Virtex2D Graphics Accelerator

Philip Amberg and Andrew Giles
E168b: Hardware/Software Codesign

Final Report
5/2/2007

Abstract

A 2D Graphics accelerator was implemented on a Virtex II-Pro FPGA/PowerPC. This graphics accelerator has the capability to draw points, lines, triangles, and convex N-Gons. This functionality was implemented with a combination of C and Verilog code. Software was generated to test drawing algorithms and validate the hardware implementation of these algorithms. The line drawing algorithm was successfully implemented in hardware and the triangle drawing algorithm was partially successful in hardware but did not match completely with its software counterpart.

Introduction

This report discusses the design and implementation of a 2D graphics accelerator module using the Virtex-II FPGA and the onboard VGA graphics chip. This graphics module can be used in the future development of a video game on the Virtex-II platform. The accelerator consists of software and hardware modules to draw points, lines, triangles, and convex N-gons. Our approach to the problem was to create a software model of our graphics algorithms in MATLAB where we can easily debug code and view the results. The next step was to port the MATLAB code to C, where it could be tested in a standard PC console, then in HyperTerminal on the PowerPC. The next objective was to translate the software algorithms into hardware to accelerate the drawing of shapes and free up processing time on the PowerPC.

Software

To draw lines and triangles, we looked for algorithms to perform these functions using integer operations and minimal multiplication. When suitable algorithms were found, they were implemented with MATLAB to test their functionality and make optimizations in an environment friendly to debugging. After the algorithms were fully optimized, they were ported to C, tested in the console on a PC, then downloaded to the PowerPC on the Virtex boards for further testing.

To draw lines we implemented Bresenham's line algorithm, which can produce lines between two points on raster displays using only additions, subtractions, and a bitshift. The algorithm we implemented was adapted from the Wikipedia article on said algorithm[1]. The algorithm assumes the line moves down and to the right and has a slope between 0 and -1. The algorithm draws the first point, then steps in the x (fast) direction. An error term is computed on each step to determine which pixel in the y direction is closer to the vector line. When the error term crosses a threshold, the algorithm steps in the y (slow) direction and draws a pixel as in Figure 1.

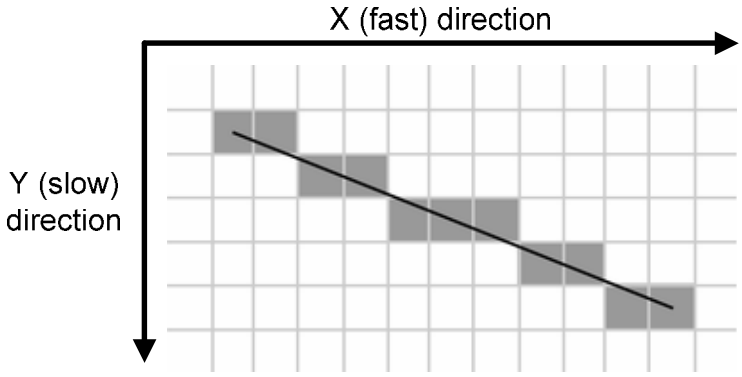

Figure 1: Sample Results from Bresenham's line algorithm. Every cycle the algorithm steps in the fast direction but only steps in the slow direction when necessary[2].

[1] http://en.wikipedia.org/wiki/Bresenham's_line_algorithm
[2] ibid

As this algorithm only works with the assumptions described above, the endpoints and/or drawing coordinates must be flipped to draw in all 8 octants of a screen.

To draw triangles, we implemented an algorithm using half-space functions to determine whether a pixel was inside a triangle or not. Our algorithm is an optimized version of an algorithm described in an article by Nicolas Capens on DevMaster.net[3]. Half-space functions provide an easy test to whether a pixel is inside a triangle. A triangle with endpoints (x_0,y_0), (x_1,y_1), and (x_2,y_2) can be decomposed into three lines with equations

$$(y-y_o)=\frac{y_1-y_o}{x_1-x_o}(x-x_o) \quad \rightarrow \quad (x_1-x_o)(y-y_o)-(y_1-y_o)(x-x_o)=0$$

$$(y-y_1)=\frac{y_2-y_1}{x_2-x_1}(x-x_1) \quad \rightarrow \quad (x_2-x_1)(y-y_1)-(y_2-y_1)(x-x_1)=0 \quad . \qquad (1)$$

$$(y-y_2)=\frac{y_o-y_2}{x_o-x_2}(x-x_2) \quad \rightarrow \quad (x_0-x_2)(y-y_2)-(y_0-y_2)(x-x_2)=0$$

From the right side of equation 1, we can see that the equation evaluates to zero on the line, is positive on side of the line and negative on the other side of the line. This set of equations make up the half-space functions for the algorithm. If we define the vertices of the triangle, (x_0,y_0), (x_1,y_1), and (x_2,y_2) in a counterclockwise order, the half-space functions will all be positive when a point is inside the triangle, providing a simple test for inclusion.

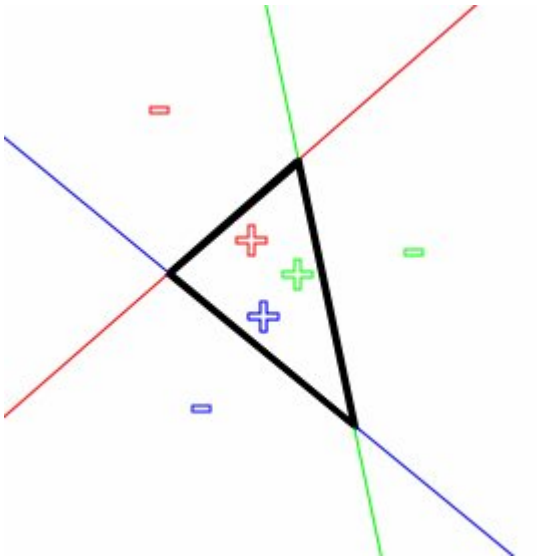

Figure 2: Illustration of how the half-space functions are positive inside a triangle[4].

The most basic form of this algorithm would step through each pixel on a screen, perform the half-space test and either turn that pixel on or off. This algorithm can be sped up by finding the bounding box of the triangle and only checking the pixels within that box. Another speedup can be made by checking the corners of 8x8 pixel blocks to determine whether that block is full, empty, or partially filled. This can be sped up by advancing to

[3] http://www.devmaster.net/forums/showthread.php?t=1884
[4] ibid

the next row when an empty block is detected after a block with drawn pixels because no other blocks in the row will have pixels drawn, shown in Figure 3. After these methodological optimizations have been implemented, the algorithm must be computationally optimized to reduce the complexity and number of mathematical operations. This reduction will allow for a simpler hardware implementation of the algorithm.

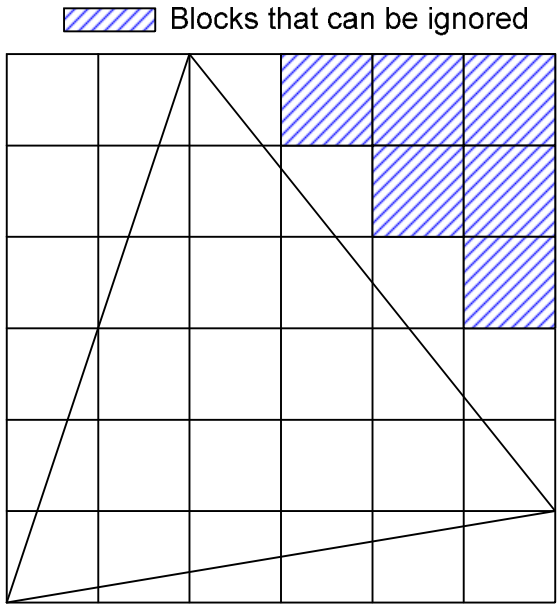

Figure 3: Illustration of how the algorithm can skip to the next row after detecting an empty block after a drawn block. Once the empty block is detected, the blocks after it in that row will also be empty.

The original algorithm had 6 multiplications outside of the pixel testing and drawing loop and 30 multiplications within. Each multiply on the Virtex PowerPC takes 4 cycles to execute so a reduction in the amount of multiplications lead to significant speedups. We discovered that with the addition of 6 multiplies outside the loop, the multiplies within the loop could be removed. The optimized algorithm uses 12 total multiplies per triangle which is an immense improvement over the previous algorithm and should provide significant speedups.

After the line and triangle drawing algorithms were completed, they were implemented in C and a test program was made to test the capabilities of the algorithm. The first generation test program displayed its results in ASCII characters in the console. The program had the capability to draw points, lines, triangles, and convex N-Gons using multiple triangles. When this functionality was adequately tested the test program was interfaced with the framebuffer and the VGA monitor. In this second generation test program a benchmark test was added to test the speed of the algorithm. This test simply filled the screen with a known number of triangles so it can be timed to determine its performance.

The benchmark test was able to fill the screen with triangles at a rate of 1000 triangles per second. This is an average of 300000 cycles per triangle. Each triangle has 1250 pixels so the software executes 240 cycles per pixel. However, most of this execution time is spent writing the data to the framebuffer. The code Xilinx Platform

Studio generates to interface with hardware peripherals is slow. When the code to write to the framebuffer is removed, the performance is increased to 5400 triangles per second, or 55000 cycles per triangle, or 44 cycles per pixel. Even at 44 cycles per pixel, hardware acceleration can significantly improve execution time.

Hardware

The first hardware we implemented was a framebuffer that allowed for a screen size of 640x480x4bit pixels. This framebuffer consisted of a block ram containing 307200 4-bit words, which fed out to a lookup table that converted the 4 bit color values to 24 bit color, and then passed those values out to the external video DAC. The framebuffer generated the necessary Hsync and Vsync signals to drive a VGA monitor.

The necessary inputs to the framebuffer to draw pixels are an address, a color, and a write-enable signal. The address is found by multiplying the line number by 640 and adding the horizontal pixel number:

$$FbAddress = x + 640y \tag{2}$$

where *FbAddress* is the address passed to the framebuffer, and *x* and *y* are the pixel coordinates from the origin in the upper left corner of the monitor.

This fully functional framebuffer was then wrapped inside of a peripheral interface to connect it to the PowerPC processors running on the FPGA. This interface allowed us to pass the pixels out from the software line and triangle generation code to the FPGA, and view the results on an external monitor. This allowed us to verify that our software algorithm functioned correctly before attempting to convert the C code to hardware. See The Guide for more details on how this is accomplished.

Software to Hardware Translation

Once our C code was finalized, we attempted to convert the two primary subroutines into hardware: the fast line and triangle drawing. This was an attempt to meet the goals of our project statement and accelerate the graphical capabilities of the PowerPC microprocessor.

First, we converted the line rendering code. To do this, we implemented a programmable, resettable counter that served as the backbone of the algorithm. This functions as the for loop in the software code. Next, we computed the constants used in Bresenham's line algorithm, and together with the programmable counter, we were able to count in the 'fast' direction of the algorithm. Adding some additional hardware to step in the slow direction at the appropriate time yielded functional line rendering hardware.

The next step was to convert the triangle drawing routine to hardware. This was much more difficult, as the triangle subroutine was significantly more complex than the line drawing subroutine in the code. To split the task up and reduce the complexity of our code, it was split into two parts: the setup constants for the triangle, and the loop that updated the half-space functions and drew pixels.

The constants were generated such that they only require a single cycle to compute, reducing the setup time for the triangle generation as compared to the software model. Once these were computed, they were passed to the loop, which cycled through the pixels, determining which pixels were on, and which fell outside the boundaries of the triangle. After the setup constants are computed, each pixel requires only 1 cycle to run. This corresponds to a 44x speedup from our software algorithm without the framebuffer

and a 240x speedup from the software algorithm with the framebuffer in our benchmark test. This would bring our theoretical performance in our benchmark test to 80000 triangles per second. Further improvements could be made implementing additional parallel processing units with a multi-ported memory. With 8 parallel pixel pipelines we would be able to draw triangles 8 times faster leading to 640000 triangles per second.

Once we had designed the hardware, we compared it to the software model to validate that we had converted it to hardware correctly. We were able to compute the initial constants to match correctly bit for bit, but ran out of time before we could fully debug the loop hardware.

We were unable to implement either of the two hardware accelerated functions as peripherals to attach to the PowerPC cores, as we ran out of time. One thing to note is that coregen components are not easily recognized by XPS, and if they are used additional steps must be taken. Thus, our implementation of hardware multipliers and block memory using coregen actually increased our time spent in the debugging loop, rather than decreasing it as we originally intended. While there are ways around these limitations for the block memory (see The Guide), we were unable to discover a workaround for the multipliers in time.

Conclusion

We successfully implemented software algorithms to draw points, lines, triangles, and convex N-Gons to a VGA monitor. We were also successful in generating hardware to accelerate the drawing of lines and partially successful in generating hardware to generate triangles. Triangles would be drawn to the screen but they did not agree with the triangles produced by the software model. With more time we could have completely debugged the triangle drawing hardware and packaged these modules as peripherals to be controlled by the PowerPC.

