Packet Telephony Remote Diagnostics on the StarCore SC140 Core

by Lúcio F.C. Pessoa, Robert Barrett, Raquel Flores, and Kim-chyan Gan

This application note presents an intrusive remote diagnostic device that dynamically estimates and emulates hybrid circuits in packet telephony systems. The device uses a robust FFT-based channel identification scheme common to asymmetric digital subscriber line (ADSL) systems to estimate the actual impulse response h_{est} of a hybrid circuit. It then dynamically emulates a target hybrid circuit impulse response $h_t(n)$ by injecting an echo signal based on the effective impulse response $h_{eff}(n) = h_t(n) – h_{est}$. Using this method, any linear time-invariant hybrid circuit can efficiently be transformed to have any target impulse response, thereby enabling dynamic correction of poorly designed hybrid circuits or cascaded connections of hybrid circuits, which generate multiple reflections. This method was implemented and extensively tested in real time on a Freescale StarCore™-based DSP device. The resulting implementation requires less than 2.0 million cycles per second (MCPS) on average for a target impulse response with up to a 128 ms span. The remote diagnostic device was validated with a carrier-class network echo canceller, which emulated various target impulse responses, regardless of the actual hybrid circuit(s) in the system. The remote diagnostic device was demonstrated to be a valuable asset for developing and deploying packet telephony systems, especially when proper control is provided over the communication network

CONTENTS

1 Basics of Remote Diagnostics 2
2 Remote Diagnostic Architecture 5
2.1 Estimating the FFT-Based Hybrid Circuit Impulse Response 6
2.2 Generating the Training Signal 6
2.3 Transmitting the Training Signal 7
2.4 Calculating the Impulse Response 8
2.5 Emulating Target Impulse Responses 8
2.6 Considerations for Real-Time Implementation 8
2.7 Typical Results 9
3 Remote Diagnostics on StarCore 11
3.1 System-Level Optimization 11
3.2 Kernel-Level Optimization 12
4 Conclusion 13
5 References 13

1 Basics of Remote Diagnostics

Remote diagnostic tests are becoming important maintenance applications in packet telephony systems for monitoring and sometimes correcting anomalies that affect performance of voice or data communication services [1], [2]. Maintenance tests are performed either in-service while other users are accessing the same channel of the communication network or out-of-service (same channel not in use). The tests employ either intrusive methods that change transmit and/or receive signals or non-intrusive methods that do not change the signals.

The International Telecommunication Union (ITU) developed the P.561 Recommendation [3], which defines preferred interfaces, measurement ranges, and accuracy requirements for measuring voice-grade transmission parameters in the communication network. This recommendation is intended primarily for in-service non-intrusive measurement devices (INMDs). Typical P.561 measurements include speech level, noise level, echo loss, and speech echo path delay; optional measurements, such as double talk, signal classification, and speech activity factors, are sometimes needed.

Among the transmission parameters, echo is one of the most relevant impairments affecting quality of service. Echo is caused by a hybrid network element, which is a circuit to convert a four-wire physical interface to a two-wire connection. The main role of this hybrid is to provide an electrical interface for signals traveling in both transmit and receive directions at the same time. The hybrid is designed to minimize echo (reflection) of the receive signal, but in most practical cases the echo becomes quite noticeable, especially when communication delay is large. Therefore, an echo canceller is required to mitigate this kind of distortion [4].

Hybrid circuits are typically modeled as linear time-invariant systems so that their signal processing characteristics can be determined by estimating impulse responses. Estimates usually employ finite impulse response (FIR) filters, as detailed in the ITU recommendation G.168 [5], which defines eight basic FIR hybrid models for testing network echo canceller devices. When the impulse response is known (that is, coefficients of the FIR filter), the echo signal of a hybrid circuit can be mimicked by a linear convolution of the receive signal and the hybrid impulse response estimate, which then cancels the echo component of the signal. This is the basic foundation of echo cancellers, but additional nonlinear processing techniques are used to handle residual echo.

Echo cancellers are deployed in media gateways, which are platforms for handling communications between IP packet-based networks and circuit-switched networks. However, some network configurations may violate certain design assumptions, resulting in degraded echo canceller performance and customer complaints. To resolve configuration problems, expensive measurement devices are brought to the field to estimate telephone channel characteristics. A far more cost effective approach is to design the media gateway with a software module that is efficiently programmed to perform reliable measurements. A protocol is defined to transmit this measurement data to a remote media gateway controller (see **Figure 1**) so that specialized engineers can retrieve and analyze it.

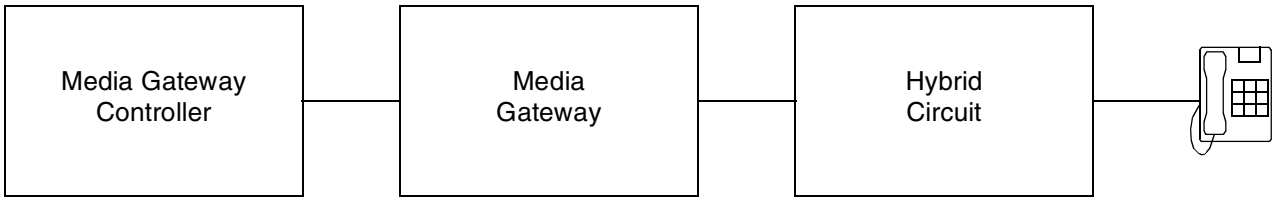


Figure 1. Portion of a Packet Telephony System

The method discussed here for estimating and dynamically emulating impulse response of hybrid circuits is motivated by channel estimation techniques in ADSL systems [5], which have been extensively employed in the field and proven to be accurate and reliable. The proposed method is part of a hybrid circuit identification device (HCID) illustrated in **Figure 2**.

