LOW-POWER AUDIO INPUT ENHANCEMENTS FOR PORTABLE DEVICES

A Dissertation
Presented to
The Academic Faculty

By

Heejong Yoo

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in Electrical Engineering

School of Electrical and Computer Engineering
Georgia Institute of Technology
January 2005

LOW-POWER AUDIO INPUT ENHANCEMENTS FOR PORTABLE DEVICES

Approved by:

Dr. David V. Anderson, Advisor
School of Electrical & Computer Engineering
Georgia Institute of Technology

Dr. W. Marshall Leach Jr.
School of Electrical & Computer Engineering
Georgia Institute of Technology

Dr. Douglas B. Williams
School of Electrical & Computer Engineering
Georgia Institute of Technology

Dr. Brani Vidakovic
School of Industrial and Systems Engineering
Georgia Institute of Technology

Dr. Paul E. Hasler
School of Electrical & Computer Engineering
Georgia Institute of Technology

Date Approved: January 2005

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my advisor, Dr. David Anderson, for his guidance, support, endurance, and encouragement during my staying at Georgia Tech. He has always been supportive of me not only when I made progress but also when I made mistakes. I am also deeply impressed with his dedication to research and students. I am happy to say that this research is only a small part of what I've learned from him. I also would like to thank to Dr. Paul Hasler for his extensive guidance on my continuous-time audio signal processing research. I am deeply impressed by the depth of his knowledge in both analog signal processing and neuro-morphic signal processing. This research would not have been possible without the leadership and the support of Dr. Anderson and Dr. Hasler.

I was fortunate to be a member of the CADSP research group. In particular, I've enjoyed working with David Graham and Rich Ellis and would like to thank them for their constructive feedback, outstanding achievements on VLSI circuit layout, measurement, and testing, and especially their willingness to share invaluable experimental results with me. I also had an opportunity to work closely with Daniel Allred, Venketash Khrishnan, and Walter Huang. I would like to thank them for rewarding research experience that we shared together.

I am grateful to my parents and other family members for their love and support. Most of all, I thank my lovely wife, Jaeeun, and beloved son, Taehwan, for their endless love, support, and perseverance during my time at Georgia Tech.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

With the development of VLSI and wireless communication technology, portable devices such as personal digital assistants (PDAs), pocket PCs, and mobile phones have gained a lot of popularity. Many such devices incorporate a speech recognition engine, enabling users to interact with the devices using voice-driven commands and text-to-speech (TTS) synthesis. It is well known that the speech recognition rate can be reduced when the signal-to-noise ratio (SNR) of the input is low unless the speech recognition engine is designed to be robust to high environmental noises.

The power consumption of DSP microprocessors has been consistently decreasing by half about every 18 months, following Gene's law. The capacity of signal processing, however, is still significantly constrained by the limited power budget of these portable devices. In addition, analog-to-digital (A/D) converters can also limit the signal processing of portable devices. Many systems require very high-resolution and high-performance A/D converters, which often consume a large fraction of the limited power budget of portable devices.

The proposed research develops a low-power audio signal enhancement system that combines programmable analog signal processing and traditional digital signal processing. By utilizing analog signal processing based on floating-gate transistor technology, the power consumption of the overall system as well as the complexity of the A/D converters can be reduced significantly. The system can be used as a front end of portable devices in which enhancement of audio signal quality plays a critical role in automatic speech recognition systems on portable devices. The proposed system performs background audio noise suppression (ANS) in a continuous-time domain using analog computing elements and acoustic echo cancellation (AEC) in a discrete-time domain using a field programmable gate array (FPGA).

CHAPTER 1
INTRODUCTION

With the development of VLSI and wireless communication technology, portable devices such as personal digital assistants (PDAs), pocket PCs, and mobile phones have gained a lot of popularity. Many such devices incorporate a speech recognition engine that enables users to interact with devices using voice-driven commands and text-to-speech (TTS) synthesis. A microphone, whether it is external or internal, often picks up a disturbing noise signal as well as a speech signal. The low signal-to-noise ratio (SNR) of the input adversely affects the performance of the speech recognition rate of the systems unless the speech recognition engine is designed to be robust to a high noise variance. In general, different characteristics of noise, such as acoustic echo, background noise, and babble noise, will be picked up in a microphone signal, so each type of noise requires a specific enhancement approach.

In this thesis, two common audio-enhancement techniques, acoustic noise suppression (ANS) and acoustic echo cancellation (AEC), are discussed. Figure 1 shows a system diagram of the research. We propose to implement ANS in a continuous-time domain for the benefits of the low-power parallel processing of analog computing elements and AEC in a discrete-time domain using a reconfigurable custom IC to gain high throughput of digital computing elements.

Research in ANS and AEC techniques has been popular for the last few decades and continues to be an active research area in the speech signal enhancement field. These two speech-enhancement techniques are normally solved separately and then are combined for better performance; some exceptions merge these two algorithms into one problem to find a global optimization solution [57, 76]. However, most previous attempts have been made in a purely discrete-time domain and the computational complexity of the existing algorithms tend to increase when seeking better performance. The increase in the computational complexity of digital domain speech-enhancement algorithms can be interpreted as

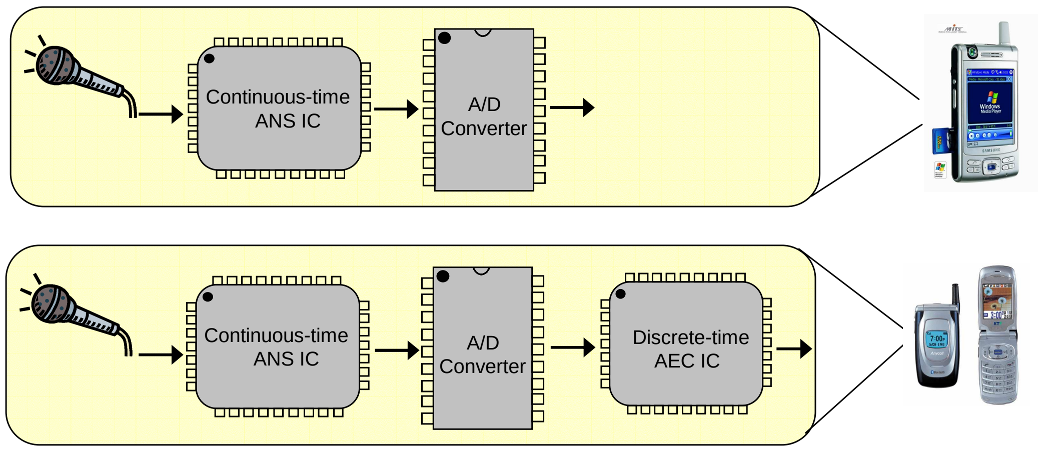


Figure 1. Proposed audio input enhancement system for portable devices. For portable PDA devices with speech recognition functionality, a continuous-time ANS IC is employed before the A/D converter. For portable communication devices, a continuous-time ANS and a discrete-time AEC IC are connected for low-power and real-time processing, respectively.

more millions of instructions per second (MIPS), causing a real-time processing problem in systems such as portable devices, in which higher MIPS are hard to attain because of the power constraint.

Power consumption in DSP microprocessors has been consistently decreasing by half about every 18 months, following Gene's law [37], as shown in Figure 2. The capacity of signal processing, however, is still significantly constrained by the limited power budget of these portable devices. In addition, analog-to-digital (A/D) converters can also limit the signal processing of portable devices. Many systems require very high-resolution and high-performance A/D converters, but these A/D converters often consume a large portion of the limited power budget of portable devices.

1.1 Cooperative Analog and Digital Signal Processing

Cooperative analog and digital signal processing (CADSP) can be defined as a method of intelligently combining programmable analog signal processing and digital signal processing techniques to achieve low-power and real-time signal processing. Figure 3 shows the difference between the traditional DSP approach and the CADSP approach. All signals in

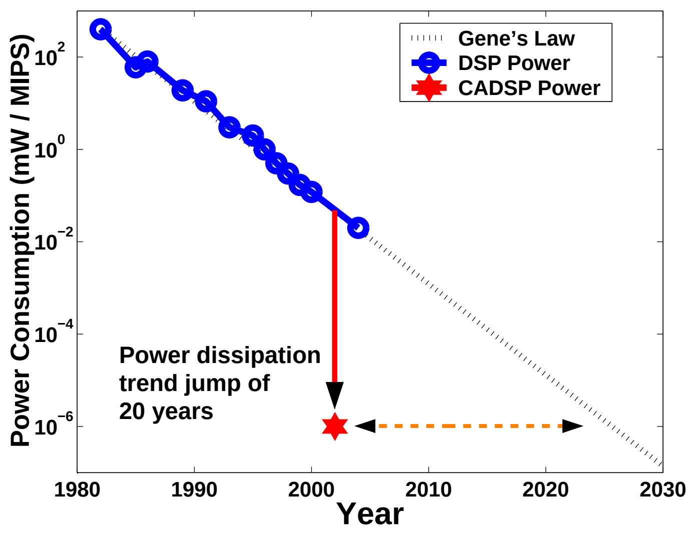


Figure 2. DSP power consumption per MIPS has been decreased for the past few decades, following Gene's Law [37]. The trends can be dramatically improved by shifting some of the processing from the digital domain to the analog domain. VLSI chips fabricated using cooperative analog and digital signal processing (CADSP) techniques show an equivalent of up to a 20-year increase in power per MIPS.

the real world are analog, while most modern signal processing and communications occur in a digital. Current trends immediately convert real-world analog signals into the digital domain using an A/D converter, so the majority of the system is implemented in the digital domain. Finally, the outputs of the digital system are reconverted to the analog signal using a digital-to-analog (D/A) converter. One drawback of this approach is that it consumes too much power, which is not acceptable, particulary for portable devices in which power consumption and battery life are critical. CADSP allows more freedom of movement for the partition between analog and digital computation. CADSP performs some of the processing in the analog domain prior to the A/D converter, reducing the computational load of the digital domain.

Many analog techniques are orders of magnitude more efficient than their digital counterparts in terms of speed and power dissipation [68]. The low-power consumption of analog computing is possible for floating-gate technology, which is explained in the following section. The output of the analog computing system gives more refined information such as Fourier coefficients [60] and Cepstrum [88] than a literal map of the incoming signal.

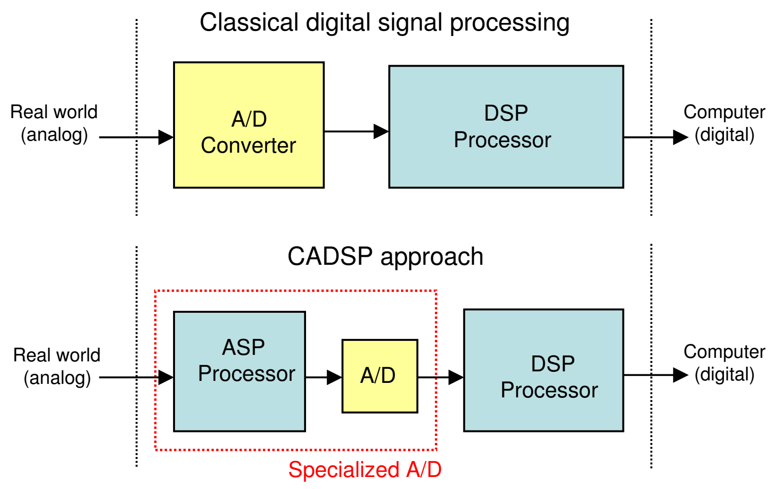


Figure 3. In traditional DSP systems, the A/D converter is placed as close to the real-world as possible. However, significant power savings can be achieved by moving some of the signal processing functionality into the analog domain (prior to the A/D converter).

Since this refined information requires much lower resolution when converted to the digital domain, much simpler and smaller A/D converters can be used instead of traditional A/D converters.

Although analog signal processing is capable of several important functions, the effects of resolution on these analog computing systems still remain unknown. The computational cost generally involves chip area, power consumption, design time, and development cost. While the computational cost of digital computation increases linearly as the bits of required resolution increase, the computational cost of analog computation increases exponentially. Sarpeskar [81] showed that analog computation has significant advantages when the resolution of the input signal is no more than 10 to 12 bits.

1.2 Analog Computing Elements: The Floating-Gate Approach

Floating-gate transistors such as EPROMs, EEPROMs, and flash memories have been used for some time now. Since the late 1980s, considerable research introducing new ways of using floating-gate devices [48, 51] has been published, providing researchers with a greater

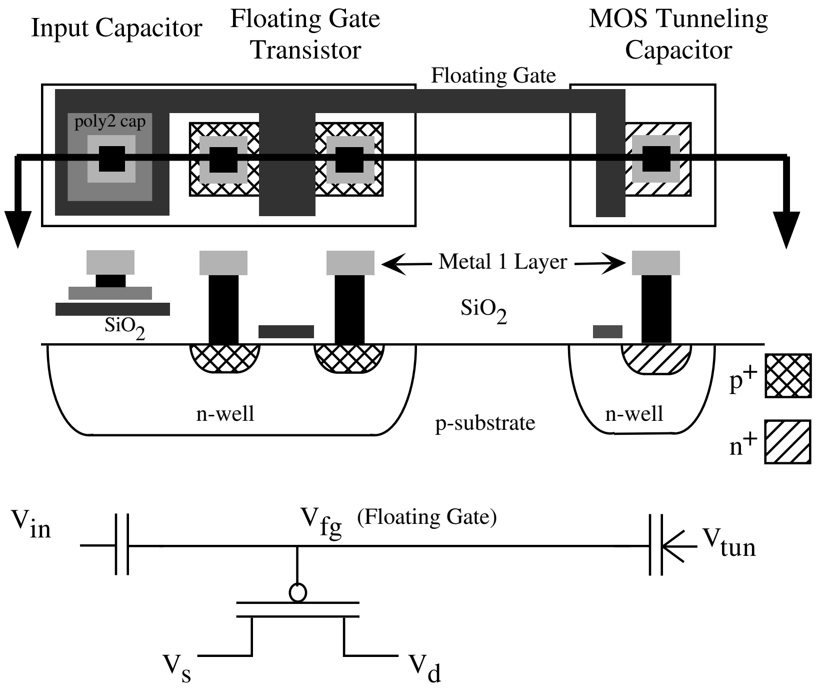


Figure 4. Layout, cross section, and circuit diagram of the floating-gate pFET in a standard double-poly, n-well MOSIS process: The cross section corresponds to the horizontal line slicing through the layout view. The pFET transistor is a standard pFET transistor in the n-well process. The gate input capacitively couples to the floating-gate by either a poly-poly capacitor, a diffused linear capacitor, or an MOS capacitor, as seen in the circuit diagram. Between V_{tun} and the floating-gate is our symbol for a tunneling junction—a capacitor with an added arrow designating the charge flow.

number of solutions. As a result, floating-gate devices are not just for memory anymore, but are circuit elements with analog memory and important time-domain dynamics [45, 69]. Floating-gate devices and circuits can be divided into three major categories: analog memory elements, capacitive-based circuits, and adaptive circuit elements.

As shown in Figure 4, a floating gate is a polysilicon layer that has no contact with other layers and thus has no DC path to a fixed potential. The polysilicon gate of an MOS transistor is completely wrapped in silicon dioxide, storing a charge almost permanently. Charge on the gate can be modified by three process–UV photo injection, electronic tunneling, and hot-electron injection [48]. The last two are the primary means of programming floating-gate circuits. For the tunneling process [51], a very large voltage is placed on the tunneling capacitor, as shown in Fig 4. As this large tunneling voltage increases, the effective width of the barrier decreases. This allows some electrons to breach the gap without adversely affecting the insulator. Through this process, electrons can be removed from the gate in a controlled manner. Hot-electron injection [48] has two requirements for changing the gate charge. First, an appreciable amount of current must flow through the device. Second, the source-to-drain voltage must be large. When both of these criteria are met, holes in a pFET that are flowing through the channel can build up enough energy for the impact ionization of an electron-hole pair. The electron can have enough energy to pass through the insulator and onto the floating gate. Therefore, hot-electron injection puts electrons onto the floating gate in a controlled manner.

1.3 Acoustic Noise Suppression (ANS)

ANS has been employed in many telecommunication systems to increase the intelligibility of the audio signal and/or reduce adverse artifacts. Speech communication and speech recognition systems can be degraded by various sources of background noise, which can vary from nonstationary noises, such as heaters, air conditioners, and computer fan noise, to stationary noises, such as transportation, roadway, and babble noise.

Various noise suppression algorithms have been proposed. Such algorithms include the short-time Fourier transform method (spectral modification) [38], the hidden Markov model (HMM) [33, 80], Wiener filtering [52], and parameter estimation methods using, for instance, maximum likelihood (ML) estimation, maximum *a posteriori* (MAP) estimation [94], or minimum mean-square error (MMSE) estimation [66]. Short-time Wiener filters are presented in [32, 66].

One of the popular short-time Fourier transform methods was originally developed by Boll [12], who coined the term “spectral subtraction.” The spectral subtraction method, however, creates a significant amount of high-frequency noise with a very short duration when used without any modification. This noise is referred to as “musical noise,” and many researchers have published various modified versions of spectral subtraction in conjunction

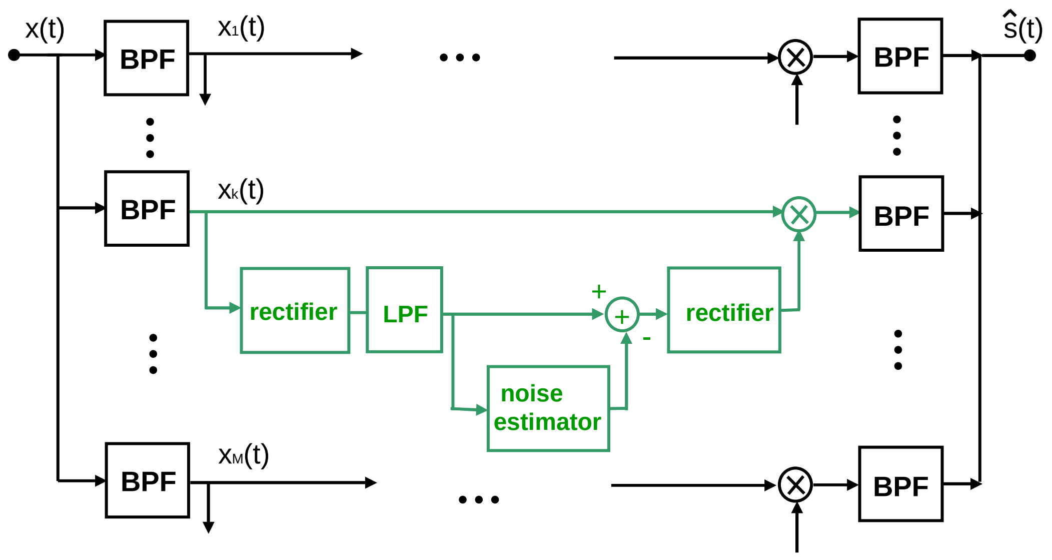


Figure 5. Simplified Schroeder's continuous-time noise suppression system. Schroeder's system was a purely analog implementation of a speech-enhancement system. A bank of band-pass filters separates the noisy signal into *M* different subbands whose bandwidth is about 300 Hz. Each subband signal is rectified and averaged to estimate a short-time noisy speech envelope.

with subband processing [34, 86] to alleviate the musical noise problem.

The first use of spectral gain modification in speech enhancement dates back to Schroeder's noise reduction systems [82, 83] in the 1960s. Figure 5 shows the block diagram of Schroeder's subband noise reduction algorithm based on subband gain modification. Schroeder's system was a purely analog implementation of a speech-enhancement system. A bank of band-pass filters separates the noisy signal into *M* different subbands whose bandwidths are about 300 Hz. Each subband signal is rectified and averaged to estimate a short-time noisy speech envelope. The noisy speech envelope is then subtracted from an estimate of the noise envelope. The subtracted result is rectified at another rectifier and multiplied by the subband signal. Finally, the subband signal is reconstructed to form a full-band estimate of the speech signal [42].

Diethorn [26] also proposed a subband noise suppression method based on an *a posteriori* SNR voice activity detector (VAD). Diethorn's system, as shown in Figure 6, implemented in the discrete-time domain, employed a single-pole recursive envelope estimator

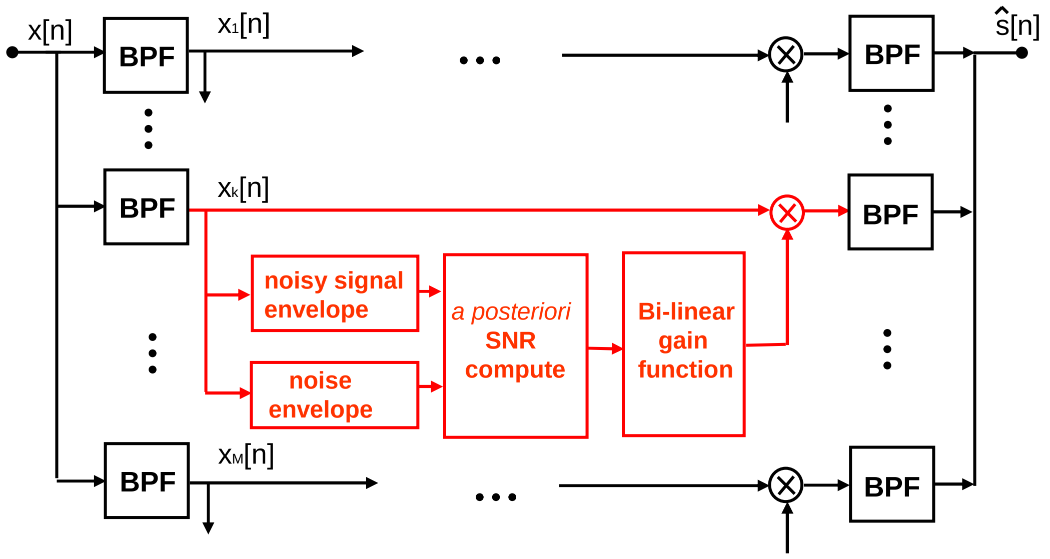


Figure 6. Diethorn's discrete-time noise suppression system. VAD and a single-pole recursive envelope estimator with different attack and decay time constants are used. A bi-linear gain function was used with noise suppression threshold, γ, specifying the certainty of speech. The bi-linear gain function, however, suffers severe loss of signal magnitude at low SNRs, especially, when large γ is used for more noise suppression.

with different attack and decay time constants for estimating both noise and the noisy envelope [42]. The bi-linear gain function was used with noise suppression threshold, γ, specifying the certainty of speech. The bi-linear gain function, however, suffers severe loss of signal magnitude at low SNRs, especially when large γ is used for more noise suppression.

1.4 Acoustic Echo Cancellation (AEC)

Sondhi [38, 90, 91] developed an echo canceler in the 1960s to address the network (electrical) echo problem, which occurs as a result of the impedance mismatching of a four-wire long distance connection and a two-wire local loop of the communication circuits. Network echo cancellation is a relatively easy problem compared to AEC because the network echo is very short and stationary, while the acoustic echo is long and time varying. For a typical office room, the length of the room impulse response can last up to 250 msec. At an 8 KHz sampling frequency, the number of filter taps of an adaptive filter should be larger

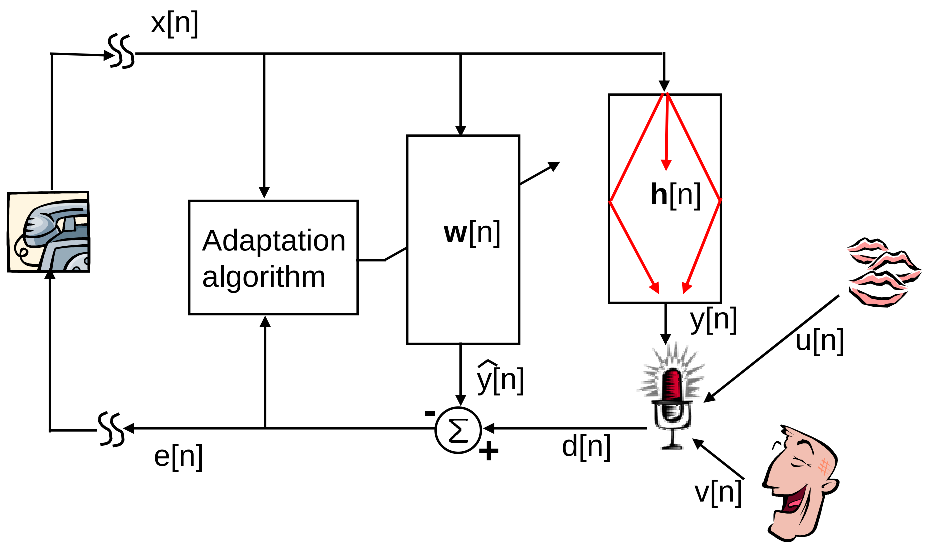


Figure 7. AEC system for a teleconferencing application. The far-end speech signal, $x[n]$**, is played through a loudspeaker in a receiving room and its echo,** $y[n]$**, is picked up by a microphone for a single channel system. The microphone also picks up near-end speech signal,** $v[n]$**, as well as background noise,** $u[n]$**. An adaptive filter estimates the room impulse response,** $\mathbf{w}[n]$**, and then the echo replica,** $\hat{y}[n]$**, is subtracted from the microphone signal.**

than 2000 for higher echo return loss enhancement (ERLE).

Figure 7 shows an AEC system for a teleconferencing application. The far-end speech signal, $x[n]$, is played through a loudspeaker in a receiving room, and its echo, $y[n]$, is picked up by a single microphone for a single channel system. The microphone also picks up a near-end speech signal, $v[n]$, as well as background noise, $u[n]$. The room impulse response, $\mathbf{h}[n]$, may rapidly change during the connection as the position and the direction of either loudspeaker or microphone vary. An adaptive filter in an AEC system continuously estimates the room impulse response, $\mathbf{w}[n]$, and the echo replica, $\hat{y}[n]$, is subtracted from the microphone signal before the error signal, $e[n]$, is transmitted back to the far-end side over a communication channel.

1.4.1 Adaptive Filters

Adaptive filters have been used in many areas such as system identification, equalization, signal detection, noise cancellation, and echo cancellation [38]. In particular, finite impulse response (FIR) adaptive filters have gained considerable popularity over infinite impulse response (IIR) for several reasons. First, FIR adaptive filters are more stable than IIR adaptive filters since filter coefficients are more robust to the quantization noises. Second, FIR adaptive filters have a simple form in terms of weight update. Finally, the performance of these algorithms is well understood in terms of their convergence and stability.

The goal of an FIR adaptive filter in the mean-square error sense is to find the vector, $\mathbf{w}[n]$, at time n that minimizes the quadratic function

$$\xi[n] = E\{|e[n]|^2\}, \tag{1}$$

where $e[n] = d[n] - y[n]$, $d[n]$ is a desired signal, and $y[n]$ is a filtered output. Equation (1) can be solved in many ways, but the steepest descent algorithm is the most common iterative procedure. The steepest descent algorithm is a method that finds the extrema of non-linear functions. It estimates the solution vectors at every time index by adding a correction term to the estimate vector of a previous time index such that the current estimate is closer to the optimal solution than the previous estimate.

In the realization of an FIR adaptive filter, the most extensively used adaptive filter is the least mean-square (LMS) algorithm, summarized in Table 1. The LMS algorithm, which was first introduced by Widrow and Huff [97], is simply an approximated realization of the steepest descent algorithm. The LMS algorithm uses a one-time sample mean as the approximation of the cross-correlation between the error signal and the input vectors, $E\{e[n]\mathbf{x}[n]\}$. The adaptation characteristic of the LMS algorithm, however, is good enough in most applications. The computational complexity of the LMS algorithm is $O(K)$, where K is the number of filter taps.

One drawback of the LMS algorithm is its slow convergence rate for colored input signals. The LMS algorithm converges to the vicinity of its optimal value in the mean-square sense if and only if $0 < \mu < 2/\lambda_{\max}$, where $\lambda_{\max}$ is the maximum eigenvalue of the correlation matrix of input samples. For colored input signals (a speech signal is the typical example of a colored signal), $\lambda_{\max}$ is much greater than 1 and consequently the upper limit of the step-size, μ, is severely restricted, limiting the convergence speed of the

Table 1. LMS algorithm has three equations: the filtering operation, the error calculation, and the weight update operation. The computational complexity of the LMS algorithm is $O(K)$, where K is the number of filter taps.

The LMS algorithm	
Filtering operation:	$y[n] = \sum_{k=0}^{K-1} w_k x[n-k]$
Error calculation:	$e[n] = d[n] - y[n]$
Weight adaptation:	$w_k[n+1] = w_k[n] + \mu e[n]x[n-k]$ $(0 \leq k \leq K-1)$

LMS algorithm.

In the standard LMS algorithm, the weight correction term is directly proportional to the magnitude of the input samples. Therefore, when the magnitude of $x[n-k]$ is large, the LMS algorithm suffers a “gradient noise amplification” [52, 53] problem. To overcome this problem, the normalized LMS (NLMS), shown in Table 2, was introduced. The NLMS algorithm is a special implementation of the LMS algorithm, which has a more stable and fast converging properties. The NLMS algorithm takes into account any variation in the signal level of the input signals in the selection of the step size. The convergence of the NLMS algorithm in the first and second moment is guaranteed for a stationary process when the normalized step size, β, satisfies $0 < \beta < 2$. Many variants of the LMS, including SS-LMS, SD-LMS, SE-LMS [52, 53], LMS-Newton [35], and PNLMS [42], are also available in the literature.

Narayan et al. [73] proposed a transform-domain LMS algorithm to enhance the convergence speed of the LMS algorithm by using an orthogonal transform to decorrelate the input signal. A transform-domain adaptive filter converts a time-domain input signal into a transform-domain input signal using the orthogonal data-independent transform followed by a power normalization. The Karhunen Loéve transform (KLT) is the optimal transform [9] in the sense that it performs an ideal decorrelation of the input data by projecting them onto the eigenvectors of their autocorrelation matrix [35]. However, in

Table 2. NLMS algorithm is a special implementation of the LMS algorithm, which has more stable and fast converging properties. It takes into account the variation in the signal level of the input signals in selection of the normalized step size, β. The convergence of the NLMS algorithm is guaranteed for a stationary process when $0 < \beta < 2$.

The NLMS algorithm	
Filtering operation:	$y[n] = \sum_{k=0}^{K-1} w_k x[n-k]$
Error calculation:	$e[n] = d[n] - y[n]$
Weight adaptation:	$w_k[n+1] = w_k[n] + \beta \frac{x[n-k]}{\epsilon + \hat{P}_x[n]} e[n]$ $(0 \leq k \leq K-1)$
Power estimation:	$\hat{P}_x[n+1] = \beta \hat{P}_x[n] + (1-\beta) x[n]^2$

practice, KLT is not the best choice since the KLT is signal dependent, and the input statistics are usually unavailable in advance. Many suboptimal orthogonal transforms such as the discrete Fourier transform (DFT), discrete cosine transform (DCT), discrete Hartley transform (DHT), and Walsh-Hadamard transform (WHT) have been proposed in the literature [9, 35, 62, 73]. The $O(K)$ complexity transform-domain LMS using a sliding FFT algorithm is also presented in [36].

The recursive least-square (RLS) algorithm [35, 53] is also a popular adaptive filter that recursively minimizes the error in the least-square sense. The convergence speed of the RLS is generally known to be faster than that of the LMS-based algorithms, but its high computational complexity, $O(K^2)$, is generally too expensive for a real-world system. Numerous fast versions of the RLS algorithm [35] were developed, but they still have instability problems resulting from multiple simplifications.

Ozeki and Umeda [75] developed the affine projection algorithm (APA), a generalization of the NLMS algorithm, using affine subspace projections [42]. The APA updates its weights based on the previous P input vectors, where P is referred to as the projection order, while the NLMS algorithm updates its weights based on the current input vector. The complexity of the APA is $2KP + O(P^2)$, and the fast version APA, which is known as fast affine projection (FAP), was first introduced by Gay [40, 41]. The FAP is known to have

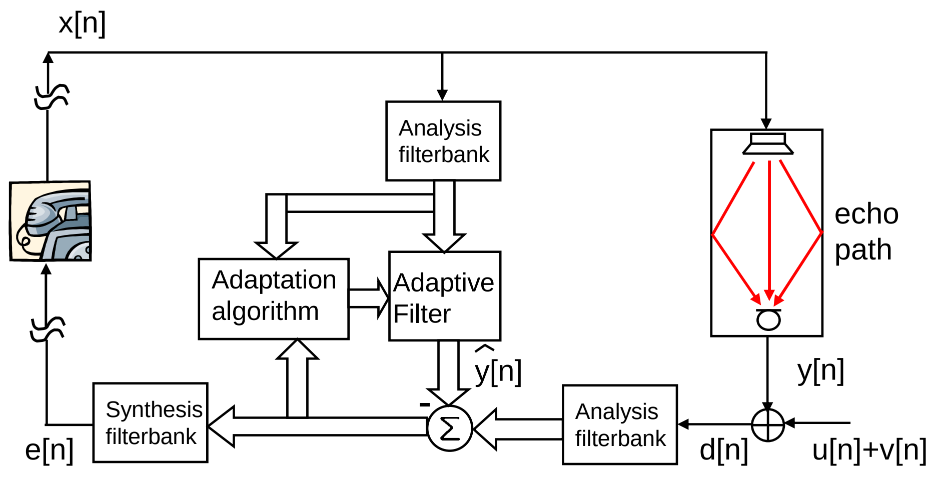


Figure 8. Block diagram of subband AEC (SAEC) for a teleconferencing application. The SAEC can result in faster convergence than the LMS-based algorithms since each subband signal has a smaller eigenvalue spread than the original wideband signal due to band-partitioning property of the analysis filter bank.

lower computational complexity than the APA and a faster convergence speed than that of the NLMS algorithm. Different versions of FAPs [27, 28, 63, 93] are also available in the literature, depending on the way the inverse of the autocorrelation matrix is estimated.

When the required number of adaptive filter taps, K, is extremely large, as in the AEC system, even a low complexity algorithm such as LMS can cause a problem for the real-time processing of hands-free devices or a teleconferencing system. For a multi-channel AEC, real-time processing becomes even more challenging. Among the many different adaptive filters, subband AEC (SAEC) is very attractive for real-time processing applications. Figure 8 shows the synthesis-independent structure of the subband adaptive filter. In SAEC, a number of subband adaptive filters can independently estimate subband echo using subband far-end and microphone signals. By analyzing input statistics, different adaptive filter algorithms can be adopted for different subbands to achieve a faster convergence rate. Subband echoes are merged into a wideband echo using a synthesis filter bank, and the wideband echo is transmitted to a far-end side. For applications in which a delay in the signal path is critical, other subband AEC structures, referred to as synthesis-dependent structures, can also be used [35, 72].

Subband processing ensures faster convergence for the LMS-based algorithms since it reduces the eigenvalue spread of the autocorrelation function of the input process by band-partitioning a wideband signal into a subband signal. Subband processing can result in low-power computing by optimizing parallelism and employing dedicated hardware for repeated data transform. Furthermore, a significant amount of computational savings can be achieved because of the reduced sampling rate, which is related to the number of subbands and the properties of the analysis and synthesis filter banks. DFT filter banks are commonly used for the efficient realization of the analysis and synthesis filter banks. Polyphase structure [42] and weighted-overlap-add (WOA) structures [22, 23, 35] have been used as DFT filter banks for critical-sampling and over-sampling systems, respectively.

1.4.2 Double-Talk Detector

In real applications, the performance of the adaptive filter suffers significant degradation in double-talk situations. Double talk occurs when two speakers on both near-end and far-end sides speak simultaneously. In Figure 7, the signal level of the near-end speech signal, $v[n]$, becomes dominant in the microphone signal, $d[n]$, when a double-talk situation occurs. This causes the adaptive filter to diverge since the correlation between echo and the microphone signal drops significantly. One common solution for this problem is to use a double-talk detector (DTD). A DTD, in general, monitors the relevance of the various signals of Figure 7. Once double talk is detected, the update of adaptive filters can be either completely stopped or at least slowed down, depending on the correlation between echo and the microphone signal.

The general procedure of most DTDs is described by the following [42]:

1. A detection statistics, ξ, is calculated using available signals, e.g., $x[n], d[n], e[n]$, and the estimated filter coefficients, $\mathbf{w}[n]$.

2. The detection statistic, ξ, is compared to a threshold, T, and double talk is declared if $\xi < T$.

3. Once double talk is declared, the filter update is disabled for a minimum period of time, T_{hold}.

4. If $\xi > T$ consecutively over T_{hold}, the filter resumes adaptation and the comparison of ξ to T continues until $\xi < T$.

A number of DTDs have been developed, and an objective method of evaluating various DTDs was proposed in [19]. The Giegel algorithm [29] was successful for a line echo; however, it has not always been successful for an acoustic echo. The two-path model DTD [74] is based on the combination of an adaptive background filter and a fixed foreground filter. The foreground filter constantly models the acoustic echo. Whenever the background filter performs better than the foreground filter, its filter coefficients are copied to the foreground. Other methods based on cross-correlations [10, 98, 77] and coherence [39] appear to be more popular for AEC applications. A few examples of the most widely used cross-correlations are cross-correlation coefficients between $\mathbf{x}[n]$ and $d[n]$, $\rho_{xd}[n]$, and cross-correlation coefficients between $\mathbf{x}[n]$ and $e[n]$, $\rho_{xe}[n]$. The cross-correlations $\rho_{xd}[n]$ and $\rho_{xe}[n]$ are defined as follows:

$$\rho_{xd}[n] = \frac{P_{xd}[n]}{\sqrt{P_x[n]P_d[n]}}, \tag{2}$$

$$\rho_{xe}[n] = \frac{P_{xe}[n]}{\sqrt{P_x[n]P_e[n]}}, \tag{3}$$

where $P_x[n]$ is the power of the far-end input signal, $P_e[n]$ is the power of the error signal, $P_d[n]$ is the power of the near-end microphone signal, $P_{xe}[n]$ is the cross-power between the far-end input and the error signal, and $P_{xd}[n]$ is the cross-power between the far-end input and the near-end microphone signal. The detection statistic, ξ, is then calculated from these cross-coefficients and compared to a threshold. The fundamental problem of these methods is that these cross-correlations are not well normalized; thus, setting a universal threshold for reliable performance in various situations is difficult. To address this problem, "normalized cross-correlation," a special case of the "generalized cross-correlation" technique, is presented in [42].

