LOHITH'S CODE

```
// This is a MATLAB script to perform the MDCT/IMDCT

function [] = Checkoff( wav_file )
%UNTITLED1 Summary of this function goes here
%   Detailed explanation goes here

COS_T = zeros(16,8);
N=16;
Nhalf=8;
for i = 1:N
    for j = 1:Nhalf
        COS_T(i,j) = round(cos(pi/(Nhalf) * (i-1 + (Nhalf+1)/2) * (j-
1+1/2))*128);
    end
end



%x = rand(1,24); %This is any input you want
x = wavread(wav_file);
figure; plot(x);

x = x(1:floor(size(x)/8)*8)';

n = 1;
temp = size(x);
X=zeros(1,temp/2);
while (n < size(x,2) - 7)
    X(n:n+7) = x(n:n+15)*COS_T;
    n = n + 8;
end

%X(1:8) = x(1:16)*COS_T;
%X(9:16) = x(9:24)*COS_T
figure; plot(X); % This is MDCT

n = 1;

y = zeros(1,size(x,2));
z = zeros(1,size(x,2));
temp = 0;
while (n < size(x,2)-7)
    y(n:n+15) = 1/8*X(n:n+7)*COS_T';
    z(n:n+7) = temp + y(n:n+7);
    temp = y(n+8:n+15);
    n = n+8;
end

figure; plot(z/size(x,2))

wavwrite(z/size(x,2),'output.wav');


% Inverse
```

```
%y=zeros(1,24);      %Inverse of first 16
%z=zeros(1,24);      %inverse of next 16, overlap of 8
%y(1:16) = 1/8*X(1:8)*COS_T';
%z(9:24) = 1/8*X(9:16)*COS_T';
%figure;plot(y+z);
%sum(abs(x(9:16)-(y(9:16)+z(9:16))))    %this should be 0, i.e. inverse
is
                                        %correct for 9 to 16


// This module handles outputing 8 audio points after the FIFO is
filled up.
// The reason for having the FIFO is to control data flow between two
different
// clock domains.
module fifo_controller(clock_27mhz, reset, from_ac97_data, ready,
prog_full, rd_en, data_r, ready_r, start);
      input reset;
      input clock_27mhz;
      input [7:0] from_ac97_data;
      input prog_full;
      input ready;

      output reg ready_r;
      output reg rd_en;
      output reg [7:0] data_r;
      output reg start;

      reg ready_q;
      reg ready_d;
      reg [7:0] data_q;
      reg [7:0] count;

      // Maximum number of audio points for a 2N -> N MDCT Transform
      parameter max_point = 8; //In our case, N = 8

      //States
      parameter start_state = 0;
      parameter MDCT = 1;

      reg [1:0] state;

      always @(posedge clock_27mhz)
            begin
                  // Two register delays to prevent metastability
                  data_r <= data_q;
                  data_q <= from_ac97_data;

                  // Ready signal from AC97 should also be delayed
likewise
                  ready_r <= ready_d;
                  ready_d <= ready;

                  if (reset)
                        begin
                              state <= start_state;
                              start <= 0;
```

```
                              count <= 0;
                        end

                  case(state)
                        // Start state initiates the output of the FIFO
once the FIFO
                        // asserts that it is full (prog_full)
                        start_state:
                              begin
                                    if (prog_full) //Prog_full goes
high when FIFO has stored 8 audio points
                                          begin
                                                start <= 1;
                                                rd_en <= 1;
                                                count <= count + 1;
//Increment count
                                                state <= MDCT;
                                          end
                                    else
                                          begin
                                                start <= 0;
                                                state <= start_state;
                                                rd_en <= 0;
                                          end
                              end
                        MDCT:
                              begin
                                    if(count !== max_point)
                                    //Keep incrementing until we have
output all 8 audio points
                                          begin
                                                count <= count + 1;
                                                state <= MDCT;
                                                rd_en <= 1; //Enable
signal to read from the FIFO
                                          end
                                    else
                                          begin
                                                start <= 0;
                                                count <= 0;
                                                state <= start_state;
                                                rd_en <= 0;
                                          end
                              end
                  endcase
            end
endmodule


// fifoANDmdct module integrate the MDCT and the FIFO controller
// together. The module was also used to communication signals
// between the different modules
module fifoANDmdct(clock_27mhz, reset, start, from_fifo_data,
            to_mdct_data, flush, shift, coefficients, c_avail,
            X0, X1, X2, X3, X4, X5, X6, X7, Xall);
    input clock_27mhz;
    input reset;
```

```
    input start;
    input [7:0] from_fifo_data;
   output [7:0] to_mdct_data;

       output flush;

       output shift;

       output [63:0] coefficients;
       output c_avail;

       wire shift;
       wire [63:0] coefficients;
       wire c_avail;


       reg flush_mult;

       output [7:0] X0;
       output [7:0] X1;
       output [7:0] X2;
       output [7:0] X3;
       output [7:0] X4;
       output [7:0] X5;
       output [7:0] X6;
       output [7:0] X7;

       output reg [7:0] Xall;

       reg [3:0] cnt;

       wire ready;

      // FIFO2MDCT Controller that implements lapping and the flush
signal
       fifo2mdct_controller f2mc1(clock_27mhz, reset, start,
from_fifo_data, to_mdct_data, flush, shift, ready);

      //The actual MDCT module
       mdct mdct2(.clock_27mhz(clock_27mhz), .reset(reset),
.data(to_mdct_data), .ready(ready),
            .flush(flush), .coefficients(coefficients),
.c_avail(c_avail), .shift(shift),
            .X0out(X0), .X1out(X1), .X2out(X2), .X3out(X3), .X4out(X4),
.X5out(X5), .X6out(X6), .X7out(X7));

            always @(posedge clock_27mhz)
                  begin
                        if (reset)
                              begin
                                    cnt <= 8;
                              end
                        if (shift)
                              begin
                                    if (cnt == 8)
                                       begin
                                             cnt <= 0;
```

```
                                    end
                              end
                        // The following script simply separates the
coefficients
                        // variable to look at the individual MDCT
coefficients. This was
                        // used for testing purposes only.
                        if (cnt == 0)
                              begin
                                    cnt <= cnt + 1;
                                    Xall <= coefficients[7:0];
                              end
                        else if (cnt == 1)

                              begin
                                    cnt <= cnt + 1;
                                    Xall <= coefficients[15:8];
                              end
                        else if (cnt == 2)

                              begin
                                    cnt <= cnt + 1;
                                    Xall <= coefficients[23:16];
                              end
                        else if (cnt == 3)
                              begin
                                    cnt <= cnt + 1;
                                    Xall <= coefficients[31:24];
                              end
                        else if (cnt == 4)
                              begin
                                    cnt <= cnt + 1;
                                    Xall <= coefficients[39:32];
                              end
                        else if (cnt == 5)
                              begin
                                    cnt <= cnt + 1;
                                    Xall <= coefficients[47:40];
                              end
                        else if (cnt == 6)
                              begin
                                    cnt <= cnt + 1;
                                    Xall <= coefficients[55:48];
                              end
                        else if (cnt == 7)
                              begin
                                    cnt <= cnt+1;
                                    Xall <= coefficients[63:56];
                              end
                        else
                              begin
                              end
                  end

endmodule

//The fifo2mdct_controller module implements lapping in a weird manner.
```

```
The input data
// is delayed numerous times to let previous stored data in the BRAM to
be output and then
// is finally fed into the system.

module fifo2mdct_controller(clock_27mhz, reset, start, from_fifo_data,
to_mdct_data, flush, shift, ready);
    input clock_27mhz;
    input reset;
    input start;
    input [7:0] from_fifo_data;
    output [7:0] to_mdct_data;

    output flush;
       output shift;
       output ready;

       wire [3:0] ADDR;
       wire [7:0] DIN;
       wire WE;
       wire [7:0] DOUT;

       //DELAY WIRES
       wire [7:0] from_fifo_data_delayed;
       wire [7:0] delay1;
       wire [7:0] delay2;
       wire [7:0] delay3;
       wire [7:0] delay4;
       wire [7:0] delay5;
       wire [7:0] delay6;
       wire [7:0] delay7;
       wire [7:0] delay8;
       wire [7:0] delay9;
       wire [7:0] delay10;
       wire [7:0] delay11;

       //This controller implements the FSM that keeps everything
       // in perfect timing
       lap_controller lc1(.clock_27mhz(clock_27mhz), .reset(reset),
                  .start(start),
.from_fifo_data_delayed(from_fifo_data_delayed),
                  .dout(DOUT), .addr(ADDR), .din(DIN),
                  .we(WE), .flush(flush), .to_mdct_data(to_mdct_data),
.shift(shift), .ready(ready));

      // The COREGEN internal BRAM module
       bram_mdct bm1(.addr(ADDR), .din(DIN), .we(WE),
.clk(clock_27mhz), .dout(DOUT));

      // 8 delays for the previous 8 data points to be output and 4
more for pipeline latency in the
      // BRAM module
       delay
reg_delay1(.D(from_fifo_data),.Q(delay1),.CLK(clock_27mhz));
       delay reg_delay2(.D(delay1),.Q(delay2),.CLK(clock_27mhz));
       delay reg_delay3(.D(delay2),.Q(delay3),.CLK(clock_27mhz));
       delay reg_delay4(.D(delay3),.Q(delay4),.CLK(clock_27mhz));
```

```
       delay reg_delay5(.D(delay4),.Q(delay5),.CLK(clock_27mhz));
       delay reg_delay6(.D(delay5),.Q(delay6),.CLK(clock_27mhz));
       delay reg_delay7(.D(delay6),.Q(delay7),.CLK(clock_27mhz));
       delay reg_delay8(.D(delay7),.Q(delay8),.CLK(clock_27mhz));
       delay reg_delay9(.D(delay8),.Q(delay9),.CLK(clock_27mhz));
       delay reg_delay10(.D(delay9),.Q(delay10),.CLK(clock_27mhz));
       delay reg_delay11(.D(delay10),.Q(delay11),.CLK(clock_27mhz));
       delay
reg_delay12(.D(delay11),.Q(from_fifo_data_delayed),.CLK(clock_27mhz));

endmodule

// The lap_controller module takes care of overlapping audio frames to
allow for MDCT computation
module lap_controller(clock_27mhz, reset, start,
from_fifo_data_delayed, dout, addr, din, we, flush, to_mdct_data,
shift, ready);
    input clock_27mhz;
    input reset;
    input start;
    input [7:0] from_fifo_data_delayed;
    input [7:0] dout;


    output reg [3:0] addr;
    output reg [7:0] din;
    output reg we;
    reg ready;
    output reg flush;
    output reg [7:0] to_mdct_data;


       output reg shift;
       output reg ready;

       reg [3:0] count;

       reg [7:0] from_fifo_data_delayed2;

      // Manul delay parameter for flush signal before it asserts
       parameter flush_delay = 4;

       //STATES
       reg [1:0] state;
       parameter state0 = 0;
       parameter state1 = 1;
       parameter state2 = 2;
       parameter state3 = 3;

       always @(posedge clock_27mhz)
            begin
                  if (reset)
                        begin
                              count <= 0;
                              addr <= 0;
                              flush <= 1;
                              ready <= 0;
```

```
                              state <= state0;
                              we <= 0;
                        end
                        from_fifo_data_delayed2 <=
from_fifo_data_delayed;
                  case (state)
                        state0:
                              begin
                                    if (start)
                                          begin
                                                we <= 0;
                                                addr <= 0;
                                                to_mdct_data <= dout;
                                                state <= state1;
                                          end
                                    else
                                          begin
                                                state <= state0;
                                          end
                                    shift <= 0;
                              end
                        state1:
                        // This FSM was created AD-HOC to hack around
timing issues.
                        // A much cleaner version without the use of a
BRAM or FIFO
                        // was implemented for the IMDCT side of
things.
                              begin
                                    if (addr == 10)
                                          begin
                                                we <= 1;
                                                addr <= addr + 1;
                                                din <=
from_fifo_data_delayed2;
                                                to_mdct_data <=
from_fifo_data_delayed;
                                                state <= state1;
                                          end
                                    else if (addr == 11)
                                          begin
                                                we <= 1;
                                                addr <= 0;
                                                din <=
from_fifo_data_delayed2;
                                                to_mdct_data <=
from_fifo_data_delayed;
                                                state <= state2;
                                          end

                                    else
                                          begin
                                                if (addr == 2)
                                                      begin
                                                            ready <= 1;
                                                      end
//                                              if (addr == 8)
```

```
//                                                      begin
//                                                            din <=
from_fifo_data_delayed2;
//                                                      end
                                                  if (addr == 9)
                                                      begin
                                                            din <=
from_fifo_data_delayed2;
//
      to_mdct_data <= from_fifo_data_delayed;

      to_mdct_data <= dout;
                                                            flush <= 0;
                                                      end
                                                  else
                                                      begin

      to_mdct_data <= dout;
                                                      end
                                                  addr <= addr + 1;
                                                  if (addr == 9)
                                                      begin
                                                            we <= 1;
                                                      end
                                                  else
                                                      begin
                                                            we <= 0;
                                                      end

                                                  state <= state1;
                                            end

                              end
                        state2:
                              begin
                                    if (addr == 8)
                                          begin
                                                we <= 0;
                                                count <= 0;

                                                state <= state3;
                                          end
                                    else
                                          begin
                                                addr <= addr + 1;
                                                we <= 1;
                                                din <=
from_fifo_data_delayed2;
                                                to_mdct_data <=
from_fifo_data_delayed;
                                                state <= state2;
                                                if (addr == 6)
