International Journal of Recent Technology and Engineering (IJRTE)
ISSN: 2277-3878, Volume-8 Issue-3, September 2019

Radix-10 Fixed Point Division Hardware

Diganta Sengupta, Mahamuda Sultana, Atal Chaudhuri

Abstract*: Standardization of decimal floating-point formats by IEEE in IEEE 754-2008 Standards fuelled the interest on decimal floating-point architectures among the global research community. Although decimal arithmetic architecture research attracted computer scientists for the last two decades, the major thrust was observed past the year 2008. Multiple proposals have been witnessed for decimal arithmetic units, mostly adders/subtractors, and multipliers. Very few designs have been proposed in the division domain. This article proposes decimal division hardware based on sutras from Vedic Mathematics, the ancient mathematics system. We present a Reduced Magnitude Divisor Generator which converts each digit of the actual divisor into a reduced digit set [-5, 5] using a unique combination/modification of the Vedic Sutras. The divisor digit magnitude reduction also minimizes the product set of multiplication as the single-digit multiplier belongs to the reduced digit set [0, 5] barring the sign. The sign of the dividend or the divisor is not attended during division as a simple XOR operation on the two signs provides the sign of the quotient. Peer comparison has exhibited better results for our design in terms of space and time.

Keywords: Decimal division, Vedic division, Vedic sutras, Division architecture, Reduced Magnitude Divisor Generator, IEEE 754-2008.

I. INTRODUCTION

Division is generally observed to be a low priority operation with respect to its counterparts, viz. addition, subtraction, and multiplication, by the processor designers. Thereby chip and resource allocation get affected in addition to development efforts. Most of the research on decimal division has primarily focused on iterative procedures for concise approximation of the divisor reciprocal using Newton-Raphson or other such recurrence expressions. Vedic mathematics provides a few sutras which deal with division procedure either standalone or in combination with other sutras and corollaries. The sutras generate reduced magnitude operands and eliminate subtraction from the whole process of division, altogether making the division process faster. Software implementation for division using Vedic sutras has been observed in [1] [2] [3] [4]. We provide a hardware realization for the *Vedivision Algorithm* presented in [3]. The algorithm is generic and consists of contribution from two sutras from Vedic Mathematics – the *Nikhilam* and the *Paravartya* sutras. Limitations imposed by both the *Nikhilam* and the *Paravartya* Sutra regarding the magnitude of the divisor compelled the design of the generic algorithm.

The rest of the paper is arranged as follows. The next section provides the related work. Section 3 presents the two Vedic sutras which have been used – *Nikhilam* and *Paravartya* sutra concluding with the algorithm - *Vedivision*. Section 4 presents the proposed hardware design and details. Section 5 provides the performance analysis followed by the conclusion.

II. RELATED WORK

Decimal division hardware inventions have mostly been patented [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16] [17]. The decimal division is realized using *Millicode* (vertical microcode) instructions as a sequence of simple operations [6]. *Millicode* instructions are a special type of instructions in IBM Power4 Systems [18]. These instructions are dispatched in groups having a maximum of five IOPs (internal operations). The decimal division is split into *Millicodes* (smaller instructions) and the most significant digit of the normalized divisor and the partial remainder generates the *trial quotient* using digit recurrence. This *trial quotien*t so obtained accesses the *Dividend Multiple Table* wherein trial quotient adjustments are determined using a simple addition/subtraction. Nikmehr, Phillips, and Lim [19] have proposed a fast decimal floating-point division based on the high radix SRT division [20] [21] [22]. Research in [23] and [24] also provided VLSI implementations based on SRT digit recurrence division algorithms. *Quotient Digit Selection (QDS)* [25] function complexity governs the efficiency of an SRT divider; where the QDS function calculates a fixed number of bits of the quotient every iteration and is implemented using a lookup table. In [19], the partial remainder is represented in *Decimal Signed Digit (DSD)* format named by the authors and the SRT recurrence is performed using *Decimal Carry Free (DCF)* addition [26]. Lang and Nannarelli have offered a decimal digit recurrence division unit in [27] where the authors decompose the quotient digit into a radix 5 and a radix 2 component with values of radix 5 component in {-2, -1, 0, -1, 2}. Other state of the art proposals in literature are provided in [28] [29] [30] [31] [32] [33] [34] [35]. Gorgin and Jaberipur have addressed the important issues of storage of redundant results and intermixed operations in decimal arithmetic in [36]. The taxonomy of decimal misconceptions has been provided in [37].

Manuscript published on 30 September 2019
* Correspondence Author
Diganta Sengupta*, Dept. of Computer Science and Engineering, Techno International Batanagar, Kolkata, West Bengal 700141, India, sg.diganta@gmail.com
Mahamuda Sultana, Dept. of Information Technology, Techno International New Town, Kolkata, West Bengal 700150, India, sg.mahamuda@gmail.com
Atal Chaudhuri, Vice-Chancellor, Veer Surendra Sai University of Technology, Sambalpur, Odisha 768018, India, atalc23@gmail.com

Bouvier and Zimmermann have proposed a clever binary to decimal conversion technique void of any division in [38] using a sub-quadratic algorithm. This technique has been intelligently attended to in decimal arithmetic calculations performed on binary platforms and thereafter decimal corrected.

A combined binary and decimal floating point divider has been proposed in [39] dividing decimal numbers in the *Binary Integer Decimal (BID)* format. The authors in [39] report that their combined *BFP/BID* divider design has the same clock period and same latency for *BID* division with respect to a standard *BID* divider but with an overhead of 17% increase in area owing to a large binary multiplier in the normalization unit. The binary multiplier in this design can be doubled in other arithmetic operations, viz. addition, subtraction or multiplication. The state of the art decimal divider architecture proposals in literature utilize the Newton – Raphson iteration method [40] [41] [42] [43]. In the operation $Q = Y \div X$, first an initial approximation of the divisor's reciprocal is obtained; $R_0 \approx 1/X$, next *'m'* Newton Raphson iterations are performed on R_0 to obtain an improved approximation R_m . This R_m multiplies the dividend Y to obtain an approximate quotient Q' which is rounded to generate the final quotient Q in [40] [41]. The procedure in [40] and [41] demand two decimal multiplications for each iteration, doubling the size of the multiplications as each iteration quadratically increases the accuracy of the result. The accuracy of the initial linear approximation is determined by the number of Newton Raphson iterations which in turn is dictated by the look up table bucketing the slope and intercept approximations. In [42] and [43], the initial approximation of the reciprocal is obtained using a piecewise first order *minimax polynomial* [44]

The next section presents the algorithms using the two Vedic sutras – *Nikhilam* and *Paravartya*, followed by the Vedivision algorithm of [3].

