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\(^2\).](http://en.wikipedia.org/wiki/Bresenham's_line_algorithm)

\(^1\) 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

\[
\begin{align*}
(y - y_o) &= \frac{y_1 - y_o}{x_1 - x_o} (x - x_o) \quad \rightarrow \quad (x_1 - x_o)(y - y_o) - (y_1 - y_o)(x - x_o) = 0 \\
(y - y_1) &= \frac{y_2 - y_1}{x_2 - x_1} (x - x_1) \quad \rightarrow \quad (x_2 - x_1)(y - y_1) - (y_2 - y_1)(x - x_1) = 0 \\
(y - y_2) &= \frac{y_o - y_2}{x_o - x_2} (x - x_2) \quad \rightarrow \quad (x_0 - x_2)(y - y_2) - (y_o - y_2)(x - x_2) = 0
\end{align*}
\]

(1)

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.

\[\text{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.](image)

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:

\[ FbAddress = x + 640y \]  \hspace{1cm} (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.
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"
#include <xparameters.h> //define io devices
#include <xuartlite_1.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)

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=");
    }
```c
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 triangle

} 

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("%d.d.x",i);
    (vertices[i]).x=userInput(" "); //assign x,y pts to vertices
    printf("%d.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
```
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 )
    
    userChoice = inputArray[i];
```c
226         { i=i+1;
227             inputArray[i] = XUartLite_RecvByte(XPAR_RS232_UART_1_BASEADDR);
228             if(inputArray[i]!=13)
229                 { printf("%d",ascii2dec(inputArray[i]));
230                     }
231                 if(i==2)
232                     { userChoice=ascii2dec(inputArray[1]);
233                     } else if(i==3)
234                     { userChoice=10*ascii2dec(inputArray[1])+ascii2dec(inputArray[2]);
235                     } else
236                     { userChoice=100*ascii2dec(inputArray[1])+10*ascii2dec(inputArray[2])+ascii2dec(in
237                     }
238                 printf("\r\n");
239             return userChoice;
240         }
241         int ascii2dec(int ascii)
242         { int dec;
243             //LUT
244             switch (ascii)
245             { case 48: dec=0; break;
246                case 49: dec=1; break;
247                case 50: dec=2; break;
248                case 51: dec=3; break;
249                case 52: dec=4; break;
250                case 53: dec=5; break;
251                case 54: dec=6; break;
252                case 55: dec=7; break;
253                case 56: dec=8; break;
254                case 57: dec=9; break;
255                default: dec=0; break;
256             }
257             return dec;
258         }
259         /* function max3: this function returns the maximum value of eihter x1, x2, or x3 */
260         int max3(int x1, int x2, int x3)
261         { if(x1>x2)
262             { if(x1>x3)
263                 { return x1;
264                     }
265             } else
266             { return x2;
267             }
268         } else
269         { return x3;
270         }
271         /* function min3: this function returns the minimum value of eihter x1, x2, or x3 */
272         int min3(int x1, int x2, int x3)
273         { if(x1<x2)
274             { if(x1<x3)
275                 { return x1;
276                     }
277             } else
278             { return x2;
279             }
280         } else
281         { return x3;
282         }
283         }
284         */
285 ```
if(x2<x3) {
  return x2;
} else {
  return x3;
}

/* function clearVirtScreen: this function takes the active screen as input and returns a blank screen (filled with 0)
*/
int clearVirtScreen(char (screen[20])[50]) {
  char i,j;
  for(i=0;i<20;i++) {
    for(j=0;j<50;j++) {
      screen[i][j]=0;
    }
  }
}

/* function printScreen: this function prints the contents of the screen buffer to the console
*/
int printScreen(char (screen[40])[50]) {
  char i,j;
  int fbAddress=0;
  printf("\n\n");
  for(i=0;i<40;i++) {
    for(j=0;j<50;j++) {
      fbAddress=j+640*i;
      //write peripheral((fbAddress<<13)|(screen[i][j]));
      VGADRIVER_mWriteSlaveReg0(XPAR_VGADRIVER_0_BASEADDR, (fbAddress<<13)|(screen[i][j]));
      printf("%d",screen[i][j]);
    }
  }
  printf("\n\n");
}

/* function printMenu: this function prints the menu of options to the console, this function serves little purpose without the ability for scanf()
*/
int printMenu() {
  printf("\n\nTriangle Drawing Program!\n\nWhat would you like to do?\n\n" (0) Draw Triangle from vertices\n" (1) Draw Quad from vertices\n" (2) Draw N-gon from vertices\n" (3) Draw line from vertices\n" (4) Draw point from vertex\n" (9) Quit\n"
  return 0;
}

/* function drawQuad: this function uses the triangleFill algorithm to draw a quad. two triangles are combined to make a quad. the input vertices v1,v2,v3,v4 must be defined in a counterclockwise order
*/
int drawQuad(vertex v1, vertex v2, vertex v3, vertex v4, int color, char (*screen)[50]) {
  int i,j;
  triangleFill( v1, v2, v3, color,screen); //draw triangle 1
  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  /   \\
    | \
    |  
    \\  
    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=abs(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=ytemp;
            yend=ytemp;
        }
        //recompute dx and dy
        dx=xend-xstart;
        dy=abs(yend-ystart);
int x=xstart;
int y=ystart;

if(steep) //swap variables if steep
{
    screen[x][y]=color;
}
else
{
    screen[y][x]=color;
}

int error=dx>>1;
if(ystart<yend)
{
    ystep=1;
}
else
{
    ystep=-1;
}

while(x<xend)
{
    //step in fast direction
    x=x+1;
    error=error-dy;
    if(error<0)
    {
        //step in slow direction
        y=y+ystep;
        error=error+dx;
    }
    if(steep)
    {
        //screen[x][y]=color;
        fbAddress=x*640+y;
        //write peripheral((fbAddress<<13)|(screen[i][j]));
        VGADRIVER_mWriteSlaveReg0(XPAR_VGADRIVER_0_BASEADDR, (fbAddress<<13)|color);
    }
    else
    {
        //screen[y][x]=color;
        fbAddress=y*640+x;
        //write peripheral((fbAddress<<13)|(screen[i][j]));
        VGADRIVER_mWriteSlaveReg0(XPAR_VGADRIVER_0_BASEADDR, (fbAddress<<13)|color);
    }
}

/* function drawPoint: this function simply draws a point on the screen */
int drawPoint(vertex v1, int color, char (*screen)[50])
{
    int fbAddress;
    //screen[v1.y][v1.x]=color;
    fbAddress=v1.y*640+v1.x;
    //write peripheral((fbAddress<<13)|(screen[i][j]));
    VGADRIVER_mWriteSlaveReg0(XPAR_VGADRIVER_0_BASEADDR, (fbAddress<<13)|color);
}

/* function triangleFill: this function draws a triangle given 3 vertices.
