High-Efficiency Logarithmic Number Unit Design based on an Improved Cotransformation Scheme

Youri Popoff*, Florian Scheidegger*, Michael Schaffner*, Michael Gautschi*, Frank K. Gürkaynak*, Luca Benini*†
*ETH Zürich, 8092 Zürich, Switzerland, †Università di Bologna, Italy

***Abstract*— The logarithmic number system (LNS) has always been an interesting alternative for floating point calculations since the implementation of several arithmetic operations such as divisions, exponentiations and square-roots, which are required for computationally intensive nonlinear functions, is greatly simplified in the logarithmic space. However, additions and subtractions become nonlinear operations that have to be approximated using polynomials for area efficient realizations. A particular challenge is the accuracy within the so-called *critical region* which is encountered for subtractions where the difference between the operands is close to zero. In the literature, several arithmetic cotransformations that reduce the overhead of approximating these operations have been presented. Even so, the main problem with practical LNS realizations is the area overhead when compared to standard FPUs with comparable accuracy. In this paper, we propose a highly hardware-efficient novel cotransformation concept that not only reduces the area requirements by up to 35% when compared to the state-of-the-art, but also allows the LNU to calculate single cycle logarithms and exponentiations within the same datapath. We present comprehensive results for a complete processing system that includes the LNU and an OpenRISC based core in 65nm, and 28nm technologies. We compare this implementation with a system using a standard IEEE compliant FPU and show that the LNS based system can outperform its FP counterpart by up to 4.35× in speed. The final, pipelined LNU system when implemented in 65nm occupies an area of 54.3 kGE, allows 89 MFLOP per second and consumes 15.9-136.7 pJ per operation at 1.2 V under typical conditions and 25°C.**

I. Introduction

The logarithmic number system (LNS) has been considered several times [1–8] as a replacement for conventional single-precision floating-point (FP) arithmetic since computationally intensive nonlinear function kernels can be evaluated with very low-latency in LNS. This is not only relevant for high-performance computing, but also increasingly needed for low-power, low-cost embedded applications where the demand on intensive signal processing capabilities continue to grow on a regular basis. However, the drawback of LNS is that additions and subtractions become nonlinear and have to be approximated accordingly with a dedicated LNS unit (LNU). These functions can be efficiently approximated using piecewise polynomial interpolation, with the exception of subtractions resulting in numbers close to zero. In this critical region, conventional interpolation fails to meet accuracy requirements at reasonable lookup-table cost, and more elaborate approximations using so-called *cotransformations* have to be used [2]. To this end, this paper makes the following contributions:

- We present a novel cotransformation which reduces circuit area by up to 35% based on synthesis and post-layout results for 65 nm and 28 nm technologies.
- The novel LNU also provides additional functionality such as single cycle logs/antilogs (LOG, EXP) and typecasts between signed 31 bit integers and the LNS format at no additional cost.
- We integrate this LNU, and for comparison a standard IEEE single precision FPU, into an OpenRISC processor and demonstrate that LNS can have a speedup of up to 4.35× on our benchmark set which comprises several nonlinear function kernels from computer vision, classification and regression applications.

The paper is structured as follows: first a short introduction into LNS and related work is given in sections II and III. The new cotransformation and the corresponding LNU architecture are explained in sections IV and V, respectively. Finally the results are presented in Section VI.

II. Preliminaries

A. LNS Number Representation and Format

Standard FP number systems represent a real number a as

$$a = (-1)^{s_{\text{fp}}} \cdot m_{\text{fp}} \cdot 2^{l_{\text{fp}}}, \tag{1}$$

where s_{fp} is the sign, m_{fp} the mantissa and l_{fp} the exponent. LNS represents real numbers in a similar way, but without using a mantissa. In its place the number is represented by an exponent l_{lns}, that has a fractional part:

$$a = (-1)^{s_{\text{lns}}} \cdot 2^{l_{\text{lns}}}. \tag{2}$$

The encoding used in this work has been chosen to be aligned with the IEEE 754 32-bit single-precision format [9] where 32 bit numbers consist of a sign bit and a 31 bit fixed-point exponent with 8 integer and 23 fractional bits:

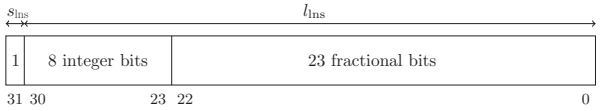

Fig. 1. Encoding of the LNS numbers used in this work.

As in the IEEE 754 standard, special values such as infinity, NaN (not a number) and zeros are encoded using special bit patterns.

B. Arithmetic Operations in LNS

Certain operations can be implemented very efficiently when working with LNS. For example, multiplications, divisions and square-roots

$$a \cdot b = (-1)^{s_a+s_b} \cdot 2^{l_a+l_b}, \tag{3}$$

$$a/b = (-1)^{s_a+s_b} \cdot 2^{l_a-l_b}, \tag{4}$$

$$\sqrt{|a|} = \left(2^{l_a}\right)^{0.5} = 2^{0.5 \cdot l_a}, \tag{5}$$

can be calculated using a single addition, subtraction or bitshift, respectively. This is an important advantage because the equivalent FP implementations are much more complex and have longer latency.

