a slightly different kind. After experimenting with different keys and controllers, the decision was made to work around it as one might go about debouncing keys. By inserting a delay ('button buffer') at the moment the keys are pressed, the random jittering is ignored and only the value of the true key pressed is used in the subsequent code.

IV. FUTURE IMPROVEMENTS

Given a chance to redo this project, certain changes should likely be made in the code structure and separating different entities. This means:

- Converting tank FSM into individual entities, much more like the bullet FSM
- Separating the VGA adapter into a top level entity earlier, and using an FSM to control which submodules have permission to write to it (via multiplexers)

Such a structural modification would take advantage of an intrinsic feature of Verilog, to create standalone modules which can easily be reused. A stricter hierarchy of modules helps with readability, maintainability, and makes it easier to test then interconnect various pieces of a project. These benefits have been witnessed firsthand, especially with a 'MakeBullet' module instantiated numerous times, and probably should have been applied to the entire code.

Another helpful modification would be to simplify how values are stored in registers. Occasionally in the code, there exist multiple register declarations which hold very similar values that could be interchangeable with a minor modification to surrounding logic. For instance, when translating a tank, its x-y coordinates are incremented through a counter, but at the same time are being stored under more than one separate register names, which feed into different submodules. It is clear these can be harmonized into a single counter. Again, it was later done within the bullet module. The difference is, in the early stages of self-learning how to translate objects, the coding was inefficient, but the progress of individual coding experience is evident from tanks to bullets.

On a separate note, given more time to elaborate the existing game, the focus would be on adding features to the user interface. Portions of a start screen, countdown clock, round counter, and end-screen message are present in the final version of code, however they were never connected into the final product. Having said that, the core game is still entertaining and fully reaches the initial goals.

V. CONCLUSION

The project successfully demonstrated our knowledge by implementing a functioning two player interactive tank game. The biggest outcome is how it taught us the complexities of a large assignment that requires multiple FSMs and modules to interact between with one another to produce the desired outcome. It took a lot of teamwork and patience to get through some quirks of Verilog, in the end we are proud of what we have accomplished.

```
Tank FSM

module TankFSM
   (
      CLOCK_50,                  // On Board 50 MHz
      KEY,                       // Push Button[3:0]
      SW,                        // Switches [17:0]
      resetLife,
      resetn,
      LEDG,
      LEDR,
      GPIO_0,
      GPIO_1,
      x_animation,
      y_animation,
      color_animation,
      enable_animation,
      P1Life,
      P2Life
   );

// inputs
input CLOCK_50;            // 50 MHz
input resetn;
input [3:0] KEY;           // Button[3:0]
input [17:0] SW;           // Switches [17:0]
input [9:0]GPIO_0;         // USER CONTROL CONNECTIONS USING GPIO (For Player1)
input [9:0]GPIO_1;         // USER CONTROL CONNECTIONS USING GPIO (For Player2)

// testing purposes
output [7:0] LEDG;
output [17:0] LEDR;

// outputs
output [8:0]x_animation;
output [7:0]y_animation;
output [2:0]color_animation;
output enable_animation;

output P1Life;
output P2Life;

assign P1Life = (P1HitCount == 2'b00);
assign P2Life = (P2HitCount == 2'b00);


// wires to outputs assignemnts
assign x_animation = x_plot;
assign y_animation = y_plot;
assign color_animation = color_plot;
assign enable_animation = enable_plot;

// internal wires
wire Up, Down, Left, Right, Shoot1;          // control wires for player 1
wire Up2, Down2, Left2, Right2, Shoot2;      // control wires for player 2
wire Tank_plot, Bullet_plot, enable_plot;    // wires for different plot signals
wire [2:0]color_plot;                        // color fed into submodules
wire [8:0]x_plot;                            // x position fed into submodules
wire [7:0]y_plot;                            // y position fed into submodules

```

```
61   // internal wires assignemnts EXTERNAL CONTROLLERS' CONNECTIONS
62
63   assign Up = ~GPIO_0[0];
64   assign Down = ~GPIO_0[2];
65   assign Left = ~GPIO_0[4];
66   assign Right = ~GPIO_0[6];
67   assign Shoot1 = ~GPIO_0[8];
68   assign Up2 = ~GPIO_0[1];
69   assign Down2 = ~GPIO_0[3];
70   assign Left2 = ~GPIO_0[5];
71   assign Right2 = ~GPIO_0[7];
72   assign Shoot2 = ~GPIO_0[9];
73
74   wire [8:0]x_input;                    // initial x position for player 1
75   wire [7:0]y_input;                    // initial x position for player 2
76   assign x_input = 30;                  // assigns switches to x position
77   assign y_input = 30;                  // assigns switches to y position
78
79   reg rErase;                           // to enable Erase in Draw_and_Erase FSM
80   reg rDraw;                            // to enable Draw in Draw_and_Erase FSM
81   reg [8:0]x_count = 1;                 // keeps track of x counter player 1
82   reg [7:0]y_count = 1;                 // keeps track of y counter player 1
83   reg [8:0]x_count2 = 268;              // keeps track of x counter player 1
84   reg [7:0]y_count2 = 188;              // keeps track of y counter player 2
85   reg [8:0]tank1_pos_x;                 // tank 1 x position for collision detection
86   reg [7:0]tank1_pos_y;                 // tank 1 y position for collision detection
87   reg [8:0]tank2_pos_x;                 // tank 2 x position for collision detection
88   reg [7:0]tank2_pos_y;                 // tank 2 y position for collision detection
89   reg [8:0]tank1_x_abs;
90   reg [7:0]tank1_y_abs;
91   reg [8:0]tank2_x_abs;
92   reg [7:0]tank2_y_abs;
93   reg [8:0]x_translated;                // x position for the Draw_and_Erase FSM
94   reg [7:0]y_translated;                // y position for the Draw_and_Erase FSM
95   reg [26:0] delayCount;                // delay counter for player 1
96   reg [19:0] delayCount2;               // delay counter for state between the 2 Draw States
97   reg DoneDelay;                        // allows Draw_and_Erase FSM to send a Done signal
     for player1 DRAW/ERASE
98   reg DoneDelay2;                       // allows Draw_and_Erase FSM to send a Done signal
     for player2 DRAW/ERASE
99   wire DoneDrawOrErase;                 // the output signal of the FSM
100
101  wire Tank1Hit, Tank2Hit;
102  assign Tank1Hit = (P1B1_coll || P1B2_coll || P1B3_coll || P1B4_coll);
103  assign Tank2Hit = (P2B1_coll || P2B2_coll || P2B3_coll || P2B4_coll);
104
105  // BEGIN OF STATE DECLARATIONS
106     parameter [4:0] ST_Idle = 0, ST_ButtonBuffer = 20, ST_EraseCurrent = 1,
     ST_TurnOffErase = 2, ST_EraseCurrent2 = 3,
107                     ST_Up = 4, ST_Down = 5, ST_Left = 6, ST_Right = 7,
108                     ST_Draw_Translated = 8, ST_TurnOffDraw = 9, ST_Draw_Translated2 = 10,
109                     ST_Delay = 11,
110                     ST_Player1Bullet1 = 12, ST_Player1Bullet2 = 13, ST_Player1Bullet3 =
     14, ST_Player1Bullet4 = 15,
111                     ST_Player2Bullet1 = 16, ST_Player2Bullet2 = 17, ST_Player2Bullet3 =
     18, ST_Player2Bullet4 = 19,
112                     ST_EraseP1Heart = 21, ST_EraseP2Heart = 22, ST_ResetPositons = 23,
     ST_ClearField = 24, ST_TurnOffClear = 25, ST_DrawWinner = 26;
113  // END OF STATE DECLARATIONS
114
```

```
115   // BEGIN OF STATE FLIPFLOPS
116   reg [4:0]CState, NState;
117   always@(posedge CLOCK_50) begin
118      if (!resetn)
119         CState <= ST_Idle;
120      else if(P1B1_coll || P1B2_coll || P1B3_coll || P1B4_coll)
121         CState <= ST_EraseP2Heart;
122      else if(P2B1_coll || P2B2_coll || P2B3_coll || P2B4_coll)
123         CState <= ST_EraseP1Heart;
124      //else if(P1HitCount == 0 || P2HitCount == 0)
125         //CState <= ST_DrawWinner;
126      else
127         CState <= NState;
128   end
129   // END OF STATE FLIPFLOPS
130
131   assign LEDG[4:0] = CState;
132   assign LEDR[7] = StartClearField;
133
134         // BEGIN OF STATE TABLE
135         always@(*) begin
136            case(CState)
137                  ST_EraseP1Heart: begin
138                     if(DoneEraseHeart)
139                        NState <= ST_ResetPositons;
140                     else
141                        NState <= ST_EraseP1Heart;
142                  end
143                  ST_EraseP2Heart: begin
144                     if(DoneEraseHeart)
145                        NState <= ST_ResetPositons;
146                     else
147                        NState <= ST_EraseP2Heart;
148                  end
149                  ST_ResetPositons: begin
150                        NState <= ST_ClearField;
151                  end
152                  ST_ClearField: begin
153                        NState <= ST_TurnOffClear;
154                  end
155                  ST_TurnOffClear: begin
156                     if(DoneClearDelay)
157                        NState <= ST_Idle;
158                     else
159                        NState <= ST_TurnOffClear;
160                  end
161
162
163                  ST_Idle: begin
164                     if(Up || Down || Left || Right ||
165                        Up2 || Down2 || Left2 || Right2 ||
166                        Shoot1 || Shoot2 ||
167                        P1B1_Active || P1B2_Active || P1B3_Active || P1B4_Active ||
168                        P2B1_Active || P2B2_Active || P2B3_Active || P2B4_Active)