   it works by computing the half space functions for the three lines
   defined by {v1,v2},{v2,v3},{v3,v1}. if all the half space functions
   are positive, the point is inside the triangle. this algorithm is
   sped up by processing the screen in 8x8 pixel blocks. we can check
   the corners of the block to determine if the block is empty, full,
   or partially covered. this speeds up processing and leaves opportunities
   for hardware acceleration and parallelism. this algorithm was adapted
   from an online tutorial by Nicolas Capens, http://www.devmaster.net/forums/showthread.php?t=1884
/*
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=DX12*Y1-DX12*X1;
C2=DX23*X2-DX23*Y2;
C3=DX31*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 partially 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)|((A01>0)<<1)|((A10>0)<<2)|((A11>0)<<3));
        B=((B00>0)|((B01>0)<<1)|((B10>0)<<2)|((B11>0)<<3));
        C=((C00>0)|((C01>0)<<1)|((C10>0)<<2)|((C11>0)<<3));

        Axacc=Axacc+Ax;
        Bxacc=Bxacc+Bx;
        Cxacc=Cxacc+Cx;
A00 = A00 - Ax;
A10 = A10 - Ax;
A01 = A01 - Ax;
A11 = A11 - Ax;
B00 = B00 - Bx;
B10 = B10 - Bx;
B01 = B01 - Bx;
B11 = B11 - Bx;
C00 = C00 - Cx;
C10 = C10 - Cx;
C01 = C01 - Cx;
C11 = C11 - Cx;

if{(A==0 || B==0 || C==0) && ACTIVATED==1}
{
    break;  // when empty blocks are encountered to the right
    // of a block that had pixels drawn, advance to the
    // next row
}
else    // when we were not skipping right hand blocks
{
    // if all the corners have outside edges, we can skip it
    if(A==0 || B==0 || C==0 ){}

    // accept whole block when all corners are inside triangle
    else if(A==15 && B==15 && C==15)
    {
        // printf("In whole block\r\n");
        ACTIVATED=1;
        for(iy=y; iy<y+q; iy++)
        {
            for(ix=x; ix<=x+q; ix++)
            {
                // screen[iy][ix]=2;
                fbAddress=iy*640+ix;
                // write peripheral((fbAddress<<13)|screen[i][j]);
                VGADRIVER_mWriteSlaveReg0(XPAR_VGADRIVER_0_BASEADDR, (fbAddress<<
                }
            }
        }
    }
    // partially covered blocks
    else
    {
        ACTIVATED=1;
        // printf("In partially loop\r\n");
        Cy1=A00+Ax;
        Cy2=B00+Bx;
        Cy3=C00+Cx;
        for(iy=y; iy<y+q; iy++)
        {
            Cx1=Cy1;
            Cx2=Cy2;
            Cx3=Cy3;
            for(ix=x; ix<=x+q; ix++)
            {
                if(Cx1>0 && Cx2>0 && Cx3>0)
                {
                    //screen[iy][ix]=color;
                    fbAddress=iy*640+ix;
                    // write peripheral((fbAddress<<13)|screen[i][j]);
                    VGADRIVER_mWriteSlaveReg0(XPAR_VGADRIVER_0_BASEADDR, (fbAddress<<
                    }
                }
            }
        }
    }
    Cy1=Cy1+FDX12;
    Cy2=Cy2-FDY12;
    Cy3=Cy3-FDY31;
}

C01=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;
}
module toplevelrun(
    CLOCK,
    VGA_OUT_PIXEL_CLOCK,  // pixel clock for the video DAC
    VGA_COMP_SYNCH_N,     // composite sync for the video DAC
    VGA_OUT_BLANK_N,      // composite blanking for the video DAC
    VGA_HSYNCH,           // horizontal sync for the VGA output connector
    VGA_VSYNCH,           // vertical sync for the VGA output connector
    VGA_OUT_RED,          // RED DAC data
    VGA_OUT_GREEN,        // GREEN DAC data
    VGA_OUT_BLUE);        // BLUE DAC data

input    CLOCK;       // 100MHz system clock
output   VGA_OUT_PIXEL_CLOCK;
output   VGA_COMP_SYNCH_N;
output   VGA_OUT_BLANK_N;
output   VGA_HSYNCH;
output   VGA_VSYNCH;
output   [7:0] VGA_OUT_RED;
output   [7:0] VGA_OUT_GREEN;
output   [7:0] VGA_OUT_BLUE;

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

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

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


module toplevel(clk, reset, q, x1, x2, x3, y1, y2, y3, writePixel, fbAddress);

input clk;
input reset;
input [13:0] q;
input [9:0] x1, x2, x3;
input [9:0] y1, y2, y3;
output [18:0] fbAddress;
output writePixel;

wire [13:0] x1s, x2s, x3s, y1s, y2s, y3s;
wire [13:0] dx12, dx23, dx31, dy12, dy23, dy31;
wire [17:0] fdx12, fdx23, fdx31, fdy12, fdy23, fdy31;
wire [13:0] maxx, maxy;
wire [13:0] minxAligned, minyAligned;
wire [27:0] c1, c2, c3;
wire [27:0] c1out, c2out, c3out;
wire [13:0] minx, miny;
wire [17:0] miny7, minx7, minxShift, minyShift;
wire [31:0] dx12miny, dy12minx, dx12miny7, dy12minx7, dx23miny, dy23minx, dx23miny7, dy23minx7, dx31miny, dy31minx, dx31miny7, dy31minx7;
wire [20:0] Ay, By, Cy, Ax, Bx, Cx;
wire [31:0] a00, a01, a10, a11, b00, b01, b10, b11, c00, c01, c10, c11;

