Finger Photo Editor

Nadia Makan
12.10.09

Abstract

The Finger Photo Editor is a simple photo editor with a more natural user interface. The user will use their fingers to edit the image, instead of the standard keyboard/mouse combination. The editing functions one can perform are Crop and Rotate.

Table of Contents

List of Figures

Introduction

The Finger Photo Editor is a unique tool for editing digital photographs. Instead of having to use a mouse to select editing options and areas of the picture, the user does all of this with their fingers, making this editor a much more natural tool to use. This project breaks up into two main parts: gesture recognition using the camera as a sensor, and the graphics involved with the manipulation of the photograph on the screen. The camera is pointed downward at a black table surface. The camera image is divided into two sections, left and right, in which the user's corresponding fingers reside. The left finger can select editing options down the left side, including: Crop, Rotate, and Restore. Then the right hand can select, for example, the locations of two opposite corners of the desired cropped picture. The user is given visual feedback on the screen of where his left and right index fingers are pointing, a blue arrow to mark the left finger location, and a red plus sign to mark the right finger location. Because the main focus of this project was the natural interface for the manipulation of digital images, using an actual digital photograph presented unrelated challenges and, because of time constraints, was not implemented.

Functionality

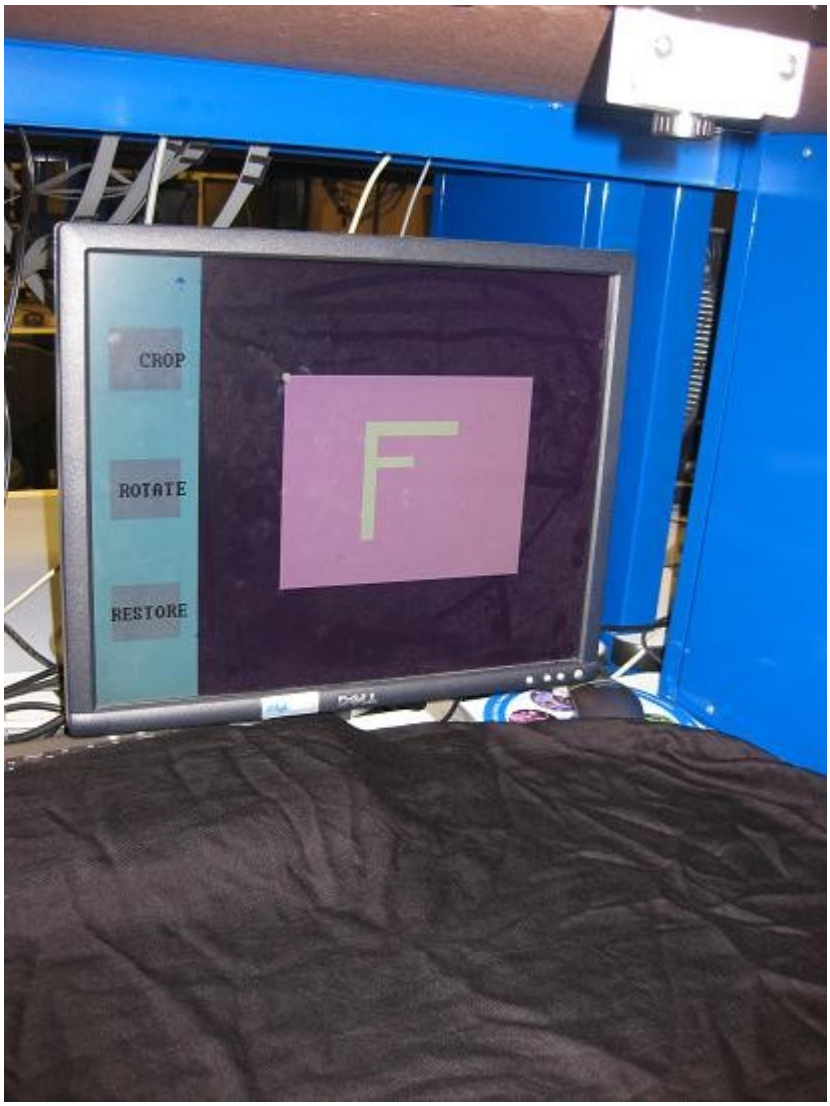

Figure 1. *This is what the user sees on the screen. The screen is divided into a left section and a right section, where the left and right fingers, respectively, will reside. There is a camera pointed down at the black table surface, which serves as the input.*

As mentioned above, a camera is pointed at a black table surface. The camera image is divided into two virtual sections that are displayed on the screen as shown in *Figure 1* above. The blue arrow in the left section marks the position of the tip of the user's left finger. The red plus sign in the right section marks the position of the tip of the user's right finger. These can be seen more clearly in *Figure 2* below.

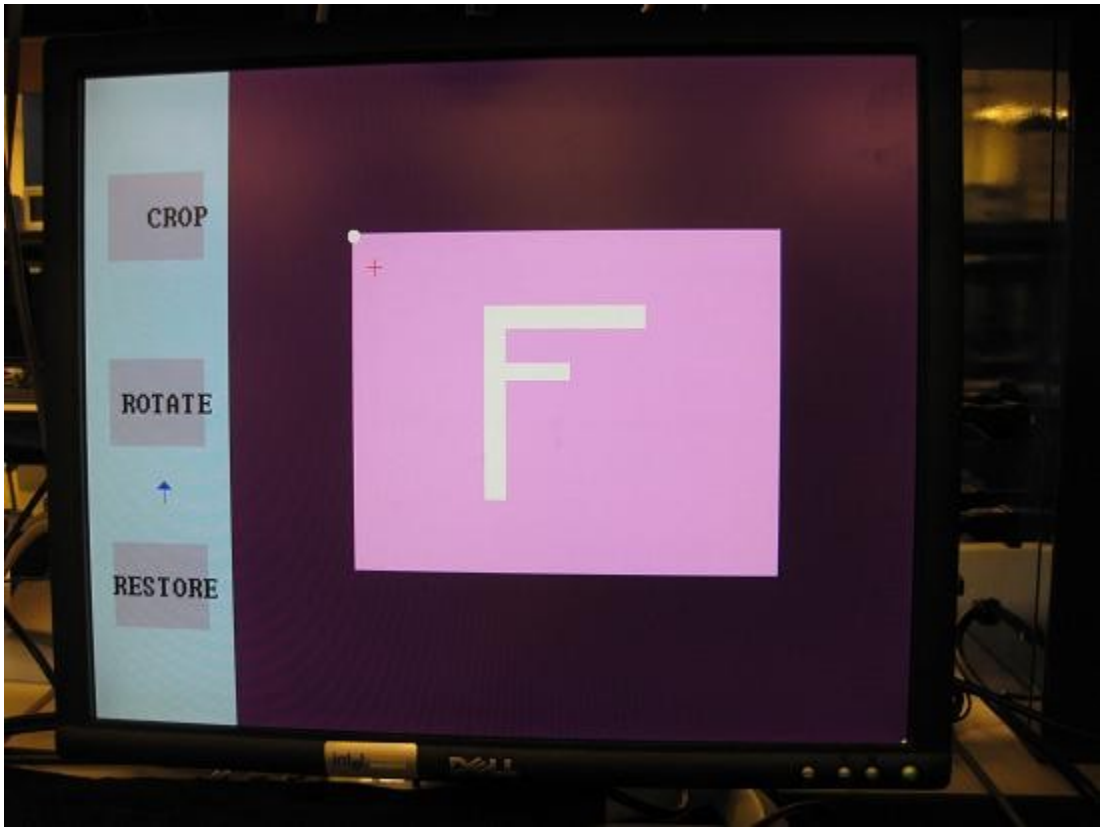

Figure 2. The user's left finger is marked by a blue arrow, and the right finger by a red plus sign.

In order to crop the image, the user needs to first make sure the arrow is not pointing at any editing option by moving their left finger. Then they will move their right finger so that the plus sign is sitting where they would like to start cropping (See *Figure 2* above). This position will become the top left corner of the cropped picture. The user will then move their left finger so that the arrow's tip is within CROP's rectangular boundary. Keeping the left finger still, they will drag their right finger to where they want the bottom right corner of the cropped picture to be. As they do this, they will see a rectangle drawn on the screen between the initial right finger position and the current right finger position (See *Figure 3* below). This helps them visualize the final image.

Figure 3. A rectangle is drawn so the user can visualize the final cropped image.

When they are satisfied with the boundary they have formed, the user will keep their right finger still, and move their left finger so that the arrow is no longer on CROP. The rest of the image is then cut away, and the cropped image is displayed on the screen (See *Figure 4* below).

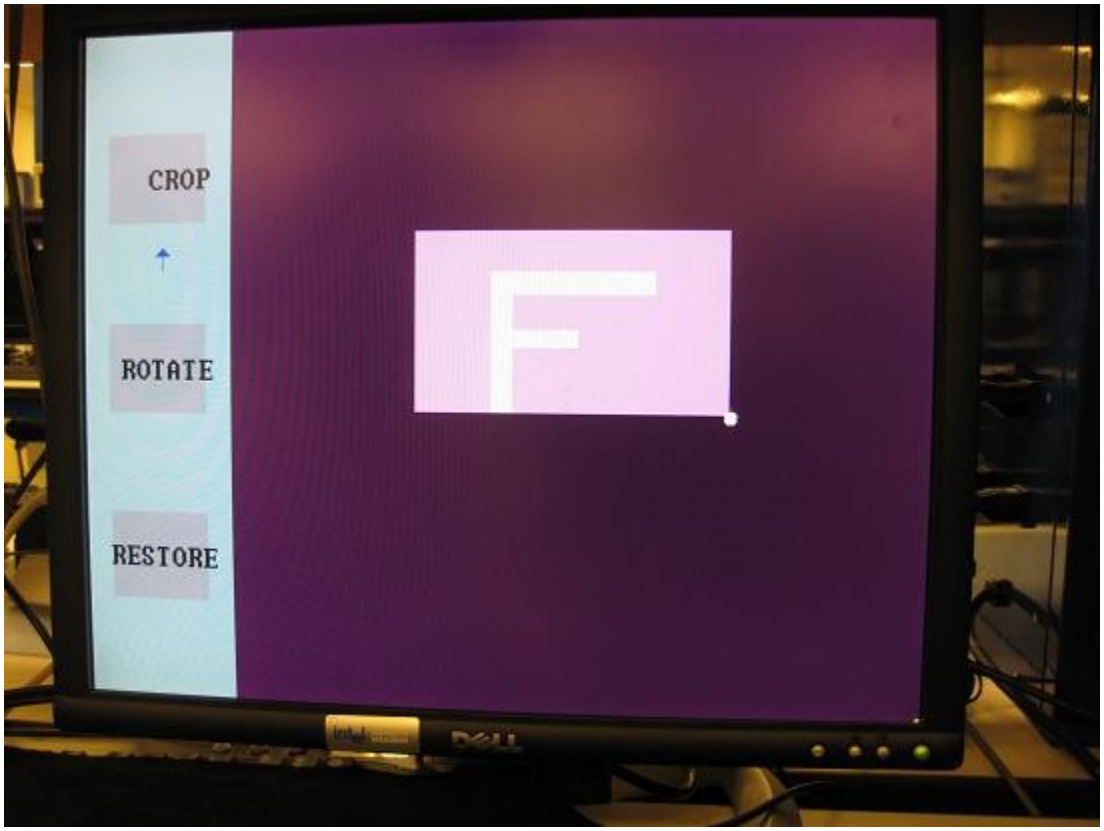

Figure 4. Cropped image.

The user can also rotate the image in multiples of 90 degrees. To do so, they will first make sure the arrow is not pointing at any editing option. Then they will select a reference corner by moving their right finger around until that the plus sign is nearest to this corner. As the user moves their finger around, they will see a circle drawn on the corner their finger is nearest to (See *Figure 2* above). This visual feedback helps them to be certain they have successfully selected the desired corner. Then keeping their right finger still, they will move their left finger so that the arrow's tip is within ROTATE's boundary. The circle over the reference corner that they selected will now remain stationary. Keeping the arrow on ROTATE, as they move their right finger around, a second circle will appear on the corner their right finger is closest to (See *Figure 5* below). Depending on how many degrees and in which direction they would like to rotate the image, they will decide which second corner to select. For example, if the reference corner is the top left corner and they would like to rotate the image 90 degrees clockwise, they should select the top right corner as the second corner (See *Figures 5 and 6* below). If they would like to rotate the image 90 degrees counter clockwise, they will select the bottom left corner as the second corner. And if they would like to rotate the image 180 degrees, they will select the bottom right corner as the second corner. Once they have selected the second corner, they will move the arrow off of ROTATE, and the image will rotate the desired amount.

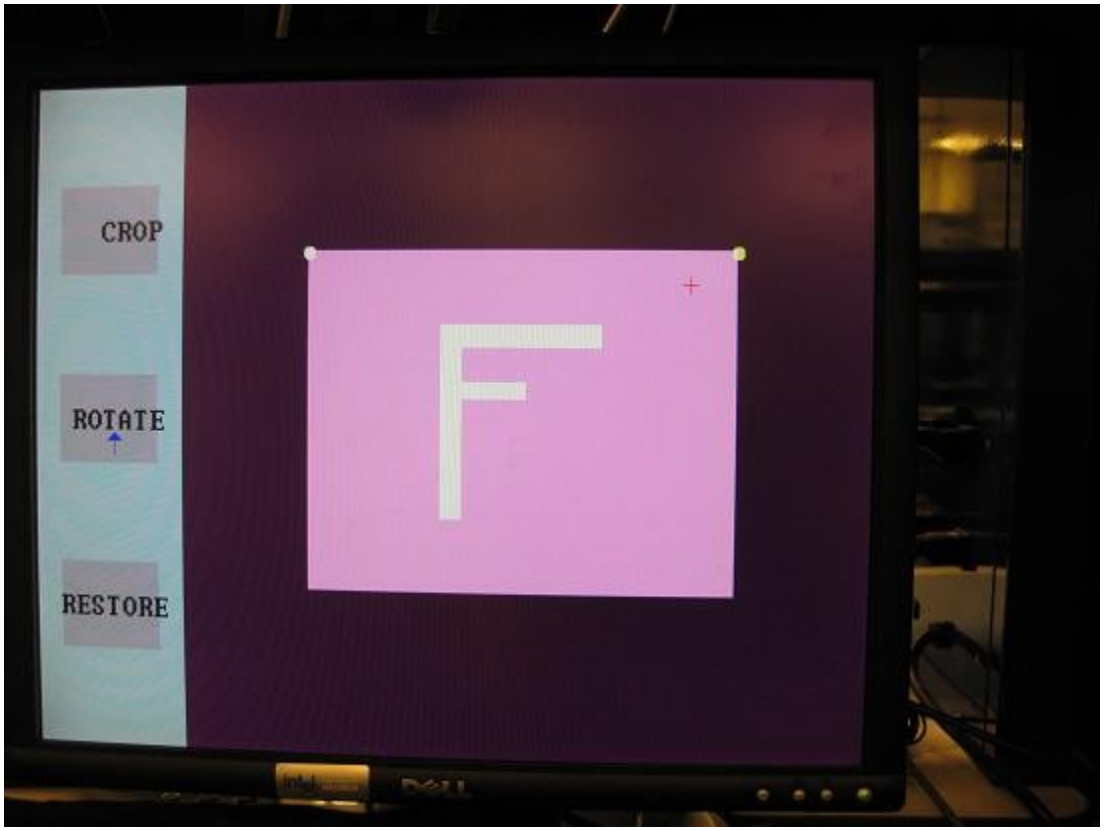

Figure 5. The circle around the reference corner is stationary, and a second circle is drawn around the corner the right finger is nearest to.

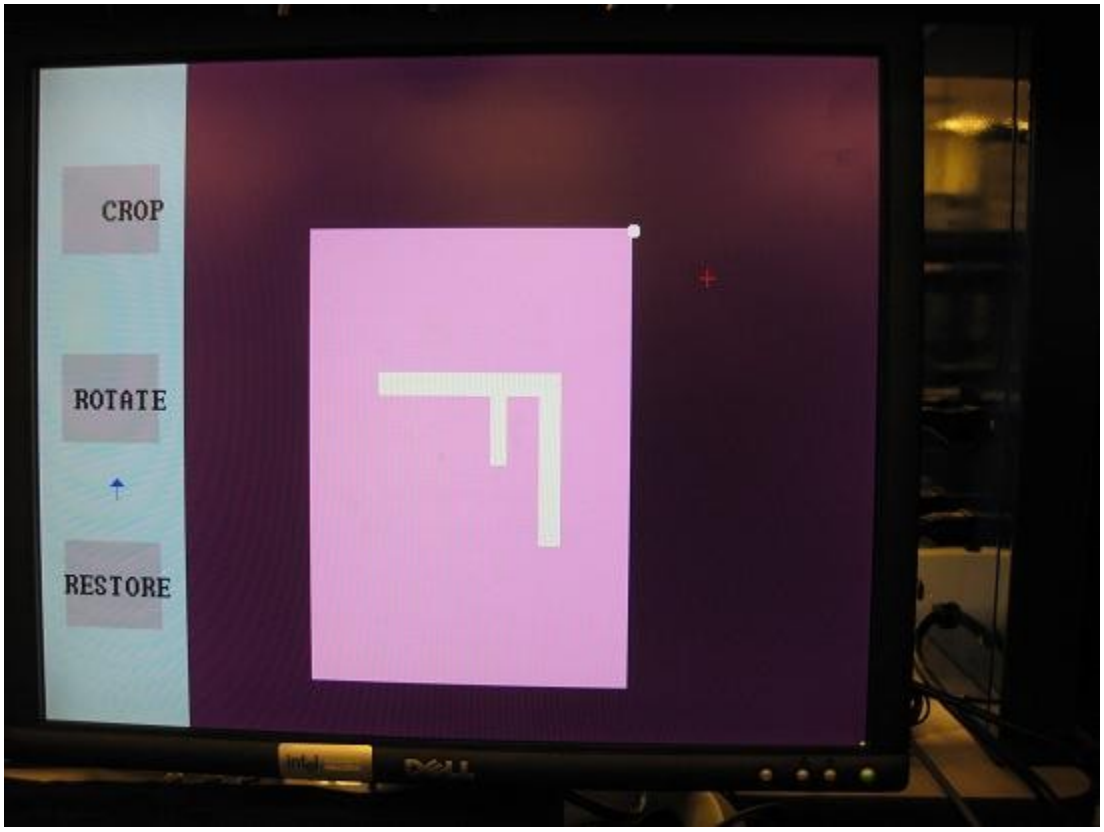

Figure 6. Image rotated 90 degrees clockwise.

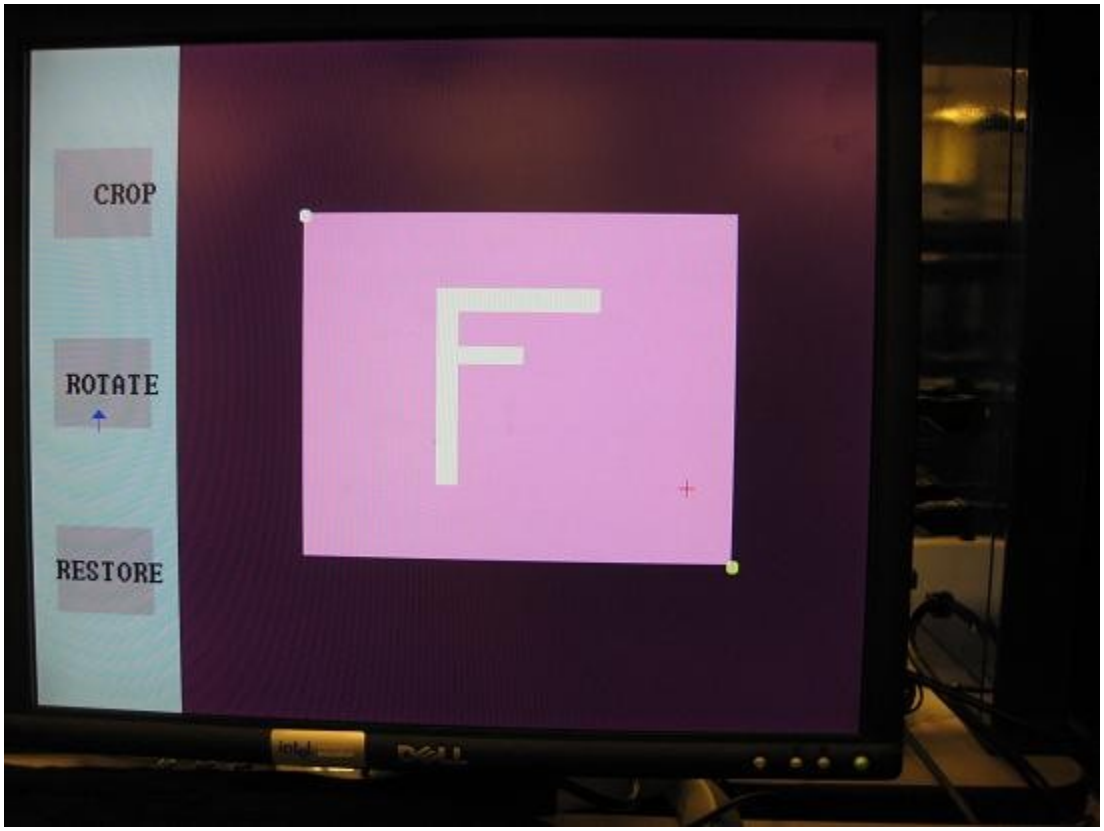

Figure 7. Right finger is nearest to bottom right corner, so circle is drawn around it.

