HERA: 360° Image Warping Surveillance System

Final Report

Team Members
Bryan Gwin [bjg2119@columbia.edu]
David Lariviere [dal2103@columbia.edu]
DJ Park [dp2067@columbia.edu]
Michael Verbalis [mjv2008@columbia.edu]

Table of Contents

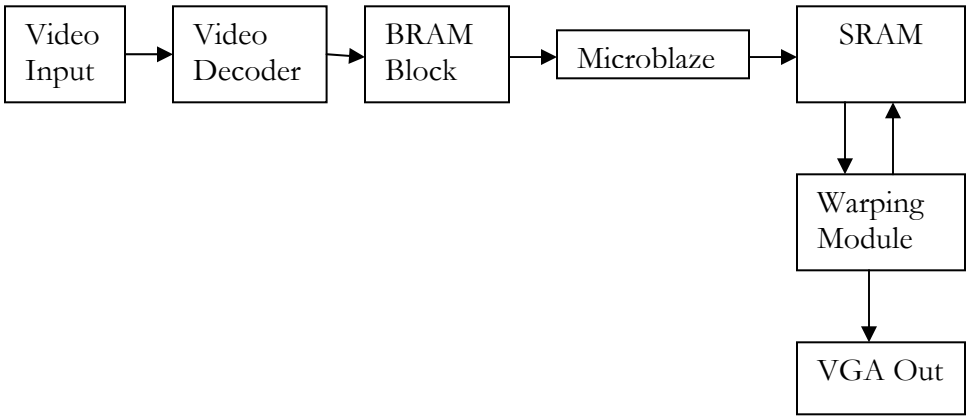
1.0 Introduction

1.1 System Overview

The goal of this project is to take a real-time video stream containing a 360° image into the video input of the XESS XSB-300E board and to warp a portion of it in order to display a properly scaled image onto a monitor.

1.2 Block Diagram

The following block diagram describes the current flow of data in this system:



2.0 Components

2.1 Camera System

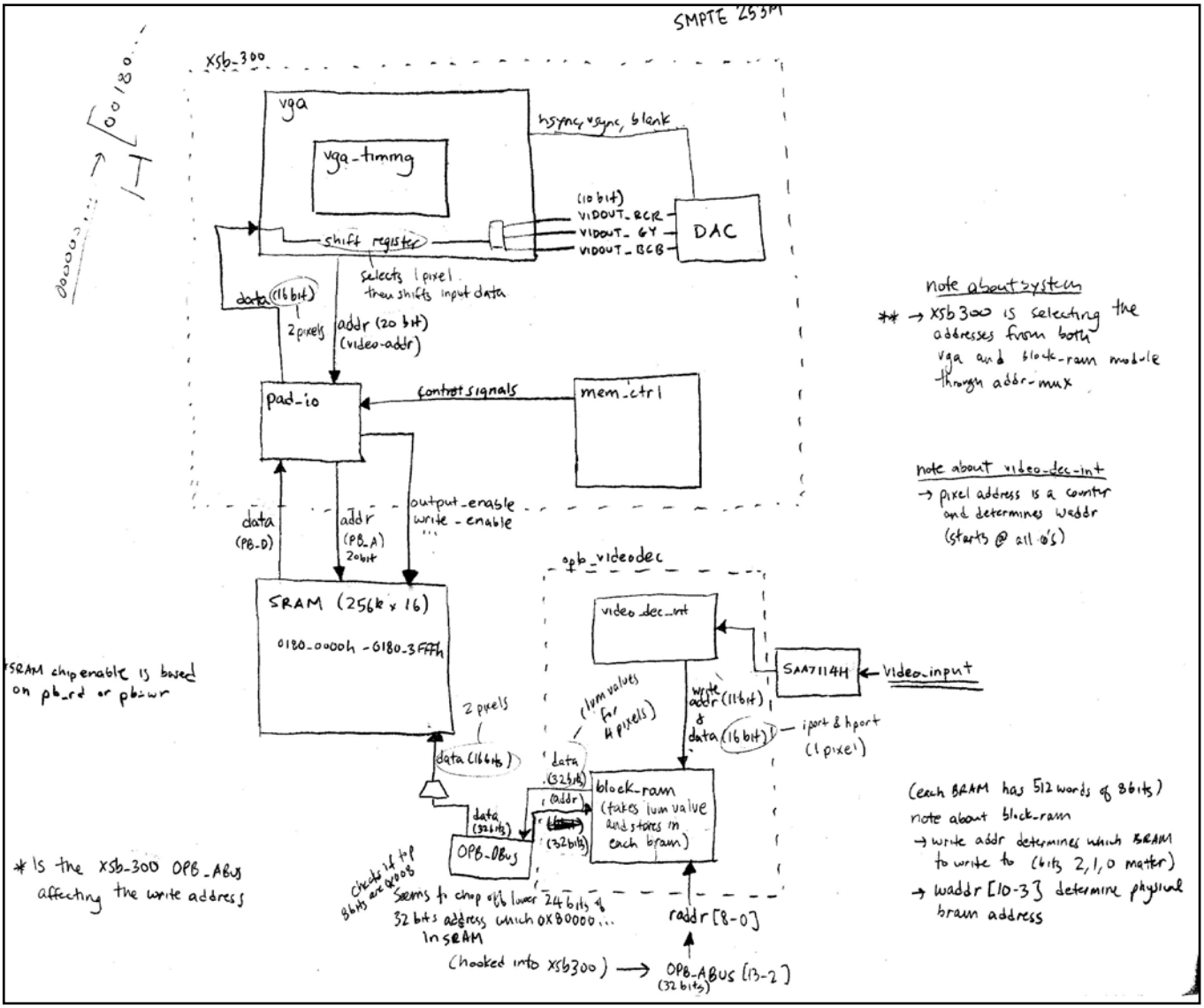
In order to capture the initial 360° video stream, we will be using a standard parabolic 360° lens mounted onto a PC165C Color ExView CMount Video Camera. This camera is in the family of high performance security cameras and has extremely sharp resolution (480 x 480) for a camera its size.

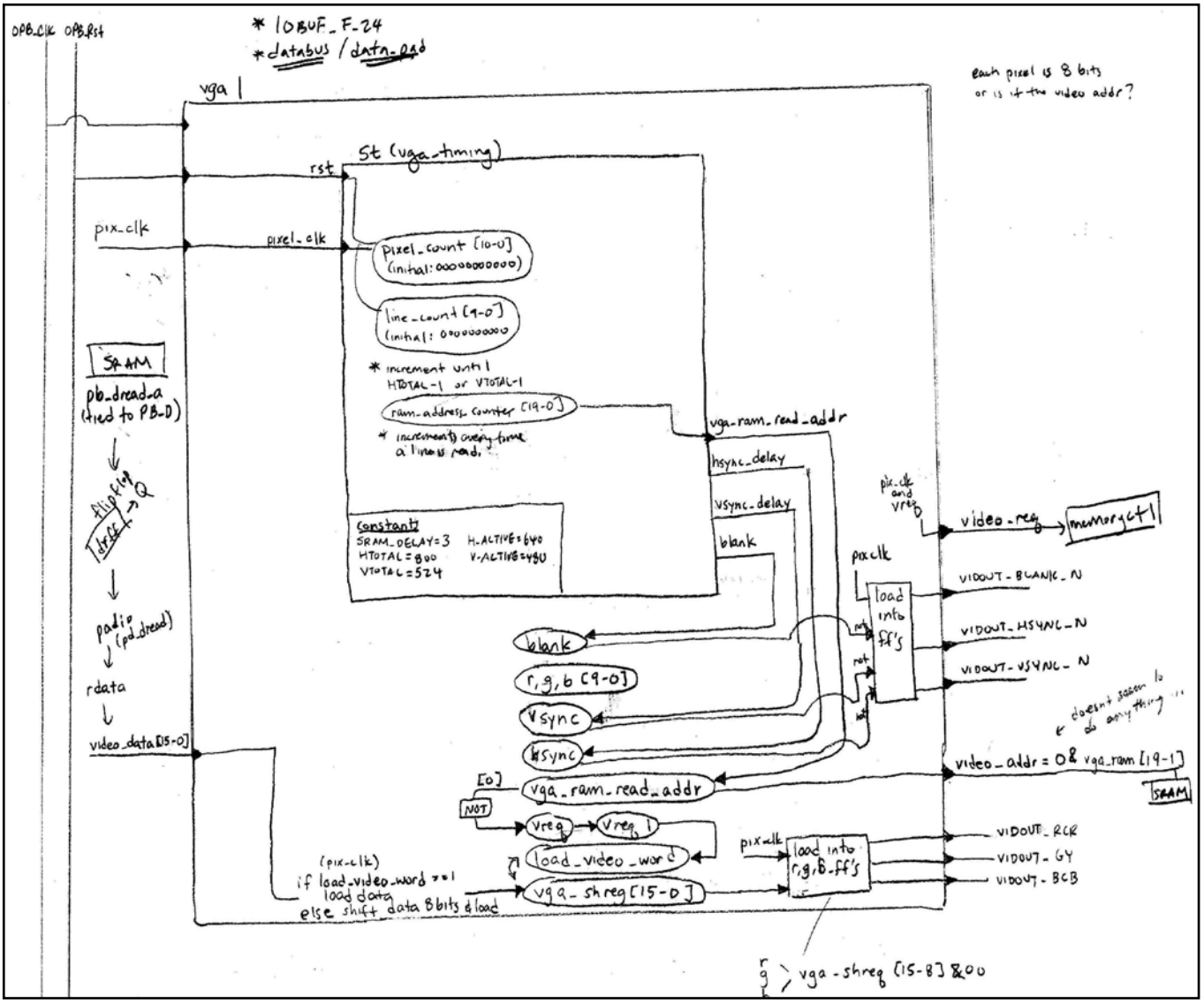
2.2 Video System

2.2.1 Overview

Video is captured by a video input module that digitizes each pixel of the input video stream into an 8 bit luminance signal and stores it in the SRAM (16 bits at a time). A video output module generates the appropriate memory address in the SRAM, fetches this data, converts it into data suitable for the DAC, and then it is processed into output.

