Appendix of Verilog Code

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

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

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

`define ADV7185_REGISTER_0 {`INPUT_MODE, `INPUT_SELECT}

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

`define VIDEO_QUALITY                           2'h0
  // 0: Broadcast quality
  // 1: TV quality
  // 2: VCR quality
  // 3: Surveillance quality
`define SQUARE_PIXEL_IN_MODE                    1'b0
  // 0: Normal mode
  // 1: Square pixel mode
`define DIFFERENTIAL_INPUT                      1'b0
  // 0: Single-ended inputs
```

```
  // 1: Differential inputs
`define FOUR_TIMES_SAMPLING                     1'b0
  // 0: Standard sampling rate
  // 1: 4x sampling rate (NTSC only)
`define BETACAM                                 1'b0
  // 0: Standard video input
  // 1: Betacam video input
`define AUTOMATIC_STARTUP_ENABLE                1'b1
  // 0: Change of input triggers reacquire
  // 1: Change of input does not trigger reacquire

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

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

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

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

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

`define INTERFACE_SELECT                        2'h0
  // 0: Philips-compatible
  // 1: Broktree API A-compatible
  // 2: Broktree API B-compatible
  // 3: [Not valid]
`define OUTPUT_FORMAT                           4'h0
  // 0: 10-bit @ LLC, 4:2:2 CCIR656
  // 1: 20-bit @ LLC, 4:2:2 CCIR656
  // 2: 16-bit @ LLC, 4:2:2 CCIR656
  // 3: 8-bit @ LLC, 4:2:2 CCIR656
  // 4: 12-bit @ LLC, 4:1:1
  // 5-F: [Not valid]
  // (Note that the 6.111 labkit hardware provides only a 10-bit interface to
```

```
  // the ADV7185.)
`define TRISTATE_OUTPUT_DRIVERS                 1'b0
  // 0: Drivers tristated when ~OE is high
  // 1: Drivers always tristated
`define VBI_ENABLE                              1'b0
  // 0: Decode lines during vertical blanking interval
  // 1: Decode only active video regions

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

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

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

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

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


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

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

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

`define FIFO_FLAG_MARGIN                        5'h10
```

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

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

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

`define INPUT_CONTRAST_ADJUST                           8'h80

`define ADV7185_REGISTER_8 {`INPUT_CONTRAST_ADJUST}

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

`define INPUT_SATURATION_ADJUST                         8'h8C

`define ADV7185_REGISTER_9 {`INPUT_SATURATION_ADJUST}

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

`define INPUT_BRIGHTNESS_ADJUST                         8'h00

`define ADV7185_REGISTER_A {`INPUT_BRIGHTNESS_ADJUST}

////////////////////////////////////////////////////////////////////////////
//
// Register B
////////////////////////////////////////////////////////////////////////////
//

`define INPUT_HUE_ADJUST                                8'h00

`define ADV7185_REGISTER_B {`INPUT_HUE_ADJUST}

////////////////////////////////////////////////////////////////////////////
//
// Register C
```

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

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

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

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

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

`define ADV7185_REGISTER_D {`DEFAULT_CB_VALUE, `DEFAULT_CR_VALUE}

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

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

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

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

`define POWER_SAVE_CONTROL                      2'h0
  // 0: Full operation
  // 1: CVBS only
```

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

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

////////////////////////////////////////////////////////////////////////////
//
// Register 33
////////////////////////////////////////////////////////////////////////////
//

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

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

`define ADV7185_REGISTER_10 8'h00
`define ADV7185_REGISTER_11 8'h00
`define ADV7185_REGISTER_12 8'h00
`define ADV7185_REGISTER_13 8'h45
`define ADV7185_REGISTER_14 8'h18
```

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

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

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


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

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

   initial begin
```

```
   $display("ADV7185 Initialization values:");
   $display("  Register 0:  0x%X", `ADV7185_REGISTER_0);
   $display("  Register 1:  0x%X", `ADV7185_REGISTER_1);
   $display("  Register 2:  0x%X", `ADV7185_REGISTER_2);
   $display("  Register 3:  0x%X", `ADV7185_REGISTER_3);
   $display("  Register 4:  0x%X", `ADV7185_REGISTER_4);
   $display("  Register 5:  0x%X", `ADV7185_REGISTER_5);
   $display("  Register 7:  0x%X", `ADV7185_REGISTER_7);
   $display("  Register 8:  0x%X", `ADV7185_REGISTER_8);
   $display("  Register 9:  0x%X", `ADV7185_REGISTER_9);
   $display("  Register A:  0x%X", `ADV7185_REGISTER_A);
   $display("  Register B:  0x%X", `ADV7185_REGISTER_B);
   $display("  Register C:  0x%X", `ADV7185_REGISTER_C);
   $display("  Register D:  0x%X", `ADV7185_REGISTER_D);
   $display("  Register E:  0x%X", `ADV7185_REGISTER_E);
   $display("  Register F:  0x%X", `ADV7185_REGISTER_F);
   $display("  Register 33: 0x%X", `ADV7185_REGISTER_33);
end

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

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

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

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

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

assign reset_slow = reset_count != 0;

//
// I2C driver
//

reg load;
```

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

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

   //
   // State machine
   //

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

   always @(posedge clock_slow)
      if (reset_slow)
      begin
         state <= 0;
         load <= 0;
         tv_in_reset_b <= 0;
         old_source <= 0;
      end
      else
      case (state)
        8'h00:
          begin
             // Assert reset
             load <= 1'b0;
             tv_in_reset_b <= 1'b0;
             if (!ack)
             state <= state+1;
          end
        8'h01:
          state <= state+1;
        8'h02:
          begin
             // Release reset
             tv_in_reset_b <= 1'b1;
             state <= state+1;
                      end
        8'h03:
          begin
             // Send ADV7185 address
             data <= 8'h8A;
             load <= 1'b1;
             if (ack)
             state <= state+1;
          end
        8'h04:
          begin
             // Send subaddress of first register
             data <= 8'h00;
             if (ack)
             state <= state+1;
          end
```

```
8'h05:
  begin
     // Write to register 0
     data <= `ADV7185_REGISTER_0 | {5'h00, {3{source}}};
     if (ack)
     state <= state+1;
  end
8'h06:
  begin
     // Write to register 1
     data <= `ADV7185_REGISTER_1;
     if (ack)
     state <= state+1;
  end
8'h07:
  begin
     // Write to register 2
     data <= `ADV7185_REGISTER_2;
     if (ack)
     state <= state+1;
  end
8'h08:
  begin
     // Write to register 3
     data <= `ADV7185_REGISTER_3;
     if (ack)
     state <= state+1;
  end
8'h09:
  begin
     // Write to register 4
     data <= `ADV7185_REGISTER_4;
     if (ack)
     state <= state+1;
  end
8'h0A:
  begin
     // Write to register 5
     data <= `ADV7185_REGISTER_5;
     if (ack)
     state <= state+1;
  end
8'h0B:
  begin
     // Write to register 6
     data <= 8'h00; // Reserved register, write all zeros
     if (ack)
     state <= state+1;
  end
8'h0C:
  begin
     // Write to register 7
     data <= `ADV7185_REGISTER_7;
     if (ack)
     state <= state+1;
  end
8'h0D:
```

```
  begin
     // Write to register 8
     data <= `ADV7185_REGISTER_8;
     if (ack)
     state <= state+1;
  end
8'h0E:
  begin
     // Write to register 9
     data <= `ADV7185_REGISTER_9;
     if (ack)
     state <= state+1;
  end
8'h0F: begin
   // Write to register A
   data <= `ADV7185_REGISTER_A;
 if (ack)
   state <= state+1;
end
8'h10:
  begin
     // Write to register B
     data <= `ADV7185_REGISTER_B;
     if (ack)
     state <= state+1;
  end
8'h11:
  begin
     // Write to register C
     data <= `ADV7185_REGISTER_C;
     if (ack)
     state <= state+1;
  end
8'h12:
  begin
     // Write to register D
     data <= `ADV7185_REGISTER_D;
     if (ack)
     state <= state+1;
  end
8'h13:
  begin
     // Write to register E
     data <= `ADV7185_REGISTER_E;
     if (ack)
     state <= state+1;
  end
8'h14:
  begin
     // Write to register F
     data <= `ADV7185_REGISTER_F;
     if (ack)
     state <= state+1;
  end
8'h15:
  begin
     // Wait for I2C transmitter to finish
```

```
      load <= 1'b0;
      if (idle)
      state <= state+1;
   end
8'h16:
   begin
      // Write address
      data <= 8'h8A;
      load <= 1'b1;
      if (ack)
      state <= state+1;
   end
8'h17:
   begin
      data <= 8'h33;
      if (ack)
      state <= state+1;
   end
8'h18:
   begin
      data <= `ADV7185_REGISTER_33;
      if (ack)
      state <= state+1;
   end
8'h19:
   begin
      load <= 1'b0;
      if (idle)
      state <= state+1;
   end

8'h1A: begin
    data <= 8'h8A;
    load <= 1'b1;
    if (ack)
      state <= state+1;
end
8'h1B:
   begin
      data <= 8'h33;
      if (ack)
      state <= state+1;
   end
8'h1C:
   begin
      load <= 1'b0;
      if (idle)
      state <= state+1;
   end
8'h1D:
   begin
      load <= 1'b1;
      data <= 8'h8B;
      if (ack)
      state <= state+1;
   end
8'h1E:
```

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

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//////
// Company:
// Engineer: Rob Crowell
//
// Create Date:    17:04:37 05/15/2007
// Design Name:
// Module Name:    crosshair
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
//////
module crosshair(line_count, pixel_count, com_x, com_y, contains);
      input wire [9:0] line_count;
      input wire [9:0] pixel_count;
      input wire [9:0] com_x;
      input wire [9:0] com_y;
      output reg contains;

      parameter HEIGHT = 10'd25;
      parameter THICKNESS = 10'd23;

      always @ ( * ) begin
            if((line_count > com_y - HEIGHT) && (line_count < com_y -
THICKNESS) && (pixel_count > com_x - HEIGHT) && (pixel_count < com_x +
HEIGHT)) begin
                  //top of the box
                  contains = 1;
            end else if((line_count > com_y + THICKNESS) && (line_count <
com_y + HEIGHT) && (pixel_count > com_x - HEIGHT) && (pixel_count < com_x +
HEIGHT)) begin
                  //bottom of box
                  contains = 1;
            end else if((line_count > com_y - HEIGHT) && (line_count < com_y
+ HEIGHT) && (pixel_count > com_x - HEIGHT) && (pixel_count < com_x -
THICKNESS)) begin
                  //left of box
                  contains = 1;
            end else if((line_count > com_y - HEIGHT) && (line_count < com_y
+ HEIGHT) && (pixel_count > com_x + THICKNESS) && (pixel_count < com_x +
HEIGHT)) begin
                  //right of box
                  contains = 1;
            end else begin
                  contains = 0;
            end
      end
```

```
endmodule
```

```
`timescale 1ns / 1ps
`default_nettype none

//////////////////////////////////////////////////////////////////////////////////
/////
// Company:
// Engineer: Rob Crowell
//
// Create Date:    21:27:10 04/29/2007
// Design Name:
// Module Name:    fifo_controller
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
/////
module fifo_controller(llc, clock, reset, ntsc_data_in, ntsc_type_in,
ntsc_row_in, ntsc_col_in, ntsc_data_out, ntsc_row_out, ntsc_col_out,
ntsc_type_out);
      input wire llc;
      input wire clock;
      input wire reset;
      input wire [9:0] ntsc_data_in;
      input wire [1:0] ntsc_type_in;
      input wire [9:0] ntsc_row_in;
      input wire [9:0] ntsc_col_in;
      output [9:0] ntsc_data_out;
      output [1:0] ntsc_type_out;
      output [9:0] ntsc_row_out;
      output [9:0] ntsc_col_out;

      // An 30x16 FIFO which writes on llc and reads on clock
      // Store {row, col, type, data} into the FIFO
      wire [31:0] fifo_in, fifo_out;
      wire wr_en, rd_en, almost_empty, empty, full;
      assign fifo_in = {ntsc_row_in[9:0], ntsc_col_in[9:0],
ntsc_type_in[1:0], ntsc_data_in[9:0]};
      fifo fifo1(.wr_clk(llc),
                         .wr_en(wr_en),
                         .din(fifo_in),
                         .rd_clk(clock),
                         .rd_en(rd_en),
                         .dout(fifo_out),
                         .full(full),
                         .empty(empty),
                         .almost_empty(almost_empty),
                         .rst(reset));

      parameter NO_DATA = 2'd0;
```

```
	wire [1:0] ntsc_type_out;
	wire [9:0] ntsc_data_out;
	wire [9:0] ntsc_row_out;
	wire [9:0] ntsc_col_out;
	assign ntsc_row_out = fifo_out[31:22];
	assign ntsc_col_out = fifo_out[21:12];
	assign ntsc_type_out = fifo_out[11:10];
	assign ntsc_data_out = fifo_out[9:0];

	// Control signals for the fifo
	//assign wr_en = (ntsc_type_in != NO_DATA);
	assign wr_en = 1'b1;
	assign rd_en = ~empty;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
/////
// Company:
// Engineer: Lyric Doshi
//
// Create Date:    19:42:16 03/12/2007
// Design Name:
// Module Name:    generator
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/*

      This module generates control signals for the VGA. line_count and
pixel_count identify the current pixel being displayed. hsyncInv, vsyncInv,
and blankInv are the inverted sync and blanking signals. the syncs need to be
delayed twice before being sent to the vga unit.
*/
//////////////////////////////////////////////////////////////////////////////////
/////
module generator(reset, clock, hsyncInv, vsyncInv, pixel_count, line_count,
blankInv);
    input reset;
    input clock;
    output reg hsyncInv;
    output reg vsyncInv;
    output reg [9:0] pixel_count;
    output reg [9:0] line_count;
    output reg blankInv;

      parameter COUNT_MAX_HORIZ = 799;
      parameter COUNT_MAX_VERT = 524;

      parameter HSYNC_LOWER_BOUND = 656;
      parameter HSYNC_UPPER_BOUND = 751;

      parameter VSYNC_LOWER_BOUND = 491;
      parameter VSYNC_UPPER_BOUND = 492;

      parameter HORIZ_BLANK = 640;
      parameter VERT_BLANK = 480;

      /*
            testing:
            - testing was carried out assuming a 14 x 10 VGA Frame instead of
800 x 525
            - the display region is 8 x 6 instead of 640 x 480
            - the hsyncInv is low at 10 - 12 [from a range of 0 to 13]
```

```
            - the vsyncInv is low at 7 - 8 [from a range of 0 - 9]
            - blankInv is low from 8 to 13 horiz, 6 to 9 vert.
            - these all mirror the actual parameters in that the bounds will
be set in the same way as for the real
              dimensions to make sure all the >= signs and param values and
everything else is all set correctly
              based on the spec.

      parameter COUNT_MAX_HORIZ = 13;
      parameter COUNT_MAX_VERT = 9;

      parameter HSYNC_LOWER_BOUND = 10;
      parameter HSYNC_UPPER_BOUND = 12;

      parameter VSYNC_LOWER_BOUND = 7;
      parameter VSYNC_UPPER_BOUND = 8;

      parameter HORIZ_BLANK = 8;
      parameter VERT_BLANK = 6;
      */

      always @ (posedge clock) begin
            if(reset) begin
                  //restore initial values on reset
                  pixel_count <= 0;
                  line_count <= 0;
            end
            else begin
                  //increment pixel counts
                  if(pixel_count == COUNT_MAX_HORIZ) begin
                        pixel_count <= 0;

                        //increment line count because we just finished a
line!
                        if(line_count == COUNT_MAX_VERT) begin
                              line_count <= 0;
                        end else begin
                              line_count <= line_count + 1;
                        end

                  end else begin
                        pixel_count <= pixel_count + 1;
                  end

            end
      end

      //adjust hsync
      always @ (pixel_count) begin
            if(pixel_count >= HSYNC_LOWER_BOUND && pixel_count <=
HSYNC_UPPER_BOUND) begin
                  hsyncInv = 0;
            end
            else begin
                  hsyncInv = 1;
            end
      end
```

```
	//adjust the vsync
	always @ (line_count) begin
		if(line_count >= VSYNC_LOWER_BOUND && line_count <=
VSYNC_UPPER_BOUND) begin
			vsyncInv = 0;
		end
		else begin
			vsyncInv = 1;
		end
	end

	//calculate blank
	always @ (line_count or pixel_count) begin
		if(line_count >= VERT_BLANK || pixel_count >= HORIZ_BLANK) begin
			blankInv = 0;
		end
		else begin
			blankInv = 1;
		end
	end

endmodule
```

```
`default_nettype none

//from Nathan Ickes

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
```

```
`default_nettype none

//////////////////////////////////////////////////////////////////////////////
//
//
// 6.111 FPGA Labkit -- Lab 2 Template
//
// Created: Feb 15, 2007
// Author: Nathan Ickes (orginal labkit template)
//
// This is a template for implementing the traffic light control system for
Lab 2.
// This file only includes the top-level labkit module. Users should add
functionality
// according to the specifications outlined in the lab manual.
//
//////////////////////////////////////////////////////////////////////////////
//




//////////////////////////////////////////////////////////////////////////////
//
//
// 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)
//
//
// 2006-Mar-08: Corrected default assignments to "vga_out_red",
"vga_out_green"
//              and "vga_out_blue". (Was 10'h0, now 8'h0.)
//
// 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 lab2 (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;

   inout mouse_clock, mouse_data;
      input keyboard_clock, keyboard_data;

   input  clock_27mhz, clock1, clock2;
      // synthesis attribute period of clock_27mhz is 37ns

   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

   // VGA Output
//   assign vga_out_red = 10'h0;
//   assign vga_out_green = 10'h0;
//   assign vga_out_blue = 10'h0;
//   assign vga_out_sync_b = 1'b1;
//   assign vga_out_blank_b = 1'b1;
//   assign vga_out_pixel_clock = 1'b0;
//   assign vga_out_hsync = 1'b0;
//   assign vga_out_vsync = 1'b0;

   // 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
   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
    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;


   // 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;


