Freescale Semiconductor
Application Note

AN2824
Rev. 1, 9/2004

Noise Reduction on StarCore-Based DSP Devices

by Kim-chyan Gan and Roman A. Dyba

High-level background noise in a telecommunications channel may degrade in-band signaling and lower the perceived voice quality of speech signals. To ensure quality of service in voice-band transmission, it is essential to minimize the degradation caused by the background noise. Various noise reduction techniques, which are effective for a large class of signals, can be used. The challenge is to reduce the noise to a satisfactory level while minimizing the use of computational resources. This application note describes an efficient noise reduction algorithm and its implementation on a high-performance Freescale DSP based on the StarCore™ SC140 core. The noise reduction component is intended for deployment in tandem with an echo canceller, an automatic level controller, and a vocoder, which are essential parts of a media gateway in a Voice-over-IP (VoIP) system environment. The synergy of individual components can be leveraged by sharing the voice activity detector (VAD) between the vocoder and the noise reduction component. The implementation methodology of noise reduction takes advantage of the StarCore architecture and Metrowerks® development tools to reduce engineering efforts. After the code is optimized via an exploratory mix of high-level and machine-level languages, the computational cost of noise reduction is reduced to approximately 0.56 millions of cycles per second (MCPS) in the worst-case scenario. We evaluated the algorithm using different approaches, including subjective evaluation by expert listeners. An overall noise reduction of 10–12 dB was achieved for most natural-speech signals polluted with noise.

CONTENTS

1 Basics of Noise Reduction

High-level background noise pollutes speech signals transmitted through wired and wireless packet networks. It degrades the voice quality or even, in severe cases, renders the speech unintelligible. The background noise also affects in-band signaling and therefore degrades overall voice-band communication. To alleviate this problem, noise reduction techniques can be used to enhance speech quality and improve the signal-to-noise ratio (SNR) of noisy signals. Noise reduction (NR) algorithms, also called noise suppression algorithms, are based on methods such as linear and nonlinear spectral subtraction, [5], [12], Wiener filtering, [6], the wavelet transform, [11], and other methods (see [6], [7], [8], [4]).

The noise reduction algorithm explored in this paper is similar to the IS-127 NR contributed by the Freescale Research Lab, [1], [15]. This NR algorithm uses an advanced version of spectral subtraction. The NR can be deployed in VoIP systems as a part of the voice enhancement devices (VED). Other VED components include network echo canceller (NEC) and automatic level control (ALC) devices—all indispensable in a typical media gateway. We have implemented these components on a high-performance DSP based on the StarCore SC140 core, which is a six-issue variable-length execution set (VLES) architecture that can deliver up to 1600 millions of multiply-accumulate (MMAC) operations per second running at 400 MHz.

The voice quality and the number of channels supported are very important in the overall design of a media gateway. It is challenging to design an NR module that consumes few MCPS[1] while offering outstanding NR performance for a large class of transmitted signals. To contribute to the high channel density offered by the VoIP system, we modified the NR algorithm to take full advantage of the StarCore features and capabilities. The result is NR code that is efficiently implemented and highly maintainable. During implementation, there is a trade-off between precision requirements to meet system specifications and computational complexity. To illustrate this trade-off, we describe different ways to implement a low-pass filter (LPF). We evaluated the NR module to appraise its performance benchmarking points. We measured NR performance using natural speech signals as well as tone signals combined with additive stationary and non-stationary noise signals.

According to the ITU-T G.160 draft, [9], the noise reduction module must be situated after the network echo canceller (NEC) module and before the automatic level control (ALC) module (see **Figure 1**). A similar structure has been adopted for other voice processing systems (see [4]). The NR component reduces the effect of high acoustic background noise originating at S_{gen} and passing through the echo cancellation/control block. The NEC is viewed as a 4-port device. The input to the NR module is S_{out}.

1. Million of cycles per second (MCPS) is a benchmark of parallel-architecture CPU computational effort. Million of instructions per second (MIPS) is not used herein because the SC140 is a six-issue (parallel) processor.

