IJCSE International Journal of Computer Sciences and Engineering Open Access
Review Paper Volume-4, Issue-9 E-ISSN: 2347-2693

Comparative Study of Different Division Algorithms for Fixed and Floating Point Arithmetic Unit for Embedded Applications

Rohini.S.Hongal[1*] and Anita DJ[2]

[1*]Department of Electronic and Communication, BVBCET, Hubli, India
[2]Department of Electronic and Communication, BVBCET, Hubli, India

Available online at: www.ijcseonline.org

Received: 22/Aug/2016 Revised: 02/Sept/2016 Accepted: 20/Sept/2016 Published: 30/Sep/2016

Abstract— **ALU is very essential unit of any embedded processors. Very basic operations like addition, subtraction, multiplication and division are part of ALU unit. In literature, we have many algorithms to perform addition, subtraction and multiplication but less on division algorithm. The division algorithm performs, either by addition or subtraction, based on the signs of the divisor and partial remainder. Floating point division is considered as a high latency operation. Division algorithms have been developed to reduce latency and to improve the computational efficiency, hardware cost, area and power of processors. This paper presents different division algorithms such as Digit Recurrence Algorithm restoring, non-restoring and SRT Division (Sweeney, Robertson, and Tocher), Multiplicative Algorithm, Approximation Algorithms, CORDIC Algorithm and Continued Product Algorithm. This paper intended to compare various techniques used and their features relevant for various applications.**

Keywords- Division; SRT; Non Restoring; Restoring; FPGA; CORDIC; DSP

I. INTRODUCTION

Computers have evolved rapidly since their creation. However, there is one thing that has not changed: The main purpose of computers is to do the arithmetic to run programs and applications. Basically, computers handle lots of numbers based on the three basic arithmetic operations of addition, multiplication and division. Compared to addition and multiplication, division is the least used operation. However, computers will experience performance degradation if division is ignored [1], [2], [3]. A survey by Oberman and Flynn [9] presents the main algorithms used for implementing division in hardware. There are three main classes for hardware-oriented division algorithms: digit recurrence, functional iteration and table based methods. Each method has its own advantages [1], however digit recurrence division is most common algorithm for division and square root in many floating point units, since it is simple and lower in complexity than division by convergence [2], [4], [5]. Restoring, non-restoring and SRT dividers are representative algorithms for digit recurrence division. Division is equivalent to repeat subtraction of the divisor from the dividend until the quantity left is smaller in magnitude than the divisor. The number of subtractions is the quotient, and the quantity left is the remainder. This process, if done straight forwardly, is very time consuming. It is substantially speeded if the most significant digits of the divisor and dividend are aligned before the first subtraction, and the divisor then shifted to the right one position whenever the partial remainder become smaller than the divisor before shifting. One shift may be necessary before any subtraction, if the initial alignment makes the divisor larger than the dividend. In binary, at most one subtraction can be made between shifts except as noted below. Two conventional techniques avoid the need to compare the remainder with the divisor after every subtraction. In restoring division, subtraction continues until the sign of the partial remainder changes; the change causes an immediate addition of the divisor and a corresponding decrement of the accumulating quotient, before the right shift. In non-restoring division, the sign change causes a shift followed by one or more additions until the sign changes back.

Of the four basic arithmetic operations: addition, subtraction, multiplication, and division, division is the hardest to implement in hardware. One of the main reasons for this is that while addition, subtraction and multiplication are well defined and give exact answers, division is less so. The result of a division between integers (or even between floating point numbers with finite precision) will in general be a rational number, which in many cases cannot be represented exactly in binary with a fixed number of bits. This leads either to an approximate answer, or to a second definition of division: integer division. Integer division considers both the dividend and divisor to be integers, and expresses the result uniquely as a quotient and remainder:

