114 MFLOPS Logarithmic Number System Arithmetic Unit for DSP Applications

David M. Lewis

***Abstract*—An arithmetic core for DSP applications, comprising two multiplier/dividers and an adder/subtractor, using logarithmic number system (LNS) arithmetic, is described. For most operands, precision better than the worst-case precision of IEEE-754 is obtained. Three operations per cycle are performed using 69k transistors integrated in a 16 mm^2 core in 1.2 μm CMOS, offering better performance than the best previous result in only 43% of the area. The use of a new interleaved memory ROM structure and a second-order function interpolator are the key techniques that result in reduced area. This modest area allows several core units to be integrated on a chip for high performance DSP applications.**

I. Introduction

FLOATING POINT (FP) arithmetic is ubiquitous in numerically intensive computations that require a small relative error over a large dynamic range by virtue of its automatic scaling of results to preserve precision. DSP applications could benefit from these advantages, but FP is less common here due to the large silicon area required for its implementation.

Logarithmic number system (LNS) arithmetic is another number system with similar properties. LNS has origins that date back to the 19th century and has been studied extensively [1]–[3]. While its suitability for digital filtering has been demonstrated [2], only a modest number of limited precision implementations exist. This paper shows that LNS arithmetic can be used to implement arithmetic with the accuracy of FP but in lower silicon area. The LNS ALU described in this paper demonstrates the use of new algorithms and architectures for LNS arithmetic that provide slightly better numerical accuracy for most operands and much lower silicon area than does FP. The algorithms used are based on [4], which describes a new approach based on polynomial interpolation using function values stored in a interleaved-memory function interpolator. This paper demonstrates architectural and circuit techniques used to implement LNS arithmetic with superior performance per unit area compared to FP arithmetic. This result is achieved largely through use of a new ROM circuit leads to a memory that has the same effective throughput as a 3 port ROM with only a 25% area overhead compared to a single port ROM. The resulting LNS arithmetic core occupies 16.6 mm^2 in 1.2 μm CMOS and is capable of performing three single-precision operations per clock cycle at a 38 MHz clock rate. The performance per unit silicon area is a factor of about 2.6$\times$ better than any previous demonstrated FP or LNS arithmetic core.

Manuscript received May 12, 1995; revised September 23, 1995. This work was supported by Defense Research Establishment Atlantic. Fabrication was provided by the Canadian Microelectronics Corporation.

The author is with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, Canada.

IEEE Log Number 9416022.

The remainder of the paper is organized as follows. Section II provides an overview of LNS arithmetic and algorithms, briefly summarizing the techniques described in [4]. Section III describes the architecture and circuit design of the processor. Section IV presents measured results, and Section V concludes the paper.

II. LNS Arithmetic and Algorithms

LNS represents a number x by $\langle s_x, e_x \rangle$, with s_x the sign of x, and e_x the fixed-point base-2 logarithm of x. Multiplication and division are implemented with fixed-point additions and subtractions of the logarithms and are hence error-free within the number system as well as having low delay and small silicon area. Addition and subtraction in LNS are difficult and the primary focus of this paper.

Addition or subtraction of x and y with result z in LNS, where $x \geq 0$ and $y \geq 0$ is most commonly performed using (2.1) and (2.2), followed by (2.3) for addition and (2.5) for subtraction, where $f_a(\cdot)$ and $f_s(\cdot)$ are defined in (2.4) and (2.6)

$$p = \max(e_x, e_y) \tag{2.1}$$
$$r = -|e_x - e_y| \tag{2.2}$$
$$e_z = p + f_a(r) \tag{2.3}$$
$$f_a(r) = \log(1 + 2^r) \tag{2.4}$$
$$e_z = p + f_s(r) \tag{2.5}$$
$$f_s(r) = \log(1 - 2^r). \tag{2.6}$$

The central challenge in LNS arithmetic is accurate, low cost approximation of $f_a(\cdot)$ and $f_s(\cdot)$. The accuracy of calculations performed using an arithmetic processor is limited by both the precision of the number representation and precision of operations performed in it. IEEE-754 FP [5] specifies the highest level of precision in which results are required to be the number closest to the infinitely precise result.

The precisions of the LNS and FP number systems are similar, but LNS has a slight advantage. An FP representation with a F bit significand with hidden bit and I bit exponent offers similar representational accuracy and range compared to a LNS representation with F fractional and I integer bits in the exponent. The relative representational error of FP varies from 2^{-F-2} to 2^{-F-1} depending on the number being represented, but the representational error of LNS is constant, with the value $2^{-F-1+\log_2(\ln(2))} \approx 2^{-F-1.528}$. While the average error of the number systems is similar, LNS offers a uniform relative representational error that is smaller than the worst case FP

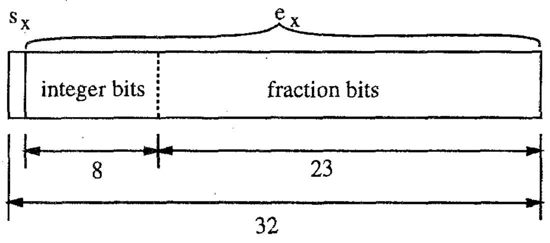

Fig. 1. LNS representation.

error. This paper adopts the accuracy of IEEE-754 single precision [5] as its goal, leading to the 32-b representation with $F = 23$ and $I = 8$ as shown in Fig. 1.

