Efficient Implementation of Constant Coefficient Division Under Quantization Constraints

JUAN A. LOPEZ, GABRIEL CAFFARENA, RUZICA JEVTIC, CARLOS CARRERAS, OCTAVIO NIETO-TALADRIZ
Electronics Engineering Department
Technical University of Madrid
E.T.S.I. Telecomunicación, Ciudad Universitaria, s/n, 28036, Madrid
SPAIN
http://www.die.upm.es/personal/JA-lopez.html.en

Abstract: Division is one of the basic operations of arithmetic algorithms, but the cost associated to its hardware implementation exceeds the reasonable limits for most dedicated architectures. This paper provides a systematic algorithm that (a) transforms the constant coefficient division into a constant coefficient multiplication, selectable under some given constraints, and (b) optimizes the resulting multiplier by analyzing the quantization noise inherent to the finite wordlength implementation process. Consequently, this algorithm achieves reduced-area, high-speed constant coefficient dividers that maintain the accuracy in the range of represented numbers. The theorems and presented results confirm that the proposed algorithm computes the most suitable fraction under a given set of noise constraints.

Key-Words: Computer Arithmetic, Division Algorithms, Constant Division, Constant Multiplication, Dedicated Hardware, Embedded Systems, Implementation, Optimization, CAD Tools.

1 Introduction

Although extensive literature exists describing the theory of division [1]-[5], the implementation of this operation in dedicated hardware is too expensive for many applications. Generic division algorithms are divided into the following classes [6]: digit recurrence [7],[8], functional iteration [9],[10], very high radix [11],[12], table look-up [13] and variable latency. Many practical architectures also combine several of these classes to achieve high speed and low latency, and to surpass given accuracy constraints [6]. Consequently, resulting architectures are exceedingly complex, and they must be simplified before its hardware implementation.

Conversely, constant coefficient dividers [14]-[19] significantly reduce the implementation requirements of the operator, and their application is more suited to embedded applications. Their properties have been deeply analyzed to generate efficient binary to digital converters [15], but the obtained results are focused on coefficients whose values are factorized in terms 2^k and $2^k \pm 1$, so they are not valid in the general case. Due to this fact, the most common types of implementation procedures perform constant multiplications of the quantized reciprocal values [16]-[19]. These procedures improve the performance of the divider (in terms of area, speed, latency, etc.), but they also introduce certain amount of quantization noise.

This paper analyses the quantization variations inherent to the finite wordlength implementation of constant coefficient dividers, and proposes some alternatives to provide more efficient implementations. Savings in resources directly improve area and speed, but the proposed procedure allows to minimize any other metric, such as power consumption. The theorems and results given prove that the optimal coefficients achieve significant reductions in area, while preserving the quantization noise below the specified constraint.

The structure of the paper is as follows: Section 2 presents the basic algorithm and some theorems to formalize the selectable accuracy of the divider. Section 3 explains several alternatives to improve the application of the basic algorithm. Section 4 presents the comparative results for a representative set of constant dividers. Finally, section 5 summarizes the conclusions of this work.

2 Proposed Algorithm

Even though algorithms for generic division have evolved remarkably fast in the past decades, there has been little evolution in algorithms to perform constant dividers [14]-[19]. The formal steps traditionally followed to perform this operation are:

1. Obtain the rational representation, q, of the reciprocal of divisor d.
2. Truncate q to the number of bits selected to perform the operation, initially 1.
3. Compute the output difference or noise power for the range of the input signal.
4. If the specifications are not met, increase the number of bits and return to step 2.

This method is intuitive and it rapidly returns a valid constant coefficient, but the final value is restricted to a quantized rational representation of $1/d$. As it will be shown, there are many other constants that meet the specifications (constant q is only one of this set), and the proposed algorithm supports the selection of other constants in order to improve the efficiency of the final divider.

