Echo Canceller Unit (ECU)
Users Manual

Version 1.10

January 19, 2010

TABLE OF CONTENTS

1 Introduction

This document provides detailed information regarding integration of the Telogy Echo Canceller Unit (ECU) ECO[1] into a real-time system to eliminate line echo.

1.1 Revision History

Version	Date	Description
1.1	07-07-97	Initial version of the document.
1.2	07-19-00	Updated for MSEC R8.0.
1.3	01-08-01	Updated for MSEC R8.1.
1.4	09-26-01	Updated for MSEC R9.0.
1.5	08-20-02	Updated for ECU version 9.14.0.
1.6	10-28-02	Updated for ECU version 9.23.0 & 9.10.5.
1.7	04-23-03	Updated for ECU version 10.23.0.
1.8	02-25-04	Updated for ECU version 10.58.0.
1.9	09-25-08	Updated for ECU version 10.87.0.
1.10	01-19-10	Updated for ECU version 10.89.0.

[1] ECO - embedded communication object

2 Overview

Echo cancellers are voice operated devices placed in the 4-wire portion of a circuit and are used for reducing echo by subtracting an estimated echo from the circuit echo. They may be characterized by whether the transmission path or the subtraction of the echo is by analogue or digital means. Figure 1 shows a block diagram of a digital echo canceller. Figure 2 shows an echo canceller used for canceling near-end echo.

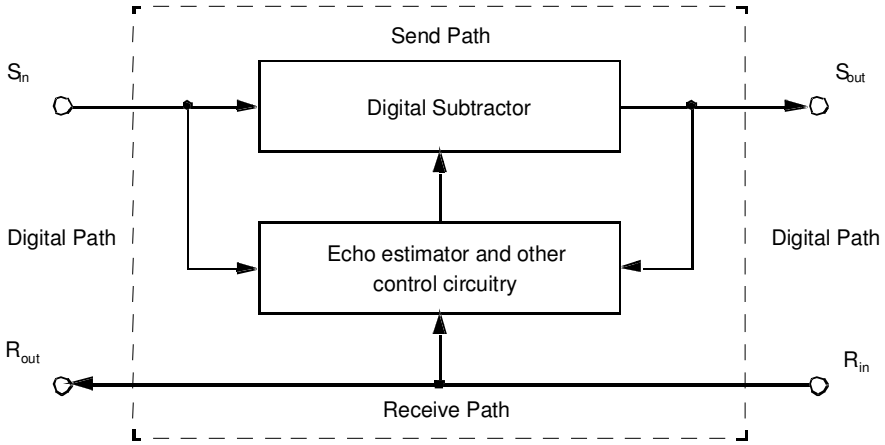

Figure 1. Digital echo canceller. Digital echo canceller interfaces at 64kbit/s

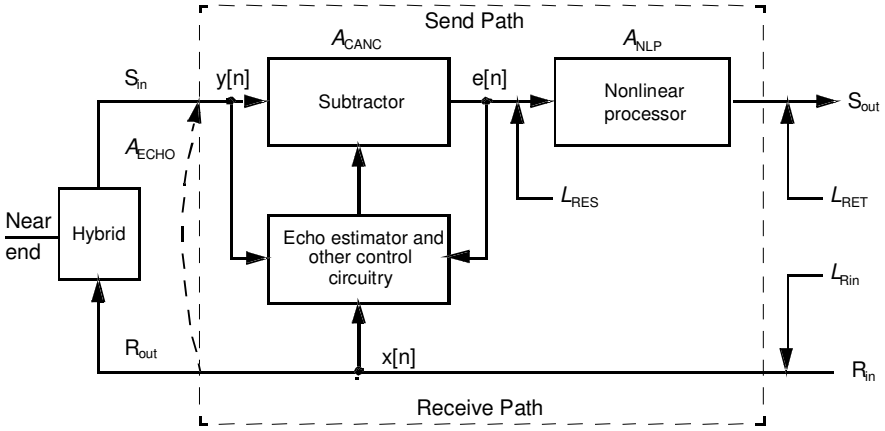

Figure 2. Echo canceller for near-end echo cancellation. This echo canceller is used for reduction of near-end echo present on the send path. It subtracts an estimated echo from the near-end echo.

The line echo canceller described in this text may be configured to operate as both a multiple segment as well as a single segment (full filter) echo canceller for near-end echo cancellation. In general, the echo canceller (EC) is configured for multiple segments only for tail lengths above 32ms. The EC does not include any tone disabling logic, a separate module should be used for that purpose.

2.1 Terminology

Throughout the text we will refer to several important terms and parameters:

- **Foreground (active) filter** – filter used within the transmit thread for echo removal.
- **Background filter** - filter used for adaptation to the echo path impulse response. The coefficients are continuously updated. Only those coefficients within *segments* identified by the search process are adapted. The background filter adaptation will generally occur in the transmit thread but could be also be performed within a separate low priority thread if necessary.
- **x signal** - far end signal (see Figure 2.)
- **y signal** - near end *composite* signal (contains both near end signal and echoed far end signal, as shown in Figure 2.)
- **e signal** - error signal prior to the NLP. This signal contains the *actual* error in estimating the echo path impulse response when there is no near end activity, i.e. when the echoed far end signal is the only component present within the y signal. In all other cases, the signal is a mixture of error and near end signals (see Figure 2.)
- **Doubletalk** - the condition of having considerable activity in both the x and y signals. One should note that this condition may also occur when, due to a large echo, the far end signal is propagated to the y signal without having any actual near end signal activity.
- **nonlinear processor (NLP)** – logic to provide additional echo suppression when doubletalk is not present
- **Search filter** - an adaptive FIR filter used to estimate the echo path impulse response over the entire *tail.* This filter is used to identify *segments* of the filter that contain hybrid reflections.
- **Tail length (*t*)** – represents the greatest echo path impulse response that the EC may successfully cancel. In the MSEC, this is the number of filter coefficients (taps) in the search filter. For a full filter configuration, this is the length, in taps, of both the foreground and background filters. This parameter is configurable, although some restrictions apply.

- **Filter segment –** a portion of the search filter that has been identified (through *search placement*) as likely containing a hybrid reflection. Each segment is specified by the pair (*delay, length)* where *delay* refers to the offset from the beginning of the search filter in taps and *length* refers to the segment length in taps. For the MSEC, only those taps that fall within filter segments are adapted in the background update.

