ACTION TRACKING SYSTEM

Shubhang Chaudhary, Srinivasan Raghavendra

6.111 final project report

December 13,2011

ABSTRACT

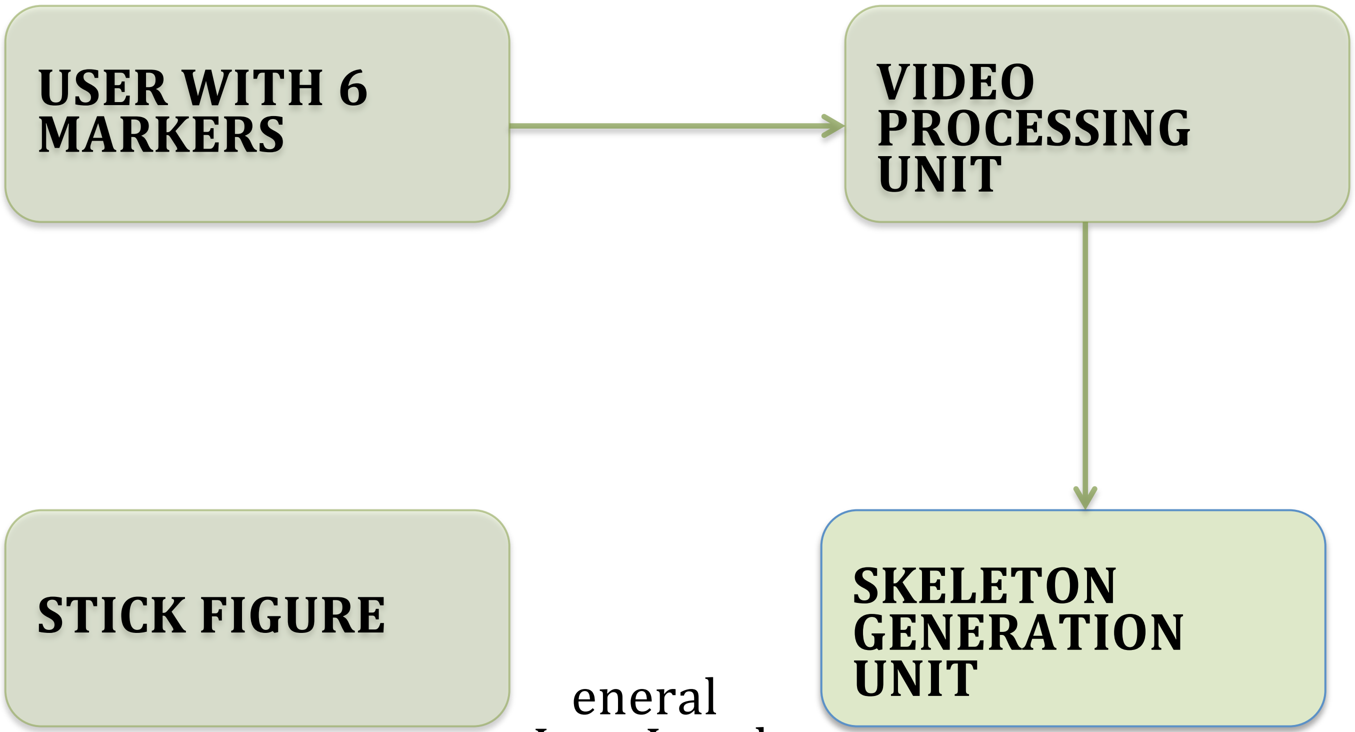
Using the National Television System Committee-standard (NTSC) camera for providing the feedback, the Field Programmable Gate Array (FPGA) can be used to depict a basic animation in the form of a Stick figure that would mimic a human performing real time actions. Our project can basically be divided into two high level tasks, namely-Visual perception and feedback and graphic generation on the screen.

We will begin by using the color markers, which will be placed on the human, indicating the joints. We will try to analyze the proper colors and intensities of light at which they could be detected and then implement the algorithm to combine all the modules logically. The stick figure being displayed will not be an actual image of the person, but rather an intuitive indication of movements performed.

This project has a potential for great learning experience and also has a wide variety of applications in Gaming and animation industry and the Study of Dynamics of anything that can move.

TABLE OF CONTENTS

1.1) GENERAL DESCRIPTION



eneral Low Level description of the system:

1. The user would provide the six markers as inputs to the video processing unit

2. The video processing unit would generate six distinct points indicating 2 for hands, 2 for legs and the 2 points for upper and lower torso.

3. Skeleton generation unit connects these points to result in a beautiful stick figure

2.1 OVERVIEW

The **Action Tracking System (ATS)** is a project using FPGA to achieve real time video tracking with position feedbacks from an NTSC camera. Video tracking is the process of locating a moving object (or multiple objects) over time using a camera. It has a variety of uses, some of which are: **human-computer interaction, security and surveillance, video communication and compression, augmented reality, traffic control, medical imaging and video editing.** Video tracking can be a time consuming process due to the amount of data that is contained in video. Adding further to the complexity is the possible need to use object recognition techniques for tracking.

The objective of **video tracking** is to associate target objects in consecutive video frames. The association can be especially difficult when **the objects are moving fast relative to the frame rate**. Another situation that increases the complexity of the problem is when the tracked object changes orientation over time. For these situations video tracking systems usually employ a motion model which describes how the image of the target might change for different possible motions of the object.

The system uses one camera to obtain a straight view of the field, where the target human will be**. Using the 2-D information** from the camera, an object recognition module will perform **threshold filtering to track the positions** of various joints of the human in the view of the camera. The joints being tracked are used to reproduce a Stick figure of the human on the screen. The Action Tracking system, also involves the usage of artificial intelligence to enumerate the right connections between the joints being tracked. More information on the algorithm is given in the points_decoder module.

2.1.1 BACKGROUND RESEARCH

The idea of tracking motion on a computer using a video camera has been around a couple of decades, and still is not fully perfect, because the construction of vision is a complex subject. We don't just "see"; we construct colors, edges, objects, depth, and other aspects of vision from the light that reaches our retinas. If we want to program a computer to see in the same way, it has to have subroutines that define the characteristics of vision and allow it to distinguish those characteristics in the array of pixels that comes from a camera.

There are a number of toolkits available for getting data from a camera and manipulating it. They vary from very high-level simple graphical tools to low-level tools that allow us to manipulate the pixels directly. Regardless of our application, the first step is always the same: we get the pixels from the camera in an array of numbers, one frame at a time, and do things with the array. Typically, our array is a list of numbers, including the location, and the relative levels or red, green, and blue light at that location

2.1.2 APPLICATIONS

There are a few popular applications that people tend to develop when they attach a camera to a computer:

2.1.2.1 **Video manipulation** takes the image from the camera, changes it somehow, and re-presents it to the viewer in changed form. In this case, the computer doesn't need to be able to interpret objects in the image, because we are basically just applying filters.

2.1.2.2 **Tracking** looks for a blob of pixels that's unique, perhaps the brightest blob, or the reddest blob, and tracks its location over a series of frames. Tracking can be complicated, because the brightest blob from one frame to another might not be produced by the same object.

2.1.2.3 **Object recognition** looks for a blob that matches a particular pattern, like a face, identifies that blob as an object, and keeps track of its location over time. Object recognition is the hardest of all three applications, because it involves both tracking and pattern recognition. If the object rotates, or if its colors shift because of a lighting change, or it gets smaller as it moves away from the camera, the computer has to be programmed to compensate. If it's not, it may fail to "see" the object, even though it's still there.

2.1.3 TECHNICAL APPROACH

2.1.3.1 Video Processing

At the most basic level, we will be obtaining a pixel's position, and its color .From those facts, other information can be determined:

- The brightest pixel can be determined by seeing which pixel has the highest color values;
- A "blob" of color can be determined by choosing a starting color, setting a range of variation, and checking the neighboring pixels of a selected pixel to see if they are in the range of variation.
- Areas of change can be determined by comparing one frame of video with a previous frame, and seeing which pixels have the most significantly different color values.
- Areas of pattern can be followed by selecting an area to track, and continuing to search for areas that match the pattern of pixels selected. Again, a range of variation can be set to allow for "fuzziness".

These approaches will be followed to process the video data and calculate the center of masses of the color markers placed on the Human.

2.1.3.2 Image Reproduction

This part of the system is pretty easy when compared to the video processing part. To make the whole system interesting, we plan to make use of six markers of the same color.

This poses a big challenge and we solve it by using artificial intelligence. The system knows the various properties of the processed video output and so it makes use of these properties to figure out the right dots to connect to successfully reproduce the image of the human in the form of a stick figure.

The properties are encoded into the system in the form of a module by taking into account various constraints. The system assumes that-

- Human cannot cross his/her hands or legs
- Human cannot turn around as this a 2D motion tracking.
- Human cannot put his/her hands below his/her legs.
- Human cannot join their hands/legs.

The reproduction of the image of the human is done in the form of a stick figure and more details on the algorithm are given in the module. The image reproduction also implements antialiasing of the stick figure, to make it seem more lively.

3.1.1 INFORMATION FROM MODULES

- **video_decoder** : This module takes signals from a NTSC camera and decodes that information into YCrCb that is 30 bits long.

- **ntsc_to_ZBT** : stores pixel information as RGB into the on-board ZBT memory. This module also has the RGB to HSV converter.

- **threshold_filter** : filters the incoming pixels with color thresholds in order to find the position of the human joints.

- **line_drawer** : Takes 2 points in 2---D space as the inputs and connects them with a line of desired colour. It also implements anti aliasing of the line.
- **Points_Decider** : With the position information of the human joints and artificial intelligence, the dots_connector module decides the respective arm, leg and torso points.
- **Connector** : This module takes the arm ,leg and torso information from the points_decider module and instantiates the line_drawer module multiple times to make a meaningful skeleton.
- **Buffer** : This module store the line points on the ZBT memory and makes sure that the pixels in the output do not flicker.

3.1.2 DETAILED EXPLANATION OF MODULES

3.1.2.1 Video_decoder (Shubhang): This module's job is to convert the input signals from the camera and output a 30---bit YCrCb value along with other signals such as field, vsync and hsync.

This module

- Interfaces with the NTSC camera.
- Decodes the input signal and converts the pixel information into a 30 bit value, with 10 bits each for Y (luma), Cr (red difference chroma) and Cb (blue difference chroma).
- Since NTSC video standard implements interlacing mechanism, this module also outputs 'f' as a one bit value to indicate whether it is in odd or even field.
- One bit values, 'v' and 'h' are outputted to show the position of that pixel.
- It outputs a data_valid bit to tell other modules that the data is ready for use.

3.1.2.2 ntsc_to_zbt (Shubhang): This module is responsible for generating corresponding address in ZBT memory to store the current filtered incoming pixel information. Since the camera uses a different clock frequency, this module is the entry point to synchronize data from the camera with the rest of the system. A sample module is already written, but changes have to be made in order for it to store colored pixels. Input into A 24bit RGB String. Such module is already written, but we note that this module has a 5 clock cycle delay between inputs and Outputs. Therefore, the address into the ZBT memory should be delayed appropriately in order to store in the correct location.

3.1.2.3 Threshold-Filter(Shubhang): This is the heart of the system. The objective of this module is to do the most difficult part of the system, that is, image processing. This module processes every frame it receives from the ntsc_zbt module and figures out the existence of a reference image, in the frame. The reference image is the image of a color marker. The outputs of this module are the center of masses of various color markers. Making this module is going to be a big challenge as all the color markers are of the same color. This module uses most of the computation and memory available. It also has the highest latency.This filter uses the mechanism of grids to find the center of mass of the detected red marker.The marker that gets detected, satisfies the condition of minimum number of pixels per block that should be within a range of the HSV values.This block also has the special mechanism of filtering the appearance and occurance of the red pixels in adjacent blocks.It uses the concept of tracking the previous blocks for red,and if the condition satisfies then the current block never provides the center of mass to the line drawing module.

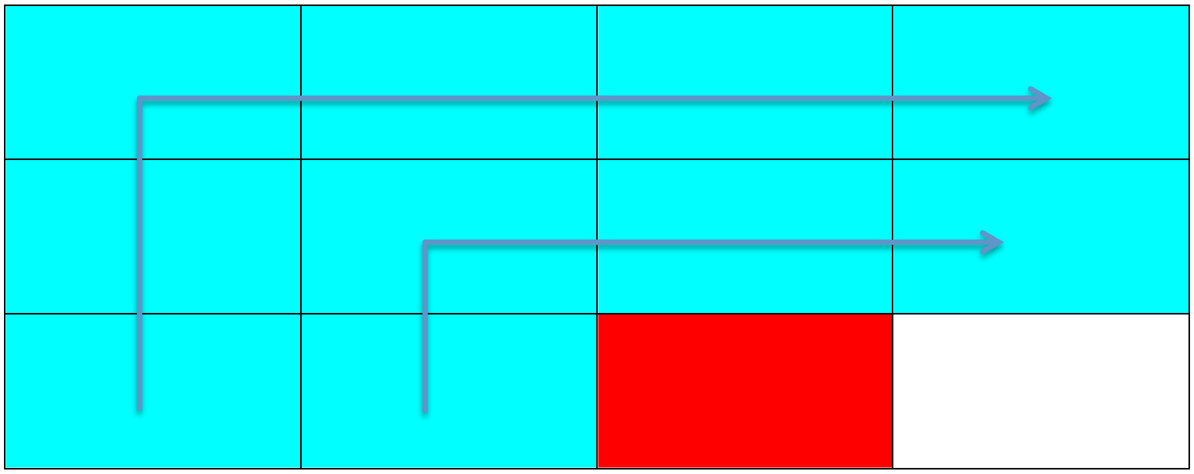


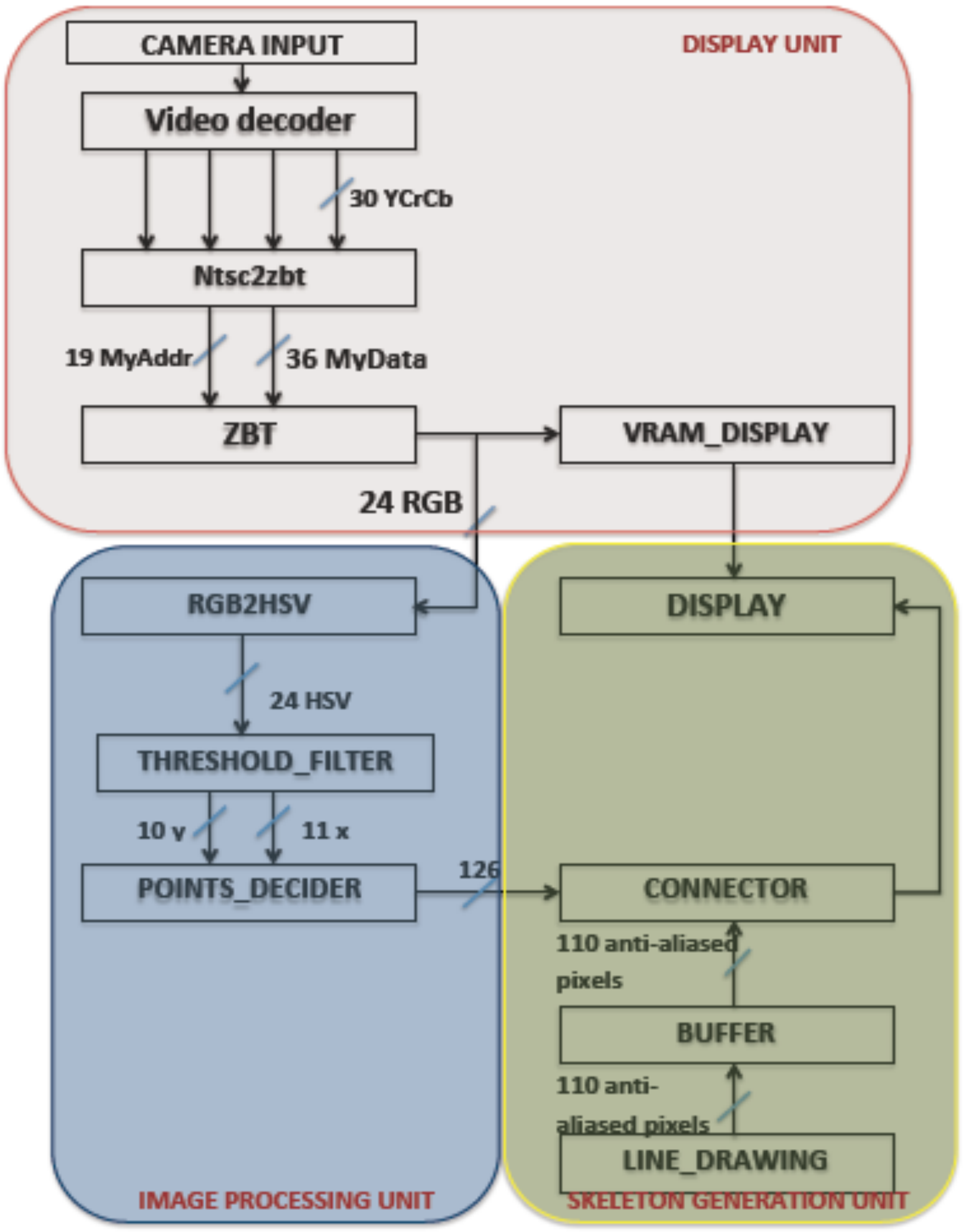
Figure 3: grid detection

3.1.2.4 Line-drawer: The line_decoder module implements Jack Bresenham's line drawing algorithm, described at http://en.wikipedia.org/wiki/Bresenham%27s line algorithm This algorithm draws a straight line between any two points on the screen, it is a particularly good choice because its implementation does not require any division. This module also implements antialiasing of a line, to make the line look more natural. The module's inputs are the two endpoints of the line segment it will draw, as well as a ready signal from the Connector module. Its outputs are the pixels to be written to the offscreen buffer, a write_enable signal to the buffer, and a ready signal back to the connector module.

The timing of the ready signals is as follows. When the line_drawer module is not in the process of drawing a segment, it asserts its ready signal. When the connector module sees this, it proceeds to iterate through entities and segments until it finds one that is not ignored. Once it finds one, it sends out the segment's data along with a one---cycle long ready signal.Upon receiving this, the line_drawer module takes down its ready signal until it finishes processing the segment.

3.1.2.5 Points_Decider: This is the brain of the system. It receives 6 points from the threshold_filter module and figures out the right points to be joined to make a meaningful skeleton on the screen. This module assumes a few constraints- the human cannot cross hands, cross legs, put hi/her hands below his/her legs and finally cannot turn around. This modules functions using artificial intelligence, it uses the fact that the torso of the human is always situated in the center of the left and right side of the body. It has a latency of 8 clock cycles and does not have any start or stop signal as the buffer in the threshold_filter module makes sure that all the points are synchronized. Making this module was of a fair amount of challenge as there are a lot of possible combinations of orientations of hands and legs.

3.1.2.6 Double Buffer: The buffer module allows random-access pixels to be displayed to the screen without needing huge arrays of combinational logic and without causing flicker due to erased pixels. It physically interfaces with the Virtex2's two BRAM memories, clearing and then writing to one while the other is displayed on-screen. This module is synchronized with the SVGA module's hcount, vcount, and vsync signals and operates on a two-frame period, bounded by vsync. During the first frame of the period, hcount and vcount are used as indices to the off-
- screen buffer and the value at each address is set to zer
o. During the second frame, the module accepts pixel coo
rdinates and color values from the line_drawer module an
d writes them
to the offscreen buffer. During both frames, hcount
and vcount also index the on--screen buffer, the
contents of which ae sent to the VGA interface. Upon the
rising edge of the vsync signal, the off--
screen buffer becomes on--screen and vice



SYSTEM BLOCK DIAGRAM

4.1 TESTING AND DEBUGGING

Since most of our project is reception and representation of visual data, we had to do most of our testing and debugging on the screen and the logic analyzer. Every Module mentioned above has several modules that were created as test fixtures, created for specific purpose of testing.

For Simulating most of our modules, we used Modelsim and since most of our modules require hcount , vcount, as parameters, we had to recreate them in out simulations. We also had to scale our timing as the timescale was not good enough for testing what could be happening in a complete frame. The time was scale down by 1000.

Threshold Filter

Although every module is a challenge in its own way, the biggest challenge that we faced was the threshold filter module, which basically requires real time testing and cannot be completely tested in any of the available testing devices. First, we had to make sure that the video data was buffered and outputted on the screen correctly. To test the change from YCrCb to RGB and HSV, we used the computer monitor to detect if we had the right red, green, and blue signals. HSV was harder to test; we ended up putting the hue to all the red, green, and blue signals, to see which objects had the greatest hue. While changing the video from YCrCb, RGB, and HSV, we used different shapes and colors to find what color and object would be best for our threshold filter module to detect. We also had to solve a the problem of detection of multiple points adjacent to each other. This had a trade off with the accuracy of the center of mass being detected. At first, we used LED lights, but soon found that the video camera could only detect it if it was not flashed directly at the video camera. This was quite cumbersome.

Few tests for threshold filter were,

1) We model simulated the results of particular grids being detected for accurate results.

2) We generated the cursors, which would give out the HSV values for the particular pixel present on the screen.
3) These cursors were used for the calibrating the system with new colors.
4) Then we also had a color palette because of which we could decide that Hue, saturation and value detection was perfect way to detect the colors.
5) We also analyzed the HSV values for various colors. It was very easy only to catch RED and YELLOW in the light conditions we were working under.

Double Buffer

This is also one of those modules which gave a hard time while testing. Since this module uses 2 BRAMs we had to test this module by writing a predefined pattern into the BRAMs ,reading it back and checking if the desired output is acquired.

We also had a switch to indicate the BRAM to which the data was being written and an LED to indicate the BRAM from which data is being read from.

The output which was read back was displayed on the screen. Initially we had a few problems and glitches but, eventually with the combination of screen and the logic analyzer, we were able to attain the complete working status of the BRAM.

LINE DRAWER AND POINTS DECIDER

Since these modules are a combination of algorithms, arithmetic and logic, it was quite simple to test these modules and get them working. The primary tool that was used to test these modules was modelsim.

However, we were able to emphasize their working status after combining them with the double buffer module and drawing lines on the screen.

CONCLUSION

Our Primary conclusion would be – “ We had Fun and we learnt a Ton”.

By the end of the project we had a system ready that would detect 6 color markers of the same color (RED in this case) and then the coordinates of these points would then reach the appropriate module to draw a human looking stick figure appear on the screen.
The stick figure showed the following characterstics:

1) Followed the actions of the real time users with a decent amount of accuracy.
2) The six points were almost all the time being detected.
3) The calculation of the six points, their allotment to respective parts (leg,hand,torso) and drawing the stick figure was doneon the screen with a high level of accuracy.

Some aspects that we had previously thought would be trivial parts of the total project, such as displaying the video and detecting the six red points, turned out to be more challenging than expected. Initially we decided to implement the detection of different colors as the markers on the human body. But to make the project more innovative and practically challenging we changed the idea and were successfully able to detect the six points with the same color ,which we believe was a great initiative. .However, the integration took longer than expected due to a small logic error which became harder to find once the system consisted of all modules combined and interacting.

We were able to successfully accomplish the task of making the stick figure on the screen detect the actions of the human with markers but with a few glitches due to the inefficiency color detection process in the highly noisy environment.

