Computer Arithmetic Architectures with Redundant Number Systems *

Hosahalli R. Srinivas, and Keshab K. Parhi
Department of Electrical Engineering
University of Minnesota, Minneapolis, MN 55455

Abstract

Redundant arithmetic number systems are gaining popularity in computationally intensive environments particularly because of the carry-free addition/subtraction properties they possess. This property has enabled arithmetic operations such as addition, multiplication, division, square root, etc., to be performed much faster than with conventional binary number systems. In this paper, some of the recent contributions to the area of design of redundant arithmetic based addition, multiplication, division, and square root algorithms and architectures are briefly discussed. Also, only the use of bit/digit-parallel implementation for architectures is discussed so that the enhancement in speed through the use of redundant arithmetic becomes immediately apparent as opposed to the use of bit/digit-serial architectures, where the primary justification for their use is to conserve area. A new radix 2 division algorithm using over-redundant radix 2 quotient digits and requiring a 2 digit quotient selection function is also presented.

1 Introduction

Digital computer arithmetic operations play a crucial role in many applications, where speed is essential, e.g., in digital signal processing, communications, cryptography, etc. VLSI circuit implementation of sophisticated arithmetic operators, which were considered prohibitively complex to implement in the past, can be easily put on a tiny wafer of silicon. Current day VLSI technology boasts of over a million transistors on a single chip. Such capabilities have enabled microprocessor designers to provide on-chip floating point arithmetic units and avoid the pin input/output bandwidth limitations that would otherwise arise with off-chip floating point units provided as co-processors.

Redundant number systems [1] [2] are positional number systems similar to the conventional decimal system, but allow more values for a digit than the base. This results in more than one representation for a number and hence its name. Arithmetic operators designed using this number system achieve considerable speed improvement compared with operators designed using conventional number system.

Speed of an arithmetic operator is an issue that is directly related to the architecture chosen. Architectures and implementation styles are usually decided by the number system employed to represent numbers. Architectures are concerned with the direct hardware configuration of logic units (e.g., logic gates) for the physical realization of the arithmetic algorithm. Implementation style dictates the manner in which communication between different basic units of the architecture and the communication with the external world has to be achieved. For example, in conventional binary number systems, many implementation styles are possible such as bit-parallel (all bits at a time), bit-serial (one bit at a time), digit-serial (multiple bits at a time), word-parallel (multiple words at a time), etc. In redundant arithmetic architectures two implementation styles are prominent. They are digit-parallel and digit-serial. Digit-parallel is similar to bit-parallel (for binary number system) implementation. Digit-serial arithmetic implementation is also referred to as the *on-line* arithmetic implementation [3]. An interesting feature of this type of implementation is that both most-significant digit first and the least-significant digit first realizations can be easily achieved with low computational latencies.

A new radix 2 division algorithm that is faster than previous radix 2 division algorithms, which do not use input prescaling, and area efficient when compared with radix 2 division algorithms which require input prescaling is also presented in this paper.

This paper is divided into the following sections. Section 2 gives a brief overview of redundant arithmetic number system. Section 3 discusses some of the recently proposed algorithms and architectures for different arithmetic operations, which employ redundant number systems.

*This research was supported by the Office of Naval Research under contract number N00014-91-J-1008.

2 Redundant Number Systems

The base of redundant number systems is referred to as its radix. Redundant number systems belong to a superset of number systems referred to as Generalized Signed Digit Number Systems [4]. A radix β redundant number system is allowed to possess digits from the set $\{-(\beta-1),\ldots,-1,0,+1,\ldots,(\beta-1)\}$. The value of a n-digit radix β number, $X = x_{n-1}x_{n-2}\ldots x_0$, is given by $X = x_{n-1}\beta^{n-1}+x_{n-2}\beta^{n-2}+\cdots+x_0\beta^0 = \sum_{i=0}^{n-1} x_i\beta^i$, where $x_i \in \{-(\beta-1),\ldots,(\beta-1)\}$. So, the number of values a digit is allowed to possess by this number system is $(2\beta-1)$, which is larger than the radix. This, therefore, results in more than one representation for a given value of a number. However, there are certain restrictions on the choice of the absolute maximum values for the digits used in the system. Note that the popular carry-save representation is actually a radix 2 redundant number representation (since the number of digits allowed, 3 ($\{0,1,2\}$), is more than the base).

