Journal of Engineering Research and Education
Volume 18, 2026 [87-102]

The Evolution and Future Prospects of Logarithmic Number Systems: from Origins to Advanced Architectures

Mohammed Abdulaziz Saleh Bin Ali Gaber[1*], Siti Zarina Md Naziri[1,2] and Rizalafande Che Ismail[1,2]

[1]Faculty of Electronic Engineering & Technology, Universiti Malaysia Perlis, Arau, Perlis, Malaysia
[2]Centre of Excellence for Micro System Technology (MiCTEC), Universiti Malaysia Perlis, Arau, 02600, Perlis, Malaysia

ABSTRACT

The Logarithmic Number System (LNS) has emerged as a promising alternative to floating-point arithmetic, particularly in digital signal and image processing applications. While LNS efficiently manages multiplication, division, and square-root operations through fixed-point arithmetic, addition and subtraction remain challenging due to inherent non-linearity in both operations and a singularity issue unique to subtraction. These problems require significant memory resources to maintain floating-point precision, using interpolation techniques to defeat non-linearity and co transformation procedures to resolve singularities in subtraction. This review provides a complete overview of the evolutionary development of LNS, highlighting these main innovations and new developments aimed at reducing memory usage and hardware efficiency in LNS implementations. The paper finishes with an exploration of promising directions aimed at increasing the efficiency of LNS so that it will be a competitive option to floating-point arithmetic in a wider variety of computing environments.

Keywords: Logarithmic Number System, digital signal processing, image processing, co-transformation, hardware optimization.

1. INTRODUCTION

The Logarithmic Number System (LNS) has attracted significant attention as a potential substitute for floating-point (FP) arithmetic in those applications tackling digital signal processing (DSP) and image processing, where heightened computation efficiency is of critical concern. Beginning in the 1990s, scholarly literature focused more on LNS as a prospective substitute for FP systems due to its capability to perform complex operations such as multiplication, division, and square-root computation using fixed-point addition and subtraction, thus enhancing processing speed and precision under specific conditions [1], [2]. Early use, such as in the European Logarithmic Microprocessor (ELM) and the Gravity Pipeline (GRAPE) system, had demonstrated the efficiency of the logarithmic number system (LNS) in special computing applications, from astronomical calculations to Hidden Markov Models [3], [4]. Those early use cases had shown that, if well-designed, LNS-based processors could match or even exceed the performance of floating-point (FP) systems, particularly in circumstances with ubiquitous arithmetic operations.

However, the widespread application of LNS is hindered by the non-linearity of subtraction and addition. In addition to this non-linearity, subtraction operations are further marred by singularity problems, which make it hard to balance accuracy and computing and storage efficiency [5]. To solve such issues, several methods have been put forward. They involve interpolation techniques to resolve non-linearity and co-transformation procedures to tackle the issue of singularity [6], [7], and [31]. Implementing such methods involves large lookup tables (LUTs) for storing values, hence hardware complexity as well as high memory demand [8].

*abdulazizsaleh@studentmail.unimap.edu.my

Recent research has centered on the optimization of LNS architecture using memory-saving techniques, including multipartite tables and sophisticated interpolation methods, that greatly compress LUT sizes without sacrificing FP-equivalent accuracy [9], [10]. In addition, studies incorporating hardware-based testing, such as FPGA implementations, have demonstrated encouraging performance in accelerating computational efficiency, elevating processing speed, and simultaneously lowering memory usage, thus paving the way for the implementation of LNS in real-time digital signal processing [11].

As high-speed arithmetic applications of LNS matures, work continues in an effort to further uncover optimization opportunities in LNS. This paper reviews the advances of LNS based on approaches and recent advances attempting to circumvent the prevalent challenges presented with addition and subtraction. While discussing these approaches, we hope to lay out ways forward that will show further development and usage of LNS as a legitimate alternative to FP arithmetic in many areas of computation.

2. EMERGENCE OF LNS

Kingsbury and Rayner first introduced the Logarithmic Number System (LNS) in 1971 when using it in the design of digital filters, which was a landmark in the development of digital signal processing (DSP) [12]. By using a logarithmic instead of a linear representation of signal amplitudes, they converted intricate calculations such as multiplication and division into easy addition and subtraction operations on logarithmic values, thus enhancing computational speed.

This logarithmic representation not only enabled simpler computation but also significantly improved the dynamic range to as much as 4×10^{10}, several orders of magnitude better than the situation of standard fixed-point arithmetic. LNS also provided the same precision for all signal levels, and thus small and large signals were equally precise.

One of the key challenges in the implementation of LNS was the difficulty of addition and subtraction operations, particularly in calculations involving functions like $\log_p(1 \pm p^{-d})$, where d is the logarithmic difference. Two potential solutions to this challenge were proposed, one being direct computation utilizing the logarithm and exponential functions where used in a derived manner and the other being the use of pre-computed lookup tables (LUTs) to hold the original values for a larger amount of memory but quicker access.

To illustrate the practical feasibility of LNS, a second-order low-pass filter was built using 16-bit logarithmic arithmetic that showed improved step response and dynamic range over a related filter synthesized using fixed-point arithmetic. This progress demonstrated the potential of LNS for processing low-amplitude signals and expands its application by overcoming precision limits of fixed-point arithmetic, leaving open hallmarks for infinite future possibilities LNS use in DSP.

In 1975, Swartzlander and Alexopoulos introduced the Sign/Logarithm Number System (S/LNS) [15], which represents a significant advancement in the development of logarithmic number systems by breaking free from one of their most basic limitations, which was the representation of negative numbers. The S/LNS combines the sign bit with the binary logarithm of an absolute value of the number scaled properly to ensure logarithmic values are always positive. The S/LNS greatly improved the flexibility and ability of LNS to represent negative and positive integers so that applications of LNS could now include digital image processing, pattern recognition, and radar signal processing.

The authors presented step-by-step algorithms for conducting arithmetic operations in the S/LNS system, with multiplication and division being carried out via summation and subtraction of logarithmic values. Addition and Subtraction was even more challenging because of the non-linear functions needed in the computation of $\beta(X) = \log_2 (1+2^X)$ and $\gamma(X) = \log_2 (1-2^X)$, where X is the logarithmic difference between operands. These operations needed special hardware to be treated efficiently. Figure 1 shows how an adder/subtractor for S/LNS is implemented, along with the logic for determining operand order and carrying out the needed computations for these difficult operations.