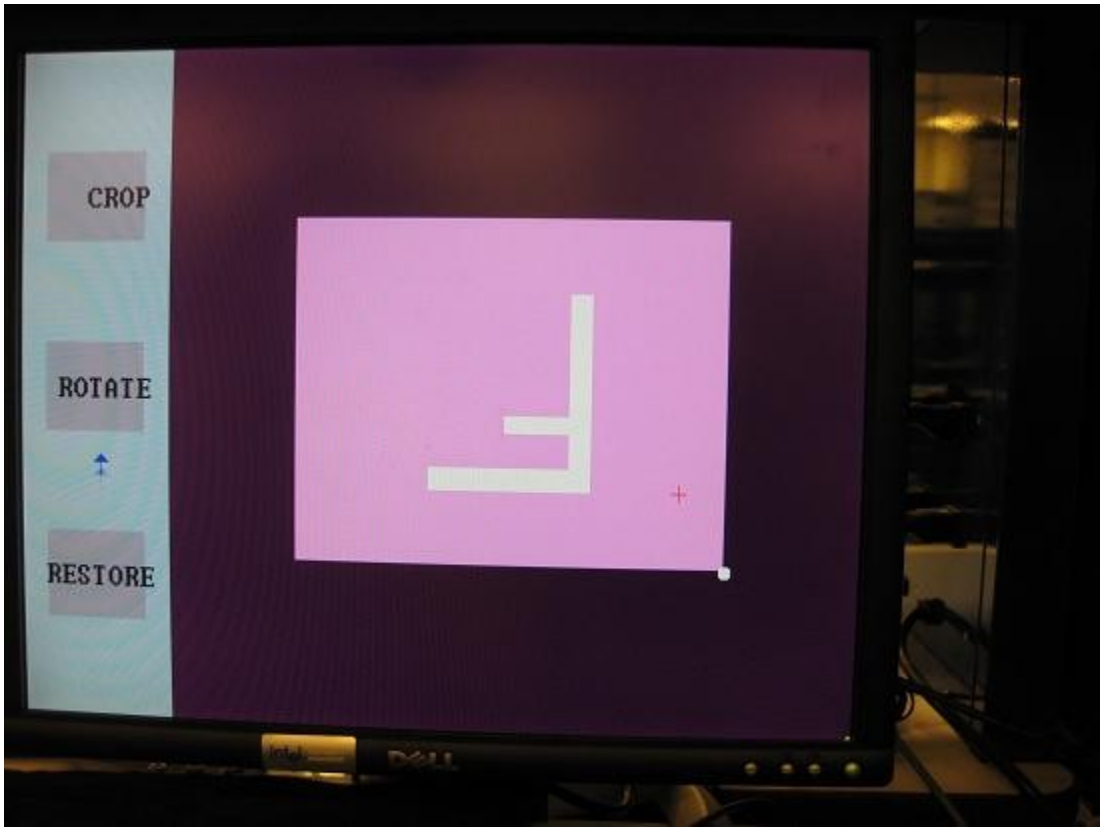

Figure 8. Image rotated 180 degrees.

If the user has made an edit that they wish to undo, they can use the third editing option to restore the image to it's original form. To do this, all they need to do is to move their left finger so that the arrow's tip is within RESTORE's boundary, and hold it there for three seconds.

Module Descriptions

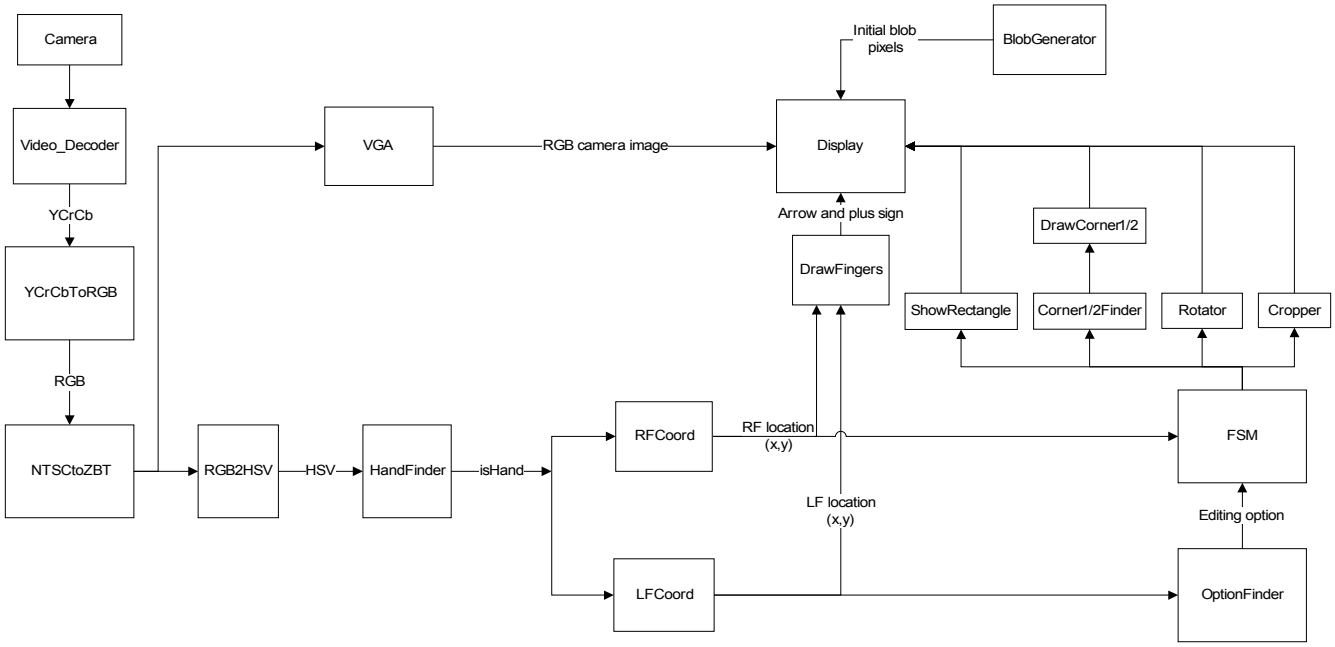

Figure 9. Block Diagram for the Finger Photo Editor

Video_Decoder

This module grabs the camera image, decodes it and spits out the pixels in YCrCb format. This module was obtained from the 6.111 website and not modified.

YCrCbToRGB

This module takes YCrCb values as input and converts them to RGB values. This module was obtained from the 6.111 website and not modified.

NTSCtoZBT

Here, the RGB values are stored in the ZBT RAM. This module was obtained from the 6.111

website. It originally only stored a black and white image, but was modified to store 18 bits per pixel in order to store color information as well. From the ZBT, the RGB image is sent both through a VGA cable to the screen (for testing purposes) and to the RGB2HSV module.

RGB2HSV

This module takes RGB values as input and converts them to HSV values, which are sent to the HandFinder. Only the V value is actually used to detect hand pixels.

HandFinder

This module determines whether or not each pixel is a 'hand pixel'. It takes in the V value from RGB2HSV and outputs a one-bit isHand value. If V is above some V_threshold, then isHand is 1, otherwise 0.

RFCoord & LFCoord

These modules are responsible for finding the tips of the right and left fingers, respectively. They do this by keeping track of the highest line of 12 'hand' pixels in a row in the image. This value updates on each new frame. RFCoord only looks at the right section of the image, and LFCoord only looks at the left section. They then output the (x,y) coordinate of the middle of this line to represent the position of the corresponding finger tip. These locations are fed to the DrawFingers module to be drawn on the screen for visual feedback.

DrawFingers

This module takes as input the x and y locations of the left and right fingers, and outputs a finger_pixel value which is comprised of the left finger marker and the right finger marker to be displayed on the screen. It generates pixel values for each marker by calling the arrow and plus modules, arrow for the left finger marker, and plus for the right finger marker.

Arrow

The arrow module takes the left finger x and y as input and outputs the pixels for a blue arrow marking it's position.

Plus

The plus module takes the right finger x and y as input and outputs the pixels for a red plus sign marking it's position.

OptionFinder

This module figures out which editing option the user has selected by looking at the coordinate of their left finger tip. If this point lies within one of the three rectangular editing option 'buttons', the corresponding option will be output to the FSM.

FSM

The FSM module takes as input the Editing Option from the OptionFinder module and the RF location from the RFCoord module, among other things. It outputs various signals to the modules responsible for doing the actual cropping and rotating (Cropper, Rotator), as well as the modules responsible for giving visual feedback before the actual cropping and rotating are done (ShowRectangle, Corner1Finder, and Corner2Finder,).

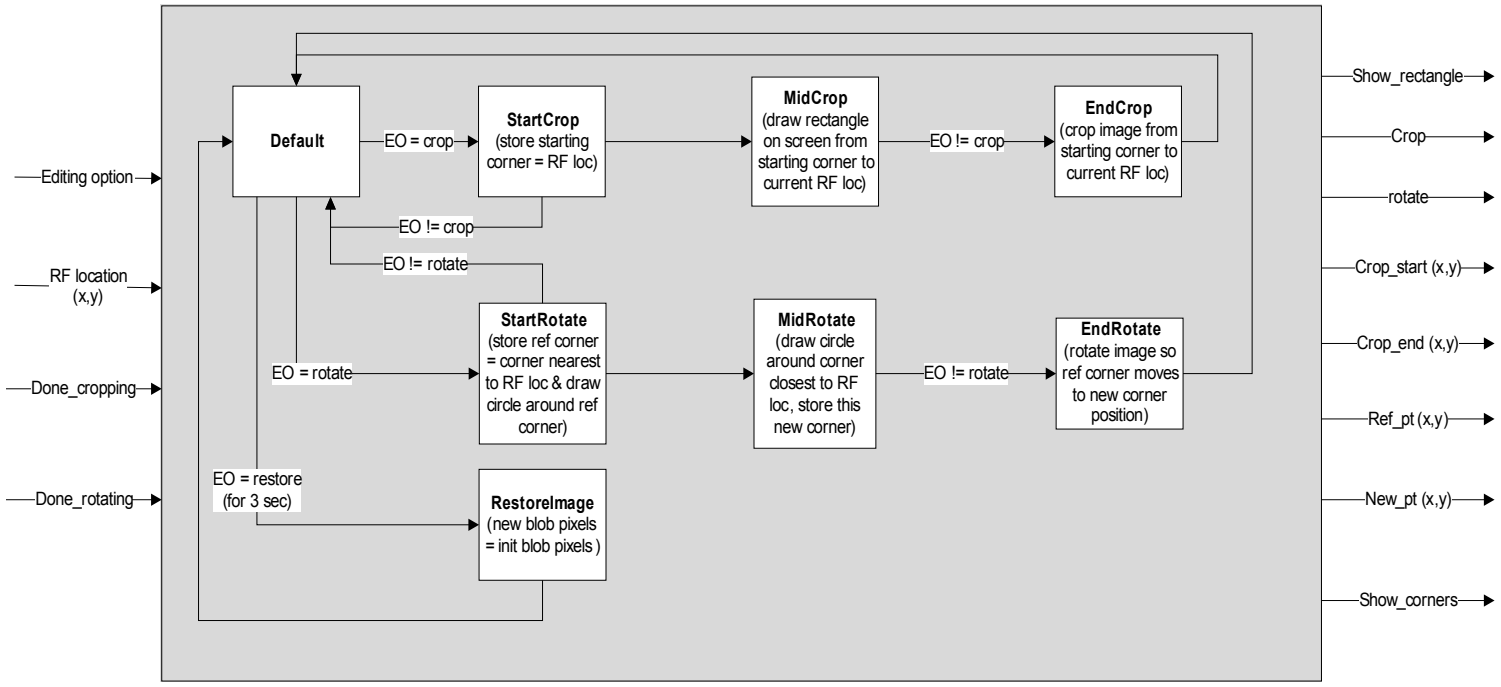

Figure 10. Finite State Machine

Default
This is the state the FSM will start in, and return to after an edit has been performed. If CROP is selected, the system will enter the StartCrop state, if ROTATE is selected, it will enter the StartRotate state, and if RESTORE is selected it will enter the Restore state.

StartCrop
When the CROP option is selected, the system will enter this state. The location of the right finger will be stored as a starting point, and then the system will immediately enter the MidCrop state.

MidCrop
When the system enters this state the *show_rectangle* signal goes high, and this tells the ShowRectangle module to draw a rectangle between the starting point and the current right finger location. The system will stay in this state until CROP is no longer selected as the editing option, then the current right finger x and y locations are stored as the bottom right corner of the cropped image, and the system enters the EndCrop state.

EndCrop
When the system enters this state, the *crop* signal goes high and this tells the Cropper module to cut the image down to the rectangle formed between the starting corner and the current right finger location, and this cropped image will replace it on the screen. When the Cropper module is done cropping the image, it sends a high *done_cropping* signal back to the FSM, and the system returns to the Default state.

StartRotate
When the ROTATE option is selected, the system will enter this state. The location of the right finger is stored as a reference point, and a stationary circle is drawn around the corner nearest to this point for visual feedback. Then the system will immediately enter the MidRotate state.

MidRotate
While in this state, a circle will remain around the reference corner, and another circle will be drawn around the corner closest to the right finger location. The right finger's x and y locations are sent to the Corner2Finder module, which figures out which corner it is closest to and gives this corner to the DrawCorner2 module to do the actual drawing. When ROTATE is no longer selected as the editing option, the system will store the current right finger location as the ending location and enter the EndRotate state.

EndRotate
In this state, the rotate signal is high, allowing the Rotator module to rotate the image the desired multiple of 90 degrees based on the identities of the reference corner and the new corner. Then the system will return to the Default state.

Restore
When the RESTORE option is selected for at least 3 seconds, the system will enter this state. Here, the image on the screen will be replaced with the original image that was generated in the BlobGenerator module. The system will then return to the Default state.

BlobGenerator

This module generates the screen layout (as shown in *Figure 2*, sans the arrow and plus sign). The image in the center of the screen is the 'blob' that will be edited by the user. It should be an image that is unique at every 90 degree rotation, for testing and debugging purposes, hence the big F in the middle. The BlobGenerator also makes sure the image parameters are updated whenever the image has been cropped or rotated and keeps track of them, so that the user gets visual feedback of the actual cropping and rotation.

DrawRectangle

This module takes the starting and ending crop locations from the FSM and draws a white rectangle on the screen while in the MidCrop state (See *Figure 3*).

Cropper

This module does the actual cropping when the *crop* signal is high and the system is in the EndCrop state. It takes in the starting and ending corner locations from the FSM, and the *image_width* and *image_height* values from the BlobGenerator module. It outputs a *done_cropping* signal back to the FSM.

Corner1Finder & Corner2Finder

Both of these modules take as input an x,y location and output the identity of the corner closest to that location. Corner1Finder takes the reference corner as input, from the FSM while the system is in the StartRotate state. Corner2Finder takes the current right finger location as input from the FSM while the system is in the MidRotate state.

DrawCorner1 & DrawCorner2

These modules take as input the respective corner from Corner1Finder and Corner2Finder, as

well as the *image_x*, *image_y*, *image_width*, *image_height* values from the BlobGenerator and draw a circle around that corner of the image.

GetRotatedParams

This module is called whenever a rotation occurs. It changes all of the parameters that have anything to do with the 'F' in the middle of the image.

Rotator

This module is reponsible for doing the actual rotation. It takes as input *corner1* and *corner2*, as well as all of the image and 'F' parameters, and based on the identities of the two corners, rotates the image some multiple of 90 degrees by calling GetRotatedParams.