169                        NState <= ST_ButtonBuffer;
170                     else
171                        NState <= ST_Idle;
172                  end
173                  ST_ButtonBuffer: begin
174                     if(DoneButtonBuffer)
```

```
175                         NState <= ST_EraseCurrent;
176                      else
177                         NState <= ST_ButtonBuffer;
178                   end
179                   ST_EraseCurrent: begin
180                      if(DoneDrawOrErase)
181                         NState <= ST_TurnOffErase;
182                      else
183                         NState <= ST_EraseCurrent;
184                   end
185                   ST_TurnOffErase: begin // a state to turn off the rErase signal
186                         NState <= ST_EraseCurrent2;
187                   end
188                   ST_EraseCurrent2: begin
189                      if(DoneDrawOrErase)
190                         NState <= ST_Up;
191                      else
192                         NState <= ST_EraseCurrent2;
193                   end
194                   ST_Up:
195                         NState <= ST_Down;
196                   ST_Down:
197                         NState <= ST_Left;
198                   ST_Left:
199                         NState <= ST_Right;
200                   ST_Right:
201                         NState <= ST_Draw_Translated;
202                   ST_Draw_Translated: begin
203                      if(DoneDrawOrErase)
204                         NState <= ST_TurnOffDraw;
205                      else
206                         NState <= ST_Draw_Translated;
207                   end
208                   ST_TurnOffDraw: begin
209                      if(DoneDelay2)
210                         NState <= ST_Draw_Translated2;
211                      else
212                         NState <= ST_TurnOffDraw;
213                   end
214                   ST_Draw_Translated2: begin
215                      if(DoneDrawOrErase)
216                         NState <= ST_Delay;
217                      else
218                         NState <= ST_Draw_Translated2;
219                   end
220                   ST_Delay: begin
221                      if(DoneDelay)
222                         NState <= ST_Player1Bullet1;
223                      else
224                         NState <= ST_Delay;
225                   end
226                   ST_Player1Bullet1: begin
227                      if(Shoot1 && !P1B1_Active) begin
228                         if((!P1B2_Active || BulletBufferP1B2 > 45) && (!P1B3_Active ||
      BulletBufferP1B3 > 45) && (!P1B4_Active || BulletBufferP1B4 > 45)) begin
229                            if(P1B1_DoneDraw)
230                               NState <= ST_Player1Bullet2;
231                            else
232                               NState <= ST_Player1Bullet1;
233                         end
```

```
234                         else
235                            NState <= ST_Player1Bullet2;
236                   end
237                      else if(P1B1_Active) begin
238                         if(P1B1_DoneDraw)
239                            NState <= ST_Player1Bullet2;
240                         else
241                            NState <= ST_Player1Bullet1;
242                      end
243                      else
244                            NState <= ST_Player1Bullet2;
245               end
246               ST_Player1Bullet2: begin
247                   if(Shoot1 && P1B1_Active && !P1B2_Active) begin
248                      if((!P1B1_Active || BulletBufferP1B1 > 45) && (!P1B3_Active ||
      BulletBufferP1B3 > 45) && (!P1B4_Active || BulletBufferP1B4 > 45)) begin
249                         if(P1B2_DoneDraw)
250                            NState <= ST_Player1Bullet3;
251                         else
252                            NState <= ST_Player1Bullet2;
253                      end
254                         else
255                            NState <= ST_Player1Bullet3;
256                   end
257                      else if(P1B2_Active) begin
258                         if(P1B2_DoneDraw)
259                            NState <= ST_Player1Bullet3;
260                         else
261                            NState <= ST_Player1Bullet2;
262                      end
263                      else
264                            NState <= ST_Player1Bullet3;
265               end
266               ST_Player1Bullet3: begin
267                   if((Shoot1 && P1B1_Active && P1B2_Active && !P1B3_Active)) begin
268                      if((!P1B1_Active || BulletBufferP1B1 > 45) && (!P1B2_Active ||
      BulletBufferP1B2 > 45) && (!P1B4_Active || BulletBufferP1B4 > 45)) begin
269                         if(P1B3_DoneDraw)
270                            NState <= ST_Player1Bullet4;
271                         else
272                            NState <= ST_Player1Bullet3;
273                      end
274                         else
275                            NState <= ST_Player1Bullet4;
276                   end
277                      else if(P1B3_Active) begin
278                         if(P1B3_DoneDraw)
279                            NState <= ST_Player1Bullet4;
280                         else
281                            NState <= ST_Player1Bullet3;
282                      end
283                         else
284                            NState <= ST_Player1Bullet4;
285               end
286               ST_Player1Bullet4: begin
287                   if((Shoot1 && P1B1_Active && P1B2_Active && P1B3_Active && !
      P1B4_Active)) begin
288                      if((!P1B1_Active || BulletBufferP1B1 > 45) && (!P1B2_Active ||
      BulletBufferP1B2 > 45) && (!P1B3_Active || BulletBufferP1B3 > 45)) begin
289                         if(P1B4_DoneDraw)
```

```
290                             NState <= ST_Player2Bullet1;
291                          else
292                             NState <= ST_Player1Bullet4;
293                       end
294                          else
295                             NState <= ST_Player2Bullet1;
296                    end
297                       else if(P1B4_Active) begin
298                          if(P1B4_DoneDraw)
299                             NState <= ST_Player2Bullet1;
300                          else
301                             NState <= ST_Player1Bullet4;
302                       end
303                       else
304                             NState <= ST_Player2Bullet1;
305                end
306                ST_Player2Bullet1: begin
307                   if(Shoot2 && !P2B1_Active) begin
308                      if((!P2B2_Active || BulletBufferP2B2 > 45) && (!P2B3_Active ||
       BulletBufferP2B3 > 45) && (!P2B4_Active || BulletBufferP2B4 > 45)) begin
309                         if(P2B1_DoneDraw)
310                            NState <= ST_Player2Bullet2;
311                         else
312                            NState <= ST_Player2Bullet1;
313                      end
314                         else
315                            NState <= ST_Player2Bullet2;
316                   end
317                      else if(P2B1_Active) begin
318                         if(P2B1_DoneDraw)
319                            NState <= ST_Player2Bullet2;
320                         else
321                            NState <= ST_Player2Bullet1;
322                      end
323                      else
324                            NState <= ST_Player2Bullet2;
325                end
326                ST_Player2Bullet2: begin
327                   if(Shoot2 && P2B1_Active && !P2B2_Active) begin
328                      if((!P2B1_Active || BulletBufferP2B1 > 45) && (!P2B3_Active ||
       BulletBufferP2B3 > 45) && (!P2B4_Active || BulletBufferP2B4 > 45)) begin
329                         if(P2B2_DoneDraw)
330                            NState <= ST_Player2Bullet3;
331                         else
332                            NState <= ST_Player2Bullet2;
333                      end
334                         else
335                            NState <= ST_Player2Bullet3;
336                      end
337                      else if(P2B2_Active) begin
338                         if(P2B2_DoneDraw)
339                            NState <= ST_Player2Bullet3;
340                         else
341                            NState <= ST_Player2Bullet2;
342                      end
343                      else
344                            NState <= ST_Player2Bullet3;
345                end
346                ST_Player2Bullet3: begin
347                   if((Shoot2 && P2B1_Active && P2B2_Active && !P2B3_Active)) begin
```

```
348                        if((!P2B1_Active || BulletBufferP2B1 > 45) && (!P2B2_Active ||
    BulletBufferP2B2 > 45) && (!P2B4_Active || BulletBufferP2B4 > 45)) begin
349                           if(P2B3_DoneDraw)
350                              NState <= ST_Player2Bullet4;
351                           else
352                              NState <= ST_Player2Bullet3;
353                        end
354                           else
355                              NState <= ST_Player2Bullet4;
356                     end
357                        else if(P2B3_Active) begin
358                           if(P2B3_DoneDraw)
359                              NState <= ST_Player2Bullet4;
360                           else
361                              NState <= ST_Player2Bullet3;
362                        end
363                        else
364                              NState <= ST_Player2Bullet4;
365                  end
366                  ST_Player2Bullet4: begin
367                     if((Shoot2 && P2B1_Active && P2B2_Active && P2B3_Active && !
