An Extended Shared Logarithmic Unit for Non-linear Function Kernel Acceleration in a 65 nm CMOS Multi-Core Cluster

Michael Gautschi*, Michael Schaffner*, Frank K. Gürkaynak*, Luca Benini*†
*ETH Zürich, Switzerland, †Università di Bologna, Italy
Email: {gautschi, schaffner, kgf, benini}@iis.ee.ethz.ch
Tel: +41 44 63 {299 58, 265 56, 227 26 , 266 64}

***Abstract*—Energy-efficient and ultra-low power computing are strong requirements for various application areas such as IoT and wearables. While for some applications integer and fixed-point arithmetic suffice, others require a larger dynamic range, typically obtained using floating point (FP) numbers. Logarithmic Number Systems (LNS) have been proposed as energy-efficient alternative, since several complex FP operations translate into simple integer operations. However, additions and subtractions become non-linear operations which have to be approximated via interpolation. Even efficient LNS units (LNUs) are still larger than standard FPUs, rendering them impractical for most general purpose processors. We show that, when shared among several cores, LNUs become a very attractive solution. A series of compact LNUs is developed which provide significantly more functionality (such as transcendental functions) than other state-of-the-art designs. This allows, for example, to evaluate the $\mathrm{atan2}$ function with three instructions for only 183.2 pJ/op at 0.8V. We present the first shared-LNU architecture where these LNUs have been integrated into a multi-core system with four 32b-OpenRISC cores and show measurement results demonstrating that the shared LNU-design can be up to 4.1$\times$ more energy-efficient in common non-linear processing kernels, compared to a similar area design with four private FPUs.

***Index Terms*—Logarithmic Number System (LNS), Shared Floating Point Unit (FPU), Special Function Unit (SFU), Multi-Core, Low-Power Design

I. Introduction

Energy-efficient computing and ultra-low power operation are strong requirements for a vast number of application areas such as IoT and wearables. While for some applications integer and fixed-point processor instructions suffice, several others (eg. classification [1], [2], vision applications like detection [3], SLAM [4] or camera pose estimation [5]) require a larger dynamic range, typically obtained using single-precision floating point (FP) numbers. In addition, new algorithms are usually first developed for general-purpose computing systems (PCs, workstations) assuming high-dynamic range (HDR) arithmetic. Porting these algorithms to integer or fixed-point arithmetic is a labor-intensive and technically challenging task which requires extensive test and verification [6]. Hence, there is a trend toward supporting HDR arithmetic, currently in the form of single-precision FP, also in low-power microcontrollers, such as the ARM Cortex M4 [7]. Unfortunately, it is well known that FP is energy-hungry, and significant research effort is devoted toward reducing the energy required for HDR computing. *Logarithmic Number Systems* (LNS) have been proposed [8–17] as an energy-efficient alternative to conventional FP, since several complex FP operations such as MUL, DIV, and SQRT translate into simple integer operations 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 continues to grow on a regular basis.

However, the drawback of LNS is that additions and subtractions become non-linear operations and, when implemented in hardware, have to be approximated accordingly with a dedicated *LNS unit* (LNU). The area of LNUs grows exponentially with the desired precision, and for an accuracy equivalent to single-precision FP, LNUs are larger than traditional *floating-point units* (FPUs), which makes it difficult to motivate their use in general purpose processors. We show that in multi-core systems optimized for ultra-low-power operation such as the PULP system [18], one LNU can be efficiently shared in a cluster. This arrangement not only reduces the per-core area overhead, but more importantly allows several operations such as MUL/DIV to be processed within the integer cores without contention and additional overhead. Based on several application benchmarks, we show that in a system with one LNU shared among four cores, access contentions are minimal as in most algorithms the percentage of ADD/SUB operations remains below 30%.

In this work we introduce a series of compact 32b LNS units which have similar area compared with the best designs in literature [15–17], while at the same time providing significantly more functionality. We enhance the LNU architecture to implement transcendental functions with only small area overhead similar to *special function units* (SFUs) present in today's GPUs [19]. Therefore, we not only support standard LNS ADD/SUB operations, but also fused multiply/divide-add operations (FMA, FDA) and the non-linear function intrinsics 2^x, $\log_2(x), \sin(x), \cos(x)$ and $\mathrm{atan2}(y, x)$, which are useful for many embedded applications ranging from computer vision [5] to power electronics [20].

For accurate comparison, we designed four chips with a quad-core cluster system using the UMC 65 nm LL CMOS technology. Each chip contains an identical cluster with four 32b OpenRISC cores. The first chip is based on four private single-precision FPUs, and the remaining three chips share differently parameterized LNUs. Three of these chips (the FP, LNS A and LNS C, see Section V) have been taped out and measured. Using these designs, we show that for typical non-linear processing tasks, our LNS design can be up to 4.1$\times$ more energy-efficient than a private-FP design and achieve similar energy efficiency when running pure linear algebra kernels with many ADD/SUB operations. Further, the use of a 16b vector LNU is investigated, which can be an interesting design alternative for applications that require HDR and can tolerate lower precision.

II. Related Work

The LNS has been proposed as a replacement for standard fixed-point and FP arithmetic already in the 1970's [8], [9]. The main challenge of finding efficient approximation methods to implement the non-linear ADD/SUB operations has driven research in the LNS domain. In early papers, implementation of LNUs with accuracy higher than 12b was considered infeasible due to exponentially increasing lookup-table (LUT) sizes. Since then, several improved implementations have been proposed. In the low-precision FP calculation domain, with bit-widths lower than 16 bits, so-called multi-partite table [21] and high-order table based methods [22] have been shown to be effective [23]. LNS based operations have been used to replace fixed-point operations in several applications such as QR decomposition [24], non-linear SVM kernel evaluation [1], embedded model predictive control [25], neural network accelerators [26] and low power digital filtering [27]. LNS numbers have also been extended to be used for complex numbers [28] and quaternions [29]. Attempts to combine both the advantages of standard FP and LNS into hybrid systems have been made in [30], where the main drawback is the cost of non-linear typecasts.

Coleman, et al. [12] introduced the concept of a *cotransformation* to alleviate approximation difficulties related to SUB operations where the operand difference is close to 0.0. As explained in Section III-B, such cotransformations are analytical decompositions of the problematic functions allowing to implement the same functionality with significantly smaller LUTs. Following the example of Coleman, et al., several different cotransformation variations have been presented in [10], [11], [13–17], [31]. Complete LNUs for ASIC processors with accuracy equivalent to single-precision FP have only been presented in [15–17] so far. Coleman, et al. [15] describe the *European Logarithmic Microprocessor* (ELM), the first single-core microprocessor featuring an LNU. Their design combines a custom interpolation scheme with the cotransformation developed by [12]. In [16], [17] Coleman, et al. improve their ELM design and propose LNUs with lookup tables small enough to be implemented without ROMs. These LNU designs are able to execute only basic LNS ADD/SUB instructions and do not have support for casts.

FPUs for standard FP are designed in [32–35] and often only contain support for ADD, SUB, MUL, casts and FMA operations. Support for divisions is then added in form of SW emulations or iterative HW dividers since single-cycle HW divisions are expensive [33]. On top of the basic algebraic operations, there is a growing need to support HDR computations of common non-linear functions. A significant body of work [19], [36–40] studies the efficient implementation of special function units (SFUs) for GPUs which implement non-linear functions such as cosine, sine, arctangent, square-roots, etc. in FP. Compared to complete numerical function libraries (e.g., as part of $\mathrm{math.h}$ in C++) these *intrinsics* are much faster (a few cycles instead of hundreds), but they do not provide the same accuracy level. Also, the intrinsics usually evaluate the special functions on reduced ranges (e.g., $[0, \pi/2)$), and have to be wrapped with range reduction routines [41].

In contrast to above listed work, we combine both the LNU and the SFU into one unit, since they share many architectural properties. Based on the PULP system previously developed in [18], we design the first multi-processor with a shared LNU, and show that this can be an energy-efficient alternative to a standard FP design for various non-linear function kernels. PULP is an OpenRISC multi-processor platform without FP support, and we used it as an architectural template. Further, this paper extends our previous work on LNS [31], [42], [43]. While [31], [42] focus on a first LNU implementation without all special function (SF) extensions, [43] explains the fitting framework in more detail and explores differently parameterized LNUs with lower precision than 32b. The presented work represents the culmination of these efforts and provides multiple silicon implementations and measurements, more comparisons, and an improved LNU architecture with SF-extensions, and a 16b vector LNU.

![thb]

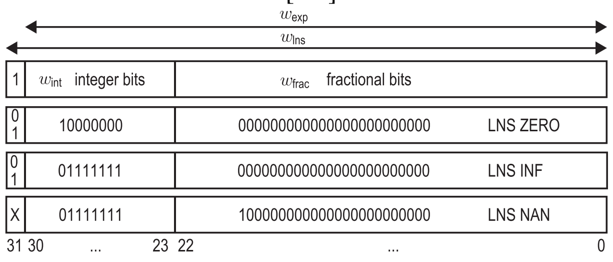


Figure 1: Encoding and special values of the 32b LNS numbers used in this work.

III. Preliminaries

A. LNS Number Representation, Format and Arithmetic Operations

Standard FP number systems represent a real number as $a = (-1)^s \cdot m_{\text{frac}} \cdot 2^{l_{\text{exp}}}$, where s is the sign, m_{frac} the mantissa and l_{exp} the exponent. In LNS, numbers are only represented by an exponent l_{exp} which now has a fractional part: $a = (-1)^s \cdot 2^{l_{\text{exp}}}$. In this case, the exponent is an unbiased two's complement number and its width is denoted as $w_{\text{exp}} = w_{\text{int}} + w_{\text{frac}}$, where w_{int} and w_{frac} are the number of integer and fractional bits, respectively. The bit-width of the complete number including the sign bit is $w_{\text{lns}} = w_{\text{exp}} + 1$. For $w_{\text{int}} = 8$ and $w_{\text{frac}} = 23$, the encoding is aligned with the single-precision FP (IEEE 754) format [46], and for $w_{\text{int}} = 5$ and $w_{\text{frac}} = 10$, it is equivalent to the half-precision format. Similar to the IEEE 754 standard, special values such as zeros (ZERO), infinities (INF) and not a number (NAN) are encoded using special bit patterns (Figure 1). In the following, we will use lowercase variables for real numbers (e.g. a). The corresponding LNS sign and exponent are denoted as s_a, l_a and the machine representation as $A = [s_a, l_a]$.

