VGA Display Code

```
/*****************************************************************************
*     This file is owned and controlled by Xilinx and must be used           *
*     solely for design, simulation, implementation and creation of          *
*     design files limited to Xilinx devices or technologies. Use            *
*     with non-Xilinx devices or technologies is expressly prohibited        *
*     and immediately terminates your license.                               *
*                                                                            *
*     XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS"          *
*     SOLELY FOR USE IN DEVELOPING PROGRAMS AND SOLUTIONS FOR                *
*     XILINX DEVICES.  BY PROVIDING THIS DESIGN, CODE, OR INFORMATION        *
*     AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE, APPLICATION            *
*     OR STANDARD, XILINX IS MAKING NO REPRESENTATION THAT THIS              *
*     IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT,                *
*     AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE       *
*     FOR YOUR IMPLEMENTATION.  XILINX EXPRESSLY DISCLAIMS ANY               *
*     WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE                *
*     IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR         *
*     REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF        *
*     INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS        *
*     FOR A PARTICULAR PURPOSE.                                              *
*                                                                            *
*     Xilinx products are not intended for use in life support               *
*     appliances, devices, or systems. Use in such applications are          *
*     expressly prohibited.                                                  *
*                                                                            *
*     (c) Copyright 1995-2004 Xilinx, Inc.                                   *
*     All rights reserved.                                                   *
*****************************************************************************/
// The synopsys directives "translate_off/translate_on" specified below are
// supported by XST, FPGA Compiler II, Mentor Graphics and Synplicity synthesis
// tools. Ensure they are correct for your synthesis tool(s).

// You must compile the wrapper file addressrom.v when simulating
// the core, addressrom. 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 addressrom (
       addr,
       clk,
       dout);    // synthesis black_box

input [10 : 0] addr;
input clk;
output [7 : 0] dout;

// synopsys translate_off

       BLKMEMSP_V6_1 #(
              11,    // c_addr_width
              "0",   // c_default_data
              2015,  // c_depth
              0,     // c_enable_rlocs
              0,     // c_has_default_data
              0,     // c_has_din
              0,     // c_has_en
              0,     // c_has_limit_data_pitch
              0,     // c_has_nd
              0,     // c_has_rdy
              0,     // c_has_rfd
              0,     // c_has_sinit
              0,     // c_has_we
              18,    // c_limit_data_pitch
              "addressrom.mif",    // c_mem_init_file
              0,     // c_pipe_stages
              0,     // c_reg_inputs
              "0",   // c_sinit_value
              8,     // c_width
              0,     // c_write_mode
```

```
                "0",   // c_ybottom_addr
                1,     // c_yclk_is_rising
                1,     // c_yen_is_high
                "hierarchy1",        // c_yhierarchy
                0,     // c_ymake_bmm
                "16kx1",       // c_yprimitive_type
                1,     // c_ysinit_is_high
                "1024",        // c_ytop_addr
                0,     // c_yuse_single_primitive
                1,     // c_ywe_is_high
                1)     // c_yydisable_warnings
        inst (
                .ADDR(addr),
                .CLK(clk),
                .DOUT(dout),
                .DIN(),
                .EN(),
                .ND(),
                .RFD(),
                .RDY(),
                .SINIT(),
                .WE());


// synopsys translate_on

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

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

endmodule


////////////////////////////////
////       Check MP3 Decoder       //
///           Alma E. Rico         //
////////////////////////////////


//CHANGED RECEIVE ACK TO HIGH

module checkFSM(clk, reset, start,
                      SDA, SCL, done, check_cond, state);

input clk, reset, start;
output SDA, SCL;
output done;
output check_cond;
output [5:0] state;//ADDED


parameter IDLE=0;
parameter START_WRITE=1;
parameter LISTEN_BYTE0=2;
parameter LISTEN_BYTE1=3;
parameter LISTEN_BYTE2=4;
parameter LISTEN_BYTE3=5;
parameter LISTEN_BYTE4=6;
parameter ACK0=7;
parameter ACK1=8;
parameter ACK2=9;
parameter SEND_ADDR0=10;
parameter SEND_ADDR_LOOP=11;
parameter SEND_ADDR1=12;
parameter SEND_ADDR2=13;
parameter SEND_ADDR3=14;
parameter ACK_ADDR0=15;
```

```
parameter ACK_ADDR1=16;
parameter ACK_ADDR2=17;
parameter STOP0=18;
parameter STOP1=19;
parameter STOP2=20;
parameter START_READ0=21;
parameter START_READ1=22;
parameter READ_BYTE0=23;
parameter READ_BYTE_LOOP=24;
parameter READ_BYTE1=25;
parameter READ_BYTE2=26;
parameter READ_BYTE3=27;
parameter READ_ACK0=28;
parameter READ_ACK1=29;
parameter READ_ACK2=30;
parameter RECEIVE_BYTE0=31;
parameter RECEIVE_LOOP=32;
parameter     RECEIVE_BYTE1=33;
parameter RECEIVE_BYTE2=34;
parameter RECEIVE_BYTE3=35;
parameter RECEIVE_ACK0=36;
parameter RECEIVE_ACK1=37;
parameter RECEIVE_ACK2=38;
parameter FINISH0=39;
parameter FINISH1=40;
parameter FINISH2=41;
parameter STOP3= 42;               //***added
parameter STOP4= 43;
parameter ACK_ADDR01=44;
parameter ACK01=45;

reg [5:0] state, next;
reg [3:0] bit, bitt;
reg SDA_int, SCL_int, SCL, done, SDA;
reg [7:0] listen, data, addr, read;

assign check_cond= (state==7 | state==8 | state==9 | state==10| state== 15| state==16|
state==17| state==18|
                           state== 44| state== 45|
                           state==28| state==29| state==30| state==31| state==32|
state==33| state==34| state==35);

always @ (posedge clk or negedge reset)
begin
       if(!reset) begin
       read<= 8'b10000111;
       listen<= 8'b10000110;
       addr<= 8'b00000001;
       data<= 8'b10101100;
       bit<= 0;
       bitt<= 0;
       state<= IDLE; end

       else begin
               if(start) begin

                     if(state== LISTEN_BYTE4) begin
                     listen<= {listen[6:0], 1'b0};
                     state<= next; end

                     else begin
                             if (state== SEND_ADDR3) begin
                             addr<= {addr[6:0], 1'b0};
                             state<= next; end

                             else begin
                                     if(state== READ_BYTE3) begin
                                     read<= {read[6:0], 1'b0};
                                     bit<= bit +1;
                                     state<= next;
                                     end
```

```
                                        else begin
                                                if (state== RECEIVE_BYTE3) begin
                                                data<= {data[6:0], 1'b0};
                                                bitt<= bitt + 1;
                                                state<= next;
                                                end

                                                else state<= next;
                                                end
                                        end

                                end
                        end

                else state<= next;
                end

end

 always @(posedge clk)
begin
 SCL<= SCL_int;
 SDA<= SDA_int;
end


always @ (state or next or start or listen or SDA or addr or bit or read or bitt)
                //removed SDA
begin
done= 0;
case(state)
IDLE:  begin
                SDA_int= 1;
                SCL_int= 1;
                if(start) next= START_WRITE;
                else next= IDLE;
            end

START_WRITE:  begin
                                SDA_int= 0;
                                SCL_int= 1;
                                next= LISTEN_BYTE0;
                                end

LISTEN_BYTE0:           begin
                                SDA_int= 0;
                                SCL_int= 0;
                                next= LISTEN_BYTE1;
                                end

LISTEN_BYTE1:           begin
                                SDA_int= listen[7];
                                SCL_int= 0;
                                if(SDA==0 & listen==0) next= ACK0;
                                else next= LISTEN_BYTE2;
                                end

LISTEN_BYTE2:           begin
                                SDA_int= listen[7];
                                SCL_int= 1;
                                next= LISTEN_BYTE3;
                                end

LISTEN_BYTE3:           begin
                                SDA_int= listen[7];
                                SCL_int= 1;
                                next= LISTEN_BYTE4;
                                end

LISTEN_BYTE4:           begin
```

```
                          SDA_int= listen[7];
                          SCL_int=0;
                          next= LISTEN_BYTE1;
                          end

ACK0: begin
         SCL_int= 0;        //WAS 1!!!
         next= ACK01;
         end

ACK01:        begin
              SCL_int=1;
              next= ACK1;
              end

ACK1: begin
         SCL_int=1;
         next= ACK2;
         end

ACK2: begin
         SCL_int=0;
         next= SEND_ADDR0;
         end

SEND_ADDR0:   begin
                   SCL_int=0;
                     SDA_int= addr[7];
                     next= SEND_ADDR1;
                     end

SEND_ADDR_LOOP: begin
                  SCL_int=0;
                    SDA_int= addr[7];
                    if(addr== 0 & SDA==1) next= ACK_ADDR0;
                    else next= SEND_ADDR1;
                    end

SEND_ADDR1:   begin
                    SCL_int=1;
                    SDA_int= addr[7];
                    next= SEND_ADDR2;
                    end

SEND_ADDR2: begin
                    SCL_int=1;
                    SDA_int= addr[7];
                    next= SEND_ADDR3;
                    end

SEND_ADDR3: begin
                    SCL_int=0;
                    SDA_int= addr[7];
                    next= SEND_ADDR_LOOP;
                    end

ACK_ADDR0:    begin
                    SCL_int=0; //WAS 1!!!
                    next= ACK_ADDR01;
                    end

ACK_ADDR01:   begin
                    SCL_int=1;
                    next= ACK_ADDR1;
                    end

ACK_ADDR1:    begin
                    SCL_int=1;
                    next= ACK_ADDR2;
                    end
```

```
ACK_ADDR2: begin
                SCL_int=0;
                next= STOP0;
                end

STOP0:        begin
              SCL_int=0;
              SDA_int=0;
              next= STOP1;
              end

STOP1:        begin
              SCL_int=1;
              SDA_int=0;
              next= STOP2;
              end

STOP2: begin
           SCL_int=1;
           SDA_int=1;
           next= STOP3;
           end

STOP3: begin
         SCL_int=1;
         SDA_int=1;
         next= STOP4;
         end

STOP4:        begin                              ///****added to satisfy timing
              SCL_int=1;
              SDA_int=1;
              next= START_READ0;
              end

START_READ0:  begin
                     SDA_int= 0;
                     SCL_int= 1;
                     next= START_READ1;
                     end

START_READ1:  begin
                           SDA_int= 0;
                           SCL_int= 0;
                           next= READ_BYTE0;
                           end

READ_BYTE0:   begin
                           SDA_int= read[7];
                           SCL_int= 0;
                           next= READ_BYTE1;
                           end

READ_BYTE_LOOP:      begin
                           SDA_int= read[7];
                           SCL_int= 0;
                           if (bit== 8) next= READ_ACK0;
                           else next= READ_BYTE1;
                           end

READ_BYTE1:   begin
                           SDA_int= read[7];
                           SCL_int= 1;
                           next= READ_BYTE2;
                           end

READ_BYTE2:   begin
                           SDA_int= read[7];
                           SCL_int= 1;
                           next= READ_BYTE3;
                           end
```

```
READ_BYTE3:   begin
                          SDA_int= read[7];
                          SCL_int=0;
                          next= READ_BYTE_LOOP;
                          end

READ_ACK0: begin
                     SCL_int= 1;
                     next= READ_ACK1;
                     end

READ_ACK1: begin
         SCL_int=1;
         next= READ_ACK2;
         end

READ_ACK2: begin
         SCL_int=0;
         next= RECEIVE_BYTE0;
         end

RECEIVE_BYTE0:       begin
                   SCL_int=0;
                   next= RECEIVE_BYTE1;
                     end

RECEIVE_LOOP: begin
                   SCL_int=0;
                     if(bitt== 8) next= RECEIVE_ACK0;
                     else next= RECEIVE_BYTE1;
                     end

RECEIVE_BYTE1:       begin
                     SCL_int=1;
                     next= RECEIVE_BYTE2;
                     end

RECEIVE_BYTE2: begin
                     SCL_int=1;
                     next= RECEIVE_BYTE3;
                     end

RECEIVE_BYTE3: begin
                     SCL_int=0;
                     next= RECEIVE_LOOP;
                     end

RECEIVE_ACK0:        begin
                     SCL_int=0;                                    //THIS MIGHT HAVE TO BE
HIGH--- NO ACK
                     SDA_int=1;
                     next= RECEIVE_ACK1;
                     end

RECEIVE_ACK1:        begin
                     SCL_int=1;//0
                     SDA_int=1;
                     next= RECEIVE_ACK2;
                     end

RECEIVE_ACK2: begin
                  SCL_int=1;
                  SDA_int=1; //0
                  next= FINISH0;
                  end

FINISH0:      begin
              SCL_int=0;
              SDA_int=0;
              next= FINISH1;
              end
```

```
FINISH1:      begin
              SCL_int=1;
              SDA_int=0;
              next= FINISH2;
              end

FINISH2: begin
          done= 1;
          SCL_int=1;
          SDA_int=1;
          next= IDLE;
          end



endcase
end

endmodule
```


```
/*****************************************************************************
*     This file is owned and controlled by Xilinx and must be used           *
*     solely for design, simulation, implementation and creation of          *
*     design files limited to Xilinx devices or technologies. Use            *
*     with non-Xilinx devices or technologies is expressly prohibited        *
*     and immediately terminates your license.                               *
*                                                                            *
*     XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS"          *
*     SOLELY FOR USE IN DEVELOPING PROGRAMS AND SOLUTIONS FOR                *
*     XILINX DEVICES.  BY PROVIDING THIS DESIGN, CODE, OR INFORMATION        *
*     AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE, APPLICATION            *
*     OR STANDARD, XILINX IS MAKING NO REPRESENTATION THAT THIS              *
*     IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT,                *
*     AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE       *
*     FOR YOUR IMPLEMENTATION.  XILINX EXPRESSLY DISCLAIMS ANY               *
*     WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE                *
*     IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR         *
*     REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF        *
*     INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS        *
*     FOR A PARTICULAR PURPOSE.                                              *
*                                                                            *
*     Xilinx products are not intended for use in life support               *
*     appliances, devices, or systems. Use in such applications are          *
*     expressly prohibited.                                                  *
*                                                                            *
*     (c) Copyright 1995-2004 Xilinx, Inc.                                   *
*     All rights reserved.                                                   *
*****************************************************************************/
```