2.2.2 *Block Diagram*





2.2.3 Memory Details

2.2.4

BRAM

We will be using the on-chip, dual-ported BRAMs on the FPGA. Each of these BRAMs have adjustable widths for each port and are .5K in size. The following chart summarizes the use of BRAMs in this system:

Component	# of BRAMs used	Size of BRAMs
Video Decoder	4	512 words x 8 bits
Warper	3	256 words x 16 bits

The Video Decoder component uses the BRAMs to buffer the image data before sending it off to the SRAM. The Warper component uses the BRAMs to store the warping parameters that are generated by software.

SRAM

We will be using the Toshiba TC55V16256J SRAM, which features 512Kb (256K x 16bits) of memory, to store each frame of video.

2.2.5 Video Input

We will be using the on-board Phillips SAA7114-H video decoder chip to handle the analog video signal obtained from the camera. The chip is capable of digitizing any NTSC, SECAM, or PAL video signal and scaling it to an appropriate frame size. The chip also allows for precise control over the luminance, chrominance, hue, saturation, etc. of the video through numerous controls.

Each pixel of the input video is digitized into an 8-bit luminance value and sent to a block ram module that takes care of merging the pixel data together so that it can be sent to the SRAM. This system uses a software module to perform actual transfer of the pixel data from the BRAM to the SRAM by passing the information through the OPB Bus. Currently, the SRAM stores 16 bits of pixel data (2 pixels) at each memory address.

2.2.6 Video Output

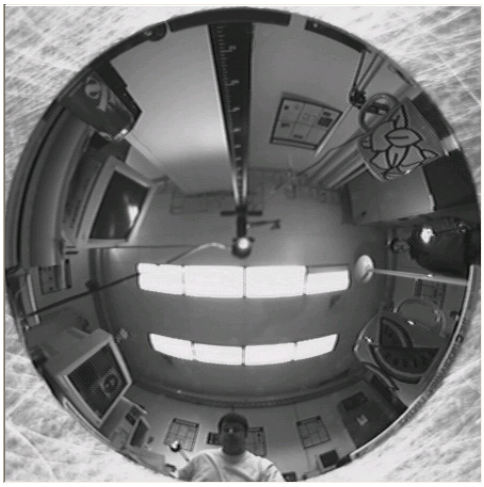
We will be using the on-board Texas Instruments THS8133B Digital-To-Analog Converter that features a triple 10-bit Digital to Analog Converter (DAC) to translate and deliver the desired image to a VGA display. The warping module (explained in **section 2.3**) identifies the pixels and their corresponding memory addresses in SRAM that are needed to generate the properly warped image. Once the proper pixel data is grabbed from the SRAM, this information is then converted into a data format recognizable by the DAC and then processed into output.

2.3 Warping Module

2.3.1 Overview

The input signal consists of a video stream from a 360 degree catadioptric lens (see image on the next page). The purpose of warping is to take a portion of the 360 degree image and unwarp it into a standard perspective-view image representing the image that would have been captured from a standard camera taken in the corresponding direction.

The computational cost of mapping input pixels to corresponding output pixels exactly is rather large and is therefore not possible in real-time. However, a series of simplifications and approximations were used to make it feasible.



360° Input Image

Unwarped Perspective View

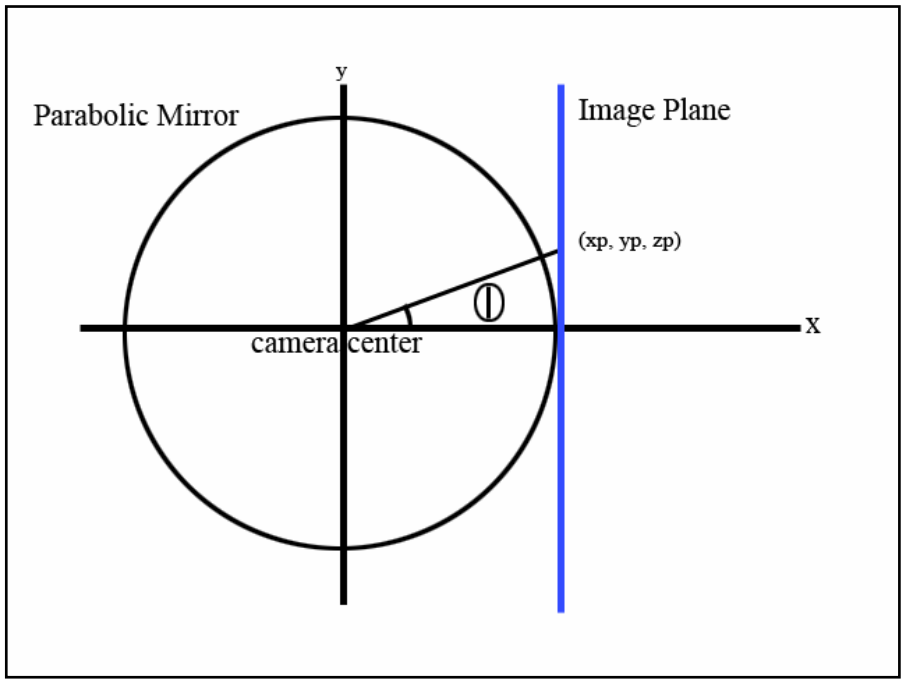
Please see Appendix for an example C program that generates unwarped images.

2.3.2 *Theory*

2.3.3

The original formulation was in polar coordinates utilizing transcendental functions:

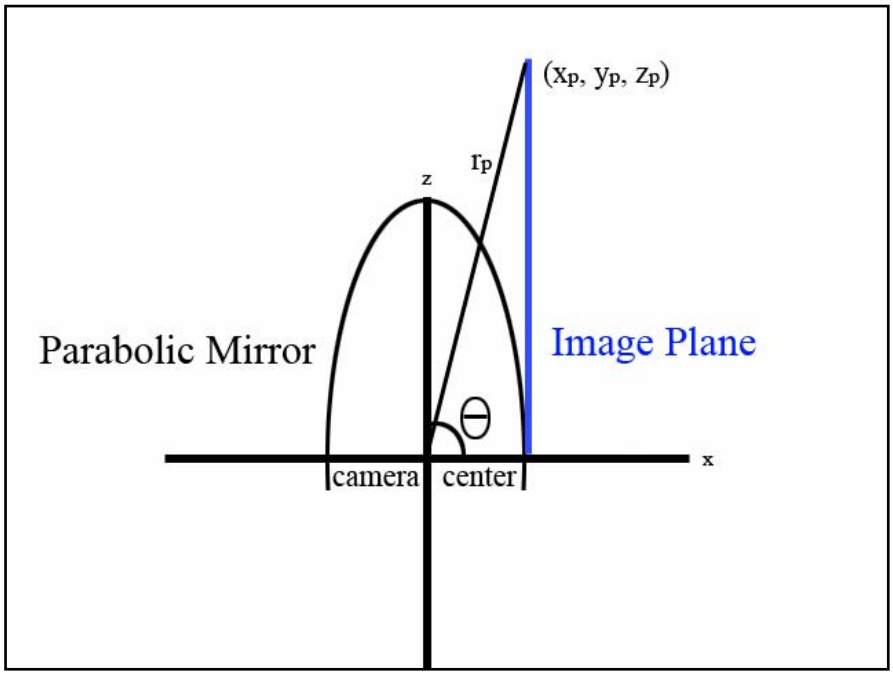
Consider a set of coplanar points in 3D space.
Given a specific point representing a on the image plane, (xp, yp, zp): Trace a ray between the optical center of the camera and the point to be displayed. Calculate the intersection of the ray with the paraboloid that defines the mirror, and its projection onto the (x,y) plane which yields the corresponding input pixel.



Given:
$\Theta = \cos\text{-}1((zp) / \text{sqrt}(xp2 + yp2 + zp2))$
$\Phi = \tan\text{-}1(yp/xp)$

$X_i = p * \sin \Theta * \cos \varphi$
$Y_i = p * \sin \Theta * \sin \varphi$

By restricting possible camera planes to a single frame that is orthogonal to the (x,y) plane and sitting atop the point (r,0,0), it is possible to derive a clean euclidian formulation:
$X = (h * xp) / (rp + zp)$
$Y = (h * yp) / (rp + zp)$.



By parameterizing the arc traced out upon the (x,y) plane in terms of a rotation about a point on the plane, for each scanline, it becomes possible to iteratively calculate every point on a scanline, provided only with the first point on the scanline and the angle of increment.

To calculate this parameterization, simply calculate the first and last points along a scanline, and then the angle between them. Lastly divide the angle by the number of pixels per scanline, which results in the "angle of increment." This is the amount Θ, by which one must rotate about the center of the particular circle.

