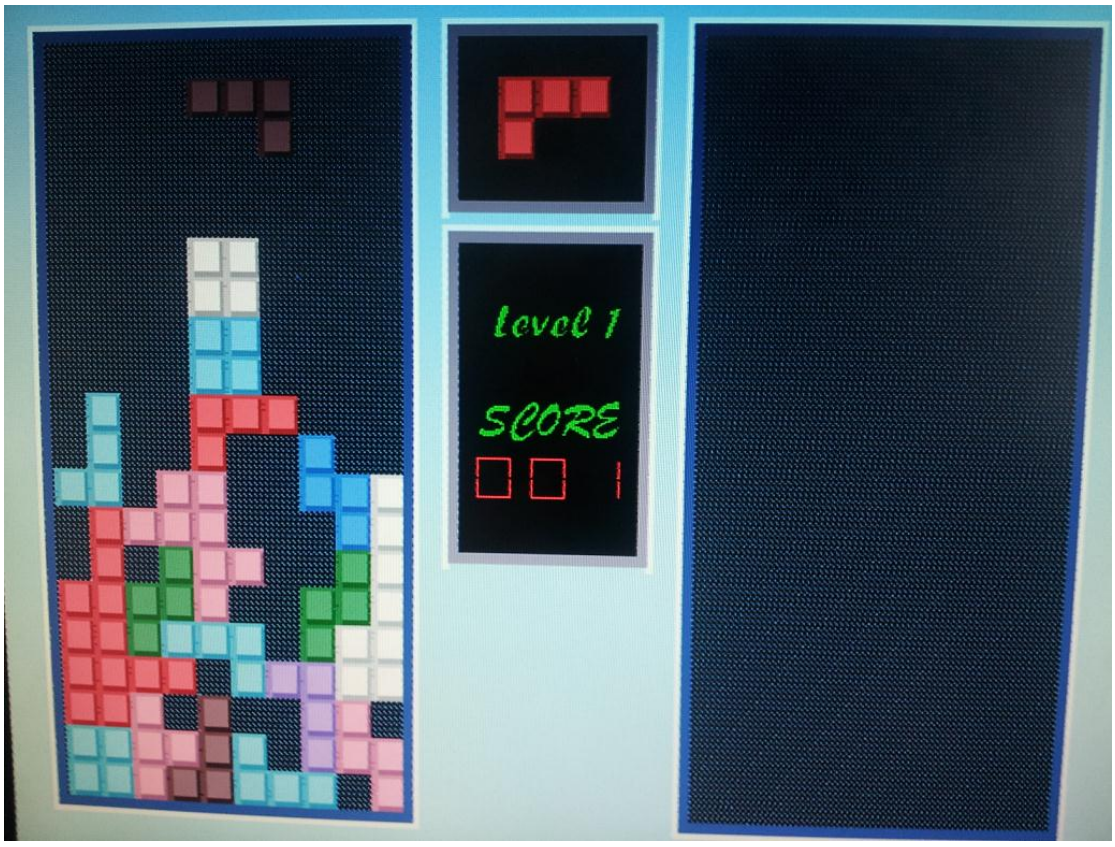
Tetris Game

CSEE4840 Embedded System
Project Report



Yunfan Dong,yd2238

Directory

Introduction

Tetris (Russian: Тетрис) is a tile-matching puzzle video game originally designed and programmed by Alexey Pajitnov in the Soviet Union. It was released on June 6, 1984, while he was working for the Dorodnicyn Computing Centre of the Academy of Science of the USSR in Moscow. He derived its name from the Greek numerical prefix tetra- (all of the game's pieces contain four segments) and tennis, Pajitnov's favorite sport.

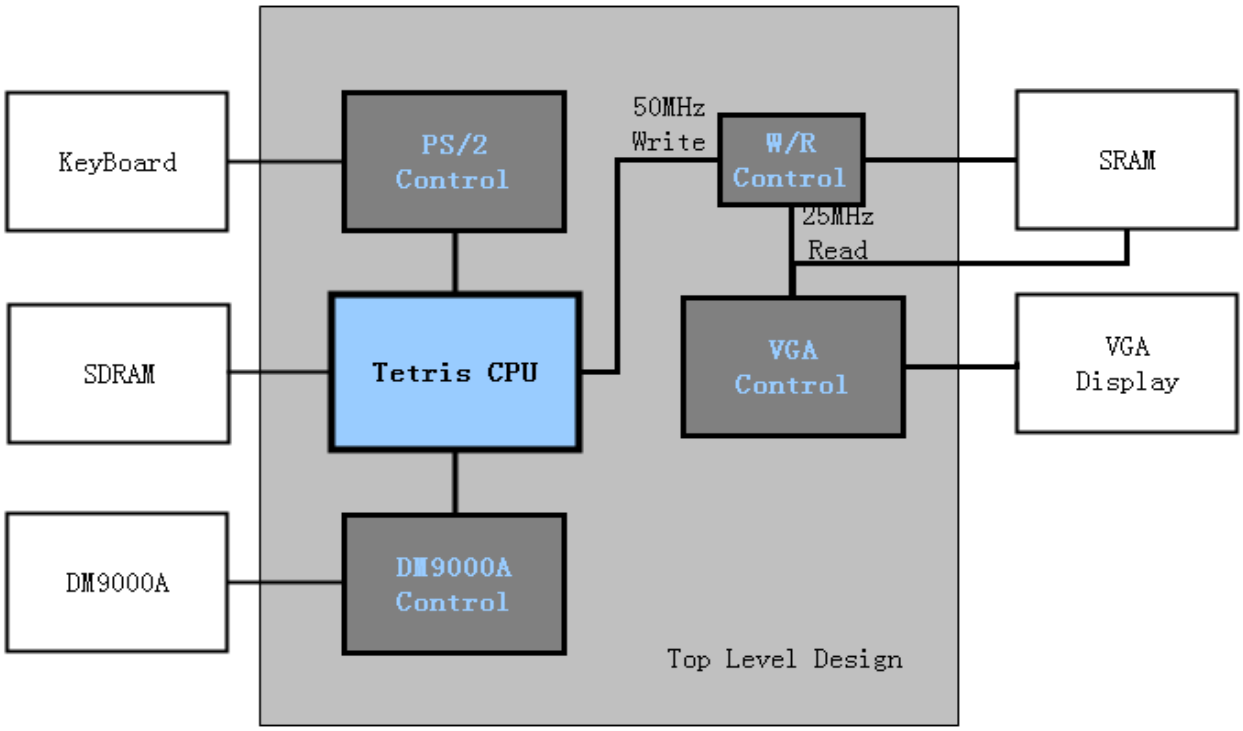
It is also the first entertainment software to be exported from the USSR to the U.S. and published by Spectrum Holobyte for Commodore 64 and IBM PC. TheTetris game is a popular use of tetrominoes, the four

element special case of polyominoes. Polyominoes have been used in popular puzzles since at least 1907, and the name was given by the mathematician Solomon W. Golomb in 1953. However, even the enumeration of pentominoes is dated to antiquity.

In this project , I implemented a classic tetris game. During the game the player will deal with 7 different kinds of falling blocks with random colors. Each block contains four tiles. The aim is to eliminate as many blocks as possible, and more importantly , to live on. Because as score grow, tiles fall faster and faster, the player would probably find out that even to live on is a hard task.

I have also introduced ethernet connections into this game ,so that two players can fight against each other in two separate running game clients and sync between each other.

I. System Structure



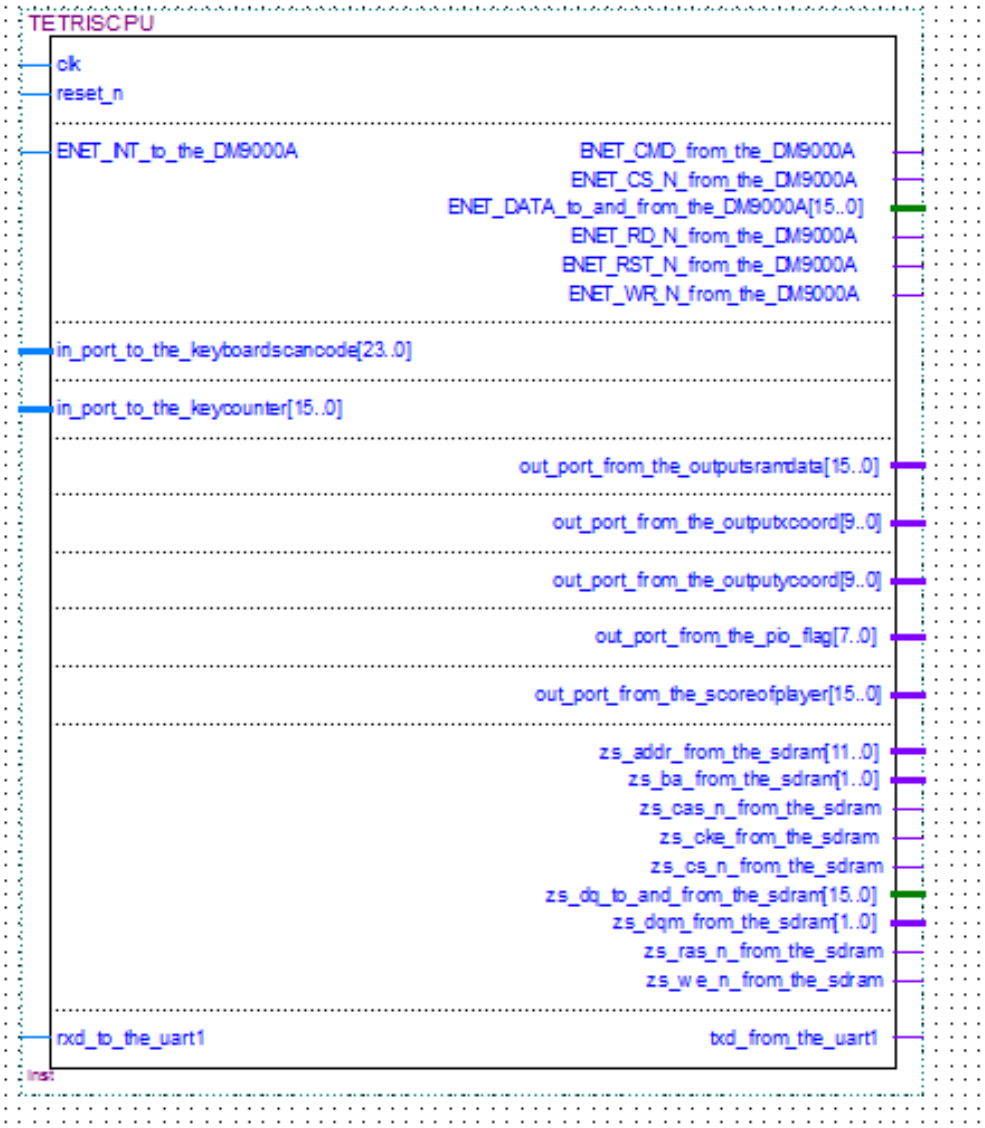
For a game like Tetris, the screen display dynamically changes. As game goes on, blocks can be created as a whole and fall down as a whole but parts of them might get eliminated. It means that lots of block data will be written dynamically into SRAM and then output to VGA display.

The clock rate of CPU and VGA are different. So there will be a control module to monitor the read and write of SRAM.

Based on these considerations, the system structure is designed as shown in the picture.

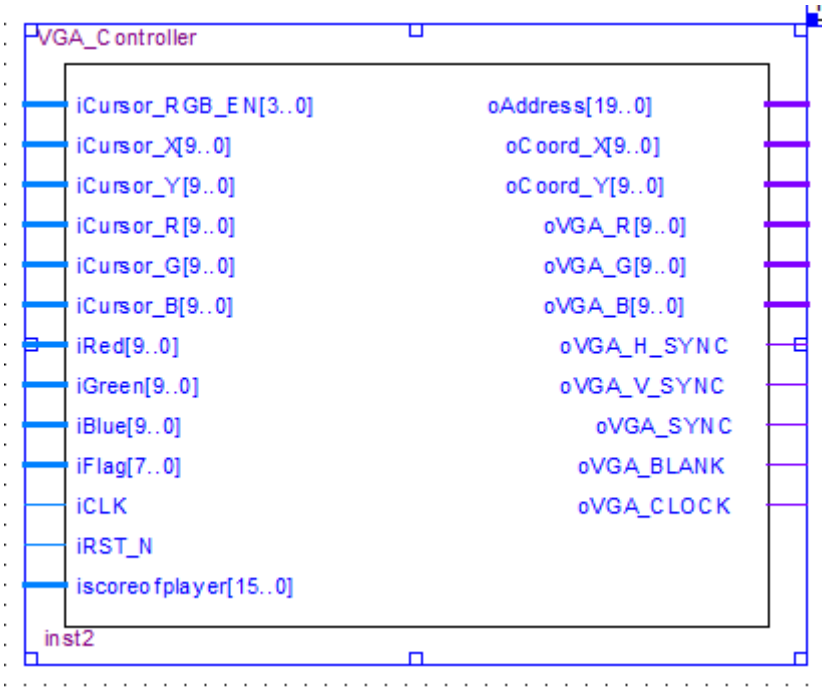
The top-level entity of the system contains :

Tetris CPU



Tetris CPU is the hardware-software interface of the game. The CPU contains interface to DM9000A, PS/2 Controller, SDRAM, and W/R Control Logic.

VGA control

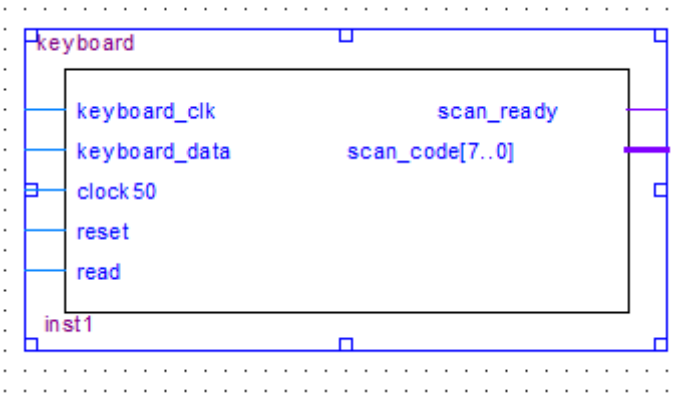


reads image data from SRAM and output them into VGA Display.

W/R Control

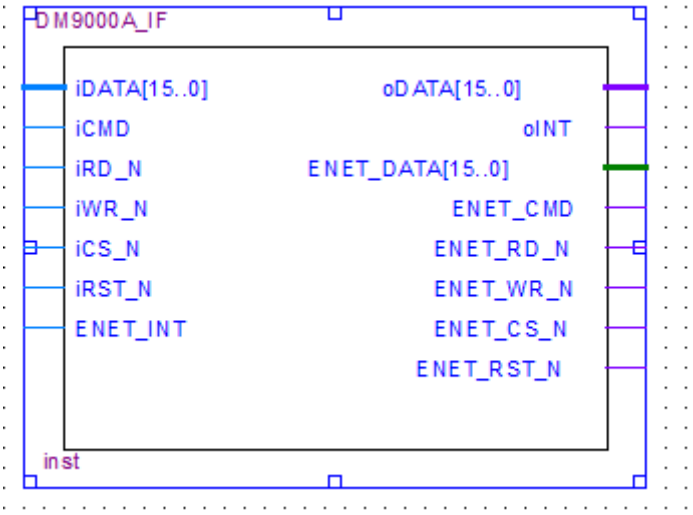
W/R control is to control the read and write process of SRAM. Because VGA Display works at 25MHz and CPU works at 50MHz, there is always a switch between VGA cycle and CPU cycle. The W/R control makes sure that during VGA cycle, the SRAM reads. And other times ,with rising signal of CPU clock, the SRAM writes. W/R control also makes sure that address switch between VGA cycle and CPU cycle.