CHAPTER 2

DISTRIBUTED ARITHMETIC FIR FILTER

A discrete-time linear FIR filter generates the output, $y[n]$, as a sum of delayed and scaled input samples, $x[n]$,

$$y[n] = \sum_{k=0}^{K-1} w_k x[n-k], \tag{4}$$

where w_k could be fixed coefficients or time-varying coefficients. Multiply-and-accumulate (MAC) is widely used for such filtering, but it is well known that MAC is expensive to implement with hardware because of its complex logic and large area size. Alternatively, MAC operations may be replaced by a series of look-up-table (LUT) accesses and summations. Implementation of the filter, known as distributed arithmetic (DA) [24, 78], achieves higher throughput and lower logic complexity at the cost of increased memory usage. DA provides a multiplier-less implementation of FIR filters through a bit-serial computation, utilizing all possible combination sums of the filter coefficients [96].

2.1 LUT-Based DA FIR Filter

When input samples are represented as two's-complement binary numbers scaled such that $|x[n-k]| < 1$, then $x[n-k]$ can be written as

$$x[n-k] = -b_{k,0} + \sum_{l=1}^{B-1} b_{k,l} 2^{-l}, \tag{5}$$

where $b_{k,n} \in \{0, 1\}$, $b_{k,0}$ is a sign bit, and $b_{k,B-1}$ is a least significant bit (LSB). Now, $y[n]$ of Equation (4) can be written as

$$y[n] = -\sum_{k=0}^{K-1} b_{k,0} w_k + \sum_{k=0}^{K-1} \left[\sum_{l=1}^{B-1} b_{k,l} w_k \right] 2^{-l}. \tag{6}$$

After interchanging the order of summation, we have

$$\begin{aligned} y[n] &= \underbrace{-\sum_{k=0}^{K-1} b_{k,0} w_k}_{C_0} + \sum_{l=1}^{B-1} \underbrace{\left[\sum_{k=0}^{K-1} b_{k,l} w_k\right]}_{C_l} 2^{-l} \\ y[n] &= \sum_{l=0}^{B-1} C_l 2^{-l}. \end{aligned} \tag{7}$$

It is noted that the C_l only takes one of 2^K possible combination sums of the filter coefficients since $b_{k,n} \in \{0, 1\}$. These values can be pre-computed and stored in an LUT if the filter coefficients are fixed. The filtering operation, then, can be done by the B look-up, shift, and accumulate operations.

One example of a typical DA implementation for a 4-tap ($K = 4$) FIR filter is shown in Figure 9. Most DA architecture can be categorized into three small units: the shift register unit, the DA base unit, and the adder/shifter unit. For the traditional LUT-based DA architecture of fixed filters, the DA base unit consists of only one 2^K size LUT. As one can see in Chapters 5 and 7, the DA base unit can be more complicated than just a single LUT, but both the shift register unit and the adder/shifter unit are common for different DA architectures. A ROM is often used as a realization of the LUT for fixed filters, and a RAM is used for adaptive filters [4]. The shift register unit stores K most recent input samples and its contents are shift righted at every system clock cycle. The concatenation of the output of the shift register unit (rightmost bits of the shift register unit) becomes the address of the LUT inside the DA base unit. LSBs of input samples are used first as LUT addresses, and every output of LUT is shifted and accumulated B consecutive times, where B is the precision of the input data, adding up with one bit shift-righted value of the previous accumulator. For most significant bits (MSBs) of input samples, sign control, S1, is set to 1, and thus, the output of LUT is subtracted from the shift-righted accumulator. Since DA can complete the FIR filtering operation in B clock cycles regardless of K DA has been widely used for high-speed filter implementation, especially when $K >> B$.

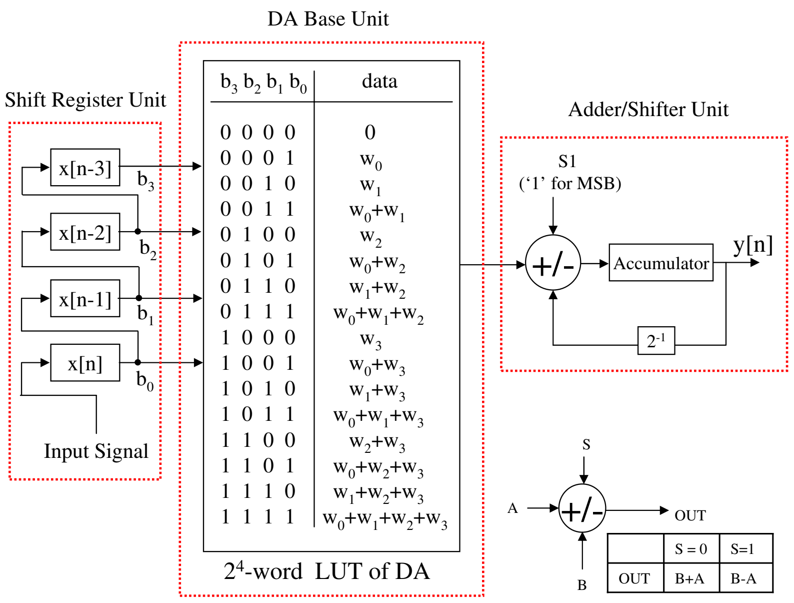


Figure 9. Block diagram of a 4-tap ($K = 4$) DA FIR filter. The DA block consists of three small units: the shift register unit, the DA base unit, and the adder/shifter unit. Each coefficient has B bits of precision. As one can see in Chapters 5 and 7, the DA base unit can be more complicated than just a single LUT, but both the shift register unit and the adder/shifter unit are common for different DA architectures. A ROM is oftentimes used as a realization of the LUT for fixed filters, and a RAM is used for adaptive filters.

2.2 Adder-Based DA FIR Filter

As an alternative implementation of a memory-intensive LUT-based DA, an adder-based DA was first proposed in [18] for a one-dimensional IDCT processor and further expanded to a two-dimensional IDCT processor [14]. The adder-based DA architecture was also attempted in [15] for LUT-based FPGAs with "vertical subexpression sharing" to minimize area.

The conventional LUT-based DA architecture decomposes the input signal into bit-serial forms (Equation (5)) and stores all possible combination sums of the filter coefficients into the LUT. The adder-based DA architecture is proposed in [14, 18, 15] in contrast to decompose filter coefficients into bit-serial forms. Filter coefficients, w_k, can be written as

$$w_k = -a_{k,0} + \sum_{l=1}^{B_c-1} a_{k,l} 2^{-l}, \tag{8}$$

where $a_{k,n} \in \{0, 1\}$, $a_{k,0}$ is a sign bit, B_c is the word length of the filter coefficients, and

a_{k,B_c-1} is an LSB. Now, $y[n]$ of Equation (4) can be written as

$$y[n] = -\sum_{k=0}^{K-1} a_{k,0}x[n-k] + \sum_{k=0}^{K-1}\left[\sum_{l=1}^{B_c-1} a_{k,l}x[n-k]\right]2^{-l}. \tag{9}$$

After interchanging the order of summation, we have

$$\begin{aligned} y[n] &= \underbrace{-\sum_{k=0}^{K-1} a_{k,0}x[n-k]}_{A_0} + \sum_{l=1}^{B_c-1} \underbrace{\left[\sum_{k=0}^{K-1} a_{k,l}x[n-k]\right]}_{A_l} 2^{-l} \\ y[n] &= \sum_{l=0}^{B_c-1} A_l 2^{-l}. \end{aligned} \tag{10}$$

The main difference between LUT-based and adder-based DA architecture is that the adder-based DA uses an adder network instead of a pre-computed LUT to implement A_l. Unlike C_l, in which $b_{k,l}$ can't be previously known, A_l can be constructed using an adder network since bit representations of filter coefficients, $a_{k,l}$ are fixed values for all k and l. Adder-based DA, as shown in [14, 18, 15], can be more area efficient than LUT-based DA for the following reasons. First, adder-based DA architecture constructs adders only for non-zero bits and thus saves more area with fewer non-zero filter coefficients, whereas the size of LUT-based DA is not influenced by the number of non-zero bits of filter coefficients. Another reason is that the area of adder-based DA architecture can be further minimized by bit-level common subexpression sharing [16, 17, 47]. For instance, word-level common subexpression sharing can be explained in the following example. The filtering operation

$$y[n] = x[n]+x[n] << 1+x[n] << 3+x[n-1]+x[n-1] << 2+x[n-2]+x[n-2] << 1, \tag{11}$$

where '$<< b$' denotes a b digit shift-left operation. If we define $w[n]$ as,

$$w[n] = x[n] << 1 + x[n-1], \tag{12}$$

then Equation (11) can be represented as

$$y[n] = w[n] + w[n-1] << 1 + x[n] + x[n] << 3 + x[n-2]. \tag{13}$$

Direct implementation of Equation (11) requires six adders while the implementation of Equation (13) requires only four adders. The additional intermediate delay problem [47] was addressed by the introduction of vertical subexpression sharing [15], ensuring bit-level computations and communications suitable for bit-level grain FPGA.

2.3 DA for High-Order FIR Filter

It must be noted that, as filter size increases, the memory requirements of LUT-based DA architecture grow exponentially. This problem may be alleviated by breaking a larger DA base unit into smaller DA base units that require tractable memory sizes and then summing the outputs of these units. This technique has been referred to as a “multiple memory bank,” or the “partial sum technique” [103].

The summation in the square braces in Equation (7) may be split so that the K-tap filter is divided into m smaller filters, each with k-tap DA base units ($K = m \times k$). Here it is assumed that K is not prime. Thus, Equation (7) can be written as

$$y[n] = -\left(\sum_{j=0}^{m-1}\left[\sum_{i=jk}^{(j+1)k-1} b_{i,0}w_i\right]\right) + \sum_{l=1}^{B-1}\left(\sum_{j=0}^{m-1}\left[\sum_{i=jk}^{(j+1)k-1} b_{i,l}w_i\right]\right)2^{-l}. \tag{14}$$

The terms in parentheses in Equation (14) may be implemented using m DA base units, each implementing the expression in brackets. In Figure 10, DA architecture based on Equation (14) is shown for 4-tap FIR filters when $m = 2$ and $k = 2$. In addition to the three units of the original DA architecture, an adder tree unit of m-depth is inserted into K-tap filters consisting of m units of k-tap DA base units. The area-efficient DA architecture for the partial sum technique approach should satisfy the following conditions: first, that the control of the adder/shifter unit be independent of the input samples stored in the shifter register unit; second, that the control of the adder/shifter unit be independent of the filter coefficients; third, that a global adder/shifter unit and a global adder/shifter be employed; fourth, that each k-tap DA base unit, storing unique k number of filter coefficients, be independent of the remaining DA base units; and finally, that the adder tree unit, consisting of $m - 1$ number of adders, be independent of the shift register unit.

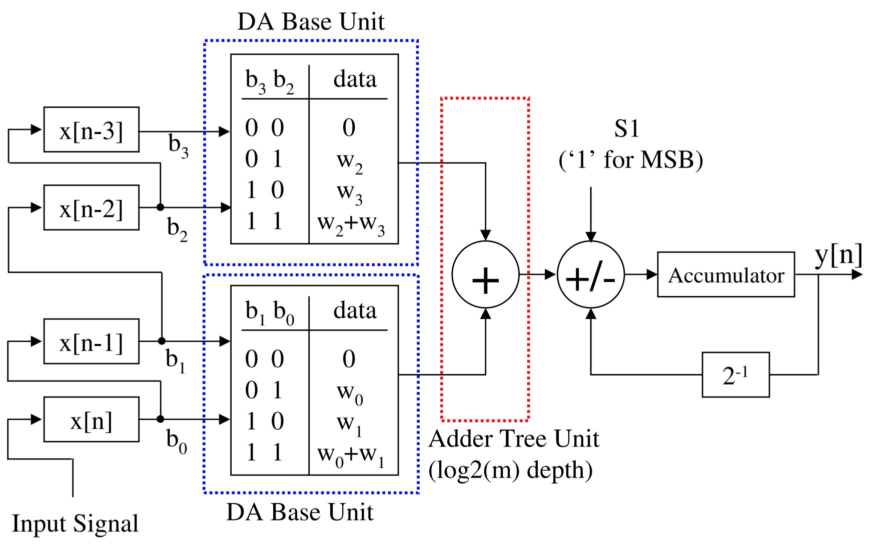


Figure 10. Block diagram of a 4**-tap DA FIR filter when** m = 2 **and** k = 2**. The DA block, which uses the partial sum technique, consists of four small units: the shift register unit, the DA base unit, the adder tree unit, and the adder/shifter unit.**

For the K-tap filter divided into m units of k-tap DA base units ($K = m \times k$), the total memory requirement would be $m \times 2^k$ memory words. The total number of clock cycles required for this implementation would be $B + \lceil \log_2(m) \rceil$; the additional second term is the number of clock cycles required for the adder tree unit to calculate the sums of the DA base units. Thus, the decrease in throughput of this implementation is marginal. For instance, if $K = 128$, then instead of 2^{128} in a full LUT implementation, one can choose $k = 4$ and $m = 32$, which would require only 512 memory words. The number of clock cycles required for this implementation would be 21 clock cycles compared to 16 clock cycles for a single LUT implementation.

2.4 DA-Offset Binary Coding (DA-OBC)

The LUT size can be reduced by interpreting input samples as the offset binary code, $\{1, -1\}$. This method is called "DA-offset binary coding (DA-OBC)" [20, 55, 96]. In DA-OBC, $x[n-k]$ is written as,

$$x[n-k] = \frac{1}{2}[x[n-k] - (-x[n-k])], \tag{15}$$

and the negative of $x[n-k]$ is written as

$$-x[n-k] = -\overline{b_{k,0}} + \sum_{l=1}^{B-1} \overline{b_{k,l}} 2^{-l} + 2^{-(B-1)}, \tag{16}$$

where the overscore indicates one's complement of a binary bit. Plugging Equation (16) into Equation (15) yields,

$$\begin{aligned} x[n-k] &= \frac{1}{2}\Big[-(b_{k,0} - \overline{b_{k,0}}) + \sum_{l=1}^{B-1}(b_{k,l} - \overline{b_{k,l}})2^{-l} - 2^{-(B-1)}\Big] \\ &= \frac{1}{2}\Big[\underbrace{-(b_{k,0} - \overline{b_{k,0}})}_{d_{k,0}} + \sum_{l=1}^{B-1}\underbrace{(b_{k,l} - \overline{b_{k,l}})}_{d_{k,l}}2^{-l} - 2^{-(B-1)}\Big] \\ &= \frac{1}{2}\Big[\sum_{l=0}^{B-1} d_{k,l}2^{-l} - 2^{-(B-1)}\Big], \end{aligned} \tag{17}$$

where new variables, $d_{k,n}$, satisfy $d_{k,n} \in \{1, -1\}$. Finally, $y[n]$ can be rewritten as

$$\begin{aligned} y[n] &= \frac{1}{2}\sum_{k=0}^{K-1} w_k\Big[\sum_{l=0}^{B-1} d_{k,l}2^{-l} + -2^{-(B-1)}\Big] \\ &= \sum_{l=0}^{B-1}\underbrace{\Big(\sum_{k=0}^{K-1}\frac{1}{2}w_k d_{k,l}\Big)}_{D_l}2^{-l} + \underbrace{\Big(-\frac{1}{2}\sum_{k=0}^{K-1} w_k\Big)}_{D_{\text{init}}}2^{-(B-1)} \\ &= \sum_{l=0}^{B-1} D_l 2^{-l} + D_{\text{init}}2^{-(B-1)}. \end{aligned} \tag{18}$$

Now, D_l also can be pre-computed and stored in an LUT since D_l only takes one of 2^K possible combination sums of filter coefficients with $d_{k,n} \in \{-1, 1\}$. Direct LUT implementation of D_l is shown in Table 3. The size of Table 3 can be reduced by half due to the fact that the lower half of the LUT is a mirror version of the upper half of the LUT, but with reversed signs.

Figure 11 illustrates the block diagram of DA-OBC for a 4-tap FIR filter. The LUT size of DA-OBC is half of the LUT size of DA using the mirror image of Table 3 at the cost of an additional four (K) XORs. The control signal, S2, is only set to 1 for the LSBs of the input samples to select D_{init}. Unlike the original LUT-based DA architecture, the adder/shifter unit of DA-OBC is directly controlled by an input signal through XOR and

Table 3. Original LUT contents of DA-OBC. The LUT size can be reduced by half with additional circuitry using the observation that the lower half of the LUT is a mirror image of the upper half of the LUT, but with reversed signs.

Addresses				
$b_{3,n}$	$b_{2,n}$	$b_{1,n}$	$b_{0,n}$	Memory Contents
0	0	0	0	$-(w_3 + w_2 + w_1 + w_0)/2$
0	0	0	1	$-(w_3 + w_2 + w_1 - w_0)/2$
0	0	1	0	$-(w_3 + w_2 - w_1 + w_0)/2$
0	0	1	1	$-(w_3 + w_2 - w_1 - w_0)/2$
0	1	0	0	$-(w_3 - w_2 + w_1 + w_0)/2$
0	1	0	1	$-(w_3 - w_2 + w_1 - w_0)/2$
0	1	1	0	$-(w_3 - w_2 - w_1 + w_0)/2$
0	1	1	1	$-(w_3 - w_2 - w_1 - w_0)/2$
1	0	0	0	$(w_3 - w_2 - w_1 - w_0)/2$
1	0	0	1	$(w_3 - w_2 - w_1 + w_0)/2$
1	0	1	0	$(w_3 - w_2 + w_1 - w_0)/2$
1	0	1	1	$(w_3 - w_2 + w_1 + w_0)/2$
1	1	0	0	$(w_3 + w_2 - w_1 - w_0)/2$
1	1	0	1	$(w_3 + w_2 - w_1 + w_0)/2$
1	1	1	0	$(w_3 + w_2 + w_1 - w_0)/2$
1	1	1	1	$(w_3 + w_2 + w_1 + w_0)/2$

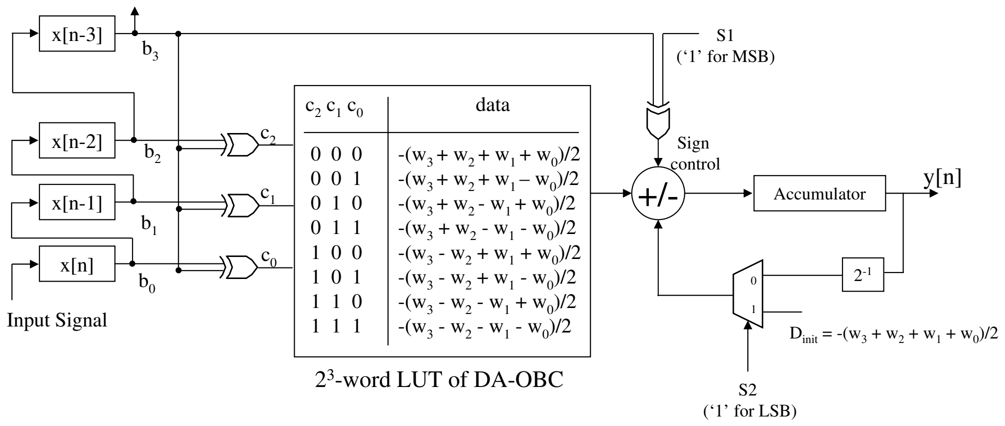
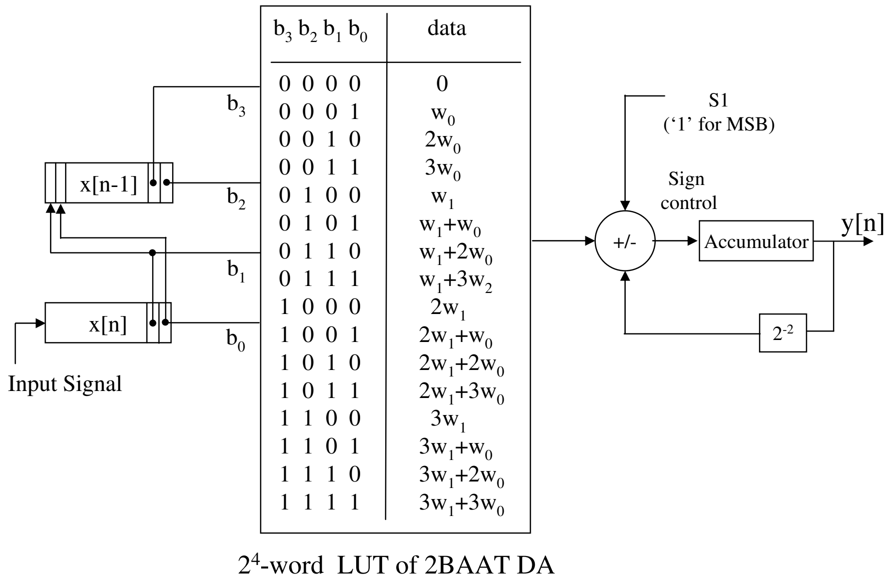

x[n-3]
b3
x[n-2]
b2
x[n-1]
b1
x[n]
b0
Input Signal
c2
c1
c0
c2 c1 c0
data
0 0 0 -(w3 + w2 + w1 + w0)/2
0 0 1 -(w3 + w2 + w1 – w0)/2
0 1 0 -(w3 + w2 - w1 + w0)/2
0 1 1 -(w3 + w2 - w1 - w0)/2
1 0 0 -(w3 - w2 + w1 + w0)/2
1 0 1 -(w3 - w2 + w1 - w0)/2
1 1 0 -(w3 - w2 - w1 + w0)/2
1 1 1 -(w3 - w2 - w1 - w0)/2
2^3-word LUT of DA-OBC
S1
('1' for MSB)
Sign control
+/-
Accumulator
y[n]
2^-1
0
1
Dinit = -(w3 + w2 + w1 + w0)/2
S2
('1' for LSB)

Figure 11. Block diagram of a 4-tap ($K = 4$) DA-OBC FIR filter. The LUT size of DA-OBC is half of the LUT size of DA using the mirror image of Table 3 at the cost of an additional four (K) XORs. The control signal, S2, is only set to 1 for the LSBs of the input samples to select D_{init}.

the sum of the weight coefficients through 2x1 MUX. Thus, DA-OBC architecture is not suitable for an area-efficient modular approach such as the partial sum technique without modification.

2.5 High-Speed DA

Additional speed-up of DA implementation is possible by using n bit-at-a-time (BAAT) access to the LUT with $n > 1$. The clock cycle required for a filtering operation can be reduced to $\lceil B/n \rceil$ clock cycles from B clock cycles. However, due to its even faster-increasing memory size than 1BAAT DA, to the best of our knowledge, no hardware implementation of nBAAT DA ($n > 1$) has been reported in the literature for relatively high-order digital filters. The LUT size of nBAAT DA for K-tap FIR filter implementation increases substantially from 2^K to 2^{nK} with a single LUT-based DA, and from $m \times 2^k$ to $m \times 2^{nk}$ with the partial sum technique, having m number of k-tap DA base units ($K = m \times k$). For instance, the size of a single LUT for a 4-tap FIR filter increases from 2^4 words for 1BAAT access to 2^8 words for 2BAAT access. An example of 2BAAT DA architecture for a 2-tap FIR



$b_3\ b_2\ b_1\ b_0$	data
0 0 0 0	0
0 0 0 1	w_0
0 0 1 0	$2w_0$
0 0 1 1	$3w_0$
0 1 0 0	w_1
0 1 0 1	w_1+w_0
0 1 1 0	w_1+2w_0
0 1 1 1	w_1+3w_2
1 0 0 0	$2w_1$
1 0 0 1	$2w_1+w_0$
1 0 1 0	$2w_1+2w_0$
1 0 1 1	$2w_1+3w_0$
1 1 0 0	$3w_1$
1 1 0 1	$3w_1+w_0$
1 1 1 0	$3w_1+2w_0$
1 1 1 1	$3w_1+3w_0$

2^4-word LUT of 2BAAT DA

Figure 12. Block diagram of a 2-tap DA FIR filter with 2BAAT access to the LUT. The filtering operation can terminate at $\lceil B/n \rceil$ clock cycles at the expense of exponentially increasing memory usage.

filter is shown in Figure 12 . More architectures that combine the DA-OBC and *n*BAAT approaches are also available in [96].

CHAPTER 3
PROGRAMMABLE CONTINUOUS-TIME ANS

3.1 Background of ANS

While most noise suppression methods focus on discrete-time audio signal processing, a continuous-time ANS based on the CADSP framework is introduced in this section. The system, which is for single-channel background audio noise suppression, is unique in that it is implemented in programmable analog VLSI circuits operating at a subthreshold range [68] for low-power consumption. By performing a significant portion of the processing in low-power analog circuits [50], the overall functionality of an entire system can be enhanced by utilizing analog/digital computation in a mutually beneficial way.

3.2 Continuous-Time ANS

A common model for a noisy signal, $x(t)$, is a signal, $s(t)$, plus additive noise, $n(t)$, that is uncorrelated with the signal

$$x(t) = s(t) + n(t). \tag{19}$$

The goal of the research is to design a real-time, low-power system that generates some optimal estimate, $\hat{s}(t)$, of $s(t)$ from $x(t)$. Additive background noise is assumed to be uncorrelated and stationary over a long period of time, relative to the short-term stationary patterns of normal speech. The signal estimate, $\hat{s}(t)$, can be found in the frequency domain by spectral subtraction [12, 34, 86] or by applying a Wiener filter gain [32, 66]. The Wiener gain can be expressed in terms of the frequency-dependent SNR as

$$H(\omega) = \frac{\Gamma^2(\omega)}{1 + \Gamma^2(\omega)}, \tag{20}$$

where $\Gamma^2(\omega) = \Phi_s(\omega)/\Phi_n(\omega)$ with $\Phi_s(\omega)$ and $\Phi_n(\omega)$ representing the power spectral densities (PSDs) of the signal and noise, respectively.

In this section, we are interested in a single-channel ANS algorithm with short-time

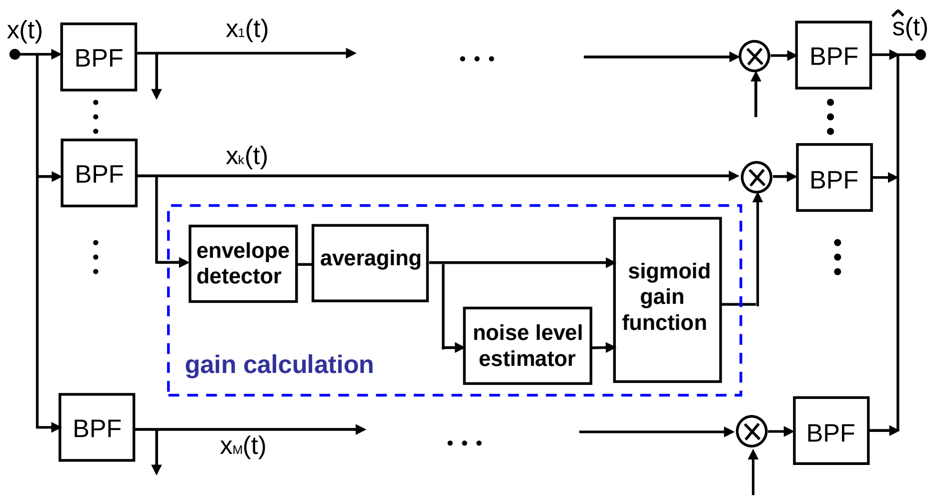


Figure 13. Block diagram of the continuous-time ANS system. At each subband, gain is calculated from the non-linear gain function. The gain is then multiplied by a subband signal and the results are summed to build a full-band signal estimate.

spectral modification techniques using a continuous-time speech signal. The method used in the research [7, 30] is based on Schroeder [82, 83] and Diethorn's noise suppression system [26], but differs in its filter bank structure, type of gain function, method of the estimating noise, and method of estimating the subband gain.

Frequency-domain processing is accomplished using a one-third octave filter bank. In each subband, a noisy signal envelope is detected and smoothed. From the smoothed subband signal envelope, the noise envelope is estimated in each subband independently. The SNR in each band is estimated from the noisy signal and noise envelopes. A non-linear (sigmoid) gain function is used to approximate the Wiener gain. Finally, the original band-limited signal in each band is multiplied by its corresponding gain, and the result is summed to construct the full-band "clean" signal estimate. The overall structure of the system is shown in Figure 13, with a more detailed view of the gain calculation block of a single band shown in Figure 14.

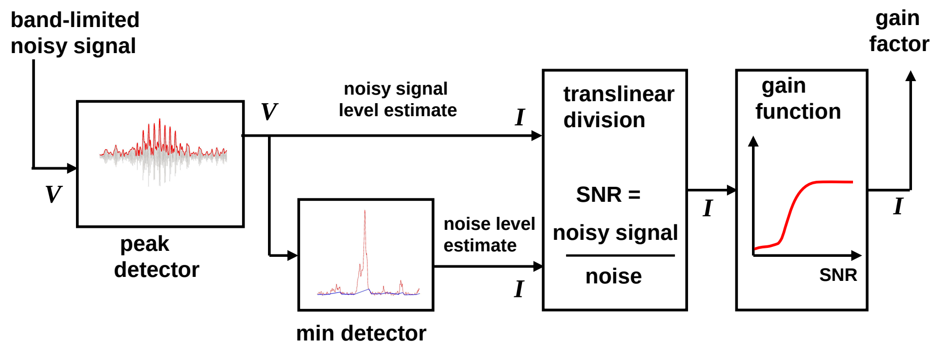


Figure 14. Detailed view of the subband gain calculation block. Within each frequency band, the noisy signal envelope is estimated using a peak detector. Based on the voltage output of the peak detector, the noise level is estimated using a minimum detector. Translinear division circuit outputs a current that represents the estimated SNR. A nonlinear function is applied to the SNR current, and the resulting gain factor is multiplied with the band-limited noisy signal to produce a band-limited “clean” signal.

3.3 Analog VLSI Implementation

A continuous-time ANS algorithm has been fabricated on a $0.5\mu m$ CMOS VLSI chip. Since each subband operates independently of the others in the array, we have 32 parallel signal processors operating simultaneously on 32 band-limited signals. The following sections elaborate on the algorithm and relate the circuits that perform the underlying functions.

3.3.1 Frequency Decomposition

The filter bank in Figure 13 separates the noisy signal into 32 subbands whose center frequencies are logarithmically spaced similar to the human auditory system [6], resulting in approximately one-third octave spacing in the center frequencies of the filter bank. By using one-third octave filters, any frequency distortions whose bandwidth is on the order of the bandwidth of each band also lie within the same critical band and can be minimized for the perceptual impact [6].

3.3.1.1 Band-pass filter element

The implementation of the band-pass filter used to separate a wideband signal into subband signals is based on the capacitively-coupled current conveyer (C^4) presented in [49], characterized in [44, 87], and shown in Figure 15(a). The C^4 is a capacitively-based band-pass filter with electronically tunable corner frequencies that are independent of each other. The

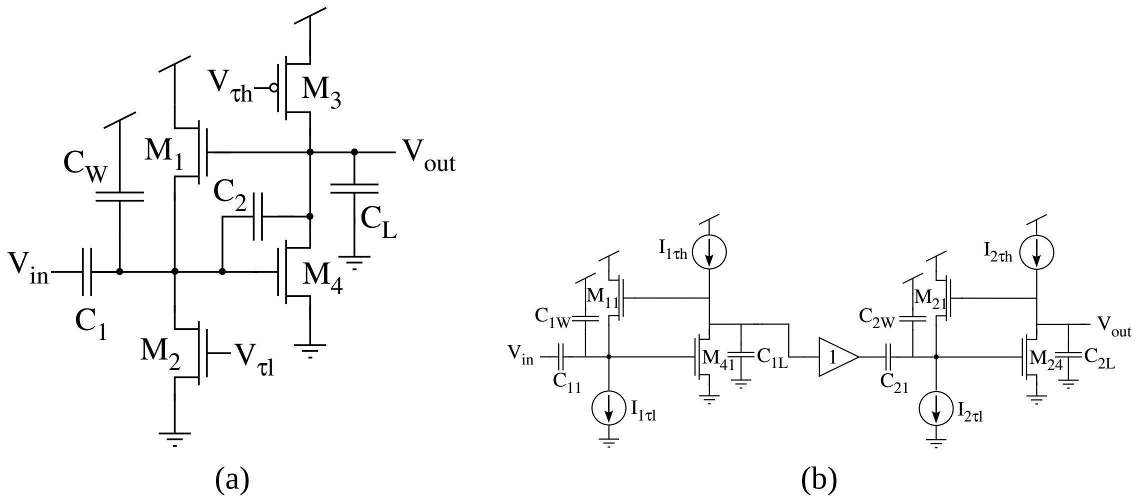


Figure 15. (a) C^4 takes on the form of a band-pass filter within the region of interest with ±20 dB/decade slopes outside the pass band. The mid-band gain is $-C_1/C_2$. (b) The schematic of the C^4 second-order section (C^4 SOS). Tuning of the bias currents sources is accomplished by programming floating-gate transistors to the desired current.

frequency response of the C^4 is governed by

$$\frac{V_{out}}{V_{in}} = -\frac{C_1}{C_2}\frac{s\tau_l(1 - s\tau_f)}{s^2\tau_h\tau_l + s(\tau_l + \tau_f(\frac{C_o}{\kappa C_2} - 1)) + 1}, \tag{21}$$

where the time constants are given by

$$\tau_l = \frac{C_2 U_T}{\kappa I_{\tau_l}} \qquad \tau_f = \frac{C_2 U_T}{\kappa I_{\tau_h}} \qquad \tau_h = \frac{C_T C_O - C_2^2}{C_2}\frac{U_T}{\kappa I_{\tau_h}},$$

and the total capacitance, C_T, and the output capacitance, C_O, are defined as $C_T = C_1 + C_2 + C_W$ and $C_O = C_2 + C_L$, respectively. The currents I_{τ_l} and I_{τ_h} are the currents through M_2 and M_3, respectively, as shown in Figure 15(a). With normal usage, τ_f is so fast that the zero it produces lies far outside of the operating range. Hence, the C^4 takes on the form of a band-pass filter within the region of interest with ±20 dB/decade slopes outside the pass band. The mid-band gain is $-C_1/C_2$.

Removing the feedback capacitor C_2, a configuration called a *vanilla* C^4, transforms the C^4 into a high-gain filter with a much larger Q peak value [87]. The C^4 second-order section (C^4 SOS) [31, 45], shown in Figure 15(b), is simply a cascade of two vanilla C^{4}s isolated by a buffer. By tuning each of the vanilla C^{4}s so that they have identical time

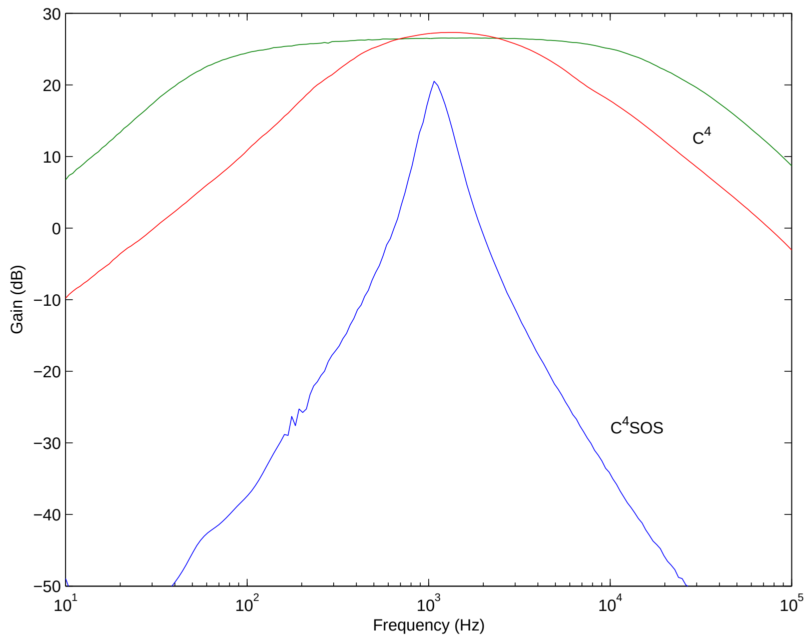


Figure 16. Magnitude responses of C^4 and C^4 SOS. C^4 takes on the form of a band-pass filter within the region of interest with ±20 dB/decade slopes outside the pass band. The mid-band gain is $-C_1/C_2$. By tuning each of the vanilla C^{4}s so that they have identical time constants, the overall response of the C^4 SOS has ±40 dB/decade slopes outside the pass band and a potentially high Q peak.

constants, the overall response of the C^4 SOS has ±40 dB/decade slopes outside the pass band and a potentially high Q peak. A high Q peak is useful in many applications because it helps isolate the respective center frequency. More information about the details of the C^4 SOS is available in [45]. Tuning of the bias currents is accomplished by programming floating-gate transistors to the desired current, shown in Figure 15(b) as current sources. Figure 16 shows the magnitude responses of C^4 and C^4 SOS. C^4 has ±20 dB/decade slopes outside the pass band while C^4 SOS has ±40 dB/decade slopes outside the pass band.

