Audio Conference Communication System
(ACCS)
Final Report

Team Members:
Xianzhen Zhu (King)
Saher Ahwal
Wegene Tadele

1. Abstract (All)

As digital system engineers, we plan to design and implement an audio digital communication system for group conferencing. Whether you are an employee in a big company or a small one, individuals and groups in your building need to meet and communicate to make plans and work on milestones. We propose a design of a wired conference communication system which enables short meetings and calls between individuals and teams in the company with a single communication channel. A certain number of bases are to be installed in different regions, and each connected to a keypad, a headset and an FPGA. The system has a central coordinator that directs and manages communication among node stations. Each base transmits the audio/video signals to the coordinator. Then the coordinator routes/forwards the call to the corresponding node station(s) depending on the identification address(s) sent by the caller.

2. Introduction and Overview (All)

Digital communication has been advancing rapidly in the recent years, giving rise to a great deal of communication devices (e.g smartphones) and communication software (e.g Skype). Whether we are talking about cell phones or video communication software, there is dependency on the cellphone network and the internet connection respectively. For workplaces that rely heavily on audio and video communication between teams, there is a need for a more robust communication system. Therefore, we are planning to design and implement a digital communication system that allows people within a certain workplace to initiate audio conference without relying on unpredictable network failures and limited bandwidth of cellphone networks.

The system has a coordinator base which act as a central phone station. A certain number of bases will be installed in different regions and are going to be connected to the coordinator base. The system allow node stations (bases) to call to more than one node stations concurrently. If no one was available at a called base or the base was busy in a call, the caller will have an option to record a voicemail. A caller at the base will be able to listen to his voicemail and delete chosen messages from memory as well. A participant in a group call has the option to add a new person to the same call. If one tries to call a busy base (in active call), the busy tone should be heard by the caller.

The project was divided into multiple phases. In the first phase, the main features of voice communication were implemented like: having an audio conference conversation, voicemail and simple display of ongoing calls. Our next goal was to implement extra features like: incorporating multiple ringtones, busy tones, and using phone pads to initiate calls and listen to voicemail as well as video communication scheme. Unfortunately, we were not able to reach the second stage. We had to make important design decisions in the first stage. For simplicity and proof of concept, we decided to implement the design with three stations and a coordinator (total of four FPGAs used). We were able to integrate the coordinator display and the station displays with the overall conference system. We have implemented voicemail recorder that can store up to four voicemails on a ZBT RAM Thus, the voicemail is stored in volatile memory, however, we are assuming the stations are on all the time and the important part of the project is the actual audio conference, not the fact that you can leave messages to other stations.

3. Implementation

The following is a detailed description of the implementation of the Audio-Video Conference System (AVCS). First, we will present the block diagram of the system including the modules. Next, we will go through the detailed description of the modules.

3.1. Block Diagram

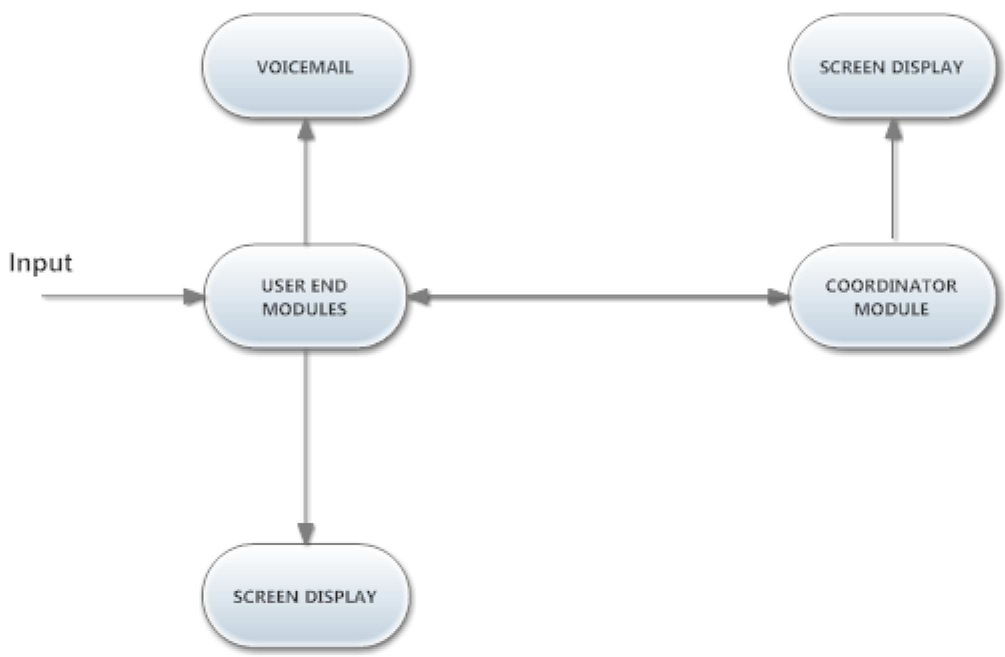

Figure 1-Higher Level Block Diagram

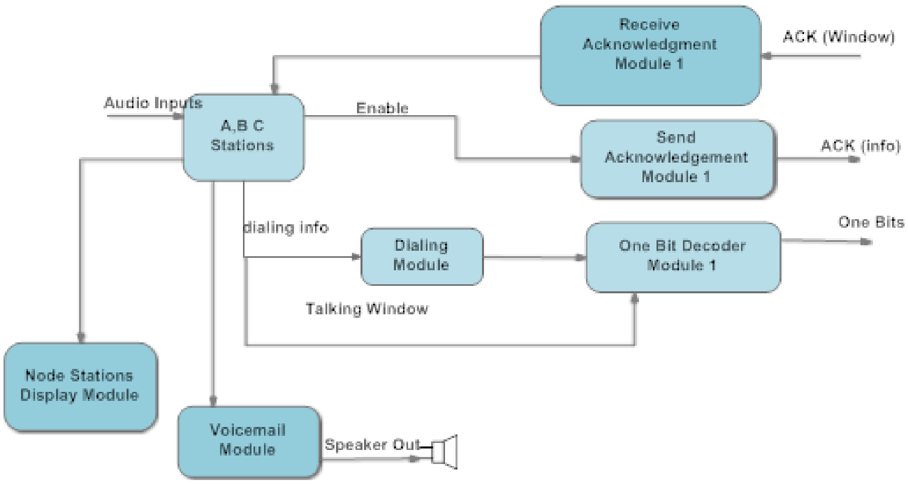

Figure-2: User end System Diagram

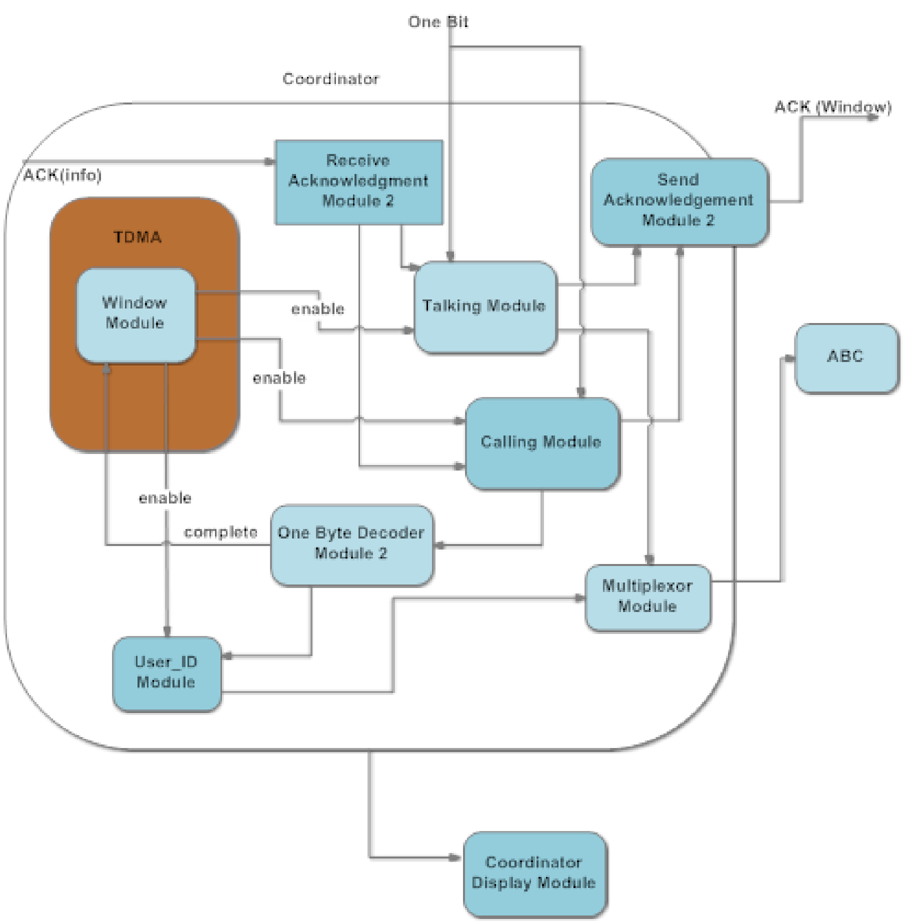

Figure-3: Coordinator System Diagram

3.2. Module Descriptions

3.2.1. User End Module:

3.2.1.1. Dialing Module: (*Wegene*)

This module is implemented on all stations and is responsible for decoding the dial/call. It takes the input possibly from lab kit switch or external keypad as calling a specific user end and transmits the information to One Bit Decoder Module which is to be transmitted to the coordinator side. We use the internal register from the FPGA to keep track of the node identification numbers. The register is called connections with a width of 9 like [a b c d e f g h i]. i represents the condition if station A is calling station B. h is the bit that indicates whether station B is calling station A. When both h and i are high (set by User- ID module) , the connection of A to B is set up, so A and B can talk to each other. Similar functionality applies for bits d, e, f , g. The remaining bits a, b, c indicate if somebody is leaving station ,A , B ,C a voice mail. When a, b, c is high , a command package is sent to user end stations to inform a new voice mail audio package is coming.

3.2.1.2 Voicemail Module *(Saher)*

This module is responsible for recording and listening to voicemail. The inputs determine whether the user wants to listen to voicemail, or delete/clear his voicemail. Furthermore, this module is responsible for saving the audio voicemail received after a certain timeout is passed.

We proposed to create voicemail module that utilizes the Flash ROM nonvolatile memory of the FPGA. However, we later decided to write voicemail messages to the ZBT RAM instead. Even though, this implies that messages will be lost when labkit is turned off, however, we have a simple approach of writing and reading from ZBT utilizing 32 bits of the 36 in each address location.

Even though we didn't integrate the voicemail with the overall system, the fact that we were able to integrate the displays and send and detect acknowledgement packets, tells us with confidence that the voicemail module can be integrated simply by adding a new acknowledgement packet in the coordinator that the user station can detect to start voicemail.

3.2.1.3 Node **Stations Display Module and Ringing Tones(Saher)**

This module is responsible for displaying a simple user interface on the screen at each station. This includes a display of incoming call information in addition to some simple graphics. The graphics were be saved on the FPGA within its' read-only memory (ROM). In idle state, the station displays an idle phone. When there is an incoming phone call from a station, we produce a ringing animation of phone images moving along the screen with different sizing indicating an incoming phone call. This module is connected to sound ringing modules to start a ringing tone stored in the ROM of the FPGA.

The plan was to create a different ringing tone for each caller. We have implemented one ringing tone as a proof of concept. The code for the node display module and the ringing tone implementation is shown at the end of this document.

3.2.1.4 Command packages send Module (King)

This module is responsible for sending different command information to user-ends. There are 7 different command packages. All of them are one byte.

These are the packets we send to inform user ends. 8’b01111110 is a package indicating it is the time for a specific user end to transmit calling information (who the station is calling). 8’b01011110 is a package indicating it is the time for a specific user end to transmit audio package to other connected stations. 8’b01101110 is a package indicating the next incoming audio info is from user end A. 8’b01110110 is a package indicating the next incoming audio info is from user end B. 8’b01111010 is a package indicating the next incoming audio info is from user end C.

It is necessary to indicate whether an audio package is from user A or B or C is because when there are two sound, we need use an algorithm to mix sound together.

This module takes input from window module, outputs corresponding command package to user-ends.

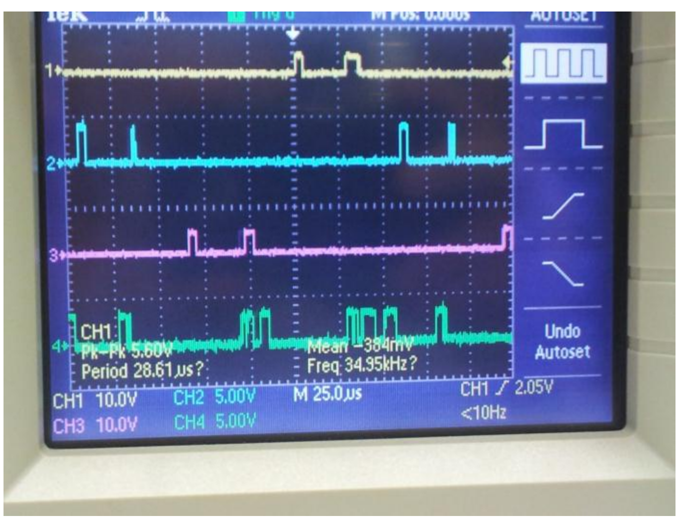

Figure-4

Figure-4 shows how the serial package works. The green signal informs the corresponding user end that it’s his time to talk and with audio data from other stations. The purple, the blue and yellow signals are signals received from user ends, sending informaiton of audio data and calling information to the coordinator.

3.2.1.5 Command **packages detection Module (King)**

This module is responsible for detecting whether there is command package. The module has several important features.

First, it has to oversample the inputs for 27 clock cycles (our system is operating at 1 MHz). The bit by bit decoding process is depending on how many 1's received for this 27 clock cycles.

Second, it need to detect edges to synchronize the incoming information to decode the correct data. The edge detection is important because different stations has different internal clocks, one can sample faster or slower than the others. The edge detection promises there is no accumulative errors due to different clock frequencies.

Third, it has a large state machine to detect different command.s from coordinator. When a command is detected, the module would turn a corresponding detection bit high. The input of the modules is serial data from coordinator. The output of the module is several package detection indicator bits.

