HD Acoustic Echo Canceller Software User's Guide

Single Mic, Multi-Mic, and Dual-Mic Noise Reduction Variants

USER's GUIDE

High Definition Acoustic Echo Canceller (HD AEC™)

Table of Contents

1. INTRODUCTION

This document describes HD software acoustic echo canceller. There are three variants of AEC: single microphone (standard), multi-microphone and dual-microphone noise reduction. Single- microphone is used in most applications. Multi-microphone is used in applications such as high-end conference phones that make use of multiple microphones that are placed around conference room table. Multi-microphone noise reduction is used in devices that have a primary microphone closest to the user and a secondary microphone that is used to measure background noise.

1.1 Features

- True full-duplex performance, even when microphone input signal is weak.
- Operates both narrowband and wideband with programmable sampling rate
- Supports true cancellation at tail lengths up to 320 msec
- Non-linear processor
- Fast Convergence and reconvergence
- No divergence due to doubletalk
- Integrated Automatic Gain Control
- Integrated Noise reduction
- Integrated Transmit Equalization
- Instantly adjusts to user-controlled speaker gain changes

1.2 Background on Echo Cancellers

Acoustic echo is caused by direct and indirect feedback from speaker to microphone. Figure 1-1 below shows the acoustic echo path.

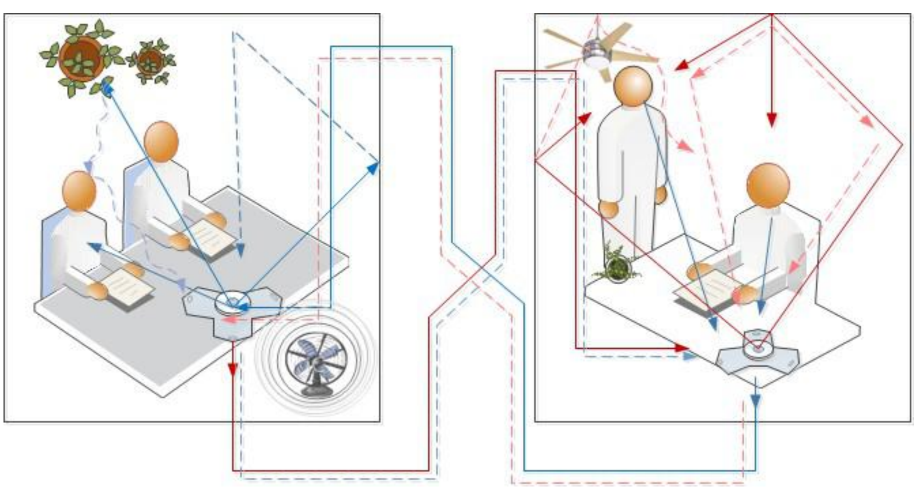

Figure 1-1 - Acoustic Echo

In order to combat the echo phenomenon, an echo canceller is employed. Today's echo cancellers use sophisticated algorithms running on high speed Digital Signal Processors (DSPs) to combat the echo.

Thes acoustic echo canceller, electronically removes both direct coupling and reflected echo, enabling true full-duplex hands-free telephony for both mobile phones and desktop speakerphones. By using acoustic echo canceller to eliminate this unwanted echo and reverberating interference, echo-free conversation can be achieved.

2. Functional Description

Figure 3-1 is a simplified block diagram of the Acoustic Echo Canceller when used in single microphone mode. A description follows. In the description, the parameter names are underlined (i.e. parameter). This indicates that this is a user-controlled parameter which is being described. Detailed descriptions of the user-controlled parameters can be found in the API description found in a later section of this document.

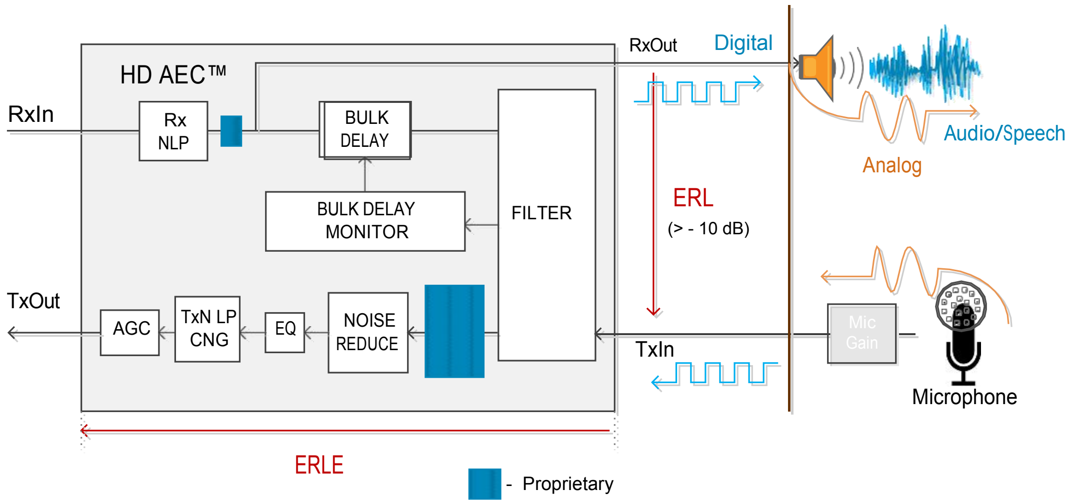

Figure 2-1: Simplified Acoustic Echo Canceller Block Diagram - Single Microphone

The top half of the diagram shows the receive signal path, or the signal path from some network (i.e. telephone, cloud etc.) to the speaker. The bottom half of the diagram shows the transmit signal path from the local microphone toward the aforementioned network. The HD-AEC cancels the echo that occurs between the speaker output and the microphone input.

The RxIn signal coming from the network is fed into the RxNLP (receive Nonlinear Processor). The RxNLP can attenuate (MaxRxAttendB) the received signal by a variable amount based upon the talk state (single talk vs. double-talk). This attenuation improves the overall echo attenuation. The Rx signal processing can be controlled by (rxBypassEnable).

The output of the RxNLP is fed into a noise generator, which adds noise (maxRxNoisedBm) to the receive signal. This noise helps the AEC converge and remain converged even when nobody is speaking. The output of the noise generator is fed both to the transmit output (TxOut) and into the bulk delay block, whose delay is contolled by bulkDelaySamples. The bulk delay block compensates for any non-acoustic buffering delay in the path between RxOut and TxIn. These delays, for example, could be due to hardware design or operating system buffers. The output of the bulk delay is fed to the filter.

