On digit-recurrence division algorithms for self-timed circuits

Nicolas Boullis, Arnaud Tisserand

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

On digit-recurrence division algorithms for self-timed circuits

Nicolas Boullis and Arnaud Tisserand

N° 4221

July 2001

THÈME 2

ISSN 0249-6399 ISRN INRIA/RR--4221--FR+ENG

RHÔNE-ALPES

On digit-recurrence division algorithms for self-timed circuits

Nicolas Boullis and Arnaud Tisserand

Thème 2 — Génie logiciel
et calcul symbolique
Projet Arénaire

Rapport de recherche n° 4221 — July 2001 — 13 pages

Abstract: The optimization of algorithms for self-timed or asynchronous circuits requires specific solutions. Due to the variable-time capabilities of asynchronous circuits, the average computation time should be optimized and not only the worst case of the signal propagation. If efficient algorithms and implementations are known for asynchronous addition and multiplication, only straightforward algorithms have been studied for division. This paper compares several digit-recurrence division algorithms (speed, area and circuit activity for estimating the power consumption). The comparison is based on simulations of the different operators described at the gate level. This work shows that the best solutions for asynchronous circuits are quite different from those used in synchronous circuits.

Key-words: Computer arithmetic, division algorithms, SRT tables, asynchronous circuits, self-timed circuits

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38330 Montbonnot-St-Martin (France)
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

Algorithmes de division chiffres à chiffres pour les circuits asynchrones

Résumé : L'optimisation des algorithmes pour les circuits asynchrones nécessite des solutions spécifiques. Du fait des capacités de calcul en temps variable des circuits asynchrones, le temps moyen de calcul doit être optimisé et plus seulement le temps du pire cas. Si de bons algorithmes et des implantations efficaces sont connus pour l'addition et la multiplication asynchrones, seules des solutions simplistes ont été étudiées pour la division. Ce papier compare plusieurs algorithmes de division basés sur des récurrences sur les chiffres du quotient (les critères de comparaison sont la vitesse, la surface et la consommationn d'énergie). Cette comparaison est basée sur des simulations au niveau des portes logiques. Ce travail montre que les meilleures solutions pour les circuits asynchrones sont véritablement différentes de celles admises pour les circuits synchrones.

Mots-clés : Arithmétique des ordinateurs, algorithmes de division, tables SRT, circuits asynchrones, circuits auto-séquencés

1 INTRODUCTION

In clocked digital systems, speed is limited by the worst-case delay of the slowest part of the circuit. The clock was originally introduced to simplify the design of the circuits. This global signal synchronizes all the blocks of the circuit to prevent from latching unstable signals. The clock period is determined using the worst-case delay of the slowest block and an additional margin for limiting mismatch effects [1]. This solution guaranties to only memorize and use stable values. The clocked or synchronous design style has some limitations. Clocking circuits at a high speed is more and more complex and costly. It is very difficult to manage several clock signals, hence the design of circuits with blocks that operate at different speeds is hard. All latches, registers and combinatorial blocks start to work at the same time due to the global synchronization provided by the clock; this leads to noise in the power supply lines and electromagnetic interferences (EMI). Standby mode requires additional hardware resources to manage the local clock signals using gated clock for instance. For some operators, the worst case of the propagation delays is very unlikely and leads to waste time in waiting for the clock signal transition.

Self-timed or asynchronous circuits have been introduced to avoid some of the drawbacks of synchronous circuits. In self-timed circuits, the synchronization is performed directly by the operators. They use a communication protocol that handles the data as well as request and acknowledgment signals. There are many possible implementations and protocols. Self-timed or asynchronous circuits require specific optimization and design solutions. The best algorithms known for clocked circuits are based on the optimization of the worst case of the propagation signals. Asynchronous circuits allow an efficient variable computation time [2]. For this class of circuits, the best solution should be based on the optimization of the average computation time that depends on the input values and not only on the worst case as in synchronous circuits. This new constraint leads to different optimization solutions especially for some arithmetic operations like the addition [3, 4]. Variable-time operators are able to perform very fast computations in those algorithms where the average computation time is significantly different from the worst-case computation time.

Fast arithmetic operators have always been a main goal in computer and high-performance circuit design. Algorithms for addition, multiplication and division have been widely studied in the case of synchronous circuits [5, 6, 7, 8]. Many algorithms have been proposed for implementing division in clocked circuits. The main solutions and some implementation issues are summarized in an article by Oberman and Flynn [9]. In the case of asynchronous circuits, the addition and the multiplication can be efficiently implemented using optimized algorithms. But in the case of the division operation, only straightforward algorithms have been studied. Most of the previous works use a radix-2 SRT algorithm (with quotient digits in $\{-1, 0, 1\}$) and a residual represented using a carry-save number system [10, 11, 12, 13] or a borrow-save number system [14]. A recent high-radix approach [15] presents a speculative radix-64 or radix-128 SRT algorithm.