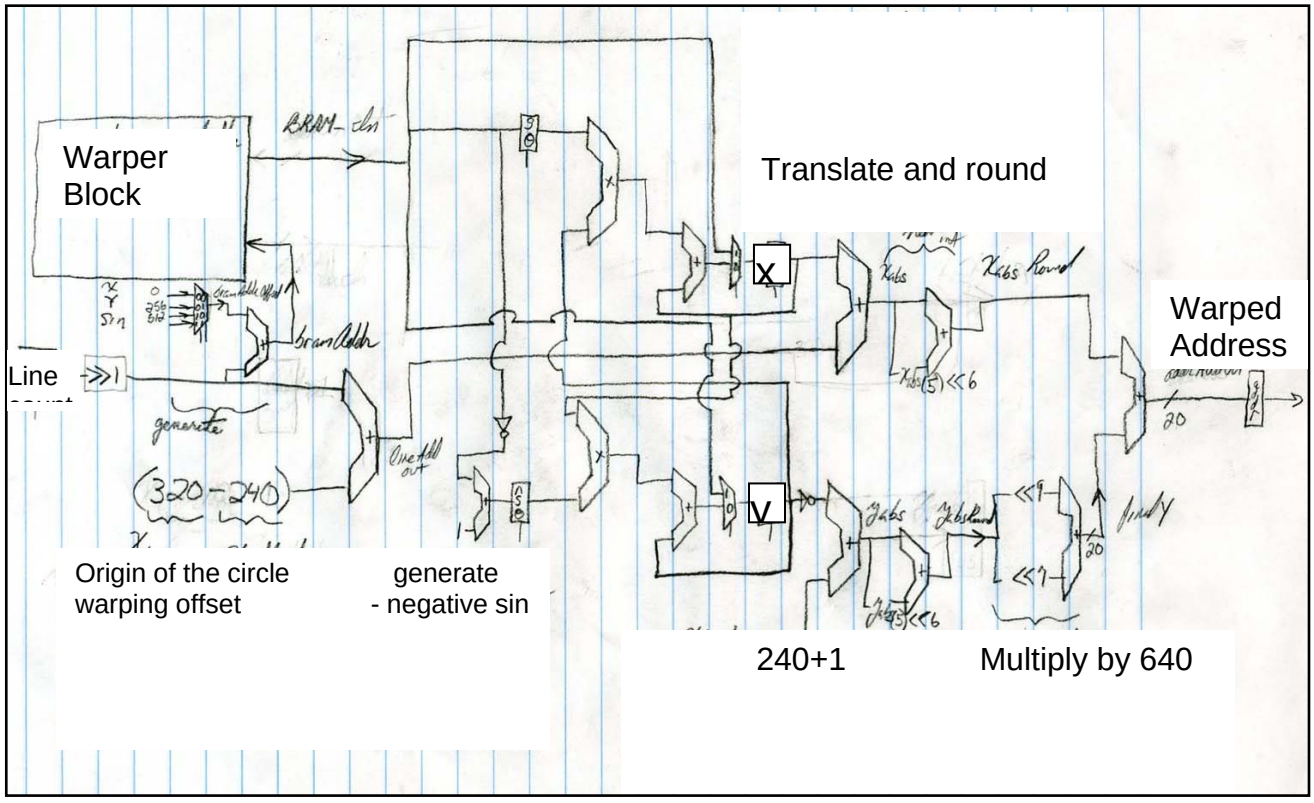
$Xi = x \cos \Theta + y \sin \Theta$
$Yi = - x \sin \Theta + y \cos \Theta$

Finally, through a first order linear approximation, we can accurately derive:
$X = x + y * \sin \Theta$
$Y = x * (-\sin \Theta) + y$

Note that this formulation has one additional benefit as well. While the initial points calculated are for a view centered about (r,0,0) one can, with great ease, generate any viewpoint tangential to the circle. To do this, simply rotate the initial starting point for every scanline by some angle to the desired starting position.

These formulas were tested in software. (see appendix for the code we based our software simulations on).

2.3.4 Block Diagram



2.3.5 VHDL Design

Block Diagram description

Stage 1 consists of the RAM and addressing controls for the loading sequence that loads the initial x/y data points and the sin(theta) and -sin(theta) values needed.

Stage 2 is the multipliers and adders that form the critical path to generates the next x-y coordinates (using the formulas from theory). This path is critical because it must be able to keep up with the pixel clock. Each multiplication (and addition) must take place in 4 clock cycles (or 2 pixel clock cycles; 2 because reading from the SRAM is actually the rate determining step at 3 clock cycles, so we have to repeat each pixel once anyway). From here the x/y values are returned the the input of stage 2 and also sent to stage 3.

Stage 3 consists of a nmuber of adders and a register to hold the final address value. In stage 3a the x/y values are translated to the coordinate system of the output VGA. In stage 3b, they are rounded to the nearest pixel and converted from fixed point to simple integers. Finally, in stage 3c the y coordinate is multiplied by 640 (a constant, so just shift and add) and added to the x coordinate to get the final address necessary.

State Machine

We used a state machine to control the timing between the stages. Initially, before any pixel-data is needed (during the front porch), the initial x/y/sin values are loaded one at a time from a BRAM using an address generated from the linecount (divided by 2) and an offset based on which (x/y/sin) value is being loaded. The negative sin value is calculated from sin (in 2's complement form) and loaded at the same time. The first address is then loaded once the initial x/y values have propagated through stage 3.

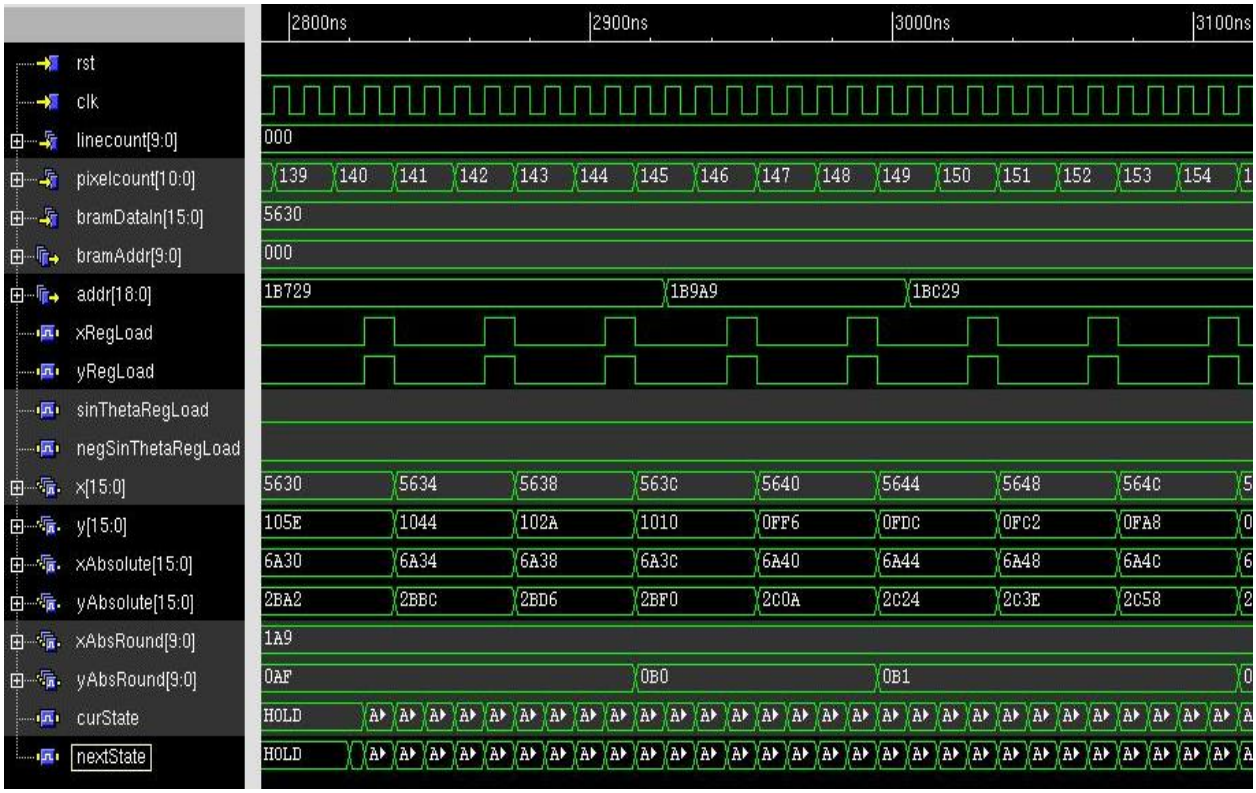
The state machine then sits in a holding state until just before the pixel data is needed (as determined by the pixel-count input from the VGA-module). At that point it cycles through four states, loading the next x/y values from stage 2 once every fourth clock cycle, and loading the address register in the state (clock cycle) immediately after.
Finally, the loading cycles are exited when the pixeldata is no longer needed (the back porch) and the module returns to its initial state.

Fixed Point

For the x/y values we use a consistent fixed point system, with the first 10 bits being the integer part (enough to store -512 to 511) and the lower 6 being the fractional part. Because the sin(theta) value is so small, we use all 16 bits for the fractional part, which in software simulation is enough precision to draw the circle.

2.3.6 *Testing*

Once the warped module was coded in VHDL, we used a simple test-bench (just providing the initial values and the pixel-count) to test it in simulation. The values observed correspond exactly to our software simulation and are reported in the desired timing scheme.



The next step was to test it in isolation in hardware. We hooked up an artificial BRAM and used the OPB to report the address output from the warper. The module (plus fake BRAM and OPB controls) compiled and ran in hardware, reporting correct address values for a number of specific points (pixel-count values) that we checked.

Next, we hooked up a real BRAM, writing to it from C-code, and ran the warper, reading those values from the BRAM. This did not work very well, and we eventually abandoned it in favor of hard-coding look-up-tables into the vhdl.

Finally, we were able to successfully integrate the warping module into the video component and unwarp our video. Although there was a bug with the math, and the end result was not 100% unwarped, it is only a matter of going through the math and fixing some constants in the code to finish it completely.

2.3.6 Abbreviated vga timing diagram

	pixel_count	line_count	VGA_TIMING.VHD is tricky -- nothing interesting happens
			until many 25.175 clock cycles...this divides everything relative to pixel_count
h_sync <= '1'	655 to 753		which is equivalent to the clock cycle until it reaches 799 in which it is reset.
h_blank <= '1'	658 to 798		
hold_vga_ram_address <= 1	637		
hold_vga_ram_address <= '0'	796		
pixel_count <= "0000000000"	796		
line_count <= "00000000000"	799		
Conditionals			
linecount <= "0000000000"	799		
elseif			
linecount <= linecount + 1	799	523	
v_sync <= '1'	799	490	
elseif			
v_sync <= '0'	799	475	
h_blank <= '1'	638		
elseif			
h_blank <= '0'	798		
v_blank <= '1'	798	479	
elseif			
v_blank <= '0'	798	523	
		hold_ram_read_address	
ram_read_address_counter <= '0000000000'		0	
elseif			
ram_address_counter <= ram_address_counter + 1		< 0 , > 0	

3.0 Miscellaneous

3.1 Team Structure

Team Member	Responsibilities
Bryan Gwin	Video input/output, warper_block_ram module
David Lariviere	Warping theory and module
DJ Park	Video input/output, warper_block_ram module,
Mike Verbalis	Warping theory and module

3.2 Lessons learned

Bryan Gwin

I was in charge of coordinating the output signals to the display based on the pixel address selection in the warper component. I also aided in the design of the BRAM storage component which stores the x, y, and theta values generated by calculations in a C-software calculation module. I also mapped out the timing for the vga components a reference to steer the warper design so it could be integrated without conflict during the pixel address calculation and DAC processing in order to optimize the time complexity from input to output. I also designed the presentation, website, and worked on the documentation.