VERILOG CODE

Top Level Module

```
//
// File:   zbt_6111_sample.v
// Date:   26-Nov-05
// Author: I. Chuang <ichuang@mit.edu>
//
// Sample code for the MIT 6.111 labkit demonstrating use of the ZBT
// memories for video display.  Video input from the NTSC digitizer is
// displayed within an XGA 1024x768 window.  One ZBT memory (ram0) is used
// as the video frame buffer, with 8 bits used per pixel (black & white).
//
// Since the ZBT is read once for every four pixels, this frees up time for
// data to be stored to the ZBT during other pixel times.  The NTSC decoder
// runs at 27 MHz, whereas the XGA runs at 65 MHz, so we synchronize
// signals between the two (see ntsc2zbt.v) and let the NTSC data be
// stored to ZBT memory whenever it is available, during cycles when
// pixel reads are not being performed.
//
// We use a very simple ZBT interface, which does not involve any clock
// generation or hiding of the pipelining.  See zbt_6111.v for more info.
//
// switch[7] selects between display of NTSC video and test bars
// switch[6] is used for testing the NTSC decoder
// switch[1] selects between test bar periods; these are stored to ZBT
//           during blanking periods
// switch[0] selects vertical test bars (hardwired; not stored in ZBT)

//`include "display_16hex.v"
//`include "debounce.v"
//`include "video_decoder.v"
//`include "zbt_6111.v"
//`include "ntsc2zbt.v"

////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Template Toplevel Module
//
// For Labkit Revision 004
```

```
//
//
// Created: October 31, 2004, from revision 003 file
// Author: Nathan Ickes
//
////////////////////////////////////////////////////////////////////////////////
//
// CHANGES FOR BOARD REVISION 004
//
// 1) Added signals for logic analyzer pods 2-4.
// 2) Expanded "tv_in_ycrcb" to 20 bits.
// 3) Renamed "tv_out_data" to "tv_out_i2c_data" and "tv_out_sclk" to
//    "tv_out_i2c_clock".
// 4) Reversed disp_data_in and disp_data_out signals, so that "out" is an
//    output of the FPGA, and "in" is an input.
//
// CHANGES FOR BOARD REVISION 003
//
// 1) Combined flash chip enables into a single signal, flash_ce_b.
//
// CHANGES FOR BOARD REVISION 002
//
// 1) Added SRAM clock feedback path input and output
// 2) Renamed "mousedata" to "mouse_data"
// 3) Renamed some ZBT memory signals. Parity bits are now incorporated into
//    the data bus, and the byte write enables have been combined into the
//    4-bit ram#_bwe_b bus.
// 4) Removed the "systemace_clock" net, since the SystemACE clock is now
//    hardwired on the PCB to the oscillator.
//
////////////////////////////////////////////////////////////////////////////////
//
// Complete change history (including bug fixes)
//
// 2005-Sep-09: Added missing default assignments to "ac97_sdata_out",
//              "disp_data_out", "analyzer[2-3]_clock" and
//              "analyzer[2-3]_data".
//
// 2005-Jan-23: Reduced flash address bus to 24 bits, to match 128Mb devices
//              actually populated on the boards. (The boards support up to
//              256Mb devices, with 25 address lines.)
//
// 2004-Oct-31: Adapted to new revision 004 board.
//
// 2004-May-01: Changed "disp_data_in" to be an output, and gave it a default
//              value. (Previous versions of this file declared this port to
//              be an input.)
//
// 2004-Apr-29: Reduced SRAM address busses to 19 bits, to match 18Mb devices
//              actually populated on the boards. (The boards support up to
//              72Mb devices, with 21 address lines.)
//
// 2004-Apr-29: Change history started
//
////////////////////////////////////////////////////////////////////////////////

module zbt_6111_sample(beep, audio_reset_b,
                       ac97_sdata_out, ac97_sdata_in, ac97_synch,
                ac97_bit_clock,

                vga_out_red, vga_out_green, vga_out_blue, vga_out_sync_b,
                vga_out_blank_b, vga_out_pixel_clock, vga_out_hsync,
                vga_out_vsync,

                tv_out_ycrcb, tv_out_reset_b, tv_out_clock, tv_out_i2c_clock,
                tv_out_i2c_data, tv_out_pal_ntsc, tv_out_hsync_b,
                tv_out_vsync_b, tv_out_blank_b, tv_out_subcar_reset,

                tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1,
                tv_in_line_clock2, tv_in_aef, tv_in_hff, tv_in_aff,
                tv_in_i2c_clock, tv_in_i2c_data, tv_in_fifo_read,
                tv_in_fifo_clock, tv_in_iso, tv_in_reset_b, tv_in_clock,

                ram0_data, ram0_address, ram0_adv_ld, ram0_clk, ram0_cen_b,
                ram0_ce_b, ram0_oe_b, ram0_we_b, ram0_bwe_b,

                ram1_data, ram1_address, ram1_adv_ld, ram1_clk, ram1_cen_b,
                ram1_ce_b, ram1_oe_b, ram1_we_b, ram1_bwe_b,

                clock_feedback_out, clock_feedback_in,

                flash_data, flash_address, flash_ce_b, flash_oe_b, flash_we_b,
                flash_reset_b, flash_sts, flash_byte_b,

                rs232_txd, rs232_rxd, rs232_rts, rs232_cts,
```

```
            mouse_clock, mouse_data, keyboard_clock, keyboard_data,

            clock_27mhz, clock1, clock2,

            disp_blank, disp_data_out, disp_clock, disp_rs, disp_ce_b,
            disp_reset_b, disp_data_in,

            button0, button1, button2, button3, button_enter, button_right,
            button_left, button_down, button_up,

            switch,

            led,

            user1, user2, user3, user4,

            daughtercard,

            systemace_data, systemace_address, systemace_ce_b,
            systemace_we_b, systemace_oe_b, systemace_irq, systemace_mpbrdy,

            analyzer1_data, analyzer1_clock,
            analyzer2_data, analyzer2_clock,
            analyzer3_data, analyzer3_clock,
            analyzer4_data, analyzer4_clock);

output beep, audio_reset_b, ac97_synch, ac97_sdata_out;
input  ac97_bit_clock, ac97_sdata_in;

output [7:0] vga_out_red, vga_out_green, vga_out_blue;
output vga_out_sync_b, vga_out_blank_b, vga_out_pixel_clock,
       vga_out_hsync, vga_out_vsync;

output [9:0] tv_out_ycrcb;
output tv_out_reset_b, tv_out_clock, tv_out_i2c_clock, tv_out_i2c_data,
       tv_out_pal_ntsc, tv_out_hsync_b, tv_out_vsync_b, tv_out_blank_b,
       tv_out_subcar_reset;

input  [19:0] tv_in_ycrcb;
input  tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2, tv_in_aef,
       tv_in_hff, tv_in_aff;
output tv_in_i2c_clock, tv_in_fifo_read, tv_in_fifo_clock, tv_in_iso,
       tv_in_reset_b, tv_in_clock;
inout  tv_in_i2c_data;

inout  [35:0] ram0_data;
output [18:0] ram0_address;
output ram0_adv_ld, ram0_clk, ram0_cen_b, ram0_ce_b, ram0_oe_b, ram0_we_b;
output [3:0] ram0_bwe_b;

inout  [35:0] ram1_data;
output [18:0] ram1_address;
output ram1_adv_ld, ram1_clk, ram1_cen_b, ram1_ce_b, ram1_oe_b, ram1_we_b;
output [3:0] ram1_bwe_b;

input  clock_feedback_in;
output clock_feedback_out;

inout  [15:0] flash_data;
output [23:0] flash_address;
output flash_ce_b, flash_oe_b, flash_we_b, flash_reset_b, flash_byte_b;
input  flash_sts;

output rs232_txd, rs232_rts;
input  rs232_rxd, rs232_cts;

input  mouse_clock, mouse_data, keyboard_clock, keyboard_data;

input  clock_27mhz, clock1, clock2;

output disp_blank, disp_clock, disp_rs, disp_ce_b, disp_reset_b;
input  disp_data_in;
output  disp_data_out;

input  button0, button1, button2, button3, button_enter, button_right,
       button_left, button_down, button_up;
input  [7:0] switch;
output [7:0] led;

inout [31:0] user1, user2, user3, user4;

inout [43:0] daughtercard;

inout  [15:0] systemace_data;
```

```
   output [6:0]  systemace_address;
   output systemace_ce_b, systemace_we_b, systemace_oe_b;
   input  systemace_irq, systemace_mpbrdy;

   output [15:0] analyzer1_data, analyzer2_data, analyzer3_data,
                 analyzer4_data;
   output analyzer1_clock, analyzer2_clock, analyzer3_clock, analyzer4_clock;

   ////////////////////////////////////////////////////////////////////////////
   //
   // I/O Assignments
   //
   ////////////////////////////////////////////////////////////////////////////

   // Audio Input and Output
   assign beep= 1'b0;
   assign audio_reset_b = 1'b0;
   assign ac97_synch = 1'b0;
   assign ac97_sdata_out = 1'b0;
/*
*/
   // ac97_sdata_in is an input

   // Video Output
   assign tv_out_ycrcb = 10'h0;
   assign tv_out_reset_b = 1'b0;
   assign tv_out_clock = 1'b0;
   assign tv_out_i2c_clock = 1'b0;
   assign tv_out_i2c_data = 1'b0;
   assign tv_out_pal_ntsc = 1'b0;
   assign tv_out_hsync_b = 1'b1;
   assign tv_out_vsync_b = 1'b1;
   assign tv_out_blank_b = 1'b1;
   assign tv_out_subcar_reset = 1'b0;

   // Video Input
   //assign tv_in_i2c_clock = 1'b0;
   assign tv_in_fifo_read = 1'b1;
   assign tv_in_fifo_clock = 1'b0;
   assign tv_in_iso = 1'b1;
   //assign tv_in_reset_b = 1'b0;
   assign tv_in_clock = clock_27mhz;//1'b0;
   //assign tv_in_i2c_data = 1'bZ;
   // tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2,
   // tv_in_aef, tv_in_hff, and tv_in_aff are inputs

   // SRAMs

/* change lines below to enable ZBT RAM bank0 */

/*
   assign ram0_data = 36'hZ;
   assign ram0_address = 19'h0;
   assign ram0_clk = 1'b0;
   assign ram0_we_b = 1'b1;
   assign ram0_cen_b = 1'b0;      // clock enable
*/

/* enable RAM pins */

   assign ram0_ce_b = 1'b0;
   assign ram0_oe_b = 1'b0;
   assign ram0_adv_ld = 1'b0;
   assign ram0_bwe_b = 4'h0;

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

   assign ram1_data = 36'hZ;
   assign ram1_address = 19'h0;
   assign ram1_adv_ld = 1'b0;
   assign ram1_clk = 1'b0;
   assign ram1_cen_b = 1'b1;
   assign ram1_ce_b = 1'b1;
   assign ram1_oe_b = 1'b1;
   assign ram1_we_b = 1'b1;
   assign ram1_bwe_b = 4'hF;

   //assign clock_feedback_out = 1'b0;
   // clock_feedback_in is an input

   // Flash ROM
   assign flash_data = 16'hZ;
   assign flash_address = 24'h0;
   assign flash_ce_b = 1'b1;
   assign flash_oe_b = 1'b1;
```

```
   assign flash_we_b = 1'b1;
   assign flash_reset_b = 1'b0;
   assign flash_byte_b = 1'b1;
   // flash_sts is an input

   // RS-232 Interface
   assign rs232_txd = 1'b1;
   assign rs232_rts = 1'b1;
   // rs232_rxd and rs232_cts are inputs

   // PS/2 Ports
   // mouse_clock, mouse_data, keyboard_clock, and keyboard_data are inputs

   // LED Displays
/*
   assign disp_blank = 1'b1;
   assign disp_clock = 1'b0;
   assign disp_rs = 1'b0;
   assign disp_ce_b = 1'b1;
   assign disp_reset_b = 1'b0;
   assign disp_data_out = 1'b0;
*/
   // disp_data_in is an input

   // Buttons, Switches, and Individual LEDs
   //lab3 assign led = 8'hFF;
   // button0, button1, button2, button3, button_enter, button_right,
   // button_left, button_down, button_up, and switches are inputs

   // User I/Os
   assign user1 = 32'hZ;
   assign user2 = 32'hZ;
   assign user3 = 32'hZ;
   assign user4 = 32'hZ;

   // Daughtercard Connectors
   assign daughtercard = 44'hZ;

   // SystemACE Microprocessor Port
   assign systemace_data = 16'hZ;
   assign systemace_address = 7'h0;
   assign systemace_ce_b = 1'b1;
   assign systemace_we_b = 1'b1;
   assign systemace_oe_b = 1'b1;
   // systemace_irq and systemace_mpbrdy are inputs

   // Logic Analyzer
   assign analyzer1_data = 16'h0;
   assign analyzer1_clock = 1'b1;
   assign analyzer2_data = 16'h0;
   assign analyzer2_clock = 1'b1;
   assign analyzer3_data = 16'h0;
   assign analyzer3_clock = 1'b1;
   assign analyzer4_data = 16'h0;
   assign analyzer4_clock = 1'b1;

   ////////////////////////////////////////////////////////////////////////
   // Demonstration of ZBT RAM as video memory

//   // use FPGA's digital clock manager to produce a
//   // 65MHz clock (actually 64.8MHz)
//   wire clock_65mhz_unbuf,clock_65mhz;
//   DCM vclk1(.CLKIN(clock_27mhz),.CLKFX(clock_65mhz_unbuf));
//   // synthesis attribute CLKFX_DIVIDE of vclk1 is 10
//   // synthesis attribute CLKFX_MULTIPLY of vclk1 is 24
//   // synthesis attribute CLK_FEEDBACK of vclk1 is NONE
//   // synthesis attribute CLKIN_PERIOD of vclk1 is 37
//   BUFG vclk2(.O(clock_65mhz),.I(clock_65mhz_unbuf));
//
//   wire clk = clock_65mhz;

      ////////////////////////////
      // Generate clock signals
      ////////////////////////////
      // use FPGA's digital clock manager to produce a
      // 40MHz clock (actually 39.86MHz)
      wire clock_40mhz_unbuf,clock_40mhz_unphased;
      DCM vclk1(.CLKIN(clock_27mhz),.CLKFX(clock_40mhz_unbuf));
      // synthesis attribute CLKFX_DIVIDE of vclk1 is 21
      // synthesis attribute CLKFX_MULTIPLY of vclk1 is 31
      // synthesis attribute CLK_FEEDBACK of vclk1 is NONE
      // synthesis attribute CLKIN_PERIOD of vclk1 is 37
      BUFG vclk2(.O(clock_40mhz_unphased),.I(clock_40mhz_unbuf));

      // Lock the clock to avoid timing delays
```

```
        wire clock_40mhz, locked;
//      ramclock rc (clock_40mhz_unphased, clock_40mhz, ram0_clk, ram1_clk,
//                                         clock_feedback_in, clock_feedback_out, locked);
        ramclock rc (clock_40mhz_unphased, clock_40mhz, ram0_clk,
                                           clock_feedback_in, clock_feedback_out, locked);

        wire clk = clock_40mhz;          // !!!!


   // power-on reset generation
   wire power_on_reset;    // remain high for first 16 clocks
   SRL16 reset_sr (.D(1'b0), .CLK(clk), .Q(power_on_reset),
                   .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
   defparam reset_sr.INIT = 16'hFFFF;

   // ENTER button is user reset
   wire reset,user_reset;
   debounce db1(power_on_reset, clk, ~button_enter, user_reset);
   assign reset = user_reset | power_on_reset;

   // display module for debugging

   reg [63:0] dispdata;
   display_16hex hexdisp1(reset, clk, dispdata,
                          disp_blank, disp_clock, disp_rs, disp_ce_b,
                          disp_reset_b, disp_data_out);

   // generate basic XVGA video signals
   wire [10:0] hcount;
   wire [9:0]  vcount;
   wire hsync,vsync,blank;
//   xvga xvga1(clk,hcount,vcount,hsync,vsync,blank);
        svga svga1(.vclock(clk),.hcount(hcount),.vcount(vcount),
                                        .hsync(hsync),.vsync(vsync),.blank(blank));

   // wire up to ZBT ram

   wire [35:0] vram_write_data;
   wire [35:0] vram_read_data;
   wire [18:0] vram_addr;
   wire        vram_we;


//      zbt_6111 zbt1(clock_40mhz, 1'b1, vram_we, vram_addr,
//       vram_write_data, vram_read_data,
//       ram0_clk, ram0_we_b, ram0_address, ram0_data, ram0_cen_b);

        // TAKEN FROM FINGER ART

        // Do not use zbt_6111 clock. Use fix in order to avoid timing
        // delays that cause ZBT error.

        // clock enable (should be synchronous and one cycle high at a time)
        assign ram0_cen_b = 0;

        // create delayed ram_we signal: note the delay is by two cycles!
        // ie we present the data to be written two cycles after we is raised
        // this means the bus is tri-stated two cycles after we is raised.

        reg [1:0] we0_delay;

        always @(posedge clk)
                we0_delay <= {we0_delay[0],vram_we};

        // create two-stage pipeline for write data

        reg [35:0] write_data0_old1;
        reg [35:0] write_data0_old2;
        always @(posedge clk)
                {write_data0_old2, write_data0_old1} <= {write_data0_old1, vram_write_data};

        // wire to ZBT RAM signals
        assign ram0_we_b = ~vram_we;
        assign ram0_address = vram_addr;
        assign ram0_data = we0_delay[1] ? write_data0_old2 : {36{1'bZ}};
        assign vram_read_data = ram0_data;


   // generate pixel value from reading ZBT memory
   wire [17:0]   vr_pixel;
   wire [18:0]   vram_addr1;

   vram_display vd1(reset,clk,hcount,vcount,vr_pixel,
                    vram_addr1,vram_read_data);
```

```
   // ADV7185 NTSC decoder interface code
   // adv7185 initialization module
   adv7185init adv7185(.reset(reset), .clock_27mhz(clock_27mhz),
                       .source(1'b0), .tv_in_reset_b(tv_in_reset_b),
                       .tv_in_i2c_clock(tv_in_i2c_clock),
                       .tv_in_i2c_data(tv_in_i2c_data));

   wire [29:0] ycrcb;     // video data (luminance, chrominance)
   wire [2:0] fvh;        // sync for field, vertical, horizontal
   wire       dv;         // data valid

   ntsc_decode decode (.clk(tv_in_line_clock1), .reset(reset),
                       .tv_in_ycrcb(tv_in_ycrcb[19:10]),
                       .ycrcb(ycrcb), .f(fvh[2]),
                       .v(fvh[1]), .h(fvh[0]), .data_valid(dv));

         // convert ycrcb to rgb

         wire [23:0] rgb; // video data (red, green, blue)

         //module YCrCb2RGB ( R, G, B, clk, rst, Y, Cr, Cb );
         YCrCb2RGB ycrcb2rgb( .R(rgb[23:16]), .G(rgb[15:8]), .B(rgb[7:0]), .clk(tv_in_line_clock1), .rst(reset),
                                                                  .Y(ycrcb[29:20]), .Cr(ycrcb[19:10]),
.Cb(ycrcb[9:0]));



   // code to write NTSC data to video memory

   wire [18:0] ntsc_addr;
   wire [35:0] ntsc_data;
   wire        ntsc_we;
         //module ntsc_to_zbt(clk, vclk, fvh, dv, din, ntsc_addr, ntsc_data, ntsc_we, sw);
   ntsc_to_zbt n2z (clk, tv_in_line_clock1, fvh, dv,
                                                        {rgb[23:18], rgb[15:10], rgb[7:2]},               //
ycrcb[29:22] for b&w
                                                        ntsc_addr, ntsc_data, ntsc_we, switch[6]);

   // code to write pattern to ZBT memory
   reg [31:0]    count;
   always @(posedge clk) count <= reset ? 0 : count + 1;

   wire [18:0]   vram_addr2 = count[0+18:0];
   wire [35:0]   vpat = ( switch[1] ? {4{count[3+3:3],4'b0}}
                          : {4{count[3+4:4],4'b0}} );

   // mux selecting read/write to memory based on which write-enable is chosen

   wire  sw_ntsc = ~switch[7];
   //wire        my_we = sw_ntsc ? (hcount[1:0]==2'd2) : blank;             // b&w
         wire    my_we = sw_ntsc ? (hcount[0]==1'd1) : blank;
   wire [18:0]   write_addr = sw_ntsc ? ntsc_addr : vram_addr2;
   wire [35:0]   write_data = sw_ntsc ? ntsc_data : vpat;

//   wire        write_enable = sw_ntsc ? (my_we & ntsc_we) : my_we;
//   assign      vram_addr = write_enable ? write_addr : vram_addr1;
//   assign      vram_we = write_enable;

   assign        vram_addr = my_we ? write_addr : vram_addr1;
   assign        vram_we = my_we;
   assign        vram_write_data = write_data;


         // manipulation of pixels read from ZBT

         // convert rgb to hsv for current pixel

         wire [13:0] H_vr_pixel;
         wire [11:0] S_vr_pixel;
         wire [5:0] V_vr_pixel;
         RGB2HSV rgb2hsv(.clk(clk), .reset(reset), .R(vr_pixel[17:12]), .G(vr_pixel[11:6]), .B(vr_pixel[5:0]),
                                                        .H(H_vr_pixel), .S(S_vr_pixel), .V(V_vr_pixel));

         // find hands, color them green

         wire [13:0] Hdes_min = 14'h0F;    //
         wire [13:0] Hdes_max = 14'hD9;
         wire isHand;
         HandFinder handfinder(.clk(clk), .reset(reset), .Hdes_min(Hdes_min), .Hdes_max(Hdes_max),
.H_pixel(H_vr_pixel), .V(V_vr_pixel), .isHand(isHand));

//       module LFCoord(
//
//input clk, reset;
//input [10:0] hcount;
```

```
//input [9:0] vcount;
//input [8:0] left_width;
//input [5:0] finger_width;
//input isHand;
//output [10:0] lf_x;
//output [9:0] lf_y;
//
//);

        // get scaled (x,y) coords of left and right fingers

        wire [10:0] lf_x;
        wire [9:0] lf_y;
        localparam left_width = 150;
        wire [5:0] finger_width = 14;
        LFCoord lf(.clk(clk), .reset(reset), .hcount(hcount), .vcount(vcount), .left_width(left_width),
                                        .finger_width(finger_width), .isHand(isHand), .lf_x_current(lf_x),
.lf_y_current(lf_y));

        wire [10:0] rf_x;
        wire [9:0] rf_y;
        RFCoord rf(.clk(clk), .reset(reset), .hcount(hcount), .vcount(vcount), .left_width(left_width),
                                        .finger_width(finger_width), .isHand(isHand), .rf_x_current(rf_x),
.rf_y_current(rf_y));


/*

        // find HSV of middle pixel, display on hex

        wire [13:0] middle_H;
        wire [11:0] middle_S;
        wire [5:0] middle_V;
        wire [17:0] crosshair_pixel;
        FindHSV findhsv(.clk(clk), .reset(reset), .hcount(hcount), .vcount(vcount),
                                                .R(vr_pixel[17:12]), .G(vr_pixel[11:6]), .B(vr_pixel[5:0]),
                                                .H(middle_H), .S(middle_S), .V(middle_V), .pixel(crosshair_pixel));

*/


        reg [10:0] lf_xx;
        reg [9:0] lf_yy;
        reg [10:0] rf_xx;
        reg [9:0] rf_yy;

        // DEBUGGING
        always @(posedge clk) begin
                lf_xx <= switch[2] ? 100 : lf_x;
                lf_yy <= switch[2] ? 400 : lf_y;
                rf_xx <= switch[2] ? 600 : rf_x;
                rf_yy <= switch[2] ? 400 : rf_y;
        end




        wire [1:0] option;
        wire show_rectangle;
        wire crop;
        wire [10:0] crop_start_x;
        wire [9:0] crop_start_y;
        wire [10:0] crop_end_x;
        wire [9:0] crop_end_y;
        wire [10:0] crop_end_xx; // adjusted so that can't be outside borders of image
        wire [9:0] crop_end_yy;
        wire [10:0] ref_pt_x;
        wire [9:0] ref_pt_y;
        wire [10:0] new_pt_x;
        wire [9:0] new_pt_y;
        wire [2:0] state;
        wire [23:0] rectangle_pixel;
        wire done_cropping;
        wire done_rotating;
        wire [23:0] crop_pixel;
        wire [10:0] image_x;
        wire [9:0] image_y;
        wire [23:0] blob_pixel;
        wire [23:0] finger_pixel;
        wire [10:0] crop_image_width = crop_end_x - crop_start_x;
        wire [9:0] crop_image_height = crop_end_y - crop_start_y;
        wire [10:0] rotate_image_width;
        wire [9:0] rotate_image_height;
        wire [10:0] image_width;
        wire [9:0] image_height;
```

```
        wire [2:0] corner1;
        wire [2:0] corner2;
        wire [23:0] corner1_color = {8'd255, 8'd255, 8'd255};                       //24'hFF9900;
        wire [23:0] corner2_color = {8'd255, 8'd255, 8'd0};
        wire [23:0] corner1_pixel;
        wire [23:0] corner2_pixel;
        wire show_corners;
        wire rotate;
        wire start_timer;
//      wire waiting;
        // TIMER for RESTORE
        wire oneHZ_enable;
        wire [3:0] time_value = 3;                // seconds
        wire expired;
        wire [3:0] counter;
        wire restored;



        // OPTION FINDER
        OptionFinder optionfind(clk, reset, lf_xx, lf_yy, option);


        // FSM!
        FSM fsm(clk, reset, hcount, vcount, rf_xx, rf_yy, image_x, image_y, image_width,
        image_height, option, done_cropping, done_rotating, expired, show_rectangle, crop, rotate,
        crop_start_x, crop_start_y, crop_end_x, crop_end_y, ref_pt_x, ref_pt_y, new_pt_x,
        new_pt_y, show_corners, start_timer, state);

        // RECTANGLE DRAWER
        //DrawRectangle drawrect(clk, reset, hcount, vcount, show_rectangle, crop_start_x, crop_start_y, rf_xx, rf_yy,
image_x, image_y, image_width, image_height, rectangle_pixel);
        //DrawRectangle drawrect(clk, reset, hcount, vcount, show_rectangle, crop_start_x, crop_start_y, crop_end_x,
crop_end_y, image_x, image_y, image_width, image_height, crop_end_xx, crop_end_yy, rectangle_pixel);
        //DrawRectangle drawrect(clk, reset, hcount, vcount, show_rectangle, crop_start_x, crop_start_y, rf_xx, rf_yy,
image_x, image_y, image_width, image_height, crop_end_xx, crop_end_yy, rectangle_pixel);

        DrawRectangle drawrect(
                                                        .clk(clk), .reset(reset), .hcount(hcount), .vcount(vcount),
.show_rectangle(show_rectangle),
                    .crop_start_x(crop_start_x), .crop_start_y(crop_start_y),
                                                        .rf_x(rf_xx), .rf_y(rf_yy), .image_x(image_x),
.image_y(image_y), .image_width(image_width), .image_height(image_height),
                                                        .crop_end_xx(crop_end_xx), .crop_end_yy(crop_end_yy),
.rectangle_pixel(rectangle_pixel));

        // CROPPER
        Cropper cropper(.clk(clk), .reset(reset), .hcount(hcount), .vcount(vcount), .crop(crop),
        .crop_start_x(crop_start_x), .crop_start_y(crop_start_y), .crop_end_x(crop_end_x), .crop_end_y(crop_end_y),
        .image_x(image_x), .image_y(image_y), .image_width(image_width), .image_height(image_height),
        .done_cropping(done_cropping), .crop_pixel(crop_pixel));


        // F
        localparam screen_width = 800;
        localparam screen_height = 600;
        localparam right_width = screen_width - left_width;
        localparam right_x = 150;
        localparam F_width = 150;
        localparam F_height = 170;
        localparam image_width_initial = 500;
        localparam image_height_initial = 300;
        localparam image_x_initial = (right_x + (right_width - image_width_initial) / 2);
        localparam image_y_initial = (screen_height - image_height_initial) / 2;
        wire [10:0] F_x_initial = image_x_initial + (image_width_initial - F_width)/2;
        wire [9:0] F_y_initial = image_y_initial + (image_height_initial - F_height)/2;
        localparam F_color = 24'h00FF00;          // green

        wire [10:0] F_x;
        wire [9:0] F_y;
        wire [23:0] F_pixel;
        wire [23:0] image_pixel;

        wire [10:0] A_x;
        wire [9:0] A_y;
        wire [10:0] A_width;
        wire [9:0] A_height;
        wire [10:0] A_x_new;
        wire [9:0] A_y_new;
        wire [10:0] A_width_new;
        wire [9:0] A_height_new;
        wire [10:0] B_x;
        wire [9:0] B_y;
        wire [10:0] B_width;
        wire [9:0] B_height;
```

```
        wire [10:0] B_x_new;
        wire [9:0] B_y_new;
        wire [10:0] B_width_new;
        wire [9:0] B_height_new;
        wire [10:0] C_x;
        wire [9:0] C_y;
        wire [10:0] C_width;
        wire [9:0] C_height;
        wire [10:0] C_x_new;
        wire [9:0] C_y_new;
        wire [10:0] C_width_new;
        wire [9:0] C_height_new;



        // Draw letter F in whatever rotation it's supposed to be in
        DrawF f(.clk(clk), .reset(reset), .hcount(hcount), .vcount(vcount),
                                          .F_x_initial(F_x_initial), .F_y_initial(F_y_initial),
                                          .A_x_new(A_x_new), .A_y_new(A_y_new), .A_width_new(A_width_new),
.A_height_new(A_height_new),
                                          .B_x_new(B_x_new), .B_y_new(B_y_new), .B_width_new(B_width_new),
.B_height_new(B_height_new),
                                          .C_x_new(C_x_new), .C_y_new(C_y_new), .C_width_new(C_width_new),
.C_height_new(C_height_new),
                                          .color(F_color), .expired(expired),
                                          .A_x(A_x), .A_y(A_y), .A_width(A_width), .A_height(A_height),
                                          .B_x(B_x), .B_y(B_y), .B_width(B_width), .B_height(B_height),
                                          .C_x(C_x), .C_y(C_y), .C_width(C_width), .C_height(C_height),
                                          .F_pixel(F_pixel));

        // change all parameters for F when rotate goes high
        Rotator rotator(.clk(clk), .reset(reset), .rotate(rotate), .corner1(corner1), .corner2(corner2),
                                          .A_x(A_x), .A_y(A_y), .A_width(A_width), .A_height(A_height),
                                          .B_x(B_x), .B_y(B_y), .B_width(B_width), .B_height(B_height),
                                          .C_x(C_x), .C_y(C_y), .C_width(C_width), .C_height(C_height),
                                          .image_x(image_x), .image_y(image_y), .image_width(image_width),
.image_height(image_height),
                                          .A_x_new(A_x_new), .A_y_new(A_y_new), .A_width_new(A_width_new),
.A_height_new(A_height_new),
                                          .B_x_new(B_x_new), .B_y_new(B_y_new), .B_width_new(B_width_new),
.B_height_new(B_height_new),
                                          .C_x_new(C_x_new), .C_y_new(C_y_new), .C_width_new(C_width_new),
.C_height_new(C_height_new),
                                          .image_width_new(rotate_image_width),
.image_height_new(rotate_image_height), .done_rotating(done_rotating));


        // Draw initial display w/ options, labels, 'image'
        // blob_pixel = everything but image!
        BlobGenerator blobgen(clk, reset, hcount, vcount,
                                                                hsync, vsync, blank, crop,
                                                                crop_start_x, crop_start_y,
                                                                crop_end_x, crop_end_y,
                                                                crop_image_width, crop_image_height,
                                                                rotate_image_width, rotate_image_height,
                                                                done_rotating, expired,
                                                                image_width, image_height,
                                                                image_x, image_y, image_pixel, blob_pixel,
restored);

//      module BlobGenerator(input vclock, input reset, input [10:0] hcount, input [9:0] vcount,
//                                                          input hsync, input vsync, input blank,    input [10:0]
new_image_x, input [9:0] new_image_y,
//                                                          input [10:0] crop_end_x, input [9:0] crop_end_y,
//                                                          input [10:0] crop_image_width, input [9:0]
crop_image_height,
//                                                          input [10:0] rotate_image_width, input [9:0]
rotate_image_height,
//                                                          input done_rotating, output reg [10:0] image_width, output
reg [9:0] image_height,
//                                                          output [10:0] image_x, output [9:0] image_y, output reg
[23:0] pixel);


        // finger markers
        DrawFingers drawfingers(clk, reset, hcount, vcount, lf_xx, lf_yy, rf_xx, rf_yy, finger_pixel);

        // find corner RF is closest to
//      CornerFinder findcorner(clk, reset, rf_xx, rf_yy, ref_pt_x, ref_pt_y, new_pt_x, new_pt_y, show_corners,
corner1, corner2);
        Corner1Finder findcorner1(clk, reset, rf_xx, rf_yy, ref_pt_x, ref_pt_y, show_corners, corner1);
        Corner2Finder findcorner2(clk, reset, rf_xx, rf_yy, new_pt_x, new_pt_y, show_corners, corner2);

        // draw corner RF is nearest to
//      DrawCorner drawcorner(clk, reset, hcount, vcount, corner1, corner2, corner1_color, corner2_color, image_x,
```

```
image_y, image_width, image_height, corner1_pixel, corner2_pixel);
        DrawCorner1 drawcorner1(clk, reset, hcount, vcount, corner1, corner1_color, image_x, image_y, image_width,
image_height, corner1_pixel);
        DrawCorner2 drawcorner2(clk, reset, hcount, vcount, corner2, corner2_color, image_x, image_y, image_width,
image_height, corner2_pixel);



        timer timer1(.clk(clk), .reset(reset), .Time_Value(time_value), .Start_Timer(start_timer),
.oneHZ_enable(oneHZ_enable),
                                                .Expired(expired), .counter(counter));

//      module timer
//(
//       input clk,
//       input [3:0] Time_Value,
//    input Start_Timer,
//    input oneHZ_enable,
//    output reg Expired,
//       output reg [3:0] counter
//);


        divider divider1(.clk(clk), .Start_Timer(start_timer), .oneHZ_enable(oneHZ_enable));

//module divider
//(
//       input clk,
//       input debug_on,
//    input Start_Timer,
//    output oneHZ_enable,
//       output reg blink
//);


        // for DEBUGGING display
        //wire [17:0] pixel_with_crosshair_and_hand = isHand ? ({6'b0, 6'd255, 6'b0} | lf_pixel) : (vr_pixel |
crosshair_pixel | lf_pixel);
        wire [17:0] pixel_with_hand = isHand ? {6'd0, 6'd63, 6'd0} : vr_pixel;


//      localparam image_width = 500;
//      localparam image_height = 300;
        wire inside_image = ((rf_xx>=image_x & rf_xx<=(image_x+image_width)) & (rf_yy>=image_y &
rf_yy<=(image_y+image_height)));

        // display on hex16
        // rf_x, rf_y for now
        always @(posedge clk)
          dispdata <= switch[5] ? {rotate_image_width, 6'b0, rotate_image_height} : switch[4] ? {A_x, 6'b0, A_y, 5'b0,
A_width, 6'b0, A_height} : switch[1] ? {inside_image, 6'b0, image_x, 6'b0, image_y, 7'b0, option, reset,
show_rectangle, done_cropping, crop, 1'b0, state} : {lf_xx, 8'b0, lf_yy, 8'b0, rf_xx, 8'b0, rf_yy};
     //dispdata <= {rf_pixel, 8'b0, H_vr_pixel, 8'b0, middle_H};

        // if hand pixel, white, else just whatever it should be
        //wire [23:0] rgb_with_hands = isHand ? 24'hFFFFFF : rgb;


   // select output pixel data

   //reg [7:0]   pixel;           // b&w
        reg [23:0] pixel;
   wire  b,hs,vs;

   delayN dn1(clk,hsync,hs);      // delay by 3 cycles to sync with ZBT read
   delayN dn2(clk,vsync,vs);
   delayN dn3(clk,blank,b);


        reg [23:0] total_pixel;
        always @(posedge clk) begin
                if (finger_pixel!=0)
                        total_pixel <= finger_pixel;
                else if (rectangle_pixel!=0)
                        total_pixel <= rectangle_pixel;
                else if (crop_pixel!=0)
                        total_pixel <= crop_pixel;
                else if (image_pixel !=0)
                        total_pixel <= (image_pixel | F_pixel);
                else
                        total_pixel <= blob_pixel;
        end


   always @(posedge clk)
```

```
     begin
                          pixel <= switch[0]
//                                      ? {hcount[8:6],5'b0,hcount[8:6],5'b0,hcount[8:6],5'b0}             // lines
                                        ? (total_pixel | corner1_pixel | corner2_pixel)
                                        //? (blob_pixel | rectangle_pixel | crop_pixel)            // blob_pixel =
initial display + finger markers
                                        : switch[4]
                                        ? F_pixel                // F
                                        : switch[3]
                                        ? {vr_pixel[17:12], 2'b0, vr_pixel[11:6], 2'b0, vr_pixel[5:0], 2'b0}
         // normal color camera image
                                        : {pixel_with_hand[17:12],2'b0, pixel_with_hand[11:6],2'b0,
                                        pixel_with_hand[5:0],2'b0};              // debugging display
     end

   // VGA Output.  In order to meet the setup and hold times of the
   // AD7125, we send it ~clock_65mhz.
   assign vga_out_red = pixel[23:16];
   assign vga_out_green = pixel[15:8];
   assign vga_out_blue = pixel[7:0];
   assign vga_out_sync_b = 1'b1;    // not used
//   assign vga_out_pixel_clock = ~clock_65mhz;
        assign vga_out_pixel_clock = ~clk;
   assign vga_out_blank_b = ~b;
   assign vga_out_hsync = hs;
   assign vga_out_vsync = vs;

   // debugging

//   assign led[7:1] = ~{vram_addr[18:13],reset};
//       assign led[7:3] = ~{vram_addr[18:14]};
         assign led[7] = ~expired;
         assign led[6] = oneHZ_enable;
         assign led[5] = 1;
         assign led[4] = ~done_rotating;
         assign led[3] = ~rotate;
         assign led[2] = ~done_cropping;
         assign led[1] = ~crop;
         assign led[0] = ~isHand;

   //always @(posedge clk)
     //dispdata <= {vram_read_data,9'b0,vram_addr};                // ??
     //dispdata <= {ntsc_data,9'b0,ntsc_addr};

endmodule

//////////////////////////////////////////////////////////////////////////////
// xvga: Generate XVGA display signals (1024 x 768 @ 60Hz)

module xvga(vclock,hcount,vcount,hsync,vsync,blank);
   input vclock;
   output [10:0] hcount;
   output [9:0] vcount;
   output        vsync;
   output        hsync;
   output        blank;

   reg    hsync,vsync,hblank,vblank,blank;
   reg [10:0]     hcount;    // pixel number on current line
   reg [9:0] vcount;      // line number

   // horizontal: 1344 pixels total
   // display 1024 pixels per line
   wire      hsyncon,hsyncoff,hreset,hblankon;
   assign    hblankon = (hcount == 1023);
   assign    hsyncon = (hcount == 1047);
   assign    hsyncoff = (hcount == 1183);
   assign    hreset = (hcount == 1343);

   // vertical: 806 lines total
   // display 768 lines
   wire      vsyncon,vsyncoff,vreset,vblankon;
   assign    vblankon = hreset & (vcount == 767);
   assign    vsyncon = hreset & (vcount == 776);
   assign    vsyncoff = hreset & (vcount == 782);
   assign    vreset = hreset & (vcount == 805);

   // sync and blanking
   wire      next_hblank,next_vblank;
   assign next_hblank = hreset ? 0 : hblankon ? 1 : hblank;
   assign next_vblank = vreset ? 0 : vblankon ? 1 : vblank;
   always @(posedge vclock) begin
      hcount <= hreset ? 0 : hcount + 1;
      hblank <= next_hblank;
      hsync <= hsyncon ? 0 : hsyncoff ? 1 : hsync;  // active low
```

```
      vcount <= hreset ? (vreset ? 0 : vcount + 1) : vcount;
      vblank <= next_vblank;
      vsync <= vsyncon ? 0 : vsyncoff ? 1 : vsync;  // active low

      blank <= next_vblank | (next_hblank & ~hreset);
   end
endmodule

//////////////////////////////////////////////////////////////////////////
// generate display pixels from reading the ZBT ram
// note that the ZBT ram has 2 cycles of read (and write) latency
//
// We take care of that by latching the data at an appropriate time.
//
// Note that the ZBT stores 36 bits per word; we use only 32 bits here,
// decoded into four bytes of pixel data.

module vram_display(reset,clk,hcount,vcount,vr_pixel,
                    vram_addr,vram_read_data);

   input reset, clk;
   input [10:0] hcount;
   input [9:0]   vcount;
   //output [7:0] vr_pixel;               // b&w
         output [17:0] vr_pixel;
   output [18:0] vram_addr;
   input [35:0]  vram_read_data;

//
//       // modified code borrowed from Virtual Juggling 2005)
//       parameter HMID = 9'd367; // The horizontal center of the image in MEMORY
//       parameter HSTART = HMID-9'd256; // The horizontal counter decrements!!!
//       parameter VMID = 9'd287; // The vertical center of the image in MEMORY
//       parameter VSTART = VMID-9'd192;
//
//       wire [18:0] vram_addr = {1'b0,vcount[9:1]+VSTART, ~hcount[10:2]-9'd180};
//       wire [1:0] hc4 = hcount[1:0];
//       reg [17:0] vr_pixel;
//       reg [35:0] vr_data_latched;
//       reg [35:0] last_vr_data;
//
//       always @(posedge clk)
//               last_vr_data <= (hc4==2'd3) ? vr_data_latched : last_vr_data;
//       always @(posedge clk)
//               vr_data_latched <= (hc4==2'd1) ? vram_read_data : vr_data_latched;
//
//       always @* // each 36-bit word from RAM is decoded to 4 bytes
//               case (hc4)
//                       2'd3: vr_pixel = last_vr_data[17:0];//last_vr_data[8:0];
//                       2'd2: vr_pixel = last_vr_data[17:0];//last_vr_data[8+9:0+9];
//                       2'd1: vr_pixel = last_vr_data[35:18];//last_vr_data[8+18:0+18];
//                       2'd0: vr_pixel = last_vr_data[35:18];//last_vr_data[8+27:0+27];
//       endcase



   //wire [18:0]          vram_addr = {1'b0, vcount, hcount[9:2]};          // b&w
         wire [18:0]       vram_addr = {vcount, hcount[9:1]};

   //wire [1:0]   hc4 = hcount[1:0];              // b&w
         wire hc2 = hcount[0];
   //reg [7:0]    vr_pixel;                               // b&w
         reg [17:0]        vr_pixel;
   reg [35:0]     vr_data_latched;
   reg [35:0]     last_vr_data;

   always @(posedge clk)
     //last_vr_data <= (hc4==2'd3) ? vr_data_latched : last_vr_data;                // b&w
          last_vr_data <= (hc2) ? vr_data_latched : last_vr_data;

   always @(posedge clk)
     //vr_data_latched <= (hc4==2'd1) ? vram_read_data : vr_data_latched;   // b&w
          vr_data_latched <= (!hc2) ? vram_read_data : vr_data_latched;

//   always @*           // each 36-bit word from RAM is decoded to 4 bytes
//     case (hc4)
//       2'd3: vr_pixel = last_vr_data[7:0];
//       2'd2: vr_pixel = last_vr_data[7+8:0+8];
//       2'd1: vr_pixel = last_vr_data[7+16:0+16];
//       2'd0: vr_pixel = last_vr_data[7+24:0+24];
//     endcase

         always @* // each 36-bit word from RAM is decoded to 4 bytes
                 case (hc2)
```

```
                        1'd1: vr_pixel = last_vr_data[17:0];
                        1'd0: vr_pixel = last_vr_data[17+18:0+18];
                endcase

endmodule // vram_display

//////////////////////////////////////////////////////////////////////////
// parameterized delay line

module delayN(clk,in,out);
   input clk;
   input in;
   output out;

   parameter NDELAY = 3;

   reg [NDELAY-1:0] shiftreg;
   wire     out = shiftreg[NDELAY-1];

   always @(posedge clk)
     shiftreg <= {shiftreg[NDELAY-2:0],in};

endmodule // delayN




module FindHSV(clk, reset, hcount, vcount, R, G, B, H, S, V, pixel);

        input clk, reset;
        input [10:0] hcount;
        input [9:0] vcount;
        input [5:0] R, G, B;
        output reg [13:0] H;
        output reg [11:0] S;
        output reg [5:0] V;
        output [17:0] pixel;

        // draw crosshairs

        localparam screen_width = 800;
        localparam screen_height = 600;

        wire [17:0] vert_pixel;
        wire [17:0] horiz_pixel;
        localparam vert_x = (screen_width) / 2;
        localparam horiz_y = (screen_height) / 2;

        //assign pixel = (vert_pixel | horiz_pixel);              // don't draw this crosshair for now

        blob #(.WIDTH(1),.HEIGHT(screen_height),.COLOR(18'd262143))   // white
                  vertical(.x(vert_x),.y(0),.hcount(hcount),.vcount(vcount),
                                   .pixel(vert_pixel));

        blob #(.WIDTH(screen_width),.HEIGHT(1),.COLOR(18'd262143))   // white
                  horizontal(.x(0),.y(horiz_y),.hcount(hcount),.vcount(vcount),
                                   .pixel(horiz_pixel));

        wire [13:0] center_H;
        wire [11:0] center_S;
        wire [5:0] center_V;

        RGB2HSV rgb2hsv1(clk, reset, R, G, B, center_H, center_S, center_V);


        // convert center pixel RGB to HSV & display values on hex

        always @(posedge clk) begin
                if (reset) begin
                        H <= 0;
                        S <= 0;
                        V <= 0;
                end
                // if at middle pixel, find HSV           // actually 1 over to left, 1 up of middle
                if ((hcount == 299) && (vcount == 399)) begin
                        H <= center_H;
                        S <= center_S;
                        V <= center_V;
                end
//              else begin
//                      H <= 0;
//                      S <= 0;
//                      V <= 0;
```

```
//              end
        end

endmodule

//////////////////////////////////////////////////////////////////////
//
// blob: generate rectangle on screen
//
//////////////////////////////////////////////////////////////////////
module blob
//      #(parameter WIDTH = 64,      // default width: 64 pixels
//                                       HEIGHT = 64,     // default height: 64 pixels
//                                       COLOR = 24'd16777215)  // default color: white
        (input [10:0] WIDTH, input [9:0] HEIGHT, input [23:0] COLOR, input [10:0] x,hcount,
         input [9:0] y,vcount,
         output reg [23:0] pixel);

        always @ (x or y or hcount or vcount) begin
                if ((hcount >= x && hcount < (x+WIDTH)) && (vcount >= y && vcount < (y+HEIGHT)))
                        pixel = COLOR;
                else pixel = 0;
        end
endmodule


//////////////////////////////////////////////////////////////////////
//
// circle: generate circle on screen
//
//////////////////////////////////////////////////////////////////////
module circle
//   #(parameter RADIUS = 32,     // default radius: 32 pixels
//               COLOR = 3'b111)  // default color: white
   (input [3:0] RADIUS,
         input [23:0] COLOR,
         input [10:0] x,hcount,
    input [9:0] y,vcount,
    output reg [23:0] pixel);

         wire [10:0] x_center;
         wire [9:0] y_center;
         reg [21:0] x_sq;
         reg [21:0] y_sq;
         reg [21:0] r_sq;

         assign x_center = x + RADIUS;
         assign y_center = y + RADIUS;

   always @ (x or y or hcount or vcount) begin

                x_sq <= (hcount-x_center)*(hcount-x_center);
                y_sq <= (vcount-y_center)*(vcount-y_center);
                r_sq <= RADIUS*RADIUS;

                if ((x_sq + y_sq) < r_sq)
                        pixel = COLOR;
      else pixel = 0;
   end
endmodule
```