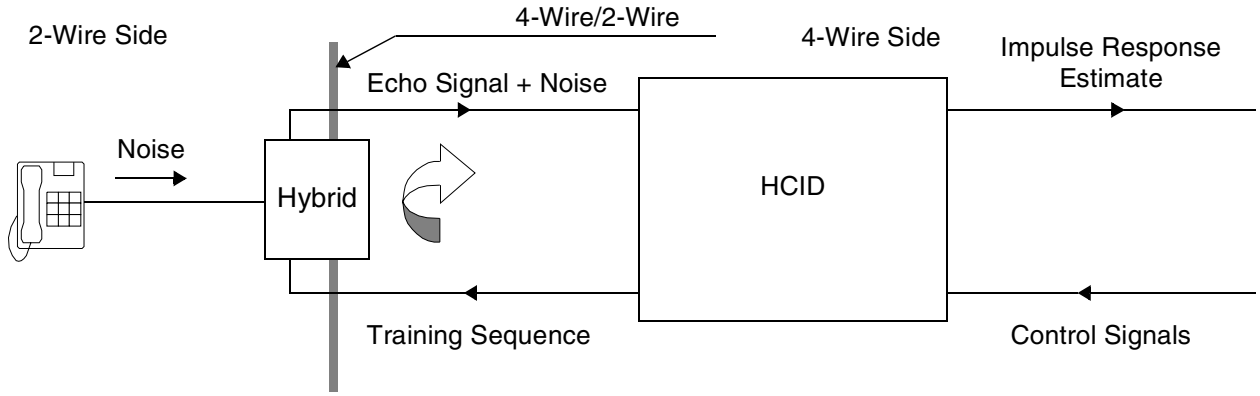


Figure 2. Hybrid Circuit Identification Device (HCID).

The HCID unit is an intrusive measurement component of the remote diagnostic device that uses control signals to specify how the identification process is performed. Upon activation, a periodic training sequence is transmitted to the hybrid and its reflection is captured. During this process, uncorrelated spurious noise may be injected, such as audio signals from a telephone unit connected to the hybrid, but the HCID can usually mitigate the noise, even if the training sequence and the spurious noise coexist as double talk. The goal of the HCID is to provide accurate impulse response estimates of the hybrid circuit.

The HCID unit can connect to multiple hybrid circuits, using time-division multiplexing (TDM) and different channel data states to control its operation. **Figure 3** shows a prototype of a multi-channel device using the MSC8101ADS board [7]. Multiple hybrid circuits connect to the ADS board via a T1 line. Software is loaded through the debug port from a personal computer; control signals are provided, and impulse responses are obtained via RS-232. The MSC8101 DSP device digitally generates the training signal to estimate hybrid circuits. This simple prototype can be used to develop an optimized software module for integration on the packet telephony system illustrated in **Figure 1**.

A similar configuration for non-intrusive measurements uses an adaptive filter structure that operates in parallel with an echo canceller infrastructure. Non-intrusive methods are not considered here, but alternatives are readily available in the literature (see [2]). Echo cancellers can be equipped with a built-in debugging/testing feature that is selectively enabled to estimate most required measurements during regular in-service operation and pass these estimates to the media gateway as needed. After a physical hybrid impulse response is estimated, the user can selectively emulate another hybrid circuit by compensating for the existing echo and injecting a desired echo signal. Intrusive methods are usually required to perform hybrid circuit emulation, although non-intrusive methods can be selectively employed. Hybrid emulation proceeds as follows:

1. Estimate the impulse response of a hybrid circuit.
2. Use the estimated impulse response to estimate the echo signal generated by the hybrid circuit.
3. Subtract the estimated echo signal from the actual echo signal to generate a residual echo signal.
4. Use a target impulse response to generate a target echo signal.
5. Add the target echo signal to the residual echo signal to generate an emulated echo signal.