computeConstants generateConstants(x1, x2, x3, y1, y2, y3, x1s, x2s, x3s, y1s, y2s, y3s, dx12, dx23, dx31, dy12, dy23, dy31);

minmax3 boundingBox(x1s, x2s, x3s, y1s, y2s, y3s, minx, miny, maxx, maxy);
blockAlign startAtPow2(minx, miny, q, minxAligned, minyAligned);
initHalfEdge halfEdgeConstants(x1s, x2s, x3s, y1s, y2s, y3s, dx12, dx23, dx31, dy12, dy23, dy31, c1, c2, c3);
fillConvention adjustConstants(dy12, dx12, dy23, dx23, dx31, dy31, c1, c2, c3, c1out, c2out, c3out);
qShiftPlus7 qShift(minxAligned, minyAligned, miny7, minx7, minxShift, minyShift);
preMults preCompute(dx12, dx23, dx31, dy12, dy23, dy31, minxShift, miny7, miny7, dx12miny, dy12minx, dx12miny7, dy12minx7, dx23miny, dy23minx, dx23miny7, dy23minx7, dx31miny, dy31minx, dx31miny7, dy31minx7, Ay, By, Cy, Ax, Bx, Cx);

halfEdgeIncrement halfEdgeInc(dx12, dx23, dx31, dy12, dy23, dy31, a00, a01, a10, a11, b00, b01, b10, b11, c00, c01, c10, c11);

constHalfEdge theConst(c1out, dx12miny, dy12minx, dx12miny7, dy12minx7, a00, a10, a01, a11, c00, c10, c01, c11, dx23miny, dy23minx, dx23miny7, dy23minx7, b00, b10, b01, b11, c3out, dx31miny, dy31minx, dx31miny7, dy31minx7, c00, c10, c01, c11);

loopToplevel loopz(clk, reset, minxAligned, maxx, minyAligned, maxy, fdx12, fdx23, fdx31, fdy12, fdy23, fdy31, a00, a01, a10, a11, b00, b01, b10, b11, c00, c01, c10, c11, Ax, Ay, Bx, By, Cx, Cy, writePixel, fbAddress);
endmodule

module constHalfEdge(c1, dx12miny, dy12minx, dx12miny7, dy12minx7, a00, a10, a01, a11, c00, c10, c01, c11, dx23miny, dy23minx, dx23miny7, dy23minx7, b00, b10, b01, b11, c3out, dx31miny, dy31minx, dx31miny7, dy31minx7, c00, c10, c01, c11);
input [27:0] c1,c2,c3;
input [31:0] dx12miny,dy12minx,dx23miny,dy23minx,dx31miny,dy31minx;
input [31:0] dx12miny7,dy12minx7,dx23miny7,dy23minx7,dx31miny7,dy31minx7;
output [31:0] a00,a10,a01,a11,b00,b10,b01,b11,c00,c10,c01,c11;

const HalfEdgeUno aa(c1,dx12miny,dy12minx,dx12miny7,dy12minx7,a00,a10,a01,a11);
const HalfEdgeUno bb(c2,dx23miny,dy23minx,dx23miny7,dy23minx7,b00,b10,b01,b11);
const HalfEdgeUno cc(c3,dx31miny,dy31minx,dx31miny7,dy31minx7,c00,c10,c01,c11);

endmodule
module computeConstants(x1,x2,x3,y1,y2,y3,x1s,x2s,x3s,y1s,y2s,y3s,dx12,dx23,dx31,dy12,dy23,dy31,fdx12,fdx23,fdx31,fdy12,fdy23,fdy31);

input [9:0] x1,x2,x3; //input vertices
input [9:0] y1,y2,y3;
output [13:0] x1s,x2s,x3s,y1s,y2s,y3s;
output [13:0] dx12,dx23,dx31,dy12,dy23,dy31;
output [17:0] fdx12,fdx23,fdx31,fdy12,fdy23,fdy31; //fixed point representations

assign x1s=x1<<4;
assign x2s=x2<<4;
assign x3s=x3<<4;
assign y1s=y1<<4;
assign y2s=y2<<4;
assign y3s=y3<<4;

assign dx12=(x1s-x2s);
assign dx23=(x2s-x3s);
assign dx31=(x3s-x1s);
assign dy12=(y1s-y2s);
assign dy23=(y2s-y3s);
assign dy31=(y3s-y1s);

assign fdx12=dx12<<4;
assign fdx23=dx23<<4;
assign fdx31=dx31<<4;
assign fdy12=dy12<<4;
assign fdy23=dy23<<4;
assign fdy31=dy31<<4;

endmodule

module minmax3(x1s,x2s,x3s,y1s,y2s,y3s,xmin,ymin,xmax,ymax);
input [13:0] x1s,x2s,x3s,y1s,y2s,y3s;
output [13:0] xmin,ymin,xmax,ymax;

wire [2:0] sxmin,symin;
wire [13:0] xmintemp,xmaxtemp,ymintemp,xmaxtemp;

minmax2 xset1(x1s,x2s,minx1);
minmax2 xset2(x1s,x3s,minx2);
minmax2 xset3(x2s,x3s,minx3);

minmax2 yset1(y1s,y2s,ymin1);
minmax2 yset2(y1s,y3s,ymin2);
minmax2 yset3(y2s,y3s,ymin3);

mux8 xminselect(x3s,x2s,14'b0,x2s,x3s,14'b0,x1s,xls,xmintemp,sxmin);
mux8 xmaxselect(x3s,x2s,14'b0,x2s,x3s,14'b0,x1s,xls,xmaxtemp,~sxmin);

mux8 yminselect(y3s,y2s,14'b0,y2s,y3s,14'b0,yls,yls,ymintemp,symin);
mux8 ymaxselect(y3s,y2s,14'b0,y2s,y3s,14'b0,yls,yls,xmaxtemp,~symin);

endmodule
assign sxmin={minx1,minx2,minx3};
assign symin={ymin1,ymin2,ymin3};
assign xmin=(xmintemp+13'hF)>>4;
assign ymin=(ymintemp+13'hF)>>4;
assign xmax=(xmaxtemp+13'hF)>>4;
assign ymax=(ymaxtemp+13'hF)>>4;

endmodule

module minmax2(d1,d2,min);
  input [13:0] d1,d2;
  output min;
  assign min=(d1<d2);
endmodule

module blockAlign(minx,miny,q,minxAligned,minyAligned);
  input [13:0] q;
  input [13:0] minx,miny;
  output [13:0] minxAligned,minyAligned;
  assign minxAligned=minx&(~(q-1));
  assign minyAligned=miny&(~(q-1));
endmodule

module minmults(minxshift, minyshift, minx7shift, miny7shift, dxab, dyab, dxabminy,
  dyabminx, dxab7miny, dyab7minx);
  input [17:0] minxshift, minyshift, minx7shift, miny7shift;
  input [13:0] dxab, dyab;
  output [31:0] dxabminy, dyabminx, dxab7miny, dyab7minx;
  mult1317 dxabminymult(minyshift, dxab, dxabminy);
  mult1317 dyabminxmult(minxshift, dyab, dyabminx);
  mult1317 dxab7minymult(miny7shift, dxab, dxab7miny);
  mult1317 dyab7minxmult(minx7shift, dyab, dyab7minx);
endmodule

module qShiftPlus7(miny,minx,miny7,minx7, minyShift, minxShift);
  input [13:0] miny,minx;
  output [17:0] miny7,minx7,minyShift,minxShift;
  assign miny7=(miny+7)<<4;
  assign minx7=(minx+7)<<4;