In this work, we compare several standard algorithms that produce the quotient digit by digit using simple operations such as additions, shifts and small multiplications (one digit by one number products). This kind of algorithms, called "digit-recurrence algorithms" in the literature, is extensively presented in a dedicated book by Ercegovac and Lang [16]. Our comparisons are based on simulations at the gate level. A specific simulator has been developed ($13,000$ lines of C++ code). It uses a C++ description of the circuit at the gate level. The gates are instantiated from a standard cell library that includes some specific asynchronous cells. In order to determine the statistical behavior of the circuit, a lot of simulations are performed using different input values (random generation). For each solution, we can measure the average computation time, its standard deviation and its distribution. The simulator also computes the area required by the circuit. An estimation of the power consumption is determined using the circuit activity (this is reasonable due to non-glitching logic style used in self-timed circuits). Based on the simulations analysis, we discus the results and we point out some possible improvements.

Section 2 introduces self-timed circuits with a specific emphasis on the design style used in this work. After a brief description of the main division algorithms types, Section 3 presents the algorithms that produce the quotient digit by digit using mainly additions, shifts and digit by number multiplications. The simulation model used in this work is presented Section 4. Section 5 presents the obtained results and an analysis of those results. Finally, we conclude and present our future works in this field in Section 6.

2 SELF-TIMED CIRCUITS

They are many possible models and design styles for implementing self-timed circuits. These models depend on the timing assumption used in the circuit. A classification has been proposed by the asynchronous community, and it is presented in Table 1.

self-timed circuit type	timing constraints	limitations	robustness
delay insensitive (DI)	none	unfeasible common gates	$++$
quasi delay insensitive (QDI)	isochronic fork	none	$+$
speed independent (SI)	delay(wire)=0	none	$-$
micropipeline	bounded delays for datapath DI for control	none	$--$

Table 1: Self-timed circuits classification.

The DI circuit class offers arbitrary delays in gates and in wires. Martin [17] has shown that DI circuits have some limitations. He has also shown that the isochronic fork is the smallest assumption that must be added to the DI model for allowing real circuits. Other models with more and more timing assumptions are possible. As an example, the SI model assumes that the delays in the wires can be neglected (this assumption can be dangerous with deep-submicron technologies and automatic routing tools). With stronger and stronger timing assumptions, we arrive at the standard clocked model. In our case, we use the QDI model.

Among all possible implementations of the QDI circuit style, we use the 4-phase handshake protocol and the dual-rail signal coding. There are four phases due to acknowledgment of all transitions as illustrated in Figure 1(a). The data validity is coded at a very low level using the dual-rail bit representation presented in Figure 1(b).

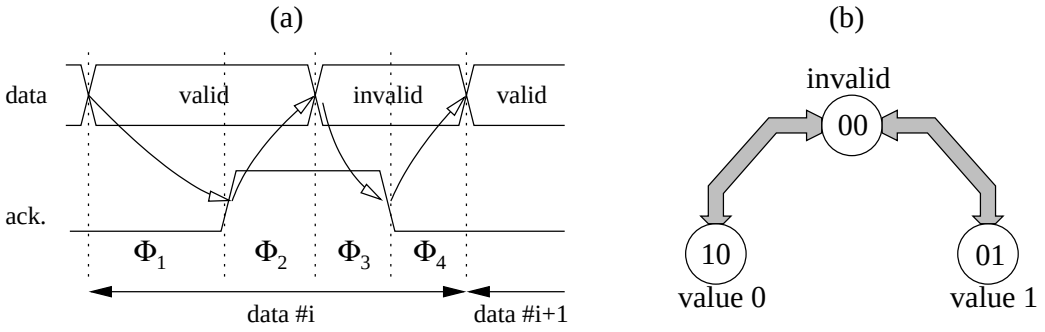


Figure 1: 4-phase handshake protocol (a), dual-rail coding (b).

Due to the Gray-style coding adopted in the dual-rail signal representation, there is no glitch in self-timed circuits. Each transition is a functional one. This can be used for estimating the power consumption. Indeed, the circuit activity can be determined from the netlist. Each signal will change twice for each computation cycle: invalid $\rightarrow$ valid (0 or 1) and valid $\rightarrow$ invalid. The circuit activity is just an approximation of the power consumption because load and fanout problems are neglected. In our case, this approximation makes sense because we compare very similar algorithms.

Self-timed circuits have many advantages over synchronous circuits. The average computation time capability allows the use of simple solutions even for high-performance implementations. In many algorithms, the worst case is very unlikely and its optimization requires a lot of additional resources. As an example, the addition of two n-bit numbers can be computed with a $\mathcal{O}(\sqrt{\log n})$ average computation time algorithm[4] in a self-timed circuit (a theoretical $\mathcal{O}(\log \log n)$ algorithm exists, but it cannot be used for a practical implementation). The best synchronous adder has only a computation time of $\mathcal{O}(\log n)$ (without a

redundant number system). This work will show a similar result in the specific case of digit-recurrence division algorithms. Self-timed blocks can be easily integrated in new designs. The only constraint that must be respected is the handshake protocol. A complete system can be build with several blocks and it will operate as fast as possible. In the case of synchronous circuits, the global clock rate should be determined using the slowest block of the system. Self-timed circuits allow a natural standby mode without additional hardware. Other advantages such as the elimination of the clock distribution problem or low EMI emission can be noticed.

Self-timed circuits also have some drawbacks. The most important today is the lack of design tools. Self-timed circuits can lead to larger circuits because of the multi-rail coding of each bit (a dual-rail coding and an acknowledgment signal in the design style used in this work). Another important problem is the optimization of the code generated by a compiler. The code generation problem is already a very hard problem in the case of synchronous processors. With self-timed processors, this problem will be even harder because of the duration of the instructions that is a probability density function. New dedicated solutions should be studied to allow good optimizations.