A Appendices

A.1 Top Level Verilog

```
///////////////////////////////////////////////////////////////////////////////
//
// 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)
//
// 2009-May-11: Fixed memory management bug by 8 clock cycle forecast.
//              Changed resolution to 1024 * 786 was ... 800 * 600.
//              Reduced clock speed to 40MHz.
//              Disconnected zbt_6111's ram_clk signal.
//              Added ramclock to control RAM.
//              Added notes about ram1 default values.
//              Commented out clock_feedback_out assignment.
//              Removed delayN modules because ZBT's latency has no more effect.
//
// 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;

  //These values has to be set to 0 like ram0 if ram1 is used.
  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;

  //clock_feedback_out will be assigned by ramclock
  //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
  wire clock_65mhz_unbuf,clock_65mhz;
  wire [24:0] x_com_op, x_accumulated_op, counted_value_op;
   // Logic Analyzer
   //assign analyzer1_data = 16'h0;
   //assign analyzer1_clock = 1'b1;
          assign analyzer1_data = {clock_65mhz,x_com_op[10:0],4'b0000};
   assign analyzer1_clock = clock_65mhz;
   //assign analyzer3_data = 16'h0;
   //assign analyzer3_clock = 1'b1;
          assign analyzer3_data = {counted_value_op[15:0]};
   assign analyzer3_clock = clock_65mhz;
   assign analyzer2_data = 16'h0;
   assign analyzer2_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)

   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;  // use ramclock source

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

   // use FPGA's digital clock manager to produce a
   // 40MHz clock (actually 40.5MHz)
   wire clock_40mhz_unbuf,clock_40mhz;
   DCM vclk1(.CLKIN(clock_27mhz),.CLKFX(clock_40mhz_unbuf));
   // synthesis attribute CLKFX_DIVIDE of vclk1 is 2
   // synthesis attribute CLKFX_MULTIPLY of vclk1 is 3
   // synthesis attribute CLK_FEEDBACK of vclk1 is NONE
   // synthesis attribute CLKIN_PERIOD of vclk1 is 37
   BUFG vclk2(.O(clock_40mhz),.I(clock_40mhz_unbuf));
   wire clk = clock_40mhz;

*/
         wire locked;

//       assign clock_feedback_out = 0;

   ramclock rc(.ref_clock(clock_65mhz), .fpga_clock(clk),
                                                   .ram0_clock(ram0_clk),
                                                   //.ram1_clock(ram1_clk),   //uncomment if
ram1 is used
                                                   .clock_feedback_in(clock_feedback_in),
                                                   .clock_feedback_out(clock_feedback_out),
.locked(locked));


   // 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);
         debounce db2(power_on_reset, clk, ~button0, db_button0);
         debounce db3(power_on_reset, clk, ~button1, db_button1);
         debounce db4(power_on_reset, clk, ~button2, db_button2);
         debounce db5(power_on_reset, clk, ~button3, db_button3);
         debounce db6(power_on_reset, clk, ~button_up, db_button_up);
         debounce db7(power_on_reset, clk, ~button_down, db_button_down);
         debounce db8(power_on_reset, clk, ~button_left, left);
         debounce db9(power_on_reset, clk, ~button_right, right);

   assign reset = user_reset | power_on_reset;

   // display module for debugging
         reg [23:0] HSVdisp;
   reg [63:0] dispdata;
         reg [23:0] dum_HSV;
         wire [23:0] RGB;
         wire [20:0] xy1,xy2,xy3,xy4,xy5,xy6,xy7,xy8;

         reg [7:0]          hewlow = 8'hE0;
         //assign hk=HSVdisp[23:16];
         //assign sk=HSVdisp[15:8];
         //assign vk=HSVdisp[7:0];
   /*display_16hex hexdisp1(reset, clk, dispdata,
                              disp_blank, disp_clock, disp_rs, disp_ce_b,
                         disp_reset_b, disp_data_out);*/
        //wire [7:0] hk,sk,vk;
        reg [10:0] dummy1 = 1;
  reg [9:0] dummy2 = 1;
        display_16hex hexdisp2(reset,
clock_27mhz,{1'b0,xy1[20:10],2'b0,xy1[9:0],HSVdisp[23:16],4'b0,HSVdisp[15:8],4
'b0,HSVdisp[7:0],8'b0},
                                                                      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);

  // 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;

  wire ram0_clk_not_used;
  zbt_6111 zbt1(clk, 1'b1, vram_we, vram_addr,
                vram_write_data, vram_read_data,
                ram0_clk_not_used,          //to get good timing, don't
connect ram_clk to zbt_6111
                ram0_we_b, ram0_address, ram0_data, ram0_cen_b);

  // generate pixel value from reading ZBT memory
  //wire [7:0]    vr_pixel;
        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));

         YCrCb2RGB convert( RGB[23:16], RGB[15:8], RGB[7:0],
tv_in_line_clock1,
                                                      1'b0, ycrcb[29:20],
ycrcb[19:10], ycrcb[9:0] );

   // code to write NTSC data to video memory

   wire [18:0] ntsc_addr;
   wire [35:0] ntsc_data;
   wire        ntsc_we;
         reg [17:0]       pixel;
   ntsc_to_zbt n2z (clk,
                                                tv_in_line_clock1,
                                                fvh,
                                                dv,

         {RGB[23:18],RGB[15:10],RGB[7:2]},
                                                ntsc_addr,
                                                ntsc_data,
                                                ntsc_we,
                                                switch[6]);
         wire [23:0] HSV;
         // RGB to hsv module instantiation
         /*rgb2hsv converter(.clock(clk),.reset(0), .r(RGB[23:16]),
                         .g(RGB[15:8]), .b(RGB[7:0]), .h(HSV[23:16]),
.s(HSV[15:8]),
                         .v(HSV[7:0]));*/
         //RGB values directly come from the ZBT in the instantiation
         rgb2hsv converter(.clock(clk),
                                                .reset(0),
                                                .r({pixel[17:12],2'b0}),
                                                .g({pixel[11:6],2'b0}),
                                                .b({pixel[5:0],2'b0}),
                                                .h(HSV[23:16]),
                                                .s(HSV[15:8]),
                                                .v(HSV[7:0]));

   // 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;
        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;

  // select output pixel data


  reg   b,hs,vs,b1,hs1,vs1,b2,hs2,vs2;
        reg v1;
        wire vs3,hs3,b3;
  always @(posedge clk)
  begin
        pixel <= switch[0] ? {hcount[8:6],5'b0} : vr_pixel;
        b1 <= blank;
        b2 <= b1;
        b <= b2;
        hs1 <= hsync;
        hs2 <= hs1;
        hs <= hs2;
        vs1 <= vsync;
        vs2 <= vs1;
        vs <= vs2;
        end
        wire [17:0] pixel1;
        delayV #(.NDELAY(22),.SIZE(1)) delayhsync(.clk(clk), .in(hs), .out(hs3));
        delayV #(.NDELAY(22),.SIZE(1)) delayvsync(.clk(clk), .in(vs), .out(vs3));
        delayV #(.NDELAY(22),.SIZE(1)) delayblank(.clk(clk), .in(b), .out(b3));
        delayV #(.NDELAY(22),.SIZE(18)) delaypixel(.clk(clk), .in(pixel),
.out(pixel1));
/*// for RGB color detection
        always @(posedge clk) begin
		v1 <= vsync;
	end
	reg [5:0] index;
	reg [13:0] Rtotal;
	reg [13:0] Gtotal;
	reg [13:0] Btotal;
	always@(posedge clk) begin
		if(!v1 && vsync) begin
			if ((dummy1 > 0) && (dummy1 < 1024) && (dummy2 > 0)
&& (dummy2 < 768)) begin
				if (!button_up && dummy1<=1024) begin
				dummy1 <= dummy1 + 1;
				end
				if (!button_down && dummy1 >=1) begin
				dummy1 <= dummy1 - 1;
				end
				if (!button_right && dummy2<=768) begin
				dummy2 <= dummy2 + 1;
				end
				if (!button_left && dummy2 >=1) begin
				dummy2 <= dummy2 - 1;
				end
			end
		end
		if(hcount == dummy1 && vcount == dummy2 && db_button0)
begin
		index <= index + 1;
		Rtotal <= (index == 63) ? 0 : (Rtotal + RGB[23:16]);
		if(index == 63) HSVdisp[23:16] <= (Rtotal/64);
		Gtotal <= (index == 63) ? 0 : (Gtotal + RGB[15:8]);
		if(index == 63) HSVdisp[15:8] <= (Gtotal/64);
		Btotal <= (index == 63) ? 0 : (Btotal + RGB[7:0]);
		if(index == 63) HSVdisp[7:0] <= (Btotal/64);
		end
	end
	//changes happen till here	*/
	// for HSV color detection
	always @(posedge clk) begin
		v1 <= vsync;
	end
	reg [20:0] xy1val;
	reg [5:0] index;
	reg [3:0] index1;
	reg [36:0] xy1total;
	reg [13:0] Rtotal,Htotal;
	reg [13:0] Gtotal,Stotal;
	reg [13:0] Btotal,Vtotal;
	reg [10:0] x1_avg,x2_avg,x3_avg,x4_avg,x5_avg,x6_avg,x7_avg;
	reg [9:0] y1_avg,y2_avg,y3_avg,y4_avg,y5_avg,y6_avg,y7_avg;
	reg [24:0] tot_x1,tot_x2,tot_x3,tot_x4,tot_x5,tot_x6;
	reg [24:0] tot_y1,tot_y2,tot_y3,tot_y4,tot_y5,tot_y6;
	always@(posedge clk) begin
		//logic for displaying the cursor
		if(!v1 && vsync) begin
			if ((dummy1 > 0) && (dummy1 < 1024) && (dummy2 > 0)
&& (dummy2 < 768)) begin
				if (!button_up && dummy1<=1024) begin
				dummy1 <= dummy1 + 1;
				end
				if (!button_down && dummy1 >=1) begin
				dummy1 <= dummy1 - 1;
				end
				if (!button_right && dummy2<=768) begin
				dummy2 <= dummy2 + 1;
				end
				if (!button_left && dummy2 >=1) begin
				dummy2 <= dummy2 - 1;
				end
			end
		end
		//for averaging the HSV value at 64 counts
		if(hcount == dummy1 && vcount == dummy2) begin // &&
db_button0) begin
		index <= index + 1;
		Htotal <= (index == 63) ? 0 : (Htotal + HSV[23:16]);
		if(index == 63) HSVdisp[23:16] <= (Htotal/64);
		Stotal <= (index == 63) ? 0 : (Stotal + HSV[15:8]);
		if(index == 63) HSVdisp[15:8] <= (Stotal/64);
		Vtotal <= (index == 63) ? 0 : (Vtotal + HSV[7:0]);
		if(index == 63) HSVdisp[7:0] <= (Vtotal/64);
		end
	end
	//changes happen till here
	// VGA Output.  In order to meet the setup and hold times of the
  // AD7125, we send it ~clk.
	/*
	//for the display of RGB pixels from the ZBT on the screen
  //assign vga_out_red = {pixel[17:12],2'b0};
	//assign vga_out_green = {pixel[11:6],2'b0};
	//assign vga_out_blue = {pixel[5:0],2'b0};
	*/
	/*
	//for the generation of two intersecting lines as the cursor
	assign vga_out_red = (hcount == dummy1 || vcount == dummy2) ? 8'hFF
: {pixel[17:12],2'b0};// GENERATION OF THE CURSOR
```

```
   assign vga_out_green = (hcount == dummy1 || vcount == dummy2) ? 8'hFF :
{pixel[11:6],2'b0};// GENERATION OF THE CURSOR
        assign vga_out_blue = (hcount == dummy1 || vcount == dummy2) ? 8'hFF
: {pixel[5:0],2'b0};// GENERATION OF THE CURSOR
        */
        reg match;
        reg [7:0] factor;
        reg [7:0]       hewhigh = 8'hFF;
        reg [7:0]       saturationlow = 8'hB0;
        reg [7:0]       valuelow = 8'h80;
        reg [7:0]       valuehigh = 8'hFF;
        always @(posedge clk) begin
        factor <= (db_button_down) ? switch[7:0] : factor;
        hewlow <= (db_button0) ? switch[7:0] : hewlow;
        hewhigh <= (db_button1) ? switch[7:0] : hewhigh;
        saturationlow <= (db_button2) ? switch[7:0] : saturationlow;
        valuelow <= (db_button3) ? switch[7:0] : valuelow;
        valuehigh <= (db_button_up) ? switch[7:0] : valuehigh;
        //match <= (HSV[23:16] >= 8'hE0 && HSV[23:16] <= 8'hFF && HSV[15:8]
>= 8'hA4 && HSV[7:0] >= 8'h40 && HSV[7:0] <= 8'hFF);
        match <= (HSV[23:16] >= hewlow) && (HSV[23:16] <= hewhigh) &&
                (HSV[15:8] >= saturationlow) && (HSV[7:0] >= valuelow) &&
                (HSV[7:0] <= valuehigh);
        end
        wire [10:0] delay_hcount;
        wire [9:0] delay_vcount;
        delayV #(.NDELAY(22),.SIZE(11)) delayhcount(.clk(clk), .in(hcount),
.out(delay_hcount));
        //delayV #(.NDELAY(22),.SIZE(10)) delayvcount(.clk(clk), .in(vcount),
.out(delay_vcount));
        //code for the color recognition
        //threshold_filter filter(.clk(clk), .x(hcount), .y(vcount), .hsv(HSV),
.xy1(xy1), /*.xy2, .xy3, .xy4, .xy5, .xy6, */.hewlow(hewlow), .hewhigh(hewhigh),
.saturationlow(saturationlow), .valuelow(valuelow), .valuehigh(valuehigh));
        //threshold_filter filter(.clk(clk), .x(hcount), .y(vcount), .hsv(HSV),
.xy1(xy1),/* xy2, xy3, xy4, xy5, xy6, xy7, xy8,*/ .hewlow(hewlow),
.hewhigh(hewhigh), .saturationlow(saturationlow), .valuelow(valuelow),
.valuehigh(valuehigh));

          threshold_filter filter(.clk(clk),

                .reset(0),

                .x(delay_hcount),

                .y(vcount),

                .factor(factor),
                .match(match),
//
                .x1(xy1[20:10]),
//
                .y1(xy1[9:0]),

                .x2(xy2[20:10]),

                .y2(xy2[9:0]),

                .x3(xy3[20:10]),

                .y3(xy3[9:0]),

                .x4(xy4[20:10]),

                .y4(xy4[9:0]),

                .x5(xy5[20:10]),

                .y5(xy5[9:0]),

                .x6(xy6[20:10]),

                .y6(xy6[9:0]),

                .x7(xy7[20:10]),

                .y7(xy7[9:0]),
//
                .x8(xy8[20:10]),
//                                                              .y8(xy8[9:0]),

        .x_accumulated_op(x_accumulated_op),

        .counted_value_op(counted_value_op),

        .x_com_op(x_com_op));
        always@(posedge clk) begin

                if(!v1 && vsync)  begin
                x2_avg <= xy2[20:10];
                x3_avg <= xy3[20:10];
                x4_avg <= xy4[20:10];
                x5_avg <= xy5[20:10];
                x6_avg <= xy6[20:10];
                x7_avg <= xy7[20:10];
```

```
                    y2_avg <= xy2[9:0];
                    y3_avg <= xy3[9:0];
                    y4_avg <= xy4[9:0];
                    y5_avg <= xy5[9:0];
                    y6_avg <= xy6[9:0];
                    y7_avg <= xy7[9:0];

//                      index1 <= (index1 == 2) ? 0 : (index1 + 1);
////                    x1_avg <= (index1 == 2) ? xy1[20:10] : x1_avg;
//                      x2_avg <= (index1 == 2) ? xy2[20:10] : x2_avg;
//                      x3_avg <= (index1 == 2) ? xy3[20:10] : x3_avg;
//                      x4_avg <= (index1 == 2) ? xy4[20:10] : x4_avg;
//                      x5_avg <= (index1 == 2) ? xy5[20:10] : x5_avg;
//                      x6_avg <= (index1 == 2) ? xy6[20:10] : x6_avg;
//                      x7_avg <= (index1 == 2) ? xy7[20:10] : x7_avg;
////                    y1_avg <= (index1 == 2) ? xy1[9:0] : y1_avg;
//                      y2_avg <= (index1 == 2) ? xy2[9:0] : y2_avg;
//                      y3_avg <= (index1 == 2) ? xy3[9:0] : y3_avg;
//                      y4_avg <= (index1 == 2) ? xy4[9:0] : y4_avg;
//                      y5_avg <= (index1 == 2) ? xy5[9:0] : y5_avg;
//                      y6_avg <= (index1 == 2) ? xy6[9:0] : y6_avg;
//                      y7_avg <= (index1 == 2) ? xy7[9:0] : y7_avg;
//                      //averaging all six x
//                      tot_x1 <= (index1 == 15) ? 0 : (tot_x1 + xy1[20:10]);
//                      x1_avg <= (index1 == 15) ? (tot_x1/16) : x1_avg;
//                      tot_x2 <= (index1 == 15) ? 0 : (tot_x2 + xy2[20:10]);
//                      x2_avg <= (index1 == 15) ? (tot_x2/16) : x2_avg;
//                      tot_x3 <= (index1 == 15) ? 0 : (tot_x3 + xy3[20:10]);
//                      x3_avg <= (index1 == 15) ? (tot_x3/16) : x3_avg;
//                      tot_x4 <= (index1 == 15) ? 0 : (tot_x4 + xy4[20:10]);
//                      x4_avg <= (index1 == 15) ? (tot_x4/16) : x4_avg;
//                      tot_x5 <= (index1 == 15) ? 0 : (tot_x5 + xy5[20:10]);
//                      x5_avg <= (index1 == 15) ? (tot_x5/16) : x5_avg;
//                      tot_x6 <= (index1 == 15) ? 0 : (tot_x6 + xy6[20:10]);
//                      x6_avg <= (index1 == 15) ? (tot_x6/16) : x6_avg;
//                      //averaging all six y
//                      tot_y1 <= (index1 == 15) ? 0 : (tot_y1 + xy1[9:0]);
//                      y1_avg <= (index1 == 15) ? (tot_y1/16) : y1_avg;
//                      tot_y2 <= (index1 == 15) ? 0 : (tot_y2 + xy2[9:0]);
//                      y2_avg <= (index1 == 15) ? (tot_y2/16) : y2_avg;
//                      tot_y3 <= (index1 == 15) ? 0 : (tot_y3 + xy3[9:0]);
//                      y3_avg <= (index1 == 15) ? (tot_y3/16) : y3_avg;
//                      tot_y4 <= (index1 == 15) ? 0 : (tot_y4 + xy4[9:0]);
//                      y4_avg <= (index1 == 15) ? (tot_y4/16) : y4_avg;
//                      tot_y5 <= (index1 == 15) ? 0 : (tot_y5 + xy5[9:0]);
//                      y5_avg <= (index1 == 15) ? (tot_y5/16) : y5_avg;
//                      tot_y6 <= (index1 == 15) ? 0 : (tot_y6 + xy6[9:0]);
//                      y6_avg <= (index1 == 15) ? (tot_y6/16) : y6_avg;
```

```
                end
        end
wire [23:0] pixel11,pixel2,pixel3,pixel4,pixel5,pixel6,pixel7;

////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////
//////////////////////////// RAGA"S MODULES //////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////
wire [23:0]
pixel1raga,pixel2raga,pixel3raga,pixel7raga,pixel8raga,pixelraga,pixelmn,pixelk;
reg [23:0] pixeld1,pixeld2;
  wire phsync,pvsync,pblank;
        wire write_enable;
        wire [1:0] colour_denote;
        wire [10:0] xx1,xx2,xx3,xx4,xx5,xx6;
        wire [9:0]  yy1,yy2,yy3,yy4,yy5,yy6;
        wire [10:0] write_x;
        wire [9:0] write_y;
        wire [20:0] torsoup,torsodown,leftarm,rightarm,leftleg,rightleg;
        wire [10:0] torso_up_x,arm_left_x,arm_right_x,

        leg_left_x,leg_right_x,torso_down_x,x1,x2,x3,x4,x5,x6;
        wire [9:0] torso_up_y,arm_left_y,arm_right_y,

        leg_left_y,leg_right_y,torso_down_y,y1,y2,y3,y4,y5,y6;
        wire [7:0] valuue;
        wire vhsync,vvsync,vblank;
        wire finished_drawing,calc_done,start_decide,dude_start,send_next;
        wire [10:0] final_torso_up_x,final_arm_left_x,final_arm_right_x,

        final_leg_left_x,final_leg_right_x,final_torso_down_x;
        wire [9:0] final_torso_up_y,final_arm_left_y,final_arm_right_y,

        final_leg_left_y,final_leg_right_y,final_torso_down_y;
        reg [10:0] calc_done_delay;
        parameter GAME_RADIUS = 15;




        pulse gen_pulse( .go(left),
                        .clk(clk),
                        .pulse(pulse));


//      constantpoint cnstpt(.clk(clk),
//                                                      .reset(reset),
//                                                      .vsync(vs3),
//                                                          .pixel(pixelmn),
//
          .hcount(delay_hcount),
//                                                          .vcount(vcount),
//                                                          .x1(x2_avg - 298),
//                                                          .y1(y2_avg-2),
//                                                          .x2(x3_avg - 302),
//                                                          .y2(y3_avg+ 4),
//                                                          .x3(x4_avg - 301),
//                                                          .y3(y4_avg-6),
//                                                          .x4(x5_avg - 296),
//                                                          .y4(y5_avg-3),
//                                                          .x5(x6_avg - 303),
//                                                          .y5(y6_avg-1),
//                                                          .x6(x7_avg - 304),
//                                                          .y6(y7_avg-3),
//                                                          .xx1(xx1),
//                                                          .yy1(yy1),
//                                                          .xx2(xx2),
//                                                          .yy2(yy2),
//                                                          .xx3(xx3),
//                                                          .yy3(yy3),
//                                                          .xx4(xx4),
//                                                          .yy4(yy4),
//                                                          .xx5(xx5),
//                                                          .yy5(yy5),
//                                                          .xx6(xx6),
//                                                          .yy6(yy6));

          pupuma pupu(.clk(clk),
                                              .reset(reset),
                                              .x1(x2_avg - 298),
                                                          .y1(y2_avg-2),
                                                          .x2(x3_avg - 302),
                                                          .y2(y3_avg+ 4),
                                                          .x3(x4_avg - 301),
                                                          .y3(y4_avg-6),
                                                          .x4(x5_avg - 296),
                                                          .y4(y5_avg-3),
                                                          .x5(x6_avg - 303),
                                                          .y5(y6_avg-1),
                                                          .x6(x7_avg - 304),
                                                          .y6(y7_avg-3),
                                                          .xx1(xx1),
                                                          .yy1(yy1),
                                                          .xx2(xx2),
                                                          .yy2(yy2),
                                                          .xx3(xx3),
                                                          .yy3(yy3),
                                                          .xx4(xx4),
                                                          .yy4(yy4),
                                                          .xx5(xx5),
                                                          .yy5(yy5),
                                                          .xx6(xx6),
                                                          .yy6(yy6));



          accumulator acc(.xx1(xx1),
                                                          .yy1(yy1),
                                                          .xx2(xx2),
                                                          .yy2(yy2),
                                                          .xx3(xx3),
                                                          .yy3(yy3),
                                                          .xx4(xx4),
                                                          .yy4(yy4),
                                                          .xx5(xx5),
                                                          .yy5(yy5),
                                                          .xx6(xx6),
                                                          .yy6(yy6),
                                      .finished_drawing(finished_drawing),
                                      .game_start(pulse),
                                      .reset(reset),
                                      .clk(clk),
                                      .start_decide(start_decide),
                                      .fin(fin),
                                      .x1(x1),
                                      .x2(x2),
                                      .x3(x3),
                                      .x4(x4),
                                      .x5(x5),
                                      .x6(x6),
                                      .y1(y1),
                                      .y2(y2),
                                      .y3(y3),
                                      .y4(y4),
                                      .y5(y5),
                                      .y6(y6));

          double_buffer dbf(.vclock(clk),
                                                    .reset(reset),
                                                    .hcount(delay_hcount),
                                                    .vcount(vcount),
                                                    .hsync(hs3),
                                                    .vsync(vs3),
                                                    .antialiased(right),
                                                .blank(b3),
                                                .write_x(write_x),
                                                .write_y(write_y),

.colour_denote(colour_denote),
                                                .write_enable(write_enable),
                                                .pixel(pixel1raga),
                                                .vhsync(vhsync),
                                                .vvsync(vvsync),
                                                .vbalnk(vblank));


line_drawer linedraw(.vclock(clk),
                                                     .reset(reset),
                                                     .vsync(vsync),

.start_drawing(calc_done),
                                                     .torsoup(torsoup),

.torsodown(torsodown),
                                                     .leftarm(leftarm),
                                                     .rightarm(rightarm),
                                                     .leftleg(leftleg),
                                                     .rightleg(rightleg),

.finished_drawing(finished_drawing),

.write_enable(write_enable),

.colour_denote(colour_denote),
                                                     .write_x(write_x),
                                                     .write_y(write_y));



points_decider getpoints(.clk(clk),
                                                .x1(x1),
                                                .x2(x2),
                                                .x3(x3),
                                                .x4(x4),
                                                .x5(x5),
                                                .x6(x6),
                                                .y1(y1),
                                                .y2(y2),
                                                .y3(y3),
                                                .y4(y4),
                                                .y5(y5),
                                                            .y6(y6),
                                                            .start_decide(start_decide),
                                                            .torso_up_x(torso_up_x),
                                                            .arm_left_x(arm_left_x),
                                                            .arm_right_x(arm_right_x),
                                                            .leg_left_x(leg_left_x),
                                                            .leg_right_x(leg_right_x),

.torso_down_x(torso_down_x),
                                                            .torso_up_y(torso_up_y),
                                                            .arm_left_y(arm_left_y),
                                                            .arm_right_y(arm_right_y),
                                                            .leg_left_y(leg_left_y),
                                                            .leg_right_y(leg_right_y),

.torso_down_y(torso_down_y),
                                                            .calc_done(calc_done));


        //always @ (posedge clk) calc_done_delay <= (reset? 0 :
{calc_done,calc_done_delay[10:1]});
        virtual_colours vc( .clk(clk),
                                                                    .reset(reset),
                                                                    .go(right),

        .torso_up_x(torso_up_x),

        .arm_left_x(arm_left_x),

        .arm_right_x(arm_right_x),
                                                                    .leg_left_x(leg_left_x),

        .leg_right_x(leg_right_x),

        .torso_down_x(torso_down_x),

        .torso_up_y(torso_up_y),

        .arm_left_y(arm_left_y),

        .arm_right_y(arm_right_y),
                                                                    .leg_left_y(leg_left_y),

        .leg_right_y(leg_right_y),

        .torso_down_y(torso_down_y),

        .final_torso_up_x(final_torso_up_x),
			.final_arm_left_x(final_arm_left_x),

			.final_arm_right_x(final_arm_right_x),

			.final_leg_left_x(final_leg_left_x),

			.final_leg_right_x(final_leg_right_x),

			.final_torso_down_x(final_torso_down_x),

			.final_torso_up_y(final_torso_up_y),

			.final_arm_left_y(final_arm_left_y),

			.final_arm_right_y(final_arm_right_y),

			.final_leg_left_y(final_leg_left_y),

			.final_leg_right_y(final_leg_right_y),

			.final_torso_down_y(final_torso_down_y));

//averager vc( .clk(clk),
//								.reset(reset),
//
			.calc_done(calc_done),
//
			.torso_up_x(torso_up_x),
//
			.arm_left_x(arm_left_x),
//
			.arm_right_x(arm_right_x),
//								.leg_left_x(leg_left_x),
//
			.leg_right_x(leg_right_x),
//
			.torso_down_x(torso_down_x),
//
			.torso_up_y(torso_up_y),
//
			.arm_left_y(arm_left_y),
//
			.arm_right_y(arm_right_y),
//								.leg_left_y(leg_left_y),
//
			.leg_right_y(leg_right_y),
//
            .torso_down_y(torso_down_y),
//
            .final_torso_up_x(final_torso_up_x),
//
            .final_arm_left_x(final_arm_left_x),
//
            .final_arm_right_x(final_arm_right_x),
//
            .final_leg_left_x(final_leg_left_x),
//
            .final_leg_right_x(final_leg_right_x),
//
            .final_torso_down_x(final_torso_down_x),
//
            .final_torso_up_y(final_torso_up_y),
//
            .final_arm_left_y(final_arm_left_y),
//
            .final_arm_right_y(final_arm_right_y),
//
            .final_leg_left_y(final_leg_left_y),
//
            .final_leg_right_y(final_leg_right_y),
//
            .final_torso_down_y(final_torso_down_y),
//
            .start_drawing(dude_start),
//
            .send_next(send_next));


            face_generator          face_gen(.x(final_torso_up_x-32),
                                                        .hcount(delay_hcount),
                                                        .y(final_torso_up_y - 64),
                                                        .vcount(vcount),
                                                        .clk(clk),
                                                        .reset(reset),
                                                        .pixel(pixel2raga));

            defparam face_gen.RADIUS = 32;

            gamelogic logicunit(.lefthandx(final_arm_left_x - GAME_RADIUS),

            .hcount(delay_hcount),

            .righthandx(final_arm_right_x - GAME_RADIUS),
            .leftlegx(final_leg_left_x - GAME_RADIUS),

            .rightlegx(final_leg_right_x - GAME_RADIUS),

            .torsoupx(final_torso_up_x - GAME_RADIUS),

            .torsodownx(final_torso_down_x - GAME_RADIUS),

            .lefthandy(final_arm_left_y - GAME_RADIUS),

            .righthandy(final_arm_right_y - GAME_RADIUS),

            .leftlegy(final_leg_left_y - GAME_RADIUS),

            .rightlegy(final_leg_right_y - GAME_RADIUS),

            .torsoupy(final_torso_up_y - GAME_RADIUS),

            .torsodowny(final_torso_down_y- GAME_RADIUS),
                                                            .vcount(vcount),
                                                            .vclockn(clk),
                                                            .reset(reset),
                                                            .pixel(pixel3raga));


            ///////////////////////////////////////////////////////////////////////////////
            //////////////////////////////FONT DISPLAY/////////////////////////////////////
            ///////////////////////////////////////////////////////////////////////////////
            //files included are char_string_display and font rom.v

            wire [2:0] pixel222,pixel333;
            wire [175:0] cstring = "ACTION TRACKING SYSTEM";
            wire [159:0] cstring2 = "By Raga And Shubhang";
            reg [10:0]  cx2=11'd325;
            reg [9:0]   cy2=10'd550;
            reg [10:0]  cx3=11'd350;
            reg [9:0]   cy3=10'd600;

            char_string_display ch2(.vclock(clock_65mhz),

            .hcount(hcount),

            .vcount(vcount),

            .pixel(pixel222),

            .cstring(cstring),
                                                            .cx(cx2),
                                                            .cy(cy2));
            defparam ch2.NCHAR = 22;
        defparam ch2.NCHAR_BITS = 5;

            char_string_display ch3(.vclock(clock_65mhz),

            .hcount(hcount),

            .vcount(vcount),

            .pixel(pixel333),

            .cstring(cstring2),
                                                            .cx(cx3),
                                                            .cy(cy3));
            defparam ch3.NCHAR = 20;
        defparam ch3.NCHAR_BITS = 5;


            assign pixel7raga[7:0] = (pixel222[0] == 1)? 8'b1111_1111:8'b0000_0000;
            assign pixel7raga[15:8] = (pixel222[1] == 1)?
    8'b1111_1111:8'b0000_0000;
            assign pixel7raga[23:16] = (pixel222[2] == 1)?
    8'b1111_1111:8'b0000_0000;
            assign pixel8raga[7:0] = (pixel333[0] == 1)? 8'b1111_1111:8'b0000_0000;
            assign pixel8raga[15:8] = (pixel333[1] == 1)?
    8'b1111_1111:8'b0000_0000;
            assign pixel8raga[23:16] = (pixel333[2] == 1)?
    8'b1111_1111:8'b0000_0000;

            ////////////////////////////////////////////////////////////////////////////////
            ////////////////////////////////////////////////////////////////////////////////
            ////////////////////////////////////////////////////////////////////////////////

            assign valuue = switch[7:0];
            assign torsoup = {final_torso_up_x,final_torso_up_y};
            assign torsodown = {final_torso_down_x,final_torso_down_y};
            assign leftarm = {final_arm_left_x,final_arm_left_y};
            assign rightarm = {final_arm_right_x,final_arm_right_y};
            assign leftleg = {final_leg_left_x,final_leg_left_y};
            assign rightleg = {final_leg_right_x,final_leg_right_y};
            assign pixelraga = pixeld1 | pixeld2;
            always @(posedge clk) begin
            pixeld1 <= pixel1raga| pixel2raga;
            pixeld2 <= pixel3raga | pixel7raga| pixel8raga;
            end
    ////////////////////////////////////////////////////////////////////////////////////
```

