

MASSACHUSETTS INSTITUTE OF TECHNOLOGY  
6.111 Introductory Digital Systems Lab  
Final Project Code Appendix

Masood Qazi  
Zhongying Zhou

May 18, 2006

**Contents**

<b>A Verilog Code</b>	<b>1</b>
A.1 labkit.v . . . . .	1
A.2 audio.v . . . . .	15
A.3 fft_subsystem.v . . . . .	20
A.4 synth.v . . . . .	26
A.5 k_to_note.v . . . . .	30
A.6 harmonic_check.v . . . . .	33
A.7 dots_num_2x.v . . . . .	33
A.8 dots_note_2x.v . . . . .	35
A.9 alphanumeric_displays.v . . . . .	39
A.10 audio_sel.v . . . . .	39
A.11 bcontrol.v . . . . .	39
A.12 fall2pu.v . . . . .	40
A.13 debounce.v . . . . .	40
A.14 checker.v . . . . .	41
A.15 vgacon.v . . . . .	42
A.16 compare_display.v . . . . .	46
A.17 divider.v . . . . .	57
A.18 draw_fft.v . . . . .	58
A.19 draw_graph.v . . . . .	60
A.20 mr_fsm.v . . . . .	62
A.21 rectangle.v . . . . .	63

**A Verilog Code****A.1 labkit.v**

```

////////////////////////////////////
// 6.111 final project by:
// MASOOD QAZI and ZHONGYING ZHOU
// TEAM 9
// MAY2006
//
////////////////////////////////////
//
// 6.111 FPGA Labkit -- Template Toplevel Module
//
// For Labkit Revision 004
//
//
// Created: October 31, 2004, from revision 003 file
// Author: Nathan Ickes
//
////////////////////////////////////
//
// CHANGES FOR BOARD REVISION 004
//
// 1) Added signals for logic analyzer pods 2-4.
// 2) Expanded "tv_in_ycrcb" to 20 bits.

```

```

// 3) Renamed "tv_out_data" to "tv_out_i2c_data" and "tv_out_sclk" to
// "tv_out_i2c_clock".
// 4) Reversed disp_data_in and disp_data_out signals, so that "out" is an
// output of the FPGA, and "in" is an input.
//
// CHANGES FOR BOARD REVISION 003
//
// 1) Combined flash chip enables into a single signal, flash_ce_b.
//
// CHANGES FOR BOARD REVISION 002
//
// 1) Added SRAM clock feedback path input and output
// 2) Renamed "mousedata" to "mouse_data"
// 3) Renamed some ZBT memory signals. Parity bits are now incorporated
// into
// the data bus, and the byte write enables have been combined into the
// 4-bit ram#_bwe_b bus.
// 4) Removed the "systemace_clock" net, since the SystemACE clock is now
// hardwired on the PCB to the oscillator.
//
////////////////////////////////////
//
// Complete change history (including bug fixes)
//
// 2006-Mar-08: Corrected default assignments to "vga_out_red", "vga_out_green"
// and "vga_out_blue". (Was 10'h0, now 8'h0.)
//
// 2005-Sep-09: Added missing default assignments to "ac97_sdata_out",
// "disp_data_out", "analyzer[2-3]_clock" and
// "analyzer[2-3]_data".
//
// 2005-Jan-23: Reduced flash address bus to 24 bits, to match 128Mb devices
// actually populated on the boards. (The boards support up to
// 256Mb devices, with 25 address lines.)
//
// 2004-Oct-31: Adapted to new revision 004 board.
//
// 2004-May-01: Changed "disp_data_in" to be an output, and gave it a default
// value. (Previous versions of this file declared this port to
// be an input.)
//
// 2004-Apr-29: Reduced SRAM address busses to 19 bits, to match 18Mb devices
// actually populated on the boards. (The boards support up to
// 72Mb devices, with 21 address lines.)
//
// 2004-Apr-29: Change history started
//
////////////////////////////////////

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

```

```
vga_out_red, vga_out_green, vga_out_blue, vga_out_sync_b,  
vga_out_blank_b, vga_out_pixel_clock, vga_out_hsync,  
vga_out_vsync,  
  
tv_out_ycrfb, 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_ycrfb, 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_ycrfb;
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_ycrfb;
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;
/* Commented out by masood qazi:
assign audio_reset_b = 1'b0;
assign ac97_synch = 1'b0;
assign ac97_sdata_out = 1'b0;
*/
// ac97_sdata_in is an input

// VGA Output
/* commented out by masood qazi
assign vga_out_red = 8'h0;
assign vga_out_green = 8'h0;
assign vga_out_blue = 8'h0;
assign vga_out_sync_b = 1'b1;
assign vga_out_blank_b = 1'b1;
assign vga_out_pixel_clock = 1'b0;
assign vga_out_hsync = 1'b0;
assign vga_out_vsync = 1'b0;
*/

// Video Output
assign tv_out_ycrcb = 10'h0;
assign tv_out_reset_b = 1'b0;
assign tv_out_clock = 1'b0;
assign tv_out_i2c_clock = 1'b0;
assign tv_out_i2c_data = 1'b0;
assign tv_out_pal_ntsc = 1'b0;

```

```
assign tv_out_hsync_b = 1'b1;
assign tv_out_vsync_b = 1'b1;
assign tv_out_blank_b = 1'b1;
assign tv_out_subcar_reset = 1'b0;

// Video Input
assign tv_in_i2c_clock = 1'b0;
assign tv_in_fifo_read = 1'b0;
assign tv_in_fifo_clock = 1'b0;
assign tv_in_iso = 1'b0;
assign tv_in_reset_b = 1'b0;
assign tv_in_clock = 1'b0;
assign tv_in_i2c_data = 1'bZ;
// tv_in_ycrb, 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
/* commented out by masood qazi
assign disp_blank = 1'b1;
assign disp_clock = 1'b0;
assign disp_rs = 1'b0;
assign disp_ce_b = 1'b1;
assign disp_reset_b = 1'b0;
assign disp_data_out = 1'b0;
*/
// disp_data_in is an input

// Buttons, Switches, and Individual LEDs
// commented out by masood qazi:
// 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
/* commented out by masood qazi ...
assign analyzer1_data = 16'h0;
assign analyzer1_clock = 1'b1;
assign analyzer2_data = 16'h0;
assign analyzer2_clock = 1'b1;
assign analyzer3_data = 16'h0;
assign analyzer3_clock = 1'b1;
assign analyzer4_data = 16'h0;
assign analyzer4_clock = 1'b1;
*/
```

```

////////////////////////////////////
// Begin code for 6.111 final project by
// MASOOD QAZI and ZHONGYING ZHOU
// TEAM # 9
// MAY 2006
//

////////////////////////////////////
// CLOCK MULTIPLICATION FOR REST OF SYSTEM (MQ)
// produces 36.864MHz pixel_clock from 12.288MHz ac97_bit_clock
// every third falling edge of the pixel_clock is aligned to the
// rising edge of ac97_bit_clock ... this eliminates the need
// for synchronizers

wire pixel_clock, clk_ac97, CLKFX_OUT, LOCKED_OUT;

mydcm pixel_clock_dcm (
    .CLKIN_IN(ac97_bit_clock),
    .CLKFX_OUT(CLKFX_OUT),
    .CLKFX180_OUT(pixel_clock),
    .CLKO_OUT(clk_ac97),
    .LOCKED_OUT(LOCKED_OUT)
);

////////////////////////////////////
// BUTTON SYNCHRONIZATION:
// note the ac97 has a separate reset for startup issues
// all other buttons are synchronized to the master clock: pixel_clock

debounce debounce_reset_ac97(.reset(1'b1), .clock(clock_27mhz),
    .noisy(user1[0]), .clean(audio_reset_b));
defparam debounce_reset_ac97.DELAY = 270000; // 0.01 sec with 27mhz

wire reset_sync_b, reset_sync;
debounce debounce_reset(.reset(1'b1), .clock(pixel_clock),
    .noisy(button0), .clean(reset_sync_b));
assign reset_sync = ~reset_sync_b;

wire outv_up_sync_b;
debounce debounce_outv_up(.reset(reset_sync_b), .clock(pixel_clock),
    .noisy(button_enter), .clean(outv_up_sync_b));
wire outv_dn_sync_b;
debounce debounce_outv_dn(.reset(reset_sync_b), .clock(pixel_clock),
    .noisy(button3), .clean(outv_dn_sync_b));
wire micv_up_sync_b;
debounce debounce_micv_up(.reset(reset_sync_b), .clock(pixel_clock),
    .noisy(button2), .clean(micv_up_sync_b));
wire micv_dn_sync_b;
debounce debounce_micv_dn(.reset(reset_sync_b), .clock(pixel_clock),
    .noisy(button1), .clean(micv_dn_sync_b));

```

```

// The following buttons and switches exhibit two different behaviors
// depending on whether the "testmode_sync" signal is high or low:

wire testmode_sync, testmode_sync_b;
debounce debounce_testmode(.reset(reset_sync_b), .clock(pixel_clock),
                           .noisy(switch[3]), .clean(testmode_sync));
assign testmode_sync_b = ~testmode_sync;

// to gate an active low signal, OR it with its "enable_bar"
// to gate an active high signal, AND it with its "enable"

wire button_right_sync, fforward_sync, note_up_sync_b;
debounce debounce_butn_right(.reset(reset_sync_b), .clock(pixel_clock),
                             .noisy(button_right), .clean(button_right_sync));
assign note_up_sync_b = button_right_sync || testmode_sync_b;
assign fforward_sync = ~button_right_sync && testmode_sync_b;

wire button_left_sync, rewind_sync, note_dn_sync_b;
debounce debounce_butn_left(.reset(reset_sync_b), .clock(pixel_clock),
                             .noisy(button_left), .clean(button_left_sync));
assign note_dn_sync_b = button_left_sync || testmode_sync_b;
assign rewind_sync = ~button_left_sync && testmode_sync_b;

wire button_up_sync, start_sync, octv_up_sync_b;
debounce debounce_butn_up(.reset(reset_sync_b), .clock(pixel_clock),
                           .noisy(button_up), .clean(button_up_sync));
assign octv_up_sync_b = button_up_sync || testmode_sync_b;
assign start_sync = ~button_up_sync && testmode_sync_b;

wire button_down_sync, pause_sync, octv_dn_sync_b;
debounce debounce_butn_down(.reset(reset_sync_b), .clock(pixel_clock),
                              .noisy(button_down), .clean(button_down_sync));
assign octv_dn_sync_b = button_down_sync || testmode_sync_b;
assign pause_sync = (~button_down_sync) || testmode_sync;

////////////////////////////////////
// AUDIO WIRING (MQ) --- note different clock: clk_ac97

wire signed [15:0] audio_in, audio_out;
wire [7:0] bitcount;

// for probing
wire [4:0] micv, outv;

// instantiate LM4550 (ac97) controller:
audio audio0 ( .resetb(reset_sync_b), .clk_ac97(clk_ac97),
               .sdata_in(ac97_sdata_in), .audio_out(audio_out),
               .outv_up_b(outv_up_sync_b), .outv_dn_b(outv_dn_sync_b),
               .micv_up_b(micv_up_sync_b), .micv_dn_b(micv_dn_sync_b),
               .sdata_out(ac97_sdata_out), .audio_in(audio_in),

