J Sign Process Syst (2010) 58:29–40
DOI 10.1007/s11265-008-0282-7

A Novel Cotransformation for LNS Subtraction

Panagiotis D. Vouzis · Caroline Collange · Mark G. Arnold

Received: 31 October 2007 / Revised: 3 June 2008 / Accepted: 12 September 2008 / Published online: 16 October 2008

Abstract The Logarithmic Number System (LNS) can be considered a simplification of the Floating Point (FP) Number System that assumes the mantissa is always equal to one, and has a binary fixed-point exponent. LNS converts multiplication/division to a single addition/subtraction, which make LNS a very attractive choice for applications where these operations predominate, such as in some signal-processing algorithms. However, for wordlengths greater than 20 bits LNS becomes expensive because of the hardware-demanding LNS operations of addition and subtraction, which are typically important for most signal-processing algorithms. This paper gives an overview of the family of LNS subtraction algorithms called "Cotransformations," and proposes a "Novel Cotransformation Combination" that offers improvements in terms of area and speed without sacrificing accuracy compared to previous methods. The hardware requirements of the proposed method are analyzed mathematically, and the results are verified by using synthesis and simulations.

Keywords Logarithmic number system · Multipartite tables · Interpolation · Cotransformation · Hardware description languages

P. D. Vouzis (✉) · M. G. Arnold
Department of Computer Science & Engineering,
Lehigh University, Bethlehem, PA 18015, USA
e-mail: vouzis@lehigh.edu

M. G. Arnold
e-mail: maab@lehigh.edu

C. Collange
Université de Perpignan Via Domitia,
52 avenue Paul Alduy, F-66860 Perpignan Cedex, France
e-mail: caroline.collange@univ-perp.fr

Present Address:
P. D. Vouzis
Department of Chemical Enginering,
Carnegie Mellon University, Pittsburgh, PA 15213, USA
e-mail: pvouzis@cmu.edu

1 Introduction

The Floating Point (FP) Number System [1] and the Logarithmic Number System (LNS) [2] are two alternatives for applications that require an increased dynamic range for their arithmetic operations, i.e., a fixed-point number system is not adequate. Both the dynamic range and the accuracy of the two alternatives can be adjusted by choosing the wordlength of their representations appropriately. For a particular application, the choice of the most suitable arithmetic system depends largely on the implementation requirements of area, delay, and accuracy.

The difference between LNS and FP is that LNS uses a fixed-point representation of the logarithm whose real value, unlike FP, does not require the steps of shifting, normalization, postnormalization, etc., to carry out any arithmetic operation. LNS is more efficient for multiplication, division and powering, since these are reduced to addition, subtraction, and multiplication. The difficulty with LNS lies in the operations of addition and subtraction where the functions $s_b(z) = \log_b(1 + b^z)$ and $d_b(z) = \log_b(1 - b^z)$ are needed. The memory requirements for these two functions grow exponentially with respect to the fractional bits of the representation, rendering an LNS implementation less efficient than an FP one as the wordlength increases.

Although the penetration of LNS in industrial applications is low, mainly due to the absence of a standard (like the IEEE-754 for FP), there are several implementations that use LNS, and it has been proven a viable alternative for special-purpose-hardware designs. For the Fast Fourier Transform (FFT) in [3] it is shown that an LNS implementation uses a smaller word size than a comparable fixed-point system, while achieving the same error performance. LNS has also been used for matrix [4] and filter [5] applications. Bleris et al. [6] show that a 16-bit LNS Application-Specific Integrated Processor (ASIP) is capable of implementing Model Predictive Control (MPC) in embedded applications, a possibility hindered previously by the high computational requirements of MPC using FP arithmetic. LNS is useful for multimedia applications, such as Wrigley's ray-tracing engine [7]. The most notable and widespread uses of LNS are the GRAPE supercomputers designed by Makino et al. [8], which won the Gordon Bell Prize award in 1995, 1996, and 1999 for the computation of N-body gravitational forces among stars, and Hidden Markov Modeling (HMM) for computing log-probabilities in applications like speech recognition [9] and DNA sequencing [10]. It is noteworthy that GRAPE and HMMs use only LNS addition; it appears the difficulty of LNS subtractions has precluded a wider range of practical LNS applications which may become more cost effective with the cotransformation technique described here.

There are numerous studies that compare FP against LNS. Among others, in [11] it is shown how the ratio of the number of additions over the number of multiplications of an algorithm affects the area and the latency of both LNS and FP for a Field-Programmable-Gate-Array (FPGA) implementation. In Collange et al. [12] compare LNS against FP in terms of area, latency and accuracy for simple arithmetic manipulations. They show that the LNS-FP library used to solve two arithmetic case studies renders LNS smaller and faster for precisions smaller than 15 bits, and faster for precisions up to 23 bits, while their FP implementation produces slightly better visual quality than their LNS implementation for a toy-graphics benchmark application, a conclusion shown here to relate to their LNS implementation [13] rather than a fundamental deficiency in LNS itself. Another work that compares LNS against FP for FPGAs is by Fu et al. [14] where it is shown that a polynomial-approximation LNS-arithmetic unit can have better error characteristics than an FP unit with a penalty in area.

A substantial effort to explore the applicability of LNS as a viable alternative to FP for general-purpose processing of single-precision real numbers is described in the context of the European Logarithmic Microprocessor (ELM) [15]. A fabricated prototype of the processor, which has a 32-bit cotransformation-based LNS Arithmetic Logic Unit (ALU), shows that LNS can have "better than FP" accuracy, with improved speed.

The successful implementation of the ELM in terms of speed and accuracy is attributed to the adoption of cotransformation for the implementation of the LNS subtraction unit in the microprocessor, which reveals the potential of this method. Since the appearance of the first cotransformation technique in 1995 [16] different versions have been proposed that constantly improve the area, speed and accuracy characteristics of the LNS subtraction implementation which is the bottleneck of any LNS circuit. Consequently, any improved cotransformation makes LNS a more attractive arithmetic system for real-number applications.

In this context, this paper, being in part an extension of the work in [17], gives an overview of the family of cotransformation algorithms, and it presents a novel cotransformation technique that reduces the complexity of an earlier cotransformation [18] in terms of area and delay. Moreover, we study, by simulation, how many guard bits are required by this new cotransformation to guarantee faithful rounding, and we show that, even without using any guard bits, the error behavior of the LNS subtraction is improved considerably compared to the library in [19]. Additionally, cotransformation reduces the area of the LNS subtraction, with the drawback of an increase in the latency of the circuit due to the multiple tables required [18]. However, a designer interested in reduced area and, more importantly, interested in increased accuracy of an LNS unit should consider the alternative of cotransformation. The scope of this work is to broaden the design space of LNS implementations by presenting a new cotransformation technique that leads to a more economical implementation without sacrificing accuracy or performance compared to previous cotransformation methods.

