Virtual Ping Pong

Zachary Clifford and Mark Stevens

December 15, 2007

1 Overview

The Virtual Ping Pong game uses a Xilinx Virtex-II FPGA chip to implement a virtual ping pong game with a video camera and a physical paddle. The user stands in front of the system camera and holds a bright red ping pong paddle. The screen shows a fake three dimensional playfield with a ball bouncing in it. The user then hits the ball back into the screen to play against himself or another player. In its multiplayer configuration the project utilizes two labkits to allow two players to hit the ball back and forth.

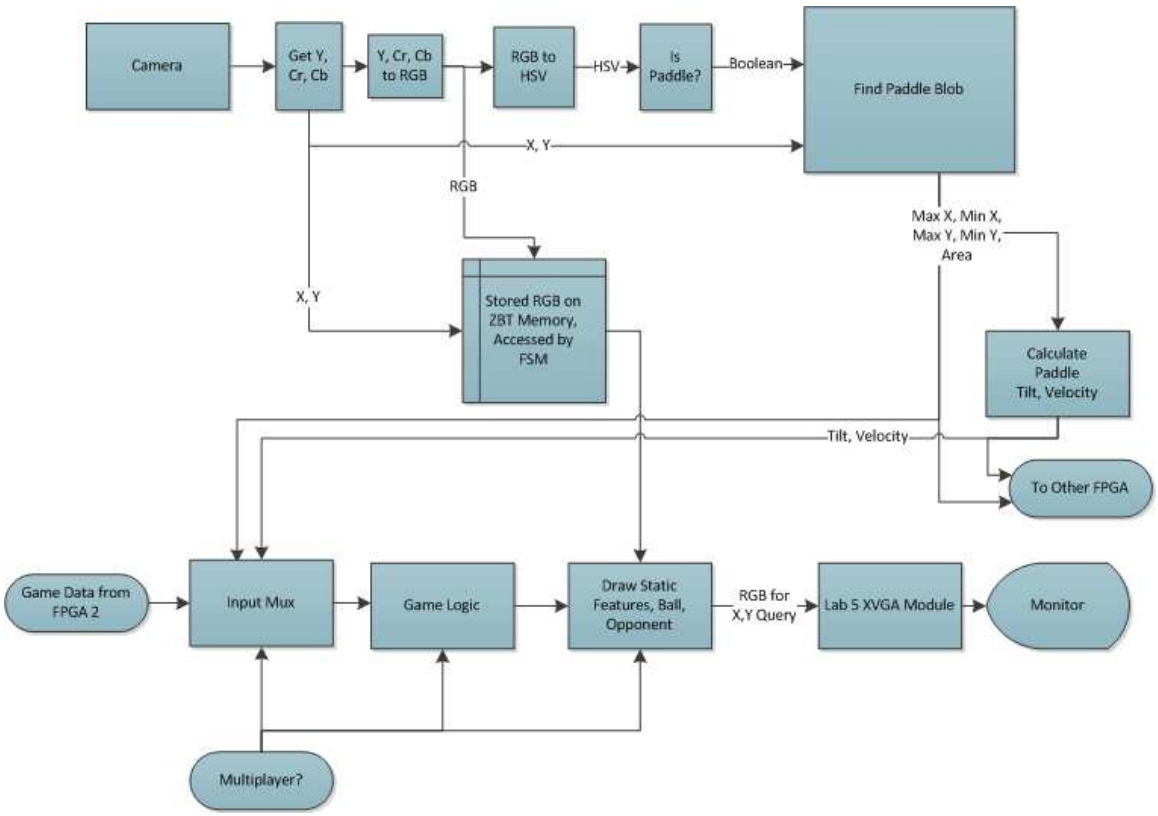

Figure 1: Project Overview

2 Vision Overview (M. Stevens)

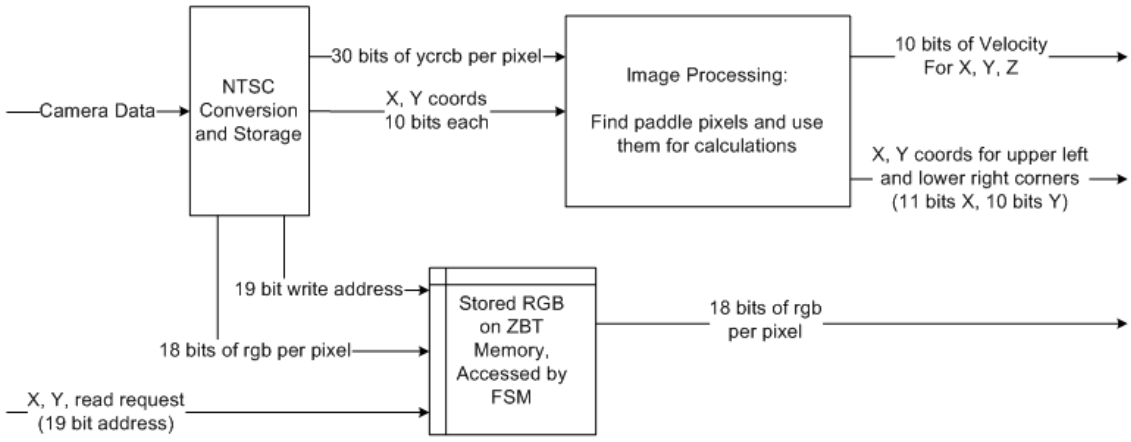

Figure 2: Vision System

Each labkit in the game uses a camera to capture the image of a player and his paddle. In multiplayer mode, this is the opponent's paddle, while in single player mode, it is the user's paddle. All such distinctions are handled by the multiplayer module; the vision modules process the data the same in either case. Data from the camera arrives in serial YCrCb at roughly 27MHz. In order to play the game, the following processing must be done with the incoming data stream:

- Conversion of serial stream into 30 bits of YCrCb data
- Conversion from YCrCb to RGB
- Storage of RGB pixels into a ZBT memory
- Reading RBG pixels back out of the ZBT to be displayed on screen
- Detection of red pixels (in YCrCb) that denote the paddle
- Filtering of red pixels to remove noise
- Calculation of edges and corners of the paddle
- Calculation of the X, Y, and Z velocity of the paddle
- Calculation of paddle tilt
- Coordinate conversion of paddle edges and velocities to be sent to game logic
- Optional display of debugging mode

To achieve this functionality, three different groups of modules are used: preprocessing and storage, image procesing, and debugging.

3 Vision - Preprocessing and Storage

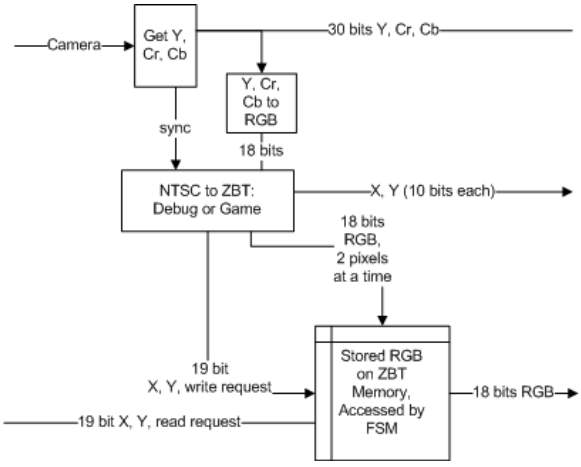

Figure 3: Vision Preprocessing and Storage

3.1 Data Preprocessing

The first two modules that preprocess the data are adv7185init and ntsc_decode. These modules were taken directly from previous years. adv7185init initializes the camera; ntsc_decode translates NTSC serial data into much more useful YCrCb data. It also outputs when the vertical and horizontal syncs happen on the camera.

Next, YCrCb data is converted into RGB data. Again, the module to do this, YCrCb2RGB, came from a previous year.

3.2 Data Storage

Once RGB data is available, it can be stored to the ZBT memory. This process uses the wrapper block for the ZBT, zbt_6111. In order to use the memory to read and write at the same time, two pixels must be stored for each address in the ZBT. This allows the memory to alternate between reading and writing; it reads and writes every other cycle, getting two pixels of information at each time, so that full information can be transmitted.

The key module in doing this conversion is ntsc_to_zbt_center, which is based on the 2005 code ntsc_to_zbt. However, our module stores two pixels per address, instead of four (allowing 18 bits per pixel), and it rescales the locations in memory. The purpose of this data is to be able to render the opponent in the center of the screen, so pixels are stored in memory pixels that map to coordinates in the center of the screen. This image is also scaled by a factor of two, so the ntsc_to_zbt_center module stores every other pixel into memory. On the other end, the image is displayed below any of the game objects. Display uses code from 2005, again altered to use two 18 bit pixels per memory

address instead of four 9 bit pixels.

The ntsc_to_zbt module, which is similar and will be discussed in the debugging section, also outputs X and Y coordinates for each pixel produced by the camera.

4 Vision - Image Processing

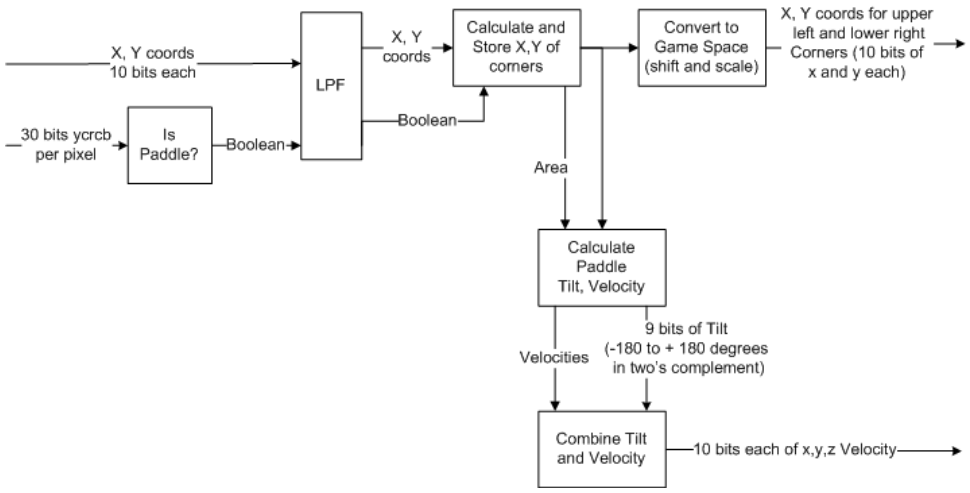

Figure 4: Vision Preprocessing and Storage

4.1 Paddle Pixel Detection

The first stage of image processing is determining which of the pixels are part of the paddle. This process is done with two modules. First, the isPaddle module determines whether or not a pixel is within the proper threshold of Y, Cr, and Cb values to be red. These are adjustable by switches on the labkit, but good values are preloaded.

A boolean value of whether or not the pixel was the right color, along with the coordinates of that pixel, is then passed on to the low pass filter module, lpf. This module looks for a certain percentage of recent pixels (those to the left of the current pixel) to also be red, based on values stored in a shift register. It also checks values above; this is done by using a small block RAM to record whether or not the past few pixels in each column were red, and then shifting these values after they are used to determine whether the current pixel is part of the paddle.

While the data coming out of the color detection is very noisy, the final filtered result detects only relatively large red/orange blobs.

4.2 Paddle Calculations

Once pixels are determined to be part of the paddle, they can be used to perform a number of calculations:

- Calculate the edges of the paddle, which is done by get_extrema within paddle_boundaries. The module keeps track of the max/min of the value passed in, and returns the result after every camera frame. The module is instantiated four times, using generics to change between tracking max and min.

- Calculate the corners of the paddle, which is done by get_extrema_signed within paddle_boundaries. The corners can be find by maximizing/minimizing the sum of X and Y values; for instance, the bottom right corner will have the greatest $X+Y$ value, and the top right corner will have the greatest $X-Y$ value.

- Calculate paddle velocity, which is done by the lateral_velocity and forward_velocity. For lateral velocity, the midpoint of the paddle is calculated, and a simple difference equation is used to approximate the velocity. For forward velocity, the change in calculated paddle area, as calculated by multiplying the height and width of the paddle, is used to approximate velocity in the Z direction.

- Calculate paddle tilt, which is done by using the height/width ratio of the paddle in the tilt_calculation module. This ratio is fed into a ROM I generated, which outputs an angle in degrees. The angle is then made positive or negative by whether the top or bottom of the paddle is wider (as determined by corners). For instance, if the top of the paddle is farther away, the width observed between the top corners is shorter, so the paddle tilt is positive.

- Adjust the paddle velocities to take into account the paddle tilt. The angle is fed into two trigonometry ROMs, which determine how much velocity should be added to the Y and Z velocity of the ball. Holding the paddle at zero degrees merely adds speed to the Z direction of the ball, so it gets faster as the game progresses, while tilting the paddle back adds velocity to the Y direction instead.

- Convert paddle coordinates into the game space, which is done by the coordinate_conversion module. The coordinates of pixels in the camera image have to be shifted and scaled up to occupy the entire screen. This conversion causes the edges of the camera, which are noisy in both filtering and color detection (the thresholds are different at the edges), to be ignored, so that game performance is better.

5 Vision - Debugging Mode

In addition to gameplay, the project provides a vision debugging mode. This mode allows for identifying bugs and was our method for verifying the components of the vision system as they were created.

By flipping a switch, the FPGA will switch from game mode to debug mode, which shows a full size image (no scaling) of what the camera sees. This image can be toggled to show either a grayscale image or a black and white image of which pixels are being detected as paddle pixels.

To accomodate this mode, the ntsc_to_zbt module is used; this module is close to the ntsc_to_zbt_center module, but does not shift and scale pixels sent to the ZBT. In addition to storing pixels, this module outputs the X and Y coordinates of each pixel for use in the rest of the system. These are

calculated by counting each pixel and the image syncing signals.

In addition to showing which pixels are detected, debugging mode displays colored blobs to indicate where the system thinks the edges and corners of the paddle are. Velocities and tilt are output to the hex display, allowing them to be verified as well.

Through using the debugging mode, paddle detection and filtering was made quite robust, and all timing errors related to finding paddle edges and corners were resolved.

6 Output Logic (Z. Clifford)

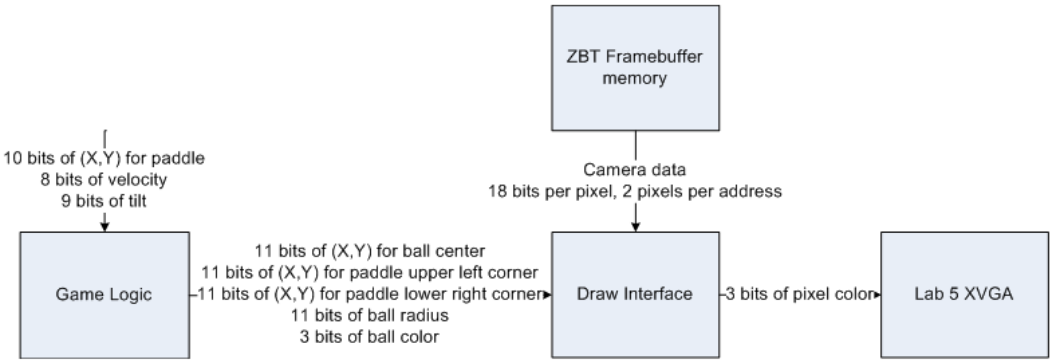

Figure 5: Output

The output logic of the project is responsible for drawing the game elements to the screen. These elements consist of the static background lines, the representation of the ball, the flashing panels when the ball hits a wall, and the recognized position of the paddle. This section interfaces with the XVGA module from Lab 5 to create a 1024 by 768 pixel display for the game. This is the only part of the logic that runs on the full 65 MHz pixel clock. Everything else in the game logic runs at only 27 MHz.

Each object is implemented as a parametrized sprite module that outputs the color that the sprite wants. A large OR gate combines the output from the sprites to prepare a final pixel color for the screen. A priority system is in place to allow the objects to stack, with the paddle on top, the ball in the middle, and the other lines in the back.

The line and ball modules are pipelined because of the complication in calculating them. The line sprites take as parameters the (X,Y) coordinates of the two endpoints. They then calculate the slope of the line between them. The XVGA module outputs the horizontal and vertical pixel counts of the pixel it needs. The line uses this to see if it fits the line $Y - Y_1 = M(X - X_1)$. It also allows for some thickness of the line as a parameter. The line will also output whether the pixel is above or below the line. This is essential for determining whether the wall should blink.

The ball takes as input the (X,Y) of its center, its color, and the square of its radius. It calculates the distance pixels are away from it using $\sqrt{X^2 + Y^2}$ to see if they are less than the theshold. This allows it to change size and color based on the output of the ball projection module. This module is pipelined because of the time involved in squaring these terms.

The output logic also allows walls to flash when the ball hits them. Because each line segment outputs whether a pixel is above or below them, the output logic knows where spaces bounded by lines are. By using a logical AND it can identify spaces to be flashed when necessary.

7 Ball Projection

The ball's position is calculated using a projection matrix. After examining some literature on machine vision, a popular choice is the "Projective Matrix." This technique uses a 4 by 3 matrix that operates on a point in three dimensional space identified by (X, Y, Z, λ), where λ is an arbitrary scaling constant (that I set to one) that makes working with infinity a linear problem. This matrix is usually generated by collecting a set of desired points that map between the three dimensional space and the two dimensional monitor. Using least squares a matrix can be fit to the test data points. I used Matlab to generate the matrix and draw the proposed field. The origin is at the lower left side of the screen with X pointing right, Y pointing into the screen, and Z pointing up.

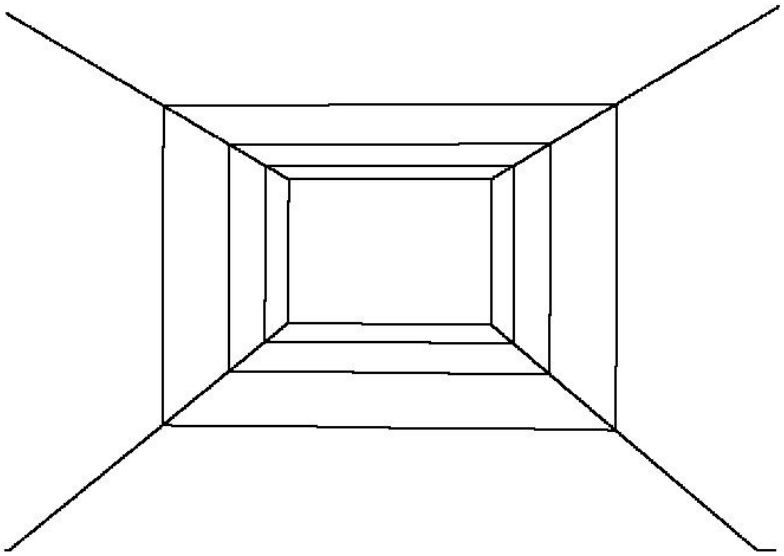

Figure 6: Matlab playfield image

The projection uses the following formula to compute the (U,V) coordinates of the ball where U is the horizontal pixel number and V is the vertical one with (0,0) at the top left of the screen. The arbitrary scaling factor λ has to be divided out of the result of the matrix multiplication.

$$\begin{bmatrix} 3816 & 1412 & 6 & 74997 \\ -46 & 1012 & -3701 & 3158556 \\ 0 & 3 & 0 & 4096 \end{bmatrix} * \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} U\lambda \\ V\lambda \\ \lambda \end{bmatrix} \quad (1)$$

The hard part was translating the method into hardware. Most of the matrix elements were less than one, so I had to scale them up by 4096 because I had no desire to work with floating point numbers in Verilog. Even with this change, the roundoff error (numerical noise) distorts the projected field. The error is reasonably small, so I chose to draw an idealized field on the screen and accept that the back of the field does not match up with the projected back of the field.

8 Multiplayer

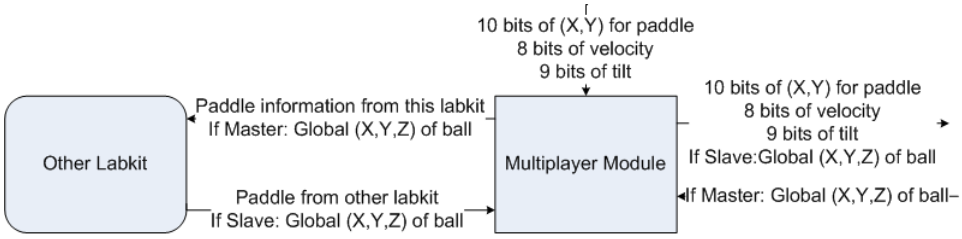

Figure 7: Multiplayer

The multiplayer module serves as a passthrough for most signals of the game. The paddle position, ball position, ball color, and blinking wall. are all passed into the Multiplayer module. Depending on whether the labkit is in Single, Master, or Slave configurations (as set by switches on the labkit), the multiplayer module decides which signals to pass forward to the rest of the game logic. This way none of the other modules need even be aware of the multiplayer state of the machine.

In single player mode, the Multiplayer module acts as a simple passthrough, but for multiplayer modes, it usually swaps data with another labkit. Two labkits are to be connected via a null modem cable between their serial ports. The null modem cable is the serial equivalent of an Ethernet crossover cable that twists the transmit and the receive lines. The multiplayer module uses 115200 baud, 8 bit, 1 stop bit, no parity connection settings to transmit packets of four bytes of data. These are always headed by an ATTENTION marker (currently an exclamation point). Then a COMMAND character is sent to identify the packet. The high byte of data is then sent followed by the low byte. The paddle is identified by an (X, Y) coordinate of its upper left corner and a similar set of data for its lower right corner. The (X, Y, Z) coordinates of the ball and the projected (U, V) are also transmitted. Finally, the status of the walls is shared. Each command identifies which of these pieces of data is about to follow. When a full packet is received, the multiplayer module writes that data to its appropriate output register. On the transmitting side, the multiplayer module is always transmitting each piece of data in sequence from its input registers.

This is implemented as three state machines. The Transmit state machine sets up packets to send and steps through the pieces of information every time a packet is sent. The transmit sub-state machine sends the four bytes of a packet dictated by the main state machine. The receive

state machine waits for an ATTENTION byte and then records the next three bytes. After the full packet is received it places the data in the appropriate register specified by the COMMAND byte.

The serial communication is borrowed from a previous 6.111 term. It encapsulates creating a serial clock by dividing down the system 27 MHz clock and managing the transmission and reception of data packets. Some modification was necessary because the old code depended on a 10 MHz reference crystal clock and not a 27MHz one.

The Master/Slave setting comes into play after data is swapped. The Master labkit is responsible for running the game. It's ball position is the global position, and it is responsible for calculating collisions. The Multiplayer module makes sure to supply the projection module with a reflected ball for the Slave labkit. The projection module of the slave is essential for displaying the correct ball position, but the slave's game logic module gets completely bypassed by the multiplayer module.

9 Game Logic

This module is primarily responsible for calculating the position of the ball in three dimensional space. It internally records the X, Y, and Z velocities of the ball in distance units per frame. It also holds the actual position of the ball. The space is 4096 units long, 1024 units wide, and 768 units tall. Whenever the ball hits a boundary, the appropriate velocity vector is simply negated. When the ball hits the front or rear of the box, the game logic uses the coordinates of the paddle to determine whether a "hit" occurred. It must delay a frame to wait for the ball projection module to give it the pixel coordinates of the ball. This way a hit can be determined by simply seeing if the pixel coordinates of the ball lie within a rectangle representing the paddle. If the player missed the ball, the game stops and that player loses.

Hits are calculated as cheaply as possible. The possible ranges for the positions are 0 - 1023 in X, 0 - 767 in Z, and 0 - 4095 in Y. The ball position variables are stored as signed registers that are too big for this space. That way if the sign bit is set, the module knows a collision occurred. The module can also look at bit 10 to see if X went above 1023 or bit 12 to see if Y went above 4095. The only real comparison operation occurs in the Z axis.

10 Synchronization

One major problem we faced was the synchronization of the video output stage with the rest of the system. The problem involved the fact that the system runs on two different clocks, a 27 MHz one for logic and a 65 MHz one for output to the screen. The signals that pass between these stages must be precisely controlled.

The current mechanism uses synchronizing registers on the signals that must pass between the clock domains and by precisely limiting when those registers are allowed to change. The screen rendering logic always samples its inputs on the falling edge of the vertical sync pulse when the VGA screen is blanking. The inputs to that module are held valid at all times and changed only on the rising edge of the vertical sync signal. The synchronizing modules help the Xilinx router find better paths for the signals. By giving more registers the router is free to space the paths out more.

11 Conclusion

The system works as we expect. The camera can accurately identify the paddle and transmit this information to the game logic side. The inter-labkit communication appears to work well despite being very simple in formatting. Presumably a better system that only sent information when necessary would use less bandwidth, but sending redundant information does not carry any additional cost in this case.

Dealing with the multiple clock domain issue was a problem because we frequently saw glitches in our graphical display. Precisely entering timing constraints, adding synchronization registers, and telling the tools to not remove redundant registers helped with these problems.

We hope you enjoy playing our game!

A Verilog implementation

A.1 arccos.v

```
/*****************************************************************************
*     This file is owned and controlled by Xilinx and must be used            *
*     solely for design, simulation, implementation and creation of           *
*     design files limited to Xilinx devices or technologies. Use             *
*     with non-Xilinx devices or technologies is expressly prohibited         *
*     and immediately terminates your license.                                *
*                                                                             *
*     XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS"           *
*     SOLELY FOR USE IN DEVELOPING PROGRAMS AND SOLUTIONS FOR                 *
*     XILINX DEVICES.  BY PROVIDING THIS DESIGN, CODE, OR INFORMATION         *
*     AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE, APPLICATION             *
*     OR STANDARD, XILINX IS MAKING NO REPRESENTATION THAT THIS               *
*     IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT,                 *
*     AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE        *
*     FOR YOUR IMPLEMENTATION.  XILINX EXPRESSLY DISCLAIMS ANY                *
*     WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE                 *
*     IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR          *
*     REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF         *
*     INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS         *
*     FOR A PARTICULAR PURPOSE.                                               *
*                                                                             *
*     Xilinx products are not intended for use in life support                *
*     appliances, devices, or systems. Use in such applications are           *
*     expressly prohibited.                                                   *
*                                                                             *
*     (c) Copyright 1995-2006 Xilinx, Inc.                                    *
*     All rights reserved.                                                    *
*****************************************************************************/
// The synopsys directives "translate_off/translate_on" specified below are
// supported by XST, FPGA Compiler II, Mentor Graphics and Synplicity synthesis
// tools. Ensure they are correct for your synthesis tool(s).

// You must compile the wrapper file arccos.v when simulating
// the core, arccos. When compiling the wrapper file, be sure to
// reference the XilinxCoreLib Verilog simulation library. For detailed
// instructions, please refer to the "CORE Generator Help".

`timescale 1ns/1ps

module arccos(
addr,
clk,
```

```
dout);


input [7 : 0] addr;
input clk;
output [6 : 0] dout;

// synopsys translate_off

      BLKMEMSP_V6_2 #(
8,// c_addr_width
"0",// c_default_data
256,// c_depth
0,// c_enable_rlocs
0,// c_has_default_data
0,// c_has_din
0,// c_has_en
0,// c_has_limit_data_pitch
0,// c_has_nd
0,// c_has_rdy
0,// c_has_rfd
0,// c_has_sinit
0,// c_has_we
18,// c_limit_data_pitch
"arccos.mif",// c_mem_init_file
0,// c_pipe_stages
0,// c_reg_inputs
"0",// c_sinit_value
7,// c_width
0,// c_write_mode
"0",// c_ybottom_addr
1,// c_yclk_is_rising
1,// c_yen_is_high
"hierarchy1",// c_yhierarchy
0,// c_ymake_bmm
"16kx1",// c_yprimitive_type
1,// c_ysinit_is_high
"1024",// c_ytop_addr
0,// c_yuse_single_primitive
1,// c_ywe_is_high
1) // c_yydisable_warnings
inst (
.ADDR(addr),
.CLK(clk),
.DOUT(dout),
```

```
.DIN(),
.EN(),
.ND(),
.RFD(),
.RDY(),
.SINIT(),
.WE());


// synopsys translate_on

// FPGA Express black box declaration
// synopsys attribute fpga_dont_touch "true"
// synthesis attribute fpga_dont_touch of arccos is "true"

// XST black box declaration
// box_type "black_box"
// synthesis attribute box_type of arccos is "black_box"

endmodule
```

A.2 background.v

```
//////////////////////////////////////////////////////////////////////
//
// background : sprite to draw the static background
//
//////////////////////////////////////////////////////////////////////
module background(hcount,vcount,pixel, vclock, hit_wall);

   input [10:0] hcount;
   input [9:0] vcount;
input vclock;
   output [2:0] pixel;

   wire [2:0] pixel;

input[3:0] hit_wall;

parameter NO_WALL = 4'd0;
parameter RIGHT_WALL = 4'd1;
parameter LEFT_WALL = 4'd2;
parameter TOP_WALL = 4'd3;
```

```
parameter BOTTOM_WALL = 4'd4;
parameter BACK_WALL = 4'd5;
parameter FRONT_WALL = 4'd6;


//wires hold status from lines.  Lines also give whether the pixel is above the line or below.
wire toprightlow,toprighthigh,bottomrightlow,bottomrighthigh,topleftlow,toplefthigh,bottomleftl

wire [2:0] flash_pixel,back_pixel,back_pixel2,back_pixel3, back_pixel4,topleft_pixel,topright_p

wire right_wall_zone, left_wall_zone, bottom_wall_zone, top_wall_zone, back_wall_zone;

assign right_wall_zone = bottomrighthigh & toprightlow;
assign left_wall_zone = bottomlefthigh & topleftlow;
assign bottom_wall_zone = bottomleftlow | bottomrightlow | belowback;
assign top_wall_zone = toplefthigh | toprighthigh | aboveback;
assign back_wall_zone = inback;

//Basic mechanism for flashing the walls
assign flash_pixel = (((hit_wall == LEFT_WALL) && left_wall_zone) ||
((hit_wall == RIGHT_WALL) && right_wall_zone) ||
((hit_wall == TOP_WALL) && top_wall_zone) ||
((hit_wall == BOTTOM_WALL) && bottom_wall_zone) ||
((hit_wall == BACK_WALL) && back_wall_zone)) ? 3'b111 : 0;

assign pixel = flash_pixel | back_pixel | topleft_pixel | topright_pixel | back_pixel2 | back_p
bottomright_pixel | bottomleft_pixel;

//Draw the diagonal lines
drawline topleftline(hcount,vcount,topleft_pixel,vclock, topleftlow, toplefthigh);
drawline toprightline(hcount,vcount,topright_pixel,vclock, toprightlow,toprighthigh);
drawline bottomrightline(hcount,vcount,bottomright_pixel,vclock, bottomrightlow, bottomrighthig
drawline bottomleftline(hcount,vcount,bottomleft_pixel,vclock, bottomleftlow,bottomlefthigh);

//Draw the boxes
   drawbox backbox(hcount, vcount, back_pixel, vclock, aboveback, belowback, inback);

wire above2,below2, in2, above3, below3, in3, above4,below4,in4;

drawbox backbox2(hcount, vcount, back_pixel2, vclock, above2, below2, in2);
defparam backbox2.MINX = 11'd352;
defparam backbox2.MAXX = 11'd671;
defparam backbox2.MINY = 10'd268;  //was 273
defparam backbox2.MAXY = 10'd502;
```

```
drawbox backbox3(hcount, vcount, back_pixel3, vclock, above3, below3, in3);
defparam backbox3.MINX = 11'd305;
defparam backbox3.MAXX = 11'd717;
defparam backbox3.MINY = 10'd231;  //was 245 //was 237
defparam backbox3.MAXY = 10'd540;

drawbox backbox4(hcount, vcount, back_pixel4, vclock, above4, below4, in4);
defparam backbox4.MINX = 11'd225;  //was 221 //was 228
defparam backbox4.MAXX = 11'd800;   //was 802
defparam backbox4.MINY = 10'd172;  //was 196 //was 181
defparam backbox4.MAXY = 10'd600;  //was 612 //was 604

defparam backbox.MINX = 11'd384;
defparam backbox.MAXX = 11'd640;
defparam backbox.MINY = 10'd288;
defparam backbox.MAXY = 10'd480;

defparam topleftline.X1 = 11'd0;
defparam topleftline.Y1 = 10'd0;
defparam topleftline.X2 = 11'd384;
defparam topleftline.Y2 = 10'd288;

defparam toprightline.X1 = 11'd640;
defparam toprightline.Y1 = 10'd288;
defparam toprightline.X2 = 11'd1024;
defparam toprightline.Y2 = 10'd0;

defparam bottomrightline.X1 = 11'd640;
defparam bottomrightline.Y1 = 10'd480;
defparam bottomrightline.X2 = 11'd1024;
defparam bottomrightline.Y2 = 10'd768;

defparam bottomleftline.X1 = 11'd384;
defparam bottomleftline.Y1 = 10'd480;
defparam bottomleftline.X2 = 11'd0;
defparam bottomleftline.Y2 = 10'd768;

endmodule
```