Usually, the digit set for redundant arithmetic based architectures is restricted to $\{-a,-(a-1),\ldots,0,\ldots,(a-1),a\}$, where $\lceil\frac{\beta-1}{2}\rceil \leq a \leq (\beta-1)$ so that it can result in simpler designs, as each digit is represented in a digital system using bits, conforming to a particular encoding. Therefore, to represent a radix β digit we require at least $\lceil\log_2(2a+1)\rceil$ bits. The lower limit on the value of a is given by the necessity of at least β different values for a digit required to represent a number in radix β number system. The upper limit (or maximum absolute value) of a allowed is restricted to $(\beta-1)$ so as to have a unique representation for zero. The redundant number system with $a = \lceil\frac{\beta-1}{2}\rceil$ is referred to as *minimally* redundant and with $a = (\beta-1)$ is referred to as *maximally* redundant.

There are certain features of this number system which are in a way disadvantageous. Comparison of two redundant numbers is not easy since multiple valid representations exist for the same value of a number. Also, the sign of a redundant number is the sign of the most-significant non-zero digit in it and this is not easy to detect as in one's-complement and two's-complement numbers.

Binary arithmetic being an accepted standard in current day digital systems, any use of redundant arithmetic is done internal to the systems and communication with the external world is done only after converting the results (outputs) of computation (which are in redundant form) to binary form. Input conversion is not an issue as the bits 0 and 1 are subsets of redundant digit set. Output conversion requires extra hardware and time. Conversion of digits to bits can be performed starting from the lsd (least-significant digit) and progressing towards the msd (most-significant digit), also called the lsd-first method [5], or in the opposite direction called the msd-first method [6].

3 Algorithms and Architectures

3.1 Addition/Subtraction

The most important feature of redundant arithmetic number systems is that they allow carry-free additions/subtractions. Carry propagation is the main cause of delay in adders. The sum of two numbers is not generated in an adder until carry has propagated from the least-significant bit position to the most-significant bit position. A typical example for this is a W-b ripple carry adder, which has a delay of $W.\delta_{fa}$ (δ_{fa} is the delay of 1-bit full adder, a 3-to-2 counter cell). Eliminating carry propagation can result in fast adder/subtractor designs. In [7] it is shown that to allow carry-free addition/subtraction, for a radix β number system with digits from set $\{-a,\ldots,a\}$, a should satisfy $\lceil\frac{\beta+1}{2}\rceil \leq a$. This is true for any radix greater than 2. But for radix 2, the carry and sum digits had to be selected in a particular way only [2].

Carry-save type of addition of conventional binary numbers also results in carry-free addition/subtraction [1][2]. A W-digit carry-save adder has a delay of a 1-bit full adder cell (also called 3-to-2 counter) and is independent of the word-length, W. An important feature of carry-save architectures is that they allow carry-free multi-operand addition through the use of *Wallace* trees [8]. The number of levels required for adding k operands is approximately $\log(k/2)/\log(3/2)$ [2]. A major disadvantage of *Wallace*-trees is that they require complex interconnection, when implemented in VLSI.

A variation of the carry-save type of addition that employs signed binary digits (belonging to the set $\{-1,0,+1\}$) and conventional binary bits ($\{0,+1\}$) has been used in [5]. The adders used here are similar to the 1-bit full adders both in area and time complexity. Hybrid number based architectures that employ a representation formed by mixing of redundant and conventional representations is possible [5] [9]. In [5], the architectures presented accept(generate) inputs(outputs) in binary form whereas internally radix 2 redundant arithmetic is used. In [9] an idea of allowing both positive and negative digits only at certain digit positions instead of all the digits has been presented. Such a technique limits the length of the carry propagation chain to the region between the signed digit positions.