The filter estimates the echo and subtracts it from the TxIn signal to form the residual signal. The residual signal is fed to the noise reduction block. This noise reduction block removes background noise and therefore improves the signal to noise ratio of the transmit signal. Noise Reduction parameters include (nrSTIntervalMSec, nrLTIntervalMSec, nrMaxLossdB, nrHighSNRMarkdB, nrLowSNRMarkdB.)

The output of the noise reduction block is fed into an equalizer. The equalizer is used to flatten out the frequency response of the transmit channel. This may be necessary due to the acoustics of the hands-free device and due to the characteristics of the microphone itself. The equalizer gain vs. frequency band table is provided by the user parameter (pTxEqualizerdB10).

The output of the transmit equalizer is fed into the TxNLP. The TxNLP is the transmit non-linear processor. The TxNLP increases the echo attenuation by attenuating the residual by a variable amount based upon the talk state. The TxNLP is controlled by (maxTxLossB and targetResidualLeveldBm, worstExpectedERLdB, and maxTxNLPThresholddB). The TxNLP block also includes a comfort noise generator.

The compute gain block computes the AGC gain. The output of the TxNLP is fed into the AGC gain block, which provides gain or loss depending upon the residual signal level. The AGC is controlled by (agcEnable, agcMaxGaindB, agcMaxLossdB, agcTargetLeveldBm, and agcLowSigThreshdBm.)

The output of the AGC is fed to the TxOut output of the AEC.

In the multi-microphone case, there is still a single receive path but there is one transmit path per microphone.

In the case of multi-microphone noise reduction, there is a single receive path, a complete transmit path for the primary microphone, and a partial transmit path for the secondary microphone. In this case, there are two transmit inputs (one for each microphone) but only one transmit output containing the echo cancelled and noise reduced signal.

3. Echo Canceller API

3.1 Parameters

The AEC initialization and control parameters are summarized in table 4-1 below. Detailed descriptions of the parameter usage follow.

Param Name	Param Type	Init/Control/Status (I,C,S)	Recommended Value
lockCallback	LockCallback_t	I	Pointer to user-supplied software lock handling function.
frameSize	XDAS_Int16	I	App Dependent
antiHowlEnable	XDAS_Int16	I	App Dependent
samplingRate	XDAS_Int32	I	App Dependent
maxAudioFreq	XDAS_Int32	I	samplingRate/2
fixedBulkDelayMSec	XDAS_Int16	I	App Dependent
variableBulkDelayMSec	XDAS_Int16	I	App Dependent (Beta version under development)
initialBulkDelayMSec	XDAS_Int16	I	Reserved – must be set to 0
activeTailLengthMSec	XDAS_Int16	I	App Dependent Must be multiple of 8 msec
totalTailLengthMSec	XDAS_Int16	I	App Dependent same as activeTailLengthMSec
txNLPAggressiveness	XDAS_Int16	I	Deskphone: 1 Mobile Handsfree: 10 Mobile Handset: 1
maxTxLossSTdB	XDAS_Int16	I	20 dB (handset) 40 dB (hands-free)
maxTxLossDTdB	XDAS_Int16	I	10 dB (handset) 20 dB (hands-free)
maxRxLossdB	XDAS_Int16	I	12 dB
initialRxOutAttendB	XDAS_Int16	I	Reserved – must be set to 0
targetResidualLeveldBm	XDAS_Int16	I	-85 dBm
maxRxNoiseLeveldBm	XDAS_Int16	I	-90 dBm
worstExpectedERLdB	XDAS_Int16	I	App Dependent. But should include margin.
rxSaturateLeveldBm	XDAS_Int16	I	App Dependent
noiseReduction1Setting	XDAS_Int16	I	1
noiseReduction2Setting	XDAS_Int16	I	0
cngEnable	XDAS_Int16	I	1
fixedGaindB10	XDAS_Int16	I	0
txAGCEnable	XDAS_Int8	I	App Dependent
txAGCMaxGaindB	XDAS_Int8	I	10 dB
txAGCMaxLossdB	XDAS_Int8	I	6 dB
txAGCTargetLeveldBm	XDAS_Int8	I	-10 dBm
txAGCLowSigThreshdBm	XDAS_Int8	I	-45 dBm

rxAGCEnable	XDAS_Int8	I	App Dependent
rxAGCMaxGaindB	XDAS_Int8	I	10 dB
rxAGCMaxLossdB	XDAS_Int8	I	15 dB
rxAGCTargetLeveldBm	XDAS_Int8	I	-10 dBm
rxAGCLowSigThreshdBm	XDAS_Int8	I	-40 dBm
rxBypassEnable	XDAS_Int8	I	1
maxTrainingTimeMSec	XDAS_Int16	I	0
TrainingRxNoiseLeveldBm	XDAS_Int16	I	0
*pTxEqualizerdB10	XDAS_Int16 *	I	Reserved
mipsMemReductionSetting	XDAS_Int8	I	0
mipsReductionSetting2	XDAS_Int8	I	0
txrxMode	XDAS_Int8	I	Reserved – must be set to 0

Table 3-1 – Parameter Summary

3.1.1 lockCallback

This parameter should point to a user supplied callback function that handles software locking. The function prototype for this function is defined in voipengine_user_v4.h. This function is described in a later section of this document titled Lock Callback Function.

3.1.2 frameSize

The number of PCM samples contained in the input and output buffers (TxIn, TxOut, RxIn, RxOut).

3.1.3 antiHowlEnable

This flag, when enabled, causes the AEC to take measures to reduce the chance that howling (feedback) will occur. Howling can occur when there is a full-duplex communication link that has echo at both ends. Howling tends to occur when there is net gain when adding the gain of both echo sources. The total gain does not have to be positive. It is sufficient to have gain at only a single frequency.

3.1.4 samplingRate

The samplingRate parameter tells the AEC what the input and output sampling rates are. Sampling rate is specified in units of Hz.

3.1.5 maxAudioFreq

The maxAudioFreq, specified in Hz, is the highest frequency component in the audio signal. In most applications, a bandpass filter is used to prevent aliasing due to the sampling process. The theoretical maximum audio frequency is equal to half the sampling rate, but with a bandpass filter in place, the maximum audio frequency will be somewhat less. By setting maxAudioFrequency less than half the sampling frequency, CPU utilization (MIPS) can be reduced.

3.1.6 fixedBulkDelayMSec

The fixedBulkDelayMSec parameter, specified in milliseconds, controls amount of bulk delay that is inserted into the receive path. The reason for the bulk delay is to compensate for non-acoustic delays in the path between the AEC RxOut (speaker) interface and the TxIn (microphone interface). Non-acoustic delays are typically the buffering delay caused by double-buffering of the speaker output and microphone input ports. The buffering delay is therefore typically equal to twice the frame size.

