Simplified Logic for First-Order and Second-Order Mismatch-Shaping Digital-to-Analog Converters

Jared Welz, *Student Member, IEEE*, Ian Galton, *Member, IEEE*, and Eric Fogleman, *Member, IEEE*

***Abstract*—Mismatch-shaping digital-to-analog converters (DACs) have become widely used in high-performance delta–sigma data converters because they facilitate delta–sigma modulators with multibit quantization. Relative to single-bit quantization, multibit quantization significantly relaxes the analog circuit performance necessary to achieve a given level of data converter precision, but significant digital logic is required to perform the mismatch shaping. In modern very large scale integration processes optimized for digital circuitry, this tends to be a good tradeoff in terms of both area and power consumption. It is nonetheless desirable to minimize the digital complexity as much as possible. Moreover, in delta–sigma analog-to-digital converters the mismatch-shaping logic is in the feedback path of the delta–sigma modulator, so it is essential to maintain a sufficiently small propagation delay through the mismatch-shaping logic. This paper presents and analyzes several variations of the switching blocks within a tree-structured mismatch-shaping DAC that result in the most hardware-efficient first-order and second-order mismatch-shaping DAC implementations yet known to the authors. The variations presented allow designers to tradeoff complexity for propagation-delay reduction so as to tailor designs to specific applications.**

***Index Terms*—Analog-to-digital conversion, delta–sigma modultation, digital-to-analog conversion, mismatch shaping, tree-structured digital-to-analog converter.**

I. Introduction

IN $\Delta\Sigma$ DATA CONVERTERS, both $\Delta\Sigma$ analog-to-digital converters (ADCs) and $\Delta\Sigma$ digital-to-analog converters (DACs), coarse quantization is used in conjunction with quantization-noise shaping and filtering to achieve high-precision data conversion. In both cases, coarse DACs are required. Unlike the error introduced by the coarse quantization, the error introduced by at least one of the coarse DACs in a $\Delta\Sigma$ data converter is not attenuated inside the data converter's signal band. In switched-capacitor implementations, most of the DAC error arises from static capacitor mismatches, which give rise to step-size mismatches in the multibit DAC. The resulting step-size mismatches are memoryless functions of the DAC input, so the DAC can be viewed as an ideal DAC followed by a memoryless nonlinear function. The nonlinearity tends to fold out-of-band quantization noise into the signal band, thereby limiting the overall accuracy of the data converter.

To avoid this problem, many $\Delta\Sigma$ data converters employ 1-bit quantization. With 1-bit quantization, the coarse DAC is implemented by a 1-bit DAC. Since a 1-bit DAC only generates two levels, it only has one step, and so it is inherently linear. However, with 1-bit quantization in the $\Delta\Sigma$ modulator, quantization-noise shaping must be limited to maintain the $\Delta\Sigma$ modulator's stability. Additionally, the power of the quantization noise in the 1-bit $\Delta\Sigma$ modulator exceeds that of its input, so $\Delta\Sigma$ data converters with 1-bit quantization are extremely sensitive to any nonlinearity or timing error, such as op-amp slewing or clock jitter, which can fold this quantization noise into the signal band.

To avoid these problems, multibit *mismatch-shaping* DACs have been developed [1]–[52]. In these DACs, digital logic is used to scramble the DAC capacitor or current-source connections in such a fashion that the error introduced by the device mismatches, referred to as *DAC noise*, is suppressed within the data converter's signal band. For low-pass mismatch-shaping DACs, the DAC noise is suppressed near dc so that its power spectral density (PSD) is shaped like the magnitude response of a first-order, or in some cases, second-order high-pass filter. The five main classes of mismatch-shaping DACs include individual-level averaging (ILA) [11], [12], vector feedback [13]–[16], data-weighted averaging (DWA) [17]–[31], butterfly shuffler [32]–[37], and tree structured [38]–[44]. The criteria used to compare these DACs include complexity, propagation delay, spurious-tone avoidance, and the order, or degree, of DAC noise suppression.

In [40], a tree-structured mismatch-shaping DACs is introduced that has led to the most efficient implementations of dithered first-order and second-order mismatch-shaping DACs known to the authors [43], [44]. Moreover, the first-order tree-structured DAC is the only one for which dither is known to completely eliminate spurious tones in its DAC noise [46]. This paper furthers the development of this DAC by presenting new implementations of its digital logic that are more hardware efficient and have less propagation delay than those presented in [40]. The digital logic is first partitioned into functional blocks, one of which determines the shape of the DAC noise's PSD and another that is responsible for the digital logic's propagation delay. The hardware for the digital logic is presented through interchangeable variations of these functional blocks so that the DAC can be tailored to meet varying specifications for signal-band DAC noise power, propagation delay, and complexity. Efficient first and second-order mismatch-shaping logic are presented and the resulting DAC noise from each is

Manuscript received June 30, 2000; revised October 23, 2001. This work was supported by the Office of Naval Research under Grant N00014-98-1-0830. This paper was recommended by Associate Editor R. Shi.

J. Welz and I. Galton are with the Department of Electrical and Computer Engineering, University of California at San Diego, La Jolla, CA 92093 USA (e-mail: jwelz@ece.ucsd.edu; galton@ece.ucsd.edu).

E. Fogleman was with the University of California at San Diego, La Jolla, CA 92093 USA. He is now with Silicon Wave Inc., San Diego, CA 92122 USA (e-mail: fogleman@ieee.org).

Publisher Item Identifier S 1057-7130(01)11308-X.

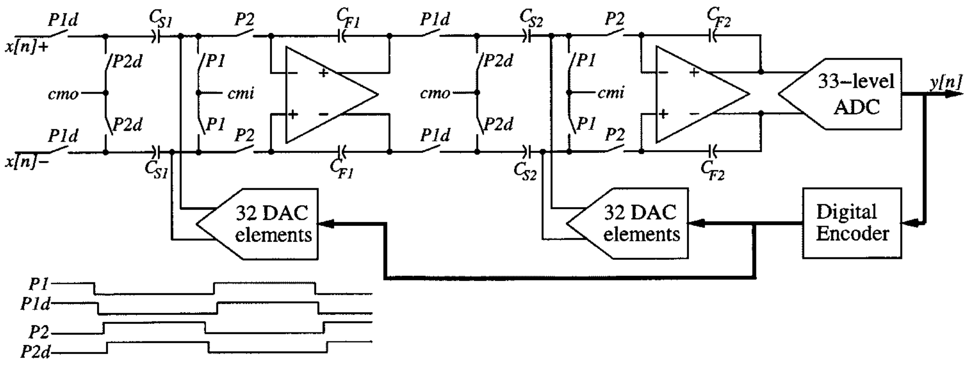


Fig. 1. An example of a second-order, 33-level, low-pass analog $\Delta\Sigma$ modulator realized with switched capacitors.

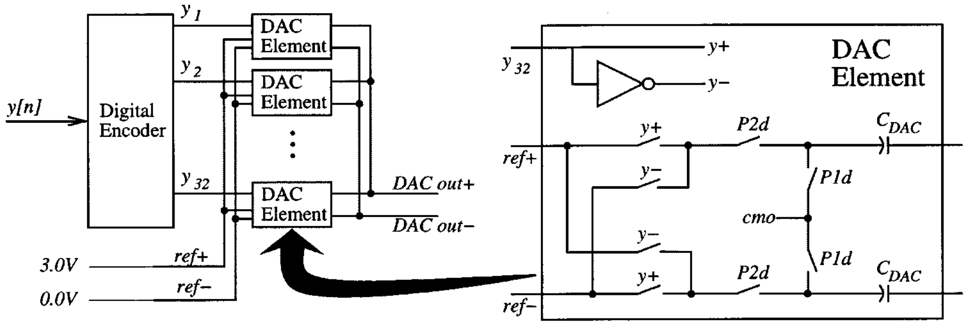


Fig. 2. A 33-level mismatch-shaping DAC with switched capacitor DAC elements.

analyzed to show it has the desired spectral shape. Additionally, medium-speed and high-speed implementations of the DAC are presented that offer a tradeoff between propagation delay and complexity.

This paper is divided into six sections. Section II reviews the tree-structured DAC and presents the functional partitioning of its digital logic. Additionally, this section presents an example application of a 5-bit second-order ADC $\Delta\Sigma$ modulator that is used throughout the paper to illustrate the DAC performance and complexity. Section III presents and analyzes the first-order and second-order mismatch-shaping logic, while Section IV presents the medium-speed and high-speed implementations of the DAC. Section V presents a hardware comparison between the different tree-structured DAC implementations and other mismatch-shaping DACs presented in literature.

II. The Tree-Structured DAC

A. The $\Delta\Sigma$ Modulator Application

The 5-bit ADC $\Delta\Sigma$ modulator presented in [43] is shown in Fig. 1. It consists of two delayed switched-capacitor integrators, a 33-level flash ADC, and two 33-level DACs. As shown in Figs. 1 and 2, each 33-level DAC consists of a bank of 32 *DAC elements* and a shared digital encoder whose outputs, $y_i[n]$ $(i = 1, \ldots, 32)$, are 1-bit sequences. Each DAC element can be viewed as a 1-bit DAC whose analog output is a charge packet applied to the summing node of an integrator. A DAC element is said to be "selected high" when its input is high; otherwise, it is said to be "selected low." For convenience, the output of the ADC, $y[n]$, is interpreted as an integer between 0 and 32. For each ADC output sample, the digital encoder chooses which $y[n]$ of the DAC elements to select high and which $(32 - y[n])$ of the DAC elements to select low. In other words, if $y_i[n]$ is interpreted numerically as one when high and zero when low, the DAC encoder ensures that $y[n] = y_1[n] + \cdots + y_{32}[n]$.

Mismatches among the capacitor values of the DAC elements cause the output of each multibit DAC to be a nonlinear function of its input. The resulting nonlinear error is represented, without approximation, as an additive noise source referred to as *DAC noise*. As shown in Fig. 1, an output from one of the DACs is added to the $\Delta\Sigma$ modulator's input. Thus, the $\Delta\Sigma$ modulator does not attenuate any of the signal-band noise power from this DAC. However, the digital encoder can select the DAC elements such that most of the DAC noise power resides outside of the signal band.

To demonstrate the improvements that are realized by mismatch shaping, the DAC presented in [44] was tested with and without mismatch shaping. The input for each test was a 1.5 kHz, -1 dB (relative to full scale) sinusoid. With mismatch shaping, the resultant signal-to-noise-and-distortion ratio (SINAD) was 100 dB, whereas without mismatch shaping, the resultant SINAD was 64 dB. In general, the tradeoff for the improved performance is the additional hardware and propagation delay incurred by the digital encoder. However, the propagation delay of the digital encoder only affects the design of high-speed $\Delta\Sigma$ data converters. Examples of commercially available data converters that employ mismatch-shaping DACs to a similar advantage are presented in [47]–[52].