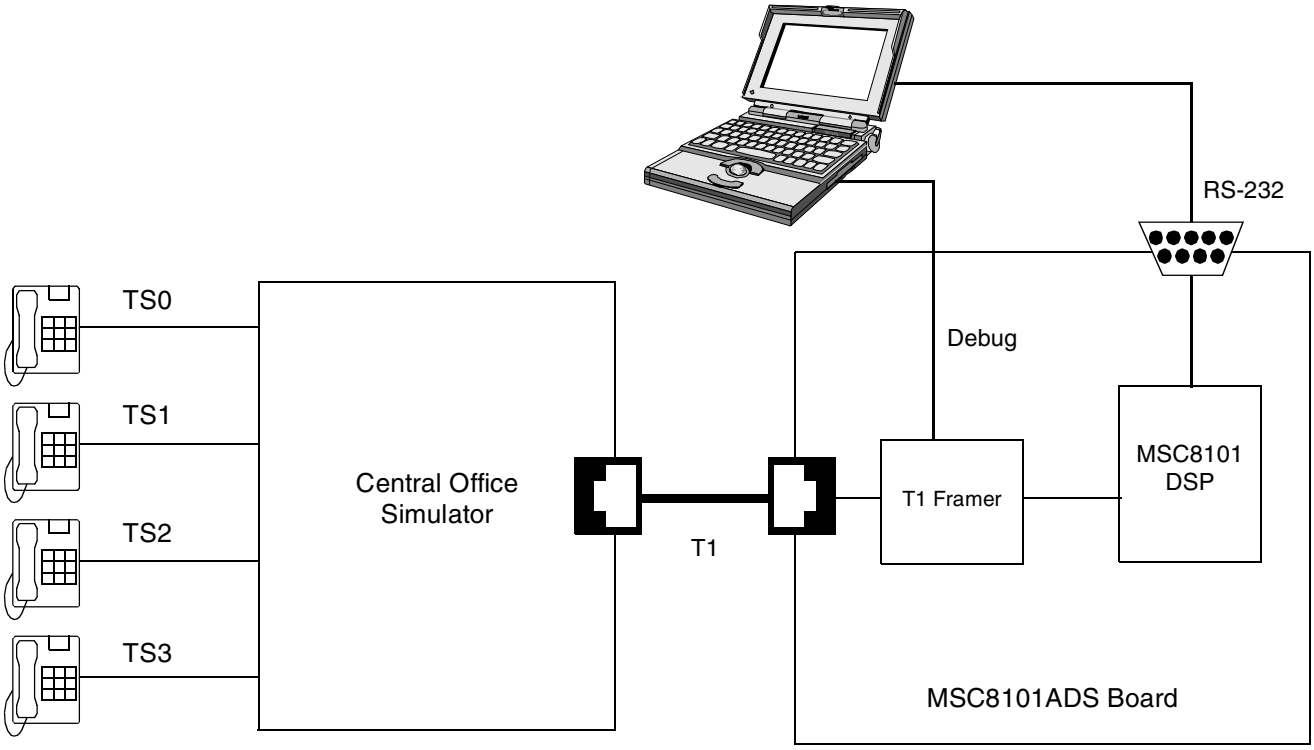
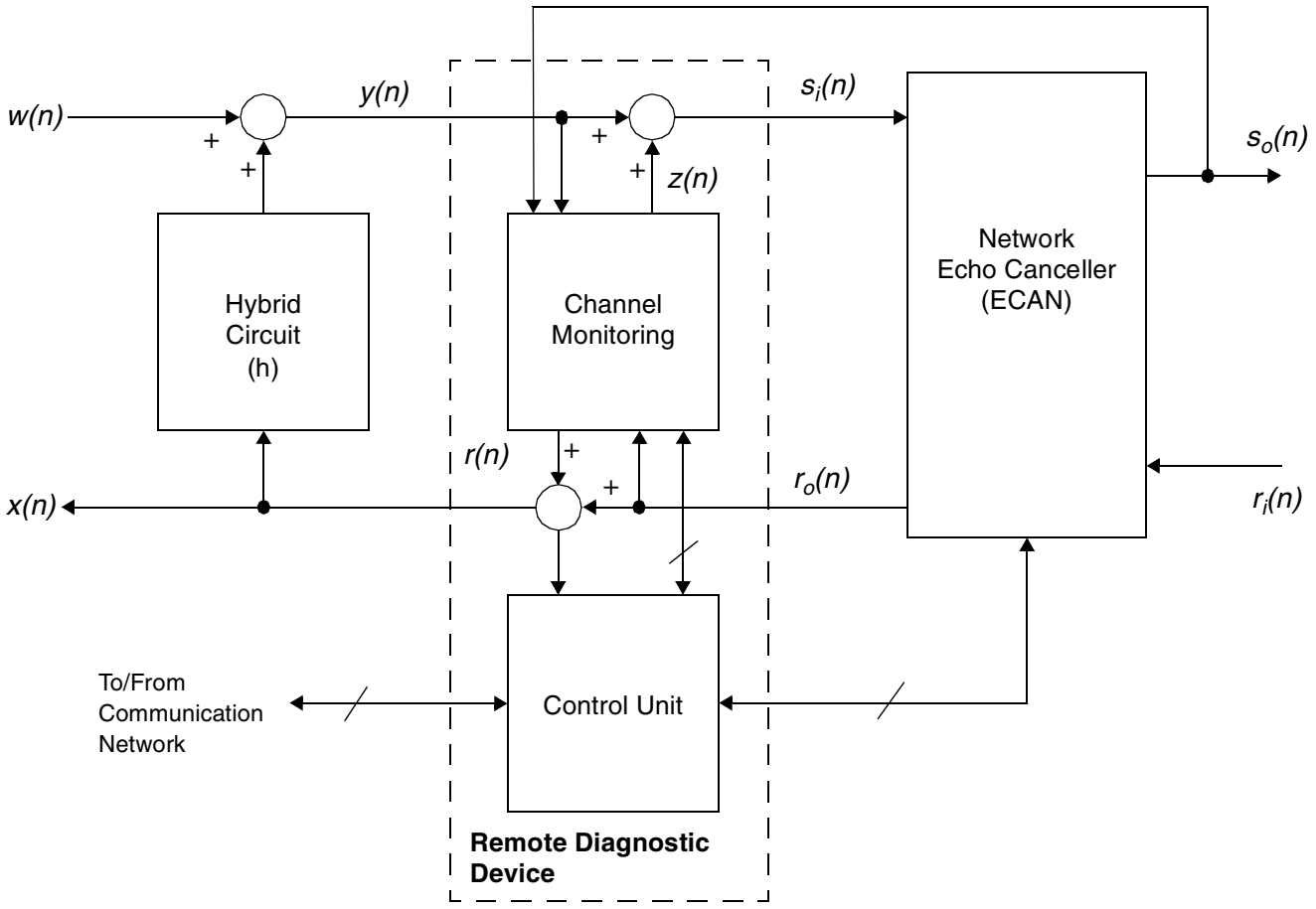


Figure 3. Multi-Channel HCID Implemented on the MSC8101ADS Board

2 Remote Diagnostic Architecture

Figure 4 shows a top-level view of the remote diagnostic device, with respect to the hybrid circuit and network echo canceller (ECAN). The device monitors the $r_o(n)$, $y(n)$, and $s_o(n)$ signals for performing various measurements requested from the communication network.



Legend

$r_o(n)$ Far-end receive signal from the ECAN.

$s_o(n)$ Sending signal from the ECAN.

$x(n)$ Training signal ($r(n) + r_o(n)$).

$y(n)$ Receive signal from the hybrid ($w(n) + h * x(n)$).

Figure 4. Top-Level Architecture of the Proposed Remote Diagnostic Device

2.1 Estimating the FFT-Based Hybrid Circuit Impulse Response

The impulse response of hybrid circuits is estimated with an FFT-based channel identification scheme similar to the method used in ADSL systems, as shown in **Figure 5**.

