

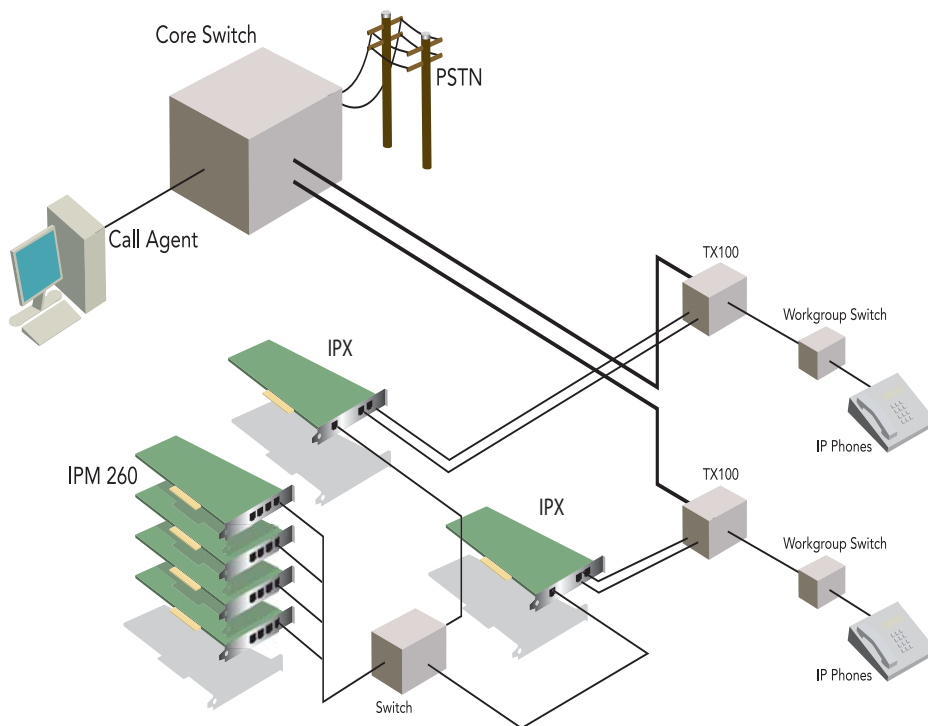
Using the IPM-260 with the IPX for VoIP Recording

Issue Date: 3/28/07

This paper explains how to use the IPM-260 to record media (RTP) packets. This document provides a high level explanation of a logging system designed with AudioCodes' IPX, along with specific instructions for configuring the IPM-260 for use. This document assumes the reader is familiar with the capabilities of the IPX board as described in the *IPX Integration Guide*.

The Tapping System

The IPX is AudioCodes' patent pending VoIP tapping component. It is designed with packet filtering resources, and VoIP protocol stacks capable of decoding call control and proprietary D-channel data. The on-board session management resource tags each connected call with a unique Session ID. This ID allows the user application to track all events and the RTP packets associated with each active call on the line. If a call needs to be recorded, the IPX's RTP forwarding resources are used to pass the media packets to the IPM-260. The IPM-260 is a VoIP communication board with transcoding capabilities. In this application it is responsible for summation and recording the voice packets to a file. The following diagram shows one example of a tapping system using IPX and IPM-260 boards.



The IPX

The IPX provides call control and D-channel decoding, plus RTP forwarding services. It is positioned on the VoIP network and forwards media packets selected by the user application to the recording component. This section provides a high level discussion of the features and capabilities of the IPX. For a detailed explanation about the capabilities and development environment refer to the *IPX User and Integration Guide*. The following is an example of the basic use of the IPX:

1. Two ethernet ports collect upstream (Tx) and downstream (Rx) traffic. The two sides of the conversation are not summed on the IPX. **NOTE:** Upstream and downstream are always discussed from the point of reference from the endpoint. The station transmits upstream to the local PBX.
2. Signaling packets are passed to the appropriate protocol decoding stack. The IPX is designed so that multiple VoIP protocols can be decoded on the same board. The protocol stacks decode all signaling, terminal and media control information. The event mailer reports events to the user application.
3. When the call is connected and an RTP session is established, the session manager assigns a Session ID.
4. The user application, relying on the Session ID, controls packet forwarding to the destination recording device (IPM-260). Upstream and downstream traffic are treated as two separate streams on the IPX and are forwarded to two separate ports on the IPM-260.

The IPM-260 Board

The IPM-260 is a PCI form factor VoIP Communication board designed predominately for VoIP Gateway applications. This section explains how the IPM-260 is used to receive incoming traffic from the IPX and process it for recording. In this application the IPM-260 performs the following tasks:

- receive incoming RTP packets
- decoding - Vcoders: GSM-FR, GSM-EFR, AMR, G711, G732.1, G729.A, G728, G727, and EVRC_QCELP (more are supported, contact AudioCodes for a complete list)
- summation
- recording - Vcoders: GSM-FR, GSM-EFR, AMR, G711, G732.1, G729.A, G728, G727, and EVRC_QCELP (more are supported, contact AudioCodes for a complete list).