3 DIVISION ALGORITHMS

3.1 Notations

In this paper, we use the notations proposed by Ercegovac and Lang [16]. The dividend x and the divisor d are fixed-point n-bit binary numbers such that $\frac{1}{2} \leq d < 1$ and $0 \leq x < d$. In the case of inputs on a larger domain, for floating-point numbers for instance, simple pre- and post-treatments must be used (shifts and a quotient correction after the last iteration). The quotient q is represented using n radix-r digits of the digit set $\mathcal{D}_q$ (i.e. $q = \sum_{i=1}^{n} q_i r^{-i}$). The quotient q and the remainder rem are defined such that $x = qd + rem$ and $0 \leq rem < dr^{-n}$.

3.2 Division Algorithms Classification

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
- table based methods

Digit by digit recurrence algorithms produce the quotient in a serial way with the most significant digit first. As our work deals with this class of algorithms, we will give a more comprehensive description in the next subsection. Those algorithms are based on quite simple operators such as additions, shifts, very small multiplications (one digit by one number product) and some glue logic or small tables for the quotient digits selection. The digit by digit recurrence algorithms produce the quotient and the remainder of the division such as $x = qd + rem$ and $0 \leq rem < dr^{-n}$. This solution is often used for a stand-alone division unit in processors (Intel Pentium, HP PA 8000, Sun UltraSPARC).

Newton-Raphson or Goldschmidt algorithms are based on a functional iteration. The idea is to build an iteration that converges on the quotient value. As an example, the Newton-Raphson algorithm uses the following functional iteration to evaluate the reciprocal $1/b$:

$$y_{j+1} = y_j(2 - by_j)$$

The convergence of this solution is quadratic, the number of correct bits doubles every iteration. The obtained reciprocal should be multiplied by x in order to complete the computation of the quotient. The initial approximation y_0 can be read in a look-up table to reduce the number of iterations. This kind of algorithm requires simple arithmetic operators and full-width multiplications (at least in the last iteration). This solution is used in processors without a division unit (IBM 360/91 and RS/6000, Intel IA64) by the reuse of the other arithmetic or floating-point units.

The last main class of division algorithms is based on table lookups and very simple operations like addition and logical functions. This class leads to very fast operators but it is limited to small precision (up to 24 bits). Depending on the size of the operands, one can use direct approximations, linear approximations or multipartite solutions [18]. These solutions are mainly used in specific circuits.

3.3 Digit-Recurrence Division Algorithms

In this work, we want to determine which division algorithm type should be used for a dedicated division unit in an asynchronous processor. Based on the specific unit criteria, we choose the digit-recurrence algorithms class. A comprehensive description and an evaluation of some implementations for synchronous circuits is presented in a book by Ercegovac and Lang [16].

As the digit-recurrence algorithms produce their output digit by digit (with the most significant one first), we define the quotient after j iterations $q[j] = \sum_{i=1}^{j} q_i r^{-i}$. We can notice that the value of the radix r is very important in this kind of algorithm. Indeed, if we assume that r is a power of 2 (that is quite reasonable), $\log_2(r)$ new bits of the quotient are computed at each iteration. So, the total latency should be reduced for large values of r.

We also define the residual or partial remainder w after j iterations such that $w[j] = r^j(x - dq[j])$. From the residual computed during the last iteration $w[n]$, we can easily deduce the remainder of the division rem.

The digit-recurrence is based on the following residual iteration :

$$\begin{cases} w[0] &= x \\ w[j+1] &= rw[j] - q_{j+1}d \end{cases} \tag{1}$$

This iteration can be implemented using several solutions. Indeed, the method for the new quotient digit q_{j+1} selection is not precised. We use two standard algorithms: the restoring and the SRT[1] algorithms.

Restoring algorithm: The quotient q is represented using a non-redundant number system ($\mathcal{D}_q = \{0, 1, \ldots, r-1\}$ for radix r). This is the "paper-and-pencil" usual algorithm. Its main characteristic is the $r - 1$ full-width comparisons required to deduce the new quotient digit.

SRT algorithm: It was introduced to avoid the full-width comparisons of the restoring algorithm. The quotient is represented using a redundant number system. The idea is based on a faster choice for the values of q_{j+1} by the examination of a few most significant digits of the residual and the divisor. This is possible due to redundant representation of the quotient. A small "error" in the choice of a quotient digit can be canceled in the next iterations. The redundant representation of q allows a speed and area improvement. But, it also requires to convert the result toward a conventional representation.

The computation performed in the recurrence of Equation 1 is quite simple, and it is illustrated in Figure 2. The multiplication of the residual by the the radix is straightforward if r is a power of 2. In this case, the multiplication is a constant shift implemented using wires.

The selection function depends on the algorithm type: restoring or SRT. It is implemented using comparators or tables. This block of the iteration can require an important part of the computation time. In our case, we only implement a radix-2 restoring algorithm and some SRT algorithms with radix between 2 and 16.

The quotient digit by divisor product is quite simple for low radix values. The choice of the digits in $\mathcal{D}_q$ is also important. The product $q_{j+1} \times d$ should be implemented using a few simple operations (additions, constant shifts and complements). This is possible if the digits are small integers (3=2+1, 5=4+1, 7=8-1...).