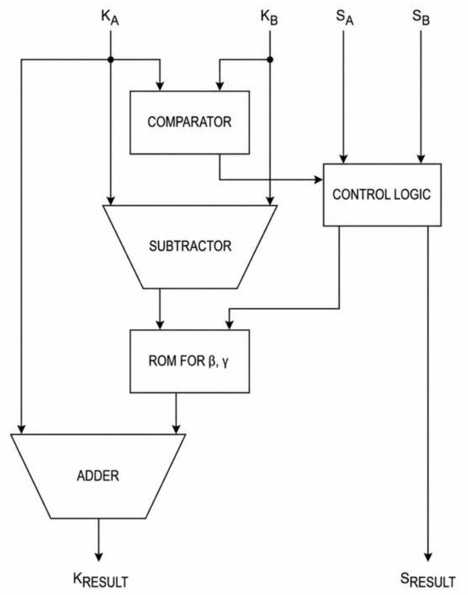

Figure 1. adder/subtractor for S/LNS

In assessing the performance of S/LNS, Swartzlander and Alexopoulos conducted a comparative analysis against conventional arithmetic structures, indicating its superiority in terms of uniform latency of operation. Table 1 illustrates a comparative breakdown of delay for various arithmetic operations, demonstrating the system's capability to provide consistent performance, unlike in conventional systems where addition registers a very short time relative to multiplication or division. Authors identified that S/LNS was very valuable in contexts requiring high computing uniformity but that its relative fixed error represented an obstacle for applications requiring high precision accuracy.

Table 1 Delay Comparison

Arithmetic Unit Operation	Sign/Logarithm	Conventional
Add	4 T_{ADD}	T_{ADD}
Substract	4 T_{ADD}	T_{ADD}
Multiply	2 T_{ADD}	7 T_{ADD}
Divide	2 T_{ADD}	14 T_{ADD}

Swartzlander and Alexopoulos established a basis for subsequent developments in logarithmic arithmetic, developing a framework that traded off computational speed for increased applicability. Their work is still at the center of LNS-based architecture development

3. METHODS OF LNS ARCHITECTURE DESIGN

Optimization of LNS addition and subtraction has focused predominantly on minimizing the size of lookup tables for efficiency. By addressing issues such as hardware area reduction and improving the speed of operation, LNS implementations have been able to approach the performance of floating-point systems. This has generated a significant amount of interest in optimizing the methods used for arithmetic operations, with the objective of improving approximation schemes for addition and subtraction.

Historically, the design of LNS architectures is categorized into three broad techniques: direct and indirect lookup tables, digit-serial methods, and piecewise polynomial techniques. The choice of the technique typically depends on the word length of the numbers employed in the logarithm representation because different implementations could yield optimal outcomes depending on the characteristics of the application itself. The selection of the correct technique is crucial in achieving optimal accuracy, efficiency, and utilization of resources in implementing LNS.

3.1 Direct And Indirect LUT Method

The 1975 Sign/Logarithm Number System (S/LNS) employed lookup tables (LUTs) constructed from Read-Only Memories (ROMs) to effectively handle addition and subtraction in the logarithmic domain [13]. The system precomputed logarithmic values for significant diminishing of real-time computations increasing operational efficiency with less. LUT memory spaces grew linearly with word length, an 8-bit design requiring 4096 bits, under the available ROM technology, an acceptable number. The LUT-based scheme has limitations or scalability problems associated with increasing word lengths; nevertheless it established a rudimentary basis to developing efficient logarithmic number system architectures.

After the first use of lookup tables (LUTs) in the Sign/Logarithm Number System, the implementation of a 20-bit logarithmic number system (LNS) processor improved this technique by reducing the memory requirements of addition and subtraction operations [14]. This investigation utilized LUTs to calculate the nonlinear functions which are required for those operations, which were $\log_2(1 + 2^{-D})$ and $\log_2(1 - 2^{-D})$, where D is the difference of the logarithmic exponents.

To address memory utilization, the lookup table (LUT) was divided into a number of read-only memory (ROM) banks. The ROM was assigned to specific ranges of D. By resizing the ROM so that it would fit in the function behavior in the regions of the curve that changed less, the design was able to successfully ignore redundant storage. In addition, for some ranges, a programmable logic array (PLA) was used, minimizing total memory utilized. In the end, the total LUT size was reduced to 154K bits, a significant decrease from previous versions. The memory-driven design was evidence of LNS processors with great arithmetic performance and overall good speed/resource trade-offs; therefore, LUT based methods were critical when developing logarithmic arithmetic systems.

The study by David M. Lewis [15] builds on previous research in logarithmic number systems (LNS) and presents a new architecture for minimizing the size of lookup tables (LUTs) required in LNS addition and subtraction. By combining this study with previous work that emphasized the important need for LUT efficiency, Lewis is targeting high accuracy through an optimized configuration.

In this novel architecture, the LUT size is diminished through the use of a Taylor series first order approximation. This approximation establishes an input range into variable length intervals, according to the degree of nonlinearity of the function being performed. As an example, it uses single input LUTs, to store $f_a(r) = \log(1 + 2^r)$, and $f_s(r) = \log(1 - 2^r)$, for LNS addition, and LNS subtraction respectively, instead of dual-input LUTs, which is what would typically be used. Therefore, this novel approach would greatly facilitate design and help increase the scaling of such a design for applications where accuracy is paramount.

The system architecture takes a systematic approach towards accuracy distribution among the sections of the LUTs. It mitigates the accuracy errors due to Taylor approximation and finite precision. For example, it minimizes the table size while preserving the accuracy by modifying the associated table parameters (e.g. input range and number of fractional bits) as appropriate based on the precision required. The improved architecture of Lewis indicates a large reduction in utilizing ROM, potentially providing as much as 50% more fractionary precision with the same size of ROM compared to the previous architectures. This represents a significant improvement in the use of LNS, particularly in applications requiring long word lengths.

This research broadens the practical application of LNS arithmetic by showing that complete mathematical analysis can be used to devise small, efficient hardware architectures. By reducing the overall LUT size, we not only improve memory utilization, but we can also perform high-precision arithmetic operations that would have otherwise not been possible. This is aligned with the continued development of LUT methods in LNS design.

Lewis [16] has added to the LNS arithmetic literature with interleaved memory to build upon improved stored lookup tables (LUTS). His intended methodology of interleaved memory-based LUTS relies on the polynomial interpolation approach he earlier developed, but he is intending to solve memory and accuracy tradeoffs by storing the actual function values in bifurcated sections of overlapping intervals. Interleaved memory LUTS can have a smaller footprint than traditional LUTS or coefficient-based interpolators, while still attaining virtually equivalent design precision outcomes as before. Demo the 32-bit LNS arithmetic unit, and interleaved memory LUTS can mitigate memory by greater than 91-K bits of ROM memory and still produce precision results that are nearly comparable to floating-point systems. This represents a significant step toward the scalability and efficiency of LNS designs based on LUTS.

The European Logarithmic Microprocessor (ELM) project [17], represents an important advancement in LNS arithmetic through better lookup table (LUT) utilization of LNS addition and subtraction functions. ELM built on the previous ideas of Lewis and previous work by using partitioned LUTs to compute functions of the form $F(r) = \log_2(1+2^r)$. Partitions were created by saving a number of LUTs that were pre-computed over the range of values and their derivatives, and this allowed for interpolation over the intervals.

