Appendix

Figure 1. Test Bench for Menu FSM

Figure 2. Test Bench #1 for Game FSM

Figure 3. Test Bench #2 for Game FSM

Figure 4. Test Bench for Report Module
Figure 4. Test Bench for Score Keeper Module

Figure 5. Test Bench #1 for Control Unit

Figure 6. Test Bench #2 for Control Unit

Figure 7. Test Bench for Accuracy Controller
Figure 8. Test Bench for Arrow Controller

Figure 9: Picture of the grid created by the 6 infrared sensors.

Figure 10: Close up shot of the circuitry for each Sharp trigger.

Figure 11: Picture of the sensors grid, and how it translates into up, down, right, left footwork.
Figures 12, 13, and 14: Screen shots, in respective order, of the menu screen, in game screen, and report card screen.
Figures 15, 16, and 17: Sensor Specification Sheets (3 pages)
**Electro-optical Characteristics**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Conditions</th>
<th>MIN</th>
<th>TYP</th>
<th>MAX</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distance measuring range</td>
<td>ΔL</td>
<td>≥-1</td>
<td>20</td>
<td>—</td>
<td>150</td>
<td>cm</td>
</tr>
<tr>
<td>Output terminal voltage</td>
<td>V_{OH}</td>
<td>Output voltage at high level</td>
<td>V_{CC}-0.3</td>
<td>—</td>
<td>—</td>
<td>V</td>
</tr>
<tr>
<td>Output terminal voltage</td>
<td>V_{OL}</td>
<td>Output voltage at low level</td>
<td>—</td>
<td>—</td>
<td>0.6</td>
<td>V</td>
</tr>
<tr>
<td>Distance characteristics of output</td>
<td>V_{O}</td>
<td>—</td>
<td>70</td>
<td>80</td>
<td>90</td>
<td>cm</td>
</tr>
<tr>
<td>Average dissipation current</td>
<td>I_{CC}</td>
<td>—</td>
<td>—</td>
<td>23</td>
<td>50</td>
<td>mA</td>
</tr>
</tbody>
</table>

**Note:**
1. Distance to reflective object
2. Using reflective object: White paper (Made by Kodak Co., Ltd, gray cards R-27, white face, reflective ratio 99%)
3. We ship the device after the following adjustment: Output vs. sensing distance L; Terminal 1 should meet the output
4. Distance measuring range of the optical sensor system
5. Output switching has a hysteresis width. The distance specified by V_{O} should be the one with which the output I switches to the output II.

---

**Fig.1 Internal Block Diagram**

![Internal Block Diagram](image)

---

**Fig.2 Timing Chart**

![Timing Chart](image)
Verilog Project Code
Labkit.v

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,

dauthercard,

systemace_data, systemace_address, systemace_ce_b,
systemace_we_b, systemace_ce_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;
in  put 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_ycrb;
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_ycrb;
in  put 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;
in input systemace_irq, systemace_mpbrdy;

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

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

// Audio Input and Output
assign beep= 1'b0;
// assign audio_reset_b = 1'b0;
// assign ac97_synch = 1'b0;
// assign ac97_sdata_out = 1'b0;

// 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_infifo_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;

// 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;

// 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;

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

// LED Displays
assign disp_blank = 1'b1;
assign disp_clock = 1'b0;
assign disp_rs = 1'b0;
assign disp_ce_b = 1'b1;
assign disp_reset_b = 1'b0;
assign disp_data_out = 1'b0;

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

// User I/Os
assign user1 = 32'hZ;
assign user2 = 32'hZ;
// assign user3 = 32'hZ;
// assign user4 = 32'hZ;
 assign user3[31:6] = 26'b11111111111111111111111111;

// Daughtercard Connectors
assign daughtercard = 44'hZ;

// SystemACE Microprocessor Port
assign systemace_data = 16'hZ;
assign systemace_address = 7'h0;
assign systemace_ce_b = 1'b1;
assign systemace_we_b = 1'b1;
assign systemace_oe_b = 1'b1;

// Logic Analyzer
assign analyzer1_data = 16'h0;
assign analyzer1_clock = 1'b1;
assign analyzer3_data = 16'h0;
assign analyzer3_clock = 1'b1;

// Generate a 31.5MHz pixel clock from clock_27mhz
wire pclk, pixel_clock;
DCM pixel_clock_dcm (.CLKIN(clock_27mhz), .CLKFX(pclk));
// synthesis attribute CLKFX_DIVIDE of pixel_clock_dcm is 6
// synthesis attribute CLKFX_MULTIPLY of pixel_clock_dcm is 7
// synthesis attribute CLK_FEEDBACK of pixel_clock_dcm is "NONE"
// synthesis attribute CLKIN_PERIOD of pixel_clock_dcm is 37
BUFG pixel_clock_buf (.I(pclk), .O(pixel_clock));

// VGA output signals

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

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

// The following assignments should be deleted and replaced with your own
// code to implement the Pong game.
//assign vga_out_red = 8'h0;
//assign vga_out_green = 8'h0;
//assign vga_out_blue = 8'h0;
//assign vga_out_blank_b = 1'b1;
//assign vga_out_hsync = 1'b0;
//assign vga_out_vsync = 1'b0;
wire [9:0] pixel_count, line_count, piblock_y1, piblock_y2, piblock_y3,
    piblock_y4, piblock_y5, piblock_y6, piblock_y7, piblock_y8,
    piblock_y9, piblock_y10;
wire ready1, ready2, ready3, ready4, ready5, ready6, ready7, ready8,
    ready9, ready10, ready;
wire up1, up2, up3, up4, up5, up6, up7, up8, up9, up10;
wire down1, down2, down3, down4, down5, down6, down7, down8, down9,
    down10;
wire right1, right2, right3, right4, right5, right6, right7, right8,
    right9, right10;
wire left1, left2, left3, left4, left5, left6, left7, left8, left9,
    left10;
wire reset_sync, up_sync, down_sync, right_sync, left_sync, select;
wire [5:0] sensors;
wire [1:0] difficulty, mode, letter_score;
wire [5:0] accuracy;
wire [3:0] arrows;
wire [23:0] rgb_out;
wire vga_out_vsync;
wire frame_sync, m_busy_g, start_game, stop, start_song;
wire [9:0] beat_count;
wire enable, enable2;
wire [10:0] score;
wire IR1, IR2, IR3, IR4, IR5, IR6, IR1_temp, IR2_temp, IR3_temp,
    IR4_temp, IR5_temp, IR6_temp, up_temp, down_temp, right_temp,
    left_temp;
wire control_mode;

assign frame_sync = ((pixel_count == 639) && (line_count == 479));
assign vga_out_red = rgb_out[23:16];
assign vga_out_green = rgb_out[15:8];
assign vga_out_blue = rgb_out[7:0];
assign analyzer2_data = {sensors[0], /*accuracy[1:0]*/2'b0, score[10:0], 2'b0};
assign analyzer2_clock = vga_out_pixel_clock;
assign analyzer4_data = {beat_count[9:0], arrows[3:0], ready, start_game};
assign analyzer4_clock = vga_out_pixel_clock;
assign reset_sync = sensors[5];

// assign led[7:4] = 4'b1111;
assign led[3] = ~sensors[0];
assign led[2] = ~sensors[1];
assign led[1] = ~sensors[2];
assign led[0] = ~sensors[3];
//assign led[7:6] = ~difficulty;
//assign led[4:3] = ~letter_score;
//assign led[5] = 1'b1;
//assign led[2:0] = 1'b1;

assign led[7:4] = ~score[3:0];

// assign mode = switch[3:2];

Control_Unit game_control(pixel_clock, frame_sync, accuracy, sensors, ready, letter_score,
arrows, start_song, difficulty, mode, m_busy_g, beat_count, score, enable, stop);

assign user4[31:26] = 6'b111111;
wire [7:0] data;
assign data = user4[25:18];

audio roboto_song(reset_sync, pixel_clock, audio_reset_b, ac97_sdata_out, ac97_sdata_in,
ac97_synch, ac97_bit_clock, start_song, stop, user4[17:0],
data);

Decode_IR_Sensors woohoo(pixel_clock, reset_sync, IR1, IR2, IR3, IR4, IR5, IR6, sensors[2],
sensors[1], sensors[3], sensors[0],
up_temp, down_temp, right_temp,
left_temp, control_mode);
up10, down1, down2, down3, down4, down5, down6, down7, down8, down9, down10, right1, right2, right3, right4, right5, right6, right7, right8, right9, right10, left1, left2, left3, left4, left5, left6, left7, left8, left9, left10, control_mode);

arrow_controller arrow_c(pixel_clock, reset_sync, up1, up2, up4, up5, up6, up7, up8, up9, up10, down1, down2, down3, down4, down5, down6, down7, down8, down9, down10, right1, right2, right3, right4, right5, right6, right7, right8, right9, right10, left1, left2, left3, left4, left5, left6, left7, left8, left9, left10, ready1, ready2, ready3, ready4, ready5, ready6, ready7, ready8, ready9, ready10, arrows, ready);

divider menu_enable(pixel_clock, reset_sync, enable);

piblock piblock1(.pixel_clock(pixel_clock), .reset_sync(reset_sync), .piblock_y(piblock_y1), .frame_sync(frame_sync), .ready(ready1), .ystart(10’d550), .start_game(start_song));

piblock piblock2(.pixel_clock(pixel_clock), .reset_sync(reset_sync), .piblock_y(piblock_y2), .frame_sync(frame_sync), .ready(ready2), .ystart(10’d490), .start_game(start_song));

piblock piblock3(.pixel_clock(pixel_clock), .reset_sync(reset_sync), .piblock_y(piblock_y3), .frame_sync(frame_sync), .ready(ready3), .ystart(10’d430), .start_game(start_song));

piblock piblock4(.pixel_clock(pixel_clock), .reset_sync(reset_sync), .piblock_y(piblock_y4), .frame_sync(frame_sync), .ready(ready4), .ystart(10’d370), .start_game(start_song));

piblock piblock5(.pixel_clock(pixel_clock), .reset_sync(reset_sync), .piblock_y(piblock_y5), .frame_sync(frame_sync), .ready(ready5), .ystart(10’d310), .start_game(start_song));

piblock piblock6(.pixel_clock(pixel_clock), .reset_sync(reset_sync), .piblock_y(piblock_y6), .frame_sync(frame_sync), .ready(ready6), .ystart(10’d250), .start_game(start_song));

piblock piblock7(.pixel_clock(pixel_clock), .reset_sync(reset_sync), .piblock_y(piblock_y7), .frame_sync(frame_sync), .ready(ready7), .ystart(10’d190), .start_game(start_song));

piblock piblock8(.pixel_clock(pixel_clock), .reset_sync(reset_sync), .piblock_y(piblock_y8), .frame_sync(frame_sync), .ready(ready8), .ystart(10’d130), .start_game(start_song));

piblock piblock9(.pixel_clock(pixel_clock), .reset_sync(reset_sync), .piblock_y(piblock_y9), .frame_sync(frame_sync), .ready(ready9), .ystart(10’d70), .start_game(start_song));

piblock piblock10(.pixel_clock(pixel_clock), .reset_sync(reset_sync), .piblock_y(piblock_y10), .frame_sync(frame_sync), .ready(ready10), .ystart(10’d10), .start_game(start_song));
debounce reset_debounce(.reset(1'b0), .clock(vga_out_pixel_clock), .noisy(~button_enter),
   .clean(sensors[5]));
   defparam reset_debounce.DELAY = 270000;

debounce up_debounce(.reset(1'b0), .clock(vga_out_pixel_clock), .noisy(~button_up), .clean(up_temp));
   debounce down_debounce(.reset(1'b0), .clock(vga_out_pixel_clock), .noisy(~button_down),
   .clean(down_temp));

debounce right_debounce(.reset(1'b0), .clock(vga_out_pixel_clock), .noisy(~button_right),
   .clean(right_temp));

debounce left_debounce(.reset(1'b0), .clock(vga_out_pixel_clock), .noisy(~button_left), .clean(left_temp));

debounce select_debounce(.reset(1'b0), .clock(vga_out_pixel_clock), .noisy(~button0), .clean(sensors[4]));
   defparam select_debounce.DELAY = 270000;