      //lyric's code for servo test!
            // Create an fpga clock
      wire pixel_clock;
      wire pixel_clock_unbuf;
      wire ram_clock;
      wire dcm_reset;
      DCM fpga_clock_dcm(
            .RST(dcm_reset),
            .CLKIN(clock_27mhz),
            .CLKFB(pixel_clock),
            .CLK0(pixel_clock_unbuf),
            .CLK2X(ram_clock)
      );
      // synthesis attribute CLKFX_MULTIPLY of fpga_clock_dcm is 7
      // synthesis attribute CLKFX_DIVIDE of fpga_clock_dcm is 6
      // synthesis attribute CLKIN_PERIOD of fpga_clock_dcm is 37
      // synthesis attribute DLL_FREQUENCY_MODE of int_dcm is "LOW"
   // synthesis attribute DUTY_CYCLE_CORRECTION of int_dcm is "TRUE"
   // synthesis attribute STARTUP_WAIT of int_dcm is "FALSE"
   // synthesis attribute DFS_FREQUENCY_MODE of int_dcm is "LOW"
   // synthesis attribute CLK_FEEDBACK of int_dcm  is "1X"
   // synthesis attribute CLKOUT_PHASE_SHIFT of int_dcm is "NONE"
   // synthesis attribute PHASE_SHIFT of int_dcm is 0

      // Buffer the fpga clock
      BUFG bufg1(
            .I(pixel_clock_unbuf),
            .O(pixel_clock)
      );

      // Create a DCM reset signal
   SRL16 dcm_rst_sr(
            .D(1'b0),
            .CLK(clock_27mhz),
            .Q(dcm_reset),
            .A0(1'b1),
            .A1(1'b1),
            .A2(1'b1),
            .A3(1'b1)
      );
   // synthesis attribute init of dcm_rst_sr is "000F";

//    wire pclk, pixel_clock;
//   DCM pixel_clock_dcm (.CLKIN(clock_27mhz), .CLKFX(pclk));
//    // synthesis attribute CLKFX_MULTIPLY of pixel_clock_dcm is 7
//    // synthesis attribute CLKFX_DIVIDE of pixel_clock_dcm is 6
//    //    synthesis attribute CLKIN_PERIOD of pixel_clock_dcm is 37
//    // synthesis attribute CLK_FEEDBACK of pixel_clock_dcm is NONE
//   BUFG pixel_clock_buf (.I(pclk), .O(pixel_clock));

//    wire rclk, ram_clock;
//   DCM ram_clock_dcm (.CLKIN(clock_27mhz), .CLKFX(rclk));
//    // synthesis attribute CLKFX_MULTIPLY of ram_clock_dcm is 21
//    // synthesis attribute CLKFX_DIVIDE of ram_clock_dcm is 6
```

```
//    //    synthesis attribute CLKIN_PERIOD of ram_clock_dcm is 37
//    // synthesis attribute CLK_FEEDBACK of ram_clock_dcm is NONE
//   BUFG ram_clock_buf (.I(rclk), .O(ram_clock));

      //used to reset after debouncing button input
      wire reset_sync;
      debounce debounce_reset(.clock(pixel_clock), .reset(1'd0),
.noisy(!button_enter), .clean(reset_sync));


      // SERVO RELATED **********************************
   assign user4[31:1] = 31'hZ;


      //used to send signals from feedback module to servo_pwm module
      wire clockwise, counter_clockwise;

      wire [9:0] com_x;
      wire [9:0] com_y;
      wire hasValidTarget;

      servo_feedback servo_feedback1(.clock(pixel_clock), .x(com_x),
.clockwise(clockwise), .counter_clockwise(counter_clockwise));
      servo_pwm servo_pwm1(.clock(pixel_clock),.reset(reset_sync),
.clockwise(clockwise),
.counter_clockwise(counter_clockwise),.pulse(user4[0]),.hasValidTarget(hasVal
idTarget));
      // ********************************** END SERVO RELATED

   //
   // VGA output signals
   //

   // Inverting the clock to the DAC provides half a clock period for signals
   // to propagate from the FPGA to the DAC.
   assign vga_out_pixel_clock = ~pixel_clock;

   // The composite sync signal is used to encode sync data in the green
   // channel analog voltage for older monitors.  It does not need to be
   // implemented for the monitors in the 6.111 lab, and can be left at 1'b1.
   assign vga_out_sync_b = 1'b1;

   // The following assignments should be deleted and replaced with your own
   // code to implement the Pong game.

      wire [9:0] pixel_count;
      wire [9:0] line_count;
      wire hsyncInv;
      wire vsyncInv;
      wire blankInv;
      wire [23:0] rgb_out;
      reg [23:0] rgb_out_delayed;

      generator
generator1(.reset(reset_sync),.clock(pixel_clock),.hsyncInv(hsyncInv),.vsyncI
nv(vsyncInv),.pixel_count(pixel_count),.line_count(line_count),.blankInv(blan
kInv));
```

```
      //MEMORY AND PARTICLE FSM ***********************************

//    wire start_sync_before;
      wire start_sync;
//    debounce debounce_44(.clock(pixel_clock), .reset(1), .noisy(!button0),
.clean(start_sync_before));
      wire [1:0] pulse_state;
      pulser pulser1(.clock(pixel_clock), .reset(reset_sync),
.signal(switch[7]), .pulse(start_sync), .state(pulse_state));
      wire initialize_sync;
      debounce debounce_14(.clock(pixel_clock), .reset(1'd0),
.noisy(!button1), .clean(initialize_sync));


      wire [1:0] pulse_state_2;
      wire draw_square;
      pulser pulser2(.clock(pixel_clock), .reset(reset_sync),
.signal(switch[6]), .pulse(draw_square), .state(pulse_state_2));

      //**** mouse codes
      wire [10:0] mouse_x, mouse_y;
      wire [2:0] mouse_btn_click;
      ps2_mouse_xy ps2_mouse_xy1(.clk(pixel_clock), .reset(reset_sync),
.ps2_clk(mouse_clock),.ps2_data(mouse_data),.mx(mouse_x), .my(mouse_y),
.btn_click(mouse_btn_click));

      wire [7:0] h, s, v;

      wire [23:0] pixel_color;
      // Outputs

      wire [9:0] particle_row;
      wire [9:0] particle_col;
      wire particle_fsm_done;
      wire particle_fsm_all_dead;

      wire [4:0] state;
      wire [4:0] next;
      wire [7:0] dead_pointer, live_pointer, particle_check_counter;
      //TEMP
      wire particle_detector_want_address;
      wire [8:0] particle_address;
      wire [35:0] particle_dout;
   wire particle_busy;
      wire particle_is_alive_result;
      wire contains, pixel_matches;
      wire [10:0] row_delay_3, col_delay_3;

      wire [7:0] quotient, divisor;
      wire[17:0] dividend;
      wire [10:0] pixel_x,pixel_y;

      wire particle_fsm_start;
      wire [9:0] particle_map_row;
      wire [9:0] particle_map_col;
```

```
      wire particle_present;   //the value returned by the RAM; 1=particle
present, 0=particle absent
      particle_fsm particle_fsm1 (
            .clock(pixel_clock),
            .reset(reset_sync),
            .start(particle_fsm_start),
            .initialize(mouse_btn_click[2]),
            .ext_target_x(mouse_x[10:0]),
            .ext_target_y(mouse_y[10:0]),
            .random_number_x(11'd0),
            .random_number_y(11'd0),
            .com_x(com_x),
            .com_y(com_y),
            .particle_row(particle_row),
            .particle_col(particle_col),
            .pixel_color({h,s,v}),
            .done(particle_fsm_done),
            .ext_addr(particle_address),
            .dout(particle_dout),
            .want_address(particle_detector_want_address),
            .state(state),
            .next(next),
            .busy(particle_busy),
            .dead_pointer(dead_pointer),
            .live_pointer(live_pointer),
            .particle_check_counter(particle_check_counter),
            .particle_is_alive_result(particle_is_alive_result),
            .contains(contains),
            .pixel_matches(pixel_matches),
            .row_delay_3(row_delay_3),
            .col_delay_3(col_delay_3),
            .all_dead(particle_fsm_all_dead),
            .divisor(divisor),
            .dividend(dividend),
            .quotient(quotient),
            .pixel_x(pixel_x),
            .pixel_y(pixel_y),
            .addr_vga({particle_map_col,particle_map_row}),
            .dout_particle_status(particle_present),
            .SAT_INPUT(switch[6:4]),
            .VAL_INPUT(switch[3:0]),
            .hasValidTarget(hasValidTarget)
      );


//    // Inputs
//    reg [23:0] ntsc_hsv;
//    reg ntsc_data_ready;
//    reg [9:0] ntsc_row;
//    reg [9:0] ntsc_col;
//
//
//    // Outputs
//    wire [23:0] vga_hsv;
//    assign rgb_out = vga_hsv;
//
//    bram_controller bram_controller1 (
```

```
//          .clock_3x(ram_clock),
//          .reset(reset_sync),
//          .vga_row(line_count),
//          .vga_col(pixel_count),
//          .particle_row(particle_row),
//          .particle_col(particle_col),
//          .ntsc_row(ntsc_row),
//          .ntsc_col(ntsc_col),
//          .ntsc_hsv(ntsc_hsv),
//          .ntsc_data_ready(ntsc_data_ready),
//          .vga_hsv(vga_hsv),
//          .particle_hsv(pixel_color)
//    );

//    parameter MEM_BITS = 6;
//
//    // Create a function for computing an address from row+col
//    parameter ACTIVE_WIDTH = 12'd64;    //1024 x 24 RAM
//    parameter ACTIVE_HEIGHT = 12'd64;
//    function [11:0] ADDRESS;
//          input [9:0] row;
//          input [9:0] col;
//          begin
//                ADDRESS = {row[5:0],col[5:0]};
//          end
//    endfunction
//
//



//    reg [23:0] ram_input;
//    wire [23:0] ram_output;
//    wire [9:0] ram_row;
//    wire [9:0] ram_col;
//    reg ram_we;
//    sing_port_larger_test_mem mem(.addr(ADDRESS(ram_row, ram_col) ),
.clk(pixel_clock), .din(ram_input), .dout(ram_output), .we(ram_we));
//
//    assign ram_row = (particle_busy) ?  particle_row : line_count;
//    assign ram_col = (particle_busy) ? particle_col : pixel_count;
//
//    //introduce a cycle delay to forge zbt behavior
//    always @ (posedge pixel_clock) begin
//          pixel_color <= ram_output;
//          rgb_out <= ram_output;
//    end



      //INITIALIZING MEMORY
      wire store_black;
      debounce debounce_1(.clock(pixel_clock), .reset(1), .noisy(!button2),
.clean(store_black));
      wire store_red;
      debounce debounce_2(.clock(pixel_clock), .reset(1), .noisy(!button3),
.clean(store_red));
```

```
	reg [5:0] black_count_x;
	reg [5:0] black_count_y;
	parameter BLACK_COUNT_MAX_X = 6'd32;
	parameter BLACK_COUNT_MAX_Y = 6'd32;
	reg [4:0] red_count_x;
	reg [4:0] red_count_y;
	parameter RED_COUNT_MAX_X = 5'd15;
	parameter RED_COUNT_MAX_Y = 5'd9;
	parameter RED_SHIFT_X = 5'd8;
	parameter RED_SHIFT_Y = 5'd10;
	parameter RED_LEFT = 6'd17;
	parameter RED_RIGHT = 6'd50;
	parameter RED_TOP = 6'd10;
	parameter RED_BOTTOM =6'd50;

	reg [22:0] green_count;

	parameter [22:0] GREEN_MAX = 22'd3150000;
	//alternator
	reg alternator;
	reg alternator2;
	reg alternator3;
	reg start_particles;
	reg pulsed_it;
	reg going_right;
	reg going_down;
	reg write_green;
	reg [9:0] green_left, green_right, green_top, green_bottom;

	//////////////////////////////////////////////////////////////////////
/////////////
	// DISPLAY THE VIDEO, PARTICLES, MOUSE, AND CENTER OF MASS ON THE
SCREEN
	parameter PARTICLE_COLOR = 24'h00FF00;
	parameter COM_COLOR = 24'h0000FF;
	parameter COM_DEAD_COLOR = 24'hFF0000;
	parameter MOUSE_COLOR = 24'h00FFFF;

	// Read out of the BRAM to see if a particle should be drawn here
	assign particle_map_row = line_count;
	assign particle_map_col = pixel_count;

	// The delay between setting line_count and getting a new value from
the BRAM is 1 cycle
	// The RGB delay is 3 cycles (2 ZBT, 1 register)
	// Thus, we delay the bram output by 2 cycles
	// Also delay the VGA row and col by 2 cycles to use in computing COM
	reg particle_present_delay1, particle_present_delay2;
	reg [9:0] col_delay1, col_delay2;
	reg [9:0] row_delay1, row_delay2;
	always @ (posedge pixel_clock) begin
		particle_present_delay1 <= particle_present;
		particle_present_delay2 <= particle_present_delay1;

		col_delay1 <= pixel_count;
		col_delay2 <= col_delay1;
```

```
            row_delay1 <= line_count;
            row_delay2 <= row_delay1;
      end

      // Create a mouse_pointer module to draw the pointer on the screen
      wire mouse_contains;
      mouse_pointer mouse_pointer1(
            .mouse_x(mouse_x),
            .mouse_y(mouse_y),
            .line_count(row_delay2),
            .pixel_count(col_delay2),
            .contains(mouse_contains)
      );

      // Create a crosshair to hilight the target
      wire crosshair_contains;
      crosshair crosshair1(
            .line_count(row_delay2),
            .pixel_count(col_delay2),
            .com_x(com_x),
            .com_y(com_y),
            .contains(crosshair_contains)
      );

      // Drive the VGA with either the video, the particles, or the COM
      reg [7:0] vga_out_red;
      reg [7:0] vga_out_green;
      reg [7:0] vga_out_blue;
      always @ ( * ) begin
            if(mouse_contains) begin
                  vga_out_red = MOUSE_COLOR[23:16];
                  vga_out_green = MOUSE_COLOR[15:8];
                  vga_out_blue = MOUSE_COLOR[7:0];
            end else if(switch[7] && particle_present_delay2) begin   //show
every particle
                  vga_out_red = PARTICLE_COLOR[23:16];
                  vga_out_green = PARTICLE_COLOR[15:8];
                  vga_out_blue = PARTICLE_COLOR[7:0];
            end else if(~switch[7] && crosshair_contains && hasValidTarget)
begin
                  vga_out_red = COM_COLOR[23:16];
                  vga_out_green = COM_COLOR[15:8];
                  vga_out_blue = COM_COLOR[7:0];
            end else if(~switch[7] && crosshair_contains) begin
                  vga_out_red = COM_DEAD_COLOR[23:16];
                  vga_out_green = COM_DEAD_COLOR[15:8];
                  vga_out_blue = COM_DEAD_COLOR[7:0];
            end else begin
                  vga_out_red = rgb_out_delayed[23:16];
                  vga_out_green = rgb_out_delayed[15:8];
                  vga_out_blue = rgb_out_delayed[7:0];
            end
      end

      // Introduce two cycle delay on hsyncInv and vsyncInv
      // This is needed because the VGA output is pipelined in the labkit
```

```
	reg hsync_delay1, hsync_delay2;
	reg vsync_delay1, vsync_delay2;
	always @ (posedge pixel_clock) begin
		hsync_delay1 <= hsyncInv;
		hsync_delay2 <= hsync_delay1;

		vsync_delay1 <= vsyncInv;
		vsync_delay2 <= vsync_delay1;
	end

	// Add three delays in the VGA signals, two for the ZBT and one for
registering the ZBT's output
	reg hsync_delay3, hsync_delay4, vga_out_hsync;
	reg vsync_delay3, vsync_delay4, vga_out_vsync;
	reg blank_delay3, blank_delay4, vga_out_blank_b;
	always @ (posedge pixel_clock) begin
		hsync_delay3 <= hsync_delay2;
		hsync_delay4 <= hsync_delay3;
		vga_out_hsync <= hsync_delay4;

		vsync_delay3 <= vsync_delay2;
		vsync_delay4 <= vsync_delay3;
		vga_out_vsync <= vsync_delay4;

		blank_delay3 <= blankInv;
		blank_delay4 <= blank_delay3;
		vga_out_blank_b <= blank_delay4;

		// The ZBT output is registered to prevent data corruption
		rgb_out_delayed <= rgb_out;
	end
	////////////////////////////////////////////////////////////////////
////////////

	// Initialize the ADV7185 video decoder
	wire tv_in_reset_b;
	wire tv_in_i2c_clock;
	wire tv_in_i2c_data;
	adv7185init adv7185init1(
		.reset(reset_sync),
		.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)
	);
	PULLUP pu_out(.O(tv_in_i2c_data));
	assign tv_in_clock = clock_27mhz;
	assign tv_in_fifo_read = 1'b1;
   assign tv_in_fifo_clock = clock_27mhz;
	assign tv_in_iso = 1'b0;


	// The NTSC decoder
	wire [9:0] decoder_ycrcb;
	wire [1:0] decoder_type;
	wire [9:0] decoder_row;
```

```
wire [9:0] decoder_col;
wire [6:0] decoder_state;
ntsc_decoder ntsc_decoder1(
      .llc(~tv_in_line_clock1),
      .reset(reset_sync),
      .tv_in_ycrcb(tv_in_ycrcb),
      .ycrcb_out(decoder_ycrcb),
      .ntsc_type(decoder_type),
      .row(decoder_row),
      .col(decoder_col),
      .state(decoder_state)
);

// Put the data from the NTSC decoder into an async FIFO
wire [9:0] fifo_ycrcb;
wire [9:0] fifo_row;
wire [9:0] fifo_col;
wire [1:0] fifo_type;
fifo_controller fifo_controller1(
      .llc(~tv_in_line_clock1),
      .clock(pixel_clock),
      .reset(reset_sync),
      .ntsc_data_in(decoder_ycrcb),
      .ntsc_type_in(decoder_type),
      .ntsc_row_in(decoder_row),
      .ntsc_col_in(decoder_col),
      .ntsc_data_out(fifo_ycrcb),
      .ntsc_row_out(fifo_row),
      .ntsc_col_out(fifo_col),
      .ntsc_type_out(fifo_type)
);


// Create 4:4:4 ycrcb video from the 4:2:2 YCrCb signal
wire [9:0] interpolator_y;
wire [9:0] interpolator_cr;
wire [9:0] interpolator_cb;
wire [9:0] interpolator_row, interpolator_col;
wire interpolator_done;
ntsc_interpolator ntsc_interpolator1(
      .clock(pixel_clock),
      .reset(reset_sync),
      .ntsc_data(fifo_ycrcb),
      .ntsc_row(fifo_row),
      .ntsc_col(fifo_col),
      .ntsc_type(fifo_type),
      .y(interpolator_y),
      .cr(interpolator_cr),
      .cb(interpolator_cb),
      .row(interpolator_row),
      .col(interpolator_col),
      .data_ready(interpolator_done)
);