3.3.1.2 Programming of the filter bank

Programming a large number of floating-gate devices, as would be required for a programmable filter bank, requires that the floating-gate transistors be arranged in an array for ease of programming as is shown in Figure 17 [48, 89]. Tunnelling can be used to program currents accurately, but selectivity is not completely controllable. When one element is tunnelled, the charge of all the other devices in the array will be altered. As a result, the

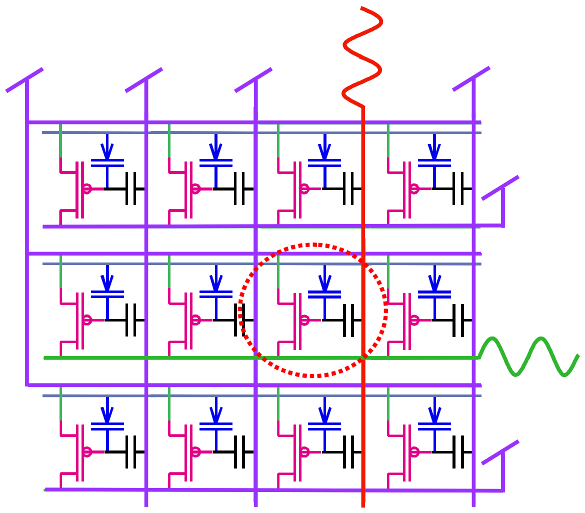


Figure 17. Architecture used for programming the arrays of floating-gate devices. Two conditions must exist for injection to occur: (1) a high source-to-drain field to make the electron "hot" and (2) a channel for device current to flow. Because these conditions can be created orthogonally to each other with source-to-drain voltage and gate voltage (to modulate the channel), a single element can be selected for injection or measurement.

tunnelling operation is reserved for "erasing" a charge that is already on the floating gate and not for very accurate programming.

However, hot-electron injection allows complete selectivity of an individual element. Hence, injection is used for precise and accurate programming of floating-gate arrays [89]. The method of programming by injection, depicted in Figure 17, is explained as follows. After selecting a specific floating-gate device, the gate lines of all the columns not containing that device are connected to V_{DD}. Then all the drains of the rows not containing the selected element are also connected to V_{DD}. Therefore, the gates, the drains, or both of all the non-elected elements are connected to V_{DD}, ensuring that no appreciable current will flow in any of the other devices, indicating that they cannot be injected, and that their floating-gate charge cannot be changed. A voltage can be supplied to the input of the selected element, allowing current to flow. Finally, the drain of the selected element is pulsed down so that the source-to-drain voltage is temporarily large. The two criteria for injection are met, so electrons will be added to the floating gate of the selected element. Any such element in the array can be chosen and programmed, and when all the currents have been set to the desired values, the terminals of the transistor are connected to the rest of the circuit in which they are operating as fully functional transistors.

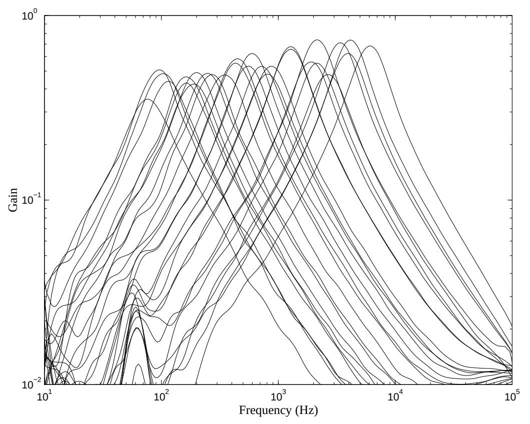
A filter bank of 32 C^4 SOSs was fabricated with a $0.5\mu m$ process available through MOSIS [46]. While band-pass filters can be programmed to any desired center frequency spacing and bandwidth, exponentially-spaced center frequencies with narrow bandwidths and moderate Qs are programmed, as this type of configuration is highly advantageous in audio signal processing. This configuration, which closely models the biology of the human cochlea, allows subbands of frequency to be independently manipulated since real-time frequency decomposition occurs.

Figure 18 shows the frequency responses of each of the 32 filter taps. Only the output from the first stage (a single C^4) is shown. The filter taps are programmed so that the $I_{\tau l}$s and $I_{\tau h}$s for each have exponentially-spaced currents with 95% programming accuracy.

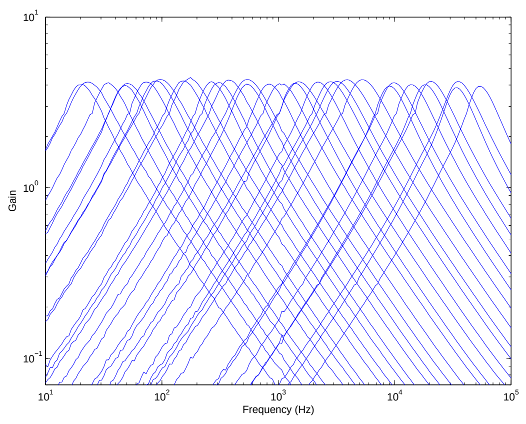
Figure 18(a) shows the magnitude response of the original filter bank [45] that was programmed using large resistive lines to bias the transistors, and Figure 18(b) shows the enhanced magnitude response of the filter bank whose corner frequencies are programmed using floating-gate transistors. Figure 18(b) illustrates a fundamental design issue with analog circuits. Regardless of how accurately biases can be set, circuit performance is affected by mismatches that occur during the fabrication process. The programmable band-pass filter bank is monotonically spaced, and this monotonicity and spacing is simply programmed by the floating-gate biases. Figure 18(b) shows that corner frequencies are not perfectly spaced, but that is of no concern because the errors due to the mismatch of transistor and capacitor sizes can be programmed out with the floating gates.

3.3.2 Signal and Noise Level Estimation

After the incoming noisy signal is band limited by the C^4 SOS filter bank, a gain factor is then calculated based on the characteristics of the noisy signal of each subband. The first step in the gain calculation is to find the noisy signal envelope estimate, $\hat{e}_x(k,t)$, when the k-th subband noisy audio signal, $x(k,t)$, is represented as $x(k,t) = e_x(k,t)v_x(k,t)$ with



(a)



(b)

Figure 18. Magnitude responses of filter bank of 32 programmable vanilla C^{4}s. (a) Magnitude response of the original filter bank [45], which is programmed using large resistive lines to bias the transistors. (b) Magnitude response of the enhanced filter bank in which the time constants are set by floating-gate transistors. Corner frequencies are programmed to be exponentially spaced with 95% programming accuracy. This programmed filter bank shows a marked improvement over the original, non-programmable filter bank.

$v_x(k,t)$, a band-limited signal having a relatively flat magnitude and $e_x(k,t)$, the envelope variation over time. The noise envelope estimate, $\hat{e}_n(k,t)$, is then calculated using a technique closely related to the minimum statistics method [65], whereby the noise is found from the envelope of the noisy signal. The SNR of each subband is estimated from both $\hat{e}_x(k,t)$ and $\hat{e}_n(k,t)$.

$$\hat{\Gamma}(k,t) = \frac{\hat{e}_s(k,t)}{\hat{e}_n(k,t)} \approx \frac{\hat{e}_x(k,t) - \hat{e}_n(k,t)}{\hat{e}_n(k,t)} = \frac{\hat{e}_x(k,t)}{\hat{e}_n(k,t)} - 1, \tag{22}$$

where $\hat{e}_x(k,t)$ is represented as the sum of the signal envelope estimate, $\hat{e}_s(k,t)$, and the noise envelope estimate, $\hat{e}_n(k,t)$.

3.3.2.1 Noisy signal envelope estimate using a peak detector

A peak detector is used to estimate the envelope of each noisy subband signal [31]. When a speech signal is active in the input, the peak detector follows the envelope of the signal, rising rapidly with increasing signal amplitude and decaying slowly enough to result in a smooth envelope. The peak detector also follows the level of the additive noise, particularly in times when the signal is absent. Each peak detector has a programmable time constant

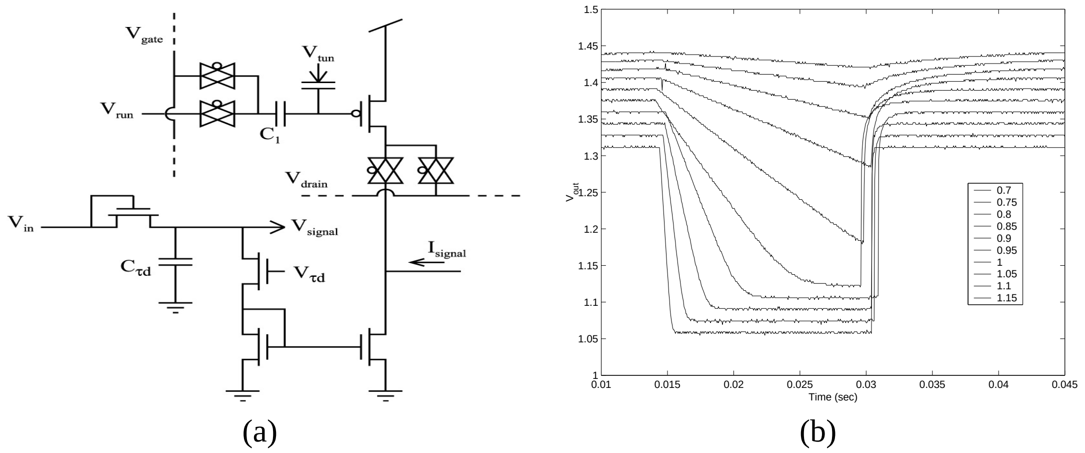


Figure 19. (a) The peak detector circuit shown in this schematic tracks the incoming band-limited noisy signal, yielding the noisy signal envelope estimate. Two outputs in this circuit are the current I_{signal}, which goes into the division circuit, and the voltage V_{signal}, which is the input to the minimum detector. The bias voltage V_{τ_d} sets the time constant at which the output will decay after a peak. (b) Step responses of the peak detector for various time constants.

so that an appropriate smoothed envelope can be determined for each of the bands [100].

When an initial noisy signal envelope estimate, $\hat{e}_x(k,t)$, is available, $\hat{e}_x(k,t)$ is averaged via low-pass filtering. Other averaging methods are analyzed in [42]. Averaging is beneficial to the noise envelope estimate, which is assumed to be stationary. The averaged noisy signal estimate, $\overline{e}_x(k,t)$, which will be used to obtain a smoothed SNR, is very critical to making the ANS system robust to the artifacts such as musical noise and missing consonants. The circuit diagram of the peak detector block [31] and its step responses for various time constants are shown in Figure 19.

3.3.2.2 Noise envelope estimate using a minimum detector

One effective method of noise-level estimation is the minimum statistics approach [65]. An approximation of this approach is to use an inverted peak detector or minimum detector with a long time constant on the averaged noisy signal envelope estimate, $\overline{e}_x(k,t)$ [8]. The inverted peak detector operates on $\overline{e}_x(k,t)$, maintaining an estimate of the minimum value, which is assumed to be the noise floor. A “minimum detector” is therefore used to estimate the averaged noise envelope, $\overline{e}_n(k,t)$, by following the minimum values of $\overline{e}_x(k,t)$. Figure 20

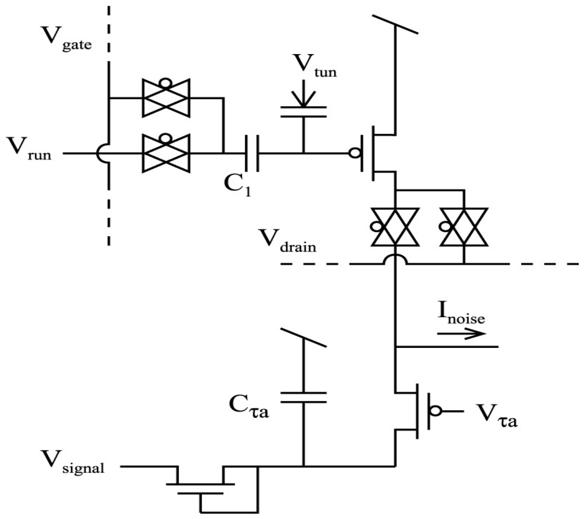


Figure 20. Minimum detector is biased with V_{τ_a} to have a much slower time constant than the peak detector, so as to follow the slowly changing curve that results at the bottom of the noisy signal envelope. The output I_{noise} is an estimate of the noise envelope.

shows the circuit diagram of the minimum detector [31]. A bias voltage sets the attack time constant to run more slowly than that of the peak detector so that the slow changes found in the amplitude of relatively stationary noise can be followed. When the signal is present, the output will maintain a slow-rising level; when the signal is not present, the minimum detector will track the noise level.

3.3.3 SNR Calculation and Normalization

After $\overline{e}_x(k,t)$ and $\overline{e}_n(k,t)$ are obtained, multiplication and division operations can be performed using the translinear principle [84] for the SNR calculation. Figure 21 is the circuit implementation of a division operation in which the estimated SNR, I_{SNR}, has the following equation:

$$I_{SNR} = I_{scale}\frac{I_{signal}}{I_{noise}} - I_{scale}. \tag{23}$$

The current I_{scale}, set by the bias voltage V_{scale}, and is used to put the output current into the proper range for the gain function.

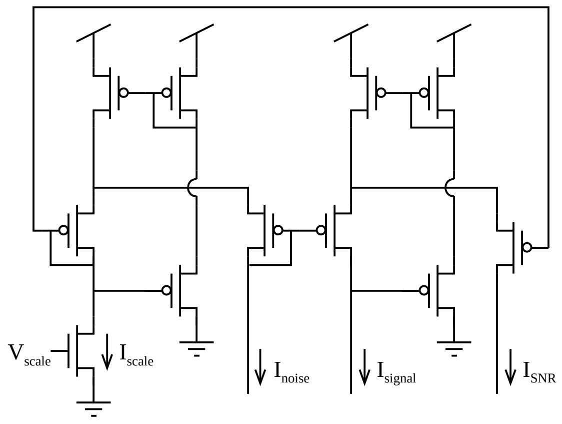
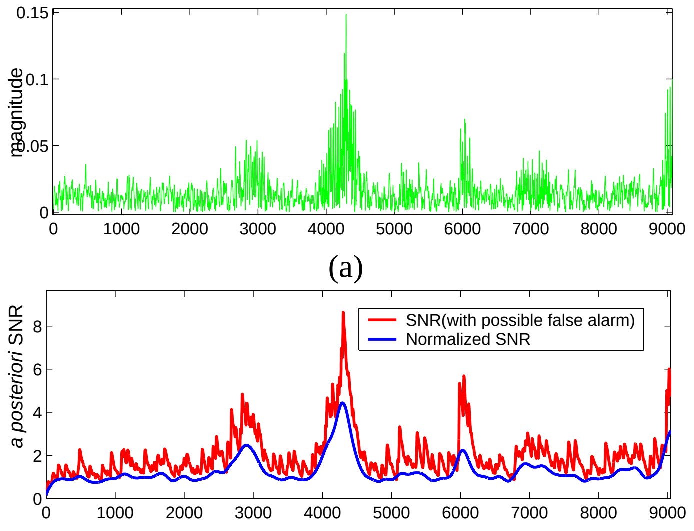


Figure 21. The translinear circuit implements the division of I_{noise} into I_{signal}, yielding a current representing the SNR, I_{SNR}. I_{scale} is subtracted from the output current to yield the function described in Equation (22). The voltage V_{scale} sets the bias current I_{scale}, which is used to put I_{SNR} into the proper range for the gain function circuit.

Diethorn [26] estimated $\hat{e}_x(k,t)$ and $\hat{e}_n(k,t)$ in parallel directly from a band-limited signal. This approach, however, when implemented in a continuous-time domain with relatively simple and sample-by-sample analog circuit estimators, causes many false alarms, shown in Figure 22(b), and deteriorates the overall intelligibility. To decrease the number of false alarms, $\overline{e}_x(k,t)$ is estimated from $x(k,t)$, and $\overline{e}_n(k,t)$ is estimated not from $x(k,t)$ but from $\overline{e}_x(k,t)$, as shown in both Figures 13 and 14. This approach, also called the "normalized SNR method," helps overall SNR to be unity or very close to unity when the signal is assumed to be absent. The normalized SNR method forces the SNR of the noise-only period to unity, independent of input noise variance, and accordingly the suppression rate will be consistently the same. This normalized SNR is used as the input of the sigmoid gain function, which is explained in the next section.

3.3.4 Programmable Non-Linear Gain Function

The final elements of the gain calculation algorithm are the gain function and multiplier. Several different gain functions may be used, but all of them have the general characteristics of low gain for low SNR and high gain (at or near unity) for high SNR, with varying



(a) (b)

Figure 22. (a) Averaged noisy signal envelope, $\bar{e}_x(k,t)$. The data are obtained from the computer simulation as explained in Sec 3.4. (b) SNR obtained from the algorithm of [26] and normalized SNR obtained from the simulation of the proposed continuous-time ANS. The normalized SNR method generates unity SNR or very close to unity SNR when the signal is assumed to be absent.

smoothing between these two regions. The transition between low gain and high gain must be smooth to avoid adverse auditory effects, yet it must quickly follow the SNR changes to ensure an acceptable amount of noise suppression.

A few examples of well-known gain functions are as follows. The Wiener gain, Equation (20), is widely used in discrete-time ANS since it minimizes error energy when the input process is non-stationary. Bi-linear gain function [26], $H_{bi}(t)$, is also frequently used in discrete-time noise suppression methods.

$$H_{bi}(t) = min\left[1, \left\{\frac{\bar{e}_x(k,t)}{\gamma\bar{e}_n(k,t)}\right\}\right], \tag{24}$$

where γ is the threshold that specifies the certainty of the presence of speech. The bi-linear gain function, however, suffers severe loss of signal magnitude at low SNRs, particularly when big γ is used for more noise suppression. To minimize this signal energy reduction at low SNRs, we proposed a sigmoid gain function [100], $H_{sig}(t)$, which is suitable for the continuous-time circuit implementation because its transfer function is similar to that of

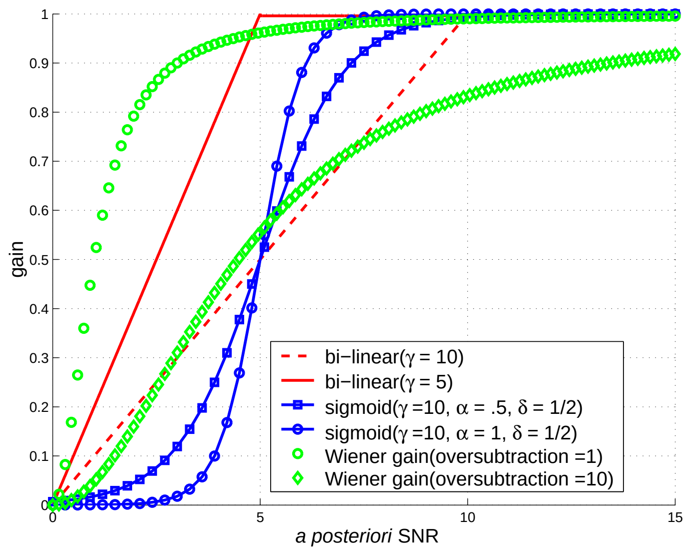


Figure 23. Wiener, bi-linear, and sigmoid gain functions. Wiener gain minimizes error energy when the input process is non-stationary. Bi-linear gain function is easy to program but suffers severe signal magnitude reduction at low SNRs, especially, when big γ is used for more noise suppression. Sigmoid gain function has three control parameters and thus can achieve a low gain at low SNRs and a high gain (at or near unity) for high SNRs.

current mirror circuits.

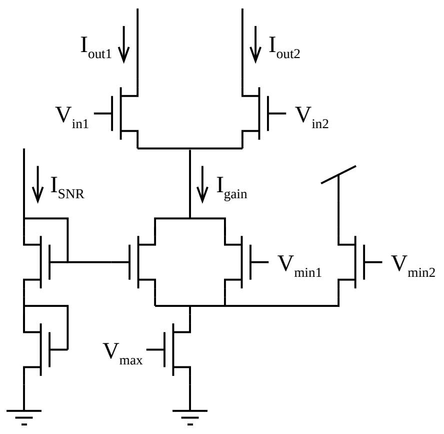
$$H_{sig}(t) = \frac{1}{2}\left\{\tanh\left(\alpha\left(\frac{\bar{e}_x(k,t)}{\bar{e}_n(k,t)} - \phi\right)\right)\right\} + \delta, \tag{25}$$

where, α, ϕ, and δ are parameters for slope, horizontal, and vertical shift, respectively.

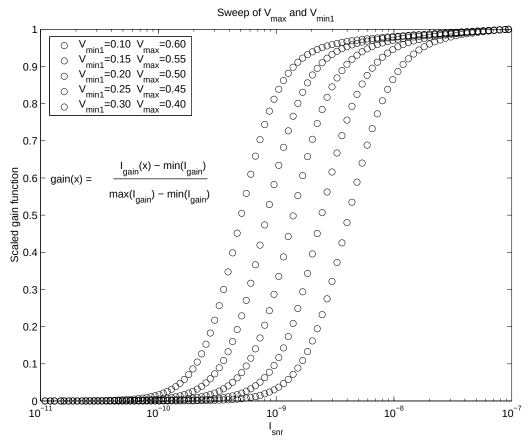
Figure 24(a) shows the combined circuit schematic of the gain function and the multiplier. The input to the multiplier is the band-limited noisy signal, i.e., the output of the C^4 SOS band-pass filter. The transistors at the top of the schematic are the differential pair where the actual multiplication takes place; the multiplier will be described shortly. The transistors below the differential pair create the behavior of the gain function. The output of the gain function circuit, I_{gain}, can be approximated by

$$I_{gain} = \frac{(I_{SNR}^2 + I_{min1})I_{max}}{(I_{SNR}^2 + I_{min1}) + I_{max}}. \tag{26}$$

Equation 26 has almost the exact form of the Wiener gain, with the addition of a bias current that can be used to tune the circuit operation as needed. I_{max} and I_{min1} are set by



(a)



(b)

Figure 24. (a) The schematic of the multiplication and gain function circuit. The output current, $I_{out1} - I_{out2}$, is the product of the band-limited input signal, $V_{in1} - V_{in2}$, and the gain factor, I_{gain}. Each voltage bias sets a current that forms the overall behavior of the gain function circuit. (b) The gain function of the circuit depicted in (a) is plotted versus I_{snr}. At the extremes of the SNR ratio, either a low gain factor or a unity gain factor is the result. Multiple curves show the bias voltages being adjusted.

voltage biases V_{max} and V_{min1} and effectively create the upper and lower bounds of the gain output. The circuit is biased to operate in the linear range of the following equation:

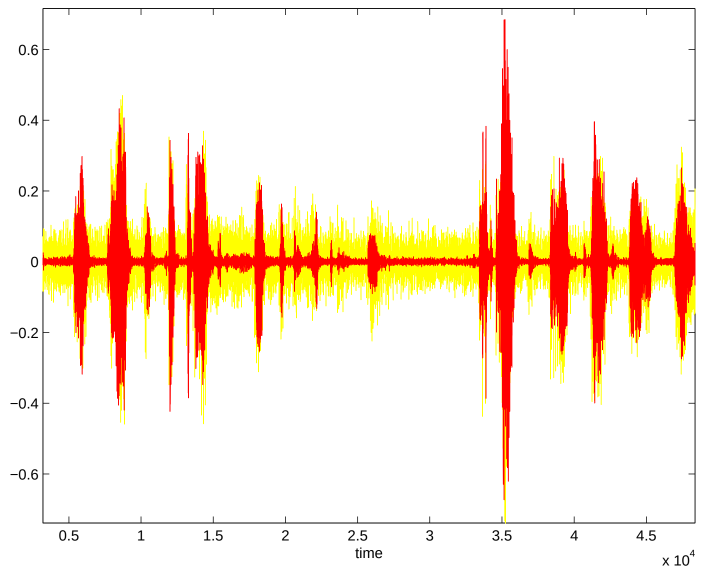
$$I_{out1} - I_{out2} = I_{gain} \cdot \tanh\left(\frac{\kappa(V_{in1} - V_{in2})}{2U_T}\right), \tag{27}$$

where U_T is the thermal voltage of the transistors. Figure 24(b) shows the measured data of the gain functions with different parameters.

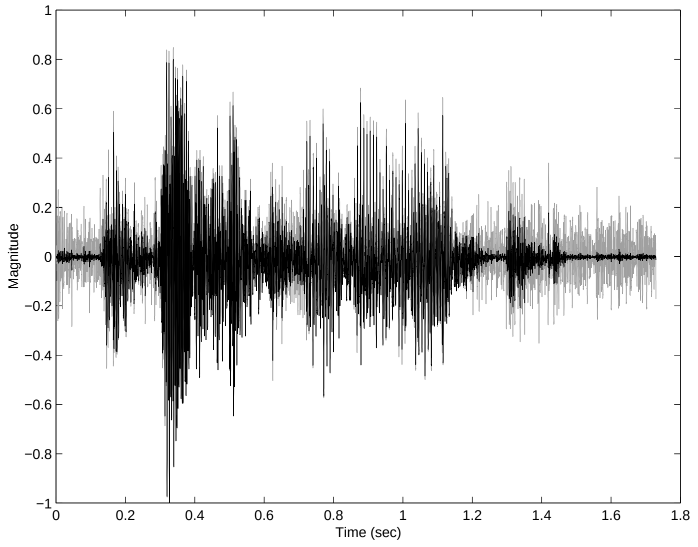
3.4 Simulation and Implementation Results

The initial continuous-time ANS algorithm was functionally simulated in MATLAB. The simulated system contained the same functional blocks, shown in both Figures 13 and 14, and was implemented in the analog system with continuous-time transfer functions converted to discrete-time functions using the bi-linear transform [5]. The sampling rate in the simulator was chosen to be at least four times the required Nyquist rate to avoid the frequency warping that occurs near the Nyquist rate with the bi-linear transform. Multi-rate signal processing was used to improve efficiency, but the high over-sampling rule was always followed. The simulation gain function parameters were chosen so that the attenuation was limited to less than 40 dB, leaving some residual background noise. Noise suppressed output from the simulation overlapped with the original waveform is shown in Figure 25(a). The perceived quality of the noise-suppressed signal is remarkably free from artifacts normally associated with Wiener filtering and spectral subtraction, which can be attributed to the method of frequency decomposition that matches human auditory critical bands and the proportional bandwidths of the subband envelopes.

Figure 25(b) shows an actual measured subband noisy speech signal and a noise suppressed signal processed by the components in our system. The system is effective at adaptively reducing the amplitude of noise-only portions of the signal while leaving the desired portions relatively intact. The power consumption of the noise suppression system is remarkably small because of its subthreshold operation. The core circuitry, assuming 32 frequency bands and a 3.3 V power supply, consumes less than 50 μW. Therefore, integrating this circuit into an existing system will not likely have a measurable impact on battery performance. Any noise or distortion created by the gain calculation circuits minimally affects the output signal because these circuits are not directly in the signal path. While the band-pass filters and the multipliers inject a certain amount of noise into each frequency band, this noise will be averaged out by the summation of the signals at the output of the system. Distortion in the signal path will arise from the band-pass filters and the multipliers. In the band-pass filter array, the C^4 SOS structure is not cascaded, as in the cochlea models [68]. Therefore, there is no distortion or noise accumulation. The distortion level in each frequency band for a 30 mV single-ended input signal is second-harmonic limited at -30 dB at peak. A differential filter bank will eliminate second-harmonic distortion and reduce the third-harmonic level to -40 dB at peak. The distortion introduced by the multiplier is dependent on the output levels of the band-pass filters. If the signal is near 30 mV, third-harmonic distortion will be -20 dB less; however, and if the signal is near 7.5 mV, the third-harmonic distortion will approach -46 dB.



(a)



(b)

Figure 25. (a) Time-domain waveform of original noisy signal (gray) and noise-suppressed signal (dark) from our functional circuit simulation. (b) Noise suppression from the analog VLSI circuit in one frequency band. The light gray waveform is the subband noisy speech input signal and the black waveform is the corresponding subband output, after the gain function has been applied.

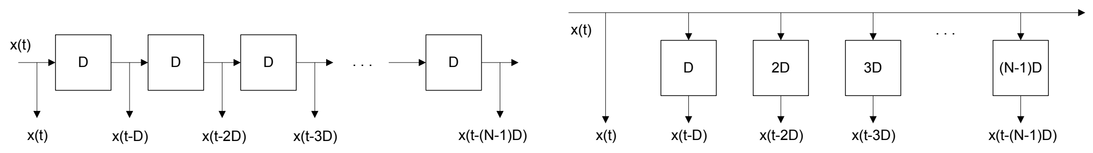
CHAPTER 4

CONTINUOUS-TIME DELAY FILTER

Adaptive filters have gained lots of popularity in the high-quality audio signal processing applications such as adaptive ANS and AEC. The filter order increase of these adaptive filters because of the high sampling frequency has challenged real-time and low-power computing in the digital domain. Analog circuit engineers have made considerable effort to resolve real-time and low-power constraints of the digital-domain approach by implementing adaptive FIR filters with analog circuitry.

The adaptive FIR filter consists of weight adaptation, tapped-delay, and multiplication-and-addition. The multiplication-and-addition is relatively straightforward to implement with analog circuitry. Weight adaptation in the continuous-time domain is needed to solve a differential equation [67] requiring basic circuit primitives such as adders, multipliers, and integrators, which can be efficiently implemented with analog VLSI. Fully differential current-mode implementation of the weight update using the multiple input translinear element (MITE) [70] is explained in [67]. Above all, the implementation of tapped-delayed lines with analog circuitry is challenging since it requires ideal delay elements. Sample mode (discrete mode) delay elements [64] using a charge coupled device (CCD) or a switched capacitor were also investigated in [61]. However, the necessity of clocking and the occurrence of aliasing effects still remain as drawbacks. Several attempts [11, 13] have been made to implement delay elements in a purely continuous-time domain. Delay implementation in the continuous-time domain, however, can experience accumulated noise problems as the number of cascaded delay increases. Figure 26(a) shows a typical tapped-delay line commonly used in standard FIR filtering structures. The accumulated noise problem can be significantly alleviated by a parallel-delay line, shown in Figure 26(b), when the following conditions are satisfied:

- The filter order for multiple delay should be the same as the filter order for unity



(a) (b)

Figure 26. (a) The tapped-delay line where unit delay elements are positioned in tandem. When implemented with analog circuitry, the performance of the tapped-delay line is not robust to accumulated noise. (b) The parallel-delay line, which requires a multiple delay filter, is very robust to accumulated noise, when implemented with analog circuitry.

delay, or at least it should not increase significantly for a large delay.

- Maximum delay should be achieved by a single delay element.
- The performance of the adaptive filter should not be significantly affected by the non-integer fractional delay caused by imperfections of the analog devices.

4.1 Property of The Delay Filter

Most analog delay elements in a continuous-time domain have been implemented with first-order low-pass (Gamma) filters [58], first-order all-pass filters [13], or a combination of first-order low-pass and all-pass (Laguerre) filters [58] for their constant group delay property within their pass band and small area size. The constant group delay of low-pass and all-pass filters is important in many applications in which preserving a waveform is highly desired.

The transfer function of a first-order low-pass filter and group delay becomes

$$H_l(s) = \frac{\omega_o}{s+\omega_o}, \quad D_l(w) = \frac{\omega_o}{w^2+\omega_o^2}, \tag{28}$$

where ω_0 is the corner frequency, and for the first-order all-pass filter,

$$H_a(s) = \frac{s-\omega_o}{s+\omega_o}, \quad D_a(w) = \frac{2\omega_o}{w^2+\omega_o^2}. \tag{29}$$

Equations (28) and (29) show that both first-order low-pass and all-pass filters have constant group delays of $D_l(w) \approx 1/\omega_o$ and $D_a(w) \approx 2/\omega_o$, when $w \ll \omega_o$. The constant group delay property of low-pass and all-pass filters is desirable in many applications. One drawback of these delay filters is that their group delays are inversely proportional to the corner frequency [101], limiting the maximum group delay of the delay filters to be small for a wideband audio signal [101, 102]. For example, when the input signal is band limited by 8 KHz ($\omega_o > 8000$), then the maximum group delays of first-order low-pass and all-pass delay filters are limited to 0.125 msec and 0.25 msec, respectively.

Among the many available analog filters, the Bessel filter, or the Bessel-Thompson filter, is best known for its maximally flat group delay property. The Bessel filter is less known compared to other analog filters such as the Butterworth, Chebyshev, and Elliptic filters because the magnitude response of the Bessel filter is not superior to the others. The transfer function of the Bessel filter can be derived in many ways [5]. The first method is the "recursive fraction method," which is based on the fact that the output of the delay filter, $x_o(t)$, ideally should be the delayed version of the input signal, $x_i(t)$. When unity delay is considered, $x_o(t)$ becomes

$$x_o(t) = x_i(t-1). \tag{30}$$

Then, the transfer function of the filter is given by

$$H(s) = \frac{X_o(s)}{X_i(s)} = \frac{X_i(s)e^{-s}}{X_i(s)} = \frac{1}{e^s}. \tag{31}$$

More generally, $H(s)$ can be written as

$$H(s) = \frac{K}{Q(s)}, \tag{32}$$

where K is constant and denominator, $Q(s)$, satisfies

$$Q(s) = E(s) + O(s), \tag{33}$$

where $E(s)$ is an even polynomial and $O(s)$ is an odd polynomial. From Equation (31), $Q(s) = e^s$, and e^s can be expressed in terms of even and odd functions such as

$$e^s = \cosh(s) + \sinh(s). \tag{34}$$

Now, cosh(s) and sinh(s) can be approximated by the even and odd series

$$\cosh(s) = 1 + \frac{s^2}{2!} + \frac{s^4}{4!} + \cdots \quad \sinh(s) = s + \frac{s^3}{3!} + \frac{s^5}{5!} + \cdots . \tag{35}$$

The ratio of cosh(s) and sinh(s) yields

$$\frac{\cosh(s)}{\sinh(s)} = \frac{E(s)}{O(s)} = \frac{1}{s} + \cfrac{1}{\frac{3}{s} + \cfrac{1}{\frac{5}{s} + \cfrac{1}{\ddots \, \frac{2n-1}{s} + \cfrac{1}{\ddots}}}}. \tag{36}$$

The transfer function of the Bessel low-pass filter can be determined if we truncate Equation (36) to the required order and then assemble $Q(s)$ using Equation (33). Then we determine K so that the DC gain of the transfer function is normalized to unity. For instance, $H_1(s)$ and $H_2(s)$ become

$$H_1(s) = \frac{1}{s+1}, \tag{37}$$

$$H_2(s) = \frac{3}{s^2 + 3s + 3}, \tag{38}$$

for $n = 1$ and $n = 2$, respectively. The other method is to systematically evaluate $Q(s)$, which is also referred to as "Bessel polynomials," or "Thompson polynomials," using the following recursion:

$$Q_n(s) = (2n-1)Q_{n-1}(s) + s^2 Q_{n-2}(s), \tag{39}$$

where $Q_0(s) = 1$ and $Q_1(s) = s + 1$. Table 4 shows the Bessel polynomials, $Q_n(s)$, in unfactored form for $0 \leq n \leq 7$.

Figure 27 shows the group delays of fourth-order Butterworth, Chebyshev 1, Elliptic, and Bessel low-pass filters with corner frequency normalized to 1 Hz. The Elliptic filter, which is preferred among all the analog IIR filters because of its equi-ripple property in both the pass band and the stop band, shows the highest peak group delay around the corner frequency. It is obvious from Figure 27 that the Bessel low-pass filter generates a maximally flat group delay among these filters.

Table 4. Bessel polynomials, $Q_n(s)$, in unfactored form for $0 \leq n \leq 7$. The transfer function of the Bessel low-pass filter is given as $H(s) = K/Q_n(s)$ by choosing K so that the DC gain of the transfer function is normalized to unity.

Order n	Bessel polynomials $Q_n(s)$
0	1
1	$s + 1$
2	$s^2 + 3s + 3$
3	$s^3 + 6s^2 + 15s + 15$
4	$s^4 + 10s^3 + 45s^2 + 105s + 105$
5	$s^5 + 15s^4 + 105s^3 + 420s^2 + 945s + 945$
6	$s^6 + 21s^5 + 210s^4 + 1260s^3 + 4725s^2 + 10395s + 10395$
7	$s^7 + 28s^6 + 378s^5 + 3150s^4 + 17325s^3 + 62370s^2 + 135135s + 135135$

The general transfer function for the second-order band-pass filter is

$$H(s) = \frac{K(\frac{w_o}{Q})s}{s^2 + (\frac{w_o}{Q})s + w_o^2}, \tag{40}$$

where w_o is the center frequency, K is the gain, Q is the quality factor defined as w_o/BW, and BW is the bandwidth of the filter. Group delay, $D(w)$, of $H(s)$ becomes

$$D(w) = \frac{(\frac{w_o}{Q})({w_o}^2 + w^2)}{({w_o}^2 - w^2)^2 + (\frac{w_o}{Q})^2 w^2}. \tag{41}$$

Group delays of second-order band-pass filters for different Qs are shown in Figure 28. It is obvious from Equation (41) that the maximum delay of $D(w)$ is $2/BW$ at $w \approx w_o$. It should be noted that the maximum delay of the band-pass delay filter, $2/BW$, is independent of the center frequency and only the function of the bandwidth. As Q increases, the group delay around the corner frequency increases, approaching $2/BW$, and group delay at $w << w_0$ decreases.

4.2 Low-Pass-To-Band-Pass Transformation

The design procedure of analog filters such as high-pass, band-pass, band-stop, or all-pass filters generally begins with the design of a low-pass filter with the desired specifications and then transforms the low-pass filter into different filters.