video_decoder.v

```
//
// File:   video_decoder.v
// Date:   31-Oct-05
// Author: J. Castro (MIT 6.111, fall 2005)
//
// This file contains the ntsc_decode and adv7185init modules
//
// These modules are used to grab input NTSC video data from the RCA
// phono jack on the right hand side of the 6.111 labkit (connect
// the camera to the LOWER jack).
//

//////////////////////////////////////////////////////////////////////////////
//
// NTSC decode - 16-bit CCIR656 decoder
// By Javier Castro
// This module takes a stream of LLC data from the adv7185
```

```
// NTSC/PAL video decoder and generates the corresponding pixels,
// that are encoded within the stream, in YCrCb format.

// Make sure that the adv7185 is set to run in 16-bit LLC2 mode.

module ntsc_decode(clk, reset, tv_in_ycrcb, ycrcb, f, v, h, data_valid);

   // clk - line-locked clock (in this case, LLC1 which runs at 27Mhz)
   // reset - system reset
   // tv_in_ycrcb - 10-bit input from chip. should map to pins [19:10]
   // ycrcb - 24 bit luminance and chrominance (8 bits each)
   // f - field: 1 indicates an even field, 0 an odd field
   // v - vertical sync: 1 means vertical sync
   // h - horizontal sync: 1 means horizontal sync

   input clk;
   input reset;
   input [9:0] tv_in_ycrcb; // modified for 10 bit input - should be P[19:10]
   output [29:0] ycrcb;
   output        f;
   output        v;
   output        h;
   output        data_valid;
   // output [4:0] state;

   parameter     SYNC_1 = 0;
   parameter     SYNC_2 = 1;
   parameter     SYNC_3 = 2;
   parameter     SAV_f1_cb0 = 3;
   parameter     SAV_f1_y0 = 4;
   parameter     SAV_f1_cr1 = 5;
   parameter     SAV_f1_y1 = 6;
   parameter     EAV_f1 = 7;
   parameter     SAV_VBI_f1 = 8;
   parameter     EAV_VBI_f1 = 9;
   parameter     SAV_f2_cb0 = 10;
   parameter     SAV_f2_y0 = 11;
   parameter     SAV_f2_cr1 = 12;
   parameter     SAV_f2_y1 = 13;
   parameter     EAV_f2 = 14;
   parameter     SAV_VBI_f2 = 15;
   parameter     EAV_VBI_f2 = 16;



   // In the start state, the module doesn't know where
   // in the sequence of pixels, it is looking.

   // Once we determine where to start, the FSM goes through a normal
   // sequence of SAV process_YCrCb EAV... repeat

   // The data stream looks as follows
   // SAV_FF | SAV_00 | SAV_00 | SAV_XY | Cb0 | Y0 | Cr1 | Y1 | Cb2 | Y2 | ... | EAV sequence
   // There are two things we need to do:
   //   1. Find the two SAV blocks (stands for Start Active Video perhaps?)
   //   2. Decode the subsequent data

   reg [4:0]     current_state = 5'h00;
   reg [9:0]     y = 10'h000;  // luminance
   reg [9:0]     cr = 10'h000; // chrominance
   reg [9:0]     cb = 10'h000; // more chrominance

   assign        state = current_state;

   always @ (posedge clk)
     begin
        if (reset)
          begin

          end
        else
          begin
             // these states don't do much except allow us to know where we are in the stream.
             // whenever the synchronization code is seen, go back to the sync_state before
             // transitioning to the new state
             case (current_state)
               SYNC_1: current_state <= (tv_in_ycrcb == 10'h000) ? SYNC_2 : SYNC_1;
               SYNC_2: current_state <= (tv_in_ycrcb == 10'h000) ? SYNC_3 : SYNC_1;
               SYNC_3: current_state <= (tv_in_ycrcb == 10'h200) ? SAV_f1_cb0 :
                                        (tv_in_ycrcb == 10'h274) ? EAV_f1 :
                                        (tv_in_ycrcb == 10'h2ac) ? SAV_VBI_f1 :
                                        (tv_in_ycrcb == 10'h2d8) ? EAV_VBI_f1 :
                                        (tv_in_ycrcb == 10'h31c) ? SAV_f2_cb0 :
                                        (tv_in_ycrcb == 10'h368) ? EAV_f2 :
```

```
                                        (tv_in_ycrcb == 10'h3b0) ? SAV_VBI_f2 :
                                        (tv_in_ycrcb == 10'h3c4) ? EAV_VBI_f2 : SYNC_1;

                SAV_f1_cb0: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f1_y0;
                SAV_f1_y0: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f1_cr1;
                SAV_f1_cr1: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f1_y1;
                SAV_f1_y1: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f1_cb0;

                SAV_f2_cb0: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f2_y0;
                SAV_f2_y0: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f2_cr1;
                SAV_f2_cr1: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f2_y1;
                SAV_f2_y1: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f2_cb0;

                // These states are here in the event that we want to cover these signals
                // in the future. For now, they just send the state machine back to SYNC_1
                EAV_f1: current_state <= SYNC_1;
                SAV_VBI_f1: current_state <= SYNC_1;
                EAV_VBI_f1: current_state <= SYNC_1;
                EAV_f2: current_state <= SYNC_1;
                SAV_VBI_f2: current_state <= SYNC_1;
                EAV_VBI_f2: current_state <= SYNC_1;

              endcase
            end
      end // always @ (posedge clk)

   // implement our decoding mechanism

   wire y_enable;
   wire cr_enable;
   wire cb_enable;

   // if y is coming in, enable the register
   // likewise for cr and cb
   assign y_enable = (current_state == SAV_f1_y0) ||
                     (current_state == SAV_f1_y1) ||
                     (current_state == SAV_f2_y0) ||
                     (current_state == SAV_f2_y1);
   assign cr_enable = (current_state == SAV_f1_cr1) ||
                      (current_state == SAV_f2_cr1);
   assign cb_enable = (current_state == SAV_f1_cb0) ||
                      (current_state == SAV_f2_cb0);

   // f, v, and h only go high when active
   assign {v,h} = (current_state == SYNC_3) ? tv_in_ycrcb[7:6] : 2'b00;

   // data is valid when we have all three values: y, cr, cb
   assign data_valid = y_enable;
   assign ycrcb = {y,cr,cb};

   reg     f = 0;

   always @ (posedge clk)
     begin
        y <= y_enable ? tv_in_ycrcb : y;
        cr <= cr_enable ? tv_in_ycrcb : cr;
        cb <= cb_enable ? tv_in_ycrcb : cb;
        f <= (current_state == SYNC_3) ? tv_in_ycrcb[8] : f;
     end

endmodule



//////////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- ADV7185 Video Decoder Configuration Init
//
// Created:
// Author: Nathan Ickes
//
//////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////
// Register 0
//////////////////////////////////////////////////////////////////////////////////

`define INPUT_SELECT                            4'h0
  // 0: CVBS on AIN1 (composite video in)
  // 7: Y on AIN2, C on AIN5 (s-video in)
  // (These are the only configurations supported by the 6.111 labkit hardware)
`define INPUT_MODE                              4'h0
  // 0: Autodetect: NTSC or PAL (BGHID), w/o pedestal
  // 1: Autodetect: NTSC or PAL (BGHID), w/pedestal
  // 2: Autodetect: NTSC or PAL (N), w/o pedestal
```

```
  // 3: Autodetect: NTSC or PAL (N), w/pedestal
  // 4: NTSC w/o pedestal
  // 5: NTSC w/pedestal
  // 6: NTSC 4.43 w/o pedestal
  // 7: NTSC 4.43 w/pedestal
  // 8: PAL BGHID w/o pedestal
  // 9: PAL N w/pedestal
  // A: PAL M w/o pedestal
  // B: PAL M w/pedestal
  // C: PAL combination N
  // D: PAL combination N w/pedestal
  // E-F: [Not valid]

`define ADV7185_REGISTER_0 {`INPUT_MODE, `INPUT_SELECT}

////////////////////////////////////////////////////////////////////////////////
// Register 1
////////////////////////////////////////////////////////////////////////////////

`define VIDEO_QUALITY                           2'h0
  // 0: Broadcast quality
  // 1: TV quality
  // 2: VCR quality
  // 3: Surveillance quality
`define SQUARE_PIXEL_IN_MODE                    1'b0
  // 0: Normal mode
  // 1: Square pixel mode
`define DIFFERENTIAL_INPUT                      1'b0
  // 0: Single-ended inputs
  // 1: Differential inputs
`define FOUR_TIMES_SAMPLING                     1'b0
  // 0: Standard sampling rate
  // 1: 4x sampling rate (NTSC only)
`define BETACAM                                 1'b0
  // 0: Standard video input
  // 1: Betacam video input
`define AUTOMATIC_STARTUP_ENABLE                1'b1
  // 0: Change of input triggers reacquire
  // 1: Change of input does not trigger reacquire

`define ADV7185_REGISTER_1 {`AUTOMATIC_STARTUP_ENABLE, 1'b0, `BETACAM, `FOUR_TIMES_SAMPLING, `DIFFERENTIAL_INPUT,
`SQUARE_PIXEL_IN_MODE, `VIDEO_QUALITY}

////////////////////////////////////////////////////////////////////////////////
// Register 2
////////////////////////////////////////////////////////////////////////////////

`define Y_PEAKING_FILTER                        3'h4
  // 0: Composite =  4.5dB,  s-video =  9.25dB
  // 1: Composite =  4.5dB,  s-video =  9.25dB
  // 2: Composite =  4.5dB,  s-video =  5.75dB
  // 3: Composite =  1.25dB, s-video =  3.3dB
  // 4: Composite =  0.0dB,  s-video =  0.0dB
  // 5: Composite = -1.25dB, s-video = -3.0dB
  // 6: Composite = -1.75dB, s-video = -8.0dB
  // 7: Composite = -3.0dB,  s-video = -8.0dB
`define CORING                                  2'h0
  // 0: No coring
  // 1: Truncate if Y < black+8
  // 2: Truncate if Y < black+16
  // 3: Truncate if Y < black+32

`define ADV7185_REGISTER_2 {3'b000, `CORING, `Y_PEAKING_FILTER}

////////////////////////////////////////////////////////////////////////////////
// Register 3
////////////////////////////////////////////////////////////////////////////////

`define INTERFACE_SELECT                        2'h0
  // 0: Philips-compatible
  // 1: Broktree API A-compatible
  // 2: Broktree API B-compatible
  // 3: [Not valid]
`define OUTPUT_FORMAT                           4'h0
  // 0: 10-bit @ LLC, 4:2:2 CCIR656
  // 1: 20-bit @ LLC, 4:2:2 CCIR656
  // 2: 16-bit @ LLC, 4:2:2 CCIR656
  // 3: 8-bit @ LLC, 4:2:2 CCIR656
  // 4: 12-bit @ LLC, 4:1:1
  // 5-F: [Not valid]
  // (Note that the 6.111 labkit hardware provides only a 10-bit interface to
  // the ADV7185.)
`define TRISTATE_OUTPUT_DRIVERS                 1'b0
  // 0: Drivers tristated when ~OE is high
  // 1: Drivers always tristated
```

```
`define VBI_ENABLE                              1'b0
  // 0: Decode lines during vertical blanking interval
  // 1: Decode only active video regions

`define ADV7185_REGISTER_3 {`VBI_ENABLE, `TRISTATE_OUTPUT_DRIVERS, `OUTPUT_FORMAT, `INTERFACE_SELECT}

////////////////////////////////////////////////////////////////////////////////
// Register 4
////////////////////////////////////////////////////////////////////////////////

`define OUTPUT_DATA_RANGE                       1'b0
  // 0: Output values restricted to CCIR-compliant range
  // 1: Use full output range
`define BT656_TYPE                              1'b0
  // 0: BT656-3-compatible
  // 1: BT656-4-compatible

`define ADV7185_REGISTER_4 {`BT656_TYPE, 3'b000, 3'b110, `OUTPUT_DATA_RANGE}

////////////////////////////////////////////////////////////////////////////////
// Register 5
////////////////////////////////////////////////////////////////////////////////


`define GENERAL_PURPOSE_OUTPUTS                 4'b0000
`define GPO_0_1_ENABLE                          1'b0
  // 0: General purpose outputs 0 and 1 tristated
  // 1: General purpose outputs 0 and 1 enabled
`define GPO_2_3_ENABLE                          1'b0
  // 0: General purpose outputs 2 and 3 tristated
  // 1: General purpose outputs 2 and 3 enabled
`define BLANK_CHROMA_IN_VBI                     1'b1
  // 0: Chroma decoded and output during vertical blanking
  // 1: Chroma blanked during vertical blanking
`define HLOCK_ENABLE                            1'b0
  // 0: GPO 0 is a general purpose output
  // 1: GPO 0 shows HLOCK status

`define ADV7185_REGISTER_5 {`HLOCK_ENABLE, `BLANK_CHROMA_IN_VBI, `GPO_2_3_ENABLE, `GPO_0_1_ENABLE,
`GENERAL_PURPOSE_OUTPUTS}

////////////////////////////////////////////////////////////////////////////////
// Register 7
////////////////////////////////////////////////////////////////////////////////

`define FIFO_FLAG_MARGIN                        5'h10
  // Sets the locations where FIFO almost-full and almost-empty flags are set
`define FIFO_RESET                              1'b0
  // 0: Normal operation
  // 1: Reset FIFO. This bit is automatically cleared
`define AUTOMATIC_FIFO_RESET                    1'b0
  // 0: No automatic reset
  // 1: FIFO is autmatically reset at the end of each video field
`define FIFO_FLAG_SELF_TIME                     1'b1
  // 0: FIFO flags are synchronized to CLKIN
  // 1: FIFO flags are synchronized to internal 27MHz clock

`define ADV7185_REGISTER_7 {`FIFO_FLAG_SELF_TIME, `AUTOMATIC_FIFO_RESET, `FIFO_RESET, `FIFO_FLAG_MARGIN}

////////////////////////////////////////////////////////////////////////////////
// Register 8
////////////////////////////////////////////////////////////////////////////////

`define INPUT_CONTRAST_ADJUST                       8'h80

`define ADV7185_REGISTER_8 {`INPUT_CONTRAST_ADJUST}

////////////////////////////////////////////////////////////////////////////////
// Register 9
////////////////////////////////////////////////////////////////////////////////

`define INPUT_SATURATION_ADJUST                     8'h8C

`define ADV7185_REGISTER_9 {`INPUT_SATURATION_ADJUST}

////////////////////////////////////////////////////////////////////////////////
// Register A
////////////////////////////////////////////////////////////////////////////////

`define INPUT_BRIGHTNESS_ADJUST                     8'h00

`define ADV7185_REGISTER_A {`INPUT_BRIGHTNESS_ADJUST}

////////////////////////////////////////////////////////////////////////////////
// Register B
```

```
////////////////////////////////////////////////////////////////////////////

`define INPUT_HUE_ADJUST                              8'h00

`define ADV7185_REGISTER_B {`INPUT_HUE_ADJUST}

////////////////////////////////////////////////////////////////////////////
// Register C
////////////////////////////////////////////////////////////////////////////

`define DEFAULT_VALUE_ENABLE                    1'b0
  // 0: Use programmed Y, Cr, and Cb values
  // 1: Use default values
`define DEFAULT_VALUE_AUTOMATIC_ENABLE          1'b0
  // 0: Use programmed Y, Cr, and Cb values
  // 1: Use default values if lock is lost
`define DEFAULT_Y_VALUE                         6'h0C
  // Default Y value

`define ADV7185_REGISTER_C {`DEFAULT_Y_VALUE, `DEFAULT_VALUE_AUTOMATIC_ENABLE, `DEFAULT_VALUE_ENABLE}

////////////////////////////////////////////////////////////////////////////
// Register D
////////////////////////////////////////////////////////////////////////////

`define DEFAULT_CR_VALUE                        4'h8
  // Most-significant four bits of default Cr value
`define DEFAULT_CB_VALUE                        4'h8
  // Most-significant four bits of default Cb value

`define ADV7185_REGISTER_D {`DEFAULT_CB_VALUE, `DEFAULT_CR_VALUE}

////////////////////////////////////////////////////////////////////////////
// Register E
////////////////////////////////////////////////////////////////////////////

`define TEMPORAL_DECIMATION_ENABLE              1'b0
  // 0: Disable
  // 1: Enable
`define TEMPORAL_DECIMATION_CONTROL             2'h0
  // 0: Supress frames, start with even field
  // 1: Supress frames, start with odd field
  // 2: Supress even fields only
  // 3: Supress odd fields only
`define TEMPORAL_DECIMATION_RATE                4'h0
  // 0-F: Number of fields/frames to skip

`define ADV7185_REGISTER_E {1'b0, `TEMPORAL_DECIMATION_RATE, `TEMPORAL_DECIMATION_CONTROL,
`TEMPORAL_DECIMATION_ENABLE}

////////////////////////////////////////////////////////////////////////////
// Register F
////////////////////////////////////////////////////////////////////////////

`define POWER_SAVE_CONTROL                      2'h0
  // 0: Full operation
  // 1: CVBS only
  // 2: Digital only
  // 3: Power save mode
`define POWER_DOWN_SOURCE_PRIORITY              1'b0
  // 0: Power-down pin has priority
  // 1: Power-down control bit has priority
`define POWER_DOWN_REFERENCE                    1'b0
  // 0: Reference is functional
  // 1: Reference is powered down
`define POWER_DOWN_LLC_GENERATOR                1'b0
  // 0: LLC generator is functional
  // 1: LLC generator is powered down
`define POWER_DOWN_CHIP                         1'b0
  // 0: Chip is functional
  // 1: Input pads disabled and clocks stopped
`define TIMING_REACQUIRE                        1'b0
  // 0: Normal operation
  // 1: Reacquire video signal (bit will automatically reset)
`define RESET_CHIP                              1'b0
  // 0: Normal operation
  // 1: Reset digital core and I2C interface (bit will automatically reset)

`define ADV7185_REGISTER_F {`RESET_CHIP, `TIMING_REACQUIRE, `POWER_DOWN_CHIP, `POWER_DOWN_LLC_GENERATOR,
`POWER_DOWN_REFERENCE, `POWER_DOWN_SOURCE_PRIORITY, `POWER_SAVE_CONTROL}

////////////////////////////////////////////////////////////////////////////
// Register 33
////////////////////////////////////////////////////////////////////////////
```

```
`define PEAK_WHITE_UPDATE                       1'b1
  // 0: Update gain once per line
  // 1: Update gain once per field
`define AVERAGE_BIRIGHTNESS_LINES               1'b1
  // 0: Use lines 33 to 310
  // 1: Use lines 33 to 270
`define MAXIMUM_IRE                             3'h0
  // 0: PAL: 133, NTSC: 122
  // 1: PAL: 125, NTSC: 115
  // 2: PAL: 120, NTSC: 110
  // 3: PAL: 115, NTSC: 105
  // 4: PAL: 110, NTSC: 100
  // 5: PAL: 105, NTSC: 100
  // 6-7: PAL: 100, NTSC: 100
`define COLOR_KILL                              1'b1
  // 0: Disable color kill
  // 1: Enable color kill

`define ADV7185_REGISTER_33 {1'b1, `COLOR_KILL, 1'b1, `MAXIMUM_IRE, `AVERAGE_BIRIGHTNESS_LINES, `PEAK_WHITE_UPDATE}

`define ADV7185_REGISTER_10 8'h00
`define ADV7185_REGISTER_11 8'h00
`define ADV7185_REGISTER_12 8'h00
`define ADV7185_REGISTER_13 8'h45
`define ADV7185_REGISTER_14 8'h18
`define ADV7185_REGISTER_15 8'h60
`define ADV7185_REGISTER_16 8'h00
`define ADV7185_REGISTER_17 8'h01
`define ADV7185_REGISTER_18 8'h00
`define ADV7185_REGISTER_19 8'h10
`define ADV7185_REGISTER_1A 8'h10
`define ADV7185_REGISTER_1B 8'hF0
`define ADV7185_REGISTER_1C 8'h16
`define ADV7185_REGISTER_1D 8'h01
`define ADV7185_REGISTER_1E 8'h00
`define ADV7185_REGISTER_1F 8'h3D
`define ADV7185_REGISTER_20 8'hD0
`define ADV7185_REGISTER_21 8'h09
`define ADV7185_REGISTER_22 8'h8C
`define ADV7185_REGISTER_23 8'hE2
`define ADV7185_REGISTER_24 8'h1F
`define ADV7185_REGISTER_25 8'h07
`define ADV7185_REGISTER_26 8'hC2
`define ADV7185_REGISTER_27 8'h58
`define ADV7185_REGISTER_28 8'h3C
`define ADV7185_REGISTER_29 8'h00
`define ADV7185_REGISTER_2A 8'h00
`define ADV7185_REGISTER_2B 8'hA0
`define ADV7185_REGISTER_2C 8'hCE
`define ADV7185_REGISTER_2D 8'hF0
`define ADV7185_REGISTER_2E 8'h00
`define ADV7185_REGISTER_2F 8'hF0
`define ADV7185_REGISTER_30 8'h00
`define ADV7185_REGISTER_31 8'h70
`define ADV7185_REGISTER_32 8'h00
`define ADV7185_REGISTER_34 8'h0F
`define ADV7185_REGISTER_35 8'h01
`define ADV7185_REGISTER_36 8'h00
`define ADV7185_REGISTER_37 8'h00
`define ADV7185_REGISTER_38 8'h00
`define ADV7185_REGISTER_39 8'h00
`define ADV7185_REGISTER_3A 8'h00
`define ADV7185_REGISTER_3B 8'h00

`define ADV7185_REGISTER_44 8'h41
`define ADV7185_REGISTER_45 8'hBB

`define ADV7185_REGISTER_F1 8'hEF
`define ADV7185_REGISTER_F2 8'h80


module adv7185init (reset, clock_27mhz, source, tv_in_reset_b,
                    tv_in_i2c_clock, tv_in_i2c_data);

   input reset;
   input clock_27mhz;
   output tv_in_reset_b; // Reset signal to ADV7185
   output tv_in_i2c_clock; // I2C clock output to ADV7185
   output tv_in_i2c_data; // I2C data line to ADV7185
   input source; // 0: composite, 1: s-video

   initial begin
      $display("ADV7185 Initialization values:");
      $display("  Register 0:  0x%X", `ADV7185_REGISTER_0);
      $display("  Register 1:  0x%X", `ADV7185_REGISTER_1);
```

```
   $display("  Register 2:  0x%X", `ADV7185_REGISTER_2);
   $display("  Register 3:  0x%X", `ADV7185_REGISTER_3);
   $display("  Register 4:  0x%X", `ADV7185_REGISTER_4);
   $display("  Register 5:  0x%X", `ADV7185_REGISTER_5);
   $display("  Register 7:  0x%X", `ADV7185_REGISTER_7);
   $display("  Register 8:  0x%X", `ADV7185_REGISTER_8);
   $display("  Register 9:  0x%X", `ADV7185_REGISTER_9);
   $display("  Register A:  0x%X", `ADV7185_REGISTER_A);
   $display("  Register B:  0x%X", `ADV7185_REGISTER_B);
   $display("  Register C:  0x%X", `ADV7185_REGISTER_C);
   $display("  Register D:  0x%X", `ADV7185_REGISTER_D);
   $display("  Register E:  0x%X", `ADV7185_REGISTER_E);
   $display("  Register F:  0x%X", `ADV7185_REGISTER_F);
   $display("  Register 33: 0x%X", `ADV7185_REGISTER_33);
end

//
// Generate a 1MHz for the I2C driver (resulting I2C clock rate is 250kHz)
//

reg [7:0] clk_div_count, reset_count;
reg clock_slow;
wire reset_slow;

initial
  begin
     clk_div_count <= 8'h00;
     // synthesis attribute init of clk_div_count is "00";
     clock_slow <= 1'b0;
     // synthesis attribute init of clock_slow is "0";
  end

always @(posedge clock_27mhz)
  if (clk_div_count == 26)
    begin
       clock_slow <= ~clock_slow;
       clk_div_count <= 0;
    end
  else
    clk_div_count <= clk_div_count+1;

always @(posedge clock_27mhz)
  if (reset)
    reset_count <= 100;
  else
    reset_count <= (reset_count==0) ? 0 : reset_count-1;

assign reset_slow = reset_count != 0;

//
// I2C driver
//

reg load;
reg [7:0] data;
wire ack, idle;

i2c i2c(.reset(reset_slow), .clock4x(clock_slow), .data(data), .load(load),
        .ack(ack), .idle(idle), .scl(tv_in_i2c_clock),
        .sda(tv_in_i2c_data));

//
// State machine
//

reg [7:0] state;
reg tv_in_reset_b;
reg old_source;

always @(posedge clock_slow)
   if (reset_slow)
     begin
        state <= 0;
        load <= 0;
        tv_in_reset_b <= 0;
        old_source <= 0;
     end
   else
     case (state)
       8'h00:
         begin
            // Assert reset
            load <= 1'b0;
            tv_in_reset_b <= 1'b0;
            if (!ack)
```

```
      state <= state+1;
  end
8'h01:
  state <= state+1;
8'h02:
  begin
     // Release reset
     tv_in_reset_b <= 1'b1;
     state <= state+1;
          end
8'h03:
  begin
     // Send ADV7185 address
     data <= 8'h8A;
     load <= 1'b1;
     if (ack)
       state <= state+1;
  end
8'h04:
  begin
     // Send subaddress of first register
     data <= 8'h00;
     if (ack)
       state <= state+1;
  end
8'h05:
  begin
     // Write to register 0
     data <= `ADV7185_REGISTER_0 | {5'h00, {3{source}}};
     if (ack)
       state <= state+1;
  end
8'h06:
  begin
     // Write to register 1
     data <= `ADV7185_REGISTER_1;
     if (ack)
       state <= state+1;
  end
8'h07:
  begin
     // Write to register 2
     data <= `ADV7185_REGISTER_2;
     if (ack)
       state <= state+1;
  end
8'h08:
  begin
     // Write to register 3
     data <= `ADV7185_REGISTER_3;
     if (ack)
       state <= state+1;
  end
8'h09:
  begin
     // Write to register 4
     data <= `ADV7185_REGISTER_4;
     if (ack)
       state <= state+1;
  end
8'h0A:
  begin
     // Write to register 5
     data <= `ADV7185_REGISTER_5;
     if (ack)
       state <= state+1;
  end
8'h0B:
  begin
     // Write to register 6
     data <= 8'h00; // Reserved register, write all zeros
     if (ack)
       state <= state+1;
  end
8'h0C:
  begin
     // Write to register 7
     data <= `ADV7185_REGISTER_7;
     if (ack)
       state <= state+1;
  end
8'h0D:
  begin
     // Write to register 8
     data <= `ADV7185_REGISTER_8;
```

```
      if (ack)
        state <= state+1;
    end
  8'h0E:
    begin
      // Write to register 9
      data <= `ADV7185_REGISTER_9;
      if (ack)
        state <= state+1;
    end
  8'h0F: begin
     // Write to register A
     data <= `ADV7185_REGISTER_A;
   if (ack)
     state <= state+1;
  end
  8'h10:
    begin
      // Write to register B
      data <= `ADV7185_REGISTER_B;
      if (ack)
        state <= state+1;
    end
  8'h11:
    begin
      // Write to register C
      data <= `ADV7185_REGISTER_C;
      if (ack)
        state <= state+1;
    end
  8'h12:
    begin
      // Write to register D
      data <= `ADV7185_REGISTER_D;
      if (ack)
        state <= state+1;
    end
  8'h13:
    begin
      // Write to register E
      data <= `ADV7185_REGISTER_E;
      if (ack)
        state <= state+1;
    end
  8'h14:
    begin
      // Write to register F
      data <= `ADV7185_REGISTER_F;
      if (ack)
        state <= state+1;
    end
  8'h15:
    begin
      // Wait for I2C transmitter to finish
      load <= 1'b0;
      if (idle)
        state <= state+1;
    end
  8'h16:
    begin
      // Write address
      data <= 8'h8A;
      load <= 1'b1;
      if (ack)
        state <= state+1;
    end
  8'h17:
    begin
      data <= 8'h33;
      if (ack)
        state <= state+1;
    end
  8'h18:
    begin
      data <= `ADV7185_REGISTER_33;
      if (ack)
        state <= state+1;
    end
  8'h19:
    begin
      load <= 1'b0;
      if (idle)
        state <= state+1;
    end
```

```
            8'h1A: begin
               data <= 8'h8A;
               load <= 1'b1;
               if (ack)
                 state <= state+1;
            end
            8'h1B:
              begin
                 data <= 8'h33;
                 if (ack)
                   state <= state+1;
              end
            8'h1C:
              begin
                 load <= 1'b0;
                 if (idle)
                   state <= state+1;
              end
            8'h1D:
              begin
                 load <= 1'b1;
                 data <= 8'h8B;
                 if (ack)
                   state <= state+1;
              end
            8'h1E:
              begin
                 data <= 8'hFF;
                 if (ack)
                   state <= state+1;
              end
            8'h1F:
              begin
                 load <= 1'b0;
                 if (idle)
                   state <= state+1;
              end
            8'h20:
              begin
                 // Idle
                 if (old_source != source) state <= state+1;
                 old_source <= source;
              end
            8'h21: begin
               // Send ADV7185 address
               data <= 8'h8A;
               load <= 1'b1;
               if (ack) state <= state+1;
            end
            8'h22: begin
               // Send subaddress of register 0
               data <= 8'h00;
               if (ack) state <= state+1;
            end
            8'h23: begin
               // Write to register 0
               data <= `ADV7185_REGISTER_0 | {5'h00, {3{source}}};
               if (ack) state <= state+1;
            end
            8'h24: begin
               // Wait for I2C transmitter to finish
               load <= 1'b0;
               if (idle) state <= 8'h20;
            end
          endcase

endmodule

// i2c module for use with the ADV7185

module i2c (reset, clock4x, data, load, idle, ack, scl, sda);

   input reset;
   input clock4x;
   input [7:0] data;
   input load;
   output ack;
   output idle;
   output scl;
   output sda;

   reg [7:0] ldata;
   reg ack, idle;
   reg scl;
   reg sdai;
```

```
reg [7:0] state;

assign sda = sdai ? 1'bZ : 1'b0;

always @(posedge clock4x)
  if (reset)
    begin
        state <= 0;
        ack <= 0;
    end
  else
    case (state)
       8'h00: // idle
         begin
            scl <= 1'b1;
            sdai <= 1'b1;
            ack <= 1'b0;
            idle <= 1'b1;
            if (load)
              begin
                 ldata <= data;
                 ack <= 1'b1;
                 state <= state+1;
              end
         end
       8'h01: // Start
         begin
            ack <= 1'b0;
            idle <= 1'b0;
            sdai <= 1'b0;
            state <= state+1;
         end
       8'h02:
         begin
            scl <= 1'b0;
            state <= state+1;
         end
       8'h03: // Send bit 7
         begin
            ack <= 1'b0;
            sdai <= ldata[7];
            state <= state+1;
         end
       8'h04:
         begin
            scl <= 1'b1;
            state <= state+1;
         end
       8'h05:
         begin
            state <= state+1;
         end
       8'h06:
         begin
            scl <= 1'b0;
            state <= state+1;
         end
       8'h07:
         begin
            sdai <= ldata[6];
            state <= state+1;
         end
       8'h08:
         begin
            scl <= 1'b1;
            state <= state+1;
         end
       8'h09:
         begin
            state <= state+1;
         end
       8'h0A:
         begin
            scl <= 1'b0;
            state <= state+1;
         end
       8'h0B:
         begin
            sdai <= ldata[5];
            state <= state+1;
         end
       8'h0C:
         begin
            scl <= 1'b1;
```

```
      state <= state+1;
    end
  8'h0D:
    begin
      state <= state+1;
    end
  8'h0E:
    begin
      scl <= 1'b0;
      state <= state+1;
    end
  8'h0F:
    begin
      sdai <= ldata[4];
      state <= state+1;
    end
  8'h10:
    begin
      scl <= 1'b1;
      state <= state+1;
    end
  8'h11:
    begin
      state <= state+1;
    end
  8'h12:
    begin
      scl <= 1'b0;
      state <= state+1;
    end
  8'h13:
    begin
      sdai <= ldata[3];
      state <= state+1;
    end
  8'h14:
    begin
      scl <= 1'b1;
      state <= state+1;
    end
  8'h15:
    begin
      state <= state+1;
    end
  8'h16:
    begin
      scl <= 1'b0;
      state <= state+1;
    end
  8'h17:
    begin
      sdai <= ldata[2];
      state <= state+1;
    end
  8'h18:
    begin
      scl <= 1'b1;
      state <= state+1;
    end
  8'h19:
    begin
      state <= state+1;
    end
  8'h1A:
    begin
      scl <= 1'b0;
      state <= state+1;
    end
  8'h1B:
    begin
      sdai <= ldata[1];
      state <= state+1;
    end
  8'h1C:
    begin
      scl <= 1'b1;
      state <= state+1;
    end
  8'h1D:
    begin
      state <= state+1;
    end
  8'h1E:
    begin
      scl <= 1'b0;
```

```
            state <= state+1;
          end
        8'h1F:
          begin
            sdai <= ldata[0];
            state <= state+1;
          end
        8'h20:
          begin
            scl <= 1'b1;
            state <= state+1;
          end
        8'h21:
          begin
            state <= state+1;
          end
        8'h22:
          begin
            scl <= 1'b0;
            state <= state+1;
          end
        8'h23: // Acknowledge bit
          begin
            state <= state+1;
          end
        8'h24:
          begin
            scl <= 1'b1;
            state <= state+1;
          end
        8'h25:
          begin
            state <= state+1;
          end
        8'h26:
          begin
            scl <= 1'b0;
            if (load)
              begin
                 ldata <= data;
                 ack <= 1'b1;
                 state <= 3;
              end
            else
              state <= state+1;
          end
        8'h27:
          begin
            sdai <= 1'b0;
            state <= state+1;
          end
        8'h28:
          begin
            scl <= 1'b1;
            state <= state+1;
          end
        8'h29:
          begin
            sdai <= 1'b1;
            state <= 0;
          end
      endcase

endmodule
```