$$Dividend = Quotient \times Divisor + Remainder \qquad (1)$$

where the quotient is an integer, and the remainder satisfies

*Corresponding Author:
Rohini. S. Hongal
e-mail: rohini_sh@bvb.edu , Tel.: +91-99803-03578

$$0 \leq Remainder < Divisor \qquad (2)$$

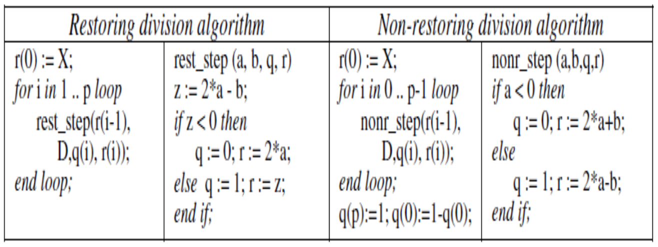
Within an image processing context, division occurs within several common image processing operations: contrast expansion, intensity normalisation, contrast ratio calculation, colour conversion (calculating the hue and saturation) are a few.

1. ALGORITHMS

Given two non-negative real numbers X (the dividend) and D (the divisor), the quotient q and the remainder r are non-negative real numbers defined by the following expression: $X = q.D + r$ with $r < D.ulp$, where ulp is the unit in the least significant position. If X<D and D are the (unsigned) significant of two IEEE-754 floating-point numbers, then they belong to the range [1,2), and q lies in the range [0.5, 1). This result can be normalized by shifting the quotient by one bit to the left, and adjusting the exponent accordingly. Division generally does not provide finite length result. The accuracy must be defined beforehand by setting the allowed maximum length of the result (p). The number of algorithmic cycles will therefore depend upon the aimed accuracy, not upon the operand length (n).

A. Restoring and Non-Restoring Algorithm

To divide two integers, the most well-known procedures are restoring and non-restoring digit-recurrence algorithms [3], [4]. The corresponding FPGA implementations are straightforward, and the area/time figure is always better for non-restoring. Figure1 depicts restoring and non-restoring division algorithm. In the latter one, a correction step must be added in order to modify the last remainder whenever negative.



Restoring division algorithm		*Non-restoring division algorithm*	
r(0) := X;	rest_step (a, b, q, r)	r(0) := X;	nonr_step (a,b,q,r)
for i *in* 1 .. p *loop*	z := 2*a - b;	*for* i *in* 0 .. p-1 *loop*	*if* a < 0 *then*
rest_step(r(i-1),	*if* z < 0 *then*	nonr_step(r(i-1),	q := 0; r := 2*a+b;
D,q(i), r(i));	q := 0; r := 2*a;	D,q(i), r(i));	*else*
end loop;	*else* q := 1; r := z;	*end loop;*	q := 1; r := 2*a-b;
	end if;	q(p):=1; q(0):=1-q(0);	*end if;*

Figure 1. Comparison of Restoring and Nonrestoring division algorithms

B. SRT Division

As others digit-recurrence algorithms, SRT generates a fixed number of quotient bits at every iteration as shown in figure 2. The algorithm can be implemented with the standard radix-r (r = 2k) SRT iteration architecture. The n-bit integer division requires $t = n/k$ iterations. An additional step is required in order to convert the signed-digit quotient representation into a standard radix-2 notation. The division x/d produces k bits of the quotient q per iteration. The quotient digit qj is represented using a radix-r notation (radix complement or sign-magnitude). The first remainder $w0$ is initialized to X. At iteration j, the residual wj is multiplied by the radix r (shifted by k bits on the left, producing $r.wj$). Based on a few most significant bits of $r.wj$ and d (nr and nd bits respectively), the next quotient digit $qj+1$ can be inferred using a quotient digit selection table ($Qsel$). Finally, the product $qj+1 \times d$ is subtracted to $r.wj$ to form the next residual $wj+1$. At the last bit position we get incorrect results. This is the main drawback of SRT division algorithm.