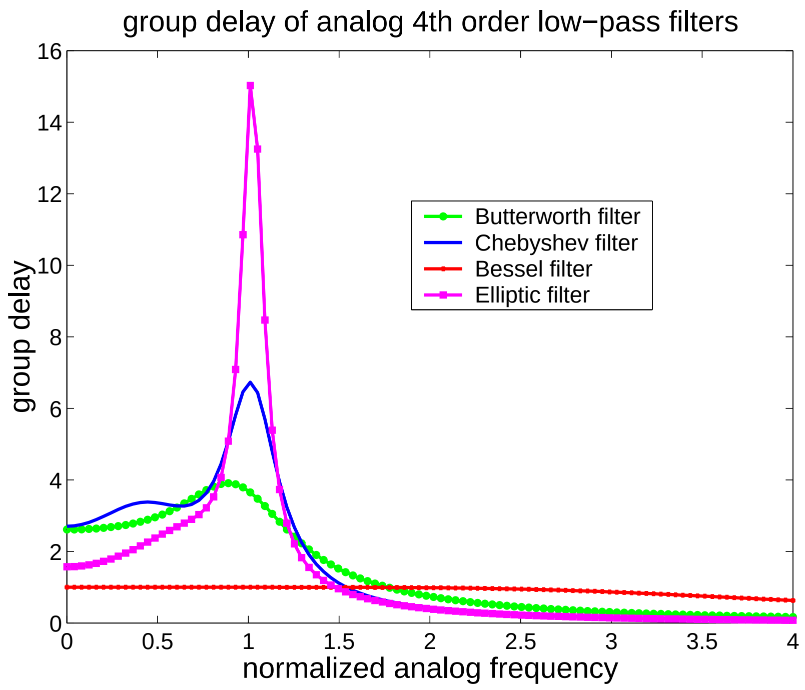


Figure 27. Group delay of analog filters. The filter order for all low-pass filters is 4, and the corner frequency is normalized to 1 Hz. The Elliptic filter has the highest group delay peak while the Bessel filter has almost a constant group delay.

4.2.1 Geometrically Symmetrical Transformation

A widely known normalized low-pass-to-band-pass transformation, referred to as "geometrically symmetrical transformation," is [25]

$$\tilde{\omega} = \frac{\omega}{\omega_H - \omega_L} - \frac{\omega_H \omega_L}{(\omega_H - \omega_L)\omega}, \tag{42}$$

where $\tilde{\omega}$ is the frequency variable of the original low-pass filter, and ω is the frequency variable of the band-pass filter. Equation (42) transforms low-pass frequencies $\tilde{\omega} = -1, 0, +1$ into band-pass frequencies $\omega = \omega_L,\ \sqrt{\omega_L \omega_H},\ \omega_H$. Transfer functions of the low-pass filter, $h(j\omega)$, and the band-pass filter, $H(j\tilde{\omega})$, are related by

$$h(j\omega) = H(j\tilde{\omega})|_{\tilde{\omega}=\tilde{\omega}(\omega)}, \tag{43}$$

where $\tilde{\omega}(\omega)$ is given by Equation (42) for the geometrically symmetrical transformation, and the phase responses are related by

$$\arg h(j\omega) = \arg H[j\tilde{\omega(\omega)}]. \tag{44}$$

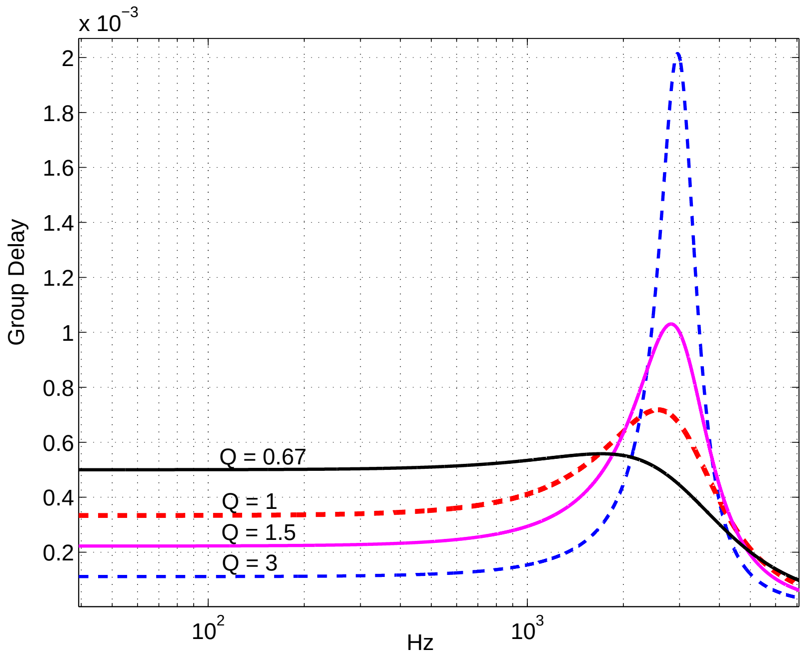


Figure 28. Group delay of the second-order band-pass filter for different Qs with $w_o = 3000$ Hz. When Q is low, the group delay of the band-pass filter has a constant group delay for a frequency lower than the lower cut-off frequency. As Q increases, the maximum group delay of the center frequency approaches $2/BW$, and group delay at the constant group delay region, where the frequency is smaller than the lower corner frequency, decreases.

When $D(\tilde{\omega})$ is the group delay function of the low-pass filter, and $d(\omega)$ is the group delay function of the transformed band-pass filter, it follows that

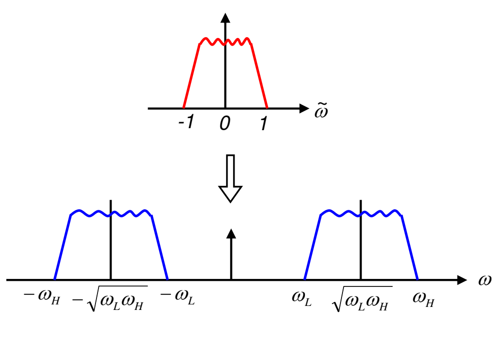
$$\begin{aligned} d(\omega) &= \frac{d}{d\omega} \arg h(j\omega) \\ &= \frac{d}{d\omega} \arg H[j\tilde{\omega}(\omega)] \\ &= \frac{d}{d\tilde{\omega}} \arg H(j\tilde{\omega}) \frac{d\tilde{\omega}}{d\omega} \\ &= D(\tilde{\omega}) \frac{d\tilde{\omega}}{d\omega}. \end{aligned} \tag{45}$$

Combining Equation (42) and Equation (45), the group delay of the band-pass filter, $d(\omega)$, becomes

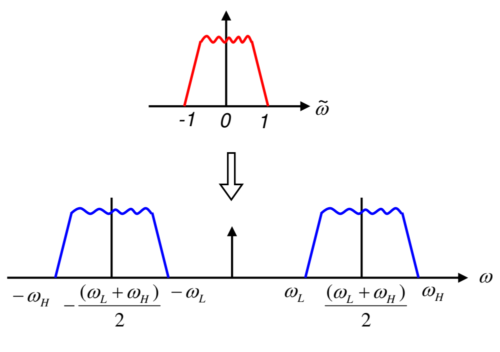
$$d(\omega) = \left[\frac{1}{\omega_H - \omega_L} \left(1 + \frac{\omega_L \omega_H}{\omega^2} \right) \right] D(\tilde{\omega}). \tag{46}$$

The term inside the brackets contributes to the distortion of the group delay of the band-pass filter. Even if $D(\tilde{\omega})$ is constant within its pass band, as is the case with the Bessel low-pass filter, $d(\omega)$ is not constant because of this distortion. For frequencies close to the center frequency ($\omega^2 \approx \omega_L \omega_H$), the distortion is quite small. However, as the frequency, w, deviates from the center frequency, $\sqrt{\omega_L \omega_H}$, significant group delay distortion is unavoidable. Figure 29(a) illustrates the geometrically symmetrical transformation from the low-pass variable, $\tilde{\omega}$, to the band-pass variable, ω. Since this transformation is geometrically symmetric, exhibiting symmetry on a logarithmic frequency scale, the linear-phase property of low-pass filters is no longer preserved in the transformed band-pass filter.

Figure 30 shows the distortions of the group delay of band-pass filters transformed from a normalized low-pass filter using the geometrically symmetrical transformation. The corner frequency of every low-pass filter is normalized to 1 Hz, and the center frequency, ω_o, and bandwidth, BW, of the band-pass filter are also normalized to 1 Hz. It can be shown in Figure 30(h) that the group delay of the Bessel band-pass filter, which is transformed using Equation (42), fails to maintain constant group delay property. It also can be noted from Figures. 30(e)–(g) that group delay of the Butterworth, Chebyshev 1, and Elliptic band-pass filters show an asymmetric group delay property at the vicinity of the normalized



(a)



(b)

Figure 29. Comparison of the geometrically symmetrical transformation and the arithmetically symmetrical transformation. (a) Low-pass frequencies $\tilde{\omega} = -1, 0, +1$ are transformed into band-pass frequencies $\omega = \omega_L, \sqrt{\omega_L\omega_H}, \omega_H$ or $\omega = -\omega_H, -\sqrt{\omega_L\omega_H}, -\omega_L$. Since this transformation is geometrically symmetric, the linear-phase property of low-pass filters is no longer preserved in the transformed band-pass filter. (b) Low-pass frequencies $\tilde{\omega} = -1, 0, +1$ are transformed to band-pass frequencies $\omega = \omega_L, (\omega_L + \omega_H)/2, \omega_H$ or $\omega = -\omega_H, -(\omega_L + \omega_H)/2, -\omega_L$. This transform doesn't distort the linearity of the group delay of the low-pass filter and also, if center frequency shift $\pm\omega_o$ is sufficiently large, then the group delay distortion affected by each components of Equation (51) will be negligible [11].

center frequency.

4.2.2 Arithmetically Symmetrical Transformation

Geffe [43] and Szentirmai [92] introduced another low-pass-to-band-pass filter transform, which is referred to as "arithmetically symmetrical transformation." Unlike the geometrically symmetrical transformation, this transform does not distort the linearity of the group delay, satisfying

$$\tilde{\omega} = \frac{2}{\omega_H - \omega_L}(\omega - \omega_o), \tag{47}$$

where $\tilde{\omega}$ is the frequency variable of the original low-pass filter, ω is the frequency variable of the band-pass filter, and the center frequency, ω_o, is equal to $(\omega_H + \omega_L)/2$. It transforms low-pass frequencies $\tilde{\omega} = -1, 0, +1$ to band-pass frequencies $\omega = \omega_L, (\omega_L + \omega_H)/2, \omega_H$. Since Equation (47) only yields the pass-band region $\omega_L \le \omega_o \le \omega_H$, a transformation that generates a negative frequency pass band should also be introduced [25]. The complete arithmetically symmetrical transformation using both transfer functions can be written as

$$h(j\omega) = H(j\tilde{\omega}_+)H(j\tilde{\omega}_-), \tag{48}$$

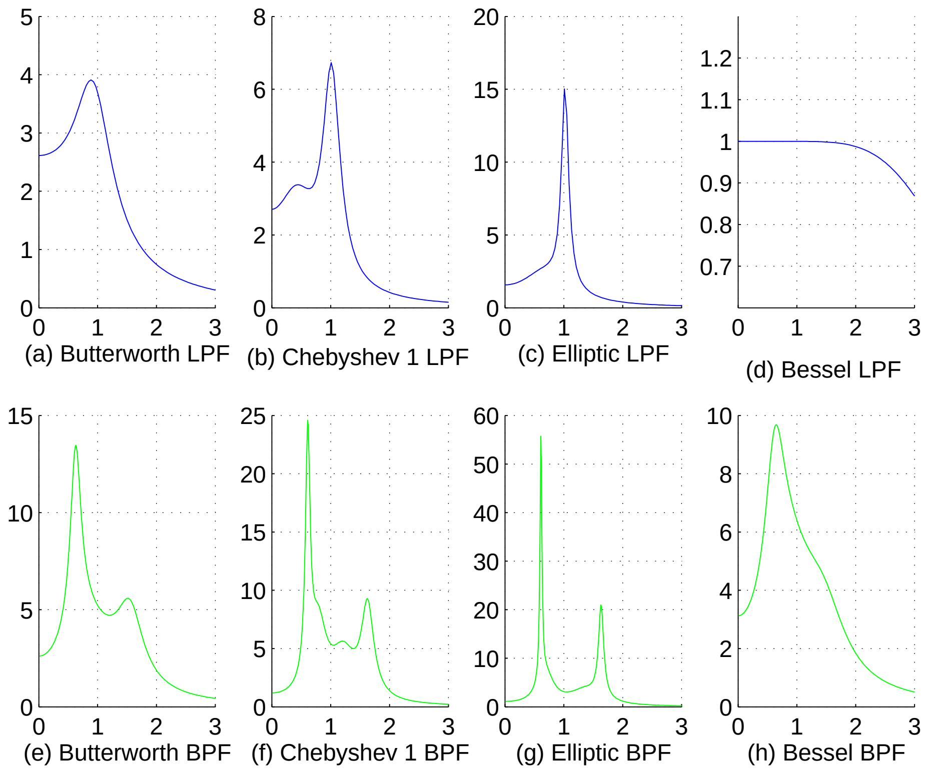


Figure 30. Simulation results of the group delay of the band-pass filter transformed from a normalized low-pass filter using the geometrically symmetrical transformation (Equation (42)). The filter orders of the low-pass and band-pass filters are 4 and 8, respectively.

where

$$\tilde{\omega}_+ = 2(\omega - \omega_o)/(\omega_H - \omega_L), \tag{49}$$

$$\tilde{\omega}_- = 2(\omega + \omega_o)/(\omega_H - \omega_L). \tag{50}$$

The group delay of the band-pass filter, $d(\omega)$, is

$$\begin{aligned} d(\omega) &= \frac{2}{\omega_H - \omega_L}\{D(\tilde{\omega}_+) + D(\tilde{\omega}_-)\} \\ &= \frac{2}{\omega_H - \omega_L}\Big(D\Big[\frac{2(\omega - \omega_o)}{\omega_H - \omega_L}\Big] + D\Big[\frac{2(\omega + \omega_o)}{\omega_H - \omega_L}\Big]\Big), \end{aligned} \tag{51}$$

where $D(\cdot)$ is the group delay function of the normalized low-pass filter. If the center frequency shift $\pm\omega_o$ is sufficiently large, then the influence between the two $D[\cdot]$s of Equation (51) can be neglected [11]. Figure 31 shows the simulation results for the group delay of band-pass filters transformed using the arithmetically symmetrical transformation. The results show that the group delay maintains symmetry and that the constant group delay property of the Bessel low-pass filter is well preserved over a large bandwidth, compared to that shown in Figure 30.

Figure 32 shows the group delay of the third-order Bessel low-pass and the sixth-order Bessel band-pass filter. The center frequency of the band-pass filter is 2 KHz, and the bandwidth is 1 KHz. The solid curve in Figure 32(b) is the group delay of the band-pass filter transformed using the arithmetically symmetrical transformation, whereas the dotted curve is the group delay of the band-pass filter transformed using the geometrically symmetrical transformation. This clearly shows that the arithmetically symmetrical transformation doesn't distort the linearity of the group delay of the Bessel low-pass filter. It also should be noted that the difference between the two transformations is noticeable when the filter order of the band-pass filter is no less than third order.

4.3 Subband Delay Structure

We have seen the group delay properties of general low-pass and band-pass filters, the group delay of the Bessel filter, and two low-pass-to-band-pass transformations. Based

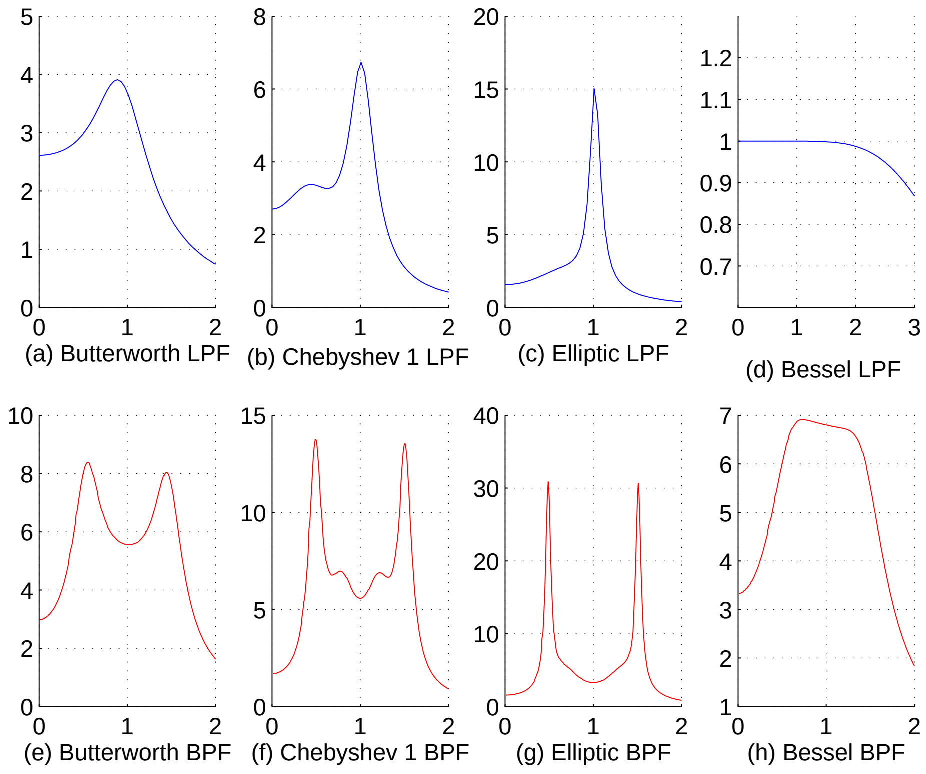


Figure 31. Simulation results of group delay of the band-pass filter transformed from a normalized low-pass filter using the arithmetically symmetrical transformation. The filter orders of the low-pass and band-pass filters are 4 **and** 8**, respectively.**

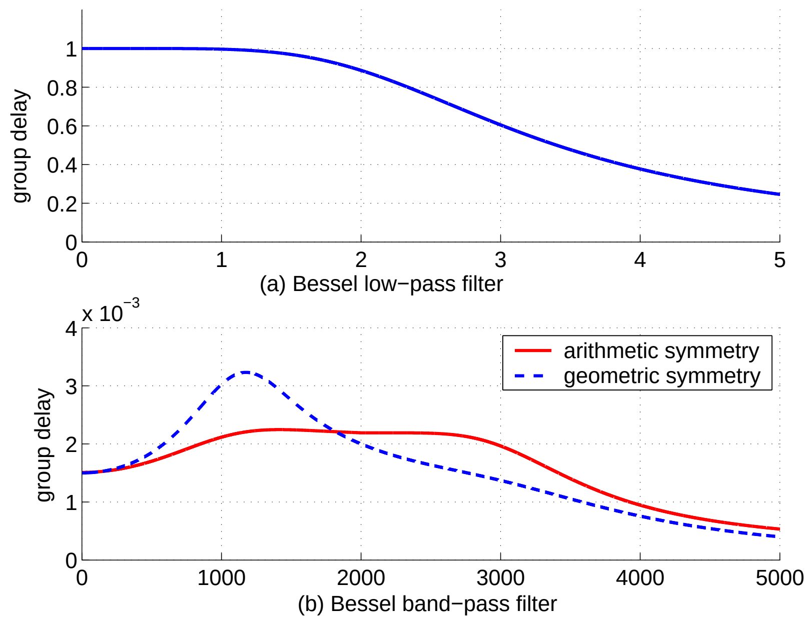


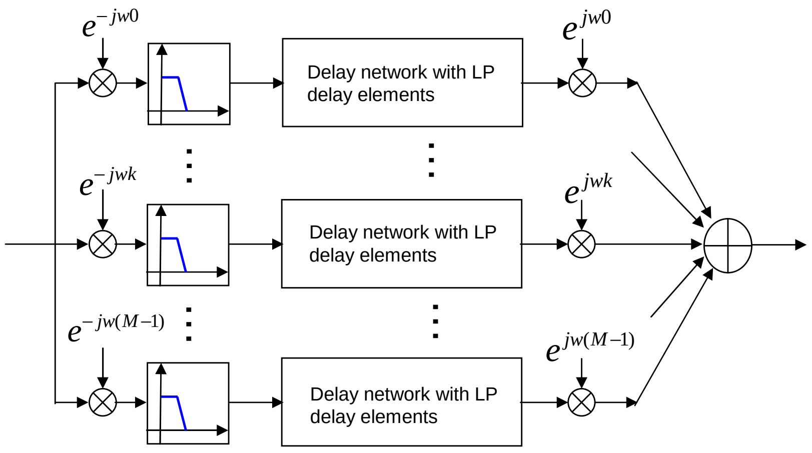
Figure 32. Group delay of the Bessel low-pass and band-pass filters. The filter orders of the low-pass and band-pass filters are 3 **and** 6**, respectively. The corner frequency of the low-pass filter is normalized to** 1 **Hz, and the center frequency and the bandwidth of the band-pass filter is de-normalized to** 2 **KHz and** 1 **KHz. The arithmetically symmetrical transformation doesn't distort the linearity of the group delay of the Bessel low-pass filter.**

on these information, two different delay structures for band-limited input signals are presented in this section. The former is best suited for applications in which the linearity of the group delay is very critical, while the latter is for applications in which the maximum group delay of a delay element is even more important than the linearity of the group delay [101].

4.3.1 Delay Network with Low-Pass Filters and Modulation

The first delay network consists of low-pass delay elements, shown in Figure 33. This structure can be applicable to situations in which the linearity of the group delay is more crucial than the maximum group delay that the delay network can create. Figure 33 is very similar to a complex modulator model of the DFT filter bank structure in a digital domain except it is for continuous-time implementation, in this case the sampling rate change is unnecessary.

In the beginning of each subband, the band-limited signal is modulated to a baseband by multiplying a complex number to the band-limited signal. Modulation to a baseband is

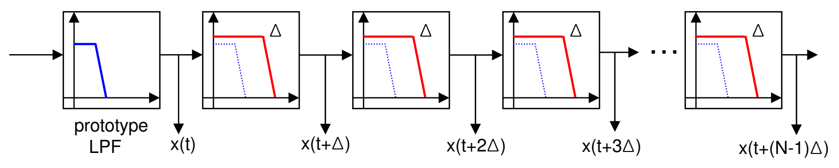


e^{-jw0}
e^{-jwk}
$e^{-jw(M-1)}$
Delay network with LP delay elements
Delay network with LP delay elements
Delay network with LP delay elements
e^{jw0}
e^{jwk}
$e^{jw(M-1)}$

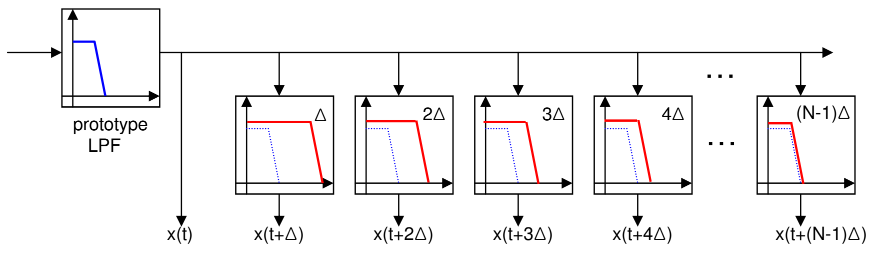
Figure 33. Delay network with low-pass delay filters. This structure can be applicable to situations where the linearity of the group delay is more crucial than the maximum group delay that the delay network can create. The delay element in each subband can be implemented with the low-pass filter because each band is multiplied by a complex number to modulate the band-limited signal into a base band. Modulation to a baseband is advantageous in many ways. First, it enables a delay network to create a larger group delay by decreasing the highest frequency of the band-limited input signal to a low-frequency baseband. Second, it allows the delay network to have the same time-constant (corner frequency) distribution of delay elements throughout different subbands, simplifying programming complexity of the corner frequency.

advantageous in many ways. First, it enables a delay network to create a larger group delay by decreasing the highest frequency of the band-limited input signal to a low-frequency baseband. As shown in Equation (28), the group delay of the low-pass filter is inversely proportional to the corner frequency. In fact, the lowest corner frequency of the low-pass delay filter can't be smaller than the highest frequency of the band-limited input signal to avoid the spectral magnitude of the input signal. This can be a serious problem for higher-frequency subbands that have to create the same amount of group delay as lower-frequency subbands, because the dynamic range of the corner frequency of higher subbands is much smaller than that of lower subbands. Second, it allows the delay network to have the same time-constant (corner frequency) distribution of delay elements throughout different subbands, simplifying the programming complexity of the corner frequency.

Figure 34 shows two examples of delay lines for each subband. The Bessel low-pass filter is best suited for each delay filter because of its maximally flat group delay, explained in Sec. 4.1. Figure 34(a) shows the tapped-delay line, which consists of low-pass delay



(a)



(b)

Figure 34. Two examples of delay lines for a delay network with low-pass delay filters. (a) Tapped-delay line consisting of low-pass delay filters with a uniform corner frequency distribution. Each delay element generates the same delay and the number of cascades can be significantly restricted by the amount of accumulated noise at each delay filter. (b) Parallel-delay line consisting of low-pass delay filters with a non-uniform corner frequency distribution. Each delay element is programmed to have a larger bandwidth for a smaller delay or a smaller bandwidth for a larger delay.

filters with a uniform corner frequency distribution. Each delay element generates the same delay and the number of cascades can be significantly restricted by the amount of accumulated noise at each delay filter. Figure 34(b) shows the parallel-delay line, which consists of low-pass delay filters with a non-uniform corner frequency distribution. Each delay element is programmed to have a larger bandwidth for a smaller delay or a smaller bandwidth for a larger delay. The accumulated noise problem can be mitigated with this structure, but the corner frequency mismatch between the other subbands can limit the performance of this structure.

4.3.2 Delay Network with Band-Pass Filters

A delay network with band-pass delay elements can be used for applications in which creating a larger group delay out of a delay network is more important. Unlike the low-pass

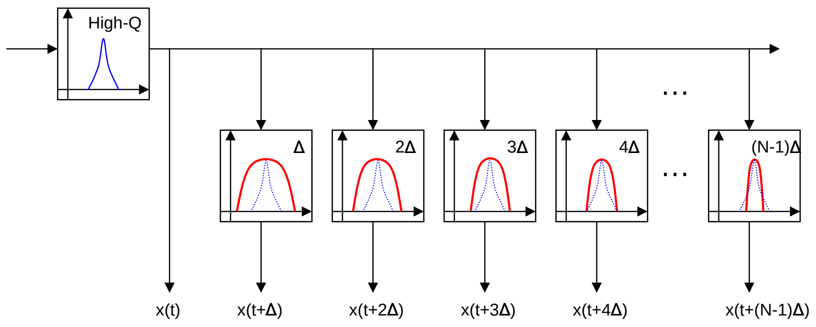


Figure 35. Delay network with the band-pass delay filters for a single subband. A high-*Q* band-pass filter can be used as an element of the analysis filter bank to increase frequency selectivity and thus increase the maximum group delay of the delay filter. The bandwidth of the delay element should be decreased to increase the group delay.

delay element, the constant group delay of the band-pass delay filter within its pass band can be obtained only when the band-pass delay filters are transformed using the arithmetically symmetrical transformation from Bessel low-pass filters, explained in Sec. 4.2.2.

Figure 35 shows the delay network with band-pass delay elements. A parallel-delay line is adopted instead of a tapped-delay line since the band-pass delay element is known to be less robust to the accumulated noise than the low-pass delay element. A wideband input signal is transformed into subband signals through the analysis filter bank at the beginning of each subband. High-*Q* band-pass filters can be used as the analysis filter bank to increase frequency selectivity and thus increase the maximum group delay of the delay filter. However, band-pass delay elements don't need to be high-*Q* filters as long as their bandwidth is wider than that of the band-pass filter used in the analysis filter bank. Figure 35 shows that the bandwidth of the delay element is programmed to be narrower for larger group delay.

Figure 36 presents another example of a delay network that can be used for applications in which the maximum group delay out of a delay element is not large enough to avoid the cascade at all, which occurs when the filter order of the delay filter is rather small or in applications such as AEC, which require an extremely long delay line. The first row in Figure 36 is designed to create a larger delay by programming a smaller bandwidth while

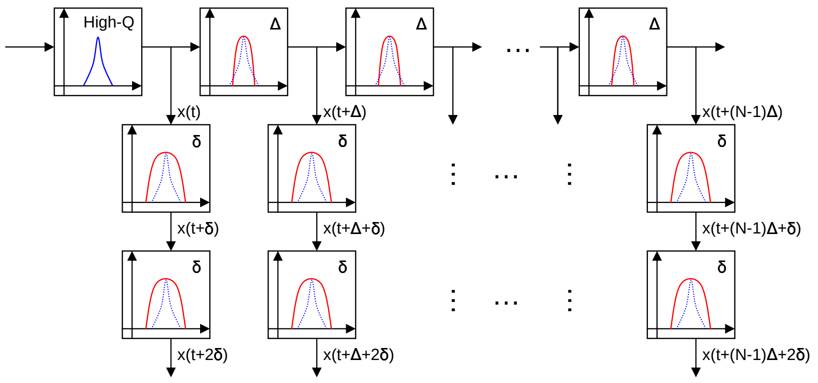


Figure 36. Multi-level delay network with the band-pass delay filters for a single subband. The first row is designed to create a larger delay by programming a smaller bandwidth while the remaining rows are designed to provide a finer delay resolution smaller than the delay of the first row.

the ramaining rows are designed to provide a finer delay resolution smaller than the delay of the first row. Other variants of Figure 36 can be made by allowing a different number of rows for different columns of delay elements in the first row. Since a typical room impulse response carries most of its energy at the first decaying part rather than at the long reverberation part, more depth at the beginning columns and less depth at the latter columns can be advantageous for continuous-time adaptive filter implementation.

CHAPTER 5

HYBRID DA FIR FILTER

In this chapter, a new DA architecture based on an alternative implementation of an LUT is presented. We refer to this type of DA as “hybrid architecture” since it can use both an LUT and adders simultaneously in contrast to LUT-based or adder-based DA. The hybrid architecture is originally based on the LUT size reduction technique for DA-OBC [20], and we further develop new memory reduction technique for the original DA [99].

In the original LUT-based DA, LUT size is fixed to 2^k for a k-tap DA base unit or 2^{k-1} for a k-tap DA-OBC base unit. The hybrid architecture, however, provides more flexibility than the LUT-based or adder-based DA by allowing the arbitrary selection of LUT size. In hybrid architectures, the LUT size of a k-tap DA base unit can vary from as low as 0 to 2^k, and the LUT size of a k-tap DA-OBC base unit can vary from as low as 0 to 2^{k-1}. It should be noted that LUT-based DA and adder-based DA are only two extreme cases of hybrid DA.

5.1 Hybrid DA-OBC Architecture

One memory reduction technique for DA-OBC was proposed in [20]. The memory size of DA-OBC can decrease exponentially at the expense of linearly increased control logic. The memory reduced DA-OBC architecture, however, is still not suitable for high-order FIR filter implementation based on the partial sum technique, since the adder/shifter unit is directly controlled by filter coefficients and input samples. To address this problem, I propose “hybrid DA-OBC architecture” for the area-efficient implementation of high-order FIR filters. The hybrid DA-OBC architecture features the standard adder/shifter unit of DA, which is not controlled by sum of the filter coefficients and input samples.

The memory reduction technique for DA-OBC can be best understood with examples. Figure 11 is a block diagram of a 4-tap ($K = 4$) original DA-OBC FIR filter. The LUT size of DA-OBC can decrease to 2^{K-1} at the additional K XORs and a 2x1 MUX using the mirrored property of Table 3. Further memory reduction technique proposed in [20] depends on following two steps:

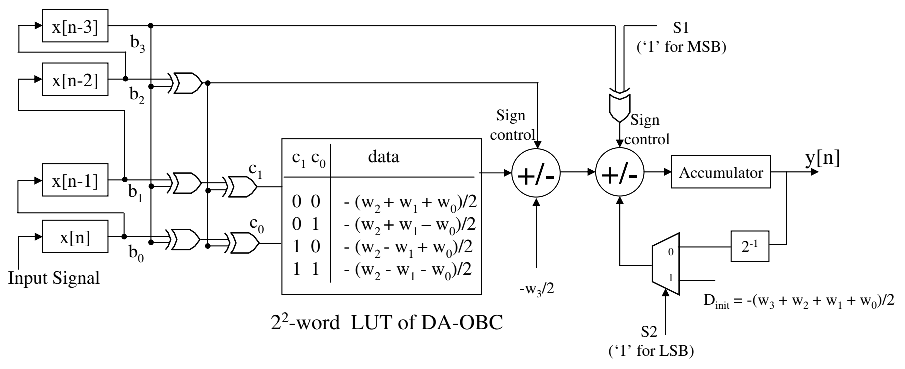
1. Memory reduction by a factor of 2.

2. Control logic (XOR) optimization.

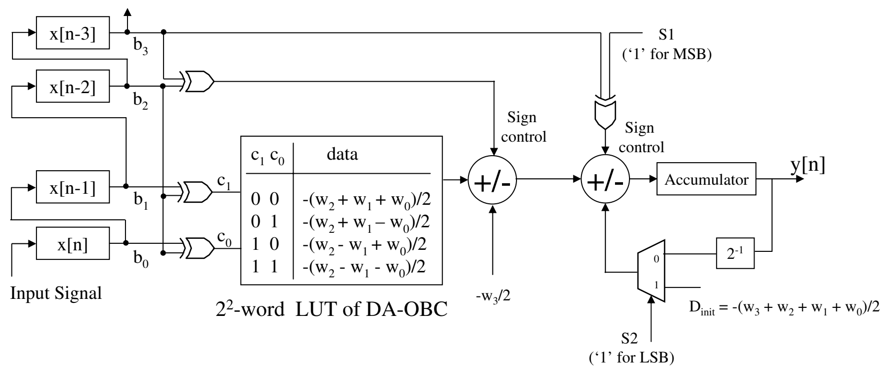
The first step is based on the observation that if $-w_3/2$ term can be located outside the LUT, then the lower half of the LUT is a mirror image of the upper half of the LUT with the signs reversed. Thus, if we can move $-w_3/2$ term to the outside of the LUT, only half of the LUT of DA-OBC can be used at the cost of additional small control logic. Figure 37(a) shows the new DA-OBC architecture. The LUT size is reduced by a factor of 2 over the original DA-OBC with additional control logic such as XORs, a full adder with subtraction, and a register for storing $-w_3/2$. The second step is to reduce the increased number of XORs using Boolean algebra simplification. After the second step, the total number of XORs inside the base unit is reduced to K. The final DA-OBC architecture for a 4-tap FIR filter with 2^2 size LUT is shown in Figure 37(b).

The memory reduction technique can also be recursively applied to the system shown in Figure 37(b). Figure 38(a) shows the system after the first step, and Figure 38(b) shows the system after the second step. Finally, one more memory reduction of Figure 38(b) results in LUT-less DA-OBC architecture. Figure 39 illustrates the block diagram of the LUT-less DA-OBC for a 4-tap FIR filter.

Figure 39, however, poses several problems for high-order FIR filters. As explained in the previous section, DA architecture can't be used for high-order filters without the partial sum technique because of the exponentially increasing memory size. To be area efficient for high-order filters, DA architecture, as explained in Sec. 2.3, should have a global shift register unit, a global adder/shifter unit, m number of k-tap base units, and an adder tree unit consisting of $m-1$ adders. The adder/shifter unit of Figure 39, however, can't make a

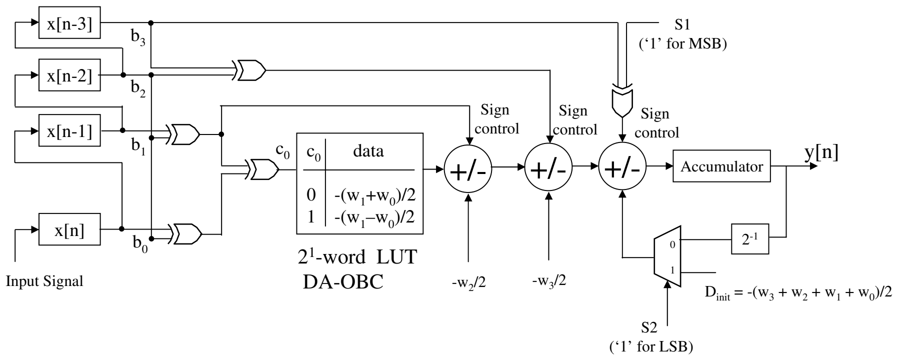


(a)

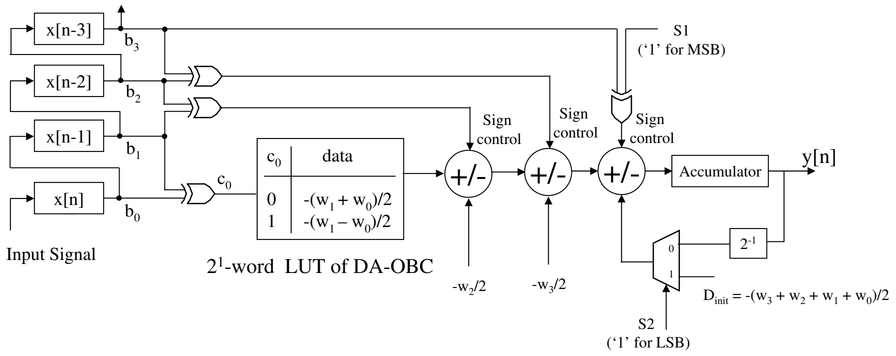


(b)

Figure 37. Block diagram of a 4-tap DA-OBC FIR filter with 2^2 size LUT. The LUT size is reduced by a factor of 2 over the original DA-OBC. (a) DA-OBC architecture for a 4-tap FIR filter with 2^2 size LUT after the first step of the memory reduction technique. (b) The overall additional control logics are simplified to a full adder with subtraction and a register for storing $-w_3/2$.



(a)



(b)

Figure 38. Block diagram of a 4-tap DA-OBC FIR filter with 2^1 size LUT. The LUT size is reduced by a factor of 2 over the original DA-OBC. (a) DA-OBC architecture for a 4-tap FIR filter with 2^1 size LUT after the first step of the memory reduction technique. (b) The overall additional control logics are simplified to a full adder with subtraction and a register for storing $-w_2/2$.