// Convert the 4:4:4 YCrCb video into RGB pixels
wire [23:0] interpolator_rgb;
ycrcb2rgb ycrcb2rgb1(
```

```
            .clk(pixel_clock),
            .rst(reset_sync),
            .R(interpolator_rgb[23:16]),
            .G(interpolator_rgb[15:8]),
            .B(interpolator_rgb[7:0]),
            .Y(interpolator_y),
            .Cr(interpolator_cr),
            .Cb(interpolator_cb)
      );
      reg [9:0] int_row_delay1, int_row_delay2, int_row_delay3;
      reg [9:0] int_col_delay1, int_col_delay2, int_col_delay3;
      reg int_done_delay1, int_done_delay2, int_done_delay3;

      //apply median filter

      wire [23:0] filtered_rgb;
      wire [9:0] filtered_col;
      wire [9:0] filtered_row;
      wire filtered_data_ready;

      wire [1:0] median_state;

       median_filter median_filter1(
            .clock(pixel_clock),
            .reset(reset_sync),
            .rgb(interpolator_rgb),
            .data_ready(int_done_delay3),
            .col_count(int_col_delay3),
            .row_count(int_row_delay3),
            .filtered_rgb(filtered_rgb),
            .filtered_col(filtered_col),
            .filtered_row(filtered_row),
            .filtered_data_ready(filtered_data_ready),
            .state(median_state)
      );

      //center image in VGA display to eliminate dead space
      parameter DEAD_X = 10'b11_1111_1111;
      parameter DEAD_Y = 10'b11_1111_1111;
      parameter SHIFT_ROW = 10'd22;
      parameter SHIFT_COL = 10'd9;

      always @ (posedge pixel_clock) begin
            int_row_delay1 <= interpolator_row;
            int_row_delay2 <= int_row_delay1;

            int_col_delay1 <= interpolator_col;
            int_col_delay2 <= int_col_delay1;

            if(int_row_delay2 >= SHIFT_ROW && int_col_delay2 >= SHIFT_COL)
begin
                  int_row_delay3 <= int_row_delay2 - SHIFT_ROW;
                  int_col_delay3 <= int_col_delay2 - SHIFT_COL;
            end else begin
                  int_row_delay3 <= DEAD_Y;
                  int_col_delay3 <= DEAD_X;
            end
```

```
            int_done_delay1 <= interpolator_done;
            int_done_delay2 <= int_done_delay1;
            int_done_delay3 <= int_done_delay2;
      end

      // Test the rgb2hsv converter

      rgb2hsv rgb2hsv1(
            .clock(pixel_clock),
            .reset(reset_sync),
            .r(pixel_color[23:16]),
            .g(pixel_color[15:8]),
            .b(pixel_color[7:0]),
            .h(h),
            .s(s),
            .v(v)
      );

      // Create a delayed 'frame_done' signal to start the particle filter
iteration (particle_fsm_start)
      // Reset the counter every time int_done_delay3 goes high (writes the
ram)
      // When it hasn't gone high in a long time, we're vertical blanking
      // This is slightly random since the FIFO doesn't guarantee timing so
we're not quite sure when data writing is done
      reg [8:0] frame_done_cnt;
      always @ (posedge pixel_clock) begin
            if(reset_sync || initialize_sync || int_done_delay3) begin
                  frame_done_cnt <= 0;
            end else begin
                  if(frame_done_cnt < 9'b1_1111_1111) begin
                        frame_done_cnt <= frame_done_cnt + 1;   //gets stuck
at 31 until reset or frame_done (set by decoder)
                  end
            end
      end
      assign particle_fsm_start = (frame_done_cnt == 9'b1_1111_1110);


      //IMPLEMENT ZOOM
      wire zoom_on;
      assign zoom_on = switch[0];
      //zoom x defines how many times zoom by defining how many bits to shift
to divide
      parameter ZOOM_X_SHIFT = 2'd3;
      //these describe which part of the image should be displayed zoomed in
on
      parameter ZOOM_HORIZONTAL_SHIFT = 10'd280;
      parameter ZOOM_VERTICAL_SHIFT = 10'd210;

      wire [9:0] particle_row_to_memory;
      wire [9:0] particle_col_to_memory;
      wire [9:0] line_count_to_memory;
      wire [9:0] pixel_count_to_memory;

      wire [9:0] zoomed_particle_row;
```

```
      wire [9:0] zoomed_particle_col;
      wire [9:0] zoomed_line_count;
      wire [9:0] zoomed_pixel_count;

      assign zoomed_particle_row = (particle_row >> ZOOM_X_SHIFT) +
ZOOM_HORIZONTAL_SHIFT;
      assign zoomed_particle_col = (particle_col >> ZOOM_X_SHIFT) +
ZOOM_VERTICAL_SHIFT;
      assign zoomed_pixel_count = (pixel_count >> ZOOM_X_SHIFT) +
ZOOM_HORIZONTAL_SHIFT;
      assign zoomed_line_count = (line_count >> ZOOM_X_SHIFT) +
ZOOM_VERTICAL_SHIFT;


      assign particle_row_to_memory = zoom_on ? zoomed_particle_row :
particle_row;
      assign particle_col_to_memory = zoom_on ? zoomed_particle_col :
particle_col;
      assign line_count_to_memory = zoom_on ? zoomed_line_count : line_count;
      assign pixel_count_to_memory = zoom_on ? zoomed_pixel_count :
pixel_count;

      //THE MEMORY IS CONTROLLED HERE
      memory_controller memory_controller1(.clock_2x(ram_clock),
.reset(reset_sync), .vga_row(line_count_to_memory),
.vga_col(pixel_count_to_memory),
            .vga_rgb(rgb_out), .ntsc_row(filtered_row),
.ntsc_col(filtered_col), .ntsc_rgb(filtered_rgb),
.ntsc_data_ready(filtered_data_ready),
            .particle_row(particle_row_to_memory),
.particle_col(particle_col_to_memory), .particle_rgb(pixel_color),
            .ram_address(ram0_address), .ram_data(ram0_data),
.ram_ce_b(ram0_ce_b), .ram_cen_b(ram0_cen_b),
            .ram_oe_b(ram0_oe_b), .ram_we_b(ram0_we_b),
.ram_bwe_b(ram0_bwe_b),.ram_adv_ld(ram0_adv_ld));
      assign ram0_clk = ~ram_clock;

//   assign analyzer1_data = {h, s};
//    assign analyzer2_data = {particle_fsm_start, particle_fsm_all_dead,
pixel_matches,state,v};
   assign analyzer1_clock = pixel_clock;
////  assign analyzer3_data = ram0_address[15:0];
//    assign analyzer4_data = {int_done_delay3,ram0_we_b,ram_clock,
ram0_address[18:16], ram0_data[9:0]};
//
//    assign analyzer3_data = {com_x, clockwise, counter_clockwise,
user4[0],3'd0};




            assign analyzer1_data = filtered_rgb[15:0];
            assign analyzer2_data =
{filtered_rgb[23:16],int_col_delay3[7:0]};
            assign analyzer3_data =
{filtered_col,filtered_data_ready,int_done_delay3,int_col_delay3[9:8],median_
state};
```

```
            assign analyzer4_data = {filtered_row,6'd0};
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//////
// Company:
// Engineer: Lyric Doshi
//
// Create Date:    00:42:14 05/09/2007
// Design Name:
// Module Name:    lfsr_23
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/*
      Linear Feedback Shift Register: outputs next_bit every clock cycle,
shifts its bit down one, and adds a new bit to the left using xor on the
existing bits. generates  2^23 - 1 different conigurations of bits on the
shift register. used for pseudo-random number generation.
*/
//
//////////////////////////////////////////////////////////////////////////////////
//////
module lfsr_23(clock, reset, next_bit);
    input clock;
    input reset;
    output next_bit;

      parameter [22:0] INIT = 23'h7FFFFF;

      reg [22:0] shifter;
      wire result;

      always @ (posedge clock) begin
            if(reset) begin
                  shifter <= INIT;
            end else begin
                  //shift bits to the right, add new bit on the left
                  shifter <= {result,shifter[22:1]};
            end
      end

      assign next_bit = shifter[0];
      assign result = ( shifter[4] ^ shifter[22] ) ;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//////
// Company:
// Engineer: Lyric Doshi
//
// Create Date:    23:22:07 05/08/2007
// Design Name:
// Module Name:    lfsr
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//    Linear Feedback Shift Register: outputs next_bit every clock cycle,
shifts its bit down one, and adds a new bit to the left using xor on the
existing bits. generates  2^25 - 1 different conigurations of bits on the
shift register. used for pseudo-random number generation.
//////////////////////////////////////////////////////////////////////////////////
//////
module lfsr_25(clock, reset, next_bit);
    input clock;
    input reset;
    output next_bit;

      parameter [24:0] INIT = 25'h1FFFFFF;

      reg [24:0] shifter;
      wire result;

      always @ (posedge clock) begin
            if(reset) begin
                  shifter <= INIT;
            end else begin
                  //shift bits to the right, add new bit on the left
                  shifter <= {result,shifter[24:1]};
            end
      end

      assign next_bit = shifter[0];
      assign result = ( shifter[2] ^ shifter[24] ) ;

endmodule
```

```
`timescale 1ns / 1ps
`default_nettype none

//////////////////////////////////////////////////////////////////////////////////
//////
// Company:
// Engineer: Lyric Doshi
//
// Create Date:    22:51:19 05/14/2007
// Design Name:
// Module Name:    median_filter
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/*
      This module applies a median filter to the image data. it takes five
samples at a time and chooses the median. rgb, data_ready, col_count,
row_count come in, and filtered_rgb, filtered_col, filtered_row,
filtered_data_ready go out. the row and ready are just piped out, the
filtered_rgb is the output after the median filter calculations. the module
introduces a four cycle delay on the data.

      algorithm to find median quickly in parallel: find which element of the
5 elt buffer is greater than or equal and less than or equal to at least 2
other elements in the buffer.
*/
//////////////////////////////////////////////////////////////////////////////////
//////
module median_filter(clock, reset, rgb, data_ready, col_count, row_count,
filtered_rgb, filtered_col, filtered_row, filtered_data_ready, state);
    input clock;
    input reset;
       input data_ready;
       output [1:0] state;
    input [23:0] rgb;
    input [9:0] col_count;
       input [9:0] row_count;
    output [23:0] filtered_rgb;
       output [9:0] filtered_col;
       output [9:0] filtered_row;
       output filtered_data_ready;


      wire [9:0] filtered_col;
      assign filtered_col = col_count - 10'd3;

      reg [9:0] row_count_delayed;
      reg data_ready_delayed;
```

```
assign filtered_data_ready = data_ready_delayed;
assign filtered_row = row_count_delayed;


reg [23:0] filtered_rgb, next_filtered_rgb;

reg[1:0] state,next;

//for readability
wire [7:0] red, green, blue;
assign red = rgb [23:16];
assign green = rgb [15:8];
assign blue = rgb [7:0];

reg [39:0] next_buffer_red, buffer_red;
reg [39:0] next_buffer_green, buffer_green;
reg [39:0] next_buffer_blue, buffer_blue;

reg [7:0] next_filtered_red, next_filtered_green, next_filtered_blue;


//used for computing the median
wire[2:0] red_0_greater_than_score;
wire[2:0] red_0_less_than_score;
wire[2:0] red_1_greater_than_score;
wire[2:0] red_1_less_than_score;
wire[2:0] red_2_greater_than_score;
wire[2:0] red_2_less_than_score;
wire[2:0] red_3_greater_than_score;
wire[2:0] red_3_less_than_score;
wire[2:0] red_4_greater_than_score;
wire[2:0] red_4_less_than_score;
wire[2:0] green_0_greater_than_score;
wire[2:0] green_0_less_than_score;
wire[2:0] green_1_greater_than_score;
wire[2:0] green_1_less_than_score;
wire[2:0] green_2_greater_than_score;
wire[2:0] green_2_less_than_score;
wire[2:0] green_3_greater_than_score;
wire[2:0] green_3_less_than_score;
wire[2:0] green_4_greater_than_score;
wire[2:0] green_4_less_than_score;
wire[2:0] blue_0_greater_than_score;
wire[2:0] blue_0_less_than_score;
wire[2:0] blue_1_greater_than_score;
wire[2:0] blue_1_less_than_score;
wire[2:0] blue_2_greater_than_score;
wire[2:0] blue_2_less_than_score;
wire[2:0] blue_3_greater_than_score;
wire[2:0] blue_3_less_than_score;
wire[2:0] blue_4_greater_than_score;
wire[2:0] blue_4_less_than_score;

parameter IDLE = 2'd0;
parameter SET_UP_1 = 2'd1;
parameter SET_UP_2 = 2'd2;
parameter STEADY_STATE = 2'd3;
```

```
	always @ (posedge clock) begin

		if(reset) begin
			state <= IDLE;
			buffer_red <= 40'd0;
			buffer_green <= 40'd0;
			buffer_blue <= 40'd0;
			filtered_rgb <= 40'd0;
			row_count_delayed <= 24'd0;
			data_ready_delayed <= 4'd0;
		end else begin

			row_count_delayed <= row_count;
			data_ready_delayed <= data_ready;

			state <= next;
			buffer_red <= next_buffer_red;
			buffer_green <= next_buffer_green;
			buffer_blue <= next_buffer_blue;
			filtered_rgb <= next_filtered_rgb;
		end
	end

	always @ * begin
		//hold values by default
		next_buffer_red = buffer_red;
		next_buffer_green = buffer_green;
		next_buffer_blue = buffer_blue;
		next_filtered_rgb = filtered_rgb;

		case(state)

			//wait for the first three data samples to come in
			IDLE: begin
				if(col_count == 10'd0 && data_ready) begin
					//we're at the first in the row
					//initialize the buffer!
					next_buffer_red = {red,red,red,red,red};
					next_buffer_green =
{green,green,green,green,green};
					next_buffer_blue = {blue,blue,blue,blue,blue};

					next = SET_UP_1;
				end else begin
					next = IDLE;
				end
			end

			//wait for the first three data samples to come in. update
buffers with new data
			SET_UP_1: begin
				if(col_count == 10'd1 && data_ready) begin
					next_buffer_red = {buffer_red[31:0],red};
					next_buffer_green = {buffer_green[31:0],green};
					next_buffer_blue = {buffer_blue[31:0],blue};
```

```
                              next = SET_UP_2;
                        end else begin
                              next = SET_UP_1;
                        end
                  end

                  //wait for the first three data samples to come in. update
buffers with new data
                  SET_UP_2: begin
                        if(col_count == 10'd2 && data_ready) begin
                              next_buffer_red = {buffer_red[31:0],red};
                              next_buffer_green = {buffer_green[31:0],green};
                              next_buffer_blue = {buffer_blue[31:0],blue};
                              next = STEADY_STATE;
                        end else begin
                              next = SET_UP_2;
                        end
                  end

                  //compute median and register data. also update buffers
with new data.
                  STEADY_STATE: begin
                        if(col_count == 10'd719) begin
                              //dont care about last ones because they are
not displayed or used
                              next = IDLE;
                        end else if(data_ready) begin

                              //shift the buffer to lose the oldest and add
the newest red value
                              next_buffer_red = {buffer_red[31:0],red};
                              next_buffer_green = {buffer_green[31:0],green};
                              next_buffer_blue = {buffer_blue[31:0],blue};
                              next_filtered_rgb = {next_filtered_red,
next_filtered_green, next_filtered_blue};
                        end else begin
                              //if data_ready low, col is 0
                              next = STEADY_STATE;
                        end
                  end

            endcase

      end

      always @ * begin
            if(red_0_greater_than_score >= 3'd2 && red_0_less_than_score >=
3'd2) begin
                  next_filtered_red = buffer_red[31:24];
            end else if(red_1_greater_than_score >= 3'd2 &&
red_1_less_than_score >= 3'd2) begin
                  next_filtered_red = buffer_red[23:16];
            end else if(red_2_greater_than_score >= 3'd2 &&
red_2_less_than_score >= 3'd2) begin
                  next_filtered_red = buffer_red[15:8];
            end else if(red_3_greater_than_score >= 3'd2 &&
red_3_less_than_score >= 3'd2) begin
```

```
                  next_filtered_red = buffer_red[7:0];
            end else begin
                  next_filtered_red = red;
            end

            if(green_0_greater_than_score >= 3'd2 && green_0_less_than_score
>= 3'd2) begin
                  next_filtered_green = buffer_green[31:24];
            end else if(green_1_greater_than_score >= 3'd2 &&
green_1_less_than_score >= 3'd2) begin
                  next_filtered_green = buffer_green[23:16];
            end else if(green_2_greater_than_score >= 3'd2 &&
green_2_less_than_score >= 3'd2) begin
                  next_filtered_green = buffer_green[15:8];
            end else if(green_3_greater_than_score >= 3'd2 &&
green_3_less_than_score >= 3'd2) begin
                  next_filtered_green = buffer_green[7:0];
            end else begin
                  next_filtered_green = green;
            end

            if(blue_0_greater_than_score >= 3'd2 && blue_0_less_than_score >=
3'd2) begin
                  next_filtered_blue = buffer_blue[31:24];
            end else if(blue_1_greater_than_score >= 3'd2 &&
blue_1_less_than_score >= 3'd2) begin
                  next_filtered_blue = buffer_blue[23:16];
            end else if(blue_2_greater_than_score >= 3'd2 &&
blue_2_less_than_score >= 3'd2) begin
                  next_filtered_blue = buffer_blue[15:8];
            end else if(blue_3_greater_than_score >= 3'd2 &&
blue_3_less_than_score >= 3'd2) begin
                  next_filtered_blue = buffer_blue[7:0];
            end else begin
                  next_filtered_blue = blue;
            end

      end

      //used for the median calculator to determine which elt is >= and <= to
at least two others
      assign red_0_greater_than_score = (buffer_red[31:24] >=
buffer_red[23:16]) + (buffer_red[31:24] >= buffer_red[15:8]) +
(buffer_red[31:24] >= buffer_red[7:0]) + (buffer_red[31:24] >= red);
      assign red_0_less_than_score = (buffer_red[31:24] <= buffer_red[23:16])
+ (buffer_red[31:24] <= buffer_red[15:8]) + (buffer_red[31:24] <=
buffer_red[7:0]) + (buffer_red[31:24] <= red);

      assign red_1_greater_than_score = (buffer_red[23:16] >=
buffer_red[31:24]) + (buffer_red[23:16] >= buffer_red[15:8]) +
(buffer_red[23:16] >= buffer_red[7:0]) + (buffer_red[23:16] >= red);
      assign red_1_less_than_score = (buffer_red[23:16] <= buffer_red[31:24])
+ (buffer_red[23:16] <= buffer_red[15:8]) + (buffer_red[23:16] <=
buffer_red[7:0]) + (buffer_red[23:16] <= red);
```