While all our objectives were not met we learned a considerable amount about how to interface hardware peripherals with the PowerPC on the Virtex board. Where there have been previous challenges in getting the PowerPC and the FPGA to communicate with each other, we have overcome them and demystified this link. The “Definitive Guide to Hardware Software Interfacing” details the process to create these links.

```c
//Graphics Drawing Routines
//Philip Amberg and Andrew Giles
//4-8-2007
/* This program is capable of drawing points, lines
    quads, and convex n-gon polygons
*/


/* 4/9/07 - need to make program shorter so it fits in memory
          - this will be accomplished by getting rid of scanf (large fn)

    4/10/07 - program now fits in memory, full test program is loaded.
              - no user input available from console, scanf function too large
              - would like to find alternate ways of user input for demo
              - want to integrate the hardware frame buffer as a peripheral
*/
//#include "stdio.h"
//#include "stdlib.h"   //standard c libraries

#include <xparameters.h>    //define io devices
#include <xuartlite_l.h>
//#include "uart.h" //import uart funtions for user input
//#include "xgpio.h"    //define general purpose io function for accessing peripherals
#include "vgadriver.h"


#define printf xil_printf //define printf (shorter than std. printf)

//define datatype vertex
typedef struct vertex
{
    int x;
    int y;
}vertex;

//function prototypes
int triangleFill(vertex v1, vertex v2, vertex v3,  int color, char (screen[20])[50]);
int clearVirtScreen(char (screen[20])[50]);
int printScreen(char (screen[20])[50]);
int printMenu();
int max3(int x1, int x2, int x3);
int min3(int x1, int x2, int x3);
int drawPoint(vertex v1, int color, char (screen[20])[50]);
int drawLine(vertex v1, vertex v2, int color, char (screen[20])[50]);
int drawNGon(vertex verts[], int color, int nGonSides, char (screen[20])[50]);
int drawQuad(vertex v1, vertex v2, vertex v3, vertex v4,  int color, char (screen[20])[50]);

//main routine

int main(void)
{

    vertex v1,v2,v3,v4;     //initialize variables
    int color,i,j,nGonSides;
    char screen[20][50];    //initialize the screen array
    int userChoice;
    int r,x,y;
    int s;

    while(1)                //for always
    {
        clearVirtScreen(screen);        //clear screen

        printMenu();        //print menu
        //printf("What would you like to do: ");

        userChoice=userInput("What would you like to do: ");
        //printf("What would you like to do: ");
        //scanf("%d",&userChoice);


        if(userChoice==0)   //draw triangle
        {

            v1.x=userInput("v1.x=");
```

```
            v1.y=userInput("v1.y=");
            v2.x=userInput("v2.x=");
            v2.y=userInput("v2.y=");
            v3.x=userInput("v3.x=");
            v3.y=userInput("v4.y=");

            /*printf("Input some vertices:\n");
            printf("Vertex 1, x:");
            scanf("%d",&v1.x);
            printf("Vertex 1, y:");
            scanf("%d",&v1.y);
            printf("Vertex 2, x:");
            scanf("%d",&v2.x);
            printf("Vertex 2, y:");
            scanf("%d",&v2.y);
            printf("Vertex 3, x:");
            scanf("%d",&v3.x);
            printf("Vertex 3, y:");
            scanf("%d",&v3.y);*/

            color=15;       //color of triangle

            triangleFill( v1, v2, v3, color, screen);    //draw triangle

            //printScreen(screen);    //draw screen
        }

        else if(userChoice==1)  //draw Quad
        {
            v1.x=userInput("v1.x=");
            v1.y=userInput("v1.y=");
            v2.x=userInput("v2.x=");
            v2.y=userInput("v2.y=");
            v3.x=userInput("v3.x=");
            v3.y=userInput("v3.y=");
            v4.x=userInput("v4.x=");
            v4.y=userInput("v4.y=");

            color=15;        //color of quad

            drawQuad( v1, v2, v3, v4, color, screen);    //draw quad

            //printScreen(screen);    //draw screen
        }
        else if(userChoice==2)   //draw convex N-gon
        {
            nGonSides=userInput("How many sides: ");    //sides on N-Gon

            vertex vertices[nGonSides]; //initialize array of vertices

            for(i=0;i<nGonSides;i++)
               {
                   printf("v%d.x=",i);
                   (vertices[i]).x=userInput("");    //assign x,y pts to vertices
                   printf("v%d.y=",i);
                   (vertices[i]).y=userInput("");
               }

            color=15;    //color of N-gon

            drawNGon( vertices, color, nGonSides, screen);  //draw N-Gon

            //printScreen(screen);    //draw screen
        }
        else if(userChoice==3)  //draw line
        {
            v1.x=userInput("v1.x=");
            v1.y=userInput("v1.y=");
            v2.x=userInput("v2.x=");
            v2.y=userInput("v2.y=");

               color=15;    //color of line

            drawLine(v1,v2,color,screen);   //draw line

```

```
            //printScreen(screen);    //draw screen
        }

        else if(userChoice==4)  //draw point
        {
            v1.x=userInput("v1.x=");
            v1.y=userInput("v1.y=");

            color=15;    //color of point

            drawPoint(v1,color,screen); //draw point

            //printScreen(screen);    //draw screen
        }

          else if(userChoice==5)  //benchmark demo!
        {
                while(1)
                {
                    for(y=10;y<400;y++)
                    {
                        for(x=10;x<560;x++)
                        {
                            v1.x = x;
                            v1.y = y;
                            v2.x = x+50;
                            v2.y = y+50;

                            color = x%15;

                            if((v2.y-v1.y)>0)
                            {
                                v3.x=v2.x;
                                v3.y=v1.y;
                            }
                            else
                            {
                                v3.x=v1.x;
                                v3.y=v2.y;
                            }

                            triangleFill( v1, v2, v3, color, screen);

                        }
                    }


                    //s=0;
                    while(s!=13)
                    {
                        s=XUartLite_RecvByte(XPAR_RS232_UART_1_BASEADDR);
                    }
                    s=0;
                }
        }

        else if(userChoice==9)  //quit
        {
            printf("Goodbye!\r\n");
                return 0;
        }
          else
          {
              printf("INPUT ERROR");
          }
    }
}
int userInput(int msg)
{
    char inputArray[5]={0,0,0,0,0};
     int userChoice=0;
     int i=0;
          printf("%s",msg);
          while( inputArray[i] != 13 )
```

```
            {
                i=i+1;
                  inputArray[i] = XUartLite_RecvByte(XPAR_RS232_UART_1_BASEADDR);
                  if(inputArray[i]!=13)
                  {
                      printf("%d",ascii2dec(inputArray[i]));
                  }
            }
            if(i==2)
            {
                userChoice=ascii2dec(inputArray[1]);
            }
            else if(i==3)
            {
                userChoice=10*ascii2dec(inputArray[1])+ascii2dec(inputArray[2]);
            }
            else
            {
               userChoice=100*ascii2dec(inputArray[1])+10*ascii2dec(inputArray[2])+ascii2dec(inp
            }
          printf("\r\n");
            return userChoice;
}
int ascii2dec(int ascii)
{
     int dec;
    //LUT
     switch (ascii)
     {
        case 48: dec=0; break;
        case 49: dec=1; break;
        case 50: dec=2; break;
        case 51: dec=3; break;
        case 52: dec=4; break;
        case 53: dec=5; break;
        case 54: dec=6; break;
        case 55: dec=7; break;
        case 56: dec=8; break;
        case 57: dec=9; break;
        default: dec=0; break;
     }
     return dec;
}
/* function max3:   this function returns the maximum value of eihter
                    x1, x2, or x3
*/
int max3(int x1, int x2, int x3)
{
    if(x1>x2)
    {
        if(x1>x3)
        {
            return x1;
        }
    }
    if(x2>x3)
    {
        return x2;
    }
    else
    {
        return x3;
    }
}
/* function min3:   this function returns the minimum value of eihter
                    x1, x2, or x3
*/
int min3(int x1, int x2, int x3)
{
    if(x1<x2)
    {
        if(x1<x3)
        {
            return x1;
        }
```

```
301     }
302     if(x2<x3)
303     {
304         return x2;
305     }
306     else
307     {
308         return x3;
309     }
310 }
311 /* function clearVirtScreen:    this function takes the active screen as input
312                                 and returns a blank screen (filled with 0)
313 */
314 int clearVirtScreen(char (screen[20])[50])
315 {
316     char i,j;
317 
318     for(i=0;i<20;i++)
319         {
320             for(j=0;j<50;j++)
321             {
322                 screen[i][j]=0;
323             }
324         }
325 }
326 /* function printScreen:    this function prints the contents of the screen
327                             buffer to the console
328 */
329 int printScreen(char (screen[40])[50])
330 {
331     char i,j;
332     int fbAddress=0;
333 
334     printf("\r\n");
335 
336     for(i=0;i<40;i++)
337             {
338                 for(j=0;j<50;j++)
339                 {
340                     fbAddress=j+640*i;
341                     //write peripheral((fbAddress<<13)|(screen[i][j]));
342                     VGADRIVER_mWriteSlaveReg0(XPAR_VGADRIVER_0_BASEADDR, (fbAddress<< 13)|(scr
343                     printf("%d",screen[i][j]);
344                 }
345                 printf("\r\n");
346             }
347             printf("\r\n");
348 }
349 /* function printMenu:  this function prints the menu of options to the
350                         console, this function serves little purpose without
351                         the ability for scanf()
352 */
353 int printMenu()
354 {
355     printf("\r\nTriangle Drawing Program!\r\n");
356     printf("What would you like to do?\r\n");
357     printf("    (0) Draw Triangle from vertices\r\n");
358     printf("    (1) Draw Quad from vertices\r\n");
359     printf("    (2) Draw N-gon from vertices\r\n");
360     printf("    (3) Draw line from vertices\r\n");
361     printf("    (4) Draw point from vertex\r\n");
362     printf("    (9) Quit\r\n");
363     return 0;
364 }
365 /* function drawQuad:   this function uses the triangleFill algorithm to
366                         draw a quad.  two triangles are combined to make
367                         a quad. the input vertices v1,v2,v3,v4 must be
368                         defined in a counterclockwise order
369 */
370 int drawQuad(vertex v1, vertex v2, vertex v3, vertex v4,  int color, char (*screen)[50])
371 {
372     int i,j;
373 
374     triangleFill( v1, v2, v3, color,screen);     //draw triangle 1
375     triangleFill( v1, v3, v4, color,screen);     //draw triangle 2
```

```
}
/* function drawNGon:   this function uses the triangleFill algorithm to
                        draw a convex N-Gon.  N-2 triangles are combined to make
                        a N-Gon. the input verts[] is an array of vertices
                        in counterclockwise order
*/
int drawNGon(vertex verts[], int color, int nGonSides, char (*screen)[50])
{
    int i,j;

    /*example: for hexagon, triangles are {v0,v1,v2},{v0,v2,v3},{v0,v3,v4},
    {v0,v4,v5}
                            v0______v5
                             /|\_    \
                            / |\ \_   \
                        v1/   | \  \__\v4
                          \   |  \     /
                           \  |   \   /
                            \|____\/
                            v2     v3

    This generalizes to N-Gons in the same fashion
    */

    for(i=0;i<nGonSides-2;i++)  //N-2 triangles
    {
        triangleFill( verts[0], verts[i+1], verts[i+2], color,screen);
    }
}
/* function drawLine:   this function uses Bresenham's Line algorithm
                        to draw a line between 2 vertices
*/
int drawLine(vertex v1, vertex v2, int color, char (*screen)[50])
{
    int xstart,ystart,xtemp,ytemp,yend,xend,ystep,dx,dy,steep,fbAddress;

    xstart=v1.x;
    ystart=v1.y;

    xend=v2.x;
    yend=v2.y;

    dx=xend-xstart;
    dy=yend-ystart;

    steep=abs(dy)>abs(dx);

    if(steep)   //slope > 1
    {
        //swap variables
        xtemp=v1.y;
        ytemp=v1.x;
        ystart=ytemp;
        xstart=xtemp;

        xtemp=v2.y;
        ytemp=v2.x;
        yend=ytemp;
        xend=xtemp;
    }

    if(xstart>xend)
    {
        xtemp=xstart;
        xstart=xend;
        xend=xtemp;

        ytemp=ystart;
        ystart=yend;
        yend=ytemp;
    }

    //recompute dx and dy
    dx=xend-xstart;
    dy=abs(yend-ystart);
```

```
451
452     int x=xstart;
453     int y=ystart;
454
455     if(steep)   //swap variables if steep
456     {
457         screen[x][y]=color;
458     }
459     else
460     {
461         screen[y][x]=color;
462     }
463
464     int error=dx>>1;
465
466     if(ystart<yend)
467     {
468         ystep=1;
469     }
470     else
471     {
472         ystep=-1;
473     }
474
475     while(x<xend)
476     {
477         //step in fast direction
478         x=x+1;
479         error=error-dy;
480
481         if(error<0)
482         {
483             //step in slow direction
484             y=y+ystep;
485             error=error+dx;
486         }
487         if(steep)
488         {
489             //screen[x][y]=color;
490             fbAddress=x*640+y;
491             //write peripheral((fbAddress<<13)|(screen[i][j]));
492             VGADRIVER_mWriteSlaveReg0(XPAR_VGADRIVER_0_BASEADDR, (fbAddress<< 13)|color);
493         }
494         else
495         {
496             //screen[y][x]=color;
497             fbAddress=y*640+x;
498             //write peripheral((fbAddress<<13)|(screen[i][j]));
499             VGADRIVER_mWriteSlaveReg0(XPAR_VGADRIVER_0_BASEADDR, (fbAddress<< 13)|color);
500          }
501     }
502
503 }
504 /* function drawPoint:  this function simply draws a point on the screen
505 */
506 int drawPoint(vertex v1, int color, char (*screen)[50])
507 {
508     int fbAddress;
509     //screen[v1.y][v1.x]=color;
510     fbAddress=v1.y*640+v1.x;
511     //write peripheral((fbAddress<<13)|(screen[i][j]));
512     VGADRIVER_mWriteSlaveReg0(XPAR_VGADRIVER_0_BASEADDR, (fbAddress<< 13)|color);
513 }
514 /* function triangleFill:   this function draws a triangle given 3 vertices.
515                             it works by computing the half space functions for
516                             the three lines defined by {v1,v2},{v2,v3},{v3,v1}.
517                             if all the half space functions are positive, the
518                             point is inside the triangle.  this algorithm is
519                             sped up by processing the screen in 8x8 pixel blocks.
520                             we can check the corners of the block to determine if
521                             the block is empty, full, or partially covered.  this
522                             speeds up processing and leaves opportunities for
523                             hardware acceleration and parallelism. this algorithm
524                             was adapted from an online tutorial by Nicolas Capens,
525                             http://www.devmaster.net/forums/showthread.php?t=1884
```

```
*/
int triangleFill(vertex v1, vertex v2, vertex v3,  int color, char (screen[40])[50])
{
    //initialize variables
    int ix,iy,x,y,i,j;                  //loop indices
    int Cx1,Cx2,Cx3,Cy1,Cy2,Cy3;        //pixel constants
    int fbAddress;                      //frame buffer address for hardware
    char ACTIVATED;                     //status variable for right hand checking
    int A,Ax,Ay,Axacc,A00,A10,A01,A11;  //half space constants
    int B,Bx,By,Bxacc,B00,B10,B01,B11;
    int C,Cx,Cy,Cxacc,C00,C10,C01,C11;
    int C1,C2,C3;
    int MINY7,MINX7;
    int DX12MINY,DY12MINX,DX23MINY,DY23MINX,DX31MINY,DY31MINX;
    int DX12MINY7,DY12MINX7,DX23MINY7,DY23MINX7,DX31MINY7,DY31MINX7;

    //shift the vertex levels over to create a 28.4 fixed point representation
    const int Y1=v1.y<<4;
    const int Y2=v2.y<<4;
    const int Y3=v3.y<<4;

    const int X1=v1.x<<4;
    const int X2=v2.x<<4;
    const int X3=v3.x<<4;

    //compute deltas for defining lines
    const int DX12=X1-X2;
    const int DX23=X2-X3;
    const int DX31=X3-X1;

    const int DY12=Y1-Y2;
    const int DY23=Y2-Y3;
    const int DY31=Y3-Y1;

    //fixed point representation for deltas
    const int FDX12=DX12<<4;
    const int FDX23=DX23<<4;
    const int FDX31=DX31<<4;

    const int FDY12=DY12<<4;
    const int FDY23=DY23<<4;
    const int FDY31=DY31<<4;

    //determine bounding rectangle of triangle
    int MINX=(min3(X1, X2, X3)+0xF)>>4;
    int MAXX=(max3(X1, X2, X3)+0xF)>>4;
    int MINY=(min3(Y1, Y2, Y3)+0xF)>>4;
    int MAXY=(max3(Y1, Y2, Y3)+0xF)>>4;

    //define the block size for fast block processing
    int q=8;

    //start in the corner of a q x q block
    MINX=MINX&(~(q-1));
    MINY=MINY&(~(q-1));

    //half-edge equation constants
    C1=DY12*X1-DX12*Y1;
    C2=DY23*X2-DX23*Y2;
    C3=DY31*X3-DX31*Y3;

    //filling convention
    //the top left edge belongs to the triangle
    //this prevents double drawing and gaps of pixels with triangles
    //that share edges and vertices
    if(DY12<0 || (DY12==0 && DX12>0))
    {
        C1=C1+1;
    }
    if(DY23<0 || (DY23==0 && DX23>0))
    {
        C2=C2+1;
    }
    if(DY31<0 || (DY31==0 && DX31>0))
    {
```

```
        C3=C3+1;
    }

    //precompute this for savings in hardware implementation
    MINY7=((MINY+7)<<4);
    MINX7=((MINX+7)<<4);

    //precompute these to go from 24 to 12 multiplies
    DX12MINY=DX12*(MINY<<4);
    DY12MINX=DY12*(MINX<<4);
    DX12MINY7=DX12*MINY7;
    DY12MINX7=DY12*MINX7;

    DX23MINY=DX23*(MINY<<4);
    DY23MINX=DY23*(MINX<<4);
    DX23MINY7=DX23*MINY7;
    DY23MINX7=DY23*MINX7;

    DX31MINY=DX31*(MINY<<4);
    DY31MINX=DY31*(MINX<<4);
    DX31MINY7=DX31*MINY7;
    DY31MINX7=DY31*MINX7;

    //computes constant part of half edge equations
    A00=C1+DX12MINY-DY12MINX;
    A10=C1+DX12MINY-DY12MINX7;
    A01=C1+DX12MINY7-DY12MINX;
    A11=C1+DX12MINY7-DY12MINX7;

    B00=C2+DX23MINY-DY23MINX;
    B10=C2+DX23MINY-DY23MINX7;
    B01=C2+DX23MINY7-DY23MINX;
    B11=C2+DX23MINY7-DY23MINX7;

    C00=C3+DX31MINY-DY31MINX;
    C10=C3+DX31MINY-DY31MINX7;
    C01=C3+DX31MINY7-DY31MINX;
    C11=C3+DX31MINY7-DY31MINX7;

    //precomputes increment for half edge equations
    Ay=(DX12<<7);
    By=(DX23<<7);
    Cy=(DX31<<7);
    Ax=(DY12<<7);
    Bx=(DY23<<7);
    Cx=(DY31<<7);

    //loop through filling blocks
    for(y=MINY; y<=MAXY; y=y+q)
    {
        ACTIVATED=0;    //reset activated at the beginning of a row
        Axacc=0;        //reset accumulators
        Bxacc=0;
        Cxacc=0;

        for(x=MINX; x<=MAXX; x=x+q)
        {
            //check the blocks corners
            //based on the corner values, we can decide
            //whether the block is completely filled,
            //not filled, or partiall filled

            //evaluate the half space function
            //determines if the pixel is on the inside
            //of the line (inside figure), or outside
            //of the line (outside figure)

            A=((A00>0)|((A10>0)<<1)|((A01>0)<<2)|((A11>0)<<3));
            B=((B00>0)|((B10>0)<<1)|((B01>0)<<2)|((B11>0)<<3));
            C=((C00>0)|((C10>0)<<1)|((C01>0)<<2)|((C11>0)<<3));

            Axacc=Axacc+Ax;
            Bxacc=Bxacc+Bx;
            Cxacc=Cxacc+Cx;

