LNS with Co-Transformation Competes with Floating-Point

J. N. Coleman and R. Che Ismail, *Senior Member, IEEE*

Abstract—The logarithmic number system has been proposed as an alternative to floating-point arithmetic. Multiplication, division and square-root operations are accomplished with inexpensive fixed-point methods, but addition and subtraction are considerably more challenging. Recent work has demonstrated that these operations too can be done with similar speed and accuracy to their FP equivalents, but the necessary circuitry is complex. In particular, it is dominated by the need for large ROM tables for the storage and interpolation of non-linear functions. We describe a new co-transformation procedure that eliminates much of the ROM space and allows the easy synthesis of the remainder in logic, and we then evaluate several interpolation methods that might benefit from it. Synthesised 32-bit implementations are compared with floating-point units, and show substantial reductions in delay, with equivalent accuracy and area.

Index Terms—High-speed arithmetic, logarithmic number system

1 Introduction

There are five parameters of interest in the design of a real arithmetic unit: wordlength (which governs range and precision), accuracy (i.e., error, both in quantisation and processing), speed, area, and power dissipation. The floating-point (FP) system is commonly implemented at wordlengths of 32, 64 and 80-bits, and has a maximum error of 0.5 l.s.b.. The other three parameters are open-ended and therefore subject to continuous attempts at improvement.

In 2008 we described a 32-bit microprocessor in which the real arithmetic subsystems had been replaced by ones based on the Logarithmic Number System (LNS) [1]. We showed that its speed and accuracy exceeded that of a commercial FP device fabricated in a similar technology, and concluded that further improvements in LNS techniques would open a clear gap between these and FP implementations. Herein, we describe such a development.

In an LNS a real number is represented as its fixed-point logarithm. Range and precision of the represented numbers are very similar to floating-point values of the same wordlength. Multiplication, division and square root operations become fixed-point addition, subtraction and right shift respectively. Unlike their FP counterparts, these operations are fast and cheap (often effectively cost-free, as they share the existing fixed-point unit), and multiplication and division return exactly quantised results. Until recently, however, these inherent advantages were offset by the difficulty of implementing addition and subtraction, which involve the evaluation of the non-linear functions (1) and (2). For $i = \log_2 x, j = \log_2 y, r = j - i$, and assuming $j \leq i$:

$$\log_2 (2^i + 2^j) = i + \log_2(1 + 2^r) = i + F_{\mathrm{A}}(r), \tag{1}$$

$$\log_2 (2^i - 2^j) = i + \log_2(1 - 2^r) = i + F_{\mathrm{S}}(r). \tag{2}$$

The functions $F_{\mathrm{A}}(r)$ and $F_{\mathrm{S}}(r)$, generically referred to as $F(r)$, are illustrated in Fig. 1. Up to about 20 bits these function values can be stored directly in a ROM. Being irrational, they are subject to a rounding error. Beyond this, memory requirements become prohibitive, and instead the function is stored at intervals with intervening values obtained by interpolation, which adds to the error. Following an operation the error in a result can be exponentiated into the linear domain and compared directly with the corresponding FP error, as described in Section 2 below. A desirable objective has always been to keep it within the worst-case FP error of 0.5 l.s.b., but this has not always been achieved. The problem is compounded by the singularity in the subtraction function, where the rapidly changing derivative as $r \to 0$ necessitates the use of successively smaller interpolation intervals that require a significant increase in storage, often to the point of impracticality. To some extent the issues of function evaluation in general, and dealing with the singularity in particular, have been addressed separately.

Interpolation increases the delay and also introduces its own error. A variety of interpolation techniques have been devised, some of which maintain better accuracy than FP arithmetic. Of these, some operate over the entire range of the subtraction curve and consequently have a large storage requirement. Others do not attempt to operate near the singularity, instead deploying some algebraic technique that transforms subtractions in this region into an equivalent calculation comprising only of easier operations. This itself introduces further delays, arithmetic elements and storage, and while the impact of these might be significantly less than that of an interpolation in this

- *J.N. Coleman is with the School of Electrical and Electronic Engineering, University of Newcastle upon Tyne, NE1 7RU, United Kingdom. E-mail: j.n.coleman@ncl.ac.uk.*
- *R. Che Ismail is with the School of Microelectronic Engineering, Universiti Malaysia Perlis, 02600 Arau, Malaysia. E-mail: rizalafande@unimap.edu.my.*

Manuscript received 28 Apr. 2013; revised 14 Nov. 2014; accepted 23 Dec. 2014. Date of publication 15 Mar. 2015; date of current version 16 Dec. 2015. Recommended for acceptance by F. de Dinechin.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2015.2409059

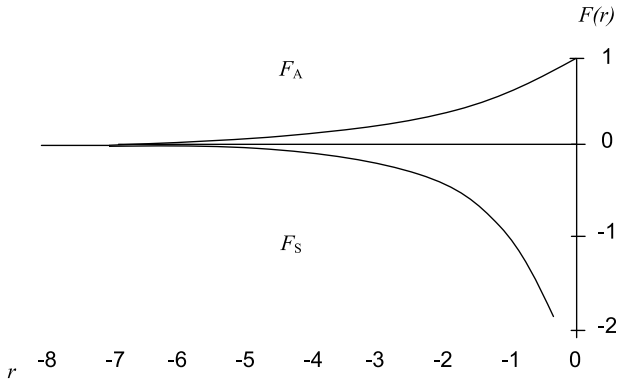

Fig. 1. LNS addition and subtraction functions.

region, they still impose a burden that is vastly disproportionate to the small part of the range that these subtractions represent.

Section 2 will illustrate the problem and set the baseline for this work by reviewing a number of existing 32-bit interpolators, all of which achieve FP-equivalent accuracy. They vary in their treatment of r close to zero, but all manifest the basic difficulty in dealing with subtractions in this region.

The method we describe in this paper is sometimes referred to as co-transformation. It takes i and r, and converts them to new values where r is of larger magnitude and hence easier to interpolate. We will take an existing first-order co-transform and develop it into a second-order arrangement (cf. [10] eqn (30)). By doing so, we will significantly reduce its storage to an extent that allows the remaining tables to be conveniently synthesised in logic. By removing explicit ROM elements from the design, together with their associated timing and layout considerations, a significant increase in speed is also achieved. The trade-off is that, whenever the new transform is used, the subsequent interpolation needs to be performed twice.

Sections 3 and 4 deal, respectively, with the algebra of the original and modified co-transforms, while Section 5 deals with the specific question of what proportion of subtractions will require its use. Section 6 revisits the interpolator schemes introduced in Section 2, and shows how each would be affected if used with the new co-transform. Two interpolators are then chosen for further development, one emphasising high speed and the other a small area.

Section 7 will then develop these latter two schemes into full arithmetic units, which will be compared with a variety of FP units independently designed by other workers using a similar fabrication technology. Previous studies, e.g. [19], [20], have concluded that the choice between FP and LNS is best made individually for each application. We will show that, for general use, 32-bit LNS arithmetic offers significantly less delay than FP, roughly equal area, and equivalent accuracy.

2 Existing Methods

In this section we will describe a number of existing LNS implementations that achieve accuracy equivalent to that of floating-point arithmetic. The relevant definitions of 'accuracy' were presented in [4] and, for ease of reference, we open with a short summary of this material.