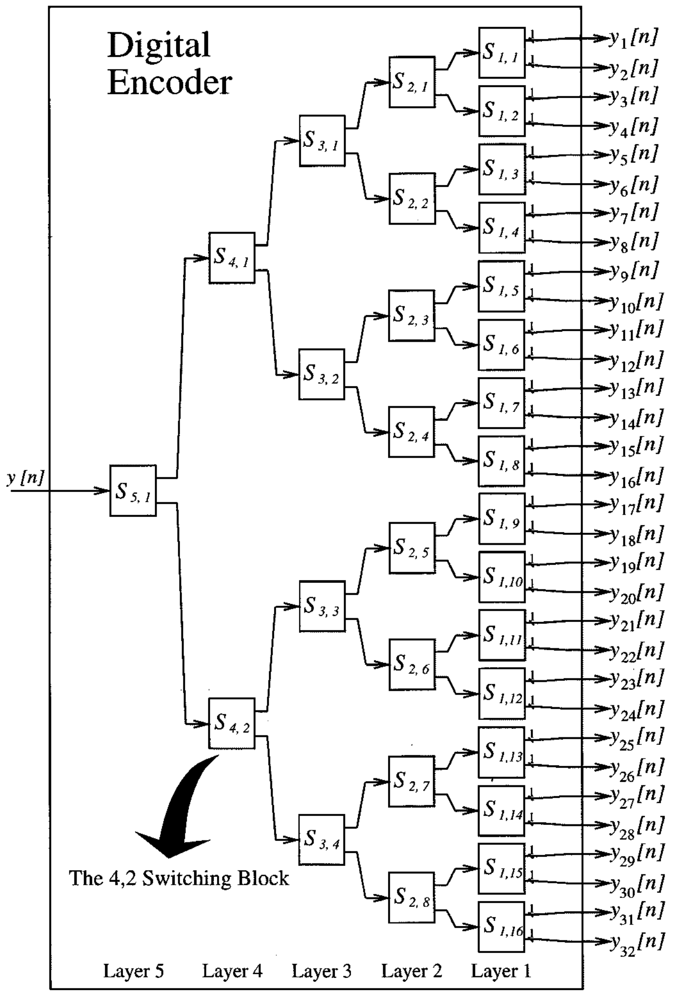


Fig. 3. The 33-level tree-structured digital encoder.

B. The Tree-Structured Digital Encoder

The architecture for a 33-level, tree-structured digital encoder is shown in Fig. 3. The nodes of this digital encoder are called *switching blocks*. Each switching block is labeled $S_{k,r}$, where k and r represent the switching block's layer number and position within the layer, respectively. Each switching block $S_{k,r}$ has a single input, which is denoted $x_{k,r}[n]$, and two outputs. If each digital encoder output sequence $y_i[n]$ is also denoted $x_{0,i}[n]$, then the switching blocks are interconnected such that the top output of $S_{k,r}$ is $x_{k-1,2r-1}[n]$ and the bottom output is $x_{k-1,2r}[n]$. The *switching sequence* $s_{k,r}[n]$ is defined as the difference between the top and bottom output sequences of $S_{k,r}$

$$s_{k,r}[n] = x_{k-1,2r-1}[n] - x_{k-1,2r}[n]. \tag{1}$$

Fig. 4 illustrates the input and output sequences of $S_{k,r}$ along with the relationship between its switching sequence and output sequences.

As shown in [40], the DAC noise is a linear combination of the switching sequences. In general, for a DAC of the type shown in Fig. 3 with 2^b DAC elements, the output can be written as

$$u[n] = \gamma y[n] + \beta + e[n] \tag{2}$$

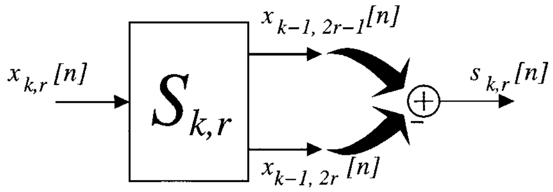


Fig. 4. The switching block $S_{k,r}$.

where

$$e[n] = \sum_{k=1}^{b} \sum_{r=1}^{2^{b-k}} \Delta_{k,r} s_{k,r}[n] \tag{3}$$

and γ, β, and $\Delta_{k,r}$ are constants that are functions of the inevitable, static errors that result from process variations during VLSI circuit fabrication.

Therefore, if the switching sequences are all uncorrelated and share the same characteristics in their PSDs (e.g., first-order high-pass shaping), the DAC noise also possesses these characteristics. The problem of shaping the PSD of the DAC noise reduces to the problem of creating switching sequences with the desired spectral shaping. Unfortunately, this problem is complicated by the constraints on the switching sequence described next.

C. Constraints on the Switching Sequence

The switching sequence is generated within the switching block to obtain the desired spectral properties of the DAC noise. However, the switching sequences must be constrained to satisfy restrictions inherent to the digital encoder. As previously described, each of the digital encoder's outputs, $y_i[n]$ $(i = 1, \ldots, 32)$, is limited to the set $\{0, 1\}$ and their sum must equal the DAC input: $y[n] = y_1[n] + \cdots + y_{32}[n]$. It is shown in [40] that these conditions are met if each switching block satisfies the following two-part *Number Conservation Rule:* the two outputs of each switching block must be in the range $\{0, 1, \ldots, 2^{k-1}\}$ where k is the layer number, and their sum must equal the input to the switching block

$$x_{k-1,2r-1}[n] + x_{k-1,2r}[n] = x_{k,r}[n]. \tag{4}$$

From (1) and (4), the input/output relationships of switching block $S_{k,r}$ are

$$x_{k-1,2r-1}[n] = \frac{1}{2}(x_{k,r}[n] + s_{k,r}[n])$$

and

$$x_{k-1,2r}[n] = \frac{1}{2}(x_{k,r}[n] - s_{k,r}[n]). \tag{5}$$

The above expressions are implemented by the block diagram shown in Fig. 5.

It can be shown that the number conservation rule is satisfied by each switching block $S_{k,r}$ if

$$s_{k,r}[n] = \begin{cases} 0, & \text{if } x_{k,r}[n] \text{ is even} \\ \pm 1, & \text{if } x_{k,r}[n] \text{ is odd.} \end{cases} \tag{6}$$

This is more restrictive than necessary; however, it significantly simplifies the switching block's hardware. In Fig. 5, this restriction is reflected by the switching sequence generator's dependence on the input sequence $x_{k,r}[n]$.

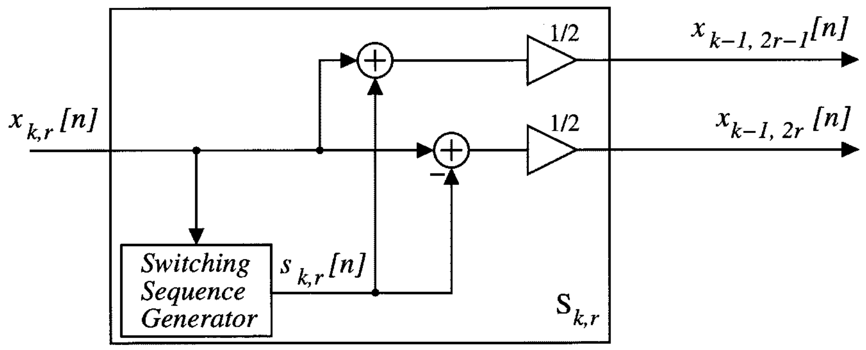


Fig. 5. The signal processing performed in the switching block.

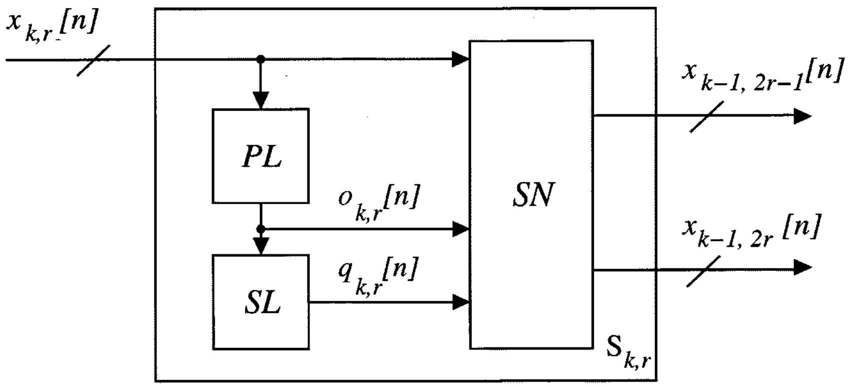


Fig. 6. A functional partitioning of the switching block, where SN is the *splitting network*, PL is the *parity logic*, and SL is the *sequencing logic*.

D. Implementation of the Switching Block

The switching sequence $s_{k,r}[n]$ is a ternary sequence, and so it can be represented as two single-bit sequences. It follows from (6) that the magnitude of the switching sequence is entirely determined by the input to the switching block, so the switching block can only control the sign of the switching sequence. To separate the magnitude and sign of the switching sequence, let $o_{k,r}[n]$ and $q_{k,r}[n]$ represent $s_{k,r}[n]$ as

$$s_{k,r}[n] = \begin{cases} 0, & \text{if } o_{k,r}[n] = 0 \\ 1, & \text{if } o_{k,r}[n] = 1,\ q_{k,r}[n] = 1 \\ -1, & \text{if } o_{k,r}[n] = 1,\ q_{k,r}[n] = 0 \end{cases} \tag{7}$$

where

$$o_{k,r}[n] = \begin{cases} 1, & \text{if } x_{k,r}[n] \text{ is odd} \\ 0, & \text{if } x_{k,r}[n] \text{ is even.} \end{cases} \tag{8}$$

The sequence $q_{k,r}[n]$ represents the sign of $s_{k,r}[n]$. It is chosen by the switching block to ensure the switching sequence is appropriately shaped as described in Section III. The $o_{k,r}[n]$ sequence is referred to as the *parity sequence* and represents the magnitude of $s_{k,r}[n]$.

Fig. 6 displays a convenient functional partitioning of the switching block. The parity logic determines the parity of the switching block's input and generates the parity sequence $o_{k,r}[n]$. The sequencing logic produces the sign sequence $q_{k,r}[n]$ and is responsible for the spectral shaping of the switching sequence. The combination of the sequencing logic and parity logic constitute the switching sequence generator shown in Fig. 5. Given $x_{k,r}[n]$ and the binary representation of $s_{k,r}[n]$ (i.e., $o_{k,r}[n]$ and $q_{k,r}[n]$), the role of the splitting network is to perform the arithmetic operations shown in Fig. 5 that generate the switching block's two output sequences.