However, these simplifications come at the cost of more complex additions and subtractions which become nonlinear operations in LNS and have to be calculated accordingly:

$$a \pm b = c, \tag{6}$$

$$l_c = \max(l_a, l_b) + \log_2(1 \pm 2^{|l_a - l_b|}). \tag{7}$$

Using the positive absolute difference $r := |l_a - l_b|$, the two nonlinear functions for addition and subtraction can be defined as $\mathrm{F}^+(r) := \log_2(1+2^r)$ and $\mathrm{F}^-(r) := \log_2(1-2^r)$. These functions are shown in Fig. 2a.

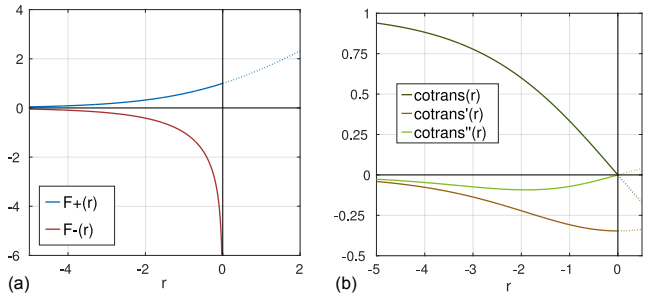

Fig. 2. (a) Plot of the $F^+(r), F^-(r)$ functions - note the singularity for $r \to 0$. (b) Plot of the cotransformation and its first two derivatives.

III. Related Work

Logarithmic number systems (LNS) have already been proposed back in 1971 and 1975 in order to increase the dynamic range of signal values in digital filters [10] and as a replacement for standard fixed-point and floating-point arithmetic [7]. The formulae for LNS addition and subtraction have already been derived in these papers, but the implementation of LNS number systems with accuracy higher than 12 bit was considered infeasible due to exponentially increasing lookup-table (LUT) sizes for these functions. Since then, several improved implementations have been proposed. For LNS with bit-widths lower than 16 bits, so-called multi-partite table [11] and high-order table based methods (HOTBM) [12] have been shown to be effective approximation methods. LNS based operations have been used to replace fixed-point operations with 16 or fewer bits in several applications such as QR decomposition [13], embedded model predictive control processors [14] and low power digital filtering with LNS [15]. LNS numbers have also been extended to be used for complex numbers [16] and quaternions [17].

As can be seen in Fig. 2a, for the most part, both functions can be easily approximated. The challenge is the function F^- as it approaches its singularity at zero. In this so-called *critical region* this singularity leads to precision problems when using piecewise polynomial interpolation or any of the aforementioned multi-partite table methods. Hence, Coleman, et al. [2] introduced the concept of a *cotransformation*. Essentially, LNS subtractions in the critical region are split into two successive subtractions where the first lies outside, and the second only in a small subset of the initial critical region. This scheme allows to significantly reduce the coefficient table size, and several improved variations thereof have been presented in [1], [3–5], [8]. At the time of writing, the most advanced design of an LNU with equivalent accuracy to IEEE single-precision FP is the one presented by Ismail, et al. [5], which is an improved version of the unit which was fabricated as part of the *European Logarithmic Microprocessor* [4].

In a paper by [18], a solution is presented that tries to combine the advantages of both standard FP and LNS representations. The main drawback in these approaches are the required typecasts between the representations which themselves are costly operations.

We present an LNU architecture that has been optimized for hardware efficiency by implementing a novel cotransformation suitable for applications where higher accuracy (IEEE single precision) is required. In addition, the resulting datapath can be reused to implement single cycle logs - and therefore also single cycle typecasts from signed 31 bit integers to LNS, greatly improving the usability of the LNU in a traditional processor. The presented LNU is part of the four-core processing system presented in [19].

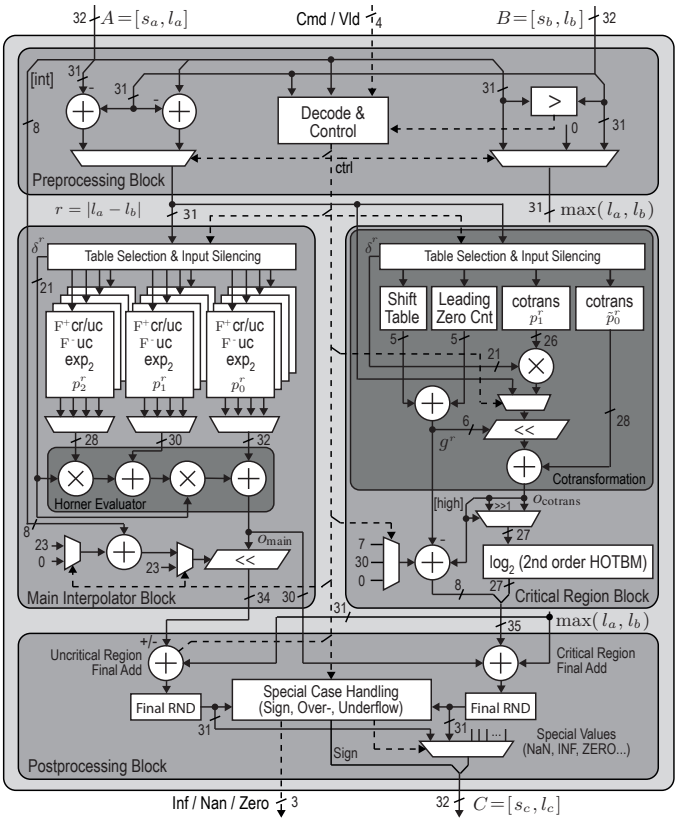

Fig. 3. Architecture of the LNU (*cr* and *uc* stand for *critical* and *uncritical*).