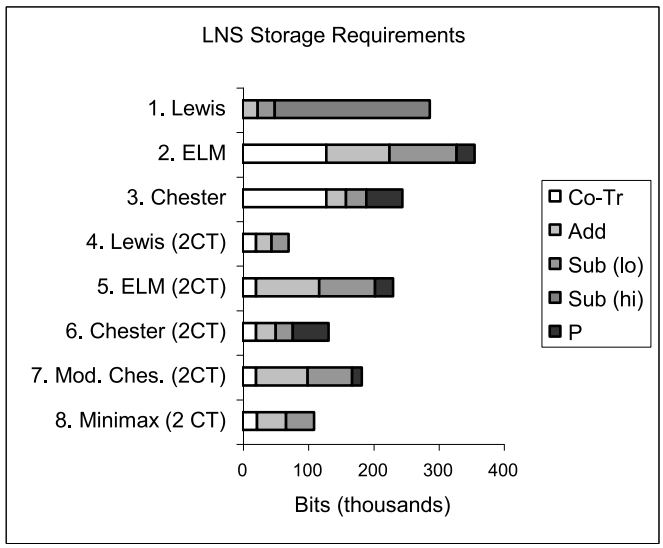

Fig. 2. Storage requirements for various LNS interpolation schemes.

2.1 Measurement of Accuracy

In an FP system, assuming that both operands represent exact values, a result $\hat{A}$ as returned by a practical implementation can be regarded as an approximation to the corresponding exact result A. Each result is thus in error by $e = \hat{A} - A$. This error may be quoted relative to the exact result itself and, with an f-bit mantissa, may be expressed in terms of the weight of the l.s.b.. In [4] we denoted this quantity as the 'relative arithmetic error',

$$e_{\text{rel arith}} = \frac{1}{2^{-f}} \frac{\hat{A} - A}{A}. \tag{3}$$

In the equivalent LNS (i.e., with f-bit fractional part), it may be correspondingly assumed that the inputs to an operation are exact. Direct comparison can be made between the two number systems by exponentiating the LNS operands and result to their FP representation.

For $f = 23$, FP arithmetic has worst-case errors $e_{\text{max rel arith}} \approx 0.5$, $e_{\text{min rel arith}} \approx -0.5$, and mean errors $e_{\text{av rel arith}} = 0$, $|e|_{\text{av rel arith}} = 0.1733$.

2.2 LNS Implementation Methods

Lewis A high-order polynomial interpolator was presented by Lewis [2], and was the first published 32-bit design that maintained FP-equivalent accuracy. Its coefficients were calculated dynamically which imposed a delay but, apart from the storage for the subtraction function in the range $r \geq -1$, its memory was extremely compact. The interpolator tables were optimally partitioned, but implemented with an address generator that mapped each partition into a single physical device. The lookup table sizes, shown in Fig. 2, row 1, illustrate the problem associated with subtractions for which r is close to zero. The segment to the left represents the storage for the addition operation. The middle segment ('Sub (lo)') shows the storage for subtractions in the range $r < -1$, and the rightmost segment ('Sub (hi)') that for $r \geq -1$. The latter is 9.2 times that needed for the remainder of the subtraction curve, and accounts for 83 percent of the total storage in this design. A fabricated variant [3]

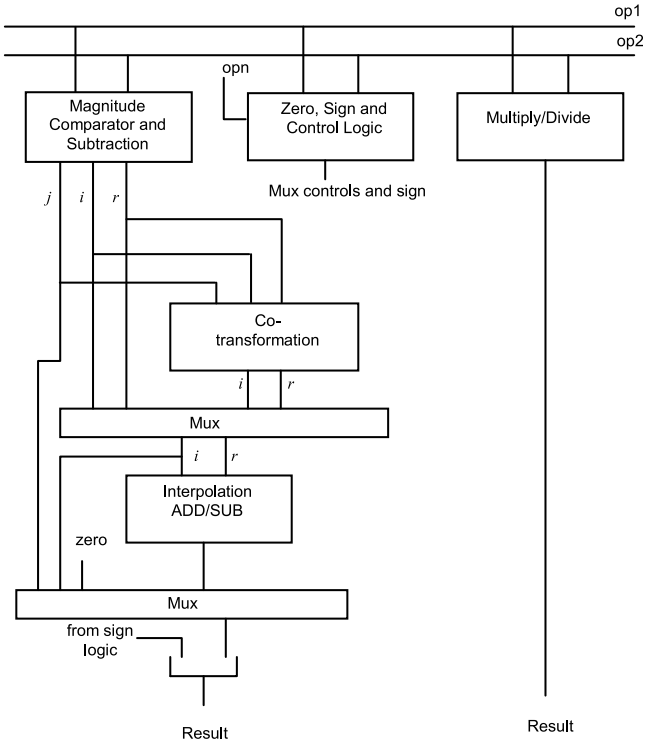

Fig. 3. Block diagram of ELM ALU (from [4]).

therefore relaxed the requirement for accurate interpolation in this region, as did other implementations e.g. [9] which is based on multipartite tables.

ELM We described an alternative 32-bit interpolation technique [4] that was based on a first-order Taylor approximation with a correction scheme that approximated the combined effect of the higher-order terms. The two curves were each partitioned at successive powers of 2. Each partition was divided into 256 intervals, n, of width Δ. The value of Δ doubled from one partition to the next, although for clarity in what follows we shall omit reference to this variation. For each interval was stored the value of the function $F(-n\Delta)$ at the start of the interval, together with the slope of the tangent, $D(-n\Delta)$, at that point. The error in an interpolation increased as the interpolation point δ moved along the tangent away from the stored point, up to a maximum $\mathrm{E}(-n\Delta)$ immediately before the next stored point. This maximum error was also stored for each interval. It was observed that the error at δ, expressed as a proportion of the maximum error E for that interval, was approximately constant in all intervals on both curves. By calculating this proportion throughout one interval and storing its values on another table, P, the error in any particular interpolation could be determined by multiplying the interval-specific value $\mathrm{E}(n)$ by $P(\delta)$. This was accumulated with $\delta D(-n\Delta)$ and $F(-n\Delta)$ to obtain the result. That is,

$$F(-n\Delta - \delta) \approx F(-n\Delta) - \delta D(-n\Delta) + E(n)P(\delta).$$

This maintained FP-equivalent accuracy and offered a much shorter delay path than hitherto because inputs to the multipliers were delivered directly from the tables. These were implemented as physically separate devices for each partition, (and it is interesting to note that such a multi-table structure has been identified as a useful factor in power reduction [26]). However, it did not attempt to operate near the singularity. Instead, a co-transform [7] was applied in the case of any subtraction with r close to zero (>-0.5), which it converted to an equivalent subtraction with r well away from zero. The co-transform is implemented with two tables F1 and F2, which together comprise a far smaller amount of storage than would the tables necessary for accurate interpolation. These two techniques were combined in the first 32-bit silicon with FP-equivalent accuracy, the European Logarithmic Microprocessor [1]. Fig. 3 shows an outline of the arrangement.

The storage sizes for this scheme are shown in row 2 of Fig. 2. The co-transform unit comprises F1 and F2 tables each of 2,048 words, 63,488 and 65,536 bits respectively, or 129,024 bits in total. It is now 1.3 times that of the remainder of the subtraction curve ('Sub (lo)'), and accounts for 36 percent of the total storage. A die plot of the ELM was shown in [21]. There are two adder/subtractor units, known as 'multi-cycle ALUs'. The logic for the two units is merged, but their tables are mirrored on either side of the die. The F1 and F2 co-transform tables (labeled G on the plot), in exact agreement with the proportion of bits that they contain, comprise 36 percent of the total ROM area.

Chester An improvement to this interpolation technique was suggested by Chester [8], who proposed using a chord as the interpolating line. This reduced the size of the F and E tables at the expense of a larger P table, a significant decrease in the net total. Values of D were calculated dynamically, by looking up and subtracting two consecutive values of F, which was therefore held in an interleaved memory. Thus

$$F(-n\Delta - \delta) \approx F(-n\Delta) - (\delta/\Delta)[F(-n\Delta) - F(-\{n+1\}\Delta)] + E(n)P(\delta).$$

Calculation of the indices to the F tables imposed an additional delay, but the subtraction to obtain D passed a carry-save result to the following multiplier and elimination of the D table resulted in a further substantial memory saving. Implementation of the co-transform was unchanged from [4]. The co-transform tables were now 4.1 times the size of the remaining subtraction table and accounted for 53 percent of total storage (Fig. 2, row 3).

