Scientia Iranica D (2014) 21(3), 826–841

Sharif University of Technology
Scientia Iranica
Transactions D: Computer Science & Engineering and Electrical Engineering
www.scientiairanica.com

High speed multiplier using high accuracy floating point logarithmic number system

P. Saha[a], A. Banerjee[b], A. Dandapat[a] and P. Bhattacharyya[c,*]

a. *Department of Electronics and Communication Engineering, National Institute of Technology, Meghalaya, Shillong, Meghalaya-793003, India.*
b. *Department of Electronics and Communication Engineering, JIS College of Engineering, Kalyani, West Bengal-741235, India.*
c. *Department of Electronics and Telecommunication Engineering, Bengal Engineering and Science University, Shibpur, Howrah-711103, India.*

Received 21 November 2012; received in revised form 22 July 2013; accepted 15 October 2013

KEYWORDS
Canonical sign digit code;
Logarithmic number system;
Multiplier;
High speed;
High accuracy.

Abstract. ASIC implementation of a high speed multiplier using a high accuracy floating point logarithmic number system is reported in this paper. The most popularly used techniques for computing logarithmic calculations for digital signal processors are: Lookup table based implementation, polynomial approximation, and Taylor series expansion. But, all these techniques suffer from low accuracy, due to the choice of having only lower order terms of the expanded series. In the present work, logarithmic conversion is implemented by a floating point (IEEE-754 single precision) converting methodology, thereby eliminating series expansion, which eventually results in high accuracy. The improvement in speed, by avoidance of carry propagation, was achieved through Canonical Signed Digit Code (CSDC) implementation, while the high accuracy was achieved through an error minimization circuitry especially designed for this purpose. The functionality of these circuits was checked, and performance parameters, like propagation delay and dynamic power consumption, were calculated by spice spectre using 90 nm CMOS technology. The propagation delay and power consumption of the resulting (128 × 128) bit multiplier (divider) was only ∼ 93 ns and ∼ 80 mW, respectively, for a layout area of ∼ 25mm^2. This implementation offered a significant improvement in terms of accuracy, delay and power from those reported earlier.

1. Introduction

Computation of logarithmic and exponential numbers plays a pivotal role in the field of digital signal processing [1-5]. Multiplication of higher order bits (order of 64×64 and higher) requires a large number of hardware components, due to the generation and processing of huge numbers of partial products. The generation of partial products can be avoided by using a Logarithmic Number System (LNS), where the multiplications are converted to direct addition [$\log(x.y) = \log x + \log y$]. But, LNS suffers from poor accuracy, because most conventional approximate techniques use only lower order terms of its expanded series. So, if this particular bottleneck of the above mentioned methodology is addressed properly, realization of a high accuracy multiplier with appreciable speed becomes feasible.

A substantial amount of research work has so far been reported for LNS approximation, and performance parameters, like accuracy, speed and power optimization, were addressed [1-17]. Mitchell [1] and Ramaswamy and Siferd [2] reported a floating point

*. *Corresponding author. Tel: +913 326684561; Fax: +913 326682916*
E-mail addresses: sahaprabir1@gmail.com (P. Saha); banerjee.arindam1@gmail.com (A. Banerjee); anup.dandapat@gmail.com (A. Dandapat); pb_etc_besu@yahoo.com (P. Bhattacharyya)

conversion methodology of binary logarithmic numbers through straight line approximation, with an error in accuracy of $\sim 12\%$ during multiplication operation.

Numerous efforts have so far been made to improve the accuracy of straight line approximating algorithms [3-7]. Babic et al. [3] introduced an iterative logarithmic multiplier, and Khalid et al. [4,5] investigated a correcting procedure aiming to reduce error. Among these techniques, McLaren's method [6] uses a look-up table with 64 correction coefficients, which is responsible for calculation of the dependencies of the mantissas values, thereby, increasing the complexity of the realization. Mahalingam and Rangantathan [7] reported an operand decomposition-based straight line approximating algorithm. The reported implementation decreases the error percentage of the original straight line approximating algorithm, which was investigated by Mitchell [1], but eventually increases ($\sim$ double) the hardware compared to the original.

Many alternative techniques, like lookup table based implementation [8-10], polynomial approximation [11-13], and Taylor series based approaches, like digit by digit [14,15] implementation etc., have been used so far for approximation to implement a log-based multiplier. Hardware implementation through a direct look-up table [8-10] is a straightforward and easily implementable approach, but with poor accuracy. The lookup table based implementation consists of precomputed function values for every possible argument in tabular form. Thereby, the number of table entries increases exponentially for the number of bits representation of the argument, leading towards the requirement of large memory size, slower operation and higher power consumption. Other approaches, such as piecewise polynomial approximations [11,12], use Lagrange interpolation to compute LNS numbers. Lee and Burgess [13] implemented the 32-bit LNS arithmetic functions using Chebyshev polynomial approximation. One major drawback of the polynomial approximation method is its slow linear convergence rate. Other than polynomial approximation methods, there are also digit-serial methods (also known as on-line or iterative methods), which calculate the result, digit by digit, also based on Taylor series expansion [14,15]. The greatest disadvantage of these approximation [11-15] techniques is their low accuracy, due to the choice of having only lower order terms of the expanded log-series. With the aim of solving this problem, Chen and Chen [16] proposed a pipelined addition/subtraction unit for LNS numbers of very large word-length. The algorithm, which is similar to the CORDIC method [18], approximates the functions with digit-serial sequences of computations, based on the continued product normalization of the terms of $(1 + S_k 2^{-k})$, which ultimately results in better speed and power compared to earlier reports [1,2,8-15]. But, this technique also suffers from the same accuracy problem mentioned earlier. Very recently, Fu et al. [17] theoretically investigated a LNS processor based on floating point conversion and reported the potentiality of the method from an area and latency point of view.

ASIC implementation of a high accuracy floating point (IEEE-754 single precision) conversion methodology for an LNS facilitating high speed multiplier is reported in this paper. The functionality of these circuits has been designed and verified by spice spectre in 90 nm CMOS technology. The proposed method offers a substantial reduction of propagation delay compared to earlier reported architectures, like the look up table [8], polynomial approximation [11], digit by digit [15], and implementation methodology, because the iterative process was eliminated from this design. IEEE-754 single precision floating point conversion techniques were considered for converting binary numbers to binary logarithm numbers. Moreover, CSDC techniques have been applied to the addition/subtraction of two binary numbers to achieve high speed operation [19,20]. Propagation delay for the proposed (128×128) multiplier/divider was only $\sim$ 93 ns, with only $\sim$ 80 mw power for a layout area of ~ 25 mm^2.

2. Design preliminaries of logarithmic number system

In this section, mathematical analysis of design preliminaries for LNS and their conversion methodology, like binary number to LNS and LNS to binary number conversion, has been described.

2.1. Logarithm arithmetic

For the same bit length, floating point and logarithmic number have the same formulation, though their arithmetic representations are different. In the case of floating point numbers, addition/subtraction is easier than multiplication because iterative addition is required to perform the same [21]. But, in LNS, the multiplication procedure obeys the easier implementing techniques because logarithmic multiplication can be executed by floating point addition, whereas LNS addition can be implemented by normalization techniques, like straight line approximation [2], polynomial approximation [11-13], Taylor series expansion [14,15] etc., for which complex circuitry is required. In LNS, a number, X, is represented as a signed exponent word of the form represented in [12,21]:

$$X \cong \pm r^{\pm E_x}. \tag{1}$$

More generally, we can describe the logarithmic number as:

$$X = (-1)^{s_x} \times r^{\pm E_x}, \tag{2}$$

where X is the set of the distinct components of $<$

$s_x, E_x >$, 's_x' represents the sign of numbers, and 'r' represents the base of the logarithm. To implement the hardware of logarithms, $r = 2$ (base '2' logarithm or binary logarithm) has been considered. 'E_x' is the 'N' bit number, where $N = I + F$; I is the integer part and F is the fractional part of 'E_x'. Thus, logarithmic numbers can be represented as a set of floating point numbers, which indicate that the existence of 'zero' was missing in Eqs. (1) and (2). A small modification is required to determine the exact value of the logarithmic number representation. The modified equation can be written as:

$$X = (1 - p_x) \times (-1)^{s_x} \times r^{\pm E_x}, \tag{3}$$

where $p_x \in (0, 1)$. If the value of the 'p_x' is equal to '1', then it represents the value of '0', which cannot be included in the logarithmic number system, because '$\log_r 0$' is indeterminate. 'E_x' can be represented as:

$$E_x = \sum_{i=-F}^{I-1} x_i 2^i. \tag{4}$$

The arithmetic formulation of LNS can be represented in the following manner: $X = r^{E_x}$, $Y = r^{E_y}$, $Z = r^{E_z}$ and $S =$ the sign bit of the number.

i) Multiplication:

$$Z = XY,$$

and:

$$E_z = E_x + E_y; S_z = S_x \oplus S_y. \tag{5}$$

ii) Division:

$$Z = \frac{Y}{X},$$

and:

$$E_z = E_x + E_y; S_z = S_x \oplus S_y. \tag{6}$$

iii) Addition:

$$Z = X + Y = E_x + \log_r[1 + r^{(E_y - E_x)}],$$

$$S_z = S_x. \tag{7}$$

iv) Subtraction:

$$Z = X - Y = E_x + \log_r[1 - r^{(E_y - E_x)}],$$

$$S_z = S_x. \tag{8}$$

Thus, from the above expressions (Eqs. (5)-(8)), it can be observed that multiplication and division can be performed using only addition or subtraction, respectively.