Implementation of LNS with this accuracy is achieved by using new algorithms for interpolation described in [4]. This uses a quadratic function interpolator operating on precomputed function values stored in an interleaved ROM. The interpolation is performed by determining an interpolation interval to use, calculating polynomial coefficients that approximate the function in that interval, and performing the polynomial evaluation. These are summarized as follows:

$$i = \left\lfloor \frac{x}{h} \right\rfloor \tag{2.7}$$

$$u = \frac{x}{h} - i \tag{2.8}$$

$$a_0 = f(x_i) \tag{2.9}$$

$$a_1 = \frac{1}{2} \times (f(x_{i+1}) - f(x_{i-1})) \tag{2.10}$$

$$a_2 = \frac{1}{2} \times (f(x_{i+1}) - 2 \times f(x_i) + f(x_{i-1})) \tag{2.11}$$

$$\hat{f}(x) = a_0 + u \times (a_1 + u \times a_2). \tag{2.12}$$

The interpolation interval h is chosen based on the value of x, and precomputed values of $f(x_i)$ are stored in a ROM [4]. Each interpolation requires access to three contiguous words in the ROM and uses the interleaved ROM described later in this paper.

The fact that $f_a(\cdot)$ and $f_s(\cdot)$ are transcendental makes the infinitely-precise rounding of IEEE-FP impossible in LNS arithmetic. Furthermore, the use of interpolation introduces some approximation error, which can be made arbitrarily small but not zero. This suggests that computations of exactly rounded infinitely precise calculations cannot be performed, and a weaker error goal must be chosen. In fact, recent work shows that exact rounding is possible but only at a much higher cost [6], so this paper adopts a weaker goal. The goal chosen in this paper is that the worst case numerical error of the LNS processor must be smaller than the worst case error performed in IEEE single precision, which corresponds to a relative error of 2^{-F-1} in the result, with $F = 23$ for this ALU. This relative error corresponds to a absolute error of $2^{-F-1}/\ln(2)$ in $f_a(\cdot)$ and $f_s(\cdot)$. This goal is costly for subtraction of numbers that are close in value, so the goal in this region is relaxed. Subtraction of similar numbers (defined as differing in magnitude by a factor smaller than 2) is allowed to produce an error of up to 2^{-F-1} relative to the larger input number. This is justified on the grounds that catastrophic

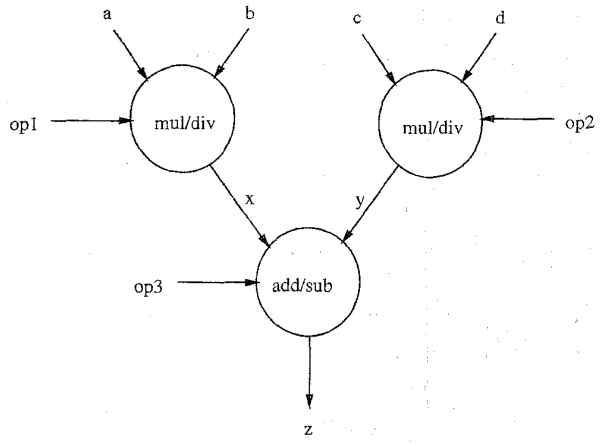

Fig. 2. LNS arithmetic unit overview.

cancellation can occur in this region, making the results less meaningful.

The interpolation intervals are determined by dividing the domain of r into a collection of regions, each of which has a fixed value of h that meets the error goal across its width. While $r \in [-256, 0]$, it is not necessary to consider this entire region. Only $r \in [-24, 0)$ needs to be considered, as both $|f_a(r)| < 2^{-24}$ and $|f_s(r)| < 2^{-24}$ for $r < -24$. It is convenient to choose the interpolation interval h based upon $\lfloor r \rfloor$, dividing the range into a set of regions $[-i-1, -i)$, except for subtraction in $r \in [-1, 0)$, where $f_s(\cdot)$ is highly nonlinear. This region is divided into 24 regions $[-2^{-i}, -2^{-i-1}), i = 0 \cdots 23,$, and h is chosen to meet the error goal within each interval. A total of 71 regions containing a total of 2218 words of ROM result. This corresponds to three modes of operation (add, subtract, subtract of similar quantities) and 24 regions for each, except that subtract in $[-1, 0)$ is handled by subtract of similar quantities.

III. Architecture and Detailed Circuit Design

The LNS ALU core described in this paper is intended for multiply-add intensive DSP applications. It can perform two multiply/divide and one add/sub per cycle, as shown in Fig. 2. The low area of the core allows integration of multiple units on a chip; for example, integration of two allows a complex multiply per cycle, and integration of six allows a FFT butterfly step per clock cycle. Although the ratio of two multiplies to a single add cannot be fully used in most applications, an LNS multiplier costs less than 2% of an adder, so it is reasonable to include two on the ALU for applications that can use them both.