```

```

        .sync(ac97_synch),
        .bitcount(bitcount),
        //probing outputs from audio.v
        .micv(micv), .outv(outv));

// generate en48k pulse, single-clock-cycle long pulse:
wire    monitor;
assign monitor = (bitcount == 8'd77)? 1'b1 : 1'b0;

reg     old_monitor, old_old_monitor;
always @ (posedge pixel_clock) begin
    old_monitor <= monitor;
    old_old_monitor <= old_monitor;
end

// detect a rising edge with a pulse synchronous to pixel_clock
wire    en48k;
assign en48k = (old_monitor && (!old_old_monitor))? 1'b1 : 1'b0;

// *note* this is with respect to pixel_clock, NOT clk_ac97
// also *note* that bitcount is clocked by clk_ac97 but its
// transitions are synchronous with pixel_clock thanks to DCM

////////////////////////////////////
// SYNTHESIZER & AUDIO OUTPUT (MQ)

wire signed [15:0] tone;
wire [7:0] music_note;

// if "testmode_sync" is high, the following button controls
// determine the octave and note for the synthesizer input
// otherwise ZZ's memory reader module determines the octave
// and note for the synthesizer module. Also see in ZZ's section
// for the conditional assignment statement for music_note[7:0]
wire [4:0] note_sel;
bcontrol bcontrol_note( .resetb(reset_sync_b),
                       .clk(pixel_clock),
                       .incb(note_up_sync_b),
                       .decb(note_dn_sync_b),
                       .val(note_sel));
defparam bcontrol_note.delta = 5'b00010;
defparam bcontrol_note.init_val = {4'd0, 1'b0};

wire [4:0] octv_sel;
bcontrol bcontrol_octv( .resetb(reset_sync_b),
                       .clk(pixel_clock),
                       .incb(octv_up_sync_b),
                       .decb(octv_dn_sync_b),
                       .val(octv_sel));
defparam bcontrol_octv.delta = 5'b00010;
defparam bcontrol_octv.init_val = {4'd5, 1'b0};

```

```

synth synth0( .resetb(reset_sync_b),
              .clk(pixel_clock),
              .note(music_note[3:0]),
              .octave(music_note[7:4]),
              .tone(tone));

wire          signed [15:0] voice; // 2's complement

audio_sel audio_sel_out(.sel(switch[7:6]),
                       .a(tone),
                       .b(voice),
                       .out(audio_out));

/////////////////////////////////////////////////////////////////
//      FFT  (MQ)

wire          clk, resetb;
wire signed [15:0] fft_data; // 2's complement
wire fft_disp;
wire [6:0] fft_out; // unsigned magnitude
wire [7:0] singer_note;
wire signed [15:0] offset_accum; // 2's complement
wire [11:0] xk_index, kmax;

assign        voice = audio_in - offset_accum; // subtract out DC

audio_sel audio_sel_fft(.sel(switch[5:4]),
                       .a(tone),
                       .b(voice),
                       .out(fft_data));

fft_subsystem fft_subsystem( .resetb(reset_sync_b),
                             .clk(pixel_clock),
                             .en48k(en48k),
                             .fft_data(fft_data),
                             .fft_disp(fft_disp),
                             .fft_out(fft_out),
                             .singer_note(singer_note),
                             .offset_accum(offset_accum),
                             .xk_index(xk_index),
                             .kmax(kmax));

/////////////////////////////////////////////////////////////////
// ALPHANUMERIC DISPLAY OF singer_note (left) AND music_note (right)
// (MQ)
wire [639:0] mydots;
assign        mydots[479:160] = 320'b0;
dots_note_2x dotsnum_singernote(pixel_clock, singer_note[3:0],
                               mydots[639:560]);

```

```

dots_num_2x dotsnum_singeroctave(pixel_clock, singer_note[7:4],
                                mydots[559:480]);
dots_note_2x dotsnum_musicnote(pixel_clock, music_note[3:0],
                                mydots[159:80]);
dots_num_2x dotsnum_musicoctave(pixel_clock, music_note[7:4],
                                mydots[79:0]);
alphanumeric_displays alpha_disp0(.global_clock(pixel_clock),
                                   .manual_reset(1'b0),
                                   .disp_test(1'b0),
                                   .disp_blank(disp_blank),
                                   .disp_clock(disp_clock),
                                   .disp_rs(disp_rs),
                                   .disp_ce_b(disp_ce_b),
                                   .disp_reset_b(disp_reset_b),
                                   .disp_data_out(disp_data_out),
                                   .dots(mydots));

////////////////////////////////////
// VIDEO CONTROLLER VGA with 36.864MHz pixel_clock

wire          hsync, vsync;
wire [10:0]   pixel_count, line_count;

assign        vga_out_pixel_clock = ~pixel_clock;

// two delay registers for each sync signal
reg          vga_out_hsync, hsync_int, vga_out_vsync, vsync_int;
always @ (posedge pixel_clock)
begin
    hsync_int <= hsync;
    vga_out_hsync <= hsync_int;
    vsync_int <= vsync;
    vga_out_vsync <= vsync_int;
end

// 800x600 VGA
vgacon vgacon0 (.clk(pixel_clock), .resetb(reset_sync_b), .hsync(hsync),
               .vsync(vsync), .pixel_count(pixel_count[10:0]),
               .line_count(line_count[10:0]),
               .vga_out_sync_b(vga_out_sync_b),
               .vga_out_blank_b(vga_out_blank_b));

////////////////////////////////////
// video testing ...
// checker checker0 (.clk(pixel_clock), .resetb(reset_sync_b),
//                  .pixel_count(pixel_count[9:0]),
//                  .line_count(line_count[9:0]),
//                  .vga_out_red(vga_out_red),
//                  .vga_out_green(vga_out_green),
//                  .vga_out_blue(vga_out_blue));
// ... video testing

```

```

////////////////////////////////////
//  assign vga_out_red = 8'h0;
//  assign vga_out_green = 8'h0;
//  assign vga_out_blue = 8'h0;
//  assign vga_out_sync_b = 1'b1;          ... yes wired by vgacon0
//  assign vga_out_blank_b = 1'b1;        ... yes wired by vgacon0
//  assign vga_out_pixel_clock = 1'b0;    ... yes wired by vgacon0
//  assign vga_out_hsync = 1'b0;          ... yes wired by vgacon0
//  assign vga_out_vsync = 1'b0;          ... yes wired by vgacon0
////////////////////////////////////
// BEGIN: zhongying's stuff ...

// only display lower segment of frequency spectrum (k = [0,199]):
wire fft_disp2;
assign fft_disp2 =
    ( fft_disp && (xk_index <= 12'd199)) ? fft_disp : 1'b0;

wire [6:0] add_num, interm, inters;
//convert the music and singer notes into unsigned
assign add_num = 7'b 1000000;
assign interm = add_num + tone[9:3];
assign inters = add_num + {voice[15], voice[11:6]};

wire video_pause;
assign video_pause = switch[2];

//memory files
wire reset, rewind, fforward, pause, start;

//internal wires
wire pause_p;
wire startmr;
wire [11:0] addr;
wire pause_d, enable;
wire [1:0] tempo_sel, song_sel;
wire [7:0] data;
wire [7:0] measure;
assign tempo_sel=switch [1:0];
assign song_sel=switch[7:6];
//note switch [7:6] is being used for voice-mic selection,
// however this is only effective when reset is pressed so
// they will not interfere with each other in play mode

//output LEDs
assign led[6:0]= ~addr[6:0];

compare_display display1( .clock(pixel_clock),
                          .reset_sync(reset_sync),
                          .pcount(pixel_count[10:0]),

```

```

        .lcount(line_count[10:0]),
        // note only lower four lsbs:
        .music_note(music_note[3:0]),
        .singer_note(singer_note[3:0])
        .copy_internal ((pixel_count==800)&&(line_count==599))
        //end of screen allow time to compute new RGB
        .new_m((en48k && video_pause)),
        .new_s((en48k && video_pause)),
        .new_f((fft_disp2 && video_pause)),
        .tone(interm),
        .voice(inters),
        .fft_out(fft_out),
        .measure (measure),
        .song_sel(song_sel),
        .vga_red(vga_out_red),
        .vga_green(vga_out_green),
        .vga_blue(vga_out_blue));

//memory files

//clock
//tempo enable
reg counttemp;
always@ (posedge pixel_clock)
    if (reset_sync) counttemp=0;
    else if (enable) counttemp=~counttemp;
assign led[7]= counttemp;

pulsify pi(pixel_clock, reset_sync, pause_sync, pause_p);
pulsify star(pixel_clock, reset_sync, start_sync, startmr);
//convert pause and start into pulses

mr_fsm yamapi(pixel_clock, reset_sync, rewind_sync,
              fforward_sync, startmr, pause_p,
              pause_d, enable, data, addr, music_note_zz,
              measure, song_sel);
divider news(reset_sync, pixel_clock, tempo_sel, pause_d, enable);

//memory
// read-only (ROM)
// width: 8 bits
// depth: 4096 memory locations
// contents: song1.coe
song_mem crazy(.addr(addr), .clk(pixel_clock), .dout(data));

// select control of synthesizer based on testmode
assign music_note = testmode_sync?
    {octv_sel[4:1], note_sel[4:1]} : data;

// ... END: zhongying's stuff
////////////////////////////////////

```

```

////////////////////////////////////
// PROBING (MQ)
// Logic Analyzer assignments (only 2 and 4 used)
assign analyzer1_data = 16'h0;
assign analyzer1_clock = 1'b1;
assign analyzer3_data = 16'h0;
assign analyzer3_clock = 1'b1;

assign analyzer2_data = {fft_disp,
                        music_note[6:4],
                        xk_index[11:0]};
assign analyzer2_clock = clk_ac97;

assign analyzer4_data = {kmax[8:0],
                        fft_out[6:0]};
assign analyzer4_clock = pixel_clock;

// repository of commonly probed signals:
// offset_accum[15:0]
// en48k
// bitcount[7:0]
// ac97_sdata_out
// ac97_sdata_in
// ac97_synch
// reset_sync_b
// audio_reset_b
// micv
// outv
// audio_in
//
// clock signals:
// ac97_bit_clock
// pixel_clock

endmodule

```

## A.2 audio.v

```

////////////////////////////////////
// audio.v ... module for audio interface
// by MQ
// citation: code adapted from chris's email and 6.111 fall 2005
//           example code ... however my code was written from scratch
//           and is substantially different (and in my opinion) more clear
////////////////////////////////////
module audio(resetb, clk_ac97,
            sdata_in, audio_out,

```

```

        outv_up_b, outv_dn_b, micv_up_b, micv_dn_b,
        sdata_out, sync, audio_in,
        bitcount,
        //probing outputs:
        micv, outv);

// MUST have these pseudo-comments to enable start-up of ac97 codec
//   LM4550, this is tricky because the ac97_bit_clock is used
//   as the clock for this module, but the clock doesn't start
//   until the LM4550 is successfully turned on!
//
// synthesis attribute init of sync is "0";
// synthesis attribute init of sdata_out is "0";

// note how audio_out is an *input* to the audio module and
// audio_in is an *output* from the audio module ...
// it is the voice from the mic. through the ac97 codec

input      resetb, clk_ac97, sdata_in;
input [15:0] audio_out;
input      outv_up_b, outv_dn_b, micv_up_b, micv_dn_b;
output [15:0] audio_in;
output      sdata_out, sync;
output [7:0] bitcount;

// probing outputs:
output [4:0] micv, outv;

// BIT COUNTER: bitcount cycles from 0 through 255
// slot 0 ("tag" slot) runs from bitcount == 0 through 15
// slot N runs from bitcount == 16 + (N-1)*20 through
//           bitcount == 15 + N * 20
// N is in {1, 2, ..., 11, 12}
reg [7:0] bitcount;
always @ (posedge clk_ac97) begin
    if (!resetb) bitcount <= 8'b0;
    else bitcount <= bitcount + 1;
end

// SYNC GENERATOR
reg sync;
// sequential block because sync musn't glitch
// sync will be high during bitcount == 255 through
//   bitcount = 14
// recall: slot 0, ("tag" slot) corresponds to bitcount == 0
//           through bitcount == 15
always @ (posedge clk_ac97) begin
    if (!resetb) sync <= 1'b0;
    else if (bitcount == 254) sync <= 1'b1;
    else if (bitcount == 14) sync <= 1'b0;
end