The basic algorithm proposed here to generate efficient implementations of constant coefficient dividers has the following iterative scheme:

1. Select a power of two, 2^k, initially 2^0.
2. Find the nearest multiple of divisor d to the power of two, 2^k, called A here.
3. Multiply numerator and denominator of $1/d$ by A to obtain the equivalent fraction A / D.
4. Substitute the denominator of the equivalent fraction, D, by 2^k. Consequently, the computed fraction, $A / 2^k$, only requires one constant coefficient multiplication and one k-bit shift.
5. Calculate the maximum difference, or any other selected criteria, between the ideal and the quantized outputs in the valid range of the dividend.
6. If the specifications are not met, increase k and return to step 2.

The main features of this algorithm are: (i) the divider is automatically normalized (i.e. the algorithm detects the leading '1' and operates the following k-1 bits of q), thus it automatically computes the result with maximum precision; (ii) the parameters of this procedure, specially the end conditions, are easily specified; and (iii) in each iteration, the multiplying constant A is selected according to the target criteria: minimum implementation complexity, maximum Signal-to-Noise Ratio (SNR), maximum range with outputs identical to the ideal divider, etc.

The following example compares the behavior of the two procedures. Let divisor d be equal to 10, dividend x be an 8-bit signed integer value, and let the result y be represented using 8 fractional bits. Assume that two's-complement and truncation are the representation and quantization strategies.

The value of q that performs the required division is $0.0001100110011..._b$. Using the 4-step algorithm, q is truncated and represented as 0.00011001_b, so dividend x is multiplied by 0.09765625, instead of 0.1 as it should. This difference can be significant depending on the specific value of x and the required precision of the result. In this example, the maximum difference is ±0.3. The application of the proposed algorithm, assuming identical hardware resources, returns A = 13 and 2^k = 128. In this case, x is multiplied by 0.1015625, and the maximum error is reduced to ±0.2. The increased performance of this result clearly indicates the beneficial application of the proposed algorithm.

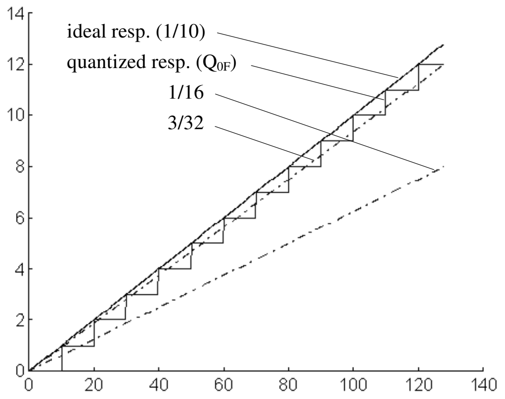

Fig. 1. Ideal and quantized responses of the divider (d = 10). Last two fractions computed by the algorithm when the output is truncated to zero fractional bits.

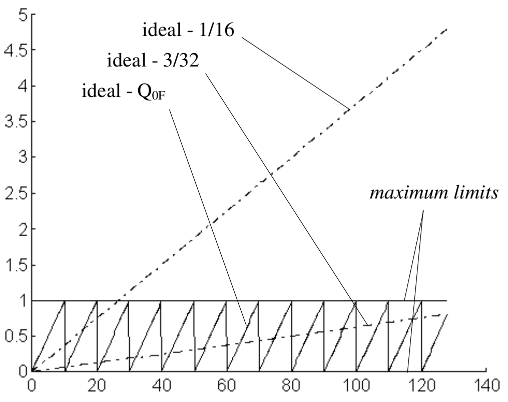

Fig. 2. Differences between the ideal response and the computed ones. Maximum limits allowed for the variations when the output is truncated to zero fractional bits.

Figure 1 shows the ideal response and the last two fractions of the algorithm for d = 10. In this case, y is quantized to zero fractional bits, Q_{0F} (for the sake of clarity), instead of the 8 fractional bits as in the example. The iterative search finishes when the difference between the two quantized results is zero in the whole range of values of the dividend. The quantized response is also given. Figure 2 shows the differences between the computed responses and the ideal one, and it also indicates the maximum variations allowed in this case.