YCrCbToRGB.v

```
module YCrCb2RGB ( R, G, B, clk, rst, Y, Cr, Cb );
        output [7:0] R, G, B;
        input clk,rst;
        input[9:0] Y, Cr, Cb;
        wire [7:0] R,G,B;
        reg [20:0] R_int,G_int,B_int,X_int,A_int,B1_int,B2_int,C_int;
        reg [9:0] const1,const2,const3,const4,const5;
        reg[9:0] Y_reg, Cr_reg, Cb_reg;
        //registering constants
        always @ (posedge clk)
        begin
                const1 = 10'b 0100101010; //1.164 = 01.00101010
```

```
                const2 = 10'b 0110011000; //1.596 = 01.10011000
                const3 = 10'b 0011010000; //0.813 = 00.11010000
                const4 = 10'b 0001100100; //0.392 = 00.01100100
                const5 = 10'b 1000000100; //2.017 = 10.00000100
        end
        always @ (posedge clk or posedge rst)
                if (rst)
                        begin
                        Y_reg <= 0; Cr_reg <= 0; Cb_reg <= 0;
                        end
                else
                        begin
                        Y_reg <= Y; Cr_reg <= Cr; Cb_reg <= Cb;
                        end
        always @ (posedge clk or posedge rst)
                if (rst)
                        begin
                        A_int <= 0; B1_int <= 0; B2_int <= 0; C_int <= 0; X_int <= 0;
                        end
                else
                        begin
                        X_int <= (const1 * (Y_reg - 'd64)) ;
                        A_int <= (const2 * (Cr_reg - 'd512));
                        B1_int <= (const3 * (Cr_reg - 'd512));
                        B2_int <= (const4 * (Cb_reg - 'd512));
                        C_int <= (const5 * (Cb_reg - 'd512));
                        end
        always @ (posedge clk or posedge rst)
                if (rst)
                        begin
                        R_int <= 0; G_int <= 0; B_int <= 0;
                        end
                else
                        begin
                        R_int <= X_int + A_int;
                        G_int <= X_int - B1_int - B2_int;
                        B_int <= X_int + C_int;
                        end
                assign R = (R_int[20]) ? 0 : (R_int[19:18] == 2'b0) ? R_int[17:10] :
                8'b11111111;
                assign G = (G_int[20]) ? 0 : (G_int[19:18] == 2'b0) ? G_int[17:10] :
                8'b11111111;
                assign B = (B_int[20]) ? 0 : (B_int[19:18] == 2'b0) ? B_int[17:10] :
                8'b11111111;
endmodule
```

