3D Pottery Game

CSEE 4840 Embedded System Design
Final Report

Hao Jin (hj2354)
Chengxue Qian (cq2159)
Xiaowen Han (xh2204)
Muqing Liu (ml3466)

May 14th, 2014

Content

Introduction

In this project, we are going to implement a recreation application using FPGA board in which the users can create a pottery on the screen.

We have two playing modes: freestyle and playing style. In freestyle, you are going to create a pottery whatever you like and there's no limitation on your creation. In playing style, you will first choose a pottery pattern you like by rolling the mouse middle button and then you should try your best to create a pottery as similar to the reference pottery as possible. If you finish creating the pottery, you can press the middle button of the mouse and there will be words on the screen indication whether you win or you should keep doing. There is a welcome page in which you can choose a pottery shape or a freestyle mode. On the making pottery page, the pottery is rotating continuously and the user can use a mouse to change the shape of the pottery. Right click or drag is to shrink the pottery and left click or drag is to enlarge the pottery. Dragging will change the shape more efficiently than clicking. There is a red triangle on the edge of the pottery indicating the current position of the cursor, which is also the position where the user can change the shape of the pottery.

The project consists two parts: software and hardware. The software part will keep track of the position of the mouse and the mouse states. Since our pottery is made of ellipses of different sizes, so the software part will do curve fitting of the control points and calculate the size and color of the ellipses at real time and send the radius messages and the RGB values to the hardware. The hardware part is responsible for drawing ellipses with the information from software and checks the overlap relationship to display the pottery on the screen.

System Overview

A high-level system overview of our design is shown in Figure 1. We are using two different clocks, one is 50 MHz and the other is 25 MHz, which are generated from the Phased Locked Loop (PLL) block. The system is updating the 640*480 screen at 60 frames per second.

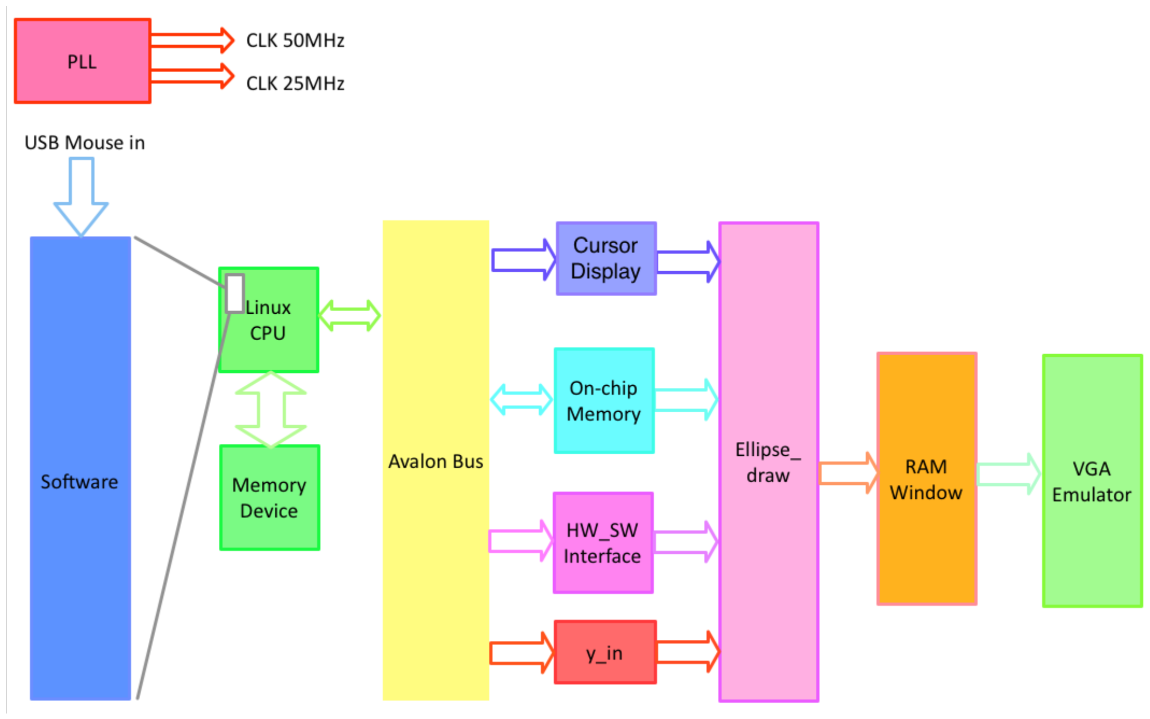

Figure 1. System Architecture

1. Hardware overview

The overall hardware part of our system is shown in Figure 2.

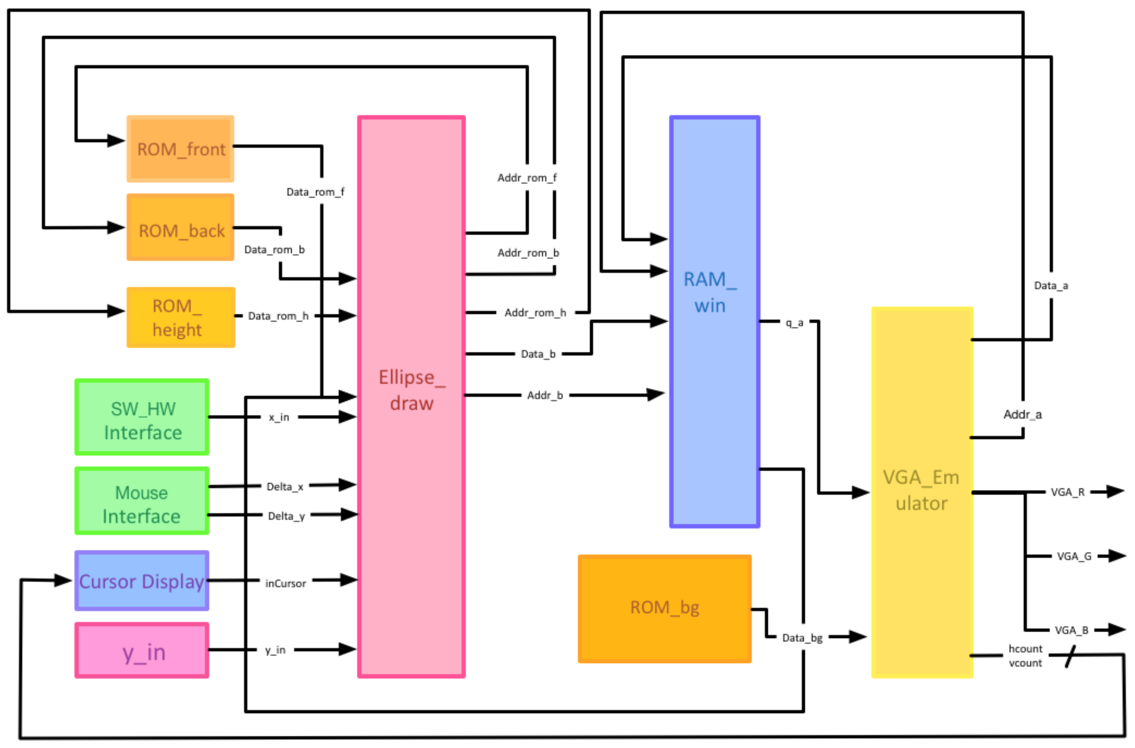

Figure 2. Top-level view of the hardware system

The hardware system mainly contains 7 parts, which can be divided into 4 categories: the interface modules which perform the interaction and communication between the software and hardware, and between the I/O device and software; the memory part including 4 ROMs containing the ellipse grayscale and ellipse height data calculated by the software and the background information and a RAM, which is the memory window containing the position information of the ellipses being drawn and it is the data being displayed on the screen; the execution part is the ellipse draw module, in which we draw ellipses in the hardware; and the display part includes a VGA Emulator and cursor display, and VGA Emulator is responsible for displaying the drawn ellipses on the screen and cursor display will draw a triangle on the screen indicating the position of the cursor where the user is changing the shape of the pottery.

Most of the modules are using the 50 MHz clock, only the VGA display part is using the 25 MHz clock.

2. Software overview

The software overview is discussed mainly from the main function side in "hello_with_datamouse.c". Most of the important variables (control points array; mode number; discrete points along the outline curve, etc) are set as global variables. Aside from the main function, there is a struct defined for 140 discrete points and two important functions called “getY()” and “changeY()” to get or to update points’ value using mainly the curve fitting algorithm we will discuss later.

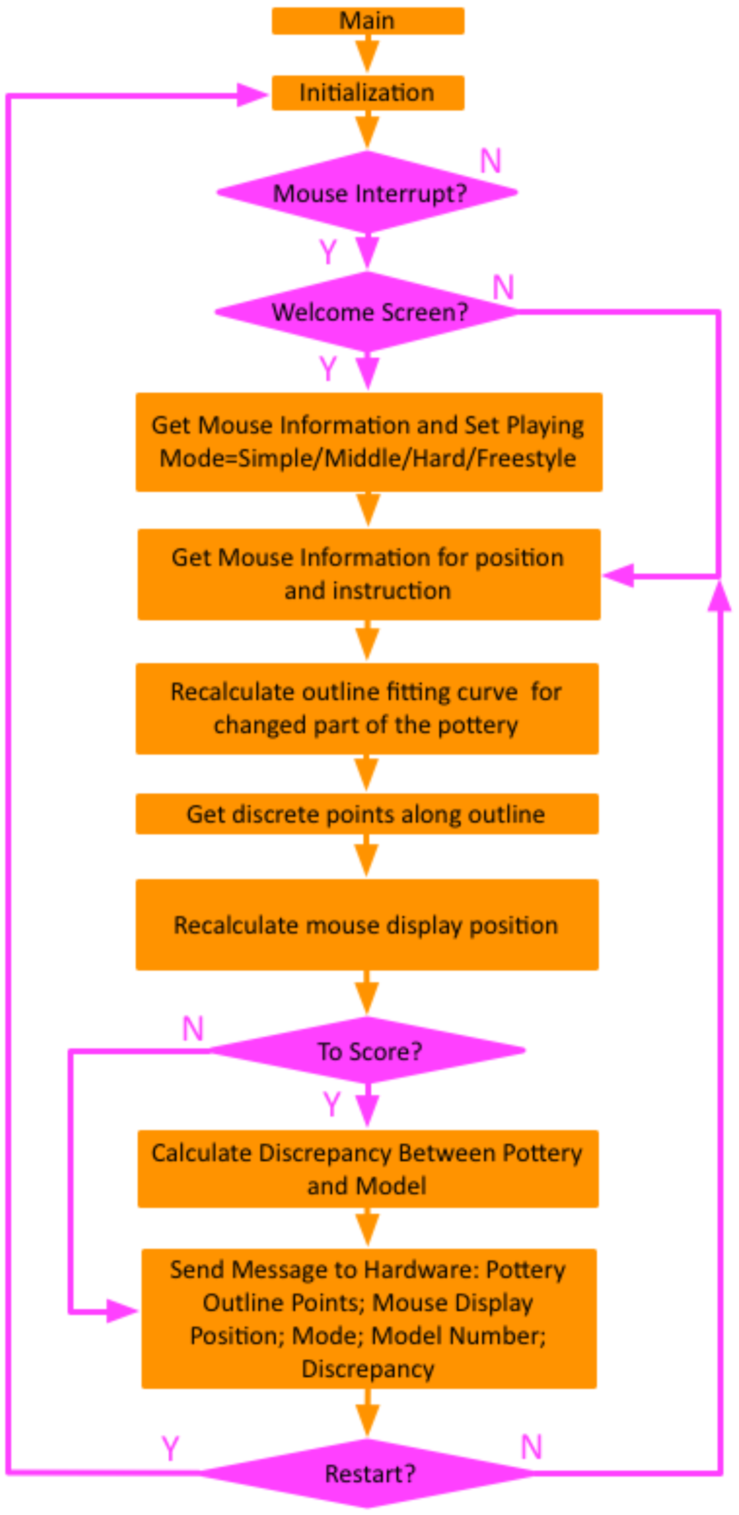
Main
Initialization
Mouse Interrupt?
N
Y
Welcome Screen?
N
Y
Get Mouse Information and Set Playing Mode=Simple/Middle/Hard/Freestyle
Get Mouse Information for position and instruction
Recalculate outline fitting curve for changed part of the pottery
Get discrete points along outline
Recalculate mouse display position
N
To Score?
Y
Calculate Discrepancy Between Pottery and Model
Send Message to Hardware: Pottery Outline Points; Mouse Display Position; Mode; Model Number; Discrepancy
Y
Restart?
N

Figure 3. Software flowchart for main function

In the software main function, we continuously look for any mouse interrupt whose information is used to calculate/update pottery outline curve fitted and mouse display position etc. The mode transfer status is showed in Figure.4 state-transition diagram of game mode.

If it is in the "WELCOME screen" mode, we set the playing modes first.

There are four choices in the welcome screen: easy pottery making; middle-difficulty pottery making; hard-mode pottery making, which corresponds to three difficulty level target pottery models respectively; and there is a fourth choice of freestyle pottery making where you can make a pottery without any target model that is used to give a score to the pottery you have made.

If it is in the "SIMPLE/MIDDLE/HARD mode" or it is in the "FREESTYLE mode", we go straight forwards to calculate, or to be more accurate, to update a series of discrete points on the current newly fitted outline curve. These discrete points are calculated using third-order uniform B-spline curve fitting algorithm, which will be discussed in detail later. The points are maintained in a 140-element array and are sent to hardware together with other information. Note that while one user is making the pottery, every time, we just recalculate the modified part along the pottery outline for speed optimization. It is possible because of the B-spline curve fitting algorithm we use, which will be further discussed later in the software part of the report.

Additionally, if the mouse information indicates "TO SCORE", we calculate the discrepancy between the current pottery and the target pottery model. Then the score is given to determine whether you win or you should keep doing based on the discrepancy parameter.

Finally, we send all information needed (pottery outline points; mouse display position; mode number; target model number; discrepancy or score) in several messages one by one to the hardware and let the hardware do screen displaying based on these information.

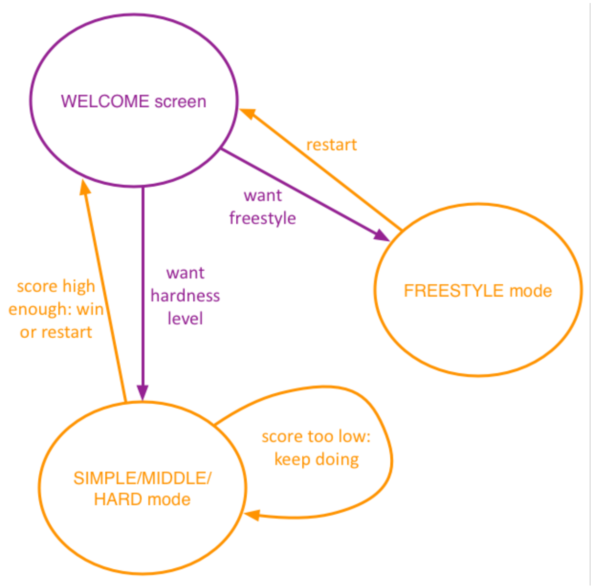

Figure 4. State-Transition Diagram of Game Mode

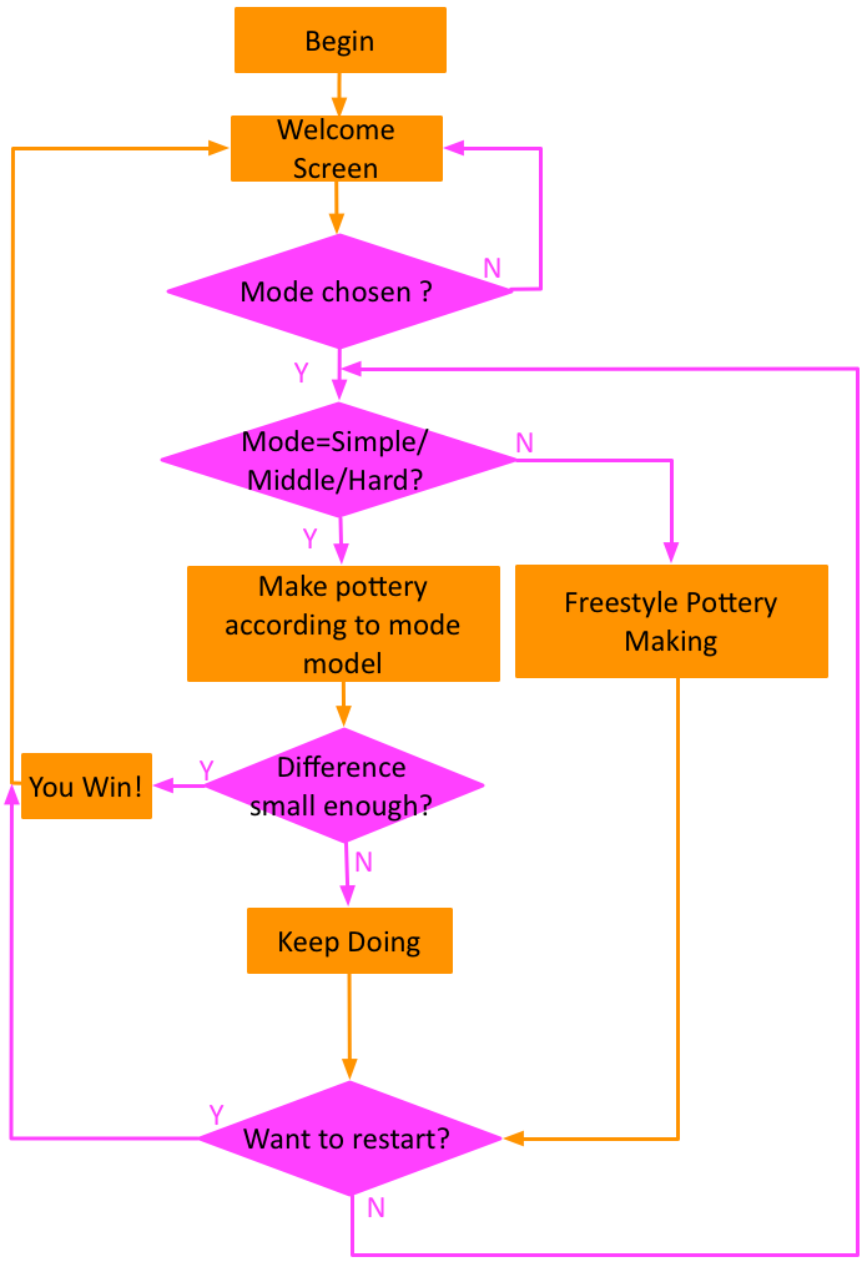

Figure 5. Flowchart of game playing

Here are the steps one user may go through to play the game as is shown in Figure.5:
Step1: First you have to choose from three hard level models; or you can do freestyle as well on the WELCOME screen;
Step2: Next you will enter the game interface where you can begin to play around with your original pottery with or without a target pottery model according to your previous choice;
Step3: (If there exists a target pottery model) When feel satisfied, just click to see if you win or more efforts should be made. If you win, the game will automatically redirect you to the WELCOME screen again.
Note that you can also restart all over, which will directed you to the WELCOME screen .