```

```
676             A00=A00-Ax;
677             A10=A10-Ax;
678             A01=A01-Ax;
679             A11=A11-Ax;
680
681             B00=B00-Bx;
682             B10=B10-Bx;
683             B01=B01-Bx;
684             B11=B11-Bx;
685
686             C00=C00-Cx;
687             C10=C10-Cx;
688             C01=C01-Cx;
689             C11=C11-Cx;
690
691             if( (A==0 || B==0 || C==0) && ACTIVATED==1)
692             {
693                 break;  //when empty blocks are encountered to the right
694                         //of a block that had pixels drawn, advance to the
695                         //next row
696             }
697             else    //when were not skipping right hand blocks
698             {
699                 //if all the corners have outside edges, we can skip it
700                 if(A==0 || B==0 || C==0 ){}
701
702                 //accept whole block when all corners are inside triangle
703                 else if(A==15 && B==15 && C==15)
704                 {
705                     //printf("In whole block\r\n");
706                     ACTIVATED=1;
707                     for(iy=y; iy<y+q; iy++)
708                     {
709                         for(ix=x; ix<=x+q; ix++)
710                         {
711                             //screen[iy][ix]=2;
712
713                             fbAddress=iy*640+ix;
714                             //write peripheral((fbAddress<<13)|(screen[i][j]));
715                             VGADRIVER_mWriteSlaveReg0(XPAR_VGADRIVER_0_BASEADDR, (fbAddress<<
716                         }
717                     }
718                 }
719                 //partially covered blocks
720                 else
721                 {
722                     ACTIVATED=1;
723                     //printf("In partially loop\r\n");
724                     Cy1=A00+Ax;
725                     Cy2=B00+Bx;
726                     Cy3=C00+Cx;
727                     for(iy=y; iy<=y+q; iy++)
728
729                     {
730                         Cx1=Cy1;
731                         Cx2=Cy2;
732                         Cx3=Cy3;
733
734                         for(ix=x; ix<=x+q; ix++)
735                         {
736                             if(Cx1>0 && Cx2>0 && Cx3>0)
737                             {
738                                 //screen[iy][ix]=color;
739
740                                 fbAddress=iy*640+ix;
741                                 //write peripheral((fbAddress<<13)|(screen[i][j]));
742                                 VGADRIVER_mWriteSlaveReg0(XPAR_VGADRIVER_0_BASEADDR, (fbAddres
743                             }
744
745                             Cx1=Cx1-FDY12;
746                             Cx2=Cx2-FDY23;
747                             Cx3=Cx3-FDY31;
748                         }
749
750                         Cy1=Cy1+FDX12;
```

```
                        Cy2=Cy2+FDX23;
                        Cy3=Cy3+FDX31;
                    }
                }
            }
        }
        //update half edge equations, flush accumulator and increment the y
        A00=A00+Ay+Axacc;
        A10=A10+Ay+Axacc;
        A01=A01+Ay+Axacc;
        A11=A11+Ay+Axacc;

        B00=B00+By+Bxacc;
        B10=B10+By+Bxacc;
        B01=B01+By+Bxacc;
        B11=B11+By+Bxacc;

        C00=C00+Cy+Cxacc;
        C10=C10+Cy+Cxacc;
        C01=C01+Cy+Cxacc;
        C11=C11+Cy+Cxacc;
    }

}
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    00:40:18 04/23/2007
// Design Name:
// Module Name:    toplevelrun
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module toplevelrun(
CLOCK,
VGA_OUT_PIXEL_CLOCK,
VGA_COMP_SYNCH_N,
VGA_OUT_BLANK_N,
VGA_HSYNCH,
VGA_VSYNCH,
VGA_OUT_RED,
VGA_OUT_GREEN,
VGA_OUT_BLUE);

input          CLOCK;               // 100MHz system clock
output         VGA_OUT_PIXEL_CLOCK; // pixel clock for the video DAC
output         VGA_COMP_SYNCH_N;    // composite sync for the video DAC
output         VGA_OUT_BLANK_N;     // composite blanking for the video DAC
output         VGA_HSYNCH;          // horizontal sync for the VGA output connector
output         VGA_VSYNCH;          // vertical sync for the VGA output connector
output [7:0]   VGA_OUT_RED;         // RED DAC data
output [7:0]   VGA_OUT_GREEN;       // GREEN DAC data
output [7:0]   VGA_OUT_BLUE;        // BLUE DAC data

wire reset;
wire writePixel;
wire [18:0] fbAddress;

toplevel triGPU(CLOCK,reset,8,14'd2,14'd2,14'd16,14'd2,14'd200,14'd2, writePixel, fbAddr
);


VGAFB_CTRL theFB(CLOCK,
VGA_OUT_PIXEL_CLOCK,
VGA_COMP_SYNCH_N,
VGA_OUT_BLANK_N,
VGA_HSYNCH,
VGA_VSYNCH,
VGA_OUT_RED,
VGA_OUT_GREEN,
VGA_OUT_BLUE,
4'b1010,
fbAddress,
writePixel,reset);

endmodule
```