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


face_generator face2(.x(x2_avg),
                                        .hcount(delay_hcount),
                                        .y(y2_avg),
                                        .vcount(vcount),
                                        .clk(clk),
                                        .reset(reset),
                                        .pixel(pixel2));

//defparam face2.COLOR = 24'b1111_1111_1111_1111_1111_1111;

face_generator face3(.x(x3_avg),
                                        .hcount(delay_hcount),
                                        .y(y3_avg),
                                        .vcount(vcount),
                                        .clk(clk),
                                        .reset(reset),
                                        .pixel(pixel3));

//defparam face3.COLOR = 24'b0000_0000_0000_0000_1111_1111;
face_generator face4(.x(x4_avg),
                                        .hcount(delay_hcount),
                                        .y(y4_avg),
                                        .vcount(vcount),
                                        .clk(clk),
                                        .reset(reset),
                                        .pixel(pixel4));

//defparam face4.COLOR = 24'b0000_0000_1111_1111_0000_0000;

face_generator face5(.x(x5_avg),
                                        .hcount(delay_hcount),
                                        .y(y5_avg),
                                        .vcount(vcount),
                                        .clk(clk),
                                        .reset(reset),
                                        .pixel(pixel5));

//defparam face5.COLOR = 24'b1111_1111_1111_1111_0000_0000;

face_generator face6(.x(x6_avg),
                                        .hcount(delay_hcount),
                                        .y(y6_avg),
                                        .vcount(vcount),
                                        .clk(clk),
                                                            .reset(reset),
                                                            .pixel(pixel6));

//defparam face6.COLOR = 24'b1111_1111_0000_0000_1111_1111;

face_generator face7(.x(x7_avg),
                                                            .hcount(delay_hcount),
                                                            .y(y7_avg),
                                                            .vcount(vcount),
                                                            .clk(clk),
                                                            .reset(reset),
                                                            .pixel(pixel7));

//defparam face7.COLOR = 24'b0000_0000_1111_1111_1111_1111;
facepipe facep(.clk(clk),
                    .reset(reset),
                    .pixel2(pixel2),
                    .pixel3(pixel3),
                    .pixel4(pixel4),
                    .pixel5(pixel5),
                    .pixel6(pixel6),
                    .pixel7(pixel7),
                    .pixelk(pixelk));




      /*assign vga_out_red = ( (pixel7[1] || pixel2[1]|| pixel3[1]|| pixel4[1]||
pixel5[1]|| pixel6[1])? (pixel7[23:16] | pixel2[23:16]
                                                                        | pixel3[23:16]
| pixel4[23:16]
                                                                        | pixel5[23:16]
|pixel6[23:16]) : ((hcount == dummy1 || vcount == dummy2) ?  8'hFF : (match ?
8'hFF : {pixel1[17:12],2'b0})));
      assign vga_out_green = ((pixel7[1] || pixel2[1]|| pixel3[1]|| pixel4[1]||
pixel5[1]|| pixel6[1])? (pixel7[15:8] | pixel2[15:8]
                                                                        | pixel3[15:8] |
pixel4[15:8]
                                                                        | pixel5[15:8]
|pixel6[15:8]) : ((hcount == dummy1 || vcount == dummy2) ?  8'hFF : (match ?
8'h00 : {pixel1[11:6],2'b0})));
      assign vga_out_blue =  ((pixel7[1] || pixel2[1]|| pixel3[1]|| pixel4[1]||
pixel5[1]|| pixel6[1])?(pixel7[7:0] | pixel2[7:0]
                                                                        | pixel3[7:0] |
pixel4[7:0]
```

```
                                                        | pixel5[7:0]
|pixel6[7:0]) : ((hcount == dummy1 || vcount == dummy2) ?  8'hFF : (match ?
8'h00 : {pixel1[5:0],2'b0})));
*/ //best working


        assign vga_out_red = (switch[3] ? pixelraga[23:16] : (pixelk[1] ?
pixelk[23:16] : (match ? 8'hFF : {pixel1[17:12],2'b0})));
        assign vga_out_green = (switch[3] ? pixelraga[15:8] :(pixelk[1] ?
pixelk[15:8] : (match ? 8'h00 : {pixel1[11:6],2'b0})));
        assign vga_out_blue = (switch[3] ? pixelraga[7:0] :(pixelk[1] ? pixelk[7:0] :
(match ? 8'h00 : {pixel1[5:0],2'b0})));






        /*assign vga_out_red = (hcount == dummy1 || vcount == dummy2) ?
8'hFF : (match ? 8'hFF : {pixel[17:12],2'b0});
        assign vga_out_green = (hcount == dummy1 || vcount == dummy2) ?
8'hFF : (match ? 8'h00 : {pixel[11:6],2'b0});
        assign vga_out_blue = (hcount == dummy1 || vcount == dummy2) ?
8'hFF : (match ? 8'h00 : {pixel[5:0],2'b0});*/

        assign vga_out_sync_b = 1'b1;    //not used
  assign vga_out_pixel_clock = ~clk;

        /*assign vga_out_blank_b = ~b;
  assign vga_out_hsync = hs;
  assign vga_out_vsync = vs;*/
        //delayed hsync vsync and blank

        assign vga_out_blank_b = ~b3;
  assign vga_out_hsync = hs3;
  assign vga_out_vsync = vs3;

  // debugging

  assign led = ~{vram_addr[18:13],reset,switch[0]};

  always @(posedge clk) begin

          // dispdata <= {vram_read_data,9'b0,vram_addr};

          dispdata <= {ntsc_data,9'b0,ntsc_addr};

        end
```

```
endmodule
```

A.2 XVGA Module

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

A.3 VRAM Display Module

```
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;
        output [17:0] vr_pixel;
  output [18:0] vram_addr;
  input [35:0]  vram_read_data;

  //forecast hcount & vcount 8 clock cycles ahead to get data from ZBT
  wire [10:0] hcount_f = (hcount >= 1048) ? (hcount - 1048) : (hcount + 8);
  wire [9:0] vcount_f = (hcount >= 1048) ? ((vcount == 805) ? 0 : vcount + 1) :
vcount;

  //wire [18:0]   vram_addr = {1'b0, vcount_f, hcount_f[9:2]};

        wire [18:0]       vram_addr = {vcount_f, hcount_f[9:1]};

  //wire [1:0]     hc4 = hcount[1:0];
        wire hc4 = hcount[0];
  //reg [7:0]      vr_pixel;
        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;
          last_vr_data <= (hc4==1'd1) ? vr_data_latched : last_vr_data;

  always @(posedge clk)
    //vr_data_latched <= (hc4==2'd1) ? vram_read_data : vr_data_latched;
          vr_data_latched <= (hc4==1'd0) ? 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];*/
                  1'b1 : vr_pixel = last_vr_data[17:0];
                  1'b0 :  vr_pixel = last_vr_data[35:18];
    endcase

endmodule // vram_display
```

A.4 RAM clock Module

```
///////////////////////////////////////////////////////////////////////////
// ramclock module

///////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- ZBT RAM clock generation
//
//
// Created: April 27, 2004
// Author: Nathan Ickes
//
///////////////////////////////////////////////////////////////////////////////
//
// This module generates deskewed clocks for driving the ZBT SRAMs and FPGA
// registers. A special feedback trace on the labkit PCB (which is length
// matched to the RAM traces) is used to adjust the RAM clock phase so that
// rising clock edges reach the RAMs at exactly the same time as rising clock
// edges reach the registers in the FPGA.
//
// The RAM clock signals are driven by DDR output buffers, which further
// ensures that the clock-to-pad delay is the same for the RAM clocks as it is
// for any other registered RAM signal.
//
// When the FPGA is configured, the DCMs are enabled before the chip-level I/O
// drivers are released from tristate. It is therefore necessary to
// artificially hold the DCMs in reset for a few cycles after configuration.
// This is done using a 16-bit shift register. When the DCMs have locked, the
// <lock> output of this mnodule will go high. Until the DCMs are locked, the
// ouput clock timings are not guaranteed, so any logic driven by the
// <fpga_clock> should probably be held inreset until <locked> is high.
//
///////////////////////////////////////////////////////////////////////////////

module ramclock(ref_clock, fpga_clock, ram0_clock, ram1_clock,
                clock_feedback_in, clock_feedback_out, locked);

  input ref_clock;                // Reference clock input
```

```
output fpga_clock;              // Output clock to drive FPGA logic
output ram0_clock, ram1_clock;   // Output clocks for each RAM chip
input  clock_feedback_in;        // Output to feedback trace
output clock_feedback_out;       // Input from feedback trace
output locked;                  // Indicates that clock outputs are stable

wire  ref_clk, fpga_clk, ram_clk, fb_clk, lock1, lock2, dcm_reset;

////////////////////////////////////////////////////////////////////////////

//To force ISE to compile the ramclock, this line has to be removed.
//IBUFG ref_buf (.O(ref_clk), .I(ref_clock));

        assign ref_clk = ref_clock;

BUFG int_buf (.O(fpga_clock), .I(fpga_clk));

DCM int_dcm (.CLKFB(fpga_clock),
                  .CLKIN(ref_clk),
                  .RST(dcm_reset),
                  .CLK0(fpga_clk),
                  .LOCKED(lock1));
// synthesis attribute DLL_FREQUENCY_MODE of int_dcm is "LOW"
// synthesis attribute DUTY_CYCLE_CORRECTION of int_dcm is "TRUE"
// synthesis attribute STARTUP_WAIT of int_dcm is "FALSE"
// synthesis attribute DFS_FREQUENCY_MODE of int_dcm is "LOW"
// synthesis attribute CLK_FEEDBACK of int_dcm  is "1X"
// synthesis attribute CLKOUT_PHASE_SHIFT of int_dcm is "NONE"
// synthesis attribute PHASE_SHIFT of int_dcm is 0

BUFG ext_buf (.O(ram_clock), .I(ram_clk));

IBUFG fb_buf (.O(fb_clk), .I(clock_feedback_in));

DCM ext_dcm (.CLKFB(fb_clk),
                     .CLKIN(ref_clk),
                     .RST(dcm_reset),
                     .CLK0(ram_clk),
                     .LOCKED(lock2));
// synthesis attribute DLL_FREQUENCY_MODE of ext_dcm is "LOW"
// synthesis attribute DUTY_CYCLE_CORRECTION of ext_dcm is "TRUE"
// synthesis attribute STARTUP_WAIT of ext_dcm is "FALSE"
// synthesis attribute DFS_FREQUENCY_MODE of ext_dcm is "LOW"
// synthesis attribute CLK_FEEDBACK of ext_dcm  is "1X"
// synthesis attribute CLKOUT_PHASE_SHIFT of ext_dcm is "NONE"
// synthesis attribute PHASE_SHIFT of ext_dcm is 0

SRL16 dcm_rst_sr (.D(1'b0), .CLK(ref_clk), .Q(dcm_reset),
                    .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
  // synthesis attribute init of dcm_rst_sr is "000F";


  OFDDRRSE ddr_reg0 (.Q(ram0_clock), .C0(ram_clock), .C1(~ram_clock),
                    .CE (1'b1), .D0(1'b1), .D1(1'b0), .R(1'b0), .S(1'b0));
  OFDDRRSE ddr_reg1 (.Q(ram1_clock), .C0(ram_clock), .C1(~ram_clock),
                    .CE (1'b1), .D0(1'b1), .D1(1'b0), .R(1'b0), .S(1'b0));
  OFDDRRSE ddr_reg2 (.Q(clock_feedback_out), .C0(ram_clock),
.C1(~ram_clock),
                    .CE (1'b1), .D0(1'b1), .D1(1'b0), .R(1'b0), .S(1'b0));

  assign locked = lock1 && lock2;

endmodule
```

A.5 ZBT Driver Module

```
//
// File:   zbt_6111.v
// Date:   27-Nov-05
// Author: I. Chuang <ichuang@mit.edu>
//
// Simple ZBT driver for the MIT 6.111 labkit, which does not hide the
// pipeline delays of the ZBT from the user.  The ZBT memories have
// two cycle latencies on read and write, and also need extra-long data hold
// times around the clock positive edge to work reliably.
//

/////////////////////////////////////////////////////////////////////////////
// Ike's simple ZBT RAM driver for the MIT 6.111 labkit
//
// Data for writes can be presented and clocked in immediately; the actual
// writing to RAM will happen two cycles later.
//
// Read requests are processed immediately, but the read data is not available
// until two cycles after the intial request.
//
// A clock enable signal is provided; it enables the RAM clock when high.

module zbt_6111(clk, cen, we, addr, write_data, read_data,
                  ram_clk, ram_we_b, ram_address, ram_data, ram_cen_b);

  input clk;                    // system clock
  input cen;                    // clock enable for gating ZBT cycles
  input we;                     // write enable (active HIGH)
  input [18:0] addr;            // memory address
  input [35:0] write_data;      // data to write
  output [35:0] read_data;      // data read from memory
  output           ram_clk;         // physical line to ram clock
  output           ram_we_b;      // physical line to ram we_b
  output [18:0] ram_address; // physical line to ram address
  inout [35:0]  ram_data;       // physical line to ram data
  output           ram_cen_b;    // physical line to ram clock enable

  // clock enable (should be synchronous and one cycle high at a time)
  wire   ram_cen_b = ~cen;

  // 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]   we_delay;

  always @(posedge clk)
    we_delay <= cen ? {we_delay[0],we} : we_delay;

  // create two-stage pipeline for write data

  reg [35:0]  write_data_old1;
  reg [35:0]  write_data_old2;
  always @(posedge clk)
    if (cen)
      {write_data_old2, write_data_old1} <= {write_data_old1, write_data};

  // wire to ZBT RAM signals

  assign      ram_we_b = ~we;
  assign      ram_clk = 1'b0; // dummy output
//   assign      ram_clk = ~clk;     // RAM is not happy with our data hold
                             // times if its clk edges equal FPGA's
                             // so we clock it on the falling edges
                             // and thus let data stabilize longer
  assign      ram_address = addr;

  assign      ram_data = we_delay[1] ? write_data_old2 : {36{1'bZ}};
  assign      read_data = ram_data;

endmodule // zbt_6111
```

A.6 YCRCB to RGB Module

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
```

```
// Company:
// Engineer:
//
// Create Date:    21:05:44 11/08/2011
// Design Name:
// Module Name:    YCrCb2RGB
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////
/***************************************************************************
 **
 ** Module: ycrcb2rgb
 **
 ** Generic Equations:
 ***************************************************************************/

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



/*always @ (posedge clk or posedge rst)
   if (rst)
      begin
       R_int <= 0; G_int <= 0; B_int <= 0;
      end
   else
     begin
     X_int <= (const1 * (Y_reg - 'd64)) ;
     R_int <= X_int + (const2 * (Cr_reg - 'd512));
     G_int <= X_int - (const3 * (Cr_reg - 'd512)) - (const4 * (Cb_reg - 'd512));
     B_int <= X_int + (const5 * (Cb_reg - 'd512));
     end

*/
/* limit output to 0 - 4095, <0 equals o and >4095 equals 4095 */
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
```

A.7 RGB to HSV Module

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    18:45:01 11/10/2010
// Design Name:
// Module Name:    rgb2hsv
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////
module rgb2hsv(clock, reset, r, g, b, h, s, v);
                input wire clock;
                input wire reset;
                input wire [7:0] r;
                input wire [7:0] g;
                input wire [7:0] b;
                output reg [7:0] h;
                output reg [7:0] s;
                output reg [7:0] v;
                reg [7:0] my_r_delay1, my_g_delay1, my_b_delay1;
                reg [7:0] my_r_delay2, my_g_delay2, my_b_delay2;
                reg [7:0] my_r, my_g, my_b;
                reg [7:0] min, max, delta;
                reg [15:0] s_top;
                reg [15:0] s_bottom;
                reg [15:0] h_top;
                reg [15:0] h_bottom;
                wire [15:0] s_quotient;
                wire [15:0] s_remainder;
                wire s_rfd;
                wire [15:0] h_quotient;
                wire [15:0] h_remainder;
                wire h_rfd;
                reg [7:0] v_delay [19:0];
                reg [18:0] h_negative;
                reg [15:0] h_add [18:0];
                reg [4:0] i;
                // Clocks 4-18: perform all the divisions
                //the s_divider (16/16) has delay 18
                //the hue_div (16/16) has delay 18

                divider hue_div1(
                .clk(clock),
                .dividend(s_top),
                .divisor(s_bottom),
                .quotient(s_quotient),
                .fractional(s_remainder),
                .rfd(s_rfd)
                );
                divider hue_div2(
                .clk(clock),
                .dividend(h_top),
                .divisor(h_bottom),
                .quotient(h_quotient),
                .fractional(h_remainder),
                .rfd(h_rfd)
                );
                always @ (posedge clock) begin