III. Division Algorithms Based on Nikhilam and Paravartya Sutras, and Vedivision

This section presents the two Vedic sutras which have been used to design the generic algorithm *Vedivision* [3].

A. Division Algorithm using the Nikhilam Sutra

We provide the division algorithm using the Nikhilam Sutra in successive steps.

Step 1: This step divides the dividend in two parts – the Quotient and the Remainder; digit count of the Remainder equaling the digit count of the Divisor.

Step 2: This step is *Divisor Recoding* which is done by subtracting each digit of the divisor from '9' barring the last digit which is subtracted from '10'. This results in a *Recoded Divisor* as follows:

Let

$D_i = i^{th} Digit\ of\ the\ Divisor$

$D_i^R = i^{th} Digit\ of\ the\ Recoded\ Divisor$

$$D = \sum_{i=0}^{15} 10.D_i\ ; where\ D = Original\ Divisor \quad (1)$$

$$D^R = \sum_{i=0}^{15} 10.D_i^R ; where\ D^R = Recoded\ Divisor \quad (2)$$

$$D_i^R = \begin{cases} B - 1 - D_i\ \forall\ i \in [1,15] \\ B - D_i\ \forall\ i = 0 \end{cases} \quad (3)$$

Where B = *Radix*

Therefore, for the given random Divisor (89998), the *Recoded Divisor* is 1 0 0 0 2.

Step 3: In this step, the first digit of the quotient part is divided by the first digit of Divisor. This division is done having the dividend as well as the divisor as single digit operands which can be realized using *Read Only Memory* (*ROM*) or an equivalent *Programmable Logic Array* (*PLA*) on hardware.

This single digit division is inevitable and can be assumed to be the penalty incurred for performing 16 digit operand divisions. The *Look up Table*/*Read Only Memory* (*ROM*) furnishes the Remainder of single digit divisions. Hence, an array of 10×9 cells can be used to design the *Look up Table*/*Read Only Memory* (*ROM*) treating division by 0 differently. Treating division by 1 as special case, the array can be further reduced to 8×8 cells.

Step 4: The remainder obtained in the previous step doubles as the *Most Significant Digit* (*MSD*) of Quotient. This digit is used to multiply the *Recoded Divisor* and the product is placed below the dividend after a single digit Right Shift (henceforth denoted as $R1_{Shift}$). The $R1_{Shift}$ of the product is followed by addition with the dividend. Addition operation results in the second digit of the Quotient. The addition can be performed using *The Radix 10 Fixed Point Adder* [45] and the multiplication can be realized using the *Reduced Magnitude Partial Product Generator* in *Radix 10 Vedic Multiplier* [46] on hardware.

Step 5: The Recoded Divisor is further multiplied by the single Quotient digit obtained after addition in Step 4 and the product is placed beneath the dividend after further $R1_{Shift}$. Since the product in Step 4 is completely placed in Remainder part of Step-1, hence division concludes and all the intermediate results are added.

Note: For complete illustration of the division process used in this sub-section, readers are directed to [3]. Division using *Nikhilam* sutra provides accurate results with *High Magnitude Divisors* (*HMD*) – operands proximal to powers of Radix 10. Division involving *Low Magnitude Divisors* (*LMD*) generate incorrect results on multiple occasions, thereby reported unreliable in [47] for *Low Magnitude Divisors* (*LMD*). For LMD, *Paravartya* Sutra is beneficial and reflects exact result. This procedure is mentioned in the next sub-section.

B. Division Algorithm using the Paravartya Sutra

This sutra is capable of division using any magnitude of divisor, albeit exhibits better performance time with lower magnitude divisors; divisors non-proximal to powers of Radix-10. This sutra can be easily explained using the famous Chinese Remainder Theorem [48] [49].

1. Given a relationship $E = DQ + R$, $E = (x - p)Q + R$ also holds good if $(x - p)$ happens to be a divisor, where E = dividend, D = divisor, Q = quotient and R = remainder.
2. Now, if $(x - p)$ is equated to zero, then the expression 'E' itself becomes the remainder R, and 'p' becomes equal to 'x', with a reversal in sign for 'p'.

Hence, *Paravartya* sutra utilized the negation virtus of 'p'; i.e. if a divisor is *112*, the final *Recoded Divisor* is *-1-1-2*. Thereafter, the first digit of the *Recoded Divisor* is omitted and the remaining *-1-2* become the new *Sub Recoded Divisor*. The following steps illustrate the process.

Step 1: This step divides the dividend in two parts – the Quotient and the Remainder; digit count of the Remainder equaling the digit count of the Divisor. This step is similar to III.A.Step 1.

Step 2: This step is Divisor Recoding. Each digit of the divisor is negated to form the corresponding digits of the Recoded Divisor. Divisor Recoding is done according to the following equations:

Let $D_i = i^{th} Digit\ of\ the\ Divisor$

$D_i^R = i^{th} Digit\ of\ the\ Recoded\ Divisor$

$D = \sum_{i=0}^{15} 10.D_i\ ; where\ D = Original\ Divisor$ (4)

$D^R = \sum_{i=0}^{15} 10.D_i^R ; where\ D^R = Recoded\ Divisor$ (5)

$D_i^R = -D_i$ (6)

Therefore, for the given random Divisor 112, the *Recoded Divisor* is -1-1 -2.

Step 3: This step is similar to Step 3 of *Nikhilam* Sutra. The first digit of the quotient part is divided by the first digit of Divisor. This division is done having the dividend as well as the divisor as single digit operands which can be realized using *Read Only Memory* (ROM) or an equivalent *Programmable Logic Array* (PLA) on hardware. The difference with Step 3 of *Nikhilam* Sutra is that in *Paravartya* Sutra, the digit which has been omitted from the *Recoded Divisor* is used to divide the first digit of the quotient part; whereas in *Nikhilam* Sutra, first digit of the quotient part is divided by the first digit of the original divisor.

Step 4 and **Step 5:** These two steps are exactly similar to Step 4 and Step 5 of division using *Nikhilam* Sutra.

Note: For complete illustration of the division process used in this sub-section, readers are directed to [3]. *Vedivision* eliminates the ambiguity posted by the above mentioned two sutras. The next sub-section illustrates *Vedivision*.

