Implementing a Line-Echo Canceller Using the Block Update and NLMS Algorithms on the TMS320C54x DSP

APPLICATION REPORT: SPRA188

Jelena Nikolic
Associate Technical Staff, DSP Applications
SC Group Technical Marketing
April 1997

TEXAS INSTRUMENTS

TRADEMARKS

Contents

Implementing a Line-Echo Canceller Using the Block Update and NLMS Algorithms on the TMS320C54x DSP

Abstract

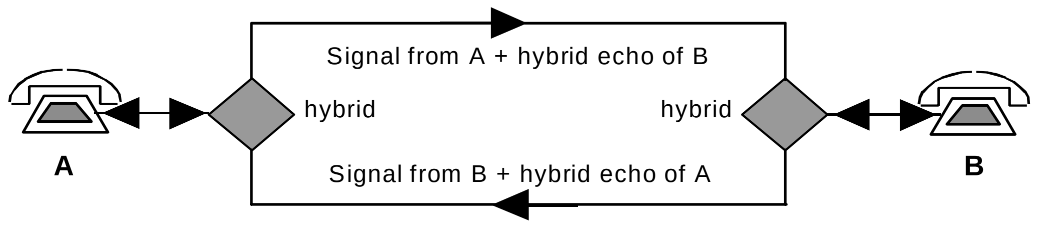
This document describes the implementation of a line echo canceller (LEC) on the TMS320C54x family of 16-bit fixed-point digital signal processors. Two different implementations of the adaptive transversal echo canceller filter are covered: block update and normalized least-mean square (also called stochastic gradient). Full assembler code is given for both examples.

1. Line Echo: Background

The cause of line echo in telephone networks is an analog device called a 2-to-4-wire hybrid. The hybrid is a pair of transformers that use inductive coupling to split a duplex signal into two simplex signals. Such a device is required since only simplex pairs of wire allow for signal processing over long distances, such as equalization and amplification. Hybrids are physically located in the central office.

Owing to electrical leakage of current in the hybrid, a part of the signal energy is reflected back to the source of the signal. This reflection, in conjunction with path delays, causes the talker on either side of the connection to hear an echo of his or her own voice, see Figure 1.

Figure 1. The Source of Signal Echo


Signal from A + hybrid echo of B
hybrid
hybrid
A
B
Signal from B + hybrid echo of A

The audio effect of the leakage in the hybrid depends on the distance between the talker and the hybrid of the other party, as well as on the magnitude of the leakage current. If the path delay is short, the talker typically is not able to distinguish line echo from side tone, which is always present in telephones. However, if the delay exceeds a few tens of milliseconds (which might be the case in long distance or satellite communications), the echo is quite disturbing to the talker. The other parameter that determines the impact of the line echo is the hybrid loss; i.e. what portion of the transmitted signal is reflected back. Typically, it is assumed that the loss is at least 6dB.

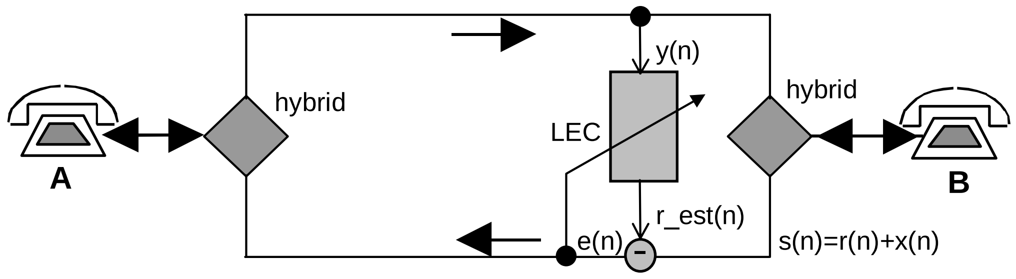
Early techniques of dealing with the line echo problem evolved around echo suppression, i.e. the return path from party B to party A was physically disconnected when it detected that party A was the speaker and party B, the listener. These techniques performed poorly in the case of double talk and resulted in chopped audio signals.

The technique of dealing with the line echo described in this report is based on a prediction filter which, given a reference signal, can predict what the echo will be and then subtract it from the signal which suffers from line echo.

2. Echo Cancellation: Theory and Algorithms

A Line Echo Canceller (LEC) is placed in the network as shown in Figure 2. It is important that the cancellation is performed as close to the source of echo (in this case, hybrid B) as possible. Otherwise, the transversal filter has to match the impulse response of a system consisting of the hybrid *and* the delay path, which increases the number of required filter taps.

Figure 2. Echo Canceller Operation



An echo canceller is typically a transversal filter with the far-end speech signal, *y(n)*, as the reference input and a near-end input, *s(n)*. This signal consists of two components: the echo of the reference signal, *r(n)*, and the near-end speech signal coming from party B, *x(n)*. The output of the filter is estimated echo, *r_est(n)*. The estimated echo is subtracted from the near-end signal, *s(n).* The difference, called the error signal, *e(n) = s(n)–r_est(n),* is fed back into the echo canceller and used to adapt the coefficients of the transversal filter. The objective of the coefficient adaptation process is to minimize the average of the Mean-Squared Error (MSE) between the actual echo and the echo estimate. Mathematical analysis shows that this will be achieved if the coefficients of the transversal filter, a(k), are adapted according to the following equation:

$$a_k(i+1) = a_k(i) + 2\beta E[e(i)y(i-k)] \qquad \text{(Equation 1)}$$

In practice, the expectation operator is substituted with a short-term average over M samples, giving:

$$a_k(i+1) = a_k(i) + 2\beta \frac{1}{M}\sum_{m=0}^{M-1} e(i-m)y(i-m-k) \qquad \text{(Equation 2)}$$

With M=1, we obtain the so-called Least-Mean-Square (LMS) algorithm, also known as stochastic gradient.

An alternative approach is to update coefficients every M samples rather than every sample. In this case, the coefficient update formula would be:

$$a_k(i+M+1) = a_k(i) + 2\beta \sum_{m=0}^{M-1} e(i+M-m)y(i+M-m-k)$$

(Equation 3)

The parameter beta (step size or loop gain) determines the convergence performance of the adaptive filter. It is chosen to be a trade-off between rapid convergence and asymptotic cancellation error. Large beta contributes to rapid convergence, whereas small beta results in a smaller asymptotic error. However, if the loop gain is too large, the coefficients may diverge.

The other factor that influences convergence, is the power of the input signal. For low-power signals, the algorithm will converge slowly. The solution to this problem is to adapt the step size in such a manner that it reflects the changes of input power, i.e. normalizing it by the average power of the input signal:

$$2\beta(i) = \frac{\beta_1}{P_y(i)}$$

(Equation 4)

The second algorithm that is necessarily encountered in a line echo canceller implementation is near-end speech detection. The above derivations are valid only if the near-end input signal, *s(n)* is truly the reference signal which was been passed through a network, thus being subject to path delay and hybrid reflections. However, in reality, the echo of *y(n)* is only a portion of the near-end input signal *s(n)*. The other component is the speech signal coming from the other party in the conversation. This adaptation is only valid if the near-end speech component is not present. Otherwise, the error signal will have the following form:

$$error(i) = echo(i) - estimated\_echo(i) + near\_end\_speech(i)$$

(Equation 5)

As such, it is not a valid input to the adaptation algorithm. Therefore, the situation in which the input has a component other than the echo of the reference signal has to be detected and the adaptation of the coefficients frozen as long as this situation persists.

The most frequently used detection algorithm is the Geigel algorithm, according to which near-end speech is detected if the following condition is true:

$$|s(i)| \geq \frac{1}{2}\max\left(|y(i)|, |y(i-1)|, \ldots, |y(i-N)|\right) \qquad \text{(Equation 6)}$$

Note that the algorithm assumes at least a 6dB loss through the hybrid, which explains the factor 1/2.

The algorithm becomes more robust if short-term estimates of y(n) and s(n) are used instead of y(n) and s(n), respectively. The short-term power estimates are obtained as follows:

$$\tilde{y}(i+1) = (1-\alpha)\tilde{y}(i) + \alpha|y(i)| \qquad \text{(Equation 7)}$$

This equation represents a low-pass IIR filter.

3. Echo Cancellation: Implementations

As discussed in the previous section, there are a number of ways that a line echo canceller can be implemented on a DSP, based on performance requirements.

In voice telephone networks, we expect to be dealing with signals with significant power level variations. Therefore, an adaptive step size seems to be a requirement. The step size is adapted based on the power of the input signal, as per Equation 7.

A less obvious decision is whether a block-update or an LMS algorithm is more appropriate. It may seem that the block-update algorithm would require less MIPS, since only a portion of the coefficients is updated for every input sample. As will become clear later, this is not necessarily true with the TMS320C54x family as the target platform. The architecture of the 'C54x accelerates adaptive filtering applications with its LMS instruction and a number of parallel instructions.

