```
/*****************************************
 *  SOURCE FILES                         *
 *****************************************/

/*** labkit.v only connects the rightmost display modules ***/

///////////////////////////////////////////////////////////////////////////////
//
// 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 labkit (beep, audio_reset_b, ac97_sdata_out, ac97_sdata_in, ac97_synch,
         ac97_bit_clock,

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

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

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

         ram0_data, ram0_address, ram0_adv_ld, ram0_clk, ram0_cen_b,
         ram0_ce_b, ram0_oe_b, ram0_we_b, ram0_bwe_b,
	   ram1_data, ram1_address, ram1_adv_ld, ram1_clk, ram1_cen_b,
	   ram1_ce_b, ram1_oe_b, ram1_we_b, ram1_bwe_b,

	   clock_feedback_out, clock_feedback_in,

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

	   rs232_txd, rs232_rxd, rs232_rts, rs232_cts,

	   mouse_clock, mouse_data, keyboard_clock, keyboard_data,

	   clock_27mhz, clock1, clock2,

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

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

	   switch,

	   led,

	   user1, user2, user3, user4,

	   daughtercard,

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

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

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

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

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

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

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

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

   input  clock_feedback_in;
   output clock_feedback_out;

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

   output rs232_txd, rs232_rts;
   input  rs232_rxd, rs232_cts;

   input  mouse_clock, mouse_data, keyboard_clock, keyboard_data;

   input  clock_27mhz, clock1, clock2;

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

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

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

```
    inout [43:0] daughtercard;

    inout  [15:0] systemace_data;
    output [6:0]  systemace_address;
    output systemace_ce_b, systemace_we_b, systemace_oe_b;
    input  systemace_irq, systemace_mpbrdy;

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

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

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

//  // VGA Output
//  assign vga_out_red = 8'h0;
//  assign vga_out_green = 8'h0;
//  assign vga_out_blue = 8'h0;
//  assign vga_out_sync_b = 1'b1;
//  assign vga_out_blank_b = 1'b1;
//
//  // half-cycle delay
//  assign vga_out_pixel_clock = ~pixel_clock;
//  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;
   assign user4 = 32'hZ;

   // Daughtercard Connectors
   assign daughtercard = 44'hZ;

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

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


   // **FINAL PROJECT ASSIGNMENTS**

   wire pclk, pixel_clock;

   wire reset_sync;

   wire locked;

   wire disp_start;
   wire [511:0] disp_data_r;
   wire [8:0] disp_addr;

   wire test_start;
   wire [511:0] test_data_w;
   wire [8:0] test_addr;

   // memory variables
   wire [8:0] addr0, addr1;
   wire [511:0] data0in, data1in;
   wire [511:0] data0out, data1out;
   wire we0, we1;

   // display variables
   wire newframe;
   wire hsync, vsync;
   wire [10:0] screen_x, screen_y;
   wire [23:0] rgb;

   // variables for testing
   wire hold;
   wire [2:0] state;
   wire ramsel;
   wire [1:0] w_state, r_state;

//  wire [19:0] dot_sel_addr;
//  wire [31:0] dot_sel_data;
//  wire [39:0] dots_addr_18_16, dots_addr_15_12,
//      dots_addr_11_8, dots_addr_7_4,
//      dots_addr_3_0;
//  wire [39:0] dots_data_35_32, dots_data_31_28,
//      dots_data_27_24, dots_data_23_20,
//      dots_data_19_16, dots_data_15_12,
//      dots_data_11_8, dots_data_7_4;



  // testing
//  assign analyzer1_data[15] = reset_sync;
//  assign analyzer1_data[14:0] = ram0_address[14:0];
//  assign analyzer1_data[10:0] = screen_x;
//  assign analyzer1_data[11] = ram0_we_b;
//  assign analyzer1_data[12] = ram1_we_b;
//  assign analyzer1_data[13] = vga_out_green[7];
//  assign analyzer1_data[14] = ram0_data[5];
//  assign analyzer1_data[15] = ram1_data[5];
//  assign analyzer1_clock = pixel_clock;
//
//  assign analyzer3_data[15] = newframe;
//  assign analyzer3_data[14:0] = ram1_address[14:0];
//  assign analyzer3_clock = 1'b1;
//

  assign analyzer1_clock = pixel_clock;
  assign analyzer1_data[15:7] = disp_addr[8:0];
  assign analyzer1_data[6] = data0out[0];
  assign analyzer1_data[5] = data1out[0];
  assign analyzer1_data[4:1] = disp_data_r[4:0];
  assign analyzer1_data[0] = ramsel;

  assign analyzer3_data[15:13] = state;
  assign analyzer3_data[12:11] = w_state;
  assign analyzer3_data[10:9] = r_state;
  assign analyzer3_data[8] = we0;
  assign analyzer3_data[7] = we1;
  assign analyzer3_data[6] = disp_start;
  assign analyzer3_data[5] = test_start;
  assign analyzer3_data[4:0] = 1'b0;

  assign analyzer3_clock = 1'b1;

  assign analyzer2_data = 16'b0;
  assign analyzer2_clock = 1'b0;
  assign analyzer4_data = 16'b0;
  assign analyzer4_clock = 1'b0;

  assign hold = switch[6];


  // display
  // generate 64.8 MHz clock - code stolen from lab4_labkit
  DCM vclk1(.CLKIN(clock_27mhz),.CLKFX(pclk));
  // synthesis attribute CLKFX_DIVIDE of vclk1 is 10
  // synthesis attribute CLKFX_MULTIPLY of vclk1 is 24
  // synthesis attribute CLK_FEEDBACK of vclk1 is NONE
  // synthesis attribute CLKIN_PERIOD of vclk1 is 37
  BUFG vclk2(.O(pixel_clock),.I(pclk));


  // VGA Output
//  assign vga_out_red = 8'h0;
//  assign vga_out_green = 8'h0;
//  assign vga_out_blue = 8'h0;
  assign vga_out_sync_b = 1'b1;
//  assign vga_out_blank_b = 1'b1;

  // half-cycle delay
  assign vga_out_pixel_clock = ~pixel_clock;
//  assign vga_out_hsync = 1'b0;
//  assign vga_out_vsync = 1'b0;

  assign {vga_out_red, vga_out_green, vga_out_blue} = rgb;



  // modules
  debounce reset_button(0, pixel_clock, ~button0, reset_sync);

  display_buffer buf0(pixel_clock, data0in, addr0, we0, data0out);
  display_buffer buf1(pixel_clock, data1in, addr1, we1, data1out);

  control timing_control(pixel_clock, reset_sync, screen_x, screen_y,
    newframe);

  bufctrl buffer_control(hold, pixel_clock, reset_sync, newframe,
    test_start, test_addr, test_data_w,
    disp_start, disp_addr, disp_data_r,
    addr0, data0in, data0out, we0, addr1, data1in, data1out, we1,
    // debugging outputs
    ramsel, w_state, r_state);


  test_display test_read(hold, pixel_clock, reset_sync, newframe,
    test_start, test_addr, test_data_w);


  display buffer_display(hold, pixel_clock, reset_sync, screen_x, screen_y, newframe,
    disp_start, disp_addr, disp_data_r, rgb,
    // debugging
    state);

  vga vga_timing(hold, pixel_clock, reset_sync, screen_x, screen_y, hsync, vsync, blank);

  delay hsync_delay(pixel_clock, reset_sync, hsync, vga_out_hsync);
  delay vsync_delay(pixel_clock, reset_sync, vsync, vga_out_vsync);
  delay blank_delay(pixel_clock, reset_sync, blank, vga_out_blank_b);

endmodule



// control.v
// Controls flow of data through all modules
`timescale 1ns / 1ps

module control(clk, system_rst, sc_x, sc_y, newframe);
  input clk;
  input system_rst;
  output module_rst;
  input done_initial, done_transform, done_project, done_linedraw_params, done_linedraw;

  input [10:0] sc_x;
  input [10:0] sc_y;
//  output start;
  output newframe;

  reg start;
  reg newframe;

  reg module_rst;

  reg mode;   // 0 while initializing, 1 while normal running

  always @ (posedge clk)  begin
    if (system_rst)   begin
      mode <= 0;
    end

    else if (!mode && done_initial)   begin
      module_rst <= 1;
      mode <= 1;
    end

    else if (mode)  begin
      module_rst <= 0;
      newframe <= (sc_x == 11'd1342 && sc_y == 11'd805);

      // reset start if high
      if (start)  begin
        start <= 0;
      end

      // when all modules are done, safe to issue another new 'start'
      else if (done_transform && done_project &&
          done_linedraw_params && done_linedraw)  begin
        start <= 1;
      end
    end
  end

endmodule


// transform.v
// Geometric transformations
`timescale 1ns / 1ps

module transform(clk, reset, start, done, newframe, eof, addr_r, addr_w, we,
    data, p1out, p2out, dirs, zoom, fnselect);
  input clk;
  input reset;
  input start;
  output done;
  input newframe;
  output eof;
  output [8:0] addr_r, addr_w;
  output we;
  input [95:0] data;
  output [47:0] p1out, p2out;
```