2.2. Binary number to LNS conversion

Consider that a binary number, X, can be written as:

$$X = \sum_{i=0}^{k} x_i 2^i,$$

where:

$$x_i \in 0 \quad \text{or} \quad 1. \tag{9}$$

Taking binary logarithm on both sides of Eq. (9), we obtain:

$$Y = \log_2 X = \log_2 \left(\sum_{i=0}^{k} x_i 2^i \right), \tag{10}$$

$$= \log_2 \left(2^k \left(x_k + \sum_{i=1}^{k} x_{k-i} 2^{-i} \right) \right), \tag{11}$$

$$= k + \log_2 \left(x_k + \sum_{i=1}^{k} x_{k-i} 2^{-i} \right). \tag{12}$$

Let us assume that $x_k = 1$ in Eq. (12) can be approximated as (higher order terms have been ignored for simplicity):

$$Y = k + \sum_{i=1}^{k} x_{k-i} 2^{-i}. \tag{13}$$

From Eq. (13), it can be observed that Y is a hybrid (exponent and mantissa) number. Then, the alternatively hybrid number can be represented as:

$$Y = \sum_{i=-k}^{m} y_i 2^i, \tag{14}$$

$$Y = \sum_{i=0}^{m} y_i 2^i + \sum_{i=-1}^{-k} y_i 2^i, \tag{15}$$

$$Y = 2^m \left(y_m + \sum_{i=-1}^{-(m+k)} y_{m+i} 2^i \right). \tag{16}$$

From the analogy in Eqs. (13) and (15), the equation can simply be represented as:

$$k = \sum_{i=0}^{m} y_i 2^i,$$

and:

$$\sum_{i=1}^{k} x_{k-i} 2^{-i} = \sum_{i=-1}^{-k} y_i 2^i.$$

The same number, Y, can also be represented in IEEE-754 single precision format, and then, Y can be expressed as:

$$Y = (-1)^s \times 2^{E_y - E_b} \times 1.M = 2^{E_y - E_b} \times (1 + 0.M), \tag{17}$$

where $E_b = 127_{10}$ (bias of single precision) and sign

bit '$s = 0$' (negative logarithmic values have not been considered). Let us assume that '$y_m = 1$' and through the analogy of Eqs. (16) and (17):

$$m = E_y - E_b,$$

and:

$$M = \sum_{i=-1}^{-(m+k)} y_{m+i}2^i.$$

In terms of the IEEE-754 format, '$m = 8$' and '$M = 23$' bit number, and M can be computed by [8] a rounding scheme.

2.3. LNS to binary number conversion

Consider a logarithmic number, represented in an IEEE-754 single precision format,

$$\log_2 X = E_x + \sum_{i=-1}^{-(k+m)} x_{m+i}2^i,$$

where E_x is the exponent, and assume that:

$$p = \sum_{i=-1}^{-(k+m)} x_{m+i}2^i,$$

is the floating point part of the logarithmic number. Assume '$E_b = 127_{10}$' is the bias of the exponent. Unbiased exponent 'm' is computed as:

$$m = E_x - E_b. \tag{18}$$

Using the IEEE-754 format, the same logarithmic number can be represented as:

$$\log_2 X = 2^m \times (1+0.p) = 2^m \left(1 + \sum_{i=-1}^{-(k+m)} x_{m+i}2^i\right), \tag{19}$$

$$= 2^m \left(1 + \sum_{i=-1}^{-m} x_{m+i}2^i + \sum_{i=-(m+1)}^{-(m+k)} x_{m+i}2^i\right), \tag{20}$$

$$= \sum_{i=0}^{m} x_i 2^i + \sum_{i=-1}^{-k} x_i 2^i, \tag{21}$$

$$= k + \sum_{i=-1}^{-k} x_i 2^i, \tag{22}$$

where $k = \sum_{i=0}^{m} x_i 2^i$. So, X can be written as:

$$X = 2^{k+\sum_{i=0}^{m} x_i 2^i}, \tag{23}$$

$$\cong 2^{k+\log_2\left(1+\sum_{j=-1}^{-k} x_i 2^i\right)}, \tag{24}$$

$$= 2^k \left(1 + \sum_{j=-1}^{-k} x_i 2^i\right), \tag{25}$$

$$= \sum_{i=0}^{k} x_i 2^i. \tag{26}$$

2.4. Examples

2.4.1. Binary to logarithmic number conversion

Assume a 16 bit number, X, $X =$"0110011100110101". (For the sake of simplicity, we assume a 16 bit number; a higher number of bits can be implemented in the same manner). For this number, the highest power of 2 is 14_{10}. That is the power of 2 corresponding to the first non-zero bit. So, $k = 14_{10} =$"1110"$_2$ and $M =$"1001110011010100". That means $Y = \log_2 X =$ '1110.1001110011010100', which is a total of 20 bits. So, the exact exponent is the highest power of the MSB, which is 3 plus the bias. Here, bias $E_b = 127$. So, $E_y = 3 + 127 = 130$ ('10000010'). Now, shift out the MSB, and the newly generated number for mantissa is $0.M = 0.11010011100110101000000$. So, the logarithm in the IEEE-754 single precision format is 010000010.11010011100110101000000, where MSB '0' is the sign bit.

2.4.2. Hybrid number to binary number conversion

Consider a 32 bit hybrid number (the logarithmic value of a number is given), whose antilogarithm is to be determined. Assume that the number is given as: $X =$ '010000010.11010011100110101000000'. For this number, $E_x =$ '010000010' and $E_b = 127_{10}$. Thus $m = 10000010 - E_b = 10000010 - 01111111 = 00000011 = 3_{10}$ is the mantissa of the same number, $1 + 0.p =$ '1.11010011100110101000000'. Now, left shift $1+0.p$ by 3 times (because '$m = 3$') and the generated number is '1110.1001110011010100000'. Thus, making the analogy with Eq. (23) "$k = 1110_2 = 14_{10}$". Place '1' at the position of $2^k = 2^{14}$, shift the 23 bit mantissa towards the left by '$k = 14$' times, and the newly generated number is '0110011100110101.000000'. Therefore, the antilogarithm of the number is equal to '0110011100110101.000000'. (N.B: If a logarithmic number, X, is represented in IEEE-754 format, then, the fixed point 8 bit number represents the characteristic.) The antilogarithm of the number must be in the range: $\text{antilog}_2 X < 2^{255}$. On the other hand, if X is a binary number, then X must be in the range $0 < X < 128$.

3. Logarithmic multiplier implementation

Minimization of error for conversion of binary to logarithmic and logarithmic to binary numbers leading towards the implementation of a high speed multiplier is the main concern of this section. In previous architecture, such as straight line approximation [1,2], the

error in accuracy was $\sim \pm 12\%$ during multiplication. In the proposed implementation methodology, error has been taken care of by a special error minimization circuitry especially designed for conversion techniques, which ensures a quantifiably explicit reduction of error in accuracy. The mathematical description for ASIC implementation to perform the operation of multiplication/division has been described using the following set of equations.

Consider two binary numbers, X and Y. The multiplication/division can be performed as:

$$S = X \times Y = \log_2 S = \log_2 X + \log_2 Y. \tag{27}$$

With the help of Eq. (17), multiplication and division can be reformulated as:

$$\log_2 S = 2^{m+E_b}(1+p_x) + 2^{n+E_b}(1+p_y), \tag{28}$$

$$= 2^{E_b}(2^m(1+p_x) + 2^n(1+p_y)). \tag{29}$$

With the help of Eq. (21), Eq. (29) can be rewritten as:

$$= 2^{E_b}\left\{\left(\sum_{i=0}^{k_1-1} x_i 2^i + \sum_{i=0}^{k_2-1} x_i 2^i\right) + \left(\sum_{i=-1}^{-(m-k_1)} x_i 2^i + \sum_{i=-1}^{-(n-k_2)} x_i 2^i\right)\right\}, \tag{30}$$

$$= 2^{E_b}\left(\sum_{i=0}^{p-1} x_i 2^i + \sum_{i=-1}^{-q} x_i 2^i\right), \tag{31}$$

where $p = k_1$ or k_2 and $q = (m-k_1)$ or $(n-k_2)$. If $k_1 > k_2$, then $p = k_1$, else $p = k_2$, and if $k_1 = k_2$, then $p = k_1$ or k_2. If $(m-k_1) > (n-k_2)$, then $q = (n-k_2)$, otherwise $q = (m-k_1)$:

$$\log_2 S = 2^{E_b} \times 2^p \left(1 + \sum_{i=-1}^{-p} x_i 2^i + \sum_{i=-(p+1)}^{-(p-q)} x_{p+i} 2^i\right). \tag{32}$$

Eq. (32) can be rewritten after rounding up to a 23 bit mantissa using a rounding toward zero schemes, and then the equation can be reformulated as:

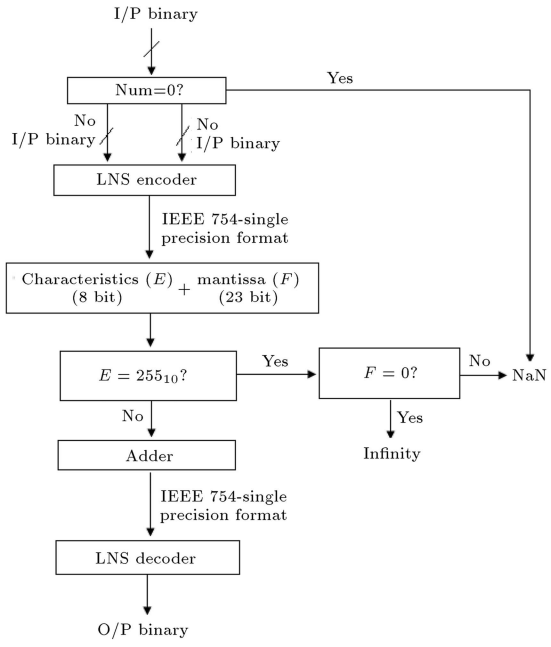
$$= 2^{E_b}(2^p(1+p_p)),$$

$$= 2^{E_b+p}(1+p_p) = 2^{E_z}(1+p_p), \tag{33}$$

where, $E_z = E_b + p$.

$$p_p = \text{round}\left(\sum_{i=-1}^{-p} x_i 2^i + \sum_{i=-(p+1)}^{-(p+q)} x_{p+i} 2^i\right). \tag{34}$$

The flowchart diagram of logarithmic multiplication is shown in Figure 1(a). Both the input numbers are fed to the LNS encoder (Log Converter). The output of the logarithmic encoder has been converted to an IEEE-754 single precision format. In the next stage, the output of the LNS encoder is passed through the exception detector block to compute the value of the exception. If the characteristic is equal to '$E = 255_{10}$', then, the corresponding mantissa is checked by the same block, namely, the exception block. If the mantissa is non-zero, then, the final result is 'NaN' (Not a Number).



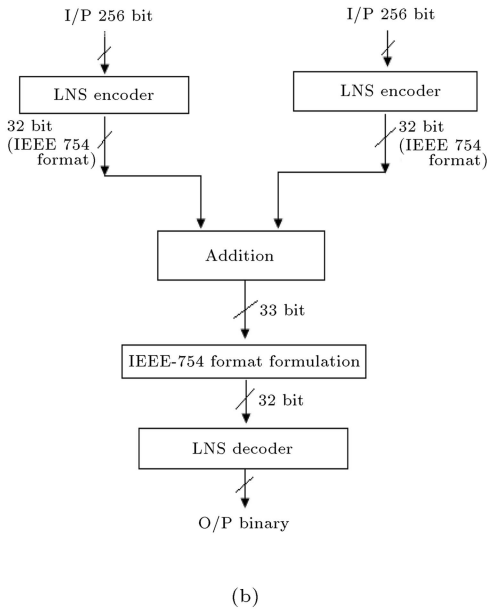


Figure 1. (a) Flow chart diagram of logarithmic multiplier. (b) Hardware implementation of logarithmic multiplier.

Furthermore, if the mantissa equals zero, then, the final result is infinity. In the next stage, the output of the conversion is fed towards the adder block to perform multiplication, respectively. The output of the adder unit has been fed towards the rounding block to perform the rounding operation, to again generate IEEE-754 single precision formation. The output of the rounding block has been fed to the LNS decoder (antilog converter), to convert the number towards binary formation.

The architecture for the hardware implementation of logarithmic multiplication has been shown in Figure 1(b). The architecture consists of five main functional blocks:

(i) LNS encoder;

(ii) Addition;

(iii) IEEE-754 format formulation;

(iv) LNS decoder.

A maximum of 128 bits input can be fed to the LNS encoder, which has a maximum characteristic value of 127, which can be easily represented in IEEE-754 single precision format. According to Eqs. (5) and (6), multiplication is simply converted towards addition. The addition module adds the values coming from the LNS encoder. After addition, the number of bits from the adder module is 33 (including the sign), and the irregular IEEE-754 format is again fed to the converter to convert the IEEE-754 format generation circuitry, which has been converted to a 32 bit pattern for single precision. The extra bits at the mantissa part are eliminated by the rounding toward zero schemes through the rounding module. Finally, the 32 bit single precision output from the rounding module is fed to the LNS decoder, which produces a hybrid binary output.

4. Circuit modules and implementation algorithm

4.1. LNS encoder

4.1.1. LNS encoder algorithm

The algorithm of the hardware implementation of the logarithmic number system can be described as:

Step 1: Count the position of first '1' from the MSB side. The position of the first '1' represents the integer part of logarithmic number conversion.

Step 2: The rest of the part, except first '1' from the MSB side, represents the fractional part of logarithmic number conversion.

Step 3: Count the position of first '1' from the MSB side of the hybrid number (combining integer and fractional parts). The position of the first '1' represents the unbiased exponent of the logarithmic number, according to the IEEE-754 format.

Step 4: The rest of the part except the first '1' from the MSB side represents the mantissa of the logarithmic number as per the IEEE-754 format.

Step 5: Add the bias for the IEEE-754 single precision format to the unbiased exponent to achieve the biased exponent.

4.1.2. Hardware implementation of LNS encoder

A flow chart diagram for hardware implementation of the LNS encoder is shown in Figure 2. The integer part of the logarithmic number from the binary fixed point number can be obtained by the maximum power of the radix. The logarithm of '0' is an invalid number; in our proposed design, it is represented as NaN (Not a Number). A shifting operation is executed using the left shift register [22]. The initial value of the select line of the shifter has been considered as '1'. A control signal is initially assigned to '0' to control left shift operation, whether the shift operation is executed or not. Moreover, at the second phase, after executing one left shift, it will check the MSB of the left shift

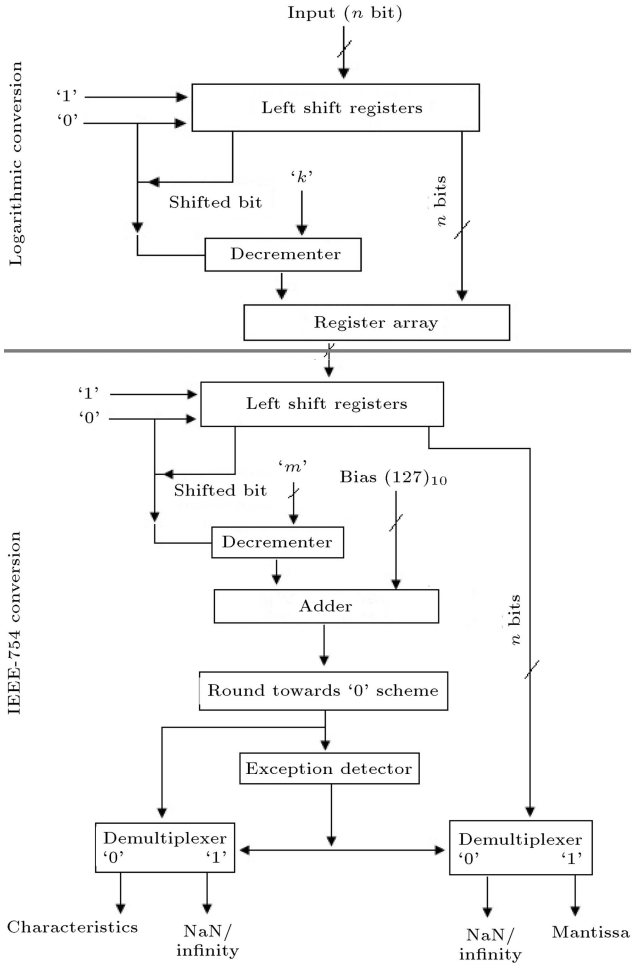


Figure 2. Hardware implementation of LNS encoder.

register. If the MSB is equal to '1', it stops the shifting, otherwise, shift operation will be continued until the MSB is equal to '1'. A decrementer [23] has been integrated in this architecture to generate the maximum power of the radix. A sequential search procedure has been implemented here to search the first '1', starting from the MSB side through shift operation. For an N bit number, the value $(N-1)_{10}$ is fed to the input of the decrementer. If the shifted bit is equal to '1', no decrement operation is executed. Otherwise, the decrement operation is executed until the MSB bit is equal to '1'. If the shifted bit is '0', then, the control signal becomes low and the decrementer starts decrementing the input value (here, the decrementer is working in active low logic). When the shifted bit is '1', then, the control signal becomes high, the decrementer stops further decrementing and the shifter also stops the shifting operation. The output of the decrementer shows the integer part of the logarithmic number and the number at the shifter is the corresponding fractional part.

Both the integer and fractional parts of the logarithmic number are stored in the left shift register again, and the maximum power of the radix of the logarithmic number is stored in a second decrementer. The above mentioned searching operation is repeated again until the most significant '1' is found. The decrementer stops operating if the most significant '1' is achieved. The value at the decrementer is the unbiased exponent of the logarithmic number, and the content at the shifter is the mantissa. An adder is used to compute the biased exponent (characteristics), which is obtained by addition of the unbiased exponent and bias, as per IEEE-754 single precision format [24]. The exception detector takes the biased exponent and the mantissa as input and decides whether the number is valid or not. If the biased exponent is 255_{10} and the mantissa is 0.0_{10}, then, the logarithmic number is infinite. Again, if the biased exponent is 255_{10} and the mantissa is any number except 0, then, the logarithmic number is NaN (Not a Number). The result is passed through demultiplexers, which are shown in Figure 2, to determine the corresponding characteristics and mantissa part of the given logarithmic number.