2 Basic Addition and Subtraction

Given $x = \log_b |X|$ and $y = \log_b |Y|$, logarithmic addition can be performed as

$$\log_b (|X| + |Y|) = \max(x, y) + s_b(-|x - y|), \tag{1}$$

or as

$$\log_b (|X| + |Y|) = \min(x, y) + s_b(|x - y|), \tag{2}$$

where $s_b(z) = \log_b(1 + b^z)$. Both (1) and (2) are valid because of commutativity, which in the logarithmic domain is reflected by the relationship

$$s_b(z) = s_b(-z) + z. \tag{3}$$

If we note that $|x - y| = \max(x, y) - \min(x, y)$, substituting (3) into (1) yields (2), and vice versa. In practice, an implementation uses either (1) or (2), and needs to tabulate $s_b(z)$ only for negative z (or only for positive z), thereby cutting the number of words required in half. Since $0 < s_b(z) \leq 1$ for $z \leq 0$, most hardware implementations have used (1) because no bits are required for storing the integer part of $s_b(z)$ for $z < 0$. In contrast, (2) requires typically three to five extra bits per word tabulated, which usually amounts to a 10- to 20-percent memory increase. In software implementations [20], where the word size is fixed to be larger than that required for the tabulation, there is no advantage to (1) over (2).

In an algebraically analogous way, logarithmic subtraction can be performed as

$$\log_b ||X| - |Y|| = \max(x, y) + d_b(-|x - y|), \tag{4}$$

or as

$$\log_b ||X| - |Y|| = \min(x, y) + d_b(|x - y|), \tag{5}$$

where $d_b(z) = \log_b(1 - b^z)$. There is only a single bit (the sign bit) of memory savings by choosing (4) over (5) because $d_b(z) < 0$ for $z < 0$, which is not true for $z > 0$. Prior implementations have chosen to use consistent addition and subtraction techniques. In other words, prior LNS implementations have used either (1) together with (4), which is the most common choice, or, more rarely some implementations [20, 21] have used (2) together with (5). This paper will deviate from this assumption of consistent addition and subtraction techniques in order to explore a novel technique that combines (1) with (5). We can generalize (1), (2), (4), and (5) as

$$t = \log_b ||X| \pm |Y|| = w + f(z, z_s), \tag{6}$$

where w is either $\max(x, y)$ or $\min(x, y)$, f is either s_b when $z_s = 0$ or d_b when $z_s = 1$, and z is either $|x - y|$ or $-|x - y|$ depending on which of the (1), (2), (4) or (5) is used. The appropriate values for w and z are calculated by a preprocessing circuit and they are fed either to the s_b or the d_b function depending on whether the sum ($z_s = 0$) or the difference ($z_s = 1$) is sought respectively. In parallel with w and z, the preprocessing circuit calculates t_s which is the value sign of the result t.

The signed LNS involves a comparison of the value-sign bits, sign(X) and sign(Y) (which are distinct from the logarithm-sign bits, sign(x) and sign(y)). Assuming the LNS functions are tabulated for $z \leq 0$, the LNS "+" operation uses (1) when the value-sign bits are the *same*, thereby evaluating s_b. The identical LNS "−" operation uses (4) when the value-sign bits are *different*, thereby evaluating d_b. Of course, if the LNS functions are tabulated for $z \geq 0$, (2) and (5) are used instead, but the selection of s_b for same signs and d_b for different signs is similar. In a typical LNS implementation, $z_s =$ sign(X) XOR sign(Y).

3 Techniques for LNS Addition and Subtraction

The bottleneck of an LNS addition/subtraction circuit is the calculation of the s_b and the d_b. In order to achieve an efficient implementation of these functions the techniques of interpolation [22], multipartite tables [23] and cotransformation [24] have been proposed. Linear interpolation, which uses two tables (one for the values of the function and one for the slopes) and a multiplier, can be applied adequately for the s_b function since $0 < s'_b(z) < 1$, where the prime denotes the derivative $s'_b(z) = b^z/(b^z + 1)$. However, since $d'_b(z) = -b^z/(1 - b^z)$, $d_b(0) = d'_b(0) = d''(0) = -\infty$, interpolation becomes very expensive close to zero if the accuracy in this region must be kept consistent with the rest of the domain. For example, Lewis [22] uses over ten times as much storage to interpolate d_b than to interpolate s_b.

The multipartite-tables technique [23, 25] is an implementation of interpolation which uses multiple tables addressed by overlapping subsets of bits to evaluate the function without a multiplier. The benefit of this approach is the reduced latency achieved by the elimination of the multiplier, which is counterbalanced by an increase in the area of the circuit. The increased memory requirements of multipartite tables become more pronounced in the singularity region of d_b close to zero making them suffer from the same storage problem with interpolation when consistent accuracy is sought. The LNS library in [19] relaxes the accuracy requirements of the d_b implementation in this region in order to avoid excessive memory size for the storage of the tables.

Other generic-function hardware-evaluation techniques that potentially can be used for the s_b and the d_b functions are the recently introduced High-Order Table-Based Method (HOTBM) [26], and the Non-Uniform Segmentation Function Evaluation [27]. The HOTBM approximates an elementary function by

an arbitrary-order interpolation in combination with multipartite-like tables at a given precision that guarantees faithful rounding [22, 28, 29]. Although the HOTBM can be used for the direct evaluation of the s_b and the d_b functions, the singularity of the d_b would again cause increased hardware requirements for a faithful evaluation. A different approach is followed in [27] where Lee et al. present a low-precision but high-speed and small-area polynomial approximation technique which uses adjustable interpolation intervals on elementary and compound functions.

The accuracy and area problem of interpolation and multipartite tables can be mitigated by the technique of cotransformation proposed initially in [16], which is dubbed "Coleman's Cotransformation." This method transforms the subtraction to

$$d_b(z) = d_b(z_l) + d_b(z_l + d_b(z_h) - d_b(z_l)), \tag{7}$$

where z_l consists of the last j bits of z and z_h consists of the remaining $(k + f - j)$ bits of z (z is a two's-complement number with k integer and f fractional bits). Although d_b is more complicated (requiring $d_b(z_h)$, $d_b(z_l)$ and $d_b(z_l + d_b(z_h) - d_b(z_l))$, it avoids the singularity by transferring the evaluation of d_b away from zero (for an algebraic proof of this property the interested reader is referred to [16]).

Another cotransformation is presented in [21], which is dubbed "Arnold's Cotransformation." It is proven that

$$d_b(z) = d_b(z_l) + s_b(z_l + d_b(z_h) - d_b(z_l)) \tag{8}$$

for $\text{sign}(z_h) = \text{sign}(z_l)$. In a practical system, the terms $d_b(z_h)$ and $d_b(z_l)$ can come from small lookup tables, but the s_b evaluation is performed by an approximation method like interpolation or multipartite tables. In other words, the more difficult evaluation of the d_b function can always be turned into an evaluation of the s_b function of a transformed argument plus an extra term obtained from small tables. The name "cotransformation" was proposed for this technique in [21] because the hardware that implements it uses the unmodified $w = \min(x, y)$, $z = |x - y|$ and $z_s = 0$ for LNS addition, but w, z and z_s are cotransformed together to be $\hat{w} = d_b(z_l) + \min(x, y)$, $\hat{z} = z_l + d_b(z_h) - d_b(z_l)$ and $\hat{z}_s = 0$ instead of the original $w = \min(x, y)$, $z = |x - y|$ and $z_s = 1$ for LNS subtraction. The notations $\hat{w}$, $\hat{z}$ and $\hat{z}_s$ denote the cotransformed value of the variables w, z and z_s. In theory, there are special cases for $z_h = 0$ and $z_l = 0$, but in practice, these can be implemented by storing sufficiently negative but finite values to approximate $-\infty$.