                                                      begin
                                                            ready <= 0;
                                                      end
                                          end
                              end
```

```
                        state3:
                              begin
                                    if (count == flush_delay)
                                          begin
                                                flush <= 1;
                                                shift <= 1;
                                                state <= state0;
                                          end
                                    else if (count == flush_delay - 1)
                                          begin

                                                count <= count + 1;
                                                state <= state3;
                                          end
                                    else
                                          begin
                                                count <= count+1;
                                                state <= state3;
                                          end
                              end
                  endcase
            end
endmodule

// The mdct module is the core of the computational intelligence behind
the MDCT implemented on the wireless
// audio effects systems
module mdct(clock_27mhz, reset, data, ready, flush, coefficients,
c_avail, shift,
      X0out, X1out, X2out, X3out, X4out, X5out, X6out, X7out);
    input clock_27mhz;
    input reset;
    input [7:0] data;
    input ready;
       input flush;
       input shift;

       output reg [63:0] coefficients;

       output reg c_avail;

       output reg [7:0] X0out;
       output reg [7:0] X1out;
       output reg [7:0] X2out;
       output reg [7:0] X3out;
       output reg [7:0] X4out;
       output reg [7:0] X5out;
       output reg [7:0] X6out;
       output reg [7:0] X7out;

      //Internal wires
       wire [7:0] data_s;

       wire [15:0] mult0;
       wire [15:0] mult1;
       wire [15:0] mult2;
       wire [15:0] mult3;
```

```
       wire [15:0] mult4;
       wire [15:0] mult5;
       wire [15:0] mult6;
       wire [15:0] mult7;


      // Two variables for the adder output
      // 1. Before register and at output of adder
      // 2. After register and at input of adder
      wire [15:0] X0;
      reg [15:0] X0a;

      wire [15:0] X1;
      reg [15:0] X1a;

      wire [15:0] X2;
      reg [15:0] X2a;

      wire [15:0] X3;
      reg [15:0] X3a;

      wire [15:0] X4;
      reg [15:0] X4a;

      wire[15:0] X5;
      reg [15:0] X5a;

      wire [15:0] X6;
      reg [15:0] X6a;

      wire [15:0] X7;
      reg [15:0] X7a;



       wire [7:0] cos_x0;
       wire [7:0] cos_x1;
       wire [7:0] cos_x2;
       wire [7:0] cos_x3;
       wire [7:0] cos_x4;
       wire [7:0] cos_x5;
       wire [7:0] cos_x6;
       wire [7:0] cos_x7;

       wire [4:0] n;
       wire x_le;

       wire [4:0] count;


      // The mdct_fsm handles outputing $n$ which is the COSINE lookup
table index
      mdct_fsm mdct_fsm1(.clock_27mhz(clock_27mhz), .reset(reset),
                                                    .data(data),
.data_s(data_s),
                                                    .ready(ready), .n(n),
.count(count));
```

```
      // The cosine table was created in MATLAB and hardcoded as a
series of MUXs
      mdct_cos_table cos_table1(.clock_27mhz(clock_27mhz),
.reset(reset),
                                                              .n(n),
.cos_x0(cos_x0),

      .cos_x1(cos_x1), .cos_x2(cos_x2),

      .cos_x3(cos_x3), .cos_x4(cos_x4),

      .cos_x5(cos_x5), .cos_x6(cos_x6),

      .cos_x7(cos_x7));

      mdct_mult
mdct_mult0(.clk(clock_27mhz),.a(data_s),.b(cos_x0),.q(mult0));
      mdct_mult
mdct_mult1(.clk(clock_27mhz),.a(data_s),.b(cos_x1),.q(mult1));
      mdct_mult
mdct_mult2(.clk(clock_27mhz),.a(data_s),.b(cos_x2),.q(mult2));
      mdct_mult
mdct_mult3(.clk(clock_27mhz),.a(data_s),.b(cos_x3),.q(mult3));
      mdct_mult
mdct_mult4(.clk(clock_27mhz),.a(data_s),.b(cos_x4),.q(mult4));
      mdct_mult
mdct_mult5(.clk(clock_27mhz),.a(data_s),.b(cos_x5),.q(mult5));
      mdct_mult
mdct_mult6(.clk(clock_27mhz),.a(data_s),.b(cos_x6),.q(mult6));
      mdct_mult
mdct_mult7(.clk(clock_27mhz),.a(data_s),.b(cos_x7),.q(mult7));

      mdct_adder mdct_adder0(.A(X0a),.B(mult0),.S(X0));
      mdct_adder mdct_adder1(.A(X1a),.B(mult1),.S(X1));
      mdct_adder mdct_adder2(.A(X2a),.B(mult2),.S(X2));
      mdct_adder mdct_adder3(.A(X3a),.B(mult3),.S(X3));
      mdct_adder mdct_adder4(.A(X4a),.B(mult4),.S(X4));
      mdct_adder mdct_adder5(.A(X5a),.B(mult5),.S(X5));
      mdct_adder mdct_adder6(.A(X6a),.B(mult6),.S(X6));
      mdct_adder mdct_adder7(.A(X7a),.B(mult7),.S(X7));

      always @(posedge clock_27mhz)
            begin
                  if (reset)
                        begin
                              X0a <= 0;
                              X1a <= 0;
                              X2a <= 0;
                              X3a <= 0;
                              X4a <= 0;
                              X5a <= 0;
                              X6a <= 0;
                              X7a <= 0;
                        end
                  // Implementation of flush signal occurs here
                  if (~flush)
```

```
                        begin
                              X0a <= X0;
                              X1a <= X1;
                              X2a <= X2;
                              X3a <= X3;
                              X4a <= X4;
                              X5a <= X5;
                              X6a <= X6;
                              X7a <= X7;
                        end
                  else
                        begin
                              X0a <= 0;
                              X1a <= 0;
                              X2a <= 0;
                              X3a <= 0;
                              X4a <= 0;
                              X5a <= 0;
                              X6a <= 0;
                              X7a <= 0;
                        end

                  // Output of MDCT module to wireless module
                  if (shift)
                        begin
                              coefficients <=
{X0a[15:8],X1a[15:8],X2a[15:8],X3a[15:8],X4a[15:8],X5a[15:8],X6a[15:8],
X7a[15:8]};
                              c_avail <= 1;
                        end
                  else
                        begin
                              c_avail <= 0;
                        end

                  X0out <= X0a[15:8];
                  X1out <= X1a[15:8];
                  X2out <= X2a[15:8];
                  X3out <= X3a[15:8];
                  X4out <= X4a[15:8];
                  X5out <= X5a[15:8];
                  X6out <= X6a[15:8];
                  X7out <= X7a[15:8];

            end

endmodule

module mdct_fsm(clock_27mhz, reset, data, data_s, ready, n, count)
    input clock_27mhz;
    input reset;
    input [7:0] data;
    input ready;

       output reg [4:0] n;
       output reg [7:0] data_s;
       reg [7:0] data_r;
```

```
reg [1:0] state;

//States
parameter start_state = 0;
parameter state1 = 1;

parameter max_double_point = 16;

reg [7:0] data_q;
reg ready_q;
reg ready_r;

output reg [4:0] count;

always @(posedge clock_27mhz)
     begin
           if (reset)
                 begin
                       count <= 0;
                       state <= start_state;
                 end
           else
                 begin
                       data_q <= data;
                       data_s <= data_q;

                       ready_r <= ready;
                 end

           case(state)
                 start_state:
                       begin
                             if (ready_r)
                                   begin
                                         n <= count;
                                         count <= count + 1;
                                         state <= state1;

                                   end
                       end
                 state1:
                       begin
                             if (count !== max_double_point)
                                   begin
                                         n <= count;
                                         count <= count + 1;
                                         state <= state1;
                                   end
                             else
                                   begin
                                         count <= 0;
                                         state <= start_state;

                                   end
                       end
```

```
                endcase
            end


endmodule

module mdct_cos_table(clock_27mhz, reset, n, cos_x0, cos_x1, cos_x2, cos_x3, cos_x4,
cos_x5, cos_x6, cos_x7);
    input clock_27mhz;
    input reset;
    input [4:0] n;
    output reg [7:0] cos_x0;
    output reg [7:0] cos_x1;
    output reg [7:0] cos_x2;
    output reg [7:0] cos_x3;
    output reg [7:0] cos_x4;
    output reg [7:0] cos_x5;
    output reg [7:0] cos_x6;
        output reg [7:0] cos_x7;

        parameter n0= 0;
        parameter n1= 1;
        parameter n2= 2;
        parameter n3= 3;
        parameter n4= 4;
        parameter n5= 5;
        parameter n6= 6;
        parameter n7= 7;
        parameter n8= 8;
        parameter n9= 9;
        parameter n10= 10;
        parameter n11= 11;
        parameter n12= 12;
        parameter n13= 13;
        parameter n14= 14;
        parameter n15= 15;


        always @(posedge clock_27mhz)
            begin
                case(n)
                    n0:
                        begin
                            cos_x0 <= 8'b01010001;
                            cos_x1 <= 8'b10001111;
                            cos_x2 <= 8'b11011011;
                            cos_x3 <= 8'b01111111;
                            cos_x4 <= 8'b11110011;
                            cos_x5 <= 8'b10000110;
                            cos_x6 <= 8'b00111100;
                            cos_x7 <= 8'b01100011;
                        end
                    n1:
                        begin
                            cos_x0 <= 8'b00111100;
                            cos_x1 <= 8'b10000001;
```

```
                cos_x2 <= 8'b01010001;
                cos_x3 <= 8'b00100101;
                cos_x4 <= 8'b10000110;
                cos_x5 <= 8'b01100011;
                cos_x6 <= 8'b00001101;
                cos_x7 <= 8'b10001111;
        end
n2:
        begin
                cos_x0 <= 8'b100101;
                cos_x1 <= 8'b10011101;
                cos_x2 <= 8'b1111111;
                cos_x3 <= 8'b10001111;
                cos_x4 <= 8'b111100;
                cos_x5 <= 8'b1101;
                cos_x6 <= 8'b10101111;
                cos_x7 <= 8'b1111010;
        end
n3:
        begin
                cos_x0 <= 8'b1101;
                cos_x1 <= 8'b11011011;
                cos_x2 <= 8'b111100;
                cos_x3 <= 8'b10101111;
                cos_x4 <= 8'b1100011;
                cos_x5 <= 8'b10001111;
                cos_x6 <= 8'b1111010;
                cos_x7 <= 8'b10000001;
        end
n4:
        begin
                cos_x0 <= 8'b11110011;
                cos_x1 <= 8'b100101;
                cos_x2 <= 8'b11000100;
                cos_x3 <= 8'b1010001;
                cos_x4 <= 8'b10011101;
                cos_x5 <= 8'b1110001;
                cos_x6 <= 8'b10000110;
                cos_x7 <= 8'b1111111;
        end
n5:
        begin
                cos_x0 <= 8'b11011011;
                cos_x1 <= 8'b1100011;
                cos_x2 <= 8'b10000001;
                cos_x3 <= 8'b1110001;
                cos_x4 <= 8'b11000100;
                cos_x5 <= 8'b11110011;
                cos_x6 <= 8'b1010001;
                cos_x7 <= 8'b10000110;
        end
n6:
        begin
                cos_x0 <= 8'b11000100;
                cos_x1 <= 8'b1111111;
                cos_x2 <= 8'b10101111;
```

```
                cos_x3 <= 8'b11011011;
                cos_x4 <= 8'b1111010;
                cos_x5 <= 8'b10011101;
                cos_x6 <= 8'b11110011;
                cos_x7 <= 8'b1110001;
        end
n7:
        begin
                cos_x0 <= 8'b10101111;
                cos_x1 <= 8'b1110001;
                cos_x2 <= 8'b100101;
                cos_x3 <= 8'b10000001;
                cos_x4 <= 8'b1101;
                cos_x5 <= 8'b1111010;
                cos_x6 <= 8'b11000100;
                cos_x7 <= 8'b10011101;
        end
n8:
        begin
                cos_x0 <= 10011101;
                cos_x1 <= 8'b111100;
                cos_x2 <= 8'b1111010;
                cos_x3 <= 8'b11110011;
                cos_x4 <= 8'b10000001;
                cos_x5 <= 8'b11011011;
                cos_x6 <= 8'b1110001;
                cos_x7 <= 8'b1010001;
        end
n9:
        begin
                cos_x0 <= 10001111;
                cos_x1 <= 8'b11110011;
                cos_x2 <= 8'b1100011;
                cos_x3 <= 8'b1111010;
                cos_x4 <= 8'b100101;
                cos_x5 <= 8'b10101111;
                cos_x6 <= 8'b10000001;
                cos_x7 <= 8'b11000100;
        end
n10:
        begin
                cos_x0 <= 10000110;
                cos_x1 <= 8'b10101111;
                cos_x2 <= 8'b11110011;
                cos_x3 <= 8'b111100;
                cos_x4 <= 8'b1110001;
                cos_x5 <= 8'b1111111;
                cos_x6 <= 8'b1100011;
                cos_x7 <= 8'b100101;
        end
n11:
        begin
                cos_x0 <= 8'b10000001;
                cos_x1 <= 8'b10000110;
                cos_x2 <= 8'b10001111;
                cos_x3 <= 8'b10011101;
```

```
                cos_x4 <= 8'b10101111;
                cos_x5 <= 8'b11000100;
                cos_x6 <= 8'b11011011;
                cos_x7 <= 8'b11110011;
            end
        n12:
            begin
                cos_x0 <= 8'b10000001;
                cos_x1 <= 8'b10000110;
                cos_x2 <= 8'b10001111;
                cos_x3 <= 8'b10011101;
                cos_x4 <= 8'b10101111;
                cos_x5 <= 8'b11000100;
                cos_x6 <= 8'b11011011;
                cos_x7 <= 8'b11110011;
            end
        n13:
            begin
                cos_x0 <= 8'b10000110;
                cos_x1 <= 8'b10101111;
                cos_x2 <= 8'b11110011;
                cos_x3 <= 8'b111100;
                cos_x4 <= 8'b1110001;
                cos_x5 <= 8'b1111111;
                cos_x6 <= 8'b1100011;
                cos_x7 <= 8'b100101;
            end
        n14:
            begin
                cos_x0 <= 10001111;
                cos_x1 <= 8'b11110011;
                cos_x2 <= 8'b1100011;
                cos_x3 <= 8'b1111010;
                cos_x4 <= 8'b100101;
                cos_x5 <= 8'b10101111;
                cos_x6 <= 8'b10000001;
                cos_x7 <= 8'b11000100;
            end
        n15:
            begin
                cos_x0 <= 8'b10011101;
                cos_x1 <= 8'b111100;
                cos_x2 <= 8'b1111010;
                cos_x3 <= 8'b11110011;
                cos_x4 <= 8'b10000001;
                cos_x5 <= 8'b11011011;
                cos_x6 <= 8'b1110001;
                cos_x7 <= 8'b1010001;
            end
        endcase
    end

endmodule

//This module is very much like fifo2mdct_controller.v
```

```
module imdctANDfifo_controller(clock_27mhz, reset, coefficients, c_avail, to_fifo_data, wr_en);
    input clock_27mhz;
    input reset;
    input [63:0] coefficients;
    input c_avail;
    output [7:0] to_fifo_data;
    output wr_en;

            wire bram_ready;
            wire [15:0] data;

            wire [15:0] temp_out;

            assign to_fifo_data = temp_out[15:8];

            imdct imdct1(.clock_27mhz(clock_27mhz), .reset(reset),
                                .coefficients(coefficients), .c_avail(c_avail),
                                .data(data), .bram_ready(bram_ready));

            imdct2fifo_controller i2f_c(.clock_27mhz(clock_27mhz), .reset(reset),
.bram_ready(bram_ready),
                                .from_imdct_data(data), .to_fifo_data (temp_out), .wr_en(wr_en));

endmodule


module imdct(clock_27mhz, reset, coefficients, c_avail, data, bram_ready);
    input clock_27mhz;
    input reset;
    input [63:0] coefficients;
    input c_avail;

           output [15:0] data;
           output bram_ready;

            reg [7:0] X0a;
            reg [7:0] X1a;
            reg [7:0] X2a;
            reg [7:0] X3a;
            reg [7:0] X4a;
            reg [7:0] X5a;
            reg [7:0] X6a;
            reg [7:0] X7a;

            wire [7:0] cos_x0;
            wire [7:0] cos_x1;
            wire [7:0] cos_x2;
            wire [7:0] cos_x3;
            wire [7:0] cos_x4;
            wire [7:0] cos_x5;
            wire [7:0] cos_x6;
            wire [7:0] cos_x7;

            wire [4:0] n;

            wire [15:0] mult0;
```

```
wire [15:0] mult1;
wire [15:0] mult2;
wire [15:0] mult3;
wire [15:0] mult4;
wire [15:0] mult5;
wire [15:0] mult6;
wire [15:0] mult7;

wire [15:0] multa;
wire [15:0] multb;
wire [15:0] multc;
wire [15:0] multd;
wire [15:0] mult1a;
wire [15:0] mult2a;

wire ready2;
wire ready3;
wire ready4;
wire ready5;
wire ready6;
wire ready7;
wire ready8;

imdct_fsm imdct_fsm1(.clock_27mhz(clock_27mhz), .reset(reset),
                                                    .ready(c_avail), .n(n));

