mathematics

MDPI

Article

RISC Conversions for LNS Arithmetic in Embedded Systems

Peter Drahoš *, Michal Kocúr, Oto Haffner, Erik Kučera and Alena Kozáková

Institute of Automotive Mechatronics, Faculty of Electrical Engineering and Information Technology, Slovak University of Technology in Bratislava, 812 19 Bratislava, Slovakia; michal.kocur@stuba.sk (M.K.); oto.haffner@stuba.sk (O.H.); erik.kucera@stuba.sk (E.K.); alena.kozakova@stuba.sk (A.K.)

* Correspondence: peter.drahos@stuba.sk

Received: 19 June 2020; Accepted: 20 July 2020; Published: 22 July 2020

Abstract: The paper presents an original methodology for the implementation of the Logarithmic Number System (LNS) arithmetic, which uses Reduced Instruction Set Computing (RISC). The core of the proposed method is a newly developed algorithm for conversion between LNS and the floating point (FLP) representations named "looping in sectors", which brings about reduced memory consumption without a loss of accuracy. The resulting effective RISC conversions use only elementary computer operations without the need to employ multiplication, division, or other functions. Verification of the new concept and related developed algorithms for conversion between the LNS and the FLP representations was realized on Field Programmable Gate Arrays (FPGA), and the conversion accuracy was evaluated via simulation. Using the proposed method, a maximum relative conversion error of less than ±0.001% was achieved with a 22-ns delay and a total of 50 slices of FPGA consumed including memory cells. Promising applications of the proposed method are in embedded systems that are expanding into increasingly demanding applications, such as camera systems, lidars and 2D/3D image processing, neural networks, car control units, autonomous control systems that require more computing power, etc. In embedded systems for real-time control, the developed conversion algorithm can appear in two forms: as RISC conversions or as a simple RISC-based logarithmic addition.

Keywords: LNS; numerical conversion; RISC; FPGA; embedded systems

1. Introduction

The Logarithmic Number System (LNS) provides comparable range and precision as the floating point (FLP) representation, however—for certain applications—it can surpass it in terms of complexity. The range of logarithmic numbers depends on the exponent's integer part, and the precision is defined by its fraction part [1]. Yet, LNS would outperform FLP only if the logarithmic addition and subtraction can be performed with at least the same speed and accuracy as FLP [2].

The long history of LNS numbers dates back to the 1970's when the "logarithmic arithmetic" for digital signal processing was introduced [3]. To avoid negative logarithms, a complementary notation for LNS was introduced in [4]. Architecture for the LNS-based processor was proposed in [5]. Implementations of basic arithmetic operations on FPGA [6] using FLP and LNS have shown that the multiplication and division operations are more effective if using LNS, as they require fewer area resources and have a significantly lower time latency. Otherwise, addition and subtraction are more suitable using FLP representation. A higher efficiency of some LNS operations was a motivation for using the LNS format for the realization of control algorithms for the autonomous electric vehicle developed within a running research project.

Nowadays, LNS representation is implemented in various applications, such as deep-learning networks [7], Cartesian to polar coordinates converters [8], or embedded model predictive control [9].

In a very interesting paper [10], it is proposed how numbers close to zero can be represented in the denormal LNS method (DLNS) using either fixed-point or LNS representations, guaranteeing constant absolute or constant relative precisions, respectively. Up to now, LNS have not been standardized.

Various methods have been developed to decrease the costs and complexity of LNS implementation, e.g., interpolation, co-transformation, multipartite tables, etc. [11–13]. Typically, there are three main categories of LNS arithmetic techniques: lookup tables, piecewise polynomial approximation, and digit–serial methods [14].

Generally, LNS addition and subtraction are carried out based on the evaluation of the transcendental functions as follows:

$$a \pm_{LNS} \; b = \log_2\left(2^a \pm 2^b\right) = a + \log_2\left(1 \pm 2^{b-a}\right), \tag{1}$$

where a, b are logarithmic numbers.

Signal processing in embedded systems based on LNS has three stages: logarithmic conversions, simple operations, and antilogarithmic conversions. In the first processing stage, logarithmic conversions are applied to convert binary numbers into logarithmic ones. In the second stage, simple operations are used to perform corresponding calculations, such as addition and subtraction. In the last stage, logarithmic numbers are converted back to binary ones. There are many approaches to solving logarithmic conversion that can be classified into three categories: memory-based methods [15,16], mathematical approximations [6], and shift-and-add-based methods [11,13,17–20]. Very fast conversions (e.g., shift-and-add) allow us to combine calculations in LNS and FLP systems and design hybrid LNS/FLP processors [13,21,22]. In hybrid systems, the conversions are carried out several times during the calculation, not only the first and last phases.

Using memory-based methods, fast and more accurate conversions are achieved; however, memory size costs may increase significantly while the bit-width of the inputs increases. On the other hand, using polynomial approximations will reduce the area costs, while sacrificing the accuracy and speed. Approximation-based methods almost always use a multiplication operation, for example, an antilogarithmic converter [6] uses 20 multipliers and achieves a latency of more than 70 ns. Compared with these two kinds of implementation, shift-and-add methods can be used to achieve better design tradeoffs between accuracy, memory costs, and speed. All the above-mentioned "shift-and-add" methods achieve a latency of less than 1 ns at the cost of a low accuracy (above 1% relative error [20]) except for [23], where the attained accuracy of the LNS/FLP conversions is 0.138%. Using the proposed looping-in-sectors method in combination with a very simple approximation based on bit manipulations, a radical increase in accuracy and an acceptably low latency can be achieved.

The logarithmic and antilogarithmic conversions are a gateway to LNS algorithms. Yet, the conversions are not freely available from FPGA producers [24] and, thus, must be implemented by our own means. Furthermore, the above methods available in the literature do not meet our requirements in that the accuracy of modern industrial sensors is better than 0.1%, and sampling periods in embedded systems for motion control are less than 1 ms, which places high demands on conversion speed and application calculations. Therefore, our motivation was to develop a simple and efficient FLP/LNS conversion guaranteeing sufficient accuracy and speed, which will be a "golden mean" between the accurate but complex approximation methods and the very efficient and fast (up to 1ns) but inaccurate (relative error higher than 1%) shift-and-add methods.

Embedded systems are expanding into increasingly demanding applications, such as camera systems, lidars and 2D/3D image processing, neural networks, car control units, autonomous control systems that require more computing power, etc. A necessary reliability and functional safety are often based on redundancy of two/three channel technologies. Therefore, alternative calculations (one option is LNS) on independent HW and SW (hardware and software) solutions are needed; outputs of the

independent channels are then compared according to the principles of fault tolerant systems (e.g., two out of three).

However, embedded control systems based on LNS arithmetic that operate in real-time necessitate an efficient conversion in every sampling period. Input data from sensors, counters, A/D converters, etc., are typically fixed-point (FXP) numbers, thus they have to be converted to LNS, and, the other way round, the LNS arithmetic results are to be converted back to the FXP format conventionally required by typical output devices (actuators). In this paper, the focus is on conversions between LNS and FLP, conversions from FXP to FLP and back are supposed to be resolved.

The Xilinx's industry-leading tool suite natively supports different FLP precisions including half (FLP16), single (FLP32), and double (FLP64) precisions, as well as fixed-point data types. The added flexibility of custom precision is also available in MATLAB System Generator for DSP toolbox. The FLP to FXP conversion is dealt with in [24], however the LNS data type is not yet officially supported and conversions from LNS to FLP and back are not available in FPGA libraries.