III. Low-Pass Sequencing Logic

A. High-Pass Switching Sequences

In low-pass mismatch-shaping DACs, the signal band is near dc, so the mismatch-shaping logic is designed such that most of the DAC noise power resides at higher frequencies. In other words, in a low-pass mismatch-shaping DAC, the PSD of the DAC noise resembles the magnitude response of a discrete-time high-pass filter. Sequences of this type are called *high-pass sequences*. Thus, the sequencing logic blocks in a low-pass tree-structured DAC create high-pass switching sequences.

To meaningfully characterize the spectral properties of the high-pass switching sequences, a quantitative definition of an Lth-order high-pass switching sequence is required. In a $\Delta\Sigma$ modulator with a quantization-noise transfer function that contains zeros only at dc, the order of the $\Delta\Sigma$ modulator corresponds to the number of dc zeros. Let *quantization noise* denote the component of the $\Delta\Sigma$ modulator output arising from the errors induced by quantization. In an Lth-order low-pass $\Delta\Sigma$ modulator, the quantization noise is commonly called Lth-order high-pass noise. A key property of this high-pass noise is that it can be processed by L cascaded accumulators such that the values in the accumulators remain bounded. The L dc poles from the accumulators "cancel" the L dc zeros in the noise transfer function. However, if one more accumulator were cascaded, its output would become unbounded regardless of the accumulators' initial values.

In contrast to the quantization noise, the switching sequence, as a result of its constraints in (6), cannot be generated by filtering a causal, bounded sequence by a system with L dc zeros. Therefore, the concept of the switching sequence's order is vague without a more applicable definition. By defining the high-pass order of a switching sequence using the accumulator property described above, a transfer function is associated with this sequence, and the desired properties of its PSD are implied.

Definition: Let $\alpha_L[n]$ be the "Lth-sum sequence" of $s_{k,r}[n]$

$$\alpha_L[n] \equiv \overbrace{\sum_{n_1=0}^{n-1}\sum_{n_2=0}^{n_1-1}\cdots\sum_{n_L=0}^{n_{L-1}-1}}^{L \text{ Summations}} s_{k,r}[n_L]. \tag{9}$$

The sequence $s_{k,r}[n]$ is an Lth-order high-pass switching sequence if its Lth-sum sequence is a *bounded sequence*—i.e., there exists a number $K < \infty$ such that $|\alpha_L[n]| < K$ for all n—, and its $(L+1)$st-sum sequence is an unbounded sequence.

If $s_{k,r}[n]$ is an Lth-order high-pass switching sequence, then it can be shown that the slope of its PSD is $20L$ dB/decade near dc provided the PSD of $\alpha_L[n]$ is continuous and nonzero in a neighborhood of dc. This definition provides a means to create switching sequences that are Lth-order high-pass shaped and conform to (6).

B. First-Order Low-Pass Sequencing Logic

To produce a switching sequence $s_{k,r}[n]$ that is a first-order high-pass switching sequence, the switching block ensures that its partial sum, $\alpha_1[n]$, is a bounded sequence. Suppose the input to switching block $S_{k,r}$ is always odd and thus, from (6), $s_{k,r}[n] = \pm 1$ for all n. One method for ensuring that $\alpha_1[n]$ is a bounded sequence is by choosing $s_{k,r}[n]$ to be the alternating sequence: $s_{k,r}[n] = (-1)^n = \cos(\pi n)$. With this switching sequence, the resulting partial sum sequence is bounded in magnitude by 1

$$|\alpha_1[n]| = \left|\sum_{m=0}^{n-1} s_{k,r}[m]\right| \le 1 \tag{10}$$

and the resulting switching sequence is a single tone of normalized frequency $\omega = \pi$. For many applications, it is desirable to have DAC noise and, thus, switching sequences that do not contain any tones.

One way to eliminate tones in this scenario and yet obtain a first-order high-pass switching sequence is to construct $s_{k,r}[n]$ by randomly choosing between the following two types of *symbols*: "$1, -1$" and "$-1, 1$". When n is even (i.e., $n = 2m$), one of the two symbols is chosen randomly by a fair coin toss, and the chosen symbol is placed in the switching sequence. With this construction, the switching sequence can be written as $s_{k,r}[2m] = \pm 1$ and $s_{k,r}[2m+1] = -s_{k,r}[2m]$. The *alternating property*—i.e., $s_{k,r}[2m+1] = -s_{k,r}[2m]$—ensures that the partial sum sequence satisfies (10), while the random symbol type selection prevents $s_{k,r}[n]$ from containing any periodicities. Therefore, the resulting switching sequence is a first-order high-pass switching sequence that does not contain tones.

This method of using symbols to construct the switching sequence can be generalized to include even inputs to the switching block. When the switching block's input is even, it follows from (6) that the switching block has no choice but to force the switching sequence to be zero. To include potential zero runs in the switching sequence, the two symbols described above are generalized to be

$$1, \underbrace{0, \ldots, 0}_{\substack{\text{Until next}\\ \text{odd } x_{k,r}[n]}}, -1, \underbrace{0, \ldots, 0}_{\substack{\text{Until next}\\ \text{odd } x_{k,r}[n]}} \text{ and } -1, \underbrace{0, \ldots, 0}_{\substack{\text{Until next}\\ \text{odd } x_{k,r}[n]}}, 1, \underbrace{0, \ldots, 0}_{\substack{\text{Until next}\\ \text{odd } x_{k,r}[n]}}. \tag{11}$$

Each symbol begins in the switching sequence with a nonzero value that corresponds to an odd switching block input. The only other nonzero *element* within a symbol has the alternate sign of the first element. For a switching sequence $s_{k,r}[n]$ composed of these symbols, this alternating property ensures that its partial sum satisfies (10), which implies that the resulting switching sequence is a first-order high-pass switching sequence. Additionally, by randomly choosing between the two symbol types, the resulting switching sequence cannot contain tones.

As an example, consider the following segment of the input sequence to the switching block $S_{k,r}$:

$$x_{k,r}[n] = \ldots, 1, 2, 2, 1, 0, 1, 1, 2, \ldots$$

where the segment starts with the value "1" and ends with the value "2". The parity sequence $o_{k,r}[n]$ for this input is

$$o_{k,r}[n] = \ldots, 1, 0, 0, 1, 0, 1, 1, 0, \ldots.$$

The parity sequence $o_{k,r}[n]$ dictates the magnitude of the switching sequence $s_{k,r}[n]$; therefore, the zeros in the parity sequence correspond to zeros in the switching sequence. Given this parity sequence, the symbols "$1, 0, 0, -1, 0$" and "$-1, 1, 0$" are used to construct the switching sequence

$$s_{k,r}[n] = \ldots, 1, 0, 0, -1, 0, -1, 1, 0, \ldots.$$

The choice of the symbol "$1, 0, 0, -1, 0$" over "$-1, 0, 0, 1, 0$" and "$-1, 1, 0$" over "$1, -1, 0$" is arbitrary as any combination of these symbols ensure that $|\alpha_1[n]| \le 1$. In this example, the resulting partial sum sequence is

$$\alpha_1[n] = \ldots, 0, 1, 1, 1, 0, 0, -1, 0, \ldots.$$

Fig. 7 displays an example of sequencing logic that generates these symbols in the switching sequence. This sequencing logic contains two D flip-flops with enables and a 2:1 multiplexer. Additionally, a pseudorandom sequence $r_k[n]$ is used to select between the two symbol types and is generated by logic that is not shown in the figure.

Each symbol type from (11) must be further decomposed into two "halves" to describe how the sequencing logic in Fig. 7 generates the desired switching sequence. The first half of the symbol—i.e., the first "$\pm 1, 0, \ldots, 0$"—is called the *head* of the symbol, and the second half is called the *tail*. The four states of the D flip-flops correspond to the two symbol types in $s_{k,r}[n]$ and the two segments, head and tail, of the symbol. The bit in the leftmost flip-flop represents the value of $|\alpha_1[n]|$. Since $|\alpha_1[n+1]| = 1$ when $s_{k,r}[n]$ is an element of a symbol's head, and $\alpha_1[n+1] = 0$ when $s_{k,r}[n]$ is an element of a symbol's tail,

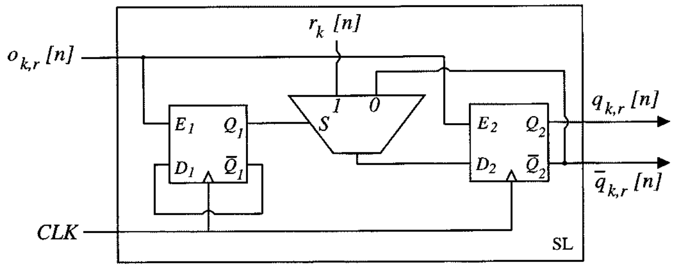


Fig. 7. The first-order low-pass sequencing logic with dither.

the bit in the leftmost flip-flop tracks whether $s_{k,r}[n]$ is an element of the head or the tail of a symbol. The rightmost flip-flop contains the $q_{k,r}[n]$ sequence that dictates the symbol types.

The symbol types are chosen randomly according to the pseudorandom sequence $r_k[n]$ so that there are no tones in the switching sequence. This pseudorandom sequence is called the *dither* sequence, and a switching block that uses a dither sequence to select its symbol types is called a *dithered* switching block. Ideally, the dither sequence is a sequence of bits that are uniformly distributed and independent. In this implementation, each switching block in a given layer shares the same dither sequence.

Undithered switching blocks may also be utilized to reduce hardware complexity and potentially decrease signal-band DAC noise power. In an undithered switching block, the same symbol type is used throughout the switching sequence, and the sequencing logic can be reduced to a single D flip-flop with enable. The resulting switching sequence can contain tones that lower the noise floor of its PSD relative to the dithered case. This reduced noise floor can give rise to less signal-band DAC noise power. However, the resulting spurious tones in the DAC noise can be prohibitive for a given application. To optimize this tradeoff, some combination of dithered and undithered switching blocks may be employed.

Fig. 8 displays the PSDs of the DAC noise and quantization noise from behavioral simulations of the 5-bit $\Delta\Sigma$ modulator that was introduced in Section II. The units of the PSDs are dB relative to Δ^2, where Δ is the step size of the ADC. The capacitor errors in the DAC banks were modeled as independent Gaussian random variables with standard deviations of 1% of their nominal value. This is *not* equivalent to "1% matching" which implies that adjacent capacitors in a given IC are matched within 1%. The input to the $\Delta\Sigma$ modulator was a $-$ 1 dB (relative to full-scale), $-$ 1 kHz ($\approx 0.0005 f_s$) sinusoid. To illustrate the effects of dither, a dither sequence was applied to selected switching blocks in the simulated $\Delta\Sigma$ modulator. The noise PSDs in Fig. 8 illustrate how the dither sequences either eliminate or reduce spurious tones in the DAC noise depending on which switching blocks are dithered.