An integral aspect of innovation in this project involved the incorporation of error correction processes into the LUT architecture. As illustrated in Table 2, additional data structures were used to store two sets of error bounds (E) and portioned corrections (P) so that interpolation and error correction could be accomplished at the same time. Thus, computing errors were diminished as much as possible in addition to maintaining memory use and accuracy.

Table 2 Error in 32-Bit High-Accuracy LNS Addition Operation

$\lvert e\rvert_{\text{max rel log}}$	0.5046
$\lvert e\rvert_{\text{av rel log}}$	0.2509
$e_{\text{max rel arith}}$	0.3489
$e_{\text{min rel arith}}$	-0.3498
$e_{\text{av rel arith}}$	0.0066
$\lvert e\rvert_{\text{av rel arith}}$	0.1739

3.2 Direct And Indirect LUT Method

The digit-serial approach, as demonstrated in this study [18], focuses on computing in an iterative fashion by processing digits one at a time to create logarithmic arithmetic. This approach has been referred to as the online algorithm and is achieved through the proposed Dual Redundant Logarithmic Number System (DRLNS). In contrast to standard logarithmic arithmetic, DRLNS employs a form of redundancy, showing numbers as non-standard representations containing two separate components. In this dual-component construction, DRLNS enhances the efficiency, error free, and the arithmetic capabilities of logarithmic representations.

This method has the advantage of alleviating the complication of computing logarithmic subtraction (db) with associated high-performance computing costs involved for the implementation of deep learning. The DRLNS transforms db into iterative algorithms through various parallel high-performance computing methods, optimal hardware (multiplexers, fixed-point adders, etc.), and computing linear approximations related to addition and subtraction to minimize the computation.

A potential example of this is the way addition occurs using these transformations, and no materiality computation overhead associated with logarithms is added.

In addition, through the application of redundancy, the digit-serial method circumvents the catastrophic errors commonplace with logarithmic systems while maintaining the numerical stability. As a result, the high degree of accuracy makes the digit-serial method ideal for scenarios involving repeated high-precision calculations on a real-time basis, including but not limited to filtering and signal processing applications. The efficiency of the digit-serial method allows for versatility in a larger system due to the way it encapsulates building blocks to retrieve the sum and difference of calculations in a single hardware block.

The digit-serial approach discussed in [19] is designed to mitigate difficulties which arise in LNS addition and subtraction, and minimizes the LUT resources required to process its output. The digit-serial method adopts a pipelined approach to handle LNS addition and subtraction in a way that reduces any requirements for large LUTs significantly. The computation occurs in two stages: an exponential stage and a logarithmic stage, both of which are iterated in a digit parallel approach. The exponential stage is additive normalized and uses small LUTs for the precomputed normalization terms, while the logarithmic stage is multiplicative normalized and uses LUTs to maintain iterative adjustments in the calculation.

For a 33-bit LNS addition/subtraction unit, the proposed method achieves a total LUT size of only 0.543 Mbits, a notable improvement compared to earlier implementations. Additionally, the pipelined architecture represents a technology that has linear hardware scaling with respect to word length, which is very favorable for high-precision applications. Overall, as a continuation of Lewis et al's work, this paper highlights the ability of digit-serial methods and new, rapid speed computation technology (pipelining, etc.) to enhance performance in LNS architectures while providing a manageable amount of hardware.

3.3 Piecewise Polynomial Method

The piecewise polynomial method, in this case linear interpolation, is implemented to solve the memory limitation issues of logarithmic number systems (LNS) [20]. This approach approximates the nonlinear function $\log_2 (1 + 2^r)$ necessary for performing addition by partitioning the range of inputs into smaller system intervals corresponding to input values and then interpolating over those values within each system interval. The input s is decomposed into its integer part (s1) and its fractional part (s2); the integer part is stored in smaller look-up tables (LUTs), and the fractional part is interpolated using primitive polynomial functions. The segmentation of the input reduces the overall size of the LUTs while maintaining a level of accuracy, as a LUT might only be 4-bits smaller than full precision representation. A modified version of this technique is also proposed which further reduces the memory requirement and provides at least 4-bits more accuracy than standard logarithmic arithmetic. The aforementioned approach provides an adequate compromise between precision and memory use for a scalable approach to high precision LNS arithmetic.

Lewis identified a linear approximation method to overcome memory constraints associated with LNS arithmetic by decreasing the size of the ROM-based look-up tables [15]. The method subdivides the domain into variable-sized segments and uses Taylor series approximation to compute the nonlinear functions f_a (r) and f_s (r), which are required for LNS addition and subtraction. The hardware design for this approximation, which is illustrated in Figure 2, consists of the function values (f), the derivatives (df/dr), multipliers (mpy), and an adder in order to compute the intermediate values. The method can reduce the memory size required to evaluate 32-bit operations by a factor of 118 with reasonable accuracy, making it an extremely compute-efficient solution for high accuracy LNS arithmetic.

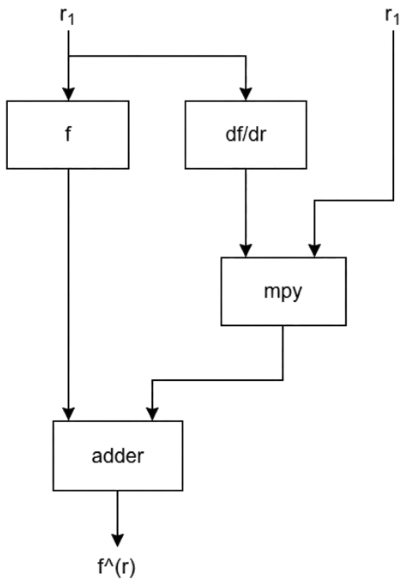

Figure 2. Taylor series approximation hardware

Lewis continued to develop the computational efficiency of LNS arithmetic by using a second order polynomial interpolation approach that is combined with interleaved memory architectures [16]. This interpolation method is different to the previous one using a Taylor series, as it stores function values in memory, instead of coefficients to create the polynomial. Interleaved memory allows multiple interpolation intervals to be supported by each stored value. The storage reduction with this memory effectively reduced memory storage while performing to achieve accuracy beyond IEEE single-precision. The interleaved memory system reduced not only the computational hardware but also sustained the precision of the computation to enable a LNS arithmetic unit with mechanical size and unit efficiency, compared to traditional arithmetic. This method significantly improves storage savings and reduces hardware implementations of polynomial interpolation than in traditional polynomial interpolation methods.