3.2.1.6 Playing audio Module (King)

This module is responsible for playing received audio to headphone. When there are inputs from more than one user, the module would mix sounds up to enable multiple people communication at the same time. It takes inputs from stored audio information in registers and mix the sound by the algorithm. OUTPUT = A + B - A x B / 256.

After we calculate the output , we send the output to ac97 to headphone.

3.2.2. <u>Coordinator Station Modules:</u>

The following modules are implemented only on the coordinator station which is, in general, responsible for controlling the conferences by time division multiplexing communication protocol, the bookkeeping of ongoing calls on the FPGA memory, and the outgoing of voicemail signals.

3.2.2.1 Window Module (Wegene)

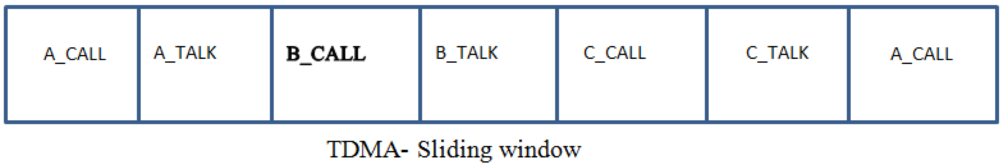

Figure-5: Sliding Window

This module is the core of the TDMA in the coordinator and the main communication block for the entire system. Two windows are allocated per stations to enable multiple communications at the same time including adding a new user to an ongoing audio conference or one to one calls. Figure-4 shows how a TDMA works. The TDMA has a counter that counts up to 656 clock cycles. If the TDMA doesn't receive a finished signal from the One Byte Decoder within the allocated clock cycle, the window moves to the next station and sends out an enable signal to the Calling and the Talking Modules depending on the current window. It successively and continuously gives enable signals to calling and talking modules so that one window is only for one calling or talking station. Since the clocks are extremely fast and the window allocated for each command (talking or calling) is really small, a transmitted packet can reach to its destination without collision. After going through all the windows, the module resets and starts over. In general, all the user end stations depend on the command signals from window module to start their own behaviors such as calling, talking or leaving a voicemail.

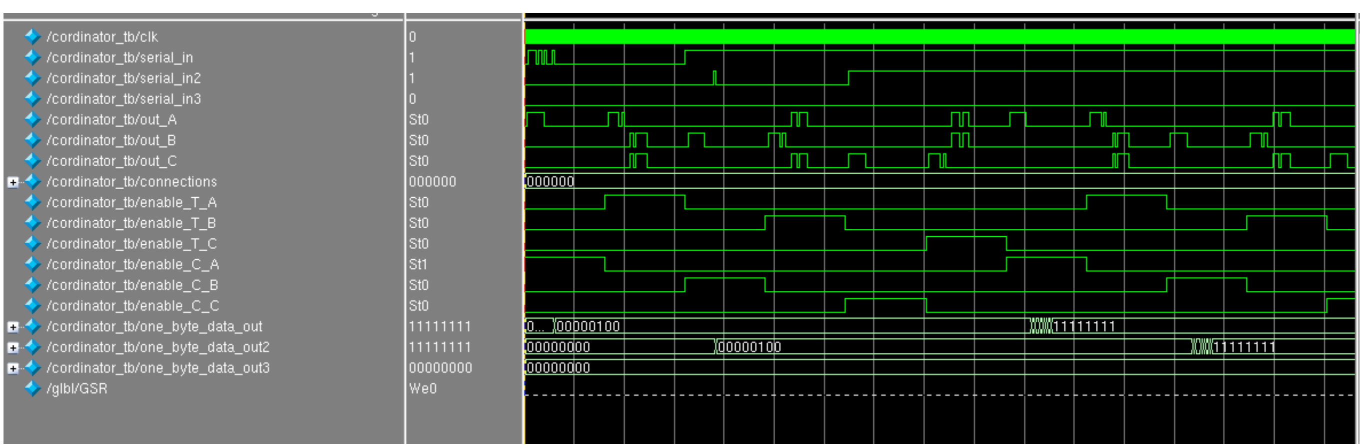

Figure-6

The diagram in Figure-6 shows how the TDMA slides the windows in action for talking and calling sessions. The column on the left side is a list of clk, incoming, enable, current connections and outgoing signals. There are three serial_ins corresponding to an incoming single bit data from a caller station, six enable signals two for each stations that enable the incoming serial ins to be transmitted when calling or create a space for data transmission when talking.

The three sets of signals shown under the serial_ins are the corresponding output single bit data from the corresponding serial_ins.

Below the one bit output signals is a six bits connections signal that shows the currently communicating sessions and can be used to inform the voicemail module whether a connection is created with a callee(s) or not. If a connection is not created, then the caller is given an option to leave a voice message by sending out a voicemail enable signal to both the caller and the callee. Below the connections are enable signals that are used to inform the Calling and Talking Modules that it is there time to transmit or receive a call. The enable signals are non-overlapping as shown in the diagram above since a single window is only used for one activity to avoid data collision and interference. After the enable signals are the three one byte data out signals that are each one byte long (8 bits long) and stored in register to be retrieved by the One Byte Data Decoder Module when transmitted.

3.2.2.2 Coordinator Display Module (Saher)

This module is similar to the User-End Display module. It is responsible for displaying information about the whole system status at the coordinator station. This shows a graphical interface of the stations and the ongoing calls. It shows occurring conferences and busy stations. The graphics/images will be saved on the FPGA's ROM on the coordinator station. In particular, when looking at these graphics on the coordinator monitor, one can tell which stations are connected.

3.2.2.3 User identification Module: (King)

This module promises a user end make correct connection to another specific user end. It includes data of each station's identification numbers. It essentially informs the address of the caller. Every user-end has corresponding one - byte information of connections. For example, if user A calls user B, this module would receive the information of 8'b00011100. If user A calls user B and user C at same time, the module would receive the information of 8'b00111100. If user A wants to leave a message to user B, the information would be 8'b00110010. By detecting the incoming information, the module would be able to establish a register of all the connections. The register of connections would be used in multiplexer module and package command send module.

3.2.2.4 One Byte Decoder Module(King)

This module enables the system to decode data from serial data transmission. The ideal situation is every eight clock cycles the module would sample a data from the last eight bits. The incoming data is serial data (0's and 1's). The information would be decoded in the form of one byte (a combination of 8 0's or 1's). The module takes the serial inputs data and converts the data to bytes, which could be used in speaker and recording memory. An important feature of this module is edge detecting. The module would correct its oversampling period by detecting edges to avoid decoding errors.

3.2.2.5 Multiplexer Module (Wegene)

This module allows the output to reach the desired end users. When one line is connected, audio data only transmits to the connected stations. The decoder module receives a single bit input from the serial data decoder module and outputs a byte (8 bit) data. The multiplexer uses the signal received from Talking or User Id Modules to choose the destination of a given call.

Thus, the multiplexor contains the outputs to all stations, an enable signals to the TDMA Window Module, the current transmitter user Id, three output bit registers as well as three serial acknowledgements registers. Up on receiving multiplexor enable signal from the Calling Module, serial connection information from the coordinator and output audio signals from the One Byte Decoder Module, it transmits the audio signal stored in the registers to the desired destination in a serial fashion. In addition to that, it sends out acknowledgements to the calling station

3.2.2.6 Calling Module (Saher)

This module takes the enable signal from window module and then sends an acknowledgement enable signal and then waits for an acknowledgement packet from user end to make sure it decodes the correct information. After having the acknowledgement back from user end, the module passes the correct information to one byte decoder module (then goes to User_ID module to setup connections).

3.2.2.7 Talking Module (Saher)

This module takes the enable signal from window module and then sends an acknowledgement enable signal and then waits for an acknowledgement packet from user end to make sure it decodes the correct information. After having the acknowledgement back from user end, the module passes the correct information to one byte decoder module. (then go to output multiplexer to go to user ends.)

4. Hardware installation and testing (Wegene)

The hardware section includes two wires from each station that are responsible for transmitting and receiving messages as well the maxim 485 transceiver chips. The maxim 485 chip drivers reduce reflections caused by improperly terminated cables, thus allowing error-free data transmission up to 250kbps. To minimize reflections, the lines are terminated at both ends at characteristic impedance of 120 Ω and 100 µF. In maxim 485, two wires (usually a twisted pair) carry the signal voltage and its inverse. The receiver detects the difference between the two. Because most noise that couples into the wires is common to both wires, it cancels out. The receiver detects the voltage difference between a signal voltage and a common ground. The ground wire tends to be noisy because it carries the return currents for all of the signals in the interface, along with whatever other noise has entered the wire from other sources. Thus, noise on the ground wire can cause the receiver to misread transmitted logic levels. *Figure 8* shows the wire layout of the transmitter and sender ends. Each station has two transceivers and the coordinator has six transceivers, two for each node stations.

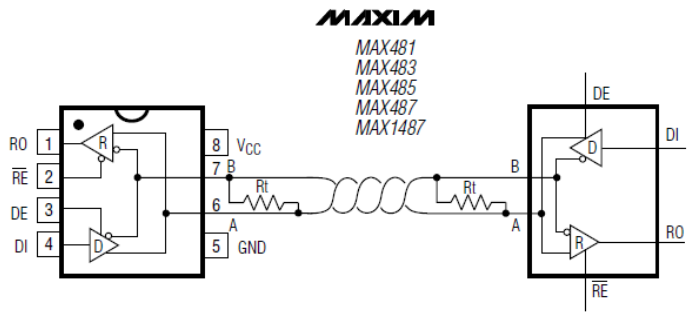

Figure-7: Maxim 485

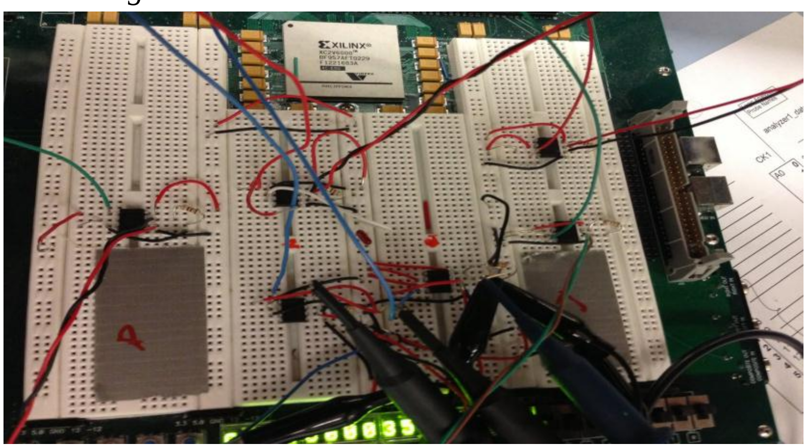

Figure-8- Left three are receivers and right three are transmitters

5. Testing

Testing was done per the following separate high level modules:

1. **The Voicemail Module (Saher)**

For Voicemail, testing was separated into two stages. First, we have tested the state machine that writes into the ZBT memory. We have tried to write recognizable patterns to different addresses and display them on the hex display of the FPGA. Since every address location stores 36 bits and our audio is 8-bit based audio, we had to use 32 out of 36 bits and disregard the 4 last bits. ModelSim was used to test the ability for the writing module to record 4 voicemail messages and start at the correct address after recording a message.

The Second stage of the Voicemail testing was to test the state machine which reads the bytes out one by one to playback the recorded message. This was done using ModelSim to make sure we are reading the correct bytes and incrementing the address after reading four bytes from each location.

The following ModelSim screenshot demonstrates the writing test. The enable signal is high when recording a message and when it is low, the register values of start_vm1/start_vm2/start_vm3 are updated accordingly to show the next starting address of the new voice message recorded. The data bus is also checked to make sure the correct audioIn 8-bit signal is recorded correctly every time a new ready pulse comes from the AC97 of the labkit. The ready signal is simulated by the newDataSignal here in the simulation.

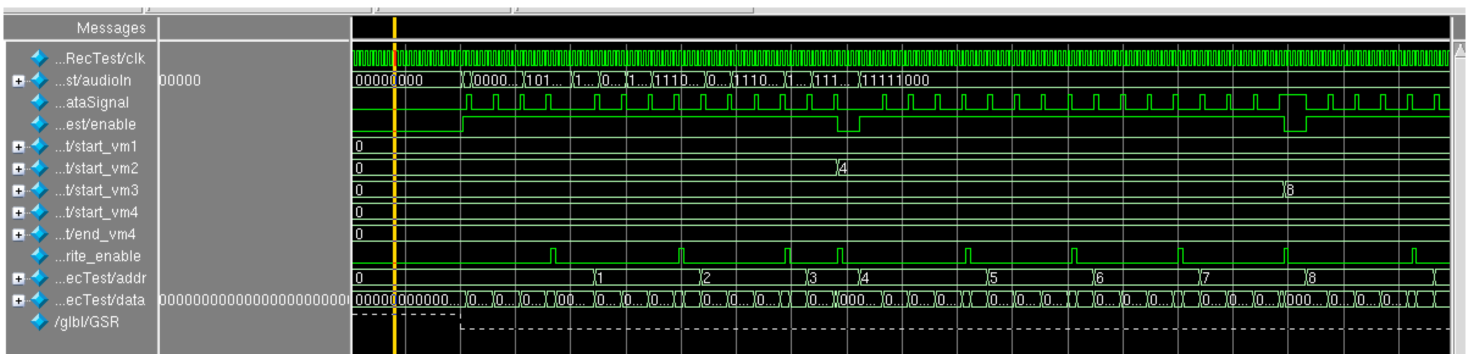

Figure-7

2. **The Display Modules (Saher)**

No simulation was needed to test the display modules. We simply used the switches to simulate different connections to be able to see the display changing accordingly to show the ongoing connections. For the node station display, we used an FPGA switch to simulate the ringing signal to test the moving display for ringing.

The ring display modules was implemented in a modular fashion to allow for integration to the overall conference system. Upon receiving a packet from the coordinator, the user-end station decoder module will be able to distinguish the packet which includes the current connections. Hence, the ringing signal can be created if no previous connection exists and so the ringing display can be activated.

The coordinator display was integrated with the system easily since it relies on the connections bus which defines the connections in the system. This bus exists within the coordinator and changes upon receiving connection packets from user-end stations.

3. The Coordinator Modules

The combined coordinator coordinator modules were tested using ModelSim.