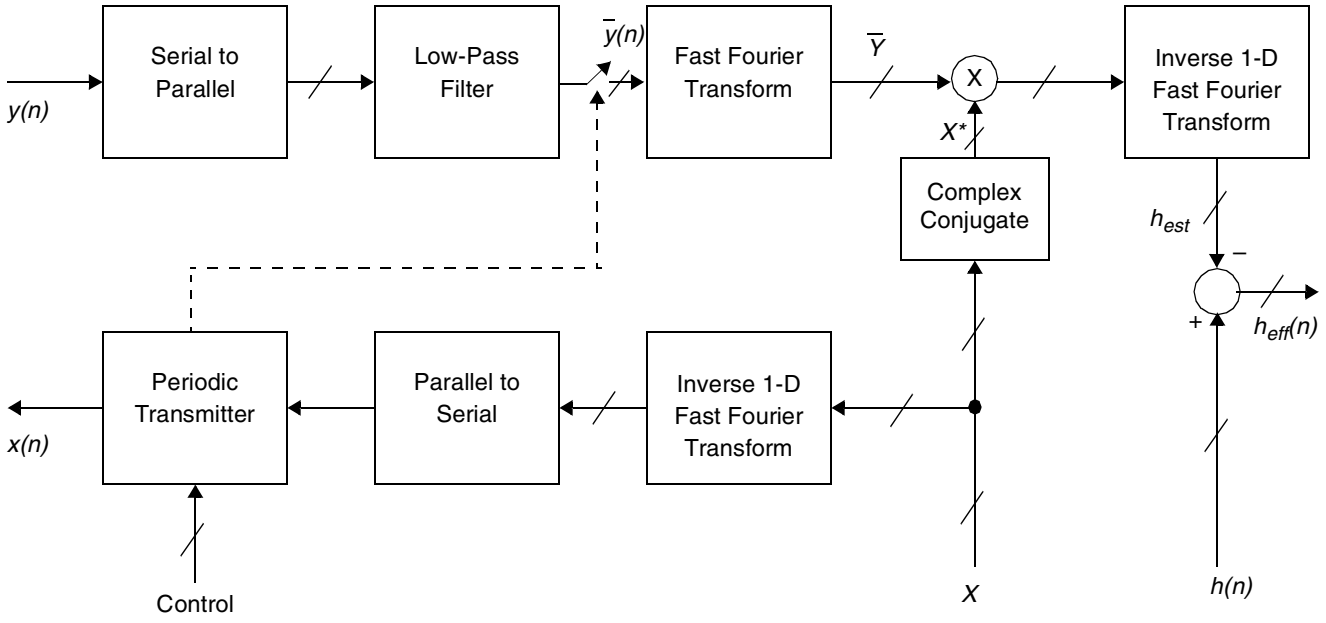


Figure 5. FFT-Based Structure for Estimating Hybrid Circuit Impulse Responses

2.2 Generating the Training Signal

The training signal is generated by a white Gaussian noise signal *x(n)* that is a function of uniformly distributed random variables Φ_k's in the frequency domain, as shown in **Equation 1**.

Equation 1

$$X(k) = e^{j \cdot \Phi_k}, k = 1, ..., \frac{N}{2} - 1, \text{ where } \Phi_o = \Phi_{N/2} = 0 \text{ and } X_k = X_{N-k}^*$$

Converting this signal into the time domain by taking the inverse discrete Fourier transform results in the signal shown in **Equation 2**.

Equation 2

$$x(n) = \frac{1 + (-1)^n}{N} + \frac{2}{N} \cdot \sum_{k=1}^{\frac{N}{2}-1} \cos\left(\frac{2\pi \cdot kn}{N} + \Phi_k\right), \qquad n = 0, 1, \ldots, N-1$$

where *N* is the period of the training signal. This real-valued signal is generated in the code. Notice that this training signal is obtained by setting $r(n) = x(n) - r_0(n)$ (see **Figure 4**). Before transmission, the signal can be adjusted to a convenient power level. This signal is generated on the basis of the two main parts of the equation: a method for efficiently computing cosines and a random number generator to produce the random phases:

$$\Phi_k, \qquad k = 1, \ldots, \frac{N}{2} - 1$$

Obtaining the Φ_k values requires the use of a random number generator. A standard code to generate random numbers can be found in [8]. This reference provides a number of random number generators, from which uniform and identically distributed random variables are easily generated. The values for Φ_k vary between 0 and 2π, but they are transformed as follows:

Equation 3

$$\cos\left(\frac{2\pi \cdot k \cdot n}{N} + \Phi_k\right) = \cos\left[\frac{2\pi}{N}(kn + l_k)\right], \text{ where } l_k = \left\lfloor \frac{N}{2\pi} \cdot \Phi_k \right\rfloor$$

such that l_k is a discrete, uniformly distributed random variable with values *0,1, . . . ,N–1*. The rotation matrix recurrence method is an efficient way to calculate cosines, as shown in **Equation 4**.

Equation 4

$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} c(n-1) \\ s(n-1) \end{bmatrix} = \begin{bmatrix} c(n) \\ s(n) \end{bmatrix}, \qquad \text{where } \theta = \frac{2\pi}{N}$$

and $n = 0, 1, \ldots, N-1$, and $c(n)$ and $s(n)$ represent the discrete cosine and discrete sine, respectively. Solving **Equation 4** for $c(n)$ results in the well-known digital oscillator [9].

Equation 5

$$c(n) = 2 \cdot \cos\theta \cdot c(n-1) - c(n-2)$$

Using the initial conditions $c(0) = 1$ and $c(1) = \cos\theta$, all other values of cosine for $n = 0, 1, \ldots N-1$ can be computed efficiently. Due to the even symmetry of the cosine function, only half of these values (that is, $n = 0, 1, \ldots, N/2-1$) are computed and stored in a look-up table for future reference. The values from this look-up table are then used in the following cosine computation:

Equation 6

$$\cos\left[\frac{2\pi}{N}(kn + l_k)\right] = c(kn + l_k)$$

The look-up table is accessed via modular addressing, and the evenness or oddness of the remainder of $(kn + l_k) / (N/2)$ indicates whether the positive or negative value of the cosine is used. An even remainder indicates the positive value of the cosine, and an odd remainder indicates the negative value. Therefore, the training signal $x(n)$ is composed by simple superposition of scaled discrete cosine values taken from a look-up table of $c(n)$, $n = 0, 1, \ldots, N/2-1$.