                        // Clock 1: latch the inputs (always positive)
                        {my_r, my_g, my_b} <= {r, g, b};

                        // Clock 2: compute min, max
                        {my_r_delay1, my_g_delay1, my_b_delay1} <= {my_r,
my_g, my_b};

                        if((my_r >= my_g) && (my_r >= my_b)) //(B,S,S)
                                max <= my_r;
                        else if((my_g >= my_r) && (my_g >= my_b)) //(S,B,S)
                                max <= my_g;
                        else    max <= my_b;
```

```
                    if((my_r <= my_g) && (my_r <= my_b)) //(S,B,B)
                            min <= my_r;
                    else if((my_g <= my_r) && (my_g <= my_b)) //(B,S,B)
                            min <= my_g;
                    else
                            min <= my_b;

                    // Clock 3: compute the delta
                    {my_r_delay2, my_g_delay2, my_b_delay2} <=
{my_r_delay1, my_g_delay1, my_b_delay1};
                    v_delay[0] <= max;
                    delta <= max - min;

                    // Clock 4: compute the top and bottom of whatever
divisions we need to do
                    s_top <= 8'd255 * delta;
                    s_bottom <= (v_delay[0]>0)?{8'd0, v_delay[0]}: 16'd1;


                    if(my_r_delay2 == v_delay[0]) begin
                            h_top <= (my_g_delay2 >=
my_b_delay2)?(my_g_delay2 - my_b_delay2) * 8'd255:(my_b_delay2 -
my_g_delay2) * 8'd255;
                            h_negative[0] <= (my_g_delay2 >=
my_b_delay2)?0:1;
                            h_add[0] <= 16'd0;
                    end
                    else if(my_g_delay2 == v_delay[0]) begin
                            h_top <= (my_b_delay2 >=
my_r_delay2)?(my_b_delay2 - my_r_delay2) * 8'd255:(my_r_delay2 -
my_b_delay2) * 8'd255;
                            h_negative[0] <= (my_b_delay2 >=
my_r_delay2)?0:1;
                            h_add[0] <= 16'd85;
                    end
                    else if(my_b_delay2 == v_delay[0]) begin
                            h_top <= (my_r_delay2 >=
my_g_delay2)?(my_r_delay2 - my_g_delay2) * 8'd255:(my_g_delay2 -
my_r_delay2) * 8'd255;
                            h_negative[0] <= (my_r_delay2 >=
my_g_delay2)?0:1;
                            h_add[0] <= 16'd170;
                    end

                    h_bottom <= (delta > 0)?delta * 8'd6:16'd6;
```

```
				//delay the v and h_negative signals 18 times
				for(i=1; i<19; i=i+1) begin
					v_delay[i] <= v_delay[i-1];
					h_negative[i] <= h_negative[i-1];
					h_add[i] <= h_add[i-1];
				end

				v_delay[19] <= v_delay[18];
				//Clock 22: compute the final value of h
				//depending on the value of h_delay[18], we need to
subtract 255 from it to make it come back around the circle
				if(h_negative[18] && (h_quotient > h_add[18])) begin
					h <= 8'd255 - h_quotient[7:0] + h_add[18];
				end
				else if(h_negative[18]) begin
					h <= h_add[18] - h_quotient[7:0];
				end
				else begin
					h <= h_quotient[7:0] + h_add[18];
				end

				//pass out s and v straight
				s <= s_quotient;
				v <= v_delay[19];
			end
endmodule
```

A.8 NTSC to ZBT Module

```
//
// File:   ntsc2zbt.v
// Date:   27-Nov-05
// Author: I. Chuang <ichuang@mit.edu>
//
// Example for MIT 6.111 labkit showing how to prepare NTSC data
// (from Javier's decoder) to be loaded into the ZBT RAM for video
// display.
//
// The ZBT memory is 36 bits wide; we only use 32 bits of this, to
// store 4 bytes of black-and-white intensity data from the NTSC
// video input.
//
// Bug fix: Jonathan P. Mailoa <jpmailoa@mit.edu>
// Date   : 11-May-09  // gph mod 11/3/2011
//
//
```

```
// Bug due to memory management will be fixed. It happens because
// the memory addressing protocol is off between ntsc2zbt.v and
// vram_display.v. There are 2 solutions:
// -. Fix the memory addressing in this module (neat addressing protocol)
//    and do memory forecast in vram_display module.
// -. Do nothing in this module and do memory forecast in vram_display
//    module (different forecast count) while cutting off reading from
//    address(0,0,0).
//
// Bug in this module causes 4 pixel on the rightmost side of the camera
// to be stored in the address that belongs to the leftmost side of the
// screen.
//
// In this example, the second method is used. NOTICE will be provided
// on the crucial source of the bug.
//
/////////////////////////////////////////////////////////////////////////////
// Prepare data and address values to fill ZBT memory with NTSC data

module ntsc_to_zbt(clk, vclk, fvh, dv, din, ntsc_addr, ntsc_data, ntsc_we, sw);

   input          clk;    // system clock
   input          vclk;   // video clock from camera
   input [2:0]    fvh;
   input          dv;
   //input [7:0]  din;
         input [17:0]     din;
   output [18:0] ntsc_addr;
   output [35:0] ntsc_data;
   output         ntsc_we;        // write enable for NTSC data
   input          sw;             // switch which determines mode (for debugging)

   parameter      COL_START = 10'd300;
   parameter      ROW_START = 10'd0;

   // here put the luminance data from the ntsc decoder into the ram
   // this is for 800 * 600 XGA display

   reg [9:0]      col = 0;
   reg [9:0]      row = 0;
   //reg [7:0]    vdata = 0;
         reg [17:0]       vdata = 0;
   reg            vwe;
   reg            old_dv;
   reg            old_frame;      // frames are even / odd interlaced
   reg            even_odd;       // decode interlaced frame to this wire

   wire   frame = fvh[2];
   wire   frame_edge = frame & ~old_frame;

   always @ (posedge vclk) //LLC1 is reference
     begin
         old_dv <= dv;
         vwe <= dv && !fvh[2] & ~old_dv; // if data valid, write it
         old_frame <= frame;
         even_odd = frame_edge ? ~even_odd : even_odd;

         if (!fvh[2])
           begin
              col <= fvh[0] ? COL_START :
                         (!fvh[2] && !fvh[1] && dv && (col < 1024)) ? col + 1 : col;
              row <= fvh[1] ? ROW_START :
                         (!fvh[2] && fvh[0] && (row < 768)) ? row + 1 : row;
              vdata <= (dv && !fvh[2]) ? din : vdata;
           end
     end

   // synchronize with system clock

   reg [9:0] x[1:0],y[1:0];
   //reg [7:0] data[1:0];
         reg [17:0] data[1:0];
   reg       we[1:0];
   reg        eo[1:0];

   always @(posedge clk)
     begin
         {x[1],x[0]} <= {x[0],col};
         {y[1],y[0]} <= {y[0],row};
         {data[1],data[0]} <= {data[0],vdata};
         {we[1],we[0]} <= {we[0],vwe};
         {eo[1],eo[0]} <= {eo[0],even_odd};
     end

   // edge detection on write enable signal

   reg old_we;
   wire we_edge = we[1] & ~old_we;
   always @(posedge clk) old_we <= we[1];

   // shift each set of four bytes into a large register for the ZBT

   //reg [31:0] mydata;
         reg [35:0] mydata;
   always @(posedge clk)
     if (we_edge)
```

```
    //mydata <= { mydata[23:0], data[1] };
                  mydata <= { mydata[17:0], data[1] };

// NOTICE : Here we have put 4 pixel delay on mydata. For example, when:
// (x[1], y[1]) = (60, 80) and eo[1] = 0, then:
// mydata[31:0] = ( pixel(56,160), pixel(57,160), pixel(58,160), pixel(59,160) )
// This is the root of the original addressing bug.


// NOTICE : Notice that we have decided to store mydata, which
//         contains pixel(56,160) to pixel(59,160) in address
//         (0, 160 (10 bits), 60 >> 2 = 15 (8 bits)).
//
//         This protocol is dangerous, because it means
//         pixel(0,0) to pixel(3,0) is NOT stored in address
//         (0, 0 (10 bits), 0 (8 bits)) but is rather stored
//         in address (0, 0 (10 bits), 4 >> 2 = 1 (8 bits)). This
//         calculation ignores COL_START & ROW_START.
//
//         4 pixels from the right side of the camera input will
//         be stored in address corresponding to x = 0.
//
//         To fix, delay col & row by 4 clock cycles.
//         Delay other signals as well.

reg [39:0] x_delay;
reg [39:0] y_delay;
reg [3:0] we_delay;
reg [3:0] eo_delay;

always @ (posedge clk)
begin
  x_delay <= {x_delay[29:0], x[1]};
  y_delay <= {y_delay[29:0], y[1]};
  we_delay <= {we_delay[2:0], we[1]};
  eo_delay <= {eo_delay[2:0], eo[1]};
end

// compute address to store data in

//wire [18:0] myaddr = {1'b0, y_delay[38:30], eo_delay[3], x_delay[39:32]};
        wire [18:0] myaddr = {y_delay[38:30], eo_delay[3], x_delay[39:31]};

// Now address (0,0,0) contains pixel data(0,0) etc.


// alternate (256x192) image data and address
//wire [31:0] mydata2 = {data[1],data[1],data[1],data[1]};
```

```
        wire [35:0] mydata2 = {data[1],data[1]};
   //wire [18:0] myaddr2 = {1'b0, y_delay[38:30], eo_delay[3], x_delay[37:30]};
        wire [18:0] myaddr2 = {y_delay[38:30], eo_delay[3], x_delay[37:29]};

   // update the output address and data only when four bytes ready

   reg [18:0] ntsc_addr;
   reg [35:0] ntsc_data;
   //wire       ntsc_we = sw ? we_edge : (we_edge & (x_delay[31:30]==2'b00));
        wire       ntsc_we = sw ? we_edge : (we_edge & (x_delay[30]==1'b0));

   always @(posedge clk)
     if ( ntsc_we )
       begin
          ntsc_addr <= sw ? myaddr2 : myaddr;        // normal and expanded
modes
          //ntsc_data <= sw ? {4'b0,mydata2} : {4'b0,mydata};
          ntsc_data <= sw ? {mydata2} : {mydata};
       end

endmodule // ntsc_to_zbt
```

A.9 Hex Display Driver Module

```
///////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Hex display driver
//
//
// File:   display_16hex.v
// Date:   24-Sep-05
//
// Created: April 27, 2004
// Author: Nathan Ickes
//
// This module drives the labkit hex displays and shows the value of
// 8 bytes (16 hex digits) on the displays.
//
// 24-Sep-05 Ike: updated to use new reset-once state machine, remove clear
// 02-Nov-05 Ike: updated to make it completely synchronous
//
// Inputs:
//
//   reset       - active high
//   clock_27mhz - the synchronous clock
//   data        - 64 bits; each 4 bits gives a hex digit
```

```
//
// Outputs:
//
//    disp_*     - display lines used in the 6.111 labkit (rev 003 & 004)
//
///////////////////////////////////////////////////////////////////////////////

module display_16hex (reset, clock_27mhz, data_in,
                      disp_blank, disp_clock, disp_rs, disp_ce_b,
                      disp_reset_b, disp_data_out);

   input reset, clock_27mhz;    // clock and reset (active high reset)
   input [63:0] data_in;            // 16 hex nibbles to display

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

   reg disp_data_out, disp_rs, disp_ce_b, disp_reset_b;

   ///////////////////////////////////////////////////////////////////////////////
   //
   // Display Clock
   //
   // Generate a 500kHz clock for driving the displays.
   //
   ///////////////////////////////////////////////////////////////////////////////

   reg [5:0] count;
   reg [7:0] reset_count;
//   reg       old_clock;
   wire      dreset;
   wire      clock = (count<27) ? 0 : 1;

   always @(posedge clock_27mhz)
     begin
        count <= reset ? 0 : (count==53 ? 0 : count+1);
        reset_count <= reset ? 100 : ((reset_count==0) ? 0 : reset_count-1);
//      old_clock <= clock;
     end

   assign dreset = (reset_count != 0);
   assign disp_clock = ~clock;
   wire   clock_tick = ((count==27) ? 1 : 0);
//   wire   clock_tick = clock & ~old_clock;

   ///////////////////////////////////////////////////////////////////////////////
   //
   // Display State Machine
```

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

reg [7:0] state;               // FSM state
reg [9:0] dot_index;           // index to current dot being clocked out
reg [31:0] control;            // control register
reg [3:0] char_index;          // index of current character
reg [39:0] dots;               // dots for a single digit
reg [3:0] nibble;              // hex nibble of current character
reg [63:0] data;

assign disp_blank = 1'b0; // low <= not blanked

always @(posedge clock_27mhz)
  if (clock_tick)
    begin
        if (dreset)
          begin
            state <= 0;
            dot_index <= 0;
            control <= 32'h7F7F7F7F;
          end
        else
          casex (state)
            8'h00:
                begin
                  // Reset displays
                  disp_data_out <= 1'b0;
                  disp_rs <= 1'b0; // dot register
                  disp_ce_b <= 1'b1;
                  disp_reset_b <= 1'b0;
                  dot_index <= 0;
                  state <= state+1;
                end

            8'h01:
                begin
                  // End reset
                  disp_reset_b <= 1'b1;
                  state <= state+1;
                end

            8'h02:
                begin
                  // Initialize dot register (set all dots to zero)
                  disp_ce_b <= 1'b0;
                  disp_data_out <= 1'b0; // dot_index[0];
                  if (dot_index == 639)
        state <= state+1;
      else
        dot_index <= dot_index+1;
    end

8'h03:
    begin
      // Latch dot data
      disp_ce_b <= 1'b1;
      dot_index <= 31;                // re-purpose to init ctrl reg
      state <= state+1;
    end

8'h04:
    begin
      // Setup the control register
      disp_rs <= 1'b1; // Select the control register
      disp_ce_b <= 1'b0;
      disp_data_out <= control[31];
      control <= {control[30:0], 1'b0};       // shift left
      if (dot_index == 0)
        state <= state+1;
      else
        dot_index <= dot_index-1;
    end

8'h05:
    begin
      // Latch the control register data / dot data
      disp_ce_b <= 1'b1;
      dot_index <= 39;                // init for single char
      char_index <= 15;               // start with MS char
      data <= data_in;
      state <= state+1;
    end

8'h06:
    begin
      // Load the user's dot data into the dot reg, char by char
      disp_rs <= 1'b0;                        // Select the dot register
      disp_ce_b <= 1'b0;
      disp_data_out <= dots[dot_index]; // dot data from msb
      if (dot_index == 0)
     if (char_index == 0)
      state <= 5;                     // all done, latch data
        else
          begin
              char_index <= char_index - 1;        // goto next char
```

```
                        data <= data_in;
                        dot_index <= 39;
                    end
                 else
                   dot_index <= dot_index-1; // else loop thru all dots
               end

           endcase // casex(state)
    end

always @ (data or char_index)
  case (char_index)
    4'h0:              nibble <= data[3:0];
    4'h1:              nibble <= data[7:4];
    4'h2:              nibble <= data[11:8];
    4'h3:              nibble <= data[15:12];
    4'h4:              nibble <= data[19:16];
    4'h5:              nibble <= data[23:20];
    4'h6:              nibble <= data[27:24];
    4'h7:              nibble <= data[31:28];
    4'h8:              nibble <= data[35:32];
    4'h9:              nibble <= data[39:36];
    4'hA:              nibble <= data[43:40];
    4'hB:              nibble <= data[47:44];
    4'hC:              nibble <= data[51:48];
    4'hD:              nibble <= data[55:52];
    4'hE:              nibble <= data[59:56];
    4'hF:              nibble <= data[63:60];
  endcase

always @(nibble)
  case (nibble)
    4'h0: dots <= 40'b00111110_01010001_01001001_01000101_00111110;
    4'h1: dots <= 40'b00000000_01000010_01111111_01000000_00000000;
    4'h2: dots <= 40'b01100010_01010001_01001001_01001001_01000110;
    4'h3: dots <= 40'b00100010_01000001_01001001_01001001_00110110;
    4'h4: dots <= 40'b00011000_00010100_00010010_01111111_00010000;
    4'h5: dots <= 40'b00100111_01000101_01000101_01000101_00111001;
    4'h6: dots <= 40'b00111100_01001010_01001001_01001001_00110000;
    4'h7: dots <= 40'b00000001_01110001_00001001_00000101_00000011;
    4'h8: dots <= 40'b00110110_01001001_01001001_01001001_00110110;
    4'h9: dots <= 40'b00000110_01001001_01001001_00101001_00011110;
    4'hA: dots <= 40'b01111110_00001001_00001001_00001001_01111110;
    4'hB: dots <= 40'b01111111_01001001_01001001_01001001_00110110;
    4'hC: dots <= 40'b00111110_01000001_01000001_01000001_00100010;
    4'hD: dots <= 40'b01111111_01000001_01000001_01000001_00111110;
    4'hE: dots <= 40'b01111111_01001001_01001001_01001001_01000001;
    4'hF: dots <= 40'b01111111_00001001_00001001_00001001_00000001;
    endcase

endmodule
```

A.10 Selective Delay Module

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    22:24:33 11/15/2011
// Design Name:
// Module Name:    delayN
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module delayV #(parameter NDELAY = 3, parameter SIZE = 10)(clk,in,out);
input clk;
input [SIZE-1:0] in;
output [SIZE-1:0] out;
reg [SIZE-1:0] shift_regs [NDELAY:0];
reg [4:0] i;
always @ (posedge clk) begin
shift_regs[0] <= in;
for (i = 1; i<NDELAY+1; i=i+1) begin
shift_regs[i] <= shift_regs[i-1];
end
end
assign out = shift_regs[NDELAY];
endmodule // delayN
```

A.11 Debounce Module

```
///////////////////////////////////////////////////////////////////////////////
//
// Pushbutton Debounce Module
//
///////////////////////////////////////////////////////////////////////////////

module debounce (reset, clk, noisy, clean);
  input reset, clk, noisy;
  output clean;

  parameter NDELAY = 650000;
  parameter NBITS = 20;

  reg [NBITS-1:0] count;
  reg xnew, clean;

  always @(posedge clk)
    if (reset) begin xnew <= noisy; clean <= noisy; count <= 0; end
    else if (noisy != xnew) begin xnew <= noisy; count <= 0; end
    else if (count == NDELAY) clean <= xnew;
    else count <= count+1;