C. Vedivision – The Vedic Division Algorithm

Step 1: Divisor Recoding is done in this step in such a manner that the magnitude of each of the digits in the *Recoded Divisor* belongs to the reduced digit set [-5, 5]. The recoding protocol follows both the *Nikhilam* and the *Paravartya* division recoding protocols.

Divisor Recoding follows a two-step approach on divisor digits partitioned into two digit sets – Digit Set 1 [0, 5] (henceforth referred as DS1) and Digit Set 2 [6, 9] (henceforth referred as DS2). The divisor digits are scanned from Right to Left (*LSD* to *MSD*).

1. If a digit belongs to DS1, it is negated following *Paravartya* Sutra.
2. If a digit belongs to DS2, 10's Complement (*Nikhilam*) of the digit is generated as the *Recoded Divisor* digit followed by an increment of the next *Higher Significant Digit*.

Mathematically, if

$D_i = i^{th} Digit\ of\ the\ Divisor$

$D_i^R = i^{th} Digit\ of\ the\ Recoded\ Divisor$

$D = \sum_{i=0}^{15} 10.D_i\ ; where\ D = Original\ Divisor$ (7)

$D^R = \sum_{i=0}^{15} 10.D_i^R ; where\ D^R = Recoded\ Divisor$ (8)

$$D_i^R = \begin{cases} -D_i\ \forall\ D_i \in [0,5] \\ B - D_i\ \forall\ D_i \in [6,9] \end{cases}$$ (9)

$D_{i+1}^R = D_{i+1}^R + 1\ if\ D_i \in [6,9]$ (10)

Single digit division is done ignoring the sign of the digit as had been done in *Nikhilam* and *Paravartya* Sutras using *Look-up-Table*/*ROM*. Reduced digit set *Divisor Recoding* further reduces the size of the *Look-up-Table*/*ROM* from 10×9 cells to 10×5 cells treating division by 0 differently. If division by 1 is treated as special case, then the size of the *Look-up-Table*/*ROM* is reduced from 8×8 cells to 8×4 cells. Since the proposed division architecture operates on BCD numbers having maximum magnitude of a *Recoded Divisor* digit '5', hence, only three bits are required to store the value of the digit. The fourth bit denotes the sign of the *Recoded Divisor* digit. Positive and negative digits have 0 and 1 at their *MSB* respectively.

Table- I: Conversion Table for Divisor Recoding.

Possible Divisor Digit Value					Recoded Divisor Digit Value				
C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
Decimal	d_i^3	d_i^2	d_i^1	d_i^0	Decimal	d_i^{R3}	d_i^{R2}	d_i^{R1}	d_i^{R0}
0	0	0	0	0	0/-0	**1**	0	0	0
1	0	0	0	1	-1	**1**	0	0	1
2	0	0	1	0	-2	**1**	0	1	0
3	0	0	1	1	-3	**1**	0	1	1
4	0	1	0	0	-4	**1**	1	0	0
5	0	1	0	1	-5	**1**	1	0	1
6	0	1	1	0	4	**0**	1	0	0
7	0	1	1	1	3	**0**	0	1	1
8	1	0	0	0	2	**0**	0	1	0
9	1	0	0	1	1	**0**	0	0	1

$D_i = \sum_{k=0}^{3} {d_i}^k 2^k \in [0,9]\ ; {d_i}^k = k^{th}\ bit\ of\ D_i$ (11)

$$D_i^R = \begin{cases} d_i^{R^3} = \text{Sign} \\ \sum_{k=0}^{2} d_i^{R^k} 2^k \in [0,5] = \text{Magnitude} \end{cases} ; d_i^{R^k} = k^{th}\ bit\ of D_i^R$$ (12)

$$D_{i+1}^R = \begin{cases} D_{i+1}^R + 1; if\ d_i^{R^3} = 0 \\ D_{i+1}^R;\ if\ d_i^{R^3} = 1 \end{cases}$$ (13)

It may be noted that the Most Significant Digit of the *Recoded Divisor* is always a *Negative Digit*.

Step 2: The $|MSD|$ of the Dividend is divided by the $|MSD|$ of the *Recoded Divisor*. This single digit inevitable division is performed using a 8×4 cell *Look up Table*/*ROM* as shown in Table 2. Division by '0' is dealt exclusively and division by '1' is treated as a special case.

Table- II: (8×4 cell) Look up Table / ROM for Generation of Quotient Digit

	2	3	4	5	6	7	8	9
2	1	1	2	2	3	3	4	4
3	0	1	1	1	2	2	2	3
4	0	0	1	1	1	1	2	2
5	0	0	0	1	1	1	1	1

For certain set of operands, clubbing of first two digits of the dividend are required and then the clubbed double digit is divided by the MSD of the Recoded Divisor to generate the Quotient. For such cases, Table 2 falls short and another *Look –Up Table (LUT2) / ROM2* has been designed as shown in Table 3.

Step 3: This step calls for Remainder Normalization. Normalization replaces a double digit number at each position by a single digit number. Traversal is from *Least Significant Position* to *Most Significant Position*.

The *LSD* of a Multi – Digit number is retained at the position followed by addition of the remaining digits to the next *HSD (Higher Significant Digit).*

IV. HARDWARE IMPLEMENTATION OF VEDIVISION (RADIX-10 DIVISION)

In this section we propose the hardware implementation for *Vedivision* (*Radix-10 Division*)