The key architectural technique, and corresponding circuit-level implementation, that leads to low silicon area for the function interpolator is the interleaved ROM structure. The function interpolation algorithm must read three words from a 2218 × 31-b ROM. Replication of the data in three ROM's, or the use of a three-ported ROM could be used to implement this, but at high cost. Fig. 3 shows a lower cost multiport ROM structure that permits access to any three consecutive words

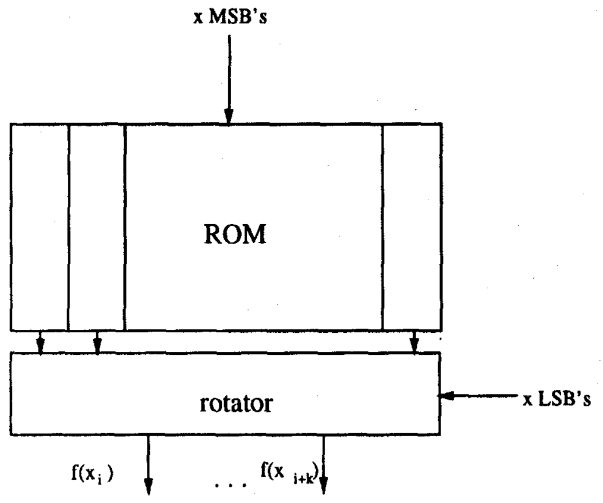

Fig. 3. Multi-ported ROM.

with less than 25% area penalty compared to a single port ROM. Each row contains 10–31-b words, partially overlapped such that row j of the ROM contains words $8 \times j \cdots 8 \times j + 9$. Eight of the ten data words in a row are unique, with an overlap of two words between rows, guaranteeing that some row contains any three consecutive words. Accessing words $i, i+1$, and $i+2$ involves reading row $i/8$ of the ROM and selecting the appropriate words.

Pipeline design is based on maximizing throughput, which results in using ROM access as the limiting factor. A 50 MHz goal was set for the circuit. Static circuitry is used where possible, and only the ROM and pipeline registers are dynamic. All pipeline registers are 3-T dynamic latches. There are a total of $6\frac{1}{2}$ stages, of which $\frac{1}{2}$ stage is used to perform multiply or divide, and 6 stages perform add or subtract. The total latency of a multiply-add is similar to FP units on contemporary microprocessors, but the latency of the individual operations is considerably different.

Fig. 4 shows the datapath for the LNS ALU. The first stage performs the multiply/divides and initial stage of the add/subtract. Careful design of this leads to fast, small logic that produces output numerical values as well as signals well suited to generating further control for interpolation intervals. The first part of this stage performs the multiply/divides using two adders/subtractors to compute $e_x = e_a \pm e_b$ and $e_y = e_c \pm e_d$. These feed into a second part that computes $r = \max(e_x, e_y)$ and $p = |e_x - e_y|$. It also decodes r to generate 24 one-hot encoded signals nx_i which, together with a signal specifying the operation, indicate which table region should be used.

The conventional approach for performing fast addition of several quantities is to use a tree of carry-save adders. This would require duplication of hardware because it would necessary to form both $e_x - e_y$ and $e_y - e_x$. The circuit shown in Fig. 5 uses e_x and e_y, each generated by a carry-propagate adder, with static carry skip with a block size of 4, and generates r and p. This circuit uses little more hardware than a single adder, and the time to compute r is overlapped