FloPoCo An alternative co-transform was proposed in [10], [11], [12], [13], [14]. It converts r for a subtraction into an argument to the addition function which, having no singularity, is easier to interpolate. Hardware complexity is roughly equivalent to that of [7]. The most recent version of this scheme [14] is used in FloPoCo [15], a VHDL generator for FPGA arithmetic. Additions and direct subtractions are executed with an extension of multipartite table methods that allow the evaluation of polynomials of any degree [16], and have a maximum error of one FP-equivalent l.s.b.. The co-transformation is applied as r approaches zero (the actual threshold is configurable), but such subtractions may then have a larger error [17].

Minimax Fu [18] examined the Minimax algorithm as a solution to the interpolation of $F(r)$, and devised an FPGA-based arrangement. Equations (1) and (2) were

rearranged to bring r onto the positive axis, and an adaptive technique selected the most optimal intervals for the application of a Minimax algorithm: for $F_A(r)$ 416 intervals were required to cover the interpolated range. As in Lewis' design, an address generating circuit provided an address input to a single physical ROM, but to maintain time-efficiency the coefficients were then delivered directly to the multipliers. At 32 bits, a second-degree polynomial achieved FP-equivalent accuracy, and an economical datapath layout evaluated this by Horner's method:

$$\begin{aligned} F_A(r) &= c_2x^2 + c_1x + c_0 \\ &= c_0 + (c_1 + c_2x)x. \end{aligned}$$

A similar approach was used for $F_S(r)$, except in the region close to the singularity, in this case $r < 4$. Using a technique first proposed by Paliouras and Stouraitis [5], these subtractions decomposed $F_S(r)$ into two separate functions, both easier to interpolate than $F_S(r)$ itself. However, they required an additional interpolator and tables for their evaluation. The scheme is particularly suitable for use on an FPGA where multiplication hardware is abundant, but it is difficult to extrapolate an estimate of its size or performance in a silicon implementation.

It will be evident from Fig. 2 that the storage requirement for the subtraction curve as it approaches zero accounts for a vastly disproportionate amount of the storage overall. A figure of 83 percent when these values are interpolated (in [2]) improves to 36 or 53 percent when the co-transform is applied ([4] and [8]), but even this is far more than is commensurate with the proportion of operations lying in this region. In the following two sections we will review the co-transform algebra, and then describe a development of it that substantially reduces this storage. As a result, the entire ALU can now be synthesised in logic.

3 First-Order Co-Transformation

The co-transformation was originally conceived for use with an interpolator based on a first-order Taylor approximation. We will describe its development within the context of that interpolator, but it should be noted that the resulting algebra is independent of this and pertains only to the region $-1 < r < 0$. It can therefore be applied in conjunction with any interpolating scheme that adequately handles the remainder of the curve. From hereon we shall be concerned only with the subtraction function so, for clarity, we abbreviate all references to $F_S(r)$ to $F(r)$.

For most of the range of r, $F(r)$ can be obtained by interpolation, using tables of the function (F) and derivative (D). The tables are stored at intervals of Δ, which is a power of 2, and is the largest possible value that permits accurate interpolation. To accommodate the varying nonlinearity, the function is partitioned at varying powers of 2, with the value of Δ in each partition being half that of the partition to its left. As $r \to 0$ the subtraction function exhibits a singularity in which the rapidly changing derivative makes accurate interpolation difficult without continuing the partitioning through several negative powers of 2, so massively increasing the storage requirement.

The co-transformation simplifies the subtraction operation by obviating the interpolation in the region $-1 < r < 0$, eliminating table D and reducing substantially the size and complexity of table F, using instead two much smaller and regularly organised tables. At and below $r = -1$, the F and D tables are implemented as before, but in this region are small. It relies on the replacement of subtraction $2^i - 2^j$ with two successive subtractions

$$2^i - 2^j = (2^i - 2^{j+k[1]}) - 2^{j+k[2]}, \tag{4}$$

where

$$2^{k[1]} + 2^{k[2]} = 1, \text{i.e.,} k[2] = \log_2(1 - 2^{k[1]}). \tag{5}$$

Factor $2^{k[1]}$ is individually chosen for each value of $r = j - i$ such that the index $r[1]$ for the inner subtraction falls on the nearest modulo-$\Delta[1]$ boundary beneath $j - i$, where $\Delta[1]$ is now fixed at a large value. $F(r[1])$ can therefore be obtained directly from lookup table F1, which contains $F(r)$ for $-1 < r < -\Delta[1]$ at modulo-$\Delta[1]$ intervals. Factor $k[1]$ is constrained to lie in the range $-\Delta[1] \le k[1] < 0$, and can therefore be used to index another lookup table F2, containing $F(r)$ for all possible values of r between $-\Delta[1] < r < 0$, to obtain $k[2]$. Since $2^{k[1]} \approx 1$, $k[2]$ is a large negative value. This has the effect of increasing the magnitude of the index for the outer subtraction, $r[2]$, such that $r[2] < -1$. It therefore falls in the linear region of $F(r[2])$, and can be obtained by interpolation from the small remaining F and D tables covering this region.

Thus:

$$r[1] = (((j - i) \operatorname{DIV} \Delta[1]) - 1) \times \Delta[1] = j + k[1] - i, \tag{6}$$

$$k[1] = -(((j - i) \operatorname{MOD} \Delta[1]) + \Delta[1]) = i - j + r[1], \tag{7}$$

$$k[2] = F(k[1]). \tag{8}$$

The original values i and j are modified to yield the following operands to the outer subtraction:

$$i[2] = i + F(r[1]), \tag{9}$$

$$j[2] = j + F(k[1]) = j + k2, \tag{10}$$

and the subtraction becomes

$$2^i - 2^j = 2^{i+F(r[1])} - 2^{j+F(k[1])}. \tag{11}$$

This generates an index $r[2]$

$$\begin{aligned} r[2] &= j - i + F(k[1]) - F(r[1]) \\ &= j - i + \log_2((1 - 2^{i-j+r[1]})/(1 - 2^{r[1]})). \end{aligned} \tag{12}$$

The value of $r[2]$ can be considered in three regions, depending on the original operands i and j. For $j - i \le -1$, $r[2]$ is taken directly as $j - i$, and will lie in the linear region of F from which $F(r)$ can be obtained by interpolation. For $-1 < j - i < -\Delta[1]$, $r[2]$ is derived as shown above, and as it also lies in the linear region of F, $F(r)$ is similarly obtained by interpolation. For the third region, $-\Delta[1] \le j - i < 0$, the

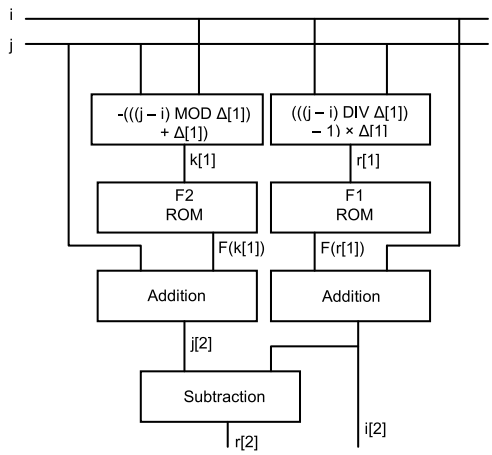

Fig. 4. Conceptual arrangement of first-order co-transformation.

derived value of $r[2]$ rises above -1. However, this range is covered by the F2 table, and $F(r)$ is therefore already available as $k[2]$. The modified values $r[2]$ and $i[2]$ are passed to the interpolator for completion of the outer subtraction.