IRdebounce IRdbIR1(reset_sync, pixel_clock, IR1_temp, IR1);
IRdebounce IRdbIR2(reset_sync, pixel_clock, IR2_temp, IR2);
IRdebounce IRdbIR3(reset_sync, pixel_clock, IR3_temp, IR3);
IRdebounce IRdbIR4(reset_sync, pixel_clock, IR4_temp, IR4);
IRdebounce IRdbIR5(reset_sync, pixel_clock, IR5_temp, IR5);
IRdebounce IRdbIR6(reset_sync, pixel_clock, IR6_temp, IR6);

debounce dbIR1(.reset(1'b0), .clock(vga_out_pixel_clock), .noisy(user3[0]), .clean(IR1_temp));
debounce dbIR2(.reset(1'b0), .clock(vga_out_pixel_clock), .noisy(user3[1]), .clean(IR2_temp));
debounce dbIR3(.reset(1'b0), .clock(vga_out_pixel_clock), .noisy(user3[2]), .clean(IR3_temp));
debounce dbIR4(.reset(1'b0), .clock(vga_out_pixel_clock), .noisy(user3[3]), .clean(IR4_temp));
debounce dbIR5(.reset(1'b0), .clock(vga_out_pixel_clock), .noisy(user3[4]), .clean(IR5_temp));
debounce dbIR6(.reset(1'b0), .clock(vga_out_pixel_clock), .noisy(user3[5]), .clean(IR6_temp));

debounce dbcontrol_mode(.reset(1'b0), .clock(vga_out_pixel_clock), .noisy(switch[7]),
   .clean(control_mode));

endmodule
Accuracy Controller
'timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////
//
//Sharmeen Browarek
//Anna Ayuso
//6.111 Final Project
//TA: Jae Lee
//
////////////////////////////////////////////////////////////////////////////

//This module is in charge of sending the control unit a 6-bit accuracy signal. The range of accuracy
//for the moving arrows is determined by comparing the dynamic piblocks to the static ones at the
//top of the screen. Each time a block nears the top, a 2-bit signal is sent to the control unit, along
//with 4 bits that designate which arrows of that block are lit.
//There are two sets of ranges because a delay had to be factored in when the user is playing in the
//sensors mode.

module accuracy_controller(pixel_clock, reset_sync, piblock_y1, piblock_y2, piblock_y3, piblock_y4, piblock_y5, piblock_y6, piblock_y7, piblock_y8, piblock_y9, piblock_y10, accuracy, up1, up2, up3, up4, up5, up6, up7, up8, up9, up10, down1, down2, down3, down4, down5, down6, down7, down8, down9, down10, right1, right2, right3, right4, right5, right6, right7, right8, right9, right10, left1, left2, left3, left4, left5, left6, left7, left8, left9, left10, control_mode);

input pixel_clock, reset_sync, control_mode;
input [9:0] piblock_y1, piblock_y2, piblock_y3, piblock_y4, piblock_y5, piblock_y6, piblock_y7, piblock_y8, piblock_y9, piblock_y10;
input up1, up2, up3, up4, up5, up6, up7, up8, up9, up10;
input down1, down2, down3, down4, down5, down6, down7, down8, down9, down10;
input right1, right2, right3, right4, right5, right6, right7, right8, right9, right10;
input left1, left2, left3, left4, left5, left6, left7, left8, left9, left10;

output [5:0] accuracy;

reg [5:0] accuracy;
reg [9:0] range1, range2, range3, range4, range5;

always @ (posedge pixel_clock)
begin
    if (reset_sync)
        begin
            accuracy <= 6'd0;
            range1 <= 10'd10;
            range2 <= 10'd15;
            range3 <= 10'd40;
            range4 <= 10'd55;
        end
        range1 <= 10'd20;
        range2 <= 10'd30;
        range3 <= 10'd60;
        range4 <= 10'd70;
    end
end
range5 <= 10'd60;
if (control_mode == 1)
begin
range1 <= 10'd510;
range2 <= 10'd515;
range3 <= 10'd545;
range4 <= 10'd565;
range5 <= 10'd570;
end
if (control_mode == 0)
begin
range1 <= 10'd10;
range2 <= 10'd25;
range3 <= 10'd40;
range4 <= 10'd50;
range5 <= 10'd60;
end

//check for piblock 1
if ((piblock_y1 >= range1) && (piblock_y1 < range2))
begin
accuracy[5:4] <= 2'd1;
accuracy[3] <= left1;
accuracy[2] <= up1;
accuracy[1] <= down1;
accuracy[0] <= right1;
end
else if ((piblock_y1 >= range2) && (piblock_y1 < range3))
begin
accuracy[5:4] <= 2'd3;
accuracy[3] <= left1;
accuracy[2] <= up1;
accuracy[1] <= down1;
accuracy[0] <= right1;
end
else if ((piblock_y1 >= range3) && (piblock_y1 < range4))
begin
accuracy[5:4] <= 2'd2;
accuracy[3] <= left1;
accuracy[2] <= up1;
accuracy[1] <= down1;
accuracy[0] <= right1;
end
else if ((piblock_y1 >= range4) && (piblock_y1 < range4))
begin
accuracy[5:4] <= 2'd1;
accuracy[3] <= left1;
accuracy[2] <= up1;
accuracy[1] <= down1;
accuracy[0] <= right1;
end

//check for piblock 2
if ((piblock_y2 >= range1) && (piblock_y2 < range2))
begin
accuracy[5:4] <= 2’d1;
accuracy[3] <= left2;
accuracy[2] <= up2;
accuracy[1] <= down2;
accuracy[0] <= right2;
end
else if ((piblock_y2 >= range2) && (piblock_y2 < range3))
begin
accuracy[5:4] <= 2’d3;
accuracy[3] <= left2;
accuracy[2] <= up2;
accuracy[1] <= down2;
accuracy[0] <= right2;
end
else if ((piblock_y2 >= range3) && (piblock_y2 < range4))
begin
accuracy[5:4] <= 2’d2;
accuracy[3] <= left2;
accuracy[2] <= up2;
accuracy[1] <= down2;
accuracy[0] <= right2;
end
else if ((piblock_y2 >= range4) && (piblock_y2 < range4))
begin
accuracy[5:4] <= 2’d1;
accuracy[3] <= left2;
accuracy[2] <= up2;
accuracy[1] <= down2;
accuracy[0] <= right2;
end

//check for piblock 3
if ((piblock_y3 >= range1) && (piblock_y3 < range2))
begin
accuracy[5:4] <= 2’d1;
accuracy[3] <= left3;
accuracy[2] <= up3;
accuracy[1] <= down3;
accuracy[0] <= right3;
end
else if ((piblock_y3 >= range2) && (piblock_y3 < range3))
begin
accuracy[5:4] <= 2’d3;
accuracy[3] <= left3;
accuracy[2] <= up3;
accuracy[1] <= down3;
accuracy[0] <= right3;
end
else if ((piblock_y3 >= range3) && (piblock_y3 < range4))
begin
accuracy[5:4] <= 2’d2;
accuracy[3] <= left3;
accuracy[2] <= up3;
accuracy[1] <= down3;
accuracy[0] <= right3;
end
else if ((piblock_y3 >= range4) && (piblock_y3 < range4))
begin
accuracy[5:4] <= 2’d1;
accuracy[3] <= left3;
accuracy[2] <= up3;
accuracy[1] <= down3;
accuracy[0] <= right3;
end
else if ((piblock_y3 >= range4) && (piblock_y3 < range4))
begin
  accuracy[5:4] <= 2'd1;
  accuracy[3] <= left3;
  accuracy[2] <= up3;
  accuracy[1] <= down3;
  accuracy[0] <= right3;
end

// check for piblock 4
if ((piblock_y4 >= range1) && (piblock_y4 < range2))
begin
  accuracy[5:4] <= 2'd1;
  accuracy[3] <= left4;
  accuracy[2] <= up4;
  accuracy[1] <= down4;
  accuracy[0] <= right4;
end
else if ((piblock_y4 >= range2) && (piblock_y4 < range3))
begin
  accuracy[5:4] <= 2'd3;
  accuracy[3] <= left4;
  accuracy[2] <= up4;
  accuracy[1] <= down4;
  accuracy[0] <= right4;
end
else if ((piblock_y4 >= range3) && (piblock_y4 < range4))
begin
  accuracy[5:4] <= 2'd2;
  accuracy[3] <= left4;
  accuracy[2] <= up4;
  accuracy[1] <= down4;
  accuracy[0] <= right4;
end
else if ((piblock_y4 >= range4) && (piblock_y4 < range4))
begin
  accuracy[5:4] <= 2'd1;
  accuracy[3] <= left4;
  accuracy[2] <= up4;
  accuracy[1] <= down4;
  accuracy[0] <= right4;
end

// check for piblock 5
if ((piblock_y5 >= range1) && (piblock_y5 < range2))
begin
  accuracy[5:4] <= 2'd1;
  accuracy[3] <= left5;
  accuracy[2] <= up5;
  accuracy[1] <= down5;
  accuracy[0] <= right5;
end
else if ((piblock_y5 >= range2) && (piblock_y5 < range3))
begin
  accuracy[5:4] <= 2'd3;
  accuracy[3] <= left5;
  accuracy[2] <= up5;
  accuracy[1] <= down5;
  accuracy[0] <= right5;
end

else if ((piblock_y5 >= range3) && (piblock_y5 < range4))
begin
  accuracy[5:4] <= 2'd2;
  accuracy[3] <= left5;
  accuracy[2] <= up5;
  accuracy[1] <= down5;
  accuracy[0] <= right5;
end

else if ((piblock_y5 >= range4) && (piblock_y5 < range4))
begin
  accuracy[5:4] <= 2'd1;
  accuracy[3] <= left5;
  accuracy[2] <= up5;
  accuracy[1] <= down5;
  accuracy[0] <= right5;
end

//check for piblock 6
if ((piblock_y6 >= range1) && (piblock_y6 < range2))
begin
  accuracy[5:4] <= 2'd1;
  accuracy[3] <= left6;
  accuracy[2] <= up6;
  accuracy[1] <= down6;
  accuracy[0] <= right6;
end

else if ((piblock_y6 >= range2) && (piblock_y6 < range3))
begin
  accuracy[5:4] <= 2'd3;
  accuracy[3] <= left6;
  accuracy[2] <= up6;
  accuracy[1] <= down6;
  accuracy[0] <= right6;
end

else if ((piblock_y6 >= range3) && (piblock_y6 < range4))
begin
  accuracy[5:4] <= 2'd2;
  accuracy[3] <= left6;
  accuracy[2] <= up6;
  accuracy[1] <= down6;
  accuracy[0] <= right6;
end

else if ((piblock_y6 >= range4) && (piblock_y6 < range4))
begin
  accuracy[5:4] <= 2'd1;
  accuracy[3] <= left6;
  accuracy[2] <= up6;
  accuracy[1] <= down6;
end

