Appendix

Motion Code Tables

Table 1. Gesture code descriptions. Combinations of acceleration and gyroscope codes result in particular gesture codes specifying the audio effect to the audio system.

Gesture Code	Acceleration Code	Gyroscope Code	Description
0	0	0	Nothing
1	9, 10	0	Volume up
2	11, 12	0	Volume down
3	1, 2	0	Low-pass filter
4	3, 4	0	High-pass filter
5	0	1	Clip0
6	0	2	Clip1
7	0	3	Clip2
8	0	4	Clip3
9	0	5	Clip4
10	0	6	Clip5
11	0	7	Clip6
12	0	8	Clip7
13	5, 6, 7, 8	0	Increment play slot
14	13, 14, 15, 16	0	Nothing

Table 2. Acceleration code descriptions. The unit can infer 17 different combinations of translational motion from the two paddles.

Acceleration Code	Left Hand	Right Hand
0		
1	Left	
2	Right	
3		Left
4		Right
5	Left	Left
6	Right	Right
7	Left	Right
8	Right	Left
9	Up	
10	Down	
11		Up
12		Down
13	Up	Up
14	Down	Down
15	Up	Down
16	Down	Up

Table 3. Gyroscope code descriptions. The unit can infer 9 different combinations of rotational motion from the two paddles.

Gyroscope Code	Left Hand	Right Hand
0		
1	Clockwise	
2	Counterclockwise	
3		Clockwise
4		Counterclockwise
5	Clockwise	Clockwise
6	Counterclockwise	Counterclockwise
7	Counterclockwise	Clockwise
8	Clockwise	Counterclockwise

Verilog Code

```
`timescale 1ns / 1ps

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

//

// Acceleration Decoder

//

// This unit receives as input the average of the past 8 accelerations
along
// the x, y, and z axis for the right hand and the left hand.  It
decodes these

// acceleration values into an acceleration code corresponding to the
action

// that has been sensed.  The unit waits in the idle state until it
receives

// a request from the master FSM for an acceleration code upon which it
// performs
all the necessary calculations to arrive at an acceleration code.

//

// Code | Left  | Right

// -----|-------|------

// 0    | X     | X

// 1    | left  | X

// 2    | right | X
// 3    | X     | left

// 4    | X     | right

// 5    | left  | left

// 6    | right | right

// 7    | left  | right

// 8    | right | left

// 9    | up    | X

// 10   | down  | X

// 11   | X     | up

// 12   | X     | down

// 13   | up    | up

// 14   | down  | down

// 15   | up    | down

// 16   | down  | up

//

// In the table above, X indicates no movement.

//

////////////////////////////////////////////////////////////////////////
/////////
module accel_decoder(clock, reset_sync, left_accel_x, left_accel_y,
left_accel_z, right_accel_x, right_accel_y, right_accel_z, request,
busy, accel_code);

    input clock;

    input reset_sync;

    input [7:0] left_accel_x;

    input [7:0] left_accel_y;

    input [7:0] left_accel_z;

    input [7:0] right_accel_x;

    input [7:0] right_accel_y;

    input [7:0] right_accel_z;
    input request;

    output reg busy;

    output reg [4:0] accel_code;



    reg [1:0] state, next;



    //threshold values

    parameter min_accel_x = 4;

    parameter min_accel_y = 3;

    parameter zero_bias_x = 85;

    parameter zero_bias_y = 65;

    //states
    parameter IDLE = 0;
    parameter decode = 1;

//state assignment
always @ (posedge clock) begin
      if (reset_sync) state <= IDLE;
      else state <= next;
end

//next state computation
always @ (state or request or left_accel_x or left_accel_y or
left_accel_z or right_accel_x or right_accel_y or right_accel_z) begin


      busy = 0;

      case (state)
            IDLE: begin
                  if (request) next = decode;
                  else next = IDLE;


                  accel_code = 0;
            end


            decode: begin
                  next = IDLE;

                  busy = 1;

                  if ((right_accel_y > zero_bias_y+min_accel_y) &
(left_accel_y > zero_bias_y+min_accel_y)) // both hands moving up
                        accel_code = 13;
                  else if ((right_accel_y < zero_bias_y-min_accel_y) &
(left_accel_y < zero_bias_y-min_accel_y)) // both hands moving down
                        accel_code = 14;
                  else if ((right_accel_y > zero_bias_y+min_accel_y) &
(left_accel_y < zero_bias_y-min_accel_y)) // RH up, LH down
                        accel_code = 16;
                  else if ((right_accel_y < zero_bias_y-min_accel_y) &
(left_accel_y > zero_bias_y+min_accel_y)) // RH down, LH up
                        accel_code = 15;
                  else if (right_accel_y > zero_bias_y+min_accel_y) //
only RH up
                        accel_code = 11;
                  else if (right_accel_y < zero_bias_y-min_accel_y) //
only RH down
                        accel_code = 12;
                  else if (left_accel_y > zero_bias_y+min_accel_y) //
only LH up
                        accel_code = 9;
                  else if (left_accel_y < zero_bias_y-min_accel_y) //
only LH down
                        accel_code = 10;

                  else if ((right_accel_x > zero_bias_x+min_accel_x) &
(left_accel_x > zero_bias_x+min_accel_x)) // both hands right
                        accel_code = 6;
                  else if ((right_accel_x < zero_bias_x-min_accel_x) &
(left_accel_x < zero_bias_x-min_accel_x)) // both hands left
                        accel_code = 5;
                  else if ((right_accel_x > zero_bias_x+min_accel_x) &
(left_accel_x < zero_bias_x-min_accel_x)) // RH right, LH left
                        accel_code = 7;
                  else if ((right_accel_x < zero_bias_x-min_accel_x) &
(left_accel_x > zero_bias_x+min_accel_x)) // RH left, LH right
                        accel_code = 8;
                  else if (right_accel_x > zero_bias_x+min_accel_x) //
only RH right
                        accel_code = 4;
                  else if (right_accel_x < zero_bias_x-min_accel_x) //
only RH left
                        accel_code = 3;
                  else if (left_accel_x > zero_bias_x+min_accel_x) //
only LH right
                        accel_code = 2;
                  else if (left_accel_x < zero_bias_x-min_accel_x) //
only LH left
                        accel_code = 1;
            end

            default:

                  begin

                        next = IDLE;

                  end
```

```
      endcase
end

endmodule


`timescale 1ns / 1ps

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

// Accelerator Numbers Unit

//

// This unit instantiates and connects the accelerator processor and
the six

// BRAM units used to store samples from the accelerators.

//

//////////////////////////////////////////////////////////////////////
/////////
module accel_num(clock, reset_sync, enable, left_accel_x, left_accel_y,
left_accel_z,

             right_accel_x, right_accel_y, right_accel_z, ADbusy,
convst_b, rd_b,

             left_accel_x_out, left_accel_y_out, left_accel_z_out,
right_accel_x_out,

             right_accel_y_out, right_accel_z_out, left_val_x,
left_read_val_x, left_sum_x, state);

    input clock;

    input reset_sync;

    input enable;

    input [7:0] left_accel_x;

    input [7:0] left_accel_y;

    input [7:0] left_accel_z;

    input [7:0] right_accel_x;

    input [7:0] right_accel_y;

    input [7:0] right_accel_z;

    input ADbusy;
    output convst_b;

    output rd_b;

    output [7:0] left_accel_x_out;

    output [7:0] left_accel_y_out;

    output [7:0] left_accel_z_out;

    output [7:0] right_accel_x_out;

    output [7:0] right_accel_y_out;

    output [7:0] right_accel_z_out;

    output [7:0] left_val_x;

    output [7:0] left_read_val_x;

    output [15:0] left_sum_x;

    output [3:0] state;


//Connecting wires

wire [7:0] left_read_x, left_read_y, left_read_z;

wire [7:0] right_read_x, right_read_y, right_read_z;

wire we, reset_busy;

wire [7:0] addr;

wire [7:0] left_data_x, left_data_y, left_data_z;

wire [7:0] right_data_x, right_data_y, right_data_z;

wire [7:0] r_left_data_x, r_left_data_y, r_left_data_z;
wire [7:0] r_right_data_x, r_right_data_y, r_right_data_z;




//Instantiate acceleration processor

      accel_processor accel_processor (

            .clock(clock),
            .reset_sync(reset_sync),
            .enable(enable),
            .left_accel_x(left_accel_x),
            .left_accel_y(left_accel_y),
            .left_accel_z(left_accel_z),
            .right_accel_x(right_accel_x),
            .right_accel_y(right_accel_y),
            .right_accel_z(right_accel_z),
            .ADbusy(ADbusy),
            .left_read_x(left_read_x),
            .left_read_y(left_read_y),
            .left_read_z(left_read_z),
            .right_read_x(right_read_x),
            .right_read_y(right_read_y),
            .right_read_z(right_read_z),

            .convst_b(convst_b),
            .rd_b(rd_b),
            .addr(addr),
            .left_data_x(left_data_x),
            .left_data_y(left_data_y),
            .left_data_z(left_data_z),
            .right_data_x(right_data_x),
            .right_data_y(right_data_y),
            .right_data_z(right_data_z),
            .we(we),
            .left_accel_x_out(left_accel_x_out),
            .left_accel_y_out(left_accel_y_out),
            .left_accel_z_out(left_accel_z_out),
            .right_accel_x_out(right_accel_x_out),
            .right_accel_y_out(right_accel_y_out),
            .right_accel_z_out(right_accel_z_out),

            .state(state),

            .left_val_x(left_val_x),

            .left_read_val_x(left_read_val_x),

            .left_sum_x(left_sum_x)

      );


//Instantiate left_accel BRAMs

accel_mem left_x_mem (

      .addr(addr),

      .clk(clock),

      .din(left_data_x),

      .dout(left_read_x),

      .we(we)

);
```

```
accel_mem left_y_mem (
      .addr(addr),
      .clk(clock),
      .din(left_data_y),
      .dout(left_read_y),
      .we(we)
);



accel_mem left_z_mem (
      .addr(addr),
      .clk(clock),
      .din(left_data_z),
      .dout(left_read_z),
      .we(we)
);



//Instantiate right_accel BRAMs

accel_mem right_x_mem (
      .addr(addr),
      .clk(clock),
      .din(right_data_x),
      .dout(right_read_x),
      .we(we)
);



accel_mem right_y_mem (
      .addr(addr),
      .clk(clock),
      .din(right_data_y),
      .dout(right_read_y),
      .we(we)
);



accel_mem right_z_mem (
      .addr(addr),
      .clk(clock),
      .din(right_data_z),
      .dout(right_read_z),
      .we(we)
);



endmodule
```

```
`timescale 1ns / 1ps

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

// Acceleration Processor
//
// This unit interfaces with the A/D converter to receive digitized raw
data
// from the accelerometers.  It then keeps a moving average of the past
eight
// accelerations along each of the three axes for each hand.  This is
done
// with a set of six BRAMs and a 3-bit pointer that automatically rolls
over to
// zero when it has counted to 7.  The pointer thus continually
increments
// each time the BRAMs are used.  The pointer indicates the slot in
which
// data should be written.  The BRAMs are instantiated in read-before-
write
// mode, which means the output displayed is the value that was in the
slot
// before the new data was written.  This unit then keeps a value that
is the
// sum of all the data in the BRAMs.  This is accomplished by
subtracting
// the value that was in the slot, and adding the value currently being
written
// to memory.  When output is to be assigned, the average is calculated
// by right-shifting the sums.  On reset, the zero-bias value is
written
// to each memory slot.
//

////////////////////////////////////////////////////////////////////////
/////////
module accel_processor(clock, reset_sync, enable, left_accel_x,
left_accel_y, left_accel_z,

             right_accel_x, right_accel_y, right_accel_z, ADbusy,
left_read_x, left_read_y,

             left_read_z, right_read_x, right_read_y, right_read_z,
convst_b, rd_b, addr, left_data_x,

             left_data_y, left_data_z, right_data_x, right_data_y,
right_data_z, we,

             left_accel_x_out, left_accel_y_out, left_accel_z_out,
right_accel_x_out,

             right_accel_y_out, right_accel_z_out, state, left_val_x,
left_read_val_x, left_sum_x);

    input clock;
    input reset_sync;
    input enable;
    input [7:0] left_accel_x;
    input [7:0] left_accel_y;
    input [7:0] left_accel_z;
    input [7:0] right_accel_x;
    input [7:0] right_accel_y;
    input [7:0] right_accel_z;
       input ADbusy;
       input [7:0] left_read_x;
       input [7:0] left_read_y;
       input [7:0] left_read_z;
       input [7:0] right_read_x;
       input [7:0] right_read_y;
       input [7:0] right_read_z;
       output reg convst_b;
       output reg rd_b;
       output reg [7:0] addr;
       output reg [7:0] left_data_x;
       output reg [7:0] left_data_y;
       output reg [7:0] left_data_z;
       output reg [7:0] right_data_x;
       output reg [7:0] right_data_y;
       output reg [7:0] right_data_z;
       output reg we;
    output reg [7:0] left_accel_x_out;
    output reg [7:0] left_accel_y_out;
    output reg [7:0] left_accel_z_out;
    output reg [7:0] right_accel_x_out;

    output reg [7:0] right_accel_y_out;

    output reg [7:0] right_accel_z_out;


       output reg [3:0] state;

       reg [3:0] next;

       reg [2:0] pointer = 0;

       output reg [7:0] left_val_x = 0;

       reg [7:0] left_val_y = 0;

       reg [7:0] left_val_z = 0;

       reg [7:0] right_val_x = 0;

       reg [7:0] right_val_y = 0;

       reg [7:0] right_val_z = 0;

       output reg [7:0] left_read_val_x = 0;

       reg [7:0] left_read_val_y = 0;

       reg [7:0] left_read_val_z = 0;

       reg [7:0] right_read_val_x = 0;

       reg [7:0] right_read_val_y = 0;

       reg [7:0] right_read_val_z = 0;

       output reg [15:0] left_sum_x = 640;

       reg [15:0] left_sum_y = 504;

       reg [15:0] left_sum_z = 0;

       reg [15:0] right_sum_x = 640;

       reg [15:0] right_sum_y = 504;

       reg [15:0] right_sum_z = 0;

       reg increment;

       reg [3:0]count = 0;

       reg count_enable;
```

```
       reg count_reset;

       reg sum;

       reg sum_reset;

       reg cap_val;

       reg cap_read;

       reg set_data;

       reg set_addr;

       reg set_out;

       reg reset_data;

       //states
       parameter IDLE = 0;
       parameter ADActivate = 1;

       parameter Pause = 2;

       parameter ADRead = 3;

       parameter MemActivate = 4;

       parameter Math = 5;

       parameter Output = 6;

       parameter Reset = 7;

       parameter ADRead2 = 8;

       parameter ADRead1 = 9;



       parameter count_goal = 7;


//state assignment
always @ (posedge clock) begin
      if (reset_sync) begin

            state <= Reset;

            pointer <= 0;

            count <= 0;

      end else begin
```

```
            state <= next;



            // incrementing the pointer
            if(increment) pointer <= pointer + 1;

            else pointer <= pointer;



            // increment the counter for writing to memory on reset
            if(count_enable) count <= count + 1;

            else count <= count;



            // set all sums to default values on reset
            if(sum_reset) begin

                  left_sum_x <= 640;
                  left_sum_y <= 504;
                  left_sum_z <= 0;
                  right_sum_x <= 640;
                  right_sum_y <= 504;
                  right_sum_z <= 0;

            end else begin

                  if(sum) begin // modify sums as apppropriate
                        left_sum_x <= left_sum_x + left_val_x -
left_read_val_x;
                        left_sum_y <= left_sum_y + left_val_y -
left_read_val_y;
                        left_sum_z <= left_sum_z + left_val_z -
left_read_val_z;
                        right_sum_x <= right_sum_x + right_val_x -
right_read_val_x;
                        right_sum_y <= right_sum_y + right_val_y -
right_read_val_y;
                        right_sum_z <= right_sum_z + right_val_z -
right_read_val_z;
                  end else begin
                        left_sum_x <= left_sum_x;
                        left_sum_y <= left_sum_y;
                        left_sum_z <= left_sum_z;
                        right_sum_x <= right_sum_x;
                        right_sum_y <= right_sum_y;
                        right_sum_z <= right_sum_z;
                  end

            end
            if(cap_val) begin
// capture input from the A/D converter
                  left_val_x <= left_accel_x;
                  left_val_y <= left_accel_y;
                  left_val_z <= left_accel_z;
                  right_val_x <= right_accel_x;
                  right_val_y <= right_accel_y;
                  right_val_z <= right_accel_z;

            end



            if(cap_read) begin
// read value that was previously in memory
                  left_read_val_x <= left_read_x;
                  left_read_val_y <= left_read_y;
                  left_read_val_z <= left_read_z;
                  right_read_val_x <= right_read_x;
                  right_read_val_y <= right_read_y;
                  right_read_val_z <= right_read_z;

            end



            if(reset_data) begin // set data to zero-bias values on
reset
                  addr[3:0] <= count;
                  addr[7:4] <= 0;
                  left_data_x <= 80;
                  left_data_y <= 63;
                  left_data_z <= 0;
                  right_data_x <= 80;
                  right_data_y <= 63;
                  right_data_z <= 0;
            end else if(set_data) begin
// set data to value read from A/D
                        left_data_x <= left_val_x;
                        left_data_y <= left_val_y;
                        left_data_z <= left_val_z;
                        right_data_x <= right_val_x;
                        right_data_y <= right_val_y;
                        right_data_z <= right_val_z;

            end



            if(set_addr) begin
// set the address to be written to
                  addr[2:0] <= pointer;

                  addr[7:3] <= 0;

            end
            if(set_out) begin
// set the output by right-shifting the sums
                  left_accel_x_out <= left_sum_x>>>3;
                  left_accel_y_out <= left_sum_y>>>3;
                  left_accel_z_out <= left_sum_z>>>3;
                  right_accel_x_out <= right_sum_x>>>3;
                  right_accel_y_out <= right_sum_y>>>3;
                  right_accel_z_out <= right_sum_z>>>3;

            end

      end
end

//next state computation
always @ (state or enable or ADbusy or pointer or count) begin

      we = 0;

      convst_b = 1;

      rd_b = 1;

      increment = 0;

      count_enable = 0;

      count_reset = 0;

      sum = 0;

      sum_reset = 0;

      cap_val = 0;

      cap_read = 0;

      set_data = 0;

      set_addr = 0;

      set_out = 0;

      reset_data = 0;

      case (state)
            IDLE:
                  begin
                        if (enable) next = ADActivate;
                        else next = IDLE;



                        count_reset = 1;
                  end
            ADActivate:
                  begin
                        next = Pause;


                        convst_b = 0; // indicate start conversion
                  end



            Pause:

                  begin

                        if (!ADbusy) next = ADRead;
// wait for conversion
                        else next = Pause;

                  end



            ADRead:

                  begin

                        next = ADRead1;



                        rd_b = 0;
// indicate start read
                        increment = 1;

                  end



            ADRead1:

                  begin

                        next = ADRead2;



                        rd_b = 0;

                        cap_val = 1;

                  end



            ADRead2:
                  begin

                        next = MemActivate;



                        set_data = 1;
// set address of memory and data to be written
                        set_addr = 1;

                  end



            MemActivate:

                  begin

                        next = Math;



                        we = 1;

                  end



            Math:

                  begin

                        next = Output;



                        cap_read = 1;
// capture previous value in memory slot
                  end



            Output:

                  begin

                        next = IDLE;



                        sum = 1;

                        set_out = 1;

                  end
            Reset:

           begin

               if (count > count_goal) begin

                   next = IDLE;

                   we = 1;

               end else begin

                                    next = Reset;


                                    count_enable = 1;

                  sum_reset = 1;

                  reset_data = 1;


                                    if(count > 0 ) begin

                       we = 1;

                                    end

               end

                        end

            default:
                  begin
                        next = IDLE;
                  end

      endcase
end


endmodule


`timescale 1ns / 1ps

///////////////////////////////////////////////////////////////////////
///////////
// Company: 6.111 Spring 2007
// Engineer: Karen L. Chu
//
// Create Date: 14:30:38 03/01/2007
// Project Name:
// Module Name: adc_divider
//
// Description: The divider module is used to divide the input clock
signal as desired.
// It takes as input a clock signal and a synchronized reset signal,
and outputs an enable
// signal that is some division of the clock.  In this project, the
divider is used to create
// a 100-Hz signal from a 27MHz clock.
//
// Dependencies: The clock is expected to be a 27MHz clock.
//
//////////////////////////////////////////////////////////////////////
///////////
module adc_divider(clock, reset_sync, enable);

    input clock;

       input reset_sync;
    output enable;

       reg enable;
       reg [24:0] count;

       //parameter countgoal = 25'd27000;
       parameter countgoal = 25'd4;

//On each clock, check for reset
//Then check to see if the goal has been reached
//If so, reset the count to 0 and restart counting
//Else, increment count
always @(posedge clock) begin
      if (reset_sync) begin
            count <= 25'd0;
      end else begin
            if (count == countgoal) begin
                  count <= 25'd0;
            end else begin
                  count <= count + 1;
            end
      end
end

//When count changes, check to see if goal has been reached
//If so, output enable high
//Else, make sure enable is low
always @ (count) begin
      if (reset_sync) begin
            enable = 1;
      end else if (count == countgoal) begin
            enable = 1;
      end else begin
            enable = 0;
      end
end
```

```
endmodule


`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////
///////////
// Company:
// Engineer: Jehan deFonseka
//
// Create Date:    20:03:25 04/25/2007
// Design Name:
// Module Name:    audioinout
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments: This module initializes and controls the ac97.
//   It also outputs the 36-bit audio input signal from the ac97, and
accepts a 36-bit input to pipe to the ac97.
//    It also controls the master volume of the ac97.
//
//////////////////////////////////////////////////////////////////////
///////////
module audioinout (clock, audio_reset_b, ac97_sdata_out, ac97_sdata_in,
                  ac97_synch, ac97_bit_clock, mutein, reset, leds,
state, bit_count, sound_done,
                        sound_out_r, sound_out_l,
                        sound_in_right, sound_in_left,
                        volume_enable_sync, audio_volume
                        );

      input mutein;
   input clock;
   output audio_reset_b;
   output ac97_sdata_out;
   input ac97_sdata_in;
   output ac97_synch;
   input ac97_bit_clock;
      input reset;

      input[1:0] volume_enable_sync;

      input[17:0] sound_out_r;
      input[17:0] sound_out_l;

      output[17:0] sound_in_right;
      output[17:0] sound_in_left;
      output sound_done;
      output[4:0] audio_volume;
      output[7:0] leds;
      output[3:0] state;
      output[7:0] bit_count;

      reg sound_done;

   reg audio_reset_b;
   reg ac97_sdata_out;
   reg ac97_synch;

   reg [7:0] bit_count;
   reg [4:0] frame_count;

   reg [23:0] command;
   wire [19:0] command_data;
   wire [19:0] command_address;

   reg [7:0] reset_count;

      reg[17:0] sound_in_right=0;
      reg[17:0] sound_out_right=0;
      reg[17:0] sound_in_left=0;
      reg[17:0] sound_out_left=0;

   reg[9:0] count;

      reg[3:0] state, next;
      reg[7:0] leds;

      reg muted;
      reg framecount;
      reg delay;

      reg[4:0] audio_volume;

      parameter RESET_STATE=0;
      parameter MASTER_VOLUME=1;
      parameter HEADPHONE_VOLUME=2;
      parameter LINE_IN=3;
      parameter MIC_INPUT=4;
      parameter RECORD_SELECT=5;
      parameter RECORD_GAIN=6;
      parameter PCM_OUT_VOLUME=7;
      parameter IDLE=8;

      // Separate the address and data portions of the command
   // and pad them to 20 bits
   assign command_address = {command[23:16], 12'h000};
   assign command_data = {command[15:0], 4'h0};

      initial begin
      reset_count = 0;
      // synthesis attribute init of reset_count is "00";
      audio_reset_b = 1'b0;
      // synthesis attribute init of audio_reset_b is "0";
            count=0;
            state=RESET_STATE;
```

```
            leds=8'b1111_1111;
   end

      //reset signal asserted before the bit clock is active
   always @(posedge clock)
      begin
     if (reset_count == 255)
       audio_reset_b <= 1;
     else
       reset_count = reset_count+1;
   end

   initial begin
      bit_count = 8'h00;
      // synthesis attribute init of bit_count is "00";
      frame_count = 4'h0;
      // synthesis attribute init of frame_count is "0";
            audio_volume=3;
   end

   always @(posedge ac97_bit_clock)
      begin
            if(reset)
            begin
                  state<=RESET_STATE;
                  audio_volume<=3;
            end
            else
            begin
                  // Generate the sync signal
                  if (bit_count == 255)
                        ac97_synch <= 1'b1;
                  if (bit_count == 15)
                        ac97_synch <= 1'b0;
                  if (bit_count == 53)
                  begin
                        sound_out_right<=sound_out_r;
                        sound_out_left<=sound_out_l;
                  end

//volume check
                  if(bit_count==3)
                  begin
                        if(volume_enable_sync==2'b01)
                        begin
                              if(audio_volume>0)
                              begin
                                    audio_volume<=audio_volume-1;
                                    state<=HEADPHONE_VOLUME;
                              end
                        end
                        else if(volume_enable_sync==2'b10)
                        begin
                              if(audio_volume<31)
                              begin
                                    audio_volume<=audio_volume+1;
```

```
                                   state<=HEADPHONE_VOLUME;

                             end
                       end
                 end


                 //pulse sound done every 2 frames (sample at 24000
Hz)
                 if(delay)
                 begin
                       if (bit_count == 95)
                       begin
                             sound_done<=1;
                             delay<=0;
                       end
                 end
                 else if(bit_count == 95)
                 begin
                       delay<=1;
                 end

                 if(bit_count == 97)
                 begin
                       sound_done<=0;
                 end

//ac97_sdata_in
                 if ((bit_count >= 57) && (bit_count <= 74))
                 begin
                 //Slot 3: PCM data left
                       sound_in_left[74-bit_count]<=ac97_sdata_in;
                 end
                 else if ((bit_count >= 77) && (bit_count <= 94))
                 begin
                 //slot 4: PCM data right
                       sound_in_right[94-bit_count]<=ac97_sdata_in;
                 end

//ac97_sdata_out
                 if ((bit_count >= 0) && (bit_count <= 15))
                 // Slot 0: Tags
                       case (bit_count[3:0])
                             4'h0: ac97_sdata_out <= 1; // Frame valid
                             4'h1: ac97_sdata_out <= 1; // Command
address valid
                             4'h2: ac97_sdata_out <= 1; // Command
data valid
                             4'h3: ac97_sdata_out <= 1; // left
                             4'h4: ac97_sdata_out <= 1; // right

                             default: ac97_sdata_out <= 1'b0;
                       endcase
                 else if ((bit_count >= 16) && (bit_count <= 35))
                 // Slot 1: Command address
                       ac97_sdata_out <= command_address[35-
bit_count];
```

```
                  else if ((bit_count >= 36) && (bit_count <= 55))
                  // Slot 2: Command data
                        ac97_sdata_out <= command_data[55-bit_count];

                  else if ((bit_count >= 56) && (bit_count <= 73))
                  begin
                  //Slot 3: PCM data left
                        ac97_sdata_out <= sound_out_left[73-bit_count];
                  end

                  else if ((bit_count >= 76) && (bit_count <= 93))
                  begin
                  //slot 4: PCM data right
                        ac97_sdata_out <= sound_out_right[93-
bit_count];
                  end
                  else if (bit_count == 255) //update frame info
                  begin
                        state<=next;
                  end
                  else
                        ac97_sdata_out<=1'b0;

                  bit_count <= bit_count+1;


        end
   end

      always @ (state)
      begin
            case (state)
                  RESET_STATE:
                  begin
                        next=MASTER_VOLUME;
                        leds=8'b1111_1110;
                  end
                  MASTER_VOLUME:
                  begin
                        command=24'h02_0000; // Unmute line outputs
                        next=LINE_IN;
                  end
                  LINE_IN:
                  begin
                        command = 24'h10_0808; // Unmute line inputs
                        next=MIC_INPUT;
                  end
                  MIC_INPUT:
                  begin
                        if(mutein)
                        begin
                              command = 24'h0E_8008; // mute analog
loopback
                              muted=1;
                        end
                        else
                        begin
```

```
                              command = 24'h0E_0808; // unmute analog
loopback
                              muted=0;
                        end
                        next=RECORD_SELECT;
                  end
                  RECORD_SELECT:
                  begin
                        command = 24'h1A_0000;  // select mic as record
source
                        next=RECORD_GAIN;
                        leds=8'b1110_1111;
                  end
                  RECORD_GAIN:
                  begin
                        command = 24'h1C_0000;  // no record gain
                        next=PCM_OUT_VOLUME;
                        leds=8'b1101_1111;
                  end
                  PCM_OUT_VOLUME:
                  begin
                        command[23:16]= 8'h18;  // volume controller
                        command[15:13] = 3'b0;
                        command[12:8] = 5'b01000; //audio_volume[4:0];
                        command[7:5] = 3'b0;
                        command[4:0] = 5'b01000; // audio_volume[4:0];
                        next=HEADPHONE_VOLUME;
                        leds=8'b1011_1111;

                  end
                  HEADPHONE_VOLUME:
                  begin
                        command[23:16]= 8'h04;  // volume controller
                        command[15:13] = 3'b0;
                        command[12:8] = audio_volume;
//audio_volume[4:0];
                        command[7:5] = 3'b0;
                        command[4:0] = audio_volume; //
audio_volume[4:0];
                        next=IDLE;
                  end
                  IDLE:
                  begin
                        next=IDLE;
                        command = 24'hFC_0000; // Read vendor ID
                        leds=8'b0111_1111;
                  end
                  default:
                        command = 24'hFC_0000; // Read vendor ID
        endcase
      end

endmodule

`timescale 1ns / 1ps
/////////////////////////////////////////////////////////////////////
///////////
```

```
// Company: 6.111 Spring 2007
// Engineer: Karen L. Chu
//
// Create Date: 14:30:38 03/01/2007
// Project Name:
// Module Name: divider
//
// Description: The divider module is used to divide the input clock
signal as desired.
// It takes as input a clock signal and a synchronized reset signal,
and outputs an enable
// signal that is some division of the clock.  In this project, the
divider is used to create
// a 10-Hz signal from a 27MHz clock.
//
// Dependencies: The clock is expected to be a 27MHz clock.
//
//////////////////////////////////////////////////////////////////////
///////////
module fsm_divider(clock, reset_sync, enable);
       input clock;
    input reset_sync;
    output enable;

       reg enable;
       reg [24:0] count;

       parameter countgoal = 25'd270000;
       //parameter countgoal = 25'd10;

//On each clock, check for reset
//Then check to see if the goal has been reached
//If so, reset the count to 0 and restart counting
//Else, increment count
always @(posedge clock) begin
      if (reset_sync) begin
            count <= 25'd0;
      end else begin
            if (count == countgoal) begin
                  count <= 25'd0;
            end else begin
                  count <= count + 1;
            end
      end
end

//When count changes, check to see if goal has been reached
//If so, output enable high
//Else, make sure enable is low
always @ (count) begin
      if (reset_sync) begin
            enable = 1;
      end else if (count == countgoal) begin
            enable = 1;
      end else begin
            enable = 0;
      end
```

```
end

endmodule

`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////
///////////
// Company:
// Engineer:
//
// Create Date:    17:09:56 05/15/2007
// Design Name:
// Module Name:    filter_enable_sync
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments: This module synchronizes filter enable signals
from the buttons and the gesture codes.
//
////////////////////////////////////////////////////////////////////////
///////////
module filter_enable_sync(clk, reset, filter_enable, filter,
filter_enable_sync);

input clk;
input reset;
input[1:0] filter_enable;
input[1:0] filter;

output[1:0] filter_enable_sync;

reg[1:0] filter_enable_sync;

always @ (posedge clk)
begin
      filter_enable_sync<=0;
      if((filter_enable==2'b01) || (filter_enable==2'b10))
      begin
            filter_enable_sync<=filter_enable;
      end
      else if((filter==2'b01) || (filter==2'b10))
      begin
            filter_enable_sync<=filter;
      end
end

endmodule

`timescale 1ns / 1ps
```