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

// You must compile the wrapper file dido_rom.v when simulating
// the core, dido_rom. 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 dido_rom (
       addr,
       clk,
       din,
       dout,
       we);    // synthesis black_box

input [16 : 0] addr;
input clk;
input [15 : 0] din;
output [15 : 0] dout;
input we;
```

```
// synopsys translate_off

	BLKMEMSP_V6_1 #(
		17,	// c_addr_width
		"0",	// c_default_data
		97600,		// c_depth
		0,	// c_enable_rlocs
		0,	// 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
		"dido_rom.mif",		// c_mem_init_file
		0,	// c_pipe_stages
		0,	// c_reg_inputs
		"0",	// c_sinit_value
		16,	// c_width
		0,	// c_write_mode
		"0",	// c_ybottom_addr
		1,	// c_yclk_is_rising
		1,	// c_yen_is_high
		"hierarchy1",		// c_yhierarchy
		0,	// c_ymake_bmm
		"16kx1",		// c_yprimitive_type
		1,	// c_ysinit_is_high
		"1024",		// c_ytop_addr
		0,	// c_yuse_single_primitive
		1,	// c_ywe_is_high
		1)	// c_yydisable_warnings
	inst (
		.ADDR(addr),
		.CLK(clk),
		.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 dido_rom is "true"

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

endmodule

//////////////////////////////////////////////////////////////////////////////
//
// 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[30:4] = 27'hZ;
assign user4[0]= 1'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;
```

```
    ////////////////////////////////////
   ///    CONFIGURATION OF MP3 DECODER ///
   ////////////////////////////////////
       wire clk_slow;
       wire reset;
       //wire SDA;
       wire SCL;
       wire [2:0] state;
       wire check_cond;
       wire config_cond;
       wire SDA_mux_out;
       wire [5:0]check_state;            //

       assign reset= switch[0];
       assign reset_mp3= switch[1];

     systemCLK slow_clock(clock_27mhz, reset,
                                  clk_slow);

       topInit initialize(clk_slow, reset,
                                SCL, state, check_cond, config_cond, SDA_mux_out,
check_state);

       assign user4[31]= (check_cond | config_cond)? 1'bz : SDA_mux_out;
         //i2c protocol control signals
       assign user4[1]= SCL;


    ////////////////////////////////////
    ////             14.318 MHz Clock        ///
    ////////////////////////////////////

       assign user4[2]= clock1;
       assign user4[3]= reset_mp3;

       ////////////////////////////////
       //               AUDIO                        //
       ////////////////////////////////
  /*   wire play, stop;
       wire play_pulse, stop_pulse;
       wire up, down;
       wire data_req;
       wire data;
       wire [2:0] audio_state;

       assign play= ~button0;               //use negative logic
       assign stop= ~button1;
       assign up= ~button_up;
       assign down= ~button_down;

       level2pulse uppulse(clock1, reset, up,
                                 up_pulse);

       level2pulse downpulse(clock1, reset, down,
                                  down_pulse);

       level2pulse playpulse(clock1, reset, play,
                                  play_pulse);

       level2pulse stoppulse(clock1, reset, stop,
                                  stop_pulse);

       song_selector selector(clock1, reset, up_pulse, down_pulse,
                                 selected_song);

       topAudio audio(clock1, reset, play_pulse, stop_pulse, data_req,
                           data, audio_state);

       assign data_req= user4[4];
       assign user4[5]= data;
       assign led[0]= ~play;
```

```
	assign led[1]= ~stop;
	assign led[2]= data_req;           */
	assign led[7:0]= 8'b11111111;

	//////////////////////////
	//     LOGIC ANALYZER          ///
	//////////////////////////

	assign analyzer4_data[2:0]= state[2:0];
	assign analyzer4_data[3]= user4[31];
	assign analyzer4_data[4]= SCL;
	assign analyzer4_data[5]= check_cond;
	assign analyzer4_data[6]= config_cond;
	assign analyzer4_data[7]= reset;
	assign analyzer4_data[15:8]= 8'h0;
	assign analyzer4_clock= clk_slow;

	assign analyzer2_data[5:0]= check_state[5:0];

	/*assign analyzer2_data[2:0]= audio_state[2:0];
	assign analyzer2_data[3]= play_pulse;
	assign analyzer2_data[4]= stop_pulse;
	assign analyzer2_data[5]= data_req;
	assign analyzer2_data[6]= data;
	assign analyzer2_data[15:7]= 9'h0;          */
	//assign analyzer4_clock= clock1;


endmodule


//////////////////////////
////      Serializer        //
///       Alma E. Rico      //
//////////////////////////

module serializer(clk, reset, enable, ROM_data, serial_data);
input clk, reset, enable;
input [15:0] ROM_data;
output serial_data;
//output [5:0] count;       //*****
//output [15:0] temp;

reg [15:0] temp;
reg [5:0] count;
reg serial_data;

always @ (posedge clk or negedge reset)
begin
	if(!reset) begin
	temp<= ROM_data;
	serial_data<= 1'bz;
	count<= 0; end
	else if (enable) begin
			if(count == 15) begin
			count<= 0;
			temp<= ROM_data;
			serial_data<= temp[15]; end
			else begin
			temp<= {temp[14:0],1'b0};
			serial_data<= temp[15];
			count<= count + 1; end
			end

		else serial_data<= 1'bz;


end

endmodule
```

```
////////////////////////////
//   Audio System Clock   ///
//     Alma E. Rico        //
////////////////////////////

module systemCLK(clk, reset, audio_clock);

input clk, reset;
output audio_clock;

reg [7:0]count;

always @ (posedge clk or negedge reset)
begin
       if(!reset) count<= 0;
     else if(audio_clock) count<= 0;
         else count<= count + 1;

end

assign audio_clock = (count == 150);                          //change back to 40
endmodule

//////////////////////////////////
///         Audio TOP               ///
///            Alma E. Rico          ///
//////////////////////////////////


module topAudio(clk, reset, play, stop, data_req,
                      data, audio_state);

input clk, reset;
input play, stop;
input data_req;
output data;
output [2:0] audio_state;          //***

wire [16:0] start_song;
wire serialize_enable;
wire serialize_reset;
wire [16:0] ROM_addr;         //
wire [15:0] ROM_data;          //
//output [5:0] count;              ///*************
//output [15:0] temp;

assign start_song= 0;                                 //will change with multiple songs!


audioFSM audiofsm(clk, reset, data_req, play, stop, start_song,
                      serialize_enable, serialize_reset, ROM_addr, audio_state) ;
       //state

DIDO_rom musicROM(ROM_addr, clk, ROM_data);

serializer serial(clk, serialize_reset, serialize_enable, ROM_data, data);    //count,
temp



endmodule


////////////////////////////////
///        AUDIO FSM             ///
///     Alma E. Rico            ///
////////////////////////////////

module audioFSM(clk, reset, data_req, play_sig, stop_sig, start_song,
                      serialize_enable, serialize_reset, ROM_addr, state);
```

