The Logarithmic Number System for Strength Reduction in Adaptive Filtering

John R. Sacha and Mary Jane Irwin
Computer Science and Engineering Department
The Pennsylvania State University
University Park, PA 16802
Phone: (814) 863-4162
E-mail: sacha@cse.psu.edu

Abstract

An important technique for reducing power consumption in VLSI systems is strength reduction, the substitution of a less-costly operation such as a shift, for a more-costly operation such a multiplication. Using a logarithmic number representation provides several opportunities for strength reductions; in particular, multiplication is performed as the fixed-point addition of logarithms, and extracting a square root is implemented via a shift. These reductions occur transparently at the hardware level; consequently relatively little algorithmic modification is required, and they are readily applicable to adaptive filtering. For performing Givens rotations in the QR decomposition recursive least squares adaptive filter, logarithmic arithmetic is shown to compare favorably to other strength reduction techniques, such as CORDIC arithmetic, in terms of switched capacitance and numerical accuracy.

1 Introduction

Adaptive filtering is a necessary component of many communication and other signal processing systems which must operate in non-stationary environments. Uses include echo cancellation, channel equalization, and system identification. A wide variety of algorithms exist, admitting tradeoffs among such varied criteria as convergence and tracking rates, computational complexity, and numerical stability, in order to meet performance goals and satisfy design constraints. In some recent applications, the traditional VLSI constraints of throughput and area have been joined by the issue of power dissipation. This has been largely motivated by the emergence of portable computing and digitally-based communications devices, where conserving energy is important because of battery limitations. In non-portable computing environments, power usage can also be an issue, since packaging costs and circuit reliability are dependent upon cooling considerations [1].

Low power optimizations take place at many levels, ranging from the gate and circuit levels up through algorithm design. One important higher level technique is the use of strength reduction transformations to replace expensive operations with mathematically equivalent operations that are more conserving of energy [2]. Another technique involves finding a number representation which reduces switching activity. Fixed point representations are the norm for low power applications. A less standard method, logarithmic representation, is potentially attractive for low power because intrinsic to it are several operator strength reductions.

2 Strength Reduction

In well-designed CMOS circuits, the main source of power dissipation is the charging and discharging of node capacitances. In exploring the tradeoffs between alternative implementations of a processing module, an appropriate measure for comparison is the power dissipated over the duration of the access. This gives the *energy per access*,

$$E_m = C_m V_{dd}^2. \quad (1)$$

where C_m is the module *switched capacitance*, and V_{dd} is the supply voltage [3]. The switched capacitance describes the average capacitance charged and discharged per access. Energy usage can be minimized by a combination of reducing V_{dd} and C_m. Reducing V_{dd} increases gate delay, so that architectural restructuring, e.g., pipelining or parallelization, may then be required in order to maintain throughput; substantial scaling of V_{dd} is also technology dependent.

The average power dissipation of various arithmetic operations varies considerably. For example, for a particular CMOS implementation of multiplication and addition in conventional two's complement fixed-point arithmetic, the average switched capacitances can be empirically modeled as [4]

$$C_{mult}(n) \approx 253n^2, \quad (2)$$

$$C_{add}(n) \approx 214n, \quad (3)$$

respectively, where n is the word length in bits.

Optimizations at various levels can be performed in order to address the relatively high cost of multiplies. Refinements at the transistor, gate, and circuit levels can be performed to reduce the size of the capacitance constant [5][6]. At the higher levels of algorithm and system design, reducing the number of multiplies (as well as the even more complicated "inverse" operations of division and square root extraction) was an important consideration even before the advent of major interest in low power designs, due to delay and area

(or discrete component) costs [7]. Such high-level changes can yield big payoffs, but can also modify system characteristics. For example, the replacement of a FIR filters with a shorter IIR design can alter frequency and phase responses, which may or may not be acceptable.

An intermediate approach to reducing the cost of arithmetic operations is strength reduction, the replacement of an operation by a set of mathematically equivalent — but less costly — alternative operations. This generally involves local transformations of the computation flow. The canonical example is the substitution of a few shifts and adds to implement a constant multiplication: since $15 = 2^4 - 1$,

$$15x \quad \rightarrow \quad (x \ll 4) - x. \tag{4}$$

Note that the mathematical structure of the computation is unaltered, but that the numerical behavior and delay might be changed.