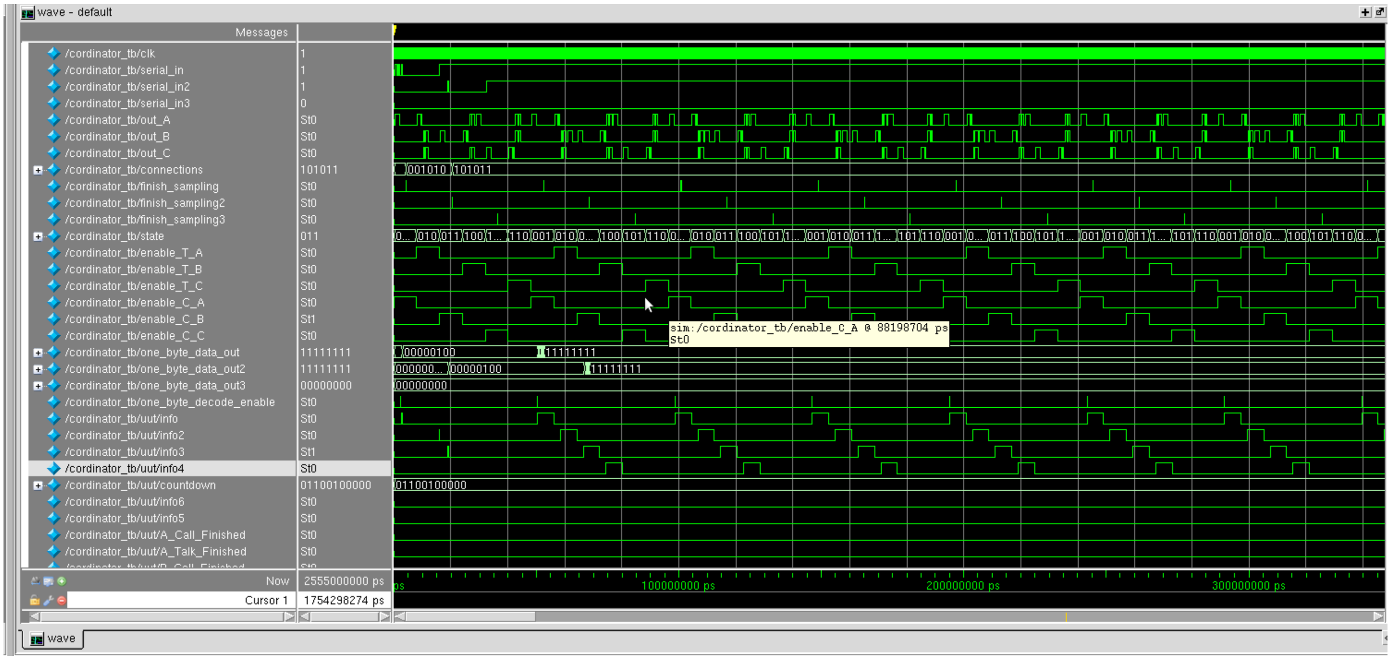

Figure-8

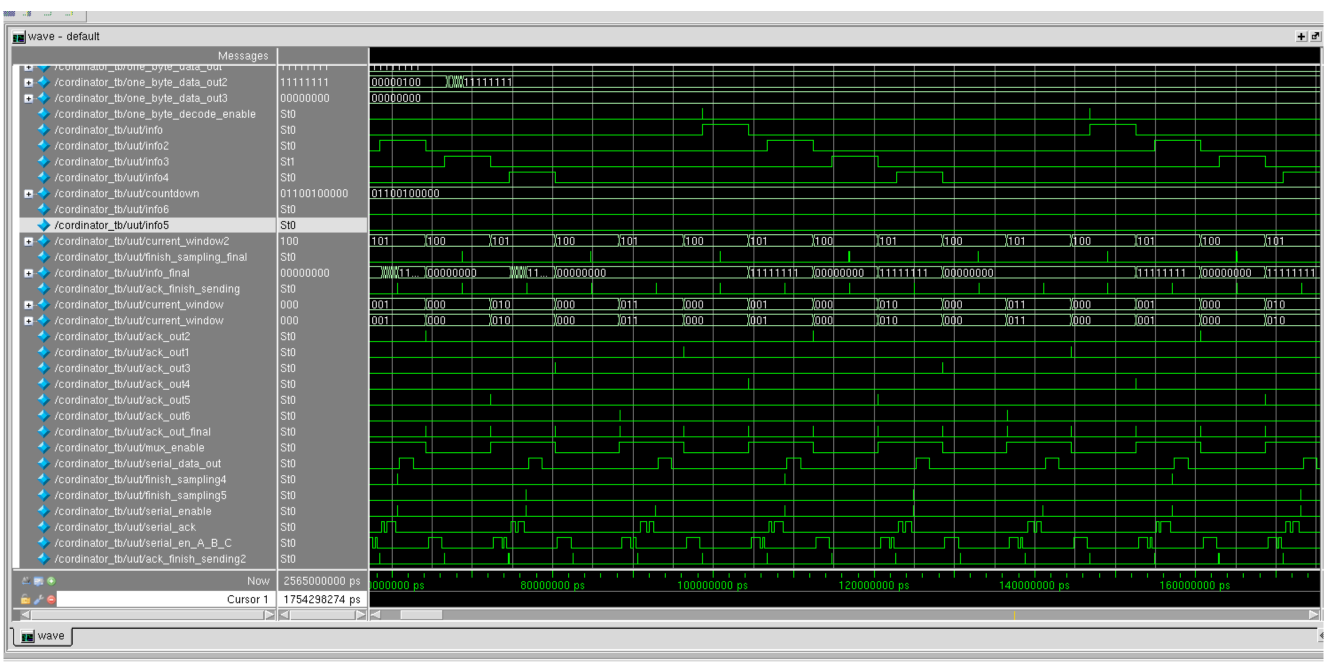

Figure-9

Figures 8 and 9 are the test benches for the coordinator. The test benches have all the necessary signals and register values

Conclusion (King) :

Overall, we followed the idea of proposal , using TDMA protocol for our audio conference system. During the process we implemented the system, we changed the acknowledgement packages to more complex command packets to direct user ends to perform different functions. Other than that , we used oversampling method to reduce data errors and edge detection to synchronize user end's time to the coordinator's time.

We also implemented ringtone and voicemail functionality, but we haven't had enough time to connect the modules to our system.

Future work includes exploring reducing noise in our system, using wireless technology to transmit information and enabling video conference.

The entire final project experience is exciting and we will definitely remember what we have accomplished in the project.

Verilog Codes
Below are the Verilog codes for our system.

1. One Bit Decoder Module

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:15:20:37 11/04/2012
// Design Name:
// Module Name:        one_bit_decoder
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module one_byte_decoder(
          input one_bit_data_in,
          input clk,
          input one_bit_data_enable,
          output reg [7:0]one_byte_data_out=0,
    output reg finish_sampling =0
          );
    reg [3:0] count =0;
    reg [5:0] loop = 0;
    reg [5:0] calculate=0;
    reg start = 0;
    reg final_bit=0;
    reg [5:0] acc=0;
//    reg tog= 0;
    reg tog2=0;
    always @(posedge clk)
    begin
           if (one_bit_data_enable)
```

```
begin
        start <=1;
        finish_sampling<=0;
end
if (start)
begin
        if (count<=7)
        begin
                if (count==7 && loop ==26)begin
                        tog2<=1;
                        finish_sampling<=1;end
                if (tog2)    begin
                        finish_sampling<=0;
                        tog2<=0;end
                begin
                if (loop <26)
                        begin
                        if (one_bit_data_in)begin
                                acc<=acc+1;end
                        loop<=loop+1;
                        end
                end

                if (loop == 26)
                begin
                        if (one_bit_data_in)begin
                        acc<=acc+1;end
                        calculate<=27;
                        begin
                        if (acc>18)
                                final_bit<=1;
                        else
                                final_bit<=0;
                        end
                        loop<=0;
                        acc<=0;
                end
                if (calculate==27)
                        begin
                        one_byte_data_out[count]<=final_bit;
                        count<=count+1;
                        calculate<=0;
                        end
        end //count end
```

```
        else
                begin
                    count<=0;
                    start<=0;
                    final_bit<=0;
                    loop<=0;
                    acc<=0;
                    calculate<=0;
                end
        end
  end
endmodule
```

2) One Bit Decoder Test Bench

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   15:28:14 11/04/2012
// Design Name:   one_bit_decoder
// Module Name:
/afs/athena.mit.edu/user/k/i/king000/6.111/sources/final_cool_project/one_bit_decoder_tb.v
// Project Name:  final_cool_project
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: one_bit_decoder
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module one_bit_decoder_tb;

  // Inputs
  reg one_bit_data_in;
  reg clk;
```

```
reg one_bit_data_enable;

// Outputs
wire [7:0] one_byte_data_out;
wire finish_sampling;

// Instantiate the Unit Under Test (UUT)
one_bit_decoder uut (
	.one_bit_data_in(one_bit_data_in),
	.clk(clk),
	.one_bit_data_enable(one_bit_data_enable),
	.one_byte_data_out(one_byte_data_out),
	.finish_sampling(finish_sampling)
);
always #5 clk = ! clk;
initial begin
	// Initialize Inputs
	one_bit_data_in = 0;
	clk = 0;
	one_bit_data_enable = 0;

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

	// Add stimulus here
	one_bit_data_enable = 1;
	#10;
	one_bit_data_enable = 0;
	one_bit_data_in =1;
	#10;
	one_bit_data_in =0;
	#10;
	one_bit_data_in =0;
	#10;
	one_bit_data_in =1;
	#10;
	one_bit_data_in =1;
	#10;
	one_bit_data_in =0;
	#10;
	one_bit_data_in =0;
	#10;
	one_bit_data_in =0;
	#50;
```

```
          one_bit_data_enable = 1;
   end


endmodule
```

3) Acknowledgement receive Module

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:15:57:32 11/11/2012
// Design Name:
// Module Name:       rec_ack
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module rcv_ack(
         input serial_ack_in,
         input clk,

//     input rec_ack_enable,
         output reg ack_talking =0,
     output reg ack_calling = 0,
     output reg ack_A_info = 0,
     output reg ack_B_info = 0,
     output reg ack_C_info = 0

         );

   reg last = 0;   //track the last bit
   reg current = 0;  //track the current bit
   reg syn = 0;           //start sync or now
```

```
   reg calculate = 1; //if syn happens, whether the clock is to fast or too slow, we choose
   //to calculate the last final bit or not

   reg toggg=0; //if continuous two signals be same, we know we should sync,
   //otherwise just a random error in the user end.

   reg enable_delay = 0;
   reg [5:0]ack_start=0;
   reg [5:0] acc=0;
   reg signal_in=0;
   reg [4:0]count=0;
   reg [1:0] tt=0;
   reg pp =0;
   reg [8:0] cooldown=0;   //cool down after a detection of who is talking (don't want to mess up
with the audio data)
   reg start =1;
parameter [5:0]state0= 6'd0;
parameter [5:0]state01= 6'd1;
parameter [5:0]state011= 6'd2;
parameter [5:0]state0111= 6'd3;//
parameter [5:0]state0110= 6'd4;//
parameter [5:0]state01110= 6'd5;//
parameter [5:0]state011101= 6'd6;
parameter [5:0]state0111011= 6'd7;
parameter [5:0]state0111010= 6'd8;
parameter [5:0]state01110101= 6'd9;
parameter [5:0]state01110111= 6'd10;
parameter [5:0]state01110110= 6'd11;
parameter [5:0]state01111= 6'd12;
parameter [5:0]state011111= 6'd13;
parameter [5:0]state011110= 6'd14;
parameter [5:0]state0111111= 6'd15;
parameter [5:0]state0111110= 6'd16;
parameter [5:0]state01111111= 6'd17;
parameter [5:0]state01111110= 6'd18;
parameter [5:0]state01111101= 6'd19;
parameter [5:0]state0111101= 6'd20;
parameter [5:0]state01111011= 6'd21;
parameter [5:0]state01111010= 6'd21;
parameter [5:0]state01101= 6'd22;
parameter [5:0]state011011= 6'd23;
parameter [5:0]state011010= 6'd24;
parameter [5:0]state0110111= 6'd25;
parameter [5:0]state0110110= 6'd26;
```

```
parameter [5:0]state01101111= 6'd27;
parameter [5:0]state01101110= 6'd28;
parameter [5:0]state01101101= 6'd29;
parameter [5:0]state0110101= 6'd30;
parameter [5:0]state01011101= 6'd31;
parameter [5:0]state010= 6'd32;
parameter [5:0]state0101= 6'd33;
parameter [5:0]state01010= 6'd34;
parameter [5:0]state01011= 6'd35;
parameter [5:0]state010101= 6'd36;
parameter [5:0]state010110= 6'd37;
parameter [5:0]state0101011= 6'd38;
parameter [5:0]state0101010= 6'd39;
parameter [5:0]state01010110= 6'd40;
parameter [5:0]state01010111= 6'd41;
parameter [5:0]state01010101= 6'd42;
parameter [5:0]state0101101= 6'd43;
parameter [5:0]state01011011= 6'd44;
parameter [5:0]state01011010= 6'd45;
parameter [5:0]state010111= 6'd46;
parameter [5:0]state0101110= 6'd47;
parameter [5:0]state0101111= 6'd48;
parameter [5:0]state01011110= 6'd49;
parameter [5:0]state01011111= 6'd50;
parameter [5:0]state01101011= 6'd51;
parameter [5:0]state01101010=6'd52;
reg [5:0] state =0;
   always @(posedge clk) begin


   last<=current;   ///////////////   synchronize clock edges
   current<=serial_ack_in;
   pp = (current!==last);
   begin
   if (current!==last)
   begin
         if (ack_start!=1)begin
         toggg<=1;
         tt<=tt+1;end
         if(tt==2) begin
                 tt<=0;
                 toggg<=0;
         end
   end
```