- **Segment length (L_s)** – length, specified in number of samples, of the smallest signal segment that the echo canceller processes. The segment length remains constant throughout a call. Currently, 2.5ms segments are the norm (L_s **=** 20), however if 5.5ms frame handling is enabled (see Section 3.2, Compile Time Options), segment lengths of 22 are also possible.

- **Block length (L_B)** – length, specified in number of samples, of a block that is used for the adaptive filter update. The block length will generally be fixed as the size of two segments (2 L_s). The block length remains constant throughout a call. Currently, 5ms blocks are the norm (L_B **=** **4**0), however if 5.5ms frame handling is enabled (see Section 3.2, Compile Time Options), block lengths of 44 are also possible.

- **Frame length (*f*)** – length, specified in number of samples, of external signal frames that are supplied to the echo canceller for real-time processing. This parameter is configurable.

The following constraints apply to the length parameters:

$$L_S < L_B, \quad L_S \le f, \; L_B = n \cdot L_S$$

where n is a positive integer. It is easy to see that n > 1. In addition, it is strongly suggested that $f \ge L_B$ in order to minimize the peak MIPS. Hence, the length values should satisfy:

$$L_S < L_B \le f, \quad L_B = n \cdot L_S$$

3 Implementation

The multi-segment echo canceller (MSEC) is implemented in the ANSI C language. Depending on the target platform parts of it may also be optimized in assembly language for speed. The same source code works on both floating point and fixed-point platforms provided that the fractional arithmetic macros are properly implemented on both. Some of the algorithms may be implemented differently on floating and fixed point platforms, e.g. block LMS algorithm.

The echo canceller may be divided into the following functional blocks:

- initialization, configuration, and control block
- foreground processing block
  - Echo removal
  - Power measurements
  - Doubletalk detection
  - Nonlinear processor
  - Preparation of update parameters
- background processing block
  - Convergence tracking and update control
  - filter update (step size calculation, echo removal, coefficient update)
  - search filter update (coefficient update, segment placement)

3.1 Assumptions

The ECU implementation makes several important assumptions:

- **Zero mean signals** - all input signals (near-end and far-end) are assumed to be zero mean (desirable when implementing block LMS adaptive algorithm.)
- **Power levels** - all signals are scaled in such a way that a digital sine wave of amplitude 2^{15} corresponds to the +3dBm0 analog sine wave. This is also the saturation point for all signals.
- **Synchronization and delay** - the near-end and far-end signals are assumed to be synchronized in time when presented to the echo canceller. Far-end delay line is assumed to be of proper length to accommodate for all sources of signal delay inside and outside of the DSP platform.
- **Minimum hybrid loss** - the minimum hybrid loss is assumed to be at least 6dB. Otherwise the echo canceller may still work, but with degraded performance.

- **Nonlinearities** - it is assumed that there is no clipping in the input signals and that the nonlinear distortion is negligible since the hybrid model is linear. Clipping of the far-end signal may sometimes be acceptable if the near-end echo path does not introduce additional nonlinear distortion or clipping, i.e. if the far-end signal on its path towards the near-end input is not additionally distorted.

- **Tone detection at near-end** - the tone detection at near-end is not performed. It is assumed that such processing will be done outside the echo canceller when necessary. In such a case echo canceller may be disabled from the outside. Doubletalk detection would take care of preventing the filter updates when tones are present at the near-end in normal operation.

3.2 Compile Time Options

There are ten compile-time options for ECU builds.

1) Multiple segment handling
2) Near mode build
3) Delay line compression
4) ECU instance relocation
5) AAL1/AAL2 (5.5ms frame) handling
6) Debug messaging
7) 0/3dB ERL configuration
8) Constant PCM pattern detection
9) RERL estimation for RTCP-XR
10) Wide band (16KHz) operation

Each option is independent and may be combined with any other option. The settings at compile-time may be determined at run-time via the ecuGetPerfomance() API function call described below in Section 4.2.3.

3.3 Memory Buffers

Each ECU instance references up to fifteen buffers. Three of the buffers are necessary only if the ECU is configured for operation as an MSEC. One additional context buffer is shared by all instances. All buffers are described in detail below.

3.3.1 Context Buffer

This buffer stores information that is common among all ECU instances. Its size is equal to sizeof(ecuContext_t). It is generally stored in external RAM (when available) since its elements are not required for intense computations and is non-volatile. This buffer is shared by all channels. It is not allocated through ecuGetSize() and ecuNew() and must be statically allocated. See Table 1 below for a description of the ecuContext_t structure.

Member	Purpose
(void *) exception	Exception handling function pointer (cast to void *).
void (*debugStrmWrite)	Debug streaming function pointer. Should default to NULL.
tint (*mipsEcuInstEvt)	MIPS Agent ECU instance open/close function pointer (see MIPS Agent documentation).
void (*srchPrep)	Search preparation function pointer. This may be used to swap search filter coefficients in/out of internal memory in builds with limited IRAM.
void (*sendOutFcn)	ECU send out function pointer.
void (*receiveOutFcn)	ECU receive out function pointer.
tint max_samples_per_frame	Maximum number of samples per frame.
tint max_filter_length	Maximum filter length in taps.
tint max_filter_seg_length	Maximum filter segment buffer length in taps.
tint max_filter_seg_count;	Maximum allowed active filter segments.
tint max_y2x_delay	Maximum extra system delay in samples.
tint max_y_delay	Maximum extra transmit delay in samples.
tulong expanded_bf	Bitfield representing those portions of the delay line already expanded. Scratch, shared by all instances. Used only if delay line compression is enabled.
linSample *rxout_buf_base	Pointer to base of the scratch delay line. Scratch, shared by all instances. Used only if delay line compression is enabled.
linSample *expand_ptr	TDM aligned pointer within scratch delay line. Scratch, shared by all instances. Used only if delay line compression is enabled.
word *pack_ptr	TDM aligned pointer within packed delay line. Scratch, shared by all instances. Used only if delay line compression is enabled.

Table 1. ecuContext_t Data Structure

3.3.2 Instance Buffer

The buffer that stores all the relevant information for the ECU instance between calls to ECU functions, including pointers to other buffers. Some compilers may require that this buffer be aligned on an even boundary. Its size is equal to sizeof (ecuInst_t). It is generally stored in external RAM (when available) since its elements are not required for intense computations. Each channel stores the context of the associated ECU instance in this buffer. Accordingly, the buffer is non-volatile.