Setting the bulk delay properly is very important. Setting too short a bulk delay will cause the echo canceller's effective tail length to be shorter. Setting the bulk delay too long will result in the AEC's perception of a non-causal echo, which can not be cancelled. In some situations, where the operating system, for example Android, can produce varying buffer delays, it is necessary to employ bulk delay search algorithms which would operate periodically and update the bulk delay when necessary.

3.1.7 variableBulkDelayMSec (optional feature)

This parameter sets the maximum bulk delay that the bulk delay finder will search. For example, if it is set to 1000 (specified in milliseconds) then the search range will be to 1 sec. If the actual bulk delay is longer than the set parameter, the bulk delay finder will not find the correct bulk delay.

Setting this parameter to zero will disable the bulk delay finder.

Note that to enable bulk delay finder, variableBulkDelayMSec should be set to larger than activeTailLengthMSec.

This optional feature is available only in accordance with licensing agreement.

3.1.8 initialBulkDelayMSec (Obsolete)

3.1.9 activeTailLengthMSec and totalTailLengthMSec

The activeTailLengthMSec parameter defines the tail length, in milliseconds, that the AEC cancels using the filter. The active tail length should be set according to the expected room acoustics. The acoustic echo attenuation in an acoustic environment tends to increase as the delay increases. When the attenuation reaches a reasonable level, cancellation need no longer be applied. Suppression (nonlinear processing) can be applied instead. The activeTailLength should therefore be set at the delay that corresponds to a reasonable acoustic attenuation level – perhaps 30 dB.

totalTailLengthMSec should be set same as activeTailLengthMSec.

Note that activeTailLengthMSec/totalTailLengthMSec should be a multiple of 8 msec.

3.1.10 maxTxLossSTdB, maxTxLossDTdB, txNLPAggressiveness, and targetResidualLeveldBm

The transmit nonlinear processor (TxNLP) suppresses residual echo that is not completely cancelled by the filter. The TxNLP suppresses primarily during periods of single talk in order to maintain a full- duplex sound to the voice conversation.

The TxNLP can be made more or less aggressive by using the maxTxNLPAggressiveness setting. The default setting is zero. The more positive the setting, the more aggressive the NLP will be. The more negative the setting, the less aggressive the NLP will be. The range is -40..40.

The maxTxLossSTdB and maxTxLossdBdB parameter defines the maximum attenuation (loss), expressed in dB, that the TxNLP will apply to the transmit signal during single-talk (ST) and double-talk (DT) conditions respectively.

During single-talk conditions (Rx-Only speech), the TxNLP will try to drive the residual signal down toward the level specified by targetResidualLeveldBm.

3.1.11 maxRxLossdB and targetResidualLeveldBm

The receive nonlinear processor (RxNLP) attenuates the receive signal to improve echo attenuation during double-talk periods. This parameter defines the maximum attenuation, expressed in dB, that the RxNLP will apply to the receive signal. Since this RxNLP engages during double-talk, an overly aggressive maxRxLossdB setting will cause a half-duplex sounding conversation.

3.1.12 initialRxOutAttendB (Obsolete)

Under some harsh acoustic conditions in which the speaker to microphone gain is excessive, it is helpful to apply attenuation to the receive output (speaker output) signal. This parameter allows you to set the initial amount of receive output attenuation that will be applied. This helps remove artifacts that would otherwise be present at the start of a phone call. Once the AEC determines the actual acoustic conditions, it will modify the rx output attenuation accordingly.

As a rule of thumb, if the speaker to microhone gain is less than 10 dB, you should leave this setting at zero.

3.1.13 maxRxNoiseLeveldBm

The AEC can optionally generate low level noise for transmission out to the speaker. The purpose of this noise is to allow the AEC to converge and reconverge its filter even when there is no receive signal present. Without this background noise, any changes in the acoustic echo path that occur during receive signal silence would not be “seen” by the AEC until the receive signal returns. Using the noise, the AEC can be ready for the next utterance of speech by being converged already.

The maxRxNoiseLeveldBm specifies the maximum added noise level in dBm.

3.1.14 worstExpectedERLdB

This parameter defines the worst expected ERL (echo return loss) between the speaker and microphone. A typical hands-free phone may have a loss of zero while a handset may have a loss of 20-30 dB.

3.1.15 rxSaturateLeveldBm

Echo the cancellation portion of echo cancellers do not handle nonlinearities in the speaker to microphone path very well. This parameter allows the user to specify the receive signal above which the speaker to microphone path may exhibit significant nonlinearity. Note that our definition of nonlinearity for this purpose includes not only saturation in analog circuitry but also mechanical vibration.

3.1.16 noiseReduction1Setting

This controls if and how standard (lower complexity) noise reduction is configured.

0: Disabled
1-30: Degree of noise reduction increases as the setting increases.

For hands-free applications the noise reduction feature is **STRONGLY** recommended. Disabling this feature affects full-duplex performance in hands-free environments.

3.1.17 noiseReduction2Setting

This controls if and how the optional higher-complexity noise reduction is configured.

0: Disabled
1-30: Degree of noise reduction increases as the setting increases.

3.1.18 cngEnable

This flag enables the comfort noise generator in the transmit direction. For hands-free applications the comfort noise generator feature is **STRONGLY** recommended.

3.1.19 fixedGaindB10

The AEC can optionally apply a fixed gain to the transmit path. The amount of this gain is controlled by the parameter fixedGaindB10, specified in tenths of a dB. If fixed gain is to be applied in the transmit path, it is preferable to do so inside the echo canceller rather than outside the echo canceller in order not to interfere with echo canceller, AGC, and noise reduction performance.

3.1.20 txAGCEnable and rxAGCEnable

This flag, when set, enables the AGC to operate in the transmit path (txAGCEnable) or receive path (rxAGCEnable).

3.1.21 tx/rxAGCMaxGaindB, tx/rxAGCMaxLossdB, tx/rxAGCTargetLeveldBm, and tx/rxAGCLowSigThreshdBm

These parameters control the AGC in the tx and rx paths respectively. For readability, the tx and rx are omitted in the description.

When the AGC is enabled, these parameters control the operation of the AGC. When the signal level is below agcLowSigThreshdBm (specified in dBm), the AGC does not change the transmit signal.

When the signal level is above agcLowSigThreshdBm but below agcTargetLeveldBm, the AGC applies gain to try to reach an output level of agcTargetLeveldBm. But the AGC will not apply more than agcMaxGaindB dB of gain.

When the signal is above agcTargetLeveldBm, the AGC applies attenuation to try to reach an output level of agcTargetLeveldBm. But the AGC will not apply more than agcMaxLossdB of attenuation.