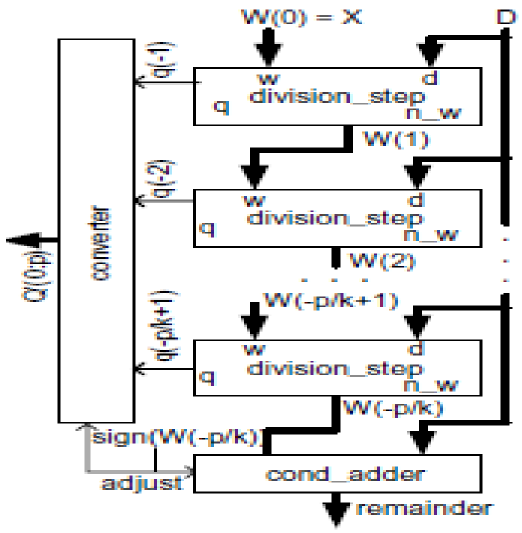


Figure 2. SRT standard array diagram

C. Adder-Cell-based Method

The design of division operator using the adder-cell-based method will always result in a very compact divider architecture. This method is classified as non-iterative technique, where the divider unit consists of half-adder and full-adder cells as well as other logic gate units and supporting modules [11]. A binary divider that uses carry save adder units is presented for example in [12].

D. Digit Recurrence Algorithm

In modern floating point arithmetic units the most common algorithm employed to division function is a digit recurrence algorithm [13] [14] [15]. The algorithm performs both operations based on shifting and subtraction as the fundamental operators. A combined floatingpoint square-root and division operation can also be implemented by using a subtractive SRT (Sweeney, Robertson and Tocher) algorithm [16], which can be classified as a digit recurrence algorithm. The subtractive SRT algorithm can be extended by using Radix-8 IDS (Interleaved Digit Set) algorithm to improve the performance of the traditional digit recurrence algorithm. Another variant of the digit recurrence method is Svoboda algorithm. A new Svoboda-Tung Division algorithm is for instance proposed in [17].

E. *Taylor's Series Expansion Algorithm*

A Taylor's Series Expansion Algorithm [18] for example can be used to calculate division operation using a sequential series of a harmonic equation. However, the Taylor's Series Expansion algorithm is rarely used and perform slow computation to calculate the division operations.

F. *Goldschmidt's Algorithm*

The basic idea behind the Goldschmidt's Algorithm is the iterative parallel multiplication of the dividend and divisor by updated factors in such as a way that the final divisor will be driven to one. Thus, the final dividend gives the quotient (the division result). Oberman et al. for example [19] proposes a floating-point divider and square root for AMD-K7 by using Goldschmidt's algorithm. The Goldschmidt's algorithm has been broadly used on many commercial microprocessors and is also known as division by multiplicative normalization or division by convergence [20]. Figure 3 shows detailed step by step procedure of Goldschmidt's algorithm. The disadvantage of the Goldschmidt's algorithm in term of the area overhead is the need for two independent parallel multiplication. As we know, a multiplier requires large number of logic area, especially when it is implemented in floating-point arithmetic. Goldschmidt Algorithm used for finding division and square root. Hardware and software implementation both are possible in this algorithm. Goldschmidt and Newton Raphson both are similar algorithms but slight changes in implementation part. Newton Raphson uses only software implementation.

G. *Newton-Raphson Algorithm*

The Newton-Raphson division algorithm is almost similar with the Goldschmidt's algorithm. In the Newton-Raphson method however, the iterative refinement is applied only to the reciprocal value of the divisor, which will be convergent after several iterations [21]. The division operation of the Newton-Raphson method can be divided into three steps, i.e. the initial estimation of the divisor's reciprocal, the iterative refinement of the divisor's reciprocal and the multiplication step between the divided and the final convergent divisor's reciprocal. The work in [22] has presented for example a decimal floating-point divider using Newton-Raphson iteration, where an accurate piece-wise linear approximation is used to obtain an initial estimate of a divisor's reciprocal. The main disadvantage of Newton-Raphson is requires large number of gate counts which is not feasible to implement on FPGA. It needs multiplication and addition/subtraction at each iteration.