3.3.3 Foreground Filter Segment Buffer

This buffer stores the coefficients for all foreground filter segments between calls to ecuSendIn(). The coefficients are stored sequentially in increasing segment order starting with segment 0. There is no space between segments within the buffer (see Figure 3). The buffer is

designed to store the maximum number of coefficients allowed for all filter segments, hence the size is max_filter_seg_length*sizeof(Fract). The coefficients in this buffer are used for echo removal. Due to the computational complexity of this operation, the buffer should be stored in internal RAM to keep MIPS at a minimum. The buffer is non-volatile and has no alignment requirements.

3.3.4 Background Filter Segment Buffer

This buffer stores the coefficients for all background filter segments between calls to ecuSendIn(). The coefficients are stored sequentially in increasing segment order starting with segment 0. There is no space between segments within the buffer (see Figure 3). The buffer is designed to store the maximum number of coefficients allowed for all filter segments, hence the size is max_filter_seg_length*sizeof(Fract). The coefficients in this buffer are used in the LMS update routine. Due to the computational complexity of this operation, the buffer should be stored in internal RAM to keep MIPS at a minimum. The buffer is non-volatile and has no alignment requirements.

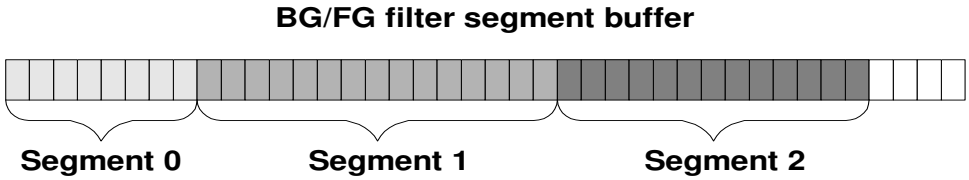

Figure 3. Filter segment buffers. The filter segment buffers contain the coefficients for all filter segments. The segments are stored back-to-back within the buffer in sequential fashion.

3.3.5 Background Error Signal Buffer

This buffer contains the background filter error signal (e signal) during background processing. The error signal is used to estimate and track the error energy over time. Since the elements of this buffer are used in computationally expensive operations, the buffer should be placed in internal RAM when available. This buffer is volatile. That is, it need not be preserved between calls to ecuAdaptiveFilter(). The buffer size is ECU_BLOCK_LENGTH*sizeof(linSample). No alignment is required.

3.3.6 Far-end Delay Line Buffer

This buffer is a delay line of far-end signal samples. An external mechanism must exist which aligns samples in this buffer with the samples in the near-end input frames for synchronization. The samples in this delay line are used in the removal of echo as well as both the background and search LMS procedures for filter coefficient adaptation.

The buffer size is calculated as L_d*sizeof(linSample), where L_d represents the worst case

number of samples calculated via ecuCalcReceiveInLength() over all possible frame lengths[2]. This buffer must be aligned for circular addressing on the target platform. The exact alignment requirements will depend upon the buffer size. For speed of execution, this buffer should be in internal RAM. This buffer is non-volatile.

It should be noted that if the delay line compression option was selected at compile-time, this buffer will take on new meaning. In this case the buffer will still hold the far-end samples, but in a packed format. In this case, the required size will be half of that returned by ecuCalcReceiveInLength() and no alignment will be necessary. For details see the document "Delay Line Compression.doc".

3.3.7 Expanded Delay Line Buffer

This buffer is referenced only if the delay line compression option was selected at compile-time. Otherwise, this buffer should be allocated with length 0 and will not be used.

In the case of delay line compression, this buffer is where the packed delay line buffer will be expanded for subsequent echo removal or search/background LMS adaptation. This buffer must be aligned for circular addressing on the target platform. The exact alignment requirements will depend upon the buffer size. For speed of execution, this buffer should be in internal RAM. This buffer is volatile and need not be retained between calls to ecuSendIn().

In the case of delay line compression, this buffer will have length (max_filter_length + fmax) where max_filter_length and fmax designate the maximum tail length (*t*) in taps and maximum frame length (*f*) respectively.

3.3.8 Background Update Buffer

This buffer serves as a work area for echo canceller functions. Specifically, this buffer provides a scratch area for the LMS update function. This buffer is volatile. That is, it does not need to be preserved between calls to ecuSendIn(). The buffer size can be calculated via the ecu_calc_bg_update_size() utility macro. Since this buffer is used in the computationally intensive LMS function, the buffer should be in internal RAM if available. This buffer does not require alignment.

It should be noted that this buffer generally requires 8 locations (8*sizeof(Fract)) for c54x platforms and 1024 locations for c55x platforms. This major discrepancy in size requirements helps in increasing the efficiency of the LMS operation on the C55x platform.

3.3.9 Doubletalk X-Power Buffer

This buffer holds receive_in signal power estimates for each of the last dt_x_power_buf_len segments. This buffer must be aligned on an even boundary. The buffer size may be calculated as floor(max_filter_length/160 + 1). The buffer may reside in external RAM if

[2] It should be noted that the worst case buffer size may not necessarily correspond to the maximum frame length. This is due to the fact that the buffer length may not monotonically increase as a function of frame length.

available and is non-volatile.

3.3.10 ERL Y-Energy Buffer

This buffer holds near-end signal energy estimates for each of the last erl_y_enr_len blocks. This buffer must be aligned on an even boundary. The buffer size may be calculated as (max_filter_length + 64)/ECU_ERL_EN_TCONST. The buffer may reside in external RAM and is non-volatile.

3.3.11 ACOM E-Energy Buffer

This buffer holds residual error energy estimates for each of the last erl_y_enr_len blocks. This buffer must be aligned on an even boundary. The buffer size may be calculated as (max_filter_length + 64)/ECU_ERL_EN_TCONST. The buffer may reside in external RAM and is non-volatile.

3.3.12 Background Filter Segment Buffer

This buffer holds the background filter segment delay/length pairs. Each pair consists of a delay with respect to the search filter[3], and length, both in taps. The buffer size is calculated as 2*max_filter_seg_count*sizeof(tint). The buffer may reside in external RAM and is non-volatile. No alignment requirements are associated with this buffer.

