2 Player Helicopter Game

PROJECT REPORT

Owen Zhang - g2zhango - 998355694

Anar Rahimli - g2gooner - 998122552

CSC258  
Steve Engels

05/04/2013

Introduction

When we were deciding on what to do for our final project we wanted to do something that would be interesting to implement, that captured the essential part of what we learned during the course, and would be fun to use afterwards. We came up with many projects ideas, but none of them appealed to us as much as implementing a game in Verilog. The reason for this was that during the course we performed various implementations during our lab sessions, that were challenging and also taught us a lot about hardware and coding for hardware. However, we never implemented anything as big as a decent game that combined most of the knowledge we learned during the course and that allowed us to explore the more interesting possibilities we could explore using Verilog. This final project gave us the freedom to do what we were interested in doing and dive into more interesting opportunities to implement in Verilog.

The game we took inspiration from is based on a famous Flash game called “Helicopter” (http://www.helicoptergame.net/). The main idea of the game is to avoid blocks for as long as you can, obtaining more points the longer you stay alive. This is one of those games that has consumed many teenagers’ time during high-school when it was launched. We decided to borrow the idea while adding a small twist to the game. Instead of making the game for a single player, we implemented the game to involve two players. In our implementation to make the game more fun we made it so that the blocks are controlled by a second user, making it harder for the original player. The main objective of the game remains the same, but now instead of playing against a computer that randomly sends out the blocks, in our game another player will have control over the blocks. The user who controls the helicopter also accumulates points for the time he or she can avoid the blocks set up by the other user.  A potential way play this game is to try to get a higher score than the guy that is controlling the blocks, perhaps alternating every time you crash.

The other fun part of the project is that we decided to use switches and keys as the control methods for the players instead of the mouse or keyboard. We wanted to make more use of the D2 board in our implementation so as to differ from the implementation of the original game, also it is much easier to simply use the controls on the DE2 board than it is to read the keyboard’s inputs. Moreover, we also used HEX displays to display the score for the player who controls the helicopter. To make the game more interesting for the players we also increased the overall speed of the game. In our game, the blocks fly faster and the helicopter flies relatively slower when compared to the helicopter in the original game. This way we assured that the game does not get boring for the gamers. The speed can be modified in the code by simply changing the clock to a higher or lower integer value.

The main thing we are proud of is that we used a lot of knowledge and experience we gained throughout the course in our project. In particular, we found that the multiplexer, the finite state machine, the binary to hex decoder, and clocks were very relevant and essential in our implementation of the game. We got experience using them during our lab sessions by applying them to very different ideas, but we found that there were extremely useful in controlling the flow of the game. This provided us with both new experiences and a different outlook on how to use them. In addition to that we also gained a great deal of insight in using the VGA adapter module (<http://www.eecg.utoronto.ca/~jayar/ece241_08F/vga/>) in our game design, which was implemented by the University of Toronto Electrical Engineering department. The VGA adapter was the essential part in making our game, because it allowed us draw pixels on the screen. More in depth information on how the game was implemented will be covered within the next section.

Methods

We implemented our game using a set of modules that are responsible for the functionality of different aspects of the game. There are six modules that we implemented for our game and each of them is described separately below.

The **Main module** (helicopter.v) is responsible for running the game, it includes most of the essential functionalities such as running the clocks as well as the finite state of machine for the helicopter, the obstacles and the score counter. It is the controller module that will ultimately determine what the game will look like.

The different states of the game are all controlled using one main 50MHz clock. There are three slowed down clocks based off of the main clock, which are the following:

1-The helicopter object finite state machine (c)

2-The obstacle object finite state machine (c2)

3-The score counter that gives the score on the hex displays (c3)

These three clocks can be modified to slow down or speed up the rate at which the game is played.

For the helicopter fsm, KEY[3] is used as the input that will change the state. It keeps track of the y-coordinate of the helicopter object. When SW[0] is set to ‘high’, the game begins. When KEY[3] is pressed the y-coordinate decreases, and the helicopter on the screen moves up. This gives the illusion of pressing gas, and causing a helicopter to rise in altitude. When KEY[3] is released, the y-coordinates increase, and the helicopter moves down. This gives the illusion of gravity when no gas is being pressed. The y-coordinate changes at a speed that is determined by the clock ‘c’. The x and y coordinates that this fsm keeps track of is a register that is fed into the Mux module.

The obstacle fsm is similar to the helicopter fsm, except that it controls the coordinates of the obstacle at every positive edge of clock ‘c2’. The coordinates change differently, the x-coordinate is always decreasing, and the y-coordinate changes depending on user inputs KEY[0] and KEY[1], which will either add or subtract from the y-coordinate. We have set limits for the x-coordinate so that when it reaches the left edge of the screen the object will reappear of the right edge, and will do so until the helicopter crashes. If neither KEY[0] or KEY[1] is pressed, the y-coordinate will remain the same.

The crash bit is set to low initially, if the helicopter touches the side or the obstacle, the crash bit will change to high.  This will trigger another state in the helicopter and obstacle fsm. When the crash is high, the mux will initializing the reset\_display module, which is will clear the screen (it is further explained below).

The score fsm uses the clock ‘c3’ to keep track of the score of the game. It uses four hex displays to output the score, so the maximum score you can achieve is 9999, and since we set the clock to change score at every 1/10th of a second, it would take approximately 1000s to reach that score. We hoped that no one would be proficient enough at the game to reach that level. Each display keeps track of a four bit registry that will increase at every positive edge of the clock. Those four bits are translated to the HEX on the DE2 board using the hex\_decoder module. The way we change the 10th, 100th, and 1000th decimal is similar to that of a ripple adder. The second hex counter will only change when the previous is at 9, so will the third and fourth.

The **Hex decoder module** (hex\_decoder.v) is a very simple module that is very similar to what we used and implemented in one of our labs. This module takes in four bits and translates them to the corresponding number on the HEX display.

The **Reset Display module** (reset\_display.v) is responsible for resetting the game into the initial state when the game ends, or when you reset the game using SW[0]. Essentially it loops over every pixel on the screen and sets them to the color white.