RGB2HSV.v

```
module RGB2HSV(clk, reset, R, G, B, H, S, V);

        input clk, reset;
        input [5:0] R, G, B;
        output reg [13:0] H;
        output reg [11:0] S;
        output reg [5:0] V;

        // keep track of direction (pos or neg)
        reg pos_R;
        reg pos_G;
        reg pos_B;

//      // signed versions of R,G,B
//      reg signed [6:0] sr = {1'b0, R};
//      reg signed [6:0] sg = {1'b0, G};
//      reg signed [6:0] sb = {1'b0, B};

        // find max & min of R,G,B
        wire [5:0] max1 = (R > G) ? R : G;
        wire [5:0] max = (max1 > B) ? max1 : B;
        wire [5:0] min1 = (R < G) ? R : G;
        wire [5:0] min = (min1 < B) ? min1 : B;

        // determine positive differences, keep track of direction moving on hue circle
        reg [5:0] sub_R;
        reg [5:0] sub_G;
        reg [5:0] sub_B;

        always @* begin
                if (G > B) begin
```

```
                        sub_R = G - B;
                        pos_R = 1;
                end
                else begin
                        sub_R = B - G;
                        pos_R = 0;
                end
                if (B > R) begin
                        sub_G = B - R;
                        pos_G = 1;
                end
                else begin
                        sub_G = R - B;
                        pos_G = 0;
                end
                if (R > G) begin
                        sub_B = R - G;
                        pos_B = 1;
                end
                else begin
                        sub_B = G - R;
                        pos_B = 0;
                end
        end

        wire [11:0] quotient_R;
        wire [5:0] remainder_R;
        wire rfd_R;
        wire [11:0] quotient_G;
        wire [5:0] remainder_G;
        wire rfd_G;
        wire [11:0] quotient_B;
        wire [5:0] remainder_B;
        wire rfd_B;
        wire [11:0] quotient_S;

        // do unsigned divisions b/c signed numbers make CoreGen complain..

        divide3 divideR(.clk(clk), .dividend(44*sub_R), .divisor(max-min),
                                                        .quotient(quotient_R), .remainder(remainder_R),
.rfd(rfd_R));
        divide3 divideG(.clk(clk), .dividend(44*sub_G), .divisor(max-min),
                                                        .quotient(quotient_G), .remainder(remainder_G),
.rfd(rfd_G));
        divide3 divideB(.clk(clk), .dividend(44*sub_B), .divisor(max-min),
                                                        .quotient(quotient_B), .remainder(remainder_B),
.rfd(rfd_B));

        divide3 divideS(.clk(clk), .dividend(max-min), .divisor(max),
                                                        .quotient(quotient_S), .remainder(remainder_S),
.rfd(rfd_S));

        // make quotients signed
        wire signed [12:0] squotient_R = pos_R ? {1'b0, quotient_R} : {1'b1, quotient_R};
        wire signed [12:0] squotient_G = pos_G ? {1'b0, quotient_G} : {1'b1, quotient_G};
        wire signed [12:0] squotient_B = pos_B ? {1'b0, quotient_B} : {1'b1, quotient_B};

        wire signed [13:0] H_R = (quotient_R + 256) % 256;
        wire signed [13:0] H_G = quotient_G + 88;
        wire signed [13:0] H_B = quotient_B + 176;

        always @(posedge clk) begin
                // calc H, if max=R, H=H_R, etc.
                if (max == min)
                        H <= 0;
                else if (max == R)
                        H <= H_R;
                else if (max == G)
                        H <= H_G;
                else if (max == B)
                        H <= H_B;

                if (max == 0)
                        S <= 0;
                else
                        S <= quotient_S;

                V <= max;

        end

endmodule
```

HandFinder.v

```
module HandFinder(clk, reset, Hdes_min, Hdes_max, H_pixel, V, isHand);

        input clk, reset;
        input [8:0] Hdes_min, Hdes_max, H_pixel;
        input [5:0] V;
        output reg isHand;

        wire [5:0] V_thresh = 21;         // check corners of camimage

        always @(posedge clk) begin
                if (reset)
                        isHand <= 0;
                //if ((H_pixel >= Hdes_min) && (H_pixel <= Hdes_max))
                if (V > V_thresh)
                        isHand <= 1;
                else
                        isHand <= 0;
        end


endmodule
```

RFCoord.v

```
// outputting (x,y) coord RF should have on display screen
module RFCoord(

input clk,
input reset,
input [10:0] hcount,
input [9:0] vcount,
input [8:0] left_width,
input [5:0] finger_width,
input isHand,
output reg [10:0] rf_x_current,
output reg [9:0] rf_y_current

);

        localparam camimage_x = 40;
        localparam camimage_y = 84;
        localparam camimage_rightedge_x = camimage_x + 710;
        localparam camimage_bottomedge_y = camimage_y + 480;
        localparam screen_width = 1024;
        localparam screen_height = 768;

        reg [5:0] counter;
        reg foundfinger;
        reg [10:0] rf_x_next;
        reg [9:0] rf_y_next;
        reg [10:0] unscaled_x;


        always @(posedge clk) begin

                if (reset) begin
                        counter <= 0;
                        foundfinger <= 0;
                        rf_x_current <= 0;
                        rf_y_current <= 0;
                        rf_x_next <= 0;
                        rf_y_next <= 0;
                        end
                else if ((hcount==0) & (vcount==0)) begin          // at top of frame, start search over
                        counter <= 0;
                        foundfinger <= 0;
                        // set x,y coord for drawing on display
                        // make sure RF can't be drawn on left side of screen, and doesn't go off screen
                        rf_x_current <= (rf_x_next < left_width) ? left_width : (rf_x_next > screen_width) ?
screen_width : rf_x_next;
                        rf_y_current <= (rf_y_next > screen_height) ? screen_height :rf_y_next;
                        end
                //else if (hcount <= left_width)          // in left side
```

```
                else if ((hcount <= (camimage_x + left_width)) | (vcount <= camimage_y) | (hcount >=
camimage_rightedge_x) | (vcount >= camimage_bottomedge_y))          // in left side, or to top/right/bottom of camera
image
                        counter <= 0;
                else begin                // in right side of camera image
                        // if already found finger, wait till you get to
                        // the top of the next frame to start checking again
                        if (foundfinger)
                                counter <= 0;
                        // if wide enough to be finger
                        else if (counter >= finger_width) begin
                                // scale finger (x,y) to map to whole screen, instead of just camera image
                                unscaled_x <= (hcount - left_width - camimage_x);
                                rf_x_next <= (unscaled_x/8)*12 + left_width;               // too much calculation?
//                              rf_x_next <= (hcount/8)*12;
                                rf_y_next <= ((vcount-camimage_y)/8)*12;

                                counter <= 0;
                                foundfinger <= 1;
                                end
                        else if (isHand)
                                counter <= counter + 1;
                        else
                                counter <= 0;
                end

        end


endmodule
```

LFCoord.v

```
module LFCoord(

input clk,
input reset,
input [10:0] hcount,
input [9:0] vcount,
input [8:0] left_width,
input [5:0] finger_width,
input isHand,
output reg [10:0] lf_x_current,
output reg [9:0] lf_y_current

);

        localparam camimage_x = 40;
        localparam camimage_y = 84;
        localparam camimage_rightedge_x = camimage_x + 710;
        localparam camimage_bottomedge_y = camimage_y + 480;
        localparam screen_width = 1024;
        localparam screen_height = 768;

        reg [5:0] counter;
        reg foundfinger;
        reg [10:0] lf_x_next;
        reg [9:0] lf_y_next;
        reg [10:0] unscaled_x;


        always @(posedge clk) begin

                if (reset) begin
                        counter <= 0;
                        foundfinger <= 0;
                        lf_x_current <= 0;
                        lf_y_current <= 0;
                        lf_x_next <= 0;
                        lf_y_next <= 0;
                        end
                else if ((hcount==0) & (vcount==0)) begin          // at top of frame, start search over
                        counter <= 0;
                        foundfinger <= 0;
                        // set x,y coord for drawing on display
                        // make sure LF can't be drawn on right side of screen, and doesn't go off screen
                        lf_x_current <= (lf_x_next > left_width) ? left_width : lf_x_next;
                        lf_y_current <= (lf_y_next > screen_height) ? screen_height :lf_y_next;
```

```
                        end
                //else if  ((hcount==0) | (hcount > left_width))           // at left edge of image or in right side
                else if ((hcount <= camimage_x) | (vcount <= camimage_y) | (vcount >= camimage_bottomedge_y) |
(hcount >= camimage_x + left_width))               // above/to left/below camera image, or in right side
                        counter <= 0;
                else begin              // in left side of camera image
                        // if already found finger, wait till you get to
                        // the top of the next frame to start checking again
                        if (foundfinger)
                                counter <= 0;
                        // if wide enough to be finger
                        else if (counter >= finger_width) begin
                                // scale finger (x,y) to map to whole screen, instead of just camera image
                                unscaled_x <= (hcount - finger_width - camimage_x);
                                lf_x_next <= (unscaled_x/8)*10;           // too much calc?
                                lf_y_next <= ((vcount-camimage_y)/8)*12;

                                counter <= 0;
                                foundfinger <= 1;
                                end
                        else if (isHand)
                                counter <= counter + 1;
                        else
                                counter <= 0;
                end

        end


endmodule
```

DrawFingers.v

```
module DrawFingers(clk, reset, hcount, vcount, lf_x, lf_y, rf_x, rf_y, finger_pixel);

        input clk, reset;
        input [10:0] hcount;
        input [9:0] vcount;
        input [10:0] lf_x;
        input [9:0] lf_y;
        input [10:0] rf_x;
        input [9:0] rf_y;
        output reg [23:0] finger_pixel;


        // lf marker
        localparam lf_width = 15;
        localparam lf_height = 15;
        localparam lf_color = {16'd0, 8'd255};
        wire [23:0] lf_pixel;

        // rf marker
        localparam rf_width = 15;
        localparam rf_height = 15;
        localparam rf_color = {8'd255, 16'd0};
        wire [23:0] rf_pixel;


        // LF
//      blob lf(.WIDTH(lf_width),.HEIGHT(lf_height),.COLOR(lf_color),
.x(lf_x),.y(lf_y),.hcount(hcount),.vcount(vcount),
//                                      .pixel(lf_pixel));
        arrow lf(clk, reset, hcount, vcount, lf_x, lf_y, lf_color, lf_pixel);

        // RF
//      blob rf(.WIDTH(rf_width),.HEIGHT(rf_height),.COLOR(rf_color),
.x(rf_x),.y(rf_y),.hcount(hcount),.vcount(vcount),
//                                      .pixel(rf_pixel));
        plus rf(clk, reset, hcount, vcount, rf_x, rf_y, rf_width, rf_height, rf_color, rf_pixel);

        always @(posedge clk) begin
                if (reset)
                        finger_pixel <= 0;
                else
                        finger_pixel <= (lf_pixel | rf_pixel);
        end

endmodule
```

Arrow.v

```
module arrow(clk, reset, hcount, vcount, x, y, color, arrow_pixel);

        input clk, reset;
        input [10:0] hcount;
        input [9:0] vcount;
        input [10:0] x;
        input [9:0] y;
        input [23:0] color;
        output reg [23:0] arrow_pixel;

        localparam height = 20;

        always @(posedge clk) begin
                if (reset)
                        arrow_pixel <= 0;
                else if ((hcount==x) & (vcount >= y) & (vcount <= (y+height)))      // vertical line
                        arrow_pixel <= color;
                else if ((vcount==y+1) & (hcount>=x-1) & (hcount<=x+1))             // sides
                        arrow_pixel <= color;
                else if ((vcount==y+2) & (hcount>=x-2) & (hcount<=x+2))
                        arrow_pixel <= color;
                else if ((vcount==y+3) & (hcount>=x-3) & (hcount<=x+3))
                        arrow_pixel <= color;
                else if ((vcount==y+4) & (hcount>=x-4) & (hcount<=x+4))
                        arrow_pixel <= color;
                else if ((vcount==y+5) & (hcount>=x-5) & (hcount<=x+5))
                        arrow_pixel <= color;
                else if ((vcount==y+6) & (hcount>=x-6) & (hcount<=x+6))
                        arrow_pixel <= color;
                else if ((vcount==y+7) & (hcount>=x-7) & (hcount<=x+7))
                        arrow_pixel <= color;
                else
                        arrow_pixel <= 0;
        end

endmodule
```

