PATTERN SHOOTING GAME

- VINAYAK PATIL AND HARSH VERMA

Introduction :-

Motivation

Our motivation for the project was the movie 'The Matrix'. In the movie the villain has the ability to dodge shots which makes it seem impossible to think that the villain can ever be shot. The protagonist is thrown off by the evasive nature but ultimately realizes a pattern in the enemy's movement, after which he is able to shoot the enemy by predicting his next move. Our game incorporates this in the form of moving 5x5 pixel block that needs to be shot down by the player.

Goal

There were several key aspects of our project that needed integration so as to obtain the final design. Our prime objective with the project was to create a shooter game that was unique. Most shooter games either generate the position of the target randomly or in a pattern which may require memorization. The movement of the target in our game seems random at first, but on closer observation one would realize that there is a pattern to its movement. In keeping with our motivation to create a unique shooter game, our focus was to create a game in which the target block had a pattern to its movement but at the same time the pattern wasn't easily recognizable. We aimed to strike a balance between randomness and predictableness to keep the game interesting.

The Design:

- The design is a target shooting game.
- The working of the game is as follows:
    - the purpose of the game is to hit a target using a cursor which is controlled through the arrow keys on the keyboard within a certain time limit.
    - the target is a 5x5 pixel block which appears on the VGA display for some time, after which it disappears and then reappears at a different spot on the VGA display. This process continues till the time limit is not exhausted.
    - the game has 3 difficulty levels to it. If the player hits the target within the given time limit then the game proceeds to the next level. With an increase in difficulty level, the target appears on the screen for a shorter time before changing its position, the aim cursor speed increases, and the time limit decreases. If the player manages to cross the 3 levels within the timeframe then the game is won or lost otherwise.
    - The plotting of the target is controlled by a 3-bit seed which is generated at the start of every level. Since the seed is a 3-bit number, a 7-digit combination is generated using an LFSR. This combination remains unchanged for the current level, and changes at the beginning of the next level by modifying the seed.
    - an example demonstrating the pattern generation algorithm:
        - suppose seed = 6 (3'b110)
        - the 7-digit pattern generated by the LFSR for this seed will be 6, 7, 3, 5, 2, 1, 4.
            - for the digits 3, 7 : the number 3 is plotted as seen on a dice dot by dot
            - for the digits 4, 2 : the number 4 is plotted as seen on a dice dot by dot
            - for the digits 5, 1 : the number 5 is plotted as seen on a dice dot by dot
            - for the digits 6, 0 : the number 6 is plotted as seen on a dice dot by dot
        - in the above case, the effective 7-digit pattern becomes 6, 3, 3, 5, 4, 5, 4
        - Step 1: the number 6 is plotted as seen on a dice dot by dot

- Step 2: the x, y coordinate is advanced to the next position
- Step 3: the number 3 is plotted as seen on a dice dot by dot
- Step 4: Step 2 occurs again
- … the above steps apply for each of the 7-digits
- after one cycle, the x, y coordinate is advanced to its initial position and the entire process is carried out again till the time runs out.

- Reading a .mif file From ROM:

- an x counter runs from 0-159, y counter runs from 0-119
- the corresponding memory address is calculated
- the color for the corresponding memory address is extracted
- each triplet of x, y, color is passed to the VGA display to be plotted