Certain operations such as MUL, DIV, and SQRT can be implemented very efficiently using the LNS format with a single integer addition, subtraction or bitshift, respectively. This is an important advantage because such operations can be efficiently calculated in slightly enhanced integer ALUs and result in much shorter latencies and energy costs than the equivalent FP implementations. However, these simplifications come at the cost of more complex additions and subtractions which become non-linear operations in LNS:

$$a \pm b = z, l_z = \max(l_a, l_b) + \log_2(1 \pm 2^{-|l_a - l_b|}). \qquad (1)$$

Using the absolute difference $r = |l_a - l_b|$, the two non-linear functions for addition and subtraction can be defined as $\mathrm{F}^{+}(r) = \log_2(1 + 2^{-r})$ and $\mathrm{F}^{-}(r) = -\log_2(1 - 2^{-r})$ shown in Figure 2.

As discussed in more detail in [13], [43], an LNS design with $w_{\text{int}} = 8$ and $w_{\text{frac}} = 23$ should have a maximum relative error of 0.7213 ulp in the LNS domain to be equivalent to FP with round to nearest mode (0.5 ulp). In the following, all error figures are given w.r.t. the FP domain.

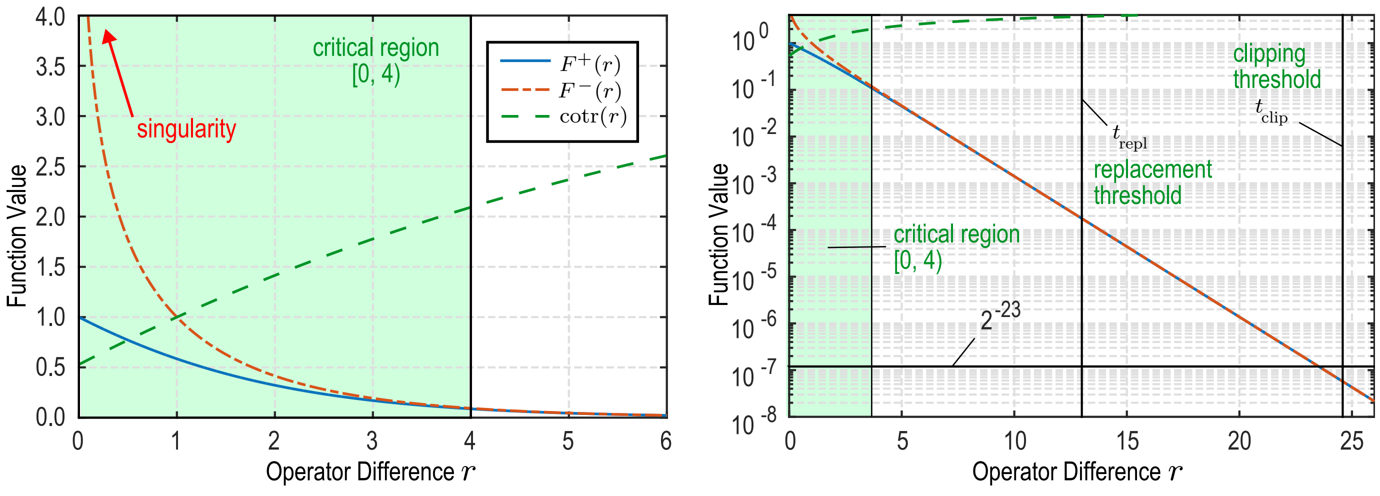


Figure 2: Plot of the $F^+(r)$, $F^-(r)$ and $\mathrm{cotr}(r)$ functions in linear and semi-logarithmic domain - note the singularity for $r \to 0$. The critical region is $[0,\ 4)$, providing the best trade-off in terms of the overall number of polynomial segments [44], [45]. $F^-(r)$ can be replaced by $F^+(r)$ for values of r above t_{repl}, and both functions can be clipped to 0.0 above t_{clip} for $w_{\text{frac}} = 23$.

B. Cotransformations and Fitting Framework

While for low precision (<12 bit) implementations $\mathrm{F}^{\pm}(r)$ can be stored in LUTs, this approach is not feasible anymore for single-precision FP. Piecewise polynomial approximations however, have been found to work well [44], [45] to reach these precision levels – except for operations where r is small since $\mathrm{F}^-(\mathrm{r})$ has a singularity at zero. In this *critical region* (CR), so-called *cotransformations* [10–12], [14–17] are usually applied to decompose $\mathrm{F}^-(\mathrm{r})$ into sub-functions which can be approximated more efficiently. The cotransformation used in this work was originally proposed by [10] and decomposes $\mathrm{F}^-(r)$ as

$$\begin{aligned}\mathrm{F}^-(r) &= -\log_2(1-2^{-r}) \\ &= -\log_2\left(\frac{1-2^{-r}}{r}\right) + \log_2(r) \\ &= \mathrm{cotr}(r) + \log_2(r).\end{aligned} \tag{2}$$

It has been successfully used to create compact LNS operators for FPGAs [44], [45] – but so far its usage for implementing an ASIC LNU design has not been closely investigated. This decomposition leverages the fact that $\mathrm{cotr}(r)$ behaves much better around 0.0 as shown in Figure 2, and can thus be readily approximated using polynomials. The size of the CR is set to $[0,\ 4)$ which provides the best trade-off for this decomposition in terms of overall number of polynomial segments [44], [45]. The $\log_2(r)$ function still has a singularity at 0, but for finite precision arithmetic this function can be efficiently implemented with range reduction techniques of the argument [41], where the argument range can be reduced to $[1,\ 2)$ by employing a leading zero counter and a barrel shifter. The $\log_2$ function itself can be implemented on this reduced range using a polynomial.

For the accuracy range between half-precision and single-precision FP numbers, piecewise 1st and 2nd order polynomials have been found to be efficient approximators for a wide range of non-linear functions [19], [36–40]. This also holds for the $\mathrm{F}^+(\mathrm{r}), \mathrm{F}^-(\mathrm{r})$ functions (outside the CR) [44], [45], and hence our LNU architecture has been designed using such polynomials. The employed fitting framework is described in more detail in [43] and uses a finite-precision aware implementation [47] of the Remez algorithm [48] in combination with an interval splitting heuristic [49] to compute so-called piecewise *minimax* polynomials.

IV. LNU Architecture and Extensions

The proposed LNU architecture shown in Figure 3 uses five main datapath units depending on the operation, and whether the operation falls into the CR as explained below.

A. Main LNU Blocks

The *MulDiv Block* preprocesses fused multiply/divide-add (FMA/FDA) functions and also enables arbitrary base exponentials $C = \exp(A \cdot B)$ and $C = \exp(A/B)$, termed MEX and DEX. In addition, the division capability allows for convenient range reduction of the trigonometric functions (e.g., division by π for sine/cosine). The intermediate results U and V from the *MulDiv Block* are used in the *AddSub Preprocessing Block* which calculates the absolute operator difference $r = |l_u - l_v|$ and the operator maximum for binary operations such as ADD/SUB. For unary operations such as EXP/LOG, operator V is gated to zero.

The *Main Interpolator Block* implements $\mathrm{F}^+(r)$ on the complete range $[0,\ t_{clip})$ and $\mathrm{F}^-(\mathrm{r})$ outside the CR $[4,\ t_{clip})$ using piecewise polynomial approximations. Depending on the latency-area trade-off the 1st or 2nd order approximation will be used. This block is also used for SUB operations in the CR $[0,\ 4)$ to evaluate $\mathrm{cotr}(r)$, the result of which is later added to the $\log_2(r)$ value in the *Postprocessing Block*. For a given input r, the coefficients $p_i^r = p_i(r)$ for $i = \{0, ..., N\}$ (where N is the polynomial order) are selected from a set of LUTs, and the polynomial is evaluated using the Horner scheme

$$p(r) = p_0^r + \delta_p^r \cdot \left(\dots \left(p_{(N-1)}^r + \delta_p^r \cdot (p_N^r) \right) \right) \tag{3}$$

where δ_p^r are the LSBs of r. The main interpolator datapath can be shared among $\mathrm{F}^+(r)$, $\mathrm{F}^-(r)$ and $\mathrm{cotr}(r)$. As explained in more detail in [43], each LUT is subdivided into different segments, each of which contains a set of equidistantly spaced coefficient samples. The segment boundaries have been aligned to powers of two, such that the segment index can be easily determined by looking at the MSBs of the argument r. The functions $\mathrm{F}^+(r)$ and $\mathrm{F}^-(r)$ become increasingly