2.1 Mathematical Analysis

The underlying principle of the presented algorithm is the following: The result of the divider is correct as long as the ideal and the computed quantized quotients provide the same values. Moreover, further improvements or modifications are allowed iff this condition remains satisfied. These improvements can be carried out as a result of the theorems presented below.

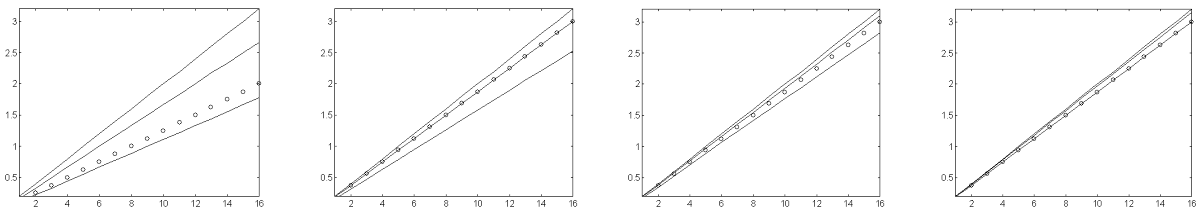

Fig. 3. First four iterations of the algorithm ($d = 5$) when it searches the best fraction *below* the ideal response.

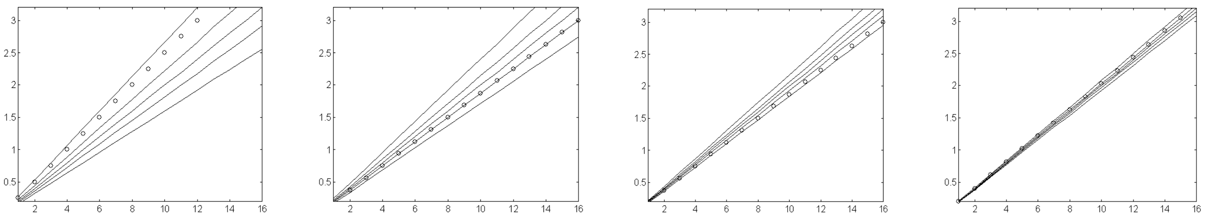

Fig. 4. First four iterations of the algorithm ($d = 5$) when it searches the best fraction *nearest to* the ideal response.

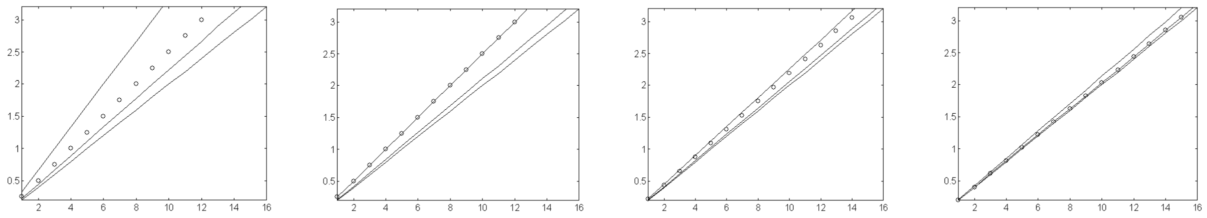

Fig. 5. First four iterations of the algorithm ($d = 5$) when it searches the best fraction *above* the ideal response.

Theorem 1: The SNR introduced by the computed fraction is independent of the range of the input signal.
Theorem 2: As the algorithm evolves, the SNR of the computed fractions tends to infinity.
Corollary: The SNR can be made as large as required by increasing the number of bits of the computed fraction.
Theorem 3: When it searches for the nearest fraction below the ideal response (*lower-nearest criterion*), the maximum and minimum deviations of the iteration-k computed fraction are bounded by

$$\frac{A}{2^{2k}} \le \left| e_{lower} \right| \le \frac{A \cdot d}{2^{2k}} \quad \textbf{(1)}$$

This theorem assumes $k >> \log_2 d$, which is the general case after some iterations of the algorithm.
Theorem 4: When it searches for the nearest fraction (*nearest criterion*) and the nearest fraction *above* the ideal response (*upper-nearest criterion*), the maximum and minimum deviations of the iteration-k computed fraction are respectively bounded by

$$\pm\frac{A}{2^{2k}} \le \left| e_{nearest} \right| \le \pm\frac{A \cdot d}{2^{2k+1}} \quad \textbf{(2)}$$

$$\frac{A}{2^{2k}} \le \left| e_{upper} \right| \le \frac{A \cdot d}{2^{2k}} \quad \textbf{(3)}$$

This theorem also assumes $k >> \log_2 d$.