```
input [3:0] dirs;
input [1:0] zoom;
input fnselect;

reg [8:0] addr_r, addr_w;
reg we, eof;
reg done;

reg signed [15:0] xin1, yin1, zin1;
reg signed [15:0] xin2, yin2, zin2;

reg signed [31:0] p1x_1_prod, p1y_1_prod, p1z_1_prod;
reg signed [31:0] p1x_2_prod, p1y_2_prod, p1z_2_prod;
reg signed [31:0] p2x_1_prod, p2y_1_prod, p2z_1_prod;
reg signed [31:0] p2x_2_prod, p2y_2_prod, p2z_2_prod;

wire signed [15:0] p1x_1, p1y_1, p1z_1;
wire signed [15:0] p1x_2, p1y_2, p1z_2;
wire signed [15:0] p2x_1, p2y_1, p2z_1;
wire signed [15:0] p2x_2, p2y_2, p2z_2;

reg [2:0] state, next;
reg [47:0] p1out, p2out;

parameter signed ONE = 16'b1000_0000_0000;
parameter signed SIN_THETA = 16'b000_0000_0010;
parameter signed COS_THETA = 16'b111_1111_1111;
parameter signed DELTA_POSITION = 16'b1000_0000;
parameter signed ZOOM_POSITION = 16'b1_0000_0000;

parameter IDLE = 3'd0;
parameter START = 3'd1;
parameter MULT_DONE = 3'd6;
parameter DONE = 3'd7;

assign p1x_1 = {p1x_1_prod[31], p1x_1_prod[25:11]};
assign p1y_1 = {p1y_1_prod[31], p1y_1_prod[25:11]};
assign p1z_1 = {p1z_1_prod[31], p1z_1_prod[25:11]};
assign p1x_2 = {p1x_2_prod[31], p1x_2_prod[25:11]};
assign p1y_2 = {p1y_2_prod[31], p1y_2_prod[25:11]};
assign p1z_2 = {p1z_2_prod[31], p1z_2_prod[25:11]};
assign p2x_1 = {p2x_1_prod[31], p2x_1_prod[25:11]};
assign p2y_1 = {p2y_1_prod[31], p2y_1_prod[25:11]};
assign p2z_1 = {p2z_1_prod[31], p2z_1_prod[25:11]};
assign p2x_2 = {p2x_2_prod[31], p2x_2_prod[25:11]};
assign p2y_2 = {p2y_2_prod[31], p2y_2_prod[25:11]};
assign p2z_2 = {p2z_2_prod[31], p2z_2_prod[25:11]};

always @ (posedge clk)  begin
  if (reset | newframe) begin
    addr_r <= 0;
    we <= 0;
    eof <= 0;
    done <= 0;
  end

  else if (start) begin
    addr_r <= addr_r + 1;
    done <= 0;

    // write ONCE back to BRAM
    addr_w <= addr_r;
    we <= 1;

    // if end of data
    if (data == 96'hFFFF_FFFF_FFFF_FFFF_FFFF_FFFF)  begin
      eof <= 1;
      state <= IDLE;
    end
    // else, latch new data
    else  begin
      {xin1, yin1, zin1} <= {data[95:80], data[79:64], data[63:48]};
      {xin2, yin2, zin2} <= {data[47:32], data[31:16], data[15:0]};

      state <= START;
      eof <= 0;
    end
  end

  else if (state == MULT_DONE)   begin
    p1out <= {p1x_1_prod + p1x_2_prod, p1y_1_prod + p1y_2_prod, p1z_1_prod + p1z_2_prod};
    p2out <= {p2x_1_prod + p2x_2_prod, p2y_1_prod + p2y_2_prod, p2z_1_prod + p2z_2_prod};
  end

  else if (state == START)  begin
    we <= 0;

    if (zoom != 2'b00)  begin
      state <= DONE;
      // zoom out
      if (zoom[0])  begin
        p1out <= {xin1, yin1, zin1 - ZOOM_POSITION};
        p2out <= {xin2, yin2, zin2 - ZOOM_POSITION};
      end
      // zoom in
      else if (zoom[1]) begin
        p1out <= {xin1, yin1, zin1 + ZOOM_POSITION};
        p2out <= {xin2, yin2, zin2 + ZOOM_POSITION};
      end
    end

    // rotating
    else if (fnselect)  begin
      // if viewer rotates up, the object rotates down relative to viewer
      // rotate up
      if (dirs[0])  begin
        state <= next;

        p1x_1_prod <= xin1 * ONE;          p1x_2_prod <= 31'b0;
        p1y_1_prod <= yin1 * COS_THETA;    p1y_2_prod <= yin1 * SIN_THETA;
        p1z_1_prod <= yin1 * SIN_THETA;    p1z_2_prod <= zin1 * COS_THETA;

        p2x_1_prod <= xin2 * ONE;          p2x_2_prod <= 31'b0;
        p2y_1_prod <= yin2 * COS_THETA;    p2y_2_prod <= yin2 * SIN_THETA;
        p2z_1_prod <= yin2 * SIN_THETA;    p2z_2_prod <= zin2 * COS_THETA;
      end
      // rotate down
      else if (dirs[1]) begin
        state <= next;

        p1x_1_prod <= xin1 * ONE;          p2x_2_prod <= 31'b0;
        p1y_1_prod <= yin1 * COS_THETA;    p2y_2_prod <= yin1 * -SIN_THETA;
        p1z_1_prod <= yin1 * -SIN_THETA;   p2z_2_prod <= zin1 * COS_THETA;

        p2x_1_prod <= xin2 * ONE;          p2x_2_prod <= 31'b0;
        p2y_1_prod <= yin2 * COS_THETA;    p2y_2_prod <= yin2 * -SIN_THETA;
        p2z_1_prod <= yin2 * -SIN_THETA;   p2z_2_prod <= zin2 * COS_THETA;
      end
      // rotate left
      else if (dirs[2]) begin
        state <= next;

        p1x_1_prod <= xin1 * COS_THETA;    p1x_2_prod <= zin1 * SIN_THETA;
        p1y_1_prod <= yin1 * ONE;          p1y_2_prod <= 31'b0;
        p1z_1_prod <= xin1 * SIN_THETA;    p1z_2_prod <= zin1 * COS_THETA;

        p2x_1_prod <= xin2 * COS_THETA;    p2x_2_prod <= zin2 * SIN_THETA;
        p2y_1_prod <= yin2 * ONE;          p2y_2_prod <= 31'b0;
        p2z_1_prod <= xin2 * SIN_THETA;    p2z_2_prod <= zin2 * COS_THETA;
      end
      // rotate right
      else if (dirs[3]) begin
        state <= next;

        p1x_1_prod <= xin1 * COS_THETA;    p1x_2_prod <= zin1 * -SIN_THETA;
        p1y_1_prod <= yin1 * ONE;          p1y_2_prod <= 31'b0;
        p1z_1_prod <= xin1 * -SIN_THETA;   p1z_2_prod <= zin1 * COS_THETA;

        p2x_1_prod <= xin2 * COS_THETA;    p2x_2_prod <= zin2 * -SIN_THETA;
        p2y_1_prod <= yin2 * ONE;          p2y_2_prod <= 31'b0;
        p2z_1_prod <= xin2 * -SIN_THETA;   p2z_2_prod <= zin2 * COS_THETA;
      end

      else  begin
        state <= DONE;
        p1out <= {xin1, yin1, zin1};
        p2out <= {xin2, yin2, zin2};
      end
    end

    // panning
    else  begin
      state <= DONE;

      // pan up
      if (dirs[0])  begin
        p1out <= {xin1, yin1 - DELTA_POSITION, zin1};
        p2out <= {xin2, yin2 - DELTA_POSITION, zin2};
      end
      // pan down
      else if (dirs[1]) begin
        p1out <= {xin1, yin1 + DELTA_POSITION, zin1};
        p2out <= {xin2, yin2 + DELTA_POSITION, zin2};
      end
      // pan left
      else if (dirs[2]) begin
        p1out <= {xin1 + DELTA_POSITION, yin1, zin1};
        p2out <= {xin2 + DELTA_POSITION, yin2, zin2};
      end
      // pan right
```