```
      assign red_2_greater_than_score = (buffer_red[15:8] >=
buffer_red[23:16]) + (buffer_red[15:8] >= buffer_red[31:24]) +
(buffer_red[15:8] >= buffer_red[7:0]) + (buffer_red[15:8] >= red);
      assign red_2_less_than_score = (buffer_red[15:8] <= buffer_red[23:16])
+ (buffer_red[15:8] <= buffer_red[31:24]) + (buffer_red[15:8] <=
buffer_red[7:0]) + (buffer_red[15:8] <= red);

      assign red_3_greater_than_score = (buffer_red[7:0] >=
buffer_red[23:16]) + (buffer_red[7:0] >= buffer_red[15:8]) + (buffer_red[7:0]
>= buffer_red[31:24]) + (buffer_red[7:0] >= red);
      assign red_3_less_than_score = (buffer_red[7:0] <= buffer_red[23:16]) +
(buffer_red[7:0] <= buffer_red[15:8]) + (buffer_red[7:0] <=
buffer_red[31:24]) + (buffer_red[7:0] <= red);

      assign red_4_greater_than_score = (red >= buffer_red[23:16]) + (red >=
buffer_red[15:8]) + (red >= buffer_red[7:0]) + (red >= buffer_red[31:24] );
      assign red_4_less_than_score = (red <= buffer_red[23:16]) + (red <=
buffer_red[15:8]) + (red <= buffer_red[7:0]) + (red <= buffer_red[31:24] );

      assign green_0_greater_than_score = (buffer_green[31:24] >=
buffer_green[23:16]) + (buffer_green[31:24] >= buffer_green[15:8]) +
(buffer_green[31:24] >= buffer_green[7:0]) + (buffer_green[31:24] >= green);
      assign green_0_less_than_score = (buffer_green[31:24] <=
buffer_green[23:16]) + (buffer_green[31:24] <= buffer_green[15:8]) +
(buffer_green[31:24] <= buffer_green[7:0]) + (buffer_green[31:24] <= green);

      assign green_1_greater_than_score = (buffer_green[23:16] >=
buffer_green[31:24]) + (buffer_green[23:16] >= buffer_green[15:8]) +
(buffer_green[23:16] >= buffer_green[7:0]) + (buffer_green[23:16] >= green);
      assign green_1_less_than_score = (buffer_green[23:16] <=
buffer_green[31:24]) + (buffer_green[23:16] <= buffer_green[15:8]) +
(buffer_green[23:16] <= buffer_green[7:0]) + (buffer_green[23:16] <= green);

      assign green_2_greater_than_score = (buffer_green[15:8] >=
buffer_green[23:16]) + (buffer_green[15:8] >= buffer_green[31:24]) +
(buffer_green[15:8] >= buffer_green[7:0]) + (buffer_green[15:8] >= green);
      assign green_2_less_than_score = (buffer_green[15:8] <=
buffer_green[23:16]) + (buffer_green[15:8] <= buffer_green[31:24]) +
(buffer_green[15:8] <= buffer_green[7:0]) + (buffer_green[15:8] <= green);

      assign green_3_greater_than_score = (buffer_green[7:0] >=
buffer_green[23:16]) + (buffer_green[7:0] >= buffer_green[15:8]) +
(buffer_green[7:0] >= buffer_green[31:24]) + (buffer_green[7:0] >= green);
      assign green_3_less_than_score = (buffer_green[7:0] <=
buffer_green[23:16]) + (buffer_green[7:0] <= buffer_green[15:8]) +
(buffer_green[7:0] <= buffer_green[31:24]) + (buffer_green[7:0] <= green);

      assign green_4_greater_than_score = (green >= buffer_green[23:16]) +
(green >= buffer_green[15:8]) + (green >= buffer_green[7:0]) + (green >=
buffer_green[31:24] );
      assign green_4_less_than_score = (green <= buffer_green[23:16]) +
(green <= buffer_green[15:8]) + (green <= buffer_green[7:0]) + (green <=
buffer_green[31:24] );

      assign blue_0_greater_than_score = (buffer_blue[31:24] >=
buffer_blue[23:16]) + (buffer_blue[31:24] >= buffer_blue[15:8]) +
(buffer_blue[31:24] >= buffer_blue[7:0]) + (buffer_blue[31:24] >= blue);
```

```
      assign blue_0_less_than_score = (buffer_blue[31:24] <=
buffer_blue[23:16]) + (buffer_blue[31:24] <= buffer_blue[15:8]) +
(buffer_blue[31:24] <= buffer_blue[7:0]) + (buffer_blue[31:24] <= blue);

      assign blue_1_greater_than_score = (buffer_blue[23:16] >=
buffer_blue[31:24]) + (buffer_blue[23:16] >= buffer_blue[15:8]) +
(buffer_blue[23:16] >= buffer_blue[7:0]) + (buffer_blue[23:16] >= blue);
      assign blue_1_less_than_score = (buffer_blue[23:16] <=
buffer_blue[31:24]) + (buffer_blue[23:16] <= buffer_blue[15:8]) +
(buffer_blue[23:16] <= buffer_blue[7:0]) + (buffer_blue[23:16] <= blue);

      assign blue_2_greater_than_score = (buffer_blue[15:8] >=
buffer_blue[23:16]) + (buffer_blue[15:8] >= buffer_blue[31:24]) +
(buffer_blue[15:8] >= buffer_blue[7:0]) + (buffer_blue[15:8] >= blue);
      assign blue_2_less_than_score = (buffer_blue[15:8] <=
buffer_blue[23:16]) + (buffer_blue[15:8] <= buffer_blue[31:24]) +
(buffer_blue[15:8] <= buffer_blue[7:0]) + (buffer_blue[15:8] <= blue);

      assign blue_3_greater_than_score = (buffer_blue[7:0] >=
buffer_blue[23:16]) + (buffer_blue[7:0] >= buffer_blue[15:8]) +
(buffer_blue[7:0] >= buffer_blue[31:24]) + (buffer_blue[7:0] >= blue);
      assign blue_3_less_than_score = (buffer_blue[7:0] <=
buffer_blue[23:16]) + (buffer_blue[7:0] <= buffer_blue[15:8]) +
(buffer_blue[7:0] <= buffer_blue[31:24]) + (buffer_blue[7:0] <= blue);

      assign blue_4_greater_than_score = (blue >= buffer_blue[23:16]) + (blue
>= buffer_blue[15:8]) + (blue >= buffer_blue[7:0]) + (blue >=
buffer_blue[31:24] );
      assign blue_4_less_than_score = (blue <= buffer_blue[23:16]) + (blue <=
buffer_blue[15:8]) + (blue <= buffer_blue[7:0]) + (blue <= buffer_blue[31:24]
);
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//////
// Company:
// Engineer: Rob Crowell
//
// Create Date:    23:54:24 05/08/2007
// Design Name:
// Module Name:    memory_controller
// Project Name:
// Target Devices:
// Tool versions:
// Description: This module wraps a dual-port memory and handles setting up
the we signals.  It also converts
//              (row, col) into an address.
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
//////
module memory_controller(clock_2x, reset, vga_row, vga_col, vga_rgb,
ntsc_row, ntsc_col, ntsc_rgb, ntsc_data_ready, particle_row, particle_col,
particle_rgb,
                                                    ram_address, ram_data,
ram_ce_b, ram_cen_b, ram_oe_b, ram_we_b, ram_bwe_b, ram_adv_ld);
      input wire clock_2x;
      input wire reset;

      input wire [9:0] vga_row;
      input wire [9:0] vga_col;
      output wire [23:0] vga_rgb;

      input wire [9:0] ntsc_row;
      input wire [9:0] ntsc_col;
      input wire [23:0] ntsc_rgb;
      input wire ntsc_data_ready;

      input wire [9:0] particle_row;
      input wire [9:0] particle_col;
      output wire [23:0] particle_rgb;

      output wire [18:0] ram_address;
      inout wire [35:0] ram_data;
      output wire ram_ce_b;
      output wire ram_cen_b;
      output wire ram_oe_b;
      output wire ram_we_b;
      output wire [3:0] ram_bwe_b;
      output wire ram_adv_ld;

      // Translate (row, col) into an address
      // Note that this function will not allow the upper half of the rows to
be used
```

```
	// In reality, we only need a small number of these, so we don't lose
too many pixels (approx 13 rows off the bottom)
	function [18:0] ADDRESS;
		input [9:0] row;
		input [9:0] col;
		begin
			ADDRESS = {row[8:0], col[9:0]};
		end
	endfunction

	// Instantiate a dual-port ZBT RAM
	// Port one is dedicated to VGA reads
	// Port two is muxed between ntsc writes and particle reads
	reg we_in2;
	reg [18:0] vga_address;
	reg [18:0] address_in2;
	wire [18:0] address_out1;
	wire [18:0] address_out2;
	wire [11:0] dummy1, dummy2;
	wire we_out1, we_out2;
	zbt_dual_port dual_port_ram(
		.clock_2x(clock_2x),
		.reset(reset),
		.we_in1(1'b0),
		.we_in2(we_in2),
		.address_in1(vga_address),
		.address_in2(address_in2),
		.data_in1(36'd0),
		.data_in2({12'd0, ntsc_rgb}),
		.we_out1(we_out1),
		.we_out2(we_out2),
		.address_out1(address_out1),
		.address_out2(address_out2),
		.data_out1({dummy1, vga_rgb}),
		.data_out2({dummy2, particle_rgb}),
		.ram_address(ram_address),
		.ram_data(ram_data),
		.ram_cen_b(ram_cen_b),
		.ram_ce_b(ram_ce_b),
		.ram_oe_b(ram_oe_b),
		.ram_we_b(ram_we_b),
		.ram_bwe_b(ram_bwe_b),
		.ram_adv_ld(ram_adv_ld)
	);

	// Switch which signals are routed to the RAM combinationally
	always @ ( * ) begin
		vga_address = ADDRESS(vga_row, vga_col);

		//to prevent the overflow, don't write any ntsc_row that's bigger
than (2^9)-1
		if(ntsc_data_ready && ntsc_row[9] == 0) begin
			we_in2 = 1;
		end else begin
			we_in2 = 0;
		end
```

```
            //if ntsc_data_ready, then give the zbt the ntsc address;
otherwise use the particle address
            if(ntsc_data_ready) begin
                  address_in2 = ADDRESS(ntsc_row, ntsc_col);
            end else begin
                  address_in2 = ADDRESS(particle_row, particle_col);
            end
      end

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
/////
// Company:
// Engineer: Rob Crowell
//
// Create Date:    16:23:19 05/15/2007
// Design Name:
// Module Name:    mouse_pointer
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
/////
module mouse_pointer(mouse_x, mouse_y, line_count, pixel_count, contains);
    input wire [9:0] mouse_x;
    input wire [9:0] mouse_y;
       input wire [9:0] line_count;
       input wire [9:0] pixel_count;
    output reg contains;

      parameter HEIGHT = 3'd3;

      always @ * begin
            if((line_count > mouse_y - HEIGHT) && (line_count < mouse_y +
HEIGHT) && (pixel_count > mouse_x - HEIGHT) && (pixel_count < mouse_x +
HEIGHT)) begin
                  contains = 1;
            end else begin
                  contains = 0;
            end
      end

endmodule
```

```
`timescale 1ns / 1ps
`default_nettype none

//////////////////////////////////////////////////////////////////////////////////
/////
// Company:
// Engineer: Rob Crowell
//
// Create Date:    22:25:37 04/24/2007
// Design Name:
// Module Name:    ntsc_decoder
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
/////

module ntsc_decoder(llc, reset, tv_in_ycrcb, ycrcb_out, ntsc_type, row, col,
state);
      input llc;
      input reset;
      input [19:0] tv_in_ycrcb;
      output [9:0] ycrcb_out;
      output [1:0] ntsc_type;
      output [9:0] row;
      output [9:0] col;
      output [6:0] state;

      // Parameters determining the type of output being decoded (for the
interpolator)
      parameter NO_DATA = 0;
      parameter NTSC_CR = 1;
      parameter NTSC_CB = 2;
      parameter NTSC_Y = 3;

      function XY_VALID;
            input [7:0] z;

            begin
                  XY_VALID = (z[7] == 1) && (z[3] == (z[5] ^ z[4])) && (z[2]
== (z[6] ^ z[4])) && (z[1] == (z[6] ^ z[5])) && (z[0] == (z[6] ^ z[5] ^
z[4]));
            end

      endfunction

      // Paramters related to the length of a line in NTSC
      parameter ACTIVE_SAMPLES = 10'd720;
      parameter LINE_SAMPLES = 10'd858;
```

```
// State machine states
parameter FIND_SAV_01_INIT = 0;
parameter FIND_SAV_01_10 = 1;
parameter FIND_SAV_01_FF = 2;
parameter FIND_SAV_01_00_0 = 3;
parameter FIND_SAV_01_00_1 = 4;
parameter FIND_SAV_01_XY = 5;
parameter FIND_SAV_00_INIT = 6;
parameter FIND_SAV_00_10 = 7;
parameter FIND_SAV_00_FF = 8;
parameter FIND_SAV_00_00_0 = 9;
parameter FIND_SAV_00_00_1 = 10;
parameter FIND_SAV_00_XY = 11;

parameter START_FIELD_0 = 12;
parameter SAV_FIELD_0 = 13;
parameter YB_FIELD_0 = 14;
parameter CR_FIELD_0 = 15;
parameter YR_FIELD_0 = 16;
parameter CB_FIELD_0 = 17;
parameter EAV_FIELD_0_INIT = 18;
parameter EAV_C_FIELD_0 = 19;
parameter EAV_Y_FIELD_0 = 20;
parameter VERTICAL_BLANKING_FIELD_0_INIT = 21;
parameter VERTICAL_BLANKING_Y_FIELD_0 = 22;
parameter VERTICAL_BLANKING_C_FIELD_0 = 23;

parameter START_FIELD_1 = 24;
parameter SAV_FIELD_1 = 25;
parameter YB_FIELD_1 = 26;
parameter CR_FIELD_1 = 27;
parameter YR_FIELD_1 = 28;
parameter CB_FIELD_1 = 29;
parameter EAV_FIELD_1_INIT = 30;
parameter EAV_C_FIELD_1 = 31;
parameter EAV_Y_FIELD_1 = 32;
parameter VERTICAL_BLANKING_FIELD_1_INIT = 33;
parameter VERTICAL_BLANKING_Y_FIELD_1 = 34;
parameter VERTICAL_BLANKING_C_FIELD_1 = 35;

reg [6:0] state, next;
reg [9:0] row, col;
reg [1:0] ntsc_type;
reg [7:0] ycrcb;
reg [9:0] ycrcb_out;
always @ (posedge llc) begin
      if(reset) begin
            state <= FIND_SAV_01_INIT;
            row <= 0;
            col <= 0;
            ycrcb <= 0;
            ycrcb_out <= 0;

      end else begin
            state <= next;
```

```
                  ycrcb <= tv_in_ycrcb[19:12];
                  ycrcb_out <= tv_in_ycrcb[19:10];
                  /*
                  if(row < 10'd150) begin
                        ycrcb_out <= 24'hFF0000;
                  end else begin
                        ycrcb_out <= 24'h00FF00;
                  end
                  */

                  //when we're processing a Y, increment the col count
                  if(next == CB_FIELD_0 || next == CR_FIELD_0 || next ==
EAV_C_FIELD_0 || next == EAV_FIELD_0_INIT || next ==
VERTICAL_BLANKING_C_FIELD_0 || next == CB_FIELD_1 || next == CR_FIELD_1 ||
next == EAV_C_FIELD_1 || next == EAV_FIELD_1_INIT || next ==
VERTICAL_BLANKING_C_FIELD_1) begin
                        col <= col + 1;

                  //when starting field 0, reset the row and col counts
                  end else if(next == START_FIELD_0) begin
                        row <= 0;   //do lines 0, 2, 4, 6, ...
                        col <= 0;

                  //when starting field 1, reset the row and col counts
                  end else if(next == START_FIELD_1) begin
                        row <= 1;   //do lines 1, 3, 5, 7, ...
                        col <= 0;

                  //when starting a new line, reset the col count to 0
                  //due to interlacing, we're actually moving down 2 rows
                  end else if(next == SAV_FIELD_0 || next ==
VERTICAL_BLANKING_FIELD_0_INIT || next == SAV_FIELD_1 || next ==
VERTICAL_BLANKING_FIELD_1_INIT) begin
                        col <= 0;
                        row <= row + 2;
                  end
            end
      end

      always @ ( * ) begin

            //defaults
            ntsc_type = NO_DATA;

            case(state)

                  //
                  // Orientation Begins
                  //

                  // Find an SAV with F=0, V=1
                  //we are liberal here; a sequence 80_10_FF_00_00_XY is
enough to match
                  FIND_SAV_01_INIT: begin
                        if(ycrcb == 8'h80) begin
                              next = FIND_SAV_01_10;
                        end else begin
```

```
                        next = FIND_SAV_01_INIT;
                    end
                end

                FIND_SAV_01_10: begin
                    if(ycrcb == 8'h10) begin
                        next = FIND_SAV_01_FF;   //80_10 has been seen
                    end else begin
                        next = FIND_SAV_01_INIT;
                    end
                end

                FIND_SAV_01_FF: begin
                    if(ycrcb == 8'hFF) begin
                        next = FIND_SAV_01_00_0;   //80_10_FF has been
seen
                    end else if(ycrcb == 8'h80) begin
                        next = FIND_SAV_01_10;
                    end else begin
                        next = FIND_SAV_01_INIT;
                    end
                end

                FIND_SAV_01_00_0: begin
                    if(ycrcb == 8'h00) begin
                        next = FIND_SAV_01_00_1;   //80_10_FF_00 has
been seen
                    end else if(ycrcb == 8'h80) begin
                        next = FIND_SAV_01_10;
                    end else begin
                        next = FIND_SAV_01_INIT;
                    end
                end

                FIND_SAV_01_00_1: begin
                    if(ycrcb == 8'h00) begin
                        next = FIND_SAV_01_XY;   //80_10_FF_00_00 has
been seen
                    end else if(ycrcb == 8'h80) begin
                        next = FIND_SAV_01_10;
                    end else begin
                        next = FIND_SAV_01_INIT;
                    end
                end