3.1.22 rxBypassEnable

When set to 1, rxBypassEnable causes all receive path signal processing to be disabled. rxBypassEnable is typically suggested to set to 1 in most of user cases.

3.1.23 maxTrainingTimeMSec and trainingRxNoiseLeveldBm

The user may optionally instruct the echo canceller to train the AEC at the beginning of a hands-free session. The AEC does this by playing out a short low-level training signal to the speaker, analyzing the microphone input, and modeling the acoustic echo path. The maxTrainingTimeMSec parameter specifies the maximum duration of the training signal. To disable training, set this parameter to zero. The trainingRxNoiseLeveldBm parameter specifies the level of the noise to be used during training, in dBm.

3.1.24 pTxEqualizerdB10 (Obsolete)

The pTxEqualizerdB10 is a pointer to an array whose elements contain equalizer gain (or loss) as a function of frequency. The elements are specified in units of tenths of a dB. The frequency spectrum is divided in 32 sections.

So, for a sampling rate of 8000, the first element of the pTxEqualizer array controls the gain or loss within the 0 – 250 Hz frequency band. (250 = 8000/32). A setting of 30, for example will cause a 3 dB (30 tenths) gain to be applied to the frequency range. A setting of –30 will cause a 3 dB attenuation to be applied to the frequency range. The second element of the array will control the gain or loss applied to the 250 – 500 Hz frequency band and so on.

Passing a null pointer will disable the equalizer.

3.1.25 mipsMemReductionSetting

The mipsMemReductionSetting reduces the MIPS and memory utilization of the AEC. The value ranges between 0 and 4. A value of 0 results in no MIPS or memory reduction, and a value of 4 results in the maximum amount of MIPS and Memory reduction.

With the reduction in MIPS and memory comes a change in performance. In particular, the initial convergence and subsequent reconvergences will be slowed down.

3.1.26 mipsReductionSetting2

The mipsReductionSetting2 reduces MIPS that are utilized in the background. Values range from 0 to 1 with 0 being no reduction and 1 being the maximum reduction. A higher setting can cause slower initial convergence and subsequent reconvergence.

3.2 Software API

The software API is loosely based upon the xDAIS standard.

3.3 Standard DAIS Functions

The standard DAIS functions are listed in the sections that follow. Since they are standard functions, there is little need for additional information. The user is referred to our sample calling code as well as to the TI xDAIS documentation for further information.

3.3.1 AECG4_RK_activate

Implemented but does nothing.

3.3.2 AECG4_RK_numAlloc

Implemented.

3.3.3 AECG4_RK_alloc

Implemented.

3.3.4 AECG4_RK_control

Implemented.

3.3.5 AECG4_RK_deactivate

Implemented but does nothing.

3.3.6 AECG4_RK_exit

Implemented but does nothing.

3.3.7 AECG4_RK_free

Implemented but does nothing.

3.3.8 AECG4_RK_init

Implemented but does nothing.

3.3.9 AECG4_RK_initObj

Implemented. See section 3-1 for parameter descriptions.

3.3.10 AECG4_RK_moved

Implemented but does nothing.

3.4 Application Specific Functions
3.4.1 Single Channel APIs

3.4.1.1 AECG4_RK_create

Prototype:

```
AECG4_Handle AECG4_RK_create(
      const IAECG4_Fxns *fxns,
      const IAECG4_Params *prms)
```

Inputs:

fxns – the IALG function table (should be set to 0)
prms – Initialization paramters

Returns:

Handle to an AEC instance

Description:

AECG4_RK_create creates an instance of the AEC using a dynamic memory allocation model. When the instance is no longer needed, AECG4_delete should be called in order to free up any allocated memory.

3.4.1.2 AECG4_RK_createStatic

Prototype:

```
AECG4_Handle
      AECG4_RK_createStatic( IAECG4_
      Fxns *fxns, IAECG4_Params
      *params, IALG_MemRec *memTab)
```

Inputs:

fxns – the IALG function table (NULL is recommended as the default)
prms – Initialization paramters
memTab – xDAIS memory allocation table

Returns:

Handle to an AEC instance

Description:

AECG4_RK_createsStatic creates an instance of the AEC using static memory allocation. The application must determine the memory requirements, allocate the required memory sections, and pass the memTab poitner to this function. The memory requirements are a function of some of the initialization parameters. In order to determine the memory requirements, a helper function – AECG4_RK_staticAllocHelper – is provided.

3.4.1.3 AECG4_RK_staticAllocHelper

Prototype:

```
void
AECG4_RK_staticAllocHelper( const
IAECG4_Params *prms)
```

Inputs:

prms – Initialization paramters

Description:

AECG4_RK_staticAllocHelper is a helper function designed to determine AEC memory requirements as a function of initialization parameters. This function is meant to operate only in "debug" mode. It prints the memTab structure to the console using printf. The memTab structure can subsequently be used by AECG4_RK_createStatic.

3.4.1.4 AECG4_RK_alloc

Prototype:

```
void AECG4_RK_alloc(const AECG4_Params *prms,
      struct IALG_Fxns ** fxns,
      IALG_MemRec memTab[])
```

Inputs:

prms: initialization parameters

fxns: pointer to location where AECG4_RK_alloc will place a pointer AECG4's IALG function pointer table. fxns may be set to zero if the function pointer table isn't needed.

Outputs:

memTab – xDAIS memory allocation table. Note that memTab must be dimensioned with size MTAB_NRECS

*fxns will return a pointer to AECG4's IALG function pointer table.

AECG4_RK_alloc returns an xDAIS memory allocation table based upon the provided parameters.

3.4.1.5 AECG4_RK_control

Prototype:

```
Int
      AECG4_RK_control( IA
      LG_Handle handle,
      IALG_Cmd cmd,
      IALG_Status *status)
```

Inputs:

cmd – a command that tells AEC_G4_control what to do

C adaptation

```
IAECG4_RESUME – resumes the AEC adaptation
IAECG4_SETSTATUS – apply a control to the AEC
IAECG4_GET_TIME_DOMEAIN_ECHO_MODEL – get estimated echo
path impulse response status (items in table below listed
as "control")
IAECG4_NOTUSE_TAILSEARCH – Do not use the bulk delay
finder's result to adjust.
IAECG4_TUSE_TAILSEARCH – Allow AEC to use the bulk delay
finder's result to adjust internal bulk delay.
IAECG4_DISABLE_TXAGC_RUN – Disable TxAGC during runtime.
IAECG4_ENABLE_TXAGC_RUN – Enable TxAGC during runtime.
IAECG4_DISABLE_RXAGC_RUN – Disable RxAGC during runtime.
IAECG4_ENABLE_RXAGC_RUN – Enable RxAGC during runtime.
```

Outputs:

status (see table below)

Description:

AECG4_RK_control performs the requested command as specified via the cmd parameter.