  assign minyShift=miny<<4;
  assign minxShift=minx<<4;
endmodule

module preMults(dx12,dx23,dx31,dy12,dy23,dy31,minxShift,minyShift,minx7,miny7,dx12minx,
  dy12minx,dx12miny7,dy12miny7, dx23minx, dy23minx, dx23miny7, dy23miny7, dx31minx, dy31minx,
  dx31miny7, dy31miny7);
  input [13:0] dx12,dx23,dx31,dy12,dy23,dy31;
  input [17:0] minxShift,minyShift,minx7,miny7;
  output [31:0] dx12minx,dy12minx,dx12miny7,dy12miny7,dx23minx,dy23minx,dx23miny7,dy23miny7,
  dx31minx,dy31minx,dx31miny7,dy31miny7;
  minmults a12(minxShift, minyShift, minx7, miny7, dx12, dy12, dx12minx, dy12minx, dx12miny7, dy12miny7);
module initHalfEdge(x1s, x2s, x3s, y1s, y2s, y3s, dx12, dx23, dx31, dy12, dy23, dy31, c1, c2, c3);

input [13:0] x1s, x2s, x3s, y1s, y2s, y3s;
input [13:0] dx12, dx23, dx31, dy12, dy23, dy31;
output [27:0] c1, c2, c3;
wire [27:0] cx1, cy1, cx2, cy2, cx3, cy3;

mult1414s mulcx1(dy12, x1s, cx1);
mult1414s mulcy1(dx12, y1s, cy1);
mult1414s mulcx2(dy23, x2s, cx2);
mult1414s mulcy2(dx23, y2s, cy2);
mult1414s mulcx3(dy31, x3s, cx3);
mult1414s mulcy3(dx31, y3s, cy3);
assign c1 = cx1 - cy1;
assign c2 = cx2 - cy2;
assign c3 = cx3 - cy3;
endmodule

module constHalfEdgeUno(c1, dx12miny, dy12minx, dx12miny7, dy12minx7, a00, a10, a01, a11);

input [27:0] c1;
input [31:0] dx12miny, dy12minx;
input [31:0] dx12miny7, dy12minx7;
output [31:0] a00, a10, a01, a11;
assign a00 = {c1[27], c1[27], c1[27], c1[27], c1} + dx12miny - dy12minx;
assign a10 = {c1[27], c1[27], c1[27], c1[27], c1} + dx12miny - dy12minx7;
assign a01 = {c1[27], c1[27], c1[27], c1[27], c1} + dx12miny7 - dy12minx;
assign a11 = {c1[27], c1[27], c1[27], c1[27], c1} + dx12miny7 - dy12minx7;
endmodule

module halfEdgeIncrementUno(dx12, Ay);

input [13:0] dx12;
output [20:0] Ay;
assign Ay = dx12 << 7;
endmodule

module halfEdgeIncrement(dx12, dx23, dx31, dy12, dy23, dy31, Ay, By, Cy, Ax, Bx, Cx);

input [13:0] dx12, dx23, dx31, dy12, dy23, dy31;
output [20:0] Ay, By, Cy, Ax, Bx, Cx;
halfEdgeIncrementUno ay(dx12, Ay);
halfEdgeIncrementUno by(dx23, By);
halfEdgeIncrementUno cy(dx31, Cy);
halfEdgeIncrementUno ax(dy12, Ax);
halfEdgeIncrementUno bx(dy23, Bx);
halfEdgeIncrementUno cx(dy31,Cx);

module fillConvention(dy12,dx12,dy23,dx23,dy31,dx31,c1,c2,c3,c1out,c2out,c3out);
  input [13:0] dy12,dx12,dy23,dx23,dy31,dx31;
  input [27:0] c1,c2,c3;
  output [27:0] c1out,c2out,c3out;
  wire comp1;
  wire comp2;
  assign comp1 = (dy12[13]);
  assign comp2 = (dy12==0) & (~dx12[13] & (~dx12));
  assign c1out=c1+(comp1 | comp2);
  assign c2out=c2+((dy23[13]) | ((dy23==0) & (~dx23[13] & (~dx23))));
  assign c3out=c3+((dy31[13]) | ((dy31==0) & (~dx31[13] & (~dx31))));
endmodule

module blockLoop();
  //actual sequential logic goes in here
endmodule

module mux8(d0,d1,d2,d3,d4,d5,d6,d7,y,s);
  input [13:0] d0,d1,d2,d3,d4,d5,d6,d7;
  input [2:0] s;
  output [13:0] y;
  wire [13:0] low,high;
  mux4 lowmux(d0,d1,d2,d3,s[1:0],low);
  mux4 highmux(d4,d5,d6,d7,s[1:0],high);
  mux2 finalmux(low,high,s[2],y);
endmodule

module mux4(d0,d1,d2,d3,s,y);
  input [13:0] d0,d1,d2,d3;
  input [1:0] s;
  output [13:0] y;
  wire [13:0] low,high;
  mux2 lowmux(d0,d1,s[0],low);
  mux2 highmux(d2,d3,s[0],high);
  mux2 finalmux(low,high,s[1],y);
endmodule

module mux2(d0,d1,s,y);
  input [13:0] d0,d1;
  input s;
  output [13:0] y;
  assign y=s?d1:d0;
endmodule
module loopToplevel(clk, reset, minx, maxx, miny, maxy, fdx12, fdx23, fdx31, fdy12, fdy23, fdy31, A00, A01, A10, A11, B00, B01, B10, B11, C00, C01, C10, C11, Ax, Ay, Bx, By, Cx, Cy, writePixel, fbAddress);

input clk, reset;
input [13:0] minx, maxx, miny, maxy;
input [17:0] fdx12, fdx23, fdx31, fdy12, fdy23, fdy31;
input [31:0] A00, A01, A10, A11, B00, B01, B10, B11, C00, C01, C10, C11;
input [20:0] Ax, Ay, Bx, By, Cx, Cy;
output writePixel;
output [18:0] fbAddress;
wire [13:0] x, y, ix, iy;
wire [1:0] state;
wire [3:0] A, B, C;
wire [31:0] A00pass, B00pass, C00pass;
wire EMPTY;
wire [31:0] Cx1, Cx2, Cx3, Cy1, Cy2, Cy3;

looptop outsideLoop(clk, reset, x, y, minx, maxx, miny, maxy, iy, ix, state);
combllooptop CLBLK(clk, reset, x, ix, y, iy, maxx, maxy, A, B, C, Ax, Ay, A00, A01, A10, A11, Bx, By, B00, B01, B10, B11, Cx, Cy, C00, C01, C10, C11, A00pass, B00pass, C00pass);
drawLoop pixelLoop(clk, reset, x, y, ix, iy, state);

C123 updateCyCx(clk, reset, A00pass, B00pass, C00pass, Cx1, Cx2, Cx3, Cy1, Cy2, Cy3, fdx12, fdx23, fdx31, fdy12, fdy23, fdy31, x, y, ix, iy);
pixelOn drawPixel(Cx1, Cx2, Cx3, iy, ix, writePixel, fbAddress);

endmodule
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;
  */
end
end

else if((iy==y+7)&&(ix==x+7))

x<=x+8;

else

begin

y <= y;
x <= x;
state <= COUNTINGSTATE;
end

end

endmodule

module ABC //1,2,3, just you and me.
