Echo Cancellation Software for the TMS320C54x

Literature Number: BPRA054
Texas Instruments Europe
March 1997

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain applications using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

Contents

List of Figures

List of Tables

Echo Cancellation Software for the TMS320C54x

ABSTRACT

This application note describes how to realize an Electrical Echo Canceller using the LMS instruction of the TMS320LC54x series DSP. Thanks to this compact instruction, it is possible to perform the Prediction Filter (FIR) and the weight updating at the same time. The software described is based on the Leaky Normalized Least Mean Square adaptive (LNLMS) filter with single precision (16 bit) coefficients. This software is able to cancel an echo path of up to 64 ms using only a few MIPS. It is also possible to implement more than one channel on the same LEAD without external memory (using LC548/9). This software can be easily customized to any echo path length.

1. Echo Canceller - A Bit of Theory

The figure below shows a typical connection between two subscribers through the central office.

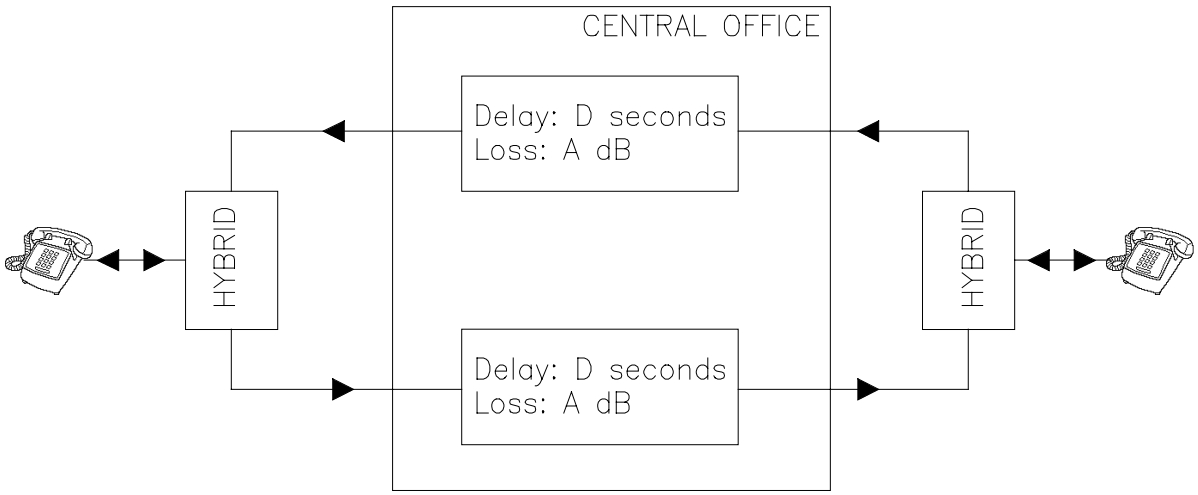


Figure 1: Subscribers - Central Office Connection

In Figure 1, the path delay D and the loss A are highlighted. To reduce the cost of wiring between the central office and the telephone set at the subscriber site the electrical connection is via a 2-wire line. The connections in the central office are based on 4 wires. To adapt 4- to 2-wire a device called a Hybrid is used. Because of the low cost wiring employed, more complex signal amplifiers are needed along the line. In the old telephone set the hybrid was realized by means of a tuned transformer. Due to the impedance mismatch

between the hybrid and the telephone line some of the signal transmitted from one side returns and corrupts the signal, generating an echo that is very disconcerting in voice communications. While the echo in human communications can be tolerated, in modem communications it can be catastrophic. To solve this problem the telephone companies employ a device called an Echo Canceller. This device is nothing more than an adaptive filter with automatically adjustable coefficients. From the electrical point of view an Echo Canceller can be represented as a quadripole connected as shown Figure 2.

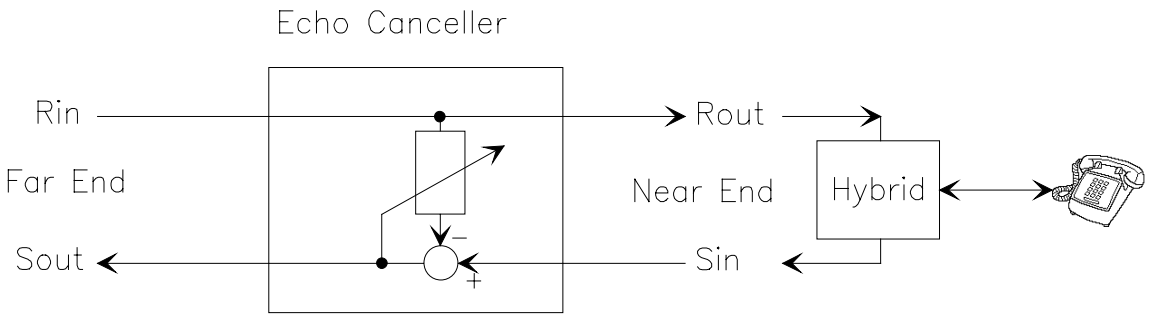


Figure 2: Echo Canceller Unit

where Rin stands for Receive-In Port, Rout for Receive-Out Port, Sin for Send-In Port and Sout for Send-Out Port as stated in ITU-T G165.

2. Echo Canceller - The Algorithm

As mentioned before, the Echo Canceller unit can be represented as a quadripole connected between the telephone line and the hybrid. The hybrid can be represented by an FIR plus an additional attenuation ranging from 6 to 11 dB.