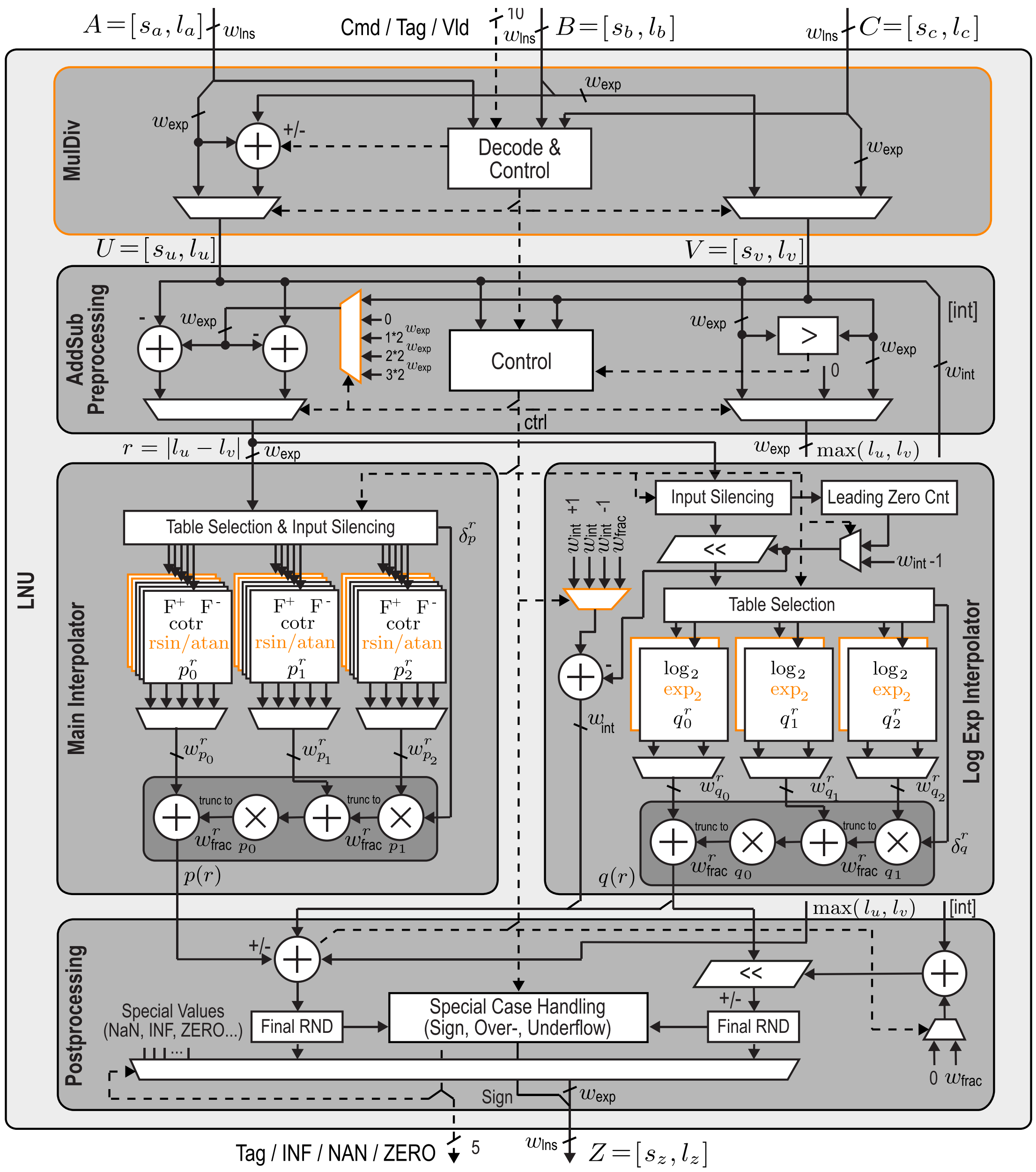


Figure 3: The LNU architecture, shown for 2nd order interpolation. Parts for SF-extensions are highlighted.

similar with increasing r such that one function can be replaced by the other without impact on precision. Therefore, we define a threshold t_{repl} and reuse the $\mathrm{F}^+(r)$ tables for $\mathrm{F}^-(r)$ when $r > t_{repl}$ (single-precision: $t_{repl} = 14$). Further, for large r, the functions values of $\mathrm{F}^+(r)$ and $\mathrm{F}^-(r)$ fall below the required precision due to their asymptotic behaviour and can be clipped to 0. This clipping threshold is denoted as t_{clip} (single-precision: $t_{clip} = 24.588$).

The main objective of the *Log/Exp Block* is to implement the $\log_2(r)$ function within the critical region $[0, 4)$ required for SUB operations. This is achieved using a barrel shifter and leading zero counter to reduce the range of the input, and an Nth order interpolator with LUTs covering the argument range $[1, 2)$. Note that it is possible to reuse this function for native typecasts from integer to LNS (I2F), and LOG2 operations in the LNS domain. For a given input r, the polynomial coefficients $q_i^r = q_i(r)$ for $i = \{0, ..., N\}$ are selected from a set of LUTs, and the approximation result $q(r)$ is again calculated as in Equation (3).

To natively support inverse typecasts (F2I) and 2^x (EXP) operations, an additional table for the 2^x function has been added. Since this function can also be efficiently implemented using range reduction and polynomial interpolation, we can reuse the existing interpolator to calculate the function value on the range $[0, 1)$, and only have to include an additional shifter at the output which has been moved to the *Postprocessing Block* such that the delay for the ADD/SUB operations is not increased. This final block combines or selects the interpolators, performs rounding and special case handling such as NAN and over/underflow detection.

B. Trigonometric Functions

Trigonometric functions can be approximated in several ways [39], [40], [50], e.g., with the well-known class of CORDIC algorithms

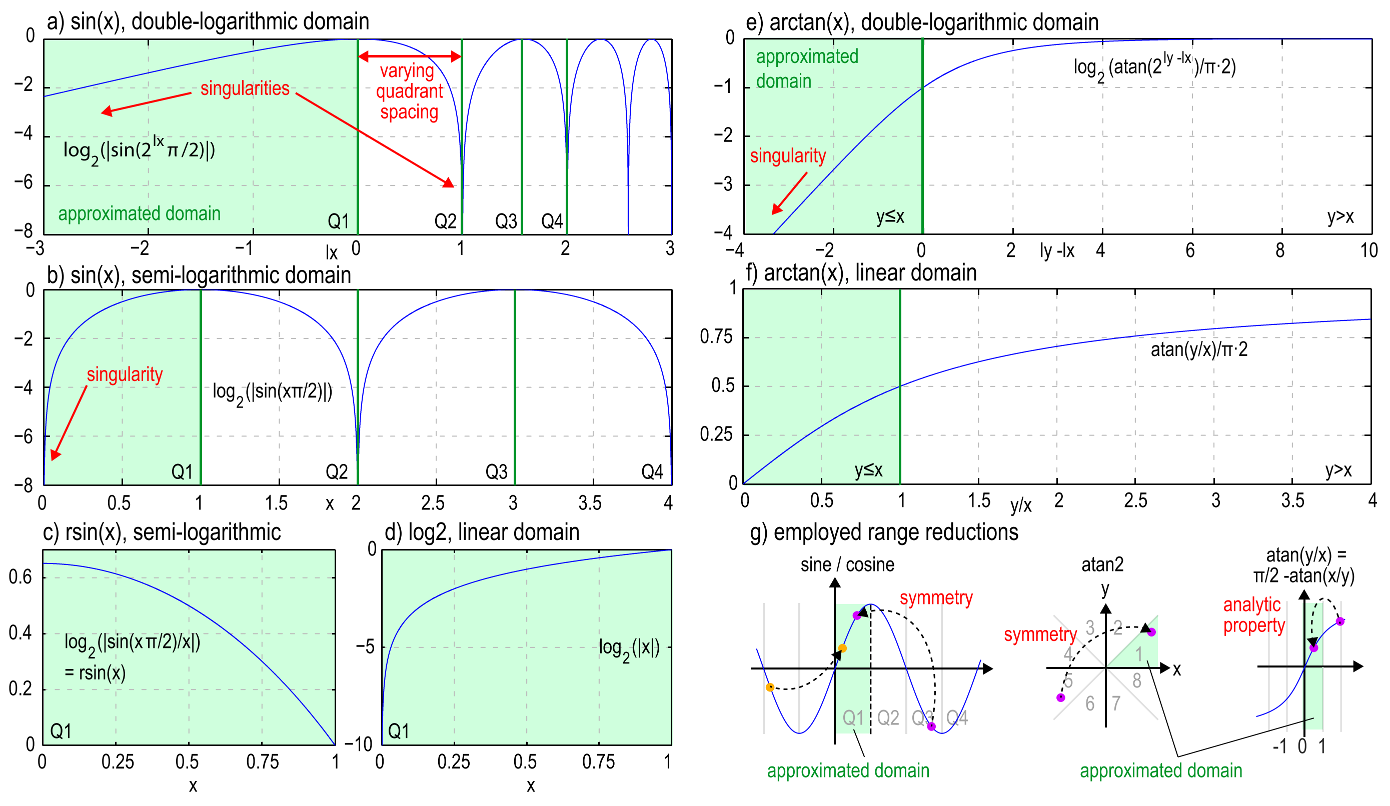


Figure 4: The first quadrants (Q1-4) of the sine function plotted in the LNS domain (a) and in the semi-logarithmic domain (b). The sine function is analytically split into (c) and (d) which can be implemented more efficiently using the LNU architecture. e) shows the arctangent function in the LNS domain, and f) in the linear domain. g) shows examples for employed range reductions.

[51]. These iterative shift-and-add methods can even be implemented on processors with limited DSP functionality. Lower-latency implementations however, use table-based methods in combination with range reduction techniques [41]. In fact, several SFUs employ 2nd order interpolation [36–38]. Hence, our LNU can be extended with trigonometric functions by only modifying a few existing components.

We restrict the set of extensions to sine, cosine and arctangent functions, since these are the most commonly used and many other functions can be efficiently derived from these using trigonometric relations. These functions are implemented using normalized angles which have the advantage of simpler modulo-2 calculations for range reductions (see [39], [40] for more details). We implement the extensions for the functions $\sin(\frac{\pi}{2}x)$, $\cos(\frac{\pi}{2}x)$, $\frac{2}{\pi}\operatorname{atan2}(y,x)$ which already implicitly contain the factor $\frac{\pi}{2}$.

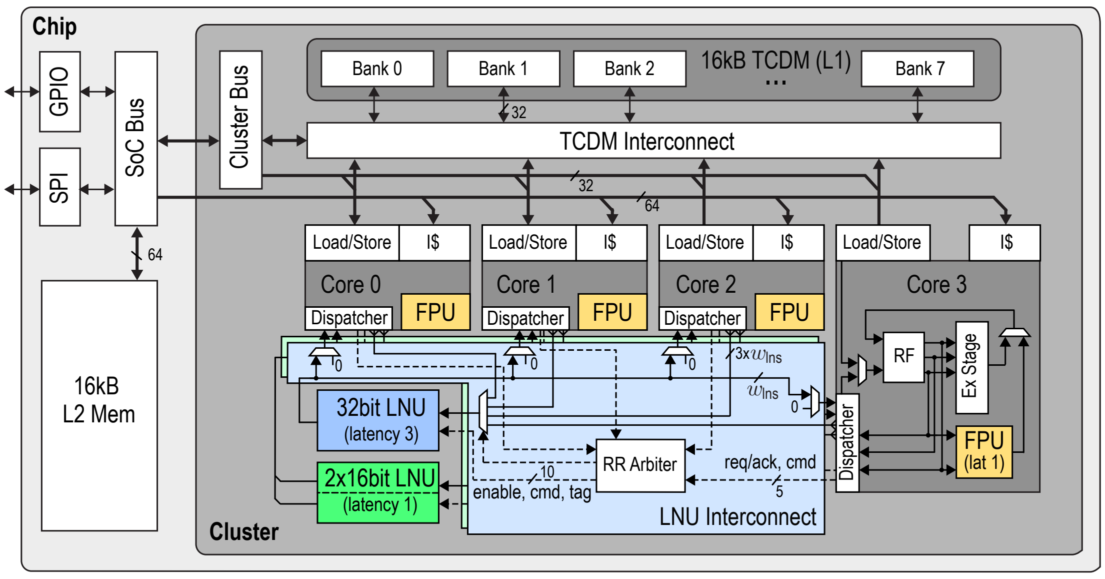
While LNS has several benefits such as low-latency DIV/MUL, it also introduces two difficulties for the particular case of trigonometric functions. First, 0.0 equals to $-\inf$ in LNS and leads to singularities as illustrated in Figure 4a and e, where the magnitude of the $\sin(\frac{\pi}{2}x)$ and $\frac{2}{\pi}\operatorname{atan}(x)$ functions are plotted in the LNS domain. Second, the logarithmic spacing complicates fast range reduction and folding techniques for periodic functions since the quadrants are not evenly spaced. These issues are addressed below for the trigonometric SF-extensions. Note that these SF-extensions do not require any additional special case handling and range reductions in software, since this is automatically performed in HW in the LNU.