```
input clk, reset;
input data_req;                         //data request signal from MP3 decoder
input play_sig, stop_sig;         //user interface signals
input [16:0]start_song;                 // start address of chosen song
output [2:0] state;                     //***

output serialize_enable;                //to serializer
output serialize_reset;
output [16:0] ROM_addr;                 //depends on size of ROM!!!

parameter IDLE=0;
parameter PLAY=1;
parameter WAIT=2;
parameter PAUSE=3;
parameter STOP=4;

reg [5:0] count;                        //change every 16 counts
reg [16:0] ROM_addr;
reg serialize_enable;
reg serialize_reset;
reg [2:0] state, next;

always @ (posedge clk or negedge reset)
begin
       if(!reset)   begin
       state<= IDLE;
       ROM_addr<= start_song;
       serialize_reset<= 0; end

       else begin
              if ((state== PLAY) & data_req) begin
                     serialize_reset<= 1;
                     serialize_enable<=1;
                     state<= next;

                     if(count== 15) count<= 0;
                     else begin
                            if(count==14) begin
                            ROM_addr<= ROM_addr +1;
                            count<= count +1;end
                            else count<= count + 1;
                            end
                                    end
              else begin


                     if(state == STOP | state == IDLE) begin
                     ROM_addr<= start_song;
                     count<= 0;
                     state<= next;
                     serialize_reset<= 0;
                     serialize_enable<=0;end
                     else   begin
                     serialize_reset<= 1;
                     state<= next;
                     serialize_enable<=0; end

                     end
              end
end

always @ (state or next or play_sig or stop_sig or data_req)
begin

case(state)

IDLE:  begin
              if(play_sig) next= PLAY;
              else next= IDLE;
              end
```

```
PLAY:  begin
              if (play_sig) next= PAUSE;
              else begin
                     if(stop_sig) next= STOP;
                     else begin
                            if(data_req) next= PLAY;
                            else next= WAIT;
                            end
                     end
              end

WAIT: begin
        if(data_req) next= PLAY;
        else begin
               if(stop_sig) next= STOP;
               else begin
                      if(play_sig) next= PAUSE;
                      else next= WAIT;
                      end

        end

PAUSE:        begin
              if(play_sig) next= PLAY;
              else begin
                     if(stop_sig) next= STOP;
                     else next= PAUSE;
                     end
              end

STOP:  begin
              if(play_sig) next= PLAY;
              else next= STOP;
              end




endcase
end



endmodule


////////////////////////////
//   Configuration FSM      ///
//        Alma E. Rico      ///
////////////////////////////

module configFSM(clk, reset, start, addr_in, data_in,
                           ROM_addr, SDA, SCL, state, done);

input clk, reset, start;
input [7:0] data_in, addr_in;            //READ from ROMs
output [5:0] state;
output [10:0] ROM_addr;
output SDA;
output SCL, done;

parameter IDLE=0;
parameter START_WRITE=1;
parameter LISTEN_BYTE0=2;
parameter LISTEN_BYTE1=3;
parameter LISTEN_BYTE2=4;
parameter LISTEN_BYTE3=5;
parameter LISTEN_BYTE4=6;
parameter ACK0=7;
parameter ACK1=8;
```

```
parameter ACK2=9;
parameter SEND_ADDR0=10;
parameter SEND_ADDR_LOOP=11;
parameter SEND_ADDR1=12;
parameter SEND_ADDR2=13;
parameter SEND_ADDR3=14;
parameter ACK_ADDR0=15;
parameter ACK_ADDR1=16;
parameter ACK_ADDR2=17;
parameter SEND_DATA0=18;
parameter SEND_DATA_LOOP=19;
parameter SEND_DATA1=20;
parameter SEND_DATA2=21;
parameter SEND_DATA3=22;
parameter ACK_DATA0=23;
parameter ACK_DATA1=24;
parameter ACK_DATA2=25;
parameter STOP0=26;
parameter STOP1=27;
parameter STOP2=28;
parameter STOP3=29;
parameter STOP4=30;
parameter WAIT_RESET=31;

reg [10:0] count;
reg [5:0] state, next;
reg [3:0] bit, bitt;
reg SDA_int, SCL_int, SDA, SCL, done;
reg [7:0] listen, addr, data;
reg [10:0] ROM_addr;
reg [6:0] wait_period;

always @ (posedge clk or negedge reset)
begin
	if(!reset) begin
	listen<= 8'b10000110;
	bit<= 0;
	bitt<= 0;
	ROM_addr<= 0;
	count<= 1;
	wait_period<= 1;
	state<= IDLE; end

	else begin
		if(start) begin

			if(state== LISTEN_BYTE4) begin
			addr<= addr_in;
			data<= data_in;
			listen<= {listen[6:0], 1'b0};
			state<= next; end

			else begin
				if (state== SEND_ADDR3) begin
				addr<= {addr[6:0], 1'b0};
				bit<= bit + 1;                          //need to get to 9
since registered
				state<= next; end

				else begin
					if(state== SEND_DATA3) begin
					data<= {data[6:0], 1'b0};
					bitt<= bitt +1;
					state<= next;    end

					else begin
						if (state== STOP2) begin
						listen<= 8'b10000110;
						bit<= 0;
						bitt<= 0;
						ROM_addr<= count;
```

```
                                                count<= count+ 1;
                                                state<= next;
                                                end

                                                else begin
                                                       if (state== WAIT_RESET) begin
                                                      state<= next;
                                                        wait_period<= wait_period + 1;end

                                                        else state<= next;
                                                        end
                                                       end
                                                end
                                        end

                                end


                else state<= next;
                end

end

always @(posedge clk)
begin
 SCL<= SCL_int;
 SDA<= SDA_int;
end

always @ (state or next or start or listen or SDA or addr or bit or data or bitt or
count or wait_period)
begin
done=0;
case(state)

IDLE:  begin
               SDA_int= 1;
               SCL_int= 1;
               if(start) next= START_WRITE;
               else next= IDLE;
            end

START_WRITE:  begin
                               SDA_int= 0;
                               SCL_int= 1;
                               next= LISTEN_BYTE0;
                               end

LISTEN_BYTE0:          begin
                               SDA_int= 0;
                               SCL_int= 0;
                               next= LISTEN_BYTE1;
                               end

LISTEN_BYTE1:          begin
                               SDA_int= listen[7];
                               SCL_int= 0;
                               if(SDA==0 & listen==0) next= ACK0;
                               else next= LISTEN_BYTE2;
                               end

LISTEN_BYTE2:          begin
                               SDA_int= listen[7];
                               SCL_int= 1;
                               next= LISTEN_BYTE3;
                               end

LISTEN_BYTE3:          begin
                               SDA_int= listen[7];
                               SCL_int= 1;
                               next= LISTEN_BYTE4;
```

```
                        end

LISTEN_BYTE4:       begin
                        SDA_int= listen[7];
                        SCL_int=0;
                        next= LISTEN_BYTE1;
                        end

ACK0: begin
        SCL_int= 1;
        next= ACK1;
        end

ACK1: begin
        SCL_int=1;
        next= ACK2;
        end

ACK2: begin
        SCL_int=0;
        next= SEND_ADDR0;
        end

SEND_ADDR0:   begin
                SCL_int=0;
                  SDA_int= addr[7];
                  next= SEND_ADDR1;
                  end

SEND_ADDR_LOOP: begin
                  SCL_int=0;
                  SDA_int= addr[7];
                  if(bit== 8) next= ACK_ADDR0;
                  else next= SEND_ADDR1;
                  end

SEND_ADDR1:   begin
                  SCL_int=1;
                  SDA_int= addr[7];
                  next= SEND_ADDR2;
                  end

SEND_ADDR2: begin
                  SCL_int=1;
                  SDA_int= addr[7];
                  next= SEND_ADDR3;
                  end

SEND_ADDR3: begin
                  SCL_int=0;
                  SDA_int= addr[7];
                  next= SEND_ADDR_LOOP;
                  end

ACK_ADDR0:    begin
                  SCL_int=1;
                  next= ACK_ADDR1;
                  end

ACK_ADDR1:    begin
                  SCL_int=1;
                  next= ACK_ADDR2;
                  end

ACK_ADDR2: begin
                SCL_int=0;
                next= SEND_DATA0;
                end

SEND_DATA0: begin
                  SCL_int=0;
```

```
                    SDA_int= data[7];
                    next= SEND_DATA1;
                    end

SEND_DATA_LOOP:begin
                    SCL_int=0;
                    SDA_int= data[7];
                    if(bitt==8) next= ACK_DATA0;
                    else next= SEND_DATA1;
                    end

SEND_DATA1:   begin
                    SCL_int=1;
                    SDA_int= data[7];
                    next= SEND_DATA2;
                    end

SEND_DATA2:   begin
                    SCL_int=1;
                    SDA_int= data[7];
                    next= SEND_DATA3;
                    end

SEND_DATA3:   begin
                    SCL_int=0;
                    SDA_int= data[7];
                    next= SEND_DATA_LOOP;
                    end

ACK_DATA0:    begin
                    SCL_int=1;
                    next= ACK_DATA1;
                    end

ACK_DATA1:    begin
                    SCL_int=1;
                    next= ACK_DATA2;
                    end

ACK_DATA2:    begin
                    SCL_int=0;
                    next= STOP0;
                    end

STOP0:        begin
              SCL_int=0;
              SDA_int=0;
              next= STOP1;
              end

STOP1:        begin
              SCL_int=1;
              SDA_int=0;                  /////check timing of this
              next= STOP2;
              end

STOP2: begin
          SCL_int=1;
          SDA_int=1;
          if(count== 1997) begin            //wait 3ms after soft reset change to 2015
if wrong
          //done= 1;
          //next= IDLE;
          next= WAIT_RESET;end
          else if (count== 2015) begin
                done=1;
                next= IDLE; end
                else next= STOP3;
          end

STOP3:        begin
```

```
              SCL_int=1;
              SDA_int=1;
              next= STOP4;
              end

STOP4: begin
         SCL_int=1;
         SDA_int=1;
         next= START_WRITE;
         end

///CHANGE//////
WAIT_RESET: begin
                SCL_int=1;
                SDA_int=1;
                if (wait_period== 80) next= START_WRITE;
                else next= WAIT_RESET;
                end



endcase
end
endmodule


////////////////////////////////
//////  Initialize FSM          ///
/////  Alma E. Rico         ///
////////////////////////////////


module initFSM(clk, reset, done_check, done_config,
                     start_config, start_check, state);

input clk, reset;
input done_check, done_config;
output start_check, start_config;
output [2:0] state;

parameter BEGIN=0;
parameter CHECK=1;
parameter WAIT_CTRL_HIGH=2;
parameter CONFIGURE=3;
parameter IDLE=4;

reg [2:0] state, next;
reg start_check, start_config;

always @ (posedge clk or negedge reset)
begin
       if (!reset) state<= BEGIN;
       else state<= next;

end


always @ (state or next or done_check or done_config)
begin

start_check= 0; start_config=0;

case(state)