A. Rounding Modes and Precision

The IEEE 754 standard defines several rounding modes that can be applied after basic arithmetic operations like multiplications and additions. The default rounding mode is *round to nearest* (ties to even), and provides average and maximum relative errors of 0.1733 and 0.5 ulp (unit in the last place), respectively. However, due to the different spacing of the machine numbers in the LNS domain, an ulp in FP is not equivalent to an ulp in LNS. Therefore in [3], Coleman introduced the relations

$$|e|_{\text{avg rel fp}} = 2^{n_{\text{frac}}} \cdot \left(2^{|e|_{\text{avg lns}}} - 1\right), \tag{8}$$

$$|e|_{\text{max rel fp}} = 2^{n_{\text{frac}}} \cdot \left(2^{|e|_{\text{max lns}}} - 1\right), \tag{9}$$

where n_{frac} is the number of LNS / FP fraction bits, $|e|_{\text{avg lns}}$ and $|e|_{\text{max lns}}$ are the average and maximum relative errors in the LNS domain, and $|e|_{\text{avg rel fp}}$ and $|e|_{\text{max rel fp}}$ are the corresponding relative errors in the FP domain. Using these relations, we can calculate that for an equivalent FP precision of 0.5 ulp, the LNS design should have a maximum error less than 0.7213 ulp in the LNS domain.

B. Approximation Using Piecewise Polynomials

In order to reach IEEE single-precision accuracy, piecewise polynomial approximations of the functions F^+, F^- (outside the critical region) have been found to be very efficient and are also used in other designs such as [4], [5]. In this work for all functions except F^- inside the critical region such piecewise polynomial approximations have been used. The polynomial coefficients have been obtained by using Remez's algorithm [20] since for a given degree the maximum error is guaranteed to be minimal (hence these polynomials are also called *minimax* polynomials). In particular, we use the finite-precision aware Remez fitting algorithm available in the Sollya library, which takes coefficient quantization into account [21].

IV. Novel Cotransformation Concept

While it is considered feasible to store the functions $\mathrm{F}^{\pm}(r)$ in LUTs for low precision implementations up to around 12 bits, this approach is not practical for designs that require higher precision, such as IEEE single-precision format, since the LUT size depends exponentially on the bit width. To achieve higher precision, polynomial approximations have been found to work well [4], [5] – except for operations in the critical region (for $r \in (-1, 0]$) where F^- has a singularity at zero. In the critical region so-called cotransformations have been proposed, which are mathematical transformations of F^- attempting to reduce the size of the critical region as in [2], or to split the evaluation of F^- into sub-functions which can be approximated more efficiently as in [1], [4], [5], [8]. Our cotransformation belongs to the latter class, i.e., F^- is decomposed into

$$\mathrm{F}^-(r) = \log_2(1-2^r) = \log_2\left(\frac{1-2^r}{1+2^r}\right) + \log_2(1+2^r) \quad (10)$$

$$= \log_2(\mathrm{cotrans}(r)) + \mathrm{F}^+(r) \quad (11)$$

It should be noted that the $\log_2$ function still has a singularity at 0, but for finite precision arithmetic this function can be efficiently implemented using range reduction of the argument [22]. In our implementation the $\log_2$ function is approximated only on a reduced range [1,2), and its argument is pre-normalized using bit-shifts, allowing large and small numbers to be calculated with equal precision. The second term is F^+ and can be efficiently approximated using a second order polynomial. As shown in Fig. 2b, $\mathrm{cotrans}(r) = \frac{1-2^r}{1+2^r}$ itself is a function which behaves increasingly linear for $r \to 0$ since all higher order derivatives are very small at this point.

The cotransformation in Eq. 10 belongs to a family of functions

$$\mathrm{F}^-(r) = \log_2\left(\frac{1-2^r}{(1+2^r)^m}\right) + m \cdot \log_2(1+2^r), \quad (12)$$

which are parametrized by $m \in \mathbb{R}$. Depending on the interpolation scheme, this parameter can be used to tailor the cotransformation scheme accordingly. For the first order polynomial interpolation used in this paper, this parameter has been set to $m = 1$.

One interesting advantage of the arrangement in Eq. 10 is the presence of the $\log_2$ block, which can be reused to natively support typecasts from integers to LNS numbers as well as LNS logarithm instructions directly in hardware.

V. Hardware Architecture

The main design goal of the LNU architecture shown in Fig. 3 is to reduce the hardware overhead, mainly caused by the LUTs needed for approximations, without sacrificing accuracy and performance. As noted in Section III, to achieve the desired accuracy efficiently, different approaches are necessary depending on whether or not the operation falls into the critical region. The presented architecture consists of four main blocks that are used for different purposes: the *Pre-* and *Postprocessing Blocks*, the *Main Interpolator Block* and the *Critical Region Block (cotransformation)*. Detailed numerical analysis has shown that the best trade-off between accuracy and area is achieved when the critical region is taken as (-0.25, 0]. Since only one instruction is evaluated at a time, interpolator data paths within the two parallel blocks can be shared among all functions in order to achieve a more compact design. The following subsections explain each block in more detail.