The total hardware required for the sequencing logic in a (2^b+1)-level digital encoder depends on how many switching blocks are dithered. When all switching blocks are dithered, $2\cdot(2^b-1)$ D flip-flops with enables, 2^b-1 2:1 multiplexers, and b pseudorandom sequences are required. When none of the switching blocks are dithered, 2^b-1 D flip-flops with enables are required. For the implementation of the $\Delta\Sigma$ modulator in [43], the 2:1 multiplexer in the sequencing logic is realized by three NAND gates, and the pseudorandom sequences are constructed using a pseudorandom number generator with 28 D flip-flops and 7 XOR gates. The total hardware required for the sequencing logic (not including the pseudorandom number generator) in the digital encoder presented in [43] is 62 D flip-flops and 93 NAND gates.

C. Second-Order Low-Pass Sequencing Logic

The first-order low-pass sequencing logic generates a first-order high-pass switching sequence regardless of the values in the switching block's input sequence. However, the restrictions on $s_{k,r}[n]$ given by (6) prevent an analogous claim for the second-order low-pass sequencing logic. For $s_{k,r}[n]$ to be a second-order high-pass switching sequence, the switching block attempts to bound the magnitude of its *double sum sequence* $\alpha_2[n]$ by a constant $K<\infty$ for all n

$$|\alpha_2[n]| = \left|\sum_{l=0}^{n-1}\alpha_1[l]\right| = \left|\sum_{l=0}^{n-1}\sum_{m=0}^{l-1} s_{k,r}[m]\right| < K.$$

Because the parity of $x_{k,r}[n]$ dictates when $s_{k,r}[n]$ is zero, the sequence $|\alpha_2[n]|$ can be made arbitrarily large by applying the appropriate $x_{k,r}[n]$. For example, suppose $x_{k,r}[0]=1$, and $x_{k,r}[n]=0$ for all $n>0$. Given $\alpha_1[0]=\alpha_2[0]=0$, then $|\alpha_1[n]|=1$ and $|\alpha_2[n]|=n-1$ for all $n>0$. However, if $x_{k,r}[n]$ is odd with some regularity (as is the case when the DAC is used in a $\Delta\Sigma$ modulator), a switching sequence can be constructed whose double sum is a bounded sequence, thereby giving rise to second-order high-pass shaped DAC noise.

One method for creating such a switching sequence is to again use symbols of the form in (11), but with the symbol type chosen to minimize the magnitude of the double sum sequence, $\alpha_2[n]$. In this case, the magnitude of $\alpha_1[n]$ is bounded by one, so the switching sequence is at least a first-order high-pass switching sequence. At any time n within a symbol's head, $|\alpha_1[n+1]|=1$, and it follows that

$$\alpha_2[n+2]=\alpha_2[n+1]+\alpha_1[n+1]=\alpha_2[n+1]\pm 1. \quad (12)$$

Thus, $\alpha_2[n+1]$ increments or decrements by one at each sample time within a symbol's head. However, at any time n within a symbol's tail, $\alpha_1[n+1]=0$ and $\alpha_2[n+2]=\alpha_2[n+1]$. It follows that the symbol's type and the length of its head determine the values in $\alpha_2[n]$: if a symbol starts at time n and its head's length is N_o samples, it can be shown using induction that

$$\alpha_2[n+N_o+1]=\alpha_2[n]+(\pm N_o) \quad (13)$$

where the sign of N_o is determined by the symbol type. To minimize $|\alpha_2[n+N_o+1]|$, the sign of N_o in the above expression should be the opposite of the sign of $\alpha_2[n]$.

To construct such a switching sequence, each switching block ideally calculates $\alpha_2[n]$ with which it selects between the two symbol types. However, when implemented with finite register sizes, the switching block can only estimate $\alpha_2[n]$. This estimate, which is denoted $\hat{\alpha}_2[n]$, has a maximum $M_{\max}\geq 0$ and minimum $M_{\min}\leq 0$ which are determined by the number of states in a finite-state machine. Therefore, the estimate $\hat{\alpha}_2[n]$

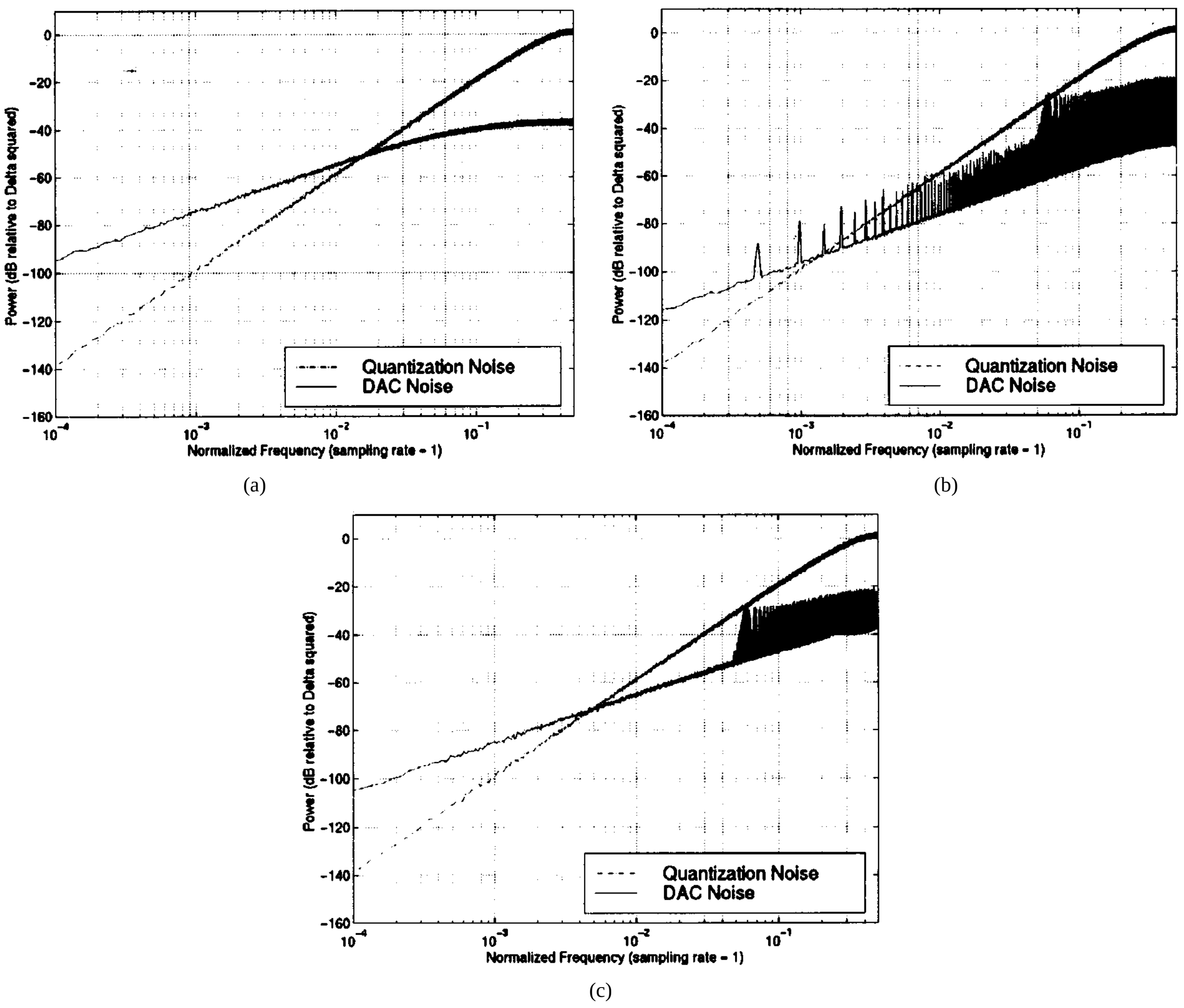


Fig. 8. DAC noise and quantization noise from a simulation of a 5-bit $\Delta\Sigma$ modulator with the first-order low-pass sequencing logic. (a) Dither in all of switching blocks. (b) Dither in none of the switching blocks. (c) Dither in the switching blocks in layers 3, 4, and 5.

equals $\alpha_2[n]$ only as long as $\alpha_2[n]$ does not exceed the estimate's range (i.e., $M_{\min} \le \alpha_2[n] \le M_{\max}$). The switching block uses the sign of $\hat{\alpha}_2[n]$ to determine the symbol types. To approximate $\alpha_2[n]$, the sequence $\hat{\alpha}_2[n]$ *saturates* when it reaches $M_{\max}$ or $M_{\min}$

$$\hat{\alpha}_2[n+1] = \begin{cases} M_{\max}, & \text{if } \hat{\alpha}_2[n] + \alpha_1[n] > M_{\max} \\ M_{\min}, & \text{if } \hat{\alpha}_2[n] + \alpha_1[n] < M_{\min} \\ \hat{\alpha}_2[n] + \alpha_1[n], & \text{otherwise.} \end{cases} \tag{14}$$

The effect of saturation in the above equation can be represented as an accumulated additive error

$$\hat{\alpha}_2[n+1] = \hat{\alpha}_2[n] + \alpha_1[n] + \varepsilon[n+1] \tag{15}$$

where $\varepsilon[n]$ is called the *saturation error*. The behavior of the saturation error determines whether the switching sequence is a second-order high-pass switching sequence. Since $\alpha_1[n]$ is constrained to the set $\{-1, 0, 1\}$, it follows that $\varepsilon[n]$ is also constrained to this set. Let $N = \min(M_{\max}, |M_{\min}|)$. For $\varepsilon[n]$ to be nonzero, there must be a run of at least N zeros in the parity sequence $o_{k,r}[n]$. Thus, $x_{k,r}[n]$ must be even for N consecutive samples to cause any saturation error. If the switching block's input is odd at least once within every N-length segment, the saturation error is always zero. From (15), it follows that

$$\hat{\alpha}_2[n] = \varepsilon[n] + \sum_{j=0}^{n-1} (\alpha_1[j] + \varepsilon[j]).$$

The sequence $\alpha_2[n]$ is the partial sum of $\alpha_1[n]$; thus, it follows that

$$\alpha_2[n] = \hat{\alpha}_2[n] - \sum_{j=0}^{n} \varepsilon[j]. \tag{16}$$

Because $\hat{\alpha}_2[n]$ is a bounded sequence, $\alpha_2[n]$ is a bounded sequence if and only if the partial sum of $\varepsilon[n]$ is a bounded sequence. Therefore, $s_{k,r}[n]$ is a second-order high-pass switching sequence if and only if the partial sum of $\varepsilon[n]$ is a bounded sequence.