```

```

// SERIAL TO PARALLEL: for audio_in
// audio_in will correspond to data in slot 3
// I do not check validity, I am willing to spit out garbage
reg [15:0] audio_in;
reg [19:0] audio_in_int;
always @ (posedge clk_ac97) begin
    if (!resetb) begin
        audio_in_int <= 20'b0;
        audio_in <= 16'b0;
    end else begin
        // if we are in slot 3 ...
        if ((bitcount >= 56) && (bitcount <= 75)) begin
            audio_in_int <= {audio_in_int[18:0], sdata_in};
        end
        // present new data ASAP ... (may change)
        if (bitcount == 76) audio_in <= audio_in_int[19:4];
    end
end

// MUX: for sdata_out:
// I believe sdata_out can be glitchy. pg. 18 of LM4550
// datasheet says how that the sdata_out for bit period
// n should be clocked on the beginning rising clock edge
// from the controller and then the LM4550 samples this
// signal on the subsequent *falling* edge.
reg sdata_out, sdata_out_a, sdata_out_c;
always @ (sdata_out or sdata_out_a or sdata_out_c
          or bitcount) begin
    if ((bitcount >= 8'd0) && (bitcount <= 8'd55)) begin
        sdata_out = sdata_out_c;
    end else if ((bitcount >= 8'd56) && (bitcount <= 8'd95)) begin
        sdata_out = sdata_out_a;
    end else sdata_out = 1'b0;
end

// PARALLEL TO SERIAL: for audio_out
// both L and R channels get same data, note how this is
// accomplished by cyclically shifting audio_out_int
// since the mux chooses sdata_out_a when appropriate I can
// leave audio_out_int continuously cyclically shifting
reg [19:0] audio_out_int;
always @ (posedge clk_ac97) begin
    if (bitcount == 54) audio_out_int <= {audio_out, 4'h0};
    else begin
        sdata_out_a <= audio_out_int[19];
        audio_out_int <= {audio_out_int[18:0], audio_out_int[19]};
    end
end

// COMMAND LOOP: for tag, slot1, slot2, mic & vol

```

```

// slot 0:
parameter slot0 = 16'b1111100000000000;

// slot 1 ("tag"):
// variables for first 8 bits
parameter r_or_w = 1'b0;
// so far I only intend to write the command registers
// If I chagne my mind I shall rever to:
// reg r_or_w; // 1 = read, 0 = write
reg [6:0] addr; // identifies status/command register for
                // read or write
// the last 12 bits should be strictly zero
wire [19:0] slot1;
assign slot1 = {r_or_w, addr, 12'b0};

// slot 2:
reg [15:0] wdata; // write data to register, should be
                  // all zeros if reading

reg [55:0] slots_012_cycle;
always @ (posedge clk_ac97) begin
    if (bitcount == 254) begin
        slots_012_cycle <= {slot0, slot1, wdata, 4'b0000};
    end else begin
        sdata_out_c <= slots_012_cycle[55];
        slots_012_cycle <= {slots_012_cycle[54:0], slots_012_cycle[55]};
    end
end
end

// microphone volume (micv) and headphone volume (outv)
// outv is an attenuation in dB: 5'b0_0000 -> 0dB attenuation
//                               5'b1_1111 -> 46.5dB atten.
// micv is a gain in dB: 5'b1_1111 -> 22.5dB gain
//                               5'b0_0000 -> 0dB gain

wire [4:0] micv, outv;
bcontrol bcontrol_micv(resetb, clk_ac97,
                      micv_up_b, micv_dn_b, micv);
defparam bcontrol_micv.init_val = 5'b10000;
// the init_val gives the best audio default

bcontrol bcontrol_outv(resetb, clk_ac97,
                      outv_dn_b, outv_up_b, outv);
defparam bcontrol_outv.init_val = 5'b00100;
// note how outv_dn_b and outv_up_b are interchanged
// from micv_dn_b and micv_up_b ... namely a decrease
// in the binary value of outv[4:0] is a
// decrease in attenuation --> increase in vol.

// micv and outv volume controls ... will later implement

```

```

// push button incrementation in a bcontrol module

// at this point, all that is left is to implement the commands
// FSM that goes through command states and sets:
// addr[7:0], wdata[15:0]

reg [2:0] state; // need to cycle through seven commands

always @ (posedge clk_ac97) begin
    if (!resetb) state <= 3'b000;
    else if (bitcount == 253) state <= state + 1;
end

// C/L look up table
always @ (state or addr or wdata or micv or outv) begin
    case (state)
        3'd0:
            begin // PCM volume out
                addr = 7'b001_1000;
                wdata = 16'b0000000000000000; // 12dB gain (max)
            end
        3'd1:
            begin // Master volume (for line out)
                addr = 7'b000_0010;
                wdata = {3'b000, outv, 3'b000, outv};
            end
        3'd2:
            begin // Headphone volume
                addr = 7'b000_0100;
                wdata = {3'b000, outv, 3'b000, outv};
            end
        3'd3:
            begin // PC_Beep volume
                addr = 7'b000_1010;
                wdata = 16'h8000;
            end
        3'd4:
            begin // Mic volume for ANALOG loop back (should be MUTED)
                addr = 7'b000_1110;
                wdata = {9'b100000000, micv[4], 6'b011111};
                // note how MSB of micv selects 20dB boost
            end
        3'd5:
            begin // Line in volume .. not used but maybe later
                addr = 7'b001_0000;
                wdata = 16'h8808;
            end
        3'd6:
            begin // Record select
                addr = 7'b001_1010;
                wdata = 16'h0000;
            end
    endcase
end

```

```

    end
    3'd7:
    begin // Record gain .. This controls Mic.'s digital vol.
        addr = 7'b001_1100;
        wdata = {4'h0, micv[3:0], 4'h0, micv[3:0]};
    end
    endcase // case(state)
end // always @ (state or addr or wdata or micv or outv)

endmodule // audio

```

### A.3 fft\_subsystem.v

```

//////////////////////////////////////////////////////////////////
// fft_subsystem.v ... module for running fft
// by MQ
//////////////////////////////////////////////////////////////////
module fft_subsystem (resetb, clk, en48k, fft_data,
                    fft_disp, fft_out, singer_note,
                    offset_accum,
                    // probing outputs
                    xk_index, kmax);

    input resetb, clk, en48k;
    input signed [15:0] fft_data;
    output          fft_disp;
    output [6:0]    fft_out;
    output [7:0]    singer_note;
    output [15:0]   offset_accum;

    // probing outputs:
    output [11:0]  xk_index;
    output [11:0]  kmax;

    // current code is for N = 4096 = 2^12

    // sequential block for state of fft controller
    // state is summarized by {load, count}
    reg          load, next_load;
    reg [11:0]    count, next_count;
    always @ (posedge clk) begin
        if (!resetb) begin
            load <= 1'b0;
            count <= 12'b0;
        end else begin
            load <= next_load;
            count <= next_count;
        end
    end
end

// combinational logic for next state of fft
// controller

```

```

always @ (*) begin
    next_count = count;
    next_load = load;
    if (!load) begin // recording data to memory
        if (en48k) begin
            next_count = count + 1;
            if (count == 12'd4095) next_load = 1'b1;
        end
    end else begin // loading data to fft
        next_count = count + 1;
        if (count == 12'd4095) next_load = 1'b0;
    end
end

end

// memory for storing voice data:
// wires for fft_data memory
wire signed [15:0] dout;
reg signed [15:0] din;
reg [11:0] addr;
reg we;

fftvoicebram fftvoicebram0(
    .addr(addr),
    .clk(clk),
    .din(din),
    .dout(dout),
    .we(we));
// ... look at instantiation template in proj. nav.

// instantiation of fft module:
// wires for fft module
reg signed [15:0] xn_re;
wire [11:0] xn_index;
reg start;
wire edone, unload, done, busy, rfd, dv;
wire [4:0] blk_exp;
wire [11:0] xk_index;
wire signed [15:0] xk_re, xk_im;

myfft myfft0 (
    .xn_re(xn_re),
    .xn_im(16'b0),
    .start(start),
    .unload(unload),
    .nfft(5'd12), // log2(N) = 12 for N = 4096
    .nfft_we(1'b0), // may have to make a variable to initialize
    .fwd_inv(1'b1),
    .fwd_inv_we(1'b0), // same as nfft_we
    .clk(clk),
    .xk_re(xk_re),
    .xk_im(xk_im),

```

```

        .xn_index(xn_index),
        .xk_index(xk_index),
        .rfd(rfd),
        .busy(busy),
        .dv(dv),
        .edone(edone),
        .done(done),
        .blk_exp(blk_exp));

// C/L for memory module and fft module control
//   based on the state of the controller and inputs (mealy)
//   if load = 0, we simply are writing voice data into
//   memory whenever en48k is pulsed (i.e. new sample arrives)
//   count is the memory address
//   if load = 1, we are reading voice data out EVERY CLOCK cycle
//   count is the memory address

// declare two new variables for auto-zeroing of A/D data
//   their values will be determined with note detection fsm
//   presented later in this module
reg signed [15:0] offset_accum, offset_delta;

always @ (*) begin
    we = 1'b0;
    addr = count;
    din = fft_data;
    start = 1'b0;
    // write to memory:
    if (!load) we = en48k;
    else begin // start fft:
        if (count == 12'b0) start = 1'b1;
    end
end

// sequential block for creating necessary delays to fft module
// all delays are referenced to number of clock cycles after
// when count == 12'b0 on the posedge clk
// need to delay memory dout by three additional clock cycles
// ... it is already delayed by one clock cycle by the memory
// ... reading operation !

reg signed [15:0] dout_1, dout_2;

always @ (posedge clk) begin
    dout_1 <= dout;
    dout_2 <= dout_1;
    xn_re <= dout_2;
end

// combinational logic for unloading data from fft

```

```

assign unload = edone;

// C/L for square and sum
// note explicit signed multiplication
wire [31:0] sk;
assign sk =    ({{16{xk_re[15]}},xk_re} * {{16{xk_re[15]}},xk_re})
              +({{16{xk_im[15]}},xk_im} * {{16{xk_im[15]}},xk_im});
// xk_re, and xk_im are signed 2's complement
// their squares will be signed 2's complement guaranteed to be pos.
// the sum sk will be signed 2's complement guaranteed to be pos.
//     UNLESS there is overflow. BUT overflow will not affect us
//     because sk is being interpreted as unsigned mangitude

// note detection and DC extraction FSM
// sequential block: state is summarized by {idle, skmax, kmax}
reg    idle, next_idle;
reg [31:0] skmax, next_skmax, skmax_hold;
reg [31:0] skmax2, next_skmax2;
reg [11:0] kmax, next_kmax, kmax_hold;
reg [11:0] kmax2, next_kmax2, kmax2_hold;
reg [15:0] xk_re_0; // record DC component measured by fft for
                  // auto-zeroing

always @ (posedge clk) begin
  if (!resetb) begin
    idle <= 1;
    skmax <= 32'd0;
    skmax2 <= 32'd0;
    kmax <= 12'd0;
    kmax2 <= 12'd0;
    skmax_hold <= 32'd0;
    kmax_hold <= 12'd0;
    kmax2_hold <= 12'd0;
    xk_re_0 <= 16'd0; // dc offset var
    offset_accum <= 16'd0;
  end else begin
    idle <= next_idle;
    skmax <= next_skmax;
    skmax2 <= next_skmax2;
    kmax <= next_kmax;
    kmax2 <= next_kmax2;
    if (!idle) begin
      if (xk_index == 12'd0) xk_re_0 <= xk_re;
      if ( next_idle ) begin // end of FFT output burst
        offset_accum <= offset_accum + offset_delta;
        skmax_hold <= next_skmax;
        kmax_hold <= next_kmax;
        kmax2_hold <= next_kmax2;
      end
    end
  end
end // else: !if(!resetb)

```

```

end // always @ (posedge clk)

// C/L for state update:

// For octaves 1 and 2, notes must be inferred from
// 2nd or 4th or 8th harmonics
wire valid_harmonic;
harmonic_check harmonic_check0 (.kfund(kmax),
                                .kharm(xk_index),
                                .valid(valid_harmonic) );

always @ (*) begin
    next_skmax = skmax;
    next_skmax2 = skmax2;
    next_kmax = kmax;
    next_kmax2 = kmax2;
    next_idle = idle;
    if (idle) begin // wait for fft "done" signal
        if (done) next_idle = 1'b0;
    end else begin
        if (!dv) begin // reset skmax, kmax before new data
            next_skmax = 32'd0;
            next_skmax2 = 32'd0;
            next_kmax = 12'b0;
            next_kmax2 = 12'b0;
        end else begin // find new maximum point in spectrum
            if (sk > skmax) begin
                next_skmax = sk;
                next_kmax = xk_index;
                // also check for higher harmonics (2, 4, 8):
                // the location of these harmonics adds value ONLY
                // if it is above octave 2. Hence:
            end else if (xk_index >= 12'd19) begin
                if (valid_harmonic) begin // exclude 3, 5, 6, 7
                    if (sk > skmax2) begin
                        next_skmax2 = sk;
                        next_kmax2 = xk_index;
                    end
                end
            end
            // skip over the part of the spectrum BELOW octave 1:
            if (xk_index <= 12'd4) begin
                next_skmax = 32'd0;
                next_skmax2 = 32'd0;
            end
            // in principle the above block is unnecessary because
            // auto-zeroing should eliminate the chance that
            // the maximum sk is at k=0 or nearby
        end
        if (xk_index == 12'd2048) next_idle = 1'b1;
        // we end when we are HALF way through (4096/2) because spectrum

