ON PORTABLE MACRO-CELL FPU GENERATORS FOR DIVISION AND SQUARE ROOT OPERATORS COMPLYING TO THE FULL IEEE-754 STANDARD

M. Aberbour, A. Houelle, H. Mehrez, N. Vaucher & A. Guyot*
LIP6/ASIM Laboratory, Université Pierre et Marie Curie
4, Place Jussieu, 75252 Paris Cedex 05, France
*Integrated Systems Design Group, TIMA Laboratory,
46 Av. Félix Viallet, 38031 Grenoble Cedex, France
e-mail: habib.mehrez@lip6.fr *alain.guyot@imag.fr

ABSTRACT

In this paper we investigate the design of macro-cell generators of division and square root floating-point operators. The number representation used in our operators is the IEEE-754-1985 standard for binary floating-point numbers. The design and implementation of the generators rely on a powerful multiview layout synthesis tool called GenOptim. This CAD tool is able to output a set of different descriptions for several VLSI technologies as well as FPGAs. The division and square root operators described in this paper use the signed-binary-digit representation. We start first by describing the operators for the significand, then we investigate the IEEE floating-point operators. All along this paper, and wherever appropriate, we present the implementation results using the GenOptim environment.

1. INTRODUCTION

Portability and design time are crucial issues in any system design. The procedural generator concept has proved to be of great interest since it allows the user to encapsulate his design knowledge into well-defined packages and to retrieve this information without difficulties and time penalty. So procedural generators tend to become an essential part of state-of-the-art CAD frameworks.

The generators are built up from two distinct parts. The algorithm to be realised is proper to each generator and described logically using virtual cells. The other part, called GenOptim, is common to all generators and maps the virtual cells on the target library and solves automatically the electrical and placement problems by analysing the circuit and the target cells library. Generators of addition and multiplication arithmetic operators have already been written using the GenOptim environment [1]. This methodology is now used to realise generators for division and square root operators. These operations are not so frequent in common applications, but as processor speeds increase their latency may become a penalising factor. The use of the borrow save notation and the similarity of the two operations division and square root both in fixed-point and floating-point schemes motivated us to combine them in a single operator. This combination is appealing for infrequent operation since it allows saving on area without compromising the speed.

The paper is organised as follows: section 2 describes the generator concept and the GenOptim environment. Section 3 treats the fixed-point operators and all the theory behind respectively the division, the square root and their combination. In section 4 we investigate the IEEE floating-point operators. This paper ends by presenting the implementation results and by drawing a conclusion from them.

2. METHODOLOGY

A methodology to design portable standard-cell generators has been developed [2]. This methodology is intended to ease the task of designing and implementing IEEE floating point arithmetic operators. The generator concept is supported by two main ideas. First, in the field of designing floating point arithmetic operators the designer has a large set of number formats and precision. As far as the IEEE standard [3, 12] is concerned, beyond the simple precision representation over 32 bits and the double precision representation over 64 bits, there exists a huge set of number formats in which the word lengths differ. These formats, called extended formats, increase the precision of the numbers represented. The second idea deals with the approach of floating point operators design. In fact, in order to realise these operators three main alternatives may be considered [13, 14, 15, 16]. The first is a full-custom realisation. This solution is ideal in terms of surface and propagation time, as well as other factors, but unfortunately this solution leads to huge amounts of design time and moreover the generators are not portable. The second alternative is the realisation of the operators by means of logic synthesis from the behavioral descriptions of the operators. This solution is extremely fast and portable but the complex operators may lead to an explosion of the size of the circuit. Finally we have the solution of realising the operators adopting the approach of generators using data-path or standard-cell libraries. This alternative gives the best compromise between logic synthesis and full-custom realisations.