An advantage of "Arnold's Cotransformation" is that all subtractions can be transformed into additions (that avoid the singularity), although, if desired, the cotransformation may be limited only to values of z close to the singularity. Also, another advantage is that the original argument and the transformed argument are both positive (i.e., $z > 0$ means $\hat{z} > 0$). The disadvantage, as mentioned above, is that this case increases the bit width of the s_b table.

The "Improved Cotransformation" in [24] combines the advantages from [21] and [16] by using $z < 0$ and by calculating the d_b function through the s_b as follows:

$$d_b(z) = z + \mathrm{F}_1(z_h) + s_b(\mathrm{F}_2(z_l) - z - \mathrm{F}_1(z_h)), \quad z_h \neq -\delta_h \wedge z \neq -2\delta_h \tag{9}$$

$$d_b(z) = \mathrm{F}_2(z_l), \quad z_h = -\delta_h \tag{10}$$

$$d_b(z) = d_b(-2\delta_h), \quad z = -2\delta_h, \tag{11}$$

where $\delta_h = 2^{j-f}$, $\mathrm{F}_1(z_h) = d_b(-z_h - \delta_h)$ and $\mathrm{F}_2(z_l) = d_b(z_l - \delta_h)$. The special case (10) guarantees that the argument of s_b is not infinity, and the special case (11) guarantees that the argument of s_b remains negative. Although the accuracy of this cotransformation is studied in [24] and it is shown to be better than "Coleman's Cotransformation" at no extra cost, the special case (10) is not accurately stated. This is corrected by the "Vouzis' Cotransformation" in [18], where it is proven that the argument of s_b becomes positive not only for $z = -2\delta_h$, but also for more values of z, which depend on the choice of j. Nonetheless, if the value of j is chosen appropriately, the hardware required to deal with "Vouzis' Cotransformation" is minimal and the error characteristics of the cotransformation in [24] do not change.

There is no prior technique to evaluate the d_b function directly that avoids the increased hardware requirements posed by the singularity, but rather the different cotransformation techniques mitigate the negative effects of this problem, by bypassing this issue with the transfer of the evaluation of the d_b function away from zero, while keeping the accuracy constant in the whole domain of z. In [18] it has been shown that interpolation and multipartite tables can benefit in terms of accuracy and area by adopting cotransformation, while paying a penalty in speed.

In all of the previous techniques for addition and subtraction a considerable memory reduction can be achieved by exploiting the convergence of both s_b and d_b to zero as $z \to -\infty$. The two functions become essentially zero below particular values of z, denoted as e_{s_b} and e_{d_b} respectively: $s_b(e_{s_b}) = 2^{-f-1} \approx 0$ and

$d_b(e_{d_b}) = 2^{-f-1} \approx 0$. The functions need to be tabulated only up to their essential zero. For s_b, $e_{s_b} = \log_b(b^{2^{-f}} - 1) \approx -f$, thus only $f \cdot 2^f$ words are actually used. Similarly, $e_{d_b} = \log_b(1 - b^{-2^{-f}}) \approx -f$.

4 Novel Cotransformation Combination

4.1 Algebraic Derivation

Unlike all the previous works that have used the same sign of z for both LNS addition and subtraction, a new implementation alternative that has not been considered before is to use $z < 0$ for LNS addition, which minimizes the bit-width in the s_b table, but to use $z > 0$ for LNS subtraction, which allows the use of "Arnold's Cotransformation" without any special cases.

Using the identity $s_b(z) = s_b(-z) + z$ in "Arnold's Cotransformation," we have

$$\begin{aligned} d_b(z) &= d_b(z_l) + s_b(z_l + d_b(z_h) - d_b(z_l)) \\ &= d_b(z_l) + z_l + d_b(z_h) - d_b(z_l) \\ &\quad + s_b(-(z_l + d_b(z_h) - d_b(z_l))) \\ &= z_l + d_b(z_h) + s_b((d_b(z_l) - z_l) - d_b(z_h)). \end{aligned} \tag{12}$$

In order to prove that the argument of $s_b(z)$ has the desired property of being negative we assume that $z_h > 0$ and $z_l > 0$. The identity $d_b(z) - z = d_b(-z)$ for $z = z_l$ makes the argument $(d_b(z_l) - z_l) - d_b(z_h) = d_b(-z_l) - d_b(z_h)$, which is negative for $z_h > z_l$. In this novel method, for LNS addition we use

$$\log_b ||X| + |Y|| = \max(x, y) + s_b(-|x - y|), \tag{13}$$

and for LNS subtraction

$$\begin{aligned} \log_b ||X| - |Y|| &= \min(x, y) + z_l \\ &\quad + d_b(z_h) + s_b(d_b(z_l) - z_l) - d_b(z_h)), \end{aligned} \tag{14}$$

which map on the generic formula $t = \log_b ||X| \pm |Y|| = w + f(z, z_s)$ by taking for LNS addition

$$w = \max(x, y), \quad z = -|x - y| \quad \text{and} \quad z_s = 0 \tag{15}$$

and for LNS subtraction the cotransformed

$$\hat{w} = \min(x, y) + z_l + d_b(z_h), \quad \hat{z} = |x - y| \quad \text{and} \quad \hat{z}_s = 0. \tag{16}$$

In other words, the preprocessing block provides the negative absolute difference and the $\max(x, y)$ for addition, and the positive absolute difference and $\min(x, y)$ for subtraction. The partitioning in z_h and z_l is based on the positive absolute difference, since that is only when cotransformation in applied for the substitution of $d_b(z)$. In practice, the preprocessing block described in the literature consists of two subtractors and two muxes in parallel. Changing whether a positive or negative absolute value is provided is simply a matter of exclusive-ORing the select lines of the muxes with the original z_s. A similar trivial change selects $\max(x, y)$ for addition and $\min(x, y)$ for subtraction. A slight improvement in the critical path for this circuit can be obtained by storing

$$\mathrm{F}_3(z_l) = d_b(-z_l) \tag{17}$$

in one Look-Up Table (LUT), since this way we avoid the subtraction in $d_b(z_l) - z_l$, and