                FIND_SAV_01_XY: begin
                    if(XY_VALID(ycrcb) && ycrcb[6] == 0 && ycrcb[5] == 1
&& ycrcb[4] == 0) begin
                        next = FIND_SAV_00_INIT;   //80_10_FF_00_00_XY
(H=0) has been seen!
                    end else if(ycrcb == 8'h80) begin
                        next = FIND_SAV_01_10;
                    end else begin
                        next = FIND_SAV_01_INIT;
                    end
                end
```

```
                  // Find a SAV with F=0, V=0
                  //this follows a F=1, V=1 match, meaning that we've
identified the start of field 0
                  //we have to be a tad more careful here, because we don't
have unlimited retries
                  FIND_SAV_00_INIT: begin
                        if(ycrcb == 8'h80) begin
                              next = FIND_SAV_00_10;
                        end else begin
                              next = FIND_SAV_00_INIT;
                        end
                  end

                  FIND_SAV_00_10: begin
                        if(ycrcb == 8'h10) begin
                              next = FIND_SAV_00_FF;
                        end else if(ycrcb == 8'h80) begin
                              next = FIND_SAV_00_10;
                        end else begin
                              next = FIND_SAV_00_INIT;
                        end
                  end

                  FIND_SAV_00_FF: begin
                        if(ycrcb == 8'hFF) begin
                              next = FIND_SAV_00_00_0;
                        end else if(ycrcb == 8'h80) begin
                              next = FIND_SAV_00_10;
                        end else begin
                              next = FIND_SAV_00_INIT;
                        end
                  end

                  FIND_SAV_00_00_0: begin
                        if(ycrcb == 8'h00) begin
                              next = FIND_SAV_00_00_1;
                        end else if(ycrcb == 8'h80) begin
                              next = FIND_SAV_00_10;
                        end else begin
                              next = FIND_SAV_00_INIT;
                        end
                  end

                  FIND_SAV_00_00_1: begin
                        if(ycrcb == 8'h00) begin
                              next = FIND_SAV_00_XY;
                        end else if(ycrcb == 8'h80) begin
                              next = FIND_SAV_00_10;
                        end else begin
                              next = FIND_SAV_00_INIT;
                        end
                  end

                  FIND_SAV_00_XY: begin
                        //make sure H=0 (SAV signal)
                        if(XY_VALID(ycrcb) && ycrcb[6] == 0 && ycrcb[5] == 0
&& ycrcb[4] == 0) begin
```

```
                              next = START_FIELD_0;

                        //H can be anything; H=1 means EAV found so just keep
looking
                        end else if(XY_VALID(ycrcb) && ycrcb[6] == 0 &&
ycrcb[5] == 1) begin
                              next = FIND_SAV_00_INIT;
                        end else begin
                              next = FIND_SAV_01_INIT;
                        end
                  end



                  //
                  // Field 0 Begins
                  //

                  // The beginning of video for field 0
                  START_FIELD_0: begin
                        ntsc_type = NTSC_CB;
                        next = YB_FIELD_0;
                  end

                  // A new line of field 0 is starting (not the first line)
                  SAV_FIELD_0: begin
                        ntsc_type = NTSC_CB;
                        next = YB_FIELD_0;
                  end

                  YB_FIELD_0: begin
                        ntsc_type = NTSC_Y;
                        next = CR_FIELD_0;
                  end

                  CR_FIELD_0: begin
                        ntsc_type = NTSC_CR;
                        next = YR_FIELD_0;
                  end

                  YR_FIELD_0: begin
                        ntsc_type = NTSC_Y;
                        if(col < (ACTIVE_SAMPLES - 1)) begin   //this sample
is less than 719
                              next = CB_FIELD_0;
                        end else begin   //this sample is 719; the next
sample is the start of EAV
                              next = EAV_FIELD_0_INIT;
                        end
                  end

                  CB_FIELD_0: begin
                        ntsc_type = NTSC_CB;
                        next = YB_FIELD_0;
                  end

                  EAV_FIELD_0_INIT: begin
```

```
                        next = EAV_Y_FIELD_0;
                  end

                  // Two EAV states so that we can increment the count only
every other time (valid counts)
                  // We stay in the EAV states until an SAV is found, so this
state incorporates the 80_10 horiz blanking too
                  EAV_C_FIELD_0: begin
                        next = EAV_Y_FIELD_0;
                  end

                  // This state increments the col count
                  // Somewhere in this state we will hit the end of line
sample length, and will see SAV 0 or SAV vertical blanking
                  EAV_Y_FIELD_0: begin
                        if(col < (LINE_SAMPLES - 1)) begin
                              next = EAV_C_FIELD_0;

                        end else begin   //col == (LINE_SAMPLES - 1)

                              //SAV for field 0
                              if(XY_VALID(ycrcb) && ycrcb[6] == 0 && ycrcb[5]
== 0 && ycrcb[4] == 0) begin
                                    next = SAV_FIELD_0;

                              //SAV for vertical blanking of field 0
                              end else if(XY_VALID(ycrcb) && ycrcb[6] == 0 &&
ycrcb[5] == 1 && ycrcb[4] == 0) begin
                                    next = VERTICAL_BLANKING_FIELD_0_INIT;

                              //don't see a valid SAV frame... give up and
start completely over!
                              end else begin
                                    next = FIND_SAV_01_INIT;
                              end
                        end
                  end

                  // We've started the first round of vblank for field 0
                  //col is reset to 0 when this state is entered
                  VERTICAL_BLANKING_FIELD_0_INIT: begin
                        next = VERTICAL_BLANKING_Y_FIELD_0;
                  end

                  VERTICAL_BLANKING_Y_FIELD_0: begin
                        if(col < (LINE_SAMPLES - 1)) begin
                              next = VERTICAL_BLANKING_C_FIELD_0;

                        end else begin   //col == (LINE_SAMPLES - 1)

                              //SAV for another row of blanking (field value
is ignored, since it doesn't matter)
                              if(XY_VALID(ycrcb) && ycrcb[5] == 1 && ycrcb[4]
== 0) begin
                                    next = VERTICAL_BLANKING_FIELD_0_INIT;

                              //SAV for the start of field 1
```

```
                              end else if(XY_VALID(ycrcb) && ycrcb[6] == 1 &&
ycrcb[5] == 0 && ycrcb[4] == 0) begin
                                    next = START_FIELD_1;

                              //didn't see a valid SAV frame... we lose!
                              end else begin
                                    next = FIND_SAV_01_INIT;
                              end
                        end
                  end

                  // The C part of a blanking sample is never interesting,
but needed to keep the col count right
                  VERTICAL_BLANKING_C_FIELD_0: begin
                        next = VERTICAL_BLANKING_Y_FIELD_0;
                  end



                  //
                  // Field 1 Begins
                  //

                  START_FIELD_1: begin
                        ntsc_type = NTSC_CB;
                        next = YB_FIELD_1;
                  end

                  SAV_FIELD_1: begin
                        ntsc_type = NTSC_CB;
                        next = YB_FIELD_1;
                  end

                  YB_FIELD_1: begin
                        ntsc_type = NTSC_Y;
                        next = CR_FIELD_1;
                  end

                  CR_FIELD_1: begin
                        ntsc_type = NTSC_CR;
                        next = YR_FIELD_1;
                  end

                  YR_FIELD_1: begin
                        ntsc_type = NTSC_Y;
                        if(col < (ACTIVE_SAMPLES - 1)) begin   //this sample
is less than 719
                              next = CB_FIELD_1;
                        end else begin   //this sample is 719; the next
sample is the start of EAV
                              next = EAV_FIELD_1_INIT;
                        end
                  end

                  CB_FIELD_1: begin
                        ntsc_type = NTSC_CB;
                        next = YB_FIELD_1;
```

```
                  end

                  EAV_FIELD_1_INIT: begin
                        next = EAV_Y_FIELD_1;
                  end

                  EAV_C_FIELD_1: begin
                        next = EAV_Y_FIELD_1;
                  end

                  EAV_Y_FIELD_1: begin
                        if(col < (LINE_SAMPLES - 1)) begin
                              next = EAV_C_FIELD_1;

                        end else begin   //col == (LINE_SAMPLES - 1)

                              //SAV for field 1
                              if(XY_VALID(ycrcb) && ycrcb[6] == 1 && ycrcb[5]
== 0 && ycrcb[4] == 0) begin
                                    next = SAV_FIELD_1;

                              //SAV for vertical blanking of field 1
                              end else if(XY_VALID(ycrcb) && ycrcb[6] == 1 &&
ycrcb[5] == 1 && ycrcb[4] == 0) begin
                                    next = VERTICAL_BLANKING_FIELD_1_INIT;

                              //don't see a valid SAV frame... give up and
start completely over!
                              end else begin
                                    next = FIND_SAV_01_INIT;
                              end
                        end
                  end

                  VERTICAL_BLANKING_FIELD_1_INIT: begin
                        next = VERTICAL_BLANKING_Y_FIELD_1;
                  end

                  VERTICAL_BLANKING_Y_FIELD_1: begin
                        if(col < (LINE_SAMPLES - 1)) begin
                              next = VERTICAL_BLANKING_C_FIELD_1;

                        end else begin   //col == (LINE_SAMPLES - 1)

                              //SAV for another row of blanking (field value
is ignored, since it doesn't matter)
                              if(XY_VALID(ycrcb) && ycrcb[5] == 1 && ycrcb[4]
== 0) begin
                                    next = VERTICAL_BLANKING_FIELD_1_INIT;

                              //SAV for the start of field 0
                              end else if(XY_VALID(ycrcb) && ycrcb[6] == 0 &&
ycrcb[5] == 0 && ycrcb[4] == 0) begin
                                    next = START_FIELD_0;

                              //didn't see a valid SAV frame... we lose!
                              end else begin
```

```
                                    next = FIND_SAV_01_INIT;
                              end
                        end
                  end

                  VERTICAL_BLANKING_C_FIELD_1: begin
                        next = VERTICAL_BLANKING_Y_FIELD_1;
                  end

            endcase
      end

endmodule
```

```
`timescale 1ns / 1ps
`default_nettype none

//////////////////////////////////////////////////////////////////////////////////
/////
// Company:
// Engineer: Rob Crowell
//
// Create Date:    22:25:37 04/24/2007
// Design Name:
// Module Name:    ntsc_interpolator
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
/////
module ntsc_interpolator(clock, reset, ntsc_data, ntsc_row, ntsc_col,
ntsc_type, y, cr, cb, row, col, data_ready);
      input clock;
      input reset;
      input [9:0] ntsc_data;
      input [9:0] ntsc_row;
      input [9:0] ntsc_col;
      input [1:0] ntsc_type;

      output [9:0] y;
      output [9:0] cr;
      output [9:0] cb;
      output [9:0] row;
      output [9:0] col;
      output data_ready;

      // The decoder indicates which value it is sending through ntsc_field
      parameter NO_DATA = 0;
      parameter NTSC_CR = 1;
      parameter NTSC_CB = 2;
      parameter NTSC_Y = 3;

      // A slightly better interpolator that averages the values of the Cr
and Cb from before and after
      reg [9:0] row, col;
      reg [9:0] y;
      reg [9:0] cr, cr0, cr1;
      reg [9:0] cb, cb0, cb1;
      reg [10:0] cr_sum, cb_sum;
      reg data_ready;

      always @ (posedge clock) begin
            if(reset) begin
```

```
		row <= 0;
		col <= 0;
		y <= 0;

		cr <= 0;
		cr0 <= 0;
		cr_sum <= 0;

		cb <= 0;
		cb0 <= 0;
		cb_sum <= 0;
		data_ready <= 0;
	end else begin

		// Cr[i] = (Cr[i-1] + Cr[i+1]) / 2
		cr <= (cr_sum) >> 1'b1;
		cb <= (cb_sum) >> 1'b1;
		row <= ntsc_row;
		col <= ntsc_col;

		case(ntsc_type)
			NTSC_Y: begin
				y <= ntsc_data;
				cr0 <= cr0;
				cr_sum <= cr_sum;

				cb0 <= cb0;
				cb_sum <= cb_sum;

				data_ready <= 1;
			end

			NTSC_CR: begin
				y <= y;
				cr0 <= ntsc_data;
				cr_sum <= (ntsc_data + cr0);

				cb0 <= cb0;
				cb_sum <= cb_sum;

				data_ready <= 0;
			end

			NTSC_CB: begin
				y <= y;
				cr0 <= cr0;
				cr_sum <= cr_sum;

				cb0 <= ntsc_data;
				cb_sum <= (ntsc_data + cb0);

				data_ready <= 0;
			end

			default: begin
				y <= y;
				cr0 <= cr0;
```

```
                                cr_sum <= cr_sum;

                                cb0 <= cb0;
                                cb_sum <= cb_sum;

                                data_ready <= 0;
                            end
                        endcase
                end
        end

endmodule
```

```
`timescale 1ns / 1ps
`default_nettype none

//////////////////////////////////////////////////////////////////////////////////
/////
// Company:
// Engineer: Lyric Doshi
//
// Create Date:    21:40:41 04/29/2007
// Design Name:
// Module Name:    particle_fsm
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/*
This module runs the particle filter. at a basic level, it takes a start
signal whenever it will have access to the memory controller and performs its
actions. if a new target is being specified, the initialize signal should be
asserted. the assertion will move the module to a state where it awaits a
start signal to initialize itself. at this point, the target position is
registered. operation will not start till the next start. upon a start, the
module checks to see if pixels are alive using a minor FSM and then sending
requests for data to the memory controller and receiveing pixel color data
back. the vga display can also send an address and get a 1 bit response
whethere that address corresponds to a particle existing at that location.
The module outputs a busy signal, an all_dead pulse when all particles die,
and a hasValidTarget signal that indicates if the fsm is currently tracking a
target.  The module outputs the com_x and com_y, the centers of mass in x and
y directions of the particles. it also allows saturation and value tolerances
to be tweaked with switches at present.

the rest of the signals exist for debugging purposes primarly, but were not
removed from the module because it works as is, so as not to risk breaking
some functionality, and due to time constraints.


*/

//////////////////////////////////////////////////////////////////////////////////
/////
module particle_fsm(clock, reset, start, initialize, ext_target_x,
ext_target_y, random_number_x, random_number_y, com_x, com_y, particle_row,
particle_col, pixel_color, done, ext_addr, dout,want_address, state,next,
busy, dead_pointer, live_pointer, particle_check_counter,
particle_is_alive_result, contains, pixel_matches, row_delay_3, col_delay_3,
all_dead, divisor, dividend, quotient, pixel_x,
pixel_y,addr_vga,dout_particle_status, SAT_INPUT, VAL_INPUT, hasValidTarget);
            //hue test
            input wire [2:0] SAT_INPUT;
```

```
            input wire [3:0] VAL_INPUT;

            //del me
            output [7:0] quotient, divisor;
            output [17:0] dividend;
            output [10:0] pixel_x, pixel_y;

            //for finding out if a particle exists at a location
            input [19:0] addr_vga;
            output dout_particle_status;

            //THE LAST two ARGS EXPOSE THE RAM! DELETE THESE LATER!
            input [8:0] ext_addr;
            output [35:0] dout;
            input want_address;
          output particle_is_alive_result;
            output [7:0] dead_pointer, live_pointer;
            output [7:0] particle_check_counter;
            //END EXPOSITION
        input wire clock;
    input wire reset;
    input wire start;
    input wire initialize;
    input wire signed [10:0] ext_target_x;
    input wire signed [10:0] ext_target_y;

       //these are some random numbers used to determine new velocities for
newly spawned particles
       input wire signed [10:0] random_number_x;
       input wire signed [10:0] random_number_y;
    output reg [9:0] com_x;
    output reg [9:0] com_y;
      output reg done;
      output all_dead;
      //for dealing with the ZBT interface
        input wire [23:0] pixel_color;

        output wire [9:0] particle_row;
        output wire [9:0] particle_col;

      wire [9:0] particle_is_alive_particle_row,
particle_is_alive_particle_col;
      reg [9:0] particle_fsm_particle_row, particle_fsm_particle_col;

      //allows this module to set row and col sent to ram instead of the
particle_is_alive module
      reg particle_control_ram;

      assign particle_row = particle_control_ram ?  particle_fsm_particle_row
: particle_is_alive_particle_row;
      assign particle_col = particle_control_ram ? particle_fsm_particle_col
: particle_is_alive_particle_col;

      output reg busy;

      output reg [4:0] state, next;
      //color of the particles
```

```
      reg [23:0] particle_color;



      //for using the memory
      reg [8:0] addr;

      wire [8:0] mem_address;
      reg [35:0] din;
      reg we;
      wire [35:0] dout;
      reg [35:0] dout_delayed;

       reg signed [10:0] target_x;
    reg signed [10:0] target_y;

      //has valid target
      output reg hasValidTarget;

      assign mem_address = want_address ? ext_addr : addr;
      //particle memory module
      particle_memory particle_memory1(
            .addr(mem_address),
            .clk(clock),
            .din(din),
            .dout(dout),
            .we(we)
      );

      //used for addresses to the particle memory
      reg [8:0] address_counter;
      reg init_position;
      parameter [8:0] MAX_ADDRESS = 337;

      reg init_velocity;
       reg signed [10:0] init_velocity_x_counter,
next_init_velocity_x_counter;
       reg signed [10:0] init_velocity_y_counter,
next_init_velocity_y_counter;
      //constants for initialization
      parameter signed [10:0] VELOCITY_MIN = -12;
      parameter signed [10:0] VELOCITY_STEP = 2;
      parameter signed [10:0] VELOCITY_MAX = 12;

      //used to register the particle velocity after it is read from the ram
so that it can be used later if the particle is found to be alive so that the
particle can be moved
      reg signed [10:0] velocity_x, velocity_y, next_velocity_x,
next_velocity_y;

      parameter TOTAL_PARTICLES = 8'd169;

      //used for checking which particles are still alive and killing the
ones that are dead
      //flag to reset particle_check_counter
      reg particle_check;
      reg [7:0] particle_check_counter;
```

```
      //flag to indicate counter update needed
      reg particle_checked;

      reg clear_ram_counter;
      reg [19:0] ram_counter;

      //use for checking if all particles are dead. if so, pulse the all_dead
signal to indicate this
      reg all_dead;
      reg [7:0] all_dead_counter;
      reg particle_dead;
      reg init_all_dead_counter;

      //high to register new color
      reg save_target_color;

      //a list of particles. for each particle, the reg stores its status:
      // 00 - dead, 01 - spawning [i.e. newly made but not to be used to
initialize other particles], 10 - currently alive
      reg[1:0] particles [(TOTAL_PARTICLES-1):0];
      reg status_update;
      reg [1:0] status;
      reg [7:0] status_pointer;
      reg [7:0] live_pointer, next_live_pointer, dead_pointer,
next_dead_pointer;
      reg [7:0] activate_counter;
      //flag to reset activate_counter
      reg activate;