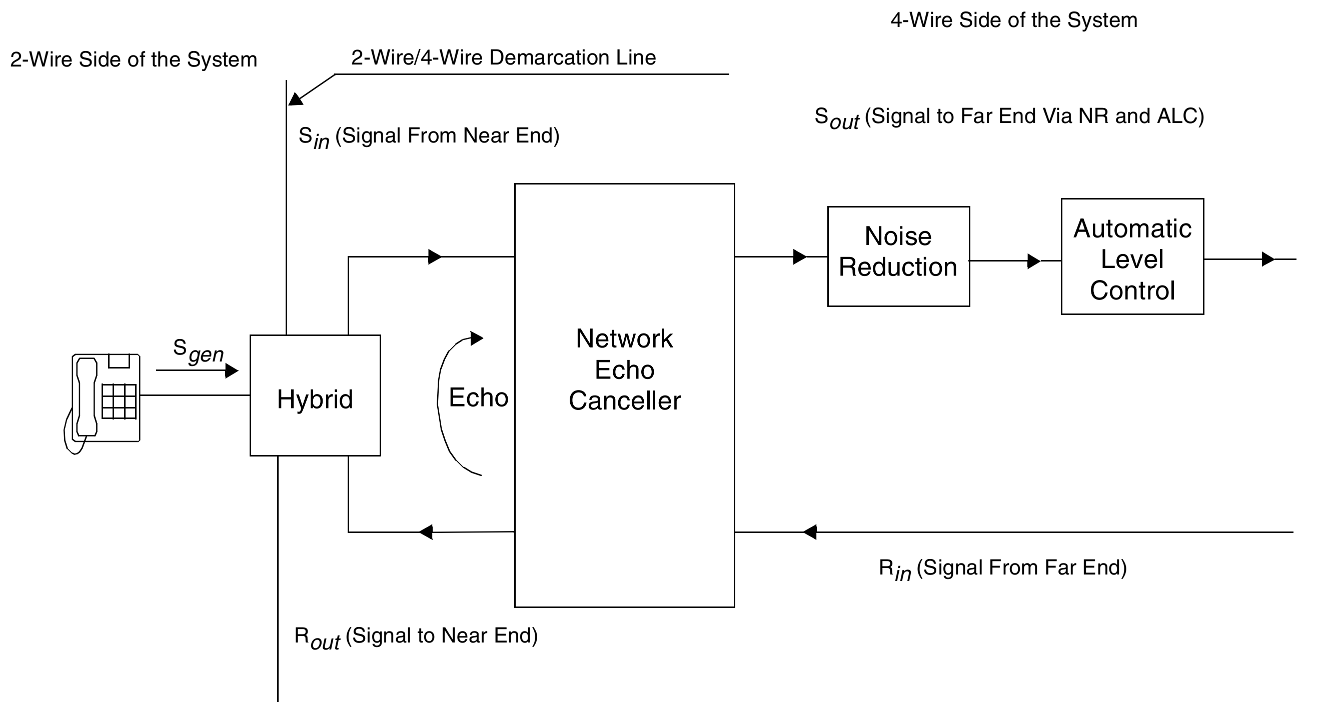

Figure 1. Noise Reduction Module In a Voice Quality Enhancement System

A classic approach to noise reduction is Boll's spectral subtraction method and algorithm. **Figure 2** shows a generalized block diagram of the algorithm architecture. If the VAD declares a frame as a speech frame, then $\xi\{in\_signal)$ is $X(e^{j\omega})$. If a frame is declared as a silence frame, the $\xi\{in\_signal)$ is $N(e^{j\omega})$. $\xi\{in\_signal)$ represents the discrete Fourier transform of *in_signal*.

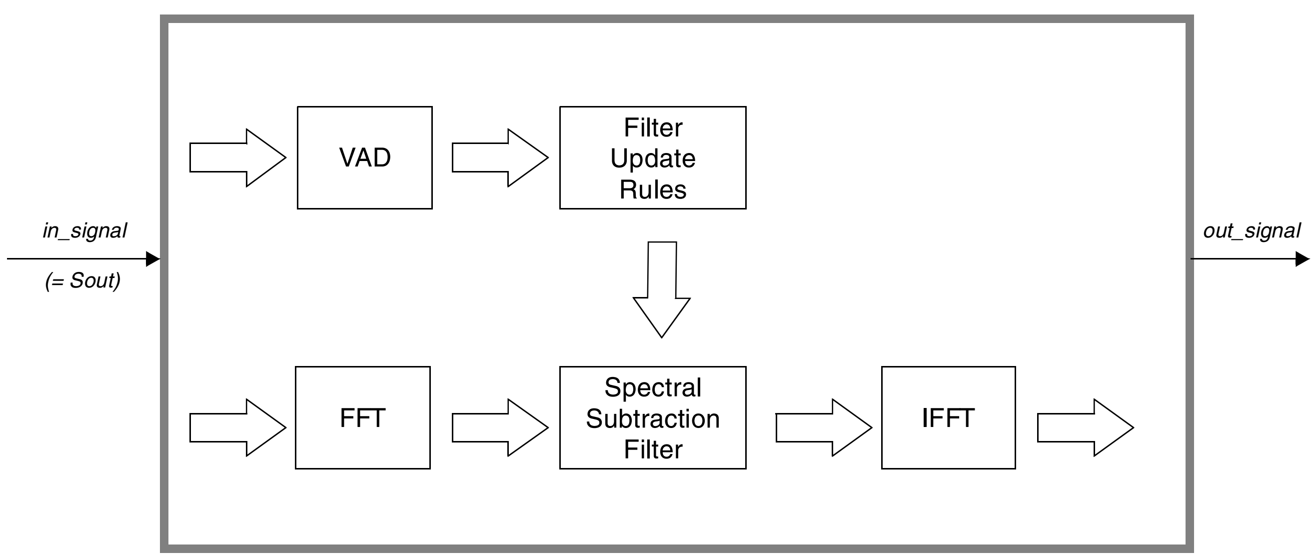

Figure 2. Noise Reduction Via Spectral Subtraction

Let's assume that noisy speech $x(k)= in\_signal(k)$ is produced by combining a noiseless speech $s(k)$ and an additive noise $n(k)$. Thus $x(k) = s(k) + n(k)$. If we limit an observation window to $[0, N - 1]$, where N denotes the length of the window (or, frame), we can express $x(k)$ as a discrete Fourier transform of the respective components, namely $X(e^{j\omega}) = S(e^{j\omega}) + N(e^{j\omega})$. We can formally rewrite the formula for $X(e^{j\omega})$ to resemble a linear filtration formula $S(e^{j\omega})$

$= H(e^{j\omega})X(e^{j\omega})$, where $H(e^{j\omega}) = 1 - N(e^{j\omega})/X(e^{j\omega})$. Thus, $H(e^{j\omega})$ can be called an ideal spectral subtraction filter. If its form is known, a restoration of $s(k)$ is possible. Since the exact form of $n(k)$, and thus $N(e^{j\omega})$, cannot reliably be determined, only an approximation (that is, $\tilde{H}(e^{j\omega})$) of the ideal spectral subtraction filter $H(e^{j\omega})$ is achievable. This approximation filter is discussed further in this application note.