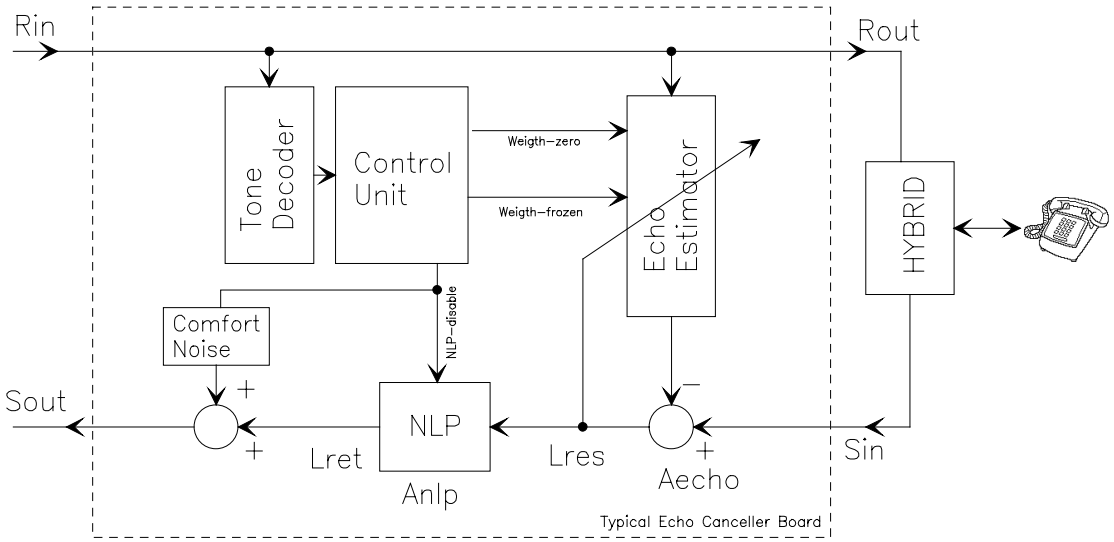


Figure 3: Electrical Echo Canceller

As shown in Figure 3, the Electrical Echo Canceller is much more than a simple Echo Estimator. The LMS routine, like all the other adaptive algorithms, can diverge if particular conditions are met. For example, the LMS routine could diverge if a pure tone appears at Rin. To avoid this inconvenience, the weight updating must be frozen. For this purpose other functional blocks are added to control the updating of the weights.

Let's have a look to the different functional blocks included in a typical Electrical Echo Canceller (EEC) board. The Tone Decoder is used to detect when tones are sent along the line, (e.g., during the dialing phase). The Tone Decoder outputs are wired to the Control Unit. This unit controls the updating of the weights, freezing or zeroing their values when needed. The Echo Estimator is the adaptive FIR used to predict the echo path. Immediately after this unit, a Non-linear Processor is used. The purpose of this is to reduce the amount of echo remaining after the subtraction from the predicted echo. This element effectively blocks the signal completely or partially if the residual echo is higher than a defined threshold. When the signal is blocked, a comfort noise is inserted by the Comfort Noise Generator.

The algorithm of the Echo Estimator is briefly described below. Each time a sample is received from the line, the following steps are executed and the corresponding values computed.

Step 1. The predicted Near End Signal is obtained by applying the FIR equation:

$$y_n = \mathbf{x_n} * \mathbf{h}^t(n)$$

where '*' corresponds to convolution, $\mathbf{h}(n) = [h_1(n), h_2(n), ...,h_N(n)]$ is the weights vector and $\mathbf{x_n}=[x_n, x_{n-1}, ..., x_{n-N}]$ are the input samples.

Step 2. The energy of the Far End Signal is obtained from:

$$\sigma_x^2(n) = (1-\alpha)\cdot\sigma_x^2(n-1) + \alpha\cdot x_n^2$$

in other words, it uses a leaky integrator with a forgetting factor of 1- α.

Step 3. The error between the Near End returned signal and the predicted one is obtained using:

$$e_n = v_n - y_n$$

Step 4. Finally the weights are updated according to:

$$\mathbf{h}(n+1) = \xi\cdot\mathbf{h}(n) + (\sigma^2_x(n))^{-1}\cdot\beta\cdot e_n\cdot x_n$$

having the following values for the constants:

- ξ is the leakage factor set to $(1- 2^{-26})$
- β is equal to $2\cdot u/L$ (u is the convergence factor and L is the FIR length

in taps). For 128 taps (16 ms of echo path) it is set to 2^{-9}

The leakage factor ξ is applied to each weight. It is very difficult measure the real effect of the leakage factor due to its small effect on the weight. In spite of this, all the commonly used EECs are equipped with this feature.

3. The LMS Instruction

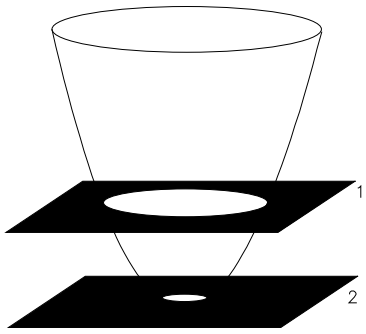
The TMS320(L)C54x has a powerful instruction set particularly suited to telecom applications. Among these, a Least Mean Square Instruction has been added for very rapid implementation of the adaptive algorithms. In only one micro-cycle the DSP is able to perform the following actions:

$$y(n) = \Sigma(x(n-k) \cdot a_k(n)) \;\; || \;\; a_k(n+1) = a_k(n) + 2 \cdot \beta \cdot e(n-1) \cdot x(n-k) \quad (1)$$

where || means 'execute in parallel' and n is the sample. This means that the weights are updated using the error e(n-1) computed in the previous step. This is different from the common LMS routine where the updating is made using the actual error e(n).

4. The Results

Before starting with the results obtained using the LMS instruction of the LEAD, some explanation should be given about the fixed point arithmetic related to the adaptive filters. From a theoretical point a view, the power of the $E[e^2(n)]$[1] can approach zero when a floating point implementation is chosen. Due to the use of the fixed point DSP the power of e(n) cannot go below the minimum value that can be expressed with 16 bits, that is ≈-90dB. In other words, even using a perfect LMS routine the power of the residual echo cannot be less than -90dB. Unfortunately the LMS uses a number of summations and multiplications that reduce the accuracy of the algorithm. The minimum value of attenuation is a function of several factors such as the accuracy of the arithmetic, the number of taps (echo path), the step size, etc. The result is that the algorithm does not reach the absolute minimum of the function (plane 2) but remains slightly above it (plane 1), as shown in the Figure 4.



[1] E[•] denotes the expected value

Figure 4: Mean Squared Error Surface

Before reporting any results let's define two quantities usually used to describe the characteristic of an Electrical Echo Canceller. The first quantity is the Echo Return Loss (ERL) defined as:

$$ERL = 10 \cdot Log_{10}(E[Rin^2]/E[Sin^2])$$

Simply put, it represents the attenuation of the hybrid. The second one is the Echo Return Loss Enhancement (ERLE) defined as:

$$ERLE = 10 \cdot Log_{10}(E[Sin^2]/E[Sout^2])$$

Taking in account all the parameters listed before (σ^2_x, β, e_n) the minimum values for the ERLE can be found using a mathematical model for the Echo Canceller with a finite arithmetic. Under the following hypothesis:

β	2^{-9}
ERL	≈9dB
Convolution	16 bit
Weight updating	16 bit

the maximum value for the ERLE can be obtained for different lengths of echo path as shown in Table 1 for linear inputs.

Table 1: Maximum Value for the ERLE

Echo Path	Taps	$ERLE_{max}$
16 ms(*)	128	57
32 ms(*)	256	51
64 ms(*)	512	45

(*) for 8 kHz of sampling frequency

This section contains the result obtained running the program with two different sets of inputs. Firstly, pure 13 bit linear inputs were used to test and validate the values of ERLE obtained from theory. Secondly, the numbers were obtained passing the same signal through PCM companders.