3.3.13 Foreground Filter Segment Buffer

This buffer holds the foreground filter segment delay/length pairs. Each pair consists of a delay with respect to the search filter[4], and length, both in taps. The buffer size is calculated as 2*max_filter_seg_count*sizeof(tint). The buffer may reside in external RAM and is non-volatile. No alignment requirements are associated with this buffer.

3.3.14 Wideband Delay Lines Buffer

This buffer holds the delay lines for tx and rx wide band split filters. The buffer size is calculated as 3*ECU_BS_LO_DL_SIZE*sizeof(Fract) + 2*ECU_BS_HI_DL_SIZE*sizeof(Fract). The buffer may reside in external RAM and is non-volatile. No alignment requirements are associated with this buffer.

---

[3] It should be noted that in the short-tail ECU implementation, there will be only one delay/length pair. This pair will be with respect to the BG/FG filter segment buffer which will contain a single "segment" at delay zero.

[4] It should be noted that in the short-tail ECU implementation, there will be only one delay/length pair. This pair will be with respect to the BG/FG filter segment buffer which will contain a single "segment" at delay zero.

3.3.15 Wideband Scratch Buffer

This buffer holds scratch buffers used by wide band ECU. The buffer size is calculated as ECU_INFRAME_MAXLENGTH*sizeof(Fract) + ECU_BS_LO_DL_SIZE*sizeof(Fract). The buffer may reside in external RAM and is volatile. No alignment requirements are associated with this buffer.

3.3.16 Search Buffer

This buffer holds the adaptive filter coefficients used by the search algorithm to identify hybrid reflections. This buffer is only necessary for long-tail (MSEC) implementations (as specified at compile-time). The length is specified as max_filter_length*sizeof(Fract). The search buffer is accessed extensively during search LMS updates. For this reason, the buffer should reside in internal RAM if available. This buffer is non-volatile. That is, it must be preserved between calls to ECU functions. No alignment requirements are associated with this buffer.

3.3.17 Search Scratch Buffer

This buffer is used by the search algorithm as a scratch buffer to perform several search related functions. Specifically, the buffer is used as a scratch area in the ecu_search_update_segs() routine for the background filter coefficient refresh as well as in the ecu_search_calc_sumsqu() routine to store the filter coefficient sum-of-squares for subsequent hybrid identification. This buffer is volatile. That is, it does not to be preserved between calls to ecuSendIn(). This buffer requires alignment on an even boundary. The length of the buffer is max(max_filter_length/8*sizeof(LFract), max_filter_seg_length*sizeof(Fract)). This buffer should reside in internal memory if available.

3.3.18 Search Counter Buffer

This buffer is used by the search algorithm to smooth the filter segment placement estimates. Each value in this buffer represents a 1ms sum-of-squares bin of the search filter coefficients. The size of the buffer is max_filter_length/8*sizeof(tint). This buffer is non-volatile and need not reside in internal RAM. No alignment requirements are associated with this buffer.

4 External API

The echo canceller API is composed of three major interfaces:

(1) Instantiation

(2) Control and Monitor

(3) Signal I/O

4.1 Instantiation API

This interface is used to create, destroy, open and close the echo canceller. It is composed of the API functions that follow.

4.1.1 ecuGetSizes

Prototype: `void ecuGetSizes (tint *nbufs, const memBuffer_t **bufs, ecuSizeConfig_t *cfg)`

Description: Obtains memory requirements for an ECU instance. Upon return, the first two parameters describe memory buffers that are requested by the echo canceller. The third parameter is used at input to help the echo canceller calculate the worst case buffer sizes. The vector of memory buffer descriptors returned in bufs must not be changed by external software. A separate copy must be obtained and used in subsequent ecuNew() and ecuDelete() functions. Note that the size configuration structure is defined as void. All buffers are described briefly below in Table 2. Refer to Section 3.3 above for details regarding buffer memory type, size, alignment, etc.

Index	Buffer Name	Description
0	Instance buffer	A buffer that stores the ECU instance context and state. The size depends on implementation, target compiler and CPU architecture.
1	Foreground filter segment	A buffer that stores the coefficients for all foreground filter segments between calls to ecuSendIn().
2	Background filter segment	A buffer that stores the coefficients for all background filter segments. The ecuAdaptiveFilter() function uses this buffer for the background filter update.
3	Background error signal	A buffer that contains the background filter error signal during the call to ecuAdaptiveFilter().
4	Far-end delay line	A delay line buffer for storing far-end signal samples. An external mechanism must exist which aligns samples in this buffer with the samples in near-end input frames.
5	Far-end expanded delay line	A buffer used only when the delay-line compression compile-time option is enabled. Packed far-end signal samples are expanded into this volatile buffer for use in echo removal and background/search LMS adaptation.

6	Background update buffer	A buffer that is used as a work area by echo canceller functions.
7	Doubletalk x-power buffer	A buffer that holds the receive_in signal power for the last dt_x_power_buf_len segments.
8	ERL y-energy buffer	A buffer that holds the near-end signal energy for the last erl_y_enr_len blocks.
9	ACOM e-energy buffer	A buffer that holds the residual signal energy for the last erl_y_enr_len blocks.
10	Background filter segment Delays/lengths	A buffer that stores the delays and lengths of all background filter segments.
11	Foreground filter segment Delays/lengths	A buffer that stores the delays and lengths of all foreground filter segments.
12	band split delay lines	A buffer that stores the delay lines for band split filters in wide band ECU.
13	wide band scratch buffers	A buffer that is used as work area for wide band ECU.
14	Search buffer	A buffer that holds the adaptive search filter coefficients. This filter is used for hybrid reflection identification.
15	Search scratch buffer	A buffer that is used as a scratch area by search algorithm functions.
16	Search counter buffer	A buffer that is used by the search algorithm to provide smoothing for the filter segment placement.

Table 2. ECU Memory Buffers

4.1.2 ecuNew

Prototype: `tint ecuNew (void **ecuInst, tint nbufs, memBuffer_t *bufs, ecuNewConfig_t *cfg)`

Description: Creates an ECU instance and distributes the supplied memory buffers. The first parameter, ecuInst, points to a memory location that will receive a pointer to the ECU instance structure for the channel. The ECU instance buffer is supplied within the vector of buffer descriptors pointed to by bufs. The number of buffers is specified by nbufs. Information on memory buffers should have been previously obtained via the ecuGetSizes() function. The number and order of memory buffers that are supplied to ecuNew() must be the same as returned in ecuGetSizes(). The ecuNew() function leaves the ECU instance in the closed state. The function returns an error code to indicate if the function completed successfully, see Table 4 below for descriptions of the error codes. The configuration structure is described below in Table 3.