A. Preprocessing Block

The preprocessing block decodes the command, generates all control signals for the LNU and performs operation dependent preparation steps on the two 32 bit operators $A = [s_a, l_a]$ and $B = [s_b, l_b]$. The *Main Interpolator Block* is used to calculate ADD, EXP, Float to Integer conversions (F2I) and SUB operations that fall outside the critical region. The *Critical Region Block* is needed for LOG and SUB operations in (-0.25, 0] as well as Integer to Float (I2F) typecasts that take advantage of the $\log_2$ block. Once the preprocessing block decodes the command, it calculates the absolute operator difference $r = |l_a - l_b|$ and the operator maximum in the case of binary operations such as ADD/SUB. For unary operations such as EXP/LOG and typecasts, operator B is gated to zero and A is passed through. In order to reduce the latency of the preprocessing step, the comparison $l_a > l_b$ and the operator differences $l_a - l_b$ and $l_b - l_a$ are calculated in parallel, and only the correct result is taken.

TABLE I
Lookup-table details (lower boundary specified).

$\mathrm{F}_+(\mathbf{r})$ **on (-25,0]**			**Total: 5 Segments, 355 Entries, 20'347 bits**		
Seg	Boundary	Δ	#	p_2, p_1, p_0 Bits	Size
0	-0.0625	2^{-4}	1	19, 26, 30	75 bit
1	-0.25	2^{-5}	6	17, 25, 30	432 bit
2	-8	2^{-5}	248	14, 22, 27	15'624 bit
3	-16	2^{-3}	64	12, 17, 20	3'136 bit
4	-25	2^{-2}	36	7, 11, 12	1'080 bit
$\mathrm{F}_-(\mathbf{r})$ **on (-14,-0.25]**			**Total: 6 Segments, 688 Entries, 41'312 bits**		
Seg	Boundary	Δ	#	p_2, p_1, p_0 Bits	Size
0	-0.5	2^{-9}	128	13, 22, 29	8'192 bit
1	-1	2^{-8}	128	13, 22, 28	8'064 bit
2	-2	2^{-7}	128	13, 21, 27	7'808 bit
3	-4	2^{-6}	128	13, 21, 26	7'680 bit
4	-8	2^{-5}	128	12, 20, 24	7'168 bit
5	-14	2^{-3}	48	12, 18, 20	2'400 bit
$\mathrm{Cotrans}(\mathbf{r})$ **on (-0.25,0]**			**Total: 10 Segments, 512 Entries, 24'020 bits**		
Seg	Boundary	Δ	#	p_1, p_0 Bits	Size
0	-0.000488	2^{-11}	1	21, 1	22 bit
1	-0.000977	2^{-11}	1	26, 28	54 bit
2	-0.001953	2^{-11}	2	25, 28	106 bit
3	-0.003906	2^{-11}	4	24, 28	208 bit
4	-0.007812	2^{-11}	8	23, 28	408 bit
5	-0.015625	2^{-11}	16	22, 28	800 bit
6	-0.031250	2^{-11}	32	21, 28	1'568 bit
7	-0.062500	2^{-11}	64	20, 28	3'072 bit
8	-0.125000	2^{-11}	128	19, 28	6'016 bit
9	-0.250000	2^{-11}	256	18, 28	11'776 bit
$\mathbf{2^r}$ **on [0,1)**			**Total: 1 Segments, 64 Entries, 4'096 bits**		
Seg	Boundary	Δ	#	p_2, p_1, p_0 Bits	Size
0	0	2^{-6}	64	14, 22, 28	4'096 bit

B. Main Interpolator Block

This main interpolator block implements the approximations for EXP, F^+ on the complete range (-25, 0] and F^- outside the critical region (-25, -0.25] using 2nd-order piecewise polynomial approximations which have been found to provide the best latency vs. LUT area trade-off. For a given input r, three polynomial coefficients $p_i^r := p_i(r)$ for $i = \{0, 1, 2\}$ are selected from a set of LUTs, and the approximation result is calculated using the Horner scheme

$$o_{\text{main}} = p_0^r + \delta^r \cdot (p_1^r + \delta^r \cdot p_2^r), \quad (13)$$

where δ^r are the least significant bits of r. Since LNU processes only one instruction at a time, the main Horner evaluation datapath can be shared among all function approximations. Each LUT has been subdivided into different segments, each of which contains a set of equidistantly spaced coefficient samples, as shown in Table I. The segment boundaries have been aligned to powers of two, such

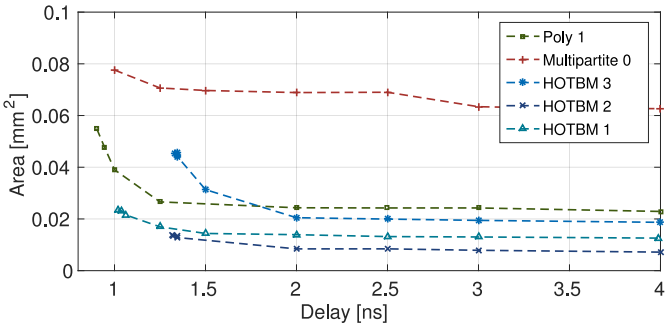

Fig. 4. AT analysis of different interpolation methods for the $\log_2(.)$ function in 65nm CMOS.

that the segment index can be easily determined by looking at the MSBs of the argument r. The coefficient bitwidths and the amount of sampling positions have been determined using a similar heuristic as described in [23]. To achieve the target accuracy, a total of five internal guard bits were found to be sufficient. Note that in Table I, F^- has only entries down to -14. Below that value, the coefficients from F^+ can be reused due to the similarity of the functions F^- and F^+ for large negative r values.