The spectral subtraction filter $\tilde{H}(e^{j\omega})$ reduces the noise $n(k)$ and approximately preserves the original input signal $x(k)$. While the spectral subtraction filter $\tilde{H}(e^{j\omega})$ is calculated, an on-line measurement of noise spectrum, $\mu(e^{j\omega})$, is performed based on the silence frame, which contains only the noise component. The magnitude $|N(e^{j\omega})|$ of $N(e^{j\omega})$ is replaced by its estimate (or a running average), $\mu(e^{j\omega})$, calculated during non-speech activity, and the phase of $N(e^{j\omega})$, $\phi_N(e^{j\omega})$ is replaced by the phase of $X(e^{j\omega})$, $\phi_N(e^{j\omega})$. These substitutions result in the following expression for the spectral subtraction estimator $\tilde{S}(e^{j\omega})$:

Equation 1

$$\tilde{S}(e^{j\omega}) = [|X(e^{j\omega})| - \mu(e^{j\omega})]e^{j\phi x(e^{j\omega})}$$

or, in its equivalent form employing the spectral subtraction filter transfer function $\tilde{H}(e^{j\omega})$.

Equation 2

$$\tilde{S}(e^{j\omega}) = \tilde{H}(e^{j\omega})X(e^{j\omega})$$

where

Equation 3

$$\tilde{H}(e^{j\omega}) = 1 - \frac{\mu(e^{j\omega})}{X(e^{j\omega})}$$

and

Equation 4

$$\mu(e^{j\omega}) = E\{|N(e^{j\omega})|\}$$

where $E\{\cdot\}$ is a statistical averaging operator; note that in practical implementations the operator is replaced by a short-term averaging of choice.

The SC140 implementation of NR is a refined version of an advanced spectral subtraction method. The NR block diagram of NR is presented in **Figure 3**. The input signal is passed to a pre-emphasis filter before it is transformed to the frequency domain, where the nonuniform-frequency-width bands/channels are created. The first lowest band starts 125 Hz and ends at 250 Hz. The filter gain of this band is extended to the band 0–125 Hz. The incoming signal energy, noise energy, and signal-to-noise ratio are computed for each channel. A voice activity detector determines whether the current input frame represents voice or silence. The VAD declares a silence frame if the SNR (voice metric) is lower than a certain threshold. Another scenario that triggers silence frame declaration occurs when the total energy is larger than the floor noise and spectral energy deviation is larger than a threshold set for the tone signal absence case. The energy of the silence frame is used to update the background noise estimation. Based on modified SNR and background noise estimation for each band, the attenuation factors of all bands (ranging from 0 to 13 dB) are calculated. Then the filtering is performed in the frequency domain.

Subsequently, the filtered signal is transformed back to the time domain. The first 48-point signal portions are overlapped and combined with the previous processed signal. Finally, the signal is passed through the de-emphasis filter to obtain a noise-suppressed signal ([15], [1], [2], [3]).