The second-order low-pass sequencing logic is shown in Fig. 9. The 3-state accumulator produces $-\alpha_1[n]$ and the M-state accumulator produces $-\hat{\alpha}_2[n]$. Therefore, the sign of the value in the M-state accumulator is used to choose the symbol types. However, when the M-state accumulator's value and hence $\hat{\alpha}_2[n]$ is zero, the dither sequence $r_k[n]$ is

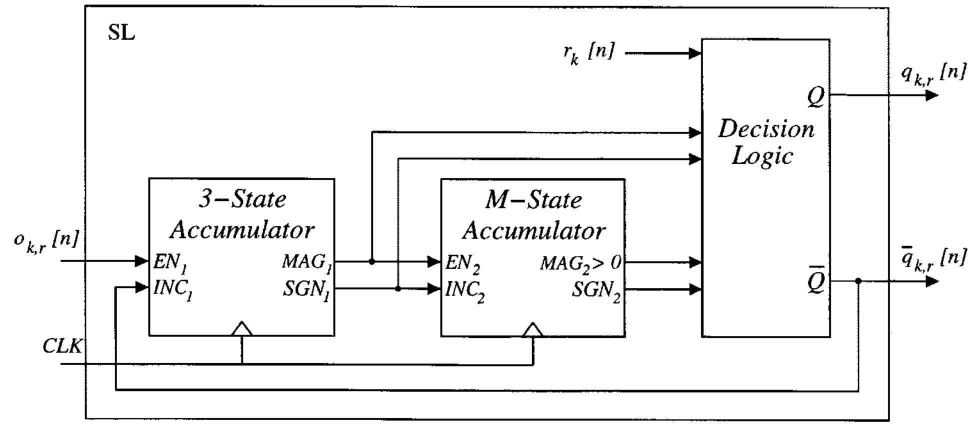


Fig. 9. The second-order, low-pass sequencing logic with dither.

used to choose the symbol type randomly as a means of reducing the spurious tones in $s_{k,r}[n]$. The 3-state accumulator tracks the intrasymbol information for the switching sequence: $|\alpha_1[n+1]| = 1$ when $s_{k,r}[n]$ is an element of the symbol's head, and $\alpha_1[n+1] = 0$ when $s_{k,r}[n]$ is an element of the symbol's tail. When $s_{k,r}[n]$ is in the head of a symbol, the sign of the 3-state accumulator's value is the sign of the tail's first element.

The following is a more detailed description of each element in Fig. 9:

1) ***3-State Accumulator:*** A state machine that implements an accumulator restricted to the following three states: $\{-1, 0, 1\}$.
   - EN_1 and INC_1 control the state transitions of the 3-state accumulator as follows:

$$I_1[n+1] = \begin{cases} I_1[n], & \text{if } \mathrm{EN}_1 = 0 \\ I_1[n]+1, & \text{if } \mathrm{EN}_1, \mathrm{INC}_1 = 1 \\ I_1[n]-1, & \text{otherwise.} \end{cases} \quad (17)$$

   where $I_1[n]$ is the accumulator's state at time n. The feedback prevents $I_1[n]$ from incrementing or decrementing beyond its three states.
   - $\mathrm{MAG}_1 = |I_1[n]|$ is the magnitude of the accumulator.
   - SGN_1 represents the sign of $I_1[n]$

$$\mathrm{SGN}_1 = \begin{cases} 1, & \text{if } I_1[n] > 0 \\ 0, & \text{if } I_1[n] < 0 \\ \text{don't care}, & \text{if } I_1[n] = 0. \end{cases} \quad (18)$$

2) ***M-State Accumulator:*** A state machine that implements a saturating accumulator restricted to the M integers in the set $\{-\lceil (M-1)/2 \rceil, \ldots, \lfloor (M-1)/2 \rfloor\}$.
   - EN_2 and INC_2 control the state transitions of the M-state logic as follows:

$$I_2[n+1] = \begin{cases} I_2[n], & \text{if } \mathrm{EN}_2 = 0 \\ \min(I_2[n]+1, N_{\max}), & \text{if } \mathrm{EN}_2, \mathrm{INC}_2 = 1 \\ \max(I_2[n]-1, N_{\min}), & \text{otherwise.} \end{cases} \quad (19)$$

   where $I_2[n]$ is the accumulator's state at time $n, N_{\min} = -\lceil (M-1)/2 \rceil$, and $N_{\max} = \lfloor (M-1)/2 \rfloor$;
   - "$\mathrm{MAG}_2 > 0$" is high when $|I_2[n]| > 0$ and low when $I_2[n] = 0$;
   - SGN_2 represents the sign of $I_2[n]$ and is analogous to SGN_1 in the 3-state accumulator.

3) ***Decision Logic:*** Combinational logic that generates Q and its complement, $\bar{Q}$, as follows:

$$Q = \begin{cases} \mathrm{SGN}_1, & \text{if } \mathrm{MAG}_1 = 1 \\ \mathrm{SGN}_2, & \text{if } \mathrm{MAG}_1 = 0, \text{“}\mathrm{MAG}_2 > 0\text{''} = 1 \\ r_k[n], & \text{otherwise} \end{cases} \quad (20)$$

   where $r_k[n]$ is a pseudorandom sequence that approximates a sequence of bits that are uniformly distributed and independent.

Fig. 10 displays DAC noise PSDs from behavioral simulations of the 5-bit $\Delta\Sigma$ modulator presented in Section II with the second-order low-pass sequencing logic. Except for the sequencing logic, all other characteristics of these simulations were the same as those for the first-order low-pass case described previously. Various M-state accumulators were implemented with counters of different sizes. For smaller values of M, the saturation error contributes more power to the DAC noise. In the limit when "no counter" is used (i.e., when $M = 1$ and $I_2[n] = 0$ for all n), the sequencing logic reduces to the first-order low-pass sequencing logic. When the M-state accumulator is implemented with a 4-bit counter, the power of the signal-band DAC noise decreases relative to the "no counter" noise, but the saturation error prevents the DAC noise from being second-order high-pass shaped. However, with the M-state accumulator realized by an 8-bit counter, the DAC noise in Fig. 10 has the spectral shape of a second-order high-pass sequence.

The additional hardware required to implement the second-order sequencing logic relative to the first-order sequencing logic includes the decision logic, which can be implemented by two 2:1 multiplexers and an inverter, and the M-state accumulator. If $M = 2^{b_o}$ and the M-state accumulator

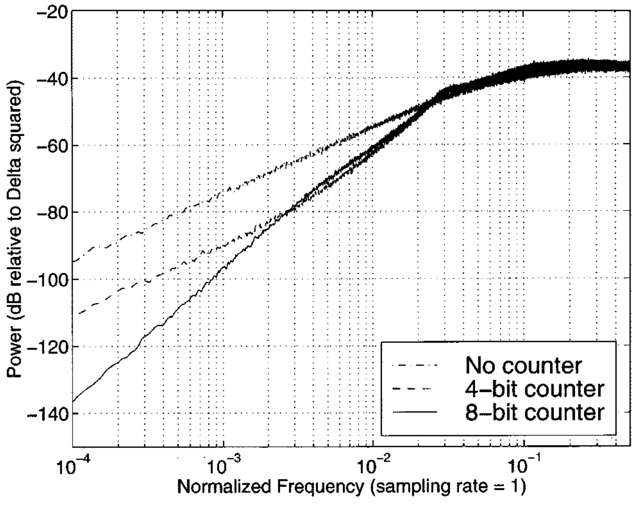


Fig. 10. DAC noise from a simulation of an analog $\Delta\Sigma$ modulator with the second-order, low-pass sequencing logic with dither and varying counter sizes for the M-state logic.

is implemented with a b_o-bit up/down counter, then SGN_2 is the MSB of the counter and "$\mathrm{MAG}_2 > 0$" can be realized by an OR gate with a fan-in of $(b_o - 1)$ bits. The second-order sequencing logic for the implementation of this switching block in [44] uses a 4-bit counter and requires 25 total gates and flip-flops.

IV. Splitting Network and Parity Logic

In an ADC $\Delta\Sigma$ modulator as in Fig. 1, the delay of the feedback DAC must be small enough so that its output is available well before the next $\Delta\Sigma$ modulator input is clocked in. Therefore, the delay introduced by the switching blocks can limit the maximum sample rate of the ADC $\Delta\Sigma$ modulator. The sequencing logic blocks presented in Section III do not contribute to the switching block's propagation delay because their outputs can be set before their next input is available. However, the splitting network and parity logic do cause propagation delay.

If the input to the switching block were a binary encoded number, the splitting network could be implemented with binary adders as shown in Fig. 5, and the parity logic would require no hardware as the input's parity bit would be its LSB. However, the propagation delay introduced by the adders could be significant. In this section, splitting networks are presented that avoid using conventional adders by employing alternative coding schemes for the switching blocks' input and output sequences. Without conventional adders, these splitting networks tend to introduce less propagation delay. The two splitting networks in this section constitute the medium-speed and high-speed switching blocks that offer a tradeoff between complexity and propagation delay. Additionally, efficient implementations of the parity logic blocks are presented for both switching block types.

A. Medium-Speed Switching Block

Fig. 11 displays the medium-speed switching block. The parity logic consists of an XOR gate and the splitting network consists of two 2:1 multiplexers. In this section, the sequence "$x_{k,r}[n]$" represents both the input of $S_{k,r}$ and its numerical value; the appropriate representation can be determined by its context. The switching block employs "extra-LSB encoding" of its input and output sequences. Motivated by [39] and detailed in [42], the extra-LSB code of $x_{k,r}[n]$ consists of $k+1$ bits that are denoted $x_{k,r}^{(i)}[n]$ $(i = 0, \ldots, k)$, each of which take on a value of one or zero. The numerical value of $x_{k,r}[n]$ is interpreted as

$$x_{k,r}[n] = \sum_{i=1}^{k} 2^{i-1} x_{k,r}^{(i)}[n] + x_{k,r}^{(0)}[n]. \tag{21}$$

Thus, the extra-LSB code contains two LSBs, $x_{k,r}^{(0)}[n]$ and $x_{k,r}^{(1)}[n]$, both with unity weighting. A conventional unsigned binary encoded number can be converted to an extra-LSB encoded number by appending the 0th bit and setting it low.

With this coding technique, the arithmetic performed by the splitting network only modifies the two LSBs of $x_{k,r}[n]$. As described in Section II, the switching sequence $s_{k,r}[n]$ is nonzero only when $x_{k,r}[n]$ is odd. It follows from (22) that whenever $x_{k,r}[n]$ is odd, one of its LSBs is one and the other is zero. Thus, the splitting network adds ± 1 to $x_{k,r}[n]$ when only one of its LSBs is high, which implies that the carry bit can never propagate beyond the two LSBs. When $x_{k,r}[n]$ is odd, the splitting network adds one to $x_{k,r}[n]$ by setting both of its LSBs high, or subtracts one from $x_{k,r}[n]$ by setting both of its LSBs low. Since the sequences $x_{k,r}[n] + s_{k,r}[n]$ and $x_{k,r}[n] - s_{k,r}[n]$ are always even valued, both LSBs are equal in each of these sequences. The splitting network performs the divide-by-two operation by right shifting the $k-1$ MSBs of $x_{k,r}[n]$ and using one of the LSBs of $x_{k,r}[n] \pm s_{k,r}[n]$ as the second LSB of each output.