    P2B4_Active)) begin
368                        if((!P2B1_Active || BulletBufferP2B1 > 45) && (!P2B2_Active ||
    BulletBufferP2B2 > 45) && (!P2B3_Active || BulletBufferP2B3 > 45)) begin
369                           if(P2B4_DoneDraw)
370                              NState <= ST_Idle;
371                           else
372                              NState <= ST_Player2Bullet4;
373                        end
374                           else
375                              NState <= ST_Idle;
376                     end
377                        else if(P2B4_Active) begin
378                           if(P2B4_DoneDraw)
379                              NState <= ST_Idle;
380                           else
381                              NState <= ST_Player2Bullet4;
382                        end
383                        else
384                              NState <= ST_Idle;
385                  end
386
387                  default:
388                           NState <= ST_Idle;
389            endcase
390         end
391         // END OF STATE TABLE
392
393         // BEGIN OF STATE LOGIC
394         always@(posedge CLOCK_50)begin
395            if(CState == ST_Idle)begin
396                  DoneDelay = 0;
397                  delayCount = 0;
398                  delayCount2 = 0;
399                  rErase = 0;
400                  rDraw = 0;
401                  Tank1Enable = 0;
402                  Tank2Enable = 0;
403                  tank1_pos_x <= x_input + x_count; // temporarily stores tank 1 x
    position for comparison
```

```
404                 tank1_pos_y <= y_input + y_count; // temporarily stores tank 1 y
    position for comparison
405                 tank2_pos_x <= x_count2;            // temporarily stores tank 2 x
    position for comparison
406                 tank2_pos_y <= y_count2;            // temporarily stores tank 2 y
    position for comparison
407                 tank1_x_abs <= (tank1_pos_x > tank2_pos_x)?(tank1_pos_x - tank2_pos_x):(
    tank2_pos_x - tank1_pos_x); // absolute value of tank1 x distance
408                 tank1_y_abs <= (tank1_pos_y > tank2_pos_y)?(tank1_pos_y - tank2_pos_y):(
    tank2_pos_y - tank1_pos_y); // absolute value of tank1 y distance
409                 tank2_x_abs <= (tank1_pos_x > tank2_pos_x)?(tank1_pos_x - tank2_pos_x):(
    tank2_pos_x - tank1_pos_x); // absolute value of tank2 x distance
410                 tank2_y_abs <= (tank1_pos_y > tank2_pos_y)?(tank1_pos_y - tank2_pos_y):(
    tank2_pos_y - tank1_pos_y); // absolute value of tank2 y distance
411                 RefreshP2B4 = 0; //reset the last bullet's triggers
412                 MakeP2B4 = 0;
413                 ButtonBuffer = 0;
414                 EraseP1Heart = 0;
415                 EraseP2Heart = 0;
416                 ErasingHeart = 0;
417                 StartClearField = 0;
418                 ClearDelayCounter = 0;
419                 DrawingWin = 0;
420              end
421           if(CState == ST_ButtonBuffer)begin
422              ButtonBuffer = ButtonBuffer + 1;
423              DoneButtonBuffer = (ButtonBuffer == 600000);
424           end
425           if(CState == ST_EraseCurrent) begin // erase tank1
426                 DrawingTank = 1;
427                 x_translated = x_input + x_count; // x-input initialized to 30
428                 y_translated = y_input + y_count; // y-input initialized to 30
429                 rErase = 1;
430              end
431           if(CState == ST_TurnOffErase) begin
432                 rErase = 0;
433              end
434           if(CState == ST_EraseCurrent2) begin // erase tank2
435                 x_translated = x_count2;
436                 y_translated = y_count2;
437                 rErase = 1;
438              end
439           if(CState == ST_Up)begin
440                 rErase = 0;
441                 if(Up) begin
442                    TankDirection1 = 2'b00;
443                    if((tank1_x_abs > 21) || (tank1_pos_y - tank2_pos_y > 22)) begin
444                       if(y_count > 0)
445                          y_count = y_count - 1;
446                       else
447                          y_count = 0;
448                    end
449                 end
450                 if(Up2) begin
451                    TankDirection2 = 2'b00;
452                    if((tank2_x_abs > 21) || (tank2_pos_y - tank1_pos_y > 22)) begin
453                       if(y_count2 > 30)
454                          y_count2 = y_count2 - 1;
455                       else
456                          y_count2 = 30;
```

```
457                     end
458                 end
459             end
460             if(CState == ST_Down)begin
461                 if(Down) begin
462                     TankDirection1 = 2'b01;
463                     if((tank1_x_abs > 21) || (tank2_pos_y - tank1_pos_y > 22)) begin
464                         if(y_count < 158)
465                             y_count = y_count + 1;
466                         else
467                             y_count = 158;
468                             end
469                     end
470                 if(Down2) begin
471                     TankDirection2 = 2'b01;
472                     if((tank2_x_abs > 21) || (tank1_pos_y - tank2_pos_y > 22)) begin
473                         if(y_count2 < 188)
474                             y_count2 = y_count2 + 1;
475                         else
476                             y_count2 = 188;
477                     end
478                 end
479             end
480             if(CState == ST_Left)begin
481                 if(Left) begin
482                     TankDirection1 = 2'b10;
483                     if((tank1_y_abs > 21) || (tank1_pos_x - tank2_pos_x > 22)) begin
484                         if(x_count > 0)
485                             x_count = x_count - 1;
486                         else
487                             x_count = 0;
488                         end
489                 end
490                 if(Left2) begin
491                     TankDirection2 = 2'b10;
492                     if((tank2_y_abs > 21) || (tank2_pos_x - tank1_pos_x > 22)) begin
493                         if(x_count2 > 30)
494                             x_count2 = x_count2 - 1;
495                         else
496                             x_count2 = 30;
497                     end
498                 end
499             end
500             if(CState == ST_Right)begin
501                 if(Right) begin
502                     TankDirection1 = 2'b11;
503                     if((tank1_y_abs > 21) || (tank2_pos_x - tank1_pos_x > 22)) begin
504                         if(x_count < 238)
505                             x_count = x_count + 1;
506                         else
507                             x_count = 238;
508                     end
509                 end
510                 if(Right2) begin
511                     TankDirection2 = 2'b11;
512                     if((tank2_y_abs > 21) || (tank1_pos_x - tank2_pos_x > 22)) begin
513                         if(x_count2 < 268)
514                             x_count2 = x_count2 + 1;
515                         else
516                             x_count2 = 268;
```

```
517                     end
518                 end
519             end
520             if(CState == ST_Draw_Translated ) begin // draw tank1
521                     Tank1Enable = 1;
522                     x_translated = x_input + x_count;
523                     y_translated = y_input + y_count;
524                     rDraw = 1;
525                 end
526             if(CState == ST_TurnOffDraw ) begin // delay to wait for tank1 to finish drawingz
527                     rDraw = 0;
528                     delayCount2 = delayCount2 + 1;
529                     DoneDelay2 = (delayCount2 == 7000);
530                 end
531             if(CState == ST_Draw_Translated2 ) begin // draw tank2
532                     Tank2Enable = 1;
533                     x_translated = x_count2;
534                     y_translated = y_count2;
535                     rDraw = 1;
536                 end
537             if(CState == ST_Delay)begin
538                     rDraw = 0;
539                     DrawingTank = 0;
540                     delayCount = delayCount + 1;
541                     DoneDelay = (delayCount == 262000);
542                     tank1_pos_x <= x_input + x_count; // temporarily stores tank 1 x
    position for comparison
543                     tank1_pos_y <= y_input + y_count; // temporarily stores tank 1 y
    position for comparison