This paper presents an application of the proposed RISC conversions for logarithmic addition using the Reduced Instruction Set Computing (RISC) realizable just by means of simple operations without using multiplication, etc. Herein, RISC indicates a set of simple computer operations (add, minus, shift by 2, i.e., multiplication/division, logical operations, and bit manipulations). The proposed approach has the ambition to apply just the above-mentioned RISC operations fully excluding multiplication, division, and all other functions (log, square, ...). Using the unified format of LNS and FLP, conversion between them can be realized only by dealing with the mantissa and the fraction. To reduce memory requirements for conversions, a novel method called "looping in sectors" was developed.

The paper presents a novel effective RISC-based method, which uses the so-called "looping-in-sectors" procedure and a simple interpolation in the conversion between FLP and LNS number representations. The novel algorithm of logarithmic addition based on the developed conversions performs differently from previously known approaches. The partial results on the development of RISC conversions and algorithms for LNS [25] are completed by the conversion algorithm from LNS to FLP and its realization on FPGA.

The paper is organized as follows. In Section 2, an overview of FLP and LNS number representations is provided. Section 3 presents two developed algorithms of the RISC conversion between both systems. A simple interpolation method along with accuracy analysis are dealt with in Section 4. Principle of the RISC-based LNS addition is explained in Section 5. FPGA realization of the RISC conversion is demonstrated on a simple example in Section 6. Discussion on obtained results, their potential, and future research concludes the paper.

2. Number Systems

Let us briefly revisit the FLP and LNS number representations. According to Table 1, a floating point (FLP) number is expressed as follows:

$$FLP = (-1)^S \times 2^E \times (1+m) = (-1)^S \times 2^E \times \left(1 + \frac{N}{M}\right), \tag{2}$$

where m is a mantissa, and N is an integer or a real number from the intervals $\langle 0, M-1\rangle$ or $\langle 0, M\rangle$, respectively. M is the maximum of the mantissa (fractional part) with t bits.

$$M = 2^t \tag{3}$$

Table 1. Floating point (FLP) number representation.

Sign Bit	Exponent: e-Bits	Mantissa: t-Bits
S	E	$m = N/M$

Table 2 shows the principle of an LNS number representation; according to it:

$$LNS = (-1)^S \times 2^{E_f} = (-1)^S \times 2^E \times 2^f. \quad (4)$$

The logarithmic fraction f can be expressed as follows:

$$f = F/M, \quad (5)$$

where F is an integer in the range $\langle 1 - M,\ M - 1\rangle$ or a real number in the range $(-M,\ M)$.

Table 2. Logarithmic number (LNS) representation.

Sign Bit	Integer: e-Bits	Fractional: t-Bits
S	E	$f = F/M$

In terms of individual bits, the whole exponent E_f consists of "integer bits" (i_x) and "fractional bits" (f_y), placed next to each other. S_E denotes the sign of the exponent.

$$E_f = S_E i_{e-2\ldots} i_1 i_0 \quad f_{t-1\ldots} f_1 f_0, \quad (6)$$

For both numerical systems, the number of bits of the exponent E corresponds to the range of the numbers, and the number of the fractional part bits reflects the accuracy.

2.1. Two Possible Representations of LNS Numbers

The mantissa m is always a positive number ($0 \le m < 1$), but the logarithmic fraction depends on the sign of the exponent S_E. Still, there is also another possibility to represent LNS fraction as always positive, similar to mantissas.

Let a number $X < 1$, $E \le 0$ and a fraction $f < 0$. F_{SE} and F_{AP} are positive real numbers from $\langle 0, \mathrm{M}\rangle$.

$$X = (-1)^S \times 2^E \times 2^{-\frac{F_{SE}}{M}} = (-1)^S \times 2^{E-1} \times 2^{\frac{F_{AP}}{M}}, \quad (7)$$

where F_{AP} is a complement of F_{SE} to the range of the fraction M, i.e.,

$$F_{AP} = M - F_{SE} \quad (8)$$

For numbers $X > 1$, $F_{AP} = F_{SE}$ and the integer E is unchanged. F_{AP} is an always-positive fraction.

The sign of F_{SE} is the same as the sign S_E. For the sake of completeness note that for $X = 1$ there are two possible ways (i.e., possible codes) to represent zero. In the F_{SE} representation, $E = \pm 0$ and the fraction $F_{SE} = 0$ (the same as for $X = -1$). In the F_{AP} representation there is no such anomaly; the conversion between F_{AP} and F_{SE} proceeds (7) and (8).

2.2. Equivalence between FLP and LNS

The FLP (2) and LNS (4) representations are equivalent if integer parts of both representations are equal numbers of e-bits, and both the fraction and the mantissa are equal numbers of t-bits. It is also necessary to use an always positive fraction F_{AP}. The sign S and the exponent E are matching:

$$X = (-1)^S \times 2^E \times 2^{\frac{L_X}{M}} = (-1)^S \times 2^E \times \left(1 + \frac{N_X}{M}\right). \quad (9)$$

Let N_X denote the mantissa (FLP) and $L_X = F_{AP}$ is the always positive fraction (LNS). The subscript "x" specifies that they represent (code) the equivalent number X in diverse number systems. Using the

following conversion between the mantissa and the fraction, equivalence of FLP and LNS can be attained:

$$Z = 2^{\frac{L_Z}{M}} = \left(1 + \frac{N_Z}{M}\right), \tag{10}$$

where $Z \in \langle 1,\ 2\rangle$ and L_Z, N_Z are positive integers from the interval $\langle 0,\ M-1\rangle$ or positive real numbers within the interval $\langle 0, M\rangle$. In the same range, sequences of integers for L and N are geometric and arithmetic, respectively. The integer form of L and N is used to address the look up table (LUT) memory, while their real form is needed to attain a required accuracy. By extending the number of bits of the lower fraction and mantissa to $t + r$ bits, the accuracy can be improved. Using the following corrections, the mutual number conversions over the interval $\langle 0,\ M\rangle$ can be defined as follows:

$$L_Z = N_z + C_{NZ}(N_Z), \tag{11}$$

$$N_z = L_Z - C_{LZ}(L_Z), \tag{12}$$

where C_{NZ} and C_{LZ} are correction functions for conversions in both directions:

$$C_{NZ}(N_Z) = M \times \log_2\left(1 + \frac{N_Z}{M}\right) - N_Z, \tag{13}$$

$$C_{LZ}(L_Z) = -M \times \left(2^{\frac{L_Z}{M}} - 1\right) + L_Z. \tag{14}$$

From the corresponding diagrams in Figure 1 it is evident that both functions have the same maximum, however at various arguments:

$$\max_{NZ} C_{NZ}(453.319721) = \max_{LZ} C_{LZ}(541.456765) = 88.137044\ .$$

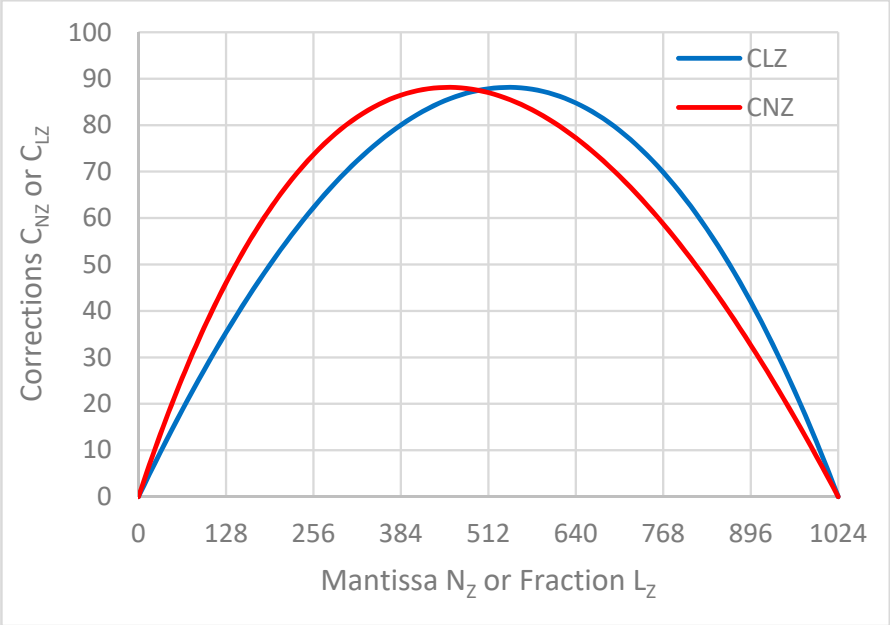

Figure 1. Diagram of correction functions from N_Z to L_Z (red plot) and from L_Z to N_Z (blue plot) for $M = 1024$.

3. RISC-Based Conversion between FLP and LNS

If we assume equivalence of the FLP and LNS numbers, the conversion can be completed between the fraction and mantissa. The proposed algorithm aims to use RISC-type computing operations to reduce memory consumption and costs, while achieving as high accuracy as possible. In the Reduced

Instruction Set Computing (RISC), operations of multiplication, division, square, square root, logarithm, etc., are not used.

3.1. Conversion from LNS to FLP

The conversion from LNS to FLP is carried out using (12) and (14). When converting L_Z it is required to cover the whole range of the fraction, guarantee a sufficient accuracy, and avoid a large memory consumption. According to the proposed approach, L_Z is split in two parts:

$$L_Z = L_S + L_B, \tag{15}$$

where L_S is relevant for each sector and L_B for the common base. The fraction is split into sectors (e.g., 32 sectors in our case). In (15), L_S represents a relocation of the converted number to the base L_B —a part of the C_{LZ} function diagram placed in the close vicinity of the extreme (Figure 2).

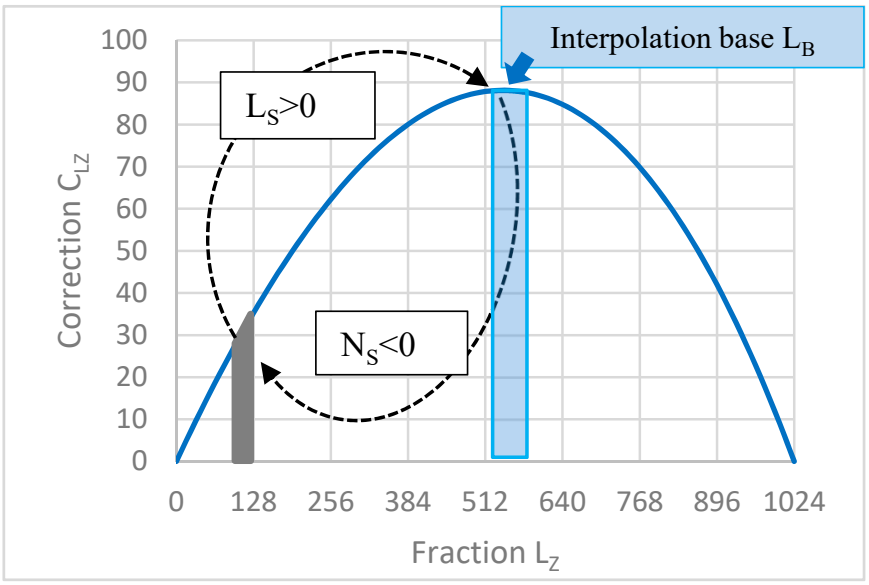

Figure 2. Looping in sectors: conversion from L_Z to N_Z for $M = 1024$.

To obtain N_B from L_B, a simple interpolation is performed (as described in Section 5). The calculation loop closes by applying *Ns* in (18). This procedure was named "looping in sectors" (LiS) by the first author. Mathematically, the LiS method is expressed by the Equations (16)–(18):

$$Z = 2^{\frac{L_Z}{M}} = 2^{\frac{L_S}{M}} \times 2^{\frac{L_B}{M}}. \tag{16}$$

The equivalent FLP representation in the mantissa form is as follows:

$$Z = \left(1 + \frac{N_Z}{M}\right) = \left(1 + \frac{N_S}{M}\right) \times \left(1 + \frac{N_B}{M}\right). \tag{17}$$

From (17) results:

$$N_Z = N_B + N_S + \frac{N_S}{M} \times N_B. \tag{18}$$

By appropriately choosing *Ns* and calculating corresponding *Ls* according to (11), a conversion look-up table LUT1LS was generated using 10 bits, i.e., $M = 1024$ (Table 3). For each sector, the numbers *Ns* and *Ls* form a pair when using the looping-in-sectors method.

Table 3. Look-up table LUT1LS for 32 sectors.

Min Lz	Max Lz	Address	Ns	Ls
0	31	0	−304	520.3424
32	63	1	−290	491.8925
...	...	...	...	...
480	511	15	−32	46.9030
512	543	16	−8	11.5869
544	575	17	12	−17.2117
576	607	18	36	−51.0449
...	...	...	...	...
960	991	30	352	−436.4951
992	1023	31	380	−466.2551

The procedure for selecting the sector number *Ns* is as follows. Choose the interval for the "interpolation base L_B" in the vicinity of the argument (541.456765) of the extreme of the correction function in Figure 2. In our case, the minimum interval is min L_B = 520. Each sector has a minimum, denoted "min *Lz*" in Table 3. We choose *Ns* with three active bits (Table 4) under the condition *Ls* > (520-min*Lz*) where *Ls* is calculated according to (11) with the required accuracy. The choice of *Ns* is specific in each sector; as a result the interval for the interpolation base is not 32 but 40 (wider) where $L_B \in (520,\ 560)$.

The RISC conversion procedure from L to N is described in the Algorithm 1 (input variable is the fraction L_Z; the sign *S* and the exponent *E* do not change):

Algorithm 1 RISC conversion from L to N

1. Input: LNS fraction LZ
2. Output: FLP mantissa N_Z
3. Round L_Z to the upper 5 bits = address (cut the lower bits).
4. Read L_S and N_S from the memory LUT1LS (Table 3), where 5 MSB of L_Z represent the LUT1S address.
5. $L_B = L_Z + L_S$.
6. Read N_{B0} and N_{B1} from the memory LUT2NB, where the 9 MSB of L_B represent the LUT2NB address.
7. Calculate N_B using the interpolation between N_{B0} and N_{B1} (see the next Section).
8. Calculate N_Z using bit shifting (19).