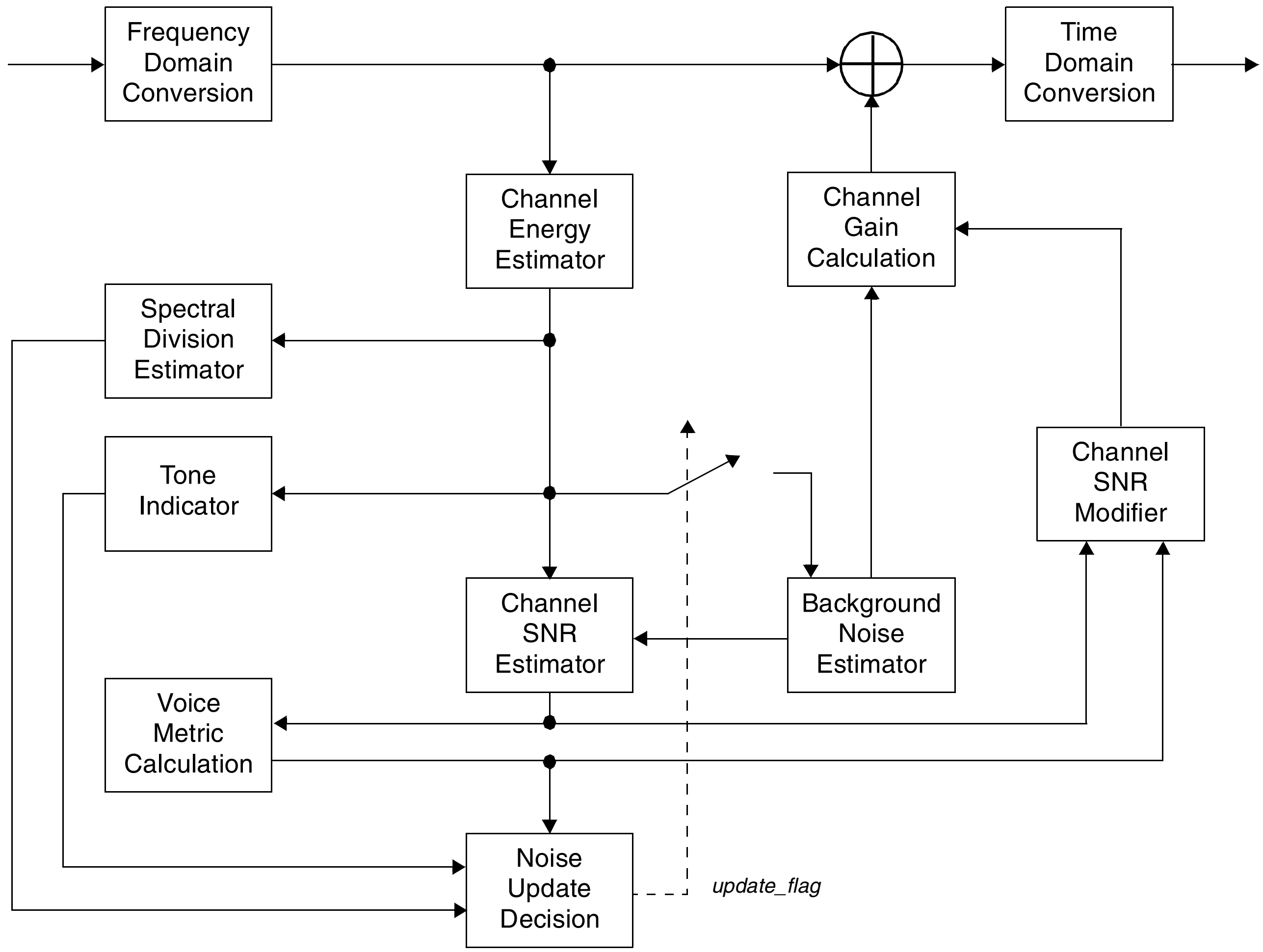

Figure 3. Noise Reduction Block Diagram

2 Noise Reduction on the SC140 Core

This section discusses only the performance-critical blocks of the SC140 implementation, including Fast Fourier Transform (FFT) and Inverse Fast Fourier Transform (IFFT), logarithmic and exponential functions, and the one-tap low-pass filter. The remaining blocks, including decision-making, control, windowing, and frequency domain filtering, raise fewer implementation issues in fixed-point arithmetic and are thus easier to optimize. Only the FFT and IFFT are implemented in the assembly-level code. The remaining sections are coded in the optimized C language.

2.1 Fast Fourier Transform

Based on NR profiling results, the FFT and IFFT consume 60 percent of NR processing. Thus, the assembly-level optimization effort was directed here. The FFT and IFFT are differentiated by their twiddle and scaling factors, they and share the same computation algorithm. Thus, this discussion of the FFT implementation also applies to

IFFT, unless otherwise mentioned. The 64-point complex FFT implementation performs the radix-2 decimation-in-time (DIT) computation. With two complex inputs, A and B, the butterfly computes two output points as shown here:

$$(A + BW_N^k) \text{ and } A - BW_N^k \qquad W_N^k \text{ is the twiddle factor}$$

The SC140 core supports bit-reverse addressing. The bit-reverse operation is embedded in the first stage of FFT computation. The hardware bit-reverse operation adds no cost other than set-up time to the butterfly computation. All the butterflies in the first stage share a common twiddle factor W_N^0. The imaginary value of the twiddle factor is zero. Omitting the redundant multiplication of imaginary values reduces the critical path of a butterfly computation by 50 percent in the SC140 implementation. To take advantage of the hardware bit-reverse mechanism, the input buffer must be aligned by 256 bytes.

In assembly-level optimization, two butterflies can be computed simultaneously in the kernel. The processing steps are rearranged so that the overall computation is reduced (see **Table 1**). Only operations involving the real portion of complex data are shown. The imaginary portion includes similar operations. The results for the forward FFT are scaled at every stage, but no scaling occurs in the inverse FFT. **subl** *y, x* is an application-specific SC140 instruction that computes $2x - y$ with saturation in one cycle. It speeds up the IFFT kernel by 25 percent.

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

FFT Computation	Instructions	IFFT Computation	Instructions
A + BW	**mac, mac**	A + BW	**mac, mac**
(A + BW)/2	**shr**	A – BW = 2A – (A – BW)	**subl**
(A – BW)/2 = A – (A + BW)/2	**sub**		

The FFT kernel has a triple-nested loop. The innermost loop computes the butterfly using the same twiddle factor; the middle loop goes through the butterfly with different twiddle factors; the outermost loop goes through the number of FFT stages, which depends on the number of computation points. The input to the FFT consists of 128 real data points. To increase efficiency, one half of the data is treated as imaginary and a 64-point complex FFT is performed. Post-FFT processing is needed to recover the 128-point real FFT results. The $\frac{N}{2}$-point complex FFT data is decomposed into N-point real FFT data. The decomposition formula is presented in **Equation 5**. F_k is the N-point real FFT data and H_k is the $\frac{N}{2}$-point complex FFT data.

Equation 5

$$F_k = \frac{1}{2}\left[H_k + H^*_{\frac{N}{2}-k}\right] - \frac{j}{2}\left[H_k - H^*_{\frac{N}{2}-k}\right]e^{-j2\pi k/N}$$

It is much more efficient to compute two data points in parallel. Another point of computation is $\frac{N}{2} - k$, which has the same twiddle factor value (with only sign change) as point k and shares the common terms (H_k and $H_{\frac{N}{2}-k}$) with point k.

Equation 6

$$F_{\frac{N}{2}-k} = \frac{1}{2}\left[H_{\frac{N}{2}-k} + H^*_k\right] - \frac{j}{2}\left[H_{\frac{N}{2}-k} - H_k^*\right]e^{-j\frac{2\pi}{N}\left(\frac{N}{2}-k\right)}$$

Thus, the decomposition kernel computes **Equation 5** and **Equation 6** simultaneously.

2.2 Logarithmic Function

Both the input and output are in the Q17.15 format. To ensure that the input to this function is within the range of 1 and 2, a normalization process is conducted.