2.1 GenOptim CAD Environment

Experience has shown that generator programs are quite often written by VLSI designers, as they hold the empirical knowledge better anyone. However, their ability does not necessarily include programming and debugging abilities: these designers have to focus on the problem at hand, not on the tools or the language they use to solve it. GenOptim has been created to quickly design efficient IEEE floating-point macro-cell generators that do not rely on particular target technologies. These macro-functions (shifters, multiplexors, priority encoders,...) are very common in the VLSI design world and may be used in many different contexts.

2.2 GenOptim : the Tool

GenOptim is a VLSI CAD tool allowing a very fast implementation of architecture ideas on different technologies and different commercial sets of CAD tools. It furnishes the designer a set of high level C functions allowing to write quickly the architecture, the test patterns and the placement. During the mapping on the wanted technology placement and timing optimisation operations are carried out by GenOptim.
Generators created with this tool can be mapped on any standard cell library (actually we have tried three different libraries: Cmos standard cell library, Cmos datapath cell library as well as FPGAs).

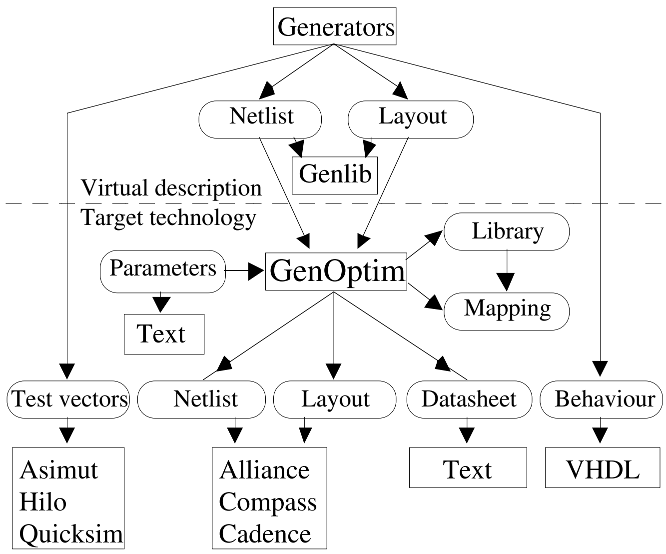


Figure 1: GenOptim description.

As depicted in Fig. 1, the circuit generation starts with a virtual description (netlist and layout of virtual cells) of the circuit. Then, GenOptim maps the virtual description to the target library and optimises the circuit.

2.3 Generators Description

From simple parameters (name of the circuit and the number of bits), the generators create automatically a netlist view, a placement view, a behavioral view, test vectors and finally a file summarising all the circuit characteristics.
The generators are built up from two distinct parts. One proper to each generator and describes logically the algorithm to be realised using virtual cells. The other is common to all generators, called GenOptim that maps the virtual cells on the target library and solves automatically the electrical and placement problems by analysing the circuit and the logic cells library used.
GenOptim allows the designer to concentrate exclusively on the algorithm to be realised. Furthermore it makes it possible to undergo modifications without any intervention on the source code of the generator. Consequently the developer of generators is never bothered by the electrical, placement and timing constraints.
We list below some of the most important descriptions furnished by the generators.

2.3.1 Placement View

Several GenOptim directives such as row compression or slice reconfiguration modify the initial placement in order to insert the generator in any specific topological context and to obtain homogenuous physical structures.

2.3.2 Netlist View

In order to enhance portability, GenOptim uses multiple drivers and is able to generate netlist and layout views for various CAD systems and technologies.
One of the essential motivations behind the generator approach is the portability. So the generators use a library of virtual cells. A virtual cell is defined as a cell having a unique behavior but having many possible constructions. This allows the mapping of the generators over many different technologies and libraries.

2.3.3 The Datasheet View

It is a report of all timings and critical paths of the generated circuit. It is very useful when designing a generator. The user can easily and quickly see where are the critical paths and the evolution in performance/surface following the activated optimisations.

