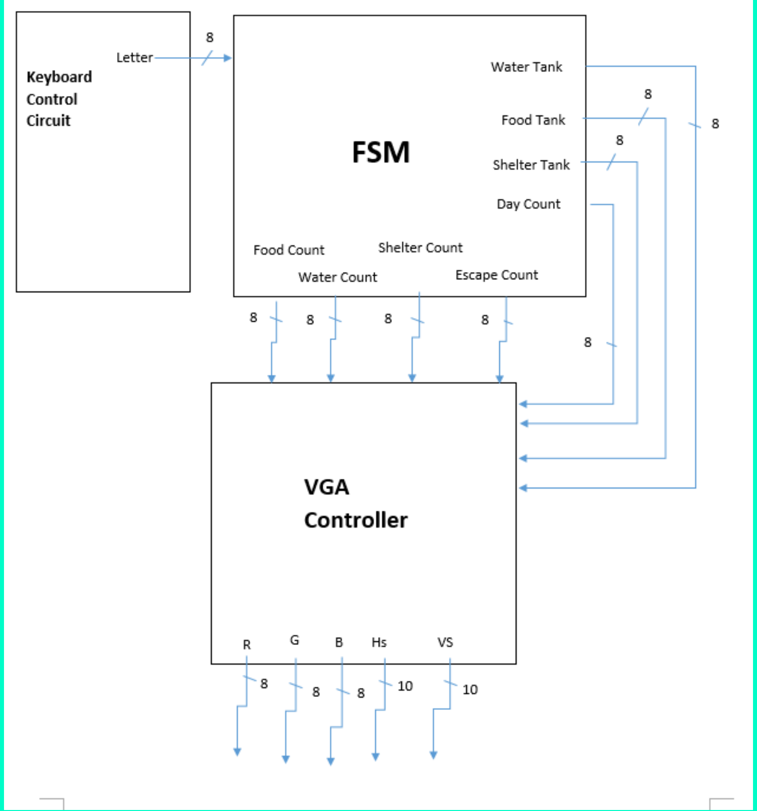
# The Island

Garrett Heiser, Mark Heiser, Alex McInerney, Tyler Wiegand

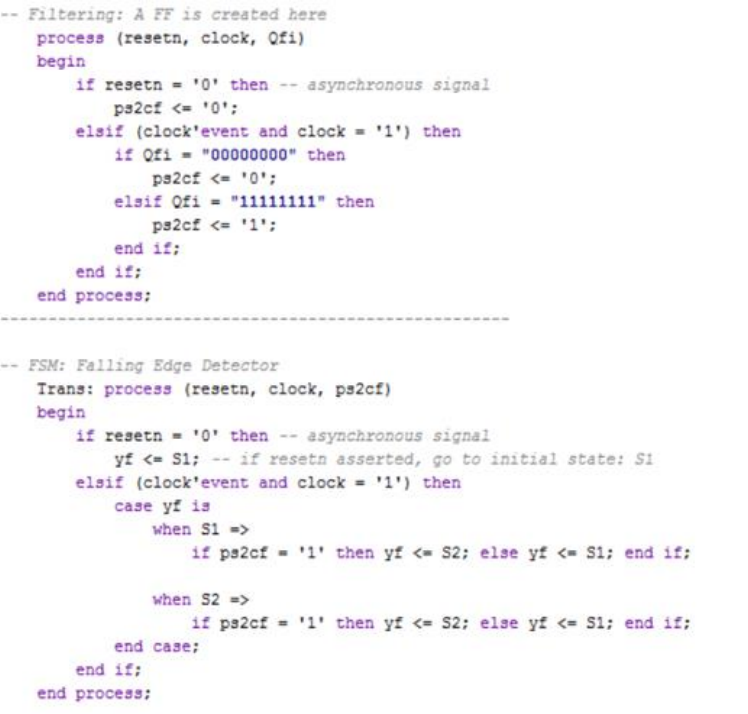
# Top Level Design



# Keyboard Control

The keyboard sends any pressed letter's 8 bit scan code via a 1 bit signal stream of data starting with the MSB.