$$\mathrm{F}_4(z_h) = d_b(z_h) \tag{18}$$

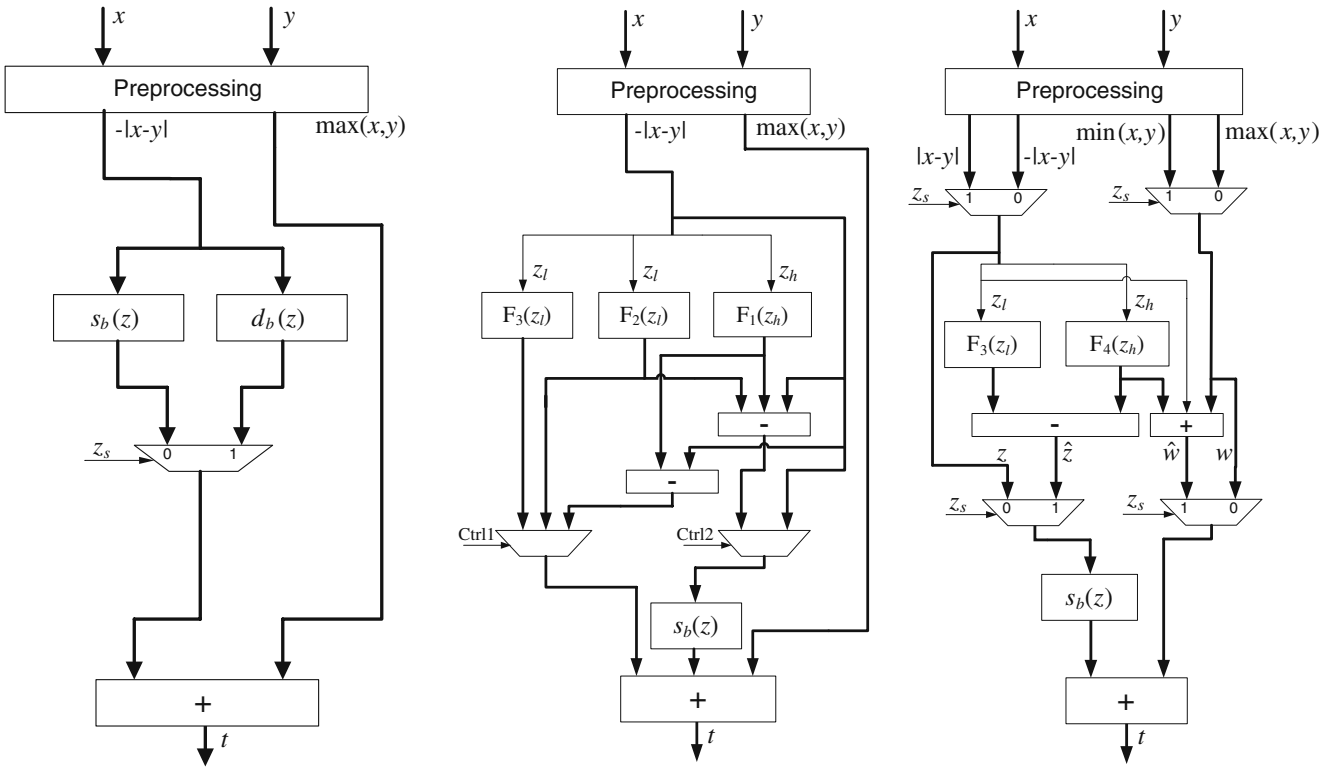
in the other. In this case, the cotransformed arguments for LNS subtraction are

$$\hat{w} = z_l + \mathrm{F}_4(z_h) + \min(x, y) \quad \text{and} \quad \hat{z} = \mathrm{F}_3(z_l) - \mathrm{F}_4(z_h). \tag{19}$$

Figure 1c depicts the architecture of this new cotransformation dubbed "Novel Cotransformation Combination" because it combines positive and negative z for subtraction and addition. The so-called "Vouzis' Cotransformation" technique described by Eqs. (9)–(11) and amended in [18] is depicted in Fig. 1b which requires three tables and additional circuitry to accommodate its increased complexity, resulting in an overall area increase as analyzed in Section 4.4. Finally, Fig. 1a presents a conventional LNS adder/subtractor, where it can be observed that the $d_b(z)$ function is implemented by a single table. In this case, the absence of the multiplexers and adders, appearing in the previous two implementations, results in a circuit with smaller latency, but the tabulation of the $d_b(z)$ requires more area overall [18].

4.2 Eliminating Special Cases

Two minor modifications to the hardware eliminate the special cases for $z_h = 0$ and $z_l = 0$ required by "Arnold's Cotransformation" and inherited by the "Novel Cotransformation Combination." First we need to define the interpolator that is used for the implementation of the s_b function for $z < 0$. In unpartitioned interpolation only certain values, $s_b(z_{\mathrm{MSB}})$, of the function are stored in a memory, and in between these values the function is evaluated by using $s_b(z) = s_b(z_{\mathrm{MSB}}) + s'_b(z_{\mathrm{LSB}} + \epsilon)$, where z_{MSB} consists of the $(k + n)$ Most-Significant Bits (MSB) and z_{LSB} of



(a) Conventional Adder/ Subtractor [26].

(b) "Vouzis' Cotransformation" [18].

(c) "Novel Cotransformation Combination" (Section 4).

Figure 1 Comparison of three different architectures for LNS Addition/Subtraction.

the $(f-n)$ Least Significant Bits (LSB) of the two's-complement fixed-point z. The slope, $s_b'(z_{\text{MSB}}+\epsilon)$, can either be stored in a separate ROM, or computed from function values [28]. "Unpartitioned" means z_{MSB} is a multiple of a constant ($\Delta = 2^{-n}$) and $0 \le z_{\text{LSB}} < \Delta$. The variable ϵ can take any value in the interval $[0, \Delta]$, but the best accuracy occurs with $\epsilon \approx \Delta/2$. In particular for the "Novel Cotransformation Combination" the s_b function has to accommodate values of both negative and positive z, thus is most naturally defined as:

$$s_b(z) = \begin{cases} 0, & z \le e_{s_b} \\ s_b(z_{\text{MSB}}) + s_b'(z_{\text{MSB}}+\epsilon)\cdot z_{\text{LSB}}, & e_{s_b} < z < -e_{s_b} \\ z, & z \ge -e_{s_b}. \end{cases} \tag{20}$$

Because the use of $z > 0$ only occurs for the cotransformation special cases, an alternative to (20) is possible:

$$s_b(z) = \begin{cases} 0, & z \le e_{s_b} \\ s_b(z_{\text{MSB}}) + s_b'(z_{\text{MSB}}+\epsilon)\cdot z_{\text{LSB}}, & e_{s_b} < z < 0 \\ \log_b(2), & z = 0 \\ z, & z > 0. \end{cases} \tag{21}$$

The approximation (21) of s_b for $z > 0$ makes its implementation inexpensive, and although it would be inaccurate in the normal (20) context, the use of cotransformation insures the inaccurate $z > 0$ case is used only in Case B below, which is proved to work correctly.

Second, let Ω be a highly negative finite value, like $-2f$, which is used as the result for $d_b(0)$, such that $\Omega < e_{s_b} + d_b(2^{-f})$.

Case A: $z_l = 0$, $z_h > 0$ (We examine the case $z_h > 0$, and not $z \ge 0$, because when $z_h = 0$ and $z_l = 0$ the $z = 0$ branch of the s_b interpolator (21) is followed; thus it is not necessary to cover the $z_h = 0$ case here.)

$$\begin{aligned} t &= \min(x, y) + z_l + d_b(z_h) + s_b((d_b(z_l) - z_l) - d_b(z_h)) \\ &= \min(x, y) + z_l + d_b(z_h) + s((\Omega - z_l) - d_b(z_h)) \\ &= \min(x, y) + d_b(z_h) + s(\Omega - d_b(z_h)). \end{aligned} \tag{22}$$

Since $z_h > 0$ and z_h consists of the $(k+f-j)$ MSB of z, it follows that the quantized $2^{-f} < z_h$. From this we derive that $d_b(2^{-f}) \le d_b(z_h)$ since d_b is monotone increasing for positive arguments. Adding e_{s_b} to both sides and rearranging, we have $-d(z_h) \le -d(2^{-f}) \Rightarrow e_{s_b} + d_b(2^{-f}) - d_b(z_h) \le e_{s_b}$. Combining this with the

definition of $\Omega < e_{s_b} + d(2^{-f})$ we prove $\Omega - d_b(z_h) < e_{s_b}$. This means $s_b(\Omega - d_b(z_h)) = 0$, since the argument of s_b is smaller than the essential zero, e_{s_b}, thus $t = \min(x, y) + d(z_h)$, which is the correct result in this case.