endmodule
```

A.12 NTSC Decoder Module

```
//
// 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
```

A.13 Threshold Filter Module

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    14:04:13 11/20/2011
// Design Name:
// Module Name:    new_treshold_filter
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module threshold_filter(clk,reset, x, y, factor, match, x0,x1,y0,y1,x2,y2, x3,y3, x4,y4,
x5,y5, x_accumulated_op, counted_value_op, x_com_op);

        input  clk;

        input [10:0] x;

        input [9:0] y;
        input [3:0] factor;
        input match;
        input reset;
        output wire [24:0] x_accumulated_op;
        output wire [24:0] counted_value_op;
        output wire [24:0] x_com_op;

        //markers x and y coordinates, lower 10 bits are for y, and upper 11 are for x
        output wire [10:0] x0,x1,x2,x3,x4,x5;
        output wire [9:0] y0,y1,y2,y3,y4,y5;

        reg [24:0] x_com [31:0];

        reg [24:0] y_com [31:0];

        reg [24:0] count [31:0];

        wire [24:0] x_com_wire, y_com_wire, counted_value_wire;

        reg [3:0] i = 0;

        wire [5:0] hindex,vindex,hnew_index;

        assign hindex = x [10:5];       //top 5 bits of count assigned to a wire

        assign vindex = y [9:5];
        wire [10:0] pixel_count_minimum;

        reg [10:0] xx [15:0];
        reg [9:0] yy [15:0];                                        //reg to store
variables

        //reg [10:0] xx;
        //reg [9:0] yy;                                     //reg to store variables

        wire reset;
        wire[11:0] k;
        reg [11:0] td [7:0];
        assign x0 = xx[0];
        assign y0 = yy[0];
        assign x1 = xx[1];
        assign y1 = yy[1];
        assign x2 = xx[2];
        assign y2 = yy[2];
        assign x3 = xx[3];
        assign y3 = yy[3];
        assign x4 = xx[4];
        assign y4 = yy[4];
        assign x5 = xx[5];
        assign y5 = yy[5];
        assign counted_value_wire = count[hindex];
        assign x_accumulated_op = x_com_wire;
        assign counted_value_op = counted_value_wire;
        assign x_com_op = y_com_wire;
        assign k = hindex + 41*vindex;
assign pixel_count_minimum = (50 * factor);

always @(posedge clk) begin
        if(reset || ((x == 0) && (y[4:0]==5'b00000))) begin
                                count[0] <= 0;
                //initialize count completely for the entire row

                                x_com[0] <= 0;
                //initialize x accumulator completely for the entire row

                                y_com[0] <= 0;
                //initialize y accumulator completely for the entire row

                                count[1] <= 0;
                //initialize count completely for the entire row

                                x_com[1] <= 0;
                //initialize x accumulator completely for the entire row

                                y_com[1] <= 0;
                //initialize y accumulator completely for the entire row

                                count[2] <= 0;
                //initialize count completely for the entire row

                                x_com[2] <= 0;
                //initialize x accumulator completely for the entire row

                                y_com[2] <= 0;
                //initialize y accumulator completely for the entire row

                                count[3] <= 0;
                //initialize count completely for the entire row

                                x_com[3] <= 0;
                //initialize x accumulator completely for the entire row

                                y_com[3] <= 0;
                //initialize y accumulator completely for the entire row

                                count[4] <= 0;
                //initialize count completely for the entire row

                                x_com[4] <= 0;
                //initialize x accumulator completely for the entire row

                                y_com[4] <= 0;
                //initialize y accumulator completely for the entire row
```

```
                        count[5] <= 0;
//initialize count completely for the entire row

                        x_com[5] <= 0;
//initialize x accumulator completely for the entire row

                        y_com[5] <= 0;
//initialize y accumulator completely for the entire row

                        count[6] <= 0;
//initialize count completely for the entire row

                        x_com[6] <= 0;
//initialize x accumulator completely for the entire row

                        y_com[6] <= 0;
//initialize y accumulator completely for the entire row

                        count[7] <= 0;
//initialize count completely for the entire row

                        x_com[7] <= 0;
//initialize x accumulator completely for the entire row

                        y_com[7] <= 0;
//initialize y accumulator completely for the entire row

                        count[8] <= 0;
//initialize count completely for the entire row

                        x_com[8] <= 0;
//initialize x accumulator completely for the entire row

                        y_com[8] <= 0;
//initialize y accumulator completely for the entire row

                        count[9] <= 0;
//initialize count completely for the entire row

                        x_com[9] <= 0;
//initialize x accumulator completely for the entire row

                        y_com[9] <= 0;
//initialize y accumulator completely for the entire row

                        count[10] <= 0;
//initialize count completely for the entire row
```

```
                    x_com[10] <= 0;
//initialize x accumulator completely for the entire row

                    y_com[10] <= 0;
//initialize y accumulator completely for the entire row

                    count[11] <= 0;
//initialize count completely for the entire row

                    x_com[11] <= 0;
//initialize x accumulator completely for the entire row

                    y_com[11] <= 0;
//initialize y accumulator completely for the entire row

                    count[12] <= 0;
//initialize count completely for the entire row

                    x_com[12] <= 0;
//initialize x accumulator completely for the entire row

                    y_com[12] <= 0;
//initialize y accumulator completely for the entire row

                    count[13] <= 0;
//initialize count completely for the entire row

                    x_com[13] <= 0;
//initialize x accumulator completely for the entire row

                    y_com[13] <= 0;
//initialize y accumulator completely for the entire row

                    count[14] <= 0;
//initialize count completely for the entire row

                    x_com[14] <= 0;
//initialize x accumulator completely for the entire row

                    y_com[14] <= 0;
//initialize y accumulator completely for the entire row

                    count[15] <= 0;
//initialize count completely for the entire row

                    x_com[15] <= 0;
//initialize x accumulator completely for the entire row
```

```
                    y_com[15] <= 0;
//initialize y accumulator completely for the entire row

                    count[16] <= 0;
//initialize count completely for the entire row

                    x_com[16] <= 0;
//initialize x accumulator completely for the entire row

                    y_com[16] <= 0;
//initialize y accumulator completely for the entire row

                    count[17] <= 0;
//initialize count completely for the entire row

                    x_com[17] <= 0;
//initialize x accumulator completely for the entire row

                    y_com[17] <= 0;
//initialize y accumulator completely for the entire row

                    count[18] <= 0;
//initialize count completely for the entire row

                    x_com[18] <= 0;
//initialize x accumulator completely for the entire row

                    y_com[18] <= 0;
//initialize y accumulator completely for the entire row

                    count[19] <= 0;
//initialize count completely for the entire row

                    x_com[19] <= 0;
//initialize x accumulator completely for the entire row

                    y_com[19] <= 0;
//initialize y accumulator completely for the entire row

                    count[20] <= 0;
//initialize count completely for the entire row

                    x_com[20] <= 0;
//initialize x accumulator completely for the entire row

                    y_com[20] <= 0;
//initialize y accumulator completely for the entire row
```

```
                    count[21] <= 0;
//initialize count completely for the entire row

                    x_com[21] <= 0;
//initialize x accumulator completely for the entire row

                    y_com[21] <= 0;
//initialize y accumulator completely for the entire row

                    count[22] <= 0;
//initialize count completely for the entire row

                    x_com[22] <= 0;
//initialize x accumulator completely for the entire row

                    y_com[22] <= 0;
//initialize y accumulator completely for the entire row

                    count[23] <= 0;
//initialize count completely for the entire row

                    x_com[23] <= 0;
//initialize x accumulator completely for the entire row

                    y_com[23] <= 0;
//initialize y accumulator completely for the entire row

                    count[24] <= 0;
//initialize count completely for the entire row

                    x_com[24] <= 0;
//initialize x accumulator completely for the entire row

                    y_com[24] <= 0;
//initialize y accumulator completely for the entire row

                    count[25] <= 0;
//initialize count completely for the entire row

                    x_com[25] <= 0;
//initialize x accumulator completely for the entire row

                    y_com[25] <= 0;
//initialize y accumulator completely for the entire row

                    count[26] <= 0;
//initialize count completely for the entire row
```

```
                        x_com[26] <= 0;
//initialize x accumulator completely for the entire row

                        y_com[26] <= 0;
//initialize y accumulator completely for the entire row

                        count[27] <= 0;
//initialize count completely for the entire row

                        x_com[27] <= 0;
//initialize x accumulator completely for the entire row

                        y_com[27] <= 0;
//initialize y accumulator completely for the entire row

                        count[28] <= 0;
//initialize count completely for the entire row

                        x_com[28] <= 0;
//initialize x accumulator completely for the entire row

                        y_com[28] <= 0;
//initialize y accumulator completely for the entire row

                        count[29] <= 0;
//initialize count completely for the entire row

                        x_com[29] <= 0;
//initialize x accumulator completely for the entire row

                        y_com[29] <= 0;
//initialize y accumulator completely for the entire row

                        count[30] <= 0;
//initialize count completely for the entire row

                        x_com[30] <= 0;
//initialize x accumulator completely for the entire row

                        y_com[30] <= 0;
//initialize y accumulator completely for the entire row

                        count[31] <= 0;
//initialize count completely for the entire row

                        x_com[31] <= 0;
//initialize x accumulator completely for the entire row
```

```
                                        y_com[31] <= 0;
                                //initialize y accumulator completely for the entire row


                        end
                        else begin
                                if(match) begin
                                        count[hindex] <= count[hindex] + 1;
                                end

                        if ((k-84) >= 0) begin
                                if ((count[hindex] >= pixel_count_minimum)
                                                && (td[6] != (k - 1)) && (td[6] != (k - 42)) && (td[6] !=
(k - 41)) && (td[6] != (k - 40)) && (td[6] != (k - 2)) && (td[6] != (k - 43)) && (td[6] != (k - 84))
&& (td[6] != (k - 83)) && (td[6] != (k - 82)) && (td[6] != (k - 81))
                                                && (td[1] != (k - 1)) && (td[1] != (k - 42)) && (td[1] !=
(k - 41)) && (td[1] != (k - 40)) && (td[1] != (k - 2)) && (td[1] != (k - 43)) && (td[1] != (k - 84))
&& (td[1] != (k - 83)) && (td[1] != (k - 82)) && (td[1] != (k - 81))
                                                && (td[2] != (k - 1)) && (td[2] != (k - 42)) && (td[2] !=
(k - 41)) && (td[2] != (k - 40)) && (td[2] != (k - 2)) && (td[2] != (k - 43)) && (td[2] != (k - 84))
&& (td[2] != (k - 83)) && (td[2] != (k - 82)) && (td[2] != (k - 81))
                                                && (td[3] != (k - 1)) && (td[3] != (k - 42)) && (td[3] !=
(k - 41)) && (td[3] != (k - 40)) && (td[3] != (k - 2)) && (td[3] != (k - 43)) && (td[3] != (k - 84))
&& (td[3] != (k - 83)) && (td[3] != (k - 82)) && (td[3] != (k - 81))
                                                && (td[4] != (k - 1)) && (td[4] != (k - 42)) && (td[4] !=
(k - 41)) && (td[4] != (k - 40)) && (td[4] != (k - 2)) && (td[4] != (k - 43)) && (td[4] != (k - 84))
&& (td[4] != (k - 83)) && (td[4] != (k - 82)) && (td[4] != (k - 81))
                                                && (td[5] != (k - 1)) && (td[5] != (k - 42)) && (td[5] !=
(k - 41)) && (td[5] != (k - 40)) && (td[5] != (k - 2)) && (td[5] != (k - 43)) && (td[5] != (k - 84))
&& (td[5] != (k - 83)) && (td[5] != (k - 82)) && (td[5] != (k - 81))
                                                && (td[7] != (k - 1)) && (td[7] != (k - 42)) && (td[7] !=
(k - 41)) && (td[7] != (k - 40)) && (td[7] != (k - 2)) && (td[7] != (k - 43)) && (td[7] != (k - 84))
&& (td[7] != (k - 83)) && (td[7] != (k - 82)) && (td[7] != (k - 81))) begin
                                        x_com[hindex] <= x - 15;
                                        y_com[hindex] <= y - 15;
                                end
                                else begin
                                        x_com[hindex] <= x_com[hindex];
                                        y_com[hindex] <= y_com[hindex];
                                end
else if ((k-44) >= 0) begin//from here
                if ((x[4:0] == 5'b11101)
                        && (y[4:0] == 5'b11111)
                        && (count[hindex] >= pixel_count_minimum)
                        && (td[6] != (k - 1)) && (td[6] != (k - 42)) && (td[6] != (k - 41)) && (td[6] !=
(k - 44)) && (td[6] != (k - 2)) && (td[6] != (k - 43)) //&& //(td[6] != (k - 84)) && (td[6] != (k -
83)) && (td[6] != (k - 82)) && (td[6] != (k - 85))
```

```
                    && (td[1] != (k - 1)) && (td[1] != (k - 42)) && (td[1] != (k - 41)) && (td[1] !=
(k - 44)) && (td[1] != (k - 2)) && (td[1] != (k - 43)) //&& //(td[1] != (k - 84)) && (td[1] != (k -
83)) && (td[1] != (k - 82)) && (td[1] != (k - 85))
                    && (td[2] != (k - 1)) && (td[2] != (k - 42)) && (td[2] != (k - 41)) && (td[2] !=
(k - 44)) && (td[2] != (k - 2)) && (td[2] != (k - 43)) //&& //(td[2] != (k - 84)) && (td[2] != (k -
83)) && (td[2] != (k - 82)) && (td[2] != (k - 85))
                    && (td[3] != (k - 1)) && (td[3] != (k - 42)) && (td[3] != (k - 41)) && (td[3] !=
(k - 44)) && (td[3] != (k - 2)) && (td[3] != (k - 43)) //&& //(td[3] != (k - 84)) && (td[3] != (k -
83)) && (td[3] != (k - 82)) && (td[3] != (k - 85))
                    && (td[4] != (k - 1)) && (td[4] != (k - 42)) && (td[4] != (k - 41)) && (td[4] !=
(k - 44)) && (td[4] != (k - 2)) && (td[4] != (k - 43)) ///&& //(td[4] != (k - 84)) && (td[4] != (k -
83)) && (td[4] != (k - 82)) && (td[4] != (k - 85))
                    && (td[5] != (k - 1)) && (td[5] != (k - 42)) && (td[5] != (k - 41)) && (td[5] !=
(k - 44)) && (td[5] != (k - 2)) && (td[5] != (k - 43)) //&& //(td[5] != (k - 84)) && (td[5] != (k -
83)) && (td[5] != (k - 82)) && (td[5] != (k - 85))
                    && (td[7] != (k - 1)) && (td[7] != (k - 42)) && (td[7] != (k - 41)) && (td[7] !=
(k - 44)) && (td[7] != (k - 2)) && (td[7] != (k - 43))) begin// && (td[7] != (k - 84)) && (td[7] !=
(k - 83)) && (td[7] != (k - 82)) && (td[7] != (k - 85))) begin
                x_com[hindex] <= x - 15;
                y_com[hindex] <= y - 15;
                i <= i+1;
            end
            else begin
                x_com[hindex] <= x_com[hindex];
                y_com[hindex] <= y_com[hindex];
            end
        end
        else begin
            if ((x[4:0] == 5'b11101)
                    && (y[4:0] == 5'b11111)
                    && (count[hindex] >= pixel_count_minimum))begin
                x_com[hindex] <= x - 15;
                y_com[hindex] <= y - 15;
                i <= i+1;
            end
            else begin
                x_com[hindex] <= x_com[hindex];
                y_com[hindex] <= y_com[hindex];
            end

                            if((i <= 5) && (x_com[hindex] > 0) && (y_com[hindex] > 0)) begin
                                    xx[i] <= x_com[hindex];

                                    yy[i] <= y_com[hindex];


                                    td[i] <= k;
                                    i <= i+1;
```

```
                        end

                        if((x == 0) && (y == 0)) begin
                                i <= 0;
                        end
                end
        end
        end
        end
endmodule
```

A.14 Double Buffer Module

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    20:39:49 11/29/2011
// Design Name:
// Module Name:    double_buffer
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module double_buffer(input vclock,
input reset,
input [10:0] hcount,
input [9:0] vcount,
input hsync,
input vsync,
input antialiased,
input blank,
input [10:0] write_x,
input [9:0] write_y,
input [1:0] colour_denote,
input write_enable,
//input write_buf_switch,
output vhsync,
output vvsync,
output vbalnk,
output reg [23:0] pixel);


parameter DELAY = 2;
reg [18:0] addr0;
reg [18:0] addr1;
reg [1:0] write_data0;
reg [1:0] write_data1;
reg we_0;
reg we_1;
reg erase_cycle;
reg write_buf_select;
wire [1:0] douta1,douta2;
reg [DELAY:0] hsync_delay, vsync_delay, blank_delay;
wire [18:0] sync_addr;
wire [18:0] drawing_addr;
assign sync_addr = (720 * vcount) + hcount;
assign drawing_addr = (720 * write_y) + write_x;
assign vhsync = hsync_delay[0];
assign vvsync = vsync_delay[0];
assign vblank = blank_delay[0];

bram store0(.clka(vclock),
                                        .dina(write_data0),
                                        .addra(addr0),
                                        .wea(we_0),
                                        .douta(douta1));

bram store1(.clka(vclock),
                                        .dina(write_data1),
                                        .addra(addr1),
                                        .wea(we_1),
                                        .douta(douta2));


always @(posedge vclock) begin // delay the video sync and blank signals to account for
latency in reading and writing ZBT memory
hsync_delay <= (reset? 0 : {hsync,hsync_delay[DELAY:1]});
vsync_delay <= (reset? 0 : {vsync,vsync_delay[DELAY:1]});
blank_delay <= (reset? 0 : {blank,blank_delay[DELAY:1]});

if(hcount<721 && vcount<526) begin
        if (antialiased == 1) begin
                        if(douta1 == 3) pixel <= 24'b1111_1111_1111_1111_1111_1111;
                        else if (douta2 == 3) pixel <=
24'b1111_1111_1111_1111_1111_1111;
```

```
                        else if (douta1 == 2) pixel <=
24'b0111_1111_0111_1111_0111_1111;
                        else if (douta2 == 2) pixel <=
24'b0111_1111_0111_1111_0111_1111;
                        else if (douta1 == 1) pixel <=
24'b0011_1111_0011_1111_0011_1111;
                        else if (douta2 == 1) pixel <=
24'b0011_1111_0011_1111_0011_1111;
                        else pixel <= 0;
                end
                else if (antialiased == 0) begin
                        if(douta1 == 3) pixel <= 24'b1111_1111_1111_1111_1111_1111;
                        else if (douta2 == 3) pixel <=
24'b1111_1111_1111_1111_1111_1111;
                        else pixel <=0;
                end
end
else pixel <= 0;


if (reset) begin
        erase_cycle <= 1;
        write_buf_select <= 0;
        we_0 <= 0;
        we_1 <= 0;
        addr0 <= 0;
        addr1 <= 0;
        write_data0 <= 0;
        write_data1 <= 0;
end
        else begin

                if (vsync_delay[DELAY] && !vsync) begin // on a negative edge transition
in vsync, either:
                        if (erase_cycle) begin // enter the draw cycle
                                erase_cycle <= 0;
                        end
                        else begin // or swap buffers and enter the erase cycle
                                        erase_cycle <= 1;
                                        write_buf_select <= !write_buf_select;
                        end
                end

                if (write_buf_select == 0) begin
                        if (erase_cycle) begin
                                addr0 <= sync_addr;
                                write_data0 <= 2'b00;
                                we_0 <= (hcount < 721 && vcount < 526);
```

```
				end
				else begin
					addr0 <= drawing_addr;
					write_data0 <= colour_denote;
					we_0 <= write_enable;
				end
					we_1 <= 0;
					addr1 <= (hcount < 721 && vcount < 526? sync_addr :0);
			end
			else begin
				if (erase_cycle) begin
					addr1 <= sync_addr;
					write_data1 <= 2'b00;
					we_1 <= (hcount < 721 && vcount < 526);
				end
				else begin
					addr1 <= drawing_addr;
					write_data1 <= colour_denote;
					we_1 <= write_enable;
				end
				we_0 <= 0;
				addr0 <= (hcount < 721 && vcount < 526? sync_addr :0);
			end
		end
end
endmodule
```

A.15 Face Generator Module

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    13:17:56 11/16/2011
// Design Name:
// Module Name:    face_generator
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
```

```
//
////////////////////////////////////////////////////////////////////////////////
module face_generator
   #(parameter RADIUS = 32,
             COLOR = 24'b1111_1111_1111_1111_1111_1111)
   (input [10:0] x,hcount,
    input [9:0] y,vcount,
          input vclockn,
          //input draw_face,
          output reg [23:0] pixel);

          reg [24:0] z; //variables for pipelining
          reg [21:0] za;
          reg [20:0] zb;
          reg [11:0] rd;
          reg [10:0] dummy_x;


   always @(posedge vclockn)  begin
                    if (hcount>(x+RADIUS)) za<=(hcount-(x+RADIUS))*(hcount-
(x+RADIUS));//checking the higher values and subtracting the lower from higher
                    else if ((x+RADIUS)>hcount) za<=((x+RADIUS)-hcount)*((x+RADIUS)-
hcount);
                    if (vcount>(y+RADIUS)) zb<=(vcount-(y+RADIUS))*(vcount-(y+RADIUS));
                    else if ((y+RADIUS)>vcount) zb<=((y+RADIUS)-vcount)*((y+RADIUS)-
vcount);
                    z<=za+zb;
                    if (z<=(RADIUS*RADIUS))
      pixel <= COLOR;
      else pixel <= 0;
   end
endmodule
```

A.16 Game Color Logic Module

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    21:57:14 11/30/2011
// Design Name:
// Module Name:    gamelogic
// Project Name:
// Target Devices:
// Tool versions:
// Description:
```

```
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////
module gamelogic#(parameter RADIUS = 15,
              COLOR1 = 24'b1111_1111_1111_1111_1111_1111,
                                                      COLOR2 =
24'b1111_1111_0000_0000_0000_0000)
   (input [10:0] lefthandx,hcount,righthandx,leftlegx,rightlegx,torsoupx,torsodownx,
    input [9:0] lefthandy,righthandy,leftlegy,rightlegy,torsoupy,torsodowny,vcount,
            input vclockn,
            input reset,
            output reg [23:0] pixel1,pixel2,pixel3,pixel4);

            reg [24:0] z1,z2,z3,z4; //variables for pipelining
            reg [21:0] za1,za2,za3,za4;
            reg [20:0] zb1,zb2,zb3,zb4;


   always @(posedge vclockn)  begin
                                if (reset) begin
                                pixel1 <= 0;
                                pixel2 <= 0;
                                pixel3 <= 0;
                                pixel4 <= 0;
                                end
                                else begin
                                //left hand
                                if (hcount>(lefthandx+RADIUS)) za1<=(hcount-
(lefthandx+RADIUS))*(hcount-(lefthandx+RADIUS));
                                else if ((lefthandx+RADIUS)>hcount) za1<=((lefthandx+RADIUS)-
hcount)*((lefthandx+RADIUS)-hcount);
                                if (vcount>(lefthandy+RADIUS)) zb1<=(vcount-
(lefthandy+RADIUS))*(vcount-(lefthandy+RADIUS));
                                else if ((lefthandy+RADIUS)>vcount) zb1<=((lefthandy+RADIUS)-
vcount)*((lefthandy+RADIUS)-vcount);

                                //right hand
                                if (hcount>(righthandx+RADIUS)) za2<=(hcount-
(righthandx+RADIUS))*(hcount-(righthandx+RADIUS));
                                else if ((righthandx+RADIUS)>hcount)
za2<=((righthandx+RADIUS)-hcount)*((righthandx+RADIUS)-hcount);
                                if (vcount>(righthandy+RADIUS)) zb2<=(vcount-
(righthandy+RADIUS))*(vcount-(righthandy+RADIUS));
```

```
                        else if ((righthandy+RADIUS)>vcount)
zb2<=((righthandy+RADIUS)-vcount)*((righthandy+RADIUS)-vcount);

                        //left leg
                        if (hcount>(leftlegx+RADIUS)) za3<=(hcount-
(leftlegx+RADIUS))*(hcount-(leftlegx+RADIUS));
                        else if ((leftlegx+RADIUS)>hcount) za3<=((leftlegx+RADIUS)-
hcount)*((leftlegx+RADIUS)-hcount);
                        if (vcount>(leftlegy+RADIUS)) zb3<=(vcount-
(leftlegy+RADIUS))*(vcount-(leftlegy+RADIUS));
                        else if ((leftlegy+RADIUS)>vcount) zb3<=((leftlegy+RADIUS)-
vcount)*((leftlegy+RADIUS)-vcount);

                        //right leg
                        if (hcount>(rightlegx+RADIUS)) za4<=(hcount-
(rightlegx+RADIUS))*(hcount-(rightlegx+RADIUS));
                        else if ((rightlegx+RADIUS)>hcount) za4<=((rightlegx+RADIUS)-
hcount)*((rightlegx+RADIUS)-hcount);
                        if (vcount>(rightlegy+RADIUS)) zb4<=(vcount-
(rightlegy+RADIUS))*(vcount-(rightlegy+RADIUS));
                        else if ((rightlegy+RADIUS)>vcount) zb4<=((rightlegy+RADIUS)-
vcount)*((rightlegy+RADIUS)-vcount);


                        z1<=za1+zb1;
                        z2<=za2+zb2;
                        z3<=za3+zb3;
                        z4<=za4+zb4;
                        if (z1<=(RADIUS*RADIUS)) begin
                                if(torsoupy <= lefthandy - RADIUS) pixel1 <= COLOR1;
                                else pixel1 <= COLOR2;
                        end
                        else pixel1 <= 0;

                        if (z2<=(RADIUS*RADIUS)) begin
                                if(torsoupy <= righthandy - RADIUS) pixel2 <= COLOR1;
                                else pixel2 <= COLOR2;
                        end
                        else pixel2 <= 0;

                        if (z3<=(RADIUS*RADIUS)) begin
                                if(torsodowny <= leftlegy - RADIUS) pixel3 <= COLOR1;
                                else pixel3 <= COLOR2;
                        end
                        else pixel3 <= 0;

                        if (z4<=(RADIUS*RADIUS)) begin
                                if(torsodowny <= rightlegy - RADIUS) pixel4 <= COLOR1;
```

```
                                        else pixel4 <= COLOR2;
                              end
                              else pixel4 <= 0;