imdct_cos_table icos_table1(.clock_27mhz(clock_27mhz), .reset(reset),
                                                                .n(n),
.cos_x0(cos_x0),

.cos_x1(cos_x1), .cos_x2(cos_x2),

.cos_x3(cos_x3), .cos_x4(cos_x4),

.cos_x5(cos_x5), .cos_x6(cos_x6),

.cos_x7(cos_x7));

imdct_mult imdct_mult0(.clk(clock_27mhz),.a(X0a),.b(cos_x0),.q(mult0));
imdct_mult imdct_mult1(.clk(clock_27mhz),.a(X1a),.b(cos_x1),.q(mult1));
imdct_mult imdct_mult2(.clk(clock_27mhz),.a(X2a),.b(cos_x2),.q(mult2));
imdct_mult imdct_mult3(.clk(clock_27mhz),.a(X3a),.b(cos_x3),.q(mult3));
imdct_mult imdct_mult4(.clk(clock_27mhz),.a(X4a),.b(cos_x4),.q(mult4));
imdct_mult imdct_mult5(.clk(clock_27mhz),.a(X5a),.b(cos_x5),.q(mult5));
imdct_mult imdct_mult6(.clk(clock_27mhz),.a(X6a),.b(cos_x6),.q(mult6));
imdct_mult imdct_mult7(.clk(clock_27mhz),.a(X7a),.b(cos_x7),.q(mult7));

imdct_adder imdct_add0n1(.CLK(clock_27mhz),.A(mult0),.B(mult1),.Q(multa));
imdct_adder imdct_add2n3(.CLK(clock_27mhz),.A(mult2),.B(mult3),.Q(multb));
imdct_adder imdct_add4n5(.CLK(clock_27mhz),.A(mult4),.B(mult5),.Q(multc));
imdct_adder imdct_add6n7(.CLK(clock_27mhz),.A(mult6),.B(mult7),.Q(multd));

imdct_adder imdct_addanb(.CLK(clock_27mhz),.A(multa),.B(multb),.Q(mult1a));
imdct_adder imdct_addcnd(.CLK(clock_27mhz),.A(multc),.B(multd),.Q(mult2a));

imdct_adder imdct_final_add(.CLK(clock_27mhz),.A(mult1a),.B(mult2a),.Q(data));
```

```
        imdct_ready_delay idelay1(.CLK(clock_27mhz), .D(c_avail), .Q(ready2));
        imdct_ready_delay idelay2(.CLK(clock_27mhz), .D(ready2), .Q(ready3));
        imdct_ready_delay idelay3(.CLK(clock_27mhz), .D(ready3), .Q(ready4));
        imdct_ready_delay idelay4(.CLK(clock_27mhz), .D(ready4), .Q(ready5));
        imdct_ready_delay idelay5(.CLK(clock_27mhz), .D(ready5), .Q(ready6));

        imdct_ready_delay idelay6(.CLK(clock_27mhz), .D(ready6), .Q(ready7));
        imdct_ready_delay idelay7(.CLK(clock_27mhz), .D(ready7), .Q(ready8));
        imdct_ready_delay idelay8(.CLK(clock_27mhz), .D(ready8), .Q(bram_ready));


        always @(posedge clock_27mhz)
                begin
                        if (c_avail)
                                begin
                                        X0a <= coefficients[63:56];
                                        X1a <= coefficients[55:48];
                                        X2a <= coefficients[47:40];
                                        X3a <= coefficients[39:32];
                                        X4a <= coefficients[31:24];
                                        X5a <= coefficients[23:16];
                                        X6a <= coefficients[15:8];
                                        X7a <= coefficients[7:0];
                                end
                end


endmodule

// This module is almost exact as the one located at the exit of the MDCT module
module imdct_cos_table(clock_27mhz, reset, n, cos_x0, cos_x1, cos_x2, cos_x3, cos_x4,
cos_x5, cos_x6, cos_x7);
    input clock_27mhz;
    input reset;
    input [4:0] n;
    output reg [7:0] cos_x0;
    output reg [7:0] cos_x1;
    output reg [7:0] cos_x2;
    output reg [7:0] cos_x3;
    output reg [7:0] cos_x4;
    output reg [7:0] cos_x5;
    output reg [7:0] cos_x6;
         output reg [7:0] cos_x7;

        parameter n0= 0;
        parameter n1= 1;
        parameter n2= 2;
        parameter n3= 3;
        parameter n4= 4;
        parameter n5= 5;
        parameter n6= 6;
        parameter n7= 7;
        parameter n8= 8;
        parameter n9= 9;
        parameter n10= 10;
```

```
parameter n11= 11;
parameter n12= 12;
parameter n13= 13;
parameter n14= 14;
parameter n15= 15;


 always @(posedge clock_27mhz)
        begin
                case(n)
                        n0:
                                begin
                                        cos_x0 <= 8'b01010001;
                                        cos_x1 <= 8'b10001111;
                                        cos_x2 <= 8'b11011011;
                                        cos_x3 <= 8'b01111111;
                                        cos_x4 <= 8'b11110011;
                                        cos_x5 <= 8'b10000110;
                                        cos_x6 <= 8'b00111100;
                                        cos_x7 <= 8'b01100011;
                                end
                        n1:
                                begin
                                        cos_x0 <= 8'b00111100;
                                        cos_x1 <= 8'b10000001;
                                        cos_x2 <= 8'b01010001;
                                        cos_x3 <= 8'b00100101;
                                        cos_x4 <= 8'b10000110;
                                        cos_x5 <= 8'b01100011;
                                        cos_x6 <= 8'b00001101;
                                        cos_x7 <= 8'b10001111;
                                end
                        n2:
                                begin
                                        cos_x0 <= 8'b100101;
                                        cos_x1 <= 8'b10011101;
                                        cos_x2 <= 8'b1111111;
                                        cos_x3 <= 8'b10001111;
                                        cos_x4 <= 8'b111100;
                                        cos_x5 <= 8'b1101;
                                        cos_x6 <= 8'b10101111;
                                        cos_x7 <= 8'b1111010;
                                end
                        n3:
                                begin
                                        cos_x0 <= 8'b1101;
                                        cos_x1 <= 8'b11011011;
                                        cos_x2 <= 8'b111100;
                                        cos_x3 <= 8'b10101111;
                                        cos_x4 <= 8'b1100011;
                                        cos_x5 <= 8'b10001111;
                                        cos_x6 <= 8'b1111010;
                                        cos_x7 <= 8'b10000001;
                                end
                        n4:
                                begin
```

```
                cos_x0 <= 8'b11110011;
                cos_x1 <= 8'b100101;
                cos_x2 <= 8'b11000100;
                cos_x3 <= 8'b1010001;
                cos_x4 <= 8'b10011101;
                cos_x5 <= 8'b1110001;
                cos_x6 <= 8'b10000110;
                cos_x7 <= 8'b1111111;
        end
n5:
        begin
                cos_x0 <= 8'b11011011;
                cos_x1 <= 8'b1100011;
                cos_x2 <= 8'b10000001;
                cos_x3 <= 8'b1110001;
                cos_x4 <= 8'b11000100;
                cos_x5 <= 8'b11110011;
                cos_x6 <= 8'b1010001;
                cos_x7 <= 8'b10000110;
        end
n6:
        begin
                cos_x0 <= 8'b11000100;
                cos_x1 <= 8'b1111111;
                cos_x2 <= 8'b10101111;
                cos_x3 <= 8'b11011011;
                cos_x4 <= 8'b1111010;
                cos_x5 <= 8'b10011101;
                cos_x6 <= 8'b11110011;
                cos_x7 <= 8'b1110001;
        end
n7:
        begin
                cos_x0 <= 8'b10101111;
                cos_x1 <= 8'b1110001;
                cos_x2 <= 8'b100101;
                cos_x3 <= 8'b10000001;
                cos_x4 <= 8'b1101;
                cos_x5 <= 8'b1111010;
                cos_x6 <= 8'b11000100;
                cos_x7 <= 8'b10011101;
        end
n8:
        begin
                cos_x0 <= 10011101;
                cos_x1 <= 8'b111100;
                cos_x2 <= 8'b1111010;
                cos_x3 <= 8'b11110011;
                cos_x4 <= 8'b10000001;
                cos_x5 <= 8'b11011011;
                cos_x6 <= 8'b1110001;
                cos_x7 <= 8'b1010001;
        end
n9:
        begin
                cos_x0 <= 10001111;
```

```
                cos_x1 <= 8'b11110011;
                cos_x2 <= 8'b1100011;
                cos_x3 <= 8'b1111010;
                cos_x4 <= 8'b100101;
                cos_x5 <= 8'b10101111;
                cos_x6 <= 8'b10000001;
                cos_x7 <= 8'b11000100;
        end
n10:
        begin
                cos_x0 <= 10000110;
                cos_x1 <= 8'b10101111;
                cos_x2 <= 8'b11110011;
                cos_x3 <= 8'b111100;
                cos_x4 <= 8'b1110001;
                cos_x5 <= 8'b1111111;
                cos_x6 <= 8'b1100011;
                cos_x7 <= 8'b100101;
        end
n11:
        begin
                cos_x0 <= 8'b10000001;
                cos_x1 <= 8'b10000110;
                cos_x2 <= 8'b10001111;
                cos_x3 <= 8'b10011101;
                cos_x4 <= 8'b10101111;
                cos_x5 <= 8'b11000100;
                cos_x6 <= 8'b11011011;
                cos_x7 <= 8'b11110011;
        end
n12:
        begin
                cos_x0 <= 8'b10000001;
                cos_x1 <= 8'b10000110;
                cos_x2 <= 8'b10001111;
                cos_x3 <= 8'b10011101;
                cos_x4 <= 8'b10101111;
                cos_x5 <= 8'b11000100;
                cos_x6 <= 8'b11011011;
                cos_x7 <= 8'b11110011;
        end
n13:
        begin
                cos_x0 <= 8'b10000110;
                cos_x1 <= 8'b10101111;
                cos_x2 <= 8'b11110011;
                cos_x3 <= 8'b111100;
                cos_x4 <= 8'b1110001;
                cos_x5 <= 8'b1111111;
                cos_x6 <= 8'b1100011;
                cos_x7 <= 8'b100101;
        end
n14:
        begin
                cos_x0 <= 10001111;
                cos_x1 <= 8'b11110011;
```

```
                                        cos_x2 <= 8'b1100011;
                                        cos_x3 <= 8'b1111010;
                                        cos_x4 <= 8'b100101;
                                        cos_x5 <= 8'b10101111;
                                        cos_x6 <= 8'b10000001;
                                        cos_x7 <= 8'b11000100;
                                end
                        n15:
                                begin
                                        cos_x0 <= 8'b10011101;
                                        cos_x1 <= 8'b111100;
                                        cos_x2 <= 8'b1111010;
                                        cos_x3 <= 8'b11110011;
                                        cos_x4 <= 8'b10000001;
                                        cos_x5 <= 8'b11011011;
                                        cos_x6 <= 8'b1110001;
                                        cos_x7 <= 8'b1010001;
                                end
                        endcase
                end

endmodule


module imdct2fifo_controller(clock_27mhz, reset, bram_ready, from_imdct_data, to_fifo_data
,wr_en);
    input clock_27mhz;
    input reset;
    input bram_ready;
    input [15:0] from_imdct_data;
    output [15:0] to_fifo_data;
    output wr_en;


        ilap_controller ilc1(.clock_27mhz(clock_27mhz), .reset(reset),
                        .bram_ready(bram_ready), .from_imdct_data(from_imdct_data),
                        .to_fifo_data(to_fifo_data), .wr_en(wr_en));


endmodule

// This module is important in the IMDCT. The module implements a fifo-like structure too keep
memory of
// different incidnets
module ilap_controller(clock_27mhz, reset, bram_ready, from_imdct_data, to_fifo_data, wr_en);
    input clock_27mhz;
    input reset;
    input bram_ready;
    input [15:0] from_imdct_data;

        output reg [15:0] to_fifo_data;

    reg [3:0] n;
        output reg wr_en;
```

```
	reg add_load;

	reg [2:0] state;
	//STATES
	parameter state0 = 0;
	parameter state1 = 1;
	parameter state2 = 2;

	parameter change_n = 8;

	reg [15:0] x0old;
	reg [15:0] x1old;
	reg [15:0] x2old;
	reg [15:0] x3old;
	reg [15:0] x4old;
	reg [15:0] x5old;
	reg [15:0] x6old;
	reg [15:0] x7old;

	wire [15:0] x0;
	wire [15:0] x1;
	wire [15:0] x2;
	wire [15:0] x3;
	wire [15:0] x4;
	wire [15:0] x5;
	wire [15:0] x6;
	wire [15:0] x7;


	//Adder
	ilap_controller_adder ilc_adder1(.CLK(clock_27mhz), .A(from_imdct_data), .B(x0old),
.Q(x0));
	ilap_controller_adder ilc_adder2(.CLK(clock_27mhz), .A(from_imdct_data), .B(x1old),
.Q(x1));
	ilap_controller_adder ilc_adder3(.CLK(clock_27mhz), .A(from_imdct_data), .B(x2old),
.Q(x2));
	ilap_controller_adder ilc_adder4(.CLK(clock_27mhz), .A(from_imdct_data), .B(x3old),
.Q(x3));
	ilap_controller_adder ilc_adder5(.CLK(clock_27mhz), .A(from_imdct_data), .B(x4old),
.Q(x4));
	ilap_controller_adder ilc_adder6(.CLK(clock_27mhz), .A(from_imdct_data), .B(x5old),
.Q(x5));
	ilap_controller_adder ilc_adder7(.CLK(clock_27mhz), .A(from_imdct_data), .B(x6old),
.Q(x6));
	ilap_controller_adder ilc_adder8(.CLK(clock_27mhz), .A(from_imdct_data), .B(x7old),
.Q(x7));

	reg add_load_r;
	reg [15:0] from_imdct_data_delayed;

	always @(posedge clock_27mhz)
		begin
			add_load_r <= add_load;
			from_imdct_data_delayed <= from_imdct_data;
			if (reset)
				begin
```

```
                n <= 0;
                wr_en <= 0;
                state <= state0;
                add_load <= 1;
        end
case (state)
        state0:
                begin
                        if (bram_ready)
                                begin
                                        n <= 0;

                                        state <= state1;
                                        wr_en <= 0;
                                end
                        else
                                begin
                                        state <= state0;
                                end
                end
        state1:
                begin
                        if (n == change_n)
                                begin

                                        state <= state2;
                                        add_load <= 0;

                                        n <= 0;
                                end
                        else
                                begin
                                        n <= n+1;
                                        if (n == 1)
                                                begin
                                                        wr_en <= 1;
                                                end
                                end
                end
        state2:
                begin
                        if (n == change_n)
                                begin

                                        state <= state0;
                                        add_load <= 1;
                                        n <= 0;
                                        wr_en <= 0;
                                end
                        else
                                begin
                                        n <= n+1;
                                end
                end
endcase
```

```
                if(add_load)
                        begin
                                case (n)
                                        1:      to_fifo_data <= x0;
                                        2:      to_fifo_data <= x1;
                                        3:      to_fifo_data <= x2;
                                        4:      to_fifo_data <= x3;
                                        5:      to_fifo_data <= x4;
                                        6:      to_fifo_data <= x5;
                                        7:      to_fifo_data <= x6;
                                        8:      to_fifo_data <= x7;
                                endcase
                        end
                else
                        begin
                                case (n)
                                        0:
                                                begin
                                                        x0old <=
from_imdct_data_delayed;
                                                        to_fifo_data <=
from_imdct_data_delayed;
                                                end
                                        1:
                                                begin
                                                        x1old <=
from_imdct_data_delayed;
                                                        to_fifo_data <=
from_imdct_data_delayed;
                                                end
                                        2:
                                                begin
                                                        x2old <=
from_imdct_data_delayed;
                                                        to_fifo_data <=
from_imdct_data_delayed;
                                                end
                                        3:
                                                begin
                                                        x3old <=
from_imdct_data_delayed;
                                                        to_fifo_data <=
from_imdct_data_delayed;
                                                end
                                        4:
                                                begin
                                                        x4old <=
from_imdct_data_delayed;
                                                        to_fifo_data <=
from_imdct_data_delayed;
                                                end
                                        5:
                                                begin
                                                        x5old <=
from_imdct_data_delayed;
                                                        to_fifo_data <=
```