The two LSBs of $x_{k,r}[n]$ determine its parity. The value of $x_{k,r}[n]$ is odd only when one of its LSBs is one and the other is zero; otherwise, it is even. Therefore, the parity logic implements

$$o_{k,r}[n] = x_{k,r}^{(0)}[n] \oplus x_{k,r}^{(1)}[n] \tag{22}$$

where $\oplus$ represents the XOR operation.

The hardware in each switching block is independent of its location in the digital encoder; therefore, the $(2^b + 1)$-level digital encoder requires $2 \cdot (2^b - 1)$ 2:1 multiplexers for its splitting networks and $2^b - 1$ XOR gates for its parity logic. The efficiency of this implementation increases as the number of bits are increased because the complexity of each switching block does not depend on the bit width—i.e., number of bits—of its input. The medium-speed switching block is used in the 33-level digital encoder presented in [43] wherein the two multiplexers of each splitting network are realized by five NAND gates. The splitting networks and parity logic blocks for this implementation require a total of 186 logic gates. For the $\Delta\Sigma\mathrm{ADC}$ shown in Fig. 1, additional hardware is required to convert the thermometer coded output of the flash ADC to an extra-LSB code.

The delay performance for the digital encoder is determined by the digital encoder's *critical path*, which is defined as the longest path that an input bit must traverse in a given clock period to set an output bit. Within the medium-speed switching block, the longest path from its input to its outputs consists of

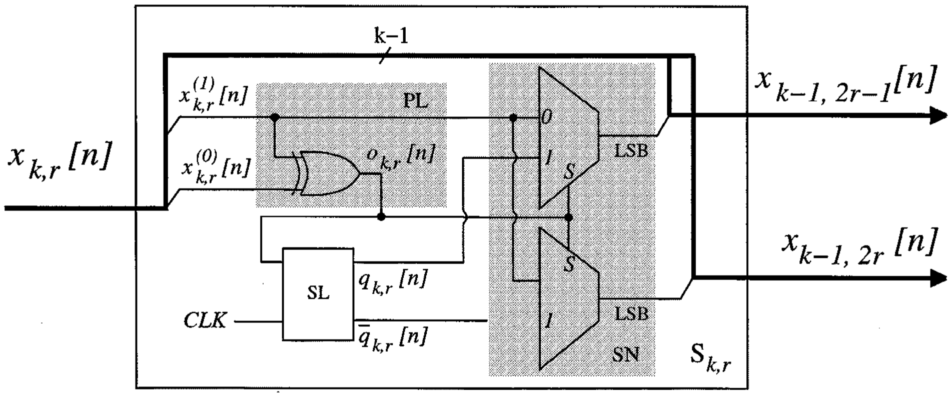


Fig. 11. The medium-speed switching block.

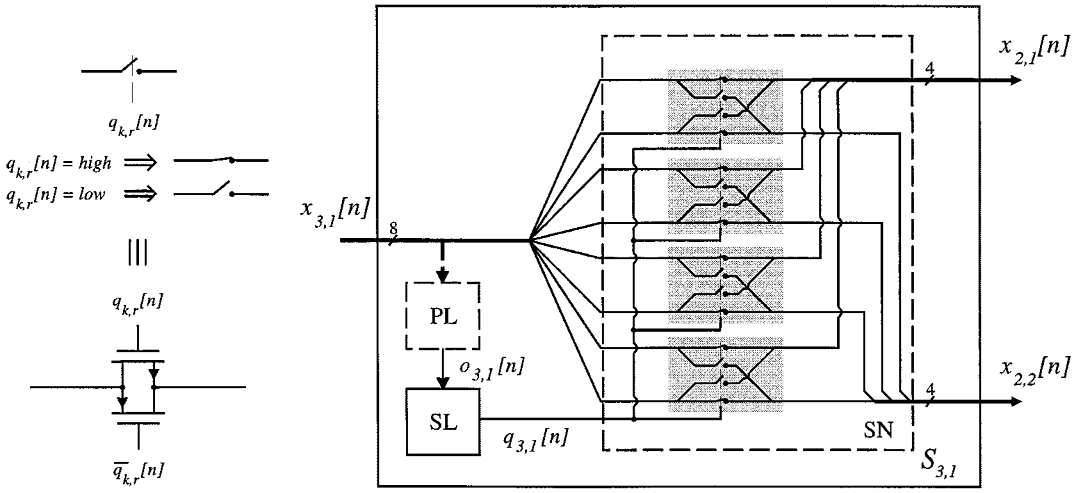


Fig. 12. The splitting network for a high-speed switching block and the CMOS implementation of a transmission gate.

an XOR gate and a 2:1 multiplexer. Therefore, the critical path of the (2^b+1)-level digital encoder consists of b XOR gates and b 2:1 multiplexers. HSPICE 0.5-μm CMOS simulations of the 33-level digital encoder presented in [43] showed that this digital encoder has a delay of approximately 5.7 ns. This does not include the propagation delay of the thermometer-to-binary conversion performed in the $\Delta\Sigma$ADC's digital common-mode rejection flash ADC.

B. The High-Speed Switching Block

Fig. 12 displays an example high-speed switching block whose splitting network consists entirely of switches implemented by CMOS transmission gates. In this architecture, the parity logic does not physically reside within the switching block. The parity sequences are generated by an XOR tree as shown in Fig. 13. The high-speed switching block employs thermometer encoding of its input and output sequences. The sequence $x_{k,r}[n]$ is thermometer encoded if it has 2^k bits $(x_{k,r}^{(i)}[n], i = 1, \dots, 2^k)$ that are assigned as follows:

$$x_{k,r}^{(i)}[n] = \begin{cases} 1, & \text{if } i \le x_{k,r}[n] \\ 0, & \text{else.} \end{cases} \tag{23}$$

Thus, with thermometer encoding

$$x_{k,r}[n] = \sum_{i=1}^{2^k} x_{k,r}^{(i)}[n]. \tag{24}$$

With thermometer encoding, the splitting network performs the desired arithmetic by routing the odd indexed bits of $x_{k,r}[n]$ to one output and the even indexed bits of $x_{k,r}[n]$ to the other output, or vice versa, depending upon $q_{k,r}[n]$. It can be shown that the numerical values of the sequences that comprise the even indexed bits and odd indexed bits of $x_{k,r}[n]$ are

$$\sum_{i=1}^{2^{k-1}} x_{k,r}^{(2i)}[n] = \left\lfloor \frac{x_{k,r}[n]}{2} \right\rfloor \tag{25}$$

and

$$\sum_{i=1}^{2^{k-1}} x_{k,r}^{(2i-1)}[n] = \left\lceil \frac{x_{k,r}[n]}{2} \right\rceil \tag{26}$$

respectively. Because $s_{k,r}[n]$ is limited to the set $\{-1, 0, 1\}$, it follows that

$$x_{k-1,2r-1}[n] = \begin{cases} \left\lfloor \frac{x_{k,r}[n]}{2} \right\rfloor, & \text{if } q_{k,r}[n] = 0 \\ \left\lceil \frac{x_{k,r}[n]}{2} \right\rceil, & \text{if } q_{k,r}[n] = 1 \end{cases} \tag{27}$$

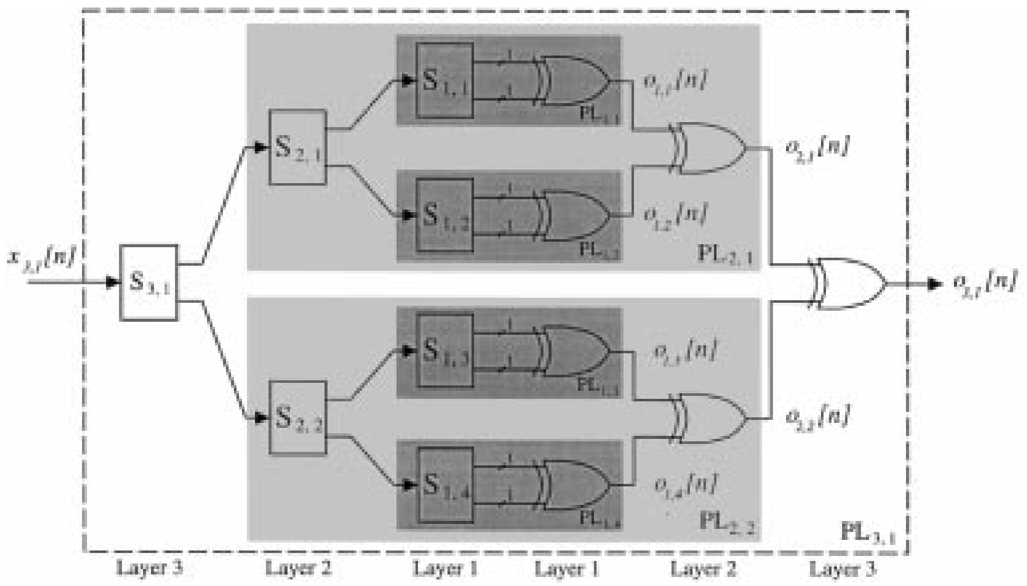


Fig. 13. The parity logic for the high-speed switching block.

and

$$x_{k-1,2r}[n] = \begin{cases} \left\lceil \frac{x_{k,r}[n]}{2} \right\rceil, & \text{if } q_{k,r}[n] = 0 \\ \left\lfloor \frac{x_{k,r}[n]}{2} \right\rfloor, & \text{if } q_{k,r}[n] = 1. \end{cases} \tag{28}$$

Therefore, by routing the input's even and odd indexed bits to separate outputs based on $q_{k,r}[n]$, the splitting network realizes the arithmetic in (27) and (28). Moreover, by preserving the order of these bits, the splitting network ensures its outputs are thermometer encoded.

Since the splitting network does not rely on $o_{k,r}[n]$ to route the bits of $x_{k,r}[n]$, the current sample of $o_{k,r}[n]$ can be determined after the outputs of the digital encoder are set. The parity logic block in this section exploits this flexibility to minimize its hardware. The number of gates required to directly determine the parity of a thermometer encoded number is proportional to its bit width. However, using the XOR tree as shown in Fig. 13, each parity logic block accounts for only one XOR gate.