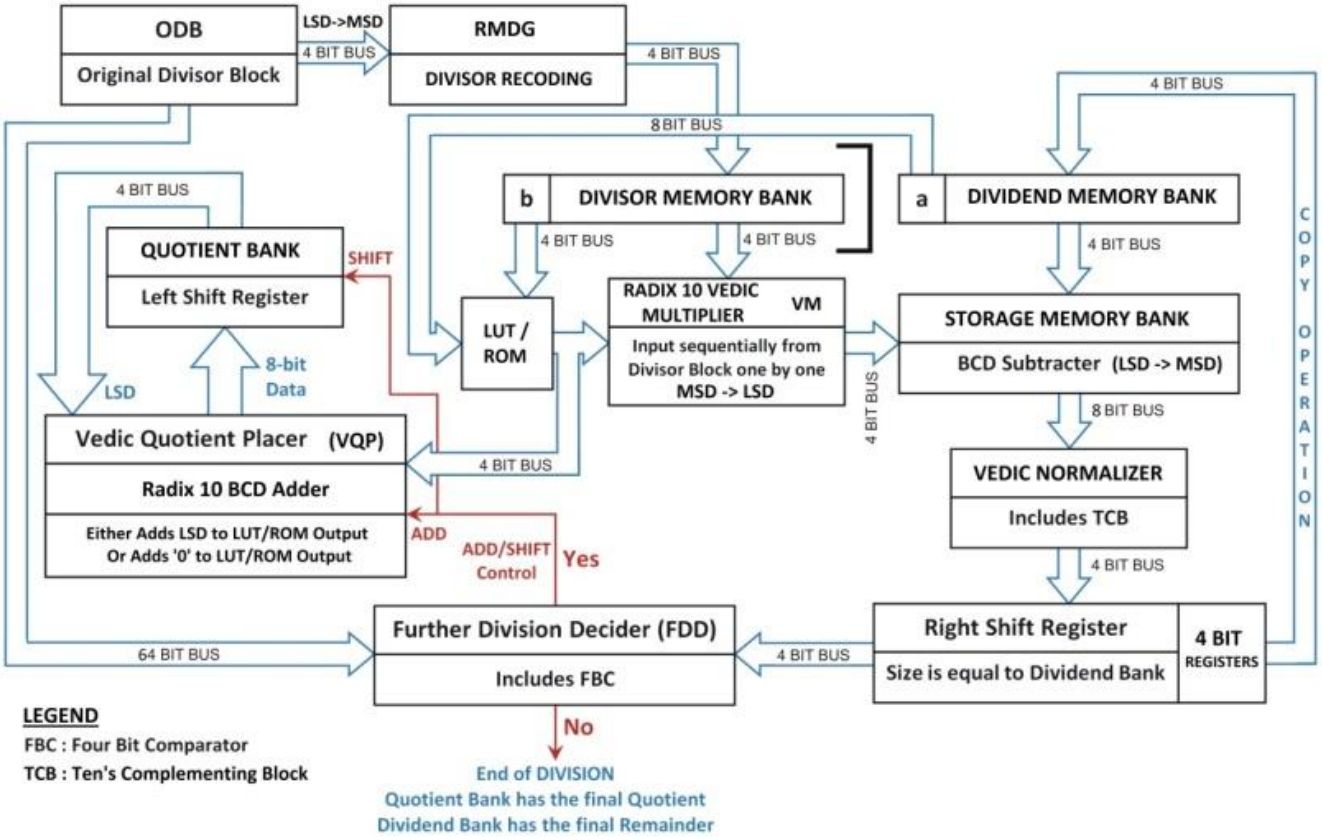

Fig. 1.High Level Diagram for Vedivision Hardware

illustrated in the previous section. We first provide the schematic for the architecture in Figure 1, and then illustrate the schematic in the subsequent sub-sections.

Table- III: Look-Up Table for Double Digit Dividend Divisions (*LUT2/ROM2*)

	10	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**
2	5	5	6	6	7	7	8	8	9	9	0	0	0	0	0	0	0	0
3	3	3	4	4	4	5	5	5	6	6	6	7	7	7	8	8	8	9
4	2	2	3	3	3	3	4	4	4	4	5	5	5	5	6	6	6	6
5	2	2	2	2	2	3	3	3	3	3	4	4	4	4	4	5	5	5
	28	**29**	**30**	**31**	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**	**41**	**42**	**43**	**44**	**45**
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	9	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	7	7	7	7	8	8	8	8	9	9	9	9	0	0	0	0	0	0
5	5	5	6	6	6	6	6	7	7	7	7	7	8	8	8	8	8	9

A. ODB – Original Divisor Block

This block is mainly a 64 bit memory cell containing sixteen divisor digits in BCD 8421 code. Since *ODB* contains the divisor, hence this block has monitored permission for data storage with validity analysis of the operand. That is, operand value '0' is blocked at the input interface itself elimination the error – 'division by 0'. If the input interface receives a '0' operand, a respective signal is sent to the error handler.

B. RMDG – Reduced Magnitude Divisor Generator

This block contains a 68 bit memory cell and results in Divisor Recoding. It accepts 64 bit data (16 digits) from the *ODB* block and generates the *Recoded Divisor* according to Equation (9) through Equation (12). Divisor recoding follows digit traversal from *LSD* to *MSD* and the recoded digits are generated using Table 1.

A simple Boolean logic generates the architecture for implementing Table 1. Since, the input may be a maximum of 16 digit operand, hence, this block has provision for storing 17 recoded digits – hence the memory cell of 68 bits (each digit consumes 4 bits – 1 for sign and three for magnitude, Equation (11)-(12)). *'n'* divisor digits produce a maximum of *'n+1' Recoded Divisor* digits. Moreover, the *MSB* of valid data in 68 bit memory is definitely 1, as the *MSD* of the *Recoded Divisor* is always a negative number. Divisors having digit count less than 16 are zero padded at the Higher Significant Positions.

From this block itself, the iterative procedure starts. The divisor digits are recoded from *LSD* for *MSD* and once a digit is recoded it is passed to the next block (Divisor Memory Bank) using a four bit data bus. Division from this block onwards follows a pipeline.

C. Divisor Memory Bank

This block contains a 68 bit memory cell for bucketing the 17 digits fed by *RMDG* Block. The input digits are fed sequentially one at a time at MSD from the *RMDG* Block with $R1_{Shift}$ for every input, in parallel processing to digit wise recoding in *RMDG* Block.

After all the Recoded Divisor digits are fed to this block, the Most Significant valid Digit is fed to the *LUT / ROM* for fetching the quotient values. Four single bit memory cells are concatenated into a 4 bit register to hold a single digit of the operand. Similarly explanation holds true for the *Dividend Memory Bank*.

D. Dividend Memory Bank

This block contains the dividend in sixteen 4 bit registers, having a total of 64 bit memory cell. The Most Significant valid Digit is fed to the *LUT / ROM* for fetching the quotient values along with that from the *Divisor Memory Bank*. This block is refreshed with a net set of data from the *Right Shift Register Block* (sub-section IV.K) after every iteration.

E. LUT / ROM

This stores the results for single digit divisions as discussed earlier and represented by Table 2 and Table 3. The choice for Table 2 or Table 3 depends upon the size of the dividend.

The data sent from the Dividend Memory Bank is mostly 4 bit single digit data, but for some operands, clubbing of two digits result in an 8 bit data being sent to the *LUT / ROM*. In such cases *LUT / ROM* corresponding to Table 3 is selected. Data is fetched from either Table 2 or Table 3 as follows: The Row Number represents the Divisor Digit and the Column Number represents the Dividend Digit.