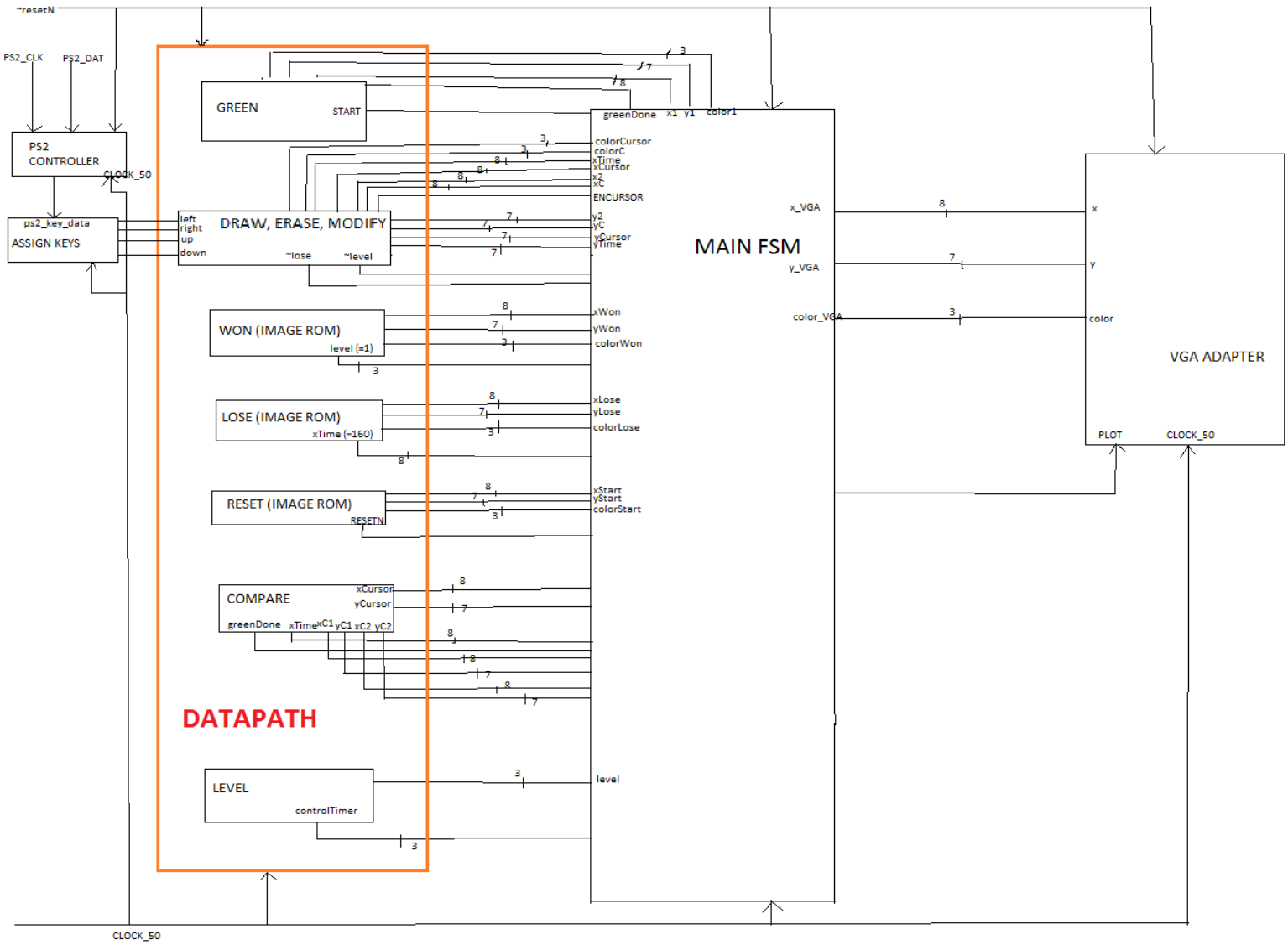


FIGURE 1: MAIN FSM CONTROLLING THE DATAPATH AND PARSING DATA TO THE VGA ADAPTER

- The states of the design (**Figure 1**):
    - The screenshots of the VGA during the different states are included in **Appendix B.**
    - Reset
        - the “start.mif” file is read and plotted on the VGA according to the procedure explained above for reading data from ROM.
        - the triplet, *xStart*, *yStart* & *colorStart* is passed to the VGA.
    - Green
        - a seed (*nReg*) is initialized.
        - depending on the level (*controlTimer*), the time limit (*timerSpeedReg*) is determined.
        - the *Green* module is enabled.
        - *x1, y1, color1* is iterated at posedge of *CLOCK_50* to set up the environment to begin play.
        - as soon as the environment is set, *greenDone* becomes 1.
        - goes to next state: *Compare*
    - Compare
        - checks if the time limit has been exhausted (*xTime==160)*. If yes, then goes to *Lose* state
        - checks if target has been hit by comparing the coordinates of the cursor and that of the target. If hit, then goes to *level* state.
        - If neither of the above two cases is satisfied, then the control goes to the Draw/Erase/Modify state.
    - Draw/Erase/Modify (**Figure 2**):
        - *number = nReg*
        - *x_, y_, color_* are passed to *x_VGA, y_VGA, color_VGA*
        - the value of *enCursor* determines whether:
            - *enCursor=0;* the target is being drawn/erased
            - *enCursor=1;* the aim cursor is being drawn/erased
            - *enCursor=2;* the timer bar is being erased
        - Every number is plotted dot by dot as it is seen on a dice. For each dot, *xC, yC* is the top left pixel.
        - Draw/Erase/Modify module is enabled. This module has 5 main states:
            - *Draw5by5, DelayPattern, reduceTime, EraseNModifyCoordinates, changeNumber*
        - *Draw5by5:*
            - *enCursor=0*
            - *xC, yC* is the coordinate of the top left pixel of the 5x5 target block that needs to be plotted.
            - *countX* and *countY* are iterated in a nested manner from 0-4 to achieve the required plotting.
            - *x_ = xC + countX; y_ = yC + countY; color_ = colorC;*
        - *DelayPattern:*
            - *enCursor=1*
            - a delay counter is implemented to delay the erasing of the target.
            - during this delay, the aim cursor is drawn (x*Cursor = xCursorDraw, yCursor = yCursorDraw, colorCursor = 3’b010)* and erased (*xCursor = xCursorErase, yCursor = yCursorErase, colorCursor = 3’b000)* alternatively according to the value of the arrow keys on the keyboard. *x_ = xCursor, y_ = yCursor, color_ = colorCursor*
            - when the delay gets over, the control shifts to the next state.
        - *reduceTime:*

        - *enCursor=2*
        - in similar manner to *Draw5by5,* nested iterative counters *x2, y2* are added to *xTime* and assigned to *x_, y_, color_ = 3'b000;*
        - the above step erases a portion of the time bar and reduces the time.
    - *EraseNModifyCoordinates:*
        - *enCursor=0*
        - Erase target: the target block is erased using nested iterative counters *countX, countY* and adding them to *xC, yC* and assigning it to *x_, y_, color_ = 3'b000;*
        - Depending on the digit that is being plotted, *xC, yC* is modified so as to change the plotting position of the next dot for the given number.
        - Repeats the Draw5by5, *DelayPattern, reduceTime, EraseNModifyCoordinates* till all the dots for the digit have been plotted. Once all dots are plotted, the control shifts to *changeNumber.*
    - *changeNumber*
        - modify the seed (*number* using an LFSR)
        - change *xC, yC* back to initial position
        - shift control back to *Draw5by5*
- Won:
    - the "Won.mif" file is read and plotted on the VGA according to the procedure explained above for reading data from ROM.
    - the triplet, *xWon*, *yWon*& *colorWon* is passed to the VGA.
- Lose:
    - the "Lose.mif" file is read and plotted on the VGA according to the procedure explained above for reading data from ROM.
    - the triplet, *xLose*, *yLose* & *colorLose* is passed to the VGA.
- Level:
    - enters this state when the player hits the target
    - checks the level (*controlTimer)*
    - if level=3 (i.e *controlTimer==7),* then shifts control to Won state
    - else, increases the difficulty level.