The last operation of the iteration is the subtraction $rw[j] - q_{j+1}d$. In synchronous circuits, this subtraction is accelerated by the use of redundant number system for the residual w. Most of the time, a *carry-save* representation is used. One can notice that the number system used to represent w can be different from the one used to represent the quotient. The only condition required to simplify the computations is that both should be a power of 2.

[1] SRT stands for Sweeney, Robertson and Tocher the fathers of this algorithm.

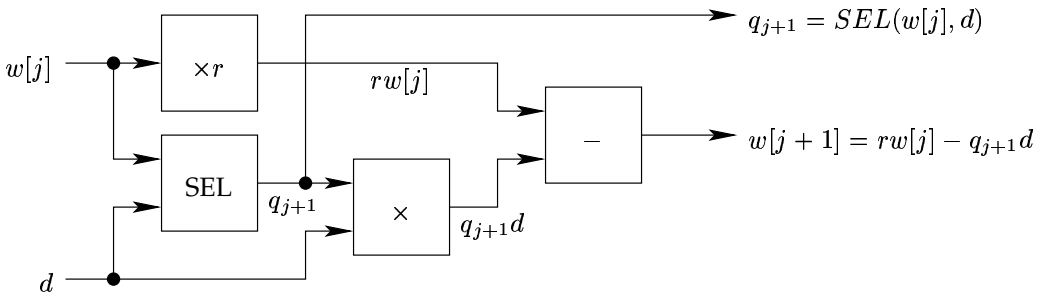


Figure 2: Architecture of the digit-recurrence stage.

3.4 Previous Works on Division Algorithms for Asynchronous Circuits

Table 2 presents the main characteristics of the previous published works. For each reference, the table gives the algorithm type (SRT, restoring, non-restoring, ...), the operands and result size n, the radix r, the digit set $\mathcal{D}_q$ and the number system used for the representation of residue $w[j]$. All the previous works are based on a SRT solution (with small improvements in some cases). Most of them use a simple radix-2 signed-digit quotient representation. In order to reduce the long latency due to the low radix quotient production, one reference presents a high-radix solution.

work	type	n (bits)	r	$\mathcal{D}_q$	w representation
ref [10]	SRT	54	2	$\{-1, 0, 1\}$	carry-save
ref [14]	SRT	32	2	$\{-1, 0, 1\}$	borrow-save
ref [11]	SRT	55	2	$\{-1, 0, 1\}$	carry-save
ref [12]	SRT	55	2	$\{-1, 0, 1\}$	carry-save
ref [13]	SRT	8	2	$\{-1, 0, 1\}$	carry-save
ref [15]	SRT	54	64 and 128	$\{-a, \ldots, a\}$	carry-save

Table 2: Previous works on asynchronous division algorithms characteristics.

For all the previous published works, there is no comprehensive study of the parameter set. Most of the time, the only comparison is performed with a simple radix-2 restoring algorithm.

4 SIMULATION MODEL

4.1 Standard and Asynchronous Cell Library

Specific standard cells are necessary for implementing asynchronous circuits. We use in this work the gate description of a 0.18 μm library that includes some specific asynchronous gates. Figure 3 presents some of those asynchronous cells as well as some more standard ones among the 250+ cells of the library. The delay and area are expressed using arbitrary units. The standard (non asynchronous) inverter is the unit (its delay and area are equal to one).

4.2 Specific Simulator

As the purpose of this work is the comparison of several division algorithms, we have to determine the level of description that allows a discerning comparison. In order to obtain faithful simulation results, one can use electrical simulations. But, we need to characterize the probabilistic behavior of the tested algorithms. Electrical simulators are accurate but slow. A lot of simulations are performed for each algorithm

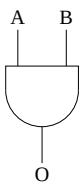
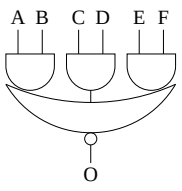
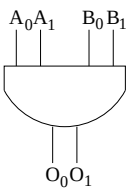
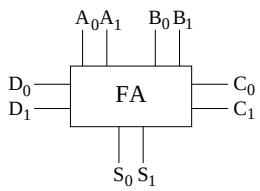
	Standard cells			Asynchronous cells	
					
Name	INV	AND2	AOI222	AND2_2R	FA_2R
Area	1.00	2.00	4.00	7.00	19.00
Delay	1.00	1.63	1.86	2.82	3.02

Figure 3: Standard and asynchronous cells samples (area and delay values are expressed using arbitrary units).

(approximately $100,000$ input patterns for each parameter set). Electrical simulators are too slow to allow such a number of patterns. So, a higher level such as the logical level seems to be necessary. Then, a gate-level description of the circuit is used. This leads to more accurate results than a RTL description. Another reason is the lack of synthesis tools for asynchronous circuits, we have to specify the circuit at the gate level.

A specific simulator was developed during this work. The simulator is based on a C++ class hierarchy that represents signals, gates, simulation and measurement functions. The circuit is described in C++ using instantiations of the dedicated objects. The characteristics of the C++ gate models come from the cell library presented above. The signals have no delay and no area. This assumption is reasonable in our case because most of the wires in the division unit are small. The whole C++ code represents approximately $13,000$ lines, including both the simulator and the code for the description of the different algorithms. The whole simulation process required more than one month of computation, and it was distributed on a few computers. With the simulator we get estimations of the computation time (average value, standard deviation and distribution), the area and the power consumption. The power consumption is estimated by the circuit activity (i.e. the count of the transition number).