36
accuracy[0] <= right6;
end

//check for piblock 7
if ((piblock_y7 >= range1) && (piblock_y7 < range2))
begin
  accuracy[5:4] <= 2'd1;
  accuracy[3] <= left7;
  accuracy[2] <= up7;
  accuracy[1] <= down7;
  accuracy[0] <= right7;
end
else if ((piblock_y7 >= range2) && (piblock_y7 < range3))
begin
  accuracy[5:4] <= 2'd3;
  accuracy[3] <= left7;
  accuracy[2] <= up7;
  accuracy[1] <= down7;
  accuracy[0] <= right7;
end
else if ((piblock_y7 >= range3) && (piblock_y7 < range4))
begin
  accuracy[5:4] <= 2'd2;
  accuracy[3] <= left7;
  accuracy[2] <= up7;
  accuracy[1] <= down7;
  accuracy[0] <= right7;
end
else if ((piblock_y7 >= range4) && (piblock_y7 < range4))
begin
  accuracy[5:4] <= 2'd1;
  accuracy[3] <= left7;
  accuracy[2] <= up7;
  accuracy[1] <= down7;
  accuracy[0] <= right7;
end

//check for piblock 8
if ((piblock_y8 >= range1) && (piblock_y8 < range2))
begin
  accuracy[5:4] <= 2'd1;
  accuracy[3] <= left8;
  accuracy[2] <= up8;
  accuracy[1] <= down8;
  accuracy[0] <= right8;
end
else if ((piblock_y8 >= range2) && (piblock_y8 < range3))
begin
  accuracy[5:4] <= 2'd3;
  accuracy[3] <= left8;
  accuracy[2] <= up8;
  accuracy[1] <= down8;
  accuracy[0] <= right8;
end
else if ((piblock_y8 >= range3) && (piblock_y8 < range4))
    begin
        accuracy[5:4] <= 2'd2;
        accuracy[3] <= left8;
        accuracy[2] <= up8;
        accuracy[1] <= down8;
        accuracy[0] <= right8;
    end
else if ((piblock_y8 >= range4) && (piblock_y8 < range4))
    begin
        accuracy[5:4] <= 2'd1;
        accuracy[3] <= left8;
        accuracy[2] <= up8;
        accuracy[1] <= down8;
        accuracy[0] <= right8;
    end

//check for piblock 9
if ((piblock_y9 >= range1) && (piblock_y9 < range2))
    begin
        accuracy[5:4] <= 2'd1;
        accuracy[3] <= left9;
        accuracy[2] <= up9;
        accuracy[1] <= down9;
        accuracy[0] <= right9;
    end
else if ((piblock_y9 >= range2) && (piblock_y9 < range3))
    begin
        accuracy[5:4] <= 2'd3;
        accuracy[3] <= left9;
        accuracy[2] <= up9;
        accuracy[1] <= down9;
        accuracy[0] <= right9;
    end
else if ((piblock_y9 >= range3) && (piblock_y9 < range4))
    begin
        accuracy[5:4] <= 2'd2;
        accuracy[3] <= left9;
        accuracy[2] <= up9;
        accuracy[1] <= down9;
        accuracy[0] <= right9;
    end
else if ((piblock_y9 >= range4) && (piblock_y9 < range4))
    begin
        accuracy[5:4] <= 2'd1;
        accuracy[3] <= left9;
        accuracy[2] <= up9;
        accuracy[1] <= down9;
        accuracy[0] <= right9;
    end

//check for piblock 10
if ((piblock_y10 >= range1) && (piblock_y10 < range2))
    begin

end
accuracy[5:4] <= 2'd1;
accuracy[3] <= left10;
accuracy[2] <= up10;
accuracy[1] <= down10;
accuracy[0] <= right10;
end

else if ((piblock_y10 >= range2) && (piblock_y10 < range3))
begin
accuracy[5:4] <= 2'd3;
accuracy[3] <= left10;
accuracy[2] <= up10;
accuracy[1] <= down10;
accuracy[0] <= right10;
end

else if ((piblock_y10 >= range3) && (piblock_y10 < range4))
begin
accuracy[5:4] <= 2'd2;
accuracy[3] <= left10;
accuracy[2] <= up10;
accuracy[1] <= down10;
accuracy[0] <= right10;
end

else if ((piblock_y10 >= range4) && (piblock_y10 < range4))
begin
accuracy[5:4] <= 2'd1;
accuracy[3] <= left10;
accuracy[2] <= up10;
accuracy[1] <= down10;
accuracy[0] <= right10;
end

if ((piblock_y1 <= range1 || piblock_y1 >= range5) &&
    (piblock_y2 <= range1 || piblock_y2 >= range5) &&
    (piblock_y3 <= range1 || piblock_y3 >= range5) &&
    (piblock_y4 <= range1 || piblock_y4 >= range5) &&
    (piblock_y5 <= range1 || piblock_y5 >= range5) &&
    (piblock_y6 <= range1 || piblock_y6 >= range5) &&
    (piblock_y7 <= range1 || piblock_y7 >= range5) &&
    (piblock_y8 <= range1 || piblock_y8 >= range5) &&
    (piblock_y9 <= range1 || piblock_y9 >= range5) &&
    (piblock_y10 <= range1 || piblock_y10 >= range5))
accuracy[5:4] <= 2'd0;
end
endmodule
// This module is in charge of interpreting the arrows signal from the control unit. Whenever a dynamic // piblock is reset, it sends a ready signal to the control unit. In return, the control unit sends the 4-bit // signal designating the new assignment for which arrows are lit. The arrow controller takes in this // signal and sends it to the correct dynamic piblock on the screen.

class Arrow Controller:

'timescale 1ns / 1ps

// This module is in charge of interpreting the arrows signal from the control unit. Whenever a dynamic // piblock is reset, it sends a ready signal to the control unit. In return, the control unit sends the 4-bit // signal designating the new assignment for which arrows are lit. The arrow controller takes in this // signal and sends it to the correct dynamic piblock on the screen.

module arrow_controller(pixel_clock, reset_sync, up1, up2, up3, up4, up5, up6, up7, up8, up9, up10, down1, down2, down3, down4, down5, down6, down7, down8, down9, down10, right1, right2, right3, right4, right5, right6, right7, right8, right9, right10, left1, left2, left3, left4, left5, left6, left7, left8, left9, left10, ready1, ready2, ready3, ready4, ready5, ready6, ready7, ready8, ready9, ready10, arrows, ready);

input pixel_clock, reset_sync;
input [3:0] arrows;
input ready1, ready2, ready3, ready4, ready5, ready6, ready7, ready8, ready9, ready10;
output up1, up2, up3, up4, up5, up6, up7, up8, up9, up10;
output down1, down2, down3, down4, down5, down6, down7, down8, down9, down10;
output right1, right2, right3, right4, right5, right6, right7, right8, right9, right10;
output left1, left2, left3, left4, left5, left6, left7, left8, left9, left10;
output ready;

reg up1, up2, up3, up4, up5, up6, up7, up8, up9, up10;
reg down1, down2, down3, down4, down5, down6, down7, down8, down9, down10;
reg right1, right2, right3, right4, right5, right6, right7, right8, right9, right10;
reg left1, left2, left3, left4, left5, left6, left7, left8, left9, left10;
reg ready;
reg left, up, down, right;

reg ready1_temp, ready2_temp, ready3_temp, ready4_temp, ready5_temp, ready6_temp, ready7_temp, ready8_temp, ready9_temp, ready10_temp, ready_flag;
/*assign left = arrows[3];
assign up = arrows[2];
assign down = arrows[1];
assign right = arrows[0];*/

always @ (posedge pixel_clock)
begin
    if (reset_sync)
        begin
            ready <= 0;
            up1 <= 0;
            up2 <= 0;
            up3 <= 0;
            up4 <= 0;
            up5 <= 0;
            up6 <= 0;
            up7 <= 0;
            up8 <= 0;
            up9 <= 0;
            up10 <= 0;
            down1 <= 0;
            down2 <= 0;
            down3 <= 0;
            down4 <= 0;
            down5 <= 0;
            down6 <= 0;
            down7 <= 0;
            down8 <= 0;
            down9 <= 0;
            down10 <= 0;
            right1 <= 0;
            right2 <= 0;
            right3 <= 0;
            right4 <= 0;
            right5 <= 0;
            right6 <= 0;
            right7 <= 0;
            right8 <= 0;
            right9 <= 0;
            right10 <= 0;
            left1 <= 0;
            left2 <= 0;
            left3 <= 0;
            left4 <= 0;
            left5 <= 0;
            left6 <= 0;
            left7 <= 0;
            left8 <= 0;
            left9 <= 0;
            left10 <= 0;
        end
    else if (ready_flag)
        begin
            ready <= 1;
        end
end
if (ready1_temp)
    begin
        up1 <= arrows[2];
        down1 <= arrows[1];
        right1 <= arrows[0];
        left1 <= arrows[3];
    end

if (ready2_temp)
    begin
        up2 <= arrows[2];
        down2 <= arrows[1];
        right2 <= arrows[0];
        left2 <= arrows[3];
    end

if (ready3_temp)
    begin
        up3 <= arrows[2];
        down3 <= arrows[1];
        right3 <= arrows[0];
        left3 <= arrows[3];
    end

if (ready4_temp)
    begin
        up4 <= arrows[2];
        down4 <= arrows[1];
        right4 <= arrows[0];
        left4 <= arrows[3];
    end

if (ready5_temp)
    begin
        up5 <= arrows[2];
        down5 <= arrows[1];
        right5 <= arrows[0];
        left5 <= arrows[3];
    end

if (ready6_temp)
    begin
        up6 <= arrows[2];
        down6 <= arrows[1];
        right6 <= arrows[0];
        left6 <= arrows[3];
    end

if (ready7_temp)
    begin
        up7 <= arrows[2];
        down7 <= arrows[1];
        right7 <= arrows[0];
        left7 <= arrows[3];
    end

if (ready8_temp)
    begin
        up8 <= arrows[2];
        down8 <= arrows[1];
        right8 <= arrows[0];
        left8 <= arrows[3];
    end
if (ready9_temp)
  begin
    up9 <= arrows[2];
    down9 <= arrows[1];
    right9 <= arrows[0];
    left9 <= arrows[3];
  end
if (ready10_temp)
  begin
    up10 <= arrows[2];
    down10 <= arrows[1];
    right10 <= arrows[0];
    left10 <= arrows[3];
  end
else
  ready <= 0;
end

always @ (ready1 or ready2 or ready3 or ready4 or ready5 or ready6 or ready7 or ready8 or ready9 or ready10)
begin
  ready1_temp = 0;
  ready2_temp = 0;
  ready3_temp = 0;
  ready4_temp = 0;
  ready5_temp = 0;
  ready6_temp = 0;
  ready7_temp = 0;
  ready8_temp = 0;
  ready9_temp = 0;
  ready10_temp = 0;
  ready_flag = 0;
  if (ready1 || ready2 || ready3 || ready4 ||
      ready5 || ready6 || ready7 || ready8 ||
      ready9 || ready10)
    ready_flag = 1;
  if (ready1)
    ready1_temp = 1;
  if (ready2)
    ready2_temp = 1;
  if (ready3)
    ready3_temp = 1;
  if (ready4)
    ready4_temp = 1;
  if (ready5)
    ready5_temp = 1;
  if (ready6)
    ready6_temp = 1;
  if (ready7)
    ready7_temp = 1;
  if (ready8)
    ready8_temp = 1;
if (ready9)
    ready9_temp = 1;
if (ready10)
    ready10_temp = 1;
end
endmodule
Audio
/**
Anna Ayuso
Audio module
handles playing the Mr. Roboto song
adapted from:
The 6.111 SFX (sound effects module), fall 2005
By Eric Fellheimer
AC'97 Sound driver by Nathan Ickes
reset - stop playing sound and go idle
audio_reset_b, ac97* - feed directly to corresponding signal in labkit
play - pulse high to start a sound effect
mode - selects with of the 4 sound effects to play (latched in on rising edge of play)
/**/
module Audio(reset, clock_27mhz, audio_reset_b, ac97_sdata_out, ac97_sdata_in, ac97_synch, ac97_bit_clock,
    play, stop, address, data);
    parameter VOL_PARAM = 5'd30;
    input reset, clock_27mhz;
    output audio_reset_b;
    output ac97_sdata_out;
    input ac97_sdata_in;
    output ac97_synch;
    input ac97_bit_clock;
    //sfx interface
    input play; //start playing selected effect
        input stop;
        input [7:0] data;
        output [17:0] address;
        wire ready;
        wire [7:0] command_address;
        wire [15:0] command_data;
        wire command_valid;
        wire [19:0] left_out_data;
        wire [19:0] right_out_data;
        wire [19:0] left_in_data, right_in_data;
        wire [4:0] volume;
        wire source;
        //hard code volume/source
        assign volume = VOL_PARAM; //a reasonable volume
        assign source = 1'b1; //microphone
// Reset controller
//
reg audio_reset_b;
reg [9:0] reset_count;