In the following figures the spectrum of the Far End (Figure 5), and Near End (Figure 6) signals are shown. Both are Gaussian White Noise signals filtered with a narrow-band filter having the same bandwidth as the telephone line. The Far End power is around -20 dB while the Near End power is ≈9 dB less.

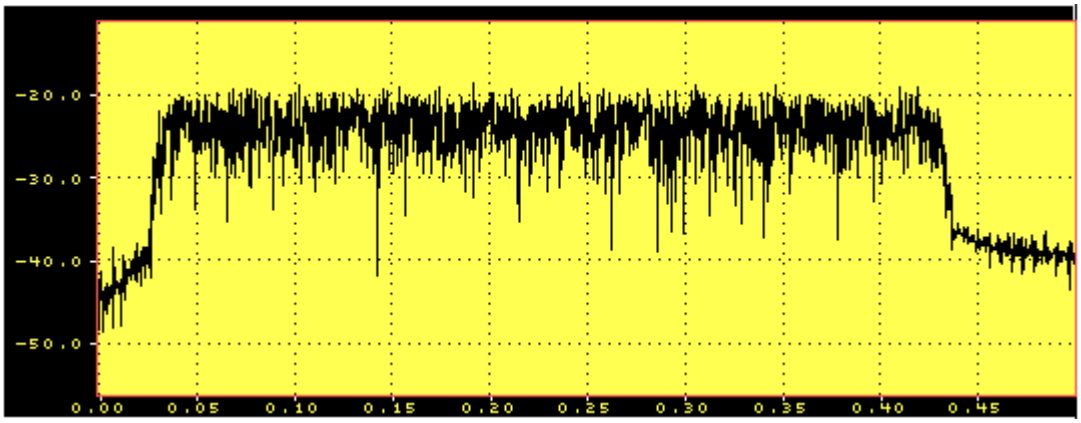


Figure 5: Spectrum of the Far End Signal (dB vs. Normalized Frequency)

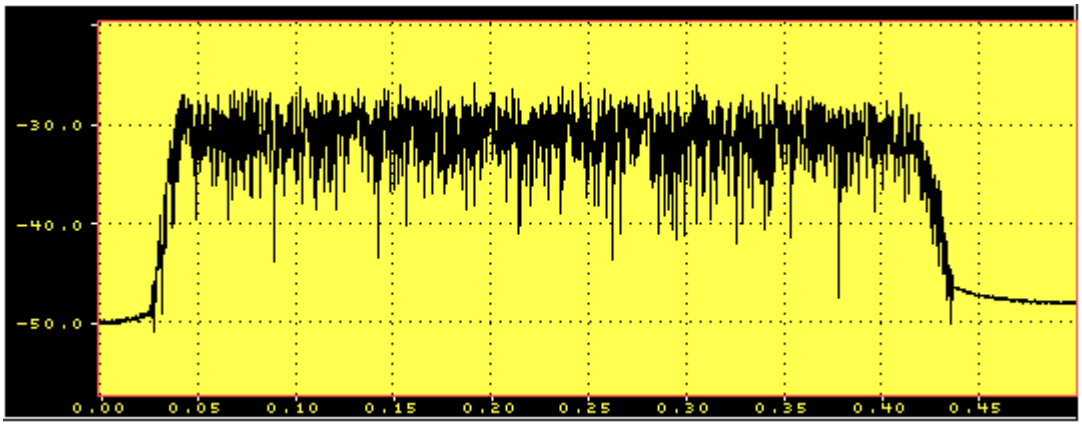


Figure 6: Spectrum of the Near End Signal (dB vs. Normalized Frequency)

4.1 Results Obtained in a Simulated Environment

Table 2, below shows the numbers obtained using a simulator.

Table 2: Results obtained in a Simulated Environment

	Echo Path							
	16 ms (128 taps)				32 ms (256 taps)			
					32 ms			
Far End	Linear input 13 bit		PCM input		Linear input 13 bit		PCM input	
Power	Eout	ERLE	Eout	ERLE	Eout	ERLE	Eout	ERLE
-15	-78	53	-60	33	-76	51	-57	32
-20	-82	53	-64	32	-81	51	-62	32
-30	-86	47	-71	32	-88	46	-71	30

Note: all the numbers are in dB

Note that due to the noise introduced by the non-linear quantization of the compander, the numbers obtained using PCM inputs are less than those obtained with the linear input. On the other hand, using 256 taps some dB are lost due to the noise introduced by the weights having a value near zero. In fact, analyzing the impulse response of the hybrid, it is limited to 16 ms (Figure 7). Figure 8shows the reconstructed hybrid impulse response.

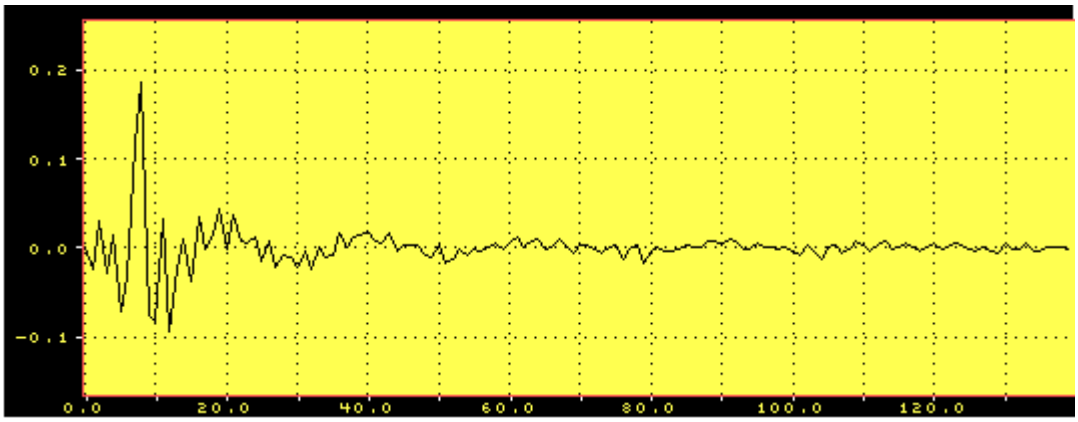


Figure 7: Impulse Time Response of the Simulated Hybrid (weight vs. number of weight)