4.2. Addition/subtraction using CSD

4.2.1. CSD number representation

The CSD number representation is one of the Signed Digit Number Representations (SDNR's) proposed by Avizienis [20] to reduce the carry propagation in addition, subtraction, multiplication and division. They differ from conventional numbers, where the numbers contain negative as well as positive signs. If x_i is the set of distinct values, $\{0,\ 1,\ -1\}$, for each 'i', in long format, x_i can be represented as $x_0, x_1, x_2 \cdots x_{N-1}$, and the mathematical representation can be formulated as:

$$X = \sum_{i=0}^{N-1} x_i 2^i. \tag{35}$$

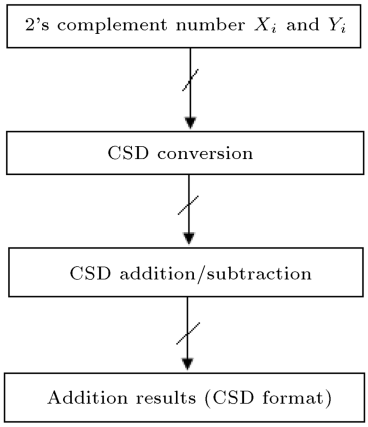


Figure 3. Logical flow chart diagram of CSD addition.

4.2.2. CSD addition/subtraction

The canonical sign digit adder/subtractor (CSD adder/subtractor) performs carry propagation free addition [19]. Carry propagation free addition has been performed by determining the intermediate carry and intermediate sum digits. Figure 3 represents the logical flow chart diagram of CSD addition, where carry propagation free addition has been performed in three steps:

i. Check the type of operation (addition). For addition, the sign of the individual bits remains unchanged. For subtraction, the signs of the individual nonzero bits are inverted.

ii. Determine the intermediate carry, $\{C_i \in (\bar{1}, 0, 1)\}$, and intermediate sum digits, $\{S_i \in (\bar{1}, 0, 1)\}$, satisfying the condition, $x_i + y_i = z_i + C_{i-1}$, where x_{i+1} and y_{i+1} are the augends and addend digits, respectively.

iii. Obtain the sum digits, $\{Z_i \in (\bar{1}, 0, 1)\}$, at each position by adding the intermediate sum digits, S_i and C_i, from the next lower order positions.

The truth table implementation from step (ii) is shown in Table 1. Boolean expressions have been formed from the above steps and shown in Eqs. (36)-(41). Here, 'z_i' and 'c_{i-1}' represent the intermediate sum and the intermediate carry. 'signx_i' and 'signy_i' represent the sign magnitude of 'x_i' and 'y_i', respectively.

'Signc_{i-1}' and 'signz_i' are the sign magnitude of the intermediate carry and intermediate sum, respectively. 'sum$_i$' and 'signsum$_i$' are the final stage sum and its sign magnitude, respectively. A Boolean expression has been formed for the final stage sum and its sign magnitude is shown in Eqs. (40) and (41). A canonical

Table 1. Truth table for determining the intermediate sum and intermediate carry.

Augend digits (x_i)	Addend digits (y_i)	Digits of the previous higher order posions (x_{1+1}, y_{i+1})	Intermediate carry (C_{i-1})	Intermediate sum (z_i)
0	0	_____	0	0
0	1	Both are non-negative	1	$\bar{1}$
1	0	Otherwise	0	1
1	1	_____	1	0
0	$\bar{1}$	Both are non-negative	0	$\bar{1}$
$\bar{1}$	0	Otherwise	$\bar{1}$	1
1	$\bar{1}$	_____	0	0
$\bar{1}$	1	_____	0	0
$\bar{1}$	$\bar{1}$	_____	$\bar{1}$	0

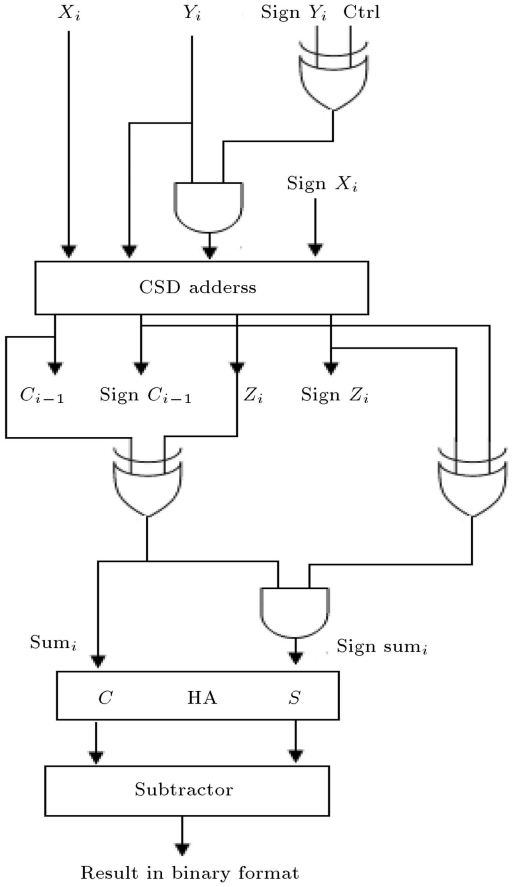


Figure 4. CSD adder/subtractor.

sign digit adder circuit has been used here for both addition and subtraction [19].

To implement the subtractor, a small hardware was added with the adder circuit. Hardware implementation of the adder/subtractor is shown in Figure 4. The architecture for the CSD adder/subtractor can be decomposed into two sections, viz, addition/subtraction, through CSDC and CSD, to binary conversion.

For the first segment, Boolean expressions are to be formed from the above steps, as shown in Eqs. (36)-(41). Here, 'z_i' and 'c_{i-1}' represent the intermediate sum and intermediate carry. 'signx_i' and 'signy_i' represent the sign magnitude of 'x_i' and 'y_i' respectively. 'signc$_{i-1}$' and 'signz$_i$' are the sign magnitudes of intermediate carry and intermediate sum, respectively, and 'sum$_i$' and 'signsum$_i$' are the final stage sum and its sign magnitude, respectively. A Boolean expression can be formed for the final stage sum, and its sign magnitude is shown in Eqs. (40) and (41). The second segment, consisting of a half adder and a sub-tractor, is used for the conversion of CSD to a binary number system. A Boolean expression is shown here only for addition. For subtraction purposes, '$\oplus$ and signy_i' is the inverted value, if the value of 'y_i' is nonzero.

$$Z_i = x_i \oplus y_i, \tag{36}$$

$$c_{i-1} = (\text{sign}x_{i+1} + \text{sign}y_{i+1})\,(x_i \oplus y_i) \oplus (\text{sign}x_i \oplus \text{sign}y_i), \tag{37}$$

$$\text{sign}z_i = z_i\left(\overline{\text{sign}x_{i+1} + \text{sign}y_{i+1}}\right), \tag{38}$$

$$\text{sign}c_{i-1} = (\overline{x_i \oplus y_i})(\text{sign}x_i \text{sign}y_i) + (x_i \oplus y_i)(\text{sign}x_{i+1} + \text{sign}y_{i+1}), \tag{39}$$

$$\text{sum}_i = (z_i \oplus c_{i-1}), \tag{40}$$

$$\text{signsum}_i = (\text{sign}z_i \oplus \text{sign}c_{i-1})(z_i \oplus c_{i-1}). \tag{41}$$

4.3. IEEE 754 format formulation

Consider the hybrid number, Y, which is coming from the output of the addition/subtraction module. The mathematical representation of the hybrid number, Y, can be expressed as:

$$Y = \sum_{j=0}^{m} y_j 2^j + \sum_{j=-1}^{-k} y_j 2^j, \quad (42)$$

$$= 2^m \left(1 + \sum_{j=-1}^{-m} y_{m+j} 2^j + \sum_{j=-(m+1)}^{-(m+k)} y_{m+j} 2^j \right), \quad (43)$$

$$= 2^m \left(1 + \sum_{j=-1}^{-(m+k)} y_{m+j} 2^j \right). \quad (44)$$

According to IEEE-754 single precision format, Y can be represented as:

$$Y = (-1)^s 2^{E_y - E_b} (1.M) = 2^{E_y - E_b} (1 + 0.M). \quad (45)$$

Here, s is the sign bit and $E_b = 127$ is the bias mentioned in IEEE-754 format. By comparing Eqs. (44) and (45), it can be written as $m = E_y - E_b$ thus:

$$E_y = m + E_b,$$

and:

$$0.M = \sum_{j=-1}^{-(m+k)} y_{m+j} 2^j.$$

For IEEE-754 format, 'm is of 8 bits' and 'M is of 23 bits', i.e. $m + k = 23$. In the IEEE-754 format generator circuit, the maximum power of the radix indicates the most significant '1' positioned through the left shift operation as shown in Figure 2. The decrementer, initialized by the maximum power of the radix, indicates the MSB. After each of the iterations, the decrementer is decremented if the searched bit is '0'. The decrement operation stops when the searched bit is '1'. The output of the decrementer provides the unbiased exponent (m), which is added to the bias (E_b) to obtain the biased exponent (E_y). The content of the shifter is the mantissa (M), which is fed to the rounding module to maintain a 23 bit single precision floating point format by eliminating the extra bits.