      //counter to wait for particle color initialization data
      reg clear_particle_color_initialize_counter;
      reg [4:0] particle_color_initialize_counter;

      //used to calculate center of mass of particles
      reg [18:0] sum_x, sum_y;
   reg [18:0] next_sum_x, next_sum_y;

      //to stall to wait for quotients from the divider
      reg [3:0] division_wait_count, next_division_wait_count;
      //inputs and outputs to division unit
      reg [7:0] divisor;
   reg [17:0] dividend;
      wire [17:0] quotient;
   wire [7:0] remainder;
      wire rfd;

      reg x_quotient, y_quotient;

       division_unit division_unit1 (
            .clk(clock),
            .dividend(dividend),
            .divisor(divisor),
            .quotient(quotient),
            .remainder(remainder),
            .rfd(rfd)
      );
```

```
            // Inputs
      reg we_particle_status;

      reg [19:0] addr_particle_status;
      wire [19:0] addr_vga;

      reg din_particle_status;
      // Outputs
      wire dout_particle_status;
      wire dout_dummy;

      reg [10:0] pixel_x_delayed;
      reg [10:0] pixel_y_delayed;
      reg [10:0] velocity_x_delayed;
      reg [10:0] velocity_y_delayed;

      // a ram to hold the positions of all the particles
      // the a ports are used to write positions
      // the b ports are used only to read positions
      particle_map_dual_port particle_map_dual_port1(
            .wea(we_particle_status),
            .web(1'd0),
            .clka(clock),
            .clkb(clock),
            .addra(addr_particle_status),
            .addrb(addr_vga),
            .douta(dout_dummy),
            .doutb(dout_particle_status),
            .dina(din_particle_status),
            .dinb(1'd0)
      );


      wire signed [10:0] random_x, random_y;
      reg next_random_x, next_random_y;

      //pseudo random number generators
      prng_25 prgn_x(.clock(clock), .reset(reset), .next(next_random_x),
.random(random_x));
      prng_23 prgn_y(.clock(clock), .reset(reset), .next(next_random_y),
.random(random_y));

      //particle status params
      parameter DEAD = 0;
      parameter SPAWNING = 1;
      parameter ALIVE = 2;

      wire [10:0] temp_position_x;
      wire [10:0] temp_position_y;

      //states
      parameter IDLE = 5'd0;
      parameter INITIALIZE_POSITION = 5'd1;
      parameter INITIALIZE_VELOCITY = 5'd2;
      parameter SEND_TARGET_COLOR_ADDRESS = 5'd3;
      parameter WAIT_FOR_TARGET_COLOR = 5'd4;
      parameter SAVE_TARGET_COLOR = 5'd5;
```

```
	parameter PARTICLE_CHECK_CONTROL = 5'd6;
	parameter CHECK_PARTICLE = 5'd7;
	parameter REGISTER_PARTICLE_VELOCITY = 5'd8;
	parameter WAIT_FOR_CHECK_RESULT = 5'd9;
	parameter PREPARE_TO_CALCULATE_COM = 5'd10;
	parameter DIVISION_COM_X = 5'd11;
	parameter DIVISION_COM_Y = 5'd12;
	parameter RESULT_X = 5'd13;
	parameter RESULT_Y = 5'd14;
	parameter SPAWN_CONTROL = 5'd15;
	parameter SPAWN_WRITE_POSITION = 5'd16;
	parameter SPAWN_READ_VELOCITY = 5'd17;
	parameter SPAWN_GET_READ_DATA = 5'd18;
	parameter SPAWN_WRITE_VELOCITY = 5'd19;
	parameter ACTIVATE_PARTICLES = 5'd20;
	parameter UPDATE_PARTICLE_POSITION = 5'd21;
	parameter CLEAR_RAM = 5'd22;
	parameter INITIALIZE_WAIT_FOR_START = 5'd23;
	//Initialize particle_is_alive module for internal use ***********
	// Inputs
	reg particle_is_alive_start;
	reg signed [10:0] particle_is_alive_x;
	reg signed [10:0] particle_is_alive_y;

		// Outputs
	wire particle_is_alive_busy;
	wire particle_is_alive_result;
	wire signed [10:0] pixel_x, pixel_y;

	//TEMP
	output wire contains;
	output wire pixel_matches;
	output wire [10:0] row_delay_3, col_delay_3;

	//checks to see if a particle is alive by checking the colors of pixels
in the particle
	particle_is_alive particle_is_alive1 (
		.reset(reset),
		.clock(clock),
		.start(particle_is_alive_start),
		.particle_color(particle_color),
		.x_external(particle_is_alive_x),
		.y_external(particle_is_alive_y),
		.row_count_external(particle_is_alive_particle_row),
		.col_count_external(particle_is_alive_particle_col),
		.pixel_color(pixel_color),
		.busy(particle_is_alive_busy),
		.result(particle_is_alive_result),
		.x(pixel_x),
		.y(pixel_y),
		.contains(contains),
		.pixel_matches(pixel_matches),
		.row_delay_3(row_delay_3),
		.col_delay_3(col_delay_3),
		.SAT_INPUT(SAT_INPUT),
		.VAL_INPUT(VAL_INPUT)
	);
```

```
      //******************************** end making particle_is_alive module
for internal use

      always @ (posedge clock) begin


            if(reset) begin
                  //temp CODE
                  com_x <= 10'd320;
                  com_y <= 10'd240;
                  state <= IDLE;
                  activate_counter <= 0;
                  address_counter <= 0;
                  live_pointer <= 0;
                  dead_pointer <= 0;
                  particle_color <= 0;
                  hasValidTarget <= 0;
                  ram_counter <= 0;
            end else begin
                  //initialize the color
                  if(initialize) begin
                        target_x <= ext_target_x;
                        target_y <= ext_target_y;
                        hasValidTarget <= 1;
                  end

                  //registers the color of the target during intialization
                  if(save_target_color) begin
                        particle_color <= pixel_color;
                  end else begin
                        particle_color <= particle_color;
                  end

                  //clears intializiation counter
                  if(clear_particle_color_initialize_counter) begin
                        particle_color_initialize_counter <= 0;
                  end else begin
                        particle_color_initialize_counter <=
particle_color_initialize_counter + 1;
                  end

                  //if all_dead, the module doesnt have a target any more
                  if (all_dead) begin
                        hasValidTarget <= 0;
                  end

                  if(init_position) begin
                        address_counter <= 0;
                  //if prepare to initialize velocities by resetting counters

                  end else if(init_velocity) begin
                        init_velocity_x_counter <= VELOCITY_MIN;
                        init_velocity_y_counter <= VELOCITY_MIN;
                        address_counter <= 1;
                  end else begin
                        //update to next value;
```

```
                        init_velocity_x_counter <=
next_init_velocity_x_counter;
                        init_velocity_y_counter <=
next_init_velocity_y_counter;
                        //for when this signal is being used, it needs to be
incremented by 2's. after that, we dont really care what it does
                        address_counter <= address_counter + 9'd2;
                  end

                  if(activate) begin
                        //clear the counter and then it starts incrementing
again
                        activate_counter <= 0;
                  end else begin
                        //after it clears, it counts up. we stop using it
after a while, and don't care if it overflows
                        //this is only used with a clear first
                        //this behavior works because when activate_counter
is being used, it needs to be updated every cycle
                        activate_counter <= activate_counter + 1;
                  end


                  if(particle_check) begin
                        //clear the counter and then it starts incrementing
again
                        particle_check_counter <= 0;
                        //also clear the sum_x, sum_y variables so that we
can add up the x's and y's for the living particles
                        sum_x <= 0;
                        sum_y <= 0;
                  end else begin
                        //otherwise update sum_x,sum_y normally
                        sum_x <= next_sum_x;
                        sum_y <= next_sum_y;

                        //after it clears, it counts up whenever
particle_checked goes high to indicate that the counter should be
incremented. we need this set-up because the counter is not incremented every
cycle
                        if(particle_checked) begin
                              particle_check_counter <=
particle_check_counter + 1;
//                            if(particle_check_counter == 8'd97) begin
//                                  com_x <= pixel_x[9:0];
//                                  com_y <= pixel_y[9:0];
//                            end
                        end else begin
                              //otherwise leave the counter as is
                              particle_check_counter <=
particle_check_counter;
                        end
                  end

                  //if there is a status_update, the status_pointer indicates
the particle to update and status provides the updated information for the
particle
```

```
                  if(status_update) begin
                        particles[status_pointer] <= status;
                  end

                  //if a dead particle is found, augment the
all_dead_counter. clear the counter if init_all_dead_counter is high
                  if(init_all_dead_counter) begin
                        //clear the counter
                        all_dead_counter <= 0;
                  end else if (particle_dead) begin
                        //another dead one is found!
                        all_dead_counter <= all_dead_counter + 1;
                  end else begin
                        // no change
                        all_dead_counter <= all_dead_counter;
                  end

                  //if the quotient for com_x is ready, register the value
                  if(x_quotient) begin
                        com_x <= quotient;
                  end else begin
                        com_x <= com_x;
                  end

                  //if the quotient for com_y is ready, register the value
                  if(y_quotient) begin
                        com_y <= quotient;
                  end else begin
                        com_y <= com_y;
                  end

                  //clear counter used to go through all address in the
particle_map_dual_port and clear data
                  if(clear_ram_counter) begin
                        ram_counter <= 0;
                  end else begin
                        ram_counter <= ram_counter + 1;
                  end

                  live_pointer <= next_live_pointer;
                  dead_pointer <= next_dead_pointer;
                  velocity_x <= next_velocity_x;
                  velocity_y <= next_velocity_y;
                  division_wait_count <= next_division_wait_count;
                  //dout one cycle delayed
                  dout_delayed <= dout;
                  state <= next;

                  pixel_x_delayed <= pixel_x;
                  pixel_y_delayed <= pixel_y;
                  velocity_x_delayed <= velocity_x;
                  velocity_y_delayed <= velocity_y;

            end

      end
```

```
	always @ ( activate ) begin

		//default these to zero
		init_position = 0;
		init_velocity = 0;
		activate = 0;
		status_pointer = 0;
		status = 0;
		status_update = 0;
		addr = 0;
		din = 0;
		we = 0;
		done = 0;
		particle_check = 0;
		particle_checked = 0;
		particle_is_alive_start = 0;
		particle_is_alive_x = 11'd0;
		particle_is_alive_y = 11'd0;
		particle_dead = 0;
		init_all_dead_counter = 0;
		all_dead = 0;
		dividend = 0;
		divisor = 0;
		x_quotient = 0;
		y_quotient = 0;
		next_division_wait_count = 0;
		next_random_x = 0;
		next_random_y = 0;
		save_target_color = 0;
		particle_control_ram = 0;
		clear_particle_color_initialize_counter = 0;
		we_particle_status = 0;
		addr_particle_status = 20'd0;
		din_particle_status = 0;
		clear_ram_counter = 0;
		particle_fsm_particle_row = 0;
		particle_fsm_particle_col = 0;
		//default these to stay they same so they don't change unless
they need to be changed;
		next_live_pointer = live_pointer;
		next_dead_pointer = dead_pointer;
		next_velocity_x = velocity_x;
		next_velocity_y = velocity_y;
		next_sum_x = sum_x;
		next_sum_y = sum_y;
		//busy high except in idle
		busy = 1 ;



		case(state)

			IDLE: begin
				busy = 0;
				if(initialize) begin
					//if initializing, clear the RAM and select a
target
```

```
                              next = INITIALIZE_WAIT_FOR_START;
                        end else if(start && hasValidTarget) begin
                              //otherwise just begin checking particles if we
have at target we're tracking
                              particle_check = 1;
                              init_all_dead_counter = 1;
                              next = PARTICLE_CHECK_CONTROL;
                        end else begin
                              //stay here since everything has died (we have
no target, so wait for a new initialize)
                              next = IDLE;
                        end
                  end

                  INITIALIZE_WAIT_FOR_START: begin
                        //acknowledge initialize signal, now wait for start
to access data
                        if(start) begin
                              next = INITIALIZE_POSITION;
                              init_position = 1;
                        end else begin
                              next = INITIALIZE_WAIT_FOR_START;
                        end
                  end

                  //go through every position in memory that is 0 mod 2 and
write in the position for the particle. at initialization, these are all
(target_x, target_y)
                  INITIALIZE_POSITION: begin
                              //write them to ram
                              we = 1;
                              din = {14'd0,target_x,target_y};
                              addr = address_counter;
                              //update status
                              status_update = 1;

                              status = ALIVE;
                              //don't need the least sig bit... want to count
from 0-168 by ones vs 0-336 by twos
                              status_pointer = address_counter[8:1];
                        //this means the current address is the last one
                        if(address_counter + 9'd2 > MAX_ADDRESS) begin
                              next = INITIALIZE_VELOCITY;
                              init_velocity = 1;
                        end else begin
                              // go on to next position
                              next = INITIALIZE_POSITION;
                        end
                  end

                  INITIALIZE_VELOCITY: begin
                        //the following code emulates:
                        //for ( int vx = -12; vx <= 12; vx = vx+2)
                        //          for ( int vy = -12; vy <= 12; vy = vy+2)
                        //                write data for initialization
                        we = 1;
```

```
                        din =
{14'd0,init_velocity_x_counter,init_velocity_y_counter};
                        addr = address_counter;
                        if(init_velocity_x_counter > VELOCITY_MAX) begin
                              //now grab the color of the target from the zbt
                              next = SEND_TARGET_COLOR_ADDRESS;
                        end else begin
                              if(init_velocity_y_counter + VELOCITY_STEP >
VELOCITY_MAX) begin
                                    //completed one loop on y, increment x
and start next y loop
                                    next_init_velocity_x_counter =
init_velocity_x_counter + VELOCITY_STEP;
                                    next_init_velocity_y_counter =
VELOCITY_MIN;
                              end else begin
                                    //go to next y value for current x value
                                    next_init_velocity_y_counter =
init_velocity_y_counter + VELOCITY_STEP;
                                    next_init_velocity_x_counter =
init_velocity_x_counter;
                              end

                              // go on to next velocity
                              next = INITIALIZE_VELOCITY;
                        end
                  end

                  SEND_TARGET_COLOR_ADDRESS: begin
                        //ask for the color of the target location. sent to
memory controller
                        particle_control_ram = 1;
                        particle_fsm_particle_row = target_y;
                        particle_fsm_particle_col = target_x;
                        clear_particle_color_initialize_counter = 1;
                        next = WAIT_FOR_TARGET_COLOR;
                  end

                  WAIT_FOR_TARGET_COLOR: begin
                        //wait for zbt read and color conversion
                        if(particle_color_initialize_counter == 5'd23) begin
                              next = SAVE_TARGET_COLOR;
                        end else begin
                              next = WAIT_FOR_TARGET_COLOR;
                        end
                  end

                  SAVE_TARGET_COLOR: begin
                        //we're done initializing now, change state to
PARTICLE_CHECK_CONTROL
                        //dont care about x,y counters anymore
                        save_target_color = 1;

                        clear_ram_counter = 1;
                        next = CLEAR_RAM;
                  end
```

```
                  CLEAR_RAM: begin
                        //goes through all memory locations in the
particle_map_dual_port and sets all data = 0
                        if(ram_counter == 20'b1111_1111_1111_1111_1111) begin
                              next = IDLE;
                        end else begin
                              next = CLEAR_RAM;
                        end

                        we_particle_status = 1;
                        addr_particle_status = ram_counter;
                        din_particle_status = 0;
                  end

                  PARTICLE_CHECK_CONTROL: begin

                        if(all_dead_counter == TOTAL_PARTICLES) begin
                              //all the particles are dead. pulse all_dead
and return to IDLE
                              all_dead = 1;
                              done = 1;
                              busy = 0;
                              next = IDLE;
                        end else if(particle_check_counter ==
TOTAL_PARTICLES) begin
                              // all particles checked! now go to spawning
phase
                              next = PREPARE_TO_CALCULATE_COM;
                        end else if (particles[particle_check_counter] ==
ALIVE) begin
                              //we found a particle that needs to be
investigated. initiate a read to get it's position from the ram. the address
is the particle_check_counter * 2
                              addr = particle_check_counter << 1;
                              next = CHECK_PARTICLE;
                        end else begin
                              //current particle does not need further
processing. move on to the next one
                              particle_checked = 1;
                              next = PARTICLE_CHECK_CONTROL;
                        end
                  end

                  CHECK_PARTICLE: begin
                              //starts the FSM
                              particle_is_alive_start = 1;

                              particle_is_alive_x = dout[21:11];
                              particle_is_alive_y = dout[10:0];
                              //get the next read going for the particles's
velocity. address is particle_check_counter * 2 + 1
                              addr = (particle_check_counter << 1) + 1;
                              next = REGISTER_PARTICLE_VELOCITY;
                  end

                  REGISTER_PARTICLE_VELOCITY: begin
```

```
                        //update the velocity for the particle using data
from the ram
                        next_velocity_x = dout[21:11];
                        next_velocity_y = dout[10:0];
                        next = WAIT_FOR_CHECK_RESULT;
                  end

                  WAIT_FOR_CHECK_RESULT: begin
                        if(particle_is_alive_busy) begin
                              //until the check is complete, just wait here
                              next = WAIT_FOR_CHECK_RESULT;
                        end else begin

                              //the check is done. now if the result is high,
the particle is still alive. move it using velocities and write it's new
position to the ram
                              if(particle_is_alive_result) begin
                                    we = 1;
                                    //add velocity_x to the x component of
position, add velocity_y to the y component of position.
                                    din = {14'd0,{pixel_x +
velocity_x},{pixel_y + velocity_y}};
                                    //modifying the position of the particle,
address = particle_check_counter * 2
                                    addr = particle_check_counter << 1;

                                    //if the particle is alive, add its x and
y to sum_x and sum_y to calc center of mass of living particles
                                    //sign extend pixel_x
                                    if(pixel_x[10] == 0) begin
                                          next_sum_x = sum_x +
{7'd0,pixel_x};
                                    end else begin
                                          next_sum_x = sum_x +
{7'd1,pixel_x};
                                    end
                                    //sign extend pixel_x
                                    if(pixel_y[10] == 0) begin
                                          next_sum_y = sum_y +
{7'd0,pixel_y};
                                    end else begin
                                          next_sum_y = sum_y +
{7'd1,pixel_y};
                                    end


                                    //we write the new position
                                    next = UPDATE_PARTICLE_POSITION;