Status / Control Item	Description	Status / Control
pReturnedModel	Pointer to impulse response of echo path to be returned when control function is called with IAECG4_GET_TIME_DOMAIN_ECHO_MODEL	Control
nCoefToReturn	Number of impulse response samples to return when control function is called with IAECG4_GET_TIME_DOMAIN_ECHO_MODEL	Control
txInPowerdBm10	Transmit input power specified in tenths of dBm	Status
txOutPowerdBm10	Transmit output power specified in tenths of dBm	Status
rxInPowerdBm10	Receive input power specified in tenths of dBm	Status
rxOutPowerdBm10	Receive input power specified in tenths of dBm	Status
residualPowerdBm10	Residual power specified in tenths of dBm	Status
erlDirectdB10	Echo Return Loss measured based upon signals, specified in tenths of dB	Status
erlIndirectdB10	Echo Return Loss measured based upon filter, specified in tenths of dB	Status
erldB10BestEstimate	The best current ERL estimate – averaged across the whole audio bandwidth, expressed in units of tenths of a dB	Status
worstPerBinERLdB10BestEstimate	Worst (lowest numerically) ERL estimate across the entire frequency band, expressed in units of tenths of a dB	Status
worstPerBinERLdB10BestEstimateConfidence	On a scale from 0 to 100, the level of confidence that the AEC has in the worstPerBinERLdB10BestEstimate statistic	Status
erledB10	Echo Return Loss Enhancement (pre-NLP), specified in tenths of dB	Status
shortTermERLEdB10	Echo Return Loss Enhancement (pre-NLP), specified in tenths of dB	Status
instantaneousERLEdB100	Instantaneous ERLE, in units of hundredths of a dB	Status
dynamicNLPAggressivenessAdjustdB10	Current added degree of dynamic NLP aggressiveness, in units of tenths of a dB	Status
shadowERLEdB10 (Not used)	Echo Return Loss Enhancement based upon shadow filter residual output), specified in tenths of dB	Status
rxVADState	Receive VAD state 0: silence or noise 1: speech 2: speech hangover	Status
txVADState	Transmit VAD state 0: silence or noise 1: speech 2: speech hangover	Status
rxVADStateLatched	Latched copy of receive VAD State	Status
currentBulkDelaySamples	The current value of bulk delay, measured in samples When the bulk delay finder is enabled, this status will return current bulk delay used by AEC.	Status
estimatedBulkDelaySamples	The value of bulk delay computed by bulk delay finder, measured in samples When the bulk delay finder is enabled, this status will return finder's result, which is not adjusted yet by AEC.	Status
txAttenuationdB10	Transmit NLP attenuation, measured in tenths of a dB	Status
rxAttenuationdB10	Receive NLP attenuation, measured in tenths of a dB	Status
rxOutAttenuationdB10	The current receive output attenuation, measured in tenths of a dB	Status
nlpThresholddB10	Current NLP threshold, measured in tenths of a dB	Status
nlpSaturateFlag	If set to 1, a saturation has been detected in the NLP	Status
aecState	Reserved	Status

sbcngResidualPowerdBm10	Subband CNG residual power – expressed in units of tenths of a dB	Status
sbCNGPowerdBm10	Comfort Noise Generator signal power, in units of tenths of a dBm	Status
rxOutAttendB	Receive out attenuation, expressed in units of dB	Status
sbMaxAttendB10	Maximum transmit subband attenuation, expressed in units of tenths of dB	Status
sbMaxClipLeveldBm10	Maximum transmit clip level, expressed in units of tenths of dB	Status
sbInitFlags	Word of flags indicating whether or not each frequency bin is in the initialized state (Somewhat converged)	Status
txFreqOffsetHz	Transmit frequency offset for anti-howling, in Hz	Status
rxFreqOffsetHz	Receive frequency offset for anti-howling, in Hz	Status
sbTxVRKotalExceeddB10	Indication of transmit voice activity. Zero indicates no voice activity. As this statistic increases, the probability that there is voice activity increases	Status
speakerLevelChangeDeltadB (obsolete)	Change in analog speaker gain, specified in dB. This is used to inform that AEC of a user-controlled change in analog speaker gain. A value of 127 indicates that the level has changed by an unknown amount.	Control

Status Structure

It should be noted that many of these status variables are advanced and primarily for the use of when doing troubleshooting. For this reason, more detailed explanations are omitted.

3.4.1.6 AECG4_RK_apply

AECG4_RK_apply performs the echo cancellation. It operates on both the transmit and receive speech buffers. (If the transmit and receive are processed in separate functions, use AECG4_RK_applyTx and AECG4_RK_applyRx instead.)

Prototype:

```
XDAS_Void
      AECG4_RK_apply( IAECG4
      _Handle handle,
      XDAS_Int16 * ptrRxIn,
      XDAS_Int16 * ptrRxOut,
      XDAS_Int16 * ptrTxIn,
      XDAS_Int16 * ptrTxOut)
```

Inputs:

handle – handle to the xDAIS object
ptrRxIn – pointer to receive input buffer, data: 16-bit PCM, length: frame size
ptrTxIn – pointer to transmit input buffer, data: 16 bit PCM, length: frame size

Outputs:

pRxOut – pointer to receive output buffer, data: 16 bit PCM, length: frame size
pTxOut – pointer to transmit output buffer, 16 bit PCM, length: frame size

Description:

Use the apply function must be called once per frame.

3.4.1.7 AECG4_RK_applyTx

AECG4_RK_applyTx performs the transmit half of the echo cancellation.

Prototype:

```
XDAS_Void
        AECG4_RK_applyTx( IAEC
        G4_Handle handle,
        XDAS_Int16 * ptrTxIn,
        XDAS_Int16 * ptrTxOut)
```

Inputs:

handle – handle to the xDAIS object
ptrTxIn – pointer to transmit input buffer, data: 16 bit PCM, length: frame size

Outputs:

pTxOut – pointer to transmit output buffer, 16 bit PCM, length: frame size

Use the applyTx, used in conjunction with the applyRx function must be called once per frame.

3.4.1.8 AECG4_RK_applyRx

AECG4_RK_applyRx performs the receive half of the echo cancellation.