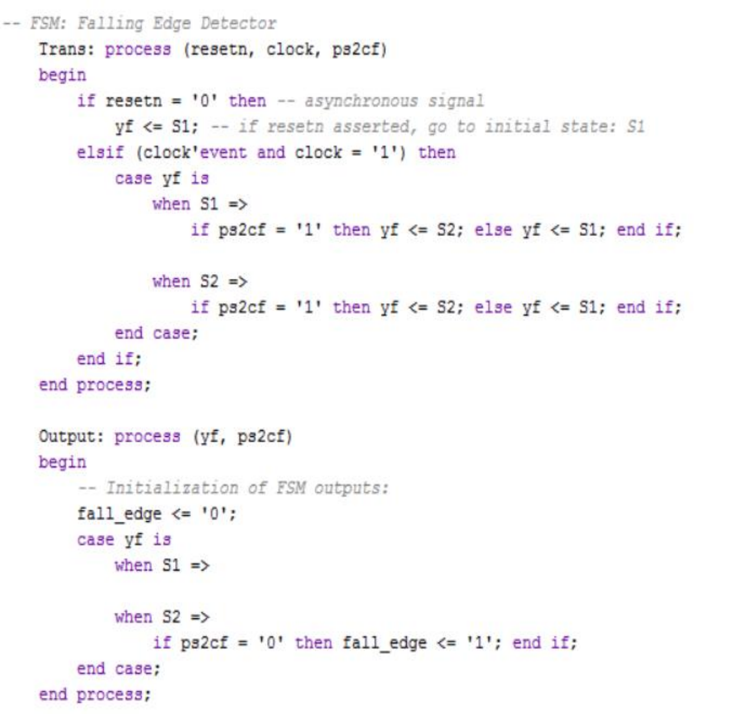
The first sent signal is F0 to indicate an incoming letter, then the code receives the incoming letters scan code along with start and stop bits.



```
-- Filtering: A FF is created here
   process (resetn, clock, Qfi)
   begin
       if resetn = '0' then -- asynchronous signal
           ps2cf <= '0';
       elsif (clock'event and clock = '1') then
           if Qfi = "00000000" then
               ps2cf <= '0';
           elsif Qfi = "11111111" then
               ps2cf <= '1';
           end if;
       end if;
   end process;
-----------------------------------------------------


-- FSM: Falling Edge Detector
   Trans: process (resetn, clock, ps2cf)
   begin
       if resetn = '0' then -- asynchronous signal
           yf <= S1; -- if resetn asserted, go to initial state: S1
       elsif (clock'event and clock = '1') then
           case yf is
               when S1 =>
                   if ps2cf = '1' then yf <= S2; else yf <= S1; end if;

               when S2 =>
                   if ps2cf = '1' then yf <= S2; else yf <= S1; end if;
           end case;
       end if;
   end process;
```

# Keyboard Control

The scan code of the letter is then put into a signal 9 bit output signal from which the 8 LSBs indicating the letter are sent to the FSM.



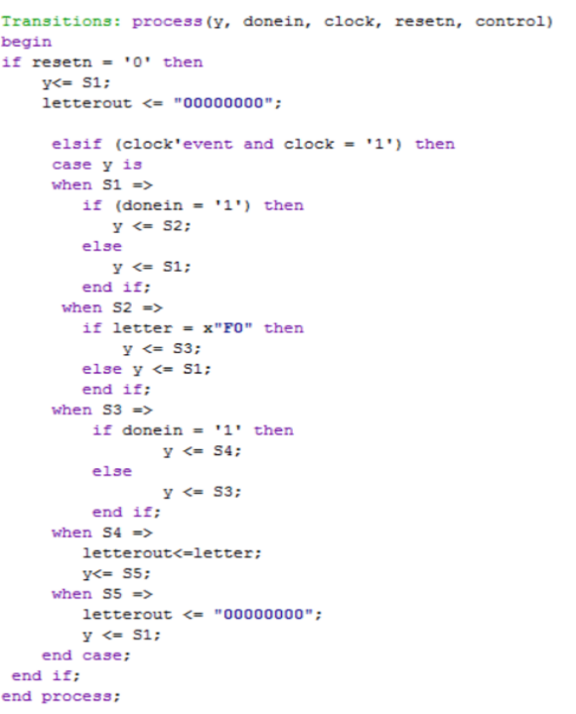
```
-- FSM: Falling Edge Detector
  Trans: process (resetn, clock, ps2cf)
  begin
    if resetn = '0' then -- asynchronous signal
      yf <= S1; -- if resetn asserted, go to initial state: S1
    elsif (clock'event and clock = '1') then
      case yf is
        when S1 =>
          if ps2cf = '1' then yf <= S2; else yf <= S1; end if;

        when S2 =>
          if ps2cf = '1' then yf <= S2; else yf <= S1; end if;
      end case;
    end if;
  end process;

  Output: process (yf, ps2cf)
  begin
    -- Initialization of FSM outputs:
    fall_edge <= '0';
    case yf is
      when S1 =>

      when S2 =>
        if ps2cf = '0' then fall_edge <= '1'; end if;
    end case;
  end process;
```

# Letter Send

The letter control state machine controls the entire process.

Waiting for the F0 and done signal to indicate an incoming letter.

Then waiting for the letter scan code and the second done signal to indicate the full scan code has been received.