```
        else if (dirs[3]) begin
          p1out <= {xin1 - DELTA_POSITION, yin1, zin1};
          p2out <= {xin2 - DELTA_POSITION, yin2, zin2};
        end

        else  begin
          p1out <= {xin1, yin1, zin1};
          p2out <= {xin2, yin2, zin2};
        end
      end
    end

    else  begin
      state <= next;
    end
  end

  always @ (state)  begin
    next = state + 1;
    case (state)
      IDLE:   next = IDLE;
      DONE:   begin
        next = IDLE;
        done = 1;
      end
    endcase
  end

endmodule


// transform_reg.v
// Holds value of button pushes and applies them next frame

`timescale 1ns / 1ps

module transform_reg(clk, reset, newframe, in, out);
  input clk;
  input reset;
  input newframe;
  input in;
  output out;

  reg pushed;

  always @ (posedge clk)  begin
    if (reset)  begin
      pushed <= 0;
      out <= 0;
    end

    else if (newframe)  begin
      pushed <= 0;
      out <= pushed;
    end

    else  begin
      pushed <= in ? 1 : pushed;
    end
  end

endmodule


// project.v
// Projects a line segment (specified by p1in, p2in) onto the plane
// z = screen, with viewer at (x_eye, y_eye, z_eye)


`timescale 1ns / 1ps

module project(clk, reset, start, done, x_eye, y_eye, z_eye, screen, p1in, p2in, p1out, p2out,
p1out_x, p1out_y, p2out_x, p2out_y,
// the following are debugging outputs
z1prop, z2prop,
x1disp, y1disp, x2disp, y2disp,
z1num, z1denom, z2num, z2denom,
x1diff, y1diff, x2diff, y2diff,
state);
  input clk;
  input reset;
  input start;
  output done;
  input signed [15:0] x_eye;
  input signed [15:0] y_eye;
  input signed [15:0] z_eye;
  input signed [15:0] screen;
  input [47:0] p1in;
  input [47:0] p2in;
  output [31:0] p1out;
  output [31:0] p2out;

// following unindented lines are for debugging
output signed [15:0] p1out_x, p1out_y;
output signed [15:0] p2out_x, p2out_y;
output signed [15:0] z1prop, z2prop;
output signed [15:0] x1disp, y1disp, x2disp, y2disp;
output signed [15:0] z1num, z1denom, z2num, z2denom;
output signed [15:0] x1diff, y1diff, x2diff, y2diff;
output [5:0] state;

wire signed [15:0] p1out_x, p1out_y;
wire signed [15:0] p2out_x, p2out_y;

assign p1out_x = p1out[31:16];
assign p1out_y = p1out[15:0];
assign p2out_x = p1out[31:16];
assign p2out_y = p1out[15:0];

  reg done;

  reg signed [15:0] x1in, y1in, z1in;
  reg signed [15:0] x2in, y2in, z2in;

  reg signed [15:0] z1num;
  reg signed [15:0] z2num;
  reg signed [15:0] x1diff, y1diff, z1denom;
  reg signed [15:0] x2diff, y2diff, z2denom;

  // temporary registers to hold products (which are twice as many bits as
  // operands)
  reg signed [31:0] x1prod, y1prod;
  reg signed [31:0] x2prod, y2prod;
  wire signed [15:0] x1disp, y1disp;
  wire signed [15:0] x2disp, y2disp;

  // extract appropriate bits while keeping sign bit
  assign x1disp = {x1prod[31], x1prod[25:11]};
  assign y1disp = {y1prod[31], y1prod[25:11]};
  assign x2disp = {x2prod[31], x2prod[25:11]};
  assign y2disp = {y2prod[31], y2prod[25:11]};

  wire signed [15:0] z1prop_shift, z2prop_shift;
  wire signed [11:0] z1frac, z2frac;
  wire signed [15:0] z1prop;
  wire signed [15:0] z2prop;

  reg signed [15:0] x1out, y1out;
  reg signed [15:0] x2out, y2out;

  reg [5:0] state, next;

  parameter IDLE = 6'd0;
  parameter START = 6'd1;
  parameter SUB_DONE = 6'd2;
  parameter DIV_DONE = 6'd35;
  parameter MULT_DONE = 6'd41;

  assign p1out = {x1out, y1out};
  assign p2out = {x2out, y2out};


  division prop1(clk, 1'd0, z1num, z1denom, z1prop_shift, z1frac, rfd1);
  division prop2(clk, 1'd0, z2num, z2denom, z2prop_shift, z2frac, rfd2);
  assign z1prop = {z1prop_shift[15], z1prop_shift[3:0], z1frac[10:0]};
  assign z2prop = {z2prop_shift[15], z2prop_shift[3:0], z2frac[10:0]};

  always @ (posedge clk)  begin
    // latch values when ready
    if (start)  begin
      x1in <= p1in[47:31];
      y1in <= p1in[31:16];
      z1in <= p1in[15:0];

      x2in <= p2in[47:31];
      y2in <= p2in[31:16];
      z2in <= p2in[15:0];

      state <= 6'd1;
      done <= 0;
    end

    // if at least one point is behind screen (i.e., something is to be
    // displayed)
    // if one of points is on or in front of the screen, it will appear "cut
    // off", i.e., the line stops where it hits the screen
    else if (z1in <= screen || z2in <= screen)  begin
      state <= next;

      case (state)
```

```
        START:  begin
          // point 1
          z1num <= (z1in > screen) ? z1in - screen : z_eye - screen;
          z1denom <= (z1in > screen) ? z1in - z2in : z_eye - z1in;
          x1diff <= (z1in > screen) ? x1in - x2in : x_eye - x1in;
          y1diff <= (z1in > screen) ? y1in - y2in : y_eye - y1in;

          // point 2
          z2num <= (z2in > screen) ? z2in - screen : z_eye - screen;
          z2denom <= (z2in > screen) ? z2in - z1in : z_eye - z2in;
          x2diff <= (z2in > screen) ? x2in - x1in : x_eye - x2in;
          y2diff <= (z2in > screen) ? y2in - y1in : y_eye - y2in;
        end

        DIV_DONE:   begin
          // point 1
          x1prod <= z1prop * x1diff;
          y1prod <= z1prop * y1diff;

          // point 2
          x2prod <= z2prop * x1diff;
          y2prod <= z2prop * y1diff;
        end

        MULT_DONE:  begin
          // point 1
          x1out <= (z1in <= screen) ? x1in - x1disp : x_eye - x1disp;
          y1out <= (z1in <= screen) ? y1in - y1disp : y_eye - y1disp;

          // point 2
          x2out <= (z2in <= screen) ? x2in - x2disp : x_eye - x2disp;
          y2out <= (z2in <= screen) ? y2in - y2disp : y_eye - y2disp;

          done <= 1;
        end
      endcase

    end

    // both points in front of screen; nothing to display
    // output sentinel value 16'FFFF
    else  begin
      x1out <= 16'hFFFF;
      y1out <= 16'hFFFF;
      x2out <= 16'hFFFF;
      y2out <= 16'hFFFF;

      done <= 1;
    end
  end

  always @ (state)  begin
    next = state + 1;

    case (state)
      IDLE:   next = IDLE;
      MULT_DONE:   next = IDLE;
    endcase
  end

endmodule


// linedraw.v
// Very much not done! Specificies pixels that should be lit, given a line
// segment
//
// linedraw_params - calculates how many pixels in x direction changes for
// ever change in y-pixel (see paper)
//
// linedraw - takes output of linedraw_params and writes to memory
//
// Does not deal with with 16'hFFFF "nothing to display" value yet
`timescale 1ns / 1ps

module linedraw_params(clk, reset, start, done, p1x, p1y, p2x, p2y,
    win_x_lower, win_y_lower, win_x_upper, win_y_upper,
    data_ready);

  input clk;
  input reset;
  input start;
  output done;
  input signed [15:0] p1x, p1y;
  input signed [15:0] p2x, p2y;
  input signed [15:0] win_x_lower, win_y_lower;
  input signed [15:0] win_x_upper, win_y_upper;
  output data_ready;

  reg [5:0] state, next;
  reg data_ready;

  // (x0, y0) and (x1, y1) are same as inputs p1, p2, except
  // (maybe) swapped for y0 >= y1
  wire signed [15:0] x0, y0;
  wire signed [15:0] x1, y1;

  reg signed [15:0] x0_from_left, y0_from_bot;
  reg signed [15:0] x1_from_left, y1_from_bot;
  wire signed [26:0] x0_from_left_27bit, y0_from_bot_27bit;
  wire signed [26:0] x1_from_left_27bit, y1_from_bot_27bit;
  reg signed [15:0] win_x_diff, win_y_diff;
  reg signed [15:0] seg_x_diff, seg_y_diff;


  // 12 bits integer part, 11 bits fractional

  // 512 / (x_h - x_l)
  wire signed [26:0] dpix_x_per_x;
  wire signed [15:0] dpix_x_per_x_int;
  wire signed [11:0] dpix_x_per_x_frac;