Hardware overview

There are 7 main blocks in hardware including memories (storage roms, ram window), interfaces between hardware and software (radius, cursor position), ellipse drawing and VGA emulator.

1. Interfaces

The interfaces in hardware received 10 messages from software. Message0-7 is the radius of the ellipse and message8-9 is the cursor positions from the mouse.

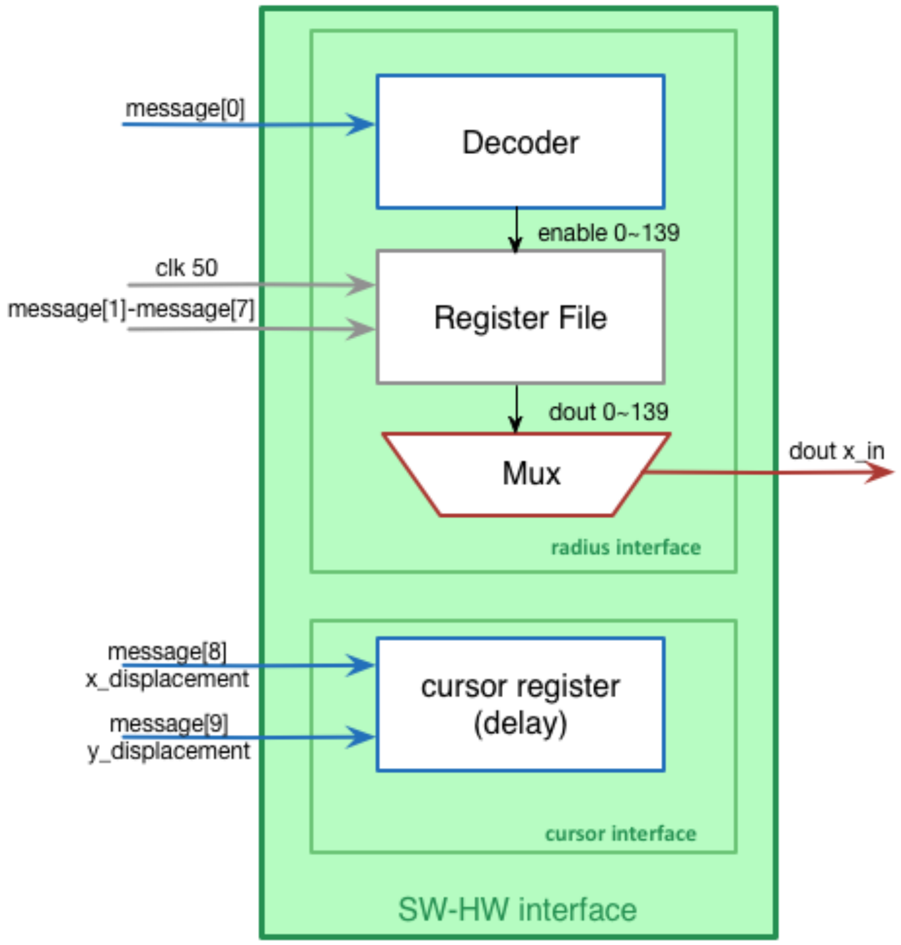

Figure 6. SW-HW Interface Block

Message[0]: encoded address of the sending radiuses(7of140), input of the decoder.
Message[1]-Message[7]: 7 radiuses calculated by curve fitting, input of the radiuses register.
Message[8]-Message[9]: x and y displacement of the cursor, scaled properly by the software, delayed by cursor registers then input of the cursor display.

2. RAM window

2.1 Implementation

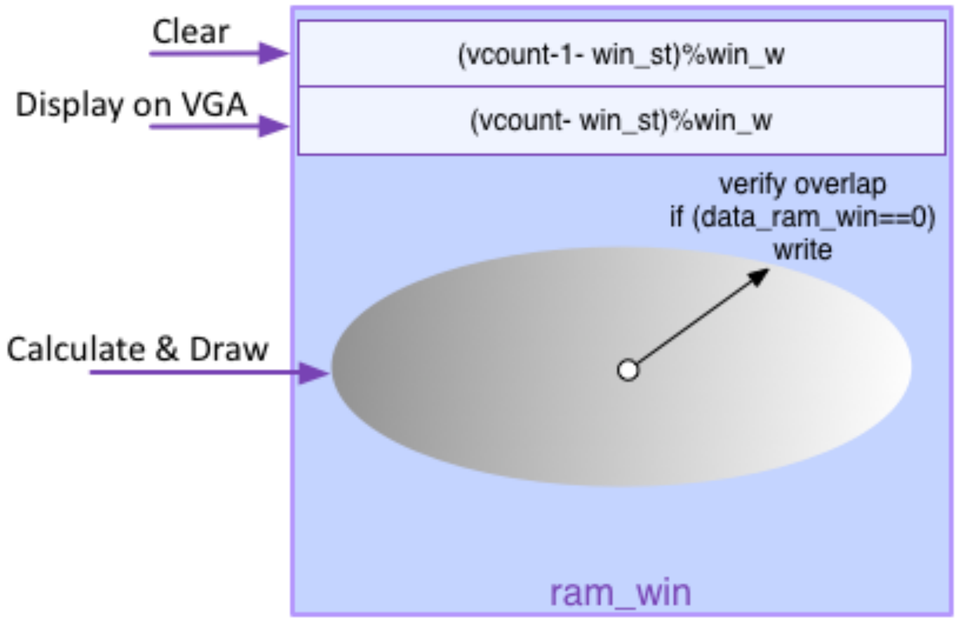

Figure 7. RAM_window block

The main idea of the RAM window is to use as few FPGA resources as possible. The memory only needs to store the information in the range of a small window, and we will continuously reuse the window to draw the whole pottery on the screen. The detailed implementation is shown in Figure 4. The second line in the window is the line being displayed on the VGA screen, and when we are displaying its content, we will erase the line above it, and the rest of the window is used to draw ellipse. If we reach the end of the window, we will write on top of the window, so the window will wrap around. In this way, we can reuse only one window and draw ellipses continuously. We are using a two-port RAM.

Port A: Display and clear

```
  else if (VGA_CLK==0&&enable==1&&vcount==65)begin
        rden_a<=0;
        wren_a<=1;
        address_a <= ((win_h-1)%win_h)*win_w + (hcount[10:1]-120);
        data_a<=0;
end
```

Port A is used to display the data in the ram window on VGA and clear the previous data line. When the data line A is displayed on screen, data line (A-1) is being written value ‘0’.

Port B: Write data and determine overlap

```
    cent_y =(y_in-65);
      if (four==2'd0) begin
        win_y = (cent_y + data_rom_h+ecc_count)% win_h;
        win_x = win_w/2 + x_count;
        addr_win = win_y*win_w + win_x;
      end
```

The way of getting the position of which line is being displayed and which line is being erased is by taking the mod: (vcount-win_st)%win_w, (vcount-1-win_st)%win_w, and the address is address_a <= ((win_h-1)%win_h)*win_w + (hcount[10:1]-120). Win_st is the start position of our window and win_w is the window width.

2.2 Timing Analysis

The timing diagram of this process is shown in Figure 5.

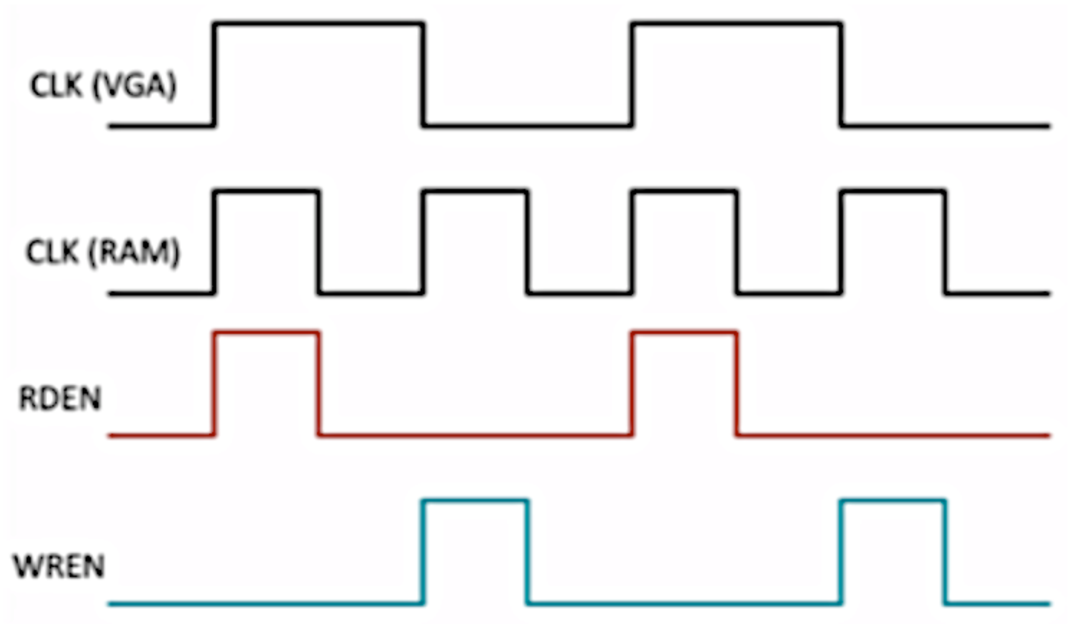

Figure 8. Timing diagram of the RAM_window block

VGA display uses the 25 MHz clock and our RAM uses the 50 MHz clock. We are using a two-port RAM. In the first cycle of the RAM clock, the RAM gets a read enable signal and port_a is going to read the data from RAM window and display it on the screen, and port_b is going to read the data from RAM window, and check the overlap information. If the pixel going to be drawn has already been colored, it will not be drawn and if not, it is going to be drawn. These checks are done in the first cycle. In the second cycle of the RAM clock, the RAM is write enabled and port_a will write zero to the first line of the window, which is the erasing of the first line. Port_b is going to be written and draw ellipse.

3. Ellipse draw block

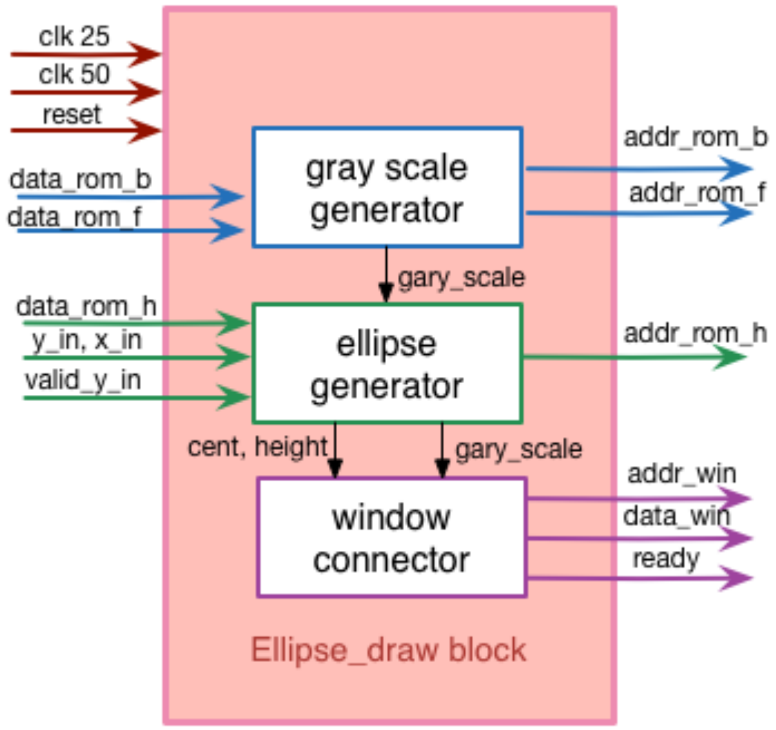

Figure 9. Ellipse_draw module

Ellipse draw block is the core block of our hardware design part. It include the gray scale generator which gives the pottery its color according to the radius; the ellipse generator which draw 140 different y position ellipses group, and the window connector to connect the drawing canvas to the ram window.

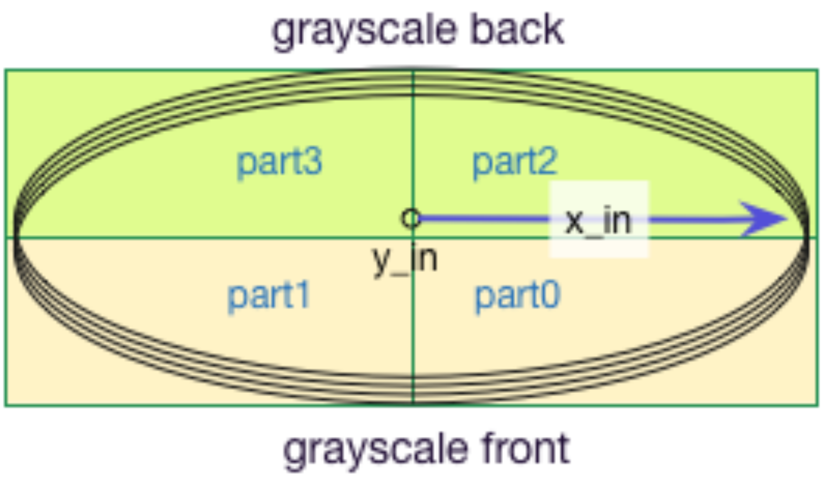

Figure 10. Gray scale and ellipse generator

3.1 Gray scale generator

```
addr_rom_h =    (2+rad)*(rad-1)/2 + x_count;
    if (four==2'd0)
        addr_rom_f = (rad+1)*(rad-1)+ (rad + x_count[7:0]);
else if (four==2'd1)
        addr_rom_f = (rad+1)*(rad-1)+ ( rad - x_count[7:0]);
else if (four==2'd2&&y_in<=y_in_st+'d20)
        addr_rom_b = (rad+1)*(rad-1)+rad - x_count[7:0];
else if (four==2'd2&&y_in>y_in_st+'d20)
        addr_rom_b = (rad+1)*(rad-1)+rad + x_count[7:0];
else if (four==2'd3&&y_in<=y_in_st+'d20)
        addr_rom_b = (rad+1)*(rad-1)+rad + x_count[7:0];
else if (four==2'd3&&y_in>y_in_st+'d20)
        addr_rom_b = (rad+1)*(rad-1)+rad - x_count[7:0];
```

The gray scale of the pottery is stored in the gray sacle rom. In order to have the 3-D effect, we use different gray scale for the front and back ellipse. The back ellipse group (part2 and part3) is darker than the front ellipse. The data stored in the rom according to different radius, the address of the front and back rom is (rad+1)*(rad-1)+rad-xcount.

3.2 Ellipse generator

In order to save the time to calculation the ellipse, we just calculate only one forth of the total ellipse and mirror the other 3 part. In this way, we could finish the calculation before VGA display that part. The order we draw the ellipse is first front and second back because the front part will overlap the back part of the other ellipse.

3.3 Window connector

When we draw ellipses, we have to connect it with the ellipses drawn in the memory window, so the window connector is a block from which we can transform the location of the ellipses on the screen to the address of the ellipses drawn in the memory window.

To determine the ellipse center position in the window, we will take the mod: (cent_y + data_rom_h+ecc_count)% win_h.

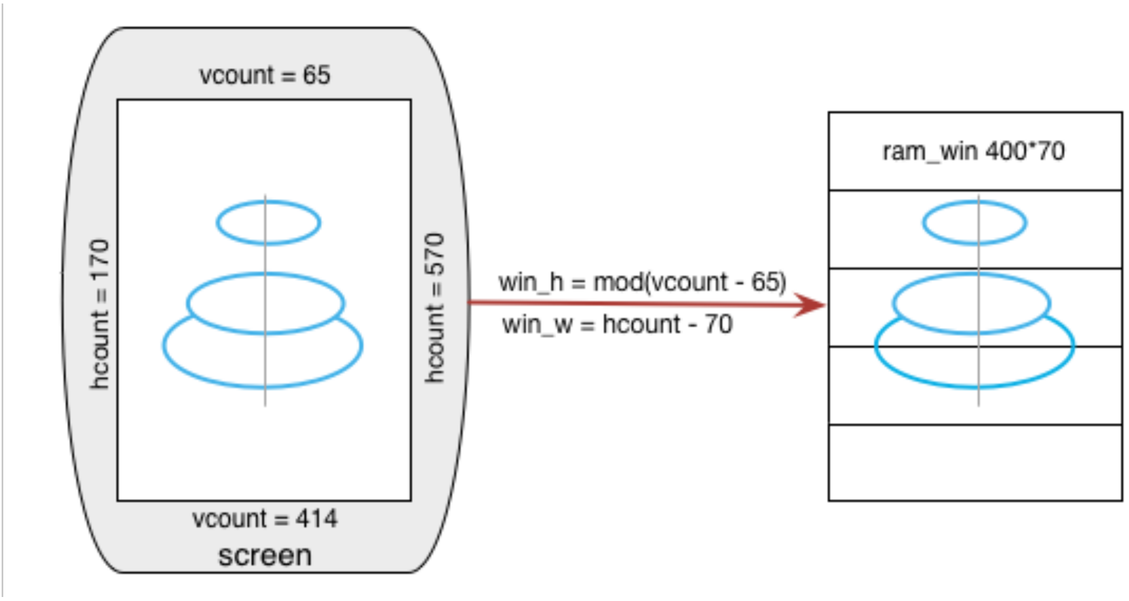

Figure 11. Window connector

To synchronize the ellipse drawing and window movement, we have to determine whether the cent of the ellipse is in the window and leave some time(window lines) to draw it.