3. FIXED POINT OPERATORS

3.1 Division

The table below shows the notations and variables used along this part: ⊕, ∨, • denote XOR, OR and AND operators ; +, –, ∗, ÷ denote add, subtract, multiply and divide. i, j, k, n represent integers.

Number	Digit	
A	a_j	dividend or radicand
D	d_j	divisor
Q	p_j	quotient or square root
"	q_j	same Q in redundant notation
$R^{(i)}$	r_j	i^{th} partial remainder
$R^{(i+1)}$	s_j	next partial remainder
$\hat{R}^{(i)}$	$r_2\ r_1\ r_0$	a 3-digit coarse estimate of $R^{(i)}$
	c_j	borrow (negative carry to subtract)
	g_j	borrow generate
	t_j	borrow transmit
	$\mathbf{G}_i^j$	group generate
	$\mathbf{T}_i^j$	group transmit

In common applications the division operation is considerably slower than addition or multiplication operations [4]. There are two approaches for the hardware implementation of division $Q = A \div D$. The Newton-Raphson method uses a series of multiplications and additions to develop an increasingly accurate approximation of the desired quotient Q. Its main interest is the use of readily available adders and multipliers [4, 5, 6, 7, 8, 9, 10, 18, 19]. The digit-recurrence approach relies on subtraction and multiplication by the radix "b" and by the quotient digits "q_j".

Step 0

$$R^{(0)} = A - D; \quad Q^{(0)} = 1$$

Step $j \le 0 \;\; j \le n-1$

$$R^{(j+1)} = R^{(j)} - q_{j+1} * D * b^{-j-1}$$

$$Q^{(j+1)} = Q^{(j)} + q_{j+1} * b^{-j-1}$$

3.1.1 Cellular Array Division: Borrow Save Division

In the borrow save radix 2 notation, each signed digit $q_j \in \{-1,0,+1\}$ is the difference of 2 bits $(q_j^- - q_j^+)$. This redundant notation [5] can be used for quotient digits selection through the test of only 3 digits of the partial remainder [6,7] and for carry-propagation-free addition/subtraction as well. But the quotient has to be translated from the redundant notation into a standard one.

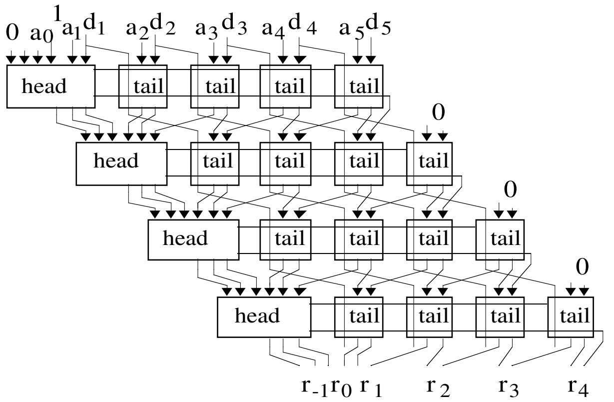


Figure 2. A borrow save divider array

A combinatorial divider is a stack of n identical slices, each one formed out of a head cell and a linear array of n tail cells [10] as shown on Fig. 2.

3.1.2 Slice of the Divider

To simplify the reading of Fig. 2, and since we are interested in the jth slice only, let us put $R = R^{(j)} * 2^j$ and $S = R^{(j+1)} * 2^j$.

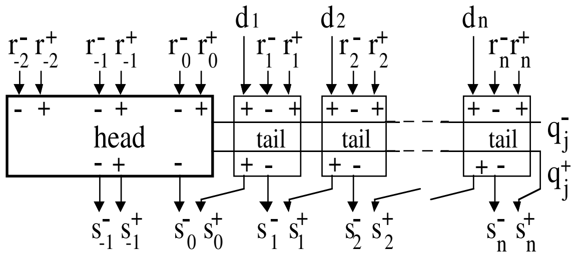


Figure 3. Slice of a radix-2 borrow-save divider

The role of the head cells (Fig. 3) is to :

1. select q_j and the operation to execute
2. execute this operation on the head digits

The role of the tail cells (Fig. 3) is to execute the operation dictated by the head cells on the tail digits.