```
    end
```

4) Acknowledgement Receive test bench

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   15:48:56 12/07/2012
// Design Name:   ack_rcv
// Module Name:
/afs/athena.mit.edu/user/k/i/king000/6.111/sources/FINAL___PRO/fuck_ack_tb.v
// Project Name:  FINAL___PRO
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: fuck_ack
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module rcv_ack_tb;

    // Inputs
    reg serial_ack_in;
    reg clk;


    // Outputs
    wire ack_talking;
    wire ack_calling;
    wire ack_A_info;
    wire ack_B_info;
    wire ack_C_info;
```

```
// Instantiate the Unit Under Test (UUT)
rcv_ack uut (
	.serial_ack_in(serial_ack_in),
	.clk(clk),

	.ack_talking(ack_talking),
	.ack_calling(ack_calling),
	.ack_A_info(ack_A_info),
	.ack_B_info(ack_B_info),
	.ack_C_info(ack_C_info)
);

always #5 clk =!clk;
initial begin
	// Initialize Inputs
	serial_ack_in = 0;
	clk = 0;


	// Wait 100 ns for global reset to finish
	#105;

	serial_ack_in =1;
	#250;
	serial_ack_in =0;
	#270;
	serial_ack_in =1;
	#270;
	serial_ack_in =1;
	#270;
	serial_ack_in=1;
	#270;
	serial_ack_in=1;
	#270;
	serial_ack_in=0;
	#270;
	// Add stimulus here

	#3000;
	serial_ack_in =0;
	#250;
	serial_ack_in =1;
	#270;
	serial_ack_in =1;
```

```
        #270;
        serial_ack_in =1;
        #270;
        serial_ack_in=1;
        #270;
        serial_ack_in=0;
        #270;
        serial_ack_in=1;
        #270;
        serial_ack_in=0;

    end

endmodule
```

5) calling Module

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   15:48:56 12/07/2012
// Design Name:   ack_rcv
// Module Name:
/afs/athena.mit.edu/user/k/i/king000/6.111/sources/FINAL___PRO/fuck_ack_tb.v
// Project Name:  FINAL___PRO
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: fuck_ack
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module rcv_ack_tb;
```

```
// Inputs
reg serial_ack_in;
reg clk;


// Outputs
wire ack_talking;
wire ack_calling;
wire ack_A_info;
wire ack_B_info;
wire ack_C_info;

// Instantiate the Unit Under Test (UUT)
rcv_ack uut (
	.serial_ack_in(serial_ack_in),
	.clk(clk),

	.ack_talking(ack_talking),
	.ack_calling(ack_calling),
	.ack_A_info(ack_A_info),
	.ack_B_info(ack_B_info),
	.ack_C_info(ack_C_info)
);

always #5 clk =!clk;
initial begin
	// Initialize Inputs
	serial_ack_in = 0;
	clk = 0;


	// Wait 100 ns for global reset to finish
	#105;

	serial_ack_in =1;
	#250;
	serial_ack_in =0;
	#270;
	serial_ack_in =1;
	#270;
	serial_ack_in =1;
	#270;
	serial_ack_in=1;
	#270;
```

```
		serial_ack_in=1;
		#270;
		serial_ack_in=0;
		#270;
		// Add stimulus here

		#3000;
		serial_ack_in =0;
		#250;
		serial_ack_in =1;
		#270;
		serial_ack_in =1;
		#270;
		serial_ack_in =1;
		#270;
		serial_ack_in=1;
		#270;
		serial_ack_in=0;
		#270;
		serial_ack_in=1;
		#270;
		serial_ack_in=0;

	end

endmodule
```

6) Calling Module Test bench
7) Pulse

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   15:48:56 12/07/2012
// Design Name:   ack_rcv
// Module Name:
/afs/athena.mit.edu/user/k/i/king000/6.111/sources/FINAL___PRO/fuck_ack_tb.v
// Project Name:  FINAL___PRO
// Target Device:
// Tool versions:
// Description:
//
```

```
// Verilog Test Fixture created by ISE for module: fuck_ack
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module rcv_ack_tb;

   // Inputs
   reg serial_ack_in;
   reg clk;


   // Outputs
   wire ack_talking;
   wire ack_calling;
   wire ack_A_info;
   wire ack_B_info;
   wire ack_C_info;

   // Instantiate the Unit Under Test (UUT)
   fuck_ack uut (
         .serial_ack_in(serial_ack_in),
         .clk(clk),

         .ack_talking(ack_talking),
         .ack_calling(ack_calling),
         .ack_A_info(ack_A_info),
         .ack_B_info(ack_B_info),
         .ack_C_info(ack_C_info)
   );

   always #5 clk =!clk;
   initial begin
         // Initialize Inputs
         serial_ack_in = 0;
         clk = 0;


         // Wait 100 ns for global reset to finish
```

```
		#105;

		serial_ack_in =1;
		#250;
		serial_ack_in =0;
		#270;
		serial_ack_in =1;
		#270;
		serial_ack_in =1;
		#270;
		serial_ack_in=1;
		#270;
		serial_ack_in=1;
		#270;
		serial_ack_in=0;
		#270;
		// Add stimulus here

		#3000;
		serial_ack_in =0;
		#250;
		serial_ack_in =1;
		#270;
		serial_ack_in =1;
		#270;
		serial_ack_in =1;
		#270;
		serial_ack_in=1;
		#270;
		serial_ack_in=0;
		#270;
		serial_ack_in=1;
		#270;
		serial_ack_in=0;

	end

endmodule
```

8) Dialing Module

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
```

```
//
// Create Date:    16:37:23 11/04/2012
// Design Name:
// Module Name:    dialing_module_A
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////
module dialing_module_A(
   input clk,
   input call_B,
   input call_C,
   output reg [7:0]User_ID_from_A
   );
        wire [1:0]condition;
        assign condition[0] = call_C;
        assign condition[1] = call_B;
        always @(posedge clk)
        begin
        case (condition)
                2'b00:
                        User_ID_from_A = 8'd0;     //hang off
                2'b01:
                        User_ID_from_A = 8'b00000010; //B
                2'b10:
                  User_ID_from_A = 8'b00000011; //C
                2'b11:
                        User_ID_from_A = 8'b00000100; //B &C
        endcase
end


endmodule
```

9) Dialing Module test bench

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

```
////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   16:50:47 11/04/2012
// Design Name:   dialing_module_A
// Module Name:
/afs/athena.mit.edu/user/k/i/king000/6.111/sources/final_cool_project/dialing_module_tb.v
// Project Name:  final_cool_project
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: dialing_module_A
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module dialing_module_tb;

	// Inputs
	reg clk;
	reg call_B;
	reg call_C;

	// Outputs
	wire [7:0] User_ID_from_A;

	// Instantiate the Unit Under Test (UUT)
	dialing_module_A uut (
		.clk(clk),
		.call_B(call_B),
		.call_C(call_C),
		.User_ID_from_A(User_ID_from_A)
	);
  always #5 clk = ! clk;
	initial begin
		// Initialize Inputs
```

```
		clk = 0;
		call_B = 0;
		call_C = 0;

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

		// Add stimulus here
		call_B = 1;
		#10;
		call_B = 0;
		#10;
		call_C =1;
		call_B = 1;
		#50;
		call_B =0;

	end

endmodule
```

10) Divider Module

```
module Divider #(parameter [24:0] COUNT_TO = 25'd27)//25'd26999999)
					(input clk, start_timer,
	    output one_hz_enable);

	reg [24:0] count = 25'd0;

	always @(posedge clk) begin
		count <= count + 1;
	  if (count == COUNT_TO || start_timer == 1'b1) begin
			count <= 0;
		end
	end

	assign one_hz_enable = (count == COUNT_TO);
```

11) multiplexer Module

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

```
////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    16:12:03 11/04/2012
// Design Name:
// Module Name:    output_multiplexer
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////
module output_multiplexer(
    input [5:0]connections,                                      //indication of connections
    input serial_audio_input,    //output from talking module (from user end)
    output  reg out_A ,                                          //output to station A
    output  reg out_B ,
    output  reg out_C ,
            input enable_C_A,
            input enable_C_B,
            input enable_C_C,
            input enable_T_A,
            input enable_T_B,
            input enable_T_C,
    input clk,
            input mux_enable,
            input serial_ack,        //ack pack of who is talking
            input serial_en_A_B_C,   //ack pack of calling or talking
    input [1:0]current_transmit_user //indicator of current transmitting station
    );                              //A 00, B 01, C 10;
            reg start = 0;
            reg info_to_A =0;
            reg info_to_B = 0;
            reg info_to_C = 0;
            reg ack_to_A = 0;
            reg ack_to_B = 0;
            reg ack_to_C = 0;
```

```
		always @(posedge clk)
		begin

		case (current_transmit_user)
		2'b00 :
			begin
//				if((connections[0] ==1 )&&(connections[1] ==1 )) //A and B connected
//					out_B <=serial_audio_input;
				info_to_B <=
(mux_enable)&&(connections[0]==1)&&(connections[1]==1)&&serial_audio_input;
				info_to_C <=
(mux_enable)&&(connections[2]==1)&&(connections[3]==1)&&serial_audio_input;
				info_to_A<=0;
			end
		2'b01 :
			begin
				info_to_A <=
(mux_enable)&&(connections[0]==1)&&(connections[1]==1)&&serial_audio_input;
				info_to_C <=(mux_enable)&&
(connections[4]==1)&&(connections[5]==1)&&serial_audio_input;
				info_to_B<=0;
			end
		2'b01 :
			begin

			info_to_A <=
(mux_enable)&&(connections[3]==1)&&(connections[2]==1)&&serial_audio_input;
			info_to_B <=
(mux_enable)&&(connections[4]==1)&&(connections[5]==1)&&serial_audio_input;
			info_to_C<=0;
			end
		endcase
		if (enable_C_A)
		begin
			ack_to_A <=serial_en_A_B_C;
			out_A <=ack_to_A;
			out_B <=0;
			out_C <=0;
			ack_to_B <=0;
			ack_to_C <=0;
		end
			if (enable_C_B)
		begin
			ack_to_B <=serial_en_A_B_C;
```

```
        out_B <=ack_to_B;

        out_A <=0;
        out_C <=0;
        ack_to_A <=0;
        ack_to_C <=0;
end

if (enable_C_C)
begin
        ack_to_C <=serial_en_A_B_C;
        out_C<=ack_to_C;

        out_A <=0;
        out_B <=0;
        ack_to_A <=0;
        ack_to_B <=0;
end
if (enable_T_A)
begin
        ack_to_A <=serial_en_A_B_C;
        out_A<=ack_to_A;
        ack_to_B <=serial_ack;
        out_B<=ack_to_B+info_to_B;
        ack_to_C <=serial_ack;
        out_C<=ack_to_C+info_to_C;
end
if (enable_T_B)
begin
        ack_to_B <=serial_en_A_B_C;
        out_B<=ack_to_B;
        ack_to_A <=serial_ack;
        out_A<=ack_to_A+info_to_A;
        ack_to_C <=serial_ack;
        out_C<=ack_to_C+info_to_C;
end
if (enable_T_C)
begin
        ack_to_C <=serial_en_A_B_C;
        out_C<=ack_to_C;
        ack_to_A <=serial_ack;
        out_A<=ack_to_A+info_to_A;
        ack_to_B <=serial_ack;
        out_B<=ack_to_B+info_to_B;
```

```
	end
	end

endmodule
```

12) Multiplexor Test Bench

13) Receiving user ID

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Company: 
// Engineer: 
// 
// Create Date:    16:00:58 11/18/2012 
// Design Name: 
// Module Name:    receiving_path 
// Project Name: 
// Target Devices: 
// Tool versions: 
// Description: 
//
// Dependencies: 
//
// Revision: 
// Revision 0.01 - File Created
// Additional Comments: 
//
////////////////////////////////////////////////////////////////////////////////
module receiving_user_id(
	input clk,
	input enable,
	input serial_in,
	input rec_ack_enable,
	input [1:0]current_window_calling,
	output ack_out,
	output one_byte_decode_enable,
	output ack_detect,
	output finish_sampling2,
	output [7:0]one_byte_data_out,

	 output [5:0]connections );
	 wire info2;
//	 wire ack_detect;

rec_ack rcvack2(
		.serial_ack_in(serial_in),
```

```
                .clk(clk),
                .rec_ack_enable(rec_ack_enable),
                .ack_detect(ack_detect)
        );

CallingOrTalkingModule callingModule2(
                .enable(enable),
                .serial_in(serial_in),
                .clk(clk),
                .ack_in(ack_detect),
                .ack_out(ack_out),
                .to_one_byte_decode(info2),
                .one_byte_decode_enable(one_byte_decode_enable)
        );

        one_byte_decoder fuckmylife (
                .one_bit_data_in(serial_in),
                .clk(clk),
                .one_bit_data_enable(one_byte_decode_enable),
                .one_byte_data_out(one_byte_data_out),
                .finish_sampling(finish_sampling2)
        );
User_ID fuckmeagain (
                .User_ID_in(one_byte_data_out),
                .User_ID_enable(finish_sampling2),
                .clk(clk),
                .connections(connections),
                .current_window_calling(current_window_calling)
        );

endmodule
```

16) Serial Data Testbench

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   14:57:08 11/04/2012
// Design Name:   serial_data
```

```
// Module Name:
/afs/athena.mit.edu/user/k/i/king000/6.111/sources/final_cool_project/serial_data_tb.v
// Project Name:  final_cool_project
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: serial_data
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module serial_data_tb;

    // Inputs
    reg clk;
    reg [7:0] serial_data_in;
    reg serial_enable;

    // Outputs
    wire serial_data_out;


    // Instantiate the Unit Under Test (UUT)
    serial_data uut (
        .clk(clk),
        .serial_data_in(serial_data_in),
        .serial_enable(serial_enable),
        .serial_data_out(serial_data_out)
    );
    always #5 clk = ! clk;
    initial begin
        // Initialize Inputs
        clk = 0;
        serial_data_in = 8'b01101110;
        serial_enable = 0;

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

```
		// Add stimulus here
		serial_enable = 1;
		#10;
		serial_enable =0;
	end

endmodule
```

17) Station Sending

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:21:26:46 11/25/2012
// Design Name:
// Module Name:    station_sending
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////
module station_sending(
		input clk,
		input finish_ack_send,
		output dialing_enable,
	output talking_enable
		);
	reg status=0; //calling or talking
	reg start =0;
	reg [1:0] acc=0;
	reg [10:0]count<=0

	always @ (posedge clk) begin
		if (~start)begin
			if (finish_ack_send)begin
```

```
                acc<=acc+1;end
            if (acc==1)
                if (count<=1500)
                begin
                    if (acc==2)
                        start<=1;
                    else
                        count<=count+1;
                end
                else
                    start<=1

        if (start)
            if (~status && finish_ack_send)

endmodule
```

18) Timer Module

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:21:26:46 11/25/2012
// Design Name:
// Module Name:        station_sending
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////
module station_sending(
        input clk,
        input finish_ack_send,
        output dialing_enable,
    output talking_enable
```

```
        );
    reg status=0; //calling or talking
    reg start =0;
    reg [1:0] acc=0;
    reg [10:0]count<=0

   always @ (posedge clk) begin
        if (~start)begin
            if (finish_ack_send)begin
                acc<=acc+1;end
            if (acc==1)
                if (count<=1500)
                begin
                    if (acc==2)
                        start<=1;
                    else
                        count<=count+1;
                end
                else
                    start<=1

        if (start)
            if (~status && finish_ack_send)

endmodule
```