Hence, if the *LUT / ROM* are considered as an Array, then division generates quotient as follows:

$$\frac{9}{4} \xrightarrow{Access} LUT/ROM[4,9] = Quotient\ 2, [RowNumber_4, ColumnNumber_9] \quad (14)$$

$$\frac{34}{4} \xrightarrow{Access} LUT2/ROM2[4,34] = Quotient\ 8, [RowNumber_4, ColumnNumber_{34}] \quad (15)$$

The four bit retrieved Quotient is fed to the *Radix 10 Vedic Multiplier Block* (Subsection IV.H) as well as the *Vedic Quotient Placer Block* (Subsection IV.F)

F. Vedic Quotient Placer (VQP)

This block is responsible for generating the final Quotient digit. The Quotient fetched from the *LUT/ROM* is added to the previous Quotient digit depending upon a control signal generated by the *FDD* Block (Subsection IV.L) according to Step 4 of the *Vedic Division Algorithm* (Section III). If addition is required, it is realized using the *Radix 10 Fixed Point Adder* [45].

G. Quotient Bank

This block stores the final Quotient digit post generation from the *VQP* Block. The digit enters at the LSD position, and thereafter realizes $L1_{Shift}$ with each incoming digit from *VQP* Block on mark of a control signal from the *FDD* Block (Subsection IV.L). Actually, depending upon the control signal, the *LSD* is fed back to the *VQP* Block for addition of the present generated Quotient or shifted left by one digit as the new quotient digit will be placed at the *LSD* position in this block.

H. VM Block – Radix 10 Vedic Multiplier

Digit by Digit multiplication of the *Recoded Divisor* by the Quotient Digit from the *LUT/ROM* takes place in this block. The multiplication result is fed to the *Storage Memory Block* (Subsection IV.I) via a 4 bit bus sequentially. The multiplication takes place using the *Reduced Magnitude Partial Product Generator* of the *Radix 10 Fixed Point Vedic Multiplier* Architecture [46] generating valid *BCD* result digits.

I. Storage Memory Bank

This Block receives single digit product results from the *VM Block*. BCD Subtraction (*10's Complement Addition*) with the respective digit of the dividend is performed and fed to the *Vedic Normalizer Block* (Subsection IV.J) in a pipelined fashion.

J. Vedic Normalizer

This block performs Normalization according to Section III.C. The complete Result from the *Storage Memory bank* is received in this block via 8 bit bus as the number range at each position belongs to [-45, 54] as follows:

- The range of the multiplier digit is [-5, 5]. That states the product range as [-45, 45].
- Digit range of the dividend is [0, 9]. Therefore single digit addition of a dividend digit with product from *VM Block* has a range of [-45 ± 0, 45 ± 9] = [-45, 54].

K. Right Shift Register

As the *Vedic Normalizer Block* normalizes from *LSD* to *MSD*, hence the lower significant digit post normalization is fed to this block. Therefore this block performs $R1_{Shift}$ with every digit entry. The data in this block is fed back to the *Dividend Memory Bank* (Subsection IV.D) via a write back operation and to the *Further Divider Decider Block* (Subsection IV.L) via write through mechanism.

L. FDD – Further Division Decider

The job of this block is to generate a control signal for the *Quotient Bank* and the *Vedic Quotient Placer Block*. The control signal signifies whether the present Quotient is a standalone Quotient digit or requires to be added to the previous Quotient digit to generate a valid Quotient digit. The control signal is of the form $ADD/\overline{SHIFT}$.

The data from the *Right Shift Register* is compared with the original divisor from the *Original Divisor Block* using Four bit comparators. If the original divisor is greater than the incoming data from *Right Shift Register*, then further division is required. If the digit count of the normalized data from *Right Shift Register* equals the digit count of the Recoded Divisor, then the *ADD* control signal is generated otherwise the *SHIFT* control signal is generated.

V. Performance Analysis for Radix-10 Division Hardware

We modeled the architecture presented in Figure 1 in *Verilog* at the *Register Transfer Level* using XC7A200T device (Package FBG484) of Artix7 Family. The codes were synthesized using the *ISE Simulator* of *Xilinx 14.3* on a 32 bit machine having *Intel Core i3 5005U CPU @ 2.00 GHz and 4.00 GB RAM*. Due to unavailability of the *Synopsis Design Compiler* or any *ASIC Synthesis Software* and relevant *Foundary Development Kit (FDK)* or any *Process Design Kit (PDK)*, the *Primetime Time Analysis* could not be performed. It is assumed that the dedicated CMOS implementations will be much faster than the general purpose FPGA implementation results provided in this article. The FPGA Synthesis of the proposed hardware implementation has proven to support the Software Delay given in [3]. The blocks in Figure 1 were designed independently. The final *Vedivision Hardware Architecture* has been generated connecting the interfaces of the individual blocks in Figure 1. Subsection V.A provides the Time and Area Analysis and Subsection V.B presents the Power and Temperature Analysis of *Radix-10 Division Hardware Architecture*. Since, the algorithm produces varied time for different set of operands; hence the synthesis analysis has been performed for the given random set of operands.

Operand 1: 7319842657351956 (Dividend) (Precision = 16 Digits)

Operand 2: 7083741825678 (Divisor) (Precision = 13 Digits)

A. Time and Area Analysis

Table IV presents the Synthesis Time for the major blocks in Figure 1. The graphical analysis of Table 4 is presented in Figure 2.

Table- IV: Time Analysis for Major Blocks in Figure 1

Block #	Block Name	Delay (nS)
1	Reduced Magnitude Divisor Generator	11.361
2	Vedic Normalizer	65.724
3	Ten's Complementing Block	1.639
4	Further Division Decider	5.524
5	Vedic Quotient Placer	7.713
6	Storage Memory Bank	5.133
7	Radix 10 Vedic Multiplier	6.052
8	Look Up Table / Read Only Memory	1.366
9	Quotient Bank	0.280

Table 6 presents the power and temperature analysis for the *Vedivision* Hardware.

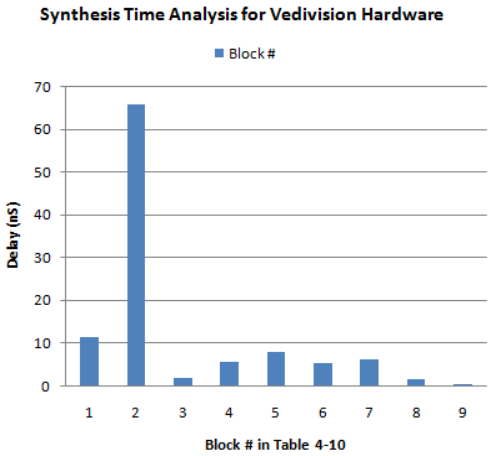

Fig. 2.Graphical Synthesis Time Analysis for Table 4

It can be observed from Table 4 as well as Figure 2 that the Vedic Normalizer Block accounts for the maximum synthesis time as discussed earlier and hence it governs the total time consumption of Radix-10 Division.