                              end else begin
                                    //if the result is low, the particle is
dead, so we change its status to DEAD
                                    status_update = 1;
                                    status = DEAD;
                                    status_pointer = particle_check_counter;
                                    //incremend the number of dead particles
found
```

```
                                    particle_dead = 1;


                                    //it's dead. we continue now to the next
particle
                                    particle_checked = 1;
                                    next = PARTICLE_CHECK_CONTROL;
                              end

                                    //clear position of particle
                                    we_particle_status = 1;
                                    addr_particle_status =
{pixel_x[9:0],pixel_y[9:0]};
                                    din_particle_status = 0;


                              end
                        end

                        UPDATE_PARTICLE_POSITION: begin
                              particle_checked = 1;
                              //update position of particle
                              we_particle_status = 1;
                              addr_particle_status =
{temp_position_x[9:0],temp_position_y[9:0]};
                              din_particle_status = 1;
                              next = PARTICLE_CHECK_CONTROL;
                        end

                        PREPARE_TO_CALCULATE_COM: begin
                              //wait till division unit is ready to allow adequate
time
                              if(rfd) begin
                                    next = DIVISION_COM_X;
                              end else begin
                                    next = PREPARE_TO_CALCULATE_COM;
                              end
                        end

                        DIVISION_COM_X: begin
                              dividend = sum_x[17:0];
                              //the number of living particles
                              divisor = TOTAL_PARTICLES - all_dead_counter;
                              //wait till division unit is ready, send it the first
division
                              if(rfd) begin
                                    next = DIVISION_COM_Y;
                              end else begin
                                    next = DIVISION_COM_X;
                              end
                        end

                        DIVISION_COM_Y: begin
                              dividend = sum_y[17:0];
                              //the number of living particles
                              divisor = TOTAL_PARTICLES - all_dead_counter;
```

```
                        //wait till division unit is ready, send it the
second division
                        if(rfd) begin
                              next = RESULT_X;
                        end else begin
                              next = DIVISION_COM_Y;
                        end
                  end

                  RESULT_X: begin
                        //wait till division completes, then save it and move
on to waiting for result_y
                        if(division_wait_count == 4'd12) begin
                              next = RESULT_Y;
                              x_quotient = 1;
                        end else begin
                              next_division_wait_count = division_wait_count
+ 1;
                              next = RESULT_X;
                        end

                  end

                  RESULT_Y: begin
                        //wait till division completes, then save the
quotient and move on to spawn control
                        if(division_wait_count == 4'd7) begin
                              next = SPAWN_CONTROL;
                              y_quotient = 1;
                        end else begin
                              next_division_wait_count = division_wait_count
+ 1;
                              next = RESULT_Y;
                        end
                  end

                  SPAWN_CONTROL: begin
                        if(dead_pointer == TOTAL_PARTICLES) begin
                              //all particles are alive now, so we now need
to change their statuses to alive
                              activate = 1;
                              next = ACTIVATE_PARTICLES;
                              next_dead_pointer = 0;
                              next_live_pointer = 0;
                        end else begin
                              //if we have found a living particle to use to
initialize a currently dead particle and a dead particle that needs to be
initialize, use them for processing
                              if(particles[live_pointer] == ALIVE &&
particles[dead_pointer] == DEAD) begin
                                    next = SPAWN_WRITE_POSITION;
                              end else if(particles[live_pointer] == ALIVE)
begin
                                    //we found a living particle, but not a
dead one yet, so keep looking for a dead one!
                                    next_dead_pointer = dead_pointer + 1;
                                    //try again!
```

```
                                    next = SPAWN_CONTROL;
                              end else if(particles[dead_pointer] == DEAD)
begin
                                    //we found a dead particle, but not a
living one yet, so keep looking for the next living one!
                                    //if live_pointer reached end of list,
reset it to front of list of particles
                                    if(live_pointer + 1 == TOTAL_PARTICLES)
begin
                                          next_live_pointer = 0;
                                    end else begin
                                          next_live_pointer = live_pointer +
1;
                                    end
                                    //try again!
                                    next = SPAWN_CONTROL;
                              end else begin
                                    //we still need the next living and dead
particles
                                    next_dead_pointer = dead_pointer + 1;

                                    //if live_pointer reached end of list,
reset it to front of list of particles
                                    if(live_pointer + 1 == TOTAL_PARTICLES)
begin
                                          next_live_pointer = 0;
                                    end else begin
                                          next_live_pointer = live_pointer +
1;
                                    end
                                    //try again!
                                    next = SPAWN_CONTROL;
                              end
                        end
                  end

                  SPAWN_WRITE_POSITION: begin
                        //the value of dead_pointer*2 is the address to which
we should write the new position for the particle being spawned. position is
the position of the center of mass
                        //    write data for spawned particle position
                        we = 1;
                        //the indiv 0's are there because com_x,com_y are 10
bits, but the particle x,y are 11 bits. they are shown explicitly to make it
more clear what is going on
                        din = {14'd0,1'd0,com_x,1'd0,com_y};
                        addr = dead_pointer << 1;
                        //also set the status of this particle to SPAWNING.
we set the status to spawning so that these particles are not used to
generate velocites for other particles that are spawned. thus, living, dead,
and currently being spawned particles can be distinguished.
                        status_update = 1;
                        status = SPAWNING;
                        status_pointer = dead_pointer;

                        //update position of particle
                        we_particle_status = 1;
```

```
                        addr_particle_status = {com_x[9:0],com_y[9:0]};
                        din_particle_status = 1;
                        next = SPAWN_READ_VELOCITY;
                  end

                  SPAWN_READ_VELOCITY: begin
                        //read out the velocites of the live particle being
used as to get velocity values for the spawning one. the (live_pointer * 2 +
1 ) gives the address for the live particles velocites in the ram
                        addr = ( live_pointer << 1 ) + 1;
                        next = SPAWN_GET_READ_DATA;
                  end

                  SPAWN_GET_READ_DATA: begin
                        //a wait state since we have to wait for data, can't
do write after read
                        next = SPAWN_WRITE_VELOCITY;
                  end

                  SPAWN_WRITE_VELOCITY: begin
                        //modify live particle's velocity using random
numbers and write to ram the velocity for the spawning particle
                        //the addr is the dead_pointer + 1
                        we = 1;
                        //dout has the data we just read out, use that to
determine data to write in
                        din =
{14'd0,dout_delayed[21:11]+random_x,dout_delayed[10:0]+random_y};
                        addr = (dead_pointer << 1) + 1;

                        //get next random number ready!
                        next_random_x = 1;
                        next_random_y = 1;

                        //increment the live and dead pointers by one to
reflect that we just took care of a spawning
                        next_dead_pointer = dead_pointer + 1;

                        //if live_pointer reached end of list, reset it to
front of list of particles
                        if(live_pointer + 1 == TOTAL_PARTICLES) begin
                              next_live_pointer = 0;
                        end else begin
                              next_live_pointer = live_pointer + 1;

                        end

                        next = SPAWN_CONTROL;
                  end

                  ACTIVATE_PARTICLES: begin
                        //now that all DEAD particles have been replaced, set
all particles status to ALIVE to prepare for the next iteration
                        if( activate_counter == TOTAL_PARTICLES) begin
                              next = IDLE;
                              done = 1;
                              busy = 0;
```

```
                        end else begin
                              //update status
                              status_update = 1;
                              status = ALIVE;
                              status_pointer = activate_counter;
                              next = ACTIVATE_PARTICLES;
                        end
                  end

            endcase
      end

      //temp variable so this value can be used in multiple places more
easily
      assign temp_position_x = pixel_x_delayed + velocity_x_delayed;
      assign temp_position_y = pixel_y_delayed + velocity_y_delayed;
endmodule
```

```
`timescale 1ns / 1ps
`default_nettype none

//////////////////////////////////////////////////////////////////////////////
/////
// Company:
// Engineer: Lyric Doshi
//
// Create Date:    22:20:29 04/24/2007
// Design Name:
// Module Name:    particle_is_alive
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
// this module takes a particle_color, and (x,y) and then when the start
signal is asserted, it goes through and checks all nine pixels in the
particle to make sure they are the same color as the particle [with some
range of difference from the exact pixel values allowed, eg, sat must be
within 60]. if at least THRESHOLD pixels are of the right color, the result
is 1. else the result is 0. to do this, the module gets data from the ZBT ram
for pixel hsv values. it sends row_count and col_count and receives a
pixel_color in return. the internal FSM accounts for the delay in sending an
addres to the ZBT and receiving data for that address.
// x_external, y_external are registered so that changes in the signal after
the start signal don't affect this module. Since particle_color is now
supposed to change, that value is *not* registered
// start must be pulsed
//////////////////////////////////////////////////////////////////////////////
/////
module particle_is_alive(reset, clock, start, particle_color, x_external,
y_external, row_count_external, col_count_external, pixel_color, busy,
result, x, y, contains, pixel_matches, row_delay_3, col_delay_3, SAT_INPUT,
VAL_INPUT);
    //temp
       output contains,pixel_matches;
       output [10:0] row_delay_3, col_delay_3;

       input wire reset;
       input wire clock;

    input wire start;
    input wire signed [10:0] x_external;
    input wire signed [10:0] y_external;
       input wire [23:0] particle_color;
       input wire [23:0] pixel_color;

       input wire [2:0] SAT_INPUT;
       input wire [3:0] VAL_INPUT;

       output reg busy;
```

```
    output reg result;
      // the x,y position associated with the outputted result
      // also register in the x,y so they are available throughout operation
      output reg signed [10:0] x, y;

      output wire [9:0] row_count_external, col_count_external;
      //for internal use. these are signed for checks to see if the point is
on the display
      reg signed [10:0] row_count, col_count;
      //row_count_external and col_count_external should be unsigned, so we
delete the MSB from row/col count. if they were negative, the external
versions will be garbage, but we dont care because these data will be ignored
anyway if row or col count are negative. otherwise this practice works fine
if they are both positive.
      assign row_count_external = row_count[9:0];
      assign col_count_external = col_count[9:0];

      reg clear_wait_counter;
      reg [4:0] wait_counter;


      reg [3:0] state, next;
      //count number of good pixels. if count > threshold, particle lives
      reg [3:0] count;
      //pixel_matches is one each cycles if the current pixel data is within
the correct ranges of the particle's color, otherwise it is 0
      reg pixel_matches;

      //used to register the pixel_color for a cycle
      reg [23:0] pixel_color_registered;

      //used to delay address sent to zbt for two cycles so that the address
sent is available when the data returns
      reg [10:0] row_delay_1, col_delay_1, row_delay_2, col_delay_2;
      //used to delay address sent to zbt for one more cycle because after
data gets back, we register it and use it on the next cycle
      reg [10:0] row_delay_3, col_delay_3;
      //this will be a shift register to delay the contains signal for 3
cycles so that it can be used with the data when it gets back from the ram
      reg[22:0] contains_delayer;

      //tells if the (row_count,col_count) point is on the vga display or not
      wire contains;

      //important constants
      //these define the buffer for which the hsv value is still considered
correct
      parameter HUE_VARIATION = 8'd10;
//    parameter SAT_VARIATION = 8'd60;
//    parameter VAL_VARIATION = 8'd60;
      parameter VAL_LOWER_THRESHOLD = 8'd25;
      parameter VAL_UPPER_THRESHOLD = 8'd235;
      parameter SAT_LOWER_THRESHOLD = 8'd25;

      wire[7:0] SAT_VARIATION;
      wire[7:0] VAL_VARIATION;
      //allow for thresholds to be changed
```

```
	assign SAT_VARIATION = {SAT_INPUT,5'b1_1111};
	assign VAL_VARIATION = {VAL_INPUT,4'b1111};

	//the number of particles that must be the right color for the particle
to survive
	parameter THRESHOLD = 4'd3;

	//boundary rectangle. if a pixel is outside the rectangle, the pixel is
considered to not have the right color
	vga_display_rectangle vga_display_rectangle1(.x(col_count),
.y(row_count), .contains(contains));

	//states
	parameter IDLE = 0;
	parameter ADD1 = 1;
	parameter ADD2 = 2;
	parameter ADD3 = 3;
	parameter ADD4 = 4;
	parameter ADD5 = 5;
	parameter ADD6 = 6;
	parameter ADD7 = 7;
	parameter ADD8 = 8;
	parameter WAIT = 9;
	parameter DATA = 10;
	parameter RESULTS = 11;

	always @ (posedge clock) begin
		if(reset) begin
			state <= IDLE;
			count <= 0;
			contains_delayer <= 0;
			x <= 0;
			y <= 0;
		end else begin
			state <= next;
			pixel_color_registered <= pixel_color;

			//clear count when we start the operations
			if(start) begin
				count <= 0;
				x <= x_external;
				y <= y_external;
			end else begin
				count <= count + pixel_matches;
				x <= x;
				y <= y;
			end

			//clear counter for use
			if(clear_wait_counter) begin
				wait_counter <= 0;
			end else begin
				wait_counter <= wait_counter + 1;
			end

			//update x,y delay registers: PROBABLY NO LONGER USED
			row_delay_1 <= row_count;
```

```
                col_delay_1 <= col_count;
                row_delay_2 <= row_delay_1;
                col_delay_2 <= col_delay_1;
                row_delay_3 <= row_delay_2;
                col_delay_3 <= col_delay_2;
                //update contains delay
                contains_delayer <= {contains,contains_delayer[22:1]};
          end

     end

     always @ ( * ) begin

          //its always busy except in the idle state, so easier to write it
this way
          busy = 1;
          //these don't matter unless we are sending an address..
          col_count = 0;
          row_count = 0;
          //this defaults to low so the counter keeps going
          clear_wait_counter = 0;
          //result doesn't matter until we're done
          result = 0;
          //check to see the pixel_color_registered of the data returned
from the ZBT for the pixel requested 2 states ago is within the ranges
allowed for the particle_color
          //this is complicated, so put this as default and force to 0
whenever we don't want this calcluation to take place
          if(contains_delayer[0]) begin
                //the pixel is in the display
                //set up special cases for extreme values of VAL and SAT
                if(particle_color[7:0] < VAL_LOWER_THRESHOLD &&
pixel_color_registered[7:0] < VAL_LOWER_THRESHOLD) begin
                     //the particle is essentially black... hue and sat
don't matter
                     pixel_matches = 1;
                end else if(particle_color[7:0] > VAL_UPPER_THRESHOLD &&
pixel_color_registered[7:0] > VAL_UPPER_THRESHOLD) begin
                     //the particle is essentially white... hue and sat
don't matter
                     pixel_matches = 1;
                end else if(particle_color[15:8] < SAT_LOWER_THRESHOLD &&
pixel_color_registered[15:8] < SAT_LOWER_THRESHOLD) begin
                     //the particle is gray... hue doesn't matter
                     pixel_matches =

                                          (

     (pixel_color_registered[7:0] >= particle_color[7:0]) &&

     (pixel_color_registered[7:0] - particle_color[7:0] ) < VAL_VARIATION
                                          )
                                          ||
                                          (
                                               (particle_color[7:0] >=
pixel_color_registered[7:0]) &&
```

```
                                                      (particle_color[7:0] -
pixel_color_registered[7:0] ) < VAL_VARIATION
                                                ) ;
                  end else begin
                        pixel_matches =
                                              ( //check hue. hue wraps around in
a circle, so unsigned subtraction takes care of this for us. we get x - y
(mod 256) which is what we want here
                                                (
pixel_color_registered[23:16] - particle_color[23:16] ) < HUE_VARIATION ||
                                                ( particle_color[23:16] -
pixel_color_registered[23:16] ) < HUE_VARIATION
                                               )
                                               &&
                                               ( //check sat. sat does not wrap
around, so we must make sure that x > y before computing x - y to ensure we
get a meaningful result
                                                (

      (pixel_color_registered[15:8] >= particle_color[15:8]) &&

      (pixel_color_registered[15:8] - particle_color[15:8] ) < SAT_VARIATION
                                                )
                                                ||
                                                (
                                                      (particle_color[15:8]
>= pixel_color_registered[15:8]) &&
                                                      (particle_color[15:8] -
pixel_color_registered[15:8] ) < SAT_VARIATION
                                                )
                                               )

                                               &&
                                               ( //check val. val does not wrap
around, so we must make sure that x > y before computing x - y to ensure we
get a meaningful result
                                                (

      (pixel_color_registered[7:0] >= particle_color[7:0]) &&

      (pixel_color_registered[7:0] - particle_color[7:0] ) < VAL_VARIATION
                                                )
                                                ||
                                                (
                                                      (particle_color[7:0] >=
pixel_color_registered[7:0]) &&
                                                      (particle_color[7:0] -
pixel_color_registered[7:0] ) < VAL_VARIATION
                                                )
                                               );
                  end

            end else begin
                        //if !contains, this does not contribute to pixels
found to be of the right color
                        pixel_matches = 0;
            end
```

```
            case(state)

                  IDLE: begin
                        //not busy when idle
                        busy = 0;
                        //not data to check for matching yet
                        pixel_matches = 0;

                        //begin checking particle
                        if(start) begin
                        //ask for data for (x,y)
                              //we use the external signals this time because
we know they are correct. they are also being registered at this point
                              col_count = x_external;
                              row_count = y_external;
                              next = ADD1;
                        end else begin
                              //just stay here
                              next = IDLE;
                        end
                  end

                  ADD1: begin
                        //not data to check for matching yet
                        pixel_matches = 0;
                        //ask for data for (x+1,y)
                        col_count = x+1;
                        row_count = y;
                        next = ADD2;
                  end

                  ADD2: begin
                        //not data to check for matching yet
                        pixel_matches = 0;
                        //ask for data for (x+2,y)
                        col_count = x+2;
                        row_count = y;
                        next = ADD3;
                  end

                  ADD3: begin
                        pixel_matches = 0;
                        //ask for data for (x,y+1)
                        col_count = x;
                        row_count = y+1;
                        next = ADD4;
                  end

                  ADD4: begin
                        pixel_matches = 0;
                        //ask for data for (x+1,y+1)
                        col_count = x+1;
                        row_count = y+1;
                        next = ADD5;
                  end
```

```
                ADD5: begin
                      pixel_matches = 0;
                      //ask for data for (x+2,y+1)
                      col_count = x+2;
                      row_count = y+1;
                      next = ADD6;
                end

                ADD6: begin
                      pixel_matches = 0;
                      //ask for data for (x,y+2)
                      col_count = x;
                      row_count = y+2;
                      next = ADD7;

                end

                ADD7: begin
                      pixel_matches = 0;
                      //ask for data for (x+1,y+2)
                      col_count = x+1;
                      row_count = y+2;
                      next = ADD8;
                end

                ADD8: begin
                      pixel_matches = 0;
                      //ask for data for (x+2,y+2)
                      col_count = x+2;
                      row_count = y+2;
                      clear_wait_counter = 1;
                      next = WAIT;
                end

                WAIT: begin
                      pixel_matches = 0;
                      //await for a bit because of divider latency in
getting hsv data to this module
                      if(wait_counter == 16 ) begin
                            clear_wait_counter = 1;
                            next = DATA;
                      end else begin
                            next = WAIT;
                      end
                end

                DATA: begin
                      //just waiting for data to come in and be processed
                      if(wait_counter ==  8) begin
                            next = RESULTS;
                      end else begin
                            next = DATA;
                      end
                end

                //send out result of calculations, whether particle is
still alive or not
```

```
                  RESULTS: begin
                        busy = 0;
                        //not data to check for matching yet
                        pixel_matches = 0;
                        //if count > threshold, the particle is still over
the right color and it is alive (true). else it is dead (false)
                        result = (count > THRESHOLD) ;
                        next = IDLE;
                  end


            endcase

      end

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
/////
// Company:
// Engineer: Lyric Doshi
//
// Create Date:    00:36:49 05/09/2007
// Design Name:
// Module Name:    prng_23
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/*
This module uses a 23-bit linear feedback shift register (lfsr) to generate
pseudo random numbers from -7 to 8 by taking 4 bits from the lfsr at a time.
the output random is the current random number. Setting input next high for a
cycle produces the next random number 5 cycles later. using the value
intermediately will give some number underconstruction. after next, the
random output is not held constant.