The piecewise polynomial approach, as investigated in [21], builds on earlier developments by using Chebyshev polynomials to increase the effectiveness of LNS addition and subtraction for FPGA-based systems. This method efficiently lowers memory needs while preserving better-than-floating-point (BTFP) precision by dividing the function domain into smaller parts and using polynomial interpolation within each.

The method decomposes the subtraction function into a correction function and a base two logarithm function, both of which can be approximated via polynomial interpolation, thus avoiding the issues when subtracting near zero and related singularity problems. The additional precision bits (x) required over f-bit LNS addition or subtraction operation are shown in Figure 3 to demonstrate the level of precision required to achieve BTFP accuracy.

For its FPGA implementation, the system uses distributed lookup tables for auxiliary computations such as $\log_2(x)$, and block RAM to store polynomial coefficients. This unique strategy decreases computational latency and resource usage when compared to non-approximation methods, confirming that polynomial approximation techniques are feasible for use within modern reconfigurable hardware systems.

The European Logarithmic Microprocessor Project [3] provides a comprehensive description of the implementation of LNS arithmetic with linear Taylor interpolation for a polynomial approximation. The approach allows for a high level of accuracy while minimizing storage space. This method drastically reduces ROM storage by separating addition from subtraction and using a simple first-order approximation along with an error correction process. The project demonstrated that this polynomial method took up a very small fraction of storage space and achieved omnidirectional accuracy equal to that of a floating point equivalence. The additional logging coordinate locations for interpolation increased the level of accuracy in these important areas of the LNS functions, using tiny table-based error correction related to the storage architecture.

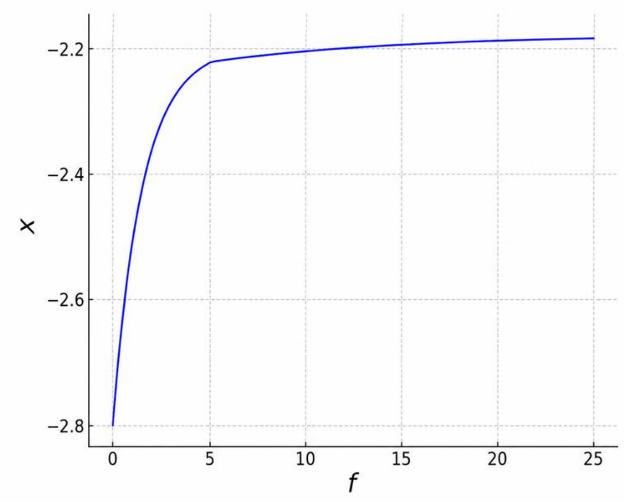

Figure 3. The number of bits x of extra precision required to calculate an f-bit LNS add/sub function with BTFP accuracy

In order to optimize LNS implementation on FPGAs, polynomial approximation methodologies were significantly advanced in FPGA Designs with Optimized Logarithmic Arithmetic [22]. The authors developed a library generator that implemented effective segment division and coefficient calculation through the use of a minimax polynomial approximation. The focus of the work was on exploring the trade-offs between precision (or accuracy) and use of hardware resources (e.g. hardware multipliers and block RAM). In particular, the adaptive segmentation method under test maintained better-than-floating-point accuracy and reduced approximation error while enabling significant reductions in area and response time improvements particularly for high-throughput applications compared to earlier implementations.

3.4 Co-transformation Method

The co-transform method was developed in [23] to enhance computational efficiency within logarithmic number systems (LNS). This method is particularly valuable for addressing the inherent operations of addition and subtraction in an LNS when values near a singularity present themselves, as this may result in expensive methods of interpolation. Co-transformation is the act of taking a calculation one would not prefer to perform and changing it from a difficult region to one that is more manageable. For example, we may prefer to transform a subtraction near a singularity into an equal computation in addition with fewer costs. The method uses mathematical transformation; clearly emerging from these transformations are constraints on interpolation usage and the operands still maintain a relationship.

In the paper, it discusses two main types of co-transformations. The first type is basically a truncated series expansion, from which we can approximate the subtraction logarithm close to zero. We can reduce the size of lookup tables without sacrificing accuracy by using lookup tables with precomputed values and breaking the operands into smaller sizes. The second type is an algebraic co-transformation, which provides similar results using simple algebraic operations. One advantage is that these co-transformations do not require a large, precomputed tables, making them useful for software or hardware applications.

A new co-transformation method is presented in [24] to avoid the issues caused by the logarithms of subtraction (db), which is a significant advancement to LNS subtraction. The co-transformation method not only improves accuracy and simplifies computations, it reduces hardware complexity by converting the subtraction operation into an addition operation. The co-transformation method also reduces lookup table size hence offers efficiencies in the hardware implementation. The co-transformation method is distinct from past methods because of its special commutation properties for operands, which makes it more suited to real-time systems.

Expanding on the aforementioned purpose, [25] further investigates the employment of co-transformation in HDL library optimization. This report shows exactly how co-transformation improved area efficiency over both interpolation and multipartite methods, obtaining similar or better accuracy for LNS subtraction. Co-transformation is acknowledged as an important mechanism for overcoming issues relating to subtraction operation constraints in LNS by taking into consideration both the overall computational accuracy and limitations on hardware resources.

In addressing singularities in LNS subtraction, Coleman et al. [3] detail the process of co-transformation in the European Logarithmic Microprocessor (ELM). In addition to bypassing the singularity issue, the co-transformation method establishes a relation between subtraction, where the logarithmic difference r is approaching 0, to an addition in a new domain. To accomplish this diversion from the singularity, co-transformation takes advantage of a first-order approximation which brings the logarithmic arguments to locations that are more suitable for interpolation. The dedicated transformation section in the hardware implementation uses lookup tables and precomputed values to convert the input to the co-transformation domain. These tables use some form of error coding to utilize correction factors to realign the transformed subtraction back to the logarithmic scale after interpolation. This method of purposely working in a smaller table and the reduction of control logic simplifies the design of the LEM, optimizing the design for space-limited development of the hardware.

By improving the processing of changed arguments, Coleman and Ismail [6] expand on the co-transformation technique. Their method employs a two-tiered co-transformation architecture, with the second stage applying finer corrections for accuracy and the first stage handling large-scale modifications using a coarse transformation. Because of this arrangement, the converted subtraction is always within an interpolation-manageable range. To remove the nonlinear singularity behavior and move the interpolation to a more linear domain, the transformation is algebraically defined and the arguments are changed. To ensure low memory utilization and good precision, the lookup tables needed for this method are hierarchically arranged and smaller at each level. Furthermore, by avoiding repetitive operations during transformation and guaranteeing that the outcomes are directly compatible with normal LNS addition processes, the method makes use of certain LNS features to streamline the computational pipeline.