On the other hand, it may seem that updating all coefficients for every input sample would lead to a better convergence than updating one block at a time. Empirical measurements have shown (see Reference [1]) that under certain conditions, the block-update mode will converge slightly faster.

For these reasons, it was decided to pursue both the block-update and the LMS algorithm implementations. Testing in real network environments will show which algorithm performs better under given conditions.

3.1 Normalized LMS

Based on the discussion in the section on algorithms, we conclude that the following equations need to be realized in a normalized LMS implementation:

$$estimated\_echo(i) = \sum_{k=0}^{N-1} a_k y(i-k) \qquad \text{(Equation 8)}$$

$$a_k(i+1) = a_k(i) + \frac{2\beta_1}{P_y(i)} e(i) y(i-k) \qquad \text{(Equation 9)}$$

where *a(k)* are the coefficients of the adaptive transversal filter, *y(n)* are reference samples, *e(n)* is the error signal, β_1 a fixed loop-gain parameter. N is the number of taps in the transversal filter and *Py(i)* is the power estimate of the reference signal given by following equation:

$$P_y(i) = (average(|y(i)|))^2 \qquad \text{(Equation 10)}$$

For finite precision consideration, the factors in the last term in Equation 9 are regrouped as follows:

$$a_k(i+1) = a_k(i) + \frac{2\beta_1 e(i)}{average(|y(i)|)} \frac{y(i-k)}{average(|y(i)|)} \qquad \text{(Equation 11)}$$

The first factor in the product term is referred to as *n_error_mu* (normalized error multiplied by a fixed step size). The second factor represents a normalized reference sample.

Using the LMS instruction, the convolution (Equation 8) and coefficient adaptation (Equation 11) are implemented in the same loop which represents the kernel code of any adaptive filter on a 'C54x as follows:

```
   STM   #TAPS-2,BRC   ; repeat #TAPS–1 times
                       ; (first iteration is done in the RPTBD
                       ; delay slot).
   SBX   FRCT          ; set fractional mode
   LD    #0,B          ; B = 0
   LD    n_error_mu, T ; T = e(i–1) * beta1 / average(|y(i)|)
                       ; filter_ptr points to a(0)
                       ; n_ref_pt points to normalized reference buffer
                       ; ref_ptr points to y(0) in the reference buffer
   RPTBD $nlms_end–1
   MPY   *n_ref_ptr+0%,A           ; delay slot 1: A = y(i–1)/
                                   ; average(|y(i)|)*T
   LMS   *filter_ptr,*ref_ptr+0%   ; delay slot 2: A = A + a(0)<<16
                                   ; B = B + a(0)*y(i)

   ST    A,*filter_ptr+            ; store out coefficient: a(k–1) = A
|| MPY   *n_ref_ptr+0%,A           ; coef. increment: A = y(i–k–1)/
                                   ; average(|y(i)|) * T
   LMS   *filter_ptr,*ref_ptr+0%   ; new coefficient: A = A + a(k)<<16
                                   ; convolution:   B = B + a(k)*y(i–k)

$nlms_end:
   ST    A,*filter_ptr    ; store out last coefficient: a(TAPS–1) = A
|| SUB   *AR5,A           ; near end signal - estimated echo:
                          ; A = s(0) << 16– B
   STH   A,error_out      ; error_out = A
```

The loop is shown in bold print and takes two cycles per iteration (assuming no more than one external access to zero-wait state memory). Therefore, the total number of instructions per input sample period is about 2*TAPS.

You must maintain a buffer of reference samples normalized by the average power. The size of the buffer is TAPS words. This approach is more MIPS-efficient than calculating the normalized reference on the fly, inside the kernel loop.

In addition to this kernel loop, the operations required for the algorithm are:

1) Calculate long-term-power average and its inverse (IIR and division)

2) Normalize the latest reference sample

3) Calculate the error term, *n_error_mu*.

For implementation details on these routines, refer to the code listing of module **nlms.asm** in Appendix A.

3.2 Normalized Block Update

This implementation is adapted from the TMS320C5x example in Reference[2] and is therefore described in less detail than the NLMS example. The following equations need to be realized in this LEC implementation:

$$estimated\_echo(i) = \sum_{k=0}^{N-1} a_k y(i-k)$$

(Equation 12)

$$a_k(i+M+1) = a_k(i) + 2\frac{\beta_1}{P_y(i)} \sum_{m=0}^{M-1} e(i+M-m)y(i+M-m-k)$$

(Equation 13)

For finite precision and implementation reasons, the summation in Equation 13 will be factorized as follows:

$$a_k(i+M+1) = a_k(i) + 2\frac{\beta_1}{average(|y(i)|)} \sum_{m=0}^{M-1} \frac{e(i+M-m)}{average(|y(i)|)} y(i+M-m-k)$$

(Equation 14)

In each pass, TAPS/BLOCKSIZE coefficients are updated. For efficiency, they will not be consecutive coefficients, but rather coefficients that are spaced BLOCKSIZE taps apart: *a(h)*, *a(h+BLOCKSIZE)*, *a(2h+BLOCKSIZE)*, etc. The variable *h* is a modulo BLOCKSIZE counter that is incremented once every input sample period. By taking this approach, the reference sample buffer will be addressed in a linearly increasing fashion, minimizing the overhead of pointer management.

The kernel loop that implements the convolution is a standard FIR implementation:

```
STM    #FILTER,filter_ptr         ; points to a(0)
LD     #0,A
STM    #0FFFFh,AR0                ; index = –1
RPT    #(TAPS–2)                  ; repeat main loop TAPS-1 timer
MAC    *ref_ptr+0%,*filter_ptr+,A ; A = A + a(k)*y(i–k)
MACR   *ref_ptr+0%,*filter_ptr+,A ; last multiply-accumulate: add
                                  ; rounding offset.
STH    A,1,est_echo               ; store the result out.
```

Coefficients are adapted in the following kernel loop:

```
  RPTBD   $block_end–1
  STM     #N_ERROR, n_error_ptr

  RPTZ    A,#(BLOCKSIZE–2)
  MAC     *ref_ptr+0%,*n_error_ptr+,A   ; calculate coefficient
                                        ; increment:
  MACR    *ref_ptr+0%,*n_error_ptr,A    ; A = sum(error(i–m)*y(i–m–k))
                                        ; m = 0,...,BLOCKSIZE–1

  MPYA    IABSY                         ; B = AH / average(|y(i)|)
  LD      *+filter_ptr(16),16,A         ; AH = ak(i)
  SFTA    B,#(15+GAIN)                  ; B = B * beta1 (beta1 is a
                                        ; power of 2)
  ADD     B,A                           ; A = A+B
  STH     A, *filter_ptr                ; ak(i+1) = AH
  STM     #N_ERROR, n_error_ptr         ; rewind the pointer to
                                        ; normalized error.
$block_end:
  RSBX    FRCT
```

This routine consists of two nested loops. The inner loop calculates the coefficient increment and takes BLOCKSIZE cycles, and the outer loop consists of loading, updating and storing the coefficients. These instructions take about 8 cycles per iteration, plus inner loop overhead. The total number of instructions per input sample period is approximately (10+BLOCKSIZE)*TAPS/BLOCKSIZE.

In addition to the kernel code, the following routines are required:

(1) Calculate the long term power average of the reference signal and its inverse (IIR and division).

(2) Normalize error signals and maintain the error signal buffer. The size of this buffer is BLOCKSIZE. Since the sizes of the reference sample buffer and the normalized error buffer are different and required in the same loop, one of the buffers has to be linear. The overhead to maintain a linear buffer for normalized errors means applying the DELAY operation BLOCKSIZE times, for every input sample, and rewinding the pointer for every coefficient (STM instruction on the outer adaptation loop).

3.3 Near-End Speech Detection

This portion of the algorithm acts as a control input to the coefficient adaptation algorithm. It basically decides when the coefficient update process should be frozen. The two algorithms that are implemented are Geigel (for the NLMS version) and modified Geigel (for the block-update version). In theory, there are no restrictions on mixing and matching the adaptive filter algorithms with speech detection algorithms.

3.4 Geigel Speech Detection Algorithm

The following code implements Equation 6:

```
LD     *ref_ptr+%,16,A     ; y(0) -> AH
ABS    A                   ; |y(0)| -> AH
SUB    max_m,16,A, B       ; |y(0)| – max_m -> BH
NOP                        ; delay slots for
                           ; conditional execution
NOP
XC     1,BGT               ; refers to B from 2 cycles ago
STH    A,max_m             ; if |y(0)| > max_m, update max_m
                           ; ref_ptr points to y(0)
```

In addition to this routine, a check must be performed to determine whether the previous maximum is equal to the sample that is removed from the buffer the next time around. In this case, the entire buffer needs to be searched through to find the new maximum.