In the above Algorithm 1, N_Z is calculated as follows:

$$N_z = N_B + N_S \pm shift1\ (N_B) \pm shift2\ (N_B) \pm shift3\ (N_B) \tag{19}$$

where *shift1*, *shift2*, *shift3* denote shifting bits of N_B according to Table 4, which is suitable for FPGA implementation (it substitutes the original product N_B * N_S/M). In (18), we modified the expression N_S/M to 3 "shift operations" according to the weights (*w* bits) of the 3 active bits (Table 4). $M = 2^{10}$ represents a shift by 10 places to the right.

Table 4. Generating *Ns* using 3 active bits in 32 sectors.

|Ns|\w Bits	8	7	6	5	4	3	2	1
304	1	0	1	0	−1	0	0	0
290	1	0	0	1	0	0	0	1
...	...	...	...	...	...	...	...	...
32	0	0	0	1	0	0	0	0
8	0	0	0	0	0	1	0	0
12	0	0	0	0	0	1	1	0
36	0	0	0	1	0	0	1	0
...	...	...	...	...	...	...	...	...
352	1	1	0	−1	0	0	0	0
380	1	1	0	0	0	0	−1	0

The variable N_S was chosen to include just 3 "active bits", i.e., bits with nonzero (+1 or −1) values, see Table 4. The algorithm results that the conversion can be performed as RISC, i.e., using just elementary computer operations: addition, subtraction, shifting (multiplication and division by 2), and addressing/reading from the memory.

3.2. Conversion from FLP to LNS

Conversion from FLP to LNS is based on Equations (11) and (13) and modified Equations (16)–(18). Details are described in [25].

The RISC conversion from N to L (input variable is the mantissa N_Z; the sign S and the exponent E do not change) is described in Algorithm 2:

Algorithm 2 RISC conversion from N to L

1. Input: FLP mantissa NZ
2. Output: LNS fraction L_Z
3. Round N_Z to the upper 5 bits = address (cut the lower bits).
4. Read L_S and N_S from the memory LUT1LS (Table 3), where 5 MSB of N_Z are the memory address.
5. Calculate N_B according to (20).
6. Read L_{B0} and L_{B1} from the memory.
7. Calculate L_B using the interpolation between L_{B0} and L_{B1}.
8. Calculate $L_Z = L_B + L_S$.

In the above Algorithm 2, N_B is calculated as follows:

$$N_B = N_Z + N_S \pm shift1\ (N_Z) \pm shift2\ (N_Z) \pm shift3\ (N_Z). \tag{20}$$

Note that (20) is a modification of (18). Again, N_S has to be chosen to include just 3 "active bits". It can be concluded that both conversions from FLP to LNS and vice versa can be performed as RISC, using just elementary computer operations.

4. Interpolation and Accuracy

For both FLP and LNS, the accuracy is given by the number of bits of the mantissa and the fraction; it usually decreases due to approximation or interpolation. In the effective RISC conversion design, a simple interpolation is based on bit manipulation. The interpolation is demonstrated on the LNS to FLP conversion. The base generated according to (12) and (14) and the corresponding look-up table LUT2NB were chosen so that the base falls within the plateau in the close vicinity of the C_{LZ} function maximum (Figure 2).

The plateau region was intentionally chosen as the interpolation base because differences between the "adjacent" logarithmic values of L_B and the arithmetic sequence values of N_B are approximately matching; this allows us to interpolate between two points in the table in a very simple way (Figure 3).

ADDRESS			**Number/MEMORY**
Higher bits	**Lower bits**	**Operation**	**All bits**
$L_{B1} = L_{B0}+1$	000...000	MEM ⇨	N_{B1}
$L_B = L_{B0} +$	**$+ LL_B$**	APROX.	**$N_B = N_{B0} + LL_B + ...$**
L_{B0}	000...000	MEM ⇨	N_{B0}

Figure 3. Principle of approximating N_B using a simple bit manipulation.

Approximating the mantissa N_B (denoted as the interpolation base L_B in Figure 2) using the look-up table LUT2NB is demonstrated in Figure 3:

- The fraction L_B is rounded to the higher n bits (e.g., n = 10), thus obtaining the address L_{B0}. Then, N_{B0} is read from the memory.
- Lower bits of L_B are denoted as LL_B:

$$L_B = L_{B0} + LL_B. \tag{21}$$

- The simplest approximation is the "quasilinear" interpolation as follows:

$$N_B = N_{B0} + LL_B. \tag{22}$$

According to this method, the error dN_{B01} is calculated as a difference of adjacent memory locations in the considered memory LUT2NB:

$$dN_{B01} = N_{B1} - N_{B0} - 1. \tag{23}$$

In case of a 10-bit memory quantization in the range $L_B \in (520, 560)$, the maximum positive error is 1.1988×10^{-5} ($dN_{B01} = 1.2275 \times 10^{-2}$) at the right limit of the interval, and the maximum negative error is -1.3742×10^{-5} ($dN_{B01} = -1.4072 \times 10^{-2}$) at the left limit, which corresponds to an accuracy of 16 bits (10 + 6). These errors of the variable N_B will then be influenced by a factor ($1 + N_S/M$) in the range $\langle 0.70312,\ 1.37109 \rangle$, according to (18). The resulting error analysis is in Section 6. The accuracy can be improved using a finer memory quantization in the plateau region. Note that the above accuracy levels are attained using RISC, i.e., multiplication is not used.

If a higher accuracy is needed, the following linear interpolation can be applied:

$$N_B = N_{B0} + LL_B + dN_{B01} \times LL_B. \tag{24}$$

In the memory region LUT2NB with 40 cells where $L_B \in (520, 560)$, the approximation error occurs due to nonlinearity. Local extremes between adjacent memory cells (approximately in the middle of them) were examined. For a 10-bit memory quantization, the related accuracy is 23 bits (10 + 13). The proposed linear interpolation is a RISC extended by one multiplication operation.

Finally, it has to be noted that the FLP to LNS interpolation procedure is similar, only based on Equations (11) and (13). Details are described in [25]. Accuracy assessment in terms of memory quantization is the same.

5. Application of RISC Conversions for Logarithmic Addition

The principle of the logarithmic addition based on the developed RISC conversions is briefly presented in this section. The aim is to show the possibility of applying the developed conversions for RISC-based LNS addition without additional memory requirements. Similar algorithms developed for data conversion at the input and output of the embedded system can be used for the LNS adder.

Consider two real numbers A, B, where $A > B$ represented in LNS using integer exponents E_A, E_B, and fractions L_A, L_B, respectively:

$$A + B = 2^{E_A} \times 2^{\frac{L_A}{M}} + 2^{E_B} \times 2^{\frac{L_B}{M}}. \tag{25}$$

The proposed operation of logarithmic addition will be demonstrated under the assumptions $E_A \geq E_B$ and $L_A \geq L_B$. Applying the distributive law, we obtain:

$$A + B = 2^{E_A} \times 2^{\frac{L_B}{M}} \times \left(2^{\frac{L_A - L_B}{M}} + \frac{2^t}{M} \times 2^{-(E_A - E_B)} \right). \tag{26}$$

Assume $E_A > E_B$ and $L_A < L_B$, we obtain:

$$A + B = 2^{E_A - 1} \times 2^{\frac{L_B}{M}} \times \left(2^{\frac{M + L_A - L_B}{M}} + \frac{2^t}{M} \times 2^{-(E_A - 1 - E_B)} \right). \tag{27}$$

