International Journal of Research in Engineering, Science and Management
Volume-2, Issue-2, February-2019
www.ijresm.com | ISSN (Online): 2581-5792

Implementation of VLSI Binary 64 Division with Redundant Number Systems

M. Lakshmi Priya[1], M. Janani[2]
[1]*M.E. Student, Department of Applied Electronics, TPGIT, Vellore, India*
[2]*Assistant Professor, Department of ECE, TPGIT, Vellore, India*

***Abstract*: Binary division usually use redundant representation of partial remainders and quotient digits. VLSI realizations of digit-recurrence allows for fast carry-free computation of the next partial remainder, and the latter leads to reduce number of the required divisor multiples. The binary carry save (CS) number system is prevalent in the representation of partial remainders, and redundant high radix representation of quotient digits in order to reduce the cycle count. Design a space containing four division architectures. These are based on binary CS or radix-16 signed digit (SD) representations of partial remainders. The other hand, they use full or partial pre-computation of divisor multiples. The uses of smaller multiplexer at the cost two extra adders, where one of the operands is constant within all cycles. The quotient digits are represented by radix-16 Singed Digits. Synthesis-based on the evaluation of VLSI realizations is the best work and the four proposed designs compute to reduced power and energy figures in the proposed designs at the cost of more area and delay measures.**

***Keywords*: Carry save (CS), digit recurrence binary division, energy efficiency, radix -16 signed digit (SD), redundant number system**

1. Introduction

Division is the less frequent operation among the four basic arithmetic operations that are carried out within the execution of a typical task on digital processors. On the other hand, it is the most complex and time consuming operation. VLSI realization of dividers is generally based on two classes of algorithms, namely, subtractive (aka digit recurrence) and multiplicative (aka functional). . Quotient digit selection (QDS) is very simple in conventional binary division algorithms, such as Non- restoring division scheme. Where the next quotient bit is obtained just by examining the sign of partial remainder. However, in order to reduce the number of recurrences, radix-2h (e.g., h = 4) division schemes have been proposed at the cost of more complex QDS, since one out of 2h possible digit values is to be selected. This is undertaken via comparing the partial remainder with a set of divisor multiples [2]. In order to reduce the generation, cost of such multiples, the quotient digits are often selected from a signed digit (SD) set [3] (e.g., [−2h−1, 2h−1]), and converted on the fly into the desired binary output. On the other hand, the SD representation of partial remainders has been employed to enable borrow free subtraction that shortens the cycle time. However, sign detection of SD numbers, which is required in QDS, is not a trivial operation. Despite the less frequent occurrence of division in comparison with other basic operations, the several addition operations that are embedded within a digit recurrence division contribute to extra energy consumption. The division operation can be defined as,

A = QD + R

Where,

A – Dividend
B – Divisor
Q – Quotient
R – Remainder

In hardware realization of digit recurrence division algorithms, it is often postulated that A < B, and the divisor is a normalized fraction, such that in radix-2h division 1/2h ≤ B< 1. Equation (1) describes the j th recurrence, where W[j], q j+1, and Q[j], represent the j th partial remainder, the next quotient digit and the partial quotient, respectively. In addition, 0 ≤ j < n, W[0] = A, Q[0] = 0, and n denotes the precision of B and Q W[j + 1] = 2hW[j] − q j+1B Q[j + 1] = Q[j] + 2−h(j+1)q j+1. QDS can be simplified via selecting q j+1 from an SD set [−α, α].Use the radix-16 digit set [−10, 10], where the next quotient digit is obtained as q j+1 = 4qhj+1+ql j+1, with qhj+1, qlj+1 ∈ [−2, 2]. The common QDS approach is comparing the shifted partial remainder with a set of comparison constants (i.e., a set of selected divisor multiples), such that q j+1 is selected based where Mk and M k+1 represent two consecutive comparison constants, and k ∈ [−α, α]. In practice, all the required comparisons take place in parallel.

$$Mk \leq 2Hw\ [\ j\ ] < Mk+1\ \_\Rightarrow q\ j+1 = k \quad (1)$$

$$-\rho B \leq W[j + 1] \leq \rho B,\ \rho = (\alpha/(2h - 1) \quad (2)$$

To speed up the partial remainder computation (PRC), the partial remainders are often represented a redundant number system.