  // (y_h - y_l) / 384
  wire signed [26:0] dy_per_pix_y;
  wire signed [15:0] dy_per_pix_y_int;
  wire signed [11:0] dy_per_pix_y_frac;

  // 384 / (y_h - y_l)
  wire signed [26:0] dpix_y_per_y;
  wire signed [15:0] dpix_y_per_y_int;
  wire signed [11:0] dpix_y_per_y_frac;
  wire signed [15:0] dpix_y_per_y_16bit;

  // (x_1 - x_0) / (y_1 - y_0)
  wire signed [26:0] inv_slope;
  wire signed [15:0] inv_slope_int;
  wire signed [11:0] inv_slope_frac;

  // (x_1 - x_0) / (y_1 - y_0)  *  (y_h - y_l) / 384
  // For every change in y_pixel, x_coordinate changes:
  reg signed [53:0] dx_per_pix_y_prod;
//  wire signed [26:0] dx_per_pix_y_27bit;
  wire signed [26:0] dx_per_pix_y;

  // 512 / (x_h - x_l)  *  (y_h - y_l) / 384
  reg signed [53:0] window_scale_prod;
  wire signed [26:0] window_scale;

  // (x_1 - x_0) / (y_1 - y_0) * window_scale
  // For every change in y_pixel, x_pixel changes:
  reg signed [53:0] x_pixel_change_prod;
  reg signed [26:0] x_pixel_change;

  // pixel coords of (x0, y0), (x1, y1)
  // actual coords are just integer parts
  reg signed [53:0] x0_pix_prod, y0_pix_prod;
  reg signed [53:0] x1_pix_prod, y1_pix_prod;
  wire signed [15:0] x0_pix_tmp, y0_pix_tmp;
  wire signed [15:0] x1_pix_tmp, y1_pix_tmp;
  reg signed [15:0] x0_pix, y0_pix;
  reg signed [15:0] x1_pix, y1_pix;

  // named states
  parameter IDLE = 6'd0;
  parameter DIV_DONE = 6'd33;
  parameter MULT_DONE = 6'd39;

  // (y_h - y_l) / 384
  division (clk, 1'd0, win_y_diff, 384,
      dy_per_pix_y_int, dy_per_pix_y_frac, rfd1);

  // 384 / (y_h - y_l)
  division (clk, 1'd0, 384, win_y_diff,
      dpix_y_per_y_int, dpix_y_per_y_frac, rfd2);

  // 512 / (x_h - x_l)
  division (clk, 1'd0, 512, win_x_diff,
      dpix_x_per_x_int, dpix_x_per_x_frac, rfd3);

  // (x_1 - x_0) / (y_1 - y_0)
  division (clk, 1'd0, seg_x_diff, seg_y_diff,
      inv_slope_int, inv_slope_frac, rfd4);

  // assign (x0, y0) to be one with the larger y
  assign x0 = (p1y >= p2y) ? p1x : p2x;
  assign y0 = (p1y >= p2y) ? p1y : p2y;
  assign x1 = (p1y >= p2y) ? p2x : p1x;
  assign y1 = (p1y >= p2y) ? p2y : p1y;
  // collapse 54-bit to 27-/16-bit, keeping sign
//  assign dx_per_pix_y_27bit = {dx_per_pix_y_prod[53], dx_per_pix_y_prod[36:11]};
  assign dx_per_pix_y = {dx_per_pix_y_prod[53], dx_per_pix_y_prod[25:11]};
  assign x_pixel_change = {x_pixel_change_prod[53], x_pixel_change_prod[36:11]};

  // expand 16-bit to 24-bit, keeping sign
  assign x0_from_left_27bit = { {12{x0_from_left[15]}}, x0_from_left[14:0] };
  assign y0_from_bot_27bit = { {12{y0_from_bot[15]}}, y0_from_bot[14:0] };
  assign x1_from_left_27bit = { {12{x1_from_left[15]}}, x1_from_left[14:0] };
  assign y1_from_bot_27bit = { {12{y1_from_bot[15]}}, y1_from_bot[14:0] };

  // concatenate integer and fractional parts
  assign dpix_x_per_x = {dpix_x_per_x_int, dpix_x_per_x_frac[10:0]};
  assign dpix_y_per_y = {dpix_y_per_y_int, dpix_x_per_y_frac[10:0]};
  assign dy_per_pix_y = {dy_per_pix_y_int, dy_per_pix_y_frac[10:0]};
  assign inv_slope = {inv_slope_int, inv_slope_frac[10:0]};

  // collapse 27-bit to 16-bit, keeping sign
  assign dpix_y_per_y_16bit = {dpix_y_per_y[26], dpix_y_per_y[14:0]};

  // pick out integer parts
  assign x0_pix_tmp = x0_pix_prod[37:22];
  assign y0_pix_tmp = y0_pix_prod[37:22];
  assign x1_pix_tmp = x1_pix_prod[37:22];
  assign y1_pix_tmp = y1_pix_prod[37:22];

  always @ (posedge clk)  begin
    win_x_diff <= win_x_upper - win_x_lower;
    win_y_diff <= win_y_upper - win_y_lower;

    window_scale_prod <= dpix_x_per_x * dy_per_pix_y;

    if (reset)  begin
      state <= IDLE;
      data_ready <= 0;
    end

    else if (start)   begin
      seg_x_diff <= x1 - x0;
      seg_y_diff <= y1 - y0;
      x0_from_left <= x0 - win_x_lower;
      y0_from_bot <= y0 - win_y_lower;
      x1_from_left <= x1 - win_x_lower;
      y1_from_bot <= y1 - win_y_lower;

      state <= 6'd1;

      data_ready <= 0;
    end

    else  begin
      state <= next;

      case (state)
        DIV_DONE:   begin
          x_pixel_change_prod <= inv_slope * window_scale;
          dx_per_pix_y_prod <= inv_slope * dy_per_pix_y;

          x0_pix_prod <= dpix_x_per_x * x0_from_left_27bit;
          y0_pix_prod <= dpix_y_per_y * y0_from_bot_27bit;
          x1_pix_prod <= dpix_x_per_x * x1_from_left_27bit;
          y1_pix_prod <= dpix_y_per_y * y1_from_bot_27bit;

          data_ready <= 0;
        end

        MULT_DONE:  begin
          // the *_pix_tmp are the integer parts of *_pix_prod
          x0_pix <= x0_pix_tmp;
          y0_pix <= 16'd384 - y0_pix_tmp;
          x1_pix <= x0_pix_tmp;
          y1_pix <= 16'd384 - y1_pix_tmp;

          data_ready <= 1;
        end

        default:  begin
          data_ready <= 0;
        end
      endcase

    end
  end

  always @ (state)  begin
    next = state + 1;
    case (state)
      IDLE:   next = IDLE;
      MULT_DONE:  next = IDLE;
    endcase
  end
endmodule


module linedraw(clk, reset, start, done, data_ready, x0, y0, x1, y1,
    x0pix, y0pix, x1pix, y1pix,
    dx, dx_pixel, dy,
    addr, data_w, we);
  input clk;
  input reset;
  input start;
  output done;
  input data_ready;
  input signed [15:0] x0, y0;
  input signed [15:0] x1, y1;
  input signed [15:0] x0pix, y0pix;
  input signed [15:0] x0pix, y0pix;
  input signed [15:0] dx;
  input signed [26:0] dx_pixel;
  input signed [15:0] dy;
  output output_ready;
  output [9:0] addr;
  output [511:0] data_w;
  output we;

  // non-integers
  reg signed [15:0] start_x, start_y;
  reg signed [15:0] end_x, end_y;