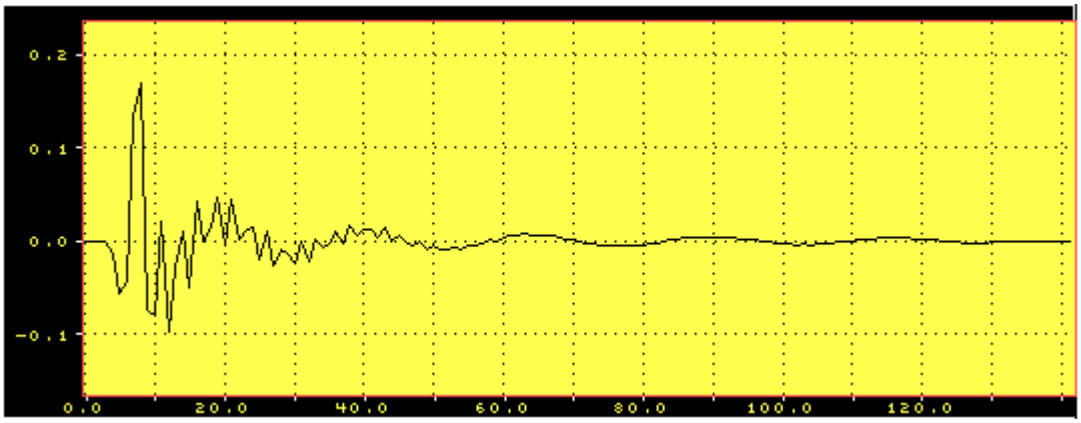


Figure 8: Impulse Time Response of the Reconstructed Echo Path (weight vs. number of weight)

The figures below show the diagrams of the power of Rout (Figure 9) and ERLE (Figure 10), during the convergence phase. Convergence is reached after 2000 samples (25 ms) using 128 taps FIR.

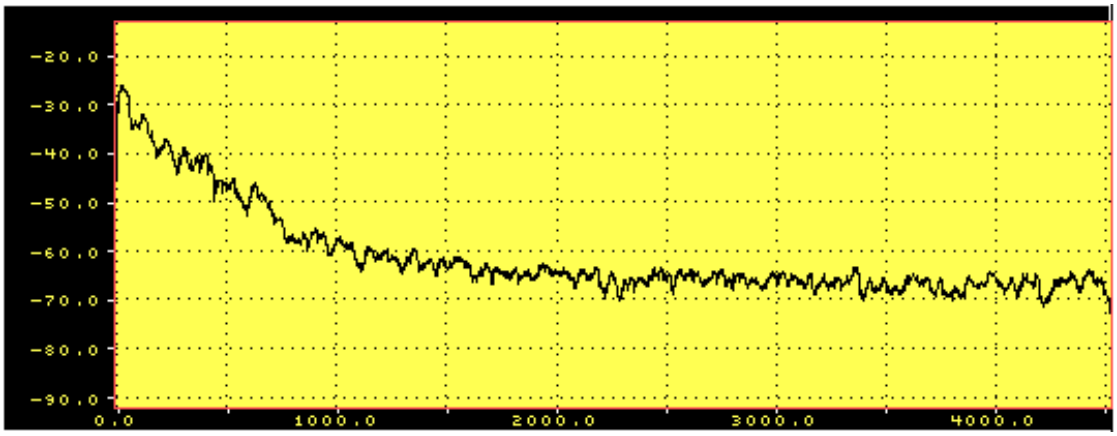


Figure 9: En (dB vs. Samples)

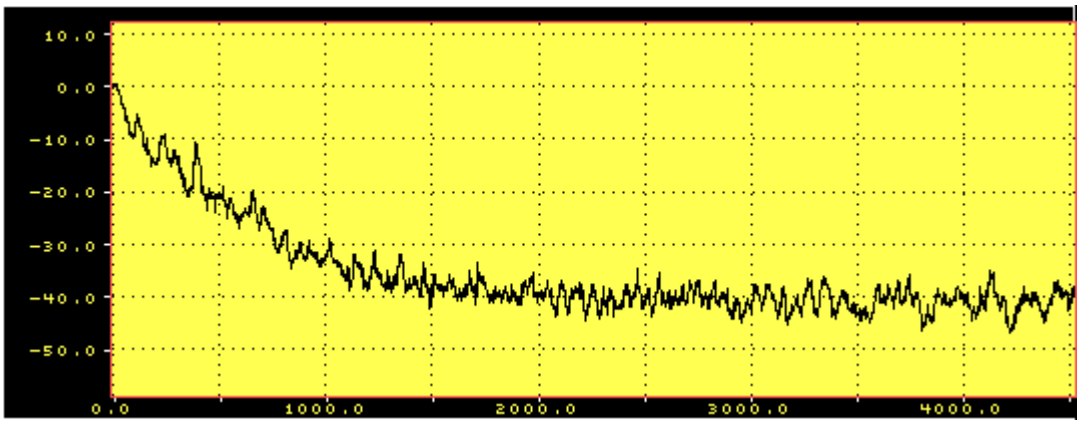


Figure 10: ERLE (dB vs. Samples)

5. The Program

This section contains the EEC program when running on the simulator. Nevertheless, it can be easily modified to suit every type of hardware. The core of the program is the LMS instructions. As already explained, this routine performs in the same micro-cycle one convolution step and the update of one weight. This is the section where it is applied.

```
; ************************************************************************************
;                                   LMS Routine
;
        stm     #COEFF,COEFF_ptr                ; points the vector of weights
        stm     #ERRORE,ERRORE_ptr              ; points the previous step error
        stm     #(N_COEFF-1),brc                ; set repeat counter
        rptbd   LMS_end-1                       ; repeat block to compute the LMS
        ld      #0,b                            ; zero of acc.B (Yn)
        mpy     *ERRORE_ptr,*X_in_ptr+0%,a      ; compute error*X(n-0)
         lms    *COEFF_ptr,*X_in_ptr            ; FIR + weight updating
         st     a,*COEFF_ptr+                   ; store new weight
         || mpy *X_in_ptr+0%,a                  ; compute error*X(n-k)
LMS_end:sth     b,@Yn                           ; store the result (Yn)

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

Table 3, below contains the MIPS and the Data Program requirements to run the EEC for one voice channel:

Table 3: MIPS & Data Program Requirements to Run EEC for One Voice Channel

Performance			DATA (words)		
Echo Path	Taps	MIPS	Buffers	Static	Program
16	128	2.85	256	11	250
32	256	4.90	512	11	250
64	512	9.00	1024	11	250
0.80 MIPS should be added if the A-law compander is used compander.					

The section of code below explains how the leakage is implemented. According to the following formula:

$$\mathbf{h}(n+1) = \xi \cdot \mathbf{h}(n) + (\sigma^2_x(n))^{-1} \cdot \beta \cdot e_n \cdot x_n$$

the effect of the leakage is to move one step along the gradient from the minimum, progressively improving the performance during runtime. As previously reported, the value of ξ is $1- 2^{-26}$ and this value cannot be represented with a 16 bit machine. To overcome this issue the leakage is implemented by applying the leakage factor every N samples (eight for 128 taps). This task is performed by the following section of code.