5 RESULTS AND ANALYSIS

Table 3 presents the simulation results for the computation time. For each tested algorithm, two values representing the computation time are measured: its average value and its standard deviation. Those values are respectively represented by the first and the second line of each cell in Table 3. The unit used for those values is the arbitrary timing unit introduced in Section 4.1. The evolution of the average computation time with an increasing word length is illustrated by referring to Figure 4.

The average computation time results show that the fastest algorithm is a quite simple one: the radix-2 SRT algorithm. The restoring and the other non-redundant residual SRT algorithms have closed speed results. The use of a redundant number system for the residual representation does not speed up the computation. This is surprising because it is totally different from synchronous circuits knowledge. All the algorithms seem to have a linear dependency between the word length n and the average computation time. This result was predictable for SRT algorithms with a redundant representation of the residual, but it is surprising for the other ones. As the asynchronous sequential adder has an average computation time of $\mathcal{O}(\log n)$, one can predict a global $\mathcal{O}(n \log n)$ average computation time for the SRT algorithm.

Table 4 presents the estimation of the area required for the different algorithms (expressed using the area arbitrary unit). Those results are illustrated in Figure 5.

The results concerning the area are more foreseeable. The plots are linear, and the simplest solutions have the smallest area. The restoring algorithms leads to the smallest division unit. Higher radices can lead to very large solutions.

	8	12	16	24	32	48	64
restoring	100.1	158.9	217.0	330.9	443.2	665.8	887.4
	9.5	14.2	18.4	25.0	29.9	37.4	43.6
SRT2-1	97.2	145.2	192.2	284.9	376.4	558.0	738.9
	14.1	18.1	21.6	27.8	33.4	43.9	54.0
SRT4-2	134.1	193.4	250.8	363.5	475.9	698.5	920.6
	16.4	21.4	24.9	30.4	35.2	43.5	51.1
SRT8-6	114.0	181.8	226.2	333.1	416.6	628.1	816.8
	11.8	17.1	19.4	24.3	26.7	33.9	39.0
SRT16-10	122.9	174.0	223.0	316.8	409.1	591.7	773.5
	11.8	15.7	18.9	23.1	26.4	31.7	36.4
SRT2-1red	182.9	270.9	357.7	530.2	701.5	1043.5	1385.0
	13.8	18.0	21.3	27.2	32.8	43.0	52.9
SRT4-2red	187.4	258.8	328.7	467.3	604.3	878.6	1151.7
	13.4	15.7	17.8	20.8	23.5	28.4	32.4

Table 3: Simulation results for the computation time: average value (first line) and standard deviation (second line).

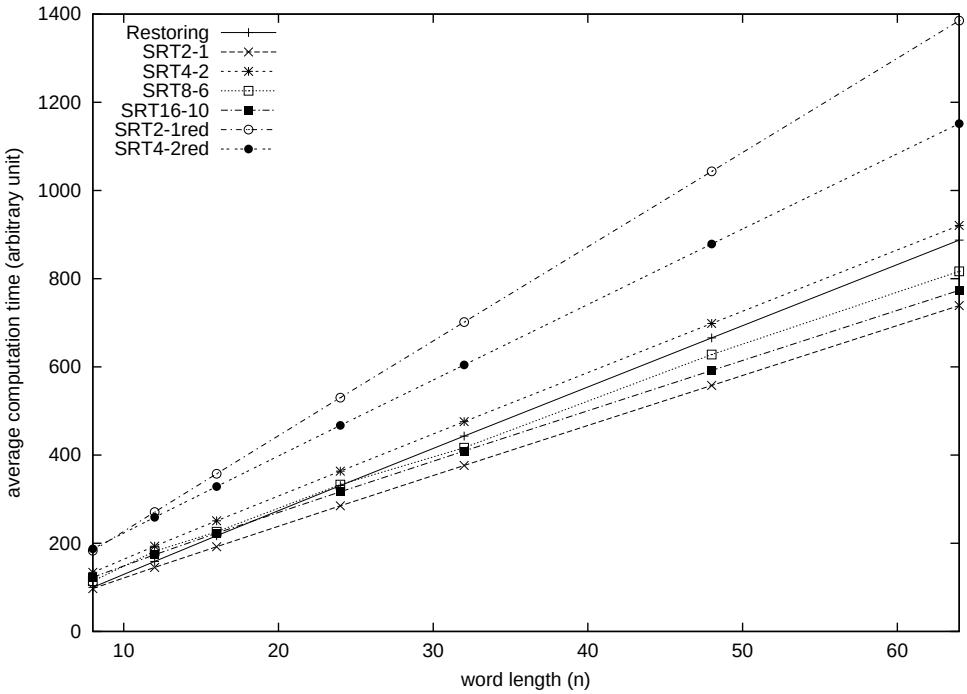


Figure 4: Average computation time evolution with the word length.

The power consumption is approximated using the circuit activity estimation described in Section 4.2. The simulator counts the number of transitions for each implemented divider. The corresponding results are presented in Table 5 and illustrated in Figure 6.