Installing the Software

Complete instructions are available in Chapter Two: Using the VoP Library of the *VoPLib User's Manual*.

1. Software Installation
2. Profiling the VoP Library
3. Perform VoP Library Functions - `acInitLib/acCloseLib`

Event Reporting

The board places a new event packet in the outgoing status buffer every time a new event is detected. This buffer is an internal buffer, shared between the board and the PCI bus, and common to all of the board's channels.

In this application, the user is only interested in obtaining error events. To monitor board errors and handle an event, call ***acGetEvent()*** to get the oldest event from the board's status buffer and the related information structure. The event handling methodology can be configured for polling-oriented or blocking. It is the User's responsibility to make sure events are collected from the buffer in a timely fashion. A list of board error events can be found in the VopLib API Reference Manual.

Application Configuration

When using the v5.0 SDK, the AudioCodes IPM-260 offers various recording scenarios. The following section provides an explanation of when to use each recording scenario and how your board and channels must be configured. Four recording methods are introduced in this document:

- Using [three channel](#) resources to record (summed conversation)
- Using [two channel](#) resources to record (summed conversation) **NOTE:** This method requires firmware 5.00.18.002 or greater.
- [Half-duplex](#) recording, no summation
- [Stereo recording](#) - **NOTE:** This recording model is not part of the demonstration application, but has been included in this documentation.

This document provides application development instructions for each of these recording scenarios.

Developer's Notes

This demonstration was originally built when running the IPX board. To ensure that this application supports both the IPX and the IPX-C add the following:

```
if (bInfo.AdapterType == (NET_TAP_CARD | IPX) ||  
(bInfo.AdapterType == (NET_TAP_CARD | IPX_C)))
```

Also before your application calls ***MTIpEnableSignalingProtocol()*** have it call ***MTIpDisableSignalingProtocol()***. At times the protocol is already enabled because the application was previously run or SmartView was run and enabled for the protocol. This eliminates the possibility of a returned error.

Three Channel Recording

The following section explains how an application must be developed when using three channels on the IPM-260 for summation and recording. To preserve channel resources, users should follow the two channel recording scenario. The two channel recording application model can be used when encoding is not required - the file can be saved into one of the following CODECs: G.711, LinearPCM, G.711MuLaw64 or in the same CODEC used by the network. When encoding is required then the three channel recording model must be applied.

NOTE: When running the demonstration application that supports v 5.0, the **NumRecordingApplication** parameter in the **ipRecord.ini** file must be set to two (2) to use this application method. Should encoding be required, then this parameter must be set to three (3) for the three channel method of recording.

Board Initialization & Configuration

To prepare the board for operation in this tapping environment the IPM-260 must be assigned an IP address, and subnet address, plus the TDM options must be set.

1. Initialize the VoP library using the `acinitLib()` function.
2. Initialize all boards in the system. This is a two step process first by invoking **`acGetDefaultBoardParameters()`** to obtain board defaults. Modify the parameters which require configuration and then set these values to the board when using the **`acOpenBoard()`** function. When the board is opened and configuration is complete the number of type of boards detected is returned by the `acinitLib()` function.

The following board parameters should be set:

`m_boardParam.DownloadableFiles.ImageFile` - this parameter points to the board's firmware.

`Im_boardParam.NetworkSettings.BoardIPAddr` - the IP address for this board

`m_boardParam.NetworkSettings.BoardSubNetAddr` - the SubNet address for this board

TDM Bus Settings:

`m_boardParam.TDMBusSettings.TDMBusType` - this example relies on the H.100 bus

`m_boardParam.TDMBusSettings.TDMBusClockSource` - this example relies on the Master setting

`m_boardParam.TDMBusSettings.TDMBusSpeed` - the fastest bus speed is used in this example

`m_boardParam.TDMBusSettings.TDMBusMasterSlaveSelection` - user preference, this example relies on Master mode

PSTN Interface Settings:

This use of the IPM260 does not require any PSTN interfaces, therefore any

trunk protocols must be disabled.