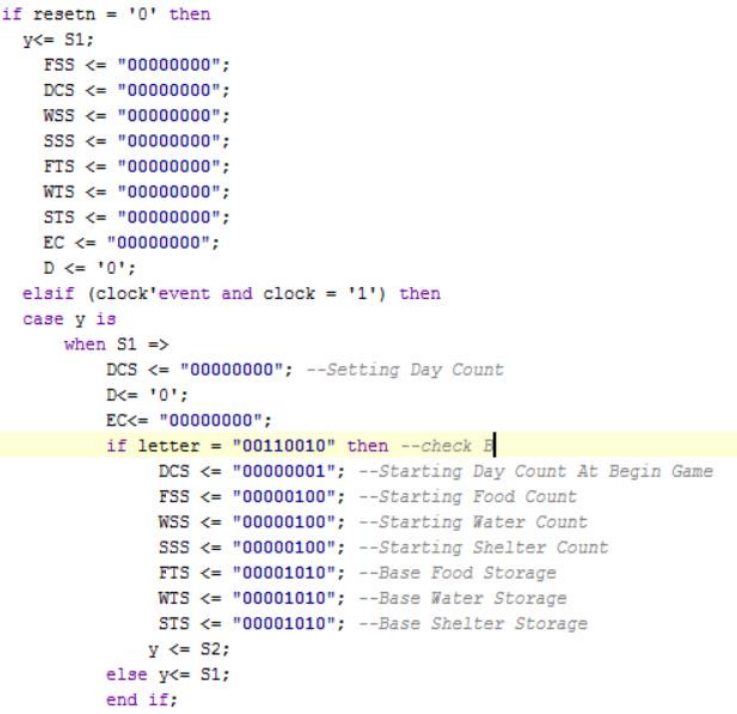


```
Transitions: process(y, donein, clock, resetn, control)
begin
if resetn = '0' then
    y<= S1;
    letterout <= "00000000";

     elsif (clock'event and clock = '1') then
     case y is
     when S1 =>
        if (donein = '1') then
           y <= S2;
        else
           y <= S1;
        end if;
      when S2 =>
        if letter = x"F0" then
            y <= S3;
        else y <= S1;
        end if;
     when S3 =>
         if donein = '1' then
                y <= S4;
         else
                y <= S3;
         end if;
     when S4 =>
        letterout<=letter;
        y<= S5;
     when S5 =>
        letterout <= "00000000";
        y <= S1;
    end case;
 end if;
end process;
```

# Survival FSM

The Survival FSM controls the players state at any point in the game.

Determines the start screen timing and when the player indicates they are ready it starts the game, provides starting materials and limits, and begins counting turns.



```
if resetn = '0' then
  y<= S1;
    FSS <= "00000000";
    DCS <= "00000000";
    WSS <= "00000000";
    SSS <= "00000000";
    FTS <= "00000000";
    WTS <= "00000000";
    STS <= "00000000";
    EC <= "00000000";
    D <= '0';
  elsif (clock'event and clock = '1') then
  case y is
      when S1 =>
          DCS <= "00000000"; --Setting Day Count
          D<= '0';
          EC<= "00000000";
          if letter = "00110010" then --check B
               DCS <= "00000001"; --Starting Day Count At Begin Game
               FSS <= "00000100"; --Starting Food Count
               WSS <= "00000100"; --Starting Water Count
               SSS <= "00000100"; --Starting Shelter Count
               FTS <= "00001010"; --Base Food Storage
               WTS <= "00001010"; --Base Water Storage
               STS <= "00001010"; --Base Shelter Storage
             y <= S2;
          else y<= S1;
          end if;
```

# Survival FSM

The FSM keeps track of all of your resources and ensures that resource limits have not