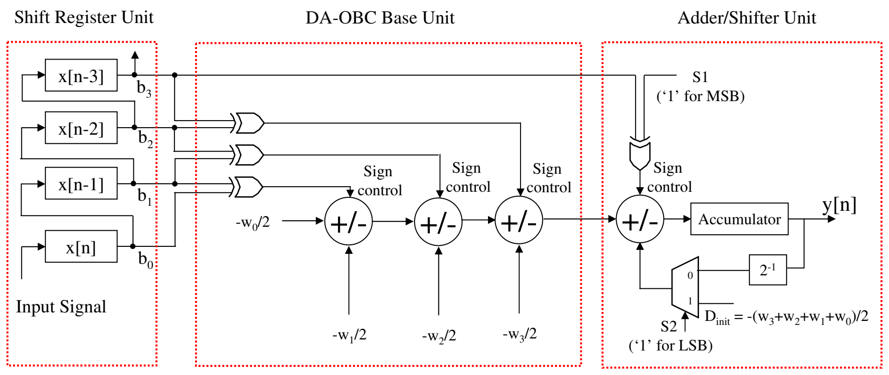


Figure 39. Block diagram of the LUT-less DA-OBC for a 4-tap FIR filter. The adder/shifter unit is controlled by the input sample, $x[n-K]$, and the filter coefficients, D_{init}.

global adder/shifter unit for the following two reasons.

- The adder in the adder/shifter unit is controlled by the bit streams from $x[n-K]$.
- The adder in the adder/shifter unit is multiplexed with D_{init}, which is the function of the filter coefficients.

The proposed architecture, shown in Figure 40, prevent the input sample, $x[n-K]$, and the filter coefficients, D_{init}, from controlling the adder/shifter unit. This architecture can be derived from the following three steps:

1. The sign control of the adder located inside the adder/shifter unit is disconnected from the input samples by adding an inverter and a 2x1 MUX inside the base unit. The select line of the 2x1 MUX is connected to the input sample, $x[n-K]$, coming from the shift register unit. The 2x1 MUX selects the positive/negative sums of $K-1$ adders located inside the base unit, depending on the 0/1 values of the $x[n-K]$ bitstream.

2. D_{init} and 2x1 MUX are moved into the base unit with an additional full adder. The select line of this 2x1 MUX is unchanged ("1" for LSB and "0" for others) but it selects $0/D_{\text{init}}$, depending on the 0/1 values of the select line.

3. The 2x1 MUX controlled by $x[n-K]$, an inverter, and a full adder are replaced by a full adder with a subtraction selector.

The overall cost for converting the LUT-less DA-OBC architecture into the architecture shown in Figure 40 is a single full adder with a subtraction selector. By following the same three steps, other DA-OBC architectures shown in this section can be transformed into hybrid architectures, which we refer to as "hybrid DA-OBC," at the cost of a single full adder with a subtraction selector. Although the base unit of hybrid DA-OBC architecture requires one more full adder with a subtraction selector than the base unit of its counterpart DA-OBC, it is much more area efficient for high-order FIR filters due to the

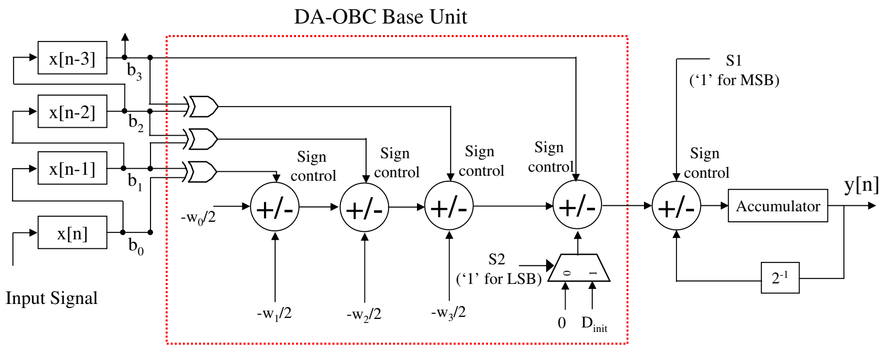


Figure 40. Block diagram of the LUT-less hybrid DA-OBC for a 4-tap FIR filter. The adder/shifter unit has a standard structure in which the control of the adder/shifter unit is independent of the input sample and filter coefficients.

global adder/shifter unit. In addition, adders used inside the hybrid DA-OBC base unit can be configured to have a tree architecture instead of a linear connection to reduce the critical data path delay, shown in Figure 40.

5.2 Hybrid DA Architecture

The hybrid DA architecture presented here is based on the memory reduction technique of DA, which minimizes the redundancy of LUT contents of DA architecture. The memory reduction technique for DA can also be best understood with examples. The LUT of the original LUT-based K-tap DA architecture has the following property:

$$\mathrm{LUT}(1, b_{K-2}, \cdots, b_1, b_0) = \mathrm{LUT}(0, b_{K-2}, \cdots, b_1, b_0) + w_{K-1}, \tag{52}$$

where $\mathrm{LUT}(b_{K-1}, b_{K-2}, \cdots, b_1, b_0)$ is the LUT content addressed by $b_{K-1}, b_{K-2}, \cdots, b_1, b_0$. Figure 9 shows that the lower half of LUT (locations whose addresses have a 1 in the MSB) is equal to the upper half of LUT (locations whose addresses have a 0 in the MSB) plus the w_3 term. Hence, LUT size can be reduced by a factor of 2 with an additional 2x1 MUX and a full adder, as shown in Figure 41. The 2x1 MUX selects $0/w_3$, depending on 0/1 values of $x[n-K]$. Inspection of the LUT of Figure 41 shows that the symmetry property of Equation (52) still holds, enabling further memory reduction. Figure 42 shows various

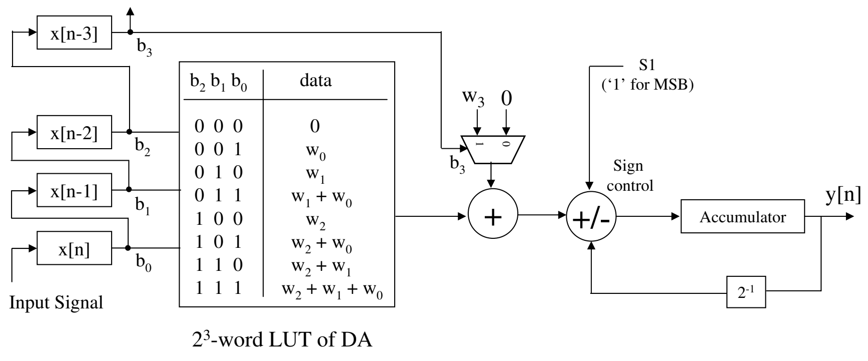


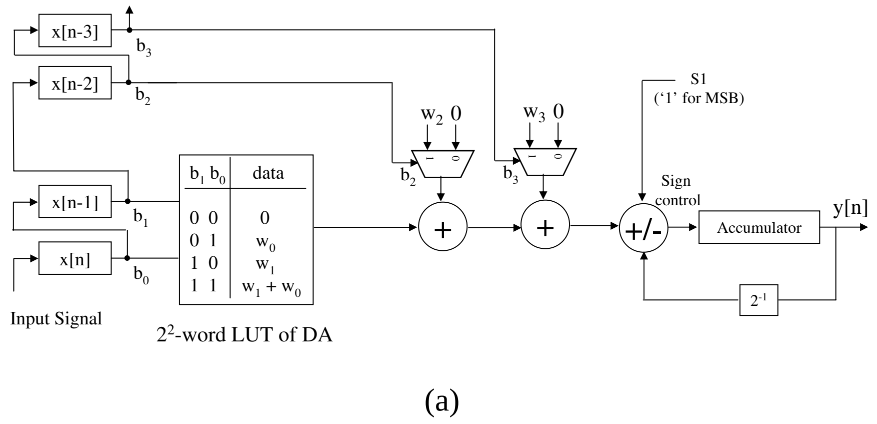
Figure 41. Hybrid DA architecture for a 4-tap FIR filter with 2^3 size LUT. LUT size was reduced by a factor of 2 with an additional 2x1 MUX and a full adder using the observation that the lower half of LUT (locations whose addresses have a 1 in the MSB) of Figure 9 is equal to the upper half of LUT (locations whose addresses have a 0 in the MSB) plus the w_3 term.

hybrid DA architectures with 2^2 size LUT and 2^1 size LUT. After several iterations of the memory reduction, one can finally obtain the LUT-less hybrid DA architecture shown in Figure 43.

5.3 Hardware Cost Analysis

Hardware complexity for LUT-based DA, DA-OBC, hybrid DA-OBC, and hybrid DA architectures are compared in terms of a transistor count estimate and FPGA resource utilization. Hybrid DA-OBC and hybrid DA architectures generally describe DA-OBC and DA architectures with various LUT sizes including the LUT-less architecture. For simplicity, we consider only the LUT-less hybrid DA-OBC and the LUT-less hybrid DA architectures in this section. In addition, only base units are considered for transistor count comparison since both the shift register unit and the adder/shifter unit are common for all four architectures.

Estimating transistor count is a frequently used technique for comparing custom VLSI chip size among different architectures. Digital logic functions, however, can be implemented in many different ways, depending on their targeting design constraints such as silicon area, power consumption, and speed. For example, XOR logic can be implemented



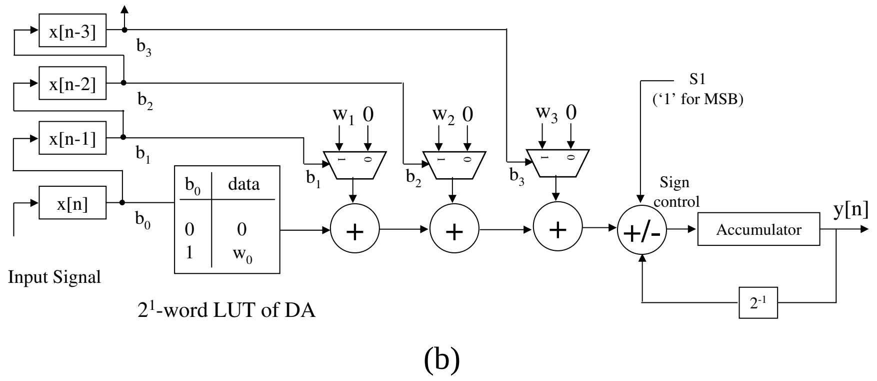


Figure 42. (a) Hybrid DA architecture for a 4-tap FIR filter with 2^2 size LUT. (b) Hybrid DA architecture for a 4-tap FIR filter with 2^1 size LUT.

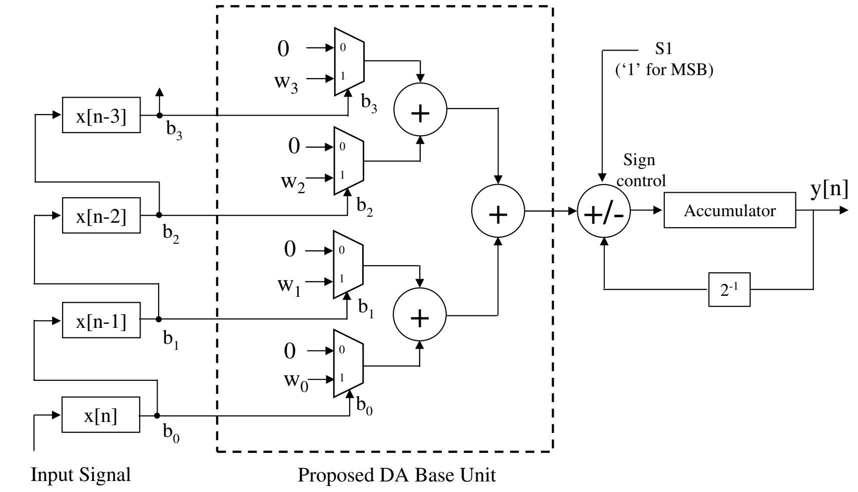


Figure 43. LUT-less hybrid DA architecture for a 4-tap FIR filter. Adders inside the base unit are configured to have a tree architecture instead of a linear connection, to reduce the critical data path delay.

Table 5. Transistor counts for digital logic functions. Estimating transistor count is a frequently used technique for comparing custom VLSI chip size among different architectures. Digital logic functions, however, can be implemented in many different ways, depending on their targeting design constraints such as silicon area, power consumption, and speed. Transistor count is estimated based on the static CMOS implementation of digital logic or the assumptions made in [20].

Logic		Transistor count
INV (1 bit)		2
XOR (1 bit)		8
2x1 MUX (1 bit)		6
Adder (fixed C_{in})	$C_{in} = 0$ ($B_c/2$ bit)	$12(B_c/2)$
	$C_{in} = 1$ ($B_c/2$ bit)	$14(B_c/2)$
Adder (B_c bit)		$30B_c$
Adder/Subtractor (fixed C_{in})	$C_{in} = 0$ ($B_c/2$ bit)	$(12 + 8)(B_c/2)$
	$C_{in} = 1$ ($B_c/2$ bit)	$(14 + 8)(B_c/2)$
Adder/Subtractor (B_c bit)		$30B_c + 8B_c$
$2^k \times B_c$ (ROM)	Decoder	$C(1, k)$
	Data	$D(1, k, B_c)$
Register (1 bit)		16

with four 2x1 NAND gates, which are equivalent to 16 transistors, or with full static CMOS logics, which are equivalent to 8 transistors [59, 79]. Table 5 lists the transistor count assumed in the analysis for each logic function. The transistor count estimate in Table 5 is based on static CMOS implementation of digital logic or assumptions made in [20]. Cost functions in Table 5 are defined as

$$C(a,b) = 4 \cdot \left(2 \sum_{i=a}^{b-1} (b-i) + 2^{b-a+1}\right) \tag{53}$$

$$D(a,b,c) = 2^{b-a+1} \times c. \tag{54}$$

When one of the three inputs of a full adder has a fixed value (we assume the fixed input is carry, C_{in}, for convenience), 12 and 14 transistors can be used for $C_{in} = 0$ and $C_{in} = 1$, respectively, rather 30 transistors for a regular full adder [79]. The occurrences of $C_{in} = 0$ and $C_{in} = 1$ are statistically assumed to be the same with $B_c/2$ bits for $C_{in} = 0$ and the remaining $B_c/2$ bits for $C_{in} = 1$. The adder with a subtraction selector requires 8 more transistors for an extra 2x1 MUX and an inverter in addition to the original full adder [79].

When the transistor count for various DA architectures are estimated, two versions–the

Table 6. Transistor count comparison of various base units for the k-tap FIR filter. The filter coefficients of the hybrid DA-OBC and hybrid DA architectures are stored in registers to allow the programming of filter coefficients prior to the filtering operation.

Logic functions	LUT-based DA	DA-OBC	LUT-less Hybrid DA-OBC (register version)	LUT-less Hybrid DA (register version)
ROM decoder	$C(1,k)$	$C(2,k)$	0	0
ROM data	$D(1,k,B_c)$	$D(2,k,B_c)$	0	0
XOR	0	$8k$	$8(k-1)$	0
2x1 MUX	0	$6B_c$	$6B_c$	$6k \times B_c$
Register	0	$16B_c$	$80B_c$	$16k \times (B_c - \lceil log_2 k \rceil)$
Adder	0	0	0	$(k-1) \times 30B_c$
Adder/Sub. $C_{in} = 0$	0	0	0	0
Adder/Sub. $C_{in} = 1$	0	0	0	0
Adder/Sub.	0	0	$k \times 38B_c$	0

register version and the hardwired coefficient version– are considered, depending on how filter coefficients are implemented. These two versions only differ in the hybrid DA-OBC and the hybrid DA architectures since the original LUT-based DA and DA-OBC store their coefficients only inside the LUT. In the register version, each coefficients are assumed to be stored in explicitly created registers. This case assumes manual load and clear operations of filter coefficients stored in the registers. This version is suitable for general FIR filtering processors whose coefficients can be programmed prior to the actual filtering operation. Table 6 shows the transistor count comparison of various base units of the k-tap FIR filter for the register version.

Figure 44 compares the silicon area of hybrid architectures, such as the hybrid DA-OBC and the hybrid DA, whose filter coefficients are stored in registers, with the original DA and DA-OBC in terms of transistor count when $B_c = 18$. Figure 44 is re-created from Table 6 and thus only considers base unit. It is obvious from Figure 44 that hybrid architectures require a smaller area than LUT-based DA and DA-OBC for high-order filter

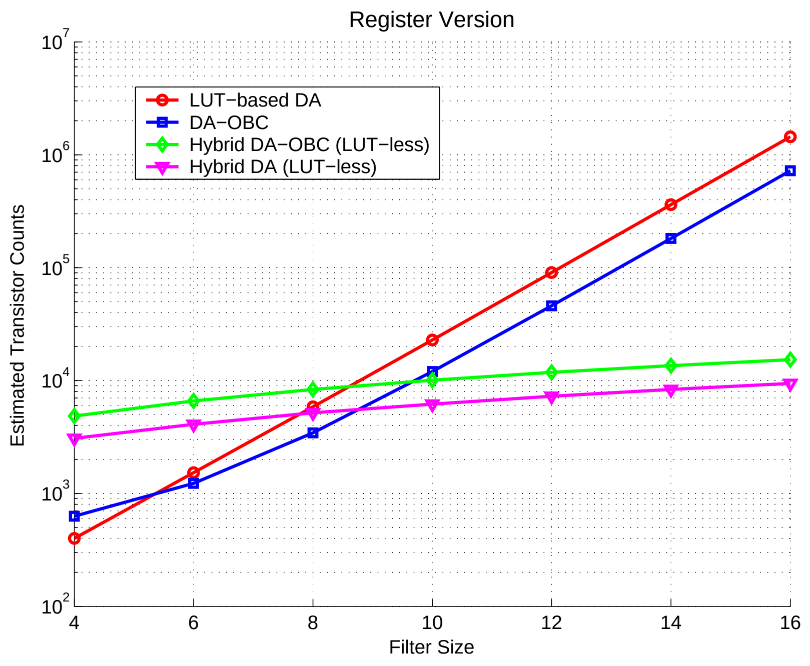


Figure 44. Transistor count estimation comparison of various base units for different filter sizes with $B_c = 18$. Hybrid architectures require a smaller area than LUT-based DA and DA-OBC for high-order filter implementation; in particular, the hybrid DA architecture requires the smallest transistor counts among the four different base units throughout all filter sizes.

implementation. Originally, hybrid architectures are derived to reduce memory usage exponentially at a cost of linearly increased control logic complexity, but Figure 44 shows that hybrid architectures are not only memory efficient but also area efficient for high-order filters. Furthermore, it should be noted that the hybrid DA architecture requires the smallest transistor counts among the four different base units throughout all filter sizes.

The other version is the hardwired version, in which filter coefficients of the hybrid DA-OBC and the hybrid DA are assumed to be hardwired to either VCC or GND. The disadvantage of the hardwired version is a lack of flexibility and programmability. Once the VLSI chip is fabricated, the filter coefficients can't be modified. The hardwired version, however, is smaller than the register version since it is maximally area optimized for pre-chosen specific coefficients. The hardwired version is useful for well-known transform operations such as DCT, inverse DCT (IDCT), DFT, or inverse DFT (IDFT) processors. These processors have been extensively investigated for VLSI implementation [14, 15, 103]

Table 7. Transistor count comparison of various base units for the k-tap FIR filter. The filter coefficients of the hybrid DA-OBC and hybrid DA architectures are hardwired, assuming they are not changed. Due to this assumption, the original adder can be replaced with the smaller adder with fixed carry, C_{in}.

Logic functions	LUT-based DA	DA-OBC	LUT-less Hybrid DA-OBC (hardwired version)	LUT-less Hybrid DA (hardwired version)
ROM decoder	$C(1, k)$	$C(2, k)$	0	0
ROM data	$D(1, k, B_c)$	$D(2, k, B_c)$	0	0
XOR	0	$8k$	$8(k-1)$	0
2x1 MUX	0	$6B_c$	$6B_c$	$6k \times B_c$
Register	0	$16B_c$	$16B_c$	0
Adder	0	0	0	$(k-1) \times 30B_c$
Adder/Sub. $C_{in} = 0$	0	0	$(k-1) \times 10B_c$	0
Adder/Sub. $C_{in} = 1$	0	0	$(k-1) \times 11B_c$	0
Adder/Sub.	0	0	$38B_c$	0

and their coefficients do not need to be updated once the size and type of transform are determined. Table 7 shows the transistor count comparison of various base units of the k-tap FIR filter for the hardwired version.

Figure 45 compares the silicon area of hybrid architectures, whose filter coefficients are hardwired, with the original DA and DA-OBC architectures in terms of transistor count when $B_c = 18$. As seen in Figure 44, it is obvious from Figure 45 that hybrid architectures require a smaller area than LUT-based DA and DA-OBC architectures for high-order filters, and especially the hybrid DA-OBC requires the smallest transistor counts among the four different base units for high-order filters because it can be implemented with fixed-carry adders, which are much smaller than regular adders.

5.4 FPGA Implementation

To illustrate the merits of hybrid architectures, LUT-based DA, the hybrid DA-OBC, and the hybrid DA are physically implemented on an Altera Stratix EP1S80F1508C6 FPGA chip. Implementation results of the FPGA synthesis are provided by the compilation reports of Altera's Quartus II 4.0 software.

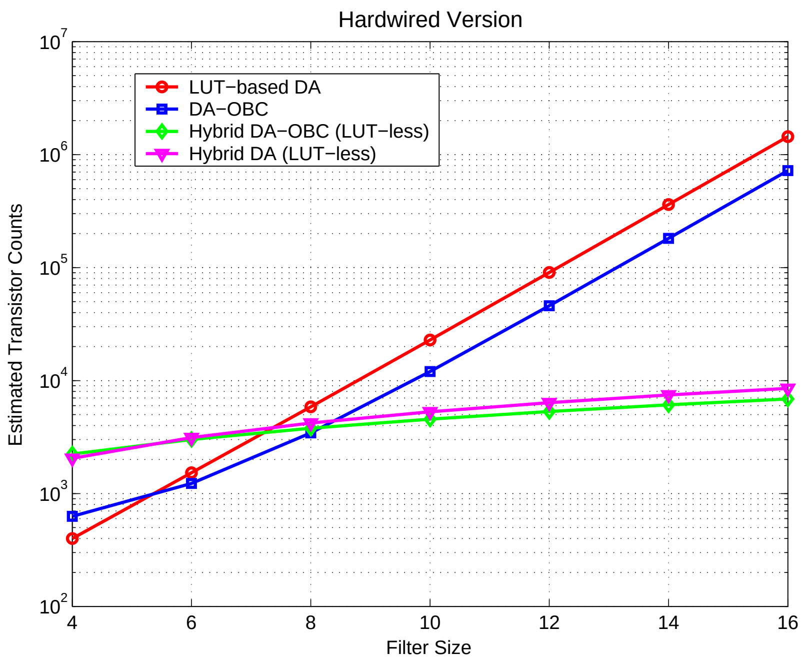


Figure 45. Transistor count estimation comparison of various base units for different filter sizes with $B_c = 18$. Filter coefficients of the hybrid DA-OBC and hybrid DA architectures are hardwired, assuming they are not changed. Hybrid architectures require a smaller area than LUT-based DA and DA-OBC architectures for high-order filters; in particular, the hybrid DA-OBC architecture requires the smallest transistor counts among the four different base units for high-order filters because it can be implemented with fixed-carry adders, which are much smaller than regular adders.

Table 8 shows the implementation results for randomly generated FIR filters, ranging from 4 to 1024 taps when the word length of the LUT, B_c, is 18 and base unit size, k, is 4. One can conclude from Table 8 that the hybrid DA architecture is hardware-efficient, requiring fewer logic elements (LEs) and memory than the original LUT-based DA, while the hybrid DA-OBC architecture is memory-efficient, requiring more LEs and less memory than the original LUT-based DA. The hybrid DA architecture is more promising than the hybrid DA-OBC for FPGA implementation since it requires fewer LEs and the same memory. For instance, a 1024-tap hybrid DA architecture implemented with a 4-tap base unit uses only 48% of LEs and 16% of memory over the LUT-based DA architecture at a cost of two additional clock cycles (assuming the adders inside the k-tap base unit are pipelined). Likewise, a 1024-tap hybrid DA architecture requires only 37% of LEs and the same memory over the hybrid DA-OBC.

Other implementation results for randomly generated FIR filters whose sizes vary from 8 to 2048 taps are presented in Table 9. The word length of the LUT, B_c, is 18, and the base unit size, k, is 8. Table 9 clearly shows that LUT-based DA architecture can synthesize only up to 512-tap FIR filters, while the hybrid DA-OBC and the hybrid DA can synthesize 1024-tap FIR filters or more higher-order FIR filters, respectively, on the same FPGA chip. In particular, the hybrid DA architecture uses only 17% of LEs and less than 1% of the memory of an Altera Stratix EP1S80F1508C6 FPGA chip.

The implementation results of the hybrid DA architecture are outstanding in terms of saving hardware resources of an FPGA for several reasons. First, the hybrid DA architecture has a more compact base unit, which consists of three adders, four 2x1 MUXs, and four registers, than the base unit of the hybrid DA-OBC, which consists of three XORs, four adders with a subtraction selector, one 2x1 MUX, and five registers. Second, even with the same compilation option and the same manners of definition of filter coefficients, Altera's Quartus II software optimizes the hybrid DA architecture more than the hybrid DA-OBC architecture, automatically hardwiring the filter coefficients of the hybrid DA architecture [1].

Table 8. Summary of the FPGA implementation results (Altera Stratix EP1S80F1508C6 FPGA chip) for randomly generated FIR filters, ranging from 4 taps to 1024 taps, when the $B_c = 18$ and $k = 4$. Two hybrid DA architectures require fewer LEs and less memory than the original LUT-based DA.

		Filter size (K)						
		4	16	64	128	256	512	1024
LUT-based DA	LE	272	551	1639	3056	5890	11547	22862 (100%)
	Memory	344	1376	5504	11008	22016	44032	88064 (100%)
LUT-less Hybrid DA-OBC	LE	300	667	2104	3984	7746	15259	30286 (132%)
	Memory	56	224	896	1792	3584	7168	14336 (16%)
LUT-less Hybrid DA	LE	210	367	887	1569	2946	5659	11086 (48%)
	Memory	56	224	896	1792	3584	7168	14336 (16%)

Table 9. Summary of the FPGA implementation results (Altera Stratix EP1S80F1508C6 FPGA chip) for randomly generated FIR filters, ranging from 8 taps to 2048 taps, when the $B_c = 18$ and $k = 8$. The LUT-based DA architecture can synthesize only up to 512-tap FIR filters, while the hybrid DA-OBC and the hybrid DA can synthesize 1024-tap FIR filters or more higher-order FIR filters, respectively. In particular, the hybrid DA architecture uses only 17% of LEs and less than 1% of memory of the Altera FPGA chip.

		Filter size (K)							
		8	16	64	128	256	512	1024	2048
LUT-based DA	LE	327	481	1359	2521	4841	9474	X	X
	Memory	4976	9952	39808	79616	159232	318464	X	X
LUT-less Hybrid DA-OBC	LE	395	609	1862	3538	6849	13468	26700	X
	Memory	112	224	896	1792	3584	7168	14336	X
LUT-less Hybrid DA	LE	228	304	638	1073	1938	3676	7116	13992
	Memory	112	224	896	1792	3584	7168	14336	28672

5.5 Reusable DA Block

The DCT has been widely used in image and audio signal processing, for its properties are very close to an optimal KLT [9, 35, 85]. The N-point one dimensional DCT is given by

$$X(k) = c(k)\sqrt{\frac{2}{N}}\sum_{n=0}^{N-1} x(n)\cos\Big(\frac{(2n+1)k\pi}{2N}\Big), \tag{55}$$

where $x(n)$ is input sequence, $c(0) = 1/\sqrt{2}$, and $c(k) = 1$ for $k = 0, 1, 2, \cdots, N-1$. The constant scaling factor, $\sqrt{2/N}$, can be neglected without loss of generality. Then Equation (55) can be rewritten in matrix form as

$$\mathbf{X} = T(N)\mathbf{x}, \tag{56}$$

where $\mathbf{X} = [X(0), X(1), \cdots, X(N-1)]^T$, $\mathbf{x} = [x(0), x(1), \cdots, x(N-1)]^T$, and $T(N)$ is an $N \times N$ matrix whose (k, n)-th component is

$$T(N)_{(k,n)} = c(k)\cos\Big(\frac{(2n+1)k\pi}{2N}\Big). \tag{57}$$

For $N = 8$, Equation (56) becomes

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \\ X(4) \\ X(5) \\ X(6) \\ X(7) \end{bmatrix} = \begin{bmatrix} A & A & A & A & A & A & A & A \\ D & E & F & G & -G & -F & -E & -D \\ B & C & -C & -B & -B & -C & C & B \\ E & -G & -D & -F & F & D & G & -E \\ A & -A & -A & A & A & -A & -A & A \\ F & -D & G & E & -E & -G & D & -F \\ C & -B & B & -C & -C & B & -B & C \\ G & -F & E & -D & D & -E & F & -G \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ x(4) \\ x(5) \\ x(6) \\ x(7) \end{bmatrix}, \tag{58}$$

where $A = \cos(\frac{\pi}{4})$, $B = \cos(\frac{\pi}{8})$, $C = \cos(\frac{3\pi}{8})$, $D = \cos(\frac{\pi}{16})$, $E = \cos(\frac{3\pi}{16})$, $F = \cos(\frac{5\pi}{16})$, $G = \cos(\frac{7\pi}{16})$, or

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \\ X(4) \\ X(5) \\ X(6) \\ X(7) \end{bmatrix} = \begin{bmatrix} \longleftarrow \mathbf{r_0} \longrightarrow \\ \longleftarrow \mathbf{r_1} \longrightarrow \\ \longleftarrow \mathbf{r_2} \longrightarrow \\ \longleftarrow \mathbf{r_3} \longrightarrow \\ \longleftarrow \mathbf{r_4} \longrightarrow \\ \longleftarrow \mathbf{r_5} \longrightarrow \\ \longleftarrow \mathbf{r_6} \longrightarrow \\ \longleftarrow \mathbf{r_7} \longrightarrow \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ x(4) \\ x(5) \\ x(6) \\ x(7) \end{bmatrix}, \tag{59}$$

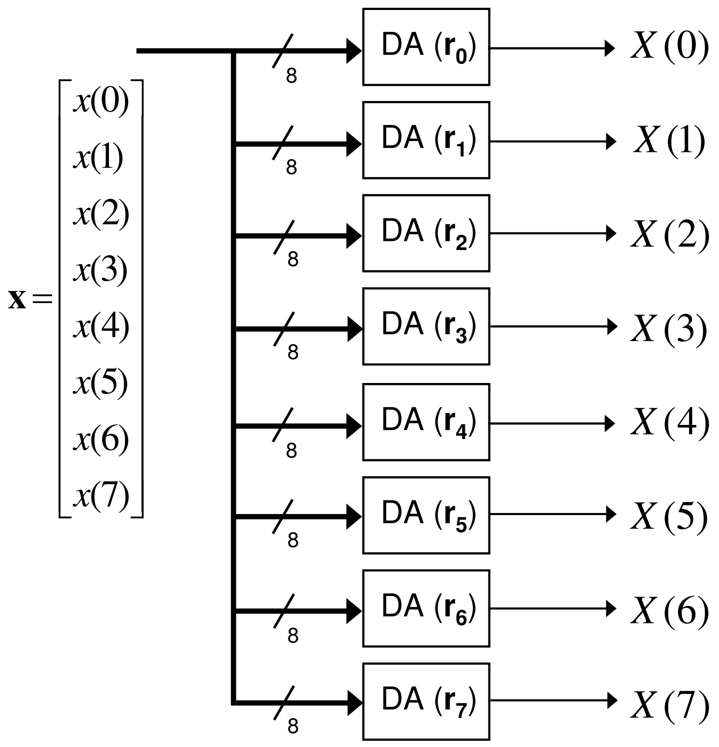


Figure 46. Direct implementation of the 8-point DCT with DA. Each DA unit stores all the possible sums of the corresponding row vector in the LUT. The shift register unit, which is not drawn, generates 8-bit address lines for 2^8 LUT.

where each $\mathbf{r_k}$ is k-th row vector.

Figure 46 shows the direct implementation of the 8-point DCT using DA. The single shift register unit, which is not drawn here, creates 8-bit address lines for eight DA units. Each DA unit generates $X(0), X(1), \cdots, X(7)$, respectively. The LUT in each DA unit stores each row vector $(\mathbf{r_0}, \mathbf{r_1}, \cdots, \mathbf{r_7})$ of transform matrix, $T(8)$. The direct implementation of the 8-point DCT requires 8-bit address lines for 2^8 size LUT.

The DCT architecture shown in Figure 46 can be enhanced, for example, by using the symmetric or periodic properties of the DCT matrix, as explained in Equation (58). An example of an enhanced 8-point DCT architecture using the even-odd decomposition method [103] is shown in Figure 47. The even-odd decomposition method [14, 15, 103] exploits that 8×8 matrix, $T(8)$, can be decomposed into two 4×4 matrices with the following coefficients:

$$\begin{bmatrix} X(0) \\ X(2) \\ X(4) \\ X(6) \end{bmatrix} = \begin{bmatrix} A & A & A & A \\ B & C & -C & -B \\ A & -A & -A & A \\ C & -B & B & -C \end{bmatrix} \begin{bmatrix} x(0)+x(7) \\ x(1)+x(6) \\ x(2)+x(5) \\ x(3)+x(4) \end{bmatrix}, \tag{60}$$

$$\begin{bmatrix} X(1) \\ X(3) \\ X(5) \\ X(7) \end{bmatrix} = \begin{bmatrix} D & E & F & G \\ E & -G & -D & -F \\ F & -D & G & E \\ G & -F & E & -D \end{bmatrix} \begin{bmatrix} x(0)-x(7) \\ x(1)-x(6) \\ x(2)-x(5) \\ x(3)-x(4) \end{bmatrix}. \tag{61}$$

Although this architecture still requires eight DA units, it requires only 4-bit address lines for 2^4 size LUT instead of 8-bit address lines for 2^8 size LUT. The extra hardware required for this architecture increases to four adders and four subtractors. The DCT architecture of Figure 47 requires significantly less silicon area since the LUT size is decreased exponentially for all DA units. As the size of DCT, N, increases, the area savings of Figure 47 become more dramatic.

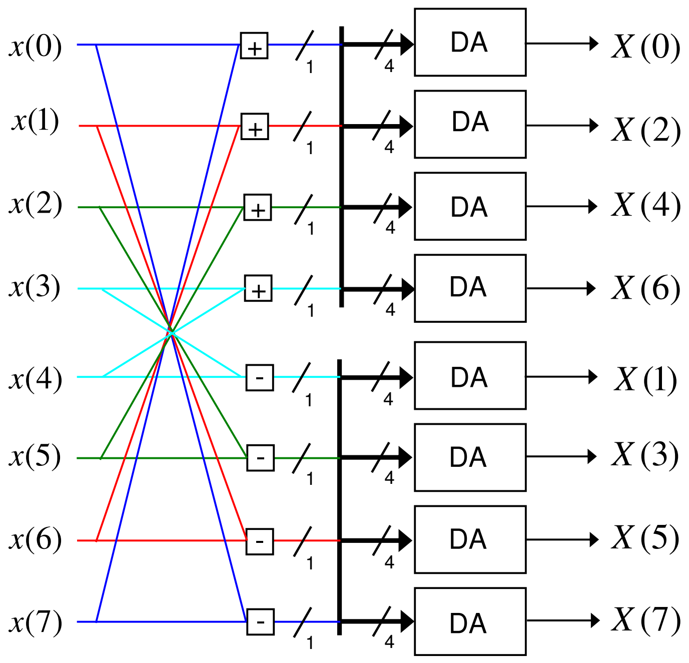


Figure 47. DA architecture for the even-odd decomposition of the 8-point DCT. Although this architecture still requires eight DA units, each DA unit has 4-bit address lines for 2^4 size LUT instead of 8-bit address lines for 2^8 size LUT. The extra hardware required for this architecture increases to four adders and four subtractors.

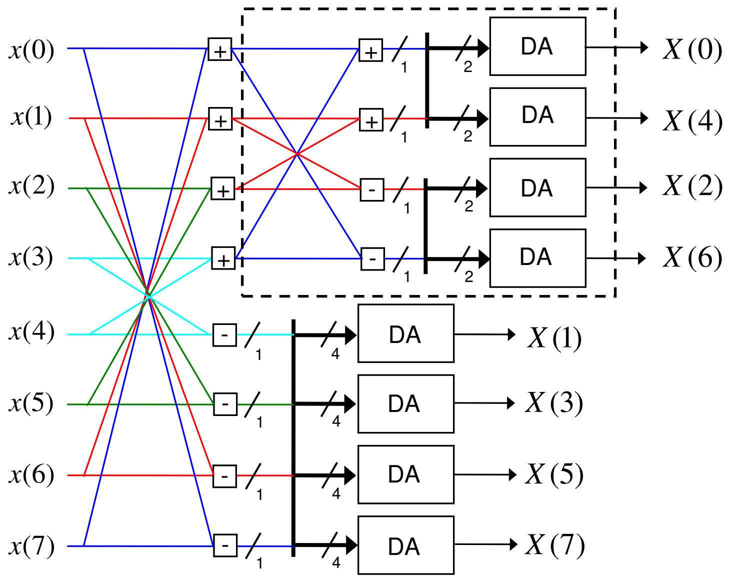


Figure 48. DA architecture for the second recursive even-odd decomposition of the 8-point DCT. Equation (60) is further decomposed into even and odd matrices of a smaller size, exploiting the symmetric and periodic properties.