```
; ************************************************************************************
;                         Routine for the Leakage
;
Leak:   ld      @LEAKAGE_count,b          ; load the leakage counter
        sub     #(N_LEAKAGE-1),b          ; check if the leakage has to be applied
        rcd     blt                       ; return if it isn't applied
         addm    #0001h,@LEAKAGE_count    ; increment the leakage counter
        mpy     *LEAKAGE_ptr,#7fffh,a     ; mpy to scaled value of Leakage 1-(2^-16)
        sth     a,*LEAKAGE_ptr+%          ; store new value and point next weight
        retd                              ; return
         andm   #0,@LEAKAGE_count         ; reset the leakage counter

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

The program for the Echo Canceller is divided in three parts:

Vectors tables	specifies the vector table
A-law Compander	performing the decompression from 8 bit A-law to 13 bit linear samples and the compression from 13 bit linear sample to 8 bit A-law.
Echo Canceller	performs the adaptive filter using the LMS instruction

along with these parts the linker command file and the simulator configuration file are included.

References and Bibliography

1. "Digital Communication" - John Proakis - McGraw Hill
2. “TMS320C54x User Guide” - Texas Instruments - SPRU131C
3. "Adaptive Filter - A Review of Techniques" - P.Hughes, J.Cook, B.T. Technol. Vol.10 No.1 January 1992
4. “A-law Compander Routines for LEAD” - G.Peake - Texas Instruments
5. “General Characterisitcs of International Telephone Connections and Internatioanl Telephone Circuits - Echo Canceller - ITU-T Recommendation G.165 03/93