m_boardParam.TrunkConfig[i].ProtocolType = acPROTOCOL_TYPE_NONE

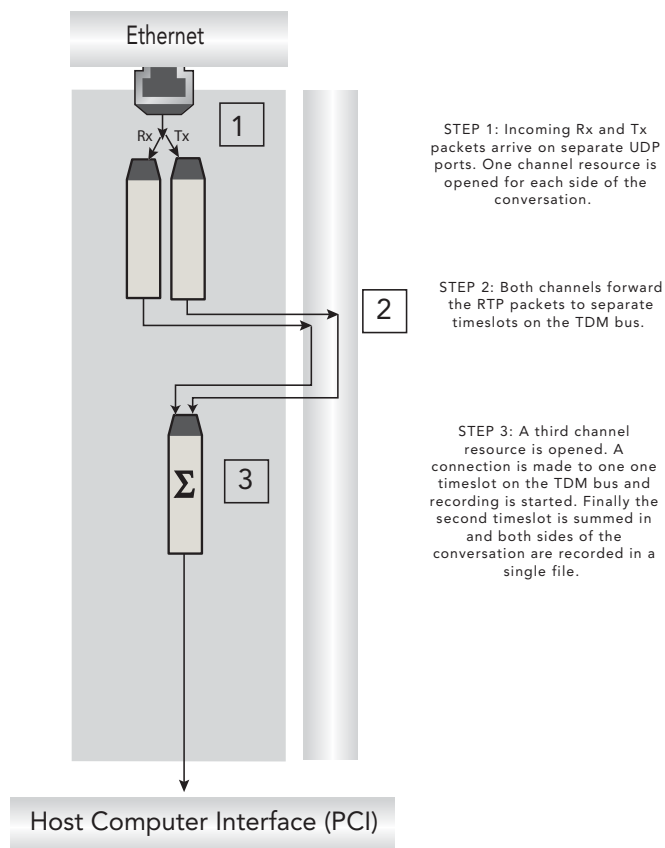
3. Invoke **acGetEvent()**. Wait for an acEV_BOARD_STARTED event to indicate the board has started successfully. Any other event indicates some type of board failure.

Channel Initialization & Configuration

For each call that needs to be recorded on the network, three unique channels (resources) must be opened on the IPM-260. The upstream and downstream packets of the conversation are not summed on the IPX. As a result, RTP packets are forwarded as two separate streams to the IPM-260. Two channels are required to receive incoming packets encompassing both sides of a single conversation. These channels are then connected via the local TDM bus to a third channel which sums the conversation and then records to a file.

The following diagram shows an example of how both sides of one conversation are recorded using the IPM-260. **NOTE:** When recording without summation, only two channel resources are required. This will be explained later in this document.

Figure 1-1: A Single Call Session is Recorded Using the IPM-260



This section outlines the required channel configuration on the IPM-260.

Configure Receive Channels

The IPX forwards both streams of the VoIP conversation to two separate channels (each with a single port) on the IPM-260.

NOTE: This example assumes the user is invoking the SmartWORKS API *MTIpSetSessionMediaDest()* API to control media forwarding to the IPM260.

On the IPX board, the *MTSetSessionMediaDest()* API sets the recording destination for a specified Session ID. When using this API, the recording device's IP Address and UDP port numbers are provided to the IPX. Since the RTP packets are not summed on the IPX, two unique port numbers must be set - one to receive downstream (Rx) traffic and the other for upstream traffic (Tx). To do this, application developers must first open two receiving channels on the IPM-260, and then invoke an API to obtain the port numbers assigned to each upon initialization. The following section explains this logic and shows the configuration requirements when opening the two recording channels on the IPM-260.

NOTE: Some parameters can be changed “on-the-fly” meaning these values can be modified after the channel has been opened. Parameters which are not labeled as such must be set when the channel is opened.

The following steps are recommendations on how to open and configure two channels for receiving incoming RTP packets on the IPM-260:

1. Open & Configure Channel two channels. This is a two step process where the user invokes *acGetDefaultChannelParameters()* to obtain default settings. The user then modifies applicable parameters and then invokes *acOpenChannel()* to set these values and open the channel. This function returns the channel handle to the user application. The *acTextChannelParam* structure holds all configuration parameters. The following can be used as a guideline for channel configuration:

Voice Settings:

ChannelParam.VoiceSettings.Coder - the CODEC of the incoming packets (this information can be obtained from the IPX when the SmartWORKS *EVT_MEDIA_SESSION_BEGIN* event has been reported)

ChannelParam.VoiceSettings.ECE - For this passive recording application, set this field to 0 to disable echo cancellation. **NOTE:** This parameter can be changed “on-the-fly”.

ChannelParam.VoiceSettings.SCE - set this to *acSilenceCompresionDisable* to disable silence compression

ChannelParam.VoiceSettings.BrokenConnectionEventTimeout - a recommend setting is 300ms. This sets the timer for how long RTP connection should be

broken before the board issues a broken connection event.

Transport Settings:

ChannelParam.TransportSettings.DisableSoftIPLoopback - set this to 1 to disable soft loopback

ChannelParam.TransportSettings.UseNIorPCI - the packets are received from the network, therefore this field is set to NI (network interface).

ChannelParam.TransportSettings.UniDirectionalRTP - set the direction of RTP packets to acRTPRxOnly since this channel is only used for receiving incoming packets

Jitter Buffer:

ChannelParam.DJBSettings.DJBufMinDelay - the delay in the jitter buffer is set to 150 ms - the board's maximum value. AudioCodes recommends setting the maximum value supported by the current SDK. **NOTE:** This parameter can be changed "on-the-fly".

- Two channels are used to receive incoming packets for a single conversation - one for each side of the conversation. These two channels output the decoded voice packets (in raw PCM format) to TDM timeslots. The following parameters must be set by the user to connect the output of the channels to the TDM:

ChannelParam.TDMBusSettings.TDMBusInputPort

ChannelParam.TDMBusSettings.TDMBusOutputPort

ChannelParam.TDMBusSettings.TDMBusInputChannel

ChannelParam.TDMBusSettings.TDMBusOutputChannel

- Once the channels are open and configured, the user must then enable the channels to receive incoming packets. The function ***acActivateRTP_RTCPChannel()*** is used to do this. When this function is invoked, users are required to pass in the IP Address of the device which is transmitting RTP packets to the IPM-260. In this case it is the active port on AudioCodes'IPX.