```

```

    // axis is symmetric about midpoint k value
    end // else: !if(idle)
end // always @ (*)

// C/L for A/D digital DC offset correction
// need to scale xk_re(k=0) by 2-(12 - blk_exp)
wire signed [4:0] dc_exp;
assign dc_exp = 5'd12 - blk_exp;
// I will crudely implement this scaling by clipping off LSBs
always @ (*) begin
    case (dc_exp)
        5'd0: offset_delta = xk_re_0;
        5'd1: offset_delta = {xk_re_0[15], xk_re_0[15:1]};
        5'd2: offset_delta = {{2{xk_re_0[15]}}, xk_re_0[15:2]};
        5'd3: offset_delta = {{3{xk_re_0[15]}}, xk_re_0[15:3]};
        5'd4: offset_delta = {{4{xk_re_0[15]}}, xk_re_0[15:4]};
        5'd5: offset_delta = {{5{xk_re_0[15]}}, xk_re_0[15:5]};
        5'd6: offset_delta = {{6{xk_re_0[15]}}, xk_re_0[15:6]};
        5'd7: offset_delta = {{7{xk_re_0[15]}}, xk_re_0[15:7]};
        5'd8: offset_delta = {{8{xk_re_0[15]}}, xk_re_0[15:8]};
        5'd9: offset_delta = {{9{xk_re_0[15]}}, xk_re_0[15:9]};
        5'd10: offset_delta = {{10{xk_re_0[15]}}, xk_re_0[15:10]};
        5'd11: offset_delta = {{11{xk_re_0[15]}}, xk_re_0[15:11]};
        5'd12: offset_delta = {{12{xk_re_0[15]}}, xk_re_0[15:12]};
        default: offset_delta = xk_re_0;
    endcase // case(dc_exp)
end // always @ (*)

// C/L for display (spectrum data, and control)
// enable ZZ's display module w/ fft_disp:
reg fft_disp;
always @ (*) begin
    if ( (!idle) && dv) fft_disp = 1'b1;
    else fft_disp = 1'b0;
end

// provide 7-bit unsigned magnitude to ZZ's display module:
reg [6:0] fft_out;
always @ (*) begin
    if (skmax_hold[31]) fft_out = sk[31:25];
    else if (skmax_hold[30]) fft_out = sk[30:24];
    else if (skmax_hold[29]) fft_out = sk[29:23];
    else if (skmax_hold[28]) fft_out = sk[28:22];
    else if (skmax_hold[27]) fft_out = sk[27:21];
    else if (skmax_hold[26]) fft_out = sk[26:20];
    else if (skmax_hold[25]) fft_out = sk[25:19];
    else if (skmax_hold[24]) fft_out = sk[24:18];
    else if (skmax_hold[23]) fft_out = sk[23:17];
    else if (skmax_hold[22]) fft_out = sk[22:16];
    else if (skmax_hold[21]) fft_out = sk[21:15];
    else if (skmax_hold[20]) fft_out = sk[20:14];
end

```

```

    else if (skmax_hold[19]) fft_out = sk[19:13];
    else if (skmax_hold[18]) fft_out = sk[18:12];
    else if (skmax_hold[17]) fft_out = sk[17:11];
    else if (skmax_hold[16]) fft_out = sk[16:10];
    else if (skmax_hold[15]) fft_out = sk[15:9 ];
    else if (skmax_hold[14]) fft_out = sk[14:8 ];
    else if (skmax_hold[13]) fft_out = sk[13:7 ];
    else if (skmax_hold[12]) fft_out = sk[12:6 ];
    else if (skmax_hold[11]) fft_out = sk[11:5 ];
    else if (skmax_hold[10]) fft_out = sk[10:4 ];
    else if (skmax_hold[9])  fft_out = sk[9 :3  ];
    else if (skmax_hold[8])  fft_out = sk[8 :2  ];
    else if (skmax_hold[7])  fft_out = sk[7 :1  ];
    else if (skmax_hold[6])  fft_out = sk[6 :0  ];
    else
        fft_out = sk[5 :0  ];
end // always @ (*)

// C/L for note determination:
wire [7:0] singer_note; // top four bits -> octave
                        // bot four bits -> note
wire [7:0] note1, note2;
k_to_note k_to_note1( .k(kmax_hold), .note(note1));
k_to_note k_to_note2( .k(kmax2_hold), .note(note2));

// if note1 is invalid (i.e. kmax_hold is below octave 3)
// try to detect note from kmax2_hold
assign singer_note = (note1[3])?
    {note1[7:4], note2[3:0]} : note1;
// * "note1" still contains correct octave info.
// if you look at the code for k_to_note.v, you will see
// that an invalid note is assigned to 4'b1111 in the LSBs
// Thus, I check validity from simply looking at the [3] bit

endmodule // fft_subsystem

```

#### A.4 synth.v

```

////////////////////////////////////
// synth.v module for synthesizing tones
// by MQ
////////////////////////////////////

module synth(resetb, clk, note, octave, tone);

    input resetb, clk;
    input [3:0] note, octave;
    output [15:0] tone;

    // this amplitude gives a good volume
    parameter amplitude = 16'h00FF;

    // frequency mappings

```

```

reg [18:0] pby2;
always @ (*) begin
  case (octave)
    4'd1:
      case (note) // octave 1
        4'd0: pby2 = 19'd335127;
        4'd1: pby2 = 19'd316318;
        4'd2: pby2 = 19'd298564;
        4'd3: pby2 = 19'd281807;
        4'd4: pby2 = 19'd265991;
        4'd5: pby2 = 19'd251062;
        4'd6: pby2 = 19'd236971;
        4'd7: pby2 = 19'd223671;
        4'd8: pby2 = 19'd211117;
        4'd9: pby2 = 19'd199268;
        4'd10: pby2 = 19'd188084;
        4'd11: pby2 = 19'd177527;
        default:
          pby2 = 19'd1023; // higher than G# 8
      endcase // case(note)
    4'd2:
      case (note) // octave 2
        4'd0: pby2 = 19'd167564;
        4'd1: pby2 = 19'd158159;
        4'd2: pby2 = 19'd149282;
        4'd3: pby2 = 19'd140904;
        4'd4: pby2 = 19'd132995;
        4'd5: pby2 = 19'd125531;
        4'd6: pby2 = 19'd118485;
        4'd7: pby2 = 19'd111835;
        4'd8: pby2 = 19'd105558;
        4'd9: pby2 = 19'd99634;
        4'd10: pby2 = 19'd94042;
        4'd11: pby2 = 19'd88764;
        default:
          pby2 = 19'd1023; // higher than G# 8
      endcase // case(note)
    4'd3:
      case (note) // octave 3
        4'd0: pby2 = 19'd83782;
        4'd1 : pby2 = 19'd79080;
        4'd2 : pby2 = 19'd74641;
        4'd3 : pby2 = 19'd70452;
        4'd4 : pby2 = 19'd66498;
        4'd5 : pby2 = 19'd62765;
        4'd6 : pby2 = 19'd59243;
        4'd7 : pby2 = 19'd55918;
        4'd8 : pby2 = 19'd52779;
        4'd9 : pby2 = 19'd49817;
        4'd10: pby2 = 19'd47021;
        4'd11: pby2 = 19'd44382;

```

```

    default:
      pby2 = 19'd1023; // higher than G# 8
    endcase // case(note)
4'd4:
  case (note) // octave 4
    4'd0: pby2 = 19'd41891;
    4'd1: pby2 = 19'd39540;
    4'd2: pby2 = 19'd37321;
    4'd3: pby2 = 19'd35226;
    4'd4: pby2 = 19'd33249;
    4'd5: pby2 = 19'd31383;
    4'd6: pby2 = 19'd29621;
    4'd7: pby2 = 19'd27959;
    4'd8: pby2 = 19'd26390;
    4'd9: pby2 = 19'd24908;
    4'd10: pby2 = 19'd23510;
    4'd11: pby2 = 19'd22191;
  default:
    pby2 = 19'd1023; // higher than G# 8
  endcase // case(note)
4'd5:
  case (note) // octave 5
    4'd0: pby2 = 19'd20945;
    4'd1: pby2 = 19'd19770;
    4'd2: pby2 = 19'd18660;
    4'd3: pby2 = 19'd17613;
    4'd4: pby2 = 19'd16624;
    4'd5: pby2 = 19'd15691;
    4'd6: pby2 = 19'd14811;
    4'd7: pby2 = 19'd13979;
    4'd8: pby2 = 19'd13195;
    4'd9: pby2 = 19'd12454;
    4'd10: pby2 = 19'd11755;
    4'd11: pby2 = 19'd11095;
  default:
    pby2 = 19'd1023; // higher than G# 8
  endcase // case(note)
4'd6:
  case (note) // octave 6
    4'd0: pby2 = 19'd10473;
    4'd1 : pby2 = 19'd9885;
    4'd2 : pby2 = 19'd9330;
    4'd3 : pby2 = 19'd8806;
    4'd4 : pby2 = 19'd8312;
    4'd5 : pby2 = 19'd7846;
    4'd6 : pby2 = 19'd7405;
    4'd7 : pby2 = 19'd6990;
    4'd8 : pby2 = 19'd6597;
    4'd9 : pby2 = 19'd6227;
    4'd10: pby2 = 19'd5878;
    4'd11: pby2 = 19'd5548;

```

```

        default:
            pby2 = 19'd1023; // higher than G# 8
        endcase // case(note)
4'd7:
    case (note) // octave 7
        4'd0: pby2 = 19'd5236;
        4'd1: pby2 = 19'd4942;
        4'd2: pby2 = 19'd4665;
        4'd3: pby2 = 19'd4403;
        4'd4: pby2 = 19'd4156;
        4'd5: pby2 = 19'd3923;
        4'd6: pby2 = 19'd3703;
        4'd7: pby2 = 19'd3495;
        4'd8: pby2 = 19'd3299;
        4'd9: pby2 = 19'd3114;
        4'd10: pby2 = 19'd2939;
        4'd11: pby2 = 19'd2774;
    default:
        pby2 = 19'd1023; // higher than G# 8
    endcase // case(note)
4'd8:
    case (note) // octave 8
        4'd0: pby2 = 19'd2618;
        4'd1: pby2 = 19'd2471;
        4'd2: pby2 = 19'd2333;
        4'd3: pby2 = 19'd2202;
        4'd4: pby2 = 19'd2078;
        4'd5: pby2 = 19'd1961;
        4'd6: pby2 = 19'd1851;
        4'd7: pby2 = 19'd1747;
        4'd8: pby2 = 19'd1649;
        4'd9: pby2 = 19'd1557;
        4'd10: pby2 = 19'd1469;
        4'd11: pby2 = 19'd1387;
    default:
        pby2 = 19'd1023; // higher than G# 8
    endcase // case(note)
    default: pby2 = 19'd1023; // higher than G# 8
    endcase // case(octave)
end // always @ (*)

// square wave FSM:
reg        phase, next_phase;
reg [18:0] pby2count, next_pby2count;

// sequential state registers
always @ (posedge clk) begin
    if (!resetb) begin
        phase <= 1'b0;
        pby2count <= 19'd1;
    end else begin

```

```

        phase <= next_phase;
        pby2count <= next_pby2count;
    end
end

// C/L for next state
always @ (*) begin
    if (pby2count == pby2) begin
        next_phase = ~phase; // toggle square wave
        next_pby2count = 19'd1; // reset counter
    end else begin
        next_phase = phase;
        next_pby2count = pby2count + 1;
    end
end

reg [15:0] tone;
always @ (*) begin
    // all notes outside of 4'd0 -- 4'd11 are interpreted
    // as "rest" notes:
    if ( pby2 == 19'd1023 ) tone = 16'd0;
    else begin
        if (phase) tone = amplitude;
        else tone = (~amplitude) + 1;
    //     else tone = 16'd0; // for verifying DC extraction
    end
end

endmodule // synth