BEGIN:  next= CHECK;

CHECK:        begin
              start_check=1;
              if(done_check) next= WAIT_CTRL_HIGH;
              else next= CHECK;
              end
```

```
WAIT_CTRL_HIGH: next= CONFIGURE;

CONFIGURE:    begin
                      start_config=1;
                      if(done_config) next= IDLE;
                      else next= CONFIGURE;
                      end


IDLE:  next= IDLE;

endcase
end

endmodule


////////////////////////////////
///  Level 2 Pulse Converter   ///
///            Alma E. Rico          ///
////////////////////////////////

module level2pulse(clk, reset, level_sig, pulse);

input clk, reset;
input level_sig;
output pulse;

reg level_sync;
reg pulse;
reg [1:0] state, next;

parameter IDLE=0;
parameter EDGE_DETECTED=1;
parameter WAIT_FALL=2;


always @ (posedge clk or negedge reset)
begin
       if(!reset) state<= IDLE;
       else state<= next;
end

always @ (posedge clk)
begin
level_sync<= level_sig;
end

always @ (state or next or level_sync)
begin
case(state)

IDLE:  begin
              pulse=0;
              if(level_sync) next= EDGE_DETECTED;
              else next= IDLE;
              end


EDGE_DETECTED:       begin
                      pulse=1;
                      if(level_sync) next= WAIT_FALL;
                      else next= IDLE;
                      end

WAIT_FALL:    begin
                      pulse=0;
                      if(level_sync) next= WAIT_FALL;
                      else next= IDLE;
                      end
```

```
endcase
end



endmodule


////////////////////////////////
////     Song Selector Module     //
///          Alma E. Rico        //
////////////////////////////////

module song_selector(clk, reset, up, down,
                              selected_song);

input clk, reset;
input up, down;
output [1:0] selected_song;

reg [1:0] count, selected_song;   //support for 2 songs!

always @ (posedge clk or negedge reset)
begin
       if(!reset) selected_song<= 0; //since ROM begins at zero
       else if(up)
              count<= count + 1;
              else if (down)
                     count<= count - 1;
                     else  selected_song<= count;

end



endmodule


 ////////////////////////////////
//     Configuration Testbench     //
///       Alma E. Rico             //
////////////////////////////////


//********MAKE SURE TO CONSIDER TIMING SPECS!!!


`timescale 1ns/1ps
module topConfig_test_v_tf();

// DATE:     16:17:56 04/30/2005
// MODULE:   topConfig
// DESIGN:   topConfig
// FILENAME: test.v
// PROJECT:  FinalProject
// VERSION:


// Inputs
    reg clk;
    reg reset;
    reg start;


// Outputs
    wire SCL;
    //wire [5:0] state;
    wire config_cond;
    wire [7:0] data_in, addr_in;
    wire done;
```

```
// Bidirs
    wire SDA_val;


// Instantiate the UUT
    topConfig uut (
        .clk(clk),
        .reset(reset),
        .start(start),
        .SDA_val(SDA_val),
        .SCL(SCL)     ,
          .done(done),
          .config_cond(config_cond),
          .data_in(data_in),
          .addr_in(addr_in)
        );

  always #5 clk= ~clk;

// Initialize Inputs
  //  `ifdef auto_init

        initial begin
           clk= 0;
               reset= 0;
               start= 0;
               #20;
               reset= 1;
               #20;
               start=1;
               if(done==1) start= 0;
               else start=1;

        end

 //   `endif


endmodule

////////////////////////////////
///    Configuration Top           ///
///           Alma E. Rico          ///
////////////////////////////////

//CHANGED SDA TO OUTPUT

module topConfig(clk, reset, start,
                      SDA_val, SCL, done, config_cond);

input clk, reset, start;
output SCL, done;
output SDA_val;
output config_cond;

wire[5:0] state;
wire [10:0] ROM_addr;
wire [7:0] data_in, addr_in;


configFSM fsm(clk, reset, start, addr_in, data_in,
                              ROM_addr, SDA_val, SCL, state, done);

addressrom addr(ROM_addr, clk, addr_in);

datarom data(ROM_addr, clk, data_in);

assign config_cond=
(state==7|state==8|state==9|state==10|state==15|state==16|state==17|state==18|
                             state==23|state==24|state==25|state==26);
```

```
endmodule

////////////////////////////////
//     Audio Testbench            //
///       Alma E. Rico             //
////////////////////////////////

`timescale 1ns/1ps

module topAudio_audioTestBench_v_tf();

// DATE:     22:29:20 05/06/2005
// MODULE:   topAudio
// DESIGN:   topAudio
// FILENAME: audioTestBench.v
// PROJECT:  FinalProject
// VERSION:


// Inputs
    reg clk;
    reg reset;
    reg play;
    reg stop;
    reg data_req;


// Outputs
    wire data;
    wire [2:0] state;


// Bidirs


// Instantiate the UUT
    topAudio uut (
        .clk(clk),
        .reset(reset),
        .play(play),
          .stop(stop),
        .data_req(data_req),
        .data(data)   ,
          .audio_state(state)

        );

 always #5 clk= ~clk;

// Initialize Inputs
    //`ifdef auto_init                              //isn't working well check values here

        initial begin
            clk = 0;
            reset = 0;
            play = 0;
                stop= 0;
            data_req = 0;
                #20
                reset=1;
                #20
                play=1;
                data_req=1;
                #20
                play=0;
                #50
                play=1;
                #10
                play=0;
                #50
                stop=1;
```

```
                #20
                stop=0;
                #50
                play=1;
                #10
                play=0;
                #60
                data_req=0;
                #40
                data_req=1;


        end

   // `endif

endmodule

/*****************************************************************************
*     This file is owned and controlled by Xilinx and must be used           *
*     solely for design, simulation, implementation and creation of          *
*     design files limited to Xilinx devices or technologies. Use            *
*     with non-Xilinx devices or technologies is expressly prohibited        *
*     and immediately terminates your license.                               *
*                                                                            *
*     XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS"          *
*     SOLELY FOR USE IN DEVELOPING PROGRAMS AND SOLUTIONS FOR                *
*     XILINX DEVICES.  BY PROVIDING THIS DESIGN, CODE, OR INFORMATION        *
*     AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE, APPLICATION            *
*     OR STANDARD, XILINX IS MAKING NO REPRESENTATION THAT THIS              *
*     IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT,                *
*     AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE       *
*     FOR YOUR IMPLEMENTATION.  XILINX EXPRESSLY DISCLAIMS ANY               *
*     WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE                *
*     IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR         *
*     REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF        *
*     INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS        *
*     FOR A PARTICULAR PURPOSE.                                              *
*                                                                            *
*     Xilinx products are not intended for use in life support               *
*     appliances, devices, or systems. Use in such applications are          *
*     expressly prohibited.                                                  *
*                                                                            *
*     (c) Copyright 1995-2004 Xilinx, Inc.                                   *
*     All rights reserved.                                                   *
*****************************************************************************/
// The synopsys directives "translate_off/translate_on" specified below are
// supported by XST, FPGA Compiler II, Mentor Graphics and Synplicity synthesis
// tools. Ensure they are correct for your synthesis tool(s).

// You must compile the wrapper file datarom.v when simulating
// the core, datarom. 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 datarom (
       addr,
       clk,
       dout);    // synthesis black_box

input [10 : 0] addr;
input clk;
output [7 : 0] dout;

// synopsys translate_off

       BLKMEMSP_V6_1 #(
              11,    // c_addr_width
```

```
            "0",   // c_default_data
            2015,  // c_depth
            0,     // c_enable_rlocs
            0,     // c_has_default_data
            0,     // c_has_din
            0,     // c_has_en
            0,     // c_has_limit_data_pitch
            0,     // c_has_nd
            0,     // c_has_rdy
            0,     // c_has_rfd
            0,     // c_has_sinit
            0,     // c_has_we
            18,    // c_limit_data_pitch
            "datarom.mif",       // c_mem_init_file
            0,     // c_pipe_stages
            0,     // c_reg_inputs
            "0",   // c_sinit_value
            8,     // c_width
            0,     // c_write_mode
            "0",   // c_ybottom_addr
            1,     // c_yclk_is_rising
            1,     // c_yen_is_high
            "hierarchy1",        // c_yhierarchy
            0,     // c_ymake_bmm
            "16kx1",      // c_yprimitive_type
            1,     // c_ysinit_is_high
            "1024",       // c_ytop_addr
            0,     // c_yuse_single_primitive
            1,     // c_ywe_is_high
            1)     // c_yydisable_warnings
      inst (
            .ADDR(addr),
            .CLK(clk),
            .DOUT(dout),
            .DIN(),
            .EN(),
            .ND(),
            .RFD(),
            .RDY(),
            .SINIT(),
            .WE());


// synopsys translate_on

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

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

endmodule


////////////////////////////////
//     Initialization Testbench    //
///       Alma E. Rico             //
////////////////////////////////

`timescale 1ns/1ps

module topInit_initializationTest_v_tf();

// DATE:     20:59:38 05/02/2005
// MODULE:   topInit
// DESIGN:   topInit
// FILENAME: initializationTest.v
// PROJECT:  FinalProject
// VERSION:
```

```
// Inputs
    reg clk;
    reg reset;


// Outputs
    wire SCL;
    wire [2:0] state;
    wire check_cond;
    wire config_cond;


// Bidirs
    wire SDA;