Following interpolation of $F(r[2])$, the result of the subtraction is

$$\log_2(2^i - 2^j) = i + F(r[1]) + F(r[2]). \tag{13}$$

The arrangement is illustrated in Fig. 4.

It is shown in [7] that the combined size of the F1 and F2 tables is about one-seventh of that of the F and D tables that would be required to yield an interpolation of similar accuracy.

A number of simplifications are possible in an implementation of this scheme. In the calculation of $k[1]$ and $r[1]$ the subtraction $j - i$ is not necessary because this term is already available as r. Operation DIV returns a truncated result, and since $\Delta[1]$ is a power of 2 the DIV, MOD and $\times$ operations involve only bit-partitioning and concatenation of zeroes. Thus the only arithmetic operations required in these calculations are the addition or subtraction of the single-bit constants $\Delta[1]$ and 1. However, it will be noted that there is a deterministic relationship between the bit-partitioned values of r and the functions $k[1]$ and $r[1]$ that form the indices to the F1 and F2 tables. The addition and subtraction can therefore be avoided completely by rearranging the mapping of addresses to function values in these tables. Calculation of the index $r[1]$ and coefficient $k[1]$ can thus be done with no time overhead at all. Finally, the subtraction to obtain $r[2]$ can be rearranged to use the precalculated value of r, and to use cumulative additions instead of an addition and a subtraction. The entire unit can thus be implemented with a worst-case delay of one ROM access, a carry-propagate adder and a carry-save stage.

4 Second-Order Co-Transformation

The fractionating coefficient $k[1]$ can be applied recursively. Substituting

$$2^{j+k[2]} = 2^j - 2^{j+k[1]}$$

into (4), and designating a new value $k[11]$ for use in a manner analogous to $k[1]$:

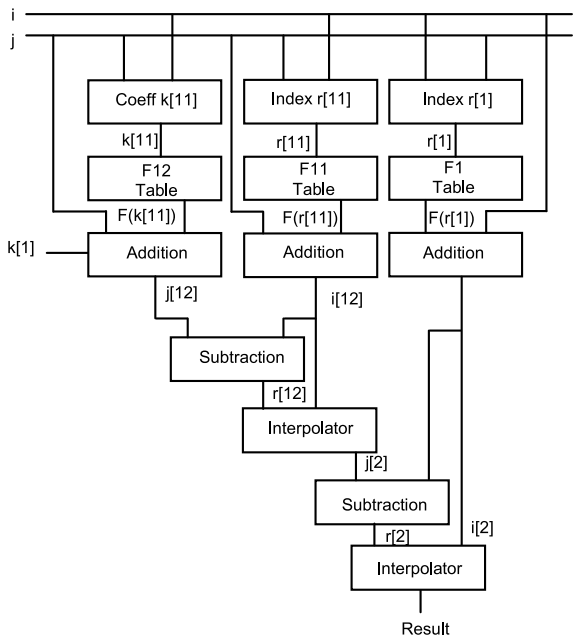

Fig. 5. Conceptual arrangement of second-order co-transformation.

$$\begin{aligned} 2^i - 2^j &= (2^i - 2^{j+k[1]}) - (2^j - 2^{j+k[1]}) \\ &= (2^i - 2^{j+k[1]}) - ((2^j - 2^{j+k[1]+k[11]}) - 2^{j+k[1]+k[12]}), \end{aligned} \tag{14}$$

where

$$2^{k[11]} + 2^{k[12]} = 1, \text{i.e.,} k[12] = \log_2(1 - 2^{k[11]}). \tag{15}$$

The four subtractions comprising (14), and their respective indices r, will now be numbered as follows:

$$2^i - 2^j = \underbrace{\underbrace{(2^i - 2^{j+k[1]})}_{1} - \underbrace{(\underbrace{(2^j - 2^{j+k[1]+k[11]})}_{11} - 2^{j+k[1]+k[12]})}_{12}}_{2}. \tag{14A}$$

Again, $k[1]$ is selected such that index $r[1]$ falls on the nearest modulo-$\Delta[1]$ boundary beneath $j - i$, and $F(r[1])$ is obtained directly from lookup table F1, containing $F(r)$ for $-1 < r < -\Delta[1]$ at modulo-$\Delta[1]$ intervals. However, $\Delta[1]$ is now fixed at a larger value than was the case in the first-order arrangement, thereby shortening the index to the F1 table by the number of additional bits used. Previously, this would have caused a corresponding increase in size of the index to the F2 table. Now, however, coefficient $k[11]$ is similarly selected such that $r[11]$ falls on the modulo-$\Delta[11]$ boundary beneath $j + k[1] - j = k[1]$, and $F(r[11])$ is obtained from table F11 which contains $F(r)$ for $-\Delta[1] \le r < -\Delta[11]$ at modulo-$\Delta[11]$ intervals. This reduces the index to the F11 table by the number of bits representing $\Delta[11]$. The final coefficient, $k[12]$, is obtained by lookup of table F12 indexed by $k[11]$, itself represented by the same number of bits as $\Delta[11]$. The conceptual arrangement is shown in Fig. 5. The index r has effectively been split into three

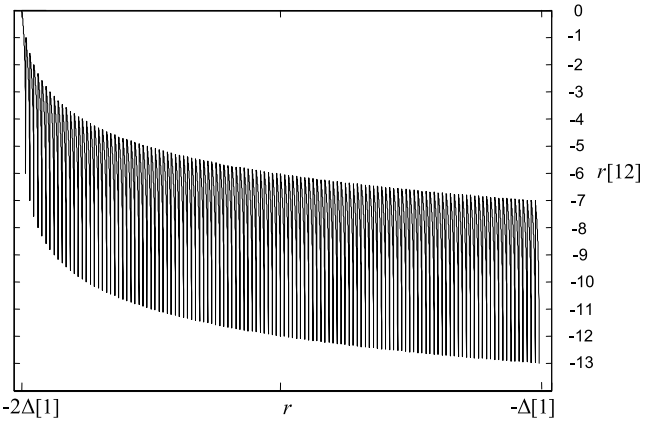

Fig. 6. Value of $r[12]$ for $-2\Delta[1] < r < -\Delta[1]$.

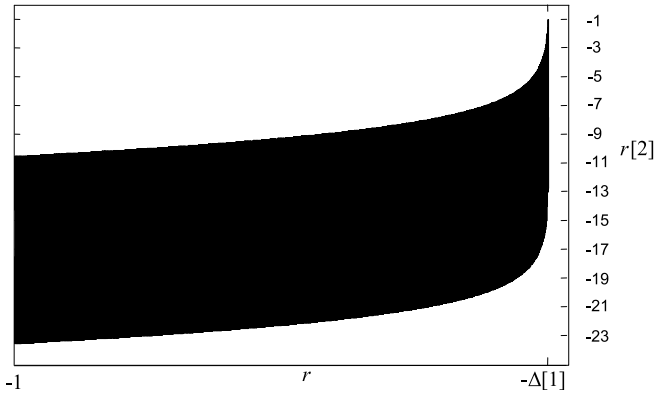

Fig. 7. Value of $r[2]$ for $-1 < r < -\Delta[1]$.

partitions, each of which will optimally be about a third of the length of the original.