Prototype:

```
XDAS_Void
        AECG4_RK_applyRx( IAEC
        G4_Handle handle,
        XDAS_Int16 * ptrTxIn,
        XDAS_Int16 * ptrTxOut)
```

Inputs:

handle – handle to the xDAIS object
ptrRxIn – pointer to transmit input buffer, data: 16 bit PCM, length: frame size

Outputs:

pRxOut – pointer to transmit output buffer, 16 bit PCM, length: frame size

Description:

Use the applyRx, used in conjunction with the applyTx function must be called once per frame.

3.4.1.9 AECG4_RK_backgroundHandler (obsolete since Version 5)

Prototype:

```
XDAS_Void
        AECG4_RK_backgroundHandler( IAEC
        G4_Handle handle)
```

Input:

handle – handle to the xDAIS object

Description:
In case the legacy AECG4 is used (before version 4.0) this function must be called as often as possible from a lower priority thread or idle task. If backgroundHandler is starved, the AEC will not converge or reconverge.

Note: After AECG4 version 4.0, when the aecMode set to 1, there is no need to call this function any more.

3.4.1.10 AECG4_RK_reset

Prototype:

```
XDAS_Int32 AECG4_RK_reset(
IAECG4_Handle handle, const IAECG4_Params *prms)
```

Inputs:

Handle – handle to the xDAIS object
Prms – AEC parameters structure

Use: This function is used to reset the state of the echo canceller without allocating memory. This function is intended to be used after an instance has already been created.

3.4.1.11 AECG4_RK_saveRestoreState (Obsolete)

Prototype:

```
XDAS_Int32
    AECG4_RK_saveRestoreState( IAECG
    4_Handle handle,
    XDAS_Int8 *pState,
    XDAS_Int32 Length,
    XDAS_Int8 Action)
```

Inputs:

Handle – handle to the xDAIS object
Action –
SAVE_RESTORE_ACTION_GET_LENGTH causes AECG4_saveRestoreState to return the length of the state information. This length, in bytes, can be used by the host application to allocate a buffer into which to store the state information.
SAVE_RESTORE_ACTION_SAVE causes AECG4_saveRestoreState to save the current state into the buffer pointed to by pState
SAVE_RESTORE_ACTION_RESTORE causes AECG4_saveRestoreState to save the restore the state of the echo canceller using the state information that is stored in the buffer pointed to by pState.
PState = pointer to state state information to be saved or restored

Outputs:

pState – if a save operation is executed, pState will contain the saved state.

Returns: the size of the state information

Description: Since it takes a finite amount of time for the AEC to converge after reset, it is useful in many applications to save a known converged state for use in subsequent phone calls. For example, the state information may be saved at power up after initial training. That state can be loaded into the echo canceller at the start of each phone call. Alternately, the state information can be saved at the end of each phone call, to be used for the subsequent phone call.

Furthermore, the different states can be maintained for hands-free and handset operation.

3.4.2 Multi-Microphone and Dual-Microphone Noise Reduction API Functions

! NOTE: In accordance with your licensing agreement, your product may be restricted for Single Mic Use only.

The multi-microphone and dual-microphone API functions are similar to the single-microphone API functions with a few notable exceptions:

- The classic DAIS functions are not completely extended to the user-level API. Only higher level “concrete” functions are available.
- Dynamic creation is the only supported create method. Static creation is not supported.
- Separate tx and rx apply functions are not supplied. Only a single combined apply function is supported.

The multi-microphone and dual-microphone variants share the same API functions with the execption of the “create” API, where they each have their own.

3.4.2.1 AECG4_RK_createMMIC

Prototype:

```
IMMICAECG4_Handle
        AECG4_RK_createMMIC( const
        IAECG4_Fxns *fxns, const
        IAECG4_Params *prms,
        const unsigned char NMicrophones,
        XDAS_UInt8 *pMicGroups);
```

Inputs:

fxns – the IALG function table (should be set to 0)
prms – Initialization paramters
pMicGroups – Reserved, must be set to 0.

Returns:

Handle to an multi-mic AEC object

Description:
AECG4_RK_createMMIC creates an instance of the multi-mic AEC using a dynamic memory allocation model. When the instance is no longer needed, AECG4_deleteMMIC should be called in order to free up any allocated memory.

3.4.2.2 AECG4_RK_createDMNR

Prototype:

```
IMMICAECG4_Handle
        AECG4_RK_createDMNR( const
        IAECG4_Fxns *fxns, const
        IAECG4_Params *prms
        );
```

Inputs:

fxns – the IALG function table (should be set to 0)
prms – Initialization paramters

Returns:

Handle to an multi-mic AEC object

Description:
AECG4_RK_createDMNR creates an instance of the dual-mic noise reduction AEC using a dynamic memory allocation model. When the instance is no longer needed, AECG4_deleteMMIC should be called in order to free up any allocated memory.

3.4.2.3 AECG4_RK_applyMMIC

Prototype:

```
XDAS_Void
        AECG4_RK_applyMMIC( IMMICA
        ECG4_Handle handle,
        XDAS_Int16 * ptrRxIn,
        XDAS_Int16 * ptrRxOut,
        XDAS_Int16 * ptrTxIn[],
        XDAS_Int16 * ptrTxOut[]);
```

Inputs:

Handle – handle to the multi-mic AEC object
ptrRxIn – pointer to receive input buffer : 16-bit PCM, length: frame size
ptrRxOut – pointer to the receive output buffer : 16-bit PCM, length: frame size
ptrTxIn[ ] – array of pointers – each pointing to a single microphone's transmit input buffer : 16-bit PCM, length: frame size
ptrTxOut[ ] – array of pointers – each pointing to a single microphone's transmit input buffer : 16-bit PCM, length: frame size

Outputs:

ptrRxOut – receive ouput signal is placed here: 16-bit PCM, length: frame size
ptrTxOut[] – array of pointers – each pointing to a single microphone's transmit output buffer: 16-bit PCM, length frame size

Returns:

Use: Call the applyMMIC function once per frame.

3.4.2.4 AECG4_RK_backgroundHandlerMMIC (obsolete)

Prototype:

```
XDAS_Void
AECG4_RK_backgroundHandlerMMIC( IMMICAECG4
_Handle handle);
```

Inputs:

Handle – handle to the multi-mic AEC object

Outputs:

Returns:

Use:

In case the legacy AECG4 is used (before version 4.0) Your application must call this function as often as possible in a lower priority thread or in an idle thread.

After AECG4 version 4.0, when the aecMode set to 1, there is no need to call this function any more.