(x00x, x01x, x10x, x11x, X);

input [31:0] x00x, x01x, x10x, x11x;

output [3:0] X;

assign X = {(~x00x[31] & (|x00x)), (~x01x[31] & (|x01x)), (~x10x[31] & (|x10x)), (~x11x[31] & (|x11x))};

endmodule

module xloop(x00initial, x01initial, x10initial, x11initial, Xx, Xy, x00, x01, x10, x11, c:
reset, x, ix, y, iy, maxx);

input [31:0] x00initial, x01initial, x10initial, x11initial;
input [20:0] Xx, Xy;
output reg [31:0] x00, x01, x10, x11;
//output [31:0] Xxacc;
input clk, reset;
input [13:0] x, ix, y, iy, maxx;

always@(posedge clk or posedge reset)
begin

if (reset)

begin

//Xxacc <= 32'b0;
x00 <= x00initial;
x01 <= x01initial;
x10 <= x10initial;
x11 <= x11initial;
end

else if ((x == maxx)&&(ix == maxx)&&(iy == (y+7)))

begin

//Xxacc <= 32'b0;
x00 <= x00initial:// + {
{Xy[20]}, {Xy[20]}, {Xy[20]}, {Xy[20]}, {Xy[20]}, {Xy[20]}, {Xy[20]}, {Xy[20]},
y[20]}, Xy[20:0]);
x01 <= x01initial:// + {
{Xy[20]}, {Xy[20]}, {Xy[20]}, {Xy[20]}, {Xy[20]}, {Xy[20]}, {Xy[20]}, {Xy[20]},
y[20]}, Xy[20:0]);
x10 <= x10initial:// + {
{Xy[20]}, {Xy[20]}, {Xy[20]}, {Xy[20]}, {Xy[20]}, {Xy[20]}, {Xy[20]}, {Xy[20]},
y[20]}, Xy[20:0]);
x11 <= x11initial:// + {
{Xy[20]}, {Xy[20]}, {Xy[20]}, {Xy[20]}, {Xy[20]}, {Xy[20]}, {Xy[20]}, {Xy[20]},
y[20]}, Xy[20:0]);

end

else if ((ix == x+6)&&(iy == y+7))

begin

//Xxacc <= Xxacc + Xx;
}
module yloop(clk, reset, y, iy, x, ix, maxx, maxy, x00initial, x01initial, x10initial, x11initial, Xy, x00init, x01init, x10init, x11init);

    input clk, reset;
    input [13:0] x, y, ix, iy;
    input [20:0] Xy;
    input [13:0] maxx, maxy;
    input [31:0] x00initial, x01initial, x10initial, x11initial;
    output reg [31:0] x00init, x01init, x10init, x11init;

    always@ (posedge clk or posedge reset)
    begin
        if (reset)
            begin
                x00init <= x00initial;
                x01init <= x01initial;
                x10init <= x10initial;
                x11init <= x11initial;
            end
        else if (y == maxy)
            begin
                x00init <= x00initial;
                x01init <= x01initial;
                x10init <= x10initial;
                x11init <= x11initial;
            end
        else if ((x == maxx) && (ix == maxx) && (iy == (y+7)))
            begin
                x00init <= x00init + { {Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]}, Xy[20:0]};
                x01init <= x01init + { {Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]}, Xy[20:0]};
                x10init <= x10init + { {Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]}, Xy[20:0]};
                x11init <= x11init + { {Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]},{Xy[20]}, Xy[20:0]};
            end
    end
endmodule

module comblooptop(clk, reset, x, ix, y, iy, maxx, maxy, A, B, C, Ax, Ay, A00, A01, A10, A11, Bx, By, B00, B01, B10, B11, Cx, Cy, C00, C01, C10, C11, A00out, B00out, C00out);

    input clk, reset;
    input [13:0] x, y, ix, iy;
    input [13:0] maxx, maxy;
    output [3:0] A, B, C;
input [20:0] Ax, Ay, Bx, By, Cx,Cy;
input [31:0] A00, A01, A10, A11, B00, B01, B10, B11, C00, C01, C10, C11;
output [31:0] A00out, B00out, C00out;
wire [31:0] A00pass, A01pass, A10pass, A11pass, B00pass, B01pass, B10pass, B11pass, C00pass, C01pass, C10pass, C11pass;
wire [31:0] A00out, A01out, A10out, A11out, B00out, B01out, B10out, B11out, C00out, C01out, C10out, C11out;
yloop Ayloop(clk, reset, y, iy,x, ix,maxx, maxy, A00, A01, A10, A11,Ay, A00pass, A01pass, A10pass, A11pass);
xloop Axloop(A00pass, A01pass, A10pass, A11pass, Ax, Ay, A00out, A01out, A10out, A11out, clk, reset, x, ix,y,iy, maxx);
yloop Byloop(clk, reset, y, iy,x, ix,maxx, maxy, B00, B01, B10, B11,By, B00pass, B01pass, B10pass, B11pass);
xloop Bxloop(B00pass, B01pass, B10pass, B11pass, Bx, By, B00out, B01out, B10out, B11out, clk, reset, x, ix,y,iy, maxx);
yloop Cyloop(clk, reset, y, iy,x, ix,maxx, maxy, C00, C01, C10, C11,Cy, C00pass, C01pass, C10pass, C11pass);
xloop Cxloop(C00pass, C01pass, C10pass, C11pass, Cx, Cy, C00out, C01out, C10out, C11out, clk, reset, x, ix,y,iy, maxx);
ABC Agen(A00out, A01out, A10out, A11out, A);
ABC Bgen(B00out, B01out, B10out, B11out, B);
ABC Cgen(C00out, C01out, C10out, C11out, C);
endmodule

module C123(clk,reset,A00,B00,C00,Cx1,Cx2,Cx3,Cy1,Cy2,Cy3,fdx12,fdx23,fdx31,fdy12,fdy23,fdy31,x,y,ix,iy);

input [31:0] A00, B00, C00;
input [17:0] fdx12, fdx23, fdx31,fdy12, fdy23, fdy31;
output [31:0] Cx1, Cx2, Cx3,Cy1,Cy2,Cy3;
input [13:0] x,y,ix,iy;
input clk, reset;
//input [2:0] q;
reg [31:0] Cy1, Cy2, Cy3;
reg [31:0] Cx1, Cx2, Cx3;
reg [2:0] count;

always@(posedge clk or posedge reset)
begin
    if (reset)
        count <= 0;
    //else if (count == q)
    // count <= 0;
    else
        count <= count +1;
end

always@(posedge clk or posedge reset)
begin

if (reset)
begin
Cx1 <= A00;
Cx2 <= B00;
Cx3 <= C00;
end
else if (ix == x)
begin
Cx1 <= Cy1;// 
{fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17]:0};
Cx2 <= Cy2;// 
{fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17]:0};
Cx3 <= Cy3;// 
{fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17]:0};
end
else
begin
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],fdy12[17],fdy12[17],fdy12[17],fdy12[17],fdy12[17]:0};
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],fdy23[17],fdy23[17],fdy23[17],fdy23[17],fdy23[17]:0};
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],fdy31[17],fdy31[17],fdy31[17],fdy31[17],fdy31[17]:0};
end
end
always@(posedge clk or posedge reset)
if (reset)
begin
Cy1 <= A00;
Cy2 <= B00;
Cy3 <= C00;
end
else if ((ix == x +7) && (iy == y+7))
begin
Cy1 <= A00;
Cy2 <= B00;
Cy3 <= C00;
end
else if (count == 7)
begin
Cy1 <= Cy1 + {fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17],fdx12[17]:0};
Cy2 <= Cy2 + {fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17],fdx23[17]:0};
Cy3 <= Cy3 + {fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17],fdx31[17]:0};
end
endmodule
drawLoop(clk,reset,x,y,ix,iy,state);
module pixelOn(Cx1,Cx2,Cx3,iy,ix,writePixel,fbAddress);

    input [31:0] Cx1,Cx2,Cx3;
    input [13:0] ix,iy;
    output writePixel;
    output [18:0] fbAddress;

    wire [27:0] multOut;

    assign writePixel=((~Cx1[31] & (|Cx1)) & (~Cx2[31] & (|Cx2)) & (~Cx3[31] & (|Cx3)));
    mult1414u addressGenerate(iy,14'd640,multOut);
    assign fbAddress=ix+multOut[18:0];

endmodule
// This is the top level module of the 640x480 framebuffer
// The framebuffer runs on a 100 MHz clock
// When writing pixels/colors to the framebuffer, write should
// be synchronized to the 100 MHz clock.