It is obvious that a more recursive decomposition of the DCT matrix can lead to more area savings. Equation (61), unfortunately, can't be easily decomposed into smaller matrices, but Equation (60) can be further decomposed into even and odd matrices of a smaller size [103] as follows:

$$\begin{bmatrix} X(0) \\ X(4) \end{bmatrix} = \begin{bmatrix} A & A \\ A & -A \end{bmatrix} \begin{bmatrix} (x(0)+x(7)) + (x(3)+x(4)) \\ (x(1)+x(6)) + (x(2)+x(5)) \end{bmatrix}, \tag{62}$$

$$\begin{bmatrix} X(2) \\ X(6) \end{bmatrix} = \begin{bmatrix} B & C \\ C & -B \end{bmatrix} \begin{bmatrix} (x(0)+x(7)) - (x(3)+x(4)) \\ (x(1)+x(6)) - (x(2)+x(5)) \end{bmatrix}. \tag{63}$$

Figure 48 shows the DA architecture for the 8-point DCT, which is simplified using Equations (62) and (63). For smaller DA units, only 2-bit address lines are required for 2^2 size LUT. The more recursive decomposition of an even decomposition matrix is still possible to optimize area usage, as proposed in [103].

From these DCT implementations, one can observe that LUT size could be reduced or

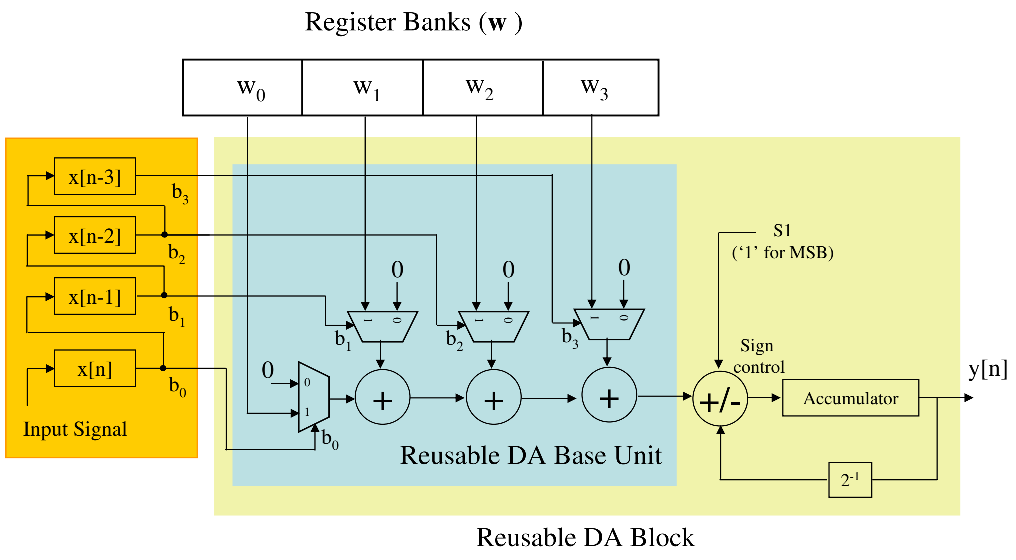


Figure 49. Block diagram of the reusable DA base unit and the reusable DA. The LUT-less hybrid DA architecture can lead to a new architecture, called "reusable DA block," by moving registers storing filter coefficients to the outside of DA base unit. Since the DA base unit has a generic architecture independent of filter coefficients, it can be reused for different filter coefficients.

the LUT itself could be replaced with optimized adder networks using common subexpression sharing approaches. However, the number of DA units can't be changed since each DA unit stores its own unique filter coefficients in the LUT, even after the application of various area saving techniques such as recursive matrix decomposition [103] and adder-based DA [14, 15].

To eliminate this drawback of the DA architecture for DCT, we propose a new DA architecture that enables the recycling of the DA unit. Among the hybrid DA architectures introduced in the previous section, the LUT-less hybrid DA architecture can lead to a new architecture that we name "reusable DA block (RDA)," shown in Figure 49. The RDA consists of a regular adder/shifter unit and a "reusable DA base unit," whose filter coefficients are not located inside DA base unit, so the DA base unit has a generic architecture independent of the filter coefficients. A new 8-point DCT architecture using the RDA is shown in Figure 50. Two clockwise commutators are used to feed the row vectors $(\mathbf{r_0}, \mathbf{r_1}, \cdots, \mathbf{r_7})$ of the original DCT matrix, $T(8)$, into the RDA, and pass the filtered output of the RDA to

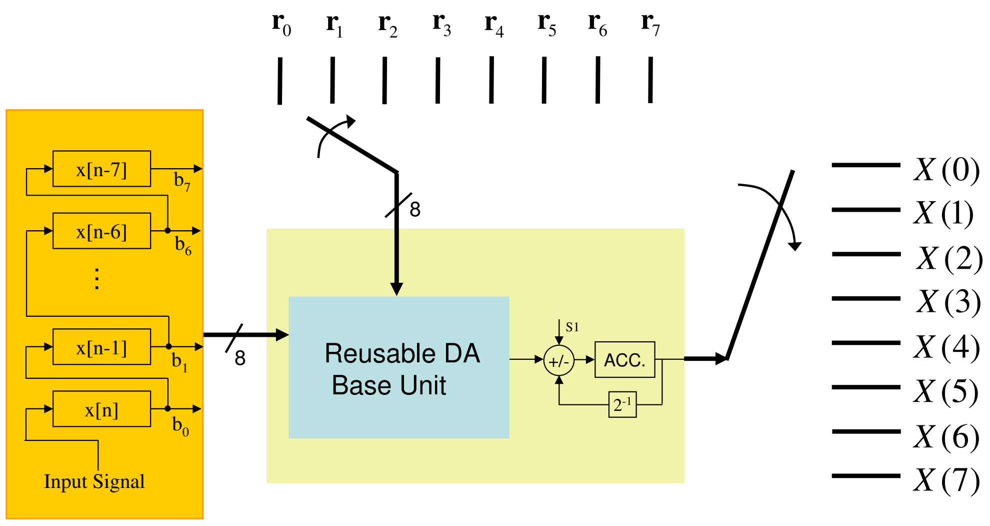


Figure 50. Multiplexed architecture for the 8-point DCT with reusable DA. The RDA consists of a regular adder/shifter unit and a "reusable DA base unit," whose filter coefficients are not located inside of the DA base unit so the DA base unit has a generic architecture independent of filter coefficients. Two commutators, rotating clockwise, are used to feed the row vectors of the original DCT matrix, $T(8)$, into the RDA, and pass the filtered output of the RDA to the eight DCT coefficients sequentially.

the eight DCT coefficients $(X(0), X(1), \cdots, X(7))$ sequentially. The RDA is a highly area-optimized architecture for the matrix-vector multiplication such as DCT and DFT. Even though the RDA requires more clock cycles than the typical DA, this architecture is very promising compared to non-DA approaches in terms of throughput.

CHAPTER 6
DISCRETE-TIME AEC

In Chapter 2, it was shown that the DA implementation of FIR filters enables the filtering operation to end at a fixed number of clock cycles, which is equal to the bit precision of the input samples, regardless of the filter size. Thus, the DA architecture is popular for the high-speed implementation of high-order FIR filters. DA can also be extended to fixed-coefficient IIR filters without loss of generality. However, the application of LUT-based DA architecture to adaptive filters has not been very successful. Several past attempts have been made to implement adaptive filters using DA [21, 95], but the approximations made to standard adaptation algorithms may be unsuitable for practical applications.

The application of traditional LUT-based DA architecture to adaptive filters shows several challenges. First, RAMs, instead of ROMs, are required for the realization of the LUT since filter coefficients need to be updated at every sample clock. Many different types of RAMs are available in the market, but the silicon sizes of RAMs are generally larger than those of ROMs because ROMs use as little as a single transistor to store a single-bit data. The larger area size of RAMs is a particularly serious problem for high-order adaptive filters. Second, the implementation of the LMS adaptive filter using DA demands that entries of the LUT containing all possible combination sums of the filter coefficients need to be recalculated and updated on a sample-by-sample basis. A conventional implementation method updating each filter coefficient individually and reconstructing the contents of the LUT with new coefficients is computationally expensive and time consuming, causing a significant reduction in the filter throughput. A few attempts have been made [21, 95], but updating a large LUT still poses a major problem in DA-based adaptive filters. For instance, a conventional update of the LUT could take approximately 1000 clock cycles for a 128-tap FIR filter [4].

In this chapter, new architecture for a DA-based LMS adaptive filter (DAAF) is proposed. The DAAF architecture is a hardware implementation of the LMS adaptive filter with higher throughput than traditional hardware implementation, particularly for high-order filter sizes. Since the DAAF architecture is targeted to the high-speed implementation of high-order adaptive filters used in a discrete-time AEC, we will attempt to implement up to 1024 taps. Furthermore, the throughput is almost independent of the filter size and largely depends on the bit precision of the input signal. An area-optimized DAAF architecture for high-order filter sizes is also presented in this section.

6.1 Traditional Hardware Implementation

A typical implementation of the LMS adaptive filter on a hardware system with a single multiply-and-accumulator (MAC) unit will require K MAC operations to perform the filtering and K MAC operations to perform the weight adaptation shown in Table 1. For real-time implementation of high-order filters, the system clock must be much faster than the input signal sampling rate.

Alternatively, multiple MAC units may be employed to parallelize the adaptive filter implementation [2]. In the multiple MAC-based LMS adaptive filter (MMAF) system described in [2], the filtering and the weight adaptation are done using one or more custom hardware MAC units. The implementation on the MMAF system is similar to the implementation of the adaptive filter on a DSP processor. In many modern DSP processors, up to four MAC units process input samples simultaneously. The throughput of the MMAF system depends on the filter size and the number of MAC units. As the number of MAC units increases, higher throughput can be achieved. However, the required silicon area and power consumption increase as well.

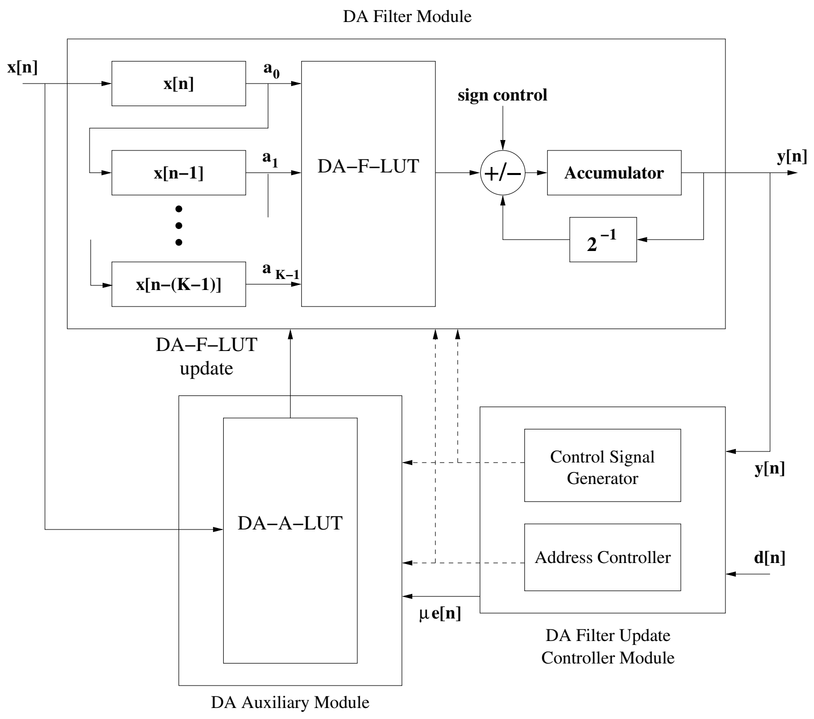


Figure 51. Block diagram of the DAAF for a single LUT-based implementation. The DA-A-LUT contains all possible combination sums of K most recent input samples, and the DA-F-LUT contains all possible combination sums of the filter coefficients. The DA-A-LUT in the DA auxiliary module is used for the fast update of DA-F-LUT of the DA filter module.

6.2 DA LMS Adaptive Filter

In this section, a novel filter coefficient update structure that requirs fewer clock cycles than the conventional LUT recalculation is described. The conventional update method of the LUT must update each new filter coefficient, recalculate all possible combination sums of the new coefficients, and then store the new sums in the LUT. This is computationally expensive and time consuming, causing a significant reduction in filter throughput. The proposed method applies the LMS algorithm directly to the contents of the LUT by introducing an additional auxiliary LUT.

The overall system diagram of the DAAF is shown in Figure 51. It has a "DA filter module" for the filtering operation and a "DA auxiliary table module" for an efficient filter coefficient update, and a "DA filter update control module" for controlling the previous two modules. The DA filtering LUT (DA-F-LUT) in the DA filter module contains all possible combination sums of the coefficients. The DA-F-LUT is exactly the same with LUT used for fixed-coefficient FIR filters. The entries of the DA-F-LUT must be recalculated and updated at every sample clock, once the weight update term of Table 1, $\mu e[n]x[n-k]$, is available. The DA auxiliary table module contains an auxiliary LUT, denoted as the DA auxiliary LUT (DA-A-LUT), which contains all possible combination sums of the K most recent input samples. The DA-A-LUT is mainly introduced for the speed-efficient update of the DA-F-LUT. The table size and architecture of the DA-A-LUT are identical to those of the DA-F-LUT. It is this analogous architecture that allows the adaptation scheme described below to work efficiently. The DA filter update control module generates addresses and control signals for the DA-A-LUT and DA-F-LUT. It also calculates the error signal, $e[n]$, and generates $\mu e[n]x[n-k]$ for the LMS adaptive filter. The address for updating the DA-A-LUT is rotated to the left at every sample clock for an efficient update mechanism of the DA-A-LUT, which is explained later.

A single filtering and adaptation step of the DAAF algorithm at the time instance n is described in Figure 52. The notations DA-A-LUT[n] and DA-F-LUT[n] are used to refer to the DA-A-LUT and DA-F-LUT at the time instance n. The filtering operation on $x[n], x[n-1] \ldots x[n-K+1]$ is performed using the DA-F-LUT[n]. While the filtering operation is in progress, the DA-A-LUT is updated from DA-A-LUT[$n-1$] to DA-A-LUT[n]. When both the filtering and the update of the DA-A-LUT have been completed, the DA-F-LUT is updated from DA-F-LUT[n] to DA-F-LUT[$n+1$]. Once DA-F-LUT[$n+1$] is updated, the filtering and adaptation step at time n is complete. The DAAF then awaits the arrival of a new sample $x[n+1]$, and the entire algorithm of Figure 52 is repeated. The updates of the DA-A-LUT and DA-F-LUT, upon the arrival of the sample $x[n]$, are described in the next section.

6.2.1 DA-A-LUT Update

The simple and slow update method for the DA-A-LUT would be to recalculate all possible combination sums of the K most recent samples (after each new sample arrives) and then

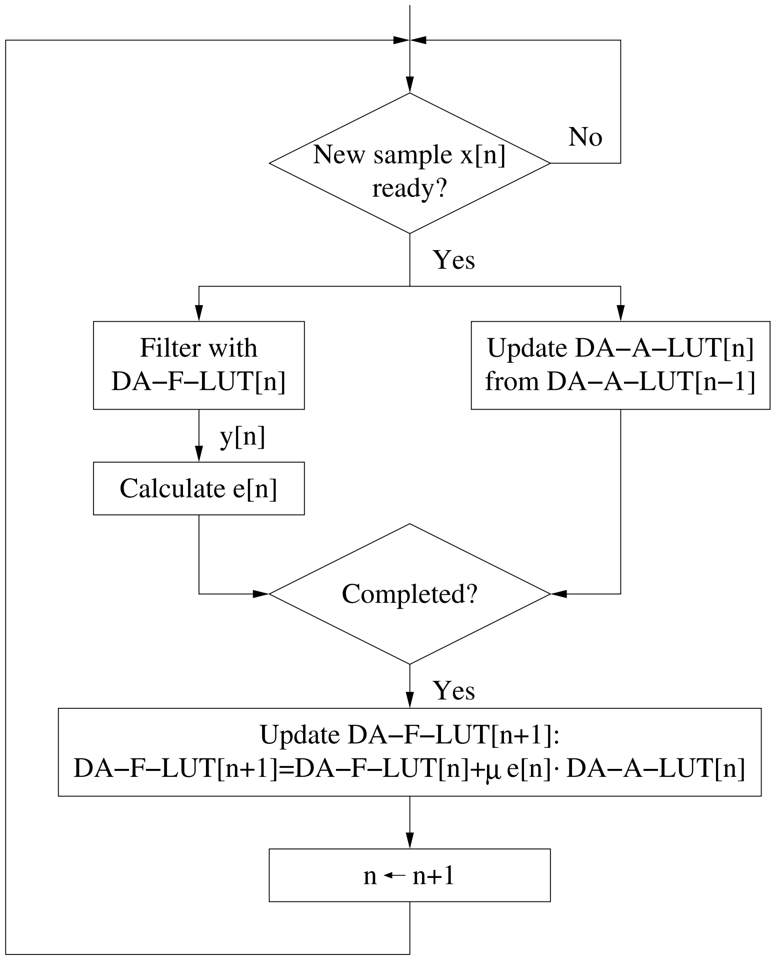
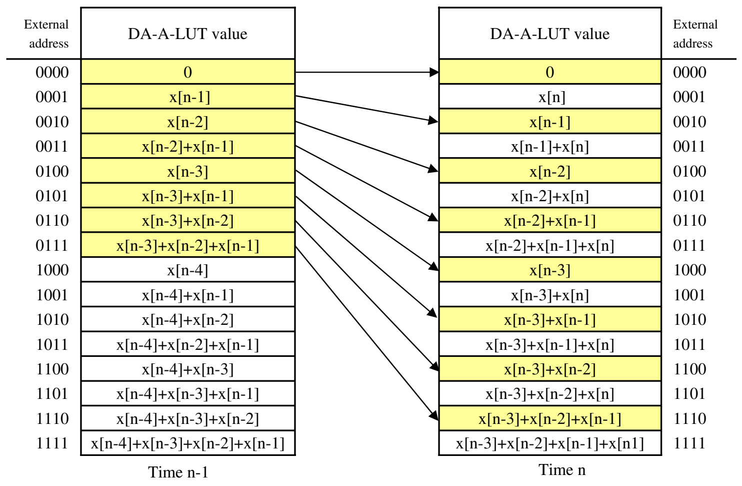


Figure 52. Simplified flowchart of the DAAF. While the filtering operation is in progress, the DA-A-LUT is updated from DA-A-LUT[$n-1$] to DA-A-LUT[n]. When both the filtering and the update of the DA-A-LUT are complete, the DA-F-LUT is updated from DA-F-LUT[n] to DA-F-LUT[$n+1$].



External address	DA-A-LUT value
0000	0
0001	x[n-1]
0010	x[n-2]
0011	x[n-2]+x[n-1]
0100	x[n-3]
0101	x[n-3]+x[n-1]
0110	x[n-3]+x[n-2]
0111	x[n-3]+x[n-2]+x[n-1]
1000	x[n-4]
1001	x[n-4]+x[n-1]
1010	x[n-4]+x[n-2]
1011	x[n-4]+x[n-2]+x[n-1]
1100	x[n-4]+x[n-3]
1101	x[n-4]+x[n-3]+x[n-1]
1110	x[n-4]+x[n-3]+x[n-2]
1111	x[n-4]+x[n-3]+x[n-2]+x[n-1]

Time n-1

DA-A-LUT value	External address
0	0000
x[n]	0001
x[n-1]	0010
x[n-1]+x[n]	0011
x[n-2]	0100
x[n-2]+x[n]	0101
x[n-2]+x[n-1]	0110
x[n-2]+x[n-1]+x[n]	0111
x[n-3]	1000
x[n-3]+x[n]	1001
x[n-3]+x[n-1]	1010
x[n-3]+x[n-1]+x[n]	1011
x[n-3]+x[n-2]	1100
x[n-3]+x[n-2]+x[n]	1101
x[n-3]+x[n-2]+x[n-1]	1110
x[n-3]+x[n-2]+x[n-1]+x[n1]	1111

Time n

Figure 53. Update of the DA-A-LUT entries from time $n-1$ to time n. The entries of the DA-A-LUT addressed from 0000 to 0111 at time $n-1$ can be reused by mapping these values to the even address locations of the table at time n. The values in the odd address locations at time n can be updated by adding the newest sample $x[n]$ with preceding even-addressed data.

store the results in the DA-A-LUT. A more efficient update method can be devised by considering how the DA-A-LUT changes as a new sample arrives and the oldest sample is discarded [54].

Figure 53 shows the update of DA-A-LUT[n] from DA-A-LUT[$n-1$] for $K = 4$. It may be observed that the contents of even-addressed locations (locations whose addresses have a 0 in the LSB) of the DA-A-LUT[n] are the contents of the lower half (locations whose addresses have a 0 in the MSB) of the DA-A-LUT[$n-1$]. It may also be observed that the contents of the odd addressed locations (locations whose addresses have a 1 in the LSB) of the DA-A-LUT[n] can be obtained from the even-addressed locations of the DA-A-LUT[n] according to

$$\text{DA-A-LUT}_{(2l+1)}[n] = \text{DA-A-LUT}_{(2l)}[n] + x[n], \quad l = 0, \ldots, 2^{K-1} - 1. \tag{64}$$

The update of the DA-A-LUT[n] from DA-A-LUT[$n-1$] is summarized in the following two steps:

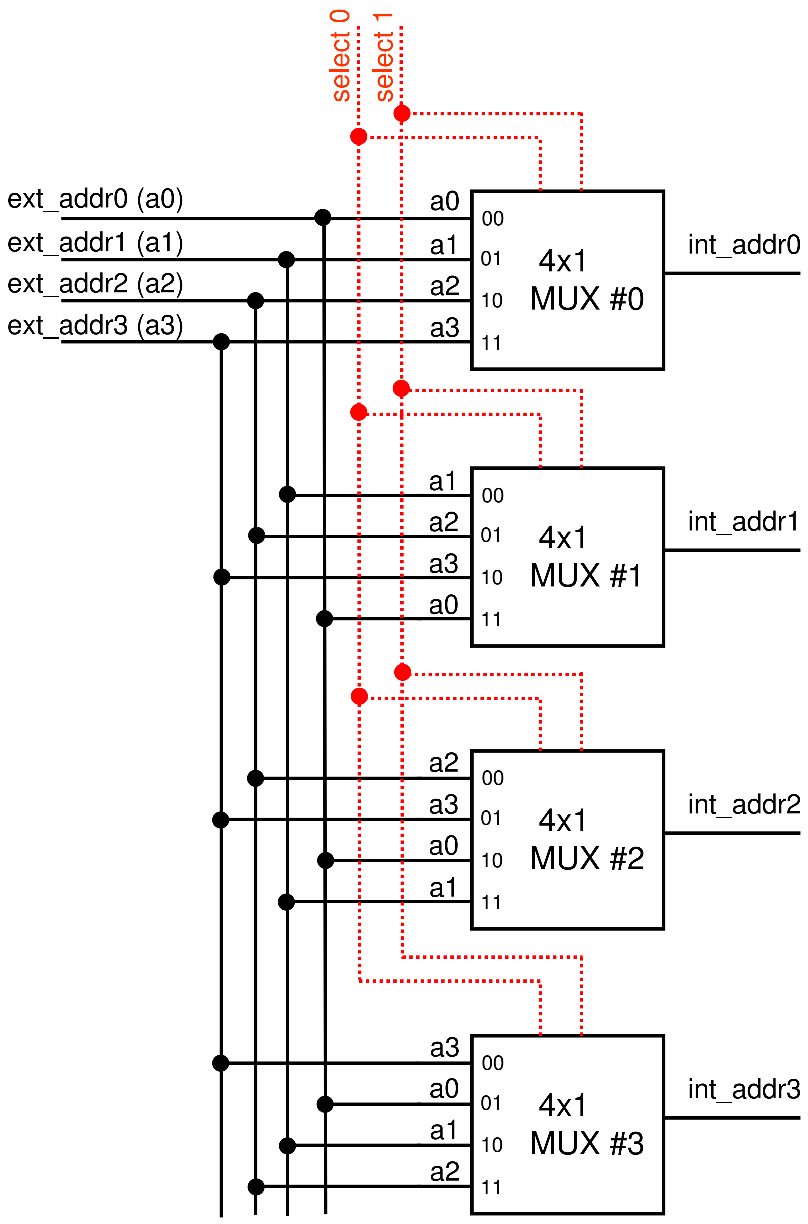


Figure 54. Block diagram for the DA-A-LUT address rotation. The lower half of the DA-A-LUT[$n-1$] is re-mapped to even-addressed locations of the DA-A-LUT[n]. Instead of physically moving the contents of the DA-A-LUT, this re-mapping operation can be performed by a simple left rotation of the K address lines of the DA-A-LUT. Address rotation allows the physical contents of memory to remain the same, even as the external logic sees the table as re-mapped.

Table 10. Rotation of address lines for the DA-A-LUT when $K = 4$. The address in the table is illustrated in Figure 54. External addresses (a0, a1, a2, and a3) are left rotated at every sample clock cycle.

Select	Internal Address			
	int_addr0	int_addr1	int_addr2	int_addr3
00	a0	a1	a2	a3
01	a1	a2	a3	a0
10	a2	a3	a0	a1
11	a3	a0	a1	a2

- The lower half of the DA-A-LUT[$n-1$] is re-mapped to even-addressed locations of the DA-A-LUT[n], as shown by the arrows in Figure 53. Instead of physically moving the contents of the DA-A-LUT, this re-mapping operation can be performed by a simple left rotation of the K address lines of the DA-A-LUT. Address rotation allows the physical contents of memory to remain the same, even as the external logic sees the table as re-mapped. The address rotation can be achieved using the circuit shown in Figure 54. The relationship between the internal and the external addresses for $K = 4$ is shown in Table 10. The term "int_addr" in the Table 10 refers to the physical addresses of the DA-A-LUT, and "ext_addr" refers to the address as seen by the external logic. It can be observed that the external address referring to a given internal address at time n is the left-rotated version of the external address referring to the same internal address at time $n-1$. Therefore, the effect of address rotation can be accomplished by connecting "ext_addr" and "int_addr" via K, K-to-1 input multiplexers, as shown in Figure 54. The $\log_2(K)$ select lines of each of the K multiplexers are connected to the $\log_2(K)$ bits of a counter, which is incremented with the sample clock. Thus, by address rotation, the entire mapping of the DA-A-LUT can be done instantaneously at the arrival of the new sample $x[n]$.

- It must be noted that address rotation maps the upper half of the DA-A-LUT[$n-$ 1], which contains sums involving the oldest sample $x[n-4]$, to the odd-addressed locations at time n. The entries in these odd-addressed locations of the DA-A-LUT[n]

are overwritten by values obtained according to Equation (64). In other words, the contents of the odd-addressed locations of the DA-A-LUT[n] are obtained by reading the contents of the corresponding preceding even-addressed locations, adding the newest sample $x[n]$, and then storing the result back in the odd-addressed locations.

6.2.2 DA-F-LUT Update

The DAAF updates the contents of the DA-F-LUT directly, instead of updating the filter coefficients and then storing the sums of the filter coefficients to the DA-F-LUT. Once the update of the DA-A-LUT[n] as well as the filtering operation are done, an update of the DA-F-LUT[$n + 1$] is performed. The update from time n to $n + 1$ of the rth entry of the DA-F-LUT is given by

$$\text{DA-F-LUT}_{(r)}[n+1] = \text{DA-F-LUT}_{(r)}[n] + \mu e[n]\text{DA-A-LUT}_{(r)}[n]. \tag{65}$$

The DA-F-LUT[$n + 1$] is updated by reading the same memory location in both the DA-F-LUT[n] and the DA-A-LUT[n], multiplying the output of the DA-A-LUT[n] by $\mu e[n]$, adding this quantity to the output of the DA-F-LUT[n], and finally storing the result back in the same memory location of the DA-F-LUT[$n + 1$]. This process is repeated from address 1 to address $2^K - 1$. The entry in address 0 is not updated since it always has a zero value. The addresses and the control signals for the DA-F-LUT update are provided by the DA filter update controller module.

The multiplication operation of the entries of the DA-F-LUT[n] by $\mu e[n]$ can be accomplished by using a custom hardware multiplier. To save the number of hardware multipliers required to compute $\mu e[n]\text{DA-F-LUT}_{(r)}[n]$, μ is often set to a power of two, replacing multiplication with a right shift. More saving of hardware multipliers can be achieved by using the well-known sign error (SE)-LMS. However, it is well known that the performance of SE-LMS is worse than the performance of the LMS algorithm, especially when the absolute value of an error is large at the beginning of the convergence [35, 52]. Another method would be quantize error (QE)-LMS with μ fixed to some power of two and the

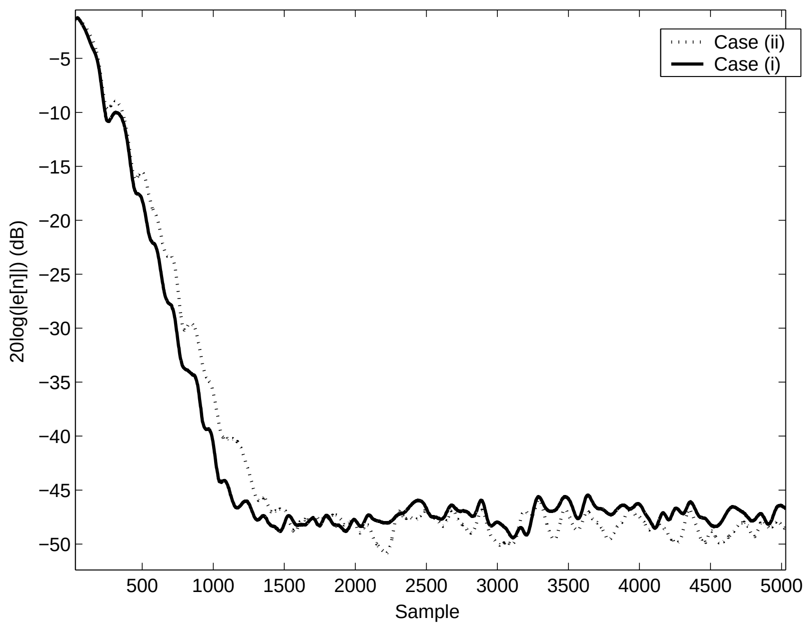


Figure 55. MATLAB simulation of the learning curve for LMS implementations: Case (i) Actual $\mu e[n] \times$ DA-A-LUT$[n]$ is used, and Case (ii) $\mu e[n]$ is quantized to one of $L = 8$ values (powers of 2) using a round function. The $e[n]$'s used in the plot for both the above mentioned cases are obtained by averaging 150 independent trials of the respective MATLAB simulations.

error signal, $e[n]$, quantized to the closest power of two, essentially quantizing the product $\mu e[n]$ to a power of two. This enables us to minimize the on-chip area usage by replacing the hardware multiplier by a simple barrel shifter. In other words, the product of the contents of the DA-A-LUT$[n]$ with $\mu e[n]$ is approximated by a right shift of the contents of the DA-A-LUT$[n]$.

Depending on the quantization method of the error signal, $e[n]$, the performance of the learning curve can suffer degradation. This degradation of the QE-LMS, however, can be minimized, as shown in Figure 55, by employing the rounding function, which gives the closest power of the 2 value of the error signal. In both cases, a white Gaussian random noise signal with a zero mean and unit variance was used as the input and the desired signal, $d[n]$, was generated by filtering the input with a 256-tap low-pass FIR filter. The $e[n]$'s used in Figure 55 for both the above mentioned cases are obtained by averaging 150 independent trials of MATLAB simulations.

6.2.3 DAAF for High-Order Filters

For large values of K, the DAAF can be implemented using multiple smaller DA base filtering and adaptation units (DA-BFAUs). The idea of breaking up the DAAF into multiple DA-BFAUs is similar to the partial sum technique for fixed FIR filters. The filter output, $y[n]$, is generated by summing the filtered outputs of all DA-BFAUs using an adder tree. Figure 56 shows the K-tap DAAF architecture implemented with m DA-BFAUs, each of size k. For simplicity, a k-tap DA-BFAU will be referred to as DA-BFAU(k). A K-tap DAAF architecture having m DA-BFAU(k), when $K = m \times k$, will be referred to as DAAF(m,k). For example, when a 4-tap DA-BFAU is used to implement a 128-tap adaptive LMS filter, the DAAF architecture will be denoted as DAAF(4,32).

A detailed view of the DA-BFAU for $k = 4$ and $B = 16$ is shown in Figure 57. Each DA-BFAU contains a system similar to the one shown in Figure 51. However, the following optimizations are performed to make the DAAF more area and memory efficient:

- Since the addresses and control signals for all the DA-BFAUs are essentially the same, a single DA filter update controller module that fans its control signals and addresses to all the DA-BFAUs is used.

- Instead of using one accumulator and shift register in each DA-BFAU, a single DA accumulator and shift register are used at the output of the adder tree. Thus, the outputs of the DA-F-LUT of the DA-BFAUs are directly connected to the adder tree unit, and the resulting sum is accumulated and shifted B times to generate the output, $y[n]$.

- A single shift register unit containing the bits of the input samples ($x[n], x[n-1], \ldots, x[n-K+1]$) is used. The addresses for accessing the contents of the DA-F-LUTs of the DA-BFAUs are derived from this shift register.

The “adder1” in Figure 57 is a half-adder for the update of the DA-A-LUT, and it adds the current input sample to the odd addressed locations (locations whose addresses have

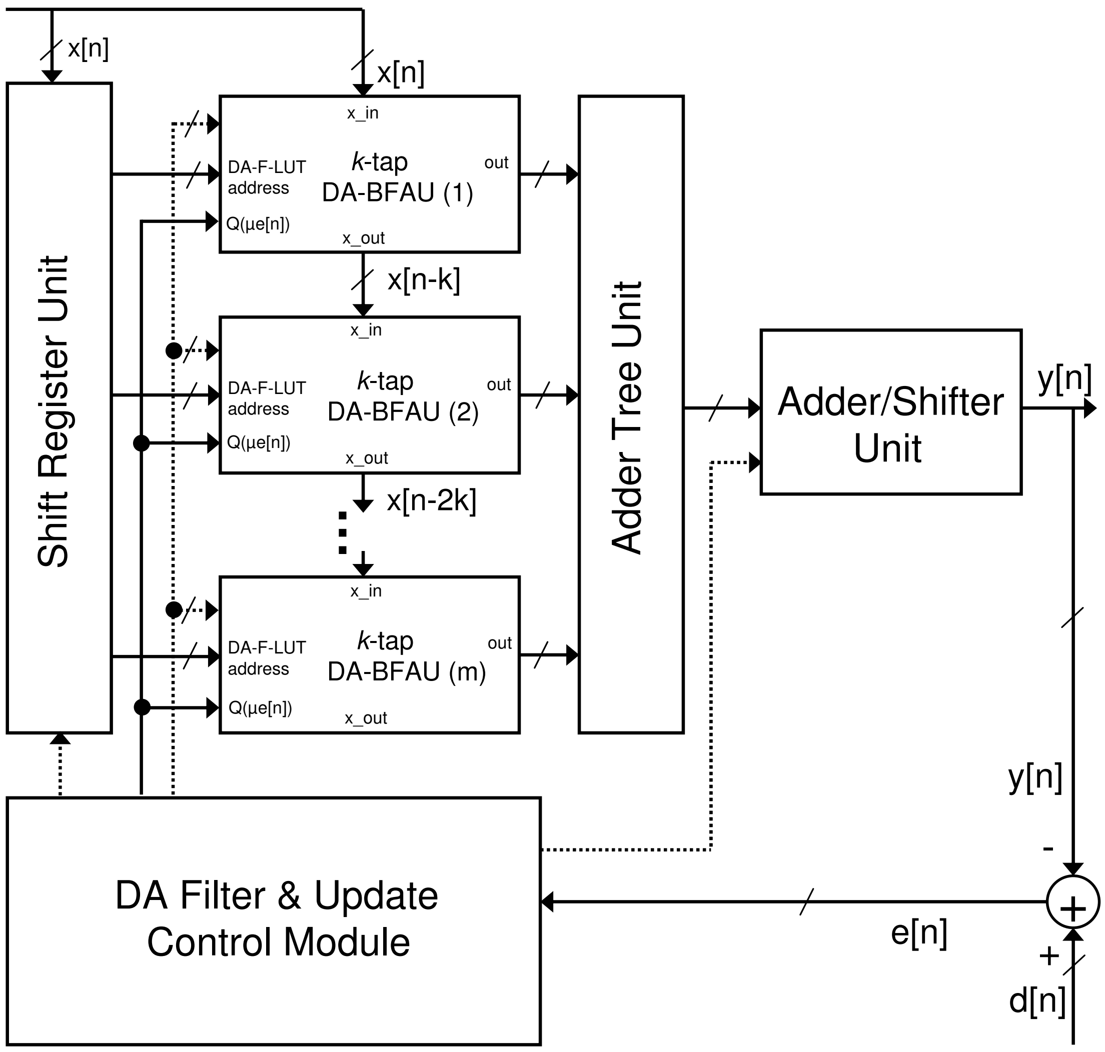


Figure 56. Block diagram of the area-optimized DAAF for high-order filters. The shift register unit, the adder/shift unit, and the adder tree unit are the same with fixed FIR filters. A *K*-tap DAAF architecture having *m* DA-BFAU(*k*), when $K = m \times k$, is referred to as DAAF(*m*,*k*).

