Appendix:

Audio/Video Code:

```
//////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Template Toplevel Module
//
// For Labkit Revision 004
//
//
// Created: October 31, 2004, from revision 003 file
// Author: Nathan Ickes
//
//////////////////////////////////////////////////////////////////////////////
//
// CHANGES FOR BOARD REVISION 004
//
// 1) Added signals for logic analyzer pods 2-4.
// 2) Expanded "tv_in_ycrcb" to 20 bits.
// 3) Renamed "tv_out_data" to "tv_out_i2c_data" and "tv_out_sclk" to
//    "tv_out_i2c_clock".
// 4) Reversed disp_data_in and disp_data_out signals, so that "out" is an
//    output of the FPGA, and "in" is an input.
//
// CHANGES FOR BOARD REVISION 003
//
// 1) Combined flash chip enables into a single signal, flash_ce_b.
//
// CHANGES FOR BOARD REVISION 002
//
// 1) Added SRAM clock feedback path input and output
// 2) Renamed "mousedata" to "mouse_data"
// 3) Renamed some ZBT memory signals. Parity bits are now incorporated into
//    the data bus, and the byte write enables have been combined into the
//    4-bit ram#_bwe_b bus.
// 4) Removed the "systemace_clock" net, since the SystemACE clock is now
//    hardwired on the PCB to the oscillator.
//
//////////////////////////////////////////////////////////////////////////////
//
// Complete change history (including bug fixes)
//
// 2005-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-Apr-29: Change history started
//
// 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-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-Oct-31: Adapted to new revision 004 board.
//
//////////////////////////////////////////////////////////////////////////////

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;
// ac97_sdata_out and ac97_sdata_out are inputs;

// 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;
// disp_data_out 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;

   ////////////////////////////////////////////////////////////////////////
//
// Reset Generation
//
// A shift register primitive is used to generate an active-high reset
// signal that remains high for 16 clock cycles after configuration finishes
// and the FPGA's internal clocks begin toggling.
//
////////////////////////////////////////////////////////////////////////

wire reset;
SRL16 reset_sr (.D(1'b0), .CLK(clock_27mhz), .Q(reset),
                .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
defparam reset_sr.INIT = 16'hFFFF;

////////////////////////////////////////////////////////////////////////
//
// Audio Input and Output
//
////////////////////////////////////////////////////////////////////////

wire [3:0] vol;
wire [17:0] music;
wire [39:0] vdisp;
wire volume_up, volume_down;
wire [15:0] filtered_music;

assign filtered_music = music [17:2];

audio audio1 (reset, clock_27mhz, audio_reset_b, ac97_sdata_out,
              ac97_sdata_in, ac97_synch, ac97_bit_clock, switch[1:0],
              {vol, 1'b0}, switch[2], music, enable, filtered_music);

        assign analyzer3_clock = ac97_bit_clock;
        assign analyzer3_data = music;

        assign analyzer4_clock = ac97_bit_clock;
        assign analyzer4_data = enable;

beeper beep1 (reset, clock_27mhz, beep, ~button_enter);


          wire [6:0] x,y;
          wire enable_up, enable_down;
```

```
        fake_video_input fakeit(clock_27mhz, button_enter, button_up, button_down, button_right, button_left, x, y);
        amp_change ychange(clock_27mhz, button_enter, y, enable_up, enable_down);

debounce vol_up (reset, clock_27mhz, button_up, volume_up);//enable_up, volume_up);

debounce vol_down (reset, clock_27mhz, button_down, volume_down);//enable_down, volume_down);

volume vol1 (reset, clock_27mhz, volume_up, volume_down, vol, vdisp);

//////////////////////////////////////////////////////////////////////
//
// VGA Output
//
//////////////////////////////////////////////////////////////////////

vga vga_test (reset, clock_27mhz, 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);

//////////////////////////////////////////////////////////////////////
//
// Video I/O
//
//////////////////////////////////////////////////////////////////////

wire [1:0] videomode;

debounce vmode0 (reset, clock_27mhz, switch[3], videomode[0]);
debounce vmode1 (reset, clock_27mhz, switch[4], videomode[1]);

video video_test (reset, clock_27mhz, 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[19:10],
                     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, videomode);

PULLUP pu_in (.O(tv_in_i2c_data));
PULLUP pu_out (.O(tv_out_i2c_data));

assign user1 = {6'b000000,
                tv_in_line_clock1,  // 25
                tv_in_i2c_clock,    // 24
                tv_in_i2c_data,     // 23
                tv_in_ycrcb[19:10], // 22-13
                clock_27mhz,        // 12
                tv_out_i2c_clock,   // 11
                tv_out_i2c_data,    // 10
                tv_out_ycrcb};      // 9-0

assign analyzer1_clock = clock_27mhz;
assign analyzer1_data = {tv_out_i2c_clock,
                         tv_out_i2c_data,
                         4'h0, tv_out_ycrcb};

assign analyzer2_clock = tv_in_line_clock1;
assign analyzer2_data = {tv_in_i2c_clock,
                         tv_in_i2c_data,
                         4'h0, tv_in_ycrcb[19:10]};

//////////////////////////////////////////////////////////////////////
//
// Alphanumeric Displays
//
//////////////////////////////////////////////////////////////////////
//
// 7654321076543210
```

```
// V: 0 MIC CLRBARS
// V:32 SIN COMPOSI
// V: 0 SQR S-VIDEO
// V: 0 SIL MITLOGO
// V: 0 LIN

reg [79:0] volume_dots;
reg [119:0] audio_source_dots;
reg [279:0] video_source_dots;

always @(vol)
  case (vol)
    5'd15:
            volume_dots <= {40'b00000000_01000010_01111111_01000000_00000000, //15
                            40'b00100111_01000101_01000101_01000101_00111001};
    5'd14:
            volume_dots <= {40'b00000000_01000010_01111111_01000000_00000000, //14
                            40'b00011000_00010100_00010010_01111111_00010000};
    5'd13:
            volume_dots <= {40'b00000000_01000010_01111111_01000000_00000000, //13
                            40'b00100010_01000001_01001001_01001001_00110110};
    5'd12:
            volume_dots <= {40'b00000000_01000010_01111111_01000000_00000000, //12
                            40'b01100010_01010001_01001001_01001001_01000110};
    5'd11:
            volume_dots <= {40'b00000000_01000010_01111111_01000000_00000000, //11
                            40'b00000000_01000010_01111111_01000000_00000000};
    5'd10:
            volume_dots <= {40'b00000000_01000010_01111111_01000000_00000000, //10
                            40'b00111110_01010001_01001001_01000101_00111110};
    5'd09:
            volume_dots <= {40'b00000000_00000000_00000000_00000000_00000000, //9
                            40'b00000110_01001001_01001001_00101001_00011110};
    5'd08:
            volume_dots <= {40'b00000000_00000000_00000000_00000000_00000000, //8
                            40'b00110110_01001001_01001001_01001001_00110110};
    5'd07:
            volume_dots <= {40'b00000000_00000000_00000000_00000000_00000000, //7
                            40'b00000001_01110001_00001001_00000101_00000011};
    5'd06:
            volume_dots <= {40'b00000000_00000000_00000000_00000000_00000000, //6
                            40'b00111100_01001010_01001001_01001001_00110000};
    5'd05:
            volume_dots <= {40'b00000000_00000000_00000000_00000000_00000000, //5
                            40'b00100111_01000101_01000101_01000101_00111001};
    5'd04:
            volume_dots <= {40'b00000000_00000000_00000000_00000000_00000000, //4
                            40'b00011000_00010100_00010010_01111111_00010000};
    5'd03:
            volume_dots <= {40'b00000000_00000000_00000000_00000000_00000000, //3
                            40'b00100010_01000001_01001001_01001001_00110110};
    5'd02:
            volume_dots <= {40'b00000000_00000000_00000000_00000000_00000000, //2
                            40'b01100010_01010001_01001001_01001001_01000110};
    5'd01:
            volume_dots <= {40'b00000000_00000000_00000000_00000000_00000000, //1
                            40'b00000000_01000010_01111111_01000000_00000000};
    5'd00:
            volume_dots <= {40'b00000000_00000000_00000000_00000000_00000000, //0
                            40'b00111110_01010001_01001001_01000101_00111110};
    default:
            volume_dots <= {40'b00000000_00000000_00000000_00000000_00000000, //?