                              end//else
                    end//posedge
endmodule
```

A.17 Line Drawer Module

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: MIT 6.111
// Engineer: Raghavendra Srinivasan
//
// Create Date:    12:50:25 11/09/2011
// Design Name:          Line Drawing module
// Module Name:    line_drawer (with anti aliasing)
// Project Name:   Motion Sensing and Graphic reproduction using artificaial intelligence
// Target Devices:
// Tool versions:
// Description:   This module takes two points as the input and draws a line between
them.This
//               uses antialiasing too. :)
//
// Dependencies:  Just the 2D points it receives.
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
                                        module line_drawer (
                                        input vclock,//clock of the diaplay
                                        input reset,
                                        input vsync,
                                        input start_drawing,//can be a pulse or a complete high
                                        input [20:0]
                                        torsoup,torsodown,leftarm,rightarm,leftleg,rightleg,
                                        output reg line_ready,//pulse
                                        output reg [10:0] final_display_x_white,

                                                  final_display_x_dark_gray_one,

                                                  final_display_x_dark_gray_two,
          final_display_x_gray_one,

          final_display_x_gray_two,
                                                  write_x,//imp
output reg [9:0]
final_display_y_white,final_display_y_dark_gray_one,

          final_display_y_dark_gray_two,final_display_y_gra
y_one,

          final_display_y_gray_two,
                                                  write_y,//imp
output reg display_enable,
output reg write_enable, //imp
output reg [1:0] colour_denote, //imp
output reg finished_drawing,
output reg [3:0] state);

reg slope;
reg signed [11:0] delta_x;
reg signed [10:0] delta_y;
reg signed [11:0] error;
reg signed [1:0] ystep;
reg signed [11:0] x0,x1, x;
reg signed [10:0] y0,y1,y;
//reg [2:0] state;//change this back after analyzing
reg signed [11:0] display_x_white,

          display_x_dark_gray_one,

          display_x_dark_gray_two,

          display_x_gray_one,

          display_x_gray_two;
reg signed [10:0]
display_y_white,display_y_dark_gray_one,

display_y_dark_gray_two,display_y_gray_one,

display_y_gray_two;
reg [2:0] memory_fill_denote;
//reg [20:0] addr;
reg draw_drawing;
reg [2:0] connection_denoter =0;
//reg [1:0] colour_denote;
reg [19:0] address_count;
```

```
//reg draw_command; //since start_drawing comes only
once, i am using this register.
//wire [1:0] douta;
/*bram store(.clka(vclock),
                                        .dina(colour_denote),
                                        .addra(addr),
                                        .wea(write_enable),
                                        .douta(douta));*/
//connection 1 is torsoup and leftarem
parameter INITIAL =0;
parameter SLOPE_CHECK=1;
parameter SWAPS=2;
parameter ERROR_DEFINE =3;
parameter DRAW_LINE_CALCULATE =4;
parameter LOGICAL_ASSIGN =5;
parameter MEMORY =6;
parameter MEMORY_CLEAR_ONE =7;
parameter MEMORY_CLEAR_TWO =8;

        always @(posedge vclock) begin
                /*if (antialiased == 1) begin
                        pixel <= ((douta1 == 3) || (douta2 ==
3)) ? 24'b1111_1111_1111_1111_1111_1111 : // gotta
change colors
                                                        ((douta1 == 2)
|| (douta2 == 2)) ? 24'b0111_1111_0111_1111_0111_1111
:
                                                        ((douta1 == 1)
|| (douta2 == 1)) ? 24'b0011_1111_0011_1111_0011_1111
: 0;
                end// for write_enable = 0 if
                else if (antialiased == 0) begin
                                pixel <= ((douta1 == 3) ||
(douta2 == 3)) ? 24'b1111_1111_1111_1111_1111_1111
:0;// gotta change colour
                end// for else if above*/

                /*if (antialiased == 1) begin
                        if(douta1 == 3) pixel <=
24'b1111_1111_1111_1111_1111_1111;
                        else if (douta2 == 3) pixel <=
24'b1111_1111_1111_1111_1111_1111;
                        else if (douta1 == 2) pixel <=
24'b0111_1111_0111_1111_0111_1111;
                        else if (douta2 == 2) pixel <=
24'b0111_1111_0111_1111_0111_1111;
                        else if (douta1 == 1) pixel <=
24'b0011_1111_0011_1111_0011_1111;
```

```
                                else if (douta2 == 1) pixel <=
24'b0011_1111_0011_1111_0011_1111;
                                else pixel <= 0;
                        end
                        else if (antialiased == 0) begin
                                if(douta1 == 3) pixel <=
24'b1111_1111_1111_1111_1111_1111;
                                else if (douta2 == 3) pixel <=
24'b1111_1111_1111_1111_1111_1111;
                                else pixel <=0;
                        end*/

                        if(start_drawing) draw_drawing<=1;

                        if (reset ll !vsync) begin
                                memory_fill_denote <=0;
                                write_enable <= 0;
                                finished_drawing <=0;
                                final_display_x_white <= 0;
                                final_display_y_white <= 0;
                                final_display_x_dark_gray_one <=0;
                                final_display_y_dark_gray_one <=0;
                                final_display_x_gray_one<=0;
                                final_display_y_gray_one<=0;
                                final_display_x_dark_gray_two <=0;
                                final_display_y_dark_gray_two <=0;
                                final_display_x_gray_two<=0;
                                final_display_y_gray_two<=0;
                                delta_x <= 0;
                                delta_y <= 0;
                                error <= 0;
                                ystep <= 1;
                                display_enable <= 0;
                                slope <= 0;
                                state <= INITIAL;
                                line_ready <= 1;
                        end
                        else begin
                                case(state)
                                        INITIAL:begin

                finished_drawing<=0;

                if(draw_drawing) begin

                line_ready <=0;

                display_enable <= 0;
```

```
state <= SLOPE_CHECK;

if (connection_denoter == 0) begin

x0 <= torsoup[20:10];

y0 <= torsoup[9:0];

x1 <= leftarm[20:10];

y1 <= leftarm[9:0];

end

else if (connection_denoter == 1) begin

x0 <= torsoup[20:10];

y0 <= torsoup[9:0];

x1 <= rightarm[20:10];

y1 <= rightarm[9:0];

end

else if (connection_denoter == 2) begin

x0 <= torsodown[20:10];

y0 <= torsodown[9:0];

x1 <= rightleg[20:10];

y1 <= rightleg[9:0];

end

else if (connection_denoter == 3) begin

x0 <= torsodown[20:10];

y0 <= torsodown[9:0];

x1 <= leftleg[20:10];

y1 <= leftleg[9:0];
```

```
        end

        else if (connection_denoter == 4) begin

        x0 <= torsodown[20:10];

        y0 <= torsodown[9:0];

        x1 <= torsoup[20:10];

        y1 <= torsoup[9:0];

        end

        else if (connection_denoter == 5) begin

                connection_denoter <=0;

                draw_drawing <= 0;

                finished_drawing <= 1;

                state <= INITIAL;

        end


        end
                                                        end
                                    SLOPE_CHECK:begin //in
this state we check if slope is greater than 1.

        if (y1 > y0) begin

        if (x1 > x0) begin

                slope <= ((y1 - y0) > (x1 - x0));

        end

        else begin

                slope <= ((y1 - y0) > (x0 - x1));

        end
        end

        else begin

        if (x1 > x0) begin

                slope <= ((y0 - y1) > (x1 - x0));

        end

        else begin

                slope <= ((y0 - y1) > (x0 - x1));

        end

        end

        state <= SWAPS;

        end//for SLOPE_CHECK

                                        SWAPS:begin // in
this state we swap if slope is not valid.
                                                        if
(slope) begin

        if (y0 > y1) begin

        x0 <= y1;

        x1 <= y0;

        y0 <= x1;

        y1 <= x0;

        delta_x <= y0 - y1;

        delta_y <= (x1 > x0? x1-x0 :x0-x1);

        end

        else begin

        x0 <= y0;
```

```
            x1 <= y1;

            y0 <= x0;

            y1 <= x1;

            delta_x <= y1 - y0;

            delta_y <= (x1 > x0? x1-x0 :x0-x1);

            end
                                                                        end
                                                                        else
begin

            if (x0 > x1) begin

            x0 <= x1;

            x1 <= x0;

            y0 <= y1;

            y1 <= y0;

            delta_x <= x0-x1;

            delta_y <= (y1 > y0? y1-y0 :y0-y1);

            end

            else begin

            delta_x <= x1-x0;

            delta_y <= (y1 > y0? y1-y0 :y0-y1);

            end
                                                                        end
                                                                        state
<= ERROR_DEFINE;
                                                    end//for state SWAPS
                                    ERROR_DEFINE:begin //in this state
we define various essential parameters

            error <= {delta_x[11],delta_x[11:1]}; //delta_x / 2 for
a signed value
```

```
            ystep <= (y0 < y1 ? 1 : -1);

            x <= x0;

            y <= y0;

            state <= DRAW_LINE_CALCULATE; // draw line
                                                                    end //
for state ERROR_DEFINE
                          DRAW_LINE_CALCULATE:begin

            display_enable <= 1;

            if (slope) begin

            display_x_white <= y[9:0];

            display_y_white <= x[10:0];

            display_x_gray_one  <= y[9:0]+1;

            display_y_gray_one  <= x[10:0]+1;

            display_x_gray_two  <= y[9:0]-2;

            display_y_gray_two  <= x[10:0]-2;

            display_x_dark_gray_one <=y[9:0]+2;

            display_y_dark_gray_one <=x[10:0]+2;

            display_x_dark_gray_two <=y[9:0]-4;

            display_y_dark_gray_two <=x[10:0]-4;

            end

            else begin

            display_x_white <= x[10:0];

            display_y_white <= y[9:0];

            display_x_gray_one  <= x[10:0]+1;

            display_y_gray_one  <= y[9:0]+1;
```

```
display_x_gray_two  <= x[10:0]-2;

display_y_gray_two  <= y[9:0]-2;

display_x_dark_gray_one <=x[10:0]+2;

display_y_dark_gray_one <=y[9:0]+2;

display_x_dark_gray_two <=x[10:0]-4;

display_y_dark_gray_two <=y[9:0]-4;

end

x <= x + 1;

state <= LOGICAL_ASSIGN;

if (error - delta_y < 0) begin

error <= error - delta_y + delta_x;

y <= y + ystep;

end

else begin

error <= error - delta_y;

end

if (x == x1) begin

state <= INITIAL;

connection_denoter <= connection_denoter +1;

if (connection_denoter == 4) begin

line_ready <=1;

write_enable <=0;

end

end
```

```
                                                            end//
for state DRAW_LINE_CALCULATE
                          LOGICAL_ASSIGN:begin

        if ((display_x_white != display_x_gray_one) &&

        (display_x_white != display_x_gray_two))

                final_display_x_white <= display_x_white;

        if ((display_y_white != display_y_gray_one) &&

        (display_y_white != display_y_gray_two))

                final_display_y_white <= display_y_white;

        if ((display_x_gray_one > 0) &&

        (display_x_gray_one != display_x_dark_gray_one))

                final_display_x_gray_one <=
display_x_gray_one;

        if ((display_y_gray_one > 0) &&

        (display_y_gray_one != display_y_dark_gray_one))

                final_display_y_gray_one <=
display_y_gray_one;

        if ((display_x_gray_two > 0) &&

        (display_x_gray_two != display_x_dark_gray_two))

                final_display_x_gray_two <=
display_x_gray_two;

        if ((display_y_gray_two > 0) &&

        (display_y_gray_two != display_y_dark_gray_two))

                final_display_y_gray_two <=
display_y_gray_two;

        if ((display_x_dark_gray_one > 0))

                final_display_x_dark_gray_one <=
display_x_dark_gray_one;
```

```
            if ((display_y_dark_gray_one > 0))

                        final_display_y_dark_gray_one <=
display_y_dark_gray_one;

            if ((display_x_dark_gray_two > 0))

                        final_display_x_dark_gray_two <=
display_x_dark_gray_two;

            if ((display_y_dark_gray_two > 0))

                        final_display_y_dark_gray_two <=
display_y_dark_gray_two;

            state <= MEMORY;

            end//for begin of LOGICAL ASSIGN
                                          MEMORY: begin

            write_enable <=1;

            if ((write_enable) && (memory_fill_denote==0))
begin

            write_x <= final_display_x_white;

            write_y <= final_display_y_white;

            colour_denote <= 2'b11;

            memory_fill_denote <= 1;

            end

            else if ((write_enable) && (memory_fill_denote==1))
begin

            write_x <= final_display_x_gray_one;

            write_y <= final_display_y_gray_one;

            colour_denote <= 2'b10;

            memory_fill_denote <= 2;

            end
```

```
          else if ((write_enable) && (memory_fill_denote==2))
begin
          write_x <= final_display_x_gray_two;
          write_y <= final_display_y_gray_two;
          colour_denote <= 2'b10;
          memory_fill_denote <= 3;
          end
          else if ((write_enable) && (memory_fill_denote==3))
begin
          write_x <= final_display_x_dark_gray_one;
          write_y <= final_display_y_dark_gray_one;
          colour_denote <= 2'b01;
          memory_fill_denote <= 4;
          end
          else if ((write_enable) && (memory_fill_denote==4))
begin
          write_x <= final_display_x_dark_gray_two;
          write_y <= final_display_y_dark_gray_two;
          colour_denote <= 2'b01;
          memory_fill_denote <= 5;
          end

          if (memory_fill_denote ==5) begin
          state <= DRAW_LINE_CALCULATE;
          memory_fill_denote <=0;
          end
```

```
                                                            end// for
                                MEMORY begin
                                                    default :state <= INITIAL;
                                               endcase//for case state
                                          end// for else begin
                                     end//for always vclock
                                endmodule
```

A.18 Points Decider Module

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:  MIT 6.111
// Engineer: Raghavendra Srinivasan
//
// Create Date:    19:24:06 11/07/2011
// Design Name:    Points Sorter or Points Decoder
// Module Name:    points_decider
// Project Name:   Motion Sensing and Graphic reproduction using artificaial intelligence
// Target Devices:
// Tool versions:
// Description: This Module takes six points from the video module, sorts them out into
//                                          hands,legs and torso points. The output of this
module can be directly
//                                          used in coordination with the Line Drawing module,
to get lines.
//                                          The cool thing about this module is that, it just has
a latency of 8.
// Dependencies: Video module.
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
                                module points_decider(
                                          input clk,
                                          input [10:0] x1,x2,x3,x4,x5,x6,
                                          input [9:0] y1,y2,y3,y4,y5,y6,
                                          input start_decide,
                                          //these are the six points tracked by the camera
                                   output reg [10:0] torso_up_x,arm_left_x,arm_right_x,

                                         leg_left_x,leg_right_x,torso_down_x,
                                          output reg [9:0] torso_up_y,arm_left_y,arm_right_y,
          leg_left_y,leg_right_y,torso_down_y,
           output reg calc_done
   //the outputs are basically the figured out respective
points
           );
parameter LEVELS_ONE = 0;
parameter LEVELS_TWO = 1;
parameter CENTER_GET_ONE=2;
parameter CENTER_GET_TWO =3;
parameter MIDDLE =4;
parameter MAP_MIDDLE_ONE =5;
parameter MAP_MIDDLE_TWO =6;
parameter FINAL_MAP = 7;
reg [10:0]
lowest_x,highest_x,lowest_y,highest_y,second_lowest_x,s
econd_lowest_y,second_highest_x,second_highest_y,

          dummy_store_y,center_dummy_one_x,center_dum
my_one_y,center_dummy_two_x,center_dummy_two_y;
reg [2:0]state;
//reg [2:0] next_state;
reg [2:0]
x_low_denote,x_high_denote,y_low_denote,y_high_denote
,new_y_low_denote;

initial begin
lowest_x = 0;
highest_x = 0;
lowest_y = 0;
highest_y = 0;
second_lowest_x = 0;
second_lowest_y = 0;
second_highest_x = 0;
second_highest_y = 0;
state = 0;
//next_state = 0;
x_low_denote = 0;
x_high_denote = 0;
y_low_denote = 0;
y_high_denote = 0;
calc_done = 0;
end
          always @(posedge clk) begin
          //state <= next_state;
                    case (state)
                    LEVELS_ONE:         begin //in this state
the highest and the lowest points are detected
```

```
                                        //checking for lowest x
value
                                        calc_done <=0;
                                        if(start_decide) begin
///gave this newly

                                                if((x1<x2) &&
(x1<x3) && (x1<x4) && (x1<x5) && (x1<x6)) begin

        lowest_x <= x1;

        x_low_denote <= 1;
                                                        end
                                                else if((x2<x1)
&& (x2<x3) && (x2<x4) && (x2<x5) && (x2<x6)) begin

        lowest_x <= x2;

        x_low_denote <= 2;
                                                        end
                                                else if((x3<x1)
&& (x3<x2) && (x3<x4) && (x3<x5) && (x3<x6)) begin

        lowest_x <= x3;

        x_low_denote <= 3;
                                                        end
                                                else if((x4<x1)
&& (x4<x2) && (x4<x3) && (x4<x5) && (x4<x6)) begin

        lowest_x <= x4;

        x_low_denote <= 4;
                                                        end
                                                else if((x5<x1)
&& (x5<x2) && (x5<x3) && (x5<x4) && (x5<x6)) begin

        lowest_x <= x5;

        x_low_denote <= 5;
                                                        end
                                                else begin

        lowest_x <= x6;

        x_low_denote <= 6;
                                                        end
```

```
                                                    //checking for
highest x value
                                                    if((x1>x2) &&
(x1>x3) && (x1>x4) && (x1>x5) && (x1>x6)) begin

        highest_x <= x1;

        x_high_denote <= 1;
                                                            end
                                                    else if((x2>x1)
&& (x2>x3) && (x2>x4) && (x2>x5) && (x2>x6)) begin

        highest_x <= x2;

        x_high_denote <= 2;
                                                            end
                                                    else if((x3>x1)
&& (x3>x2) && (x3>x4) && (x3>x5) && (x3>x6)) begin

        highest_x <= x3;

        x_high_denote <= 3;
                                                            end
                                                    else if((x4>x1)
&& (x4>x2) && (x4>x3) && (x4>x5) && (x4>x6)) begin

        highest_x <= x4;

        x_high_denote <= 4;
                                                            end
                                                    else if((x5>x1)
&& (x5>x2) && (x5>x3) && (x5>x4) && (x5>x6)) begin

        highest_x <= x5;

        x_high_denote <= 5;
                                                            end
                                                    else begin

        highest_x <= x6;

        x_high_denote <= 6;
                                                            end
                                                    //checking for
lowest y value
                                                    if((y1<y2) &&
(y1<y3) && (y1<y4) && (y1<y5) && (y1<y6)) begin
```

```
            lowest_y <= y1;

            y_low_denote <= 1;
                                                                        end
                                                            else if((y2<y1)
&& (y2<y3) && (y2<y4) && (y2<y5) && (y2<y6)) begin

            lowest_y <= y2;

            y_low_denote <= 2;
                                                                        end
                                                            else if((y3<y1)
&& (y3<y2) && (y3<y4) && (y3<y5) && (y3<y6)) begin

            lowest_y <= y3;

            y_low_denote <= 3;
                                                                        end
                                                            else if((y4<y1)
&& (y4<y2) && (y4<y3) && (y4<y5) && (y4<y6)) begin

            lowest_y <= y4;

            y_low_denote <= 4;
                                                                        end
                                                            else if((y5<y1)
&& (y5<y2) && (y5<y3) && (y5<y4) && (y5<y6)) begin

            lowest_y <= y5;

            y_low_denote <= 5;
                                                                        end
                                                            else begin

            lowest_y <= y6;

            y_low_denote <= 6;
                                                                        end
                                                            // checking for
highest value of y
                                                            if((y1>y2) &&
(y1>y3) && (y1>y4) && (y1>y5) && (y1>y6)) begin

            highest_y <= y1;

            y_high_denote<= 1;
                                                                        end
                                                            else if((y2>y1)
&& (y2>y3) && (y2>y4) && (y2>y5) && (y2>y6)) begin

          highest_y <= y2;

          y_high_denote <= 2;
                                                                      end
                                                            else if((y3>y1)
&& (y3>y2) && (y3>y4) && (y3>y5) && (y3>y6)) begin

          highest_y <= y3;

          y_high_denote <= 3;
                                                                      end
                                                            else if((y4>y1)
&& (y4>y2) && (y4>y3) && (y4>y5) && (y4>y6)) begin

          highest_y <= y4;

          y_high_denote <= 4;
                                                                      end
                                                            else if((y5>y1)
&& (y5>y2) && (y5>y3) && (y5>y4) && (y5>y6)) begin

          highest_y <= y5;

          y_high_denote <= 5;
                                                                      end
                                                            else begin

          highest_y <= y6;

          y_high_denote <= 6;
                                                                      end
                                                                      state
<= LEVELS_TWO;// go to level 2
                                                            end//for
start_decide

                                                  end // for begin of
LEVELS_ONE
                    LEVELS_TWO: begin// in this state the
second highest and second lowest points are detected
                                                            //figure out
second lowest x

          case(x_low_denote)
```

```
                                                    1 : if
((x2<x3) && (x2<x4) && (x2<x5) && (x2<x6))
second_lowest_x <= x2;

else if ((x3<x2) && (x3<x4) && (x3<x5) && (x3<x6))
second_lowest_x <= x3;

else if ((x4<x2) && (x4<x3) && (x4<x5) && (x4<x6))
second_lowest_x <= x4;

else if ((x5<x2) && (x5<x3) && (x5<x4) && (x4<x6))
second_lowest_x <= x5;

else second_lowest_x <= x6;
                                                    2 : if
((x1<x3) && (x1<x4) && (x1<x5) && (x1<x6))
second_lowest_x <= x1;

else if ((x3<x1) && (x3<x4) && (x3<x5) && (x3<x6))
second_lowest_x <= x3;

else if ((x4<x1) && (x4<x3) && (x4<x5) && (x4<x6))
second_lowest_x <= x4;

else if ((x5<x1) && (x5<x3) && (x5<x4) && (x4<x6))
second_lowest_x <= x5;

else second_lowest_x <= x6;
                                                    3 : if
((x2<x1) && (x2<x4) && (x2<x5) && (x2<x6))
second_lowest_x <= x2;

else if ((x1<x2) && (x1<x4) && (x1<x5) && (x1<x6))
second_lowest_x <= x1;

else if ((x4<x2) && (x4<x1) && (x4<x5) && (x4<x6))
second_lowest_x <= x4;

else if ((x5<x2) && (x5<x1) && (x5<x4) && (x4<x6))
second_lowest_x <= x5;

else second_lowest_x <= x6;
                                                    4 : if
((x2<x3) && (x2<x1) && (x2<x5) && (x2<x6))
second_lowest_x <= x2;

else if ((x3<x2) && (x3<x1) && (x3<x5) && (x3<x6))
second_lowest_x <= x3;
```

```
else if ((x1<x2) && (x1<x3) && (x1<x5) && (x1<x6))
second_lowest_x <= x1;

else if ((x5<x2) && (x5<x3) && (x5<x1) && (x1<x6))
second_lowest_x <= x5;

else second_lowest_x <= x6;
                                                        5 : if
((x2<x3) && (x2<x4) && (x2<x1) && (x2<x6))
second_lowest_x <= x2;

else if ((x3<x2) && (x3<x4) && (x3<x1) && (x3<x6))
second_lowest_x <= x3;

else if ((x4<x2) && (x4<x3) && (x4<x1) && (x4<x6))
second_lowest_x <= x4;

else if ((x1<x2) && (x1<x3) && (x1<x4) && (x4<x6))
second_lowest_x <= x1;

else second_lowest_x <= x6;
                                                        6 : if
((x2<x3) && (x2<x4) && (x2<x5) && (x2<x1))
second_lowest_x <= x2;

else if ((x3<x2) && (x3<x4) && (x3<x5) && (x3<x1))
second_lowest_x <= x3;

else if ((x4<x2) && (x4<x3) && (x4<x5) && (x4<x1))
second_lowest_x <= x4;

else if ((x5<x2) && (x5<x3) && (x5<x4) && (x4<x1))
second_lowest_x <= x5;

else second_lowest_x <= x1;

        endcase// for case of x_low_denote
                                                        //figure
out second lowest y

        case(y_low_denote)
                                                        1 : if
((y2<y3) && (y2<y4) && (y2<y5) && (y2<y6))
second_lowest_y <= y2;

else if ((y3<y2) && (y3<y4) && (y3<y5) && (y3<y6))
second_lowest_y <= y3;
```

```
else if ((y4<y2) && (y4<y3) && (y4<y5) && (y4<y6))
second_lowest_y <= y4;

else if ((y5<y2) && (y5<y3) && (y5<y4) && (y4<y6))
second_lowest_y <= y5;

else second_lowest_y <= y6;
                                                                2 : if
((y1<y3) && (y1<y4) && (y1<y5) && (y1<y6))
second_lowest_y <= y1;

else if ((y3<y1) && (y3<y4) && (y3<y5) && (y3<y6))
second_lowest_y <= y3;

else if ((y4<y1) && (y4<y3) && (y4<y5) && (y4<y6))
second_lowest_y <= y4;

else if ((y5<y1) && (y5<y3) && (y5<y4) && (y4<y6))
second_lowest_y <= y5;

else second_lowest_y <= y6;
                                                                3 : if
((y2<y1) && (y2<y4) && (y2<y5) && (y2<y6))
second_lowest_y <= y2;

else if ((y1<y2) && (y1<y4) && (y1<y5) && (y1<y6))
second_lowest_y <= y1;

else if ((y4<y2) && (y4<y1) && (y4<y5) && (y4<y6))
second_lowest_y <= y4;

else if ((y5<y2) && (y5<y1) && (y5<y4) && (y4<y6))
second_lowest_y <= y5;

else second_lowest_y <= y6;
                                                                4 : if
((y2<y3) && (y2<y1) && (y2<y5) && (y2<y6))
second_lowest_y <= y2;

else if ((y3<y2) && (y3<y1) && (y3<y5) && (y3<y6))
second_lowest_y <= y3;

else if ((y1<y2) && (y1<y3) && (y1<y5) && (y1<y6))
second_lowest_y <= y1;

else if ((y5<y2) && (y5<y3) && (y5<y1) && (y1<y6))
second_lowest_y <= y5;
```

```
else second_lowest_y <= y6;
                                                        5 : if
((y2<y3) && (y2<y4) && (y2<y1) && (y2<y6))
second_lowest_y <= y2;

else if ((y3<y2) && (y3<y4) && (y3<y1) && (y3<y6))
second_lowest_y <= y3;

else if ((y4<y2) && (y4<y3) && (y4<y1) && (y4<y6))
second_lowest_y <= y4;

else if ((y1<y2) && (y1<y3) && (y1<y4) && (y4<y6))
second_lowest_y <= y1;

else second_lowest_y <= y6;
                                                        6 : if
((y2<y3) && (y2<y4) && (y2<y5) && (y2<y1))
second_lowest_y <= y2;

else if ((y3<y2) && (y3<y4) && (y3<y5) && (y3<y1))
second_lowest_y <= y3;

else if ((y4<y2) && (y4<y3) && (y4<y5) && (y4<y1))
second_lowest_y <= y4;

else if ((y5<y2) && (y5<y3) && (y5<y4) && (y4<y1))
second_lowest_y <= y5;

else second_lowest_y <= y1;

        endcase// for case  of y_low_denote

                                                //figure out
second highest x

        case(x_high_denote)
                                                        1 : if
((x2>x3) && (x2>x4) && (x2>x5) && (x2>x6))
second_highest_x <= x2;

else if ((x3>x2) && (x3>x4) && (x3>x5) && (x3>x6))
second_highest_x <= x3;

else if ((x4>x2) && (x4>x3) && (x4>x5) && (x4>x6))
second_highest_x <= x4;
```

```
else if ((x5>x2) && (x5>x3) && (x5>x4) && (x4>x6))
second_highest_x <= x5;

else second_highest_x <= x6;
                                                        2 : if
((x1>x3) && (x1>x4) && (x1>x5) && (x1>x6))
second_highest_x <= x1;