Member	Purpose
tuint ID	The unique ECU instance identifier. Will be stored within the ECU instance and used when reporting exceptions.

Table 3. ecuNewConfig_t Data Structure

Error Code	Description
ECU_NOMEMORY	The function call failed due to memory allocation.
ECU_NOERR	The function call was successful
ECU_ERROR	The function did not complete successfully.

Table 4. ECU Error Codes

4.1.3 ecuOpen

Prototype: `void ecuOpen (void *ecuInst, ecuConfig_t *cfg)`

Description: Initializes and configures an ECU instance for the channel identified by ecuInst. The actual ECU buffer sizes are calculated by this function depending on information supplied by the configuration structure. It is possible that some of the buffers previously sized for the worst case situation by ecuGetSizes() and ecuNew() will not be completely used if the configuration parameters in ecuOpen() result in smaller requirements. Upon completion, this function sets the ECU instance to the open state. The configuration structure is described below in Table 5.

Member	Purpose
tint samples_per_frame	The number of samples per input frame. Must be less than or equal to the worst case specified in the ECU context max_samples_per_frame.
tint y2x_delay	The additional system delay in samples between the far-end and near-end signals. Must be less than or equal to the worst case specified in the ECU context max_y2x_delay.
tint y_delay	The estimated additional (near-end) processing delay in samples. Must be less than or equal to the worst case specified in the ECU context max_y_delay
void *sendOutInst	Pointer to instance data structure to accept near-end signal after echo removal. The function pointer to call is contained in the ECU context.
void *recOutInst	Pointer to instance data structure to pass voice samples to near-end. The function pointer to call is contained in the ECU context.
ecuConfigParam_t *cfgParam	Pointer to structure containing ECU configuration parameters (see Table 6)
const tint *pcm_expand_tbl	Pointer to the table to be used for PCM expansion (ulaw or alaw). Only necessary if delay line compression is enabled.
word pcm_zero	representation of PCM 0 in ulaw or alaw. Only necessary if delay line compression is enabled.

Table 5. ecuConfig_t Data Structure

Member	Purpose
tint filter_length	ECU filter length in taps.
tint config_bitfield	Configuration bitfield part 1 for setting ECU functionality on open. The bitfield must be the constructed as the bitwise OR of bit values from Table 9.

tint config_bitfield1	Configuration bitfield part 2 for setting ECU functionality on open. The bitfield must be the constructed as the bitwise OR of bit values from Table 9.
tint noise_level	Fixed comfort noise level in dBm0. Applies only if NLP is configured to generate fixed comfort noise.
Fract nlp_aggress	NLP aggressiveness. Q15 number [-1,1) indicating how aggressive the NLP should be in replacing near-end signal with comfort noise. Positive and negative values indicate a more and less aggressive NLP function respectively.
Fract cn_config	Comfort noise configuration. Q15 value [0,1) indicating comfort noise mixture. Zero and one correspond to pink and white noise respectively.

Table 6. ecuConfig_t Data Structure

4.1.4 ecuClose

Prototype: `void ecuClose (void *ecuInst)`

Description: Closes the echo canceller on the channel associated with ecuInst. This function has no effect upon an already closed instance of the ECU.

4.1.5 ecuDelete

Prototype:
```
void ecuDelete (void **ecuInst, tint nbufs, memBuffer_t
                *bufs)
```

Description: Deletes the echo canceller on a channel identified by ecuInst. Clears the instance pointer to NULL. This function results in an exception if the echo canceller has not been previously closed with ecuClose(). The number of memory buffers (nbufs) and their descriptors (bufs) are currently ignored.

4.2 Control and Monitor API

This interface is used to configure the functionality of the echo canceller and monitor its performance and statistics. It is composed of the following functions:

4.2.1 ecuControl

Prototype: tint ecuControl (void *ecuInst, ecuControl_t *ctl)

Description: External control of echo canceller functionality. A pointer to an ecuControl_t structure (see Table 7) passed as the second argument in the call provides both an enumerated "control code" (see Table 8) for selection of how to control the ECU, as well as a union of parameters each corresponding to a particular control code. To produce the desired effect,

this function should be invoked after the ECU instance has been opened with ecuOpen().

When the control code is ECU_CTL_MASK the bitfield in u.ctl_mask is built from the control modes of Table 9. The function returns an error code to indicate if the function completed successfully, see Table 4 above for descriptions of the error codes.

Member	Purpose
tint ctl_code	Specifies the category of action to be performed. Refer to Table 8 below for valid control codes.
union { Tint ctl_mask[2]; Tint bg_speed; Tint srch_speed; Tint place_speed; bool enable; Tint n_level; Fract nlp_aggress; Fract cn_config; } u;	 Applies when ctl_code is ECU_CTL_MASK. Applies when ctl_code is ECU_CTL_BG_LMS_SPEED. Applies when ctl_code is ECU_CTL_SEARCH_LMS_SPEED. **Obsolete** Applies when ctl_code is ECU_CTL_BG_LMS_ENABLE. Applies when ctl_code is ECU_CTL_FIXED_NOISE_LEVEL Applies when ctl_code is ECU_CTL_NLP_AGGRESSIVE Applies when ctl_code is ECU_CTL_NLP_CN_CONFIG

Table 7. ecuControl_t data structure.

Control Code	Description
ECU_CTL_MASK	Bitfield used to enable/disable ECU functionality. Each defined bit controls a specific feature (refer to Table 9 for bit definitions).
ECU_CTL_BG_LMS_SPEED	Controls rate of convergence of BG LMS. Defined values are ECU_FORCE_SLOW and ECU_ALLOW_FAST. If u.bg_speed is set to ECU_FORCE_SLOW, the BG LMS will update for every 2nd input sample. If u.bg_speed is set to ECU_ALLOW_FAST, the BG LMS will update for every input sample unless the BG filter has already converged.
ECU_CTL_BG_LMS_ENABLE	Enables/disables filter updates. Does not affect echo removal, i.e. echo is removed as long as the echo canceller is enabled.
ECU_CTL_SEARCH_LMS_SPEED	Controls rate of convergence of the search LMS. Defined values are ECU_FORCE_SLOW and ECU_ALLOW_FAST. If u.srch_speed is set to ECU_FORCE_SLOW, the search LMS will update for every 2nd input sample. If u.srch_speed is set to ECU_ALLOW_FAST, the search LMS will update for every input sample.
ECU_CTL_SEARCH_LMS_ENABLE	Enables/disables the search filter updates as well as filter segment placement.
ECU_CTL_FIXED_NOISE_LEVEL	Sets the comfort noise level (in dBm0) to be used when the NLP is configured to provide a fixed comfort noise level.
ECU_CTL_NLP_AGGRESSIVE	Controls NLP aggressiveness. Aggressiveness values range from [–1,1) (Q15) where positive and negative values imply a more and less aggressive NLP function respectively.