The mathematical proofs of these theorems have been extracted from this section to emphasize the operation and conclusions of the proposed algorithm. However, the examples provided throughout this paper confirm the validity of this analysis.

Figures 3-5 show the first four iterations of the algorithm for divisor $d = 5$ in the three cases. Each plot provides the ideal response of the divider (diagonal line in each plot), the computed fraction (circles), and the maximum and minimum error bounds (closer and more separated lines from the ideal response, respectively) allowed for the computed fraction in each case.
Final remark: For simplicity, the theorems presented in this section assume integer arithmetic. However, the conclusions given here are also valid in rational dividers by applying the required normalization.

3 Alternative Approaches

This section describes some interesting variations of the basic procedure described in the previous section: (i) constant multiplication and division by rational values; (ii) use of range decomposition; and (iii) selection of the multiplicative constant A, according to the underflow strategy, to enhance the results of the proposed algorithm.

3.1 Rational Multiplication and Division

In the general case, constant factors are rational numbers, represented as n / d, being n and d two integers. Since integer arithmetic is not required in the calculations, the previous discussion is also valid for rational dividers $d' = d / n$. Consequently, the proposed algorithm computes rational multiplications or divisions without changes. Some applications of the proposed algorithm for fractions of the form n / d are:

1) <u>To obtain a numerator near a power of two:</u> In this approach the computed fraction takes the form $2^k / A{\cdot}d$. One division is still required, but this operation can be easier to perform. In this case, the SNR also tends to infinity as the algorithm evolves, and it can be made as large as required.

2) <u>To compute several fractions using the same divider:</u> Fractions take the general form N / D, but they share specialized dividers. This method requires one additional multiplication per fraction, but the overall complexity is reduced.

3.2 Range Decomposition

Another interesting optimization technique consists in splitting the input range into several adjacent intervals and computing approximating functions for each part. This operation is shown in the following example.

Let x lay in the range [0,127], d be equal to 5, and let y be implemented using zero fractional bits. Figure 6 shows the differences between two approximating functions and the ideal one, and the maximum limits permitted for the variations. The first function, $f_1(x) = 3x/16$, provides the exact results in the range $[0, L_1]$, and the second one, $f_2(x) = 3x/16 + 0.75$, in the range $[L_2,128]$. Consequently, $f_1(x)$ is used when $0 \le x \le 63$, and $f_2(x)$ is used otherwise. Since the only difference between the two functions is the constant value 0.75, this approach obtains the required results with very little increase of the hardware resources.

3.3 Selection of the Multiplicative Constant

The last part of this section provides some guidelines to select the multiplicative constant A. Obviously, the best performance of the algorithm is achieved when the selection of A compensates the underflow effects of the multipliers. Consequently,

a) if truncation is selected as the underflow strategy (in the partial products or in the final sum), the computed results are below the ideal ones, so the *upper-nearest* criterion is preferred.

b) if rounding is selected as the underflow strategy, the computed results can be above or below the ideal ones, so the *nearest* criterion is preferred.

c) if ceiling is selected as the underflow strategy, the computed results are above the ideal ones, so the *lower-nearest* criterion is preferred.