Naziri et al. [26] upgraded LNS arithmetically, by including second-degree interpolation with wider co-transformation range where r can be equal to as much as 2. On par with first-degree methods, this method of interpolation retains accuracy while saving 24% memory. The second-degree interpolator removes the need for independent error-correcting systems to increase precision and simplify the architecture. The paper discusses the importance of co-transformation, emphasizing the required trade-off of memory and accuracy to solve issues related to LNS subtraction, further establishing the importance of co-transformation to new LNS designs.

3.5 Special Function Unit

Gautschi et al. [32] have developed the shared logarithmic number unit (LNU) architecture with an emphasis on low-energy consumption for nonlinear function kernel processing. Therefore, it is a suitable architecture for the energy-constraints of wearables and The Internet of Things. In this architecture, the Special Function Unit (SFU) is integrated as an element in a 65-nm CMOS multicore cluster. The SFU has the ability to perform addition, subtraction, multiplication, division, and square root i.e., all logarithmic operations and is intended to speed-up complex nonlinear functions such as 2^x, $\log_2(x)$, sine, cosine, and atan2.

The SFU approximates nonlinear computations reliably and efficiently by using polynomial interpolation and range reduction. By using these methods, the SFU reduces the computation cost and eliminates the large lookup tables that are frequently connected to logarithmic arithmetic.

Additionally, by allocating resources across several cores, the shared LNU implementation reduces the per-core area overhead. To guarantee low latency and energy consumption, it makes use of specific datapaths and effective control logic. Performance evaluations show that this architecture achieves up to 4.1× more energy efficiency for nonlinear calculations than traditional floating-point units (FPUs). Its scalability is demonstrated by its integration into a multicore cluster, where shared processing further minimizes redundant circuitry. By enabling precise and efficient nonlinear function computations, this architecture offers a compelling option for low-power, high-performance computing in modern embedded systems, which is a significant improvement over conventional technique.

4. HYBRID ARCHITECTURE

By integrating their capabilities, hybrid designs take advantage of the complementary advantages of both floating point (FLP) and logarithmic number systems (LNS). While FLP effectively handles addition and subtraction across a wide dynamic range, LNS excels at multiplication and division by restricting mathematics to fixed-point addition and subtraction. A well-rounded solution for applications like digital signal processing (DSP) and computational graphics can be obtained by integrating the two systems into a single architecture. The following sections discuss significant contributions to hybrid architectures, emphasizing their evolution and technological innovations.

The Florida University Floating Unit $(FU)^2$ is a hybrid architecture in [27] that combines the efficiency of the logarithmic number system (LNS) with the accuracy of floating-point (FLP). It combines addition and multiplication into a single hardware channel and does away with FLP exponent alignment, which cuts down on computation time. Multiplication becomes straightforward exponent addition when logarithmic representations are used, and addition makes advantage of effective table lookups. Along with a common adder-multiplier path and an optimized accumulator, the architecture enhances speed and reduces hardware complexity for high-performance computing applications by completing non-pipelined adds in three memory access times ($3T_m$).

To capitalize on their respective advantages, the floating-point (FLP), signed-digit (SD), and logarithmic number systems (LNS) are combined in the hybrid architecture suggested in [28]. Through the representation of FLP numbers in SD/LNS format and back, this method enables efficient computing. For instance, a block diagram with a log lookup table and SD addition is used to transform FLP numbers into SD/LNS, as seen in Figure 4. Compared to conventional FLP processors, this architecture provides notable efficiency gains, lowering addition and subtraction latency by 10 gate delays and multiplication/division latency by 8 gate delays. Furthermore, by integrating SD representation, digit-level parallelism is improved by reducing carry propagation delays. Small programmable logic arrays (PLAs) are used to efficiently convert the SD format to sign-magnitude form, reducing redundancy and guaranteeing compact lookup tables. Such optimization is a crucial component of hybrid architecture for computational efficiency since it produces a more regular layout for VLSI design and significant speed gains.

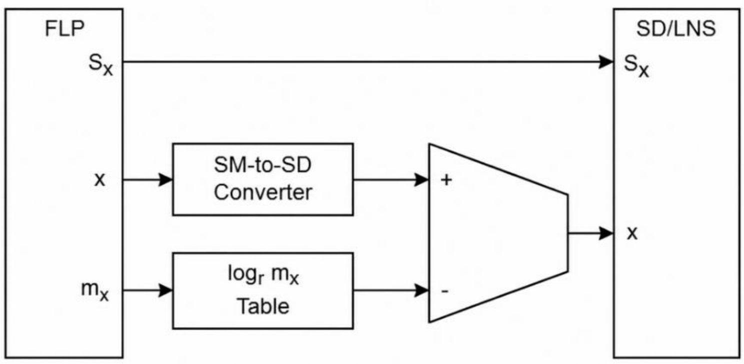

Figure 4. Block Diagram of the FLP-to-SD/LNS Conversion

Optimized for geometric and complicated arithmetic applications, the hybrid architecture in [29] combines conventional and logarithmic number systems. LNS is utilized for efficient multiplication and division, whilst classical arithmetic ensures precision in addition and subtraction. The architecture uses a standard IEEE 754 32-bit floating-point processor for input and output, which allows for compatibility with existing systems.

The study in [30] introduces three hybrid number systems: Denormal Logarithmic Number System (DLNS), Denormal Mitchell LNS (DMLNS), and Denormal Offset Mitchell LNS (DOMLNS) to combine the benefits of fixed-point (FXNS) and signed logarithmic number systems (SLNS). To satisfy the requirements of applications that require accuracy and resource efficiency, these systems use gradual underflow handling, much like the IEEE-754 floating-point standard. DLNS uses SLNS operations with additional data channels for mixed arithmetic; nevertheless, because it relies on SLNS tables, it is expensive for operations like as division and multiplication. In order to address this, DMLNS and DOMLNS employ Mitchell's method to reduce hardware costs and streamline processes for division and multiplication, albeit at the expense of a slight decrease in accuracy in the denormal range. DMLNS maintains SLNS compatibility for large values, but DOMLNS prioritizes resource minimization over compatibility. The versatility of hybrid designs in striking a balance between accuracy, performance, and resource efficiency for a range of computing requirements is demonstrated by this work.

5. APPLICANTION OF LNS

As discussed in previous sections, the logarithmic number system (LNS) can simplify some computations without sacrificing accuracy, and in some cases, it can even outperform floating-point (FLP) representation in terms of round-off or absolute error. Because of these features, designers of digital signal processing (DSP) have been more inclined to use LNS, especially in signal processing, video, graphics, and control applications. Furthermore, LNS has been documented in a number of fields, including as encryption [35], digital regulators [34], and commercial aviation cabin air-pressure management systems [23].

Many arithmetic operations are needed for filters in signal processing. As a result, a number of implementations have used the logarithmic number system (LNS) in filter designs, taking advantage of its benefits in reducing round-off errors and allowing for possible word length reductions. For example, studies by [36] and [37] investigated the use of LNS in filters, whereas the latter concentrated on its use in digital hearing aids. Furthermore, [38] created logarithmic decoders to investigate how speech and sinusoidal inputs affect quantization noise.