IV-B1 Sine/Cosine: First, the argument range is reduced via an LNS division in the *MulDiv Block*, followed by a transformation into the linear domain by reusing the EXP functionality. The modulo-1 operation (rem) and symmetric folding can then be performed by truncating the MSBs and using integer additions, making it possible to implement the sine/cosine functions by tabulation of the first sine quadrant. Second, the evaluation of this first sine quadrant is analytically split into two terms in order to circumvent the singularity in a similar way as using a cotransformation (Section III-B). As shown in Figure 4b, the sine function $\log_2(|\sin(\frac{\pi}{2}x)|)$ still has a singularity in the semi-logarithmic domain. To implement it we use the decomposition

$$\begin{aligned}\log_2(|\sin(\frac{\pi}{2}x)|) &= \log_2\left(\left|\frac{\sin(\frac{\pi}{2}x)}{x}\right|\right) + \log_2(|x|) \\ &= \operatorname{rsin}(x) + \log_2(|x|),\end{aligned} \tag{4}$$

since the first term (Figure 4c) can be efficiently approximated with a 2nd order polynomial, and the second term (Figure 4d) is already available as part of the *LogExp Block* in the LNU. To maximize datapath reuse these two steps are mapped onto different instructions termed SCA (sine/cosine argument), SIN and COS (the actual interpolation) which have to be issued sequentially (e.g., SIN(SCA(A,B)) with A=X, B= $\frac{\pi}{2}$). The SCA(A,B) is a DEX variant leveraging a cheap LNS division to divide the sine or cosine argument by $\frac{\pi}{2}$.

IV-B2 Arc Tangent: The choice of the approximation scheme depends on whether only the single argument version $\operatorname{atan}(x)$ or the two-argument version $\operatorname{atan2}(y,x)$ is implemented. We target the latter version in this work, such that no additional software wrapping of the arctangent function is required to calculate the correct phase. As shown in [40], the $\operatorname{atan2}(y,x)$ function can be implemented by



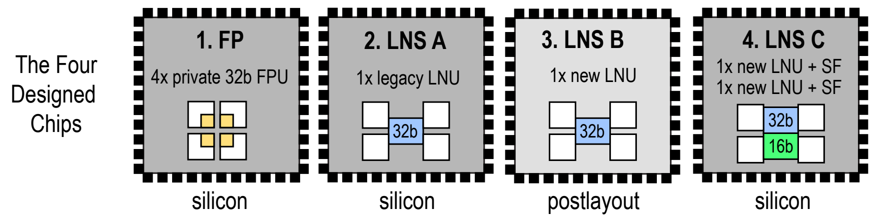


Figure 5: Integration of the shared 32b LNU and the 16b vector LNU into an OpenRISC cluster. For clarity connection details of the LNU are shown only for core 3.

only tabulating the $\mathrm{atan}(y/x)$ function on $y/x \in [0,1]$ (i.e., on the first cartesian octant when interpreting x and y as coordinate values). Since $\mathrm{atan2}$ is an odd function, the inputs can be reduced to the positive quadrant, and using the relation $\mathrm{atan}\,(y/x) = \pi/2 - \mathrm{atan}\,(x/y)$ this can be further reduced to the first octant.

To address the LNS singularity issue, the existing EXP and LOG functionality of the LNU is reused to approximate the arctangent in the linear domain (Figure 4f). The evaluation of one $\mathrm{atan2}$ intrinsic is therefore split in three LNU instructions ATA (arctan argument), ATN (arctangent table lookup) and ATL (arctangent logarithm), and the corresponding evaluation sequence is ATL(ATN(ATA(A,B))), where A=Y and B=X.

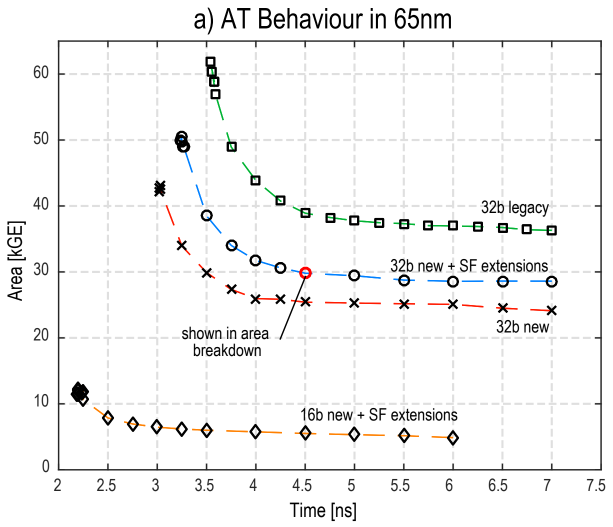
The ATA(A,B) instruction is a variant of the DEX instruction. The difference is that before actually applying the EXP operation, the control logic detects cases where the result $u = y/x > 1$ and inverts the value l_u such that the result of the ATA instruction is always within [0,1]. To mirror the output of the arctangent to the correct octant, the information is encoded into the result of the ATA operation. The ATN instruction performs the actual arctangent interpolation, and mirrors the result to the correct octant via the constant multiplexor in the *LogExp Block*. The ATL instruction is a LOG variant which just bypasses negative signs in this case.

V. Processor Integration

To evaluate the performance of different LNUs in a shared setting, we have designed a multi-core processor system based on a 32b OpenRISC core [52] using the UMC 65 nm LL technology. As shown in Figure 5, the system consists of four cores with 1 KB private instruction caches which share a single LNU and contains a total of 32 kBytes of memory. The four-to-one sharing ratio is motivated by the fact that in most FP programs, the fraction of ADD/SUB instructions, rarely exceeds 0.25 (see Figure 8a for examples). For comparison purposes, an identical system has been designed featuring 4 cores with private IEEE 754 32b compliant FPUs including HW support for ADD, SUB, MUL and typecasts. For DIV we use SW emulations as described in Section VI-D which is a common approach of adding FP support to small embedded processors [33]. The implemented FPU is a shared normalizer design similar to [32] (but without divider) with a complexity of 10 kGE – which is competitive with state-of-the-art implementations [33], [34], [53].

To show the evolution of the LNUs we present four chips (Figure 5), where the last chip is extended with a vectorized half-precision LNU allowing to perform two half-precision LNU operations in parallel as well as providing dot-product (DOTP) instructions that utilize MulDiv blocks of both half-precision LNUs. All clusters have been designed to run at 500 MHz at 1.2V under typical case conditions. To meet the timing constraints of the cluster, the FPU and the 16b vector LNU have been pipelined once and the 32b LNUs three times.

Note that the tool flow has evolved over the project and the newest versions are more efficient w.r.t. the first silicon implementations of the FP and LNS A designs [42]. Therefore, the backends have been re-peated using the current design flow to compensate for any systematic offsets in the measurements from [42]. The cotransformation has also evolved over the project, and while all new designs (LNS B, C) use



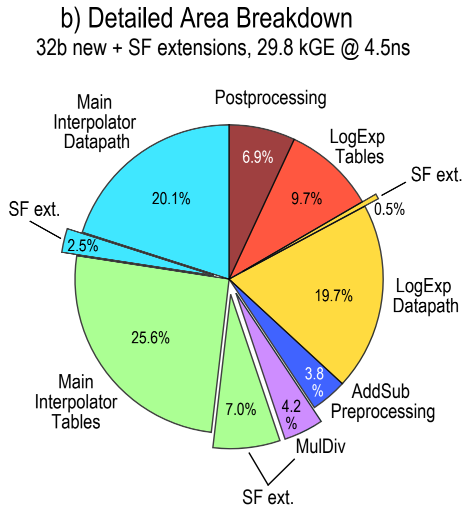


Figure 6: a) Post-synthesis AT behaviours of the LNUs analyzed in this work (combinational, unpipelined entities), and b) detailed area breakdown of the new LNU with SF-extensions (overheads due to these extensions are exploded in the pie chart).

[10], the first silicon implementation (LNS A) used the one described in detail in [31], [42]. Thus, the main difference between the legacy LNU and the new LNU designs is the way how subtractions in the CR are handled. The new LNU designs have a smaller interpolator and require about 45% less LUT bits than the legacy LNU design (as can be seen in Table I).

A. Modifications to the Processor Core

The LNU is shared in a completely transparent way, the programmer sees a system with as many LNUs as there are cores. A dispatcher that is tightly integrated in the execution stage of each core is responsible to offload LNU instructions and stall the cores if necessary, and silence the operator ports in case no instruction has to be offloaded. Our OpenRISC architecture contains two write ports to the register file, which are used by the ALU and the load-store-unit (LSU). Instead of adding a third write port for the LNU, the LSU-port has been shared with the dispatcher since a single-issue in-order pipeline cannot execute multiple LNU and load operations concurrently. However, due to contentions in the LNU interconnect and LSU, write back collisions can occur, which are handled by a small controller which stalls the pipeline if necessary. The integer ALUs of the cores have been slightly modified to handle the LNS sign bit in single, and half-precision format correctly. Since the OpenRISC core has already been extended to support packed SIMD integer operations [52], the sliced adder and shifters were reused to support vectorized LNS MUL/DIV/SQRT operations. In LNS C, where the system is enhanced with a 16b vector LNU, a second interconnect is used to alleviate contention. Both interconnects consist of a fair round-robin arbiter for core-request handling. Whenever several cores access the LNU concurrently, all cores but one have to stall their pipelines and wait for an LNU idle cycle.