from_imdct_data_delayed;
end
6:
begin
x6old <=
from_imdct_data_delayed;
to_fifo_data <=
from_imdct_data_delayed;
end
7:
begin
x7old <=
from_imdct_data_delayed;
to_fifo_data <=
from_imdct_data_delayed;
end
endcase
end
end

endmodule

RAHULS CODE

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

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

   input reset, clock, noisy;
   output clean;

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

   always @(posedge clock)
     if (reset)
       begin
        count <= 0;
        new <= noisy;
        clean <= noisy;
       end
     else if (noisy != new)
       begin
        new <= noisy;
        count <= 0;
       end
     else if (count == 270000)
       clean <= new;
     else
```

```
       count <= count+1;

endmodule

////////////////////////////////////////////////////////////////////////
////////
//
// bi-directional monaural interface to AC97
//
////////////////////////////////////////////////////////////////////////
////////

module lab4audio (clock_27mhz, reset, volume,
                  audio_in_data, audio_out_data, ready,
                audio_reset_b, ac97_sdata_out, ac97_sdata_in,
                  ac97_synch, ac97_bit_clock);

   input clock_27mhz;
   input reset;
   input [4:0] volume;
   output [7:0] audio_in_data;
   input [7:0] audio_out_data;
   output ready;

   //ac97 interface signals
   output audio_reset_b;
   output ac97_sdata_out;
   input ac97_sdata_in;
   output ac97_synch;
   input ac97_bit_clock;

   wire [2:0] source;
   assign source = 0;      //mic

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

   reg audio_reset_b;
   reg [9:0] reset_count;

   //wait a little before enabling the AC97 codec
   always @(posedge clock_27mhz) begin
      if (reset) begin
         audio_reset_b = 1'b0;
         reset_count = 0;
      end else if (reset_count == 1023)
        audio_reset_b = 1'b1;
      else
        reset_count = reset_count+1;
   end

   wire ac97_ready;
   ac97 ac97(ac97_ready, command_address, command_data, command_valid,
             left_out_data, 1'b1, right_out_data, 1'b1, left_in_data,
```

```
            right_in_data, ac97_sdata_out, ac97_sdata_in, ac97_synch,
            ac97_bit_clock);

   // ready: one cycle pulse synchronous with clock_27mhz
   reg [2:0] ready_sync;
   always @ (posedge clock_27mhz) begin
     ready_sync <= {ready_sync[1:0], ac97_ready};
   end
   assign ready = ready_sync[1] & ~ready_sync[2];

   reg [7:0] out_data;
   always @ (posedge clock_27mhz)
     if (ready) out_data <= audio_out_data;
   assign audio_in_data = left_in_data[19:12];
   assign left_out_data = {out_data, 12'b000000000000};
   assign right_out_data = left_out_data;

   // generate repeating sequence of read/writes to AC97 registers
   ac97commands cmds(clock_27mhz, ready, command_address, command_data,
                     command_valid, volume, source);
endmodule

// assemble/disassemble AC97 serial frames
module ac97 (ready,
             command_address, command_data, command_valid,
             left_data, left_valid,
             right_data, right_valid,
             left_in_data, right_in_data,
             ac97_sdata_out, ac97_sdata_in, ac97_synch,
ac97_bit_clock);

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

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

   reg ready;

   reg ac97_sdata_out;
   reg ac97_synch;

   reg [7:0] bit_count;

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

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

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

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

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

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

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

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

```
        4'h4: ac97_sdata_out <= l_right_v; // Right data valid
          default: ac97_sdata_out <= 1'b0;
        endcase

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

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

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

      bit_count <= bit_count+1;

   end // always @ (posedge ac97_bit_clock)

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

endmodule

// issue initialization commands to AC97
module ac97commands (clock, ready, command_address, command_data,
                     command_valid, volume, source);

   input clock;
   input ready;
   output [7:0] command_address;
   output [15:0] command_data;
   output command_valid;
   input [4:0] volume;
   input [2:0] source;

   reg [23:0] command;
   reg command_valid;

   reg [3:0] state;
```

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

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

   wire [4:0] vol;
   assign vol = 31-volume;  // convert to attenuation

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

      case (state)
        4'h0: // Read ID
          begin
             command <= 24'h80_0000;
             command_valid <= 1'b1;
          end
        4'h1: // Read ID
          command <= 24'h80_0000;
        4'h3: // headphone volume
          command <= { 8'h04, 3'b000, vol, 3'b000, vol };
        4'h5: // PCM volume
          command <= 24'h18_0808;
        4'h6: // Record source select
          command <= { 8'h1A, 5'b00000, source, 5'b00000, source};
        4'h7: // Record gain = max
        command <= 24'h1C_0F0F;
        4'h9: // set +20db mic gain
          command <= 24'h0E_8048;
        4'hA: // Set beep volume
          command <= 24'h0A_0000;
        4'hB: // PCM out bypass mix1
          command <= 24'h20_8000;
        default:
          command <= 24'h80_0000;
      endcase // case(state)
   end // always @ (posedge clock)
endmodule // ac97commands

//////////////////////////////////////////////////////////////////////
//////////
////
//// generate PCM data for 750hz sine wave (assuming f(ready) = 48khz)
////
//////////////////////////////////////////////////////////////////////
//////////
//
//module tone750hz (clock, ready, pcm_data);
//   input clock;
//   input ready;
```

```
//   output [19:0] pcm_data;
//
//   reg [8:0] index;
//   reg [19:0] pcm_data;
//
//   initial begin
//      // synthesis attribute init of old_ready is "0";
//      index <= 8'h00;
//      // synthesis attribute init of index is "00";
//      pcm_data <= 20'h00000;
//      // synthesis attribute init of pcm_data is "00000";
//   end
//
//   always @(posedge clock) begin
//      if (ready) index <= index+1;
//   end
//
//   // one cycle of a sinewave in 64 20-bit samples
//   always @(index) begin
//      case (index[5:0])
//        6'h00: pcm_data <= 20'h00000;
//        6'h01: pcm_data <= 20'h0C8BD;
//        6'h02: pcm_data <= 20'h18F8B;
//        6'h03: pcm_data <= 20'h25280;
//        6'h04: pcm_data <= 20'h30FBC;
//        6'h05: pcm_data <= 20'h3C56B;
//        6'h06: pcm_data <= 20'h471CE;
//        6'h07: pcm_data <= 20'h5133C;
//        6'h08: pcm_data <= 20'h5A827;
//        6'h09: pcm_data <= 20'h62F20;
//        6'h0A: pcm_data <= 20'h6A6D9;
//        6'h0B: pcm_data <= 20'h70E2C;
//        6'h0C: pcm_data <= 20'h7641A;
//        6'h0D: pcm_data <= 20'h7A7D0;
//        6'h0E: pcm_data <= 20'h7D8A5;
//        6'h0F: pcm_data <= 20'h7F623;
//        6'h10: pcm_data <= 20'h7FFFF;
//        6'h11: pcm_data <= 20'h7F623;
//        6'h12: pcm_data <= 20'h7D8A5;
//        6'h13: pcm_data <= 20'h7A7D0;
//        6'h14: pcm_data <= 20'h7641A;
//        6'h15: pcm_data <= 20'h70E2C;
//        6'h16: pcm_data <= 20'h6A6D9;
//        6'h17: pcm_data <= 20'h62F20;
//        6'h18: pcm_data <= 20'h5A827;
//        6'h19: pcm_data <= 20'h5133C;
//        6'h1A: pcm_data <= 20'h471CE;
//        6'h1B: pcm_data <= 20'h3C56B;
//        6'h1C: pcm_data <= 20'h30FBC;
//        6'h1D: pcm_data <= 20'h25280;
//        6'h1E: pcm_data <= 20'h18F8B;
//        6'h1F: pcm_data <= 20'h0C8BD;
//        6'h20: pcm_data <= 20'h00000;
//        6'h21: pcm_data <= 20'hF3743;
//        6'h22: pcm_data <= 20'hE7075;
//        6'h23: pcm_data <= 20'hDAD80;
//        6'h24: pcm_data <= 20'hCF044;
```