I definitely believe I relied too much on the TA's assurance that the imported output components necessary for our system where sufficient for our design, based on what I now know was a reasonably limited understanding of the complexity of our approach. I believe there should've been much more thorough communication between myself and the warper design leader on the specific needs for the warping component that would have motivated a deeper understanding of the system (specifically, the timing and number of pixel addresses referenced in obtrak that later contrasted with the evolving warping plans for the system). My limited exposure to hardware (VHDL) on a non-superficial level impaired me in being more critical of task-completion assumptions made as the liaison between the team and the TA, which inadvertently led to some troubles at the end. This consequence made me quickly aware of the level of communication required for complex low-level work in order to better constrain my informational search -- which would have been a big help during my initial multi-tasking planning beforehand in this time-sensitive environment. I believe it can be achieved in the future by rigorously challenging vocal interpretations of the completion status, regardless of the source, and better understand the other compartmentalized roles under the umbrella of a shared goal in order to make those decisions independent of outside input, when and if it is needed to have a team effort.

Had there been more resources available than the TA, who I found it difficult to coordinate meetings with due to our schedules, much of the low-level analysis would have gone much more smoothly than it did.

Regardless we pulled the hours in the end to compensate for any unexpected obstacles and achieved our goal. It was an interesting experience and I learned much about how to work with a hardware design group on the low-level aspect of an imaging/robotics project.

David Lariviere

I learned many things while working on this project, including many technical ones.

When the Xilinx tools attempts to synthesis the VHDL project and perform timing analysis, it may report that the timing constraint was not met, for example in the case where one has multicycle combinatorial paths. It turns out not to be necessary to actually specifically declare the timing constraints within the UCF file. While doing so would likely speed up synthesis, it had no effect on the actual circuit.

Another thing that I had learned, only after many hours of struggling around perplexed with the Xilinx tools, was that it is possible to map part of the Microblaze’s memory space onto the

SRAM, so that when can utilize software that is larger than could be fit strictly within the BRAMs of the FPGA.

Perhaps the best lesson was in recognizing the importance of simulation. We methodically developed our warper from simulation results almost in entirety before attempting to put it into hardware. Doing so cut down on development time dramatically. Further, it truly is absolutely essential to map out one's signals and systems on a per-clock level timing detail. Doing so made the development of the actual warper module almost easy, minus figuring out various tricks of optimizing the required translations and coordinate changes.

Lastly, I learned the importance of sleep. Without sleep one's brain turns to putty. Twenty hours in the lab can potentially result in the same work productivity as that of a Gerbil attempting to read the Wall Street Journal.

DJ Park

My main responsibilities during this project revolved around getting the video system working. After going through the code from Obtrak, mapping out the video system, and making changes, I was finally able to get the system up and running. In doing this, I learned a great lesson - GUI's can be cumbersome and stupid when designing hardware. After fumbling around for hours trying to get Obtrak up-to-date and to successfully compile on the XPS system, Cristian and I gave up and went back to using a custom makefile to compile and load all of the code. Another lesson I learned was the difference between good design and bad design. The video system itself was quite complex and in my attempt to use it, I began to see the necessity of good hardware design (i.e. well commented code, integration of software and hardware, good design of inputs/ouputs of components, and easy to follow data-paths logic) in a large system.

After the video system was up and running, Bryan and I also worked on building a block ram module that was intended to be integrated with the main warping component. It was designed to store parameter values that would be calculated by software and to be accessed by the warping module to generate the necessary memory addresses in the SRAM to pull pixel values from. Unfortunately, due to time constraints, we were never able to fully integrate this block ram into the warper. Luckily, we were able to hardcode the parameter values into the warper to get a working prototype in the end.

Finally, the greatest lesson that I learned was one of team work. Although I had worked in many different team environments, I had never before worked with a team of such diverse personalities on such an intense project. Fortunately, the group eventually was able to find a balance and get the project done, but it was definitely a challenge to get to that point. Overall, this project was a great learning experience and I think the final product that we came up with is pretty damn cool.

Mike Verbalis

I primarily worked on the vhdl warper code. I also helped with the math and design of the warper module. I also debugged the integrated version to get it displaying.

I learned a whole lot about video and hardware in general. I learned that VHDL really is not just C-for-hardware. In this respect I realized the awesome power of block designs. It helped so much in writing the VHDL to be working off a block design, and lowered the amount of debugging we

had to do to a minimum.

My advice to future groups would be to design more early. Of course, this is what every group advices every future group for every project I've been on. So, voiding that, my advice would be to divide the group in such a way that everyone knows one part very well, but also works on a related part with someone else. This way, everyone is paired and working in sync, but also able to work at their own pace. Also, be very clear about who's doing what from the beginning.