// The relevant signals are color_data[3:0] (the pixel color)
// pixel_addr[18:0] the memory address to write
// =(640y +x), where x and y are the coordinates based off the upper left corner
// and wen, which is the write enable port for the memory;

// To modify this into a SVGA framebuffer, the relevant constants should be looked up in
// svga_defines.v (found in one of the built in examples files) and pasted into
// SVGA_TIMING_GENERATION.v
// Also, the block memory size will need to be increased, as well as the bus widths of the
// pixel_addr wires

module VGAFB_CTRL(
    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,
    color_data,
    pixel_addr,
    wen
);

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
input [3:0] color_data;
input [18:0] pixel_addr;
input wen;

wire VGA_OUT_PIXEL_CLOCK;
wire VGA_COMP_SYNCH_N;
wire VGA_OUT_BLANK_N;
wire VGA_HSYNCH;
wire VGA_VSYNCH;
wire [7:0] VGA_OUT_RED;
wire [7:0] VGA_OUT_GREEN;
wire [7:0] VGA_OUT_BLUE;
wire CLOCK;
wire clock;
wire clk_100;
wire clock_100;
wire clk_25;
wire clock_25;
wire clk_40;
wire clock_40;
wire system_dcm_rst;
wire system_dcm_locked;
wire reset = !system_dcm_locked;
wire pixel_clock = clk_25;

wire low = 1'b0;
wire high = 1'b1;

wire color_write_clock = clk_100;
wire color_write_enable = high;
wire [18:0] data_addr = pixel_addr;
wire [3:0] color_data_in;
assign color_data_in = color_data;

// instantiate the VGA controller and display RAM
VGAFB theFB(
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,
data_addr,
color_data_in,
color_write_clock,
en,
pixel_clock,
reset
);

// instantiate the clock input buffers for the single ended clocks
//IBUFG CLOCK_INPUT_BUF (  
//.O (clock),  
//.I (CLOCK)  
//);

// instantiate the clock input buffers for the internal clocks
BUFG CLK_100MHZ_BUF (  
.O (clk_100),  
.I (clock_100)  
);

BUFG CLK_25MHZ_BUF (  
.O (clk_25),  
.I (clock_25)  
);
BUFG CLK_40MHZ_BUF (
.O (clk_40),
.I (clock_40)
);

// instantiate the DCMs and DCM reset generation
DCM SYSTEM_DCM (
.CLKFB (clk_100),
.CLKIN (CLOCK),
.DSSEN (low),
.PSCLK (low),
.PSEN (low),
.PSINCDEC (low),
.RST (system_dcm_rst),
.CLK0 (clock_100),
.CLK90 (),
.CLK180 (),
.CLK270 (),
.CLK2X (),
.CLK2X180 (),
.CLKDV (clock_25),
.CLKFX (clock_40),
.CLKFX180 (),
.LOCKED (system_dcm_locked),
.PSDONE (),
.STATUS ()
) /* 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"*/

SRL16 RESET_SYSTEM_DCM (
.Q (system_dcm_rst),
.CLK (CLOCK),
.D (low),
.A0 (high),
.A1 (high),
.A2 (high),
.A3 (high)
) /*synthesis xc_props="INIT = 000F"*/;

endmodule
module VGAFB(
  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,
  color_data_addr,
  color_data_in,
  color_data_addr,
  color_data_in,
  color_data_clock,
  wen,
  pixel_clock,
  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

input [18:0] color_data_addr;
input [3:0] color_data_in;
input color_data_clock;
input pixel_clock;
input reset;
input wen;

wire [3:0] ram_data_out;
wire [18:0]ram_data_addr;
wire [3:0] ram_data_in = 8'b0; //never going to write data using second port
wire [3:0] color_data_out; //should never use this either.
wire [3:0] VGA_COLOR_PRIMITIVE;

wire [9:0] line_num;
wire [10:0] pixel_num;
wire high = 1'b1;
wire low = 1'b0;
module COLOR_PIPELINE (color_ram_data, pixel_clock, reset, aligned_color_data);

   fbmem FBRAM(
      color_data_addr,
      ram_data_addr,
      color_data_clock,
      pixel_clock,
      color_data_in,
      ram_data_in,
      color_data_out,
      ram_data_out,
      high, //both ports are always on.