4.4. Round towards zero schemes

The mathematical expression and hardware implementation of the rounding (round towards zero schemes) are shown below (Eqs. (46)-(48)):

4.4.1. Mathematical description of rounding scheme

Let us assume a k-bit floating point number, S. Mathematically, S can be expressed as: $S = \sum_{i=-1}^{-k} x_i 2^i$. If S is rounded up to m bit through the round towards zero scheme, then, S can be rewritten as:

$$S = \sum_{i=-1}^{-k} x_i 2^i = \sum_{i=-1}^{-m} S_i 2^i + \sum_{i=-(m+1)}^{-(m+n)} S_i 2^i, \quad (46)$$

where, $k = m + n$. Eq. (46) can be rewritten as:

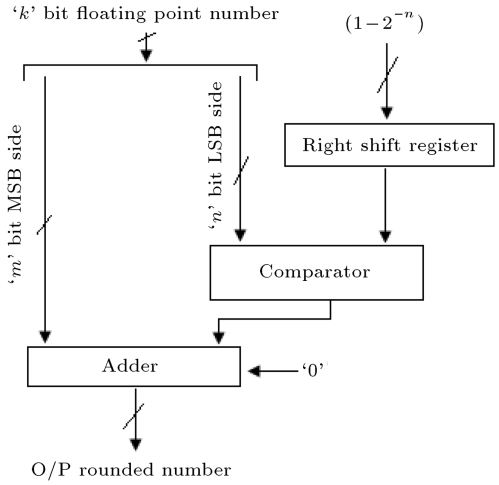


Figure 5. Hardware implementation of round towards zero scheme.

$$S = \sum_{i=-1}^{-m} S_i 2^i + 2^{-m} \sum_{i=-1}^{-n} S_{-m+i} 2^i. \quad (47)$$

Let us consider another variable, T, where $T = \sum_{i=-1}^{-n} S_{-m+i} 2^i \leq 1 - 2^{-n}$. If $T < (1 - 2^{-n})/2$, replace the value $T = 0$ and if $T \geq (1 - 2^{-n})/2$, replace the value $T = 1$.

4.4.2. Hardware implementation of rounding scheme

Figure 5 represents the hardware implementation of the rounding (round towards zero) scheme. The input k bit number is separated by a pair of numbers ($k = m + n$), where $m = 23$, the floating point part of IEEE-754 format. Right shift registers are used here to perform division operations to compute the value $(1 - 2^{-n}/2)$. Input bits (n bits) are the input of the right shift register, and its maximum value is assumed to be $T = (1 - 2^{-n})$. The comparator block compares the values of the lower order (LSB) of n bits with $(1 - 2^{-n}/2)$, and the output of the comparator block is fed to the parallel adder as a carry in function.

4.5. LNS Decoder

Consider a floating point number, X, in IEEE-754 single precision format. $X = \log_2 Y$, where Y can be expressed as:

$$X = 2^{E_x}(1 + M_x) = 2^{E_b + e_x}(1 + M_x),$$

where, e_x and M_x are unbiased exponent and mantissa, respectively.

4.5.1. LNS decoding algorithm

The algorithm for hardware implementation of the LNS to binary number conversion can be described as:

Step 1: Subtract the bias from the biased exponent (E_x) to achieve the unbiased exponent (e_x).

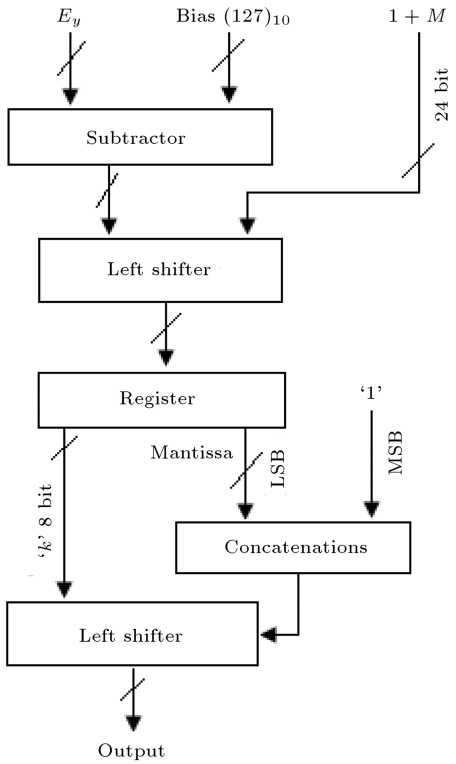


Figure 6. Hardware implementation of LNS decoder.

Step 2: Shift $(1 + M_x)$ left by 2^{e_x} times to achieve the hybrid number in LNS. In the hybrid number system, X can be expressed as $X = I + M$, where I is integer and M is the fractional part.

Step 3: Place '1' at 2^I position to get the MSB bit.

Step 4: Shift M left by 2^I position just after the MSB '1' to obtain the binary number.

4.5.2. Hardware implementation of LNS decoder

The hardware implementation for the LNS to binary number conversion algorithm is shown in Figure 6. LNS (IEEE-754 single precision format) decoder implies that the LNS to binary number system converter consists of two major sections, viz, sub-tractor and left shifter. The bias value ($E_b = 127_{10}$) is subtracted from the 8 bit exponent by the sub-tractor module. The output of the subtraction module is fed to the left shifter (first) as select inputs. The second input to the left shifter is the mantissa concatenated with '1' at the left most side. Since the output of the sub-tractor is '8' bit, the size of the left shifter (first) must be $(2^8 = 256)$ bits. The output of the left shifter (first) is stored in the register, i.e. the integral part and the floating point part is separately stored in the register. The '8' bit integral part from the register is again fed to the left shifter (second) as select inputs. Again, the floating point part, concatenated with '1' at the left most side, is fed to the left shifter as the second input to be shifted. Since the select input is an '8' bit, the size of the left shifter must be $(2^8 = 256)$ bits. The output of the left shifter is the desired binary output.

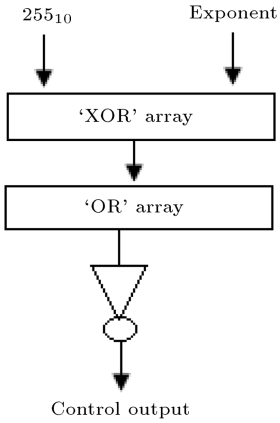


Figure 7. Architectural description of exception detector.

4.6. Exception detector

The exception detector has the task of detecting invalid number representation, such as 'infinite' or 'not a number'. The circuit level decomposition of the exception is shown in Figure 7. According to IEEE-754 single precision format, if the exponent is 255_{10}, then the number is either infinite or not a number, depending on the floating point part. XOR array and OR array are used for comparison purposes. Each bit of the exponent is compared to the given reference number (255_{10}) by the XOR array. The output is passed through the OR array. If the XOR output is '0' for all the bit positions, then the output of the OR array is '0', which means that the exponent is equal to '255_{10}'.

5. Error analysis

To compute the value of multiplication/division using LNS, generally, two types of error, i.e. conversion error and rounding error, may occur. Total error may be calculated as:

$$\% \text{ Error} = \% \text{ conversion error } (E_r) + \% \text{ rounding error } (E_{rz}). \quad (48)$$

5.1. Conversion error

The computational error can be defined as $E_r = \frac{ev - av}{ev} \times 100$, where ev has been defined as an exact value and av has been defined as an average value. Mathematically, computational error can be expressed as:

$$E_r = \frac{\log_2\left(1 + \sum_{i=-1}^{-k} x_{k+i}2^i\right) - \sum_{i=-1}^{-k} x_{k+i}2^i}{k + \log_2\left(1 + \sum_{i=-1}^{-k} x_{k+i}2^i\right)} \times 100, \quad (49)$$

$$= \frac{\ln\left(1+\sum_{i=-1}^{-k} x_{k+i}2^i\right)\times \log_2 e - \sum_{i=-1}^{-k} x_{k+i}2^i}{k+\log_2(1+\sum_{i=-1}^{-k} x_{k+i}2^i)} \times 100. \quad (50)$$

Replacing the value of $\log_2 e = 1/\ln 2 = 1.443$, Eq. (58) can be reformulated as:

$$E_r \cong \frac{\ln\left(1+\sum_{i=-1}^{-k} x_{k+i}2^i\right)\times 1.443 - \sum_{i=-1}^{-k} x_{k+i}2^i}{k+\ln\left(1+\sum_{i=-1}^{-k} x_{k+i}2^i\right)\times 1.443} \times 100, \quad (51)$$

$$\cong \frac{(1.443-1)\sum_{i=-1}^{-k} x_{k+i}2^i}{k+\ln\left(1+\sum_{i=-1}^{-k} x_{k+i}2^i\right)\times 1.443} \times 100, \quad (52)$$

$$\cong \frac{44.3\times \sum_{i=-1}^{-k} x_{k+i}2^i}{k+\ln\left(1+\sum_{i=-1}^{-k} x_{k+i}2^i\right)\times 1.443}, \quad (53)$$

$$\cong \frac{44.3}{1.443+\frac{k}{\sum_{i=-1}^{-k} x_{k+i}2^i}} = \frac{44.3}{1.443+f_k}, \quad (54)$$

where, $f_k = \frac{k}{\sum_{i=-1}^{-k} x_{k+i}2^i}$. Now, it is clear that $f_k = \frac{k}{\sum_{i=-1}^{-k} x_{k+i}2^i} \geq \frac{k}{(1-2^{-k})}$ and $f_k = \frac{k}{2^{-k}}$, which implies that $k/1-2^{-k} \leq f_k \leq \frac{k}{2^{-k}}$. So, the maximum value of f_k is $\frac{k}{2^{-k}}$ and the minimum value is $\frac{k}{1-2^{-k}}$. The minimum value of E_r is $\frac{44.3}{1.443+k/2^{-k}}$ and the maximum value is $\frac{44.3}{1.443+k/1-2^{-k}}$.