3.1.3 Equations of Head and Tail Cells

An improvement over paper [8] is to parallelise the quotient digit selection and the processing of the remainder three most significant digits [9].

Since $D = 1 + \sum_{j=1}^{n} d_j * 2^{-j}$, $-D = -2 + \sum_{j=1}^{n} \overline{d_j} * 2^{-j} + 2^{-n}$.
So the iteration $S = R - q_j * D$ can be written:

$$S = R + q_j^- * (1 + \sum_{j=1}^{n} d_j * 2^{-j}) + q_j^+ * (-2 + \sum_{j=1}^{n} \overline{d_j} * 2^{-j} + 2^{-n})$$

The head cell preserves the identity:

$$\hat{S} = \hat{R} + q_j^- + q_j^+ * (-2)$$

and the tail cells preserve the identity :

$$(2 * s_{i-1}^+ - s_i^-) = (r_i^+ - r_i^- + q_j^- * d_i + q_j^+ * \overline{d_i}).$$

The bit s_n^+ weighting $(+2^{-n})$ is connected to q_j^+.

3.1.4 Overlapped Conversion

The quotient

$$Q_{(n)} = \sum_{j=1}^{n} q_j^+ * 2^{-j} - \sum_{j=1}^{n} q_j^- * 2^{-j}.$$

In an n-bit binary subtractor, let us note $\mathbf{T}_i^j$ the group transmit and $\mathbf{G}_i^j$ the group generate, with $0 \ge i \ge j \ge n$.
We note c_i the borrow.
$\mathbf{T}_i^j$ means that the borrow propagates from position j up to i, that is c_{i+1} is equal to c_j.
$\mathbf{G}_i^j$ means that a borrow is generated somewhere between positions j and i and propagated from this location up to position i and thus $c_{i+1} = 1$.

We have

$$\mathbf{T}_i^i = t_i = \overline{q_i^+ \vee \overline{q_i^-}}, \quad \mathbf{G}_i^i = g_i = \overline{q_i^-} \text{ and } c_{i+1} = \mathbf{G}_i^n.$$

The n bits of the converted quotient are given by

$$p_i = c_i \oplus t_i.$$

For any k such that $0 \ge i \ge k \ge j \ge n$, $(\mathbf{T}_i^j, \mathbf{G}_i^j)$ can be computed from $(\mathbf{T}_i^k, \mathbf{G}_i^k)$ and $(\mathbf{T}_{k+1}^j, \mathbf{G}_{k+1}^j)$ as follows:

$$(\mathbf{T}_i^j, \mathbf{G}_i^j) = (\mathbf{T}_i^k \,.\, \mathbf{T}_{k-1}^j, \mathbf{G}_i^k \vee \mathbf{T}_i^k \,.\, \mathbf{G}_{k-1}^j).$$

We note Δ the operator introduced by Brent and Kung [11] :

$$(\mathbf{T}_i^j, \mathbf{G}_i^j) = (\mathbf{T}_i^k, \mathbf{G}_i^k)\, \Delta\, (\mathbf{T}_{k+1}^j, \mathbf{G}_{k+1}^j).$$

In Fig. 4 the icon for the Δ-cells is represented as .
Each wire carries two bits (**T**,**G**). It is easy to check that Δ is associative so to give the same result the Δ-cells can be connected in many different ways.

3.1.5 Speed-Area Trade-Off of the Converter

The quotient is produced digit by digit, starting from the most significant one that has to be transformed into standard notation by a subtraction.
Fig. 4 show two ways of organizing the Δ-cells for the generation of carries in a 10-bit subtraction. We want the result digits all at the same time, so the slice number gives the time (expressed in Δ-cell delay) when the inputs must be ready.