19) User End Top Level Module

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:21:26:46 11/25/2012
// Design Name:
// Module Name:     station_sending
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
```

```
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////
module station_sending(
	input clk,
	input finish_ack_send,
	output dialing_enable,
    output talking_enable
	);
    reg status=0; //calling or talking
    reg start =0;
    reg [1:0] acc=0;
    reg [10:0]count<=0

   always @ (posedge clk) begin
		if (~start)begin
			if (finish_ack_send)begin
				acc<=acc+1;end
			if (acc==1)
				if (count<=1500)
				begin
					if (acc==2)
						start<=1;
					else
						count<=count+1;
				end
				else
					start<=1

		if (start)
			if (~status && finish_ack_send)

endmodule
```

20) UserEnd top Level Module TestBench

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   17:26:28 12/08/2012
// Design Name:   top_A
// Module Name:   /afs/athena.mit.edu/user/k/i/king000/6.111/sources/FINAL___PRO/top_A_tb.v
```

```
// Project Name:  FINAL___PRO
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: top_A
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module top_A_tb;

   // Inputs
   reg clk;
   reg serial_ack_in;
   reg ready;
   reg call_B;
   reg call_C;
   reg [7:0] codec_audio;

   // Outputs
   wire serial_data_out;
   wire audio_to_headphone;

   // Instantiate the Unit Under Test (UUT)
   top_A uut (
         .clk(clk),
         .serial_ack_in(serial_ack_in),
         .ready(ready),
         .call_B(call_B),
         .call_C(call_C),
         .codec_audio(codec_audio),
         .serial_data_out(serial_data_out),
         .audio_to_headphone(audio_to_headphone)
   );

   always #5 clk =! clk;
   initial begin
         // Initialize Inputs
```

```
        clk = 0;
        serial_ack_in = 0;
        ready = 0;
        call_B = 0;
        call_C = 0;
        codec_audio = 0;

        // Wait 100 ns for global reset to finish
        #105;

        // Add stimulus here
        call_B=1;
        call_C=1;
        serial_ack_in=0;
        #100;
        serial_ack_in=1;
        #270;
        serial_ack_in=1;
        #270;
        serial_ack_in=1;
        #270;
        serial_ack_in=1;
        #270;
        serial_ack_in=1;
        #270;
        serial_ack_in=1;
        #200;
        serial_ack_in=0;

    end

endmodule
```

20) User End Address Module

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:02:25:39 12/08/2012
// Design Name:
// Module Name:     user_end_A
// Project Name:
// Target Devices:
```

```
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module user_end_A(
    input clk,
//    input reset,
    input ack_talking,
    input ack_calling,
    input ack_B_info,
    input ack_C_info,
    input [7:0]audio_in_byte,
    input ready, //(audio codec signal in lab kit)
    input [7:0]codec_audio,//from audio chip
    input call_B,
    input call_C,
    input finish_sampling,
    output reg [7:0] audio_to_headphone=0,
    output reg one_byte_decoder_enable=0,
    output reg serial_data_enable=0,
    output reg [7:0]to_serial_data_byte=0
        );

    reg [7:0] audio_from_codec=0;
    reg [7:0] final_to_headphone=0;
    reg [7:0] audio_from_B=0;
    reg [7:0] audio_from_C=0;
    reg [7:0] calling_reg=0;
    reg [7:0] temp_b;
    reg [7:0] temp_c;

    reg start = 0;
    reg tog = 0;
    reg tog2=0;
    reg [7:0]mul=0;
    //mixing sound algorithm x= A+B-A*B/256

    always @(posedge clk) begin
```

```
if ((ack_B_info)||(ack_C_info))
        one_byte_decoder_enable<=1;
if ((~ack_B_info)&&(~ack_C_info))
        one_byte_decoder_enable<=0;

if (ready)begin
        audio_from_codec<=codec_audio;
        start<=1;
        end



// call_reg setup
if ((call_B)&&(call_C))
        calling_reg<=8'b00000100;
if ((~call_B)&&(call_C))
calling_reg<=8'b00000011;
if ((call_B)&&(~call_C))
calling_reg<=8'b00000010;
if ((~call_B)&&(~call_C))
calling_reg<=8'b00000000;

temp_b<=1+audio_from_B>>4;
temp_c<=1+audio_from_C>>4;
mul<=temp_b*temp_c-1;
final_to_headphone<=audio_from_B+audio_from_C-mul;  //need physical test

//play sound
if (start) begin
        audio_to_headphone<=final_to_headphone;
        start<=0;
        end

//match input to right place
if (ack_B_info)
        tog<=1;

if (ack_C_info)
        tog2<=1;

if ((tog)&&(finish_sampling)) begin
        audio_from_B<=audio_in_byte;
        tog<=0;
```

```
        end

    if ((tog2)&&(finish_sampling)) begin
        audio_from_C<=audio_in_byte;
        tog2<=0;
        end

    if (ack_talking) begin
        to_serial_data_byte<=codec_audio;
        serial_data_enable<=1;
        end
    if (ack_calling) begin
        to_serial_data_byte<=calling_reg;
        serial_data_enable<=1;
        end

    if ((~ack_talking)&&(~ack_calling))
        serial_data_enable<=0;
    end
endmodule
```

21) UserEnd Address Module

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   03:25:45 12/08/2012
// Design Name:   user_end_A
// Module Name:
/afs/athena.mit.edu/user/k/i/king000/6.111/sources/FINAL___PRO/user_end_A_tb.v
// Project Name:  FINAL___PRO
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: user_end_A
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
```

```
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module user_end_A_tb;

	// Inputs
	reg clk;
	reg ack_talking;
	reg ack_calling;
	reg ack_B_info;
	reg ack_C_info;
	reg [7:0] audio_in_byte;
	reg ready;
	reg [7:0] codec_audio;
	reg call_B;
	reg call_C;
	reg finish_sampling;

	// Outputs
	wire [7:0] audio_to_headphone;
	wire one_byte_decoder_enable;
	wire serial_data_enable;
	wire [7:0] to_serial_data_byte;

	// Instantiate the Unit Under Test (UUT)
	user_end_A uut (
		.clk(clk),
		.ack_talking(ack_talking),
		.ack_calling(ack_calling),
		.ack_B_info(ack_B_info),
		.ack_C_info(ack_C_info),
		.audio_in_byte(audio_in_byte),
		.ready(ready),
		.codec_audio(codec_audio),
		.call_B(call_B),
		.call_C(call_C),
		.finish_sampling(finish_sampling),
		.audio_to_headphone(audio_to_headphone),
		.one_byte_decoder_enable(one_byte_decoder_enable),
		.serial_data_enable(serial_data_enable),
		.to_serial_data_byte(to_serial_data_byte)
	);
```

```
always #5 clk = !clk;
initial begin
	// Initialize Inputs
	clk = 0;
	ack_talking = 0;
	ack_calling = 0;
	ack_B_info = 0;
	ack_C_info = 0;
	audio_in_byte = 0;
	ready = 0;
	codec_audio = 0;
	call_B = 0;
	call_C = 0;
	finish_sampling = 0;

	// Wait 100 ns for global reset to finish
			#105;

	 finish_sampling=1;
	 call_B=1;
	 call_C=1;
	 ready=1;
	 codec_audio=8'b00001111;
	 audio_in_byte=8'b11111111;
	 #10;
	 ack_calling=1;
	 #10;
	 ack_calling=0;
	 #200;
	 ack_talking=1;
	 #10;
	 ack_talking=0;
	 #200;
	 ack_B_info=1;
	 #10;
	 ack_B_info=0;
	 #200;
	 ack_C_info=1;
	 #10;
	 ack_C_info=0;

	// Add stimulus here

end
```

```
endmodule
```

22) User End Detect Module

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:21:12:12 12/05/2012
// Design Name:
// Module Name:    user_end_start_detect
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module user_end_start_detect(
        input clk,
        input serial_in,
        output reg ack_enable=0
        );
    reg [1:0] tog = 2'b0;
    reg [5:0] count=0;
    always @(posedge clk) begin

    if (~tog)begin
         if (count<27) begin
                 if (serial_in==1)
                         count<=count+1;
                 if (serial_in==0)
                         count<=0;
                 end
         if (count==25) begin
                 tog<=1;
                 ack_enable<=1;
```

```
                end
            end
    end
endmodule
```

23) UserEnd Detect TestBench

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   22:32:53 12/05/2012
// Design Name:   user_end_start_detect
// Module Name:
/afs/athena.mit.edu/user/k/i/king000/6.111/sources/FINAL___PRO/user_end_start_detect_tb.v
// Project Name:  FINAL___PRO
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: user_end_start_detect
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module user_end_start_detect_tb;

    // Inputs
    reg clk;
    reg serial_in;

    // Outputs
    wire ack_enable;

    // Instantiate the Unit Under Test (UUT)
    user_end_start_detect uut (
        .clk(clk),
```

```
        .serial_in(serial_in),
        .ack_enable(ack_enable)
  );
  always #5 clk=!clk;
  initial begin
        // Initialize Inputs
        clk = 0;
        serial_in = 0;

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

        serial_in =1 ;
        #250;
        serial_in = 0;
        #20;
        serial_in = 1;
        #270;
        // Add stimulus here

  end

endmodule
```

24) User Id Module

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:15:35:38 11/04/2012
// Design Name:
// Module Name:       User_ID
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
```

```
//////////////////////////////////////////////////////////////////////////////////
module User_ID(
        input [7:0]User_ID_in,          //A has the ID 00000001, B has the ID 00000010
    //should come from one_byte_decoder
    //C has the ID 00000011
        input User_ID_enable,                   //since we have different time window (calling or
talking)
        input clk,
        output reg [5:0]connections=0,                  //has the form of [000000] MSB is B to C
    //then C to B , then A to C then C to A then A to B to B to A
        input [1:0]current_window_calling     //user A = 0, user B =1, user C=2
        );

    always @(posedge clk)    //maybe use negedge to make it faster
    begin
    if (User_ID_enable)
    begin
         case (current_window_calling)
         2'b00:
                  case(User_ID_in)
                  8'b0000010 : connections[1] <=1;
                  8'b0000011 : connections[3] <=1;
                  8'b0000000 : //case hang off/ resets the register

                  begin
                           connections[1]<=0;
                           connections[3]<=0;
                  end
                  8'b0000100 : //connect both
                  begin
                           connections[1]<=1;
                           connections[3]<=1;
                  end
                  endcase
         2'b01:
                  case(User_ID_in)
                  8'b0000001 : connections[0] <=1;
                  8'b0000011 : connections[5] <=1;
                  8'b0000000 :
                  begin
                           connections[0]<=0;
                           connections[5]<=0;
                  end
                  8'b0000100 :
```

```
                begin
                        connections[0]<=1;
                        connections[5]<=1;
                end
                endcase
        2'b10:
                case(User_ID_in)
                8'b0000001 : connections[2] <=1;
                8'b0000010 : connections[4] <=1;

                8'b0000000 :
                begin
                        connections[2]<=0;
                        connections[4]<=0;
                end
                8'b0000100 :
                begin
                        connections[2]<=1;
                        connections[4]<=1;
                end
        endcase
        endcase
    end
   end

endmodule
```

25) User Id Module testbench

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   15:48:53 11/04/2012
// Design Name:   User_ID
// Module Name:
/afs/athena.mit.edu/user/k/i/king000/6.111/sources/final_cool_project/User_ID_tb.v
// Project Name:  final_cool_project
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: User_ID
```

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

module User_ID_tb;

	// Inputs
	reg [7:0]User_ID_in=0;
	reg User_ID_enable;
	reg clk;
	reg [1:0] current_window_calling;

	// Outputs
	wire [5:0] connections;

	// Instantiate the Unit Under Test (UUT)
	User_ID uut (
		.User_ID_in(User_ID_in),
		.User_ID_enable(User_ID_enable),
		.clk(clk),
		.connections(connections),
		.current_window_calling(current_window_calling)
	);
	always #5 clk = ! clk;
	initial begin
		// Initialize Inputs
		User_ID_in = 0;
		User_ID_enable = 0;
		clk = 0;
		current_window_calling = 0;

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

		// Add stimulus here
		User_ID_enable = 1;
		User_ID_in = 8'b0000011; //c
		current_window_calling = 2'b00;
		#20;
```

```
        User_ID_enable = 0;
        #20;
        User_ID_enable = 1;
        User_ID_in = 8'b0000010; //b
        current_window_calling = 2'b00;