VI. Results

For hardware and energy efficiency evaluations, we have pipelined the LNU designs using the automatic retiming feature of Synopsys Design Compiler 2016.03, and integrated them into a four-core OpenRISC cluster as described in Section V-A. For each cluster we have performed a complete back-end design flow using Cadence Innovus-15.20.100 in the same 8-metal UMC 65 nm LL CMOS technology. The back-end of the OpenRISC LLVM compiler has been modified to support the LNS format, and new instructions have been added to support the additional functionality provided by our LNUs. SF intrinsics and vector instructions are provided as compiler built-in functions (compiler auto-vectorization is currently not supported). A set of benchmarks written in C was compiled and executed on the FP and LNS architectures. The FP, LNS A and LNS C chips have been manufactured, extensively tested and measured. The remaining LNS B cluster has been simulated in Mentor QuestaSim 10.3a using back-annotated post-layout gate-level netlists and power has been analyzed in Cadence Innovus-15.20.100. First, a comparison of our LNU designs with related LNUs and SFUs is given in Section VI-A, followed by a comparison of the designed chips in Section VI-B. Finally, we will give detailed results of instruction- and kernel-level efficiencies in Section VI-C and Section VI-D.

A. LNU Results and Comparison with Related Work

The AT behavior of the LNUs, and a detailed area split for an LNU with SF-extensions is shown in Figure 6a and b. A large part of the LNU area is occupied by LUTs (42.3%) and interpolators (42.8%). Table I shows a comparison of synthesis results for combinational implementations in both 65 nm and 180 nm. Our new LNU without SF-extensions is significantly smaller in terms of normalized gate-equivalents (GE) than most related designs, and provides more functionality such as casts and EXP/LOG functions. The additional FMX/TRIG instructions increase the area by only 17.3%. Compared to the best related 65 nm design *Minimax (2CT)* by [17] our new LNU has similar area, but a higher combinational delay. Note however, the only silicon-proven related design is the ELM LNU [15], [16] and *Minimax (2CT)* was first been developed in 180 nm, and the 65 nm results listed in Table I represent additional technology translations provided in [17]. The related designs [15–17] either use custom 1st order schemes, or 2nd order polynomial interpolators with a dedicated squarer. These schemes use 1.8-5.9× more LUT entries than our design, but have the advantage of providing lower latency compared to our 2nd order

TABLE I: Comparison of different LNUs and related designs (synthesis results of combinational, unpipelined entities).

	[15] ELM	[16] Mod. Chester (2CT)	[17] Minimax (2CT)	This Work Legacy	This Work New	This Work New + SF	This Work New + SF
Functionality§	ADD, SUB			ADD, SUB, I2F, F2I, LOG, EXP		ADD, SUB, I2F, F2I, LOG, EXP, FMX, TRIG	
Interpolation							
Wordwidth [bit]	32	32	32	32	32	32	16
cotr Type	[13]	[16]	[16]	[31]	[10]	[10]	[10]
cotr Interp. (N)	custom (1)	custom (1)	minimax (2)	custom (1)	minimax (2)	minimax (2)	minimax (1)
Other Functions (N)	custom (1)	custom (1)	minimax (2)	minimax (2)	minimax (2)	minimax (2)	minimax (1)
Error [ulp]							
ADD (max)	0.4544	0.4623	0.4944	0.4618	0.3806	0.3806	0.3753
ADD (avg)	0.1777	0.1745	0.1721	0.1748	0.1744	0.1744	0.1734
SUB (max)	0.4952	0.4987	0.4626	0.4786	0.4755	0.4755	0.4561
SUB (avg)	0.1776	0.1738	0.1719	0.1748	0.1750	0.1750	0.1738
LUT Size [kBit]							
F^+	227.3	162.0	?	20.3	30.4	30.4	2.8
F^-				41.3	12.2	12.2	0.6
cotr	129.0	21.2	?	24.0	4.7	4.7	1.0
other	-	-	-	27.5	13.2	23.7	2.6
total	356.4	183.3	110.1	113.1	60.9	71.5	7.0
180 nm Results							
Technology	180 nm	UMC 180 nm		UMC 180 nm			
1 GE [µm^2]	9.3744‡	9.3744		9.3744			
Delay min/max [ns]	11.74/13.50	7.1/14.79	9.3/~19.2	17.0/17.0	12.5/12.5	12.5/12.5	8.5/8.5
Area [mm^2]	0.906	0.589	0.474	0.411	0.301	0.375	0.076
Area [kGE]	96.6	62.9	50.6	43.8	32.1	40.0	8.1
Used in Silicon	yes (ELM)	no	no	no	no	no	no
65 nm Results							
Technology	-	UMC 65 nm†		UMC 65 nm			
1 GE [µm^2]	-	1.44		1.440			
Delay min/max [ns]	-	0.91/1.94	1.24/2.60	6.0/6.0	4.5/4.5	4.5/4.5	3.0/3.0
Area [mm^2]	-	0.039	0.031	0.054	0.037	0.043	0.009
Area [kGE]	-	26.8	21.8	37.5	25.4	29.8	6.5
Used in Silicon	-	no	no	yes (LNS A)	no	yes (LNS C)	yes (LNS C)

§ FMX stands for all variations FMA, FMS, FDA, FDS, DEX and MEX. TRIG stands for the instructions SCA, SIN, COS, ATA, ATN, ATL. † 65 nm numbers from [17].
‡ assumed NAND2 area (UMC 180 nm).

TABLE II: Comparison of the new LNU with SF extensions and FP SFUs (post-layout figures, i.e. pipelined, placed&routed units).

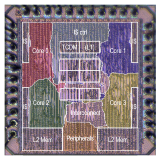
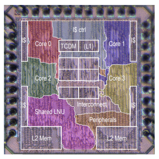
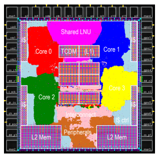
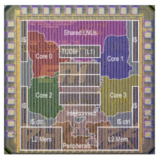
	NVidia [19], [36]				Caro et al. [37]				This Work (New LNU + SF)			
Function	Table Interval	Error×	LUT [kBit]	Lat/ TPut	Table Interval	Error×	LUT [kBit]	Lat/ TPut	Table Interval	Error×	LUT [kBit]	Lat/ TPut
$\sqrt{x}$	-	-	-	-	[1,2]	1.0 ulp	1.73	6/1	-	0.69 ulp	-	1/1§
$\sqrt{2x}$	-	-	-	-	[1,2]	1.0 ulp	1.82	6/1	-	0.69 ulp	-	1/0.5§
$1/\sqrt{x}$	[1,4]	1.52 ulp	6.5	?/1	[1,2]	1.0 ulp	3.71	6/1	-	0.69 ulp	-	1/0.5§
$1/\sqrt{2x}$	-	-	-	-	[1,2]	1.0 ulp	3.65	6/1	-	0.69 ulp	-	1/0.33§
$1/x$	[1,2]	0.98 ulp	6.5	?/1	[1,2]	1.0 ulp	4.29	6/1	-	0 ulp	-	1/1§
$\log_2(x)$	[1,2]	22.57 bit	3.25	?/1	[1,2]	24 bit	4.03	6/1	[1,4]	0.75 ulp	9.15	4/1
2^x	[0,1]	1.41 ulp	3.25	?/1	[0,1]	1.0 ulp	1.98	6/1	[0,1]	23.0 bit	4.48	4/1
$\sin(x), \cos(x)$	[0,π/2]	1.72e-7	3.25	?/1	-	-	-	-	[0,1]	1.85e-7	6.54	8/0.5
$\mathrm{atan2}(y,x)$	-	-	-	-	-	-	-	-	[0,1]	1.87e-7	4.03	12/0.33
Other Support												
Functions	2D Attribute Interpolation				no				ADD, SUB, I2F, F2I, FMA			
NaN, INF	?				no				yes			
Implementation												
Technology	?				TSMC 180 nm				UMC 65 nm			
1 GE [µm^2]	?				9.374†				1.44			
Freq. [MHz]	?				420				305			
Area [mm^2]	?				0.34				0.067			
Area [kGE]	8030 FE $\approx$ 42.2 kGE‡				36.3				45.6			

† Assumed NAND2 area (UMC 180 nm). ‡ Assuming 5.25 GE per FE. § Computed in integer ALU of the processor cores. × Error either specified in ulp or as absolute maximum error if not applicable. For $\log_2(x)$ (in FP) and 2^x (in LNS) precision given in amount of good bits returned, see [36], [37].

Horner interpolator. However, these designs exhibit variable latency, since CR subtractions have to pass through the interpolator twice, making it harder to share such an LNU.

A comparison of the new LNU with SF-extensions (post-layout) with an SFU from Tesla GPUs [19], [36] and from Caro et al. [37] is shown in Table II. Our design is comparable in complexity while providing equivalent accuracy levels and functionality. Note that an LNS-based processor does not require any additional units to be complete, whereas an FP-based system additionally requires an FPU for standard operations. When comparing the LUT sizes, we can observe that the design by Caro et al. requires the least amount of bits for special functions. The reason for this is that they additionally enforce constraints among neighboring LUT segments when fitting

TABLE III: Comparison of LNU-based and FPU-based Processor Chips.

		ELM [15]	 1. FP	 2. LNS A	 3. LNS B	 4. LNS C	
Results From		Silicon	Silicon	Silicon	Post-layout	Silicon	
Technology		180 nm	65 nm	65 nm	65 nm TT 25°C	65 nm	
Area [kGE]		-	676	702	682	725	
Area 1 Core [kGE]		-	39.6	41.3	41.4	43.4	
FPU/LNU Area [kGE]		-	4×9.6	57.1	37.7	69.2	
Supply Voltage [V]		-	0.8 - 1.3	0.8 - 1.3	1.2	0.8 - 1.3	
Max. Freq. @1.2V [MHz]		125	374	337	500	305	
Power@100MHz, 1.2V [mW]		-	23.2	24.4	22.2	24.5	
Avg FPU/LNU Utilization		-	0.22	0.39	0.39	0.42	
LNU/FPU Type		ELM LNU	FPU	Legacy LNU	New LNU	New LNU + SF	
Wordwidth [bit]		32	32	32	32	32	2×16
Max Error [ulp]		0.454	0.5	0.478	0.476	0.476	0.456
Latencies							
ADD/SUB/FMA	hw	3/3(4)[†]/-	2/2/-	4/4/-	4/4/4	4/4/4	2/2/2
	sw	-/-/-	-/-/-	-/-/-	-/-/-	-/-/-	-/-/-
MUL/DIV/SQRT	hw	1/1/1	2/-/-	1/1/1	1/1/1	1/1/1	1/1/1
	sw	-/-/-	-/62/56	-/-/-	-/-/-	-/-/-	-/-/-
EXP/LOG/casts	hw	-/-/-	-/-/-	4/4/4	4/4/4	4/4/4	2/2/2
	sw	-/-	51/85	-/-/-	-/-/-	-/-/-	-/-/-
$\sin / \cos / \mathrm{atan2}$	hw	-/-/-	-/-/-	-/-/-	-/-/-	8/8/12	4/4/6
	sw	-/-/-	69[‡]/68[‡]/340	64/60/92	64/60/92	-/-/-	-/-/-

[†] Variable latency LNU. [‡] Without range-reducing division (phase magnitude $\geq 2\pi$).