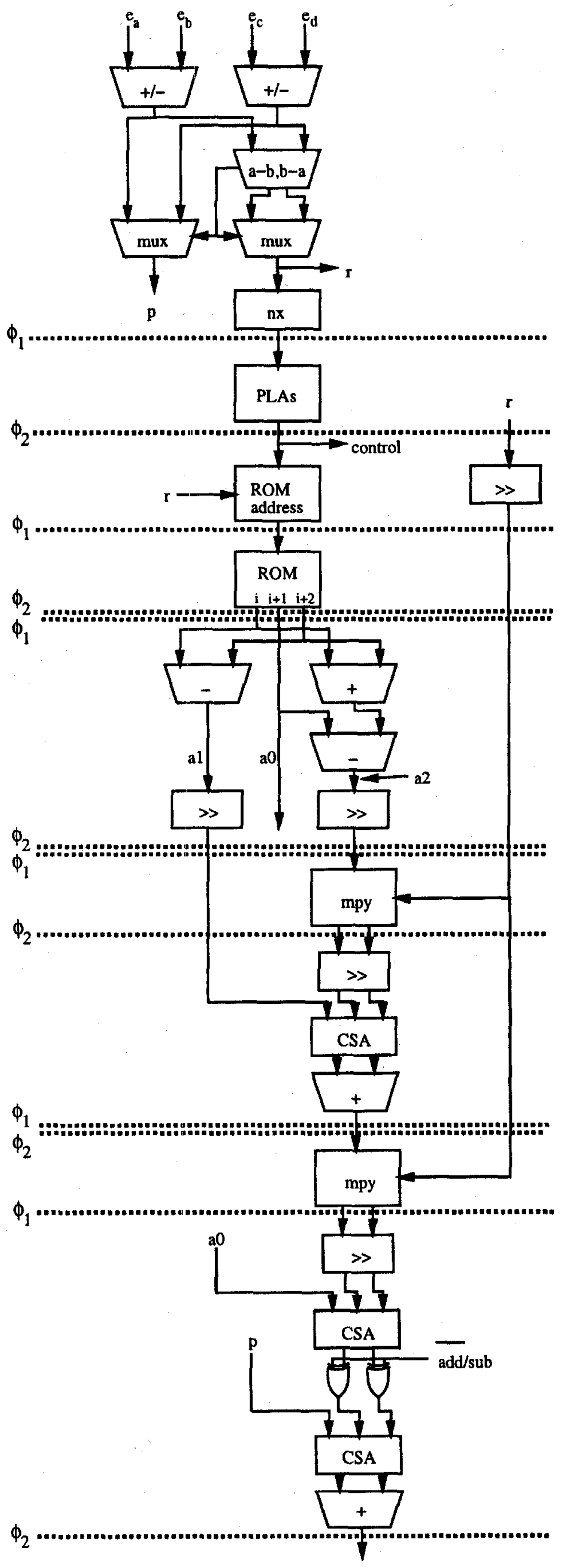

Fig. 4. Pipeline for LNS ALU.

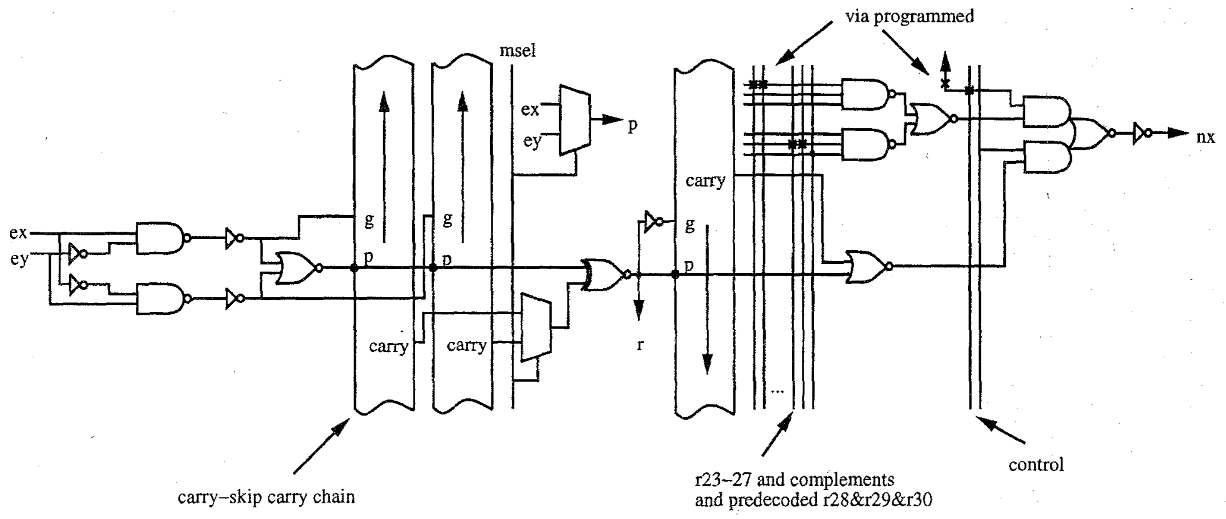

Fig. 5. Initial stage of LNS adder.