Plus.v

```
module plus(clk, reset, hcount, vcount, x, y, width, height, color, plus_pixel);

        input clk, reset;
        input [10:0] hcount;
        input [9:0] vcount;
        input [10:0] x;
        input [9:0] y;
        input [10:0] width;
        input [9:0] height;
        input [23:0] color;
        output reg [23:0] plus_pixel;

        always @(posedge clk) begin
                if (reset)
                        plus_pixel <= 0;
                else if (((hcount==x) | (vcount==y)) & ((hcount >= (x-width/2)) & (hcount <= (x+width/2))) & ((vcount
>= (y-height/2)) & (vcount <= (y+height/2))))
                        plus_pixel <= color;
                else
                        plus_pixel <= 0;
        end

endmodule
```

OptionFinder.v

```
// based on LF location, give me editing option
// 00: nothing
```

```
// 01: CROP
// 10: ROTATE
// 11: RESTORE
module OptionFinder(clk, reset, lf_x, lf_y, option);

        input clk, reset;
        input [10:0] lf_x;
        input [9:0] lf_y;
        output reg [1:0] option;

        localparam screen_width = 800;
        localparam screen_height = 600;
        localparam left_width = 150;
        localparam option_width = 100;
        localparam option_height = 80;
        localparam option_x = (left_width - option_width) / 2;
        localparam space_height = (screen_height - 3*option_height)/4;
        localparam crop_y = space_height;
        localparam rotate_y = option_height + 2*space_height;
        localparam restore_y = 2*option_height + 3*space_height;

        always @(posedge clk) begin
                if (reset)
                        option <= 0;
                // if possibly an option in terms of x loc
                else if ((lf_x > option_x) & (lf_x < (option_x + option_width))) begin
                        // if y in crop, option=1
                        if ((lf_y > crop_y) & (lf_y < (crop_y + option_height)))
                                option <= 1;
                        // if in rotate, option=2
                        else if ((lf_y > rotate_y) & (lf_y < (rotate_y + option_height)))
                                option <= 2;
                        // if in restore, option=3
                        else if ((lf_y > restore_y) & (lf_y < (restore_y + option_height)))
                                option <= 3;
                        else
                                option <= 0;
                end
                // else, option=0
                else
                        option <= 0;
        end


endmodule
```

FSM.v

```
module FSM(clk, reset, hcount, vcount, rf_x, rf_y, image_x, image_y, image_width,
image_height, option, done_cropping, done_rotating, expired, show_rectangle, crop, rotate,
crop_start_x, crop_start_y, crop_end_x, crop_end_y, ref_pt_x, ref_pt_y, new_pt_x,
new_pt_y, show_corners, start_timer, state);

        input clk, reset;
        input [10:0] hcount;
        input [9:0] vcount;
        input [10:0] rf_x;
        input [9:0] rf_y;
        input [10:0] image_x;
        input [9:0] image_y;
        input [10:0] image_width;
        input [9:0] image_height;
        input [1:0] option;
        input done_cropping;
        input done_rotating;
        input expired;
        output reg show_rectangle;
        output reg crop;
        output reg rotate;
        output reg [10:0] crop_start_x;
        output reg [9:0] crop_start_y;
        output reg [10:0] crop_end_x;
        output reg [9:0] crop_end_y;
        output reg [10:0] ref_pt_x;
        output reg [9:0] ref_pt_y;
        output reg [10:0] new_pt_x;
        output reg [9:0] new_pt_y;
        output reg show_corners;
        output reg start_timer;
```

```
        output reg [2:0] state;

        localparam DEFAULT = 3'd0;
        localparam START_CROP = 3'd1;
        localparam MID_CROP = 3'd2;
        localparam END_CROP = 3'd3;
        localparam START_ROTATE = 3'd4;
        localparam MID_ROTATE = 3'd5;
        localparam END_ROTATE = 3'd6;
        localparam RESTORE = 3'd7;

//      localparam image_width = 500;
//      localparam image_height = 300;

        wire inside_image = ((rf_x>=image_x & rf_x<=(image_x+image_width)) & (rf_y>=image_y &
rf_y<=(image_y+image_height)));

        reg [2:0] next_state;
        reg waiting;

        // sequential logic: state register
        always @(posedge clk) begin
                if (reset) begin
                        state <= DEFAULT;
                end
                else begin
                        state <= next_state;
                end
        end

        always@(posedge clk) begin
           if (reset) begin
                        crop_end_x <= 0;
                        crop_end_y <= 0;
                        crop <= 0;
                        start_timer <= 0;
                        waiting <= 0;
                end
          case (state)
            START_CROP:
                        begin
                                crop_start_x <= rf_x;
                                crop_start_y <= rf_y;
                        end
                 MID_CROP:      // 2    // draw rectangle
                        begin
                                if (option == 1) begin
                                        show_rectangle <= 1;
                                end
                                else begin
                                        crop_end_x <= rf_x;
                                        crop_end_y <= rf_y;
                                        show_rectangle <= 0;
                                        crop <= 1;
                                end
                        end

                END_CROP:       // 3    // do actual cropping
                        begin
                                if (done_cropping) begin
                                        crop <= 0;
                                end
                                else begin
                                        crop <= 1;
                                end
                        end

                START_ROTATE:   // 4    // find ref corner, draw circle around it
                        begin
                                ref_pt_x <= rf_x;
                                ref_pt_y <= rf_y;
                        end

                MID_ROTATE:             // 5    // find new corner, draw circle around it
                        begin
                                if (option==2) begin
                                        show_corners <= 1;
                                end
                                else begin
                                        show_corners <= 0;
                                        new_pt_x <= rf_x;
                                        new_pt_y <= rf_y;
                                        rotate <= 1;
                                end
                        end
```

```
                END_ROTATE:
                        begin
                                if (done_rotating)
                                        rotate <= 0;
                        end

                RESTORE:
                        begin
//                              if (expired)
//                                      waiting <= 0;
//                              else if (waiting)
//                                      start_timer <= 0;
//                              else begin
//                                      start_timer <= 1;
//                                      waiting <= 1;
//                              end
                        end

          endcase
        end


        always @* begin
                case(state)
                        DEFAULT:        // 0
                                begin
                                        if ((option==1) & inside_image)           // CROP
                                                next_state = START_CROP;
                                        else if (option == 2)     // ROTATE
                                                next_state = START_ROTATE;
                                        else if (option == 3)
                                                next_state = RESTORE;
                                        else
                                                next_state = DEFAULT;
                                end

                        START_CROP:             // 1    // store starting location
                                        next_state = MID_CROP;

                        MID_CROP:       // 2    // draw rectangle
                                begin
                                        if (option == 1) begin
                                                next_state = MID_CROP;
                                        end
                                        else begin
                                                next_state = END_CROP;
                                        end
                                end

                        END_CROP:       // 3    // do actual cropping
                                begin
                                        if (done_cropping) begin
                                                next_state = DEFAULT;
                                        end
                                        else begin
                                                next_state = END_CROP;
                                        end
                                end

                        START_ROTATE:   // 4    // find ref corner, draw circle around it
                                begin
                                        next_state = MID_ROTATE;
                                end

                        MID_ROTATE:             // 5    // find new corner, draw circle around it
                                begin
                                        if (option==2) begin
                                                next_state = MID_ROTATE;
                                        end
                                        else begin
                                                next_state = END_ROTATE;
                                        end
                                end

                        END_ROTATE:             // 6    // rotate image
                                begin
                                        if (done_rotating)
                                                next_state = DEFAULT;
                                        else
                                                next_state = END_ROTATE;
                                end

                        RESTORE:        // 7    // restore image to original form
                                begin
```

```
//                                      if (expired)
//                                              next_state = DEFAULT;
//                                      else
//                                              next_state = RESTORE;
                                        next_state = DEFAULT;
                                end


                endcase

        end


endmodule
```

BlobGenerator.v

```
module BlobGenerator(input vclock, input reset, input [10:0] hcount, input [9:0] vcount,
                                                        input hsync, input vsync, input blank,    input crop,
                                                        input [10:0] new_image_x, input [9:0] new_image_y,
                                                        input [10:0] crop_end_x, input [9:0] crop_end_y,
                                                        input [10:0] crop_image_width, input [9:0]
crop_image_height,
                                                        input [10:0] rotate_image_width, input [9:0]
rotate_image_height,
                                                        input done_rotating, input expired,
                                                        output reg [10:0] image_width, output reg [9:0]
image_height,
                                                        output reg [10:0] image_x, output reg [9:0] image_y, output
[23:0] image_pixel,
                                                        output reg [23:0] pixel, output reg restored);


        localparam screen_width = 800;
        localparam screen_height = 600;

        // left
        localparam left_width = 150;
        localparam left_height = screen_height;
        wire left_x = 0;
        wire left_y = 0;
        localparam left_color = 24'h99CCFF;             // soft blue
        wire [23:0] left_pixel;

        // right
        localparam right_width = screen_width - left_width;
        localparam right_height = screen_height;
        localparam right_x = 150;
        localparam right_y = 0;
        localparam right_color = 24'h330033;            // dark color
        wire [23:0] right_pixel;

        // editing options
        localparam option_width = 100;
        localparam option_height = 80;
        localparam option_x = (left_width - option_width) / 2;
        localparam space_height = (screen_height - 3*option_height)/4;
        localparam crop_y = space_height;
        localparam rotate_y = option_height + 2*space_height;
        localparam restore_y = 2*option_height + 3*space_height;
        localparam option_color = 24'h9999CC;           // purple
        wire [23:0] crop_pixel;
        wire [23:0] rotate_pixel;
        wire [23:0] restore_pixel;

        // image
//      wire [10:0] image_width = (new_image_width!=0) ? new_image_width : 500;
//      wire [9:0] image_height = (new_image_height!=0) ? new_image_height : 300;
        localparam image_width_initial = 400;
        localparam image_height_initial = 300;
//      assign image_width = (crop_end_x==0) ? image_width_initial : !(crop_end_x<new_image_x) ? crop_image_width :
image_width_initial;
//      assign image_height = (crop_end_y==0) ? image_height_initial : !(crop_end_y<new_image_y) ? crop_image_height :
image_height_initial;
        localparam image_color = 24'hFF99FF;      // pink
//      assign image_x = (new_image_x!=0) ? new_image_x : (right_x + (right_width - image_width) / 2);
//      assign image_y = (new_image_y!=0) ? new_image_y :(screen_height - image_height) / 2;
//      wire [10:0] image_x_initial = (right_x + (right_width - image_width) / 2);
//      wire [9:0] image_y_initial = (screen_height - image_height) / 2;
```

```
        localparam image_x_initial = (right_x + (right_width - image_width_initial) / 2);
        localparam image_y_initial = (screen_height - image_height_initial) / 2;
//      assign image_x = (crop_end_x!=0) ? new_image_x : image_x_initial;
//      assign image_y = (crop_end_y!=0) ? new_image_y : image_y_initial;
//      wire [23:0] image_pixel;

        always @(posedge vclock) begin
//              if (reset | expired) begin
                if (reset) begin
                        image_x <= image_x_initial;
                        image_y <= image_y_initial;
                end
                else if (crop) begin
                        image_x <= new_image_x;
                        image_y <= new_image_y;
                end
        end

//      always @(posedge crop) begin
//              image_x <= new_image_x;
//              image_y <= new_image_y;
//      end


        // character display module
        // crop
        //localparam cx1 = option_x - 25;
        localparam cx1 = 0;
        localparam cy1 = crop_y + (option_height / 2) - 10;
        wire [63:0] cstring1 = "CROP";
        wire [2:0]  cdpixel1;
        char_string_display cd1(vclock,hcount,vcount,
                                cdpixel1,cstring1,cx1,cy1);

        // rotate
        localparam cx2 = option_x - 22;
        localparam cy2 = rotate_y + (option_height / 2) - 10;
        wire [63:0] cstring2 = "ROTATE";
        wire [2:0]  cdpixel2;
        char_string_display cd2(vclock,hcount,vcount,
                                cdpixel2,cstring2,cx2,cy2);

        // restore
        localparam cx3 = option_x - 20;
        localparam cy3 = restore_y + (option_height / 2) - 10;
        wire [63:0] cstring3 = "RESTORE";
        wire [2:0]  cdpixel3;
        char_string_display cd3(vclock,hcount,vcount,
                                cdpixel3,cstring3,cx3,cy3);

        // image
//      wire [10:0] cx4 = image_x;
//      wire [9:0] cy4 = image_y + (image_height / 2) - 10;
//      wire [63:0] cstring4 = "IMAGE";
//      wire [2:0]  cdpixel4;
//      char_string_display cd4(vclock,hcount,vcount,
//                              cdpixel4,cstring4,cx4,cy4);



        //assign pixel = (left_pixel | right_pixel | crop_pixel | rotate_pixel | restore_pixel | cdpixel1 | cdpixel2 |
cdpixel3);

        // left side
        blob left_side(.WIDTH(left_width),.HEIGHT(left_height),.COLOR(left_color),
.x(left_x),.y(left_y),.hcount(hcount),.vcount(vcount),
                                        .pixel(left_pixel));

        // right side
        blob right_side(.WIDTH(right_width),.HEIGHT(right_height),.COLOR(right_color),
.x(right_x),.y(right_y),.hcount(hcount),.vcount(vcount),
                                        .pixel(right_pixel));

        // crop
        blob cropp(.WIDTH(option_width),.HEIGHT(option_height),.COLOR(option_color),
.x(option_x),.y(crop_y),.hcount(hcount),.vcount(vcount),
                                        .pixel(crop_pixel));

        // rotate
        blob rotate(.WIDTH(option_width),.HEIGHT(option_height),.COLOR(option_color),
.x(option_x),.y(rotate_y),.hcount(hcount),.vcount(vcount),
                                        .pixel(rotate_pixel));

        // restore
        blob restore(.WIDTH(option_width),.HEIGHT(option_height),.COLOR(option_color),
```

```
.x(option_x),.y(restore_y),.hcount(hcount),.vcount(vcount),
                                        .pixel(restore_pixel));

        // image
        blob image(.WIDTH(image_width),.HEIGHT(image_height),.COLOR(image_color),
.x(image_x),.y(image_y),.hcount(hcount),.vcount(vcount),
                                        .pixel(image_pixel));



//      assign image_width = (crop_end_x==0) ? image_width_initial : !(crop_end_x<new_image_x) ? crop_image_width :
image_width_initial;
//      assign image_height = (crop_end_y==0) ? image_height_initial : !(crop_end_y<new_image_y) ? crop_image_height :
image_height_initial;
//      assign image_x = (crop_end_x!=0) ? new_image_x : image_x_initial;
//      assign image_y = (crop_end_y!=0) ? new_image_y : image_y_initial;



        always @(posedge vclock) begin
//              if (reset | expired) begin
                if (reset) begin
                        image_width <= image_width_initial;
                        image_height <= image_height_initial;
                end
                else if (done_rotating) begin
                        image_width <= rotate_image_width;
                        image_height <= rotate_image_height;
//                      image_width <= image_height;
//                      image_height <= image_width;
                end
                else if ((crop_end_x>new_image_x) & (crop_end_y>new_image_y)) begin
                        image_width <= crop_image_width;
                        image_height <= crop_image_height;
                end
//              else begin
//                      image_width <= image_width_initial;
//                      image_height <= image_height_initial;
//              end
        end



//      reg [23:0] color;

        always @(posedge vclock) begin
                if (reset) begin
                        pixel <= 0;
                end
                else if ((cdpixel1!=0) | (cdpixel2!=0) | (cdpixel3!=0))
                        pixel <= 0;
                else if ((crop_pixel!=0) | (rotate_pixel!=0) | (restore_pixel!=0))
                        pixel <= option_color;
//              else if (F_pixel != 0)
//                      pixel <= F_color;
//              else if (image_pixel != 0)
//                      pixel <= (image_pixel);
                else if (left_pixel != 0)
                        pixel <= left_color;
                else if (right_pixel != 0)
                        pixel <= right_color;
                else
                        pixel <= {8'd255, 8'd255, 8'd255};
        end

//      always @* begin
//              if (lfpixel!=0)
//                      assign color = {8'd255, 16'd0};           // red box for LF
//              else if (rfpixel!=0)
//                      assign color = {16'd0, 8'd255};           // blue box for RF
//              else if ((cdpixel1!=0) | (cdpixel2!=0) | (cdpixel3!=0) | (cdpixel4 !=0))
//                      assign color = 3'b111;
//              else if ((crop_pixel!=0) | (rotate_pixel!=0) | (restore_pixel!=0))
//                      assign color = option_color;
//              else if (image_pixel != 0)
//                      assign color = image_color;
//              else if (left_pixel != 0)
//                      assign color = left_color;
//              else if (right_pixel != 0)
//                      assign color = right_color;
//      end

//      assign pixel = color;
```

```
endmodule
```

DrawRectangle.v

```
// draw rectangle while in MID_CROP state
module DrawRectangle(input clk, reset, input [10:0] hcount, input [9:0] vcount, input show_rectangle,
                     input [10:0] crop_start_x, input [9:0] crop_start_y, input [10:0] rf_x, input [9:0] rf_y,
                                                                 input [10:0] image_x, input [9:0] image_y, input [10:0]
image_width, input[9:0] image_height,
                                                                 output [10:0] crop_end_xx, output [10:0] crop_end_yy,
output reg [23:0] rectangle_pixel);

//      input clk, reset;
//   input [10:0] hcount;// horizontal index of current pixel (0..1023)
//   input [9:0]         vcount; // vertical index of current pixel (0..767)
//      input show_rectangle;
//      input [10:0] crop_start_x;
//      input [9:0] crop_start_y;
//      input [10:0] rf_x;
//      input [9:0] rf_y;
//      input [10:0] image_x;
//      input [9:0] image_y;
//      input [10:0] image_width;
//      input [9:0] image_height;
////     output reg [10:0] crop_end_xx;
////     output reg [9:0] crop_end_yy;
//      output [10:0] crop_end_xx;
//      output [9:0] crop_end_yy;
//      output reg [23:0] rectangle_pixel;

//      localparam image_width = 500;
//      localparam image_height = 300;
        wire [10:0] image_rightedge_x = image_x + image_width;
        wire [9:0] image_bottomedge_y = image_y + image_height;

//      wire [10:0] rf_xx = (rf_x > image_rightedge_x) ? image_rightedge_x : rf_x;
//      wire [9:0] rf_yy = (rf_y > image_bottomedge_y) ? image_bottomedge_y : rf_y;
        assign crop_end_xx = (rf_x > image_rightedge_x) ? image_rightedge_x : rf_x;
        assign crop_end_yy = (rf_y > image_bottomedge_y) ? image_bottomedge_y : rf_y;

        wire [23:0] rectangle_color = {8'd255, 8'd255, 8'd255};
        wire horizontal_edge = ((hcount >= crop_start_x && hcount <= crop_end_xx) && (vcount==crop_start_y ||
vcount==crop_end_yy));
        wire vertical_edge = ((vcount >= crop_start_y && vcount <= crop_end_yy) && (hcount==crop_start_x ||
hcount==crop_end_xx));

        always @(posedge clk) begin

                if (show_rectangle) begin
                        if (horizontal_edge | vertical_edge)
                                rectangle_pixel <= rectangle_color;
                        else
                                rectangle_pixel <= 0;
                end
                else begin
                        // store end corner for cropping
//                      crop_end_xx <= rf_xx;
//                      crop_end_yy <= rf_yy;
                        rectangle_pixel <= 0;
                end

        end


/*
        module blob
        #(parameter WIDTH = 64,      // default width: 64 pixels
                                     HEIGHT = 64,     // default height: 64 pixels
                                     COLOR = 24'd16777215)  // default color: white
        (input [10:0] x,hcount,
         input [9:0] y,vcount,
         output reg [23:0] pixel);

        always @ (x or y or hcount or vcount) begin
                if ((hcount >= x && hcount < (x+WIDTH)) && (vcount >= y && vcount < (y+HEIGHT)))
                        pixel = COLOR;
                else pixel = 0;
        end
```

```
endmodule
*/

endmodule
```

Cropper.v

```
module Cropper(input clk, reset, input [10:0] hcount, input [9:0] vcount, input crop,
               input [10:0] crop_start_x, input [9:0] crop_start_y, input [10:0] crop_end_x, input [9:0] crop_end_y,
                                          input [10:0] image_x, input [9:0] image_y, input [10:0] image_width, input
[9:0] image_height, output reg done_cropping, output reg [23:0] crop_pixel);

//       input clk, reset;
//       input [10:0] hcount;
//       input [9:0] vcount;
//       input crop;
//       input [10:0] crop_start_x;
//       input [9:0] crop_start_y;
//       input [10:0] crop_end_xx;
//       input [9:0] crop_end_yy;
//       input [10:0] image_x;
//       input [9:0] image_y;
//       input [10:0] image_width;
//       input [9:0] image_height;
//       output reg done_cropping;
//       output reg [23:0] crop_pixel;

         localparam right_color = 24'h330033;              // dark color
         localparam left_width = 150;
         reg cropping;    // high when doing actual cropping

         wire [10:0] image_rightedge_x = image_x + image_width;
         wire [9:0] image_bottomedge_y = image_y + image_height;

//       wire [10:0] crop_end_xx = (crop_end_x > image_rightedge_x) ? image_rightedge_x : crop_end_x;
//       wire [9:0] crop_end_yy = (crop_end_y > image_bottomedge_y) ? image_bottomedge_y : crop_end_y;
         reg [10:0] crop_end_xx;
         reg [9:0] crop_end_yy;

         always @(posedge crop) begin                      // can i do this?
                  if (crop_end_x > image_rightedge_x)
                           crop_end_xx <= image_rightedge_x;
                  else
                           crop_end_xx <= crop_end_x;
         end

         always @(posedge crop) begin                      // can i do this?
                  if (crop_end_y > image_bottomedge_y)
                           crop_end_yy <= image_bottomedge_y;
                  else
                           crop_end_yy <= crop_end_y;
         end


         wire unwanted_pixel = ((hcount < crop_start_x) || (hcount > crop_end_xx) || (vcount < crop_start_y) || (vcount
> crop_end_yy));


         always @(posedge clk) begin
                  done_cropping <= 0;
                  if (reset) begin
                           done_cropping <= 0;
                           crop_pixel <= 0;
                           cropping <= 0;
                  end
                  else if (crop) begin
                           if ((hcount==0) & (vcount==0)) begin
                                    if (cropping) begin               // reached top of frame again, so done cropping
                                             crop_pixel <= 0;
                                             cropping <= 0;
                                             done_cropping <= 1;
                                    end
                                    else begin                        // crop has gone hi, haven't done any cropping yet
                                             cropping <= 1;
                                             done_cropping <= 0;
                                    end
                           end
                           else if (hcount < left_width)
                                    crop_pixel <= 0;
```

```
                        else if (cropping) begin // on right side & cropping
                                if (unwanted_pixel)
                                        crop_pixel <= right_color;
                                else
                                        crop_pixel <= 0;
                        end
                        else
                                crop_pixel <= 0;
                end
                else
                        crop_pixel <= 0;
        end

endmodule
```