A single run for a given operand accounts for a time consumption of 104.792 nS (Table 4). Table 7 and Table 8 present the FPGA and CMOS realizations for the state-of-the-art proposals in literature respectively. Close observation of the latencies reveal our design to fare better in peer comparison.

Table- V: Component Utilization Summary for Radix-10 Division

HDL Synthesis Data		
Component	**#**	**Total**
Adders/Subtractors		162
4-bit adder	161	
5-bit subtractor	1	
Registers		16
1-bit register	1	
128-bit register	2	
4-bit register	2	
5-bit register	2	
64-bit register	3	
7-bit register	1	
Comparators		163
3-bit comparator greater	16	
4-bit comparator greater	144	
5-bit comparator equal	1	
5-bit comparator greater	1	
5-bit comparator lessequal	1	
Multiplexers		1038
1-bit 2-to-1 multiplexer	862	
128-bit 2-to-1 multiplexer	4	
32-bit 10-to-1 multiplexer	1	
32-bit 2-to-1 multiplexer	17	
4-bit 2-to-1 multiplexer	113	
5-bit 2-to-1 multiplexer	1	
64-bit 2-to-1 multiplexer	19	
7-bit 2-to-1 multiplexer	5	
8-bit 2-to-1 multiplexer	16	
Tristates		394
1-bit tristate buffer	394	

XORs		2396
1-bit xor2	1462	
3-bit xor2	387	
4-bit xor2	547	

Table- VI: Power and Temperature Analysis for Radix-10 Division Hardware (Figure 1)

Time Consumption		18.00 nS
Power Summary		I (mA)
Total Vccint	1.00 V	33
Total Vccaux	1.80 V	20
Total Vcco18	1.80 V	1
Total VccBRAM	1.00 V	1
Quiescent Vccint	1.00 V	33
Quiescent Vccaux	1.80 V	20
Quiescent Vcco25	1.80 V	1
Estimated Junction Temperature		25.2° C
Ambient Temperature		84.8° C
Theta J – A		2.5 C/W
Total Quiescent Power		0.073
Total Dynamic Power		0.000
Total Power		0.073

VI. CONCLUSION

In this article, we propose Radix-10 hardware architecture based on Vedic mathematics, the ancient mathematics system. The proposed architecture is based on a literary proposal on decimal division. Two of the Vedic sutras have been engineered to generate a generic algorithm which has been implemented on FPGA. The results reflect one process, named Normalization, which governs the time requirement for the division process on random sets of operands. One of the key designs in the proposal is the Reduced Magnitude Divisor Generator which converts the digits of the divisor into the set [-5, 5] so that the multiplier multiples divisors of value [0, 5] barring the sign. This itself reduces time requirement drastically. The proposed division architecture is void of any subtraction process and completely depends on addition and multiplication, thus eliminating the time consuming recursive subtraction process. Peer comparison also reflects our design to fare better than most of the literary counterparts.

Comparison of time consumption of *Radix-10 Division Architecture* with Table 7 reflects that Normalization accounts for better performance of *Radix-10 Division Architecture*, Due to unavailability of any mathematical explanation for accurate pre – estimation of the number of Normalizations required for a particular set of operands, time consumption using the proposed *Radix-10 Division Architecture* does not exhibit a noted pattern.

Table VII. FPGA Synthesis Analysis for Decimal Division Architectures

Device	Algorithm	Precision	# LUT	# Slices	Period	Latency
Virtex4	NR [32]	8	2008	2042	20.5	205
Virtex4	SRT [32]	8	2612	2196	16.4	164
Virtex6	[33]	8	1479		14.539	117
Virtex4	A8Single, Table 4 of [42]	8	2016		3.4	173
Virtex4	A8Single, Table 4 of [42]	8	1605		3.4	173
Virtex4	A8Double, Table 4 of [42]	8	2224		3.4	160
Virtex4	A8Double, Table 4 of [42]	8	1783		3.4	160
Virtex6	A8Single, Table 6 of [42]	8	1549		2.6	135
Virtex6	A8Single, Table 6 of [42]	8	987		2.6	135
Virtex6	A8Double, Table 6 of [42]	8	1737		2.6	122
Virtex6	A8Double, Table 6 of [42]	8	1166		2.6	122
Virtex4	NR [32]	16	2974	2859	21.4	386
Virtex4	SRT [32]	16	3799	2287	16.6	300
Virtex6	[33]	16	2392		15.293	245
Virtex4	A16Single, Table 6 of [42]	16	2756		3.4	401
Virtex4	A16Single, Table 6 of [42]	16	2091		3.4	401
Virtex4	A16Double, Table 6 of [42]	16	3768		3.4	326
Virtex4	A16Double, Table 6 of [42]	16	2718		3.4	326
Virtex4	NR [32]	32	4894	4503	23.9	813
Virtex4	SRT [32]	32	6533	4385	17.5	595
Virtex6	[33]	32	4066		15.139	485

Table VIII. CMOS Cell Implementation Results for Decimal Division (precision: 16 Digits)

Design	Area	Ratio	Latency	No. of Cycles	FO4 Cycle Time	Source
[19]	22600	1.74	680.2	19	35.8	[29]
[27]	13500	1.69	662	20	33.1	[29]
[50]		1.59	624	48	13	[29]
[29]	11100	0.95	371.45	19	19.55	[29]
[29]	11130	1	391	20	19.55	[29]

[51]	10500	1.21	472.5	21	22.5	[29]
[27]	59700	1.57	20	20	1	[27]
[30]	56468	1	-----	-----	0.62	[30]