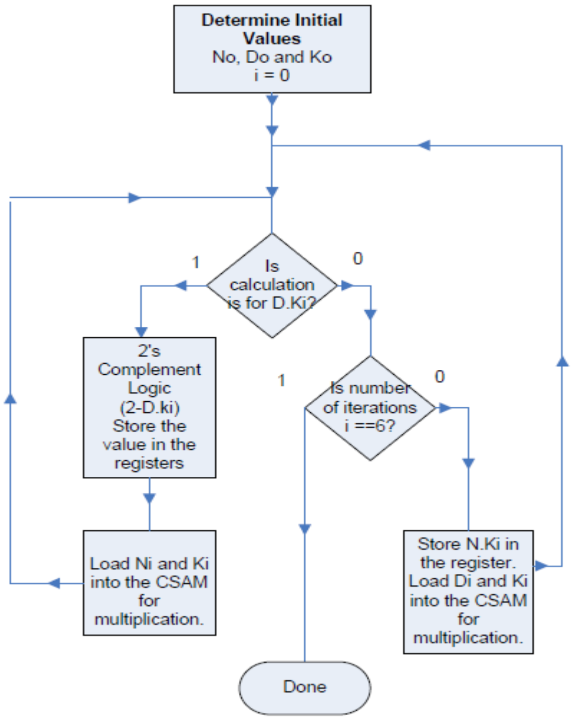

Determine Initial Values
No, Do and Ko
i = 0
Is calculation is for D.Ki?
1
0
2's Complement Logic (2-D.ki) Store the value in the registers
Is number of iterations i ==6?
1
0
Load Ni and Ki into the CSAM for multiplication.
Store N.Ki in the register. Load Di and Ki into the CSAM for multiplication.
Done

Figure 3. Flowchart of Goldschmidt Algorithm

H. *CORDIC Algorithm*

Beside the aforementioned method to implement the division operation, there is also another powerful algorithm to implement the divider unit called CORDIC (COrdinate Rotation DIgital Computer) algorithm [23]. Like digit recurrence method, CORDIC is also classified into iterative method. The main powerful characteristic of the CORDIC algorithm is the capability to implement several trigonometric function [24], [23], phase and magnitude functions [25], and hyperbolic functions [26] as well as linear operational function such as multiplication and division functions.

II. DESIGN METHODOLOGY

A. *Restoring Division Algorithm*

In the restoring division method, the quotient is represented using a non-redundant number system. This is "paper and pencil" usual algorithm. Its main characteristic is the full width comparisons required to deduce the new quotient digit [6]. In restoring division, the divisor is shift-positioned and subtracted from the dividend. If subtraction of the divisor produces a negative result at any bit position relative to the dividend, the operation at that bit position is unsuccessful, and a 0 is placed in the corresponding location of the quotient. The divisor is added back (restored) to the result of the division operation, then the next highest bit of the dividend is shifted into the left bit position of the result. As each bit of the dividend is shifted from right to left, the quotient is built up from left to right. After n shifts, where n represents the number of bits in the dividend, the division operation is complete. Complete hardware for restoring division is shown in Fig.1.In this figure an n-bit positive divider is loaded into register M and n- bit dividend is loaded into register Q at the start of the operation. After the division is complete, the n-bit quotient is in register Q and the remainder is in register A. The result after the last restore operation is the remainder.