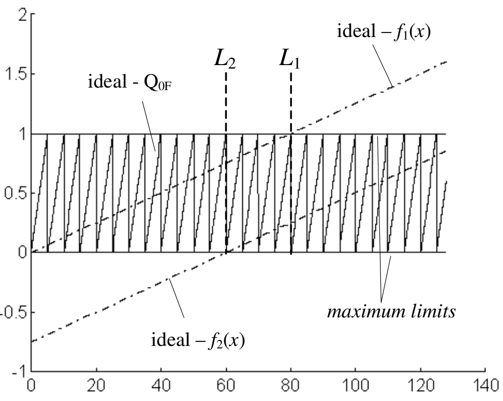

Fig. 6. Computation of the best fraction nearest to the ideal response ($d = 5$) by decomposition into 2 adjacent intervals.

3.3.1 A Particular Approach

Among the presented alternatives, one combination is of particular interest: implementation of integer dividers with truncation of the partial products and the final result, and selection of the *upper-nearest* criterion to compute the multiplicative constant of the algorithm. In this case, the algorithm transforms the constant coefficient division into a constant multiplication, truncation reduces the number of adders required, and the upper-nearest criterion compensates the variations introduced in the approximation process.

Consequently, this particular approach provides significant improvements in area and speed, it does not alter the quantized results, and it can still be optimized using existing techniques for low-power, increased speed, etc. The application of this approach obtains savings of approximately 60% in area with respect to the others criterions, while accuracy is preserved within the same limits. Section 4 provides further comparative results of this approach.

<u>Definition:</u> The *range of exactitude*, R, of a given operator is defined as the range where the quantized results are identical to the mathematical ones.

<u>Theorem 5:</u> In constant coefficient dividers without truncation or rounding in the partial products, the range of exactitude is given by

$$0 \le R < \left\lfloor \frac{Ad}{Ad - 2^k} \right\rfloor - 1$$

A direct consequence of theorems 2 and 5 is that the range of exactitude of constant coefficient dividers can always be increased to meet the specifications.

4 Comparative Results

This section provides comparative results for the most representative cases among the alternatives explained in the previous sections: (i) exact computation of integer division using constant multipliers, (ii) appro-

TABLE 1. IMPLEMENTATION OF EXACT DIVIDERS

d	*A*	*k*	*R*	Area(sl.)
3	683	11	0-2047	17
5	205	10	0-1023	17
23	713	14	0-1091	21

TABLE 2. CONSTANT DIVIDERS WITH CONSTRAINED NOISE ($d = 5$, $R = 0\text{-}1023$, SNR > 90 dB)

A	*k*	SNR(dB)	M.E.	M.S.E.	Area(sl.)
205	10	Infinity	0	0	17
103	9	102.3128	1	0.5	14
51	8	102.3519	1	0.4980	13

ximation of integer division under noise constraints, and (iii) approximation of integer division under noise constraints using multipliers with truncated partial products (also called left-sided multipliers [20]).

4.1 Computation of the Exact Division

Table 1 shows the computed fractions for three constant dividers (3, 5 and 23, respectively) when dividend *x* is a 10-bit positive integer. Constants *A* and *k* are obtained by selecting the upper-nearest criterion in the proposed algorithm. The hardware resources are provided in terms of the number of slices required by a Xilinx XC2V40 FPGA device to implement the constant multiplier, and redundant additions have been optimized to reduce the global area of the operator.

4.2 Division under Noise Constraints

The next example compares several implementations of one of the dividers ($d = 5$) under certain noise constraints given (SNR > 90 dB). Dividend *x* is equal to the previous case. Table 2 provides the pairs *A* - *k*, the range of exactitude, and the hardware required to perform the constant division (M.E. and M.S.E. stand for Maximum Error and Mean Squared Error, respectively). Note that the exact divider is also included in this case, since it fits the noise requirements. The specifications are not met for values of *k* smaller than 8. Among the three different choices, these results clearly state that the third case ($A = 51$, $k = 8$) is the one that requires minimum hardware.

4.3. Division under Noise Constraints Using Left-Sided Multipliers

In this case, the input range and the target SNR are the same than in the previous example, but truncation of partial products is also included. Left-sided multipliers introduce extra noise in the computation of the division, resulting in further degradations on the final results. The goal is to truncate the maximum number of bits, *t*, and still meeting the specifications.