There is extensive literature for replacing multiplies with other operators in FIR filtering. One approach is to design filters with coefficients that are powers of two (or sums of a few powers of two), thus allowing general-purpose multiplies to be replaced by a few shifts and adds [8][9]. Note that this technique is not readily applicable to adaptive filters, in which the filter coefficients are not known *a priori*, or may change over time.

One method for obtaining several strength reductions is to employ a logarithmic number representation. By using the Logarithmic Number System (LNS), the hardware complexity of multiplication, division, and square rooting are reduced considerably. These strength reductions occur at the architecture level, without explicit local transformations of the structure of the computation. The potential utility of LNS for power savings is illustrated using an adaptive filtering application.

3 Adaptive Filtering

Consider the system

$$y(t) = \bar{x}(t)^T \bar{w}(t) + \eta(t), \tag{5}$$

where $\bar{x}(t)$ is a length-N vector of known time-varying inputs, $\bar{w}(t)$ is a vector of unknown "slowly varying" weights, $\eta(t)$ is a zero-mean white noise process, and $y(t)$ is the known output. The goal of adaptive filtering is to determine a set of weight estimates, $\hat{w}(t)$, which minimizes the error (under some criterion) between $y(t)$ and $\bar{x}(t)^T\hat{w}(t)$. While the least mean square (LMS) algorithm is typically the first choice due to its $O(N)$ complexity, for some applications it exhibits slower than required convergence, so that alternative algorithms such as recursive least squares (RLS) must be used. The RLS algorithm solves the weighted least squares problem

$$\min_{\hat{w}} \|D(t)[Y(t) - X(t)^T\hat{w}(t)]\|_2^2, \tag{6}$$

where $D(t)$ is a weight matrix (usually $\mathrm{diag}(\lambda^t, \lambda^{t-1}, ..., \lambda)$ for $0 \leq \lambda \leq 1$); $Y(t) = [y(1), y(2), ..., y(t)]^T$, the output history; and $X(t)$ is the matrix of inputs observed through time t, $X(t) = [\bar{x}(1); \bar{x}(2); ...; \bar{x}(t)]$.

An efficient solution to this problem can be obtained by iteratively updating the QR decomposition of $D^{1/2}X^T$, giving the QR decomposition recursive least squares (QR-DRLS) algorithm [10][11]. The QR decomposition can be performed in a variety of ways. One which is amenable to VLSI implementation, and which has received recent attention including low power consideration [11][12][13] is the use of Givens rotations to annihilate the lower triangular entries of $D^{1/2}X^T$. Given a vector $[a\ b]^T$, $b \neq 0$, the Givens transformation annihilates b via [14]

$$\begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \sqrt{a^2+b^2} \\ 0 \end{bmatrix}, \tag{7}$$

where the values c and s represent the cosine and sine of an angle θ. This angle does does not have to be explicitly extracted; a numerically attractive algorithm for determining c and s is:

$$\begin{cases} \text{if } b = 0 & \text{then} \quad c = sign(a);\ s = 0 \\ \text{if } |b| > |a| & \text{then} \quad \tau = -a/b; s = 1/\sqrt{1+\tau^2}; c = s\tau \\ \text{if } |b| \leq |a| & \text{then} \quad \tau = -b/a; c = 1/\sqrt{1+\tau^2}; s = c\tau. \end{cases} \tag{8}$$

Note that determining c and s requires one add, two multiplies (one of which is actually a squaring), two divides, and one square root extraction. The standard method of applying the rotation to an arbitrary vector uses 4 multiplies and 2 adds.

Multiplies, divides, and square roots are considered to be complicated to perform in conventional fixed- and floating-point representations, and algorithmic solutions have been developed to mitigate their effects. For example, square-root free Givens rotations rely upon non-unitary "pseudo-Givens" rotation matrices; in addition to eliminating the need to extract square roots, the number of multiplies per rotation is reduced. This approach may exhibit numeric instablity [14], and will not be addressed here; however, with extra stabilization steps it can perform acceptably well [10].