```
//        6'h25: pcm_data <= 20'hC3A95;
//        6'h26: pcm_data <= 20'hB8E32;
//        6'h27: pcm_data <= 20'hAECC4;
//        6'h28: pcm_data <= 20'hA57D9;
//        6'h29: pcm_data <= 20'h9D0E0;
//        6'h2A: pcm_data <= 20'h95927;
//        6'h2B: pcm_data <= 20'h8F1D4;
//        6'h2C: pcm_data <= 20'h89BE6;
//        6'h2D: pcm_data <= 20'h85830;
//        6'h2E: pcm_data <= 20'h8275B;
//        6'h2F: pcm_data <= 20'h809DD;
//        6'h30: pcm_data <= 20'h80000;
//        6'h31: pcm_data <= 20'h809DD;
//        6'h32: pcm_data <= 20'h8275B;
//        6'h33: pcm_data <= 20'h85830;
//        6'h34: pcm_data <= 20'h89BE6;
//        6'h35: pcm_data <= 20'h8F1D4;
//        6'h36: pcm_data <= 20'h95927;
//        6'h37: pcm_data <= 20'h9D0E0;
//        6'h38: pcm_data <= 20'hA57D9;
//        6'h39: pcm_data <= 20'hAECC4;
//        6'h3A: pcm_data <= 20'hB8E32;
//        6'h3B: pcm_data <= 20'hC3A95;
//        6'h3C: pcm_data <= 20'hCF044;
//        6'h3D: pcm_data <= 20'hDAD80;
//        6'h3E: pcm_data <= 20'hE7075;
//        6'h3F: pcm_data <= 20'hF3743;
//      endcase // case(index[5:0])
//   end // always @ (index)
//endmodule

//////////////////////////////////////////////////////////////////////////
////////
//
// 6.111 FPGA Labkit -- Template Toplevel Module
//
// For Labkit Revision 004
//
//
// Created: October 31, 2004, from revision 003 file
// Author: Nathan Ickes
//
//////////////////////////////////////////////////////////////////////////
////////
//
// CHANGES FOR BOARD REVISION 004
//
// 1) Added signals for logic analyzer pods 2-4.
// 2) Expanded "tv_in_ycrcb" to 20 bits.
// 3) Renamed "tv_out_data" to "tv_out_i2c_data" and "tv_out_sclk" to
//    "tv_out_i2c_clock".
// 4) Reversed disp_data_in and disp_data_out signals, so that "out" is
an
//    output of the FPGA, and "in" is an input.
//
// CHANGES FOR BOARD REVISION 003
//
```

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

module lab4   (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
   //

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


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

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

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

   // Audio Input and Output
   assign beep= 1'b0;
   //lab3 assign audio_reset_b = 1'b0;
   //lab3 assign ac97_synch = 1'b0;
   //lab3 assign ac97_sdata_out = 1'b0;
   // ac97_sdata_in is an input

   // VGA Output
//   assign vga_out_red = 10'h0;
//   assign vga_out_green = 10'h0;
//   assign vga_out_blue = 10'h0;
```

```
//   assign vga_out_sync_b = 1'b1;
//   assign vga_out_blank_b = 1'b1;
//   assign vga_out_pixel_clock = 1'b0;
//   assign vga_out_hsync = 1'b0;
//   assign vga_out_vsync = 1'b0;

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

   // Video Input
   assign tv_in_i2c_clock = 1'b0;
   assign tv_in_fifo_read = 1'b0;
   assign tv_in_fifo_clock = 1'b0;
   assign tv_in_iso = 1'b0;
   assign tv_in_reset_b = 1'b0;
   assign tv_in_clock = 1'b0;
   assign tv_in_i2c_data = 1'bZ;
   // tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1,
tv_in_line_clock2,
   // tv_in_aef, tv_in_hff, and tv_in_aff are inputs

   // SRAMs
   assign ram0_data = 36'hZ;
   assign ram0_address = 19'h0;
   assign ram0_adv_ld = 1'b0;
   assign ram0_clk = 1'b0;
   assign ram0_cen_b = 1'b1;
   assign ram0_ce_b = 1'b1;
   assign ram0_oe_b = 1'b1;
   assign ram0_we_b = 1'b1;
   assign ram0_bwe_b = 4'hF;
   assign ram1_data = 36'hZ;
   assign ram1_address = 19'h0;
   assign ram1_adv_ld = 1'b0;
   assign ram1_clk = 1'b0;
   assign ram1_cen_b = 1'b1;
   assign ram1_ce_b = 1'b1;
   assign ram1_oe_b = 1'b1;
   assign ram1_we_b = 1'b1;
   assign ram1_bwe_b = 4'hF;
   assign clock_feedback_out = 1'b0;
   // clock_feedback_in is an input

   // Flash ROM
   assign flash_data = 16'hZ;
   assign flash_address = 24'h0;
   assign flash_ce_b = 1'b1;
   assign flash_oe_b = 1'b1;
```

```
   assign flash_we_b = 1'b1;
   assign flash_reset_b = 1'b0;
   assign flash_byte_b = 1'b1;
   // flash_sts is an input

   // RS-232 Interface
   assign rs232_txd = 1'b1;
   assign rs232_rts = 1'b1;
   // rs232_rxd and rs232_cts are inputs

   // PS/2 Ports
   // mouse_clock, mouse_data, keyboard_clock, and keyboard_data are
inputs

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

   // Buttons, Switches, and Individual LEDs
   //lab3 assign led = 8'hFF;
   // button0, button1, button2, button3, button_enter, button_right,
   // button_left, button_down, button_up, and switches are inputs

   // User I/Os
   //assign user1 = 32'hZ;
   assign user2 = 32'hZ;
   assign user3 = 32'hZ;
   assign user4 = 32'hZ;

   // Daughtercard Connectors
   assign daughtercard = 44'hZ;

   // SystemACE Microprocessor Port
   assign systemace_data = 16'hZ;
   assign systemace_address = 7'h0;
   assign systemace_ce_b = 1'b1;
   assign systemace_we_b = 1'b1;
   assign systemace_oe_b = 1'b1;
   // systemace_irq and systemace_mpbrdy are inputs

   // Logic Analyzer
   assign analyzer1_data = 16'h0;
   assign analyzer1_clock = 1'b1;
   assign analyzer2_data = 16'h0;
   //lab3
      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 [7:0] from_ac97_data, to_ac97_data;
```

```
   wire ready;

   // allow user to adjust volume
   wire vup,vdown;
   reg old_vup,old_vdown;
   debounce bup(reset, clock_27mhz, ~button_up, vup);
   debounce bdown(reset, clock_27mhz, ~button_down, vdown);
   reg [4:0] volume;
   always @ (posedge clock_27mhz) begin
     if (reset) volume <= 5'd8;
     else begin
      if (vup & ~old_vup & volume != 5'd31) volume <= volume+1;
      if (vdown & ~old_vdown & volume != 5'd0) volume <= volume-1;
     end
     old_vup <= vup;
     old_vdown <= vdown;
   end

   // AC97 driver
   lab4audio a(clock_27mhz, reset, volume, from_ac97_data,
to_ac97_data, ready,
            audio_reset_b, ac97_sdata_out, ac97_sdata_in,
            ac97_synch, ac97_bit_clock);

   // push ENTER button to record, release to playback
   wire playback;
   debounce benter(reset, clock_27mhz, button_enter, playback);

   // light up LEDs when recording, show volume during playback.
   // led is active low
   assign led = playback ? ~{3'b000, volume} : 8'h00;

   // record module
//   recorder r(clock_27mhz, reset, playback, ready, from_ac97_data,
to_ac97_data);
   // output useful things to the logic analyzer connectors
//   assign analyzer1_clock = ac97_bit_clock;
//   assign analyzer1_data[0] = audio_reset_b;
//   assign analyzer1_data[1] = ac97_sdata_out;
//   assign analyzer1_data[2] = ac97_sdata_in;
//   assign analyzer1_data[3] = ac97_synch;
//   assign analyzer1_data[15:4] = 0;
//
//   assign analyzer3_clock = clock_27mhz;
//   assign analyzer3_data = {6'b0,playback, ready, to_ac97_data};
//


      wire decide;
   debounce b1(reset, clock_27mhz, button1, decide);

////////////////////////////////////////////////////////////////////////
//////////////////
////////////////////////////////////////////////////////////////////////
//////////////////
// Changes made to Lab 4 Audio by Team 15
```

```
//////////////////////////////////////////////////////////////////////
//////////////////
//////////////////////////////////////////////////////////////////////
//////////////////
//////////////////////////////////////////////////////////////////////
//////////////////
//////////////////////////////////////////////////////////////////////
//////////////////
       wire [7:0] xk_re;
    wire [7:0] xk_im;
       wire [7:0] xk_re2;
    wire [7:0] xk_im2;
    wire  done;

      reg [7:0] temp_input;

      always @(posedge clock_27mhz)
            begin
                  temp_input <= from_ac97_data;
            end

      primary_FFT pfft1(ac97_bit_clock, clock_27mhz, temp_input, reset,
ready, xk_re, xk_im, xk_re2, xk_im2, done);


// compute sum of squares of FFT output real and imaginary parts
//    to get a power spectrum

   wire [15:0] re_sq, im_sq;
   fft_mult m8m_signed1(clk,xk_re,xk_re,re_sq);
   fft_mult m8m_signed2(clk,xk_im,xk_im,im_sq);

   wire [16:0] fft_out_sum_sq = re_sq + im_sq;

      reg [16:0] x0sq;
      reg [16:0] x1sq;
      reg [16:0] x2sq;
      reg [16:0] x3sq;
      reg [16:0] x4sq;
      reg [16:0] x5sq;
      reg [16:0] x6sq;
      reg [16:0] x7sq;

      reg [3:0] count;

      reg done_d;
      reg done_1;
      reg done_2;
      reg done_3;
      reg done_4;

      reg disp_ready;

      ///// PRIMARY MDCT //////
      wire [7:0] DIN;
      wire [7:0] DOUT;
      wire wr_en;
```

```
      wire rd_en;
      wire prog_full;
      wire start;

      fifo2 fifo1(.din(DIN),.dout(DOUT),.wr_en(wr_en),
            .rd_en(rd_en), .clk(clock_27mhz),
            .prog_full(prog_full),.rst(reset));

      fifo_controller fc1(.clock_27mhz(clock_27mhz), .reset(reset),
            .from_ac97_data(from_ac97_data), .ready(ready),
.prog_full(prog_full),
            .rd_en(rd_en),.data_r(DIN), .ready_r(wr_en),
.start(start));

      wire [63:0] coefficients;
      wire [7:0] to_mdct_data;
      wire flush;
      wire shift;
      wire c_avail;
      wire [7:0] X0;
      wire [7:0] X1;
      wire [7:0] X2;
      wire [7:0] X3;
      wire [7:0] X4;
      wire [7:0] X5;
      wire [7:0] X6;
      wire [7:0] X7;
      wire [7:0] Xall;

      fifoANDmdct fANDm1(.clock_27mhz(clock_27mhz), .reset(reset),
.start(start), .from_fifo_data(DIN),
            .to_mdct_data(to_mdct_data), .flush(flush), .shift(shift),
.coefficients(coefficients), .c_avail(c_avail),
            .X0(X0), .X1(X1), .X2(X2), .X3(X3), .X4(X4), .X5(X5),
.X6(X6), .X7(X7), .Xall(Xall));


      reg [63:0] coefficients2;


      wire [7:0] x_0new;
      wire [7:0] x_1new;
      wire [7:0] x_2new;
      wire [7:0] x_3new;
      wire [7:0] x_4new;
      wire [7:0] x_5new;
      wire [7:0] x_6new;
      wire [7:0] x_7new;


////////////////  Implement the Visualiser and Equaliser -> these could
potentially go in different modules



      fft_mult eq_mult0(clk,coefficients[63:56],8'd2,x_0new);
```

```
      fft_mult eq_mult1(clk,coefficients[55:48],8'd2,x_1new);
      fft_mult eq_mult2(clk,coefficients[47:40],8'd2,x_2new);
      fft_mult eq_mult3(clk,coefficients[39:32],8'd2,x_3new);
      fft_mult eq_mult4(clk,coefficients[31:24],8'd2,x_4new);
      fft_mult eq_mult5(clk,coefficients[23:16],8'd2,x_5new);
      fft_mult eq_mult6(clk,coefficients[15:8],8'd2,x_6new);
      fft_mult eq_mult7(clk,coefficients[7:0],8'd2,x_7new);

      always @(posedge clock_27mhz)
            begin
                  if (decide)
                        begin
                              done_1 <= done;
                              done_2 <= done_1;
                              done_3 <= done_2;
                              done_4 <= done_3;
                              done_d <= done_4;
                                    if(done_d)
                                          begin
                                                if (count == 8)
                                                      begin
                                                            count <= 0;
                                                            disp_ready
<= 1;
                                                      end
                                                else
                                                      begin
                                                            count <=
count + 1;
                                                      end
                                          end
                                    case (count)
                                          0:
                                                begin
                                                      x0sq <=
fft_out_sum_sq;
                                                      disp_ready <= 0;
                                                end
                                          1: x1sq <= fft_out_sum_sq;
                                          2: x2sq <= fft_out_sum_sq;
                                          3: x3sq <= fft_out_sum_sq;
                                          4: x4sq <= fft_out_sum_sq;
                                          5: x5sq <= fft_out_sum_sq;
                                          6: x6sq <= fft_out_sum_sq;
                                          7:
                                                begin
                                                      x7sq <=
fft_out_sum_sq;
                                                end
                                    endcase
                        end
                  else
                        begin
                              if (c_avail)
                                    begin
                                          if (~switch[7])
                                                begin
```

```
coefficients2[63:56] <= x_0new;
                                        end
                                    else
                                        begin

coefficients2[63:56] <= coefficients[63:56];
                                        end
                                    if (~switch[6])
                                        begin

coefficients2[55:48] <= x_1new;
                                        end
                                    else
                                        begin

coefficients2[55:48] <= coefficients[55:48];
                                        end
                                    if (~switch[5])
                                        begin


coefficients2[47:40] <= x_2new;
                                        end
                                    else
                                        begin

coefficients2[47:40] <= coefficients[47:40];
                                        end
                                    if (~switch[4])
                                        begin


coefficients2[39:32] <= x_3new;
                                        end
                                    else
                                        begin

coefficients2[39:32] <= coefficients[39:32];
                                        end
                                    if (~switch[3])
                                        begin


coefficients2[31:24] <= x_4new;
                                        end
                                    else
                                        begin

coefficients2[31:24] <= coefficients[31:24];
                                        end
                                    if (~switch[2])
                                        begin


coefficients2[23:16] <= x_5new;
                                        end
```

```
                                        else
                                              begin

      coefficients2[23:16] <= coefficients[23:16];
                                              end
                                        if (~switch[1])
                                              begin


      coefficients2[15:8] <= x_6new;
                                              end
                                        else
                                              begin

      coefficients2[15:8] <= coefficients[15:8];
                                              end
                                        if (~switch[0])
                                              begin


      coefficients2[7:0] <= x_7new;
                                              end
                                        else
                                              begin

      coefficients2[7:0] <= coefficients[7:0];
                                              end
                                  end
                        end
            end

            wire wr_en2;
      wire [7:0] temp_out;

      reg c_avail2;

      always @(posedge clock_27mhz)
            begin
                  c_avail2 <= c_avail;
            end


      imdctANDfifo_controller iANDf_c(.clock_27mhz(clock_27mhz),
.reset(reset),
            .coefficients(coefficients2), .c_avail(c_avail2),
            .to_fifo_data(temp_out), .wr_en(wr_en2));


 wire [7:0] temp_recorder;
 assign temp_recorder = decide ? from_ac97_data : temp_out;
 recorder r(clock_27mhz, reset, playback, ready, temp_recorder,
to_ac97_data);


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