PS/2 control



PS/2 is the interface with Keyboard.

DM9000A



DM9000A control is responsible for DM9000A send and receive packages.

II. Image Data and Storage use

Sum Up

Image Name	Number of Images	Size(KB)
Background	**1**	**37.50**
Name	**1**	**0.86**
Title	**1**	**0.73**
Columbia	**1**	**8.98**
Score	**1**	**0.33**
Score Digit	**3**	**4.39**
Level Image	**3**	**0.88**
Ready Image	**1**	**0.73**
Tiles	**0 to 200 (single player)**	**0 to 117.18**
Battle Area	**2**	**146.48**
Next Piece Area	**1**	**18.62**
Score Area	**1**	**35.73**
Total	**16 to 216**	**372.41**

In this game , images include static part and dynamic part.

The static part contains welcome page images , “score” characters ,level images indicating game level, and Battle Area background.

Dynamic part contains Tiles, Next Piece Area and Score digits.

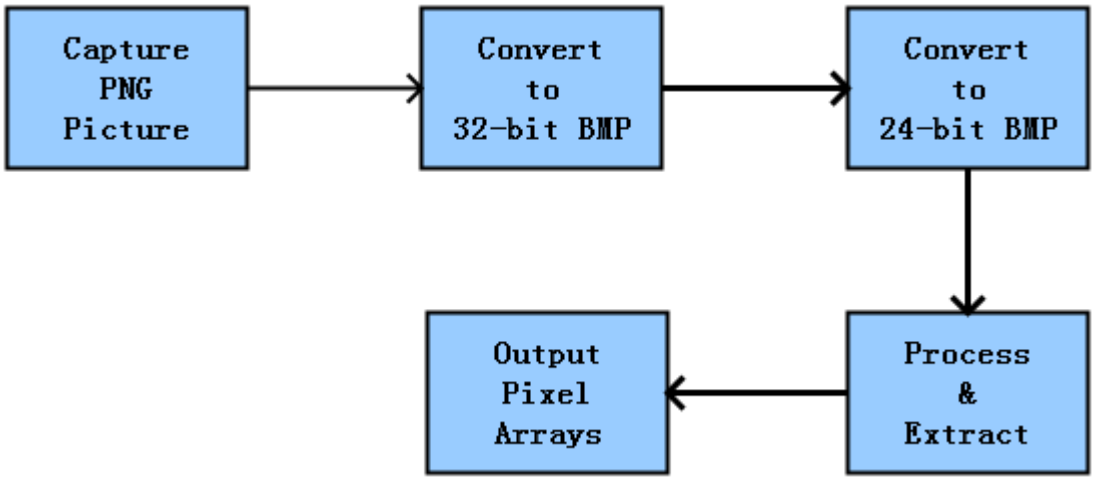
All image data are stored in SRAM on chip (size 512 KB).

The usage of SRAM for this game is 72.65% in Single Player mode.

Under Double Player mode, the Tiles data would double. Total size is 489.59 KB and usage is 95.62%.

How images are Stored and Displayed

All image data are stored as arrays either of binary or hex numbers. Arrays are extracted through a process shown below:



The **Process & Extract** part is implemented with VC++ 6.0 ,by reading RGB value of every pixel and do corresponding calculations. According to different pictures, the calculations are different.

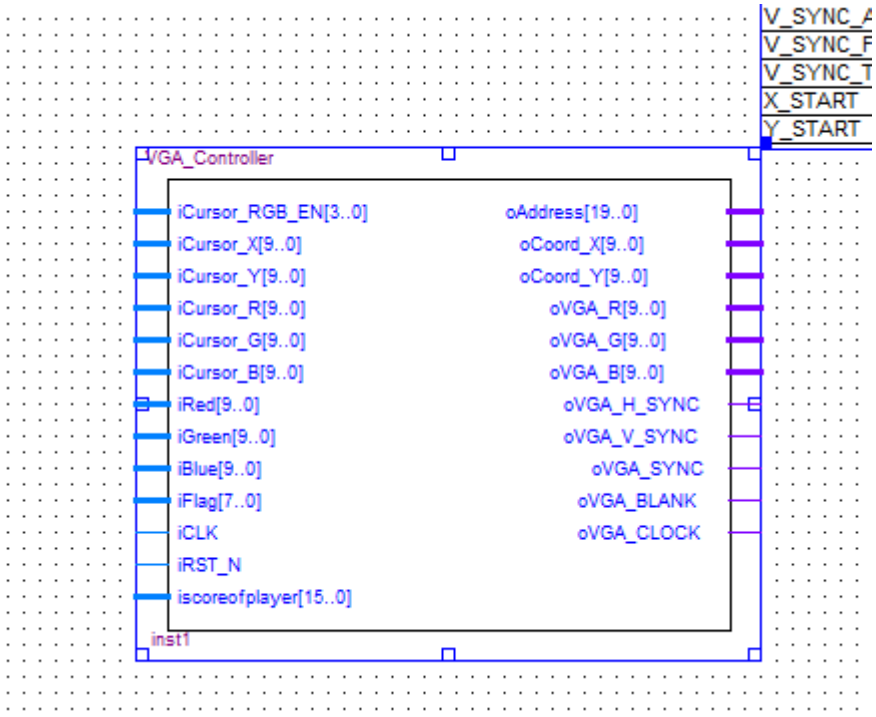
For example, the following code is how I extract pixels from a image of characters.

```
//**************Extract Pixels with Grayscale < 50 (closer to black) **************
void CFaceDetectView::Mypic()
{
	LPBYTE lpData;
	long lOffset;
	FILE *p1,*p2,*p3,*p4;

	lpData = gDib.FindDIBBits(hDIB);
	WORD gwBytesPerLine = gDib.BytePerLine(hDIB);

	unsigned ColorR,ColorG,ColorB;
	double gray;
	int gray_int;
	Gray= new int*[v_top-v_bottom+1];

	LPBITMAPINFOHEADER lpbi;
	int width,height;
	lpbi = (LPBITMAPINFOHEADER)GlobalLock(hDIB);
                                    //width and height
	width = lpbi->biWidth;
	height = lpbi->biHeight;

	if((p1 = fopen("g://out1.txt","w+"))==NULL)
		::AfxMessageBox("Wrong!");

	int k;

	for( k=0;k<v_top-v_bottom+1;k++)
	   Gray[k]=new int [v_right-v_left+1];

	 int k1,i,j;
	 CString s1,s2;
	 s1.Format("%d",width);
	 s2.Format("%d",height);

	 CString para = "width:"+s1+"; height:"+s2;
	 ::AfxMessageBox(para);
	 rewind(p1);
	for(i=height-1;i>=0;i--)
	{
```

```
            fprintf(p1,"score[%3d] = %d'b",height-1-i,width);
            for(j=0;j<=width-1; j++)
            {
                lOffset = gDib.PixelOffset(i, j, gwBytesPerLine);
                ColorB=*(lpData + lOffset);
                ColorG=*(lpData + lOffset+1);
                ColorR=*(lpData + lOffset+2);

                gray = (ColorG*50+ColorR*39+ColorB*11)/100;
                gray_int = gray;

                if(gray<50) fputc('1',p1);
                else fputc('0',p1);

                if(j==width -1) { fputc(';',p1); fputc('\n',p1); }
            }
        }
        fclose(p1);
}
```

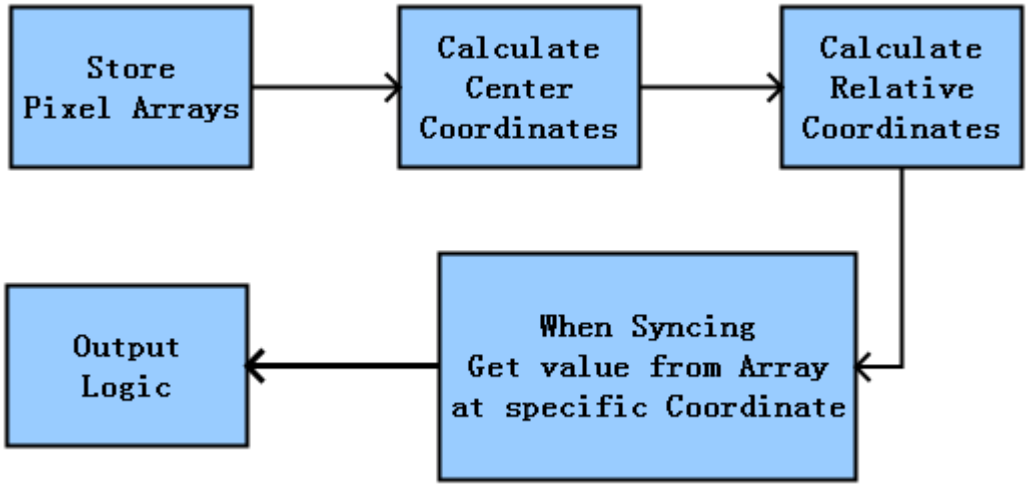
When arrays are generated, the data will then be processed in VGA controller to show the images.

III. VGA Control and Display



Basically, VGA display is controlled by the VGA Controller, which controls the signals input into the VGA display. Most of images are pre-stored in the VGA Controller, the positions and the presence of pictures are controlled by the input X, Y, FLAG and score signals which comes from CPU. In order to organize the showing sequence of images, a flag of 8 bits is introduced.

The following process shows how each image is controlled to appear in the right place of screen:



Arrays

Following is the array of the “score” characters. The array consists of binary numbers.The array is generated by the c++ code shown before.

```
score[  0]= 80'b00000000000000000000000000000000000000000000000000000000000000000000000000000000;
score[  1] = 80'b00000000000000000000000000000000000000000000000000000000000000000000000000000000;
score[  2] = 80'b00000000000000000000000000000000000000000000000000000000000000000000000000011100;
score[  3] = 80'b00000000000000000000000000000011100000000001000000000000000000000000000111111110;
score[  4] = 80'b00000000000000000000000000001111110000011111000000001111111100000011111111111;
score[  5] = 80'b00000000011111111110000000011100110000011111100000011111111111110000111111111110;
score[  6] = 80'b00000001111111111110000000111000110000111111100001111111111111110011110000000000;
score[  7] = 80'b00000011111111111110000000110000011000111111111001111111110000011101111000000000;
score[  8] = 80'b00000111100000010000000011000001100011111000110011100001100000011011000000000000;
score[  9] = 80'b00000110000000000000000111000011100111100000110000000011000000011011100000000000;
score[ 10] = 80'b00000110000000000000000111000011100011100000011000000001000000111000111111111000;
score[ 11] = 80'b00000111000000000000011100011111000111000000010000000010001111110000000011110000;
score[ 12] = 80'b00000011111110000000001100011110000110000000110000000011111111100000000001111000000;
score[ 13] = 80'b00000000011111100000011100011100001110000000110000011111100000000000011100000000;
score[ 14] = 80'b00000000000001110000011000111000001100000011000000111110000000000001110000000000;
score[ 15] = 80'b00000000000000010000111000000000001100000011000000111111110000000110000000000100;
score[ 16] = 80'b00000000000000110000111000000000001100000111000000110111111100000110000000011000;
score[ 17] = 80'b00110000001111100001100000000001001000011100000011000011111110001100000001111000;
score[ 18] = 80'b01111111111111000011100000000110001111111000000100000001111110011110011111110000;
score[ 19] = 80'b01111111111110000001110000001100000111100000000110000000001111001111111111100000;
score[ 20] = 80'b00011111110000000001111000111000000000000000000000000000000001100011111100000000;
score[ 21] = 80'b00000000000000000001111111100000000000000000000000000000000000000000000000000000;
score[ 22] = 80'b00000000000000000000111111000000000000000000000000000000000000000000000000000000;
score[ 23] = 80'b00000000000000000000000000000000000000000000000000000000000000000000000000000000;
```

Coordinates

In order to output the image in the right place, we have two sets of coordinates: center point of image, and relative coordinates in the image.

For example, for the score characters, the coordinate for the center point of image is (320,230) . Relative coordinates are calculated using the VGA Sync Counts and center point. The aim of relative coordinates is to set up a mapping between VGA Sync Counts and positions in the array. score_V and score_H are calculated as follows:

```
//**********************'score' characters********
score_V = (V_Cont - Y_START - 230 + 12) % 23;
score_H = (H_Cont - X_START - 320 + 40) % 79;
```

Output Logic

Now that we have the arrays and the coordinates, we can manage image display.

First , consider what stages of game should the image be shown. As for the “score” characters image, it should be shown after the game starts, rather than in the welcome page. We use the first bit of iFlag to judge the conditions.

Then, determine the scope of the image to be shown. In our example, the scope of “score characters” is (320-40,320+40) for Horizontal Coordinate and (230-12,230+12) for Vertical Coordinate. 320 and 230 is the center point of image , 40 is the half width of image, and 12 is the half height of image.

Finally, there is another import factor that we must consider: When inputs from software conflicts with what we have already customed in hardware, who takes the priority?

The answer is always **Software First**. Because basically we only change displays when we need. So what comes from software is always necessary.

So, in this case, we consider whether there are RGB inputs or not, then output the default logic.

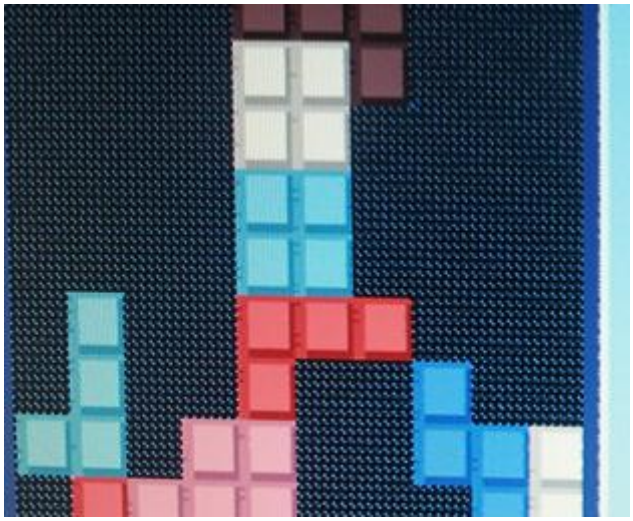
Codes are shown below.

```
//**********************'score' characters Output Logic********

else if( (iFlag[0]==1) && (H_Cont-X_START<320+40) && (H_Cont-X_START>320-40) && (V_Cont-Y_START<230+12) &&(V_Cont-Y_START > 230-12) && (score[score_V][80-score_H] == 1) )

                begin
                    if(!iRed && !iGreen && !iBlue)
                        begin
                            Cur_Color_R     <=  0;
                            Cur_Color_G     <=  800;
                            Cur_Color_B     <=  0;
                          end
                     else
                        begin
                          Cur_Color_R <=  iRed;
                          Cur_Color_G <=  iGreen;
                         Cur_Color_B  <=  iBlue;
                          end
                end
```

Detailed Distinctive Sprites

Battlefield Background

A half-translucent effect is implemented on the battle field shown as below .