TABLE 3. EFFECT OF INCREASING CONSTANT *A* IN LEFT-SIDED MULTIPLIERS ($d = 5$, $b = 10$ BITS).

A	*k*	*T*	SNR(dB)	M.E.	M.S.E.	Area(sl.)
205	10	10	80.6984	4	4.3118	9
208	10	10	90.4305	3	1.6406	7

TABLE 4. IMPLEMENTATION RESULTS OF CONSTANT DIVIDERS USING LEFT-SIDED MULTIPLIERS ($d = 5$)

A	*k*	*t*	SNR(dB)	M.E.	M.S.E.	Area(sl.)
208	10	10	90.4305	4	1.6406	7
204	10	9	90.2887	3	1.6641	10
205	10	9	94.3082	2	1.1133	10
206	10	9	101.1003	2	0.5645	10
206	10	8	106.4655	1	0.3301	10

The steps followed in the computation of the best implementation using left-sided multipliers are:

1. Compute the values of *A* and *k* that obtain exact results ($A = 205$, $k = 10$).
2. Compute the left-sided multiplication for partial products truncated to 10 bits ($t = 10$).
3. If the obtained SNR does not meet the specifications but increases its value, increase the value of constant *A*. If the SNR decreases its value, repeat the whole process decreasing *t*.

The effect of increasing the value of constant *A* is shown in figure 7. Using $A = 205$, $k = 10$ and a left-sided multiplier with $t = 10$, the curves that characterize the operator are always below the exact one. If constant *A* is increased up to 208, the new curve is situated under the exact one for small input values (figure 7.b), and above it for large ones (figure 7.c). Thus, after increasing *A*, the maximum error, the mean square error and the SNR improve, as shown on table 3.

Table 4 displays the results for $t = 8$, 9 and 10 bits. It can be observed that all the results achieve low-area and low-power implementations, and that the first one ($t = 10$) provides the best selection in terms of hardware requirements.

Since hardware implementation is highly dependent on the constant selected in the process, a rule of thumb in this case is to compute several constants that meet the specifications, as in section 4.2, and to select the one with the minimum hardware requirements.

Finally, the examples developed throughout this section confirm that the algorithm presented is a powerful tool that helps the designer in the selection of parameters that compute constant coefficient dividers under noise constraints. Another important remark is that the use of nonlinear operators, such as left-sided multipliers, can lead to significant savings in hardware.

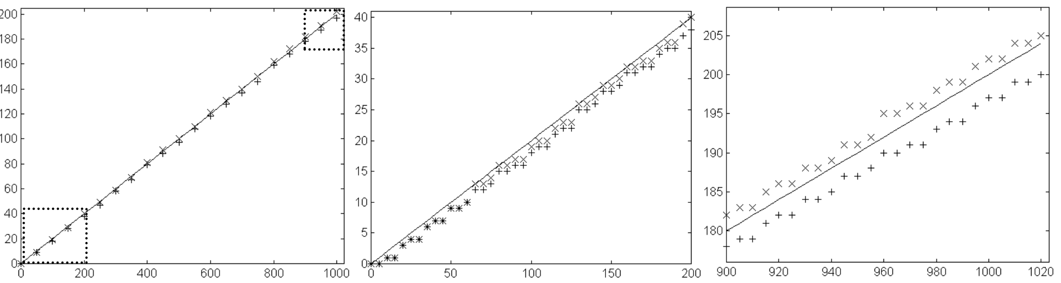
Figure 7. Integer division using the left-sided multipliers and truncation ($d = 5$, $R = 0$-1023). (a) full input range, (b) amplification for small input values, (c) amplification for large input values. Each plot contains: (i) exact results (solid line); (ii) curve for $A = 205$, $k = 10$ (+); (iii) curve for $A = 208$, $k = 10$ (×).

5 Conclusions

This paper has analyzed the quantization effects in the implementation of constant coefficient dividers. Properties, theorems and alternative approaches have been derived to obtain the most suitable implementation under a given set of constraints. These constraints can be described directly (accuracy of the result, SNR, etc.) or indirectly (area, speed, latency, etc.).