Thus:

$$r[1] = (((j-i)\,\mathrm{DIV}\,\Delta[1]) - 1) \times \Delta[1] = j + k[1] - i, \quad (16)$$

$$k[1] = -(((j-i)\,\mathrm{MOD}\,\Delta[1]) + \Delta[1]) = i - j + r[1], \quad (17)$$

$$\begin{aligned} r[11] &= -(((j-i)\,\mathrm{MOD}\,\Delta[1]) + \Delta[1]) + ((j-i)\,\mathrm{MOD}\,\Delta[11]) \\ &= k[1] + k[11], \end{aligned} \quad (18)$$

$$k[11] = ((j-i)\,\mathrm{MOD}\,\Delta[11]) = r[11] - k[1], \quad (19)$$

$$k[12] = F(k[11]). \quad (20)$$

Subtractions 11 and 1 of (14A) are performed directly by lookup of their respective function tables. Subtraction 12 then generates an index

$$\begin{aligned} r[12] &= k[1] + F(k[11]) - F(r[11]) \\ &= k[1] + \log_2((1 - 2^{k[11]})/(1 - 2^{k[1]+k[11]})). \end{aligned} \quad (21)$$

The value of $r[12]$ varies with the original r as shown in Fig. 6, where $-2\Delta[1] < r < -\Delta[1]$, i.e., r lies across the range of one $\Delta[1]$. In the arrangement used for this illustration, $\Delta[11]$ is 6 bits and $\Delta[1]$ is 13, i.e., r is partitioned into low, middle and high-order segments of 6, 7 and 10 bits respectively. This is not the most optimal partitioning, but was chosen for this illustration to keep the graph to a manageable size. The modified value $r[12]$ exhibits a repeating pattern of subintervals across each $\Delta[11]$. With the exception, discussed below, of the extreme left subinterval, $r[12] < -1$. Note, in fact, that for the point in each subinterval where $k[11] = 0$, $r[12] = -\infty$. These points have been omitted from the graph, and in practice they are ignored because the subsequent calculation of $F(r[12])$ is consequently zero.

To illustrate the difference between the leftmost subinterval and the others, it is necessary to consider the behaviour of $r[12]$ as r progresses across the range of $\Delta[1]$. In the first subinterval at the left of the figure $k[1] < \Delta[11]$, and $k[1] + k[11] = \Delta[11]$. To the far left of this subinterval, $k[11] \approx \Delta[11]$, and since $k[1]$ is small, $r[12] \approx 0$. Throughout this subinterval the middle partition is zero. It is therefore possible to treat this subinterval as a special case of the first-order arrangement, in which the second-order coefficient $k[11]$, table F12 and index $r[12]$ are analogous to the first-order $k[1]$, F2 and $r[2]$. The new value $r[2]$ bypasses the first interpolator and is passed directly to the second interpolation stage. Throughout the next subinterval, $k[1] + k[11] = 2\Delta[11]$. To the far left of this subinterval, again, $k[11] \approx \Delta[11]$, but since $k[1]$ and $k[11]$ are both small, the exponential terms are approximately linear in behaviour and $r[12]$ is therefore ≈ -1. From here on, $r[12] < -1$. Except in the case just mentioned, subtraction 12 in (14A) is then completed in the first interpolator, which is positioned as shown in Fig. 5.

The result of subtraction 12 is then itself subtracted from the result of subtraction 1. Its index $r[2]$ is

$$\begin{aligned} r[2] &= j - i + F(r[12]) + F(r[11]) - F(r[1]) \\ &= j - i + F(r[12]) + \log_2((1 - 2^{k[1]+k[11]})/(1 - 2^{k[1]+j-i})). \end{aligned} \quad (22)$$

The value of $r[2]$ is plotted in Fig. 7. In all cases, $r[2] < -1$, as illustrated in the plot over the range $-1 < r < -\Delta[1]$. The subtraction can therefore be performed with a second pass of the interpolator. The result, again, is

$$\log_2(2^i - 2^j) = i + F(r[1]) + F(r[2]). \quad (23)$$

In a manner analagous to that of the first-order arrangement, in which the value of $r[2]$ falls into one of three regions, here it is separated into four. Again, this depends on the original operands i and j. For $j - i \le -1$, $r[2]$ is taken directly as $j - i$, and will lie in the linear region of F from which $F(r)$ is obtained by interpolation. For $-1 < j - i < -\Delta[1]$, $r[2]$ is derived as shown above, and now has a maximum of approximately -1. Thus it also lies in the linear region of F, and $F(r)$ is similarly obtained by interpolation. In the third region, $-\Delta[1] \le j - i < -\Delta[11]$, the high-order partition is zero and subtractions in this region can therefore be processed with a first-order technique using the F11 and F12 tables. Finally, $F(r)$ for $-\Delta[11] \le j - i < 0$ is taken directly from the F12 table.

TABLE 1
Analysis of Subtractions in Application Examples

	1. No. of Additive Operations (millions)		2. % Subtractions in the Range	3. % Additive Operations Comprised by Subtractions in the Range	
	Add	*Subtract*	$-1 < r$	$-1 < r \leq -0.5$	$-0.5 < r$
Recursive Least Squares	0.52	0.46	11.0	1.09	4.08
Fast Affine Projection	4.42	3.55	3.20	0.72	0.71

In terms of hardware implementation, it is possible to make optimisations similar to those used in the first-order arrangement. The initial calculation of $r[1]$, $k[1]$ and $k[11]$ can be done by rearranging the bits internally, and although several additions and subtractions are required, they can be reordered so that only additions are involved. Although Fig. 5 shows the conceptual algebraic layout, the co-transformation unit itself terminates after the calculation of $r[12]$ and $i[12]$. These values are passed to the interpolator where subtraction 12 of (14A) is completed, giving $j[2]$. The arrangement of Fig. 3 is modified with a feedback path to return this result to the top of the ALU, where $i[2]$ has now been stored in a holding register. These values form the inputs to subtraction 2 which, having $r < -1$, automatically bypasses the co-transform unit and proceeds directly to the second interpolation.

Rows 4, 5 and 6 of Fig. 2 review the three interpolators in rows 1, 2 and 3, each of them now in conjunction with a second-order co-transform optimally partitioned with respect to table size, i.e., with $k[11]$ at 8 bits and $k[1]$ at 16. The results will be discussed in Section 6, but it is immediately evident that the co-transform is now realised in 21,248 bits (640 words) instead of 129,024 (4,096 words). The F1, F11 and F12 tables are now 4,096, 8,448 and 8,704 bits (128, 256 and 256 words) respectively. This represents a reduction to about 16 percent of the original size, and has the major practical advantage that the tables can now be conveniently synthesised in logic. It does, however, come at the expense of a vastly increased delay for subtractions using the co-transformation. These now require two passes through the interpolator, and will therefore have around twice the delay of a direct subtraction.

From this, two questions follow. The first concerns the best region of application of the transform, which can consist of all or any part of the range $r > -1$. Reducing the range will reduce the number of subtractions using it, while also reducing the co-transform table sizes but increasing those of the interpolator. A second question relates to the desirable properties of the interpolator itself. Each of these questions raises a number of related issues, which are examined in the following two sections.

5 Use of the Co-Transformation

The first-order co-transform, as implemented on the ELM, relied on two tables with 11 address bits each; that is, it was applied only over the range $r > -0.5$. This minimised the overall storage in this unit. The second-order arrangement has much smaller tables, and, by eliminating the corresponding interpolator table, minimal storage will consequently occur now when it is applied over its full operative range of $r > -1$. On the other hand, this will increase the number of subtractions using it, and hence the time penalty in doing so. To quantify this, we analysed two examples of the more advanced types of digital filter, this being the area that has probably aroused the most interest as an application for LNS processing. Each application was programmed in optimised assembly language and executed on the ELM simulator, which had been modified to count the number of subtractions lying in the ranges over which the co-transform might be deployed.

The first application comprised a QR-Recursive Least Squares filter of order 16, running over 2,000 timesteps. This was introduced in [1], and the program, input data and results were exactly as described there. The second was based on the Fast Affine Projection algorithm, used here for echo cancellation. From the noisy signal presented to its input, it yielded an estimate of the error at each timestep. This was also run over 2,000 timesteps. In each case, input data made significant use of all parts of the dynamic range.

The proportion of subtractions lying within the fullest range of deployment of the co-transform unit is shown in Table 1, col. 2. Normally, however, the same hardware will be used both for additions and subtractions, so any penalty resulting from the use of the transform will be incurred relative to the total number of additive operations. This is shown in col. 3, which is itemised to show the lower and higher halves of the range.