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3   // Company: uPs
4   // Engineer: Philip Amberg and Andrew Giles
5   //
6   // Create Date:    15:59:48 04/16/2007
7   // Design Name:
8   // Module Name:    toplevel
9   // Project Name: Virtex2D Graphics Accelerator
10  //
11  //////////////////////////////////////////////////////////////////////////////////
12  module toplevel(clk,reset,q,x1,x2,x3,y1,y2,y3, writePixel, fbAddress);
13
14     input clk;
15     input reset;
16     input [13:0] q;
17     input [9:0] x1,x2,x3;
18     input [9:0] y1,y2,y3;
19     output [18:0] fbAddress;
20     output writePixel;
21
22     wire [13:0] x1s,x2s,x3s,y1s,y2s,y3s;
23     wire [13:0] dx12,dx23,dx31,dy12,dy23,dy31;
24     wire [17:0] fdx12,fdx23,fdx31,fdy12,fdy23,fdy31;
25     wire [13:0] maxx,maxy;
26     wire [13:0] minxAligned,minyAligned;
27     wire [27:0] c1,c2,c3;
28     wire [27:0] c1out,c2out,c3out;
29     wire [13:0] minx, miny;
30     wire [17:0] miny7, minx7, minxShift, minyShift;
31     wire [31:0] dx12miny,dy12minx,dx12miny7,dy12minx7,dx23miny,dy23minx,dx23miny7,dy23minx
       dx31miny,dy31minx,dx31miny7,dy31minx7;
32     wire [20:0] Ay,By,Cy,Ax,Bx,Cx;
33     wire [31:0] a00,a01,a10,a11,b00,b01,b10,b11,c00,c01,c10,c11;
34
35     computeConstants generateConstants(x1,x2,x3,y1,y2,y3,x1s,x2s,x3s,y1s,y2s,y3s,dx12,dx23
       dx31,dy12,dy23,dy31,fdx12,fdx23,fdx31,fdy12,fdy23,fdy31);
36     minmax3 boundingBox(x1s,x2s,x3s,y1s,y2s,y3s,minx,miny,maxx,maxy);
37     blockAlign startAtPow2(minx,miny,q,minxAligned,minyAligned);
38     initHalfEdge halfEdgeConstants(x1s,x2s,x3s,y1s,y2s,y3s,dx12,dx23,dx31,dy12,dy23,dy31,
       c2,c3);
39     fillConvention adjustConstants(dy12,dx12,dy23,dx23,dy31,dx31,c1,c2,c3,c1out,c2out,c3o
40     qShiftPlus7 qShift(minyAligned,minxAligned,miny7,minx7, minxShift, minyShift);
41     preMults preCompute(dx12,dx23,dx31,dy12,dy23,dy31,minxShift,minyShift,minx7,miny7,
       dx12miny,dy12minx,dx12miny7,dy12minx7,dx23miny,dy23minx,dx23miny7,dy23minx7,dx31miny,
       dy31minx,dx31miny7,dy31minx7);
42     halfEdgeIncrement halfEdgeinc(dx12,dx23,dx31,dy12,dy23,dy31,Ay,By,Cy,Ax,Bx,Cx);
43
44
45     constHalfEdge theConst(c1out,dx12miny,dy12minx,dx12miny7,dy12minx7,a00,a10,a01,a11,c2
       ,dx23miny,dy23minx,dx23miny7,dy23minx7,b00,b10,b01,b11,c3out,dx31miny,dy31minx,dx31miny7
       dy31minx7,c00,c10,c01,c11);
46
47     loopToplevel loopz(clk,reset,minxAligned,maxx,minyAligned,maxy,fdx12,fdx23,fdx31,fdy1
       fdy23,fdy31,a00,a01,a10,a11,b00,b01,b10,b11,c00,c01,c10,c11,Ax,Ay,Bx,By,Cx,Cy,writePixel
       fbAddress);
48
49
50  endmodule
51
52
53  module constHalfEdge(c1,dx12miny,dy12minx,dx12miny7,dy12minx7,a00,a10,a01,a11,c2,dx23miny
       dy23minx,dx23miny7,dy23minx7,b00,b10,b01,b11,c3,dx31miny,dy31minx,dx31miny7,dy31minx7,c0
       c10,c01,c11);
54
```

```
55          input [27:0] c1,c2,c3;
56          input [31:0] dx12miny,dy12minx,dx23miny,dy23minx,dx31miny,dy31minx;
57          input [31:0] dx12miny7,dy12minx7,dx23miny7,dy23minx7,dx31miny7,dy31minx7;
58          output [31:0] a00,a10,a01,a11,b00,b10,b01,b11,c00,c10,c01,c11;
59
60
61          ////////////////////(c1,dx12miny,dy12minx,dx12miny7,dy12minx7,a00,a10,a01,a11)
62          constHalfEdgeUno aa(c1,dx12miny,dy12minx,dx12miny7,dy12minx7,a00,a10,a01,a11);
63          constHalfEdgeUno bb(c2,dx23miny,dy23minx,dx23miny7,dy23minx7,b00,b10,b01,b11);
64          constHalfEdgeUno cc(c3,dx31miny,dy31minx,dx31miny7,dy31minx7,c00,c10,c01,c11);
65
66      endmodule
67
68
```

```
1         `timescale 1ns / 1ps
2         //////////////////////////////////////////////////////////////////////////////////
3         // Company: uPs
4         // Engineer: Philip Amberg and Andrew Giles
5         //
6         // Create Date:    16:03:56 04/16/2007
7         // Design Name:
8         // Module Name:    modules
9         // Project Name: Virtex2D Graphics Accelerator
10        //
11        //////////////////////////////////////////////////////////////////////////////////
12        module computeConstants(x1,x2,x3,y1,y2,y3,x1s,x2s,x3s,y1s,y2s,y3s,dx12,dx23,dx31,dy12,dy
          dy31,fdx12,fdx23,fdx31,fdy12,fdy23,fdy31);
13
14           input [9:0] x1,x2,x3;   //input vertices
15           input [9:0] y1,y2,y3;
16           output [13:0] x1s,x2s,x3s,y1s,y2s,y3s;
17           output [13:0] dx12,dx23,dx31,dy12,dy23,dy31;
18           output [17:0] fdx12,fdx23,fdx31,fdy12,fdy23,fdy31; //fixed point representations
19
20           assign x1s=x1<<4;
21           assign x2s=x2<<4;
22           assign x3s=x3<<4;
23           assign y1s=y1<<4;
24           assign y2s=y2<<4;
25           assign y3s=y3<<4;
26
27           assign dx12=(x1s-x2s);
28           assign dx23=(x2s-x3s);
29           assign dx31=(x3s-x1s);
30           assign dy12=(y1s-y2s);
31           assign dy23=(y2s-y3s);
32           assign dy31=(y3s-y1s);
33
34           assign fdx12=dx12<<4;
35           assign fdx23=dx23<<4;
36           assign fdx31=dx31<<4;
37           assign fdy12=dy12<<4;
38           assign fdy23=dy23<<4;
39           assign fdy31=dy31<<4;
40
41        endmodule
42
43        module minmax3(x1s,x2s,x3s,y1s,y2s,y3s,xmin,ymin,xmax,ymax);
44
45           input [13:0] x1s,x2s,x3s,y1s,y2s,y3s;
46           output [13:0] xmin,ymin,xmax,ymax;
47
48           wire [2:0] sxmin,symin;
49           wire [13:0] xmintemp,xmaxtemp,ymintemp,ymaxtemp;
50
51           minmax2 xset1(x1s,x2s,minx1);
52           minmax2 xset2(x1s,x3s,minx2);
53           minmax2 xset3(x2s,x3s,minx3);
54
55           minmax2 yset1(y1s,y2s,ymin1);
56           minmax2 yset2(y1s,y3s,ymin2);
57           minmax2 yset3(y2s,y3s,ymin3);
58
59           mux8 xminselect(x3s,x2s,14'b0,x2s,x3s,14'b0,x1s,x1s,xmintemp,sxmin);
60           mux8 xmaxselect(x3s,x2s,14'b0,x2s,x3s,14'b0,x1s,x1s,xmaxtemp,~sxmin);
61
62           mux8 yminselect(y3s,y2s,14'b0,y2s,y3s,14'b0,y1s,y1s,ymintemp,symin);
63           mux8 ymaxselect(y3s,y2s,14'b0,y2s,y3s,14'b0,y1s,y1s,ymaxtemp,~symin);
64
```

```
65          assign sxmin={minx1,minx2,minx3};
66          assign symin={ymin1,ymin2,ymin3};
67          assign xmin=(xmintemp+13'hF)>>4;
68          assign ymin=(ymintemp+13'hF)>>4;
69          assign xmax=(xmaxtemp+13'hF)>>4;
70          assign ymax=(ymaxtemp+13'hF)>>4;
71
72      endmodule
73
74      module minmax2(d1,d2,min);
75
76          input [13:0] d1,d2;
77          output min;
78
79          assign min=(d1<d2);
80
81      endmodule
82
83      module blockAlign(minx,miny,q,minxAligned,minyAligned);
84
85          input [13:0] q;
86          input [13:0] minx,miny;
87          output [13:0] minxAligned,minyAligned;
88
89          assign minxAligned=minx&(~(q-1));
90          assign minyAligned=miny&(~(q-1));
91
92      endmodule
93
94      module minmults(minxshift, minyshift, minx7shift, miny7shift, dxab, dyab, dxabminy,
        dyabminx, dxab7miny, dyab7minx);
95            input [17:0] minxshift, minx7shift, minyshift, miny7shift;
96            input [13:0] dxab, dyab;
97            output [31:0] dxabminy, dyabminx, dxab7miny, dyab7minx;
98
99            mult1317 dxabminymult(minyshift, dxab, dxabminy);
100           mult1317 dyabminxmult(minxshift, dyab, dyabminx);
101           mult1317 dxab7minymult(miny7shift, dxab, dxab7miny);
102           mult1317 dyab7minxmult(minx7shift, dyab, dyab7minx);
103
104     endmodule
105
106     module qShiftPlus7(miny,minx,miny7,minx7, minyShift, minxShift);
107
108         input [13:0] miny,minx;
109         output [17:0] miny7,minx7,minyShift,minxShift;
110
111         assign miny7=(miny+7)<<4;
112         assign minx7=(minx+7)<<4;
113         assign minyShift=miny<<4;
114         assign minxShift=minx<<4;
115
116     endmodule
117
118     module preMults(dx12,dx23,dx31,dy12,dy23,dy31,minxShift,minyShift,minx7,miny7,dx12minx,
        dy12minx,dx12miny7,dy12miny7, dx23minx, dy23minx, dx23miny7, dy23miny7, dx31minx, dy31mi
         dx31miny7, dy31miny7);
119
120         input [13:0] dx12,dx23,dx31,dy12,dy23,dy31;
121         input [17:0] minxShift,minyShift,minx7,miny7;
122         output [31:0] dx12minx,dy12minx,dx12miny7,dy12miny7,dx23minx,dy23minx,dx23miny7,
        dy23miny7,dx31minx,dy31minx,dx31miny7,dy31miny7;
123
124         minmults a12(minxShift, minyShift, minx7, miny7, dx12, dy12, dx12minx, dy12minx,
        dx12miny7, dy12miny7);
```

```
125          minmults a23(minxShift, minyShift, minx7, miny7, dx23, dy23, dx23minx, dy23minx,
         dx23miny7, dy23miny7);
126          minmults a31(minxShift, minyShift, minx7, miny7, dx31, dy31, dx31minx, dy31minx,
         dx31miny7, dy31miny7);
127
128       endmodule
129
130       module initHalfEdge(x1s,x2s,x3s,y1s,y2s,y3s,dx12,dx23,dx31,dy12,dy23,dy31,c1,c2,c3);
131
132          input [13:0] x1s,x2s,x3s,y1s,y2s,y3s;
133          input [13:0] dx12,dx23,dx31,dy12,dy23,dy31;
134          output [27:0] c1,c2,c3;
135
136          wire [27:0] cx1,cy1,cx2,cy2,cx3,cy3;
137
138          mult1414s mulcx1(dy12,x1s,cx1);
139          mult1414s mulcy1(dx12,y1s,cy1);
140
141          mult1414s mulcx2(dy23,x2s,cx2);
142          mult1414s mulcy2(dx23,y2s,cy2);
143
144          mult1414s mulcx3(dy31,x3s,cx3);
145          mult1414s mulcy3(dx31,y3s,cy3);
146
147          assign c1=cx1-cy1;
148          assign c2=cx2-cy2;
149          assign c3=cx3-cy3;
150
151       endmodule
152
153       module constHalfEdgeUno(c1,dx12miny,dy12minx,dx12miny7,dy12minx7,a00,a10,a01,a11);
154
155          input [27:0] c1;
156          input [31:0] dx12miny,dy12minx;
157          input [31:0] dx12miny7,dy12minx7;
158          output [31:0] a00,a10,a01,a11;
159
160          assign a00={c1[27],c1[27],c1[27],c1[27], c1} + dx12miny - dy12minx;
161          assign a10={c1[27],c1[27],c1[27],c1[27], c1}+dx12miny-dy12minx7;
162          assign a01={c1[27],c1[27],c1[27],c1[27], c1}+dx12miny7-dy12minx;
163          assign a11={c1[27],c1[27],c1[27],c1[27], c1}+dx12miny7-dy12minx7;
164
165       endmodule
166
167
168
169       module halfEdgeIncrementUno(dx12,Ay);
170
171          input [13:0] dx12;
172          output [20:0] Ay;
173
174          assign Ay=dx12<<7;
175
176       endmodule
177
178       module halfEdgeIncrement(dx12,dx23,dx31,dy12,dy23,dy31,Ay,By,Cy,Ax,Bx,Cx);
179
180          input [13:0] dx12,dx23,dx31,dy12,dy23,dy31;
181          output [20:0] Ay,By,Cy,Ax,Bx,Cx;
182
183          halfEdgeIncrementUno ay(dx12,Ay);
184          halfEdgeIncrementUno by(dx23,By);
185          halfEdgeIncrementUno cy(dx31,Cy);
186          halfEdgeIncrementUno ax(dy12,Ax);
187          halfEdgeIncrementUno bx(dy23,Bx);
```