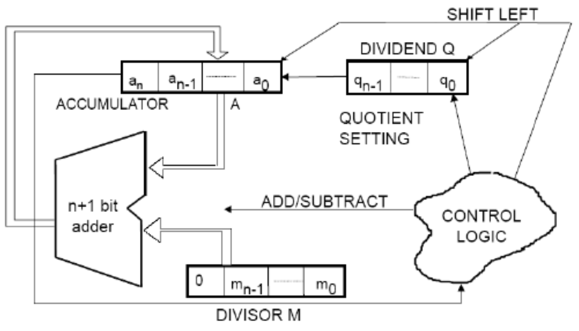


Figure 4. Hardware Design of Restoring Division Algorithm

Restoring division algorithm is very similar to manually performing long division. Algorithm for restoring division is mentioned below along with flowchart shown in Fig.5.

- Step 1: Set Count to 0 and put 0 in A register.
- Step 2: Start loop for n times.
- Step 3: Shift A & Q left one binary position.
- Step 4: Subtract M from a, placing the answer back in A.
- Step 5: if the sign of A < 0, set Q0 to 0 and add M back to A (restore A); otherwise, set q0 to 1.
- Step 6: Check for count, when count = n-1 then stop the loop.

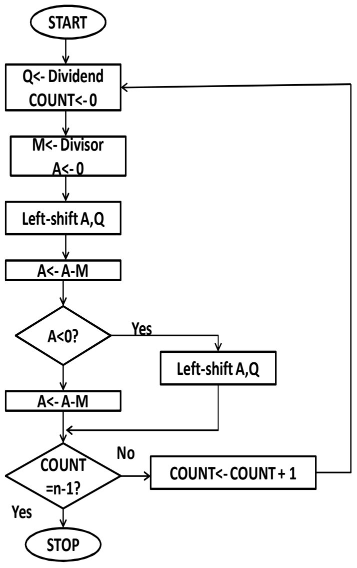


Figure 5. Flowchart of Restoring Division Algorithm

This template has been tailored for output on the US-letter paper size. If you are using A4-sized paper, please close this file and download the file for “MSW A4 format”.

B. *Non-Restoring Division Algorithm*

Non-restoring Division Algorithm (NrDA) comes from the restoring division. The restoring algorithm calculates the remainder by successively subtracting the shifted denominator from the numerator until the remainder is in the appropriate range. The operation in each step depends on the result of the previous step. Non-restoring division has a quotient digit set of {I, - I} instead of the conventional binary digit set [7], [9]. By the non-restoring division approach, we find the -1 of the quotient bit can be simply set to 0, and the quotient is the actual quotient that we want to find [8]. We dismantle Q into bits.

Algorithm for restoring division is mentioned below along with flowchart shown in Fig.6.

- Step 1: Subtract the divisor from the most significant bit (MSB) of the dividend.
- Step 2: “Bring down” the next MSB of the divisor and append it to the result of step 1.
- Step 3: Check the sign for the result of step 2. If the result of step 2 is positive:
    - o Set the next MSB of the quotient to 1.
    - o Subtract the divisor from the result to produce a new result.

  If the result from step 2 is negative:

- Set the next MSB of the quotient to 0.
- Add the divisor to the result to produce a new result.

- Step 4: Repeat steps 2 and 3 until all bits of the quotient are determined.