  // integers
  reg signed [15:0] start_x_pix, start_y_pix;
  reg signed [15:0] end_x_pix, end_y_pix;

  reg [511:0] leftmask, rightmask;
  parameter ONES = {512{1'b1}};

  reg output_ready;

  always @ (posedge clk)  begin
    // start iterating
    if (data_ready)   begin
      start_x <= x0;
      start_y <= y0;
      end_x <= x1;
      end_y <= y1;

      start_x_pix <= x0pix;
      start_y_pix <= y0pix;
      end_x_pix <= x1pix;
      end_y_pix <= y1pix;

      done <= 0;
    end

    // end iterating
    else if (start_y == end_y)  begin
      done <= 1;

      if (start_x >= 0 && start_x < 512 &&
          start_y >= 0 && start_y < 384)  begin

      end
    end

    else  begin

      if (start_x >= 0 && start_x < 512 &&
          start_y >= 0 && start_y < 384)   begin
      if (
    end
  end

  always @ (state)  begin
    IDLE:   next = IDLE;
    RIGHTMASK:  begin

    end
  end

endmodule


// bufctrl.v
// RAM controller for display double buffer
`timescale 1ns / 1ps

// ramsel = 0: writer => buf1, reader => buf0
// ramsel = 1: writer => buf0, reader => buf1

module bufctrl(hold, clk, reset, newframe,
    w_start, w_addr, w_data_in,
    r_start, r_addr, r_data_out,
    addr0, data0in, data0out, we0, addr1, data1in, data1out, we1,
// debug
ramsel, w_state, r_state);
  input hold;
  input clk;
  input reset;
  input newframe;

  input w_start, r_start;
  input [8:0] w_addr, r_addr;
  input [511:0] w_data_in;
  output [511:0] r_data_out;

  output [8:0] addr0, addr1;
  output [511:0] data0in, data1in;
  input [511:0] data0out, data1out;
  output we0, we1;
// debug
output ramsel;
output [2:0] w_state, r_state;

  reg ramsel;

  reg [8:0] addr0, addr1;
  reg [8:0] w_addr_reg, r_addr_reg;
  reg [511:0] r_data_out, w_data_reg;
  reg [511:0] w_data_or;
  reg [511:0] data0in, data1in;
  reg we0, we1;

  reg [2:0] w_state, r_state, w_next, r_next;

  parameter IDLE = 3'd0;
  parameter READ = 3'd1;
  parameter WAIT = 3'd2;
  parameter PROCESS = 3'd3;
  parameter WRITE = 3'd4;


  always @ (posedge clk)  begin
    if (reset)  begin
      ramsel <= 0;
    end

    else if (!hold)   begin
      if (newframe) begin
        ramsel <= ~ramsel;
      end

      w_state <= w_start ? READ : w_next;
      r_state <= r_start ? READ : r_next;

      if (w_start)  begin
        w_addr_reg <= w_addr;
        w_data_reg <= w_data_in;
      end

      if (r_start)  begin
        r_addr_reg <= r_addr;
      end

      case (w_state)
        IDLE:   begin
          if (ramsel)   begin
            we0 <= 0;
          end

          else  begin
            we1 <= 0;
          end
        end

        READ:   begin
          if (ramsel)   begin
            addr0 <= w_addr_reg;
            we0 <= 0;
          end

          else  begin
            addr1 <= w_addr_reg;
            we1 <= 0;
          end
        end

        PROCESS:  begin
          if (ramsel)   w_data_or <= data0out | w_data_reg;
          else          w_data_or <= data1out | w_data_reg;
        end
        WRITE:  begin
          if (ramsel)   begin
            data0in <= w_data_or;
            we0 <= 1;
          end

          else  begin
            data1in <= w_data_or;
            we1 <= 1;
          end
        end
      endcase


      case (r_state)
        IDLE:   begin
          if (ramsel)   begin
            we1 <= 0;
          end

          else  begin
            we0 <= 0;
          end
        end

        READ:   begin
          if (ramsel)   begin
            addr1 <= r_addr_reg;
            we1 <= 0;
          end

          else  begin
            addr0 <= r_addr_reg;
            we0 <= 0;
          end
        end

        PROCESS:  begin
          if (ramsel)   r_data_out <= data1out;
          else          r_data_out <= data0out;
        end

        WRITE:  begin
          if (ramsel)   begin
            data1in <= {512{1'b0}};
            we1 <= 1;
          end

          else  begin
            data0in <= {512{1'b0}};
            we0 <= 1;
          end
        end
      endcase

    end
  end


  always @ (w_state)  begin
    w_next = w_state + 1;

    case (w_state)
      IDLE:   w_next = IDLE;
      WRITE:  w_next = IDLE;
    endcase
  end


  always @ (r_state)  begin
    r_next = r_state + 1;

    case (r_state)
      IDLE:   r_next = IDLE;
      WRITE:  r_next = IDLE;
    endcase
  end

endmodule


// display.v
// Outputs RGB values based on buffer contents
`timescale 1ns / 1ps

module display(hold, clk, reset, sc_x, sc_y, newframe, get_data, addr, data_r, rgb);
  input hold;
  input clk;
  input reset;
  input [10:0] sc_x;
  input [10:0] sc_y;
  input newframe;
  output get_data;
  input [511:0] data_r;
  output [8:0] addr;
  output [23:0] rgb;

  reg get_data;
  reg [511:0] data_reg;
  wire [8:0] addr;
  reg [23:0] rgb;
  reg [2:0] state, next;

  parameter IDLE = 3'd0;
  parameter READ_0 = 3'd1;
  parameter READ_1 = 3'd2;
  parameter READ_2 = 3'd3;
  parameter READ_3 = 3'd4;
  parameter READ_4 = 3'd5;

  parameter H_BPORCH = 11'd1343;

  assign addr = sc_y[9:1];

  always @ (posedge clk)  begin
    if (reset | newframe) begin
      state <= READ_0;
      get_data <= 1;
    end

    else if (!hold) begin
      // display green pixel if current (first) bit is 1, 0 otherwise
      rgb <= data_reg[sc_x[9:1]] ? 24'h00FF00 : 24'h0;

      // shift data_reg over by 1 for next iteration on every other pixel
      data_reg <= (state == READ_4) ? data_r : data_reg;

      // read new data when we get to end of line
      if (sc_x == H_BPORCH && sc_y < 768)   begin
        state <= READ_0;
        get_data <= 1;
      end

      else  begin
        state <= next;
        get_data <= 0;
      end

    end
  end

  always @ (state)  begin
    next = state + 1;

    case (state)
      IDLE:     next = IDLE;
      READ_4:   next = IDLE;
    endcase
  end

endmodule



// vga.v
// VGA timing controller
// 1024x768@60Hz

`timescale 1ns / 1ps

module vga(hold, clk, reset, pixel_count, line_count, hsync, vsync,
    blank_bar);
  input hold;
  input clk;
  input reset;
  output [10:0] pixel_count;
  output [10:0] line_count;
  output hsync, vsync;
  output blank_bar;

  reg [10:0] pixel_count, line_count;
  reg hsync, vsync, blank_bar;
  reg [10:0] next_h, next_v;

  parameter H_ACTIVE = 11'd1023;
  parameter H_FPORCH = 11'd1047;
  parameter H_SYNC = 11'd1183;
  parameter H_BPORCH = 11'd1343;
  parameter V_ACTIVE = 11'd767;
  parameter V_FPORCH = 11'd770;
  parameter V_SYNC = 11'd776;
  parameter V_BPORCH = 11'd805;

  always @ (posedge clk)  begin
    if (reset)  begin
      pixel_count <= 0;   line_count <= 0;
      hsync <= 1;   vsync <= 1;
      blank_bar <= 1;
    end

    else if (!hold) begin
//    else  begin
      pixel_count <= next_h;
      line_count <= next_v;

      hsync <= ~(next_h > H_FPORCH && next_h <= H_SYNC);
      vsync <= ~(next_v > V_FPORCH && next_v <= V_SYNC);

      blank_bar <= (next_h <= H_ACTIVE && next_v <= V_ACTIVE);
    end
  end

  always @ (pixel_count or line_count)  begin
    if (pixel_count == H_BPORCH)  begin
      next_h = 11'b0;
      next_v = (line_count == V_BPORCH) ? 11'b0 : line_count + 1;
    end
    else  begin
      next_h = pixel_count + 1;
      next_v = line_count;
    end
  end

endmodule
`timescale 1ns / 1ps


// debounce.v
// Mechanical debouncer/synchronizer
module debounce (reset, clock, noisy, clean);
   parameter DELAY = 648000;
   input reset, clock, noisy;
   output clean;

   reg [19:0] count;
   reg new, clean;

   always @(posedge clock)
     if (reset)
       begin
     count <= 0;
     new <= noisy;
     clean <= noisy;
       end
     else if (noisy != new)
       begin
     new <= noisy;
     count <= 0;
       end
     else if (count == DELAY)
       clean <= new;
     else
       count <= count + 1;

endmodule


// delay.v
// 8-cycle delay module (for hsync/vsync/blank outputs)
`timescale 1ns / 1ps

module delay(clk, reset, in, delayed);
  input clk;
  input reset;
  input in;
  output delayed;

  reg stage1, stage2, stage3, stage4, stage5, stage6, stage7, stage8, delayed;

  always @ (posedge clk)  begin
    stage1 <= in;
    stage2 <= stage1;
    stage3 <= stage2;
    stage4 <= stage3;
    stage5 <= stage4;
    stage6 <= stage5;
    stage7 <= stage6;
    delayed <= stage7;
  end
endmodule




/****************************************
 *  MODELSIM TEST BENCHES               *
 ****************************************/