Denote L_{AB}, d and E_{AS} as follows:

$$L_{AB} = L_A - L_B \quad or \quad L_{AB} = M + L_A - L_B \tag{28}$$

$$d = E_A - E_B \quad or \quad d = E_A - E_B - 1 \tag{29}$$

$$E_{AS} = E_A \quad or \quad E_{AS} = E_A - 1\,. \tag{30}$$

Then from both assumptions results:

$$A + B = 2^{E_{AS}} \times 2^{\frac{L_B}{M}} \times \left(2^{\frac{L_{AB}}{M}} + \frac{2^{t-d}}{M} \right). \tag{31}$$

The RISC-based conversion of the fraction $L_{AB} = L_A - L_B$ to the mantissa N_{AB} is carried out using (12). Then, $1 + N_{AB}/M$ has a mantissa format; $M = 2^t$ can be considered as a mantissa or a fraction, as needed. For M, the corrections (13), (14) are zero, i.e., no conversion is required. Dividing M by powers of 2^d, i.e., applying the shifting by d, we obtain a result, which has the range and character of a mantissa; let us denote it $N_E = 2^{t-d}$, then

$$A + B = 2^{E_{AS}} \times 2^{\frac{L_B}{M}} \times \left(1 + \frac{N_{AB}}{M} + \frac{N_E}{M} \right), \tag{32}$$

where the two rightmost terms in the expression in parentheses have a mantissa format and can be simply summed:

$$N_{ABE} = N_{AB} + N_E. \tag{33}$$

If is $N_{ABE} \geq M$ (overflow), then

$$N_{ABE} = (N_{ABE} - M)/2 \quad and \quad E_{AS} = E_{AS} + 1. \tag{34}$$

Applying the RISC conversion of N_{ABE} to L_{ABE} according to (11) we obtain:

$$A + B = 2^{E_{AS}} \times 2^{\frac{L_B}{M}} \times 2^{\frac{L_{ABE}}{M}} = 2^{E_{SUM}} \times 2^{\frac{L_{SUM}}{M}}, \tag{35}$$

which is a logarithmic sum of the original numbers A, B.

When implemented, the overflow $L_{ABE} + L_B$ of the range $\langle 0,\ M - 1 \rangle$ by the fraction L has to be treated. The integer exponent of the sum E_{SUM} can take two values: either E_{AS} or E_{AS} + 1 (under overflown).

$$L_{SUM} = L_{ABE} + L_B \quad or \quad L_{SUM} = L_{ABE} + L_B - M \tag{36}$$

The above-presented original procedure of LNS addition is based solely on RISC-type operations including two RISC conversions that determine the accuracy of the adder. The LNS adder can be implemented using the six standard additions or subtractions (28)–(30), (33), (34) and (36), by comparing four pairs of numbers, two shifting operations, and two RISC conversions. The developed approach is promising for applications in embedded control systems realized, e.g., on FPGA [26]. More details on LNS addition and subtraction via RISC computing can be found in [25].

6. Implementation on FPGA

To verify the proposed conversion algorithm, a hardware realization of a simple converter from LNS to FLP number representations was realized. The provided example demonstrates the feasibility of the proposed approach and the solution accuracy.

6.1. Design of an LNS to FLP Converter

The implemented LNS input of the converter represented according to Table 2 has a 32-bit data width. The most significant bit is assigned to the sign, the next 8 bits belong to the integer part, and the last 23 significant bits are the fractional part of the number. The convertor output is a 32-bit FLP number format compatible with the IEEE 754 standard (Table 1).

The LNS to FLP converter is realized as a digital logic circuit and contains only combinational logic. The converter design was developed in VHDL (Very High-Speed Integrated Circuit Hardware Description Language) using Vivado IDE and was targeted for the FPGA (Field Programmable Gate Array) chip Xilinx Artix-7 XC7A100T-1CSG324C mounted on Nexys 4 trainer board [27]. FPGAs are a powerful tool for prototyping and testing hardware designs based on digital logic. FPGAs have broad resources for implementation of combinational and sequential logic, which allows us to implement even more complex and tailored digital designs.

The structure of the LNS to FLP convertor represented by RTL (Register Transfer Level) is shown in Figure 4. The design is based on the algorithm of RISC conversion from LNS to FLP as described in Section 3. The sign bit and the integer part of the LNS input are directly connected to the sign and exponent parts of the FPL output. The fractional part of the input signal is marked as L_Z. The five MSB bits of L_Z are the input for Table 3, LUT1LS (Address column). L_B described in (15) is computed as a sum of LUT1LS output L_S and the signal L_Z. In this converter design, only a simple approximation according to (22) is realized. L_B is calculated as the sum of N_{B0} and LL_B. The mantissa N_Z (18) as a part of the output FLP number is computed as the sum of three signals: N_B, N_S, and output product of N_B and N_S. Division in (18) is realized by a proper choice of the output product bits.

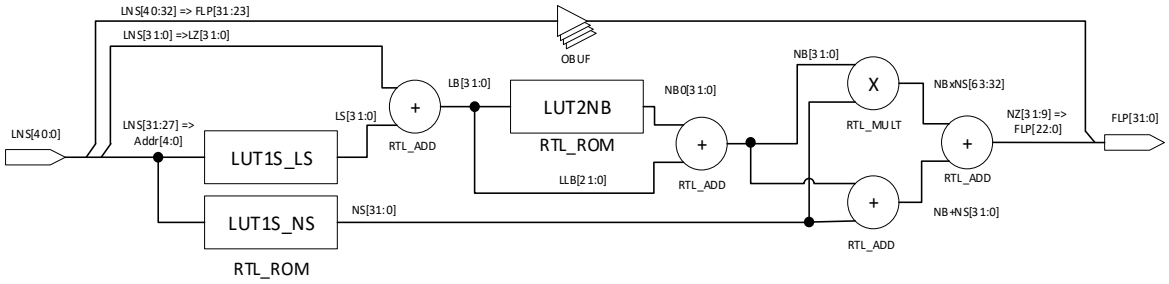

Figure 4. Illustration of the principle of approximating N_B using a simple interpolation.

Both tables LUT1LS and LUT2NB at the RTL level are implemented as an ROM memory. After performing the synthesis and implementation steps in Vivado IDE, the tables are assigned to the LUT tables [28] of FPGA. In FPGAs, LUT tables are part of configuration blocks and are the main resource for implementation of combinational logic. Multiplication and summation operations are automatically assigned to DSP cores [29]. The overall converter design needs only 50 slices representing just 0.32% of the used FPGA chip capacity. In Table 5, the obtained results are compared with the reference conventional approximation methods [6] and very fast shift-and-add methods [20]. Latency of the circuit, or the time from input to output as a performance measure, is very circuit-dependent. In the considered reference design, the FPGA Virtex II was used, which is an older product line of the Xilinx FPGA chips.

Table 5. Comparison of area, latency, and accuracy of a conversion from LNS to FLP.

	Conventional Design [6]	Proposed Design	Shift-and-Add Design [20]
Slices	236	50	NA
Multipliers	20	1	0
Memory	4 × 18 k BRAM	0	0
Latency	72 ns	22/12 ns	<1 ns
Accuracy	High	<0.001%	>1%

Previous converters are realized on 65 nm full customizable CMOS technology with latency under 1 ns [20]. For our implementation, a 28-nm prefabricated structure of FPGA Artix-7 was used. Even with this significant limitation, the achieved latency was 22.198 ns (12.083-ns logic delay and 10.115 net delay), which is a very promising result for the embedded real-time applications.