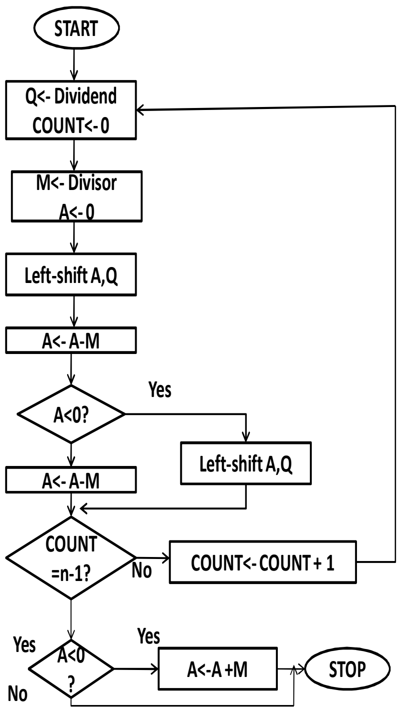


Figure 6. Flowchart of Non-Restoring Division Algorithm

Non-restoring arithmetic does not restore the result if the subtraction goes negative. Instead, it performs an addition in the next iteration. In this way the partial remainder will be kept between –D and D. The addition is equivalent to a bit weighting of q_i = -1 in the previous algorithm. Since q_i € { 1,1} , *qi* should be referred to as a quotient digit rather than a quotient bit. The test whether we add or subtract is also simpler, since we only need to test the sign of the partial remainder so far:

$$q_i = \begin{cases} -1 \text{ if } R_{i-1} < 0 \\ 1 \text{ if } R_{i-1} \geq 0 \end{cases} \quad (3)$$

$$R_i = \begin{cases} 2R_{i-1} + D \text{ if } q_i = -1 \\ 2R_{i-1} - D \text{ if } q_i = 1 \end{cases} \quad (4)$$

It is also necessary to convert the -1 and 1 weightings to conventional binary digits at the end. If a 0 bit is used to represent the -1 digit, then the obtained quotient is $Q_0 = q_1 q_2 q_3 q_4 q_5 q_6 q_7 q_8$ where the subtraction is replaced by the addition of a two's complement. Therefore a 1 is simply appended to the bits we already have. Also observe that for positive dividends the first iteration will always be a subtraction, therefore the quotient bit for this is not actually needed (unless the dividend is negative). Therefore, the iteration may be initialised by setting 0 $R = 2V\_D$. The next 8 iterations will give the 8 output bits, and again the addition or subtraction for the final iteration is not needed (since the remainder is discarded).

The advantage of using non-restoring arithmetic over the standard restoring division is that a test subtraction is not required; the sign bit determines whether an addition or subtraction is used. The disadvantage, though, is that an extra bit must be maintained in the partial remainder to keep track of the sign.

One limitation of the iteration expressed in equation is that separate hardware is used to perform the addition and subtraction, and the results are multiplexed to give the remainder. This may be simplified to a single addition, and multiplexing whether *D* or *–D* is added:

$$q_i = \begin{cases} -1 \text{ if } R_{i-1} < 0 \\ 1 \text{ if } R_{i-1} \geq 0 \end{cases} \quad (5)$$

$$R_i = 2R_{i-1} + \begin{cases} D \text{ if } q_i = -1 \\ -D \text{ if } q_i = 1 \end{cases} \quad (6)$$

A further optimisation is to replace the –D with the two's complement of D:

$$q_i = \begin{cases} -1 \text{ if } R_{i-1} < 0 \\ 1 \text{ if } R_{i-1} \geq 0 \end{cases} \quad (7)$$

$$R_i = 2R_{i-1} + \begin{cases} D \text{ if } q_i = -1 \\ \overline{D} - 1 \text{ if } q_i = 1 \end{cases} \quad (8)$$

The addition of the 1 as part of the 2's complement does not actually require additional logic because the 2*Ri*-1 will leave the least significant bit as 0. The 1 can be inserted instead if qi=1.

TABLE I. COMPARISON OF DIFFERENT DIVISION ALGORITHMS

Algorithms	Advantages	Limitations
Newton-Raphson	Similar to Goldschmidt algorithm. Software implementation is faster.	Requires large number of gate counts which is not feasible to implement on FPGA. It needs multiplication, addition/subtraction at each iteration.
Goldschmidt	Used for finding division and square root. Hardware-Software implementation is possible.	Area overhead is the need for two independent parallel multiplication. As multiplier requires large number of logic area, especially when it is implemented in floating-point arithmetic.
CORDIC	To implement trigonometric functions, and reduce area.	Cost effective because of ASIC design.
SRT	SRT generates a fixed number of quotient bits at every iteraton.	At the last bit position, we get incorrect results. Requires additional step of each iteration.
Restoring	Full width comparisons required to deduce the new quotient digit.	Slower; requires time because of restoration in each cycle.
Non-Restoring	Test subtraction is not required; the sign bit determines whether an addition or subtraction is used.	An extra bit must be maintained in the partial remainder to keep track of the sign.