In this example, when RTP is activated fax transmission is ignored and therefore the T.38 option is not enabled. **NOTE:** Before closing a channel, RTP must be deactivated using the ***acDeActivateRTP_RTCPChannel()*** function.

- The user must now invoke the ***acGetRTP_RTCPAddress*** function to obtain the default port assigned to each of the open channels by the IPM-260's resource manager. The IPM-260's resource manager utilizes the following logic when managing UDP ports:

All UDP port numbering begins with port 4000. Other ports are assigned numbers incrementally by 10. Thus, when the first channel is opened it is assigned port 4000, and the second channel opened is then assigned port 4010.

When invoking this API, the channel handle must be passed into the function. The following pointers are returned to the user application:
pLocalRTPAddress - the local RTP IP Address and port

pLocalRTCPAddress - the local RTCP IP Address and port (**NOTE:** the IPX does not support RTCP)

pLocalT38Address - the local T38 IP Address and port (**NOTE:** T38 is not enabled in this example)

The port numbers must be passed back to the IPX so that the media packets are forwarded to the correct location. On the IPX, users invoke ***MTipSetSessionMediaDest()*** to define the destination of the RTP packets associated with a particular Session ID. This API allows users to specifically configure the port numbers the RTP packets are forwarded to on the destination device. When recording with summation, this is the recommended API to use. Users supply two destination IP Addresses and port numbers - one for the upstream packets (Tx) and the other for the downstream packets (Rx). These values must point to the IP Address of the IPM-260 and the two default ports numbers assigned to the channels by the IPM-260's resource manager.

NOTE: The ***acActivateRTP_RTCPChannel()*** function also allows users to assign a port number to each channel rather than accepting board defaults.

Summing and Recording

Currently, the IPX is forwarding upstream and downstream packets to two UDP ports - one per channel. On the IPM-260 two channels are opened, RTP is activated, and each channel is configured to pass the decoded voice to a TDM timeslot.

A third channel resource is now opened to sum the two sides of the conversation and records into a file. When using the IPM-260, the following process is used:

1. Open a third channel
2. Connect a third channel to one TDM timeslot.
3. Connect this same channel to the other TDM timeslot.
4. Begin recording.
5. Close the channel (stop summation, stop recording)

Open channel and configure:

1. Open a third channel. Invoke ***acOpenChannel()*** and obtain channel handle. This channel is configured exactly as the two receiving channels, except for one parameter:
ChannelParam.TransportSettings.UseNlorPCI - the packets are received from the bus, therefore this field is set to PCI.
2. Connect this third channel to one timeslot. The function ***acMakeTDMConnection()*** creates a TDM connection between two TDM timeslots. In this case, the input is one of the timeslots that a receive channel is "talking" to, and the output is a DSP resource for recording.

The following parameters illustrate the suggested configuration:

```
acTTDMCrossConnectBITMode BitMode =
    acTDM_CROSS_CONNECT_BIT_TYPE_NORMAL
acTTDMCrossConnectDuplexMode DuplexMode =
    acHALF_DUPLEX_TDM_CONNECTION
acTTDMCrossConnectOverwriteMode OverwriteMode =
```

```

acOVERWRITE_TDM_CONNECTION
acTTDMCrossConnectSwitchingOption SwitchingOption = acVOICE_ONLY
acTTDMCrossConnectBusType InputTDMBusType = acH110_TDM_BUS
acTTDMCrossConnectBusType OutputTDMBusType = acDSP_TDM_BUS
InputTDMBusPort.H110Addr.Stream = Stream (identifying the stream one of
the receive channels is connected to)
InputTDMBusPort.H110Addr.TimeSlot = TimeSlot (identifying the timeslot
one of the receive channels is connected to)
OutputTDMBusPort.DSPAddr.TrunkId = -1
OutputTDMBusPort.DSPAddr.BChannelOrCID = ChannelId (this is the DSP
resource)
  
```

Make the second connection to sum the two streams:

The timeslot that the second receive channel is “talking” to must also be connected to this third channel resource so that both sides of the conversations can be summed prior to recording. Use the ***acChannelAdditionalTimeSlotSummation()*** to connect the second side of the conversation to this channel. Prior to closing the channel, summation must be stopped with the ***acStopChannelAdditionalTimeSlotSummation()*** function.