In order to calculate the EXP and F2I functions, the actual 2^r function is only approximated on the reduced range [0,1) for the fractional part of the incoming operator l_a. The full range can then be reached by left-shifting the result of this approximation by the integer part of the operator l_a.

C. Critical Region Block

The second order derivative of the novel cotransformation presented in Section IV tends to zero for $r \to 0$. This function can be efficiently implemented using a first-order minimax polynomial for the given precision requirement. The main challenge is implementing the $\log_2$ function required in the cotransformation. We have used an implementation based on a range reduction technique [22], which normalizes the function argument to the range [1,2). Such normalization can be achieved by shifting the output of the polynomial approximation. In the worst case, for very small values of r, the cotransformation output will have to be shifted by up to 23 places to the left. To have sufficient precision in this worst case, the cotransformation would have to be calculated with much higher precision which in turn would increase the LUT size considerably. In our implementation, we avoid this problem, by generating the approximation in normalized format and by storing the p_0^r coefficients in a pre-shifted format. The output of the p_1^r multiplication is then shifted at runtime by a pre-computed amount

$$o_{\text{cotrans}} = (p_0^r + p_1^r \cdot \delta^r) \cdot 2^{g^r} = \tilde{p}_0^r + p_1^r \cdot \delta^r \cdot 2^{g^r}, \qquad (14)$$

where $g^r \in \mathbb{N}$ is the pre-computed shift amount and $\tilde{p}_0^r$ are the pre-shifted coefficients. Numerical simulations have shown that the result o_{cotrans} is almost always dominated by $\tilde{p}_0^r$ since the coefficients p_1^r are all around 0.33 within the critical region. Only in the first interval from (-0.000488, 0] the output magnitude is determined by the product $p_1^r \cdot \delta^r$, since $\tilde{p}_0^r = 0$. This means that outside this first interval, g^r can always be chosen such that almost all the cotransformation results o_{cotrans} lie within the range [1,2). In very few cases, a carry propagation takes place and the result has to be shifted back by one bit. For the first interval (-0.000488, 0] the pre-shift g^r has to be computed at runtime (using a leading zero counter) since the output magnitude solely depends on the term $p_1^r \cdot \delta^r$.

The range-reduced $\log_2$ function lies on the critical path of the proposed LNU and an efficient implementation is key to the performance. We have considered three alternative methods:

- a basic first order polynomial interpolation.
- multi-partite tables generated by the framework presented in the Arenaire project [11].
- the HOTBM method generated by the framework presented in the FloPoCo project [12].

The resulting AT analysis depicted in Fig. 4 clearly shows that for the desired accuracy (23 fractional bits + 4 guard bits), 1st and 2nd-order HOTBM methods are more efficient. We have chosen to use a 2nd-order HOTBM implementation due to its lower area requirements.

Note that the cotransformation block can also be used to implement F2I typecasts and LOG operations by reusing the $\log_2$ block and the pre-shift leading zero counter and shifter.

D. Postprocessing Block

The postprocessing block is used to combine and/or select the results of the two main interpolation blocks. For example SUB operations in the critical region as seen in Eq. 10 requires the output of the cotransformation block ($\log_2(\text{cotrans}(r))$) and the main interpolation block ($F^+(r)$) to be combined. A final rounding step to the output precision and special case handling such as NaN, over- and underflow detection is also performed within this block.

VI. Results

In this section we first present an analysis of the precision of the implemented LNU. Then the area and timing costs are presented for a combinational implementation of the LNU. Finally the performance of a single-core processor with an integrated LNU is compared to a processor using a traditional FPU.

A. Error Analysis

For verification purposes we have performed exhaustive simulations over the one-dimensional space $r \in (-24.588, 0]$ and calculated error metrics as defined in [3]. This is the relevant domain of the functions $F^{\pm}(r)$ since they are clipped to 0 for $r < -24.588$. Table II compares the results obtained from our implementation to those published in [4] and [5]. As expected from the description in Section III-A, the corresponding relative errors in the FP domain $|e|_{\text{av rel fp}}$ remains below 0.5 for both ADD and SUB operations. On the top half of Fig. 5 maximal and average absolute relative errors in the FP domain are plotted over relevant r values for ADD/SUB operations, while on the bottom half error histograms for both operations are shown.

B. Hardware Implementation

We have implemented the LNU using a 65 nm technology (8-metal UMC65LL). For comparison purposes we have also ported the architecture to a 28 nm technology (8-metal GF28SLP). Figure 6a

TABLE II
Comparison of Error Analysis.