TABLE IV: Measured and Simulated Energy Consumption of Single Operations.

Design	**Horowitz et al. [34]**		**Intel [53]**	**1. FP**	**2. LNS A**	**3. LNS B**	**4. LNS C**	
Results From	Estimations		Silicon	Silicon[§]	Silicon[§]	Post-layout[§]	Silicon[§]	
Technology	90 nm	45 nm	32 nm	65 nm	65 nm	65 nm TT 25°C	65 nm	
Vdd [V]	1.08	0.9	1.05	0.8	0.8	0.8	0.8	
Frequency [MHz]	1200	2080	1450	165	115	150	97	
FP Width [bit]	32	32	32	32	32	32	32	2×16
Pipeline Depth	10	6	3	2	4	4	4	2
Area [mm^2]	0.113	0.016	0.045	0.015	0.089	0.054	0.066	0.017
[pJ/FLOP]								
ADD	-	-	-	16.1	55.0	40.1	58.1	50.3
SUB	-	-	-	16.2	55.1	43.7	63.4	63.9
MUL	-	-	-	18.4	12.5	12.9	13.5	16.0
DIV	-	-	-	-	12.7	13.0	14.3	16.8
SQRT	-	-	-	-	5.7	5.7	6.2	7.8
EXP	-	-	-	-	42.7	30.7	45.3	36.9
LOG	-	-	-	-	47.6	30.5	42.6	33.8
FMA[†]	55.4	14.4	38.8	-	-	-	57.3	43.6
MEX	-	-	-	-	-	-	57.4	50.6
$\sin(x)$	-	-	-	-	-	-	127.4	92.0
$\cos(x)$	-	-	-	-	-	-	126.9	95.6
$\mathrm{atan2}(\mathrm{y}, \mathrm{x})$	-	-	-	-	-	-	183.2	152.1

[†] One 1 FMA = 1 FLOP in this comparison. [§] Background power (NOP) subtracted, includes core activity (estimated core overheads are on average 2.6 pJ/FLOP for single-operand instructions and 4.2 pJ/FLOP for all other instructions).

the non-linear functions. This allows to obtain even smaller LUTs than with the minimax fitting heuristics employed in [19], [36] and our design. However, the design by Caro et al. does not have support for LNS ADD/SUB, trigonometric functions and the special cases (like NAN/INF). Note that the complexity of the design by [19], [36] is specified in full-adder equivalents (FE) of a proprietary library and has been converted assuming 5.25 GE per FE. Also, the trigonometric intrinsics for sine/cosine do not have complete range reduction from arbitrary values to the first function period. Our design in contrast is able to perform automatic range reduction on all trigonometric functions thanks to cheap LNS divisions.

B. Designed Chip Variants

Table III lists all designs and gives a comparison of the measured and simulated chip variants. Our reference design is the FP chip which includes one private FPU per core (four total). LNS A is the first chip which includes a shared legacy LNU. The LNS B design includes a new LNU as well as optimizations in the multi-core system. The LNS C design includes a new LNU with SF-extensions and a 2×16b vector LNU, allowing to calculate kernels more efficiently at lower precision. In terms of related work, there exist many application-specific implementations in the literature (e.g., [5], [24–27], [38]) which use LNS and its benefits to compute faster and/or more energy-efficiently. However, there are surprisingly few designs where an LNU

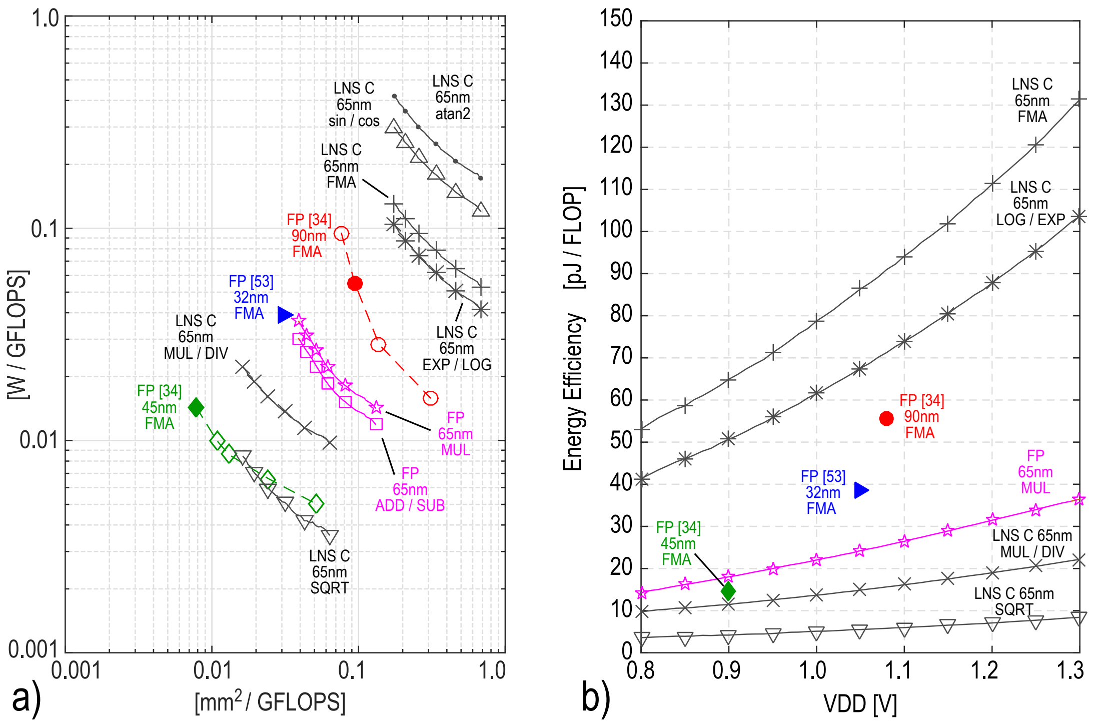


Figure 7: Measured efficiency results: a) W/GFLOPS versus mm^2/GFLOPS and b) pJ/FLOP versus supply voltage. The parameterized plots of our designs in a) have been obtained by sweeping the supply voltage over the range 0.8-1.3 V. Curves for related designs [34] also represent differently pipelined designs. The LNS MUL/DIV/SQRT operations in a) have been normalized with the ALU area of one processor core (0.0061 mm^2 or 4.3 kGE). The processor overheads mentioned in Table IV have been subtracted from all shown results.

equivalent to single-precision FP is designed and integrated into a programmable processor. In fact, the only comparable ASIC design where this has been done is the ELM [15], which is also listed in Table III.

C. Instruction Level Performance

Figure 7a shows an efficiency trade-off analysis similar to the one conducted in [34], enhanced with datapoints from related FPU designs [34], [53]. Figure 7b shows the energy-efficiencies over a range of different VDD conditions for a selection of operations. A complete comparison of the operator efficiencies of all designs is given in Table IV. While LNS ADD/SUB are clearly less energy-efficient than the FP equivalents, LNS MUL requires ~30% less energy than in FP for all LNS designs. Apart from these basic instructions, LNS supports extremely energy-efficient, single-cycle square-roots (6.2 pJ/op @0.8V) and divisions (14.3 pJ/op @0.8V) utilizing the shifter and adder of the processor ALUs. Also complex functions such as 2^x and $\log_2(x)$ can be computed in the LNU for only 45.3 pJ/op and 42.6 pJ/op, respectively (@0.8V). The architectural improvements of the LNU result in 27.2 % more efficient ADD/SUB instructions when comparing the LNS A design with LNS B. The trigonometric function extensions of LNS C cause a small increase in LUT size, but at the same time enable completely range-reduced sine/cosine and arctangent function instrinsics with 2-3 LNU instructions and a low energy consumption of 127.4-183.2 pJ/op @0.8V. In addition, the LNS C design supports 16b instructions which have a 1.6-2.8$\times$ lower energy consumption w.r.t. to their 32b equivalents.

D. Function Kernel Performance

To analyze the performance of the shared LNU in the multi-core cluster, a representative set of benchmark kernels (Table V) reflecting different signal processing applications has been compiled. The set ranges from linear algebra operations, geometry calculations, matrix decompositions and regression to image and audio processing kernels. The FP instruction ratio and the instruction mix of the benchmarks are shown in Figure 8a and we observe that the ratio of ADD/SUB operations is below 30% for most benchmarks, which further reinforces our sharing concept. It should be noted that a second, *shared* FPU design with an overall complexity of 651 kGE has also been evaluated but not included in this comparison as it was much slower (up to 46%) due to many contentions in the FPU interconnect (up to 96% of accesses resulted in stalls).

The reference FPU is compact but does not include support for more complex operations which have to be emulated in SW. For DIV operations, we perform a range reduction to [1,2) and generate a linear estimate for the inverse that is refined using three Newton-Raphson iterations. A similar technique is used for the SQRT, where the initial estimate is generated using the fast-inverse square-root approach. EXP/LOG operations and trigonometric functions combine range reduction with a standard high-order interpolation technique [41]. A detailed listing of the HW and SW instruction latencies can be found in Table III.