3.4.1 Modified Geigel Speech Detection Algorithm

This speech detection algorithm is a modified version of Equation 6, where the absolute values of samples are substituted with their long-term filtered version, obtained by applying the IIR filter described in Equation 7. In order to minimize the memory requirement for such an implementation, rather than keeping the filtered versions of TAPS previous samples, the information is maintained within (TAPS/BLOCKSIZE+1) partial maxima. The updating algorithm is described in Reference [1], pp.431. The TMS320C54x version is a straight translation from the TMS320C2x/C5x implementation.

3.5 Additional Control Algorithms

In addition to the two main algorithms typically seen in LEC implementations, the performance of the echo canceller in real-world conditions might be enhanced using some of the following algorithms. An easy way to incorporate these routines into the rest of the code is to maintain flags that could be set/reset by control algorithms and checked by signal processing routines.

3.5.1 Cutoff Detection

When the far-end speaker is silent and the power of the reference input falls below a set threshold, the coefficients should not be updated. The parameter associated with this algorithm is the cutoff level.

3.5.2 Echo Suppression

When the error at the output of the LEC (*s(n) – est_echo(n)*) falls below a certain threshold, instead of outputting the small error, the output will be set to zero. The threshold value could be fixed or adaptive (in this implementation, it is fixed).

3.5.3 Summary of Parameters

Table 1 summarizes the parameters which govern the processing requirements as well as the performance of an LEC. When setting the values of a parameter, keep in mind that there is usually a trade-off to be made between one aspect of performance or another, based on the weight which the individual requirements carry.

Table 1. Parameters Governing LEC Performance

Parameter	Usage	Description	Value
TAPS	Length of the adaptive filter	A larger number of taps allows the LEC to compensate for larger tail-circuit path delays, but increases processing and memory requirements.	48 to 128
BLOCKSIZE	Block of samples used to update each filter coefficient	A large block means less frequent update of each particular coefficient, therefore less processing but also diminished capability of tracking fast changes.	16
GAIN	Fixed step size: beta1 = 2^GAIN	Large beta means faster convergence and larger asymptotic convergence error.	(-9) to (-11)
CUTOFF	When the far-end speaker is silent, freeze tap update.	Prevents the normalized step size from becoming excessively large when the signal input power is low.	-48dB
THRESHOLD	Controls echo suppression	When the output of EC is sufficiently low, set the output to 0	-24dB
HANG0	Hangover counter, control near-end speech declaration	Declares near-end speech a certain amount of time after the last sample for which the Geigel inequality was satisfied, to filter short-term power peaks.	75 ms
LTAU	IIR filter pole	Controls low-pass filtering of input signal power, i.e., the length of long-term average.	2^ (-7)

4. Speed and Memory Requirements

Table 2 provides estimates of cycle counts and memory requirements for the kernel code of the two different implementations of the LEC. The numbers are given relative to the number of taps (N) and the block size (M).

The NLMS version seems to require less instruction cycles and code space. For example, if N=128 and M=16, the block update version filter adaptation kernel requires 77 instruction words and 206 instruction cycles, whereas the NLMS version requires 60 instruction words and 326 instruction cycles.

Table 2. LEC Cycle Counts & Memory Requirements

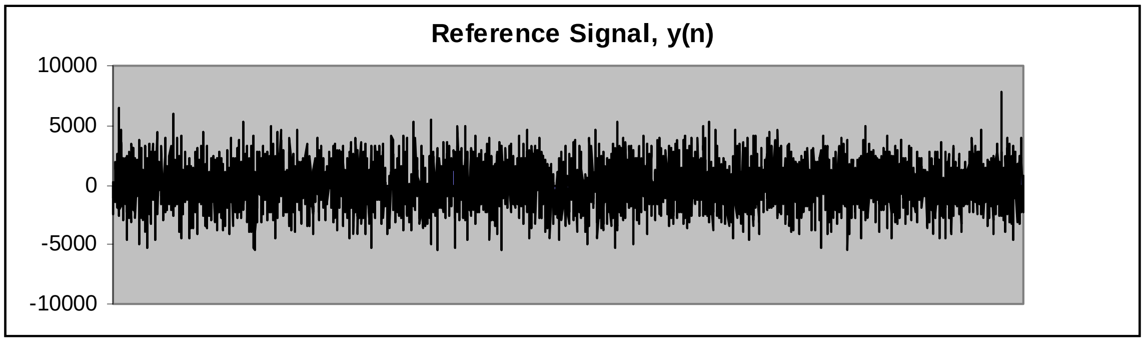
Module	Description	Code Size	Cycles	Data	Size
BLOCK UPDATE					
init.asm	processor init + RAM clearing	70	2*(N+M)	reference buffer	N + M
detect_b.asm	speech detection (robust Geigel)	94	70 (+ 5*N/M)†	normalized error buffer	M
fir.asm	convolution	17	N+25	filter coefficients	N
update.asm	adaptation	60	2*M+N/M(M+10)+15	partial maxima	N/M
				other	26
NLMS					
init.asm	processor init + RAM clearing	70	3*N	filter coefficients	N
detect.asm	speech detection (Geigel)	38	31 (+3*N)†	reference buffer	N
nlms.asm	convolution and adaptation	60	2*N+70	normalized reference buffer	N
				other	22

NOTE:

† In the cycle number column for detect_b.asm and detect.asm, the number in parentheses is added whenever the location of the previous maximum is the last sample in the buffer, making it necessary for the maximum to be recalculated by searching through the buffer.

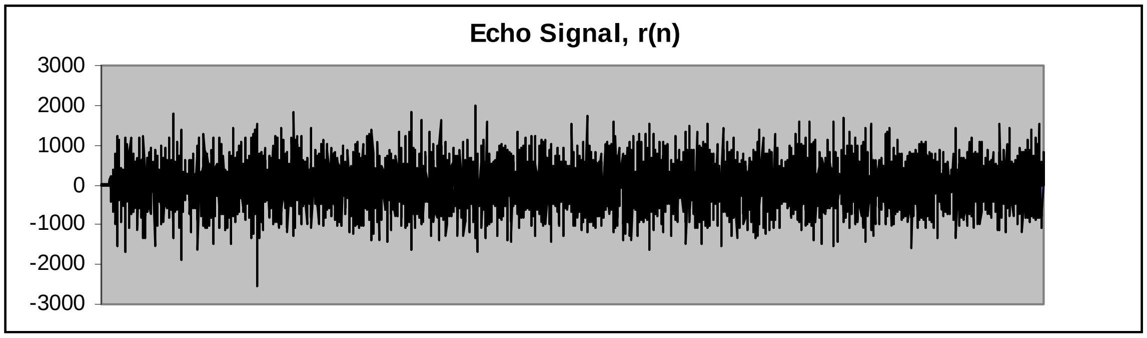
5. Testing and Verification

The functionality of the LEC, with the emphasis on the transversal filter adaptation, was tested and verified on TI's TMS320C5x DSP Starter Kit (DSKplus) development platform. Since this platform does not have the capability to connect to a telephone network, verifications were performed using Code Explorer's File I/O capabilities. Test data from two files, representing a reference signal, *y(n)*, and the corresponding echo signal, *r(n)*, were streamed onto the DSP via a debugger probe point. After DSP processing, the results were streamed out to a PC file. Obviously, this approach was not meant to satisfy real-time constraints.



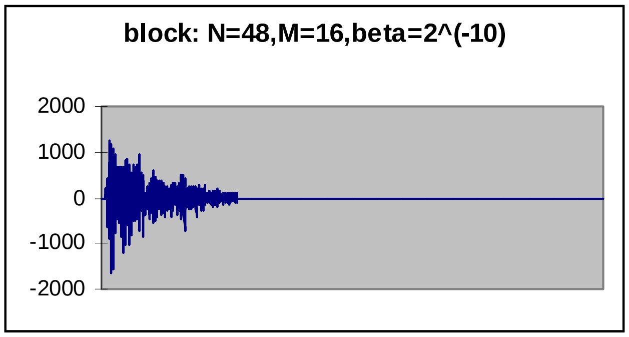
The reference input signal represents 4000 samples (0.5 sec) of band-limited white noise. The echo signal is the reference signal passed through a digital filter with 32 samples flat delay, a total of 50 taps and 10dB loss. This filter simulates the response of a hybrid in a telephone network.