The commonly used redundant number system that is required for the representation of partial remainders is the binary carry save (CS), which roughly doubles the required number of bits for representing the same value. Although the extra register cost might be affordable, the corresponding extra power dissipation could be in question. There are less costly redundant number systems that represent the same range of numbers exploiting less number of additional bits. For example, the case of redundant digit floating point arithmetic uses the radix-16 maximally redundant SD (MRSD) representation that requires only 25% extra bits.

A. *Radix-16 signed digit*

A two stage algorithm for fixed point radix-16 signed digit division is presented. The algorithm uses two limited precision radix-4 quotient digit selection stages to produce the full radix-16 quotient digit (0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15). The algorithm requires a two-digit estimate of the (initial) partial remainder and a three-digit estimate of the divisor to correctly select each successive quotient digit.

B. *Digit recurrence binary division*

Prevent a radix-10-digit recurrence division algorithm that decomposes the quotient digits into three parts and requires only the computation of two and five times the divisor.

C. *Carry save*

Carry save is a type of digital adder, used in computer micro architecture to compute the sum of three or more n-bit number in binary.

2. Proposed system

Static and semi dynamic DMGS and two different representations for partial remainders provide us with a design space based on the following options.

1) Radix-16 quotient digit set: This choice, as in the previous relevant works, leads to the reduced number of cycles versus the direct generation of quotient bits.

2) SD representation of quotient digits: Use [−9, 9] radix-16 SD set for the intermediate representation of quotient digits.

3) Semi dynamic DMG: The [−9, 9] multiples of divisor that are needed in the PRC are normally obtained within the initialization cycle, where ten-way multiplexer is required for selecting q j +1 d.

4) Use of redundant number systems for PRC: The advantages of semi dynamic DMG also use CS due to doubling representation storage does not seem to be a proper choice when lower power dissipation is desirable. The other choice is use higher radix redundant number systems for partial remain representation.

A. *Conventional method*

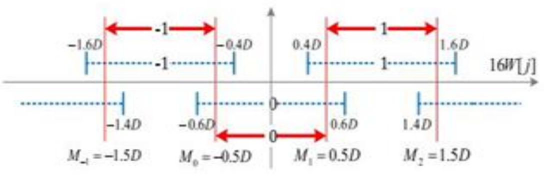

Fig. 1. Overlapped zones for specific quotient digit values

The quotient digit set of our choice is [−9, 9]. In general, the next quotient digit q j+1 should be selected from [−α, α], such that the convergence condition that is described by (3) where in this case (i.e., α = 9), ρ = (a / (16 − 1)) = (9/15) = 0.6. The convergence condition (−0.6d ≤ w [j + 1] ≤ 0.6d) is partially unfolded as in fig.1, which suggests the comparison of the partial remainder with a set of divisor multiples (e.g., −1.6d and −0.4d, for q j+1 = −1) in order to decide the value of the next quotient digit. However, for some w[j] values (e.g., 16w[j] = −.5 d), there may be more than one valid q j+1. For example, see the overlapped zone between the dashed lines for q j+1 = −1 and q j+1 = 0. To ease the qds process and reduce the number of comparisons, it is common to pre compute a set of comparison constants, as fixed multiples of divisor, to be used for exact qds. the proper range of mk s, fork ∈ [−9, 9], therefore, an ease to compute choice is mk = (k + 0.5) d, which leads to the case that the exact interval of 16w[j] (for a particular value of q j+1 = −1) falls between m−2 = −1.5d and m−1 = −.5d (see the corresponding bold arrows in fig. 2).

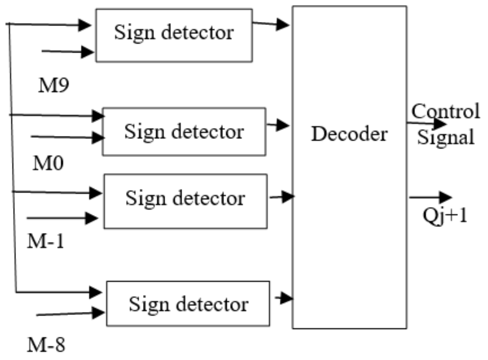

Fig. 2. QDS Architecture

$$d\,(k + 0.4) \le mk \le d\,(k + 0.6) \quad (3)$$

B. *CS-4 and MRSD-4 designs*

On the other hand, the comparison of an SD or CS number (i.e., partial remainder) with a non-redundant one (i.e., comparison constants) cannot be trusted to a digit by digit comparator (most significant digits first). Replacing the operands of (4) with the corresponding truncated operands