5.2. Rounding error

The rounding error for the computation of rounding towards zero schemes can be formulated as:

$$E_{rz} = \frac{S_{\text{exact}} - S_{\text{rounded}}}{S_{\text{exact}}} \times 100. \quad (55)$$

If $\sum_{i=-1}^{-m} S_{-m+i}2^i < 1-2^{-n}/2$ then:

$$E_{rz} = \frac{\sum_{i=-1}^{-m} S_i 2^i + 2^{-m}\sum_{i=-1}^{-n} S_{-m+i}2^i - \sum_{i=-1}^{-m} S_i 2^i}{\sum_{i=-1}^{-m} S_i 2^i + 2^{-m}\sum_{i=-1}^{-n} S_{-m+i}2^i} \times 100, \quad (56)$$

$$E_{rz} = \frac{2^{-m}\sum_{i=-1}^{-n} S_{-m+i}2^i}{\sum_{i=-1}^{-m} S_i 2^i + 2^{-m}\sum_{i=-1}^{-n} S_{-m+i}2^i} \times 100, \quad (57)$$

$$\cong \frac{1}{1+\frac{2^m\sum_{i=-1}^{-m} S_i 2^i}{\sum_{i=-1}^{-n} S_{-m+i}2^i}} \times 100. \quad (58)$$

From Eq. (58), it is clear that $2^m\sum_{i=-1}^{-m} S_i 2^i$ is a fixed point number, whereas $\sum_{i=-1}^{-n} S_{-m+i}2^i$ is a floating point number. E_{rz} can be minimized, assuming $2^m\sum_{i=-1}^{-m} S_i 2^i \gg \sum_{i=-1}^{-n} S_{-m+i}2^i$. If $\sum_{i=-1}^{-m} S_{-m+i}2^i \geq \frac{1-2^{-n}}{2}$, then:

$$E_{rz} = \frac{\sum_{i=-1}^{-m} S_i 2^i + 2^{-m}\sum_{i=-1}^{-n} S_{-m+i}2^i - \sum_{i=-1}^{-m} S_i 2^i - 2^{-m}}{\sum_{i=-1}^{-m} S_i 2^i + 2^{-m}\sum_{i=-1}^{-n} S_{-m+i}2^i} \times 100, \quad (59)$$

$$E_{rz} = \frac{2^{-m}\times\left(\sum_{i=-1}^{-n} S_{-m+i}2^i - 1\right)}{\sum_{i=-1}^{-m} S_i 2^i + 2^{-m}\sum_{i=-1}^{-n} S_{-m+i}2^i} \times 100, \quad (60)$$

$$= \frac{1}{1+\frac{\sum_{i=-1}^{-m} S_i 2^i + 2^{-m}}{2^{-m}\sum_{i=-1}^{-n} S_{-m+i}2^i - 1}} \times 100, \quad (61)$$

$$= \frac{1}{1+\frac{2^m\sum_{i=-1}^{-m} S_i 2^i + 2^{-m}}{\sum_{i=-1}^{-n} S_{-m+i}2^i - 1}} \times 100. \quad (62)$$

From Eq. (62), it is clear that $2^m(\sum_{i=-1}^{-m} S_i 2^i + 2^{-m})$ is a fixed point number, whereas $(\sum_{i=-1}^{-n} S_{-m+i}2^i - 1)$ is a negative floating point number. So, E_r is negative and can be minimized if $2^m(\sum_{i=-1}^{-m} S_i 2^i + 2^{-m}) \gg (\sum_{i=-1}^{-n} S_{-m+i}2^i - 1)$. In order to evaluate the average error, the proposed algorithm is applied to all combinations of n-bit non-negative numbers, and the average error is calculated from:

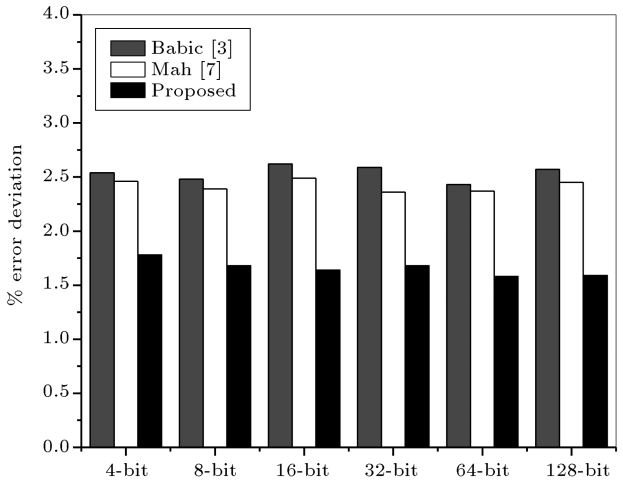


Figure 8. Error deviation graph as a function for input number of bits.

$$ae = \frac{1}{N}\sum_{i=1}^{N}(E_r + E_{rz}), \tag{63}$$

where N is the number of multiplications performed. For example, for 10-bit numbers, all the combinations of numbers ranging from 1 to 1027 are multiplied and the average error is calculated.

Figure 8 shows that the percentage of deviation of error for the computation of the LNS conversion scheme leads towards the multiplication/division implementation of different schemes, like Babic et al. [3], Mahalingam (Mah) [7], and the proposed algorithm, with respect to the actual logarithmic value. As shown in Figure 8, average deviation is almost $\sim 1.54\%$ in our proposed algorithm based implementation, with a decreasing trend of percentage error, with increasing input bits. So, it can be envisaged that from an accuracy point of view, the proposed algorithm provides a considerable amount of precision compared to other normalized implementations, such as Babic et al. (~ 2.54), and Mah (~ 2.3) based implementations.

6. Results and discussion

Transistor level simulation was performed using a Spice Spectre simulator using 90 nm CMOS technology with a 1 volt power supply. A dual threshold voltage (V_T) operating mode was considered for simulation to determine the performance parameters.

6.1. Propagation delay analysis

The hardware cost of the architecture can be computed based on the number of complex operations performed in its critical path. Hence, total propagation delay can be estimated. The reported architecture for multiplication/division using LNS has three major sub-sections, viz. (i) LNS encoder; (ii) addition/subtraction unit; and (iii) LNS decoder. So, the total latency can be computed in terms of the propagation delay of the individual sub-section. The total propagation delay of the proposed architecture (t_{pd}) can be computed as:

$$t_{pd} = t_{\ln se} + t_{adsb} + t_{\ln sd}, \tag{64}$$

where:

$t_{\ln se}$	Propagation delay of LNS encoder (in IEEE-754 single precision format);
t_{RM}	Propagation delay of rounding module;
t_{adsb}	Propagation delay of Addition/Subtraction Unit (ASU);
$t_{\ln sd}$	Propagation delay of LNS decoder.

6.1.1. Propagation delay of LNS encoder ($t_{\ln se}$)

$$t_{\ln se} = 2t_{shft} + 2t_{dec} + t_{\mathrm{adder}} + t_{\mathrm{RM}} + t_{\mathrm{ED}} + t_{dmx}, \tag{65}$$

where:

t_{shft}	Propagation delay of left shifter;
t_{dec}	Propagation delay of decrementer;
t_{RM}	Propagation delay of rounding module;
t_{adder}	Propagation delay of adder;
t_{ED}	Propagation delay of exception detector;
t_{dmx}	Propagation delay of demultiplexer.

Since the number of input bits is 'n', the maximum number of checking operations needed for searching the first '1', starting from MSB, is 'n' (if the LSB is '1' and it is preceded by '0's). Here, n must not exceed '128'. So, propagation delay due to the shifting operation can be computed as $t_{shft} = 128t_{\mathrm{MUX}} = 128t_{\mathrm{XOR}}$. For a similar reason, $t_{dec} = 128\log_2 128 \times 2t_{\mathrm{XOR}} = 1792t_{\mathrm{XOR}}$, since a maximum of '$n$' times the decrement operation is executed and we need the $\log_2 n$ number of full adders for the decrement and also the carry propagation delay for each full adder is equal to $2t_{\mathrm{XOR}}$. From the architecture, it is clear that we need an 8 bit adder to add the final result from the decrementer and the bias value, which is 7 bit. So, t_{adder} can be expressed as $t_{\mathrm{adder}} = 8t_{\mathrm{FA}} = 8 \times 2t_{\mathrm{XOR}} = 16t_{\mathrm{XOR}}$. The exception detector is nothing but a comparator, one of which checks whether the exponent is equal to '255' or not. Since '255' is an 8 bit number, the left most comparator must be '8' bit. From the architecture of the 8 bit comparator, it is clear that '8' 2 input XOR gates are required in parallel, followed by seven 2 bit OR gates in three stages and one inverter at the end. So, t_{ED} can be calculated as $t_{\mathrm{ED}} \cong t_{\mathrm{XOR}} + 3t_{\mathrm{OR}} = 2t_{\mathrm{XOR}}$. Demultiplexers are used to select the exact result or the invalid result. Here, 8 bit and 23 bit demultiplexers have been used in parallel.