6.2. Simulation and Verification

For synthesis, simulation, and verification of the presented converter design, the Matlab–Simulink environment combined with the system generator toolbox [30] were used. The system generator toolbox is an additional part of Vivado IDE connecting synthesis and simulation tools of digital designs targeted for FPGA with Matlab–Simulink. After a successful installation of both environments, it is possible to add system generator blocks to the Simulink simulation scheme. These blocks are not simulated in the Matlab environment but separately in the Vivado simulator. The simulation scheme is shown in Figure 5. Specialized input and output blocks serve as an interface between Matlab and Vivado simulation tools. This approach is more flexible than a standard testbench created in Vivado IDE.

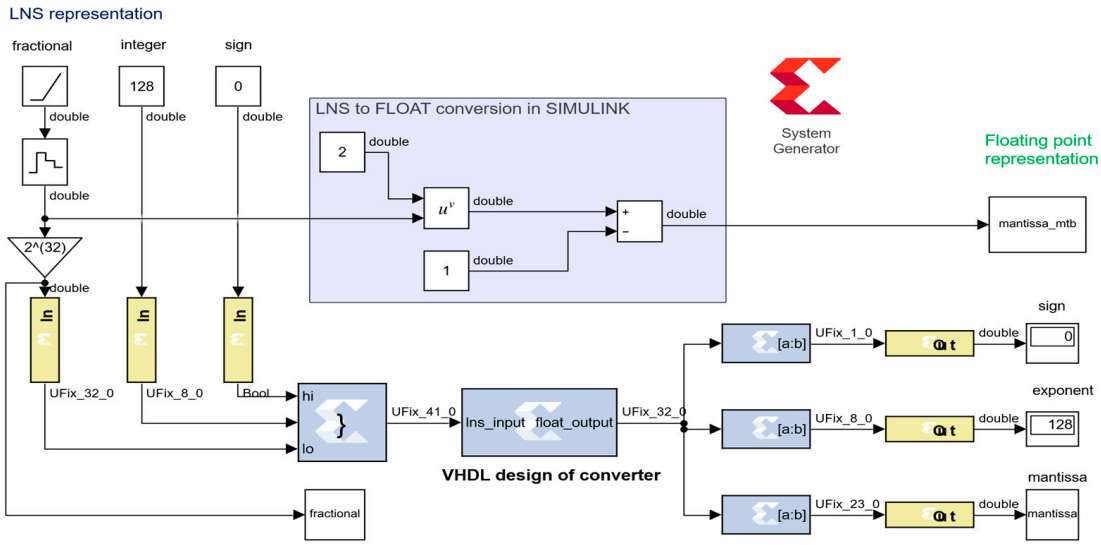

Figure 5. Converter simulation and verification in the Matlab–Simulink environment.

The core of the simulation scheme is the system generator block, which represents the VHDL code of the converter developed in Vivado IDE. The output signal from the converter is split into the sign, exponent, and mantissa parts. For verification, the mantissa signal simulated in Vivado using system-generator-for-DSP blocks and the mantissa_mtb obtained directly in Simulink were compared. The mantissa error was calculated as the difference between the mantissa and the mantissa_mtb.

From the simulated fractional input in the interval $\langle 0, 1\rangle$, the maximum positive mantissa error is 1.5806×10^{-5} and the maximum negative mantissa error is -1.4047×10^{-5}, which corresponds to the maximum error as mentioned in Section 4. The average error across the input range is 1.9979×10^{-7}.

Overall dependence of the conversion error on the fractional part L_Z is depicted in Figure 6. Dependence of the error focused for one sector on Address 30 in Table 3 is shown in Figure 7.

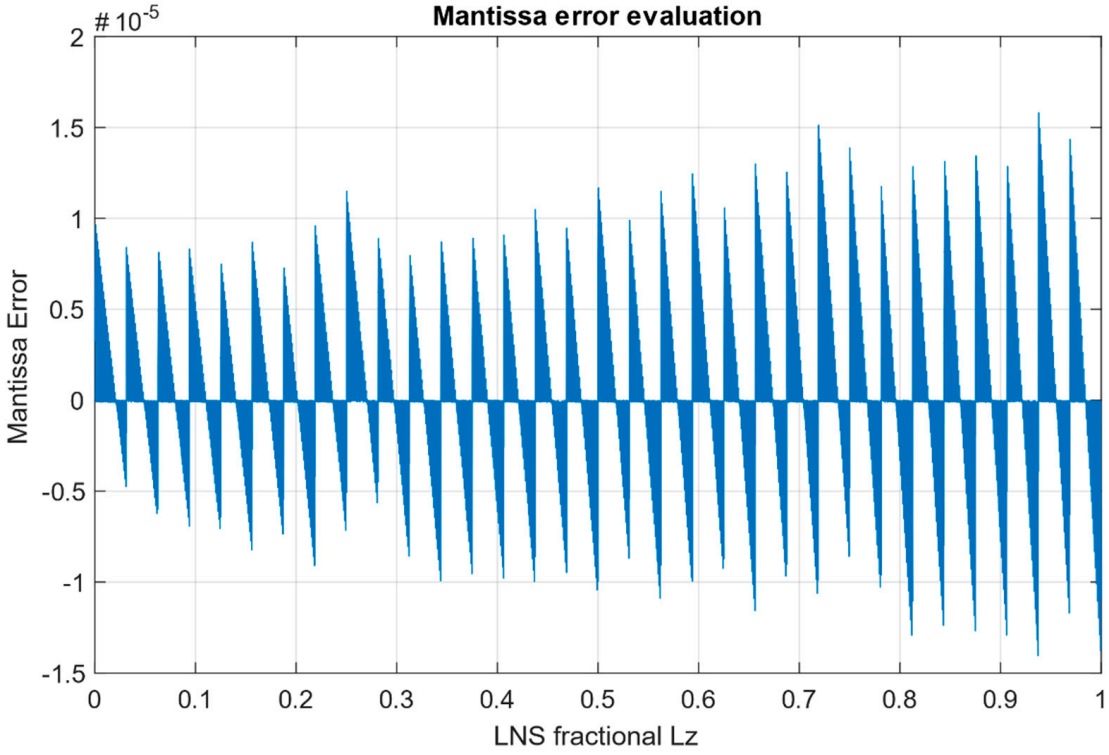

Figure 6. Mantissa error evaluation.