2.3 Transmitting the Training Signal

After the training signal $x(n)$ is generated, it is periodically transmitted across the channel through the hybrid, and the average of the receive signal (echo signal + noise) is computed. N samples of the transmit signal $x(n)$ make up one transmit frame, and a user-specified number of frames (M) determines how many times the signal is repeatedly transmitted. The buffer storing the signal is accessed circularly. Multiple transmissions of the same data frame are important for mitigating spurious noise and improving estimation accuracy.

When the receive frame is obtained, it is used to compute the average, that is, the low-pass filter (LPF), of the receive signal ($\overline{y}(n)$). In this computation, the receive samples from at least the first two frames and the last two frames are usually ignored due to boundary effects of the linear convolution and do not represent valid circular convolution.

2.4 Calculating the Impulse Response

To find the frequency response of the hybrid impulse response, the discrete Fourier transforms of $x(n)$ and $\overline{y}(n)$ must be computed (that is, X and $\overline{Y}$). A standard code for computing these transforms via FFTs is available in [8], but optimized libraries for StarCore DSP cores are freely available on the Freescale Semiconductor website listed on the back cover of this document. The H_{est} frequency response is the result of dividing $\overline{Y}$ by X. Because the magnitude of the training signal $x(n)$ frequency response is 1 (that is, $|X| = 1$), dividing by X is the same as multiplying by its complex conjugate. Calculating the frequency response this way eliminates costly division operations and therefore is the method used in the code. The h_{est} impulse response estimate is then found by taking the inverse discrete Fourier transform of the H_{est} frequency response.

2.5 Emulating Target Impulse Responses

As discussed in **Section 1**, after a physical hybrid impulse response is estimated, you can selectively emulate another hybrid circuit by compensating the existing echo and then injecting a desired echo signal. As **Figure 4** shows, a target impulse response $h_t(n)$ is easily emulated by defining $z(n) = (h_t(n) - h_{est}) * r_0(n)$, where h_{est} is the estimate of the physical hybrid circuit impulse response. In this process, $h_t(n)$ can be any linear filter, not necessarily time-invariant. This method was validated in real time with a carrier-class network echo canceller using different target impulse responses.

2.6 Considerations for Real-Time Implementation

Implementing FFT-based channel estimation in real time requires careful control of other modules, such as the ECAN. As described in **Section 2.2**, during the training phase, the ECAN must be disabled and $r_0(n)$ must be suppressed. The following algorithm summarizes the resulting steps:

1. For every channel, generate a zero-mean, white Gaussian noise training signal $x(n)$ using the following equation:

Equation 7

$$x(n) = \frac{1 + (-1)^n}{N} + \frac{2}{N} \cdot \sum_{k=1}^{\frac{N}{2}-1} \cos\left(\frac{2\pi \cdot kn}{N} + \Phi_k\right), \qquad n = 0, 1, \ldots N-1$$

   where the Φ_k are independent, identically distributed random variables having uniform distribution between 0 and 2π. ($\Phi_k \sim U(0,2\pi)$, $\forall_k$). The $r_0(n)$ receive signal is suppressed by setting $r(n) = x(n) - r_0(n)$ in **Figure 4** so that the resulting spectrum of $x(n)$ is flat (that is, unit gain per frequency) during the training period ($M \cdot N$ samples), regardless of the spectrum of $r_0(n)$.

2. Periodically transmit $x(n)$, read the receive signal $y(n)$, and compute the averaged receive signal $\overline{y}(n)$, that is, a low-pass filtered version of $y(n)$.

   The $x(n)$ signal is transmitted periodically so that the receive signal $y(n)$ is a circular convolution with the channel impulse response h. This allows use of point-by-point multiplication of discrete Fourier transforms (DFTs) in step 4 to obtain the frequency response of the channel. During the training, the

ECAN must be disabled (see **Figure 4**). Notice that the signals are converted from serial to parallel (S/P) and from parallel to serial (P/S), as needed.

3. Compute the DFT of $x(n)$, which is typically pre-computed, and $\overline{y}(n)$: $X = FFT[x(n)]$ and $\overline{Y} = FFT[\overline{y}(n)]$.
4. Use X and $\overline{Y}$ to compute $H = FFT[h_{est}]$: $H_{est} = \overline{Y} / X = \overline{Y} \cdot X^*$, since $|X|=1$, where $(\cdot)^*$ denotes a complex conjugate.
5. Get the impulse response estimate of the hybrid circuit, which is the channel impulse response (CIR), by taking the inverse discrete Fourier transform of H_{est}: $h_{est} = IFFT[H_{est}]$.

A simple MATLAB script corresponding to these steps is listed as follows (for simplicity, $r_0(n) = 0$, such that $x(n) = r(n)$):

```
N    = 2^10; % Set maximum echo path span to 128 ms
X    = ones(1, N);

X(2 : N / 2) = exp(2 * pi * j * rand(1, N / 2 -1)); % Define Random phase
X(N : -1 : N/2 + 2) = conj(X(2 : N / 2)); % Impose Hermitian symmetry
x    = real(ifft(X)); % Generate the training signal

M    = 100; % Number of frames to be transmitted
xx   = zeros(1, M * N);
for m = 1 : M  % Generate training sequence
      xx((m - 1) * M + (1 : N)) = x;
end

ho   = load('hybrid_impulse_response.txt'); % Reference impulse response

y    = conv(xx, ho); % Generate expected receive signal
y    = y(1, 1 : M * N);
SNR  = 30;  % Target SNR (in dB)
d    = sqrt(mean(y.^2)) * 10^(-SNR / 20) * randn(1, M * N);
yd   = y + d; % Inject additive noise to achieve target SNR

ya   = zeros(1, N);
for m = 1 : M   % Main estimation loop
      ya = ya + ya((m - 1) * M + (1 : N)) / M;
end

Ya   = fft(ya); % Generate the FFT of the average receive signal
hest = real(ifft(Ya .* conj(X))); % Estimate hybrid impulse response

ye   = conv(xx, hest);  % Generate estimated receive signal
ye   = ye(1, 1 : M * N);
e    = yd - ye;  % Estimate residual echo (error signal)
ep   = 10*log10(mean(e.^2)) + 6.2;  % Estimate residual echo power (dBm0)
erle = 20*log10(norm(ho) / norm(ho - hest));  % Estimate ERLE (dB)
```