else if ((x3>x1) && (x3>x4) && (x3>x5) && (x3>x6))
second_highest_x <= x3;

else if ((x4>x1) && (x4>x3) && (x4>x5) && (x4>x6))
second_highest_x <= x4;

else if ((x5>x1) && (x5>x3) && (x5>x4) && (x4>x6))
second_highest_x <= x5;

else second_highest_x <= x6;
                                                        3 : if
((x2>x1) && (x2>x4) && (x2>x5) && (x2>x6))
second_highest_x <= x2;

else if ((x1>x2) && (x1>x4) && (x1>x5) && (x1>x6))
second_highest_x <= x1;

else if ((x4>x2) && (x4>x1) && (x4>x5) && (x4>x6))
second_highest_x <= x4;

else if ((x5>x2) && (x5>x1) && (x5>x4) && (x4>x6))
second_highest_x <= x5;

else second_highest_x <= x6;
                                                        4 : if
((x2>x3) && (x2>x1) && (x2>x5) && (x2>x6))
second_highest_x <= x2;

else if ((x3>x2) && (x3>x1) && (x3>x5) && (x3>x6))
second_highest_x <= x3;

else if ((x1>x2) && (x1>x3) && (x1>x5) && (x1>x6))
second_highest_x <= x1;

else if ((x5>x2) && (x5>x3) && (x5>x1) && (x1>x6))
second_highest_x <= x5;

else second_highest_x <= x6;
```

```
                                                                5 : if
((x2>x3) && (x2>x4) && (x2>x1) && (x2>x6))
second_highest_x <= x2;

else if ((x3>x2) && (x3>x4) && (x3>x1) && (x3>x6))
second_highest_x <= x3;

else if ((x4>x2) && (x4>x3) && (x4>x1) && (x4>x6))
second_highest_x <= x4;

else if ((x1>x2) && (x1>x3) && (x1>x4) && (x4>x6))
second_highest_x <= x1;

else second_highest_x <= x6;
                                                                6 : if
((x2>x3) && (x2>x4) && (x2>x5) && (x2>x1))
second_highest_x <= x2;

else if ((x3>x2) && (x3>x4) && (x3>x5) && (x3>x1))
second_highest_x <= x3;

else if ((x4>x2) && (x4>x3) && (x4>x5) && (x4>x1))
second_highest_x <= x4;

else if ((x5>x2) && (x5>x3) && (x5>x4) && (x4>x1))
second_highest_x <= x5;

else second_highest_x <= x1;

        endcase// for case of x_high_denote

                                                                //figure
out second highest y

        case(y_high_denote)
                                                                1 : if
((y2>y3) && (y2>y4) && (y2>y5) && (y2>y6))
second_highest_y <= y2;

else if ((y3>y2) && (y3>y4) && (y3>y5) && (y3>y6))
second_highest_y <= y3;

else if ((y4>y2) && (y4>y3) && (y4>y5) && (y4>y6))
second_highest_y <= y4;

else if ((y5>y2) && (y5>y3) && (y5>y4) && (y4>y6))
second_highest_y <= y5;
```

```
else second_highest_y <= y6;
                                                                2 : if
((y1>y3) && (y1>y4) && (y1>y5) && (y1>y6))
second_highest_y <= y1;

else if ((y3>y1) && (y3>y4) && (y3>y5) && (y3>y6))
second_highest_y <= y3;

else if ((y4>y1) && (y4>y3) && (y4>y5) && (y4>y6))
second_highest_y <= y4;

else if ((y5>y1) && (y5>y3) && (y5>y4) && (y4>y6))
second_highest_y <= y5;

else second_highest_y <= y6;
                                                                3 : if
((y2>y1) && (y2>y4) && (y2>y5) && (y2>y6))
second_highest_y <= y2;

else if ((y1>y2) && (y1>y4) && (y1>y5) && (y1>y6))
second_highest_y <= y1;

else if ((y4>y2) && (y4>y1) && (y4>y5) && (y4>y6))
second_highest_y <= y4;

else if ((y5>y2) && (y5>y1) && (y5>y4) && (y4>y6))
second_highest_y <= y5;

else second_highest_y <= y6;
                                                                4 : if
((y2>y3) && (y2>y1) && (y2>y5) && (y2>y6))
second_highest_y <= y2;

else if ((y3>y2) && (y3>y1) && (y3>y5) && (y3>y6))
second_highest_y <= y3;

else if ((y1>y2) && (y1>y3) && (y1>y5) && (y1>y6))
second_highest_y <= y1;

else if ((y5>y2) && (y5>y3) && (y5>y1) && (y1>y6))
second_highest_y <= y5;

else second_highest_y <= y6;
                                                                5 : if
((y2>y3) && (y2>y4) && (y2>y1) && (y2>y6))
second_highest_y <= y2;
```

```
else if ((y3>y2) && (y3>y4) && (y3>y1) && (y3>y6))
second_highest_y <= y3;

else if ((y4>y2) && (y4>y3) && (y4>y1) && (y4>y6))
second_highest_y <= y4;

else if ((y1>y2) && (y1>y3) && (y1>y4) && (y4>y6))
second_highest_y <= y1;

else second_highest_y <= y6;
                                                6 : if
((y2>y3) && (y2>y4) && (y2>y5) && (y2>y1))
second_highest_y <= y2;

else if ((y3>y2) && (y3>y4) && (y3>y5) && (y3>y1))
second_highest_y <= y3;

else if ((y4>y2) && (y4>y3) && (y4>y5) && (y4>y1))
second_highest_y <= y4;

else if ((y5>y2) && (y5>y3) && (y5>y4) && (y4>y1))
second_highest_y <= y5;

else second_highest_y <= y1;

        endcase// for case of y_high_denote
                                                state
<= CENTER_GET_ONE;
                                                end//
for begin of LEVELS_TWO
                        CENTER_GET_ONE:begin// in this
state we store y of leftarm in a variable

        if(((x1== lowest_x) || (x1 == second_lowest_x)) &&
((y1 != highest_y) && (y1 != second_highest_y)))
dummy_store_y<=y1;

        else if(((x2== lowest_x) || (x2 == second_lowest_x))
&& ((y2 != highest_y) && (y2 != second_highest_y)))
dummy_store_y<=y2;

        else if(((x3== lowest_x) || (x3 == second_lowest_x))
&& ((y3 != highest_y) && (y3 != second_highest_y)))
dummy_store_y<=y3;

        else if(((x4== lowest_x) || (x4 == second_lowest_x))
&& ((y4 != highest_y) && (y4 != second_highest_y)))
dummy_store_y<=y4;

          else if(((x5== lowest_x) || (x5 == second_lowest_x))
&& ((y5 != highest_y) && (y5 != second_highest_y)))
dummy_store_y<=y5;

          else if(((x6== lowest_x) || (x6 == second_lowest_x))
&& ((y6 != highest_y) && (y6 != second_highest_y)))
dummy_store_y<=y6;

          state <= CENTER_GET_TWO;

          end// for CENTER_GET_ONE
                              CENTER_GET_TWO:begin//in this
state we update lowest and second lowest y values

          if(((x1== highest_x) || (x1 == second_highest_x))
&& ((y1 != highest_y) && (y1 != second_highest_y))) begin

                    if (y1> dummy_store_y) begin

                              lowest_y <= dummy_store_y;

                              second_lowest_y <= y1;

                    end

                    else begin

                              second_lowest_y <=
dummy_store_y;

                              lowest_y <= y1;

                    end

          end

          else if(((x2== highest_x) || (x2 ==
second_highest_x)) && ((y2 != highest_y) && (y2 !=
second_highest_y))) begin

                    if (y2> dummy_store_y) begin

                              lowest_y <= dummy_store_y;

                              second_lowest_y <= y2;
                    end

                    else begin

                              second_lowest_y <=
dummy_store_y;

                              lowest_y <= y2;

                    end

          end

          else if(((x3== highest_x) || (x3 ==
second_highest_x)) && ((y3 != highest_y) && (y3 !=
second_highest_y))) begin

                    if (y3> dummy_store_y) begin

                              lowest_y <= dummy_store_y;

                              second_lowest_y <= y3;

                    end

                    else begin

                              second_lowest_y <=
dummy_store_y;

                              lowest_y <= y3;

                    end

          end

          else if(((x4== highest_x) || (x4 ==
second_highest_x)) && ((y4 != highest_y) && (y4 !=
second_highest_y))) begin

                    if (y4> dummy_store_y) begin

                              lowest_y <= dummy_store_y;

                              second_lowest_y <= y4;

                    end
                    else begin

                              second_lowest_y <=
dummy_store_y;

                              lowest_y <= y4;

                    end

          end

          else if(((x5== highest_x) || (x5 ==
second_highest_x)) && ((y5 != highest_y) && (y5 !=
second_highest_y))) begin

                    if (y5> dummy_store_y) begin

                              lowest_y <= dummy_store_y;

                              second_lowest_y <= y5;

                    end

                    else begin

                              second_lowest_y <=
dummy_store_y;

                              lowest_y <= y5;

                    end

          end

          else if(((x6== highest_x) || (x6 ==
second_highest_x)) && ((y6 != highest_y) && (y6 !=
second_highest_y))) begin

                    if (y6> dummy_store_y) begin

                              lowest_y <= dummy_store_y;

                              second_lowest_y <= y6;

                    end

                    else begin
```

```
                              second_lowest_y <=
dummy_store_y;

                              lowest_y <= y6;

                    end

          end

          state <= MIDDLE;

          end//for CENTER_GET_TWO

                              MIDDLE: begin // in this state the
arms and legs are detected.
                                                            //
figuring out left leg
                                                            //point
1
                                                            if (!((x1
< highest_x) && (x1 < second_highest_x) && (x1 >
lowest_x) && (x1 > second_lowest_x))) begin

          if ((highest_y == y1) && (lowest_x == x1)) begin

          leg_left_x <=x1;

          leg_left_y <=y1;

          end

          else if ((second_highest_y == y1) &&
(second_lowest_x == x1)) begin

          leg_left_x <=x1;

          leg_left_y <=y1;

          end

          else if ((second_highest_y == y1) && (lowest_x ==
x1)) begin

          leg_left_x <=x1;

          leg_left_y <=y1;
```

```
        end

        else if ((highest_y == y1) && (second_lowest_x ==
x1)) begin

        leg_left_x <=x1;

        leg_left_y <=y1;

        end
                                                    end
                                                    //point
2
                                                    if (!((x2
< highest_x) && (x2 < second_highest_x) && (x2 >
lowest_x) && (x2 > second_lowest_x))) begin

        if ((highest_y == y2) && (lowest_x == x2)) begin

        leg_left_x <=x2;

        leg_left_y <=y2;

        end

        else if ((second_highest_y == y2) &&
(second_lowest_x == x2)) begin

        leg_left_x <=x2;

        leg_left_y <=y2;

        end

        else if ((second_highest_y == y2) && (lowest_x ==
x2)) begin

        leg_left_x <=x2;

        leg_left_y <=y2;

        end

        else if ((highest_y == y2) && (second_lowest_x ==
x2)) begin

        leg_left_x <=x2;
```

```
            leg_left_y <=y2;

            end
                                                                        end
                                                                        //point
3
                                                                        if (!((x3
< highest_x) && (x3 < second_highest_x) && (x3 >
lowest_x) && (x3 > second_lowest_x))) begin

            if ((highest_y == y3) && (lowest_x == x3)) begin

            leg_left_x <=x3;

            leg_left_y <=y3;

            end

            else if ((second_highest_y == y3) &&
(second_lowest_x == x3)) begin

            leg_left_x <=x3;

            leg_left_y <=y3;

            end

            else if ((second_highest_y == y3) && (lowest_x ==
x3)) begin

            leg_left_x <=x3;

            leg_left_y <=y3;

            end

            else if ((highest_y == y3) && (second_lowest_x ==
x3)) begin

            leg_left_x <=x3;

            leg_left_y <=y3;

            end
                                                                        end
                                                                        //point
4
```

```
                                                                  if (!((x4
< highest_x) && (x4 < second_highest_x) && (x4 >
lowest_x) && (x4 > second_lowest_x))) begin

          if ((highest_y == y4) && (lowest_x == x4)) begin

          leg_left_x <=x4;

          leg_left_y <=y4;

          end

          else if ((second_highest_y == y4) &&
(second_lowest_x == x4)) begin

          leg_left_x <=x4;

          leg_left_y <=y4;

          end

          else if ((second_highest_y == y4) && (lowest_x ==
x4)) begin

          leg_left_x <=x4;

          leg_left_y <=y4;

          end

          else if ((highest_y == y4) && (second_lowest_x ==
x4)) begin

          leg_left_x <=x4;

          leg_left_y <=y4;

          end
                                                                  end
                                                                  //point
5
                                                                  if (!((x5
< highest_x) && (x5 < second_highest_x) && (x5 >
lowest_x) && (x5 > second_lowest_x))) begin

          if ((highest_y == y5) && (lowest_x == x5)) begin

          leg_left_x <=x5;
```

```
            leg_left_y <=y5;

            end

            else if ((second_highest_y == y5) &&
(second_lowest_x == x5)) begin

            leg_left_x <=x5;

            leg_left_y <=y5;

            end

            else if ((second_highest_y == y5) && (lowest_x ==
x5)) begin

            leg_left_x <=x5;

            leg_left_y <=y5;

            end

            else if ((highest_y == y5) && (second_lowest_x ==
x5)) begin

            leg_left_x <=x5;

            leg_left_y <=y5;

            end
                                                  end
                                                  //point
6
                                                  if (!((x6
< highest_x) && (x6 < second_highest_x) && (x6 >
lowest_x) && (x6 > second_lowest_x))) begin

            if ((highest_y == y6) && (lowest_x == x6)) begin

            leg_left_x <=x6;

            leg_left_y <=y6;

            end

            else if ((second_highest_y == y6) &&
(second_lowest_x == x6)) begin
```

```
            leg_left_x <=x6;

            leg_left_y <=y6;

            end

            else if ((second_highest_y == y6) && (lowest_x ==
x6)) begin

            leg_left_x <=x6;

            leg_left_y <=y6;

            end

            else if ((highest_y == y6) && (second_lowest_x ==
x6)) begin

            leg_left_x <=x6;

            leg_left_y <=y6;

            end
                                                                        end


            //figuring out right leg
                                                                        //point
1
                                                                        if (!((x1
< highest_x) && (x1 < second_highest_x) && (x1 >
lowest_x) && (x1 > second_lowest_x))) begin

            if ((highest_y == y1) && (highest_x == x1)) begin

            leg_right_x <= x1;

            leg_right_y <= y1;

            end

            else if ((second_highest_y == y1) &&
(second_highest_x == x1)) begin

            leg_right_x <= x1;

            leg_right_y <= y1;
```

```
          end

          else if ((highest_y == y1) && (second_highest_x ==
x1)) begin

          leg_right_x <= x1;

          leg_right_y <= y1;

          end

          else if ((second_highest_y == y1) && (highest_x ==
x1)) begin

          leg_right_x <= x1;

          leg_right_y <= y1;

          end
                                                            end
                                                            //point
2
                                                            if (!((x2
< highest_x) && (x2 < second_highest_x) && (x2 >
lowest_x) && (x2 > second_lowest_x))) begin

          if ((highest_y == y2) && (highest_x == x2)) begin

          leg_right_x <= x2;

          leg_right_y <= y2;

          end

          else if ((second_highest_y == y2) &&
(second_highest_x == x2)) begin

          leg_right_x <= x2;

          leg_right_y <= y2;

          end

          else if ((highest_y == y2) && (second_highest_x ==
x2)) begin

          leg_right_x <= x2;
```

```
            leg_right_y <= y2;

            end

            else if ((second_highest_y == y2) && (highest_x ==
x2)) begin

            leg_right_x <= x2;

            leg_right_y <= y2;

            end
                                                                            end
                                                                            //point
3
                                                                            if (!((x3
< highest_x) && (x3 < second_highest_x) && (x3 >
lowest_x) && (x3 > second_lowest_x))) begin

            if ((highest_y == y3) && (highest_x == x3)) begin

            leg_right_x <= x3;

            leg_right_y <= y3;

            end

            else if ((second_highest_y == y3) &&
(second_highest_x == x3)) begin

            leg_right_x <= x3;

            leg_right_y <= y3;

            end

            else if ((highest_y == y3) && (second_highest_x ==
x3)) begin

            leg_right_x <= x3;

            leg_right_y <= y3;

            end

            else if ((second_highest_y == y3) && (highest_x ==
x3)) begin
```

```
          leg_right_x <= x3;

          leg_right_y <= y3;

          end
                                                            end
                                                            //point
4
                                                            if (!((x4
< highest_x) && (x4 < second_highest_x) && (x4 >
lowest_x) && (x4 > second_lowest_x))) begin

          if ((highest_y == y4) && (highest_x == x4)) begin

          leg_right_x <= x4;

          leg_right_y <= y4;

          end

          else if ((second_highest_y == y4) &&
(second_highest_x == x4)) begin

          leg_right_x <= x4;

          leg_right_y <= y4;

          end

          else if ((highest_y == y4) && (second_highest_x ==
x4)) begin

          leg_right_x <= x4;

          leg_right_y <= y4;

          end

          else if ((second_highest_y == y4) && (highest_x ==
x4)) begin

          leg_right_x <= x4;

          leg_right_y <= y4;

          end
                                                            end
                                                            //point
5
                                                            if (!((x5
< highest_x) && (x5 < second_highest_x) && (x5 >
lowest_x) && (x5 > second_lowest_x))) begin

        if ((highest_y == y5) && (highest_x == x5)) begin

        leg_right_x <= x5;

        leg_right_y <= y5;

        end

        else if ((second_highest_y == y5) &&
(second_highest_x == x5)) begin

        leg_right_x <= x5;

        leg_right_y <= y5;

        end

        else if ((highest_y == y5) && (second_highest_x ==
x5)) begin

        leg_right_x <= x5;

        leg_right_y <= y5;

        end

        else if ((second_highest_y == y5) && (highest_x ==
x5)) begin

        leg_right_x <= x5;

        leg_right_y <= y5;

        end
                                                            end
                                                            //point
6
                                                            if (!((x6
< highest_x) && (x6 < second_highest_x) && (x6 >
lowest_x) && (x6 > second_lowest_x))) begin

        if ((highest_y == y6) && (highest_x == x6)) begin
```

```
            leg_right_x <= x6;

            leg_right_y <= y6;

            end

            else if ((second_highest_y == y6) &&
(second_highest_x == x6)) begin

            leg_right_x <= x6;

            leg_right_y <= y6;

            end

            else if ((highest_y == y6) && (second_highest_x ==
x6)) begin

            leg_right_x <= x6;

            leg_right_y <= y6;

            end

            else if ((second_highest_y == y6) && (highest_x ==
x6)) begin

            leg_right_x <= x6;

            leg_right_y <= y6;

            end
                                                                      end


            //figuring out left hand
                                                                      //point
1
                                                                      if (!((x1
< highest_x) && (x1 < second_highest_x) && (x1 >
lowest_x) && (x1 > second_lowest_x))) begin

            if((lowest_x == x1) && (lowest_y == y1)) begin

            arm_left_x <= x1;

            arm_left_y <= y1;
            end

            else if((second_lowest_x == x1) && (lowest_y ==
y1)) begin

            arm_left_x <= x1;

            arm_left_y <= y1;

            end

            else if((second_lowest_x == x1) &&
(second_lowest_y == y1)) begin

            arm_left_x <= x1;

            arm_left_y <= y1;

            end

            else if((lowest_x == x1) && (second_lowest_y ==
y1)) begin

            arm_left_x <= x1;

            arm_left_y <= y1;

            end
                                                                    end
                                                                    //point
2
                                                                    if (!((x2
< highest_x) && (x2 < second_highest_x) && (x2 >
lowest_x) && (x2 > second_lowest_x))) begin

            if((lowest_x == x2) && (lowest_y == y2)) begin

            arm_left_x <= x2;

            arm_left_y <= y2;

            end

            else if((second_lowest_x == x2) && (lowest_y ==
y2)) begin

            arm_left_x <= x2;
            arm_left_y <= y2;

            end

            else if((second_lowest_x == x2) &&
(second_lowest_y == y2)) begin

            arm_left_x <= x2;

            arm_left_y <= y2;

            end

            else if((lowest_x == x2) && (second_lowest_y ==
y2)) begin

            arm_left_x <= x2;

            arm_left_y <= y2;

            end
                                                            end
                                                            //point
3
                                                            if (!((x3
< highest_x) && (x3 < second_highest_x) && (x3 >
lowest_x) && (x3 > second_lowest_x))) begin

            if((lowest_x == x3) && (lowest_y == y3)) begin

            arm_left_x <= x3;

            arm_left_y <= y3;

            end

            else if((second_lowest_x == x3) && (lowest_y ==
y3)) begin

            arm_left_x <= x3;

            arm_left_y <= y3;

            end

            else if((second_lowest_x == x3) &&
(second_lowest_y == y3)) begin
```

```
          arm_left_x <= x3;

          arm_left_y <= y3;

          end

          else if((lowest_x == x3) && (second_lowest_y ==
y3)) begin

          arm_left_x <= x3;

          arm_left_y <= y3;

          end
                                                            end
                                                            //point
4
                                                            if (!((x4
< highest_x) && (x4 < second_highest_x) && (x4 >
lowest_x) && (x4 > second_lowest_x))) begin

          if((lowest_x == x4) && (lowest_y == y4)) begin

          arm_left_x <= x4;

          arm_left_y <= y4;

          end

          else if((second_lowest_x == x4) && (lowest_y ==
y4)) begin

          arm_left_x <= x4;

          arm_left_y <= y4;

          end

          else if((second_lowest_x == x4) &&
(second_lowest_y == y4)) begin

          arm_left_x <= x4;

          arm_left_y <= y4;

          end
```

```
            else if((lowest_x == x4) && (second_lowest_y ==
y4)) begin

            arm_left_x <= x4;

            arm_left_y <= y4;

            end
                                                        end
                                                        //point
5
                                                        if (!((x5
< highest_x) && (x5 < second_highest_x) && (x5 >
lowest_x) && (x5 > second_lowest_x))) begin

            if((lowest_x == x5) && (lowest_y == y5)) begin

            arm_left_x <= x5;

            arm_left_y <= y5;

            end

            else if((second_lowest_x == x5) && (lowest_y ==
y5)) begin

            arm_left_x <= x5;

            arm_left_y <= y5;

            end

            else if((second_lowest_x == x5) &&
(second_lowest_y == y5)) begin

            arm_left_x <= x5;

            arm_left_y <= y5;

            end

            else if((lowest_x == x5) && (second_lowest_y ==
y5)) begin

            arm_left_x <= x5;

            arm_left_y <= y5;
```

```
          end
                                                            end
                                                            //point
6
                                                            if (!((x6
< highest_x) && (x6 < second_highest_x) && (x6 >
lowest_x) && (x6 > second_lowest_x))) begin

          if((lowest_x == x6) && (lowest_y == y6)) begin

          arm_left_x <= x6;

          arm_left_y <= y6;

          end

          else if((second_lowest_x == x6) && (lowest_y ==
y6)) begin

          arm_left_x <= x6;

          arm_left_y <= y6;

          end

          else if((second_lowest_x == x6) &&
(second_lowest_y == y6)) begin

          arm_left_x <= x6;

          arm_left_y <= y6;

          end

          else if((lowest_x == x6) && (second_lowest_y ==
y6)) begin

          arm_left_x <= x6;

          arm_left_y <= y6;

          end
                                                            end


          //figuring out right hand
```

```
                                                                    //point
1
                                                                    if (!((x1
< highest_x) && (x1 < second_highest_x) && (x1 >
lowest_x) && (x1 > second_lowest_x))) begin

        if((highest_x == x1) && (lowest_y == y1)) begin

        arm_right_x <= x1;

        arm_right_y <= y1;

        end

        else if((second_highest_x == x1) && (lowest_y ==
y1)) begin

        arm_right_x <= x1;

        arm_right_y <= y1;

        end

        else if((second_highest_x == x1) &&
(second_lowest_y == y1)) begin

        arm_right_x <= x1;

        arm_right_y <= y1;

        end

        else if((highest_x == x1) && (second_lowest_y ==
y1)) begin

        arm_right_x <= x1;

        arm_right_y <= y1;