REFERENCES

1. Diganta Sengupta and Atal Chaudhuri, "Vedic Sutras – A New Paradigm for Optimizing Arithmetic Operations," in Handbook of Research on Natural Computing for Optimizing Problems, J K Mandal and Somnath Mukhopadhyay, Eds.: IGI Global, 2016, ch. 36, pp. 890-915.
2. Diganta Sengupta, Mahamuda Sultana, and Atal Chaudhuri, "A New paradigm in Fast BCD Division Using Ancient Indian Vedic Mathematics Sutras," in Third International Conference on Computer Science, Engineering & Applications (ICCSEA), New Delhi, India, May 2013, pp. 11-19.
3. Diganta Sengupta, Mahamuda Sultana, and Atal Chaudhuri, "Vedivision – A Fast BCD Division Algorithm Facilitated by Vedic Mathematics," International Journal of Computer Science & Information Technology (IJCSIT), vol. 5, no. 4, pp. 67 - 80, August 2013.
4. Diganta Sengupta, Mahamuda Sultana, and Atal Chaudhuri, "An algorithm facilitating Fast BCD Division on Low End Processors using Ancient Indian Vedic Mathematics Sutras," in 2012 International Conference on Communication, Devices and Intelligent Systems (CODIS), Kolkata, India, December, 2012, pp. 373-376.
5. F Y Busaba, C A Krygowski, W H Li, E M Schwarz, and S R Carlough, "The IBM z900 Decimal Arithmetic Unit," in Conference Record of the Thirty-Fifth Asilomar Conference on Signals, Systems and Computers, 2001, Pacific Grove, CA, USA , 2001, pp. 1335-1339, vol 2.
6. Charles Franklin Webb and Wen He Li, "Specialized Millicode Instructions for Packed Decimal Division," Patent 6067617, May 2000.
7. D Bolt and J Reitsma, "Coded decimal non-restoring divider," Patent US 3735108 A, May 22, 1973.
8. Akira Yamaoka, Kenichi Wada, and Kazunori Kuriyama, "Coded decimal non-restoring divider," Patent US 4692891 A, September 8, 1987.
9. Toru Ohtsuki, Yoshio Oshima, Sako Ishikawa, and Masaharu Fukuta, "Binary coded decimal number division apparatus," Patent US 4603397 A, July 29, 1986.
10. Davis Claud M and John A Veer De, "Divider utilizing multiples of a divisor," Patent US 3234366 A, November 15, 1961.
11. David E Ferguson, "Non-heuristic decimal divide method and apparatus," Patent US 5587940 A, December 24, 1996.
12. Steven R Carlough, Paulomi Kadakia, Wen H Li, and Eric M Schwarz, "Method for performing decimal division," Patent US 8229993 B2, July 24, 2012.
13. Steven R Carlough, Paulomi Kadakia, Wen H Li, and Eric M Schwarz, "System and method for performing decimal division ," Patent US 7519649 B2, April 14, 2009.
14. Fadi Y Busaba, Steven R Carlough, Christopher A Krygowski, and John G Rell Jr., "Method and system for determining quotient digits for decimal division in a superscaler processor," Patent US 7149767 B2, December 12, 2006.
15. Freiman V Charles and Wang Chung Chian, "Division system and method," Patent US 3591787 A, July 6, 1971.
16. Wang Liang-Kai and Schulte J Michael, "Processing unit having decimal floating-point divider using Newton-Raphson iteration," Patent US 7467174 B2, December 16, 2008.
17. Miu Ming-Tzer, "Preconditioned divisor for expedite division by successive subtraction," Patent US 3578961 A, May 18, 1971.
18. J M Tendler, J S Dodson, J S Fields Jr., H Le, and B Sinharoy, "POWER4 System Microarchitecture," IBM Journal of Research and Development, vol. 46, no. 1, pp. 5-26, January 2002.
19. Hooman Nikmehr, Braden Phillips, and Cheng-Chew Lim, "Fast Decimal Floating-Point Division," IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, vol. 14, no. 9, pp. 951-961, September 2006.
20. John Cocke and D W Sweeney, High speed arithmetic in a parallel device. San Jose: IBM Corporation, 1957.
21. J E Robertson, "A New Class of Digital Division Methods," IRE Transactions on Electronic Computers, vol. EC-7, no. 3, pp. 218 - 222, September 1958.
22. K D Tocher, "Techniques of multiplication and division for automatic binary computers," The quarterly journal of Mechanics and Applied Mathematics, vol. 11, no. 3, pp. 364-384, 1958.
23. P Soderquist and M Leeser, "Area and Performance Tradeoffs in Floating-point Divide and Square-root Implementations," ACM Comput. Surv., vol. 28, no. 3, pp. 518-564, 1996.
24. Stuart Oberman, Nhon Quach, and Michael Flynn, "The Design and Implementation of A High-Performance Floating-Point Divider," Stanford University, Stanford, California, Technical CSL-TR-94-599, January, 1994.
25. S F Oberman and M J Flynn, "Division algorithms and implementations," IEEE Transactions on Computers, vol. 48, no. 8, pp. 833-854, August 1997.
26. Hooman Nikmehr, Braden Phillips, and Cheng-Chew Lim, "A decimal carry-free adder," in SPIE 5649, Smart Structures, Devices and Systems II, Sydney, Australia, 2005, pp. 786-797.
27. Tomas Lang and Alberto Nannarelli, "A Radix-10 Digit-Recurrence Division Unit:Algorithm and Architecture," IEEE TRANSACTIONS ON COMPUTERS, vol. 56, no. 6, pp. 727-739, June 2007.
28. E Antelo, T Lang, P Montuschi, and A Nannarelli, "Digit-recurrence dividers with reduced logical depth," IEEE Transaction On Computers, vol. 54, no. 7, pp. 837 - 851, July 2005.
29. A Kaivani, A Hosseiny, and G Jaberipur, "Improving the speed of decimal division," IET Computers & Digital Techniques, vol. 5, no. 5, pp. 393-404, September 2011.
30. Amir Kaivani and Seok-Bum Ko, "Decimal Division Algorithms: The Issue of Partial Remainders," Journal of Signal Processing Systems, vol. 73, no. 2, pp. 181-188, November 2013.
31. Tomas Lang and Alberto Nannarelli, "Comments on 'Improving the speed of decimal division'," IET Computers & Digital Techniques, vol. 6, no. 6, pp. 370-371, November 2012.
32. J Deschamps and G Sutter, "Decimal division: Algorithms and FPGA implementations," in VI Southern Programmable Logic Conference (SPL), 2010, Ipojuca, 2010, pp. 67-72.
33. M D Ercegovac and R Mcllhenny, "Design and FPGA implementation of radix-10 combined division/square root algorithm with limited precision primitives," in 2010 Conference Record of the Forty Fourth Asilomar Conference on Signals, Systems and Computers (ASILOMAR), Pacific Grove, CA, 2010, pp. 87-91.
34. Ivan D Castellanos and J E Stine, "Experiments for Decimal Floating-Point Division by Recurrence," in Fortieth Asilomar Conference on Signals, Systems and Computers, 2006. ACSSC '06, Pacific Grove, CA, 2006, pp. 1716-1720.
35. T Lang and A Nannarelli, "Division Unit for Binary Integer Decimals," in 20th IEEE International Conference on Application-specific Systems, Architectures and Processors, 2009. ASAP 2009., Boston, MA, 2009, pp. 1-7.
36. Saeid Gorgin and Ghassem Jaberipur, "Fully Redundant Decimal Arithmetic," in 2009 19th IEEE International Symposium on Computer Arithmetic, 2009, pp. 145-152.
37. Seiji Isotani, Bruce M McLaren, and Max Altman, "Towards Intelligent Tutoring with Erroneous Examples: A Taxonomy of Decimal Misconceptions," in Intelligent Tutoring Systems, Vincent Aleven, Judy Kay, and Jack Mostow, Eds. Pittsburgh, US: Springer Berlin Heidelberg, 2010, vol. LNCS 6095, pp. 346-348.
38. Cyril Bouvier and Paul Zimmermann, "Division-Free Binary-to-Decimal Conversion," IEEE Transactions on Computers, vol. 63, no. 8, pp. 1895-1901, August 2014.
39. S Gonzalez-Navarro, A Nannarelli, Michael J Schulte, and S Tsen, "A combined decimal and binary floating-point divider," in 2009 Conference Record of the Forty-Third Asilomar Conference on Signals, Systems and Computers, Pacific Grove, CA, 2009, pp. 930-934.
40. Liang-Kai Wang and Michael J Schulte, "Decimal floating-point division using Newton-Raphson iteration," in 15th IEEE International Conference on Application-Specific Systems, Architectures and Processors, 2004. Proceedings., 2004, pp. 84-95.
41. Liang-Kai Wang and Michael J Schulte, "A Decimal Floating-Point Divider Using Newton–Raphson Iteration," The Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology, vol. 49, no. 1, pp. 3-18, October 2007.
42. Mario P Vestias and Horacio C Neto, "Decimal Division Using the Newton–Raphson Method and Radix-1000 Arithmetic," in Embedded Systems Design with FPGAs, Peter Athanas, Dionisios Pnevmatikatos, and Nicolas Sklavos, Eds. New York, US: Springer New York, 2013, pp. 31-54.
43. M P Vestias and H C Neto, "Revisiting the Newton-Raphson Iterative Method for Decimal Division," in 2011 International Conference on Field Programmable Logic and Applications (FPL), Chania, 2011, pp. 138-143.