```
   // Lab 4 Components
   //

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

   //
   // Generate a 31.5MHz pixel clock from clock_27mhz
   //

   wire pclk, pixel_clock;
   DCM pixel_clock_dcm (.CLKIN(clock_27mhz), .CLKFX(pclk));
   // synthesis attribute CLKFX_DIVIDE of pixel_clock_dcm is 6
   // synthesis attribute CLKFX_MULTIPLY of pixel_clock_dcm is 7
   // synthesis attribute CLK_FEEDBACK of pixel_clock_dcm  is "NONE"
   BUFG pixel_clock_buf (.I(pclk), .O(pixel_clock));

   //
   // VGA output signals
   //

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

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

   // The following assignments should be deleted and replaced with
your own
   // code to implement the Pong game.
//   assign vga_out_red = 8'h0;
//   assign vga_out_green = 8'h0;
//   assign vga_out_blue = 8'h0;
//   assign vga_out_blank_b = sync  1'b1;
//   assign vga_out_hsync = 1'b0;
//   assign vga_out_vsync = 1'b0;


      wire up_sync;
      wire down_sync;
      wire reset_sync;

      wire up;
      wire down;
      wire reset2;
      wire [23:0] RGB;

      assign reset2 = ~button0;
      assign up = ~button_up;
      assign down = ~button_down;
```

```
      wire [1:0] speed_x;
      wire [1:0] speed_y;

      wire [9:0] pixel_count;
      wire [9:0] line_count;

      wire [8:0] paddle_y;
      wire [9:0] ball_x;
      wire [8:0] ball_y;

      assign vga_out_red = RGB[23:16];
   assign vga_out_green = RGB[15:8];
   assign vga_out_blue = RGB[7:0];

      assign speed_x = switch[1:0];
      assign speed_y = switch[3:2];

      //assign ball_x  = 10'b0110010000;
      //assign ball_y  = 9'b010000000;
      //assign paddle_y = 9'b011011100;

      wire [7:0] signals;
      wire [2:0] state_from_ball;

      //assign led = signals;

      synchroniser
synchroniser1(reset2,up,down,pixel_clock,reset_sync);




      DisplayField df(clock_27mhz, pixel_clock, reset_sync,
            x0sq, x1sq, x2sq, x3sq, x4sq, x5sq, x6sq, x7sq,
            pixel_count, line_count, RGB, disp_ready);
      vga vga1(pixel_clock, reset_sync, vga_out_hsync, vga_out_vsync,
vga_out_sync_b, vga_out_blank_b, pixel_count, line_count);


//assign analyzer1_clock = clock_27mhz;
//assign analyzer1_data[0] = xk_re;
//assign analyzer1_data[1] = xk_im;
//assign analyzer1_data[2] = xk_re2
//assign analyzer1_data[3] = xk_im2;
//assign analyzer1_data[4] = done;


//    Alphanumeric Display

assign clk = clock_27mhz;

              reg [63:0] dispdata;
        reg [30:0] counta;
        reg en;
//        always @(posedge clock_27mhz)
//                begin
//                        if (counta == 30'd10000000)
```

```
//                                  begin
//                                          counta <= 30'b0;
//                                          en <= 1;
//                                  end
//                            else if (counta == 30'b0)
//                                  begin
//                                          en <= 0;
//                                          counta <= counta + 1;
//                                  end
//                            else
//                                  begin
//                                          counta <= counta + 1;
//                                  end
//                            if (en)
//                                  begin
//                                          dispdata <=
{x0sq[16:9],x1sq[16:9],x2sq[16:9],x3sq[16:9],x4sq[16:9],x5sq[16:9],x6sq
[16:9],x7sq[16:9]};
//                                  end
//                               end

                                always @(posedge clock_27mhz)
                    begin
                                    if (decide)
                                          begin
                            if (counta == 30'd10000000)
                                    begin
                                          counta <= 30'b0;
                                          en <= 1;
                                    end
                            else if (counta == 30'b0)
                                    begin
                                          en <= 0;
                                          counta <= counta + 1;
                                    end
                            else
                                    begin
                                          counta <= counta + 1;
                                    end
                            if (en)
                                    begin
                                          dispdata <=
{x0sq[16:9],x1sq[16:9],x2sq[16:9],x3sq[16:9],x4sq[16:9],x5sq[16:9],x6sq
[16:9],x7sq[16:9]};
                                    end
                                          end
                                       else
                                          begin
                                                if (counta ==
30'd10000000)
                                    begin
                                          counta <= 30'b0;
                                          en <= 1;
                                    end
                            else if (counta == 30'b0)
                                    begin
                                          en <= 0;
```

```
                                        counta <= counta + 1;
                                end
                        else

                                begin
                                        counta <= counta + 1;
                                end
                        if (en)
                                begin
                                        dispdata <= coefficients2;
                                end
                                  end
                              end


   display_16hex d1(reset, clk, dispdata,
                disp_blank, disp_clock, disp_rs, disp_ce_b,
                disp_reset_b, disp_data_out);
endmodule

////////////////////////////////////////////////////////////////////////
//////////
////
//// Record/playback
////
////////////////////////////////////////////////////////////////////////
//////////
//
//module recorder(clock_27mhz, reset, playback, ready, from_ac97_data,
to_ac97_data);
//   input clock_27mhz;              // 27mhz system clock
//   input reset;                    // 1 to reset to initial state
//   input playback;                 // 1 for playback, 0 for record
//   input ready;                    // 1 when AC97 data is available
//   input [7:0] from_ac97_data;     // 8-bit PCM data from mic
//   output [7:0] to_ac97_data;      // 8-bit PCM data to headphone
//
//   // test: playback 750hz tone, or loopback using incoming data
//   wire [19:0] tone;
//   tone750hz xxx(clock_27mhz, ready, tone);
//   reg [7:0] to_ac97_data;
//   always @ (posedge clock_27mhz) begin
//      if (ready) begin
//     // just received new data from the AC97
//     to_ac97_data <= playback ? tone[19:12] : from_ac97_data;
//      end
//   end
//
//endmodule

`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////
//////////
// Company:
// Engineer:
//
// Create Date:    20:07:56 05/01/2007
```

```
// Design Name:
// Module Name:    block_mem_fft_controller
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////
///////////
module block_mem_fft_controller(clock_27mhz, reset, from_ac97_data,
ready, prog_full, ready_r, data_r, rd_en, start, fwd_inv, fwd_inv_we);
      input reset;
      input clock_27mhz;
      input [7:0] from_ac97_data;
      input prog_full;
      input ready;


      output reg fwd_inv;
      output reg fwd_inv_we;


      output reg ready_r;
      output reg rd_en;
      output reg [7:0] data_r;
      output reg start;
      reg [7:0] r1;
      reg [7:0] r2;
      reg [7:0] r3;
      reg [7:0] r4;
      reg [7:0] r5;


//    always @(posedge ac97_bit_clock)
//          begin
//                if (reset)
//                      begin
//                            wr_en <= 0;
//                      end
//                if (ready)
//                      begin
//                            wr_en <= 1;
//                            DIN <= from_ac97_data;
//                      end
//                else
//                      begin
//                            wr_en <= 0;
//                      end
//          end
//
```

```
//    always @(posedge clock_27mhz)
//          begin
//                if (reset)
//                      begin
//                            rd_en <= 0;
//                            wr_en <= 0;
//                      end
//                if (ready)
//                      begin
//                            wr_en <= 1;
//                            DIN <= from_ac97_data;
//                      end
//                else
//                      begin
//                            wr_en <= 0;
//                      end
//                if (prog_full)
//                      begin
//                            rd_en <= 1;
//                            start <= 1;
//                      end
//                else
//                      begin
//                            rd_en <= 0;
//                            start <= 0;
//                      end
//          end

      reg ready_q;
      reg ready_d;
      reg [7:0] data_q;
      reg [7:0] count;

      parameter max_point = 8;

      parameter start_state = 0;
      parameter FFT = 1;

      reg [1:0] state;

//    always @(ready_d or ready)
//          begin
//                ready_q = ready_d ^ ready;
//          end

      always @(posedge clock_27mhz)
            begin
                  data_r <= data_q;
                  data_q <= from_ac97_data;
                  ready_r <= ready_d;
                  //ready_q <= ready_d;
                  ready_d <= ready;

                  if (reset)
                        begin
                              state <= start_state;
                              start <= 0;
```

```
                        count <= 0;
                        fwd_inv <= 1;
                        fwd_inv_we <=1;
                  end

            case(state)
                  start_state:
                        begin
                              if (prog_full)
                                    begin
                                          start <= 1;
                                          //////////

                                          /////////
                                          rd_en <= 1;
                                          count <= count + 1;
                                          state <= FFT;
                                    end
                              else
                                    begin
                                          start <= 0;
                                          state <= start_state;
                                          rd_en <= 0;
                                    end
                        end
                  FFT:
                        begin
                              if(count !== max_point)
                                    begin
                                          count <= count + 1;
                                          state <= FFT;
                                          rd_en <= 1;
                                    end
                              else
                                    begin
                                          start <= 0;
                                          count <= 0;
                                          state <= start_state;
                                          rd_en <= 0;
                                    end
                        end
            endcase
      end

endmodule



`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//////////
// Company:
// Engineer:
//
// Create Date:    23:06:22 05/03/2007
// Design Name:
// Module Name:    block_mem_ifft_controller
```

```
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
///////////
module block_mem_ifft_controller(clock_27mhz, reset, xn_im, xn_re,
ready, start, fwd_inv2, fwd_inv_we2);
    input clock_27mhz;
    input reset;
    input [7:0] xn_im;
    input [7:0] xn_re;
    input ready;

       output reg start;
       output reg fwd_inv2;
       output reg fwd_inv_we2;

       reg [7:0] count;
       reg [1:0] state;

       parameter max_point = 8;

       parameter start_state = 0;
       parameter ifft = 1;

       always @(posedge clock_27mhz)
            begin
                  if (reset)
                        begin
                              state <= start_state;
                              start <= 0;
                              count <= 0;
                        end

                  case(state)
                        start_state:
                              begin
                                    if (ready)
                                          begin
                                                state <= ifft;
                                                count <= count + 1;
                                                start <= 1;
                                                fwd_inv2 <= 0;
                                                fwd_inv_we2 <= 1;
                                          end
                              end
                        ifft:
                              begin
                                    if (count == max_point)
```

```
                                        begin
                                              count <= 0;
                                              state <= start_state;
                                              start <= 0;
                                        end
                                  else
                                        begin
                                              count <= count + 1;
                                              state <= ifft;
                                              start <= 1;
                                        end
                              end
                  endcase
      end
endmodule



//////////////////////////////////////////////////////////////////////
////////
//
// 6.111 FPGA Labkit -- Hex display driver
//
//
// File:   display_16hex.v
// Date:   24-Sep-05
//
// Created: April 27, 2004
// Author: Nathan Ickes
//
// This module drives the labkit hex displays and shows the value of
// 8 bytes (16 hex digits) on the displays.
//
// 24-Sep-05 Ike: updated to use new reset-once state machine, remove
clear
//
// Inputs:
//
//   reset       - active high
//   clock_27mhz - the synchronous clock
//   data        - 64 bits; each 4 bits gives a hex digit
//
// Outputs:
//
//    disp_*     - display lines used in the 6.111 labkit (rev 003 &
004)
//
//////////////////////////////////////////////////////////////////////
////////

module display_16hex (reset, clock_27mhz, data,
            disp_blank, disp_clock, disp_rs, disp_ce_b,
            disp_reset_b, disp_data_out);

   input reset, clock_27mhz;    // clock and reset (active high reset)
   input [63:0] data;         // 16 hex nibbles to display
```

```
   output disp_blank, disp_clock, disp_data_out, disp_rs, disp_ce_b,
        disp_reset_b;

   reg disp_data_out, disp_rs, disp_ce_b, disp_reset_b;


////////////////////////////////////////////////////////////////////////
/////
   //
   // Display Clock
   //
   // Generate a 500kHz clock for driving the displays.
   //

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

   reg [4:0] count;
   reg [7:0] reset_count;
   reg clock;
   wire dreset;

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

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

   assign dreset = (reset_count != 0);

   assign disp_clock = ~clock;