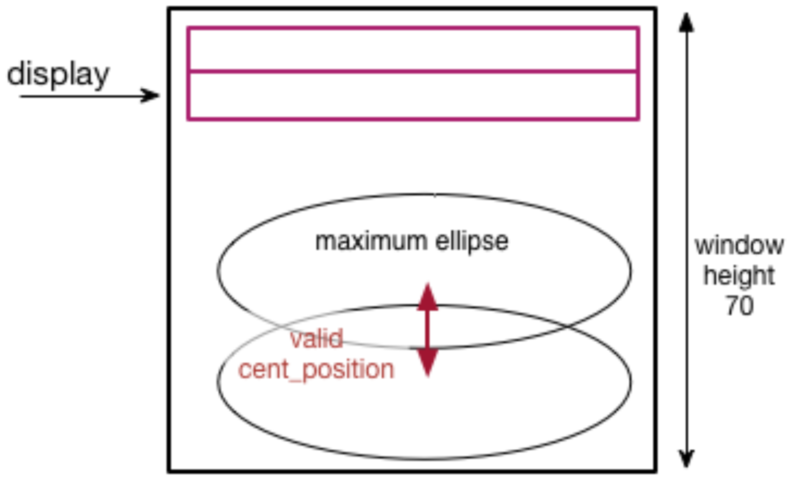

Figure 12. Valid range of cent_position

```
cent_y =(y_in-65);
    if (four==2'd0) begin
         win_y = (cent_y + data_rom_h+ecc_count)% win_h;
         win_x = win_w/2 + x_count;
         addr_win = win_y*win_w + win_x;
    end
    else if (four==2'd1) begin
         win_y = (cent_y + data_rom_h+ecc_count)% win_h;
         win_x = win_w/2 - x_count;
         addr_win = win_y*win_w + win_x;
    end
    else if (four==2'd2) begin
    win_y = (cent_y - data_rom_h+ecc_count)% win_h;
           win_x = win_w/2 + x_count;
         addr_win = win_y*win_w + win_x;
    end
    else if (four==2'd3) begin
         win_y = (cent_y - data_rom_h+ecc_count)% win_h;
         win_x = win_w/2 - x_count;
         addr_win = win_y*win_w + win_x;
    end
   else begin
         win_y=win_y;
         win_x=win_x;
         addr_win=addr_win;
   end
```

4. VGA emulator

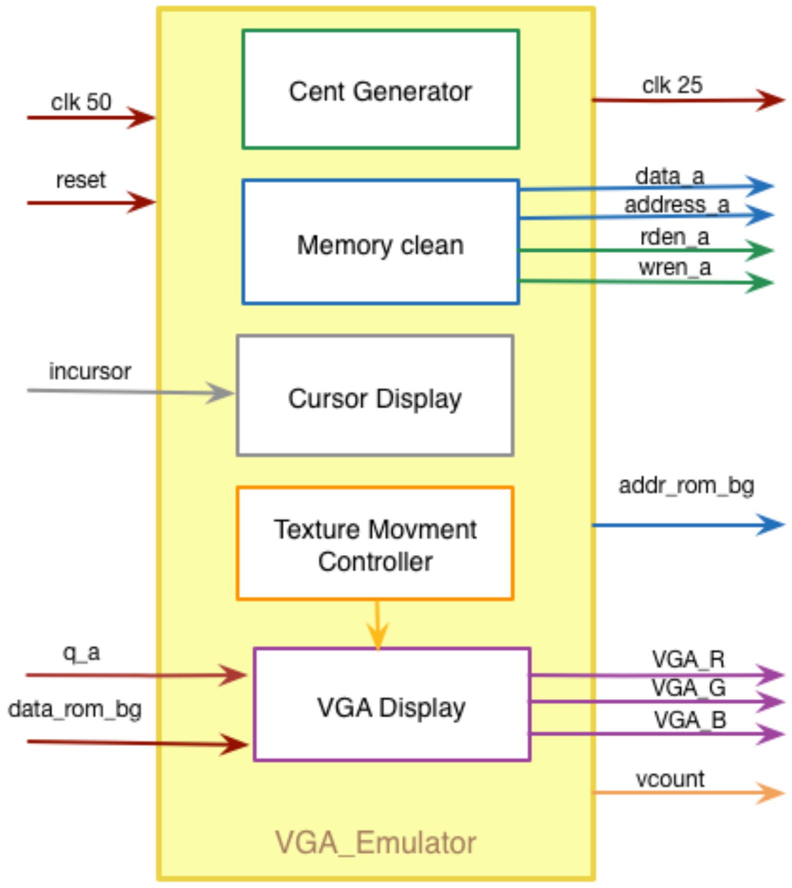

Figure 13. VGA Emulator Block

VGA emulator is responsible for all the VGA display, including the generator of the cent of the ellipse being drawn, ellipse drawing, memory cleaning, cursor displaying, texture movement controller and the final VGA display on the screen.

4.1 Cent generator and memory clean

The idea of the cent generator and memory clean is to determine the position range of the ellipse center being drawn in the window. Drawing ellipses has delay, so our scheme of reusing the memory window requires some gap lines between the line being displayed and the area where the ellipses can be drawn, otherwise the ellipses may not have enough time to finish before they are displayed on the screen if they are big in size.

```
   logic inwin;

   assign
inwin=(hcount[10:1]>=120&&hcount[10:1]<520&&vcount>=65&&vcount<=414);

   always@* begin
            if (inwin) begin
              enable = 1;
            // address_a = ((vcount-10)%win_h)*win_w + (hcount[10:1]-170);
                end
              else enable = 0;
        end

       always_ff@ (negedge clk50)begin
        if (reset)begin
             rden_a <=0;
             wren_a <=0;
             address_a<=0;
             end
        else if (VGA_CLK==1&&enable==1)begin
             rden_a<=1;
             wren_a<=0;
             address_a <= ((vcount-65)%win_h)*win_w + (hcount[10:1]-120);
        end
        else if (VGA_CLK==0&&enable==1&&vcount==65)begin
            rden_a<=0;
             wren_a<=1;
             address_a <= ((win_h-1)%win_h)*win_w + (hcount[10:1]-120);
             data_a<=0;
        end
        else if (VGA_CLK==0&&enable==1&&vcount!=65)begin
            rden_a<=0;
             wren_a<=1;
             address_a <= ((vcount-66)%win_h)*win_w + (hcount[10:1]-120);
             data_a<=0;
        end
        else begin
            rden_a<=0;
             wren_a<=0;
       end
       end
```

We first have to determine whether hcount and vcount are in the window that we can draw ellipses and then calculate the address using address_a <= ((vcount-66)%win_h)*win_w + (hcount[10:1]-120). While if it is the first line in the window, there is a little difference in the address: address_a <= ((vcount-65)%win_h)*win_w + (hcount[10:1]-120). By this way, we can continuously drawing ellipse on the screen.

4.2 Cursor display

We are using a little triangle to represent the cursor. The cursor is on the edge of the pottery and will move up and down with the mouse. The position information that determines the position of the cursor receives from the software part.

4.3 Texture movement controller

Since our pottery is central symmetry, it will have no difference if it rotates around its central line. So we decided to use a constantly moving highlight line to represent the rotation of the pottery.

We include a counter in our scheme and compare the grayscale of the pottery with a reference value, which keeps changing, and the changing speed is determined by the counter_hl. If the grayscale of the pottery equals to the reference value, it will be highlighted and if it doesn’t, it will keep its original value. The comparing range of the grayscale is from 20 to 220. By this way, the highlight line will keeps moving from one side of the pottery to the other and have a moving effect.

```
    logic[19:0] counter_hl ;
    logic[8:0]    gray_hl;

  always@(posedge VGA_CLK)begin
        if (reset) begin
        counter_hl<=20'h0;
        gray_hl<=8'd220;
        end
        else if (counter_hl<20'd8191&&gray_hl>8'd20)begin
        counter_hl<=counter_hl+'d1;
        gray_hl<=gray_hl;
        end
        else if (counter_hl<20'd8191&&gray_hl<=8'd20)begin
        counter_hl<=counter_hl+'d1;
        gray_hl<=gray_hl;
        end
        else if (counter_hl==20'd8191&&gray_hl>8'd20)begin
        counter_hl<=0;
        gray_hl<=gray_hl-8'd2;
        end
        else if (counter_hl==20'd8191&&gray_hl<=8'd20)begin
        counter_hl<=0;
        gray_hl<=8'd220;
        end
        else begin
        counter_hl<=counter_hl;
        gray_hl<=gray_hl;
        end
    end
```

4.4 VGA display

There are two main operating windows in our application: welcome window and the playing window.

The welcome window displays the four example potteries as well as the mode selection circles. If the mode is selected, it will be highlighted. The displaying part will receive the information from software indicating which example is selected and then the center position of the corresponding selection circle is determined. We can draw a circle according to that center position.

The playing window displays the rotating pottery. When we display on the screen, the cursor has the highest priority and will display on top of others; then comes to the pottery; the background has the lowest priority to display.

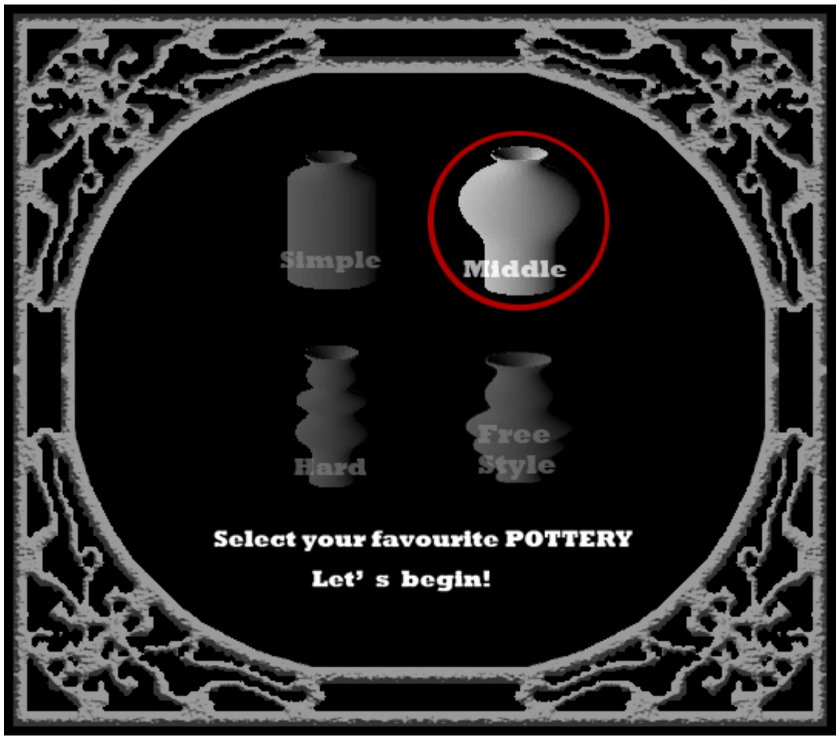

Figure 14. Welcome window

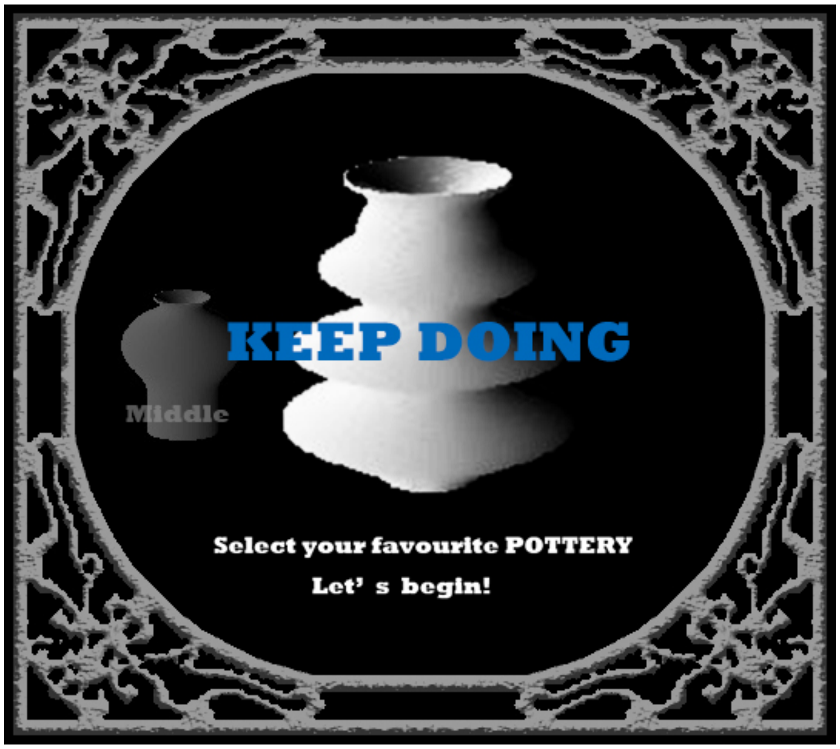

Figure 15. Playing window

Software Overview

First we discuss it from the algorithm aspect which is early tested and simulated in MATLAB then is converted to c (Here we use MATLAB code to illustrate our original idea; however, a lot have been modified and further optimized when converting to c and when collaborating with the hardware or interface parts of the project). The software modeling of the project can be mainly split into three parts in the following discussion.

1. Curve-fitting;
2. 3D modeling;
3. Recasting;

In the fourth part, we will discuss modification and optimization whereas in the fifth part, the score rules are shown. For the sixth part, we will briefly discuss the mouse functions. Last part shows some ROM size numbers.

1. Curve-fitting according to the input normalized control points and maximum radius preferred

By default, we maintain 17 control points. Using third-order uniform B-spline curve fitting, we obtain one B-spline from four adjacent points as is showed in Figure 16, which corresponds to a total of 17-3=14 B-splines. (B-spline, or Basis Spline, is a spline function that has minimal support with respect to a given degree, smoothness, and domain partition. It is used for curve fitting. Each third-order B-Spline will exactly fit four points.) Then, we make 9 uniform partitions between every two adjacent B-splines to split the interval into 10 parts. That is to say, we maintain 14*10=140 points to represent the curve-fitted. Every point involves at most 8 multiplications. (Note that the results of 10 sets of b0~b3 can be pre-determined) If the number of control points is fixed, then it only needs 4 multiplications at most to form y. To sum up, we need 140*8=1120 multiplications at most under unfixed number of control points assumption.

```
        for i=1:B;                                   %construct each B spline
        for  u=0:0.111:0.999;                            %uniform  partition
    between B spline
```

```
b0=1.0./6.*(1-u).^3;                                                  %construct base functions b0~b3
b1=1.0./6.*(3.*u.^3-6.*u.^2+4);
b2=1.0./6.*(-3.*u.^3+3.*u.^2+3.*u+1);
b3=1.0./6.*u.^3;
x(1,j)=b0.*a(1,i)+b1.*a(1,i+1)+b2.*a(1,i+2)+b3.*a(1,i+3);    %discrete height values
y(1,j)=b0.*a(2,i)+b1.*a(2,i+1)+b2.*a(2,i+2)+b3.*a(2,i+3);%discrete radial distance values
```

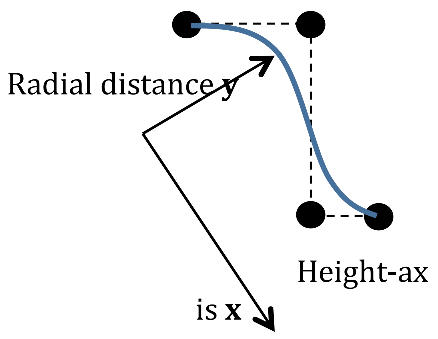

Figure 16. Third-order B-Spline Curve Fitting

2. Construct 3D jar model using cylindrical coordinate system

Radial distance is denoted by one element in row vector y(1, j). Azimuth is denoted by one element in column vector t0. Height is denoted by one element in row vector x(1,j). We turn around the fitted curve on height-axis to form a jar. By default, we use 140 points to represent a curve. Along the rotating path, we maintain 96 samples each with an increasing azimuth interval of pi/48. In total we need 96*140*2=26880 multiplications at most and 96*2 cos & sin (The results of sin and cos can be pre-determined.)

```
t0=pi/2:pi/48:2.5*pi; %discrete azimuth sample values 3d model values for azimuth
x0=ecc*y(1,j)*cos(t0)+x(1,j);    %form 3d model values for height
y0=y(1,j)*sin(t0);        %form 3d model values for radial distance
```

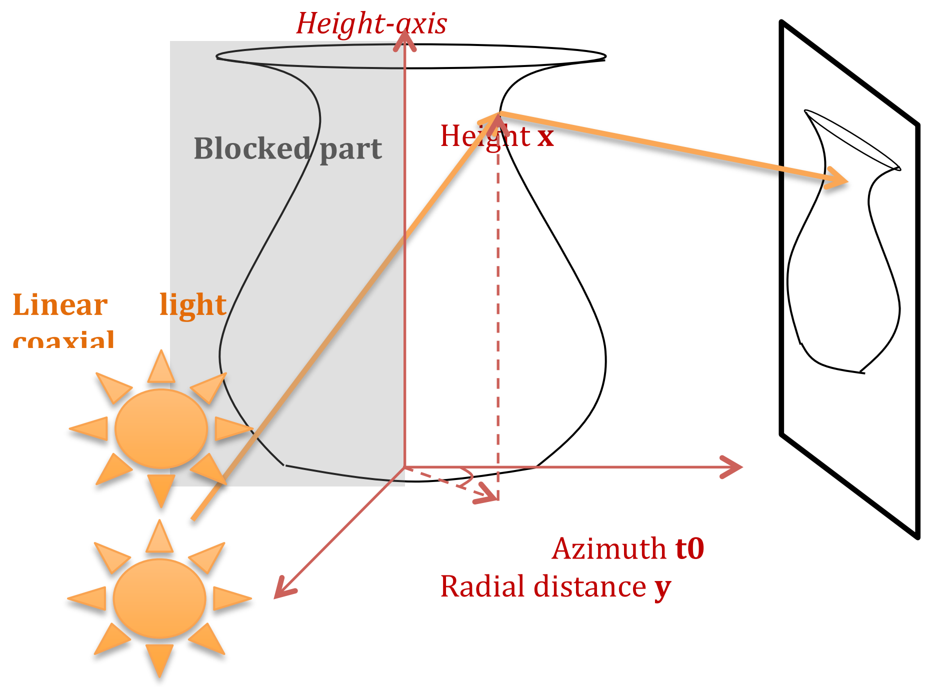

Figure 17. 3D Modeling and Recasting

3. Recasting

3.1 Recasting to the screen using linear light coaxial with the height coordinate

Here we mainly consider which part of the jar is to be displayed. That is to say, we want to cast the 3d model to a 2d screen. Part of information of the 3d jar will be discarded if the part is blocked by other part of the jar. If a point in 3D model is blocked by other points when casting to the screen, then we discard the information of the point. To realize this, we need to do at most 96*140* 140*0.5=940800 comparisons, each with 6 multiplications at most. That is a total: 5644800 multiplications at most. See Figure 17 for our 3Dmodeling and recasting.

```
    for k=1:97
    for z=1:(j-1)
    if j>1
    par=k1_u*(x0(k)-x(1,j))^2/(y(1,j)^2)+y0(k)^2/(y(1,j)^2); %to form comparison
parameter
    end
```

```
        if par>1                    %display on screen if not blocked by other
    part of jar
        x2greyStart=round(x0(k)*x_scale)+x_offset-x_halfWidth;
        …
```

3.2 Also consider one highlight line and the shadow in jar bore

We also highlight a line on the jar to show the effect of rotation and make some adjustment of the shadow effect in jar bore. First we need to remove highlight-line in the area of jar bore. Second we need to adjust the shadow of jar bore (to horizontally flip the shadow effect within the jar bore). There are a total of 48*140*6+10*MaxRadius*6=on the order of 10^4 multiplications to do at most for two comparison procedures.

```
        if angle>49         %only consider back half of the jar 50~96/96 samples
        for i=1:1:B*10      %for 140 points along a curve
        par1=(k1_d*(g_line_x(1,i)-g_bore_x(1,2)-x_halfWidth)^2/((x_scale*y(1,B*10))
^2)+(g_line
        _y(1,i)-g_bore_y(1,2)-y_halfWidth)^2/((y_scale*y(1,B*10))^2));
        if  par1<1              %remove  highlight  line  if  the  line  is  in  jar  bore
<COMPARISON 1>
        g(g_line_x(1,i)-x_halfWidth:g_line_x(1,i)+x_halfWidth,g_line_y(1,i)-y_halfWidt
h:g_line_y(
        1,i)+y_halfWidth)=245*sin(0.01*angle*pi)*cos(0.01*(angle-49.5)*pi);
        for xb=g_bore_x(1,1)+x_halfWidth:1:g_bore_x(1,3)+x_halfWidth
        for yb=g_bore_y(1,2):1:g_bore_y(1,3)+y_halfWidth
        par2=(k1_d*(xb-g_bore_x(1,2)-x_halfWidth)^2/((x_scale*y(1,B*10))^2)+(yb-g_
bore_y(1,
        j)-y_halfWidth)^2/((y_scale*y(1,B*10))^2));
        if  par2<1                      %if  within  jar  bore,  adjust  shadow
<COMPARISON 2>
        temp=g(xb,yb);
        …
```

4. Further modification and optimization when collaborating with the hardware and interface parts

For the sake of fast speed, we made some modification to the original idea of displaying the ellipse and also to the selection of greyscale value and the highlight part. The basic idea here is to get rid of all the loops or sin/cos calculation when deciding ellipse peripheral.