4.0 Appendix

4.1 Software Simulation Code

```
/*****
Modified by David Lariviere
Homework 5
W4165– Pixel Processing, Fall 2005
*****/

/* ======================================================================
 * IMPROC: Image Processing Software Package
 * Copyright (C) 2001 by George Wolberg
 *
 * omniview.skel - Skeleton program for generating perspective views
 *                 from omnivideo images.
 *
 * Written by: George Wolberg, 2001
 * ======================================================================
 */

#include "IP.h"
#include "pgm.h"

#define SGN(A)          ((A) > 0 ? 1 : ((A) < 0 ? -1 : 0 ))
#define FLOOR(A)        ((int) (A))
#define CEILING(A)      ((A)==FLOOR(A) ? FLOOR(A) : SGN(A)+FLOOR(A))
#define CLAMP(A,L,H)  ((A)<=(L) ? (L) : (A)<=(H) ? (A) : (H))

typedef double  vectorS[3];

/* Dot product: compute scalar A dot B */
#define vectorDot(A, B)  ((A[0]*B[0]) + (A[1]*B[1]) + (A[2]*B[2]))

/* Norm: compute length of vector A */
#define vectorNorm(A)   (sqrt( vectorDot(A,A) ))

/* Normalize vector A into unit vector */
#define vectorNormalize(A)      \
        {                       \
        double len;             \
        len = vectorNorm(A);    \
        if(len == 0) len = 1;   \
        A[0] /= len;            \
        A[1] /= len;            \
        A[2] /= len;            \
        }

/* Cross product: C = A x B */
#define vectorCross(A, B, C)    \
```

```
        {                               \
        vectorS T;                      \
        T[0] = A[1]*B[2] - A[2]*B[1];       \
        T[1] = A[2]*B[0] - A[0]*B[2];       \
        T[2] = A[0]*B[1] - A[1]*B[0];       \
        memcpy((char*) C, (char*) T, sizeof(vectorS));    \
        }

#define NN       0
#define LINEAR1
#define CUBIC  2

extern void omniview(imageP, int, int, int, int, int, int, imageP);

main(int argc, char **argv)
{
  char *inputFile, *outputFile;
  int newWidth, newHeight, func, dimensions;
  imageP inImg, outImg;
  int xc, yc, cop;
  int r, i;
  int numDivisions=12;
  double theta;
  if (argc!=9) {
        printf("Invalid arguements\n");
        printf("Usage: omniview input_img new_width new_height method xc yc cop output_img\n");
        exit(-1);
  }

  //usage: omniview input_img new_width new_height method xc yc cop output_img
  inputFile = argv[1];
  newWidth = atoi(argv[2]);
  newHeight = atoi(argv[3]);
  func = atoi(argv[4]); //0=>box, 1=>triangle, 2=>cubic
  xc = atoi(argv[5]);
  yc = atoi(argv[6]);
  cop = atoi(argv[7]);
  outputFile = argv[8];

  //read in source image
  inImg = IP_readImage(inputFile);

  xc = 223;
  yc = 223;
  r = 111;

  for (i=0; i<numDivisions; i++) {
        //allocate destination output image
        outImg = IP_allocImage(newWidth, newHeight, sizeof(uchar));
        theta = (double)(i) / (double) numDivisions * 2.0 * 3.14159;
        //resample the input image and put into outputImg
        omniview(inImg, xc+(r*cos(theta)), yc+(r*sin(theta)), cop, newWidth, newHeight, func, outImg);

        sprintf(outputFile, "output%d.pgm", i);
        //save image
        //IP_saveImage(outImg, outputFile);
        IP_savePGMImage(outImg, outputFile);
   }
}

/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * cubicConv:
```

```
 *
 * Cubic convolution filter.
 * A is preset here to -1
 */
double cubicConv(double t) {
        double A;
        double t2, t3;
        A = -1;

        if(t < 0) t = -t;
        t2 = t  * t;
        t3 = t2 * t;

        if(t < 1.0) return((A+2)*t3 - (A+3)*t2 + 1);
        if(t < 2.0) return(A*(t3 - 5*t2 + 8*t - 4));
        return(0.0);
}



/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * omniview:
 *
 * Compute perspective view from the omnivideo (parabolic) image in I1.
 * Output image I2 consists of the scene projected on the viewport.
 * The viewport dimensions are w x h. Its normal axis N passes through
 * the center of projection (origin) and the 3D point that corresponds
 * to the selected (xc,yc) on the omnivideo image.
 * The viewport lies a distance cop from the origin of the world coord sys.
 * filter specifies the intrp method: 0=NN; 1=bilinear.
 * NOTE: we assume that I1 has been cropped so that it is a square image
 * of dimensions 2r*2r, where r is the radius of the parabolic image.
 */
void
omniview(I1, xc, yc, cop, w, h, filter, I2)
imageP  I1, I2;
int         xc, yc, cop, w, h, filter;
{
        int         i, j, x, y, ix, iy, ww, hh;
        uchar    *ip, *op, *pp, bf[4];
        double   xx, yy, zc, r, rr, rmax, phi, theta;
        double   dx, dy;
        vectorS  U, V, N, P, S;
        double weight; //used during debugging
        double sum;
        /* allocate output buffer */
        //.....

        /* init input and output buffer pointers */
        ip = (uchar *) I1->image;
        op = (uchar *) I2->image;

        /* radius of parabolic mirror (in pixels) */
        rr = (ww = I1->width) / 2;

        //height
        hh = I1->height;

        /* move center (xc,yc) to origin and find zc, the position on mirror */
        xc = xc - rr;
        yc = yc - rr;
        zc = (rr*rr - (xc*xc + yc*yc)) / (2*rr);
```

```
/* N is unit vector along VPN: passes through (0,0,0) and (xc,yc,zc) */
N[0] = xc;
N[1] = yc;
N[2] = zc;
vectorNormalize(N);

/* U is unit vector of the perpendicular to N projected on z=0 plane */
U[0] = -N[1];
U[1] =  N[0];
U[2] =    0 ;
vectorNormalize(U);

/* V is the cross product of U and N: V = U x N */
vectorCross(U, N, V);

/* use P to store the world coords of viewport points */
P[2] = cop;

/* init rmax to account for neighboring pixels needed by filter */
rmax = (rr-filter) * (rr-filter);

/* project all viewport points into the parabolic image */
for(y=0; y<h; y++) {
        P[1] = y - h/2;
        for(x=0; x<w; x++) {
                P[0] = x - w/2;

                /* transform world coord point into UVN coord sys */
                S[0] = U[0]*P[0] + V[0]*P[1] + N[0]*P[2];
                S[1] = U[1]*P[0] + V[1]*P[1] + N[1]*P[2];
                S[2] = U[2]*P[0] + V[2]*P[1] + N[2]*P[2];
                vectorNormalize(S);

                /* spherical coords for vector through current pixel */
                phi   = atan2(S[1], S[0]);
                theta = acos (S[2]);

                /* radius of intersection on parabolic mirror */
                r  = 2 * rr * (1-cos(theta)) / (1-cos(2*theta));

                /* convert from polar to world coordinates */
                xx = r * sin(theta) * cos(phi);
                yy = r * sin(theta) * sin(phi);

                /* check if image point lies inside parabolic image */
                if(xx*xx + yy*yy < rmax) {
                        ix = xx + rr;
                        iy = yy + rr;
                        pp = &ip[iy*ww + ix];

                        switch(filter) {
                        case NN:
                                /* nearest neighbor */
                                *op++ = *pp;
                                break;
                        case LINEAR:
                                /* bilinear intrp */
                                dx = (xx + rr) - (double) FLOOR(xx + rr);
                                dy = (yy + rr) - (double) FLOOR(yy + rr);
                                *op++ = ((pp[0]*(1.0-dx) + pp[1]*dx)*(1-dy) + (*(pp+ww)*(1.0-dx) +
*(pp+ww+1)*dx)*dy);
```

```
                                        break;
                                case CUBIC:
                                        /* cubic convolution */
                                        dx = (xx + rr) - (double) FLOOR(xx + rr);
                                        dy = (yy + rr) - (double) FLOOR(yy + rr);
                                        //calculate cubic interpolation for rows
                                        sum = 0;
                                        for (i=0; i<5; i++) {
                                                for (j=0; j<5; j++) {
                                                        weight = cubicConv(dx-2.0 + (double) j) * cubicConv(dy-
2.0 + (double)i);
                                                        sum += ip[CLAMP((iy-2+i),0,hh)*ww + CLAMP((ix-