ECU_CTL_NLP_CN_CONFIG	Controls the ratio of white to pink noise for comfort noise generation. Values range from [0,1) (Q15) where 0 and 1 correspond to pink and white noise respectively.

Table 8. ECU control code enumeration and descriptions.

Control Mode	Description
ECU_ENABLE_ECHO_CANCELLER	When specified, the echo canceller will be processing the near-end signal in attempt to remove the echo. Otherwise, the near-end signal is passed through without changes and no processing is done on any of the inputs.
ECU_ENABLE_UPDATE	Enables/disables the filter updates. Does not affect the echo removal, i.e. echo is removed as long as the echo canceller is enabled. This control flag may be used to freeze the filter coefficients when testing the echo canceller performance.
ECU_ENABLE_NLP	Enables/disables the nonlinear processor. May be used in situations when the nonlinear processor is not needed or when testing the echo canceller performance.
ECU_ENABLE_AUTO_UPDATE	Enables/disables the automatic switch between fast and slow filter updates. If enabled, the echo canceller automatically switches to slow filter updates after convergence.
ECU_ENABLE_SEARCH	Enables/disables the search filter updates as well as filter segment placement.
ECU_ENABLE_CNG_ADAPT	Enables/disables adaptive comfort noise generation.
ECU_ENABLE_OPNLP_DETECT	Enables/disables 4-wire detection.
ECU_CLEAR_FG	Immediately clears the foreground filter coefficients. May be used when testing echo canceller performance.
ECU_CLEAR_BG	Immediately clears the background filter coefficients. May be used when testing echo canceller performance.
ECU_CLEAR_SEARCH	Immediately clears the search filter coefficients and initializes the hybrid search algorithm and related variables.
ECU_DISABLE_DEMEANING	Enables/disables demeaning of search filter.
ECU_FORCED_NLP_CNG	Enables/disables constant comfort noise generation.
ECU_ NLP_NORMAL_LEVEL	Enables/disables further NLP aggressiveness if it is known that the near-end and far-end levels are balanced. Note: should disable by default.
ECU_ENABLE_NLP_PHASE_RND	Enables/disables NLP phase randomization.
ECU_ERL_CONFIG_BIT_0	0/3dB ERL configuration:

ECU_ERL_CONFIG_BIT_1	`ECU_ERL_CONFIG_BIT_1 = 0,`
	`ECU_ERL_CONFIG_BIT_0 = 0: 6dB ERL`
	`ECU_ERL_CONFIG_BIT_1 = 0,`
	`ECU_ERL_CONFIG_BIT_0 = 1: 3dB ERL`
	`ECU_ERL_CONFIG_BIT_1 = 1,`
	`ECU_ERL_CONFIG_BIT_0 = 0: 0dB ERL`
	`ECU_ERL_CONFIG_BIT_1 = 1,`
	`ECU_ERL_CONFIG_BIT_0 = 1: Invalid`

ECU_ENABLE_ECP_CHG_DETECT	Enables/disables echo path change detection. Note: should disable by default.
ECU_ENABLE_CPD_DETECT	Enables/disables constant PCM pattern detection
ECU_ENABLE_NONLINEAR_EP	Enables/disables non-linear echo path handling. Note: should disable by default.

ECU_ENABLE_FAST_CPS	Enables/disables fast constant power signal detection. This feature is used to better handle modem signals.

Table 9. ECU Control Bit Masks

4.2.2 ecuGetFilter

Prototype:
```
tint ecuGetFilter (void *ecuInst, tint select, tint
                   start, tint num, Fract *hbuf)
```

Description: This function returns ECU filter coefficients from the filter buffer selected by *select* in the buffer pointed to by *hbuf*. It may be necessary to call this routine several times for each filter (BG, FG, search) if the output buffer is not big enough to hold all coefficients. The function returns an error code to indicate if the function completed successfully, see Table 4 above for descriptions of the error codes. Table 10 below describes the filter select codes.

Select Code	Description
ECU_FLT_FOREGROUND	Specifies the ECU's foreground filter segment buffer. It should be noted that the coefficients selected for retrieval might span more than one filter segment.
ECU_FLT_BACKGROUND	Specifies the ECU's background filter segment buffer. It should be noted that the coefficients selected for retrieval might span more than one filter segment.
ECU_FLT_SEARCH	Specifies the ECU's search filter. This code is only valid for multisegment echo cancellers.

Table 10. ECU Filter Select Codes

4.2.3 ecuGetPerformance

Prototype:
```
void ecuGetPerformance (void *ecuInst, tint *state_bf,
                        ecuVersion_t *version,
                        ecuPerform_t *perform,
                        ecuUpdateStat_t *ustat,
                        LFract *noise_x, LFract
                        *noise_y, tint *tail_len,
                        bool rustat);
```

Description: Reports ECU states, performance, version and debug statistics for the channel identified by ecuInst. The caller must provide memory for state variables, the performance and statistics buffers. The state_bf is a two word vector. The statistics will be reset if the rustat parameter is TRUE. The noise_x is far-end noise estimate. The noise_y is near-end noise estimate. The tail_len is echo canceller tail length in samples. It should be noted that the performance parameters are calculated and recorded for each update block prior to the background process being executed. As a

result, the measurements may not reflect real performance during periods of doubletalk.

The contents of the performance, version and statistics reports are described below in Table 11, Table 12 and Table 13 respectively.

Member	Purpose
LFract Px	Far-end signal power over the last 20ms of update.
LFract Py	Near-end signal power over the last 20ms of update.
LFract Pe	Residual error power over the last 20ms of update. Measured before the nonlinear processor.
Fract erlest	The estimated echo return loss (ERL) in dB (Q4).
Fract acomest	The estimated combined loss (ACOM) in dB (Q4) prior to NLP.