// Instantiate the UUT
    topInit uut (
        .clk(clk),
        .reset(reset),
        .SDA(SDA),
        .SCL(SCL),
        .state(state),
          .check_cond(check_cond),
        .config_cond(config_cond)
        );

always #5 clk= ~clk;
// Initialize Inputs
    //`ifdef auto_init

        initial begin
           clk= 0;
               reset= 0;
               #20;
               reset= 1;
               #20;

        end

    //`endif


endmodule

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

```
*     (c) Copyright 1995-2004 Xilinx, Inc.                                     *
*     All rights reserved.                                                     *
*******************************************************************************/
// The synopsys directives "translate_off/translate_on" specified below are
// supported by XST, FPGA Compiler II, Mentor Graphics and Synplicity synthesis
// tools. Ensure they are correct for your synthesis tool(s).

// You must compile the wrapper file madonnarom.v when simulating
// the core, madonnarom. 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 madonnarom (
       addr,
       clk,
       dout);    // synthesis black_box

input [15 : 0] addr;
input clk;
output [15 : 0] dout;

// synopsys translate_off

       BLKMEMSP_V6_1 #(
              16,    // c_addr_width
              "0",   // c_default_data
              38252,        // c_depth
              0,     // c_enable_rlocs
              0,     // c_has_default_data
              0,     // c_has_din
              0,     // c_has_en
              0,     // c_has_limit_data_pitch
              0,     // c_has_nd
              0,     // c_has_rdy
              0,     // c_has_rfd
              0,     // c_has_sinit
              0,     // c_has_we
              18,    // c_limit_data_pitch
              "madonnarom.mif",    // c_mem_init_file
              0,     // c_pipe_stages
              0,     // c_reg_inputs
              "0",   // c_sinit_value
              16,    // c_width
              0,     // c_write_mode
              "0",   // c_ybottom_addr
              1,     // c_yclk_is_rising
              1,     // c_yen_is_high
              "hierarchy1",        // c_yhierarchy
              0,     // c_ymake_bmm
              "16kx1",      // c_yprimitive_type
              1,     // c_ysinit_is_high
              "1024",       // c_ytop_addr
              0,     // c_yuse_single_primitive
              1,     // c_ywe_is_high
              1)     // c_yydisable_warnings
       inst (
              .ADDR(addr),
              .CLK(clk),
              .DOUT(dout),
              .DIN(),
              .EN(),
              .ND(),
              .RFD(),
              .RDY(),
              .SINIT(),
              .WE());


// synopsys translate_on

// FPGA Express black box declaration
```

```
// synopsys attribute fpga_dont_touch "true"
// synthesis attribute fpga_dont_touch of madonnarom is "true"

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

endmodule

 ////////////////////////////////
//     Configuration Testbench     //
///       Alma E. Rico             //
////////////////////////////////


//********MAKE SURE TO CONSIDER TIMING SPECS!!!


`timescale 1ns/1ps

module checkFSM_testCheck_v_tf();

// DATE:     20:19:05 05/02/2005
// MODULE:   checkFSM
// DESIGN:   checkFSM
// FILENAME: testCheck.v
// PROJECT:  FinalProject
// VERSION:


// Inputs
    reg clk;
    reg reset;
    reg start;


// Outputs
    wire SDA;
    wire SCL;
    wire done;
    wire check_cond;
    wire [6:0] state;


// Bidirs


// Instantiate the UUT
    checkFSM uut (
        .clk(clk),
        .reset(reset),
        .start(start),
        .SDA(SDA),
        .SCL(SCL),
        .done(done),
        .check_cond(check_cond),
          .state(state)
        );

always #5 clk= ~clk;

// Initialize Inputs
    //`ifdef auto_init

        initial begin
           clk= 0;
               reset= 0;
               start= 0;
               #20;
               reset= 1;
               #20;
```

```
                start=1;
        end

    //`endif


endmodule

////////////////////////////////
///  Initialize MP3 Decoder    ///
///     Alma E. Rico           ///
////////////////////////////////

module topInit(clk, reset,
                      SCL, state, check_cond, config_cond, SDA_mux_out, check_state);

input clk, reset;
output SCL;
output [2:0] state;
output SDA_mux_out;
output check_cond, config_cond;     //*****
output [5:0] check_state;

wire done_check;
wire done_config;
wire start_check;
wire start_config;
wire SDA_config, SDA_check;
wire SCL_config, SCL_check;

initFSM major(clk, reset, done_check, done_config,
                     start_config, start_check, state);

checkFSM check(clk, reset, start_check,
                     SDA_check, SCL_check, done_check, check_cond, check_state);

topConfig configModule(clk, reset, start_config,
                               SDA_config, SCL_config, done_config, config_cond);


assign SCL= (start_config)? SCL_config : SCL_check;     //not bidirectional

assign SDA_mux_out= (start_config)? SDA_config : SDA_check;

//assign SDA= (check_cond | config_cond)? 1'bz : SDA_mux_out;



endmodule
```

LED Display Code

labkit.v

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

```
//
// CHANGES FOR BOARD REVISION 004
//
// 1) Added signals for logic analyzer pods 2-4.
// 2) Expanded "tv_in_ycrcb" to 20 bits.
// 3) Renamed "tv_out_data" to "tv_out_i2c_data" and "tv_out_sclk" to
//    "tv_out_i2c_clock".
// 4) Reversed disp_data_in and disp_data_out signals, so that "out" is an
//    output of the FPGA, and "in" is an input.
//
// CHANGES FOR BOARD REVISION 003
//
// 1) Combined flash chip enables into a single signal, flash_ce_b.
//
// CHANGES FOR BOARD REVISION 002
//
// 1) Added SRAM clock feedback path input and output
// 2) Renamed "mousedata" to "mouse_data"
// 3) Renamed some ZBT memory signals. Parity bits are now incorporated into
//    the data bus, and the byte write enables have been combined into the
//    4-bit ram#_bwe_b bus.
// 4) Removed the "systemace_clock" net, since the SystemACE clock is now
//    hardwired on the PCB to the oscillator.
//
//////////////////////////////////////////////////////////////////////////////
//
// Complete change history (including bug fixes)
//
// 2005-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;
```

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

   reg [3:0] song_playing, song_displaying;
   wire [39:0]pixel_output;
   wire [7:0] read_addr;
   wire we;
   assign we = 1'b0;
   wire [15:0] din;
   assign din = 16'b0;

   wire [15:0] hex_input;
   wire [639:0] dots;

   wire done;
   wire reset;
   assign reset = switch[0];
   wire write;
   assign write = switch[1];

wire [3:0]number_of_songs;
assign number_of_songs = 3'd1;

always @ (posedge clock_27mhz) begin

if (reset) begin song_playing = 4'b0;
                        song_displaying = 4'b1;
end

else begin
if (~button0) if (song_displaying < (number_of_songs + 1)) song_displaying =
song_displaying + 1;
if (~button1) if (song_displaying > 0) song_displaying = song_displaying - 1;
if (~button2) song_displaying = song_playing;
else begin song_playing = song_playing;
song_displaying = song_displaying;
end
```

```
end


end


title_to_leds my_title_to_leds(reset, clock_27mhz, write, song_playing, song_displaying,
pixel_output,

read_addr, dots, done);


hex_to_char my_hex_to_char(reset, clock_27mhz, hex_input, pixel_output);


title_sram my_title_sram(

        read_addr,

        clock_27mhz,

        din,

        hex_input,

        we);


display my_display(reset, clock_27mhz,
                disp_blank, disp_clock, disp_rs, disp_ce_b,
                disp_reset_b, disp_data_out, dots);




assign led = read_addr;




endmodule
```

display.v

```
//////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Alphanumeric Display Interface
//
//
// Created: November 5, 2003
// Author: Nathan Ickes
//
//////////////////////////////////////////////////////////////////////////////
//
// Change history
//
// 2005-05-09: Made <dots> input registered, and converted the 640-input MUX
//             to a 640-bit shift register.
//
//////////////////////////////////////////////////////////////////////////////

module display (reset, clock_27mhz,
              disp_blank, disp_clock, disp_rs, disp_ce_b,
              disp_reset_b, disp_data_out, dots);

   input  reset, clock_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 [4:0] count;
   reg [7:0] reset_count;
   reg clock;
   wire dreset;

   always @(posedge clock_27mhz)
     begin
       if (reset)
         begin
            count = 0;
            clock = 0;
         end
       else if (count == 26)
         begin
            clock = ~clock;
            count = 5'h00;
         end
       else
         count = count+1;
     end

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

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

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

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

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

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

     8'h05:
```

```
          begin
             // Latch the control register data
             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_rs <= 1'b0; // Select 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

endmodule
```

title_to_led.v

```
module title_to_leds(reset, clk, write, song_playing, song_displaying, pixel_output,
read_addr, dot_output, done);


input reset, clk, write;
//input [15:0] ascii_value;
input [3:0] song_playing, song_displaying;
input [39:0] pixel_output;
wire [7:0] start_addr;
assign start_addr = song_displaying * 15;


output [7:0]read_addr;
output [639:0] dot_output;
output done;
reg done;


parameter idle = 6'd0;
parameter converting1 =6'd1;
parameter converting2 = 6'd2;
parameter converting3 =6'd3;
parameter converting4 = 6'd4;
parameter converting5 = 6'd5;
parameter converting6 = 6'd6;
```

```
parameter converting7 = 6'd7;
parameter converting8 =6'd8;
parameter converting9 = 6'd9;
parameter converting10 =6'd10;
parameter converting11 = 6'd11;
parameter converting12 = 6'd12;
parameter converting13 = 6'd13;
parameter converting14 =6'd14;
parameter converting15 = 6'd15;
parameter waiting = 6'd16;
parameter waiting2 = 6'd17;

reg [7:0] read_addr;
reg [7:0] read_addr_int;
reg [39:0] pixel_output_register;
reg [639:0] dot_output;
reg [599:0] dot_output_int;
reg [5:0] state, next;
reg inc;
reg [6:0] count;

reg [39:0]converting1_reg_int;
reg [39:0]converting2_reg_int;
reg [39:0]converting3_reg_int;
reg [39:0]converting4_reg_int;
reg [39:0]converting5_reg_int;
reg [39:0]converting6_reg_int;
reg [39:0]converting7_reg_int;
reg [39:0]converting8_reg_int;
reg [39:0]converting9_reg_int;
reg [39:0]converting10_reg_int;
reg [39:0]converting11_reg_int;
reg [39:0]converting12_reg_int;
reg [39:0]converting13_reg_int;
reg [39:0]converting14_reg_int;
reg [39:0]converting15_reg_int;
```

```
reg [39:0]converting1_reg;
reg [39:0]converting2_reg;
reg [39:0]converting3_reg;
reg [39:0]converting4_reg;
reg [39:0]converting5_reg;
reg [39:0]converting6_reg;
reg [39:0]converting7_reg;
reg [39:0]converting8_reg;
reg [39:0]converting9_reg;
reg [39:0]converting10_reg;
reg [39:0]converting11_reg;
reg [39:0]converting12_reg;
reg [39:0]converting13_reg;
reg [39:0]converting14_reg;
reg [39:0]converting15_reg;

//hex_to_char my_hex_to_char(reset, clk, ascii_value, pixel_output);

always @ (posedge clk) begin

	if (reset) begin
		state = idle;
		count = 7'b0000000;
		read_addr = start_addr;
	end
	else begin
		state = next;
		count = count;
		pixel_output_register = pixel_output;

		converting1_reg = converting1_reg_int;
		converting2_reg = converting2_reg_int;
		converting3_reg = converting3_reg_int;
		converting4_reg = converting4_reg_int;
		converting5_reg = converting5_reg_int;
```