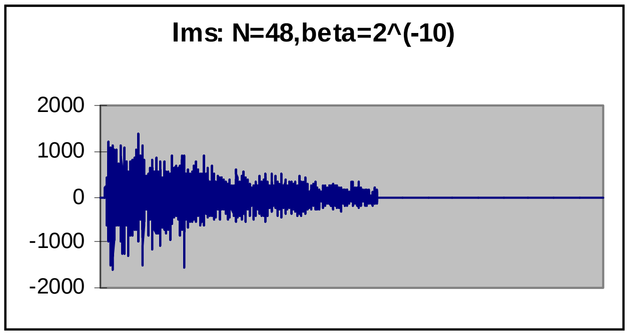
The error outputs of the LEC were recorded for the same input signals, but different values of design parameters: tap length N and step size beta, for each of the two implemented algorithms. The block update version of the adaptation algorithm seems to lead to a faster convergence in all cases. This is in accordance with simulation results reported in Reference [1] for the case of noise input.

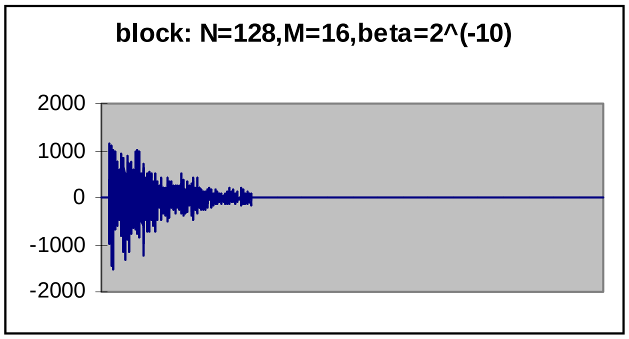


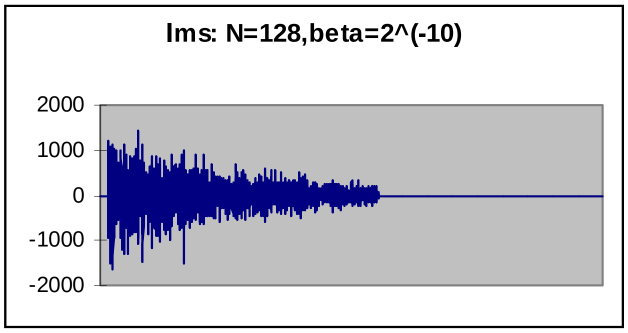
The recorded outputs show that the convergence rate increases with increasing step size. This result agrees with Reference [1]. Note that if the step-size becomes too large, the filter might become unstable.

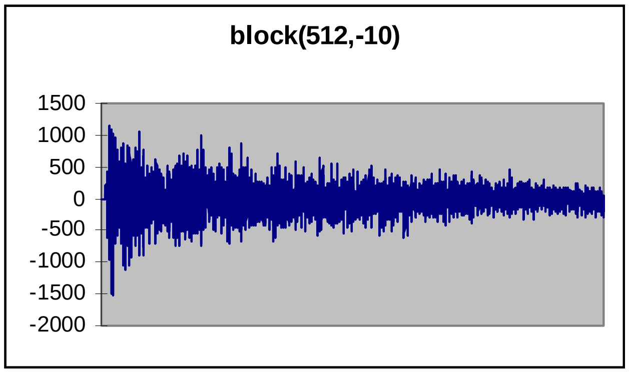
Both implementations were tested with tap length N = 48, 128 and 256. It remained inconclusive how the tap length influences the LEC convergence performance. In real telephone networks, this design parameter will determine the maximum tail circuit length that the LEC can tolerate.

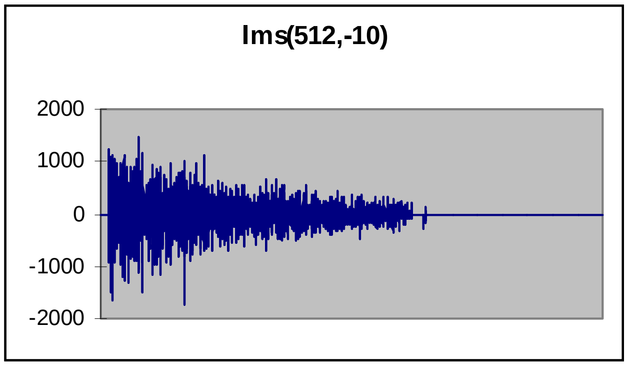


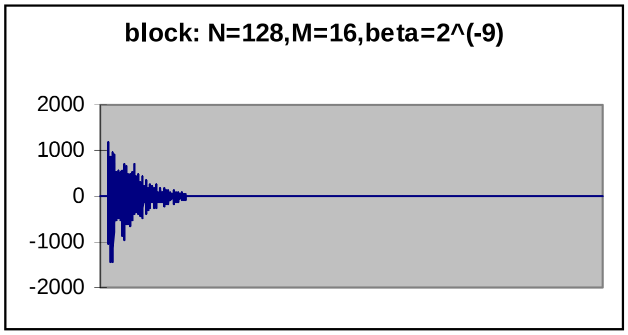


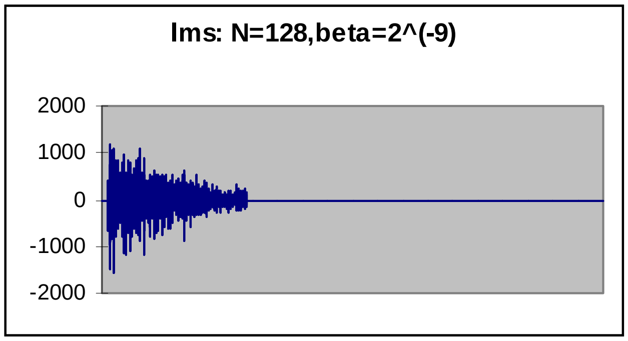


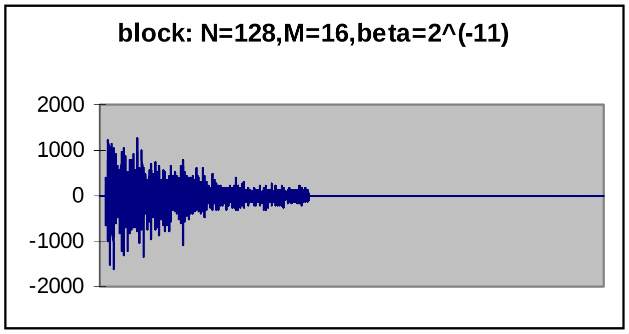

lms: N=128,beta=2^(-10)
2000
1000
0
-1000
-2000

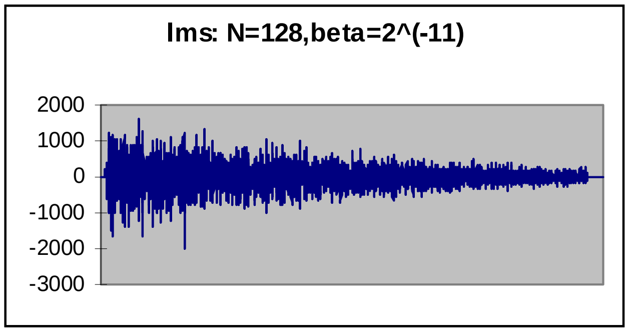

block(512,-10)
1500
1000
500
0
-500
-1000
-1500
-2000


lms(512,-10)
2000
1000
0
-1000
-2000


block: N=128,M=16,beta=2^(-9)
2000
1000
0
-1000
-2000


lms: N=128,beta=2^(-9)
2000
1000
0
-1000
-2000


block: N=128,M=16,beta=2^(-11)
2000
1000
0
-1000
-2000


lms: N=128,beta=2^(-11)
2000
1000
0
-1000
-2000
-3000

6. Summary

The implementation of two different adaptive filter algorithms on the TMS320C54x family of processors has shown that this family of processors is slightly better suited to the LMS approach owing to the dedicated LMS instruction. This results in smaller code size and less instruction cycles.

In terms of the performance of the two different algorithms, for white noise input with no near-end speech, the block update version seems to perform better. However, the ultimate judgement should be made after testing in a real telephone network environment and against industry standards which is outside the scope of this application report.

7. References

Reference [1]

Digital Voice Echo Canceller with the TMS32020, Digital Signal Processing Applications with the TMS320 Family, Volume 1, Texas Instruments, 1986, pp. 415-454, Literature number: SPRA012.

Reference [2]

Digital Voice Echo Canceller Implementation on the TMS320C5x, Telecommunications Applications with the TMS320C5x DSPs, Texas Instruments, 1994, p. 189-202, Literature number: SPRA033.