544                     tank2_pos_x <= x_count2;            // temporarily stores tank 2 x
    position for comparison
545                     tank2_pos_y <= y_count2;            // temporarily stores tank 2 y
    position for comparison
546                     Tank1Enable = 0;
547                     Tank2Enable = 0;
548                 end
549             if(CState == ST_Player1Bullet1 )begin
550                     BulletSelector = 3'b000;
551                     if(Shoot1 && !P1B1_Active ) begin // if P1B1 "should" be created
552                           if((!P1B2_Active || BulletBufferP1B2 > 45) && (!P1B3_Active ||
    BulletBufferP1B3 > 45) && (!P1B4_Active || BulletBufferP1B4 > 45)) begin // if any
    bullet's buffer is < 45
553                              MakeP1B1 = 1;
554                           end
555                     end
556                     else if(P1B1_Active ) begin
557                        RefreshP1B1 = 1;
558                     end
559                 end
560             if(CState == ST_Player1Bullet2 )begin
561                     BulletSelector = 3'b001;
562                     RefreshP1B1 = 0;
563                     MakeP1B1 = 0;
564                     if(Shoot1 && !P1B2_Active && P1B1_Active ) begin // if P1B2 "should"
    be created
565                           if((!P1B1_Active || BulletBufferP1B1 > 45) && (!P1B3_Active ||
    BulletBufferP1B3 > 45) && (!P1B4_Active || BulletBufferP1B4 > 45)) begin // if any
    bullet's buffer is < 45
566                              MakeP1B2 = 1;
567                           end
```

```
568                     end
569                     else if(P1B2_Active) begin
570                        RefreshP1B2 = 1;
571                     end
572                  end
573            if(CState == ST_Player1Bullet3 )begin
574                     BulletSelector = 3'b010;
575                     RefreshP1B2 = 0;
576                     MakeP1B2 = 0;
577                     if(Shoot1 && !P1B3_Active && P1B1_Active && P1B2_Active) begin // if
      P1B3 "should" be created
578                           if((!P1B1_Active || BulletBufferP1B1 > 45) && (!P1B2_Active ||
      BulletBufferP1B2 > 45) && (!P1B4_Active || BulletBufferP1B4 > 45)) begin // if any
      bullet's buffer is < 45
579                              MakeP1B3 = 1;
580                           end
581                     end
582                     else if(P1B3_Active) begin
583                        RefreshP1B3 = 1;
584                     end
585                  end
586            if(CState == ST_Player1Bullet4 )begin
587                     BulletSelector = 3'b011;
588                     RefreshP1B3 = 0;
589                     MakeP1B3 = 0;
590                     if(Shoot1 && !P1B4_Active && P1B1_Active && P1B2_Active && P1B3_Active
      ) begin // if P1B4 "should" be created
591                           if((!P1B1_Active || BulletBufferP1B1 > 45) && (!P1B2_Active ||
      BulletBufferP1B2 > 45) && (!P1B3_Active || BulletBufferP1B3 > 45)) begin // if any
      bullet's buffer is < 45
592                              MakeP1B4 = 1;
593                           end
594                     end
595                     else if(P1B4_Active) begin
596                        RefreshP1B4 = 1;
597                     end
598                  end
599            if(CState == ST_Player2Bullet1 )begin
600                     BulletSelector = 3'b100;
601                     RefreshP1B4 = 0;
602                     MakeP1B4 = 0;
603                     if(Shoot2 && !P2B1_Active) begin // if P2B1 "should" be created
604                           if((!P2B2_Active || BulletBufferP2B2 > 45) && (!P2B3_Active ||
      BulletBufferP2B3 > 45) && (!P2B4_Active || BulletBufferP2B4 > 45)) begin // if any
      bullet's buffer is < 45
605                              MakeP2B1 = 1;
606                           end
607                     end
608                     else if(P2B1_Active) begin
609                        RefreshP2B1 = 1;
610                     end
611                  end
612            if(CState == ST_Player2Bullet2 )begin
613                     BulletSelector = 3'b101;
614                     RefreshP2B1 = 0;
615                     MakeP2B1 = 0;
616                     if(Shoot2 && !P2B2_Active && P2B1_Active) begin // if P2B2 "should"
      be created
617                           if((!P2B1_Active || BulletBufferP2B1 > 45) && (!P2B3_Active ||
      BulletBufferP2B3 > 45) && (!P2B4_Active || BulletBufferP2B4 > 45)) begin // if any
```

```
      bullet's buffer is < 45
618                              MakeP2B2 = 1;
619                           end
620                     end
621                     else if(P2B2_Active) begin
622                        RefreshP2B2 = 1;
623                     end
624                  end
625            if(CState == ST_Player2Bullet3 )begin
626                     BulletSelector = 3'b110;
627                     RefreshP2B2 = 0;
628                     MakeP2B2 = 0;
629                     if(Shoot2 && !P2B3_Active && P2B1_Active && P2B2_Active) begin // if
      P2B3 "should" be created
630                           if((!P2B1_Active || BulletBufferP2B1 > 45) && (!P2B2_Active ||
      BulletBufferP2B2 > 45) && (!P2B4_Active || BulletBufferP2B4 > 45)) begin // if any
      bullet's buffer is < 45
631                              MakeP2B3 = 1;
632                           end
633                     end
634                     else if(P2B3_Active) begin
635                        RefreshP2B3 = 1;
636                     end
637                  end
638            if(CState == ST_Player2Bullet4 )begin
639                     BulletSelector = 3'b111;
640                     RefreshP2B3 = 0;
641                     MakeP2B3 = 0;
642                     if(Shoot2 && !P2B4_Active && P2B1_Active && P2B2_Active && P2B3_Active
      ) begin // if P2B4 "should" be created
643                           if((!P2B1_Active || BulletBufferP2B1 > 45) && (!P2B2_Active ||
      BulletBufferP2B2 > 45) && (!P2B3_Active || BulletBufferP2B3 > 45)) begin // if any
      bullet's buffer is < 45
644                              MakeP2B4 = 1;
645                           end
646                     end
647                     else if(P2B4_Active) begin
648                        RefreshP2B4 = 1;
649                     end
650                  end
651            if(CState == ST_EraseP1Heart )begin
652               EraseP1Heart = 1;
653               ErasingHeart = 1;
654            end
655            if(CState == ST_EraseP2Heart )begin
656               EraseP2Heart = 1;
657               ErasingHeart = 1;
658            end
659            if(CState == ST_ResetPositons )begin
660                  x_count = 1;
661                  y_count = 1;
662                  x_count2 = 268;
663                  y_count2 = 188;
664            end
665            if(CState == ST_ClearField )begin
666                  StartClearField = 1;
667            end
668            if(CState == ST_TurnOffClear )begin
669               ClearDelayCounter = ClearDelayCounter + 1;
670               DoneClearDelay = (ClearDelayCounter == 140000);
```

```
671             end
672             if(CState == ST_DrawWinner)begin
673                DrawingWin = 1;
674                if (P1HitCount == 0)
675                   Player1Win = 1;
676                else if (P2HitCount == 0)
677                   Player2Win = 1;
678             end
679       end
680       // END OF STATE LOGIC
681
682        DrawPlayerWin winner(
683       CLOCK_50,
684       resetn,
685       Player1Win,
686       Player2Win,
687       color_DrawPlayerWin,
688       x_DrawPlayerWin,
689       y_DrawPlayerWin,
690       plot_DrawPlayerWin
691       );
692
693       reg Player1Win;
694       reg Player2Win;
695       reg DrawingWin;
696
697          reg DoneClearDelay;
698          reg [17:0]ClearDelayCounter;
699
700          wire [8:0]x_DrawPlayerWin;
701          wire [7:0]y_DrawPlayerWin;
702          wire [2:0]color_DrawPlayerWin;
703          wire plot_DrawPlayerWin;
704
705          // INPUTS INTO MULTIPLEXER
706          // signals from tanks
707          wire [8:0]Tank_x;
708          wire [7:0]Tank_y;
709          wire [2:0]Tank_color;
710          reg [1:0]TankDirection1, TankDirection2;
711          reg Tank1Enable, Tank2Enable;
712
713          // selection signals
714          reg DrawingTank;
715          reg [2:0] BulletSelector; //increase mux selector bit size for more bullet capacity
716          reg [22:0] ButtonBuffer;
717          reg DoneButtonBuffer;
718
719          // signals from 8 different bullets