```
                converting6_reg = converting6_reg_int;
                converting7_reg = converting7_reg_int;
                converting8_reg = converting8_reg_int;
                converting9_reg = converting9_reg_int;
                converting10_reg = converting10_reg_int;
                converting11_reg = converting11_reg_int;
                converting12_reg = converting12_reg_int;
                converting13_reg = converting13_reg_int;
                converting14_reg = converting14_reg_int;
                converting15_reg = converting15_reg_int;

                end

        if (song_displaying == song_playing) dot_output[639:600] =
40'b01111111_00111110_00011100_00001000_00000000;
        else dot_output[639:600] = 40'b0;

        if (done) begin
                count = 6'b0;
                dot_output[599:560] = converting1_reg;
                dot_output[559:520] = converting2_reg;
                dot_output[519:480] = converting3_reg;
                dot_output[479:440] = converting4_reg;
                dot_output[439:400] = converting5_reg;
                dot_output[399:360] = converting6_reg;
                dot_output[359:320] = converting7_reg;
                dot_output[319:280] = converting8_reg;
                dot_output[279:240] = converting9_reg;
                dot_output[239:200] = converting10_reg;
                dot_output[199:160] = converting11_reg;
                dot_output[159:120] = converting12_reg;
                dot_output[119:80] = converting13_reg;
                dot_output[79:40] = converting14_reg;
                dot_output[39:0] = converting15_reg;
        end
        else dot_output[599:0] = dot_output[599:0];
```

```
        if (inc) begin
                read_addr = read_addr + 1;
                count = count + 1;
        end

end

always @ (state or write or count or pixel_output) begin

case (state)

idle:  begin
                dot_output_int[599:0] = 600'b0;
                done = 1'b0;
                if (write) next = converting1;
                else next = idle;
                inc = 1'b0;

                converting2_reg_int = 40'b1;
                converting3_reg_int = 40'b1;
                converting4_reg_int = 40'b1;
                converting5_reg_int = 40'b1;
                converting6_reg_int = 40'b1;
                converting7_reg_int = 40'b1;
                converting8_reg_int = 40'b1;
                converting9_reg_int = 40'b1;
                converting10_reg_int = 40'b1;
                converting11_reg_int = 40'b1;
                converting12_reg_int = 40'b1;
                converting13_reg_int = 40'b1;
                converting14_reg_int = 40'b1;
                converting15_reg_int = 40'b1;

                end
```

```
waiting:      begin
              done = 1'b0;
              inc = 1'b1;
              converting1_reg_int = converting1_reg_int;
              converting2_reg_int = converting2_reg_int;
              converting3_reg_int = converting3_reg_int;
              converting4_reg_int = converting4_reg_int;
              converting5_reg_int = converting5_reg_int;
              converting6_reg_int = converting6_reg_int;
              converting7_reg_int = converting7_reg_int;
              converting8_reg_int = converting8_reg_int;
              converting9_reg_int = converting9_reg_int;
              converting10_reg_int = converting10_reg_int;
              converting11_reg_int = converting11_reg_int;
              converting12_reg_int = converting12_reg_int;
              converting13_reg_int = converting13_reg_int;
              converting14_reg_int = converting14_reg_int;
              converting15_reg_int = converting15_reg_int;
              next = waiting2;
              end

waiting2: begin
              done = 1'b0;
              inc = 1'b0;
              converting1_reg_int = converting1_reg_int;
              converting2_reg_int = converting2_reg_int;
              converting3_reg_int = converting3_reg_int;
              converting4_reg_int = converting4_reg_int;
              converting5_reg_int = converting5_reg_int;
              converting6_reg_int = converting6_reg_int;
              converting7_reg_int = converting7_reg_int;
              converting8_reg_int = converting8_reg_int;
              converting9_reg_int = converting9_reg_int;
              converting10_reg_int = converting10_reg_int;
              converting11_reg_int = converting11_reg_int;
              converting12_reg_int = converting12_reg_int;
```

```
                converting13_reg_int = converting13_reg_int;
                converting14_reg_int = converting14_reg_int;
                converting15_reg_int = converting15_reg_int;

                if (count == 0) next = converting1;
                else if (count ==1)  next = converting2;
                else if (count ==2)  next = converting3;
                else if (count ==3)  next = converting4;
                else if (count ==4)  next = converting5;
                else if (count ==5)  next = converting6;
                else if (count ==6)  next = converting7;
                else if (count ==7)  next = converting8;
                else if (count ==8)  next = converting9;
                else if (count ==9)  next = converting10;
                else if (count ==4'd10) next = converting11;
                else if (count ==4'd11) next = converting12;
                else if (count ==4'd12) next = converting13;
                else if (count ==4'd13) next = converting14;
                else if (count ==4'd14) next = converting15;
                else next = idle;
                end


converting1:  begin
                       inc = 1'b0;
                       //dot_output_int[639:600] = dot_output_int[639:600];
                       converting1_reg_int = pixel_output_register;
                       next = idle;
                       done = 1'b1;

                converting2_reg_int = converting2_reg_int;
                converting3_reg_int = converting3_reg_int;
                converting4_reg_int = converting4_reg_int;
                converting5_reg_int = converting5_reg_int;
                converting6_reg_int = converting6_reg_int;
                converting7_reg_int = converting7_reg_int;
```

```
                converting8_reg_int = converting8_reg_int;
                converting9_reg_int = converting9_reg_int;
                converting10_reg_int = converting10_reg_int;
                converting11_reg_int = converting11_reg_int;
                converting12_reg_int = converting12_reg_int;
                converting13_reg_int = converting13_reg_int;
                converting14_reg_int = converting14_reg_int;
                converting15_reg_int = converting15_reg_int;

                        end


converting2:  begin
                        inc = 1'b0;
                        converting2_reg_int = pixel_output_register;
                        next = waiting;
                        done = 1'b0;

                converting1_reg_int = converting1_reg_int;
                converting3_reg_int = converting3_reg_int;
                converting4_reg_int = converting4_reg_int;
                converting5_reg_int = converting5_reg_int;
                converting6_reg_int = converting6_reg_int;
                converting7_reg_int = converting7_reg_int;
                converting8_reg_int = converting8_reg_int;
                converting9_reg_int = converting9_reg_int;
                converting10_reg_int = converting10_reg_int;
                converting11_reg_int = converting11_reg_int;
                converting12_reg_int = converting12_reg_int;
                converting13_reg_int = converting13_reg_int;
                converting14_reg_int = converting14_reg_int;
                converting15_reg_int = converting15_reg_int;
                        end

converting3:  begin
                        inc = 1'b0;
```

```
                converting3_reg_int = pixel_output_register;
                next = waiting;
                done = 1'b0;

            converting1_reg_int = converting1_reg_int;
            converting2_reg_int = converting2_reg_int;
            converting4_reg_int = converting4_reg_int;
            converting5_reg_int = converting5_reg_int;
            converting6_reg_int = converting6_reg_int;
            converting7_reg_int = converting7_reg_int;
            converting8_reg_int = converting8_reg_int;
            converting9_reg_int = converting9_reg_int;
            converting10_reg_int = converting10_reg_int;
            converting11_reg_int = converting11_reg_int;
            converting12_reg_int = converting12_reg_int;
            converting13_reg_int = converting13_reg_int;
            converting14_reg_int = converting14_reg_int;
            converting15_reg_int = converting15_reg_int;
                end

converting4:  begin
                inc = 1'b0;
                converting4_reg_int = pixel_output_register;
                next = waiting;
                done = 1'b0;

            converting1_reg_int = converting1_reg_int;
            converting2_reg_int = converting2_reg_int;
            converting3_reg_int = converting3_reg_int;
            converting5_reg_int = converting5_reg_int;
            converting6_reg_int = converting6_reg_int;
            converting7_reg_int = converting7_reg_int;
            converting8_reg_int = converting8_reg_int;
            converting9_reg_int = converting9_reg_int;
            converting10_reg_int = converting10_reg_int;
            converting11_reg_int = converting11_reg_int;
```

```
                converting12_reg_int = converting12_reg_int;
                converting13_reg_int = converting13_reg_int;
                converting14_reg_int = converting14_reg_int;
                converting15_reg_int = converting15_reg_int;

                        end

converting5:  begin
                        inc = 1'b0;
                        converting5_reg_int = pixel_output_register;
                        next = waiting;
                        done = 1'b0;

                converting1_reg_int = converting1_reg_int;
                converting2_reg_int = converting2_reg_int;
                converting3_reg_int = converting3_reg_int;
                converting4_reg_int = converting4_reg_int;
                converting6_reg_int = converting6_reg_int;
                converting7_reg_int = converting7_reg_int;
                converting8_reg_int = converting8_reg_int;
                converting9_reg_int = converting9_reg_int;
                converting10_reg_int = converting10_reg_int;
                converting11_reg_int = converting11_reg_int;
                converting12_reg_int = converting12_reg_int;
                converting13_reg_int = converting13_reg_int;
                converting14_reg_int = converting14_reg_int;
                converting15_reg_int = converting15_reg_int;
                        end

converting6:  begin
                        inc = 1'b0;
                        converting6_reg_int = pixel_output_register;
                        next = waiting;
                        done = 1'b0;

                converting1_reg_int = converting1_reg_int;
```

```
                converting2_reg_int = converting2_reg_int;
                converting3_reg_int = converting3_reg_int;
                converting4_reg_int = converting4_reg_int;
                converting5_reg_int = converting5_reg_int;
                converting7_reg_int = converting7_reg_int;
                converting8_reg_int = converting8_reg_int;
                converting9_reg_int = converting9_reg_int;
                converting10_reg_int = converting10_reg_int;
                converting11_reg_int = converting11_reg_int;
                converting12_reg_int = converting12_reg_int;
                converting13_reg_int = converting13_reg_int;
                converting14_reg_int = converting14_reg_int;
                converting15_reg_int = converting15_reg_int;
                        end

converting7:  begin
                        inc = 1'b0;
                        converting7_reg_int = pixel_output_register;
                        next = waiting;
                        done = 1'b0;

                converting1_reg_int = converting1_reg_int;
                converting2_reg_int = converting2_reg_int;
                converting3_reg_int = converting3_reg_int;
                converting4_reg_int = converting4_reg_int;
                converting5_reg_int = converting5_reg_int;
                converting6_reg_int = converting6_reg_int;
                converting8_reg_int = converting8_reg_int;
                converting9_reg_int = converting9_reg_int;
                converting10_reg_int = converting10_reg_int;
                converting11_reg_int = converting11_reg_int;
                converting12_reg_int = converting12_reg_int;
                converting13_reg_int = converting13_reg_int;
                converting14_reg_int = converting14_reg_int;
                converting15_reg_int = converting15_reg_int;
                        end
```