                            40'b00000010_00000001_01010001_00001001_00000110};
  endcase

always @(switch[2:0])
  case (switch[1:0])
    2'b00:
            if (switch[2])
              audio_source_dots
```

```
            <= {40'b01111111_00000010_00001100_00000010_01111111,
                40'b00000000_01000001_01111111_01000001_00000000,
                40'b00111110_01000001_01000001_01000001_00100010}; // MIC
          else
            audio_source_dots
            <= {40'b01111111_01000000_01000000_01000000_01000000,
                40'b00000000_01000001_01111111_01000001_00000000,
                40'b01111111_00000010_00000100_00001000_01111111}; // LIN
  2'b01:
          audio_source_dots
          <= {40'b00100110_01001001_01001001_01001001_00110010,
            40'b00000000_01000001_01111111_01000001_00000000,
            40'b01111111_01000000_01000000_01000000_01000000}; // SIL

  2'b10:
          audio_source_dots
          <= {40'b00100110_01001001_01001001_01001001_00110010,
            40'b00000000_01000001_01111111_01000001_00000000,
            40'b01111111_00000010_00000100_00001000_01111111}; // SIN
  2'b11:
          audio_source_dots
          <= {40'b00100110_01001001_01001001_01001001_00110010,
            40'b00111110_01000001_01010001_00100001_01011110,
            40'b01111111_00001001_00011001_00101001_01000110}; // SQR
 endcase

always @(videomode)
 case (videomode)
  2'b00:
          video_source_dots
          <= {40'b00111110_01000001_01000001_01000001_00100010, // C
            40'b01111111_01000000_01000000_01000000_01000000, // L
            40'b01111111_00001001_00011001_00101001_01000110, // R
            40'b01111111_01001001_01001001_01001001_00110110, // B
            40'b01111110_00001001_00001001_00001001_01111110, // A
            40'b01111111_00001001_00011001_00101001_01000110, // R
            40'b00100110_01001001_01001001_01001001_00110010};// S
  2'b01:
          video_source_dots
          <= {40'b01111111_00000010_00001100_00000010_01111111, // M
            40'b00000000_01000001_01111111_01000001_00000000, // I
            40'b00000001_00000001_01111111_00000001_00000001, // T
            40'b01111111_01000000_01000000_01000000_01000000, // L
            40'b00111110_01000001_01000001_01000001_00111110, // O
            40'b00111110_01000001_01001001_01001001_00111010, // G
            40'b00111110_01000001_01000001_01000001_00111110};// O
  2'b10:
          video_source_dots
          <= {40'b00111110_01000001_01000001_01000001_00100010, // C
            40'b00111110_01000001_01000001_01000001_00111110, // O
            40'b01111111_00000010_00001100_00000010_01111111, // M
            40'b01111111_00001001_00001001_00001001_00000110, // P
            40'b00111110_01000001_01000001_01000001_00111110, // O
            40'b00100110_01001001_01001001_01001001_00110010, // S
            40'b00000000_01000001_01111111_01000001_00000000};// I
  2'b11:
          video_source_dots
          <= {40'b00100110_01001001_01001001_01001001_00110010, // S
            40'b00001000_00001000_00001000_00001000_00001000, // -
            40'b00000111_00011000_01100000_00011000_00000111, // V
            40'b00000000_01000001_01111111_01000001_00000000, // I
            40'b01111111_01000001_01000001_01000001_00111110, // D
            40'b01111111_01001001_01001001_01001001_01000001, // E
            40'b00111110_01000001_01000001_01000001_00111110};// O
 endcase

wire [639:0] dots;
display disp (reset, clock_27mhz, disp_blank, disp_clock, disp_rs,
                disp_ce_b, disp_reset_b, disp_data_out, dots);
assign dots = { 40'b00000111_00011000_01100000_00011000_00000111, // 'V'
```

```
                        40'b00000000_00110110_00110110_00000000_00000000, // ':'
                        volume_dots,
                        40'b00000000_00000000_00000000_00000000_00000000, // ' '
                        audio_source_dots,
                        40'b00000000_00000000_00000000_00000000_00000000, // ' '
                        video_source_dots };

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

// SRAMs
assign ram0_data = 36'hZ;
assign ram0_address = 21'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 = 21'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;

// Flash ROM
assign flash_data = 16'hZ;
assign flash_address = 15'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;

// RS-232 Interface
assign rs232_txd = 1'b1;
assign rs232_rts = 1'b1;

// Buttons, Switches, and Individual LEDs
assign led = 8'hFF;

// User I/Os
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;

// Logic Analyzer
//assign analyzer3_data = 16'h0;
//assign analyzer3_clock = 1'b1;
//assign analyzer4_data = 16'h0;
//assign analyzer4_clock = 1'b1;
```

```
endmodule

////////////////////////////////////////////////////////////////////////////
//
// Switch Debounce Module
//
////////////////////////////////////////////////////////////////////////////

module debounce (reset, clock, noisy, clean);

  input reset, clock, noisy;
  output clean;

  reg [18:0] 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 == 270000)
      clean <= new;
    else
      count <= count+1;

endmodule


Psuedo code for video analyzer:

module analyzer.v(clock_27mhz, reset, vertical_pulse, horizontal_pulse, lumience, enable, x, y)

input clock_27mhz, reset, vertical_pulse, horizontal_pulse, pix_change, lumience, enable;
output [5:0] x, y;

//Internal Regs
reg [1:0] state;
reg [1:0] next;
reg [3:0] check_reg;
reg [3:0] check;

reg [10:0] horiz;
reg [10:0] vert;

reg [5:0] x, y;
reg [5:0] x_reg, y_reg;

reg threshold;

parameter main = 0;
parameter check = 0;
parameter output = 0;
parameter wait = 0;


always @ (posedge pix_change or posedge horizontal_pulse)
begin
                        horiz <= horiz + 1;

                        if (horizontal_pulse)
                                begin
```

```
                                horiz <= 0;
                                vert <= vert + 1;
                                end

end



always @ (posedge clk or posedge reset) begin
  if (reset) begin
                        //initialize everything
                        state <= main;
                        horiz <= 0;
                        vert <= 0;
                        x <= 0;
                        y <= 0;
                        check <= 0;
                        end
  else begin
                        if (vertical_pulse == 1)
                                begin
                                horiz <= 0;
                                vert <= 0;
                                end

                        state <= next;
                        x <= x_reg;
                        y <= y_reg;
                        check <= check_reg;
            end
end


always @ (state or horiz or vert or lumience)
begin
            case(state)
                        main:
                        begin
                                next = idle;
                                x_reg = x;
                                y_reg = y;
                        end
                        check:
                        begin
                                if (check >= 5)
                                        next = wait;
                                else if(lumience >= threshold)
                                begin
                                        check_reg = check + 1;
                                        next = check;
                                end
                                else next = check;

                        end
                        output:
                        begin
                                x_reg <= horiz /20 ;
                                y_reg <= vert / 10;

                        end
                        wait:
                        begin
                                if(vertical_pulse) begin
                                                        next = main;
                                                        check_reg =0;
                                                        end
                                else next = wait;
                        end
```

```
		endcase



endmodule


module audio (reset, clock_27mhz, audio_reset_b, ac97_sdata_out, ac97_sdata_in,
		ac97_synch, ac97_bit_clock, mode, volume, source,
			music, enable, filtered_data);

  input reset, clock_27mhz;
  output audio_reset_b;
  output ac97_sdata_out;
  input ac97_sdata_in;
  output ac97_synch;
  input ac97_bit_clock;
  input [1:0] mode;
  input [4:0] volume;
  input source;

  wire ready;
  wire [7:0] command_address;
  wire [15:0] command_data;
  wire command_valid;
  reg [19:0] left_out_data, right_out_data;
  wire [19:0] left_in_data, right_in_data, sine_data, square_data;

		//Added
		output [17:0] music;
		input [15:0] filtered_data;
		wire [17:0] music;
		wire [19:0] music_data;
		output enable;
		wire enable;

		assign music = music_data[19:2];

		reg [15:0] filtered_music;


  //
  // Reset controller
  //

  reg audio_reset_b;
  reg [9:0] reset_count;

		always @ (posedge clock_27mhz)
			begin
