UNIVERSITY OF CALIFORNIA AT BERKELEY
COLLEGE OF ENGINEERING
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

CHECKPOINT 4

Basic i50Phone

1.0 Introduction

Throughout the semester, you have been building the basic components needed for creating a wireless phone system. In Checkpoint 1, you designed the audio component, which reads audio input from a microphone and plays it back on your local speakers. In Checkpoint 2, you designed the display and user interface components. In Checkpoint 3, you designed a means of transferring generic data between two systems via wireless communications. In this checkpoint you will put all of these components together.

The primary goal of this checkpoint is to define a communications protocol that allows two FPGAs to selectively exchange audio data. This goal involves several steps:

1. Defining the **application layer protocol**, i.e. giving meaning to the 41-byte packets that are being exchanged.
2. Designing a communications FSM that handles the handshaking between systems.
3. Designing an intuitive user interface that allows you to connect with another station.

2.0 Prelab

The region in the dotted red line shown in the figure below shows the components that you will be adding in Checkpoint 4.

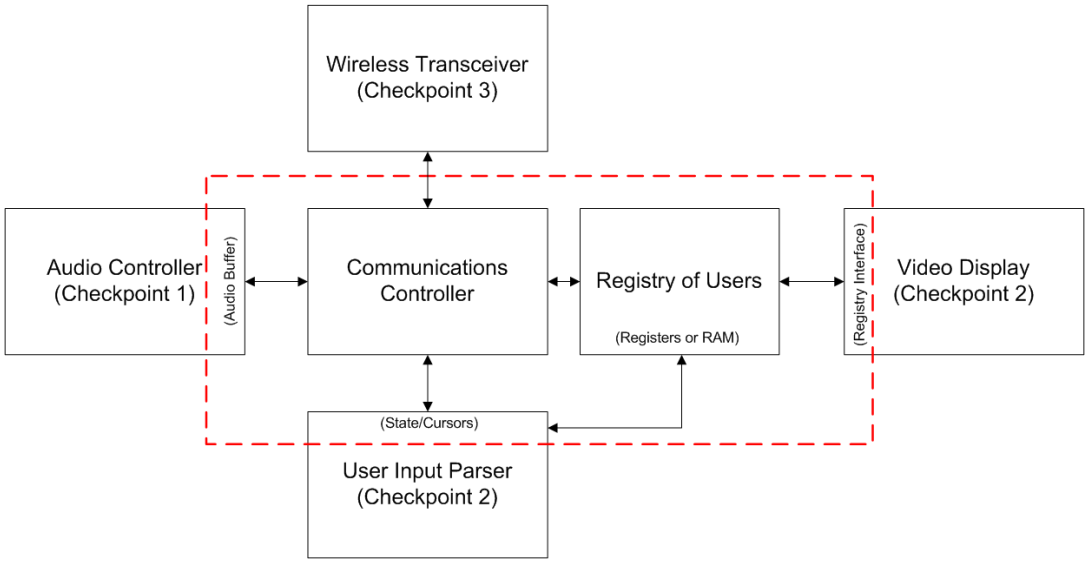

Figure 1: Checkpoint 4 Additions

This region encompasses the changes that you will need to make to the previous checkpoints, and the modules you will be adding:

1. **Checkpoint 1** – The audio buffer must allow sending and receiving audio data from the wireless transceiver.
2. **Registry** – Users will be coming in and out of channels, and in and out of calls. A registry is need to keep track of who is connected.
3. **Checkpoint 2**
   a. The input parser must support the complete navigation of the phone system, and send appropriate messages to the communications module.
   b. The video display module must support reading a registry of users in the channel, a registry of connected users, and a console that displays messages.
4. **Communications** – This module ties everything together and handles all of the handshaking between two stations, and within the local phone system.

3.0 Lab Procedure

Remember to **manage your Verilog, projects and folders well**. Doing a poor job of managing your files can cost you **hours of rewriting code**, if you accidentally delete your files.