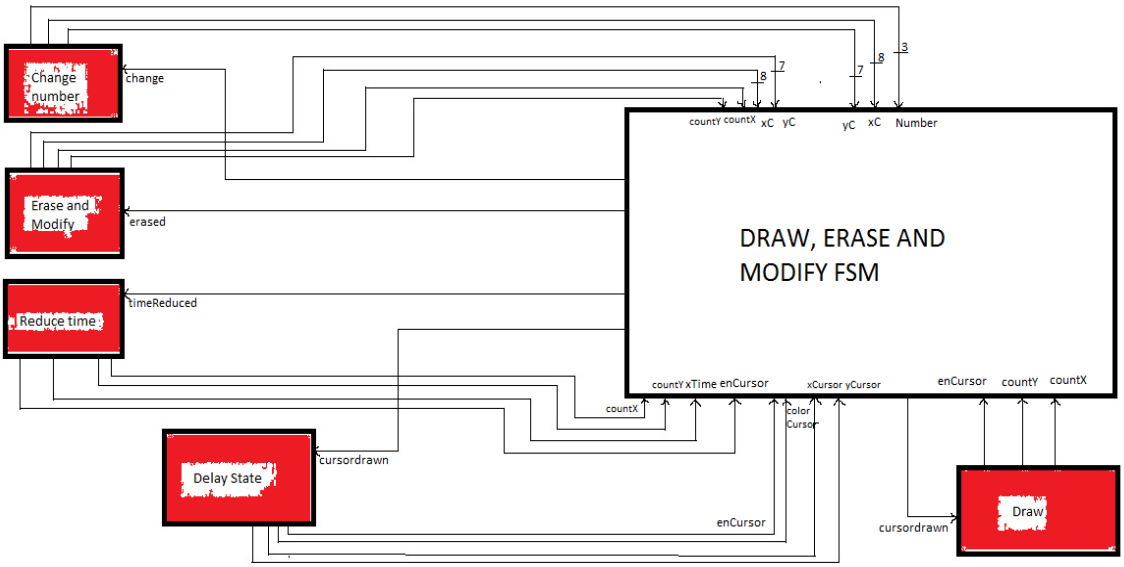


FIGURE 2: BLOCK DIAGRAM OF DRAW, ERASE, AND MODIFY FSM CHOOSING THE OPERATION (DRAW/ERASE/MODIFY)

What was successful :-

- **"Inception"** - We never imagined that we could end up with nested FSMs but we did. It was probably one of the inadvertent highlights of our coding framework and something that eased the debugging process immensely.
- **Keyboard** - Though it might seem easy to get the keyboard to function properly it required a lot of going back and forth to get instantiations correct. Also, interpreting and converting the 8-bit signal from keyboard into individual signals (i.e *left, right, up, down*) was tricky.
- **Modularization** - We were able to simplify our code by breaking it down into modules. It made our job of keeping track of the operations being carried out at any point of time much easier and also made debugging a lot easier.
- **Clubbing operations within the same module** - Even though modularizing is a good practice and helps in debugging we learnt that sometimes it is more convenient to carry out certain operations such as draw/erase in the same module as a *wire* cannot be assigned values in multiple modules. Also, since the VGA can receive one triplet (*x,y,color)* at a time, such a practise helped us transitioning from plotting the pattern to plotting the aim cursor to reducing the timer effectively. This was successfully implemented in the Draw/Erase/Modify module.
- **Levels by Implementing a Linear Feedback Shift Register (LFSR):** Not only did we have to change the seed at the beginning of each level but we also had to modify the seed every time the user hit reset. To change the seed at each the beginning of each level we simply incremented it, while to obtain the 7-digit pattern we used a 3-bit LFSR to modify the seed after the plotting of the pattern for the corresponding digit.

What went wrong :-

- **Lack of Memory -** One problem that we had in our project was the lack of memory towards the end. We didn't keep track of the Read Only Memory (ROM) that we kept allocating and finally we didn't have any memory left so we had to plot certain elements in the page where the game begins.

- **Asynchronous timer** - Another problem that we faced was implementing an asynchronous timer, we began working on the timer almost at the end of our project so it was difficult to integrate it with the 'Draw and Erase' module. Therefore, we were forced to build a synchronous timer that was in sync with the previously stated module.

These were the two major issues that we had to overcome, the rest were minor bugs that we had to fix.

What we could've done differently :-

- **Timer** - If we had brainstormed in a better manner at the beginning of our project then we would've been able to implement an asynchronous timer. Since we didn't think about the timer until the cursor and pattern had been plotted, we weren't able to integrate the timer with the 'DrawandErase' module and implement it in our framework. We had to make do with a synchronous timer that was in sync with the 'DrawAndErase' module.

- **Mouse** - We thought about using the mouse instead of the keyboard to move the cursor but we didn't have enough time to implement it since we also had to get our timer and the compare module to work. To save time we decided to use the arrow keys from the keyboard.

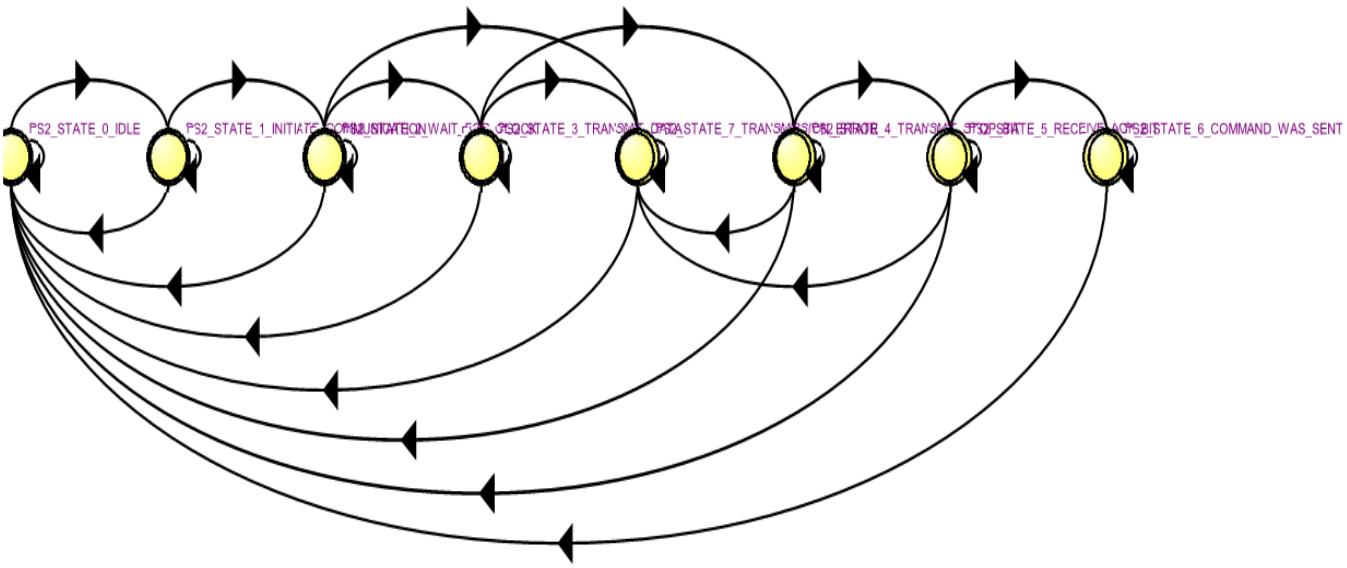
References-

1. Jason Anderson. VGA module skeleton. [Online] Accessed November 18, 2013. Available: https://portal.utoronto.ca under ECE241/lab7