Table 11. ecuPerform_t Data Structure

Member	Purpose
tuint release	ECU release number. May not match DSP release number.
tuint version	ECU version number.
tuint revision	ECU revision number.
tuint features	Bitfield representing the compile-time features included in the ECU build. See Table 14 for bit definitions

Table 12. ecuVersion_t Data Structure

Member	Purpose
tulong attempt_update	The number of blocks in which the background filter adaptation was executed.
tulong attempt_search	The number of blocks in which search filter adaptation or placement was executed.
tulong erle_bypass	The number of blocks in which the background LMS adaptation was slowed due to already high echo return loss enhancement (ERLE).
tulong xtone_bypass	The number of blocks in which the background LMS adaptation was slowed due to the suspected presence of tones in the far-end signal.
tulong other_bypass	The number of blocks in which the background LMS adaptation was not performed due to suspected divergence or already low error power.
tulong xidle	The number of blocks in which the background LMS adaptation was not performed due to low far-end signal power.
tuint divergence	The number of times the BG filter coefficients were reset due to suspected divergence.
tuint srch_divergence	The number of times the search filter coefficients were reset due to suspected divergence.
tuint fg_switch	The number of times foreground (active) filter coefficients were replaced by those in the background filter.
tuint bg_switch	The number of times the background filter coefficients were replaced by those in the foreground filter. May indicate problems with convergence in noisy environments.

tuint openloop_found	The number of times open loop is detected.
tuint numseg_change	The number of times the number of background filter segments has been changed by the segment placement routine
tuint segment_change	The number of times the filter segment placement changed while the number of segments remained constant
tuint converge_exit	The number of times the converged state was exited
tuint txbssat_events	The number of times saturation happened in tx band split filter in wide band ECU
tuint rxbssat_events	The number of times saturation happened in rx band split filter in wide band ECU

Table 13. ecuUpdateStat_t Data Structure

Version Bit	Meaning
ECU_VERF_MULTISEGMENT	Multiple segment (long-tail) ECU enabled.
ECU_VERF_5P5MS_FRAME	5.5ms frame handling enabled.
ECU_VERF_DLINE_COMPRESS	Delay line compression enabled. Note that this requires accompanying PIU changes.
ECU_VERF_DEBUG_STREAM	Debug streaming enabled.
ECU_VERF_NEAR_MODE	ECU built with near mode set for near function calls only.
ECU_VERF_03DB_CONFIG	0/3dB ERL support enabled.
ECU_VERF_CONST_PCM_DETECT	Constant PCM pattern detection enabled.
ECU_VERF_INST_RELOCATION	ECU instance relocation enabled.
ECU_VERF_RERL_ESTIMATE	RERL estimation for RTCP-XR enabled.
ECU_VERF_WIDE_BAND	Wide band ECU enabled.
ECU_VERF_TELOGY_ECAN	Telogy ECU integrated.

Table 14. ECU compile time options

4.3 Signal I/O API

This interface is used to exchange the input and output signals between the echo canceller and other modules within the DSP architecture. It is composed of the following functions.

4.3.1 ecuSendIn

Prototype:
```
void ecuSendIn (void *ecuInst, void *vsend_in, void
                *vrecv_in, void *vsend_out)
```

Description: This is the main echo canceller routine responsible for echo removal, doubletalk detection, signal buffer management, tracking of signal powers, posting of filter updates, etc. The function accepts near-end and far-end signal frames and starts by saving the far-end samples into a delay line. Next, the input (near-end) frame is processed as outlined below:

- The frame is divided into 2.5ms segments for sequential processing.

- Echo is removed from the segment using the foreground filter and delay line.
- Signal power tracking is performed.
- Doubletalk detection is performed.
- If necessary, nonlinear processing is performed.
- LMS adaptation is performed for every block (unless it is disabled via ecuControl())

The output (echo cancelled) samples are available via the vsend_out pointer.

4.3.2 ecuReceiveIn

Prototype: `void ecuReceiveIn (void *ecuInst, void *speech_samples)`

Description: This function is called for any processing needed on the far-end voice samples. Presently, this function performs no processing.

5 Integration

Following are the major requirements and issues regarding integration.

5.1 General Buffer Placement

In order to save MIPS, the guidelines shown in the following table should be followed regarding buffer placement:

No	Buffer	SA/DA RAM	Static/Scratch	Requirement
1	FG_FILTER	SARAM	Static	Different bank from 2
2	RX_BUF	DARAM	Static	None
3	BG_FILTER	SARAM	Static	Different bank from 2,4
4	BG_UPDATE_BUF	SARAM	Scratch	Different bank from 2,3
5	SRCH_FILTER	SARAM	Static	Different bank from 2,4
6	Stack			Different bank from 2

Table 15. Buffer placement strategy

In addition to the guidelines in the above table, the buffers RX_BUF, BG_FILTER, BG_UPDATE_BUF and SRCH_FILTER should not be in the same bank as the object code eculms.obj (.text). Additionally, the buffers RX_BUF and FG_FILTER should not be in the same bank as ecuerm.obj (.text).

5.2 Search Buffer Swapping

The search filter buffer(s) may need to be swapped in and out of external RAM. This is implementation dependent. If internal RAM is tight, a swap facility has been incorporated into the ECU search algorithm. The srchPrep function pointer passed in the ecuNewConfig_t structure to ecuInit() should point to a function that will conform to the API for this function type and will swap the search filter in and out of external RAM. Otherwise a pointer to NULL will suffice.

5.3 ECU Context

The ECU context should be statically allocated and initialized in a separate file.

6 Performance

Below we list the MIPS and memory requirements for specific functional areas of the ECU for both full-filter and multi-segment ECU.

6.1 Program Memory

Table below provides insight into the program memory requirements for the ECU component when various compile-time options are enabled/disabled.

Functionality	**Program Memory** (including .text, .bss and .const)
	C55x (words)
Base (full filter ECU)	6208
Search (MSEC)	1484
Delay line compression	503
5.5ms frame handling	25

Table 16. Program memory requirements

6.2 Data Memory

The following sections provide data memory buffer requirements for various compile-time configurations. Each table, MEM Type specifies the type of memory needed for the data buffer. It can be external or internal. ALIGN specifies the log2 alignment requirement for the buffer. SIZE is the size of the buffer in words. If VOLATILE is true, the buffer is scratch memory. Otherwise, it is permanent memory.