44. Jean Michel Muller, Elementary Functions : Algorithms and Implementation, 2nd ed.: Birkhäuser Basel, 2006.
45. D Sengupta, M Sultana, and A Chaudhuri, "Proposal for Fast BCD Addition," in Third IEEE International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN 2017), Kolkata, Communicated 2016, pp. 343-348.
46. D Sengupta, M Sultana, and A Chaudhuri, "Digit By Digit Fast Decimal Multiplication," in INDICON 2017 (14th IEEE India Council International Conference 2017), Roorkee, Communicated 2016.
47. Jagadguru Swami Sri Bharathi Krsna Tirathji Maharaj, Vedic Mathematics or Sixteen Simple Sutras From The Vedas. Varanasi: Motilal Banarsidas Publishers, 1986.
48. H Toyoshima, K Satoh, and K Ariyama, "High-speed hardware algorithms for Chinese remainder theorem," in IEEE International Symposium on Circuits and Systems, 1996. ISCAS '96., Connecting the World., 1996, Atlanta, GA, 1996, pp. 265 - 268 vol.2.
49. Shuangching Chen and Shugang Wei, "A High-Speed Realization of Chinese Remainder Theorem," in Proceedings of the 2007 WSEAS Int. Conference on Circuits, Systems, Signal and Telecommunications, Gold Coast, Australia, 2007.

AUTHORS PROFILE

Dr. Diganta Sengupta, B.Tech (2004) Electronics and Instrumentation Engineering from University of Kalyani, WB, IN. M.Tech (2010) CSE from Jadavpur University, WB, IN. Ph.D. (2016) from Jadavpur University, WB, IN. He is an IEEE member since 2016 and an ACM member in 2017. He is a life member of Computer Society of India (LM'16) and The Institution of Engineers (India) (M'16). He is also a member of IEEE Electron Device Society. Presently he is serving as the State Student Coordinator for West Bengal, India for Computer Society of India.
Dr. Diganta Sengupta is presently working in the capacity of Associate Professor in the Dept. of CSE, Techno International Batanagar, Kolkata, India. Formerly he was associated with the School of Computer Engineering (SCOPE), VIT University, Vellore, India.
His research interests include Decimal Numeric processors, Vedic Mathematics, Reversible Logic, Quantum Dot Cellular Automata, Taxonomy Generation Process and Hardware Accelerators for both classical as well as reversible computational engines. He has served as a reviewer for IEEE Access, JIKM (World Scientific), IJSAEM (Springer), IJBDCN (IGI-Global), ISSE (Springer) to name a few.

Mahamuda Sultana, B.Tech (2004) Computer Science and Engineering, University of Kalyani, WB, IN. M.Tech (2010), Computer Science and Engineering, Jadavpur University, WB, IN 700032. She is currently pursuing PhD in Computer Science and Engineering at Jadavpur University. Presently, she is associated in the capacity of Assistant Professor in the Dept. of Computer Science and Engineering, Techno International New Town, Kolkata, WB, IN 700156. She has a total of 12 years of teaching experience. Her research interests include Reversible Logic, Computer Architecture, Quantum Dot Cellular Automata and Vedic Mathematics.

Dr. Atal Chaudhuri received his Master of Electronics and Telecommunication Degree with Computer Science specialization in the year 1982 and Doctorate of Philosophy in Engineering from Jadavpur University, IN in 1989. Presently he is the Vice-Chancellor of Veer Surendra Si University of Technology (Burla University), Odisha, India. Previously he served as a Professor in the Dept. of Computer Science and Engineering at Jadavpur University. He has worked in the capacity of R&D Engineer and Project Engineer in various research projects in India and abroad. Formerly Dr. Atal Chaudhuri was a Senior Professor in the Department of Computer Science & Engineering of Jadavpur University, Kolkata, India. He is a Fellow at The Institution of Engineers (India) and a Life Member of Computer Society of India.