Appendix A. Source Code

```
;                        **** ECHO CANCELLER SOFTWARE FOR LC54X ****
;                              *** D.Comunale TI-Italy ***


        .title "ECHO CANCELLER"

        .mmregs

;*******************************************************************************
;*******************************************************************************
;*******************************************************************************
;
; Echo canceller. Algorithm description
;
; Arithmetic range:
; both coefficients and samples are stored in Q15 format (-1<..<+1)
;
;
;                             /--------------------\
;                X_in ------>|                    |-------> (N/U)
;                            |  ECHO CANCELLER    |
;     Far End      Yn <------|   N_COEFF taps     |                Near End
;                            |                    |
;                  En <------|                    |<------- V_in
;                            \--------------------/
;
; Algorithm flow:
;
; 0) Read new samples -> X_IN[n] and V_in[n]
;
; 1) Sigma is computed as follow:
;          Sigma[K]^2 = (1-alpha)*Sigma[K-1]^2 + alpha*X_in[n]^2
;          alpha = 0.01
; 2) A non-linear filter is applied:
;           if C*X_in[n]^2 < Sigma[K]^2  goto step 3 else Sigma[K]^2 = X_in[n]^2
;           C = 2^-8
; 3) Perform the FIR computation: Yn=Sum(X_in[i]*FILTER[i])
;    the prediction error is obtained as: e[n] = V[n] - Y[n]
;    the weights are updated:
;           FILTER[i,K+1] = Zita*FILTER[i,K] + Beta/(Sigma[K]^2)*e[K]*X_in[i]
;           Zita = 2^-26
;           Beta = 2^-9
;
; 4) The far end samples is outputed
;
; 5) goto 0)
;
; *******************************************************************************
; *******************************************************************************
; *******************************************************************************

DRR             .set    20h
DXR             .set    21h
SPC             .set    22h
SPCE            .set    23h
AXR             .set    38h
BKX             .set    39h
ARR             .set    3ah
BKR             .set    3bh

                                            ; Buffers definition

N_WEIGHT        .set    127                 ; Filter length - 1
BUFFER_LEN      .set    N_WEIGHT+1          ; Buffer length

BUFFER_IN       .usect  "in_buff",BUFFER_LEN,1  ; Incoming data buffer
WEIGHT          .usect  "filter",N_WEIGHT+1,1   ; Weights

ON              .set    1
OFF             .set    0

COMPANDER       .set    ON                  ; if ON emulate the dB loss due to
                                            ; the compander

N_LEAKAGE       .set    0008h               ; specified the number of cycles to
                                            ; apply the leakage

                                            ; I/O ports for the simulator
```

```
PORTA_R1        .set    1000h                   ; SW Input  port used to read the far
                                                ; end sample coming from the line
PORTA_R2        .set    1001h                   ; SW Input  port used to read the
                                                ; near end sample coming from the
                                                ; hybrid
PORTA_W1        .set    2000h                   ; SW Output port sample to line
PORTA_W2        .set    2001h                   ; SW Output port output from the FIR
                                                ; filter (debug purpose only)

; Constants

ALPHA_val       .set    0x0147                  ; alpha     = 0.01      16 bit Q15
                                                ; (.01*32767)
ONE_ALPHA_val   .set    0x7eb8                  ; 1 - alpha = 0.99      16 bit Q15
                                                ; (.99*32767)
C_val           .set    0x0080                  ; C = 2^-8              16 bit Q15

                                                ; Variables
                .bss    ALPHA_pos,1             ; dummy variable
                .bss    V_in,1                  ; near end sample from
                                                ; hybrid  -> LINEAR
                .bss    Yn,1                    ; sample from the FIR -> LINEAR
                                                ;(debug purpose only)
                .bss    X_in,1                  ; far end sample
                                                ; from the line -> LINEAR
                .bss    ONE_ALPHA_pos,1         ; constant
                .bss    C_pos,1                 ; constant
                .bss    En,1                    ; (En = Vn - Yn) far end output (to
                                                ; the line) -> LINEAR
                .bss    One_Sigma_L,1           ; 1/Sigma2 - low 16 bit
                .bss    One_Sigma_H,1           ; 1/Sigma2 - high 16 bit

                .bss    SIGMA,1                 ; Sigma2
                .bss    ERROR,1                 ; En*Beta/Sigma2
                .bss    LEAKAGE_count,1         ; Counter for the leakage

                .bss    V_in_PCM,1              ; near end sample

                                                ; from hybrid  -> PCM
                .bss    X_in_PCM,1              ; far end sample
                                                ; from the line -> PCM
                .bss    En_PCM,1                ; (En = Vn - Yn) far end output
                                                ; to the line -> PCM
                .bss    SAMPLE_cnt,1            ; counter of the incoming samples
                                                ; (debug)

                                                ; COMPANDER variables
                                                ; These variables must be accessed
                                                ; the DP=0 (scratch)

LINEAR_in       .usect "COMP_Dat" ,1            ; 13 bit LINEAR sample to be
                                                ; compressed
PCM_out         .usect "COMP_Dat" ,1            ; 8 bit PCM compressed output

BADDR           .usect "COMP_Dat" ,1            ; A-law decopression table
PCM_in          .usect "COMP_Dat" ,1            ; 8 bin PCM sample to be expanded
LINEAR_out      .usect "COMP_Dat" ,1            ; 13 bit LINEARized output

; Definitions

                .asg    NOP,    PIPE            ; Due to pipeline DON'T REMOVE
                .asg    AR0,    IDX_reg         ; Index register
                .asg    AR1,    LEAKAGE_ptr     ; Pointer for the weigth 'under leakage'
                .asg    AR2,    SIGMA_ptr       ; AR2..5 only
                .asg    AR2,  LINEAR_OUT_ptr    ; AR2..5 only
                .asg    AR2,    PCM_OUT_ptr     ; AR2..5 only
                .asg    AR3,    CONST_ptr       ; AR2..5 only
                .asg    AR4,    WEIGHT_ptr      ; AR2..5 only
                .asg    AR5,    X_in_ptr        ; AR2..5 only
                .asg    AR3,    ERROR_ptr       ; AR2..5 only

;*******************************************************************************

        .sect   "program"


Start:  nop

                                                ; Harware settings
                                                ; OVLY=1 MC=0
        stm     #00000h,SWWSR
        stm     #00000h,BSCR
```

```
                                        ; Echo canceller variables set up

        ssbx    frct                    ; set fractional mode
        ssbx    sxm                     ; set sign extention

        ld      #ALPHA_pos,dp           ; set the DP
        st      ALPHA_val,@ALPHA_pos    ; store the value of Alpha
        st      ONE_ALPHA_val,@ONE_ALPHA_pos ; store the value of 1-Alpha
        st      C_val,@C_pos            ; store the value of C
        stm     #SIGMA,SIGMA_ptr        ; load the pointer of Sigma2

        stm     #WEIGHT,LEAKAGE_ptr     ; load the pointer of leakage
        stm     #WEIGHT,WEIGHT_ptr      ; load the pointer of weighs
        rptz    b,#(N_WEIGHT-1)
         stl    b,*WEIGHT_ptr+          ; zeroing of weights

        stm     #BUFFER_IN,X_in_ptr     ; load the pointer of the incoming
                                        ; sample vector
        rptz    b,#(BUFFER_LEN-1)
         stl    b,*X_in_ptr+            ; zeroing incoming samples
        stl     b,@V_in                 ; reset the variable
        stl     b,@X_in                 ; reset the variable
        stl     b,@Yn                   ; reset the variable
        stl     b,@ERROR                ; reset the variable
        stl     b,@LEAKAGE_count        ; reset the variable
        stl     b,SAMPLE_cnt            ; reset the variable (debug)

        stm     #BUFFER_IN,X_in_ptr     ; load the pointer of the incoming
                                        ; sample vector
        stm     #BUFFER_LEN,BK          ; set the size o circular buffer
        stm     #1,IDX_reg              ; set the index register

; main loop

wait:   nop
        call    Receive
        call    Transmit
        b       wait

;*****************************************************************************
;*** Receive the samples from Near and Far end

Receive:
        ld      #ALPHA_pos,DP           ; set DP
        portr   PORTA_R1,@X_in          ; read one sample from Far  End
        portr   PORTA_R2,@V_in          ; read one sample from Near End

       .if      COMPANDER == ON
        ld      @X_in,a                 ; apply the A-law compander to the Far
                                        ; End (X_in)
        stlm    a,LINEAR_in
        call    Compress                ; compander routine - Linear -> A-law
        ldm     PCM_out,a
        stlm    a,PCM_in
        call    Decompress              ; compander routine - A-law -> Linear
        ld      #ALPHA_pos,DP           ; set DP
        ldm     LINEAR_out,a            ; store the new value passed throught
                                        ; the compander
        stl     a,@X_in

        ld      @V_in,a                 ; apply the A-law compander to the Near
                                        ; End (V_in)
        stlm    a,LINEAR_in
        call    Compress                ; compander routine - Linear -> A-law
        ldm     PCM_out,a
        stlm    a,PCM_in
        call    Decompress              ; compander routine - A-law -> Linear
        ld      #ALPHA_pos,DP           ; set DP
        ldm     LINEAR_out,a
        stl     a,@V_in                 ; store the new value passed throught
                                        ; the compander
       .endif

        addm    #1,SAMPLE_cnt           ; increments the sample counter (debug)

        call    Echo_can                ; call E.E.C.
        ret

;*****************************************************************************
;*** Transmit the cancelled sample to Far end

Transmit:
       .if      COMPANDER == ON
        ld      @En,a                   ; load the output
```

```
        stlm    a,LINEAR_in
        call    Compress                        ; apply the A-law compression
        ldm     PCM_out,a
        stlm    a,PCM_in
        call    Decompress                      ; apply A-law decompression
        ld      #ALPHA_pos,DP                   ; set DP
        ldm     LINEAR_out,a
        stl     a,@En                           ; store the new value passed throught
                                                ; the compander
       .endif

        portw   @En,PORTA_W1                    ; write the cancelled output onto file
        portw   @Yn,PORTA_W2                    ; write the FIR output onto file (DEBUG
                                                ; purpose only)
        ret                                     ; return

;*******************************************************************************
;*** ECHO CANCELLER MAIN ROUTINE

Echo_can:
        ld      #ALPHA_pos,dp                   ; set DP

;*******************************************************************************
;*** LMS Routine

        stm     #WEIGHT,WEIGHT_ptr              ; load the pointer
        stm     #ERROR,ERROR_ptr                ; load the pointer
        stm     #(N_WEIGHT-1),BRC               ; set the repeat counter
        rptbd   LMS_end-1                       ; LMS main loop
        ld      #0,b                            ; zeroing of B (Yn)
        mpy     *ERROR_ptr,*X_in_ptr+0%,a       ; compute error*X(n-0)
         lms    *WEIGHT_ptr,*X_in_ptr           ; FIR + weight update
         st     a,*WEIGHT_ptr+                  ; store new weight
         || mpy *X_in_ptr+0%,a                  ; compute new error*X(n-k)
LMS_end:sth     b,@Yn                           ; store FIR output (Yn)

;*******************************************************************************
;*** Load next sample
        ld      @X_in,a                         ; load next sample
        stl     a,*X_in_ptr+0%                  ; store it in sample buffer

;*******************************************************************************
;*** SIGMA2 COMPUTATION
; Sigma2[K]   = (1-alpha)*Sigma2[K-1] + alpha*sqr(X_in[n])
; alpha = 0.01
;
; alpha*sqr(X_in[n]) -> B

S_Sigma:stm     #ONE_ALPHA_pos,CONST_ptr        ; load the pointer
        ld      @X_in,16,a                      ; load sample in AccH
        squr    a,a                             ; SQR of sample -> A
        mpya    @ALPHA_pos                      ; alpha*A -> B, point 1-alpha

                                                ; Sigma2[K]   = (1-alpha)*Sigma2[K-1]+B
        macr    *SIGMA_ptr,*CONST_ptr,b,a       ; Sigma2[K-1]*(1-alpha) + B -> A, pointC
        sth     A,*SIGMA_ptr                    ; update Sigma2

; 2) Not linear filter
; if C*sqr(X_in[n]) < Sigma2[K] (1) go ahead
; else Sigma2[K] = sqrt(X_in[n])

        ld      @X_in,16,a                      ; load sample in AccH
        squr    a,a                             ; SQR of sample -> A
        mpya    @C_pos                          ; A*C -> B
        sub     *SIGMA_ptr,16,b                 ; B = A*C - Sigma2
        saccd   a,*SIGMA_ptr, bgt               ; store sqr(X_in[n]) if (1) is met

;*******************************************************************************
;*** En Computation

S_En:   ld      @V_in,16,a                      ;, Load sample of Near End (Vn)
        sub     @Yn,16,a                        ; subtract Vn with FIR output (Yn)
        sth     a,@En                           ; store in En

;*******************************************************************************
;*** B*(En*(1/Sigma2)) Computation


S_E_Gamma:                                      ; *** compute 1/Sigma2 (the sign is
                                                ;     always positive)
        ld      #7fffh,a                        ; load 1
        rpt     #(16-1)                         ; divide
         subc   *SIGMA_ptr,a                    ; ACC.L -> quote, ACC.H -> remain
        stl     a,@One_Sigma_H                  ; High 16 bit of quote
```

```
        andm    #0,@One_Sigma_L              ; zeroing of the Low 16 bit of quote
        PIPE
                                             ; *** multiply by En
        ld      @One_Sigma_H,16,a            ; load the result of 1/Sigma2
        add     @One_Sigma_L,a               ; add low part of result
        mpya    @En                          ; multiply B*En (Q15)
                                             ; *** multiply by Beta
        sat     b                            ; saturate the result
        sfta    b,7,b                        ; decimal point adjust  (2^-9*2^15=2^6)
        sth     b,@ERROR                     ; store Error (Q15)


;*** LEAKAGE


Leakage:ld      @LEAKAGE_count,b             ; load leakage counter
        sub     #(N_LEAKAGE-1),b             ; apply the leakage ?
        rcd     blt                          ; return if not
         addm    #0001h,@LEAKAGE_count       ; increment the leakage counter
        mpy     *LEAKAGE_ptr,#7fffh,a        ; apply the Leakage
        sth     a,*LEAKAGE_ptr+%             ; store the result and point next weight
        retd                                 ; return
         andm   #0,@LEAKAGE_count            ; reset the leakage counter

;*****************************************************************************
;*****************************************************************************
;************************** A-LAW OMPANDER ***********************************
;*****************************************************************************
;*****************************************************************************
;
;                 /--------------\
;                 |              |
;LINEAR_in -----> |   Compress   |-----> PCM_out
;                 |              |
;                 \--------------/
;
;                 /--------------\
;                 |              |
;   PCM_in -----> |  DeCompress  |-----> LINEAR_out
;                 |              |
;                 \--------------/
;
;
;*****************************************************************************
;************************        COMPRESSION ROUTINE    ************************
;********************** LINEAR (13 bit) to PCM A-Law (8 bit) *********************
;*****************************************************************************
;NOTE: all the variables use in this piece of code MUST be in the SCRATCH !!!

Compress:    ld       #LINEAR_in,dp          ; set DP
             ld       @LINEAR_in, 16, A      ; A-LAW COMPRESS # IN ACCH
             bc       NEGCMP, ALT            ; jump if negative
             bc       D_zero,ANEQ            ; check if Linear in is 0
             or       #0001h,a               ; force 0x0001
D_zero       exp      A                      ; T = shift
             ld       #-7, B                 ; B = -7
             add      T, B                   ; B = T - 7
             bc       SEGNZ1, BLT            ; Chck for PCM segment 0

             sftl     A,-4                   ; shift to eliminate low 4 bit
             xor      #00055h, 16, A         ; apply MiMi coding
             retd                            ; return
              mvmd     AH,@PCM_out           ; store the result
SEGNZ1:      ld       AH, TS, A              ; align Q
             and      #3c00h, A              ; mask Q
             sub      BL, 14, A              ; add the MSb
             xor      #00055h, 10, A         ; apply MiMi coding
             retd                            ; return
               sth    A,6,@PCM_out           ; store the result
               nop

NEGCMP:      abs      A                      ; Linear is negative
             exp      A                      ; T = shift
             ld       #-7, B                 ; B = -7
             add      T, B                   ; B = T - 7
             bc       SEGNZ2, BLT            ; check for segment 0

             sftl     A,-4                   ; shift to eliminate low 4 bit
             xor      #000d5h, 16, A         ; apply MiMi coding
             retd                            ; return
              mvmd     AH, @PCM_out          ; store the result

SEGNZ2:      ld       AH, TS, A              ; align Q
```

```
            and     #3c00h, A               ; mask Q
            sub     BL, 14, A               ; add MSb
            xor     #000d5h, 10, A          ; apply MiMi coding
            retd                            ; return
              sth   A,6,@PCM_out            ; store the result
              nop


;*******************************************************************************
;************************       DE-COMPRESSION ROUTINE        ************************
;************************ PCM A-Law (8 bit) to LINEAR (13 bit) ************************

;*******************************************************************************
;NOTE: all the variables use in this piece of code MUST be in the SCRATCH !!!

Decompress:
        ld      #LINEAR_in,dp               ; set DP
        st      #PCM_table, BADDR           ; BADDR = PCM Table base address
        stm     #15-7, T

        ld      @PCM_in, A
        xor     #00055h, A                  ; invert bits (MiMi)
        bitt    AL                          ; detect sign
        and     #0007fh, A
        add     @BADDR, A                   ; add table base addr
        reada   @LINEAR_out                 ; Read result into X
        rc      NTC                         ; return if positive
        ld      @LINEAR_out, A              ; AL = lookup value
        retd                                ; delayed return
         neg    A                           ; negate result
         stl    A,@LINEAR_out               ; store result

;*******************************************************************************
;*******************************************************************************
;*****                          VECTORS TABLE                                ****
;*******************************************************************************
;*******************************************************************************

 Stack setup

BOS     .usect  "stack", 0Fh
TOS     .usect  "stack", 1

        .sect   "vectors"

reset:  BD      Start
        STM     #TOS, SP
NMI:    BD      Start
        STM     #TOS, SP
        .align  64

extint1:rete
        NOP
        NOP
        NOP
extint2:RETE
        NOP
        NOP
        NOP
extint3:RETE
        NOP
        NOP
        NOP
timint: RETE
        NOP
        NOP
        NOP
sprxint0:
        ret
        NOP
        NOP
        NOP
sptxint0:
        ret
        NOP
        NOP
        NOP
sprxint1:
        RETE
        NOP
        NOP
        NOP
sptxint1:
        RETE
```

```
        NOP
        NOP
        NOP
extint4:RETE
        NOP
        NOP
        NOP

 ;*****************************************************************************
 ;*****************************************************************************

;         TABLE FOR A-LAW EXPANSION LOOKUP with 128 entries

        .sect "tables"

PCM_table:
        .word 8
        .word 24
        .word 40
        .word 56
        .word 72
        .word 88
        .word 104
        .word 120
        .word 136
        .word 152
        .word 168
        .word 184
        .word 200
        .word 216
        .word 232
        .word 248
        .word 264
        .word 280
        .word 296
        .word 312
        .word 328
        .word 344
        .word 360
        .word 376
        .word 392
        .word 408
        .word 424
        .word 440
        .word 456
        .word 472
        .word 488
        .word 504
        .word 528
        .word 560
        .word 592
        .word 624
        .word 656
        .word 688
        .word 720
        .word 752
        .word 784
        .word 816
        .word 848
        .word 880
        .word 912
        .word 944
        .word 976
        .word 1008
        .word 1056
        .word 1120
        .word 1184
        .word 1248
        .word 1312
        .word 1376
        .word 1440
        .word 1504
        .word 1568
        .word 1632
        .word 1696
        .word 1760
        .word 1824
        .word 1888
        .word 1952
        .word 2016
        .word 2112
        .word 2240
        .word 2368
        .word 2496
```

```
        .word 2624
        .word 2752
        .word 2880
        .word 3008
        .word 3136
        .word 3264
        .word 3392
        .word 3520
        .word 3648
        .word 3776
        .word 3904
        .word 4032
        .word 4224
        .word 4480
        .word 4736
        .word 4992
        .word 5248
        .word 5504
        .word 5760
        .word 6016
        .word 6272
        .word 6528
        .word 6784
        .word 7040
        .word 7296
        .word 7552
        .word 7808
        .word 8064
        .word 8448
        .word 8960
        .word 9472
        .word 9984
        .word 10496
        .word 11008
        .word 11520
        .word 12032
        .word 12544
        .word 13056
        .word 13568
        .word 14080
        .word 14592
        .word 15104
        .word 15616
        .word 16128
        .word 16896
        .word 17920
        .word 18944
        .word 19968
        .word 20992
        .word 22016
        .word 23040
        .word 24064
        .word 25088
        .word 26112
        .word 27136
        .word 28160
        .word 29184
        .word 30208
        .word 31232
        .word 32256

        .end
```