2. Jonathan Rose. PS2 module, IP cores for the Altera DE2 board [Online] Accessed November 22, 2013. Available: http://www.eecg.utoronto.ca/~jayar/ece241_08F/AudioVideoCores/

Appendix -

Appendix A - **State diagram of the PS2 module** :-

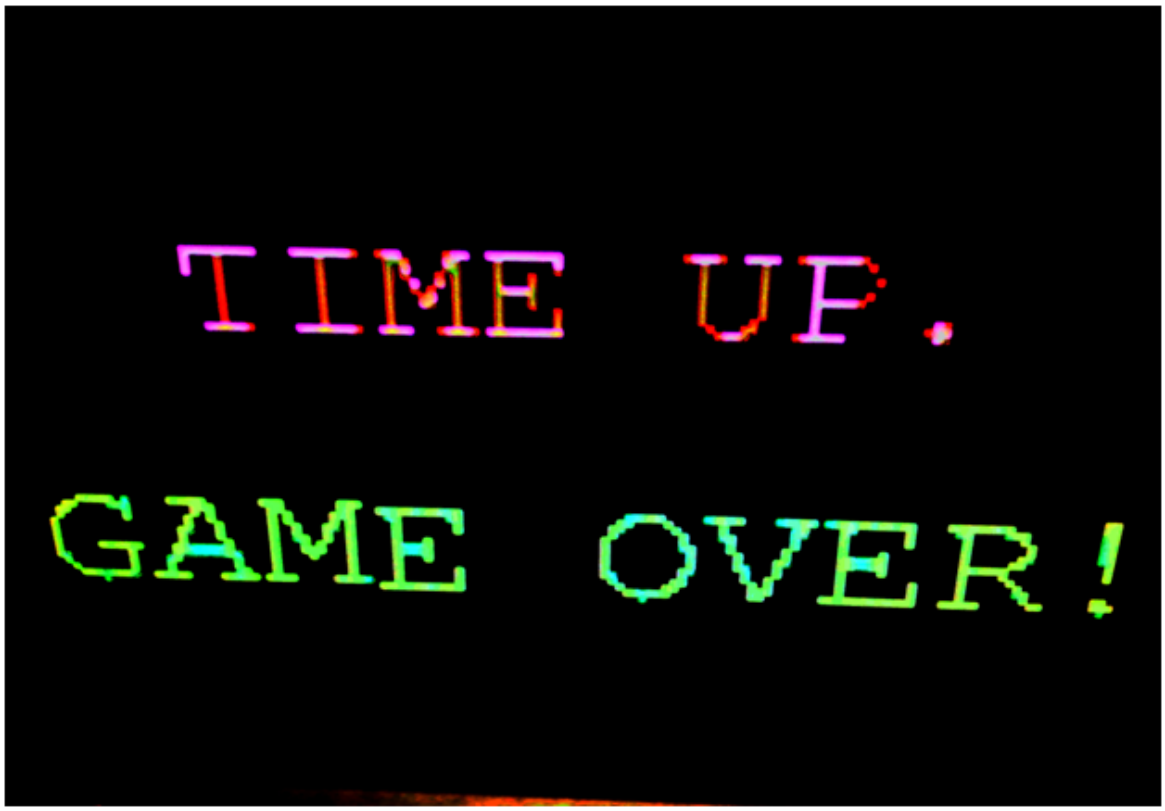


APPENDIX B: Screenshots of the VGA display during different states
Resetn state:

Green state:

Won state:

Lose state:



Appendix C - Verilog code for the project :-

```
module testPattern
   (
         CLOCK_50,                        //   On Board 50 MHz
         LEDR,
         LEDG,
         KEY,                             //   Push Button[3:0]
         SW,                              //   DPDT Switch[17:0]
         VGA_CLK,                         //   VGA Clock
         VGA_HS,                          //   VGA H_SYNC
         VGA_VS,                          //   VGA V_SYNC
         VGA_BLANK,                       //   VGA BLANK
         VGA_SYNC,                        //   VGA SYNC
         VGA_R,                           //   VGA Red[9:0]
         VGA_G,                     //   VGA Green[9:0]
         VGA_B,                           //   VGA Blue[9:0]
```

```
        PS2_CLK,
        PS2_DAT

    );
    input               CLOCK_50;                   //  50 MHz
    output[17:0]LEDR;
    output[7:0]LEDG;
    input   [3:0]   KEY;                    //  Button[3:0]
    input   [17:0]  SW;                         //  Switches[17:0]
    output              VGA_CLK;                    //  VGA Clock
    output              VGA_HS;                     //  VGA H_SYNC
    output              VGA_VS;                     //  VGA V_SYNC
    output              VGA_BLANK;                  //  VGA BLANK
    output              VGA_SYNC;                   //  VGA SYNC
    output  [9:0]   VGA_R;                  //  VGA Red[9:0]
    output  [9:0]   VGA_G;                  //  VGA Green[9:0]
    output  [9:0]   VGA_B;                  //  VGA Blue[9:0]


    // Bidirectionals
    inout                   PS2_CLK;
    inout                   PS2_DAT;

    // Internal Wires
    wire        [7:0]   ps2_key_data;
    wire                    ps2_key_pressed;

    // Internal Registers
    reg             [7:0]   last_data_received;


    wire left;
    wire right;
    wire up;
    wire down;
    wire shoot;
    wire enable;

//    assign left = SW[17];
```

```
//    assign right = SW[16];
//    assign up = SW[15];
//    assign down = SW[14];

    assign enable = left | right | up | down;


    //assign LEDR[14:8]=xC1;
    //assign LEDR[7:0]=xTime;

    assign LEDR[17] = won1;

    assign LEDR[16] = lose;

    //assign LEDG[7:0]=xCursorDraw;

    wire clock;
    //assign clock = KEY[0];

    wire resetn;
    wire start;
    //wire enable;
    //wire drawn;


    reg[2:0]y;
    reg[2:0]Y;


    assign resetn = SW[0];
    assign start =  SW[1];


    // Resetn
    wire[7:0]x1;
    wire[6:0]y1;
    wire[2:0]color1;

    wire[7:0]x2;
```

```
wire[6:0]y2;
wire[2:0]color2;

wire[7:0]xC;
wire[6:0]yC;
wire[2:0]colorC;

wire[7:0]xC1;
wire[6:0]yC1;


wire[7:0]xC2;
wire[6:0]yC2;

wire[7:0]xCursor;
wire[6:0]yCursor;
wire[2:0]colorCursor;


wire[7:0]xCursorDraw;
wire[6:0]yCursorDraw;




assign LEDG[2:0] = y;
assign LEDG[5:3] = Y;

reg[7:0]x_;
reg[6:0]y_;
reg[2:0]color_;

wire[7:0]x_VGA;
wire[6:0]y_VGA;
wire[2:0]color_VGA;

assign x_VGA = x_;
assign y_VGA = y_;
assign color_VGA = color_;
```

```
//assign number = 3'b101;

wire[1:0]enCursor;
wire won;


wire won1;
assign won1 = w;

reg w;

wire lose;
assign lose = l;
reg l;

wire[2:0]ct;


wire[7:0]xWon;
wire[6:0]yWon;
wire[2:0]colorWon;
wire[14:0]address;


wire[7:0]xStart;
wire[6:0]yStart;
wire[2:0]colorStart;
wire[14:0]addressStart;

wire[7:0]xLose;
wire[6:0]yLose;
wire[2:0]colorLose;
wire[14:0]addressLose;

wire[7:0]xTimer;
wire[6:0]yTimer;
wire[2:0]colorTimer;
wire[14:0]addressTimer;
```

```
    wire[7:0]xTime;
    wire[6:0]yTime;
    wire[2:0]colorTime;

    wire[2:0]countXt;
    wire[2:0]countYt;

    //wire[2:0]n1;
    //assign n1 = SW[17:15];

    reg[2:0]nReg;
    wire[2:0]nR;
    assign nR = nReg;

    assign LEDR[2:0] = nR;

    wire greenDone;

    reg[27:0]timerSpeedReg;
    wire[27:0]timerSpeed;
    //assign timerSpeed = 10000000;
    assign timerSpeed = timerSpeedReg;

    reg[2:0]controlTimer;

    reg[2:0]level;

    wire[2:0]lev;
    assign lev = level;

    //assign LEDR[5:3] = lev;


    parameter Resetn=3'b000, Draw=3'b001, Compare=3'b010, Won=3'b011,
Green=3'b100, Lose=3'b101, Level=3'b110;
```

```
always@(*)
begin
        case(y)


        // reset should make the values of x2,y2 to (0,0) if you want
        Resetn:         begin

                                                //timerSpeedReg=50000000;

                                                //controlTimer = 1;
                                                level = 1;
                                                x_ = xStart;
                                                y_ = yStart;
                                                color_ = colorStart;
                                                w = 0;
                                                l = 0;
                                                if(start)Y=Green;
                                                else Y=Resetn;
                                        end

        Green:          begin

//                                              x_ = xTimer;
//                                              y_ = yTimer;
//                                              color_ = colorTimer;

                                                controlTimer=level;
                                                nReg=nReg+1;

                                                if(controlTimer==1)
                                                        timerSpeedReg=50000000;
                                                else if(controlTimer==3)
                                                        timerSpeedReg=25000000;
                                                else if(controlTimer==7)
                                                        timerSpeedReg=10000000;


                                                x_ = x1;
```

```
                    y_ = y1;
                    color_ = color1;
                    if(greenDone) Y=Compare;
                    else Y=Green;

                end

        Compare:    begin


                    if(xTime==160)
                        Y=Lose;


                    else if(xCursor>xC1 & xCursor<xC2 &
yCursor>yC1 & yCursor<yC2)
                        Y=Level;

                    else
                        Y=Draw;

                end

        Draw:
                begin
                    // pattern is plotted
                    if(enCursor==0)
                    begin
                        if(xC!=8'b10011110 &
yC!=7'b1110110)
                        begin
                            x_ = xC + x2;
                            y_ = yC + y2;
                            color_ = colorC;

                        end
                    end

                    else if(enCursor==1)
```

```
                    begin


                        x_ = xCursor;
                        y_ = yCursor;
                        color_ = colorCursor;
                    end

                    else if(enCursor==2)
                    begin
                        x_ = xTime + x2;
                        y_ = yTime + y2;
                        color_ = 3'b000;
                    end