Power consumption estimation results also show that the simplest solutions are better. The restoring and low-radix SRT algorithms (without a redundant representation of the residual) present the smallest activity.

	8	12	16	24	32	48	64
restoring	525	786	1050	1580	2100	3140	4200
SRT2-1	1240	1830	2420	3620	4830	7210	9600
SRT4-2	1710	2320	2950	4180	5420	7890	10400
SRT8-6	3680	4490	5250	6820	8340	11500	14600
SRT16-10	16400	17300	18200	20100	21900	25600	29200
SRT2-1red	1870	2600	3320	4770	6230	9130	12000
SRT4-2red	24000	24700	25500	26900	28400	31400	34400

Table 4: Implemented dividers area results.

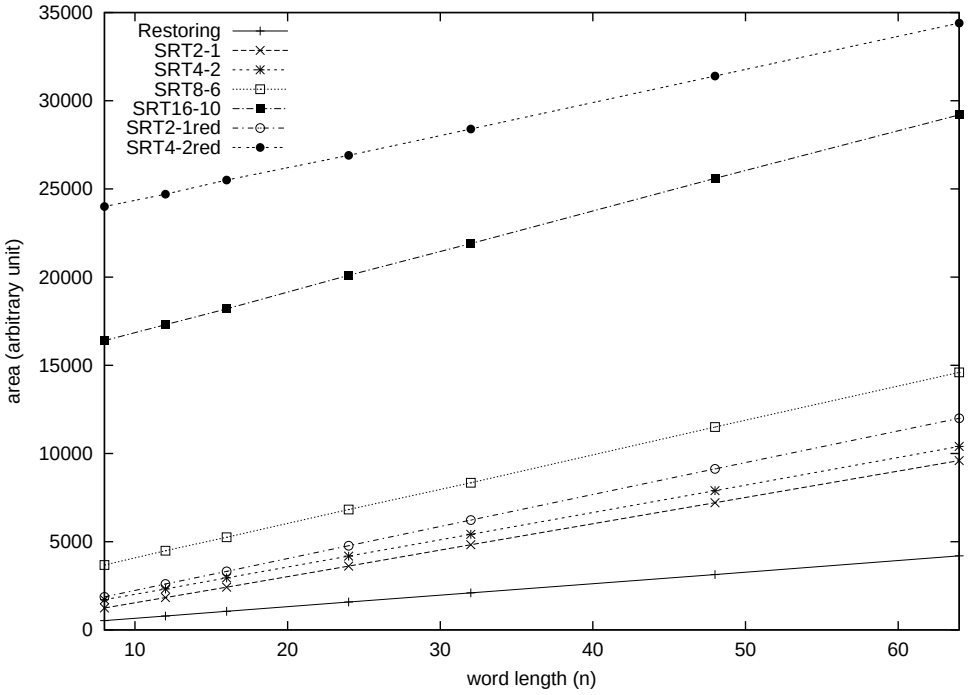


Figure 5: Dividers area evolution with the word length.

	8	12	16	24	32	48	64
restoring	2992	6592	11600	25840	45712	102352	181520
SRT2-1	5630	12086	20974	46046	80846	179630	317326
SRT4-2	7152	12636	19592	37920	62136	128232	217880
SRT8-6	11152	21200	28746	52784	76432	154544	247242
SRT16-10	50048	69218	89588	133928	183068	295748	427628
SRT2-1red	10646	20278	32790	66454	111638	236566	407574
SRT4-2red	122434	174434	228130	340610	459874	718754	1004770

Table 5: Simulation results for the circuit activity.

Self-timed circuits are also characterized by their computation time distribution. The corresponding plots show the distribution of the termination time of the operations. Figure 7 presents the computation

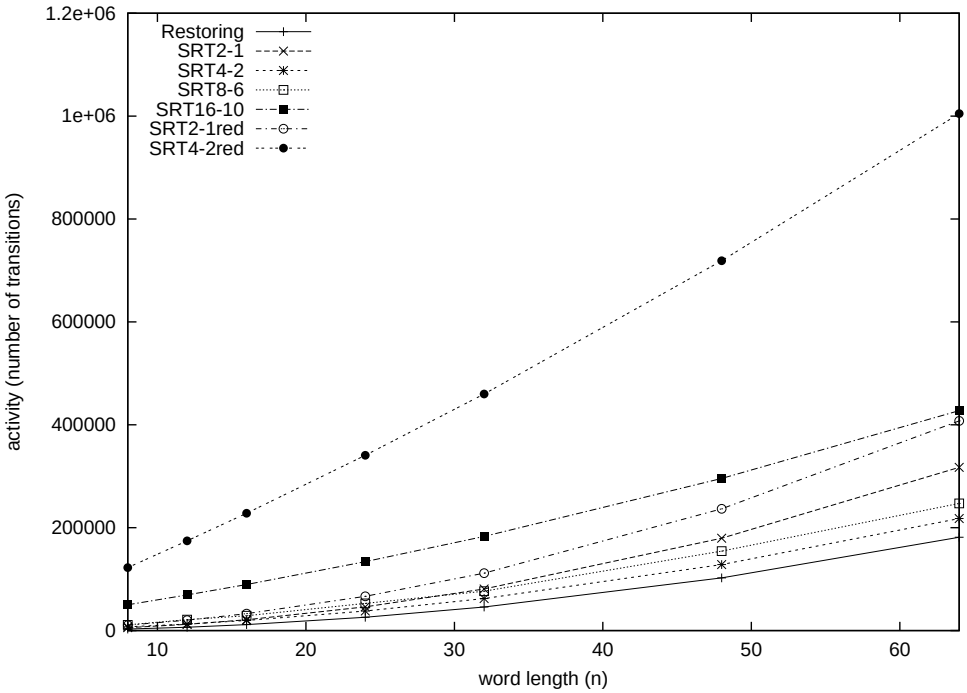


Figure 6: Dividers activity evolution with the word length.