Start Recording

Now this channel is “listening” to both sides of the conversation and the two voice streams are summed. Use the ***acRecord()*** function to start recording and set the file name and CODEC type. **Set the *PlayRecordInterface* parameter of the *acTPlayRecordParam* data structure to TDM_INTERFACE.**

The following parameters are used to set the remaining recording parameters in the *acTPlayRecordParam* data structure:

```

PlayRecordTransferType = FILE_TRANSFER
pFileName = the output file name
PlayRecordInterface = TDM_INTERFACE
recordParams.Offset = 0
recordParams.pNext = NULL
recordParams.pBuffer = NULL
  
```

Close Channel

The channel resource used for recording should be closed in the following order:

1. Stop recording - ***acStopRecord()***
2. Stop summation - ***acStopChannelAdditionalTimeSlotSummation()***
3. Close third channel - ***acCloseChannel()***

Two Channel Recording

As of release 5.0 users can introduce a two channel recording model when recording a summed conversation. The following limitations do apply:

- The IPM-260 must support a conference bridge, and this feature must be enabled via the purchase of a feature key.
- The application can only rely on two channel resources when recording either into G.711 format or recording with the same format as the received RTP data*. If transcoding is required, then the application must be modeled after three channel method to record a summed conversation.
*Provided recording with this vocoder is permitted by the feature key.
- Firmware 5.00.18.002 or greater is required, or this application design will not work

The following section outlines the required application development when recording a summed conversation using only two IPM-260 channel resources.

NOTE: When running the demonstration application that supports v 5.0, the `NumRecordingApplication` parameter in the `ipRecord.ini` file must be set to two (2) to use this application method. Should encoding be required, then this parameter must be set to three (3) for the three channel method of recording.

Board Initialization & Configuration

To prepare the board for operation in this tapping environment the IPM-260 must be assigned an IP address, and subnet address, plus the TDM options must be set.

1. Initialize the VoP library using the `acinitLib()` function.
2. Initialize all boards in the system. This is a two step process first by invoking **`acGetDefaultBoardParameters()`** to obtain board defaults. Modify the parameters which require configuration and then set these values to the board when using the **`acOpenBoard()`** function. When the board is opened and configuration is complete the number of type of boards detected is returned by the `acinitLib()` function.

The following board parameters should be set:

`m_boardParam.DownloadableFiles.ImageFile` - this parameter points to the board's firmware.

`Im_boardParam.NetworkSettings.BoardIPAddr` - the IP address for this board

`m_boardParam.NetworkSettings.BoardSubNetAddr` - the SubNet address for this board

TDM Bus Settings:

`m_boardParam.TDMBusSettings.TDMBusType` - this example relies on the H.100 bus

`m_boardParam.TDMBusSettings.TDMBusClockSource` - this example relies on the Master setting

m_boardParam.TDMBusSettings.TDMBusSpeed - the fastest bus speed is used in this example

m_boardParam.TDMBusSettings.TDMBusMasterSlaveSelection - user preference, this example relies on Master mode

PSTN Interface Settings:

This use of the IPM260 does not require any PSTN interfaces, therefore any trunk protocols must be disabled.

m_boardParam.TrunkConfig[i].ProtocolType = *acPROTOCOL_TYPE_NONE*

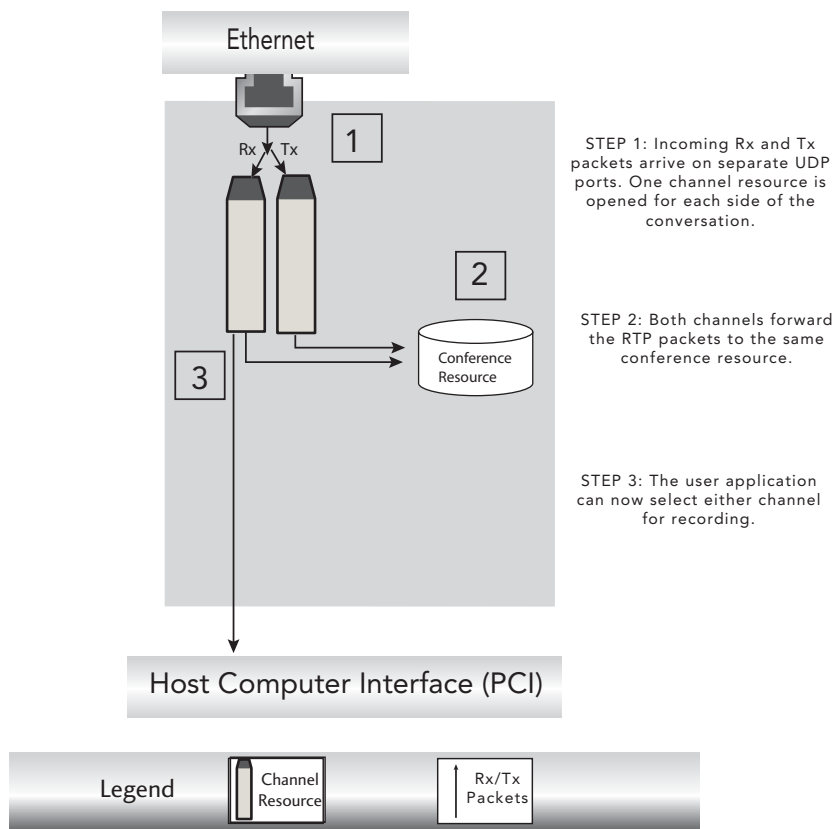
3. Invoke **acGetEvent()**. Wait for an *acEV_BOARD_STARTED* event to indicate the board has started successfully. Any other event indicates some type of board failure.