The way to implement this effect is to let 1/3 of the pixels in the area display battle field background(dark) and 2/3 of the pixels in the area display the screen background(light blue).

Corresponding Codes:

```
if(!iRed && !iGreen && !iBlue
  &&((H_Cont-X_START)%3==0)&&((V_Cont-Y_START)%3==0))
      begin
        Cur_Color_R     <=   316;
        Cur_Color_G     <=   316;
        Cur_Color_B     <=   316;
      end
else if(!iRed && !iGreen && !iBlue
  &&((H_Cont-X_START)%3==1)&&((V_Cont-Y_START)%3==1))
      begin
        Cur_Color_R     <=   280;
        Cur_Color_G     <=   520;
        Cur_Color_B     <=   720;
      end
```

```
else if(!iRed && !iGreen && !iBlue
  &&((H_Cont-X_START)%3==2)&&((V_Cont-Y_START)%3==2))
      begin
        Cur_Color_R     <=   280;
        Cur_Color_G     <=   520;
        Cur_Color_B     <=   720;
      end
```

The Columbia Relief

The most difficult part of this relief is not to show it in the display, we have discussed the way to display pics before.

The difficult part is to extract exactly the pixels you need in a bitmap so that it looks like a relief. I did all this in my way.

In this picture , all the white pixels have grayscale value between 152 and 213. The threshold is by experience.

Digital Clock Numbers

Digital Numbers are like 7-segment hex on the DE2 board. Each Digit contains 7 segments.

Score is updated by software through PIO “iscoreofplayer”. The score is then translated into 7-segment digits and displayed out through Score Display logic in the VGA controller.

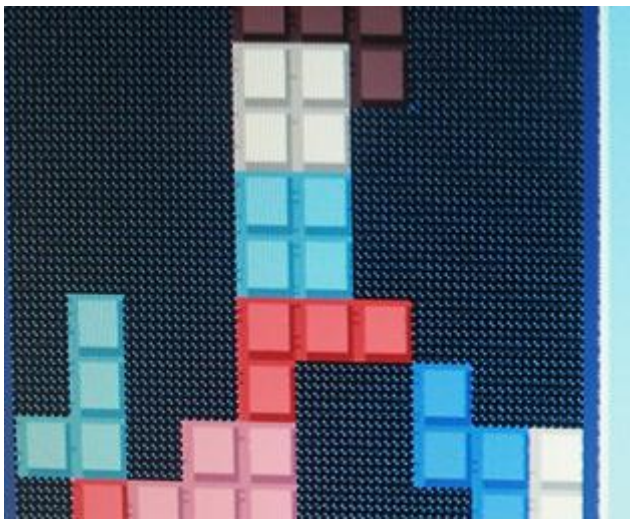
Below is how score is translated into 7-segment digits:

```
///**************hexout module***************
module hexout(
input wire [15:0] value ,
output reg    [6:0] outvalue
);

wire [3:0] tempvalue ;
assign tempvalue = value[3:0] ;
always @ (value)
  case(tempvalue)
       4'd0 : outvalue = 7'b0111111 ;
       4'd1 : outvalue = 7'b0000110 ;
       4'd2 : outvalue = 7'b1011011 ;
       4'd3 : outvalue = 7'b1001111 ;
       4'd4 : outvalue = 7'b1100110 ;
       4'd5 : outvalue = 7'b1101101 ;
       4'd6 : outvalue = 7'b1111101 ;
```

```
        4'd7 : outvalue = 7'b0000111 ;
        4'd8 : outvalue = 7'b1111111 ;
        4'd9 : outvalue = 7'b1100111 ;
        default : outvalue = 7'b0111111 ;
    endcase
endmodule
//*****************calculate*************
hexout u1(iscoreofplayer          , tdigit0) ;
hexout u2(iscoreofplayer - 10    , tdigit1) ;
hexout u3(iscoreofplayer - 20    , tdigit2) ;
hexout u4(iscoreofplayer - 30    , tdigit3) ;
hexout u5(iscoreofplayer - 40    , tdigit4) ;
hexout u6(iscoreofplayer - 50    , tdigit5) ;
hexout u7(iscoreofplayer - 60    , tdigit6) ;
hexout u8(iscoreofplayer - 70    , tdigit7) ;
hexout u9(iscoreofplayer - 80    , tdigit8) ;
hexout u10(iscoreofplayer - 90    , tdigit9) ;


always @ (*)
begin
        if (iscoreofplayer < 10)
        begin
        digit1 = tdigit0    ;
        digit2 = 7'b0111111 ;
        digit3 = 7'b0111111 ;
        end

        else if (iscoreofplayer < 20)
        begin
        digit1 = tdigit1    ;
        digit2 = 7'b0000110 ;
        digit3 = 7'b0111111 ;
        end

        else if (iscoreofplayer < 30)
        begin
        digit1 = tdigit2    ;
        digit2 = 7'b1011011 ;
        digit3 = 7'b0111111 ;
        end

        else if (iscoreofplayer < 40)
        begin
```

```
digit1 = tdigit3    ;
digit2 = 7'b1001111 ;
digit3 = 7'b0111111 ;
end
else if (iscoreofplayer < 50)
begin
digit1 = tdigit4    ;
digit2 = 7'b1100110 ;
digit3 = 7'b0111111 ;
end
else if (iscoreofplayer < 60)
begin
digit1 = tdigit5    ;
digit2 = 7'b1101101 ;
digit3 = 7'b0111111 ;
end
else if (iscoreofplayer < 70)
begin
digit1 = tdigit6    ;
digit2 = 7'b1111101 ;
digit3 = 7'b0111111 ;
end
else if (iscoreofplayer < 80)
begin
digit1 = tdigit7    ;
digit2 = 7'b0000111 ;
digit3 = 7'b0111111 ;
end
else if (iscoreofplayer < 90)
begin
digit1 = tdigit8    ;
digit2 = 7'b1111111 ;
digit3 = 7'b0111111 ;
end

else if (iscoreofplayer < 100)
begin
digit1 = tdigit9    ;
digit2 = 7'b1100111 ;
digit3 = 7'b0111111 ;
end
```

Three-dimension Tiles



The tiles in the game looks three-dimension. It seems easy to implement this feature but actually it is not as easy as it seems.

If we use 10 bits for RGB values or even 8bits(24 bitmap color) for RGB values, it is an easy issue. Because it is easy to find shadow colors.

But it would be a nightmere if we use such complicated color space on such limited on-chip resouces.

Instead I used 4 bits for RGB values.

And the RGB inputs are assigned like:

```
assign    mVGA_R = {SRAM_DQ[11:8],6'b000000};
assign    mVGA_G = {SRAM_DQ[7:4 ] ,6'b000000};
assign    mVGA_B = {SRAM_DQ[3:0 ] ,6'b000000};
```

So the colors are not changed little by little, colors change vastly with bits change.

In this case ,it is difficult to find shadow colors (shadow colors are colors that are close to but a little darker than the original color). What's more, there are two kinds of shadows for each three-dimension tile :

shadows on the left and top, shadows on the right and bottom.

I solved the problem by traversing almost all the colors in color space, By putting huge amounts of different blocks on the screen , and pick out the good ones. That's the way I finally have the 10 color types in my game. Their 4-bit values are shown below:

```
/*colors :
        * light blue     :   86cf
        * dark   blue    :   448f
        * red            :   8c66
        * green          :   64a6
        * white          :   6cee
        * orange         :   1fb5
        * yellow         :   6fd4
        * purple         :   98e
        * brown          :   5766
        * pink           :   8c9b

       */
```

IV. W/R Control

There is a challenge throughout the design of the project : keep the system as efficient as possible.

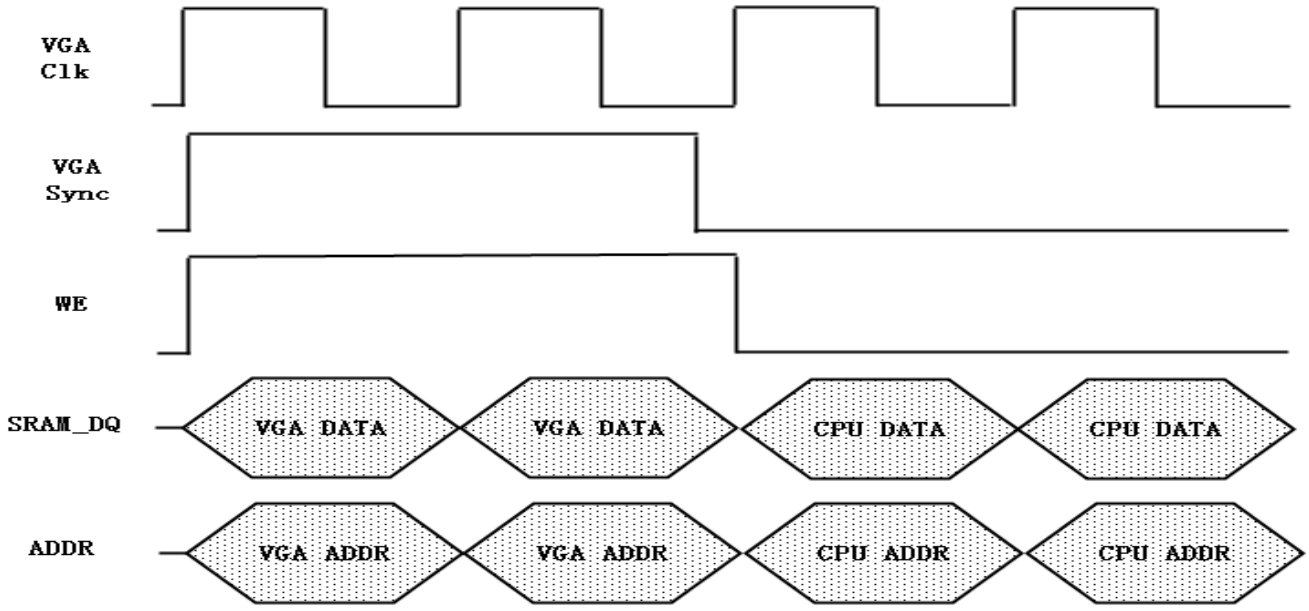
In the implementation of SRAM control, there is a challenge of how to make full use of SRAM.

Because all images are read from SRAM, it is necessary to accelerate SRAM data access speed. The improvement space comes from the fact that SRAM works at 50 MHz while VGA works at 25MHz.

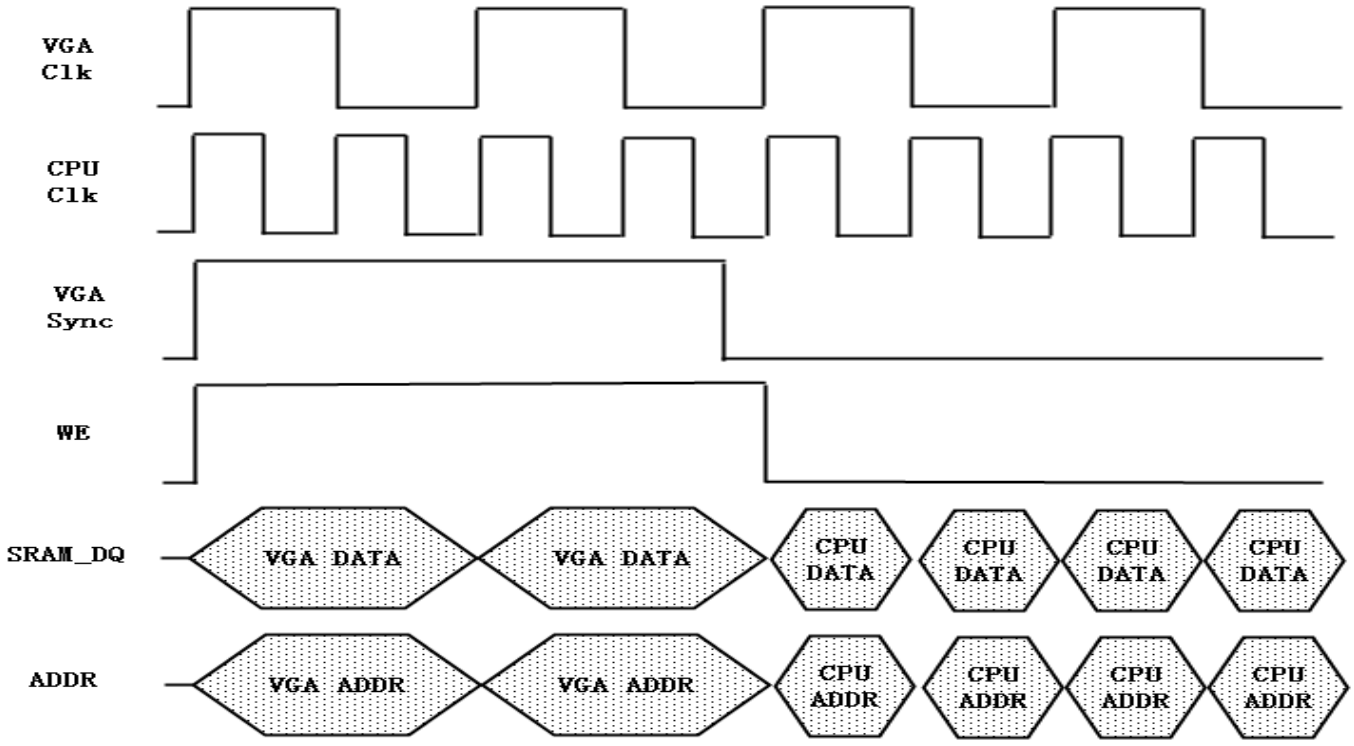
Originally, I only used the VGA_CLK to control both the read and write of SRAM. And when I run the program, I found it rather slow to refresh the game screen (I used to implement all images through software) .

Thanks to Prof. Edwards' advice, he points out that SRAM can work at a faster rate : The 'write' process can be controlled by CPU_CLK. That would make the read and write coding much more complicated but much faster.

The following pictures show clearly how the process was accelerated.:



Before Optimization



After Optimization

In order to correctly switch between CPU_CLK and VGA_CLK, a ‘flag’ was introduced to indicate whether SRAM is reading. SRAM only writes when flag == 0 .

Another trick is to switch between CPU address and VGA address. A vga_addr_flag is then introduced to indicate whether VGA is reading SRAM.