Equation 7

$$\log_{10}(y) = \log_{10}(2^{sc}y2^{-sc}) = \log_{10}(2^{sc}y) + \log_{10}(2^{-sc})$$

where $2^{sc}y$ is in the range of [1,2). The first term computes *log10* of the normalized input and the second term converts the normalized output to the original output based on the scaling factor in the normalization process. The first term is computed through the polynomial approximation. The second term is computed using a linear function ($-0.301 \cdot sc = -9864[Q1.15] \cdot sc$). The *log10* computation is based on a fifth-order polynomial approximation, which yields 16-bit accuracy. By rearranging the polynomial, the formula with the fewest SC140 computations can be expressed in **Equation 8**. This function can be computed using only five MAC instructions.

Equation 8

$$\log_{10}(x) \approx \sum_{i=0}^{5} a_i(x-1)^i = y(y(y(y(a_5y + a_4)+a_3)+a_2)+a_1)$$

where $a_5 = 429$, $a_4 = -1854$, $a_3 = 4033$, $a_2 = -6962$, $a_1 = 14217$, $a_0 = 0$, $y = x - 1$, and x is in the range of [1,2). The polynomial coefficients, derived from the minimum mean square error (MMSE), are in Q1.15 format.

2.3 Exponential Function

The input and output are in the Q17.15 format. Rewriting the formula makes it a function of the power of two, as follows:

Equation 9

$$10^{\frac{z}{20}} = 2^{z\frac{\log_2 10}{20}} = 2^y = 2^{\lfloor y \rfloor + 1}2^{y - \lfloor y \rfloor - 1}$$

where $\lfloor \cdot \rfloor$ is the floor function, which corresponds to the largest integer smaller than or equal to its argument. The $\lfloor y \rfloor$ is the integer portion of the exponential and $y - \lfloor y \rfloor$ is the fractional portion of the exponential. The integer and fractional portions are found through shifting or masking operations. The derived polynomial approximation of 2^x requires x to be within the range of –1 and 0, which is achieved by subtracting 1 from the fractional portion. To ensure that the inequality holds, 1 is added to the integer portion. $\lfloor y \rfloor + 1$ is a scaling factor to adjust the normalized output to de-normalized output (simply by shifting the normalized output to the left by $\lfloor y \rfloor + 1$). A fourth polynomial approximation of 2^x s used to calculate the normalized input data, where x is [–1,0).

Equation 10

$$2^x \approx \sum_{i=0}^{4} a_i x^i = x(x(x(xa_4 + a_3) + a_2) + a_1) + a_0$$

where $a_4 = 224$, $a_3 = 1743$, $a_2 = 7844$, $a_1 = 22709$, $a_0 = 32767$. The polynomial coefficients, which are in Q1.15 format, are derived from MMSE and yield 16-bit accuracy.

2.4 Low-Pass Filter

The low-pass filter (LPF) is extensively used in parameter estimations, including channel energy, channel background noise energy, and long-term log spectral energy. The LPF filters out random variations of the estimated parameter. The approaches used in this project to implement LPF are characterized in terms of precision and complexity (that is, number of DSP cycles). The LPF can be expressed as follows:

Equation 11

$$E(n) = \alpha \cdot E(n-1) + (1-\alpha) \cdot e(n)$$

where $E(n)$ and $E(n-1)$ are the current and past estimated parameters, respectively; $e(n)$ is a computed parameter based on a current sample; α, assumed as a constant, controls the bandwidth of the LPF. There are five different implementation approaches, as shown in **Table 2**. For details on the SC140 instructions, consult [14]. Columns four, five, and six of the table list the number of bits for α, e, and E, respectively. In Method 2, α contains only five bits since the multiplication is implemented by shifting. The number of possible shifts is 32. In Method 3, although the E and e are represented by 32 bits, the computation uses 16 bits of E and e. The StarCore SC140 DSP core supports the 32 × 32 and 32 × 16 multiplication. The 32 × 16 multiplication can be achieved by **mpyus** and **dmacss** instructions in two cycles. The 32 × 32 multiplication is computed by the **mpyuu**, **dmacsu**, **macus**, and **dmacss** instructions in four cycles. Method 1 uses the least number of cycles yet the estimated parameter contains 16 bits. Either Method 2 or 4 should be used if the E and e are represented as 32 bit. Method 5 can be chosen if the system requires a 32-bit α (rare case). Method 3 should be avoided because it has a longer critical path and a greater number of instructions than Methods 2 and 4. In Method 5, two instructions (**mpyus**, **mpyus**) and one cycle can be saved at the expense of one-bit accuracy.

Table 2. LPF Implementation on the SC140 Core

Method	SC140 Instructions	Steps of $\alpha E + (1-\alpha)e$	α	e	E	Cycles
1	**mpy** **mac**	A = (1 – α)e αE + A	16	16	16	2
2	**sub** **shr** **add**	A = E – e A = αA A + e	5	32	32	3
3	**clb,clb** **shl,shl** **mpy,mpy** **shr,shr** **add**	norm(E), norm(e), A = (1 – α)e B = αE, A + B	16	32	32	5
4	**mpyus,mpyus** **dmacss,dmacss** **add**	A = (1 –α)e, B = αE A + B	16	32	32	3
5	**mpyus,mpyus** **dmacsu,dmacsu** **macus,macus** **dmacss,dmacss** **add**	A = (1 –α)e, B = αE A + B	32	32	32	5

3 NR Performance on the SC140 DSP Core

Table 3 shows the memory consumption of the NR module on the SC140 DSP core. The persistent data, which is proportional to the number of channels, is as low as 352 bytes.