A.3 blob.v

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

```
//
// Create Date:    15:13:44 11/16/2007
// Design Name:
// Module Name:    blob
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////
//
// blob: generate rectangle on screen
//
//////////////////////////////////////////////////////////////////
module blob(x,y,hcount,vcount,pixel);
   parameter WIDTH = 64;      // default width: 64 pixels
   parameter HEIGHT = 64;     // default height: 64 pixels
   parameter COLOR = 3'b101;  // default color: white

   input [10:0] x,hcount;
   input [9:0] y,vcount;
   output [2:0] pixel;

   reg [2:0] pixel;
   always @ (x or y or hcount or vcount) begin
      if ((hcount >= x && hcount < (x+WIDTH)) &&
 (vcount >= y && vcount < (y+HEIGHT)))
pixel = COLOR;
      else pixel = 0;
   end
endmodule
```

A.4 circle.v

```
//////////////////////////////////////////////////////////////////
//
// circle: generate circle on screen
```

```
//
//////////////////////////////////////////////////////////////////////
module circle(x,y,radius_squared,hcount,vcount,pixel,ball_color,vclock);

   input [10:0] x,hcount,radius_squared;
   input [9:0] y,vcount;
input vclock;
input [2:0] ball_color;
   output [2:0] pixel;

reg [10:0] stage1_hcount, last_hcount;
reg [9:0] stage1_vcount, last_vcount;

reg [10:0] hcountdiff;
reg [9:0] vcountdiff;

reg [22:0] vcountsquared;
reg [22:0] hcountsquared;



   reg [2:0] pixel;
   always @ (posedge vclock) begin

//Stage 0. hold count locally
last_hcount <= hcount;
last_vcount <= vcount;

//Stage 1. store the hcount and vcount offset properly
stage1_hcount <= last_hcount >= 11'd1340 ? last_hcount - 11'd1340 : last_hcount + 4;
stage1_vcount <= last_vcount >= 10'd802 ? last_vcount - 10'd802 : last_vcount + 4;

//Stage 2. calculate differences

hcountdiff <= ((stage1_hcount > x) ? stage1_hcount - x : x - stage1_hcount);
vcountdiff <= ((stage1_vcount > y) ? stage1_vcount - y : y - stage1_vcount);

//Stage 3. square them

hcountsquared <= hcountdiff * hcountdiff;
vcountsquared <= vcountdiff * vcountdiff;

//Stage 4. compute pixel

if(hcountsquared + vcountsquared <= radius_squared) pixel <= ball_color;
```

```
else pixel <= 0;
   end
endmodule
```

A.5 control_paddle.v

```
//Dummy module for using the direction buttons to move the paddle.

module control_paddle(up,down,right,left,minX,minY,maxX,maxY,vsync,vclock,reset);
input up,down,right,left,vsync,vclock, reset;
output [10:0] maxX,minX;
output [9:0] maxY,minY;

reg [10:0] maxX,minX;
reg [9:0] maxY,minY;

reg last_vsync;

parameter PADDLE_SPEED = 4;

always @ (posedge vclock) begin   //When vsync goes low, we have finished a frame, so update po
if(reset) begin
maxX <= 700;
maxY <= 700;
minX <= 400;
minY <= 400;
last_vsync <= vsync;
end else begin
last_vsync <= vsync;
if(vsync == 0 && last_vsync == 1) begin
if(up && ~down && minY > PADDLE_SPEED) begin
minY <= minY - PADDLE_SPEED;
maxY <= maxY - PADDLE_SPEED;
end else if(down && ~up && maxY < 767 - PADDLE_SPEED) begin
minY <= minY + PADDLE_SPEED;
maxY <= maxY + PADDLE_SPEED;
end else begin
minY <= minY;
maxY <= maxY;
end

if(left && ~right && minX > PADDLE_SPEED) begin
minX <= minX - PADDLE_SPEED;
maxX <= maxX - PADDLE_SPEED;
```

```
end else if(right && ~left && maxX < 1023 - PADDLE_SPEED) begin
minX <= minX + PADDLE_SPEED;
maxX <= maxX + PADDLE_SPEED;
end else begin
minX <= minX;
maxX <= maxX;
end
end
end
end


endmodule
```

A.6 coordinate_conversion.v

```
‘timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    23:05:27 12/02/2007
// Design Name:
// Module Name:    coordinate_conversion
// Project Name:
// Target Devices:
// Tool versions:
// Description: The coordinates coming out of the camera do not have the
// same range as the lcd display. In order to draw an accurate paddle, these
// coordinates must be shifted and scaled up to fit the entire screen properly.
// This shift and scale also removes
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module coordinate_conversion(clk, rst, min_x, max_x, min_y, max_y,
x_min_converted, x_max_converted, y_min_converted, y_max_converted);
    input clk;
 input rst;
 input [9:0] min_x, max_x, min_y, max_y;
 output reg [10:0] x_min_converted, x_max_converted;
```

```
 output reg [9:0] y_min_converted, y_max_converted;

reg [9:0] paddleminY_IN, paddlemaxY_IN, p_y_min_q, p_y_max_q;
reg [9:0] paddleminX_IN, paddlemaxX_IN, p_y_min_shift, p_y_max_shift,
p_x_min_shift, p_x_max_shift;
reg [10:0] p_x_min_q, p_x_max_q;
always @ (posedge clk) begin
paddlemaxY_IN <= max_y;//{max_y[8:0], 0};
paddleminY_IN <= min_y;//{0, min_y[8:0]};
paddlemaxX_IN <= max_x;//{max_x, 0}; //shift and scale
paddleminX_IN <= min_x;
p_y_min_shift <= (paddleminY_IN > 128) ?
((paddleminY_IN > 511) ? 384 : paddleminY_IN - 128) : 0;
p_y_max_shift <= (paddlemaxY_IN > 128) ?
((paddlemaxY_IN > 511) ? 384 : paddlemaxY_IN - 128) : 0;
p_x_min_shift <= (paddleminX_IN > 128) ?
((paddleminX_IN > 640) ? 0 : 640 - paddleminX_IN) : 10'd512;
p_x_max_shift <= (paddlemaxX_IN > 128) ?
((paddlemaxX_IN > 640) ? 0 : 640 - paddlemaxX_IN) : 10'd512;
y_min_converted <= {p_y_min_shift[8:0], 1'b0};//paddleminY_IN;
y_max_converted <= {p_y_max_shift[8:0], 1'b0};//{paddlemaxY_IN[8:0], 1'b0};
x_max_converted <= {p_x_min_shift, 1'b0};//paddlemaxX_IN;
x_min_converted <= {p_x_max_shift, 1'b0};//{paddlemaxX_IN[9:0], 1'b0};
//y_min_converted <= paddleminY_IN;
//y_max_converted <= paddlemaxY_IN;//[8:0], 1'b0};
//x_min_converted <= {1'b0, paddleminX_IN};
//x_max_converted <= {1'b0, paddlemaxX_IN};//[9:0], 1'b0};
end

endmodule
```

A.7 coords_for_center.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    16:16:04 12/04/2007
// Design Name:
// Module Name:    coords_for_center_image
// Project Name:
// Target Devices:
// Tool versions:
// Description:
```

```
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module coords_for_center_image(clk, rst, hcount, vcount, pixel, in_range);
    input clk, rst;
 input [10:0] hcount;
 input [9:0] vcount;
 input [2:0] pixel;
 output reg in_range;



 always @ (posedge clk) begin
in_range <= (pixel[2] | pixel[1] | pixel[0]) ?
1'b0 :
(hcount > 382) & (vcount > 287)
& (~(hcount > 639)) & (~(vcount > 479));
 end




endmodule
```

A.8 cos2theta.v

```
/*****************************************************************************
*     This file is owned and controlled by Xilinx and must be used           *
*     solely for design, simulation, implementation and creation of          *
*     design files limited to Xilinx devices or technologies. Use            *
*     with non-Xilinx devices or technologies is expressly prohibited        *
*     and immediately terminates your license.                               *
*                                                                            *
*     XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS"          *
*     SOLELY FOR USE IN DEVELOPING PROGRAMS AND SOLUTIONS FOR                *
*     XILINX DEVICES.  BY PROVIDING THIS DESIGN, CODE, OR INFORMATION        *
*     AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE, APPLICATION            *
*     OR STANDARD, XILINX IS MAKING NO REPRESENTATION THAT THIS              *
*     IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT,                *
*     AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE       *
```

```
*     FOR YOUR IMPLEMENTATION.  XILINX EXPRESSLY DISCLAIMS ANY                 *
*     WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE                  *
*     IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR           *
*     REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF          *
*     INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS          *
*     FOR A PARTICULAR PURPOSE.                                                *
*                                                                              *
*     Xilinx products are not intended for use in life support                 *
*     appliances, devices, or systems. Use in such applications are            *
*     expressly prohibited.                                                    *
*                                                                              *
*     (c) Copyright 1995-2006 Xilinx, Inc.                                     *
*     All rights reserved.                                                     *
*******************************************************************************/
// The synopsys directives "translate_off/translate_on" specified below are
// supported by XST, FPGA Compiler II, Mentor Graphics and Synplicity synthesis
// tools. Ensure they are correct for your synthesis tool(s).

// You must compile the wrapper file cos2theta.v when simulating
// the core, cos2theta. When compiling the wrapper file, be sure to
// reference the XilinxCoreLib Verilog simulation library. For detailed
// instructions, please refer to the "CORE Generator Help".

`timescale 1ns/1ps

module cos2theta(
clka,
addra,
douta);


input clka;
input [7 : 0] addra;
output [7 : 0] douta;

// synopsys translate_off

      BLK_MEM_GEN_V1_1 #(
8,// c_addra_width
11,// c_addrb_width
1,// c_algorithm
9,// c_byte_size
0,// c_common_clk
"0",// c_default_data
0,// c_disable_warn_bhv_coll
```

```
0,// c_disable_warn_bhv_range
"virtex2",// c_family
0,// c_has_ena
0,// c_has_enb
0,// c_has_mem_output_regs
0,// c_has_mux_output_regs
0,// c_has_regcea
0,// c_has_regceb
0,// c_has_ssra
0,// c_has_ssrb
"cos2theta.mif",// c_init_file_name
1,// c_load_init_file
3,// c_mem_type
1,// c_prim_type
256,// c_read_depth_a
2048,// c_read_depth_b
8,// c_read_width_a
1,// c_read_width_b
"ALL",// c_sim_collision_check
"0",// c_sinita_val
"0",// c_sinitb_val
0,// c_use_byte_wea
0,// c_use_byte_web
0,// c_use_default_data
1,// c_wea_width
1,// c_web_width
256,// c_write_depth_a
2048,// c_write_depth_b
"WRITE_FIRST",// c_write_mode_a
"WRITE_FIRST",// c_write_mode_b
8,// c_write_width_a
1) // c_write_width_b
inst (
.CLKA(clka),
.ADDRA(addra),
.DOUTA(douta),
.DINA(),
.ENA(),
.REGCEA(),
.WEA(),
.SSRA(),
.CLKB(),
.DINB(),
.ADDRB(),
.ENB(),
```

```
.REGCEB(),
.WEB(),
.SSRB(),
.DOUTB());


// synopsys translate_on

// FPGA Express black box declaration
// synopsys attribute fpga_dont_touch "true"
// synthesis attribute fpga_dont_touch of cos2theta is "true"

// XST black box declaration
// box_type "black_box"
// synthesis attribute box_type of cos2theta is "black_box"

endmodule
```

A.9 count_area.v

```
‘timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    17:53:37 12/04/2007
// Design Name:
// Module Name:    count_area
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module count_area(clk, rst, use_paddle, field_change, area);
    input clk;
    input rst;
    input use_paddle;
```

```
 input field_change;
    output reg [19:0] area;

 reg [19:0] area_temp;

always @ (posedge clk) begin
if (rst)
area_temp <= 0;
else if (field_change) begin
area_temp <= 0;
area <= area_temp;
end
else
area_temp <= use_paddle ? (area_temp + 1) : area_temp;
end


endmodule
```

A.10 debounce.v

```
////////////////////////////////////////////////////////////////////////////////
//
// 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.11 debounce_block.v

```
‘timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    15:22:56 12/03/2007
// Design Name:
// Module Name:    debounce_block
// Project Name:
// Target Devices:
// Tool versions:
// Description: Wrapper for debouncers so they do not clutter
// the top level.
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module debounce_block(clk, power_on_reset, button_enter, button_up, button_down,
switch, sw0, sw1, sw2, sw3, sw4, sw5, sw6, sw7, user_reset, up, down);
    input clk, power_on_reset;
 input button_enter, button_up, button_down;
 input [7:0] switch;
 output sw0, sw1, sw2, sw3, sw4, sw5, sw6, sw7;
 output user_reset, up, down;


debounce db1(power_on_reset, clk, ~button_enter, user_reset);
debounce db2(power_on_reset, clk, switch[1], sw1);
debounce db3(power_on_reset, clk, switch[2], sw2);
debounce db4(power_on_reset, clk, switch[3], sw3);
debounce db5(power_on_reset, clk, switch[4], sw4);
debounce db6(power_on_reset, clk, switch[5], sw5);
debounce db7(power_on_reset, clk, switch[6], sw6);
debounce db8(power_on_reset, clk, button_up, up);
debounce db9(power_on_reset, clk, ~button_down, down);
debounce db10(power_on_reset, clk, switch[7], sw7);
debounce db11(power_on_reset, clk, switch[0], sw0);
endmodule
```

A.12 debug_block.v

```
‘timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    15:03:45 12/03/2007
// Design Name:
// Module Name:    debug_block
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module debug_block(clk);
    input clk;


endmodule
```

A.13 debugPixels.v

```
‘timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    14:03:17 12/03/2007
// Design Name:
// Module Name:    debugPixels
// Project Name:
// Target Devices:
// Tool versions:
// Description:
```

```
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module debugPixels(clk, rst, sw7, up, sw1, paddle_use, min_x, max_x, min_y, max_y,
x_coord, y_coord,
x_ul, x_ur, x_bl, x_br, y_ul, y_ur, y_bl, y_br, ycrcb, rgb_val);
    input clk;
 input rst;
 input sw7;
 input up, sw1;
 input paddle_use;
 input [9:0] min_x, max_x, min_y, max_y, x_coord, y_coord;
 input [9:0] x_ul, x_ur, x_bl, x_br, y_ul, y_ur, y_bl, y_br;
 input [29:0] ycrcb;
 output reg [17:0] rgb_val;

wire [2:0] ul_corner, ur_corner, bl_corner, br_corner;
wire [2:0] ul_pixel, ur_pixel, bl_pixel, br_pixel, tb_pixel;
reg [2:0] comb_pixel;

 //mark corners
blob ul(min_x, min_y, x_coord, y_coord, ul_pixel);
defparam ul.COLOR = 3'b100;
blob ur(max_x, min_y, x_coord, y_coord, ur_pixel);
defparam ur.COLOR = 3'b010;
blob bl(min_x, max_y, x_coord, y_coord, bl_pixel);
defparam bl.COLOR = 3'b001;
blob br(max_x, max_y, x_coord, y_coord, br_pixel);
defparam br.COLOR = 3'b110;



//mark true corners
blob ulc(x_ul, y_ul, x_coord, y_coord, ul_corner);
defparam ulc.COLOR = 3'b100;
defparam ulc.WIDTH = 16;
defparam ulc.HEIGHT = 16;
blob urc(x_ur, y_ur, x_coord, y_coord, ur_corner);
defparam urc.COLOR = 3'b010;
defparam urc.WIDTH = 16;
```

```
defparam urc.HEIGHT = 16;
blob blc(x_bl, y_bl, x_coord, y_coord, bl_corner);
defparam blc.COLOR = 3'b001;
defparam blc.WIDTH = 16;
defparam blc.HEIGHT = 16;
blob brc(x_br, y_br, x_coord, y_coord, br_corner);
defparam brc.COLOR = 3'b110;
defparam brc.WIDTH = 16;
defparam brc.HEIGHT = 16;

always @ (posedge clk) begin
comb_pixel <= sw7 ? (ul_pixel | ur_pixel | bl_pixel | br_pixel)
: (ul_corner | ur_corner | bl_corner | br_corner);
rgb_val <= (paddle_use == 1) ? 18'h3ffff :
(~(up & sw1)) ? {ycrcb[29:24],ycrcb[29:24],ycrcb[29:24]} :
{comb_pixel[2],5'd0,comb_pixel[1],5'd0,comb_pixel[0],5'd0};
end

endmodule
```

A.14 display_16hex.v

```
//////////////////////////////////////////////////////////////////////////////
//
// 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.15 drawbox.v

```
//Draws a box on screen out of the line sprites.
module drawbox(hcount, vcount, pixel, vclock, above, below, inside);
  input [10:0] hcount;
   input [9:0] vcount;
input vclock;
   output [2:0] pixel;

output above, below, inside;

wire [2:0] top_pixel,right_pixel,left_pixel,bottom_pixel;
wire toplow,tophigh,bottomlow,bottomhigh;

assign above = tophigh;
assign below = bottomlow;
assign inside = toplow & bottomhigh;

parameter MINX = 0;
parameter MAXX = 0;
parameter MINY = 0;
parameter MAXY = 0;

wire righthigh, rightlow, lefthigh, leftlow;

drawline topline(hcount,vcount,top_pixel,vclock,toplow, tophigh);
drawline rightline(hcount,vcount,right_pixel,vclock, rightlow, righthigh);
drawline bottomline(hcount,vcount,bottom_pixel,vclock, bottomlow, bottomhigh);
```

```
drawline leftline(hcount,vcount,left_pixel,vclock, leftlow, lefthigh);

defparam topline.X1 = MINX;
defparam topline.Y1 = MINY;
defparam topline.X2 = MAXX;
defparam topline.Y2 = MINY;

defparam rightline.X1 = MAXX;
defparam rightline.Y1 = MINY;
defparam rightline.X2 = MAXX;
defparam rightline.Y2 = MAXY;

defparam bottomline.X1 = MINX;
defparam bottomline.Y1 = MAXY;
defparam bottomline.X2 = MAXX;
defparam bottomline.Y2 = MAXY;

defparam leftline.X1 = MINX;
defparam leftline.Y1 = MINY;
defparam leftline.X2 = MINX;
defparam leftline.Y2 = MAXY;

assign pixel = top_pixel | right_pixel | bottom_pixel | left_pixel;

endmodule
```

A.16 drawline.v

```
//////////////////////////////////////////////////////////////////////
//
// drawline: generate a line on screen between
// (x1,y1) and (x2,y2)
//
//////////////////////////////////////////////////////////////////////
module drawline(hcount,vcount,pixel, vclock, toolow,toohigh);
   parameter COLOR = 3'b111;  // default color: white
parameter THICKNESS = 2'b11;  //Thickness of the line
parameter X1 = 11'd152;
parameter Y1 = 10'd144;
parameter X2 = 11'd872;
parameter Y2 = 10'd144;

   input [10:0] hcount;
   input [9:0] vcount;
```

```
input vclock;   //65 MHz clock
   output [2:0] pixel;

output toolow, toohigh;
reg toolow,toohigh;

   reg [2:0] pixel;

wire [17:0] slope;  //Going to shift left to avoid fractions

wire negslope,argnegative,shouldnegate;

reg [9:0] stage1_vcount, stage2_vcount, stage3_vcount, last_vcount;
reg [10:0] stage1_hcount, stage2_hcount, stage3_hcount, last_hcount;
reg [29:0] stage2_slope_product, stage3_slope_product;
reg stage2_shouldnegate, stage3_shouldnegate;


parameter VERTICAL_LINE = 10'b1111111111;  //If the slope is this high, I'll just draw a vertic

assign slope = (X2 == X1) ? VERTICAL_LINE : (((Y2 > Y1) ? (Y2-Y1) : (Y1-Y2)) << 8)
/
  ((X2 > X1) ?  (X2-X1) : (X1-X2));   //I don't want to deal with signed division or multiplica

//Slope will always be positive, so I need to know whether it started negative

assign negslope = (((X2 > X1) && (Y1 > Y2)) ||
 ((X1 > X2) && (Y2 > Y1))
 );  //Will be 1 if the slope was negative

assign argnegative = (X1 > stage1_hcount);  //Denotes that the (hcount - X1) is negative

assign shouldnegate = (argnegative ^ negslope);  //Tells which case I should follow (do I subt

wire [29:0] slope_product;

assign slope_product = slope*( (stage1_hcount > X1) ? (stage1_hcount - X1) : (X1 - stage1_hcou



   always @ (posedge vclock) begin

//Stage 0. Store vcount and hcount
last_vcount <= vcount;
last_hcount <= hcount;
```

```
//Stage 1. Store values

stage1_vcount <= last_vcount >= 10'd802 ? last_vcount - 10'd802 : last_vcount + 4;
stage1_hcount <= last_hcount >= 11'd1340 ? last_hcount - 11'd1340 : last_hcount + 4;


//Stage 2. Compute slope product and if it is negative

stage2_slope_product <= slope_product;
stage2_shouldnegate <= shouldnegate;
stage2_vcount <= stage1_vcount;
stage2_hcount <= stage1_hcount;

//Stage 3 Give multiply another clock cycle to do its magic

stage3_slope_product <= stage2_slope_product;
stage3_shouldnegate <= stage2_shouldnegate;
stage3_vcount <= stage2_vcount;
stage3_hcount <= stage2_hcount;

//Stage 4. Output the pixel

if(slope == VERTICAL_LINE) pixel <= ((stage3_hcount <= X1 + THICKNESS) &&
(stage3_hcount >= X1 - THICKNESS) &&
(stage3_vcount >= ((Y2 > Y1) ? Y1 : Y2)) &&
(stage3_vcount <= ((Y2 > Y1) ? Y2 : Y1))
 )  ? COLOR : 0;  //If it's a vertical line, just check X
   else case(stage3_shouldnegate)
1'b0: pixel <= ((stage3_vcount <= ((stage3_slope_product >> 8) + Y1 + THICKNESS)) &&
   (stage3_vcount >= ((stage3_slope_product >> 8) + Y1 - THICKNESS)) &&
   (stage3_vcount >= ((Y2 > Y1) ? ((Y1 < THICKNESS) ? 0 : Y1 - THICKNESS) : ((Y2 < THICKNESS) ?
   (stage3_vcount <= ((Y2 > Y1) ? Y2 + THICKNESS : Y1 + THICKNESS)) &&
   (stage3_hcount >= ((X2 > X1) ? ((X1 < THICKNESS) ? 0 : X1 - THICKNESS) : ((X2 < THICKNESS) ?
   (stage3_hcount <= ((X2 > X1) ? X2 + THICKNESS : X1 + THICKNESS))
  ) ? COLOR : 0;
1'b1: pixel <= ((stage3_vcount <= (Y1 - (stage3_slope_product >> 8) + THICKNESS)) &&
     (stage3_vcount >= (Y1 - (stage3_slope_product >> 8) - THICKNESS)) &&
   (stage3_vcount >= ((Y2 > Y1) ? ((Y1 < THICKNESS) ? 0 : Y1 - THICKNESS) : ((Y2 < THICKNESS) ?
   (stage3_vcount <= ((Y2 > Y1) ? Y2 + THICKNESS : Y1 + THICKNESS)) &&
   (stage3_hcount >= ((X2 > X1) ? ((X1 < THICKNESS) ? 0 : X1 - THICKNESS) : ((X2 < THICKNESS) ?
   (stage3_hcount <= ((X2 > X1) ? X2 + THICKNESS : X1 + THICKNESS))
  ) ? COLOR : 0;
default: pixel <= 0;
endcase
```

```
//Now compute if the pixel was below the line or above the line or neither
if(slope == VERTICAL_LINE) toolow <= 0;
else case(stage3_shouldnegate)
1'b0: toolow <= ((stage3_vcount >= ((stage3_slope_product >> 8) + Y1 + THICKNESS)) &&
   (stage3_hcount >= ((X2 > X1) ? ((X1 < THICKNESS) ? 0 : X1 - THICKNESS) : ((X2 < THICKNESS) ?
   (stage3_hcount <= ((X2 > X1) ? X2 + THICKNESS : X1 + THICKNESS))
  ) ? 1 : 0;
1'b1: toolow <= ((stage3_vcount >= (Y1 - (stage3_slope_product >> 8) + THICKNESS)) &&
   (stage3_hcount >= ((X2 > X1) ? ((X1 < THICKNESS) ? 0 : X1 - THICKNESS) : ((X2 < THICKNESS) ?
   (stage3_hcount <= ((X2 > X1) ? X2 + THICKNESS : X1 + THICKNESS))
  ) ? 1 : 0;
default: toolow <= 0;
endcase

if(slope == VERTICAL_LINE) toohigh <= 0;
else case(stage3_shouldnegate)
1'b0: toohigh <= ((stage3_vcount <= ((stage3_slope_product >> 8) + Y1 - THICKNESS)) &&
   (stage3_hcount >= ((X2 > X1) ? ((X1 < THICKNESS) ? 0 : X1 - THICKNESS) : ((X2 < THICKNESS) ?
   (stage3_hcount <= ((X2 > X1) ? X2 + THICKNESS : X1 + THICKNESS))
  ) ? 1 : 0;
1'b1: toohigh <= ((stage3_vcount <= (Y1 - (stage3_slope_product >> 8) - THICKNESS)) &&
   (stage3_hcount >= ((X2 > X1) ? ((X1 < THICKNESS) ? 0 : X1 - THICKNESS) : ((X2 < THICKNESS) ?
   (stage3_hcount <= ((X2 > X1) ? X2 + THICKNESS : X1 + THICKNESS))
  ) ? 1 : 0;
default: toohigh <= 0;
endcase

end
endmodule
```

A.17 forward_velocity.v

```
‘timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    19:10:00 12/05/2007
// Design Name:
// Module Name:    forward_velocity
// Project Name:
// Target Devices:
// Tool versions:
// Description:
```

```
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module forward_velocity(clk, rst, area, velocity);
    input clk;
    input rst;
    input signed [8:0] area;
 output reg signed [9:0] velocity;

 reg signed [8:0] old_area;

 always @ (posedge clk) begin
if (~(area == old_area)) begin
velocity <= area - old_area;
old_area <= area;
end
 end


endmodule
```

A.18 game_logic.v

```
//////////////////////////////////////////////////////////////////////////////////
//
// game_logic: computes new position for the ball
//
//////////////////////////////////////////////////////////////////////////////////

module game_logic (vclock,reset,pspeed, //angle,
paddle_x_change_LOCAL, paddle_y_change_LOCAL, paddle_z_change_LOCAL,
paddle_x_change_REMOTE, paddle_y_change_REMOTE, paddle_z_change_REMOTE,
  vsync, ballX, ballY, ballZ, ballpixelX, ballpixelY, ballpixelX_opp, ballpixelY_opp, paddlemi
  opp_paddleminX, opp_paddleminY, opp_paddlemaxX, opp_paddlemaxY, hit_wall);

   input vclock; // 65MHz clock
   input reset; // 1 to initialize module
   input [3:0] pspeed;  // puck speed in pixels/tick
//input [7:0] angle;
```

```
 //  input [10:0] hcount; // horizontal index of current pixel (0..1023)
 //  input [9:0]  vcount; // vertical index of current pixel (0..767)
input signed [9:0] paddle_x_change_LOCAL, paddle_y_change_LOCAL, paddle_z_change_LOCAL;
input signed [9:0] paddle_x_change_REMOTE, paddle_y_change_REMOTE, paddle_z_change_REMOTE;


   input vsync; // XVGA vertical sync signal (active low)

output [3:0] hit_wall;
reg [3:0] hit_wall;


//Parameters for hit_wall variable
parameter NO_WALL = 4'd0;
parameter RIGHT_WALL = 4'd1;
parameter LEFT_WALL = 4'd2;
parameter TOP_WALL = 4'd3;
parameter BOTTOM_WALL = 4'd4;
parameter BACK_WALL = 4'd5;
parameter FRONT_WALL = 4'd6;


input [10:0] paddleminX, paddlemaxX, opp_paddleminX, opp_paddlemaxX, ballpixelX, ballpixelX_opp
input [9:0] paddleminY, paddlemaxY, opp_paddleminY, opp_paddlemaxY, ballpixelY, ballpixelY_opp

output signed [12:0] ballX;
output signed [12:0] ballY;
output signed [12:0] ballZ;


wire [2:0] puck_pixel; //The pixel output by the puck sprite
reg signed [12:0] puck_y;      //The upper left corner Y value of the puck and new value
reg signed [12:0] puck_x;     //The upper left corner X value of the puck and new value
reg signed [12:0] puck_z;


wire signed [13:0] new_puck_y;  //bit 13 is sign, and bit 12 is 4096, and bit 10 is 1024
wire signed [13:0] new_puck_x;
wire signed [13:0] new_puck_z;
wire [10:0] puck_size;

reg signed [7:0] xspeed,yspeed,zspeed;

reg check_collision, has_lost, last_vsync;
```

```
assign new_puck_y = puck_y + yspeed;
assign new_puck_x = puck_x + xspeed;
assign new_puck_z = puck_z + zspeed;

//wire signed [7:0] cos2T, sin2T;
//reg signed [15:0] new_puck_y_tilt, new_puck_z_tilt;

//sin2theta sin(vclock, angle, sin2T);
//cos2theta cos(vclock, angle, cos2T);

//always @ (posedge vclock) begin
// new_puck_y_tilt <= yspeed * cos2T - zspeed * sin2T;
// new_puck_z_tilt <= -yspeed * sin2T - zspeed * cos2T;
//end

//wire [8:0] theta;
//assign theta = 9'd0; //Tilt of paddle

reg last_hit;  //Tells if we hit a wall last frame.  Lets a wall flash for 2 frames.