Considering ellipse peripheral calculation, we originally use ellipse's parameter function which involves a lot of sin and cos calculation within a big loop; later, we refer to the "middle point method" and predetermined all one-fourth peripherals with different radiuses.

Considering grey scale selection, we originally again use sin or cos functions and do the greyscale value selection in the ellipse peripheral calculation loop which is slow and costly; later, we refer to piecewise square function to make an reasonable approximate to the original theoretical effect function. We also predetermined all greyscale values for all the pixels in an ellipse with a specific radius values.

Considering the highlight part, we do it by comparing greyscale value (We round greyscale value to even value for front half and to odd number for back half, which allows us to notice the current position when doing highlighting.) The idea is based on the fact that the highlight line along the surface of the pottery should have identical greyscale value.

Here shows some detailed discussion of the modification and optimization mentioned above.

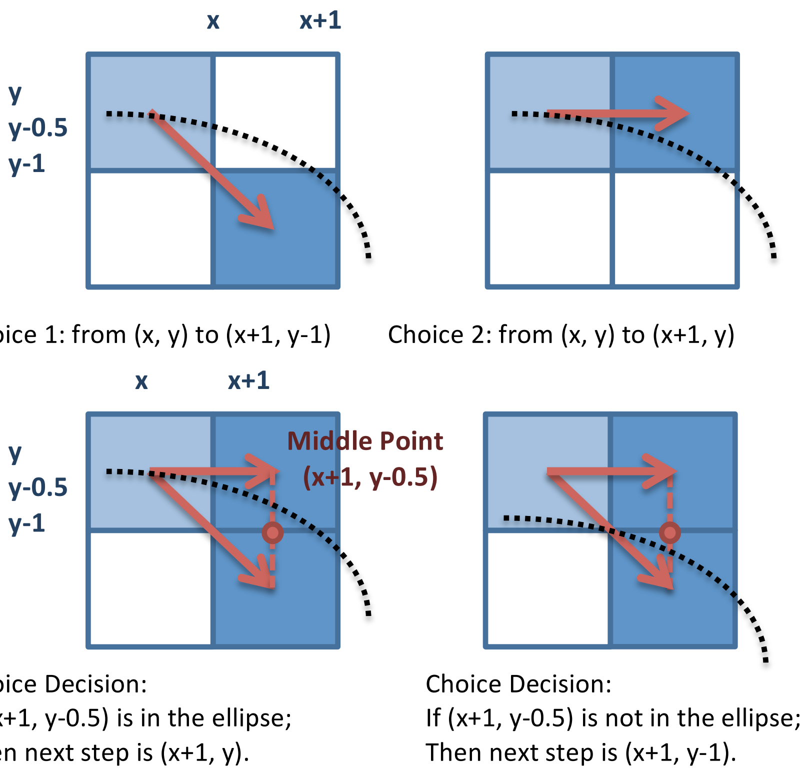
x
x+1
y
y-0.5
y-1
Choice 1: from (x, y) to (x+1, y-1)
Choice 2: from (x, y) to (x+1, y)
x
x+1
y
y-0.5
y-1
Middle Point
(x+1, y-0.5)
Choice Decision:
If (x+1, y-0.5) is in the ellipse;
Then next step is (x+1, y).
Choice Decision:
If (x+1, y-0.5) is not in the ellipse;
Then next step is (x+1, y-1).

Fig 18. Middle Point Method For Ellipse Drawing Algorithm

Actually, we use bunch of ellipses with different radiuses (we use the word radius throughout the report which actually corresponds to the value of the long axis of the

ellipse) around the pottery surface to construct an entire pottery. For each ellipse with a distinct radius value, we need to decide how to draw its peripheral quickly and accurately. Here, we use middle point method for this ellipse drawing task.
We first have to get the up-right quarter of ellipse and then mirror it to get the complete ellipse.
Each time we move on with one pixel leftward and decide whether to place it at the same vertical position or one vertical unit downwards according to the middle point of the two choices. That is to say, we use the middle point of the two choices as the decision element.
If the middle point is in the ellipse, then same vertical level point is more precise to present the ellipse; therefore we place the next pixel besides the previous one in the same vertical level. Otherwise, the point with one vertical unit downwards is more precise to present to ellipse; therefore we place the next pixel accordingly.
The decision and placing policy is showed in Figure 18. Middle point method for ellipse drawing.

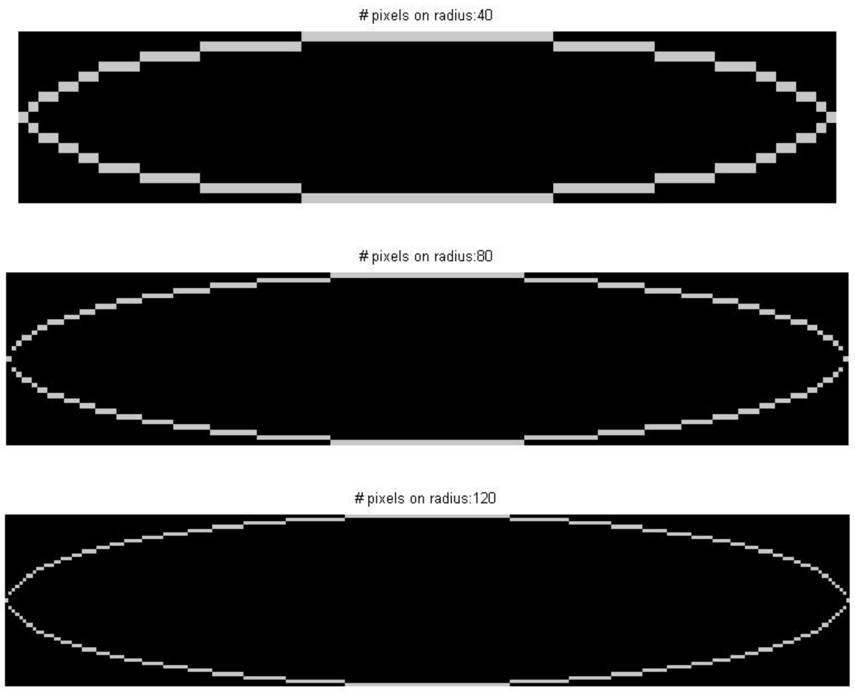

Figure 19. Ellipse drawn under various radius(top 40, middle 80, bottom 120)
Using this method, we predetermined all the positions for the up-right quarter ellipses of all the different radius values needed and store them in a ROM for hardware reference when displaying each pixel on the screen.

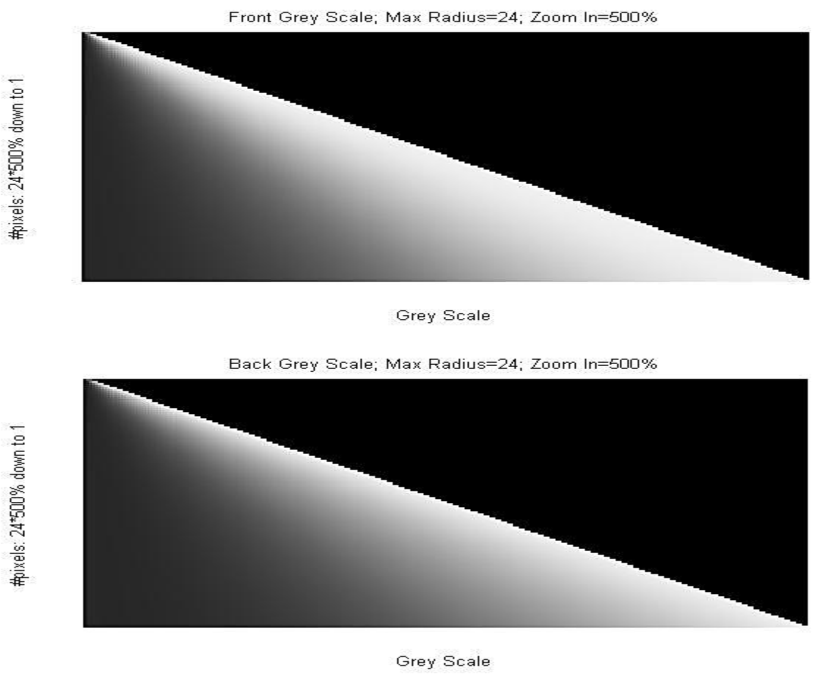

Figure 20. Grey Scale for the front half ellipse and back half ellipse

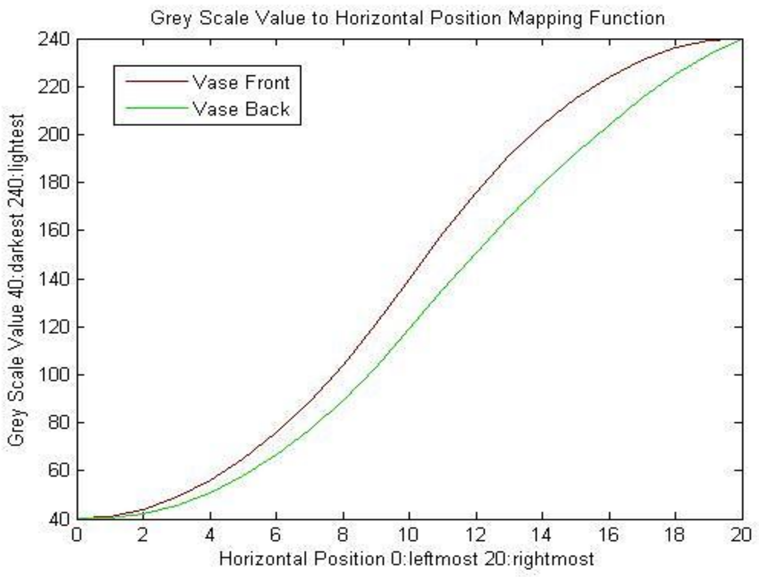

Figure 21. Grey Scale Mapping Function

The functions we use to calculate grey scale values for both the front and back of the pottery are plotted in Figure 21. Grey scale mapping function. The effect we get when using such mapping function are shown in Figure.20. We predetermine all the grey scale values needed for every different radius and again store them in two RAM (one for pottery front and another for pottery back) for later hardware reference.

5. Pattern matching and score generating method

Score is calculated using a parameter called discrepancy. The discrepancy between the model and the pottery one user make is defined as the sum of the absolute value of the difference between ideal model ratio and the current pottery ratio between every two adjacent control points along the pottery outline, which is showed as Equation.

Discrepancy=sum(abs(ideal ratio-current ratio)) Equation(1)

Where ideal ratio is the ratio between adjacent control points in ideal model and current ratio is the ratio between adjacent control points in the pottery one user made.

Note that "sum" is to sum up all the absolute ratio differences between every two adjacent control points along the pottery outline curve. Table. 1 Shows the three difficulty level models' parameter along with their ideal ratio values.

Table 1. Score generating parameter

Ctrl Point no.	Hard model parameter	Ideal Ratio CP_i+1/ CP_i	Middle model parameter	Ideal Ratio	Easy model parameter	Ideal Ratio
3	25		0.6		20	
4	33	1.32	0.75	1.3	20	1
5	31	0.94	0.89	1.2	20	1
6	23	0.74	0.9	1	20	1
7	50	2.17	0.89	1	20	1
8	36	0.72	0.75	0.8	20	1
9	33	0.92	0.6	0.8	20	1
10	48	1.45	0.5	0.8	20	1
11	50	1.04	0.5	1	20	1
12	33	0.66	0.5	1	20	1
13	26	0.79	0.5	1	20	1
14	25	0.96	0.5	1	20	1
15	27	1.08	0.5	1	20	1
16	31	1.15	0.5	1	20	1

6. Mouse Control

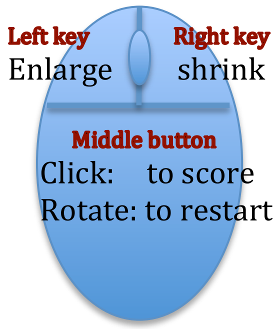

Figure 22. Function of a mouse in this game

As is shown in Figure 22. The mouse information contains left or right click, which corresponds to enlarge or shrink the current pottery position; mouse wheel/middle button click, which corresponds to begin playing or to view current score; mouse wheel/middle button rotate up or down, which correspond to restart the game. Furthermore, the mouse information also contains x and y displacements, which are used to calculate cursor's position both in current pottery (current pottery position) and on the display screen (mouse display position).

7. Several Predetermined ROM Size

Here we give the sizes for the predetermined ROMs briefly in Table2.

Table 2 ROM size

Rom	Supported radius range after scale (i.e. actual pixel number on the screen)	Data length in byte	Total rom size	Actual space allocated
Front greyscale	0~150	1	11475 bytes	16384 bytes
Back greyscale				
Ellipse peripheral				

Issues and experiences

During our design process, we have encountered many problems and issues. Here we listed several examples of our problems and how we solve them. These are the key points of our design, which facilitate us a lot in our design flow and debug process.

Data required to display the whole pottery is very huge and we don't have enough memory to store it.

Solution: We use the idea of a memory window which is of a size just large enough to hold the information needed to draw a largest possible ellipse, and keep reusing that window by erasing the information that has already been displayed on the screen. By using this method, we finish the same task, but largely shrink down the size of the storage element we need and use as little sources on the socKit board as possible. The detailed idea is as follows: the window is divided into three parts, the first line of the window is the line to be erased every time, and the second line in the window is the line being displayed on the screen, and the rest part of the window is used to draw ellipses. When we come to the bottom of the window, we will wrap around and start from the first line again.

Memory window is moving constantly, so how can we determine where the center of the ellipse being drawn in the window.

Solution: We take the mod of the vertical position by window width: (vcount-65)%win_h. We use the similar method to determine which line in the window is being displayed and which line is erased: (vcount-win_st)%win_w, (vcount-1-win_st)%win_w.

There are some patterns, which we don’t want, displaying on the screen at every window boundaries, which seemed like noises.

Solution: We will not draw ellipses starting from the third line, instead we leave several lines’ space to allow for some time for ellipses drawing. Since drawing ellipses requires time, if we have very large ellipses, they may require longer time to draw and at the time of display they may not finished, so there will be noises displaying on the screen. We assign a valid range of the positions where we can draw ellipses, and only in this range can we draw ellipses. We allow 15 lines of valid area. At the same time, we also enlarge the width of the window to allow for more time for ellipses drawing.

assign valid_y_in_buf = (ready&& (y_in_buf>=vcount+30)&&(y_in_buf<=vcount+45));

```
always@(posedge ready or posedge reset) begin
if (reset)
        y_in_buf<=y_in_st;
else if (addr_rom_xin<=138 && (y_in_buf>380||y_in_buf<100))
        y_in_buf<=y_in_st;
else if (addr_rom_xin<=138 && y_in_buf<=380&&y_in_buf>=100)
        y_in_buf<=y_in_buf+'d2;
else if (addr_rom_xin>=139 && (y_in_buf>380||y_in_buf<100))
      y_in_buf<=y_in_st;
else if (addr_rom_xin>=139 && y_in_buf<=380&&y_in_buf>=100)
        y_in_buf<=y_in_st;
else
        y_in_buf<=y_in_buf;
end
```

When we first draw the ellipses, there are bunches of ellipses continuously traveling across the screen and cannot be stable.

Solution: We change the logic of drawing ellipses from combinational logic to sequential logic. Originally, the logic of drawing ellipses was combinational logic and there was no delay, so we were in fact continuously drawing ellipses and didn't connect them with hcount and vcount. After we changed it to sequential logic, the drawing ellipses block were synchronized with the displaying block, and we got the stable ellipses on the screen.

When we connect the mouse, the movement of mouse is very sensitive and hard to stabilize.

Solution: Since the mouse movement is characterized by the x_displacement and y_displacement, and a minor movement of the mouse will result in some changes in those two variables, so the position change of our cursor was extremely sensitive and was very hard to control. So we scale down the x_displacement and y_displacement of the mouse movement in software, so the change of the mouse position is not so sensitive and can have a better user experience when they play the game.

Debugging method

Various methodology and tools have been used by us in the project:

1. RTL viewer:

   We use the RTL viewer to see the circuit structure and connection of our system. We can use it to check whether we have a right connection of the different blocks. This is a straight way to check errors in our system and usually the first step to debug.