```
188            halfEdgeIncrementUno cx(dy31,Cx);
189
190         endmodule
191
192         module fillConvention(dy12,dx12,dy23,dx23,dy31,dx31,c1,c2,c3,c1out,c2out,c3out);
193
194            input [13:0] dy12,dx12,dy23,dx23,dy31,dx31;
195            input [27:0] c1,c2,c3;
196            output [27:0] c1out,c2out,c3out;
197
198            wire comp1;
199            wire comp2;
200
201            assign comp1 = (dy12[13]);
202            assign comp2 = (dy12==0) & (~dx12[13] & (|dx12));
203
204            assign c1out=c1+(comp1 | comp2);
205            assign c2out=c2+((dy23[13]) | ((dy23==0) & (~dx23[13] & (|dx23))));
206            assign c3out=c3+((dy31[13]) | ((dy31==0) & (~dx31[13] &  (|dx31))));
207
208         endmodule
209
210         module blockLoop();
211
212            //actual sequential logic goes in here
213
214         endmodule
215
216         module mux8(d0,d1,d2,d3,d4,d5,d6,d7,y,s);
217
218            input [13:0] d0,d1,d2,d3,d4,d5,d6,d7;
219            input [2:0] s;
220            output [13:0] y;
221
222            wire [13:0] low,high;
223
224            mux4 lowmux(d0,d1,d2,d3,s[1:0],low);
225            mux4 highmux(d4,d5,d6,d7,s[1:0],high);
226            mux2 finalmux(low,high,s[2],y);
227
228         endmodule
229
230         module mux4(d0,d1,d2,d3,s,y);
231
232            input [13:0] d0,d1,d2,d3;
233            input [1:0] s;
234            output [13:0] y;
235
236            wire [13:0] low,high;
237
238            mux2 lowmux(d0,d1,s[0],low);
239            mux2 highmux(d2,d3,s[0],high);
240            mux2 finalmux(low,high,s[1],y);
241
242         endmodule
243
244         module mux2(d0,d1,s,y);
245
246            input [13:0] d0,d1;
247            input s;
248            output [13:0] y;
249
250            assign y=s?d1:d0;
251
252         endmodule
```

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3   // Company:
4   // Engineer: Philip Amberg and Andrew Giles
5   //
6   // Create Date:    11:07:49 04/22/2007
7   // Design Name:
8   // Module Name:    loopToplevel
9   // Project Name: Virtex2D Graphics Accelerator
10  //
11  //////////////////////////////////////////////////////////////////////////////////
12  module loopToplevel(clk,reset,minx,maxx,miny,maxy,fdx12,fdx23,fdx31,fdy12,fdy23,fdy31,A0
    A01,A10,A11,B00,B01,B10,B11,C00,C01,C10,C11,Ax,Ay,Bx,By,Cx,Cy,writePixel,fbAddress);
13
14     input clk,reset;
15     input [13:0] minx,maxx,miny,maxy;
16     input [17:0] fdx12,fdx23,fdx31,fdy12,fdy23,fdy31;
17     input [31:0] A00,A01,A10,A11,B00,B01,B10,B11,C00,C01,C10,C11;
18     input [20:0] Ax,Ay,Bx,By,Cx,Cy;
19     output writePixel;
20     output [18:0]fbAddress;
21
22     wire [13:0] x,y,ix,iy;
23     wire [1:0] state;
24
25     wire [3:0] A,B,C;
26
27     wire [31:0] A00pass,B00pass,C00pass;
28     wire EMPTY;
29     wire [31:0] Cx1,Cx2,Cx3,Cy1,Cy2,Cy3;
30
31     looptop outsideLoop(clk,reset,x,y,minx,maxx,miny,maxy,iy,ix,state);
32     comblooptop CLBLK(clk,reset,x,ix, y,iy, maxx,maxy,A,B,C,Ax,Ay,A00,A01,A10,A11,Bx,By,B
    B01,B10,B11,Cx,Cy,C00,C01,C10,C11,A00pass,B00pass,C00pass);
33
34     drawLoop pixelLoop(clk,reset,x,y,ix,iy,state);
35
36     C123 updateCyCx(clk,reset,A00pass,B00pass,C00pass,Cx1,Cx2,Cx3,Cy1,Cy2,Cy3,fdx12,fdx23
    fdx31,fdy12,fdy23,fdy31,x,y,ix,iy);
37     pixelOn drawPixel(Cx1,Cx2,Cx3,iy,ix,writePixel,fbAddress);
38
39
40
41  endmodule
42
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: uPs
// Engineer: Philip Amberg and Andrew Giles
//
// Create Date:    22:59:34 04/21/2007
// Design Name:
// Module Name:    looptop
// Project Name: Virtex2D Graphics Accelerator
//
//////////////////////////////////////////////////////////////////////////////////
module looptop(clk,reset,x,y,minx,maxx,miny,maxy,iy,ix,state);

   input clk;
   input reset;
   output reg [13:0] x,y;
   input [13:0] minx,maxx,miny,maxy;
   input [13:0] iy,ix;
   output [1:0] state;


   //parameterize state encodings
   parameter COUNTINGSTATE=2'b00;
   parameter DONESTATE=2'b01;
   parameter SKIPSTATE=2'b10;

   reg [1:0] state;


   always@(posedge clk, posedge reset)
      begin
         if(reset)
            begin
               y <= miny;
               x <= minx;
               state <= COUNTINGSTATE;
            end
         /*else if(state==SKIPSTATE)
            begin
               if(x==(maxx-8))
                  begin
                  y <= y+8;
                  x <= minx;
                  state <= COUNTINGSTATE;
                  end
               else if (y == maxy)
                  state <= DONESTATE;
               else
                  x<=x+8;
            end*/
         else if(state==DONESTATE)
            begin
               x<=x;
               y<=y;
            end
         //else if(EMPTY)
         // state<=SKIPSTATE;
         else if(x==maxx && ((iy==maxy-1)&&(ix==x+7)))
            //we are done, find a way out
            state <= DONESTATE;
         else if(x==maxx &&((iy==y+7)&&(ix==x+7)))
            begin
               y <= y+8;
               x <= minx;
               state <= COUNTINGSTATE;
```

```
66                   end
67                else if((iy==y+7)&&(ix==x+7))
68                   x<=x+8;
69                else
70                   begin
71                      y <= y;
72                      x <= x;
73                      state <= COUNTINGSTATE;
74                   end
75             end
76
77          endmodule
78
79          module ABC //1,2,3, just you and me.
80             (x00x, x01x, x10x, x11x, X);
81
82             input [31:0] x00x, x01x, x10x,x11x;
83             output [3:0] X;
84
85
86             assign X = {(~x00x[31] & (|x00x)), (~x01x[31] & (|x01x)) , (~x10x[31] & (|x10x)), (~x
           [31] & (|x11x))};
87
88          endmodule
89
90
91          module xloop(x00initial, x01initial, x10initial, x11initial, Xx, Xy, x00, x01,x10,x11, c
            reset,x,ix,y,iy, maxx);
92
93             input [31:0] x00initial, x01initial, x10initial, x11initial;
94             input [20:0] Xx,Xy;
95             output reg [31:0] x00, x01, x10, x11;
96             //output [31:0] Xxacc;
97             input clk, reset;
98             input [13:0] x, ix, y,iy,maxx;
99
100            always@(posedge clk or posedge reset)
101            begin
102               if (reset)
103                  begin
104                     //Xxacc <= 32'b0;
105                     x00 <= x00initial;
106                     x01 <= x01initial;
107                     x10 <= x10initial;
108                     x11 <= x11initial;
109                  end
110               else if ((x == maxx)&& (ix == maxx) && (iy == (y+7)))
111                  begin
112                     //Xxacc <= 32'b0;
113                     x00 <= x00initial;// + {
           {Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]
           y[20]}, Xy[20:0]};
114                     x01 <= x01initial;// + {
           {Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]
           y[20]}, Xy[20:0]};
115                     x10 <= x10initial;// + {
           {Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]
           y[20]}, Xy[20:0]};
116                     x11 <= x11initial;// + {
           {Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]
           y[20]}, Xy[20:0]};
117                  end
118               else if ((ix == x+6)&& (iy == y+7))
119                  begin
120                     //Xxacc <= Xxacc + Xx;
```

```
121                 x00 <= x00 - {Xx[20], Xx[20],Xx[20],Xx[20],Xx[20],Xx[20],Xx[20],Xx[20],Xx[20
        Xx[20],Xx[20], Xx[20:0]};
122                 x01 <= x01 - {Xx[20], Xx[20],Xx[20],Xx[20],Xx[20],Xx[20],Xx[20],Xx[20],Xx[20
        Xx[20],Xx[20], Xx[20:0]};
123                 x10 <= x10 - {Xx[20], Xx[20],Xx[20],Xx[20],Xx[20],Xx[20],Xx[20],Xx[20],Xx[20
        Xx[20],Xx[20], Xx[20:0]};
124                 x11 <= x11 - {Xx[20], Xx[20],Xx[20],Xx[20],Xx[20],Xx[20],Xx[20],Xx[20],Xx[20
        Xx[20],Xx[20], Xx[20:0]};
125              end
126           end
127
128        endmodule
129
130
131        module yloop(clk, reset, y, iy, x,ix, maxx, maxy, x00initial, x01initial, x10initial,
        x11initial,Xy, x00init, x01init, x10init, x11init);
132
133           input clk, reset;
134           input [13:0] x,y,ix,iy;
135           input [20:0] Xy;
136           input [13:0] maxx,maxy;
137           input [31:0] x00initial, x01initial, x10initial, x11initial;
138           output reg [31:0] x00init, x01init, x10init, x11init;
139
140
141           always@ (posedge clk or posedge reset)
142           begin
143              if (reset)
144              begin
145                 x00init <= x00initial;
146                 x01init <= x01initial;
147                 x10init <= x10initial;
148                 x11init <= x11initial;
149              end
150              else if (y == maxy)
151              begin
152                 x00init <= x00initial;
153                 x01init <= x01initial;
154                 x10init <= x10initial;
155                 x11init <= x11initial;
156              end
157              else if ((x == maxx) && (ix == maxx) && (iy == (y+7)))
158              begin
159                 x00init <= x00init + { {Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy
        20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]}, Xy[20:0]};
160                 x01init <= x01init + { {Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy
        20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]}, Xy[20:0]};
161                 x10init <= x10init + { {Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy
        20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]}, Xy[20:0]};
162                 x11init <= x11init + { {Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy
        20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]}, Xy[20:0]};
163              end
164           end
165
166        endmodule
167
168
169
170        module comblooptop(clk, reset, x, ix, y, iy, maxx, maxy, A,B,C, Ax, Ay, A00, A01, A10, A1
         Bx, By,B00, B01, B10, B11, Cx, Cy, C00,C01, C10, C11, A00out, B00out, C00out);
171
172           input clk, reset;
173           input [13:0] x,y,ix,iy;
174           input [13:0] maxx, maxy;
175           output [3:0] A,B,C;
```

```
176           input [20:0] Ax, Ay, Bx, By, Cx,Cy;
177           input [31:0] A00, A01, A10, A11, B00, B01, B10, B11, C00, C01, C10, C11;
178
179           output [31:0] A00out, B00out, C00out;
180
181           wire [31:0] A00pass, A01pass, A10pass, A11pass;
182           wire [31:0] B00pass, B01pass, B10pass, B11pass;
183           wire [31:0] C00pass, C01pass, C10pass, C11pass;
184
185           wire [31:0] A00out, A01out, A10out, A11out;
186           wire [31:0] B00out, B01out, B10out, B11out;
187           wire [31:0] C00out, C01out, C10out, C11out;
188
189
190           yloop Ayloop(clk, reset, y, iy,x, ix,maxx, maxy, A00, A01, A10, A11,Ay, A00pass, A01pa
          , A10pass, A11pass);
191           xloop Axloop(A00pass, A01pass, A10pass, A11pass, Ax, Ay, A00out, A01out, A10out, A11ou
           clk, reset, x, ix,y,iy, maxx);
192
193           yloop Byloop(clk, reset, y, iy,x, ix,maxx, maxy, B00, B01, B10, B11,By, B00pass, B01pa
          , B10pass, B11pass);
194           xloop Bxloop(B00pass, B01pass, B10pass, B11pass, Bx, By,  B00out, B01out, B10out, B11
          , clk, reset, x, ix,y,iy, maxx);
195
196           yloop Cyloop(clk, reset, y, iy,x, ix,maxx, maxy, C00, C01, C10, C11,Cy, C00pass, C01pa
          , C10pass, C11pass);
197           xloop Cxloop(C00pass, C01pass, C10pass, C11pass, Cx, Cy,  C00out, C01out, C10out, C11
          , clk, reset, x, ix,y,iy, maxx);
198
199           ABC Agen(A00out, A01out, A10out, A11out, A);
200           ABC Bgen(B00out, B01out, B10out, B11out, B);
201           ABC Cgen(C00out, C01out, C10out, C11out, C);
202
203
204
205        endmodule
206
207
208        module C123(clk,reset,A00,B00,C00,Cx1,Cx2,Cx3,Cy1,Cy2,Cy3,fdx12,fdx23,fdx31,fdy12,fdy23,
          fdy31,x,y,ix,iy);
209
210           input [31:0] A00, B00, C00;
211           input [17:0] fdx12, fdx23, fdx31,fdy12, fdy23, fdy31;
212           output [31:0] Cx1, Cx2, Cx3,Cy1,Cy2,Cy3;
213           input [13:0] x,y,ix,iy;
214           input clk, reset;
215           //input [2:0] q;
216
217           reg [31:0] Cy1, Cy2, Cy3;
218           reg [31:0] Cx1, Cx2, Cx3;
219           reg [2:0] count;
220
221
222           always@(posedge clk or posedge reset)
223              begin
224                 if (reset)
225                    count <= 0;
226                 //else if (count == q)
227                 // count <= 0;
228                 else
229                    count <= count +1;
230              end
231
232           always@(posedge clk or posedge reset)
233              begin
```

```
234                if (reset)
235                   begin
236                      Cx1 <= A00;
237                      Cx2 <= B00;
238                      Cx3 <= C00;
239                   end
240                else if (ix == x)
241                   begin
242                      Cx1 <= Cy1;// +
        {fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17
        dx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17:0]};
243                      Cx2 <= Cy2;// +
        {fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17
        dx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17:0]};
244                      Cx3 <= Cy3;// +
        {fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17
        dx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17:0]};
245                   end
246                else
247                   begin
248                      Cx1 <= Cx1 - {fdy12[17],fdy12[17],fdy12[17],fdy12[17],fdy12[17],fdy12[17]
        fdy12[17],fdy12[17],fdy12[17],fdy12[17],fdy12[17],fdy12[17],fdy12[17],fdy12[17],fdy12[17
        ]};
249                      Cx2 <= Cx2 - {fdy23[17],fdy23[17],fdy23[17],fdy23[17],fdy23[17],fdy23[17]
        fdy23[17],fdy23[17],fdy23[17],fdy23[17],fdy23[17],fdy23[17],fdy23[17],fdy23[17],fdy23[17
        ]};
250                      Cx3 <= Cx3 - {fdy31[17],fdy31[17],fdy31[17],fdy31[17],fdy31[17],fdy31[17]
        fdy31[17],fdy31[17],fdy31[17],fdy31[17],fdy31[17],fdy31[17],fdy31[17],fdy31[17],fdy31[17
        ]};
251                   end
252             end
253
254          always@(posedge clk or posedge reset)
255             if (reset)
256                begin
257                   Cy1 <= A00;
258                   Cy2 <= B00;
259                   Cy3 <= C00;
260                end
261             else if ((ix == x +7) && (iy == y+7))
262                begin
263                   Cy1 <= A00;
264                   Cy2 <= B00;
265                   Cy3 <= C00;
266                end
267             else if (count == 7)
268                begin
269                   Cy1 <= Cy1 + {fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx
        [17],fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17:0]};
270                   Cy2 <= Cy2 + {fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx
        [17],fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17:0]};
271                   Cy3 <= Cy3 + {fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx
        [17],fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17:0]};
272                end
273
274       endmodule
275
276       module drawLoop(clk,reset,x,y,ix,iy,state);
277
278          input clk,reset;
279          input [13:0] x,y;
280          input [1:0] state;
281          output [13:0] ix,iy;
282
283          reg [2:0] xCount,yCount;
```

```
284          //reg [13:0] iy,diy;
285
286          always@(posedge clk, posedge reset)
287             if(reset)
288                begin
289                   xCount<=0;
290                   yCount<=0;
291                end
292             else if(state==2'b01)
293                begin
294                   xCount<=0;
295                   yCount<=0;
296                end
297             else if(xCount==7)
298                begin
299                   yCount<=yCount+1;
300                   xCount<=0;
301                end
302
303             else
304                xCount<=xCount+1;
305
306          assign iy=y+yCount;
307          assign ix=x+xCount;
308
309       endmodule
310
311       module pixelOn(Cx1,Cx2,Cx3,iy,ix,writePixel,fbAddress);
312
313          input [31:0] Cx1,Cx2,Cx3;
314          input [13:0] ix,iy;
315          output writePixel;
316          output [18:0] fbAddress;
317
318          wire [27:0] multOut;
319
320          assign writePixel=((~Cx1[31] & (|Cx1)) & (~Cx2[31] & (|Cx2)) & (~Cx3[31] & (|Cx3)));
321          mult1414u addressGenerate(iy,14'd640,multOut);
322          assign fbAddress=ix+multOut[18:0];
323
324
325       endmodule
326
```

```
1
2         //Andrew Giles 4/7/07
3
4         //This is the top level module of the 640x480 framebuffer
5         //The framebuffer runs on a 100 MHz clock
6         //when writing pixels/colors to the framebuffer, write should
7         //be synchronized to the 100 MHz clock.
8         //The relevant signals are color_data[3:0] (the pixel color)
9         // pixel_addr[18:0] the memory address to write
10        //       =(640y +x), wherex and y are the coordinates based off the upper left corner
11        // and wen, which is the write enable port for the memory;
12
13        //to modify this into a SVGA framebuffer, the relevant constants should be looked up in
14        //svga_defines.v (found in one of the built in examples files) and pasted into
          SVGA_TIMING_GENERATION.v
15        //also, the block memory size will need to be increased, as well as the bus widths of the
          pixel_addr wires
16
17
18        module VGAFB_CTRL(
19        CLOCK,
20        VGA_OUT_PIXEL_CLOCK,
21        VGA_COMP_SYNCH_N,
22        VGA_OUT_BLANK_N,
23        VGA_HSYNCH,
24        VGA_VSYNCH,
25        VGA_OUT_RED,
26        VGA_OUT_GREEN,
27        VGA_OUT_BLUE,
28        color_data,
29        pixel_addr,
30        wen
31        );
32
33        input          CLOCK;               // 100MHz system clock
34        output         VGA_OUT_PIXEL_CLOCK; // pixel clock for the video DAC
35        output         VGA_COMP_SYNCH_N;    // composite sync for the video DAC
36        output         VGA_OUT_BLANK_N;     // composite blanking for the video DAC
37        output         VGA_HSYNCH;          // horizontal sync for the VGA output connector
38        output         VGA_VSYNCH;          // vertical sync for the VGA output connector
39        output [7:0]   VGA_OUT_RED;         // RED DAC data
40        output [7:0]   VGA_OUT_GREEN;       // GREEN DAC data
41        output [7:0]   VGA_OUT_BLUE;        // BLUE DAC data
42
43        input [3:0] color_data;
44        input [18:0] pixel_addr;
45        input wen;
46
47
48
49        wire        VGA_OUT_PIXEL_CLOCK;
50        wire        VGA_COMP_SYNCH_N;
51        wire        VGA_OUT_BLANK_N;
52        wire        VGA_HSYNCH;
53        wire        VGA_VSYNCH;
54        wire [7:0]     VGA_OUT_RED;
55        wire [7:0]     VGA_OUT_GREEN;
56        wire [7:0]     VGA_OUT_BLUE;
57
58        wire CLOCK;
59        wire clock;
60        wire clk_100;
61        wire clock_100;
62        wire clk_25;
63        wire clock_25;
```

```
64        wire clk_40;
65        wire clock_40;
66        wire system_dcm_rst;
67        wire system_dcm_locked;
68        wire reset = !system_dcm_locked;
69        wire pixel_clock = clk_25;
70
71        wire low = 1'b0;
72        wire high =1'b1;
73
74
75        wire color_write_clock = clk_100;
76        wire color_write_enable = high;
77        wire [18:0] data_addr = pixel_addr;
78
79        wire [3:0] color_data_in;
80
81        assign color_data_in = color_data;
82
83
84
85
86
87
88
89
90        // instantiate the VGA controller and display RAM
91        VGAFB theFB(
92        VGA_OUT_PIXEL_CLOCK,
93        VGA_COMP_SYNCH_N,
94        VGA_OUT_BLANK_N,
95        VGA_HSYNCH,
96        VGA_VSYNCH,
97        VGA_OUT_RED,
98        VGA_OUT_GREEN,
99        VGA_OUT_BLUE,
100
101       data_addr,
102       color_data_in,
103       color_write_clock,
104       wen,
105
106
107       pixel_clock,
108
109       reset
110       );
111
112
113       // instantiate the clock input buffers for the single ended clocks
114       //IBUFG CLOCK_INPUT_BUF (
115       //.O  (clock),
116       //.I  (CLOCK)
117       //);
118
119       // instantiate the clock input buffers for the internal clocks
120       BUFG CLK_100MHZ_BUF (
121       .O  (clk_100),
122       .I  (clock_100)
123       );
124
125       BUFG CLK_25MHZ_BUF (
126       .O  (clk_25),
127       .I  (clock_25)
128       );
```

```
129
130         BUFG CLK_40MHZ_BUF (
131         .O  (clk_40),
132         .I  (clock_40)
133         );
134
135         // instantiate the DCMs and DCM reset generation
136         DCM SYSTEM_DCM (
137         .CLKFB      (clk_100),
138         .CLKIN      (CLOCK),
139         .DSSEN      (low),
140         .PSCLK      (low),
141         .PSEN    (low),
142         .PSINCDEC   (low),
143         .RST     (system_dcm_rst),
144         .CLK0    (clock_100),
145         .CLK90      (),
146         .CLK180     (),
147         .CLK270     (),
148         .CLK2X      (),
149         .CLK2X180   (),
150         .CLKDV      (clock_25),
151         .CLKFX      (clock_40),
152         .CLKFX180   (),
153         .LOCKED     (system_dcm_locked),
154         .PSDONE     (),
155         .STATUS     ()
156         )
157         /* synthesis xc_props="DLL_FREQUENCY_MODE =LOW,DUTY_CYCLE_CORRECTION =TRUE,STARTUP_WAIT
            =FALSE,DFS_FREQUENCY_MODE =LOW,CLKFX_DIVIDE =10,CLKFX_MULTIPLY =4,CLKDV_DIVIDE
            =4,CLK_FEEDBACK =1X,CLKOUT_PHASE_SHIFT =NONE,PHASE_SHIFT =0"*/;
158
159
160         SRL16 RESET_SYSTEM_DCM (
161         .Q (system_dcm_rst),
162         .CLK (CLOCK),
163         .D (low),
164         .A0 (high),
165         .A1 (high),
166         .A2 (high),
167         .A3 (high)
168         )/*synthesis xc_props="INIT = 000F"*/;
169
170
171         endmodule
172
173
174
175
176
```