time distribution of all the tested dividers for a 32-bit word length. Figure 8 presents the computation time distribution of the SRT4-2 divider for several word lengths.

6 CONCLUSION AND FUTURE PROSPECTS

This work shows that simple algorithms can be very efficient for asynchronous circuits. In the case of digit-recurrence division algorithms, the best solutions use low-radix values such as 2 or 4 and a straightforward representation of the residual (non-redundant number system). This result is surprising. Indeed, it is really different from the knowledge established from synchronous designs.

The restoring and radix-2 SRT algorithms seem to be good candidates for implementing dedicated division units in asynchronous processors. They allow the design of small functional units with a small latency. For low-power consumption considerations, simple solutions seem to be the best ones too. Some improvements are possible. For instance, the tables can be optimized in order to generate the most probable signals first. Multi-rail coding should be investigated for digits with more than two values ($\{-1, 0, 1\}$ for instance).

The result of this work can be extended to digit-recurrence square root algorithms. This class of algorithms is similar to the algorithms described in this paper. The specification of the digit-recurrence square root algorithms parameters is slightly more complex. Another extension is possible for algorithms evaluating elementary functions (sine, cosine, exponential, logarithms...). Many algorithms can be used in the case of synchronous circuits[19], but there is no result in the case of self-timed circuits. Variable-time shift-and-add algorithms should be very efficient for asynchronous architectures such as *self-timed rings* [20, 14].

ACKNOWLEDGMENT

The authors wish to thank F. Robin and P. Vivet from ST Microelectronics for their help and enthusiastic discussions.

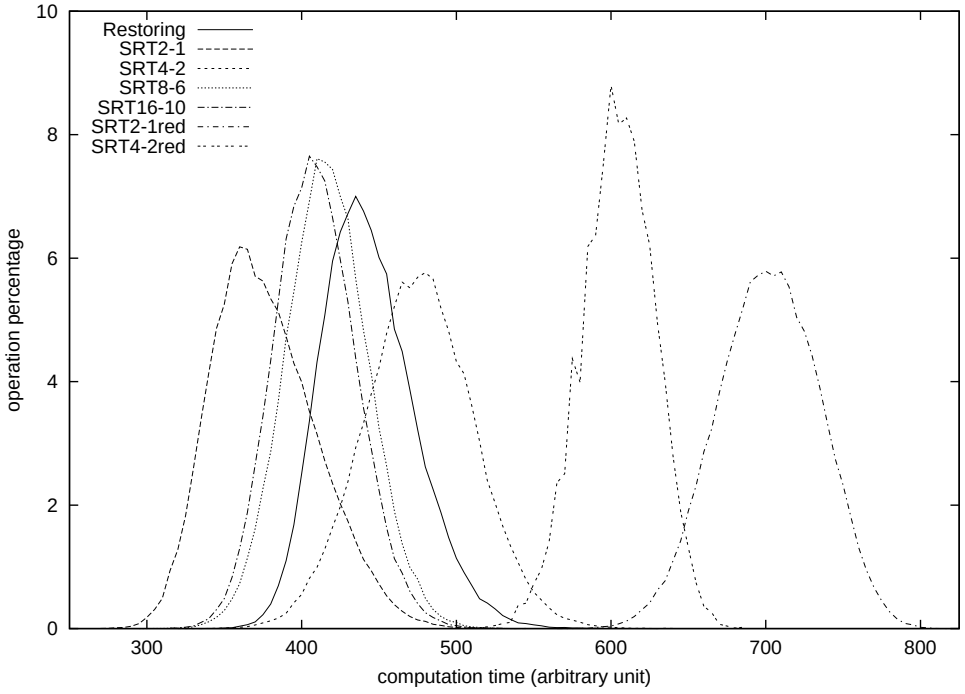


Figure 7: Computation time distribution of some dividers for n=32 bits.