3.4.2.5 AECG4_RK_saveRestoreStateMMIC (obsolete)

Prototype:

```
XDAS_Int32
      AECG4_RK_saveRestoreStateMMIC( IMMIC
      AECG4_Handle handle,
      XDAS_Int8 *pState,
      XDAS_Int32 Length,
      XDAS_Int8 Action);
```

Inputs:

Handle – handle to the multi-mic AEC object

Action –

SAVE_RESTORE_ACTION_GET_LENGTH causes AECG4_saveRestoreStateMMICto return the length of the state information. This length, in bytes, can be used by the host application to allocate a buffer into which to store the state information.

SAVE_RESTORE_ACTION_SAVE causes AECG4_saveRestoreStateMMIC to save the current state into the buffer pointed to by pState

SAVE_RESTORE_ACTION_RESTORE causes AECG4_saveRestoreStateMMIC to save the restore the state of the echo canceller using the state information that is stored in the buffer pointed to by pState.

PState = pointer to state state information to be saved or restored

Outputs:

pState – if a save operation is executed, pState will contain the saved state.

Returns: the size of the state information

Use: Since it takes a finite amount of time for the AEC to converge after reset, it is useful in many applications to save a known converged state for use in subsequent phone calls. For example, the state information may be saved at power up after initial training. That state can be loaded into the echo canceller at the start of each phone call. Alternately, the state information can be saved at the end of each phone call, to be used for the subsequent phone call.

3.4.2.6 AECG4_RK_resetMMIC

Prototype:
Int
AECG4_RK_resetMMIC(IMMICAECG4_Handl
e handle,
const IAECG4_Params *iAECG4Params);

Inputs:

Handle – handle to the multi-mic AEC object
Prms – AEC parameters structure

Use: This function is used to reset the state of the echo canceller without allocating memory. This function is intended to be used after an instance has already been created.

3.4.2.7 AECG4_RK_controlMMIC

Prototype:
XDAS_Void
AECG4_RK_controlMMIC(IMMICAECG
4_Handle handle, IALG_Cmd cmd,
XDAS_Int32 MicNumber,
IAECG4_Status * status);

Inputs:
Handle – handle to the multi-mic handle cmd – control command

IAECG4_GETSTATUS – returns status
IAECG4_PAUSE – pauses the AEC adaptation
IAECG4_RESUME – resumes the AEC adaptation
IAECG4_SETSTATUS – apply a control to the AEC
IAECG4_GET_TIME_DOMEAIN_ECHO_MODEL – get estimated echo path impulse response
IAECG4_NOTUSE_TAILSEARCH – Do not use the bulk delay finder's result to adjust.
IAECG4_TUSE_TAILSEARCH – Allow AEC to use the bulk delay finder's result to adjust internal bulk delay.
IAECG4_DISABLE_TXAGC_RUN – Disable TxAGC during runtime.
IAECG4_ENABLE_TXAGC_RUN – Enable TxAGC during runtime.

IAECG4_DISABLE_RXAGC_RUN – Disable RxAGC during runtime.
IAECG4_ENABLE_RXAGC_RUN – Enable RxAGC during runtime.

MicNumber – Selects which mic to apply the control function to
status – pointer to returned status information (if cmd is IAECG4_GETSTATUS) or pointer to status to be set (if cmd is IAECG4_SETSTATUS)

Outputs:
status – returned status if cmd is IAECG4_GETSTATUS

Returns:

3.4.2.8 AECG4_RK_enableMic

Prototype:
XDAS_Int32
AECG4_RK_enableMic(IMMICAECG4_Handle
mmhandle, XDAS_Int8 MicIndex);

Inputs:
Handle – handle to the multi-mic handle

Outputs:

Returns:

Use:

3.4.2.9 AECG4_RK_disableMMIC

Prototype:

XDAS_Int32
AECG4_RK_disableMic(IMMICAECG4_Handle
mmhandle, XDAS_Int8 MicIndex);

Inputs:
Handle – handle to the multi-mic handle

Outputs:

Returns:

Use:

3.4.2.10 AECG4_RK_deleteMMIC

Prototype:
IMMICAECG4_Handle
AECG4_RK_deleteMMIC(IMMICAECG4_Handle
mmhandle);

Inputs:

Handle – handle to the multi-mic handle

Returns:

Handle to an AEC instance

Description:

AECG4_RK_deleteMMIC deletes an instance of the multi-mic AEC. AECG4_deleteMMIC should be called in order to free up any allocated memory.

3.4.3 Miscellaneous Functions

3.4.3.1 AECG4_RK_getBuildInfoString

Prototype:

char *AECG4_RK_getBuildInfoString();

Returns: Text string listing built-in features and component (AEC, AGC, Noise Reduction 2) software version numbers.

3.4.3.2 AECG4_RK_getBuildInfo

Prototype:

AECG4_BuildInfo_t *AECG4_RK_getBuildInfo();

Returns: BuildInfo structure indicating built-in features. (See iaecg4.h for structure members.)

3.4.3.3 AECG4_RK_getParamCount

RK_API XDAS_Int16 AECG4_RK_getParamCount();

3.4.3.4 AECG4_RK_getParamNames

```
RK_API XDAS_Int16 AECG4_RK_getParamNames(char *pParamNameTable[], RK_Int16 TableSize);
```

3.4.4 Lock Callback Function

The lock callback function is a user supplied function that enables the AEC to request, use, and delete a software lock.

Prototype:

```
typedef enum
{
        CREATE_LOCK,
        LOCK,
        UNLOCK,
        DELETE_LOCK
}               LockAction_e;

typedef RK_UInt32 (LockCallback_t)(
                void *LockHandle,
                char *Name,
                LockAction_e Action,
                void **CreatedLock
                );
```

Inputs:

LockHandle – unless Action is CREATE_LOCK, this specifies the handle to the lock
Name – optional, used for debugging
Action:
CREATE_LOCK – creates a software lock instance
LOCK – requests a lock
UNLOCK – releases a lock
DELETE_LOCK – deletes the software lock instance