The **Mux module** (display\_mux.v) is responsible for the flow and states of the game. We used the idea of the multiplexers, which we learned during our course. What it does is, depending on the state the game is in, it either displays both the helicopter and the block at their initial positions on the screen indicating that the game might be played, or move to the end of game state by removing all the objects from the screen. This is done by checking the states of the ‘crash’, ‘reset’, and ‘done’ variables. Both the helicopter and the block will be displayed whenever switch SW[0] is ‘high’. When SW[0] is ‘low’, the reset state is triggered and the Mux feeds the reset\_display coordinates into the VGA module, thereby clearing the screen.

The **Obstacle module** (display\_obs.v) is responsible for drawing the obstacle in the game. In order to display the obstacle on the screen we had to draw each pixel separately, choose the color, and combine the pixels as a block unit. It was tedious work to do, as each pixel had to be separately specified. Moreover, during our first implementation, whenever the block moved we had a tracing red line behind the block. To fix the problem we drew more pixels at the back of the block and filled them in with white color. This allowed to have smooth movement for the block. (See pictures to see the way we designed the obstacle on paper).

The **Helicopter module** (display\_heli.v),like the obstacle module described above, is responsible for drawing pixels of the helicopter. The idea behind this module is exactly the same as that behind the obstacle module. We just used different pixels for the helicopter to draw it in and used different colors to make it aesthetically pleasant. (See pictures to see the way we designed the helicopter on paper)

Result

Overall, the project went well and the final outcome of the project came out as we planned. We managed to display both the helicopter and the block on the screen and run the game smoothly; we were able to control them and make them interact with each other (i.e. collision). We also managed to implement the score counter for the players. The final project met all the requirements we set in our proposal and provided us with lots of experience in hardware coding. To see the final product in action, go to this link:

<https://www.youtube.com/watch?v=0JtXGyf9G0o&feature=youtube_gdata_player>

Discussion

We achieved all the milestones and personal goals we set in our proposal. At first the project was very hard to implement, but it went much better when we started breaking it down in simpler terms. We started off by looking into topics we did not learn during the course such as megafunctions to display objects on the screen, but then we figured that it would be easier to do by just drawing each object pixel by pixel.  We encountered many challenges along the way of implementing the game; however, with the help of TA’s and the experience from the lab as well as some simple tutorials on how to use the VGA module by the Electrical Engineering department, we figured our way out. The challenges ranged from figuring out how to synchronize everything to make the game run smoothly and properly drawing figures when the objects moved.

One of the major challenges we had to overcome was how to display objects onto the screen. The VGA module only allowed one pixel to be drawn at every clock cycle. So in order to draw a figure, we needed to create a module that quickly drew the object at every clock cycle onto the screen. It needed to be fast enough so that we wouldn’t be able to see the drawing process on the screen. This was accomplished in the display\_helicopter module, which was the first object we implemented. The next issue was that we needed to also draw the obstacle in synchronicity with the helicopter. Since the VGA module only allows for one pixel to be drawn at a time, we needed to incorporate a delay in the clock so that it would draw both objects. The delay needs to also be short enough so that we aren’t able to see that the two objects are drawing at every clock cycle. This is where we needed to implement a mux that would feed the two object’s coordinates. The done signal was added to determine when an object was finished drawing, so that when this signal was fed into the mux it would feed that coordinate into the VGA module. When the done signal is low, it would check the done signal of the other object and feed that through the VGA module. This means that the final product that you see is the two objects being drawn at alternate intervals. Because this is occurring using a 50MHz clock, you see a smooth transition.

Another problem we encountered was less difficult. The VGA module does not have a reset setting, this means that every drawn pixel will stay on the screen until it is called again. So during our first implementations we would see the object movement, but it was accompanied by a trail of duplicated pixels. The way we solved this problem was by surrounding our objects with white pixels, so that in whichever direction they moved, they would be trailed by white, which would blend in with the background giving the illusion of a single object moving. This method worked for the most part, except when the helicopter object crashed. That static image would still remain on the screen even though no more signals were being sent to it. To solve this we implemented a reset\_display module, which essentially just loops over every pixel and sets them white.

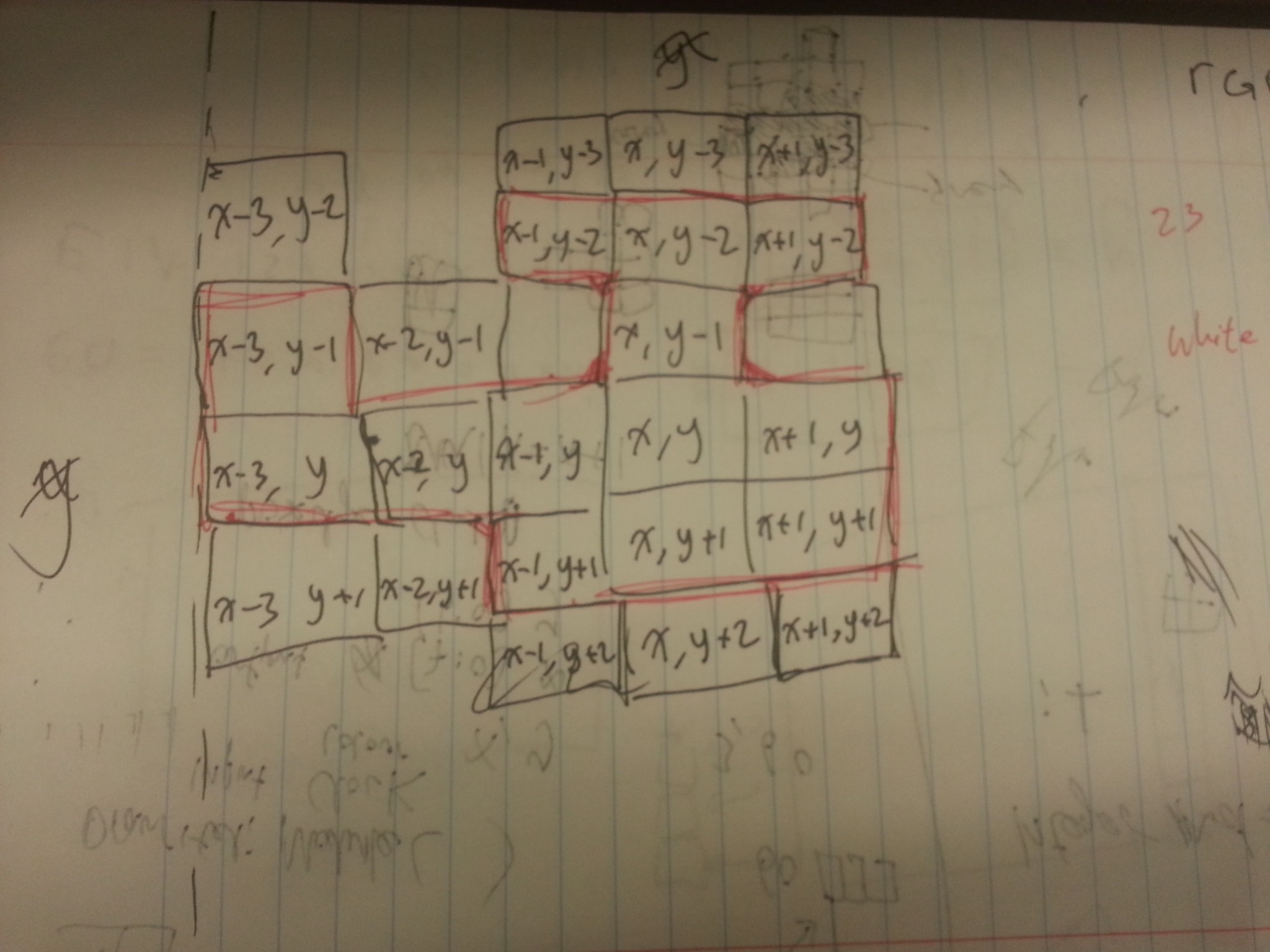
The project allowed us to gain a lot of valuable experience in hardware coding. We have learned a lot of new concepts while implementing the game. Specifically, we learned how to use the VGA adapter and how to draw pixel by pixel and display objects on the screen. Moreover, we learned how to implement the movement of the objects by simply redrawing each pixel at each clock cycle and making the game work smoothly. We have also deepened our understanding of core topics that we explored during the course by exemplifying them in a different environment. In our case, we have learned more about multiplexers, and how they can be used to give the illusion of multiple objects if the clock speed is fast enough. We have a better understanding of the finite state machine by having implemented it for our game as well as clocks by using them a lot in our implementation. All these increased our insight in hardware coding, improved our skills in coding in general, and created more interest in hardware coding and understanding in how everything runs at low level.