```
converting8:  begin
                      inc = 1'b0;
                      converting8_reg_int = pixel_output_register;
                      next = waiting;
                      done = 1'b0;

                converting1_reg_int = converting1_reg_int;
                converting2_reg_int = converting2_reg_int;
                converting3_reg_int = converting3_reg_int;
                converting4_reg_int = converting4_reg_int;
                converting5_reg_int = converting5_reg_int;
                converting6_reg_int = converting6_reg_int;
                converting7_reg_int = converting7_reg_int;
                converting9_reg_int = converting9_reg_int;
                converting10_reg_int = converting10_reg_int;
                converting11_reg_int = converting11_reg_int;
                converting12_reg_int = converting12_reg_int;
                converting13_reg_int = converting13_reg_int;
                converting14_reg_int = converting14_reg_int;
                converting15_reg_int = converting15_reg_int;
                      end

converting9:  begin
                      inc = 1'b0;
                      converting9_reg_int = pixel_output_register;
                      next = waiting;
                      done = 1'b0;

                converting1_reg_int = converting1_reg_int;
                converting2_reg_int = converting2_reg_int;
                converting3_reg_int = converting3_reg_int;
                converting4_reg_int = converting4_reg_int;
                converting5_reg_int = converting5_reg_int;
                converting6_reg_int = converting6_reg_int;
                converting7_reg_int = converting7_reg_int;
```

```
                converting8_reg_int = converting8_reg_int;
                converting10_reg_int = converting10_reg_int;
                converting11_reg_int = converting11_reg_int;
                converting12_reg_int = converting12_reg_int;
                converting13_reg_int = converting13_reg_int;
                converting14_reg_int = converting14_reg_int;
                converting15_reg_int = converting15_reg_int;
                        end

converting10:           begin
                        inc = 1'b0;
                        converting10_reg_int = pixel_output_register;
                        next = waiting;
                        done = 1'b0;

                converting1_reg_int = converting1_reg_int;
                converting2_reg_int = converting2_reg_int;
                converting3_reg_int = converting3_reg_int;
                converting4_reg_int = converting4_reg_int;
                converting5_reg_int = converting5_reg_int;
                converting6_reg_int = converting6_reg_int;
                converting7_reg_int = converting7_reg_int;
                converting8_reg_int = converting8_reg_int;
                converting9_reg_int = converting9_reg_int;
                converting11_reg_int = converting11_reg_int;
                converting12_reg_int = converting12_reg_int;
                converting13_reg_int = converting13_reg_int;
                converting14_reg_int = converting14_reg_int;
                converting15_reg_int = converting15_reg_int;
                        end

converting11:           begin
                        inc = 1'b0;
                        converting11_reg_int = pixel_output_register;
                        next = waiting;
                        done = 1'b0;
```

```
                converting1_reg_int = converting1_reg_int;
                converting2_reg_int = converting2_reg_int;
                converting3_reg_int = converting3_reg_int;
                converting4_reg_int = converting4_reg_int;
                converting5_reg_int = converting5_reg_int;
                converting6_reg_int = converting6_reg_int;
                converting7_reg_int = converting7_reg_int;
                converting8_reg_int = converting8_reg_int;
                converting9_reg_int = converting9_reg_int;
                converting10_reg_int = converting10_reg_int;
                converting12_reg_int = converting12_reg_int;
                converting13_reg_int = converting13_reg_int;
                converting14_reg_int = converting14_reg_int;
                converting15_reg_int = converting15_reg_int;
                        end

converting12:           begin
                        inc = 1'b0;
                        converting12_reg_int = pixel_output_register;
                        next = waiting;
                        done = 1'b0;

                converting1_reg_int = converting1_reg_int;
                converting2_reg_int = converting2_reg_int;
                converting3_reg_int = converting3_reg_int;
                converting4_reg_int = converting4_reg_int;
                converting5_reg_int = converting5_reg_int;
                converting6_reg_int = converting6_reg_int;
                converting7_reg_int = converting7_reg_int;
                converting8_reg_int = converting8_reg_int;
                converting9_reg_int = converting9_reg_int;
                converting10_reg_int = converting10_reg_int;
                converting11_reg_int = converting11_reg_int;
                converting13_reg_int = converting13_reg_int;
                converting14_reg_int = converting14_reg_int;
                converting15_reg_int = converting15_reg_int;
```

```
                    end

converting13:       begin
                    inc = 1'b0;
                    converting13_reg_int = pixel_output_register;
                    next = waiting;
                    done = 1'b0;

                converting1_reg_int = converting1_reg_int;
                converting2_reg_int = converting2_reg_int;
                converting3_reg_int = converting3_reg_int;
                converting4_reg_int = converting4_reg_int;
                converting5_reg_int = converting5_reg_int;
                converting6_reg_int = converting6_reg_int;
                converting7_reg_int = converting7_reg_int;
                converting8_reg_int = converting8_reg_int;
                converting9_reg_int = converting9_reg_int;
                converting10_reg_int = converting10_reg_int;
                converting11_reg_int = converting11_reg_int;
                converting12_reg_int = converting12_reg_int;
                converting14_reg_int = converting14_reg_int;
                converting15_reg_int = converting15_reg_int;
                    end

converting14:       begin
                    inc = 1'b0;
                    converting14_reg_int = pixel_output_register;
                    next = waiting;
                    done = 1'b0;

                converting1_reg_int = converting1_reg_int;
                converting2_reg_int = converting2_reg_int;
                converting3_reg_int = converting3_reg_int;
                converting4_reg_int = converting4_reg_int;
                converting5_reg_int = converting5_reg_int;
                converting6_reg_int = converting6_reg_int;
```

```
                converting7_reg_int = converting7_reg_int;
                converting8_reg_int = converting8_reg_int;
                converting9_reg_int = converting9_reg_int;
                converting10_reg_int = converting10_reg_int;
                converting11_reg_int = converting11_reg_int;
                converting12_reg_int = converting12_reg_int;
                converting13_reg_int = converting13_reg_int;
                converting15_reg_int = converting15_reg_int;
                        end

converting15:           begin
                        inc = 1'b0;
                        done =1'b1;
                        converting15_reg_int = pixel_output_register;
                        next = idle;

                converting1_reg_int = converting1_reg_int;
                converting2_reg_int = converting2_reg_int;
                converting3_reg_int = converting3_reg_int;
                converting4_reg_int = converting4_reg_int;
                converting5_reg_int = converting5_reg_int;
                converting6_reg_int = converting6_reg_int;
                converting7_reg_int = converting7_reg_int;
                converting8_reg_int = converting8_reg_int;
                converting9_reg_int = converting9_reg_int;
                converting10_reg_int = converting10_reg_int;
                converting11_reg_int = converting11_reg_int;
                converting12_reg_int = converting12_reg_int;
                converting13_reg_int = converting13_reg_int;
                converting14_reg_int = converting14_reg_int;
                        end

default:                begin
                                next = idle;
                                                converting2_reg_int = 40'b1;
                converting3_reg_int = 40'b1;
```

```
                converting4_reg_int = 40'b1;
                converting5_reg_int = 40'b1;
                converting6_reg_int = 40'b1;
                converting7_reg_int = 40'b1;
                converting8_reg_int = 40'b1;
                converting9_reg_int = 40'b1;
                converting10_reg_int = 40'b1;
                converting11_reg_int = 40'b1;
                converting12_reg_int = 40'b1;
                converting13_reg_int = 40'b1;
                converting14_reg_int = 40'b1;
                converting15_reg_int = 40'b1;

                        end


endcase
end
endmodule


hex_to_char.v
module hex_to_char(reset, clk, hex_input, pixel_output);input reset, clk;input[15:0]
hex_input;output
[39:0] pixel_output;   reg[39:0] pixel_output;
//thanks to Eleanor for most of these
parameter displayA = 40'b01111000_00010110_00010011_00010110_01111000; // 'A'parameter
displayB =
40'b01111111_01001001_01001001_01001001_00110110; // 'B'parameter displayC =
40'b00111110_01000001_01000001_01000001_00100010; // 'C'parameter displayD =
40'b01111111_01000001_01000001_01000001_00111110; // 'D'parameter displayE =
40'b01111111_01001001_01001001_01001001_01001001; // 'E'parameter displayF =
40'b01111111_00001001_00001001_00001001_00001001; // 'F'parameter displayG =
40'b00111110_01000001_01010001_00110001_01010010; // 'G'parameter displayH =
40'b01111111_00001000_00001000_00001000_01111111; // 'H'parameter displayI =
40'b01000001_01000001_01111111_01000001_01000001; // 'I'parameter displayJ =
40'b00110000_01000000_01000000_01000000_00111111; // 'J'parameter displayK =
40'b01111111_00001000_00010100_00100010_01000001; // 'K'parameter displayL =
40'b01111111_01000000_01000000_01000000_01000000; // 'L'parameter displayM =
40'b01111111_00000010_00001100_00000010_01111111; // 'M'parameter displayN =
```

```
40'b01111111_00000110_00001100_00110000_01111111; // 'N'parameter displayO =

40'b00111110_01000001_01000001_01000001_00111110; // 'O'parameter displayP =

40'b01111111_00001001_00001001_00001001_00000110; // 'P'parameter displayQ =

40'b00111110_01000001_01010001_00111110_00100000; // 'Q'parameter displayR =

40'b01111111_00001001_00011001_01101001_01000110; // 'R'parameter displayS =

40'b01000110_01001001_01001001_01001001_00110001; // 'S'parameter displayT =

40'b00000001_00000001_01111111_00000001_00000001; // 'T'parameter displayU =

40'b00111111_01000000_01000000_01000000_00111111; // 'U'parameter displayV =

40'b00000111_00011000_01100000_00011000_00000111; // 'V'parameter displayW =

40'b00011111_01100000_00111000_01100000_00011111; // 'W'parameter displayX =

40'b01100011_00010100_00001000_00010100_01100011; // 'X'parameter displayY =

40'b00000111_00001000_01110000_00001000_00000111; // 'Y'parameter displayZ =

40'b01000100_01100100_01010100_01001100_01000100; // 'Z'parameter displayspace =

40'b00000000_00000000_00000000_00000000_00000000; // 'parameter displaycolon =

40'b00000000_00110110_00110110_00000000_00000000; //parameter displaydash =

40'b00000000_00001100_00001100_00001100_00000000; //parameter display0 =

40'b00111110_01000001_01000001_01000001_00111110; // 'O'parameter display1 =

40'b00000000_01000010_01111111_01000000_00000000; // '1'parameter display2 =

40'b00100010_00110001_00101001_00100101_00100010; // '2'parameter display3 =

40'b01001001_01001001_01001001_01001001_00110110; // '3'parameter display4 =

40'b00001111_00001000_00001000_00001000_01111111; // '4'parameter display5 =

40'b01001111_01001001_01001001_01001001_00110001; // '5'parameter display6 =

40'b00111110_01001001_01001001_01001001_00110001; // '6'parameter display7 =

40'b00000001_00000001_00000001_00000001_01111111; // '7'parameter display8 =

40'b00111110_01001001_01001001_01001001_00110110; // '8'parameter display9 =

40'b00000110_00001001_00001001_00001001_01111110; // '9'always @ (posedge clk) begin if
(reset)

pixel_output = 40'b0;else if ((hex_input==65) | (hex_input==97)) pixel_output =
displayA;else if