Channel Initialization & Configuration

Two channel resources are opened to receive the RTP data. The upstream and downstream packets of the conversation are not summed on the IPX. As a result, RTP packets are forwarded as two separate streams to the IPM-260. Two channels are required to receive incoming packets encompassing both sides of a single conversation. These channels are then bridged onto the conference bridge where both sides of the conversation can be summed. Encoding resources are not available therefore, when recording in this scenario, the file can only be recorded in G.711, LinearPCM, G.711MuLaw64 and the coder the channel was opened in.

The following diagram shows an example of how both sides of one conversation are recorded using the IPM-260 when using the conference bridge.

Figure 1-2: Recording while using Conference Resource



This section outlines the required channel configuration on the IPM-260.

Configure Receive Channels

The IPX forwards both streams of the VoIP conversation to two separate channels (each with a single port) on the IPM-260.

On the IPX board, the application provides the IP Address and port number of the recording device. Two unique port numbers must be used - one to receive downstream (Rx) traffic and the other for upstream traffic (Tx). To do this, application developers must first open two receiving channels on the IPM-260, and then invoke an API to obtain the port numbers assigned to each upon initialization. The following section explains this logic and shows the configuration requirements when opening the two recording channels on the IPM-260.

NOTE: Some parameters can be changed “on-the-fly” meaning these values can be modified after the channel has been opened. Parameters which are not labeled as such must be set when the channel is opened.

The following steps are recommendations on how to open and configure two channels for receiving incoming RTP packets on the IPM-260:

1. Open & Configure Channel two channels. This is a two step process where the user invokes **acGetDefaultChannelParameters()** to obtain default settings. The user then modifies applicable parameters and then invokes **acOpenChannel()** to set these values and open the channel. This function returns the channel handle to the user application. The **acTextChannelParam** structure holds all configuration parameters. The following can be used as a guideline for channel configuration:

Voice Settings:

ChannelParam.VoiceSettings.Coder - the CODEC of the incoming packets (this information can be obtained from the IPX when the SmartWORKS EVT_MEDIA_SESSION_BEGIN event has been reported)

ChannelParam.VoiceSettings.ECE - For this passive recording application, set this field to 0 to disable echo cancellation. **NOTE:** This parameter can be changed "on-the-fly".

ChannelParam.VoiceSettings.SCE - set this to **acSilenceCompresionDisable** to disable silence compression

ChannelParam.VoiceSettings.BrokenConnectionEventTimeout - a recommend setting is 300ms. This sets the timer for how long RTP connection should be broken before the board issues a broken connection event.

Transport Settings:

ChannelParam.TransportSettings.DisableSoftIPLoopback - set this to 1 to disable soft loopback

ChannelParam.TransportSettings.UseNIorPCI - the packets are received from the network, therefore this field is set to NI (network interface).

ChannelParam.TransportSettings.UniDirectionalRTP - set the direction of RTP packets to **acRTPRxOnly** since this channel is only used for receiving incoming packets

Jitter Buffer:

ChannelParam.DJBSettings.DJBufMinDelay - the delay in the jitter buffer is set to 150 ms - the board's maximum value. AudioCodes recommends setting the maximum value supported by the current SDK. **NOTE:** This parameter can be changed "on-the-fly".

2. Two channels are used to receive incoming packets for a single conversation - one for each side of the conversation. These two channels output the decoded voice packets (in raw PCM format) to TDM timeslots. The following parameters must be set by the user to connect the output of the channels to the TDM:

ChannelParam.TDMBusSettings.TDMBusInputPort

ChannelParam.TDMBusSettings.TDMBusOutputPort

ChannelParam.TDMBusSettings.TDMBusInputChannel

ChannelParam.TDMBusSettings.TDMBusOutputChannel

3. Once the channels are open and configured, the user must then enable the channels to receive incoming packets. The function **acActivateRTP RTPCChannel()** is used to do this. When this function is invoked, users are required to pass in the IP Address of the device which is

transmitting RTP packets to the IPM-260. In this case it is the active port on AudioCodes' IPX.

In this example, when RTP is activated fax transmission is ignored and therefore the T.38 option is not enabled. **NOTE:** Before closing a channel, RTP must be deactivated using the ***acDeActivateRTP_RTCPChannel()*** function.

4. The user must now invoke the ***acGetRTP_RTCPAddress*** function to obtain the default port assigned to each of the open channels by the IPM-260's resource manager. The IPM-260's resource manager utilizes the following logic when managing UDP ports:

All UDP port numbering begins with port 4000. Other ports are assigned numbers incrementally by 10. Thus, when the first channel is opened it is assigned port 4000, and the second channel opened is then assigned port 4010.