III. CONCLUSION

Traditionally dividers have been avoided by DSP algorithm designers due to the complexity and cost of the hardware implementation. This paper compares different division algorithms which are used in Embedded applications. Restoring algorithms are slower than non-restoring algorithms, and a properly implemented non-restoring algorithm uses the least resources. A further advantage of the non-restoring algorithm is that it works without change with signed dividends, and only a relatively trivial change is required for it to work with a signed divisor.

REFERENCES

[1] S. F. Oberman and M. J. Flynn, "Design Issues in Division and other Floating-Point Operations," IEEE Transactions on Computers, Vol-46, Page No (154-161), 1997.

[2] Inwook Kong, "Modified Improved Algorithms and Hardware Designs for Division by Convergence," Doctoral dissertation. University of Texas at Austin, 2009.

[3] Peter Soderquist and Miriam Leeser, "Division and Square Root: Choosing the right Implementation", IEEE Micro, Vol-17, Issue-04, pp (56-66), 1997.

[4] Milos D. Ercegovac and Tomas Lang, "Division and Square Root: Digit Recurrence Algorithms and Implementations", Boston: KJuwer Academic Publishers, 1994.

[5] S. F. Oberman and M. J. Flynn, "Division Algorithms and Implementations", IEEE Transactions on Computers, Vol-46, no.-08, Page No (833-854), Aug 1997.

[6] Nicolas Boullis and Arnaud Tisserand, "On digit-recurrence division algorithms for self-timed circuits", in Research Report published at Institute National De Recherche En Informatique Et En Automatique, France, 2012.

[7] Kihwan Jun and Earl E. Swartzlander, "Modified Non-restoring Division Algorithm with Improved Delay Profile and Error Correction", Signals Systems and Computers (ASILOMAR), Page No (1460-1464), 2012.

[8] Jen-Shiun Chiang, Eugene Lai and Jun-Yao Liao,"A Radix-2 Non-Restoring 32-b/32-b Ring Divider with Asynchronous Control Scheme", Tamkang Journal of Science and Engineering, Vol-02, Issue-01 Page No (37-43), 1999.

[9] Jagannath Samanta, Mousam Halder, Bishnu Prasad De, "Performance Analysis of High Speed Low Power Carry Look-Ahead Adder Using Different Logic Styles", International Journal of Soft Computing and Engineering (IJSCE), ISBN:2231-2307, Volume-02, Issue-06, Page No (330–336), 2013.

[10] A. Beaumont Smith and S. Samudrala, "Method and System of a Microprocessor Subtraction- Division Floating-Point Divider", Patent US 7, 127, 483 B2, Oct. 24, 2006.

[11] D. J. Desmonds, "Binary Divider with Carry Save Adders", Patent US 4, 320, 464, March 16, 1982.

[12] J. Ebergen, I. Sutherland, and A. Chakraborty, "New Division Algorithms by Digit Recurrence," in Conference Record of the Thirty-Eighth Asilo- mar Conference on Signals, Systems and Computers, Vol-02, Page No (1849–1855), 2004.

[13] L. Montalvo, K. Parhi, and A. Guyot, "A Radix-10 Digit-Recurrence Division Unit: Algorithm and Architecture", IEEE Transaction on Computers, Vol-56, No.6, Page No (727–739), June 2007.

[14] I. Rust and T. Noll, "A digit-set-interleaved radix-8 division/square root kernel for double precision floating point", IEEE International Symposium on System on Chip (SoC), Page No (150–153), Nov 2010.