Radix 2 redundant arithmetic has also been used to design a fast adder for addition of 2 two's-complement

binary numbers with carry-propagation [10].

3.2 Multiplication

Fast multiplication schemes use carry-free adders for addition of partial products to generate the product of two numbers. For example, the cellular array multiplier like the *Baugh-Wooley* [11] multiplier employs carry-free adders in each and every row of the array to add all the partial products. The final product is in redundant form which is converted to binary form using a fast adder or a converter. The latency of such multiplier arrays is of the order $O(W)$, where W is the word-length of the operands being multiplied.

The radix 4 modified *Booth* multiplier [12], which recodes the multiplier bits using minimally redundant radix 4 digits ($\{-2,-1,0,+1,+2\}$), also uses carry-save adders to add all the $W/2$ partial products. These array type multipliers actually employ redundant arithmetic (in the form of carry-save arithmetic) although this has not been indicated explicitly. The latency of the *Booth* multiplier arrays is of the order $O(W/2)$. The *Wallace*-tree method of adding the partial products employed with modified *Booth* encoding results in computational latencies of $O(\log W/2)$.

Multipliers that add the partial products in a binary-tree of carry-free redundant arithmetic adders have also been designed [13]. The binary-tree type adder uses 2-to-1 compressor cells instead of the 3-to-2 used by the *Wallace* tree.

Left-to-right multiplication scheme in which the most-significant half of the product is produced without any carry-propagation has been proposed in [14].

3.3 Division

Division operation can be broadly classified into the indirect and direct methods. The indirect methods are based on approximation techniques and rely on multiplication schemes whereas the direct methods are similar to the paper and pencil division algorithm (also referred to as digit-by-digit division since one quotient digit is produced in each step of the algorithm) and require a linear number of steps to be completed. In this section we review digit-by-digit based division algorithms only.

The digit-by-digit based division algorithm involves a division recursion in which the quotient is generated (after observing the partial remainder), one digit per recursion starting with the most-significant digit and ending with the least-significant digit [1] [2].

The *SRT* division algorithm [15] is the most prominent among the redundant number based division algorithms. Another type is the *Svoboda* type division algorithm [16]. These algorithms allow redundant representations for the partial remainders and quotient digits, and perform quotient digit selection by observing a constant number (independent of the word-length, W) of most-significant digits (msds) of the partial remainder. Since addition/subtraction involving redundant numbers is carry-free (hence fast), the use of redundant arithmetic for division recursion is a natural choice for a high-speed divider design. The quotient selection logic, that observes the partial remainders and the divisor, constitutes a major part in the critical path involved in the division recursion hardware. Much of the research efforts in design of redundant arithmetic based high-speed division algorithms and architectures have been focussed on reducing this hardware (hence time) required for quotient selection. These algorithms can be designed in any radix and the choice of a radix directly determines the number of recursions required to complete the division operation and also the design complexity. Higher the radix, lower the number of division recursions (the number of recursions is reduced to $W/\log_2(radix)$, where W is the word-length) required.

A number of radix 2 *SRT* division algorithms have been designed [6] [13]. Higher radix (with radix $\geq$ 4) division algorithms have been designed in [6] [17]. A radix 16 version of the *Svoboda* algorithm was developed in [18]. Three radix 4 division algorithms, which employ input prescaling have been presented in [19] [20] [21]. The divider architecture of [20] is very interesting since it produces over-redundant radix 2 quotient digits, i.e., digits from the set $\{-4,-3,-2,-1,0,1,+2,+3,+4\}$. Radix 2 versions of the division algorithm that employ prescaling have been presented in [22]. The prescaling algorithms incur hardware (also scheduling and time) penalty in the form of availability of a fast carry propagate adder to perform the input prescaling.

The most-significant digit first conversion scheme proposed in [6] is the ideal conversion scheme for converting quotient to binary form in these division algorithms.