//wire signed [7:0] yspeedangle, zspeedangle; //Modified bounce angles from the paddle
reg signed [7:0] xspeedPaddle_LOCAL, yspeedPaddle_LOCAL, zspeedPaddle_LOCAL;
reg signed [7:0] xspeedPaddle_REMOTE, yspeedPaddle_REMOTE, zspeedPaddle_REMOTE;

//assign yspeedangle = yspeed;
//assign zspeedangle = -zspeed;
always @ (posedge vclock) begin
xspeedPaddle_LOCAL <= xspeed - paddle_x_change_LOCAL;
yspeedPaddle_LOCAL <= yspeed + paddle_y_change_LOCAL;
zspeedPaddle_LOCAL <= -zspeed + paddle_z_change_LOCAL;
xspeedPaddle_REMOTE <= xspeed + paddle_x_change_REMOTE;
yspeedPaddle_REMOTE <= yspeed + paddle_y_change_REMOTE;
zspeedPaddle_REMOTE <= -zspeed - paddle_z_change_REMOTE;
end

//anglebouncer myanglebouncer(theta, zspeed, yspeed, zspeedangle, yspeedangle);

always @ (posedge vclock) begin   //When vsync goes low, we have finished a frame, so update po
if(reset) begin               //Reset puck positions
puck_y <= 600;
puck_x <= 600;
puck_z <= 700;
xspeed <= pspeed;
yspeed <= pspeed;
zspeed <= pspeed;
```

```
last_vsync <= vsync;
check_collision <= 0;
has_lost <= 0;
hit_wall <= NO_WALL;
last_hit <= 0;
end else begin
last_vsync <= vsync;

if(vsync == 0 && last_vsync == 1) begin
//If we've lost, freeze
if(has_lost == 1) begin
has_lost <= 1;

//If we said to check a collision last frame, do it.
end else if(check_collision == 1) begin
check_collision <= 0;
if(puck_z <= 13'd2048 && ((ballpixelX > paddlemaxX) ||
(ballpixelX < paddleminX) ||
(ballpixelY > paddlemaxY) ||
(ballpixelY < paddleminY))) begin
xspeed <= 0;
yspeed <= 0;
zspeed <= 0;
has_lost <= 1;
end else if(puck_z > 13'd2048 && (ballpixelX_opp > opp_paddlemaxX || ballpixelX_opp < opp_paddl
//Opponent missed
xspeed <= 0;
yspeed <= 0;
zspeed <= 0;
has_lost <= 1;
end

end else if(new_puck_x[13]) begin  //x is negative, hit left wall
xspeed <= -xspeed;
yspeed <= yspeed - 1;
zspeed <= zspeed;
hit_wall <= LEFT_WALL;
last_hit <= 1;

end else if(new_puck_y[13]) begin //y is negative, hit bottom wall
xspeed <= xspeed;
yspeed <= -yspeed - 1;
zspeed <= zspeed;
hit_wall <= BOTTOM_WALL;
last_hit <= 1;
```

```
end else if(new_puck_z[13]) begin  //Hit front of screen
check_collision <= 1;
// xspeed <= xspeed;
// yspeed <= yspeed;
// zspeed <= -zspeed;
xspeed <= xspeedPaddle_LOCAL;
yspeed <= yspeedPaddle_LOCAL;
zspeed <= zspeedPaddle_LOCAL;
//yspeed <= new_puck_y_tilt[14:7];
//zspeed <= new_puck_z_tilt[14:7];
hit_wall <= FRONT_WALL;

end else if(new_puck_x[10]) begin //x greater than or equal to 1024, right wall

xspeed <= -xspeed;
yspeed <= yspeed;
zspeed <= zspeed;
hit_wall <= RIGHT_WALL;
last_hit <= 1;

end else if(new_puck_y >= 768) begin //y greater than 768, top wall
xspeed <= xspeed;
yspeed <= -yspeed;
zspeed <= zspeed;
hit_wall <= TOP_WALL;
last_hit <= 1;

end else if(new_puck_z[12]) begin //Z greater than 4096, back wall
check_collision <= 1;
//xspeed <= xspeed;
//yspeed <= yspeed;
//zspeed <= -zspeed;
xspeed <= xspeedPaddle_REMOTE;
yspeed <= yspeedPaddle_REMOTE;
zspeed <= zspeedPaddle_REMOTE;
//yspeed <= new_puck_y_tilt[21:8];
//zspeed <= new_puck_z_tilt[21:8];
hit_wall <= BACK_WALL;
last_hit <= 1;
end else begin //No collisions, so just move forward
puck_y <= new_puck_y;
puck_x <= new_puck_x;
puck_z <= new_puck_z;
xspeed <= xspeed;
```

```
yspeed <= yspeed - 1;
zspeed <= zspeed;
if(last_hit) last_hit <= 0;  //Clear blinked wall.
else hit_wall <= NO_WALL;
end
end
end
end

//In projection-land, the coordinates are a bit different...
assign ballX = puck_x;
assign ballY = puck_z;
assign ballZ = puck_y;




endmodule
```

A.19 gamelogic.v

```
//////////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Template Toplevel Module
//
// For Labkit Revision 004
//
//
// Created: October 31, 2004, from revision 003 file
// Author: Nathan Ickes
//
//////////////////////////////////////////////////////////////////////////////////
//
// CHANGES FOR BOARD REVISION 004
//
// 1) Added signals for logic analyzer pods 2-4.
// 2) Expanded "tv_in_ycrcb" to 20 bits.
// 3) Renamed "tv_out_data" to "tv_out_i2c_data" and "tv_out_sclk" to
//    "tv_out_i2c_clock".
// 4) Reversed disp_data_in and disp_data_out signals, so that "out" is an
//    output of the FPGA, and "in" is an input.
//
// CHANGES FOR BOARD REVISION 003
//
// 1) Combined flash chip enables into a single signal, flash_ce_b.
```

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

module lab5   (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'b0;
assign tv_in_fifo_clock = 1'b0;
assign tv_in_iso = 1'b0;
assign tv_in_reset_b = 1'b0;
assign tv_in_clock = 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
assign ram0_data = 36'hZ;
assign ram0_address = 19'h0;
assign ram0_adv_ld = 1'b0;
assign ram0_clk = 1'b0;
assign ram0_cen_b = 1'b1;
assign ram0_ce_b = 1'b1;
assign ram0_oe_b = 1'b1;
assign ram0_we_b = 1'b1;
assign ram0_bwe_b = 4'hF;
assign ram1_data = 36'hZ;
assign ram1_address = 19'h0;
assign ram1_adv_ld = 1'b0;
assign ram1_clk = 1'b0;
assign ram1_cen_b = 1'b1;
assign ram1_ce_b = 1'b1;
assign ram1_oe_b = 1'b1;
assign ram1_we_b = 1'b1;
assign ram1_bwe_b = 4'hF;
assign clock_feedback_out = 1'b0;
// clock_feedback_in is an input

// Flash ROM
assign flash_data = 16'hZ;
assign flash_address = 24'h0;
assign flash_ce_b = 1'b1;
assign flash_oe_b = 1'b1;
assign flash_we_b = 1'b1;
assign flash_reset_b = 1'b0;
assign flash_byte_b = 1'b1;
// flash_sts is an input
```

```
   // RS-232 Interface

//Made by Zach

   //assign rs232_txd = 1'b1;
   assign rs232_rts = 1'b1;

//End made by Zach

   // rs232_rxd and rs232_cts are inputs

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

   // LED Displays

//Made by Zach

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;

//End Made by Zach

// disp_data_in is an input

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

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

   // Daughtercard Connectors
   assign daughtercard = 44'hZ;

   // SystemACE Microprocessor Port
```

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

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

   ////////////////////////////////////////////////////////////////////////
   //
   // lab5 : a simple pong game
   //
   ////////////////////////////////////////////////////////////////////////

wire logic_clock;

BUFG vclk3(.O(logic_clock),.I(clock_27mhz));


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

//Added to make constraints happen

// synthesis attribute period of clock_65mhz is "15";

//Done with addition

   BUFG vclk2(.O(clock_65mhz),.I(clock_65mhz_unbuf));
```

```
   // power-on reset generation
   wire power_on_reset;    // remain high for first 16 clocks
   SRL16 reset_sr (.D(1'b0), .CLK(clock_65mhz), .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, logic_clock, ~button_enter, user_reset);
   assign reset = user_reset | power_on_reset;

   // UP and DOWN buttons for pong paddle
   wire up,down,right,left;
   debounce db2(reset, logic_clock, ~button_up, up);
   debounce db3(reset, logic_clock, ~button_down, down);
debounce db4(reset, logic_clock, ~button_right, right);
   debounce db5(reset, logic_clock, ~button_left, left);

   // generate basic XVGA video signals
   wire [10:0] hcount;
   wire [9:0]  vcount;
   wire hsync,vsync,blank;
   xvga xvga1(clock_65mhz,hcount,vcount,hsync,vsync,blank);

   // feed XVGA signals to game
   wire [2:0] pixel;
   wire phsync,pvsync,pblank;


assign phsync = hsync;
assign pvsync = vsync;
assign pblank = blank;

   // switch[1:0] selects which video generator to use:
   //  00: user's pong game
   //  01: 1 pixel outline of active video area (adjust screen controls)
   //  10: color bars
   reg [2:0] rgb;
   reg b,hs,vs;
   always @(posedge clock_65mhz) begin
      if (switch[1:0] == 2'b01) begin
 // 1 pixel outline of visible area (white)
 hs <= hsync;
 vs <= vsync;
 b <= blank;
```

```
 rgb <= (hcount==0 | hcount==1023 | vcount==0 | vcount==767) ? 7 : 0;
      end else if (switch[1:0] == 2'b10) begin
 // color bars
 hs <= hsync;
 vs <= vsync;
 b <= blank;
 rgb <= hcount[8:6];
      end else begin
         // default: pong
 hs <= phsync;
 vs <= pvsync;
 b <= pblank;
 rgb <= pixel;
      end
   end

   // VGA Output.  In order to meet the setup and hold times of the
   // AD7125, we send it ~clock_65mhz.
   assign vga_out_red = {8{rgb[2]}};
   assign vga_out_green = {8{rgb[1]}};
   assign vga_out_blue = {8{rgb[0]}};
   assign vga_out_sync_b = 1'b1;    // not used
   assign vga_out_blank_b = ~b;
   assign vga_out_pixel_clock = ~clock_65mhz;
   assign vga_out_hsync = hs;
   assign vga_out_vsync = vs;

   assign led = ~{3'b000,up,down,reset,switch[1:0]};

wire [10:0] paddleminX_IN,paddlemaxX_IN;
wire [9:0] paddleminY_IN,paddlemaxY_IN;


control_paddle mycontrol_paddle(up,down,right,left,paddleminX_IN,paddleminY_IN,paddlemaxX_IN,pa


gamelogic_lump zachpart(reset,logic_clock,clock_65mhz, switch[7:4],switch[3],switch[2],
vsync, hcount,vcount, pixel, blank, rs232_txd, rs232_rxd, paddleminX_IN, paddleminY_IN,paddlema



endmodule
```

A.20 gamelogic_lump.v

```
module gamelogic_lump(reset,logic_clock,clock_65mhz, initial_velocity,
single_player,master_player,
vsync, slow_vsync,hcount,vcount, pixel, rs232_txd, rs232_rxd,
paddleminX_IN, paddleminY_IN,paddlemaxX_IN,paddlemaxY_IN,
paddle_x_change_LOCAL, paddle_y_change_LOCAL, paddle_z_change_LOCAL);

input reset,logic_clock,clock_65mhz,single_player,master_player,vsync, rs232_rxd;
input [10:0] paddleminX_IN, paddlemaxX_IN, hcount;
input [9:0] paddleminY_IN, paddlemaxY_IN, vcount;
input signed [10:0] paddle_x_change_LOCAL, paddle_y_change_LOCAL, paddle_z_change_LOCAL;

wire signed [10:0] paddle_x_change_MINE, paddle_y_change_MINE, paddle_z_change_MINE;
wire signed [10:0] paddle_x_change_OPP, paddle_y_change_OPP, paddle_z_change_OPP;

input [3:0] initial_velocity;
//input [7:0] angle;
input slow_vsync;

output rs232_txd;
output [2:0] pixel;

wire fast_reset, buffered_reset;

//This is mainly to ease the strain on the router

synchronizer reset_sync(logic_clock, clock_65mhz, reset,fast_reset);

synchronizer reset_buffer(logic_clock, logic_clock, reset, buffered_reset);




//Wires to hold ball coordinates and paddle coordinates
//IN means from this labkit
//MINE means the one I should show (depends on singleplayer/multiplayer)
//OPP means the opponent's information
//LOCAL means within this module.  Depending on multiplayer settings this is either ignored or

wire [10:0] ballpixelX, paddleminX_IN,paddlemaxX_IN,paddleminX_MINE,paddlemaxX_MINE, paddleminX
wire [9:0] ballpixelY, paddleminY_IN,paddlemaxY_IN,paddleminY_MINE,paddlemaxY_MINE, paddleminY_
wire [10:0] ballsize;
wire [2:0] ball_color;
```

```
wire signed [12:0] ballX, ballX_LOCAL;
wire signed [12:0] ballY, ballY_LOCAL;
wire signed [12:0] ballZ, ballZ_LOCAL;


// wire signed [12:0] ballX_opp,ballY_opp,ballZ_opp;
wire [10:0] ballpixelX_opp;
wire [9:0] ballpixelY_opp;
wire [10:0] ballsize_opp;
wire [2:0] ball_color_opp;

//Flippling ball coordinates because the opponent is a mirror image
// assign ballX_opp = 1023 - ballX;
// assign ballY_opp = 4096 - ballY;
// assign ballZ_opp = ballZ;

//Holds whether a wall should blink because it was hit
wire [3:0] hit_wall;
wire [3:0] hit_wall_buf, hit_wall_in;

//Game logic for calculating X,Y,Z of ball and managing paddle collisions
//In multiplayer it is also responsible for the opponent's paddle.
game_logic pg(logic_clock,buffered_reset,initial_velocity, //angle,
paddle_x_change_MINE[10:1], paddle_y_change_MINE[10:1], paddle_z_change_MINE[10:1],
paddle_x_change_OPP[10:1], paddle_y_change_OPP[10:1], paddle_z_change_OPP[10:1],
  slow_vsync, ballX_LOCAL, ballY_LOCAL, ballZ_LOCAL,
  ballpixelX, ballpixelY, ballpixelX_opp, ballpixelY_opp, paddleminX_MINE,
  paddleminY_MINE, paddlemaxX_MINE, paddlemaxY_MINE, paddleminX_OPP,
  paddleminY_OPP, paddlemaxX_OPP, paddlemaxY_OPP, hit_wall_in);


//Updates at RISING edge of vsync
//Projects the X,Y,Z of ball to X,Y on screen
projectball myprojectball(ballX,ballY,ballZ,ballpixelX,
ballpixelY,ballsize,ball_color,logic_clock,slow_vsync,reset);

//Render_screen is running on 65 mhz, so I have to worry about the interface to logic_clock (27
//Render_screen samples at the falling edge of vsync (synchronous to 65 mhz clock)
//I will make inputs to render screen transition only at rising edges of vsync in order to avoi
//I also have to look out for inputs to projectball.  It will only sample at that point too.

wire [10:0] ballpixelX_buf, paddleminX_buf, paddlemaxX_buf;
wire [9:0] ballpixelY_buf, paddleminY_buf, paddlemaxY_buf;
wire [10:0] ballsize_buf;
```

```
wire [2:0] ball_color_buf;


//This buffers the inputs going to renderscreen from their 27 MHz versions to 65 MHz versions
render_screen_input_buffer bufferlump(clock_65mhz, logic_clock, ballpixelX, ballpixelY, ballsiz
 ballpixelX_buf, ballpixelY_buf, ballsize_buf, ball_color_buf, paddleminX_buf, paddleminY_buf,


//Interfaces with screen to output pixels.  It does no computation except placing pixels on scr
render_screen rs(clock_65mhz,fast_reset,
  hcount,vcount,hsync,vsync,
  pixel, ballpixelX_buf, ballpixelY_buf, ballsize_buf,ball_color_buf, paddleminX_buf, paddlemin
  //paddleminX_OPP,paddleminY_OPP,paddlemaxX_OPP,paddlemaxY_OPP,ballpixelX_opp,ballpixelY_opp)



//Updates outputs on RISING edge of vsync
//Implements multiplayer mode.  Most signals go through here and are given to either the other
//one.
multiplayer mymultiplayer(logic_clock,slow_vsync,rs232_txd,rs232_rxd,reset,
paddleminX_IN,paddleminY_IN,paddlemaxX_IN,paddlemaxY_IN,
paddleminX_MINE,paddleminY_MINE,paddlemaxX_MINE,paddlemaxY_MINE,
paddleminX_OPP,paddleminY_OPP,paddlemaxX_OPP,paddlemaxY_OPP,
ballpixelX_opp, ballpixelY_opp, ballpixelX, ballpixelY,
ballX_LOCAL,ballY_LOCAL,ballZ_LOCAL,ballX,ballY,ballZ,
single_player,master_player, hit_wall_in, hit_wall,
paddle_x_change_LOCAL, paddle_y_change_LOCAL, paddle_z_change_LOCAL,
paddle_x_change_MINE, paddle_y_change_MINE, paddle_z_change_MINE,
paddle_x_change_OPP, paddle_y_change_OPP, paddle_z_change_OPP);




endmodule
```

A.21 get_extrema.v

```
‘timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    17:08:30 11/15/2007
// Design Name:
```

```
// Module Name:    get_extrema
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module get_extrema(clk, rst, data, vsync, extreme);
    input clk;
    input rst;
    input [9:0] data;
 input vsync;
    output reg [9:0] extreme;

 parameter MAX = 1;//if MAX = 0, this block will minimize
 parameter SYNC = 1;
 parameter READ = 0;
 parameter FIRSTLINE = 1;
 parameter OTHER = 0;
 parameter DEFAULT = MAX ? 0 : 10'h3ff;

reg [9:0] extreme_temp;
reg state;
/*
always @ (posedge clk)
begin
if (rst)
extreme_temp <= MAX ? 0 : 10'h3ff;
else begin
if (state == READ) begin
if (vsync == 1) begin
extreme <= extreme_temp;
state   <= SYNC;
extreme_temp <= MAX ? 0 : 10'h3ff;
end
else
extreme_temp <= (MAX ? (data > extreme_temp) : (data < extreme_temp)) ?
data : extreme_temp;
end
```

```
else begin
if (~vsync) begin
extreme_temp <= (MAX ? (data > extreme_temp) : (data < extreme_temp)) ?
data : extreme_temp;
state <= READ;
end
end
end
end*/
/*
//vsync is now firstline
always @ (posedge clk)
begin
if (rst)
extreme_temp <= DEFAULT;
else begin
if (state == OTHER) begin
if (vsync == 1) begin
//first pixel of first line is ignored
extreme <= extreme_temp;
state   <= FIRSTLINE;
extreme_temp <= DEFAULT;
end
else
extreme_temp <= (MAX ? (data > extreme_temp) : (data < extreme_temp)) ?
data : extreme_temp;
end
else begin
extreme_temp <= (MAX ? (data > extreme_temp) : (data < extreme_temp)) ?
data : extreme_temp;
if (~vsync) begin
state <= OTHER;
end
end
end
end*/

always @ (posedge clk)
begin
if (rst)
extreme_temp <= DEFAULT;
else begin
if (vsync == 1) begin
extreme <= extreme_temp;
extreme_temp <= DEFAULT;
```

```
end
else begin
if (MAX)
extreme_temp <= (data > extreme_temp) ? data : extreme_temp;
//if (data > extreme_temp)
// extreme_temp <= data;
else
extreme_temp <= (data > extreme_temp) ? extreme_temp : data;
//extreme_temp <= (MAX ? (data > extreme_temp) : (data < extreme_temp)) ?
// data : extreme_temp;
//if (~(data > extreme_temp))
//extreme_temp <= data;
//end
end
end
end

endmodule
```

A.22 get_extrema_signed.v

```
‘timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    15:59:48 11/19/2007
// Design Name:
// Module Name:    get_extrema_signed
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module get_extrema_signed(clk, rst, x, y, data, vsync, best_x, best_y);
    input clk;
    input rst;
 input [9:0] x;
```

```
 input [9:0] y;
    input signed [11:0] data;
 input vsync;
    output reg [9:0] best_x;
 output reg [9:0] best_y;

 parameter MAX = 1;//if MAX = 0, this block will minimize
 parameter SYNC = 1;
 parameter READ = 0;
 parameter FIRSTLINE = 1;
 parameter OTHER = 0;
 parameter DEFAULT = MAX ? 12'h800 : 12'h7ff;

reg signed [11:0] extreme;
reg [9:0] best_x_temp;
reg [9:0] best_y_temp;

always @ (posedge clk)
begin
if (rst)
extreme <= DEFAULT;
else begin
if (vsync == 1) begin
best_x <= best_x_temp;
best_y <= best_y_temp;
extreme <= DEFAULT;
end
else begin
if (MAX) begin
if (data > extreme) begin
extreme <= data;
best_x_temp <= x;
best_y_temp <= y;
end
end
else begin
if (~(data > extreme)) begin
extreme <= data;
best_x_temp <= x;
best_y_temp <= y;
end
end
end
//extreme_temp <= (MAX ? (data > extreme_temp) : (data < extreme_temp)) ?
// data : extreme_temp;
```

```
end
end

endmodule
```

A.23 isPaddle.v

```
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Mark Stevens
//
// Create Date:
// Design Name:
// Module Name:    isPaddle
// Project Name:
// Target Devices:
// Tool versions:
// Description: Determine whether or not a pixel is part of the paddle, using
// thresholding of y, cr, cb. Default values that have been determined to work
// well are automatically loaded, but the threshholds can be changed using
// switches on the labkit.
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////



module isPaddle(clk, rst, ycrcb_in, sw1, sw2, sw7, sw6, sw5, sw4, sw3, up, down, paddle);
input clk;
input rst;
input [29:0] ycrcb_in;
input sw1;//controls whether you are setting the cr or cb threshhold
//0 for cr, 1 for cb
input sw2; //1 for upper limit, 0 for lower limit
input sw7; //1 for set y limits, 0 for cr/cb
input sw6; //bit 10 of thresh
input sw5; //bit 9 of thresh
input sw4; //bit 8 of thesh
input sw3; //bit 7 of thresh
input up; //not used yet; later will define finer controls of thresholds
```

```
input down; //not used yet; later will define finer controls of thresholds
output paddle; //1 for paddle, 0 for not paddle

reg [9:0] upper_thresh_cr;
reg [9:0] upper_thresh_cb;
reg [9:0] lower_thresh_cr;
reg [9:0] lower_thresh_cb;
reg [9:0] upper_thresh_y;
reg [9:0] lower_thresh_y;
wire [9:0] assigned_limit = {sw6, sw5, sw4, sw3, 6'b0};

reg cr_in_limit0;
reg cb_in_limit0;
reg cr_in_limit1;
reg cb_in_limit1;
reg y_in_limit0;
reg y_in_limit1;
reg [29:0] ycrcb;
//reg paddle;


always @ (posedge clk) begin
if (rst) begin
upper_thresh_cr <= {4'b1111, 6'h00}; //1111
upper_thresh_cb <= {4'b1010, 6'h00}; //1001 - 1011
lower_thresh_cr <= {4'b1011, 6'h00}; //1011
lower_thresh_cb <= 0;   //0000
lower_thresh_y  <= {4'b0100, 6'h00}; //0100 - 0101
upper_thresh_y  <= {4'b1100, 6'h00}; //1100
end
else begin
if (down) begin
if (sw7) begin
case (sw2)
1'b0: lower_thresh_y <= assigned_limit;
1'b1: upper_thresh_y <= assigned_limit;
endcase
end
else begin
case ({sw1, sw2})
2'b00: lower_thresh_cr <= assigned_limit;
2'b01: upper_thresh_cr <= assigned_limit;
2'b10: lower_thresh_cb <= assigned_limit;
2'b11: upper_thresh_cb <= assigned_limit;
endcase
```

```
end
end
end
end

always @ (posedge clk) begin
   ycrcb <= ycrcb_in;
//paddle <= 1;
//paddle <= (ycrcb[29:20] > {3'b001, 7'h00});
cr_in_limit0 <= (ycrcb[19:10] > lower_thresh_cr);
cr_in_limit1 <= (ycrcb[19:10] < upper_thresh_cr);
cb_in_limit0 <= (ycrcb[9:0] > lower_thresh_cb);
cb_in_limit1 <= (ycrcb[9:0] < upper_thresh_cb);
y_in_limit0  <= (ycrcb[29:20] > lower_thresh_y);
y_in_limit1  <= (ycrcb[29:20] < upper_thresh_y);
end


//assign paddle = (cr_in_limit & cb_in_limit);
//assign paddle = 0;
assign paddle = (cr_in_limit0 & cb_in_limit0 &
  cr_in_limit1 & cb_in_limit1 &
  y_in_limit0  & y_in_limit1);

endmodule
```

A.24 labkit.ucf

```
##############################################################################
#
# 6.111 FPGA Labkit -- Constraints File
#
# For Labkit Revision 004
#
#
# Created: Oct 30, 2004 from revision 003 contraints file
# Author: Nathan Ickes and Isaac Cambron
#
##############################################################################
#
# CHANGES FOR BOARD REVISION 004
#
# 1) Added signals for logic analyzer pods 2-4.
```

```
# 2) Expanded tv_in_ycrcy bus to 20 bits.
# 3) Renamed tv_out_sclk to tv_out_i2c_clock for consistency
# 4) Renamed tv_out_data to tv_out_i2c_data for consistency
# 5) Moved flash_address<1> to AE14.
# 6) 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.
# 2) Moved SRAM feedback clock loop to FPGA pins AL28 (out) and AJ16 (in).
# 3) Moved rs232_rts to FPGA pin R3.
#
# CHANGES FOR BOARD REVISION 002
#
# 1) Moved ZBT_BANK1_CLK signal to pin Y9.
# 2) Moved user1<30> to J14.
# 3) Moved user3<29> to J13.
# 4) Added SRAM clock feedback loop between D15 and H15.
# 5) Renamed ram#_parity and ram#_we#_b signals.
# 6) Removed the constraint on "systemace_clock", since this net no longer
#    exists. The SystemACE clock is now hardwired to the 27MHz oscillator
#    on the PCB.
#
##############################################################################
#
# Complete change history (including bug fixes)
#
# 2005-Sep-09: Added missing IOSTANDARD attribute to "disp_data_out".
#
# 2005-Jan-23: Added a pullup to FLASH_STS
#
# 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.)
#
# 2005-Jan-23: Change history started.
#
##############################################################################
#
# Audio CODEC
#
NET "beep"           LOC="ac19" | IOSTANDARD=LVDCI_33;
NET "audio_reset_b"  LOC="ae18" | IOSTANDARD=LVTTL;
NET "ac97_sdata_out" LOC="ac18" | IOSTANDARD=LVDCI_33;
```

```
NET "ac97_sdata_in" LOC = "aj24";
NET "ac97_synch"     LOC="ac17" | IOSTANDARD=LVDCI_33;
NET "ac97_bit_clock" LOC = "ah24";
#
# VGA Output
#
NET "vga_out_red<7>" LOC="ae9"  | IOSTANDARD=LVTTL;
NET "vga_out_red<6>" LOC="ae8"  | IOSTANDARD=LVTTL;
NET "vga_out_red<5>" LOC="ad12" | IOSTANDARD=LVTTL;
NET "vga_out_red<4>" LOC="af8"  | IOSTANDARD=LVTTL;
NET "vga_out_red<3>" LOC="af9"  | IOSTANDARD=LVTTL;
NET "vga_out_red<2>" LOC="ag9"  | IOSTANDARD=LVTTL;
NET "vga_out_red<1>" LOC="ag10" | IOSTANDARD=LVTTL;
NET "vga_out_red<0>" LOC="af11" | IOSTANDARD=LVTTL;
NET "vga_out_green<7>" LOC="ah8"  | IOSTANDARD=LVTTL;
NET "vga_out_green<6>" LOC="ah7"  | IOSTANDARD=LVTTL;
NET "vga_out_green<5>" LOC="aj6"  | IOSTANDARD=LVTTL;
NET "vga_out_green<4>" LOC="ah6"  | IOSTANDARD=LVTTL;
NET "vga_out_green<3>" LOC="ad15" | IOSTANDARD=LVTTL;
NET "vga_out_green<2>" LOC="ac14" | IOSTANDARD=LVTTL;
NET "vga_out_green<1>" LOC="ag8"  | IOSTANDARD=LVTTL;
NET "vga_out_green<0>" LOC="ac12" | IOSTANDARD=LVTTL;
NET "vga_out_blue<7>" LOC="ag15" | IOSTANDARD=LVTTL;
NET "vga_out_blue<6>" LOC="ag14" | IOSTANDARD=LVTTL;
NET "vga_out_blue<5>" LOC="ag13" | IOSTANDARD=LVTTL;
NET "vga_out_blue<4>" LOC="ag12" | IOSTANDARD=LVTTL;
NET "vga_out_blue<3>" LOC="aj11" | IOSTANDARD=LVTTL;
NET "vga_out_blue<2>" LOC="ah11" | IOSTANDARD=LVTTL;
NET "vga_out_blue<1>" LOC="aj10" | IOSTANDARD=LVTTL;
NET "vga_out_blue<0>" LOC="ah9"  | IOSTANDARD=LVTTL;
NET "vga_out_sync_b"      LOC="aj9"  | IOSTANDARD=LVTTL;
NET "vga_out_blank_b"     LOC="aj8"  | IOSTANDARD=LVTTL;
NET "vga_out_pixel_clock" LOC="ac10" | IOSTANDARD=LVTTL;
NET "vga_out_hsync"       LOC="ac13" | IOSTANDARD=LVTTL;
NET "vga_out_vsync"       LOC="ac11" | IOSTANDARD=LVTTL;
#
# Video Output
#
NET "tv_out_ycrcb<9>" LOC="p27" | IOSTANDARD=LVDCI_33;
NET "tv_out_ycrcb<8>" LOC="r27" | IOSTANDARD=LVDCI_33;
NET "tv_out_ycrcb<7>" LOC="t29" | IOSTANDARD=LVDCI_33;
NET "tv_out_ycrcb<6>" LOC="h26" | IOSTANDARD=LVDCI_33;
NET "tv_out_ycrcb<5>" LOC="j26" | IOSTANDARD=LVDCI_33;
NET "tv_out_ycrcb<4>" LOC="l26" | IOSTANDARD=LVDCI_33;
NET "tv_out_ycrcb<3>" LOC="m26" | IOSTANDARD=LVDCI_33;
```

```
NET "tv_out_ycrcb<2>" LOC="n26" | IOSTANDARD=LVDCI_33;
NET "tv_out_ycrcb<1>" LOC="p26" | IOSTANDARD=LVDCI_33;
NET "tv_out_ycrcb<0>" LOC="r26" | IOSTANDARD=LVDCI_33;
NET "tv_out_reset_b" LOC="g27" | IOSTANDARD=LVDCI_33;
NET "tv_out_clock" LOC="l27" | IOSTANDARD=LVDCI_33;
NET "tv_out_i2c_clock" LOC="j27" | IOSTANDARD=LVDCI_33;
NET "tv_out_i2c_data" LOC="h27" | IOSTANDARD=LVDCI_33;
NET "tv_out_pal_ntsc" LOC="j25" | IOSTANDARD=LVDCI_33;
NET "tv_out_hsync_b" LOC="n27" | IOSTANDARD=LVDCI_33;
NET "tv_out_vsync_b" LOC="m27" | IOSTANDARD=LVDCI_33;
NET "tv_out_blank_b" LOC="h25" | IOSTANDARD=LVDCI_33;
NET "tv_out_subcar_reset" LOC="k27" | IOSTANDARD=LVDCI_33;
#
# Video Input
#
NET "tv_in_ycrcb<19>" LOC = "ag17";
NET "tv_in_ycrcb<18>" LOC = "ag18";
NET "tv_in_ycrcb<17>" LOC = "ag19";
NET "tv_in_ycrcb<16>" LOC = "ag20";
NET "tv_in_ycrcb<15>" LOC = "ae20";
NET "tv_in_ycrcb<14>" LOC = "af21";
NET "tv_in_ycrcb<13>" LOC = "ad20";
NET "tv_in_ycrcb<12>" LOC = "ag23";
NET "tv_in_ycrcb<11>" LOC = "aj26";
NET "tv_in_ycrcb<10>" LOC = "ah26";
NET "tv_in_ycrcb<9>" LOC = "w23";
NET "tv_in_ycrcb<8>" LOC = "v23";
NET "tv_in_ycrcb<7>" LOC = "u23";
NET "tv_in_ycrcb<6>" LOC = "t23";
NET "tv_in_ycrcb<5>" LOC = "t26";
NET "tv_in_ycrcb<4>" LOC = "t24";
NET "tv_in_ycrcb<3>" LOC = "r25";
NET "tv_in_ycrcb<2>" LOC = "l30";
NET "tv_in_ycrcb<1>" LOC = "m31";
NET "tv_in_ycrcb<0>" LOC = "m30";
NET "tv_in_data_valid" LOC = "ah25";
NET "tv_in_line_clock1" LOC="ad16" | IOSTANDARD=LVDCI_33;
NET "tv_in_line_clock2" LOC="ad17" | IOSTANDARD=LVDCI_33;
NET "tv_in_aef" LOC = "aj23";
NET "tv_in_hff" LOC = "ah23";
NET "tv_in_aff" LOC = "aj22";
NET "tv_in_i2c_clock" LOC="ad21" | IOSTANDARD=LVDCI_33;
NET "tv_in_i2c_data" LOC="ad19" | IOSTANDARD=LVDCI_33;
NET "tv_in_fifo_read" LOC="ac22" | IOSTANDARD=LVDCI_33;
NET "tv_in_fifo_clock" LOC="ag22" | IOSTANDARD=LVDCI_33;
```