b

have run c

When the game has

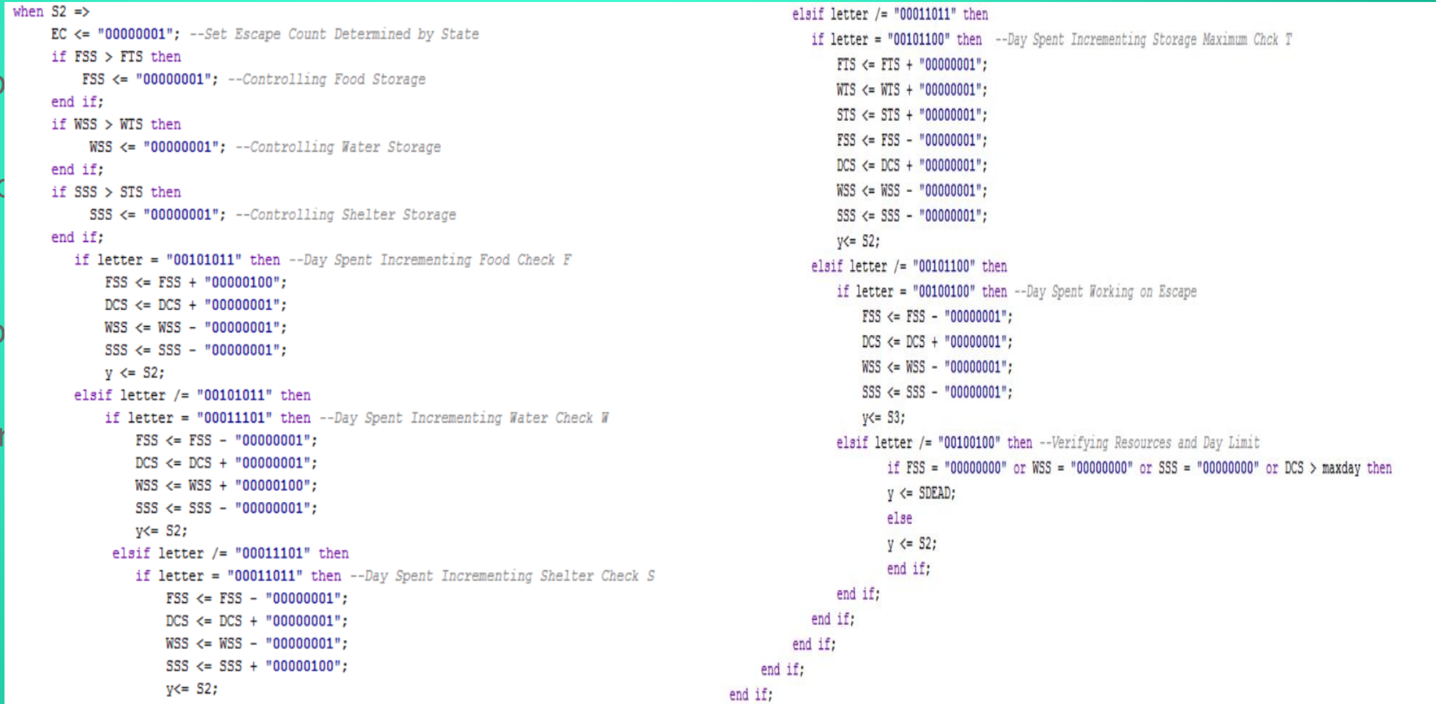
b

the player

appropriate

state and

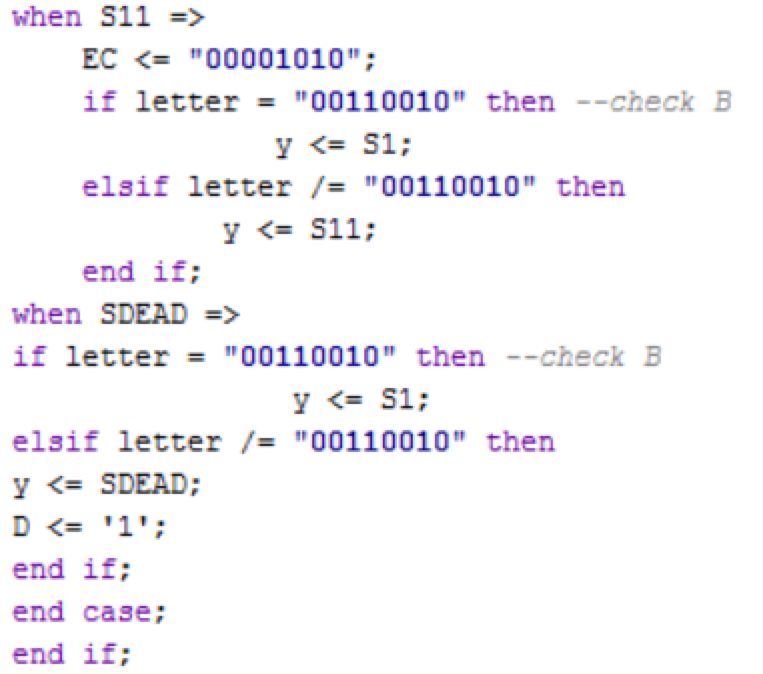
the play



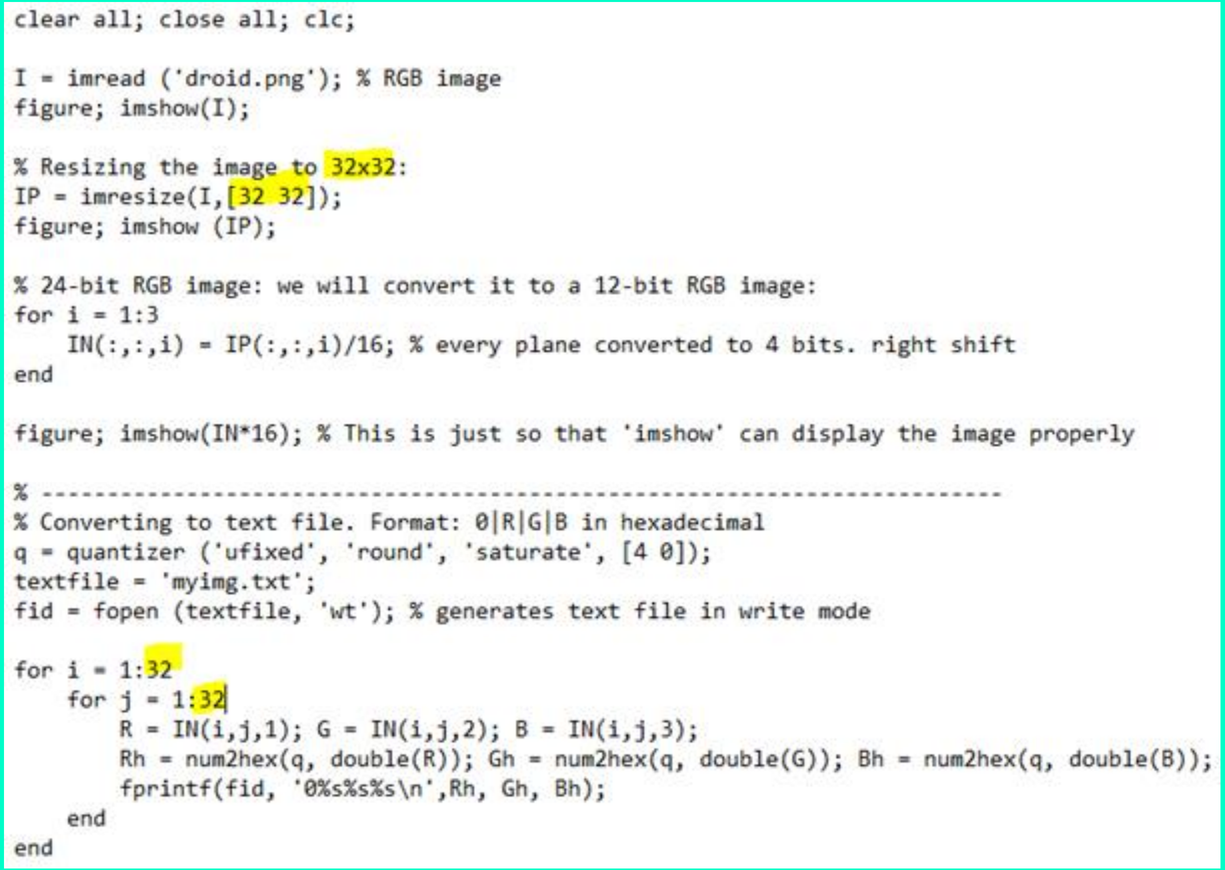
```
when S2 =>
    EC <= "00000001"; --Set Escape Count Determined by State
    if FSS > FTS then
        FSS <= "00000001"; --Controlling Food Storage
    end if;
    if WSS > WTS then
         WSS <= "00000001"; --Controlling Water Storage
    end if;
    if SSS > STS then
         SSS <= "00000001"; --Controlling Shelter Storage
    end if;
       if letter = "00101011" then --Day Spent Incrementing Food Check F
           FSS <= FSS + "00000100";
           DCS <= DCS + "00000001";
           WSS <= WSS - "00000001";
           SSS <= SSS - "00000001";
           y <= S2;
       elsif letter /= "00101011" then
           if letter = "00011101" then --Day Spent Incrementing Water Check W
               FSS <= FSS - "00000001";
               DCS <= DCS + "00000001";
               WSS <= WSS + "00000100";
               SSS <= SSS - "00000001";
               y<= S2;
            elsif letter /= "00011101" then
               if letter = "00011011" then --Day Spent Incrementing Shelter Check S
                   FSS <= FSS - "00000001";
                   DCS <= DCS + "00000001";
                   WSS <= WSS - "00000001";
                   SSS <= SSS + "00000100";
                   y<= S2;