Appendix A. TMS320C54x LEC code listing

```
*******************************************************************
* Application: C54x LEC                                           *
* File name:   main.asm                                           *
* Description: Variable declarations + memory allocation          *
*                                                                 *
* Author:      Jelena Nikolic, Associate Technical Staff          *
* Date:        January 97                                         *
*******************************************************************

               .title   "C54x LEC: main.asm"
               .mmregs

               .include "echoequ.inc"

               .global  error_out, OUTPUT, FILTER, REFERENCE, INPUT_B, OUTPUT_B
               .global  THRES,S0,Y0, max_m, HANGT
               .global  HCNTR, ABSS0, ABSS0F, ABSE0, ABSOUT, AELSBS, ABSY0, ABSY
               .global  AYLSBS, IABSY, CUTOFF, ABSY0F, control_flags, input_sample
       .if mode=NLMS
               .global  N_REFERENCE, n_error_mu
       .elseif mode=BLOCK
               .global  N_ERROR, M0, H, est_echo, y0_ptr
       .endif


error_out      .usect   "daram",1                    ; canceller output (before supression)
OUTPUT         .usect   "daram",1                    ; canceller output (after supression)
S0             .usect   "daram",1                    ; input near-end sample
Y0             .usect   "daram",1                    ; input reference sample
max_m          .usect   "daram",1                    ; short-term reference maximum,
                                                     ; used for near-end speech detection.

THRES          .usect   "daram",1                    ; residual output suppression threshold
control_flags  .usect   "daram",1                    ; echo canceller control bits
HANGT          .usect   "daram",1                    ; hangover counter reset value
HCNTR          .usect   "daram",1                    ; hangover counter
ABSS0          .usect   "daram",1                    ; abs(S0)
ABSS0F         .usect   "daram",1                    ; short tau LPF 2*abs(S0)
ABSE0          .usect   "daram",1                    ; abs(output)
ABSOUT         .usect   "daram",1                    ; long tau LPF abs(output) MSB's
AELSBS         .usect   "daram",1                    ; long tau LPF abs(output) LSB's
ABSY0          .usect   "daram",1                    ; abs(Y0)
ABSY           .usect   "daram",1                    ; long tau LPF abs(Y0) MSB's
AYLSBS         .usect   "daram",1                    ; long tau LPF abs(Y0) LSB's
IABSY          .usect   "daram",1                    ; 1/ABSY
CUTOFF         .usect   "daram",1                    ; ABSY cutoff level for no update
ABSY0F         .usect   "daram",1                    ; short tau LPF abs(Y0)

FILTER         .usect   "filter",TAPS                ; FIR filter coefficient
REFERENCE      .usect   "ref",TAPS+BLOCKSIZE         ; reference samples
       .if mode=NLMS
n_error_mu     .usect   "daram",1                    ; normalized error * step size
N_REFERENCE    .usect   "n_ref",TAPS                 ; normalized reference samples
       .elseif mode=BLOCK
y0_ptr         .usect   "daram",1                    ; pointer to y(0)
H              .usect   "daram",1                    ; modulo 16 counter.
est_echo       .usect   "daram",1                    ; output of the tranvversal filter
M0             .usect   "daram",TAPS/BLOCKSIZE+1     ; partial reference maxima
N_ERROR        .usect   "n_error",BLOCKSIZE          ; normalized reference samples
       .endif

OUTPUT_B       .usect   "o_buffer", TAPS             ; for DSKplus demo
INPUT_B        .usect   "i_buffer",TAPS              ; for DSKplus demo
```

```
*******************************************************************
* Application: C54x LEC                                           *
* File name:   init.asm                                           *
* Description: Processor and variable initialization              *
*                                                                 *
* Author:      Jelena Nikolic, Associate Technical Staff          *
* Date:        January 97                                         *
*******************************************************************

               .title   "C54x LEC: init.asm"
               .mmregs
               .include "echoequ.inc"

               .global  start, CUTOFF, IABSY, HANGT, ABSY0F,REFERENCE, FILTER,S0
               .global  max_m, THRES,A0, ABSY, control_flags, y_ptr, input_sample
               .global  INPUT_B, OUTPUT_B

       .if mode=NLMS
               .global  N_REFERENCE
       .elseif mode=BLOCK
               .global  N_ERROR, y0_ptr
       .endif

               .text
start:
               STM      #01a0h,PMST                 ; setup for DSKplus environment
               STM      #0FFAh,SP
               SSBX     OVM                         ; Enable overflow saturation mode.
;
;      Setup ST0
;

               SSBX     SXM                         ; 1 -> SXM   sign extension enabled
               RSBX     TC                          ; 0 -> TC    TC flag cleared
               RSBX     C                           ; 0 -> C     carry cleared
               SSBX     XF                          ; probably don't need this.

;
;      Initialize data pointers and setup circular buffers.
;

               STM      #FILTER,filter_ptr
               STM      #REFERENCE,ref_ptr
       .if mode=NLMS
               STM      #N_REFERENCE,n_ref_ptr
       .elseif mode=BLOCK
               MVMD     ref_ptr,y0_ptr
       .endif
               STM      #(TAPS+BLOCKSIZE),BK
               STM      #S0,s0_ptr

               STM      #INPUT_B,AR6
               STM      #OUTPUT_B,AR7

;
;      Setup Software-Programmable Wait-State Generators
;

K_SWWSR_IO     .set     2000h                       ; set the I/O space

K_BNKCMP       .set     0000b << 12                 ; bank size = 64K
K_PS_DS        .set     0b << 11
K_BSCR_RESR    .set     000000000b <<2              ; reserved space
K_BH           .set     0b << 1                     ; BH = 0 at reset
K_EXIO         .set     0b << 0                     ; EXIO = 0 at reset

K_BSCR         .set     K_BNKCMP|K_PS_DS|K_BSCR_RESR|K_BH|K_EXIO

               STM      #K_BSCR, BSCR               ; 0 wait states for BANK switch
               STM      #K_SWWSR_IO, SWWSR          ; 2 wait states for I/O operations
```

```
;
;   Clear Internal RAM Blocks
;
                STM      #FILTER, AR1
                RPTZ     A, #TAPS
                STL      A, *AR1+

                STM      #REFERENCE, AR1
                RPTZ     A, #(TAPS+BLOCKSIZE)
                STL      A, *AR1+

        .if mode=NLMS
                STM      #N_REFERENCE, AR1
                RPTZ     A, #TAPS
                STL      A, *AR1+
        .elseif mode=BLOCK
                STM      #N_ERROR, AR1
                RPTZ     A, #BLOCKSIZE
                STL      A, *AR1+
        .endif

                LD       #4,DP                         ; variables start at 0x200

                LD       #THRES0, A
                STL      A, THRES
                LD       #HANGT0, A
                STL      A, HANGT
                LD       #0400h, A
                STL      A, ABSY
                LD       #020h, A
                STL      A, IABSY
                LD       #CUTOF0, A
                STL      A, CUTOFF
                LD       #0400h, A
                STL      A, ABSY0F
                LD       #(1<<ECAN|1<<ESUP|1<<UNFREEZE),A
                                                       ; 1 -> ECAN     Enable cancellation
                                                       ; 1 -> ESUP     Enable resid. suppress.
                                                       ; 1 -> UNFREEZE Enable coef. adaptation
                STL      A,control_flags

                LD       #no_nesp_level, A

                STM      #200h,IMR                     ; enable hpi (for DSK debugger)

                BD       input_sample
                STL      A, max_m
                RSBX     INTM

                .end
```

```
********************************************************************
* Application: C54x LEC                                            *
* File name:   input.asm                                           *
* Description: Get new samples:                                    *
*              reference input              : y(0)                 *
*              near-end input : s(0)                               *
*                                                                  *
* Author:      Jelena Nikolic, Associate Technical Staff           *
* Date:        January 97                                          *
********************************************************************
               .title   "INPUT.ASM: 'C54x Echo Canceller - Sample input module"
               .mmregs
               .include "echoequ.inc"

               .global  input_sample, detect_near_speech, Y0, S0
       .if mode=BLOCK
               .global  y0_ptr
       .endif

               .text

input_sample:
       .if mode=BLOCK
               MVDM     y0_ptr,ref_ptr
               MAR      *ref_ptr+%                  ; increment pointer into ref. buffer
               MVMD     ref_ptr,y0_ptr              ; remember location of y(0)
       .endif


;
;      For DSKplus demo, samples are injected through File IO.
;      Set probe point at label input_sample
;

               ;PORTR   #0100h,Y0                   ; get reference sample
               ;PORTR   #0200h,S0                   ; get near-end sample

               BD       detect_near_speech
               LD       Y0,A
               STL      A,*ref_ptr          ; store new sample into reference buffer.
```

```
*********************************************************************************************
* Application: C54x LEC                                                    *
* File name:   input.asm                                                   *
* Description: Suppress the error signal if necessary and output sample    *
*              (1) Calculate long-term power estimate of  u(n):            *
*                  power(u(i+1)) = (1-alpha) power(u(i)) + alhpa*|u(i)|    *
*                  alpha = 2^(-7)                                          *
*              (2) output a sample as follows:                             *
*                  if (canceller disabled)                                 *
*                    output = s(0)                                         *