`timescale 1ns / 1ps

module test_display(hold, clk, reset, newframe, write_next, addr, data_w);
  input hold;
  input clk;
  input reset;
  input newframe;
  output write_next;
  output [8:0] addr;
  output [511:0] data_w;

  reg write_next;
  reg [511:0] data_w;
  reg [8:0] addr;
  reg [2:0] count;

  always @ (posedge clk)  begin
    if (reset)  begin
      write_next <= 0;
      data_w <= 512'hZ;
      addr <= 9'h0;
      count <= 0;
    end

    else if (newframe)  begin
      addr <= 9'h0;
      write_next <= 1;
      count <= 0;
      data_w <= {512{1'b1}};
    end

    else if (addr == 9'd384)  begin
      write_next <= 0;
      data_w <= {512{1'b1}};
    end

    else if (!hold && count == 5) begin
//    else  begin

      // reset count
      count <= 0;

      write_next <= 1;
      addr <= addr + 1;

      // generate pattern

      if (addr == 9'd120 || addr == 9'd121)   begin
        data_w <= {512{1'b1}};
      end
      else if (addr[3:0] == 4'b1111) begin
        data_w <= { {126{1'b1}}, 2'b00, {126{1'b1}}, 4'b00, {126{1'b1}}, 2'b00, {126{1'b1}} };
      end
      else  begin
        data_w <= {512{1'b1}};
      end
    end

    else  begin
      write_next <= 0;
      count <= count + 1;
    end

  end

endmodule



module test_project_v;

  // Inputs
  reg clk;
  reg reset;
  reg start;
  reg [15:0] x_eye;
  reg [15:0] y_eye;
  reg [15:0] z_eye;
  reg [15:0] screen;
	reg [47:0] p1in;
	reg [47:0] p2in;

	// Outputs
	wire done;
	wire [31:0] p1out;
	wire [31:0] p2out;
wire [15:0] p1out_x, p1out_y, p2out_x, p2out_y;
wire [15:0] z1prop, z2prop;
wire [15:0] x1disp, y1disp, x2disp, y2disp;
wire [15:0] z1num, z1denom, z2num, z2denom;
wire [15:0] x1diff, y1diff, x2diff, y2diff;
wire [5:0] state;


	// Instantiate the Unit Under Test (UUT)
	project uut (
		.clk(clk),
		.reset(reset),
		.start(start),
		.done(done),
		.x_eye(x_eye),
		.y_eye(y_eye),
		.z_eye(z_eye),
		.screen(screen),
		.p1in(p1in),
		.p2in(p2in),
		.p1out(p1out),
		.p2out(p2out),
.p1out_x(p1out_x),
.p1out_y(p1out_y),
.p2out_x(p2out_x),
.p2out_y(p2out_y),
.z1prop(z1prop),
.z2prop(z2prop),
.x1disp(x1disp),
.y1disp(y1disp),
.x2disp(x2disp),
.y2disp(y2disp),
.z1num(z1num),
.z1denom(z1denom),
.z2num(z2num),
.z2denom(z2denom),
.x1diff(x1diff),
.y1diff(y1diff),
.x2diff(x2diff),
.y2diff(y2diff),
.state(state)
	);

	initial begin
		// Initialize Inputs
		clk = 0;
		reset = 0;
		start = 0;
		// eye = (3, 5, 10)
		x_eye = 16'h1800;
		y_eye = 16'h2800;
		z_eye = 16'h5000;
		// screen = 7
		screen = 16'h3800;
		p1in = 0;
		p2in = 0;

		// Wait 100 ns for global reset to finish
		#100;

		// Add stimulus here
		start = 1;  #20;  start = 0;  #50;

		p1in = 48'h4000_3800_2800;  // (8, 7, 5)
		p2in = 48'h1000_2000_0800;  // (2, 4, 1)
		start = 1;  #20;  start = 0;  #500;

		p1in = 48'h4000_3800_7000;  // (8, 7, 14)
		p2in = 48'h1000_E000_F800;  // (2, -4, -1)
		start = 1;  #20;  start = 0;  #500;

		p1in = 48'hC000_3800_D800;  // (-8, 7, -5)
		p2in = 48'h1000_2000_6000;  // (2, 4, 12)
		start = 1;  #20;  start = 0;  #500;

		p1in = 48'h4000_3800_7000;  // (8, 7, 14)
		p2in = 48'h1000_2000_6000;  // (2, 4, 12)
		start = 1;  #20;  start = 0;  #500;
	end

	always #5 clk = ~clk;

endmodule
```

```
`timescale 1ns / 1ps

module test_transform_v;

	// Inputs
	reg clk;
	reg reset;
	reg start;
	reg newframe;
	reg [95:0] data;
	reg [3:0] dirs;
	reg [1:0] zoom;
	reg fnselect;

	// Outputs
	wire done;
	wire eof;
	wire [8:0] addr_r;
	wire [8:0] addr_w;
	wire we;
	wire [47:0] p1out;
	wire [47:0] p2out;

	// Instantiate the Unit Under Test (UUT)
	transform uut (
		.clk(clk),
		.reset(reset),
		.start(start),
		.done(done),
		.newframe(newframe),
		.eof(eof),
		.addr_r(addr_r),
		.addr_w(addr_w),
		.we(we),
		.data(data),
		.p1out(p1out),
		.p2out(p2out),
		.dirs(dirs),
		.zoom(zoom),
		.fnselect(fnselect)
	);

	initial begin
		// Initialize Inputs
		clk = 1;
		reset = 0;
		start = 0;
		newframe = 0;
		data = 0;
		dirs = 0;
		zoom = 0;
		fnselect = 0;

		// Wait 100 ns for global reset to finish
		#100;

		// Add stimulus here
    reset = 1;  #10;  reset = 0;  #10;

    data = 96'h1000_E000_F800_0000_0000_0000;   // (2, -4, -1) to (0, 0, 0)
    start = 1;  #10;  start = 0;  #10;
    fnselect = 0;
    dirs = 4'b1111;   zoom = 2'b00;   #50;  start = 1;  #10;  start = 0;
    dirs = 4'b1110;   zoom = 2'b00;   #50;  start = 1;  #10;  start = 0;
    dirs = 4'b1100;   zoom = 2'b00;   #50;  start = 1;  #10;  start = 0;
    dirs = 4'b1000;   zoom = 2'b00;   #50;  start = 1;  #10;  start = 0;
    dirs = 4'b0000;   zoom = 2'b00;   #50;  start = 1;  #10;  start = 0;
    dirs = 4'b0000;   zoom = 2'b10;   #50;  start = 1;  #10;  start = 0;
    dirs = 4'b0000;   zoom = 2'b11;   #70;  start = 1;  #10;  start = 0;
    fnselect = 1;
    dirs = 4'b1111;   zoom = 2'b00;   #50;  start = 1;  #10;  start = 0;
    dirs = 4'b1110;   zoom = 2'b00;   #50;  start = 1;  #10;  start = 0;
    dirs = 4'b1100;   zoom = 2'b00;   #50;  start = 1;  #10;  start = 0;
    dirs = 4'b1000;   zoom = 2'b00;   #50;  start = 1;  #10;  start = 0;
    dirs = 4'b0000;   zoom = 2'b00;   #50;  start = 1;  #10;  start = 0;
    dirs = 4'b0000;   zoom = 2'b10;   #50;  start = 1;  #10;  start = 0;
    dirs = 4'b0000;   zoom = 2'b11;   #70;

    data = 96'h1000_E000_F800_4000_3800_7000;   // (2, -4, -1) to (8, 7, 14)
    start = 1;  #10;  start = 0;  #60;
    fnselect = 0;
    dirs = 4'b1111;   zoom = 2'b00;   #50;  start = 1;  #10;  start = 0;
    fnselect = 1;
    dirs = 4'b1000;   zoom = 2'b00;   #50;  start = 1;  #10;  start = 0;
    dirs = 4'b0000;   zoom = 2'b11;   #70;
    data = 96'hFFFF_FFFF_FFFF_FFFF_FFFF_FFFF;   // eof
    start = 1;  #10;  start = 0;  #10;

    newframe = 1;   #10;  newframe = 0;   #10;
        end

  always #5 clk = ~clk;

endmodule




/****************************************
 *  OLD display.v                       *
 ****************************************/


// display.old.v
//
// Old (pre-rewrite) display module
// read to RGB takes 5 cycles
// **NB: delay VGA hsync/vsync signals by 5 cycles

`timescale 1ns / 1ps

module display(clk, reset, sc_x, sc_y, clrram, ramsel,
    we1, we2, data1, data2, addr1, addr2,
    rgb);
  input clk;
  input reset;
  input [10:0] sc_x;
  input [10:0] sc_y;
  input clrram;           // for later, if we decide to support drawing over
                          // multiple frames (so we want to keep old frame)
  input ramsel;
  inout [35:0] data1;
  inout [35:0] data2;
  output we1;
  output we2;
  output [18:0] addr1;
  output [18:0] addr2;
  output [23:0] rgb;

  wire [35:0] data;
  wire [18:0] addr;

  reg state, next;
  reg [31:0] data_reg;
  reg [23:0] rgb;

  parameter IDLE = 3'd0;
  parameter READ_0 = 3'd1;
  parameter READ_1 = 3'd2;
  parameter READ_2 = 3'd3;
  parameter READ_3 = 3'd4;
  parameter WRITE_0 = 3'd5;
  parameter WRITE_1 = 3'd6;

  assign data = (state == WRITE_0) ? 36'h0 : 36'hZ;
  assign we = ~(ramsel && state == WRITE_0);

  assign addr = (sc_y << 5) + (sc_x >> 5);

  always @ (posedge clk)  begin
    rgb <= (sc_x[3] | sc_y[3]) ? 24'h00FF00 : 24'h0;
    // display green pixel if current (first) bit is 1, 0 otherwise
    rgb <= data_reg[31] ? 24'h00FF00 : 24'h0;