      high,
      wen, //port A is write only
      low //port B is read only
   );

   COLOR_PIPELINE PIPE(
      ram_data_out,
      pixel_clock,
      reset,
      VGA_COLOR_PRIMITIVE
   );

   CLUT COLORLUT(
      VGA_COLOR_PRIMITIVE,
      high,
      pixel_clock,
      reset,
      VGA_OUT_RED,
      VGA_OUT_GREEN,
      VGA_OUT_BLUE
   );

   // instantiate the SVGA timing generator
   SVGA_TIMING_GENERATION SVGA_TIMING_GENERATION(
      pixel_clock,
      reset,
      h_synch_delay,
      v_synch_delay,
      comp_synch,
      blank,
      line_num,
      pixel_num,
      ram_data_addr
   );

   assign VGA_VSYNCH = v_synch_delay;
   assign VGA_HSYNCH = h_synch_delay;
   assign VGA_COMP_SYNCH_N = 1'b0; //disable comp_synch; or sync on green;
   assign VGA_OUT_BLANK_N = ~blank;
   assign VGA_OUT_PIXEL_CLOCK = pixel_clock;

endmodule
input [3:0] color_ram_data; // the color of the character
input pixel_clock; // pixel clock
input reset; // reset
output[3:0] aligned_color_data; // time aligned color data

reg [3:0] aligned_color_data;
reg [3:0] color_pipe0;
reg [3:0] color_pipe1;

always @(posedge pixel_clock or posedge reset) begin
  if (reset) begin
    color_pipe0 <= 19'h0;
    color_pipe1 <= 19'h0;
    aligned_color_data <= 19'h0;
  end
  else begin
    color_pipe0 <= color_ram_data;
    color_pipe1 <= color_pipe0;
    aligned_color_data <= color_pipe1;
  end
end
endmodule //COLOR_PIPE

/*
module SRL16(Q, A0, A1, A2, A3, CLK, D); // synthesis syn_black_box
output Q;
input A0;
input A1;
input A2;
input A3;
input CLK;
input D;
endmodule

module IBUFG(O, I); // synthesis syn_black_box
output O;
input I;
endmodule

module BUFG(O, I); // synthesis syn_black_box
output O;
input I;
endmodule

module DCM(CLKFB, CLKIN, DSSEN, PSCLK, PSEN, PSINCDEC, RST,
  CLK0, CLK90, CLK180, CLK270, CLK2X, CLK2X180, CLKDV, CLKFX, CLKFX180, LOCKED, PSDONE, STATUS); // synthesis syn_black_box
input CLKFB, CLKIN, DSSEN;
input PSCLK, PSEN, PSINCDEC, RST;
output CLK0, CLK90, CLK180, CLK270, CLK2X, CLK2X180;
output CLKDV, CLKFX, CLKFX180, LOCKED, PSDONE;
output [7:0] STATUS;
endmodule
*/
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 character 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
module SVGA_TIMING_GENERATION (pixel_clock, reset, h_synch_delay, v_synch_delay, comp_synch, blank, line_count, pixel_count, pixel_addr);

input pixel_clock; // pixel clock
input reset; // reset
output h_synch_delay; // horizontal synch for VGA connector
output v_synch_delay; // vertical synch for VGA connector
output comp_synch; // composite synch for DAC
output blank; // composite blanking
output [9:0] line_count; // line counter for current pixel
output [10:0] pixel_count; // column counter for current pixel
output [18:0] pixel_addr; // character mode address

reg [9:0] line_count; // counts the display lines
reg [10:0] pixel_count; // counts the pixels in a line
reg [18:0] pixel_addr;

reg h_synch; // horizontal synch
reg v_synch; // vertical synch
reg h_synch_delay; // h_synch delayed 2 clocks to line up with DAC pipeline
reg v_synch_delay; // v_synch delayed 2 clocks to line up with DAC pipeline
reg h_synch_delay0; // h_synch delayed 1 clock
reg v_synch_delay0; // v_synch delayed 1 clock

reg h_c_synch; // horizontal component of comp synch
reg v_c_synch; // vertical component of comp synch
reg comp_synch; // composite synch for DAC
reg h_blank; // horizontal blanking
reg v_blank; // vertical blanking
reg blank; // composite blanking

// CREATE THE HORIZONTAL LINE PIXEL COUNTER
always @ (posedge pixel_clock or posedge reset) begin
  if (reset) begin // on reset set pixel counter to 0
    pixel_count <= 11'h000;
  end
  else if (pixel_count == (`H_TOTAL - 1))
    begin // last pixel in the line
      pixel_count <= 11'h000; // reset pixel counter
    end
  else begin
    pixel_count <= pixel_count +1;
  end
end
// CREATE THE HORIZONTAL SYNCH PULSE
always @ (posedge pixel_clock or posedge reset) begin
  if (reset)
    begin // on reset
      h_synch <= 1'b0; // remove h_synch
    end

  else if (pixel_count == (`H_ACTIVE + `H_FRONT_PORCH -1))
    begin // start of h_synch
      h_synch <= 1'b1;
    end

  else if (pixel_count == (`H_TOTAL - `H_BACK_PORCH -1))
    begin // end of h_synch
      h_synch <= 1'b0;
    end
end

// CREATE THE VERTICAL FRAME LINE COUNTER
always @ (posedge pixel_clock or posedge reset) begin
  if (reset)
    begin // on reset set line counter to 0
      line_count <= 10'h000;
    end

  else if ((line_count == (`V_TOTAL - 1))&& (pixel_count == (`H_TOTAL - 1)))
    begin // last pixel in last line of frame
      line_count <= 10'h000; // reset line counter
    end

  else if ((pixel_count == (`H_TOTAL - 1)))
    begin // last pixel but not last line
      line_count <= line_count + 1; // increment line counter
    end
end

// CREATE THE VERTICAL SYNCH PULSE
always @ (posedge pixel_clock or posedge reset) begin
  if (reset)
    begin // on reset
      v_synch = 1'b0; // remove v_synch
    end

  else if ((line_count == (`V_ACTIVE + `V_FRONT_PORCH -1) &&
            (pixel_count == (`H_TOTAL - 1))))
    begin // start of v_synch
      v_synch = 1'b1;
    end

  else if ((line_count == (`V_TOTAL - `V_BACK_PORCH - 1)) &&
            (pixel_count == (`H_TOTAL - 1)))
    begin // end of v_synch
      v_synch = 1'b0;
    end
end

// ADD TWO PIPELINE DELAYS TO THE SYNCHS COMPENSATE FOR THE DAC PIPELINE DELAY
always @ (posedge pixel_clock or posedge reset) begin
  if (reset)
    begin
      h_synch_delay0 <= 1'b0;
      v_synch_delay0 <= 1'b0;
      h_synch_delay <= 1'b0;
      v_synch_delay <= 1'b0;
    end
end
else
begin
  h_synch_delay0 <= h_synch;
  v_synch_delay0 <= v_synch;
  h_synch_delay <= h_synch_delay0;
  v_synch_delay <= v_synch_delay0;
end
end

// CREATE THE HORIZONTAL BLANKING SIGNAL
// the "-2" is used instead of "-1" because of the extra register delay
// for the composite blanking signal
always @(posedge pixel_clock or posedge reset) begin
  if (reset)
  begin // on reset
    h_blank <= 1'b0; // remove the h_blank
  end
  else if (pixel_count == (`H_ACTIVE -2))
  begin // start of HBI
    h_blank <= 1'b1;
  end
  else if (pixel_count == (`H_TOTAL -2))
  begin // end of HBI
    h_blank <= 1'b0;
  end
end

// CREATE THE VERTICAL BLANKING SIGNAL
// the "-2" is used instead of "-1" in the horizontal factor because of the extra
// register delay for the composite blanking signal
always @(posedge pixel_clock or posedge reset) begin
  if (reset)
  begin // on reset
    v_blank <= 1'b0; // remove v_blank