These results provide little reason to restrict the application of the co-transform unit. Even when deployed over its full range of $r > -1$, only about 5 percent of additive operations will use it in RLS, and less than 2 percent in FAP. On the other hand, this study is based on only two specific examples, and we are aware of evidence from other workers suggesting that some applications might have a higher proportion of subtractions for which r is close to zero. In what follows in this paper, we have applied the co-transform throughout its full range $r > -1$, but this is undoubtedly an area that needs further work.

6 Interpolation with Second-Order Co-Transformation

The second-order co-transform can be applied to any of the interpolators described in Section 2 and shown in the first three rows of Fig. 2. Note, however, that interpolation is not the focus of this paper; rather it is the purpose of this section to discuss the issues relating to the integration of the interpolator with the co-transform unit, and to illustrate the performance of the co-transform in operation in a practical setting. For full details of the interpolators the reader is referred to the referenced publications.

TABLE 2
Error Analysis of Selected Interpolators with Second-Order Co-Transform

		$e_{\text{max rel arith}}$	$e_{\text{min rel arith}}$	$e_{\text{av rel arith}}$	$\lvert e\rvert_{\text{av rel arith}}$
Modified Chester	Add	+0.4527	−0.4623	+0.0077	+0.1745
	Subtract	+0.4987	−0.4604	+0.0024	+0.1738
Minimax	Add	+0.4720	−0.4944	−0.0015	+0.1721
	Subtract	+0.4626	−0.4617	+0.00055	+0.1719

Fig. 2, rows 4, 5 and 6 show the storage requirements for each of the interpolators previously discussed, this time in tandem with a second-order transform. In Lewis' interpolator, storage for subtractions near the singularity will now account for 30 percent of the total, which itself will see a reduction to about 25 percent of its original size. This is by far the most compact arrangement in terms of storage, but its delay path is longer than the others. With the ELM interpolator and that of Chester, storage for subtractions close to the singularity is reduced to 9 and 16 percent respectively, and the totals have been reduced to 65 and 54 percent of their original values. Chester's has less storage overall, but a longer delay path. Both, however, include large P tables of 1 kword or more, synthesis of which would be inefficient and time-consuming.

An ideal interpolator for use with the co-transform would have the shortest possible delay path, both because high speed is an objective in itself, and to avoid an unreasonably long delay when the co-transform is used twice. Coefficients would be delivered directly from the table to the multipliers, and separate tables for each partition would simplify address generation. For ease of synthesis the tables themselves would be small. Two further interpolator designs were of interest from this perspective.

Modified Chester A modification of Chester's design was also described in [8] and some minor changes to it in [21]. There is no run-time subtraction to derive D, the values of which are instead held on a separate table, and therefore no need to interleave the F table. It uses a smaller P table, at the expense of larger F, D and E tables. Essentially this arrangement is the same as the ELM interpolator, except that the D table represents the slope of the chord instead of the tangent, and the E and P tables are recalculated accordingly. Its storage requirements are given in Fig. 2, row 7.

Minimax We also developed a Minimax-based system with negative r. Unlike Fu, we did not minimise the hardware by evaluating the polynomial with Horner's method but, rather, minimised the delay by direct evaluation using three multipliers.

Similar in complexity to Modified Chester, the F, D and E tables of the latter are replaced by tables of the 0th, 1st and 2nd order coefficients. These tables have been calculated with the Sollya utility [24], which minimises the storage requirement for results of a specified accuracy. The P table of the Modified Chester interpolator is correspondingly replaced by a multiplier that forms the square of its argument. Each partition is divided into 128 intervals. Storage for this scheme is shown in Fig. 2, row 8.

In both the latter interpolators, delay paths were further minimised, e.g., by combining consecutive arithmetic elements to eliminate carry-propagate additions from all but the last element in a chain. No more aggressive optimisation was attempted, however, and it is possible that either arrangement could be further improved.

An error analysis for the two interpolators is presented in Table 2. They are broadly similar except that Minimax has a smaller bias, which reflects the objective of the utility with which it was designed. Worst-case accuracy in each case is within FP-equivalent limits. The Modified Chester interpolator requires 183,296 bits of table space, against 110,080 for Minimax.

7 LNS ALUs Compared with FP

These two designs were synthesised and routed in the Faraday UMC 0.18 μm technology, using Synopsis Silicon Compiler and Cadence SoC Encounter. Areas and delays are presented in Fig. 8. Areas are itemised between those of the multiplication-division units and the combined addition-subtraction units. Delays are itemised for multiplication-division, addition-direct subtraction, and co-transformed subtraction.

The Minimax interpolator includes an additional multiplier that forms the quadratic term, so there is less difference in the silicon areas than in the lookup table sizes. The area of the Modified Chester interpolator is 583,550 μm^2, and that of Minimax 474,438 μm^2. The Modified Chester design has a delay of 6.97 ns. Its P table returns values in which each bit is a function of its nine address inputs. For Minimax, the longer delay of 9.30 ns is due to the extra multiplier. Studies, e.g. [22], [23], have shown that a dedicated squaring circuit can reduce the delay of a general multiplier by up to 25 percent. This would amount to something less than a nanosecond in this case, but is an improvement that might be attempted.

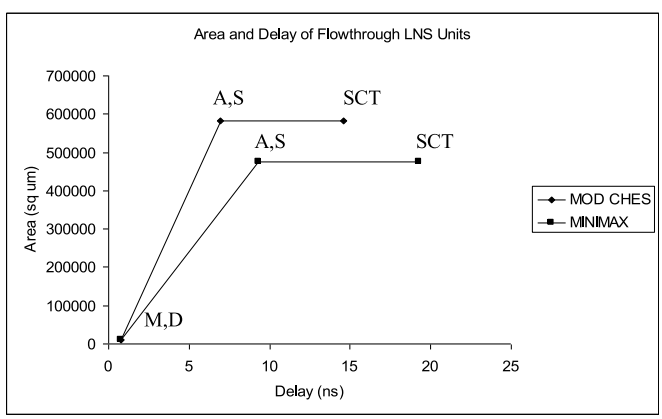

Fig. 8. Area and delay of flowthrough LNS units routed at 0.18 μ.

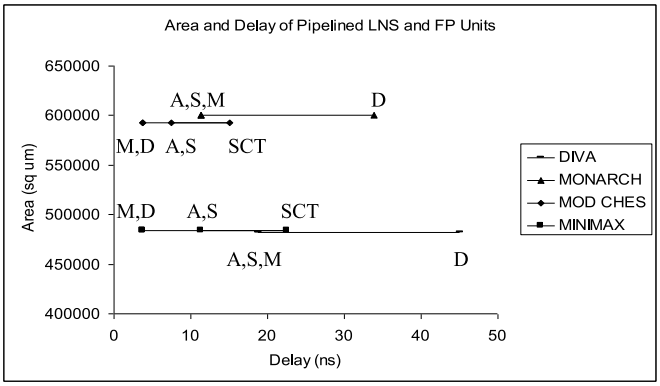

Fig. 9. Area and delay of pipelined LNS and FP units routed at 0.18 μ.

Kwon et al. [6] present a comparison of two 32-bit FP designs, also synthesised and routed for 0.18 μm fabrication, and include data for speed and area. The first design, DIVA, is optimised for minimal area. It is clocked at 266 MHz, and performs addition and multiplication in five cycles and division in 12. It has a synthesised area of 481,635 μm^2. MONARCH is also clocked at 266 MHz, but is optimised for speed. Delays and silicon areas for these devices are summarised in Fig. 9. (Kwon does not disaggregate the areas of the various functional units so the total areas are used throughout).