3.1 AudioBuffer.v

The audio buffer in Checkpoint 1, shown below, only supported local record and play back. This was implemented by using a large FIFO that stored 8 seconds of voice.

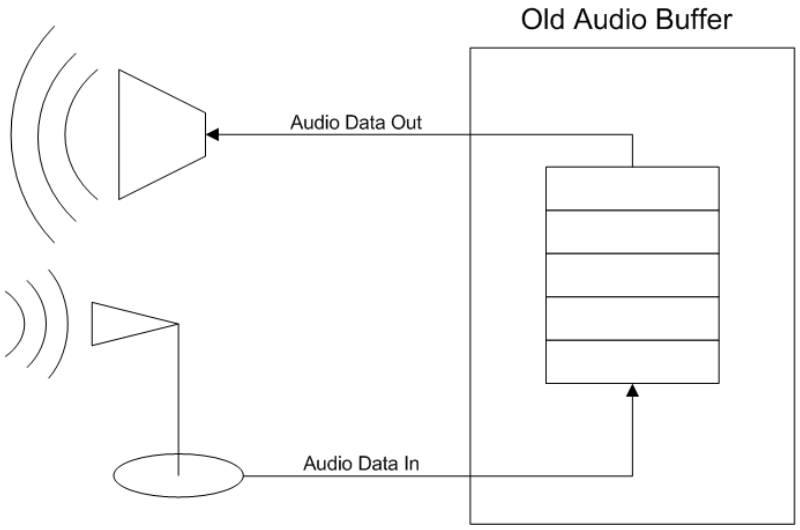

Figure 2: Checkpoint 1 Audio Buffer

In this checkpoint, such a large FIFO is unnecessary. Instead, you be changing the functionality to support **sending to and receiving from a remote source**. To do this, you will change the design of the audio buffer to look more like the figure below:

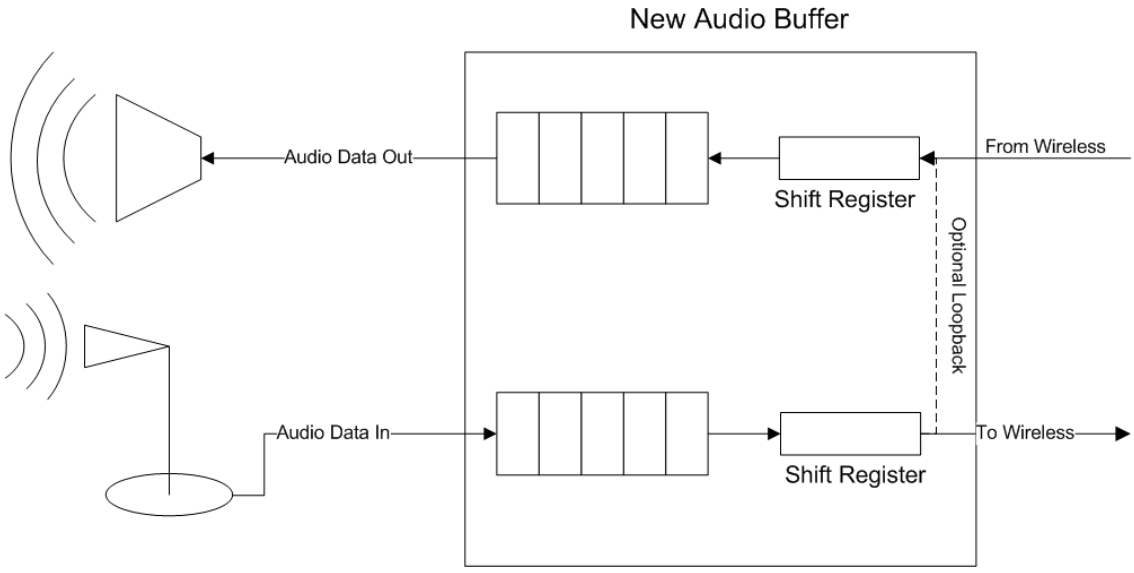

Figure 3: New Audio Buffer

The large asynchronous FIFO in checkpoint 1 will be replaced by two small asynchronous FIFOs, a single synchronous fifo, and two shift registers. Note that both of the new FIFOs must remain asynchronous because they are crossing clock domains. For debugging the new audio buffer, you may choose to support a loopback mode, such that the microphone input is played back locally.