2. SignalTap:

   By making sure each block connected correctly, we want to see how is the FPGA actually running, what value actually holds in these registers when there is a bug happens. Thus we choose SignalTap to solve our problems. While debugging, we first choose the signal nodes we want to observe and check, then compile the whole project in Quartus and run the SignalTap to see the signals on the nodes we selected. We then analyze signals on different nodes and compare them with the signals it should be. By this way, we found a lot of errors and bugs and solved a lot of problems.

3. ModelSim:

   Modelsim provides various function in our design, as we know, we cannot record the value in SignalTap or make it slow down to see the exact relation between the value and time, while ModelSim does, it provides the waveform simulation for our hardware design, thus it give us a detailed behavior of our design, it is extremely useful when tacking timing problems. Besides this debugging, it also shoulders the responsibility in verifying our design. Compared to blindly speding 15 minutes compling the project on every minor changes, we save time by doing a 15 seconds simulation to see if the change works.

4. “printf” function in C:

   As we can look at waveform to have a clear understanding of hardware, we can print the value of variables to the terminal to monitor the running of software, nearly every bug in the software are solve by first monitoring it using printf. This function also let us know the behavior of a USB-Mouse.

Lessons learned

By doing this 3D-Pottery Game project this semester, we have a clear understanding of the embedded-system design flow. From the verification of implementing algorithm to hardware implementing and interconnection between software and hardware. This project also give us deeper understanding of FPGA SoCKit board. We have learned how to exploit the powerful function of this FPGA board, and also experienced how to utilize the Linux system embedded on the FPGA, thus combining software code and hardware architecture to accomplish our project.
Besides, our projects exploit a lot about various powerful EDA tools, such as Qsys, which enable us to deal with Avalon bus and connect software kernel to the hardware; MegaWizard, which help us using the memory block on the board conveniently and efficiently; SignalTap, which help us knowing how the FPGA are actually running; and Quartus, of course, which is really a powerful tool in FPGA design.
Our coding ability also improved a lot. For software coding, we gained experience on Matlab and C code, for the hardware part, now we can handle behavioral level coding with some proficiency, and have learned how to deal with timing problem, which turns out to be a nasty but also the most skillfully and delicately part in hardware design.
However, the most important thing we learned through this project is not those knowledge and tricks, but is how to learn, work and fight as a team. Today's cool design trick may become obsoleted in the future, but our experience of fighting as a team shall never fade away.

Contribution

Xiaowen Han: Designed, and implemented various hardware modeling algorithm, such as ellipse drawing ,ram_window_drawing, cursor display and so on. Successfully implement the trick of window-drawing and also optimized this algorithm to make the modeling more smoothly and correctly.

Muqing Liu: Designed and implemented various hardware display module, such as the VGA_LED_Emulator and find ways manipulating with logic calculation concerning with display problems, also worked together in implementing hardware pottery modeling algorithm

Chengxue Qian: Created and verified the algorithm of pottery modeling in software, coded various software module such as control point generating, pattern matching, score generating and so on. Successfully find the way to model a three-dimensional object using two-dimensional drawing.

Hao Jin: Designed various hardware /software interface, such as mouse control and data transfer, to make the interaction work smoothly, also responsible for debugging and worked together in coding and solving various problems in both hardware and software part.

APPENDIX

1. Hardware Part

VGA_LED.sv

```
/*
 * Avalon memory-mapped peripheral for the VGA LED Emulator
 *
 * XiaoWen Han
 * MuQing Liu
 * Hao Jin
 * Chengxue Qian
 * Columbia University
 */

module VGA_LED(
                   input logic       clk,
                   input logic        reset,
                   input logic  [7:0]  writedata,
                   input logic        write,
                   input             chipselect,
                   input logic  [3:0]  address,
                   output logic [7:0]  VGA_R, VGA_G, VGA_B,
                   output logic        VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n,
                   output logic        VGA_SYNC_n);



    logic [7:0]       hex0,hex1,hex2,hex3,hex4,hex5,hex6,
              hex7,hex8,hex9,hex10,hex11,hex12;
    logic [14:0]     address_a,address_b;
    logic [7:0]      data_a,data_b;
    logic           rden_a,rden_b,wren_a,wren_b;
    logic [7:0]      q_a,q_b;
    logic           clk25; ///vga_emulator
    logic [8:0]      y_in;
    logic [9:0]      y_in_buf;
    logic [7:0]      data_rom_f,data_rom_b;//gray rom output
    logic [7:0]   data_rom_h;
    logic [13:0]     addr_rom_f,addr_rom_b; //gray rom address
    logic [12:0]     addr_rom_h; //height rom address
    logic           ready,enable,valid_y_in_buf;
    logic [9:0]      vcount;
    logic [10:0]     hcount;
    logic [7:0]      x_in;
    logic           rden_x_in;
    logic [7:0]      addr_rom_xin;
    logic [15:0]     addr_rom_bg;
    logic [7:0]      data_rom_bg;
    logic [9:0]      cursor_x,cursor_y;
```

```
//--------------------- logic for welcome-----------------------------
  logic [7:0]
    data_exam1,data_exam2,data_exam3,data_exam4,data_wel_ins;
  logic [13:0]      addr_exam1,addr_exam2,addr_exam3,addr_exam4;
  logic [14:0]      addr_wel_ins;




//----------------------logic for slogan---------------------
  logic [7:0]       data_rom_win,data_rom_keep;
  logic [13:0]      addr_slogan;
  logic [12:0]      addr_ins;
  logic [7:0]       data_rom_ins;

    parameter y_in_st=100;

    assign              valid_y_in_buf              =               (ready&&
(y_in_buf>=vcount+37)&&(y_in_buf<=vcount+40));


    always@(posedge ready or posedge reset)
        begin
            if (reset)
                y_in_buf<=y_in_st;
            else if (addr_rom_xin<=138 && (y_in_buf>380||y_in_buf<100))
                y_in_buf<=y_in_st;
            else if (addr_rom_xin<=138 && y_in_buf<=380&&y_in_buf>=100)
                y_in_buf<=y_in_buf+'d2;
            else if (addr_rom_xin>=139 && (y_in_buf>380||y_in_buf<100))
                y_in_buf<=y_in_st;
            else if (addr_rom_xin>=139 && y_in_buf<=380&&y_in_buf>=100)
                y_in_buf<=y_in_st;
            else
                y_in_buf<=y_in_buf;
        end //end always



//--------------------------------------modify y_in----------------------------

    always@(negedge clk)
        begin
            if (reset)
                begin
                    addr_rom_xin<=0;
                    y_in<=y_in_st;
                    rden_x_in<=1;
                end
            else if (ready&&valid_y_in_buf&&addr_rom_xin<=138)
                begin
                    y_in<=y_in_buf;
                    addr_rom_xin<=addr_rom_xin+'d1;
                    rden_x_in<=1;
                end
            else if (ready&&valid_y_in_buf&&addr_rom_xin>=139)
                begin
                    y_in<=y_in_buf;
                    addr_rom_xin<=0;
                    rden_x_in<=1;
                end
            else
            begin
                y_in<=y_in;
                addr_rom_xin<=addr_rom_xin;
                rden_x_in<=1;
            end
        end //always



    ram_win                                                              ram_win
(address_a,address_b,clk,data_a,data_b,rden_a,rden_b,wren_a,wren_b,q_a,q_b);

    ellipse_draw
    ellipse_draw(clk,reset,clk25,x_in,y_in,data_rom_f,data_rom_b,data_rom_h,q_b,
valid_y_in_buf,


    addr_rom_f,addr_rom_b,addr_rom_h,address_b,data_b,wren_b,rden_b,ready);


    rom150front rom150front(addr_rom_f,clk,data_rom_f);


    rom150back rom150back(addr_rom_b,clk,data_rom_b);


    romheight romheight(addr_rom_h,clk,data_rom_h);


    VGA_LED_Emulator led_emulator(.clk50(clk), .*);


    rom_bg rom_bg (addr_rom_bg,clk,data_rom_bg);


    inter_face_top
    inter_face_top(clk,reset,addr_rom_xin,hex0,hex1,hex2,hex3,hex4,hex5,hex6,hex
7,x_in);


    rom_exam1 rom_exam1 (addr_exam1,clk,data_exam1);


    rom_exam2 rom_exam2 (addr_exam2,clk,data_exam2);


    rom_exam3 rom_exam3 (addr_exam3,clk,data_exam3);


    rom_exam4 rom_exam4 (addr_exam4,clk,data_exam4);


    rom_select_pottery rom_select_pottery (addr_wel_ins,clk,data_wel_ins);


    rom_win rom_win (addr_slogan,clk,data_rom_win);


    rom_keep rom_keep (addr_slogan,clk,data_rom_keep);


    rom_ins rom_ins (addr_ins,clk,data_rom_ins);
    assign cursor_x = hex8+320;
    assign cursor_y = hex9*10;


   always_ff @(posedge clk)
     if (reset)
        begin
            hex0 <= 8'b0;
            hex1 <= 8'b0;
            hex2 <= 8'b0;
            hex3 <= 8'b0;
            hex4 <= 8'b0;
            hex5 <= 8'b0;
            hex6 <= 8'b0;
            hex7 <= 8'b0;
            hex8 <= 8'b0;
            hex9 <= 8'b0;
            hex10 <= 8'b0;
            hex11 <= 8'b0;
            hex12 <= 8'b0;
     end
      else if (chipselect && write)
      case (address)
            4'd0 : hex0 <= writedata;
            4'd1 : hex1 <= writedata;
            4'd2 : hex2 <= writedata;
            4'd3 : hex3 <= writedata;
            4'd4 : hex4 <= writedata;
            4'd5 : hex5 <= writedata;
            4'd6 : hex6 <= writedata;
            4'd7 : hex7 <= writedata;
            4'd8 : hex8 <= writedata;
            4'd9 : hex9 <= writedata;
            4'd10 : hex10 <= writedata;
            4'd11 : hex11 <= writedata;
            4'd12 : hex12 <= writedata;
      endcase


endmodule
```