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3   // Company:
4   // Engineer:
5   //
6   // Create Date:    14:18:45 04/04/2007
7   // Design Name:
8   // Module Name:    VGAFB
9   // Project Name:
10  // Target Devices:
11  // Tool versions:
12  // Description:
13  //
14  // Dependencies:
15  //
16  // Revision:
17  // Revision 0.01 - File Created
18  // Additional Comments:
19  //
20  //////////////////////////////////////////////////////////////////////////////////
21  module VGAFB(
22     VGA_OUT_PIXEL_CLOCK,
23     VGA_COMP_SYNCH_N,
24     VGA_OUT_BLANK_N,
25     VGA_HSYNCH,
26     VGA_VSYNCH,
27     VGA_OUT_RED,
28     VGA_OUT_GREEN,
29     VGA_OUT_BLUE,
30
31     color_data_addr,
32     color_data_in,
33     color_data_clock,
34     wen,
35     pixel_clock,
36     reset
37     );
38
39     output         VGA_OUT_PIXEL_CLOCK; // pixel clock for the video DAC
40     output         VGA_COMP_SYNCH_N;    // composite sync for the video DAC
41     output         VGA_OUT_BLANK_N;     // composite blanking for the video DAC
42     output         VGA_HSYNCH;          // horizontal sync for the VGA output connector
43     output         VGA_VSYNCH;          // vertical sync for the VGA output connector
44     output [7:0]   VGA_OUT_RED;         // RED DAC data
45     output [7:0]   VGA_OUT_GREEN;       // GREEN DAC data
46     output [7:0]   VGA_OUT_BLUE;        // BLUE DAC data
47
48     input [18:0] color_data_addr;
49     input [3:0] color_data_in;
50     input color_data_clock;
51     input pixel_clock;
52     input reset;
53     input wen;
54
55     wire [3:0] ram_data_out;
56     wire [18:0]ram_data_addr;
57     wire [3:0] ram_data_in = 8'b0; //never going to write data using second port
58     wire [3:0] color_data_out; //should never use this either.
59     wire [3:0] VGA_COLOR_PRIMITIVE;
60
61     wire [9:0] line_num;
62     wire [10:0] pixel_num;
63
64     wire high = 1'b1;
65     wire low = 1'b0;
```