```
NET "tv_in_iso" LOC="aj27" | IOSTANDARD=LVDCI_33;
NET "tv_in_reset_b" LOC="ag25" | IOSTANDARD=LVDCI_33;
NET "tv_in_clock" LOC="ab21" | IOSTANDARD=LVDCI_33;
#
# SRAMs
#
NET "ram0_data<35>" LOC="ab25" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<34>" LOC="ah29" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<33>" LOC="ag28" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<32>" LOC="ag29" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<31>" LOC="af27" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<30>" LOC="af29" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<29>" LOC="af28" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<28>" LOC="ae28" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<27>" LOC="ad25" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<26>" LOC="aa25" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<25>" LOC="ah30" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<24>" LOC="ah31" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<23>" LOC="ag30" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<22>" LOC="ag31" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<21>" LOC="af30" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<20>" LOC="af31" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<19>" LOC="ae30" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<18>" LOC="ae31" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<17>" LOC="y27"  | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<16>" LOC="aa28" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<15>" LOC="y29"  | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<14>" LOC="y28"  | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<13>" LOC="w29"  | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<12>" LOC="w28"  | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<11>" LOC="v28"  | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<10>" LOC="u29"  | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<9>"  LOC="u28"  | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<8>"  LOC="aa27" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<7>"  LOC="ad31" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<6>"  LOC="ac30" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<5>"  LOC="ac31" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<4>"  LOC="ab30" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<3>"  LOC="ab31" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<2>"  LOC="aa30" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<1>"  LOC="aa31" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<0>"  LOC="y30"  | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_address<18>" LOC="v31" | IOSTANDARD=LVDCI_33;
NET "ram0_address<17>" LOC="w31" | IOSTANDARD=LVDCI_33;
NET "ram0_address<16>" LOC="ad28" | IOSTANDARD=LVDCI_33;
```

```
NET "ram0_address<15>" LOC="ad29" | IOSTANDARD=LVDCI_33;
NET "ram0_address<14>" LOC="ac24" | IOSTANDARD=LVDCI_33;
NET "ram0_address<13>" LOC="ad26" | IOSTANDARD=LVDCI_33;
NET "ram0_address<12>" LOC="ad27" | IOSTANDARD=LVDCI_33;
NET "ram0_address<11>" LOC="ac27" | IOSTANDARD=LVDCI_33;
NET "ram0_address<10>" LOC="ab27" | IOSTANDARD=LVDCI_33;
NET "ram0_address<9>" LOC="y31" | IOSTANDARD=LVDCI_33;
NET "ram0_address<8>" LOC="w30" | IOSTANDARD=LVDCI_33;
NET "ram0_address<7>" LOC="y26" | IOSTANDARD=LVDCI_33;
NET "ram0_address<6>" LOC="y25" | IOSTANDARD=LVDCI_33;
NET "ram0_address<5>" LOC="ab24" | IOSTANDARD=LVDCI_33;
NET "ram0_address<4>" LOC="ac25" | IOSTANDARD=LVDCI_33;
NET "ram0_address<3>" LOC="aa26" | IOSTANDARD=LVDCI_33;
NET "ram0_address<2>" LOC="aa24" | IOSTANDARD=LVDCI_33;
NET "ram0_address<1>" LOC="ab29" | IOSTANDARD=LVDCI_33;
NET "ram0_address<0>" LOC="ac26" | IOSTANDARD=LVDCI_33;
NET "ram0_adv_ld" LOC="v26" | IOSTANDARD=LVDCI_33;
NET "ram0_clk" LOC="u30" | IOSTANDARD=LVDCI_33;
NET "ram0_cen_b" LOC="u25" | IOSTANDARD=LVDCI_33;
NET "ram0_ce_b" LOC="w26" | IOSTANDARD=LVDCI_33;
NET "ram0_oe_b" LOC="v25" | IOSTANDARD=LVDCI_33;
NET "ram0_we_b" LOC="u31" | IOSTANDARD=LVDCI_33;
NET "ram0_bwe_b<0>" LOC="v27" | IOSTANDARD=LVDCI_33;
NET "ram0_bwe_b<1>" LOC="u27" | IOSTANDARD=LVDCI_33;
NET "ram0_bwe_b<2>" LOC="w27" | IOSTANDARD=LVDCI_33;
NET "ram0_bwe_b<3>" LOC="u26" | IOSTANDARD=LVDCI_33;
NET "ram1_data<35>" LOC="aa9" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<34>" LOC="ah2" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<33>" LOC="ah1" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<32>" LOC="ag2" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<31>" LOC="ag1" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<30>" LOC="af2" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<29>" LOC="af1" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<28>" LOC="ae2" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<27>" LOC="ae1" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<26>" LOC="ab9" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<25>" LOC="ah3" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<24>" LOC="ag4" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<23>" LOC="ag3" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<22>" LOC="af4" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<21>" LOC="af3" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<20>" LOC="ae4" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<19>" LOC="ae5" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<18>" LOC="ad5" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<17>" LOC="v2"  | IOSTANDARD=LVDCI_33 | NODELAY;
```

```
NET "ram1_data<16>" LOC="ad1" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<15>" LOC="ac2" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<14>" LOC="ac1" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<13>" LOC="ab2" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<12>" LOC="ab1" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<11>" LOC="aa2" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<10>" LOC="aa1" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<9>"  LOC="y2"  | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<8>"  LOC="v4"  | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<7>"  LOC="ac3" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<6>"  LOC="ac4" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<5>"  LOC="aa5" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<4>"  LOC="aa3" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<3>"  LOC="aa4" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<2>"  LOC="y3"  | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<1>"  LOC="y4"  | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<0>"  LOC="w3"  | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_address<18>" LOC="ab3" | IOSTANDARD=LVDCI_33;
NET "ram1_address<17>" LOC="ac5" | IOSTANDARD=LVDCI_33;
NET "ram1_address<16>" LOC="u6" | IOSTANDARD=LVDCI_33;
NET "ram1_address<15>" LOC="v6" | IOSTANDARD=LVDCI_33;
NET "ram1_address<14>" LOC="w6" | IOSTANDARD=LVDCI_33;
NET "ram1_address<13>" LOC="y6" | IOSTANDARD=LVDCI_33;
NET "ram1_address<12>" LOC="aa7" | IOSTANDARD=LVDCI_33;
NET "ram1_address<11>" LOC="ab7" | IOSTANDARD=LVDCI_33;
NET "ram1_address<10>" LOC="ac6" | IOSTANDARD=LVDCI_33;
NET "ram1_address<9>" LOC="ad3" | IOSTANDARD=LVDCI_33;
NET "ram1_address<8>" LOC="ad4" | IOSTANDARD=LVDCI_33;
NET "ram1_address<7>" LOC="u3" | IOSTANDARD=LVDCI_33;
NET "ram1_address<6>" LOC="w4" | IOSTANDARD=LVDCI_33;
NET "ram1_address<5>" LOC="ac8" | IOSTANDARD=LVDCI_33;
NET "ram1_address<4>" LOC="ab8" | IOSTANDARD=LVDCI_33;
NET "ram1_address<3>" LOC="aa8" | IOSTANDARD=LVDCI_33;
NET "ram1_address<2>" LOC="y7" | IOSTANDARD=LVDCI_33;
NET "ram1_address<1>" LOC="y8" | IOSTANDARD=LVDCI_33;
NET "ram1_address<0>" LOC="ad7" | IOSTANDARD=LVDCI_33;
NET "ram1_adv_ld" LOC="y5" | IOSTANDARD=LVDCI_33;
NET "ram1_clk" LOC="y9" | IOSTANDARD=LVDCI_33;
NET "ram1_cen_b" LOC="v5" | IOSTANDARD=LVDCI_33;
NET "ram1_ce_b" LOC="u4" | IOSTANDARD=LVDCI_33;
NET "ram1_oe_b" LOC="w5" | IOSTANDARD=LVDCI_33;
NET "ram1_we_b" LOC="aa6" | IOSTANDARD=LVDCI_33;
NET "ram1_bwe_b<0>" LOC="u2" | IOSTANDARD=LVDCI_33;
NET "ram1_bwe_b<1>" LOC="u1" | IOSTANDARD=LVDCI_33;
NET "ram1_bwe_b<2>" LOC="v1" | IOSTANDARD=LVDCI_33;
```

```
NET "ram1_bwe_b<3>" LOC="u5" | IOSTANDARD=LVDCI_33;
NET "clock_feedback_out" LOC="al28" | IOSTANDARD=LVDCI_33;
NET "clock_feedback_in" LOC = "aj16";
#
# Flash
#
NET "flash_data<15>" LOC="ak10" | IOSTANDARD=LVTTL;
NET "flash_data<14>" LOC="ak11" | IOSTANDARD=LVTTL;
NET "flash_data<13>" LOC="ak12" | IOSTANDARD=LVTTL;
NET "flash_data<12>" LOC="ak13" | IOSTANDARD=LVTTL;
NET "flash_data<11>" LOC="ak14" | IOSTANDARD=LVTTL;
NET "flash_data<10>" LOC="ak15" | IOSTANDARD=LVTTL;
NET "flash_data<9>"  LOC="ah12" | IOSTANDARD=LVTTL;
NET "flash_data<8>"  LOC="ah13" | IOSTANDARD=LVTTL;
NET "flash_data<7>"  LOC="al10" | IOSTANDARD=LVTTL;
NET "flash_data<6>"  LOC="al11" | IOSTANDARD=LVTTL;
NET "flash_data<5>"  LOC="al12" | IOSTANDARD=LVTTL;
NET "flash_data<4>"  LOC="al13" | IOSTANDARD=LVTTL;
NET "flash_data<3>"  LOC="al14" | IOSTANDARD=LVTTL;
NET "flash_data<2>"  LOC="al15" | IOSTANDARD=LVTTL;
NET "flash_data<1>"  LOC="aj12" | IOSTANDARD=LVTTL;
NET "flash_data<0>"  LOC="aj13" | IOSTANDARD=LVTTL;
NET "flash_address<24>" LOC="al7";
NET "flash_address<23>" LOC = "aj15";
NET "flash_address<22>" LOC = "al25";
NET "flash_address<21>" LOC = "ak23";
NET "flash_address<20>" LOC = "al23";
NET "flash_address<19>" LOC = "ak22";
NET "flash_address<18>" LOC = "al22";
NET "flash_address<17>" LOC = "ak21";
NET "flash_address<16>" LOC = "al21";
NET "flash_address<15>" LOC = "ak20";
NET "flash_address<14>" LOC = "al20";
NET "flash_address<13>" LOC = "ak19";
NET "flash_address<12>" LOC = "al19";
NET "flash_address<11>" LOC = "al18";
NET "flash_address<10>" LOC = "ak17";
NET "flash_address<9>" LOC = "al17";
NET "flash_address<8>" LOC = "ah21";
NET "flash_address<7>" LOC = "aj20";
NET "flash_address<6>" LOC = "ah20";
NET "flash_address<5>" LOC = "aj19";
NET "flash_address<4>" LOC = "ah19";
NET "flash_address<3>" LOC = "ah18";
NET "flash_address<2>" LOC = "aj17";
```

```
NET "flash_address<1>" LOC = "ae14";
NET "flash_address<0>" LOC = "ah14";
NET "flash_ce_b" LOC="aj21" | IOSTANDARD=LVDCI_33;
NET "flash_oe_b" LOC="ak9" | IOSTANDARD=LVDCI_33;
NET "flash_we_b" LOC="al8" | IOSTANDARD=LVDCI_33;
NET "flash_reset_b" LOC="ak18" | IOSTANDARD=LVDCI_33;
NET "flash_sts" LOC="al9" | PULLUP;
NET "flash_byte_b" LOC="ah15" | IOSTANDARD=LVDCI_33;
#
# RS-232
#
NET "rs232_txd" LOC="p4" | IOSTANDARD=LVDCI_33;
NET "rs232_rxd" LOC = "p6";
NET "rs232_rts" LOC="r3" | IOSTANDARD=LVDCI_33;
NET "rs232_cts" LOC = "n8";
#
# Mouse and Keyboard
#
NET "mouse_clock" LOC = "ac16";
NET "mouse_data" LOC = "ac15";
NET "keyboard_clock" LOC = "ag16";
NET "keyboard_data" LOC = "af16";
#
# Clocks
#
NET "clock_27mhz" LOC = "c16";
NET "clock1" LOC = "h16";
NET "clock2" LOC = "c15";
#
# Alphanumeric Display
#
NET "disp_blank" LOC="af12" | IOSTANDARD=LVDCI_33;
NET "disp_data_in" LOC = "ae12";
NET "disp_clock" LOC="af14" | IOSTANDARD=LVDCI_33;
NET "disp_rs" LOC="af15" | IOSTANDARD=LVDCI_33;
NET "disp_ce_b" LOC="af13" | IOSTANDARD=LVDCI_33;
NET "disp_reset_b" LOC="ag11" | IOSTANDARD=LVDCI_33;
NET "disp_data_out" LOC="ae15" | IOSTANDARD=LVDCI_33;
#
# Buttons and Switches
#
NET "button0" LOC = "ae11";
NET "button1" LOC = "ae10";
NET "button2" LOC = "ad11";
NET "button3" LOC = "ab12";
```

```
NET "button_enter" LOC = "ak7";
NET "button_right" LOC = "al6";
NET "button_left" LOC = "al5";
NET "button_up" LOC = "al4";
NET "button_down" LOC = "ak6";
NET "switch<7>" LOC = "ad22";
NET "switch<6>" LOC = "ae23";
NET "switch<5>" LOC = "ac20";
NET "switch<4>" LOC = "ab20";
NET "switch<3>" LOC = "ac21";
NET "switch<2>" LOC = "ak25";
NET "switch<1>" LOC = "al26";
NET "switch<0>" LOC = "ak26";
#
# Discrete LEDs
#
NET "led<7>" LOC="ae17" | IOSTANDARD=LVTTL;
NET "led<6>" LOC="af17" | IOSTANDARD=LVTTL;
NET "led<5>" LOC="af18" | IOSTANDARD=LVTTL;
NET "led<4>" LOC="af19" | IOSTANDARD=LVTTL;
NET "led<3>" LOC="af20" | IOSTANDARD=LVTTL;
NET "led<2>" LOC="ag21" | IOSTANDARD=LVTTL;
NET "led<1>" LOC="ae21" | IOSTANDARD=LVTTL;
NET "led<0>" LOC="ae22" | IOSTANDARD=LVTTL;
#
# User Pins
#
NET "user1<31>" LOC="j15" | IOSTANDARD=LVTTL;
NET "user1<30>" LOC="j14" | IOSTANDARD=LVTTL;
NET "user1<29>" LOC="g15" | IOSTANDARD=LVTTL;
NET "user1<28>" LOC="f14" | IOSTANDARD=LVTTL;
NET "user1<27>" LOC="f12" | IOSTANDARD=LVTTL;
NET "user1<26>" LOC="h11" | IOSTANDARD=LVTTL;
NET "user1<25>" LOC="g9" | IOSTANDARD=LVTTL;
NET "user1<24>" LOC="h9" | IOSTANDARD=LVTTL;
NET "user1<23>" LOC="b15" | IOSTANDARD=LVTTL;
NET "user1<22>" LOC="b14" | IOSTANDARD=LVTTL;
NET "user1<21>" LOC="f15" | IOSTANDARD=LVTTL;
NET "user1<20>" LOC="e13" | IOSTANDARD=LVTTL;
NET "user1<19>" LOC="e11" | IOSTANDARD=LVTTL;
NET "user1<18>" LOC="e9" | IOSTANDARD=LVTTL;
NET "user1<17>" LOC="f8" | IOSTANDARD=LVTTL;
NET "user1<16>" LOC="f7" | IOSTANDARD=LVTTL;
NET "user1<15>" LOC="c13" | IOSTANDARD=LVTTL;
NET "user1<14>" LOC="c12" | IOSTANDARD=LVTTL;
```

```
NET "user1<13>" LOC="c11" | IOSTANDARD=LVTTL;
NET "user1<12>" LOC="c10" | IOSTANDARD=LVTTL;
NET "user1<11>" LOC="c9" | IOSTANDARD=LVTTL;
NET "user1<10>" LOC="c8" | IOSTANDARD=LVTTL;
NET "user1<9>" LOC="c6" | IOSTANDARD=LVTTL;
NET "user1<8>" LOC="e6" | IOSTANDARD=LVTTL;
NET "user1<7>" LOC="a11" | IOSTANDARD=LVTTL;
NET "user1<6>" LOC="a10" | IOSTANDARD=LVTTL;
NET "user1<5>" LOC="a9" | IOSTANDARD=LVTTL;
NET "user1<4>" LOC="a8" | IOSTANDARD=LVTTL;
NET "user1<3>" LOC="b6" | IOSTANDARD=LVTTL;
NET "user1<2>" LOC="b5" | IOSTANDARD=LVTTL;
NET "user1<1>" LOC="c5" | IOSTANDARD=LVTTL;
NET "user1<0>" LOC="b3" | IOSTANDARD=LVTTL;
NET "user2<31>" LOC="b27" | IOSTANDARD=LVTTL;
NET "user2<30>" LOC="b26" | IOSTANDARD=LVTTL;
NET "user2<29>" LOC="b25" | IOSTANDARD=LVTTL;
NET "user2<28>" LOC="a24" | IOSTANDARD=LVTTL;
NET "user2<27>" LOC="a23" | IOSTANDARD=LVTTL;
NET "user2<26>" LOC="a22" | IOSTANDARD=LVTTL;
NET "user2<25>" LOC="a21" | IOSTANDARD=LVTTL;
NET "user2<24>" LOC="a20" | IOSTANDARD=LVTTL;
NET "user2<23>" LOC="d26" | IOSTANDARD=LVTTL;
NET "user2<22>" LOC="d25" | IOSTANDARD=LVTTL;
NET "user2<21>" LOC="c24" | IOSTANDARD=LVTTL;
NET "user2<20>" LOC="d23" | IOSTANDARD=LVTTL;
NET "user2<19>" LOC="d21" | IOSTANDARD=LVTTL;
NET "user2<18>" LOC="d20" | IOSTANDARD=LVTTL;
NET "user2<17>" LOC="d19" | IOSTANDARD=LVTTL;
NET "user2<16>" LOC="d18" | IOSTANDARD=LVTTL;
NET "user2<15>" LOC="f24" | IOSTANDARD=LVTTL;
NET "user2<14>" LOC="f23" | IOSTANDARD=LVTTL;
NET "user2<13>" LOC="e22" | IOSTANDARD=LVTTL;
NET "user2<12>" LOC="e20" | IOSTANDARD=LVTTL;
NET "user2<11>" LOC="e18" | IOSTANDARD=LVTTL;
NET "user2<10>" LOC="e16" | IOSTANDARD=LVTTL;
NET "user2<9>" LOC="a19" | IOSTANDARD=LVTTL;
NET "user2<8>" LOC="a18" | IOSTANDARD=LVTTL;
NET "user2<7>" LOC="h22" | IOSTANDARD=LVTTL;
NET "user2<6>" LOC="g22" | IOSTANDARD=LVTTL;
NET "user2<5>" LOC="f21" | IOSTANDARD=LVTTL;
NET "user2<4>" LOC="f19" | IOSTANDARD=LVTTL;
NET "user2<3>" LOC="f17" | IOSTANDARD=LVTTL;
NET "user2<2>" LOC="h19" | IOSTANDARD=LVTTL;
NET "user2<1>" LOC="g20" | IOSTANDARD=LVTTL;
```

```
NET "user2<0>" LOC="h21" | IOSTANDARD=LVTTL;
NET "user3<31>" LOC="g12" | IOSTANDARD=LVTTL;
NET "user3<30>" LOC="h13" | IOSTANDARD=LVTTL;
NET "user3<29>" LOC="j13" | IOSTANDARD=LVTTL;
NET "user3<28>" LOC="g14" | IOSTANDARD=LVTTL;
NET "user3<27>" LOC="f13" | IOSTANDARD=LVTTL;
NET "user3<26>" LOC="f11" | IOSTANDARD=LVTTL;
NET "user3<25>" LOC="g10" | IOSTANDARD=LVTTL;
NET "user3<24>" LOC="h10" | IOSTANDARD=LVTTL;
NET "user3<23>" LOC="a15" | IOSTANDARD=LVTTL;
NET "user3<22>" LOC="a14" | IOSTANDARD=LVTTL;
NET "user3<21>" LOC="e15" | IOSTANDARD=LVTTL;
NET "user3<20>" LOC="e14" | IOSTANDARD=LVTTL;
NET "user3<19>" LOC="e12" | IOSTANDARD=LVTTL;
NET "user3<18>" LOC="e10" | IOSTANDARD=LVTTL;
NET "user3<17>" LOC="f9" | IOSTANDARD=LVTTL;
NET "user3<16>" LOC="g8" | IOSTANDARD=LVTTL;
NET "user3<15>" LOC="d14" | IOSTANDARD=LVTTL;
NET "user3<14>" LOC="d13" | IOSTANDARD=LVTTL;
NET "user3<13>" LOC="d12" | IOSTANDARD=LVTTL;
NET "user3<12>" LOC="d11" | IOSTANDARD=LVTTL;
NET "user3<11>" LOC="d9" | IOSTANDARD=LVTTL;
NET "user3<10>" LOC="d8" | IOSTANDARD=LVTTL;
NET "user3<9>" LOC="d7" | IOSTANDARD=LVTTL;
NET "user3<8>" LOC="d6" | IOSTANDARD=LVTTL;
NET "user3<7>" LOC="b12" | IOSTANDARD=LVTTL;
NET "user3<6>" LOC="b11" | IOSTANDARD=LVTTL;
NET "user3<5>" LOC="b10" | IOSTANDARD=LVTTL;
NET "user3<4>" LOC="b9" | IOSTANDARD=LVTTL;
NET "user3<3>" LOC="a7" | IOSTANDARD=LVTTL;
NET "user3<2>" LOC="a6" | IOSTANDARD=LVTTL;
NET "user3<1>" LOC="a5" | IOSTANDARD=LVTTL;
NET "user3<0>" LOC="a4" | IOSTANDARD=LVTTL;
NET "user4<31>" LOC="a28" | IOSTANDARD=LVTTL;
NET "user4<30>" LOC="a27" | IOSTANDARD=LVTTL;
NET "user4<29>" LOC="a26" | IOSTANDARD=LVTTL;
NET "user4<28>" LOC="a25" | IOSTANDARD=LVTTL;
NET "user4<27>" LOC="b23" | IOSTANDARD=LVTTL;
NET "user4<26>" LOC="b22" | IOSTANDARD=LVTTL;
NET "user4<25>" LOC="b21" | IOSTANDARD=LVTTL;
NET "user4<24>" LOC="b20" | IOSTANDARD=LVTTL;
NET "user4<23>" LOC="e25" | IOSTANDARD=LVTTL;
NET "user4<22>" LOC="c26" | IOSTANDARD=LVTTL;
NET "user4<21>" LOC="d24" | IOSTANDARD=LVTTL;
NET "user4<20>" LOC="c23" | IOSTANDARD=LVTTL;
```

```
NET "user4<19>" LOC="c22" | IOSTANDARD=LVTTL;
NET "user4<18>" LOC="c21" | IOSTANDARD=LVTTL;
NET "user4<17>" LOC="c20" | IOSTANDARD=LVTTL;
NET "user4<16>" LOC="c19" | IOSTANDARD=LVTTL;
NET "user4<15>" LOC="g24" | IOSTANDARD=LVTTL;
NET "user4<14>" LOC="e24" | IOSTANDARD=LVTTL;
NET "user4<13>" LOC="e23" | IOSTANDARD=LVTTL;
NET "user4<12>" LOC="e21" | IOSTANDARD=LVTTL;
NET "user4<11>" LOC="e19" | IOSTANDARD=LVTTL;
NET "user4<10>" LOC="e17" | IOSTANDARD=LVTTL;
NET "user4<9>" LOC="b19" | IOSTANDARD=LVTTL;
NET "user4<8>" LOC="b18" | IOSTANDARD=LVTTL;
NET "user4<7>" LOC="h23" | IOSTANDARD=LVTTL;
NET "user4<6>" LOC="g23" | IOSTANDARD=LVTTL;
NET "user4<5>" LOC="g21" | IOSTANDARD=LVTTL;
NET "user4<4>" LOC="f20" | IOSTANDARD=LVTTL;
NET "user4<3>" LOC="f18" | IOSTANDARD=LVTTL;
NET "user4<2>" LOC="f16" | IOSTANDARD=LVTTL;
NET "user4<1>" LOC="g18" | IOSTANDARD=LVTTL;
NET "user4<0>" LOC="g17" | IOSTANDARD=LVTTL;
#
# Daughter Card
#
NET "daughtercard<43>" LOC="L7" | IOSTANDARD=LVTTL;
NET "daughtercard<42>" LOC="H1" | IOSTANDARD=LVTTL;
NET "daughtercard<41>" LOC="J2" | IOSTANDARD=LVTTL;
NET "daughtercard<40>" LOC="J1" | IOSTANDARD=LVTTL;
NET "daughtercard<39>" LOC="K2" | IOSTANDARD=LVTTL;
NET "daughtercard<38>" LOC="M7" | IOSTANDARD=LVTTL;
NET "daughtercard<37>" LOC="M6" | IOSTANDARD=LVTTL;
NET "daughtercard<36>" LOC="M3" | IOSTANDARD=LVTTL;
NET "daughtercard<35>" LOC="M4" | IOSTANDARD=LVTTL;
NET "daughtercard<34>" LOC="L3" | IOSTANDARD=LVTTL;
NET "daughtercard<33>" LOC="K1" | IOSTANDARD=LVTTL;
NET "daughtercard<32>" LOC="L4" | IOSTANDARD=LVTTL;
NET "daughtercard<31>" LOC="K3" | IOSTANDARD=LVTTL;
NET "daughtercard<30>" LOC="K9" | IOSTANDARD=LVTTL;
NET "daughtercard<29>" LOC="L9" | IOSTANDARD=LVTTL;
NET "daughtercard<28>" LOC="K8" | IOSTANDARD=LVTTL;
NET "daughtercard<27>" LOC="K7" | IOSTANDARD=LVTTL;
NET "daughtercard<26>" LOC="L8" | IOSTANDARD=LVTTL;
NET "daughtercard<25>" LOC="L6" | IOSTANDARD=LVTTL;
NET "daughtercard<24>" LOC="M5" | IOSTANDARD=LVTTL;
NET "daughtercard<23>" LOC="N5" | IOSTANDARD=LVTTL;
NET "daughtercard<22>" LOC="P5" | IOSTANDARD=LVTTL;
```

```
NET "daughtercard<21>" LOC="D3" | IOSTANDARD=LVTTL;
NET "daughtercard<20>" LOC="E4" | IOSTANDARD=LVTTL;
NET "daughtercard<19>" LOC="E3" | IOSTANDARD=LVTTL;
NET "daughtercard<18>" LOC="F4" | IOSTANDARD=LVTTL;
NET "daughtercard<17>" LOC="F3" | IOSTANDARD=LVTTL;
NET "daughtercard<16>" LOC="G4" | IOSTANDARD=LVTTL;
NET "daughtercard<15>" LOC="H4" | IOSTANDARD=LVTTL;
NET "daughtercard<14>" LOC="J3" | IOSTANDARD=LVTTL;
NET "daughtercard<13>" LOC="J4" | IOSTANDARD=LVTTL;
NET "daughtercard<12>" LOC="D2" | IOSTANDARD=LVTTL;
NET "daughtercard<11>" LOC="D1" | IOSTANDARD=LVTTL;
NET "daughtercard<10>" LOC="E2" | IOSTANDARD=LVTTL;
NET "daughtercard<9>" LOC="E1" | IOSTANDARD=LVTTL;
NET "daughtercard<8>" LOC="F5" | IOSTANDARD=LVTTL;
NET "daughtercard<7>" LOC="G5" | IOSTANDARD=LVTTL;
NET "daughtercard<6>" LOC="H5" | IOSTANDARD=LVTTL;
NET "daughtercard<5>" LOC="J5" | IOSTANDARD=LVTTL;
NET "daughtercard<4>" LOC="K5" | IOSTANDARD=LVTTL;
NET "daughtercard<3>" LOC="H7" | IOSTANDARD=LVTTL;
NET "daughtercard<2>" LOC="J8" | IOSTANDARD=LVTTL;
NET "daughtercard<1>" LOC="J6" | IOSTANDARD=LVTTL;
NET "daughtercard<0>" LOC="J7" | IOSTANDARD=LVTTL;
#
# System Ace
#
NET "systemace_data<15>" LOC="F29" | IOSTANDARD=LVTTL;
NET "systemace_data<14>" LOC="G28" | IOSTANDARD=LVTTL;
NET "systemace_data<13>" LOC="H29" | IOSTANDARD=LVTTL;
NET "systemace_data<12>" LOC="H28" | IOSTANDARD=LVTTL;
NET "systemace_data<11>" LOC="J29" | IOSTANDARD=LVTTL;
NET "systemace_data<10>" LOC="J28" | IOSTANDARD=LVTTL;
NET "systemace_data<9>" LOC="K29" | IOSTANDARD=LVTTL;
NET "systemace_data<8>" LOC="L29" | IOSTANDARD=LVTTL;
NET "systemace_data<7>" LOC="L28" | IOSTANDARD=LVTTL;
NET "systemace_data<6>" LOC="M29" | IOSTANDARD=LVTTL;
NET "systemace_data<5>" LOC="M28" | IOSTANDARD=LVTTL;
NET "systemace_data<4>" LOC="N29" | IOSTANDARD=LVTTL;
NET "systemace_data<3>" LOC="N28" | IOSTANDARD=LVTTL;
NET "systemace_data<2>" LOC="P28" | IOSTANDARD=LVTTL;
NET "systemace_data<1>" LOC="R29" | IOSTANDARD=LVTTL;
NET "systemace_data<0>" LOC="R28" | IOSTANDARD=LVTTL;
NET "systemace_address<6>" LOC="E29" | IOSTANDARD=LVTTL;
NET "systemace_address<5>" LOC="F28" | IOSTANDARD=LVTTL;
NET "systemace_address<4>" LOC="H31" | IOSTANDARD=LVTTL;
NET "systemace_address<3>" LOC="J30" | IOSTANDARD=LVTTL;
```