				if(reset)
					filtered_music <= 16'b0;
				else	if (enable)
					filtered_music <= filtered_data;
				else
					filtered_music <= filtered_music;
			end



  always @(posedge clock_27mhz) begin
    if (reset)
		begin
		  audio_reset_b = 1'b0;
		  reset_count = 0;
		end
    else if (reset_count == 1023)
```

```
            audio_reset_b = 1'b1;
    else
            reset_count = reset_count+1;
  end

  ac97 ac97(ready, command_address, command_data, command_valid,
              left_out_data, 1'b1, right_out_data, 1'b1, left_in_data,
              right_in_data, ac97_sdata_out, ac97_sdata_in, ac97_synch,
              ac97_bit_clock, music_data, enable);

  ac97commands cmds(clock_27mhz, ready, command_address, command_data,
                    command_valid, volume, source);

  sinewave sine(clock_27mhz, ready, sine_data);

  squarewave square(clock_27mhz, ready, square_data);

  always @(mode or left_in_data or right_in_data or sine_data or square_data)
    case (mode)
      2'd0: begin
              left_out_data = left_in_data;
              right_out_data = left_in_data;
      end
      2'd1: begin
              left_out_data = {filtered_music, 4'b0};
              right_out_data = {filtered_music, 4'b0};
      end
      2'd2: begin
                      left_out_data = {music, 2'b0};
              right_out_data = {music, 2'b0};
//            left_out_data = sine_data;
//            right_out_data = sine_data;
      end
      2'd3: begin
              left_out_data = square_data;
              right_out_data = square_data;
      end
    endcase

endmodule

module beeper (reset, clock_27mhz, beep, enable);

  input reset, clock_27mhz, enable;
  output beep;

  reg [15:0] count;
  reg clock_1khz;

  always @(posedge clock_27mhz)
    if (reset)
      begin
              count <= 0;
              clock_1khz <= 0;
      end
    else if (count == 13499)
      begin
              clock_1khz <= ~clock_1khz;
              count <= 0;
     end
    else
      count <= count+1;

  assign beep = enable && clock_1khz;

endmodule

module volume (reset, clock, up, down, vol, disp);

  input reset, clock, up, down;
```

```
  output [3:0] vol;
  output [39:0] disp;

  reg [3:0] vol;
  reg [39:0] disp;
  reg old_up, old_down;

  always @(posedge clock)
   if (reset)
     begin
               vol <= 0;
               old_up <= 0;
               old_down <= 0;
     end
   else
     begin
               if ((up == 1) && (old_up == 0) && (vol < 15))
                 vol <= vol+1;
               else if ((down == 1) && (old_down == 0) && (vol > 0))
                 vol <= vol-1;
               old_up <= up;
               old_down <= down;
     end

  always @(vol)
   case (vol[3:1])
     0: disp <= { 5{8'b00000000}};
     1: disp <= { 5{8'b01000000}};
     2: disp <= { 5{8'b01100000}};
     3: disp <= { 5{8'b01110000}};
     4: disp <= { 5{8'b01111000}};
     5: disp <= { 5{8'b01111100}};
     6: disp <= { 5{8'b01111110}};
     7: disp <= { 5{8'b01111111}};
   endcase

endmodule

module ac97 (ready,
                 command_address, command_data, command_valid,
                 left_data, left_valid,
                 right_data, right_valid,
                 left_in_data, right_in_data,
                 ac97_sdata_out, ac97_sdata_in, ac97_synch, ac97_bit_clock,
                         music_data, enable);

  output ready;
  input [7:0] command_address;
  input [15:0] command_data;
  input command_valid;
  input [19:0] left_data, right_data;
  input left_valid, right_valid;
  output [19:0] left_in_data, right_in_data;

  input ac97_sdata_in;
  input ac97_bit_clock;
  output ac97_sdata_out;
  output ac97_synch;

  reg ready;

  reg ac97_sdata_out;
  reg ac97_synch;

  reg [7:0] bit_count;

  reg [19:0] l_cmd_addr;
  reg [19:0] l_cmd_data;
  reg [19:0] l_left_data, l_right_data;
  reg l_cmd_v, l_left_v, l_right_v;
```

```
reg [19:0] left_in_data, right_in_data;

        //Added
                output [19:0] music_data;
                reg [19:0] music_data;
                output enable;
                reg enable;



initial begin
  ready <= 1'b0;
  // synthesis attribute init of ready is "0";
  ac97_sdata_out <= 1'b0;
  // synthesis attribute init of ac97_sdata_out is "0";
  ac97_synch <= 1'b0;
  // synthesis attribute init of ac97_synch is "0";

  bit_count <= 8'h00;
  // synthesis attribute init of bit_count is "0000";
  l_cmd_v <= 1'b0;
  // synthesis attribute init of l_cmd_v is "0";
  l_left_v <= 1'b0;
  // synthesis attribute init of l_left_v is "0";
  l_right_v <= 1'b0;
  // synthesis attribute init of l_right_v is "0";

  left_in_data <= 20'h00000;
  // synthesis attribute init of left_in_data is "00000";
  right_in_data <= 20'h00000;
  // synthesis attribute init of right_in_data is "00000";
end

always @(posedge ac97_bit_clock) begin
  // Generate the sync signal
  if (bit_count == 255)
        ac97_synch <= 1'b1;
  if (bit_count == 15)
        ac97_synch <= 1'b0;

  // Generate the ready signal
  if (bit_count == 128)
        ready <= 1'b1;
  if (bit_count == 2)
        ready <= 1'b0;

  // Latch user data at the end of each frame. This ensures that the
  // first frame after reset will be empty.
  if (bit_count == 255)
        begin
          l_cmd_addr <= {command_address, 12'h000};
          l_cmd_data <= {command_data, 4'h0};
          l_cmd_v <= command_valid;
          l_left_data <= left_data;
          l_left_v <= left_valid;
          l_right_data <= right_data;
          l_right_v <= right_valid;
        end

  if ((bit_count >= 0) && (bit_count <= 15))
        // Slot 0: Tags
        case (bit_count[3:0])
          4'h0: ac97_sdata_out <= 1'b1;      // Frame valid
          4'h1: ac97_sdata_out <= l_cmd_v;   // Command address valid
          4'h2: ac97_sdata_out <= l_cmd_v;   // Command data valid
          4'h3: ac97_sdata_out <= l_left_v;  // Left data valid
          4'h4: ac97_sdata_out <= l_right_v; // Right data valid
          default: ac97_sdata_out <= 1'b0;
        endcase
```

```
    else if ((bit_count >= 16) && (bit_count <= 35))
            // Slot 1: Command address (8-bits, left justified)
            ac97_sdata_out <= l_cmd_v ? l_cmd_addr[35-bit_count] : 1'b0;

    else if ((bit_count >= 36) && (bit_count <= 55))
            // Slot 2: Command data (16-bits, left justified)
            ac97_sdata_out <= l_cmd_v ? l_cmd_data[55-bit_count] : 1'b0;

    else if ((bit_count >= 56) && (bit_count <= 75))
            begin
              // Slot 3: Left channel
              ac97_sdata_out <= l_left_v ? l_left_data[19] : 1'b0;
              l_left_data <= { l_left_data[18:0], l_left_data[19] };
            end
    else if ((bit_count >= 76) && (bit_count <= 95))
            // Slot 4: Right channel
              ac97_sdata_out <= l_right_v ? l_right_data[95-bit_count] : 1'b0;
    else
            ac97_sdata_out <= 1'b0;

    bit_count <= bit_count+1;

  end // always @ (posedge ac97_bit_clock)

  always @(negedge ac97_bit_clock) begin
    if ((bit_count >= 57) && (bit_count <= 76))
            // Slot 3: Left channel
            begin
            music_data <= {music_data[18:0],ac97_sdata_in};
            left_in_data <= { left_in_data[18:0], ac97_sdata_in };
            end
   else if ((bit_count >= 77) && (bit_count <= 96))
            // Slot 4: Right channel
            //right_in_data <= { right_in_data[18:0], ac97_sdata_in };

            if (bit_count == 78)
                    begin
                    enable <= 1;
                    end

            else enable <= 0;

  end

endmodule

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

module ac97commands (clock, ready, command_address, command_data,
                                command_valid, volume, source);

  input clock;
  input ready;
  output [7:0] command_address;
  output [15:0] command_data;
  output command_valid;
  input [4:0] volume;
  input source;

  reg [23:0] command;
  reg command_valid;

  reg old_ready;
  reg done;
  reg [3:0] state;

  initial begin
    command <= 4'h0;
    // synthesis attribute init of command is "0";
    command_valid <= 1'b0;
```

```
    // synthesis attribute init of command_valid is "0";
    done <= 1'b0;
    // synthesis attribute init of done is "0";
    old_ready <= 1'b0;
    // synthesis attribute init of old_ready is "0";
    state <= 16'h0000;
    // synthesis attribute init of state is "0000";
  end

  assign command_address = command[23:16];
  assign command_data = command[15:0];

  wire [4:0] vol;
  assign vol = 31-volume;

  always @(posedge clock) begin
    if (ready && (!old_ready))
            state <= state+1;

    case (state)
            4'h0: // Read ID
              begin
                command <= 24'h80_0000;
                command_valid <= 1'b1;
              end
            4'h1: // Read ID
              command <= 24'h80_0000;
            4'h2: // Master volume
              command <= { 8'h02, 3'b000, vol, 3'b000, vol };
            4'h3: // Aux volume
              command <= { 8'h04, 3'b000, vol, 3'b000, vol };
            4'h4: // Mono volume
              command <= 24'h06_8000;
            4'h5: // PCM volume