                    Y=Compare;
                    //Y=Time;
                end

Won:            begin

                    x_ = xWon;
                    y_ = yWon;
                    color_ = colorWon;

                    w = 1;
                    l=0;


                    Y=Won;

                end

Lose:           begin

                    x_ = xLose;
                    y_ = yLose;
                    color_ = colorLose;
                    l=1;
```

```
                    w=0;

                    Y=Lose;

                end

    Level:      begin

                    if(controlTimer==7)
                    begin
                        level=1;
                        Y=Won;
                    end


                    else if(controlTimer==1)
                    begin
                        level=3;
                        Y=Green;
                    end

                    else if(controlTimer==3)
                    begin
                        level=7;
                        Y=Green;
                    end

                    else
                        Y=Level;



                end


    default: ;
    endcase
end
```

```
   always@(posedge CLOCK_50)
   begin

         if(resetn==0)
         begin

               //nReg<=nReg+1;
               y<=Resetn;

         end

         else
         begin

               y<=Y;

         end

   end

   //Reset inst1(.clk(CLOCK_50), .x1(x1), .y1(y1), .y(y), .color1(color1));
   Green inst1(.clk(CLOCK_50), .x1(x1), .y1(y1), .color1(color1), .y(y),
.greenDone(greenDone), .level(lev));

   DrawEraseNModify inst2(.clk(CLOCK_50), .countX(x2), .countY(y2), .y(y),
.drawn(drawn), .ct(ct), .color(colorC), .xC(xC), .yC(yC), .resetn(resetn),
   .enCursor(enCursor), .xCursor(xCursor), .yCursor(yCursor), .colorCursor(colorCursor),
.left(left), .right(right), .up(up), .down(down), .shoot(shoot),
   .won(won), .xC1(xC1), .yC1(yC1), .xC2(xC2), .yC2(yC2), .xCursorDraw(xCursorDraw),
.yCursorDraw(yCursorDraw),
   .xTime(xTime), .yTime(yTime), .n1(nR), .timerSpeed(timerSpeed));

   assignKeys inst3(.ps2_key_data(ps2_key_data), .left(left), .right(right), .up(up),
.down(down), .shoot(shoot), .resetn(resetn), .y(y));
```

```
    // won ROM
    getAddress inst4(.x(xWon), .y(yWon), .address(address));

    Won inst5(.clk(CLOCK_50), .xWon(xWon), .yWon(yWon), .y(y));

    ROM inst6(.address(address), .clock(CLOCK_50), .q(colorWon));

    // start ROM
    getAddress inst7(.x(xStart), .y(yStart), .address(addressStart));

    Start inst8(.clk(CLOCK_50), .xStart(xStart), .yStart(yStart), .y(y));

    StROM inst9(.address(addressStart), .clock(CLOCK_50), .q(colorStart));

    // lose ROM
    getAddress inst10(.x(xLose), .y(yLose), .address(addressLose));

    Lose inst11(.clk(CLOCK_50), .xLose(xLose), .yLose(yLose), .y(y));

    loseROM inst12(.address(addressLose), .clock(CLOCK_50), .q(colorLose));


/*////////////////////////////////////

PS2 INSTANTIATION

*////////////////////////////////////
    PS2_Controller PS2 (
    // Inputs
    .CLOCK_50                   (CLOCK_50),
    .reset                      (~KEY[0]),

    // Bidirectionals
    .PS2_CLK                    (PS2_CLK),
          .PS2_DAT              (PS2_DAT),

    // Outputs
    .received_data      (ps2_key_data),
    .received_data_en    (ps2_key_pressed)
```

```
);


////////////////////////////////////////////

    // Create an Instance of a VGA controller - there can be only one!
    // Define the number of colours as well as the initial background
    // image file (.MIF) for the controller.
    vga_adapter VGA(
                .resetn(KEY[0]),
                .clock(CLOCK_50),
                .colour(color_VGA),
                .x(x_VGA),
                .y(y_VGA),
                .plot(1),
                /* 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 = "";

    // Put your code here. Your code should produce signals x,y,color and writeEn
    // for the VGA controller, in addition to any other functionality your design may require.
endmodule

module Green(clk,x1,y1,color1,y,greenDone,level);
    input clk;
    output reg [7:0]x1;
    output reg [6:0]y1;
    output reg[2:0]color1;
    input[2:0]y;
```

```
   output reg greenDone;
   reg d;
   input[2:0]level;

   always@(posedge clk)
   if(y==3'b100)                                                    // y==Green state
   begin
         if(~d)
         begin

                  // timer text
                  if(
                  (y1==1 & (x1==7 | x1==8 | x1==9 | x1==12 | x1==15 | x1==16 | x1==17 |
x1==18 | x1==19 | x1==22 | x1==23 |
                  x1==24 | x1==27 | x1==28 | x1==29 | x1==102 | x1==107 | x1==108 |
x1==109 | x1==112 | x1==116 | x1==119 | x1==120 |
                  x1==121 | x1==124 | (x1==129 &(level==3 | level==7)) | x1==130 | (x1==131
& (level==3 | level==7))  )) |

                  (y1==2 & (x1==8 | x1==12 | x1==15 | x1==17 | x1==19 | x1==22 | x1==27 |
x1==29 | x1==102 | x1==107 | x1==112 |
                  x1==116 | x1==119 | x1==124 | (x1==130 & level==1) | (x1==131 & (level==3
| level==7))    ))|

                  (y1==3 & (x1==8 | x1==12 | x1==15 | x1==17 | x1==19 | x1==22 | x1==23 |
x1==24 | x1==27 | x1==28 | x1==29 |
                  x1==102 | x1==107 | x1==108 | x1==109 | x1==112 | x1==116 | x1==119 |
x1==120 | x1==121 | x1==124 |
                  (x1==129 & (level==3 | level==7)) | x1==130 | (x1==131 & (level==3 |
level==7))      )) |

                  (y1==4 & (x1==8 | x1==12 | x1==15 | x1==17 | x1==19 | x1==22 | x1==27 |
x1==28 | x1==102 | x1==107 | x1==113 |
                  x1==115 | x1==119 | x1==124 | (x1==129 & level==3) | (x1==130 & level==1)
| (x1==131 & level==7)       ))    |

                  (y1==5 & (x1==8 | x1==12 | x1==15 | x1==17 | x1==19 | x1==22 | x1==23 |
x1==24 | x1==27 | x1==29 |
                  x1==102 | x1==103 | x1==104 | x1==107 | x1==108 | x1==109 | x1==114 |
```

```
x1==119 | x1==120 | x1==121 | x1==124 | x1==125 |
		x1==126 | (x1==129 & (level==3 | level==7)) | x1==130 | (x1==131 &
(level==3 | level==7))	))

		)
		begin

			color1<=3'b110;

			if(x1==160)
			begin
				x1<=0;
				y1<=y1+1;
			end
			else
				x1<=x1+1;

		end

		// divider
		else if(y1==17 & x1!=160)
		begin

			color1<=3'b010;

			if(x1==160)
			begin
				x1<=0;
				y1<=y1+1;
			end
			else
				x1<=x1+1;

		end


		// timer bar
		else if((y1==10 & x1==160) | (y1>10 & y1<15) | (y1==15 & x1<160))
		begin
```

```
                color1<=3'b100;

                if(x1==160)
                begin
                        x1<=0;
                        y1<=y1+1;
                end
                else
                        x1<=x1+1;
        end

        // background
        else
        begin
                color1<=3'b000;
                if(x1==160)
                begin
                        x1<=0;

                        if(y1==120)
                        begin
                                y1<= 0;
                                d<=1;
                        end
                        else
                                y1<=y1+1;