When invoking this API, the channel handle must be passed into the function. The following pointers are returned to the user application:

pLocalRTPAddress - the local RTP IP Address and port

pLocalRTCPAddress - the local RTCP IP Address and port (**NOTE:** the IPX does not support RTCP)

pLocalT38Address - the local T38 IP Address and port (**NOTE:** T38 is not enabled in this example)

The port numbers must be passed back to the IPX so that the media packets are forwarded to the correct location. On the IPX, users invoke ***MTipSetSessionMediaDest()*** to define the destination of the RTP packets associated with a particular Session ID. This API allows users to specifically configure the port numbers the RTP packets are forwarded to on the destination device. Users supply two destination IP Addresses and port numbers - one for the upstream packets (Tx) and the other for the downstream packets (Rx). These values must point to the IP Address of the IPM-260 and the two default ports numbers assigned to the channels by the IPM-260's resource manager.

NOTE: The ***acActivateRTP_RTCPChannel()*** function also allows users to assign a port number to each channel rather than accepting board defaults.

Summing and Recording

On the IPM-260 two channels are opened, RTP is activated, and each channel is configured to pass the decoded voice to a TDM timeslot. The application now needs to obtain a conference resource and then bridge the calls onto the conference bridge. When channels are opened from the NI (network interface) side, then the output of these channels must be disconnected from the TDM bus prior to joining these channels with a conference bridge. The following actions are required:

1. Disconnect the output of the two channels from the TDM timeslot
2. Obtain a conference resource
3. Add both channels as participants to the same conference
4. Begin recording

Obtain Conference:

1. To create a conference on AudioCodes boards, invoke the ***acOpenConference()*** function. This API, when completed, returns the conference handle to the user application. A data structure, *acTConferenceParam* holds parameters allowing the application to control configurable aspects of each conference.

The following setting is needed to disable conference tone generation
ConfParam.SignalGenerationEnable = 0

Break TDM Connections:

2. Break both channels' connection to the boards TDM bus. For this application, ***acBreakTDMConnection()*** is used to break the output of both channels away from the TDM bus

Add Conference Participants:

To join conference participants, invoke the ***acAddConferenceParticipant()*** function once per each channel. This function uses a data structure, *acTConferenceParticipantParam*, to control configurable aspects. The demonstration application does not modify any parameters.

Start Recording

Currently, each channel is can hear both sides of the conversation because they are now connected to a single conference bridge. The application then invokes ***acRecord()*** on either channel one or channel two.

It is recommended that you set the *PlayRecordInterface* parameter of the *acTPlayRecordParam* data structure to *TDM_INTERFACE* as this allows you more recording options. The audio on this channel, though the channel is opened as an NI (network interface) channel, can be treated as if it is coming from the TDM due to the conference bridge connection. This allows you to record to file using the same CODEC selected when the channel was opened, as opposed to obtaining another channel for encoding resources.

The following parameters are used to set the remaining recording parameters in the *acTPlayRecordParam* data structure:

PlayRecordTransferType = *FILE_TRANSFER*
pFileName = the output file name
PlayRecordInterface = *TDM_INTERFACE*
recordParams.Offset = 0
recordParams.pNext = NULL
recordParams.pBuffer = NULL

Close Channel

The channel resource used for recording should be closed in the following order:

1. Stop recording - ***acStopRecord()***
2. Close Conference Bridge - the ***acCloseConference()*** function closes a conference bridge, disconnects all participants and frees all conference resources
3. Close both receiving channels - ***acCloseChannel()***. RTP is deactivated when this function is invoked.

Half-Duplex Recording

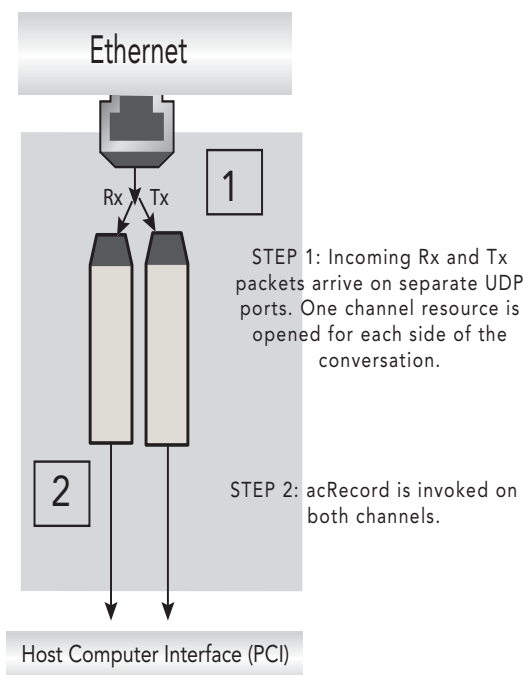
When recording without summation, only two channel resources are required on the IPM-260. This results in two separate files per conversation - upstream (Tx) packets recorded in one, and the downstream (Rx) packets recorded in the other. This recording style can only be used when transcoding is not required. Should transcoding be necessary, then the application should be designed with the [three channel](#) model.

As complete board and channel configuration instructions have been presented in the previous sections, this section only provides a brief look at the design of the application, plus provides an illustration of this application model.