3.1 Matrix-Vector Multiply Recoding

One form of strength reduction which has been proposed for adaptive filtering converts the straightforward 4-multiply, 2-add implementation of a plane rotation into a 3-multiply, 3-add form [15]. (Actually, this transformation was used for complex multiplication in [2] and [15], but there is a direct correspondence between complex multiplication and 2×2 plane rotation.) Note that

$$\begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} ca - sb + (cb - cb) \\ sa + cb + (ca - ca) \end{bmatrix} \tag{9}$$

$$= \begin{bmatrix} c(a+b) - (c+s)b \\ c(a+b) + (s-c)a \end{bmatrix}, \tag{10}$$

with the savings occurring because $c(a+b)$ has to be performed only once. While this transformed formulation is mathematically identical to the conventional matrix-vector multiply, numerically it may not be, because of the precision potentially lost in forming the intermediate sums $a+b$, $c+s$, and $s-c$. (Also note also that this transformation generates six distinct operands, potentially reducing the correlation between successive multiplier inputs.)

This transformation is applicable to both fixed- and floating-point number representations. Although floating point enjoys widespread use in scientific computing and signal processing because it simplifies programming and provide wide dynamic range, it has so far not found favor in low power applications due to its perceived complexity and overhead [16]. Nonetheless, floating point may provide a power saving, depending upon the precision and dynamic range requirements of a particular application. The underlying mechanism is

that reducing the mantissa length by a factor of a leads to a reduction in multiplicative power dissipation by a factor of a^2, without appreciably changing the dynamic range. The practical question is whether or not this savings offsets the added complexity of other floating-point operations for a particular application.

3.2 CORDIC Arithmetic

Another approach to implementing Givens rotations using conventional fixed-point representations, which has received recent attention is to use CORDIC arithmetic [12] [13][17] [18]. In the CORDIC formulation, the rotation angle θ is decomposed into a sum of predefined "microrotation" angles, $\theta = s_0\theta_0 + \cdots + s_m\theta_m$, $s_i \in \{-1, 1\}$. Strength reduction is achieved by choosing the θ_i such that $\arctan\theta_i$ is a power of two. This allows the microrotation matrix to be rewritten as

$$\begin{bmatrix} \cos\theta_i & -\sin\theta_i \\ \sin\theta_i & \cos\theta_i \end{bmatrix} = \frac{1}{\cos\theta_i}\begin{bmatrix} 1 & -\tan\theta_i \\ \tan\theta_i & 1 \end{bmatrix} \quad (11)$$

$$= K_i\begin{bmatrix} 1 & -s_i 2^{-i} \\ s_i 2^{-i} & 1 \end{bmatrix}, \quad (12)$$

so that apart from the scaling by the unitary normalization constant, $K_i = 1/\sqrt{1+2^{-2i}}$, the product is computable with two shifts and two adds. In standard CORDIC, the requirement $s_i \in \{-1, 1\}$ ensures that the product of the unitary normalizing constants, $K = \prod_{i=0}^{m} K_i$, is a constant which can itself be further decomposed offline into shifts and adds.

Several low power CORDIC implementations have been designed [13] [18]. In one version [13], $s_i \in \{-1, 0, 1\}$, eliminating the need to perform rotations $s_i = 0$, though $\prod_i K_i$ is no longer a constant and must be computed explicitly. To further save power, a truncated CORDIC can be performed, in which only the r most significant non-zero rotations, $r \ll m$, are applied.

4 Logarithmic Number System

In the Logarithmic Number System, a number X is represented by its sign, s_X, and the logarithm to the base β of its absolute value:

$$L_X = \log_\beta |X|. \quad (13)$$

Generally the base used is 2, and the n-bit fixed-point value is treated as consisting of a k-bit integer part and an l-bit fraction with $n = k+l$. LNS has numeric properties similar to those of floating point. The logarithmic system encodes 2^l values in the interval $[2^d, 2^{d+1})$, with distances which are harmonically related by the factor $2^{1/2^l}$, while a floating-point scheme with an l-bit mantissa plus hidden bit would divide the same interval into 2^l equal-sized pieces. This representation similarity means that, like floating point, LNS provides a wider dynamic range than fixed point for a given number of bits, and simplifies algorithm coding helping to keep track of the radix point. Unlike floating point, LNS does not allow double precision accumulation.