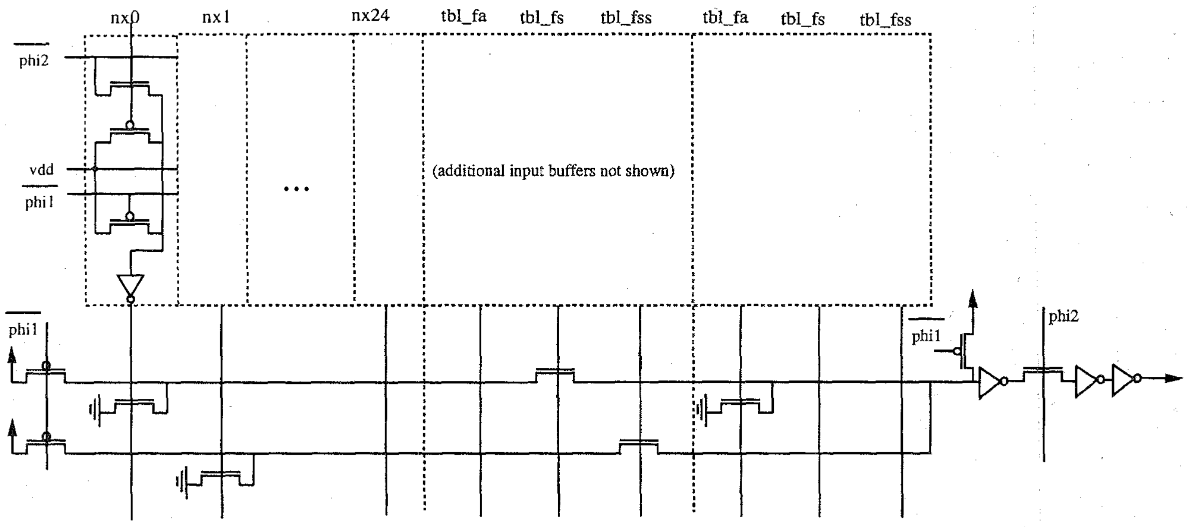

Fig. 6. Control circuit.

with computation of e_x and e_y due to the static carry chain. The circuit generates carry propagate and generate signals for both $e_x - e_y$ and $e_y - e_x$. The logic to produce the generate signals is also used to produce the propagate signal, which is the same for both operations. Two carry chains generate the carry signals for each of these. Carry-out from the MSB is used to select between $e_x - e_y$ and $e_y - e_x$ to generate the appropriate carry, which feeds into a final XOR to generate r and to select one of e_x and e_y for p. A decoder on the integer bits of r uses bits r_{23-27} and the predecoded term $r_{28} \cdot r_{29} \cdot r_{30}$ to generate a one-hot term corresponding to $\lfloor r \rfloor$. This asserts one of the nx_i unless the operation is subtraction and $r \in [-1, 0)$. In this case, a final carry chain with using $\overline{r}$ for propagate and r for generate, proceeding from MSB to LSB, is used to detect the most significant 0 in r, indicating which of the $[-2^{-i}, -2^{-i-1})$ regions r is in. This circuit design allows both the multiply/divide, generation of r, p, and nx_i to proceed in one clock cycle.

The next stage generates control for the remainder of the ALU. Control is completely defined by one of three modes and the nx_i signals. The mode signals are generated by the preceding stage and are tbl_fa corresponding to addition, tbl_fs for subtraction, and $r < -1$, and tbl_fss for subtraction and $r \geq -1$. All control for the ALU is completely defined using this two-hot encoding. The remainder of the circuit contains a number of barrel shifters that are most easily controlled using one-hot encoding. Control is simplified by directly using the two-hot control in the early stages of the pipe as input to a set of control circuits that generate the one-hot signals used further down the pipe. Because of the two-hot

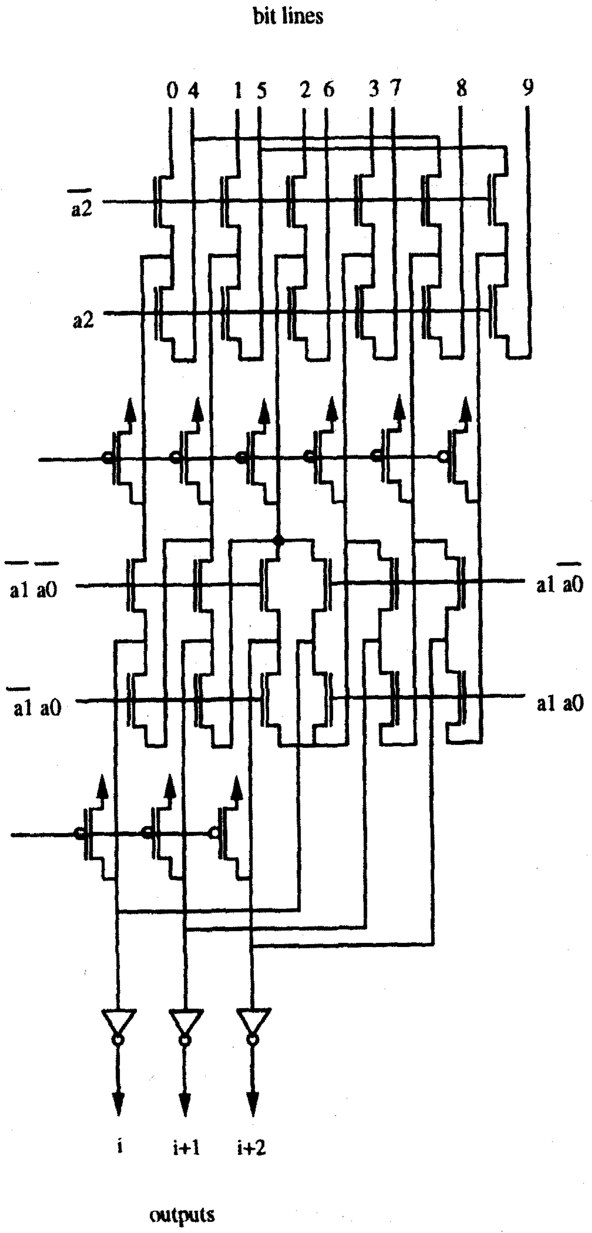

Fig. 7. Bit steering circuit for multi-ported ROM.