$$Mk - 16^{-t} < (Mk)t \le (16W\,[\,j\,])t < 16W\,[\,j\,] + 16^{-t} \Rightarrow Mk - 2 \times 16^{-t} - k\,D < 16W\,[\,j\,] - k\,D = W\,[\,j + 1\,] \quad (4)$$

$$16W\,[\,j\,] - 16^{-t} < (16W\,[\,j\,])t < (Mk)t \le Mk \quad (5)$$

$$Mk + 16^{-t} - (k - 1)D > 16W\,[\,j\,] - (k - 1)D = W\,[\,j + 1] - \rho\,D < M\,k - 2 \times 16^{-t} - k\,D \Rightarrow 2 \times 16^{-t} + k\,D\rho\,D < Mk \quad (6)$$

$$Mk + 16^{-t} - (k - 1)\,D < \rho\,D \Rightarrow Mk < \rho\,D - 16^{-t} + (k - 1)\,D \quad (7)$$

Recalling that ρ = 0.6 and 1/16 ≤ D < 1, and combining the inequalities (6) and (7), we get at the following:

$$\rho D + 2 \times 16^{-t} + k\,D < \rho\,D - 16^{-t} + (k - 1)\,D \Rightarrow 3 \times 16^{-t} < (2\rho - 1)\,D = 0.2 \times D \Rightarrow 16t > 15/D \quad (8)$$

Since the latter should hold for all the values of D (in particular D = 16−1), we need to assert 16 t > 240, which leads to t > 1.

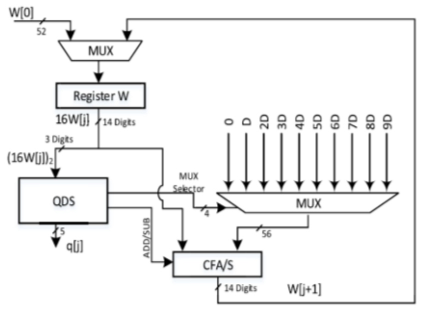

Fig. 3. PRC for designs CS-10 and MRSD-10

C. *CS-4 and MRSD-4 designs*

In order to utilize a smaller selector of divisor multiples, propose the architecture of fig.2, where again qds box represents fig.3, whose output control signals are shown as “mux selector” and “add/sub.” let q j+1 = 6α + β, where α ∈ [−1, 1] and β ∈ [0, 3]. The overlapped PRC computes w [j + 1] = 16w[j]+6αd via a cfa and a cfs, since w[j] is represented in binary cs or radix-16 mrsd formats. The operation of this part overlaps with the qds. the rest of prc is carried out after qds, where βd is selected via the four-input multiplexer, followed by computing w [j + 1] = w [j + 1]’ + βd, via another cfa.

The straightforward PRC would use a unified carry-free adder/subtract (CFA/S) and a 10:1 multiplexer, for the quotient digit set [−9, 9].

D. *Initialization AN clean –up*

Converting x to w [0]:

This is a cost- and delay free operation. In case of CS designs, zero-valued bits are inserted as the second bits of CS representation. However, in MRSD cases, a zero-valued negabit (with logical status 1) is added per each four bits of x to lead to the radix-16 MRSD representation of w [0].

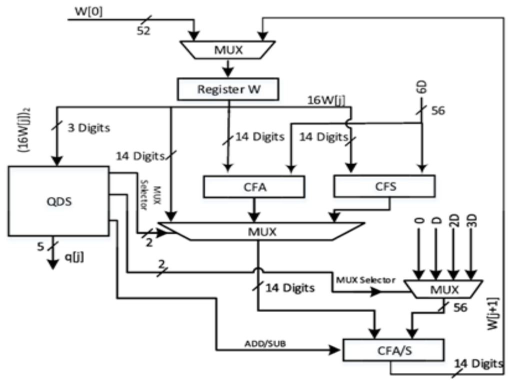

Fig. 4. PRC for designs CS-4 and MRSD-4

Parallel computation of the divisor multiples:

a) CS-10 and MRSD-10 designs: there are four shifters (for 2d, 4d, 6d, and 8d) and four parallel adders (for 3d = 2d + d, 5d = 4d + d, 7d = 8d − d, and 9d = 8d + d).

b) CS-4 and MRSD-4 designs: there are only two shift operations (for 2d and 6d), and one addition (for 3d = 2d + d).

Parallel pre-computation of the truncated comparison constants: This is done, as in the following expressions for m0 to m8.