[15] L. Montalvo, K. Parhi, and A. Guyot, "New Svoboda-Tung Division", IEEE Transaction on Computers, Vol-47, No. 9, Page No (1014–1020), Sep 1997.

[16] T.-J. Kwon, J. Sondeen, and J. Draper, "Floating-point division and square root using a Taylor-series expansion algorithm", in the 50th Midwest Symposium on Circuits and Systems (MWSCAS 2007), Page No (305–308), 2007.

[17] S. F. Oberman, "Floating Point Division and Square Root Algorithms and Implementation in the AMD-K7 Microprocessor", in 14th IEEE Symposium on Computer Arithmetic, Page No (106–115), 1999.

[18] M. J. Schulte, D. Tan, and C. L. Lemonds, "Floating-Point Division Algorithms for an x86 Microprocessor with a

Rectangular Multiplier," in Proc. Int'l. Conf. on Computer Design (ICCD), Page No (304–310), 2007.

[19] W. Gallagher and E. Swartzlander, "Fault-Tolerance Newton-Raphson and Goldschmidt Dividers using Time Shared TMR", IEEE Transaction on Computers, Vol-49, No. 6, Page No (588–595), June 2000.

[20] L. K. Wang and M. Schulte, "Processing Unit Having Decimal Floating-Point Divider Using Newton-Raphson Iteration", Patent US 7,467,174 B2, Dec 16, 2008.

[21] P. Surapong, F. A. Samman, and M. Glesner, "Design and Analysis of Extension-Rotation CORDIC Algorithms based on Non-Redundant Method", International Journal of Signal Processing, Image Processing and Pattern Recognition, Vol-05, No. 1, Page No (65–84), March 2012.

[22] K. Maharatna, S. Banerjee, E. Grass, M. Krstic, and A. Troya, "Modified Virtually Scaling-Free Adaptive CORDIC Rotator Algorithm and Architecture", IEEE Trans. on Circuits and Systems Floating-Point Division Operator based on CORDIC Algorithm 87 tems for Video Technology, Vol-15, No. 11, Page No (1463–1474), Nov 2005.

[23] F. A. Sammany, P. Surapong, C. Spies, and M. Glesner, "Floating-point-based Hardware Accelerator of a Beam Phase-Magnitude Detector and Filter for a Beam Phase Control System in a Heavy-Ion Synchrotron Application", in Proc. Int'l Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS), 2011.

[24] H. Hahn, D. Timmermann, B. Hosticka, and B. Rix, "A Unified and Division-Free CORDIC Argument Reduction Method with Unlimited Convergence Domain Including Inverse Hyperbolic Functions", IEEE Transaction on Computers, Vol-43, No. 11, Page No (1339–1344), Nov 1994.

Authors Profile

Mrs Rohini S Hongal pursed Bachelor of Science from University Visvesvaraya, Belgaum in 2002 and Master of Technology in VLSI Design and Embedded Systems from same University in year 2009. She is currently pursuing Ph.D. and working as Assistant Professor in Department of Electronic and Communication, BVBCET, Hubli since 2003. She is a member of IEI. She has published more than 25 research papers in reputed international journals and conferences (includes IEEE,Springer) and it's also available online. Her main research work focuses on Analog and Digital VLSI, Quantum Computation, Embedded Design and IoT. She has 13 years of teaching experience and 3 years of Research Experience.

Ms Anita DJ pursed Bachelor of Engineering from Karnatak University, Dharwad India in year 2000 and Masters in VLSI Design and Embedded Systems Design from Visvesvaraya Technological University, Belgaum, India in year 2008. She served the Institutions as Assistant Professor in Department of ECE, from colleges SDMCET Dharwad, BVBCET Hubli and PESIT Bangalore, India . She is a life member of IEI and IETE. She has attended many FDP's to upgrade her, to latest trends in the field, both National & International Level including IEEE. Her main research area focuses on VLSI Design and Embedded Systems. She has 8 years of teaching experience and 1 year of Research Experience.