2+j),0,ww)] * weight;
                                                }
                                        }
                                        *op++ = CLAMP(sum, 0, 255);
                                        //*op++ = CLAMP(cubRows[1],0,255);
                                        //*op++ = CLAMP(cubRows[0] * cubicConv(dy - .0) + cubRows[1] *
cubicConv(dy) + cubRows[2] * cubicConv(dy+1.0), 0, 255);
                                        //*op++ = .....
                                        break;
                                }
                        } else  *op++ = 0;
                }
        }
}
```

4.2 Video Sub-system Code

```
----------------------------------------------------------------------------
--
-- VGA timing and address generator
--
-- Fixed-resolution address generator.  Generates h-sync, v-sync, and blanking
-- signals along with a 20-bit RAM address.  H-sync and v-sync signals are
-- delayed two cycles to compensate for the DAC pipeline.
--
-- Cristian Soviani, Dennis Lim, and Stephen A. Edwards
--
-- Modified by Team Hera
--
----------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity vga_timing is
  port (
    opb_clk      : in std_logic;
    pixel_clock  : in std_logic;
    reset        : in std_logic;
    h_sync_delay : out std_logic;
    v_sync_delay : out std_logic;
    blank        : out std_logic;
    vga_ram_read_address : out std_logic_vector(19 downto 0));
end vga_timing;

architecture Behavioral of vga_timing is
```

```
  constant SRAM_DELAY : integer := 3;

-- 640 X 480 @ 60Hz with a 25.175 MHz pixel clock
  constant H_ACTIVE      : integer := 640;
  constant H_FRONT_PORCH : integer := 16;
  constant H_BACK_PORCH  : integer := 48;
  constant H_TOTAL       : integer := 800;

  constant V_ACTIVE      : integer := 480;
  constant V_FRONT_PORCH : integer := 11;
  constant V_BACK_PORCH  : integer := 31;
  constant V_TOTAL       : integer := 524;

  signal line_count  : std_logic_vector (9 downto 0);   -- Y coordinate
  signal pixel_count : std_logic_vector (10 downto 0);  -- X coordinate

  signal h_sync : std_logic;  -- horizontal sync
  signal v_sync : std_logic;  -- vertical sync

  signal h_sync_delay0 : std_logic; -- h_sync delayed 1 clock
  signal v_sync_delay0 : std_logic; -- v_sync delayed 1 clock

  signal h_blank : std_logic;       -- horizontal blanking
  signal v_blank : std_logic;       -- vertical blanking

  -- flag to reset the ram address during vertical blanking
  signal reset_vga_ram_read_address : std_logic;

  -- flag to hold the address during horizontal blanking
  signal hold_vga_ram_read_address : std_logic;

  signal ram_address_counter : std_logic_vector (19 downto 0);
  signal T_addr : std_logic_vector(19 downto 0);

component warper is
    port(   rst, clk: in std_logic;
          linecount: in std_logic_vector( 9 downto 0 );
            pixelcount: in std_logic_vector( 10 downto 0 );
          addr: out std_logic_vector( 19 downto 0 )
    );
end component;

begin

  -- Pixel counter

  W1:warper port map(reset, opb_clk, line_count,pixel_count, T_addr);


  process ( pixel_clock, reset )
  begin
    if reset = '1' then
      pixel_count <= "00000000000";
    elsif pixel_clock'event and pixel_clock = '1' then
      if pixel_count = (H_TOTAL - 1) then
        pixel_count <= "00000000000";
      else
        pixel_count <= pixel_count + 1;
      end if;
    end if;
  end process;
```

```
-- Horizontal sync

process ( pixel_clock, reset )
begin
  if reset = '1' then
    h_sync <= '0';
  elsif pixel_clock'event and pixel_clock = '1' then
    if pixel_count = (H_ACTIVE + H_FRONT_PORCH - 1) then
      h_sync <= '1';
    elsif pixel_count = (H_TOTAL - H_BACK_PORCH - 1) then
      h_sync <= '0';
    end if;
  end if;
end process;

-- Line counter

process ( pixel_clock, reset )
begin
  if reset = '1' then
    line_count <= "0000000000";
  elsif pixel_clock'event and pixel_clock = '1' then
    if ((line_count = V_TOTAL - 1) and (pixel_count = H_TOTAL - 1)) then
      line_count <= "0000000000";
    elsif pixel_count = (H_TOTAL - 1) then
      line_count <= line_count + 1;
    end if;
  end if;
end process;

-- Vertical sync

process ( pixel_clock, reset )
begin
  if reset = '1' then
    v_sync <= '0';
  elsif pixel_clock'event and pixel_clock = '1' then
    if line_count = (V_ACTIVE + V_FRONT_PORCH -1) and
       pixel_count = (H_TOTAL - 1) then
      v_sync <= '1';
    elsif line_count = (V_TOTAL - V_BACK_PORCH - 1) and
          pixel_count = (H_TOTAL - 1) then
      v_sync <= '0';
    end if;
  end if;
end process;

-- Add two-cycle delays to h/v_sync to compensate for the DAC pipeline

process ( pixel_clock, reset )
begin
  if reset = '1' then
    h_sync_delay0 <= '0';
    v_sync_delay0 <= '0';
    h_sync_delay  <= '0';
    v_sync_delay  <= '0';
  elsif pixel_clock'event and pixel_clock = '1' then
    h_sync_delay0 <= h_sync;
    v_sync_delay0 <= v_sync;
    h_sync_delay  <= h_sync_delay0;
    v_sync_delay  <= v_sync_delay0;
  end if;
end process;
```

```
-- Horizontal blanking

-- The constants are offset by two to compensate for the delay
-- in the composite blanking signal

process ( pixel_clock, reset )
begin
  if reset = '1' then
    h_blank <= '0';
  elsif pixel_clock'event and pixel_clock = '1' then
    if pixel_count = (H_ACTIVE - 2) then
      h_blank <= '1';
    elsif pixel_count = (H_TOTAL - 2) then
      h_blank <= '0';
    end if;
  end if;
end process;

-- Vertical Blanking

-- The constants are offset by two to compensate for the delay
-- in the composite blanking signal

process ( pixel_clock, reset )
begin
  if reset = '1' then
    v_blank <= '0';
  elsif pixel_clock'event and pixel_clock = '1' then
    if line_count = (V_ACTIVE - 1) and pixel_count = (H_TOTAL - 2) then
      v_blank <= '1';
    elsif line_count = (V_TOTAL - 1) and pixel_count = (H_TOTAL - 2) then
      v_blank <= '0';
    end if;
  end if;
end process;

-- Composite blanking

process ( pixel_clock, reset )
begin
  if reset = '1' then
    blank <= '0';
  elsif pixel_clock'event and pixel_clock = '1' then
    if (h_blank or v_blank) = '1' then
      blank <= '1';
    else
      blank <= '0';
    end if;
  end if;
end process;

-- RAM address counter

-- Two control signals:

-- reset_ram_read_address is active from the end of each field until the
-- beginning of the next

-- hold_vga_ram_read_address is active from the end of each line to the
-- start of the next

process ( pixel_clock, reset )
```

```
  begin
    if reset = '1' then
      reset_vga_ram_read_address <= '0';
    elsif pixel_clock'event and pixel_clock = '1' then
      if line_count = V_ACTIVE - 1 and
         pixel_count = ( (H_TOTAL - 1 ) - SRAM_DELAY ) then
        -- reset the address counter at the end of active video
        reset_vga_ram_read_address <= '1';
      elsif line_count = V_TOTAL - 1 and
           pixel_count =((H_TOTAL -1) - SRAM_DELAY) then
        -- re-enable the address counter at the start of active video
        reset_vga_ram_read_address <= '0';
      end if;
    end if;
  end process;

  process ( pixel_clock, reset )
  begin
    if reset = '1' then
      hold_vga_ram_read_address <= '0';
    elsif pixel_clock'event and pixel_clock = '1' then
      if pixel_count = ((H_ACTIVE - 1) - SRAM_DELAY) then
        -- hold the address counter at the end of active video
        hold_vga_ram_read_address <= '1';
      elsif pixel_count = ((H_TOTAL - 1) - SRAM_DELAY) then
        -- re-enable the address counter at the start of active video
        hold_vga_ram_read_address <= '0';
      end if;
    end if;
  end process;

  process ( pixel_clock, reset )
  begin
    if reset = '1' then
      ram_address_counter <= "00000000000000000000";
    elsif pixel_clock'event and pixel_clock = '1' then
      if reset_vga_ram_read_address = '1' then
        ram_address_counter <= "00000000000000000000";
      elsif hold_vga_ram_read_address = '0' then
        ram_address_counter <= ram_address_counter + 1;
      end if;
    end if;
  end process;

  vga_ram_read_address <= T_addr;--<= ram_address_counter;

end Behavioral;
```