The computation process has been formalized in an algorithm that computes the best fraction in a selectable way. Among the alternatives proposed to improve the implementation efficiency of the divider, special emphasis has been made in a specific approach that provides efficient, low-cost, dividers and preserves the efficiency of the operator due to the cancellation of the variations. Examples have shown that application of the proposed approach provides up to 60% savings in area, with increased the speed of the divider.

Acknowledgement: This work has been partially supported by the spanish “CICYT - Subdirección General de Proyectos de Investigación” under project number TIC2003-09061-C03-02.

References:

[1] E.E. Swartzlander, Jr: *Computer Arithmetic,* vol. 1, Los Alamitos, CA: IEEE Computer Society Pr., 1990.

[2] M.C. Ercegovac, T. Lang: *Division and Square Root: Digit Recurrence Algorithms and Implementations*. Kluwer Acad. Publ., 1994.

[3] J.M. Muller. *Elementary Functions. Algorithms and Implementation*. Birkhauser, 1997.

[4] B. Parhami: *Computer Arithmetic: Algorithms and Hardware Designs,* New York: Oxford University Press, 2000.

[5] M.J. Flynn, S.F. Oberman: *Advanced Computer Arithmetic Design*, J. Wiley & Sons, New York, 2001.

[6] S.F. Oberman, M.J. Flynn: "Division Algorithms and Implementations". *IEEE Trans. Computers,* vol. 46 (8), pp. 833 -854, Aug 1997.

[7] J.B. Wilson, R.S. Ledley: "An Algorithm for Rapid Binary Division", *IRE Trans. Electr. Comput.* Vol. EC-10, pp. 662-670, 1961.

[8] D.E. Atkins: "Higher-Radix Division Usign Estimates of the Divisor and Partial Remainders". *IEEE Trans. Computers,* vol c-17, 10, pp. 925-934, Oct, 1968.

[9] D. Ferrari: "A Division Method Using a Parallel Multiplier", *IEEE Trans. Electr. Comput.* Vol. EC-16, pp. 224-226, 1967.

[10] M.J. Flynn: "On Division by Functional Iteration". *IEEE Trans. Computers,* vol. c-19, pp. 702-706. 1970.

[11] D. Wong, M. Flynn: "Fast Division usign accurate quotient approximations to reduce the number of iterations". *10th IEEE Symp. Computer Arithmetic,* pp. 191-201, June, 2001.

[12] W. Briggs, D. Matula: "Method and apparatus for performing division using a rectangular aspect ratio multiplier". U.S. Patent Nº 5,046,038, Sept, 1991.

[13] M. Ito, N. Takagi, S. Yajima: "Efficient Initial Approximation and Fast Converging Methods for Division and Square Root". *Proc. 12th Symp. Computer Arithmetic (ARITH12)* pp. 2-9. 1995.

[14] P. Srinivasan: "Generalized approaches to constant division". *PhD Dissertation,* Tulane University, 1968.

[15] P. Srinivasan, F.E. Petry: "Constant-division algorithms" *IEE Proc. Computers and Digital Techniques,* vol. 141, pp. 334 - 340, Nov. 1994.

[16] E. Artzy, J.A. Hinds, H.J. Saal: "A Fast Division Technique for Constant Divisors". *Communications of the ACM,* 19 (2), pp. 98-101, February, 1976.

[17] S.-Y.R. Li: "Fast Constant Division Routines". *IEEE Trans. Computers*, C-34, pp. 866-869, Sept, 1985.

[18] R.Alverson et. al.: "The Tera Computer System". *Int. Conf. on Supercomputing,* pp. 1-6, June, 1990.

[19] R. Alverson: "Integer division using reciprocals", *Proceedings 10th IEEE Symposium on Computer Arithmetic*, pp. 186 -190, 1991.

[20] Trân-Thông: “A New Sum of Products Implementation for Digital Signal Processing”. *IEEE Trans. Circuits and Systems.* Vol CAS-25, nº1, pp. 27-31. Jan, 1978.