```

## A.5 k\_to\_note.v

```

////////////////////////////////////
// k_to_note.v look-up table mapping k (location in frequency)
//                               to note (name of note)
// by MQ
////////////////////////////////////
module k_to_note ( k, note);

    input  [11:0] k;
    output [7:0] note;

    parameter noteA = 4'd0;
    parameter noteAsharp = 4'd1;
    parameter noteB = 4'd2;
    parameter noteC = 4'd3;
    parameter noteCsharp = 4'd4;
    parameter noteD = 4'd5;
    parameter noteDsharp = 4'd6;
    parameter noteE = 4'd7;
    parameter noteF = 4'd8;
    parameter noteFsharp = 4'd9;

```

```

parameter noteG = 4'd10;
parameter noteGsharp = 4'd11;

reg [7:0] note;
always @ (k or note) begin
    note = 8'b11111111;
    if ( (k >= 12'd5) && (k <= 12'd9) ) note = {4'd1, 4'b1111};
    else if ( (k >= 12'd10) && (k <= 12'd18) ) note = {4'd2, 4'b1111};
    else if ( (k >= 12'd19) && (k <= 12'd36) ) begin
        // octave 3
        note [7:4] = 4'd3;
        if (k <= 12'd19) note[3:0] = noteA;
        else if (k <= 12'd20) note[3:0] = noteAsharp;
        else if (k <= 12'd21) note[3:0] = noteB;
        else if (k <= 12'd22) note[3:0] = noteC;
        else if (k <= 12'd24) note[3:0] = noteCsharp;
        else if (k <= 12'd25) note[3:0] = noteD;
        else if (k <= 12'd27) note[3:0] = noteDsharp;
        else if (k <= 12'd28) note[3:0] = noteE;
        else if (k <= 12'd30) note[3:0] = noteF;
        else if (k <= 12'd32) note[3:0] = noteFsharp;
        else if (k <= 12'd34) note[3:0] = noteG;
        else
            note[3:0] = noteGsharp;
    end
    else if ( (k >= 12'd37) && (k <= 12'd72) ) begin
//    if ( (k >= 12'd37) && (k <= 12'd72) ) begin
        // octave 4
        note [7:4] = 4'd4;
        if (k <= 12'd38) note[3:0] = noteA;
        else if (k <= 12'd40) note[3:0] = noteAsharp;
        else if (k <= 12'd43) note[3:0] = noteB;
        else if (k <= 12'd45) note[3:0] = noteC;
        else if (k <= 12'd48) note[3:0] = noteCsharp;
        else if (k <= 12'd51) note[3:0] = noteD;
        else if (k <= 12'd54) note[3:0] = noteDsharp;
        else if (k <= 12'd57) note[3:0] = noteE;
        else if (k <= 12'd61) note[3:0] = noteF;
        else if (k <= 12'd64) note[3:0] = noteFsharp;
        else if (k <= 12'd68) note[3:0] = noteG;
        else
            note[3:0] = noteGsharp;
    end
    else if ( (k >= 12'd73) && (k <= 12'd145) ) begin
        // octave 5
        note [7:4] = 4'd5;
        if (k <= 12'd77) note[3:0] = noteA;
        else if (k <= 12'd81) note[3:0] = noteAsharp;
        else if (k <= 12'd86) note[3:0] = noteB;
        else if (k <= 12'd91) note[3:0] = noteC;
        else if (k <= 12'd97) note[3:0] = noteCsharp;
        else if (k <= 12'd103) note[3:0] = noteD;
        else if (k <= 12'd109) note[3:0] = noteDsharp;
    end
end

```

```

    else if (k <= 12'd115) note[3:0] = noteE;
    else if (k <= 12'd122) note[3:0] = noteF;
    else if (k <= 12'd129) note[3:0] = noteFsharp;
    else if (k <= 12'd137) note[3:0] = noteG;
    else
        note[3:0] = noteGsharp;
end
else if ( (k >= 12'd146) && (k <= 12'd291) ) begin
    // octave 6
    note [7:4] = 4'd6;
    if (k <= 12'd154) note[3:0] = noteA;
    else if (k <= 12'd163) note[3:0] = noteAsharp;
    else if (k <= 12'd173) note[3:0] = noteB;
    else if (k <= 12'd183) note[3:0] = noteC;
    else if (k <= 12'd194) note[3:0] = noteCsharp;
    else if (k <= 12'd206) note[3:0] = noteD;
    else if (k <= 12'd218) note[3:0] = noteDsharp;
    else if (k <= 12'd231) note[3:0] = noteE;
    else if (k <= 12'd245) note[3:0] = noteF;
    else if (k <= 12'd259) note[3:0] = noteFsharp;
    else if (k <= 12'd275) note[3:0] = noteG;
    else
        note[3:0] = noteGsharp;
end
else if ( (k >= 12'd292) && (k <= 12'd583) ) begin
    // octave 7
    note [7:4] = 4'd7;
    if (k <= 12'd309) note[3:0] = noteA;
    else if (k <= 12'd327) note[3:0] = noteAsharp;
    else if (k <= 12'd347) note[3:0] = noteB;
    else if (k <= 12'd367) note[3:0] = noteC;
    else if (k <= 12'd389) note[3:0] = noteCsharp;
    else if (k <= 12'd412) note[3:0] = noteD;
    else if (k <= 12'd437) note[3:0] = noteDsharp;
    else if (k <= 12'd463) note[3:0] = noteE;
    else if (k <= 12'd490) note[3:0] = noteF;
    else if (k <= 12'd519) note[3:0] = noteFsharp;
    else if (k <= 12'd550) note[3:0] = noteG;
    else
        note[3:0] = noteGsharp;
end
else if ( (k >= 12'd584) && (k <= 12'd1167) ) begin
    // octave 8
    note [7:4] = 4'd8;
    if (k <= 12'd618) note[3:0] = noteA;
    else if (k <= 12'd655) note[3:0] = noteAsharp;
    else if (k <= 12'd694) note[3:0] = noteB;
    else if (k <= 12'd735) note[3:0] = noteC;
    else if (k <= 12'd779) note[3:0] = noteCsharp;
    else if (k <= 12'd825) note[3:0] = noteD;
    else if (k <= 12'd874) note[3:0] = noteDsharp;
    else if (k <= 12'd926) note[3:0] = noteE;
    else if (k <= 12'd984) note[3:0] = noteF;
    else if (k <= 12'd1039) note[3:0] = noteFsharp;

```

```

        else if (k <= 12'd1101) note[3:0] = noteG;
        else
            note[3:0] = noteGsharp;
        end
    end // always @ (k or note)
endmodule // k_to_note

```

## A.6 harmonic\_check.v

```

/////////////////////////////////////////////////////////////////
// harmonic_check.v
// by MQ
/////////////////////////////////////////////////////////////////
module harmonic_check( kfund, kharm, valid );

    input  [11:0] kfund, kharm;
    output          valid;

    reg  [11:0]  kby2, kx2, kx4, kx8;
    reg          valid;

    always @ (kfund or kharm or valid) begin
        valid = 1'b0;
        kby2 = {1'b0, kfund[11:1]} + 1; // coarse rounding up
        kx2 = {kfund[10:0], 1'b0};
        kx4 = {kfund[9:0], 2'b0};
        kx8 = {kfund[8:0], 3'b0};
        if (kharm <= (kx2 + kby2))
            valid = 1'b1;
        else if ( (kharm >= kx4 - kby2) && (kharm <= kx4 + kby2))
            valid = 1'b1;
        else if ( (kharm >= kx8 - kby2) && (kharm <= kx8 + kby2) )
            valid = 1'b1;
        // The goal of the above sequence of conditions was to
        // exclude 3rd, 5th, 6th, 7th harmonics, which are trickier
        // to map to a note name based on the fundamental
    end // always @ (kfund or kharm or valid)
endmodule // harmonic_check

```

## A.7 dots\_num\_2x.v

```

/////////////////////////////////////////////////////////////////
// dots_num_2x.v
// by MQ
//
// citation: adapted from 6.111 alphanumeric code
//
//
// This module converts a 4-bit input to a 80-dot (2 digit)
// bitmap representing the numbers ' 1' through ' 8'.
//

```

```

//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
module dots_num_2x(clk, num, dots);
  input clk;
  input [3:0] num;
  output [79:0] dots;

  parameter noteX = 40'b01100011_00110110_00001000_00110110_01100011;

  reg [79:0] dots;
  always @ (posedge clk)
    case (num)
      4'd15: dots <= {noteX,
                    noteX};
      4'd14: dots <= {noteX,
                    noteX};
      4'd13: dots <= {noteX,
                    noteX};
      4'd12: dots <= {noteX,
                    noteX};
      4'd11: dots <= {noteX,
                    noteX};
      4'd10: dots <= {noteX,
                    noteX};
      4'd09: dots <= {noteX,
                    noteX};
      4'd08: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 8'
                    40'b00110110_01001001_01001001_01001001_00110110};
      4'd07: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 7'
                    40'b00000001_01110001_00001001_00000101_00000011};
      4'd06: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 6'
                    40'b00111100_01001010_01001001_01001001_00110000};
      4'd05: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 5'
                    40'b00100111_01000101_01000101_01000101_00111001};
      4'd04: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 4'
                    40'b00011000_00010100_00010010_01111111_00010000};
      4'd03: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 3'
                    40'b00100010_01000001_01001001_01001001_00110110};
      4'd02: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 2'
                    40'b01100010_01010001_01001001_01001001_01000110};
      4'd01: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 1'
                    40'b00000000_01000010_01111111_01000000_00000000};
      4'd00: dots <= {noteX,
                    noteX};
      // No default case, because every case is already accounted for.
    endcase

endmodule
// A string of 40 bits is {left col, ..., right col}, with the
// LSB of each 8-bit segment correspondint to the TOP.

```

```
// Also only 7 bits are displayed, the MSB should be zero always.
// Basically, you feed in vertical collumns running from bottom (MSB)
// to top (LSB), then left to right, with the MSB getting clipped
//
```

## A.8 dots\_note\_2x.v

```
////////////////////////////////////
// dots_note_2x.v module to display note (with sharp symbol)
// by MQ
// citation: adapted from 6.111 alphanumeric code, however I hand-wrote
// the bit matrix for the note letters
////////////////////////////////////

module dots_note_2x(clk, note, dots);
    input clk;
    input [3:0] note;
    output [79:0] dots;

    parameter noteA = 40'b01111110_00001001_00001001_00001001_01111110;
    parameter noteB = 40'b01111111_01001001_01001001_01001001_00110110;
    parameter noteC = 40'b00111110_01000001_01000001_01000001_00100010;
    parameter noteD = 40'b01111111_01000001_01000001_01000001_00111110;
    parameter noteE = 40'b01111111_01001001_01001001_01001001_01001001;
    parameter noteF = 40'b01111111_00001001_00001001_00001001_00001001;
    parameter noteG = 40'b00111110_01000001_01001001_01001001_00111010;
    parameter sharp = 40'b00010100_00111110_00010100_00111110_00010100;
    parameter noteX = 40'b01100011_00110110_00001000_00110110_01100011;

    reg [79:0] dots;
    always @ (posedge clk)
        case (note)
            // X X (unspecified)
            4'd15: dots <= {noteX,
                           noteX};
            4'd14: dots <= {noteX,
                           noteX};
            4'd13: dots <= {noteX,
                           noteX};
            4'd12: dots <= {noteX,
                           noteX};

            // G #
            4'd11: dots <= {noteG,
                           sharp};

            // G
            4'd10: dots <= {noteG,
                           40'b0};

            // F #
            4'd09: dots <= {noteF,
                           sharp};

            // F
            4'd08: dots <= {noteF,
```

```

        40'b0};
    // E
    4'd07: dots <= {noteE,
        40'b0};
    // D #
    4'd06: dots <= {noteD,
        sharp};
    // D
    4'd05: dots <= {noteD,
        40'b0};
    // C #
    4'd04: dots <= {noteC,
        sharp};
    // C
    4'd03: dots <= {noteC,
        40'b0};
    // B
    4'd02: dots <= {noteB,
        40'b0};
    // A#
    4'd01: dots <= {noteA,
        sharp};
    // A
    4'd00: dots <= {noteA,
        40'b0};
    // No default case, because every case is already accounted for.
endcase

endmodule
// A string of 40 bits is {left col, ..., right col}, with the
// LSB of each 8-bit segment correspondint to the TOP.
// Also only 7 bits are displayed, the MSB should be zero always.
// Basically, you feed in vertical collumns running from bottom (MSB)
// to top (LSB), then left to right, with the MSB getting clipped
//
// "A"
//
// parameter noteA = 40'b01111110_00001001_00001001_00001001_01111110;
//
//
// 01110
// 10001
// 10001
// 11111
// 10001
// 10001
// 10001
// 10001
// 10001
// 00000
//
// "B"
//