CS-10 and MRSD-10 designs:

(m1 = d/2, m2 = 3d/2, m3 = 5d/2, m4 = 7d/2, m5 = 9d/2, m6 = 5d + m1, m7 = 6d + m1, m8 = 7d + m1, m9 = 8d + m1).

CS-4 and MRSD-4 designs:

(m1 = d/2, m2 = 3d/2, m3 = 2d + m1, m4 = 3d + m1, m5 = 4d + m1 m6 = 4d + m2, m7 = 6d + m1, m8 = 6d + m2, m9 = 8d + m1).

Other comparison constants are obtained as m−k = −mk+1, for 0 ≤ k ≤ 8. Note that the implementation of initialize block uses only one register for storing the d value, while the generated constants are directly passed to the prc unit.

3. Conclusion

Synthesis in this proposed technique for Binary-64-bit division with redundant number system, design space containing four architectures based on the pre-computation of divisor multiples, CS (Carry save) or MRSD (Maximally Redundant Signed Digit) is explored using HDL simulation.

References

[1] M. Ercegovac and T. Lang, Digital Arithmetic. San Mateo, CA, USA:Morgan Kaufmann, 2003.
[2] D. L. Harris, S. F. Oberman, and M. A. Horowitz, “SRT division architectures and implementations,” in Proc. 13th IEEE Symp. Comput. Arithmetic, Jul. 1997, pp. 18–25.
[3] A. Avizienis, “Signed-digit number representations for fast parallel arithmetic,” IRE Trans. Electron. Comput., vol. 10, no. 3, pp. 389–400, Sep. 1961.
[4] H. A. H. Fahmy and M. J. Flynn, “The case for a redundant format in floating point arithmetic,” in Proc. 16th IEEE Symp. Comput. Arithmetic, Santiago de Compostela, Spain, Jun. 2003, pp. 95–102
[5] S. Gorgin and G. Jaberipur, “A family of high radix signed digit adders.” in Proc. 20th IEEE Symp. Comput. Arithmetic, Tübingen, Germany, Jul. 2011, pp. 112–120.
[6] W. Liu and A. Nannarelli, “Power efficient division and square root unit,” IEEE Trans. Comput., vol. 61, no. 8, pp. 1059–1070, Aug. 2012.
[7] J. Ebergen and N. Jamadagni, “Radix-2 division algorithms with an overredundant digit set,” IEEE Trans. Comput., vol. 64, no. 9, pp. 2652–2663, Sep. 2015.
[8] A. Nannarelli and T. Lang, “Low-power divider,” IEEE Trans. Comput., vol. 48, no. 1, pp. 2–14, Jan. 1999.
[9] A. Nannarelli, “Performance/power space exploration for binary64 division units,” IEEE Trans. Comput., vol. 65, no. 5, pp. 1671–1677, May 2016
[10] E. Antelo, T. Lang, P. Montuschi, and A. Nannarelli, “Digit-recurrence dividers with reduced logical depth,” IEEE Trans. Comput., vol. 54, no. 7, pp. 837–851, Jul. 2005.
[11] A. Nannarelli and T. Lang, “Low-power division: Comparison among implementations of radix 4, 8 and 16,” in Proc. 14th IEEE Symp. Comput. Arithmetic, Apr. 1999, pp. 60–67.
[12] G. S. Taylor, “Radix 16 SRT dividers with overlapped quotient selection stages: A 225 nanosecond double precision divider for the S-1 Mark IIB,” in Proc. IEEE 7th Symp. Comput. Arithmetic (ARITH), Jun. 1985, pp. 64–71.
[13] T. M. Carter and J. E. Robertson, “Radix-16 signed-digit division,” IEEE Trans. Comput., vol. 39, no. 12, pp. 1424–1433, Dec. 1990.
[14] G. Jaberipur and B. Parhami, “Efficient realisation of arithmetic algorithms with weighted collection of posibits and negabits,” IET Comput. Digit. Techn., vol. 6, no. 5, pp. 259–268, Sep. 2012.

[15] H. Nikmehr, B. Phillips, and C.-C. Lim, “Fast decimal floating-point division,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 14, no. 9, pp. 951–961, Sep. 2006.

[16] A. Kaivani, A. Hosseiny, and G. Jaberipur, “Improving the speed of decimal division.” IET Comput. Digit. Techn., vol. 5, no. 5, pp. 393–404, Sep. 2011.