                end
                else
                begin
                        x1<=x1+1;
                end
        end



end
```

```
        else
        begin
                greenDone<=1;
        end

    end

    else
    begin
        d<=0;
        greenDone<=0;
    end
endmodule


module Won(clk, xWon, yWon, y);

    input clk;
    output reg[7:0]xWon;
    output reg[6:0]yWon;
    input[2:0]y;

    always@(posedge clk)
    begin

        if(y==3'b011)
        begin
                if(xWon == 160)
                begin
                        xWon<=0;
                        if(yWon == 120)
                                yWon<=0;
                        else
                                yWon<=yWon+1;
                end

                else
                        xWon<=xWon+1;
        end
```

```
    end

endmodule


module Start(clk, xStart, yStart, y);

    input clk;
    output reg[7:0]xStart;
    output reg[6:0]yStart;
    input[2:0]y;

    always@(posedge clk)
    begin

        if(y==3'b000)
        begin
                if(xStart == 160)
                begin
                        xStart<=0;
                        if(yStart == 120)
                                yStart<=0;
                        else
                                yStart<=yStart+1;
                end

                else
                        xStart<=xStart+1;
        end
    end

endmodule

module Lose(clk, xLose, yLose, y);

    input clk;
    output reg[7:0]xLose;
    output reg[6:0]yLose;
    input[2:0]y;
```

```
    reg[26:0]cycle_count;

    always@(posedge clk)
    begin

        if(y==3'b101)
        begin

            //if(cycle_count==10000)
            //begin
                if(xLose== 160)
                begin
                    xLose<=0;
                    if(yLose== 120)
                        yLose<=0;
                    else
                        yLose<=yLose+1;
                end

                else
                    xLose<=xLose+1;

                //cycle_count<=0;
            //end
            //else
                //cycle_count<=cycle_count+1;
        end
    end

endmodule




module getAddress(x, y, address);

    input[7:0]x;
    input[6:0]y;
    output reg[14:0]address;
```

```
    always@(*)
    begin

        address = (y*160)+x;

    end

endmodule


module DrawEraseNModify(clk, countX, countY, y, drawn, ct, color, xC, yC, resetn,
enCursor, xCursor, yCursor, colorCursor,
left, right, up, down, shoot, won, xC1, yC1, xC2, yC2, xCursorDraw, yCursorDraw, xTime,
yTime, n1, timerSpeed);
    input clk;
    output reg [2:0]countX;
    output reg [2:0]countY;
    input[2:0]y;
    output reg drawn;
    output reg[2:0]ct;
    output reg[2:0]color;
    output reg[7:0]xC;
    output reg[6:0]yC;
    input resetn;
    output reg[1:0]enCursor;
    output reg[7:0]xCursor;
    output reg[6:0]yCursor;
    output reg[2:0]colorCursor;
    input left, right, up, down, shoot;
    output reg won;

    output reg[7:0]xC1;
    output reg[6:0]yC1;
    output reg[7:0]xC2;
    output reg[6:0]yC2;


    output reg[7:0]xCursorDraw;
```

```
output reg[6:0]yCursorDraw;

output reg[7:0]xTime;
output reg[6:0]yTime;

input[2:0]n1;

input[27:0]timerSpeed;

reg[27:0]cursorSpeed;

reg[7:0]xCursorErase;
reg[6:0]yCursorErase;

//input[2:0]number;

reg[2:0]number;



reg[25:0]cycle_count;
reg[25:0]delay;
reg d;
reg erased;
reg[2:0]s;
reg finished;
reg colorCount;

reg[2:0]num;
reg[2:0]numCount;

reg cursorDrawn;

reg timeReduced;

reg shot;

parameter Draw5by5=3'b000, DelayPattern=3'b001,
```

```
EraseNModifyCoordinates=3'b010, changeNumber=3'b011, reduceTime=3'b100;


    always@(posedge clk)
    begin

    xC1<=xC-1;
    yC1<=yC-1;
    xC2<=xC+5;
    yC2<=yC+5;

    //if(~resetn)
    if(y==3'b100)
    begin
            xC<=25;
            yC<=25;
            drawn<=0;
            countX<=0;
            countY<=0;
            cycle_count<=0;
            delay<=0;
            d<=0;
            erased<=0;
            s<=0;
            ct<=0;
            finished<=0;
            number<=n1;
            colorCount<=0;
            numCount<=3'b000;
            enCursor<=0;
            xCursor<=80;
            yCursor<=60;
            xCursorDraw<=80;            // 80
            yCursorDraw<=60;                        // 60
            cursorDrawn<=0;
            won<=0;
            xC1<=xC-1;
            yC1<=yC-1;
            xC2<=xC+5;
```

```
        yC2<=yC+5;
        xTime<=0;
        yTime<=11;
        shot<=0;
end



        else if(y == 3'b001)
        begin

                if(timerSpeed==50000000)
                        cursorSpeed<=750000;

                else if(timerSpeed==25000000)
                        cursorSpeed<=500000;

                else if(timerSpeed==10000000)
                        cursorSpeed<=250000;

                if(s==Draw5by5 & ~finished)
                begin
                        if(~d)
                        begin


                                delay<=0;
                                erased<=0;

                                if(colorCount==0)
                                        color<=3'b011;

                                else
                                        color<=3'b001;

                                if(countX==4)
                                begin
                                        countX<=0;
```

```
                        if(countY==4)
                        begin

                                countY <= 0;

                                //drawn<=1;
                                d<=1;

                        end
                        else
                                countY<=countY+1;
                end
                else
                        countX<=countX+1;
        end

        else
        begin


                ct<=ct+1;
                s<=DelayPattern;
                enCursor<=1;
        end
end

// delay state
else if(s==DelayPattern)
begin

        timeReduced<=0;

        if(cycle_count==cursorSpeed)
        begin

                // draw cursor
                if(~cursorDrawn)
                begin
```

```
xCursorErase<=xCursorDraw;
yCursorErase<=yCursorDraw;

xCursor<=xCursorDraw;
yCursor<=yCursorDraw;
colorCursor<=3'b110;



if(left)
begin
	if(xCursorDraw!=8'b00000000)
	begin
		xCursorDraw<=xCursorDraw-1;
		cursorDrawn<=1;
		shot<=0;
	end
end
else if(right)
begin
	if(xCursorDraw!=8'b10011111)
	begin
		xCursorDraw<=xCursorDraw+1;
		cursorDrawn<=1;
		shot<=0;
	end
end
else if(up)
begin
	if(yCursorDraw!=7'b0010100)
	begin
		yCursorDraw<=yCursorDraw-1;
		cursorDrawn<=1;
		shot<=0;
	end
end
else if(down)
begin
	if(yCursorDraw!=7'b1110111)
```

```
                        begin
                                yCursorDraw<=yCursorDraw+1;
                                cursorDrawn<=1;
                                shot<=0;
                        end
                end

                else if(shoot)
                begin
                        colorCursor<=3'b010;
                        shot<=1;
                end


        end     //      if(~cursorDrawn).....ends