        end
                                                                    end
                                                                    //point
2
                                                                    if (!((x2
< highest_x) && (x2 < second_highest_x) && (x2 >
lowest_x) && (x2 > second_lowest_x))) begin

        if((highest_x == x2) && (lowest_y == y2)) begin
        arm_right_x <= x2;

        arm_right_y <= y2;

        end

        else if((second_highest_x == x2) && (lowest_y ==
y2)) begin

        arm_right_x <= x2;

        arm_right_y <= y2;

        end

        else if((second_highest_x == x2) &&
(second_lowest_y == y2)) begin

        arm_right_x <= x2;

        arm_right_y <= y2;

        end

        else if((highest_x == x2) && (second_lowest_y ==
y2)) begin

        arm_right_x <= x2;

        arm_right_y <= y2;

        end
                                                    end
                                                    //point
3
                                                    if (!((x3
< highest_x) && (x3 < second_highest_x) && (x3 >
lowest_x) && (x3 > second_lowest_x))) begin

        if((highest_x == x3) && (lowest_y == y3)) begin

        arm_right_x <= x3;

        arm_right_y <= y3;

        end
        else if((second_highest_x == x3) && (lowest_y ==
y3)) begin

        arm_right_x <= x3;

        arm_right_y <= y3;

        end

        else if((second_highest_x == x3) &&
(second_lowest_y == y3)) begin

        arm_right_x <= x3;

        arm_right_y <= y3;

        end

        else if((highest_x == x3) && (second_lowest_y ==
y3)) begin

        arm_right_x <= x3;

        arm_right_y <= y3;

        end
                                                    end
                                                    //point
4
                                                    if (!((x4
< highest_x) && (x4 < second_highest_x) && (x4 >
lowest_x) && (x4 > second_lowest_x))) begin

        if((highest_x == x4) && (lowest_y == y4)) begin

        arm_right_x <= x4;

        arm_right_y <= y4;

        end

        else if((second_highest_x == x4) && (lowest_y ==
y4)) begin

        arm_right_x <= x4;

        arm_right_y <= y4;
```

```
            end

            else if((second_highest_x == x4) &&
(second_lowest_y == y4)) begin

            arm_right_x <= x4;

            arm_right_y <= y4;

            end

            else if((highest_x == x4) && (second_lowest_y ==
y4)) begin

            arm_right_x <= x4;

            arm_right_y <= y4;

            end
                                                                    end
                                                                    //point
5
                                                                    if (!((x5
< highest_x) && (x5 < second_highest_x) && (x5 >
lowest_x) && (x5 > second_lowest_x))) begin

            if((highest_x == x5) && (lowest_y == y5)) begin

            arm_right_x <= x5;

            arm_right_y <= y5;

            end

            else if((second_highest_x == x5) && (lowest_y ==
y5)) begin

            arm_right_x <= x5;

            arm_right_y <= y5;

            end

            else if((second_highest_x == x5) &&
(second_lowest_y == y5)) begin

            arm_right_x <= x5;
```

```
          arm_right_y <= y5;

          end

          else if((highest_x == x5) && (second_lowest_y ==
y5)) begin

          arm_right_x <= x5;

          arm_right_y <= y5;

          end
                                                          end
                                                          //point
6
                                                          if (!((x6
< highest_x) && (x6 < second_highest_x) && (x6 >
lowest_x) && (x6 > second_lowest_x))) begin

          if((highest_x == x6) && (lowest_y == y6)) begin

          arm_right_x <= x6;

          arm_right_y <= y6;

          end

          else if((second_highest_x == x6) && (lowest_y ==
y6)) begin

          arm_right_x <= x6;

          arm_right_y <= y6;

          end

          else if((second_highest_x == x6) &&
(second_lowest_y == y6)) begin

          arm_right_x <= x6;

          arm_right_y <= y6;

          end

          else if((highest_x == x6) && (second_lowest_y ==
y6)) begin
```

```
        arm_right_x <= x6;

        arm_right_y <= y6;

        end
                                                  end
                                          state
<=MAP_MIDDLE_ONE;
                                          end//for begin
of MIDDLE
                        MAP_MIDDLE_ONE:begin

        if ((x1 != highest_x) && (x1 != second_highest_x)
&& (x1 != lowest_x) && (x1 != second_lowest_x)) begin

                center_dummy_one_x <= x1;

                center_dummy_one_y <= y1;

                state <=MAP_MIDDLE_TWO;

        end

        else if ((x2 != highest_x) && (x2 !=
second_highest_x) && (x2 != lowest_x) && (x2 !=
second_lowest_x)) begin

                center_dummy_one_x <= x2;

                center_dummy_one_y <= y2;

                state <=MAP_MIDDLE_TWO;

        end

        else if ((x3 != highest_x) && (x3 !=
second_highest_x) && (x3 != lowest_x) && (x3 !=
second_lowest_x)) begin

                center_dummy_one_x <= x3;

                center_dummy_one_y <= y3;

                state <=MAP_MIDDLE_TWO;

        end
        else if ((x4 != highest_x) && (x4 !=
second_highest_x) && (x4 != lowest_x) && (x4 !=
second_lowest_x)) begin

                center_dummy_one_x <= x4;

                center_dummy_one_y <= y4;

                state <=MAP_MIDDLE_TWO;

        end

        else if ((x5 != highest_x) && (x5 !=
second_highest_x) && (x5 != lowest_x) && (x5 !=
second_lowest_x)) begin

                center_dummy_one_x <= x5;

                center_dummy_one_y <= y5;

                state <=MAP_MIDDLE_TWO;

        end

        else if ((x6 != highest_x) && (x6 !=
second_highest_x) && (x6 != lowest_x) && (x6 !=
second_lowest_x)) begin

                center_dummy_one_x <= x6;

                center_dummy_one_y <= y6;

                state <=MAP_MIDDLE_TWO;

        end

        end //for MAP_MIDDLE_ONE
                        MAP_MIDDLE_TWO:begin

        if ((x1 != highest_x) && (x1 != second_highest_x)
&& (x1 != lowest_x) && (x1 != second_lowest_x) && (x1 !=
center_dummy_one_x)) begin

                center_dummy_two_x <= x1;

                center_dummy_two_y <= y1;
```

```
          end

          else if ((x2 != highest_x) && (x2 !=
second_highest_x) && (x2 != lowest_x) && (x2 !=
second_lowest_x) && (x2 != center_dummy_one_x)) begin

                    center_dummy_two_x <= x2;

                    center_dummy_two_y <= y2;

          end

          else if ((x3 != highest_x) && (x3 !=
second_highest_x) && (x3 != lowest_x) && (x3 !=
second_lowest_x) && (x3 != center_dummy_one_x)) begin

                    center_dummy_two_x <= x3;

                    center_dummy_two_y <= y3;

          end

          else if ((x4 != highest_x) && (x4 !=
second_highest_x) && (x4 != lowest_x) && (x4 !=
second_lowest_x) && (x4 != center_dummy_one_x)) begin

                    center_dummy_two_x <= x4;

                    center_dummy_two_y <= y4;

          end

          else if ((x5 != highest_x) && (x5 !=
second_highest_x) && (x5 != lowest_x) && (x5 !=
second_lowest_x) && (x5 != center_dummy_one_x)) begin

                    center_dummy_two_x <= x5;

                    center_dummy_two_y <= y5;

          end

          else if ((x6 != highest_x) && (x6 !=
second_highest_x) && (x6 != lowest_x) && (x6 !=
second_lowest_x) && (x6 != center_dummy_one_x)) begin

                    center_dummy_two_x <= x6;
                    center_dummy_two_y <= y6;

          end

          state <= FINAL_MAP;

          end//for MAP_MIDDLE_TWO
                              FINAL_MAP:begin
                                                                      if
(center_dummy_two_y > center_dummy_one_y) begin

          torso_up_x <= center_dummy_one_x;

          torso_up_y <= center_dummy_one_y;

          torso_down_x <= center_dummy_two_x;

          torso_down_y <= center_dummy_two_y;
                                                                      end
                                                                      else
begin

          torso_up_x <= center_dummy_two_x;

          torso_up_y <= center_dummy_two_y;

          torso_down_x <= center_dummy_one_x;

          torso_down_y <= center_dummy_one_y;
                                                                      end

          calc_done <= 1;
                                                                      state
<= LEVELS_ONE;
                                                             end//for
FINAL_MAP
                    endcase// for case of state
          end// for clock begin
endmodule
```

A.19 Integeration Module

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
```

```
// Create Date:    11:11:42 12/12/2011
// Design Name:
// Module Name:    pupuma
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////
module pupuma(clk,reset,x1, y1, x2, y2, x3, y3, x4, y4, x5, y5, x6, y6, xx1, yy1, xx2, yy2,
xx3, yy3, xx4, yy4, xx5, yy5, xx6, yy6);

        input  clk;
        //system clock

        input reset;


        //markers x and y coordinates, lower 10 bits are for y, and upper 11 are for x

        input [10:0] x1,x2,x3,x4,x5,x6;
        input [9:0] y1,y2,y3,y4,y5,y6;

        output reg [10:0] xx1,xx2,xx3,xx4,xx5,xx6;
        output reg [9:0] yy1,yy2,yy3,yy4,yy5,yy6;

        reg [10:0] x01,x02,x03,x04,x05,x06;
        reg [9:0] y01,y02,y03,y04,y05,y06;

        reg [7:0] pipe1;
        reg [1:0] state;
        reg [4:0] counter;

        reg [10:0] x001,x002,x003,x004,x005,x006;
        reg [9:0] y001,y002,y003,y004,y005,y006;

        parameter INITIAL = 0;
        parameter ONE = 1;
        parameter TWO = 2;
        parameter DUMMY =3;

always @(posedge clk) begin
```

```
pipe1[0] <= ((x001 != x002) && (x001 != x003) && (x001 != x004) && (x001 != x005) &&
(x001 != x006)) ? 1 : 0;
pipe1[1] <= ((y001 != y002) && (y001 != y003) && (y001 != y004) && (y001 != y005) &&
(y001 != y006)) ? 1 : 0;
pipe1[2] <= ((x002 != x003) && (x002 != x004) && (x002 != x005) && (x002 != x006)) ? 1
: 0;
pipe1[3] <= ((y002 != y003) && (y002 != y004) && (y002 != y005) && (y002 != y006)) ? 1
: 0;
pipe1[4] <= ((x003 != x004) && (x003 != x005) && (x003 != x006)) ? 1 : 0;
pipe1[5] <= ((y003 != y004) && (y003 != y005) && (y003 != y006)) ? 1 : 0;
pipe1[6] <= ((x004 != x005) && (x004 != x006) && (y004 != y005) && (y004 != y006)) ? 1
: 0;
pipe1[7] <= ((x005 != x006) && (y005 != y006)) ? 1 : 0;
counter <= (state == DUMMY) ? counter + 1 : counter;

if (reset) begin
        x01 <= 0;
        x02 <= 0;
        x03 <= 0;
        x04 <= 0;
        x05 <= 0;
        x06 <= 0;
        y01 <= 0;
        y02 <= 0;
        y03 <= 0;
        y04 <= 0;
        y05 <= 0;
        y06 <= 0;
        x001 <= 0;
        x002 <= 0;
        x003 <= 0;
        x004 <= 0;
        x005 <= 0;
        x006 <= 0;
        y001 <= 0;
        y002 <= 0;
        y003 <= 0;
        y004 <= 0;
        y005 <= 0;
        y006 <= 0;
        state <= INITIAL;
        counter <= 0;
end

else begin

        case(state)
```

```
		INITIAL : begin
			counter <= 0;
			x01 <= x1;
			x02 <= x2;
			x03 <= x3;
			x04 <= x4;
			x05 <= x5;
			x06 <= x6;
			y01 <= y1;
			y02 <= y2;
			y03 <= y3;
			y04 <= y4;
			y05 <= y5;
			y06 <= y6;
			state <=  ONE;
			end
		ONE: begin

			if ((x01 > 20) &&(x02 > 20) &&(x03 > 20) &&(x04 > 20) &&(x05 >
20) &&(x06 > 20) &&
				(y01 > 20) &&(y02 > 20) &&(y03 > 20) &&(y04 > 20)
&&(y05 > 20) &&(y06 > 20)) begin

			x001 <= x01;
			x002 <= x02;
			x003 <= x03;
			x004 <= x04;
			x005 <= x05;
			x006 <= x06;
			y001 <= y01;
			y002 <= y02;
			y003 <= y03;
			y004 <= y04;
			y005 <= y05;
			y006 <= y06;
			end
			else begin
			x001 <= x001;
			x002 <= x002;
			x003 <= x003;
			x004 <= x004;
			x005 <= x005;
			x006 <= x006;
			y001 <= y001;
			y002 <= y002;
			y003 <= y003;
			y004 <= y004;
			y005 <= y005;
```

```
                        y006 <= y006;
                        end
                        state <= TWO;
                 end

            TWO : begin
                  if (pipe == 8'b11111111) begin
                        xx1 <= x001;
                        xx2 <= x002;
                        xx3 <= x003;
                        xx4 <= x004;
                        xx5 <= x005;
                        xx6 <= x006;
                        yy1 <= y001;
                        yy2 <= y002;
                        yy3 <= y003;
                        yy4 <= y004;
                        yy5 <= y005;
                        yy6 <= y006;
                  end
                  else begin
                        xx1 <= xx1;
                        xx2 <= xx2;
                        xx3 <= xx3;
                        xx4 <= xx4;
                        xx5 <= xx5;
                        xx6 <= xx6;
                        yy1 <= yy1;
                        yy2 <= yy2;
                        yy3 <= yy3;
                        yy4 <= yy4;
                        yy5 <= yy5;
                        yy6 <= yy6;
                  end
                  state <= (counter > 12) ? INITIAL : DUMMY;
             end

            DUMMY: state <= TWO;


            endcase
            end//for else begin
      end //for always
      endmodule
```

<u>A.20 Pulse Generator Module</u>

```
module pulse(input go,
                                        input clk,
                                        output wire pulse);
reg [1:0] k =3;
reg [1:0] z =0;
always@(posedge clk) begin
if(go) z <= (z<3) ? z+1:(z==3) ? 3: z;
end
assign pulse = (!go) ? 0: (k-z > 0);
endmodule
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

PICTURES

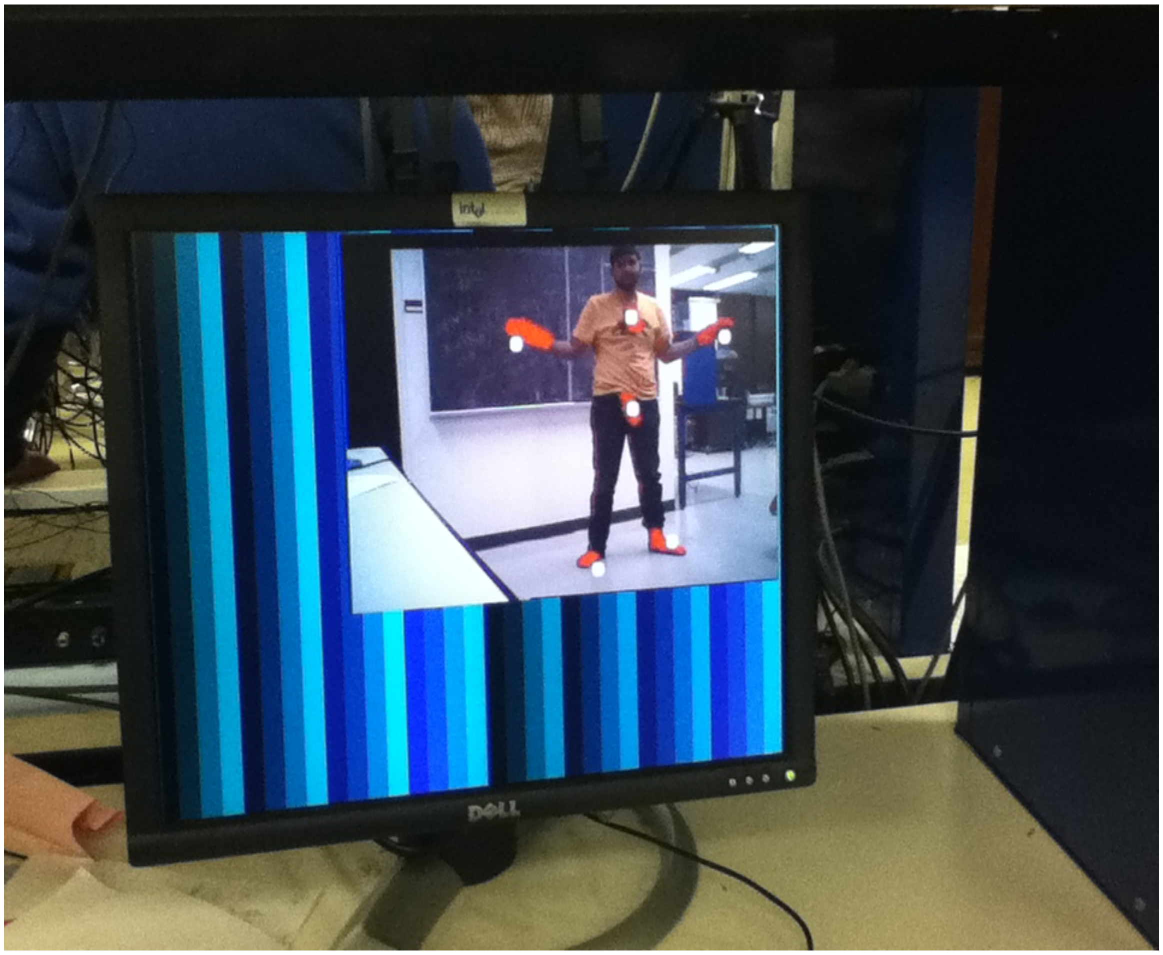


Figure 4: live action detected (detected red markers appear with white dot on them white)

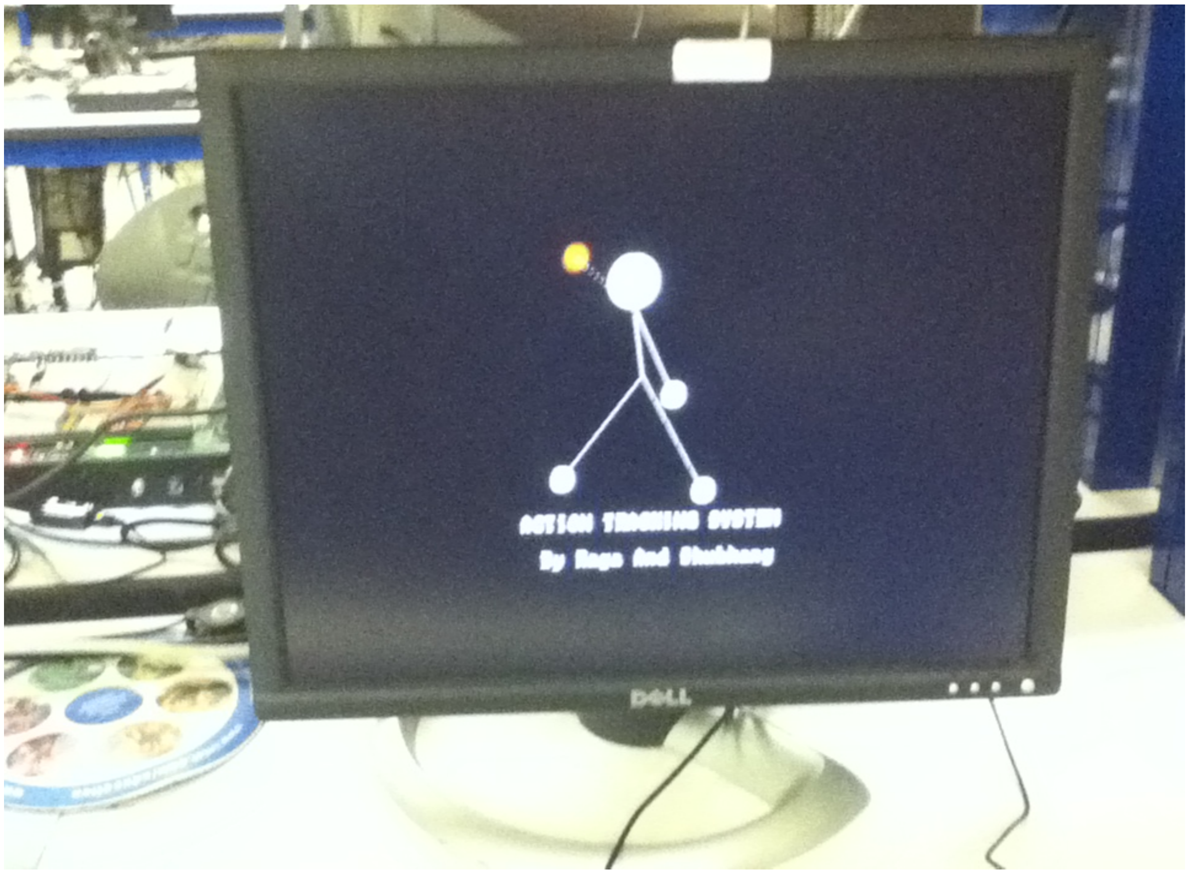


Figure 5: Stick figure generated on the screen.

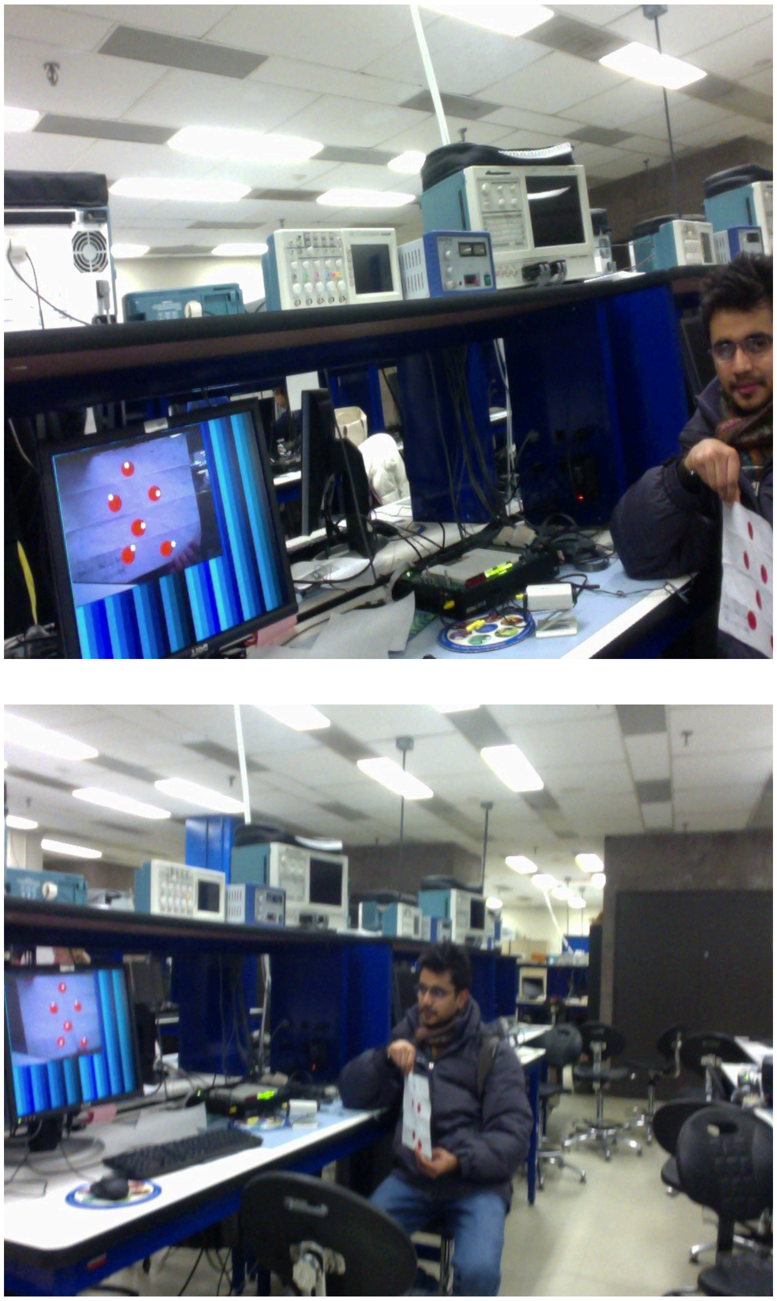


Figure 6: example for live point detection (detected points appear white)