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

// FSM Tester
//
// This unit instantiates the master FSM, the acceleration decoder, and
the
// gyro decoder, connects them, and a divider for testing purposes.

//

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

module fsm_tester(clock, reset_sync, left_accel_x, left_accel_y,
left_accel_z, right_accel_x, right_accel_y, right_accel_z,
right_rotate, left_rotate, gesture_code);

    input clock;

       input reset_sync;
    input [7:0] left_accel_x;
    input [7:0] left_accel_y;
    input [7:0] left_accel_z;
    input [7:0] right_accel_x;
    input [7:0] right_accel_y;
    input [7:0] right_accel_z;

       input [16:0] right_rotate;

       input [16:0] left_rotate;

       output [4:0] gesture_code;



       //Declare wires

       wire enable;

       wire accel_busy, gyro_busy;

       wire accel_request, gyro_request;

       wire [4:0] accel_code;

       wire [3:0] gyro_code;



// Instantiate the master fsm
      master_fsm master (
            .clock(clock),
            .reset_sync(reset_sync),
            .enable(enable),
            .accel_busy(accel_busy),
            .gyro_busy(gyro_busy),
            .accel_code(accel_code),
            .gyro_code(gyro_code),
            .gyro_request(gyro_request),
            .accel_request(accel_request),
            .gesture_code(gesture_code),

            .state(state),

            .int_accel_code(int_accel_code),

            .int_gyro_code(int_gyro_code)
      );



// Instantiate the rotational decoder
      gyro_decoder gyro_decoder (
            .clock(clock),
            .reset_sync(reset_sync),
            .left_rotate(left_rotate),
            .right_rotate(right_rotate),
            .request(gyro_request),
            .busy(gyro_busy),
            .gyro_code(gyro_code)
      );



// Instantiate the translational decoder
      accel_decoder accel_decoder (
            .clock(clock),
            .reset_sync(reset_sync),
            .left_accel_x(left_accel_x),
            .left_accel_y(left_accel_y),
            .left_accel_z(left_accel_z),
            .right_accel_x(right_accel_x),
            .right_accel_y(right_accel_y),
            .right_accel_z(right_accel_z),
            .request(accel_request),
            .busy(accel_busy),
            .accel_code(accel_code)
      );



// Instantiate the divider

      divider divider (

            .clock(clock),
            .reset_sync(reset_sync),
            .enable(enable)

      );
```

```
endmodule


`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////
///////////
// Company:
// Engineer:
//
// Create Date:    15:24:36 05/08/2007
// Design Name:
// Module Name:    gen_filter
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments: This module creates a 4-point z-transform
filter based on input coefficient parameters.
//
//////////////////////////////////////////////////////////////////////
///////////
module gen_filter(reset, audio_in, audio_out, clock, enable, advance,
enabled_reg, fil_done);
      input reset;
      input clock;
      input[35:0] audio_in;
      input enable;
      input advance;
      output[35:0] audio_out;
      output fil_done;
      output enabled_reg;

      reg fil_done;

      reg signed [17:0] delay_1B_l;
      reg signed [17:0] delay_1B_r;
      reg signed [17:0] delay_2B_l;
      reg signed [17:0] delay_2B_r;
      reg signed [17:0] delay_3B_l;
      reg signed [17:0] delay_3B_r;
      reg signed [17:0] delay_4B_l;
      reg signed [17:0] delay_4B_r;


      reg signed [17:0] delay_1A_l;
      reg signed [17:0] delay_1A_r;
      reg signed [17:0] delay_2A_l;
      reg signed [17:0] delay_2A_r;
      reg signed [17:0] delay_3A_l;
      reg signed [17:0] delay_3A_r;
      reg signed [17:0] delay_4A_l;
      reg signed [17:0] delay_4A_r;
      reg signed [35:0] temp_val_l;
      reg signed [35:0] temp_val_r;

      reg [35:0] audio_out;

      reg[4:0] state;