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

wire one_time_reset;
SRL16 reset_sr (.D(1'b0), .CLK(clock_27mhz), .Q(one_time_reset), .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
defparam reset_sr.INIT = 16'hFFFF;
always @(posedge clock_27mhz) begin
if (one_time_reset)
  begin
    audio_reset_b <= 1'b0;
    reset_count <= 0;
  end
else if (reset_count == 1023)
  audio_reset_b <= 1'b1;
else
  reset_count <= reset_count+1;
end

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

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

sound_fsm sound_fsm(reset, clock_27mhz, play, ready, left_out_data, right_out_data, stop, address, data);
endmodule

module sound_fsm(reset, clock_27mhz, play, ready, left_out_data, right_out_data, stop, address, data);

input reset, clock_27mhz;
input ready;
input play;
  input stop;

output [19:0] left_out_data;
output [19:0] right_out_data;
input [7:0] data;
output [17:0] address;

parameter S_IDLE = 2'd0;
parameter S_START = 2'd1;
parameter S_PLAY = 2'd2;

reg [19:0] left_out_data;
reg [1:0] state = S_IDLE;

// control signals for effects modules
wire done;
wire start;
wire [19:0] pcm_out;
reg done_cur;
reg stop_playing;
reg old_ready;
always @ (posedge clock_27mhz)
    old_ready <= reset ? 0 : ready;
assign next_sample = (ready && ~old_ready);

// instantiate the FX modules
MrRoboto    fx3(reset, clock_27mhz, next_sample, pcm_out, start, done, address, data);

always @ (posedge clock_27mhz)
if(reset || stop)
  //
  begin
  state <= S_IDLE;
  //
  stop_playing <= 0;
  //
  end
/* else if (stop)
  begin
    stop_playing <= 1;
    state <= S_IDLE;
  end
  else if (stop_playing)
    state <= S_IDLE;*/
else
  case (state)
  S_IDLE: 
    if(play)
      begin
        state <= S_START;
      end
  S_START : state <= S_PLAY;
  S_PLAY: state <= done_cur ? S_IDLE : state;
default: state <= S_IDLE;
endcase

assign start = (state == S_START);

always @(done)
done_cur = done;

always @(state or pcm_out)
if(state == S_PLAY)
    left_out_data = pcm_out;
else
    //left_out_data = square_data;
    left_out_data = 20'h00000;
    // left_out_data = left_out_data;
//end always
assign right_out_data = left_out_data; // mono output
endmodule

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

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

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

reg ready;

reg ac97_sdata_out;
reg ac97_synch;

reg [7:0] bit_count;
initial begin
  ready <= 1'b0;
  // synthesis attribute init of ready is "0"
  ac97_sdata_out <= 1'b0;
  // synthesis attribute init of ac97_sdata_out is "0"
  ac97_synch <= 1'b0;
  // synthesis attribute init of ac97_synch is "0"
  bit_count <= 8'h00;
  // synthesis attribute init of bit_count is "0000"
  l_cmd_v <= 1'b0;
  // synthesis attribute init of l_cmd_v is "0"
  l_left_v <= 1'b0;
  // synthesis attribute init of l_left_v is "0"
  l_right_v <= 1'b0;
  // synthesis attribute init of l_right_v is "0"
  left_in_data <= 20'h00000;
  // synthesis attribute init of left_in_data is "00000"
  right_in_data <= 20'h00000;
  // synthesis attribute init of right_in_data is "00000"
end

always @(posedge ac97_bit_clock) begin
  // Generate the sync signal
  if (bit_count == 255)
    ac97_synch <= 1'b1;
  if (bit_count == 15)
    ac97_synch <= 1'b0;
  // Generate the ready signal
  if (bit_count == 128)
    ready <= 1'b1;
  if (bit_count == 2)
    ready <= 1'b0;

  // Latch user data at the end of each frame. This ensures that the
  // first frame after reset will be empty.
  if (bit_count == 255)
    begin
      l_cmd_addr <= {command_address, 12'h000};
      l_cmd_data <= {command_data, 4'h0};
      l_cmd_v <= command_valid;
      l_left_data <= left_data;
      l_left_v <= left_valid;
      l_right_data <= right_data;
      l_right_v <= right_valid;
    end
if ((bit_count >= 0) && (bit_count <= 15))
  // Slot 0: Tags
  case (bit_count[3:0])
    4'h0: ac97_sdata_out <= 1'b1; // Frame valid
    4'h1: ac97_sdata_out <= l_cmd_v; // Command address valid
    4'h2: ac97_sdata_out <= l_cmd_v; // Command data valid
    4'h3: ac97_sdata_out <= l_left_v; // Left data valid
    4'h4: ac97_sdata_out <= l_right_v; // Right data valid
    default: ac97_sdata_out <= 1'b0;
  endcase

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

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

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

else if ((bit_count >= 76) && (bit_count <= 95))
  // Slot 4: Right channel
  ac97_sdata_out <= l_right_v ? l_right_data[95-bit_count] : 1'b0;
else
  ac97_sdata_out <= 1'b0;

bit_count <= bit_count+1;

end // always @(posedge ac97_bit_clock)

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

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

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

endmodule
input [4:0] volume;
input source;

reg [23:0] command;
reg command_valid;
reg old_ready;
reg done;
reg [3:0] state;

initial begin
    command <= 4'h0;
    // synthesis attribute init of command is "0";
    command_valid <= 1'b0;
    // synthesis attribute init of command_valid is "0";
    done <= 1'b0;
    // synthesis attribute init of done is "0";
    old_ready <= 1'b0;
    // synthesis attribute init of old_ready is "0";
    state <= 16'h0000;
    // synthesis attribute init of state is "0000";
end

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

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

always @(posedge clock) begin
    if (ready && (!old_ready))
        state <= state+1;
    case (state)
        4'h0: // Read ID
            begin
                command <= 24'h80_0000;
                command_valid <= 1'b1;
            end
        4'h1: // Read ID
            command <= 24'h80_0000;
        4'h2: // Master volume
            command <= { 8'h02, 3'b000, vol, 3'b000, vol };
        4'h3: // Aux volume
            command <= { 8'h04, 3'b000, vol, 3'b000, vol };
        4'h4: // Mono volume
            command <= 24'h06_8000;
        4'h5: // PCM volume
            command <= 24'h18_0808;
        4'h6: // Record source select
            if (source)
                command <= 24'h1A_0000; // microphone
            else
                command <= 24'h1A_0404; // line-in
        4'h7: // Record gain
            command <= 24'h1C_0000;
    endcase
end
module MrRoboto (reset, clock, next_sample, pcm_data, start, done, address, data);

input reset;
input clock;
input next_sample;
input start;

output [19:0] pcm_data;
output done;

input [7:0] data;
output [17:0] address;

reg old_ready;
reg [19:0] pcm_data;
reg [17:0] count;
reg [3:0] hold_count;
wire [17:0] address;
wire [7:0] data;
//reg [2:0] hold_count;

parameter LAST_COUNT = 209566; //the amount of lines of memory

//      roboto m(address,clock,data);

always @ (posedge clock)
begin
  if(reset || start)
    begin
      count <= 0; //count goes from 0 to the number of samples of the song
      hold_count <= 0; //hold count counts to 9, this is how long samples are held for
    end
end //because of downsampling
else if (next_sample)
    begin

        count <= (~done && hold_count == 8) ? count + 1 : count;  //because it's downsampled by 10,
        hold_count <= (hold_count == 8) ? 0: hold_count + 1;  //should be counting to 9, however
            //using 9 resulted in the song being slowed down
        end
    else
        begin
            count <= count;
            hold_count <= hold_count;
        end

    end

assign done = (count >= LAST_COUNT);
    assign address = count;


// Now actually output tone...

always @ (posedge clock)
    begin
        if(start)
            begin
                pcm_data <= {data, 12'd0};
            end
        if(next_sample)
            pcm_data <= {data, 12'd0};  //end of the song padded with 0's to increase volume
        begin
        end
// Display an ASCII encoded character string in a video window at some
// specified x,y pixel location.
//
// INPUTS:
//
// vclock - video pixel clock
// hcount - horizontal (x) location of current pixel
// vcount - vertical (y) location of current pixel
// cstring - character string to display (8 bit ASCII for each char)
// cx, cy - pixel location (upper left corner) to display string at
//
// OUTPUT:
//
// pixel - video pixel value to display at current location
//
// PARAMETERS:
//
// NCHAR - number of characters in string to display
// NCHAR_BITS - number of bits to specify NCHAR
//
// pixel should be OR'ed (or XOR'ed) to your video data for display.
//
// Each character is 8x12, but pixels are doubled horizontally and vertically
// so fonts are magnified 2x. On an XGA screen (1024x768) you can fit
// 64 x 32 such characters.
//
// Needs font_rom.v and font_rom.ngo
//
// For different fonts, you can change font_rom. For different string
// display colors, change the assignment to cpixel.

// video character string display

module char_string_display (vclock,hcount,vcount,pixel,cstring,cx,cy);

parameter NCHAR = 1; // number of 8-bit characters in cstring
parameter NCHAR_BITS = 7; // number of bits in NCHAR
    parameter mscale = 1;
    parameter dscale = 1;

input vclock; // 65MHz clock
input [10:0] hcount; // horizontal index of current pixel (0..1023)
input [9:0] vcount; // vertical index of current pixel (0..767)
output [2:0] pixel; // char display's pixel
input [NCHAR*8-1:0] cstring; // character string to display
input [10:0] cx;
input [9:0] cy;

// 1 line x 8 character display (8 x 12 pixel-sized characters)
wire [10:0] hoff = mscale*(hcount-1-cx)/dscale;
wire [9:0] voff = mscale*(vcount-cy)/dscale;
wire [NCHAR_BITS-1:0] column = NCHAR-1-hoff[NCHAR_BITS-1+4:4]; // < NCHAR
wire [2:0] h = hoff[3:1]; // 0 .. 7
wire [3:0] v = voff[4:1]; // 0 .. 11

// look up character to display (from character string)
reg [7:0] char;
integer n;
always @( *)
    for (n=0 ; n<8 ; n = n+1 ) // 8 bits per character (ASCII)
        char[n] <= cstring[column*8+n];

// look up raster row from font rom
wire reverse = char[7];
wire [10:0] font_addr = char[6:0]*12 + v; // 12 bytes per character
wire [7:0] font_byte;
font_rom f(font_addr,vclock,font_byte);

// generate character pixel if we're in the right h,v area
wire [2:0] cpixel = (font_byte[7 - h] ^ reverse) ? 7 : 0;
wire dispflag = ((hcount > cx) & (vcount >= cy) & (hcount <= cx+NCHAR*16/mscale*dscale) & (vcount < cy + 24/mscale*dscale));
wire [2:0] pixel = dispflag ? cpixel : 0;
endmodule
Control Unit