              command <= 24'h18_0808;
            4'h6: // Record source select
              if (source)
                command <= 24'h1A_0000; // microphone
              else
                command <= 24'h1A_0404; // line-in
            4'h7: // Record gain
              command <= 24'h1C_0000;
            4'h8: // Line in gain
              command <= 24'h10_8000;
            //4'h9:
            //              command <= 24'h32_AFC8;
            //4'h9: // Set jack sense pins
              //command <= 24'h72_3F00;
            4'hA: // Set beep volume
              command <= 24'h0A_0000;
            //4'hB:
            //              command <= 24'h1C_AFC8;
            //4'hF: // Misc control bits
              //command <= 24'h76_8000;
            default:
              command <= 24'h80_0000;
    endcase // case(state)

    old_ready <= ready;

  end // always @ (posedge clock)

endmodule // ac97commands


module sinewave (clock, ready, pcm_data);

  input clock;
  input ready;
  output [19:0] pcm_data;
```

```
reg rdy, old_ready;
reg [8:0] index;
reg [19:0] pcm_data;

initial begin
  old_ready <= 1'b0;
  // synthesis attribute init of old_ready is "0";
  index <= 8'h00;
  // synthesis attribute init of index is "00";
  pcm_data <= 20'h00000;
  // synthesis attribute init of pcm_data is "00000";
end

always @(posedge clock) begin
  if (rdy && ~old_ready)
          index <= index+1;
  old_ready <= rdy;
  rdy <= ready;
end

always @(index) begin
  case (index[5:0])
    6'h00: pcm_data <= 20'h00000;
    6'h01: pcm_data <= 20'h0C8BD;
    6'h02: pcm_data <= 20'h18F8B;
    6'h03: pcm_data <= 20'h25280;
    6'h04: pcm_data <= 20'h30FBC;
    6'h05: pcm_data <= 20'h3C56B;
    6'h06: pcm_data <= 20'h471CE;
    6'h07: pcm_data <= 20'h5133C;
    6'h08: pcm_data <= 20'h5A827;
    6'h09: pcm_data <= 20'h62F20;
    6'h0A: pcm_data <= 20'h6A6D9;
    6'h0B: pcm_data <= 20'h70E2C;
    6'h0C: pcm_data <= 20'h7641A;
    6'h0D: pcm_data <= 20'h7A7D0;
    6'h0E: pcm_data <= 20'h7D8A5;
    6'h0F: pcm_data <= 20'h7F623;
    6'h10: pcm_data <= 20'h7FFFF;
    6'h11: pcm_data <= 20'h7F623;
    6'h12: pcm_data <= 20'h7D8A5;
    6'h13: pcm_data <= 20'h7A7D0;
    6'h14: pcm_data <= 20'h7641A;
    6'h15: pcm_data <= 20'h70E2C;
    6'h16: pcm_data <= 20'h6A6D9;
    6'h17: pcm_data <= 20'h62F20;
    6'h18: pcm_data <= 20'h5A827;
    6'h19: pcm_data <= 20'h5133C;
    6'h1A: pcm_data <= 20'h471CE;
    6'h1B: pcm_data <= 20'h3C56B;
    6'h1C: pcm_data <= 20'h30FBC;
    6'h1D: pcm_data <= 20'h25280;
    6'h1E: pcm_data <= 20'h18F8B;
    6'h1F: pcm_data <= 20'h0C8BD;
    6'h20: pcm_data <= 20'h00000;
    6'h21: pcm_data <= 20'hF3743;
    6'h22: pcm_data <= 20'hE7075;
    6'h23: pcm_data <= 20'hDAD80;
    6'h24: pcm_data <= 20'hCF044;
    6'h25: pcm_data <= 20'hC3A95;
    6'h26: pcm_data <= 20'hB8E32;
    6'h27: pcm_data <= 20'hAECC4;
    6'h28: pcm_data <= 20'hA57D9;
    6'h29: pcm_data <= 20'h9D0E0;
    6'h2A: pcm_data <= 20'h95927;
    6'h2B: pcm_data <= 20'h8F1D4;
    6'h2C: pcm_data <= 20'h89BE6;
    6'h2D: pcm_data <= 20'h85830;
    6'h2E: pcm_data <= 20'h8275B;
```

```
      6'h2F: pcm_data <= 20'h809DD;
      6'h30: pcm_data <= 20'h80000;
      6'h31: pcm_data <= 20'h809DD;
      6'h32: pcm_data <= 20'h8275B;
      6'h33: pcm_data <= 20'h85830;
      6'h34: pcm_data <= 20'h89BE6;
      6'h35: pcm_data <= 20'h8F1D4;
      6'h36: pcm_data <= 20'h95927;
      6'h37: pcm_data <= 20'h9D0E0;
      6'h38: pcm_data <= 20'hA57D9;
      6'h39: pcm_data <= 20'hAECC4;
      6'h3A: pcm_data <= 20'hB8E32;
      6'h3B: pcm_data <= 20'hC3A95;
      6'h3C: pcm_data <= 20'hCF044;
      6'h3D: pcm_data <= 20'hDAD80;
      6'h3E: pcm_data <= 20'hE7075;
      6'h3F: pcm_data <= 20'hF3743;
    endcase // case(index[8:2])
  end // always @ (index)

endmodule



module squarewave (clock, ready, pcm_data);

  input clock;
  input ready;
  output [19:0] pcm_data;

  reg old_ready;
  reg [6:0] index;
  reg [19:0] pcm_data;

  initial begin
    old_ready <= 1'b0;
    // synthesis attribute init of old_ready is "0";
    index <= 7'h00;
    // synthesis attribute init of index is "00";
    pcm_data <= 20'h00000;
    // synthesis attribute init of pcm_data is "00000";
  end

  always @(posedge clock) begin
    if (ready && ~old_ready)
            index <= index+1;
    old_ready <= ready;
  end

  always @(index) begin
    if (index[6])
            pcm_data <= 20'hF0F00;
    else
            pcm_data <= 20'h05555;
  end // always @ (index)

endmodule

module fake_video_input(clock_27mhz, reset, y_up, y_down, x_up, x_down, x, y);

input clock_27mhz, y_up, y_down, x_up, x_down;
input reset;
output [6:0] x,y;
reg [6:0] x;
reg [6:0] y;
reg old_xup;
reg old_yup;

reg old_xdown;
reg old_ydown;
```

```
always @ (posedge clock_27mhz or posedge reset)
begin

  if (reset)
    begin
      x <= 0;
      y <= 0;

      old_xup <= 0;
      old_yup <= 0;
      old_xdown <= 0;
      old_ydown <= 0;
    end
  else
    begin
      old_xup <= x_up;
      old_yup <= y_up;
      old_xdown <= x_down;
      old_ydown <= y_down;


      if ((x_up == 1) && (old_xup != x_up))
        begin
          if (x < 32)
            x <= x + 1;
          else
            x <= 32;
        end

      if ((y_up == 1) && (old_yup != y_up))
        begin
          if (y < 32)
            y <= y + 1;
          else
            y <= 32;
        end

      if ((x_down == 1) && (old_xdown != x_down))
        begin
          if (x > -31)
            x <= x - 1;
          else
            x <= -31;
        end

      if ((y_down == 1) && (old_ydown != y_down))
        begin
          if (y > -31)
            y <= y - 1;
          else
            y <= -31;
        end
    end
end
endmodule

module amp_change(clock_27mhz, reset, y, enable_up, enable_down);

input clock_27mhz;
input [6:0] y;
input reset;

output enable_up;
output enable_down;
reg enable_up;
reg enable_down;
```

```
reg [6:0] old_y;

always @ (posedge clock_27mhz or posedge reset)
begin
	if(reset)
		begin
		old_y <= 0;
		end
	else
		old_y <= y;

end
always @ (posedge clock_27mhz or posedge reset)
begin
	if(reset)
		begin
		enable_up <= 0;
		enable_down <= 0;
		end
	else if (y>old_y)
		begin
		enable_up <= 1;
		enable_down <= 0;
		end
	else if (y<old_y)
		begin
		enable_down <= 1;
		enable_up <= 0;
		end
	else
		begin
		enable_down <= 0;
		enable_up <= 0;
		end
end

endmodule
```

Video Out Code:

```
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;
// ac97_sdata_out and ac97_sdata_out are inputs;

/*
// 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;
// disp_data_out 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


parameter music_before_filtering = 16'h0070;                //ERASE AND GET REAL VALUES FROM CHEN
parameter music_after_filtering = 16'h0070;   // ""
//wire[15:0] music_before_filtering, music_after_filtering;
//assign music_after_filtering = music_before_filtering;
wire[6:0] ram1out, ram3out, ramout;
wire[10:0] line;
wire vblanking, hblanking;

wire[1:0] switches;
wire[7:0] hvga_out_green;
wire linef,verticalf,background,vga_out_vsync, vga_out_hsync;
wire[9:0] ram1add, ram3add;

wire ram1on, ram2on, ram3on;

wire[1:0] delta_x; //from chen
wire wrtenables; //for ram1 and ram3
wire[199:0] filterout;
wire[11:0] location; //from Chen  NOT CURRENTLY ASSIGNED TO ANYTHING

//DCM code borrowed mostly from Nathan.
DCM vga_dcm(.CLKIN(clock_27mhz), .RST(1'b0), .CLKFX(pixel_clock));
  // synthesis attribute DLL_FREQUENCY_MODE of vga_dcm is "LOW"
  // synthesis attribute DUTY_CYCLE_CORRECTION of vga_dcm is "TRUE"
  // synthesis attribute STARTUP_WAIT of vga_dcm is "TRUE"
  // synthesis attribute DFS_FREQUENCY_MODE of vga_dcm is "LOW"
  // synthesis attribute CLKFX_DIVIDE of vga_dcm is 7
  // synthesis attribute CLKFX_MULTIPLY of vga_dcm is 13
  // synthesis attribute CLK_FEEDBACK of vga_dcm  is "1X"
  // synthesis attribute CLKOUT_PHASE_SHIFT of vga_dcm is "NONE"
  // synthesis attribute PHASE_SHIFT of vga_dcm is 0
  // synthesis attribute CLKIN_PERIOD of vga_dcm is "37.04ns"

assign vga_out_pixel_clock = ~pixel_clock;

//genmusic gm(pixel_clock,reset,music_before_filtering);
locationgen lg(pixel_clock, reset, location);
ram ra1(ram1add, pixel_clock, music_before_filtering[15:9],ram1out,wrtenables);
ram r3(ram3add, pixel_clock, music_after_filtering[15:9],ram3out, wrtenables);
vcontrols vc(reset, linef, vblanking, vga_out_vsync,verticalf,line);
version2 v1(pixel_clock,reset,hblanking,vga_out_hsync,linef);
genvideo gv(pixel_clock, reset, line, linef,vga_out_red,hvga_out_green,vga_out_blue,
            ramout,background,filterout,ram1on,ram2on,ram3on);
ramcontrol rc(pixel_clock, reset, verticalf,linef,ram1add,ram3add,ram1on,ram2on,
            ram3on,background,   wrtenables,record,switches,location,delta_x,filterout);
pickram pr(pixel_clock,reset,ram1on,ram2on,ram3on,ram1out,ram3out,ramout);


assign user1 = 32'bZ;
assign reset = !button0;
assign record = !button1;
assign switches[1:0] = switch[1:0];
assign delta_x = {!button3, !button2};                      //from Chen

assign vga_out_sync_b = (vga_out_vsync + vga_out_hsync); //low when either vsync xor hsync low.