     [x,y,color,plot,Make,Refresh,Active,DoneDraw,BulletBuffer]
720          wire [8:0] P1B1_x_pos, P1B2_x_pos, P1B3_x_pos, P1B4_x_pos, P2B1_x_pos, P2B2_x_pos,
      P2B3_x_pos, P2B4_x_pos;
721          wire [7:0] P1B1_y_pos, P1B2_y_pos, P1B3_y_pos, P1B4_y_pos, P2B1_y_pos, P2B2_y_pos,
      P2B3_y_pos, P2B4_y_pos;
722          wire [2:0] P1B1_color, P1B2_color, P1B3_color, P1B4_color, P2B1_color, P2B2_color,
      P2B3_color, P2B4_color;
723          wire P1B1_plot, P1B2_plot, P1B3_plot, P1B4_plot, P2B1_plot, P2B2_plot, P2B3_plot,
     P2B4_plot;
724          reg MakeP1B1, MakeP1B2, MakeP1B3, MakeP1B4, MakeP2B1, MakeP2B2, MakeP2B3, MakeP2B4;
725          reg RefreshP1B1, RefreshP1B2, RefreshP1B3, RefreshP1B4, RefreshP2B1, RefreshP2B2,
```

```
      RefreshP2B3, RefreshP2B4;
726         wire P1B1_Active, P1B2_Active, P1B3_Active, P1B4_Active, P2B1_Active, P2B2_Active,
       P2B3_Active, P2B4_Active;
727         wire P1B1_DoneDraw, P1B2_DoneDraw, P1B3_DoneDraw, P1B4_DoneDraw, P2B1_DoneDraw,
      P2B2_DoneDraw, P2B3_DoneDraw, P2B4_DoneDraw;
728         wire [8:0]BulletBufferP1B1, BulletBufferP1B2, BulletBufferP1B3, BulletBufferP1B4,
      BulletBufferP2B1, BulletBufferP2B2, BulletBufferP2B3, BulletBufferP2B4;
729         // END OF INPUTS
730
731         plotSelector plot(
732                               P1B1_x_pos, P1B1_y_pos, P1B1_color, P1B1_plot,
733                               P1B2_x_pos, P1B2_y_pos, P1B2_color, P1B2_plot,
734                               P1B3_x_pos, P1B3_y_pos, P1B3_color, P1B3_plot,
735                               P1B4_x_pos, P1B4_y_pos, P1B4_color, P1B4_plot,
736                               P2B1_x_pos, P2B1_y_pos, P2B1_color, P2B1_plot,
737                               P2B2_x_pos, P2B2_y_pos, P2B2_color, P2B2_plot,
738                               P2B3_x_pos, P2B3_y_pos, P2B3_color, P2B3_plot,
739                               P2B4_x_pos, P2B4_y_pos, P2B4_color, P2B4_plot,
740                               Tank_x, Tank_y, Tank_color, Tank_plot,
741                               BulletSelector, DrawingTank,
742                               ErasingHeart, x_heart, y_heart, color_heart, plot_heart,
743                               StartClearField, x_clear, y_clear, color_clear, plot_clear,
744                               DrawingWin, x_DrawPlayerWin, y_DrawPlayerWin,
      color_DrawPlayerWin, plot_DrawPlayerWin,
745                               x_plot, y_plot, color_plot, enable_plot
746                               );
747
748         BulletMaker P1B1(
749                               CLOCK_50, resetn,
750                               MakeP1B1, RefreshP1B1,
751                               tank1_pos_x, tank1_pos_y, TankDirection1,
752                               P1B1_x_pos, P1B1_y_pos, P1B1_color, P1B1_plot,
753                               P1B1_Active, P1B1_DoneDraw, BulletBufferP1B1,
754                               P1B1_coll, tank2_pos_x, tank2_pos_y);
755
756         BulletMaker P1B2(
757                               CLOCK_50, resetn,
758                               MakeP1B2, RefreshP1B2,
759                               tank1_pos_x, tank1_pos_y, TankDirection1,
760                               P1B2_x_pos, P1B2_y_pos, P1B2_color, P1B2_plot,
761                               P1B2_Active, P1B2_DoneDraw, BulletBufferP1B2,
762                               P1B2_coll, tank2_pos_x, tank2_pos_y);
763
764         BulletMaker P1B3(
765                               CLOCK_50, resetn,
766                               MakeP1B3, RefreshP1B3,
767                               tank1_pos_x, tank1_pos_y, TankDirection1,
768                               P1B3_x_pos, P1B3_y_pos, P1B3_color, P1B3_plot,
769                               P1B3_Active, P1B3_DoneDraw, BulletBufferP1B3,
770                               P1B3_coll, tank2_pos_x, tank2_pos_y);
771
772         BulletMaker P1B4(
773                               CLOCK_50, resetn,
774                               MakeP1B4, RefreshP1B4,
775                               tank1_pos_x, tank1_pos_y, TankDirection1,
776                               P1B4_x_pos, P1B4_y_pos, P1B4_color, P1B4_plot,
777                               P1B4_Active, P1B4_DoneDraw, BulletBufferP1B4,
778                               P1B4_coll, tank2_pos_x, tank2_pos_y);
779
780         BulletMaker P2B1(
```

```
781                         CLOCK_50, resetn,
782                         MakeP2B1, RefreshP2B1,
783                         tank2_pos_x, tank2_pos_y, TankDirection2,
784                         P2B1_x_pos, P2B1_y_pos, P2B1_color, P2B1_plot,
785                         P2B1_Active, P2B1_DoneDraw, BulletBufferP2B1,
786                         P2B1_coll, tank1_pos_x, tank1_pos_y);
787
788         BulletMaker P2B2(
789                         CLOCK_50, resetn,
790                         MakeP2B2, RefreshP2B2,
791                         tank2_pos_x, tank2_pos_y, TankDirection2,
792                         P2B2_x_pos, P2B2_y_pos, P2B2_color, P2B2_plot,
793                         P2B2_Active, P2B2_DoneDraw, BulletBufferP2B2,
794                         P2B2_coll, tank1_pos_x, tank1_pos_y);
795
796         BulletMaker P2B3(
797                         CLOCK_50, resetn,
798                         MakeP2B3, RefreshP2B3,
799                         tank2_pos_x, tank2_pos_y, TankDirection2,
800                         P2B3_x_pos, P2B3_y_pos, P2B3_color, P2B3_plot,
801                         P2B3_Active, P2B3_DoneDraw, BulletBufferP2B3,
802                         P2B3_coll, tank1_pos_x, tank1_pos_y);
803
804         BulletMaker P2B4(
805                         CLOCK_50, resetn,
806                         MakeP2B4, RefreshP2B4,
807                         tank2_pos_x, tank2_pos_y, TankDirection2,
808                         P2B4_x_pos, P2B4_y_pos, P2B4_color, P2B4_plot,
809                         P2B4_Active, P2B4_DoneDraw, BulletBufferP2B4,
810                         P2B4_coll, tank1_pos_x, tank1_pos_y);
811
812
813         draw_and_erase_tank  tank_animator(
814                                         CLOCK_50, resetn, rDraw, rErase,
815                                         x_translated, y_translated,
816                                         Tank_color, Tank_x, Tank_y, Tank_plot,
817                                         DoneDrawOrErase,
818                                         TankDirection1, TankDirection2,
819                                         Tank1Enable, Tank2Enable);
820
821               // debug
822         assign LEDR[17] = Up;
823         assign LEDR[16] = Down;
824         assign LEDR[15] = Left;
825         assign LEDR[14] = Right;
826         assign LEDR[13] = Shoot1;
827         assign LEDR[12] = Up2;
828         assign LEDR[11] = Down2;
829         assign LEDR[10] = Left2;
830         assign LEDR[9] = Right2;
831         assign LEDR[8] = Shoot2;
832
833         reg ErasingHeart;
834         reg EraseP1Heart, EraseP2Heart;
835         wire DoneEraseHeart;
836         // HITS COUNTER
837         wire P1B1_coll, P1B2_coll, P1B3_coll, P1B4_coll, P2B1_coll, P2B2_coll, P2B3_coll,
      P2B4_coll;
838         input resetLife;// = SW[10]; // control with super-top-level fsm
839         wire [1:0]P1HitCount, P2HitCount;
```

```
840         assign LEDG[7:6] = P1HitCount; // change later to erasing half a heart off the
    background every time a tank gets hit
841        // assign LEDR[10:9] = P2HitCount; // redraw the hearts from ram when the game is
    reset in super-top-level fsm
842
843         hitCount deathmatch(CLOCK_50, resetLife, P1B1_coll, P1B2_coll, P1B3_coll,
    P1B4_coll, P2B1_coll, P2B2_coll, P2B3_coll, P2B4_coll, P1HitCount, P2HitCount);
844         // END OF HITS COUNTER
845
846         ClearField clear(CLOCK_50, resetn, StartClearField, color_clear, x_clear, y_clear,
     plot_clear, DoneClearField);
847         reg StartClearField;
848         wire [8:0]x_clear;
849         wire [7:0]y_clear;
850         wire [2:0]color_clear;
851         wire plot_clear;
852         wire DoneClearField;
853
854         wire [8:0]x_heart;
855         wire [7:0]y_heart;
856         wire [2:0]color_heart;
857         wire plot_heart;
858         EraseHearts heartEraser(CLOCK_50, resetn, EraseP1Heart, EraseP2Heart, P1HitCount,
    P2HitCount, color_heart, x_heart, y_heart, plot_heart, DoneEraseHeart);
859
860   endmodule
861
862   module hitCount (iclock, iresetLife, iP1B1_coll, iP1B2_coll, iP1B3_coll, iP1B4_coll,
    iP2B1_coll, iP2B2_coll, iP2B3_coll, iP2B4_coll, oP1HitCount, oP2HitCount);
863         input iclock, iresetLife; // active high reset @!