Case B: $z_l \geq 0$, $z_h = 0$

$$t = \min(x, y) + z_l + d_b(z_h) + s_b((d_b(z_l) - z_l) - d_b(z_h))$$
$$= \min(x, y) + z_l + \Omega + s_b((d_b(z_l) - z_l) - \Omega). \quad (23)$$

By similar reasoning to Case A, $(d_b(z_l) - z_l) - \Omega > -e_{s_b}$. Since the interpolator is designed to treat all positive arguments as identities ($s_b(z) = z$ for $z > 0$), we have $s_b(d(z_l) - z_l) - d(z_h)) = d(z_l) - z_l - d(z_h)$ which means

$$t = \min(x, y) + z_l + d_b(z_h) + d_b(z_l) - z_l - d_b(z_h)$$
$$= \min(x, y) + d_b(z_l), \quad (24)$$

which is the correct result in this case.

4.3 Memory Requirements

The circuit that implements both addition and subtraction for the "Novel Cotransformation Combination" uses the interpolator (21) for both operations. LNS subtraction requires the extra tables of $F_3(z_l) = d_b(1 - b^{-z_l})$ and $F_4(z_h) = d_b(1 - b^{z_h})$, whose size depends on the choice of the lengths of z_l and z_h.

The function $F_4(z_h)$ approaches z_h from below as $z_h \to +\infty$, i.e., after some value $z_h = e_{F_4}$, the function becomes a tautology ($F_4(z_h) = z_h$) and its tabulation is not necessary from that point and forward since it is always equal to z_h. This value of e_{F_4} can be found by solving the equation

$$z_h - F_4(z_h) = 2^{-f} \Rightarrow z_h = 2^{-f} - \log_b(b^{2^{-f}} - 1) = e_{F_4}(b, f). \quad (25)$$

Since two consecutive values of z_h differ by $\delta_h = 2^{j-f}$ the number of words required for the tabulation of $F_4(z_h)$ are:

$$\xi_{F_4}(b, f, j) = \frac{e_{F_4}(b, f)}{\delta_h}. \quad (26)$$

Similarly, $F_3(z_l)$ approaches zero from below and the corresponding e_{F_3} value is equal to $e_{F_3}(b, f) = -\log_b(1 - b^{-2^{-f}})$. However, z_l consists of the j least significant bits of z, thus $0 \leq z_l < \delta_h$ and z_l spans all the values in its domain since $e_{F_3}(b, f) > \delta_h$, $\forall\, b, f, j$. In this case the required number of words is

$$\xi_{F_3}(j) = 2^j. \quad (27)$$

Table 1 Memory-size parametric study with regard to j for $b = 2$ and $f = 8$ for "Novel Cotransformation Combination."

j	3	4	5	6	7
ξ_{NovelCtr}	281	153	101	99	146
CLBs	271	216	204	195	210
Delay (ns)	37.171	35.038	33.534	32.204	32.560

Consequently, the total number of words of the "Novel Cotransformation Combination" subtraction can be evaluated by

$$\xi_{\text{NovelCtr}}(b, f, j) = \xi_{F_3}(j) + \xi_{F_4}(b, f, j). \quad (28)$$

The choice of j that leads to the circuit with the smallest memory requirements can be found by carrying out a parametric search on (28) with regard to j. Table 1 presents an example of this parametric study where we can see for $b = 2$ and $f = 8$ the value $j = 6$ gives the memory configuration with the smallest number of words. The same table presents synthesis results by Xilinx ISE on a Virtex-IV FPGA, which confirm that the value j calculated analytically gives the circuit with the smallest overall area and smallest delay. It can be observed that the bigger the deviation from the optimal value of j, the more memory words are required. For values of j smaller than the optimal value the dominant factor of Eq. 28 is $\xi_{F_4}(b, f, j)$ since its denominator becomes smaller, and the factor $\xi_{F_3}(j)$ becomes less important, since its exponent becomes smaller. On the contrary, for values of j bigger than the optimal value, $\xi_{F_3}(j)$ becomes the dominant factor, and $\xi_{F_4}(b, f, j)$ becomes less important.

4.4 Synthesis

The "Novel Cotransformation Combination" described in this paper eliminates the special cases required by [18, 21, 24] resulting in a reduction of the total area and the latency of an addition-subtraction ALU. This is illustrated in Fig. 2 which presents synthesis results for "Novel Cotransformation Combination" and "Vouzis' Cotransformation" both produced by Leonardo Spectrum with target technology of Taiwan-Manufacturing-Semiconductor-Company library for Application-Specific Integrated Circuits (ASIC) at 0.25 μm.

In Fig. 2, the "Novel Cotransformation Combination" derived above implements the s_b function via the interpolator (21), addition via (1) and subtraction via (5); the "Vouzis' Cotransformation" [18] implements the s_b function via a conventional interpolator [30], addition via (1) and subtraction via (4). In [18] the "Vouzis' Cotransformation" was shown to be better

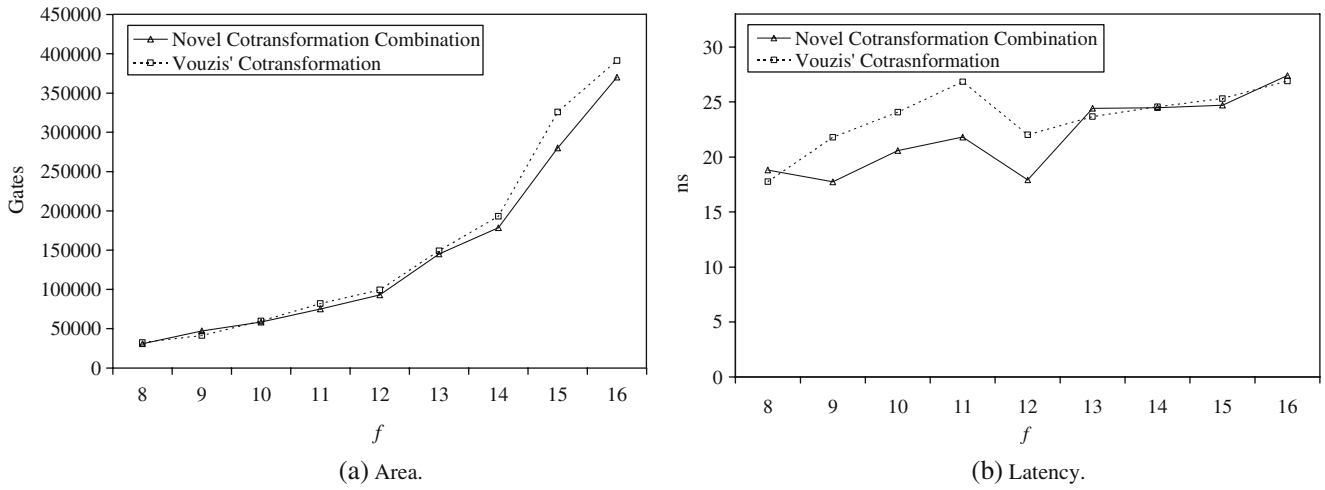


Figure 2 ASIC area and latency of "Novel Cotransformation Combination" of this paper and "Vouzis' Cotransformation" [18].

than other LNS subtraction alternatives, which in turn would be better than FP for suitable (multiply-, divide- and/or power-rich) applications. Here, Fig. 2a shows modest additional improvement with "Novel Cotransformation Combination." When $f \leq 13$, the two cotransformations occupy almost the same area; when f increases, the "Novel Cotransformation Combination" is more economical. In terms of latency, we can see in Fig. 2b that the "Novel Cotransformation Combination" has an advantage for $f \leq 12$. In other words, the novel cotransformation offers modest improvement in either area or speed. The comparison between the two cotransformations shows similar improvements in area and latency when both circuits are synthesized by using the Xilinx ISE for a Virtex-IV FPGA, as illustrated in Fig. 3. In both the ASIC and FPGA synthesis results the exponential increase in the area is because the tables that store the various functions required by LNS addition/subtraction are implemented by LUTs and gates, and no BlockRAMs or ROMs modules are used.