    // shift data_reg over by 1 for next iteration
    data_reg <= (state == READ_3) ? data[35:4] : data_reg << 1;

    // read new memory location every 32 pixels
    state <= (sc_x[4:0] == 5'b11111) ? READ_1 : next;
  end

  always @ (state)  begin
    case (state)
      IDLE:     next = IDLE;

      READ_0: begin
        next = READ_1;
      end

      READ_1:   next = READ_2;

      READ_2:   next = READ_3;

      READ_3:   next = IDLE;
      WRITE_0:  begin
//        if (clrram) begin
          next = WRITE_1;
//        end
      end

      // stop writing after one cycle (to be on the safe side)
      WRITE_1:  begin
        next = IDLE;
      end

      default:  begin
        next = IDLE;
      end
    endcase
  end

  always @ (sc_x or sc_y) begin
    addr = ( sc_y << Y_LOC_LSHIFT ) + ( sc_x >> X_LOC_RSHIFT );
    data = (sc_x[4:0] <= 4) ? 36'hZ : 36'h0;
    we = sc_x[4:0] <= 4;
  end

endmodule




/****************************************
 *  MEMORY TIMING TESTER                *
 ****************************************/

///////////////////////////////////////////////////////////////////////////////
//
// 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 labkit (beep, audio_reset_b, ac97_sdata_out, ac97_sdata_in, ac97_synch,
         ac97_bit_clock,

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

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

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

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

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

         clock_feedback_out, clock_feedback_in,

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

         rs232_txd, rs232_rxd, rs232_rts, rs232_cts,

         mouse_clock, mouse_data, keyboard_clock, keyboard_data,

         clock_27mhz, clock1, clock2,

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

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

         switch,

         led,

         user1, user2, user3, user4,

         daughtercard,

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

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

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

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

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

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

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

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

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

   output rs232_txd, rs232_rts;
   input  rs232_rxd, rs232_cts;

   input  mouse_clock, mouse_data, keyboard_clock, keyboard_data;

   input  clock_27mhz, clock1, clock2;

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

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

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

   inout [43:0] daughtercard;

   inout  [15:0] systemace_data;
   output [6:0]  systemace_address;
   output systemace_ce_b, systemace_we_b, systemace_oe_b;
   input  systemace_irq, systemace_mpbrdy;

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

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

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

   // VGA Output
   assign vga_out_red = 8'h0;
   assign vga_out_green = 8'h0;
   assign vga_out_blue = 8'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;
   assign user4 = 32'hZ;

   // Daughtercard Connectors
   assign daughtercard = 44'hZ;

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

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



  // ** MY STUFF BEGINS HERE

  // declarations
  wire reset_sync;
  wire clk_sync;
  wire newframe_sync;
  wire we0_sync;
  wire we1_sync;
  wire [19:0] dot_sel_addr;
  wire [31:0] dot_sel_data;
  wire [39:0] dots_addr_18_16, dots_addr_15_12,
      dots_addr_11_8, dots_addr_7_4,
      dots_addr_3_0;
  wire [39:0] dots_data_35_32, dots_data_31_28,
      dots_data_27_24, dots_data_23_20,
      dots_data_19_16, dots_data_15_12,
      dots_data_11_8, dots_data_7_4;

  // assignments
  assign ram0_clk = ~clock_27mhz;
  assign ram1_clk = ~clock_27mhz;

//  assign ram0_clk = ~clk_sync;
//  assign ram1_clk = ~clk_sync;

//  assign ram0_data = 36'hZ;
//  assign ram0_address = {16'h0, switch[2:0]};
  assign ram0_adv_ld = 1'b0;
  assign ram0_cen_b = 1'b0;
  assign ram0_ce_b = 1'b0;
  assign ram0_oe_b = 1'b0;
//  assign ram0_we_b = 1'b1;
  assign ram0_bwe_b = 4'hF;

//  assign ram1_data = 36'hZ;
//  assign ram1_address = {16'h0, switch[2:0]};
  assign ram1_adv_ld = 1'b0;
  assign ram1_cen_b = 1'b0;
  assign ram1_ce_b = 1'b0;
  assign ram1_oe_b = 1'b0;
//  assign ram1_we_b = 1'b1;
  assign ram1_bwe_b = 4'hF;

  assign dot_sel_addr = switch[7] ? {0, ram1_address} : {0, ram0_address};
  assign dot_sel_data = switch[7] ? ram1_data[35:4] : ram0_data[35:4];

  dots d15(clock_27mhz, dot_sel_addr[19:16], dots_addr_18_16);
  dots d14(clock_27mhz, dot_sel_addr[15:12], dots_addr_15_12);
  dots d13(clock_27mhz, dot_sel_addr[11:8], dots_addr_11_8);
  dots d12(clock_27mhz, dot_sel_addr[7:4], dots_addr_7_4);
  dots d11(clock_27mhz, dot_sel_addr[3:0], dots_addr_3_0);
  dots d7(clock_27mhz, dot_sel_data[31:28], dots_data_35_32);
  dots d6(clock_27mhz, dot_sel_data[27:24], dots_data_31_28);
  dots d5(clock_27mhz, dot_sel_data[23:20], dots_data_27_24);
  dots d4(clock_27mhz, dot_sel_data[19:16], dots_data_23_20);
  dots d3(clock_27mhz, dot_sel_data[15:12], dots_data_19_16);
  dots d2(clock_27mhz, dot_sel_data[11:8], dots_data_15_12);
  dots d1(clock_27mhz, dot_sel_data[7:4], dots_data_11_8);
  dots d0(clock_27mhz, dot_sel_data[3:0], dots_data_7_4);

  dots_display mem_disp(reset_sync, clock_27mhz, disp_blank, disp_clock,
    disp_rs, disp_ce_b, disp_reset_b, disp_data_out,
    {dots_addr_18_16, dots_addr_15_12, dots_addr_11_8, dots_addr_7_4,
    dots_addr_3_0,
    120'd0,
    dots_data_35_32, dots_data_31_28, dots_data_27_24, dots_data_23_20,
    dots_data_19_16, dots_data_15_12, dots_data_11_8, dots_data_7_4});
  assign led[7:2] = 6'b11_1111;
  assign led[1] = ram1_we_b;
  assign led[0] = ram0_we_b;

  debounce reset_button(0, clock_27mhz, ~button_enter, reset_sync);
  debounce newframe_button(0, clock_27mhz, ~button_left, newframe_sync);
  debounce clk_button(0, clock_27mhz, ~button3, clk_sync);
  debounce we0_button(0, clock_27mhz, button0, we0_sync);
  debounce we1_button(0, clock_27mhz, button1, we1_sync);

  bufctrl test_bufctrl(0, clock_27mhz, reset_sync, newframe_sync,
    {16'd0, switch[2:0]}, asdf, {switch[6:3], 32'd0}, we0_sync, w_data_ready,
    {16'd0, switch[2:0]}, asdfa, {switch[6:3], 32'd0}, we1_sync, r_data_ready,
    ram0_address, ram0_data, ram0_we_b, ram1_address, ram1_data, ram1_we_b);

//  // Using push button as manual clock
//  bufctrl test_bufctrl(0, clk_sync, reset_sync, newframe_sync,
//    {16'd0, switch[2:0]}, asdf, {switch[6:3], 32'd0}, we0_sync, w_data_ready,
//    {16'd0, switch[2:0]}, asdfa, {switch[6:3], 32'd0}, we1_sync, r_data_ready,
//    ram0_address, ram0_data, ram0_we_b, ram1_address, ram1_data, ram1_we_b);


   assign analyzer1_data[15] = ram0_we_b;
   assign analyzer1_data[14:12] = ram0_address[2:0];
   assign analyzer1_data[11:8] = ram0_data[35:32];
   assign analyzer1_data[7:0] = 8'h0;
   assign analyzer1_clock = clock_27mhz;
   assign analyzer3_data[15] = ram1_we_b;
   assign analyzer3_data[14:12] = ram1_address[2:0];
   assign analyzer3_data[11:8] = ram1_data[35:32];
   assign analyzer3_data[7:0] = 8'h0;
   assign analyzer3_clock = ram0_clk;

endmodule


// bufctrl.old.v
// Old display buffer control module for ZBT RAM
//
// ISSUES:
// - 2-cycle delay. What happens when delay spills over ramsel <= ~ramsel ?
`timescale 1ns / 1ps