VGA_LED_Emulator.sv

```
module VGA_LED_Emulator(
            input logic        clk50, reset,
            input logic [7:0]  hex0, hex1, hex2, hex3, hex4, hex5, hex6,
                        hex7,hex10,hex11,hex12,
            input logic [7:0]  q_a,
            input logic [7:0]  data_rom_bg,
            input logic [9:0]  cursor_x, cursor_y,
            input                         logic                        [7:0]
    data_exam1,data_exam2,data_exam3,data_exam4,data_wel_ins,
            input logic [7:0]  data_rom_win,data_rom_keep,data_rom_ins,
            output logic[13:0] addr_exam1,addr_exam2,addr_exam3,addr_exam4,
            output logic[14:0] addr_wel_ins,
            output logic[7:0]  VGA_R, VGA_G, VGA_B,
            output logic       VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n, VGA_SYNC_n,
            output logic       clk25, enable,
            output logic[14:0] address_a,
            output logic       rden_a,wren_a,
            output logic[7:0]  data_a,
            output logic[9:0]  vcount,
            output logic[10:0] hcount,
            output logic[15:0] addr_rom_bg,
            output logic[13:0]   addr_slogan,
            output logic[12:0]   addr_ins

 );



  parameter HACTIVE     = 11'd 1280,
          HFRONT_PORCH = 11'd 32,
          HSYNC       = 11'd 192,
          HBACK_PORCH  = 11'd 96,
          HTOTAL      = HACTIVE + HFRONT_PORCH + HSYNC + HBACK_PORCH;

  parameter VACTIVE     = 10'd 480,
          VFRONT_PORCH = 10'd 10,
          VSYNC       = 10'd 2,
          VBACK_PORCH  = 10'd 33,
          VTOTAL      = VACTIVE + VFRONT_PORCH + VSYNC + VBACK_PORCH;
  logic                endOfLine;

  always_ff @(posedge clk50 or posedge reset)
       if (reset)
           hcount <= 0;
       else if (endOfLine)
           hcount <= 0;
       else
           hcount <= hcount + 11'd 1;

  assign endOfLine = hcount == HTOTAL - 1;

  logic                endOfField;

  always_ff @(posedge clk50 or posedge reset)
       if (reset)
           vcount <= 0;
       else if (endOfLine)
           if (endOfField)
```

```
            vcount <= 0;
        else
            vcount <= vcount + 10'd 1;

assign endOfField = vcount == VTOTAL - 1;
assign VGA_HS = !( (hcount[10:8] == 3'b101) & !(hcount[7:5] == 3'b111));
assign VGA_VS = !( vcount[9:1] == (VACTIVE + VFRONT_PORCH) / 2);
assign VGA_SYNC_n = 1;
assign VGA_BLANK_n = !( hcount[10] & (hcount[9] | hcount[8]) ) &
         !( vcount[9] | (vcount[8:5] == 4'b1111) );
assign VGA_CLK = hcount[0];
assign clk25 = VGA_CLK;

parameter win_w=400;
parameter win_h=70;


logic inwin;
assign inwin= (hcount[10:1]>=120&&hcount[10:1]<520&&vcount>=65&&vcount<=414);

always@*
     begin
        if (inwin)
            begin
                enable = 1;
           end
        else
            enable = 0;
     end

always_ff@ (negedge clk50)
     begin
        if (reset)
            begin
                rden_a <=0;
                wren_a <=0;
                address_a<=0;
            end
        else if (VGA_CLK==1&&enable==1)
            begin
                rden_a<=1;
                wren_a<=0;
                address_a <= ((vcount-65)%win_h)*win_w + (hcount[10:1]-120);
            end
        else if (VGA_CLK==0&&enable==1&&vcount==65)
            begin
                rden_a<=0;
                wren_a<=1;
                address_a <= ((win_h-1)%win_h)*win_w + (hcount[10:1]-120);
                data_a<=0;
            end
        else if (VGA_CLK==0&&enable==1&&vcount!=65)
            begin
                rden_a<=0;
                wren_a<=1;
                address_a <= ((vcount-66)%win_h)*win_w + (hcount[10:1]-120);
                data_a<=0;
            end
        else
            begin
                rden_a<=0;
                wren_a<=0;
            end
 end //end always
```

```
//--------------------------------------draw texture----------------------
    logic[19:0] counter_hl ;
    logic[8:0]  gray_hl;

  always@(posedge VGA_CLK)
        begin
            if (reset)
                begin
                    counter_hl<=20'h0;
                    gray_hl<=8'd220;
                end
            else if (counter_hl<20'd8191&&gray_hl>8'd20)
                begin
                    counter_hl<=counter_hl+'d1;
                    gray_hl<=gray_hl;
                end
            else if (counter_hl<20'd8191&&gray_hl<=8'd20)
                begin
                    counter_hl<=counter_hl+'d1;
                    gray_hl<=gray_hl;
                end
            else if (counter_hl==20'd8191&&gray_hl>8'd20)
                begin
                    counter_hl<=0;
                    gray_hl<=gray_hl-8'd2;
                end
            else if (counter_hl==20'd8191&&gray_hl<=8'd20)
                begin
                    counter_hl<=0;
                    gray_hl<=8'd220;
                end
            else
                begin
                    counter_hl<=counter_hl;
                    gray_hl<=gray_hl;
                end
        end//always




//----------------------------------cursor display--------------------
    logic   incursor;

    assign
    incursor=(hcount[10:1]>=cursor_x-4&&hcount[10:1]<=cursor_x+4&&vcount==cursor
_y+2)||

(hcount[10:1]>=cursor_x-3&&hcount[10:1]<=cursor_x+3&&vcount==cursor_y+1)||
             (hcount[10:1]>=cursor_x-2       &&       hcount[10:1]<=cursor_x+2&&
vcount==cursor_y)||
             (hcount[10:1]>=cursor_x-1       &&       hcount[10:1]<=cursor_x+1&&
vcount==cursor_y-1)||
             (hcount[10:1]==cursor_x&&vcount==cursor_y-2);



 //----------------------------------welcome window-----------------------
    logic example1,example2,example3,example4,wel_ins;
logic [9:0]cent_cir_x,cent_cir_y;
logic sel_circle;
logic[7:0] data_exam10,data_exam20,data_exam30,data_exam40;

assign sel_circle=((hcount[10:1]-cent_cir_x)*(hcount[10:1]-cent_cir_x)+
           (vcount-cent_cir_y)*(vcount-cent_cir_y)>=48*48) &&
           ((hcount[10:1]-cent_cir_x)*(hcount[10:1]-cent_cir_x)+
           (vcount-cent_cir_y)*(vcount-cent_cir_y)<=51*51);

always@(posedge clk50)
    begin
        if (reset)
            begin
                cent_cir_x<=0;
                cent_cir_y<=0;
                data_exam10<=0;
                data_exam20<=0;
                data_exam30<=0;
                data_exam40<=0;
            end
        else if (hex11==1)
            begin
                cent_cir_x<=260;
                cent_cir_y<=130;
                    if (data_exam1<=5)
                        data_exam10<=data_exam1;
                    else
                        data_exam10<=data_exam1*2;
                data_exam20<=data_exam2;
                data_exam30<=data_exam3;
                data_exam40<=data_exam4;
            end
        else if (hex11==2)
            begin
                cent_cir_x=380;
                cent_cir_y=130;
                data_exam10<=data_exam1;
                    if (data_exam2<=3)
                        data_exam20<=data_exam2;
                    else
                        data_exam20<=data_exam2*2;
                data_exam30<=data_exam3;
                data_exam40<=data_exam4;
            end
        else if (hex11==3)
            begin
                cent_cir_x<=260;
                cent_cir_y<=250;
                data_exam10<=data_exam1;
                data_exam20<=data_exam2;
                    if (data_exam3<=3)
                        data_exam30<=data_exam3;
                    else
                        data_exam30<=data_exam3*2;
                data_exam40<=data_exam4;
            end
        else if (hex11==4)
            begin
                cent_cir_x<=380;
                cent_cir_y<=250;
                data_exam10<=data_exam1;
                data_exam20<=data_exam2;
                data_exam30<=data_exam3;
                    if (data_exam4<=3)
                        data_exam40<=data_exam4;
                    else
                        data_exam40<=data_exam4*2;
                end
            else
                begin
                    cent_cir_x<=cent_cir_x;
                    cent_cir_y<=cent_cir_y;
                    data_exam10<=data_exam1;
                    data_exam20<=data_exam2;
                    data_exam30<=data_exam3;
                    data_exam40<=data_exam4;
                end
        end



    assign
example1=(hcount[10:1]>=210&&hcount[10:1]<310&&vcount>=80&&vcount<180);
    assign
example2=(hcount[10:1]>=330&&hcount[10:1]<430&&vcount>=80&&vcount<180);
    assign
example3=(hcount[10:1]>=210&&hcount[10:1]<310&&vcount>=200&&vcount<300);
    assign
example4=(hcount[10:1]>=330&&hcount[10:1]<430&&vcount>=200&&vcount<300);
    assign                                                              wel_ins
=(hcount[10:1]>=170&&hcount[10:1]<470&&vcount>=320&&vcount<410);




 //-----------------------------welcome window four example---------------------
    logic[13:0] addr_exam1_wel, addr_exam2_wel, addr_exam3_wel, addr_exam4_wel;

    always@(negedge clk50)
        begin
            if (reset)
                begin
                    addr_exam1_wel<=0;
                    addr_exam2_wel<=0;
                    addr_exam3_wel<=0;
                    addr_exam4_wel<=0;
                end
            else if (example1)
                    addr_exam1_wel<=(vcount-80)*100 + (hcount[10:1]-210);
            else if (example2)
                    addr_exam2_wel<=(vcount-80)*100 + (hcount[10:1]-330);
            else if (example3)
                    addr_exam3_wel<=(vcount-200)*100 + (hcount[10:1]-210);
            else if (example4)
                    addr_exam4_wel<=(vcount-200)*100 + (hcount[10:1]-330);
            else if (wel_ins)
                    addr_wel_ins<=(vcount-320)*300 + (hcount[10:1]-170);
        end//endalways


    always@(negedge clk50)
        begin
    if (reset)
        addr_rom_bg<=0;
    else if (!incursor&& q_a==0 && hcount[10:1]>55 && hcount[10:1]<320 && vcount>=0
&& vcount<240)
        addr_rom_bg<=(239-vcount)*270+319-hcount[10:1];
    else if (!incursor&& q_a==0&& hcount[10:1]>=320 && hcount[10:1]<590 && vcount>=0
&& vcount<240)
        addr_rom_bg<=(239-vcount)*270+hcount[10:1]-320;
    else if (!incursor&&q_a==0 && hcount[10:1]>55 && hcount[10:1]<320 && vcount>=240
&& vcount<480)
        addr_rom_bg<=(vcount-240)*270+319-hcount[10:1];
    else     if     (!incursor&&     q_a==0     &&     hcount[10:1]>=320     &&
hcount[10:1]<590&&vcount>=240 && vcount<480)
        addr_rom_bg<=(vcount-240)*270+hcount[10:1]-320;
    else
        addr_rom_bg<=addr_rom_bg;
    end



 //------------------------------------------example_ref choose------------------
    logic example_ref;
    logic [7:0] data_ref;
    logic [13:0]  addr_exam1_dis,addr_exam2_dis,addr_exam3_dis,addr_exam4_dis;
    assign
example_ref=(hcount[10:1]>=120&&hcount[10:1]<220&&vcount>=190&&vcount<290);

    always@*
        begin
            if (hex11==1)
                begin
                    addr_exam1_dis=(vcount-190)*100+hcount[10:1]-120;
                    data_ref=data_exam1;
                end
            else if (hex11==2)
                begin
                    addr_exam2_dis=(vcount-190)*100+hcount[10:1]-120;
                    data_ref=data_exam2;
                end
            else if (hex11==3)
                begin
                    addr_exam3_dis=(vcount-190)*100+hcount[10:1]-120;
                    data_ref=data_exam3;
                end
            else if (hex11==4)
                begin
                    addr_exam4_dis=(vcount-190)*100+hcount[10:1]-120;
                    data_ref=data_exam4;
                end
        end// end always



 //------------------------------mux to choose addr_exam--------------------

    always@*
        begin
            if (hex10==0)
                begin
                    addr_exam1=addr_exam1_wel;
                    addr_exam2=addr_exam2_wel;
                    addr_exam3=addr_exam3_wel;
                    addr_exam4=addr_exam4_wel;
                end
            else if (hex10==1)
                begin
                    addr_exam1=addr_exam1_dis;
                    addr_exam2=addr_exam2_dis;
                    addr_exam3=addr_exam3_dis;
                    addr_exam4=addr_exam4_dis;
                end
        end //endalways




     //-------------------------------win or keep doing--------------------------
        logic       inslogan;
        logic [7:0]  data_slogan;
        assign
inslogan=(hex12!=0&&(hcount[10:1]>=170&&hcount[10:1]<470&&vcount>=225&&vcount<2
55));


        always@(negedge clk50)
        begin
            if (reset)
                begin
                    addr_slogan<=0;
                    data_slogan<=0;
                end
            else if (hex12==1&& inslogan)
                begin
                    addr_slogan<= (vcount-225)*300+hcount[10:1]-170;
                    data_slogan <= data_rom_win;
                end
            else if (hex12==2 && inslogan)
                begin
                    addr_slogan <= (vcount-225)*300+hcount[10:1]-170;
                    data_slogan <= data_rom_keep;
                end
        end // always






//---------------------------------instruction display-------------------------
        logic inins;
        assign inins=(hcount[10:1]>=220&&hcount[10:1]<420&&vcount>=395&&vcount<435);

        always @(negedge clk50)
        begin
            if (reset)
                addr_ins<=0;
            else if (inins)
                addr_ins<=(vcount-395)*200+hcount[10:1]-220;
            else
                addr_ins<=addr_ins;
        end//endalways



//---------------------------------display on VGA -------------------------------


        always@*
        begin
        {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0}; // Black
           if (hex10==0)
            begin
                if (sel_circle)
                    {VGA_R, VGA_G, VGA_B} = {8'hb0,8'h0,8'h0};
                else if (example1&&!sel_circle)
                    {VGA_R, VGA_G, VGA_B} = {data_exam10,data_exam10, data_exam10};
                else if (example2&&!sel_circle)
                    {VGA_R, VGA_G, VGA_B} = {data_exam20,data_exam20, data_exam20};
                else if (example3&&!sel_circle)
                    {VGA_R, VGA_G, VGA_B} = {data_exam30,data_exam30, data_exam30};
                else if (example4&&!sel_circle)
                    {VGA_R, VGA_G, VGA_B} = {data_exam40,data_exam40, data_exam40};
                else if (wel_ins&&data_wel_ins!=0)
                    {VGA_R,   VGA_G,   VGA_B}   =   {data_wel_ins,data_wel_ins,
data_wel_ins};
                else if (wel_ins&&data_wel_ins==0)
                    {VGA_R, VGA_G, VGA_B} = {data_rom_bg, data_rom_bg, data_rom_bg};
                else if ( hcount[10:1]>55 && hcount[10:1]<320 && vcount>=0 &&
vcount<240)
                    {VGA_R, VGA_G, VGA_B} = {data_rom_bg, data_rom_bg, data_rom_bg};
                    else if (hcount[10:1]>=320 && hcount[10:1]<590 && vcount>=0 &&
vcount<240)
                    {VGA_R, VGA_G, VGA_B} = {data_rom_bg, data_rom_bg, data_rom_bg};
                else if (hcount[10:1]>55 && hcount[10:1]<320 && vcount>=240 &&
vcount<480)
                    {VGA_R, VGA_G, VGA_B} = {data_rom_bg, data_rom_bg, data_rom_bg};
                else    if    (hcount[10:1]>=320    &&    hcount[10:1]<590    &&
vcount>=240&&vcount<480)
                    {VGA_R, VGA_G, VGA_B} = {data_rom_bg, data_rom_bg, data_rom_bg};
             end



else if (hex10==1)
            begin
                if (inslogan&&hex12==1&&data_slogan>=100)
                    {VGA_R, VGA_G, VGA_B} = {data_slogan, 8'h0, 8'h0};
                else if   (inslogan&&hex12==2&&data_slogan>=100)
                    {VGA_R, VGA_G, VGA_B} = {8'd2, 8'd147, 8'd230};
                else if (example_ref && data_ref!=0)
                    {VGA_R, VGA_G, VGA_B} = {data_ref,data_ref, data_ref};
                else if (incursor)
                    {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h0, 8'h0};
                else                                                          if
(inwin&&q_a!=0&&q_a!=(gray_hl-2)&&q_a!=gray_hl&&q_a!=(gray_hl+2))
                    {VGA_R, VGA_G, VGA_B} = {q_a, q_a, q_a}; //read win gray scale
                else if (inins && q_a==0 && data_rom_ins!=0)
                    {VGA_R,   VGA_G,   VGA_B}   =   {data_rom_ins,data_rom_ins,
data_rom_ins};
                else                       if(inwin                           &&
(q_a==(gray_hl-2)||q_a==gray_hl||q_a==(gray_hl+2)))
                    {VGA_R, VGA_G, VGA_B} = {8'd20, 8'd20, 8'd20};
                else                                                          if
(q_a==0&&hcount[10:1]>55&&hcount[10:1]<320&&vcount>=0&&vcount<240)
                    {VGA_R, VGA_G, VGA_B} = {data_rom_bg, data_rom_bg, data_rom_bg};
          else                                                                if
(!incursor&&q_a==0&&hcount[10:1]>=320&&hcount[10:1]<590&&vcount>=0 && vcount<240)
                    {VGA_R, VGA_G, VGA_B} = {data_rom_bg, data_rom_bg, data_rom_bg};
          else                                                                if
(!incursor&&q_a==0&&hcount[10:1]>55&&hcount[10:1]<320&&vcount>=240&& vcount<480)
                    {VGA_R, VGA_G, VGA_B} = {data_rom_bg, data_rom_bg, data_rom_bg};
          else                                                                if
(!incursor&&q_a==0&&hcount[10:1]>=320&&hcount[10:1]<590&&vcount>=240&&vcount<48
0)
```

```
                    {VGA_R, VGA_G, VGA_B} = {data_rom_bg, data_rom_bg, data_rom_bg};
                else if (data_rom_bg==0)
                    {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0}; // Black
                else
                    {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0};
            end
        end


endmodule // VGA_LED_Emulator
```

ellipse_draw.sv

```
module ellipse_draw(
            input logic          clk50,reset,clk25,
            input logic [7:0]     x_in,
            input logic [8:0]     y_in,
            input logic [7:0]     data_rom_f, data_rom_b,data_rom_h,
            input logic [7:0]     data_valid,
            input logic          valid_y_in_buf,
            output logic [13:0]   addr_rom_f, addr_rom_b,
            output logic [12:0]   addr_rom_h,
            output logic [14:0]   addr_win,
            output logic [7:0]    data_win,
            output logic         wren,rden,ready);



    logic [8:0]      y_in_b;
    logic [7:0]      rad;
    logic [7:0]      x_count;
    logic [1:0]      four; //0,1,back 2,3,front
    logic [8:0]      cent_y;
    logic [6:0]      win_y;
    logic [8:0]      win_x;
    logic [7:0]      data_rom_real;
    logic [2:0]      ecc_count;


    parameter win_w=400;
    parameter win_h=70;
    parameter y_in_st=100;


     always @(negedge clk50)
        begin
            if (reset)
                begin
                    rad<=0;
                    y_in_b<=y_in_st;
                end
            else if (valid_y_in_buf&&ready==1)
                begin
                    rad <= x_in;
                    y_in_b<=y_in;
                end
            else
                begin
                    rad <= rad;
                    y_in_b<=y_in_b;
                end
        end //end always



    always_ff @(posedge clk25)
        begin
            if(reset||(valid_y_in_buf&&ready==1))
                begin
                    x_count <=0;
                    four <=0;
                    ecc_count <=0;
                    ready<=0;
                end
            else if (ecc_count<3&&x_count<rad&&four<3)
                begin
                    ecc_count<=ecc_count+'d1;
                    x_count<=x_count;
                    four<=four;
                    ready<=0;
                end
           else if (ecc_count==3&&x_count<rad&&four<3)
                begin
                    ecc_count<=0;
                    x_count<=x_count+1;
                    four<=four;
                    ready<=0;
                end
           else if (ecc_count==3&&x_count==rad&&four<3)
                begin
                    ecc_count<='d0;
                    x_count<='d0;
                    four<=four+'d1;
                    ready<=0;
                end
           else if (ecc_count<3&&x_count==rad&&four<3)
                begin
                    ecc_count<=ecc_count+1;
                    x_count<=x_count;
                    four<=four ;
                    ready<='d0;
                end
           else if (ecc_count<3&&x_count<rad&&four==3)
                begin
                    ecc_count<=ecc_count+'d1;
                    x_count<=x_count;
                    four<=four;
                    ready<=0;
                end
           else if (ecc_count==3&&x_count<rad&&four==3)
                begin
                    ecc_count<=0;
                    x_count<=x_count+'d1;
                    four<=four ;
                    ready<='d0;
                end
           else if (ecc_count<3&&x_count==rad&&four==3)
```

```
                begin
                    ecc_count<=ecc_count+1;
                    x_count<=x_count;
                    four<=four ;
                    ready<='d0;
                end
            else if (ecc_count==3&&x_count==rad&&four==3)
                begin
                    ecc_count<=ecc_count;
                    x_count<=x_count;
                    four<=four ;
                    ready<='d1;
                end
            else
                begin
                    ecc_count<=ecc_count;
                    x_count<=x_count;
                    four<=four ;
                    ready<=ready;
                end


        end//end always

//------------------------- rom address && data ---------------------------
    always@*
        begin
            addr_rom_h =  (2+rad)*(rad-1)/2 + x_count;
            if (four==2'd2)
                addr_rom_f = (rad+1)*(rad-1)+ (rad + x_count[7:0]);
                else if (four==2'd3)
                addr_rom_f = (rad+1)*(rad-1)+ ( rad - x_count[7:0]);
                else if (four==2'd0&&y_in_b<=y_in_st+'d12)
                addr_rom_b = (rad+1)*(rad-1)+rad - x_count[7:0];
            else if (four==2'd0&&y_in_b>y_in_st+'d12)
                addr_rom_b = (rad+1)*(rad-1)+rad + x_count[7:0];
            else if (four==2'd1&&y_in_b<=y_in_st+'d12)
                addr_rom_b = (rad+1)*(rad-1)+rad + x_count[7:0];
            else if (four==2'd1&&y_in_b>y_in_st+'d12)
                addr_rom_b = (rad+1)*(rad-1)+rad - x_count[7:0];
        end//end always

    always@*
        begin
            if (four==2'd2 || four ==2'd3)
                data_rom_real = data_rom_f;
            else if (four ==2'd0||four==2'd1)
                data_rom_real = data_rom_b;
            else ;
        end //always




//--------------------------ram window address-------------------
    always@*
        begin
            cent_y =(y_in_b-65);
            if (four==2'd2)
                begin
                    win_y = (cent_y + data_rom_h+ecc_count)% win_h;
                    win_x = win_w/2 + x_count;
                    addr_win = win_y*win_w + win_x;
                end
            else if (four==2'd3)
                begin
                    win_y = (cent_y + data_rom_h+ecc_count)% win_h;
                    win_x = win_w/2 - x_count;
                    addr_win = win_y*win_w + win_x;
                end
            else if (four==2'd0)
                begin
                    win_y = (cent_y - data_rom_h+ecc_count)% win_h;
                    win_x = win_w/2 + x_count;
                    addr_win = win_y*win_w + win_x;
                end
            else if (four==2'd1)
                begin
                    win_y = (cent_y - data_rom_h+ecc_count)% win_h;
                    win_x = win_w/2 - x_count;
                    addr_win = win_y*win_w + win_x;
                end
            else
                begin
                    win_y=win_y;
                    win_x=win_x;
                    addr_win=addr_win;
                end
    end



//-------------------------write data to win----------------------------------

    always@*
        begin
            if (clk25==1&&clk50==0)
                begin
                    rden=1;
                    wren=0;
                end
            else if (y_in_b<100)
                begin
                    rden=0;
                    wren=0;
                end
            else      if(data_valid==8'd0       &&       y_in_b>100&&y_in_b<378&&
clk25==0&&clk50==0)
                begin
                    data_win = data_rom_real;
                    rden = 0;
                    wren = 1;
                end
            else if(data_valid==8'd0 && y_in_b==100&& clk25==0&&clk50==0)
                begin
                    data_win = data_rom_real+8;
                    rden = 0;
                    wren = 1;
                end
            else
                begin
                    rden = 0;
                    wren = 0;
                end
        end //always


endmodule
```

inter_face_top.sv

```
module
inter_face_top(clk,rst,muxsel,ctrl_in,din0,din1,din2,din3,din4,din5,din6,dout);
  input                clk,rst;
  input     [4:0]      ctrl_in;
  input     [7:0]      muxsel;
  input     [7:0]   din0,din1,din2,din3,din4,din5,din6;
  output    [7:0]   dout;
  logic     [7:0]   dmid[0:139];




   inter_face
inter_face(clk,rst,ctrl_in,din0,din1,din2,din3,din4,din5,din6,dmid);

   mux140 mux140(dmid,muxsel,dout);

endmodule// inter_face_top
```

inter_face.sv

```
typedef logic [7:0] dataout[0:139];
module inter_face(clk,rst,ctrl_in,din0,din1,din2,din3,din4,din5,din6,dout);
  input                clk,rst;
  input     [4:0]   ctrl_in;
  input     [7:0]   din0,din1,din2,din3,din4,din5,din6;
  output            dataout dout;
  wire      [139:0]    ctrl_out;

   decoder decoder(clk,rst,ctrl_in,ctrl_out);

   register_file
register_file(clk,rst,ctrl_out,din0,din1,din2,din3,din4,din5,din6,dout);

endmodule // inter_face
```

decoder.sv

```
module decoder(clk,rst,din,ctrl_out);
  input[4:0]        din;
  input         clk,rst;
  output    [139:0]    ctrl_out;

   always@ (negedge clk)
       begin
           if (rst)
               ctrl_out<=140'd0;
           else
               begin
                   case(din)
```

```
                        5'd19:ctrl_out<={7'b1111_111,133'b0};
                        5'd18:ctrl_out<={7'b0,7'b1111_111,126'b0};
                        5'd17:ctrl_out<={14'b0,7'b1111111,119'b0};
                        5'd16:ctrl_out<={21'b0,7'b1111111,112'b0};
                        5'd15:ctrl_out<={28'b0,7'b1111111,105'b0};
                        5'd14:ctrl_out<={35'b0,7'b1111111,98'b0};
                        5'd13:ctrl_out<={42'b0,7'b1111111,91'b0};
                        5'd12:ctrl_out<={49'b0,7'b1111111,84'b0};
                        5'd11:ctrl_out<={56'b0,7'b1111111,77'b0};
                        5'd10:ctrl_out<={63'b0,7'b1111111,70'b0};
                        5'd9:ctrl_out<={70'b0,7'b1111111,63'b0};
                        5'd8:ctrl_out<={77'b0,7'b1111111,56'b0};
                        5'd7:ctrl_out<={84'b0,7'b1111111,49'b0};
                        5'd6:ctrl_out<={91'b0,7'b1111111,42'b0};
                        5'd5:ctrl_out<={98'b0,7'b1111111,35'b0};
                        5'd4:ctrl_out<={105'b0,7'b1111111,28'b0};
                        5'd3:ctrl_out<={112'b0,7'b1111111,21'b0};
                        5'd2:ctrl_out<={119'b0,7'b1111111,14'b0};
                        5'd1:ctrl_out<={126'b0,7'b1111111,7'b0};
                        5'd0:ctrl_out<={133'b0,7'b1111111};
                        default:ctrl_out<=140'b0;
                    endcase // case (din)
                end // end else
        end // always_comb
endmodule // decoder
```