////////////////////////////////////////////////////////////////////////
/////
   //
   // Display State Machine
   //
```

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

   reg [7:0] state;           // FSM state
   reg [9:0] dot_index;       // index to current dot being clocked out
   reg [31:0] control;        // control register
   reg [3:0] char_index;      // index of current character
   reg [39:0] dots;           // dots for a single digit
   reg [3:0] nibble;          // hex nibble of current character

   assign disp_blank = 1'b0; // low <= not blanked

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

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

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

       8'h03:
         begin
            // Latch dot data
            disp_ce_b <= 1'b1;
            dot_index <= 31;        // re-purpose to init ctrl reg
            state <= state+1;
```

```
      end

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

    8'h05:
      begin
         // Latch the control register data / dot data
         disp_ce_b <= 1'b1;
         dot_index <= 39;        // init for single char
         char_index <= 15;       // start with MS char
         state <= state+1;
      end

    8'h06:
      begin
         // Load the user's dot data into the dot reg, char by char
         disp_rs <= 1'b0;              // Select the dot register
         disp_ce_b <= 1'b0;
         disp_data_out <= dots[dot_index]; // dot data from msb
         if (dot_index == 0)
           if (char_index == 0)
             state <= 5;               // all done, latch data
         else
         begin
           char_index <= char_index - 1;     // goto next char
           dot_index <= 39;
         end
         else
         dot_index <= dot_index-1;     // else loop thru all dots
      end

    endcase

always @ (data or char_index)
  case (char_index)
    4'h0:            nibble <= data[3:0];
    4'h1:            nibble <= data[7:4];
    4'h2:            nibble <= data[11:8];
    4'h3:            nibble <= data[15:12];
    4'h4:            nibble <= data[19:16];
    4'h5:            nibble <= data[23:20];
    4'h6:            nibble <= data[27:24];
    4'h7:            nibble <= data[31:28];
    4'h8:            nibble <= data[35:32];
    4'h9:            nibble <= data[39:36];
    4'hA:            nibble <= data[43:40];
```

```
        4'hB:            nibble <= data[47:44];
        4'hC:            nibble <= data[51:48];
        4'hD:            nibble <= data[55:52];
        4'hE:            nibble <= data[59:56];
        4'hF:            nibble <= data[63:60];
      endcase

   always @(nibble)
      case (nibble)
        4'h0: dots <= 40'b00111110_01010001_01001001_01000101_00111110;
        4'h1: dots <= 40'b00000000_01000010_01111111_01000000_00000000;
        4'h2: dots <= 40'b01100010_01010001_01001001_01001001_01000110;
        4'h3: dots <= 40'b00100010_01000001_01001001_01001001_00110110;
        4'h4: dots <= 40'b00011000_00010100_00010010_01111111_00010000;
        4'h5: dots <= 40'b00100111_01000101_01000101_01000101_00111001;
        4'h6: dots <= 40'b00111100_01001010_01001001_01001001_00110000;
        4'h7: dots <= 40'b00000001_01110001_00001001_00000101_00000011;
        4'h8: dots <= 40'b00110110_01001001_01001001_01001001_00110110;
        4'h9: dots <= 40'b00000110_01001001_01001001_00101001_00011110;
        4'hA: dots <= 40'b01111110_00001001_00001001_00001001_01111110;
        4'hB: dots <= 40'b01111111_01001001_01001001_01001001_00110110;
        4'hC: dots <= 40'b00111110_01000001_01000001_01000001_00100010;
        4'hD: dots <= 40'b01111111_01000001_01000001_01000001_00111110;
        4'hE: dots <= 40'b01111111_01001001_01001001_01001001_01000001;
        4'hF: dots <= 40'b01111111_00001001_00001001_00001001_00000001;
      endcase

endmodule




`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
////////////
// Company:
// Engineer: Rahul Shroff
//
// Create Date:    17:37:55 04/02/2007
// Design Name:
// Module Name:    DisplayField
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
////////////

module DisplayField(clock_27mhz,pixel_clock, reset_sync,
```

```
  x0sq,
  x1sq,
  x2sq,
  x3sq,
  x4sq,
  x5sq,
  x6sq,
  x7sq,
  pixel_count, line_count, RGB, disp_ready);


        input clock_27mhz;
input pixel_clock;
   input reset_sync;
input [9:0] pixel_count;
input [9:0] line_count;
output [23:0] RGB;
   input disp_ready;

  input [16:0] x0sq;
  input [16:0] x1sq;
  input [16:0] x2sq;
  input [16:0] x3sq;
  input [16:0] x4sq;
  input [16:0] x5sq;
  input [16:0] x6sq;
  input [16:0] x7sq;

  reg [16:0] x0sqa;
  reg [16:0] x1sqa;
  reg [16:0] x2sqa;
  reg [16:0] x3sqa;
  reg [16:0] x4sqa;
  reg [16:0] x5sqa;
  reg [16:0] x6sqa;
  reg [16:0] x7sqa;


  wire [23:0] RGB_i0;
  wire [23:0] RGB_i1;
  wire [23:0] RGB_i2;
  wire [23:0] RGB_i3;
  wire [23:0] RGB_i4;
  wire [23:0] RGB_i5;
  wire [23:0] RGB_i6;
  wire [23:0] RGB_i7;


   reg [63:0] dispdata;
    reg [30:0] countb;
    reg en;
    always @(posedge clock_27mhz)
            begin
                    if (countb == 30'd6000000)
                            begin
                                    countb <= 30'b0;
                                    en <= 1;
```

```
                        end
                    else if (countb == 30'b0)
                        begin
                            en <= 0;
                            countb <= countb + 1;
                        end
                    else
                        begin
                            countb <= countb + 1;
                        end
                    if (en)
                        begin
                           x0sqa <= x0sq;

      x1sqa <= x1sq;

      x2sqa <= x2sq;

      x3sqa <= x3sq;

      x4sqa <= x4sq;

      x5sqa <= x5sq;

      x6sqa <= x6sq;

      x7sqa <= x7sq;
                        end
                    end


//I instantiate various rectangles from my rectangle module to create
the required display of the MIT logo and the ball.

      rectangle i0(pixel_clock,
pixel_count,line_count,100,50,RGB_i0,1'b1, x0sqa[15:6]);
//rectangle i0(pixel_clock, pixel_count,line_count,100,50,RGB_i0,1'b1,
30);
      defparam i0.WIDTH = 39;
   //defparam i0.HEIGHT = x0sq >> 5;
   defparam i0.COLOUR = 24'd16774215;

      rectangle i1(pixel_clock,
pixel_count,line_count,150,50,RGB_i1,1'b1, x1sqa[15:6]);
//    rectangle i1(pixel_clock,
pixel_count,line_count,150,50,RGB_i1,1'b1, 39);
      defparam i1.WIDTH = 39;
//   defparam i1.HEIGHT = x1sq >> 5;
   defparam i1.COLOUR = 24'd16477215;

      rectangle i2(pixel_clock,
pixel_count,line_count,200,50,RGB_i2,1'b1, x2sqa[15:6]);
//    rectangle i2(pixel_clock,
pixel_count,line_count,200,50,RGB_i2,1'b1, 45);
      defparam i2.WIDTH = 39;
//   defparam i2.HEIGHT = x2sq >> 5;
   defparam i2.COLOUR = 24'd11777245;
```

```
      rectangle i3(pixel_clock,
pixel_count,line_count,250,50,RGB_i3,1'b1, x3sqa[15:6]);
//    rectangle i3(pixel_clock,
pixel_count,line_count,250,50,RGB_i3,1'b1, 50);
      defparam i3.WIDTH = 39;
//   defparam i3.HEIGHT = x3sq >> 5;
   defparam i3.COLOUR = 24'd14555215;

      rectangle i4(pixel_clock,
pixel_count,line_count,300,50,RGB_i4,1'b1, x4sqa[15:6]);
//    rectangle i4(pixel_clock,
pixel_count,line_count,300,50,RGB_i4,1'b1, 55);
      defparam i4.WIDTH = 39;
//   defparam i4.HEIGHT = x4sq >> 5;
   defparam i4.COLOUR = 24'd16174215;
//
      rectangle i5(pixel_clock,
pixel_count,line_count,350,50,RGB_i5,1'b1, x5sqa[15:6]);
//    rectangle i5(pixel_clock,
pixel_count,line_count,350,50,RGB_i5,1'b1, 60);
      defparam i5.WIDTH = 39;
//   defparam i5.HEIGHT = x5sq >> 5;
   defparam i5.COLOUR = 24'd16274145;
//
      rectangle i6(pixel_clock,
pixel_count,line_count,400,50,RGB_i6,1'b1, x6sqa[15:6]);
//    rectangle i6(pixel_clock,
pixel_count,line_count,400,50,RGB_i6,1'b1, 65);
      defparam i6.WIDTH = 39;
//   defparam i6.HEIGHT = x6sq >> 5;
   defparam i6.COLOUR = 24'd16747215;

      rectangle i7(pixel_clock,
pixel_count,line_count,450,50,RGB_i7,1'b1, x7sqa[15:6]);
//    rectangle i7(pixel_clock,
pixel_count,line_count,450,50,RGB_i7,1'b1, 70);
      defparam i7.WIDTH = 39;
//   defparam i7.HEIGHT = x7sq >> 5;
   defparam i7.COLOUR = 24'd16944444;


      assign RGB = RGB_i0 | RGB_i1 | RGB_i2 | RGB_i3 | RGB_i4 | RGB_i5
| RGB_i6 | RGB_i7;

//    always @(posedge pixel_clock)
//          begin
//                if (disp_ready)
//                      begin
//
//                      end
//                //assign RGB = 24'db111111111111111111111111;
//                //assign RGB = RGB_t1;
//          end


endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//////////
// Company:
// Engineer:
//
// Create Date:    22:41:47 05/01/2007
// Design Name:
// Module Name:    primary_FFT
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
//////////
module primary_FFT(ac97_bit_clock, clock_27mhz, from_ac97_data, reset,
ready, xk_re, xk_im, xk_re2, xk_im2, done);
    input ac97_bit_clock;
       input clock_27mhz;
    input [7:0] from_ac97_data;
    input reset;
    input ready;
    output [7:0] xk_re;
    output [7:0] xk_im;
       output [7:0] xk_re2;
    output [7:0] xk_im2;
    output done;

       wire [7:0] DIN;

       wire [9:0] ADDR;
//     wire [9:0] ADDR2;
    wire WE;
    wire ready_fsm;

       wire reset;
    wire clock_27mhz;
    wire [7:0] DOUT;
    wire start;
    wire fwd_inv ;
    wire fwd_inv_we;
    wire [7:0] xn_re;

      wire rfd;
      wire busy;
      wire dv;
```

```
      wire edone;

      wire rfd2;
      wire busy2;
      wire dv2;
      wire edone2;
      wire fwd_inv2;
      wire fwd_inv_we2;
      wire start2;
      wire done2;

      wire [2:0] xn_index;
      wire [2:0] xk_index;

      wire [2:0] xn_index2;
      wire [2:0] xk_index2;

      wire [7:0] bmfc_xn_re;
      wire wr_en;
      wire rd_en;
      wire prog_full;
      wire RST;


      reg [7:0]DOUT_delayed;
      reg [7:0]xk_re_delayed;
      reg [7:0]xk_im_delayed;
      reg [7:0]r1,r2,r3,r4,r5;
      reg [7:0]r6,r7,r8,r9,r10;
      reg [7:0]r11,r12,r13,r14;
      //wire DOUT;

//    wire done;
//     block_mem_fft
bmf1(.addr(ADDR),.clk(clock_27mhz),.din(DIN),.dout(DOUT), .we(WE));
       fifo_generator_v2_3 fifo1(.din(DIN),.dout(DOUT),.wr_en(wr_en),
.rd_en(rd_en), .clk(clock_27mhz), .prog_full(prog_full),.rst(reset));
//     block_mem_fft
bmf2(.addr(ADDR2),.clk(clock_27mhz),.din(DIN),.dout(DOUT), .we(WE));

       block_mem_fft_controller bmfc1(.clock_27mhz(clock_27mhz),
.reset(reset),
            .from_ac97_data(from_ac97_data), .ready(ready),
.prog_full(prog_full),
            .rd_en(rd_en),.data_r(DIN), .ready_r(wr_en),
.start(start));
       //fsm_fft_controller ffc(.reset(reset),
.clock_27mhz(clock_27mhz), .ready_fsm(ready_fsm),
.bmfc_xn_re(bmfc_xn_re), .start(start), .fwd_inv(fwd_inv),
.fwd_inv_we(fwd_inv_we), .xn_re(xn_re));
//     fsm_fft_controller ffc(.reset(reset), .clock_27mhz(clock_27mhz),
.ready_fsm(ready_fsm), .DOUT(DOUT), .ADDR(ADDR2), .start(start),
.fwd_inv(fwd_inv), .fwd_inv_we(fwd_inv_we), .xn_re(xn_re));
       fft fft_forward(.xn_index(xn_index), .xk_index(xk_index),
.rfd(rfd),
                  .edone(edone), .xk_re(xk_re), .xk_im(xk_im),
.busy(busy), .dv(dv), .done(done),
```

```
                  .fwd_inv(fwd_inv), .fwd_inv_we(fwd_inv_we),
.xn_re(DOUT_delayed), .xn_im(8'b0),.clk(clock_27mhz),.start(start));

       block_mem_ifft_controller bmifc1(.clock_27mhz(clock_27mhz),
.reset(reset), .xn_re (xk_re), .xn_im(xk_im), .ready(done),
.start(start2), .fwd_inv2(fwd_inv2), .fwd_inv_we2(fwd_inv_we2));

       fft fft2(.xn_index(xn_index2), .xk_index(xk_index2), .rfd(rfd2),
                  .edone(edone2), .xk_re(xk_re2), .xk_im(xk_im2),
.busy(busy2), .dv(dv2), .done(done2),
                  .fwd_inv(fwd_inv2), .fwd_inv_we(fwd_inv_we2),
.xn_re(xk_re_delayed),
.xn_im(xk_im_delayed),.clk(clock_27mhz),.start(start2));

                  always @ (posedge clock_27mhz)
                        begin
                              r1 <= DOUT;
                              r2 <= r1;
                              r3 <= r2;
                              r4 <= r3;
                              DOUT_delayed <= r4;
                        end
//
      always @ (posedge clock_27mhz)