There are a couple things we would do differently if we had more time to do the project. First of all, we would have a better design for the game. We could have made a separate module for fsm, and just used that for both objects. We would attempt to make the figures bigger and have a more interesting background. Since the game is implemented by drawing pixels on the background, we could not have made a more aesthetically pleasing background, since everything got erased. The second thing we would have liked to implement is displaying a ‘Game Over’ screen whenever the game ended. That required further research into the VGA module, as the only way we currently know how to display that screen is to draw the word pixel by pixel. Also, we can use switches to speed up or slow down the clock, so that the game can have different modes.

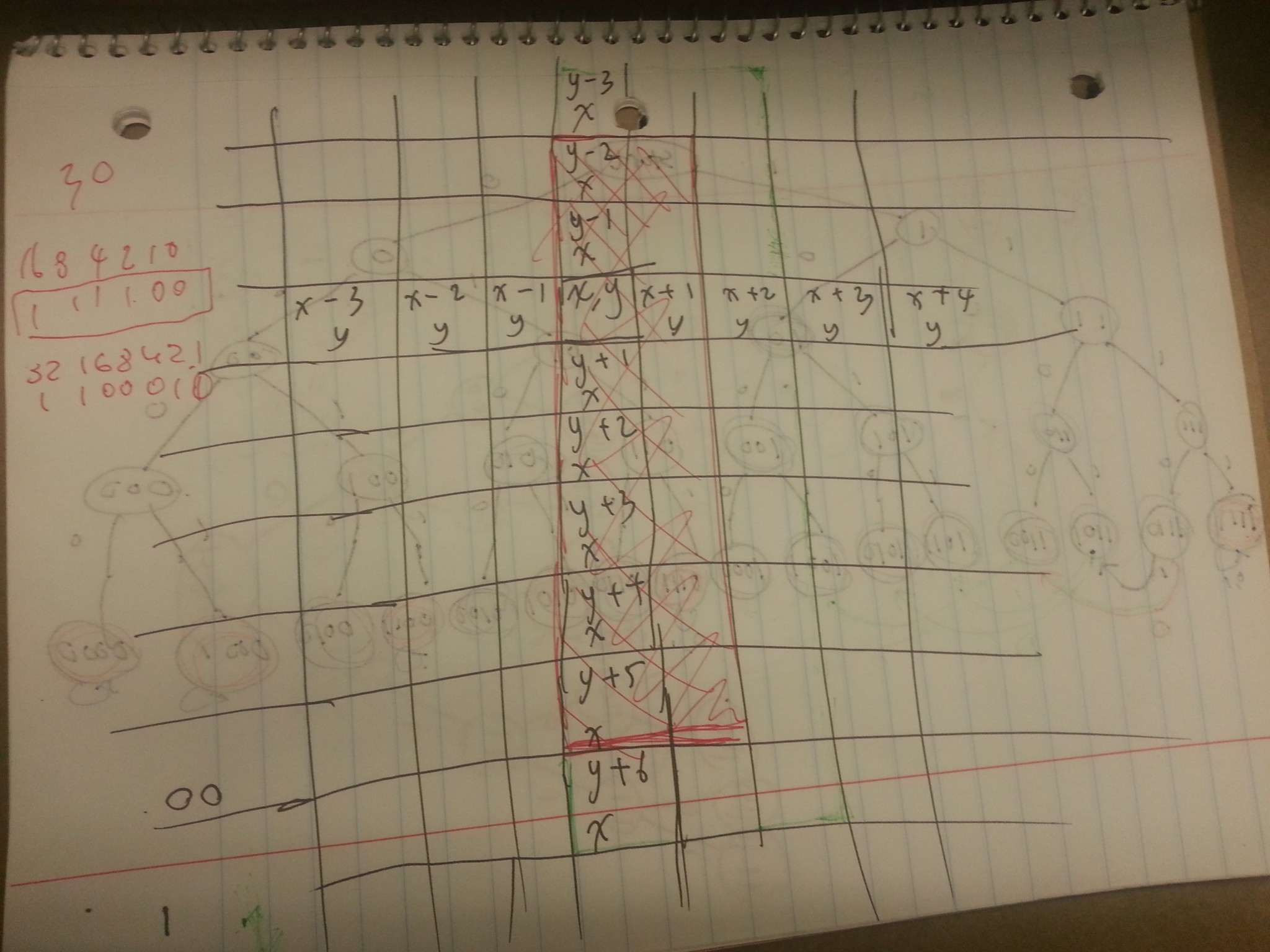
Conclusion

The project was a lot of fun to do. It also was a valuable source of experience to better understand the concepts we learned in the course. It allowed us to design our own game by setting our own milestones and trying to achieve them. Meaning the success or failure of the project is all dependent on us. The project also encouraged us to learn new forms of design and coding as well as experiment in Verilog.