Appendix B. The linker command file

```
/*******************************************************

        LINKER command file for Echo Canceller

*******************************************************/

-e Start
-asg


MEMORY
{
  PAGE 0: VECS (RWX)  : origin = 00080h  length = 00080h  /* reset vector 128W */
          ROM  (RWX)  : origin = 00100h  length = 00f00h  /* internal RAM      */

 PAGE 1:  SCRATCH (RW): origin = 00060h  length = 00020h  /* scratch  - 32W */
          RAM1 (RWX)  : origin = 01000h  length = 00400h  /* block 1 - 1KW */
          RAM2 (RWX)  : origin = 01400h  length = 00400h  /* block 2 - 1KW */
          RAM3 (RWX)  : origin = 01800h  length = 00400h  /* block 3 - 1KW */
          RAM4 (RWX)  : origin = 01c00h  length = 00400h  /* block 4 - 1KW */
          RAM5 (RWX)  : origin = 02000h  length = 00400h  /* block 5 - 1KW */

}

SECTIONS
{
        vectors         >       VECS     page   0
        program         >       ROM      page   0       /* code */
        tables          >       ROM      page   0       /* tables */

        scratch         >       SCRATCH page    1       /* scratch used by compander */
        in_buff         >       RAM2     page   1       /* input buffer */
        filter          >       RAM3     page   1       /* weights */
        .bss            >       RAM5     page   1
        stack           >       RAM4     page   1       /* stack */
}
```