               elsif letter /= "00011011" then
                   if letter = "00101100" then  --Day Spent Incrementing Storage Maximum Chck T
                       FTS <= FTS + "00000001";
                       WTS <= WTS + "00000001";
                       STS <= STS + "00000001";
                       FSS <= FSS - "00000001";
                       DCS <= DCS + "00000001";
                       WSS <= WSS - "00000001";
                       SSS <= SSS - "00000001";
                       y<= S2;
                   elsif letter /= "00101100" then
                       if letter = "00100100" then --Day Spent Working on Escape
                           FSS <= FSS - "00000001";
                           DCS <= DCS + "00000001";
                           WSS <= WSS - "00000001";
                           SSS <= SSS - "00000001";
                           y<= S3;
                       elsif letter /= "00100100" then --Verifying Resources and Day Limit
                               if FSS = "00000000" or WSS = "00000000" or SSS = "00000000" or DCS > maxday then
                               y <= SDEAD;
                               else
                               y <= S2;
                               end if;
                       end if;
                   end if;
               end if;
           end if;
       end if;
```

# Survival FSM

The final two states display if the game has been won or lost on the screen and wait for a ‘b’ scan code to start the game over again.



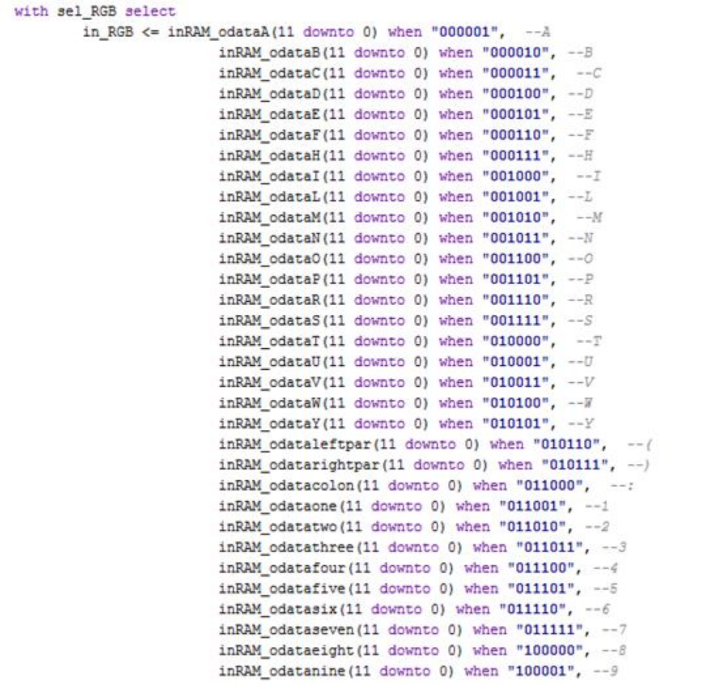
```
when S11 =>
    EC <= "00001010";
    if letter = "00110010" then --check B
               y <= S1;
    elsif letter /= "00110010" then
            y <= S11;
    end if;
when SDEAD =>
if letter = "00110010" then --check B
                y <= S1;
elsif letter /= "00110010" then
y <= SDEAD;
D <= '1';
end if;
end case;
end if;
```



```
clear all; close all; clc;

I = imread ('droid.png'); % RGB image
figure; imshow(I);

% Resizing the image to 32x32:
IP = imresize(I,[32 32]);
figure; imshow (IP);

% 24-bit RGB image: we will convert it to a 12-bit RGB image:
for i = 1:3
    IN(:,:,i) = IP(:,:,i)/16; % every plane converted to 4 bits. right shift
end

figure; imshow(IN*16); % This is just so that 'imshow' can display the image properly

% -------------------------------------------------------------------------
% Converting to text file. Format: 0|R|G|B in hexadecimal
q = quantizer ('ufixed', 'round', 'saturate', [4 0]);
textfile = 'myimg.txt';
fid = fopen (textfile, 'wt'); % generates text file in write mode

for i = 1:32
    for j = 1:32
        R = IN(i,j,1); G = IN(i,j,2); B = IN(i,j,3);
        Rh = num2hex(q, double(R)); Gh = num2hex(q, double(G)); Bh = num2hex(q, double(B));
        fprintf(fid, '0%s%s%s\n',Rh, Gh, Bh);
    end
end
```