Application Flow

The following steps must be integrated into the design of your application for half-duplex recording:

1. open board
2. get board events
3. open channels (one per each side of conversation) - following configuration instructions provided in the previous examples
4. activate RTP
5. start recording
6. stop recording
7. close channel



Stereo Recording

As of release 5.0, AudioCodes products supports stereo recording. The output from both channel resources can be written to a single file in support of stereo recording.

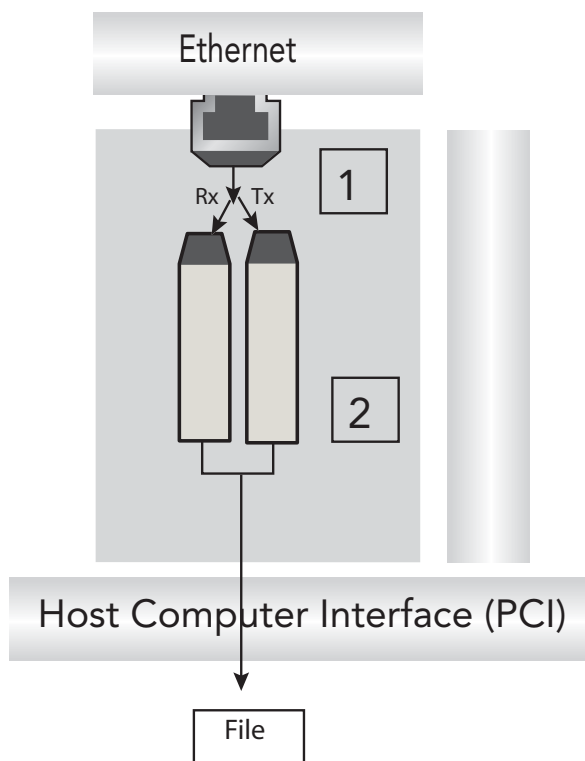
NOTE: This recording model is not part of the demonstration application, but has been included in this documentation.

Complete board and channel configuration instructions have been presented in the previous sections, however this section only provides a brief look at the design of the application, plus provides an illustration of this application model.

Application Flow

The following steps must be integrated into the design of your application for stereo recording:

1. open board
2. get board events
3. open channels (one per each side of conversation) - following configuration instructions provided in the previous examples
4. activate RTP
5. start stereo recording
6. stop recording
7. close channel



STEP 1: Incoming Rx and Tx packets arrive on separate UDP ports. One channel resource is opened for each side of the conversation.

STEP 2: Invoke acStartStereoRecord function to record output of both channels to a single file.

Troubleshooting Tips

When silent recordings are a problem, use the following troubleshooting tips to correct this problem:

1. The link between the IPX and IPM may be down, or the cable may be bad. IPM API and web interface provides the ability to ping an IP Address. Use this feature to ping the IP Address assigned to the forwarding port on the IPX board.
2. The IPM 260 may not be configured properly, or may be down. Verify that the Bootp was successful and that the IPM260 is completely configured with the IP Address (ping this address from the computer hosting the IPX).
3. Validate that the correct port is being used on the IPX and that it has been configured correctly. AudioCodes recommends using port 0 to forward all RTP media from the IPX. This port must be configured with an IP Address on the same network used by the IPM260. It is also important that the passive monitoring ports and the active media forwarding port are configured for different networks to avoid conflicts within the routing table on the IPX.
4. Check the IP addresses. Since IPX has three IP Addresses and should not conflict with the IP Address assigned to the IPM260. For example, if 192.168.0.1 / 0.2 / 0.3 are assigned to the IPX, it is important to verify that the IPM260 was not assigned a similar address by mistake: 192.168.0.1. If there is an IP address conflict, you won't be able to record anything.
5. Verify your tap position. The IPX relies on signaling information to learn of and report Media_Session events to the user application. If the tap is positioned on the network on a spot where only signaling data is present, then no RTP packets can be forwarded from the IPX, though Media session events have been reported.
6. Verify that EV_RTCP is reported by the IPM260 before starting the recording. As soon as the active channel on the IPM260 begins to receive RTP this event is reported. This method can be used to validate that RTP packets are actually received by the channel before the application invokes acRecord.
7. Codec mismatch may cause silence in recordings. When silence is observed, validate that the CODEC of the RTP on the tapped network matches the CODEC set when the channel is opened on the IPM260.
8. Wrong signaling events are used to trigger recordings. Verify the state of the call on the line when silence is noted in the recording. Validate that the call is in an active state vs. an idle or held state. Also – validate that your application logic matches that actual events reported per this particular PBX. Refer to the IPX Integration Guide for call flow examples.
9. Review the network used to transmit RTP packets from the IPX to the IPM260. 10/100 cable and devices (hubs, switches, etc) are recommended over 10T.
10. If running VoPLib 5.0 on the IPM 260:

- The two-channel record does not work with the GA version of firmware. Use firmware version IPM260_UN_F5.00.018.002.cmp (or greater).
- Make sure that the Feature Key of the IPM also supports Conferencing.