Also shown in Fig. 9 are the corresponding figures for the two LNS designs, the delay times of which have been fitted into a multiple of this 266 MHz clock period. With the Modified Chester arrangement, addition and direct subtraction could be completed in two cycles (7.52 ns) and co-transformed subtractions in 4 (15.04 ns). Multiplication and division would complete in one cycle, although this includes a large amount of slack time. The routed area of this ALU, including that of the multiplicative operators, would be 593,071 μm^2. For Minimax, addition would require three cycles (11.28 ns) and co-transformed subtractions 6 (22.56 ns). Its routed area is 483,959 μm^2.

The similarity in areas between the faster LNS and FP designs, Modified Chester and MONARCH, conveniently allows for a direct comparison of their speeds. The LNS design completes in 67 percent of the time of its FP counterpart, except in co-transformed subtractions where the ratio is 133 percent. On the assumption, based on the figures presented in Table 1, that 5 percent of subtractions require the use of the co-transform, then the LNS will have an average delay 70 percent of that of FP. The two slower designs, Minimax and DIVA, are likewise very similar in area and here the LNS design has 60 percent of the delay of the FP unit, or 120 percent for co-transformed subtractions, an average of 63 percent. Multiplications are accelerated between three and five times, and divisions up to 12.

In [25], Saleh and Swartzlander report the area and flow-through delays for 32-bit FP arithmetic in a more recent 45 nm fabrication. Their objective was to develop a fused dot-product unit, but they give the metrics for the discrete operators in [25], Table 2. These are reproduced here in Table 3, col. 1.

We do not have access to an equivalent 45 nm process, but were able to resynthesise and route the two LNS designs for the Faraday UMC 65 nm process. Results for these two units are given in Table 3, cols 2 and 3.

Interpretation of these results is a little tenuous, being circumscribed not only by the different geometries but also by the fact that the FP unit does not perform division. The 65 nm Modified Chester arrangement performs an addition or direct subtraction with 55 percent of the delay of the 45 nm FP unit in col. 1, and a co-transformed subtraction with 118 percent of this delay. Using the weighted average described above, a mixture of direct and co-transformed subtractions will complete with 58 percent of the FP delay. Multiplications are about 14 times faster. With a reduction to 45 nm fabrication, these speeds could only improve. The LNS design does occupy almost three times the area of the FP unit, but applying a 50 percent reduction as one moves from 65 to 45 nm, it would become 1.5 times the size. The comparison would then have to account for the assumed size of an FP division unit. If this were taken to be, say 25 percent of that of the adder and multiplier, then this LNS design might occupy 1.2 times the area of the FP unit. This improvement to 58 percent of the delay (or better at 45 nm) with 1.2 times the area seems roughly consistent with the results observed at 0.18 μm: 70 percent of the delay with equal area.

In the case of the Minimax design additive delays are 75 percent of those of the FP unit, and co-transformed subtraction 158 percent, a weighted average of 79 percent. On the same assumptions as above, a 45 nm realisation would occupy a similar area to that of the FP unit.

8 Processor Integration

We will now consider the integration of this unit into a processor. In [1] we discussed the architectural implications that arose in the ELM. Reasoning that the main strength of the LNS is the low latency and small resource requirement of its multiplicative operations, we developed a short vector machine with parallel replicated multipliers and, as far as

TABLE 3
Delay and Area of Some Arithmetic Units Routed in Smaller Geometries

	1. Saleh & Swartzlander 45 nm		2. Modified Chester 65 nm		3. Minimax 65 nm	
	Delay (ns)	*Area (μm^2)*	*Delay (ns)*	*Area (μm^2)*	*Delay (ns)*	*Area (μm^2)*
Add / Sub	1.644	3,811	0.91	38,661	1.24	31,432
Sub (Co-tr)	-	-	1.94		2.60	
Mul	1.804	9,482	0.13	537	0.13	537
Div	-	-				

TABLE 4
Throughput Degradation of Co-Transformed Operations

	$n=16$ $t=1$	$n=32$ $t=1$	$n=32$ $t=2$	$n=64$ $t=1$	$n=64$ $t=4$
ELM	0.96	0.98	0.96	0.99	0.96
Pipeline	0.90	0.94	0.89	0.97	0.89

possible, adders. The emphasis on low latency was consistent with a flowthrough implementation of the adders, which the various ROM elements would have made awkward to pipeline, besides restricting the clock speed.

With the removal of any explicit ROM cells, the last point in this justification becomes invalid. Registers can now be inserted at any convenient point in the adder datapath and, with regard not to have an unduly adverse impact on latency, a modest degree of pipelining can be considered. The latencies shown in col. 2 of Table 3 would then suggest that an operating frequency approaching 2 GHz will be achievable with a two-stage adder pipeline and single-stage multiplier.

On the ELM there are two parallel addition/subtraction units that stride through consecutive pairs of elements in a vector. These units are not pipelined, and normally complete in three cycles. If either element requires a co-transform then the system simply stalls for an additional cycle to allow time for it. Although this will hold up the other unit, which may not have needed the co-transform service, the relatively infrequent occurrence of these transforms and the fact that the system stalls for only one cycle more than the default three, render this a minor loss of efficiency. A vector of length n that does not involve a co-transform will execute in $3n/2$ cycles. If, however, t elements require the transform (assuming they are not located in the same pair), then this will increase to $(3n/2)+t$ cycles, degrading the throughput by a factor of $3n/(3n+2t)$.

With the two parallel units now pipelined into two stages, a pair of intermediate results including one that requires the co-transform, emerging from the second stage will now be recycled to the first. Including a one-cycle lead-off, a vector with no co-transformed elements will occupy $(n/2)+1$ cycles, increasing to $(n/2)+1+t$ when the co-transform is involved. The performance degradation for various values of n and t, where t/n is in broad agreement with the values reported in Table 1, are shown in Table 4.

The pipelined implementation will evidently lose up to about 11 percent of its peak performance, under workloads representative of those used in Section 5. This de-rating factor could be applied to any of the designs developed in Section 7. A pipelined version of the Modified Chester design, for example, with a clock period 55 percent that of an FP unit, would see this relative advantage reduce to 62 percent.

9 Conclusion

The LNS promises to outperform FP arithmetic, at least at 32 bits, but one of the most cumbersome aspects of the design of an LNS arithmetic unit is the size of the ROM-based lookup tables required for accurate interpolation of the subtraction function. Whereas the addition curve is well behaved, and uses tables that are small enough to permit convenient synthesis, that for subtraction exhibits a difficult singularity. Although a study of some sample DSP programs revealed that subtractions in this region would comprise no more than about 5 percent of executed additive operators, examination of different arithmetic units indicated that storage of the necessary values for these subtractions could account for up to 83 percent of storage overall.

Here, we have described a development that substantially reduces the storage needed to implement these subtractions. It could be used either as a replacement for the existing co-transform in units that currently include one, or as a replacement for the large amounts of storage involved in interpolating directly. Storage for the respective subtractions would now account for 30 percent or less of the totals, which themselves would be reduced to between 65 and 25 percent of their previous sizes. The removal of these large tables brings within easy reach the possibility of synthesising the remaining smaller ones, thereby eliminating the timing and layout problems associated with ROM elements, facilitating a significant improvement in speed, and enabling a pipelined architecture if desired.

The drawback of this modified transform is that, in the few percent of additive operations that require it, it necessitates the re-use of the interpolator.

In order to illustrate the benefits of the new co-transform unit in what might be a contemporary setting, we studied two other interpolators in detail, both with good speed and area characteristics, and both easy to synthesise. These were developed into complete arithmetic units, which were compared with FP designs that had been independently designed in comparable geometries. LNS units performing the four primary operations will occupy a broadly similar area to those of their FP equivalents, and additive operations will deliver equivalent accuracy. Additive delays will be between about a half and three-quarters of those of FP, and the units can be pipelined.

Co-transforms of the third (or higher) orders are possible, but would require a third pass through the interpolator and at 32 bits would yield only a marginal reduction in storage. They might, however, hold promise at longer wordlengths. Apart from continuing improvements at 32 bits, this is another avenue for exploration.