2.7 Typical Results

Figure 6 and **Figure 7** illustrate typical results obtained with the method described here, using the MATLAB script. **Figure 6** illustrates a multi-window impulse response generated by cascading multiple hybrid circuits. Additive white Gaussian noise is injected at $w(n)$ (see **Figure 4**) to simulate an SNR = 30 dB. The resulting estimate presented an accuracy of at least 9 bits, estimated as follows:

$$\left\lfloor \left| \log_2 \left(\left\| h - h_{est} \right\|_\infty \right) \right| \right\rfloor$$

The training signal is composed of $M = 100$ periods of 128 ms frames at a sampling rate of 8 KHz (that is, $N = 128 \times 8 = 1024$ samples). Two estimates of echo return loss enhancement (ERLE) are also computed, both indicating an ERLE of about 44 dB. **Figure 7** illustrates similar results, but with noise injected to simulate an SNR = 0 dB. Notice that the intrusive method can achieve at least 6 bits of precision and an ERLE of about 20 dB under this high noise condition. Similar performance would be difficult to achieve with non-intrusive standard methods, such as NLMS-based channel estimation techniques.

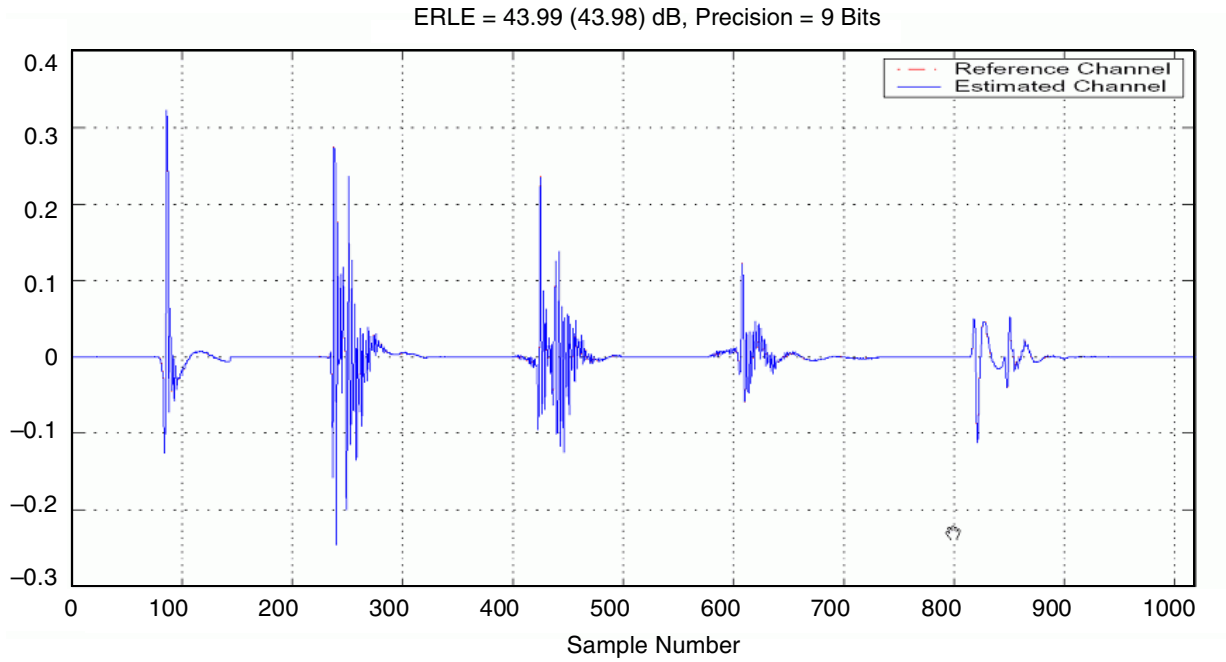


Figure 6. Example of Estimated Multi-Window Impulse Response Under SNR = 30 dB

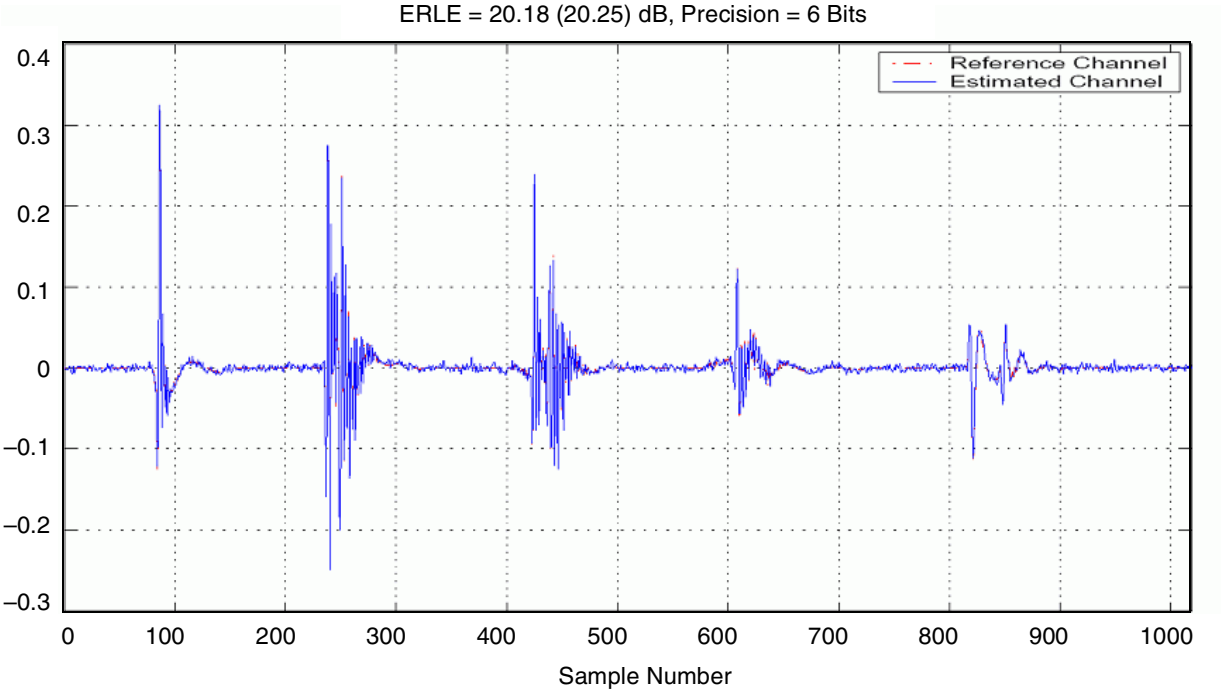


Figure 7. Example of Estimated Multi-Window Impulse Response Under SNR = 0 dB

3 Remote Diagnostics on StarCore

This section describes an implementation of the remote diagnostic architecture presented in **Section 2** on a Freescale StarCore MSC8101 DSP. Extensive real-time tests demonstrated that the remote diagnostic software requires fewer than 2.0 million cycles per second (MCPS), on average, for a target impulse response with up to a 128 ms span. Hybrid circuit impulse response estimation and emulation are a major part of the remote diagnostic functionality, but other basic measurements are included.

A flexible application programming interface (API) dynamically enables/disables hybrid estimation and emulation, providing signal levels in dBm0—such as $r_0(n)$, $s_i(n)$, and $s_0(n)$—and providing hybrid impulse response information, such as ERL, bulk delay, and active portion of the impulse response. These are a small subset of the possible remote diagnostic features in comparison to those discussed in [3], but they are critical indicators of possible anomalies in the communication network. Proprietary schemes integrated in a media gateway system can use these measurements to monitor ECAN performance (see **Figure 4**). Alert messages are generated when troubling conditions are detected, such as a level of $s_0(n)$ much larger than the level of $s_i(n)$, which is inconsistent with expected ECAN behavior. Furthermore, when the remote diagnostic device is integrated within a packet telephony development platform such as the prototype illustrated in **Figure 3**, it can emulate any target hybrid impulse response—for example, one of the 8 G.168 hybrid models. It can also emulate time-varying impulse responses, thus providing valuable help in acoustic echo canceller development and testing.