Below is a port specification for the new audio buffer:

Signal	**Width**	**Dir**	**Description**
`Clock`	1	I	The 27MHz clock signal
`Reset`	1	I	The reset signal
`AudioClock`	1	I	The audio clock signal
`PCM_Request`	1	I	Request from local AC97 controller for data
`FromAudio`	32	I	Audio data from local audio controller
`FromAudioValid`	1	I	Indicates that FromAudio is valid
`ToAudio`	32	O	Audio data going to local audio controller
`ToAudioValid`	1	O	Indicates that ToAudio is valid
`FromWireless`	320	I	Packet of audio data from transceiver
`FromWirelessValid`	1	I	Indicates that FromWireless is valid
`ToWireless`	320	O	Packet of audio data to be send to transceiver
`ToWirelessValid`	1	O	Indicates that ToWireless is valid
`SelfLoop`	1	I	Enable audio loopback

Table 1: Port Specification for New Audio Buffer.

We have also provided a small synchronous FIFO (audio_fifo) for additional buffering. You will need to use this synchronous buffer to buffer data when you are receiving data from the wireless. When the prog_full (a signal that says when the buffer is partially full) signal goes high from the audio_fifo, you should have enough audio data to begin streaming it to the audio chip. If the fifo is every empty, you should wait to buffer audio until the prog_full signal comes high again.

3.2 UserInputParser.v

This module will now be extended to support complete navigation of the phone system, including requesting to establish calls with users, accepting and rejecting calls, and changing the display appropriately. This module must keep track of **two cursor positions**: one for the users in the channel, and one for the users connected in a call. The input parser must support obey the following constraints:

1. Only allow the cursor of a particular region to change when the focus is on that region.
2. When the cursor is on a valid user in the Channel region and "A" is pressed:
   a. Change the focus to the console
   b. Send a call request and wait for a response
      i. If "B" is pressed, cancel the request and change focus back to the Channel region
      ii. When a connection is either established or refused, change the focus to the Channel region.
      iii. Ignore incoming calls while requesting a call.
3. When the cursor is on a valid user in the Connection region and "B" is pressed:
   a. Send a disconnect signal
   b. Focus remains in the same region
4. When there is an incoming call, and you are not currently requesting a call
   a. Change the focus to the console
   b. If "A" is pressed, send "Accept Call" message and return focus to Channel
   c. If "B" is pressed, send "Reject Call" message and return focus to Channel

The table below is a port specification for the new input parser module.

Signal	**Width**	**Dir**	**Description**
`Clock`	1	I	The 27Mhz `Clock` signal
`Reset`	1	I	The `Reset` signal.
`(N64 Buttons)`	30	I	Buttons from N64ButtonParser.v
`SpeakerVolume`	5	O	Speaker volume
`SpeakerMute`	1	O	Speaker mute
`MicVolume`	5	O	Mic volume
`MicMute`	1	O	Mic mute
`Channel`	4	O	Wireless channel* (See note below)
`Focus`	2	O	Region focus
`ChanCursor`	3	O	Position of the channel cursor
`ConnCursor`	3	O	Position of the connection cursor
`ChanSelectValid`	1	I	Indicates that the name that the channel cursor is pointing to a valid user entry
`ConnSelectValid`	1	I	Indicates that the name that the connection cursor is pointing to a valid user entry
`IncomingCall`	1	I	Indicates an incoming call

AcceptCall	1	O	Accept the incoming call
RejectCall	1	O	Reject the incoming call
RequestCall	1	O	Request to connect with the user at ChanCursor
DisconnectCall	1	O	Disconnect from the user at ConnCursor
ConnEstablished	1	I	Indicates that a call was successfully established.
CallRejected	1	I	Indicates that a call was not successfully established.