Appendix

A. Screenshots

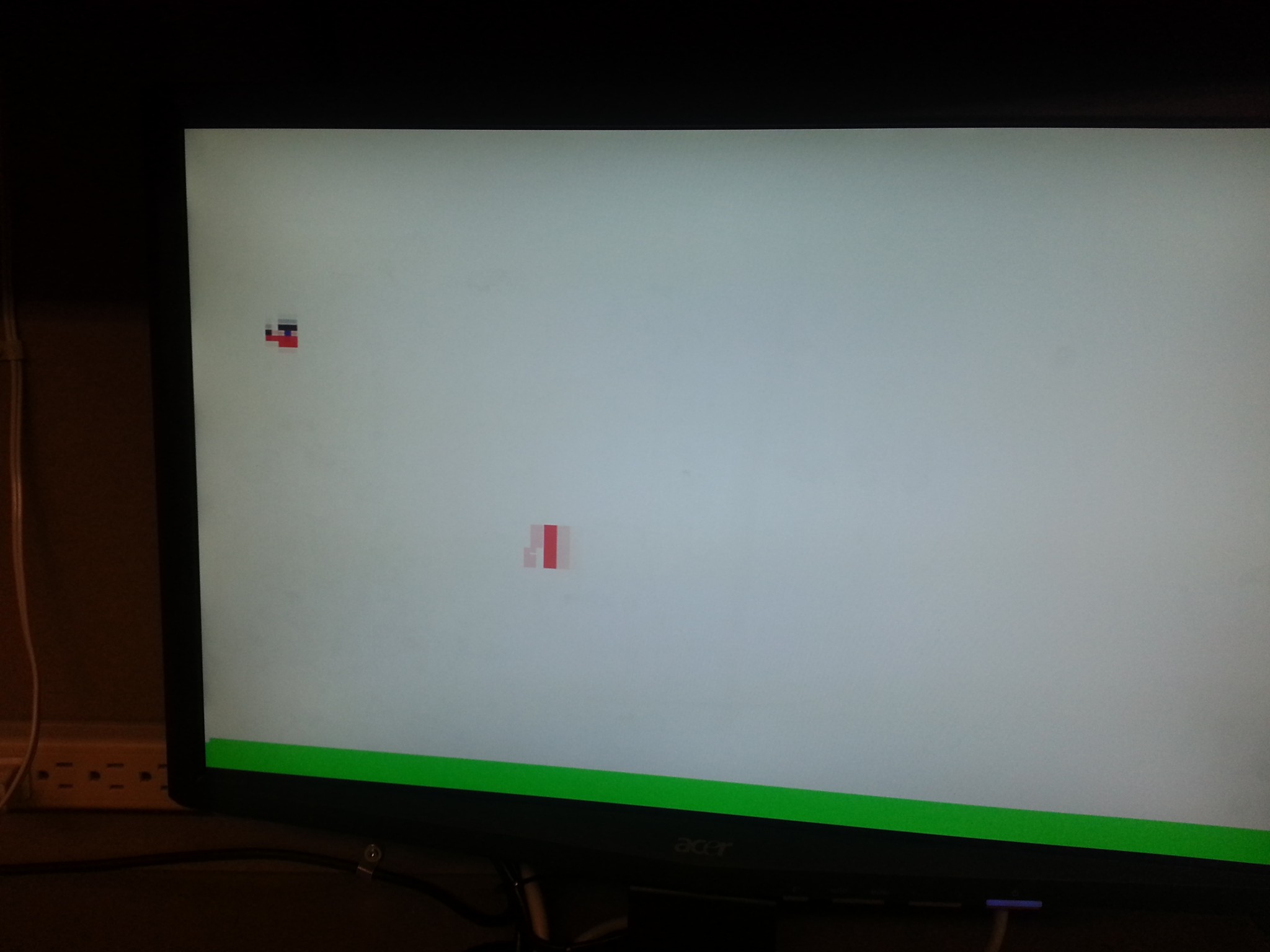
Figuring out how the helicopter should be drawn using the vga. Which pixel coordinate correspond to the x,y values