Core codes are shown below:

```
assign SRAM_ADDR = (vga_addr_flag) ? vga_addr_reg:cpu_addr_reg;
assign SRAM_DQ = (we_vga | we_cpu)? 16'hzzzz : data_reg ;
assign SRAM_UB_N = 0;                    // hi byte select enabled
assign SRAM_LB_N = 0;                    // lo byte select enabled
assign SRAM_CE_N = 0;                    // chip is enabled
assign SRAM_WE_N = (we_vga | we_cpu);                    // write when ZERO

// VGA Cycles
always @ (posedge VGA_CLK )
begin
    trial[0] = 12'hcba;
     trial[1] = 4'b1011;
     trial[2] = 4'hc;
    if (VGA_VS & VGA_HS)
     begin
     //READ
     vga_addr_flag <=1;
     flag <=0;
     vga_addr_reg <= {Coord_X[9:1],Coord_Y[8:0]} ;
     we_vga <= 1'b1;
     end
     //Write when syncing
    else
     begin
     vga_addr_flag <=0;
     flag <=1;
     we_vga <= 1'b0;
     end
end
```

```
//CPU Cycles
always@(posedge CPU_CLK)
begin
 if(flag)
    begin
     if (reset)        //synch reset assumes KEY0 is held down 1/60 second
      begin
          //clear the screen
          cpu_addr_reg <= {Coord_X[9:1],Coord_Y[8:0]} ;   // [17:0]
          we_cpu <= 1'b0;                                          //write some memory
          data_reg <= 16'b0;                                  //write all zeros (black)
          counter <= 9'o330 ;
          counter1 <= 4'b0000 ;
      end
      else
      begin
          cpu_addr_reg <= {x_ptr[9:1],y_ptr[8:0]};        // [17:0]
          we_cpu <= 1'b0;                                              //write some memory
          data_reg <= out_sram_data;
      end
     end
     end
```

V. PS/2 Control

Keyboard acts as the only controller of the Tetris game. Players use four arrows and the space key (rotation) to control the tiles.

There is an major issue for keyboard control: avoid multi-moves in one single press. The ps/2 clock works at 50 MHz, so the time is different between a single press and a single clock cycle. The controller gets a scan code every clock cycle when key is pressed. But within one press, we only need one input into the software.

So key counters are introduced into implementation. In my implementation, two count variables are used: Count and Count1.

Count adds itself by 1 when scan code from PS/2 arrives. To avoid multi-moves , set Count to 0 once Count reaches 3. And only when Count reaches 3 do we assign Count1 +=1. The CPU only responds to Count1.

The threshold ‘3’ is chosen by experience. Actually in this game, the system should respond to key inputs as agile as possible while multi-moves should be avoided. So the threshold ‘3’ is the balance of the two factors.

Within each press 3 scan codes are delivered to the CPU, namely’ press’, ‘release’ and ‘hold’. The software will judge keyboard inputs according to the combination of the 3 scan code and make relative moves.

In order to maintain these three scan codes as a keyboard input unit, I

used an reg :

```
reg [7:0] history [4:1];
```

and update the “history” reg when scan code arrives:

```
always @(posedge scan_ready or posedge reset)
begin
  if ( reset == 1'b1)
  begin
    history[4] <= 8'd0 ;
    history[3] <= 8'd0 ;
    history[2] <= 8'd0 ;
    history[1] <= 8'd0 ;
  end
  else
  begin
    history[4] <= history[3];
    history[3] <= history[2];
    history[2] <= history[1];
    history[1] <= scan_code;
    end
end
```

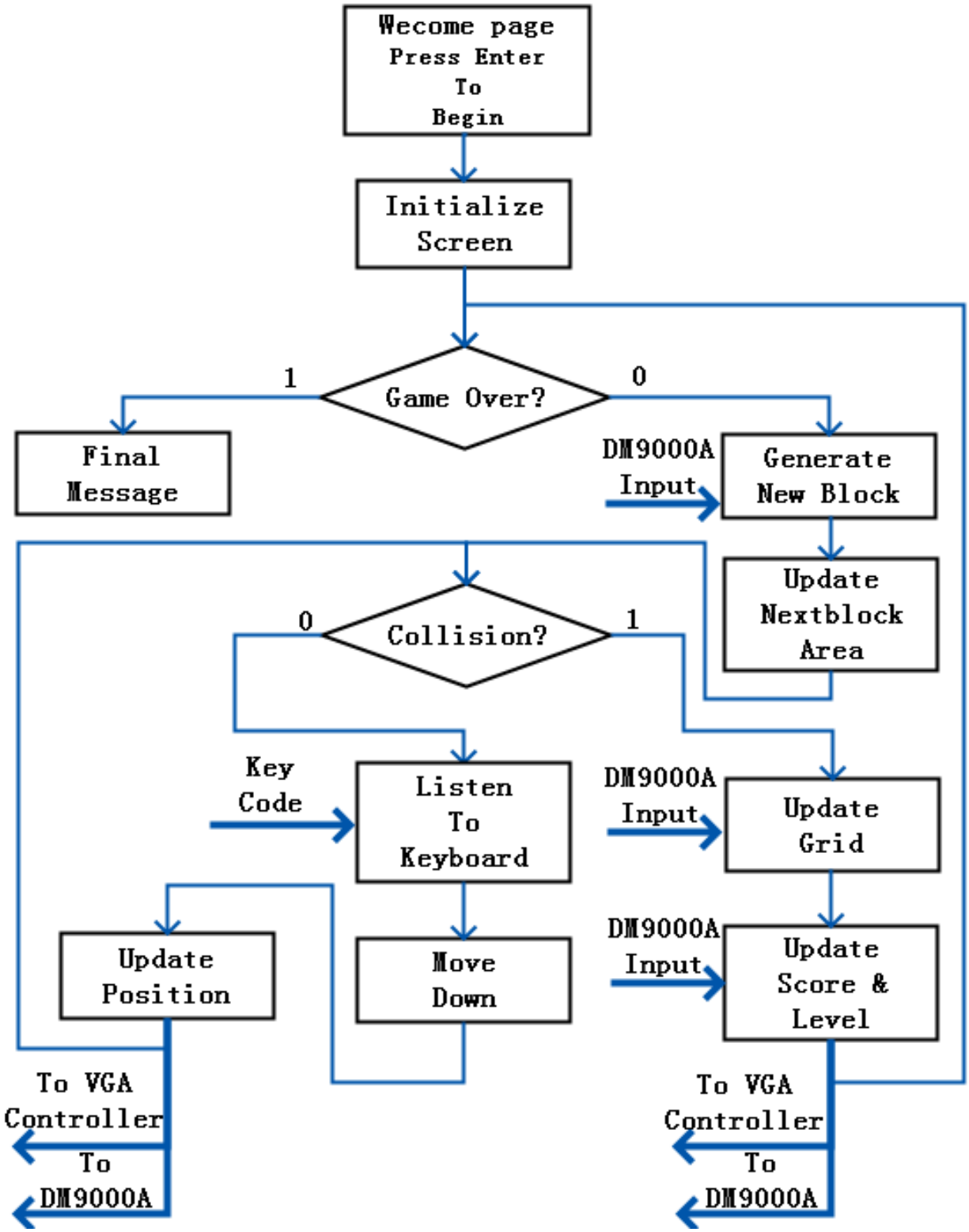
And `assign keyboardscancode = {history[3] , history[2] , history[1]};`

The “keyboardscancode” is what we need to send to the CPU to update key inputs.

VI. DM9000A

In this game, two clients are connected directly through a cross-line ,via Ethernet cable. The protocol used for communicating game messages is UDP just like Lab2. The send and receive organisms are the same as Lab2.

VII. Game Software



Software is the soul of this project. Generally it conducts the hardware to work as we wish.

The most fancy and yet complicated part of software for this game is block generating. In order to make the tiles look three-dimensional, I chose 12bits Color space ,RGB has 4 bits respectively. Color information is sent to SRAM through PIO “outsramdata”.

There is a conversion from 4 bits to 10bits to input RGB into VGA Controller. Theoretically, There can be 4096 different colors displayed. Conversion is done in pattern as follows:

```
assign    mVGA_R = {SRAM_DQ[11:8],6'b000000};
assign    mVGA_G = {SRAM_DQ[7:4 ] ,6'b000000};
assign    mVGA_B = {SRAM_DQ[3:0 ] ,6'b000000};
```

The software also have detailed outputs in Console, helping the programmer to analyze the game process:

```
[nitializing line [380] to [390] ... 79.33%
[nitializing line [390] to [400] ... 81.42%
[nitializing line [400] to [410] ... 83.51%
[nitializing line [410] to [420] ... 85.59%
[nitializing line [420] to [430] ... 87.68%
[nitializing line [430] to [440] ... 89.77%
[nitializing line [440] to [450] ... 91.86%
[nitializing line [450] to [460] ... 93.95%
[nitializing line [460] to [470] ... 96.03%
[nitializing line [470] to [480] ... 98.12%
[nitializing line [470] to [480] ... 100.00%
]creen initialized ! Game Starts!
**********************************************
 Events Log:

keycode: e0f06b  -- 'left'
keycode: e0f074  -- 'right'
keycode: e0f074  -- 'right'
keycode: e0f074  -- 'right'
keycode: e0f075  -- 'up'(rotate)
keycode: e0f074  -- 'right'
keycode: e0f06b  -- 'left'
keycode: e0f06b  -- 'left'
keycode: e0f072  -- 'down'
keycode: 29f029  -- 'space'(to bottom)
```

VIII.Experiences

The experience to implement a game all by one person is a great experience, especially for boys, because it is always cooler to build a game than to play one. In the developing process , the programmer is the rule maker .

But it is not so easy to make a gorgeous game given limited onchip resources and limited time. In my game, I originally wanted to make the background look like a digital photo. But it turned out that it exceeds the SRAM size. So I chose another pattern : make it clean and tidy ,rather than fancy.

The strive to make the blocks look three-dimensional is also a hard one: It is easy to make blocks look three-dimensional if we use 10bits for each RGB color. But it will make the game a lot slower and may exceed the memory.

In this game I used four bits for each RGB color. So it is difficult to find colors close to but darker than a given color (The shadow effects need some of the pixels to display close but darker color). Because when color bits changes, it changes a lot but not a little.

I finally solved this problem by traversing all the possible colors and choosing reasonable ones among them, which is a time-consuming work.

There are a lot more detailed difficulties : almost every final detail is

through times of errors and fixes.

But everything is worthwhile, building a whole system : from hardware to software, from bottom to top helps me solve questions in a greater vision . It is an interesting job both solving detailed problems and macroscopic problems. I think this experience will not only help me with later programings but also help me with any problem I meet in my life.

Finally, I want to thank Prof. Edwards for his generous helps on this project. Without him the project would have been a much harder one.

IX.Source Code

The VGA_Controller has 3829 lines of codes, so it is not showed here.

Tetris.c is the main c file in software.

DE2_Default.v is the top level design in hardware.