864         input iP1B1_coll, iP1B2_coll, iP1B3_coll, iP1B4_coll, iP2B1_coll, iP2B2_coll,
    iP2B3_coll, iP2B4_coll;
865         output reg [2:0]oP1HitCount, oP2HitCount; //increase size for greater life capacity
866
867         // DEATH COUNTER
868         always@(posedge iclock)begin
869            if (iresetLife) begin
870               oP1HitCount = 3;
871               oP2HitCount = 3;
872            end
873            else if (iP1B1_coll || iP1B2_coll || iP1B3_coll || iP1B4_coll) begin
874               oP2HitCount = oP2HitCount - 1;
875            end
876            else if (iP2B1_coll || iP2B2_coll || iP2B3_coll || iP2B4_coll) begin
877               oP1HitCount = oP1HitCount - 1;
878            end
879         end
880         //End of death counter
881   endmodule
882
```

```
1   Animation FSM (Tank)
2
3   module Animation_Tank
4      (
5      iCLOCK_50,
6      iresetn,
7      idrawEn,
8      ieraseEn,
9      ix_pos,
10     iy_pos,
11     ocolor_out,
12     ox,
13     oy,
14     owriteEn,
15     oDoneSignal,
16     iTankDirection1,
17     iTankDirection2,
18     iTank1Enable,
19     iTank2Enable
20     );
21
22     input iCLOCK_50, iresetn, idrawEn, ieraseEn;
23     input [1:0]iTankDirection1; // 00 = tank1 up, 01 = tank1 down, 10 = tank1 left, 11 =
   tank1 right
24     input [1:0]iTankDirection2; // 00 = tank2 up, 01 = tank2 down, 10 = tank2 left, 11 =
   tank2 right
25     input iTank1Enable, iTank2Enable;
26     input [8:0]ix_pos;
27     input [7:0]iy_pos;
28     output [2:0]ocolor_out;
29     output reg [8:0]ox;
30     output reg [7:0]oy;
31     output reg owriteEn;
32     output reg oDoneSignal;
33
34     reg [4:0]x_counter, y_counter; //up to 32
35     reg draw;
36
37     always@(*) begin
38        if(iTank1Enable)begin
39           if(iTankDirection1 == 2'b00) // tank1 up
40              mem_color = Tank_1U_Color;
41           else if(iTankDirection1 == 2'b01)
42              mem_color = Tank_1D_Color;
43           else if(iTankDirection1 == 2'b10)
44              mem_color = Tank_1L_Color;
45           else if(iTankDirection1 == 2'b11)
46              mem_color = Tank_1R_Color;
47        end
48        if(iTank2Enable)begin
49           if(iTankDirection2 == 2'b00) // tank2 up
50              mem_color = Tank_2U_Color;
51           else if(iTankDirection2 == 2'b01)
52              mem_color = Tank_2D_Color;
53           else if(iTankDirection2 == 2'b10)
54              mem_color = Tank_2L_Color;
55           else if(iTankDirection2 == 2'b11)
56              mem_color = Tank_2R_Color;
57        end
58  end
```

```
59
60      assign ocolor_out = draw ? mem_color : 3'b111;
61
62      // RAM initialization controllers
63      reg [8:0]ramAddress; //0 to 512
64      reg [2:0]idata;
65      reg ramWen;
66      reg [2:0] mem_color;
67      wire [2:0] Tank_1U_Color;
68      wire [2:0] Tank_1D_Color;
69      wire [2:0] Tank_1L_Color;
70      wire [2:0] Tank_1R_Color;
71      wire [2:0] Tank_2U_Color;
72      wire [2:0] Tank_2D_Color;
73      wire [2:0] Tank_2L_Color;
74      wire [2:0] Tank_2R_Color;
75
76      // INITIALIZIE RAM MODULES FOR TANK 1
77      TankOneUp     Tank_1U(ramAddress, iCLOCK_50, idata, ramWen, Tank_1U_Color);
78      TankOneDown   Tank_1D(ramAddress, iCLOCK_50, idata, ramWen, Tank_1D_Color);
79      TankOneLeft   Tank_1L(ramAddress, iCLOCK_50, idata, ramWen, Tank_1L_Color);
80      TankOneRight  Tank_1R(ramAddress, iCLOCK_50, idata, ramWen, Tank_1R_Color);
81
82      // INITIALIZIE RAM MODULES FOR TANK 2
83      TankTwoUp     Tank_2U(ramAddress, iCLOCK_50, idata, ramWen, Tank_2U_Color);
84      TankTwoDown   Tank_2D(ramAddress, iCLOCK_50, idata, ramWen, Tank_2D_Color);
85      TankTwoLeft   Tank_2L(ramAddress, iCLOCK_50, idata, ramWen, Tank_2L_Color);
86      TankTwoRight  Tank_2R(ramAddress, iCLOCK_50, idata, ramWen, Tank_2R_Color);
87
88      //STATES and State FFs
89      parameter [3:0] ST_idle = 0, ST_chooseColor = 4, ST_setMem = 1, ST_draw = 2, ST_count
    = 3, ST_Done = 5;
90      reg [3:0] CState, NState;
91      always@(posedge iCLOCK_50) begin
92         if (!iresetn)
93            CState = ST_idle;
94         else
95            CState = NState;
96      end
97      //end fsm setup
98
99      //CHANGING STATE
100     always@(*) begin
101        NState = ST_idle; //blocking or nonblock?
102        case(CState)
103              ST_idle: begin
104                 if (idrawEn)
105                    NState = ST_chooseColor;
106                 else if (ieraseEn)
107                    NState = ST_chooseColor;
108                 else
109                    NState = ST_idle;
110              end
111
112              ST_chooseColor:
113                    NState = ST_setMem;
114
115              ST_setMem:
116                    NState = ST_draw;
117
```

```
118                ST_draw: begin
119                   if (y_counter == 21 && x_counter == 21) // stop when y_counter reaches 16
120                      NState = ST_Done;
121                   else
122                      NState = ST_count;
123                end
124                ST_count:
125                      NState = ST_draw;
126                ST_Done:
127                      NState = ST_idle;
128
129                default:
130                      NState = ST_idle;
131          endcase
132       end
133       // end state rules
134
135       //VARIOUS COUNTERS
136       always@(posedge iCLOCK_50) begin
137          if (CState == ST_idle) begin
138             owriteEn = 0;
139             oDoneSignal = 0;
140             ramAddress = 0;
141          end
142          if (CState == ST_setMem) begin
143             ramAddress = 0;
144             x_counter = 0;
145             y_counter = 0;
146             end
147          if (CState == ST_chooseColor )begin
148             if(idrawEn)
149                draw = 1;
150             else if(ieraseEn)
151                draw = 0;
152             end
153          if (CState == ST_count) begin
154             owriteEn = 0;
155             ramAddress = ramAddress + 1;
156             if (x_counter < 21) // increments x_counter until x_counter is > 15
157                x_counter = x_counter + 1;
158             else begin
159                x_counter = 0; // resets x_counter
160                y_counter = y_counter + 1; // increments y_counter
161                end
162             end
163
164          if (CState == ST_draw)
165             owriteEn = 1;
166
167          if (CState == ST_Done)
168             oDoneSignal = 1;
169
170          ox = ix_pos + x_counter;
171          oy = iy_pos + y_counter;
172       end
173       // end counters
174    endmodule
175
```

```
1   Bullet FSM
2
3   //MAKE and MOVE single bullets to edge of boundary
4   module BulletFSM(iCLOCK_50, iresetn, iMakeBullet, iBulletRefresh, iX_starting_pos,
    iY_starting_pos, iDirection, oX, oY, oColor,
5                       owriteEn, oactive, oDoneBulletCycle, oBulletBuffer, oCollision,
    iotherTank_x, iotherTank_y);
6
7      input iCLOCK_50, iresetn, iMakeBullet, iBulletRefresh;
8      input [8:0]iX_starting_pos;
9      input [7:0]iY_starting_pos;
10     input [1:0]iDirection;
11     output [8:0]oX;
12     output [7:0]oY;
13     output [2:0]oColor;
14     output owriteEn;
15     output reg oactive, oDoneBulletCycle;
16     output reg [8:0] oBulletBuffer;
17     output reg oCollision;
18     input [8:0]iotherTank_x;
19     input [7:0]iotherTank_y;
20
21     // bullet data
22     reg [8:0] translated_bullet_x;
23     reg [7:0] translated_bullet_y;
24     reg [1:0] bullet_direction;
25     // drawing controls
26     reg rErase, rDraw;
27     wire DoneDrawOrErase;
28
29     //states in order
30     parameter [3:0] ST_Idle = 0, ST_SetValues = 1,
31                     ST_DrawBullet = 2, ST_TurnOffDraw = 3,
32                     ST_SendDoneSignal = 4,
33                     ST_Delay = 5,
34                     ST_deIncrementBullet = 6,
35                     ST_EraseBullet = 7, ST_TurnOffErase = 8,
36                     ST_CollisionDetect = 9,
37                     ST_IncrementBullet = 10;
38
39
40     // begin of state flip flops
41     reg [3:0] CState, NState;
42     always@(posedge iCLOCK_50)begin
43        if(!iresetn)