assign vga_out_blank_b = (hblanking && vblanking);  //low if either blanking is low.
assign vga_out_green = (!vga_out_vsync && !vga_out_hsync) ? hvga_out_green : 8'h00;

endmodule



module genvideo(pclk, reset, line,linef,r,g,b, ram,background,filter,ram1on,ram2on,ram3on);
input pclk, reset,linef,background,ram1on, ram2on,ram3on;
input[6:0] ram;
input[10:0] line;
input[199:0] filter;
```

```
output[7:0] r,g,b;

reg[199:0] shiftfilter;
reg[7:0] r,g,b;
reg[6:0] value;
reg[1:0] holdcnt;
reg[10:0] pcnt;
reg state1, state2;

parameter[9:0] hactvideo = 10; parameter[6:0] hfrontporch = 3;
parameter[7:0] hsyncon = 5; parameter[8:0] hbackporch = 5;
parameter waiting1 = 0; parameter waiting2 = 0;
parameter display1 = 1; parameter display2 = 1;


always @ (posedge pclk) begin
if (reset)
        begin
        r <= 8'h00;
        g <= 8'hCE;
        b <= 8'hD1;
        value <= 7'hFF;
        holdcnt <= 0;
        shiftfilter <= 200'd0;
        state1 <= 0;
        state2 <= 0;
        end

else if (ram1on || ram3on)
begin
        value <= ((line == 49) || (line == 249) || (line == 449)) ? value <= 7'b1111111 :
        linef ? value - 1 : value;
case (state1)
waiting1:
        begin
        state1 <= linef ? display1 : waiting1;
        pcnt <= 11'd0;
        end
display1:
        begin
        state1 <= (pcnt == 800) ? waiting1 : display1;
        pcnt <= pcnt + 1;
        if (ram == value)
                begin
                r <= 8'h10;
                g <= 8'h10;
                b <= 8'h10;
                end
        else
                begin
                r <= 8'h11;
                g <= 8'hCE;
                b <= 8'hD1;
                end
        end
endcase
end

else if (ram2on)
begin
case (state2)
waiting2:
        begin
        state2 <= linef ? display2 : waiting2;
        pcnt <= 11'd0;
        shiftfilter <= {filter[0],filter[199:1]};
        end
display2:
begin
```

```
		pcnt <= pcnt + 1;
		state2 <= (pcnt == 799) ? waiting2 : display2;
		holdcnt <= holdcnt + 1;
		shiftfilter <=  (holdcnt == 0) ? {shiftfilter[198:0], shiftfilter[199]} : shiftfilter;

		if ((shiftfilter[199] == 1) && (line == 249))
			begin
			r <= 8'h10;
			g <= 8'h10;
			b <= 8'h10;
			end
		else if ((shiftfilter[199] == 0) && (line == 376))
			begin
			r <= 8'h10;
			g <= 8'h10;
			b <= 8'h10;
			end
		else if ((shiftfilter[199] + shiftfilter[198]) == 1)
			begin
			r <= 8'h10;
			g <= 8'h10;
			b <= 8'h10;
			end
		else
			begin
			r <= 8'h00;
			g <= 8'hCE;
			b <= 8'hD1;
			end

end
endcase
end

else			begin
			r <= 8'h11;
			g <= 8'hCE;
			b <= 8'hD1;
			end



end
endmodule



module pickram(pclk, reset, ram1on, ram2on, ram3on, ram1out,ram3out, ramout);
input pclk, reset, ram1on, ram2on, ram3on;
input[6:0] ram1out, ram3out;
output[6:0] ramout;
reg[6:0] ramout;

always @ (posedge pclk)
if (reset)
		ramout <= 7'd0;
else
begin
if (ram1on == 1)
		ramout <= ram1out;
else if (ram2on == 1)
		ramout <= 7'd0;
else if (ram3on == 1)
		ramout <= ram3out;
else
		ramout <= 7'd0;
end

endmodule
```

```
module locationgen(pclk, reset, location);
input pclk, reset;
output[10:0] location;
reg[11:0] location;
reg[25:0] cnt;
reg[2:0] int;

always @ (posedge pclk)
begin
if (reset) location <= 12'd0;
else
begin
cnt <= cnt + 1;
location <= {int, 8'd0};
int <= (cnt == 0) ? int + 1 : int;
end
end
endmodule
```



```
// The synopsys directives "translate_off/translate_on" specified below are
// supported by XST, FPGA Compiler II, Mentor Graphics and Synplicity synthesis
// tools. Ensure they are correct for your synthesis tool(s).

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

module ram (
        addr,
        clk,
        din,
        dout,
        we);    // synthesis black_box

input [9 : 0] addr;
input clk;
input [6 : 0] din;
```

```
output [6 : 0] dout;
input we;

// synopsys translate_off

        BLKMEMSP_V6_1 #(
                10,             // c_addr_width
                "0",            // c_default_data
                800,            // c_depth
                0,              // c_enable_rlocs
                1,              // c_has_default_data
                1,              // c_has_din
                0,              // c_has_en
                0,              // c_has_limit_data_pitch
                0,              // c_has_nd
                0,              // c_has_rdy
                0,              // c_has_rfd
                0,              // c_has_sinit
                1,              // c_has_we
                18,             // c_limit_data_pitch
                "mif_file_16_1",        // c_mem_init_file
                0,              // c_pipe_stages
                0,              // c_reg_inputs
                "0",            // c_sinit_value
                7,              // c_width
                0,              // c_write_mode
                "0",            // c_ybottom_addr
                1,              // c_yclk_is_rising
                1,              // c_yen_is_high
                "hierarchy1",           // c_yhierarchy
                0,              // c_ymake_bmm
                "16kx1",        // c_yprimitive_type
                1,              // c_ysinit_is_high
                "1024",         // c_ytop_addr
                0,              // c_yuse_single_primitive
                0,              // c_ywe_is_high
                1)              // c_yydisable_warnings
        inst (
                .ADDR(addr),
                .CLK(clk),
                .DIN(din),
                .DOUT(dout),
                .WE(we),
                .EN(),
                .ND(),
                .RFD(),
                .RDY(),
                .SINIT());


// synopsys translate_on

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

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

endmodule




module ramcontrol(pclk, reset, verticalf, linef, ram1add,
ram3add, ram1on, ram2on, ram3on,background,wrtenables,record,
switches,location,delta_x, filterout); //module currently only reads
```

```
input pclk, reset, verticalf, linef,record;
input[1:0] switches, delta_x;
input[11:0] location;

output[199:0] filterout;
output[9:0] ram1add, ram3add;
output ram1on, ram2on, ram3on,background,wrtenables;

reg[199:0] filterout;
reg ram1on, ram2on, ram3on, background;
reg[9:0] ram1add, ram3add,ram1addstored,ram3addstored;
reg[9:0] cnt;
reg[2:0] state;
reg wrtenables;
reg[1:0] switchesold;

parameter waitingforline = 3'b000; parameter blanking1 = 3'b001;
parameter displayram1 = 3'b010; parameter blanking2 = 3'b011;
parameter displayram2 = 3'b100; parameter blanking3 = 3'b101;
parameter displayram3 = 3'b110; parameter turnoffwrtenables = 3'b111;

always @ (posedge pclk) begin
if (reset)
        begin
        cnt <= 10'd0;
        ram1add <= 10'd0;
        ram3add <= 10'd0;
        ram1addstored <= 10'd0;
        ram3addstored <= 10'd0;
        state <= turnoffwrtenables;
        ram1on <= 0;
        ram2on <= 0;
        ram3on <= 0;
        background <= 0;
        wrtenables <= 1; //not write enable
        end

else
begin
ram1addstored <= (record == 1) ? 10'd0 : ram1addstored;
ram3addstored <= (record == 1) ? 10'd0 : ram3addstored;
ram1on <= 0;
ram2on <= 0;
ram3on <= 0;
cnt[9:0] <= (linef == 1) ? cnt + 1 : cnt;
background <= 0;
wrtenables <= 1; //not write enable

case (state)
turnoffwrtenables: state <= waitingforline;
waitingforline:
        begin
        ram1add <= 10'd0;
        ram3add <= 10'd0;
        cnt <= 10'd0;
        state <= (verticalf == 1) ? blanking1 : waitingforline;
        end
blanking1:
        begin
        state <= (cnt == 49) ? displayram1 : blanking1;
        background <= 1;