*                  else                                                    *
*                    if (supressor disabled)                               *
*                      output = error_out                                  *
*                  else                                                    *
*                    if (|error_out|/power(y) < threshold)                 *
*                      output = 0                                          *
*                  else                                                    *
*                    output = error_out                                    *
*                                                                          *
* Author:      Jelena Nikolic, Associate Technical Staff                   *
* Date:        January 97                                                  *
*********************************************************************************************

               .title   "C54x LEC: output.asm"
               .mmregs
               .include "echoequ.inc"

               .global  OUTPUT, output_sample, input_sample, INPUT_B,OUTPUT_B
               .global  control_flags, error_out,S0
               .global  THRES, ABSOUT, IABSY, HCNTR, ABSE0

               .text

output_sample:

echo_suppressor:

;
;       Update long-time output power estimate (ABSOUT)
;

               DLD      ABSOUT,A                    ; double precision load.
               SUB      ABSOUT, LTAU, A             ; ACC-ABSOUT*2**LTAU
               ADD      ABSE0, LTAU, A              ; ACC +ABSE0*2**LTAU
               DST      A, ABSOUT                   ; double precision store.

;
;       If (HCNTR)>0 then NEAR-END SPEECH
;

               LD       HCNTR, A
               NOP
               NOP
               BC       $no_suppress, AGT

;
;       Test is the suppressor is enabled by user.
;

               BITF     control_flags, #1<<(15-(15-ESUP))
               NOP
               NOP
               BC       $no_suppress, NTC

;
;       Suppression allowed; determine if required.
;

               LD       IABSY, T                    ; 1/|Y0|*|error_out| = |error_out/Y0|.
               MPY      ABSOUT, A
               SUB      THRES, A
               LD       #0, B
               NOP                                 ; latency nop
               XC       1, AGT                     ; if |error_out/Y0| > threshold, then
                                                   ; OUTPUT = error_out
                                                   ; else, A will contain 0 since
                                                   ; LD error_out, A will not execute.

$no_suppress:
;
;       if NEAR-END SPEECH or SUPPRESSOR DISABLED, copy error_out to
;       OUTPUT and exit routine. (delayed)
;
               LD       error_out, B

canceller_on_off:

               BITF     control_flags, #1<<(15-(15-ECAN))

               STL      B, OUTPUT                  ; (previous routine's OUTPUT modify)
               NOP                                 ; latency nop inserted ny translator.
               XC       1, NTC                     ; if canceller disabled (/ECAN), then
               LD       S0, B                      ; OUTPUT = S0 (s(n): near-end RX
signal)
               STL      B, OUTPUT

;
;       Output sample.
;

;
;       Manage i & o buffers for DSKplus demo
;

               LD       S0,A
               STL      A,*AR6+

               LDM      AR6,A
               SUB      #(INPUT_B+TAPS),A
               LD       OUTPUT,B
               STL      B,*AR7+

               BC       continue, ALEQ
               STM      #INPUT_B,AR6               ; in DSKplus demo: set probe point here
               STM      #OUTPUT_B,AR7              ; and connect it to graphical display.

continue:
               B        input_sample
               ;PORTW   OUTPUT, #300h              ; output to file in simulation


               ; ********
```

```
*****************************************************************************
; File: VECTORS.ASM -> Vector Table for the 'C54x DSKplus          10.Jul.96
;
; ***************************************************************************
; The vectors in this table can be configured for processing external and
; internal software interrupts. The DSKplus debugger uses four interrupt
; vectors. These are RESET, TRAP2, INT2, and HPIINT.
;   *  DO NOT MODIFY THESE FOUR VECTORS IF YOU PLAN TO USE THE DEBUGGER  *
;
; All other vector locations are free to use. When programming always be sure
; the HPIINT bit is unmasked (IMR=200h) to allow the communications kernel and
; host PC interact. INT2 should normally be masked (IMR(bit 2) = 0) so that the
; DSP will not interrupt itself during a HINT. HINT is tied to INT2 externally.
;
               .title   "Vector Table"
               .mmregs
               .global  input_sample
               .sect    "vectors"

reset          B        #80h                ; 00; RESET  *DO NOT MODIFY IF USING DEBUGGER*
               nop
               nop
nmi            RETE                         ; 04; non-maskable external interrupt
               nop
               nop
               nop
trap2          B        #88h                ; 08; trap2  *DO NOT MODIFY IF USING DEBUGGER*
               nop
               nop
               .space   52*16               ; 0C-3F: vectors for software interrupts 18-30
int0           RETE                         ; 40; external interrupt int0
               nop
               nop
               nop
int1           RETE                         ; 44; external interrupt int1
               nop
               nop
               nop
int2           RETE                         ; 48; external interrupt int2
               nop
               nop
               nop
tint           RETE                         ; 4C; internal timer interrupt
               nop
               nop
               nop
brint          RETE                         ; 50; BSP receive interrupt
               nop
               nop
               nop
bxint          RETE                         ; 54; BSP transmit interrupt
               nop
               nop
               nop
trint          RETE                         ; 58; TDM receive interrupt
               nop
               nop
               nop
txint          RETE                         ; 5C; TDM transmit interrupt
               nop
               nop
int3           RETE                         ; 60; external interrupt int3
               nop
               nop
               nop
hpiint         B        #0e4h               ; 64; HPIint *DO NOT MODIFY IF USING DEBUGGER*
               nop
               nop
               .space   24*16               ; 68-7F; reserved area
```

```
*******************************************************************************************
* Application: C54x LEC                                                                   *
* File name:   detect.asm                                                                 *
* Description: Determine if speech is present at the near-end. If yes,                    *
*              set a flag ( required by subsequent modules)                               *
*              Speech detection is performed using the Geigel algorithm                   *
*              if                                                                         *
*                |s(i)| >= 1/2 max(|y(i)|,|y(i-1)|, |y(i-TAPS)| )                         *
*              then                                                                       *
*                start hangover counter                                                   *
*                declare near end speech                                                  *
*              else                                                                       *
*                if hangover counter > 0                                                  *
*                  decrement hangover counter                                             *
*                  declare near end speech                                                *
*                                                                                         *
* Author:      Jelena Nikolic, Associate Technical Staff                                  *
* Date:        January 97                                                                 *
*******************************************************************************************

               .title   " C54x LEC: detect.asm"
               .mmregs
               .include "echoequ.inc"

               .global  filter_and_adapt, detect_near_speech
               .global  HANGT, max_m, HCNTR, control_flags

               .text

detect_near_speech:
;
;       Update max_m by comparing it to |y(0)|
;
               LD       *ref_ptr+%,16,A               ; y(0) -> AH
               ABS      A                             ; |y(0)| -> AH
               SUB      max_m,16,A, B                 ; |y(0)| - max_m -> BH

               NOP
               NOP
               XC       1,BGT                         ; refers to B from 2 cycles ago.
               STH      A,max_m                       ; if |y(0)| > max_m, update max_m
                                                      ; ref_ptr points to y(0)

;
;       If max_m = (y(i-(N-1)), need to recalculate max_m since the sample
;       will be outside the sliding window next time around.
;
               LD       *ref_ptr-%,B                  ; ref_ptr points to y(0) again.
               ABS      B
               SUB      max_m, B
               NOP
               NOP
               BC       $continue,BNEQ

               STM      #(TAPS-2),BRC                 ; exclude y(i-(N-1))
               LD       #0,B
               RPTB     $end_update_max - 1
               LD       *ref_ptr-%,A                  ; y(i-k) -> A
               ABS      A                             ; |y(i-k)| -> A
               MAX      B                             ; max(A,B) -> B

$end_update_max:
               STL      B,max_m
               MAR      *ref_ptr-%                    ; ref+ptr points to y(0) again.

$continue:
;
;
;       If nearend signal  is greater than max_m then declare NEAREND SPEECH DETECTED:
;       load hangover counter (HCNTR) and freeze tap updates.
```

```
;
               LD       *s0_ptr, A
               ABS      A
               SUB      max_m, A
               BC       $check_hang, ALEQ
               LD       HANGT, A                    ; load hangover counter
               STL      A, HCNTR