4.5 Simulation

Standard FP arithmetic provides round-to-nearest mode, which is impractical for LNS subtraction because of the table-maker's dilemma. Several less stringent LNS modes (in order of most accurate and thus most costly) have been proposed: restricted faithful [22], unrestricted faithful [28, 29] and subfaithful [31]. Faithful modes round each result either to the nearest or next-nearest quantized point. Restricted faithful is, on average, better than floating point, but occasionally rounds

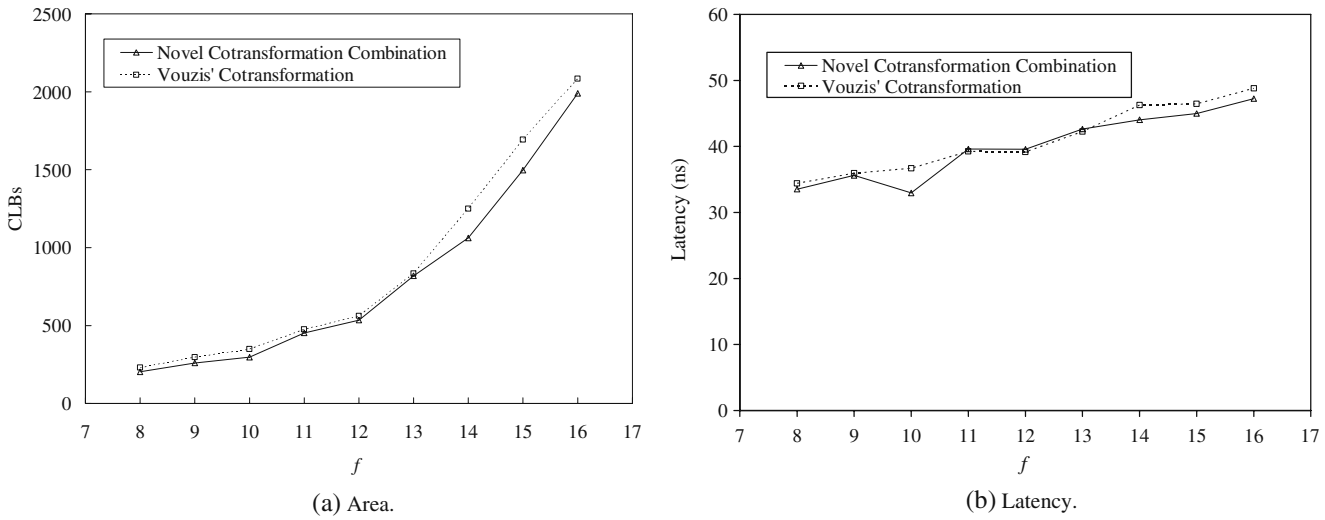


Figure 3 FPGA area and latency of "Novel Cotransformation Combination" of this paper and "Vouzis' Cotransformation" [18].

Table 2 Minimal cotransformation guard bits and next-nearest probabilities.

f	n	g	h	Probability
6	1	3	2	0.15
8	2	3	2	0.16
10	3	4	3	0.12
12	4	5	4	0.09
14	5	6	5	0.07
16	6	7	6	0.05
18	7	7	6	0.05

to next nearest. Because d_b is hard to approximate, Lewis [22] and later Detrey and de Dinechin [19] followed a weak-error model near the singularity.

The novel cotransformation described above can guarantee faithful results provided enough guard bits are used in the s_b, F_3 and F_4 LUTs. Table 2 shows probabilities of next-nearest rounding with the minimum output guard bits required (g for addition and h for subtraction) to provide faithful d_b results for $6 \le f \le 18$ using an unpartitioned s_b interpolator with the minimum $n = \lceil f/2 \rceil - 2$ possible for faithful s_b. A stand-alone implementation of the s_b function needs only $g = 2$ guard bits for faithful rounding. However, when the s_b is used for cotransformation the minimum number of guard bits increases as is shown in Table 2. The guard bits for the d_b tables are empirically similar ($h = g - 1$).

It may be more economical to omit such a large number of guard bits. Table 3 shows the worst-case errors and number of roundings to the nearest, next-nearest (1^{st}), second-next-nearest (2^{nd}) and other cases observed for all possible 2^{n+f} cases when $f = 12$ and $n = 4$, as g and h are lowered. The first line ($g = 15$, $h = 4$) is the only one that shows faithful results. When the h guard bits are eliminated (as in the next section), the worst error about doubles, which is still superior to the weak-error model used in [12, 19].

4.6 Optimized Interpolation/Cotransformation Hybrid

Coleman's method requires, and the other cotransformations allow, $d_b(z)$ for z not near zero to be implemented by interpolation. The interpolation multiplier, etc. would be shared by s_b and d_b. The only extra cost is the additional words for the portion of d_b approximated by interpolation. Although it has been suggested to combine these two techniques before, an analysis of the optimum combination has not been published previously.

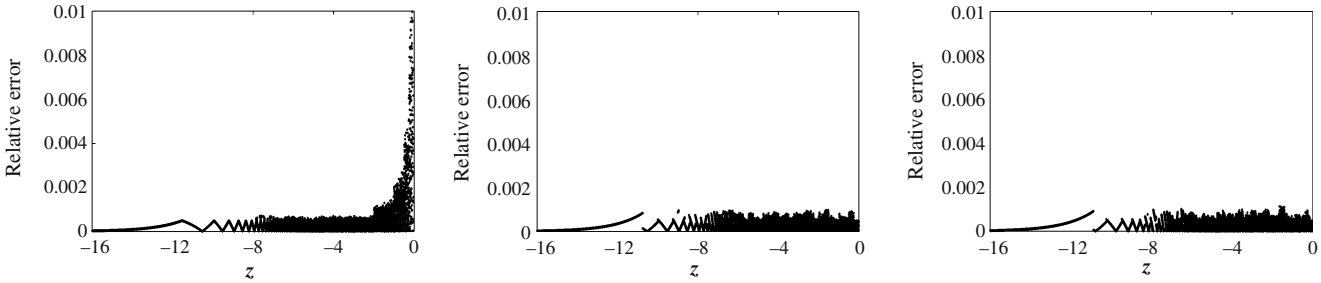
Assuming power-of-two partitioning for interpolation, and an m-bit address-bus size for the two cotransformation ROMs, the total number of words for d_b is $(k + f - 2m)2^n + 2 \cdot 2^m$. Taking the derivative with respect to m and solving to find m gives the minimum number of words: $2 \cdot 2^n = 2 \ln 2 \cdot 2^m$ or $m = n - \log_2(\ln 2) \approx n$. Thus the total cost of the optimal interpolation/cotransformation combination is about $(k + f - 2n)2^n + 2 \cdot 2^n$. If interpolation alone were used, the total cost would be $(k + f - n)2^n$, and the ratio would be $(k + f - 2n + 2)/(k + f - n)$. For linear interpolation with single precision ($f = 23$, $k = 5$ and $n = 10$), the size for the cotransformation/interpolation combination is $(28 - 20 + 2)/(28 - 10) = 5/9$ that of pure interpolation.