Tetris.c:

```
// system.h has peripheral base addresses, IRQ definitions, and cpu details
#include "system.h"

#include "sys/alt_irq.h"
#include "altera_avalon_pio_regs.h"
#include <unistd.h> //e.g. //usleep(5000000); is 5 seconds
#include <stdio.h>
#include "functions.h"
#include "altera_avalon_pio_regs.h"
#include <io.h>
#include "DM9000A.h"
#include "basic_io.h"
#include "functions.h"
#include "altera_avalon_uart_regs.h"

#define gridconst 0x12345678

//DM9000
#define MAX_MSG_LENGTH 128
#define UDP_PACKET_PAYLOAD_OFFSET 42
#define UDP_PACKET_LENGTH_OFFSET 38
#define UDP_PACKET_PAYLOAD (transmit_buffer + UDP_PACKET_PAYLOAD_OFFSET)

#define STATE_INIT          0
#define STATE_CONNECTED 1
#define STATE_ME_START    2
#define STATE_HIM_START 3
#define STATE_START         4

// Ethernet MAC address.    Choose the last three bytes yourself
unsigned char mac_address[6] = { 0x01, 0x60, 0x6E, 0x11, 0x22, 0x33    };
```

```
unsigned int interrupt_number;

unsigned int receive_buffer_length;
unsigned char receive_buffer[1600];
unsigned int state = STATE_INIT;


char spcount1    , spcount2      , spcount3    , spcount4 ;      //space vars to calculate new positions.
every block contains 4 units.
char spcount1_r, spcount2_r , spcount3_r, spcount4_r ; //right battle area vars
int score_lines = 0 ; //score
volatile int count = 0;
volatile int othermadeline ;

      //keyboard
int edgeregister =    0 ;
int previouscount = 0 ;
int currentcount = 0 ;
int w    , cgover;

char boardgrid[20][10]    , tempgrid[20][10]      , tempgrid1[20][10]      , tempgrid2[20][10] ;
char boardgrid_r[20][10], tempgrid_r[20][10] , tempgrid1_r[20][10] , tempgrid2_r[20][10];
int i , j ;

int gameover = 0 ;
int gameover_r = 0;


//on the left
struct tetris_piece current_blk    , uu    , current_blk1    ;
//on the right
struct tetris_piece current_blk_r, uu_r, current_blk1_r;

//KB_CODE_TYPE decode_mode;

#define UDP_PACKET_PAYLOAD_OFFSET 42
#define UDP_PACKET_LENGTH_OFFSET 38

#define UDP_PACKET_PAYLOAD (transmit_buffer + UDP_PACKET_PAYLOAD_OFFSET)

unsigned char transmit_buffer[] = {
  // Ethernet MAC header
  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, // Destination MAC address
```

```
  0x01, 0x60, 0x6E, 0x11, 0xE2, 0x0F, // Source MAC address
  0x08, 0x00,                          // Packet Type: 0x800 = IP

  // IP Header
  0x45,              // version (IPv4), header length = 20 bytes
  0x00,              // differentiated services field
  0x00,0x9C,         // total length: 20 bytes for IP header +
                     // 8 bytes for UDP header + 128 bytes for payload
  0x00, 0x00,        // packet ID
  0x00,              // flags
  0x00,              // fragment offset
  0x01,              // time-to-live
  0x11,              // protocol: 11 = UDP
  0xb6,0x00,         // header checksum: incorrect
  0xc0,0xa8,0x01,0x01, // source IP address
  0xFF,0xFF,0xFF,0xFF, // destination IP address

  // UDP Header
  0x67,0xd9, // source port port (26585: garbage)
  0x27,0x2b, // destination port (10027: garbage)
  0x00,0x88, // length (136: 8 for UDP header + 128 for data)
  0x00,0x00, // checksum: 0 = none

  // UDP payload
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67
};

static void ethernet_interrupt_handler() {
  unsigned int receive_status;
```

```
int i;
int sprite_num;
int curMsgChar = 0;
unsigned int packet_length;

receive_status = ReceivePacket(receive_buffer, &receive_buffer_length);

if (receive_status == DMFE_SUCCESS) {

  if (receive_buffer_length >= 14) {
    //    A real Ethernet packet
    if (receive_buffer[12] == 8 && receive_buffer[13] == 0 &&
    receive_buffer_length >= 34) {
  // An IP packet
  if (receive_buffer[23] == 0x11) {
    // A UDP packet
    if (receive_buffer_length >= UDP_PACKET_PAYLOAD_OFFSET) {

      //********************** Player 2 Logic **********************
      //    printf("%s\n",receive_buffer+UDP_PACKET_PAYLOAD_OFFSET);

        if((state == STATE_INIT ) &&(receive_buffer[42]=='z')&&(receive_buffer[43]=='z')) {

            printf("    Connection Verified! Game Ready!\n\n"); state==STATE_CONNECTED;

            for (curMsgChar=MAX_MSG_LENGTH-1; curMsgChar>0; curMsgChar--) {
            UDP_PACKET_PAYLOAD[curMsgChar] = 0;
              }
       //"zz" is the signal for connection verification

        UDP_PACKET_PAYLOAD[curMsgChar++] = 'z';
        UDP_PACKET_PAYLOAD[curMsgChar++] = 'z';
        UDP_PACKET_PAYLOAD[curMsgChar++] = 0;
        packet_length = UDP_PACKET_PAYLOAD_OFFSET + curMsgChar;
        transmit_buffer[UDP_PACKET_LENGTH_OFFSET] = packet_length >> 8;
        transmit_buffer[UDP_PACKET_LENGTH_OFFSET + 1] = packet_length & 0xff;
        if (TransmitPacket(transmit_buffer, UDP_PACKET_PAYLOAD_OFFSET + curMsgChar
+ 1)==DMFE_SUCCESS);
        if (TransmitPacket(transmit_buffer, UDP_PACKET_PAYLOAD_OFFSET + curMsgChar
+ 1)==DMFE_SUCCESS);
                }

      //game start signal
      else    if((state =
```

```
STATE_CONNECTED)&&(receive_buffer[42]=='z')&&(receive_buffer[43]=='s'))
                {

                    state = STATE_HIM_START;
                    printf("Player 2 is Ready,Pleas press 'enter' to start game!");

                }
               else if (receive_buffer[42]=='a')
               {
                  sprite_num = receive_buffer[43];
           //         printf("
received :%d\n",sprite_num);

                  current_blk_r.c1.x = piecessprites_r[sprite_num][0] ;
                  current_blk_r.c1.y = piecessprites_r[sprite_num][1] ;
                  current_blk_r.c2.x = piecessprites_r[sprite_num][2] ;
                  current_blk_r.c2.y = piecessprites_r[sprite_num][3] ;
                  current_blk_r.c3.x = piecessprites_r[sprite_num][4] ;
                  current_blk_r.c3.y = piecessprites_r[sprite_num][5] ;
                  current_blk_r.c4.x = piecessprites_r[sprite_num][6] ;
                  current_blk_r.c4.y = piecessprites_r[sprite_num][7] ;
                  current_blk_r.type = piecessprites_r[sprite_num][8] ;
                  current_blk_r.state =piecessprites_r[sprite_num][9] ;
                  current_blk_r.color =piecessprites_r[sprite_num][10] ;


                   }
               else if ((receive_buffer[42]=='e')&&(receive_buffer[43]=='q'))
               {
                  current_blk1_r = current_blk_r;

                   }
               else if ((receive_buffer[42]=='e')&&(receive_buffer[43]=='r'))
               {
                  current_blk_r = current_blk1_r;

                   }
               else if (receive_buffer[42]=='b')
               {
                  sprite_num = receive_buffer[43];
           //         printf("
received :%d\n",sprite_num);

                  current_blk1_r.c1.x = piecessprites_r[sprite_num][0] ;
```

```
                current_blk1_r.c1.y = piecessprites_r[sprite_num][1] ;
                current_blk1_r.c2.x = piecessprites_r[sprite_num][2] ;
                current_blk1_r.c2.y = piecessprites_r[sprite_num][3] ;
                current_blk1_r.c3.x = piecessprites_r[sprite_num][4] ;
                current_blk1_r.c3.y = piecessprites_r[sprite_num][5] ;
                current_blk1_r.c4.x = piecessprites_r[sprite_num][6] ;
                current_blk1_r.c4.y = piecessprites_r[sprite_num][7] ;
                current_blk1_r.type = piecessprites_r[sprite_num][8] ;
                current_blk1_r.state =piecessprites_r[sprite_num][9] ;
                current_blk1_r.color =piecessprites_r[sprite_num][10] ;


                 }
            else if ((receive_buffer[42]=='z')&&(receive_buffer[43]=='a'))
            {
                draw(current_blk_r);

                 }
            else if ((receive_buffer[42]=='z')&&(receive_buffer[43]=='p'))
            {
                uu_r = current_blk_r;
                uu_r.c1.x = receive_buffer[44]+320;      //+320 because we should display the origin
block into the right battle field
                uu_r.c1.y = receive_buffer[45];
                uu_r.c2.x = receive_buffer[46]+320;
                uu_r.c2.y = receive_buffer[47];
                uu_r.c3.x = receive_buffer[48]+320;
                uu_r.c3.y = receive_buffer[49];
                uu_r.c4.x = receive_buffer[50]+320;
                uu_r.c4.y = receive_buffer[51];

                drawpiece_tnext(current_blk_r , uu_r) ;
                delay2();
                delay2();
                delay2();

                 }

    }
  } else {
    printf("Received non-UDP packet\n");
  }
    } else {
  printf("Received non-IP packet\n");
```

```
            }
        } else {
            printf("Malformed Ethernet packet\n");
        }

    } else {
        printf("Error receiving packet\n");
    }

    /* Display the number of interrupts on the LEDs */
    interrupt_number++;
  // outport(SEG7_DISPLAY_BASE, interrupt_number);

    /* Clear the DM9000A ISR: PRS, PTS, ROS, ROOS 4 bits, by RW/C1 */
    dm9000a_iow(ISR, 0x3F);

    /* Re-enable DM9000A interrupts */
    dm9000a_iow(IMR, INTR_set);
}



int main(void)
{


        //set flag to hide beginning screen
  //      IOWR_ALTERA_AVALON_PIO_DATA(PIO_FLAG_BASE,0xff);


        //UDP packet
        int curMsgChar = 0;
        unsigned int packet_length;
        state = 0;




printf("\n=============================================================================
=============\n");
        printf("*                        Welcome to Tetris v 1.0 ....
*\n");
        printf("*                        Copy rights Open to All
```

```
*\n");
	printf("*                                                                    Yunfan
Dong     *\n");

printf("================================================================================
============\n");

	//*******************************Preparations
*************************************

	(DM9000_init(mac_address))?printf("    DM9000A failed!\n"):printf("    DM9000A successfully
initialized!\n");

printf("================================================================================
============\n");
	DM9000_init(mac_address);
	interrupt_number = 0;
	alt_irq_register(DM9000A_IRQ, NULL, (void*)ethernet_interrupt_handler);



	 //initialize the grid
	for (i = 0 ; i < 20 ; i++)
	{
		for(j = 0 ; j < 10 ; j++)
		{
			boardgrid[i][j] = 0 ;
			tempgrid [i][j] = 0 ;
		}
	}

	for (i = 0 ; i < 20 ; i++)
	{
		for(j = 0 ; j < 10 ; j++)
		{
			boardgrid_r[i][j] = 0 ;
			tempgrid_r [i][j] = 0 ;
		}
	}



	//printf("hello world \n") ;
```

```
int random = rand() % 14    ;


//initialize a block on the left
current_blk.c1.x = piecessprites[random][0] ;
current_blk.c1.y = piecessprites[random][1] ;
current_blk.c2.x = piecessprites[random][2] ;
current_blk.c2.y = piecessprites[random][3] ;
current_blk.c3.x = piecessprites[random][4] ;
current_blk.c3.y = piecessprites[random][5] ;
current_blk.c4.x = piecessprites[random][6] ;
current_blk.c4.y = piecessprites[random][7] ;
current_blk.type = piecessprites[random][8] ;
current_blk.state =piecessprites[random][9] ;
current_blk.color =piecessprites[random][10] ;

//send [random] to the other terminal

  for (curMsgChar=MAX_MSG_LENGTH-1; curMsgChar>0; curMsgChar--) {
      UDP_PACKET_PAYLOAD[curMsgChar] = 0;
    }

  UDP_PACKET_PAYLOAD[curMsgChar++] = 'a';
  UDP_PACKET_PAYLOAD[curMsgChar++] = random;
  UDP_PACKET_PAYLOAD[curMsgChar++] = 0;
  packet_length = UDP_PACKET_PAYLOAD_OFFSET + curMsgChar;
  transmit_buffer[UDP_PACKET_LENGTH_OFFSET] = packet_length >> 8;
  transmit_buffer[UDP_PACKET_LENGTH_OFFSET + 1] = packet_length & 0xff;
  if (TransmitPacket(transmit_buffer, UDP_PACKET_PAYLOAD_OFFSET + curMsgChar +
1)==DMFE_SUCCESS);
//    printf("      Message Sent! Sprite num:%d \n",random);



current_blk1      = current_blk ;

for (curMsgChar=MAX_MSG_LENGTH-1; curMsgChar>0; curMsgChar--) {
        UDP_PACKET_PAYLOAD[curMsgChar] = 0;
        }

        UDP_PACKET_PAYLOAD[curMsgChar++] = 'e';
        UDP_PACKET_PAYLOAD[curMsgChar++] = 'q';
        UDP_PACKET_PAYLOAD[curMsgChar++] = 0;
```

```
            packet_length = UDP_PACKET_PAYLOAD_OFFSET + curMsgChar;
            transmit_buffer[UDP_PACKET_LENGTH_OFFSET] = packet_length >> 8;
            transmit_buffer[UDP_PACKET_LENGTH_OFFSET + 1] = packet_length & 0xff;
            if (TransmitPacket(transmit_buffer, UDP_PACKET_PAYLOAD_OFFSET + curMsgChar
+ 1)==DMFE_SUCCESS);
   //    current_blk1_r = current_blk_r;



      //********************Preperations************************     only in sigle game
      //listen for key 'space or enter' to begin the game!




    printf("    Press ENTER to begin!\n");
    IOWR_ALTERA_AVALON_PIO_DATA(PIO_FLAG_BASE,0x00);
    IOWR_ALTERA_AVALON_PIO_DATA(KEYCOUNTER_BASE,0x00);



    while(state!=STATE_START){

        IOWR_ALTERA_AVALON_PIO_DATA(PIO_FLAG_BASE,0x02);
        usleep(1000000);
        IOWR_ALTERA_AVALON_PIO_DATA(PIO_FLAG_BASE,0x00);
        usleep(1000000);
        int cnt;



            edgeregister =
IORD_ALTERA_AVALON_PIO_DATA(KEYBOARDSCANCODE_BASE);
            previouscount = currentcount ;
            currentcount = IORD_ALTERA_AVALON_PIO_DATA(KEYCOUNTER_BASE);
      //        printf("\ncurrent count:%d    0x%x ",currentcount,edgeregister);
            if(currentcount>previouscount){

                if((edgeregister == 0x29f029)||(edgeregister == 0x5af05a))    /// 'space' or 'enter'
                  {
                  state = STATE_START;
                    }
                  IOWR_ALTERA_AVALON_PIO_DATA(KEYCOUNTER_BASE,0);
```

```
                    }


            }









    printf("****************************************\n Initializing
Screen .....\n****************************************\nlog:\n\n");
//    IOWR_ALTERA_AVALON_PIO_DATA(PIO_FLAG_BASE,0x01);
    IOWR_ALTERA_AVALON_PIO_DATA(PIO_FLAG_BASE,0x05);
    IOWR_ALTERA_AVALON_PIO_DATA(SCOREOFPLAYER_BASE,0);
    initializescreen();



//*********************************************************************************************
*********
    //**********************************main loop
**********************************************

    while( ( gameover == 0 ) && ( gameover_r == 0 ) )    //no player has ended

      {
      for (curMsgChar=MAX_MSG_LENGTH-1; curMsgChar>0; curMsgChar--) {
            UDP_PACKET_PAYLOAD[curMsgChar] = 0;
            }

            UDP_PACKET_PAYLOAD[curMsgChar++] = 'e';
            UDP_PACKET_PAYLOAD[curMsgChar++] = 'r';
            UDP_PACKET_PAYLOAD[curMsgChar++] = 0;
            packet_length = UDP_PACKET_PAYLOAD_OFFSET + curMsgChar;
            transmit_buffer[UDP_PACKET_LENGTH_OFFSET] = packet_length >> 8;
            transmit_buffer[UDP_PACKET_LENGTH_OFFSET + 1] = packet_length & 0xff;
            if (TransmitPacket(transmit_buffer, UDP_PACKET_PAYLOAD_OFFSET + curMsgChar
+ 1)==DMFE_SUCCESS);
```

```
    //assigning the initial piece
    current_blk      = current_blk1 ;
   // current_blk_r = current_blk1_r;

    // generate new blk
    random = rand() % 14;

    for (curMsgChar=MAX_MSG_LENGTH-1; curMsgChar>0; curMsgChar--) {
          UDP_PACKET_PAYLOAD[curMsgChar] = 0;
       }

     UDP_PACKET_PAYLOAD[curMsgChar++] = 'b';
     UDP_PACKET_PAYLOAD[curMsgChar++] = random;
     UDP_PACKET_PAYLOAD[curMsgChar++] = 0;
     packet_length = UDP_PACKET_PAYLOAD_OFFSET + curMsgChar;
     transmit_buffer[UDP_PACKET_LENGTH_OFFSET] = packet_length >> 8;
     transmit_buffer[UDP_PACKET_LENGTH_OFFSET + 1] = packet_length & 0xff;
     if (TransmitPacket(transmit_buffer, UDP_PACKET_PAYLOAD_OFFSET + curMsgChar +
1)==DMFE_SUCCESS);


    current_blk1.c1.x = piecessprites[random][0] ;
    current_blk1.c1.y = piecessprites[random][1] ;
    current_blk1.c2.x = piecessprites[random][2] ;
    current_blk1.c2.y = piecessprites[random][3] ;
    current_blk1.c3.x = piecessprites[random][4] ;
    current_blk1.c3.y = piecessprites[random][5] ;
    current_blk1.c4.x = piecessprites[random][6] ;
    current_blk1.c4.y = piecessprites[random][7] ;
    current_blk1.type = piecessprites[random][8] ;
    current_blk1.state =piecessprites[random][9] ;
    current_blk1.color =piecessprites[random][10] ;



    for (curMsgChar=MAX_MSG_LENGTH-1; curMsgChar>0; curMsgChar--) {
          UDP_PACKET_PAYLOAD[curMsgChar] = 0;
          }

          UDP_PACKET_PAYLOAD[curMsgChar++] = 'z';
          UDP_PACKET_PAYLOAD[curMsgChar++] = 'a';
          UDP_PACKET_PAYLOAD[curMsgChar++] = 0;
          packet_length = UDP_PACKET_PAYLOAD_OFFSET + curMsgChar;
          transmit_buffer[UDP_PACKET_LENGTH_OFFSET] = packet_length >> 8;
```

```
            transmit_buffer[UDP_PACKET_LENGTH_OFFSET + 1] = packet_length & 0xff;
            if (TransmitPacket(transmit_buffer, UDP_PACKET_PAYLOAD_OFFSET + curMsgChar
+ 1)==DMFE_SUCCESS);


    draw(current_blk);

    //have to draw the next piece to come
    //****************************************

    clearnextpiecearea();
    drawnextpiece (current_blk1) ;

    char collision = 0;
    char collision_r = 0;

    //*************************************************************************************
    //*******************************collision loop1**********************************
    //**** parallel thread for battle area 1
    while (collision != 1)
    {
        int response ;
        for(response = 0 ; response < 30 ; response++)
        {