```
66
67          fbmem FBRAM(
68             color_data_addr,
69             ram_data_addr,
70             color_data_clock,
71             pixel_clock,
72             color_data_in,
73             ram_data_in,
74             color_data_out,
75             ram_data_out,
76             high, //both ports are always on.
77             high,
78             wen, //port A is write only
79             low //port B is read only
80             );
81
82
83
84
85
86          COLOR_PIPELINE PIPE(
87             ram_data_out,
88             pixel_clock,
89             reset,
90             VGA_COLOR_PRIMITIVE
91          );
92
93          CLUT COLORLUT(
94             VGA_COLOR_PRIMITIVE,
95             high,
96             pixel_clock,
97             reset,
98             VGA_OUT_RED,
99             VGA_OUT_GREEN,
100            VGA_OUT_BLUE
101         );
102
103
104
105         // instantiate the SVGA timing generator
106      SVGA_TIMING_GENERATION SVGA_TIMING_GENERATION(
107         pixel_clock,
108         reset,
109         h_synch_delay,
110         v_synch_delay,
111         comp_synch,
112         blank,
113         line_num,
114         pixel_num,
115         ram_data_addr
116         );
117
118      assign VGA_VSYNCH = v_synch_delay;
119      assign VGA_HSYNCH = h_synch_delay;
120      assign VGA_COMP_SYNCH_N = 1'b0;//disable comp_synch; or sync on green;
121      assign VGA_OUT_BLANK_N = ~blank;
122      assign VGA_OUT_PIXEL_CLOCK = pixel_clock;
123
124
125
126
127      endmodule
128
129
130      module COLOR_PIPELINE (color_ram_data, pixel_clock, reset, aligned_color_data);
```

```
131         input  [3:0] color_ram_data;     // the color of the character
132         input     pixel_clock;        // pixel clock
133         input     reset;              // reset
134         output[3:0]  aligned_color_data; // time aligned color data
135
136         reg [3:0]   aligned_color_data;
137         reg [3:0]   color_pipe0;
138         reg [3:0]   color_pipe1;
139
140         always @ (posedge pixel_clock or posedge reset) begin
141            if (reset) begin
142               color_pipe0 <= 19'h0;
143               color_pipe1 <= 19'h0;
144               aligned_color_data <= 19'h0;
145               end
146            else begin
147               color_pipe0 <= color_ram_data;
148               color_pipe1 <= color_pipe0;
149               aligned_color_data <= color_pipe1;
150            end
151         end
152         endmodule //COLOR_PIPE
153
154
155
156
157         /*
158         module SRL16(Q, A0, A1, A2, A3, CLK, D); // synthesis syn_black_box
159         output Q;
160         input A0;
161         input A1;
162         input A2;
163         input A3;
164         input CLK;
165         input D;
166         endmodule
167
168         module IBUFG(O, I); // synthesis syn_black_box
169         output O;
170         input I;
171         endmodule
172
173         module BUFG(O, I); // synthesis syn_black_box
174         output O;
175         input I;
176         endmodule
177
178         module DCM(CLKFB, CLKIN, DSSEN, PSCLK, PSEN, PSINCDEC, RST,
179           CLK0, CLK90, CLK180, CLK270, CLK2X, CLK2X180, CLKDV, CLKFX, CLKFX180, LOCKED, PSDONE,
          STATUS); // synthesis syn_black_box
180         input CLKFB, CLKIN, DSSEN;
181         input PSCLK, PSEN, PSINCDEC, RST;
182         output CLK0, CLK90, CLK180, CLK270, CLK2X, CLK2X180;
183         output CLKDV, CLKFX, CLKFX180, LOCKED, PSDONE;
184         output [7:0] STATUS;
185         endmodule
186         */
```

```
//
//     XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS"
//     SOLELY FOR USE IN DEVELOPING PROGRAMS AND SOLUTIONS FOR
//     XILINX DEVICES.  BY PROVIDING THIS DESIGN, CODE, OR INFORMATION
//     AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE, APPLICATION
//     OR STANDARD, XILINX IS MAKING NO REPRESENTATION THAT THIS
//     IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT,
//     AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE
//     FOR YOUR IMPLEMENTATION.  XILINX EXPRESSLY DISCLAIMS ANY
//     WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE
//     IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR
//     REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF
//     INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
//     FOR A PARTICULAR PURPOSE.
//
//     (c) Copyright 2005 Xilinx, Inc.
//     All rights reserved.
//
/*
-------------------------------------------------------------------------------
   Title      : AUDIO FILTER DEMO DESIGN FOR THE XUP-V2Pro
   Project    : XUP-V2Pro
-------------------------------------------------------------------------------

   File       : SVGA_TIMING_GENERATION.v
   Company    : Xilinx, Inc.
   Created    : 2004/07/22
   Last Update: 2005/01/20
   Copyright  : (c) Xilinx Inc, 2005
-------------------------------------------------------------------------------
    Uses       : SVGA_DEFINES.v
-------------------------------------------------------------------------------
   Used by    : COLOR_CHAR_MODE_SVGA_CTRL
-------------------------------------------------------------------------------
   Description: This module creates the timing and control signals for the
      VGA output. The module provides character-mapped addressing
      in addition to the control signals for the DAC and the VGA output connector.
      The design supports screen resolutions up to 1024 x 768. The user will have
      to add the charcater memory RAM and the character generator ROM and create
      the required pixel clock.

      The video mode used is defined in the svga_defines.v file.

   Conventions:
      All external port signals are UPPER CASE.
      All internal signals are LOWER CASE and are active HIGH.


-------------------------------------------------------------------------------
*/


`define ZBT_PIPELINE_DELAY        2 // not required for the XUP-V2Pro
`define ZBT_INTERFACE_DELAY   2  // not required for the XUP-V2Pro
`define CHARACTER_DECODE_DELAY  5


//  640 X 480 @ 60Hz with a 25.175MHz pixel clock
`define H_ACTIVE     640   // pixels
`define H_FRONT_PORCH   16 // pixels
`define H_SYNCH         96 // pixels
`define H_BACK_PORCH 48 // pixels
`define H_TOTAL         800   // pixels

`define V_ACTIVE     480   // lines
```

```
66        `define V_FRONT_PORCH   11 // lines
67        `define V_SYNCH         2  // lines
68        `define V_BACK_PORCH 31 // lines
69        `define V_TOTAL         524   // lines
70
71        module SVGA_TIMING_GENERATION (
72        pixel_clock,
73        reset,
74        h_synch_delay,
75        v_synch_delay,
76        comp_synch,
77        blank,
78        line_count,
79        pixel_count,
80        pixel_addr
81        );
82
83        input          pixel_clock;   // pixel clock
84        input          reset;         // reset
85        output         h_synch_delay; // horizontal synch for VGA connector
86        output         v_synch_delay; // vertical synch for VGA connector
87        output         comp_synch;    // composite synch for DAC
88        output         blank;         // composite blanking
89        output [9:0]   line_count;    // line counter for current pixel
90        output [10:0]   pixel_count;    // column counter for current pixel
91        output [18:0]  pixel_addr; // character mode address
92
93        reg [9:0]      line_count;    // counts the display lines
94        reg [10:0]     pixel_count;   // counts the pixels in a line
95        reg [18:0]     pixel_addr;
96        reg            h_synch;    // horizontal synch
97        reg            v_synch;    // vertical synch
98        reg            h_synch_delay; // h_synch delayed 2 clocks to line up with DAC pipeline
99        reg            v_synch_delay; // v_synch delayed 2 clocks to line up with DAC pipeline
100       reg            h_synch_delay0;   // h_synch delayed 1 clock
101       reg            v_synch_delay0;   // v_synch delayed 1 clock
102
103       reg            h_c_synch;     // horizontal component of comp synch
104       reg            v_c_synch;     // vertical component of comp synch
105       reg            comp_synch;    // composite synch for DAC
106       reg            h_blank;    // horizontal blanking
107       reg            v_blank;    // vertical blanking
108       reg            blank;         // composite blanking
109
110
111
112
113
114       // CREATE THE HORIZONTAL LINE PIXEL COUNTER
115       always @ (posedge pixel_clock or posedge reset) begin
116          if (reset)
117             begin               // on reset set pixel counter to 0
118                pixel_count <= 11'h000;
119             end
120
121          else if (pixel_count == (`H_TOTAL - 1))
122             begin                  // last pixel in the line
123                pixel_count <= 11'h000;    // reset pixel counter
124             end
125
126          else  begin
127                pixel_count <= pixel_count +1;
128             end
129          end
130
```

```
131         // CREATE THE HORIZONTAL SYNCH PULSE
132         always @ (posedge pixel_clock or posedge reset) begin
133            if (reset)
134               begin                // on reset
135                  h_synch <= 1'b0;     // remove h_synch
136               end
137
138            else if (pixel_count == (`H_ACTIVE + `H_FRONT_PORCH -1))
139               begin             // start of h_synch
140                  h_synch <= 1'b1;
141               end
142
143            else if (pixel_count == (`H_TOTAL - `H_BACK_PORCH -1))
144               begin             // end of h_synch
145                  h_synch <= 1'b0;
146               end
147            end
148
149
150         // CREATE THE VERTICAL FRAME LINE COUNTER
151         always @ (posedge pixel_clock or posedge reset) begin
152            if (reset)
153               begin                   // on reset set line counter to 0
154                  line_count <= 10'h000;
155               end
156
157            else if ((line_count == (`V_TOTAL - 1))&& (pixel_count == (`H_TOTAL - 1)))
158               begin                   // last pixel in last line of frame
159                  line_count <= 10'h000;     // reset line counter
160               end
161
162            else if ((pixel_count == (`H_TOTAL - 1)))
163               begin                   // last pixel but not last line
164                  line_count <= line_count + 1;// increment line counter
165               end
166            end
167
168         // CREATE THE VERTICAL SYNCH PULSE
169         always @ (posedge pixel_clock or posedge reset) begin
170            if (reset)
171               begin                   // on reset
172                  v_synch = 1'b0;            // remove v_synch
173               end
174
175            else if ((line_count == (`V_ACTIVE + `V_FRONT_PORCH -1) &&
176                  (pixel_count == `H_TOTAL - 1)))
177               begin                   // start of v_synch
178                  v_synch = 1'b1;
179               end
180
181            else if ((line_count == (`V_TOTAL - `V_BACK_PORCH - 1))  &&
182                  (pixel_count == (`H_TOTAL - 1)))
183               begin                   // end of v_synch
184                  v_synch = 1'b0;
185               end
186            end
187
188         // ADD TWO PIPELINE DELAYS TO THE SYNCHs COMPENSATE FOR THE DAC PIPELINE DELAY
189         always @ (posedge pixel_clock or posedge reset) begin
190            if (reset)
191               begin
192                  h_synch_delay0 <= 1'b0;
193                  v_synch_delay0 <= 1'b0;
194                  h_synch_delay  <= 1'b0;
195                  v_synch_delay  <= 1'b0;
```

```
196
197            end
198         else
199            begin
200               h_synch_delay0 <= h_synch;
201               v_synch_delay0 <= v_synch;
202               h_synch_delay  <= h_synch_delay0;
203               v_synch_delay  <= v_synch_delay0;
204            end
205         end
206
207
208
209      // CREATE THE HORIZONTAL BLANKING SIGNAL
210      // the "-2" is used instead of "-1" because of the extra register delay
211      // for the composite blanking signal
212      always @ (posedge pixel_clock or posedge reset) begin
213         if (reset)
214            begin             // on reset
215               h_blank <= 1'b0;  // remove the h_blank
216            end
217
218         else if (pixel_count == (`H_ACTIVE -2))
219            begin             // start of HBI
220               h_blank <= 1'b1;
221            end
222
223         else if (pixel_count == (`H_TOTAL -2))
224            begin             // end of HBI
225               h_blank <= 1'b0;
226            end
227         end
228
229
230      // CREATE THE VERTICAL BLANKING SIGNAL
231      // the "-2" is used instead of "-1"  in the horizontal factor because of the extra
232      // register delay for the composite blanking signal
233      always @ (posedge pixel_clock or posedge reset) begin
234         if (reset)
235            begin                // on reset
236               v_blank <= 1'b0;     // remove v_blank
237            end
238
239         else if ((line_count == (`V_ACTIVE - 1) &&
240               (pixel_count == `H_TOTAL - 2)))
241            begin                // start of VBI
242               v_blank <= 1'b1;
243            end
244
245         else if ((line_count == (`V_TOTAL - 1)) &&
246               (pixel_count == (`H_TOTAL - 2)))
247            begin                // end of VBI
248               v_blank <= 1'b0;
249            end
250         end
251
252
253      // CREATE THE COMPOSITE BANKING SIGNAL
254      always @ (posedge pixel_clock or posedge reset) begin
255         if (reset)
256            begin                // on reset
257               blank <= 1'b0;       // remove blank
258            end
259
260         else if (h_blank || v_blank)  // blank during HBI or VBI
```