4.7 Optimized Multipartite/Cotranstormation Hybrid

The situation is different with multipartite. The ROMs used for addition and subtraction are entirely different from each other. There is almost no shared hardware between s_b and d_b. Synthesis results in [18] suggest it

Table 3 Effect of (g) interpolator- and (h) cotransformation-guard bits on error and rounding with $f = 12$ and $n = 4$. The "nearest," "1^{st}," "2^{nd}," and "other" represent the cases that the roundings are done to the nearest, next-nearest, second-next-nearest, and to all other cases, respectively.

g	h	Pos. error	Neg. error	Nearest	1^{st}	2^{nd}	Other
5	4	1.93e-4	−2.40e-4	44855	4297	0	0
5	3	1.93e-4	−2.52e-4	43525	5625	2	0
5	2	1.72e-4	−2.69e-4	41109	8006	37	0
5	1	1.72e-4	−3.46e-4	36257	12484	411	0
5	0	1.31e-4	−4.49e-4	20555	20668	7705	224
4	3	1.93e-4	−2.52e-4	43424	5724	4	0
4	2	1.72e-4	−2.69e-4	40984	8120	48	0
4	1	1.72e-4	−3.32e-4	36111	12595	446	0
4	0	1.31e-4	−4.80e-4	20418	20704	7789	241
3	2	1.72e-4	−2.81e-4	40957	8158	37	0
3	1	1.72e-4	−3.32e-4	36034	12665	453	0
3	0	1.31e-4	−4.80e-4	20375	20727	7807	243
2	1	1.31e-4	−3.35e-4	37638	11104	410	0
2	0	1.31e-4	−4.80e-4	20329	21355	7239	229



(a) d_b implemented by the multipartite method [19] depicted in Fig. 1(a).

(b) d_b implemented by "Vouzis' Cotransformation" [18] and s_b by the multipartite method [19] depicted in Fig. 1(b).

(c) d_b implemented by "Novel Cotransformation Combination" and s_b by 1st-order interpolation depicted in Fig. 1(c).

Figure 4 Error performance of the d_b function for different implementation choices for $f = 10$.

is better to use cotransformation exclusively for d_b and multipartite exclusively for s_b, which is possible only for "Arnold's Cotransformation" and the novel variations of it discussed in this paper.

5 Error Analysis

The singularity of d_b near zero requires excessive memory in order to attain a particular accuracy. Some implementations relax accuracy in this region to achieve reasonable memory size; as in [19], the multipartite tables give lower $d_b(z)$ accuracy for z close to zero. Fig. 4a shows the multipartite-d_b error of [19] for $f = 10$. The errors grow almost to 0.01, which is about ten times larger than desired.

The substitution of the d_b function with cotransformation essentially moves the calculation of the subtraction algorithm away from the singularity around zero. This is illustrated in Fig. 4b, which shows that the "Vouzis' Cotransformation," described in [18], ($f = 10$ using the multipartite method for s_b and special cases for d_b with no guard bits) has a more uniform error distribution closer to the target accuracy of 2^{-10} than the multipartite method alone. Finally, Fig. 4c presents the error performance of the "Novel Cotransformation Combination" derived here ($f = 10$ using 1st-order interpolation for s_b and "Novel Cotransformation Combination" for d_b). Although this last implementation is the cheapest in terms of area, its error performance is similar to Fig. 4b, which was previously shown [18] to be better than other LNS subtraction options, which in turn have been shown previously can be better than FP for certain applications. In other words, the cotransformations presented here produce much more accurate results than [19], even though, for economy, cotransformation was implemented without guard bits.

6 Conclusions

The different cotransformation methods offer different advantages in terms of area, latency and accuracy which derive from their mathematical foundation. The new cotransformation presented here merges the use of $z < 0$ for LNS addition proposed initially by "Coleman's Cotransformation" [16], with the complete elimination of the $d_b(z)$ function proposed initially by "Arnold's Cotransformation" [21], with the absence of any special cases as required by "Vouzis' Cotransformation" [18]. This combination results in a circuit for LNS addition/subraction more economical at higher precisions, and more accurate than multipartite or interpolation methods for subtraction of nearly equal values (z close to zero), as has been illustrated by exhaustive simulation. If faithful rounding is desired the number of guard bits can be determined by using simulations. Compared to previous cotransformations, the "Novel Cotransformation Combination" offers a reasonable tradeoff of area, speed and accuracy and can make LNS more beneficial as an implementation alternative for application-specific applications (multimedia [7, 32], embedded control [6], scientific simulation [8]) having a majority of easy (multiply, divide or power) operations but a minority of subtractions that need to be performed with faithful or near-faithful accuracy for successful application performance.

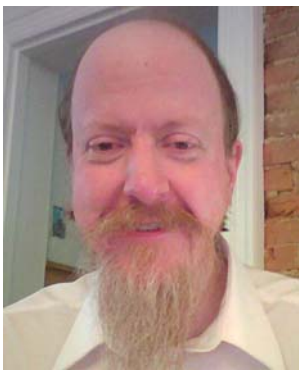
Acknowledgements The authors would like to thank Nicolas Frantzen and Jesus Garcia for their contributions in the early stages of this work.