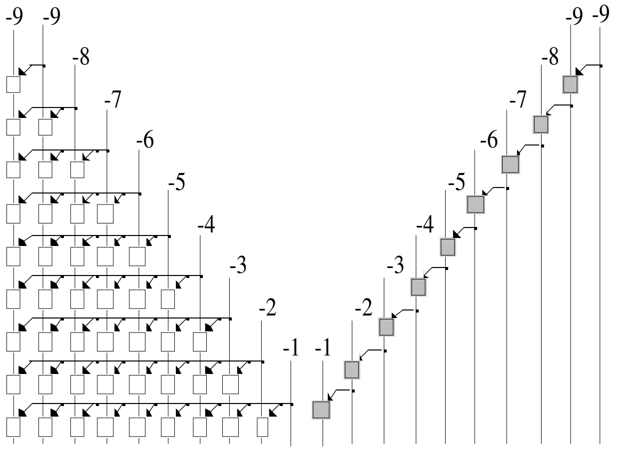


Figure 4. Converter's architecture

The subtractor of Fig. 4 (left), with a complexity of $\frac{1}{2}(n-1)^2$ Δ-cells seems well suited when the most significant digits are the first to be ready but is indeed very fast (but costly). Let us call σ the ratio between the delay of the Δ-cell and the pace at which the quotient digits are computed by the head. If σ>1, the two converters of Fig. 4 can be mixed. Fig. 5 gives different shapes of the converter according to σ.

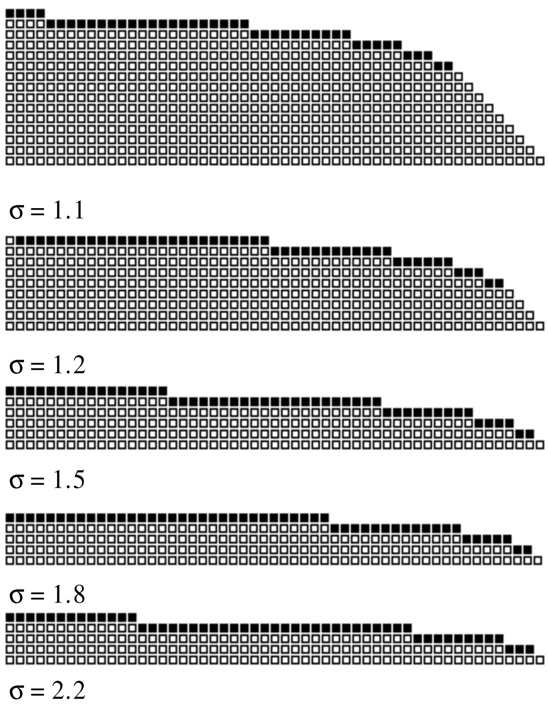


Figure 5. Area optimized converters

3.2 Square Root

The square root operation principle does not differ too much from the principle of division[4, 17, 18]. It only supposes that the divisor is identical to the quotient : Q = A ÷ Q. Like the division we have two classes of algorithms: The Newton-Raphson that uses a series of multiplications and additions and the digit-recurrence method.

3.2.1 Cellular Array Square Root

The digit-recurrence square root approach relies on subtraction and multiplication by the radix b and by the quotient digits q_j .