The XOR tree is a consequence of the functional relationship between the outputs of a switching block and its input. From (4), the values of the output sequences of a switching block must add to the value of the input. Thus, the parity of $x_{k,r}[n]$ can be determined by the parities of $x_{k-1,2r-1}[n]$ and $x_{k-1,2r}[n]$

$$o_{k,r}[n] = o_{k-1,2r-1}[n] \oplus o_{k-1,2r}[n]. \tag{29}$$

The outputs of each switching block in layer one are 1-bit sequences. This implies that $x_{0,r}[n] = o_{0,r}[n]$. By recursively implementing (29), the XOR tree generates the parity bits for each switching block.

The hardware counts in the medium-speed and high-speed switching blocks differ only in their splitting networks. With the high-speed switching block, the number of transmission gates in the splitting network depends on the bit width of the switching block's input. However, the number of transmission gates per layer is independent of the layer number: each bit of a switching block's input is processed by two transmission gates—one on and one off—and the total number of input bits is constant for each layer. Thus, with the high-speed switching block, the (2^b+1)-level digital encoder requires $b \cdot 2^{b+1}$ transmission gates for its splitting networks and $2^b - 1$ XOR gates for its parity logic. A 33-level implementation of this digital encoder for the $\Delta\Sigma$ADC shown in Fig. 1 requires 320 transmission gates for its splitting network and 31 XOR gates for its parity logic. If the input to the digital encoder were a binary encoded number, as in the case of a $\Delta\Sigma$DAC, a binary-to-thermometer encoder would also be required to implement this digital encoder.

The high-speed switching block tends to have less propagation delay than the medium-speed switching block because the parity logic in the high-speed switching block does not contribute to its delay. As previously mentioned, the sequencing logic does not require the current sample of $o_{k,r}[n]$ to produce $q_{k,r}[n]$. Therefore, $q_{k,r}[n]$ can be calculated and used to set the transmission gates before $y[n]$ is clocked into the digital encoder. Additionally, the XOR tree processes the output bits of the digital encoder and does not contribute to the digital encoder's critical path. Therefore, the critical path of the (2^b+1)-level digital encoder, which is experienced by each of its input bits, consists of b preset transmission gates. HSPICE 0.5-μm CMOS simulations of a 33-level digital encoder with high-speed switching blocks showed that this digital encoder has a delay of approximately 1.1 ns, which is approximately a five-times improvement over a 33-level digital encoder with medium-speed switching blocks. This delay does not include the propagation delay of a binary-to-thermometer encoder that would be required in a $\Delta\Sigma$DAC. An implementation that uses the high-speed switching blocks for its minimal delay is presented in [45].

V. Hardware Comparison for Various Mismatch-Shaping DACs

To compare the hardware complexity of the tree-structured mismatch-shaping DAC encoders presented here to other implementations, Tables I and II give estimated hardware requirements for mismatch-shaping DAC encoders appropriate for use

TABLE I
ESTIMATED HARDWARE REQUIREMENTS FOR UNDITHERED MISMATCH-SHAPING DAC ENCODERS FOR USE WITHIN A 5-bit $\Delta\Sigma$ADC

Tree-Structure		Data-Weighted Averaging		Butterfly Shuffler[3]
Med. Speed	High Speed	Barrel Shifter[1]	Binary/Therm[2]	
5-bit T/B encoder	320 T-gates	5-bit T/B encoder	5-bit T/B encoder	320 T-gates
93 MUXes	31 MUXes	320 T-gates	2 5-bit adders	160 XORs
31 XORs	31 XORs	5-bit adder	62 XNORs	80 INVs
31 DFFs	31 DFFs	5 DFFs	36 DFFs	80 DFFs

TABLE II
ESTIMATED HARDWARE REQUIREMENTS FOR MISMATCH-SHAPING DAC ENCODERS WITH DITHER OR OTHER HARMONIC DISTORTION COMPENSATION FOR USE WITHIN A $\Delta\Sigma$ADC OF SPECIFIED SIZE

5-bit Tree-Structure		Data-Weighted Averaging		5-bit Butterfly Shuffler[3]
Med. Speed	High Speed	3-bit Rotational[4]	5-bit BiDWA[5]	
5-bit T/B encoder	320 T-gates	3-bit T/B encoder	5-bit T/B encoder	320 T-gates
155 MUXes	93 MUXes	6×1024-bit ROM	320 T-gates	320 XORs
31 XORs	31 XORs	6 DFFs	15 MUXes	80 INVs
62 DFFs	62 DFFs	1 random bit	2 5-bit adders	80 MUXes
5 random bits	5 random bits		10 DFFs	160 DFFs
				5 random bits

in a $\Delta\Sigma$ADC. When possible, the DAC encoder hardware is estimated for an implementation in the 5-bit $\Delta\Sigma$ADC shown in Fig. 1. In both tables, the abbreviations "INV," "MUX," "XOR," and "XNOR" stand for inverter, 2:1 multiplexer, exclusive-or, and exclusive-nor, respectively. The abbreviation "T-gate" denotes a two-transistor CMOS transmission gate, and the abbreviation "T/B encoder" denotes a thermometer-to-binary encoder. A D flip-flop, denoted "DFF," is assumed to have true and complemented outputs available; the D flip-flop with enable, shown in Fig. 7, is implemented using a D flip-flop and a 2:1 multiplexer.

The mismatch-shaping DAC encoders shown in Table I provide no hardware to eliminate or reduce spurious tones and the hardware differences are not as pronounced. However, when extra hardware is utilized to combat harmonic distortion, Table II shows that both the Bidirectional DWA (BiDWA) and tree-structured DAC encoders contain the least hardware. The BiDWA DAC encoder requires minimal hardware because it depends entirely on the randomness of its input to reduce tones in its resulting DAC noise. Any dc input to a $(2^b + 1)$-level BiDWA DAC, besides the trivial inputs of 0 and 2^b, causes its DAC noise to be tonal. On the other hand, the dithered tree-structured DAC has been mathematically proven to produce no tones in its DAC noise [46]. In the Butterfly Shuffler architecture, it is assumed that the logic driving the swapper cells is implemented as the sequencing logic for the tree-structured DAC and only one random bit is used for each *column* in the swapper cell matrix. For the second-order tree-structured DAC, the hardware difference becomes more pronounced as the 5-bit implementation presented in [44] requires only 988 gates while the 3-bit second-order architecture presented in [15] requires 3500 gates.

VI. CONCLUSION

This paper has presented various implementations of the tree-structured mismatch-shaping DAC. First-order and second-order low-pass sequencing logic have been presented that provide a tradeoff between DAC-noise power and hardware complexity. High-speed and medium-speed implementations of the splitting network and parity logic have been presented that offer a tradeoff between the digital encoder's propagation delay and hardware complexity. By appropriately choosing between medium-speed, high-speed, first-order dithered or nondithered, or second-order implementations, the tree-structured DAC can be optimized for hardware complexity, propagation delay, signal-band DAC-noise power, and DAC-noise harmonic distortion.

REFERENCES

[1] W. Redman-White and D. J. L. Bourner, "Improved dynamic linearity in multi-level Σ-Δ converters by spectral dispersion of D/A distortion products," in *Proc. ECCTD European Conf. Circuit Theory and Design*, Brighton, U.K., Sept. 5–8, 1989, pp. 205–208.

[2] R. K. Henderson and O. Nys, "Dynamic element matching techniques with arbitrary noise shaping function," in *Proc. IEEE Int. Symp. Circuits and Systems*, May 1996, pp. 293–296.

[3] O. Nys and R. K. Henderson, "An analysis of dynamic element matching techniques in sigma-delta modulation," in *Proc. IEEE Int. Symp. Circuits and Systems*, May 1996, pp. 231–234.

[4] M. Adams and C. Toumazou, "A novel architecture for reducing the sensitivity of multibit sigma-delta to DAC nonlinearity," in *Proc. IEEE Symp. Circuits and Systems*, May 1995, pp. 17–20.

[5] A. Keady and C. Lyden, "Comparison of mismatch error shaping in multibit oversampled converters," *Electron. Lett.*, vol. 36, no. 6, pp. 506–508, Mar. 1998.

[6] L. Hernández, "Binary weighted D/A converters with mismatch-shaping," *Electron. Lett.*, vol. 33, no. 24, pp. 2006–2008, Nov. 1997.

[7] ——, "A model of mismatch-shaping D/A conversion for linearized DAC architectures," *IEEE Trans. Circuits Syst. I*, vol. 45, pp. 1068–1076, Oct. 1998.

[8] J. Steensgaard, U.-K. Moon, and G. Temes, "Mismatch-shaping serial digital-to-analog converter," in *Proc. IEEE Int. Symp. Circuits and Systems*, vol. 2, May 1999, pp. 5–8.

[9] D. Scholnik and J. Coleman, "Vector delta-sigma modulation with integral shaping of hardware-mismatch errors," in *Proc. IEEE Int. Symp. Circuits and Systems*, vol. 5, May 2000, pp. 677–680.

[10] J. Steensgaard, "High-resolution mismatch-shaping digital-to-analog converters," in *Proc. IEEE Int. Symp. Circuits and Systems*, vol. 1, May 2001, pp. 516–519.

[11] B. H. Leung and S. Sutarja, "Multi-bit sigma-delta A/D converter incorporating a novel class of dynamic element matching techniques," *IEEE Trans. Circuits Syst. II*, vol. 39, pp. 35–51, Jan. 1992.

[12] F. Chen and B. H. Leung, "A high resolution multibit sigma-delta modulator with individual level averaging," *IEEE J. Solid-State Circuits*, vol. 30, pp. 453–460, Apr. 1995.

[13] R. Schreier and B. Zhang, "Noise-shaped multi-bit D/A converter employing unit elements," *Electron. Lett.*, vol. 31, no. 20, pp. 1712–1713, Sept. 1995.

[14] H. Lin, J. B. da Silva, B. Zhang, and R. Schreier, "Multi-bit DAC with noise-shaped element mismatch," in *Proc. IEEE Int. Symp. Circuits and Systems*, May 1996, pp. 235–238.

[15] A. Yasuda, H. Tanimoto, and T. Iida, "A third-order $\Delta\Sigma$ modulator using second-order noise-shaping dynamic element matching," *IEEE J. Solid-State Circuits*, vol. 33, pp. 1879–1886, Dec. 1998.

[16] Z. Czarnul, K. Oda, and T. Iida, "A straightforward design of mismatch-shaped multi-bit $\Delta\Sigma$ D/A systems," in *Proc. IEEE Int. Symp. Circuits and Systems*, vol. 5, May 2000, pp. 717–720.

[17] M. J. Story, "Digital to analogue converter adapted to select input sources based on a preselected algorithm once per cycle of a sampling signal," U.S. Patent 5 138 317, Aug. 11, 1992.

[18] H. S. Jackson, "Circuit and method for cancelling nonlinearity error associated with component value mismatches in a data converter," U.S. Patent 5 221 926, June 22, 1993.