References

1. ANSI/IEEE (1985). *Standard 754-1985 for Binary Floating-Point Arithmetic*.
2. Swartzlander, E. E., & Alexopoulos, A. G. (1975). The sign/logarithm number system. *IEEE Transactions on Computers, 24*, 1238–1242, Dec.
3. Swartzlander, E. E., Chandra, D., Nagle, T., & Starks, S. A. (1983). Sign/logarithm arithmetic for FFT implementation. *IEEE Transactions on Computers, C-32*, 526–534.
4. Chester, E. I., & Coleman, J. N. (2002). Matrix engine for signal processing applications using the logarithmic number system. In *Proceedings of the 13th IEEE international conference on application-specific systems, architectures and processors* (pp. 315–324). San Jose, CA, 17–19 July.
5. Jullien, G. A. (2006). Array processing using alternate arithmetic - A 20 year legacy. In *Proceedings of the IEEE international conference on application-specific systems, architectures, and processors* (pp. 199–204). Steamboat Springs, CO. 11–13 Sept.
6. Bleris, L. G., Garcia, J. G., Kothare, M. V., & Arnold, M. G. (2006). Towards embedded model predictive control for system-on-a-chip applications. *Journal of Process Control, 16*, 255–264, March.
7. Wrigley, A. (1993). *Real-time ray tracing on a novel HDTV framestore*. PhD thesis, University of Cambridge, England.
8. Makino, J., & Taiji, M. (1998). *Scientific simulations with special-purpose computers: The GRAPE systems*. John Wiley & Son Ltd., Feb.
9. Young, S., Kershaw, D., Odell, J., Ollason, D., Valtchev, V., & Woodland, P. (2006). *The HTK book (Version 3.4)*. Cambridge University Engineering Department, Dec.
10. Hughey, R., & Blas, A. D. (2006). The UCSC Kestrel application-unspecific processor. In *Proceedings of the IEEE international conference on application-specific systems, architectures, and processors* (pp. 163–168). Steamboat Springs, CO, 11–13 Sept.
11. Haselman, M., Beauchamp, M., Wood, A., Hauck, S., Underwood, K., & Hemmert, K. S. (2005). A comparison of floating point and logarithmic number systems for FPGAs. In *Proceedings of the 13th annual IEEE symposium on field-programmable custom computing machines (FCCM'05)* (pp. 181–190). Washington, DC: IEEE Computer Society, 17–20 April.
12. Collange, C., de Dinechin, F., & Detrey, J. (2006). Floating point or LNS: Choosing the right arithmetic on an application basis. In *Proceedings of the 9ih EuroMicro digital system design (DSD 2006)* (pp. 197–203). Dubrovnik: Croatia, 30 Aug.–1 Sept.
13. Detrey, J., & de Dinechin, F. (2007). A tool for unbiased comparison between logarithmic and floating-point arithmetic. *The Journal of VLSI Signal Processing, 49*, 161–167, October.
14. Fu, H., Mencer, O., & Luk, W. (2007). Optimizing logarithmic arithmetic on FPGAs. In *Proceedings of the IEEE symposium on field-programmable custom computing machines* (pp. 163–172). Palo Alto, CA, 14–15 April.
15. Coleman, J. N., Softley, C. I., Kadlec, J., Matousek, R., Tichy, M., Pohl, Z., et al. (2008). The European logarithmic microprocessor. *IEEE Transactions on Computers, 57*, 532–546, April.
16. Coleman, J. N. (1995). Simplification of table structure in logarithmic arithmetic. *IEE Electronic Letters, 31*, 1905–1906, 26 Oct.
17. Vouzis, P., Collange, C., & Arnold, M. (2007). LNS subtraction using novel cotransformation and/or interpolation. In *Proceedings of the 18th IEEE international conference on application-specific systems, architectures, and processors* (pp. 107–114). Montreal, Canada, 9–11 July.
18. Vouzis, P., Collange, C., & Arnold, M. (2007). Cotransformation provides area and accuracy improvement in an HDL Library for LNS subtraction. In *Proceedings of the 10th EuroMicro conference on digital systems and design* (pp. 85–93). Lübeck, Germany, 27–31 August.
19. Detrey, J., & de Dinechin, F. (2003). A VHDL library of LNS operations. In *Proceedings of the 37th asilomar conference on signals, systems, and computers* (Vol. 2, pp. 2227–2231). Pacific Grove, CA, 9–12 Nov.
20. Arnold, M., & Winkel, M. (2001). Reconfiguring an FPGA-based RISC for LNS arithmetic. In *Reconfigureable technology: FPGAs and reconfigurable processors for computing and communications III, proceedings of SPIE* (Vol. 4525, pp. 88–98). Denver, CO, 21–22 Aug.
21. Arnold, M. G., Bailey, T. A., Cowles, J. R., & Winkel, M. D. (1998). Arithmetic co-transformations in the real and complex logarithmic number systems. *IEEE Transactions on Computers, 47*, 777–786, July.
22. Lewis, D. M. (1995). 114 MFLOPS logarithmic number system arithmetic unit for DSP applications. *IEEE Journal of Solid-State Circuits, 30*, 1547–1553, Dec.
23. de Dinechin, F., & Tisserand, A. (2005). Multipartite table methods. *IEEE Transactions on Computers, 54*, 319–330, March.
24. Arnold, M. G. (2002). An improved cotransformation for logarithmic subtraction. In *Proceedings of the international symposium on circuits and systems (ISCAS'02)* (pp. 752–755). Scottsdale, Arizona, 26–29 May.
25. Hassler, H., & Takagi, N. (1995). Function evaluation by table look-up and addition. In *Proceedings of the 12th symposium on computer arithmetics* (pp. 10–16). Bath, England, 19–21 July.
26. Detrey, J., & de Dinechin, F. (2005). Table-based polynomials for fast hardware function evaluation. In *Proceedings of the 16th international conference on application-specific systems, architectures and processors (ASAP'05)* (pp. 328–333). Samos, Greece, 23–25 July.
27. Lee, D., Luk, W., Villasenor, J., & Cheung, P. (2003). Non-uniform segmentation for hardware function evaluation. In *Proceedings of the international conference on field programmable logic and its applications* LNCS 2778 (pp. 796–807). Lisbon, Portugal 1–3 Sept.
28. Lewis, D. M. (1990). An architecture for addition and subtraction of long word length numbers in the logarithmic number system. *IEEE Transactions on Computers, 39*, 1325–1336, Nov.
29. Arnold, M. G. (2001). Design of a faithful LNS interpolator. In *Proceedings of the 4th EuroMicro digital system design (DSD'01)* (pp. 336–345). Warsaw, Poland, 4–6 Sept.
30. Arnold, M. G. (2001). A pipelined LNS ALU. In *Proceedings of the IEEE workshop on VLSI* (pp. 155–161). Orlando, Florida, 19–20 April.
31. Arnold, M. G. (2004). LPVIP: A low-power ROM-less ALU for low-precision LNS. In *Proceedings of the 14th international workshop on power and timing modeling, optimization and simulation* LNCS 3254, (pp. 675–684). Santorini, Greece, 15–17 Sept.
32. Arnold, M. G. (2004). Redundant logarithmic arithmetic for MPEG decoding. In *Proceedings of the international symposium on optical science SPIE annual meeting 2004* (Vol. 5559, pp. 112–122). Denver, CO, 2–6 Aug.

Panagiotis D. Vouzis is a current Postdoctoral Fellow at Carnegie Mellon University in the Process Systems Engineering Group. He earned his Ph.D. in Computer Engineering from Lehigh University in 2008. He received his M.S. in Integrated Hardware and Software Systems and diploma in Electrical and Computer Engineering from the University of Patras, Greece, in 2004 and 2002, respectively. Vouzis was awarded a scholarship for completing his M.S. with highest honors by the Greek State Scholarship Foundation. In addition, he received the best-paper award at the Application-specific Systems, Architectures and Processors Conference in 2007. His current research interests include FPGA design, embedded control systems, computer arithmetic, and general-purpose computing on Graphics Processing Units.

Caroline Collange received the Master degree in computer science from the École normale supérieure de Lyon, France in 2007. She is currently a PhD candidate at the University of Perpignan, France. In 2006, she worked as a research intern at Lehigh University, Bethlehem, Pennsylvania. Her research interests include computer arithmetic, FPGA circuit design and general-purpose computations on graphics processing units.



Mark G. Arnold received the BS and MS degrees from the University of Wyoming (USA), and the PhD degree from the University of Manchester Institute of Science and Technology (UK). From 1982 to 2000, he was on the faculty of the University of Wyoming. From 2000 to 2002, he was a lecturer at UMIST. In 2002, he joined the faculty of Lehigh University (USA), where he is an assistant professor. His research interests include computer arithmetic, embedded control systems, hardware description languages, and microrobotics.