The chief attraction of LNS arithmetic, historically as well as in the context of low power design, is that the product of two numbers A and B is computed by the fixed-point addition of their logarithmic representations:

$$s_{AB} = s_A \oplus s_B \quad (14)$$

$$L_{AB} = L_A + L_B, \quad (15)$$

which represents a significant strength reduction. Likewise, division (which in conventional fixed- and floating-point is more complicated than multiplication) is implemented as a subtraction. (If the logarithms are coded in excess notation as is commonly done with floating-point exponents, the overhead is higher since an extra full word add is required to restore the offset — clearly a drawback in the context of low power processing.)

Exponentiation and root extraction are likewise trivial under LNS. Assuming a base 2 logarithm, squaring and square root extraction can be done with 1-bit shifts:

$$L_{A^2} = L_A << 1. \quad (16)$$

$$L_{\sqrt{A}} = L_A >> 1. \quad (17)$$

For square root extraction under LNS, a 20:1 power-delay product improvement over floating-point has been reported [19]. Multiplication, division, and raising a number to an integer power are numerically exact under LNS, barring overflow; some numerical precision can be lost when extracting roots.

Some of the gains achieved above are offset by the increased complexity of addition and subtraction, which are based on the identities

$$A \pm B = A(1 \pm B/A). \quad (18)$$

In logarithmic form,

$$L_{A\pm B} = L_A + U_\pm(L_B - L_A) \quad (19)$$

where

$$U_\pm(L_X) = \log_2(1 \pm 2^{L_X}) \quad (20)$$

The functions U_+ and U_- are typically implemented using table lookup. Thus, logarithm-based addition requires two fixed-point additions and a ROM access, with an additional compare or negation used to force $L_X < 0$, in order to simplify table construction. In general, since the value for $U_\pm(L_X)$ cannot be represented exactly in a $k+l$ bit wide table, a relative error bounded by $2^{-(l+1)}$ is introduced into the final sum. A major challenge to the practical implementation and widespread adoption of LNS for general purpose computing is that table sizes grow exponentially with precision. Much LNS research activity is devoted to table compression in order to allow increased word lengths in an attempt to compete with conventional floating point [20][21].However, LNS seems to be most suited for moderate precision applications requiring high throughput, such as signal processing [7]. Another issue faced by LNS is conversion between LNS and fixed point. In some cases data can be digitized directly into logarithmic form [19].

Logarithmic representation has previously been applied to adaptive filtering using the LMS algorithm [22]. However, given the development of reduced-multiplication variants such as sign-LMS, where additions dominate, using LNS would not seem to be particularly advantageous in this case. In contrast, LNS arithmetic is well-suited to implementing Givens rotations. There are few additions (relatively expensive in LNS) in comparison to the number of multiplications, divisions, and square roots, which are cheap in LNS. The crucial issue regarding the practicality of LNS lies in the number of bits required to represent the logarithms. If precision requirements are excessive, then coding the tables for $U_\pm$ entails excessive area and/or the need for expensive interpolation computations.

5 Simulation

To study the tradeoffs between performance and power resulting from the various formulations presented above, we applied the following instances of QRDRLS algorithm implementations to the standard channel equalization example of [10][11], with $N = 11$, $SNR = 0.001$, and $\lambda^{1/2} = 0.9375$:

- 4-multiply and 3-multiply Givens formulations using 16-bit fixed-point arithmetic;
- 4-multiply and 3-multiply Givens formulations using floating-point arithmetic having 5-, 6-, 8-, and 10-bit mantissas (plus a hidden bit);
- Full and truncated CORDIC (two and three term), as described in [13], using 16-bit fixed-point arithmetic;
- LNS arithmetic with 5-,6-,8-, and 10-bit fractional parts.

5.1 Algorithm Performance

Ensemble averages for fifty trials of each configuration were generated. A time-average of the steady-state squared *a priori* error (i.e., after convergence) was used as the measure of algorithm performance, as shown in Figure 1.