The FFT and inverse 1-D fast Fourier transform (IFFT) processing block handled most of the required processing load, so it is discussed further. We also used other common code optimization strategies, such as precomputing the training signal and storing it in look-up tables and efficient multi-sample processing of the FIR filtering step [10] required for echo estimation and emulation. High-level C optimization of the FFT/IFFT increased the efficiency of the impulse response estimation process. The speed-up efforts were conducted in two phases: system-level optimization and kernel-level optimization.

3.1 System-Level Optimization

The original FFT/IFFT was performed in floating-point operations, which are not very efficient on fixed-point processors. Therefore, we converted the FFT/IFFT block to fixed-point, imposing precision limitations. To validate the functionality of the new code with ease, we defined a preprocessor so that the user can select either the fixed- or floating-point FFT/IFFT implementation.

```
#define MOT_REM_DIAG_FFT_FIX16  1       // Enable(1) -> use 16-bit fix-pointed FFT
                                        // Disable(0) -> use floating point FFT
```

During the channel estimation step, a random sequence with known probability distribution is generated in real-time and periodically transmitted through the hybrid circuit. A new training signal is generated per channel estimation task. However, the training signal can instead be precomputed, in both the time and frequency domains, and stored in look-up tables. If look-up tables are in use, the random sequence generator can be turned off, reducing the total number of FFT/IFFT operations from three to two. A preprocessor macro is defined so that the user can choose the fixed- or real-time generated random sequence.

```
#define MOT_REM_DIAG_FIX_TRAINING_SEQ 1 // Enable(1) -> use fixed training sequence
                                        // Disable(0) -> use real-time self-generate
                                        // training sequence for every frame
```

During FFT/IFFT operations, the received signal and estimated channel impulse response are purely real signals; that is, their imaginary portions are all zeroes. To increase computational efficiency, the real data set can be split into two subsets [8], [9] for the real and imaginary parts, forming an equivalent complex data of half the original size. Then a complex FFT/IFFT of half the original data size can be performed. The result is pre- and post-processed to recover the original real data from the complex data.

3.2 Kernel-Level Optimization

The FFT/IFFT bit-reverse operations use a look-up table for in-place bit-reverse processing. Only values that must be swapped are stored in the look-up table. The real and imaginary parts of the signal are 16-bit data, so the whole complex data points are stored in 32-bit groups. In other words, the data swap is performed in 32-bits. The bit-reverse operation can also be implemented through the Metrowerks® compiler-specific `br_inc()` macro. This macro uses the hardware bit-reverse mechanism provided by the SC140 DSP core to change the index through bit-reverse operations. Unfortunately, the in-place bit-reverse operation cannot be implemented directly. An additional buffer with specific alignment must be allocated. This approach speeds up the bit reverse operation by approximately 170 cycles per 10 ms frames at the expense of data placement flexibility and an additional 32 bytes. The FFT implementation performs the radix-2 decimation-in-time (DIT) computation with a butterfly kernel operation illustrated in **Figure 8**.