```
NET "systemace_address<2>" LOC="J31" | IOSTANDARD=LVTTL;
NET "systemace_address<1>" LOC="K30" | IOSTANDARD=LVTTL;
NET "systemace_address<0>" LOC="K31" | IOSTANDARD=LVTTL;
NET "systemace_ce_b" LOC="E28" | IOSTANDARD=LVTTL;
NET "systemace_we_b" LOC="P31" | IOSTANDARD=LVTTL;
NET "systemace_oe_b" LOC="R31" | IOSTANDARD=LVTTL;
NET "systemace_irq" LOC = "D29";
NET "systemace_mpbrdy" LOC = "L31";
#
# Logic Analyzer
#
NET "analyzer1_data<15>" LOC="G1" | IOSTANDARD=LVTTL;
NET "analyzer1_data<14>" LOC="H3" | IOSTANDARD=LVTTL;
NET "analyzer1_data<13>" LOC="M9" | IOSTANDARD=LVTTL;
NET "analyzer1_data<12>" LOC="M8" | IOSTANDARD=LVTTL;
NET "analyzer1_data<11>" LOC="L5" | IOSTANDARD=LVTTL;
NET "analyzer1_data<10>" LOC="L1" | IOSTANDARD=LVTTL;
NET "analyzer1_data<9>" LOC="L2" | IOSTANDARD=LVTTL;
NET "analyzer1_data<8>" LOC="N9" | IOSTANDARD=LVTTL;
NET "analyzer1_data<7>" LOC="M1" | IOSTANDARD=LVTTL;
NET "analyzer1_data<6>" LOC="M2" | IOSTANDARD=LVTTL;
NET "analyzer1_data<5>" LOC="N1" | IOSTANDARD=LVTTL;
NET "analyzer1_data<4>" LOC="N2" | IOSTANDARD=LVTTL;
NET "analyzer1_data<3>" LOC="P1" | IOSTANDARD=LVTTL;
NET "analyzer1_data<2>" LOC="P2" | IOSTANDARD=LVTTL;
NET "analyzer1_data<1>" LOC="R1" | IOSTANDARD=LVTTL;
NET "analyzer1_data<0>" LOC="R2" | IOSTANDARD=LVTTL;
NET "analyzer1_clock" LOC="G2" | IOSTANDARD=LVTTL;
NET "analyzer2_data<15>" LOC="f2" | IOSTANDARD=LVTTL;
NET "analyzer2_data<14>" LOC="k10" | IOSTANDARD=LVTTL;
NET "analyzer2_data<13>" LOC="l10" | IOSTANDARD=LVTTL;
NET "analyzer2_data<12>" LOC="m10" | IOSTANDARD=LVTTL;
NET "analyzer2_data<11>" LOC="r7" | IOSTANDARD=LVTTL;
NET "analyzer2_data<10>" LOC="n3" | IOSTANDARD=LVTTL;
NET "analyzer2_data<9>" LOC="r8" | IOSTANDARD=LVTTL;
NET "analyzer2_data<8>" LOC="r9" | IOSTANDARD=LVTTL;
NET "analyzer2_data<7>" LOC="p9" | IOSTANDARD=LVTTL;
NET "analyzer2_data<6>" LOC="n6" | IOSTANDARD=LVTTL;
NET "analyzer2_data<5>" LOC="p7" | IOSTANDARD=LVTTL;
NET "analyzer2_data<4>" LOC="n4" | IOSTANDARD=LVTTL;
NET "analyzer2_data<3>" LOC="t8" | IOSTANDARD=LVTTL;
NET "analyzer2_data<2>" LOC="t9" | IOSTANDARD=LVTTL;
NET "analyzer2_data<1>" LOC="r6" | IOSTANDARD=LVTTL;
NET "analyzer2_data<0>" LOC="r5" | IOSTANDARD=LVTTL;
NET "analyzer2_clock" LOC="f1" | IOSTANDARD=LVTTL;
```

```
NET "analyzer3_data<15>" LOC="k24" | IOSTANDARD=LVTTL;
NET "analyzer3_data<14>" LOC="k25" | IOSTANDARD=LVTTL;
NET "analyzer3_data<13>" LOC="k22" | IOSTANDARD=LVTTL;
NET "analyzer3_data<12>" LOC="l24" | IOSTANDARD=LVTTL;
NET "analyzer3_data<11>" LOC="l25" | IOSTANDARD=LVTTL;
NET "analyzer3_data<10>" LOC="l22" | IOSTANDARD=LVTTL;
NET "analyzer3_data<9>" LOC="l23" | IOSTANDARD=LVTTL;
NET "analyzer3_data<8>" LOC="m23" | IOSTANDARD=LVTTL;
NET "analyzer3_data<7>" LOC="m24" | IOSTANDARD=LVTTL;
NET "analyzer3_data<6>" LOC="m25" | IOSTANDARD=LVTTL;
NET "analyzer3_data<5>" LOC="k23" | IOSTANDARD=LVTTL;
NET "analyzer3_data<4>" LOC="m22" | IOSTANDARD=LVTTL;
NET "analyzer3_data<3>" LOC="n23" | IOSTANDARD=LVTTL;
NET "analyzer3_data<2>" LOC="p23" | IOSTANDARD=LVTTL;
NET "analyzer3_data<1>" LOC="r23" | IOSTANDARD=LVTTL;
NET "analyzer3_data<0>" LOC="r24" | IOSTANDARD=LVTTL;
NET "analyzer3_clock" LOC="j24" | IOSTANDARD=LVTTL;
NET "analyzer4_data<15>" LOC="ag7" | IOSTANDARD=LVTTL;
NET "analyzer4_data<14>" LOC="ak3" | IOSTANDARD=LVTTL;
NET "analyzer4_data<13>" LOC="aj5" | IOSTANDARD=LVTTL;
NET "analyzer4_data<12>" LOC="ak29" | IOSTANDARD=LVTTL;
NET "analyzer4_data<11>" LOC="ak28" | IOSTANDARD=LVTTL;
NET "analyzer4_data<10>" LOC="af25" | IOSTANDARD=LVTTL;
NET "analyzer4_data<9>" LOC="ag24" | IOSTANDARD=LVTTL;
NET "analyzer4_data<8>" LOC="af24" | IOSTANDARD=LVTTL;
NET "analyzer4_data<7>" LOC="af23" | IOSTANDARD=LVTTL;
NET "analyzer4_data<6>" LOC="al27" | IOSTANDARD=LVTTL;
NET "analyzer4_data<5>" LOC="ak27" | IOSTANDARD=LVTTL;
NET "analyzer4_data<4>" LOC="ah17" | IOSTANDARD=LVTTL;
NET "analyzer4_data<3>" LOC="ad13" | IOSTANDARD=LVTTL;
NET "analyzer4_data<2>" LOC="v7" | IOSTANDARD=LVTTL;
NET "analyzer4_data<1>" LOC="u7" | IOSTANDARD=LVTTL;
NET "analyzer4_data<0>" LOC="u8" | IOSTANDARD=LVTTL;
NET "analyzer4_clock" LOC="ad9" | IOSTANDARD=LVTTL;
NET "clock_27mhz" TNM_NET = "clock_27mhz";
TIMESPEC "TS_clk27" = PERIOD "clock_27mhz" 27 MHz HIGH 50 %;
NET "tv_in_line_clock1" TNM_NET = "tv_in_line_clock1";
TIMESPEC "TS_tv_in_line_clock1" = PERIOD "tv_in_line_clock1" 30 MHz HIGH 50 %;
```

A.25 lateral_velocity.v

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

```
//
// Create Date:    18:02:44 12/04/2007
// Design Name:
// Module Name:    lateral_velocity
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module lateral_velocity(clk, rst, coord_min, coord_max, change);
    input clk;
    input rst;
    input [9:0] coord_min;
 input [9:0] coord_max;
    output reg signed [10:0] change;

//reg [9:0] old_coord_min, old_coord_max;
reg signed [9:0] center, old_center;
always @ (posedge clk) begin
center <= (coord_max + coord_min) >> 1;
if (~(center == old_center)) begin
change <= (center - old_center) << 1;
//multiply by two to scale from camera to screen
old_center <= center;
end
end

endmodule
```

A.26 LPF.v

```
‘timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Mark Stevens
//
// Create Date:    18:44:36 11/14/2007
// Design Name:
```

```
// Module Name:    LPF
// Project Name:
// Target Devices:
// Tool versions:
// Description: Read in a stream of boolean values determining whether or
// not ycrcb thresholding thought the current pixel was a paddle pixel.
// Given this stream, mark pixels as paddle pixels if several pixels to
// the left of the given pixel and above the given pixel are also within
// threshhold. Pixels to the left are retained using a shift register, while
// vertical pixels are retained in a small block memory that maps x coordinates
// to data representing the past several results for that x coordinate.
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module LPF(clk, rst, isPaddle, newline, x, paddleFiltered);
    input clk;
    input rst;
    input isPaddle;
    input newline;
 input [9:0] x;
    output reg paddleFiltered;

 parameter storedPixels = 9;



reg [storedPixels - 1:0] oldPixels;
reg [3:0] count;
reg [9:0] read_addr, x_q, write_addr;
reg ipf_q, we, isPaddleFiltered;
reg [3:0] din;
wire [3:0] dout;


always @ (posedge clk)
begin
if (rst || newline) begin
oldPixels <= 0;
count <= 0;
end
```

```
else begin
oldPixels <= {oldPixels[storedPixels - 2:0], isPaddle};
case ({oldPixels[storedPixels - 1],isPaddle})
2'b00: count <= count;
2'b01: count <= count + 1;
2'b11: count <= count;
2'b10: count <= (count == 0) ? 0 : count - 1;
endcase
end
isPaddleFiltered <= (count > 7);
//read from mem

read_addr <= x;
x_q <= x;
ipf_q <= isPaddleFiltered;

//next cycle
paddleFiltered <= (ipf_q & dout[0]) & (dout[1] | dout[2]);
write_addr <= x_q;
we <= 1;
din <= {dout[1:0], ipf_q};



end

lpf_mem_vert lpf_mem0(write_addr, read_addr, clk, clk, din, dout, we);




endmodule
```

A.27 lpf_mem.v

```
/*****************************************************************************
*     This file is owned and controlled by Xilinx and must be used           *
*     solely for design, simulation, implementation and creation of          *
*     design files limited to Xilinx devices or technologies. Use            *
*     with non-Xilinx devices or technologies is expressly prohibited        *
*     and immediately terminates your license.                               *
*                                                                            *
*     XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS"          *
*     SOLELY FOR USE IN DEVELOPING PROGRAMS AND SOLUTIONS FOR                *
```

```
*     XILINX DEVICES.  BY PROVIDING THIS DESIGN, CODE, OR INFORMATION          *
*     AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE, APPLICATION              *
*     OR STANDARD, XILINX IS MAKING NO REPRESENTATION THAT THIS                *
*     IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT,                  *
*     AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE         *
*     FOR YOUR IMPLEMENTATION.  XILINX EXPRESSLY DISCLAIMS ANY                 *
*     WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE                  *
*     IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR           *
*     REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF          *
*     INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS          *
*     FOR A PARTICULAR PURPOSE.                                                *
*                                                                              *
*     Xilinx products are not intended for use in life support                 *
*     appliances, devices, or systems. Use in such applications are            *
*     expressly prohibited.                                                    *
*                                                                              *
*     (c) Copyright 1995-2006 Xilinx, Inc.                                     *
*     All rights reserved.                                                     *
*******************************************************************************/
// The synopsys directives "translate_off/translate_on" specified below are
// supported by XST, FPGA Compiler II, Mentor Graphics and Synplicity synthesis
// tools. Ensure they are correct for your synthesis tool(s).

// You must compile the wrapper file lpf_mem.v when simulating
// the core, lpf_mem. When compiling the wrapper file, be sure to
// reference the XilinxCoreLib Verilog simulation library. For detailed
// instructions, please refer to the "CORE Generator Help".

`timescale 1ns/1ps

module lpf_mem(
addra,
addrb,
clka,
clkb,
dinb,
douta,
web);


input [7 : 0] addra;
input [7 : 0] addrb;
input clka;
input clkb;
input [15 : 0] dinb;
```

```
output [15 : 0] douta;
input web;

// synopsys translate_off

      BLKMEMDP_V6_3 #(
8,// c_addra_width
8,// c_addrb_width
"0",// c_default_data
256,// c_depth_a
256,// c_depth_b
0,// c_enable_rlocs
1,// c_has_default_data
0,// c_has_dina
1,// c_has_dinb
1,// c_has_douta
0,// c_has_doutb
0,// c_has_ena
0,// c_has_enb
0,// c_has_limit_data_pitch
0,// c_has_nda
0,// c_has_ndb
0,// c_has_rdya
0,// c_has_rdyb
0,// c_has_rfda
0,// c_has_rfdb
0,// c_has_sinita
0,// c_has_sinitb
0,// c_has_wea
1,// c_has_web
18,// c_limit_data_pitch
"mif_file_16_1",// c_mem_init_file
0,// c_pipe_stages_a
0,// c_pipe_stages_b
0,// c_reg_inputsa
1,// c_reg_inputsb
"NONE",// c_sim_collision_check
"0",// c_sinita_value
"0",// c_sinitb_value
16,// c_width_a
16,// c_width_b
0,// c_write_modea
1,// c_write_modeb
"0",// c_ybottom_addr
1,// c_yclka_is_rising
```

```
1,// c_yclkb_is_rising
1,// c_yena_is_high
1,// c_yenb_is_high
"hierarchy1",// c_yhierarchy
0,// c_ymake_bmm
"16kx1",// c_yprimitive_type
1,// c_ysinita_is_high
1,// c_ysinitb_is_high
"1024",// c_ytop_addr
0,// c_yuse_single_primitive
1,// c_ywea_is_high
1,// c_yweb_is_high
1) // c_yydisable_warnings
inst (
.ADDRA(addra),
.ADDRB(addrb),
.CLKA(clka),
.CLKB(clkb),
.DINB(dinb),
.DOUTA(douta),
.WEB(web),
.DINA(),
.DOUTB(),
.ENA(),
.ENB(),
.NDA(),
.NDB(),
.RFDA(),
.RFDB(),
.RDYA(),
.RDYB(),
.SINITA(),
.SINITB(),
.WEA());


// synopsys translate_on

// FPGA Express black box declaration
// synopsys attribute fpga_dont_touch "true"
// synthesis attribute fpga_dont_touch of lpf_mem is "true"

// XST black box declaration
// box_type "black_box"
// synthesis attribute box_type of lpf_mem is "black_box"
```

```
endmodule
```

A.28 lpf_mem_vert.v

```
/*****************************************************************************
*     This file is owned and controlled by Xilinx and must be used           *
*     solely for design, simulation, implementation and creation of          *
*     design files limited to Xilinx devices or technologies. Use            *
*     with non-Xilinx devices or technologies is expressly prohibited        *
*     and immediately terminates your license.                               *
*                                                                            *
*     XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS"          *
*     SOLELY FOR USE IN DEVELOPING PROGRAMS AND SOLUTIONS FOR                *
*     XILINX DEVICES.  BY PROVIDING THIS DESIGN, CODE, OR INFORMATION        *
*     AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE, APPLICATION            *
*     OR STANDARD, XILINX IS MAKING NO REPRESENTATION THAT THIS              *
*     IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT,                *
*     AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE       *
*     FOR YOUR IMPLEMENTATION.  XILINX EXPRESSLY DISCLAIMS ANY               *
*     WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE                *
*     IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR         *
*     REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF        *
*     INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS        *
*     FOR A PARTICULAR PURPOSE.                                              *
*                                                                            *
*     Xilinx products are not intended for use in life support               *
*     appliances, devices, or systems. Use in such applications are          *
*     expressly prohibited.                                                  *
*                                                                            *
*     (c) Copyright 1995-2006 Xilinx, Inc.                                   *
*     All rights reserved.                                                   *
*****************************************************************************/
// The synopsys directives "translate_off/translate_on" specified below are
// supported by XST, FPGA Compiler II, Mentor Graphics and Synplicity synthesis
// tools. Ensure they are correct for your synthesis tool(s).

// You must compile the wrapper file lpf_mem_vert.v when simulating
// the core, lpf_mem_vert. When compiling the wrapper file, be sure to
// reference the XilinxCoreLib Verilog simulation library. For detailed
// instructions, please refer to the "CORE Generator Help".

`timescale 1ns/1ps
```

```
module lpf_mem_vert(
addra,
addrb,
clka,
clkb,
dina,
doutb,
wea);


input [9 : 0] addra;
input [9 : 0] addrb;
input clka;
input clkb;
input [3 : 0] dina;
output [3 : 0] doutb;
input wea;

// synopsys translate_off

      BLKMEMDP_V6_3 #(
10,// c_addra_width
10,// c_addrb_width
"0",// c_default_data
1024,// c_depth_a
1024,// c_depth_b
0,// c_enable_rlocs
1,// c_has_default_data
1,// c_has_dina
0,// c_has_dinb
0,// c_has_douta
1,// c_has_doutb
0,// c_has_ena
0,// c_has_enb
0,// c_has_limit_data_pitch
0,// c_has_nda
0,// c_has_ndb
0,// c_has_rdya
0,// c_has_rdyb
0,// c_has_rfda
0,// c_has_rfdb
0,// c_has_sinita
0,// c_has_sinitb
1,// c_has_wea
0,// c_has_web
```

```
18,// c_limit_data_pitch
"mif_file_16_1",// c_mem_init_file
0,// c_pipe_stages_a
0,// c_pipe_stages_b
1,// c_reg_inputsa
0,// c_reg_inputsb
"NONE",// c_sim_collision_check
"0",// c_sinita_value
"0",// c_sinitb_value
4,// c_width_a
4,// c_width_b
1,// c_write_modea
0,// c_write_modeb
"0",// c_ybottom_addr
1,// c_yclka_is_rising
1,// c_yclkb_is_rising
1,// c_yena_is_high
1,// c_yenb_is_high
"hierarchy1",// c_yhierarchy
0,// c_ymake_bmm
"16kx1",// c_yprimitive_type
1,// c_ysinita_is_high
1,// c_ysinitb_is_high
"1024",// c_ytop_addr
0,// c_yuse_single_primitive
1,// c_ywea_is_high
1,// c_yweb_is_high
1) // c_yydisable_warnings
inst (
.ADDRA(addra),
.ADDRB(addrb),
.CLKA(clka),
.CLKB(clkb),
.DINA(dina),
.DOUTB(doutb),
.WEA(wea),
.DINB(),
.DOUTA(),
.ENA(),
.ENB(),
.NDA(),
.NDB(),
.RFDA(),
.RFDB(),
.RDYA(),
```

```
.RDYB(),
.SINITA(),
.SINITB(),
.WEB());


// synopsys translate_on

// FPGA Express black box declaration
// synopsys attribute fpga_dont_touch "true"
// synthesis attribute fpga_dont_touch of lpf_mem_vert is "true"

// XST black box declaration
// box_type "black_box"
// synthesis attribute box_type of lpf_mem_vert is "black_box"

endmodule
```

A.29 mod_divide.v

```
/*****************************************************************************
*     This file is owned and controlled by Xilinx and must be used           *
*     solely for design, simulation, implementation and creation of          *
*     design files limited to Xilinx devices or technologies. Use            *
*     with non-Xilinx devices or technologies is expressly prohibited        *
*     and immediately terminates your license.                               *
*                                                                            *
*     XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS"          *
*     SOLELY FOR USE IN DEVELOPING PROGRAMS AND SOLUTIONS FOR                *
*     XILINX DEVICES.  BY PROVIDING THIS DESIGN, CODE, OR INFORMATION        *
*     AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE, APPLICATION            *
*     OR STANDARD, XILINX IS MAKING NO REPRESENTATION THAT THIS              *
*     IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT,                *
*     AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE       *
*     FOR YOUR IMPLEMENTATION.  XILINX EXPRESSLY DISCLAIMS ANY               *
*     WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE                *
*     IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR         *
*     REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF        *
*     INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS        *
*     FOR A PARTICULAR PURPOSE.                                              *
*                                                                            *
*     Xilinx products are not intended for use in life support               *
*     appliances, devices, or systems. Use in such applications are          *
*     expressly prohibited.                                                  *
```

```
*                                                                            *
*     (c) Copyright 1995-2006 Xilinx, Inc.                                   *
*     All rights reserved.                                                   *
******************************************************************************/
// The synopsys directives "translate_off/translate_on" specified below are
// supported by XST, FPGA Compiler II, Mentor Graphics and Synplicity synthesis
// tools. Ensure they are correct for your synthesis tool(s).

// You must compile the wrapper file mod_divide.v when simulating
// the core, mod_divide. When compiling the wrapper file, be sure to
// reference the XilinxCoreLib Verilog simulation library. For detailed
// instructions, please refer to the "CORE Generator Help".

`timescale 1ns/1ps

module mod_divide(
clk,
dividend,
divisor,
quotient,
remainder,
rfd);


input clk;
input [31 : 0] dividend;
input [13 : 0] divisor;
output [31 : 0] quotient;
output [13 : 0] remainder;
output rfd;

// synopsys translate_off

      DIV_GEN_V1_0 #(
1,// algorithm_type
0,// bias
0,// c_has_aclr
0,// c_has_ce
0,// c_has_sclr
0,// c_sync_enable
1,// divclk_sel
32,// dividend_width
14,// divisor_width
8,// exponent_width
0,// fractional_b
```

```
14,// fractional_width
1,// latency
8,// mantissa_width
0) // signed_b
inst (
.CLK(clk),
.DIVIDEND(dividend),
.DIVISOR(divisor),
.QUOTIENT(quotient),
.REMAINDER(remainder),
.RFD(rfd),
.CE(),
.ACLR(),
.SCLR(),
.DIVIDEND_MANTISSA(),
.DIVIDEND_SIGN(),
.DIVIDEND_EXPONENT(),
.DIVISOR_MANTISSA(),
.DIVISOR_SIGN(),
.DIVISOR_EXPONENT(),
.QUOTIENT_MANTISSA(),
.QUOTIENT_SIGN(),
.QUOTIENT_EXPONENT(),
.OVERFLOW(),
.UNDERFLOW());


// synopsys translate_on

// FPGA Express black box declaration
// synopsys attribute fpga_dont_touch "true"
// synthesis attribute fpga_dont_touch of mod_divide is "true"

// XST black box declaration
// box_type "black_box"
// synthesis attribute box_type of mod_divide is "black_box"

endmodule
```

A.30 multiplayer.v

```
//Implements the communication between two labkits over serial
//Responsible for synchronizing data
```

```
// clk : 27Mhz
// Baud rate is set to 115200
// Parity : none
// Data bits : 8
// Stop bit : 1
// Flow control : None

module multiplayer(clk,vsync,txd,rxd,reset,paddleminX_IN,paddleminY_IN,paddlemaxX_IN,paddlemax
paddleminX_MINE_out,paddleminY_MINE_out,paddlemaxX_MINE_out,paddlemaxY_MINE_out,
paddleminX_OPP,paddleminY_OPP,paddlemaxX_OPP,paddlemaxY_OPP,
ballpixelX_REMOTE, ballpixelY_REMOTE, ballpixelX_LOCAL, ballpixelY_LOCAL,
ballX_LOCAL,ballY_LOCAL,ballZ_LOCAL,
ballX_OUT,ballY_OUT,ballZ_OUT,single_player, master_player, wall_hit_in, wall_hit_out,
paddleXspeed_LOCAL, paddleYspeed_LOCAL, paddleZspeed_LOCAL,
paddleXspeed_MINE, paddleYspeed_MINE, paddleZspeed_MINE,
paddleXspeed_OPP, paddleYspeed_OPP, paddleZspeed_OPP);

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

input vsync;

input single_player;
input clk;
output txd;
input rxd;
input reset;

input master_player;

input signed [10:0] paddleXspeed_LOCAL, paddleYspeed_LOCAL, paddleZspeed_LOCAL;
reg signed [10:0] paddleXspeed_REMOTE, paddleYspeed_REMOTE, paddleZspeed_REMOTE;
output signed [10:0] paddleXspeed_MINE, paddleYspeed_MINE, paddleZspeed_MINE;
output signed [10:0] paddleXspeed_OPP, paddleYspeed_OPP, paddleZspeed_OPP;


output [10:0] paddlemaxX_MINE_out,paddleminX_MINE_out,paddlemaxX_OPP,paddleminX_OPP, ballpixel
output [9:0] paddlemaxY_MINE_out,paddleminY_MINE_out,paddlemaxY_OPP,paddleminY_OPP, ballpixelY_

reg [10:0] paddlemaxX_MINE_out,paddleminX_MINE_out;
reg [9:0] paddlemaxY_MINE_out,paddleminY_MINE_out;

wire [10:0] paddlemaxX_OPP,paddleminX_OPP;
wire [9:0] paddlemaxY_OPP,paddleminY_OPP;
```

```
input [10:0] paddlemaxX_IN,paddleminX_IN, ballpixelX_LOCAL;
input [9:0] paddlemaxY_IN,paddleminY_IN, ballpixelY_LOCAL;

reg [10:0] paddlemaxX_MINE,paddleminX_MINE, ballpixelX_REMOTE;
reg [9:0] paddlemaxY_MINE,paddleminY_MINE, ballpixelY_REMOTE;

input signed [12:0] ballX_LOCAL,ballY_LOCAL,ballZ_LOCAL;

output signed [12:0] ballX_OUT,ballY_OUT,ballZ_OUT;

wire signed [12:0] ballX_OUT,ballY_OUT,ballZ_OUT;

reg signed [12:0] ballX_REMOTE,ballY_REMOTE,ballZ_REMOTE;

input [3:0] wall_hit_in;
output [3:0] wall_hit_out;

reg [3:0] wall_hit_out, wall_hit_REMOTE;
wire [3:0] wall_hit;


//Assign outputs on rising edge of vsync that are destined for output logic
//everything else is updated in realtime
reg last_vsync;

always @ (posedge clk) begin
if(reset) begin
last_vsync <= vsync;
paddlemaxX_MINE_out <= paddlemaxX_MINE;
paddleminX_MINE_out <= paddleminX_MINE;
paddlemaxY_MINE_out <= paddlemaxY_MINE;
paddleminY_MINE_out <= paddleminY_MINE;
wall_hit_out <= wall_hit;
end else begin
last_vsync <= vsync;
if(vsync == 1 && last_vsync == 0) begin
paddlemaxX_MINE_out <= paddlemaxX_MINE;
paddleminX_MINE_out <= paddleminX_MINE;
paddlemaxY_MINE_out <= paddlemaxY_MINE;
paddleminY_MINE_out <= paddleminY_MINE;
wall_hit_out <= wall_hit;
end
end
end
```

```
wire rready;  // receiver ready, data comes out on the next clock cycle
wire [7:0] rdata;  // receiver data
reg tstart; // transmitter start
wire tbusy; // transmitter busy
wire [7:0] tdata; // transmitter data to send, data sampled on the next clock cycle of start g

//Possible command chars

parameter ATTENTION = 8'h21;
parameter NO_COMMAND = 0;
parameter MIN_X = 8'h30;
parameter MAX_X = 8'h31;
parameter MIN_Y = 8'h32;
parameter MAX_Y = 8'h33;
parameter BALLX = 8'h34;
parameter BALLY = 8'h35;
parameter BALLZ = 8'h36;
parameter BALLPIXELX = 8'h37;
parameter BALLPIXELY = 8'h38;
parameter WALLHIT = 8'h39;
parameter PADDLEXSPEED = 8'h3a;
parameter PADDLEYSPEED = 8'h3b;
parameter PADDLEZSPEED = 8'h3c;

//States for Transmitter major state machine

parameter t_s_idle = 4'd0;
parameter t_s_sendminX = 4'd1;
parameter t_s_sendmaxX = 4'd2;
parameter t_s_sendminY = 4'd3;
parameter t_s_sendmaxY = 4'd4;
parameter t_s_sendballX = 4'd5;
parameter t_s_sendballY = 4'd6;
parameter t_s_sendballZ = 4'd7;
parameter t_s_sendballpixelX = 4'd8;
parameter t_s_sendballpixelY = 4'd9;
parameter t_s_sendwallhit = 4'd10;
parameter t_s_sendpaddlexspeed = 4'd11;
parameter t_s_sendpaddleyspeed = 4'd12;
parameter t_s_sendpaddlezspeed = 4'd13;


reg [3:0] t_state;

reg t_substate_start;
```

```
wire t_substate_ready;
wire [7:0] t_command_byte,t_upper_byte,t_lower_byte;

//These are the bytes to be sent over serial.