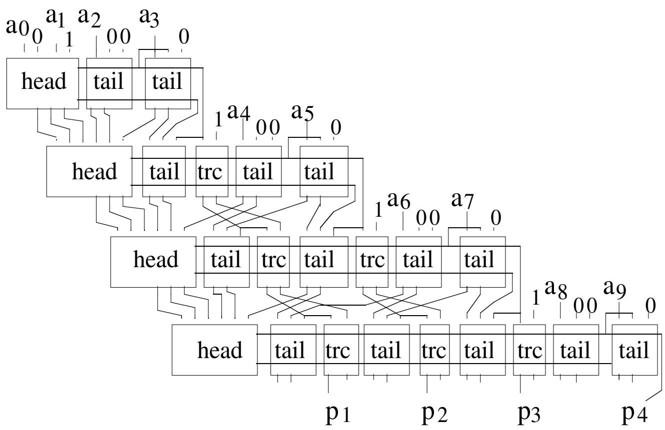


Figure 6. Borrow save square root architecture

Step 0

$$R^{(0)} = A - 1;\ Q^{(0)} = 1$$

Step j $0 \le j \le n-1$

$$R^{(j+1)} = R^{(j)} - 2 * q_{j+1} * Q^{(j)} * b^{-j-1} - q_{j+1}^2 * b^{-2j-2}$$

$$Q^{(j+1)} = Q^{(j)} + q_{j+1} * b^{-j-1}$$

It is easy to verify that $A = \left(Q^{(j+1)}\right)^2 + R^{(j+1)}$ since $\left(Q^{(j+1)}\right)^2 = \left(Q^{(j)}\right)^2 + 2 * q_{j+1} * Q^{(j)} * b^{-j-1} + q_{j+1}^2 * b^{-2j-2}$

A borrow save square root is composed of an array of carry-propagation-free adders/subtractors, and a conversion circuit that transforms the quotient from a borrow save notation to a standard binary representation.
In radix 2 the iteration becomes:

$$R^{(j+1)} = R^{(j)} - 2 * q_{j+1} * Q^{(j)} * 2^{-j-1} - (q_{j+1})^2 * 2^{-2j-2}$$

Note that in borrow-save we have:

$$(q_{j+1})^2 = q^+_{i+1} \vee q^-_{i+1} = \overline{t_{i+1}} = 1 - t_{i+1} .$$

The main skeleton of a borrow save square root operator is the same as the one of the non restoring operator, the difference resides in the fact that in the first one no propagation exists and the next operation is determined by examining the MSB's of the partial sums (Fig. 6).

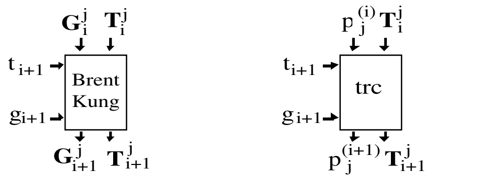
3.2.2 The TRC Cell

The TRC cell carries out the construction of Q:

$$Q^{(j+1)} = Q^{(j)} + q_{j+1} * 2^{-j-1}$$

Assuming that $Q^{(j)} = \sum_{i=0}^{j} p_i * 2^{-i}$ at step j then

$$Q^{(j+1)} = \sum_{i=0}^{j} (p_i \oplus \mathbf{T}_i^j \cdot g_{j+1}) * 2^{-i} + \overline{t_{j+1}} * 2^{-j-1}$$



$$\mathbf{T}_{i+1}^j = \mathbf{T}_i^j \cdot t_{i+1} \qquad \mathbf{T}_{i+1}^j = \mathbf{T}_i^j \cdot t_{i+1}$$

$$\mathbf{G}_{i+1}^j = \mathbf{G}_i^j \vee \mathbf{T}_i^j \cdot g_{i+1} \qquad p_j^{(i+1)} = p_j^{(i)} \oplus \mathbf{T}_i^j \cdot g_{i+1}$$

Figure 7. The Δ-cell and TRC cell equations

The square-root converter has the same structure as in Fig. 4, the cells being slightly different. This converter can not be optimised as proposed in section 3.1.5 since for the square root not only the final result $Q^{(n)}$ is needed, but the intermediate results $Q^{(i)}$ are used as well.

3.2.3 Implementation Results

A fixed point square root operator has been implemented using the GenOptim tool. Shown in Fig. 16 and Fig. 17 are two versions of the layout of the same size fixed point operators. In the first one no optimisation has been activated. We can notice the odd triangular shape of the layout, this is due to the disposition of the cells building up the complete operator. In the second schematic a nicer version of the layout is shown. In fact, placement optimisations have been activated when generating the operator. The optimisation shows to be of great interest since it allows the compacting of the circuit.