Table 2: Port Specification for New Input Parser Module

*** Note: You do not need to support channel changing in this checkpoint. You may simply Reset to your assigned channel.**

Note that this module does not have to keep track of valid entries. This is done in the Registry module described below. Thus, the Registry notifies the input parser whether or not a cursor is on a valid entry.

3.3 Registry.v and VideoDisplay.v

You will need a module that keeps track of which users are in the channel, and which users you are connected to. This module maintains not only the user names, but also the source addresses associated with each user. This information is needed to notify the communications module where to send a call or disconnect request. For simplicity, the registry keeps track of valid entries through **refreshes and timeouts** – there is no need to support a way of explicitly removing an entry. Thus, when you want to disconnect from a call, you simply send the disconnect signal to the communications controller, which will stop sending packets to the user, and the entry will simply time out. For the final product, set the timeout period to 5 seconds, but for simulation and testing purposes, you may set it lower.

When a new user arrives, add the name and associated source address to any empty entry. **You do not have to support a registry with more than 8 entries**.

Depending on how you implemented the video display module in Checkpoint 2, you may either place the registry inside the video display module or implement it as a separate unit. In either case, the registry combined with the video display module must support the following interface with the communications and input parser modules:

Signal	Width	Dir	Description
AnnounceUsername	64	I	Corresponds to the username of the station that just sent an Announce packet (see below)
AnnounceSource	8	I	Corresponds to the source address of the user that just sent an Announce packet
AnnounceValid	1	I	Indicates that AnnounceUsername and AnnounceSource are valid.
CallUsername	64	I	Corresponds to the username of the station that just established a call.
CallSource	8	I	Corresponds to the source address of the user that just established a call.
CallValid	1	I	Indicates that the user specified by CallUsername

			and CallSource has established an exclusive connection for voice communication.
ChanCursor	3	I	Channel cursor position from input parser
ChanSelUsername	64	O	Corresponds to username at ChanCursor
ChanSelSource	8	O	Corresponds to source address of user at ChanCursor
ChanSelValid	1	O	Indicates that the entry at ChanCursor is valid, and ChanSelUsername and ChanSelSource are valid.
ConnCursor	3	I	Cursor position from input parser for connection region.
ConnSelUsername	64	O	Corresponds to username at ConnCursor
ConnSelSource	8	O	Corresponds to source address of user at ConnCursor
ConnSelValid	1	O	Indicates that the entry at ConnCursor is valid, and ConnSelUsername and ConnSelSource are valid.
IncomingCall	1	I	Indicates an incoming call from CallUsername.
ConnEstablished	1	I	Indicates a call was successfully established
RequestCall	1	I	Indicates a call request to ChanSelUsername.
CallRejected	1	I	Indicates that the call was rejected.
DisconnectCall	1	I	Indicates a request to disconnect from ConnSelUsername.
ConnTimedOut	1	I	Indicates a call timed out.

Table 3: Interface of Registry with Communications and Input Parser

You may use either the provided RAM blocks or simple registers to keep track of user entries.

In addition to storing and display user names, your system must also be able to display several different types of messages to the console region. **The console must remember up to 8 lines of messages, where each line is 32 ASCII characters**. Once the console fills up, the newest message will push the oldest one out, creating a scrolling effect for the history. The following is a list of messages that your module will display:

Message	**Condition**
Call From: Username Acc(A)/Rej(B)	For incoming calls from Username
Dialing User: Username Cancel(B)	For outgoing calls to Username
Connection Established	A connection has been successfully established.
Connection Rejected	Your call was rejected
Connection Terminated	You request to disconnect from a connected user
Connection Timed Out	Your call timed out

Table 4: Messages to Display

3.4 Communications.v

This is the **main module you will build for this final checkpoint**. This module defines the application layer protocol and handles all handshaking between two systems. We strongly recommend that you design this module completely before writing any Verilog for it. **Any corner cases you may have forgotten during design may cause you many hours of redesigning**.