  end
  else if (line_count == (`V_ACTIVE - 1) &&
          (pixel_count == `H_TOTAL - 2))
    begin // start of VBI
      v_blank <= 1'b1;
    end
  else if (line_count == (`V_TOTAL - 1)) &&
          (pixel_count == (`H_TOTAL - 2))
    begin // end of VBI
      v_blank <= 1'b0;
    end
end

// CREATE THE COMPOSITE BANKING SIGNAL
always @(posedge pixel_clock or posedge reset) begin
  if (reset)
  begin // on reset
    blank <= 1'b0; // remove blank
  end
  else if (h_blank || v_blank) // blank during HBI or VBI
begin
  blank <= 1'b1;
end

else begin
  blank <= 1'b0; // active video do not blank
end
end

// CREATE THE HORIZONTAL COMPONENT OF COMP SYNCH
// the "-2" is used instead of "-1" because of the extra register delay
// for the composite synch
always @ (posedge pixel_clock or posedge reset) begin
  if (reset)
    begin // on reset
      h_c_synch <= 1'b0; // remove h_c_synch
    end
  else if (pixel_count == (`H_ACTIVE + `H_FRONT_PORCH -2))
    begin // start of h_c_synch
      h_c_synch <= 1'b1;
    end
  else if (pixel_count == (`H_TOTAL - `H_BACK_PORCH -2))
    begin // end of h_c_synch
      h_c_synch <= 1'b0;
    end
end

// CREATE THE VERTICAL COMPONENT OF COMP SYNCH
always @ (posedge pixel_clock or posedge reset) begin
  if (reset)
    begin // on reset
      v_c_synch <= 1'b0; // remove v_c_synch
    end
  else if ((line_count == (`V_ACTIVE + `V_FRONT_PORCH - 1) &&
            (pixel_count == `H_TOTAL - 2)))
    begin // start of v_c_synch
      v_c_synch <= 1'b1;
    end
  else if ((line_count == (`V_TOTAL - `V_BACK_PORCH - 1)) &&
            (pixel_count == (`H_TOTAL - 2)))
    begin // end of v_c_synch
      v_c_synch <= 1'b0;
    end
end

// CREATE THE COMPOSITE SYNCH SIGNAL
always @ (posedge pixel_clock or posedge reset) begin
  if (reset)
    begin // on reset
      comp_synch <= 1'b0; // remove comp_synch
    end
  else begin
    comp_synch <= (v_c_synch ^ h_c_synch);
  end
end

//CREATE THE PIXEL ADDRESS COUNTER
always @(posedge pixel_clock or posedge reset)
begin
  if (reset)
    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     end
335
336
337
338
339
340
341   endmodule  //SVGA_TIMING_GENERATION
342
343
344
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; // 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)
beg
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);

FB_test 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'b1011,
  FBaddress,
  wen
);

endmodule
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;

//generate the 1 cycle start signal
always@(posedge CLOCK or posedge reset)
beg

if (reset)
    start <= 0;
else if (start == 2'b0)
    start <= 2'b1;
else if (start == 2'b1)
    start <= 2'b10;
end

mult multiply1(14'd6, 18'd6, multout1);
mult multiply2(14'd9, 18'd9, multout2);

//pass everything into the lie drawing hardware, pixel coordinates, start, reset, etc.
drawline linegpu(14'b1, 14'b1, multout2[13:0],multout1[13:0], start[0], reset, CLOCK,
    FBaddress, 19'd641, wen);

FB_test 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'b1011,
FBaddress,
wren
);

endmodule
How to draw a line:
this module requires:
  x1, y1, x2, y2: the vertices of the line
  start: a 1 cycle pulse to begin drawing the line
  reset: pulse high to load the x1, y1, x2, y2 values into assorted counters
  note that this means your vertex values must be set before hitting reset
  addrinitial: the framebuffer address of the first pixel of the line;
outputs:
  fbAddress: the current pixel address
  wen: write enable signal for the framebuffer
module drawline(
    x1, y1, x2, y2, start, reset, clock, fbAddress, addrinitial, wen
);
assign dy2comp = ~dy + 1;

mux2(dx, dx2comp, dx[12], dxout);
mux2(dy, dy2comp, dy[13], dyout);

dxdy = dxout - dyout;

assign steep = dxdy[13];
/*

//assign steep = abs(dy) > abs(dx);
wire [13:0] absdy, absdx;
wire [13:0] notdx, notdy;

//assign absdx = {1'b0, ~dx[13], dx[12:0]};
//assign absdy = {1'b0, ~dy[13], dy[12:0]};
assign notdx = (~dx + 1'b1);
assign notdy = (~dy + 1'b1);
muxtwo absdxmux(dx, notdx, dx[13], absdx);
muxtwo absdymux(dy, notdy, dy[13], absdy);

always @( *)
begin
    if ({1'b0, absdy} > {1'b0, absdx})
        steep = 1'b1;
    else
        steep = 1'b0;
end

//assign ys1 = y1;
//assign ys2 = x1;
//assign xs1 = x1;
//assign xs2 = x2;
assign xs1 = steep ? y1 : x1;
assign ys1 = steep ? x1 : y1;
assign xs2 = steep ? y2 : x2;
assign ys2 = steep ? x2 : y2;

assign leftright = {1'b0, xs1} > {1'b0, x2};

assign xstart = leftright ? xs2 : xs1;
assign xend = leftright ? xs1 : xs2;
muxtwo ysmux1(ys1, ys2, leftright, ystart);
muxtwo ysmux2(ys2, ys1, leftright, yend);

assign dxx = xend - xstart;
assign dyy = yend - ystart;

always @( *)
begin
    if ({1'b0, ystart} < {1'b0, yend})
        ystep = 1;
    else
        ystep = -1;
end

//clock enable/disable
//need to fix this;

assign xenable = start | clken;
always@(posedge clock or posedge reset)
begin
  if (reset)
    clken <= 1;
  else if (({1'b0, x} == {1'b0, xend}))
    clken <= 0;
end

//the x value counter
always@(posedge clock or posedge reset)
begin
  if (reset)
    x <= xstart;
  xaddr <= addrinitial;
  else if (({1'b0, x} == {1'b0, xend}))
    x <= xstart;
  xaddr <= addrinitial;
  else if (xenable)
    x <= x+1;
  xaddr <= xaddr+14'd640;
end

//the error calculation;
wire [13:0] errstart;
assign errstart = dx>>1;

always@(posedge clock or posedge reset)
begin
  if (reset)
    begin
      err <= errstart;
      y <= ystart;
      yaddr <= addrinitial;
    end
  else if (({1'b0, x} == {1'b0, xend}))
    begin
      y <= ystart;
      yaddr <= addrinitial;
    end
  else if (xenable && (err[13] == 1'b0))
    begin
      err <= err - dyy;
    end
  else if (xenable && (err[13] == 1'b1))
    begin
      y <= y+ystep;
      yaddr <= yaddr + 14'd640;
      err <= err + dxx - dyy;
    end
end

//generate output address: 1 port memory
//assign addr1 = x*640+y;
//assign addr2 = y*640+x;
wire [18:0] addra;
wire [18:0] xhat, yhat;
assign xhat = steep ? xaddr : yaddr;
assign yhat = steep ? y : x;
assign addra = xhat;
// mult multiply({4'b0000, xhat[13:4]}, 14'd640, addra);
// assign addra = {4'b0000, xhat[13:4]}* 14'd640;
assign fbAddress = addra + yhat;

// the enable signal for the fbAddress;
assign wen = xenable;

dendmodule

module muxtwo(d0, d1, s, dout);
input [13:0] d0, d1;
input s;
output [13:0] dout;
assign dout = s ? d1 : d0;
dendmodule

module bigmuxtwo(d0, d1, s, dout);
input [18:0] d0, d1;
input s;
output [18:0] dout;
assign dout = s? d1:d0;
dendmodule