3.3 Square Root and Division Combined

Based on the architectures of borrow save division and square root, we imagined a combined operator for both of them. The circuit comprises a regular array of cells, fundamentally identical to the cells of separate operators thus only a few multiplexors are introduced.
The similarities of the recurrence for division and square root are apparent so that their combination turns to be an easy task [17]. We judge that this combination is of considerable value. Considering the frequencies of occurrence of a division or a square root and the probability of soliciting the system to perform the two operations at the same time, we are convinced that the combination of the two operators will be a contribution of value. The combination will lower down the silicon surface, thus prize, and enhance the speed performance of the system.

4. IEEE FLOATING-POINT OPERATORS

The IEEE floating-point operators must be created with generators because the precision (the width of the exponent-significand in bits) is not. The precision will be chosen following the needs of the application.
According to Fig. 8, an IEEE operator is performed in four stages, which are the prenormalisation stage, the operation stage, the renormalisation stage, and the exception computation stage.
The prenormalisation stage consists of the effective preparation of operands A and B before the operation is executed. The renormalisation stage takes the result of the operation and codes it in the IEEE standard format. This stage takes into account the rounding mode. At last, during the exception handling stage, some flags may be set whenever an exception condition has been detected.

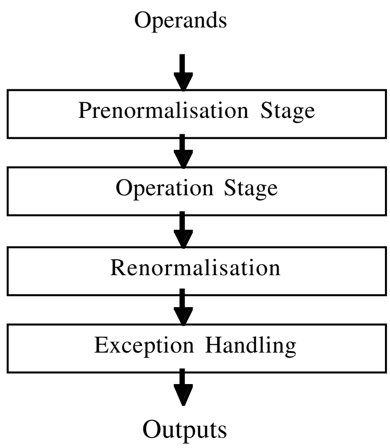


Figure 8. Main stages of an IEEE operator.

4.1 IEEE Division

4.1.1 Architecture

An IEEE division is more complex than a fixed point division, since it includes operator prenormalisation and renormalisation, exponent calculation and exception handling.

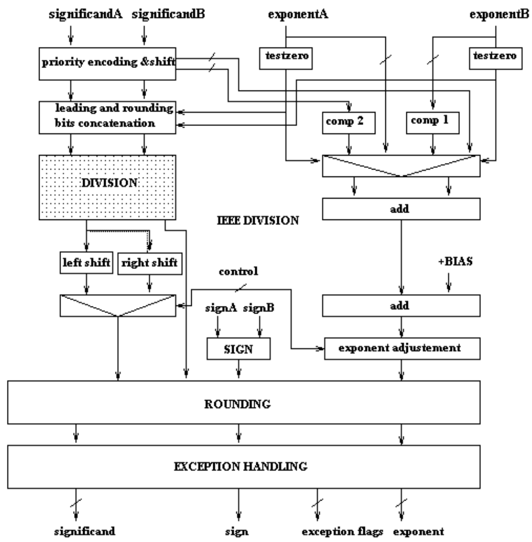


Figure 9. IEEE floating point divider architecture

The prenormalisation stage involves the alignment of the significands before the division operation is carried out. The exponent stage computes the non normalised result exponent and delivers some other signals used by the other stages. The normalization operator adjusts the exponent and significand according to the rounding mode to fit the IEEE standard. At last the exception handling stage sets some flags whenever a null value, an invalid operation or operand, and overflow or underflow conditions have been met. Fig. 9.