        if ((ram1addstored !== 799) && (cnt == 45))
                begin
                wrtenables <= 0;
                ram1add <= ram1addstored + 1;
                ram1addstored <= ram1addstored + 1;
                ram3add <= ram3addstored + 1;
                ram3addstored <= ram3addstored + 1;
        //      wrtenables <= 1;//not write enable
                end
```

```
//              else if ((ram1addstored !== 800) && (cnt == 46)) wrtenables <= 0; // NOT write enable
                else wrtenables <= 1;
                end
displayram1:
                begin
                ram1on <= 1;
                state <= (cnt == 177) ? blanking2 : displayram1;
                if (ram1add == 799) ram1add <= linef ? 10'd0 : ram1add;
                else ram1add <= ram1add + 1;
                end
blanking2:
                begin
                state <= (cnt == 247) ? displayram2 : blanking2;
                background <= 1;
                end
displayram2:
                begin
                switchesold <= switches;
                ram2on <= 1;
                state <= (cnt == 377) ? blanking3 : displayram2;
                case (switches)
                2'b11:   //bandpass
                                begin
                                if (location[11:9] == 3'b000) filterout <= 200'hFFFFFFFFFFFFFFF000000000000000000000000000000000000;
                                else if (location[11:9] == 3'b001) filterout <=
200'h00000FFFFFFFFFFFFFFF000000000000000000000000000000000;
                                else if (location[11:9] == 3'b010) filterout <=
200'h0000000000FFFFFFFFFFFFFF000000000000000000000000000;
                                else if (location[11:9] == 3'b011) filterout <=
200'h00000000000000FFFFFFFFFFFFFFFF0000000000000000000;
                                else if (location[11:9] == 3'b100) filterout <=
200'h000000000000000000000FFFFFFFFFFFFFFF000000000000000;
                                else if (location[11:9] == 3'b101) filterout <=
200'h0000000000000000000000000000FFFFFFFFFFFFFFFF00000000;
                                else if (location[11:9] == 3'b110) filterout <=
200'h00000000000000000000000000000000FFFFFFFFFFFFFFFF000;
                                else if (location[11:9] == 3'b111) filterout <=
200'h000000000000000000000000000000000000FFFFFFFFFFFFFFF;
                                else          filterout <= 200'h000000000000FFFFFFFFFFFFFF00000000000000;
                                end
                2'b01:       // low pass
                                begin
                                if (switchesold !== 2'b01)
                                                filterout <= 200'hFFFFFFFFFFFFFF000000000000000000000000000000000000;
                                else if (delta_x[0] && (cnt == 248))
                                                filterout <= {12'hFFF, filterout[199:12]}; //left to right, so expanding
                                else if (delta_x[1] && (cnt == 248))
                                                filterout <= {filterout[187:0], 12'd0};              //high diminishing
                                else filterout <= filterout;
                                end
                2'b10:       // high pass
                                begin
                                if (switchesold !== 2'b10)
                                                filterout <= 200'h000000000000000000000000000000000000000000FFFFFFFFF;
                                else if (delta_x[1] && (cnt == 248))
                                                filterout <= {12'h000, filterout[199:12]}; //left to right, so contracting
                                else if (delta_x[0] && (cnt == 248))
                                                filterout <= {filterout[187:0], 12'hFFF};          //high expanding
                                else filterout <= filterout;
                                end
                2'b00:       filterout <= 200'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF;
                default: filterout <= 200'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF;
                endcase
                end

blanking3:
                begin
                state <= (cnt == 449) ? displayram3 : blanking3;
                background <= 1;
                end
```

```
displayram3:
		begin
		ram3on <= 1;
		state <= (cnt == 577) ? turnoffwrtenables : displayram3;
		if (ram3add == 799) ram3add <= linef ? 10'd0 : ram3add;
		else ram3add <= ram3add + 1;
		end
endcase
end
end
endmodule



module vcontrols(reset, line, vblanking, vsync, verticalf,vcnt);
input reset, line;
output vblanking, vsync, verticalf;
output[10:0] vcnt;

reg vblanking, vsync,verticalf;
reg[10:0] vcnt;
reg[1:0] state;

parameter[9:0] vactvideo = 600; parameter[4:0] vfrontporch = 37;
parameter[4:0] vsyncon = 6; parameter[4:0] vbackporch = 23;
parameter[1:0] active = 2'b00; parameter[1:0] frontporch = 2'b01;
parameter[1:0] vsyncing = 2'b10; parameter[1:0] backporch = 2'b11;

always @ (posedge reset or posedge line) begin
if (reset)
		begin
		vblanking <= 1;
		vcnt <= 10'd0;
		vsync <= 1;
		state <= 2'b00;
		verticalf <= 0;
		end
else

begin
vblanking <= 1;
vcnt <= vcnt + 1;
vsync <= 1;
verticalf <= 0;

case (state)
active:
		begin
		vblanking <= 1;
		state <= (vcnt == vactvideo) ? frontporch : active;
		end

frontporch:
		begin
		vblanking <= 0;
		state <= (vcnt == (vactvideo + vfrontporch + 2)) ? vsyncing : frontporch;
		end

vsyncing:
		begin
		vsync <= 0;
		vblanking <= 0;
		state <= (vcnt == (vactvideo + vfrontporch + 2 + vsyncon)) ? backporch : vsyncing;
		end

backporch:
		begin
		vblanking <= 0;
		if (vcnt == (vactvideo + vfrontporch + vsyncon + vbackporch))
```

```
                        begin
                        vcnt <= 10'b0;
                        state <= active;
                        verticalf <= 1;
                        end
                else state <= backporch;
                end
endcase
end
end
endmodule



module version2(pclk,reset,blanking,hsync,linef);
input pclk, reset;
output blanking,hsync, linef;

reg blanking, hsync, linef,lines;
reg[9:0] hcnt;
reg[2:0] state;

parameter[9:0] hactvideo = 800; parameter[6:0] hfrontporch = 56;
parameter[7:0] hsyncon = 120; parameter[8:0] hbackporch = 64;
parameter[1:0] hav = 2'b00;parameter[1:0] hfp = 2'b01;
parameter[1:0] hsp = 2'b10;parameter[1:0] hbp = 2'b11;

always @ (posedge pclk) begin
if (reset)
                begin
                state <= 2'b0;
                hsync <= 0;
                hcnt <= 10'd0;
                linef <= 0;
                end
else
begin
blanking <= 1;
hsync <= 1;
linef <= 0;

case (state)
hav:
                begin
                hcnt <= hcnt + 1;
                state <= (hcnt == hactvideo) ? hfp : hav;
                end
hfp:
                begin
                blanking <= 0;
                hcnt <= hcnt + 1;
                state <= (hcnt == (hactvideo + hfrontporch + 2)) ? hsp : hfp;
                end
hsp:
                begin
                blanking <= 0;
                hsync <= 0;
                hcnt <= hcnt + 1;
                state <= (hcnt == (hactvideo + hfrontporch + 2 + hsyncon)) ? hbp : hsp;
                end
hbp:
                begin
                blanking <= 0;
                if (hcnt ==(hactvideo + hfrontporch + hsyncon + hbackporch))
                        begin
                        hcnt <= 10'd0;
                        state <= hav;
                        linef <= 1;
                        end
                else
```

```
                begin
                hcnt <= hcnt + 1;
                state <= hbp;