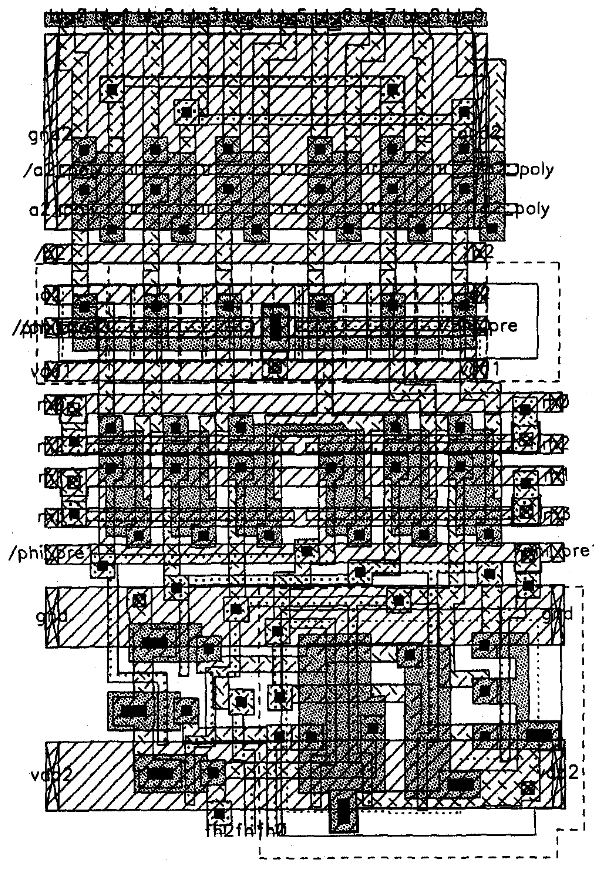

Fig. 8. Layout of bit steering circuit.

form of the control, all control signals can be generated by an expression that is the logical OR of up to three products, each with a mode ANDed with zero or more nx_i signals. The form of these is given by

$$\begin{aligned} &cntl= \\ &\quad tbl\_xx\cdot(nx_{i1}\mid nx_{i2}\mid\cdots)\mid \\ &\quad tbl\_yy\cdot(nx_{j1}\mid nx_{j2}\mid\cdots)\mid \\ &\quad tbl\_zz. \end{aligned}$$

A two-level regular structure shown in Fig. 6 generates these signals, first a set of AND's of nx_i together with a *tbl_xx* signal and then ORing these and other *tbl_yy* signals. This circuit generates outputs in half a clock cycle. The fact that the outputs are one-hot allows a denser core design than a general circuit implementing the above function.

After control signals are generated, a shifter and masker are used to generate the ROM address in half a clock cycle. Following this, ROM access is performed in half a clock cycle. Instead of the multiword approach suggested by Fig. 3, the data is grouped bit-wise, so that bits of the same significance are adjacent in the ROM. A set of 31-8:3 shifters on the outputs of the ROM select the three words. 304 × 310-b are implemented in 2465 × 1644 μm.

Sensing uses a cascade of 2:1 and 4:1 muxes, with the circuit and layout shown in Figs. 7 and 8, and permuted bit-lines. The particular set of data stored in the ROM leads to modest requirements for the electrical design. The ROM is only 37% ones, with no word-line or bit-line more than 54% populated. Further, worst case precharge supplies charge to only 28% of the array. These factors allow smaller word-line drivers and precharge circuits than would be used for a general purpose ROM. Due to small ROM size and light bit-line loading, a simple ratioed inverter suffices for sensing. An H-ROM core is used with M1 ground lines and M2 bit-line straps every 10-b and 40-b, respectively, matching the output data pitch for 1-b and 4-b, respectively.