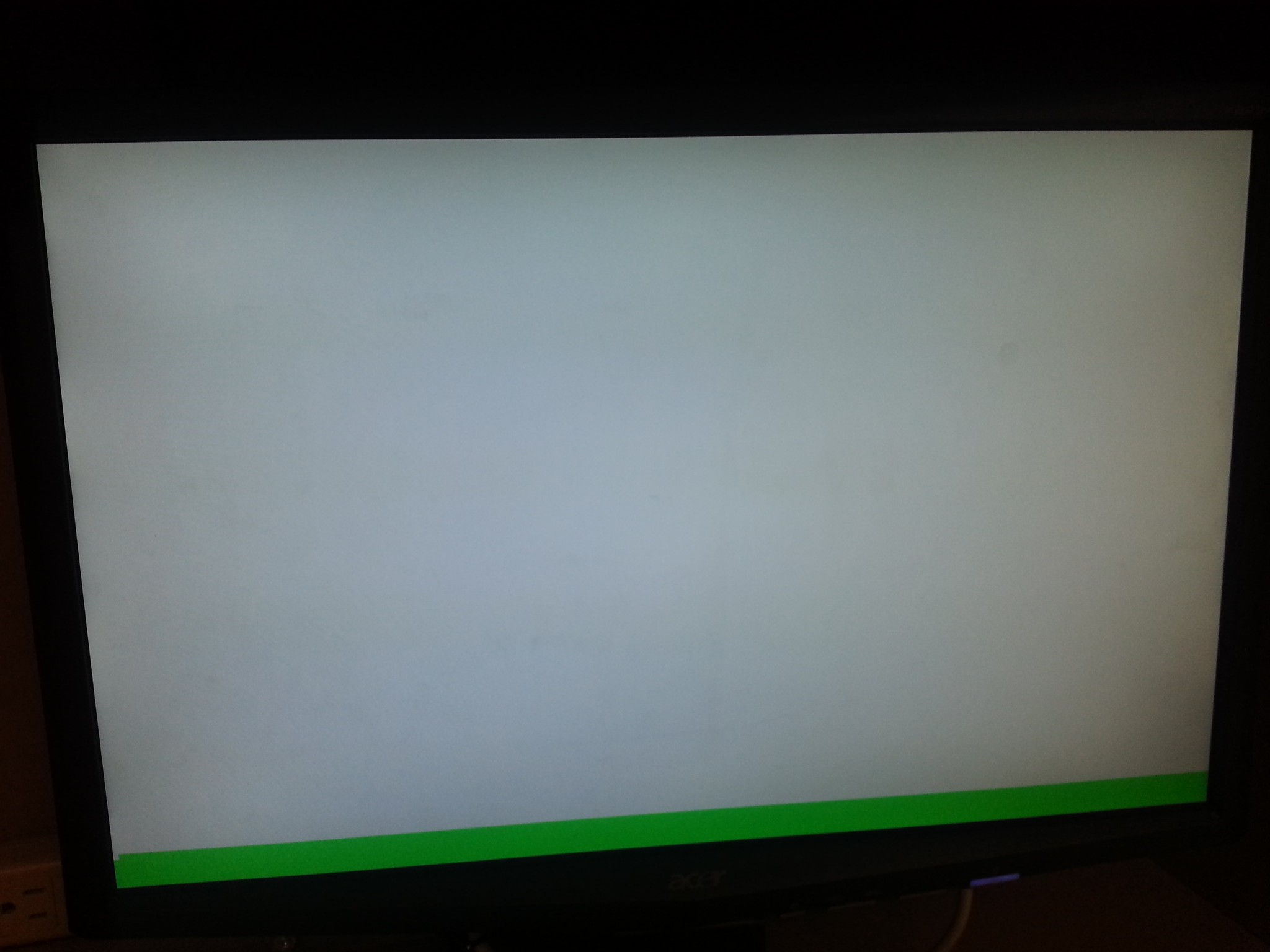
The obstacle needs to be encircled by white pixels so that it doesn’t leave a trail.



The Start screen, when the game at its start position (SW[0] is switched to low).



During the game (SW[0] is switched to high). The helicopter is controlled using KEY[3], and the obstacle is controlled using KEY[0] and KEY[1].



The game over screen, when the helicopter has hit the sides or the obstacle.

B. Code

**-------------------------------------------   Main module   -------------------------------------------**

module helicopter(CLOCK\_50, SW, KEY,VGA\_R, VGA\_G, VGA\_B,VGA\_HS, VGA\_VS, VGA\_BLANK ,VGA\_SYNC, VGA\_CLK,HEX0,HEX1,HEX2,HEX3);  
 input CLOCK\_50;  
 input [3:0] KEY;  
 input [17:0] SW;  
 output [9:0] VGA\_R, VGA\_G, VGA\_B; //for vga module.  
 output VGA\_HS, VGA\_VS, VGA\_BLANK, VGA\_SYNC, VGA\_CLK; //for vga module.  
   
 output [0:6]HEX0,HEX1,HEX2,HEX3;  
   
 wire [7:0] X, X2, X\_final;  
 wire [6:0] Y, Y2, Y\_final;  
   
 reg [7:0] X\_temp, X\_temp2, X\_temp3;  
 reg [6:0] Y\_temp, Y\_temp2, Y\_temp3;  
   
 wire en, en2, en\_final;  
 wire reset = ~SW[17];  
   
 //hex counter  
 reg [3:0] hexcounter, hexcounter2,hexcounter3,hexcounter4;  
   
 reg start;  
 reg start2;  
 wire [2:0]color, color2, color\_final;  
  
 //slowed down clock  
 reg c;  
 reg c2;  
 reg c3;  
   
 //collision detection for crashes  
 reg crash;  
   
   
 integer counter = 0;  
 integer counter2 = 0;  
 integer counter3 =0;  
   
 wire done;  
   
 //heli clock  
 always @(posedge CLOCK\_50)  
 begin  
 counter = counter + 1 ;  
 if (counter == 500000)  
 begin  
 c <= c+1'b1 ;  
 counter = 0;  
 start = 1'b1;  
 end  
 else if (counter == 499999)  
 start = 1'b1;  
 end  
  
 //obstacle clock  
 always @(posedge CLOCK\_50)  
 begin  
 counter2 = counter2 + 1 ;  
 if (counter2 == 200011)  
 begin  
 c2 <= c2+1'b1 ;  
 counter2 = 0;  
 start2 = 1'b1;  
 end  
 else if (counter2 == 200010)  
 start2 = 1'b1;  
 end  
   
 //hex clock  
 always @(posedge CLOCK\_50)  
 begin  
 if (crash == 1'b0)  
 counter3 = counter3 + 1 ;  
 if (counter3 == 5000000)  
 begin  
 c3 <= c3+1'b1 ;  
 counter3 = 0;  
 end  
 end  
   
 always @(posedge c3)  
 if(~SW[0]) begin  
 hexcounter <= 4'b0;  
 hexcounter2 <= 4'b0;  
 hexcounter3 <= 4'b0;  
 hexcounter4 <= 4'b0;  
 end  
   
 else begin  
 hexcounter <= hexcounter + 1;  
 if (hexcounter == 4'b1001)   
 begin  
 hexcounter <= 0;  
 hexcounter2 <= hexcounter2 + 1;  
 end  
   
 if ((hexcounter2 == 4'b1001) & (hexcounter == 4'b1001))  
 begin  
 hexcounter2 <= 0;  
 hexcounter3 <= hexcounter3 + 1;  
 end  
 if ((hexcounter3 == 4'b1001) & (hexcounter2 == 4'b1001)& (hexcounter == 4'b1001))  
 begin  
 hexcounter3 <= 0;  
 hexcounter4 <= hexcounter4 + 1;  
 end  
 if ((hexcounter4 == 4'b1001)&(hexcounter3 == 4'b1001) & (hexcounter2 == 4'b1001)& (hexcounter == 4'b1001))  
 begin  
 hexcounter <=0;  
 hexcounter2 <= 0;  
 hexcounter3 <= 0;  
 hexcounter4 <= 0;  
 end  
 end  
   
 hex\_decoder h1 (hexcounter, HEX0);  
 hex\_decoder h2 (hexcounter2, HEX1);  
 hex\_decoder h3 (hexcounter3, HEX2);  
 hex\_decoder h4 (hexcounter4, HEX3);  
   