6.2.1 Instance structure

Base Configuration	**MEM Type**	**ALIGN**	**SIZE**	**VOLATILE**
Full filter ECU (32ms, 10ms frame)	external	1		no
Variations				
Delay line compression	external	1		no
5.5ms frame handling	external	1		no

Table 17. Instance structure data memory requirements

6.2.2 FG/BG Filter Buffers

Base Configuration	MEM Type	ALIGN	SIZE	VOLATILE
Full filter ECU (32ms, 10ms frame)	internal	0	256	no
Variations				
None	-	-	-	-

Table 18. FG/BG filter buffer data memory requirements

6.2.3 BG Error Buffer

Base Configuration	MEM Type	ALIGN	SIZE	VOLATILE
Full filter ECU (32ms, 10ms frame)	internal	0	40	no
Variations				
5.5ms frame handling	internal	0	44	yes

Table 19. BG error buffer data memory requirements

6.2.4 Far-end Delay Line Buffer

Base Configuration[5]	MEM Type	ALIGN	SIZE	VOLATILE
Full filter ECU (32ms, 10ms frame)	internal	9	376	no
Variations				
Full filter ECU (8ms tail)	internal	8	184	no
Full filter ECU (16ms tail)	internal	8	248	no
MSEC ECU (64ms tail)	internal	10	632	no
MSEC ECU (128ms tail)	internal	11	1144	no
5ms frame	internal	no chng.	size-40	no
5.5ms frame	internal	no chng.	size-36	no
delay line compression	internal	0	size/2	no

Table 20. Far-end delay line buffer data memory requirements

[5] Assumes y2x delay of 5ms and y delay of 0ms.

6.2.5 Delay Line Expansion Buffer

Base Configuration	MEM Type	ALIGN	SIZE	VOLATILE
Full filter ECU (32ms, 10ms frame)	internal	0	0	yes
Variations				
Delay line compression (DLC)	external	9	336	yes
DLC, full filter ECU (8ms tail)	external	8	144	yes
DLC, full filter ECU (16ms tail)	external	8	208	yes
DLC, MSEC ECU (64ms tail)	external	10	592	yes
DLC, MSEC ECU (128ms tail)	external	11	1104	yes
DLC, 5ms frame	external	no chng.	size-40	yes
DLC, 5.5ms frame	external	no chng.	size-36	yes

Table 21. Delay line expansion buffer data memory requirements

6.2.6 BG Work Buffer

Base Configuration	MEM Type	ALIGN	SIZE	VOLATILE
Full filter ECU (32ms, 10ms frame)	internal	0	8	yes
Variations				
c55x , 8ms tail	internal	1	64	yes
c55x , 16ms tail	internal	1	128	yes
c55x , 32ms tail	internal	1	256	yes
c55x , MSEC (64ms tail)	internal	1	512	yes
c55x , MSEC (128ms tail)	internal	1	1024	yes

Table 22. BG work buffer data memory requirements

6.2.7 Doubletalk X-Signal Power Buffer

Base Configuration	MEM Type	ALIGN	SIZE	VOLATILE
Full filter ECU (32ms, 10ms frame)	external	1	3	no
Variations				
MSEC (64ms tail)	external	1	4	no
MSEC (128ms tail)	external	1	7	no

Table 23. Doubletalk x-signal power buffer data memory requirements

6.2.8 ERL Y-Energy Buffer

Base Configuration	MEM Type	ALIGN	SIZE	VOLATILE
Full filter ECU (32ms, 10ms frame)	external	1	8	no
Variations				
MSEC (64ms tail)	external	1	14	no
MSEC (128ms tail)	external	1	27	no

Table 24. ERL y energy signal buffer data memory requirements

6.2.9 ACOM E-Energy Buffer

Base Configuration	MEM Type	ALIGN	SIZE	VOLATILE
Full filter ECU (32ms, 10ms frame)	external	1	8	no
Variations				
MSEC (64ms tail)	external	1	14	no
MSEC (128ms tail)	external	1	27	no

Table 25. ACOM e energy signal buffer data memory requirements

6.2.10 FG/BG Filter Segment Buffer

Base Configuration	MEM Type	ALIGN	SIZE	VOLATILE
Full filter ECU (32ms, 10ms frame)	External	0	2	no
Variations				
MSEC ECU	External	0	6	no

Table 26. FG/BG filter segment buffer data memory requirements

6.2.11 Search Filter Buffer

Base Configuration	MEM Type	ALIGN	SIZE	VOLATILE
Full filter ECU (32ms, 10ms frame)	internal	0	0	no
Variations				
MSEC ECU (64ms tail)	internal	0	512	no
MSEC ECU (128ms tail)	internal	0	1024	no

Table 27. Search filter buffer data memory requirements

6.2.12 Search Scratch Buffer

Base Configuration	MEM Type	ALIGN	SIZE	VOLATILE
Full filter ECU (32ms, 10ms frame)	external	1	0	yes
Variations				
MSEC ECU (64ms tail)	external	1	128	yes
MSEC ECU (128ms tail)	external	1	256	yes

Table 28. Search scratch buffer data memory requirements

6.2.13 Search Hangover Buffer

Base Configuration	MEM Type	ALIGN	SIZE	VOLATILE
Full filter ECU (32ms, 10ms frame)	external	0	0	no
Variations				
MSEC ECU (64ms tail)	external	1	64	no
MSEC ECU (128ms tail)	external	1	128	no

Table 29. Search hangover buffer data memory requirements

6.3 MIPS

We present here the peak MIPS for the c55x core and multiple segment versus full filter approaches. All values below assume internal memory only and buffer placement as in Table 15 so as to minimize MIPS.

6.3.1 C55x Full Filter Peak MIPS

Refer to Table 30 below for peak MIPS consumption prior to and after convergence.

ecuSendIn() (10ms frame)			
Filter length (taps)	64	128	256
Initial MIPS	3.52	4.55	6.60
Converged MIPS	2.76	3.40	4.68

Table 30. C55x full filter ECU Peak MIPS

6.3.2 C55x MSEC Peak MIPS

Refer to Table 31 below for peak MIPS consumption prior to and after convergence.

ecuSendIn() (10ms frame)		
Filter length (taps)	512	1024
Initial MIPS	10.36	13.42
Converged MIPS	7.15	10.22

Table 31. C55x MSEC ECU Peak MIPS