New Radix 2 Division Algorithm

This division algorithm operates on the mantissas of the IEEE 754 1985 floating point standard. That is, both the dividend and the divisor belong to the floating point std. range. The recursive expression for the proposed algorithm can be written as $x_i = 2x_{i-1} - q_i 2^{-i} D$, and in the shifted form $x_i^* = 2x_{i-1}^* - q_i 2^{-i} D$, with $Q_0 = 0$, and $Q_i = Q_{i-1} + q_i 2^{-i}$ for $i \geq 1$, $X_i^* = X_i 2^i$, and the remaining terms defined as – i the index of recursion ($i = 1, 2, \ldots, W$), X_i the i-th partial remainder, X_i^* the i-th shifted partial remainder, X_0 (or X) the dividend (mantissa), D the divisor (mantissa), Q_i the i-th partially generated

quotient, with $Q_0 = 0$, Q_W the W-fractional digit quotient of X_0/D, and q_i the i-th quotient digit belonging to the over-redundant radix 2 digit set $\{-2, \ldots, +2\}$. In the above expression, substituting $i = 1$ to W, we get $X_W = X_0 - Q_W D$, where $Q_W < 1$ as X_0 is chosen to be less than D.

The partial remainder in the proposed algorithm is always in the region $|x^*_{j-1}| < D$. With this remainder range, the quotient digits can be so selected that whenever q_j is chosen as $+2$, the next non-zero quotient digit will be negative and whenever q_j is chosen as -2, the next non-zero quotient digit will be positive. Further, the adder subtractor used to perform recursion is similar to that explained in [5]. The use of this adder architecture and over-redundant radix 2 quotient digits ensure that the addition/subtraction operations during division recursion do not have representation overflow, i.e., the transfer digit from the adder is always zero. With this arrangement, only the two most-significant digits of the partial remainder are sufficient to perform the quotient selection.

The quotient selection function for the proposed algorithm is as follows: $q_j = -2$ if $X'_{j-1} = -3$, $q_j = -1$ if $X'_{j-1} = -2$ or -1, $q_j = 0$ if $X'_{j-1} = 0$, $q_j = +1$ if $X'_{j-1} = +2$ or $+1$,$q_j = +2$ if $X'_{j-1} = +3$, where $X'_{j-1} = x_{j-1,-1}2 + x_{j-1,0}$ is the integer part, composed of the two most-significant digits, of the shifted partial remainder $2X^*_{j-1} = X'_{j-1} + X''_{j-1} = x_{j-1,-1}2 + x_{j-1,0} + x_{j-1,1}2^{-1} + \cdots + x_{j-1,W}2^{-W}$.

The division algorithm presented in [20] cannot be extended to the radix 2 case directly because the over-redundant radix 2 quotient generated in such an algorithm cannot be reduced to the regular radix 2 digits for conversion to the binary form using the on-the-fly converter [6]. This is true since the conversion is not possible without carry-propagation. Extending the original on-the-fly converter strategy to accommodate the over-redundant radix 2 digits for conversion directly to binary form is possible as suggested in [20]. But this scheme will result in a 100% increase in the area of the converter. In the proposed division scheme, the quotient digits are produced in a manner that a $+2$ quotient digit is always followed by a subsequent non-zero negative quotient digit and a -2 quotient digit is always followed by a subsequent non-zero positive quotient digit. This allows a reduction of over-redundant radix 2 quotient digits produced to regular radix 2 digit form with minimal hardware overhead. The reduction algorithm has been described below.