- We used this matlab code in order to convert each 32x32 pixel image to a text file

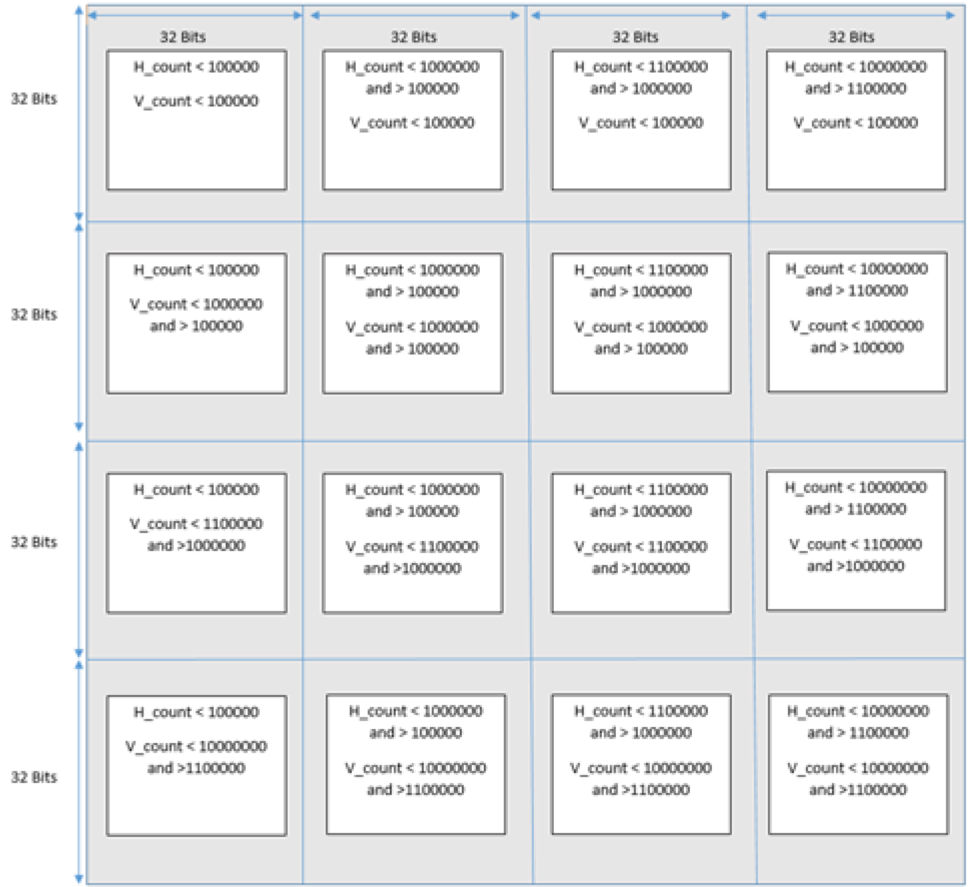
# VGA Controller Code



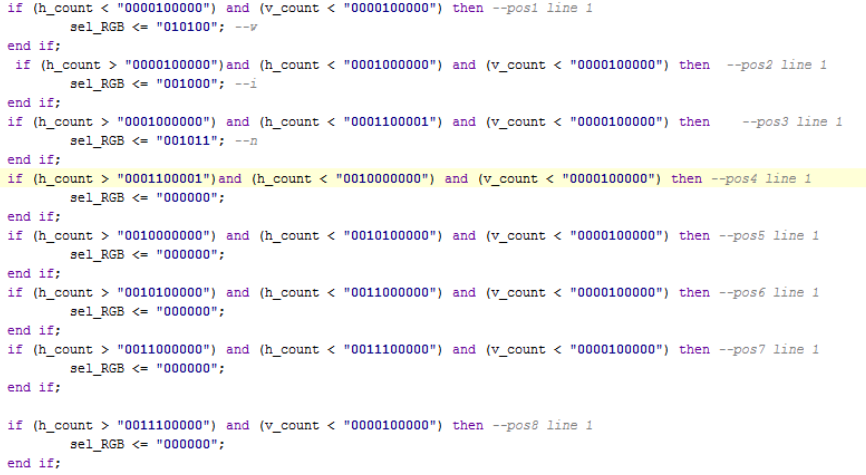
```
with sel_RGB select
        in_RGB <= inRAM_odataA(11 downto 0) when "000001",  --A
                        inRAM_odataB(11 downto 0) when "000010", --B
                        inRAM_odataC(11 downto 0) when "000011",  --C
                        inRAM_odataD(11 downto 0) when "000100", --D
                        inRAM_odataE(11 downto 0) when "000101", --E
                        inRAM_odataF(11 downto 0) when "000110", --F
                        inRAM_odataH(11 downto 0) when "000111", --H
                        inRAM_odataI(11 downto 0) when "001000",  --I
                        inRAM_odataL(11 downto 0) when "001001", --L
                        inRAM_odataM(11 downto 0) when "001010",  --M
                        inRAM_odataN(11 downto 0) when "001011", --N
                        inRAM_odataO(11 downto 0) when "001100", --O
                        inRAM_odataP(11 downto 0) when "001101", --P
                        inRAM_odataR(11 downto 0) when "001110", --R
                        inRAM_odataS(11 downto 0) when "001111", --S
                        inRAM_odataT(11 downto 0) when "010000",  --T
                        inRAM_odataU(11 downto 0) when "010001", --U
                        inRAM_odataV(11 downto 0) when "010011", --V
                        inRAM_odataW(11 downto 0) when "010100", --W
                        inRAM_odataY(11 downto 0) when "010101", --Y
                        inRAM_odataleftpar(11 downto 0) when "010110",  --(
                        inRAM_odatarightpar(11 downto 0) when "010111", --)
                        inRAM_odatacolon(11 downto 0) when "011000",  --:
                        inRAM_odataone(11 downto 0) when "011001", --1
                        inRAM_odatatwo(11 downto 0) when "011010", --2
                        inRAM_odatathree(11 downto 0) when "011011", --3
                        inRAM_odatafour(11 downto 0) when "011100", --4
                        inRAM_odatafive(11 downto 0) when "011101", --5
                        inRAM_odatasix(11 downto 0) when "011110", --6
                        inRAM_odataseven(11 downto 0) when "011111", --7
                        inRAM_odataeight(11 downto 0) when "100000", --8
                        inRAM_odatanine(11 downto 0) when "100001", --9
```

Each image was loaded to the RAM in order to hold the values for future use. We can call any given number or letter using this multiplexor.

# VGA OUTPUT TO MONITOR



| | 32 Bits | 32 Bits | 32 Bits | 32 Bits |
|---|---|---|---|---|
| 32 Bits | H_count < 100000<br>V_count < 100000 | H_count < 1000000<br>and > 100000<br>V_count < 100000 | H_count < 1100000<br>and > 1000000<br>V_count < 100000 | H_count < 10000000<br>and > 1100000<br>V_count < 100000 |
| 32 Bits | H_count < 100000<br>V_count < 1000000<br>and > 100000 | H_count < 1000000<br>and > 100000<br>V_count < 1000000<br>and > 100000 | H_count < 1100000<br>and > 1000000<br>V_count < 1000000<br>and > 100000 | H_count < 10000000<br>and > 1100000<br>V_count < 1000000<br>and > 100000 |
| 32 Bits | H_count < 100000<br>V_count < 1100000<br>and >1000000 | H_count < 1000000<br>and > 100000<br>V_count < 1100000<br>and >1000000 | H_count < 1100000<br>and > 1000000<br>V_count < 1100000<br>and >1000000 | H_count < 10000000<br>and > 1100000<br>V_count < 1100000<br>and >1000000 |
| 32 Bits | H_count < 100000<br>V_count < 10000000<br>and >1100000 | H_count < 1000000<br>and > 100000<br>V_count < 10000000<br>and >1100000 | H_count < 1100000<br>and > 1000000<br>V_count < 10000000<br>and >1100000 | H_count < 10000000<br>and > 1100000<br>V_count < 10000000<br>and >1100000 |