Corner1Finder.v

```
module Corner1Finder(clk, reset, rf_x, rf_y, ref_pt_x, ref_pt_y, show_corners, corner1);

        input clk, reset;
        input [10:0] rf_x;
        input [9:0] rf_y;
        input [10:0] ref_pt_x;
        input [9:0] ref_pt_y;
        input show_corners;
        output reg [2:0] corner1; // if =4, no corner selected

        localparam TL = 2'd0;
        localparam TR = 2'd1;
        localparam BR = 2'd2;
        localparam BL = 2'd3;
        localparam screen_width = 800;
        localparam screen_height = 600;
        wire [10:0] x1 = (show_corners) ? ref_pt_x : rf_x;
        wire [9:0] y1 = (show_corners) ? ref_pt_y : rf_y;
//      wire [10:0] x1 = rf_x;
//      wire [9:0] y1 = rf_y;

        always @(posedge clk) begin
                // FIRST CORNER
                if (x1 == 0)
                        corner1 <= 4;
                else if (x1 < (screen_width/2)) begin              // in left half
                        if (y1 < (screen_height/2))                // in top half
                                corner1 <= TL;
                        else
                                corner1 <= BL;
                end
                else begin                                                                              // in right half
                        if (y1 < (screen_height/2))                // in top half
                                corner1 <= TR;
                        else
                                corner1 <= BR;
                end
        end


endmodule
```

Corner2Finder.v

```
module Corner2Finder(clk, reset, rf_x, rf_y, new_pt_x, new_pt_y, show_corners, corner2);


        input clk, reset;
        input [10:0] rf_x;
        input [9:0] rf_y;
        input [10:0] new_pt_x;
        input [9:0] new_pt_y;
        input show_corners;
        output reg [2:0] corner2; // if =4, don't show circle

        localparam TL = 2'd0;
        localparam TR = 2'd1;
```

```
        localparam BR = 2'd2;
        localparam BL = 2'd3;
        localparam screen_width = 800;
        localparam screen_height = 600;
        wire [10:0] x2 = (show_corners) ? rf_x : 0;
        wire [9:0] y2 = (show_corners) ? rf_y : 0;

        always @(posedge clk) begin

                // SECOND CORNER
                if (x2 == 0)
                        corner2 <= 4;
                else if (x2 < (screen_width/2)) begin              // in left half
                        if (y2 < (screen_height/2))                // in top half
                                corner2 <= TL;
                        else
                                corner2 <= BL;
                end
                else begin                                                                      // in right half
                        if (y2 < (screen_height/2))                // in top half
                                corner2 <= TR;
                        else
                                corner2 <= BR;
                end
        end


endmodule
```

DrawCorner1.v

```
module DrawCorner1(clk, reset, hcount, vcount, corner1, color1, image_x, image_y, image_width, image_height,
corner1_pixel);

        input clk, reset;
        input [10:0] hcount;
        input [9:0] vcount;
        input [2:0] corner1;
        input [23:0] color1;
        input [10:0] image_x;
        input [9:0] image_y;
        input [10:0] image_width;
        input [9:0] image_height;
        output [23:0] corner1_pixel;

        localparam screen_width = 800;
        localparam screen_height = 600;
        localparam radius = 6;
        localparam TL = 2'd0;
        localparam TR = 2'd1;
        localparam BR = 2'd2;
        localparam BL = 2'd3;
        reg [10:0] x1;
        reg [9:0] y1;
//      wire [10:0] x11 = x1 - radius/2;
//      wire [9:0] y11 = y1 - radius/2;
        wire [10:0] x11 = x1 - 4;
        wire [9:0] y11 = y1 - 4;

        always @(posedge clk) begin

                // pick x,y based on which corner
                if (corner1 == TL) begin
                        x1 <= image_x;
                        y1 <= image_y;
                end
                else if (corner1 == TR) begin
                        x1 <= image_x + image_width;
                        y1 <= image_y;
                end
                else if (corner1 == BR) begin
                        x1 <= image_x + image_width;
                        y1 <= image_y + image_height;
                end
                else if (corner1 == BL) begin
                        x1 <= image_x;
                        y1 <= image_y + image_height;
                end
                else begin      // don't show marker
```

```
                        x1 <= screen_width;
                        y1 <= screen_height;
                end

        end

        circle c1(.RADIUS(radius), .COLOR(color1), .x(x11), .y(y11), .hcount(hcount), .vcount(vcount),
.pixel(corner1_pixel));


endmodule
```

DrawCorner2.v

```
module DrawCorner2(clk, reset, hcount, vcount, corner2, color2, image_x, image_y, image_width, image_height,
corner2_pixel);

        input clk, reset;
        input [10:0] hcount;
        input [9:0] vcount;
        input [2:0] corner2;
        input [23:0] color2;
        input [10:0] image_x;
        input [9:0] image_y;
        input [10:0] image_width;
        input [9:0] image_height;
        output [23:0] corner2_pixel;

        localparam screen_width = 800;
        localparam screen_height = 600;

        localparam radius = 6;
        localparam TL = 2'd0;
        localparam TR = 2'd1;
        localparam BR = 2'd2;
        localparam BL = 2'd3;
        reg [10:0] x2;
        reg [9:0] y2;
        wire [10:0] x22 = x2 - 4;
        wire [9:0] y22 = y2 - 4;

        always @(posedge clk) begin

                // pick x,y based on which corner
                if (corner2 == TL) begin
                        x2 <= image_x;
                        y2 <= image_y;
                end
                else if (corner2 == TR) begin
                        x2 <= image_x + image_width;
                        y2 <= image_y;
                end
                else if (corner2 == BR) begin
                        x2 <= image_x + image_width;
                        y2 <= image_y + image_height;
                end
                else if (corner2 == BL) begin
                        x2 <= image_x;
                        y2 <= image_y + image_height;
                end
                else begin       // don't show marker
                        x2 <= screen_width;
                        y2 <= screen_height;
                end

        end

        circle c2(.RADIUS(radius), .COLOR(color2), .x(x22), .y(y22), .hcount(hcount), .vcount(vcount),
.pixel(corner2_pixel));


endmodule
```

GetRotatedParams.v

```
module GetRotatedParams(input clk, reset, input [10:0] x, input [9:0] y, input [10:0] width,
                                                                        input [9:0] height, input [10:0] image_x, input
[9:0] image_y,
                                                                        input [10:0] image_width, input [9:0] image_height,
output reg [10:0] new_x,
                                                                        output reg [9:0] new_y, output reg [10:0]
new_width, output reg [9:0] new_height);


        always @(posedge clk) begin
                new_x <= image_x + y - image_y;
                new_y <= image_y + image_x + image_width - x - width;
                new_width <= height;
                new_height <= width;
        end

endmodule
```

Rotator.v

```
// changes appropriate F parameters based on ref corner and new corner
// only happens on rotate<=1
module Rotator(input clk, reset, input rotate, input [2:0] corner1,  input [2:0] corner2,
                                        input [10:0] A_x, input [9:0] A_y, input [10:0] A_width, input [9:0]
A_height,
                                        input [10:0] B_x, input [9:0] B_y, input [10:0] B_width, input [9:0]
B_height,
                                        input [10:0] C_x, input [9:0] C_y, input [10:0] C_width, input [9:0]
C_height,
                                        input [10:0] image_x, input [9:0] image_y, input [10:0] image_width, input
[9:0] image_height,
                                        output reg [10:0] A_x_new, output reg [9:0] A_y_new, output reg [10:0]
A_width_new, output reg [9:0] A_height_new,
                                        output reg [10:0] B_x_new, output reg [9:0] B_y_new, output reg [10:0]
B_width_new, output reg [9:0] B_height_new,
                                        output reg [10:0] C_x_new, output reg [9:0] C_y_new, output reg [10:0]
C_width_new, output reg [9:0] C_height_new,
                                        output reg [10:0] image_width_new, output reg [9:0] image_height_new, output
reg done_rotating);

        wire [10:0] A_x_temp1;
        wire [9:0] A_y_temp1;
        wire [10:0] A_width_temp1;
        wire [9:0] A_height_temp1;
        wire [10:0] B_x_temp1;
        wire [9:0] B_y_temp1;
        wire [10:0] B_width_temp1;
        wire [9:0] B_height_temp1;
        wire [10:0] C_x_temp1;
        wire [9:0] C_y_temp1;
        wire [10:0] C_width_temp1;
        wire [9:0] C_height_temp1;
        wire [10:0] A_x_temp2;
        wire [9:0] A_y_temp2;
        wire [10:0] A_width_temp2;
        wire [9:0] A_height_temp2;
        wire [10:0] B_x_temp2;
        wire [9:0] B_y_temp2;
        wire [10:0] B_width_temp2;
        wire [9:0] B_height_temp2;
        wire [10:0] C_x_temp2;
        wire [9:0] C_y_temp2;
        wire [10:0] C_width_temp2;
        wire [9:0] C_height_temp2;
        wire [10:0] A_x_temp3;
        wire [9:0] A_y_temp3;
        wire [10:0] A_width_temp3;
        wire [9:0] A_height_temp3;
        wire [10:0] B_x_temp3;
        wire [9:0] B_y_temp3;
        wire [10:0] B_width_temp3;
        wire [9:0] B_height_temp3;
        wire [10:0] C_x_temp3;
        wire [9:0] C_y_temp3;
        wire [10:0] C_width_temp3;
        wire [9:0] C_height_temp3;
```

```
        localparam TL = 2'd0;
        localparam TR = 2'd1;
        localparam BR = 2'd2;
        localparam BL = 2'd3;


        // calculate new values, even if not going to use them

        // 90 deg CCW
        GetRotatedParams A1(.clk(clk), .reset(reset), .x(A_x), .y(A_y), .width(A_width),
                                                                .height(A_height), .image_x(image_x),
.image_y(image_y),
                                                                .image_width(image_width),
.image_height(image_height), .new_x(A_x_temp1),
                                                                .new_y(A_y_temp1), .new_width(A_width_temp1),
.new_height(A_height_temp1));

        GetRotatedParams B1(.clk(clk), .reset(reset), .x(B_x), .y(B_y), .width(B_width),
                                                                .height(B_height), .image_x(image_x),
.image_y(image_y),
                                                                .image_width(image_width),
.image_height(image_height), .new_x(B_x_temp1),
                                                                .new_y(B_y_temp1), .new_width(B_width_temp1),
.new_height(B_height_temp1));

        GetRotatedParams C1(.clk(clk), .reset(reset), .x(C_x), .y(C_y), .width(C_width),
                                                                .height(C_height), .image_x(image_x),
.image_y(image_y),
                                                                .image_width(image_width),
.image_height(image_height), .new_x(C_x_temp1),
                                                                .new_y(C_y_temp1), .new_width(C_width_temp1),
.new_height(C_height_temp1));

        // 180 deg CCW
        GetRotatedParams A2(.clk(clk), .reset(reset), .x(A_x_temp1), .y(A_y_temp1), .width(A_width_temp1),
                                                                .height(A_height_temp1), .image_x(image_x),
.image_y(image_y),
                                                                .image_width(image_height),
.image_height(image_width), .new_x(A_x_temp2),
                                                                .new_y(A_y_temp2), .new_width(A_width_temp2),
.new_height(A_height_temp2));

        GetRotatedParams B2(.clk(clk), .reset(reset), .x(B_x_temp1), .y(B_y_temp1), .width(B_width_temp1),
                                                                .height(B_height_temp1), .image_x(image_x),
.image_y(image_y),
                                                                .image_width(image_height),
.image_height(image_width), .new_x(B_x_temp2),
                                                                .new_y(B_y_temp2), .new_width(B_width_temp2),
.new_height(B_height_temp2));

        GetRotatedParams C2(.clk(clk), .reset(reset), .x(C_x_temp1), .y(C_y_temp1), .width(C_width_temp1),
                                                                .height(C_height_temp1), .image_x(image_x),
.image_y(image_y),
                                                                .image_width(image_height),
.image_height(image_width), .new_x(C_x_temp2),
                                                                .new_y(C_y_temp2), .new_width(C_width_temp2),
.new_height(C_height_temp2));

        // 270 deg CCW
        GetRotatedParams A3(.clk(clk), .reset(reset), .x(A_x_temp2), .y(A_y_temp2), .width(A_width_temp2),
                                                                .height(A_height_temp2), .image_x(image_x),
.image_y(image_y),
                                                                .image_width(image_width),
.image_height(image_height), .new_x(A_x_temp3),
                                                                .new_y(A_y_temp3), .new_width(A_width_temp3),
.new_height(A_height_temp3));

        GetRotatedParams B3(.clk(clk), .reset(reset), .x(B_x_temp2), .y(B_y_temp2), .width(B_width_temp2),
                                                                .height(B_height_temp2), .image_x(image_x),
.image_y(image_y),
                                                                .image_width(image_width),
.image_height(image_height), .new_x(B_x_temp3),
                                                                .new_y(B_y_temp3), .new_width(B_width_temp3),
.new_height(B_height_temp3));

        GetRotatedParams C3(.clk(clk), .reset(reset), .x(C_x_temp2), .y(C_y_temp2), .width(C_width_temp2),
                                                                .height(C_height_temp2), .image_x(image_x),
.image_y(image_y),
                                                                .image_width(image_width),
.image_height(image_height), .new_x(C_x_temp3),
                                                                .new_y(C_y_temp3), .new_width(C_width_temp3),
.new_height(C_height_temp3));
```

```
	always @(posedge clk) begin
		done_rotating <= 0;
		if (rotate) begin
			done_rotating <= 1;
//			image_width_new <= image_height;
//			image_height_new <= image_width;

			if (corner1 == TL) begin
				if (corner2 == BL) begin
					image_width_new <= image_height;
					image_height_new <= image_width;
					A_x_new <= A_x_temp1;
					A_y_new <= A_y_temp1;
					A_width_new <= A_width_temp1;
					A_height_new <= A_height_temp1;
					B_x_new <= B_x_temp1;
					B_y_new <= B_y_temp1;
					B_width_new <= B_width_temp1;
					B_height_new <= B_height_temp1;
					C_x_new <= C_x_temp1;
					C_y_new <= C_y_temp1;
					C_width_new <= C_width_temp1;
					C_height_new <= C_height_temp1;
				end
				else if (corner2 == BR) begin
					A_x_new <= A_x_temp2;
					A_y_new <= A_y_temp2;
					A_width_new <= A_width_temp2;
					A_height_new <= A_height_temp2;
					B_x_new <= B_x_temp2;
					B_y_new <= B_y_temp2;
					B_width_new <= B_width_temp2;
					B_height_new <= B_height_temp2;
					C_x_new <= C_x_temp2;
					C_y_new <= C_y_temp2;
					C_width_new <= C_width_temp2;
					C_height_new <= C_height_temp2;
				end
				else if (corner2 == TR) begin
					image_width_new <= image_height;
					image_height_new <= image_width;
					A_x_new <= A_x_temp3;
					A_y_new <= A_y_temp3;
					A_width_new <= A_width_temp3;
					A_height_new <= A_height_temp3;
					B_x_new <= B_x_temp3;
					B_y_new <= B_y_temp3;
					B_width_new <= B_width_temp3;
					B_height_new <= B_height_temp3;
					C_x_new <= C_x_temp3;
					C_y_new <= C_y_temp3;
					C_width_new <= C_width_temp3;
					C_height_new <= C_height_temp3;
				end
			end

			else if (corner1 == BL) begin
				if (corner2 == BR) begin
					image_width_new <= image_height;
					image_height_new <= image_width;
					A_x_new <= A_x_temp1;
					A_y_new <= A_y_temp1;
					A_width_new <= A_width_temp1;
					A_height_new <= A_height_temp1;
					B_x_new <= B_x_temp1;
					B_y_new <= B_y_temp1;
					B_width_new <= B_width_temp1;
					B_height_new <= B_height_temp1;
					C_x_new <= C_x_temp1;
					C_y_new <= C_y_temp1;
					C_width_new <= C_width_temp1;
					C_height_new <= C_height_temp1;
				end
				else if (corner2 == TR) begin
					A_x_new <= A_x_temp2;
					A_y_new <= A_y_temp2;
					A_width_new <= A_width_temp2;
					A_height_new <= A_height_temp2;
					B_x_new <= B_x_temp2;
					B_y_new <= B_y_temp2;
					B_width_new <= B_width_temp2;
					B_height_new <= B_height_temp2;
					C_x_new <= C_x_temp2;
					C_y_new <= C_y_temp2;
```

```
                C_width_new <= C_width_temp2;
                C_height_new <= C_height_temp2;
        end
        else if (corner2 == TL) begin
                image_width_new <= image_height;
                image_height_new <= image_width;
                A_x_new <= A_x_temp3;
                A_y_new <= A_y_temp3;
                A_width_new <= A_width_temp3;
                A_height_new <= A_height_temp3;
                B_x_new <= B_x_temp3;
                B_y_new <= B_y_temp3;
                B_width_new <= B_width_temp3;
                B_height_new <= B_height_temp3;
                C_x_new <= C_x_temp3;
                C_y_new <= C_y_temp3;
                C_width_new <= C_width_temp3;
                C_height_new <= C_height_temp3;
        end
end

else if (corner1 == BR) begin
        if (corner2 == TR) begin
                image_width_new <= image_height;
                image_height_new <= image_width;
                A_x_new <= A_x_temp1;
                A_y_new <= A_y_temp1;
                A_width_new <= A_width_temp1;
                A_height_new <= A_height_temp1;
                B_x_new <= B_x_temp1;
                B_y_new <= B_y_temp1;
                B_width_new <= B_width_temp1;
                B_height_new <= B_height_temp1;
                C_x_new <= C_x_temp1;
                C_y_new <= C_y_temp1;
                C_width_new <= C_width_temp1;
                C_height_new <= C_height_temp1;
        end
        else if (corner2 == TL) begin
                A_x_new <= A_x_temp2;
                A_y_new <= A_y_temp2;
                A_width_new <= A_width_temp2;
                A_height_new <= A_height_temp2;
                B_x_new <= B_x_temp2;
                B_y_new <= B_y_temp2;
                B_width_new <= B_width_temp2;
                B_height_new <= B_height_temp2;
                C_x_new <= C_x_temp2;
                C_y_new <= C_y_temp2;
                C_width_new <= C_width_temp2;
                C_height_new <= C_height_temp2;
        end
        else if (corner2 == BL) begin
                image_width_new <= image_height;
                image_height_new <= image_width;
                A_x_new <= A_x_temp3;
                A_y_new <= A_y_temp3;
                A_width_new <= A_width_temp3;
                A_height_new <= A_height_temp3;
                B_x_new <= B_x_temp3;
                B_y_new <= B_y_temp3;
                B_width_new <= B_width_temp3;
                B_height_new <= B_height_temp3;
                C_x_new <= C_x_temp3;
                C_y_new <= C_y_temp3;
                C_width_new <= C_width_temp3;
                C_height_new <= C_height_temp3;
        end
end
else if (corner1 == TR) begin
        if (corner2 == TL) begin
                image_width_new <= image_height;
                image_height_new <= image_width;
                A_x_new <= A_x_temp1;
                A_y_new <= A_y_temp1;
                A_width_new <= A_width_temp1;
                A_height_new <= A_height_temp1;
                B_x_new <= B_x_temp1;
                B_y_new <= B_y_temp1;
                B_width_new <= B_width_temp1;
                B_height_new <= B_height_temp1;
                C_x_new <= C_x_temp1;
                C_y_new <= C_y_temp1;
                C_width_new <= C_width_temp1;
                C_height_new <= C_height_temp1;
```

```
                                end
                                else if (corner2 == BL) begin
                                        A_x_new <= A_x_temp2;
                                        A_y_new <= A_y_temp2;
                                        A_width_new <= A_width_temp2;
                                        A_height_new <= A_height_temp2;
                                        B_x_new <= B_x_temp2;
                                        B_y_new <= B_y_temp2;
                                        B_width_new <= B_width_temp2;
                                        B_height_new <= B_height_temp2;
                                        C_x_new <= C_x_temp2;
                                        C_y_new <= C_y_temp2;
                                        C_width_new <= C_width_temp2;
                                        C_height_new <= C_height_temp2;
                                end
                                else if (corner2 == BR) begin
                                        image_width_new <= image_height;
                                        image_height_new <= image_width;
                                        A_x_new <= A_x_temp3;
                                        A_y_new <= A_y_temp3;
                                        A_width_new <= A_width_temp3;
                                        A_height_new <= A_height_temp3;
                                        B_x_new <= B_x_temp3;
                                        B_y_new <= B_y_temp3;
                                        B_width_new <= B_width_temp3;
                                        B_height_new <= B_height_temp3;
                                        C_x_new <= C_x_temp3;
                                        C_y_new <= C_y_temp3;
                                        C_width_new <= C_width_temp3;
                                        C_height_new <= C_height_temp3;
                                end
                        end
                end
                else begin
                        image_width_new <= image_width;
                        image_height_new <= image_height;
                        A_x_new <= A_x;
                        A_y_new <= A_y;
                        A_width_new <= A_width;
                        A_height_new <= A_height;
                        B_x_new <= B_x;
                        B_y_new <= B_y;
                        B_width_new <= B_width;
                        B_height_new <= B_height;
                        C_x_new <= C_x;
                        C_y_new <= C_y;
                        C_width_new <= C_width;
                        C_height_new <= C_height;
                end

        end


endmodule
```