```

```
// parameter noteB = 40'b01111111_01001001_01001001_01001001_00110110;
//
//
// 11110
// 10001
// 10001
// 11110
// 10001
// 10001
// 11110
// 00000
//
// "C"
//
// parameter noteC = 40'b00111110_01000001_01000001_01000001_00100010;
//
//
// 01110
// 10001
// 10000
// 10000
// 10000
// 10001
// 01110
// 00000
//
// "D"
//
// parameter noteD = 40'b01111111_01000001_01000001_01000001_00111110;
//
//
// 11110
// 10001
// 10001
// 10001
// 10001
// 10001
// 10001
// 11110
// 00000
//
// "E"
//
// parameter noteE = 40'b01111111_01001001_01001001_01001001_01001001;
//
//
// 11111
// 10000
// 10000
// 11111
// 10000
// 10000
```

```
// 11111
// 00000
//
// "F"
//
// parameter noteF = 40'b01111111_00001001_00001001_00001001_00001001;
//
//
// 11111
// 10000
// 10000
// 11111
// 10000
// 10000
// 10000
// 10000
// 00000
//
// "G"
//
// parameter noteG = 40'b00111110_01000001_01001001_01001001_00111010;
//
//
// 01110
// 10001
// 10000
// 10111
// 10001
// 10001
// 01110
// 00000
//
// "#"
//
// parameter sharp = 40'b00010100_00111110_00010100_00111110_00010100;
//
//
// 00000
// 01010
// 11111
// 01010
// 11111
// 01010
// 00000
// 00000
//
// "X"
//
// parameter noteX = 40'b01100011_00110110_00001000_00110110_01100011;
//
//
// 10001
// 11011
```

```
// 01010
// 00100
// 01010
// 11011
// 10001
// 00000
//
//
//
```

## A.9 alphanumeric\_displays.v

This code was taken directly from the 6.111 code repository and, therefore, is not included here.

## A.10 audio\_sel.v

```
////////////////////////////////////
// audio_sel.v
// by MQ
////////////////////////////////////
module audio_sel(sel, a, b, out);
    input [1:0] sel;
    input [15:0] a, b;
    output [15:0] out;

    reg [15:0] out;

    always @ (*) begin
        case (sel)
            2'b00: out = 16'b0;
            2'b01: out = b;
            2'b10: out = a;
            2'b11: out = a + b;
        endcase // case(sel)
    end
endmodule // audio_sel
```

## A.11 bcontrol.v

```
////////////////////////////////////
// bcontrol.v module to control volume buttons
// by MQ
//
////////////////////////////////////
module bcontrol (resetb, clk, incb, decb, val);

    parameter delta = 5'b0001;
    parameter init_val = 5'b0000;

    input resetb, clk, incb, decb;
    output [4:0] val;
```

```

wire          inc_pu, dec_pu;

fall2pu fall2pu_incb (clk, resetb, incb, inc_pu);
fall2pu fall2pu_decb (clk, resetb, decb, dec_pu);

reg [4:0]     val;
always @ (posedge clk) begin
  if (!resetb) begin
    val <= init_val;
  end else if (inc_pu) begin
    if (val <= (5'b11111 - delta)) val <= val + delta;
    else val <= 5'b11111;
  end else if (dec_pu) begin
    if (val >= (5'b00000 + delta)) val <= val - delta;
    else val <= 5'b00000;
  end
end
end // always @ (posedge clk)
endmodule // bcontrol

```

### A.12 fall2pu.v

```

////////////////////////////////////
// fall2pu.v converts a synchronous falling edge to a one-cycle pulse
// by MQ
////////////////////////////////////
module fall2pu (clk, resetb, in, out);
  input clk, resetb, in;
  output out;

  reg    in_old, out;

  // one state var: in_old
  always @ (posedge clk) begin
    if (!resetb) in_old <= 0;
    else      in_old <= in;
  end

  always @ (*) begin
    if (!in && in_old) out = 1;
    else out = 0;
  end
endmodule // fall2pu

```

### A.13 debounce.v

```

////////////////////////////////////
//
// debounce.v input button/switch debouncer and synchronizer
// with an active-low reset
//
// citation: code adapted from 6.111 staff handout
////////////////////////////////////

```

```

module debounce (reset, clock, noisy, clean);
  parameter DELAY = 368640; // .01 sec with a 368.640Mhz clock
  input reset, clock, noisy;
  output clean;

  reg [18:0] count = 0;
  reg new = 0, clean = 0;

  always @(posedge clock)
  begin
    if (!reset)
      begin
        count <= 0;
        new <= noisy;
        clean <= noisy;
      end
    else if (noisy != new) // if outside input is not
      begin // equal to internal val
        new <= noisy;
        count <= 0;
      end
    else if (count == DELAY)
      clean <= new;
    else
      count <= count+1;
  end
endmodule

```

#### A.14 checker.v

```

/////////////////////////////////////////////////////////////////
// checker.v checkerboard pattern used to debug various vga timings
// by MQ
/////////////////////////////////////////////////////////////////
module checker (clk, resetb, pixel_count, line_count,
               vga_out_red, vga_out_green, vga_out_blue);
  input clk, resetb;
  input [9:0] pixel_count, line_count;
  output [7:0] vga_out_red, vga_out_green, vga_out_blue;
  reg outcolor;
  parameter color1 = 24'b000000000000000000000000;
  parameter color0 = 24'b111111111111111100000000;

  reg state, next;

  assign {vga_out_red, vga_out_green, vga_out_blue} = outcolor ?
    color1 : color0;

  always @ (posedge clk)
  begin
    if (!resetb) state <= 0;
    else state <= next;
  end

```

```

end

always @ (*)
begin
    if (line_count <= 47)        next = 0;
    else if (line_count <= 95)  next = 1;
    else if (line_count <= 143) next = 0;
    else if (line_count <= 191) next = 1;
    else if (line_count <= 239) next = 0;
    else if (line_count <= 287) next = 1;
    else if (line_count <= 335) next = 0;
    else if (line_count <= 383) next = 1;
    else if (line_count <= 431) next = 0;
    else if (line_count <= 479) next = 1;
    else next = 0;
end

always @ (*)
begin
    if (pixel_count <= 63)  outcolor = 0;
    else if (pixel_count <= 123) outcolor = 1;
    else if (pixel_count <= 191) outcolor = 0;
    else if (pixel_count <= 255) outcolor = 1;
    else if (pixel_count <= 319) outcolor = 0;
    else if (pixel_count <= 383) outcolor = 1;
    else if (pixel_count <= 447) outcolor = 0;
    else if (pixel_count <= 511) outcolor = 1;
    else if (pixel_count <= 575) outcolor = 0;
    else if (pixel_count <= 639) outcolor = 1;
    else outcolor = 0;
        if (!state) outcolor = ~outcolor;
end
endmodule

```

## A.15 vgacon.v

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/*
vgacon.v vga control module that generates sync signals and
reports the pixel count.
*/
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
module vgacon (clk, resetb, hsync, vsync, pixel_count,
              line_count, vga_out_sync_b, vga_out_blank_b);

// // timings for 31.5MHz pixel_clock
// parameter h_active_t = 639;
// parameter h_fp_t     = 655; // 640 + 16 - 1;
// parameter h_sync_t   = 751; // 640 + 16 + 96 - 1;
// parameter h_bp_t     = 799; // 640 + 16 + 96 + 48 - 1;
// parameter v_active_t = 479; // 480 - 1
// parameter v_fp_t     = 490; //480 + 11 - 1

```

```

// parameter v_sync_t = 492; //480 + 11 + 2 - 1
// parameter v_bp_t   = 524; //480 + 11 + 2 + 32 - 1

// // timings for 36MHz pixel_clock 640x480
// parameter h_active_t = 639;
// parameter h_fp_t     = 671; // above + 32
// parameter h_sync_t   = 719; // above + 48
// parameter h_bp_t     = 831; // above + 112
// parameter v_active_t = 479; //
// parameter v_fp_t     = 480; //above + 1
// parameter v_sync_t   = 483; //above + 3
// parameter v_bp_t     = 508; //above + 25

// timings for 38.1MHz pixel_clock 800x600
// this is being used with our 36.864MHz clock ...
// these timings work (minor ripples observed)
parameter h_active_t = 799;
parameter h_fp_t     = 831; // above + 32
parameter h_sync_t   = 959; // above + 128
parameter h_bp_t     = 1087; // above + 128
parameter v_active_t = 599; //
parameter v_fp_t     = 600; //above + 1
parameter v_sync_t   = 604; //above + 4
parameter v_bp_t     = 618; //above + 14

input  clk, resetb;
output [10:0] pixel_count, line_count;
output      hsync, vsync, vga_out_sync_b, vga_out_blank_b;

wire      vga_out_sync_b, vga_out_blank_b;

reg [10:0] pixel_count, line_count,
        pixel_count_next, line_count_next;
reg      hsync, vsync;
reg      hblank, vblank;

assign    vga_out_sync_b = 1'b1;
assign    vga_out_blank_b = hblank && vblank;

// The state is simply: {pixel_count, line_count}
always @ (posedge clk)
    begin
        if (!resetb)
            begin
                pixel_count <= 0;
                line_count <= 0;
            end
        else
            begin
                pixel_count <= pixel_count_next;
                line_count <= line_count_next;
            end
    end

```

```
    end
  end

  always @ (*)
  begin

    // code for next pixel location
    pixel_count_next = pixel_count + 1;
    line_count_next = line_count;
    if (pixel_count == h_bp_t)
      begin
        pixel_count_next = 0;
        if (line_count == v_bp_t) line_count_next = 0;
        else line_count_next = line_count + 1;
      end

    // code for hblank hsync
    if (pixel_count <= h_active_t)
      begin
        hblank = 1;
        hsync = 1;
      end
    else if (pixel_count <= h_fp_t)
      begin
        hblank = 0;
        hsync = 1;
      end
    else if (pixel_count <= h_sync_t)
      begin
        hblank = 0;
        hsync = 0;
      end
    else if (pixel_count <= h_bp_t)
      begin
        hblank = 0;
        hsync = 1;
      end
    else
      begin
        hblank = 1;
        hsync = 1;
      end

    // code for vblank vsync
    if (line_count <= v_active_t)
      begin
        vblank = 1;
        vsync = 1;
      end
    else if (line_count <= v_fp_t)
      begin
```

```
        vblank = 0;
        vsync = 1;
    end
    else if (line_count <= v_sync_t)
    begin
        vblank = 0;
        vsync = 0;
    end
    else if (line_count <= v_bp_t)
    begin
        vblank = 0;
        vsync = 1;
    end
    else
    begin
        vblank = 1;
        vsync = 1;
    end
    end
endmodule
```

**A.16 compare\_display.v**

```

//By: Zhongying Zhou
//VGA display
// Instantiates display modules for graphs and text
//inputs are global clock and reset
//pixel count, line count
//music note and singer note for displaying pitch
//video starting signal ie. copy_internal
//update data signals for music, song, fft (new_)
//7 bit data input tone, voice, fft
//8 bit input of measure
//2 bit song_sel switch input
//output: 8 bit RGB signals

module compare_display (clock, reset_sync, pcount, lcount, music_note, singer_note,
                        copy_internal, new_m, new_s, new_f, tone, voice, fft_out,measure,
                        song_sel, vga_red, vga_green, vga_blue);

    input clock, reset_sync;
    input [10:0] pcount,lcount; //x, y coords
    input [3:0] music_note, singer_note;
    input [6:0] tone, voice, fft_out; //input data into the graph modules
    input copy_internal, new_m, new_s, new_f;
    input [7:0] measure; //input is address 11:4
    input [1:0] song_sel;

    output [7:0] vga_red, vga_green, vga_blue;