      //z transform coeficients (shifted left by 12 bits)
      parameter signed COEF_1B=18'd2712;
      parameter signed COEF_2B=-18'd10846;
      parameter signed COEF_3B=18'd16270;
      parameter signed COEF_4B=-18'd10846;
      parameter signed COEF_5B=18'd2712;

      parameter signed COEF_1A=4096; //2^12
      parameter signed COEF_2A=-18'd13028;
      parameter signed COEF_3A=18'd15815;
      parameter signed COEF_4A=-18'd8651;
      parameter signed COEF_5A=18'd1795;


      parameter IDLE=0;
      parameter CALC_1B=1;
      parameter CALC_2B=2;
      parameter CALC_3B=3;
      parameter CALC_4B=4;
      parameter CALC_5B=5;
      parameter CALC_2A=6;
      parameter CALC_3A=7;
      parameter CALC_4A=8;
      parameter CALC_5A=9;
      parameter PAUSE_1=10;
      parameter PAUSE_2=11;
      parameter PAUSE_3=12;
      parameter PAUSE_4=13;
      parameter PAUSE_5=14;
      parameter PAUSE_6=15;
      parameter SHIFT=16;
      parameter UPDATE_REGS=17;
      parameter TEST=18;

      reg signed [17:0] a_l;
      reg signed [17:0] b_l;
      wire signed [35:0] q_l;
      //signed multipliers
      multiplier mul1 (.clk(clock),
                                          .a(a_l),
                                          .b(b_l),
                                          .q(q_l)
                                          );

      reg signed [17:0] a_r;
      reg signed [17:0] b_r;
      wire[35:0] q_r;

      multiplier mul2 (.clk(clock),
                                          .a(a_r),
                                          .b(b_r),
                                          .q(q_r)
                                          );


      wire [17:0] audio_l_out;
      wire [17:0] audio_r_out;

      assign audio_l_out=audio_out[35:18];
      assign audio_r_out=audio_out[17:0];

      reg enabled_reg=0;

      always @ (posedge clock)
      begin
            fil_done<=0;

            if(enable)
            begin
                  enabled_reg<=~enabled_reg;
            end

            if(reset)
            begin
                  temp_val_l<=0;
                  temp_val_r<=0;
                  delay_1B_l<=0;
                  delay_1B_r<=0;
                  delay_2B_l<=0;
                  delay_2B_r<=0;
                  delay_3B_l<=0;
                  delay_3B_r<=0;
                  delay_4B_l<=0;
                  delay_4B_r<=0;

                  delay_1A_l<=0;
                  delay_1A_r<=0;
                  delay_2A_l<=0;
                  delay_2A_r<=0;
                  delay_3A_l<=0;
                  delay_3A_r<=0;
                  delay_4A_l<=0;
                  delay_4A_r<=0;

                  state<=IDLE;
            end
            else
            case(state)
                  IDLE:
                  begin
                        if(advance && enabled_reg)
                        begin
                              temp_val_l<=0;
                              temp_val_r<=0;
                              state<=CALC_1B;
                        end
                        else if(advance)
                        begin
                              audio_out<=audio_in;
                              fil_done<=1;
                              state<=IDLE;
                        end
                        else
                        begin
                              state<=IDLE;
                        end
                  end
                  // calculate numerator of transform
                  CALC_1B:
                  begin
                        a_l<=audio_in[35:18];
                        b_l<=COEF_1B;
                        a_r<=audio_in[17:0];
                        b_r<=COEF_1B;
                        state<=CALC_2B;
                  end
                  CALC_2B:
                  begin
                        a_l<=delay_1B_l;
                        b_l<=COEF_2B;
                        a_r<=delay_1B_r;
                        b_r<=COEF_2B;
                        state<=CALC_3B;
                  end
                  CALC_3B:
                  begin
                        a_l<=delay_2B_l;
                        b_l<=COEF_3B;
                        a_r<=delay_2B_r;
                        b_r<=COEF_3B;
                        state<=CALC_4B;
                  end

                  CALC_4B:
                  begin
                        a_l<=delay_3B_l;
                        b_l<=COEF_4B;
                        a_r<=delay_3B_r;
                        b_r<=COEF_4B;
                        state<=CALC_5B;
                  end

                  CALC_5B:
                  begin
                        a_l<=delay_4B_l;
                        b_l<=COEF_5B;
                        a_r<=delay_4B_r;
                        b_r<=COEF_5B;
                        state<=CALC_2A;
                  end
                  //calculate denominator of transform
                  CALC_2A:
                  begin
```

```
            a_l<=delay_1A_l;
            b_l<=COEF_2A;
            a_r<=delay_1A_r;
            b_r<=COEF_2A;

            state<=CALC_3A;
      end

      CALC_3A:
      begin
            a_l<=delay_2A_l;
            b_l<=COEF_3A;
            a_r<=delay_2A_r;
            b_r<=COEF_3A;

            temp_val_l<=temp_val_l+q_l;
            temp_val_r<=temp_val_r+q_r;

            state<=CALC_4A;
      end

      CALC_4A:
      begin
            a_l<=delay_3A_l;
            b_l<=COEF_4A;
            a_r<=delay_3A_r;
            b_r<=COEF_4A;

            temp_val_l<=temp_val_l+q_l;
            temp_val_r<=temp_val_r+q_r;

            state<=CALC_5A;
      end

      CALC_5A:
      begin
            a_l<=delay_4A_l;
            b_l<=COEF_5A;
            a_r<=delay_4A_r;
            b_r<=COEF_5A;

            temp_val_l<=temp_val_l+q_l;
            temp_val_r<=temp_val_r+q_r;

            state<=PAUSE_1;
      end

      PAUSE_1:
      begin
            temp_val_l<=temp_val_l+q_l;
            temp_val_r<=temp_val_r+q_r;

            state<=PAUSE_2;
      end

      PAUSE_2:
      begin
```

```
                        temp_val_l<=temp_val_l+q_l;
                        temp_val_r<=temp_val_r+q_r;

                        state<=PAUSE_3;
                  end

                  PAUSE_3:
                  begin
                        temp_val_l<=temp_val_l-q_l;
                        temp_val_r<=temp_val_r-q_r;

                        state<=PAUSE_4;
                  end

                  PAUSE_4:
                  begin
                        temp_val_l<=temp_val_l-q_l;
                        temp_val_r<=temp_val_r-q_r;

                        state<=PAUSE_5;
                  end

                  PAUSE_5:
                  begin
                        temp_val_l<=temp_val_l-q_l;
                        temp_val_r<=temp_val_r-q_r;

                        state<=PAUSE_6;
                  end

                  PAUSE_6:
                  begin
                        temp_val_l<=temp_val_l-q_l;
                        temp_val_r<=temp_val_r-q_r;

                        state<=SHIFT;
                  end
                  //divide by 2^12
                  SHIFT:
                  begin
                        temp_val_l[23:0]<=temp_val_l[35:12];
                        temp_val_r[23:0]<=temp_val_r[35:12];
                        state<=TEST;
                  end
                  //check for overflow
                  TEST:
                  begin
                        if((temp_val_l[23:17]==7'b1111_111) ||
(temp_val_l[23:17]==7'b0000_000))
                        begin
                              audio_out[35:18]<=temp_val_l[17:0];
                        end
                        else if(temp_val_l[23])
                        begin

      audio_out[35:18]<=18'b1000_0000_0000_0000_00;
                        end
                        else
                        begin

      audio_out[35:18]<=18'b0111_1111_1111_1111_11;
                        end

                        if((temp_val_r[23:17]==7'b1111_111) ||
(temp_val_r[23:17]==7'b0000_000))
                        begin
                              audio_out[17:0]<=temp_val_r[17:0];
                        end
                        else if(temp_val_r[23])
                        begin

      audio_out[17:0]<=18'b1000_0000_0000_0000_00;
                        end
                        else
                        begin

      audio_out[17:0]<=18'b0111_1111_1111_1111_11;
                        end

                        state<=UPDATE_REGS;

                  end

                  //shift regs
                  UPDATE_REGS:
                  begin

                        delay_1B_l<=audio_in[35:18];
                        delay_1B_r<=audio_in[17:0];
                        delay_2B_l<=delay_1B_l;
                        delay_2B_r<=delay_1B_r;
                        delay_3B_l<=delay_2B_l;
                        delay_3B_r<=delay_2B_r;
                        delay_4B_l<=delay_3B_l;
                        delay_4B_r<=delay_3B_r;

                        delay_1A_l<=audio_out[35:18];
                        delay_1A_r<=audio_out[17:0];
                        delay_2A_l<=delay_1A_l;
                        delay_2A_r<=delay_1A_r;
                        delay_3A_l<=delay_2A_l;
                        delay_3A_r<=delay_2A_r;
                        delay_4A_l<=delay_3A_l;
                        delay_4A_r<=delay_3A_r;

                        fil_done<=1;

                        state<=IDLE;
                  end

            endcase
      end
endmodule
```

```
`timescale 1ns / 1ps

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

// Company:

// Engineer:

//

// Create Date:    22:13:53 05/14/2007

// Design Name:

// Module Name:    gesture_processor

// Project Name:

// Target Devices:

// Tool versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module gesture_processor(reset, clk, gesture_code, divider,
gesture_actions_out, volume, filter);



input reset;

input clk;

input[4:0] gesture_code;

input divider;
```

```
output[1:0] volume;

output[7:0] gesture_actions_out;

output[1:0] filter;


reg[1:0] volume;

reg[7:0] gesture_actions_out;

reg[1:0] filter;


reg[1:0] play_number;


reg[9:0] gesture0, gesture1, gesture2, gesture3, gesture4, gesture5,
gesture6, gesture7,

                  gesture8, gesture9, gesture10, gesture11, gesture12,
gesture13, gesture14;


reg[9:0] volume_count, filter_count, clip_count;


parameter GESTURE0_TIMEOUT= 5;  //nothing

parameter GESTURE1_TIMEOUT= 50;  //volume up

parameter GESTURE2_TIMEOUT= 50;  //volume down

parameter GESTURE3_TIMEOUT= 300;  //low pass

parameter GESTURE4_TIMEOUT= 300;  //high pass

parameter GESTURE5_TIMEOUT= 100;  //clip0

parameter GESTURE6_TIMEOUT= 100;  //clip1

parameter GESTURE7_TIMEOUT= 100;  //clip2

parameter GESTURE8_TIMEOUT= 100;  //clip3

parameter GESTURE9_TIMEOUT= 200;  //clip4

parameter GESTURE10_TIMEOUT= 200; //clip5

parameter GESTURE11_TIMEOUT= 200; //clip6
parameter GESTURE12_TIMEOUT= 200; //clip7

parameter GESTURE13_TIMEOUT= 100; //increment play number

parameter GESTURE14_TIMEOUT= 500;


parameter VOLUME_TIMEOUT= 50;

parameter FILTER_TIMEOUT= 300;

parameter CLIP_TIMEOUT= 50;


parameter VOLUME_UP=2'b10;

parameter VOLUME_DOWN=2'b01;


parameter FILTER_HIGH=2'b10;

parameter FILTER_LOW=2'b01;

// actions_in = record_pulse, record_stop_pulse, rec_slot_in[2:0],
play_number_in[1:0], play_pulse




always @ (posedge clk)

begin


      volume<=0;

      filter<=0;

      gesture_actions_out<=0;


      if(reset)

      begin

             gesture0<=0;

             gesture1<=0;

             gesture2<=0;
```

```
       gesture3<=0;

       gesture4<=0;

       gesture5<=0;

       gesture6<=0;

       gesture7<=0;
       gesture8<=0;

       gesture9<=0;

       gesture10<=0;

       gesture11<=0;

       gesture12<=0;

       gesture13<=0;

       play_number<=0;

end

else

begin

      // increment active gestures

      if(divider)
      begin
            if(gesture1!=0)
            begin
                  gesture1<=gesture1+1;
            end

            if(gesture2!=0)
            begin
                  gesture2<=gesture2+1;
            end

            if(gesture3!=0)
            begin
                  gesture3<=gesture3+1;
            end

            if(gesture4!=0)
            begin
                  gesture4<=gesture4+1;
            end

            if(gesture5!=0)
            begin
                  gesture5<=gesture5+1;
            end
```

```
if(gesture6!=0)
begin
      gesture6<=gesture6+1;
end

if(gesture7!=0)
begin
      gesture7<=gesture7+1;
end

if(gesture8!=0)
begin
      gesture8<=gesture8+1;
end

if(gesture9!=0)
begin
      gesture9<=gesture9+1;
end

if(gesture10!=0)
begin
      gesture10<=gesture10+1;
end

if(gesture11!=0)
begin
      gesture11<=gesture11+1;
end

if(gesture12!=0)
begin
      gesture12<=gesture12+1;
end

if(gesture13!=0)
begin
      gesture13<=gesture13+1;
end

if(gesture14!=0)
begin
      gesture14<=gesture14+1;
end



if(volume_count!=0)
begin
      volume_count<=volume_count+1;
end



if(filter_count!=0)
begin
```

```
                        filter_count<=filter_count+1;
                  end



                  if(clip_count!=0)
                  begin
                        clip_count<=clip_count+1;
                  end
            end



            case(gesture_code)

                  0:

                  begin

                        gesture0<=1;

                  end

                  1:

                  begin

                        if(((gesture1>GESTURE1_TIMEOUT) ||
(gesture1==0)) && (volume_count>VOLUME_TIMEOUT || volume_count==0))

                        begin

                              volume<=VOLUME_UP;

                              gesture1<=1;

                              volume_count<=1;

                        end

                  end

                  2:

                  begin

                        if(((gesture2>GESTURE2_TIMEOUT) ||
(gesture2==0)) && (volume_count>VOLUME_TIMEOUT || volume_count==0))
                        begin
                              volume<=VOLUME_DOWN;
                              gesture2<=1;

                              volume_count<=1;
                        end

                  end
                  3:

                  begin

                        if(((gesture3>GESTURE3_TIMEOUT) ||
(gesture3==0)) && (filter_count>FILTER_TIMEOUT || filter_count==0))
                        begin
                              filter<=FILTER_LOW;
                              gesture3<=1;

                              filter_count<=1;

                        end

                  end

                  4:

                  begin

                        if(((gesture4>GESTURE4_TIMEOUT) ||
(gesture4==0)) && (filter_count>FILTER_TIMEOUT || filter_count==0))
                        begin
                              filter<=FILTER_HIGH;
                              gesture4<=1;

                              filter_count<=1;
                        end

                  end

                  5:

                  begin

                        if(((gesture5>GESTURE5_TIMEOUT) ||
(gesture5==0)) && (clip_count>CLIP_TIMEOUT || clip_count==0))
                        begin
                              gesture_actions_out[7:3]<=5'b0000_0;


      gesture_actions_out[2:1]<=play_number[1:0];

                              gesture_actions_out[0]<=1;

                              gesture5<=1;

                              clip_count<=1;
                        end

                  end

                  6:

                  begin
```

```
                        if(((gesture6>GESTURE6_TIMEOUT) ||
(gesture6==0)) && (clip_count>CLIP_TIMEOUT || clip_count==0))
                        begin
                              gesture_actions_out[7:3]<=5'b0000_1;

      gesture_actions_out[2:1]<=play_number[1:0];
                              gesture_actions_out[0]<=1;
                              gesture6<=1;

                              clip_count<=1;
                        end

                  end

                  7:

                  begin

                        if(((gesture7>GESTURE7_TIMEOUT) ||
(gesture7==0)) && (clip_count>CLIP_TIMEOUT || clip_count==0))
                        begin
                              gesture_actions_out[7:3]<=5'b0001_0;

      gesture_actions_out[2:1]<=play_number[1:0];
                              gesture_actions_out[0]<=1;
                              gesture7<=1;

                              clip_count<=1;
                        end

                  end

                  8:

                  begin

                        if(((gesture8>GESTURE8_TIMEOUT) ||
(gesture8==0)) && (clip_count>CLIP_TIMEOUT || clip_count==0))
                        begin
                              gesture_actions_out[7:3]<=5'b0001_1;

      gesture_actions_out[2:1]<=play_number[1:0];
                              gesture_actions_out[0]<=1;
                              gesture8<=1;

                              clip_count<=1;
                        end

                  end

                  9:

                  begin

                        if(((gesture9>GESTURE9_TIMEOUT) ||
(gesture9==0)) && (clip_count>CLIP_TIMEOUT || clip_count==0))
                        begin
                              gesture_actions_out[7:3]<=5'b0010_0;

      gesture_actions_out[2:1]<=play_number[1:0];
                              gesture_actions_out[0]<=1;
                              gesture9<=1;

                              clip_count<=1;
                        end

                  end

                  10:

                  begin

                        if(((gesture10>GESTURE10_TIMEOUT) ||
(gesture10==0)) && (clip_count>CLIP_TIMEOUT || clip_count==0))
                        begin
                              gesture_actions_out[7:3]<=5'b0010_1;

      gesture_actions_out[2:1]<=play_number[1:0];
                              gesture_actions_out[0]<=1;
                              gesture10<=1;

                              clip_count<=1;
                        end

                  end

                  11:

                  begin

                        if(((gesture11>GESTURE11_TIMEOUT) ||
(gesture11==0)) && (clip_count>CLIP_TIMEOUT || clip_count==0))
                        begin
                              gesture_actions_out[7:3]<=5'b0011_0;

      gesture_actions_out[2:1]<=play_number[1:0];
                              gesture_actions_out[0]<=1;
                              gesture11<=1;

                              clip_count<=1;

                        end

                  end

                  12:

                  begin

                        if(((gesture12>GESTURE12_TIMEOUT) ||
(gesture12==0)) && (clip_count>CLIP_TIMEOUT || clip_count==0))
                        begin
                              gesture_actions_out[7:3]<=5'b0011_1;
      gesture_actions_out[2:1]<=play_number[1:0];
                              gesture_actions_out[0]<=1;
                              gesture12<=1;

                              clip_count<=1;

                        end

                  end

                  13:

                  begin

                        if((gesture13>GESTURE13_TIMEOUT) ||
(gesture13==0))
                        begin
                              play_number<=play_number+1;

                              gesture13<=1;


                        end

                  end

                  14:

                  begin

                        if((gesture14>GESTURE14_TIMEOUT) ||
(gesture14==0))
                        begin
                              gesture14<=1;
                        end

                  end


                  default:

                  begin



                  end

      endcase


      end
```

```
end




endmodule


`timescale 1ns / 1ps

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

// Gyroscope Decoder
//
// This unit receives as input the rotational acceleration around the y
axis for the right hand and the left hand.  It decodes these

// acceleration values into a gyro code corresponding to the action

// that has been sensed.  The unit waits in the idle state until it
receives

// a request from the master FSM for a gyro code upon which it
// performs
 all the necessary calculations to arrive at a gyro code.