//Anna Ayuso
//Control Unit, major FSM for system
//this module controls the logic of the game

module Control_Unit(pixel_clock, frame_sync, accuracy, sensors, ready, letter_score,
arrows, start_song, difficulty, mode, m_busy_g, beat_count, score, enable, stop);

input pixel_clock, ready, frame_sync, enable;
input [5:0] sensors;
input [5:0] accuracy;

output [1:0] difficulty, mode;
output [1:0] letter_score;
output [3:0] arrows;
output start_song, stop;
output m_busy_g;
output [9:0] beat_count;
output [10:0] score;

reg [1:0] mode;
wire [9:0] beat_count, song_rows;
wire [10:0] score;
wire r_busy_r, g_busy_r, g_busy_g, m_busy_m, m_busy_g, enable, fail;
wire [1:0] difficulty, letter_score;

//instantiations of the four submodules
Score_Keeper sk(pixel_clock, sensors[5], frame_sync, sensors, accuracy, score, g_busy_r);

Report_FSM rfsm(pixel_clock, sensors[5], frame_sync, score, difficulty, r_busy_r,
letter_score, song_rows, g_busy_r, fail);

Game_FSM gfsm(pixel_clock, sensors[5], frame_sync, m_busy_g, difficulty, score,
ready, g_busy_r, g_busy_g,
start_song, beat_count, arrows, song_rows, stop, fail, letter_score);

Menu_FSM mfsm(pixel_clock, sensors[5], enable, sensors[2], sensors[1],
sensors[4], m_busy_m, m_busy_g, difficulty);

parameter menufsm = 0;
parameter gamefsm = 1;
parameter reportfsm = 2;

reg [1:0] state, nextstate;
reg change_mode;

always @ (posedge pixel_clock)
begin
if (sensors[5])
begin
state <= menufsm;
mode <= 2’d0;
end
else if (frame_sync)
begin
state <= nextstate;
if (change_mode)
begin
if (mode == 2’d0)
    mode <= 2’d1;
else if (mode == 2’d1)
    mode <= 2’d2;
else
    mode <= 2’d2;
end
else mode <= mode;
end
end
always @ (state)
begin
//this fsm changes states based on busy signals from
change_mode = 0; //the minor fsms, the output is the current mode of the system
//the progression is menu->game->report card
case(state)
    menufsm: begin
        if (m_busy_m == 1)
            nextstate = menufsm;
        else
            begin
                nextstate = gamefsm;
                change_mode = 1;
            end
    end
    gamefsm: begin
        if (g_busy_g == 1)
            nextstate = gamefsm;
        else
            begin
                nextstate = reportfsm;
                change_mode = 1;
            end
    end
    reportfsm:
        nextstate = reportfsm;
    default: nextstate = menufsm;
endcase
end
endmodule
Debounce
// Switch Debounce Module
// use your system clock for the clock input
// to produce a synchronous, debounced output

// All of the inputs to the FPGA are sent through this module, which outputs a clean version of the signal.
// Basically, this module filters out any noise.

module debounce (reset, clock, noisy, clean);
  parameter DELAY = 400000; // .01 sec with a 27Mhz clock
  input reset, clock, noisy;
  output clean;

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

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

This module decodes the outputs of the 6 sensors and decides when the user is in the up, down, right and left positions on the grid.

```verilog
module Decode_IR_Sensors(pixel_clock, reset_sync, IR1, IR2, IR3, IR4, IR5, IR6, up, down, left, right, up_temp, down_temp, right_temp, left_temp, control_mode);

input reset_sync, pixel_clock;
input IR1, IR2, IR3, IR4, IR5, IR6, up_temp, down_temp, right_temp, left_temp;
input control_mode;
output up, down, left, right;
wire up_deb, down_deb, right_deb, left_deb;
reg up, down, left, right;
assign up_temp2 = (IR2 && IR4);
assign down_temp2 = (IR2 && IR6);
assign left_temp2 = (IR3 && IR5);
assign right_temp2 = (IR1 && IR5);
// assign reset = (IR1 && IR4);
// assign select = (IR3 && IR4);

IRdebounce dbIRup(.reset(reset_sync), .clock(pixel_clock), .noisy(up_temp2), .extraclean(up_deb));
IRdebounce dbIRdown(.reset(reset_sync), .clock(pixel_clock), .noisy(down_temp2), .extraclean(down_deb));
IRdebounce dbIRright(.reset(reset_sync), .clock(pixel_clock), .noisy(right_temp2), .extraclean(right_deb));
IRdebounce dbIRleft(.reset(reset_sync), .clock(pixel_clock), .noisy(left_temp2), .extraclean(left_deb));

always @ (posedge pixel_clock)
  if (reset_sync)
    begin
      up <= up_temp;
      down <= down_temp;
      right <= right_temp;
      left <= left_temp;
    end
  else if (control_mode == 1)
    begin
      up <= up_deb;
    end
endmodule
```
down <= down_deb;
right <= right_deb;
left <= left_deb;
end
else if (control_mode == 0)
begin
up <= up_temp;
down <= down_temp;
right <= right_temp;
left <= left_temp;
end
endmodule
module divider(pixel_clock, reset_sync, enable);
    input pixel_clock, reset_sync;
    output enable;

    wire enable;
    reg [24:0] count;

    parameter maxcount = 15145000;

    assign enable = (count == (maxcount - 1));

    always@(posedge pixel_clock)
    begin
        if (reset_sync == 1) begin
            count <= 25'd0;
        end

        else if (count == maxcount) begin
            count <= 25'd0;
        end

        else begin
            count <= count + 1;
        end
    end
endmodule

//This module outputs an enable signal every half second which is sent to the control unit.
Font ROM

`timescale 1ns/1ps

module font_rom(
    addr,
    clk,
    dout);

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

// synopsys translate_off

BLKMEMSP_V6_1 #(  
    11,   // c_addr_width  
    "0",  // c_default_data

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

    `timescale 1ns/1ps
1536, // 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
"font_rom.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_yyclock_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 font_rom is "true"

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

endmodule
Game_FSM

//Anna Ayuso
//Game Play Minor FSM
//this module controls the arrows that are sent to the video
//this module only continues outputting arrows if the song hasn't ended
//and the score is high enough

module Game_FSM(pixel_clock, reset_sync, frame_sync, start_game, difficulty,
    score, ready, busy_r, busy_g, start_song, beat_count, arrows,
    song_rows, stop_song, fail, letter_score);

input pixel_clock, reset_sync, frame_sync, ready, start_game;
input [1:0] difficulty;
input [10:0] score;
input [1:0] letter_score;

output busy_r, busy_g, start_song, stop_song, fail;
output [9:0] beat_count;
output [3:0] arrows;
output [9:0] song_rows;

reg busy_r, busy_g, start_song, stop_song;
reg [9:0] beat_count;
reg [3:0] arrows;
reg [10:0] min_score;
reg [9:0] song_rows;

reg t_busy_r, t_busy_g, t_start_song, inc_count, t_stop_song;

reg [3:0] song [0:99];
reg load_song, flip_ls;
reg fail;

parameter initialize = 0;
parameter move_arrows = 1;
parameter goto_report = 2;

reg [1:0] state, nextstate;

always @(posedge pixel_clock)
    begin
        if (reset_sync)
            begin
                state <= initialize;
                busy_g <= 1;
                busy_r <= 0;
                start_song <= 0;
                beat_count <= 10'd0;
                load_song <= 1;
                arrows <= 4'd0;
                min_score <= 11'd0;
            end
    end

65
song_rows <= 10'd0;
stop_song <= 0;
buffer3 <= 0;
end

else if (frame_sync && start_game) //start_game will be high when the menu fsm
begin                          //is in the goto game state
    if (flip_ls)
        load_song <= 0;
    if (load_song)          //the song is loaded once based on the difficulty
        begin
            //selected
            arrows <= 4'd0;
            case (difficulty)   //the song is stored as an array of registers
                0:
                begin
                    song[0] = 4'b0000;
                    song[1] = 4'b0100;
                    song[2] = 4'b0010;
                    song[3] = 4'b0000;
                    song[4] = 4'b0010;
                    song[5] = 4'b0000;
                    song[6] = 4'b0000;
                    song[7] = 4'b0000;
                    song[8] = 4'b0010;
                    song[9] = 4'b0000;
                    song[10] = 4'b0100;
                    song[11] = 4'b0000;
                    song[12] = 4'b0100;
                    song[13] = 4'b0000;
                    song[14] = 4'b0100;
                    song[15] = 4'b0000;
                    song[16] = 4'b0100;
                    song[17] = 4'b0000;
                    song[18] = 4'b0010;
                    song[19] = 4'b0010;
                    song[20] = 4'b0000;
                    song[21] = 4'b0000;
                    song[22] = 4'b0000;
                    song[23] = 4'b1000;
                    song[24] = 4'b0000;
                    song[25] = 4'b0000;
                    song[26] = 4'b0100;
                    song[27] = 4'b0000;
                    song[28] = 4'b0010;
                    song[29] = 4'b0000;
                    song[30] = 4'b0000;
                    song[31] = 4'b1000;
                    song[32] = 4'b1000;
                    song[33] = 4'b0000;
                    song[34] = 4'b0001;
                    song[35] = 4'b0000;
                    song[36] = 4'b0000;
                    song[37] = 4'b0001;
                    song[38] = 4'b0010;
            end;
            end case;
        end;
song[39] = 4'b0010;
song[40] = 4'b0010;
song[41] = 4'b0000;
song[42] = 4'b0010;
song[43] = 4'b0001;
song[44] = 4'b0010;
song[45] = 4'b0100;
song[46] = 4'b0010;
song[47] = 4'b0000;
song[48] = 4'b0001;
song[49] = 4'b0010;
song[50] = 4'b0000;
song[51] = 4'b0100;
song[52] = 4'b0000;
song[53] = 4'b0010;
song[54] = 4'b0000;
song[55] = 4'b0100;
song[56] = 4'b0000;
song[57] = 4'b0100;
song[58] = 4'b1000;
song[59] = 4'b0010;
song[60] = 4'b0010;
song[61] = 4'b0000;
song[62] = 4'b0000;
song[63] = 4'b0001;
song[64] = 4'b0001;
song[65] = 4'b0100;
song[66] = 4'b0010;
song[67] = 4'b0000;
song[68] = 4'b0100;
song[69] = 4'b0000;
song[70] = 4'b0010;
song[71] = 4'b0000;
song[72] = 4'b0010;
song[73] = 4'b0100;
song[74] = 4'b0100;
song[75] = 4'b0100;
song[76] = 4'b0000;
song[77] = 4'b0010;
song[78] = 4'b0001;
song[79] = 4'b0001;
song[80] = 4'b0100;
song[81] = 4'b1000;
song[82] = 4'b0010;
song[83] = 4'b1000;
song[84] = 4'b0001;
song[85] = 4'b0001;
song[86] = 4'b1000;
song[87] = 4'b0100;
song[88] = 4'b0010;
song[89] = 4'b0100;
song[90] = 4'b1000;
song[91] = 4'b0000;
song[92] = 4'b0001;
song[93] = 4'b0100;
song[94] = 4'b1000;
song[95] = 4'b0000;
song[96] = 4'b0010;
song[97] = 4'b0100;
song[98] = 4'b0000;
song[99] = 4'b0001;
end