In research on the Fast Fourier Transform (FFT), the logarithmic number system (LNS) has attracted a lot of interest in addition to filters. Notable contributions include [39], [40], and [41], who investigated the use of log-polar complex LNS (CLNS) in FFT implementations, and [42], who used LNS in the design of wireless modems. Additionally, LNS has been used to create specialized decoders, including the turbo decoder developed by [43] and [44], which both use the log-sum product decoding algorithm (log-SPA).

The use of the logarithmic number system (LNS) in computer graphics and video processing has been thoroughly studied by researchers. According to the results, standalone LNS and hybrid-LNS systems work well with MPEG and JPEG applications because they provide low-cost, low-power hardware architectures and integrate easily. For example, Arnold showed in [45] that LNS-based MPEG decoding reduces power consumption and makes word-length reduction easier. Similar to this, Lee in [46] used a hybrid technique to build a discrete cosine transform (DCT) and went on to create a hybrid inverse DCT (IDCT) decoder that allowed intraframes to be transmitted repeatedly while maintaining overall quality. Furthermore, in contrast to signed LNS implementations, dynamic range LNS (DRLNS) was used to do away with the necessity of LNS subtraction operations in MPEG decoder applications [47].

The logarithmic number system (LNS) is especially useful in graphics processing since it encompasses a broad range of intricate arithmetic operations. Hybrid techniques have been implemented by numerous graphics processing applications. For instance, as was covered in earlier parts, [48] looked into the use of hybrid floating-point and LNS systems in graphics applications. Furthermore, FXPHNS, a 3D vector hybrid processor created especially for portable graphics systems, was created by [49] and [50].

Moreover, as demonstrated in [51]. Because LNS can handle the non-uniform distribution of input, it is very beneficial for convolutional neural networks (CNNs) and produces excellent accuracy for image classification applications.

6. DISCUSSION

Significant developments are highlighted in the discussion of Logarithmic Number System (LNS) design methodologies, along with their effects on hardware performance, memory utilization, and computational efficiency. A comparison of various LNS architectures is given in Table 3, using the floating-point unit (FPU) design by Kwon et al. [33], Coleman and Ismail [2], [6], Gautschi et al. [32], Naziri et al. [26], and others as a point of reference. In order to ensure a fair comparison of their performance data, both designs were implemented utilizing the same technology node.

The work of Gautschi et al. [32] shows superiority in minimizing both maximum latency and LUT size among the examined systems. Gautschi's design uses a shared Special Function Unit (SFU) optimized for logarithmic arithmetic operations, resulting in a maximum latency of 12.5 ns and a LUT size of only 71.5 kbits. This reduces hardware needs and increases energy efficiency. However, compared to designs that incorporate both ADD and subtraction (SUB), Gautschi's design is less complete even if it achieves the smallest area of 375,000 μm^2. This value solely reflects the addition (ADD) unit.

Naziri et al.'s [26] design, on the other hand, is the most accurate of the LNS systems assessed, achieving the smallest maximum error of 0.4514 units of least precision (u.l.p.). All LNS designs, however, keep their maximum error below the 0.5 u.l.p. threshold of Kwon et al.'s FPU, as indicated in Table 3, indicating that all systems are precise enough for real-world use.

Table 3 Comparison of different LNS and FLP units of 0.18 μm technology

Design	Coleman & Ismail [2], [6]	Gautschi et al (LNS C) [32]	Naziri et al [26]	Kwon et al (DIVA) [33]
Technology (nm)	180	180	180	180
Type	LNU	LNU	LNU	FPU
Word width (bits)	32	32	32	32
Interpolation	First degree (Improved Lagrange)	Second degree (Remez + Minimax)	Second degree (Newton)	-
Co-transformation	Second order	Special Function	Second order	-
Max rel error Accuracy (u.l.p)	0.4987	0.4755	0.4514	0.5
LUT Size (kbits)	183.3	71.5	173.1	Not mentioned
Delay min/max (ns)	7.1/14.79	12.5/12.5	8.65/17.52	5/18.80
Area (μm^2)	589,357 (Add/Sub)	375,000 (Add only)	562,847 (Add/Sub)	481,635 (Add/Sub)

7. CONCLUSION AND FUTURE WORK

As an alternative to floating-point arithmetic (FLP), the Logarithmic Number System (LNS) has demonstrated impressive promise, especially for applications that demand effective multiplication, division, and square-root operations. But despite its inherent benefits, LNS adoption has been slow because of major issues with memory needs, addition and subtraction operations, and a lack of standardization. LNS lacks a defined standard, in contrast to FLP, which benefits from the IEEE 754 standard, which guarantees uniformity and interoperability between platforms. This lack restricts its broad adoption and leads to implementation fragmentation. Standardizing LNS should be a top priority for future research in order to facilitate wider use and encourage the creation of hardware solutions that are optimized.

By combining FLP and LNS's benefits, the hybrid design provides a workable way to boost computing efficiency. By using FLP for addition and subtraction and LNS for operations like multiplication and division, hybrid systems offer a balanced approach to precision and efficiency. The difficulty of switching between the two systems and the additional circuitry required for conversions provide significant challenges. These issues not only increase complexity but also have an impact on hardware size and power consumption. More research is needed to enhance hybrid architectures, with a focus on developing more efficient conversion mechanisms and simplifying system switching.

Combining interpolation and co-transformation approaches is the most promising method for addressing the core problems with LNS. Interpolation effectively manages the non-linearities in LNS addition and subtraction, whereas co-transformation resolves the singularity issues in subtraction operations. Recent advances indicate that these methods can significantly reduce memory requirements and improve hardware performance when incorporated into LNS systems. Trends over the past ten years make it evident that this approach is moving LNS closer to becoming widely used. Future research should focus on further enhancing this combination so that LNS can compete with FLP in a wider range of applications, such as digital signal processing and general-purpose computing.

Through standardization, improved hybrid architectures, and advancements in interpolation-co-transformation integration, LNS has the potential to transform numerical computation and provide a strong substitute for floating-point systems that meets the demands of contemporary computing.

ACKNOWLEDGEMENTS

This research is fully supported by FRGS grant, FRGS/1/2023/TK08/UNIMAP/02/1/9003-00969. The authors fully acknowledged Ministry of Higher Education (MOHE) and Universiti Malaysia Perlis for the approved fund which makes this important research viable and effective.

REFERENCES

[1] B. Parhami, "Computing with logarithmic number system arithmetic: Implementation methods and performance benefits," Computers and Electrical Engineering, vol. 87, p. 106800, 2020, doi: 10.1016/j.compeleceng.2020.106800.