register_file.sv

```
typedef logic [7:0] dataout[0:139];
module register_file(clk,rst,en,din0,din1,din2,din3,din4,din5,din6,dout);
   input                clk,rst;
   input     [139:0]    en;
   input     [7:0]   din0,din1,din2,din3,din4,din5,din6;
   output               dataout dout;

    genvar   n0;
   generate
     for (n0=0;n0<=139;n0=n0+7)begin:reg_file0
            register register(clk,rst,en[n0],din0,dout[n0]);
     end
     endgenerate
   genvar    n1;

    generate
     for (n1=1;n1<=139;n1=n1+7)begin:reg_file1
            register register(clk,rst,en[n1],din1,dout[n1]);
     end
     endgenerate

    genvar   n2;
   generate
     for (n2=2;n2<=139;n2=n2+7)begin:reg_file2
            register register(clk,rst,en[n2],din2,dout[n2]);
     end
     endgenerate

   genvar    n3;
   generate
     for (n3=3;n3<=139;n3=n3+7)begin:reg_file3
            register register(clk,rst,en[n3],din3,dout[n3]);
     end
     endgenerate
   genvar    n4;
   generate
     for (n4=4;n4<=139;n4=n4+7)begin:reg_file4
            register register(clk,rst,en[n4],din4,dout[n4]);
     end
     endgenerate

    genvar   n5;
   generate
     for (n5=5;n5<=139;n5=n5+7)begin:reg_file5
            register register(clk,rst,en[n5],din5,dout[n5]);
     end
     endgenerate

    genvar   n6;
   generate
     for (n6=6;n6<=139;n6=n6+7)begin:reg_file6
            register register(clk,rst,en[n6],din6,dout[n6]);
     end
     endgenerate
endmodule // register_file


register.sv
module register(clk,rst,en,din,dout);
   input               clk,rst,en;
   input    [7:0]   din;
   output   [7:0]   dout;

   always_ff@(posedge clk)
        begin
            if(rst)
                dout<=0;
            else if(en)
                dout<=din;
            else
                dout<=dout;
        end
endmodule // register
```

mux140.sv

```
typedef logic [7:0] datain[0:139];
module mux140(din,sel,dout);
   input [7:0]sel;
    //input clk,rst;
   input datain din;
   output [7:0]  dout;
   always@* begin
     case(sel)
    0:dout=din[0];
    1:dout=din[1];
    2:dout=din[2];
    3:dout=din[3];
    4:dout=din[4];
    5:dout=din[5];
    6:dout=din[6];
    7:dout=din[7];
    8:dout=din[8];
    9:dout=din[9];
    10:dout=din[10];
```

```
11:dout=din[11];
12:dout=din[12];
13:dout=din[13];
14:dout=din[14];
15:dout=din[15];
16:dout=din[16];
17:dout=din[17];
18:dout=din[18];
19:dout=din[19];
20:dout=din[20];
21:dout=din[21];
22:dout=din[22];
23:dout=din[23];
24:dout=din[24];
25:dout=din[25];
26:dout=din[26];
27:dout=din[27];
28:dout=din[28];
29:dout=din[29];
30:dout=din[30];
31:dout=din[31];
32:dout=din[32];
33:dout=din[33];
34:dout=din[34];
35:dout=din[35];
36:dout=din[36];
37:dout=din[37];
38:dout=din[38];
39:dout=din[39];
40:dout=din[40];
41:dout=din[41];
42:dout=din[42];
43:dout=din[43];
44:dout=din[44];
45:dout=din[45];
46:dout=din[46];
47:dout=din[47];
48:dout=din[48];
49:dout=din[49];
50:dout=din[50];
51:dout=din[51];
52:dout=din[52];
53:dout=din[53];
54:dout=din[54];
55:dout=din[55];
56:dout=din[56];
57:dout=din[57];
58:dout=din[58];
59:dout=din[59];
60:dout=din[60];
61:dout=din[61];
62:dout=din[62];
63:dout=din[63];
64:dout=din[64];
65:dout=din[65];
66:dout=din[66];
67:dout=din[67];
68:dout=din[68];
69:dout=din[69];
70:dout=din[70];
71:dout=din[71];
72:dout=din[72];
73:dout=din[73];
74:dout=din[74];
75:dout=din[75];
```

```
76:dout=din[76];
77:dout=din[77];
78:dout=din[78];
79:dout=din[79];
80:dout=din[80];
81:dout=din[81];
82:dout=din[82];
83:dout=din[83];
84:dout=din[84];
85:dout=din[85];
86:dout=din[86];
87:dout=din[87];
88:dout=din[88];
89:dout=din[89];
90:dout=din[90];
91:dout=din[91];
92:dout=din[92];
93:dout=din[93];
94:dout=din[94];
95:dout=din[95];
96:dout=din[96];
97:dout=din[97];
98:dout=din[98];
99:dout=din[99];
100:dout=din[100];
101:dout=din[101];
102:dout=din[102];
103:dout=din[103];
104:dout=din[104];
105:dout=din[105];
106:dout=din[106];
107:dout=din[107];
108:dout=din[108];
109:dout=din[109];
110:dout=din[110];
111:dout=din[111];
112:dout=din[112];
113:dout=din[113];
114:dout=din[114];
115:dout=din[115];
116:dout=din[116];
117:dout=din[117];
118:dout=din[118];
119:dout=din[119];
120:dout=din[120];
121:dout=din[121];
122:dout=din[122];
123:dout=din[123];
124:dout=din[124];
125:dout=din[125];
126:dout=din[126];
127:dout=din[127];
128:dout=din[128];
129:dout=din[129];
130:dout=din[130];
131:dout=din[131];
132:dout=din[132];
133:dout=din[133];
134:dout=din[134];
135:dout=din[135];
136:dout=din[136];
137:dout=din[137];
138:dout=din[138];
139:dout=din[139];
default:dout=8'b0;
```

```
      endcase // case (din)

   end // always_comb
endmodule // mux140
```

2. Software Part

hello_with_datamouse.c

```
/*
 * Userspace program that communicates with the led_vga device driver
 * primarily through ioctls
 *
 * ChengXue Qian
 * XiaoWen Han
 * Hao Jin
 * MuQing Liu
 * Columbia University
 */

#include <stdlib.h>
#include <stdio.h>
#include "vga_led.h"
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>

#include <math.h>

#include "usbmouse.h"



struct libusb_device_handle *mouse;
uint8_t endpoint_address;
int vga_led_fd;

int mouse_y=0;    //to calculate mouse displacement


int CtrlP[17]={17};
int score=0;
int dis=0;
int simple=20;
int start=16;
int hcnt=0;
int vcnt=0;
float startTemp=16;
int startTemp0=0;
int change=0;
int mode=0;
int example=1;

float base0[10]={0.6944,0.4879,0.3270,0.2061,0.1193,0.0612,0.0258,0.0077,0.0010,
    0.0000};
float base1[10]={2.7778,2.7293,2.5953,2.3927,2.1388,1.8505,1.5451,1.2395,0.9510,
    0.6965};
float base2[10]={0.6944,0.9485,1.2368,1.5423,1.8478,2.1363,2.3906,2.5937,2.7284,
    2.7778};
float base3[10]={0.0000,0.0009,0.0086,0.0257,0.0608,0.1188,0.2052,0.3258,0.4863,
    0.6923};
```

```
int min(int a, int b)
{
  if (a<=b)return a;
  else return b;
}

int poww(int a)
{
  if(a==0) return 1;
  else return -1;
}

struct coordinate
{
  int a[140];
}y;

struct coordinate getY()
{


  int i=0;
  int j=0;
  float y0[140]={17};
  int index;
    for(i=0; i<14; i++){
      for(j=0;j<10;j++){


     index=10*i+j;
     y0[index]=base0[j]*CtrlP[i]+base1[j]*CtrlP[i+1]+base2[j]*CtrlP[i+2]+base3[j]
*CtrlP[i+3];
       if (y0[index]-(int)y0[index]>=0.5){
         y.a[index]=y0[index]+1;
           }
       else{
         y.a[index]=y0[index];
           }
         }
    }
    return y;
      }




////////updata CtrlP and the modifeid ellipse according to the change of CtrlP
struct coordinate changeY()


/*change=0:nochange  change=1:increase  radius  by2  change=2:decrease  radius  by2
start=position_changed start=0,1,2,...,16*/

{
  if(change!=1&&change!=2)
    {
    }else if (change==1&&CtrlP[start]>18){
  }else if (change==2&&CtrlP[start]< 3){
  }
   else{
    float y0[140]={17};
      int i=0;
     int j=0;
       if(change==1&&CtrlP[start]<19){CtrlP[start]+=2;}
       if(change==2&&CtrlP[start]> 2){CtrlP[start]-=2;}
       int index;
       int end=start-4;
       startTemp0=min(13,start);
       for(i=startTemp0; i>=0&&i>end; i--){
         for(j=0;j<10;j++){
           index=10*i+j;

     y0[index]=base0[j]*CtrlP[i]+base1[j]*CtrlP[i+1]+base2[j]*CtrlP[i+2]+base3[j]
*CtrlP[i+3];
           if (y0[index]-(int)y0[index]>=0.5){
             y.a[index]=y0[index]+1;
               }
           else{
             y.a[index]=y0[index];
               }

         }
       }
  }
  return y;
    }




/* Read and print the segment values */
void print_segment_info() {
  vga_led_arg_t vla;
  int i;

for (i = 0 ; i < VGA_LED_DIGITS ; i++) {
    vla.digit = i;
    if (ioctl(vga_led_fd, VGA_LED_READ_DIGIT, &vla)) {
      perror("ioctl(VGA_LED_READ_DIGIT) failed");
      return;
    }

  }

}




/* Write the contents of the array to the display */
void write_segments(const unsigned char segs[13])
{
  vga_led_arg_t vla;
  int i;
  for (i = 0 ; i < VGA_LED_DIGITS ; i++) {
    vla.digit = i;
    vla.segments = segs[i];
    if (ioctl(vga_led_fd, VGA_LED_WRITE_DIGIT, &vla)) {
      perror("ioctl(VGA_LED_WRITE_DIGIT) failed");
      return;
    }
  }
}
```

```
int main()
{
  struct usb_mouse_packet packet;
  int transferred;
  char keystate[6];
  unsigned char message0[20][8]={0};
  unsigned char message[11]={0};
  mouse=openmouse(&endpoint_address);
  int clear_key = 1;
  vga_led_arg_t vla;
  static const char filename[] = "/dev/vga_led";// the driver.c
      if ( (vga_led_fd = open(filename, O_RDWR)) == -1) {
      fprintf(stderr, "could not open %s\n", filename);
      return -1;
    }

    ;
    print_segment_info();

    write_segments(message);

    print_segment_info();

    int seed=4;
    int rad=20;
    int i;
    int j;
    int k;
    int mes_1;
    int mes_2;
    int index;



    srand(seed);
     for (i=3;i<17;i++){
       CtrlP[i]=rand()%rad+1;
      }
       y=getY();

       CtrlP[0]=17;
       CtrlP[1]=8;
    CtrlP[2]=5;

    while(1){
      libusb_interrupt_transfer(mouse, endpoint_address,(unsigned char *) &packet,
                   sizeof(packet),&transferred, 0);


      change=0;

      if(packet.various_func==1){change=1;}
      if(packet.various_func==2){change=2;}

      if(mode==0){
     if(example==1 && packet.add0==0x01){example=4;}
     else if(example==1 && packet.add0==0xff){example=2;}
     else if(example==2 && packet.add0==0x01){example=1;}
     else if(example==2 && packet.add0==0xff){example=3;}
     else if(example==3 && packet.add0==0x01){example=2;}
     else if(example==3 && packet.add0==0xff){example=4;}
     else if(example==4 && packet.add0==0x01){example=3;}
     else if(example==4 && packet.add0==0xff){example=1;}
      }



      mouse_y=(128-abs(128-packet.Y_displacement));
      ;
      if(mouse_y>=3)start=start+poww((packet.Y_displacement & 0x80)>>7);
      if(start<3){start=3;}
      if(start>16){start=16;}


     y=changeY();

     startTemp0=min(start,13);
     vcnt=10+2*startTemp0;   //scale down 10 times
     hcnt=y.a[10*startTemp0+1];  ///no center offset 320

     if(mode==0 && packet.various_func==4){mode=1;}

     score=0;
     dis=0;

     if(mode==1 && packet.various_func==4){
        if(example==1){    //simple

 dis=abs(CtrlP[3]-simple)+abs(CtrlP[4]-simple)+abs(CtrlP[5]-simple)+abs(CtrlP[6]
 -simple)
     +abs(CtrlP[7]-simple)+abs(CtrlP[8]-simple)+abs(CtrlP[9]-simple)+abs(CtrlP[10
 ]-simple)
     +abs(CtrlP[11]-simple)+abs(CtrlP[12]-simple)+abs(CtrlP[13]-simple)+abs(CtrlP
 [14]-simple)            +abs(CtrlP[15]-simple)+abs(CtrlP[16]-simple);
         printf("(should less than 10) dis is %d  \n",dis);
         if (dis<10)score=1;
         else score=2;}

       if(example==2){     //middle

 dis=abs(CtrlP[16]-CtrlP[15])+abs(CtrlP[15]-CtrlP[14])+abs(CtrlP[14]-CtrlP[13])

     +abs(CtrlP[13]-CtrlP[12])+abs(CtrlP[12]-CtrlP[11])+abs(CtrlP[11]-CtrlP[10])

     +abs(CtrlP[10]/CtrlP[9]-0.8)+abs(CtrlP[9]/CtrlP[8]-0.8)+abs(CtrlP[8]/CtrlP[7
 ]-
     0.8)+abs(CtrlP[7]/CtrlP[6]-1)+abs(CtrlP[6]/CtrlP[5]-1)+abs(CtrlP[5]/CtrlP[4]
 -              1.2)+abs(CtrlP[4]/CtrlP[3]-1.3);
         printf("(should less than 15)dis is %d  \n",dis);
         if (dis<15)score=1;
         else score=2;}

       if(example==3){      //hard

 dis=abs(y.a[140]/y.a[130]-1.15)+abs(y.a[130]/y.a[120]-1.08)+abs(y.a[120]/y.a[11
 0]-
     0.96)+abs(y.a[110]/y.a[100]-0.79)+abs(y.a[100]/y.a[90]-0.66)+abs(y.a[90]/y.a
 [80]-
     1.04)+abs(y.a[80]/y.a[70]-1.45)+abs(y.a[70]/y.a[60]-0.92)+abs(y.a[60]/y.a[50
 ]-
     0.72)+abs(y.a[50]/y.a[40]-2.17)+abs(y.a[40]/y.a[30]-0.74)+abs(y.a[30]/y.a[20
 ]-0.94);
         printf("(should less than 6) dis is %d  \n",dis);
         if (dis<3)score=1;
         else score=2;}
     }
     index=0;


       for (mes_1=0;mes_1<20;mes_1++){
         for (mes_2=0;mes_2<8;mes_2++){
           if(mes_2==0){
          message0[mes_1][mes_2]=mes_1;

            }else{
          message0[mes_1][mes_2]=y.a[index];
            index=index+1;

              }
          }
           }


         print_segment_info();

           for(j=0;j<20;j++){

          for(k=0;k<8;k++){
            message[k]=message0[j][k];
              }


          message[8]=hcnt;
          message[9]=vcnt;
          message[10]=mode;
          message[11]=example;
          message[12]=score;

          write_segments(message);

            }

           if (mode==1){
          if (score==1||packet.add0){
            if(score)usleep(5000000);
            else    usleep(500000);
            mode=0;
            for (i=3;i<17;i++){
              CtrlP[i]=rand()%rad+1;
                }
              y=getY();

              CtrlP[0]=17;
              CtrlP[1]=8;
              CtrlP[2]=5;
          }
           }


     }



     return 0;
 }
```

usbmouse.c

```
#include "usbmouse.h"

#include <stdio.h>
#include <stdlib.h>

/* References on libusb 1.0 and the USB HID/mouse protocol
 *
 * http://libusb.org
 *
http://www.dreamincode.net/forums/topic/148707-introduction-to-using-libusb-10/
 * http://www.usb.org/developers/devclass_docs/HID1_11.pdf
 * http://www.usb.org/developers/devclass_docs/Hut1_11.pdf
 */

/*
 * Find and return a USB mouse device or NULL if not found
 * The argument con
 *
 */
struct libusb_device_handle *openmouse(uint8_t *endpoint_address) {
  libusb_device **devs;
  struct libusb_device_handle *mouse = NULL;
  struct libusb_device_descriptor desc;
  ssize_t num_devs, d;
  uint8_t i, k;

  /* Start the library */
  if ( libusb_init(NULL) < 0 ) {
    fprintf(stderr, "Error: libusb_init failed\n");
    exit(1);
  }

  /* Enumerate all the attached USB devices */
  if ( (num_devs = libusb_get_device_list(NULL, &devs)) < 0 ) {
    fprintf(stderr, "Error: libusb_get_device_list failed\n");
    exit(1);
  }

  /* Look at each device, remembering the first HID device that speaks
     the mouse protocol */

  for (d = 0 ; d < num_devs ; d++) {
    libusb_device *dev = devs[d];
    if ( libusb_get_device_descriptor(dev, &desc) < 0 ) {
      fprintf(stderr, "Error: libusb_get_device_descriptor failed\n");
      exit(1);
    }

    if (desc.bDeviceClass == LIBUSB_CLASS_PER_INTERFACE) {
      struct libusb_config_descriptor *config;
      libusb_get_config_descriptor(dev, 0, &config);
      for (i = 0 ; i < config->bNumInterfaces ; i++)
     for ( k = 0 ; k < config->interface[i].num_altsetting ; k++ ) {
       const struct libusb_interface_descriptor *inter =
         config->interface[i].altsetting + k ;
       if ( inter->bInterfaceClass == LIBUSB_CLASS_HID &&
            inter->bInterfaceProtocol == USB_HID_MOUSE_PROTOCOL) {
         int r;
         if ((r = libusb_open(dev, &mouse)) != 0) {
           fprintf(stderr, "Error: libusb_open failed: %d\n", r);
```

```
          exit(1);
        }
        if (libusb_kernel_driver_active(mouse,i))
          libusb_detach_kernel_driver(mouse, i);
        /* libusb_set_auto_detach_kernel_driver(mouse, i); */
        if ((r = libusb_claim_interface(mouse, i)) != 0) {
          fprintf(stderr, "Error: libusb_claim_interface failed: %d\n", r);
          exit(1);
        }
        *endpoint_address = inter->endpoint[0].bEndpointAddress;
        goto found;
      }
    }
   }
  }

 found:
  libusb_free_device_list(devs, 1);

  return mouse;
}
```