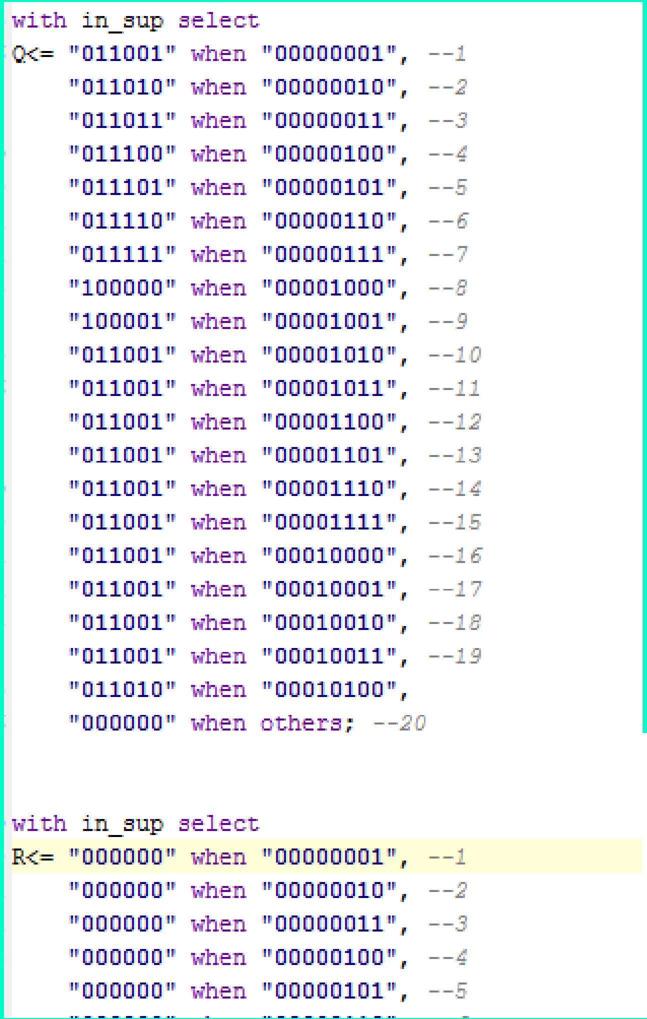


```
if (h_count < "0000100000") and (v_count < "0000100000") then --pos1 line 1
        sel_RGB <= "010100"; --v
end if;
 if (h_count > "0000100000")and (h_count < "0001000000") and (v_count < "0000100000") then  --pos2 line 1
        sel_RGB <= "001000"; --i
end if;
if (h_count > "0001000000") and (h_count < "0001100001") and (v_count < "0000100000") then    --pos3 line 1
        sel_RGB <= "001011"; --n
end if;
if (h_count > "0001100001")and (h_count < "0010000000") and (v_count < "0000100000") then --pos4 line 1
        sel_RGB <= "000000";
end if;
if (h_count > "0010000000") and (h_count < "0010100000") and (v_count < "0000100000") then --pos5 line 1
        sel_RGB <= "000000";
end if;
if (h_count > "0010100000") and (h_count < "0011000000") and (v_count < "0000100000") then --pos6 line 1
        sel_RGB <= "000000";
end if;
if (h_count > "0011000000") and (h_count < "0011100000") and (v_count < "0000100000") then --pos7 line 1
        sel_RGB <= "000000";
end if;

if (h_count > "0011100000") and (v_count < "0000100000") then --pos8 line 1
        sel_RGB <= "000000";
end if;
```

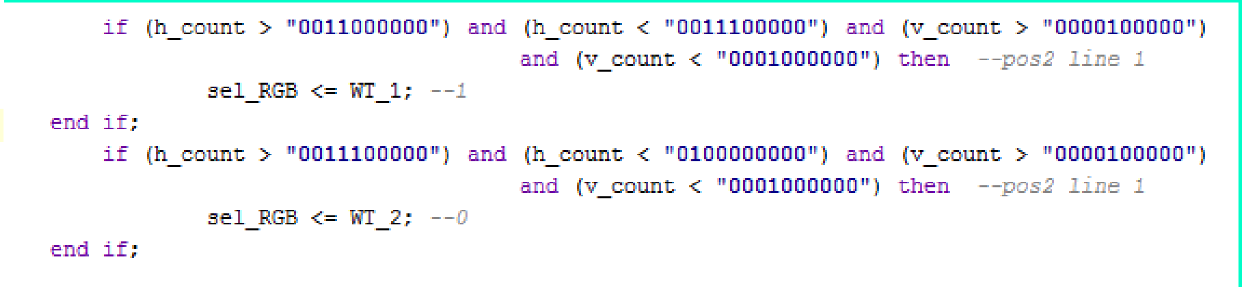
Each letter is a 32x32 bit picture created in paint. To address each picture to the VGA, h_count (horizontal address) and v_count (vertical address) must be determined. For example, the first picture in the top left corner, h_count will be less than 32 and v_count will also be less than 32. The next picture to the right, h_count will be greater than 32 but less than 64 and v_count will remain the same since you are staying on the same horizontal line.

# Incrementing Values

We used a mux in order to select the proper letters to display when you increment different values.



```
with in_sup select
Q<= "011001" when "00000001", --1
    "011010" when "00000010", --2
    "011011" when "00000011", --3
    "011100" when "00000100", --4
    "011101" when "00000101", --5
    "011110" when "00000110", --6
    "011111" when "00000111", --7
    "100000" when "00001000", --8
    "100001" when "00001001", --9
    "011001" when "00001010", --10
    "011001" when "00001011", --11
    "011001" when "00001100", --12
    "011001" when "00001101", --13
    "011001" when "00001110", --14
    "011001" when "00001111", --15
    "011001" when "00010000", --16
    "011001" when "00010001", --17
    "011001" when "00010010", --18
    "011001" when "00010011", --19
    "011010" when "00010100",
    "000000" when others; --20


with in_sup select
R<= "000000" when "00000001", --1
    "000000" when "00000010", --2
    "000000" when "00000011", --3
    "000000" when "00000100", --4
    "000000" when "00000101", --5
```



```
    if (h_count > "0011000000") and (h_count < "0011100000") and (v_count > "0000100000")
                                 and (v_count < "0001000000") then  --pos2 line 1
           sel_RGB <= WT_1; --1
end if;
    if (h_count > "0011100000") and (h_count < "0100000000") and (v_count > "0000100000")
                                 and (v_count < "0001000000") then  --pos2 line 1
           sel_RGB <= WT_2; --0
end if;
```

# Improvements

Difficulty Levels

Highscore Table

Days To Escape/ Days Survived

Instruction Screen

Additional Dialogue

Reason For Game Loss

Random Incrementation

# Goals

IGN Game of the Year 2017