Table 3. Resource Requirements for Noise Reduction

Function	Code	ROM	Stack	Persistent	Scratch
NR	3946	474	592	352	256

The code optimization methodology exploits features of the SC140 architecture [14], and the SC140 compiler [13]. The current implementation of the NR module on the SC140 DSP core consumes approximately 0.56 MCPS, in the worst-case scenario. It is feasible and even easy to deploy these NR components in tandem with other VEDs and the vocoder because the MCPS and memory resource usage are so low. Sharing the VAD between the NR and vocoder further reduces system resource requirements.

The initial experiment is designed to evaluate SNR reduction on the NR module, as measured using the silence frames. A speech file with an SNR ranging from 10 dB–40 dB is passed through the NR, and the energy of the noise signal in the silence frame is estimated. The energy difference between the input and output for silence-frame (noise-only) is directly attributed to the noise attenuation factor provided by the NR module. The results are shown in **Table 4**.

Table 4. Noise Attenuation at Different SNRs

SNR in dB	Noise Level of Input [dB – CE]	Noise Level of Output [dB – CE]	Attenuation in dB
10	–58	–66	8
20	–50	–62	12
30	–39	–52	13
40	–29	–42	13

At a high SNR, that is, when the signal integrity is excellent, the NR component can more accurately estimate the signal energy and thus perform more aggressive attenuation, which is as high as 13 dB. However, at low SNR values (10 dB or less), the noise attenuation factor is only 8 dB. Note that the attenuation factor has a positive correlation with SNR. **Figure 4** shows the frequency bands of the silent-frame speech signals, with and without NR processing. The frequency bands are non-uniformly distributed; narrower bands are at low frequencies and wider bands are at higher frequencies. The first band starts at 125 Hz and ends at 250 Hz. All the bands are attenuated by more than 10 dB.

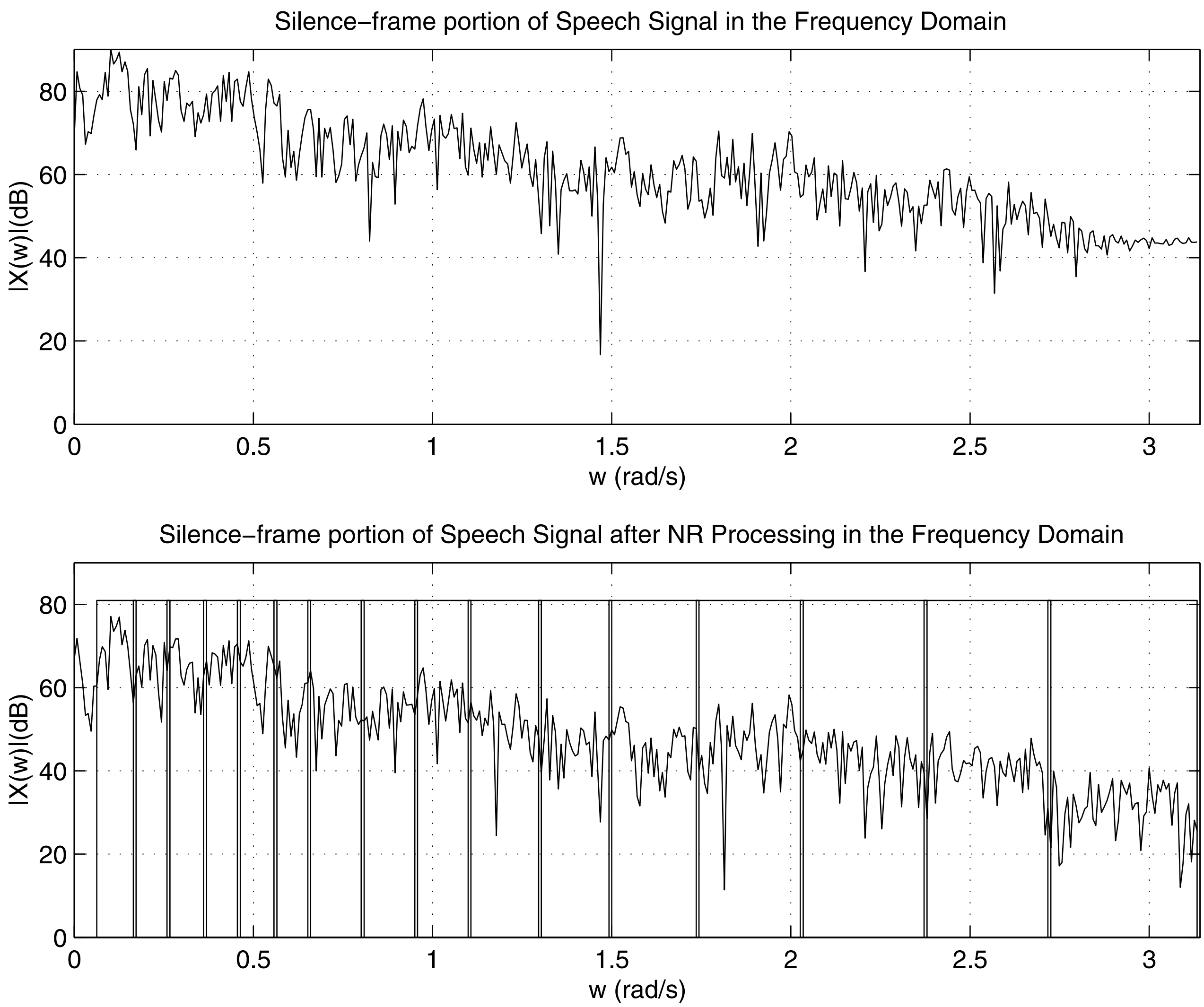

Figure 4. Frequency Domain of Silent-Frame Speech Signals Before and After NR Processing

Perceptual evaluation of speech quality (PESQ) is an objective method for end-to-end speech quality assessment of telephone networks and speech codecs [10]. One of the factors that affect PESQ score is the environmental noise of a speech signal. We collected measurements of NR input and output at different SNR levels, as listed in **Table 5**. The NR yields a better PESQ metric in the low SNR range but a worse metric in the high SNR range. The subjective tests show that NR improves the speech quality at all SNRs used in the experiment.