               ANDM     #~(1<<NONESP), control_flags ; declare near-end speech

$check_hang:
;
;       Update hangover counter (HCNTR) if nonzero,
;       otherwise undeclare near-end speech.
;

               LD       HCNTR, A
               SUB      #1,A
               XC       1,AGEQ
               STL      A, HCNTR                    ; if HCNTR<>0, then HCNTR--
               XC       2,ALT                       ; is counter was zero,
               BD       filter_and_adapt
               ORM      #(1<<NONESP), control_flags  ; un-declare NE speech
```

```
*******************************************************************************************
* Application: C54x LEC                                                                   *
* File name:   detect_blockasm                                                            *
* Description: Determine if speech is present at the near-end. If yes,                    *
*              set a flag ( required by subsequent modules)                               *
*              Speech detection is performed using the modified                           *
*      Geigel algorithm                                                                   *
*              (1)Calculate short-term power estimates:                                   *
*              ~y(i+1) = (1-alpha)~y(i) + alpha*|y(i)|                                    *
*              ~s(i+1) = (1-alpha)~y(i) + alpha*|y(i)|                                    *
*              alpha = 2^(-5)  -> STAU = 11                                               *
*                                                                                         *
*              (2) Speech detection algorithm:                                            *
*              if                                                                         *
*                ~s(i) >= 1/2 max(~y(i),~y(i-1), ~y(i-TAPS) )                             *
*              then                                                                       *
*                start hangover counter                                                   *
*                declare near end speech                                                  *
*              else                                                                       *
*                if hangover counter > 0                                                  *
*                  decrement hangover counter                                             *
*                  declare near end speech                                                *
*                                                                                         *
* Author:      Jelena Nikolic, Associate Technical Staff                                  *
*              (translated from C5x)                                                      *
* Date:        January 97                                                                 *
*******************************************************************************************

               .title   "C54x LEC: detect_b.asm"
               .mmregs
               .include "echoequ.inc"

               .global  detect_near_speech,estimate_echo
               .global  ABSS0, ABSY0, CUTOFF, H, ABSS0F, HANGT
               .global  ABSY0F, max_m, HCNTR, ABSY, M0
               .global  control_flags,Y0,S0

               .text

detect_near_speech:

               LD       Y0,A
               ABS      A
               STL      A,ABSY0                      ; absolute value of y(0)

               LD       S0,A
               SFTA     A,15
               ABS      A
               STH      A,ABSS0                      ; absolute value
                                                     ; ABSSO = |s(n)/2| to compensate
                                                     ; for elimination of DC-removal filter

;
;      Update short tau reference power estimate (ABSYOF)
;      y_tilda(i+1) = (1-alpha)y_tilda(i) + alpha*|y(i)|
;      alpha = 2^(-5)  -> STAU = 11
;
               LD       ABSY0F, 16, A                ; ABSY0F *2**16 -> ACC
               SUB      ABSY0F, STAU, A              ; ACC-ABSY0F*2**STAU -> ACC
               ADD      ABSY0, STAU, A               ; ACC+ABSY0*2**STAU -> ACC
               STH      A, ABSY0F                    ; HIGH ACC -> ABSY0F
;
;      Update short tau near end power estimate (ABSS0F)
;      s_tilda(i+1) = (1-alpha)s_tilda(i) + 2*alpha*|s(i)|
;      alpha = 2^(-5) -> STAU = 11, NER = 1 (60dB NESP thresh.)
;
               LD       ABSS0F, 16, A                ; ABSS0F*2**16 -> ACC
               SUB      ABSS0F, STAU, A              ; ACC-ABSS0F*2**STAU -> ACC
               ADD      ABSS0, STAU+NER, A           ; ACC+ABSS0*2**STAU+NER ->ACC
               STH      A, ABSS0F                    ; ACCH -> ABSS0F
```

```
;
;       Update modulo 16 counter (H)
;
               LD       H, A                         ; H -> ACC
               ADD      #1, A                        ; ACC +1 -> ACC
               AND      #000Fh, A                    ; IF ACC = 16 THEN 0 -> ACC
               STL      A, H                         ; ACC -> H
;
;       Update M's every 16 samples
;
               BC       $no_dmov_Ms, AGT             ; update every 16 samples
;
;       Continue here if H = 0 :
;
               LD       max_m, A                     ; get previous largest partial maximum;
               LAR      AR1,last_m                   ; setup AR1 & AR2 for max search loop
               STM      #(M0+TAPS/BLOCKSIZE), AR1    ; setup AR1 & AR2 for max search loop
;
;       Check if  *last_m = max_m
;

               SUB      *AR1-, A
               BC       $no_max_calc, ANEQ
;
;       *last_m == max_m, need to recalculate largest M in set (since
;       *last_m is pushed off list by "DMOV" type operation) and DMOV
;       partial maxima.
;

               STM      #(TAPS/BLOCKSIZE),BRC
               RPTBD    $find_largest_M-1
               LD       #0,A                         ; delay slot
               LD       A,B                          ; delay slot
               LD       *AR1,A                       ; A = M[last_m-i]
               MAX      B                            ; B = max(A,B)
               DELAY    *AR1-                        ; "DMOV" partial maxima M(k)'s

$find_largest_M:
               BD       $check_hang                  ; (delayed)
               STL      B, max_m                     ; largest M(k) -> max_m

$no_max_calc:
               LD       ABSY0F, A                    ; (also executed for del. branch above)
               SUB      max_m, A
               STM      #(M0+TAPS/BLOCKSIZE-1), AR1
               NOP                                   ; latency nop
               XC       2, AGT                       ; max_m = max{ max_m, ABSY0F }
               LD       ABSY0F, A                    ; (remember: ABSY0F ~= short-time
r.m.s.
               STL      A, max_m                     ; level of y(n))

               RPT      #(TAPS/BLOCKSIZE)
               DELAY    *AR1-
               B        $check_hang                  ; NO TEST FOR NEAR END SPEECH
;
;       Update most recent partial maximum (M0) and max maximum (max_m)
;
;       Continue here if 0 < H <= 15:
;
$no_dmov_Ms:
               LD       ABSY0F, A
               SUB      M0, A
               BC       $check_nesp, ALEQ
               LD       ABSY0F, A
               STL      A, M0                        ; M0 = max{ M0 , ABSY0F }
               SUB      max_m, A
               BC       $check_nesp, ALEQ
               LD       M0, A
               STL      A, max_m                     ; max_m = max{ max_m , M0 }
$check_nesp:
```

```
;
;       If nearend signal power (ABSS0F = LP filtered |s(n)|, which is
;         an estimate of the short-time r.m.s. level of s(n) ) is greater than
;         the greatest of the partial maxima (max_m) then declare NEAREND SPEECH
;         DETECTED: load hangover counter (HCNTR) and freeze tap updates.
;
               LD       ABSS0F, A                    ; If ABSS0F > max_m, then declare NEAR-
               SUB      max_m, A                     ; END SPEECH.
               BC       $check_hang, ALEQ            ; Else, skip to HCNTR update.
               LD       HANGT, A
               STL      A, HCNTR
               ANDM     #~(1<<NONESP), control_flags
$check_hang:
;
;       Update hangover counter (HCNTR) if nonzero.
;
               LD       HCNTR, A                     ; skip HCNTR-- if already zero.
               BC       $check_cutoff, AEQ
               SUB      #1, A
               STL      A, HCNTR                     ; HCNTR--
               B        estimate_echo

$check_cutoff:
;
;       If long-time "power" estimate of y(n) (ABSY: actually closer to r.m.s.
;       level) is below cutoff level (CUTOFF), then skip tap update (by
;       clearing NONESP bit in control_flags).
;
               LD       ABSY, A                      ; ABSY -> ACC
               SUB      CUTOFF, A                    ; ACC - CUTOFF -> ACC
               ANDM     #~(1<<NONESP), control_flags
               BC       estimate_echo, ALEQ
               BD       estimate_echo                ; (delayed)
               ORM      #(1<<NONESP), control_flags
```

```
*********************************************************************
* Application: C54x LEC                                             *
* File name:   fir.asm                                              *
* Description:                                                      *
*              Estimate echo:  est_echo=conv(y(n),a(n))             *
*                                                                   *
* Author:      Jelena Nikolic, Associate Technical Staff            *
*              (translated from C5x)                                *
* Date:        January 97                                           *
*********************************************************************
               .title   "C54x LEC: fir.asm"
               .mmregs
               .include "echoequ.inc"

               .global  estimate_echo,update_taps, FILTER
               .global  S0, est_echo, error_out, ABSE0

               .text
estimate_echo:

;
;       CONVOLVE REFERENCE SAMPLES WITH FIR COEFFICIENTS
;       ref_ptr points to y(0)
;

               STM      #FILTER,filter_ptr          ; points to A0
               LD       #0,A
               STM      #0FFFFh,AR0                 ; index = -1
               RPT      #(TAPS-2)
               MAC      *ref_ptr+0%,*filter_ptr+,A
               MACR     *ref_ptr+0%,*filter_ptr+,A  ; add rounding offset.