// ramsel = 0: writer => buf1, reader => buf0
// ramsel = 1: writer => buf0, reader => buf1

module bufctrl(hold, clk, reset, newframe,
    w_addr, w_data_out, w_data_in, w_we, w_data_ready,
    r_addr, r_data_out, r_data_in, r_we, r_data_ready,
    addr0, data0, we0, addr1, data1, we1);
  input hold;
  input clk;
  input reset;
  input newframe;

  input [18:0] w_addr;
  output [35:0] w_data_out;
  input [35:0] w_data_in;
  input w_we;
  output w_data_ready;

  input [18:0] r_addr;
  output [35:0] r_data_out;
  input [35:0] r_data_in;
  input r_we;
  output r_data_ready;

  output [18:0] addr0, addr1;
  inout [35:0] data0, data1;
  output we0, we1;

  reg ramsel;

  reg [18:0] addr0, addr1;
  reg [35:0] r_data_out, w_data_out;
  reg r_data_ready, w_data_ready;
  reg we0, we1;
  reg [35:0] data0reg, data1reg;
  wire [35:0] data0, data1;

  // write data delay
  reg [35:0] data0delay1, data0delay2;
  reg [35:0] data1delay1, data1delay2;
  // 3rd delay for when reading; 1 from registered r/w_we, 2 from RAM, 1 from registered on way back
  reg data0tri1, data0tri2, data0tri3;
  reg data1tri1, data1tri2, data1tri3;

  assign data0 = data0reg;
  assign data1 = data1reg;

  always @ (posedge clk)  begin
    if (reset)  begin
      ramsel <= 0;
    end

    else if (!hold)   begin
//    else  begin
      if (newframe) begin
        ramsel <= ~ramsel;
      end

      if (ramsel) begin
        //// writing to ram0
        addr0 <= w_addr;
        we0 <= w_we;

        // delays
        data0tri1 <= w_we;
        data0delay1 <= w_data_in;

        w_data_out <= data0;
//        w_data_out <= data0tri3 ? data0 : 36'hZ;
        w_data_ready <= data0tri3;

        //// displaying from ram1
        addr1 <= r_addr;
        we1 <= r_we;

        // delays
        data1tri1 <= r_we;
        data1delay1 <= r_data_in;

        r_data_out <= data1;
//        r_data_out <= data1tri3 ? data1 : 36'hZ;
        r_data_ready <= data1tri3;
      end
      else  begin
        //// displaying from ram0
        addr0 <= r_addr;
        we0 <= r_we;

        // delays
        data0tri1 <= r_we;
        data0delay1 <= r_data_in;

        r_data_out <= data0;
//        r_data_out <= data0tri3 ? data0 : 36'hZ;
        r_data_ready <= data0tri3;

        //// writing to ram1
        addr1 <= w_addr;
        we1 <= w_we;

        // delays
        data1tri1 <= w_we;
        data1delay1 <= w_data_in;

        w_data_out <= data1;
//        w_data_out <= data1tri3 ? data1 : 36'hZ;
        w_data_ready <= data1tri3;
      end

      data0delay2 <= data0delay1;
      data1delay2 <= data1delay1;
      data0tri2 <= data0tri1;
      data0tri3 <= data0tri2;
      data1tri2 <= data1tri1;
      data1tri3 <= data1tri2;

      data0reg <= data0tri2 ? 36'hZ : data0delay2;
      data1reg <= data1tri2 ? 36'hZ : data1delay2;
    end
  end

endmodule


// test_mem_display.v
// LED display (taken and adapted from lab 3)

`timescale 1ns / 1ps

module dots_display (reset, clk_27mhz, disp_blank, disp_clock, disp_rs, disp_ce_b,
    disp_reset_b, disp_data_out, dots);

  input reset; // Active high
  input clk_27mhz;
  output disp_blank, disp_clock, disp_data_out, disp_rs, disp_ce_b,
    disp_reset_b;
  input [639:0] dots;

  reg disp_data_out, disp_rs, disp_ce_b, disp_reset_b;

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

  reg [5:0] count;
  reg [7:0] reset_count;
  reg clock;
  wire dreset;

  always @ (posedge clk_27mhz)  begin
    if (reset)  begin
      count = 0;
      clock = 0;
    end

    else if (count == 6'd27) begin
      clock = ~clock;
      count = 6'd0;
    end

    else    count = count+1;
  end

  always @ (posedge clk_27mhz)  begin
    if (reset)  reset_count <= 100;
    else    reset_count <= (reset_count==0) ? 0 : reset_count-1;
  end

  assign dreset = (reset_count != 0);

  assign disp_clock = ~clock;
  //
  // Display State Machine
  //

  reg [7:0] state;
  reg [9:0] dot_index;
  reg [31:0] control;
  reg [639:0] ldots;

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

  always @(posedge clock)   begin
    if (dreset)   begin
      state <= 0;
      dot_index <= 0;
      control <= 32'h7F7F7F7F;
    end

    else  begin
      casex (state)
        8'h00:  begin
          // Reset displays
          disp_data_out <= 1'b0;
          disp_rs <= 1'b0; // 0 = dot register
          disp_ce_b <= 1'b1;
          disp_reset_b <= 1'b0;
          dot_index <= 0;
          state <= state+1;
        end

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

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

        8'h03:  begin
          // Latch dot data
          disp_rs <= 1'b1; // Select the control register
          disp_ce_b <= 1'b1;
          dot_index <= 31;
          state <= state+1;
        end

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

        8'h05:  begin
          // Latch the control register data
          disp_rs <= 1'b0; // Select the dot register
          disp_ce_b <= 1'b1;
          dot_index <= 639;
          ldots <= dots;
          state <= state+1;
        end

        8'h06:  begin
          // Load the user's dot data into the dot register
          disp_ce_b <= 1'b0;
          disp_data_out <= ldots[639];
          ldots <= ldots<<1;
          if (dot_index == 0)
            state <= 5;
          else
            dot_index <= dot_index-1;
        end
      endcase
    end
  end

endmodule
```

```verilog
module dots(clk_27mhz, num, dots);
  input clk_27mhz;
  input [3:0] num;
  output [39:0] dots;

  reg [39:0] dots;
  always @ (posedge clk_27mhz)
    case (num)
      4'd15: dots <= 40'b01111111_00001001_00001001_00001001_00000001;  // 'F'
      4'd14: dots <= 40'b01111111_01001001_01001001_01001001_01000001;  // 'E'
      4'd13: dots <= 40'b01111111_01000001_01000001_01000001_00111110;  // 'D'
      4'd12: dots <= 40'b00111110_01000001_01000001_01000001_00100010;  // 'C'
      4'd11: dots <= 40'b01111111_01001001_01001001_01001001_00110110;  // 'B'
      4'd10: dots <= 40'b01111110_00001001_00001001_00001001_01111110;  // 'A'
      4'd09: dots <= 40'b00000110_01001001_01001001_00101001_00011110;  // '9'
      4'd08: dots <= 40'b00110110_01001001_01001001_01001001_00110110;  // '8'
      4'd07: dots <= 40'b00000001_01110001_00001001_00000101_00000011;  // '7'
      4'd06: dots <= 40'b00111100_01001010_01001001_01001001_00110000;  // '6'
      4'd05: dots <= 40'b00100111_01000101_01000101_01000101_00111001;  // '5'
      4'd04: dots <= 40'b00011000_00010100_00010010_01111111_00010000;  // '4'
      4'd03: dots <= 40'b00100010_01000001_01001001_01001001_00110110;  // '3'
      4'd02: dots <= 40'b01100010_01010001_01001001_01001001_01000110;  // '2'
      4'd01: dots <= 40'b00000000_01000010_01111111_01000000_00000000;  // '1'
      4'd00: dots <= 40'b00111110_01010001_01001001_01000101_00111110;  // '0'
      // No default case, becase every case is already accounted for.
    endcase

endmodule
`timescale 1ns / 1ps


// debounce.v
// Mechanical debouncer/synchronizer (for button pushes)
module debounce (reset, clock, noisy, clean);
   parameter DELAY = 270000;
   input reset, clock, noisy;
   output clean;

   reg [19:0] count;
   reg new, clean;

   always @(posedge clock)
     if (reset)
       begin
    count <= 0;
    new <= noisy;
    clean <= noisy;
       end
     else if (noisy != new)
       begin
    new <= noisy;
    count <= 0;
       end
     else if (count == DELAY)
       clean <= new;
     else
       count <= count + 1;

endmodule
```