//Protocol is 4 byte packets consisting of:
//ATTENTION
//COMMAND
//HIGHBYTE
//LOWBYTE

//The main FSM just rotates through the chunks of information we want to send

assign t_upper_byte = ((t_state == t_s_idle) ? 8'h3b :
  (t_state == t_s_sendminX) ? {5'd1,paddleminX_IN[10:8]} :
  (t_state == t_s_sendmaxX) ? {5'd1,paddlemaxX_IN[10:8]} :
  (t_state == t_s_sendminY) ? {6'd1,paddleminY_IN[9:8]}  :
  (t_state == t_s_sendmaxY) ? {6'd1,paddlemaxY_IN[9:8]}  :
  (t_state == t_s_sendballX) ? {4'd1,ballX_LOCAL[12:8]}  :
  (t_state == t_s_sendballY) ? {4'd1,ballY_LOCAL[12:8]}  :
  (t_state == t_s_sendballZ) ? {4'd1,ballZ_LOCAL[12:8]}  :
  (t_state == t_s_sendballpixelX) ? {5'd1,ballpixelX_LOCAL[10:8]}  :
  (t_state == t_s_sendballpixelY) ? {6'd1,ballpixelY_LOCAL[9:8]}  :
  (t_state == t_s_sendwallhit) ? 8'd0 :
  (t_state == t_s_sendpaddlexspeed) ? {5'd1,paddleXspeed_LOCAL[10:8]}  :
  (t_state == t_s_sendpaddleyspeed) ? {5'd1,paddleYspeed_LOCAL[10:8]}  :
  (t_state == t_s_sendpaddlezspeed) ? {5'd1,paddleZspeed_LOCAL[10:8]}  : 0);


assign t_lower_byte = ((t_state == t_s_idle) ? 8'h3c :
  (t_state == t_s_sendminX) ? paddleminX_IN[7:0] :
  (t_state == t_s_sendmaxX) ? paddlemaxX_IN[7:0] :
  (t_state == t_s_sendminY) ? paddleminY_IN[7:0]  :
  (t_state == t_s_sendmaxY) ? paddlemaxY_IN[7:0]  :
  (t_state == t_s_sendballX) ? ballX_LOCAL[7:0]   :
  (t_state == t_s_sendballY) ? ballY_LOCAL[7:0]   :
  (t_state == t_s_sendballZ) ? ballZ_LOCAL[7:0]   :
  (t_state == t_s_sendballpixelX) ? ballpixelX_LOCAL[7:0]   :
  (t_state == t_s_sendballpixelY) ? ballpixelY_LOCAL[7:0]   :
  (t_state == t_s_sendwallhit) ? {4'd0, wall_hit_in} :
  (t_state == t_s_sendpaddlexspeed) ? paddleXspeed_LOCAL[7:0] :
  (t_state == t_s_sendpaddleyspeed) ? paddleYspeed_LOCAL[7:0] :
  (t_state == t_s_sendpaddlezspeed) ? paddleZspeed_LOCAL[7:0] : 0);

assign t_command_byte = ((t_state == t_s_idle) ? 8'h3a :
```

```
  (t_state == t_s_sendminX) ? MIN_X :
  (t_state == t_s_sendmaxX) ? MAX_X :
  (t_state == t_s_sendminY) ? MIN_Y  :
  (t_state == t_s_sendmaxY) ? MAX_Y  :
  (t_state == t_s_sendballX) ? BALLX :
  (t_state == t_s_sendballY) ? BALLY :
  (t_state == t_s_sendballZ) ? BALLZ :
  (t_state == t_s_sendballpixelX) ? BALLPIXELX :
  (t_state == t_s_sendballpixelY) ? BALLPIXELY :
  (t_state == t_s_sendwallhit) ? WALLHIT :
  (t_state == t_s_sendpaddlexspeed) ? PADDLEXSPEED :
  (t_state == t_s_sendpaddleyspeed) ? PADDLEYSPEED :
  (t_state == t_s_sendpaddlezspeed) ? PADDLEZSPEED : 0);

always @ (posedge clk) begin
if (reset) begin
t_state <= t_s_idle;
t_substate_start <= 0;
end else if(t_substate_ready && ~t_substate_start) begin  //Transmitter sub state machine is re

t_substate_start <= 1;  //Start the substate machine on the next clock cycle

//update state

case(t_state)
t_s_idle: t_state <= t_s_sendminX;
t_s_sendminX: t_state <= t_s_sendmaxX;
t_s_sendmaxX: t_state <= t_s_sendminY;
t_s_sendminY: t_state <= t_s_sendmaxY;
t_s_sendmaxY: t_state <= t_s_sendballX;
t_s_sendballX: t_state <= t_s_sendballY;
t_s_sendballY: t_state <= t_s_sendballZ;
t_s_sendballZ: t_state <= t_s_sendballpixelX;
t_s_sendballpixelX: t_state <= t_s_sendballpixelY;
t_s_sendballpixelY: t_state <= t_s_sendwallhit;
t_s_sendwallhit: t_state <= t_s_sendpaddlexspeed;
t_s_sendpaddlexspeed: t_state <= t_s_sendpaddleyspeed;
t_s_sendpaddleyspeed: t_state <= t_s_sendpaddlezspeed;
t_s_sendpaddlezspeed: t_state <= t_s_sendminX;
default: t_state <= t_s_idle;
endcase

end else begin
t_substate_start <= 0;
end
```

```
end

//States for the Transmitter sub state machine that sends packets
parameter ts_s_idle = 3'd0;
parameter ts_s_sending_attention = 3'd1;
parameter ts_s_sending_command = 3'd2;
parameter ts_s_sending_high = 3'd3;
parameter ts_s_sending_low = 3'd4;

reg[2:0] ts_state;

assign t_substate_ready = ((ts_state == ts_s_idle) && ~tbusy);  //we are ready if the state is

//Data depends on our state
assign tdata = ((ts_state == ts_s_idle) ? 8'h37 :
  (ts_state == ts_s_sending_attention) ? ATTENTION :
  (ts_state == ts_s_sending_command) ? t_command_byte :
  (ts_state == ts_s_sending_high) ? t_upper_byte  :
  (ts_state == ts_s_sending_low) ? t_lower_byte  : 0);

reg settingup;

always @ (posedge clk) begin
if(reset) begin
ts_state <= ts_s_idle;
tstart <= 0;
settingup <= 0;
end else if (t_substate_start) begin  //If we've been given a start order, get going
ts_state <= ts_s_sending_attention;
settingup <= 1;
end else if (settingup) begin  //We have set up the input to the transmitter, so transmit
tstart <= 1;
settingup <= 0;
end else if (tstart) begin  //We started transmitting last cycle, so lower the wire
tstart <= 0;
end else if (~tbusy && ~tstart && ~settingup) begin  //If the transmitter is finished and we di

//advance state

case(ts_state)
ts_s_idle: ts_state <= ts_s_idle;
ts_s_sending_attention: ts_state <= ts_s_sending_command;
ts_s_sending_command: ts_state <= ts_s_sending_high;
ts_s_sending_high: ts_state <= ts_s_sending_low;
ts_s_sending_low: ts_state <= ts_s_idle;
```

```
default: ts_state <= ts_s_idle;
endcase

settingup <= (ts_state != ts_s_idle);  //As long as we're not idle, start sending the byte


end else begin
tstart <= 0;
end
end


//Now for receiver state machine

reg [2:0] r_state;
reg [15:0] incoming_data;
reg [7:0] incoming_command;

//Receiver state.  Tells which part of the packet we are receiving
parameter r_idle = 3'd0;
parameter r_receiving_command = 3'd1;
parameter r_receiving_high = 3'd2;
parameter r_receiving_low = 3'd3;
parameter r_storing_data = 3'd4;


//Assigns outputs.  If we're on single player, the opponent does not exist, so his paddle takes
//That way gamelogic does not need to care about whether it is single or mutliplayer.
assign paddlemaxX_OPP = single_player ? 1023 : paddlemaxX_IN;
assign paddleminX_OPP = single_player ? 0 : paddleminX_IN;
assign paddlemaxY_OPP = single_player ? 767 : paddlemaxY_IN;
assign paddleminY_OPP = single_player ? 0 : paddleminY_IN;

//Same thing with ball coordinates.  If we're the master or the only one, use our ball.  Otherw
//remote because we're the slave.
assign ballX_OUT = (single_player || master_player) ? ballX_LOCAL : (13'd1023 - ballX_REMOTE);
assign ballY_OUT = (single_player || master_player) ? ballY_LOCAL : (13'd4096 - ballY_REMOTE);
assign ballZ_OUT = (single_player || master_player) ? ballZ_LOCAL : ballZ_REMOTE;

assign paddleXspeed_MINE = (single_player) ? paddleXspeed_LOCAL : paddleXspeed_REMOTE;
assign paddleYspeed_MINE = (single_player) ? paddleYspeed_LOCAL : paddleYspeed_REMOTE;
assign paddleZspeed_MINE = (single_player) ? paddleZspeed_LOCAL : paddleZspeed_REMOTE;

assign paddleXspeed_OPP = (single_player) ? 0 : paddleXspeed_LOCAL;
assign paddleYspeed_OPP = (single_player) ? 0 : paddleYspeed_LOCAL;
```

```
assign paddleZspeed_OPP = (single_player) ? 0 : paddleZspeed_LOCAL;

wire [3:0] wall_hit_FLIPPED;

parameter NO_WALL = 4'd0;
parameter RIGHT_WALL = 4'd1;
parameter LEFT_WALL = 4'd2;
parameter TOP_WALL = 4'd3;
parameter BOTTOM_WALL = 4'd4;
parameter BACK_WALL = 4'd5;
parameter FRONT_WALL = 4'd6;

//Mirror the wall in case we're the slave.
assign wall_hit_FLIPPED =  (wall_hit_REMOTE == NO_WALL) ? NO_WALL :
(wall_hit_REMOTE == RIGHT_WALL) ? LEFT_WALL:
(wall_hit_REMOTE == LEFT_WALL) ? RIGHT_WALL:
(wall_hit_REMOTE == TOP_WALL) ? TOP_WALL:
(wall_hit_REMOTE == BOTTOM_WALL) ? BOTTOM_WALL:
(wall_hit_REMOTE == FRONT_WALL) ? BACK_WALL:
(wall_hit_REMOTE == BACK_WALL) ? FRONT_WALL: NO_WALL;

assign wall_hit = (single_player || master_player) ? wall_hit_in : wall_hit_FLIPPED;

//Receiver state machine puts registers for the data it plans to receive
//These are updated every time a packet comes in.
always @ (posedge clk) begin
if(reset) begin
r_state <= r_idle;
paddlemaxX_MINE <= paddlemaxX_IN;
paddleminX_MINE <= paddleminX_IN;
paddlemaxY_MINE <= paddlemaxY_IN;
paddleminY_MINE <= paddleminY_IN;
ballX_REMOTE <= ballX_LOCAL;
ballY_REMOTE <= ballY_LOCAL;
ballZ_REMOTE <= ballZ_LOCAL;
ballpixelX_REMOTE <= ballpixelX_LOCAL;
ballpixelY_REMOTE <= ballpixelY_LOCAL;
wall_hit_REMOTE <= 0;
incoming_data <= 0;
incoming_command <= 0;
paddleXspeed_REMOTE <= paddleXspeed_LOCAL;
paddleYspeed_REMOTE <= paddleYspeed_LOCAL;
paddleZspeed_REMOTE <= paddleZspeed_LOCAL;
end else if (single_player) begin
r_state <= r_idle;
```

```
paddlemaxX_MINE <= paddlemaxX_IN;
paddleminX_MINE <= paddleminX_IN;
paddlemaxY_MINE <= paddlemaxY_IN;
paddleminY_MINE <= paddleminY_IN;
end else if (r_state == r_storing_data) begin

//We have received a full packet.  Put the data where it belongs.

r_state <= r_idle;
case(incoming_command)
MIN_X: paddleminX_MINE[7:0] <= incoming_data[7:0];
MAX_X: paddlemaxX_MINE[7:0] <= incoming_data[7:0];
MIN_Y: paddleminY_MINE[7:0] <= incoming_data[7:0];
MAX_Y: paddlemaxY_MINE[7:0] <= incoming_data[7:0];
BALLX: ballX_REMOTE[7:0] <= incoming_data[7:0];
BALLY: ballY_REMOTE[7:0] <= incoming_data[7:0];
BALLZ: ballZ_REMOTE[7:0] <= incoming_data[7:0];
BALLPIXELX: ballpixelX_REMOTE[7:0] <= incoming_data[7:0];
BALLPIXELY: ballpixelY_REMOTE[7:0] <= incoming_data[7:0];
WALLHIT: wall_hit_REMOTE[3:0] <= incoming_data[3:0];
PADDLEXSPEED: paddleXspeed_REMOTE[7:0] <= incoming_data[7:0];
PADDLEYSPEED: paddleYspeed_REMOTE[7:0] <= incoming_data[7:0];
PADDLEZSPEED: paddleZspeed_REMOTE[7:0] <= incoming_data[7:0];
default: paddleminX_MINE[7:0] <= paddleminX_MINE[7:0];
endcase

case(incoming_command)
MIN_X: paddleminX_MINE[10:8] <= incoming_data[10:8];
MAX_X: paddlemaxX_MINE[10:8] <= incoming_data[10:8];
MIN_Y: paddleminY_MINE[9:8] <= incoming_data[9:8];
MAX_Y: paddlemaxY_MINE[9:8] <= incoming_data[9:8];
BALLX: ballX_REMOTE[12:8] <= incoming_data[12:8];
BALLY: ballY_REMOTE[12:8] <= incoming_data[12:8];
BALLZ: ballZ_REMOTE[12:8] <= incoming_data[12:8];
BALLPIXELX: ballpixelX_REMOTE[10:8] <= incoming_data[10:8];
BALLPIXELY: ballpixelY_REMOTE[9:8] <= incoming_data[9:8];
WALLHIT: paddleminX_MINE[10:8] <= paddleminX_MINE[10:8];
PADDLEXSPEED: paddleXspeed_REMOTE[10:8] <= incoming_data[10:8];
PADDLEYSPEED: paddleYspeed_REMOTE[10:8] <= incoming_data[10:8];
PADDLEZSPEED: paddleZspeed_REMOTE[10:8] <= incoming_data[10:8];
default: paddleminX_MINE[10:8] <= paddleminX_MINE[10:8];
endcase
end else if (rready) begin

//If we're idle, grab the byte if it's there and advance state if it is attention
```

```
//Otherwise just advance state if we're in the middle of a packet.

case(r_state)
r_idle: r_state <= (rdata == ATTENTION) ? r_receiving_command : r_idle;
r_receiving_command: r_state <= r_receiving_high;
r_receiving_high: r_state <= r_receiving_low;
r_receiving_low: r_state <= r_storing_data;
r_storing_data: r_state <= r_idle;
default: r_state <= r_idle;
endcase

//Store data in appropriate place
case(r_state)
r_idle: incoming_command <= incoming_command;
r_receiving_command: incoming_command <= rdata;
r_receiving_high: incoming_data[15:8] <= rdata;
r_receiving_low: incoming_data[7:0] <= rdata;
r_storing_data: incoming_command <= incoming_command;
default: incoming_command <= incoming_command;
endcase

end
end

rs232c_receiver rs232r(.clk(clk),.rxdin( rxd), .data(rdata), .ready(rready));
rs232c_transmitter rs232t(.clk(clk), .txdout(txd), .data(tdata), .start(tstart), .busy(tbusy))

endmodule
```

A.31 ntsc2zbt.v

```
//
// 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.
```

```
//////////////////////////////////////////////////////////////////////////
// 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,
x_coord, y_coord);

   input   clk; // system clock
   input   vclk; // video clock from camera
   input [2:0]   fvh;
   input   dv; //data valid (probably from ntsc decoder)
   //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)
output reg [9:0] x_coord;
output reg [9:0] y_coord;

   parameter   COL_START = 10'd30;
   parameter   ROW_START = 10'd30;

   // here put the luminance data from the ntsc decoder into the ram
   // this is for 1024 x 768 XGA display

   reg [9:0]   col = 0;
   reg [9:0]   row = 0;
   //reg [7:0]   vdata = 0;
reg [17:0] din_q;
reg [17:0] vdata;
   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;

reg [9:0] next_col;

   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] | fvh[1]) ? COL_START : col + 1;
  col <= fvh[0] ? COL_START :
//(!fvh[2] && !fvh[1] && dv) ? col + 1 : col;
(!fvh[2] && !fvh[1] && dv) ? next_col : col;
next_col <= col + 1;
 //(!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;
  //din_q <= din;
  //vdata <= (dv && !fvh[2]) ? din_q : vdata;
  x_coord <= col;
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};

//x_coord <= x[1][9:0];
y_coord <= {y[1][8:0], eo[1]};
     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[35:18], data[1]};
       //mydata <= { mydata[23:0], data[1] };

   // compute address to store data in

   //wire [18:0] myaddr = {1'b0, y[1][8:0], eo[1], x[1][9:2]};
wire [18:0] myaddr = {y[1][8:0], eo[1], x[1][9:1]};

   // alternate (256x192) image data and address
   //wire [31:0] mydata2 = {data[1],data[1],data[1],data[1]};
   //wire [18:0] myaddr2 = {1'b0, y[1][8:0], eo[1], x[1][7:0]};
wire [35:0] mydata2 = {data[0], data[1]};
wire [18:0] myaddr2 = {y[1][8:0], eo[1], x[1][9:1]};

   // 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[1][1:0]==2'b00));

   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.32 ntsc_to_zbt_center.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    16:25:15 12/04/2007
```

```
// Design Name:
// Module Name:    ntsc_to_zbt_center
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module ntsc_to_zbt_center(clk, vclk, fvh, dv, din, ntsc_addr, ntsc_data, ntsc_we);

   input   clk; // system clock
   input   vclk; // video clock from camera
   input [2:0]   fvh;
   input   dv; //data valid (probably from ntsc decoder)
   //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
   wire   sw; // switch which determines mode (for debugging)
assign sw = 1;

   parameter   COL_START = 10'd30;
   parameter   ROW_START = 10'd30;

   // here put the luminance data from the ntsc decoder into the ram
   // this is for 1024 x 768 XGA display

   reg [9:0]   col = 0;
   reg [9:0]   row = 0;
   //reg [7:0]   vdata = 0;
reg [17:0] din_q;
reg [17:0] vdata;
   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;

reg [9:0] next_col;

   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] | fvh[1]) ? COL_START : col + 1;
  col <= fvh[0] ? COL_START :
//(!fvh[2] && !fvh[1] && dv) ? col + 1 : col;
(!fvh[2] && !fvh[1] && dv) ? next_col : col;
next_col <= col + 1;
 //(!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;
  //din_q <= din;
  //vdata <= (dv && !fvh[2]) ? din_q : 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];

wire [7:0] r, g, b;
YCrCb2RGB ycrcb(r, g, b, clk, rst, Y, Cr, Cb );

   always @(posedge clk)
     begin
x[0] <= col;
y[0] <= row;
if (x[0][0] == 0)
```

```
{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


   always @(posedge clk) begin
      x[1] <= ((x[0] - 128) >> 1) + 384;
y[1] <= (y[0] - 64) + 288;
end
       //mydata <= { mydata[23:0], data[1] };

   // compute address to store data in

   //wire [18:0] myaddr = {1'b0, y[1][8:0], eo[1], x[1][9:2]};
wire [18:0] myaddr = {y[1][8:0], eo[1], x[1][9:1]};

   // alternate (256x192) image data and address
   //wire [31:0] mydata2 = {data[1],data[1],data[1],data[1]};
   //wire [18:0] myaddr2 = {1'b0, y[1][8:0], eo[1], x[1][7:0]};
wire [35:0] mydata2 = {data[0], data[1]};
wire [18:0] myaddr2 = {y[1][9:0], x[1][9:1]};

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

   reg [18:0] ntsc_addr;
   reg [35:0] ntsc_data;
   wire       ntsc_we = we_edge;

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

```
endmodule // ntsc_to_zbt
```

A.33 paddle_boundaries.v

```
‘timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    22:53:18 12/02/2007
// Design Name:
// Module Name:    paddle_boundaries
// Project Name:
// Target Devices:
// Tool versions:
// Description: Given a stream of paddle pixels coming out of a low pass filter,
// determine the boundaries of the paddle. Boundaries are found in two different
// ways; first, by finding maximum and minimum coordiates for each frame of the
// camera; second, by finding corners. Corners are located by looking for the
// max x+y, max x-y, etc. The max/min coordinates are used to generate the paddle,
// while the corners are used to determine whether the paddle is tilted up or
// down.
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module paddle_boundaries(clk, rst, field_change, x_filtered, y_filtered,
min_x, max_x, min_y, max_y, x_ul, x_ur, x_bl,
x_br, y_ul, y_ur, y_bl, y_br);
    input clk;
    input rst;
 input field_change;
 input [9:0] x_filtered;
 input [9:0] y_filtered;
 output [9:0] min_x;
 output [9:0] max_x;
 output [9:0] min_y;
 output [9:0] max_y;
 output [9:0] x_ul, x_ur, x_bl, x_br, y_ul, y_ur, y_bl, y_br;
```

```
//True Corners:
reg signed [11:0] sum, diff;
always @ (posedge clk) begin
sum <= x_filtered + y_filtered;
diff <= x_filtered - y_filtered;
end

wire [9:0] d0, d1, d2, d3;
//edges
get_extrema get_x_min(clk, rst, x_filtered, field_change, min_x);
defparam get_x_min.MAX = 0;
get_extrema get_x_max(clk, rst, x_filtered, field_change, max_x);
defparam get_x_max.MAX = 1;
get_extrema get_y_min(clk, rst, y_filtered, field_change, min_y);
defparam get_y_min.MAX = 0;
get_extrema get_y_max(clk, rst, y_filtered, field_change, max_y);
defparam get_y_max.MAX = 1;
// get_extrema_signed get_x_min(clk, rst, x_filtered, y_filtered, x_filtered,
// field_change, min_x, d0);
// defparam get_x_min.MAX = 0;
// get_extrema_signed get_x_max(clk, rst, x_filtered, y_filtered, x_filtered,
// field_change, max_x, d1);
// defparam get_x_max.MAX = 1;
// get_extrema_signed get_y_min(clk, rst, x_filtered, y_filtered, y_filtered,
// field_change, d2, min_y);
// defparam get_y_min.MAX = 0;
// get_extrema_signed get_y_max(clk, rst, x_filtered, y_filtered, y_filtered,
// field_change, d3, max_y);
// defparam get_y_max.MAX = 1;

//corners
get_extrema_signed get_ul(clk, rst, x_filtered, y_filtered, sum,
field_change, x_ul, y_ul);
defparam get_ul.MAX = 0;
get_extrema_signed get_ur(clk, rst, x_filtered, y_filtered, diff,
field_change, x_ur, y_ur);
defparam get_ur.MAX = 1;
get_extrema_signed get_bl(clk, rst, x_filtered, y_filtered, diff,
field_change, x_bl, y_bl);
defparam get_bl.MAX = 0;
get_extrema_signed get_br(clk, rst, x_filtered, y_filtered, sum,
field_change, x_br, y_br);
defparam get_br.MAX = 1;
```

```
endmodule
```

A.34 paddle_divide.v

```
/*******************************************************************************
*     This file is owned and controlled by Xilinx and must be used             *
*     solely for design, simulation, implementation and creation of            *
*     design files limited to Xilinx devices or technologies. Use              *
*     with non-Xilinx devices or technologies is expressly prohibited          *
*     and immediately terminates your license.                                 *
*                                                                              *
*     XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS"            *
*     SOLELY FOR USE IN DEVELOPING PROGRAMS AND SOLUTIONS FOR                  *
*     XILINX DEVICES.  BY PROVIDING THIS DESIGN, CODE, OR INFORMATION          *
*     AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE, APPLICATION              *
*     OR STANDARD, XILINX IS MAKING NO REPRESENTATION THAT THIS                *
*     IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT,                  *
*     AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE         *
*     FOR YOUR IMPLEMENTATION.  XILINX EXPRESSLY DISCLAIMS ANY                 *
*     WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE                  *
*     IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR           *
*     REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF          *
*     INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS          *
*     FOR A PARTICULAR PURPOSE.                                                *
*                                                                              *
*     Xilinx products are not intended for use in life support                 *
*     appliances, devices, or systems. Use in such applications are            *
*     expressly prohibited.                                                    *
*                                                                              *
*     (c) Copyright 1995-2006 Xilinx, Inc.                                     *
*     All rights reserved.                                                     *
*******************************************************************************/
// The synopsys directives "translate_off/translate_on" specified below are
// supported by XST, FPGA Compiler II, Mentor Graphics and Synplicity synthesis
// tools. Ensure they are correct for your synthesis tool(s).

// You must compile the wrapper file paddle_divide.v when simulating
// the core, paddle_divide. When compiling the wrapper file, be sure to
// reference the XilinxCoreLib Verilog simulation library. For detailed
// instructions, please refer to the "CORE Generator Help".

`timescale 1ns/1ps
```

```
module paddle_divide(
clk,
dividend_mantissa,
dividend_sign,
dividend_exponent,
divisor_mantissa,
divisor_sign,
divisor_exponent,
quotient_mantissa,
quotient_sign,
quotient_exponent,
overflow,
underflow);


input clk;
input [9 : 0] dividend_mantissa;
input dividend_sign;
input [5 : 0] dividend_exponent;
input [9 : 0] divisor_mantissa;
input divisor_sign;
input [5 : 0] divisor_exponent;
output [9 : 0] quotient_mantissa;
output quotient_sign;
output [5 : 0] quotient_exponent;
output overflow;
output underflow;

// synopsys translate_off

      DIV_GEN_V1_0 #(
2,// algorithm_type
0,// bias
0,// c_has_aclr
0,// c_has_ce
0,// c_has_sclr
0,// c_sync_enable
1,// divclk_sel
16,// dividend_width
16,// divisor_width
6,// exponent_width
0,// fractional_b
16,// fractional_width
99,// latency
10,// mantissa_width
```

```
0) // signed_b
inst (
.CLK(clk),
.DIVIDEND_MANTISSA(dividend_mantissa),
.DIVIDEND_SIGN(dividend_sign),
.DIVIDEND_EXPONENT(dividend_exponent),
.DIVISOR_MANTISSA(divisor_mantissa),
.DIVISOR_SIGN(divisor_sign),
.DIVISOR_EXPONENT(divisor_exponent),
.QUOTIENT_MANTISSA(quotient_mantissa),
.QUOTIENT_SIGN(quotient_sign),
.QUOTIENT_EXPONENT(quotient_exponent),
.OVERFLOW(overflow),
.UNDERFLOW(underflow),
.CE(),
.ACLR(),
.SCLR(),
.DIVIDEND(),
.DIVISOR(),
.QUOTIENT(),
.REMAINDER(),
.RFD());


// synopsys translate_on

// FPGA Express black box declaration
// synopsys attribute fpga_dont_touch "true"
// synthesis attribute fpga_dont_touch of paddle_divide is "true"

// XST black box declaration
// box_type "black_box"
// synthesis attribute box_type of paddle_divide is "black_box"

endmodule
```

A.35 projectball.v

```
//////////////////////////////////////////////////////////////////////
//
// projectball : module to project X,Y,Z of ball to (x,y,size) in pixels
//
//////////////////////////////////////////////////////////////////////
```

```
module projectball(ballX_IN,ballY_IN,ballZ_IN,pixelX_OUT,pixelY_OUT,size_OUT,ball_color_OUT,cl

input  [12:0] ballX_IN;
input  [12:0] ballY_IN;
input  [12:0] ballZ_IN;
input clk,vsync,reset;
output [2:0] ball_color_OUT;

output [10:0] pixelX_OUT;
output [9:0] pixelY_OUT;
output [10:0] size_OUT;

reg [2:0] ball_color;
reg [10:0] pixelX;
reg [9:0] pixelY;
reg [10:0] size;

reg [2:0] ball_color_OUT;

reg [10:0] pixelX_OUT;
reg [9:0] pixelY_OUT;
reg [10:0] size_OUT;


reg  [12:0] ballX;
reg  [12:0] ballY;
reg  [12:0] ballZ;


//All multiplied by 4096
parameter P11 = 23'sd3816;
parameter P12 = 23'sd1412;
parameter P13 = 23'sd6;
parameter P14 = 23'sd74997;
parameter P21 = -23'sd46;
parameter P22 = 23'sd1012;
parameter P23 = -23'sd3701;
parameter P24 = 23'sd3158556;
parameter P31 = 23'sd0;
parameter P32 = 23'sd3;
parameter P33 = 23'sd0;
parameter P34 = 23'sd4096;

parameter WHITE = 3'b111;
parameter RED = 3'b100;
```

```
   parameter GREEN = 3'b010;

reg last_vsync;

reg [13:0] tempsize;
reg signed [32:0] unscaledX;
reg signed [32:0] unscaledY;
reg signed [32:0] xp1, xp2, xp3, yp1, yp2, yp3, tp1, tp2, tp3;

reg [31:0] unboundedpixelX, unboundedpixelY;

wire [31:0] temppixelX;
wire [31:0] temppixelY;

//This changes at the rising edge of vsync
//Keep inputs valid if possible

always @ (posedge clk) begin
if(reset) begin
ballX <= ballX_IN;
ballY <= ballY_IN;
ballZ <= ballZ_IN;
pixelX_OUT <= pixelX;
pixelY_OUT <= pixelY;
size_OUT <= size;
ball_color_OUT <= ball_color;
last_vsync <= vsync;
end else begin
last_vsync <= vsync;

if(vsync == 1 && last_vsync == 0) begin
//this is so that the projection updates every frame *after* the next frame has been grabbed by
ballX <= ballX_IN;
ballY <= ballY_IN;
ballZ <= ballZ_IN;
pixelX_OUT <= pixelX;
pixelY_OUT <= pixelY;
size_OUT <= size;
ball_color_OUT <= ball_color;
end


//Each parameter is 23 signed bits
//The balls are 12 bits of signed data (always positive, so bit 12 is always 0)
//Most are pretty sparse, so I'll go with 33 bits in unscaled X, Y, and 14 in tempsize
```

```
/*xp1 <= (ballX * P11);
xp2 <= (ballY * P12);
xp3 <= (ballZ * P13);

yp1 <= (ballX * P21);
yp2 <= (ballY * P22);
yp3 <= (ballZ * P23);

tp1 <= (ballX * P31);
tp2 <= (ballY * P32);
tp3 <= (ballZ * P33);

unscaledX <= xp1 + xp2 + xp3 + P14;
unscaledY <= yp1 + yp2 + yp3 + P24;
tempsize <= tp1 + tp2 + tp3 + P34;*/
     unscaledX <= (ballX * P11) + (ballY * P12) + (ballZ * P13) + P14;
     unscaledY <= (ballX * P21) + (ballY * P22) + (ballZ * P23) + P24;
     tempsize <= (ballX * P31) + (ballY * P32) + (ballZ * P33) + P34;


//Now get rid of the signedness

    unboundedpixelX <= unscaledX > 0 ? unscaledX : 0;
    unboundedpixelY <= unscaledY > 0 ? unscaledY : 0;

 //Divide occurs outside this always block in a pipelined divider
 //It takes unboundedpixel to temppixel

 //Now make sure everything is in bounds

 pixelX <= temppixelX > 1023 ? 1023 : temppixelX;
    pixelY <= temppixelY > 767  ? 767  : temppixelY;

 //BallY is 0 for front of field and 4096 for back.

    size <= 12'd1400 - (ballY >> 2); //Divide by 4 so it goes in 1024-0 range

 //Make the ball RED at the front and GREEN at the back.
    ball_color <= ballY < 200 ? RED : ballY < 3872 ? WHITE : GREEN;

end
end
```

```
mod_divide Xdivider(.clk(clk),.dividend(unboundedpixelX),.divisor(tempsize),.quotient(temppixel
mod_divide Ydivider(.clk(clk),.dividend(unboundedpixelY),.divisor(tempsize),.quotient(temppixel

endmodule
```