Reduction Algorithm: Reduce(Q, Q')
INPUT: Vector Q of over-redundant radix 2 digits $(Q = 0.q_1q_2q_3 \cdots q_W)$.
OUTPUT: Vector Q' of conventional radix 2 digits $(Q' = 0.q'_1q'_2q'_3 \cdots q'_W)$.
{ $set = 0$; for $i = 1$ to W {
if $(set == 0)$ { if $(q_i \in \{-1, 0, +1\})$ $q'_i = q_i$;
if $(q_i \in \{-2, +2\})$ { $q'_i = \frac{q_i}{2}$; $prev_q = q'_i$; $set = 1$;} }
else { if $(q_i \in \{-1, +1\})$ { $q'_i = -q_i$; $set = 0$; }
if $(q_i \in \{-2, +2\})$ { $q'_i = 0$; $set = 0$; }
if $(q_i == 0)$ $q'_i = prev_q$; } } }

Note that in the above pseudo code provided, the *for* loop starts with index $i = 1$ (corresponding to the most-significant digit q_1 of quotient) and terminates with $i = W$ (corresponding to the least-significant digit q_W of the quotient). This shows that the reduction technique proceeds from most-significant digit to least-significant digit end and is ideal for digit-by-digit based algorithms such as division, etc., which are inherently most-significant digit first algorithms. The hardware requirement for this algorithm is three 2-to-1 multiplexers, 4 2-input gates and a 2-bit register, which is negligible and is independent of word-length.

This algorithm is faster than previously proposed radix 2 division algorithms [6] [13] that do not require prescaling. That is because these algorithms observe at least three most-significant radix 2 partial remainder digits for quotient selection as opposed to the two required by the proposed algorithm. When compared with radix 2 division algorithm [22] that requires input prescaling the proposed algorithm is slightly slower but has the advantage of not-requiring any input prescaling and thus saving hardware and prescaling time penalties.

3.4 Square Root

Square rooting techniques are of two types, namely direct techniques (digit-recurrence or digit-by-digit) and successive approximation. The direct techniques produce one digit of the root per iteration [1], [2]. The second technique uses an approximation method which converges to the result [1], [2]. A good survey of square rooting algorithms has been presented in [23].

Most of the high-speed digit recurrence methods for square root are extensions of the redundant arithmetic based non-restoring *SRT* division algorithm [15] [17]. Some of these algorithms include the designs presented in [24], [25]. Majerski's square root algorithm is the most prominent radix 2 based algorithm [25]. This algorithm operates on mantissa in the range [1/4,1). A good analysis of higher radix square rooting algorithms has been presented in [26]. Interesting radix 4 square root unit designs have been presented in [6].

4 Conclusion

This paper briefly discussed some of the recent work on redundant arithmetic based addition, multiplication, division, and square-root operations. It

did not discuss redundant arithmetic based algorithms and architectures for elementary function generation schemes such as CORDIC (CO-ordinate ROtation DIgital Computer). Furthermore, it focussed only on digit-parallel implementation style to bring forth immediately the speed enhancement possible for algorithms and architectures through use of redundant arithmetic. The interested reader is referred to reference [27] for further reading on redundant arithmetic.

References

[1] K. Hwang, *Computer Arithmetic: Principles, Architecture, and Design.* NY: Wiley, 1979.

[2] I. Koren, *Computer Arithmetic Algorithms.* NJ 07632: Prentice Hall, Englewood Cliffs, 1993.

[3] M. J. Irwin and R. M. Owens, "Digit-Pipelined Arithmetic as Illustrated By the Paste-UP System:A Tutorial," *IEEE Computer*, pp. 61–73, April 1987.

[4] B. Parhami, "Generalized Signed-Digit Number Systems: A Unifying Framework For Redundant Number Representations," *IEEE Transactions on Computers*, vol. C-39, pp. 89–98, January 1990.

[5] H. R. Srinivas and K. K. Parhi, "High-Speed VLSI Arithmetic Processors Using Hybrid Number Systems," *Journal of VLSI Signal Processing*, vol. 4, pp. 177–198, April 1992.

[6] M. D. Ercegovac and T. Lang, *Division and Square Root.* Norwell, Massachusetts 02061: Kluwer Academic Publishers, 1994.

[7] A. Avizienis, "Signed Digit Number Representation for Fast Parallel Arithmetic," *IRE Transactions on Electronic Computers*, vol. EC-10, pp. 389–400, September 1961.