Recall from Checkpoint 3 that your transceivers exchanged packets with 41 bytes of payload. The first byte will be defined as the application layer header, and the remaining 40 bytes will be the application layer payload. The following table describes the format of the 41-byte packet:

Header (1 byte)	**Type**	**Payload (40 bytes)**	**Description**
8'h0	Announce	{256'hX, Username}	Broadcast (send to 0xFF) your presence to the channel. Send every 250ms except when in a call.
8'h1	Init Call	{256'hX, Username}	Represents a call request.
8'h2	Accept Call	{256'hX, Username}	Accept a call request
8'h3	Reject Call	{256'hX, Username}	Reject a call request.
8'h4	Ack Init	{256'hX, Username}	Acknowledgment of "Init Call," but is neither an accept nor reject (see below)
8'h5	Data	Audio Data	Payload contains audio data
8'h6	Ready	320'hX	Acknowledgment of "Accept Call." Also used to keep active connection alive in the absence of data.

Table 5: Packet Format

Your communications module must support the following behavior:

1. Sit idle and receive packets by default.
2. Send an Announce packet every 250 milliseconds.
3. If an Announce packet is received at any point, notify the Registry/VideoDisplay.
4. If you initiate a call request, do the following:
    a. Send Init Call packet and wait for response
        i. If response is Ack Init, return to (a)
        ii. If response is Accept Call, send a Ready packet and enter a call session.
        iii. If response is Reject Call, return to idle receive state.
        iv. If you press "B" (cancel), return to idle receive state
        v. If you receive no response for 1 second, return to idle receive state.
    b. During the Init Call/Ack Init exchange, you do not need to send Announce packets, but you must still be able to receive and handle them. You may ignore non-Announce packets from other sources.
5. If you receive an Init Call during your idle receive state, do the following:
    a. Send Ack Init packet and packet and wait for response

i. If you press “A” (accept), send an Accept Call packet and enter a call session.
ii. If you press “B” (reject), send a Reject Call packet and return to idle receive state.
iii. If you receive an Ack Init packet, return to (a).

b. During this exchange, you do not need to send Announce packets, but you must still be able to receive and handle them. You may ignore non-Announce packets from other sources.

6. During a call session,
   a. If you receive an audio data packet, notify the audio buffer and reset the timeout counter.
   b. If you receive a Ready packet, reset the timeout counter
   c. Send audio data whenever it is ready.
   d. Receive Announce packets and notify Registry/Video Display
   e. Optionally send Ready packets to keep the connection alive in the absence of audio data (if you choose to implement silence suppression to reduce bandwidth usage).
   f. If you manually disconnect the call, return to the idle receive state.
   g. If neither audio data nor Ready is received in 1 sec, return to idle receive state.

The following table defines the interface with the registry, input parser, and audio buffer:

Signal	**Width**	**Dir**	**Description**
`RequestCall`	1	I	Represents a call request
`CancelRequest`	1	I	Cancels a call request
`DisconnectCall`	1	I	Disconnect from a call session
`IncomingCall`	1	O	Indicates an incoming call
`AcceptCall`	1	I	Accept a call request
`RejectCall`	1	I	Reject a call request
`CallRejected`	1	O	Call request was rejected by remote user
`ConnEstablished`	1	O	Call request was accepted by remote user
`ConnTimedOut`	1	O	Call session timed out
`MyUsername`	64	I	Your username
`MySourceAddress`	8	I	Your source address
`DestinationAddress`	8	I	Destination address for call requests
`AnnounceUsername`	64	O	Username from Announce packet
`AnnounceSource`	8	O	Source address associated with AnnounceUsername
`AnnounceValid`	1	O	Indicates AnnounceUsername and AnnounceSource are valid
`CallUsername`	64	O	Username of user in call session
`CallSource`	8	O	Source address associated with CallUsername
`CallValid`	1	O	Indicates CallUsername and CallSource are valid
`AudioDataRX`	320	O	Received audio data
`AudioDataRXValid`	1	O	Indicates AudioDataRX is valid
`AudioDataTX`	320	I	Audio data to transmit

AudioDataTXValid	1	I	Indicates AudioDataTX is valid

Table 6: Interface of Communications with Registry, Input Parser, and Audio Buffer

The transceiver from Checkpoint 3 should be instantiated within the Communications module.