                end
        end

endcase
end
end
endmodule
```

FFT code:

```
module write_rom_filter_final (clock, reset, xy_enable, syn_switch, delta_x, wea, ena, dina, addra);

input clock, reset;
input xy_enable;
input [1:0] syn_switch, delta_x;

output wea, ena;
output [11:0] addra;
output [3:0] dina;

//for debug
//output write_ena;
//output cam_count;

reg wea, ena;
reg [11:0] addra;
reg [3:0] dina;

reg [12:0] write_count;
reg [7:0] cam_count;
reg write_ena;


always @ (posedge clock) begin

if (reset) begin ena <= 0; addra <= 4095;
                        dina <= 0;
                        write_count <= 0;
                        wea <= 1;
                        write_ena <= 0;
                        cam_count <= 255; end

else if (write_ena) begin

                if (xy_enable) begin
                                cam_count <= cam_count + 1;
                                wea <= 1;
                                addra <= addra; end //addra <= addra - 1;
                else begin if (cam_count < 128) ena <= 1;
                                        else ena <= 0;
                                if (cam_count < 129) begin
                                                cam_count <= cam_count + 1;
                                                wea <= 1;
                                        if (delta_x == 2'b01) begin
                                                if (addra < 4095 && ena)
                                                                begin addra <= addra + 1; end
                                                                else begin addra <= addra; end
                                                        end
                                        else if (delta_x == 2'b10)
                                                        begin if (addra > 0 && ena)
                                                                        begin addra <= addra - 1; end
                                                                        else begin addra <= addra; end
                                                        end
                                        else begin addra <= addra; end
```

```
                                                    end
                                        else begin cam_count <= 255;
                                                    wea <= 0; end
            end

if (syn_switch == 2'b00) dina <= 4'b1111;

else if (syn_switch == 2'b01) begin
                        if (delta_x == 2'b01) dina <= 4'b1111;
                        else if (delta_x == 2'b10) dina <= 0;
                        else dina <= dina; end

else if (syn_switch == 2'b10) begin
                        if (delta_x == 2'b01) dina <= 0;

                        else if (delta_x == 2'b10) dina <= 4'b1111;
                        else dina <= dina; end
else dina <= dina;
end


else begin
                        if (write_count < 512) begin
                                        if (syn_switch == 2'b01) dina <= 4'b1111;
                                        else if (syn_switch == 2'b10) dina <= 0;

                                        else dina <= 4'b1111;
                        ena <= 1; write_count <= write_count + 1; addra <= addra + 1; wea <= 1; end

            else if (write_count < 4096) begin
                                        if (syn_switch == 2'b01) dina <= 0;
                                        else if (syn_switch == 2'b10) dina <= 4'b1111;

                                        else dina <= 4'b1111;

                        ena <= 1;
                        write_count <= write_count + 1;
                        addra <= addra + 1; wea <= 1; end
            else if (write_count == 4096) begin
                                    ena <= 0;
                                    write_count <= write_count + 1;
                                    addra <= 511; wea <= 0;
                                    write_ena <= 1; end
            end



end
endmodule


module Synchronizer(clock, button_enter, switch, reset, syn_switch);

input clock, button_enter;
input [1:0] switch;

output reset;
output [1:0] syn_switch;

reg reset;
reg [1:0] syn_switch;

reg a;
reg [1:0] b;

always @ (posedge clock) begin
   a <= ~button_enter;
   reset <= a;

   b <= switch;
```

```
   syn_switch <= b;

  end

endmodule



module read_rom_filter_final (clock, reset, wea, edone, enb, addrb);

input clock, reset;
input edone;
input wea;

output enb;
output [11:0] addrb;

//for debug
//output read_en;

reg enb;
reg [11:0] addrb;
reg read_en;

always @ (posedge clock) begin

if (reset) begin enb <= 0; addrb <= 4095; read_en <= 0; end  //16384
else if (read_en) begin enb <= 1; addrb <= addrb + 1; end
else begin enb <= 0; addrb <= addrb; end

if (wea) begin enb <= 0; addrb <= 4095; read_en <= 0; end
else if (edone) begin read_en <= 1; enb <= 1; addrb <= 4095; end
else if (addrb == 4094 && read_en) read_en <= 0;
else read_en <= read_en;

end
endmodule



module mac_final (clock, reset, xk_re, xk_im, xk_index, done, doutb, syn_switch, delta_x, delta_y, xy_enable,
                                        y_enable, inv_xn_re, inv_xn_im, inv_start, inv_nfft_we, inv_fwd_inv_we, inv_scale_sch_we,
                                        inv_fwd_inv);

input clock, reset;
input done;
input [15:0] xk_re;
input [15:0] xk_im;
input [11:0] xk_index;

input [3:0] doutb;

input [1:0] syn_switch;
input [1:0] delta_x, delta_y;
input xy_enable, y_enable;

//for debug
//output [11:0] a, b, band_low, band_high, b_low, b_high;

output [15:0] inv_xn_re;
output [15:0] inv_xn_im;  //disregard later
output inv_start;
output inv_nfft_we;
output inv_fwd_inv_we;
output inv_scale_sch_we;
output inv_fwd_inv;

reg [15:0] inv_xn_re;
reg [15:0] inv_xn_im;  //disregard later
reg inv_start;
```

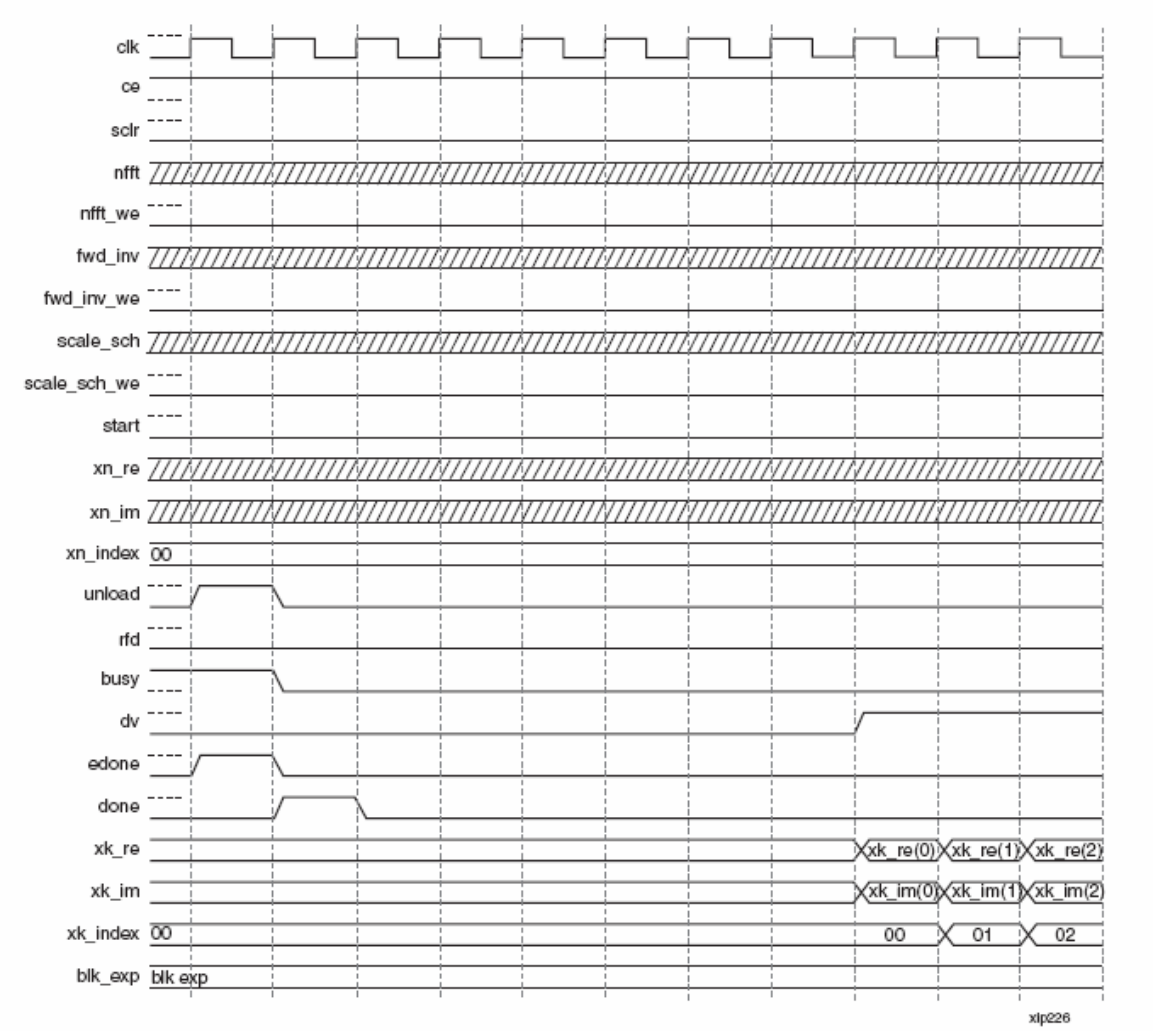
```
reg inv_nfft_we;
reg inv_fwd_inv_we;
reg inv_scale_sch_we;
reg inv_fwd_inv;

reg [15:0] extended_doutb;

reg [11:0] a, b;
reg [11:0] b_high, b_low;
reg [11:0] band_low, band_high;


always @ (posedge clock) begin

if (reset) begin inv_nfft_we <= 1;
                        inv_fwd_inv <= 0;
                        inv_fwd_inv_we <= 0;
                        inv_scale_sch_we <= 0;
                        inv_start <= 0; end
else if (inv_nfft_we) begin inv_nfft_we <= 0;
                                                inv_fwd_inv <= 1;
                                                inv_fwd_inv_we <= 1;
                                                inv_scale_sch_we <= 1;
                                                inv_start <= 1; end
else begin inv_nfft_we <= inv_nfft_we;
                                inv_fwd_inv_we <= 0;
                                inv_scale_sch_we <= 0;
                                inv_start <= inv_start; end

//assume synchronized edone most of the time
if (syn_switch != 2'b11) begin

        if (done) begin inv_xn_re <= xk_re & extended_doutb;
        inv_xn_im <= xk_im & extended_doutb; end
        else if (xk_index < 4095) begin inv_xn_re <= xk_re & extended_doutb;
        inv_xn_im <= xk_im & extended_doutb; end //
        else begin inv_xn_re <= 0;
        inv_xn_im <= 0; end

        extended_doutb <= {{12{doutb[3]}}, doutb[3:0]};
                end

 else begin
        if (done) begin band_low <= a;
                        band_high <= b; end

        if (xk_index < band_low) begin inv_xn_re <= 0;
                                                        inv_xn_im <= 0; end
        else if (xk_index < band_high) begin inv_xn_re <= xk_re ;
                                                        inv_xn_im <= xk_im; end //