Outputs

**CreatedLock – if Action is CREATE_LOCK, this points to the handle to the newly created lock

Returns

Not used

3.5 Sample Host Code

3.5.1 Single Microphone Example

```
#include <AECG4\include\iaecg4.h>
extern IAECG4_Fxns AECG4_RK_IAECG4;

#define FRAME_SIZE 64
#define FRAME_SIZE_MSEC 4

// AEC Parameters
```

```
IAECG4_Params MYAECG4_PARAMS = {
        // Base Parameters
        sizeof(AECG4Params_t),
        0,                          // lockCallback
        FRAME_SIZE,                 // frameSize (samples)
        0,                          // antiHowlEnable
        SAMPLING_RATE,              // samplingRate
        SAMPLING_RATE/2,            // maxAudioFreq
        2*FRAME_SIZE_MSEC,          // fixedBulkDelayMSec
        0,                          // variableBulkDelayMSec
        0,                          // initialBulkDelayMSec
        64,                         // activeTailLengthMSec
        64,                         //totalTailLengthMSec
        6,                          //txNLPAggressiveness
        30                          //  MaxTxLossSTdB
        10,                         //MaxTxLossDTdB
        6,                          // MaxRxLossdB;
        0,                          // initialRxOutAttendB
        -85,                        // targetResidualLeveldBm;
        -90,                        // maxRxNoiseLeveldBm;
        -12,                        // worstExpectedERLdB
        3                           // rxSaturateLeveldBm
        1,                          // noiseReduction1Setting
        0,                          // noiseReduction2Setting
        1,                          // cngEnable
        0,                          // fixedGaindB
// txAGC Parameters
        0,                          // RK_Int8 txAGCEnable;
        10,                         // RK_Int8 txAGCMaxGaindB;
        0,                          //RK_Int8 txAGCMaxLossdB;
        -10,                        // RK_Int8 txAGCTargetLeveldBm;
        -50,                        //RK_Int8 txAGCLowSigThreshdBm;
        // rxAGC Parameters
        0,                          // RK_Int8 rxAGCEnable;
        10,                         //rxAGCMaxGaindB;
        15,                         //rxAGCMaxLossdB;
        -10,                        //rxAGCTargetLeveldBm;
        -40,                        //rxAGCLowSigThreshdBm;
        1,                          //rxBypassEnable
        0,                          //maxTrainingTimeMSec
        -40,                        //trainingRxNoiseLeveldBm
        0,                          //RK_Int16 pTxEqualizer
        0,                          //mipsMemReductionSetting
        0,                          //mipsReductionSetting2
        0                           //reserved

};



IAECG4_Handle hAEC;     // define handle to AEC channel
```

```
// Instantiation
void init_channel()
{
. . .
hAEC = AECG4_RK_create(0, &MyParams);
. . .
}
void delete_channel()
{
. . .
AECG4_RK_delete(hAEC);
. . .
}

// The process channel is to be called once per frame.

short int RxIn[FRAME_SIZE], TxIn[FRAME_SIZE], RxOut[FRAME_SIZE], TxOut[FRAME_SIZE];

void
process_channel( XD
AS_Int16 *pRxIn,
XDAS_Int16 *pRxOut,
XDAS_Int16 *pTxIn,
XDAS_Int16 *pTxOut
)
{
. . .
   AECG4_RK_apply(hAEC,pRxIn,pRxOut,pTxIn,pTxOut);
. . .

}
```

3.5.2 Multi-Microphone and Dual Mic Noise Reduction Example

! NOTE: In accordance with your licensing agreement, your product may be restricted for Single Mic Use only.

```
#include <AECG4\include\iaecg4.h>

extern IAECG4_Fxns AECG4_RK_IAECG4;

// AEC Parameters
IAECG4_Params MYAECG4_PARAMS = {
        // Base Parameters
        sizeof(AECG4Params_t),
        0,                          // lockCallback
        FRAME_SIZE,                 // frameSize (samples)
        0,                          // antiHowlEnable
        SAMPLING_RATE,              // samplingRate
        SAMPLING_RATE/2,            // maxAudioFreq
        2*FRAME_SIZE_MSEC,      //fixedBulkDelayMSec
```

```
    0,                                  // reserved
```

```
        0,                              // reserved
        64,                             // activeTailLengthMSec
        64,                             //totalTailLengthMSec
        6,                              //txNLPAggressiveness
        30                              //  MaxTxLossSTdB
        10,                             //MaxTxLossDTdB
        6,                              // maxRxLossdB
        0,                              // initialRxOutAttendB
        -85,                            // targetResidualLeveldBm;
        -90,                            // maxRxNoiseLeveldBm;
        -12,                            // worstExpectedERLdB
        3                               // rxSaturateLeveldBm
        1,                              //noiseReduction1Setting
        0,                              //noiseReduction2Setting
        1,                              // cngEnable
        0,                              // fixedGaindB
// txAGC Parameters
        0,                              // RK_Int8 txAGCEnable;
        10,                             // RK_Int8 txAGCMaxGaindB;
        0,                              //RK_Int8 txAGCMaxLossdB;
        -10,                            // RK_Int8 txAGCTargetLeveldBm;
        -50,                            //RK_Int8 txAGCLowSigThreshdBm;
        // rxAGC Parameters
        0,                              // RK_Int8 rxAGCEnable;
        10,                             //rxAGCMaxGaindB;
        15,                             //rxAGCMaxLossdB;
        -10,                            //rxAGCTargetLeveldBm;
        -40,                            //rxAGCLowSigThreshdBm;
        1,                              //rxBypassEnable
        0,                              //maxTrainingTimeMSec
        -40,                            //trainingRxNoiseLeveldBm
        0,                              //RK_Int16 pTxEqualizer
        0,                              //mipsMemReductionSetting
        0,                              //mipsReductionSetting2
        0                               //reserved

};


IMMICAECG4_Handle hAEC;        // define handle to AEC channel

// Instantiation
void init_channel()
{
. . .
hAEC = AECG4_RK_create_MMIC(0, &MyParams, NMicrophones, 0);

. . .
}
void delete_channel()
{
. . .
```

```
AECG4_RK_deleteMMIC(hAEC);
. . .
}

// The process channel is to be called once per frame.
short int TxIn [FRAME_SIZE], TxIn1[FRAME_SIZE], TxIn2[FRAME_SIZE], TxIn3[FRAME_SIZE],
TxIn4[FRAME_SIZE], TxIn5[FRAME_SIZE];
short int TxOut[FRAME_SIZE], TxOut1[FRAME_SIZE], TxOut2[FRAME_SIZE], TxOut3[FRAME_SIZE],
TxOut4[FRAME_SIZE], TxOut5[FRAME_SIZE];

static short * ppTxIn [] =
   { &TxIn[0],
   &TxIn1[0],
   &TxIn2[0],
   &TxIn3[0],
   &TxIn4[0],
   &TxIn5[0]
};
static short * ppTxOut [] =
   { &TxOut[0],
   &TxOut1[0],
   &TxOut2[0],
   &TxOut3[0],
   &TxOut4[0],
   &TxOut5[0]
};

void
process_channel( XD
AS_Int16 *pRxIn,
XDAS_Int16 *pRxOut,
XDAS_Int16 **ppTxIn,
XDAS_Int16 **ppTxOut
)
{
. . .
   AECG4_RK_apply_MMIC(hAEC,pRxIn,pRxOut,ppTxIn,ppTxOut);
. . .

}
```