      //*********************************read key inputs*************************


        edgeregister =
IORD_ALTERA_AVALON_PIO_DATA(KEYBOARDSCANCODE_BASE);
        previouscount = currentcount ;
        currentcount = IORD_ALTERA_AVALON_PIO_DATA(KEYCOUNTER_BASE);

        //resetting in case some keystroke got carried over
            if(response == 0)
            {
                currentcount = previouscount ;
            }



        if(currentcount > previouscount)            //read the keyboard
```

```
{
    //if 'enter' output current grids in console (for debugging)
    if(edgeregister == 0x5af05a)
      {

        printf("left grid\n");
        for (i = 0 ; i < 20 ; i++)
         {
           for(j = 0 ; j < 10 ; j++)
              {

                printf("%d,", tempgrid [i][j] ) ;
                if(j==9)printf("\n");
               }
         }
        printf("\n\n");
        printf("right grid\n");
        for (i = 0 ; i < 20 ; i++)
         {
           for(j = 0 ; j < 10 ; j++)
              {

                printf("%d,", tempgrid_r [i][j] ) ;
                if(j==9)printf("\n");
               }
         }

      }

      //*********'left' key pressed
    //uu is a temp block to store the result of temp movement,if no collision, then
assign uu to current blk
    else if( edgeregister == 0xe0f06b )
    {
        uu    = moveleft(current_blk) ;
        printf("keycode: %x    -- 'left'\n" , edgeregister) ;
    }


    //*********'right' key pressed
   else if( edgeregister == 0xe0f074 )
    {
        uu    = moveright(current_blk);
        printf("keycode: %x    -- 'right'\n" , edgeregister) ;
```

```
                }
                //**********'up' key pressed
                else if( edgeregister == 0xe0f075 )
                {
                        uu    = rotate(current_blk);
                        printf("keycode: %x    -- 'up'(rotate)\n" , edgeregister) ;
                }
                //***********'down' key pressed
                else if( edgeregister == 0xe0f072 )
               {
                        uu = movedown(current_blk);
                        uu = movedown(uu) ;
                        printf("keycode: %x    -- 'down'\n" , edgeregister) ;
               }

               //************'spacebar' pressed
                  else if(edgeregister == 0x29f029)
                {
                        printf("keycode: %x    -- 'space'(to bottom)\n" , edgeregister) ;

                      //move the piece down to the lowest possible grids

                      //g1,g2,g2,g4 are relative positions in the grid, the grid is 10*20 , every grid
capable of one blk

                        struct coordinate g1    , g2    , g3    , g4 ;

                        g1 = coord_change(current_blk.c1);
                        g2 = coord_change(current_blk.c2);
                        g3 = coord_change(current_blk.c3);
                        g4 = coord_change(current_blk.c4);

                        // keep track of how many grids we move down in order to calculate exact
coordinates

                        spcount1 = 0 ;
                        spcount2 = 0 ;
                        spcount3 = 0 ;
                        spcount4 = 0 ;

                        // total height is 20
                        // x --- vertical      y --- horizontal

                        while( (tempgrid[g1.x + 1][g1.y] != 1) && (g1.x < 19))
```

```
{
    g1.x = g1.x + 1 ;
    spcount1 = spcount1 + 1 ;
}

while( (tempgrid[g2.x + 1][g2.y] != 1) && (g2.x < 19))
{
    g2.x = g2.x + 1 ;
    spcount2 = spcount2 + 1 ;
}

while( (tempgrid[g3.x + 1][g3.y] != 1) && (g3.x < 19))
{
    g3.x = g3.x + 1 ;
    spcount3= spcount3 + 1 ;
}

while( (tempgrid[g4.x + 1][g4.y] != 1) && (g4.x < 19))
{
    g4.x = g4.x + 1 ;
    spcount4 = spcount4 + 1 ;
}


//calculate how much the entity goes down

char min_x_down_allowed ;

min_x_down_allowed = spcount1 ;

if (spcount2 < min_x_down_allowed) min_x_down_allowed = spcount2 ;
if (spcount3 < min_x_down_allowed) min_x_down_allowed = spcount3 ;
if (spcount4 < min_x_down_allowed) min_x_down_allowed = spcount4 ;

uu = current_blk ;
uu.c1.y = uu.c1.y    + 20 * min_x_down_allowed ;
uu.c2.y = uu.c2.y    + 20 * min_x_down_allowed ;
uu.c3.y = uu.c3.y    + 20 * min_x_down_allowed ;
uu.c4.y = uu.c4.y    + 20 * min_x_down_allowed ;

}
```

```
                // check whether the corrdinates needed to be filled are    occupied,if so, the move is
illegal

                struct coordinate g1     , g2     , g3     , g4     ;     //coordinate is relative ordinate
to the grid

                g1 = coord_change(uu.c1);
                g2 = coord_change(uu.c2);
                g3 = coord_change(uu.c3);
                g4 = coord_change(uu.c4);


                //**********check if grid already occupied

                //left
                if( (tempgrid[g1.x][g1.y] == 1) || (tempgrid[g2.x][g2.y] == 1)
||(tempgrid[g3.x][g3.y] == 1)
                    || (tempgrid[g4.x][g4.y] == 1) )
                {

                    uu = current_blk ;
                }

                //now check if any of the coordinates are outside of boundaries
                else if( ((uu.c1.x < 50) || (uu.c2.x < 50) || (uu.c3.x < 50) || (uu.c4.x < 50))
                        || ((uu.c1.x > 230) || (uu.c2.x >230) || (uu.c3.x > 230) || (uu.c4.x >
230))    )
                {

                    uu = current_blk ;
                }
                else if (((uu.c1.y < 50) || (uu.c2.y < 50) || (uu.c3.y < 50) || (uu.c4.y < 50))
                        || ((uu.c1.y > 430) || (uu.c2.y > 430) || (uu.c3.y > 430) || (uu.c4.y > 430)))
                {
                  //the piece might go outside the top and bottom edges
                 // printf("33333333333333333333\n") ;
                    uu = current_blk ;
                }
                else
                {
                   //    legal move here . draw blk on new place,clear old place first
```

```
                        // send new position
                    for (curMsgChar=MAX_MSG_LENGTH-1; curMsgChar>0; curMsgChar--)
{
                     UDP_PACKET_PAYLOAD[curMsgChar] = 0;
                    }

                     UDP_PACKET_PAYLOAD[curMsgChar++] = 'z';
                     UDP_PACKET_PAYLOAD[curMsgChar++] = 'p';
                     UDP_PACKET_PAYLOAD[curMsgChar++] = uu.c1.x;
                     UDP_PACKET_PAYLOAD[curMsgChar++] = uu.c1.y;
                     UDP_PACKET_PAYLOAD[curMsgChar++] = uu.c2.x;
                     UDP_PACKET_PAYLOAD[curMsgChar++] = uu.c2.y;
                     UDP_PACKET_PAYLOAD[curMsgChar++] = uu.c3.x;
                     UDP_PACKET_PAYLOAD[curMsgChar++] = uu.c3.y;
                     UDP_PACKET_PAYLOAD[curMsgChar++] = uu.c4.x;
                     UDP_PACKET_PAYLOAD[curMsgChar++] = uu.c4.y;
                     UDP_PACKET_PAYLOAD[curMsgChar++] = 0;
                     packet_length = UDP_PACKET_PAYLOAD_OFFSET + curMsgChar;
                     transmit_buffer[UDP_PACKET_LENGTH_OFFSET] = packet_length >> 8;
                     transmit_buffer[UDP_PACKET_LENGTH_OFFSET + 1] = packet_length
& 0xff;
                     if (TransmitPacket(transmit_buffer, UDP_PACKET_PAYLOAD_OFFSET
+ curMsgChar + 1)==DMFE_SUCCESS);



                    drawpiece_tnext(current_blk , uu) ;
                    current_blk = uu ;
                }

                      delay2();


            }


        }

            //if there is a collision, set collision to 1,update the tempgrid bits
            //there is a collision if the grid bits are set at y below,or the lowest
            //y is already the bottom

            //first check if the lowest y on the grid is already at the bottom
```

```
unsigned short int max_y ;

max_y      = current_blk.c1.y ;

if (current_blk.c2.y > max_y) max_y = current_blk.c2.y ;
if (current_blk.c3.y > max_y) max_y = current_blk.c3.y ;
if (current_blk.c4.y > max_y) max_y = current_blk.c4.y ;


//***********************************************
//****** see if there is a collision with anything on the grid
int ymax1    , tempxcoord      , c1set      , c2set    , c3set      , c4set    ;


c1set =    0 ;
c2set =    0 ;
c3set =    0 ;
c4set =    0 ;

struct coordinate yy1    , yyy1    ;

// c1 check

//left area
tempxcoord = current_blk.c1.x ;

ymax1    = current_blk.c1.y ;
if ( (current_blk.c2.y > ymax1) && (current_blk.c2.x == tempxcoord)) ymax1 =
current_blk.c2.y ;
if ( (current_blk.c3.y > ymax1) && (current_blk.c3.x == tempxcoord)) ymax1 =
current_blk.c3.y ;
if ( (current_blk.c4.y > ymax1) && (current_blk.c4.x == tempxcoord)) ymax1 =
current_blk.c4.y ;
yy1.x = tempxcoord ;
yy1.y = ymax1 + 20 ;
yyy1    = coord_change(yy1) ;
if( tempgrid[yyy1.x][yyy1.y] == 1) c1set = 1 ;

//c2 check

//left area
tempxcoord = current_blk.c2.x ;

ymax1    = current_blk.c2.y ;
```

```
            if ( (current_blk.c1.y > ymax1) && (current_blk.c1.x == tempxcoord)) ymax1 =
current_blk.c1.y ;
            if ( (current_blk.c3.y > ymax1) && (current_blk.c3.x == tempxcoord)) ymax1 =
current_blk.c3.y ;
            if ( (current_blk.c4.y > ymax1) && (current_blk.c4.x == tempxcoord)) ymax1 =
current_blk.c4.y ;
            yy1.x = tempxcoord ;
            yy1.y = ymax1 + 20 ;
            yyy1    = coord_change(yy1) ;
            if( tempgrid[yyy1.x][yyy1.y] == 1) c2set = 1 ;


            //c3 check

            //left
            tempxcoord = current_blk.c3.x ;

            ymax1    = current_blk.c3.y ;
            if ( (current_blk.c1.y > ymax1) && (current_blk.c1.x == tempxcoord)) ymax1 =
current_blk.c1.y ;
            if ( (current_blk.c2.y > ymax1) && (current_blk.c2.x == tempxcoord)) ymax1 =
current_blk.c2.y ;
            if ( (current_blk.c4.y > ymax1) && (current_blk.c4.x == tempxcoord)) ymax1 =
current_blk.c4.y ;
            yy1.x = tempxcoord ;
            yy1.y = ymax1 + 20 ;
            yyy1    = coord_change(yy1) ;
            if( tempgrid[yyy1.x][yyy1.y] == 1) c3set = 1 ;


            //c4 check

            //left
            tempxcoord = current_blk.c4.x ;

            ymax1    = current_blk.c4.y ;
            if ( (current_blk.c1.y > ymax1) && (current_blk.c1.x == tempxcoord)) ymax1 =
current_blk.c1.y ;
            if ( (current_blk.c2.y > ymax1) && (current_blk.c2.x == tempxcoord)) ymax1 =
current_blk.c2.y ;
            if ( (current_blk.c3.y > ymax1) && (current_blk.c3.x == tempxcoord)) ymax1 =
current_blk.c3.y ;
            yy1.x = tempxcoord ;
            yy1.y = ymax1 + 20 ;
```

```
yyy1    = coord_change(yy1) ;
if( tempgrid[yyy1.x][yyy1.y] == 1) c4set = 1 ;

//*********check collision and update position
//left check
if(max_y == 430)                //piece is at the bottom of the grid
{
 //printf("at the bottom of the screen\n") ;
 //set the grid and update collision
 collision = 1 ;
 struct coordinate f1 , f2 , f3 , f4 ;
        f1 = coord_change(current_blk.c1);
        f2 = coord_change(current_blk.c2);
        f3 = coord_change(current_blk.c3);
        f4 = coord_change(current_blk.c4);

         ///update grid
        tempgrid[f1.x][f1.y] = 1 ;
        tempgrid[f2.x][f2.y] = 1 ;
        tempgrid[f3.x][f3.y] = 1 ;
        tempgrid[f4.x][f4.y] = 1 ;
}
else if((c4set + c3set + c2set + c1set) > 0 )              // touches blks already in the grid
{
 collision = 1 ;
 struct coordinate f1 , f2 , f3 , f4 ;
        f1 = coord_change(current_blk.c1);
        f2 = coord_change(current_blk.c2);
        f3 = coord_change(current_blk.c3);
        f4 = coord_change(current_blk.c4);
        tempgrid[f1.x][f1.y] = 1 ;
        tempgrid[f2.x][f2.y] = 1 ;
        tempgrid[f3.x][f3.y] = 1 ;
        tempgrid[f4.x][f4.y] = 1 ;
}

else    //if no collision move down and restart the collsion check loop
{
 uu = movedown(current_blk);

 for (curMsgChar=MAX_MSG_LENGTH-1; curMsgChar>0; curMsgChar--) {
              UDP_PACKET_PAYLOAD[curMsgChar] = 0;
            }
```

```
                        UDP_PACKET_PAYLOAD[curMsgChar++] = 'z';
                        UDP_PACKET_PAYLOAD[curMsgChar++] = 'p';
                        UDP_PACKET_PAYLOAD[curMsgChar++] = uu.c1.x;
                        UDP_PACKET_PAYLOAD[curMsgChar++] = uu.c1.y;
                        UDP_PACKET_PAYLOAD[curMsgChar++] = uu.c2.x;
                        UDP_PACKET_PAYLOAD[curMsgChar++] = uu.c2.y;
                        UDP_PACKET_PAYLOAD[curMsgChar++] = uu.c3.x;
                        UDP_PACKET_PAYLOAD[curMsgChar++] = uu.c3.y;
                        UDP_PACKET_PAYLOAD[curMsgChar++] = uu.c4.x;
                        UDP_PACKET_PAYLOAD[curMsgChar++] = uu.c4.y;
                        UDP_PACKET_PAYLOAD[curMsgChar++] = 0;
                        packet_length = UDP_PACKET_PAYLOAD_OFFSET + curMsgChar;
                        transmit_buffer[UDP_PACKET_LENGTH_OFFSET] = packet_length >> 8;
                        transmit_buffer[UDP_PACKET_LENGTH_OFFSET + 1] = packet_length
& 0xff;
                        if (TransmitPacket(transmit_buffer, UDP_PACKET_PAYLOAD_OFFSET
+ curMsgChar + 1)==DMFE_SUCCESS);

                drawpiece_tnext(current_blk , uu) ;
        //              printf("position updated!!");

               //speed control
                if(score_lines <3)
                delay1();
                else if (score_lines<6)
                delay11();

                else delay111();

               }

                current_blk = uu ;              //update for    next iteration of the loop

      }
        ///***************end of collision loop1*********************




    //*************************************************************
    //********** detect line fulfilment & update grid
```

```
int r , c , t , sum; //vars

r = 19 ; // x- paramter of new grid
t = 0 ;    //record score

sum = 0 ;

///**********leftside*****************
//traverse grid
for (i = 19 ; i >= 0    ; i--)
{
      for (j = 0 ; j < 10 ; j++)
      {
            sum = sum + tempgrid[i][j] ;
      }

      if(sum == 10) ///    fulfil
      {
      t = t + 1 ;
      }

      else                //if not
      {
      for (c = 0 ; c < 10 ; c++)
            {
            tempgrid1[r][c] = tempgrid[i][c] ;
            }
            r = r - 1 ;
      }
      sum = 0 ;
}


for (i = 0 ; i < t ; i++)
{
      for(j = 0 ; j < 10 ; j++)
      {
      tempgrid1[i][j] = 0 ;
      }
}
```

```
/*

///**********right side*****************
//traverse grid
for (i = 19 ; i >= 0    ; i--)
{
      for (j = 0 ; j < 10 ; j++)
      {
            sum = sum + tempgrid_r[i][j] ;
      }

      if(sum == 10) ///    fulfil
      {
      t = t + 1 ;
      }

      else                //if not
      {
      for (c = 0 ; c < 10 ; c++)
            {
            tempgrid1_r[r][c] = tempgrid_r[i][c] ;
            }
            r = r - 1 ;
      }
      sum = 0 ;
}


for (i = 0 ; i < t ; i++)
{
      for(j = 0 ; j < 10 ; j++)
      {
         tempgrid1_r[i][j] = 0 ;
      }
}


*/
```

```
    //*****************************************************
    //*****************update the score*********************
    score_lines = score_lines + t ;

    if(t!=0)printf("the score is updated to %d\n" , score_lines) ;
    IOWR_ALTERA_AVALON_PIO_DATA(SCOREOFPLAYER_BASE,score_lines);
    if(score_lines >=3) IOWR_ALTERA_AVALON_PIO_DATA(PIO_FLAG_BASE,0x09);
    if(score_lines >=6) IOWR_ALTERA_AVALON_PIO_DATA(PIO_FLAG_BASE,0x0d);
```

```
//*****************************************************************************************
*******
  //*******check    differences    between
  //*******the the boardgrid and the tempgrid1 and draw accordingly. Then update the
  //*******board grid and the tempgrid arrays
   int d , s ;
   for (d = 0 ; d < 20 ; d++)
   {
    for (s = 0 ; s < 10 ; s++)
    {
        if(tempgrid[d][s] == tempgrid1[d][s])
        {
            //do nothing
        }
        else
        {
            if(tempgrid[d][s] == 1)
            {
                createunit(20*s + 50 , 20*d + 50 , 0 );
                tempgrid[d][s] = 0 ;
            }
            else
            {
            //      createunit(20*s + 70 , 20*d + 50 , white );
                tempgrid[d][s] = 1 ;
            }
        }
    }
   }
```

```
 /*
for (d = 0 ; d < 20 ; d++)
{
 for (s = 0 ; s < 10 ; s++)
 {
      if(tempgrid_r[d][s] == tempgrid1_r[d][s])
      {
            //do nothing
      }
      else
      {
            if(tempgrid_r[d][s] == 1)
            {
                  createunit(20*s + 70 , 20*d + 50 , 0 );
                  tempgrid_r[d][s] = 0 ;
            }
            else
            {
            //       createunit(20*s + 70 , 20*d + 50 , white );
                  tempgrid_r[d][s] = 1 ;
            }
      }
 }
}

*/




//*****************************************************************************
    //check if the top row of the gris is full and stop the loop. Basically game over
        cgover = 0 ;
        for (w = 0 ; w < 10 ; w++)
        {
         cgover = cgover + tempgrid[1][w] ;
        }
        if(cgover > 1 ) {
         gameover = 1 ;
         IOWR_ALTERA_AVALON_PIO_DATA(KEYCOUNTER_BASE,0);
         printf("Game Over...");
         }
```

```
        } //this is the end of the second while loop

    // IOWR_ALTERA_AVALON_PIO_DATA(PIO_FLAG_BASE,0x00);
      finalmessage();


}
```