        #20;
        User_ID_enable = 0;
        #20;
        User_ID_enable = 1;
        User_ID_in = 8'b0000011; //b
        current_window_calling = 2'b01;
        #20;
        User_ID_enable = 0;
    end

endmodule
```

26) TDMA Window Module

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:15:40:37 11/11/2012
// Design Name:
// Module Name:      Window_Module
// Project Name:
// Target Devices:
// Tool versions:
// Description:
// This Module checks if each node station has finished talking or calling. If it finishes calling or
talking or
// the countdown signal is 0, it turns off the corresponding signal on the output side and enables
the next
// window to be high. Which is basically the talking window for the previously calling window or
the next
// calling window for the following station. It also enables the the 'enable_User_Id' signal which
goes to the
// User_ID_Module if the stations are calling.
// Dependencies:
//
// Revision:
```

```
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module Window_Module(
    input clk,
         input A_Call_Finished,
         input A_Talk_Finished,
         input B_Call_Finished,
         input B_Talk_Finished,
         input C_Call_Finished,
         input C_Talk_Finished,
    input expired,
         output reg [1:0]current_calling_window,
    output reg [2:0]state,
    output reg start_timer,
    output reg enable_C_A =0,
    output reg enable_C_B =0,
    output reg enable_C_C =0,
    output reg enable_T_A =0,
    output reg enable_T_B =0,
    output reg enable_T_C =0,
    output reg [10:0] countdown=10'd800,
    output reg [2:0] current_window, // for sending correct ack
    output reg [2:0] current_window2

         );


   // State definitions
   parameter [2:0] IDLE = 3'd0;
   parameter [2:0] A_CALL = 3'd1;
   parameter [2:0] A_TALK = 3'd2;
   parameter [2:0] B_CALL = 3'd3;
   parameter [2:0] B_TALK = 3'd4;
   parameter [2:0] C_CALL = 3'd5;
   parameter [2:0] C_TALK = 3'd6;
initial begin
  state=IDLE;
end
always @(posedge clk) begin

   case(state)
          IDLE:
```

```
begin
        start_timer <=1;
        state <= A_CALL;
        enable_C_A = 1;
        current_calling_window    <=00;
        current_window=0;
        current_window2=4; // for A_calling

end

A_CALL:
begin
        start_timer <= 0;
        if (A_Call_Finished || expired) begin
                state <= A_TALK;
                start_timer<=1;
                current_calling_window    <=00;
                enable_C_A = 0;
                enable_T_A = 1;
                current_window=1;
                current_window2=5; //for A talking

        end
end

A_TALK:
begin
        start_timer <= 0;
        if (A_Talk_Finished||expired) begin
                state<=B_CALL;
                start_timer<=1;
                current_calling_window<=2'b01;
                enable_T_A = 0;
                enable_C_B = 1;
                current_window=0;
                current_window2=4; //for B calling

        end
end
B_CALL:
begin
        start_timer <= 0;
        if (B_Call_Finished||expired) begin
                state<=B_TALK;
```

```
                start_timer<=1;
                current_calling_window<=2'b01;
                enable_C_B = 0;
                enable_T_B = 1;
                current_window=2;
                current_window2=5; //for B talking

        end
end
B_TALK:
begin
        start_timer <= 0;
        if (B_Talk_Finished||expired) begin
                state<=C_CALL;
                start_timer<=1;
                current_calling_window<=2'b10;
                enable_T_B = 0;
                enable_C_C = 1;
                current_window=0;
                current_window2=4;

                end
end
C_CALL:
begin
        start_timer <= 0;
        if (C_Call_Finished||expired) begin
                state<=C_TALK;
                start_timer<=1;
                current_calling_window<=2'b10;
                enable_C_C = 0;
                enable_T_C = 1;
                current_window=3;
                current_window2 = 5;

        end
end
C_TALK:
begin
        start_timer <= 0;
        if (C_Talk_Finished||expired) begin
                start_timer<=1;
                state<=A_CALL;
                current_calling_window<=2'b00;
```

```
                                enable_T_C = 0;
                                enable_C_A = 1;
                                current_window=0;
                                current_window2=4;

                      end
            end
   endcase
end
endmodule
```

27) TDMA window Module TestBench

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   12:21:09 11/12/2012
// Design Name:   Window_Module
// Module Name:
/afs/athena.mit.edu/user/w/e/wegene/Desktop/6111FinalProject/FinalProject/Window_Module_T
est.v
// Project Name:  FinalProject
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: Window_Module
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module Window_Module_Test;

   // Inputs
   reg clk;
   reg A_Call_Finished;
   reg A_Talk_Finished;
```

```
   reg B_Call_Finished;
   reg B_Talk_Finished;
   reg C_Call_Finished;
   reg C_Talk_Finished;
   wire expired;
   reg value;

   // Outputs
   wire [1:0]current_calling_window;
   wire [2:0]state;
   wire start_timer;
   wire [4:0] countdown;
   wire [4:0]tVal;
   wire enable_T_A;
   wire enable_C_A;
   wire enable_T_B;
   wire enable_C_B;
   wire enable_T_C;
   wire enable_C_C;

   // Instantiate the Unit Under Test (UUT)
Window_Module uut (
		.clk(clk),
		.A_Call_Finished(A_Call_Finished),
		.A_Talk_Finished(A_Talk_Finished),
		.B_Call_Finished(B_Call_Finished),
		.B_Talk_Finished(B_Talk_Finished),
		.C_Call_Finished(C_Call_Finished),
		.C_Talk_Finished(C_Talk_Finished),
		.expired(expired),
		.current_calling_window(current_calling_window),
		.state(state),
		.start_timer(start_timer),
		.countdown(countdown),
		.enable_T_A(enable_T_A),
		.enable_T_B(enable_T_B),
		.enable_T_C(enable_T_C),
		.enable_C_A(enable_C_A),
		.enable_C_B(enable_C_B),
		.enable_C_C(enable_C_C)
   );
Divider dv1 (.clk(clk),.start_timer(start_timer),.one_hz_enable(one_hz_enable));
Timer timer1(
		.clk(clk),
```

```
        .value(countdown),
        .start_timer(start_timer),
        .expired(expired),
        .tVal(tVal));



    always #5 clk=!clk;
    initial begin
        // Initialize Inputs
        clk = 0;
        A_Call_Finished = 0;
        A_Talk_Finished = 0;
        B_Call_Finished = 0;
        B_Talk_Finished = 0;
        C_Call_Finished = 0;
        C_Talk_Finished = 0;


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

        // Add stimulus here

    end

endmodule
```

28) Coordinator Module

```
//`default_nettype none
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:20:29:05 11/18/2012
// Design Name:
// Module Name:     cordinator
// Project Name:
```

```
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module cordinator( input clk,
                                   input serial_in,
                                   input serial_in2,
                                   input serial_in3,

    output   out_A ,                             //output to station A
        output   out_B ,
        output   out_C,
    output [5:0]connections,

    output finish_sampling,
    output finish_sampling2,
    output finish_sampling3,
    output [2:0]state,
    output enable_T_A,
    output enable_T_B,
    output enable_T_C,
    output enable_C_A,
    output enable_C_B,
    output enable_C_C,
    output [7:0]one_byte_data_out,
    output [7:0]one_byte_data_out2,
    output [7:0]one_byte_data_out3,

    output one_byte_decode_enable


        );


//wire one_byte_decode_enable;
wire [1:0] current_window_calling;
wire [7:0] one_byte_data_out4;
```

```
wire [7:0] one_byte_data_out5;
wire [7:0] one_byte_data_out6;

wire one_byte_decode_enable2;
wire one_byte_decode_enable3;
wire one_byte_decode_enable4;
wire one_byte_decode_enable5;
wire one_byte_decode_enable6;
wire [7:0]one_byte_data_out_final =
 (current_window_calling==00) ?one_byte_data_out :
 (current_window_calling==01)?one_byte_data_out2:
 (current_window_calling==10)?one_byte_data_out3:0;

wire info;
wire info2;
wire info3;
wire info4;
wire info5;
wire info6;
wire [10:0]countdown;
wire [10:0]tVal;
wire [10:0]value;
wire expired;
wire start_timer;
wire one_hz_enable;
wire A_Call_Finished = 0;
wire B_Call_Finished = 0;
wire C_Call_Finished = 0;

wire A_Talk_Finished =0;
wire B_Talk_Finished =0;
wire C_Talk_Finished =0;
wire [2:0]current_window2;
//wire one_bit_enable_final;
//assign one_bit_enable_final =
one_byte_decode_enable+one_byte_decode_enable3+one_byte_decode_enable5;
wire finish_sampling_final = finish_sampling + finish_sampling2 + finish_sampling3;
wire [7:0]info_final = (enable_T_A)? one_byte_data_out4 :
(enable_T_B) ? one_byte_data_out5 : (enable_T_C) ? one_byte_data_out6 : 0;
//wire [1:0]current_window_calling;
wire ack_out;
wire ack_finish_sending;
wire [2:0]current_window;
wire ack_out1;
```

```
wire ack_out2;
wire ack_out3;
wire ack_out4;
wire ack_out5;
wire ack_out6;
wire ack_out_final = ack_out1+ack_out2+ack_out3+ack_out4+ack_out5+ack_out6;
wire mux_enable = (enable_T_A + enable_T_B + enable_T_C);
wire serial_data_out;
wire finish_sampling4;
wire finish_sampling5;
wire finish_sampling6;
wire serial_enable = (finish_sampling4+finish_sampling5+finish_sampling6)
&&(enable_T_A+enable_T_B+enable_T_C);

wire serial_ack;
wire serial_en_A_B_C;
////////////////////////////////////////////////////////////////// rec_acl_modules


CallingOrTalkingModule A1(
        .enable(enable_C_A),
        .serial_in(serial_in),
        .clk(clk),
        .ack_in(ack_finish_sending2),
        .ack_out(ack_out1),
        .to_one_byte_decode(info),
        .one_byte_decode_enable(one_byte_decode_enable)
  );

CallingOrTalkingModule B1(
        .enable(enable_C_B),
        .serial_in(serial_in2),
        .clk(clk),
        .ack_in(ack_finish_sending2),
        .ack_out(ack_out2),
        .to_one_byte_decode(info3),
        .one_byte_decode_enable(one_byte_decode_enable3)
  );

CallingOrTalkingModule C1(
        .enable(enable_C_C),
        .serial_in(serial_in3),
        .clk(clk),
        .ack_in(ack_finish_sending2),
```

```
            .ack_out(ack_out3),
            .to_one_byte_decode(info5),
            .one_byte_decode_enable(one_byte_decode_enable5)
    );
////////////////////////////////////////////////////////////////////// calling

CallingOrTalkingModule A2(
            .enable(enable_T_A),
            .serial_in(serial_in),
            .clk(clk),
            .ack_in(ack_finish_sending2),
            .ack_out(ack_out4),
            .to_one_byte_decode(info2),
            .one_byte_decode_enable(one_byte_decode_enable2)
    );
CallingOrTalkingModule B2(
            .enable(enable_T_B),
            .serial_in(serial_in2),
            .clk(clk),
            .ack_in(ack_finish_sending2),
            .ack_out(ack_out5),
            .to_one_byte_decode(info4),
            .one_byte_decode_enable(one_byte_decode_enable4)
    );
CallingOrTalkingModule C2(
            .enable(enable_T_C),
            .serial_in(serial_in3),
            .clk(clk),
            .ack_in(ack_finish_sending2),
            .ack_out(ack_out6),
            .to_one_byte_decode(info6),
            .one_byte_decode_enable(one_byte_decode_enable6)
    );


/////////////////////////////////////////////////talking

one_byte_decoder fuckmylife (
            .one_bit_data_in(serial_in),
            .clk(clk),
            .one_bit_data_enable(one_byte_decode_enable),
            .one_byte_data_out(one_byte_data_out),
            .finish_sampling(finish_sampling)
    );
```

```
one_byte_decoder fuckmylife2 (
            .one_bit_data_in(serial_in2),
            .clk(clk),
            .one_bit_data_enable(one_byte_decode_enable3),
            .one_byte_data_out(one_byte_data_out2),
            .finish_sampling(finish_sampling2)
    );

one_byte_decoder fuckmylife3 (
            .one_bit_data_in(serial_in3),
            .clk(clk),
            .one_bit_data_enable(one_byte_decode_enable5),
            .one_byte_data_out(one_byte_data_out3),
            .finish_sampling(finish_sampling3)
    );

    /// for talking window
one_byte_decoder fuckmylife4 (
            .one_bit_data_in(serial_in),
            .clk(clk),
            .one_bit_data_enable(one_byte_decode_enable2),
            .one_byte_data_out(one_byte_data_out4),
            .finish_sampling(finish_sampling4)
    );

one_byte_decoder fuckmylife5 (
            .one_bit_data_in(serial_in2),
            .clk(clk),
            .one_bit_data_enable(one_byte_decode_enable4),
            .one_byte_data_out(one_byte_data_out5),
            .finish_sampling(finish_sampling5)
    );

one_byte_decoder fuckmylife6 (
            .one_bit_data_in(serial_in3),
            .clk(clk),
            .one_bit_data_enable(one_byte_decode_enable6),
            .one_byte_data_out(one_byte_data_out6),
            .finish_sampling(finish_sampling6)
    );

///////////////////////////////////////////one_bit_decoder
```

```
User_ID fuckmeagain3 (
            .User_ID_in(one_byte_data_out_final),
            .User_ID_enable(finish_sampling_final),
            .clk(clk),
            .connections(connections),
            .current_window_calling(current_window_calling)
    );


///////////////////////////////////////user_id

    output_multiplexer outputmultiplexer4 (
            .connections(connections),
            .serial_audio_input(serial_data_out), //change
            .out_A(out_A),
            .out_B(out_B),
            .out_C(out_C),
            .mux_enable(mux_enable),
            .clk(clk),
            .current_transmit_user(current_window_calling),
            .serial_ack(serial_ack),
            .serial_en_A_B_C(serial_en_A_B_C),
            .enable_T_A(enable_T_A),
            .enable_T_B(enable_T_B),
            .enable_T_C(enable_T_C),
            .enable_C_A(enable_C_A),
            .enable_C_B(enable_C_B),
            .enable_C_C(enable_C_C)


    );

            serial_data owa (
            .clk(clk),
            .serial_data_in(info_final),
            .serial_enable(serial_enable),
            .serial_data_out(serial_data_out)
    );
/////////////////////////////////mux

Window_Module uut (
            .clk(clk),
            .A_Call_Finished(A_Call_Finished),
            .A_Talk_Finished(A_Talk_Finished),
```

```
            .B_Call_Finished(B_Call_Finished),
            .B_Talk_Finished(B_Talk_Finished),
            .C_Call_Finished(C_Call_Finished),
            .C_Talk_Finished(C_Talk_Finished),
            .expired(expired),
            .current_calling_window(current_window_calling),
            .state(state),
            .start_timer(start_timer),
            .countdown(countdown),
            .enable_T_A(enable_T_A),
            .enable_T_B(enable_T_B),
            .enable_T_C(enable_T_C),
            .enable_C_A(enable_C_A),
            .enable_C_B(enable_C_B),
            .enable_C_C(enable_C_C),
            .current_window(current_window),
            .current_window2(current_window2)

    );