        // Erase
        else
        begin

                xCursor<=xCursorErase;
                yCursor<=yCursorErase;
                colorCursor<=3'b000;

                cursorDrawn<=0;

        end


        cycle_count<=0;
end     // if(cycle_count==250000)  .... ends

else
                cycle_count<=cycle_count+1;



if(delay==timerSpeed)
begin
```

```
                //s<=EraseNModifyCoordinates;
                s<=reduceTime;
                enCursor<=2;
                //enCursor<=0;
        end

        else
        begin
                delay<=delay+1;

        end    //  else ( delay != 25000000)....ends


end

else if(s==reduceTime)
begin

        if(~timeReduced)
        begin
                if(countX==4)
                begin
                        countX<=0;

                        if(countY==4)
                        begin

                                countY <= 0;
                                timeReduced<=1;

                        end
                        else
                                countY<=countY+1;
                end
                else
                        countX<=countX+1;
        end

        else
```

```
        begin
                if(xTime!=160)
                        xTime<=xTime+5;
                s<=EraseNModifyCoordinates;
                enCursor<=0;

        end


end


// Erase and modify coordinates
else if(s==EraseNModifyCoordinates)
begin


        if(~erased)
        begin
                delay<=0;
                d<=0;

                color<=3'b000;
                if(countX==4)
                begin
                        countX<=0;

                        if(countY==4)
                        begin
                                countY <= 0;
                                //drawn<=0;
                                erased<=1;
                                //xC<=xC+10;
                        end
                        else
                        begin
                                countY<=countY+1;
                        end
                end
```

```
        else
                countX<=countX+1;
end

else
begin
        num<=number;

        // number = 3
        if(number==3'b011 | number==3'b111)
        begin

                if(ct==3'b001)
                begin
                        xC<=xC+15;
                        yC<=yC+15;
                end

                else if(ct==3'b010)
                begin
                        xC<=xC+15;
                        yC<=yC+15;
                end

                else if(ct==3'b011)
                begin
                        xC<=xC-30;
                        yC<=yC-30;
                        ct<=0;
                        finished<=0;
                        if(numCount==7)
                                numCount<=numCount+2;
                        else
                                numCount<=numCount+1;
                end

        end

        // number = 4
```

```
else if(number==3'b100 | number==3'b010)
begin

        if(ct==3'b001)
                xC<=xC+30;

        else if(ct==3'b010)
                yC<=yC+30;

        else if(ct==3'b011)
                xC<=xC-30;

        else if(ct==3'b100)    // pattern for 4 has finished
        begin
                yC<=yC-30;
                ct<=0;
                finished<=0;
                if(numCount==7)
                        numCount<=numCount+2;
                else
                        numCount<=numCount+1;
        end

end

// number = 5
else if(number==3'b101 | number==3'b001)
begin

        if(ct==3'b001)
                xC<=xC+30;

        else if(ct==3'b010)
                yC<=yC+30;

        else if(ct==3'b011)
                xC<=xC-30;

        else if(ct==3'b100)
```

```
            begin
                xC<=xC+15;
                yC<=yC-15;
            end


            else if(ct==3'b101)
            begin
                xC<=xC-15;
                yC<=yC-15;
                ct<=0;
                finished<=0;
                if(numCount==7)
                    numCount<=numCount+2;
                else
                    numCount<=numCount+1;
            end

    end

    // number = 6
    else if(number==3'b110 | number==3'b000)
    begin

            if(ct==3'b001)
                yC<=yC+15;

            else if(ct==3'b010)
                yC<=yC+15;

            else if(ct==3'b011)
                xC<=xC+20;

            else if(ct==3'b100)
                yC<=yC-15;

            else if(ct==3'b101)
                yC<=yC-15;
```

```
                else if(ct==3'b110)
                begin
                        xC<=xC-20;
                        ct<=0;
                        finished<=0;
                        if(numCount==7)
                                numCount<=numCount+2;
                        else
                                numCount<=numCount+1;
                end

        end



        //s<=3'b000;
        s<=changeNumber;
    end

end




// change number
else if(s==changeNumber)
begin

    if(ct==0)
    begin


        number[2]<=(num[2] ^ num[0]);
        number[1]<=num[2];
        number[0]<=num[1];
        colorCount<=~colorCount;
```

```
        if(numCount==1)
        begin
                xC<=xC+30;
        end

        else if(numCount==2)
        begin
                xC<=xC+30;
        end

        else if(numCount==3)
        begin
                xC<=xC-60;
                yC<=yC+50;
        end

        else if(numCount==4)
        begin
                xC<=xC+30;
        end

        else if(numCount==5)
        begin
                xC<=xC+30;
        end

        else if(numCount==6)
        begin
                xC<=xC-30;
                yC<=yC-25;
        end

        else if(numCount==7)
        begin
                xC<=25;
                yC<=25;
        end

end
```

```
                    s<=Draw5by5;
                end



        end



    end
endmodule



module assignKeys(ps2_key_data, left, right, up, down, shoot, resetn, y);

    input[7:0]ps2_key_data;
    output reg left, right, up, down, shoot;
    input resetn;
    input[2:0]y;


    always@(*)
    begin

    //if(~resetn)
    if(y==3'b100)
    begin
        left=0;
        right=0;
        up=0;
        down=0;
        shoot=0;
    end

    else if(ps2_key_data== 8'b01101011)
    begin
```

```
        left=1;
        right=0;
        up=0;
        down=0;
        shoot=0;
end


else if(ps2_key_data==8'b01110100)
begin
        right=1;
        left=0;
        up=0;
        down=0;
        shoot=0;
end

else if(ps2_key_data==8'b01110101)
begin
        up=1;
        left=0;
        right=0;
        down=0;
        shoot=0;
end

else if(ps2_key_data==8'b01110010)
begin
        down=1;
        left=0;
        right=0;
        up=0;
        shoot=0;
end

else if(ps2_key_data==8'b00101001)
begin
        down=0;
        left=0;
```

```
        right=0;
        up=0;
        shoot=1;
    end

    else
    begin
        left=0;
        right=0;
        up=0;
        down=0;
        shoot=0;
    end

end

endmodule
```