DE2_Default.V

```
//*********************Verilog Edition ***************************//

module DE2_Default
      (
            ////////////////////  Clock Input            ////////////////////
            CLOCK_27,                           //    27 MHz
            CLOCK_50,                           //    50 MHz
            EXT_CLOCK,                          //    External Clock
            ////////////////////  Push Button            ////////////////////
            KEY,                                //    Pushbutton[3:0]
            ////////////////////  DPDT Switch            ////////////////////
            SW,                                 //    Toggle Switch[17:0]
            ////////////////////  7-SEG Dispaly ////////////////////
            HEX0,                               //    Seven Segment Digit 0
            HEX1,                               //    Seven Segment Digit 1
            HEX2,                               //    Seven Segment Digit 2
            HEX3,                               //    Seven Segment Digit 3
            HEX4,                               //    Seven Segment Digit 4
            HEX5,                               //    Seven Segment Digit 5
            HEX6,                               //    Seven Segment Digit 6
            HEX7,                               //    Seven Segment Digit 7
            //////////////////////      LED       //////////////////////
            LEDG,                               //    LED Green[8:0]
            LEDR,                               //    LED Red[17:0]
            //////////////////////      UART      //////////////////////
            UART_TXD,                           //    UART Transmitter
            UART_RXD,                           //    UART Receiver
            //////////////////////      IRDA      //////////////////////
            IRDA_TXD,                           //    IRDA Transmitter
```

```
        IRDA_RXD,                       //  IRDA Receiver
        ////////////////////  SDRAM Interface       ////////////////
        DRAM_DQ,                        //  SDRAM Data bus 16 Bits
        DRAM_ADDR,                          //  SDRAM Address bus 12 Bits
        DRAM_LDQM,                          //  SDRAM Low-byte Data Mask
        DRAM_UDQM,                          //  SDRAM High-byte Data Mask
        DRAM_WE_N,                          //  SDRAM Write Enable
        DRAM_CAS_N,                         //  SDRAM Column Address Strobe
        DRAM_RAS_N,                         //  SDRAM Row Address Strobe
        DRAM_CS_N,                      //  SDRAM Chip Select
        DRAM_BA_0,                      //  SDRAM Bank Address 0
        DRAM_BA_1,                      //  SDRAM Bank Address 0
        DRAM_CLK,                       //  SDRAM Clock
        DRAM_CKE,                       //  SDRAM Clock Enable
        ////////////////////  Flash Interface       ////////////////
        FL_DQ,                          //  FLASH Data bus 8 Bits
        FL_ADDR,                        //  FLASH Address bus 22 Bits
        FL_WE_N,                        //  FLASH Write Enable
        FL_RST_N,                       //  FLASH Reset
        FL_OE_N,                        //  FLASH Output Enable
        FL_CE_N,                        //  FLASH Chip Enable
        ////////////////////  SRAM Interface        ////////////////
        SRAM_DQ,                        //SRAM Data bus 16 Bits
        SRAM_ADDR,                          //  SRAM Address bus 18 Bits
        SRAM_UB_N,                      //  SRAM High-byte Data Mask
        SRAM_LB_N,                      //  SRAM Low-byte Data Mask
        SRAM_WE_N,                          //  SRAM Write Enable
        SRAM_CE_N,                      //  SRAM Chip Enable
        SRAM_OE_N,                      //  SRAM Output Enable

        ////////////////////  ISP1362 Interface     ////////////////
        OTG_DATA,                       //  ISP1362 Data bus 16 Bits
        OTG_ADDR,                       //  ISP1362 Address 2 Bits
        OTG_CS_N,                       //  ISP1362 Chip Select
        OTG_RD_N,                       //  ISP1362 Write
        OTG_WR_N,                       //  ISP1362 Read
        OTG_RST_N,                      //  ISP1362 Reset
        OTG_FSPEED,                         //  USB Full Speed,     0 = Enable, Z =
Disable
        OTG_LSPEED,                         //  USB Low Speed,      0 = Enable, Z =
Disable
        OTG_INT0,                       //  ISP1362 Interrupt 0
        OTG_INT1,                       //  ISP1362 Interrupt 1
        OTG_DREQ0,                      //  ISP1362 DMA Request 0
```

```
        OTG_DREQ1,                      //  ISP1362 DMA Request 1
        OTG_DACK0_N,                    //  ISP1362 DMA Acknowledge 0
        OTG_DACK1_N,                    //  ISP1362 DMA Acknowledge 1
        ////////////////////  LCD Module 16X2  ////////////////
        LCD_ON,                         //  LCD Power ON/OFF
        LCD_BLON,                       //  LCD Back Light ON/OFF
        LCD_RW,                             //  LCD Read/Write Select, 0 = Write, 1 =
Read
        LCD_EN,                         //  LCD Enable
        LCD_RS,                         //  LCD Command/Data Select, 0 = Command,
1 = Data
        LCD_DATA,                       //  LCD Data bus 8 bits
        ////////////////////  SD_Card Interface  ////////////////
        SD_DAT,                         //  SD Card Data
        SD_DAT3,                        //  SD Card Data 3
        SD_CMD,                             //  SD Card Command Signal
        SD_CLK,                         //  SD Card Clock
        ////////////////////  USB JTAG link  ////////////////////
        TDI,                            // CPLD -> FPGA (data in)
        TCK,                            // CPLD -> FPGA (clk)
        TCS,                            // CPLD -> FPGA (CS)
        TDO,                            // FPGA -> CPLD (data out)
        ////////////////////  I2C  ///////////////////////////
        I2C_SDAT,                       //  I2C Data
        I2C_SCLK,                       //  I2C Clock
        ////////////////////  PS2  ///////////////////////////
        PS2_DAT,                        //  PS2 Data
        PS2_CLK,                        //  PS2 Clock
        ////////////////////  VGA  ///////////////////////////
        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]
        ////////////  Ethernet Interface  ////////////////////////
        ENET_DATA,                      //  DM9000A DATA bus 16Bits
        ENET_CMD,                       //  DM9000A Command/Data Select, 0 =
Command, 1 = Data
        ENET_CS_N,                      //  DM9000A Chip Select
        ENET_WR_N,                      //  DM9000A Write
        ENET_RD_N,                      //  DM9000A Read
```

```
        ENET_RST_N,                     //  DM9000A Reset
        ENET_INT,                       //  DM9000A Interrupt
        ENET_CLK,                       //  DM9000A Clock 25 MHz
        ////////////////    Audio CODEC     ////////////////////////
        AUD_ADCLRCK,                    //  Audio CODEC ADC LR Clock
        AUD_ADCDAT,                     //  Audio CODEC ADC Data
        AUD_DACLRCK,                    //  Audio CODEC DAC LR Clock
        AUD_DACDAT,                     //  Audio CODEC DAC Data
        AUD_BCLK,                       //  Audio CODEC Bit-Stream Clock
        AUD_XCK,                        //  Audio CODEC Chip Clock
        ////////////////    TV Decoder      ////////////////////////
        TD_DATA,                        //  TV Decoder Data bus 8 bits
        TD_HS,                          //  TV Decoder H_SYNC
        TD_VS,                          //  TV Decoder V_SYNC
        TD_RESET,                       //  TV Decoder Reset
        ////////////////////  GPIO      ////////////////////////////
        GPIO_0,                         //  GPIO Connection 0
        GPIO_1                          //  GPIO Connection 1

    );

//////////////////////    Clock Input         ////////////////////////
input           CLOCK_27;               //  27 MHz
input           CLOCK_50;               //  50 MHz
input           EXT_CLOCK;              //  External Clock
//////////////////////    Push Button         ////////////////////////
input[3:0]  KEY;                        //  Pushbutton[3:0]
//////////////////////    DPDT Switch         ////////////////////////
input[17:0]     SW;                     //  Toggle Switch[17:0]
//////////////////////    7-SEG Dispaly ////////////////////////
output      [6:0]   HEX0;               //  Seven Segment Digit 0
output      [6:0]   HEX1;               //  Seven Segment Digit 1
output      [6:0]   HEX2;               //  Seven Segment Digit 2
output      [6:0]   HEX3;               //  Seven Segment Digit 3
output      [6:0]   HEX4;               //  Seven Segment Digit 4
output      [6:0]   HEX5;               //  Seven Segment Digit 5
output      [6:0]   HEX6;               //  Seven Segment Digit 6
output      [6:0]   HEX7;               //  Seven Segment Digit 7
////////////////////////    LED     ////////////////////////////
output      [8:0]   LEDG;               //  LED Green[8:0]
output      [17:0]      LEDR;           //  LED Red[17:0]
////////////////////////    UART    ////////////////////////////
output          UART_TXD;               //  UART Transmitter
input           UART_RXD;               //  UART Receiver
```

```
///////////////////////////  IRDA  ///////////////////////////
output              IRDA_TXD;                   //  IRDA Transmitter
input               IRDA_RXD;                   //  IRDA Receiver
//////////////////////        SDRAM Interface   //////////////////////
inout[15:0]     DRAM_DQ;                    //  SDRAM Data bus 16 Bits
output  [11:0]      DRAM_ADDR;                      //  SDRAM Address bus 12 Bits
output              DRAM_LDQM;                      //  SDRAM Low-byte Data Mask
output              DRAM_UDQM;                      //  SDRAM High-byte Data Mask
output              DRAM_WE_N;                      //  SDRAM Write Enable
output              DRAM_CAS_N;                     //  SDRAM Column Address Strobe
output              DRAM_RAS_N;                     //  SDRAM Row Address Strobe
output              DRAM_CS_N;                  //  SDRAM Chip Select
output              DRAM_BA_0;                  //  SDRAM Bank Address 0
output              DRAM_BA_1;                  //  SDRAM Bank Address 0
output              DRAM_CLK;                   //  SDRAM Clock
output              DRAM_CKE;                   //  SDRAM Clock Enable
//////////////////////      Flash Interface //////////////////////
inout           [7:0]   FL_DQ;                      //  FLASH Data bus 8 Bits
output          [21:0]  FL_ADDR;                    //  FLASH Address bus 22 Bits
output              FL_WE_N;                    //  FLASH Write Enable
output              FL_RST_N;                   //  FLASH Reset
output              FL_OE_N;                    //  FLASH Output Enable
output              FL_CE_N;                    //  FLASH Chip Enable
//////////////////////      SRAM Interface      //////////////////////
inout[15:0]     SRAM_DQ;                    //  SRAM Data bus 16 Bits
output  [17:0]      SRAM_ADDR;                      //  SRAM Address bus 18 Bits
output              SRAM_UB_N;                  //  SRAM High-byte Data Mask
output              SRAM_LB_N;                  //  SRAM Low-byte Data Mask
output              SRAM_WE_N;                      //  SRAM Write Enable
output              SRAM_CE_N;                  //  SRAM Chip Enable
output              SRAM_OE_N;                  //  SRAM Output Enable
////////////////////  ISP1362 Interface   //////////////////////
inout[15:0]     OTG_DATA;                   //  ISP1362 Data bus 16 Bits
output  [1:0]   OTG_ADDR;                   //  ISP1362 Address 2 Bits
output              OTG_CS_N;                       //  ISP1362 Chip Select
output              OTG_RD_N;                       //  ISP1362 Write
output              OTG_WR_N;                       //  ISP1362 Read
output              OTG_RST_N;                  //  ISP1362 Reset
output              OTG_FSPEED;                     //  USB Full Speed,     0 = Enable, Z =
Disable
output              OTG_LSPEED;                     //  USB Low Speed,      0 = Enable, Z =
Disable
input               OTG_INT0;                   //  ISP1362 Interrupt 0
input               OTG_INT1;                   //  ISP1362 Interrupt 1
```

```
input           OTG_DREQ0;              //  ISP1362 DMA Request 0
input           OTG_DREQ1;              //  ISP1362 DMA Request 1
output              OTG_DACK0_N;            //  ISP1362 DMA Acknowledge 0
output              OTG_DACK1_N;            //  ISP1362 DMA Acknowledge 1
////////////////////  LCD Module 16X2  ////////////////////////////