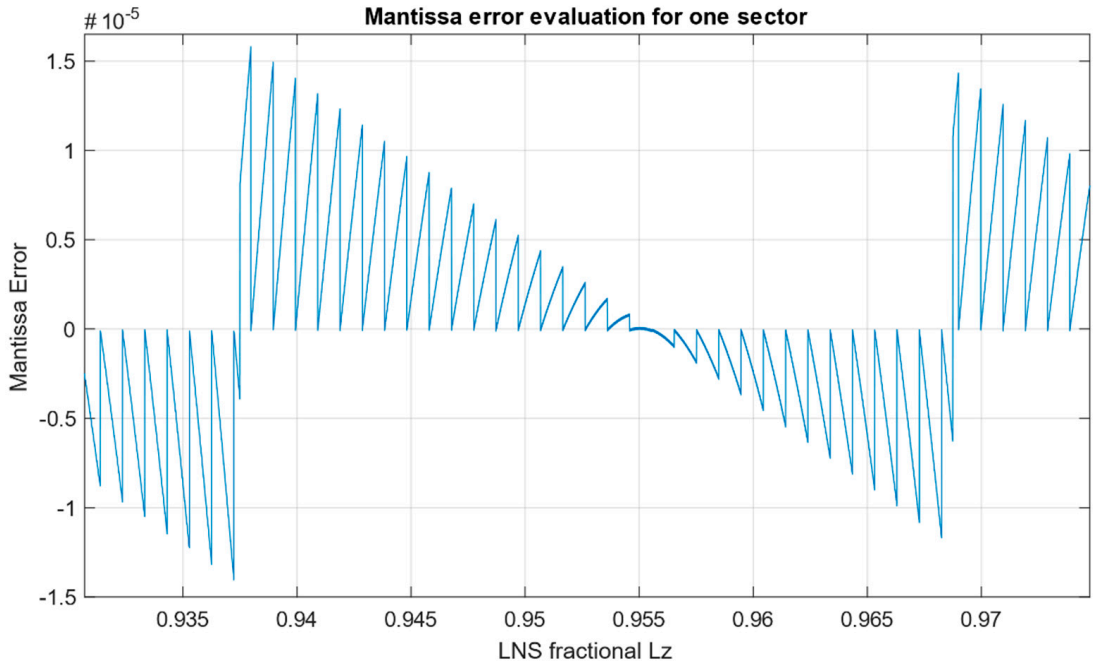

Figure 7. Zoomed mantissa error evaluation for one sector.

In Table 6, accuracy achieved using the proposed LiS method of LNS to FPL converter was compared with selected existing methods.

Table 6. Comparison of accuracies achieved using the proposed LiS method with selected conversion methods.

	Maximum Positive Error	Maximum Negative Error	Error Range	Maximum Positive Relative Error	Maximum Negative Relative Error	Relative Error Range
Mitchell [31]	0.086	0	0.086	5.791%	0	5.791%
Combet [32]	0.0253	−0.0062	0.0315	2.293%	−0.468%	2.761%
SG [33]	0.0292	−0.028	0.0572	2.503%	−1.704%	4.207%
Juang [18]	0.0369	−0.0097	0.0466	2.963%	−0.487%	3.45%
Juang [11]	0.0319	0	0.0319	2.337%	0	2.337%
AS [17]	NA	NA	NA	1.331%	−0,5631%	1,8941%
Kuo [19]	NA	NA	NA	1.2%	−0.1436%	1.3436%
Nandan [20]	NA	NA	NA	0.94%	−0.1436%	1.084%
Chen [23]	NA	NA	NA	NA	NA	0.138%
LiS method	**0.00001581**	**−0.00001407**	**0.00002988**	**0.000966%**	**−0.000783%**	**0.001749%**

It can be observed that the proposed converter has a much better accuracy compared with other presented converters.

The proposed looping-in-sectors (LiS) method can be included in a category in between the approximation methods and the shift-and-add methods, which are very fast and ROM-less, based only on logic circuits and binary operations. The proposed LiS method uses a very simple approximation at the bit-manipulation level, a simple so-called RISC operation, and needs relatively small memory (32 + 40 memory cells).

The advantage of the proposed LiS method over the shift-and-add methods is a much higher relative accuracy (up to 1000 times), the disadvantage is a higher latency. Compared to conventional approximation methods, its advantages are simplicity and a higher speed.

7. Discussion

Conversions of numbers play an important role in the LNS arithmetic, especially in real-time systems. In this paper, equivalence between the FLP (semi-logarithmic) and LNS (fully logarithmic) systems was defined, and conversion between them was reduced to a conversion between the mantissa and fraction performed within the interval $\langle 1, 2\rangle$. Derived correction functions enabled to specify an optimal interval for conversion within the plateau around their maxima where a mathematically simple and accurate interpolation can be performed. According to the developed procedure called looping in sectors (LiS), the converted number is to be moved to the plateau in the vicinity of the correction function peak and back after the interpolation accomplishment, hence reducing memory consumption. Note, that the LiS is performed without loss of accuracy, resulting in effective RISC conversions, which use only elementary computer operations. The developed conversions are then implemented in the designed LNS addition based on RISC operations and can thus be realized without a necessity to use multiplication, division, and other functions.

As a part of the development, conversion algorithms from LNS to FLP and vice versa [25] were implemented on FPGA, and their accuracy was verified by simulation. The presented methodology based on the new correction functions, the looping-in-sectors method, and the optimal choice of a base for an efficient interpolation can further be optimized for different types of applications in embedded systems. For different applications, different attributes are prioritized: in measurement and signal processing from sensors it is accuracy, in complex control algorithms it is speed, in automotive and autonomous systems reliability and credibility of the information obtained are the most essential ones.

The modern very fast "shift-and-add" methods (latencies about 1 ns) prevail only in selected complex algorithms that tolerate low accuracy (only 1%) but are inappropriate in other applications. The proposed LiS method is suitable for control applications in combination with input and output signal processing, as well as one of alternative methods for redundant signal processing and fault-detection due to a deterministically determined high accuracy. The proposed LiS method belongs to faster methods (with a latency of 22 ns) and can be used in real-time control applications even in time-critical applications with a sampling period up to 1 ms.

The future research will be directed on the functional safety of embedded system applications, usually implemented through redundancy and dual-channel technology today. In this sense, we understand LNS not only as an alternative to FLP but also as an SW/HW independent dual method of calculation to eliminate errors and increase the plausibility of results in full compliance with a new paradigm [34]: "It is much more important to know whether information is reliable or not than the accuracy of the information itself."

Author Contributions: Conceptualization, P.D. and O.H.; methodology, P.D.; software, M.K. and E.K.; validation, P.D., M.K. and E.K.; formal analysis, P.D.; investigation, P.D.; resources, M.K.; writing—original draft preparation, P.D.; writing—review and editing, P.D., M.K., and A.K.; supervision, O.H.; project administration, A.K.; funding acquisition, P.D. and A.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the SLOVAK RESEARCH AND DEVELOPMENT AGENCY, grant No. APVV-17-0190 and the SLOVAK CULTURAL EDUCATIONAL GRANT AGENCY, grant No. 038STU-4/2018.