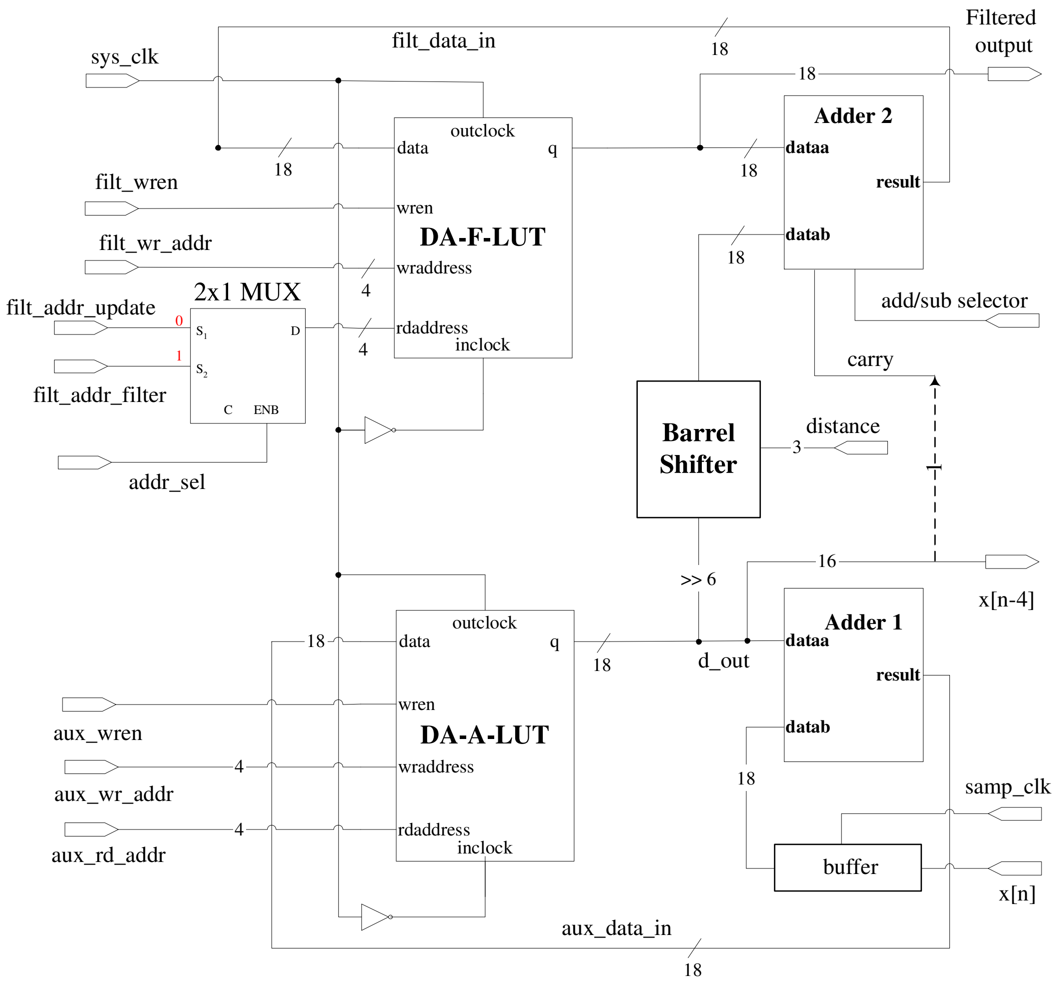


Figure 57. Detailed view of the DA base filtering and adaptation unit, DA-BFAU(4), for $\mu = 6$, and $B = 16$. Each DA-BFAU contains a system similar to Figure 51, but it is more optimized so that the DAAF implementation are more area and memory efficient. For the filtering operation, 2x1 MUX chooses address lines from the shift register unit. For the weight adaptation operation, it chooses address lines generated sequentially from the counter beginning from 1 to $2^k - 1$.

a 1 in the LSB) of the DA-A-LUT according to Equation (64). The "adder2" is a full-adder with sign control for the update of DA-F-LUT, according to Equation (65). A "barrel shifter" is used for further area optimization instead of a hardware multiplier. With μ fixed to some power of two, the barrel shifter shifts its input, μDA-A-LUT, to the right by "distance," which was calculated at the DA filter and update control module. A "2x1 MUX" selects address lines for the DA-F-LUT. For the filtering operation, 2x1 MUX chooses address lines from the shift register unit, which is defined as "filt_addr_filter" in Figure 57. For the weight adaptation operation, it chooses the address lines from the DA filter and update control module, which is defined as "filt_addr_update" in Figure 57. A "filt_addr_update" is generated sequentially from the counter, beginning from 1 to $2^k - 1$. Two address lines for DA-A-LUT, "aux_wr_addr" and "aux_rd_addr," are actually "int_addr" of Figure 54 and Table 10. The "aux_rd_addr" corresponds to the even external addresses, and the "aux_wr_addr" corresponds to the next odd external addresses of "aux_rd_addr."

6.2.4 Implementation Results

In this section, implementation results of the DAAF architecture (Figure 56) are compared in terms of four metrics to quantify design the trade-offs: throughput, memory requirements, logic complexity, and power consumption estimate [3]. These metrics are used to compare the performances of the DAAF(m, k) for different values of k and m.

Throughput: The throughput is defined as the number of signal samples processed by an adaptive filter per second. If "T" is the number of clock cycles required for filtering and updating the filter coefficients according to the adaptation algorithm, then

$$\text{Throughput} = \frac{\text{clock rate}}{\text{T}}. \tag{66}$$

For a K-tap DA adaptive FIR filter implemented as DAAF(m, k), the update of the DA-A-LUT can be done in 2^{k-1} clock cycles, as described in Section 6.2.1. This can be done along with the filtering operation, which takes B clock cycles. Thus, the total number of clock cycles for the filtering and the update of the DA-A-LUT is $\max(B, 2^{k-1})$. The updated

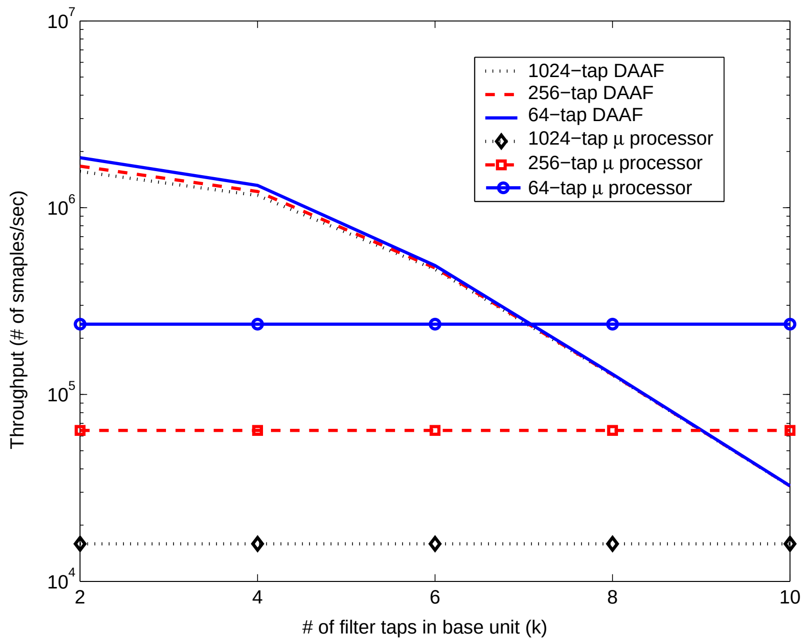


Figure 58. Throughput versus the number of filter taps in the base unit for 64-, 256-, and 1024-tap adaptive filters implemented on a microprocessor and the DAAF. A microprocessor is assumed not to have multiple hardware multipliers. The throughput of the DAAF does not vary significantly for the change of the overall filter size, *K*, but vary significantly for the change of the filter taps of DA-BFAU, *k*. It clearly shows that the throughput improvement of the DAAF over a microprocessor becomes wider as *K* increases. A maximum of two orders of throughput improvement over a microprocessor can be achieved for a 1024-tap LMS adaptive filter implemented with the DAAF(2,512) or the DAAF(4,256).

DA-A-LUT is then used to update the DA-F-LUT, requiring 2^k clock cycles. Finally, the adder tree unit requires $\lceil \log_2(m) \rceil$ clock cycles. Thus, the overall K-tap DAAF requires $2^k + \max(B, 2^{k-1}) + \lceil \log_2(m) \rceil$ clock cycles. The throughput for the DAAF is given by

$$\text{Throughput} = \frac{\text{clock rate}}{2^k + \max(B, 2^{k-1}) + \lceil \log_2(m) \rceil}. \tag{67}$$

The throughput of the DAAF and a microprocessor for 64-, 256-, and 1024-tap adaptive filters are shown in Figure 58. A microprocessor is assumed not to have multiple hardware multipliers. A 50 MHz system clock is used for both a microprocessor and DAAF. It can be observed that the throughput of the DAAF does not vary significantly due to the change in the overall filter size, K, but varies significantly due to the change of the filter taps of DA-BFAU, k. Figure 58 clearly shows that the throughput improvement of the DAAF over a microprocessor becomes wider as the overall filter size, K, increases. A maximum of two

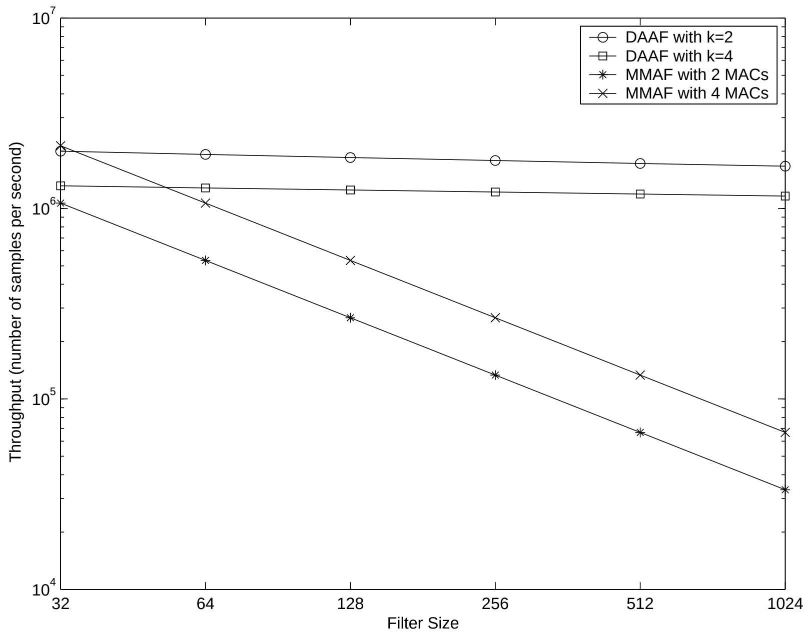


Figure 59. Throughput comparison of the DAAF and the MMAF for various filter sizes, K. For $K = 32$, the throughput of the MMAF with 4 MACs is almost the same as that of the DAAF with $k = 2$. However, as the filter tap size, K, increases, the throughput of the MMAF (note that the throughput is drawn in log scale) decreases exponentially while the DAAF maintains almost the same throughput.

orders of throughput improvement over a microprocessor can be achieved for a 1024-tap LMS adaptive filter implemented with the DAAF(2,512) or the DAAF(4,256).

As explained in Sec. 6.1, most modern DSP processors have multiple hardware multipliers. Since many different DSP processors have their own unique specialized-hardware logic for fast MAC operation, comparing their performance is challenging. Instead, we synthesized a general MMAF processor, having multiple hardware multipliers that objectively compare the throughput with the DAAF. Figure 59 shows the throughput comparison between the MMAF with 2 or 4 MACs and the DAAF with $k = 2$ or $k = 4$. For $K = 32$, the throughput of the MMAF with 4 MACs is almost the same as that of the DAAF with $k = 2$. However, as the filter tap size, K, increases, the throughput of the MMAF (note that the throughput is drawn in log scale) decreases exponentially while the DAAF maintains almost the same throughput.

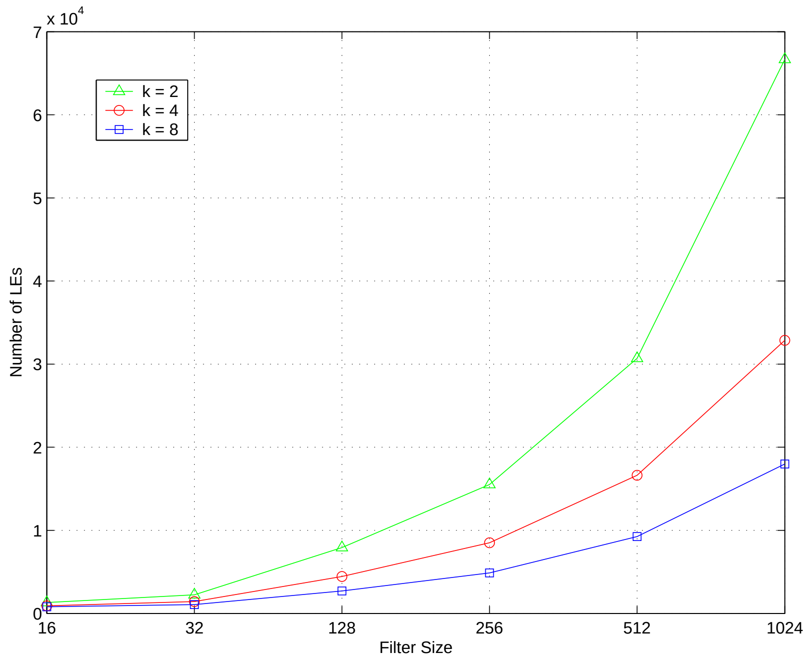


Figure 60. Number of LEs for various adaptive filter sizes, K, and base unit sizes, k. When K is fixed, DA-BFAU with larger k is more area efficient, requiring a fewer number of LEs.

Number of LEs: In programmable logic systems, the size of the logic design is measured by the number of LEs. For Altera's Stratix architecture, the platform used in implementation, the LE is the smallest unit for implementing logic functions. Each LE contains a four-input LUT, a programmable register, and a carry chain with carry select capability. Further details of the LE can be found in [1]. Ten LEs are grouped into one logic array block (LAB) and the LABs are interconnected through a row- and column-based network. In this paper, the number of LEs, instead of the number of LABs, is considered as a metric for area usage since it conveys more detailed information about the actual chip area used in the implementation.

Figure 60 shows the number of LEs for various adaptive filter sizes, K, and base unit sizes, k. When K is fixed, DA-BFAU with larger k is more area efficient, requiring a fewer number of LEs. It makes sense that DA-BFAU with smaller k requires more full adders, which take up a relatively large silicon area, in the adder tree unit. It should be noted from Figures 58 and 60 that there are trade-offs between the number of LEs and throughput. For

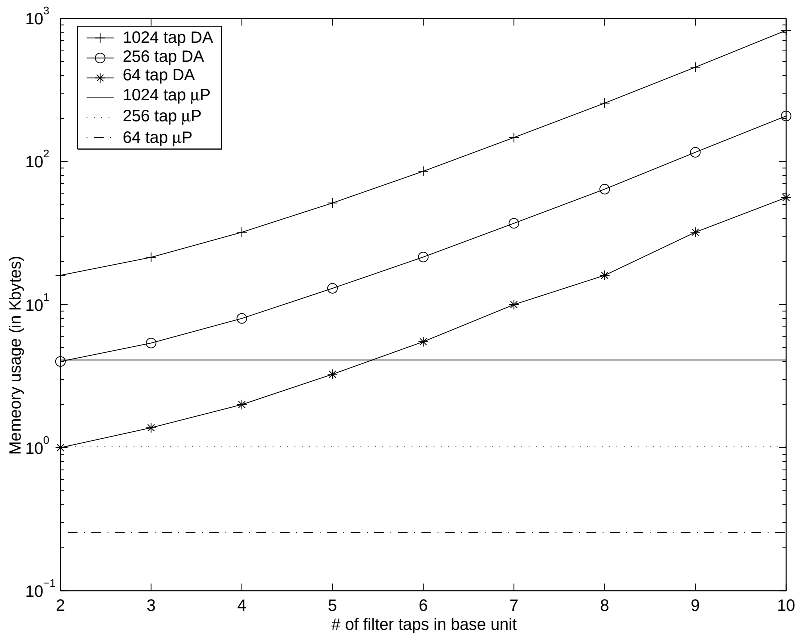


Figure 61. Memory usage versus the number of filter taps in the base unit for 64-**,** 256-**, and** 1024-**tap filters implemented on a microprocessor and the DAAF.**

area-optimized implementation, DA-BFAU with larger *k* can be a good choice, while for throughput-optimized implementation, DA-BFAU with smaller *k* is advantageous.

Memory: The DA filtering method, due to its nature of using LUTs, is known to be more memory intensive than DSP processor implementation. Altera's Stratix architecture provides three types of RAM blocks consisting of M512, M4K, and M-RAM blocks [1]. However, we compare memory usage in KB, which is available in the compilation reports of FPGA synthesis software, as a metric rather than the number of used M512, M4K, and M-RAM blocks for more general comparison with other hardware platforms.

Figure 61 illustrates the memory usage of 64-, 256-, and 1024-tap adaptive filters implemented on the DAAF and a microprocessor. Figure 61 shows that memory requirements for the DAAF grow exponentially as the base unit size increases, and DA-BFAU with smaller *k* is more memory efficient. Although the DAAF requires at least four times more memory than microprocessor-based implementation since the memory is considered

Table 11. Memory requirements in KB for various K and k. Memory requirements for the DAAF grow exponentially as the base unit size increases, and DA-BFAU with smaller k is more memory efficient.

	Overall Filter Size (K)						
k	16	32	64	128	256	512	1024
2	0.16	0.33	0.66	1.31	2.63	5.25	10.5[1]
4	0.31	0.63	1.25	2.5	5	10	20
8	2.41	4.81	9.63	19.25	38.50	77	154

cheaper than the specialized logic for a microprocessor or a DSP core, this marginal memory increase might be implemented without additional cost. Moreover, as it is found in Table 11, the overall memory requirement for a 1024-tap LMS algorithm implemented with DAAF(4,256) is no more than 20 KB.

Power Consumption Estimates: The power consumption estimates are obtained from the "Simulation Report" generated by using the PowerGauge™ feature of Altera's Quartus II software. According to [1], the PowerGauge™ estimates power through a software simulation of the hardware design.

Table 12 shows the power consumption comparison of the DAAF for various K and k. It can be seen that DA-BFAU with larger k when overall filter size (K) is fixed is more power efficient than DA-BFAU with smaller k. The power consumption of the DAAF architecture can be compared with other methods of implementation in conjunction with the throughput advantage of the DAAF architecture. For instance, two orders of throughput improvement of the 1024-tap LMS adaptive filter implemented with the DAAF(2,512) or the DAAF(4,256) over a microprocessor means that the DAAF can be clocked at a maximum of 100 times more slowly than a microprocessor with the same performance. It is well known that the overall power consumption of the system decreases as the clock speed decreases.

Finally, to verify the convergence property of the DAAF implemented on an Altera

[1]These values are extrapolated since it is not possible to fit a 1024-tap DAAF with 2-tap base units on the Stratix EP1S80F1508C6 FPGA.

Table 12. Power consumption estimates in mW for various K and k. The power estimates are obtained using the PowerGauge™ feature of Altera's Quartus II software. The DA-BFAU with larger k is more power efficient than DA-BFAU with smaller k.

	Overall Filter Size (K)						
k	16	32	64	128	256	512	1024
2	14.43	24.54	34.54	54.01	90.53	132.5	
4	15.83	19.82	37.15	53.91	84.77	115	144.8
8	11.87	14.94	24.95	31.16	37.14	48.17	72.87

Stratix EP1S80F1508C6 FPGA system, the learning curve for a 512-tap LMS adaptive filter implemented with the DAAF(4,128) is presented in Figure 62. White Gaussian noises with a zero mean are used as input samples for this measurement.

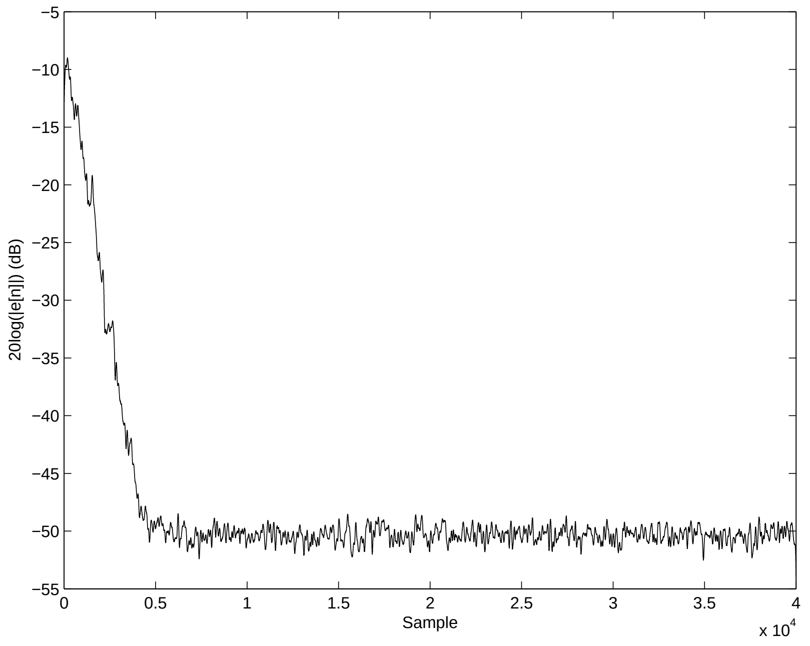


Figure 62. Learning curve for a 512-tap LMS adaptive filter implemented with the DAAF(4,128), measured from Altera Stratix EP1S80F1508C6 FPGA system. White Gaussian noises with zero mean are tested for the verification of implementation.

CHAPTER 7
HYBRID DA LMS ADAPTIVE FILTER

Two novel hybrid architectures–the hybrid DA-OBC architecture and the hybrid DA architecture–introduced in Chapter 5, enable efficient hardware implementation for fixed-coefficient filters. Hybrid architectures provide more flexibility than LUT-based DA or adder-based DA by allowing arbitrary selection of the LUT size, ranging from 2^k to as low as 0 for a k-tap DA base unit, or from 2^{k-1} to as low as 0 for a k-tap DA-OBC base unit. It should be noted that LUT-based DA and adder-based DA are only two extreme cases of hybrid architectures. FPGA implementation results confirm that hybrid architectures can implement higher-order FIR filters because of their flexibility in the selection between LEs and memory.

In addition, new DA architecture for the LMS adaptive filter (DAAF) was presented in Chapter 6. The DAAF structure is based on a LUT and provides high-speed hardware implementation of the LMS adaptive filter by introducing additional auxiliary LUT for fast update of the original filtering LUT. The DAAF, however, still requires intensive memory usage as the base unit size, k, increases since it is LUT-based architecture.

7.1 Hybrid DA LMS

To overcome increased memory problems of the DAAF for high-order base unit, hybrid architectures are extended to an LMS adaptive filter in this section. The new structure, referred to as “hybrid DA LMS,” has features of both the hybrid DA architecture and the DAAF architecture. It allows smaller LUT size, ranging from 2^k to as low as 0, for a k-tap DA base unit. The LUT update scheme of the DAAF, which makes the DAAF architecture attractive for high-speed filtering applications, can be again applied to a smaller LUT of the hybrid DA LMS. In the hybrid DA LMS architecture, some of the filter coefficients are stored in a LUT, called a DA-F-LUT, while the remaining filter coefficients are stored in

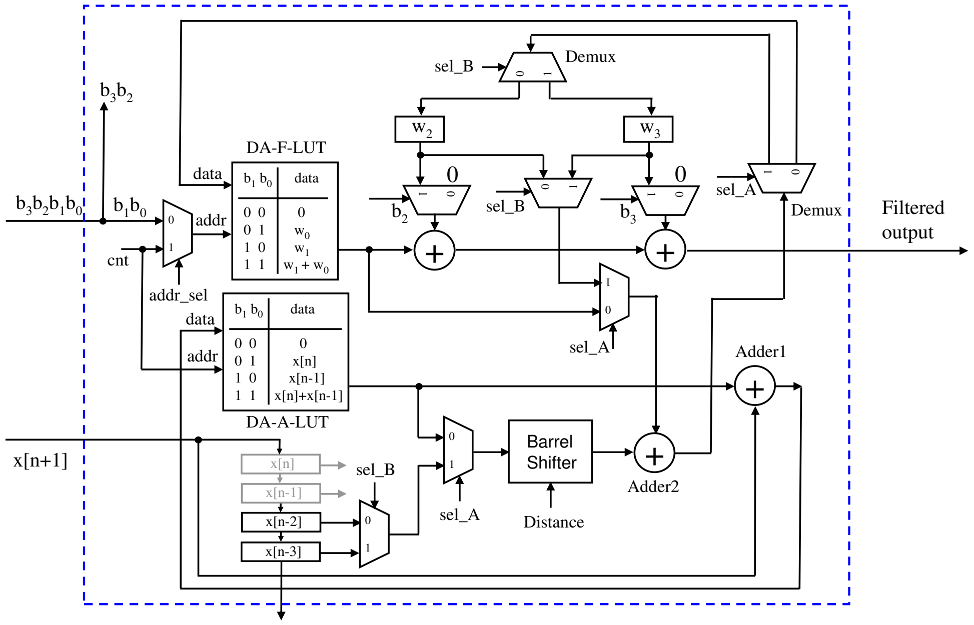


Figure 63. Example of a block diagram of a 4-tap base unit for the hybrid DA LMS architecture. Two filter coefficients ($k_L = 2$) are stored in a LUT and the remaining two coefficients ($k_A = 2$) are stored in the registers. The hybrid DA LMS has features of both the hybrid DA architecture and the DAAF architecture. The LUT update scheme of the DAAF, which makes the DAAF architecture attractive for high-speed filtering applications, can be again applied to a smaller LUT of the hybrid DA LMS.

the registers. Let k_L be the number of filter coefficients stored in the DA-F-LUT and k_A be the number of filter coefficients stored in the registers. Then the following is satisfied.

$$k = k_L + k_A, \tag{68}$$

where k is the number of base unit taps of the hybrid DA LMS.

Details of the hybrid DA LMS are presented with an example. Figure 63 shows an example of the block diagram of a 4-tap base unit for the hybrid DA LMS architecture. The shift register unit, the adder tree unit, and the adder/shifter unit are identical to those of the hybrid DA architecture. The timing control logic of the DA filter update controller module should be changed from the DAAF architecture since the base unit structure is changed to Figure 63. Two filter coefficients ($k_L = 2$) are stored in the LUT and the ramaining two

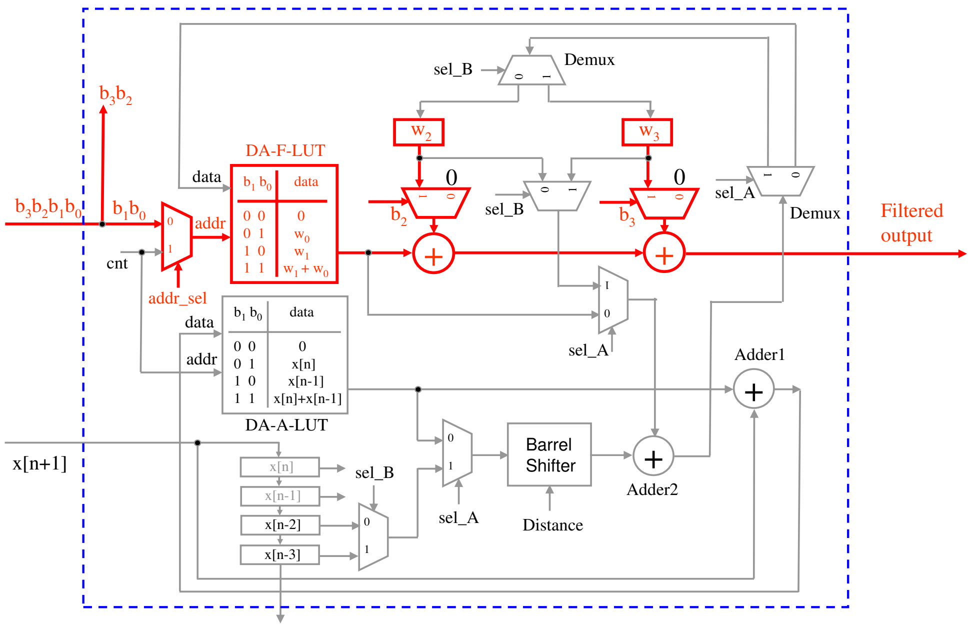


Figure 64. Filtering operation for a 4-tap base unit with $k_L = 2$ and $k_A = 2$. Dark lines (red) are active ones for the filtering operation of the hybrid DA LMS architecture. When filtering is performed, active structures are the same as the hybrid DA architecture.

coefficients ($k_A = 2$) are stored in the registers. Four bits of addresses ($b_3b_2b_1b_0$) from the shift register unit are connected to the base unit, but only two bits (b_1b_0) are used as addresses to the DA-F-LUT and the remaining two bits are used as selectors for 2x1 MUX, by selecting either filter coefficients or zeroes. It can be seen from Figure 63 that the size of both LUTs are reduced from 2^k to 2^{k_L}. Adders labelled as “Adder1” and “Adder2” are the same as adders used inside the base unit of the DAAF (DA-BFAU), and the remaining two adders are newly inserted as they were in the hybrid DA architecture.

Figure 64 illustrates the active logics for the filtering operation. Dark lines (red) are active for the filtering operation of the hybrid DA LMS architecture. When the filtering operation is in progress, these active logics are exactly the same as the hybrid DA architecture, since the filtering operation is identical in both fixed-coefficient filters and adaptive filters.

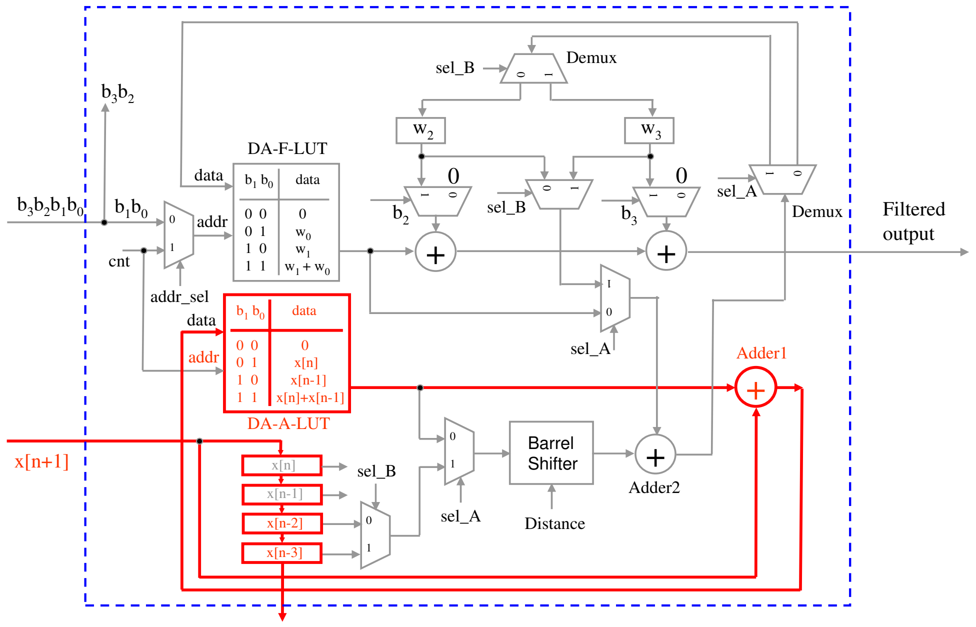


Figure 65. Update of the DA-A-LUT and input registers for a 4-tap base unit with $k_L = 2$ and $k_A = 2$. Dark lines (red) are active for the DA-A-LUT and input registers update operation. This operation takes place at the same time as the filtering operation. The same update scheme of the DAAF can be used for the hybrid DA LMS architecture; the only difference is that the update clock cycles decrease from 2^{k-1} to 2^{k_L-1}.

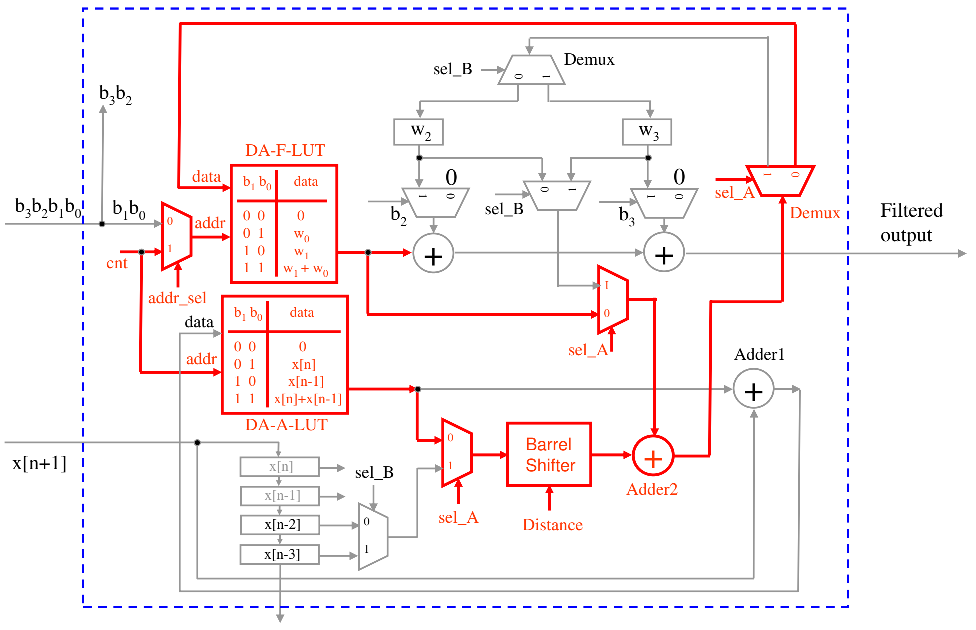


Figure 66. Update of the filter coefficients stored in the DA-F-LUT for a 4-tap base unit with $k_L = 2$ and $k_A = 2$. Dark lines (red) are active for the DA-F-LUT update operation.

Figure 65 illustrates the active logics for the update operation of the DA-A-LUT and input registers. This operation takes place at the same time as the filtering operation, shown in Figure 64. For the DAAF architecture, DA-A-LUT[$n + 1$] was updated using address rotation and data reuse of DA-A-LUT[n] to lower the required clock cycles to 2^{k-1}. The same update scheme can be used for the hybrid DA LMS architecture; the only difference is that the update clock cycles decrease from 2^{k-1} to 2^{k_L-1}. Once the update of DA-A-LUT is over, a new input sample, $x[n + 1]$, at time index $n + 1$ is loaded into the input registers shown in the lower-left corner of Figure 65. Input registers store k previous input signals and are used for the update of the filter coefficients stored in the registers.

In the hybrid DA LMS architecture, k_L filter coefficients are stored in the DA-F-LUT, and k_A filter coefficients are stored in the registers. The update of these two sets of filter coefficients is done in tandem as follows. The filter coefficients in the DA-F-LUT are updated as follows:

$$\text{DA-F-LUT}_{(r)}[n+1] = \text{DA-F-LUT}_{(r)}[n] + \mu e[n]\text{DA-A-LUT}_{(r)}[n], \quad 0 \le r \le 2^{k_L} - 1, \quad (69)$$

and then the filter coefficients in the registers are updated as follows:

$$w_i[n+1] = w_i[n] + \mu e[n]x[n-i], \quad (k_L \le i \le k-1), \quad (70)$$

where k is the base unit size of the hybrid DA LMS architecture.

Figure 66 illustrates the update of the filter coefficients stored in the DA-F-LUT, as explained in Equation (69), for a 4-tap base unit when $k_L = 2$ and $k_A = 2$. The hardware logics involved in the DA-F-LUT update are fundamentally the same as the DA-F-LUT update logics of the DAAF architecture. The DA-F-LUT[$n+1$] is updated by reading the same memory location in both the DA-F-LUT[n] and the DA-A-LUT[n], multiplying the output of the DA-A-LUT[n] by $\mu e[n]$, adding this quantity to the output of the DA-F-LUT[n], and finally storing the result back in the same memory location of the DA-F-LUT[$n+1$]. This process is repeated from address 1 to address $2^{k_L} - 1$. The product of the contents of the DA-A-LUT[n] with $\mu e[n]$ is approximated using a barrel shifter. The "sel_A" control signal is set to 0 for the DA-F-LUT update operation. The addresses of both LUTs are connected to the counter, which is incremented from 1 to $2^{k_L} - 1$.

Figure 67 illustrates the update of the filter coefficients stored in the registers, as explained in Equation (70), for a 4-tap base unit when $k_L = 2$ and $k_A = 2$. While this operation is in progress, "sel_A" is set to 1 and "sel_B" is increased from 0 to $k_A - 1$ for the sequential update of the filter coefficients stored in the registers. The size of the MUXs and the DEMUX, controlled by "sel_B," is k_Ax1.