So, the total propagation delay that will arise is for only one demultiplexer. Again, a two-to-one demultiplexer is constructed of one inverter and two AND gates in parallel, though one AND gate is activated at a time. So, the net delay t_{dmx} is equal to one AND gate delay, which can be neglected compared to one XOR gate, i.e., $t_{dmx} \cong 0$. From the architecture of the rounding module, it is clear that the required number of bits for the floating point side is '23', whereas the number of floating point bits from the LNS Encoder is '128'. So, we have to truncate the remaining $(128 - 23 = 105)$ bits by the round towards zero scheme. From the architecture, it is clear that ($k = m + n = 128$, $m = 23$ and $n = 105$). Since one input to the right shifter is $(1 - 2^{-105})$, we need a '106' bit right shifter for one bit right shift. Thus, propagation delay corresponding to single right shifting equals t_{XOR}. A comparator has been implemented through a subtractor. Here, we require a '105' bit sub-tractor because $((1 - 2^{-105})/2)$ is of '105' bit (MSB is '0'). Thus, the propagation delay corresponding to the comparison is $t_{\text{comp}} = 105 \times 2t_{\text{XOR}} = 210t_{\text{XOR}}$. Again, a '23' bit number is fed to the adder. So, the propagation delay corresponding to the adder can be expressed as $t_{\text{adder}} = 23 \times 2t_{\text{XOR}}$. So, the total propagation delay corresponding to the rounding module is $t_{\text{RM}} = t_{\text{XOR}} + 210t_{\text{XOR}} + 46t_{\text{XOR}} = 257t_{\text{XOR}}$. Thus, t_{inse} can be approximated as the summation of the individual module delay, i.e., equal to $2159t_{\text{XOR}}$.

6.1.2. Propagation delay of CSD addition/subtraction unit (t_{adsb})

To avoid carry propagation delay, an addition/subtraction unit has been designed on the basis of the Canonical Signed Digit (CSD) approach. From Eqs. (36)-(41), the propagation delay calculation can be performed for one bit. We need a conversion circuit that will generate the binary output from the CSD result. So, from the architecture, it is clear that $t_{z_i} \cong 2t_{\text{XOR}}$. Thus, $t_{sum_i} = t_{z_i} + t_{\text{XOR}} \cong 3t_{\text{XOR}}$, $t_{\text{signsum}_i} \cong 3t_{\text{XOR}}$ and $t_{\text{CSDadd}} = t_{sum_i} = 3t_{\text{XOR}}$. Again, the CSD result is fed to the Half Adder (HA) and the result from the HA is fed to the sub-tractor. So, t_{adsb} can be approximated as $t_{adsb} = t_{csdadd} + t_{\text{HA}} + 32t_{\text{sub}} \cong 3t_{\text{XOR}} + t_{\text{XOR}} + 32t_{\text{XOR}} \cong 100t_{\text{XOR}}$.

6.1.3. Propagation delay of LNS decoder

As the description taken from Figure 7, the first sub-tractor must be '8' bit (since the maximum exponent value is '254', which is '8' bit). So, the propagation delay corresponding to the first subtraction operation is $t_{\text{sub}} = 8 \times 3t_{\text{XOR}} = 24t_{\text{XOR}}$. Again, from the architecture, it is clear that the first left shifter must be a minimum of '24' bits and the select input for shifting must not exceed '2^{23}'. The output from the sub-tractor may have a maximum value of $(255_{10} - 127_{10} = 128_{10})$, which is '8' bit. So, the maximum length of the left shifter must be $2^8 = 256$ bits. So, we need 256-to-1 multiplexers. Since an n-to-1 multiplexer is constructed of '$(n-1)$ 2-to-1' multiplexers, here, for a 256-to-1 multiplexer, we need '255' 2-to-1 multiplexers. That is why the total propagation delay for the first shifting can be expressed as $t_{shft} = 255t_{MUX} \cong 255t_{\text{XOR}}$. Similarly for the second left shifter, we know that select inputs for shifting are of '8' bits. So, we need 256-to-1 multiplexers, which are made of '255' 2-to-1 multiplexers. So, from Eq. (15), propagation delay caused by the second left shifter can be formulated as $t_{shft} = 255 \times t_{\text{MUX}} = 255 \times t_{\text{XOR}}$. So, total propagation delay corresponding to the LNS decoder is $t_{\ln sd} = t_{\text{sub}} + 2t_{shft} = (16 + 2 \times 255)t_{\text{XOR}} = 526t_{\text{XOR}} t_{\ln sd} = t_{\text{sub}} + 2 \times t_{shft} = (16 + 2 \times 255)t_{\text{XOR}} = 526t_{\text{XOR}}$. Thus, from Eq. (65), propagation delay for the proposed algorithm, t_{pd}, can be formulated as $t_{pd} = (2195 + 100 + 526)t_{\text{XOR}} = 2821t_{\text{XOR}}$.

6.2. Results and discussions

In designing the logarithmic multiplier, all individual modules, such as gates, barrel shifter, and adder/subtractor were implemented using TG to make the circuit faster. Lowering supply voltage reduces power dissipation in quadratic fashion and becomes attractive. Though low supply voltage affects delay, it is compensated for by the lower RC delay of the TG circuit and dual threshold CMOS technology. It is also to be noted that each TG circuit requires a lower number of transistors than conventional CMOS implemented circuits, thus reducing the layout area. To evaluate the performance parameters, we give the values of computational effort using the CSD technique. Input data was taken in a regular fashion for experimental purposes. Delay and power were measured using the worst-case pattern and from the output, where the delay is maximum. The individual performance parameters, such as propagation delay, dynamic switching power consumption for circuits, i.e. LNS to binary (LNS encoder), CSD adder/subtractor and binary to logarithmic number conversion (LNS decoder), are shown in Figure 9.

We focused our main concentration on reducing propagation delay, and dynamic switching power consumption. Figure 10 indicates the performance parameters, such as propagation delay, and dynamic switching power consumption analyses of the logarithmic multiplier. Figure 11 represents the layout of the proposed logarithmic multiplier, with an area of only ~ 25 mm^2. All the mentioned designs have been simulated using the same technology, the spice spectre simulator, for comparison purposes. From simulation result analysis, we can claim that incorporation of TG with dual threshold voltage CMOS technology may

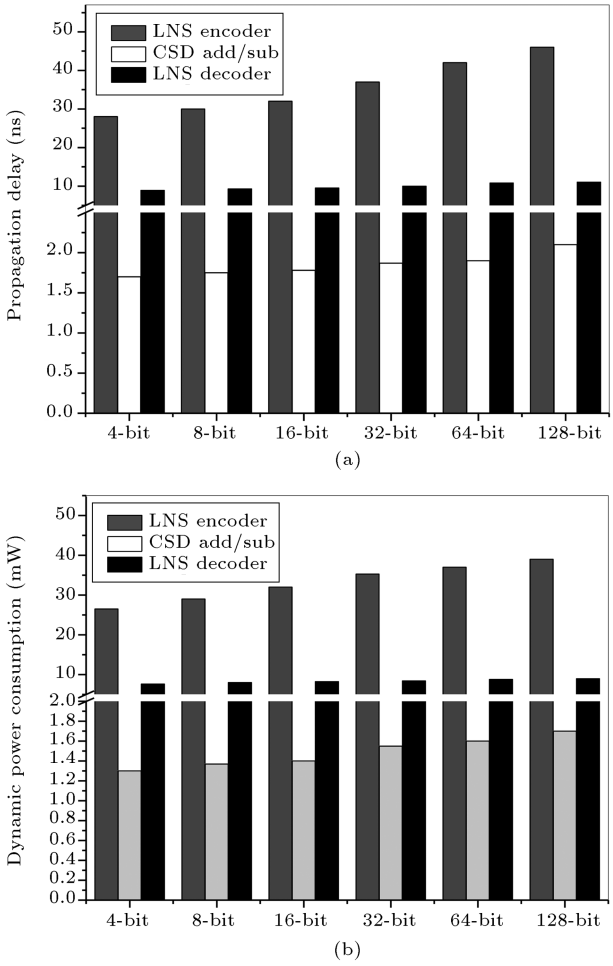


Figure 9. (a) Propagation delay (ns) analysis for individual circuitry like LNS encoder, CSD adder/subtractor, LNS decoder as a function of input number of bits. (b) Average dynamic switching power (μW) analysis for individual circuitry like LNS encoder, CSD adder/subtractor, and LNS decoder as a function of input number of bits.

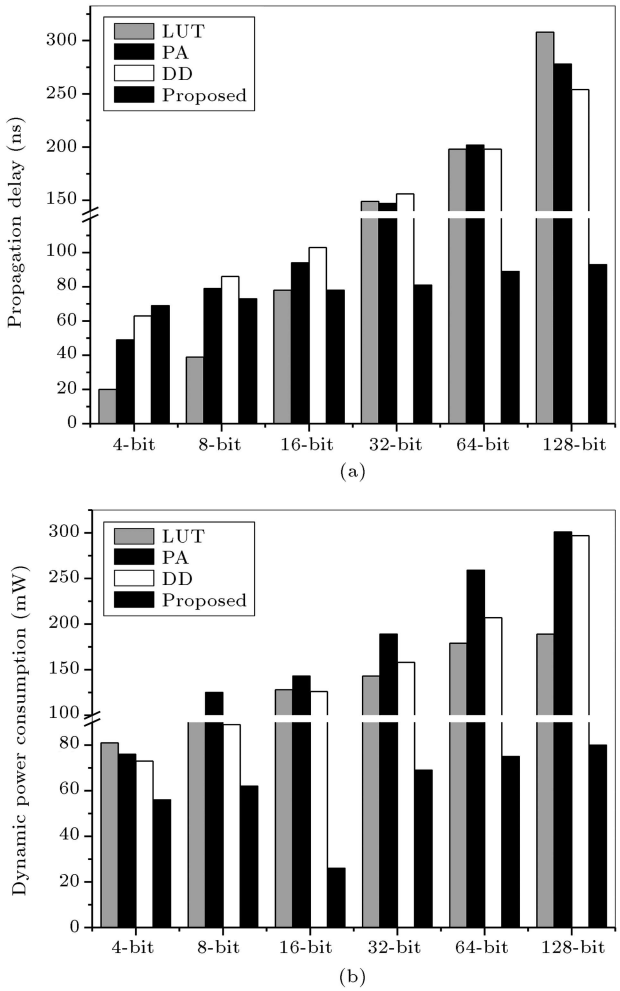


Figure 10. (a) Propagation delay (ns) analysis for multiplier as a function of input number of bits. (b) Average dynamic switching power (mW) analysis for multiplier as a function of input number of bits.