The following table illustrates a typical exchange of packets to establish a call between two stations.

CALLER	RECEIVER	CALLER UI	RECEIVER UI
ANNOUNCE	.		Populate CALLER in Channel User List
	ANNOUNCE	Populate RECEIVER in Channel User List	
ANNOUNCE	.		Persist CALLER in Channel User List
.	.	Persist RECEIVER in Channel User List	
.	.		
.	.		
INIT CALL	.	Dialing User RECEIVER	Incoming Call CALLER
	ACK INIT		
INIT CALL			
.	ACK INIT		
.	.		
.	.		
.	.		
		Connection Established Populate RECEIVER in Call User List	
.	ACCEPT CALL		
			Connection Established Populate CALLER in Call User List
READY			
	DATA		
DATA			
	DATA		
DATA			
	No Response		
DATA	.		
	.		
DATA	.		
	.		Possible Timeout
DATA	.	Connection Timed Out	
(Disconnect)	.		
.	.		Depends on direction of lost packets
.	.		
.	.		
.	.		

.	.		Possible Timeout

Table 7: Example of Handshaking and Packet Exchange

4.0 Requirements

Your system must satisfy the following requirements:

1. All of the basic user interface specifications described above
2. Ability to receive Announce packets during a conversation
3. Ability to disconnect gracefully from a call
4. Persistent 2-way communication in a clear channel between two stations using the same bit file.
   a. In a clear channel, communication must not randomly end within a minute
   b. You do not need to implement n-way communication, where n > 2.
   c. You do not need to support 2-way communication in a channel where a call is already occurring.
5. Latency between speaking into microphone and hearing on remote station must be less than 1 second.
6. Your solution does not need to be compatible with the sample TA solution.

4.1 Extra Credit Features

Notice in the packet format table that we only defined headers for 7 out of 256 possible values. This leaves plenty of room for extensibility. **However, do not start working on extra credit features until you have completed the basic requirements listed above. Any extra features added to a system that doesn't satisfy the basic requirements will not count.**

Extra credit is worth up to 20% of your entire course project grade and will be applied post-curve.

The following is a list of possible extra credit features

1. Text messaging
2. Talk with more than one person simultaneously
3. Audio effects (e.g. reverberations)
4. Using the console region for a shared game (e.g. Pong, Tetris, Guitar Hero)
5. Record and play back a long conversation using SDRAM
6. Ring tones
7. Audio/video conferencing
8. Anything special, cool, clever, creative, unique, and/or challenging

4.2 Additional Information

Note that we will not be providing any skeleton Verilog files for this checkpoint because the additions in Checkpoint 4 are design-dependent. Several of the files you will be modifying are from previous checkpoints (AudioBuffer.v and VideoDisplay.v), so you already have the basic structure for these modules. We have simply defined the minimal additions you will be making to those modules in this document. The only files we will be providing are fifo_async.v, which is a small asynchronous FIFO for crossing audio/system clock domains, and audio_fifo.v, which is a synchronous FIFO for additional audio buffering if you need it.

You may choose to implement Registry.v either with the console and registry of users as part of the VideoDisplay unit or as a separate unit. Several groups have already begun implementing their own registry system inside VideoDisplay, and we will not force them to change their design.

The interface between the Communications module and the rest of the system has been fully specified above. However, you may choose to implement a different interface or a different protocol altogether. **If you come up with a better protocol that better utilizes the available bandwidth, feel free to implement it**. However, your system must still satisfy the basic requirements listed above.

Thus, pick your favorite FPGA_TOP2+.v file and add to it.