Figure 8d shows the energy efficiency gains of the analyzed LNS clusters w.r.t. the FPU cluster. For complex algorithms with many multiplications, divisions and non-linear functions all LNS designs outperform the FPU up to 4.1$\times$ in terms of speedup and energy

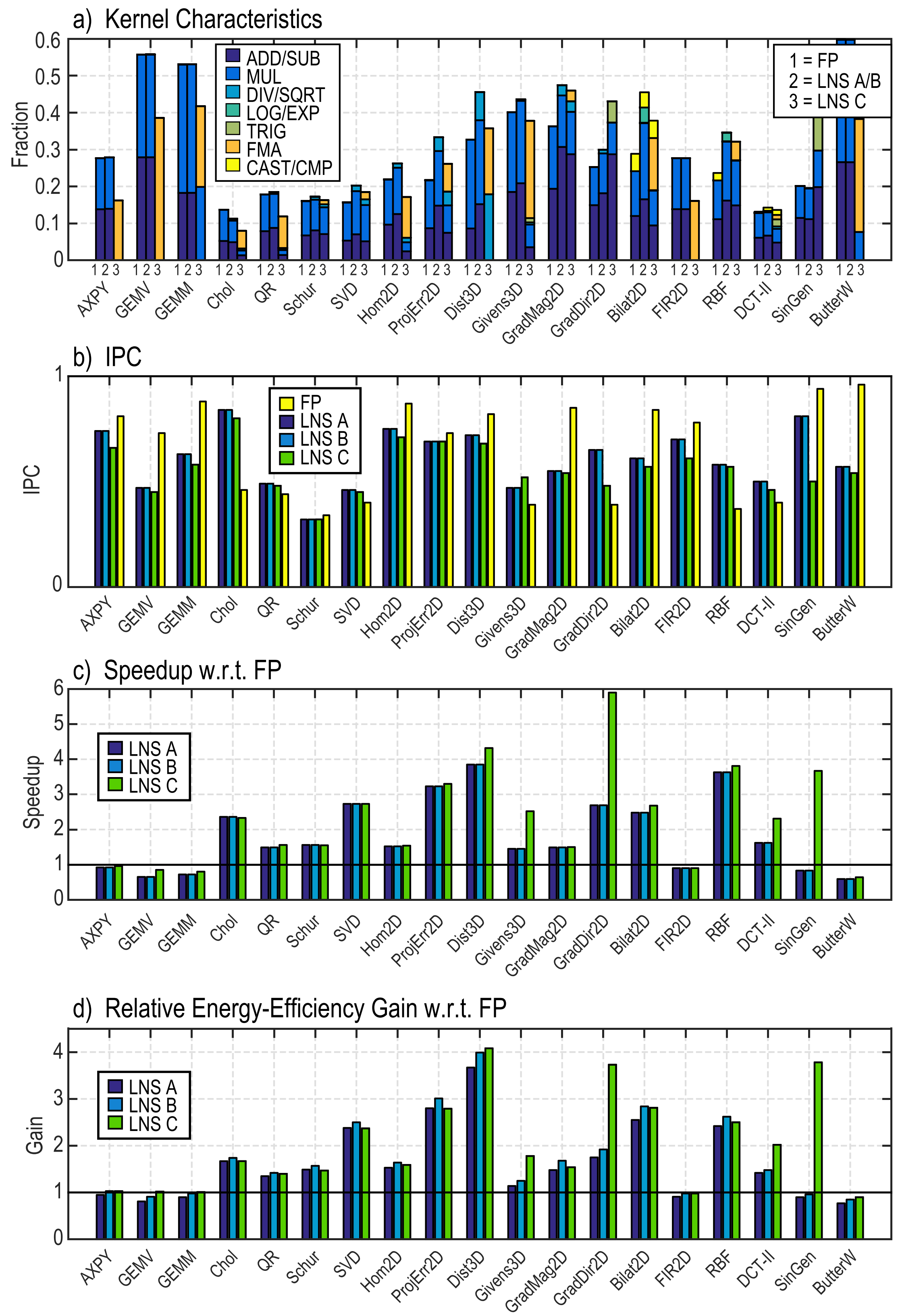


Figure 8: Application kernel results based on silicon measurements (FP, LNS A, LNS C) and post-layout simulations (LNS B): Kernel characteristics in terms of instruction mix (a) and IPC (b). Speedup (c) and energy efficiency (d) gains w.r.t. the FP design.

TABLE V: Description of Function Kernels in the Benchmark.

Kernel	Details
AXPY	BLAS 1 Kernel, [100$\times$1] vectors
GEMV	BLAS 2 Kernel, [10$\times$10] matrices
GEMM	BLAS 3 Kernel, [10$\times$10] matrices
Chol	Cholesky Decomposition, [10$\times$10] matrices
QR	QR Decomposition, [10$\times$10] matrices
Schur	Schur Decomposition, [5$\times$5] matrices
SVD	Singular value decomposition, [5$\times$5] matrices
Hom2D	Evaluation of 2D homographies (projective transforms) [5], [54]
ProjErr2D	Calculation of reprojection error of 2D projective transforms [5], [54]
Dist3D	Distance computations in 3D
Givens3D	Calculation of 3D givens matrix and rotation of 10 vectors
GradMag2D	Computation of gradient magnitude in 2D [3]
GradDir2D	Computation of gradient direction (angle) in 2D [3]
Bilat2D	Evaluation of a bilateral filter in 2D
FIR2D	Evaluation of separable 5$\times$5 Blur Filter in 2D
RBF	Evaluation of a 2D regression function with 25 Gaussian kernels
DCT-II	Evaluation of 1D 32-point DCTs
SinGen	1kHz sine generator (audio)
ButterW	6th order (3 second order sections) Butterworth IIR lowpass filter (audio)

efficiency (Figure 8c and d). The LNS C design which supports energy-efficient intrinsic SIN/COS instructions even exhibits speedups in the order of 2.3-5.9$\times$ for kernels with trigonometric functions. DCT-II for example can take advantage of fast SIN/COS evaluations while the FP design has to call expensive software emulations which take 68 cycles. For ADD/SUB intensive benchmarks like GEMM, GEMV and ButterW, the FP design is 10% more energy-efficient than LNS C with FMA extensions. In case of ButterW, for example, this drop is caused by data-dependencies in the second order sections of the filter, leading to an increased amount of stalls due to the LNS ADD latency. Note that LNS multiplications can be handled very efficiently in the processor cores with single cycle integer additions. Therefore, the use of FMA instructions for LNS does not improve efficiency the same way as in FP designs as can be seen in Figure 8c, e.g., for the AXPY, GEMV and ButterW kernels. The utilization of the shared LNU on our benchmarks was 0.42 on average with a maximum of 0.65, leading to an average of 3% stalls due to access contentions (14% in the worst case). For applications where half-precision is acceptable, speedup gains can be increased by 2-2.4$\times$ w.r.t. the 32b LNS C design, since twice as many operations can be executed per cycle and less stalls occur due to the shorter latencies.

VII. Conclusions

We have presented the first multi-core chips with support for complex, but energy-efficient HDR operations based on LNS. We designed a series of compact 32b LNUs which provide significantly more functionality than other state-of-the-art designs. Useful transcendental functions (2^x, $\log_2(x)$, $\sin(x)$, $\cos(x)$, $\operatorname{atan2}(y,x)$) can be conveniently added to the LNU incurring a small overhead of 17.3%, since most resources such as the interpolators can be reused. Extending the preprocessing stage of the LNU with an adder allows to support fused operations such as multiply-add. While the area cost of a single LNU is difficult to amortize in a single-core system, we show that a *shared* LNU can be competitive in size with a traditional private FPU design. Due to the fact that MUL, DIV, SQRT can be efficiently computed in the integer units of the cores, it is sufficient to share one LNU among a cluster of four cores. Despite the fact that additions are more complex and energy-consuming in LNS, we show that the 32b shared LNS designs can compute typical non-linear function kernels up to 4.1$\times$ more energy-efficiently than an equivalent chip with four private FPUs. This gain can be attributed to the low-latency MUL, DIV and SQRT instructions and special function extensions. A vectorized, 2$\times$16b half-precision LNU also represents an interesting design choice as it allows additional speedup gains of 2-2.4$\times$ for additional 24 kGE complexity for applications where reduced precision is tolerable. Given the current trend to incorporate more and more pre-processing steps into small embedded systems, support for HDR arithmetic becomes ever more important – not only as enabler but also for convenience and rapid application development. Especially for applications involving the evaluation of complex non-linear kernels, we think that shared-LNU architectures are a strong contender w.r.t to standard FP implementations.

Acknowledgments

This research was supported by the FP7 ERC Advance project MULTITHERMAN (g.a. 291125) and the IcySoC project, evaluated by the Swiss NSF and funded by Nano-Tera.ch with Swiss Confederation financing.

References

[1] F.M. Khan and M.G. Arnold and W.M. Pottenger, "Hardware-Based Support Vector Machine Classification in Logarithmic Number Systems," in *IEEE ISCAS*, 2005, pp. 5154–5157 Vol. 5.

[2] S. Afifi, H. Gholam Hosseini, and R. Sinha, *Image and Video Technology – PSIVT 2015 Workshops. Revised Selected Papers*. Springer International Publishing, 2016, ch. Hardware Acceleration of SVM-Based Classifier for Melanoma Images, pp. 235–245.

[3] M. Komorkiewicz, M. Kluczewski, and M. Gorgon, "Floating Point HOG Implementation for Real-Time Multiple Object Detection," in *FPL*, 2012, pp. 711–714.

[4] C. Kerl, J. Sturm, and D. Cremers, "Dense Visual SLAM for RGB-D Cameras," in *IEEE/RSJ IROS*, Nov 2013, pp. 2100–2106.

[5] I. Hong, G. Kim, Y. Kim, D. Kim, B. G. Nam, and H. J. Yoo, "A 27 mW Reconfigurable Marker-Less Logarithmic Camera Pose Estimation Engine for Mobile Augmented Reality Processor," *IEEE JSSC*, vol. 50, no. 11, pp. 2513–2523, Nov 2015.

[6] G. Frantz and R. Simar, "Comparing Fixed- and Floating-Point DSPs," http://www.ti.com.cn/cn/lit/wp/spry061/spry061.pdf, 2004, Texas Instruments, Dallas, TX, USA, Accessed: November 2016.

[7] "Cortex-M4 Processor," http://www.arm.com/products/processors/cortex-m/cortex-m4-processor.php, ARM Ltd., Cambridge, UK, Accessed: November 2016.

[8] N. Kingsbury and P. Rayner, "Digital Filtering Using Logarithmic Arithmetic," *Electronics Letters*, vol. 7, no. 2, pp. 56–58, January 1971.

[9] E.E. Swartzlander and A.G. Alexopoulos, "The Sign/Logarithm Number System," *IEEE TOC*, vol. C-24, no. 12, pp. 1238–1242, Dec 1975.

[10] V. Paliouras and T. Stouraitis, "A Novel Algorithm for Accurate Logarithmic Number System Subtraction," in *IEEE ISCAS*, 1996, pp. 268–271.

[11] M. G. Arnold, T. A. Bailey, J. R. Cowles, and M. D. Winkel, "Arithmetic Co-transformations in the Real and Aomplex Logarithmic Number Systems," *IEEE TOC*, vol. 47, no. 7, pp. 777–786, 1998.

[12] J. N. Coleman, "Simplification of Table Structure in Logarithmic Arithmetic," *Electronics Letters*, vol. 31, no. 22, pp. 1905–1906, Oct 1995.

[13] J. N. Coleman, E. I. Chester, C. I. Softley, and J. Kadlec, "Arithmetic on the European Logarithmic Microprocessor," *IEEE TOC*, vol. 49, no. 7, pp. 702–715, 2000.