A.36 ps2_mouse_xy.v

```
‘timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    15:17:14 12/04/2007
// Design Name:
// Module Name:    ps2_mouse_xy
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
// ps2_mouse_xy gives a high-level interface to the mouse, which
// keeps track of the "absolute" x,y position (within a parameterized
// range) and also returns button presses.

module ps2_mouse_xy(clk, reset, ps2_clk, ps2_data, mx, my, btn_click);

   input clk, reset;
   inout ps2_clk, ps2_data; // data to/from PS/2 mouse
   output [11:0] mx, my;   // current mouse position, 12 bits
   output [2:0]  btn_click; // button click: Left-Middle-Right

   // module parameters
   parameter   MAX_X = 1023;
   parameter   MAX_Y = 767;

   // low level mouse driver

   wire [8:0]   dx, dy;
```

```
   wire [2:0]   btn_click;
   wire   data_ready;
   wire   error_no_ack;
   wire [1:0]   ovf_xy;
   wire   streaming;

   ps2_mouse m1(clk,reset,ps2_clk,ps2_data,dx,dy,ovf_xy, btn_click,
data_ready,streaming);

   // Update "absolute" position of mouse

   reg [11:0]  mx, my;
   wire        sx = dx[8]; // signs
   wire        sy = dy[8];
   wire [8:0]  ndx = sx ? {0,~dx[7:0]}+1 : {0,dx[7:0]}; // magnitudes
   wire [8:0]  ndy = sy ? {0,~dy[7:0]}+1 : {0,dy[7:0]};

   always @(posedge clk) begin
      mx <= reset ? 0 :
    data_ready ? (sx ? (mx>ndx ? mx - ndx : 0)
  : (mx < MAX_X - ndx ? mx+ndx : MAX_X)) : mx;
      // note Y is flipped for video cursor use of mouse
      my <= reset ? 0 :
    data_ready ? (sy ? (my < MAX_Y - ndy ? my+ndy : MAX_Y)
  : (my>ndy ? my - ndy : 0))  : my;
//     data_ready ? (sy ? (my>ndy ? my - ndy : 0)
//   : (my < MAX_Y - ndy ? my+ndy : MAX_Y)) : my;
   end

endmodule

//////////////////////////////////////////////////////////////////////////////////
//  PS/2 MOUSE
//
//  6.111 Fall 2005
//
//  NOTE:  make sure to change the mouse ports (mouse_clock, mouse_data) to
// bi-directional 'inout' ports in the top-level module
//
//  specifically, labkit.v should have the line
//
//     inout  mouse_clock, mouse_data;
//
// This module interfaces to a mouse connected to the labkit's PS2 port.
// The outputs provided give dx and dy movement values (9 bits 2's comp)
```

```
// and three button click signals.
//
// NOTE: change the following parameters for a different system clock
// (current configuration for 50 MHz clock)
// CLK_HOLD : 100 usec hold to bring PS2_CLK low
//   RCV_WATCHDOG_TIMER_VALUE : (PS/2 RECEIVER) 2 msec count
//   RCV_WATCHDOG_TIMER_BITS  :   bits needed for timer
//   INIT_TIMER_VALUE : (INIT process) 0.5 sec count
//  INIT_TIMER_BITS :  bits needed for timer
////////////////////////////////////////////////////////////////////////////////
//
// Nov-8-2005: Registered the outputs (dout_dx, dout_dy, ovf_xy, btn_click) in [ps2_mouse]
// Added output "streaming"
// Nov-9-2005: synchronized ps2_clk to local clock for transmitter in [ps2_interface]
// Programmed watchdog_timer for [ps2] (receiver module)
//      Programmed init_timer for [ps2_mouse] (resets initialization)
////////////////////////////////////////////////////////////////////////////////

module ps2_mouse(clock, reset, ps2_clk, ps2_data, dout_dx,
 dout_dy, ovf_xy, btn_click, ready, streaming);

  input clock, reset;
  inout ps2_clk, ps2_data; //data to/from PS/2 mouse
  output [8:0] dout_dx, dout_dy; //9-bit 2's compl, indicates movement of mouse
  output [1:0] ovf_xy; //==1 if overflow: dx, dy
  output [2:0] btn_click; //button click: Left-Middle-Right
  output ready; //synchronous 1 cycle ready flag
  output streaming; //==1 if mouse is in stream mode

  ////////////////////////////////////////////////////////
  // PARAMETERS
  // # of cycles for clock=50 MHz
  parameter CLK_HOLD = 5000; //100 usec hold to bring PS2_CLK low
  parameter RCV_WATCHDOG_TIMER_VALUE = 100000;  // For PS/2 RECEIVER : # of sys_clks for 2msec
  parameter RCV_WATCHDOG_TIMER_BITS  = 17;     //   : bits needed for timer
  parameter INIT_TIMER_VALUE = 25000000; // For INIT process : sys_clks for 0.5 sec.(SELF-TEST
  parameter INIT_TIMER_BITS  = 28;     //       : bits needed for timer
  ////////////////////////////////////////////////////////
  wire reset_init_timer;


  ////////////////////////////////////////////////////////
  //CONTROLLER:
  //-initialization process:
  // Host:  FF  Reset command
```

```
  // Mouse: FA  Acknowledge
  // Mouse: AA  Self-test passed
  // Mouse: 00  Mouse ID
  // Host:  F4  Enable
  // Mouse: FA  Acknowledge
  parameter SND_RESET = 0,  RCV_ACK1 = 1,  RCV_STEST = 2, RCV_ID = 3;
  parameter SND_ENABLE =4,  RCV_ACK2 = 5,  STREAM = 6;
  reg [2:0] state;

  wire send, ack;
  wire [7:0] packet;
  wire [7:0] curkey;
  wire key_ready;

  //NOTE: no support for scrolling wheel, extra buttons
  always @(posedge clock) begin
    if (reset || reset_init_timer) state <= SND_RESET;
    else case (state)
       SND_RESET: state <= ack ? RCV_ACK1 : state;
       RCV_ACK1: state <= (key_ready && curkey==8'hFA) ? RCV_STEST : state;
       RCV_STEST: state <= (key_ready && curkey==8'hAA) ? RCV_ID : state;
       RCV_ID: state <= (key_ready) ? SND_ENABLE : state; //any device type
       SND_ENABLE: state <= ack ? RCV_ACK2 : state;
       RCV_ACK2: state <= (key_ready && curkey==8'hFA) ? STREAM :state;
       STREAM: state <= state;
  default: state <= SND_RESET;
    endcase
  end

  assign send = (state==SND_RESET) || (state==SND_ENABLE);
  assign packet = (state==SND_RESET) ? 8'hFF :
      (state==SND_ENABLE) ? 8'hF4 :
    8'h00;
  assign streaming = (state==STREAM);


  // Connect PS/2 interface module
  ps2_interface ps2_mouse(.reset(reset), .clock(clock),
   .ps2c(ps2_clk), .ps2d(ps2_data),
.send(send), .snd_packet(packet), .ack(ack),
.rcv_packet(curkey), .key_ready(key_ready)  );
  defparam ps2_mouse.CLK_HOLD   = CLK_HOLD;
  defparam ps2_mouse.WATCHDOG_TIMER_VALUE = RCV_WATCHDOG_TIMER_VALUE;
  defparam ps2_mouse.WATCHDOG_TIMER_BITS  = RCV_WATCHDOG_TIMER_BITS;
```

```
  ////////////////////////////////////////////////////////
  // DECODER
  //http://www.computer-engineering.org/ps2mouse/
  //  bit-7    3        bit-0
  //Byte 1:  Y-ovf  X-ovf  Y-sign  X-sign  1  Btn-M  Btn-R  Btn-L
  //Byte 2:  X movement
  //Byte 3:  Y movement
  reg [1:0] bindex, old_bindex;
  reg [7:0] status, dx, dy; //temporary storage of mouse status
  reg [8:0] dout_dx, dout_dy; //Clock the outputs
  reg [1:0] ovf_xy;
  reg [2:0] btn_click;
  wire ready;

  always @(posedge clock) begin
   if (reset) begin
  bindex <= 0;
  status <= 0;
  dx <= 0;
  dy <= 0;
end else if (key_ready && state==STREAM) begin
  case (bindex)
   2'b00: status <= curkey;
2'b01: dx <= curkey;
2'b10: dy <= curkey;
default: status <= curkey;
  endcase

  bindex <= (bindex==2'b10) ? 0 : bindex + 1;
       if (bindex == 2'b10) begin  //Now, dy is ready
   dout_dx   <= {status[4], dx};  //2's compl 9-bit
      dout_dy   <= {status[5], curkey};  //2's compl 9-bit
      ovf_xy    <= {status[6], status[7]};  //overflow: x, y
      btn_click <= {status[0], status[2], status[1]};  //button click: Left-Middle-Right
       end
end //end else-if (key_ready)
  end

  always @(posedge clock)
    old_bindex <= bindex;

  assign ready = (bindex==2'b00) && old_bindex==2'b10;

  ////////////////////////////////////////////////////////
  // INITIALIZATION TIMER
```

```
  //  ==> RESET if processs hangs during initialization
  reg [INIT_TIMER_BITS-1:0] init_timer_count;
  assign reset_init_timer = (state != STREAM) && (init_timer_count==INIT_TIMER_VALUE-1);
  always @(posedge clock)
  begin
init_timer_count <= (reset || reset_init_timer || state==STREAM) ?
0 : init_timer_count + 1;
  end

endmodule



////////////////////////////////////////////////////////////////////////
// PS/2 INTERFACE: transmit or receive data from ps/2

module ps2_interface(reset, clock, ps2c, ps2d, send, snd_packet, ack, rcv_packet, key_ready);
   input clock,reset;
   inout ps2c; // ps2 clock (BI-DIRECTIONAL)
   inout ps2d; // ps2 data (BI-DIRECTIONAL)
   input send; //flag: send packet _
   output ack; // end of transmission  | for transmitting
   input [7:0] snd_packet; // data packet to send to PS/2 _|
   output [7:0] rcv_packet; //packet received from PS/2 _
   output key_ready; // new data ready (rcv_packet) _| for receiving

   ////////////////////////////////////////////////////////////////////////
   // MAIN CONTROL
   ////////////////////////////////////////////////////////////////////////
   parameter CLK_HOLD = 5005; //hold PS2_CLK low for 100 usec (@50 Mhz)
   parameter WATCHDOG_TIMER_VALUE = 100000;  // Number of sys_clks for 2msec. _
   parameter WATCHDOG_TIMER_BITS  = 17;     // Number of bits needed for timer _| for RECEIVER

   wire serial_dout; //output (to ps2d)
   wire rcv_ack; //ACK from ps/2 mouse after data transmission

   wire we_clk, we_data;

   assign ps2c = we_clk  ? 0 : 1'bZ;
   assign ps2d = we_data ? serial_dout : 1'bZ;

   assign ack = rcv_ack;


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

```
   // TRANSMITTER MODULE
   ////////////////////////////////////////////////////////////////////////////////

   ////////////////////////////////////////////////////////
   // COUNTER: 100 usec hold
   reg [15:0] counter;
   wire en_cnt;
   wire cnt_ready;
   always @(posedge clock) begin
     counter <= reset ? 0 :
en_cnt ? counter+1 :
0;
   end
   assign cnt_ready = (counter>=CLK_HOLD);

   ////////////////////////////////////////////////////////
   // SEND DATA
   // hold CLK low for at least 100 usec
   // DATA low
   // Release CLK
   // (on negedge of ps2_clock) - device brings clock LOW
   // REPEAT: SEND data
   // Release DATA
   // Wait for device to bring DATA low
   // Wait for device to bring CLK low
   // Wait for device to release CLK, DATA
   reg [3:0] index;

   // synchronize PS2 clock to local clock and look for falling edge
   reg [2:0] ps2c_sync;
   always @ (posedge clock) ps2c_sync <= {ps2c_sync[1:0],ps2c};
   wire falling_edge = ps2c_sync[2] & ~ps2c_sync[1];

   always @(posedge clock) begin
    if (reset) begin
index <= 0;
end
else if (falling_edge) begin //falling edge of ps2c
   if (send) begin //transmission mode
if (index==0)
index <= cnt_ready ? 1 : 0; //index=0: CLK low
else
index <= index + 1; //index=1: snd_packet[0], =8: snd_packet[7],
//      9: odd parity, =10: stop bit
//   11: wait for ack
```

```
   end else
    index <= 0;
end else
   index <= (send) ? index : 0;
   end

   assign en_cnt = (index==0 && ~reset && send);
   assign serial_dout = (index==0 && cnt_ready) ? 0 : //bring DATA low before releasing CLK
(index>=1 && index <=8) ? snd_packet[index-1] :
(index==9) ? ~(^snd_packet) : //odd parity
1; //including last '1' stop bit

   assign we_clk = (send && !cnt_ready && index==0); //Enable when counter is counting up
   assign we_data = (index==0 && cnt_ready) || (index>=1 && index<=9);//Enable after 100usec C
   assign rcv_ack = (index==11 && ps2d==0); //use to reset RECEIVER module


   ////////////////////////////////////////////////////////////////////////////
   // RECEIVER MODULE
   ////////////////////////////////////////////////////////////////////////////
   reg [7:0]   rcv_packet; // current keycode
   reg key_ready; // new data
   wire        fifo_rd; // read request
   wire [7:0]  fifo_data; // data from mouse
   wire        fifo_empty; // flag: no data
   //wire        fifo_overflow; // keyboard data overflow

   assign      fifo_rd = ~fifo_empty; // continuous read

   always @(posedge clock)
   begin
  // get key if ready
  rcv_packet <= ~fifo_empty ? fifo_data : rcv_packet;
key_ready  <= ~fifo_empty;
   end

   ////////////////////////////////////////////////////
   // connect ps2 FIFO module
   reg [WATCHDOG_TIMER_BITS-1 : 0] watchdog_timer_count;
   wire [3:0] rcv_count; //count incoming data bits from ps/2 (0-11)

   wire watchdog_timer_done = watchdog_timer_count==(WATCHDOG_TIMER_VALUE-1);
   always @(posedge clock)
   begin
    if (reset || send || rcv_count==0) watchdog_timer_count <= 0;
```

```
   else if (~watchdog_timer_done)
     watchdog_timer_count <= watchdog_timer_count + 1;
   end

   ps2 ps2_receiver(.clock(clock), .reset(!send && (reset || rcv_ack) ),//RESET on reset or End
   .ps2c(ps2c), .ps2d(ps2d),
.fifo_rd(fifo_rd), .fifo_data(fifo_data),//in1, out8
.fifo_empty(fifo_empty) , .fifo_overflow(),//out1, out1
.watchdog(watchdog_timer_done), .count(rcv_count) );

endmodule



////////////////////////////////////////////////////////////////////////
// PS/2 FIFO receiver module (from 6.111 Fall 2004)

module ps2(reset, clock, ps2c, ps2d, fifo_rd, fifo_data, fifo_empty,fifo_overflow, watchdog, co
  input clock,reset,watchdog,ps2c,ps2d;
  input fifo_rd;
  output [7:0] fifo_data;
  output fifo_empty;
  output fifo_overflow;
  output [3:0] count;

  reg [3:0] count;      // count incoming data bits
  reg [9:0] shift;      // accumulate incoming data bits

  reg [7:0] fifo[7:0];   // 8 element data fifo
  reg fifo_overflow;
  reg [2:0] wptr,rptr;   // fifo write and read pointers

  wire [2:0] wptr_inc = wptr + 1;

  assign fifo_empty = (wptr == rptr);
  assign fifo_data = fifo[rptr];

  // synchronize PS2 clock to local clock and look for falling edge
  reg [2:0] ps2c_sync;
  always @ (posedge clock) ps2c_sync <= {ps2c_sync[1:0],ps2c};
  wire sample = ps2c_sync[2] & ~ps2c_sync[1];

  reg timeout;
  always @ (posedge clock) begin
    if (reset) begin
```

```
      count <= 0;
      wptr <= 0;
      rptr <= 0;
      timeout <= 0;
      fifo_overflow <= 0;
    end else if (sample) begin
      // order of arrival: 0,8 bits of data (LSB first),odd parity,1
      if (count==10) begin
        // just received what should be the stop bit
        if (shift[0]==0 && ps2d==1 && (^shift[9:1])==1) begin
          fifo[wptr] <= shift[8:1];
          wptr <= wptr_inc;
  fifo_overflow <= fifo_overflow | (wptr_inc == rptr);
        end
        count <= 0;
        timeout <= 0;
      end else begin
        shift <= {ps2d,shift[9:1]};
        count <= count + 1;
      end
    end else if (watchdog && count!=0) begin
      if (timeout) begin
        // second tick of watchdog while trying to read PS2 data
count <= 0;
        timeout <= 0;
      end else timeout <= 1;
    end

    // bump read pointer if we're done with current value.
    // Read also resets the overflow indicator
    if (fifo_rd && !fifo_empty) begin
      rptr <= rptr + 1;
      fifo_overflow <= 0;
    end
  end
endmodule
```

A.37 render_screen.v

```
////////////////////////////////////////////////////////////////////////////////
//
// render_screen: draws the game on the screen with the static background and
//                the ball
//
```

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

module render_screen (vclock,reset,
  hcount,vcount,hsync,vsync,
  pixel, ballpixelX_in, ballpixelY_in, ballsize_in, ball_color_in, paddleminX_in, paddleminY_in
  //paddleminX_OPP,paddleminY_OPP,paddlemaxX_OPP,paddlemaxY_OPP,ballpixelX_opp,ballpixelY_opp)

input vclock; // 65MHz clock
   input reset; // 1 to initialize module
   input [10:0] hcount; // horizontal index of current pixel (0..1023)
   input [9:0]  vcount; // vertical index of current pixel (0..767)
   input hsync; // XVGA horizontal sync signal (active low)
   input vsync; // XVGA vertical sync signal (active low)


input [10:0] ballpixelX_in, paddleminX_in, paddlemaxX_in;
input [9:0] ballpixelY_in, paddleminY_in, paddlemaxY_in;
input [10:0] ballsize_in;
input [2:0] ball_color_in;

input [3:0] hit_wall_in;

reg [3:0] hit_wall;

reg [10:0] ballpixelX, paddleminX, paddlemaxX;
reg [9:0] ballpixelY, paddleminY, paddlemaxY;
reg [10:0] ballsize;
reg [2:0] ball_color;

   output [2:0] pixel;

reg last_vsync;

//This is sampling at vsync = 0 and last_vsync = 1,
//a falling edge

always @ (posedge vclock) begin
if(reset) begin
last_vsync <= vsync;
ballpixelX <= ballpixelX_in;
ballpixelY <= ballpixelY_in;
paddleminX <= paddleminX_in;
paddlemaxX <= paddlemaxX_in;
paddleminY <= paddleminY_in;
paddlemaxY <= paddlemaxY_in;
```

```
ballsize <= ballsize_in;
ball_color <= ball_color_in;
hit_wall <= hit_wall_in;
end else begin
last_vsync <= vsync;

if(vsync == 0 && last_vsync == 1) begin
ballpixelX <= ballpixelX_in;
ballpixelY <= ballpixelY_in;
paddleminX <= paddleminX_in;
paddlemaxX <= paddlemaxX_in;
paddleminY <= paddleminY_in;
paddlemaxY <= paddlemaxY_in;
ballsize <= ballsize_in;
ball_color <= ball_color_in;
hit_wall <= hit_wall_in;
end
end
end

wire [2:0] background_pixel, ball_pixel, paddle_pixel; //otherpaddle_pixel, otherball_pixel;

//wire [2:0] otherpaddle_pixel, otherball_pixel;

circle puck(ballpixelX,ballpixelY,ballsize,hcount,vcount,ball_pixel,ball_color,vclock);

//circle pucktwo(ballpixelX_opp,ballpixelY_opp,ballsize,hcount,vcount,otherball_pixel,ball_colo

background mybackground(hcount,vcount,background_pixel, vclock, hit_wall);

transparentblob paddle(paddleminX,paddleminY,paddlemaxX, paddlemaxY,hcount,vcount,paddle_pixel)

//transparentblob otherpaddle(paddleminX_OPP,paddleminY_OPP,paddlemaxX_OPP, paddlemaxY_OPP,hcou
//defparam otherpaddle.COLOR = 3'b001;  // default color: white
//The following gives priority to paddle, then ball, then background.

assign pixel = paddle_pixel == 0 ? ball_pixel == 0 ? background_pixel : ball_pixel : paddle_pi

//assign pixel = paddle_pixel | ball_pixel | background_pixel | otherpaddle_pixel | otherball_p


endmodule
```

A.38 renderscreen_input_buffer.v

```
//Synchronize the various signals going to render the screen
module render_screen_input_buffer(vclock, logic_clock, ballpixelX_in, ballpixelY_in, ballsize_i
 ballpixelX_out, ballpixelY_out, ballsize_out, ball_color_out, paddleminX_out, paddleminY_out,
input logic_clock, vclock;

input [10:0] ballpixelX_in, paddleminX_in, paddlemaxX_in;
input [9:0] ballpixelY_in, paddleminY_in, paddlemaxY_in;
input [10:0] ballsize_in;
input [2:0] ball_color_in;

input [3:0] hit_wall_in;

output [10:0] ballpixelX_out, paddleminX_out, paddlemaxX_out;
output [9:0] ballpixelY_out, paddleminY_out, paddlemaxY_out;
output [10:0] ballsize_out;
output [2:0] ball_color_out;

output [3:0] hit_wall_out;

synchronizer11 sync1(logic_clock, vclock, ballpixelX_in,ballpixelX_out);
synchronizer11 sync2(logic_clock, vclock, paddleminX_in,paddleminX_out);
synchronizer11 sync3(logic_clock, vclock, paddlemaxX_in,paddlemaxX_out);
synchronizer11 sync4(logic_clock, vclock, ballsize_in,ballsize_out);
synchronizer10 sync5(logic_clock, vclock, ballpixelY_in,ballpixelY_out);
synchronizer10 sync6(logic_clock, vclock, paddleminY_in,paddleminY_out);
synchronizer10 sync7(logic_clock, vclock, paddlemaxY_in,paddlemaxY_out);

synchronizer3 sync8(logic_clock, vclock, ball_color_in,ball_color_out);
synchronizer4 sync9(logic_clock, vclock, hit_wall_in, hit_wall_out);

endmodule
```

A.39 rs232c_receiver.v

```
// this module listens to rxdin port, which is the received data port of rs232c
// when 8-bit data comes in and finish with a stop bit,
// ready goes high for one clock cycle
// and the received data appears on the data port.

// clk : 10MHz clock

// Baud rate is set to 115200
// Parity : none
```

```
// Data bits : 8
// Stop bit : 1
// Flow control : None

module rs232c_receiver(clk, rxdin, data, ready);
input clk;
////////////////////// receive port ////////////////////////////
input rxdin; //
output [7:0] data; // read data
output ready; // read data ready


wire rxd;
parameter baudrate = 115200; // baud rate
parameter clkspeed = 27000000; // clock speed  //modified by Zach because clock is 27MHz

sync sync0 (clk, ~rxdin, rxd);

reg [2:0] count; // sample counter
reg cout;

reg countexp;
reg [7:0] data;
reg ready, creset;

reg [1:0] state;

parameter s_wait = 3; // wait for data to start
parameter s_sampling = 1; // sampling data
parameter s_stop = 2; // wait for stop bit
parameter s_idle = 0; // idle, wait for start bit.

wire exp, even;
serialclk sclk(clk, creset, exp, even);
defparam sclk.baudrate = baudrate;
defparam sclk.clkspeed = clkspeed;
reg sampling_go;

always @(posedge clk) begin
ready <= (state == s_stop) & (even & exp);
creset <= (state == s_idle);
case (state)
s_idle : state <= rxd ? s_wait : state;
s_wait : state <= exp ? s_sampling : state;
s_sampling : state <= cout ? s_stop : state;
```

```
s_stop : state <= (exp & ~even) ? s_idle : state;
endcase

sampling_go <= ((state == s_sampling) & exp & ~even);
data <=  sampling_go ? {~rxd,data[7:1]} : data;
{cout, count} <= ready ? 0 : (sampling_go) ? count+1 : count;
end

endmodule
```

A.40 rs232c_transmitter.v

```
// when start goes high, the module samples data into a buffer.
// on the next clock cycle,
// busy goes high and start transmiting the buffer content into txdout using rs232c protocol.
// when the transmission is complete, busy comes low, and the module is ready to transmit agai

// clk : 10MHz clock

// Baud rate is set to 115200
// Parity : none
// Data bits : 8
// Stop bit : 1
// Flow control : None

module rs232c_transmitter(clk,
txdout, data, start, busy);
input clk;

/////// transmit port //////// //////
output txdout; // rs232c txd
input [7:0] data; // to be transmitted data
input start; // start transmission, samples data simultaneously
output busy; // transmitting...

parameter baudrate = 115200; // baud rate
parameter clkspeed = 27000000; // clock speed  //Changed by Zach because clock is 10125

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

reg [1:0] state;
parameter s_idle = 0;
parameter s_wait = 1;
parameter s_sampling = 2;
parameter s_stop = 3;
```

```
reg [2:0] count;
reg cout;

reg [7:0] buff;
reg txd;

wire exp, even;
reg go,sampling_go;

wire busy;

serialclk sclk(clk, ~busy, exp, even);
defparam sclk.baudrate = baudrate;
defparam sclk.clkspeed = clkspeed;

assign busy = ~(state == s_idle);

always @(posedge clk) begin
// busy <= ~(state == s_idle);
case (state)
s_idle : state <= start ? s_wait : state;
s_wait : state <= (exp & even) ? s_sampling : state;
s_sampling : state <= cout ? s_stop : state;
default : state <= (exp & even) ? s_idle : state;
endcase
case (state)
s_wait : txd <= 1;
s_sampling : txd <= ~buff[0];
default : txd <= 0;
endcase
sampling_go <= ((s_sampling == state) & exp & even);
buff <= sampling_go ? {1'b0, buff[7:1]} : start ? data :  buff;
{cout, count} <= ~busy ? 0 : sampling_go ? count+1 : count;
end
assign txdout = ~txd;
endmodule
```

A.41 serialclk.v

```
// 'exp' goes high for one clock cycle every (serial data bit period)/2.
// 'even' is high when exp expired even times
module serialclk(clk, reset, exp, even);
parameter baudrate = 115200;
parameter clkspeed = 10000000;
parameter div = 1024; // to save logic elements.
```

```
parameter inc = baudrate*2/div; // increment counter
parameter thd = clkspeed/div; // threshold counter
input clk, reset;
output exp, even;
reg [16:0] count;
reg exp, even;
always @(posedge clk) begin
if (reset) begin
even <= 0;
exp <= 0;
count <= 0;
end else begin
if(~exp) begin
exp <= (count >= thd-1);
count <= count+inc;
end else if (exp) begin
exp <= 0;
count <= (count + inc) - thd;
even <= ~even;
end
end
end
endmodule
```

A.42 serialclock.v

```
////////////////////////////////////////////////////////////////////////////////
// Copyright (c) 1995-2006 Xilinx, Inc.  All rights reserved.
////////////////////////////////////////////////////////////////////////////////
//   ____  ____
//  /   /\/   /
// /___/  \  /    Vendor: Xilinx
// \   \   \/     Version : 8.2.03i
//  \   \         Application : xaw2verilog
//  /   /         Filename : serialclock.v
// /___/   /\     Timestamp : 11/14/2007 15:46:34
// \   \  /  \
//  \___\/\___\
//
//Command: xaw2verilog -intstyle C:/zacharyc/finalproject/serialclock.xaw -st serialclock.v
//Design Name: serialclock
//Device: xc2v6000-4bf957
//
// Module serialclock
```

```
// Generated by Xilinx Architecture Wizard
// Written for synthesis tool: XST
`timescale 1ns / 1ps

module serialclock(CLKIN_IN,
                   CLKDV_OUT,
                   CLKIN_IBUFG_OUT,
                   CLK0_OUT,
                   LOCKED_OUT);

    input CLKIN_IN;
   output CLKDV_OUT;
   output CLKIN_IBUFG_OUT;
   output CLK0_OUT;
   output LOCKED_OUT;

   wire CLKDV_BUF;
   wire CLKFB_IN;
   wire CLKIN_IBUFG;
   wire CLK0_BUF;
   wire GND1;

   assign GND1 = 0;
   assign CLKIN_IBUFG_OUT = CLKIN_IBUFG;
   assign CLK0_OUT = CLKFB_IN;
   BUFG CLKDV_BUFG_INST (.I(CLKDV_BUF),
                         .O(CLKDV_OUT));
   IBUFG CLKIN_IBUFG_INST (.I(CLKIN_IN),
                           .O(CLKIN_IBUFG));
   BUFG CLK0_BUFG_INST (.I(CLK0_BUF),
                        .O(CLKFB_IN));
   DCM DCM_INST (.CLKFB(CLKFB_IN),
                 .CLKIN(CLKIN_IBUFG),
                 .DSSEN(GND1),
                 .PSCLK(GND1),
                 .PSEN(GND1),
                 .PSINCDEC(GND1),
                 .RST(GND1),
                 .CLKDV(CLKDV_BUF),
                 .CLKFX(),
                 .CLKFX180(),
                 .CLK0(CLK0_BUF),
                 .CLK2X(),
                 .CLK2X180(),
                 .CLK90(),
```

```
                .CLK180(),
                .CLK270(),
                .LOCKED(LOCKED_OUT),
                .PSDONE(),
                .STATUS());
   defparam DCM_INST.CLK_FEEDBACK = "1X";
   defparam DCM_INST.CLKDV_DIVIDE = 6.5;
   defparam DCM_INST.CLKFX_DIVIDE = 1;
   defparam DCM_INST.CLKFX_MULTIPLY = 4;
   defparam DCM_INST.CLKIN_DIVIDE_BY_2 = "FALSE";
   defparam DCM_INST.CLKIN_PERIOD = 15.3846;
   defparam DCM_INST.CLKOUT_PHASE_SHIFT = "NONE";
   defparam DCM_INST.DESKEW_ADJUST = "SYSTEM_SYNCHRONOUS";
   defparam DCM_INST.DFS_FREQUENCY_MODE = "LOW";
   defparam DCM_INST.DLL_FREQUENCY_MODE = "LOW";
   defparam DCM_INST.DUTY_CYCLE_CORRECTION = "FALSE";
   defparam DCM_INST.FACTORY_JF = 16'hC080;
   defparam DCM_INST.PHASE_SHIFT = 0;
   defparam DCM_INST.STARTUP_WAIT = "FALSE";
endmodule
```

A.43 serialtest.v

```
// receives a RS232C data from rxd and outputs the data to io[7:0]
// and also transmits data+1 to txd.
// clk : 10Mhz
// Baud rate is set to 115200
// Parity : none
// Data bits : 8
// Stop bit : 1
// Flow control : None
module serialtest(clk,
io,
txd,
rxd,
reset
);
////////////////////////////
input clk;
output [7:0] io;
output txd;

input rxd;
input reset;
```

```
wire rready;  // receiver ready, data comes out on the next clock cycle
wire [7:0] rdata;  // receiver data
reg tstart; // transmitter start
wire tbusy; // transmitter busy
wire [7:0] tdata; // transmitter data to send, data sampled on the next clock cycle of start g


rs232c_receiver rs232r(.clk(clk),.rxdin( rxd), .data(rdata), .ready(rready));
rs232c_transmitter rs232t(.clk(clk), .txdout(txd), .data(tdata), .start(tstart), .busy(tbusy))
reg [7:0] buff;

assign io = buff; // outputs the received data
assign tdata = rdata+1; // transmitts received data + 1 back.
always @(posedge clk) begin
if (reset) begin
buff <= 0;
tstart <= 0;
end
else begin // when data ready, transmit the data+1 back right away.
buff <= rready ? rdata : buff;
tstart <= rready;
end

end

endmodule
```

A.44 sin2theta.v

```
/*********************************************************************************
*     This file is owned and controlled by Xilinx and must be used              *
*     solely for design, simulation, implementation and creation of             *
*     design files limited to Xilinx devices or technologies. Use               *
*     with non-Xilinx devices or technologies is expressly prohibited           *
*     and immediately terminates your license.                                  *
*                                                                               *
*     XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS"             *
*     SOLELY FOR USE IN DEVELOPING PROGRAMS AND SOLUTIONS FOR                   *
*     XILINX DEVICES.  BY PROVIDING THIS DESIGN, CODE, OR INFORMATION           *
*     AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE, APPLICATION               *
*     OR STANDARD, XILINX IS MAKING NO REPRESENTATION THAT THIS                 *
*     IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT,                   *
*     AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE          *
*     FOR YOUR IMPLEMENTATION.  XILINX EXPRESSLY DISCLAIMS ANY                  *
*     WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE                   *
```

```
*     IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR          *
*     REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF         *
*     INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS         *
*     FOR A PARTICULAR PURPOSE.                                               *
*                                                                             *
*     Xilinx products are not intended for use in life support                *
*     appliances, devices, or systems. Use in such applications are           *
*     expressly prohibited.                                                   *
*                                                                             *
*     (c) Copyright 1995-2006 Xilinx, Inc.                                    *
*     All rights reserved.                                                    *
*******************************************************************************/
// The synopsys directives "translate_off/translate_on" specified below are
// supported by XST, FPGA Compiler II, Mentor Graphics and Synplicity synthesis
// tools. Ensure they are correct for your synthesis tool(s).

// You must compile the wrapper file sin2theta.v when simulating
// the core, sin2theta. When compiling the wrapper file, be sure to
// reference the XilinxCoreLib Verilog simulation library. For detailed
// instructions, please refer to the "CORE Generator Help".

`timescale 1ns/1ps

module sin2theta(
clka,
addra,
douta);


input clka;
input [7 : 0] addra;
output [7 : 0] douta;

// synopsys translate_off

      BLK_MEM_GEN_V1_1 #(
8,// c_addra_width
11,// c_addrb_width
1,// c_algorithm
9,// c_byte_size
0,// c_common_clk
"0",// c_default_data
0,// c_disable_warn_bhv_coll
0,// c_disable_warn_bhv_range
"virtex2",// c_family
```

```
0,// c_has_ena
0,// c_has_enb
0,// c_has_mem_output_regs
0,// c_has_mux_output_regs
0,// c_has_regcea
0,// c_has_regceb
0,// c_has_ssra
0,// c_has_ssrb
"sin2theta.mif",// c_init_file_name
1,// c_load_init_file
3,// c_mem_type
1,// c_prim_type
256,// c_read_depth_a
2048,// c_read_depth_b
8,// c_read_width_a
1,// c_read_width_b
"ALL",// c_sim_collision_check
"0",// c_sinita_val
"0",// c_sinitb_val
0,// c_use_byte_wea
0,// c_use_byte_web
0,// c_use_default_data
1,// c_wea_width
1,// c_web_width
256,// c_write_depth_a
2048,// c_write_depth_b
"WRITE_FIRST",// c_write_mode_a
"WRITE_FIRST",// c_write_mode_b
8,// c_write_width_a
1) // c_write_width_b
inst (
.CLKA(clka),
.ADDRA(addra),
.DOUTA(douta),
.DINA(),
.ENA(),
.REGCEA(),
.WEA(),
.SSRA(),
.CLKB(),
.DINB(),
.ADDRB(),
.ENB(),
.REGCEB(),
.WEB(),
```

```
.SSRB(),
.DOUTB());


// synopsys translate_on

// FPGA Express black box declaration
// synopsys attribute fpga_dont_touch "true"
// synthesis attribute fpga_dont_touch of sin2theta is "true"

// XST black box declaration
// box_type "black_box"
// synthesis attribute box_type of sin2theta is "black_box"

endmodule
```