//

// Code | Left  | Right

// -----|-------|------

// 0    | X     | X

// 1    | CW    | X

// 2    | CCW   | X

// 3    | X     | CW

// 4    | X     | CCW

// 5    | CW    | CW

// 6    | CCW   | CCW

// 7    | CCW   | CW
// 8    | CW    | CCW

//

// In the table above, X indicates no movement, CW stands for
clockwise, and CCW stands for counterclockwise.
//
//////////////////////////////////////////////////////////////////////
///////////

module gyro_decoder(clock, reset_sync, left_rotate, right_rotate,
request, busy, gyro_code);

    input clock;

       input reset_sync;
    input signed [16:0] left_rotate;
    input signed [16:0] right_rotate;
    input request;
    output reg busy;
    output reg [3:0] gyro_code;

       reg [1:0] state, next;

       reg [16:0] right, left;

       //threshold values
       parameter zero_bias = 128;

       parameter min_rotate = 70;

       //states
       parameter IDLE = 0;

       parameter decode = 1;

//state assignment
always @ (posedge clock) begin
      if (reset_sync) state <= IDLE;
      else begin

            state <= next;

      end
end

//next state computation
always @ (state or request or right_rotate or left_rotate) begin
      busy = 0;

      case (state)


            IDLE:
                  begin
                        if (request) next = decode;
                        else next = IDLE;



                        gyro_code = 0;
                  end

            decode:
                  begin
                        next = IDLE;


                        busy = 1;



                        if ((right_rotate > zero_bias+min_rotate) &
(left_rotate > zero_bias+min_rotate)) // both hands CW
                              gyro_code = 5;
                        else if ((right_rotate < zero_bias-min_rotate)
& (left_rotate < zero_bias-min_rotate)) // both hands CCW
                              gyro_code = 6;
                        else if ((right_rotate > zero_bias+min_rotate)
& (left_rotate < zero_bias-min_rotate)) // RH CW and LH CCW
                              gyro_code = 7;
                        else if ((right_rotate < zero_bias-min_rotate)
& (left_rotate > zero_bias+min_rotate)) // RH CCW and LH CW
                              gyro_code = 8;
                        else if (right_rotate > zero_bias+min_rotate)
// only RH CW
                              gyro_code = 3;
                        else if (right_rotate < zero_bias-min_rotate)
// only RH CCW
                              gyro_code = 4;
                        else if (left_rotate > zero_bias+min_rotate) //
only LH CW
                              gyro_code = 1;
                        else if (left_rotate < zero_bias-min_rotate) //
only LH CCW
                              gyro_code = 2;



                  end


            default:

                  begin

                        next = IDLE;

                  end

      endcase
```

```
end



endmodule


`timescale 1ns / 1ps

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

// Gyroscope Processor
//
// This unit interfaces with the A/D converter to receive digitized raw
data
// from the gyroscopes.  When output is to be assigned, the most recent
value read from the A/D is assigned to the output. On reset, the zero-
bias value is output.
//

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

module gyro_processor(clock, reset_sync, enable, left_rotate,
right_rotate, ADbusy, convst_b, rd_b,

             left_rotate_out, right_rotate_out, state);
    input clock;
    input reset_sync;
    input enable;
    input [7:0] left_rotate;
    input [7:0] right_rotate;
       input ADbusy;
       output reg convst_b;
       output reg rd_b;
       output reg [16:0] left_rotate_out;
       output reg [16:0] right_rotate_out;

       output reg [2:0] state;
       reg [2:0] next;
       reg cap_val;
       reg set_out;

       reg [7:0] left_val = 0;

       reg [7:0] right_val = 0;

       //states
       parameter IDLE = 0;
       parameter ADActivate = 1;
       parameter Pause = 2;
       parameter ADRead = 3;

       parameter ADRead1 = 4;
       parameter Convert = 5;
       parameter Add = 6;

       parameter Output = 7;

//state assignment
always @ (posedge clock) begin
      if (reset_sync) begin
            state <= IDLE;

            left_rotate_out <= 128;
//output zero-bias values
            right_rotate_out <= 128;
      end else begin

            state <= next;

            if(cap_val) begin // capture input from the A/D
                  left_val <= left_rotate;
                  right_val <= right_rotate;
            end

            if(set_out) begin
// set outputs
                  left_rotate_out <= left_val;

                  right_rotate_out <= right_val;
            end
      end
end

//next state computation
always @ (state or enable or ADbusy) begin
      convst_b = 1;
      rd_b = 1;
      cap_val = 0;
      set_out = 0;

      case (state)
            IDLE:
                  begin
                        if (enable) next = ADActivate;
                        else next = IDLE;
                  end

            ADActivate:
                  begin
                        next = Pause;

                        convst_b = 0; // indicate start conversion
                  end

            Pause:
                  begin
                        if (!ADbusy) next = ADRead; // wait for
conversion
                        else next = Pause;
                  end
            ADRead:
                  begin
                        next = ADRead1;

                        rd_b = 0; // indicate start read
                  end

            ADRead1:
                  begin
                        next = Convert;

                        rd_b = 0;
                        cap_val = 1;
                  end

            Convert:
                  begin
                        next = Add;

                  end

            Add:
                  begin
                        next = Output;

                  end

            Output:
                  begin
                        next = IDLE;

                        set_out = 1;
                  end

            default:
                  begin
                        next = IDLE;
                  end

      endcase
end




endmodule


///////////////////////////////////////////////////////////////////////
////////
//
// 6.111 FPGA Labkit -- Lab 4: Pong
//
//
// Created: March 15, 2007
// Author: Nathan Ickes
//
// This is a template for implementing the Pong game for Lab 4. This
file
// includes two modules:
//
//   - labkit: the top-level labkit module
//   - debounce: the synchronize/debounce module
//
// Students should modify and add modules according to the directions
outlined
// in the lab 4 manual.
//
///////////////////////////////////////////////////////////////////////
////////

///////////////////////////////////////////////////////////////////////
////////
//
// 6.111 FPGA Labkit -- Template Toplevel Module for Lab 4 (Spring
2007)
//
//
// Created: March 15, 2007
// Author: Nathan Ickes
//
///////////////////////////////////////////////////////////////////////
////////

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

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

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

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

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

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

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

             rs232_txd, rs232_rxd, rs232_rts, rs232_cts,

             mouse_clock, mouse_data, keyboard_clock, keyboard_data,

             clock_27mhz, clock1, clock2,

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

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

             switch,

             led,

             user1, user2, user3, user4,

             daughtercard,

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

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

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

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

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

   input  [19:0] tv_in_ycrcb;
   input  tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2,
tv_in_aef,
        tv_in_hff, tv_in_aff;
   output tv_in_i2c_clock, tv_in_fifo_read, tv_in_fifo_clock,
tv_in_iso,
        tv_in_reset_b, tv_in_clock;
   inout  tv_in_i2c_data;
   inout  [35:0] ram0_data;
   output [18:0] ram0_address;
   output ram0_adv_ld, ram0_clk, ram0_cen_b, ram0_ce_b, ram0_oe_b,
ram0_we_b;
   output [3:0] ram0_bwe_b;

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

   input  clock_feedback_in;
   output clock_feedback_out;

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

   output rs232_txd, rs232_rts;
   input  rs232_rxd, rs232_cts;

   input  mouse_clock, mouse_data, keyboard_clock, keyboard_data;

   input  clock_27mhz, clock1, clock2;

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

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

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

   inout [43:0] daughtercard;

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

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


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

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

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

      // VGA Output
   assign vga_out_red = 10'h0;
   assign vga_out_green = 10'h0;
   assign vga_out_blue = 10'h0;
   assign vga_out_sync_b = 1'b1;
   assign vga_out_blank_b = 1'b1;
   assign vga_out_pixel_clock = 1'b0;
   assign vga_out_hsync = 1'b0;
   assign vga_out_vsync = 1'b0;

   // Video Output
   assign tv_out_ycrcb = 10'h0;
   assign tv_out_reset_b = 1'b0;
   assign tv_out_clock = 1'b0;
   assign tv_out_i2c_clock = 1'b0;
   assign tv_out_i2c_data = 1'b0;
   assign tv_out_pal_ntsc = 1'b0;
   assign tv_out_hsync_b = 1'b1;
   assign tv_out_vsync_b = 1'b1;
   assign tv_out_blank_b = 1'b1;
   assign tv_out_subcar_reset = 1'b0;

   // Video Input
   assign tv_in_i2c_clock = 1'b0;
   assign tv_in_fifo_read = 1'b0;
   assign tv_in_fifo_clock = 1'b0;
   assign tv_in_iso = 1'b0;
   assign tv_in_reset_b = 1'b0;
   assign tv_in_clock = 1'b0;
   assign tv_in_i2c_data = 1'bZ;

   // 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;

   // 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 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;
```

```
            wire clk_fpga;

      //ram clock
      ramclock rmclk (.ref_clock(clock_27mhz),
                              .fpga_clock(clk_fpga),
                              .ram0_clock(ram0_clk),
                              .ram1_clock(ram1_clk),

      .clock_feedback_in(clock_feedback_in),

      .clock_feedback_out(clock_feedback_out),
                              .locked()
                              );

//////////////////////////////////////////////////////////////////////
/////
//
// Xtremix Gesture Components
//
//////////////////////////////////////////////////////////////////////
/////

//connect wires
wire fsm_enable, enable;
wire accel_busy, gyro_busy;
wire accel_request, gyro_request;
wire [4:0] accel_code;
wire [3:0] gyro_code;
wire [7:0] left_accel_x_out, left_accel_y_out, left_accel_z_out;
wire [7:0] right_accel_x_out, right_accel_y_out, right_accel_z_out;
wire [16:0] left_rotate_out, right_rotate_out;
wire [7:0] left_val_x, left_read_val_x;
wire [15:0] left_sum_x;
wire [3:0] accel_state;
wire [2:0] gyro_state;
wire [4:0] gesture_code;
reg temp, reset_sync;
wire [7:0] def_x, def_y, def_z;
wire [16:0] def_rot;
wire [4:0] int_accel_code;
wire [3:0] int_gyro_code;
wire [3:0] fsm_state;
wire ADbusy_accel, convst_b_accel, rd_b_accel;
wire ADbusy_gyro, convst_b_gyro, rd_b_gyro;

assign ADbusy_accel = user3[23] && user3[3] && user4[20] && user4[3];
assign user3[22] = convst_b_accel;
assign user3[2] = convst_b_accel;
assign user4[19] = convst_b_accel;
assign user4[2] = convst_b_accel;
assign user3[21] = rd_b_accel;
assign user3[1] = rd_b_accel;
assign user4[18] = rd_b_accel;
assign user4[1] = rd_b_accel;

assign ADbusy_gyro = user1[23] && user2[3];
assign user1[22] = convst_b_gyro;
assign user1[21] = rd_b_gyro;
assign user2[2] = convst_b_gyro;
assign user2[1] = rd_b_gyro;

assign def_x = 80;
assign def_y = 65;
assign def_z = 0;
assign def_rot = 128;

assign led[4:0] = gesture_code;
assign led[7:5] = 0;

//synchronize reset
always @ (posedge clk_fpga) begin
      temp <= ~button0;
      reset_sync <= temp;
end

//instantiate master_fsm
master_fsm master (
      .clock(clk_fpga),
      .reset_sync(reset_sync),
      .enable(fsm_enable),
      .accel_busy(accel_busy),
      .gyro_busy(gyro_busy),
      .accel_code(accel_code),
      .gyro_code(gyro_code),
      .gyro_request(gyro_request),
      .accel_request(accel_request),
      .gesture_code(gesture_code),
      .state(fsm_state),
      .int_accel_code(int_accel_code),
      .int_gyro_code(int_gyro_code)
);



//instantiate accel_num
accel_num accel_num (
      .clock(clk_fpga),
      .reset_sync(reset_sync),
      .enable(enable),
      .left_accel_x(user3[31:24]),
      .left_accel_y(user3[11:4]),
      .left_accel_z(def_z),
      .right_accel_x(user4[11:4]),
      .right_accel_y(user4[31:24]),
      .right_accel_z(def_z),
      .ADbusy(ADbusy_accel),
      .convst_b(convst_b_accel),
      .rd_b(rd_b_accel),
      .left_accel_x_out(left_accel_x_out),
      .left_accel_y_out(left_accel_y_out),
      .left_accel_z_out(left_accel_z_out),
      .right_accel_x_out(right_accel_x_out),
      .right_accel_y_out(right_accel_y_out),
      .right_accel_z_out(right_accel_z_out),
		.left_read_val_x(left_read_val_x),
		.left_val_x(left_val_x),
		.left_sum_x(left_sum_x),
		.state(accel_state)
	);

	//instantiate accel_decoder
	accel_decoder accel_decoder (
		.clock(clk_fpga),
		.reset_sync(reset_sync),
		.left_accel_x(left_accel_x_out),
		.left_accel_y(left_accel_y_out),
		.left_accel_z(left_accel_z_out),
		.right_accel_x(right_accel_x_out),
		.right_accel_y(right_accel_y_out),
		.right_accel_z(right_accel_z_out),
		.request(accel_request),
		.busy(accel_busy),
		.accel_code(accel_code)
	);

	//instantiate gyro_processor
	gyro_processor gyro_processor (
		.clock(clk_fpga),
		.reset_sync(reset_sync),
		.enable(enable),
		.left_rotate(user1[31:24]),
		.right_rotate(user2[11:4]),
		.ADbusy(ADbusy_gyro),
		.convst_b(convst_b_gyro),
		.rd_b(rd_b_gyro),
		.left_rotate_out(left_rotate_out),
		.right_rotate_out(right_rotate_out),
		.state(gyro_state)
	);

	//instantiate gyro_decoder
	gyro_decoder gyro_decoder (
		.clock(clk_fpga),
		.reset_sync(reset_sync),
		.left_rotate(left_rotate_out),
		.right_rotate(right_rotate_out),
		.request(gyro_request),
		.busy(gyro_busy),
		.gyro_code(gyro_code)
	);

	// Instantiate fsm_divider
	fsm_divider fsm_divider (
		.clock(clk_fpga),
		.reset_sync(reset_sync),
		.enable(fsm_enable)
	);

	//instantiate adc divider
	adc_divider adc_divider(
		.clock(clk_fpga),
      .reset_sync(reset_sync),
      .enable(enable)
);

assign user3[20:12] = 0;
assign user3[0] = 0;
assign user1[20:0] = 0;
assign user2[31:12] = 0;
assign user2[0] = 0;
assign user4[23:21] = 0;
assign user4[17:12] = 0;
assign user4[0] = 0;

/*assign analyzer1_clock = clock_27mhz;
assign analyzer1_data[7:0] = user4[31:24];
assign analyzer1_data[15:8] = user4[11:4];

assign analyzer2_clock = clock_27mhz;
assign analyzer2_data[7:0] = user2[11:4];
assign analyzer2_data[11:8] = fsm_state;
assign analyzer2_data[15:12] = int_gyro_code;

assign analyzer3_clock = clock_27mhz;
assign analyzer3_data[4:0] = gesture_code;
assign analyzer3_data[9:5] = int_accel_code;
assign analyzer3_data[10] = user2[3];
assign analyzer3_data[11] = user2[2];
assign analyzer3_data[12] = user2[1];
assign analyzer3_data[13] = user4[20];
assign analyzer3_data[14] = user4[19];
assign analyzer3_data[15] = user4[18];

assign analyzer4_clock = clock_27mhz;
assign analyzer4_data[15:0] = left_rotate_out[15:0];*/

//////////////////////////////////////////////////////////////////////
////
//
// XtremiX Inteface Modules
//
//////////////////////////////////////////////////////////////////////
////

//wires
wire[7:0] gesture_actions_out;
wire[1:0] volume, filter;
wire[7:0] actions_out;
wire divider;

//instantiate gesture_processor
gesture_processor gesture_processor (
            .reset(reset_sync),
            .clk(clk_fpga),
            .gesture_code(gesture_code),
            .divider(divider),
            .gesture_actions_out(gesture_actions_out),
            .volume(volume),
            .filter(filter)
      );


divider divider_1hundredth (.clock(clk_fpga),

      .reset(reset_sync),
                                                         .pulse(divider)
                                                         );

///////////////////////////////////////////////////////////////////////
////
//
// XtremiX Audio Components
//
///////////////////////////////////////////////////////////////////////
////

wire mutein;
      assign mutein=switch[0];

      wire reset;
      wire play, play_d;
      wire record, record_d;
      wire record_stop, record_stop_d;
      wire enter, enter_d;

      wire[3:0] state;
      wire[7:0] bitcount;

      not n1 (play_d, button1);
      not n2 (record_d, button2);
      not n3 (record_stop_d, button3);
      not n4 (play_macro_d, button_right);
      not n5 (stop_macro_d, button_left);
      not n6 (stop_record_macro_d, button_down);
      not n7 (start_record_macro_d, button_up);
      not n8 (enter_d, button_enter);

      debounce deb_play (.reset(reset_sync),
                                    .clock(clk_fpga),
                                    .noisy(play_d),
                                    .clean(play)
                                    );

      debounce deb_record (.reset(reset_sync),
                                    .clock(clk_fpga),
                                    .noisy(record_d),
                                    .clean(record)
                                    );
      debounce deb_record_stop (.reset(reset_sync),
                                    .clock(clk_fpga),
                                    .noisy(record_stop_d),
                                    .clean(record_stop)
                                    );

      debounce deb_play_macro (.reset(reset_sync),
                                    .clock(clk_fpga),
                                    .noisy(play_macro_d),
                                    .clean(play_macro)
                                    );

debounce deb_stop_macro (.reset(reset_sync),
                                    .clock(clk_fpga),
                                    .noisy(stop_macro_d),
                                    .clean(stop_macro)
                                    );

debounce deb_start_record_macro (.reset(reset_sync),
                                    .clock(clk_fpga),
                                    .noisy(start_record_macro_d),
                                    .clean(start_record_macro)
                                    );

debounce deb_stop_record_macro (.reset(reset_sync),
                                    .clock(clk_fpga),
                                    .noisy(stop_record_macro_d),
                                    .clean(stop_record_macro)
                                    );

      debounce deb_enter (.reset(reset_sync),
                                    .clock(clk_fpga),
                                    .noisy(enter_d),
                                    .clean(enter)
                                    );


wire read;
wire write;
wire[18:0] address;
wire[35:0] data;
wire busy;
wire[3:0] cont_state;
wire[2:0] ztest_state;
wire[6:0] counter;
wire data_oen;
wire sound_done;
wire[17:0] sound_out_r;
wire[17:0] sound_out_l;
wire[17:0] sound_in_left;
wire[17:0] sound_in_right;
wire[4:0] master_state;
wire idle_s;
wire[35:0] zbt_read_data;
wire[35:0] zbt_write_data;


wire play_pulse, record_pulse, record_stop_pulse;
wire[18:0] playback_counter;
wire[5:0] slot;
wire[1:0] play_number;

wire transmit;
```

```
      wire play_in;
      wire[2:0] rec_slot_in;
      wire[1:0] play_number_in;

      wire play_pulse_out;
      wire[2:0] rec_slot_out;
      wire[1:0] play_number_out;

      wire[1:0] filter_enable;
      wire[1:0] volume_enable;
      wire[1:0] volume_enable_sync;

      wire[4:0] audio_volume;