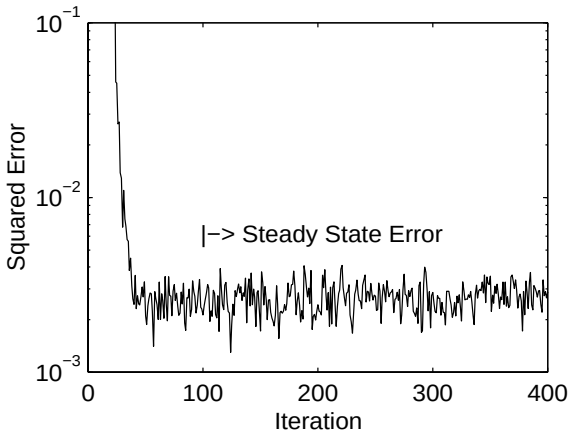

Figure 1: *A Priori* Residual Error for Fixed-Point

5.2 Power Estimation

Architecture-level estimations of switched capacitance were used to assess the relative power usage among the various computational constructs discussed. The two high-level capacitance models used wll be referred to as the "SRB-Model"[23], and the "CB-Model"[4]. These are bit-independent models comprised of algebraic formulas, such as Eqs. (1)-(3), relating "average" switched capacitances of arithmetic circuits to elementary architectural features such as word length. The total switched capacitance of a computation is obtained by summing the individual module capacitances weighted by the number of accesses. Although less accurate than gate or transistor level estimates, such high level analysis can serve as a rough guide to possible design tradeoffs. In estimating the switched capacitance for a QR update, only the calculation and application of the Givens rotations were considered; the backsubstitution to explicitly obtain the weights was not included. For the conventional fixed-point and floating-point versions, the multiplier capacitance model was assumed for division and square root extractions; each of the latter two represent less than two percent of the operations performed. Capacitances for the floating-point operations were estimated by summing their constituent fixed-point components, such as exponent additions, and the adds, multiplies and shifts of the mantissas. The CORDIC estimates were based on counts of adds and shifts, which included frequencies of shift values.

Modeling LNS capacitance was somewhat more involved because of the table lookups. The effective ROM switched capacitance in logarithmic addition table lookup depends upon the ROM organization. In the SRB-Model, the capacitance switched by a memory access is a function of total memory size, with three submodels provided; the submodel used here assumes that a memory access is proportional to the square root of the total memory size. Several assumptions were used to estimate memory size. First, since the problem under consideration was shown to involve limited precision, it was assumed that linear interpolation was unnecessary. Second, each unit interval in the domain of U_+ and U_- was given its own ROM bank, whose word length was no wider than necessary to encode the function for that interval [19]. (For a fraction length of l, there would be $M = 2^l$ entries per ROM.) Because the two functions decrease in magnitude as L_X decreases, average ROM width is reduced, decreasing both the area and the length of the active access lines [24][25]. Last, a modest compression scheme was assumed. For almost all blocks, the magnitudes of the derivatives of U_+ and U_- are less than unity, ensuring that stored values can change only by the value of the *lsb* between adjacent entries. The $M \times b$-bit ROMs are thus convertible to $M/2 \times (b+1)$-bit ROMs by tabulating only the even-numbered entries and storing one additional bit per entry to indicate if the next (untabulated) odd-numbered entry is different; the $b+1$st bit serves as a carry into the final adder stage. For 10-bit fraction LNS, the tables were estimated to require 72 kilobits.

For the CB-Model, an average capacitance for each ROM block was computed individually, using the formula for register file access. It was assumed that larger ROMs were subdivided into blocks of 128 entries. Access patterns were computed during the course of the simulations to estimate values for p_j, the probability that the jth ROM block is accessed. This yielded a capacitance model of:

$$C_{add}^{LNS}(k,l) \approx 3C_{add}(k+l) + \sum_j p_j C_{mem}^{[j]}, \qquad (21)$$

where $C_{mem}^{[j]}$ is the capacitance of the jth ROM. Figure 2 shows the relative capacitances of ROM bank accesses by unit interval for a 12-bit fraction LNS implementation, and Figure 3 shows corresponding ROM access frequencies.