1:
begin
  song[0] = 4'b0000;
song[1] = 4'b0100;
song[2] = 4'b0010;
song[3] = 4'b0000;
song[4] = 4'b0010;
song[5] = 4'b0000;
song[6] = 4'b0000;
song[7] = 4'b0000;
song[8] = 4'b0010;
song[9] = 4'b0000;
song[10] = 4'b0100;
song[11] = 4'b0000;
song[12] = 4'b0100;
song[13] = 4'b0000;
song[14] = 4'b0100;
song[15] = 4'b0000;
song[16] = 4'b0100;
song[17] = 4'b0000;
song[18] = 4'b0010;
song[19] = 4'b0010;
song[20] = 4'b0000;
song[21] = 4'b0000;
song[22] = 4'b0000;
song[23] = 4'b1000;
song[24] = 4'b0000;
song[25] = 4'b0000;
song[26] = 4'b0100;
song[27] = 4'b0000;
song[28] = 4'b0010;
song[29] = 4'b0000;
song[30] = 4'b0000;
song[31] = 4'b0000;
song[32] = 4'b0100;
song[33] = 4'b0000;
song[34] = 4'b0001;
song[35] = 4'b0000;
song[36] = 4'b0000;
song[37] = 4'b0001;
song[38] = 4'b0010;
song[39] = 4'b0010;
song[40] = 4'b0010;
song[41] = 4'b0000;
song[42] = 4'b0010;
song[43] = 4'b0001;
song[44] = 4'b0010;
song[45] = 4'b0100;
song[46] = 4'b0010;
song[47] = 4'b0000;
song[48] = 4'b0001;
song[49] = 4'b0010;
song[50] = 4'b0000;
song[51] = 4'b0100;
song[52] = 4'b0000;
song[53] = 4'b0010;
song[54] = 4'b0000;
song[55] = 4'b0100;
song[56] = 4'b0000;
song[57] = 4'b0100;
song[58] = 4'b1000;
song[59] = 4'b0010;
song[60] = 4'b0010;
song[61] = 4'b0000;
song[62] = 4'b0000;
song[63] = 4'b0011;
song[64] = 4'b0001;
song[65] = 4'b0100;
song[66] = 4'b0010;
song[67] = 4'b0000;
song[68] = 4'b0100;
song[69] = 4'b0000;
song[70] = 4'b0010;
song[71] = 4'b0000;
song[72] = 4'b0010;
song[73] = 4'b0100;
song[74] = 4'b0100;
song[75] = 4'b0100;
song[76] = 4'b0000;
song[77] = 4'b0010;
song[78] = 4'b0001;
song[79] = 4'b0001;
song[80] = 4'b0100;
song[81] = 4'b1000;
song[82] = 4'b0010;
song[83] = 4'b1000;
song[84] = 4'b0001;
song[85] = 4'b0001;
song[86] = 4'b1000;
song[87] = 4'b0100;
song[88] = 4'b0010;
song[89] = 4'b0100;
song[90] = 4'b1000;
song[91] = 4'b0000;
song[92] = 4'b0001;
song[93] = 4'b0100;
song[94] = 4'b1000;
song[95] = 4'b0000;
song[96] = 4'b0010;
song[97] = 4'b0100;
song[98] = 4'b0000;
song[99] = 4'b0001;
end
2:
begin
song[0] = 4'b0000;
song[1] = 4'b0100;
song[2] = 4'b0010;
song[3] = 4'b0000;
song[4] = 4'b0010;
song[5] = 4'b0000;
song[6] = 4'b0000;
song[7] = 4'b0000;
song[8] = 4'b0010;
song[9] = 4'b0000;
song[10] = 4'b0100;
song[11] = 4'b0000;
song[12] = 4'b0100;
song[13] = 4'b0000;
song[14] = 4'b0100;
song[15] = 4'b0000;
song[16] = 4'b0100;
song[17] = 4'b0000;
song[18] = 4'b0010;
song[19] = 4'b0010;
song[20] = 4'b0000;
song[21] = 4'b0000;
song[22] = 4'b0000;
song[23] = 4'b1000;
song[24] = 4'b0000;
song[25] = 4'b0000;
song[26] = 4'b0100;
song[27] = 4'b0000;
song[28] = 4'b0010;
song[29] = 4'b0000;
song[30] = 4'b0000;
song[31] = 4'b1000;
song[32] = 4'b1000;
song[33] = 4'b0000;
song[34] = 4'b0001;
song[35] = 4'b0000;
song[36] = 4'b0000;
song[37] = 4'b0001;
song[38] = 4'b0010;
song[39] = 4'b0010;
song[40] = 4'b0010;
song[41] = 4'b0000;
song[42] = 4'b0010;
song[43] = 4'b0001;
song[44] = 4'b0010;
song[45] = 4'b0100;
song[46] = 4'b0010;
song[47] = 4'b0000;
song[48] = 4'b0001;
song[49] = 4'b0010;
song[50] = 4'b0000;
song[51] = 4'b0100;
song[52] = 4'b0000;
song[53] = 4'b0010;
song[54] = 4'b0000;
song[55] = 4'b0100;
song[56] = 4'b0000;
song[57] = 4'b0100;
song[58] = 4'b1000;
song[59] = 4'b0010;
song[60] = 4'b0010;
song[61] = 4'b0000;
song[62] = 4'b0000;
song[63] = 4'b0001;
song[64] = 4'b0001;
song[65] = 4'b0100;
song[66] = 4'b0010;
song[67] = 4'b0000;
song[68] = 4'b0100;
song[69] = 4'b0000;
song[70] = 4'b0010;
song[71] = 4'b0000;
song[72] = 4'b0010;
song[73] = 4'b0100;
song[74] = 4'b0100;
song[75] = 4'b0100;
song[76] = 4'b0000;
song[77] = 4'b0010;
song[78] = 4'b0001;
song[79] = 4'b0001;
song[80] = 4'b0100;
song[81] = 4'b1000;
song[82] = 4'b0010;
song[83] = 4'b1000;
song[84] = 4'b0001;
song[85] = 4'b0001;
song[86] = 4'b1000;
song[87] = 4'b0100;
song[88] = 4'b0010;
song[89] = 4'b0100;
song[90] = 4'b1000;
song[91] = 4'b0000;
song[92] = 4'b0001;
song[93] = 4'b0100;
song[94] = 4'b1000;
song[95] = 4'b0000;
song[96] = 4'b0010;
song[97] = 4'b0100;
song[98] = 4'b0000;
song[99] = 4'b0001;
endcase
end

else arrows <= song[beat_count];
state <= nextstate;
busy_g <= t_busy_g;
busy_r <= t_busy_r;
start_song <= t_start_song;
stop_song <= t_stop_song;

// the beat count is incremented when the inc_count flag
// in the fsm is set high
if (inc_count)
    beat_count <= beat_count + 1;
else
    beat_count <= beat_count;

// song rows is a count of the beats (i.e. elements of song array)
// that have at least one arrow, this is for score keeping
    song_rows <= song_rows + 1;
else
    song_rows <= song_rows;

end

always @ (state)
begin
    t_busy_r = 0;
t_busy_g = 1;
t_start_song = 0;
t_stop_song = 0;
inc_count = 0;
flip_ls = 0;
fail = 0;
//
arrows = 4'd0;
end

case(state)
initialize: begin
    t_start_song = 1; // tells audio to start music
    nextstate = move_arrows;
fip_ls = 1; // the song is only loaded once
end

move_arrows: begin
    if (beat_count >= 115) // the song ends
        nextstate = goto_report;
    else if (ready) // ready is from the video, signals that
begin //video is ready for the next set of arrows
    if (song_rows <= 15 || letter_score != 2'd3) //checks if the
        begin
            inc_count = 1;
            arrows = song[beat_count];
            nextstate = move_arrows;
            end
    else
        begin
            fail = 1;
            nextstate = goto_report;
            end
end
else
    nextstate = move_arrows;

end

goto_report: begin
    t_busy_r = 1;
    t_busy_g = 0;
    t_stop_song = 1;
    nextstate = goto_report;
end

default: nextstate = initialize;
endcase
end

endmodule
module image(pixel_clock, reset_sync, frame_sync, pixel_count, line_count, bitmap_rgb);

input pixel_clock, reset_sync, frame_sync;
input [9:0] pixel_count, line_count;
output [23:0] bitmap_rgb;
wire bitmapb, bitmapc, bitmape, bitmapf;
wire imgenable;
wire [16:0] count;

ingrectangle rect_img1(pixel_count, line_count, imgenable);
image_counter counter_img1(pixel_clock, reset_sync, imgenable, count);
beavera img2(count, pixel_clock, bitmapb);
beaverb img3(count, pixel_clock, bitmapc);
beaverc img5(count, pixel_clock, bitmape);
beaverd img6(count, pixel_clock, bitmapf);
image_loop imgloop(pixel_clock, reset_sync, frame_sync, bitmap_rgb, bitmapb, bitmapc, bitmape, bitmapf);
endmodule

module image_counter(pixel_clock, reset_sync, imgenable, count); //addr = count
input pixel_clock, reset_sync, imgenable;
output [16:0] count;
reg [16:0] count;
parameter maxcount = 27594;
always @(posedge pixel_clock)
begin
    if (reset_sync == 1) begin
        count <= 17'd0;
    end

    else if (count == maxcount) begin
        count <= 17'd0;
    end

    else if (imgenable == 1)
        count <= count + 1;
end
endmodule

//this internal module decides where the picture will be located. It outputs an initial location //and a height and width of the image, these are sent to the image counter.

module imgrectangle (x, y, imgenable);
input [9:0] x, y;
output imgenable;
reg imgenable;

parameter init_x = 10'd397;
parameter init_y = 10'd140;

parameter width = 10'd146;
parameter height = 10'd189;

always@(x or y)
begin
    if ((x >= init_x && x < (init_x + width))
        &&
        (y >= init_y && y < (init_y + height)))
        imgenable = 1;
    else
        imgenable = 0;
end
endmodule

//this is the third internal module. It loops between the four images to simulate a dancing beaver on the //screen. The beaver changes images every 8th note of the song.

module image_loop (pixel_clock, reset_sync, frame_sync, bitmap_rgb, bitmapb, bitmape, bitmapf);
input pixel_clock, frame_sync, reset_sync;
input bitmapb, bitmape, bitmapf;
output [23:0] bitmap_rgb;
reg [23:0] bitmap_rgb;
reg [7:0] count;
parameter maxcount = 186;
wire [23:0] rgb_outb2 = {24{bitmapb}};
wire [23:0] rgb_outb3 = {24{bitmapc}};
wire [23:0] rgb_outb5 = {24{bitmape}};
wire [23:0] rgb_outb6 = {24{bitmapf}};
assign b = (((count < 8'd31) && (count >= 8'd0)) || (count == maxcount));
assign c = (((count < 8'd62) && (count >= 8'd31)) || (count < 8'd186) && (count >= 8'd155));
assign e = (((count < 8'd93) && (count >= 8'd62)) || (count < 8'd155) && (count >= 8'd124));
assign f = ((count < 8'd124) && (count >= 8'd93));

always @ (posedge pixel_clock)
  begin
    if (reset_sync)
      count <= 8'd0;
    if (frame_sync)
      begin
        if(count == maxcount)
          begin
            count <= 8'd0;
          end
        else
          begin
            count <= count + 1;
          end
      end
    if (b)
      begin
        bitmap_rgb <= rgb_outb2;
      end
    if (c)
      begin
        bitmap_rgb <= rgb_outb3;
      end
    if (e)
      begin
        bitmap_rgb <= rgb_outb5;
      end
    if (f)
      begin
        bitmap_rgb <= rgb_outb6;
      end
  end
endmodule
IR Debounce
	timescale 1ns / 1ps
	//Sharmeen Browarek
	//Anna Ayuso
	//6.111 Final Project
	//TA: Jae Lee
	//
	////////////////////////////////////////////////////////////////////////////////
	//This module uses logic from the debounce and divider modules to simulate a high pass filter.
	//It’s used to get rid of the continuous pulses sent from the infrared signals, to avoid any glitches
	//during the game.
	module IRdebounce (reset, clock, noisy, extraclean);
	rainput reset, clock, noisy;
output extraclean;

parameter delay = 6100000;
parameter sensor_high = 6099990;

reg [24:0] delay_count;
reg [24:0] sensor_count;
reg extraclean;

always@(posedge clock)
begin
if (reset == 1)
begin
  delay_count <= 25'd0;
sensor_count <= 25'd0;
end

if (delay_count == delay)
begin
  delay_count <= 25'd0;
  if (sensor_count >= sensor_high)
  begin
    extraclean <= 1;
sensor_count <= 25'd0;
  end
  else
  begin
    extraclean <= 0;
sensor_count <= 25'd0;
  end
end

else
begin
  delay_count <= delay_count + 1;
end
if (noisy)
    sensor_count <= sensor_count + 1;
end
endmodule
Menu FSM
//Anna Ayuso
//Menu Minor FSM
//this module will allow the user to toggle between different
//difficulty levels on the menu screen

module Menu_FSM(pixel_clock, reset_sync, enable, up, down, select, busy_m, busy_g, difficulty);

input pixel_clock, reset_sync, enable, up, down, select, busy_m, busy_g, difficulty;
output busy_m, busy_g;
output [1:0] difficulty;

reg busy_m, busy_g;
reg [1:0] difficulty;
reg [1:0] state, nextstate;

reg t_busy_m, t_busy_g;
reg [1:0] t_difficulty;
reg up_temp, down_temp;

parameter easy = 0;
parameter med = 1;
parameter hard = 2;
parameter goto_game = 3;

always @ (posedge pixel_clock)
begin
if (reset_sync)
begin
state <= easy; //initial state is easy
busy_m <= 1;
baby_g <= 0;
difficulty <= 2'd0;
up_temp <= up;
down_temp <= down;
end
else if (enable) //states transition on an enable signal that pulses
begin
//every half second
state <= nextstate;
baby_m <= t_busy_m;
baby_g <= t_busy_g;
up_temp <= up;
down_temp <= down;
if (t_difficulty != 2'b11) //difficulty is 3 after the user has

difficulty <= t_difficulty; //selected a level, so difficulty
else
//is held constant after it is

difficulty <= difficulty; //selected
end

end

always @ (state)