      //audio ac97 controller
            audioinout alp(.clock(clk_fpga),
                        .audio_reset_b(audio_reset_b),
                        .ac97_sdata_out(ac97_sdata_out),
                        .ac97_sdata_in(ac97_sdata_in),
                  .ac97_synch(ac97_synch),
                        .ac97_bit_clock(ac97_bit_clock),
                        .mutein(1),
                        .reset(reset_sync),
                        .leds(),
                        .state(state),
                        .bit_count(bitcount),
                        .sound_done(sound_done),
                        .sound_out_r(sound_out_r),
                        .sound_out_l(sound_out_l),
                        .sound_in_right(sound_in_right),
                        .sound_in_left(sound_in_left),
                        .volume_enable_sync(volume_enable_sync),
                        .audio_volume(audio_volume)
                        );

      wire[1:0] filter_enable_sync;

      filter_enable_sync fil_enable_1 (.reset(reset_sync),
                                           .clk(clk_fpga),

.filter_enable(filter_enable),
                                           .filter(filter),

.filter_enable_sync(filter_enable_sync)
                                          );

      volume_sync vol_syn (.reset(reset_sync),
                                          .clk(clk_fpga),

      .sound_done_pulse(sound_done_pulse),

      .volume_enable(volume_enable),

      .volume_enable_sync(volume_enable_sync),
                                          .volume(volume)
                                          );
```

```
	// ZBT controller for ram module 0
	zbtcontroller zcont0 (.clk(clk_fpga),
	                                    .read(read),
	                                    .write(write),
	                                    .reset_sync(reset_sync),
	                                    .input_address(address),
	                                    .input_data(zbt_write_data),
	                                    .busy(busy),
	                                    .cen(ram0_cen_b),
	                                    .we(ram0_we_b),
	                                    .ext_address(ram0_address),
	                                    .read_data(zbt_read_data),
	                                    .ext_data(ram0_data),
	                                    .state(cont_state),
	                                    .data_oen(data_oen)
	                                    );
	// ZBT 2

	wire [18:0] address1;
	wire [35:0] zbt_write_data1;
	wire [35:0] zbt_read_data1;

	// ZBT controller for ram module 1
	zbtcontroller zcont1 (.clk(clk_fpga),
	                                    .read(read1),
	                                    .write(write1),
	                                    .reset_sync(reset_sync),
	                                    .input_address(address1),
	                                    .input_data(zbt_write_data1),
	                                    .busy(busy1),
	                                    .cen(ram1_cen_b),
	                                    .we(ram1_we_b),
	                                    .ext_address(ram1_address),
	                                    .read_data(zbt_read_data1),
	                                    .ext_data(ram1_data),
	                                    .state(),
	                                    .data_oen()
	                                    );

	wire[35:0] play_data1;
		wire start_masfsm1;
		wire[4:0] master_state1;

	// play/rec tracker for zbt controller 1
	masterzbt masfsm1 (.reset(reset_sync),
	                              .clk(clk_fpga),
	                              .play(play_pulse_out),
	                              .record(record_pulse_out),
	                              .rec_data({sound_out_l,
sound_out_r}),
	                              .play_data(play_data1),
	                              .write_data(zbt_write_data1),
	                              .read_data(zbt_read_data1),
	                              .sound_done(sound_done_pulse1),
	                              .read(read1),
	                              .write(write1),
                                        .address(address1),
                                        .state(master_state1),
                                        .idle_s(),

      .record_stop(record_stop_pulse_out),

                                        .slot({4'h0, rec_slot_out[2:1]}),
                                        .play_number(play_number_out),
                                        .transmit(transmit),
                                        .busy(busy1),
                                        .address_1(),
                                        .playing_1(),
                                        .final_address_1(),
                                        .start_add(start_add1),
                                        .enable_inputs(rec_slot_out[0]),
                                        .start_record_signal(fil_low_done)

                                        );

      wire[18:0] address_1;
      wire playing_1;
      wire[18:0] final_address_1;
      wire[35:0] play_data;
      wire start_add;

      not invswitch5 (rec_enable_inv, rec_slot_out[0]);

      // play/rec tracker for zbt controller 0
      masterzbt masfsm0 (.reset(reset_sync),
                                        .clk(clk_fpga),
                                        .play(play_pulse_out),
                                        .record(record_pulse_out),
                                        .rec_data({sound_out_l,
sound_out_r}),
                                        .play_data(play_data),
                                        .write_data(zbt_write_data),
                                        .read_data(zbt_read_data),
                                        .sound_done(sound_done_pulse),
                                        .read(read),
                                        .write(write),
                                        .address(address),
                                        .state(master_state),
                                        .idle_s(idle_s),

      .record_stop(record_stop_pulse_out),

                                        .slot({4'h0, rec_slot_out[2:1]}),
                                        .play_number(play_number_out),
                                        .transmit(),
                                        .busy(busy),
                                        .address_1(address_1),
                                        .playing_1(playing_1),
                                        .final_address_1(final_address_1),
                                        .start_add(start_add),
                                        .enable_inputs(rec_enable_inv),
                                        .start_record_signal(fil_low_done)
                                        );
          wire[35:0] filter_in;
          wire[35:0] fil_high_out;
          wire[3:0] combiner_state;

          //signal combiner for both trackers
          signal_combiner sigcomb (.reset(reset_sync),
                                        .clk(clk_fpga),
                                        .play_data(filter_in),
                                        .start_add(start_add),
                                        .start_add1(start_add1),
                                        .sound_done(sound_done_pulse),
                                        .input_data(play_data),
                                        .input_data1(play_data1),
                                        .comb_done(comb_done),
                                        .stream_data({sound_in_left,
    sound_in_right}),
                                        .state(combiner_state),
                                        .start_record_signal() //must be in
    last module!!
                                        );

    //filters

    //high pass
          gen_filter fil_high (.reset(reset_sync),
                                              .audio_in(filter_in),
                                              .audio_out(fil_high_out),
                                              .clock(clk_fpga),

          .enable(filter_enable_sync[1]),
                                              .advance(comb_done),
                                              .fil_done(fil_high_done),
                                              .enabled_reg(enabled_reg)
                                              );
          defparam fil_high.COEF_1B=18'd2712;
          defparam fil_high.COEF_2B=-18'd10846;
          defparam fil_high.COEF_3B=18'd16270;
          defparam fil_high.COEF_4B=-18'd10846;
          defparam fil_high.COEF_5B=18'd2712;
          defparam fil_high.COEF_1A=4096; //2^10
          defparam fil_high.COEF_2A=-18'd13028;
          defparam fil_high.COEF_3A=18'd15815;
          defparam fil_high.COEF_4A=-18'd8651;
          defparam fil_high.COEF_5A=18'd1795;

    //low pass

          gen_filter fil_low (.reset(reset_sync),
                                              .audio_in(fil_high_out),
                                              .audio_out({sound_out_l,
    sound_out_r}),
                                              .clock(clk_fpga),

          .enable(filter_enable_sync[0]),
                                              .advance(fil_high_done),
                                              .fil_done(fil_low_done)
                                              );
          defparam fil_low.COEF_1B=18'd2;
          defparam fil_low.COEF_2B=18'd7;
          defparam fil_low.COEF_3B=18'd10;
          defparam fil_low.COEF_4B=18'd7;
          defparam fil_low.COEF_5B=18'd2;
          defparam fil_low.COEF_1A=4096; //2^12
          defparam fil_low.COEF_2A=-18'd13028;
          defparam fil_low.COEF_3A=18'd15815;
          defparam fil_low.COEF_4A=-18'd8651;
          defparam fil_low.COEF_5A=18'd1795;

          wire[1:0] controller_state;



          //takes inputs and outputs them at the correct time
          mastercontroller mascont (.reset(reset_sync),
                                                          99.clk(clk_fpga),
                                                          .play(play),
                                                          .record(record),

          .record_stop(record_stop),

          .sound_done(sound_done),

          .play_pulse(play_pulse),

          .record_pulse(record_pulse),

          .record_stop_pulse(record_stop_pulse),

          .sound_done_pulse(sound_done_pulse),

          .sound_done_pulse1(sound_done_pulse1),


          .sound_done_pulse_reg(sound_done_pulse_reg),

          .transmit(transmit),

          .rec_slot(switch[7:5]),

          .rec_slot_in(rec_slot_in),

          .play_number(switch[4:3]),

          .play_number_in(play_number_in),

          .state(controller_state),
                                                          .enter(enter),

          .filter_selector(switch[2]),

          .filter_enable(filter_enable),
          .volume_selector({switch[1], switch[0]}),

          .volume_enable(volume_enable)
                                                          );

                wire[7:0] actions_in;

                assign actions_in={record_pulse, record_stop_pulse,
    rec_slot_in[2:0], play_number_in[1:0], play_pulse};
    //          assign actions_out={rec_slot_out[2:0],
    play_number_out[1:0], play_pulse_out};
                wire[1:0] other_wires;

                //listen to actions from the master controller
          macro_listener mac2 (.clock(clk_fpga),
                                                    .reset(reset_sync),

          .start_record(start_record_macro),

          .stop_record(stop_record_macro),

          .start_play(play_macro),
                                                    .stop_play(stop_macro),

          .actions_in(actions_in),

          .gesture_actions_out(gesture_actions_out),


          .actions_out({record_pulse_out, record_stop_pulse_out,
    rec_slot_out[2:0], play_number_out[1:0], play_pulse_out})
                                                    );

          assign analyzer1_clock=clk_fpga;
          assign analyzer1_data[15:8]={volume_enable_sync[1:0],
    filter_enable[0], audio_volume};
          assign analyzer1_data[7:0]=actions_out;


       // SRAMs
       assign ram0_adv_ld = 1'b0;
       assign ram0_ce_b = 1'b0;
       assign ram0_oe_b = 1'b0;
       assign ram0_bwe_b = 4'h0;

       assign ram1_adv_ld = 1'b0;
       assign ram1_ce_b = 1'b0;
       assign ram1_oe_b = 1'b0;
       assign ram1_bwe_b = 4'h0;


          ////////////////////////////////////////////////////////////////
    ///////////
          //
          // Test Gyro_Processor Components
          //
```

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

	/*//connect wires
	reg temp, reset_sync;
	wire enable;
	wire [16:0] left_rotate_out;
	wire [2:0] state;
	wire [7:0] left_read_val_x, left_val_x;

	//synchronize reset
	always @ (posedge clock_27mhz) begin
		temp <= ~button0;
		reset_sync <= temp;
	end

	//instantiate adc divider
	adc_divider adc_divider(
		.clock(clock_27mhz),
		.reset_sync(reset_sync),
		.enable(enable)
	);

	//instantiate gyro_processor
	gyro_processor gyro_processor (
		.clock(clock_27mhz),
		.reset_sync(reset_sync),
		.enable(enable),
		.left_rotate(user1[31:24]),
		.right_rotate(user1[20:13]),
		.ADbusy(user1[23]),
		.convst_b(user1[22]),
		.rd_b(user1[21]),
		.left_rotate_out(left_rotate_out),
		.right_rotate_out(right_rotate_out),
		.state(state)
	);

	assign user1[20:0] = 0;

	assign analyzer3_clock = clock_27mhz;
	assign analyzer3_data[7:0] = user1[31:24];
	assign analyzer3_data[10:8] = state[2:0];
	assign analyzer3_data[12:11] = 0;
	assign analyzer3_data[13] = user1[23];
	assign analyzer3_data[14] = user1[22]&&user1[21];
	assign analyzer3_data[15] = reset_sync;

	assign analyzer2_clock = clock_27mhz;
	assign analyzer2_data[15:0] = left_rotate_out[15:0];*/

	////////////////////////////////////////////////////////////////
///////////
	//
	// Test Accel_Processor Components
	//
```

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

      /*//connect wires
      reg temp, reset_sync;
      wire enable;
      wire [7:0] left_accel_x_out, left_accel_y_out, left_accel_z_out,
right_accel_x_out,
                              right_accel_y_out, right_accel_z_out;
      wire [15:0] left_sum_x;
      wire [3:0] state;
      wire [7:0] left_read_val_x, left_val_x;

      //synchronize reset
      always @ (posedge clock_27mhz) begin
            temp <= ~button0;
            reset_sync <= temp;
      end

      //instantiate adc divider
      adc_divider adc_divider(
            .clock(clock_27mhz),
            .reset_sync(reset_sync),
            .enable(enable)
      );

      //instantiate accelerator unit
      accel_num accel_num (
            .clock(clock_27mhz),
            .reset_sync(reset_sync),
            .enable(enable),
            .left_accel_x(user1[31:24]),
            .left_accel_y(user1[20:13]),
            .left_accel_z(user1[20:13]),
            .right_accel_x(user1[20:13]),
            .right_accel_y(user1[20:13]),
            .right_accel_z(user1[20:13]),
            .ADbusy(user1[23]),
            .convst_b(user1[22]),
            .rd_b(user1[21]),
            .left_accel_x_out(left_accel_x_out),
            .left_accel_y_out(left_accel_y_out),
            .left_accel_z_out(left_accel_z_out),
            .right_accel_x_out(right_accel_x_out),
            .right_accel_y_out(right_accel_y_out),
            .right_accel_z_out(right_accel_z_out),
            .left_val_x(left_val_x),
            .left_read_val_x(left_read_val_x),
            .left_sum_x(left_sum_x),
            .state(state)
      );

      assign user1[20:0] = 0;

      assign analyzer3_clock = clock_27mhz;
      assign analyzer3_data[0] = left_sum_x[0];
      assign analyzer3_data[1] = left_sum_x[1];
```

```
	assign analyzer3_data[2] = left_sum_x[2];
	assign analyzer3_data[3] = left_sum_x[3];
	assign analyzer3_data[4] = left_sum_x[4];
	assign analyzer3_data[5] = left_sum_x[5];
	assign analyzer3_data[6] = left_sum_x[6];
	assign analyzer3_data[7] = left_sum_x[7];
	assign analyzer3_data[8] = left_sum_x[8];
	assign analyzer3_data[9] = left_sum_x[9];
	assign analyzer3_data[10] = left_sum_x[10];
	assign analyzer3_data[11] = left_sum_x[11];
	assign analyzer3_data[12] = left_sum_x[12];
	assign analyzer3_data[13] = user1[23];
	assign analyzer3_data[14] = user1[22]&&user1[21];
	assign analyzer3_data[15] = reset_sync;

	assign analyzer2_clock = clock_27mhz;
	assign analyzer2_data[0] = user1[24];
	assign analyzer2_data[1] = user1[25];
	assign analyzer2_data[2] = user1[26];
	assign analyzer2_data[3] = user1[27];
	assign analyzer2_data[4] = user1[28];
	assign analyzer2_data[5] = user1[29];
	assign analyzer2_data[6] = user1[30];
	assign analyzer2_data[7] = user1[31];
	assign analyzer2_data[8] = left_val_x[0];
	assign analyzer2_data[9] = left_val_x[1];
	assign analyzer2_data[10] = left_val_x[2];
	assign analyzer2_data[11] = left_val_x[3];
	assign analyzer2_data[12] = left_val_x[4];
	assign analyzer2_data[13] = left_val_x[5];
	assign analyzer2_data[14] = left_val_x[6];
	assign analyzer2_data[15] = left_val_x[7];

	assign analyzer1_clock = clock_27mhz;
	assign analyzer1_data[0] = left_accel_x_out[0];
	assign analyzer1_data[1] = left_accel_x_out[1];
	assign analyzer1_data[2] = left_accel_x_out[2];
	assign analyzer1_data[3] = left_accel_x_out[3];
	assign analyzer1_data[4] = left_accel_x_out[4];
	assign analyzer1_data[5] = state[0];
	assign analyzer1_data[6] = state[1];
	assign analyzer1_data[7] = state[2];
	assign analyzer1_data[8] = left_read_val_x[0];
	assign analyzer1_data[9] = left_read_val_x[1];
	assign analyzer1_data[10] = left_read_val_x[2];
	assign analyzer1_data[11] = left_read_val_x[3];
	assign analyzer1_data[12] = left_read_val_x[4];
	assign analyzer1_data[13] = left_read_val_x[5];
	assign analyzer1_data[14] = left_read_val_x[6];
	assign analyzer1_data[15] = left_read_val_x[7];*/

	////////////////////////////////////////////////////////////////
///////////
	//
	// A/D Testing Unit
	//
	//////////////////////////////////////////////////////////////////
///////////

	/*//connect wires
	reg temp, reset_sync;
	wire enable;

	//synchronize reset
	always @ (posedge clock_27mhz) begin
		temp <= ~button0;
		reset_sync <= temp;
	end

	//instantiate adc divider
	adc_divider ADtester_divider(
		.clock(clock_27mhz),
		.reset_sync(reset_sync),
		.enable(enable)
	);

	//instantiate ADtester
	ADtester ADtester (
		.clock(clock_27mhz),
		.reset_sync(reset_sync),
		.enable(enable),
		.ADout(user1[31:24]),
		.ADbusy(user1[23]),
		.convst_b(user1[22]),
		.rd_b(user1[21]),
		.out(out)
	);

	assign user1[20:0] = 0;

	assign analyzer3_data[0] = user1[31];
	assign analyzer3_data[1] = user1[30];
	assign analyzer3_data[2] = user1[29];
	assign analyzer3_data[3] = user1[28];
	assign analyzer3_data[4] = user1[27];
	assign analyzer3_data[5] = user1[26];
	assign analyzer3_data[6] = user1[25];
	assign analyzer3_data[7] = user1[24];
	assign analyzer3_data[8] = user1[23];
	assign analyzer3_data[9] = user1[22];
	assign analyzer3_data[10] = user1[21];
	assign analyzer3_data[15:11] = 0;*/

endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////
///////////
// Company:
// Engineer:
//
// Create Date:    15:29:00 05/11/2007
// Design Name:
```