44           CState <= ST_Idle;
45        else
46           CState <= NState;
47     end
48     // end of state flip flops
49
50     // begin of state table
51     always@(*) begin
52        case(CState)
53           ST_Idle: begin
54              if(iMakeBullet)
55                 NState <= ST_SetValues;
56              else
57                 NState <= ST_Idle;
58           end
```

```
59              ST_SetValues: begin
60                    NState <= ST_DrawBullet;
61              end
62              ST_DrawBullet: begin
63                 if(DoneDrawOrErase)
64                    NState <= ST_TurnOffDraw;
65                 else
66                    NState <= ST_DrawBullet;
67              end
68              ST_TurnOffDraw: begin
69                 if (donebuffering)
70                    NState <= ST_SendDoneSignal;
71                 else
72                    NState <= ST_TurnOffDraw;
73              end
74              ST_SendDoneSignal: begin
75                    NState <= ST_Delay;
76              end
77              ST_Delay: begin
78                 if(iBulletRefresh)
79                    NState <= ST_deIncrementBullet;
80                 else
81                    NState <= ST_Delay;
82              end
83              ST_deIncrementBullet: begin
84                    NState <= ST_EraseBullet;
85              end
86              ST_EraseBullet: begin
87                 if(DoneDrawOrErase)
88                    NState <= ST_TurnOffErase;
89                 else
90                    NState <= ST_EraseBullet;
91              end
92              ST_TurnOffErase: begin
93                    NState <= ST_CollisionDetect;
94              end
95              ST_CollisionDetect: begin
96                 if (Bcollided_w_tank || Bcollided_w_wall)
97                    NState <= ST_Idle;
98                 else
99                    NState <= ST_IncrementBullet;
100             end
101             ST_IncrementBullet: begin
102                   NState <= ST_DrawBullet;
103             end
104
105             default:
106                   NState <= ST_Idle;
107          endcase
108       end
109       // eoftable
110
111       // begin of state logic
112       always@(posedge iCLOCK_50)begin
113          if(CState == ST_Idle)begin
114             oactive = 0;
115             oDoneBulletCycle = 0;
116             oBulletBuffer = 0;
117             oCollision = 0;
118             bullet_direction = iDirection;
```

```
119         end
120         else if(CState == ST_SetValues)begin
121            oactive = 1;
122            if(bullet_direction == 2'b00)begin // UP
123               translated_bullet_x = iX_starting_pos + 10;
124               translated_bullet_y = iY_starting_pos - 1;
125            end
126            else if(bullet_direction == 2'b01)begin // DOWN
127               translated_bullet_x = iX_starting_pos + 10;
128               translated_bullet_y = iY_starting_pos + 22;
129            end
130            else if(bullet_direction == 2'b10)begin // LEFT
131               translated_bullet_x = iX_starting_pos - 2;
132               translated_bullet_y = iY_starting_pos + 10;
133            end
134            else if(bullet_direction == 2'b11)begin // RIGHT
135               translated_bullet_x = iX_starting_pos + 22;
136               translated_bullet_y = iY_starting_pos + 10;
137            end
138         end
139         else if (CState == ST_deIncrementBullet) begin
140            if(bullet_direction == 2'b00) // UP
141               translated_bullet_y = translated_bullet_y + 1;
142            else if(bullet_direction == 2'b01) // DOWN
143               translated_bullet_y = translated_bullet_y - 1;
144            else if(bullet_direction == 2'b10) // LEFT
145               translated_bullet_x = translated_bullet_x + 1;
146            else if(bullet_direction == 2'b11) // RIGHT
147               translated_bullet_x = translated_bullet_x - 1;
148            end
149         else if(CState == ST_EraseBullet)begin
150            rErase = 1;
151         end
152         else if(CState == ST_TurnOffErase)begin
153            rErase = 0;
154         end
155         else if(CState == ST_IncrementBullet)begin
156            if(bullet_direction == 2'b00) // UP
157               translated_bullet_y = translated_bullet_y - 2;
158            else if(bullet_direction == 2'b01) // DOWN
159               translated_bullet_y = translated_bullet_y + 2;
160            else if(bullet_direction == 2'b10) // LEFT
161               translated_bullet_x = translated_bullet_x - 2;
162            else if(bullet_direction == 2'b11) // RIGHT
163               translated_bullet_x = translated_bullet_x + 2;
164         end
165         else if(CState == ST_DrawBullet)begin
166            rDraw = 1;
167            drawbuffer = 0;
168            donebuffering = 0;
169         end
170         else if(CState == ST_TurnOffDraw)begin
171            rDraw = 0;
172            drawbuffer = drawbuffer + 1;
173            donebuffering = (drawbuffer == 32);
174         end
175         else if(CState == ST_SendDoneSignal)begin
176            oBulletBuffer = oBulletBuffer + 1;
177            oDoneBulletCycle = 1;
178         end
```

```
179         else if(CState == ST_Delay)begin
180            oDoneBulletCycle = 0;
181         end
182         else if (CState == ST_CollisionDetect )begin
183            if (Bcollided_w_tank )
184               oCollision = 1;
185            else
186               oCollision = 0;
187         end
188      end
189      // eoflogic
190
191      // buffer to fix drawing problem, allow time for output to vga -> monitor
192      reg donebuffering;
193      reg [5:0]drawbuffer;
194
195      // collision detection flags
196      wire Bcollided_w_tank;
197      assign Bcollided_w_tank = ((translated_bullet_x + 1) > iotherTank_x) && (
       translated_bullet_x < (iotherTank_x + 21)) && ((translated_bullet_y + 1) > iotherTank_y)
        && (translated_bullet_y < (iotherTank_y + 21));
198      wire Bcollided_w_wall;
199      assign Bcollided_w_wall = (translated_bullet_x < 30) || (translated_bullet_x > 287)
       || (translated_bullet_y < 30) || (translated_bullet_y > 207);
200
201      // outputting back to vga->monitor
202      drawPLUSerase drawneraseit(iCLOCK_50, iresetn, rDraw, rErase,
203                                 translated_bullet_x, translated_bullet_y, oColor,
204                                 oX, oY, owriteEn, DoneDrawOrErase);
205
206   endmodule
207
```

```
1   Animation FSM (Bullet)
2
3   module Animation_Bullet (iCLOCK_50, iresetn, idrawEn, ieraseEn,
4                            ix_pos, iy_pos, ocolor_out,
5                            ox, oy, owriteEn, oDoneSignal);
6
7      input iCLOCK_50, iresetn, idrawEn, ieraseEn;
8      input [8:0]ix_pos;
9      input [7:0]iy_pos;
10     output [2:0]ocolor_out;
11     output reg [8:0]ox;
12     output reg [7:0]oy;
13     output reg owriteEn;
14     output reg oDoneSignal;
15
16     //store starting values
17     reg [8:0]x_start;
18     reg [7:0]y_start;
19
20     reg [2:0]x_counter, y_counter; //up to 2
21     reg draw;
22     assign ocolor_out = draw ? 3'b000 : 3'b111; // DRAW RED, ERASE WHITE
23
24     //STATES and State FFs
25     parameter [2:0] ST_idle = 0, ST_chooseColor = 1, ST_setMem = 2, ST_draw = 3, ST_count
       = 4, ST_Done = 5, ST_Done_Delay = 6;
26
27     reg [2:0] CState, NState;
28     always@(posedge iCLOCK_50) begin
29        if (!iresetn)
30           CState <= ST_idle;
31        else
32           CState <= NState;
33     end
34     //end fsm setup
35
36     //CHANGING STATE
37     always@(*) begin
38        NState <= ST_idle;
39        case(CState)
40              ST_idle: begin
41                 if (idrawEn || ieraseEn)
42                    NState <= ST_chooseColor;
43                 else
44                    NState <= ST_idle;
45                 end
46              ST_chooseColor:
47                    NState <= ST_setMem;
48              ST_setMem:
49                    NState <= ST_draw;
50              ST_draw: begin
51                 if (y_counter == 1 && x_counter == 1) // stop when y_counter at pt (1,1)
52                    NState <= ST_Done;