        else if (xk_index < 4095) begin inv_xn_re <= 0;
                                                        inv_xn_im <= 0; end
        else begin inv_xn_re <= 0; inv_xn_im <= 0; end
end


if (reset) begin a <= 511; b <= 575; end   //a <= 511; b <= 575;
else

if (xy_enable) begin

        if (!y_enable) begin
                if (delta_x == 2'b01) begin if (b < 4032) begin a <= a + 64; b <= b + 64; end
                                                                                        else begin a <= 4031; b <= 4095;
end
                                                                end
                else if (delta_x == 2'b10) begin if (a > 64) begin a <= a - 64; b <= b - 64; end
                                                                                        else begin a <= 0; b <= 63; end
                                                                end
                else begin a <= a; b <= b; end       //b_low <= a + 4; b_high <= a + 484; end
```

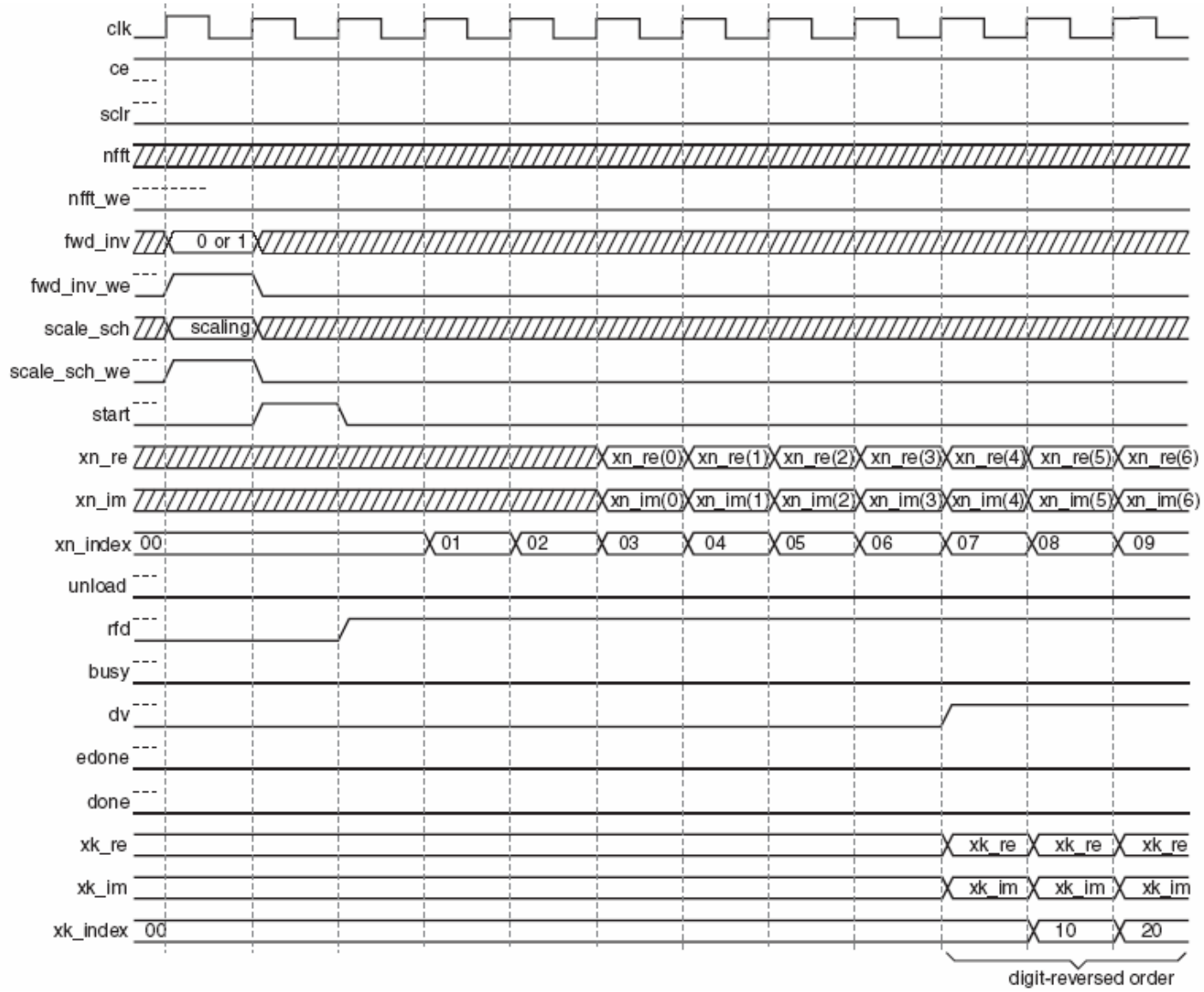
```
                end
        else begin b_low <= a + 4; b_high <= a + 484;
                        if (delta_y == 2'b01) begin if (b < b_high) begin b <= b + 28; end
                                                                        else begin b <= b; end
                                                                end
                else if (delta_y == 2'b10) begin if (b > b_low) begin b <= b - 4; end
                                                                        else begin b <= b; end
                                                                end
                else begin b <= b; end
        end
end
else begin a <= a; b <= b; end

end
endmodule


module control_rom_audio_final (clock, fwd_inv_we, enable_audio, data_audio_in,
                                                        wea_audio, ena_audio, dina_audio, addra_audio,
                                                                enb_audio, addrb_audio);

input clock, fwd_inv_we;
input enable_audio;
input [17:0] data_audio_in;

output [15:0] dina_audio;
output wea_audio, ena_audio, enb_audio;
output [11:0] addra_audio, addrb_audio;

//for debug
//output write_done;
//output counter;

reg [15:0] data_audio;
reg [15:0] dina_audio;
reg wea_audio, ena_audio, enb_audio;
reg [11:0] addra_audio, addrb_audio;

reg [1:0] write_done;
reg [12:0] counter;

always @ (posedge clock) begin

if (fwd_inv_we) begin wea_audio <= 0;
                ena_audio<= 0; enb_audio <= 0;
                addra_audio <= 0;
                addrb_audio <= 0;
                write_done <= 3; end
else if (enable_audio) begin ena_audio <= 0;
                wea_audio <= 1;
                write_done <= write_done + 1;
                addra_audio <= addra_audio + 1; end
else if (write_done < 2) begin write_done <= write_done + 1;
                                                                wea_audio <= 1;
                                                                dina_audio <= data_audio;
                                                        if (write_done == 0) ena_audio <= 1;
                                                        else ena_audio <= 0;
                                                end
else begin addra_audio <= addra_audio;
                dina_audio <= data_audio;
                wea_audio <= 0; ena_audio <= 0;
                write_done <= 3; end

if (enable_audio) enb_audio <= 0;
else if (write_done < 2) enb_audio <= 0;
else enb_audio <= 1;

if (enb_audio) begin counter <= 0;
        if (counter < 4096) begin addrb_audio <= addrb_audio + 1;
        counter <= counter +1; end //4096
```

```
        else addrb_audio <= addra_audio +1;
end
else addrb_audio <= addrb_audio;

data_audio <= {data_audio_in[17:2]};

end
endmodule


module control_fft_forward (clock, reset, nfft_we, fwd_inv, fwd_inv_we, scale_sch_we, start);

input clock, reset;
output nfft_we, fwd_inv, fwd_inv_we, scale_sch_we, start;

reg nfft_we, fwd_inv, fwd_inv_we, scale_sch_we, start;

always @ (posedge clock)
if (reset) begin nfft_we <= 1;
                                    fwd_inv <= 0;
                                    fwd_inv_we <= 0;
                                    scale_sch_we <= 0;
                                    start <= 0; end

else if (nfft_we) begin nfft_we <= 0;
                                                fwd_inv <= 1;
                                                fwd_inv_we <= 1;
                                                scale_sch_we <= 1;
                                                start <= 1; end

else begin nfft_we <= nfft_we;
                                    fwd_inv_we <= 0;
                                    scale_sch_we <= 0;
                                    start <= start; end

endmodule


module camera_enable (clk, reset, xy_enable);
input clk, reset;
output xy_enable;

reg [21:0] count;
reg xy_enable;

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

        else if (count < 40000)  //should be at least 40000
                begin
                   xy_enable <= 0;
            count <= count + 1;
                end
        else begin
                                      count <= 0;
                                    xy_enable <= 1;
                end
end
endmodule

module buffer (clock, xy_enable, x, delta_x, y, delta_y);

input clock, xy_enable;
input [5:0] x;
input [4:0] y;
output [1:0] delta_x, delta_y;

//for debug
```

```
//output [6:0] change_x;

reg [6:0] change_x;
reg [5:0] change_y;
reg [1:0] delta_x, delta_y;

reg [5:0] x_earlier, x_later, x_earlier_inv;
reg [5:0] x_temp;

reg [4:0] y_earlier, y_later, y_earlier_inv;
reg [4:0] y_temp;

always @ (posedge clock) begin

x_temp = x;
x_earlier_inv = ~x_earlier + 1;

if (xy_enable) begin change_x = {x_later[5], x_later} + {x_earlier_inv[5], x_earlier_inv};
                                                                x_earlier <= x_temp;
                                                                if (change_x == 0) begin delta_x <= 2'b00; end
                                                                else if (change_x[6]) begin delta_x <= 2'b10; end
                                                                else begin delta_x <= 2'b01; end
                                        end
else begin x_earlier <= x_earlier; x_later <= x_temp; end


y_temp = y;
y_earlier_inv = ~y_earlier + 1;

if (xy_enable) begin change_y = {y_later[4], y_later} + {y_earlier_inv[4], y_earlier_inv};
                                                                y_earlier <= y_temp;
                                                                if (change_y == 0) begin delta_y <= 2'b00; end
                                                                else if (change_y[5]) begin delta_y <= 2'b10; end
                                                                else begin delta_y <= 2'b01; end
                                        end
else begin y_earlier <= y_earlier; y_later <= y_temp; end


end
endmodule
```

FFT Diagrams:


clk
ce
sclr
nfft
nfft_we
fwd_inv
fwd_inv_we
scale_sch
scale_sch_we
start
xn_re
xn_im
xn_index
00
unload
rfd
busy
dv
edone
done
xk_re
xk_re(0)
xk_re(1)
xk_re(2)
xk_im
xk_im(0)
xk_im(1)
xk_im(2)
xk_index
00
00
01
02
blk_exp
blk exp
xlp226


clk
ce
sclr
nfft
nfft_we
fwd_inv
0 or 1
fwd_inv_we
scale_sch
scaling
scale_sch_we
start
xn_re
xn_re(0)
xn_re(1)
xn_re(2)
xn_re(3)
xn_re(4)
xn_re(5)
xn_re(6)
xn_im
xn_im(0)
xn_im(1)
xn_im(2)
xn_im(3)
xn_im(4)
xn_im(5)
xn_im(6)
xn_index
00
01
02
03
04
05
06
07
08
09
unload
rfd
busy
dv
edone
done
xk_re
xk_re
xk_re
xk_re
xk_im
xk_im
xk_im
xk_im
xk_index
00
10
20
digit-reversed order