Metric	[4]	[5]	This Work
ADD			
$\|e\|_{\text{max rel log}}$	0.6556	-	0.6662
$\|e\|_{\text{av rel log}}$	0.2563	-	0.2522
$\|e\|_{\text{max rel arith}}$	0.4544	0.4623	0.4618
$\|e\|_{\text{av rel arith}}$	0.1777	0.1745	0.1748
SUB			
$\|e\|_{\text{max rel log}}$	0.7144	-	0.6905
$\|e\|_{\text{av rel log}}$	0.2563	-	0.2521
$\|e\|_{\text{max rel arith}}$	0.4952	0.4987	0.4786
$\|e\|_{\text{av rel arith}}$	0.1776	0.1738	0.1748

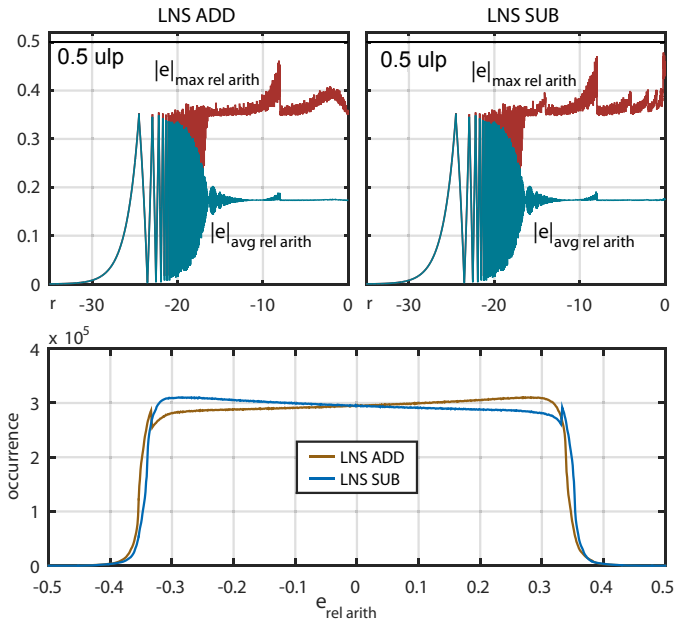

Fig. 5. Relative arithmetic errors of the ADD/SUB operations in the LNU

shows the Area vs Timing (AT) plot obtained from both synthesis and post-layout results using a fully combinational implementation of the LNU under different timing constraints. A comparison in area and timing to similar works published in [4] and [5] is given in Table III. Technology independent gate equivalent (GE) numbers show that our novel implementation is between 20%-35% more area efficient while supporting more functionality such as casts, exponentiations and logarithms. A more detailed analysis showing the area distribution of the blocks with the LNU for different clock constraints is also given in Fig. 6b.

C. Performance Comparison in a Processing System

To evaluate the performance of the presented LNU in a real system, we have designed a single processor system based on a custom 32-bit OpenRISC core [24] and have integrated the LNU into the datapath of the processor core using a 65 nm process. For comparison purposes, an identical system has been designed featuring a standard IEEE 754 single precision compliant FPU architecture with hardware support for additions, subtractions, multiplications and typecasts.

The overall system consisting of the processor, 32 kBytes of

TABLE III
COMPARISON OF AREA AND DELAY

Metric	**[4]**	**[5]**	**This Work**	
Functionality	ADD, SUB	ADD, SUB	ADD, SUB, I2F, F2I, LOG, EXP	
Technology	180 nm	180 nm	65 nm	28 nm
1 GE [μm^2]	9.374	9.374	1.440	0.364
Synthesized				
Delay min[ns]	11.74	7.10	6.00	5.00
Delay max[ns]	13.50	14.79	6.00	5.00
Area [mm^2]	0.906	0.589	0.057	0.016
Area [kGE]	96.6	62.9	40.0	43.5
Routed				
Delay min [ns]	-	6.97	6.00	5.00
Delay max [ns]	-	14.60	6.00	5.00
Area [mm^2]	-	0.584	0.071	0.017
Area [kGE]	-	62.2	49.3	46.6

TABLE IV
COMPARISON OF EXECUTION LATENCY/THROUGHPUT/ENERGY EFFICIENCY OF THE FPU AND LNU AT 1.2 V.

Operation	**FPU**		**LNU**	
	Latency cycles	**Energy** pJ/FLOP	**Latency** cycles	**Energy** pJ/FLOP
I2F/F2I	2	n.a.	4	n.a.
ADD	2	40.7	4	133.2
SUB	2	39.7	4	136.8
MUL	2	47.6	1	29.4
DIV	62*	525.0*	1	30.15
SQRT	56*	609.3*	1	15.85
EXP	51*	566.6*	4	131.671
LOG	85*	695.7*	4	108.471

* Emulation. DIV: range reduction, linear estimate and 3 Newton Raphson iterations. SQRT: fast inverse SQRT and 3 Newton Raphson iterations. EXP/LOG: range reduction and high-order polynomial interpolation [22].

memory, the LNU/FPU and basic peripherals has been designed to operate at 125 MHz at 1.2 V under typical conditions. In order to meet the timing constraints for both architectures, the FPU has been pipelined once, and the LNU three times. The latency of all supported FPU and LNU operations is listed in Table IV. Note that, for the FPU only ADD, SUB, MUL operations and casts are performed in hardware, while software emulations are needed for DIV, SQRT, EXP, and LOG operations. The LNU can perform all operations directly in hardware which leads to significant gains in energy per operation as listed in Table IV. As mentioned earlier in Section II-B some complex operations such as MUL, DIV, and SQRT are greatly simplified in LNS and can therefore implemented directly within the integer ALU of the processor core without major modifications, resulting in significant gains in both latency and energy efficiency.