               STH      A,1,est_echo

                                                    ; ref_ptr points to y(TAPS) now.
;
;       COMPUTE THE OUTPUT (error_out)
;
               LD       S0, A                       ; NEAR END SAMPLE
               SUB      est_echo, A                 ; SUBTRACT ECHO ESTIMATE
               STL      A, error_out                ; SAVE OUTPUT FOR UN(0)
               ABS      A
               STL      A, ABSE0                    ; ABSOLUTE FOR POWER ESTIMATE

               B        update_taps
```

```
*******************************************************************
* Application: C54x LEC                                           *
* File name:   update.asm                                         *
* Description: Adapt coefficients using the normalized block-update *
*              algorithm.                                         *
*                                                                 *
*              Refer to C20 application report for details.       *
*                                                                 *
* Author:      Jelena Nikolic, Associate Technical Staff          *
*              (translated from C5x)                              *
* Date:        January 97                                         *
*******************************************************************
               .title   "C54x LEC: update.asm"

               .mmregs
               .include "echoequ.inc"

               .global  control_flags, H, UN0, y0_ptr, error_out
               .global  update_taps, output_sample
               .global  N_ERROR, FILTER
               .global  ABSY0, AYLSBS, CUTOFF, IABSY, AELSBS, ABSOUT, ABSE0, ABSY

               .text
calc_power:
;
;       Update long-time reference power estimate (ABSY)
;
               STM      #LTAU,T
               DLD      ABSY,A
               SUB      ABSY, TS, A                 ; ACC - ABSY *2**LTAU
               ADD      ABSY0, TS, A                ; ACC + ABSY0*2**LTAU
               ADD      CUTOFF, TS, A               ; ACC + CUTOFF*2**LTAU
               DST      A, ABSY                     ; NEW ABSY
;
;       Compute 1/ABSY
;
               LD       #1, 16, A
               RPT      #14
               SUBC     ABSY, A
               STL      A, IABSY

update_taps:
;
;       Move UN0,UN1...UN14 to next higher memory location
;
               STM      #(N_ERROR+BLOCKSIZE-2), n_error_ptr     ; ADUN14 -> ACC
               RPT      #(BLOCKSIZE-3)                          ; K= 14,13....1
               DELAY    *n_error_ptr-                           ; UN(K) -> UN(K+1)
               DELAY    *n_error_ptr                            ; UN(0) -> UN(1)
;
;       Compute normalized output
;
               LD       error_out, T
               MPY      IABSY, A
               SFTA     A,15
               SFTA     A,1                         ; Q.31 format
               SAT      A                           ; saturate to +/- 1.0

               STH      A, *n_error_ptr             ; Save new UN0

               BITF     control_flags, #(1<<(15-(15-NONESP))|1<<(15-(15-UNFREEZE)))
               BC       output_sample, NTC          ; if near-end speech, skip update.

               LD       y0_ptr,A
               SUB      H,A
               STLM     A,ref_ptr
               STM      #0FFFFh,AR0
                                                    ; AR2 points to appropriate reference
                                                    ; sample block.
               STM      #(TAPS/BLOCKSIZE-1),BRC
               LD       #FILTER, A
               ADD      H, A
               SUB      #16,A                        ; because we preincrement: *+AR4(16)
               STLM     A, filter_ptr                ; TEMP3 = address of A(H-16)

               SSBX     FRCT

               RPTBD    $block_end-1
               STM      #N_ERROR, n_error_ptr
               RPTZ     A,#(BLOCKSIZE-2)
               MAC      *ref_ptr+0%,*n_error_ptr+,A  ; Y (Q0) * UN (Q15)
               MACR     *ref_ptr+0%,*n_error_ptr,A   ; last sumation: round.

               MPYA     IABSY                        ; A(32-16) * T -> B <=>
                                                     ; INC(Q0)*IABSY(Q15)->B(Q16)
               LD       *+filter_ptr(16), 16, A      ; a_k(i)(Q15 format) -> AH
               SFTA     B,#(15+GAIN)                 ; B(Q.16) << 15 = B(Q.31)
                                                     ; B(Q.31) * 2^(GAIN)
               ADD      B,A                          ; B  + A -> A
               STH      A, *filter_ptr               ; A -> a_k(i+1), k -> k - 16
               STM      #N_ERROR, n_error_ptr
$block_end:

               RSBX     FRCT
               B        output_sample
```

```
*******************************************************************************************
* Application: C54x LEC                                                      *
* File name:   nlms.asm                                                      *
* Description: Filtering and filter coefficients adaptation using            *
*              normalized LMS algorithm:                                     *
*              (1) Calculate long-term power estimate of y(n):               *
*                  power(y(i+1)) = (1-alpha) power(y(i)) + alhpa*|y(i)|      *
*                  alpha = 2^(-7)                                            *
*              (2) Check if the long-term power estimate of y(i) is          *
*                  below cutoff                                              *
*              (4) If allowed, adapt coefficients:                           *
*                  a_k(i+1) = a_k(i) + d_error_mu*(y(i-1-k)/L_y              *
*                  where:                                                    *
*                  d_error_mu = 2beta*e(i-1)/L_y                             *
*                  L_y=long term average of absolute reference signal        *
*                  2beta = 2^(GAIN)                                          *
* Author:      Jelena Nikolic, Associate Technical Staff                     *
* Date:        January 97                                                    *
*******************************************************************************************

               .title   "C54x LEC: nlms.asm"
               .mmregs
               .include "echoequ.inc"

               .global  control_flags, IABSY, n_error_mu, error_out, S0
               .global  output_sample, filter_and_adapt
               .global  OUTPUT, ABSE0, Y0, FILTER, control_flags
               .global  ABSY0, CUTOFF, ABSY, AYLSBS, IABSY, AELSBS

               .text

filter_and_adapt:
calc_power:
               LD       *ref_ptr,0,A
               ABS      A
               STL      A,ABSY0
;
;       Update long-time reference power estimate (ABSY)
;

               STM      #LTAU,T                     ; for dymamic shift.

               DLD      ABSY,A                      ; double presicion load
               SUB      ABSY,TS ,A                  ; ACC - ABSY *2**LTAU
               ADD      ABSY0,TS,A                  ; ACC + ABSY0*2**LTAU
               ADD      CUTOFF,TS,A                 ; ACC + CUTOFF*2**LTAU
               DST      A,ABSY                      ; double precision store.

;
;       Compute 1/ABSY
;
               LD       #1, 16, A
               RPT      #14
               SUBC     ABSY, A
               STL      A, IABSY

$check_cutoff:
;
;       If long-time "power" estimate of y(n) (ABSY: actually closer to r.m.s.
;       level) is below cutoff level (CUTOFF), then skip tap update (by
;       clearing NONESP bit in control_flags).
;
               LD       ABSY, A                     ; ABSY -> ACC
               SUB      CUTOFF, A                   ; ACC - CUTOFF -> ACC

               STM      #FILTER,filter_ptr          ; AR3 points to filter coefficients
               STM      #0FFFFh,AR0                 ; index

               XC       2,ALEQ
               ANDM     #~(1<<NONESP), control_flags ; skip update.
```

```
nlms_routine:
;
;       If the filter coeff's shouldn't be updated, load T with zero,
;       which will cause coefficients to remain the same where and the
;       convolution will still be performed.
;

               BITF     control_flags, #(1<< (15-(15-NONESP))|1<< (15-(15-UNFREEZE)))

               STM      #(TAPS-2),BRC
               SSBX     FRCT

               XC       2, NTC                        ; if update frozen, skip update.
               ST       #0, n_error_mu

               LD       #0,B
               LD       n_error_mu, T                 ; norm_error * mu -> T

               RPTBD    $nlms_end-1
               MPY      *n_ref_ptr+0%,A
               LMS      *filter_ptr,*ref_ptr+0%

               ST       A,*filter_ptr+
            || MPY      *n_ref_ptr+0%,A               ; error (in T) * n_ref_sample -> A
               LMS      *filter_ptr,*ref_ptr+0%       ; A + coef<<16 + round. -> A
                                                      ; coef*ref+B -> B
$nlms_end:
               ST       A,*filter_ptr
            || SUB      *AR5,A                        ; S0 << 16 -  B (est.echo) -> A
               STH      A,error_out
               ABS      A
               STH      A,ABSE0                       ; abs value for power estimate.

;
;       Update the normalized reference sample buffer.
;
               LD       *ref_ptr+%,16,A               ; y(0) -> A
               MPYA     IABSY                         ; y(Q0) * iabsy(Q15) -> Q16 ( frct!)
               SFTA     B,15                          ; it's Q31 now.
               SAT      B
                                                      ; n_ref_ptr points to the newest sample
               MAR      *n_ref_ptr+%
               STH      B,*n_ref_ptr                  ; overwrite the oldest sample
                                                      ; in the norm. reference buffer.

               RSBX     FRCT
;
;       Compute normalized error * stepsize (for next time)
;

               LD       error_out,T                   ; error_out -> T
               MPY      IABSY, A                      ; T * IABSY -> A ( in Q.15 format)
               SFTA     A,15
               SFTA     A,1                           ; in Q.31 format
               SAT      A                             ; saturate norm. output @ +/- 1.0

               BD       output_sample

               SFTA     A,#GAIN                       ; norm_error * 2^GAIN (GAIN = 10)
               STH      A,n_error_mu
```