Table 5. PESQ Measurement Results

SNR (dB)	PESQ of NR Input Signal	PESQ of NR Output Signal
10	1.983	2.229
20	2.687	2.897
30	3.366	3.257
40	4.088	3.336

A set of clean dual-tone-multiple-frequency (DTMF) signals, also called touch tone signals, is passed to a DTMF detector. The DTMF detector has the same number of detections for input signals, with and without NR processing. The NR processing does not affect the tone integrity, and it preserves the original frequencies. This behavior is partly attributed to the architecture of the NR module, which includes a tone indicator block to detect the presence of tone signals. Consequently, tone signals passed to/through the media gateway equipped with the NR module are not adversely affected.

The NR evaluation experiment with DTMF tones is designed to emulate the action of an end user pressing the keypad of a telephone set with its handset in off-hook position, in a noisy acoustical environment. The high-level background noise affects the DTMF signal integrity. The ten-digit exemplary tones, which correspond to the dialing digits in North America, are polluted with car, pink, and white noise at 6–15 dB SNR.[2] These signals are passed to a typical DTMF detector, and the digit detection results are recorded. For comparison, the same noisy DTMF signals are enhanced by NR operations and then passed to the DTMF detector. The number of detections with and without NR processing are shown in **Table 6**.

Table 6. Detections of Car, Pink, and White Noise at Different Low-Level SNRs

SNR	Car Noise		Pink Noise		White Noise	
	No NR	NR	No NR	NR	No NR	NR
6	0	8	2	10	0	9
9	1	10	8	10	3	10
12	9	10	8	10	8	10
15	9	10	10	10	8	10

These experiments demonstrate that the NR enhances the DTMF detector when the tone signal is corrupted by a high level of noise. The corrupted tone signals before and after the NR are shown in **Figure 5**. We conducted an extensive performance characterization of the fixed-point version of the NR. The performance of the noise reduction algorithm, as evaluated subjectively through expert listening and objectively via SNR measurements, depends on the spectral and time characteristics of both signal components: noise and speech. An overall noise reduction of 10 to 12 dB is achieved for most natural speech signals polluted with noise, provided that the signal-to-noise ratio of the input signal is reasonable and the spectral characteristics of the polluting noise do not change very rapidly.

2. Bellcore specifies that the detector must work down to at least 23 dB SNR. These extremely low SNR tests were intended to fail the DTMF detector used in this experiment.