5.3 Results

Figures 4 and 5 show the average steady-state error achieved by each Givens rotation implementation plotted as functions of the corresponding switched capacitances estimated from the two power models. (Estimates for 3-multiply Givens rotations are shown as triangles, and those for 4-multiply rotations are indicated by squares.) The capacitance figures are for the data path only, and do not include such overheads as clocking, data fetch, or decoding. Also, any needed cost of conversion to/from LNS is not included. Although the numeric values for the two plots differ, the overall orderings

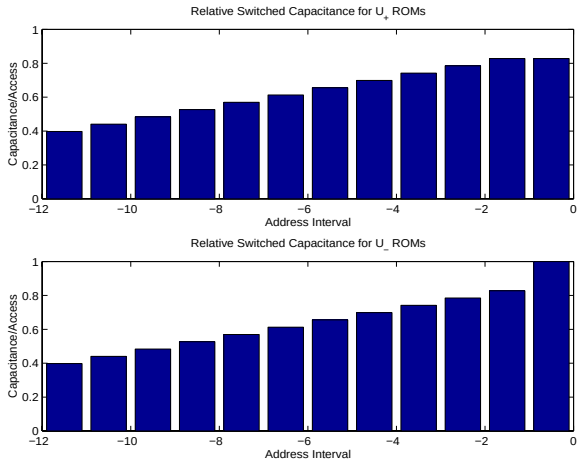

Figure 2: Relative Switched Capacitances for ROM Banks

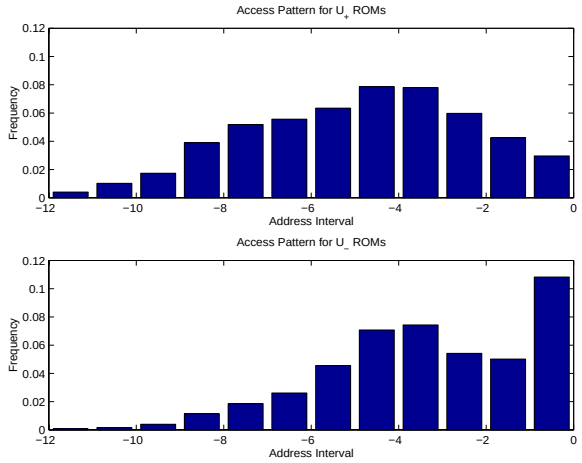

Figure 3: ROM Bank Access Frequencies

are consistent from plot to plot, with LNS using the least power and conventional fixed point the most.

- The conventional fixed-point implementations switched the most capacitance. As expected, the 3-multiply Givens used about 20% less power, at a cost of a minor increase in error.
- The CORDIC formulations used less power than the conventional fixed point, but were less accurate. The squared residual error was greater for the the two-term truncated CORDIC ($r = 2$), than for the three-term truncated CORDIC ($r = 3$) or full CORDIC. While it may not be readily apparent from the plot, the three-term mean error averaged over time was actually slightly lower than that of the full CORDIC, although an examination of residual error time series revealed that the former exhibited a higher variance. None of the CORDIC methods achieved as low an error as the conventional 16-bit implementations.
- Floating point outperformed fixed-point and CORDIC. As with the fixed-point case, each 3-multiply Givens used less power than its 4-multiply counterpart, but produced an increase in the steady-state residual error. Although floating point used more power than LNS, it does not require lookup tables, and may therefore be a good candidate in its own right.
- LNS exhibited the best overall accuracy and cost. It was about six times less costly than conventional fixed point, even with the elimination of one multiply per Givens rotation in the fixed point implementation. As might be expected, larger fractional parts yielded lower errors, up to a point. The steady-state error for a 6-bit fraction was better than those of the best CORDIC formulations, while LNS with a 10-bit fraction yielded a residual error less than those of the 16-bit fixed-point implementations.