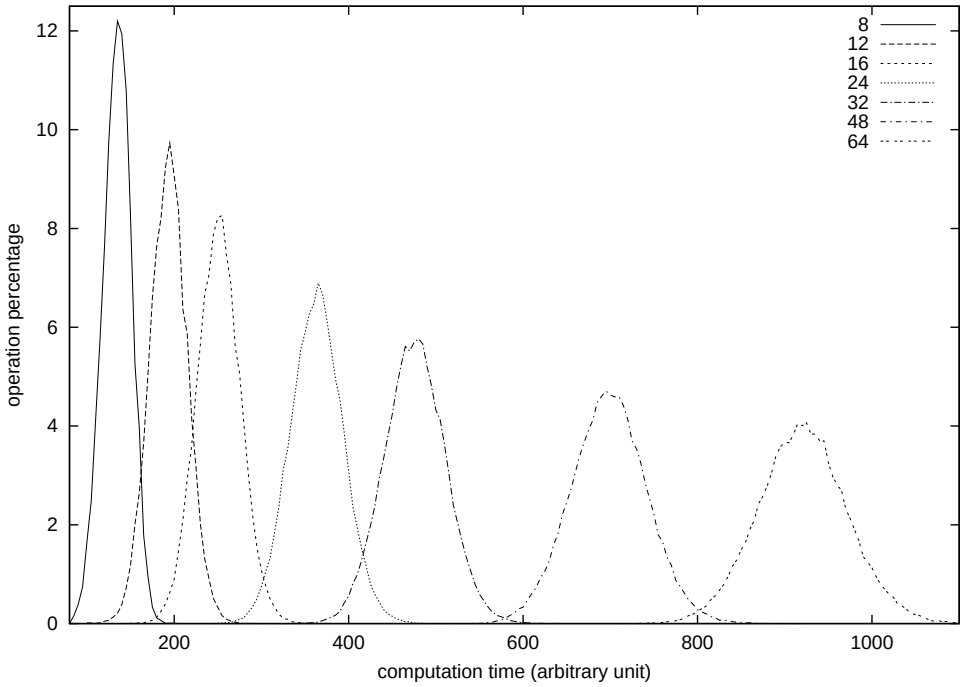


Figure 8: SRT4-2 divider computation time distribution for various word lengths.

References

[1] N. Weste and K. Eshragian, *Principles of CMOS VLSI Design*, Addison Wesley, 1993.

[2] R. Manohar and J. Tierno, "Asynchronous parallel prefix computation," *IEEE Transactions on Computers* **47**, pp. 1244–1252, Nov. 1998.

[3] F.-C. Cheng, S. Unger, and M. Theobald, "Self-timed carry-lookahead adders," *IEEE Transactions on Computers* **49**, pp. 659–672, July 2000.

[4] J.-M. Muller, A. Tisserand, and J.-M. Vincent, "Asynchronous sub-logarithmic adders," in *1997 IEEE Pacific Rim Conference on Communication, Computers and Signal Processing (PACRIM97)*, IEEE, ed., vol. 2, pp. 515–518, Aug. 1997. Victoria Canada.

[5] I. Koren, *Computer Arithmetic Algorithms*, Prentice-Hall, 1993.

[6] A. Omondi, *Computer Arithmetic Systems, Algorithms, Architecture and Implementations*, Prentice Hall International Series in Computer Science, 1994.

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

[8] M. Flynn and S. Oberman, *Advanced Computer Arithmetic Design*, Wiley-Interscience, 2001.

[9] S. Oberman and M. Flynn, "Division algorithms and implementations," *IEEE Transactions on Computers* **46**, pp. 833–854, Aug. 1997.

[10] T. Williams and M. Horowitz, "A zero-overhead self-timed 160-ns 54-b CMOS divider," *IEEE Journal of Solid-State Circuits* **26**, pp. 1651–1661, Nov. 1991.

[11] G. Matsubara, N. Ide, H. Tago, S. Suzuki, and N. Goto, "30-ns 55-b shared radix 2 division and square root using a self-timed circuit," in *Proceedings of the International Symposium on Computer Arithmetic*, S. Knowles and W. McAllister, eds., pp. 98–105, IEEE Computer Society Press, July 1995.

[12] G. Matsubara and N. Ide, "A low-power zero-overhead self-timed division and square-root unit combining a single-rail static circuit with a dual-rail dynamic circuit," in *Proceedings of the International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pp. 198–209, IEEE Computer Society Press, Apr. 1997.

[13] J.-L. Yang, C.-S. Choy, and C.-F. Chan, "A self-timed divider using a new fast and robust pipeline scheme," *IEEE Journal of Solid-State Circuits* **36**, pp. 917–923, June 2001.

[14] M. Renaudin, B. El Hassan, and A. Guyot, "A new asynchronous pipeline scheme: Application to the design of a self-timed ring divider," *IEEE Journal of Solid-State Circuits* **31**, pp. 1001–1013, July 1996.

[15] G. Cornetta and J. Cortadella, "A multi-radix approach to asynchronous division," in *Proceedings of the International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pp. 25–34, IEEE Computer Society Press, Mar. 2001.

[16] M. Ercegovac and T. Lang, *Division and Square-Root Algorithms: Digit-Recurrence Algorithms and Implementations*, Kluwer Academic, 1994.

[17] A. Martin, "The limitations to delay-insensitivity in asynchronous circuits," in *Sixth MIT Conference on Advanced Research in VLSI*, W. J. Dally, ed., pp. 263–278, MIT Press, 1990.

[18] F. de Dinechin and A. Tisserand, "Some improvements on mutlipartite tables methods," in *15th International Symposium on Computer Arithmetic ARITH15*, IEEE, (Vail, Colorado), jun 2001.

[19] J.-M. Muller, *Elementary Functions: Algorithms and Implementation*, Birkhauser, Boston, 1997.

[20] T. E. Williams, "Performance of iterative computation in self-timed rings," *Journal of VLSI Signal Processing* **7**, pp. 17–31, Feb. 1994.

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399