4.3 Warping Module Code

```
-- Component: MULTIPLEXOR ------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity mux is
    port( rst, sLine: in std_logic;
        load, result: in std_logic_vector( 15 downto 0 );
        output: out std_logic_vector( 15 downto 0 )
    );
```

```
end mux;

architecture mux_arch of mux is
begin
    process( rst, sLine, load, result )
    begin
      if( rst = '1' ) then
          output <= (others => '0');
      elsif sLine = '0' then
          output <= load;
      else
          output <= result;
      end if;
    end process;
end mux_arch;

-- Component: 4to1 MULTIPLEXOR ----------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity mux4to1 is
    port( rst: in std_logic;
          sel: in std_logic_vector(1 downto 0);
        load00, load01, load10, load11: in std_logic_vector( 9 downto 0 );
        output: out std_logic_vector( 9 downto 0 )
    );
end mux4to1;

architecture mux4_arch of mux4to1 is
begin
    process( rst, sel, load00, load01, load10, load11)
    begin
      if( rst = '1' ) then
          output <= (others => '0');
      elsif sel = "00" then
          output <= load00;
      elsif sel = "01" then
          output <= load01;
        elsif sel = "10" then
          output <= load10;
        elsif sel="11" then
          output <= load11;
      end if;
    end process;
end mux4_arch;

-- Component: ADDER ---------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity adder is
    port( rst: in std_logic;
        x, y: in std_logic_vector( 15 downto 0 );
        aout: out std_logic_vector( 15 downto 0 )
    );
end adder;
```

```
architecture adder_arch of adder is
begin
    process( rst, x, y )
    begin
      if( rst = '1' ) then
          aout <= (others => '0');
        else
            aout <= x + y;
      end if;
    end process;
end adder_arch;

-- Component: REGISTER --------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity regis is
    port( rst, clk, load: in std_logic;
        input: in std_logic_vector( 15 downto 0 );
        output: out std_logic_vector( 15 downto 0 )
    );
end regis;

architecture regis_arc of regis is
begin
    process( rst, clk, load, input )
    begin
      if( rst = '1' ) then
          output <= (others => '0');
      elsif( clk'event and clk = '1') then
          if( load = '1' ) then
              output <= input;
          end if;
        end if;
    end process;
end regis_arc;

-- Component: 19-bit REGISTER -------------------------------------------------
-------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity regis20 is
    port( rst, clk, load: in std_logic;
        input: in std_logic_vector( 19 downto 0 );
        output: out std_logic_vector( 19 downto 0 )
    );
end regis20;

architecture regis20_arc of regis20 is
begin
    process( rst, clk, load, input )
    begin
      if( rst = '1' ) then
          output <= (others => '0');
      elsif( clk'event and clk = '1') then
          if( load = '1' ) then
```

```
              output <= input;
           end if;
         end if;
    end process;
end regis20_arc;


-- Component: COUNTER ----------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity counter is
    port( rst, clk: in std_logic;
        output: out std_logic_vector( 10 downto 0 )
    );
end counter;

architecture counter_arc of counter is
  signal count : std_logic_vector(10 downto 0);

begin
    process( rst, clk )
    begin
      if( rst = '1' ) then
          count <= (others => '0');
      elsif( clk'event and clk = '1') then
            count <= count + 1;
        end if;
    end process;

    output <= count;

end counter_arc;

-- component: Multiplier -------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.all;
USE ieee.std_logic_signed.all;


ENTITY mymult IS
      port (
        rst, clk: in std_logic;
      a: IN std_logic_VECTOR(15 downto 0);
      b: IN std_logic_VECTOR(15 downto 0);
      o: OUT std_logic_VECTOR(31 downto 0));
END mymult;

ARCHITECTURE mymult_a OF mymult IS

  signal aint: signed (15 downto 0);
  signal bint: signed (15 downto 0);

begin

  aint <= signed(a);
  bint <= signed(b);
```

```
  o <= std_logic_vector(aint*bint);

END mymult_a;


-- Top level design. Puts all components together

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
-- use work.all;

entity warper is
    port(   rst, clk: in std_logic;
          linecount: in std_logic_vector( 9 downto 0 );
            pixelcount: in std_logic_vector( 10 downto 0 );
--            bramDataIn: in std_logic_vector( 15 downto 0 );
--            bramAddr: out std_logic_vector( 9 downto 0 );
--            x_out: out std_logic_vector(15 downto 0 );
          addr: out std_logic_vector( 19 downto 0 )
    );
end warper;

architecture warper_arch of warper is

component mux is
    port(   rst, sLine: in std_logic;
            load, result: in std_logic_vector( 15 downto 0 );
            output: out std_logic_vector( 15 downto 0 )
    );
end component;

component mux4to1 is
    port( rst: in std_logic;
          sel: in std_logic_vector(1 downto 0);
        load00, load01, load10, load11: in std_logic_vector( 9 downto 0 );
        output: out std_logic_vector( 9 downto 0 )
    );
end component;

component adder is
    port(   rst: in std_logic;
            x, y: in std_logic_vector( 15 downto 0 );
            aout: out std_logic_vector( 15 downto 0 )
    );
end component;

component inverter is
    port(   rst: in std_logic;
            x: in std_logic_vector( 15 downto 0 );
            o: out std_logic_vector( 15 downto 0 )
    );
end component;

component regis is
    port(   rst, clk, load: in std_logic;
            input: in std_logic_vector( 15 downto 0 );
            output: out std_logic_vector( 15 downto 0 )
    );
end component;

component regis20 is
```

```
    port( rst, clk, load: in std_logic;
        input: in std_logic_vector( 19 downto 0 );
        output: out std_logic_vector( 19 downto 0 )
    );
end component;

component mymult is
    port(   rst, clk: in std_logic;
                a, b: in std_logic_vector( 15 downto 0 );
                o: out std_logic_vector(31 downto 0 )
    );
end component;


-- These are the key components to the math of the warper

constant yOriginTranslatePlusOne: std_logic_vector(15 downto 0):=X"F001";
            --:= "0011110000"&"000001";  -- 240 . .1 (fixed point)
constant xOriginTranslateAndLineMax: std_logic_vector(15 downto 0):=X"0000";--
:=X"D800";
            --:= "0001010000"&"000000";  -- 320 - 240 = 80 (fixed point)
constant bramAddrXOffset: std_logic_vector(9 downto 0)
            := "0000000000";            -- 0
constant bramAddrYOffset: std_logic_vector(9 downto 0)
            := "0100000000";            -- 256
constant bramAddrSinOffset: std_logic_vector(9 downto 0)
            := "1000000000";            -- 512 (address doesn't care about 2's
comp)
constant unused: std_logic_vector(9 downto 0 )
            := "0000000000";            -- Just for the 4to1 mux
constant pixelCountSetup: std_logic_vector(10 downto 0):= "01100010000";
--            := "00000000000";           -- 0
constant pixelCountStart: std_logic_vector(10 downto 0):= "00000000000";
--            := "00010001100";           -- ~140
constant pixelCountStop : std_logic_vector(10 downto 0):= "01010000000";
--            := "01100001101";           -- 781

type ram_type is array(0 to 239) of std_logic_vector(15 downto 0);
constant X_RAM : ram_type :=
  (
    X"5367",
X"5337",X"5307",X"52D8",X"52A8",X"5278",
X"5248",X"5219",X"51E9",X"51BA",X"518B",
X"515B",X"512C",X"50FD",X"50CE",X"509F",
X"5070",X"5041",X"5012",X"4FE4",X"4FB5",
X"4F87",X"4F58",X"4F2A",X"4EFC",X"4ECD",
X"4E9F",X"4E71",X"4E43",X"4E15",X"4DE7",
X"4DBA",X"4D8C",X"4D5F",X"4D31",X"4D04",
X"4CD6",X"4CA9",X"4C7C",X"4C4F",X"4C22",
X"4BF5",X"4BC8",X"4B9B",X"4B6F",X"4B42",
X"4B16",X"4AE9",X"4ABD",X"4A91",X"4A65",
X"4A39",X"4A0D",X"49E1",X"49B5",X"498A",
X"495E",X"4933",X"4907",X"48DC",X"48B1",
X"4886",X"485B",X"4830",X"4805",X"47DA",
X"47B0",X"4785",X"475B",X"4730",X"4706",
X"46DC",X"46B2",X"4688",X"465E",X"4634",
X"460B",X"45E1",X"45B8",X"458E",X"4565",
X"453C",X"4513",X"44EA",X"44C1",X"4498",
X"4470",X"4447",X"441F",X"43F7",X"43CE",
X"43A6",X"437E",X"4357",X"432F",X"4307",
X"42E0",X"42B8",X"4291",X"426A",X"4243",
X"421C",X"41F5",X"41CE",X"41A7",X"4181",
X"415A",X"4134",X"410E",X"40E8",X"40C2",
```

```
X"409C",X"4076",X"4051",X"402B",X"4006",
X"3FE1",X"3FBB",X"3F96",X"3F72",X"3F4D",
X"3F28",X"3F04",X"3EDF",X"3EBB",X"3E97",
X"3E73",X"3E4F",X"3E2B",X"3E07",X"3DE4",
X"3DC0",X"3D9D",X"3D7A",X"3D57",X"3D34",
X"3D11",X"3CEE",X"3CCC",X"3CAA",X"3C87",
X"3C65",X"3C43",X"3C21",X"3C00",X"3BDE",
X"3BBC",X"3B9B",X"3B7A",X"3B59",X"3B38",
X"3B17",X"3AF6",X"3AD6",X"3AB5",X"3A95",
X"3A75",X"3A55",X"3A35",X"3A15",X"39F6",
X"39D6",X"39B7",X"3998",X"3979",X"395A",
X"393B",X"391C",X"38FE",X"38DF",X"38C1",
X"38A3",X"3885",X"3867",X"384A",X"382C",
X"380F",X"37F2",X"37D5",X"37B8",X"379B",
X"377E",X"3762",X"3745",X"3729",X"370D",
X"36F1",X"36D5",X"36BA",X"369E",X"3683",
X"3668",X"364D",X"3632",X"3617",X"35FD",
X"35E2",X"35C8",X"35AE",X"3594",X"357A",
X"3560",X"3547",X"352D",X"3514",X"34FB",
X"34E2",X"34C9",X"34B0",X"3498",X"3480",
X"3467",X"344F",X"3437",X"3420",X"3408",
X"33F1",X"33D9",X"33C2",X"33AB",X"3394",
X"337E",X"3367",X"3351",X"333B",X"3324",
X"330F",X"32F9",X"32E3",X"32CE",X"32B8",
X"32A3",X"328E",X"3279",X"3265",X"3250",
X"323C",X"3228",X"3213",X"3200"
    );

constant Y_RAM : ram_type :=
  (
X"0F9A",
X"0FA5",X"0FAF",X"0FBA",X"0FC5",X"0FD0",
X"0FDB",X"0FE6",X"0FF1",X"0FFC",X"1007",
X"1012",X"101D",X"1028",X"1034",X"103F",
X"104B",X"1056",X"1061",X"106D",X"1079",
X"1084",X"1090",X"109C",X"10A8",X"10B3",
X"10BF",X"10CB",X"10D7",X"10E3",X"10EF",
X"10FC",X"1108",X"1114",X"1121",X"112D",
X"1139",X"1146",X"1152",X"115F",X"116C",
X"1179",X"1185",X"1192",X"119F",X"11AC",
X"11B9",X"11C6",X"11D3",X"11E0",X"11EE",
X"11FB",X"1208",X"1216",X"1223",X"1231",
X"123F",X"124C",X"125A",X"1268",X"1276",
X"1284",X"1292",X"12A0",X"12AE",X"12BC",
X"12CA",X"12D9",X"12E7",X"12F5",X"1304",
X"1312",X"1321",X"1330",X"133F",X"134D",
X"135C",X"136B",X"137A",X"1389",X"1399",
X"13A8",X"13B7",X"13C7",X"13D6",X"13E5",
X"13F5",X"1405",X"1414",X"1424",X"1434",
X"1444",X"1454",X"1464",X"1474",X"1485",
X"1495",X"14A5",X"14B6",X"14C6",X"14D7",
X"14E8",X"14F8",X"1509",X"151A",X"152B",
X"153C",X"154D",X"155E",X"1570",X"1581",
X"1592",X"15A4",X"15B6",X"15C7",X"15D9",
X"15EB",X"15FD",X"160F",X"1621",X"1633",
X"1645",X"1658",X"166A",X"167C",X"168F",
X"16A2",X"16B4",X"16C7",X"16DA",X"16ED",
X"1700",X"1713",X"1726",X"173A",X"174D",
X"1761",X"1774",X"1788",X"179C",X"17AF",
X"17C3",X"17D7",X"17EB",X"1800",X"1814",