 //heli fsm  
 always @(posedge c)  
 begin  
 if (!SW[0])   
 begin // reset state  
 X\_temp = 8'b00001111;  
 Y\_temp = 7'b0111110;  
 crash = 1'b0;  
 end  
 if (crash == 1'b0)  
 begin  
 if (KEY[3]) // down  
 begin  
 if(Y\_temp + 1'b1 != 7'b1110000)  
 begin  
 Y\_temp = Y\_temp + 1'b1;  
   
 if ((X\_temp2 == 8'b00001111) ||(X\_temp2 == 8'b00010000))  
 begin  
 if (Y\_temp == Y\_temp2)  
 crash <= 1'b1;  
 else if (Y\_temp == (Y\_temp2 - 1))  
 crash <= 1'b1;  
 else if (Y\_temp == (Y\_temp2 - 2))  
 crash <= 1'b1;  
 else if (Y\_temp == (Y\_temp2 - 3))  
 crash <= 1'b1;  
 else if (Y\_temp == (Y\_temp2 + 1))  
 crash <= 1'b1;  
 else if (Y\_temp == (Y\_temp2 + 2))  
 crash <= 1'b1;  
 else if (Y\_temp == (Y\_temp2 + 3))  
 crash <= 1'b1;  
 else if (Y\_temp == (Y\_temp2 + 4))  
 crash <= 1'b1;  
 else if (Y\_temp == (Y\_temp2 + 5))  
 crash <= 1'b1;  
 else if ((Y\_temp-2)==(Y\_temp2 +5 ))  
 crash <= 1'b1;  
 end  
 end  
   
 else if (Y\_temp + 1'b1 == 7'b1110000)  
 crash <= 1'b1;  
   
 end  
 else if (!KEY[3]) // up  
 begin  
 if(Y\_temp - 1'b1 != 7'b0000010)  
 begin  
 Y\_temp = Y\_temp - 1'b1;  
   
 if ((X\_temp2 == 8'b00001111) ||(X\_temp2 == 8'b00010000))  
 begin  
 if (Y\_temp == Y\_temp2)  
 crash <= 1'b1;  
 else if (Y\_temp == (Y\_temp2 - 1))  
 crash <= 1'b1;  
 else if (Y\_temp == (Y\_temp2 - 2))  
 crash <= 1'b1;  
 else if (Y\_temp == (Y\_temp2 - 3))  
 crash <= 1'b1;  
 else if (Y\_temp == (Y\_temp2 + 1))  
 crash <= 1'b1;  
 else if (Y\_temp == (Y\_temp2 + 2))  
 crash <= 1'b1;  
 else if (Y\_temp == (Y\_temp2 + 3))  
 crash <= 1'b1;  
 else if (Y\_temp == (Y\_temp2 + 4))  
 crash <= 1'b1;  
 else if (Y\_temp == (Y\_temp2 + 5))  
 crash <= 1'b1;  
 else if ((Y\_temp-2)==(Y\_temp2 +5 ))  
 crash <= 1'b1;  
 end  
 end  
   
 else if (Y\_temp - 1'b1 == 7'b0000010)  
 crash <= 1'b1;  
   
 end  
 end  
 end  
   
 //obstacle fsm  
 always @(posedge c2)  
 begin  
 if (!SW[0])   
 begin // reset state  
 X\_temp2 = 8'b10011011;  
 Y\_temp2 = 7'b0111110;  
 end  
 else // left  
 begin  
 X\_temp2 = X\_temp2 - 1'b1;  
 if (X\_temp2 == (8'b0 - 8'b11))   
 begin  
 X\_temp2 = 8'b10011011;  
 end  
 if (!KEY[0])   
 begin  
 if(Y\_temp2 + 1'b1 != 7'b1110000)  
 Y\_temp2 = Y\_temp2 + 1'b1;  
 end  
 else if (!KEY[1])  
 begin  
 if(Y\_temp2 - 1'b1 != 7'b0000010)  
 Y\_temp2 = Y\_temp2 - 1'b1;  
 end  
 end  
  
 end  
   
   
 display\_heli d0 (CLOCK\_50, X\_temp, Y\_temp, X, Y, color, done, en, start);  
   
 display\_obs d1 (CLOCK\_50, X\_temp2, Y\_temp2, X2, Y2, color2, done2, en2, start2);  
   
 reset\_display d2(CLOCK\_50, X\_temp3,Y\_temp3, SW[0]);  
   
 display\_mux m0 (done, done2, X, Y, X2, Y2,color, color2, color\_final, en, en2, en\_final, X\_final, Y\_final, crash, X\_temp3, Y\_temp3, SW[0]);  
   
   
 vga\_adapter VGA(  
 .resetn(reset),  
 .clock(CLOCK\_50),  
 .colour(color\_final),  
 .x(X\_final),  
 .y(Y\_final),  
 .plot(en\_final),  
 /\* Signals for the DAC to drive the monitor. \*/  
 .VGA\_R(VGA\_R),  
 .VGA\_G(VGA\_G),  
 .VGA\_B(VGA\_B),  
 .VGA\_HS(VGA\_HS),  
 .VGA\_VS(VGA\_VS),  
 .VGA\_BLANK(VGA\_BLANK),  
 .VGA\_SYNC(VGA\_SYNC),  
 .VGA\_CLK(VGA\_CLK));  
 defparam VGA.RESOLUTION = "160x120";  
 defparam VGA.MONOCHROME = "FALSE";  
 defparam VGA.BITS\_PER\_COLOUR\_CHANNEL = 1;  
 defparam VGA.BACKGROUND\_IMAGE = "background.mif";  
endmodule

**-------------------------------------------   Hex decoder module   -------------------------------------------**

module hex\_decoder(bcd\_in, ss\_out);  
  input [3:0] bcd\_in;  
  output [0:6] ss\_out;  
  reg [0:6] ss\_out;  
    
  always @(bcd\_in)  
     case (bcd\_in)  
        4'b0000 :  
           ss\_out = 7'b0000001;  
        4'b0001 :  
           ss\_out = 7'b1001111;  
        4'b0010 :  
           ss\_out = 7'b0010010;  
        4'b0011 :  
           ss\_out = 7'b0000110;  
        4'b0100 :  
           ss\_out = 7'b1001100;  
        4'b0101 :  
           ss\_out = 7'b0100100;  
        4'b0110 :  
           ss\_out = 7'b1100000;  
        4'b0111 :  
           ss\_out = 7'b0001111;  
        4'b1000 :  
           ss\_out = 7'b0000000;  
        4'b1001 :  
           ss\_out = 7'b0001100;  
 default :  
           ss\_out = 7'b0000001;  
     endcase  
endmodule