7.2 Performance Analysis

Throughput: Section 6.2.4 showed that the number of clock cycles (T) of the DAAF(k,m) architecture for the K-tap LMS adaptive filter ($K = k \times m$) was

$$\text{T} = 2^k + \max(B, 2^{k-1}) + \lceil \log_2(m) \rceil. \quad (71)$$

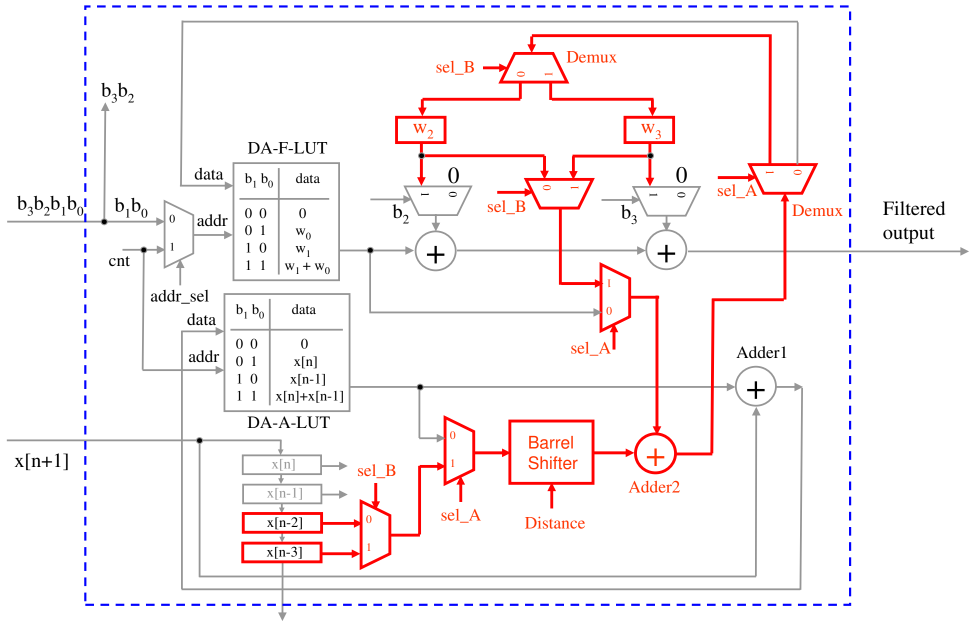


Figure 67. Update of the filter coefficients stored in the registers for a 4-tap base unit with $k_L = 2$ and $k_A = 2$. Dark lines (red) are active for this operation of the hybrid DA LMS architecture. The size of the MUXs and the DEMUX, controlled by "sel_B," is k_Ax1.

In the hybrid DA LMS architecture, the number of clock cycles (T2) is

$$\mathrm{T2} = \begin{cases} 2^{k_L} + k_A + \max(B, 2^{k_L - 1}) + \lceil \log_2(m) \rceil & \text{if} \quad k_L \neq 0 \\ k + B + \lceil \log_2(m) \rceil & \text{if} \quad k_L = 0 \end{cases} \tag{72}$$

The adder tree unit summing up the filtered output of the base unit requires $\lceil \log_2(m) \rceil$ for both the DAAF and the hybrid DA LMS, since these two architectures have differences only inside the base unit. The filtering operation requires $\max(B, 2^{k_L - 1})$ clock cycles for $k_L \neq 0$ and B clock cycles for $k_L = 0$, respectively. Neither the DA-F-LUT nor the DA-A-LUT are required for $k_L = 0$. This structure is called the “LUT-less hybrid DA LMS” and is a special example of the hybrid DA LMS architecture, where k_L is 0 and k_A is k. This architecture minimizes memory usage and maximizes the LEs of an FPGA. If we increase k_L from 0 to k gradually, the number of LEs can be decreased at the expense of more increased memory usage.

The major difference between the clock cycles of the DAAF and the hybrid DA LMS results from the way the filter coefficients are updated. It takes 2^k clock cycles for updating 2^k size DA-F-LUT for the DAAF, while it takes $2^{k_L} + k_A$ clock cycles for the hybrid DA LMS architecture. 2^{k_L} clock cycles are for the update of the filter coefficients stored in the DA-F-LUT, and k_A are for the update of the filter coefficient stored in the registers. Some additional overheads are necessary in the hybrid DA LMS architecture due to the MUXs and the DEMUX in the critical data path. However, for the LUT-less hybrid DA LMS architecture, it takes only $k_A (= k)$ clock cycles for the filter update since all the filter coefficients are stored in the registers. This clearly shows that the LUT-less hybrid DA LMS is superior to the DAAF in terms of required clock cycles. For instance, to update the filter coefficients of a 4-tap base unit, the LUT-less hybrid DA LMS requires only 4 clock cycles with additional small overheads, while the DAAF requires 16 clock cycles.

This indicates that the base unit size of the LUT-less hybrid DA can increase to 16 to achieve the same throughput as the DAAF. In the DAAF, the increase of base unit size causes a decrease in throughput, as shown in Figure 58. Increasing the base unit size without additional clock cycles has the following two advantages. First, the number of adders inside the adder tree unit decreases as the number of base unit, m, decreases. This can save additional clock cycles for the adder tree unit. Second, the barrel shifter required for the filter update equation inside the base unit can be replaced by a hardware multiplier for more precision. If the base unit size is small for larger K, the number of the base unit, m, becomes large, prohibiting the simultaneous use of m amount of dedicated hardware because of the limited number of hardware multipliers.

Resource Usage: As an example of the hybrid DA LMS architecture, the LUT-less hybrid DA LMS was implemented on an Altera's Stratix EP1S80F1508C6 FPGA chip. In addition, the LUT-less hybrid DA-OBC architecture was also implemented on the same type of FPGA chip for comparison with the hybrid DA LMS architecture. It should be noted that the general hybrid DA-OBC architecture is harder to apply to adaptive filters since the efficient update method for a LUT of DA-OBC is yet known. However, the LUT-less hybrid DA-OBC can be easily extended to adaptive filters since no LUT was required inside the base unit.

Table 13. Summary of the FPGA implementation results (Altera Stratix EP1S80F1508C6 FPGA chip) for LMS adaptive filters, ranging from 4 **to** 1024 **taps, when the** $B_c = 18$ **and** $k = 4$**.**

		Filter size (K)						
		4	16	64	128	256	512	1024
DAAF	LE	543	916	2432	4474	8554	16705	X
	Memory	632	2528	10112	20224	40448	80896	X
LUT-less Hybrid DA-OBC	LE	601	1595	5539	10781	21264	42096	X
	Memory	92	368	1472	2944	5888	11776	X
LUT-less Hybrid DA	LE	571	1434	4882	9500	18702	37097	74132
	Memory	96	384	1536	3072	6144	12288	24576

Tables 13 to 15 show the FPGA implementation results of various LMS adaptive filter sizes for 4-tap, 8-tap, and 16-tap base units, respectively. Bit precision of filter coefficients, B_c, is 18. In Table 13, one can see that only the hybrid DA LMS architecture is capable of implementing a 1024-tap LMS adaptive filter on the same type of FPGA chip. The hybrid

DA-OBC fails to implement a 1024-tap LMS adaptive filter since its LE exceeds the maximum LE limit (79040) of an FPGA chip. Interestingly, the DAAF also fails to implement a 1024-tap LMS adaptive filter although its expected number of required LEs and memory usage are only about 42% and 1% of the full capacity of the FPGA, respectively.

Table 14. Summary of the FPGA implementation results (Altera Stratix EP1S80F1508C6 FPGA chip) for LMS adaptive filters, ranging from 8 to 1024 taps, when the $B_c = 18$ and $k = 8$.

		Filter size (K)						
		8	16	64	128	256	512	1024
DAAF	LE	652	796	1630	2704	4895	9267	18024
	Memory	9840	19680	78720	157440	314880	629760	1259520
LUT-less Hybrid DA-OBC	LE	871	1461	4963	9622	18967	37558	74920
	Memory	184	368	1472	2944	5888	11776	23552
LUT-less Hybrid DA	LE	791	1299	4321	8356	16407	32495	64674
	Memory	192	384	1536	3072	6144	12288	24576

Table 14 shows the implementation results for various LMS adaptive filter sizes with a 8-tap base unit size. Unlike Table 13, all three structures can successfully synthesize a 1024-tap LMS adaptive filter. The hybrid DA LMS architecture for a 1024-tap LMS adaptive filter requires three and a half times more LEs than the DAAF and only 9.5% of memory usage of the DAAF. This, however, is not a fair comparison in the sense that required clock cycles for the 8-tap DAAF and the 8-tap hybrid DA LMS are different. Silicon area comparison between different architectures should be done at the same clock cycles or at the same throughput. Figure 68 shows the FPGA resource usage rate of the hybrid DA LMS architecture over the DAAF at the same throughput. It can be seen that the hybrid DA LMS requires 25% to 65% more LEs and 80% less memory than the DAAF when compared at the same throughput condition.

Table 15 shows the implementation results for various LMS adaptive filter sizes with 16-tap base unit size. The DAAF architecture fails to synthesize a 16-tap base unit for all filter sizes attempted, while both hybrid LMS architectures can synthesize a 16-tap base unit.

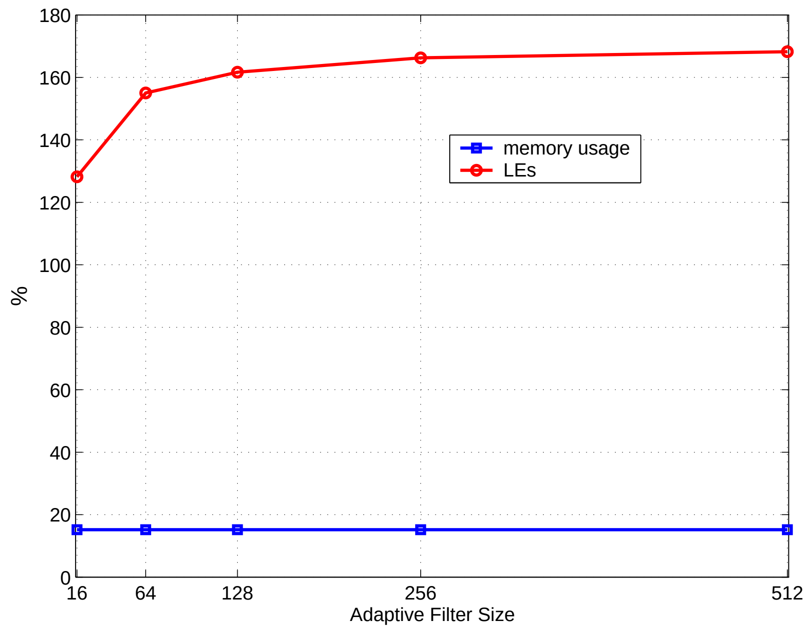


Figure 68. FPGA resource usage rate of the hybrid DA LMS architecture over the DAAF architecture at the same throughput condition. The hybrid DA LMS requires 25% **to** 65% **more LEs and** 80% **less memory than the DAAF for the same throughput condition.**

Table 15. Summary of the FPGA implementation results (Altera Stratix EP1S80F1508C6 FPGA chip) for LMS adaptive filters, ranging from 16 **to** 1024 **taps, when the** $B_c = 18$ **and** $k = 16$**. The DAAF architecture fails to synthesize a** 16**-tap base unit for all filter sizes attempted, while both hybrid LMS architectures can synthesize a** 16**-tap base unit.**

		Filter size (K)						
		16	32	64	128	256	512	1024
DAAF	LE	x	x	x	x	x	x	x
	Memory	x	x	x	x	x	x	x
LUT-less Hybrid DA-OBC	LE	1366	2438	4576	8842	17372	34422	68519
	Memory	368	736	1472	2944	5888	11776	23552
LUT-less Hybrid DA	LE	1174	2034	3770	7232	14221	28102	55864
	Memory	384	768	1536	3072	6144	12288	24576

CHAPTER 8

CONCLUSION AND FUTURE RESEARCH

New architectures for audio enhancement algorithms–ANS and AEC–are proposed, implemented, and tested for applications, in which power budget is limited. In this chapter, we will summarize our work by listing the contributions of this thesis and addressing future research directions as well as remaining work.

8.1 Summary of Contributions

The primary contributions of the thesis are summarized as follows:

- In Chapter 3, a continuous-time ANS system, based on the spectral gain modification algorithm, was developed. This architecture provides the framework for advanced analog audio signal processing by allowing programmability for each computation block. The continuous-time ANS system was fabricated on a $0.5\mu m$ CMOS VLSI chip using programmable floating-gate computing elements for ultra low-power computation.

- In Chapter 3, a noisy signal envelope and a noise envelope were estimated using a programmable peak detector and a minimum detector, respectively. The selection of the decay time constant of the peak detector and the attack time constant of the minimum detector is crucial for the overall performance of the continuous-time ANS system.

- In Chapter 3, a noise signal envelope was estimated not from band-limited input signal, but from the averaged noisy signal envelope estimate. This approach, also called the “normalized SNR method,” helps the overall SNR to be unity or very close to unity when the signal is assumed to be absent. The normalized SNR method forces the SNR of the noise-only period to unity, independent of input noise variance, and accordingly, the suppression rate will be consistently the same.

- In Chapter 3, programmable non-linear gain functions were described. The non-linear gain function should be chosen carefully since it is very crucial not only for overall noise suppression rate but also for the intelligibility of the noise suppressed output signal. Wiener gain functions with or without over-subtraction, bi-linear gain functions with different thresholds, and sigmoid gain functions with various parameters were compared.

- In Chapter 4, a tapped-delay line and a parallel delay line as well as a low-pass delay filter and a band-pass delay filter were described and compared. In addition, low-pass-to-band-pass transformation rule, with an emphasis on the preserving group delay of low-pass filters, are introduced.

- In Chapter 4, two subband delay networks are proposed. The first one is a delay network with low-pass delay elements and modulation, which is desirable for an application in which the linearity of the group delay is important. The other one is a delay network with band-pass delay elements, which is desirable for creating larger group delay.

- In Chapter 5, a hybrid DA-OBC architecture was proposed. The hybrid DA-OBC, which was based on the memory reduction of DA-OBC [20], was re-structured so that it could be applied to high-order FIR filters in conjunction with the "partial sum technique."

- In Chapter 5, a hybrid DA architecture was proposed. Unlike the conventional LUT-based DA structure and the adder-based DA structure, the hybrid DA architecture allows the use of both the LUT and adders simultaneously for the filtering operation. It was shown that the hybrid DA architecture is not only memory efficient but also silicon-area efficient.

- In Chapter 5, a reusable DA base unit and reusable DA (RDA) block was proposed. The RDA can significantly reduce the custom VLSI silicon size for matrix-vector multiplication such as DCT and DFT by multiplexing a reusable DA base unit. Even though the RDA requires more clock cycles than the typical DA, this architecture is still very attractive in terms of throughput compared to non-DA approaches.

- In Chapter 6, LUT-based DA architecture was extended to the LMS adaptive filter algorithm. This structure efficiently updates the DA filtering LUT (DA-F-LUT) by introducing auxiliary LUT (DA-A-LUT) and using an address rotation schemes. LMS adaptive filters whose sizes vary from 16 to 1024 taps were implemented and the results are analyzed.

- In Chapter 7, a hybrid DA LMS adaptive filter architecture was proposed. Implementation results of the LUT-less hybrid DA LMS shows that the hybrid DA LMS architecture is more suitable not only for higher-order adaptive filters but also for larger-size base units than the DAAF architecture.

8.2 Directions for Future Research

In this section, the remaining issues related to the current structure and future research directions are addressed.

- Continuous-time ANS using analog computing elements operating in the subthreshold range was proposed for low-power computation. Each programmable analog computation block for the continuous-time ANS was fabricated, programmed, tested, and measured individually. Thus, integrating all 32 subbands into a single VLSI chip is still challenging because of the limited silicon area. More efforts should be made to minimize the size of the computation blocks, to increase the precision of the programming, and to reduce circuit noises caused by imperfect properties and mismatches of analog circuitry.

- Continuous-time low-pass and band-pass delay elements were investigated, and delay networks based on these delay elements were presented. It is worthwhile to characterize the group delay properties of subthreshold continuous-time filters such as voltage-based C4 circuits and current-based MITE circuits.

- For adaptive filter applications, LUT-based LMS architectures and LUT-less LMS architectures have been implemented in this research. Hybrid LMS architectures are generalizations of the DA LMS architecture, enabling simultaneous usage of LUT and adders; these two implementations are only special cases of hybrid LMS architectures, maximizing memory usage or LE usage, respectively. Future research should include the implementation of hybrid LMS architectures that use both LUT and adders at the same time for the balanced usage of memory and LE and the estimation of the optimal k_L and k_A values that can maximize an implementable number of filter taps for various FPGAs.

- LMS adaptive filters with long filter taps were implemented on a reconfigurable architecture for high-speed implementation targeting AEC applications. In the real world, the actual performance of the LMS adaptive filter is severely limited by the double-talk situation. For more realistic system implementation, implementing a double-talk detector is also highly desirable.

- It is well known that the NLMS adaptive filter works better than the LMS adaptive filter in terms of convergence speed and stability. LMS adaptive filter implementation based on the efficient LUT update method using an auxiliary LUT for high-speed applications was proposed in this research. Therefore, development of new DA architectures for the NLMS adaptive filter, which can make convergence speed faster than that of LMS adaptive filter while maintaining throughput advantage of the DAAF or the hybrid DA LMS, can be a very interesting research field.

- The architecture for a highly area-optimized matrix-vector multiplication processor, e.g., the DCT processor, based on a reusable DA (RDA) was proposed. Since this architecture comes with area-throughput trade-offs, I believe that future research should include the theoretical analysis of the area-throughput trade-offs and the fabrication of custom VLSI processors.

- Transform-domain adaptive filters show better performance than the same type of adaptive filter implemented in the time-domain. Most commonly used transforms include DCT, DFT, or DWT, and they can be implemented using RDA architecture in a very area-efficient manner, as explained in Chapter 5.5.

- Subband AEC (SAEC) can work better than wideband AEC as long as corner frequencies of prototype filters for the analysis filter bank and the synthesis filter bank are carefully chosen to prevent becoming badly conditioned because of the low excitation levels near the band edges [56, 71]. These prototype low-pass filters normally have a few hundred filter taps for extremely higher stop-band attenuation. As the number of subbands increases, a number of high-order prototype filters need to be implemented. All parallel implementation of these prototype filters can't be feasible because of fast increasing area size. I believe that the RDA architecture could also provide a very interesting solution to reduce this hardware cost problem of the SAEC.

REFERENCES

[1] "Stratix device family data sheet." Altera Corporation, http://www.altera.com/literature/lit-index.html.

[2] Allred, D. J., Krishnan, V., Huang, W., and Anderson, D., "Implementation of an LMS adaptive filter on an FPGA employing multiplexed multiplier architecture," in *Proceedings of the Asilomar Conference on Signals, Systems, and Computers*, vol. 1, (Pacific Grove, CA), pp. 918–921, Nov. 2003.

[3] Allred, D. J., Yoo, H., Krishnan, V., Huang, W., and Anderson, D. V., "LMS adaptive filters using distributed arithmetic for high throughput," *IEEE Transactions on Circuits and Systems I*. Resubmitted with minor changes.

[4] Allred, D. J., Yoo, H., Krishnan, V., Huang, W., and Anderson, D. V., "A novel high performance distributed arithmetic adaptive filter implementation on an FPGA," in *Proceedings of the IEEE ICASSP*, vol. 5, (Montreal, Canada), pp. 161–164, May 2004.

[5] Ambardar, A., *Analog and digital signal processing*. Boston, MA: PWS Publishing Company, 1995.

[6] Anderson, D. V., "Model based development of a hearing aid," Master's thesis, Brigham Young University, Provo, Utah, 1994.

[7] Anderson, D. V., Hasler, P., Ellis, R., Yoo, H., Graham, D., and Hans, M., "A low-power system for audio noise suppression: A cooperative analog-digital signal processing approach," in *Proceedings of the 10th IEEE DSP workshop*, vol. 2, (Pine Mountain, GA), pp. 728–731, Oct. 2002.

[8] Arslan, L., McCree, A., and Viswanathan, V., "New methods for adaptive noise suppression," in *Proceedings of the IEEE ICASSP*, vol. 1, May 1995.

[9] Beaufays, F., "Transform-domain adaptive filters: An analytical approach," *IEEE Transactions on Signal Processing*, vol. 43, pp. 422–431, Feb. 1995.

[10] Benesty, J., Morgan, D. R., and Cho, J. H., "A new class of doubletalk detectors based on cross-correlation," *IEEE Transactions on Speech and Audio Processing*, vol. 8, pp. 168–172, Mar. 2000.

[11] Blinchikoff, H., "A note on wide-band group delay," *IEEE Trans. on Circuit Theory*, pp. 577–578, Sept. 1971.

[12] Boll, S. F., "Suppression of acoustic noise in speech using spectral subtraction," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 29, pp. 113–120, Apr. 1979.

[13] Bult, K. and Wallinga, H., "A CMOS analog continuous-time delay line with adaptive delay-time control," *IEEE Journal of Solid-State Circuits*, vol. 23, no. 3, pp. 759–766, 1988.

[14] Chang, T.-S., Chen, C., and Jen, C.-W., "New distributed arithmetic algorithm and its application to IDCT," *IEE Proceedings Circuits, Devices and Systems*, vol. 146, pp. 159–163, Aug. 1999.

[15] Chang, T.-S. and Jen, C.-W., "Hardware-efficient implementations for discrete function transforms using LUT-based FPGAs," *IEE Proceedings Circuits, Devices and Systems*, vol. 146, pp. 309–315, Nov. 1999.

[16] Chatterjee, A. and Roay, R. K., "An architectural transformation program for optimization of digital systems by multi-level decomposition," in *Proceedings of 30th ACM/IEEE Digital Automation Conference*, pp. 343–348, 1993.

[17] Chatterjee, A., Roay, R. K., and d'Abreu, M. A., "Greedy hardware optimization for linear digital circuits using number splitting and refactorization," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 1, pp. 423–431, Dec. 1993.

[18] Chen, C., Chang, T.-S., and Jen, C.-W., "The IDCT processor on the adder-based distributed arithmetic," in *1996 Symposium on VLSI Circuits Digest of Technical Papers*, pp. 36–37, 1996.

[19] Cho, J. H., Morgan, D. R., and Benesty, J., "An objective technique for evaluating doubletalk detectors in acoustic cancelers," *IEEE Transactions on Speech and Audio Processing*, vol. 7, pp. 718–724, Nov. 1999.

[20] Choi, J., Shin, S., and Chung, J., "Efficient ROM size reduction for distributed arithmetic," in *Proceedings of the IEEE ISCAS*, vol. 2, (Geneva, Switzerland), pp. 61–64, May 2000.

[21] Cowan, C. F. N. and Mavor, J., "New digital-adaptive filter implementation using distributed-arithmetic techniques," *IEE Proceedings*, vol. 128, Pt. F, pp. 225–230, Aug. 1981.

[22] Crochiere, R. E., "A weighted overlap-add method of shoft-time fourier analysis/synthesis," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP-28, pp. 99–102, Feb. 1980.

[23] Crochiere, R. E. and Rabiner, L. R., *Multirate Digital Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1983.

[24] Croisier, A., Esteban, D. J., Levilion, M. E., and Rizo, V., "Digital filter for PCM encoded signals." U.S. Patent No. 3,777,130, issued Apr, 1973.

[25] Daniels, R. W., *Approximation Methods for Electronic Filter Design*. New York, NY: McGraw-Hill, 1974.

[26] Diethorn, E. J., “A subband noise-reduction method for enhancing speech in telephony & teleconferencing,” in *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, (Mohonk Mountain House, New Paltz, NY), pp. 19–22, Oct. 1997.

[27] Ding, H., “A stable fast affine projection adaptation algorithm suitable for low-cost processors,” in *Proceedings of the IEEE ICASSP*, vol. 1, pp. 360–363, June 2000.

[28] Douglas, S. C., “Efficient approximate implementations of the fast affine projection algorithm using orthogonal transforms,” in *Proceedings of the IEEE ICASSP*, vol. 3, (Atlanta, GA), pp. 1656–1659, May 1996.

[29] Duttweiler, D. L., “A twelve-channel digital echo canceler,” *IEEE Transactions on Communications*, vol. 26, pp. 647–653, May 1978.

[30] Ellis, R., Yoo, H., Graham, D., Hasler, P., and Anderson, D. V., “Analog audio signal enhancement system using a noise suppression algorithm.” U.S. Patent Serial No. 394783, filed Mar. 21, 2003.

[31] Ellis, R., Yoo, H., Graham, D., Hasler, P., and Anderson, D. V., “A continuous-time speech enhancement front-end for microphone inputs,” in *Proceedings of the IEEE ISCAS*, vol. 2, (Phoenix, AZ), pp. 728–731, May 2002.

[32] Ephraim, Y. and Malah, D., “Speech enhancement using a minimum mean-square error short-time spectral amplitude estimator,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP-32, pp. 1109–1121, Dec. 1984.

[33] Ephraim, Y., Malah, D., and Juang, B. H., “On the application of hidden Markov models for enhancing noisy speech,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP-37, pp. 1846–1856, Dec. 1989.

[34] Etter, W. and Moschytz, G. S., “Noise reduction by noise adaptive spectral magnitude expansion,” *Journal of the Audio Engineering Society*, vol. 42, May 1994.

[35] Farhang-Boroujeny, B., *Adaptive Filters: Theory and Applications*. Chichester, England: John Wiley and Sons, 1998.

[36] Farhang-Boroujeny, B., Lee, Y., and Ko, C. C., “Sliding transforms for efficient implementation of transform domain adaptive filters,” *Signal Processing*, vol. 52, pp. 83–96, 1996.

[37] Frantz, G., “Digital signal processor trends,” *IEEE Micro*, pp. 52–59, Nov. 2000.

[38] Furui, S. and Sondhi, M. M., *Advances in speech signal processing*. New York, NY: Marcel Dekker, 1991.

[39] Gänsler, T., Hansson, M., Ivarsson, C. J., and Salomonsson, G., “A double-talk detector based on coherence,” *IEEE Transactions on Communications*, vol. 44, pp. 1421–1427, Nov. 1996.

[40] Gay, S. L., “A fast converging, low complexity adaptive filtering algorithm,” in *Proceedings of 3rd International Workshop on Acoustic Echo Control*, pp. 223–226, Sept. 1993.

[41] Gay, S. L., “The fast affine projection algorithm,” in *Proceedings of the IEEE ICASSP*, pp. 3023–3026, 1995.

[42] Gay, S. L. and Benesty, J., *Acoustic Signal Processing for Telecommunication*. Boston, MA: Kluwer Academic Publishers, 2000.

[43] Geffe, P. R., “On the approximation problem for band-pass delay lines,” in *Proceeding of IRE*, pp. 1986–1987, Sept. 1962.

[44] Graham, D., *Continuous–time bandpass second–order sections and their applications in cochlea modeling*. Atlanta, GA: MS thesis, Georgia Institute of Technology, 2003.

[45] Graham, D. and Hasler, P., “Capacitively-coupled current conveyer second-order section for continuous-time bandpass filtering and cochlea modeling,” in *Proceedings of the IEEE ISCAS*, vol. 5, (Phoenix, AZ), pp. 485–488, May 2002.

[46] Graham, D., Smith, P., Ellis, R., Chawla, R., and Hasler, P., “A programmable bandpass array using floating gates,” in *Proceedings of the IEEE ISCAS*, vol. 1, (Vancouver, British Columbia), pp. 97–100, May 2004.

[47] Hartley, R. I., “Subexpression sharing in filters using canonic signed digit multipliers,” *IEEE Transactions on Circuits and Systems II*, vol. 43, pp. 677–688, Oct. 1996.

[48] Hasler, P., Diorio, C., Minch, B. A., and Mead, C., *Advances in Neural Information Processing Systems 7*, ch. Single transistor learning synapses, pp. 817–824. Cambridge, MA: MIT Press, 1995.

[49] Hasler, P., Kucic, M., and Minch, B. A., “A transistor–only circuit model of the autozeroing floating–gate amplifier,” in *Proceedings of the IEEE Midwest Symposium on Circuits and Systems*, (Las Cruces), pp. 157–160, 1999.

[50] Hasler, P. and Lande, T. S., “Overview of floating-gate devices, circuits, and systems,” *IEEE Transactions on Circuits and Systems II*, vol. 48, pp. 1–3, Jan. 2001.

[51] Hasler, P., Minch, B. A., Dugger, J., and Diorio, C., “Adaptive circuits and synapses using pFET floating-gate devices,” in *Learning in Silicon* (Cauwenbergs, G., ed.), pp. 33–65, Kluwer Academic, 1999.

[52] Hayes, M. H., *Statistical digital signal processing and modeling*. New York, NY: John Wiley and Sons, 1996.

[53] Haykin, S., *Adaptive Filter Theory*. Upper Saddle River, NJ: Prentice Hall, 1996.

[54] Huang, W., Krishnan, V., Allred, D. J., and Yoo, H., "Design analysis of a distributed arithmetic adaptive FIR filter on an FPGA," in *Proceedings of the Asilomar Conference on Signals, Systems, and Computers*, vol. 1, (Pacific Grove, CA), pp. 926–930, Nov. 2003.

[55] Hwang, S., Han, G., Kang, S., and Kim, J., "New distributed arithmetic algorithm for low-power FIR filter implementation," *IEEE Signal Processing Letters*, vol. 11, pp. 463–466, May 2004.

[56] II, P. L. D. L. and Etter, D. M., "Experimental results with increased bandwidth analysis filters in oversampled, subband acoustic echo cancellers," *IEEE Signal Processing Letters*, vol. 2, no. 1, pp. 1–3, 1995.

[57] Jeannès, R. L. B., Scalart, P., Faucon, G., and c. Beaugeant, "Combined noise and echo reduction in hands-free systems: A survey," *IEEE Transactions on Speech and Audio Processing*, vol. 9, pp. 808–820, Nov. 2001.

[58] Juan, J., Harris, J. G., and Principe, J. C., "Analog hardware implementation of adaptive filter structures," in *Proceedings of the International Joint Conference on Neural Networks*, vol. 2, pp. 916–921, June 1997.

[59] Ko, U., Balsara, P. T., and Lee, W., "Low-power design techniques for high-performance CMOS adders," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 3, pp. 327–333, June 1995.

[60] Kucic, M., Low, A., Hasler, P., and Neff, J., "A programmable continuous-time analog fourier processor," *IEEE Transactions on Circuits and Systems II*, vol. 48, pp. 90–99, Jan. 2001.

[61] Laker, K. R., Ganesan, A., and Fleischer, P. E., "Design and implementation of cascaded switched-capacitor delay equalizers," *IEEE Transactions on Circuits and Systems I*, vol. 32, pp. 700–711, July 1985.

[62] Lee, J. C. and Un, C. K., "Performance of transform domain LMS adaptive algorithms," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP-34, pp. 499–510, June 1986.

[63] Lie, Q. G., Champagne, B., and Ho, K. C., "On the use of a modified fast affine projection algorithm in subbands for acoustic echo cancellation," in *Proceedings of the 10th IEEE DSP workshop*, pp. 354–357, 1996.

[64] Lie, S. and Mead, C. A., "Continuous-time adaptive delay system," *IEEE Transactions on Circuits and Systems II*, vol. 43, pp. 744–751, Nov. 1996.

[65] Martin, R., "Noise power spectral density estimation based on optimal smoothing and minimum statistics," *IEEE Transactions on Speech and Audio Processing*, vol. 9, pp. 504–512, July 2001.

[66] McAulay, R. J. and Malpass, M. L., “Speech enhancement using a soft-decision noise suppression filter,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 28, pp. 137–145, Apr. 1980.

[67] Mcdonald, E. J. and Minch, B. A., “Synthesis of a translinear analog adaptive filter,” in *Proceedings of the IEEE ICASSP*, (Orlando, FL), pp. 321–324, May 2002.

[68] Mead, C. A., *Analog VLSI and Neural Systems*. Reading, MA: Addison-Wesley, 1989.

[69] Minch, B. A., “Multiple-input translinear-element log-domain filters,” *IEEE Transactions on Circuits and Systems II*, vol. 48, pp. 29–36, Jan. 2001.

[70] Minch, P. H. B. A. and Diorio, C., “An autozeroing floating-gate bandpass filter,” in *Proceedings of the IEEE ISCAS*, (Monterey, CA), pp. 131–134, 1998.

[71] Morgan, D. R., “Slow asymptotic convergence of LMS acoustic echo cancellers,” *IEEE Transactions on Speech and Audio Processing*, vol. 3, no. 2, pp. 126–136, 1995.

[72] Morgan, D. R. and Thi, J. C., “A delayless subband adaptive filter architecture,” *IEEE Transactions on Signal Processing*, vol. 43, pp. 1819–1830, Aug. 1995.

[73] Narayan, S. S., Peterson, A. M., and Narasimha, M. J., “Transform domain LMS algorithm,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP-31, pp. 609–615, June 1983.

[74] Ochiai, K., Araseki, T., and Ogihara, T., “Echo canceler with two echo path models,” *IEEE Transactions on Communications*, vol. COM-25, pp. 589–595, 1977.

[75] Ozeki, K. and Umeda, T., “An adaptive filtering algorithm using an orthogonal projection to an affine subspace and its propoerties,” *Electronics and Communications in Japan*, vol. 67-A, no. 5, pp. 19–27, 1984.

[76] Park, S. J., Cho, C. G., Lee, C., and Youn, D. H., “Integrated echo and noise canceler for hands-free applications,” *IEEE Transactions on Circuits and Systems II*, vol. 49, pp. 188–194, Mar. 2002.

[77] Park, S. J., Cho, C. G., Lee, C., and Youn, D. H., “Integrated echo and noise canceler for hands-free applications,” *IEEE Transactions on Circuits and Systems II*, vol. 49, pp. 188–195, Mar. 2002.

[78] Peled, A. and Lie, B., “A new hardware realization of digital filters,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 22, pp. 456–462, Dec. 1974.

[79] Rabaey, J. M., Chandrakasan, A., and Nilolic, B., *Digital Integrated Circuits: A Design Perspective*. Upper Saddle River, NJ: Prentice Hall, 2002.

[80] Sameti, H., Sheikhzadeh, H., and Deng, L., "HMM-based strategies for enhancement of speech signals embedded in nonstationary noise," *IEEE Transactions on Speech and Audio Processing*, vol. 6, pp. 445–455, Sept. 1998.

[81] Sarpeshkar, R., *Efficient precise computation with noisy components: extrapolating from an electronic cochlea to the brain*. Pasadena, CA: PhD thesis, California Institute of Technology, 1997.

[82] Schroeder, M. R., "Apparatus for suppressing noise and distortion in communication signals." U.S. Patent No. 3,180,936, issued Apr. 27, 1965.

[83] Schroeder, M. R., "Processing of communications signals to reduce effects of noise." U.S. Patent No. 3,403,224, issued Sept. 24, 1968.

[84] Serrano-Gotarredona, T., Linares-Barranco, B., and Andreou, A. G., "A general translinear principle for subthreshold MOS transistors," *IEEE Transactions on Circuits and Systems I*, vol. 46, pp. 607–616, May 1999.

[85] Shlien, S., "The modulated lapped transform, its time-varying forms, and its application to audio coding standards," *IEEE Transactions on Speech and Audio Processing*, vol. 5, no. 4, pp. 359–366, 1997.

[86] Sim, B. L., Tong, Y. C., and Chang, J. S., "A parametric formulation of the generalized spectral subtraction method," *IEEE Transactions on Speech and Audio Processing*, vol. 6, pp. 328–337, July 1998.

[87] Smith, P., Chawla, R., Graham, D., and Hasler, P., "A five–transistor bandpass filter element," in *Proceedings of the IEEE ISCAS*, vol. 1, (Vancouver, British Columbia), pp. 861–864, May 2004.

[88] Smith, P., Kucic, M., Ellis, R., Hasler, P., and Anderson, D. V., "Mel-frequency cepstrum encoding in analog floating-gate circuitry," in *Proceedings of the IEEE ISCAS*, vol. IV, (Phoeniz, AZ), pp. 671–674, May 2002.

[89] Smith, P., Kucic, M., and Hasler, P., "Accurate programming of analog floating–gate arrays," in *Proceedings of the IEEE ISCAS*, vol. 5, (Scottsdale, AZ), pp. 489–492, May 2002.

[90] Sondhi, M. M., "An adaptive echo canceler," *Bell System Technical Journal*, vol. 46, pp. 497–511, 1967.

[91] Sondhi, M. M. and Presti, A. J., "A self-adaptive echo canceler," *Bell System Technical Journal*, vol. 45, pp. 1851–1854, 1966.

[92] Szentirmai, G., "The design of arithmetically symmetrical band-pass filters," *IEEE Trans. on Circuit Theory*, pp. 367–375, Sept. 1963.

[93] Tanaka, M., Makino, S., and Kojima, J., "A block exact fast affine projection algorithm," *IEEE Transactions on Speech and Audio Processing*, vol. 7, pp. 79–86, Jan. 1999.

[94] Trees, H. L. V., *Detection, Estimation, and Modulation Theory, part I*. New York, NY: Wiley, 1968.

[95] Wei, C. H. and Lou, J. J., “Multi memory block structure for implementing a digital adaptive filter using distributed arithmetic,” *IEE Proceedings*, vol. 133, Pt. G, pp. 19–26, Feb. 1986.

[96] White, S. A., “Applications of distributed arithmetic to digital signal processing: A tutorial review,” *IEEE ASSP Magazine*, vol. 6, pp. 4–19, July 1989.

[97] Widrow, B. and Stearns, S. D., *Adaptive Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall Inc., 1985.

[98] Ye, H. and Wu, B. X., “A new double-talk detection algorithm based on the orthogonality theorem,” *IEEE Transactions on Communications*, vol. 39, pp. 1542–1545, Nov. 1991.

[99] Yoo, H. and Anderson, D. V., “Hardware-efficient distributed arithmetic architecture for high-order digital filters,” in *Proceedings of the IEEE ICASSP*, (Philadelphia, PA), May 2005. to be published.

[100] Yoo, H., Anderson, D. V., and Hasler, P., “Continuous-time audio noise suppression and a custom low-power IC implementation,” in *Proceedings of the IEEE ICASSP*, (Orlando, FL), pp. 3980–3983, May 2002.

[101] Yoo, H., Anderson, D. V., and Hasler, P., “On delay structures for the analog adaptive filters with long filter taps,” in *Proceedings of the Asilomar Conference on Signals, Systems, and Computers*, vol. 2, (Pacific Grove, CA), pp. 2021–2025, Nov. 2003.

[102] Yoo, H., Graham, D., Anderson, D. V., and Hasler, P., “C^4 band-pass delay filter for continuous-time subband adaptive tapped-delay filter,” in *Proceedings of the IEEE ISCAS*, vol. 5, (Vancouver, Canada), pp. 792–795, May 2004.

[103] Yu, S. and Swartzlander, E. E., “DCT implementation with distributed arithmetic,” *IEEE Transactions on Computers*, vol. 50, pp. 985–991, Sept. 2001.