Acknowledgments: The paper was partially supported by the Slovak Research and Development Agency, grant No. APVV-17-0190 and the Slovak Cultural Educational Grant Agency, grant No. 038STU-4/2018.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Lee, B.R. A Comparison of Logarithmic and Floating Point Number Systems Implemented on Xilinx Virtex-II Field Programmable Arrays. Ph.D. Thesis, Cardiff University, Cardiff, UK, 2004.
2. Chugh, M.; Parhami, M. Logarithmic arithmetic as an alternative to floating-point: A review. In Proceedings of the 2013 Asilomar Conference on Signals, Systems and Computers, Pacific Grove, CA, USA, 3–6 November 2013; pp. 1139–1143. [CrossRef]
3. Kingsbury, N.G.; Rayner, P.J.W. Digital filtering using logarithmic arithmetic. *Electron. Lett.* **1971**, *7*, 56–58. [CrossRef]
4. Swartzlander, E.E.; Alexopoulos, A.G. The Sign/Logarithm Number System. *IEEE Trans. Comput.* **1975**, *24*, 1238–1242. [CrossRef]
5. Arnold, M.G. A VLIW architecture for logarithmic arithmetic. In Proceedings of the Euromicro Symposium on Digital System Design, Belek-Antalya, Turkey, 1–6 September 2003; pp. 294–302. [CrossRef]
6. Haselman, H.M.; Beau champ, M.; Wood, A.; Hauck, S.; Underwood, K.; Hemmert, K.S. A comparison of floating point and logarithmic number systems for FPGAs. In Proceedings of the 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'05), Napa, CA, USA, 18–20 April 2005; pp. 181–190. [CrossRef]
7. Kouretas, I.; Paliouras, V. Logarithmic number system for deep learning. In Proceedings of the 2018 7th International Conference on Modern Circuits and Systems Technologies (MOCAST), Thessaloniki, Greece, 7–9 May 2018; pp. 1–4. [CrossRef]
8. Juang, T.; Lin, C.; Lin, G. Design of High-Speed and Area-Efficient Cartesian to Polar Coordinate Converters Using Logarithmic Number Systems. In Proceedings of the 2019 International SoC Design Conference (ISOCC), Jeju, Korea, 6–9 October 2019; pp. 180–181. [CrossRef]
9. Garcia, J.; Arnold, M.G.; Bleris, L.; Kothare, M.V. LNS architectures for embedded model predictive control processors. In Proceedings of the 2004 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES '04), New York, NY, USA, 7–16 September 2004; pp. 79–84. [CrossRef]
10. Arnold, M.G.; Collange, S. The Denormal Logarithmic Number System. In Proceedings of the 2013 IEEE 24th International Conference on Application-Specific Systems, Architectures and Processors, Washington, DC, USA, 5–7 June 2013; pp. 117–124. [CrossRef]
11. Juang, T.; Meher, P.K.; Jan, K. High-performance logarithmic converters using novel two-region bit-level manipulation schemes. In Proceedings of the 2011 International Symposium on VLSI Design, Automation and Test, Hsinchu, Taiwan, 25–28 April 2011; pp. 1–4. [CrossRef]

12. Arnold, M.G.; Kouretas, I.; Paliouras, V.; Morgan, A. One-Hot Residue Logarithmic Number Systems. In Proceedings of the 2019 29th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS), Rhodes, Greece, 1–8 March 2019; pp. 97–102. [CrossRef]
13. Zaghar, D.R. Design and Implementation of a High Speed and Low Cost Hybrid FPS/LNS Processor Using FPGA. *J. Eng. Sustain. Dev.* **2010**, *14*, 86–104.
14. Naziri, S.Z.M.; Ismail, R.C.; Shakaff, A.Y.M. The design revolution of logarithmic number system architecture. In Proceedings of the 2nd International Conference on Electrical, Electronics and System Engineering (ICEESE), Kuala Lumpur, Malaysia, 8–9 November 2014; pp. 5–10. [CrossRef]
15. Nam, B.; Kim, H.; Yoo, H. Power and Area-Efficient Unified Computation of Vector and Elementary Functions for Handheld 3D Graphics Systems. *IEEE Trans. Comput.* **2008**, *57*, 490–504. [CrossRef]
16. Arnold, M.G.; Collange, S. A Real/Complex Logarithmic Number System ALU. *IEEE Trans. Comput.* **2011**, *60*, 202–213. [CrossRef]
17. Abed, K.H.; Siferd, R.E. CMOS VLSI implementation of a low-power logarithmic converter. *IEEE Trans. Comput.* **2003**, *52*, 1421–1433. [CrossRef]
18. Juang, T.; Chen, S.; Cheng, H. A Lower Error and ROM-Free Logarithmic Converter for Digital Signal Processing Applications. *IEEE Trans. Circuits Syst. II Express Briefs* **2009**, *56*, 931–935. [CrossRef]
19. Kuo, C.; Juang, T. Area-efficient and highly accurate antilogarithmic converters with multiple regions of constant compensation schemes. *Microsyst. Technol.* **2018**, *24*, 219–225. [CrossRef]
20. Nandan, D. An Efficient Antilogarithmic Converter by Using Correction Scheme for DSP Processor. *Trait. Signal* **2020**, *37*, 77–83. [CrossRef]
21. Chen, C.; Chow, P. Design of a versatile and cost-effective hybrid floating-point/LNS arithmetic processor. In Proceedings of the 17th ACM Great Lakes symposium on VLSI (GLSVLSI '07), Stresa-Lago Maggiore, Italy, 11–13 March 2007; pp. 540–545. [CrossRef]
22. Ismail, R.C.; Zakaria, M.K.; Murad, S.A.Z. Hybrid logarithmic number system arithmetic unit: A review. In Proceedings of the 2013 IEEE International Conference on Circuits and Systems (ICCAS), Kuala Lumpur, Malaysia, 18–19 September 2013; pp. 55–58. [CrossRef]
23. Chen, C.; Tsai, T.C. Application-specific instruction design for LNS addition/subtraction computation on an SOPC system. In Proceedings of the 2009 6th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, Pattaya, Thailand, 6–9 May 2009; pp. 640–643. [CrossRef]
24. Finnerty, A.; Ratigner, H. *Reduce Power and Cost by Converting from Floating Point to Fixed Point—White Paper: Floating vs Fixed Point*; Xilinx: San Jose, CA, USA, 2017.
25. Drahos, P.; Kocur, M. Logarithmic addition and subtraction for embedded control systems. In Proceedings of the 2020 Cybernetics & Informatics: 30th International Conference, Velke Karlovice, Czech Republic, 29 January–1 February 2020. [CrossRef]
26. Klimo, I.; Kocúr, M.; Drahoš, P. Implementation of logarithmic number system in control application using FPGA. In Proceedings of the 16th IFAC Conference on Programmable Devices and Embedded Systems (PDEeS´19), High Tatras, Slovak Republic, 29–31 October 2019. [CrossRef]
27. Xilinx. 7 Series FPGAs Data Sheet: Overview. Xilinx. 2018. Available online: https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf (accessed on 10 April 2020).
28. Xilinx. 7 Series FPGAs Configurable Logic Block: User Guide. Xilinx. 2016. Available online: https://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf (accessed on 10 April 2020).
29. Xilinx. 7 Series DSP48E1 Slice: User Guide. Xilinx. 2018. Available online: https://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf (accessed on 10 April 2020).
30. Xilinx. Vivado Design Suite User Guide: Model-Based DSP Design Using System Generator. Xilinx. 2019. Available online: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_2/ug897-vivado-sysgen-user.pdf (accessed on 10 April 2020).
31. Mitchell, J.N. Computer Multiplication and Division Using Binary Logarithms. *IRE Trans. Electron. Comput.* **1962**, *11*, 512–517. [CrossRef]
32. Combet, M.; Zonneveld, H.V.; Verbeek, L. Computation of the base two logarithm of binary numbers. *IEEE Trans. Electron. Comput.* **1965**, *14*, 863–867. [CrossRef]

33. SanGregory, S.L.; Siferd, R.E.; Brother, C.; Gallagher, D. A fast, low-power logarithm approximation with CMOS VLSI implementation. In Proceedings of the 42nd IEEE Midwest Symposium on Circuits and Systems (MWSCAS), Las Cruces, NM, USA, 8–11 August 1999; Volume 1, pp. 388–391. [CrossRef]
34. The European Safety Critical Applications Positioning Engine (ESCAPE). GSA/GRANT/02/2015. Available online: http://www.gnss-escape.eu/ (accessed on 15 June 2020).