                        begin
                              r6 <= xk_re;
                              r7 <= r6;
                              r8 <= r7;
                              r9 <= r8;
                              xk_re_delayed <= r9;
                        end

always @ (posedge clock_27mhz)
                        begin
                              r11 <= xk_im;
                              r12 <= r11;
                              r13 <= r12;
                              r14 <= r13;
                              xk_im_delayed <= r14;
                        end

endmodule



`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
////////////
// Company:
// Engineer:
//
// Create Date:    14:30:37 04/29/2007
// Design Name:
// Module Name:    recirder
// Project Name:
// Target Devices:
// Tool versions:
```

```
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
///////////
module recorder(clock_27mhz, reset, playback, ready, from_ac97_data,
to_ac97_data);
    input clock_27mhz;             // 27mhz system clock
   input reset;                   // 1 to reset to initial state
   input playback;                // 1 for playback, 0 for record
   input ready;                   // 1 when AC97 data is available
   input [7:0] from_ac97_data;    // 8-bit PCM data from mic
   output [7:0] to_ac97_data;     // 8-bit PCM data to headphone

   // test: playback 750hz tone, or loopback using incoming data
   wire [19:0] tone;
   tone750hz xxx(clock_27mhz, ready, tone);
   reg [7:0] to_ac97_data;
   always @ (posedge clock_27mhz) begin
      if (ready) begin
       // just received new data from the AC97
       to_ac97_data <= playback ? tone[19:12] : from_ac97_data;
//     to_ac97_data <= playback ? 8'b0 : from_ac97_data;
      end
   end
endmodule


`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
///////////
// Company:
// Engineer:
//
// Create Date:    19:34:59 04/02/2007
// Design Name:
// Module Name:    rectangle
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
///////////
module rectangle(pixel_clock, x, y, rect_x, rect_y, RGB,
vga_out_sync_b, HEIGHT);
```

```
       input pixel_clock;
    input [9:0] x;
    input [8:0] y;
    input [9:0] rect_x;
    input [8:0] rect_y;
    input vga_out_sync_b;

       input [9:0] HEIGHT;
    output [23:0] RGB;



      //We need to declare RGB as a register for use in the always
block

      reg [23:0] RGB = 24'b0;

      parameter WIDTH = 39; // default width: 39 pixels
//    parameter HEIGHT = 195; // default height: 195 pixels
      parameter COLOUR = 24'h0;


// We check to see if it is within the rectangle and then colour
accordingly. If not we wet the colour to black.


      always @(posedge pixel_clock)
            begin

                  if (((x > rect_x) && (x <= rect_x + WIDTH)) && ((y >
rect_y) && (y <= rect_y + HEIGHT + 50)))

                        begin
                              RGB <= COLOUR;
                        end

                  else

                        begin
                              RGB <= 24'b0;
                        end

            end

endmodule



`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////
////////
// Company:
// Engineer:
```

```
//
// Create Date:   21:00:01 05/02/2007
// Design Name:   block_mem_fft_controller
// Module Name:
C:/lkini/FINAL/tests/audiotest/tb_block_mem_fft_controller.v
// Project Name:  audiotest
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module:
block_mem_fft_controller
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////
/////////

module tb_block_mem_fft_controller_v;

      // Inputs
      reg clock_27mhz;
      reg reset;
      reg [7:0] from_ac97_data;
      reg ready;

      // Outputs
      wire [7:0] DIN;
      wire [9:0] ADDR;
      wire WE;
      wire ready_fsm;
      wire xn_re;

      // Instantiate the Unit Under Test (UUT)
      block_mem_fft_controller uut (
            .clock_27mhz(clock_27mhz),
            .reset(reset),
            .from_ac97_data(from_ac97_data),
            .ready(ready),
            .DIN(DIN),
            .ADDR(ADDR),
            .WE(WE),
            .ready_fsm(ready_fsm),
            .xn_re(xn_re)
      );

      always #10 clock_27mhz = ~clock_27mhz;

      initial begin
            // Initialize Inputs
            clock_27mhz = 0;
            reset = 0;
            from_ac97_data = 0;
```

```
ready = 0;

// Wait 100 ns for global reset to finish
#100;

// Add stimulus here
#20;
reset = 1;
#20;
reset = 0;
#20;

#100;
from_ac97_data = 8'b01010101;
ready = 1;
#20;
ready = 0;

#80;
from_ac97_data = 8'b01110101;
ready = 1;
#20;
ready = 0;

from_ac97_data = 8'b01011101;
ready = 1;
#20;
ready = 0;

from_ac97_data = 8'b11110101;
ready = 1;
#20;
ready = 0;

from_ac97_data = 8'b01000001;
ready = 1;
#20;
ready = 0;

from_ac97_data = 8'b01000100;
ready = 1;
#20;
ready = 0;

from_ac97_data = 8'b00000111;
ready = 1;
#20;
ready = 0;

from_ac97_data = 8'b01001101;
ready = 1;
#20;
ready = 0;

from_ac97_data = 8'b11011111;
ready = 1;
#20;
```

```
            ready = 0;


      end

endmodule



`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
/////////
// Company:
// Engineer:
//
// Create Date:   18:01:33 05/12/2007
// Design Name:   fft
// Module Name:   C:/rshroff-final/audiotest/tb_FFT.v
// Project Name:  audiotest
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: fft
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////
/////////

module tb_FFT_v;

      // Inputs
      reg fwd_inv_we;
      reg start;
      reg fwd_inv;
      reg clk;
      reg [7:0] xn_re;
      reg [7:0] xn_im;

      // Outputs
      wire rfd;
      wire dv;
      wire done;
      wire busy;
      wire edone;
      wire [11:0] xk_im;
      wire [2:0] xn_index;
      wire [11:0] xk_re;
      wire [2:0] xk_index;
```

```
	// Instantiate the Unit Under Test (UUT)
	fft uut (
		.fwd_inv_we(fwd_inv_we),
		.rfd(rfd),
		.start(start),
		.fwd_inv(fwd_inv),
		.dv(dv),
		.done(done),
		.clk(clk),
		.busy(busy),
		.edone(edone),
		.xn_re(xn_re),
		.xk_im(xk_im),
		.xn_index(xn_index),
		.xk_re(xk_re),
		.xn_im(xn_im),
		.xk_index(xk_index)
	);
always #10 clk = ~clk;

	initial begin
		// Initialize Inputs
		fwd_inv_we = 0;
		start = 0;
		fwd_inv = 0;
		clk = 0;
		xn_re = 0;
		xn_im = 0;

		// Wait 100 ns for global reset to finish
		#100;

		// Add stimulus here


//		//start = 1;
////		start = 1;
////		#5;
//		fwd_inv_we = 1 ;
//		fwd_inv = 1;
////		fwd_inv_we = 0 ;
////
//		start = 1;
//		#5;
//
//		#75;

//		#25;
//		#10;
		#15
		start = 1;
		fwd_inv_we = 1 ;
		fwd_inv = 1;

		#65;
		#15;
```

```
//          xn_re = 8'd3;
//          #20;
//          xn_re = 8'b0;
//          #20;
//          xn_re = 8'b0;
//          #20;
//          xn_re = 8'd100;
//          #20;
//          xn_re = 8'b0;
//          #20;
//          xn_re = 8'b0;
//          #20;
//          xn_re = 8'b0;
//          #20;
//          xn_re = 8'b0;
//          #20;
//

            xn_re = -8'd68;
            xn_im = 8'd72;
            #20;
            xn_re = 8'd3;
            xn_im = -8'd100;
            #20;
            xn_re = 8'd75;
            xn_im = 8'd72;
            #20;
            xn_re = -8'd97;
            xn_im = 8'd0;
            #20;
            xn_re = 8'd74;
            xn_im = -8'd72;
            #20;
            xn_re = 8'd3;
            xn_im = 8'd100;
            #20;
            xn_re = -8'd69;
            xn_im = -8'd72;
            #20;
            xn_re = 8'd103;
            xn_im = 8'd0;
            #20;



            #100;
            start = 0;



            #200;
```

```
	end

endmodule




`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
////////
// Company:
// Engineer:
//
// Create Date:   23:14:12 05/03/2007
// Design Name:   primary_FFT
// Module Name:   C:/lkini/FINAL/tests/audiotest/tb_primary_FFT.v
// Project Name:  audiotest
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: primary_FFT
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////
////////

module tb_primary_FFT_v;

	// Inputs
	reg ac97_bit_clock;
	reg clock_27mhz;
	reg [7:0] from_ac97_data;
	reg reset;
	reg ready;

	// Outputs
	wire [7:0] xk_re;
	wire [7:0] xk_im;
	wire [7:0] xk_re2;
	wire [7:0] xk_im2;
	wire done;

	// Instantiate the Unit Under Test (UUT)
	primary_FFT uut (
		.ac97_bit_clock(ac97_bit_clock),
		.clock_27mhz(clock_27mhz),
		.from_ac97_data(from_ac97_data),
		.reset(reset),
```

```
	.ready(ready),
	.xk_re(xk_re),
	.xk_im(xk_im),
	.xk_re2(xk_re2),
	.xk_im2(xk_im2),
	.done(done)
);

always #10 clock_27mhz = ~clock_27mhz;

initial begin
	// Initialize Inputs
	ac97_bit_clock = 0;
	clock_27mhz = 0;
	from_ac97_data = 0;
	reset = 0;
	ready = 0;

	// Wait 100 ns for global reset to finish
	#100;

	// Add stimulus here
	#20;

	reset = 1;
	#20;
	reset = 0;
	#2000;
	//from_ac97_data = 8'b11000010;
	//from_ac97_data = 8'b01011111;
	from_ac97_data = 8'd3;
	ready = 1;
	#20;
	ready = 0;


	#1980;
	//from_ac97_data = 8'b1110100;
	//from_ac97_data = 8'b01011111;
	from_ac97_data = 8'd0;
	//from_ac97_data = 8'b0;
	ready = 1;
	#20;
	ready = 0;


	#1980;
	//from_ac97_data = 8'b100;
	//from_ac97_data = 8'b01011111;
	from_ac97_data = 8'b0;
	ready = 1;
	#20;
	ready = 0;


	#1980;
	//from_ac97_data = 8'b11010001;
```

```
//from_ac97_data = 8'b01011111;
from_ac97_data = 8'd100;
//from_ac97_data = 8'b0;
ready = 1;
#20;
ready = 0;


#1980;
//from_ac97_data = 8'b1110001;
from_ac97_data = 8'b0;
//from_ac97_data = 8'b01011111;
ready = 1;
#20;
ready = 0;


#1980;
//from_ac97_data = 8'b10011100;
from_ac97_data = 8'b0;
//from_ac97_data = 8'b01011111;
ready = 1;
#20;
ready = 0;


#1980;
//from_ac97_data = 8'b11001001;
from_ac97_data = 8'b0;
//from_ac97_data = 8'b01011111;
ready = 1;
#20;
ready = 0;


#1980;
//from_ac97_data = 8'b11101011;
from_ac97_data = 8'b0;
//from_ac97_data = 8'b01011111;
ready = 1;
#20;
ready = 0;


#1980;
from_ac97_data = 8'b01110000;
ready = 1;
#20;
ready = 0;


#1980;
from_ac97_data = 8'b01110000;
ready = 1;
#20;
ready = 0;
```

```
#1980;
from_ac97_data = 8'b01110000;
ready = 1;
#20;
ready = 0;


#1980;
from_ac97_data = 8'b01110000;
ready = 1;
#20;
ready = 0;


#1980;
from_ac97_data = 8'b0;
ready = 1;
#20;
ready = 0;


#1980;
from_ac97_data = 8'b0;
ready = 1;
#20;
ready = 0;


#1980;
from_ac97_data = 8'b0;
ready = 1;
#20;
ready = 0;


#1980;
from_ac97_data = 8'b0;
ready = 1;
#20;
ready = 0;


#1980;
from_ac97_data = 8'b1011001;
ready = 1;
#20;
ready = 0;


#1980;
from_ac97_data = 8'b11001111;
ready = 1;
#20;
ready = 0;
```

```
#1980;
from_ac97_data = 8'b10;
ready = 1;
#20;
ready = 0;


#1980;
from_ac97_data = 8'b100011;
ready = 1;
#20;
ready = 0;


#1980;
from_ac97_data = 8'b110011;
ready = 1;
#20;
ready = 0;


#1980;
from_ac97_data = 8'b110010;
ready = 1;
#20;
ready = 0;


#1980;
from_ac97_data = 8'b10011001;
ready = 1;
#20;
ready = 0;


#1980;
from_ac97_data = 8'b1000101;
ready = 1;
#20;
ready = 0;


#1980;
from_ac97_data = 8'b110010;
ready = 1;
#20;
ready = 0;


#1980;
from_ac97_data = 8'b11;
ready = 1;
#20;
ready = 0;


#1980;
```

```
from_ac97_data = 8'b10111110;
ready = 1;
#20;
ready = 0;


#1980;
from_ac97_data = 8'b1110001;
ready = 1;
#20;
ready = 0;


#1980;
from_ac97_data = 8'b11101101;
ready = 1;
#20;
ready = 0;


#1980;
from_ac97_data = 8'b1110110;
ready = 1;
#20;
ready = 0;


#1980;
from_ac97_data = 8'b1101010;
ready = 1;
#20;
ready = 0;


#1980;
from_ac97_data = 8'b11010111;
ready = 1;
#20;
ready = 0;


#1980;
from_ac97_data = 8'b10000101;
ready = 1;
#20;
ready = 0;


#1980;
from_ac97_data = 8'b110011;
ready = 1;
#20;
ready = 0;


#1980;
from_ac97_data = 8'b10101011;
```

```
            ready = 1;
            #20;
            ready = 0;


            #1980;
            from_ac97_data = 8'b11010101;
            ready = 1;
            #20;
            ready = 0;


            #1980;
            from_ac97_data = 8'b101;
            ready = 1;
            #20;
            ready = 0;


            #1980;
            from_ac97_data = 8'b10101101;
            ready = 1;
            #20;
            ready = 0;


            #1980;
            from_ac97_data = 8'b1100000;
            ready = 1;
            #20;
            ready = 0;


            #1980;
            from_ac97_data = 8'b11010100;
            ready = 1;
            #20;
            ready = 0;

      end

endmodule
```