X"1828",X"183D",X"1851",X"1866",X"187A",
X"188F",X"18A4",X"18B9",X"18CE",X"18E3",
X"18F8",X"190E",X"1923",X"1939",X"194E",
```

```
X"1964",X"197A",X"1990",X"19A6",X"19BC",
X"19D2",X"19E8",X"19FE",X"1A15",X"1A2B",
X"1A42",X"1A59",X"1A6F",X"1A86",X"1A9D",
X"1AB4",X"1ACC",X"1AE3",X"1AFA",X"1B12",
X"1B29",X"1B41",X"1B59",X"1B71",X"1B89",
X"1BA1",X"1BB9",X"1BD1",X"1BE9",X"1C02",
X"1C1A",X"1C33",X"1C4C",X"1C65",X"1C7E",
X"1C97",X"1CB0",X"1CC9",X"1CE2",X"1CFC",
X"1D15",X"1D2F",X"1D49",X"1D62",X"1D7C",
X"1D96",X"1DB1",X"1DCB",X"1DE5",X"1E00",
X"1E1A",X"1E35",X"1E4F",X"1E6A",X"1E85",
X"1EA0",X"1EBB",X"1ED7",X"1EF2",X"1F0D",
X"1F29",X"1F45",X"1F60",X"1F7C",X"1F98",
X"1FB4",X"1FD0",X"1FED",X"2009",X"2025",
X"2042",X"205F",X"207B",X"2098",X"20B5",
X"20D2",X"20F0",X"210D",X"212A"
    );

constant SIN_RAM : ram_type :=
  (
X"004B",
X"004C",X"004C",X"004C",X"004D",X"004D",
X"004D",X"004E",X"004E",X"004F",X"004F",
X"004F",X"0050",X"0050",X"0051",X"0051",
X"0051",X"0052",X"0052",X"0053",X"0053",
X"0053",X"0054",X"0054",X"0055",X"0055",
X"0055",X"0056",X"0056",X"0056",X"0056",
X"0057",X"0057",X"0057",X"0058",X"0058",
X"0059",X"0059",X"005A",X"005A",X"005B",
X"005B",X"005B",X"005B",X"005C",X"005C",
X"005C",X"005D",X"005D",X"005E",X"005E",
X"005E",X"005E",X"005F",X"005F",X"0060",
X"0060",X"0061",X"0061",X"0061",X"0061",
X"0062",X"0062",X"0063",X"0063",X"0064",
X"0065",X"0064",X"0065",X"0065",X"0066",
X"0066",X"0067",X"0066",X"0067",X"0067",
X"0068",X"0069",X"0069",X"0069",X"0069",
X"006A",X"006A",X"006B",X"006B",X"006B",
X"006C",X"006C",X"006D",X"006D",X"006D",
X"006E",X"006E",X"006F",X"006F",X"006F",
X"0070",X"0070",X"0071",X"0071",X"0071",
X"0072",X"0073",X"0072",X"0073",X"0074",
X"0074",X"0075",X"0075",X"0075",X"0076",
X"0077",X"0076",X"0077",X"0078",X"0078",
X"0078",X"0079",X"0079",X"0079",X"007A",
X"007A",X"007B",X"007B",X"007B",X"007C",
X"007D",X"007D",X"007D",X"007E",X"007E",
X"007F",X"007F",X"0080",X"0080",X"0081",
X"0081",X"0081",X"0082",X"0083",X"0082",
X"0083",X"0084",X"0084",X"0084",X"0085",
X"0085",X"0086",X"0086",X"0086",X"0087",
X"0088",X"0088",X"0088",X"0089",X"0089",
X"008A",X"008B",X"008A",X"008B",X"008C",
X"008C",X"008D",X"008E",X"008D",X"008E",
X"008F",X"008F",X"0090",X"0090",X"0090",
X"0091",X"0091",X"0092",X"0092",X"0093",
X"0094",X"0093",X"0094",X"0094",X"0095",
X"0096",X"0096",X"0097",X"0097",X"0097",
X"0098",X"0098",X"0099",X"0099",X"009A",
X"009B",X"009B",X"009C",X"009B",X"009C",
X"009C",X"009D",X"009D",X"009E",X"009F",
X"009F",X"00A0",X"00A0",X"00A1",X"00A1",
X"00A1",X"00A1",X"00A2",X"00A2",X"00A3",
```

```
X"00A4",X"00A4",X"00A5",X"00A5",X"00A6",
X"00A6",X"00A7",X"00A7",X"00A8",X"00A7",
X"00A8",X"00A8",X"00A9",X"00A9",X"00AA",
X"00AA",X"00AB",X"00AB",X"00AC",X"00AC",
X"00AD",X"00AD",X"00AE",X"00AE",X"00AF",
X"00AF",X"00B0",X"00AF",X"00AF");

--------------------------------------------------------------------------------

--holds the output of the respective adders for x & y
signal xAdderOut, yAdderOut, lineAddOut : std_logic_vector( 15 downto 0 );
signal yMultByConstAddOut : std_logic_vector(19 downto 0);
signal multSinTimesYOut, multNegSinTimesXOut: std_logic_vector( 31 downto 0);

-- Intermediate signals
signal originalX, originalY, originalSinTheta, originalNegSinTheta:
std_logic_vector(15 downto 0);
signal xAbsRound, yAbsRound : std_logic_vector(9 downto 0);
signal notY, notLine, extLine, negBramDataIn: std_logic_vector(15 downto 0);
signal yTenShift,yNineShift,ySevenShift,yFiveShift: std_logic_vector(19 downto 0);
signal finalX, finalY, addrAddOut, addrDrop : std_logic_vector(19 downto 0);

-- signals to control loading of registers and selection of muxes
signal
  xRegLoad,                             -- load X_REG with output of X_MUX
  yRegLoad,                             -- load Y_REG with output of Y_MUX
  addrRegLoad,                          -- load ADDR_REG with addrAddOut
  xRegInMuxSel,                         -- 0->load X from X_ADD, 1->load x_original
  yRegInMuxSel,                         -- 0->load Y from Y_ADD, 1->load y_original
  sinThetaRegLoad,                      -- load SIN_REG with original sinTheta
  negSinThetaRegLoad                    -- load NEG_SIN_REG with original
negSinTheta
  : std_logic;
signal bramAddrMuxSelect: std_logic_vector(1 downto 0); -- selects which bram to
load from

signal
  x,                                    -- holds the value of X_REG
  x_temp,
  y,                                    -- holds the value of Y_REG
  xAbsolute,                            -- holds the value of X_ABS_REG
  yAbsolute,                            -- holds the value of Y_ABS_REG
  sinTheta,                             -- holds the value of the SIN_REG
  negSinTheta                           -- holds the value of NEG_SIN_REG
  : std_logic_vector(15 downto 0);

-- these muxes feed the input of the x/y register inputs
-- begining of each scanline, we first load the initial x/y (from BRAM)
-- after that, we will load x/y with the output of their adders
signal
  xRegInMuxOut,                         -- output of X_REG_IN_MUX
  yRegInMuxOut                          -- output of Y_REG_IN_MUX
  : std_logic_vector(15 downto 0);
signal bramAddrOffset : std_logic_vector(9 downto 0);

--state machine
type WARPER_STATES is (INIT_STATE, LOAD_X, WAIT_X, LOAD_Y, LOAD_SIN, HOLD,
ADD_MULT_ONE, ADD_MULT_TWO, ADD_MULT_THREE, ADD_MULT_FOUR);
signal curState : WARPER_STATES := INIT_STATE;
signal nextState : WARPER_STATES;
```

```
begin

  originalX <= X_RAM(conv_integer(linecount(9 downto 1)));
  originalY <= Y_RAM(conv_integer(linecount(9 downto 1)));
  originalSinTheta <= SIN_RAM(conv_integer(linecount(9 downto 1)));
  originalNegSinTheta <= not originalSinTheta + "0000000000000001";

  X_REG_IN_MUX: mux port map( rst, xRegInMuxSel, xAdderOut, originalX, xRegInMuxOut
);
  Y_REG_IN_MUX: mux port map( rst, yRegInMuxSel, yAdderOut, originalY, yRegInMuxOut
);
  BRAM_ADDR_MUX: mux4to1 port map( rst, bramAddrMuxSelect, bramAddrXOffset,
bramAddrYOffset, bramAddrSinOffset, unused, bramAddrOffset);

  X_REG: regis port map( rst, clk, xRegLoad, xRegInMuxOut, x );
  Y_REG: regis port map( rst, clk, yRegLoad, yRegInMuxOut, y );
  SIN_REG: regis port map (rst, clk, sinThetaRegLoad, originalSinTheta, sinTheta);
  NEG_SIN_REG: regis port map ( rst, clk, negSinThetaRegLoad, originalNegSinTheta,
negSinTheta);
  ADDR_REG: regis20 port map( rst, clk, addrRegLoad, addrDrop, addr);

  X_MULT: mymult port map( rst, clk, sinTheta,    y, multSinTimesYOut );
  Y_MULT: mymult port map( rst, clk, negSinTheta, x, multNegSinTimesXOut );

  X_ADD: adder port map( rst, x, multSinTimesYOut   (31 downto 16), xAdderOut );
  Y_ADD: adder port map( rst, y, multNegSinTimesXOut(31 downto 16), yAdderOut );
  LINE_ADD: adder port map( rst, extLine, xOriginTranslateAndLineMax, lineAddOut );

  X_ABS_ADD: adder port map( rst, x, lineAddOut, xAbsolute );
  Y_ABS_ADD: adder port map( rst, notY, yOriginTranslatePlusOne, yAbsolute );

-- Non-16-bit adders:
  --bramAddr <= linecount(9 downto 1) + bramAddrOffset;
 -- negBramDataIn <= not bramDataIn + "0000000000000001";

  xAbsRound <= xAbsolute(15 downto 6) + ("000000000" & xAbsolute(5));
  yAbsRound <= yAbsolute(15 downto 6) + ("000000000" & yAbsolute(5)); --+
("000000000" & xAbsRound(0));

  yMultByConstAddOut <= yNineShift + ySevenShift;  -- multiply by 640 =
"0110000000"
  addrAddOut <= finalX + finalY;
  addrDrop <= addrAddOut(19 downto 1)&"0";
-- extending, inverting, chopping
  notY <= not y;
   -- (linecount+1)/2 -> fixed-pt&16-bit
  extLine <= "0" & linecount(9 downto 1) & "000000";

--  notLine <= not extLine + X"0001";

  --yTenShift <= yAbsRound & "0000000000";
  yNineShift  <= "0" & yAbsRound & "000000000";  -- shift by 9 for mult-by-640
  ySevenShift <= "000" & yAbsRound & "0000000";  -- again by 7
  --yFiveShift <= "00000" & yAbsRound & "00000";

  finalX <= "0000000000" & xAbsRound;  -- drop frac-part & extend
  finalY <= yMultByConstAddOut;  -- drop frac-part

  fsm_seq : process(rst, clk)
  begin
    if rst = '1' then
      curState <= INIT_STATE;
    elsif clk'event and clk = '1' then
```

```
    curState <= nextState;
  end if;
end process fsm_seq;

fsm_comb : process(rst, clk, curState, pixelcount)
begin
  xRegLoad <= '0';
  yRegLoad <= '0';
  sinThetaRegLoad <= '0';
  negSinThetaRegLoad <= '0';
  xRegInMuxSel <= '0';
  yRegInMuxSel <= '0';
  addrRegLoad <= '0';
  bramAddrMuxSelect <= "11";

  case curState is
    when INIT_STATE =>
      xRegInMuxSel <='1';
      yRegInMuxSel <='1';

      if (pixelcount = pixelCountSetup) then
        nextState <= LOAD_X;
      else
        nextState <= INIT_STATE;
      end if;

    when LOAD_X =>
      bramAddrMuxSelect <= "00";
      xRegLoad <= '1';
      xRegInMuxSel <= '1';

      nextState <= WAIT_X;

    when WAIT_X =>
      bramAddrMuxSelect <= "00";
      xRegLoad <= '1';
      xRegInMuxSel <= '1';

      nextState <= LOAD_Y;

    when LOAD_Y =>
      bramAddrMuxSelect <= "01";
      yRegLoad <= '1';
      yRegInMuxSel <= '1';

      nextState <= LOAD_SIN;

    when LOAD_SIN =>
      bramAddrMuxSelect <= "10";
      sinThetaRegLoad <= '1';
      negSinThetaRegLoad <= '1';
      -- Also set to load address here, so that once x/y have propagated, the
      -- initial value of addr is valid (otherwise we would skip the first one)
      addrRegLoad <= '1';


      nextState <= HOLD;

    when HOLD =>
      -- Everything must be low here (it is by default)

      if (pixelcount = pixelCountStart) then
        nextState <= ADD_MULT_ONE;
```

```
        else
          nextState <= HOLD;
        end if;

      when ADD_MULT_ONE =>
        xRegLoad <= '1';
        yRegLoad <= '1';

        nextState <= ADD_MULT_TWO;

      when ADD_MULT_TWO =>

        xRegLoad <= '0';
        yRegLoad <= '0';
        addrRegLoad <= '1';

        nextState <= ADD_MULT_THREE;

        -- Because we're outputting at 640x480, but the virtual projection to
        -- 320x240, we use each pixel twice
      when ADD_MULT_THREE =>
        addrRegLoad <= '0';

        nextState <= ADD_MULT_FOUR;

      when ADD_MULT_FOUR =>
        if (pixelcount >= pixelCountStop) then
          nextState <= INIT_STATE;
        else
          nextState <= ADD_MULT_ONE;
        end if;

      when others => null;
    end case;
  end process fsm_comb;

  -- for now just set these, later they will be from output of BRAM
--  originalX           <= "0101011000" & "110000";  -- 104.746 + 240
--  originalY           <= "0001000001" & "011110";  -- 65.4665
--  originalSinTheta    <= "0000000001001101";  -- ~.001172
--  originalNegSinTheta <= "1111111110110011";  -- ~(-.001172)



end warper_arch;
```