usbmouse.h

```
#ifndef _USBMOUSE_H
#define _USBMOUSE_H

#include <libusb-1.0/libusb.h>

#define USB_HID_MOUSE_PROTOCOL 2


struct usb_mouse_packet {
  uint8_t various_func;
  uint8_t X_displacement;
  uint8_t Y_displacement;
  uint8_t add0;
  uint8_t add1;
};


extern struct libusb_device_handle *openmouse(uint8_t *);

#endif
```

vga_led.c

```
/*
 * Device driver for the VGA LED Emulator
 *
 * A Platform device implemented using the misc subsystem
 *
 * Stephen A. Edwards
 * Columbia University
 *
 * References:
 * Linux source: Documentation/driver-model/platform.txt
 *               drivers/misc/arm-charlcd.c
 * http://www.linuxforu.com/tag/linux-device-drivers/
 * http://free-electrons.com/docs/
 *
 * "make" to build
 * insmod vga_led.ko
 *
 * Check code style with
 * checkpatch.pl --file --no-tree vga_led.c
 */

#include <linux/module.h>
#include <linux/init.h>
#include <linux/errno.h>
#include <linux/version.h>
#include <linux/kernel.h>
#include <linux/platform_device.h>
#include <linux/miscdevice.h>
#include <linux/slab.h>
#include <linux/io.h>
#include <linux/of.h>
#include <linux/of_address.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#include "vga_led.h"

#define DRIVER_NAME "vga_led"

/*
 * Information about our device
 */
struct vga_led_dev {
    struct resource res; /* Resource: our registers */
    void __iomem *virtbase; /* Where registers can be accessed in memory */
    u8 segments[VGA_LED_DIGITS];
} dev;

/*
 * Write segments of a single digit
 * Assumes digit is in range and the device information has been set up
 */
static void write_digit(int digit, u8 segments)
{
    iowrite8(segments, dev.virtbase + digit);
    dev.segments[digit] = segments;
}

/*
 * Handle ioctl() calls from userspace:
 * Read or write the segments on single digits.
 * Note extensive error checking of arguments
 */
static long vga_led_ioctl(struct file *f, unsigned int cmd, unsigned long arg)
{
    vga_led_arg_t vla;

    switch (cmd) {
    case VGA_LED_WRITE_DIGIT:
        if (copy_from_user(&vla, (vga_led_arg_t *) arg,
                   sizeof(vga_led_arg_t)))
            return -EACCES;
        if (vla.digit > 13)
            return -EINVAL;
        write_digit(vla.digit, vla.segments);
        break;

    case VGA_LED_READ_DIGIT:
        if (copy_from_user(&vla, (vga_led_arg_t *) arg,
                   sizeof(vga_led_arg_t)))
            return -EACCES;
        if (vla.digit > 13)
            return -EINVAL;
        vla.segments = dev.segments[vla.digit];
        if (copy_to_user((vga_led_arg_t *) arg, &vla,
                 sizeof(vga_led_arg_t)))
            return -EACCES;
        break;

    default:
        return -EINVAL;
    }

    return 0;
}



/* The operations our device knows how to do */

static const struct file_operations vga_led_fops = {
    .owner      = THIS_MODULE,
    .unlocked_ioctl = vga_led_ioctl,
};

/* Information about our device for the "misc" framework -- like a char dev */
static struct miscdevice vga_led_misc_device = {
    .minor      = MISC_DYNAMIC_MINOR,
    .name       = DRIVER_NAME,
    .fops       = &vga_led_fops,
};

/*
 * Initialization code: get resources (registers) and display
 * a welcome message
 */
static int __init vga_led_probe(struct platform_device *pdev)
{
    static unsigned char welcome_message[VGA_LED_DIGITS] = {
        0x3E, 0x7D, 0x77, 0x08, 0x38, 0x79, 0x5E, 0x00};
    int i, ret;

    /* Register ourselves as a misc device: creates /dev/vga_led */
    ret = misc_register(&vga_led_misc_device);

    /* Get the address of our registers from the device tree */
    ret = of_address_to_resource(pdev->dev.of_node, 0, &dev.res);
    if (ret) {
        ret = -ENOENT;
        goto out_deregister;
    }

    /* Make sure we can use these registers */
    if (request_mem_region(dev.res.start, resource_size(&dev.res),
                  DRIVER_NAME) == NULL) {
        ret = -EBUSY;
        goto out_deregister;
    }

    /* Arrange access to our registers */
    dev.virtbase = of_iomap(pdev->dev.of_node, 0);
    if (dev.virtbase == NULL) {
        ret = -ENOMEM;
        goto out_release_mem_region;
    }

    /* Display a welcome message */
    for (i = 0; i < VGA_LED_DIGITS; i++)
        write_digit(i, welcome_message[i]);

    return 0;

out_release_mem_region:
    release_mem_region(dev.res.start, resource_size(&dev.res));
out_deregister:
    misc_deregister(&vga_led_misc_device);
    return ret;
}

/* Clean-up code: release resources */
static int vga_led_remove(struct platform_device *pdev)
{
    iounmap(dev.virtbase);
    release_mem_region(dev.res.start, resource_size(&dev.res));
    misc_deregister(&vga_led_misc_device);
    return 0;
}

/* Which "compatible" string(s) to search for in the Device Tree */
#ifdef CONFIG_OF
static const struct of_device_id vga_led_of_match[] = {
    { .compatible = "altr,vga_led" },
    {},
};


MODULE_DEVICE_TABLE(of, vga_led_of_match);
#endif

/* Information for registering ourselves as a "platform" driver */
static struct platform_driver vga_led_driver = {
    .driver = {
        .name   = DRIVER_NAME,
        .owner  = THIS_MODULE,
        .of_match_table = of_match_ptr(vga_led_of_match),
    },
    .remove = __exit_p(vga_led_remove),
};

/* Called when the module is loaded: set things up */
static int __init vga_led_init(void)
```

```
{
    pr_info(DRIVER_NAME ": init\n");
    return platform_driver_probe(&vga_led_driver, vga_led_probe);
}

/* Called when the module is unloaded: release resources */
static void __exit vga_led_exit(void)
{
    platform_driver_unregister(&vga_led_driver);
    pr_info(DRIVER_NAME ": exit\n");
}

module_init(vga_led_init);
module_exit(vga_led_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Stephen A. Edwards, Columbia University");
MODULE_DESCRIPTION("VGA 7-segment LED Emulator");
```

vga_led.h

```
#ifndef _VGA_LED_H
#define _VGA_LED_H

#include <linux/ioctl.h>

#define VGA_LED_DIGITS 13

typedef struct {
  unsigned char digit;    /* 0, 1, .. , VGA_LED_DIGITS - 1 */
  unsigned char segments; /* LSB is segment a, MSB is decimal point */
} vga_led_arg_t;

#define VGA_LED_MAGIC 'q'

/* ioctls and their arguments */
#define VGA_LED_WRITE_DIGIT _IOW(VGA_LED_MAGIC, 1, vga_led_arg_t *)
#define VGA_LED_READ_DIGIT  _IOWR(VGA_LED_MAGIC, 2, vga_led_arg_t *)

#endif
```

DrawEli2Height.m

(Used to draw ellipse with given radius)

```
function  [AllHeights]=DrawEli2HeightAllRadius()
g=DrawEli2Height(1);
i=2;
while i<151
   r1=DrawEli2Height(i);
   temp=g;
   g=[temp r1];
   i=i+1;
end

AllHeights=g;


end



function [ results ] = DrawEli2Height( radius )

x_end=radius;
g=zeros(1,radius+1);

x=1;
y=radius*0.2;
g(1,1)=y;


par1=1/(radius*radius);   %long axis square inverse
par2=par1*5*5;           %short axis square inverse

while x<=x_end
   mid_y=y-0.5;
   if(par1*x*x+par2*mid_y*mid_y-1<0)
      g(1,x+1)=y;
      x=x+1;
   end
   if(par1*x*x+par2*mid_y*mid_y-1>=0)
      g(1,x+1)=y-1;
      x=x+1;
      y=y-1;
   end
end

results=g;
results=uint8(results);

end
```

getEven.m

(used to get the greyscale of the first half ellipse)

```
function [ even ] = getEven( g )
[row col]=size(g);
even=zeros(1,1);
index=1;
for i=1:1:row
   for j=1:1:col;
      if(g(i,j)>0&&mod(g(i,j),2)==0)
         even(1,index)=g(i,j);
         index=index+1;
      end
      if(g(i,j)>0&&mod(g(i,j),2)==1)
         even(1,index)=g(i,j)-1;
         index=index+1;
      end
   end
end


end
```

getOdd.m

used to get the greyscale of the second half ellipse

```
function [ odd ] = getOdd( g )
[row col]=size(g);
odd=zeros(1,1);
index=1;
for i=1:1:row
   for j=1:1:col;
      if(g(i,j)>0&&mod(g(i,j),2)==1)
         odd(1,index)=g(i,j);
         index=index+1;
      end
      if(g(i,j)>0&&mod(g(i,j),2)==0)
         odd(1,index)=g(i,j)-1;
         index=index+1;
      end
   end
end


end
```

rad2greyAfterScale.m

(used to generate the greyscale corresponding to the radius that have been scaled)

```
function [ g ] = rad2greyAfterScale( maxRadius, scale )

%rad=linspace(1,maxRadius,maxRadius);
%number=scale*2*rad;

radAfterScale=linspace(1,maxRadius*scale,maxRadius*scale);
number=2*radAfterScale;
dLinear=20./number;

gHeight=maxRadius*scale;
gWidth=max(number)+1;


g=zeros(gHeight,gWidth);

for i=1:1:gHeight
   a=0;
   for j=1:1:gWidth
      if(j<=i)
         g(i,j)=a*a+40;
         a=a+dLinear(1,i);
      end
   end
end

for i=1:1:gHeight
   a=10;
   for j=1:1:gWidth
      if(j>i)
         if(j<=2*i)
            g(i,j)=200-(a-20)^2+40;
            a=a+dLinear(1,i);
         end
         if(j==2*i+1)
            g(i,j)=g(i,j-1);
         end
      end
   end
end
imshow(uint8(g));
g=uint8(g);

end
```

rad2greyAfterScale.m

(used to generate the greyscale for the back half of ellipse corresponding to the radius after scaled)

```
function [ g ] = rad2greyBackAfterScale( maxRadius, scale )



%rad=linspace(1,maxRadius,maxRadius);
%number=scale*2*rad;

radAfterScale=linspace(1,maxRadius*scale,maxRadius*scale);
number=2*radAfterScale;
dLinear=20./number;
dBack=pi./number;

gHeight=maxRadius*scale;
gWidth=max(number)+1;


g=zeros(gHeight,gWidth);

for i=1:1:gHeight
    a=0;
    b=0;
    for j=1:1:gWidth
        if(j<=i)
            g(i,j)=(a*a+40)*(1-0.25*sin(b));
            a=a+dLinear(1,i);
            b=b+dBack(1,i);
        end
    end
end

for i=1:1:gHeight
    a=10;
    b=0.5*pi;
    for j=1:1:gWidth
        if(j>i)
            if(j<=2*i)
                g(i,j)=(200-(a-20)^2+40)*(1-0.25*sin(b));
                a=a+dLinear(1,i);
                b=b+dBack(1,i);
            end
            if(j==2*i+1)
                g(i,j)=g(i,j-1);
            end
        end
    end
end

imshow(uint8(g));

g=uint8(g);


end
```

VaseWithCtrl.m

(used to verify our algorithm of pottert-modeling)

```
function                [                 g                 ,g1]                =
VaseWithCtrl( theta,ControlPoints,MaxRadius,HeightPixels,WidthPixels )

if nargin<5, WidthPixels=300;        end
if nargin<4, HeightPixels=400;       end
if nargin<3, MaxRadius=24;           end
if nargin<2, ControlPoints=[rand(1,3)*0.3+0.1 rand(1,5)*0.6+0.2 rand(1,4)*0.5+0.1
rand(1,5)*0.3+0.1]; end
if nargin<1, theta=0;               end

greyMaxX=HeightPixels;
greyMaxY=WidthPixels;
rad=MaxRadius;
[~, ConP]=size(ControlPoints);
angle=mod(round(48*theta+1),97);


ControlPoints(1,ConP)=0.7;
ControlPoints(1,ConP-1)=0.7*0.5;
ControlPoints(1,ConP-2)=0.7*0.3;


g=zeros(greyMaxX,greyMaxY);
a=[linspace(10,90,ConP);floor(rad*ControlPoints)];

plot(a(1,:),a(2,:),':');

B=ConP-3;

par=2;
ecc=0.1*ceil(100/rad);
k1_u=roundn(1/(ecc^2),-2)+0.01;
k1_d=roundn(1/(ecc^2),-2)-0.01;
x_bound=[a(1,1)-rad*ecc a(1,ConP)+rad*ecc];
y_bound=[-rad rad];
x=zeros(1,B*10);
y=zeros(1,B*10);
x_line=zeros(1,B*10);
y_line=zeros(1,B*10);




x_scale=floor(greyMaxX/x_bound(1,2));
y_scale=floor(0.5*greyMaxY/y_bound(1,2));
x_offset=0;
y_offset=y_scale*y_bound(1,2);
x_halfWidth=2;
y_halfWidth=5;

for i=1:B;
   for u=0:0.111:0.999;
      b0=1.0./6.*(1-u).^3;
      b1=1.0./6.*(3.*u.^3-6.*u.^2+4);
      b2=1.0./6.*(-3.*u.^3+3.*u.^2+3.*u+1);
      b3=1.0./6.*u.^3;
      j=10*(i-1)+floor(10*u)+1;
```

```
        x(1,j)=b0.*a(1,i)+b1.*a(1,i+1)+b2.*a(1,i+2)+b3.*a(1,i+3);
        y(1,j)=b0.*a(2,i)+b1.*a(2,i+1)+b2.*a(2,i+2)+b3.*a(2,i+3);

        t0=pi/2:pi/48:2.5*pi;x0=ecc*y(1,j)*cos(t0)+x(1,j);y0=y(1,j)*sin(t0);
        x_line(1,j)=x0(angle);
        y_line(1,j)=y0(angle);
        for k=1:97
            for z=1:(j-1)
                if j>1
                    par=k1_u*(x0(k)-x(1,j))^2/(y(1,j)^2)+y0(k)^2/(y(1,j)^2);
                end
                if par>1
                    x2greyStart=round(x0(k)*x_scale)+x_offset-x_halfWidth;
                    y2greyStart=round(y0(k)*y_scale)+y_offset-y_halfWidth;
                    x2greyEnd=x2greyStart+2*x_halfWidth;
                    y2greyEnd=y2greyStart+2*y_halfWidth;
                    if k<50
                        if k==angle

g(x2greyStart:x2greyEnd,y2greyStart:y2greyEnd)=20+50*sin(0.01*k*pi);
                        else

g(x2greyStart:x2greyEnd,y2greyStart:y2greyEnd)=245*sin(0.01*k*pi);
                        end
                    else
                        if k==angle

g(x2greyStart:x2greyEnd,y2greyStart:y2greyEnd)=20+50*sin(0.01*k*pi)*cos(0.01*(k
-49.5)*pi);
                        else

g(x2greyStart:x2greyEnd,y2greyStart:y2greyEnd)=245*sin(0.01*k*pi)*cos(0.01*(k-4
9.5)*pi);
                        end
                    end
                end
            end
        end
    end
end

bore_x=[x(1,B*10)-ecc*y(1,B*10) x(1,B*10) x(1,B*10)+ecc*y(1,B*10)];
bore_y=[-y(1,B*10) 0 y(1,B*10)];
g_bore_x=round(bore_x*x_scale)+x_offset;
g_bore_y=round(bore_y*y_scale)+y_offset;
g_line_x=round(x_line*x_scale)+x_offset;
g_line_y=round(y_line*y_scale)+y_offset;

if angle>49
    for i=1:1:B*10

par1=(k1_d*(g_line_x(1,i)-g_bore_x(1,2)-x_halfWidth)^2/((x_scale*y(1,B*10))^2)+
(g_line_y(1,i)-g_bore_y(1,2)-y_halfWidth)^2/((y_scale*y(1,B*10))^2));
        if par1<1

g(g_line_x(1,i)-x_halfWidth:g_line_x(1,i)+x_halfWidth,g_line_y(1,i)-y_halfWidth
:g_line_y(1,i)+y_halfWidth)=245*sin(0.01*angle*pi)*cos(0.01*(angle-49.5)*pi);
        end
    end
end

for xb=g_bore_x(1,1)+x_halfWidth:1:g_bore_x(1,3)+x_halfWidth
    for yb=g_bore_y(1,2):1:g_bore_y(1,3)+y_halfWidth

par2=(k1_d*(xb-g_bore_x(1,2)-x_halfWidth)^2/((x_scale*y(1,B*10))^2)+(yb-g_bore_
y(1,2)-y_halfWidth)^2/((y_scale*y(1,B*10))^2));
        if par2<1
            temp=g(xb,yb);
            g(xb,yb)=g(xb,2*y_offset-yb);
            g(xb,2*y_offset-yb)=temp;
        end
    end
end


g=flipud(g);
imshow(uint8(g));
g1=y;
end
```

transfer_pic_grayscale.m

(use to transfer picture.jpg to rom.mif)

```
function [outfname, rows, cols] = transfer_pic_grayscale(infile,outfname, numrows,
numcols)

    img = imread(infile);

    imgresized = imresize(img, [numrows numcols]);

    [rows, cols, rgb] = size(imgresized);

    imgscaled = imgresized;
    imshow(imgscaled);

     fid = fopen(outfname,'w');

     fprintf(fid,'-- %3ux%3u 24bit image color values\n\n',rows,cols);
     fprintf(fid,'WIDTH = 24;\n');
     fprintf(fid,'DEPTH = %4u;\n\n',rows*cols);
     fprintf(fid,'ADDRESS_RADIX = UNS;\n');
     fprintf(fid,'DATA_RADIX = UNS;\n\n');
     fprintf(fid,'CONTENT BEGIN\n');

     count = 0;
     for r = 1:rows
         for c = 1:cols
             grayscale = uint32(imgscaled(r,c,1));

            color = grayscale
            fprintf(fid,'%4u : %u;\n',count, color);
            count = count + 1;
        end
    end

     fprintf(fid,'END;');
    fclose(fid);
```