((hex_input==66)| (hex_input==98)) pixel_output = displayB;else if ((hex_input==67)|
(hex_input==99))

pixel_output = displayC;else if ((hex_input==68)| (hex_input==100)) pixel_output =
displayD;else if

((hex_input==69)| (hex_input==101)) pixel_output = displayE;else if ((hex_input==70)|
(hex_input==102))

pixel_output = displayF;else if ((hex_input==71)| (hex_input==103)) pixel_output =
displayG;else if

((hex_input==72)| (hex_input==104)) pixel_output = displayH;else if ((hex_input==73)|
(hex_input==105))
```

```
pixel_output = displayI;else if ((hex_input==74)| (hex_input==106)) pixel_output =
displayJ;else if

((hex_input==75)| (hex_input==107)) pixel_output = displayK;else if ((hex_input==76)|
(hex_input==108))

pixel_output = displayL;else if ((hex_input==77)| (hex_input==109)) pixel_output =
displayM;else if

((hex_input==78)| (hex_input==110)) pixel_output = displayN;else if ((hex_input==79)|
(hex_input==111))

pixel_output = displayO;else if ((hex_input==80)| (hex_input==112)) pixel_output =
displayP;else if

((hex_input==81)| (hex_input==113)) pixel_output = displayQ;else if ((hex_input==82)|
(hex_input==114))

pixel_output = displayR;else if ((hex_input==83)| (hex_input==115)) pixel_output =
displayS;else if

((hex_input==84)| (hex_input==116)) pixel_output = displayT;else if ((hex_input==85)|
(hex_input==117))

pixel_output = displayU;else if ((hex_input==86)| (hex_input==118)) pixel_output =
displayV;else if

((hex_input==87)| (hex_input==119)) pixel_output = displayW;else if ((hex_input==88)|
(hex_input==120))

pixel_output = displayX;else if ((hex_input==89)| (hex_input==121)) pixel_output =
displayY;else if

((hex_input==90)| (hex_input==122)) pixel_output = displayZ;else if (hex_input==48)
pixel_output =

display0;else if (hex_input==49) pixel_output = display1;else if (hex_input==50)
pixel_output =

display2;else if (hex_input==51) pixel_output = display3;else if (hex_input==52)
pixel_output =

display4;else if (hex_input==53) pixel_output = display5;else if (hex_input==54)
pixel_output =

display6;else if (hex_input==55) pixel_output = display7;else if (hex_input==56)
pixel_output =

display8;else if (hex_input==57) pixel_output = display9;else if (hex_input==58)
pixel_output =

displaycolon;else if (hex_input==45) pixel_output = displaydash;else pixel_output =

displayspace;endendmodule


title SRAM

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

*     This file is owned and controlled by Xilinx and must be used             *

*     solely for design, simulation, implementation and creation of            *

*     design files limited to Xilinx devices or technologies. Use              *

*     with non-Xilinx devices or technologies is expressly prohibited          *

*     and immediately terminates your license.                                 *

*                                                                              *
```

*     XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS"           *
*     SOLELY FOR USE IN DEVELOPING PROGRAMS AND SOLUTIONS FOR                 *
*     XILINX DEVICES.  BY PROVIDING THIS DESIGN, CODE, OR INFORMATION         *
*     AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE, APPLICATION             *
*     OR STANDARD, XILINX IS MAKING NO REPRESENTATION THAT THIS               *
*     IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT,                 *
*     AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE        *
*     FOR YOUR IMPLEMENTATION.  XILINX EXPRESSLY DISCLAIMS ANY                *
*     WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE                 *
*     IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR          *
*     REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF         *
*     INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS         *
*     FOR A PARTICULAR PURPOSE.                                               *
*                                                                             *
*     Xilinx products are not intended for use in life support                *
*     appliances, devices, or systems. Use in such applications are           *
*     expressly prohibited.                                                   *
*                                                                             *
*     (c) Copyright 1995-2004 Xilinx, Inc.                                    *
*     All rights reserved.                                                    *
*******************************************************************************/
// The synopsys directives "translate_off/translate_on" specified below are
// supported by XST, FPGA Compiler II, Mentor Graphics and Synplicity synthesis
// tools. Ensure they are correct for your synthesis tool(s).

// You must compile the wrapper file title_sram.v when simulating
// the core, title_sram. 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 title_sram (
       addr,
       clk,
       din,
       dout,
       we);    // synthesis black_box
```

```
input [7 : 0] addr;
input clk;
input [15 : 0] din;
output [15 : 0] dout;
input we;

// synopsys translate_off

        BLKMEMSP_V6_1 #(
                8,      // c_addr_width
                "0",    // c_default_data
                150,    // c_depth
                0,      // c_enable_rlocs
                0,      // 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
                "title_sram.mif",     // c_mem_init_file
                0,      // c_pipe_stages
                0,      // c_reg_inputs
                "0",    // c_sinit_value
                16,     // c_width
                0,      // c_write_mode
                "0",    // c_ybottom_addr
                1,      // c_yclk_is_rising
                1,      // c_yen_is_high
                "hierarchy1",         // c_yhierarchy
                0,      // c_ymake_bmm
                "16kx1",      // c_yprimitive_type
```

```
            1,     // c_ysinit_is_high
            "1024",       // c_ytop_addr
            0,     // c_yuse_single_primitive
            1,     // c_ywe_is_high
            1)     // c_yydisable_warnings
      inst (
            .ADDR(addr),
            .CLK(clk),
            .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 title_sram is "true"

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

endmodule


testbenches
`timescale 1ns/10ps
module test_hex_to_char(pixel_output);
   output [39:0]pixel_output;
   reg reset, clock_27mhz;
```

```
   reg [15:0] counter;

   initial begin
   reset = 1'b1;
   clock_27mhz = 1'b0;
   counter = 15'd65;
   end

   always #270 clock_27mhz = ~clock_27mhz;

hex_to_char my_hex_to_char (reset, clock_27mhz, counter, pixel_output);

always @ (posedge clock_27mhz) begin
reset = 1'b0;
counter = counter + 1;
end

endmodule
`timescale 1ns/10ps
module title_wr_tester(dot_output, read_addr, done);

output [639:0]dot_output;
output [7:0] read_addr;
output done;

   reg reset, clock_27mhz;
   reg [7:0] counter;
   reg write;

   initial begin
   reset = 1'b1;
   write = 1'b0;
   clock_27mhz = 1'b0;
   counter = 7'b0;
   end
```

```
   always #270 clock_27mhz = ~clock_27mhz;

title_to_leds my_title_to_leds(reset, clock_27mhz, write, 1, 1, 40'h00FF00FF00,
read_addr, dot_output, done);

always @ (posedge clock_27mhz) begin

if (counter == 7'd3) write = 1'b1;
else write = 1'b0;
if (counter < 7'd3) reset = 1'b1;
else reset = 1'b0;

counter = counter + 1;

end

endmodule

module higher_test(dot_output, done);

output [639:0]dot_output;
output done;

   reg reset, clock_27mhz;
   reg [7:0] counter;
   reg write;

   initial begin
   reset = 1'b1;
   write = 1'b0;
   clock_27mhz = 1'b0;
   counter = 7'b0;
   end

   always #270 clock_27mhz = ~clock_27mhz;
```

```
   wire [3:0] song_playing, song_displaying;
   assign song_playing = 4'b0;
   assign song_displaying = 4'b0;
   wire [39:0]pixel_output;
   wire [7:0] read_addr;
   wire we;
   assign we = 1'b0;
   wire [15:0]hex_input;

title_to_leds my_title_to_leds(reset, clock_27mhz, write, song_playing, song_displaying,
pixel_output,
read_addr, dot_output, done);

hex_to_char my_hex_to_char(reset, clock_27mhz, hex_input, pixel_output);

title_sram my_title_sram(
       read_addr,
       clock_27mhz,
       din,
       hex_input,
       we);

always @ (posedge clock_27mhz) begin

if (counter == 7'd3) write = 1'b1;
else write = 1'b0;
if (counter < 7'd3) reset = 1'b1;
else reset = 1'b0;

counter = counter + 1;

end

endmodule
module display_test(clk, reset, hsync, vsync, hblank, vblank);
```

```
input clk, reset;
output hsync, vsync, hblank, vblank;

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


wire hsyncon, hsyncoff, hreset, hblankon;
wire vsyncon, vsyncoff, vreset, vblankon;

// 640 X 480, 60 Hz format, 25.175 MHz pixel clock
// Timing parameters
parameter row_width = 639;
parameter hsync_start = 655;
parameter hsync_end = 751;
parameter row_period = 799;
parameter frame_height = 479;
parameter vsync_start = 490;
parameter vsync_end = 492;
parameter frame_period = 523;

// display row_width+1 pixels per line
assign hblankon = (hcount == row_width); // turn on blanking
assign hsyncon = (hcount == hsync_start); // sync active
assign hsyncoff = (hcount == hsync_end); // sync inactive
assign hreset = (hcount == row_period) || reset; // new row

// display frame_height lines
assign vblankon = hreset & (v_count == frame_height); // turn on blanking
assign vsyncon = hreset & (v_count == vsync_start); // sync active
assign vsyncoff = hreset & (v_count == vsync_end); // sync inactive
assign vreset = hreset & (v_count == frame_period) || reset; // new frame

always @(posedge clk)
begin
// On reset, clear all counter and flag state.
       if (reset)
              begin
              hsync <= 1; hblank <= 0; hcount <= 0;
              vsync <= 1; vblank <= 0; v_count <= 0;
              end

// Otherwise, update counters and flags on every clock edge

       else begin
              hcount <= hreset ? 0 : hcount + 1;
              hblank <= hreset ? 0 : hblankon ? 1 : hblank;
              hsync <= hsyncon ? 0 : hsyncoff ? 1 : hsync;
              v_count <= hreset ? (vreset ? 0 : v_count + 1) : v_count;
              vblank <= vreset ? 0 : vblankon ? 1 : vblank;
              vsync <= vsyncon ? 0 : vsyncoff ? 1 : vsync;
       end
end

endmodule
```