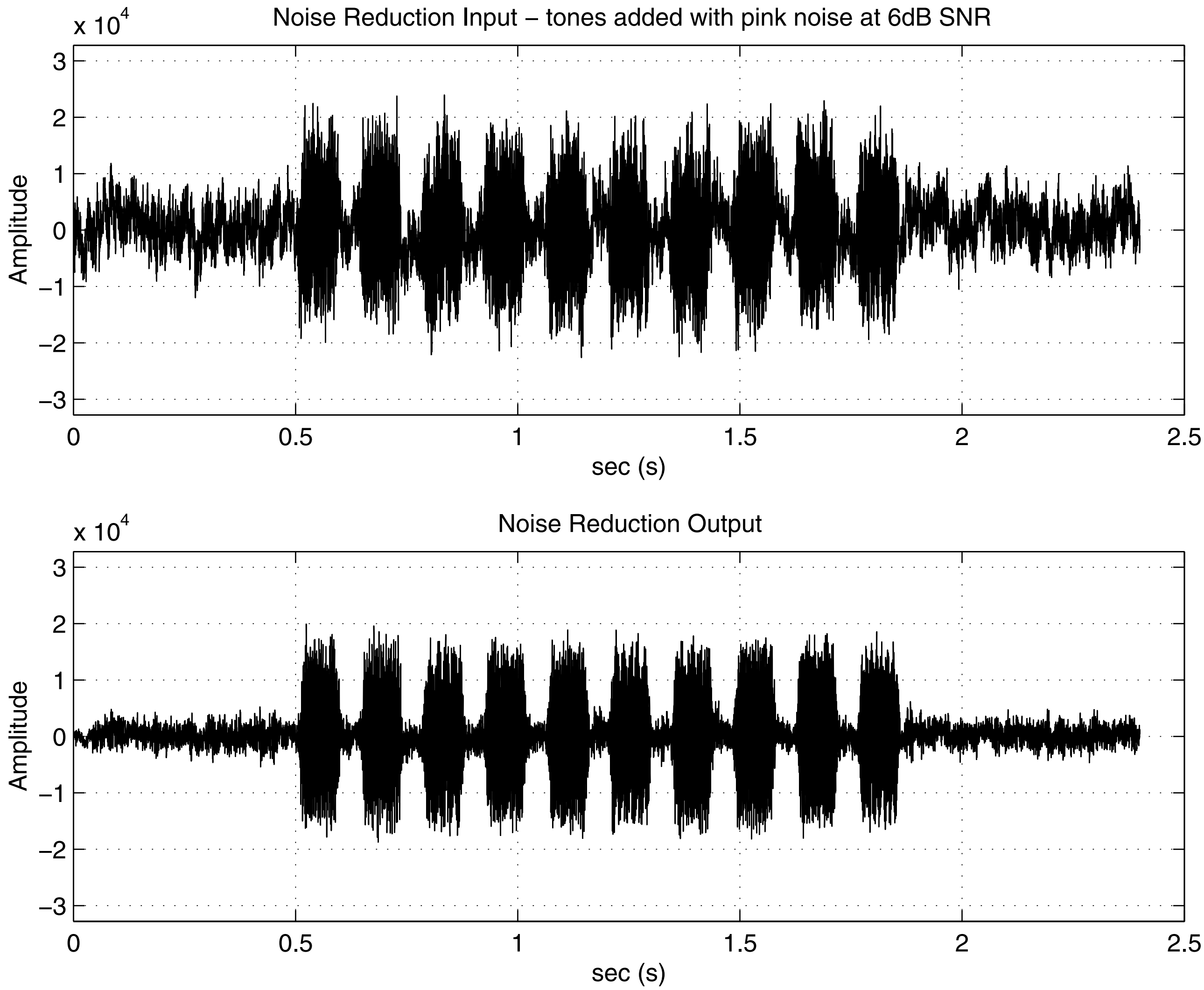

Figure 5. Tone Signals With/Without NR Processing

4 Conclusion

In the fixed-point NR implementation presented in this application note, the efficient computation and achieved precision of each block were analyzed. This analysis resulted in a version of NR that meets all system requirements and uses minimal resources. The NR is optimized yet conforms with requirements for code maintainability. Except for the FFT/IFFT sections, which have been hand-coded in assembly, all of the code can be adequately optimized at the C language level. Further algorithm enhancements can be made through high-level code changes. Based on profiling, 60 percent of the NR computation is spent on the FFT/IFFT. The SC140 DSP core can speed up FFT/IFFT computation using hardware support bit reversing and application-specific instructions. The current version of the NR consumes 0.56 MCPS. The NR operation does not adversely affect the signaling tones used in the networks. Under extremely noisy conditions, the NR improves the DTMF detection rate. The VAD information available in the NR can be efficiently shared with the vocoder, [4]. This resource sharing improves the efficiency of the media gateway in terms of cost and performance by leveraging the synergy of the individual components. As the results of the NR performance evaluation demonstrate, the NR module implemented on the SC140 DSP core can reduce background noise by 10–12 dB, thus enhancing speech quality in VoIP systems.

5 References

[1] J. Ashely. “Method and Apparatus for Suppressing Noise in a Communication System.” *US Patent 5,659,622 (assignee: Freescale Semiconductor, Inc.)*, 1997.

[2] J. Ashely. “Apparatus and Method for Rate Determination in a Communication System.” *US Patent 6,104,993 (assignee: Freescale Semiconductor, Inc.)*, 2000.

[3] J. Ashely. “Method and Apparatus for Suppressing Acoustic Background Noise in a Communication System by Equalization of a Pre- and Post-Comb-Filtered Subband Spectral Energies.” *US Patent 6,366,880 (assignee: Freescale Semiconductor, Inc.)*, 2002.

[4] F. Basburg, K. Swaminathan, and S. Nandkumar. “Noise Reduction and Echo Cancellation Front-End for Speech Codecs.” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 11(1):1–13, January 2003.

[5] S. Boll. “Suppression of Acoustic Noise in Speech Using Spectral Subtraction.” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 27(2):113–120, April 1979.

[6] G. M. Davis, Editor. “Signal and Feature Compensation Methods for Robust Speech Recognition.” *Noise Reduction in Speech Applications*. CRC Press, 2002.

[7] Y. Guelou, A. Benamar, and P. Scalart. “Analysis of Two Structures for Combined Acoustic Echo Cancellation and Noise Reduction.” *ICASSP-96, IEEE International Conference on Acoustics, Speech, and Signal Processing*, 1996.

[8] S. Gustafsson, R. Martin, P. Jax, and P. Vary. “A Psychoacoustic Approach to Combined Acoustic Echo Cancellation and Noise Reduction.” *IEEE Transactions on Speech and Audio Processing*, 10(5):245–256, July 2002.

[9] ITU-T Recommendation G. VED. *Draft G. VED/G.160, Voice Enhancement Devices*. International Telecommunication Union, Geneva, Draft No. 9, 2003.

[10] ITU-T Recommendation P.862. *Perceptual Evaluation of Speech Quality (PESQ): An Objective Method for End-to-End Speech Quality Assessment of Narrowband Telephone Networks and Speech Codecs*. International Telecommunication Union, Geneva, February 2003.

[11] M. Lang., H. Guo, J.E. Odegard, C.S. Burrus, and R. O. Wells Jr. “Noise Reduction Using an Undecimated Discrete Wavelet Transform.” *IEEE Signal Processing Letters*, 3(1):10–12, January 1996.

[12] P. Lockwood and J. Boundy. “Experiments With a Nonlinear Spectral Subtractor (NSS), Hidden Markov Models and the Projection for Robust Speech Recognition in Cars.” *Speech Communication*, 11:215–228, 1992.

[13] *SC100 C Compiler User’s Manual*, Freescale Semiconductor, Inc., 2001.

[14] *SC140 DSP Core Reference Manual*. Freescale Semiconductor, Inc., 2001.

[15] Technical Requirements (TR45). *Enhanced Variable Rate Codec, Speech Service Option 3 for Wideband Spread Spectrum Digital Systems*. TIA/EIA/IS-127-1, September 1, 1996.

NOTES:

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

AN2824
Rev. 1
9/2004