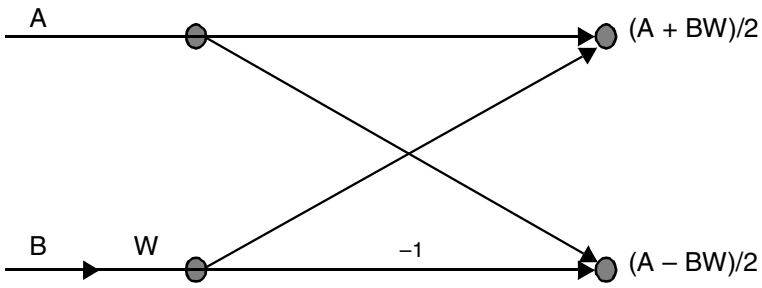


Figure 8. Radix-2 Decimation-in-Time (DIT) FFT Butterfly Using In-Place Computation

In assembly-level optimization, two butterflies can be computed in the kernel. In C-level implementation, only one butterfly is computed in the kernel. The butterfly processing steps are rearranged so that the overall computation is reduced from six to four (see **Table 1**). Only operations on the real portion of the complex data are considered. The imaginary portion has the same number of operations.

Table 1. Butterfly Computation of the Real Portion of Complex Data

Optimized Computation	Instructions	Normal Computation	Instructions
A + BW	MAC, MAC	A + BW	MAC, MAC
(A + BW)/2	SHR	A – BW	MSU, MSU
(A – BW)/2 [A– (A + BW)/2]	SUB	(A + BW)/2	SHR
		(A – BW)/2	SHR

Another portion of the kernel decomposes N/2-point complex FFT/IFFT data into N-point real FFT/IFFT data. The decomposition formula is presented **Equation 8**. F_k is the FFT of the original real data and G_k is the FFT of the complex data corresponding to half the real data size.

Equation 8

$$F_k = \frac{1}{2}(G_k + G^*_{N/2-k}) + \frac{j}{2}(G^*_{N/2-k} - G_k)e^{-j2\pi k/N}$$

It is much more efficient to compute two data points in parallel. Another point of computation is *N/2 – k*, which shares the same twiddle factor as point *k*:

Equation 9

$$F_{N/2-k} = \frac{1}{2}(G_{N/2-k} + G^*_k) + \frac{j}{2}(G^*_k - G_{N/2-k})e^{-j2\pi(N/2-k)/N}$$

Finally, the FFT/IFFT kernel has a triple nested loop, one loop for looping through the number of processing nodes within a butterfly set, another for processing through the number of butterfly sets, and a third for looping through the number of FFT/IFFT stages, which depends on the number of computation points. A pragma statement provides the loop counter boundary range information to the compiler for more aggressive loop optimization.

```
#pragma loop_count (1,64)
```

4 Conclusion

Remote diagnostics are an important maintenance application in packet telephony systems for monitoring, and sometimes correcting, anomalies affecting performance of voice or data communication services. This application note presents an intrusive remote diagnostic device that dynamically estimates and emulates hybrid circuits in packet telephony systems. This device uses a robust FFT-based channel identification scheme to estimate the actual impulse response h_{est} of a hybrid circuit and then dynamically emulate a target hybrid circuit impulse response $h_t(n)$ by injecting an echo signal based on the effective impulse response $h_{eff}(n) = h_t(n) – h_{est}$. Typical performance results for this remote diagnostic device demonstrate the robustness of its contribution in packet telephony systems. Finally, selected code optimization strategies make effective use of the key parallel architecture features of the StarCore DSP. The remote diagnostic device was efficiently implemented and extensively tested in real time on the Freescale StarCore-based DSP, requiring less than 2.0 million cycles per second (MCPS), on average, for a target impulse response of up to 128 ms span. The remote diagnostic device was validated with a carrier-class network echo canceller as it emulated various target impulse responses, regardless of the actual hybrid circuit(s) in the system. This device is demonstrated to be a valuable component in developing and deploying packet telephony systems, especially when proper control is provided over the communication network

5 References

[1] M. Bertocco and P. Paglierani, “In-Service Nonintrusive Measurement of Echo Parameters in Telephone-Type Networks,” *IEEE Transactions on Instrumentation and Measurement*, vol. 47, no. 5, pp. 1322–1325, 1998.

[2] T. Gänsler and G. Salomonsson, “Nonintrusive Measurements of the Telephone Channel,” *IEEE Transactions on Communications*, vol. 47, no. 1, pp. 158–167, 1999.

[3] ITU-T, Recommendation P.561: *In-Service Non-intrusive Device–Voice Service Measurements*, 2002.

[4] R. A. Dyba, P. P. He, and L. F. C. Pessoa, *Network Echo Cancellers and Motorola Solutions Using the StarCore SC140 core*, Freescale Application Note, AN2598/D, 2004.

[5] ITU-T, Recommendation G.168: *Digital Network Echo Canceller*, 2002.

[6] ANSI, “Network and Customer Installation Interfaces: Asymmetric Digital Subscriber Line (ADSL) Metallic Interface,” *American National Standard for Telecommunications*, no. T1E1.413, 1998.

[7] *MSC8101 Application Development System User’s Manual*, Freescale Semiconductor, 2001.

[8] W. H. Press, et al., *Numerical Recipes in C++: The art of Scientific Computing*, Cambridge University Press, Feb. 2002.

[9] A. V. Oppenheim, R.W. Schafer, and J. R. Buck, *Discrete-Time Signal Processing*, Prentice Hall, 2nd edition, 1999.

[10] E. Roy and D. Crawford, *Introduction to the StarCore SC140 tools: An approach in Nine Exercises*, Freescale Semiconductor, Application Note, AN2009/D, 2001.

NOTES:

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations not listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GMBH
Technical Information Center
Schatzbogen 7
81829 München, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Technical Information Center
3-20-1, Minami-Azabu. Minato-ku
Tokyo 106-8573, Japan
0120 191014 or +81-3-3440-3569
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T. Hong Kong
+800 2666 8080

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

AN2832
Rev. 1
9/2004

freescale™
semiconductor