inout[7:0] LCD_DATA;                //  LCD Data bus 8 bits
output              LCD_ON;                 //  LCD Power ON/OFF
output              LCD_BLON;               //  LCD Back Light ON/OFF
output              LCD_RW;                 //  LCD Read/Write Select, 0 = Write, 1 =
Read
output              LCD_EN;                 //  LCD Enable
output              LCD_RS;                 //  LCD Command/Data Select, 0 = Command,
1 = Data
////////////////////  SD Card Interface   ////////////////////////
inout           SD_DAT;                 //  SD Card Data
inout           SD_DAT3;                //  SD Card Data 3
inout           SD_CMD;                 //  SD Card Command Signal
output              SD_CLK;                 //  SD Card Clock
////////////////////////    I2C     ////////////////////////////////
inout           I2C_SDAT;               //  I2C Data
output              I2C_SCLK;               //  I2C Clock
////////////////////////    PS2     ////////////////////////////////
input           PS2_DAT;                //  PS2 Data
input           PS2_CLK;                //  PS2 Clock
////////////////////  USB JTAG link   ////////////////////////////
input               TDI;                // CPLD -> FPGA (data in)
input               TCK;                // CPLD -> FPGA (clk)
input               TCS;                // CPLD -> FPGA (CS)
output              TDO;                // FPGA -> CPLD (data out)
////////////////////////    VGA         ////////////////////////////
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]
////////////////    Ethernet Interface  ////////////////////////////
inout[15:0]     ENET_DATA;              //  DM9000A DATA bus 16Bits
output              ENET_CMD;               //  DM9000A Command/Data Select, 0 =
Command, 1 = Data
output              ENET_CS_N;              //  DM9000A Chip Select
output              ENET_WR_N;              //  DM9000A Write
```

```
output          ENET_RD_N;          //  DM9000A Read
output          ENET_RST_N;             //  DM9000A Reset
input       ENET_INT;           //  DM9000A Interrupt
output          ENET_CLK;           //  DM9000A Clock 25 MHz
////////////////////  Audio CODEC       ////////////////////////////
output/*inout*/ AUD_ADCLRCK;        //  Audio CODEC ADC LR Clock
input       AUD_ADCDAT;             //  Audio CODEC ADC Data
inout       AUD_DACLRCK;        //  Audio CODEC DAC LR Clock
output          AUD_DACDAT;             //  Audio CODEC DAC Data
inout       AUD_BCLK;           //  Audio CODEC Bit-Stream Clock
output          AUD_XCK;            //  Audio CODEC Chip Clock
////////////////////  TV Devoder        ////////////////////////////
input[7:0] TD_DATA;             //  TV Decoder Data bus 8 bits
input       TD_HS;              //  TV Decoder H_SYNC
input       TD_VS;              //  TV Decoder V_SYNC
output          TD_RESET;           //  TV Decoder Reset
////////////////////////  GPIO      ////////////////////////////////
inout[35:0]     GPIO_0;             //  GPIO Connection 0
inout[35:0]     GPIO_1;             //  GPIO Connection 1

//////////////////////////////////////////////////////////////////
////////////////////////////////////
//DLA state machine variables
wire reset;
reg [17:0] cpu_addr_reg; //memory address register for SRAM
reg [17:0] vga_addr_reg;
reg [15:0] data_reg; //memory data register    for SRAM
reg we_vga;
reg we_cpu ;            //write enable for SRAM
reg flag;


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

//      LCD ON
assign      LCD_ON      =   1'b0;
assign      LCD_BLON    =   1'b0;

//      All inout port turn to tri-state
assign      DRAM_DQ         =   16'hzzzz;
assign      FL_DQ           =   8'hzz;
assign      SRAM_DQ         =   16'hzzzz;
assign      OTG_DATA    =   16'hzzzz;
```

```
assign    SD_DAT      =    1'bz;
//assign  ENET_DATA   =    16'hzzzz;
assign    GPIO_0      =    36'hzzzzzzzzz;
assign    GPIO_1      =    36'hzzzzzzzzz;

//DM9000A Clock
reg                   ENET_CLK;
always@(posedge CPU_CLK) ENET_CLK=~ENET_CLK;

wire [31:0]    mSEG7_DIG;
reg  [31:0]    Cont;
wire           VGA_CTRL_CLK;
wire           AUD_CTRL_CLK;
wire [9:0] mVGA_R;
wire [9:0] mVGA_G;
wire [9:0] mVGA_B;
wire [19:0]    mVGA_ADDR;              //video memory address
wire [9:0]  Coord_X, Coord_Y;    //display coods
wire           DLY_RST;

assign    TD_RESET    =    1'b1; //    Allow 27 MHz input
//assign  AUD_ADCLRCK =    AUD_DACLRCK;
//assign  AUD_XCK     =    AUD_CTRL_CLK;

//assign  LEDR =   18'h0;

Reset_Delay             r0    (     .iCLK(CLOCK_50),.oRESET(DLY_RST)     );

VGA_Audio_PLL           p1    (
    .areset(~DLY_RST),.inclk0(CLOCK_27),.c0(VGA_CTRL_CLK),.c1(AUD_CTRL_CLK),.c2(V
GA_CLK)      );



/*****************************SDRAM
PLL*****************************************************/
wire CPU_CLK ;
sdrampll pll1(CLOCK_50, DRAM_CLK ,    CPU_CLK);

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

//*************************************************************************************
**
```

```
//*******************************CPU instantiation*************************************

wire [15:0] out_sram_data ;
wire [23:0] keyboardscancode , keyboardscancode1;
reg [23:0] keyscanreg ;
wire [7:0] out_flag;
wire [15:0] scoreofplayer ;



  TETRISCPU cpu1
    ( //DM9000A
      .ENET_CMD_from_the_DM9000A                (ENET_CMD),
      .ENET_CS_N_from_the_DM9000A               (ENET_CS_N),
      .ENET_DATA_to_and_from_the_DM9000A        (ENET_DATA),
      .ENET_INT_to_the_DM9000A                  (ENET_INT),
      .ENET_RD_N_from_the_DM9000A               (ENET_RD_N),
      .ENET_RST_N_from_the_DM9000A              (ENET_RST_N),
      .ENET_WR_N_from_the_DM9000A               (ENET_WR_N),

      .clk                                      (CPU_CLK),
        .reset_n                                (KEY[0]),

      .in_port_to_the_keyboardscancode          (keyscanreg),
      .in_port_to_the_keycounter                (count1),



        //VGA control

      .out_port_from_the_outputsramdata         (out_sram_data),

      .out_port_from_the_outputxcoord           (x_ptr),
      .out_port_from_the_outputycoord           (y_ptr),


      .out_port_from_the_pio_flag               (out_flag),
      .out_port_from_the_scoreofplayer          (scoreofplayer),



      .rxd_to_the_uart1                         (UART_RXD),
      .txd_from_the_uart1                       (UART_TXD),
```

```
        .zs_addr_from_the_sdram         (DRAM_ADDR),
        .zs_ba_from_the_sdram           ({DRAM_BA_1, DRAM_BA_0}),
        .zs_cas_n_from_the_sdram        (DRAM_CAS_N),
        .zs_cke_from_the_sdram          (DRAM_CKE),
        .zs_cs_n_from_the_sdram         (DRAM_CS_N),
        .zs_dq_to_and_from_the_sdram    (DRAM_DQ),
        .zs_dqm_from_the_sdram          ({DRAM_UDQM, DRAM_LDQM}),
        .zs_ras_n_from_the_sdram        (DRAM_RAS_N),
        .zs_we_n_from_the_sdram         (DRAM_WE_N)
    );


    // assign LEDR[9:0] =    row161514write[9:0] ;

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


VGA_Controller          u1  (
                        //  Host Side
                                .iCursor_RGB_EN(4'b0111),

                                .oAddress(mVGA_ADDR),

                                .oCoord_X(Coord_X),
                                .oCoord_Y(Coord_Y),

                                .iRed(mVGA_R),
                                .iGreen(mVGA_G),
                                .iBlue(mVGA_B),
                                .iFlag(out_flag[7:0]),
                        //      .iFlag1(out_flag[1]|SW[1]),

                                //  VGA Side
                                .oVGA_R(VGA_R),
                                .oVGA_G(VGA_G),
                                .oVGA_B(VGA_B),
                                .oVGA_H_SYNC(VGA_HS),
                                .oVGA_V_SYNC(VGA_VS),
                                .oVGA_SYNC(VGA_SYNC),
                                .oVGA_BLANK(VGA_BLANK),

                                //  Control Signal
```

```
                                .iCLK(ENET_CLK),
                                .iRST_N(DLY_RST),

                                .iscoreofplayer (scoreofplayer) );




wire [9:0]    x_ptr,y_ptr;        //nios inputs to the sram

reg    [8:0] counter ;        //for the
reg [3:0] counter1 ;        //keeps track of the number of rows of the character drawn
reg [15:0] counter3;

reg vga_addr_flag;

wire [7:0] romout ;


//assign LEDR[7:0] = romout[7:0] ;
//assign LEDR[7:0] = counter3 ;

// SRAM_control
assign SRAM_ADDR = (vga_addr_flag) ? vga_addr_reg:cpu_addr_reg;
assign SRAM_DQ = (we_vga | we_cpu)? 16'hzzzz : data_reg ;
assign SRAM_UB_N = 0;                                    // hi byte select enabled
assign SRAM_LB_N = 0;                            // lo byte select enabled
assign SRAM_CE_N = 0;                            // chip is enabled
assign SRAM_WE_N = (we_vga | we_cpu);                                    // write when ZERO
assign SRAM_OE_N = 0;                            //output enable is overidden by WE



// Show SRAM on the VGA
//assign    mVGA_R = {(Coord_X[0]?SRAM_DQ[15:14]:SRAM_DQ[7:6]), 8'b0} ;
//assign    mVGA_G = {(Coord_X[0]?SRAM_DQ[13:10]:SRAM_DQ[5:2]), 6'b0} ;
//assign    mVGA_B = {(Coord_X[0]?SRAM_DQ[9:8]:SRAM_DQ[1:0]), 8'b0} ;

assign    mVGA_R = {SRAM_DQ[11:8],6'b000000};
assign    mVGA_G = {SRAM_DQ[7:4],6'b000000};
assign    mVGA_B = {SRAM_DQ[3:0],6'b000000};
```

```
// DLA state machine
assign reset = ~KEY[0];
//assign LEDG = led;


reg [11:0]trial[2:0];




// VGA Cycles
always @ (posedge ENET_CLK )
begin
    trial[0] = 12'hcba;
      trial[1] = 4'b1011;
      trial[2] = 4'hc;

    if (VGA_VS & VGA_HS)
      begin
      //READ
      vga_addr_flag <=1;
      flag <=0;
      vga_addr_reg <= {Coord_X[9:1],Coord_Y[8:0]} ;
      we_vga <= 1'b1;
      end
      //Write when syncing
      else
      begin
      vga_addr_flag <=0;
      flag <=1;
      we_vga <= 1'b0;
      end

end


//CPU Cycles
always@(posedge CPU_CLK)
begin
```

```
 if(flag)
    begin
     if (reset)          //synch reset assumes KEY0 is held down 1/60 second
       begin
            //clear the screen
            cpu_addr_reg <= {Coord_X[9:1],Coord_Y[8:0]} ;   // [17:0]
            we_cpu <= 1'b0;                                          //write some memory
            data_reg <= 16'b0;                                 //write all zeros (black)
            counter <= 9'o330 ;
            counter1 <= 4'b0000 ;
       end
       else
       begin
            cpu_addr_reg <= {x_ptr[9:1],y_ptr[8:0]};        // [17:0]
            we_cpu <= 1'b0;                                                 //write some memory
            data_reg <= out_sram_data;
       end
      end
      end

//*************************************************************************************
************************
//*****************************Keyboardfiles. Source cited in the
report************************************
wire reset1 = 1'b0;
wire [7:0] scan_code;

reg [7:0] history[1:4];
wire read, scan_ready;

oneshot pulser(
    .pulse_out(read),
    .trigger_in(scan_ready),
    .clk(CLOCK_50)
);

keyboard kbd(
   .keyboard_clk(PS2_CLK),
   .keyboard_data(PS2_DAT),
   .clock50(CLOCK_50),
   .reset(reset1),
   .read(read),
   .scan_ready(scan_ready),
   .scan_code(scan_code)
```

```
);



assign keyboardscancode    = {history[3] , history[2] , history[1]};

always @(posedge scan_ready or posedge reset)
begin
       if ( reset == 1'b1)
       begin
         history[4] <= 8'd0 ;
       history[3] <= 8'd0 ;
       history[2] <= 8'd0 ;
       history[1] <= 8'd0 ;
       end
       else
       begin
       history[4] <= history[3];
       history[3] <= history[2];
       history[2] <= history[1];
       history[1] <= scan_code;
       end
end

reg [2:0]    count ;
reg [15:0] count1 ;    //counts number of keypresses


always @ (posedge scan_ready or posedge reset)
begin
              if (reset == 1'b1)
              begin
              count    <=    3'd0      ;
              count1 <=    16'd0    ;
              end
              else if ( count == 3'd2)
              begin
              count1 <= count1 + 1 ;
              count <= 3'd0 ;
              end
              else
              begin
              count <= count + 1 ;
              end
```

```
end



always @ (count1 or    reset)
begin
            if (reset == 1'b1)
            begin
            keyscanreg    <=    24'd0      ;
            end
            else
            begin
            keyscanreg    <= keyboardscancode ;
            end
end

assign LEDG[2:0]    = count ;

//assign LEDR[7:0]    = count1 ;
assign LEDR[5:0] = count1;




hex_7seg dsp0(keyscanreg [3:0],HEX0);
hex_7seg dsp1(keyscanreg [7:4],HEX1);

hex_7seg dsp2(keyscanreg [11:8],HEX2);
hex_7seg dsp3(keyscanreg [15:12],HEX3);

hex_7seg dsp4(keyscanreg [19:16],HEX4);
hex_7seg dsp5(keyscanreg [23:20],HEX5);

hex_7seg dsp6(history[4][3:0],HEX6);
hex_7seg dsp7(history[4][7:4],HEX7);


endmodule //top module
```