[19] R. T. Baird and T. S. Fiez, "Linearity enhancement of multi-bit $\Delta\Sigma$ A/D and D/A converters using data weighted averaging," *IEEE Trans. Circuits Syst. II*, vol. 42, pp. 753–762, Dec. 1995.

[20] T. Hamasaki, Y. Shinohara, H. Tersawa, K. Ochiai, M. Hiraoka, and H. Hanayama, "A 3-V, 22-mW multibit current-mode $\Delta\Sigma$ DAC with 100 dB dynamic range," *IEEE J. Solid-State Circuits*, vol. 31, pp. 1888–1894, Dec. 1996.

[21] O. Nys and R. K. Henderson, "A 19-bit low-power multi-bit sigma-delta ADC based on data weighted averaging," *IEEE J. Solid-State Circuits*, vol. 32, pp. 933–942, July 1997.

[22] I. Fujimori and T. Sugimoto, "A 1.5 V, 4.1 mW dual-channel audio delta-sigma D/A converter," *IEEE J. Solid-State Circuits*, vol. 33, pp. 1863–1870, Dec. 1998.

[23] I. Fujimori, A. Nogi, and T. Sugimoto, "A multi-bit delta-sigma audio DAC with 120-dB dynamic range," *IEEE J. Solid-State Circuits*, vol. 35, pp. 1066–1073, Aug. 2000.

[24] I. Fujimori, L. Longo, A. Hairapetian, K. Seiyama, S. Kosic, J. Cao, and S. Chan, "A 90 dB SNR, 2.5 MHz output-rate ADC using cascaded multibit delta-sigma modulation at $8\times$ oversampling ratio," *IEEE J. Solid-State Circuits*, vol. 35, pp. 1820–1828, Dec. 2000.

[25] K. Chen and T. Kuo, "An improved technique for reducing baseband tones in sigma-delta modulators employing data weighted averaging algorithm without adding dither," *IEEE Trans. Circuits Syst. II*, vol. 46, pp. 63–68, Jan. 1999.

[26] R. Radke, A. Eshraghi, and T. Fiez, "A spurious-free delta-sigma DAC using rotated data weighted averaging," in *Proc. IEEE Custom Integrated Circuits Conf.*, May 1999, pp. 125–128.

[27] D. Cini, C. Samori, and A. Lacaita, "Double-index averaging: A novel technique for dynamic element matching in Δ-Σ A/D converters," *IEEE Trans. Circuits Syst. II*, vol. 46, pp. 353–358, Apr. 1999.

[28] Y. Geerts, M. Steyaert, and W. Sansen, "A high-performance multibit $\Delta\Sigma$ CMOS ADC," *IEEE J. Solid-State Circuits*, vol. 35, pp. 1829–1840, Dec. 2000.

[29] X. Gong, E. Gaalaas, M. Alexander, D. Hester, E. Walburger, and J. Bian, "A 120 dB multi-bit SC audio DAC with second-order noise shaping," in *Proc. IEEE ISSCC Dig. Tech. Papers*, Feb. 2000, pp. 344–345.

[30] M. Vadipour, "Techniques for preventing tonal behavior of data weighted averaging algorithm in Δ-Σ modulators," *IEEE Trans. Circuits Syst. II*, vol. 47, pp. 1137–1144, Nov. 2000.

[31] K. Vleugels, S. Rabii, and B. Wooley, "A 2.5 V broadband multi-bit $\Delta\Sigma$ modulator with 95 dB dynamic range," in *Proc. IEEE ISSCC Dig. Tech. Papers*, Feb. 2001, pp. 50–51.

[32] R. W. Adams and T. W. Kwan, "Data-directed scrambler for multi-bit noise shaping D/A converters," U.S. Patent 5 404 142, Apr. 4, 1995.

[33] T. W. Kwan, R. W. Adams, and R. Libert, "A stereo multibit sigma delta DAC with asynchronous master-clock interface," *IEEE J. Solid-State Circuits*, vol. 31, pp. 1881–1887, Dec. 1996.

[34] R. Adams, K. Nguyen, and K. Sweetland, "A 113-dB SNR oversampling DAC with segmented noise-shaped scrambling," *IEEE J. Solid-State Circuits*, vol. 33, pp. 1871–1878, Dec. 1998.

[35] T. Brooks, D. Robertson, D. Kelly, A. Del Muro, and S. Harston, "A 16 b sigma-delta pipeline ADC with 2.5 MHz output data rate," in *Proc. IEEE ISSCC Dig. Tech. Papers*, Feb. 1997, pp. 209–210.

[36] ——, "A cascaded sigma-delta pipeline A/D converter with 1.25 MHz signal bandwidth and 89 dB SNR," *IEEE J. Solid-State Circuits*, vol. 32, pp. 1896–1906, Dec. 1997.

[37] P. Ferguson, X. Haurie, and G. Temes, "A highly linear low-power 10-bit DAC for GSM," in *Proc. IEEE Custom Integrated Circuits Conf.*, May 2000, pp. 261–264.

[38] I. Galton, "Noise-shaping D/A converters for delta sigma modulation," in *Proc. IEEE Int. Symp. Circuits and Systems*, May 1996.

[39] A. Keady and C. Lyden, "Tree structure for mismatch noise-shaping multibit DAC," *Electron. Lett.*, vol. 33, no. 17, pp. 1431–1432, Aug. 1997.

[40] I. Galton, "Spectral shaping of circuit errors in digital-to-analog converters," *IEEE Trans. Circuits Syst. II*, vol. 44, pp. 808–817, Oct. 1997.

[41] ——, "Spectral shaping of circuit errors in digital-to-analog converters," U.S. Patent 5 684 482, Nov. 4, 1997.

[42] H. T. Jensen and I. Galton, "A reduced-complexity mismatch-shaping DAC for delta-sigma data converters," in *Proc. IEEE Int. Symp. Circuits and Systems*, May 31–June 3 1998, pp. 504–507.

[43] E. Fogleman, I. Galton, W. Huff, and H. Jensen, "A 3.3 V single-poly CMOS audio ADC delta-sigma modulator with 98 dB peak SINAD and 105-dB peak SFDR," *IEEE J. Solid State Circuits*, vol. 35, pp. 297–307, Mar. 2000.

[44] E. Fogleman, J. Welz, and I. Galton, "An audio ADC delta-sigma modulator with 100 dB SINAD and 102 dB DR using a second-order mismatch-shaping DAC," *IEEE J. Solid State Circuits*, vol. 36, pp. 339–348, Mar. 2001.

[45] J. Grilo, I. Galton, K. Wang, and R. Montemayor, "A 12 mW ADC delta-sigma modulator with 80 dB of dynamic range integrated in a single-chip bluetooth transceiver," in *Proc. IEEE Custom Integrated Circuits Conf.*, May 2001, pp. 23–26.

[46] J. Welz and I. Galton, "The mismatch-noise PSD from a tree-structured DAC in a second-order delta-sigma modulator with a midscale input," in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, vol. 4, May 7–11, 2001.

[47] "High Precision, High Speed ADC: 16-Bits, 2.5 MSPS," Product Data Sheet AK2700, Asahi Kasei Microsystems, Tokyo, Japan, Mar. 2000.

[48] "High speed DAC w/16-bits resolution at 2.5 MSPS," Product Data Sheet AK2710, Asahi Kasei Microsystems, Tokyo, Japan, Dec. 1999.

[49] "High-speed oversampling CMOS ADC with 16-bit resolution at a 2.5 MHz output word rate," Product Data Sheet AD9260, Analog Devices, Inc., Rev. B, Norwood, MA, 2000.

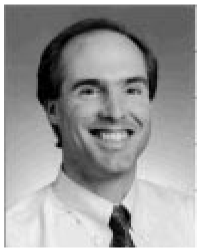
[50] "Stereo, 24-bit, 192 kHz, multibit $\Delta\Sigma$ DAC," Product Data Sheet AD1853, Analog Devices, Inc., Rev. A, Norwood, MA, 1999.

[51] "24-bit, 192 kHz Sampling enhanced multi-level, delta-sigma, audio digital-to-analog converter," Product Data Sheet PCM1737, Burr-Brown Corporation, Victoria, Australia, Mar. 2000.

[52] "24-bit, 192 kHz D/A converter for digital audio," Product Data Sheet CS4396, Cirrus Logic, Inc., Austin, TX, July 1999.

Jared Welz (S'98) received the B.S.E.E. degree from the University of California, Irvine, in 1993 and the M.S.E.E. degree, with an emphasis in communication theory, from the University of California, Los Angeles, in 1994. He is currently working toward the Ph.D. degree at the University of California at San Diego.

From 1994 to 1997, he was with AirTouch International (now Vodafone AirTouch), Pacific Bell Wireless (now Cingular Wireless), and L.A. Cellular (now AT&T Wireless). His research interests include data converters, signal processing, communication systems, and probability theory.



Ian Galton (M'92) received the Sc.B. degree from Brown University, Providence, RI, in 1984, and the M.S. and Ph.D. degrees from the California Institute of Technology, Pasadena, in 1989 and 1992, respectively, all in electrical engineering.

Since 1996, he has been an associate professor of electrical engineering at the University of California at San Diego, where he teaches and conducts research in the field of mixed-signal integrated circuits and systems for communications. Prior to 1996, he was with the University of California at Irvine, the NASA Jet Propulsion Laboratory, Acuson, and Mead Data Central. His research involves the invention, analysis, and integrated circuit implementation of key communication system blocks such as data converters, frequency synthesizers, and clock recovery systems. The emphasis of the research is on the development of digital signal processing techniques to mitigate the effects of nonideal analog circuit behavior with the objective of generating enabling technology for highly integrated, low-cost, communication systems. He regularly consults at several communications and semiconductor companies and teaches portions of various industry-oriented short courses on the design of data converters, PLLs, and wireless transceivers, and has served on a corporate Board of Directors and several Technical Advisory Boards.

Prof. Galton is a member of the IEEE Circuits and Systems Society Board of Governors, and is the Editor-in-Chief of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—II: ANALOG AND DIGITAL SIGNAL PROCESSING.

Eric Fogleman (S'96–M'00) received the B.S. degree in electrical engineering from the University of Maryland, College Park, in 1990, and the M.S. and Ph.D. degrees in electrical and computer engineering from the University of California at San Diego (UCSD), in 1998 and 2000, respectively.

From 1990 to 1996, he was a Product Engineer for data converter and computer audio products with Analog Devices, Wilmington, MA, and Brooktree Corporation, San Diego, CA. From 1996 to 2000, he was a Graduate Student Researcher at UCSD. He is currently a Design Engineer with Silicon Wave Inc., San Diego, CA. His research interests include data converter and mixed-signal circuit design for communications ICs.