53                 else
54                    NState <= ST_count;
55                 end
56              ST_count:
57                    NState <= ST_draw;
58              ST_Done:
59                    NState <= ST_idle;
```

```
60              default:
61                    NState <= ST_idle;
62          endcase
63       end
64       // end state rules
65
66       //VARIOUS COUNTERS
67       always@(posedge iCLOCK_50) begin
68          if (CState == ST_idle) begin
69             owriteEn = 0;
70             oDoneSignal = 0;
71
72             if(idrawEn)
73                draw = 1;
74             else if(ieraseEn)
75                draw = 0;
76
77             x_start = ix_pos;
78             y_start = iy_pos;
79
80             end
81          if (CState == ST_chooseColor) begin
82
83             end
84          if (CState == ST_setMem) begin
85             x_counter = 0;
86             y_counter = 0;
87             end
88          if (CState == ST_count) begin
89             owriteEn = 0;
90             if (x_counter < 1) // increments x_counter until x_counter is > 1
91                x_counter = x_counter + 1;
92             else begin
93                x_counter = 0; // resets x_counter
94                y_counter = y_counter + 1; // increments y_counter
95                end
96             end
97          if (CState == ST_draw)
98             owriteEn = 1;
99          if (CState == ST_Done)
100            oDoneSignal = 1;
101
102         ox = x_start + x_counter;
103         oy = y_start + y_counter;
104      end
105      // end counters
106   endmodule
107
```

```
 1    Plot Selector
 2
 3    module plotSelector(
 4                              iP1B1_x_pos, iP1B1_y_pos, iP1B1_color, iP1B1_plot,
 5                              iP1B2_x_pos, iP1B2_y_pos, iP1B2_color, iP1B2_plot,
 6                              iP1B3_x_pos, iP1B3_y_pos, iP1B3_color, iP1B3_plot,
 7                              iP1B4_x_pos, iP1B4_y_pos, iP1B4_color, iP1B4_plot,
 8                              iP2B1_x_pos, iP2B1_y_pos, iP2B1_color, iP2B1_plot,
 9                              iP2B2_x_pos, iP2B2_y_pos, iP2B2_color, iP2B2_plot,
10                              iP2B3_x_pos, iP2B3_y_pos, iP2B3_color, iP2B3_plot,
11                              iP2B4_x_pos, iP2B4_y_pos, iP2B4_color, iP2B4_plot,
12                              iTank_x, iTank_y, iTank_color, iTank_plot,
13                              iBulletSelector, iDrawingTank,
14                              iErasingHeart, ix_heart, iy_heart, icolor_heart, iplot_heart,
15                              iStartClearField, ix_clear, iy_clear, icolor_clear,
    iplot_clear,
16                              DrawingWin, x_DrawPlayerWin, y_DrawPlayerWin,
    color_DrawPlayerWin, plot_DrawPlayerWin,
17                              oX_coordinate, oY_coordinate, oColor, oPlot
18                              );
19
20   // inputs
21   input [8:0]iP1B1_x_pos, iP1B2_x_pos, iP1B3_x_pos, iP1B4_x_pos, iP2B1_x_pos, iP2B2_x_pos,
      iP2B3_x_pos, iP2B4_x_pos, iTank_x, ix_heart, ix_clear, x_DrawPlayerWin;
22   input [7:0]iP1B1_y_pos, iP1B2_y_pos, iP1B3_y_pos, iP1B4_y_pos, iP2B1_y_pos, iP2B2_y_pos,
      iP2B3_y_pos, iP2B4_y_pos, iTank_y, iy_heart, iy_clear, y_DrawPlayerWin;
23   input [2:0]iP1B1_color, iP1B2_color, iP1B3_color, iP1B4_color, iP2B1_color, iP2B2_color,
      iP2B3_color, iP2B4_color, iTank_color, icolor_heart, icolor_clear, color_DrawPlayerWin;
24   input [2:0]iBulletSelector;
25   input iP1B1_plot, iP1B2_plot, iP1B3_plot, iP1B4_plot, iP2B1_plot, iP2B2_plot, iP2B3_plot
    , iP2B4_plot, iTank_plot, iDrawingTank, iErasingHeart, iplot_heart, iStartClearField,
    iplot_clear;
26   input DrawingWin, plot_DrawPlayerWin;
27
28   // outputs
29   output [8:0]oX_coordinate;
30   output [7:0]oY_coordinate;
31   output [2:0]oColor;
32   output oPlot;
33
34   // internal registers
35   reg [8:0]bullet_x;
36   reg [7:0]bullet_y;
37   reg [2:0]bullet_color;
38   reg bullet_plot;
39
40   wire [8:0]temp_x;
41   wire [7:0]temp_y;
42   wire [2:0]temp_color;
43   wire temp_plot;
44
45   wire [8:0]temp_x2;
46   wire [7:0]temp_y2;
47   wire [2:0]temp_color2;
48   wire temp_plot2;
49
50   wire [8:0]temp_x3;
51   wire [7:0]temp_y3;
52   wire [2:0]temp_color3;
53   wire temp_plot3;
```

```
54
55    assign oX_coordinate = DrawingWin ? x_DrawPlayerWin : temp_x3;                    // mux
      to choose x position to vga
56    assign oY_coordinate = DrawingWin ? y_DrawPlayerWin : temp_y3;                    // mux
      to choose y position to vga
57    assign oColor = DrawingWin ? color_DrawPlayerWin : temp_color3;                     //
      mux to choose color to vga
58    assign oPlot = DrawingWin ? plot_DrawPlayerWin : temp_plot3;                        //
      mux to choose enable signal for vga
59
60    assign temp_x3 = iStartClearField ? ix_clear : temp_x2;                    // mux to
      choose x position to vga
61    assign temp_y3 = iStartClearField ? iy_clear : temp_y2;                    // mux to
      choose y position to vga
62    assign temp_color3 = iStartClearField ? icolor_clear : temp_color2;
      // mux to choose color to vga
63    assign temp_plot3 = iStartClearField ? iplot_clear : temp_plot2;
      // mux to choose enable signal for vga
64
65    assign temp_x2 = iErasingHeart ? ix_heart : temp_x;                      // mux to
      choose x position to vga
66    assign temp_y2 = iErasingHeart ? iy_heart : temp_y;                      // mux to
      choose y position to vga
67    assign temp_color2 = iErasingHeart ? icolor_heart : temp_color;                     //
      mux to choose color to vga
68    assign temp_plot2 = iErasingHeart ? iplot_heart : temp_plot;                        //
      mux to choose enable signal for vga
69
70    assign temp_x = iDrawingTank ? iTank_x : bullet_x;                   // mux to choose x
      position to vga
71    assign temp_y = iDrawingTank ? iTank_y : bullet_y;                   // mux to choose y
      position to vga
72    assign temp_color = iDrawingTank ? iTank_color : bullet_color;                    // mux
      to choose color to vga
73    assign temp_plot = iDrawingTank ? iTank_plot : bullet_plot;                        // mux
      to choose enable signal for vga
74
75    always@(*)begin
76       if(iBulletSelector == 3'b000) begin
77          bullet_x = iP1B1_x_pos;
78          bullet_y = iP1B1_y_pos;
79          bullet_color = iP1B1_color;
80          bullet_plot = iP1B1_plot;
81       end
82       else if(iBulletSelector == 3'b001) begin
83          bullet_x = iP1B2_x_pos;
84          bullet_y = iP1B2_y_pos;
85          bullet_color = iP1B2_color;
86          bullet_plot = iP1B2_plot;
87       end
88       else if(iBulletSelector == 3'b010) begin
89          bullet_x = iP1B3_x_pos;
90          bullet_y = iP1B3_y_pos;
91          bullet_color = iP1B3_color;
92          bullet_plot = iP1B3_plot;
93       end
94       else if(iBulletSelector == 3'b011) begin
95          bullet_x = iP1B4_x_pos;
96          bullet_y = iP1B4_y_pos;
97          bullet_color = iP1B4_color;
```

```
98          bullet_plot = iP1B4_plot;
99       end
100      else if(iBulletSelector == 3'b100) begin
101         bullet_x = iP2B1_x_pos;
102         bullet_y = iP2B1_y_pos;
103         bullet_color = iP2B1_color;
104         bullet_plot = iP2B1_plot;
105      end
106      else if(iBulletSelector == 3'b101) begin
107         bullet_x = iP2B2_x_pos;
108         bullet_y = iP2B2_y_pos;
109         bullet_color = iP2B2_color;
110         bullet_plot = iP2B2_plot;
111      end
112      else if(iBulletSelector == 3'b110) begin
113         bullet_x = iP2B3_x_pos;
114         bullet_y = iP2B3_y_pos;
115         bullet_color = iP2B3_color;
116         bullet_plot = iP2B3_plot;
117      end
118      else if(iBulletSelector == 3'b111) begin
119         bullet_x = iP2B4_x_pos;
120         bullet_y = iP2B4_y_pos;
121         bullet_color = iP2B4_color;
122         bullet_plot = iP2B4_plot;
123      end
124      else begin
125         bullet_x = 0;
126         bullet_y = 0;
127         bullet_color = 0;
128         bullet_plot = 0;
129      end
130   end
131
132   endmodule
133
```