    //internal wires
    wire [7:0] r, g, b;
    wire [7:0] red_m, green_m, blue_m, red_s, green_s, blue_s,red_f, green_f, blue_f;
    wire [7:0] red_fm, green_fm, blue_fm, red_fs, green_fs, blue_fs, red_ff, green_ff, blue_ff;
    //internal RGB signals

    wire [2:0] pixel1, pixel2, pixel3, pixel4, pixel5; //new stuff
    //output from the existing string display module

    wire [63:0] cstring1= " Song ";
    wire [63:0] cstring2= " Singer ";
    wire [63:0] cstring3= " FFT ";
    //labels for the graphs

    reg [63:0] music_string, singer_string;

    parameter A4= 0;
    parameter AS4# ;
    parameter B4# ;
    parameter C4# ;
    parameter CS4# ;
    parameter D4# ;
    parameter DS4# ;

```

```

parameter E4=7;
parameter F4=8;
parameter FS4=9;
parameter G4=10;
parameter GS4=11;

always @ (music_note or singer_note) begin
//decides which letter to output corresponding to the music note
    case (music_note)
    A4:    music_string= "A";
    AS4:   music_string= "A#";
    B4:    music_string= "B";
    C4:    music_string= "C";
    CS4:   music_string= "C#";
    D4:    music_string= "D";
    DS4:   music_string= "D#";
    E4:    music_string= "E";
    F4:    music_string= "F";
    FS4:   music_string= "F#";
    G4:    music_string= "G";
    GS4:   music_string= "G#";
    default music_string= " ";
    endcase

    case (singer_note)
    //decides which character to output corresponding to the singer note
    A4:    singer_string= "A";
    AS4:   singer_string= "A#";
    B4:    singer_string= "B";
    C4:    singer_string= "C";
    CS4:   singer_string= "C#";
    D4:    singer_string= "D";
    DS4:   singer_string= "D#";
    E4:    singer_string= "E";
    F4:    singer_string= "F";
    FS4:   singer_string= "F#";
    G4:    singer_string= "G";
    GS4:   singer_string= "G#";
    default singer_string= " ";
    endcase
end

char_string_display musics_label(clock, pcount, lcount,pixel4,music_string,10'd550,9'd20);
char_string_display singers_label(clock, pcount, lcount,pixel5,singer_string,10'd550,9'd190);
//display the notes

char_string_display song_label(clock, pcount, lcount,pixel1,cstring1,10'd380,9'd20);
char_string_display singer_label(clock, pcount, lcount,pixel2,cstring2,10'd380,9'd190);
char_string_display fft_label(clock, pcount, lcount,pixel3,cstring3,10'd380,9'd360);
//displays the labels

```

```

reg [63:0] title1_string, title2_string, author1_string, author2_string
wire [2:0] pixelt1, pixelt2, pixela1, pixela2;
//variables for displaying title and artist of song

always @ (posedge clock) begin
  if (reset_sync)
    case(song_sel)
      2'd0: begin
        title1_string<= "Crazy";
        title2_string<="";
        author1_string<="Britney";
        author2_string<="Spears";
      end
      2'd1: begin
        title1_string<= "She";
        title2_string<= "Bangs";
        author1_string<="Ricky";
        author2_string<="Martin";
      end
      default begin
        title1_string<= "Crazy";
        title2_string<="";
        author1_string<="Britney";
        author2_string<="Spears";
      end
    endcase
  else begin
    title1_string<= title1_string;
    title2_string<= title2_string;
    author1_string<=author1_string;
    author2_string<=author2_string;
  end
end

char_string_display title1_label(clock, pcount, lcount, pixelt1, title1_string, 10'd20, 9'd20);
char_string_display title2_label(clock, pcount, lcount, pixelt2, title2_string, 10'd150, 9'd20);
char_string_display artist1_label(clock, pcount, lcount, pixela1, author1_string, 10'd20, 9'd50);
char_string_display artist2_label(clock, pcount, lcount, pixela2, author2_string, 10'd150, 9'd50);
//displays the string

reg [63:0] w1, w2, w3, w4;
wire [2:0] pixw1, pixw2, pixw3, pixw4;

always @ (measure) begin
  case (measure)
    //finds lyrics to corresponding measure
    8'h1: begin
      w1="Baby I'm";
      w2="";
      w3="";
      w4="";
    end
  endcase
end

```

```
end
8'h2: begin
    w1="so into ";
    w2="you    " ;
    w3="";
    w4="";
end
8'h3: begin
    w1="You've ";
    w2="got that" ;
    w3="some-  ";
    w4="thing  ";
end

8'h4: begin
    w1="What   ";
    w2="can I do" ;
    w3="";
    w4="";
end

8'h5: begin
    w1="Baby   ";
    w2="you    " ;
    w3="";
    w4="";
end

8'h6: begin
    w1="spin me ";
    w2="around  " ;
    w3="";
    w4="";
end

8'h7: begin
    w1="The     ";
    w2="earth is" ;
    w3="moving  ";
    w4="but I   ";
end

8'h8: begin
    w1="can't   ";
    w2="feel the" ;
    w3="ground  ";
    w4="";
end

8'h9: begin
    w1="Every   " ;
```

```

        w2="time    " ;
        w3="you     ";
        w4="";
    end

8'hA: begin
    w1="look     ";
    w2="at me   " ;
    w3="";
    w4="";
    end
8'hB: begin
    w1="My heart";
    w2="is      " ;
    w3="jumping ";
    w4="It's    ";
    end
8'hC: begin
    w1="easy to  ";
    w2="see     " ;
    w3="";
    w4="";
    end
8'hD: begin
    w1="Lovin   ";
    w2="you     " ;
    w3="means   ";
    w4="";
    end
8'hE: begin
    w1="so much  ";
    w2="more     " ;
    w3="";
    w4="";
    end
8'hF: begin
    w1="more     ";
    w2="than     " ;
    w3="anything";
    w4="I ever  ";
    end

8'h10: begin
    w1="felt     ";
    w2="before   " ;
    w3="You     ";
    w4="drive me";
    end
8'h11: begin
    w1="crazy    ";
    w2="I       " ;

```

```

        w3="";
        w4="";
    end
8'h12: begin
    w1="just ";
    w2="can't " ;
    w3="sleep ";
    w4="";
    end
8'h13: begin
    w1="I'm so ";
    w2="excited " ;
    w3="I'm ";
    w4="";
    end

8'h14: begin
    w1="in too ";
    w2="deep " ;
    w3="you ";
    w4="drive me";
    end
8'h15: begin
    w1="crazy ";
    w2="but it " ;
    w3="";
    w4="";
    end
8'h16: begin
    w1="feels ";
    w2="alright " ;
    w3="";
    w4="";
    end
8'h17: begin
    w1="Baby ";
    w2="thinkin " ;
    w3="of you ";
    w4="keeps me";
    end
8'h18: begin
    w1="up all ";
    w2="night! " ;
    w3="";
    w4="";
    end

//next song
8'h1B: begin
    w1="Talk ";
    w2="to me " ;

```

```

        w3="";
        w4="";
    end
8'h1C: begin
    w1="tell me ";
    w2="your " ;
    w3="name ";
    w4="";
    end
8'h1D: begin
    w1="You ";
    w2="blow me " ;
    w3="off like";
    w4="it's ";
    end
8'h1E: begin
    w1="all the ";
    w2="same " ;
    w3="";
    w4="";
    end
8'h1F: begin
    w1="You lit ";
    w2="a fuse " ;
    w3="and now ";
    w4="I'm ";
    end
8'h20: begin
    w1="tickin ";
    w2="away " ;
    w3="like ";
    w4="a bomb ";
    end
8'h21: begin
    w1="yeah ";
    w2="ba-(by) " ;
    w3="";
    w4="";
    end
8'h22: begin
    w1="(ba)-by ";
    w2="";
    w3="";
    w4="";
    end
8'h23: begin
    w1="Talk ";
    w2="to me " ;
    w3="";
    w4="";
    end
end

```

```

8'h24: begin
    w1="tell me ";
    w2="your " ;
    w3="sign ";
    w4="";
end
8'h25: begin
    w1="You're ";
    w2="switch- " ;
    w3="in sides";
    w4="like a";
end
8'h26: begin
    w1="Gemini ";
    w2="";
    w3="";
    w4="";
end
8'h27: begin
    w1="You're ";
    w2="playing " ;
    w3="games & ";
    w4="now you ";
end
8'h28: begin
    w1="hittin ";
    w2="my heart" ;
    w3="like a ";
    w4="drum ";
end
8'h29: begin
    w1="yeah ";
    w2="ba-(by) " ;
    w3="";
    w4="";
end
8'h2A: begin
    w1="(ba)-by ";
    w2="";
    w3="Well, if";
    w4="";
end
8'h2B: begin
    w1="Lady ";
    w2="Luck " ;
    w3="gets ";
    w4="";
end
8'h2C: begin
    w1="on my ";
    w2="side " ;

```

```

        w3="we're  ";
        w4="gonna  ";
    end
8'h2D: begin
    w1="rock    ";
    w2="this    "  ;
    w3="town    ";
    w4="alive   ";
    end
8'h2E: begin
    w1="I'll let";
    w2="her     "  ;
    w3="";
    w4="";
    end
8'h2F: begin
    w1="rough   ";
    w2="me up  "  ;
    w3="till she";
    w4="";
    end
8'h30: begin
    w1="knocks  ";
    w2="me out  "  ;
    w3="'cause  ";
    w4="she    ";
    end
8'h31: begin
    w1="walks   ";
    w2="like she"  ;
    w3="talks   ";
    w4="and she  ";
    end
8'h32: begin
    w1="talks   ";
    w2="like she"  ;
    w3="walks   ";
    w4="She     ";
    end
8'h33: begin
    w1="bangs   ";
    w2="she     "  ;
    w3="bangs   ";
    w4="";
    end
8'h34: begin
    w1="Oh baby  ";
    w2="when she"  ;
    w3="";
    w4="";
    end
end

```

```

8'h35: begin
    w1="moves  ";
    w2="she    " ;
    w3="moves  ";
    w4="";
end
8'h36: begin
    w1="I go   ";
    w2="crazy  " ;
    w3="cuz   she ";
    w4="";
end
8'h37: begin
    w1="looks  ";
    w2="like a  " ;
    w3="flower ";
    w4="but she ";
end
8'h38: begin
    w1="stings ";
    w2="like a  " ;
    w3="bee    ";
    w4="";
end
8'h39: begin
    w1="like   ";
    w2="every  " ;
    w3="girl  in ";
    w4="";
end
8'h3A: begin
    w1="history ";
    w2="She    " ;
    w3="";
    w4="";
end
8'h3B: begin
    w1="bangs  ";
    w2="She    " ;
    w3="bangs  ";
    w4="I'm was-";
end
8'h3C: begin
    w1="-ted by ";
    w2="the way " ;
    w3="she    ";
    w4="";
end
8'h3D: begin
    w1="moves  ";
    w2="she    " ;

```

```

        w3="moves  ";
        w4="No";
    end
    8'h3E: begin
        w1="one ever";
        w2="looked " ;
        w3="so      ";
        w4="";
    end
    8'h3F: begin
        w1="fine   ";
        w2="";
        w3="";
        w4="";
    end
    8'h40: begin
        w1="She     ";
        w2="reminds " ;
        w3="me that ";
        w4="a       ";
    end
    8'h41: begin
        w1="woman's ";
        w2="got one  " ;
        w3="thing   ";
        w4="";
    end
    8'h42: begin
        w1="on her   ";
        w2="mind     " ;
        w3="";
        w4="";
    end

    default: begin
        w1="";
        w2="";
        w3="";
        w4="";
    end
endcase
end

char_string_display lyric11(clock, pcount, lcount, pixw1, w1, 10'd40, 9'd150);
char_string_display lyric12(clock, pcount, lcount, pixw2, w2, 10'd40, 9'd190);
char_string_display lyric13(clock, pcount, lcount, pixw3, w3, 10'd40, 9'd230);
char_string_display lyric14(clock, pcount, lcount, pixw4, w4, 10'd40, 9'd270);
//displays lyrics

draw_graph d_music(clock, reset_sync, pcount, lcount, 10'd380, 10'd50, tone, new_m, copy_internal,
               red_m, green_m, blue_m);

```