[8] C. S. Wallace, "A Suggestion for a Fast Multiplier," *IEEE Transactions on Computers*, vol. EC-13, pp. 14–17, February 1964.

[9] D. S. Phatak, I. Koren, and H. Choi, "Hybrid Number Representations with Bounded Carry Propagation Chains," in *Proc. of IEEE International Conference on Computer Design: VLSI in Computers and Processors*, (Cambridge, MA), pp. 272–275, October 3 - 6 1993.

[10] H. R. Srinivas and K. K. Parhi, "A Fast VLSI Adder Architecture," *IEEE Journal of Solid State Circuits*, vol. 27, pp. 761–767, May 1992.

[11] C. R. Baugh and B. A. Wooley, "A Two's-Complement Parallel Array Multiplication Algorithm," *IEEE Transactions on Computers*, vol. C-22, pp. 1045–1047, December 1973.

[12] A. D. Booth, "A Signed Binary Multiplication Technique," *Q. J. mech. appl. math:4*, pp. 236–240, Oxford University Press, 1951.

[13] S. Kuninobu, T. Nishiyama, H. Edamatsu, T. Tanaguchi, and N. Takagi, "Design of High Speed MOS Multiplier and Divider Using Redundant Binary Representation," in *Proc. of 8th Symposium on Computer Arithmetic*, (Como, Italy), pp. 80–86, 1987.

[14] M. D. Ercegovac and T. lang, "Fast Multiplication Without Carry-Propagate Addition," *IEEE Transactions on Computers*, vol. 39, no. 11, pp. 1385–1390, November 1990.

[15] J. E. Robertson, "A New Class of Digital Division Methods," *IRE Transactions on Electronic Computers*, vol. EC-7, pp. 218–222, September 1958.

[16] A. Svoboda, "An Algorithm for Division," *Information Processing Machines*, vol. 9, pp. 183–190, March 1963.

[17] D. E. Atkins, "Higher-Radix Division Using Estimates of the Divisor and Partial Remainders," *IEEE Transactions on Computers*, vol. C-17, pp. 925–934, October 1968.

[18] C. Tung, "A Division Algorithm for Signed-Digit Arithmetic," *IEEE Transactions on Computers*, vol. 17, pp. 887–889, September 1968.

[19] M. D. Ercegovac and T. Lang, "Simple Radix-4 Division with Operands Scaling," *IEEE Transactions on Computers*, vol. C-39, pp. 1204–1208, September 1990.

[20] P. Montuschi and L. Ciminiera, "Over-Redundant Digit Sets and the Design of Digit-by-Digit Division Units," *IEEE Transactions on Computers*, vol. 43, pp. 269–277, March 1994.

[21] H. R. Srinivas and K. K. Parhi, "A Fast Radix 4 Division Algorithm," in *Proc. of IEEE International Symposium on Circuits and Systems*, (London), pp. 311 – 314, May 30 - June 2 1994.

[22] S. E. McQuillan, J. V. McCanny, and R. Hamill, "New Algorithms and VLSI Architectures for SRT Division and Square Root," in *Proc. of 11th Symposium on Computer Arithmetic*, (Windsor, Ontario), pp. 80–86, June 29-July 2 1993.

[23] P. Montuschi and M. Mezzalama, "Survey of Square Rooting Algorithms," *IEE Proceedings*, vol. 137 Pt. E, pp. 31–40, January 1990.

[24] G. Metze, "Minimal Square Rooting," *IEEE Transactions on Electronic Computers*, vol. EC-14, pp. 181–185, April 1965.

[25] S. Majerski, "Square-Rooting Algorithms for high-speed digital circuits," *IEEE Transactions on Computers*, vol. 34, pp. 724–733, 1985.

[26] L. Ciminiera and P. Montuschi, "Higher Radix Square Rooting," *IEEE Transactions on Computers*, vol. 39, pp. 1220–1231, October 1990.

[27] E. E. Swartzlander JR., ed., *Computer Arithmetic*, vol. 1 and 2. Los Alamitos, California: IEEE Computer Society Press, 1990.