The pipelined and optimized LNU used in the system occupies 54.3 kGE and is larger than the FPU with a size of only 12 kGE. However, the FPU does not include hardware support for division and our estimations indicate that an FPU that also includes hardware support for divider with a latency of 4 cycles would be around 30 kGE closing the gap significantly. The size of the LNU must also be seen in relation with the rest of the system. In our implementation with a complexity of 331 kGE, the core occupied 42 kGE, peripherals 28 kGE and the 32 kB memory 175 kGE. Thus in our application, the total area increase due to the LNU was less than 13%. We have modified the backend of the OpenRISC LLVM compiler to support the LNS format, and added new instructions to support the additional functionality provided by our LNU core. A set of benchmarks written in C was compiled and executed on the FP and LNU cluster architectures, which have been simulated in Mentor QuestaSim 10.3a using back-annotated postlayout gate-level netlists. Finally, the obtained VCD files were used to analyze the power dissipation in Cadence EDI 14.24. As expected, linear algebra kernels AXPY, GEMM, GEMV that heavily feature ADD/SUB instructions resulted in small performance degradations (less than 25%) as these operations are costly in LNS. However for more complex applications, such as the 3D distance computations, the LNU outperformed the traditional FPU by a factor of 4.35. The floating point instruction ratio and the instruction mix of the benchmark applications is shown in Fig. 7a and the speed up of the LNU vs FPU is shown in Fig. 7b.

VII. CONCLUSIONS

In this paper, we present a novel cotransformation for hardware efficient LNU implementation with equivalent accuracy offered by the IEEE 754 single precision format. We show that the area overhead

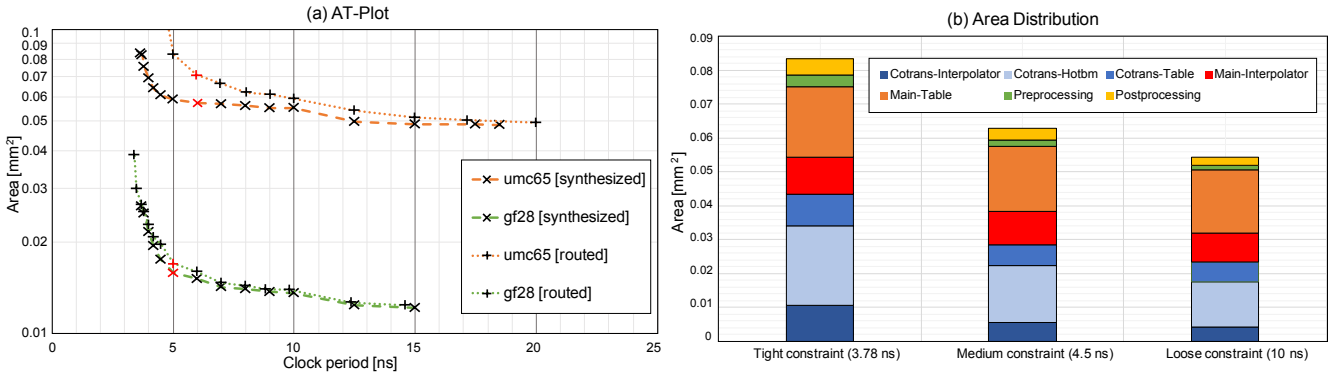

Fig. 6. (a) Area vs Timing plot for the LNU implemented in 28 nm, and 65 nm. (b) Area split of the LNU for tight and loose timing constraints in 65 nm.

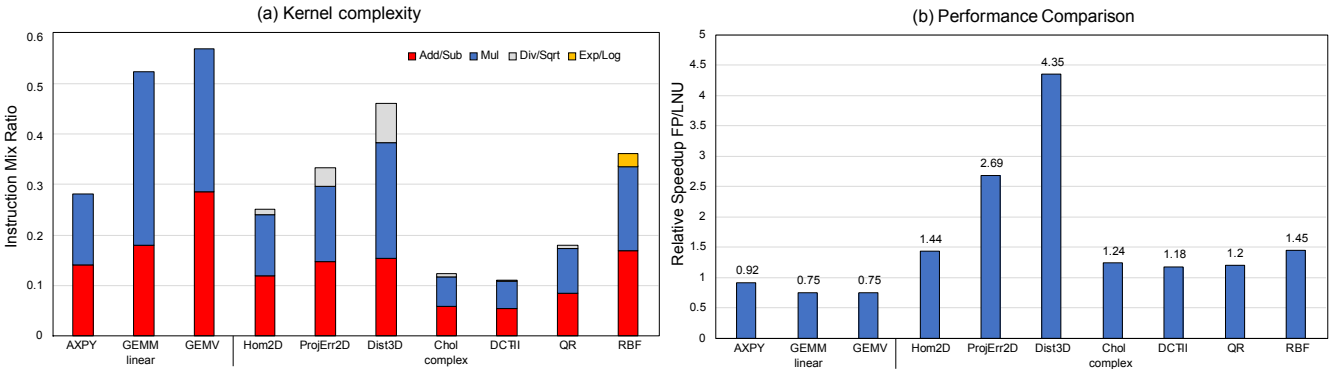

Fig. 7. (a) Kernel characteristics (instruction mix and the total number of FP-instructions). (b) The achieved speedup of the LNU w.r.t. an FPU implementation.