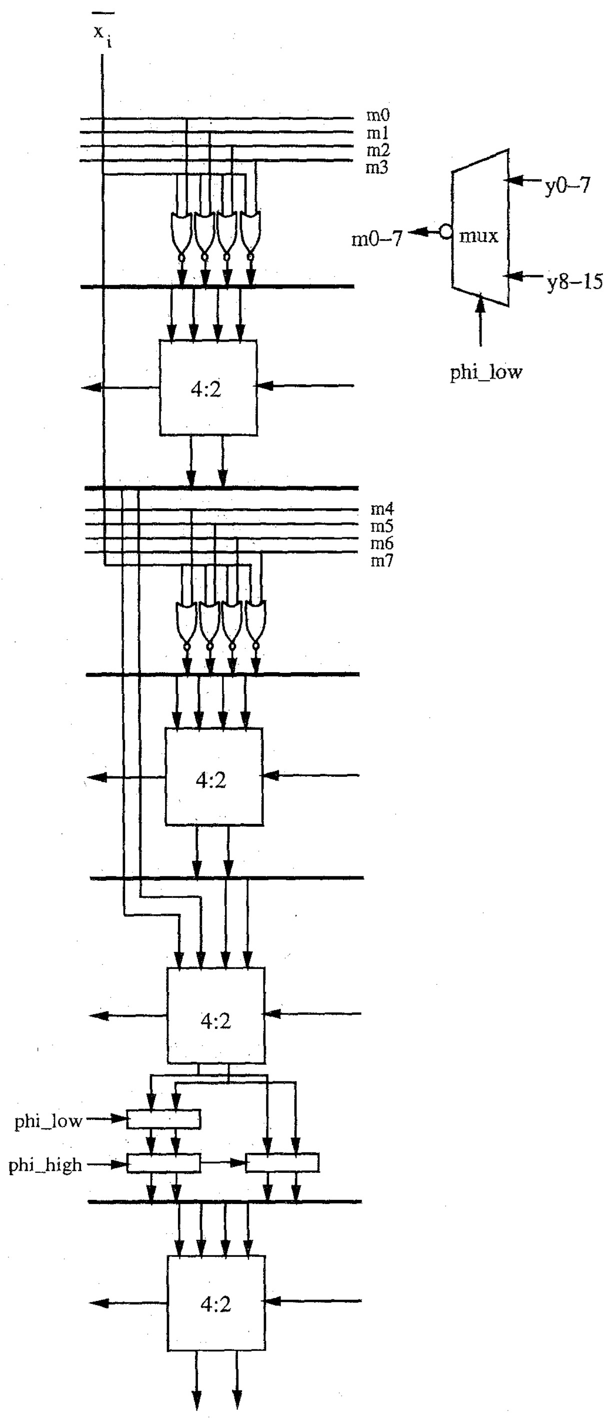

Fig. 9. Multiplier bit slice. Thick lines are wire shifters.

The remainder of the data path performs the polynomial interpolation. Barrel shifters are used at each stage to preserve the necessary precision while using the smallest possible multipliers. Data is kept in carry-save form as long as possible and only converted to irredundant form when necessary as input to a multiplier or at the final output.

The interpolation requires a 19×16 and 12×16 multiplier, which are based on the 4:2 compressor described in [12]. These are completely regular due to the use of unsigned arithmetic. Due to the low delay of this circuit, each multiplier is constructed as a $n\times$ 8 multiplier clocked twice per cycle. Fig. 9 shows the bit slice of the multiplier cell that can be cascaded without customization. The delay through the $\times$ 8 multiplier is a NOR gate plus 6 XOR gate delays, which can be performed in half a clock cycle.

The final summing stage adds or subtracts the various contributions from the interpolation in a set of carry-save adders and a final carry-propagate adder. 37-b of precision are used in these adders and rounded to 31-b to meet the accuracy goal.

IV. Design and Measured Results

Algorithm design for the processor was initially performed using a symbolic math package to verify the interpolation method and determine the required interpolation interval. Detailed design used a functional model written in "C." Because there are only $\approx 4 \times 10^8$ values of r that produce nonzero results, exhaustive verification of the design could be performed in 6 CPU hours on a 20MIP machine. A gate-level logic design was implemented but was not functionally identical to the "C" model, as the multipliers were implemented in "C" using integer arithmetic instead of the 4:2 compressor tree used in the detailed logic design. Automated testing was done at the gate-level and extracted transistor-level designs.

Ten circuits were fabricated in a 1.2 μm CMOS double metal process, of which nine were functional up to 38 MHz (5 V, 25°C), which is believed to be due to tester clock degradation. Measured worst case power is 71 mA. Fig. 10 is a die photo. Die size is 6.1×3.9 mm with a core of 5.3×3.1 mm, with area 16.6 mm^2, containing 69k MOS, of which 35k are in the ROM.

These results are superior to previously published FP and LNS arithmetic units, as shown by the comparison to other 32-b FP and LNS units in Table I. Areas are core only, estimated from photos where necessary, including raw data and data normalized to 1.2 μm technology assuming linear scaling of dimensions and inverse linear scaling for performance. F is the feature size μm, and performance is throughput in MFLOPS, using measured speeds. Performance per unit area, in MFLOP's/mm^2 is also given. This work uses higher throughput than any other unit, in only 43% of the area of the smallest previous unit [7], $2.6 \times$ better performance per unit area.

V. Conclusions

This paper has demonstrated an implementation of new algorithms and architectures for LNS arithmetic. The fabricated ALU offers precision comparable to IEEE-754 but with throughput per unit silicon area at least $2.6\times$ better than the best previous reported result. Integration of several of these ALU's on a single die can yield high throughput in modest silicon area.