Aside from this, the question of power dissipation is coming increasingly to the fore, and remains for further consideration. Likewise, the widespread interest in custom datapaths, or compound primitives like fused multiply-add, may lead to some interesting application studies.

Acknowledgments

The idea of partitioning r into three segments instead of two was suggested by the late David Kinniment, on reading the

draft of [7] before it was submitted. The initial high-level code for the Recursive Least Squares algorithm was written by Jiri Kadlec, and that for the Fast Affine Projection by Felix Albu. Section 3, which forms the background to the new work presented in this paper, is based on [7], which was originally published by the IEE. The authors are grateful to the reviewers for their comments, including a particularly helpful suggestion as to the line of approach that was subsequently adopted. J.N. Coleman is the corresponding author.

REFERENCES

[1] J. N. Coleman, C. I. Softley, J. Kadlec, R. Matousek, M. Tichy, Z. Pohl, A. Hermanek, and N. F. Benschop, "The European logarithmic microprocessor," *IEEE Trans. Comput.*, vol. 57, no. 4, pp. 532–546, Apr. 2008.

[2] D. M. Lewis, "Interleaved memory function interpolators with application to an accurate LNS arithmetic unit," *IEEE Trans. Comput.*, vol. 43, no. 8, pp. 974–982, Aug. 1994.

[3] D. M. Lewis, "114 MFLOPS logarithmic number system arithmetic unit for DSP applications," *IEEE J. Solid State Circuits*, vol. 30, no. 12, pp. 1547–1553, Dec. 1995.

[4] J. N. Coleman, E. I. Chester, C. I. Softley, and J. Kadlec, "Arithmetic on the European logarithmic microprocessor," *IEEE Trans. Comput.*, vol. 49, no. 7, pp. 702–715, Jul. 2000.

[5] V. Paliouras and T. Stouraitis, "A novel algorithm for accurate logarithmic number system subtraction," in *Proc. IEEE Int. Symp. Circiuts Syst.*, 1996, pp. 268–271.

[6] T-J. Kwon, J. Sondeen, and J. Draper, "Design trade-offs in floating-point unit implementation for embedded and processing-in-memory systems," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2005, vol. 4, pp. 3331–3334.

[7] J. N. Coleman, "Simplification of table structure in logarithmic arithmetic," *Electron. Lett.*, vol. 31, pp. 1905–1906, 1995.

[8] E. I. Chester, "The LNS and its application in a high-performance matrix processor," Ph.D. dissertation, Univ. of Newcastle Upon Tyne, U.K., 2002.

[9] J. Detrey and F. de Dinechin, "A VHDL library of LNS operations," in *Proc. 37th Asilomar Conf. Signals, Syst. Comput.*, 2003, pp. 2227–2231.

[10] M. G. Arnold, T. A. Bailey, J. R. Cowles, and M. D. Winkel, "Arithmetic co-transformations in the real and complex logarithmic number systems," *IEEE Trans. Comput.*, vol. 47, no. 7, pp. 777–786, Jul. 1998.

[11] M. G. Arnold, "An improved co-transformation for logarithmic subtraction," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2002, pp. 752–755.

[12] P. Vouzis, S. Collange, and M. G. Arnold, "LNS subtraction using novel cotransformation and/or interpolation," in *Proc. IEEE Int. Conf. Appl. Specific Syst., Archit. Process.*, 2007, pp. 107–114.

[13] P. Vouzis, S. Collange, and M. G. Arnold, "Cotransformation provides area and acuracy improvement in an HDL library for LNS subtraction," in *Proc. 10th Euromicro Conf. Digital Syst. Des., Archit. Tools*, 2007, pp. 85–93.

[14] P. D. Vouzis, S. Collange, and M. G. Arnold, "A novel cotransformation for LNS subtraction," *J. Signal Process. Syst.*, vol. 58, pp. 29–40, 2010.

[15] F. de Dinechin and B. Pasca, "Designing custom arithmetic datapaths with FloPoCo," *IEEE Design Test Comput.*, vol. 28, no. 4, pp. 18–27, Jul.-Aug. 2011.

[16] J. Detrey and F. de Dinechin, "Table-based polynomials for fast hardware function evaluation," in *Proc. 16th Int. Conf. Appl.-Specific Archit. Process.*, 2005, pp. 328–333.

[17] S. Collange, research scientist, Institut National de Recherche en Informatique et en Automatique (INRIA), Rennes, France. Former developer of FloPoCo.

[18] H. Fu, O. Mencer, and W. Luk, "FPGA designs with optimised logarithmic arithmetic," *IEEE Trans. Comput.*, vol. 59, no. 7, pp. 1000–1006, Jul. 2010.

[19] M. Haselman, M. Beauchamp, A. Wood, S. Hauck, K. Underwood, and K. S. Hemmert, "A comparison of floating point and logarithmic number systems for FPGAs," in *Proc. 15th Annu. Symp. Field Programm. Custom Comput. Mach.*, 2005, pp. 181–190.

[20] S. Collange, F. de Dinechin, and J. Detrey, "Floating point or LNS: Choosing the right arithmetic on an application basis," in *Proc. 9th Euromicro Conf. Digital Syst. Des., Archit. Tools*, 2006, pp. 197–203.

[21] R. Che Ismail and J. N. Coleman, "ROM-less LNS," in *Proc. 20th IEEE Symp. Comput. Arithmetic*, 2011, pp. 43–51.

[22] K.-J. Cho and J.-G. Chung, "Parallel squarer design using pre-calculated sums of partial products," *Electron. Lett.*, vol. 43, no. 25, pp. 1414–1416, 2007.

[23] J. Pihl and E. J. Aas, "A multiplier and squarer generator for high performance DSP applications," in *Proc. IEEE 39th Midwest Symp. Circuits Syst.*, 1996, pp. 109–112.

[24] S. Chevillard, M. Joldes, and C. Lauter, "Sollya: An environment for the development of numerical codes," in *Proc. 3rd Int. Congress Conf. Math. Softw*, 2010, pp. 28–31.

[25] H. H. Saleh and E. E. Swartzlander Jr., "A floating-point fused dot-product unit," in *Proc. IEEE 26th Int. Conf. Comput. Des.*, 2008, pp. 427–431.

[26] C. Basetas, I. Kouretas, and V. Paliouras, "Low-power digital filtering based on the logarithmic number system," in *Proc Comput. Sci. IC Syst. Des.: Power Timing Modeling, Optim. Simul.*, vol. 4644, pp. 546–555, 2007.

J. Nicholas Coleman received the BA degree in music from the University of York. Subsequently, after spending a short time in commercial applications and systems programming, he joined Plessey Telecommunications Research Ltd, Poole, Dorset, initially as a software engineer but later as a hardware engineer. Here, he worked on the specification and design of a custom processor for real-time production test of the System-X telephone exchange. He then entered Brunel University, Uxbridge, Middlesex, to work for a PhD in the area of CPU architectures for VLSI design acceleration. On graduation, he joined the Department of Electrical and Electronic Engineering at the University of Newcastle upon Tyne. He lectures in computer and digital engineering, and was the co-ordinator of the ESPRIT project that resulted in the development of the ELM, and which forms the basis of the work presented here. His research interests are in high-speed processor design, and processor-intensive applications such as graphics. He is a chartered engineer.

Rizalafande Che Ismail received the PhD degree from Newcastle University, United Kingdom, in Microelectronics System Design, in 2012. He is now a senior lecturer in the School of Microelectronic Engineering, Universiti Malaysia Perlis, and is engaged in designing low-power and high-speed architectures for digital systems. He is a senior member of the Institute of Electrical and Electronics Engineers (IEEE), a member of the British Computer Society (BCS), and of the Board of Engineers Malaysia (BEM).