79
begin
  t_busy_m = 1;
  t_busy_g = 0;
  t_difficulty = 2'b11;

case(state)
  easy: begin
    t_difficulty = 2'd0;
    if (select)
      begin
        nextstate = goto_game;
        end
    else if (up_temp)
      begin
        nextstate = hard;
        end
    else if (down_temp)
      begin
        nextstate = med;
        end
    else
      begin
        nextstate = easy;
        end
  end

  med: begin
    t_difficulty = 2'd1;
    if (select)
      begin
        nextstate = goto_game;
        end
    else if (up_temp)
      begin
        nextstate = easy;
        end
    else if (down_temp)
      begin
        nextstate = hard;
        end
    else
      nextstate = med;
  end

  hard: begin
    t_difficulty = 2'd2;
    if (select)
      begin
        nextstate = goto_game;
        end
    else if (up_temp)
      begin
        nextstate = med;
        end
    else if (down_temp)
      begin
        nextstate = med;
        end
end
begin
  nextstate = easy;
end
else
  nextstate = hard;
end

goto_game:
  begin  // this state is reached after user
    t_busy_m = 0;  // makes a selection, difficulty from
    t_busy_g = 1;  // the previous state is held constant
    nextstate = goto_game;  // fsm forever loops in this state until a reset
  end

default: nextstate = easy;
endcase
end

endmodule
module piblock(pixel_clock, reset_sync, piblock_y, frame_sync, ready, ystart, start_game);

input pixel_clock, reset_sync, frame_sync, start_game;
input [9:0] ystart;
output [9:0] piblock_y;
output ready;
reg [9:0] piblock_y;
parameter ystop = 9;
assign ready = (piblock_y == 10'd500);
always @ (posedge pixel_clock)
begin
    if (reset_sync || start_game)
        begin
            piblock_y <= ystart;
        end
    else if (frame_sync)
        begin
            if (piblock_y > ystop)
                begin
                    piblock_y <= piblock_y - 2;
                end
            else
                begin
                    piblock_y <= 10'd606;
                end
        end
end
endmodule
Report FSM
//Anna Ayuso
//report module
//this module converts a score into a letter score (A, B, C, F) for the game
//to gage the user's performance

module Report_FSM(pixel_clock, reset_sync, frame_sync, score,
                   difficulty, busy_r, letter_score, song_rows, stop, fail);

input pixel_clock, reset_sync, frame_sync, stop, fail;
input [10:0] score;
input [1:0] difficulty;
input [9:0] song_rows;

output busy_r;
output [1:0] letter_score;

reg busy_r;
reg [1:0] letter_score;
reg [5:0] buffer1, buffer2, buffer3;

parameter easy = 0;
parameter med = 1;
parameter hard = 2;

always @(posedge pixel_clock)
begin

  if (reset_sync)
    begin
      busy_r <= 1;
      letter_score <= 2’d0;
      buffer1 <= 10;
      buffer2 <= 10;
      buffer3 <= 7;
    end

  else if (stop)
    begin
      letter_score <= letter_score;
    end

  else if (frame_sync) //buffers are set that determine
    begin //how many arrows a user can have missed
      busy_r <= 1; //as difficulty increases, the buffer
                   //decreases
      if (difficulty == 0) //easy
        begin
          buffer1 <= 10;
          buffer2 <= 10;
          buffer3 <= 7;
        end
      else if (difficulty == 1) //medium
        begin
          buffer1 <= 10;
          buffer2 <= 10;
          buffer3 <= 7;
        end
    end

end

buffer1 <= 7;
buffer2 <= 7;
buffer3 <= 5;
end
else if (difficulty == 2)  //hard
begin
buffer1 <= 5;
buffer2 <= 5;
buffer3 <= 2;
end
else
begin
buffer1 <= buffer1;
buffer2 <= buffer2;
buffer3 <= buffer3;
end
if (fail)
  letter_score <= 2'd3;  //letter scores are set
else if (song_rows <= 10)  //based on score ranges
  letter_score <= 2'd0;
else if (score >= (((song_rows - 15) * 3) - buffer1))
  letter_score <= 2'd0;
else if (score >= (((song_rows - 8) * 2) - buffer2))
  letter_score <= 2'd1;
else if (score > ((song_rows - 8) - buffer3))
  letter_score <= 2'd2;
else
  letter_score <= 2'd3;
end
end
endmodule
Score Keeper

//Anna Ayuso
//score keeper
//this module continually computes the score given an accuracy signal from
//the video component and the user inputs from the IR sensors

module ScoreKeeper(pixel_clock, reset_sync, frame_sync, sensors, accuracy, score, stop);

input pixel_clock, reset_sync, frame_sync, stop;
input [5:0] accuracy;  //the first two bits of the accuracy signal correspond to
input [5:0] sensors;   //an accuracy of either 0, 1, 2, or 3
                        //the next 4 bits correspond to what arrows
                        //the accuracy refers to, the first two bits
                        //accuracy looks like this 0012310001231000...
output [10:0] score;
reg [10:0] score;
reg inc;

always @ (posedge pixel_clock)
begin
    if (reset_sync)
        begin
            score <= 11'd0;
            inc <= 0;
        end
    else if (stop)
        begin
            score <= score;
        end
    else if (frame_sync)
        begin
            if (accuracy[5:4] == 2'd0)  //when accuracy indicator is set to 0,
                inc <= 0;           //inc flag is set to 0 to allow scoring again
            else if (inc != 1 && accuracy[5:4] != 2'd0  && sensors[3:0] != 4'd0)
                begin
                        (sensors[0] == 1  && accuracy[0] == 1)))
                        //if the score hasn't been incremented yet, the accuracy indicator
                        //is nonzero, and there is input from the user...
                        begin
                        (sensors[0] == 1  && accuracy[0] == 1)))
                            //if the accuracy indicator is 1, and any of the user inputs and accuracy arrows
                            //match, increment score, and set inc flag to 1 to indicate that the score
                            //has already been incremented
                            begin
                                score <= score + 1;
                                inc <= 1;
                            end
                        end
                    end
                end
            else
                begin
                        (sensors[0] == 1  && accuracy[0] == 1)))
                        //if the accuracy indicator is 1, and any of the user inputs and accuracy arrows
                        //match, increment score, and set inc flag to 1 to indicate that the score
                        //has already been incremented
                        begin
                            score <= score + 1;
                            inc <= 1;
                        end
                    else
                        begin
                            score <= score + 1;
                            inc <= 1;
                        end
                end
        end
end

   (sensors[0] == 1 && accuracy[0] == 1)))
   // if the accuracy indicator is 2, and any of the user inputs and accuracy arrows
   // match, increment score by 2, and set inc flag to 1 to indicate that the score
   // has already been incremented
   begin
   score <= score + 2;
   inc <= 1;
   end
   (sensors[0] == 1 && accuracy[0] == 1)))
   // if the accuracy indicator is 3, and any of the user inputs and accuracy arrows
   // match, increment score by 3, and set inc flag to 1 to indicate that the score
   // has already been incremented
   begin
   score <= score + 3;
   inc <= 1;
   end
else
   score <= score;
end
else score <= score;
end
endmodule
module VGA (pixel_clock, reset_sync, hsync, vsync, sync_b, blank_b, pixel_count, line_count);

input pixel_clock; // 25.175 MHz pixel clock
input reset_sync; // system reset
output hsync; // horizontal sync
output vsync; // vertical sync
output sync_b; // hardwired to Vdd
output blank_b; // composite blank
output [9:0] pixel_count; // number of the current pixel
output [9:0] line_count; // number of the current line

// 640x480 75Hz parameters

parameter PIXELS = 800;
parameter LINES = 525;
parameter HACTIVE_VIDEO = 640;
parameter HFRONT_PORCH = 16;
parameter HSYNC_PERIOD = 96;
parameter HBACK_PORCH = 48;
parameter VACTIVE_VIDEO = 480;
parameter VFRONT_PORCH = 11;
parameter VSYNC_PERIOD = 2;
parameter VBACK_PORCH = 28;

parameter PIXELS = 832;
parameter LINES = 520;
parameter HACTIVE_VIDEO = 640;
parameter HFRONT_PORCH = 24;
parameter HSYNC_PERIOD = 40;
parameter HBACK_PORCH = 128;
parameter VACTIVE_VIDEO = 480;
parameter VFRONT_PORCH = 9;
parameter VSYNC_PERIOD = 3;
parameter VBACK_PORCH = 28;

// current pixel count
reg [9:0] pixel_count = 10'b0;
reg [9:0] line_count = 10'b0;

// registered outputs
reg hsync = 1'b1;
reg vsync = 1'b1;
reg blank_b = 1'b1;

wire sync_b; // connected to Vdd
wire pixel_clock;
wire [9:0] next_pixel_count;
wire [9:0] next_line_count;

always @(posedge pixel_clock)
begin
    if (reset_sync)
        begin
            pixel_count <= 10'b0;
            line_count <= 10'b0;
            hsync <= 1'b1;
            vsync <= 1'b1;
            blank_b <= 1'b1;
        end
    else
        begin
            pixel_count <= next_pixel_count;
            line_count <= next_line_count;
            hsync <=
                (next_pixel_count < HACTIVE_VIDEO + HFRONT_PORCH) |
                (next_pixel_count >= HACTIVE_VIDEO + HFRONT_PORCH + HSYNC_PERIOD);
            vsync <=
                (next_line_count < VACTIVE_VIDEO + VFRONT_PORCH) |
                (next_line_count >= VACTIVE_VIDEO + VFRONT_PORCH + VSYNC_PERIOD);
            blank_b <=
                (next_pixel_count < HACTIVE_VIDEO) &
                (next_line_count < VACTIVE_VIDEO);
        end
end

// next state is computed with combinational logic

assign next_pixel_count = (pixel_count == PIXELS-1) ? 10'h000 : pixel_count + 1'b1;

assign next_line_count = (pixel_count == PIXELS-1) ?
    (line_count == LINES-1) ? 10'h000 :
    line_count + 1'b1 : line_count;
// since we are providing hsync and vsync to the display, we
// can hardwire composite sync to Vdd.

    assign sync_b = 1'b1;

endmodule
Verilog Testbench Code

Test Control

`timescale 1ns / 1ps

`timescale 1ns / 1ps

// Anna Ayuso
// created to test the control unit
/>.ensures that when select goes high, the major fsm transitions to the game state
/>.verifies that in the game state, arrows are being outputted
/>.when the score is too low, the major fsm transitions to the final report state

module test_control_v;