```
// Module Name:    macro_listener
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments: This module records and play back macros.
//
//////////////////////////////////////////////////////////////////////
///////////
module macro_listener(clock, reset, start_record, stop_record,
start_play, stop_play,
                                                  actions_in,
gesture_actions_out, actions_out);

      input clock;
      input reset;

      input[7:0] actions_in;

      input start_record;
      input stop_record;
      input start_play;
      input stop_play;
      input[7:0] gesture_actions_out;
      output[7:0] actions_out;

      reg[7:0] actions_out=0;
      reg[7:0] actions_reg=0;
      reg[7:0] actions_reg_next=0;

      reg recording=0;
      reg playing=0;
      reg[10:0] play_count=0;
      reg[10:0] play_lag=0;

      reg[10:0] record_count=0;

      reg[18:0] din_rec=0;

      wire[18:0] dout_rec;
      wire[18:0] dout_play;

      reg[9:0] addr_rec=0;
      reg[9:0] addr_play=0;
      reg[9:0] addr_temp=0;

      reg we_rec;
      reg we_play;

      //latch data after address change
      reg latch;
      reg delay;
      reg copy_over;
      reg copy_delay;
      reg copy_delay_2;

      //block ram records instructions
      actionstore act_rec (
            .clka(clock),
            .dina(din_rec),
            .addra(addr_rec),
            .wea(we_rec),
            .douta(dout_rec)
                        );
      //block ram plays instructions
      actionstore act_play (
            .clka(clock),
            .dina(dout_rec),
            .addra(addr_play),
            .wea(we_play),
            .douta(dout_play)
                        );


      wire divider;
      divider div1 (
            .clock(clock),
            .reset(reset),
            .pulse(divider)
            );



      parameter STOP_CODE=19'b1111_1111_1111_1111_111;


      always @ (posedge clock)
      begin

            actions_out<=actions_reg;
            we_play<=0;
            we_rec<=0;

            //count pulses
            if(playing && divider)
            begin
                  play_count<=play_count+1;
            end

            if(recording && divider)
            begin
                  record_count<=record_count+1;
            end

            //capture button presses
            if(reset)
            begin
                  addr_rec<=0;
                  addr_play<=0;
```

```
			we_rec<=0;
			we_play<=0;
			recording<=0;
			playing<=0;

			copy_delay<=0;
			copy_delay_2<=0;

			play_count<=0;
			play_lag<=0;
			record_count<=0;
			actions_reg<=actions_in;
			actions_out<=actions_in;
		end
		//copy over instructions from record to play when record is
done
		else if(copy_delay)
		begin
			copy_delay<=0;
			addr_play<=0;
			addr_rec<=0;
		end
		else if(copy_delay_2)
		begin
			copy_delay_2<=0;
			addr_play<=0;
			addr_rec<=addr_rec+1;
		end
		else if(copy_over)
		begin
			if((dout_rec==STOP_CODE) || (addr_rec==1023))
			begin
				copy_over<=0;
				addr_play<=addr_temp;
			end
			addr_play<=addr_play+1;
			addr_rec<=addr_rec+1;
			we_play<=1;
		end
		//record macro
		else if(start_record)
		begin
			recording<=1;
			addr_rec<=0;
			record_count<=0;
		end
		else if(stop_record)
		begin
			recording<=0;
			din_rec<=STOP_CODE;
			we_rec<=1;
			copy_over<=1;
			copy_delay<=1;
			copy_delay_2<=1;
			addr_temp<=addr_play;
			addr_rec<=addr_rec+1;
		end
```

```
            //play macro
            else if(start_play)
            begin
                  actions_reg_next<=actions_reg;
                  playing<=1;
                  addr_play<=0;
                  play_lag<=0;
                  play_count<=0;
            end
            else if(stop_play)
            begin
                  playing<=0;
            end
            //end capture button presses
            else
            begin
                  // give priority to current changes. pass changes
through.
                  if(actions_in!=actions_reg)
                  begin
                        actions_reg<=actions_in;
                        // if you're recording, record the change.

                        if(recording)
                        begin
                              //din--write
                              din_rec[18:8]<=record_count[10:0];
                              din_rec[7:0]<=actions_reg[7:0];
                              addr_rec<=addr_rec+1;
                              we_rec<=1;
                              record_count[10:0]<=0;
                        end
                  end
                  else if((gesture_actions_out!=actions_reg) &&
(gesture_actions_out!=0))
                  begin
                        actions_reg<=gesture_actions_out;
                        if(recording)
                        begin
                              //din--write
                              din_rec[18:8]<=record_count[10:0];
                              din_rec[7:0]<=actions_reg[7:0];
                              addr_rec<=addr_rec+1;
                              we_rec<=1;
                              record_count[10:0]<=0;
                        end
                  end
                  else
                  begin
                  // if you're playing, check the time, pass it
through.

                        //output previous instruction
                        if(delay)
                        begin
                              delay<=0;
                              latch<=1;
                              actions_reg<=actions_reg_next;
                        end
                        //latch output of block ram
                        else if(latch)
                        begin
                              latch<=0;
                              //check to see if it is the last stop
code
                              if(dout_play==STOP_CODE)
                              begin
                                    playing<=0;
                              end
                              else
                              begin
                                    play_lag[10:0]<=dout_play[18:8];


      actions_reg_next[7:0]<=dout_play[7:0];
                                    play_count<=0;
                              end
                        end
                        else if((playing) && (play_count>=play_lag))
                        begin
                        //request next play
                              addr_play<=addr_play+1;
                              delay<=1;
                        end
                  end
            end
      end

endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////
///////////
// Company:
// Engineer:
//
// Create Date:    19:58:58 04/25/2007
// Design Name:
// Module Name:    master
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments: This module controls play and record
functionality of the zbt controllers.
//
//////////////////////////////////////////////////////////////////////
///////////
module masterzbt (reset, clk, play, record, sound_done, enable_inputs,
                              read, write, address, state, idle_s,
                              rec_data, play_data, write_data,
read_data,
                              record_stop,
                              slot, play_number, transmit, busy,
                              address_1, playing_1, final_address_1,
                              start_add, start_record_signal);

input play;
input record;
input reset;
input clk;
input sound_done;
input[35:0] rec_data;
input[35:0] read_data;
input record_stop;
input[5:0] slot;
input[1:0] play_number;
input busy;
input enable_inputs;
input start_record_signal;

output[35:0] write_data;
output[35:0] play_data;
output read;
output write;
output[18:0] address;
output[4:0] state;
output idle_s;
output transmit;
output[18:0] address_1;
output playing_1;
output[18:0] final_address_1;
output start_add;


reg play_1_done;
reg transmit;

reg start_add;

reg idle_s;

reg read;
reg write;

reg[4:0] state;

//output to zbt
reg[18:0] address;

reg[18:0] final_address_0;
reg[18:0] final_address_1;
reg[18:0] final_address_2;
reg[18:0] final_address_3;

reg[18:0] address_record;
```

```
reg[18:0] address_record_start;

reg[18:0] address_0;
reg[18:0] address_1;
reg[18:0] address_2;
reg[18:0] address_3;

reg playing_0;
reg playing_1;
reg playing_2;
reg playing_3;

reg recording;

reg[18:0] record_max;
reg[35:0] write_data;

reg[35:0] play_data;

reg[1:0] play_number_int;

reg play_serve;

parameter IDLE=0;
parameter RECORD_BEGIN=1;
parameter RECORD=2;
parameter RECORD_HOLD=3;
parameter RECORD_DONE=4;
parameter PLAY_BEGIN=5;
parameter PLAY_LOAD=6;
parameter PLAY_BEGIN_PAUSE_1=7;
parameter PLAY_BEGIN_PAUSE_2=8;
parameter PLAY_BEGIN_PAUSE_3=9;
parameter PLAY_READ_TIME=10;
parameter NEW_PLAY_WAIT=11;
parameter PLAY_ADDRESS0=12;
parameter PLAY_ADDRESS1=13;
parameter PLAY_ADDRESS2=14;
parameter PLAY_ADDRESS3=15;
parameter PLAY_HOLD=16;
parameter PLAY_DONE_CHECK=17;
parameter DONE=18;
parameter RETRIEVE_DATA0=19;
parameter RETRIEVE_DATA1=20;
parameter RETRIEVE_DATA2=21;
parameter RETRIEVE_DATA3=22;
parameter RECORD_WAIT1=23;
parameter RECORD_WAIT2=24;
parameter WAIT_TO_RECORD=25;

parameter SLOT_0=0;
parameter SLOT_0_MAX=64000;
parameter SLOT_1=64002;
parameter SLOT_1_MAX=128000;
parameter SLOT_2=128002;
parameter SLOT_2_MAX=250000;
parameter SLOT_3=250002;
parameter SLOT_3_MAX=524000;

reg play_reg;
reg record_reg;
reg record_stop_reg;

always @ (posedge clk)
begin
      read<=0;
      write<=0;
      idle_s<=0;
      transmit<=0;
      start_add<=0;

      //if you receive a play command, store relevant slot and play
number data
      if(play && (!play_serve) && enable_inputs)
      begin
            play_number_int<=play_number;
            play_serve<=1;
            if(play_number==0)
            begin
                  if(slot==0)
                  begin
                        address_0<=SLOT_0;
                  end
                  else if(slot==1)
                  begin
                        address_0<=SLOT_1;
                  end
                  else if(slot==2)
                  begin
                        address_0<=SLOT_2;
                  end
                  else if(slot==3)
                  begin
                        address_0<=SLOT_3;
                  end
            end
            else if(play_number==1)
            begin
                  if(slot==0)
                  begin
                        address_1<=SLOT_0;
                  end
                  else if(slot==1)
                  begin
                        address_1<=SLOT_1;
                  end
                  else if(slot==2)
                  begin
                        address_1<=SLOT_2;
                  end
                  else if(slot==3)
                  begin
                        address_1<=SLOT_3;
                  end
            end
            else if(play_number==2)
            begin
                  if(slot==0)
                  begin
                        address_2<=SLOT_0;
                  end
                  else if(slot==1)
                  begin
                        address_2<=SLOT_1;
                  end
                  else if(slot==2)
                  begin
                        address_2<=SLOT_2;
                  end
                  else if(slot==3)
                  begin
                        address_2<=SLOT_3;
                  end
            end
            else if(play_number==3)
            begin
                  if(slot==0)
                  begin
                        address_3<=SLOT_0;
                  end
                  else if(slot==1)
                  begin
                        address_3<=SLOT_1;
                  end
                  else if(slot==2)
                  begin
                        address_3<=SLOT_2;
                  end
                  else if(slot==3)
                  begin
                        address_3<=SLOT_3;
                  end
            end
      end
      //if you receive a record command, store record max, starting
address, and record address
      if(record && !(recording) && enable_inputs)
      begin
            recording<=1;
            if(slot==0)
            begin
                  address_record<=SLOT_0;
                  address_record_start<=SLOT_0;
                  record_max<=SLOT_0_MAX;
            end
            else if(slot==1)
            begin
                  address_record<=SLOT_1;
                  address_record_start<=SLOT_1;
                  record_max<=SLOT_1_MAX;
            end
```

```
        else if(slot==2)
        begin
              address_record<=SLOT_2;
              address_record_start<=SLOT_2;
              record_max<=SLOT_2_MAX;
        end
        else if(slot==3)
        begin
              address_record<=SLOT_3;
              address_record_start<=SLOT_3;
              record_max<=SLOT_3_MAX;
        end
  end

  //stop recording
  if(recording && record_stop && enable_inputs)
  begin
        record_stop_reg<=1;
  end



  if(reset)
  begin
        state<=IDLE;
        record_stop_reg<=0;
        address_0<=0;
        address_1<=0;
        address_2<=0;
        address_3<=0;
        final_address_0<=0;
        final_address_1<=0;
        final_address_2<=0;
        final_address_3<=0;
        recording<=0;
        play_serve<=0;
        playing_0<=0;
        playing_1<=0;
        playing_2<=0;
        playing_3<=0;
  end
  else
  begin
        case(state)
              //first play, then record
              IDLE:
              begin
                    if(sound_done)
                    begin
                          state<=PLAY_BEGIN;
                    end
                    else
                    begin
                          state<=IDLE;
                    end
              end
              WAIT_TO_RECORD:
                  begin
                        if(start_record_signal)
                        begin
                              state<=RECORD_HOLD;
                        end
                        else
                        begin
                              state<=WAIT_TO_RECORD;
                        end
                  end

                  RECORD_HOLD:
                  begin
                        if(recording==0)
                        begin
                              state<=DONE;
                        end
                        //you are done recording!
                        else if(record_stop_reg ||
(address_record>record_max))
                        begin
                              state<=RECORD_DONE;
                              recording<=0;
                        end
                        //time to record
                        else
                        begin
                              state<=RECORD;
                        end
                  end
                  RECORD:
                  begin
                        write<=1'b1;
                        address_record<=address_record+1;
                        address<=address_record;
                        write_data<=rec_data;
                        state<=RECORD_WAIT1;
                  end
                  RECORD_DONE:
                  begin
                        write<=1'b1;
                        address<=address_record_start;
                        record_stop_reg<=0;
                        recording<=0;
                        write_data<=address;
                        state<=RECORD_WAIT1;
                  end
                  RECORD_WAIT1:
                  begin
                        if(!busy)
                        begin
                              state<=RECORD_WAIT1;
                        end
                        else
                        begin
                              state<=RECORD_WAIT2;
                        end
```

```
end
RECORD_WAIT2:
begin
      if(!busy)
      begin
            state<=DONE;
      end
      else
      begin
            state<=RECORD_WAIT2;
      end
end
//see if you have a new play request to service
PLAY_BEGIN:
begin
      if(play_serve)
      begin
            state<=PLAY_LOAD;
            play_serve<=0;
      end
      else
      begin
            state<=PLAY_ADDRESS0;
      end
end
// load new play
PLAY_LOAD:
begin
      if(play_number_int==0)
      begin
            address<=address_0;
            address_0<=address_0+1;
      end
      else if(play_number_int==1)
      begin
            address<=address_1;
            address_1<=address_1+1;

      end
      else if(play_number_int==2)
      begin
            address<=address_2;
            address_2<=address_2+1;

      end
      else if(play_number_int==3)
      begin
            address<=address_3;
            address_3<=address_3+1;

      end
      read<=1'b1;
      state<=PLAY_BEGIN_PAUSE_1;
end
PLAY_BEGIN_PAUSE_1:
begin
      state<=PLAY_BEGIN_PAUSE_2;
```

```
end
PLAY_BEGIN_PAUSE_2:
begin
      state<=PLAY_BEGIN_PAUSE_3;
end
PLAY_BEGIN_PAUSE_3:
begin
      state<=PLAY_READ_TIME;
end
//see how long the play lasts for
PLAY_READ_TIME:
begin
      play_serve<=0;
      if(play_number_int==0)
      begin
            playing_0<=1;
            final_address_0<=read_data[18:0];
      end
      else if(play_number_int==1)
      begin
            playing_1<=1;
            final_address_1<=read_data[18:0];
      end
      else if(play_number_int==2)
      begin
            playing_2<=1;
            final_address_2<=read_data[18:0];
      end
      else if(play_number_int==3)
      begin
            playing_3<=1;
            final_address_3<=read_data[18:0];
      end
      state<=NEW_PLAY_WAIT;

end
//wait till ram is no longer busy
NEW_PLAY_WAIT:
begin
      if(!busy)
      begin
            state<=PLAY_ADDRESS0;
      end
      else
      begin
            state<=NEW_PLAY_WAIT;
      end
end
//play addresses, increment address
PLAY_ADDRESS0:
begin
      read<=1;
      if((playing_0) && (address_0<final_address_0))
      begin
            address<=address_0;
            address_0<=address_0+1;
            state<=PLAY_ADDRESS1;
      end
      else
      begin
            playing_0<=0;
            state<=PLAY_ADDRESS1;
      end
end
PLAY_ADDRESS1:
begin
      if((playing_1) && (address_1<final_address_1))
      begin
            address<=address_1;
            address_1<=address_1+1;
            state<=PLAY_ADDRESS2;
      end
      else
      begin
            playing_1<=0;
            state<=PLAY_ADDRESS2;
      end
end
PLAY_ADDRESS2:
begin
      if((playing_2) && (address_2<final_address_2))
      begin
            address<=address_2;
            address_2<=address_2+1;
            state<=PLAY_ADDRESS3;
      end
      else
      begin
            playing_2<=0;
            state<=PLAY_ADDRESS3;
      end
end
PLAY_ADDRESS3:
begin
      if((playing_3) && (address_3<final_address_3))
      begin
            address<=address_3;
            address_3<=address_3+1;
            state<=RETRIEVE_DATA0;
      end
      else
      begin
            playing_3<=0;
            state<=RETRIEVE_DATA0;
      end
end
// feed data to signal combiner
RETRIEVE_DATA0:
begin
      start_add<=1;
      if(playing_0)
      begin
            play_data<=read_data;
            state<=RETRIEVE_DATA1;
```

```
		end
		else
		begin
			play_data<=0;
			state<=RETRIEVE_DATA1;

		end
	end
	RETRIEVE_DATA1:
	begin
		if(playing_1)
		begin
			play_data<=read_data;
			state<=RETRIEVE_DATA2;
		end
		else
		begin
			play_data<=0;
			state<=RETRIEVE_DATA2;

		end
	end
	RETRIEVE_DATA2:
	begin
		if(playing_2)
		begin
			play_data<=read_data;
			state<=RETRIEVE_DATA3;
		end
		else
		begin
			play_data<=0;
			state<=RETRIEVE_DATA3;

		end
	end
	RETRIEVE_DATA3:
	begin
		if(playing_3)
		begin
			play_data<=read_data;
			state<=WAIT_TO_RECORD;
		end
		else
		begin
			play_data<=0;
			state<=WAIT_TO_RECORD;

		end
	end


	DONE:
	begin
		transmit<=1;
		state<=IDLE;
		play_data<=0;
```

```
                  end

                  default:
                  begin
                        state<=IDLE;
                  end
            endcase
      end
end



endmodule

`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////
///////////
//
// Master FSM
//
// This unit coordinates the operation of the subunits processing
accelerator
// input and gyroscope input and processes the respective results from
those
// subunits to provide a final gesture code indicating which
controlling motion
// has been performed.  Specifically, the unit receives as input an
action code
// from the accelerator processing subunit and an action code from the
gyroscope
// processing unit, and then determines a final gesture code based on
the two
// input action codes.
//
// In the current implementation, it senses motion of each hand along
the x and
// the y axes, both independently and together, and rotation of each
hand around
// the y axis, both independently and together.
//
// Code | Accel | Gyro | Description
// -----|-------|------|--------------------------------------------------
---------
// 0    | 0     | 0    | Nothing
// 1    | 9,10  | 0    | Volume up
// 2    | 11,12 | 0    | Volume down
// 3    | 1,2   | 0    | Low-pass filter enable/disable
// 4    | 3,4   | 0    | High-pass filter enable/disable
// 5    | 0     | 1    | Play Clip 0
// 6    | 0     | 2    | Play Clip 1
// 7    | 0     | 3    | Play Clip 2
// 8    | 0     | 4    | Play Clip 3
// 9    | 0     | 5    | Play Clip 4
// 10   | 0     | 6    | Play Clip 5
// 11   | 0     | 7    | Play Clip 6
// 12   | 0     | 8    | Play Clip 7
// 13   | 5,6,  | 0    | Increment play slot
//      | 7,8   |      |
// 14   | 13,14 | 2    | Nothing
//      | 15,16 |      |
//
//////////////////////////////////////////////////////////////////////
///////////
module master_fsm(clock, reset_sync, enable, accel_busy, gyro_busy,
accel_code, gyro_code,
             gyro_request, accel_request, gesture_code, state,
int_accel_code, int_gyro_code);
    input clock;
    input reset_sync;
    input enable;
    input accel_busy;
    input gyro_busy;
    input [4:0] accel_code;
    input [3:0] gyro_code;
    output reg gyro_request;
       output reg accel_request;
    output [4:0] gesture_code;

       output reg [3:0] state;
       reg [3:0] next;
       output reg [4:0] int_accel_code;
       output reg [3:0] int_gyro_code;
       reg [4:0] gesture_code = 0;