A.45 sync.v

```
module sync(clk, in, out);
input clk, in;
output out;
reg r1, out;
always @(posedge clk) begin
r1 <= in;
out <= r1;
end
endmodule
```

A.46 synchronizer.v

```
//These modules help synchronize between clock domains and relax load on the router for
//signals that do not have to be fast.


module synchronizer(input_clock, output_clock, inputdata,outputdata);
    input input_clock;
    input output_clock;
    input inputdata;
    output outputdata;

 reg input_reg, middle_reg, output_reg, outputdata;

 always @ (posedge input_clock) begin
```

```
  input_reg <= inputdata;
end
 always @ (posedge output_clock) begin
   middle_reg <= input_reg;
output_reg <= middle_reg;
outputdata <= output_reg;
end

endmodule

module synchronizer11(input_clock, output_clock, inputdata,outputdata);
    input input_clock;
    input output_clock;
    input [10:0] inputdata;
    output [10:0] outputdata;

 reg [10:0] input_reg, middle_reg, output_reg, outputdata;

 always @ (posedge input_clock) begin
  input_reg <= inputdata;
end

 always @ (posedge output_clock) begin
middle_reg <= input_reg;
output_reg <= middle_reg;
outputdata <= output_reg;
end

endmodule

module synchronizer10(input_clock, output_clock, inputdata,outputdata);
    input input_clock;
    input output_clock;
    input [9:0] inputdata;
    output [9:0] outputdata;

 reg [9:0] input_reg, middle_reg, output_reg, outputdata;

 always @ (posedge input_clock) begin
  input_reg <= inputdata;
end

 always @ (posedge output_clock) begin
middle_reg <= input_reg;
output_reg <= middle_reg;
```

```
outputdata <= output_reg;
end

endmodule

module synchronizer3(input_clock, output_clock, inputdata,outputdata);
    input input_clock;
    input output_clock;
    input [2:0] inputdata;
    output [2:0] outputdata;

 reg [2:0] input_reg, middle_reg, output_reg, outputdata;

 always @ (posedge input_clock) begin
  input_reg <= inputdata;
end

 always @ (posedge output_clock) begin
middle_reg <= input_reg;
output_reg <= middle_reg;
outputdata <= output_reg;
end

endmodule

module synchronizer4(input_clock, output_clock, inputdata,outputdata);
    input input_clock;
    input output_clock;
    input [3:0] inputdata;
    output [3:0] outputdata;

 reg [3:0] input_reg, middle_reg, output_reg, outputdata;

 always @ (posedge input_clock) begin
  input_reg <= inputdata;
end

 always @ (posedge output_clock) begin
middle_reg <= input_reg;
output_reg <= middle_reg;
outputdata <= output_reg;
end

endmodule
```

A.47 testclock.v

```
//////////////////////////////////////////////////////////////////////////////
// Copyright (c) 1995-2006 Xilinx, Inc.  All rights reserved.
//////////////////////////////////////////////////////////////////////////////
//   ____  ____
//  /   /\/   /
// /___/  \  /    Vendor: Xilinx
// \   \   \/     Version : 8.2.03i
//  \   \         Application : xaw2verilog
//  /   /         Filename : testclock.v
// /___/   /\     Timestamp : 11/14/2007 15:08:18
// \   \  /  \
//  \___\/\___\
//
//Command: xaw2verilog -intstyle C:/zacharyc/finalproject/testclock.xaw -st testclock.v
//Design Name: testclock
//Device: xc2v6000-4bf957
//
// Module testclock
// Generated by Xilinx Architecture Wizard
// Written for synthesis tool: XST
`timescale 1ns / 1ps

module testclock(CLKIN_IN,
                 RST_IN,
                 CLKDV_OUT,
                 CLKIN_IBUFG_OUT,
                 CLK0_OUT,
                 CLK2X_OUT,
                 LOCKED_OUT);

    input CLKIN_IN;
    input RST_IN;
   output CLKDV_OUT;
   output CLKIN_IBUFG_OUT;
   output CLK0_OUT;
   output CLK2X_OUT;
   output LOCKED_OUT;

   wire CLKDV_BUF;
   wire CLKFB_IN;
   wire CLKIN_IBUFG;
   wire CLK0_BUF;
   wire CLK2X_BUF;
   wire GND1;
```

```
assign GND1 = 0;
assign CLKIN_IBUFG_OUT = CLKIN_IBUFG;
assign CLK0_OUT = CLKFB_IN;
BUFG CLKDV_BUFG_INST (.I(CLKDV_BUF),
                      .O(CLKDV_OUT));
IBUFG CLKIN_IBUFG_INST (.I(CLKIN_IN),
                        .O(CLKIN_IBUFG));
BUFG CLK0_BUFG_INST (.I(CLK0_BUF),
                     .O(CLKFB_IN));
BUFG CLK2X_BUFG_INST (.I(CLK2X_BUF),
                      .O(CLK2X_OUT));
DCM DCM_INST (.CLKFB(CLKFB_IN),
              .CLKIN(CLKIN_IBUFG),
              .DSSEN(GND1),
              .PSCLK(GND1),
              .PSEN(GND1),
              .PSINCDEC(GND1),
              .RST(RST_IN),
              .CLKDV(CLKDV_BUF),
              .CLKFX(),
              .CLKFX180(),
              .CLK0(CLK0_BUF),
              .CLK2X(CLK2X_BUF),
              .CLK2X180(),
              .CLK90(),
              .CLK180(),
              .CLK270(),
              .LOCKED(LOCKED_OUT),
              .PSDONE(),
              .STATUS());
defparam DCM_INST.CLK_FEEDBACK = "1X";
defparam DCM_INST.CLKDV_DIVIDE = 2.5;
defparam DCM_INST.CLKFX_DIVIDE = 1;
defparam DCM_INST.CLKFX_MULTIPLY = 4;
defparam DCM_INST.CLKIN_DIVIDE_BY_2 = "FALSE";
defparam DCM_INST.CLKIN_PERIOD = 37.037;
defparam DCM_INST.CLKOUT_PHASE_SHIFT = "NONE";
defparam DCM_INST.DESKEW_ADJUST = "SYSTEM_SYNCHRONOUS";
defparam DCM_INST.DFS_FREQUENCY_MODE = "LOW";
defparam DCM_INST.DLL_FREQUENCY_MODE = "LOW";
defparam DCM_INST.DUTY_CYCLE_CORRECTION = "TRUE";
defparam DCM_INST.FACTORY_JF = 16'hC080;
defparam DCM_INST.PHASE_SHIFT = 0;
defparam DCM_INST.STARTUP_WAIT = "FALSE";
```

```
endmodule
```

A.48 tilt_calculation.v

```
‘timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    23:16:32 12/02/2007
// Design Name:
// Module Name:    tilt_calculation
// Project Name:
// Target Devices:
// Tool versions:
// Description: Calculates tilt using the width-height ratio of the paddle
// blob. Once the ratio is calculated, using a divider, this is fed into a memory
// that maps ratios to angles. I generated  this memory as a ROM in coregen, and
// created the COE file to suit our paddles’ geometry.
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module tilt_calculation(clk, rst, field_change, min_x, max_x, min_y, max_y,
x_ul, x_ur, x_bl, x_br,
tilt_degrees, tilt_addr, area);
    input clk;
    input rst;
 input field_change;
 input [9:0] min_x, max_x, min_y, max_y;
 input [9:0] x_ul, x_ur, x_bl, x_br;
 output reg signed [7:0] tilt_degrees;
 output [7:0] tilt_addr;
 output [19:0] area;

 wire [7:0] tilt_degrees_pos;
 wire [6:0] tilt_degrees_mem;
 reg [19:0] area;
 reg [10:0] top_width, bottom_width;

 //Tilt Calculation
```

```
reg [5:0] tilt_expt;
wire [5:0] tilt_expt_dirty;
reg [9:0] width, height;
reg [9:0] tilt_man;
wire [9:0] tilt_man_dirty;
wire tilt_sign, tilt_overflow, tilt_underflow;
always @ (posedge clk) begin
if (field_change) begin
width <= max_x - min_x;
height <= max_y - min_y;
tilt_man <= tilt_man_dirty;
tilt_expt <= tilt_expt_dirty;
end
area <= width * height;
top_width <= x_ur - x_ul;
bottom_width <= x_br - x_bl;
tilt_degrees <= (top_width > bottom_width) ?
-tilt_degrees_pos :
tilt_degrees_pos;

end

paddle_divide ratio(clk,
height, 0, 1,//man, sign, expt
width, 0, 1, //man, sign, expt
tilt_man_dirty, tilt_sign, tilt_expt_dirty,
tilt_overflow, tilt_underflow);


assign tilt_addr = {tilt_expt[0],tilt_man[9:3]};
assign tilt_degrees_pos = {1'b0, tilt_degrees_mem};

arccos tilt_mem(tilt_addr, clk, tilt_degrees_mem);

endmodule
```

A.49 transparentblob.v

```
//////////////////////////////////////////////////////////////////////
//
// transparentblob: generate transparent blob on screen
//
//////////////////////////////////////////////////////////////////////
```

```
module transparentblob(minX,minY,maxX,maxY,hcount,vcount,pixel);
   parameter COLOR = 3'b111;  // default color: white

   input [10:0] minX,maxX,hcount;
   input [9:0] minY,maxY,vcount;
   output [2:0] pixel;

   reg [2:0] pixel;
   always @ (minX or minY or maxX or maxY or hcount or vcount) begin
      if ((hcount >= minX && hcount < maxX && hcount[0]) &&
 (vcount >= minY && vcount < maxY && vcount[0]))
pixel = COLOR;
      else pixel = 0;
   end
endmodule
```

A.50 video_decoder.v

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

//////////////////////////////////////////////////////////////////////////
//
// NTSC decode - 16-bit CCIR656 decoder
// By Javier Castro
// This module takes a stream of LLC data from the adv7185
// NTSC/PAL video decoder and generates the corresponding pixels,
// that are encoded within the stream, in YCrCb format.

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

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

   // clk - line-locked clock (in this case, LLC1 which runs at 27Mhz)
   // reset - system reset
```

```
// tv_in_ycrcb - 10-bit input from chip. should map to pins [19:10]
// ycrcb - 24 bit luminance and chrominance (8 bits each)
// f - field: 1 indicates an even field, 0 an odd field
// v - vertical sync: 1 means vertical sync
// h - horizontal sync: 1 means horizontal sync

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

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




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

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

// The data stream looks as follows
// SAV_FF | SAV_00 | SAV_00 | SAV_XY | Cb0 | Y0 | Cr1 | Y1 | Cb2 | Y2 | ... | EAV sequence
```

```
   // There are two things we need to do:
   //   1. Find the two SAV blocks (stands for Start Active Video perhaps?)
   //   2. Decode the subsequent data

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

   assign  state = current_state;

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

  end
else
  begin
     // these states don't do much except allow us to know where we are in the stream.
     // whenever the synchronization code is seen, go back to the sync_state before
     // transitioning to the new state
     case (current_state)
       SYNC_1: current_state <= (tv_in_ycrcb == 10'h000) ? SYNC_2 : SYNC_1;
       SYNC_2: current_state <= (tv_in_ycrcb == 10'h000) ? SYNC_3 : SYNC_1;
       SYNC_3: current_state <= (tv_in_ycrcb == 10'h200) ? SAV_f1_cb0 :
(tv_in_ycrcb == 10'h274) ? EAV_f1 :
(tv_in_ycrcb == 10'h2ac) ? SAV_VBI_f1 :
(tv_in_ycrcb == 10'h2d8) ? EAV_VBI_f1 :
(tv_in_ycrcb == 10'h31c) ? SAV_f2_cb0 :
(tv_in_ycrcb == 10'h368) ? EAV_f2 :
(tv_in_ycrcb == 10'h3b0) ? SAV_VBI_f2 :
(tv_in_ycrcb == 10'h3c4) ? EAV_VBI_f2 : SYNC_1;

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

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

       // These states are here in the event that we want to cover these signals
```

```
      // in the future. For now, they just send the state machine back to SYNC_1
      EAV_f1: current_state <= SYNC_1;
      SAV_VBI_f1: current_state <= SYNC_1;
      EAV_VBI_f1: current_state <= SYNC_1;
      EAV_f2: current_state <= SYNC_1;
      SAV_VBI_f2: current_state <= SYNC_1;
      EAV_VBI_f2: current_state <= SYNC_1;

    endcase
  end
    end // always @ (posedge clk)

  // implement our decoding mechanism

  wire y_enable;
  wire cr_enable;
  wire cb_enable;

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

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

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

  reg    f = 0;

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

```
endmodule



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

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

`define INPUT_SELECT                            4'h0
  // 0: CVBS on AIN1 (composite video in)
  // 7: Y on AIN2, C on AIN5 (s-video in)
  // (These are the only configurations supported by the 6.111 labkit hardware)
`define INPUT_MODE                              4'h0
  // 0: Autodetect: NTSC or PAL (BGHID), w/o pedestal
  // 1: Autodetect: NTSC or PAL (BGHID), w/pedestal
  // 2: Autodetect: NTSC or PAL (N), w/o pedestal
  // 3: Autodetect: NTSC or PAL (N), w/pedestal
  // 4: NTSC w/o pedestal
  // 5: NTSC w/pedestal
  // 6: NTSC 4.43 w/o pedestal
  // 7: NTSC 4.43 w/pedestal
  // 8: PAL BGHID w/o pedestal
  // 9: PAL N w/pedestal
  // A: PAL M w/o pedestal
  // B: PAL M w/pedestal
  // C: PAL combination N
  // D: PAL combination N w/pedestal
  // E-F: [Not valid]

`define ADV7185_REGISTER_0 {`INPUT_MODE, `INPUT_SELECT}

//////////////////////////////////////////////////////////////////////////////
// Register 1
//////////////////////////////////////////////////////////////////////////////
```

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

`define ADV7185_REGISTER_1 {`AUTOMATIC_STARTUP_ENABLE, 1'b0, `BETACAM, `FOUR_TIMES_SAMPLING, `

////////////////////////////////////////////////////////////////////////////
// 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_S

//////////////////////////////////////////////////////////////////////////
// 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_ENA

////////////////////////////////////////////////////////////////////////////
// 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_FL

////////////////////////////////////////////////////////////////////////////
// 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_

////////////////////////////////////////////////////////////////////////////
// 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, `TE

////////////////////////////////////////////////////////////////////////////
// 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_

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

`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.51 virtual_ping_pong.v

```
//
// File:   virtualPingPong.v
// Date:   26-Nov-05
// Author: Mark Stevens <markstev@mit.edu>
//
```

```
// Top level module for Mark and Zach's Virtual Ping Pong final project.
// This file was adapted from zbt_6111_sample, but most of the original
// content has been removed or changed.
//
// Original Documentation:
// Sample code for the MIT 6.111 labkit demonstrating use of the ZBT
// memories for video display.  Video input from the NTSC digitizer is
// displayed within an XGA 1024x768 window.  One ZBT memory (ram0) is used
// as the video frame buffer, with 8 bits used per pixel (black & white).
//
// Since the ZBT is read once for every four pixels, this frees up time for
// data to be stored to the ZBT during other pixel times.  The NTSC decoder
// runs at 27 MHz, whereas the XGA runs at 65 MHz, so we synchronize
// signals between the two (see ntsc2zbt.v) and let the NTSC data be
// stored to ZBT memory whenever it is available, during cycles when
// pixel reads are not being performed.
//
// We use a very simple ZBT interface, which does not involve any clock
// generation or hiding of the pipelining.  See zbt_6111.v for more info.
//
// switch[7] selects between display of NTSC video and test bars
// switch[6] is used for testing the NTSC decoder
// switch[1] selects between test bar periods; these are stored to ZBT
//           during blanking periods
// switch[0] selects vertical test bars (hardwired; not stored in ZBT)

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

////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Template Toplevel Module
//
// For Labkit Revision 004
//
//
// Created: October 31, 2004, from revision 003 file
// Author: Nathan Ickes
//
////////////////////////////////////////////////////////////////////////////
//
// CHANGES FOR BOARD REVISION 004
```

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

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

module virtualPingPong(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;

//wire clock_27mhz_buf;
//BUFG vclk3(.O(clock_27mhz_buf),.I(clock_27mhz));

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

   // Audio Input and Output
```

```
   assign beep= 1'b0;
   assign audio_reset_b = 1'b0;
   assign ac97_synch = 1'b0;
   assign ac97_sdata_out = 1'b0;
/*
*/
   // ac97_sdata_in is an input

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

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

   // SRAMs

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

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

/* enable RAM pins */
```

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

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

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

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

   // Flash ROM
   assign flash_data = 16'hZ;
   assign flash_address = 24'h0;
   assign flash_ce_b = 1'b1;
   assign flash_oe_b = 1'b1;
   assign flash_we_b = 1'b1;
   assign flash_reset_b = 1'b0;
   assign flash_byte_b = 1'b1;
   // flash_sts is an input

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

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

   // LED Displays
/*
   assign disp_blank = 1'b1;
   assign disp_clock = 1'b0;
   assign disp_rs = 1'b0;
   assign disp_ce_b = 1'b1;
   assign disp_reset_b = 1'b0;
```

```
   assign disp_data_out = 1'b0;
*/
   // disp_data_in is an input

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

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

   // Daughtercard Connectors
   assign daughtercard = 44'hZ;

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

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

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

   // use FPGA's digital clock manager to produce a
   // 65MHz clock (actually 64.8MHz)
   wire clock_65mhz_unbuf,clock_65mhz;
wire clock_27;
   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));
BUFG vclk3(.O(clock_27),.I(tv_in_line_clock1));

   wire clk = clock_65mhz;

   // 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, sw0, sw1, sw2, sw3, sw4, sw5, sw6, sw7, up, down;
debounce_block db_all(clk, power_on_reset, button_enter, button_up, button_down,
switch, sw0, sw1, sw2, sw3, sw4, sw5, sw6, sw7, user_reset, up, down);

   assign reset = user_reset | power_on_reset;

   // display module for debugging

   reg [63:0] dispdata, dispdata_q;
always @ (posedge clock_27)
dispdata_q <= dispdata;

   display_16hex hexdisp1(reset, clk, dispdata_q,
  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;

   zbt_6111 zbt1(clk, 1'b1, vram_we, vram_addr,
```

```
   vram_write_data, vram_read_data,
   ram0_clk, 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(clock_27), .reset(reset),//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));

   // code to write NTSC data to video memory

wire [17:0] colorData;


   wire [18:0] ntsc_addr, ntsc_addr_c, ntsc_addr_debug;
   wire [35:0] ntsc_data, ntsc_data_c, ntsc_data_debug;
   wire        ntsc_we, ntsc_we_c, ntsc_we_debug;
wire [17:0] rgb_val, rgb_val_debug;
wire paddle;
wire paddle_filtered;
wire paddle_use;

wire [9:0] x_coord, y_coord, max_x, min_x, max_y, min_y;
reg [9:0] x_filtered, y_filtered;
//wire [2:0] ul_pixel, ur_pixel, bl_pixel, br_pixel, tb_pixel;
//reg [2:0] comb_pixel;
```

```
reg old_field, field_change, y_zero;
//reg signed [11:0] sum, diff;
wire [9:0] x_ul, x_ur, x_bl, x_br, y_ul, y_ur, y_bl, y_br;
//wire [2:0] ul_corner, ur_corner, bl_corner, br_corner;
always @ (posedge clock_27) begin
old_field <= fvh[2];
if (~(fvh[2] == old_field))
field_change <= old_field;
else
field_change <= 0;
y_zero <= (y_coord == 10'd30);
end

wire [7:0] tilt_addr;
wire [7:0] tilt_degrees;
wire [10:0] x_min_converted, x_max_converted;
wire [9:0] y_min_converted, y_max_converted;
wire [19:0] area;
wire [10:0] x_change, y_change, z_change;

isPaddle paddle_detect(clock_27, reset, ycrcb,
sw1, sw2, sw7, sw6, sw5, sw4, sw3, up, down, paddle);

LPF lpf0(clock_27, reset, paddle, fvh[1] | fvh[2], x_coord,
paddle_filtered);

assign paddle_use = switch[0] ? paddle : paddle_filtered;

always @ (posedge clock_27)
begin //ADJUST FOR PIPELINING
x_filtered <= paddle_use ? x_coord : x_filtered;
y_filtered <= paddle_use ? y_coord : y_filtered;
end

paddle_boundaries pb(clock_27, rst, field_change, x_filtered, y_filtered,
min_x, max_x, min_y, max_y, x_ul, x_ur, x_bl,
x_br, y_ul, y_ur, y_bl, y_br);

tilt_calculation tc(clock_27, reset, field_change, min_x, max_x, min_y,
max_y, x_ul, x_ur, x_bl, x_br, tilt_degrees, tilt_addr, area);

//count_area area_counter(clock_27, reset, paddle_use, field_change, area);

forward_velocity z_velocity_calc(clock_27, reset, area[17:9], z_change);
//assign area = 0;
```

```
//assign x_change = 0;
//assign y_change = 0;
lateral_velocity x_velocity_calc(clock_27, reset, x_min_converted[9:0],
x_max_converted[9:0], x_change);

lateral_velocity y_velocity_calc(clock_27, reset, y_min_converted,
y_max_converted, y_change);

//boost velocities based on angle
wire signed [7:0] sin2T, cos2T;
wire signed [10:0] sin2Text, cos2Text;
assign sin2Text = {sin2T[7], sin2T[7], sin2T[7],
sin2T[7], sin2T[7], sin2T[7], sin2T[7],
sin2T[3:0]};
assign cos2Text = {cos2T[7], cos2T[7], cos2T[7],
cos2T[7], cos2T[7], cos2T[7], cos2T[7],
cos2T[3:0]};
reg signed [10:0] y_velocity, z_velocity;
sin2theta sin(clock_27, tilt_degrees, sin2T);
cos2theta cos(clock_27, tilt_degrees, cos2T);
always @ (posedge clock_27) begin
z_velocity <= z_change + cos2Text;
y_velocity <= y_change + sin2Text;
end

coordinate_conversion cv(clock_27, reset, min_x, max_x, min_y, max_y,
x_min_converted, x_max_converted, y_min_converted, y_max_converted);

debugPixels zbtPixels(clock_27, reset, sw7, up, sw1, paddle_use, min_x,
max_x, min_y, max_y,
//x_min_converted, x_max_converted, y_min_converted, y_max_converted,
x_coord, y_coord,
x_ul, x_ur, x_bl, x_br, y_ul, y_ur, y_bl, y_br, ycrcb, rgb_val_debug);

wire [7:0] r_3q, g_3q, b_3q;
YCrCb2RGB ycrcb_convert(r_3q, g_3q, b_3q, clock_27, reset, ycrcb[29:20],
ycrcb[19:10], ycrcb[9:0]);

assign rgb_val = sw1 ? rgb_val_debug : {r_3q[7:2], g_3q[7:2], b_3q[7:2]};

wire logic_clock;
assign logic_clock = clock_27;
wire [2:0] pixel;
//wire serial_out_dummy;
//wire serial_in_dummy;
```

```
assign serial_in_dummy = 1'b1;
//NEED: slow_vsync,
wire slow_vsync;

synchronizer vsync_sync(clock_65mhz,logic_clock,vsync,slow_vsync);
//always @ (posedge clock_27)
// slow_vsync <= vsync;
gamelogic_lump logic(reset,logic_clock,clock_65mhz, switch[7:4],// tilt_degrees,
switch[3],switch[2],vsync, slow_vsync,
hcount,vcount, pixel, rs232_txd,
rs232_rxd, x_min_converted, y_min_converted,
x_max_converted,y_max_converted,
x_change, y_velocity, z_velocity);

/*gamelogic_lump zachpart(reset,logic_clock,clock_65mhz,switch[7:4],
switch[3],switch[2], vsync, hcount,vcount, pixel, blank,
serial_in_dummy, serial_out_dummy, x_min_converted,
y_min_converted,x_max_converted,y_max_converted);*/
//assign pixel = 0;

/*
//wire [8:0] dout_dx, dout_dy;
//wire ovf_xy, ready, streaming;
//wire [2:0] btn_click;
//reg signed [9:0] mouse_x, mouse_y, mouse_x_temp, mouse_y_temp;
//wire [11:0] mx, my;
//reg mouse_row, mouse_col;
reg [5:0] dbp_r, dbp_g, dbp_b;

always @ (posedge clk) begin
/* mouse_x_temp <= mouse_x_temp + {dout_dx[8], dout_dx};
mouse_y_temp <= mouse_y_temp + {dout_dy[8], dout_dy};
if (mouse_x_temp[9]) //overflow
mouse_x <= mouse_x[8] ? 1023 : 0;
else
mouse_x <= mouse_x_temp;
if (mouse_y_temp[9]) //overflow
mouse_y <= mouse_x[8] ? 768 : 0;
else
mouse_y <= mouse_y_temp;*//*
mouse_col = (mx == hcount);
mouse_row = (my == vcount);
dbp_r <= (mouse_col | mouse_row) ? 6'h3f : vr_pixel[17:12];
dbp_g <= (mouse_col | mouse_row) ? 6'h3f : vr_pixel[11:6];
dbp_b <= (mouse_col | mouse_row) ? 6'h3f : vr_pixel[5:0];
```

```
end*/



//ps2_mouse_xy mouse(clk, reset, mouse_clock, mouse_data, mx, my, btn_click);

/*always @ (posedge clock_27) begin
comb_pixel <= sw7 ? (ul_pixel | ur_pixel | bl_pixel | br_pixel)
: (ul_corner | ur_corner | bl_corner | br_corner);
rgb_val <= (paddle_use == 1) ? 18'h3ffff :
(~up) ? {ycrcb[29:24],ycrcb[29:24],ycrcb[29:24]} :
{comb_pixel[2],5'd0,comb_pixel[1],5'd0,comb_pixel[0],5'd0};
end*/

//assign rgb_val = sw1 ? {ycrcb[29:24],ycrcb[29:24],ycrcb[29:24]} :
// sw2 ? {ycrcb[19:14],ycrcb[19:14],ycrcb[19:14]} :
//  {ycrcb[9:4],ycrcb[9:4],ycrcb[9:4]};
   ntsc_to_zbt n2z (clk, clock_27, fvh, dv, //ycrcb[29:22],
 rgb_val,
 ntsc_addr_debug, ntsc_data_debug, ntsc_we_debug, 1,
 x_coord, y_coord);

ntsc_to_zbt_center n2zc(clk, clock_27, fvh, dv,
rgb_val,
ntsc_addr_c, ntsc_data_c, ntsc_we_c);

assign ntsc_addr = ~sw1 ? ntsc_addr_c : ntsc_addr_debug;
assign ntsc_data = ~sw1 ? ntsc_data_c : ntsc_data_debug;
assign ntsc_we   = ~sw1 ? ntsc_we_c   : ntsc_we_debug;

assign analyzer1_data = {6'h0, x_coord[9:0]};
   assign analyzer1_clock = clock_27;
    //ntsc_addr, ntsc_data, ntsc_we, switch[6]);

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

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

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

   wire  sw_ntsc = 1;//~switch[7];
```

```
wire  my_we = sw_ntsc ? (hcount[0]==1) : blank;
   //wire  my_we = sw_ntsc ? (hcount[1:0]==2'd2) : 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 [7:0]  pixel;
reg [7:0]  pixel_r;
reg [7:0]  pixel_g;
reg [7:0]  pixel_b;
   wire  b,hs,vs;

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

wire in_range;
coords_for_center_image center(clk, rst, hcount, vcount, pixel, in_range);



   always @(posedge clk)
     begin
if (switch[1]) begin
pixel_r <= {vr_pixel[17:12],2'b0};
pixel_g <= {vr_pixel[11:6],2'b0};
pixel_b <= {vr_pixel[5:0],2'b0};
end
else begin
pixel_r <= in_range ? {vr_pixel[17:12],2'b0} : {pixel[2],7'b0};
pixel_g <= in_range ? {vr_pixel[11:6],2'b0} : {pixel[1],7'b0};
pixel_b <= in_range ? {vr_pixel[5:0],2'b0} : {pixel[0],7'b0};
end
     end

   // VGA Output.  In order to meet the setup and hold times of the
```

```
   // AD7125, we send it ~clock_65mhz.
   //assign vga_out_red = pixel;
   //assign vga_out_green = pixel;
   //assign vga_out_blue = pixel;
   assign vga_out_red = pixel_r;
   assign vga_out_green = pixel_g;
   assign vga_out_blue = pixel_b;
   assign vga_out_sync_b = 1'b1;    // not used
   assign vga_out_pixel_clock = ~clock_65mhz;
   assign vga_out_blank_b = ~b;
   assign vga_out_hsync = hs;
   assign vga_out_vsync = vs;

   // debugging

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

   always @(posedge clock_27)
     // dispdata <= {vram_read_data,9'b0,vram_addr};
     //dispdata <= {ntsc_data,9'b0,ntsc_addr};
  dispdata <= {4'd0, //blank
1'd0, x_change, //3 chars
//****
4'd0, //blank char
1'd0, y_velocity, //3 chars
//****
4'd0, //blank char
1'd0, z_velocity, //3 chars
//*****
//8'd0, //2 blanks
//4'd0, //blank
//area};  //+ 5 characters
tilt_degrees, 8'b0};//2 chars + 2 blanks
  /*
4'd0, tilt_addr, 4'b0, //tilt_man,//blank + 2 chars + blank

//4'd0, 2'd0, tilt_expt,//blank + 2 chars
//8'd0, //2 blank chars
1'b0, tilt_degrees, 8'b0, //2chars + 2 blanks
6'd0, y_min_converted,//blank + 3 chars
6'd0, y_max_converted};//blank + 3 chars*/

endmodule

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

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

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

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

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

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

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

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

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

```
endmodule

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

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

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

   //wire [18:0]   vram_addr = {1'b0, vcount, hcount[9:2]};
wire [18:0]   vram_addr = {vcount, hcount[9:1]};
//look two spaces ahead to compensate for two cycles of delay

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

assign last_vr_data = (hc2==0) ? vram_read_data : vr_data_latched;
   //always @(posedge clk)
//last_vr_data <= (hc2==0) ? vram_read_data : last_vr_data;
     //last_vr_data <= (hc4==2'd3) ? vr_data_latched : last_vr_data;

   always @(posedge clk)
vr_data_latched <= (hc2==0) ? vram_read_data : vr_data_latched;
     //vr_data_latched <= (hc4==2'd1) ? vram_read_data : vr_data_latched;

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

```
0: vr_pixel = last_vr_data[17:0];
1: vr_pixel = last_vr_data[35:18];
endcase
//     case (hc4)
//       2'd3: vr_pixel = last_vr_data[7:0];
//       2'd2: vr_pixel = last_vr_data[7+8:0+8];
//       2'd1: vr_pixel = last_vr_data[7+16:0+16];
//       2'd0: vr_pixel = last_vr_data[7+24:0+24];
//     endcase

endmodule // vram_display

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

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

   parameter NDELAY = 3;

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

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

endmodule // delayN
```

A.52 xvga.v

```
//////////////////////////////////////////////////////////////////////////////
//
// 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.53 ycrcb2rgb.v

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

```
** Module: ycrcb2rgb
** YCrCb to RGB Converter
** The module from Xilinx converts a 30-bit YCrCb value to a 24-bit RGB
** value. Constants are hardcoded into the conversion.
** NOT WRITTEN BY KEVIN, JON, OR CHRIS
**
**
** 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'b0100101010; //1.164 = 01.00101010
const2 = 10'b0110011000; //1.596 = 01.10011000
const3 = 10'b0011010000; //0.813 = 00.11010000
const4 = 10'b0001100100; //0.392 = 00.01100100
const5 = 10'b1000000100; //2.017 = 10.00000100
end
// Stores the YCrCb colors in registers
// Clears the registers on reset
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
// Creates intermediate values for RGB calculations
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
// Pipelined continuation of the RBG calculation
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
// Assigns RGB values
/* 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.54 zbt_6111.v

```
//
// 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 = ~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
```