    // Inputs
    reg pixel_clock;
    reg frame_sync;
    reg [5:0] accuracy;
    reg [5:0] sensors;
    reg ready;

    // Outputs
    wire [1:0] letter_score;
    wire [3:0] arrows;
    wire start_song;
    wire [1:0] difficulty;
    wire [1:0] mode;
    wire m_busy_g;
    wire [9:0] beat_count;
    wire [10:0] score;
    wire stop_song;

    // Instantiate the Unit Under Test (UUT)
    Control_Unit uut (  
        .pixel_clock(pixel_clock),
        .frame_sync(frame_sync),
        .accuracy(accuracy),
        .sensors(sensors),
        .ready(ready),
        .letter_score(letter_score),
        .arrows(arrows),
        .start_song(start_song),
        .difficulty(difficulty),
        .mode(mode),
        .m_busy_g(m_busy_g),
        .beat_count(beat_count),
        .score(score),
        .stop_song(stop_song)
    );

always #5 pixel_clock = ~pixel_clock;
initial begin
    // Initialize Inputs
    pixel_clock = 0;
frame_sync = 1;
accuracy = 0;
sensors = 0;
ready = 1;

// Wait 100 ns for global reset to finish
#10;
sensors[5] = 1;

#10;
sensors[5] = 0;

#10;
sensors[4] = 1;

// Add stimulus here
end
endmodule
Test Game

`timescale 1ns / 1ps

/*******************************************************************************/
// Anna Ayuso
// created to test the game fsm
// -tested with the score hardcoded to 10
// -arrows should be changing while in the move arrows state
// */
/**********************************************************************************/

module test_game_v;

    // Inputs
    reg pixel_clock;
    reg reset_sync;
    reg frame_sync;
    reg start_game;
    reg [1:0] difficulty;
    reg [10:0] score;
    reg ready;

    // Outputs
    wire busy_r;
    wire busy_g;
    wire start_song;
    wire [9:0] beat_count;
    wire [3:0] arrows;
    wire [9:0] song_rows;
    wire stop_song;
    wire [1:0] state;

    // Instantiate the Unit Under Test (UUT)
Game_FSM uut (    .pixel_clock(pixel_clock),
        .reset_sync(reset_sync),
        .frame_sync(frame_sync),
        .start_game(start_game),
        .difficulty(difficulty),
        .score(score),
        .ready(ready),
        .busy_r(busy_r),
        .busy_g(busy_g),
        .start_song(start_song),
        .beat_count(beat_count),
        .arrows(arrows),
        .song_rows(song_rows),
        .stop_song(stop_song),
        .state(state)
    );

    always #5 pixel_clock = ~pixel_clock;
    initial begin
        // Initialize Inputs
        pixel_clock = 0;
    end

92
reset_sync = 0;
frame_sync = 1;
start_game = 1;
difficulty = 0;
score = 10;
ready = 1;

// Wait 100 ns for global reset to finish
#10;
reset_sync = 1;

#10;
reset_sync = 0;

// Add stimulus here

end

dendmodule
Test Menu

`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Anna Ayuso
// created to test the menu fsm
// - up and down signals are altered to make sure
// the fsm changes states correctly, difficulty should be changing accordingly
// - when select goes high, difficulty should remain the same and the busy signal
// should go low
//////////////////////////////////////////////////////////////////////////////////

module test_menu_v;

    // Inputs
    reg pixel_clock;
    reg reset_sync;
    reg frame_sync;
    reg up;
    reg down;
    reg select;

    // Outputs
    wire busy_m;
    wire busy_g;
    wire [1:0] difficulty, state;

    // Instantiate the Unit Under Test (UUT)
    Menu_FSM uut (
        .pixel_clock(pixel_clock),
        .reset_sync(reset_sync),
        .frame_sync(frame_sync),
        .up(up),
        .down(down),
        .select(select),
        .busy_m(busy_m),
        .busy_g(busy_g),
        .difficulty(difficulty),
        .state(state)
    );

    always #5 pixel_clock = ~pixel_clock;
    initial begin
        // Initialize Inputs
        pixel_clock = 0;
        reset_sync = 0;
        frame_sync = 1;
        up = 0;
        down = 0;
        select = 0;

        // Wait 100 ns for global reset to finish
reset_sync = 1;
#10
reset_sync = 0;
#10
up = 1;
#100
up = 0;
#100
down = 1;
#100
select = 1;

// Add stimulus here

end

dendumodule
Test Report

`timescale 1ns / 1ps

// Anna Ayuso
// created to test the report module
// - ensures that the proper letter score is outputted
// when the score in a certain range
//

module test_report_v;

// Inputs
reg pixel_clock;
reg reset_sync;
reg frame_sync;
reg [10:0] score;
reg [1:0] difficulty;
reg [9:0] song_rows;
reg stop;

// Outputs
wire busy_r;
wire [1:0] letter_score;

// Instantiate the Unit Under Test (UUT)
Report_FSM uut (  
  .pixel_clock(pixel_clock),
  .reset_sync(reset_sync),
  .frame_sync(frame_sync),
  .score(score),
  .difficulty(difficulty),
  .busy_r(busy_r),
  .letter_score(letter_score),
  .song_rows(song_rows),
  .stop(stop)
);

always #5 pixel_clock = ~pixel_clock;
initial begin
  // Initialize Inputs
  pixel_clock = 0;
  reset_sync = 0;
  frame_sync = 1;
  score = 0;
  difficulty = 0;
  song_rows = 20;
  stop = 0;

  // Wait 100 ns for global reset to finish
  #10;
  reset_sync = 1;
reset_sync = 0;

score = 1;

score = 10;

difficulty = 1;
score = 1;

score = 10;

difficulty = 2;
score = 1;

score = 5;

score = 10;

// Add stimulus here

end

endmodule
Test Score

`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Anna Ayuso
// created to test the score keeper
// - should increment the score when there is a match in the arrows and the accuracy signal
// - score shouldn't increment again until the accuracy signal goes back to 0
// - more points should be given when the accuracy is higher
//////////////////////////////////////////////////////////////////////////////////

module test_score_v;

    // Inputs
    reg pixel_clock;
    reg reset_sync;
    reg frame_sync;
    reg [5:0] sensors;
    reg [5:0] accuracy;

    // Outputs
    wire [10:0] score;

    // Instantiate the Unit Under Test (UUT)
    ScoreKeeper uut (
        .pixel_clock(pixel_clock),
        .reset_sync(reset_sync),
        .frame_sync(frame_sync),
        .sensors(sensors),
        .accuracy(accuracy),
        .score(score)
    );

    always #5 pixel_clock = ~pixel_clock;
    initial begin
        // Initialize Inputs
        pixel_clock = 0;
        reset_sync = 0;
        frame_sync = 1;
        sensors = 4'b1001;
        accuracy = 0;

        // Wait 100 ns for global reset to finish
        #10;
        reset_sync = 1;

        #10
        reset_sync = 0;

        #10
        accuracy = 6'b010000;

        #10
    end

endmodule
accuracy = 6'b011001;

#10
accuracy = 6'b001001;

#10
accuracy = 6'b001001;

accuracy = 6'b111001;

#10
accuracy = 6'b011001;

#10
accuracy = 6'b011001;

#10
accuracy = 6'b000000;

#10
accuracy = 6'b010000;

#10
accuracy = 6'b011001;

// Add stimulus here

end

endmodule
Other Code (Audio Matlab script, Audio Java script, Image Matlab conversion script)

Java script for Audio

//Anna Ayuso
//this java code takes in csv files
//and downsamples the song by 10
//the code then writes the downsampled
//song to a coe file to be loaded onto
//the ROM through verilog

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.FilterWriter;
import com.Ostermiller.util.*;

public class downsample {

    public static void main(String args[]) {
        BufferedReader in;
        String[][] values = {};
        String[] downsampled = new String[100000];

        try {
            in = new BufferedReader(new FileReader("downsamplefile2"));
            values = CSVParser.parse(in);
        }
        catch (Exception e) {
        }

        int a = values[0].length;

        String[] downsampled = new String[100000];

        try {
            in = new BufferedReader(new FileReader("downsamplefile2"));
            values = CSVParser.parse(in);
        }
        catch (Exception e) {
        }

        int a = values[0].length;

        // System.out.println(samp);
        int temp = 0;
        long avg;
        for (int i = 0; i < a; i++) {
            // System.out.println("i = " + i);
            // System.out.println(values[0][i]);
            if (i%10 == 0 & i != 10) {
                avg = (temp + Integer.parseInt(values[0][i])/10;
            }
        }
    }
}
```java
// System.out.println("TEMP " + temp);
// System.out.println("AVG = " + avg);
String dec_st = "";
if (avg < 0) {} // converts negative numbers to binary, and then converts back
    String entry = convert2Bin(avg);
    int dec = convert2Dec(entry);
    dec_st = Integer.toString(dec);
else {
    dec_st = Long.toString(avg);
}

downsamp[downsampled[(i/10) - 1] = dec_st /*Long.toString(avg)*/ + ",";
// downsampled[(i/6) - 1] = avg;
temp = 0;
}
else {
temp = temp + Integer.parseInt(values[0][i]);
}
}

try {
    CSVPrinter csvp = new CSVPrinter(new BufferedWriter(new FileWriter("ds10_2")));
    for (int i = 0; i < downsampled.length; i++) {
        csvp.writeln(downsampled[i]);
    }
}

} catch (Exception e) {};

}

public static String convert2Bin(long l) {
    Long lg = new Long(l);
    Integer i = lg.intValue();
    String binstr = Integer.toBinaryString(i);
    int length = binstr.length();
    if (length > 8) {
```
```java
int a = length - 8;
binstr = binstr.substring(a);
}
else {
    int a = 8 - length;
    for (int j = 0; j < a; j++) {
        binstr = "0" + binstr;
    }
}
return binstr;
}

public static int convert2Dec(String s) {
    int dec = 0;
    if (s.charAt(0) == '1') {
        dec = dec + 128;
    }
    if (s.charAt(1) == '1') {
        dec = dec + 64;
    }
    if (s.charAt(2) == '1') {
        dec = dec + 32;
    }
    if (s.charAt(3) == '1') {
        dec = dec + 16;
    }
    if (s.charAt(4) == '1') {
        dec = dec + 8;
    }
    if (s.charAt(5) == '1') {
        dec = dec + 4;
    }
    if (s.charAt(6) == '1') {
        dec = dec + 2;
    }
    if (s.charAt(7) == '1') {
        dec = dec + 1;
    }
    return dec;
}

Matlab – Audio conversion script

%Ayugu
%matlab code used to convert a music sample in wav form
%to a csv file that can be processed with java code
```
[y,Fs,bits] = wavread('MrRoboto5', [1, 1000000]);
a = y(:,1);
a = floor(a * 128);
csvwrite('downsamplefile', a);

[y1,Fs1,bits1] = wavread('MrRoboto5', [1000001, 2000000]);
a1 = y1(:,1);
a1 = floor(a1 * 128);
csvwrite('downsamplefile1', a1);

[y2,Fs2,bits2] = wavread('MrRoboto5', [2000001, 2068992]);
a2 = y2(:,1);
a2 = floor(a2 * 128);
csvwrite('downsamplefile2', a2);

Matlab – Image conversion script

A = imread('beavers.bmp', 'bmp');

fid = fopen('beaver.coe', 'w');
fprintf(fid, 'memory_initialization_radix=16;
memory_initialization_vector=');
for m=1:384
  for n=1:512
    for v=1:3
      fprintf(fid, '%x', A(m,n,v)/16);
    end
    fprintf(fid, ',
');
  end
  fprintf(fid, 'n');
end
fclose(fid);