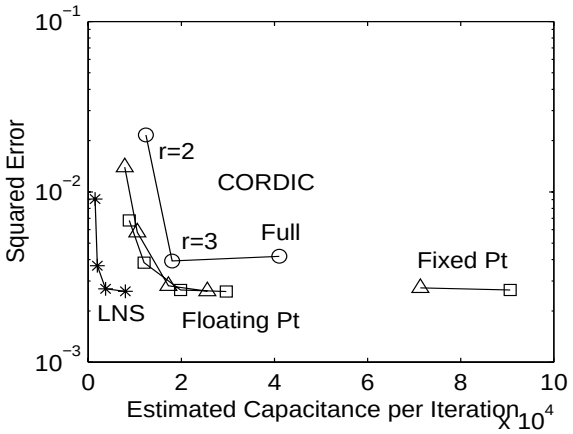

Figure 4: SRB Capacitance Model Results

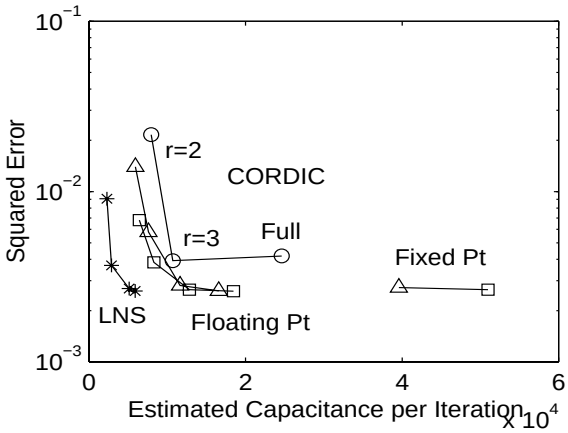

Figure 5: CB Capacitance Model Results

The use of bit-independent capacitance models can serve to reveal general trends and perform rough comparisons. In real systems, however, implementation details and data correlations come into play. For instance, in the 4-multiply Givens rotation, performing the multiplies in the "natural order" ca, sb, cb, sa can cause both inputs to the multiplier to change (assuming a single multiplier). In contrast, only a single input changes each time when performing the multiplies in the order ca, cb, sb, sa. By using a bit-dependent

power model [3], it was observed that the second ordering reduced switched capacitance by 10% relative to the first. As previously noted, the 3-multiply transformation generates six distinct multiplier inputs; even though it requires only three multiplies, the bit-dependent model indicates that it further reduces switched capacitance by only 6% relative to the better of the 4-multiply Givens rotation orderings.

6 Conclusions

Several techniques for performing QRDRLS adaptive filtering were compared with regard to numeric accuracy and energy consumption. These included conventional fixed- and floating-point configurations, CORDIC arithmetic, and logarithmic representations. Architecture-level power modeling showed that logarithic arithmetic switched less capacitance for a given residual error level than did the other methods tested. The use of floating point also yielded energy savings. CORDIC was the least accurate, while fixed-point exhibited high accuracy but consumed the most energy. The logarithmic number system (LNS) and floating point have similar dynamic range and precision properties. In the context of low power, the chief attraction of LNS is that multiplication is performed as a fixed-point addition, reducing overall switching activity for multiplication-intensive operations such as Givens rotations. Drawbacks to the use of LNS include potentially large memory requirements, and the cost of conversion between logarithmic and more conventional representations.

References

[1] M. Pedram, "Power Minimization in IC Design: Principles and Applications," *ACM Trans. Design Auto. Electr. Sys.*, Vol. 1, No. 1, Jan. 1996.

[2] A. P. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey, R. W. Broderson, "Optimizing Power Using Transforms," *IEEE Trans. CAD*, Vol. 14, No. 1, Jan. 1995, pp. 12-31.

[3] H. A. Mehta, "System Level Power Analysis," Ph.D. Dissertation, Computer Science and Engineering Department, The Pennsylvania State University, University Park, PA, 1996.

[4] A. K. Chandrakasan, R. W. Broderson, *Low Power Digital CMOS Design*, Kluwer, 1995, pp. 282-283.

[5] T. K. Calaway, E. E. Swartzlander, Jr., "Power-Delay Characteristics of CMOS Multipliers," *Proc. 1997 Int. Symp. Comp. Arith.*, pp. 26-32.

[6] C. J. Nicol, P. Larsson, "Low Power Multiplication for FIR Filters," *Proc. 1997 Int. Symp. Low Power Elec. Design*, pp. 76-79.