of the LNU is reduced by up to 35% when compared to previous state-of-the-art implementations while supporting more functionality such as casts, logarithms, and exponentiations. Using a 65 nm technology node the final design occupies 54.3 kGE and is able to calculate 89 MFLOP/s and achieve an energy efficiency between 15.9-136.7 pJ/FLOP at typical conditions, 1.2 V supply and 25 °C. We show that when integrated in a single core system the novel LNU can be up to $4.35\times$ faster when compared to an implementation using a standard FPU. Traditionally, LNUs were deemed to be too large to be of practical use. We believe that by reducing the area overhead and bringing additional functionality makes our new LNU architecture a very interesting candidate for computationally intensive applications even for embedded ultra-low-power applications.

Acknowledgments

We thank M. Burger, T. Gautschi, L. Mueller, and F. Schuiki for their commitment during their semester projects. This research was supported by the IcySoC project, evaluated by the Swiss NSF and funded by Nano-Tera.ch with Swiss Confederation financing.

References

[1] M.G. Arnold et al., "Arithmetic co-transformations in the real and complex logarithmic number systems," *IEEE TOC*, vol. 47, no. 7, pp. 777–786, Jul 1998.
[2] J. Coleman, "Simplification of table structure in logarithmic arithmetic," *Electronics Letters*, vol. 31, no. 22, pp. 1905–1906, Oct 1995.
[3] J.N. Coleman et al., "Arithmetic on the European Logarithmic Microprocessor," *IEEE TOC*, vol. 49, no. 7, pp. 702–715, Jul 2000.
[4] ——, "The European Logarithmic Microprocessor," *IEEE TOC*, vol. 57, no. 4, pp. 532–546, April 2008.
[5] R. Ismail and J. Coleman, "ROM-less LNS," in *IEEE ARITH*, July 2011.
[6] R.C. Ismail, et al., "Interpolator algorithms for approximating the LNS addition and subtraction: Design and analysis," in *IEEE ICCAS*, Oct 2012, pp. 174–179.
[7] E. Swartzlander and A. Alexopoulos, "The Sign/Logarithm Number System," *IEEE TOC*, vol. C-24, no. 12, pp. 1238–1242, Dec 1975.
[8] P. Vouzis et al., "LNS Subtraction Using Novel Cotransformation and/or Interpolation," in *IEEE ASAP*, July 2007, pp. 107–114.
[9] *IEEE standard for binary floating-point arithmetic*. New York: Institute of Electrical and Electronics Engineers, 1985, note: Standard 754–1985.
[10] N. Kingsbury and P. Rayner, "Digital filtering using logarithmic arithmetic," *Electronics Letters*, vol. 7, no. 2, pp. 56–58, January 1971.
[11] F. de Dinechin and A. Tisserand, "Multipartite table methods," *IEEE TOC*, vol. 54, no. 3, pp. 319–330, March 2005.
[12] J. Detrey and F. de Dinechin, "Table-based polynomials for fast hardware function evaluation," in *IEEE ASAP*, July 2005, pp. 328–333.
[13] J. Rust et al., "Low Complexity QR-Decomposition Architecture using the Logarithmic Number System," in *IEEE DATE*, March 2013.
[14] J. Garcia et al., "LNS Architectures for Embedded Model Predictive Control Processors," ser. CASES. ACM, 2004, pp. 79–84.
[15] I. Kouretas et al., "Low-Power Logarithmic Number System Addition/Subtraction and Their Impact on Digital Filters," *IEEE TOC*, vol. 62, no. 11, pp. 2196–2209, Nov 2013.
[16] M. Arnold and S. Collange, "A Real/Complex Logarithmic Number System ALU," *IEEE TOC*, vol. 60, no. 2, pp. 202–213, Feb 2011.
[17] M.G. Arnold et al., "Towards a Quaternion Complex Logarithmic Number System," in *IEEE ARITH*, 2011, pp. 33–42.
[18] R.C. Ismail, et al., "Hybrid logarithmic number system arithmetic unit: A review," in *IEEE ICCAS*, Sept 2013, pp. 55–58.
[19] M. Gautschi et. al, "A 65 nm CMOS 6.4-to-29.2 pJ/FLOP@0.8V Shared Logarithmic Floating Point Unit for Acceleration of Nonlinear Function Kernels in a Tightly Coupled Processor Cluster," in *ISSCC*, 2016.
[20] J.-M. Muller, *Elementary Functions*. Springer, 2006.
[21] S. Chevillard et al., "Sollya: An Environment for the Development of Numerical Codes," in *ICMS*, September 2010, pp. 28–31.
[22] J. F. Hart, *Computer Approximations*. Krieger Publishing Co., 1978.
[23] F. de Dinechin et al., "Automatic generation of polynomial-based hardware architectures for function evaluation," in *IEEE ASAP*, July 2010.
[24] M. Gautschi et. al, "Tailoring Instruction-Set Extensions for an Ultra-Low Power Tightly-Coupled Cluster of OpenRISC Cores," in *VLSI-SoC*, 2015, pp. pp25–30.