       //variables indicating whether to set/reset the internal accel
and gyro codes
       reg set_accel;
       reg reset_accel;
       reg set_gyro;
       reg reset_gyro;

       //states
       parameter IDLE = 0;
       parameter GyroRequest = 1;
       parameter GyroPause = 2;
       parameter GyroPause2 = 3;
       parameter AccelRequest = 4;
       parameter AccelPause = 5;
       parameter AccelPause2 = 6;
       parameter Decode = 7;

//state assignment
always @ (posedge clock) begin
      if (reset_sync) state <= IDLE;
      else begin
            state <= next;

            if(reset_accel) int_accel_code <= 0;
            else if(set_accel) int_accel_code <= accel_code;
            else int_accel_code <= int_accel_code;

            if(reset_gyro) int_gyro_code <= 0;
            else if(set_gyro) int_gyro_code <= gyro_code;
            else int_gyro_code <= int_gyro_code;
      end
end

//next state computation
always @ (state or enable or int_accel_code or int_gyro_code or
gyro_code or accel_code or gyro_busy or accel_busy) begin
      gyro_request = 0;
      accel_request = 0;
      set_accel = 0;
      reset_accel = 1;
      set_gyro = 0;
      reset_gyro = 1;

      case (state)
            IDLE:
                  begin
                        if (enable) next = GyroRequest;
                        else next = IDLE;
                  end

            GyroRequest:
                  begin
                        next = GyroPause;

                        gyro_request = 1; // request gyro code
                  end

            GyroPause:
                  begin
                        next = GyroPause2;

                        set_gyro = 1; // capture gyro code value
                        reset_gyro = 0;
                  end

            GyroPause2: // wait for gyro decoder to finish
                  begin
                        if (!gyro_busy) begin
                              if (int_gyro_code != 0) next = Decode;
                              else next = AccelRequest;
                        end else next = GyroPause2;

                        reset_gyro = 0;
                  end

            AccelRequest:
                  begin
                        next = AccelPause;

                        accel_request = 1; // request accel code
                  end

            AccelPause:
                  begin
                        next = AccelPause2;
                        set_accel = 1; // capture accel code value
                        reset_accel = 0;
                  end

            AccelPause2: // wait for accel decoder to finish
                  begin
                        if (!accel_busy) next = Decode;
                        else next = AccelPause2;

                        reset_accel = 0;
                  end

            Decode:
                  begin
                        next = IDLE;

                        if ((int_accel_code == 9) || (int_accel_code ==
10)) gesture_code = 1;
                        else if ((int_accel_code == 11) ||
(int_accel_code == 12)) gesture_code = 2;
                        else if ((int_accel_code == 1) ||
(int_accel_code == 2)) gesture_code = 3;
                        else if ((int_accel_code == 4) ||
(int_accel_code == 3)) gesture_code = 4;
                        else if ((int_accel_code == 5) ||
(int_accel_code == 6) ||
                                          (int_accel_code == 7) ||
(int_accel_code == 8)) gesture_code = 13;
                        else if ((int_accel_code == 13) ||
(int_accel_code == 14) ||
                                          (int_accel_code == 15) ||
(int_accel_code == 16)) gesture_code = 14;
                        else if (int_accel_code == 0) begin
                              if (int_gyro_code == 1) gesture_code = 5;
                              else if (int_gyro_code == 2) gesture_code
= 6;
                              else if (int_gyro_code == 3) gesture_code
= 7;
                              else if (int_gyro_code == 4) gesture_code
= 8;
                              else if (int_gyro_code == 5) gesture_code
= 9;
                              else if (int_gyro_code == 6) gesture_code
= 10;
                              else if (int_gyro_code == 7) gesture_code
= 11;
                              else if (int_gyro_code == 8) gesture_code
= 12;
                              else gesture_code = 0;
                        end else gesture_code = 0;
                  end

            default:
                  begin
                        next = IDLE;
                  end
```

```
      endcase
end

endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////
///////////
// Company:
// Engineer:
//
// Create Date:    20:41:30 05/01/2007
// Design Name:
// Module Name:    mastercontroller
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments: This module synchronizes the ac97 controller
with the other modules.
//
//////////////////////////////////////////////////////////////////////
///////////
module mastercontroller(reset, clk, play, record, record_stop,
                                                play_pulse,
record_pulse, record_stop_pulse,
                                                sound_done,
sound_done_pulse, sound_done_pulse_reg, sound_done_pulse1,
                                                transmit,
                                                rec_slot, rec_slot_in,
play_number, play_number_in,
                                                state, enter,
filter_selector, filter_enable, volume_selector, volume_enable);
      input reset;
      input clk;
      input play;
      input record;
      input record_stop;
      input sound_done;
      input transmit;
      input[2:0] rec_slot;
      input[1:0] play_number;
      input filter_selector;
      input enter;
      input[1:0] volume_selector;

      output play_pulse;
      output record_pulse;
      output record_stop_pulse;
      output sound_done_pulse;
      output sound_done_pulse1;
      output sound_done_pulse_reg;
```

```
output[2:0] rec_slot_in;
output[1:0] play_number_in;
output[1:0] state;
output[1:0] filter_enable;
output[1:0] volume_enable;

reg enter_reg;

reg[1:0] filter_enable;
reg[1:0] volume_enable;

reg[2:0] rec_slot_in;
reg[1:0] play_number_in;

reg play_pulse, play_pulse_reg;
reg record_pulse, record_pulse_reg;
reg record_stop_pulse, record_stop_pulse_reg;
reg sound_done_pulse, sound_done_pulse1, sound_done_pulse_reg;
reg sound_done_pulse_reg_2;

reg[1:0] state;

reg[6:0] pause;

parameter IDLE=0;
parameter PLAY_REC_STOP=1;
parameter SOUND_DONE_STATE_0=2;
parameter SOUND_DONE_STATE_1=3;


always @ (posedge clk)
begin
      play_pulse<=0;
      record_pulse<=0;
      record_stop_pulse<=0;
      sound_done_pulse<=0;
      sound_done_pulse1<=0;
      rec_slot_in<=0;
      play_number_in<=0;
      filter_enable<=0;
      volume_enable<=0;

      //synchronize sound_done from ac97
      if(sound_done)
      begin
            sound_done_pulse_reg<=1;
      end

      if(sound_done_pulse_reg && !(sound_done))
      begin
            sound_done_pulse_reg_2<=1;
      end

      if(play)
      begin
            play_pulse_reg<=1;
      end
```

```
if(record)
begin
      record_pulse_reg<=1;
end

if(record_stop)
begin
      record_stop_pulse_reg<=1;
end

if(enter)
begin
      enter_reg<=1;
end

//synchronize/pulse sound_done
if(reset)
begin
      state<=SOUND_DONE_STATE_0;
      sound_done_pulse_reg<=0;
      play_pulse_reg<=0;
      record_pulse_reg<=0;
      record_stop_pulse_reg<=0;
      pause<=0;
end
else
begin
      case(state)
            IDLE:
            begin
            // wait till play/rec modules are done
                  if(transmit)
                  begin
                        state<=PLAY_REC_STOP;
                  end
            end
            //send new instructions
            PLAY_REC_STOP:
            begin
                  state<=SOUND_DONE_STATE_0;
                  if(play_pulse_reg)
                  begin
                        rec_slot_in<=rec_slot;
                        play_number_in<=play_number;

                        play_pulse<=1;
                        play_pulse_reg<=0;
                  end
                  else if(record_pulse_reg)
                  begin
                        rec_slot_in<=rec_slot;
                        play_number_in<=play_number;

                        record_pulse<=1;
                        record_pulse_reg<=0;
                  end
```

```
                        else if(record_stop_pulse_reg)
                        begin
                              rec_slot_in<=rec_slot;
                              play_number_in<=play_number;

                              record_stop_pulse<=1;
                              record_stop_pulse_reg<=0;

                        end
                        else if(enter_reg)
                        begin
                              enter_reg<=0;
                              if(volume_selector==2'b10)
                              begin
                                    volume_enable<=2'b10;
                              end
                              else if(volume_selector==2'b01)
                              begin
                                    volume_enable<=2'b01;
                              end
                              else if(filter_selector)
                              begin
                                    filter_enable<=2'b10;
                              end
                              else
                              begin
                                    filter_enable<=2'b01;
                              end
                        end
                  end
                  //wait for sound_done to activate play/rec
module 0
                  SOUND_DONE_STATE_0:
                  begin
                        if(sound_done_pulse_reg_2)
                        begin
                              sound_done_pulse<=1;
                              sound_done_pulse_reg<=0;
                              sound_done_pulse_reg_2<=0;
                              state<=SOUND_DONE_STATE_1;
                              pause<=0;
                        end
                        else
                        begin
                              state<=SOUND_DONE_STATE_0;
                        end
                  end
                  //wait 30 additional cycles to activate
play/rec module 1
                  SOUND_DONE_STATE_1:
                  begin
                        pause<=pause+1;
                        if(pause==30)
                        begin
                              pause<=0;
                              sound_done_pulse1<=1;
                              state<=IDLE;
                              end
                              else
                              begin
                                    state<=SOUND_DONE_STATE_1;
                              end
                        end

                  endcase
            end
      end


endmodule

`timescale 1ns / 1ps

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

// System Tester
//
// This unit instantiates the master FSM, the decoders and processors
for all
// sensors, as well as needed dividers, and connects them.  This is
used
// for testing the operation of the system as a whole using simulation.
//

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

module system_tester(clock, reset_sync, left_accel_x, left_accel_y,
left_accel_z, right_accel_x,

             right_accel_y, right_accel_z, right_rotate, left_rotate,
ADbusy, convst_b, rd_b, gesture_code,

             accel_code, gyro_code, state);

      input clock;

      input reset_sync;

      input [7:0] left_accel_x;
   input [7:0] left_accel_y;
   input [7:0] left_accel_z;
   input [7:0] right_accel_x;
   input [7:0] right_accel_y;
   input [7:0] right_accel_z;
      input [7:0] right_rotate;
      input [7:0] left_rotate;

      input ADbusy;

      output convst_b;

      output rd_b;
      output [4:0] gesture_code;

      output [4:0] accel_code;

      output [3:0] gyro_code;

      output [3:0] state;



//connect wires

wire fsm_enable, enable;
wire accel_busy, gyro_busy;
wire accel_request, gyro_request;
wire [4:0] accel_code;
wire [3:0] gyro_code;

wire [7:0] left_accel_x_out, left_accel_y_out, left_accel_z_out;

wire [7:0] right_accel_x_out, right_accel_y_out, right_accel_z_out;

wire [16:0] left_rotate_out, right_rotate_out;

wire [7:0] left_val_x, left_read_val_x;

wire [15:0] left_sum_x;

wire [3:0] accel_state;

wire [2:0] gyro_state;




//instantiate master_fsm

master_fsm master (
      .clock(clock),
      .reset_sync(reset_sync),
      .enable(fsm_enable),
      .accel_busy(accel_busy),
      .gyro_busy(gyro_busy),
      .accel_code(accel_code),
      .gyro_code(gyro_code),
      .gyro_request(gyro_request),
      .accel_request(accel_request),
      .gesture_code(gesture_code),

      .state(state),

      .int_accel_code(int_accel_code),

      .int_gyro_code(int_gyro_code)
);
```

```
//instantiate accel_num

accel_num accel_num (
      .clock(clock),
      .reset_sync(reset_sync),
      .enable(enable),
      .left_accel_x(left_accel_x),
      .left_accel_y(left_accel_y),
      .left_accel_z(left_accel_z),
      .right_accel_x(right_accel_x),
      .right_accel_y(right_accel_y),
      .right_accel_z(right_accel_z),
      .ADbusy(ADbusy),
      .convst_b(convst_b_accel),
      .rd_b(rd_b_accel),
      .left_accel_x_out(left_accel_x_out),
      .left_accel_y_out(left_accel_y_out),
      .left_accel_z_out(left_accel_z_out),
      .right_accel_x_out(right_accel_x_out),
      .right_accel_y_out(right_accel_y_out),
      .right_accel_z_out(right_accel_z_out),

      .left_val_x(left_val_x),

      .left_read_val_x(left_read_val_x),
      .left_sum_x(left_sum_x),
      .state(accel_state)
);



//instantiate accel_decoder

accel_decoder accel_decoder (
      .clock(clock),
      .reset_sync(reset_sync),
      .left_accel_x(left_accel_x_out),
      .left_accel_y(left_accel_y_out),
      .left_accel_z(left_accel_z_out),
      .right_accel_x(right_accel_x_out),
      .right_accel_y(right_accel_y_out),
      .right_accel_z(right_accel_z_out),
      .request(accel_request),
      .busy(accel_busy),
      .accel_code(accel_code)
);



//instantiate gyro_processor

gyro_processor gyro_processor (
      .clock(clock),
      .reset_sync(reset_sync),
      .enable(enable),
      .left_rotate(left_rotate),
      .right_rotate(right_rotate),
      .ADbusy(ADbusy),
      .convst_b(convst_b_gyro),
      .rd_b(rd_b_gyro),
      .left_rotate_out(left_rotate_out),
      .right_rotate_out(right_rotate_out),
      .state(gyro_state)
);


//instantiate gyro_decoder

gyro_decoder gyro_decoder (
      .clock(clock),
      .reset_sync(reset_sync),
      .left_rotate(left_rotate_out),
      .right_rotate(right_rotate_out),
      .request(gyro_request),
      .busy(gyro_busy),
      .gyro_code(gyro_code)
);



// Instantiate fsm_divider
divider fsm_divider (
      .clock(clock),
      .reset(reset_sync),
      .pulse(fsm_enable)
);



//instantiate adc divider
adc_divider adc_divider(
      .clock(clock),
      .reset_sync(reset_sync),
      .enable(enable)
);



endmodule


`timescale 1ns / 1ps



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

// Company:

// Engineer:
```

```
//

// Create Date:   15:40:42 05/02/2007

// Design Name:   accel_decoder

// Module Name:   C:/chukl/XtremiX/tb_accel_decoder.v

// Project Name:  XtremiX

// Target Device:

// Tool versions:

// Description:

//

// Verilog Test Fixture created by ISE for module: accel_decoder

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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


module tb_accel_decoder_v;


      // Inputs

      reg clock;

      reg reset_sync;

      reg [7:0] left_accel_x;

      reg [7:0] left_accel_y;

      reg [7:0] left_accel_z;

      reg [7:0] right_accel_x;
      reg [7:0] right_accel_y;

      reg [7:0] right_accel_z;

      reg request;


      // Outputs

      wire busy;

      wire [4:0] accel_code;


      // Instantiate the Unit Under Test (UUT)

      accel_decoder uut (

            .clock(clock),

            .reset_sync(reset_sync),

            .left_accel_x(left_accel_x),

            .left_accel_y(left_accel_y),

            .left_accel_z(left_accel_z),

            .right_accel_x(right_accel_x),

            .right_accel_y(right_accel_y),

            .right_accel_z(right_accel_z),

            .request(request),

            .busy(busy),

            .accel_code(accel_code)

      );


      always #10 clock = ~clock;


      initial begin

            // Initialize Inputs

            clock = 0;
```

```
reset_sync = 1;

left_accel_x = 5;

left_accel_y = 5;

left_accel_z = 5;

right_accel_x = 5;

right_accel_y = 5;

right_accel_z = 5;

request = 0;



#70;
reset_sync = 0;
#40;
right_accel_x = 12;
#20;
right_accel_x = 5;
#40;
request = 1;
#40;
right_accel_x = 7;
#40;
right_accel_x = 5;
#40;
right_accel_x = 3;
#40;
right_accel_x = 5;
#40;
left_accel_x = 7;
#40;
left_accel_x = 5;
#40;
left_accel_x = 3;
#40;
left_accel_x = 5;
#40;
right_accel_x = 7;
left_accel_x = 7;
#40;
right_accel_x = 5;
left_accel_x = 5;
#40;
right_accel_x = 3;
left_accel_x = 3;
#40;
right_accel_x = 5;
left_accel_x = 5;
#40;
right_accel_x = 7;
left_accel_x = 3;
```

```
#40;
right_accel_x = 5;
left_accel_x = 5;
#40;
right_accel_x = 3;
left_accel_x = 7;
#40;
right_accel_x = 5;
left_accel_x = 5;
#40;
reset_sync = 1;

#40;

request = 0;



#30;
reset_sync = 0;
#40;
right_accel_y = 12;
#20;
right_accel_y = 5;
#40;
request = 1;
#40;
right_accel_y = 7;
#40;
right_accel_y = 5;
#40;
right_accel_y = 3;
#40;
right_accel_y = 5;
#40;
left_accel_y = 7;
#40;
left_accel_y = 5;
#40;
left_accel_y = 3;
#40;
left_accel_y = 5;
#40;
right_accel_y = 7;
left_accel_y = 7;
#40;
right_accel_y = 5;
left_accel_y = 5;
#40;
right_accel_y = 3;
left_accel_y = 3;
#40;
right_accel_y = 5;
left_accel_y = 5;
#40;
right_accel_y = 7;
left_accel_y = 3;
```