*/
//////////////////////////////////////////////////////////////////////////////////
/////
module prng_23(clock, reset, next, random);
       input clock;
    input reset;
    input next;
    output [10:0] random;

      parameter signed [10:0] base = -7;
      reg signed [10:0] bits;
      wire next_bit;

      reg [2:0] state;

      parameter ONE = 3'd0;
      parameter TWO = 3'd1;
      parameter THREE = 3'd2;
      parameter FOUR = 3'd3;
      parameter READY = 3'd4;
      //linear feedback shift register to be random bits from
      lfsr_23 lfsr1(.clock(clock), .reset(reset),.next_bit(next_bit));

      always @ (posedge clock) begin
            if(reset) begin
                  state <= ONE;
                  bits <= 0;
            end else begin
                  case(state)
```

```
                        //obtain first bit
                        ONE: begin
                              bits[0] <= next_bit;
                              state <= TWO;
                        end
                        //obtain second bit
                        TWO: begin
                              bits[1] <= next_bit;
                              state <= THREE;
                        end
                        //obtain third bit
                        THREE: begin
                              bits[2] <= next_bit;
                              state <= FOUR;
                        end

                        //obtain fourth bit
                        FOUR: begin
                                    bits[3] <= next_bit;
                                    state <= READY;
                        end
                        //hold 4-bit number as output until next signal, then
generate the next one
                        READY: begin
                              if(next) begin
                                    state <= ONE;

                              end else begin
                                    state <= READY;
                              end
                        end

                  endcase
            end
      end

      //generates
      assign random = base + bits;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//////
// Company:
// Engineer: Lyric Doshi
//
// Create Date:    00:36:49 05/09/2007
// Design Name:
// Module Name:    prng_25
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/*
This module uses a 25-bit linear feedback shift register (lfsr) to generate
pseudo random numbers from -7 to 8 by taking 4 bits from the lfsr at a time.
the output random is the current random number. Setting input next high for a
cycle produces the next random number 5 cycles later. using the value
intermediately will give some number underconstruction. after next, the
random output is not held constant.

*/
//////////////////////////////////////////////////////////////////////////////////
//////
module prng_25(clock, reset, next, random);
       input clock;
    input reset;
    input next;
    output [10:0] random;

      parameter signed [10:0] base = -7;
      reg signed [10:0] bits;
      wire next_bit;

      reg [2:0] state;

      parameter ONE = 3'd0;
      parameter TWO = 3'd1;
      parameter THREE = 3'd2;
      parameter FOUR = 3'd3;
      parameter READY = 3'd4;
      //linear feedback shift register to be random bits from
      lfsr_25 lfsr1(.clock(clock), .reset(reset),.next_bit(next_bit));

      always @ (posedge clock) begin
            if(reset) begin
                  state <= ONE;
                  bits <= 0;
            end else begin
                  case(state)
```

```
                        //obtain first bit
                        ONE: begin
                              bits[0] <= next_bit;
                              state <= TWO;
                        end
                        //obtain second bit
                        TWO: begin
                              bits[1] <= next_bit;
                              state <= THREE;
                        end
                        //obtain third bit
                        THREE: begin
                              bits[2] <= next_bit;
                              state <= FOUR;
                        end

                        //obtain fourth bit
                        FOUR: begin
                                    bits[3] <= next_bit;
                                    state <= READY;
                        end
                        //hold 4-bit number as output until next signal, then
generate the next one
                        READY: begin
                              if(next) begin
                                    state <= ONE;

                              end else begin
                                    state <= READY;
                              end
                        end

                  endcase
            end
      end

      //generates
      assign random = base + bits;

endmodule
```

```
//CODE BORROWED FROM THE 6.111 Fall 2005 website


// 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 phase takes several miliseconds)
```

```
  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 CLK hold
   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 of Transmission
                           .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, count);
  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
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
/////
// Company:
// Engineer: Rob Crowell
//
// Create Date:    16:13:01 05/11/2007
// Design Name:
// Module Name:    rgb2hsv
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
/////
module rgb2hsv(clock, reset, r, g, b, h, s, v);
      input wire clock;
      input wire reset;
      input wire [7:0] r;
      input wire [7:0] g;
      input wire [7:0] b;
      output reg [7:0] h;
      output reg [7:0] s;
      output reg [7:0] v;

      reg [7:0] my_r_delay1, my_g_delay1, my_b_delay1;
      reg [7:0] my_r_delay2, my_g_delay2, my_b_delay2;
      reg [7:0] my_r, my_g, my_b;
      reg [7:0] min, max, delta;
      reg [15:0] s_top;
      reg [15:0] s_bottom;
      reg [15:0] h_top;
      reg [15:0] h_bottom;

      wire [15:0] s_quotient;
      wire [15:0] s_remainder;
      wire s_rfd;
      wire [15:0] h_quotient;
      wire [15:0] h_remainder;
      wire h_rfd;

      reg [7:0] v_delay [19:0];
      reg [18:0] h_negative;
      reg [15:0] h_add [18:0];

      reg [4:0] i;

            // Clocks 4-18: perform all the divisions
            //the s_divider (16/16) has delay 18
            //the hue_div (16/16) has delay 18
```

```
hue_div hue_div1(
            .clk(clock),
            .dividend(s_top),
            .divisor(s_bottom),
            .quotient(s_quotient),
            .remainder(s_remainder),
            .rfd(s_rfd)
);

hue_div hue_div2(
            .clk(clock),
            .dividend(h_top),
            .divisor(h_bottom),
            .quotient(h_quotient),
            .remainder(h_remainder),
            .rfd(h_rfd)
);

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

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

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

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

      // Clock 3: compute the delta
```

```
            {my_r_delay2, my_g_delay2, my_b_delay2} <= {my_r_delay1,
my_g_delay1, my_b_delay1};
            v_delay[0] <= max;
            delta <= max - min;

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

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

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

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

            if(delta > 0) begin
                  h_bottom <= delta * 8'd6;
            end else begin
                  h_bottom <= 16'd6;
            end

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

```
            v_delay[19] <= v_delay[18];

            //Clock 22: compute the final value of h

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

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

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//////
// Company:
// Engineer: Lyric Doshi
//
// Create Date:    15:04:18 04/24/2007
// Design Name:
// Module Name:    servo_feedback
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//  This module takes the corrent x position of the target and checks if it
is centered in the camera image [within some range of the center line
considered to be the center of the image]. if the x is to the right of the
center, the module commands the servo to turn clockwise. if the x is to the
left of the center, the module commands the servo to turn counter_clockwise.
if x is in the center region, the module tells the servo not to move.
//////////////////////////////////////////////////////////////////////////////////
//////
module servo_feedback(clock, x, clockwise, counter_clockwise);
    input clock;
    input [9:0] x;
    output reg clockwise;
    output reg counter_clockwise;

      //FIX THESE TO APT VALUES
      //ranges that define the center of the image in x and y directions
      parameter x_lower_buffer = 270;
      parameter x_higher_buffer = 370;
      parameter y_lower_buffer = 4;
      parameter y_higher_buffer = 6;

      always @ (posedge clock) begin

            if( x > x_higher_buffer) begin
                  clockwise = 1; //turn the camera clockwise to center x if
x is to the right of the center
                  counter_clockwise = 0;
            end else if (x < x_lower_buffer) begin
                  counter_clockwise = 1; //turn the camera counterclockwise
to center x is to the left of the center
                  clockwise = 0;
            end else begin
                  counter_clockwise = 0; //don't turn the servo, the target
is sufficiently centered
                  clockwise = 0;
            end
      end
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
/////
// Company:
// Engineer: Lyric Doshi
//
// Create Date:    17:09:16 04/23/2007
// Design Name:
// Module Name:    servo_pwm
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
// This module generates the PWM signal to send to the servo. It takes an
input of clockwise or counterclockwise. If both are high or low, the servo is
not moved. Otherwise, it rotates the servo clockwise or counterclockwise
depending on which one is high. It does so by incrementally decreasing or
increasing the pulse width respectively to rotate the servo. If the servo is
rotated to 0 degrees, it cannot be turned further clockwise and it ignores
the input. If the servo is rotated to 180 degrees, it cannot be turned
further counterclockwise and ignores the input. The speed of the rotation can
be changed by changing ROTATION_PERIOD. The PERIOD variable is set to make
the pulse period every 20 ms, as required by the servo. reset places the
servo at 90 degrees.
//
//////////////////////////////////////////////////////////////////////////////////
/////
module servo_pwm(clock, reset, clockwise, counter_clockwise, pulse,
hasValidTarget);
    input clockwise;
       input counter_clockwise;
       input clock;
       input reset;
       input hasValidTarget;
    output pulse;

      reg [19:0] count;
      reg [19:0] pulse_time;
      reg [19:0] next_pulse_time;
      reg [13:0] rotation_count;

      //PERIOD is a count high enough to get to 20 ms with the 31.5 mhz clock
      parameter PERIOD = 20'd629999;

      //parameter ZERO_DEGREES = 15'd17584;
      parameter ZERO_DEGREES = 20'd18000;
      parameter NINETY_DEGREES = 20'd45500;
      parameter HUNDRED_EIGHTY_DEGREES = 20'd73000;
      parameter ROTATION_PERIOD = 14'd15000;

      always @ (posedge clock) begin
```

```
            if(reset) begin
                  pulse_time <= NINETY_DEGREES;
            end else begin
                  //update pulse time with the next value
                  pulse_time <= next_pulse_time;
            end

            //count from 0 to PERIOD, which generates the 20 ms frequency of
the Servo PWM
            if(count == PERIOD) begin
                  count <= 0;
            end else begin
                  count <= count + 1;
            end

            //count rotation_count from 0 to ROTATION_PERIOD, which controls
how often the servo pulse_time is updated
            if(rotation_count == ROTATION_PERIOD) begin
                  rotation_count <= 0;
            end else begin
                  rotation_count <= rotation_count + 1;
            end


      end

      always @ (clockwise or counter_clockwise) begin
            if(rotation_count == ROTATION_PERIOD && hasValidTarget) begin
                  //make sure only one command is asserted
                  if(clockwise && !counter_clockwise) begin
                        //make sure we can still turn more in the clockwise
direction
                        if(pulse_time > ZERO_DEGREES) begin
                              //if so, decrement the pulse_time by one
                              next_pulse_time = pulse_time - 1;
                        end else begin
                              //dont change pulse_time, because can't be any
lower
                              next_pulse_time = pulse_time;
                        end
                  end else if (!clockwise && counter_clockwise) begin
                        //make sure we can still turn more in the counter
clockwise direction
                        if(pulse_time < HUNDRED_EIGHTY_DEGREES) begin
                              //if so, increment pulse time by 1
                              next_pulse_time = pulse_time + 1;
                        end else begin
                              //dont change pulse_time, because can't be any
higher
                              next_pulse_time = pulse_time;
                        end
                  end else begin
                        next_pulse_time = pulse_time;
                  end
            end else begin
                  next_pulse_time = pulse_time;
            end
```

```
      end

      //this code allows the pulse width to be controlled using pulse_time so
that the pulse is low for exactly pulse_time counts of the 31.5 mhz clock
      assign pulse = (count < pulse_time) ? 0 : 1;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//////
// Company:
// Engineer: Lyric Doshi
//
// Create Date:    15:17:23 05/01/2007
// Design Name:
// Module Name:    vga_display_rectangle
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
//////
module vga_display_rectangle(x, y, contains);
    input signed [10:0] x;
    input signed [10:0] y;
    output contains;

      //boundaries of rectangles
      parameter signed LEFT = 11'd0;
      parameter signed RIGHT = 11'd639;
      parameter signed TOP = 11'd0;
      parameter signed BOTTOM = 11'd479;

      assign contains = ~( (x < LEFT) || (x > RIGHT) || (y < TOP ) || (y >
BOTTOM) );

endmodule
```

```
`timescale 1ns / 1ps
`default_nettype none

//////////////////////////////////////////////////////////////////////////////////
//////
// Company:
// Engineer: Rob Crowell
//
// Create Date:    12:27:39 05/04/2007
// Design Name:
// Module Name:    zbt_controller
// Project Name:
// Target Devices:
// Tool versions:
// Description: Latches in the data and address, sends it to ram, waits, and
returns.  The delays
//              are due to the pipelined ZBT RAM.  If clock is twice as fast
as the system clock,
//              then a one-cycle delay is present and the ram simulates a
dual-port ram.
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
//////
module zbt_controller(clock, reset, we_in, address_in, data_in, we_out,
address_out, data_out, ram_address, ram_data, ram_cen_b, ram_ce_b, ram_oe_b,
ram_we_b, ram_bwe_b, ram_adv_ld);
      input wire clock;
      input wire reset;
      input wire we_in;
      input wire [18:0] address_in;
      input wire [35:0] data_in;
      output reg we_out;
      output reg [18:0] address_out;
      output wire [35:0] data_out;

      output wire [18:0] ram_address;
      inout wire [35:0] ram_data;
      output reg ram_cen_b;
      output reg ram_ce_b;
      output reg ram_oe_b;
      output wire ram_we_b;
      output reg [3:0] ram_bwe_b;
      output reg ram_adv_ld;

      // On the neg edge of each clock cycle, dump new values to the RAM
      reg [18:0] address_delay;
      reg we_delay;
      reg [35:0] data_delay, write_data;
      always @ (posedge clock) begin
            if(reset) begin
                  we_out <= 0;
```

```
                address_out <= 0;

                ram_cen_b <= 0;
                ram_ce_b <= 0;
                ram_oe_b <= 0;
                ram_adv_ld <= 0;
                ram_bwe_b <= 4'd0;

                address_delay <= 0;
                we_delay <= 0;
                data_delay <= 0;
                write_data <= 0;

          end else begin
                address_delay <= address_in;
                address_out <= address_delay;

                we_delay <= we_in;
                we_out <= we_delay;

                data_delay <= data_in;
                write_data <= data_delay;

          end
    end

    // Send the address and we out immediately
    assign ram_address = address_in;
    assign ram_we_b = ~we_in;

    // Tristate ram_data
    assign ram_data = we_out ? write_data : 36'hZ;
    assign data_out = ram_data;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
/////
// Company:
// Engineer: Rob Crowell
//
// Create Date:    16:25:57 05/08/2007
// Design Name:
// Module Name:    zbt_dual_port
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
/////
module zbt_dual_port(clock_2x, reset, we_in1, address_in1, data_in1, we_in2,
address_in2, data_in2, we_out1, address_out1, data_out1, we_out2,
address_out2, data_out2,
                                          ram_address, ram_data, ram_ce_b,
ram_cen_b, ram_oe_b, ram_we_b, ram_bwe_b, ram_adv_ld);
      input wire clock_2x;
      input wire reset;
      input wire we_in1;
      input wire [18:0] address_in1;
      input wire [35:0] data_in1;
      input wire we_in2;
      input wire [18:0] address_in2;
      input wire [35:0] data_in2;

      output reg we_out1;
      output reg [18:0] address_out1;
      output reg [35:0] data_out1;
      output wire we_out2;
      output wire [18:0] address_out2;
      output wire [35:0] data_out2;

      output wire [18:0] ram_address;
      inout wire [35:0] ram_data;
      output wire ram_ce_b;
      output wire ram_cen_b;
      output wire ram_oe_b;
      output wire ram_we_b;
      output wire [3:0] ram_bwe_b;
      output wire ram_adv_ld;

      // Create a ZBT controller
      reg we_in;
      reg [18:0] address_in;
      reg [35:0] data_in;
      zbt_controller controller(
```

```
	.clock(clock_2x),
	.reset(reset),
	.we_in(we_in),
	.address_in(address_in),
	.data_in(data_in),
	.we_out(we_out2),
	.address_out(address_out2),
	.data_out(data_out2),
	.ram_address(ram_address),
	.ram_data(ram_data),
	.ram_cen_b(ram_cen_b),
	.ram_ce_b(ram_ce_b),
	.ram_oe_b(ram_oe_b),
	.ram_we_b(ram_we_b),
	.ram_bwe_b(ram_bwe_b),
	.ram_adv_ld(ram_adv_ld)
);

// Register in all values specified on the posedge of the slow clock
reg [18:0] my_address_in2;
reg [35:0] my_data_in2;
reg my_we_in2;
reg do_ram;
always @ (posedge clock_2x) begin
	if(reset) begin
		do_ram <= 0;

		address_in <= 0;
		data_in <= 0;
		we_in <= 0;

		my_address_in2 <= 0;
		my_data_in2 <= 0;
		my_we_in2 <= 0;

		address_out1 <= 0;
		data_out1 <= 0;
		we_out1 <= 0;

	end else begin
		if(do_ram == 1) begin
			do_ram <= 0;

			address_in <= address_in1;
			data_in <= data_in1;
			we_in <= we_in1;

			my_address_in2 <= address_in2;
			my_data_in2 <= data_in2;
			my_we_in2 <= we_in2;

		end else begin
			do_ram <= 1;

			address_in <= my_address_in2;
			data_in <= my_data_in2;
			we_in <= my_we_in2;
```

```
                        address_out1 <= address_out2;
                        data_out1 <= data_out2;
                        we_out1 <= we_out2;
                    end
                end
            end

endmodule
```