**-------------------------------------------   Display reset module   -------------------------------------------**

module reset\_display(clock, x\_out, y\_out, reset);  
  
input clock;  
input reset;  
output reg [7:0]x\_out;  
output reg [6:0]y\_out;  
  
always @(posedge clock)  
 if(~reset) begin  
 x\_out <= 8'b0;  
 y\_out <= 7'b0;  
 end  
 else begin  
 x\_out <= x\_out + 1;  
   
 if (x\_out == 8'd159) begin  
 x\_out <= 0;  
 y\_out <= y\_out + 1;  
 end  
   
 if (y\_out == 7'd119)  
 y\_out <= 0;  
 end  
endmodule

**-------------------------------------------   Mux module   -------------------------------------------**

module display\_mux(done, done2, x1,y1,x2,y2, ,color, color2,color\_out, en1,en2,en\_out,x\_out, y\_out, crash, x\_crash, y\_crash, reset);  
 input done, done2;  
 input [7:0] x1,x2, x\_crash;  
 input [6:0] y1,y2, y\_crash;  
 input [2:0]color, color2;  
 input en1,en2;  
 input crash;  
 input reset;  
   
 reg en\_on = 1'b1;  
   
 output reg[7:0] x\_out;  
 output reg[6:0] y\_out;  
 output reg en\_out;  
 output reg[2:0] color\_out;  
   
 always @(\*)  
 begin  
   
 if (!reset)  
 begin   
 x\_out <= x\_crash;  
 y\_out <= y\_crash;  
 color\_out <= 3'b111;  
 end  
   
 if(crash == 1'b1)   
 begin   
 x\_out <= x\_crash;  
 y\_out <= y\_crash;  
 color\_out <= 3'b111;  
 end  
 else if (crash == 1'b0)  
 begin  
 if ((done == 1'b1) & (done2 == 1'b0))  
 begin  
 x\_out <= x2;  
 y\_out <= y2;  
 en\_out <= en2;  
 color\_out <= color2;  
 end  
 else if ((done == 1'b0) & (done2 == 1'b1))  
 begin  
 x\_out <= x1;  
 y\_out <= y1;  
 en\_out <= en1;  
 color\_out <= color;  
 end   
 end  
 end  
 endmodule

**-------------------------------------------   Obstacle module   -------------------------------------------**

module display\_obs(clock, x, y, out\_x, out\_y, color, done, en,start);  
 input clock;  
 input [7:0]x;  
 input [6:0]y;  
 input start;  
 output reg en;  
 output reg[7:0] out\_x;  
 output reg[6:0] out\_y;  
 output reg[2:0] color;  
   
 reg [5:0]d = 6'b000000;  
 output reg done;  
   
 always @(posedge clock)  
 if (d == 6'b000000)  
 begin  
 en = 1'b1;  
 out\_x <= x;  
 out\_y <= y - 2;  
 color <= 3'b100;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d == 6'b000001)  
 begin  
 en = 1'b1;  
 out\_x <= x;  
 out\_y <= y - 1;  
 color <= 3'b100;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==6'b000010)  
 begin  
 en = 1'b1;  
 out\_x <= x;  
 out\_y <= y;  
 color <= 3'b100;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==6'b000011)  
 begin  
 en = 1'b1;  
 out\_x <= x;  
 out\_y <= y + 1;  
 color <= 3'b100;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==6'b000100)  
 begin  
 en = 1'b1;  
 out\_x <= x;  
 out\_y <= y + 2;  
 color <= 3'b100;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==6'b000101)  
 begin  
 en = 1'b1;  
 out\_x <= x;  
 out\_y <= y + 3;  
 color <= 3'b100;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==6'b000110)  
 begin  
 en = 1'b1;  
 out\_x <= x;  
 out\_y <= y + 4;  
 color <= 3'b100;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==6'b000111)  
 begin  
 en = 1'b1;  
 out\_x <= x;  
 out\_y <= y + 5;  
 color <= 3'b100;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==6'b001000)  
 begin  
 en = 1'b1;  
 out\_x <= x + 1;  
 out\_y <= y - 2;  
 color <= 3'b100;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==6'b001001)  
 begin  
 en = 1'b1;  
 out\_x <= x + 1;  
 out\_y <= y - 1;  
 color <= 3'b100;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
   