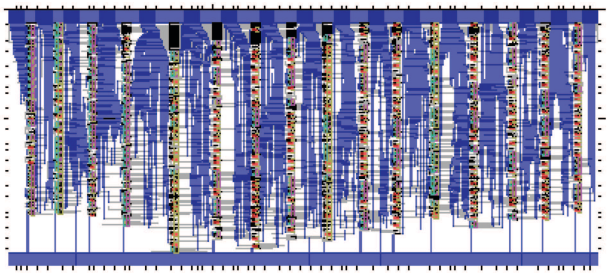


Figure 11. Layout of the proposed (128 × 128) bit multiplier circuitry which have implemented LNS encoding and decoding methodology. The layout has implemented T-Spice V-13 simulator, with ∼ 25 mm^2 layout area.

be a plausible choice for the future technology of a logarithmic multiplier.

7. Conclusions

A high speed multiplier, using a high accuracy logarithmic number conversion methodology, was designed for practical digital signal processors. Multiplication of higher order bits requires a large number of hardware components, due to the generation and processing of huge partial products. In these schemes, partial product handling was avoided by using LNS, where multiplication reduces to direct addition. The improvement in speed, by avoidance of carry propagation, was achieved through Canonical Signed Digit Code (CSDC) implementation, while high accuracy was taken care of by the floating point conversion methodology. An error minimization technique was especially considered for this purpose. The implementation results of the proposed 128 × 128 bit multiplier were compared with mostly used architecture, like LUT, PA, and DD based implementation. This multiplier offered ∼ 39%, ∼ 42% and ∼ 44% improvement, in terms of propagation delay, in comparison with LUT, PA

and DD based implementation. The corresponding improvement, in terms of power, was found to be $\sim 54\%$, $\sim 66\%$, and $\sim 61\%$, respectively, with reference to the above mentioned methodologies (LUT, PA, DD) for a layout area of ~ 25 mm^2, thereby, emphasizing the possibility of the scheme for low power VLSI implementation.

Abbreviations

LNS	Logarithmic Number System
CSDC	Canonical Sign Digit Code
CORDIC	COordinate Rotation DIgital Computer

References

1. Mitchell, J.N. "Computer multiplication and division using binary logarithms", *IRE Trans. on Electronic Computers*, **EC-11**(4), pp. 512-517 (1962).
2. Ramaswamy, S. and Siferd, R.E. "CMOS VLSI implementation of a digital logarithmic multiplier", *IEEE Aerospace and Electronics Conf.*, (1), Dayton, OH, pp. 291-294 (1996).
3. Babic, Z., Avramovic, A. and Bulic, P. "An iterative logarithmic multiplier", *Microprocessors and Microsystems*, **35**(1), pp. 23-33 (2011).
4. Khalid, H., Abed, K.H. and Siferd, R.E. "CMOS VLSI implementation of a low-power logarithmic converter", *IEEE Trans. on Computers*, **52**(11), pp. 1421-1433 (2003)
5. Khalid, H., Abed, K.H. and Siferd, R.E. "VLSI implementation of a low-power antilogarithmic converter", *IEEE Trans. on Computers*, **52**(9), pp. 1221-1228 (2003).
6. McLaren, D.J. "Improved Mitchell-based logarithmic multiplier for low-power DSP applications", *IEEE International SOC Conference*, pp. 53-56 (2003).
7. Mahalingam, V. and Ranganathan, N. "Improving accuracy in Mitchell's logarithmic multiplication using operand decomposition", *Computers, IEEE Trans. on Computers*, **55**(12), pp. 1523-1535 (2006).
8. Lewis, D.M. and Yu, L.K. "Algorithm design for a 30 bit integrated logarithmic processor", *9th IEEE Symp. on Computer Arithmetic*, Santa Monica, CA, pp. 192-199 (1989).
9. Yu, L.K. and Lewis, D.M. "A 30-b integrated logarithmic number system processor", *IEEE Int. J. of Solid-state Circuits*, **26**(10), pp. 1433-1440 (1991).
10. Lewis, D.M. "114 MFLOPS logarithmic number system arithmetic unit for DSP application", *IEEE Int. J. of Solid-state Circuits*, **30**(12), pp. 1547-1553 (1995).
11. Lewis, D. "An accurate LNS arithmetic unit using interleaved memory function interpolator", *11th IEEE Symp. on Computer Arithmetic*, Windsor, Ont, pp. 2-9 (1993).
12. Koren, I. and Zinaty, O. "Evaluating elementary functions in a numerical co-processor based on rational approximations", *IEEE Trans. on Computers*, **39**(8), pp. 1030-1037 (1990).
13. Lee, B. and Burgess, N. "A parallel look-up logarithmic number system addition/subtraction scheme for FPGA", *Proc. IEEE International Conference on Field Programmable Technology*, pp. 76-83 (2003).
14. Hart, J.F., Cheney, E.W., Lawson, C.L., Maehly, H.J., Mesztenyi, C.K., Rice, J.R., Thacher, H.G., Thacher, C. and Witzgall, Jr H.G., *Computer Approximations*, Wiley, New York (1968).
15. Kantabutra, V. "On hardware for computing exponential and trigonometric functions", *IEEE Int. J. on Computers*, **45**(3), pp. 328-339 (1996).
16. Chen, C. and Chen, R. "Performance-improved computation of very large word-length LNS addition/subtraction using signed-digit arithmetic", *IEEE Int. Conf. on Application-Specific Systems*, Architectures and Processors, pp. 337-347 (2003).
17. Fu, H., Mencer, O. and Luk, W. "FPGA designs with optimized logarithmic arithmetic", *IEEE Trans. on Computers*, **59**(7), pp. 1000-1006 (2010).
18. Volder, J. "The CORDIC trigonometric computing technique", *IRE Trans. on Electronic Computing*, **EC-8**(3), pp. 330-334 (1959).
19. Saha, P., Banerjee, A., Banerjee, I. and Dandapat, A. "High speed low power floating point multiplier design based on CSD (Canonical Sign Digit)", *IEEE symposium on VLSI Design and Testing*, VDAT (2010).
20. Avizienis, A. "Signed-digit number representations for fast parallel arithmetic", *IRE Trans. on Electronic Computers*, **EC-10**(3), pp. 389-400 (1961).
21. Deschamps, J.P., Bioul, G.J.A. and Sutter, G.D., *Synthesis of Arithmetic Circuits, FPGA, ASIC and Embedded Systems*, Wiley Interscience Publications, New Jersey (2006).
22. Khandekar, P.D. and Subbaraman, S. "Low power 2:1 MUX for barrel shifter", *1st IEEE Int. Conf. on Emerging Trends in Engineering and Technology*, Nagpur, India, pp. 404-407 (2008).
23. Saha, P., Banerjee, A. and Dandapat, A. "High speed low power factorial design in 22 nm technology", *AIP Int. Conf. on Nanomaterials and Nanotechnology*, Guwahati, India, pp. 294-301 (2009).
24. Chen, C. and Cheng, K. "An efficient exponential algorithm with exponential convergence rate", *IEEE Euromicro Symp. on Digital System Design*, pp. 548-555 (2004).

Biographies

Prabir Saha was born in Kolkata, India, on February 1980. He received BTech degree from AMIETE in 2003, and MTech from Tezpur University in 2008. Presently he is pursuing PhD at Bengal Engineering and Science University, Shibpur, Howrah, India. His research interest includes VLSI design, Digital Signal Processing and Digital Image Processing.

Arindam Banerjee received a MTech degree from West Bengal University of Technology, West Bengal, India, in 2008. Presently, he is Lecturer in the Department of Electronics and Communication Engineering at the JIS College of Engineering in Kalyani, India. His current research interests are low power VLSI design, digital signal processing and digital image processing.

Anup Dandapat received BS and MS degrees in Electronics from the University of Calcutta, India, in 2002 and 2004, respectively, and a PhD degree from the Department of Electronics and Telecommunication Engineering at Jadavpur University, India, in 2008. His current research interests are in the areas of digital and low-power VLSI design. He has authored about twenty research papers in reputed journals, and has presented about 25 papers at national and international conferences. He is currently Associate Professor and Head of the Department at the National Institute of Technology, Meghalaya, India.

Partha Bhattacharyya received a BE degree (Electronics and Telecommunication Engineering), an ME degree (Electron Devices) and a PhD degree (MEMS based gas sensor and its integration with CMOS circuits) from Jadavpur University, Kolkata, India, in 2002, 2004 and 2008, respectively. Presently, he is Assistant Professor in the Department of Electronics and Telecommunication Engineering at Bengal Science University, Shibpur, India. His current research interests include nanomaterial based sensors, MEMS-based chemical sensors and CMOS integration and low power VLSI design. He has published about seventy five research articles in reputed national and international journals and conferences. He received the Young Engineer's Award from the Institution of Engineers, India, in 2010, the Career Award for Young Teachers (CAYT) in 2011-12 from the All India Council for Technical Education (AICTE), the Young Engineer Award in 2012 from the Indian National Academy of Engineering (INAE) and the Young Scientist Award in 2012 from the Indian National Science Academy (INSA) for his teaching and research contributions.