[2] R. C. Ismail and J. N. Coleman, "ROM-less LNS," 2011 IEEE 20th Symposium on Computer Arithmetic, 2011, pp. 43–51, doi:10.1109/ARITH.2011.15.

[3] J. N. Coleman et al., "The European Logarithmic Microprocesor," IEEE Transactions on Computers, vol. 57, no. 4, pp. 532–546, 2008, doi: 10.1109/TC.2007.70791.

[4] J. Makino, M. Taiji, T. Ebisuzaki, and D. Sugimoto, "GRAPE-4: A Massively Parallel Special-Purpose Computer for Collisional N-Body Simulations," Astrophys. J., vol. 480, no. 1, pp. 432–446, May 1997, doi: 10.1086/303972.

[5] S. Z. M. Naziri, R. C. Ismail, and A. Y. Shakaff, "The Design Revolution of Logarithmic Number System Architecture," Electrical, Electronics and System Engineering (ICEESE), 2014 International Conference on, pp. 5–10, 2014.

[6] J. N. Coleman and R. Che Ismail, "LNS with Co-Transformation Competes with Floating-Point," IEEE Transactions on Computers, vol. 65, no. 1, pp. 136–146, 2016, doi: 10.1109/TC.2015.2409059.

[7] M. G. Arnold, E. Chester, and C. Johnson, "Training neural nets using only an approximate tableless LNS ALU," Proceedings of the International Conference on Application-Specific Systems, Architectures and Processors, IEEE, Jul. 2020, pp. 69–72, doi:10.1109/ASAP49362.2020.00020.

[8] M. S. S. M. Basir, R. C. Ismail, and M. N. M. Isa, "A Novel Double Co-Transformation for a Simple and Memory Efficient Logarithmic Number System," 2020 IEEE International Conference on Semiconductor Electronics (ICSE), IEEE, Jul. 2020, pp. 25–28, doi: 10.1109/ICSE49846.2020.9166879.

[9] F. De Dinechin and A. Tisserand, "Multipartite table methods," IEEE Transactions on Computers, vol. 54, no. 3, pp. 319–330, 2005, doi: 10.1109/TC.2005.41.

[10] G. Tsiaras and V. Paliouras, "Logarithmic number system addition-subtraction using fractional normalization," in 2017 IEEE International Symposium on Circuits and Systems (ISCAS), IEEE, May 2017, pp. 1–4, doi: 10.1109/ISCAS.2017.8050569.

[11] V. Paliouras and T. Stouraitis, "Logarithmic Number System," in Arithmetic Circuits for DSP Applications, Hoboken, NJ, USA: John Wiley & Sons, Inc., 2017, pp. 237–272, doi: 10.1002/9781119206804.ch7.

[12] Kingsbury, N. G., & Rayner, P. J. W. (1971). Digital filtering using logarithmic arithmetic. Electronics Letters, 7(2), 56–58. https://doi.org/10.1049/EL:19710039

[13] Alexopoulos, A. G. (1975). The Sign/Logarithm Number System. IEEE Transactions on Computers, C–24(12), 1238–1242. https://doi.org/10.1109/T-C.1975.224172

[14] Taylor, F. J., Gill, R., Joseph, J., & Radke, J. (1988). A 20 Bit Logarithmic Number System Processor. IEEE Transactions on Computers, 37(2), 190–200. https://doi.org/10.1109/12.2148

[15] Lewis, D. M. (1990). An Architecture for Addition and Subtraction of Long Word Length Numbers in the Logarithmic Number System. IEEE Transactions on Computers, 39(11), 1325–1336. https://doi.org/10.1109/12.61042

[16] Lewis, D. M. (1994). Interleaved Memory Function Interpolators with Application to an Accurate LNS Arithmetic Unit. IEEE Transactions on Computers, 43(8), 974–982. https://doi.org/10.1109/12.295859

[17] Transactions on Computers, 43(8), 974–982. https://doi.org/10.1109/12.295859 Transactions on Computers, vol. 49, no. 7, pp. 702-715, July 2000, doi: 10.1109/12.863040.

[18] Arnold, M. G., Bailey, T. A., Cowles, J. R., & Cupal, J. J. (1989). Redundant logarithmic number systems. Proceedings - Symposium on Computer Arithmetic, 144–151. https://doi.org/10.1109/ARITH.1989.72820

[19] Chen, C., & Yang, C. H. (1998). Pipelined computation of LNS addition/subtraction with very small lookup tables. Proceedings - IEEE International Conference on Computer Design: VLSI in Computers and Processors, 292–297. https://doi.org/10.1109/ICCD.1998.727064

[20] Taylor, F. J. (1983). An Extended Precision Logarithmic Number System. IEEE Transactions on Acoustics, Speech, and Signal Processing, 31(1), 232–234. https://doi.org/10.1109/TASSP.1983.1164042

[21] Lee, B. R., & Burgess, N. (2003). A parallel look-up logarithmic number system addition/subtraction scheme for FPGA. Proceedings - 2003 IEEE International Conference on Field-Programmable Technology, FPT 2003, 76–83. https://doi.org/10.1109/FPT.2003.1275734

[22] Fu, H., Mencer, O., & Luk, W. (2010). FPGA designs with optimized logarithmic arithmetic. IEEE Transactions on Computers, 59(7), 1000–1006. https://doi.org/10.1109/TC.2010.51

[23] Arnold, M. G., Bailey, T. A., Cowles, J. R., & Winkel, M. D. (1998). Arithmetic Co-transformations in the real and complex logarithmic number systems. IEEE Transactions on Computers, 47(7), 777–786. https://doi.org/10.1109/12.709377

[24] P. Vouzis, C. Collange and M. Arnold, "LNS Subtraction Using Novel Cotransformation and/or Interpolation," 2007 IEEE International Conf. on Application-specific Systems, Architectures and Processors (ASAP), Montreal, QC, Canada, 2007, pp. 107-114, doi: 10.1109/ASAP.2007.4429966.