Timer timer1(
            .clk(clk),
            .value(countdown),
            .start_timer(start_timer),
            .expired(expired),
            .tVal(tVal));
//////////////////////////ack_send

    ack_send ack_fcuk (
            .ack_send_enable(ack_out_final),
            .clk(clk),
            .current_window(current_window2),
            .ack_send_finish(ack_finish_sending2),
            .serial_ack(serial_en_A_B_C) //transmitting to idle stations who is talking
    );
            ack_send ack_to_listeners (
            .ack_send_enable(ack_finish_sending2),//enable after send ack to talking window user
            .clk(clk),
            .current_window(current_window),
            .ack_send_finish(ack_finish_sending),
            .serial_ack(serial_ack)
    );
endmodule
```

29) Coordinator Module TestBench

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company: 
// Engineer:
//
// Create Date:   16:52:52 12/02/2012
// Design Name:   cordinator
// Module Name:   
/afs/athena.mit.edu/user/k/i/king000/6.111/sources/FINAL___PRO/cordinator_tb.v
// Project Name:  FINAL___PRO
// Target Device:  
// Tool versions:  
// Description: 
//
// Verilog Test Fixture created by ISE for module: cordinator
//
// Dependencies:
// 
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
// 
////////////////////////////////////////////////////////////////////////////////

module cordinator_tb;

	// Inputs
	reg clk;
	reg serial_in;
	reg serial_in2;
	reg serial_in3;

	// Outputs
	wire out_A;
	wire out_B;
	wire out_C;
	wire [5:0] connections;
	wire finish_sampling;
	wire finish_sampling2;
	wire finish_sampling3;
	wire [2:0] state;
	wire enable_T_A;
```

```
wire enable_T_B;
wire enable_T_C;
wire enable_C_A;
wire enable_C_B;
wire enable_C_C;
wire [7:0] one_byte_data_out;
wire [7:0] one_byte_data_out2;
wire [7:0] one_byte_data_out3;
wire one_byte_decode_enable;


// Instantiate the Unit Under Test (UUT)
cordinator uut (
	.clk(clk),
	.serial_in(serial_in),
	.serial_in2(serial_in2),
	.serial_in3(serial_in3),
	.out_A(out_A),
	.out_B(out_B),
	.out_C(out_C),
	.connections(connections),
	.finish_sampling(finish_sampling),
	.finish_sampling2(finish_sampling2),
	.finish_sampling3(finish_sampling3),
	.state(state),
	.enable_T_A(enable_T_A),
	.enable_T_B(enable_T_B),
	.enable_T_C(enable_T_C),
	.enable_C_A(enable_C_A),
	.enable_C_B(enable_C_B),
	.enable_C_C(enable_C_C),
	.one_byte_data_out(one_byte_data_out),
	.one_byte_data_out2(one_byte_data_out2),
	.one_byte_data_out3(one_byte_data_out3),
	.one_byte_decode_enable(one_byte_decode_enable)

);
always #5 clk = ! clk;
initial begin
	// Initialize Inputs
	clk = 0;
	serial_in = 0;
	serial_in2 = 0;
	serial_in3 = 0;
```

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

#270;
serial_in = 1;
#270;
serial_in = 1;
#270;
serial_in = 1;
#270;
serial_in = 0;
#270;
serial_in = 1;
#270;
serial_in = 0;
#270;
serial_in = 1;
#270;
serial_in = 0;
#270;
serial_in = 0;
#270;
serial_in = 1; // a to b,c
#270;
serial_in = 0;
#8000;

#3000;
#1990;
serial_in = 1;
serial_in2 = 0;
#1800
#270;
serial_in2 = 0;
#270;
#270;
#270;
serial_in2 = 1;
#270;
serial_in2 =0;
#270;
serial_in2 = 0;
#13000;
```

```
        serial_in2 = 1;
        //talking in for

        // Add stimulus here

  end

endmodule
```

30) Node Stations Complete Code

```
////////////////////////////////////////////////////////////////////////////////
//
// Pushbutton Debounce Module (video version - 24 bits)
//
////////////////////////////////////////////////////////////////////////////////

module debounce (input reset, clock, noisy,
                 output reg clean);

  reg [19:0] count;
  reg new;

  always @(posedge clock)
       if (reset) begin new <= noisy; clean <= noisy; count <= 0; end
       else if (noisy != new) begin new <= noisy; count <= 0; end
       else if (count == 650000) clean <= new;
       else count <= count+1;

endmodule

module lab5audio (
 input wire clock_27mhz,
 input wire reset,
 input wire [4:0] volume,
 output wire [7:0] audio_in_data,
 input wire [7:0] audio_out_data,
 output wire ready,
 output reg audio_reset_b,   // ac97 interface signals
 output wire ac97_sdata_out,
 input wire ac97_sdata_in,
 output wire ac97_synch,
 input wire ac97_bit_clock
);
```

```
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;

// wait a little before enabling the AC97 codec
reg [9:0] reset_count;
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(.ready(ac97_ready),
	.command_address(command_address),
	.command_data(command_data),
	.command_valid(command_valid),
	.left_data(left_out_data), .left_valid(1'b1),
	.right_data(right_out_data), .right_valid(1'b1),
	.left_in_data(left_in_data), .right_in_data(right_in_data),
	.ac97_sdata_out(ac97_sdata_out),
	.ac97_sdata_in(ac97_sdata_in),
	.ac97_synch(ac97_synch),
	.ac97_bit_clock(ac97_bit_clock));

// ready: one cycle pulse synchronous with clock_27mhz
reg [2:0] ready_sync;
always @ (posedge clock_27mhz) ready_sync <= {ready_sync[1:0], ac97_ready};
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(clock_27mhz), .ready(ready),
                     .command_address(command_address),
                     .command_data(command_data),
                     .command_valid(command_valid),
                     .volume(volume),
                     .source(3'b000));       // mic
endmodule

// assemble/disassemble AC97 serial frames
module ac97 (
  output reg ready,
  input wire [7:0] command_address,
  input wire [15:0] command_data,
  input wire command_valid,
  input wire [19:0] left_data,
  input wire left_valid,
  input wire [19:0] right_data,
  input wire right_valid,
  output reg [19:0] left_in_data, right_in_data,
  output reg ac97_sdata_out,
  input wire ac97_sdata_in,
  output reg ac97_synch,
  input wire ac97_bit_clock
);
  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;

  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 (
  input wire clock,
  input wire ready,
  output wire [7:0] command_address,
  output wire [15:0] command_data,
  output reg command_valid,
  input wire [4:0] volume,
  input wire [2:0] source
);
  reg [23:0] command;

  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 (
  input wire clock,
  input wire ready,
  output reg [19:0] pcm_data
);
   reg [8:0] index;

   initial begin
         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)
//
// 2012-Sep-15: Converted to 24bit RGB
//
// 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 lab3   (beep, audio_reset_b, ac97_sdata_out, ac97_sdata_in, ac97_synch,
         ac97_bit_clock,

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

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

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

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

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

          clock_feedback_out, clock_feedback_in,

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

          rs232_txd, rs232_rxd, rs232_rts, rs232_cts,

          mouse_clock, mouse_data, keyboard_clock, keyboard_data,

          clock_27mhz, clock1, clock2,

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

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

          switch,

          led,

          user1, user2, user3, user4,

          daughtercard,

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

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

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

   output [7:0] vga_out_red, vga_out_green, vga_out_blue;
```

```
output vga_out_sync_b, vga_out_blank_b, vga_out_pixel_clock,
       vga_out_hsync, vga_out_vsync;

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

input  [19:0] tv_in_ycrcb;
input  tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2, tv_in_aef,
       tv_in_hff, tv_in_aff;
output tv_in_i2c_clock, tv_in_fifo_read, tv_in_fifo_clock, tv_in_iso,
       tv_in_reset_b, tv_in_clock;
inout  tv_in_i2c_data;

inout  [35:0] ram0_data;
output [18:0] ram0_address;
output ram0_adv_ld, ram0_clk, ram0_cen_b, ram0_ce_b, ram0_oe_b, ram0_we_b;
output [3:0] ram0_bwe_b;

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

input  clock_feedback_in;
output clock_feedback_out;

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

output rs232_txd, rs232_rts;
input  rs232_rxd, rs232_cts;

input  mouse_clock, mouse_data, keyboard_clock, keyboard_data;

input  clock_27mhz, clock1, clock2;

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

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

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

inout [43:0] daughtercard;

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

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

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

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

// 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

        //ram0 used

//assign ram0_data = 36'hZ;
//assign ram0_address = 19'h0;
        //assign ram0_clk = 1'b0;
//assign ram0_cen_b = 1'b1;
        //assign ram0_we_b = 1'b1;

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

        //ram 1 here

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;
//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;

////////////////////////////////////////////////////////////////////////////
//
// lab3 : a simple pong game
//
////////////////////////////////////////////////////////////////////////////

// use FPGA's digital clock manager to produce a
// 65MHz clock (actually 64.8MHz)
wire clock_65mhz_unbuf,clock_65mhz;
DCM vclk1(.CLKIN(clock_27mhz),.CLKFX(clock_65mhz_unbuf));
// synthesis attribute CLKFX_DIVIDE of vclk1 is 10
// synthesis attribute CLKFX_MULTIPLY of vclk1 is 24
// synthesis attribute CLK_FEEDBACK of vclk1 is NONE
// synthesis attribute CLKIN_PERIOD of vclk1 is 37
BUFG vclk2(.O(clock_65mhz),.I(clock_65mhz_unbuf));

// power-on reset generation
wire power_on_reset;        // remain high for first 16 clocks
SRL16 reset_sr (.D(1'b0), .CLK(clock_65mhz), .Q(power_on_reset),
        .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
defparam reset_sr.INIT = 16'hFFFF;

// ENTER button is user reset
wire global_reset,user_reset;
debounce
db1(.reset(power_on_reset),.clock(clock_65mhz),.noisy(~button3),.clean(user_reset));
assign global_reset = user_reset | power_on_reset;

// UP and DOWN buttons for pong paddle
wire up,down;
debounce db2(.reset(global_reset),.clock(clock_65mhz),.noisy(~button_up),.clean(up));
debounce db3(.reset(global_reset),.clock(clock_65mhz),.noisy(~button_down),.clean(down));
```

```
   // generate basic XVGA video signals
   wire [10:0] hcount;
   wire [9:0]  vcount;
   wire hsync,vsync,blank;
   xvga xvga1(.vclock(clock_65mhz),.hcount(hcount),.vcount(vcount),
          .hsync(hsync),.vsync(vsync),.blank(blank));

   // feed XVGA signals to user's pong game
   wire [23:0] pixel;
   wire phsync,pvsync,pblank;
          wire [7:0] from_ac97_data, to_ac97_data;
   wire ready;
//   pong_game pg(.vclock(clock_65mhz),.reset(reset),
//                    .up(up),.down(down),.pspeed(switch[7:4]),
//          .hcount(hcount),.vcount(vcount),
//                    .hsync(hsync),.vsync(vsync),.blank(blank),
//          .phsync(phsync),.pvsync(pvsync),.pblank(pblank),.pixel(pixel));

//          wire [23:0] pixel_A, pixel_B, pixel_C;
//
//          DisplayCoordinatorStationA staA(.clk(clock_65mhz),.hcount(hcount),.vcount(vcount),
//          .x(11'd200), .y(10'd100), .pixel(pixel_A), .hsync(hsync),
//          .vsync(vsync),.blank(blank),.phsync(phsync),.pvsync(pvsync),.pblank(pblank));
//
//          DisplayCoordinatorStationB staB(.clk(clock_65mhz),.hcount(hcount),.vcount(vcount),
//          .x(11'd700), .y(10'd100), .pixel(pixel_B), .hsync(hsync),
//          .vsync(vsync),.blank(blank),.phsync(phsync),.pvsync(pvsync),.pblank(pblank));
//
//          DisplayCoordinatorStationC staC(.clk(clock_65mhz),.hcount(hcount),.vcount(vcount),
//          .x(11'd450), .y(10'd500), .pixel(pixel_C), .hsync(hsync),
//          .vsync(vsync),.blank(blank),.phsync(phsync),.pvsync(pvsync),.pblank(pblank));
//
//
//          assign pixel = pixel_A + pixel_B + pixel_C;


//          CoordinatorDisplay dispAll
//          (
//          .clk(clock_65mhz),
//          .xa(11'd200),
//          .xb(11'd700),
//          .xc(11'd450),
//          .hcount(hcount),
```

```
//	.ya(10'd100),
//	.yb(10'd100),
//	.yc(10'd400),
//	.vcount(vcount),
//	.hsync(hsync),
//	.vsync(vsync),
//	.blank(blank),
//	.state(switch[3:2]),
//	.phsync(phsync),
//	.pvsync(pvsync),
//	.pblank(pblank),
//	.pixel(pixel)
//	);

//wire [7:0]audioxxxx;
	StationDisplayModule stationDispModule(
	.clk(clock_65mhz),
	.hcount(hcount),
	.vcount(vcount),
	.hsync(hsync),
	.vsync(vsync),
	.blank(blank),
	.ringing(switch[2]),
	.phsync(phsync),
	.pvsync(pvsync),
	.pblank(pblank),
	.pixel(pixel)
	);

//    SoundRinging sRing(
//	.clk(clock_27mhz),
//	.ready(ready),
//	.soundOut(ads),
//	.ringing(switch[2])
//	);




	wire [7:0]ack_call;
	wire [7:0]one_byte;
	wire [7:0]serial_enable;
	wire to_serial;
```

```
	// deal with the two to_ac97 signals
//	wire [7:0] audioOutToHeadphones;
	wire [7:0] to_ac97_from_recorder=0;
	wire [7:0] audio_to_headphone;
	wire [7:0] codec_audio;
//	assign to_ac97_data = (switch[6]) ? to_ac97_from_recorder: audioOutToHeadphones;


	top_A user_end_A (
	.clk(clock_27mhz),
	.serial_ack_in(user1[31]), //receive
	.ack_call(ack_call),
	.one_byte(one_byte),
	.serial_enable(serial_enable),
	.to_serial(to_serial),
	.ready(ready),
	.call_A(switch[0]),
	.call_C(switch[1]),
	.codec_audio(codec_audio),
	.serial_data_out(user1[30]),
	.audio_to_headphone(audio_to_headphone)
	);

	fir31 fir(.clock(clock_27mhz),.reset(global_reset),.ready(ready),.x(audio_to_headphone)
	,.to_chip(to_ac97_data));
    fir31 fir2(.clock(clock_27mhz),.reset(global_reset),.ready(ready),.x(from_ac97_data)
	,.to_chip(codec_audio));

//	blob #(.WIDTH(64),.HEIGHT(64),.COLOR(24'hFF_FF_00))   // yellow!
//	paddle1(.x(11'd0),.y(10'd0),.hcount(hcount),.vcount(vcount),
//	.pixel(paddle_pixel));