[7] G. A. Jullien, "High Performance Arithmetic for DSP Systems," in *VLSI Signal Processing Technology*, M. A. Bayoumi and E. E. Swartzlander, Jr. eds., Kluwer, 1995, pp. 59-96.

[8] G. Wade, *Signal Coding and Processing*, Cambridge University Press, 1994, pp. 247-248, 291-301.

[9] B. W. Wah, Y. Shang, Z. Wu, "Discrete Lagrangian Method for Optimizing the Design of Multiplierless QMF Filter Banks," *Proc. 1997 IEEE Int. Conf. ASAP*, July 14-16, 1997.

[10] N. Kalouptsidis, S. Theodoridis, eds., *Adaptive System Identification and Signal Processing Algorithms*, Prentice-Hall (UK), 1993, pp. 7-14, 123-130, 260-321.

[11] S. Haykin, *Adaptive Filter Theory*, Prentice-Hall, Englewood Cliffs, NJ, pp. 244ff.

[12] J. Ma, E. F. Deprettere, K. K. Parhi, "Pipelined Cordic Based QRD-RLS Adaptive Filtering Using Matrix Lookahead," *1997 IEEE Workshop on Sig. Proc. Sys.*, pp. 131-140.

[13] B. Haller, J. Götze, J. R. Cavallaro, "Efficient Implementation of Rotation Operations for High Performance QRD-RLS Filtering," *Proc. 1997 IEEE Int. Conf. ASAP*, July 14-16, 1997, pp. 162-174.

[14] E. Golub, C. F. Van Loan *Matrix Computations, 2nd Ed.*, Johns Hopkins University Press, 1989, pp. 200-210.

[15] M. Goel, N. R. Shanbhag, "Low-power Adaptive Filter Architectures via Strength Reduction," *Proc. 1996 Int. Symp. Low Power Elec. Design*, Aug. 1996, pp. 217-220.

[16] K. P. Acken, "Low Power Architectural Optimizations for 3D Graphics Subsystems," Ph.D. Dissertation, Computer Science and Engineering Department, The Pennsylvania State University, University Park, PA, 1997.

[17] E. Antelo, J. Villalba, J. D. Bruguera, E. L. Zapata, 'High Performance Rotation Architectures Based on the Radix-4 CORDIC Algorithm,"' *IEEE Trans. Comp.*, Vol. 46, No. 8, August 1997, pp. 855-870.

[18] C. V. Schimpfle, S. Simon, J. A. Nossek, "Low Power CORDIC Implementation Using Redundant Number Representation," *Proc. 1997 Int. Conf. Appl.-Specific Sys., Arch. and Proc.*, July 14-16, 1997, pp. 154-161.

[19] F. J. Taylor, R. Gill, J. Joseph, J. Radke, "A 20 Bit Logarithmic Number System Processor," *IEEE Trans. Comp.*, Vol. 37, No. 2, February 1988, pp. 190-200.

[20] D. M. Lewis, "An Architecture for Addition and Subtraction of Long Word Length Numbers in the Logarithmic Number System," *IEEE Trans. Comp.*, Vol. 39, No. 11, November 1990, pp. 1325-1336.

[21] J. N. Coleman, "Simplification of Table Structure in Logarithmic Arithmetic," *Electron. Let.*, Vol. 31, No. 22, October 26, 1995, pp. 1905-1906.

[22] V. P. Shenoy, F. J. Taylor, "Error Analysis of LMS Adaptive Digital Filter Implemented with Logarithmic Number System," *Proc. ICASSP'84*, pp. 30.10.1-30.10.4.

[23] N. Sankarayya, K. Roy, D. Bhattacharya, "Algorithms for Low Power and High Speed FIR Filter Realization Using Differential Coefficients," *IEEE Trans. Circ. Sys.*, Vol. 44, No. 6, 1997, pp. 488-497.

[24] P. E. Landman, J. M. Rabaey, "Architectural Power Analysis: The Dual Bit Type Method," *IEEE Trans. VLSI Sys.*, Vol. 3, No. 2, June 1995, pp. 173-187.

[25] E. de Angel, E. E. Swartzlander, Jr., "Survey of Low Power Techniques for ROMs," *Proc. 1997 Int. Symp. Low Power Elec. Design*, Aug. 1997, pp. 7-11.