Appendix C. The simulator command file

```
;                LEAD Simulator startup command file
;                ===================================
; ========================================================================
;|This file contains 'ma' (map add) commands to define some memory maps. |
;|On startup only the internal PROG/DATA memory are configured.          |
; ========================================================================

mr

ma 0x0080, 0, 0x0380, r|w
ma 0x0400, 0, 0x0400, r|w
ma 0x0800, 0, 0x0400, r|w
ma 0x0c00, 0, 0x0400, r|w
ma 0x1000, 0, 0x0400, r|w
ma 0xfd00, 0, 0x00ff, r|w
ma 0xff00, 0, 0x0080, r|w
ma 0xff80, 0, 0x0080, r|w

ma 0x0000, 1, 0x0060, r|w
ma 0x0060, 1, 0x0020, r|w
ma 0x0080, 1, 0x0f80, r|w
ma 0x1000, 1, 0x0400, r|w
ma 0x1400, 1, 0x0400, r|w
ma 0x1800, 1, 0x0400, r|w
ma 0x1c00, 1, 0x0400, r|w
ma 0x2000, 1, 0x0400, r|w

map on

alias  l,"take siminit.cmd"
alias  r,"reset"
alias  g,"go Start"
alias  f,"fill 0x400,1,0x800,0x00,fill CONST,1,0x010,0x00"
alias  fm,"fill 0x400,1,0x800,0x00"
alias  fc,"fill CONST,1,0x010,0x00"

E PMST=0X00A0

MA 0X1000, 2, 0X0002, R|P                       ;
MA 0X2000, 2, 0X0002, W|P                       ;

MC 0X1000, 2, 0X0001,  FAR_END.INP, R|NR        ; Far End input file  (Rout)
MC 0X1001, 2, 0X0001, NEAR_END.INP, R|NR        ; Near End input file (Sin)

MC 0X2000, 2, 0X0001, E_OUT.OUP, WRITE          ; Far End output file (Sout)
MC 0X2001, 2, 0X0001, Y_OUT.OUP, WRITE          ; Adaptative FIR output file (debug)

LOAD ECHO_AN.OUT

E PC=reset

mem  BUFFER_IN
mem1 WEIGHT

wa *X_in         ,X_in....,x
wa *V_in         ,V_in....,x
wa *Yn           ,Yn......,x
wa *En           ,En......,x
wa *LEAKAGE_count,LEAKcnt.,d

wa *LINEAR_in    ,LIN.in..,x
wa *PCM_out      ,PCM.out.,x
wa *PCM_in       ,PCM.in..,x
wa *LINEAR_out   ,LIN.out.,x

wa *SAMPLE_cnt   ,SAMPLE.#,d

wa clk           ,uC......,d
```