[25] Vouzis, P. D., Collange, S., & Arnold, M. G. (2007). Cotransformation provides area and accuracy improvement in an HDL library for LNS subtraction. Proceedings - 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools, DSD 2007, 85–93. https://doi.org/10.1109/DSD.2007.4341454

[26] Naziri, S. Z. M., Ismail, R. C., Isa, M. N. M., & Hussin, R. (2021). Less memory and high accuracy logarithmic number system architecture for arithmetic operations. Indonesian Journal of Electrical Engineering and Computer Science, 23(3), 1708–1717. https://doi.org/10.11591/ijeecs.v23.i3.pp1708-1717

[27] Taylor, F. J. (1985). A Hybrid Floating-Point Logarithmic Number System Processor. IEEE Transactions on Circuits and Systems, 32(1), 92–95. https://doi.org/10.1109/TCS.1985.1085588

[28] Stouraitis, T. (1989). Hybrid floating-point/logarithmic number system digital signal processor. ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings, 2, 1079–1082. https://doi.org/10.1109/ICASSP.1989.266619

[29] Lai, F. S., & Eric Wu, C. F. (1991). A Hybrid Number System Processor with Geometric and Complex Arithmetic Capabilities. IEEE Transactions on Computers, 40(8), 952–962. https://doi.org/10.1109/12.83639

[30] Arnold, M. G., & Collange, S. (2014). Options for Denormal Representation in Logarithmic Arithmetic. Journal of Signal Processing Systems, 77(1–2), 207–220. https://doi.org/10.1007/S11265-014-0874-3

[31] Basir, M. S. S. M., Ismail, R. C., & Naziri, S. Z. M. (2020). An Investigation of Extended Co-Transformation using Second-Degree Interpolation for Logarithmic Number System. Proceeding - 1st FORTEI-International Conference on Electrical Engineering, FORTEI-ICEE 2020, 59–63. https://doi.org/10.1109/FORTEI-ICEE50915.2020.9249931

[32] Gautschi, M., Schaffner, M., Gürkaynak, F. K., & Benini, L. (2017). An Extended Shared Logarithmic Unit for Nonlinear Function Kernel Acceleration in a 65-nm CMOS Multicore Cluster. IEEE Journal of Solid-State Circuits, 52(1), 98–112. https://doi.org/10.1109/JSSC.2016.2626272

[33] Taek-Jun Kwon, Sondeen, J., & Draper, J. (2005). Design Trade-Offs in Floating-Point Unit Implementation for Embedded and Processing-In-Memory Systems. In 2005 IEEE International Symposium on Circuits and Systems (pp. 3331–3334). IEEE. https://doi.org/10.1109/ISCAS.2005.1465341

[34] Lamaire, R. O., & Lang, J. H. (1986). Performance of Digital Linear Regulators Which Use Logarithmic Arithmetic. IEEE Transactions on Automatic Control, 31(5), 394–400. https://doi.org/10.1109/TAC.1986.1104309

[35] Kumar, C. S., Prathiba, A., & Kanchana Bhaskaran, V. S. (2016). Implementation of RNS and LNS based addition and subtraction units for cryptography. 2016 International Conference on VLSI Systems, Architectures, Technology and Applications, VLSI-SATA 2016. https://doi.org/10.1109/VLSI-SATA.2016.7593055

[36] Kurokawa, T., Payne, J. A., & Lee, S. C. (1980). Error Analysis of Recursive Digital Filters Implemented with Logarithmic Number Systems. IEEE Transactions on Acoustics, Speech, and Signal Processing, 28(6), 706–715. https://doi.org/10.1109/TASSP.1980.1163466

[37] Morley, R. E., Engel, G. L., Sullivan, T. J., & Natarajan, S. M. (1988). VLSI BASED DESIGN OF A BATTERY-OPERATED DIGITAL HEARING AID. ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings, 2512–2515. https://doi.org/10.1109/ICASSP.1988.197154

[38] Kwa, S. W., Engel, G. L., & Morley, R. E. (2000). Quantization noise analysis of sign/logarithm data encoders when excited by speech or sinusoidal inputs. IEEE Transactions on Signal Processing, 48(12), 3578–3581. https://doi.org/10.1109/78.887052

[39] Swartzlander, E. E., Chandra, D. V. S., Nagle, H. T., & Starks, S. A. (1983). Sign/Logarithm Arithmetic for FFT Implementation. IEEE Transactions on Computers, C–32(6), 526–534. https://doi.org/10.1109/TC.1983.1676274

[40] Hongyuan, W., & Lee, S. C. (1986). Comments on "Sign/Logarithm Arithmetic for FFT Implementation." IEEE Transactions on Computers, C–35(5), 482–484. https://doi.org/10.1109/TC.1986.1676792

[41] Arnold, M. G., & Walter, C. (2001). Unrestricted faithful rounding is good enough for some LNS applications. Proceedings - Symposium on Computer Arithmetic, 237–246. https://doi.org/10.1109/ARITH.2001.930125

[42] Brokalakis, A., & Paliouras, V. (2011). Using the arithmetic representation properties of data to reduce the area and power consumption of FFT circuits for wireless OFDM systems. 2011 IEEE Workshop on Signal Processing Systems, SiPS 2011, Proceedings, 7–12. https://doi.org/10.1109/SIPS.2011.6088941

[43] Kang, B., Vijaykrishnan, N., Irwin, M. J., & Theocharides, T. (2004). Power-efficient implementation of turbo decoder in SDR system. Proceedings - IEEE International SOC Conference, 119–122. https://doi.org/10.1109/SOCC.2004.1362372

[44] Wang, H., Yang, H., & Yang, D. (2006). Improved Log-MAP decoding algorithm for turbo-like codes. IEEE Communications Letters, 10(3), 186–188. https://doi.org/10.1109/LCOMM.2006.1603379

[45] Arnold, M. G. (2002). Reduced power consumption for MPEG decoding with LNS. Proceedings of the International Conference on Application-Specific Systems, Architectures and Processors, 2002-January, 65–75. https://doi.org/10.1109/ASAP.2002.1030705

[46] Lee, P. (2005). An evaluation of a hybrid-logarithmic number system DCT/IDCT algorithm. Proceedings - IEEE International Symposium on Circuits and Systems, 4863–4866. https://doi.org/10.1109/ISCAS.2005.1465722

[47] Arnold, M. G. (2004). Redundant logarithmic arithmetic for MPEG decoding. Https://Doi.Org/10.1117/12.559080, 5559, 112–122. https://doi.org/10.1117/12.559080

[48] Lai, F. S., & Eric Wu, C. F. (1991). A Hybrid Number System Processor with Geometric and Complex Arithmetic Capabilities. IEEE Transactions on Computers, 40(8), 952–962. https://doi.org/10.1109/12.83639

[49] Nam, B. G., & Yoo, H. J. (2007). A low-power vector processor using logarithmic arithmetic for handheld 3D graphics systems. ESSCIRC 2007 - Proceedings of the 33rd European Solid-State Circuits Conference, 232–235. https://doi.org/10.1109/ESSCIRC.2007.4430286

[50] Nam, B. G., Kim, H., & Yoo, H. J. (2007). A low-power unified arithmetic unit for programmable handheld 3-D graphics systems. IEEE Journal of Solid-State Circuits, 42(8), 1767–1778. https://doi.org/10.1109/JSSC.2007.900243

[51] Zhang, W., Geng, X., Wang, Q., Han, J., & Jiang, H. (2024). A Low-Power and High-Accuracy Approximate Adder for Logarithmic Number System. Proceedings of the ACM Great Lakes Symposium on VLSI, GLSVLSI, 125–131. https://doi.org/10.1145/3649476.3658706