[14] P. Vouzis, S. Collange, and M. Arnold, "LNS Subtraction Using Novel Cotransformation and/or Interpolation," in *IEEE ASAP*, 2007, pp. 107–114.

[15] 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 TOC*, vol. 57, no. 4, pp. 532–546, 2008.

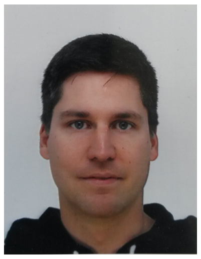
[16] R. C. Ismail and J. N. Coleman, "ROM-less LNS," in *IEEE ARITH*, 2011, pp. 43–51.

[17] J. N. Coleman and R. C. Ismail, "LNS with Co-Transformation Competes with Floating-Point," *IEEE TOC*, vol. 65, no. 1, pp. 136–146, Jan 2016.

[18] D. Rossi, A. Pullini, I. Loi, M. Gautschi, F. K. Grkaynak, A. Bartolini, P. Flatresse, and L. Benini, "A 60 GOPS/W, 1.8 V to 0.9 V Body Bias ULP Cluster in 28nm UTBB FD-SOI Technology ," *Solid-State Electronics*, vol. 117, pp. 170 – 184, 2016.

[19] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, "NVIDIA Tesla: A Unified Graphics and Computing Architecture," *IEEE Micro*, vol. 28, no. 2, pp. 39–55, March 2008.

[20] A. Ukil, V. H. Shah, and B. Deck, “Fast Computation of Arctangent Functions for Embedded Applications: A Comparative Analysis,” in *IEEE ISIE*, June 2011, pp. 1206–1211.
[21] F. de Dinechin and A. Tisserand, “Multipartite Table Methods,” *IEEE TOC*, vol. 54, no. 3, pp. 319–330, March 2005.
[22] J. Detrey and F. de Dinechin, “Table-based Polynomials for Fast Hardware Function Evaluation,” in *IEEE ASAP*, July 2005, pp. 328–333.
[23] ——, “A Tool for Unbiased Comparison Between Logarithmic and Floating-Point Arithmetic,” *J VLSI SIG PROC SYST*, vol. 49, no. 1, pp. 161–175, 2007.
[24] J. Rust, F. Ludwig, and S. Paul, “Low Complexity QR-Decomposition Architecture using the Logarithmic Number System,” in *DATE*, 2013, pp. 97–102.
[25] J. Garcia, M. G. Arnold, L. Bleris, and M. V. Kothare, “LNS Architectures for Embedded Model Predictive Control Processors,” in *ACM CASES*, 2004, pp. 79–84.
[26] D. Miyashita, E. H. Lee, and B. Murmann, “Convolutional Neural Networks using Logarithmic Data Representation,” *ArXiv*, March 2016.
[27] I. Kouretas, C. Basetas, and V. Paliouras, “Low-Power Logarithmic Number System Addition/Subtraction and Their Impact on Digital Filters,” *IEEE TOC*, vol. 62, no. 11, pp. 2196–2209, 2013.
[28] M.G. Arnold and S. Collange, “A Real/Complex Logarithmic Number System ALU,” *IEEE TOC*, vol. 60, no. 2, pp. 202–213, Feb 2011.
[29] M. G. Arnold, J. Cowles, V. Paliouras, and I. Kouretas, “Towards a Quaternion Complex Logarithmic Number System,” in *IEEE ARITH*, 2011, pp. 33–42.
[30] R. Ismail, M. Zakaria, and S. Murad, “Hybrid Logarithmic Number System Arithmetic Unit: A Review,” in *IEEE ICCAS*, 2013, pp. 55–58.
[31] Y. Popoff, F. Scheidegger, M. Schaffner, M. Gautschi, F. K. Gürkaynak, and L. Benini, “High-Efficiency Logarithmic Number Unit Design Based on an Improved Co-Transformation Scheme,” in *DATE*, 2016.
[32] T.-J. Kwon, J. Sondeen, and J. Draper, “Design Trade-Offs in Floating-Point Unit Implementation for Embedded and Processing-in-Memory Systems,” in *IEEE ISCAS*, 2005, pp. 3331–3334.
[33] K. Karuri, R. Leupers, G. Ascheid, H. Meyr, and M. Kedia, “Design and Implementation of a Modular and Portable IEEE 754 Compliant Floating-Point Unit,” in *DATE*, vol. 2, 2006, pp. 1–6.
[34] S. Galal and M. Horowitz, “Energy-Efficient Floating-Point Unit Design,” *IEEE TOC*, vol. 60, no. 7, pp. 913–922, July 2011.
[35] S. Galal, O. Shacham, J. S. B. II, J. Pu, A. Vassiliev, and M. Horowitz, “Fpu generator for design space exploration,” in *Computer Arithmetic (ARITH), 2013 21st IEEE Symposium on*, April 2013, pp. 25–34.
[36] S. F. Oberman and M. Y. Siu, “A High-Performance Area-Efficient Multifunction Interpolator,” in *IEEE ARITH*, June 2005, pp. 272–279.
[37] D. D. Caro, N. Petra, and A. G. M. Strollo, “High-Performance Special Function Unit for Programmable 3-D Graphics Processors,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 56, no. 9, pp. 1968–1978, Sept 2009.
[38] B. G. Nam, H. Kim, and H. J. Yoo, “A Low-Power Unified Arithmetic Unit for Programmable Handheld 3-D Graphics Systems,” *IEEE JSSC*, vol. 42, no. 8, pp. 1767–1778, Aug 2007.
[39] F. de Dinechin, M. Istoan, and G. Sergent, “Fixed-point Trigonometric Functions on FPGAs,” *SIGARCH Comput. Archit. News*, vol. 41, no. 5, pp. 83–88, Jun. 2014.
[40] F. de Dinechin and M. Istoan, “Hardware Implementations of Fixed-Point Atan2,” in *IEEE ARITH*, June 2015, pp. 34–41.
[41] J. F. Hart, *Computer Approximations*. Krieger Publishing Co., 1978.
[42] M. Gautschi, M. Schaffner, F. K. Gürkaynak, and L. Benini, “A 65nm CMOS 6.4-to-29.2pJ/FLOP@0.8V Shared Logarithmic Floating Point Unit for Acceleration of Nonlinear Function Kernels in a Tightly Coupled Processor Cluster,” in *IEEE ISSCC*, 2016.
[43] M. Schaffner, M. Gautschi, F. K. Gürkaynak, and L. Benini, “Accuracy and Performance Trade-offs of Logarithmic Number Units in Multi-Core Clusters,” in *IEEE ARITH*, 2016.
[44] H. Fu, O. Mencer, and W. Luk, “Optimizing Logarithmic Arithmetic on FPGAs,” in *IEEE FCCM*, 2007, pp. 163–172.
[45] ——, “FPGA Designs With Optimized Logarithmic Arithmetic,” *IEEE TOC*, vol. 7, no. 59, pp. 1000–1006, 2010.
[46] J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres, *Handbook of Floating-Point Arithmetic*. Birkhäuser Boston, 2010.
[47] S. Chevillard, M. Joldeş, and C. Lauter, “Sollya: An Environment for the Development of Numerical Codes,” in *ICMS*. Springer, 2010, pp. 28–31.
[48] J.-M. Muller, *Elementary Functions*. Springer, 2006.
[49] F. De Dinechin, M. Joldes, and B. Pasca, “Automatic Generation of Polynomial-Based Hardware Architectures for Function Evaluation,” in *IEEE ASAP*, 2010, pp. 216–222.
[50] R. Gutierrez, V. Torres, and J. Valls, “FPGA-Implementation of Atan(Y/X) Based on Logarithmic Transformation and LUT-based Techniques,” *J. Syst. Archit.*, vol. 56, no. 11, pp. 588–596, Nov. 2010.
[51] P. K. Meher, J. Valls, T. B. Juang, K. Sridharan, and K. Maharatna, “50 Years of CORDIC: Algorithms, Architectures, and Applications,” *IEEE TCAS-I: Regular Papers*, vol. 56, no. 9, pp. 1893–1907, Sept 2009.
[52] M. Gautschi, A. Traber, A. Pullini, L. Benini, M. Scandale, A. Di Federico, M. Beretta, and G. Agosta, “Tailoring Instruction-Set Extensions for an Ultra-Low Power Tightly-Coupled Cluster of OpenRISC Cores,” in *IFIP/IEEE VLSI-SoC*, 2015, pp. 25–30.
[53] H. Kaul, M. Anders, S. Mathew, S. Hsu, A. Agarwal, F. Sheikh, R. Krishnamurthy, and S. Borkar, “A 1.45GHz 52-to-162GFLOPS/W Variable-Precision Floating-Point Fused Multiply-Add Unit With Certainty Tracking in 32nm CMOS,” in *IEEE ISSCC*, Feb 2012, pp. 182–184.
[54] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge university press, 2003.



Michael Gautschi (S'14) received the M.Sc. degree in electrical engineering and information technology from ETH Zürich, Switzerland, in 2012.

Since then he has been with the Integrated Systems Laboratory, ETH Zürich, pursuing a Ph.D. degree. His current research interests include energy-efficient systems, multi-core SoC design, mobile communication, and low-power integrated circuits.

Michael Schaffner (S'13) received the B.Sc. and M.Sc. degrees from ETH Zürich, Zürich, Switzerland, in 2009 and 2012, respectively, where he is currently pursuing the Ph.D. degree.

He has been a Research Assistant with the Integrated Systems Laboratory, ETH Zürich, and Disney Research, Zürich, since 2012. His current research interests include digital signal processing, video processing, and the design of very large scale integration circuits and systems.

Michael Schaffner received the ETH Medal for his master thesis in 2013.

Frank K. Gürkaynak obtained his BSc. and M.Sc. degrees from Electrical and Electronical Engineering Department of the Istanbul Technical University and his Ph.D. degree from ETH Zürich. He is employed by the Microelectronics Design Center of ETH Zürich and his research interests include design of VLSI systems, cryptography, and energy efficient processing systems.

Luca Benini (F'07) is the Chair of Digital Circuits and Systems with ETH Zürich, Zürich, Switzerland, and a Full Professor with the University of Bologna, Bologna, Italy. He received the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, in 1997.

He has served as the Chief Architect for the Platform2012/STHORM project with STMicroelectronics, Grenoble, France in 2009-2013. He has held visiting/consulting positions at EPFL, Stanford University, IMEC. He has authored over 700 papers in peer-reviewed international journals and conferences, four books, and several book chapters. His current research interests include energy-efficient system design and multicore system-on-chip design.

Dr. Benini is a member of Academia Europaea.