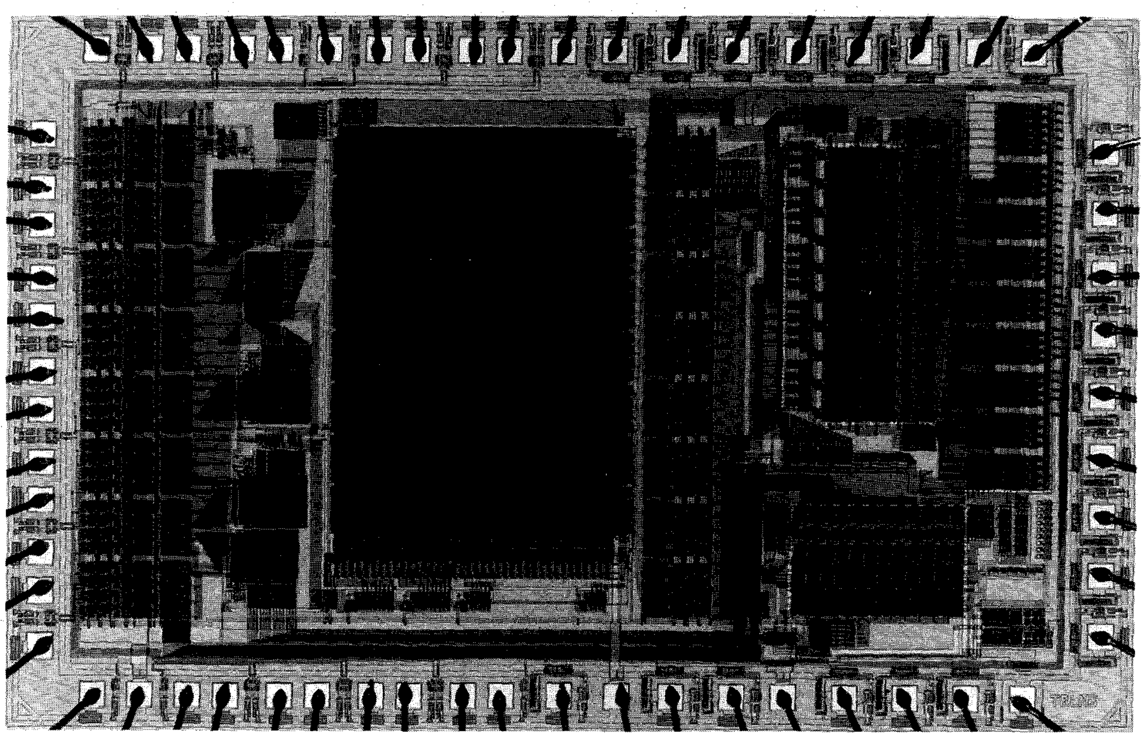

Fig. 10. Die photo.

TABLE I
COMPARISON TO 32-b FP AND LNS UNITS

Ref.	F	Area	Perf	Area (norm)	Perf (norm)	Perf/Area (norm)
[7]	1.2	38.3	100	38.3	100	2.61
[8]	0.8	52.7	100	118.6	67	0.56
[9]	1.2	43.2	30	43.2	30	0.69
[10]	1.5	86.4	30	55.3	38	0.69
[11]	1.3	120.9	40	103.0	43	0.42
this paper	1.2	16.6	114	16.6	114	6.87

REFERENCES

[1] M. G. Arnold *et al.*, "Redundant logarithmic arithmetic," *IEEE Trans. Comput.*, vol. 39, pp. 1077–1086, Aug. 1990.

[2] N. G. Kingsbury and P. J. W. Rayner, "Digital filtering using logarithmic arithmetic," *Electron. Lett.*, vol. 7, pp. 56–68, 1971.

[3] E. E. Swartzlander and A. G. Alexopolous, "The sign/logarithm number system," *IEEE Trans. Comput.*, vol. 24, pp. 1238–1242, Dec. 1975.

[4] D. M. Lewis, "Interleaved memory function interpolators with application to an accurate LNS arithmetic unit," *IEEE Trans. Comput.*, vol. 43, pp. 974–983, Aug. 1994.

[5] IEEE Standard for Floating-Point Arithmetic, ANSI/IEEE Std. 754-1985, 1985.

[6] M. J. Schulte and E. E. Swartzlander, "Hardware designs for exactly rounded elementary functions," *IEEE Trans. Comput.*, vol. 43, pp. 964–973, Aug. 1994.

[7] H. Fujii *et al.*, "A floating-point cell library and a 100-MFLOPS image signal processor," *IEEE J. Solid-State Circuits*, vol. 27, pp. 1080–1088, July 1992.

[8] F.-S. Lai, "A 10-ns hybrid numbers system data execution unit for digital signal processing systems," *IEEE J. Solid-State Circuits*, vol. 26, pp. 590–599, Apr. 1991.

[9] D. A. Staver, C.-Y. Ho, K. J. Molnar, and R. D. Baertsch, "A 30-MFLOPs CMOS single precision floating point multiply/accumulate chip," in *ISSCC 1987*, pp. 274–276.

[10] P. Y. Lu, A. Jain, J. Kung, and H. Ang, "A 30 MFLOP CMOS floating point processor," in *ISSCC 1988*, pp. 28–29.

[11] S. Komori *et al.*, "A 40MFLOPS 32-bit floating point processor," in *ISSCC 1989*, pp. 47–48.

[12] G. Goto, T. Sato, M. Nakajima, and T. Sukemara, "A 54-b × 54-b regularly structured multiplier," *IEEE J. Solid-State Circuits*, vol. 27, pp. 1229–1236, Sept. 1992.