```

frame frame_m( pcount,lcount,10'd379,10'd49, 10'd401, 10'd129, red_fm, green_fm, blue_fm);
//draws graph and frame for the music

draw_graph d_singer(clock, reset_sync, pcount,lcount,10'd380,10'd220, voice, new_s, copy_intern
                red_s, green_s, blue_s);

frame frame_s( pcount,lcount,10'd379,10'd219, 10'd401, 10'd129, red_fs, green_fs, blue_fs);

//draws graph and frame for singer

draw_fft d_fft(clock, reset_sync, pcount,lcount,10'd0,10'd390, fft_out, new_f,
                red_f, green_f, blue_f);
frame frame_f( pcount,lcount,10'd0,10'd389, 10'd799, 10'd129, red_ff, green_ff, blue_ff);
//draws fft graph and frame

//OR'ing the colours together

assign vga_red= red_m|red_s|red_f|red_fm|red_fs|red_ff
            |{8{pixel1[2]}}|{8{pixel2[2]}}|{8{pixel3[2]}}|{8{pixel4[2]}}|{8{pixel5[2]}}
            |{8{pixelt1[2]}}|{8{pixelt2[2]}}|{8{pixela1[2]}}|{8{pixela2[2]}}
            |{8{pixw1[2]}}|{8{pixw2[2]}}|{8{pixw3[2]}}|{8{pixw4[2]}};
assign vga_green= green_m | green_s | green_f | green_fm | green_fs | green_ff
            |{8{pixel1[1]}}|{8{pixel2[1]}}|{8{pixel3[1]}}|{8{pixel4[1]}}|{8{pixel5[1]}}
            |{8{pixelt1[1]}}|{8{pixelt2[1]}}|{8{pixela1[1]}}|{8{pixela2[1]}}
            |{8{pixw1[1]}}|{8{pixw2[1]}}|{8{pixw3[1]}}|{8{pixw4[1]}};
assign vga_blue= blue_m | blue_s | blue_f | blue_fm | blue_fs | blue_ff
            |{8{pixel1[0]}}|{8{pixel2[0]}}|{8{pixel3[0]}}|{8{pixel4[0]}}|{8{pixel5[0]}}
            |{8{pixelt1[0]}}|{8{pixelt2[0]}}|{8{pixela1[0]}}|{8{pixela2[0]}}
            |{8{pixw1[0]}}|{8{pixw2[0]}}|{8{pixw3[0]}}|{8{pixw4[0]}};

endmodule

```

## A.17 divider.v

```

//By: Zhongying Zhou
//converts a 36.864 MHz clock signal into 10 or 8 or 6 or 4 Hz enable
//global inputs reset, and clock
//user input tempo_sel effective on reset
//FSM input pause_d pauses divider when high
//enable output to the FSM
module divider (reset, clock, tempo_sel, pause_d, enable);
    input reset, clock, pause_d;
    input [1:0] tempo_sel;
    output enable;
    reg [24:0] count; //25 bits required to count 1Hz signal
    reg[24:0] final;
    reg enable;

    parameter LARGO=0; //60 quarter notes/minute

```

```

parameter ANDANTE=1; //90
parameter MODERATO=2; //120
parameter ALLEGRO=3; //150

always @ (posedge clock)

begin
  if (reset) //if reset is high, start count over at 0
  begin
    count<=0;
    enable<=0;
    case (tempo_sel)
      //test for clock = 36.864 MHz
      LARGO: final<= 9215999; //0.25 seconds/1 sixteenth
      ANDANTE: final<= 6143999; //0.1666... seconds
      MODERATO: final<= 4607999; //0.125 s
      ALLEGRO: final<= 3686399; //0.1 s
    endcase
  end
  else if (!pause_d) //user has not paused
  if (count == final)
  //if counter reaches note duration
  begin
    count<=0; //start count over
    enable<=1; //enable is high
  end
  else
  //otherwise enable is low and count is increased
  begin
    enable<=0;
    count <= count +1;
  end
  else
  //user paused so count is the same and enable is low
  begin
    count<=count;
    enable<=0;
  end
end
endmodule

```

## A.18 draw\_fft.v

```

//By: Zhongying Zhou
//draw fft module
//global inputs clock and reset
//pixel count and line count
//then x, y value of the left hand corner
//finally data (0-127) and the data update pulse to signal new data
//outputs the RGB for that certain pixel/line count
//draws a time domain rectangle graph of input signal
//stretches the signal to 800 wide ie 4 pixels per data

```

```

module draw_fft(clock, reset_sync, pcount,lcount,l_x,l_y, data, update,
               red, green, blue);
    input clock, reset_sync;
    input [10:0] pcount,lcount; //x, y coords
    input [10:0] l_x,l_y; //x, y of left corner of box
    input [6:0] data;
    //7 bits ie max of 128 pixel difference in magnitude
    input update; //high for one cycle when new data is available

    output [7:0] red, green, blue; //rgb values
    reg [7:0] red, green, blue;
    reg [6:0] hold_data [199:0]; //stores 200 previous values of data
    reg [7:0] w_ptr; //write pointer up to 200
    reg [6:0] data_y;//data based on input pcount
    reg [10:0]index_x;

    parameter RED= 8'hFF;
    parameter GREEN=8'hFF;
    parameter BLUE= 8'hFF;
    //output colour

    always @ (posedge clock) begin
        if (reset_sync) begin
            w_ptr<=0;
            //initialize write pointer
        end
        else if (update) begin //updates data
            hold_data[w_ptr]<= data; //write data into next slot
            if (w_ptr==199)
                w_ptr<=0; //starts at memory 0 again
            else
                w_ptr<=w_ptr+1; //increases w_ptr
        end
    end

    //rgb values
    always @ (pcount or lcount) begin

        if ((pcount+1)>l_x && pcount< (l_x+800))//within the 800 window width
            begin
                assign index_x=pcount-l_x; //relative index of the data

                if ((index_x[9:2]+w_ptr) <200) //if the index is less than 200
                    //note the opposite direction
                    data_y=hold_data [(w_ptr+ index_x[9:2])];
                    //finds corresponding data mod 200
                else
                    data_y=hold_data [(w_ptr+index_x[9:2]-200)];
                    //finds data at (write pointer + relative index) mod 200
                if ((lcount<(l_y+128)) && ((lcount+{{3{0}},data_y})>(l_y+126))) begin

```

```

        //checks if both above bottom edge and under the data position
        //since
        red=RED;
        green=GREEN;
        blue=BLUE;
        //white
    end
    else begin
        red=8 'h00;
        green=8 'h00;
        blue=8 'h00;
        //black
    end
end
else begin
    red=8 'h00;
    green=8 'h00;
    blue=8 'h00;
    //black

end

end

endmodule

```

## A.19 draw\_graph.v

```

//Zhongying Zhou
//draw graph module
//global inputs clock and reset
//pixel count and line count
//then x, y value of the left hand corner
//finally data (0-127) and the data update pulse to signal new data
//copy signal to make a copy of the data in the array and the pointers
//outputs the RGB for that certain pixel/line count
//draws a time domain line graph of input signal

module draw_graph(clock, reset_sync, pcount,lcount,l_x,l_y, data, update,
                 copy_internal, red, green, blue);
    input clock, reset_sync;
    input [10:0] pcount,lcount; //x, y coords
    input [10:0] l_x,l_y; //x, y of left corner of box
    input [6:0] data; //is input data signed?
    //7 bits ie max of 128 pixel difference in magnitude
    input update, copy_internal; //high for one cycle when new data is available

    output [7:0] red, green, blue; //rgb values
    reg [7:0] red, green, blue;
    reg [6:0] hold_data [399:0]; //stores 400 previous values of data
    reg [6:0] copy_data [399:0]; //copy of data

```

```

reg [8:0] w_ptr, copy_w_ptr; //write pointer up to 400
reg [8:0] r_ptr, copy_r_ptr; //read pointer up to 400
reg [6:0] data_y;//data based on input pcount

parameter RED= 8'hFF;
parameter GREEN=8'hFF;
parameter BLUE= 8'hFF;
//output colour

integer i; //index for copying data

always @ (posedge clock) begin
  if (reset_sync) begin
    w_ptr<=0;
    r_ptr<=0;
    //initialize the write and read pointers
  end
  else if (update) begin //updates data
    hold_data[w_ptr]<= data; //write data into next slot
    r_ptr<=w_ptr; //where the newest data is

    if (w_ptr==399)
      w_ptr<=0; //loops to memory 0 again
    else
      w_ptr<=w_ptr+1; //increases w_ptr
    end

  else if (copy_internal) begin //every screen refresh
    for(i=0; i<400; i=i+1)
      copy_data[i]<=hold_data[i];
      //copy all 400 into another array
    copy_w_ptr<=w_ptr;
    copy_r_ptr<=r_ptr;
    //copy the pointers
  end
end

//rgb outputs
always @ (pcount or lcount) begin

  if ((pcount+1)>l_x && pcount< (l_x+400))//within the 400 window
  begin
    //finds corresponding data accordingly mod 400
    if (pcount <(copy_r_ptr+l_x+1)) //if the index would've been more than 0
      data_y=copy_data [(copy_r_ptr+ l_x-pcount)];
    else
      data_y=copy_data [(copy_r_ptr+l_x-pcount+400)];
      //finds data according to pcount

    if ((lcount+{{4{0}}},data_y)==(l_y+127)) begin
      //if the line count corresponds to the data

```

```

        red=RED;
        green=GREEN;
        blue=BLUE;
        //white
    end
    else begin
    //otherwise output black
        red=8 'h00;
        green=8 'h00;
        blue=8 'h00;
        //black
    end
end
else begin
    red=8 'h00;
    green=8 'h00;
    blue=8 'h00;
    //black
end
end
endmodule

```

## A.20 mr\_fsm.v

```

//By: Zhongying Zhou
//converts a lo-> high transition to a pulse
//global inputs clock and reset
//pause_sync input from debounced user pause signal
//pause_p is the output pulse if pause_sync rises

module pulsify (clock, reset, pause_sync, pause_p);

    input clock, reset, pause_sync;
    output pause_p;

    reg pause1, pause_p;

    always @(posedge clock)
        if (reset) pause_p<=0;
        //initializes pause_p to be 0
        else
        begin
            pause1<=~pause_sync;
            //inverted and delayed pause
            pause_p<=(pause1 && pause_sync);
            //pulse
        end
endmodule

```

**A.21 rectangle.v**

```

//By: Zhongying Zhou
//Module for outputting a frame
//inputs are the pixel and line count and the top left coordinates of frame
//also the width and height of the fram
//outputs are the 8 bits for red, green and blue each

module frame (pcount,lcount,f_x,f_y, w, h,vga_out_red, vga_out_green, vga_out_blue);
input [10:0] pcount,lcount,f_x,f_y,w,h;
output [7:0] vga_out_red, vga_out_green, vga_out_blue;
reg [7:0] vga_out_red, vga_out_green, vga_out_blue;

    parameter RED= 8'hFF;
    parameter GREEN=8'hFF;
    parameter BLUE= 8'hFF;
    //displays white

    always @ (pcount or lcount)
        if (((pcount==f_x) || (pcount==(f_x+ w))) && ((lcount+1)>f_y) && (lcount<(f_y+ h+1)))
            //if on the left and right edges of the rectangle
            //assign rectangle RGB values
            //white
            vga_out_red= RED;
            vga_out_green= GREEN;
            vga_out_blue= BLUE;
            end

        else if (((pcount+1)>f_x) && (pcount<(f_x+ w+1)) && ((lcount==f_y) || (lcount==f_y+ h)))
            //top and bottom edges of the rectangle
            //assign rectangle RGB values
            //white
            vga_out_red= RED;
            vga_out_green= GREEN;
            vga_out_blue= BLUE;
            end

        else begin
            //not on the edges
            //black
            vga_out_red= 8'h00;
            vga_out_green= 8'h00;
            vga_out_blue= 8'h00;
            end

endmodule

```