```
261             begin
262               blank <= 1'b1;
263             end
264          else begin
265                blank <= 1'b0;       // active video do not blank
266             end
267          end
268
269       // CREATE THE HORIZONTAL COMPONENT OF COMP SYNCH
270       // the "-2" is used instead of "-1" because of the extra register delay
271       // for the composite synch
272       always @ (posedge pixel_clock or posedge reset) begin
273          if (reset)
274             begin             // on reset
275                h_c_synch <= 1'b0;   // remove h_c_synch
276             end
277
278          else if (pixel_count == (`H_ACTIVE + `H_FRONT_PORCH -2))
279             begin             // start of h_c_synch
280                h_c_synch <= 1'b1;
281             end
282
283
284          else if (pixel_count == (`H_TOTAL - `H_BACK_PORCH -2))
285             begin             // end of h_c_synch
286                h_c_synch <= 1'b0;
287             end
288          end
289
290       // CREATE THE VERTICAL COMPONENT OF COMP SYNCH
291       always @ (posedge pixel_clock or posedge reset) begin
292          if (reset)
293             begin                // on reset
294                v_c_synch <= 1'b0;      // remove v_c_synch
295             end
296
297          else if ((line_count == (`V_ACTIVE + `V_FRONT_PORCH - 1) &&
298                (pixel_count == `H_TOTAL - 2)))
299             begin                // start of v_c_synch
300                v_c_synch <= 1'b1;
301             end
302
303          else if ((line_count == (`V_TOTAL - `V_BACK_PORCH - 1))  &&
304                (pixel_count == (`H_TOTAL - 2)))
305             begin                // end of v_c_synch
306                v_c_synch <= 1'b0;
307             end
308          end
309
310       // CREATE THE COMPOSITE SYNCH SIGNAL
311       always @ (posedge pixel_clock or posedge reset) begin
312          if (reset)
313             begin                // on reset
314                comp_synch <= 1'b0;     // remove comp_synch
315             end
316          else begin
317             comp_synch <= (v_c_synch ^ h_c_synch);
318             end
319          end
320
321       //CREATE THE PIXEL ADDRESS COUNTER
322       always@(posedge pixel_clock or posedge reset)
323       begin
324          if (reset)
325             pixel_addr <= 19'b0;
```

```
326            else if (pixel_addr == 307199)
327               pixel_addr <= 19'b0;
328            else if (~blank)
329               pixel_addr <= pixel_addr +1;  //don't want to worry about negative addresses
          (nothing displayed)
330
331            //if (pixel_addr == 307199) //reset at the end of the screen
332            // pixel_addr <= 19'b0;
333
334
335         end
336
337
338
339
340
341         endmodule //SVGA_TIMING_GENERATION
342
343
344
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    09:57:15 04/21/2007
// Design Name:
// Module Name:    GPUtop
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module GPUtop(CLOCK,
reset,
VGA_OUT_PIXEL_CLOCK,
VGA_COMP_SYNCH_N,
VGA_OUT_BLANK_N,
VGA_HSYNCH,
VGA_VSYNCH,
VGA_OUT_RED,
VGA_OUT_GREEN,
VGA_OUT_BLUE);

input CLOCK;
input reset;
output         VGA_OUT_PIXEL_CLOCK; // pixel clock for the video DAC
output         VGA_COMP_SYNCH_N;    // composite sync for the video DAC
output         VGA_OUT_BLANK_N;     // composite blanking for the video DAC
output         VGA_HSYNCH;          // horizontal sync for the VGA output connector
output         VGA_VSYNCH;          // vertical sync for the VGA output connector
output [7:0]   VGA_OUT_RED;         // RED DAC data
output [7:0]   VGA_OUT_GREEN;       // GREEN DAC data
output [7:0]   VGA_OUT_BLUE;        // BLUE DAC data

reg [1:0] start;
wire [18:0] FBaddress;

wire [31:0] multout1;
wire [31:0] multout2;
wire wen;


always@(posedge CLOCK or posedge reset)
begin
   if (reset)
      start <= 0;
   else if (start == 2'b0)
      start <= 2'b1;
   else if (start == 2'b1)
      start <= 2'b10;
end

mult multply1(14'd6, 18'd6, multout1);
mult multply2(14'd9, 18'd9, multout2);


drawline linegpu(14'b1, 14'b1, multout2[13:0],multout1[13:0], start[0], reset, CLOCK,
```

```
        FBaddress, 19'd641, wen);
66
67      FB_test theFB(
68      CLOCK,
69      VGA_OUT_PIXEL_CLOCK,
70      VGA_COMP_SYNCH_N,
71      VGA_OUT_BLANK_N,
72      VGA_HSYNCH,
73      VGA_VSYNCH,
74      VGA_OUT_RED,
75      VGA_OUT_GREEN,
76      VGA_OUT_BLUE,
77      4'b1011,
78      FBaddress,
79      wen
80      );
81
82
83
84
85
86
87      endmodule
88
```

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3   // Company:
4   // Engineer: Andrew Giles and Phil Amberg
5   //
6   // Create Date:    09:57:15 04/21/2007
7   // Design Name:
8   // Module Name:    GPUtop
9   // Project Name:
10  //
11  //////////////////////////////////////////////////////////////////////////////////
12  module GPUtop(CLOCK,
13  reset,
14  VGA_OUT_PIXEL_CLOCK,
15  VGA_COMP_SYNCH_N,
16  VGA_OUT_BLANK_N,
17  VGA_HSYNCH,
18  VGA_VSYNCH,
19  VGA_OUT_RED,
20  VGA_OUT_GREEN,
21  VGA_OUT_BLUE);
22
23  input CLOCK;
24  input reset;
25  output         VGA_OUT_PIXEL_CLOCK; // pixel clock for the video DAC
26  output         VGA_COMP_SYNCH_N;    // composite sync for the video DAC
27  output         VGA_OUT_BLANK_N;     // composite blanking for the video DAC
28  output         VGA_HSYNCH;          // horizontal sync for the VGA output connector
29  output         VGA_VSYNCH;          // vertical sync for the VGA output connector
30  output [7:0]   VGA_OUT_RED;         // RED DAC data
31  output [7:0]   VGA_OUT_GREEN;       // GREEN DAC data
32  output [7:0]   VGA_OUT_BLUE;        // BLUE DAC data
33
34  reg [1:0] start;
35  wire [18:0] FBaddress;
36
37  wire [31:0] multout1;
38  wire [31:0] multout2;
39  wire wen;
40
41  //generate the 1 cycle start signal
42  always@(posedge CLOCK or posedge reset)
43  begin
44     if (reset)
45        start <= 0;
46     else if (start == 2'b0)
47        start <= 2'b1;
48     else if (start == 2'b1)
49        start <= 2'b10;
50  end
51
52  mult multply1(14'd6, 18'd6, multout1);
53  mult multply2(14'd9, 18'd9, multout2);
54
55  //pass everything into the lie drawing hardware, pixel coordinates, start, reset, etc.
56  drawline linegpu(14'b1, 14'b1, multout2[13:0],multout1[13:0], start[0], reset, CLOCK,
    FBaddress, 19'd641, wen);
57
58  FB_test theFB(
59  CLOCK,
60  VGA_OUT_PIXEL_CLOCK,
61  VGA_COMP_SYNCH_N,
62  VGA_OUT_BLANK_N,
63  VGA_HSYNCH,
64  VGA_VSYNCH,
```

```
65          VGA_OUT_RED,
66          VGA_OUT_GREEN,
67          VGA_OUT_BLUE,
68          4'b1011,
69          FBaddress,
70          wen
71          );
72
73
74
75
76
77
78          endmodule
79
```

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3   // Company:
4   // Engineer: Andrew Giles
5   //
6   // Create Date:    09:58:10 04/21/2007
7   // Design Name:
8   // Module Name:    GPUprecompute
9   //////////////////////////////////////////////////////////////////////////////////
10
11  //How to draw a line:
12  //this module reqires:
13  // x1, y1, x2,y2: the vertices of the line
14  // start: a 1 cycle pulse to begin drawing the line
15  // reset: pulse high to load the x1,y1,x2,y2 values into assorted counters
16  //       note that this means your vertex values must be set before hitting reset
17  // addrinitial: the framebuffer address of the first pixel of the line;
18  // outputs:
19  // fbAdress: the current pixel address
20  // wen: write enable signal for the framebuffer
21  module drawline(
22        x1, y1, x2,y2, start, reset, clock, fbAddress, addrinitial,wen
23     );
24
25
26     input [13:0] y1, y2;
27     input [13:0] x1, x2;
28     input start;
29     input reset;
30     input clock;
31     output [18:0] fbAddress;
32     input [18:0] addrinitial;
33     output wen;
34
35     wire [13:0] xstart;
36     wire [13:0] ystart;
37     wire [13:0] yend;
38     wire [13:0] xend;
39     reg  [13:0] ystep;
40     wire [13:0] dx;
41     wire [13:0] dy;
42     wire [13:0] dxx;
43     wire [13:0] dyy;
44     reg steep;
45
46
47     wire [13:0] xs1;
48     wire [13:0] xs2;
49     wire [13:0] ys1;
50     wire [13:0] ys2;
51
52
53
54     reg [13:0] x;
55     reg [13:0] err;
56     reg [13:0] y;
57     reg clken;
58
59     reg [18:0] xaddr;
60     reg [18:0] yaddr;
61
62     assign dx = x2 - x1;
63     assign dy = y2 - y1;
64
65     /*assign dx2comp = ~dx +1;
```

```
66          assign dy2comp = ~dy +1;
67
68          mux2(dx, dx2comp, dx[12], dxout);
69          mux2(dy, dy2comp, dy[13], dyout);
70
71          dxdy = dxout - dyout;
72
73          assign steep = dxdy[13];
74          */
75
76
77
78          //assign steep = abs(dy) > abs(dx);
79          wire [13:0] absdy, absdx;
80          wire [13:0] notdx, notdy;
81
82
83
84          //assign absdx = {1'b0, ~dx[13], dx[12:0]};
85          //assign absdy = {1'b0, ~dy[13], dy[12:0]};
86
87          assign notdx = (~dx+1'b1);
88          assign notdy = (~dy+1'b1);
89
90          muxtwo absdxmux(dx, notdx, dx[13], absdx);
91          muxtwo absdymux(dy, notdy, dy[13], absdy);
92
93
94          always@( * )
95          begin
96             if ({1'b0, absdy} > {1'b0,absdx})
97                steep = 1'b1;
98             else
99                steep = 1'b0;
100         end
101
102         //assign ys1 = y1;
103         //assign ys2 = x1;
104         //assign xs1 = x1;
105         //assign xs2 = x2;
106
107         assign xs1 = steep ? y1 : x1;
108         assign ys1 = steep ? x1 : y1;
109         assign xs2 = steep ? y2 : x2;
110         assign ys2 = steep ? x2 : y2;
111
112         assign leftright = {1'b0, xs1} > {1'b0, x2};
113
114         assign xstart = leftright ? xs2 : xs1;
115         assign xend = leftright ? xs1 : xs2;
116         muxtwo ysmux1(ys1, ys2, leftright, ystart);
117         muxtwo ysmux2(ys2, ys1, leftright, yend);
118
119         assign dxx = xend - xstart;
120         assign dyy = yend - ystart;
121
122         always@( * )
123         begin
124            if ({1'b0, ystart} < {1'b0, yend})
125               ystep = 1;
126            else
127               ystep = -1;
128         end
129
130         //clock enable/disable
```

```
131
132           //need to fix this;
133           assign xenable = start | clken;
134           always@(posedge clock or posedge reset )
135           begin
136              if (reset)
137                 clken <= 1;
138              else if (({1'b0, x} == {1'b0, xend}))
139                 clken <= 0;
140           end
141
142           //the x value counter
143           always@(posedge clock or posedge reset)
144           begin
145              if (reset)
146              begin
147                 x <= xstart;
148                 xaddr <= addrinitial;
149              end
150              else if ({1'b0, x} == {1'b0, xend})
151              begin
152                 x <= xstart;
153                 xaddr <= addrinitial;
154              end
155              else if (xenable)
156              begin
157                 x <= x+1;
158                 xaddr <= xaddr+14'd640;
159              end
160           end
161
162           //the error calculation;
163           wire [13:0] errstart;
164           assign errstart = dx>>1;
165
166           always@(posedge clock or posedge reset)
167           begin
168              if (reset)
169              begin
170                 err <= errstart;
171                 y <= ystart;
172                 yaddr <= addrinitial;
173              end
174              else if ({1'b0, x} == {1'b0, xend})
175              begin
176                 y <= ystart;
177                 yaddr <= addrinitial;
178              end
179              else if (xenable && (err[13] == 1'b0))
180              begin
181                 err <= err - dyy;
182              end
183              else if (xenable && (err[13] == 1'b1))
184                 begin
185                    y <= y+ystep;
186                    yaddr <= yaddr + 14'd640;
187                    err <= err + dxx - dyy;
188                 end
189           end
190
191
192           //generate output address: 1 port memory
193           //assign addr1 = x*640+y;
194           //assign addr2 = y*640+x;
195           wire [18:0] addra;
```

```
196
197            wire [18:0] xhat, yhat;
198
199            assign xhat = steep ? xaddr : yaddr;
200            assign yhat = steep ? y : x;
201            assign addra = xhat;
202            //mult multiply({4'b0000, xhat[13:4]}, 14'd640, addra);
203            //assign addra = {4'b0000, xhat[13:4]}* 14'd640;
204            assign fbAddress = addra + yhat;
205
206            //the enable signal for the fbAddress;
207            assign wen = xenable;
208
209
210
211
212
213        endmodule
214
215
216        module muxtwo(d0, d1, s, dout);
217
218            input [13:0] d0, d1;
219            input s;
220            output [13:0] dout;
221
222            assign dout = s ? d1 : d0;
223        endmodule
224
225
226        module bigmuxtwo(d0, d1, s, dout);
227
228            input [18:0] d0, d1;
229            input s;
230            output [18:0] dout;
231
232            assign dout = s? d1:d0;
233        endmodule
234
235
236
237
```