 else if (d ==6'b001010)  
 begin  
 en = 1'b1;  
 out\_x <= x + 1;  
 out\_y <= y;  
 color <= 3'b100;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==6'b001011)  
 begin  
 en = 1'b1;  
 out\_x <= x + 1;  
 out\_y <= y + 1;  
 color <= 3'b100;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==6'b001100)  
 begin  
 en = 1'b1;  
 out\_x <= x + 1;  
 out\_y <= y + 2;  
 color <= 3'b100;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==6'b001101)  
 begin  
 en = 1'b1;  
 out\_x <= x + 1;  
 out\_y <= y + 3;  
 color <= 3'b100;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==6'b001110)  
 begin  
 en = 1'b1;  
 out\_x <= x + 1;  
 out\_y <= y + 4;  
 color <= 3'b100;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==6'b001111)  
 begin  
 en = 1'b1;  
 out\_x <= x + 1;  
 out\_y <= y + 5;  
 color <= 3'b100;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==6'b010000)  
 begin  
 en = 1'b1;  
 out\_x <= x;  
 out\_y <= y + 6;  
 color <= 3'b111;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==6'b010001)  
 begin  
 en = 1'b1;  
 out\_x <= x + 1;  
 out\_y <= y + 6;  
 color <= 3'b111;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==6'b010010)  
 begin  
 en = 1'b1;  
 out\_x <= x;  
 out\_y <= y - 3;  
 color <= 3'b111;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==6'b010011)  
 begin  
 en = 1'b1;  
 out\_x <= x + 1;  
 out\_y <= y - 3;  
 color <= 3'b111;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==6'b010100)  
 begin  
 en = 1'b1;  
 out\_x <= x + 2;  
 out\_y <= y - 3;  
 color <= 3'b111;  
 d <= d + 1;  
 done <= 1'b0;  
 end   
 else if (d ==6'b010101)  
 begin  
 en = 1'b1;  
 out\_x <= x + 2;  
 out\_y <= y - 2;  
 color <= 3'b111;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==6'b010110)  
 begin  
 en = 1'b1;  
 out\_x <= x + 2;  
 out\_y <= y - 1;  
 color <= 3'b111;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==6'b010111)  
 begin  
 en = 1'b1;  
 out\_x <= x + 2;  
 out\_y <= y;  
 color <= 3'b111;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==6'b011000)  
 begin  
 en = 1'b1;  
 out\_x <= x + 2;  
 out\_y <= y + 1;  
 color <= 3'b111;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==6'b011001)  
 begin  
 en = 1'b1;  
 out\_x <= x + 2;  
 out\_y <= y + 2;  
 color <= 3'b111;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==6'b011010)  
 begin  
 en = 1'b1;  
 out\_x <= x + 2;  
 out\_y <= y + 3;  
 color <= 3'b111;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==6'b011011)  
 begin  
 en = 1'b1;  
 out\_x <= x + 2;  
 out\_y <= y + 4;  
 color <= 3'b111;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==6'b011100)  
 begin  
 en = 1'b1;  
 out\_x <= x + 2;  
 out\_y <= y + 5;  
 color <= 3'b111;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==6'b011101)  
 begin  
 en = 1'b1;  
 out\_x <= x + 2;  
 out\_y <= y + 6;  
 color <= 3'b111;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 //break  
  
 else if (d==6'b011110)  
 begin  
 en = 1'b0;  
 done <= 1'b1;  
 if (start == 1)    
 begin  
 d <= 6'b000000;  
 end  
 end  
endmodule

**-------------------------------------------   Helicopter module   -------------------------------------------**

module display\_heli(clock, x, y, out\_x, out\_y, color, done, en,start);  
 input clock;  
 input [7:0]x;  
 input [6:0]y;  
 input start;  
 output reg en;  
 output reg[7:0] out\_x;  
 output reg[6:0] out\_y;  
 output reg[2:0] color;  
   
 reg [4:0]d;  
 output reg done;  
   
 always @(posedge clock)  
 if (d == 5'b00000)  
 begin  
 en = 1'b1;  
 out\_x <= x - 3;  
 out\_y <= y - 2;  
 color <= 3'b111;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d == 5'b00001)  
 begin  
 en = 1'b1;  
 out\_x <= x - 1;  
 out\_y <= y - 3;  
 color <= 3'b111;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==5'b00010)  
 begin  
 en = 1'b1;  
 out\_x <= x;  
 out\_y <= y - 3;  
 color <= 3'b111;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==5'b00011)  
 begin  
 en = 1'b1;  
 out\_x <= x + 1;  
 out\_y <= y - 3;  
 color <= 3'b111;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==5'b00100)  
 begin  
 en = 1'b1;  
 out\_x <= x - 1;  
 out\_y <= y - 2;  
 color <= 3'b000;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==5'b00101)  
 begin  
 en = 1'b1;  
 out\_x <= x;  
 out\_y <= y - 2;  
 color <= 3'b000;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==5'b00110)  
 begin  
 en = 1'b1;  
 out\_x <= x + 1;  
 out\_y <= y - 2;  
 color <= 3'b000;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==5'b00111)  
 begin  
 en = 1'b1;  
 out\_x <= x - 3;  
 out\_y <= y - 1;  
 color <= 3'b000;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==5'b01000)  
 begin  
 en = 1'b1;  
 out\_x <= x - 2;  
 out\_y <= y - 1;  
 color <= 3'b111;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==5'b01001)  
 begin  
 en = 1'b1;  
 out\_x <= x;  
 out\_y <= y - 1;  
 color <= 3'b001;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==5'b01010)  
 begin  
 en = 1'b1;  
 out\_x <= x-3;  
 out\_y <= y;  
 color <= 3'b100;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==5'b01011)  
 begin  
 en = 1'b1;  
 out\_x <= x-2;  
 out\_y <= y;  
 color <= 3'b100;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==5'b01100)  
 begin  
 en = 1'b1;  
 out\_x <= x-1;  
 out\_y <= y;  
 color <= 3'b100;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==5'b01101)  
 begin  
 en = 1'b1;  
 out\_x <= x;  
 out\_y <= y;  
 color <= 3'b100;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==5'b01110)  
 begin  
 en = 1'b1;  
 out\_x <= x+1;  
 out\_y <= y;  
 color <= 3'b100;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==5'b01111)  
 begin  
 en = 1'b1;  
 out\_x <= x-3;  
 out\_y <= y+1;  
 color <= 3'b111;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==5'b10000)  
 begin  
 en = 1'b1;  
 out\_x <= x-2;  
 out\_y <= y+1;  
 color <= 3'b111;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==5'b10001)  
 begin  
 en = 1'b1;  
 out\_x <= x-1;  
 out\_y <= y+1;  
 color <= 3'b100;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==5'b10010)  
 begin  
 en = 1'b1;  
 out\_x <= x;  
 out\_y <= y+1;  
 color <= 3'b100;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==5'b10011)  
 begin  
 en = 1'b1;  
 out\_x <= x+1;  
 out\_y <= y+1;  
 color <= 3'b100;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==5'b10100)  
 begin  
 en = 1'b1;  
 out\_x <= x-1;  
 out\_y <= y+2;  
 color <= 3'b111;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==5'b10101)  
 begin  
 en = 1'b1;  
 out\_x <= x-1;  
 out\_y <= y+2;  
 color <= 3'b111;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==5'b10110)  
 begin  
 en = 1'b1;  
 out\_x <= x;  
 out\_y <= y+2;  
 color <= 3'b111;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 else if (d ==5'b10111)  
 begin  
 en = 1'b1;  
 out\_x <= x+1;  
 out\_y <= y+2;  
 color <= 3'b111;  
 d <= d + 1;  
 done <= 1'b0;  
 end  
 //endstate  
 else if (d==5'b11000)  
 begin  
 en = 1'b0;  
 done <= 1'b1;  
 if (start == 1)    
 begin  
 d <= 5'b00000;  
 end  
 end  
endmodule