```
		#40;
		right_accel_y = 5;
		left_accel_y = 5;
		#40;
		right_accel_y = 3;
		left_accel_y = 7;
		#40;
		right_accel_y = 5;
		left_accel_y = 5;
		#40;

		reset_sync = 1;



	end



endmodule





`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////
/////////
// Company:
// Engineer:
//
// Create Date:   17:55:58 05/03/2007
// Design Name:   accel_num
// Module Name:   C:/chukl/XtremiX/tb_accel_num.v
// Project Name:  XtremiX
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: accel_num
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////
/////////

module tb_accel_num_v;

	// Inputs
	reg clock;
	reg reset_sync;
	reg enable;
	reg [7:0] left_accel_x;
reg [7:0] left_accel_y;
reg [7:0] left_accel_z;
reg [7:0] right_accel_x;
reg [7:0] right_accel_y;
reg [7:0] right_accel_z;
reg ADbusy;

// Outputs
wire convst_b;
wire rd_b;
wire [7:0] left_accel_x_out;
wire [7:0] left_accel_y_out;
wire [7:0] left_accel_z_out;
wire [7:0] right_accel_x_out;
wire [7:0] right_accel_y_out;
wire [7:0] right_accel_z_out;
wire [7:0] left_read_val_x;
wire [15:0] left_sum_x;
wire [3:0] state;

// Instantiate the Unit Under Test (UUT)
accel_num uut (
      .clock(clock),
      .reset_sync(reset_sync),
      .enable(enable),
      .left_accel_x(left_accel_x),
      .left_accel_y(left_accel_y),
      .left_accel_z(left_accel_z),
      .right_accel_x(right_accel_x),
      .right_accel_y(right_accel_y),
      .right_accel_z(right_accel_z),
      .ADbusy(ADbusy),
      .convst_b(convst_b),
      .rd_b(rd_b),
      .left_accel_x_out(left_accel_x_out),
      .left_accel_y_out(left_accel_y_out),
      .left_accel_z_out(left_accel_z_out),
      .right_accel_x_out(right_accel_x_out),
      .right_accel_y_out(right_accel_y_out),
      .right_accel_z_out(right_accel_z_out),
      .left_read_val_x(left_read_val_x),
      .left_sum_x(left_sum_x),
      .state(state)
);

always #10 clock = ~clock;
always begin
      #80;
      enable = ~enable;
      #20;
      enable = ~enable;
end

initial begin
      // Initialize Inputs
      clock = 0;
      reset_sync = 1;
```

```
            enable = 0;
            left_accel_x = 80;
            left_accel_y = 63;
            left_accel_z = 0;
            right_accel_x = 80;
            right_accel_y = 63;
            right_accel_z = 0;
            ADbusy = 0;

            #100;
            reset_sync = 0;
            #200;
            left_accel_x = 81;
            left_accel_y = 64;
            left_accel_z = 3;
            right_accel_x = 81;
            right_accel_y = 64;
            right_accel_z = 3;
            #500;
            ADbusy = 1;
            #60;
            ADbusy = 0;
            #40;
            #600;
            reset_sync = 1;
            #30;
            reset_sync = 0;



      end

endmodule

`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////
/////////
// Company:
// Engineer:
//
// Create Date:   16:55:18 05/02/2007
// Design Name:   fsm_tester
// Module Name:   C:/chukl/XtremiX/tb_fsm_tester.v
// Project Name:  XtremiX
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: fsm_tester
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
```

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

module tb_fsm_tester_v;

      // Inputs
      reg clock;
      reg reset_sync;
      reg [7:0] left_accel_x;
      reg [7:0] left_accel_y;
      reg [7:0] left_accel_z;
      reg [7:0] right_accel_x;
      reg [7:0] right_accel_y;
      reg [7:0] right_accel_z;
      reg [16:0] right_rotate;
      reg [16:0] left_rotate;

      //Outputs
      wire [4:0] gesture_code;

      // Instantiate the Unit Under Test (UUT)
      fsm_tester uut (
            .clock(clock),
            .reset_sync(reset_sync),
            .left_accel_x(left_accel_x),
            .left_accel_y(left_accel_y),
            .left_accel_z(left_accel_z),
            .right_accel_x(right_accel_x),
            .right_accel_y(right_accel_y),
            .right_accel_z(right_accel_z),
            .right_rotate(right_rotate),
            .left_rotate(left_rotate),
            .gesture_code(gesture_code)
      );

      always #10 clock = ~clock;

      initial begin
            // Initialize Inputs
            clock = 0;
            reset_sync = 1;
            left_accel_x = 80;
            left_accel_y = 63;
            left_accel_z = 0;
            right_accel_x = 80;
            right_accel_y = 63;
            right_accel_z = 0;
            right_rotate = 0;
            left_rotate = 0;

            #100;
            reset_sync = 0;
            #50;
            left_accel_x = 70;
      #50;
            left_accel_x = 80;
            #50;
            right_accel_x = 90;
            #150;
            right_accel_x = 80;
            #50;
            left_accel_y = 90;
            right_accel_y = 90;
            #200;
            left_accel_y = 63;
            right_accel_y = 63;
            #50;
            left_accel_y = 50;
            right_accel_y = 50;
            #150;
            left_accel_y = 63;
            right_accel_y = 63;
            #50;
            right_rotate = 20;
            #200;
            right_rotate = 0;
            #50;
            left_rotate = -20;
            #150;
            left_rotate = 0;
            #50;
            right_rotate = 20;
            left_rotate = -20;
            #200;
            right_rotate = 0;
            left_rotate = 0;
            #50;


      end

endmodule

`timescale 1ns / 1ps

///////////////////////////////////////////////////////////////////////
/////////
// Company:
// Engineer:
//
// Create Date:   15:05:19 05/02/2007
// Design Name:   gyro_decoder
// Module Name:   C:/chukl/XtremiX/tb_gyro_decoder.v
// Project Name:  XtremiX
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: gyro_decoder
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////
/////////

module tb_gyro_decoder_v;

      // Inputs
      reg clock;
      reg reset_sync;
      reg [16:0] left_rotate;
      reg [16:0] right_rotate;
      reg request;

      // Outputs
      wire busy;
      wire [3:0] gyro_code;

      // Instantiate the Unit Under Test (UUT)
      gyro_decoder uut (
            .clock(clock),
            .reset_sync(reset_sync),
            .left_rotate(left_rotate),
            .right_rotate(right_rotate),
            .request(request),
            .busy(busy),
            .gyro_code(gyro_code)
      );

      always #10 clock = ~clock;

      initial begin
            // Initialize Inputs
            clock = 0;
            reset_sync = 1;
            left_rotate = 0;
            right_rotate = 0;
            request = 0;

            #70;
            reset_sync = 0;
            #40;
            right_rotate = 12;
            #20;
            right_rotate = 0;
            #40;
            request = 1;
            #40;
            right_rotate = 15;
            #50;
            right_rotate = 0;
            #50;
            right_rotate = -15;
            #50;
            right_rotate = 0;
            #50;
            left_rotate = 15;
            #50;
            left_rotate = 0;
            #50;
            left_rotate = -31;
            #50;
            left_rotate = 0;
            #50;
            right_rotate = 15;
            left_rotate = 15;
            #50;
            right_rotate = 0;
            left_rotate = 0;
            #50;
            right_rotate = -15;
            left_rotate = -15;
            #50;
            right_rotate = 0;
            left_rotate = 0;
            #50;
            right_rotate = 15;
            left_rotate = -15;
            #50;
            right_rotate = 0;
            left_rotate = 0;
            #50;
            right_rotate = -15;
            left_rotate = 15;
            #50;
            right_rotate = 0;
            left_rotate = 0;
            #50;
            reset_sync = 1;

      end

endmodule

`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////
/////////
// Company:
// Engineer:
//
// Create Date:   17:57:20 05/11/2007
// Design Name:   gyro_processor
// Module Name:   C:/chukl/XtremiX/tb_gyro_processor.v
// Project Name:  XtremiX
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: gyro_processor
//
// Dependencies:
//
// Revision:
```

```
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////
/////////

module tb_gyro_processor_v;

      // Inputs
      reg clock;
      reg reset_sync;
      reg enable;
      reg [7:0] left_rotate;
      reg [7:0] right_rotate;
      reg ADbusy;

      // Outputs
      wire convst_b;
      wire rd_b;
      wire [16:0] left_rotate_out;
      wire [16:0] right_rotate_out;
      wire [2:0] state;

      // Instantiate the Unit Under Test (UUT)
      gyro_processor uut (
            .clock(clock),
            .reset_sync(reset_sync),
            .enable(enable),
            .left_rotate(left_rotate),
            .right_rotate(right_rotate),
            .ADbusy(ADbusy),
            .convst_b(convst_b),
            .rd_b(rd_b),
            .left_rotate_out(left_rotate_out),
            .right_rotate_out(right_rotate_out),
            .state(state)
      );

      always #10 clock = ~clock;
      always begin
            #80;
            enable = ~enable;
            #20;
            enable = ~enable;
      end

      initial begin
            // Initialize Inputs
            clock = 0;
            reset_sync = 1;
            enable = 0;
            left_rotate = 128;
            right_rotate = 128;
            ADbusy = 0;

            #100;
            reset_sync = 0;
            #200;
            left_rotate = 200;
            right_rotate = 200;
            #500;
            left_rotate = 100;
            right_rotate = 100;
            #500;
            reset_sync = 1;
            #50;
            left_rotate = 128;
            right_rotate = 128;
            #50;
            reset_sync = 0;

      end

endmodule

`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////
/////////
// Company:
// Engineer:
//
// Create Date:   14:54:46 05/02/2007
// Design Name:   master_fsm
// Module Name:   C:/chukl/XtremiX/tb_master_fsm.v
// Project Name:  XtremiX
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: master_fsm
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////
/////////

module tb_master_fsm_v;

      // Inputs
      reg clock;
      reg reset_sync;
      reg enable;
      reg accel_busy;
      reg gyro_busy;
      reg [4:0] accel_code;
      reg [3:0] gyro_code;

      // Outputs
      wire gyro_request;
      wire accel_request;
      wire [4:0] gesture_code;

      // Instantiate the Unit Under Test (UUT)
      master_fsm uut (
            .clock(clock),
            .reset_sync(reset_sync),
            .enable(enable),
            .accel_busy(accel_busy),
            .gyro_busy(gyro_busy),
            .accel_code(accel_code),
            .gyro_code(gyro_code),
            .gyro_request(gyro_request),
            .accel_request(accel_request),
            .gesture_code(gesture_code)
      );

      always #10 clock = ~clock;
      always begin
            #20;
            enable = ~enable;
            #40;
            enable = ~enable;
      end

      initial begin
            // Initialize Inputs
            clock = 0;
            reset_sync = 1;
            enable = 0;
            accel_busy = 0;
            gyro_busy = 0;
            accel_code = 0;
            gyro_code = 0;

            #40;
            reset_sync = 0;
            #130;
            gyro_code = 1;
            #120;
            gyro_code = 0;
            #120;
            gyro_code = 2;
            #120;
            gyro_code = 3;
            #120;
            gyro_code = 4;
            #120;
            gyro_code = 5;
            #120;
            gyro_code = 6;
            #120;
            gyro_code = 7;
            #120;
            gyro_code = 8;
            #120;
            gyro_code = 0;
```

```
            #120;
            accel_code = 1;
            #120;
            accel_code = 0;
            #120;
            accel_code = 4;
            #120;
            accel_code = 13;
            #120;
            accel_code = 14;
            #120;
            accel_code = 0;
            #120;
            reset_sync = 1;

      end

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////
/////////
// Company:
// Engineer:
//
// Create Date:   15:36:32 05/12/2007
// Design Name:   system_tester
// Module Name:   C:/chukl/XtremiX/tb_system_tester.v
// Project Name:  XtremiX
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: system_tester
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////
/////////

module tb_system_tester_v;

      // Inputs
      reg clock;
      reg reset_sync;
      reg [7:0] left_accel_x;
      reg [7:0] left_accel_y;
      reg [7:0] left_accel_z;
      reg [7:0] right_accel_x;
      reg [7:0] right_accel_y;
      reg [7:0] right_accel_z;
      reg [7:0] right_rotate;
      reg [7:0] left_rotate;
      reg ADbusy;

      // Outputs
      wire convst_b;
      wire rd_b;
      wire [4:0] gesture_code;
      wire [4:0] accel_code;
      wire [3:0] gyro_code;
      wire [3:0] state;

      // Instantiate the Unit Under Test (UUT)
      system_tester uut (
            .clock(clock),
            .reset_sync(reset_sync),
            .left_accel_x(left_accel_x),
            .left_accel_y(left_accel_y),
            .left_accel_z(left_accel_z),
            .right_accel_x(right_accel_x),
            .right_accel_y(right_accel_y),
            .right_accel_z(right_accel_z),
            .right_rotate(right_rotate),
            .left_rotate(left_rotate),
            .ADbusy(ADbusy),
            .convst_b(convst_b),
            .rd_b(rd_b),
            .gesture_code(gesture_code),
            .accel_code(accel_code),
            .gyro_code(gyro_code),
            .state(state)
      );

      always #10 clock = ~clock;

      initial begin
            // Initialize Inputs
            clock = 0;
            reset_sync = 1;
            left_accel_x = 85;
            left_accel_y = 65;
            left_accel_z = 0;
            right_accel_x = 85;
            right_accel_y = 65;
            right_accel_z = 0;
            right_rotate = 128;
            left_rotate = 128;
            ADbusy = 0;

            #100;
            reset_sync = 0;
            #300;
            right_accel_y = 70;
            left_accel_y = 70;
            #1000;
            right_accel_y = 65;
            left_accel_y = 65;
            left_accel_x = 70;
```

```
            #1500;
            left_accel_x = 85;
            right_accel_x = 90;
            #1500;
            right_accel_x = 85;
            right_rotate = 200;
            #1000;
            right_rotate = 128;



      end

endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////
///////////
// Company:
// Engineer:
//
// Create Date:    12:00:58 05/15/2007
// Design Name:
// Module Name:    volume_sync
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments: This module synchronizes volume enable button
presses with volume enable gestures.
//
//////////////////////////////////////////////////////////////////////
///////////
module volume_sync(reset, clk, sound_done_pulse, volume_enable,
volume_enable_sync, volume);
      input reset;
      input clk;
      input sound_done_pulse;
      input[1:0] volume_enable;
      input[1:0] volume;

      output[1:0] volume_enable_sync;
      reg volume_enable_sync;

      reg wait_for_pulse;
      reg[1:0] volume_enable_int;

      always @ (posedge clk)
      begin
            if(reset)
            begin
                  volume_enable_sync<=0;
                  wait_for_pulse<=0;
            end
//if you detect a pulse
            else if((volume_enable==2'b10) || (volume_enable==2'b01))
            begin
                  wait_for_pulse<=1;
                  volume_enable_int<=volume_enable;
            end
            else if((volume==2'b10) || (volume==2'b01))
            begin
                  wait_for_pulse<=1;
                  volume_enable_int<=volume;
            end
//hold for 1 sample
            else if(wait_for_pulse)
            begin
                  if(sound_done_pulse)
                  begin
                        wait_for_pulse<=0;
                        volume_enable_sync<=volume_enable_int;
                  end
            end
            //hold value for entire sample
            else if(sound_done_pulse)
            begin
                  volume_enable_sync<=0;
            end
      end



endmodule

`timescale 1ns / 1ps
///////////////////////////////////////////////////////////////////////
///////////
// Company:
// Engineer:
//
// Create Date:    14:18:07 04/24/2007
// Design Name:
// Module Name:    zbtcontroller
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments: This module controls reads and writes to the
zbt ram.
//
///////////////////////////////////////////////////////////////////////
///////////
module zbtcontroller(clk, read, write, reset_sync, input_address,
                              input_data, busy, cen, we, ext_address,
read_data,
                              ext_data, state, data_oen, start_read);
      input clk;
      input[18:0] input_address;
      input[35:0] input_data;
      input read;
      input write;
      input reset_sync;

      output busy;
      output cen; //chip enable--active low
      output we; //write enable --active low
      output[35:0] read_data;
      output[18:0] ext_address;
      output[3:0] state;
      output data_oen;
      output start_read;

      inout[35:0] ext_data;


      reg[3:0] state;
      reg data_oen;
      reg data_oen_int;
      reg we;
      reg we_int;
      reg[18:0] ext_address;
      reg[35:0] read_data;
      reg[35:0] int_data;
      reg busy;
      reg data_sample;
      reg start_read;

      assign ext_data= data_oen ? int_data : 36'hz;

      parameter IDLE=0;
      parameter WRITING_PAUSE=1;
      parameter WRITING=2;
      parameter READING=3;
      parameter READING_1=4;
      parameter READING_2=5;
      parameter READING_3=6;
      parameter READING_4=7;
      parameter READING_5=8;

      always @ (posedge clk)
      begin
            //if reset, go back to default, idle
            if(reset_sync)
            begin
                  state<=IDLE;
                  int_data<=0;
                  ext_address<=0;
                  read_data<=0;
                  we<=1;
                  start_read<=0;
```

```
end
else
begin
      // load instantiated values from state block
      data_oen<=0;
      we<=1;
      start_read<=0;
      case(state)
      //check to see if signal from checkfsm
            IDLE:
            begin
                  if(write)
                  begin
                        state<=WRITING_PAUSE;
                        ext_address<=input_address;
                        int_data<=input_data;

                        busy<=1;
                        we<=0;
                  end
                  else if(read)
                  begin
                        state<=READING_1;
                        ext_address<=input_address;
                        busy<=1;
                  end
                  else
                  begin
                        busy<=0;
                        state<=IDLE;
                  end
            end
            WRITING_PAUSE:
            begin
                  busy<=1;
                  state<=WRITING;
            end
            //drive data
            WRITING:
            begin
                  busy<=1;
                  state<=IDLE;
                  data_oen<=1;
            end
            READING_1:
            begin
                  busy<=1;
                  ext_address<=input_address;
                  state<=READING_2;
            end
            //get, sample data
            READING_2:
            begin
                  busy<=1;
                  start_read<=1;
                  ext_address<=input_address;
                  read_data<=ext_data;
```

```
                        state<=READING_3;
                    end
                    READING_3:
                    begin
                        busy<=1;
                        ext_address<=input_address;

                        read_data<=ext_data;
                        state<=READING_4;
                    end
                    READING_4:
                    begin
                        busy<=1;
                        read_data<=ext_data;
                        state<=READING_5;
                    end
                    READING_5:
                    begin
                        busy<=1;
                        read_data<=ext_data;
                        state<=IDLE;
                    end
                    default:
                    begin
                        state<=IDLE;
                    end
                endcase
            end
        end

    endmodule
```