   // switch[1:0] selects which video generator to use:
   //  00: user's pong game
   //  01: 1 pixel outline of active video area (adjust screen controls)
   //  10: color bars
   reg [23:0] rgb;
   wire border = (hcount==0 | hcount==1023 | vcount==0 | vcount==767);

   reg b,hs,vs;
   always @(posedge clock_65mhz) begin
	if (switch[1:0] == 2'b01) begin
	// 1 pixel outline of visible area (white)
	hs <= hsync;
```

```
        vs <= vsync;
        b <= blank;
        rgb <= {24{border}};
        end else if (switch[1:0] == 2'b10) begin
        // color bars
        hs <= hsync;
        vs <= vsync;
        b <= blank;
        rgb <= {{8{hcount[8]}}, {8{hcount[7]}}, {8{hcount[6]}}} ;
        end else begin
        // default: pong
        hs <= phsync;
        vs <= pvsync;
        b <= pblank;
        rgb <= pixel;
        end
    end

    // VGA Output.  In order to meet the setup and hold times of the
    // AD7125, we send it ~clock_65mhz.
    assign vga_out_red = rgb[23:16];
    assign vga_out_green = rgb[15:8];
    assign vga_out_blue = rgb[7:0];
    assign vga_out_sync_b = 1'b1;      // not used
    assign vga_out_blank_b = ~b;
    assign vga_out_pixel_clock = ~clock_65mhz;
    assign vga_out_hsync = hs;
    assign vga_out_vsync = vs;

    //assign led = ~{3'b000,up,down,reset,switch[1:0]};


        ////////////////////////////////////////////////////////////////////////////
    //
    // 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 #(.INIT(16'hFFFF)) reset_sr(.D(1'b0), .CLK(clock_27mhz), .Q(reset),
//                              .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
```

```
  // allow user to adjust volume
  wire vup,vdown;
  reg old_vup,old_vdown;
  debounce bup(.reset(global_reset),.clock(clock_27mhz),.noisy(~button_up),.clean(vup));
  debounce
bdown(.reset(global_reset),.clock(clock_27mhz),.noisy(~button_down),.clean(vdown));
  reg [4:0] volume;
  always @ (posedge clock_27mhz) begin
        if (global_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
  lab5audio a(clock_27mhz, global_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(global_reset),.clock(clock_27mhz),.noisy(button_enter),.clean(playback));

  // switch 0 up for filtering, down for no filtering
  wire filter;
  debounce sw0(.reset(global_reset),.clock(clock_27mhz),.noisy(switch[0]),.clean(filter));

  // light up LEDs when recording, show volume during playback.
  // led is active low
  assign led = playback ? ~{filter,2'b00, volume} : ~{filter,7'hFF};

        wire [7:0] audioOut;
        wire [1:0] byteNum;
  // record module
  recorder r(.clock(clock_27mhz), .reset(global_reset), .ready(ready),
```

```
          .playback(playback), .filter(filter), .listen(switch[7]),
          .from_ac97_data(from_ac97_data),.to_ac97_data(to_ac97_from_recorder),
                    .outAudio(audioOut), .switchWrite(1'b1), .byteNum(byteNum),
                    .msgNum(switch[5:4]),
                    .ram0_clk(ram0_clk),
                    .ram0_we_b(ram0_we_b),
                    .ram0_address(ram0_address),
                    .ram0_data(ram0_data),
                    .ram0_cen_b(ram0_cen_b) );

          display_16hex dsp(global_reset, clock_27mhz,
{16'd0,8'd0,serial_enable,8'd0,one_byte,8'd0,ack_call},
          disp_blank, disp_clock, disp_rs, disp_ce_b,
          disp_reset_b, disp_data_out);


   // 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 = ready;
   assign analyzer3_data = {from_ac97_data, to_ac97_data};

  assign analyzer2_data =  {   8'b0,ack_call};
  assign analyzer2_clock = clock_27mhz;
endmodule

////////////////////////////////////////////////////////////////////////////////
//
// xvga: Generate XVGA display signals (1024 x 768 @ 60Hz)
//
////////////////////////////////////////////////////////////////////////////////

module xvga(input vclock,
          output reg [10:0] hcount,       // pixel number on current line
          output reg [9:0] vcount,        // line number
          output reg vsync,hsync,blank);

   // horizontal: 1344 pixels total
   // display 1024 pixels per line
```

```
   reg hblank,vblank;
   wire hsyncon,hsyncoff,hreset,hblankon;
   assign hblankon = (hcount == 1023);
   assign hsyncon = (hcount == 1047);
   assign hsyncoff = (hcount == 1183);
   assign hreset = (hcount == 1343);

   // vertical: 806 lines total
   // display 768 lines
   wire vsyncon,vsyncoff,vreset,vblankon;
   assign vblankon = hreset & (vcount == 767);
   assign vsyncon = hreset & (vcount == 776);
   assign vsyncoff = hreset & (vcount == 782);
   assign vreset = hreset & (vcount == 805);

   // sync and blanking
   wire next_hblank,next_vblank;
   assign next_hblank = hreset ? 0 : hblankon ? 1 : hblank;
   assign next_vblank = vreset ? 0 : vblankon ? 1 : vblank;
   always @(posedge vclock) begin
         hcount <= hreset ? 0 : hcount + 1;
         hblank <= next_hblank;
         hsync <= hsyncon ? 0 : hsyncoff ? 1 : hsync;  // active low

         vcount <= hreset ? (vreset ? 0 : vcount + 1) : vcount;
         vblank <= next_vblank;
         vsync <= vsyncon ? 0 : vsyncoff ? 1 : vsync;  // active low

         blank <= next_vblank | (next_hblank & ~hreset);
   end
endmodule

///////////////////////////////////////////////////////////////////////////////
//
// Record/playback
//
///////////////////////////////////////////////////////////////////////////////

module recorder(
  input wire clock,                  // 27mhz system clock
  input wire reset,                  // 1 to reset to initial state
  input wire playback,               // 1 for playback, 0 for record
  input wire listen, //listen to voicemail
  input wire switchWrite,
```

```
  input wire [1:0] msgNum,
  input wire ready,                  // 1 when AC97 data is available
  input wire filter,                 // 1 when using low-pass filter
  input wire [7:0] from_ac97_data, // 8-bit PCM data from mic
  output reg [7:0] to_ac97_data,         // 8-bit PCM data to headphone
  output [7:0] outAudio,
  output [1:0] byteNum,
  output ram0_clk,
  output ram0_we_b,
  output [18:0] ram0_address,
  inout [35:0] ram0_data,
  output ram0_cen_b
);
   // test: playback 750hz tone, or loopback using incoming data
   wire [19:0] tone;
   tone750hz xxx(.clock(clock),.ready(ready),.pcm_data(tone));

        wire [7:0] audioOut;
        reg [7:0] audioIn;
        //wire [1:0] byteNum;

        Voicemail VMModule(
        .clk(clock),
        .reset(reset),
        .enable(1'b0),
        .writePulse(ready),
        .readPulse(ready),
        .recording(!playback),
        .listening(listen),
        .audioIn(audioIn),
        .msgNum(msgNum),
        .audioOut(audioOut),
        .byteNum(byteNum),

        //the ram control signals
        .ram_clk(ram0_clk),
        .ram_we_b(ram0_we_b),
        .ram_address(ram0_address),
        .ram_data(ram0_data),
        .ram_cen_b(ram0_cen_b)
        );

        reg [7:0] cntr = 8'd11001010;
```

```
  always @ (posedge clock) begin

        if (ready) begin
        // get here when we've just received new data from the AC97
        //to_ac97_data <= playback ? tone[19:12] : from_ac97_data;
        //to_ac97_data <= audioOut;
        //to_ac97_data <= playback ? audioOut : from_ac97_data;
        //if(playback) to_ac97_data <= from_ac97_data;
        //to_ac97_data <= (listen) ? audioOut : from_ac97_data;

        to_ac97_data <= playback ? audioOut : from_ac97_data;


        //audioIn <= from_ac97_data;
        if (switchWrite) begin
                audioIn <= from_ac97_data;
        end else begin
                audioIn <= 8'd0;
        end


        end
  end

        assign outAudio = audioOut;

endmodule

///////////////////////////////////////////////////////////////////////////////
//
// Verilog equivalent to a BRAM, tools will infer the right thing!
// number of locations = 1<<LOGSIZE, width in bits = WIDTH.
// default is a 16K x 1 memory.
//
///////////////////////////////////////////////////////////////////////////////

module mybram #(parameter LOGSIZE=14, WIDTH=1)
        (input wire [LOGSIZE-1:0] addr,
        input wire clk,
        input wire [WIDTH-1:0] din,
        output reg [WIDTH-1:0] dout,
        input wire we);
  // let the tools infer the right number of BRAMs
  (* ram_style = "block" *)
```

```
  reg [WIDTH-1:0] mem[(1<<LOGSIZE)-1:0];
  always @(posedge clk) begin
        if (we) mem[addr] <= din;
        dout <= mem[addr];
  end
endmodule

///////////////////////////////////////////////////////////////////////////////
//
// 31-tap FIR filter, 8-bit signed data, 10-bit signed coefficients.
// ready is asserted whenever there is a new sample on the X input,
// the Y output should also be sampled at the same time.  Assumes at
// least 32 clocks between ready assertions.  Note that since the
// coefficients have been scaled by 2**10, so has the output (it's
// expanded from 8 bits to 18 bits).  To get an 8-bit result from the
// filter just divide by 2**10, ie, use Y[17:10].
//
///////////////////////////////////////////////////////////////////////////////

module fir31(
 input wire clock,reset,ready,
 input wire signed [7:0] x,
 output reg signed [7:0] to_chip=0
);
 // for now just pass data through
 reg signed [7:0] sample [31:0];       //sample arrays
 reg signed [17:0] accumulator =0;
        reg [17:0] y;  //accumulator
        reg [5:0] count =0;             //counter for first 32 samples
        reg [4:0] i=0;                  //counter for calculation
        wire signed [9:0] coeff;        //coeff to convolution
        reg [5:0] iii=31;               //counter for shift sample one array
        reg start = 1;                  //indicator to start shifting
        wire [4:0] index;               //index to find corresponding coeff
        assign index = i;
 always @(posedge clock) begin
        if (reset) //reset sample, count, accumulator
        begin
        count<=0;
        accumulator<=0;
        sample[0]=0;
        sample[1]=0;
        sample[2]=0;
        sample[3]=0;
```

```
sample[4]=0;
sample[5]=0;
sample[6]=0;
sample[7]=0;
sample[8]=0;
sample[9]=0;
sample[10]=0;
sample[11]=0;
sample[12]=0;
sample[13]=0;
sample[14]=0;
sample[15]=0;
sample[16]=0;
sample[17]=0;
sample[18]=0;
sample[19]=0;
sample[20]=0;
sample[21]=0;
sample[22]=0;
sample[23]=0;
sample[24]=0;
sample[25]=0;
sample[26]=0;
sample[27]=0;
sample[28]=0;
sample[29]=0;
sample[30]=0;
sample[31]=0;
i<=0;
end

begin
if ((ready)&&(count<32)) //if one sample arrive and it is the first 32 sample
begin
sample[count]=x;        //add to array
count<=count+1;
end
else if ((count==32)&&(~ready))
// if there are 32 samples and no new one comes
begin
if (start)          //start shitfing
        begin
        if (iii<31)         //loop to shift
        begin
```

```
                sample[iii]=sample[iii+1];
                iii<=iii+1;
                end
                else
                begin
                sample[31]=x; //after shifting 31 samples, take in new sample
                i<=0;           //reset values
                accumulator<=0;
                iii<=0;
                start<=0;
                end
                end

        else    //after shifting
                begin
                if (i<31)       //loop for calculation
                begin
                accumulator <= accumulator + coeff * sample[31-i];
                i<=i+1;
                end
                else
                begin
                y<=accumulator;         //set y to accumulator value
                to_chip[7]<=y[17];
                to_chip[6]<=y[16];
                to_chip[5]<=y[15];
                to_chip[4]<=y[14];
                to_chip[3]<=y[13];
                to_chip[2]<=y[12];
                to_chip[1]<=y[11];
                to_chip[0]<=y[10];
                end
                end
        end
        else if ((count==32) &&(ready))         //new sample arrives
        begin
        start<=1;       //enable shifting
        end
        end
end
  coeffs31 xxx(.index(index),.coeff(coeff));
  //initiate module for finding coeff
endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
//
// Coefficients for a 31-tap low-pass FIR filter with Wn=.125 (eg, 3kHz for a
// 48kHz sample rate).  Since we're doing integer arithmetic, we've scaled
// the coefficients by 2**10
// Matlab command: round(fir1(30,.125)*1024)
//
////////////////////////////////////////////////////////////////////////////////

module coeffs31(
  input wire [4:0] index,
  output reg signed [9:0] coeff
);
  // tools will turn this into a 31x10 ROM
  always @(index)
        case (index)
        5'd0:  coeff = -10'sd1;
        5'd1:  coeff = -10'sd1;
        5'd2:  coeff = -10'sd3;
        5'd3:  coeff = -10'sd5;
        5'd4:  coeff = -10'sd6;
        5'd5:  coeff = -10'sd7;
        5'd6:  coeff = -10'sd5;
        5'd7:  coeff = 10'sd0;
        5'd8:  coeff = 10'sd10;
        5'd9:  coeff = 10'sd26;
        5'd10: coeff = 10'sd46;
        5'd11: coeff = 10'sd69;
        5'd12: coeff = 10'sd91;
        5'd13: coeff = 10'sd110;
        5'd14: coeff = 10'sd123;
        5'd15: coeff = 10'sd128;
        5'd16: coeff = 10'sd123;
        5'd17: coeff = 10'sd110;
        5'd18: coeff = 10'sd91;
        5'd19: coeff = 10'sd69;
        5'd20: coeff = 10'sd46;
        5'd21: coeff = 10'sd26;
        5'd22: coeff = 10'sd10;
        5'd23: coeff = 10'sd0;
        5'd24: coeff = -10'sd5;
        5'd25: coeff = -10'sd7;
        5'd26: coeff = -10'sd6;
        5'd27: coeff = -10'sd5;
```

```
    5'd28: coeff = -10'sd3;
    5'd29: coeff = -10'sd1;
    5'd30: coeff = -10'sd1;
    default: coeff = 10'hXXX;
    endcase

endmodule
```