4.1.2 Practical Results

Many floating point operators have been generated and validated, we present below some implementation results related to a simple precision (8:23 bits) IEEE divider (Fig. 10). All the results presented in this paper are for a 1µm technology.

operation	transistors nb.	size (μm^2)	propagation delay
8:23 divider	50000	9888 x 5598	300 ns

Figure 10. Characteristics of a simple precision IEEE divider

4.2 IEEE Square Root

4.2.1 Architecture

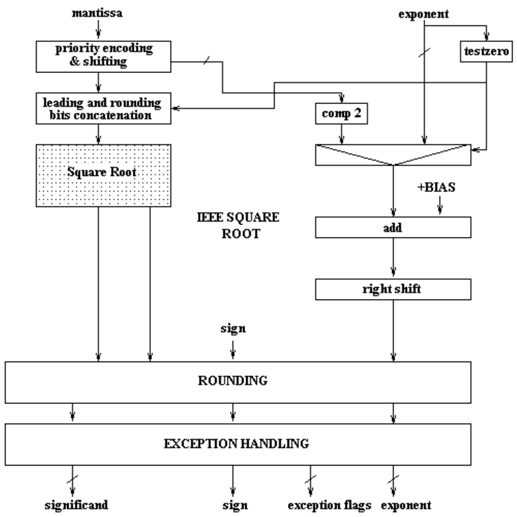


Figure 11. Architecture of an IEEE floating point square root operator.

An IEEE square root is more complex than a fixed point square root, since it includes operator prenormalisation and renormalisation, exponent calculation and exception handling. The prenormalisation stage involves the alignment of the significand before the square root operation is carried out. The exponent stage computes the non normalised result

4.2.2 Practical Results

A simple precision IEEE floating-point operator has been generated and validated. The table below summarises some of its characteristics. (Fig. 12)

operation	transistors nb.	size (μm^2)	propagation delay
8:23 square root	40000	11328x4684	290 ns

Figure 12. Characteristics of a simple precision square root IEEE operator

4.3 IEEE Square Root and Division

4.3.1 Architecture

The realization of the separate operators for division and square root encouraged us to consider the design of a combined IEEE floating point operator. This is of a course a harder challenge. A careful study of the separate operators shows that the same material is "reused" for both operators. A very few multiplexors (taking a very small space) have been introduced in the combined operator.

We think that there is no need to lay down every single detail of the overall architecture of the IEEE combined operator. The main idea that must be kept in mind is that an IEEE operator skeleton is the same for any operation. we will always encounter the now traditional stages of the IEEE operators, i.e., prenormalization, exponent, normalization and exception handling stages. (Fig. 13)

The signal *divrac* selects the operation to be carried out (1 for division and 0 for square root).

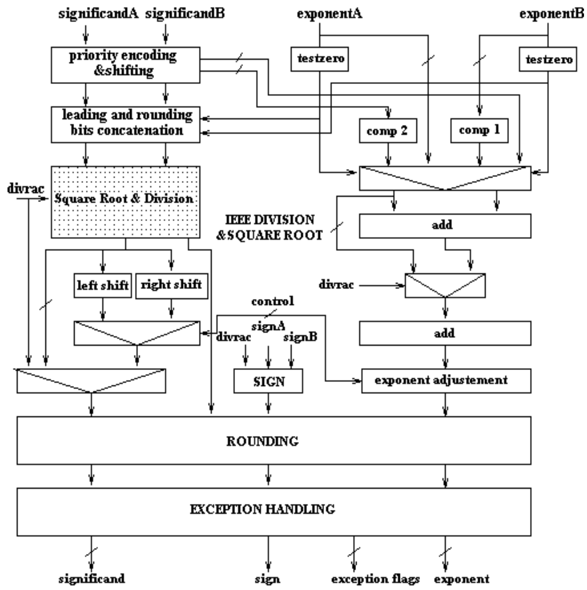


Figure 13. Architecture of an IEEE floating point combined operator for division and square root.

4.3.2 Practical Results

A simple precision IEEE floating-point operator has been generated and validated. The table below summarizes some of its characteristics. (Fig. 14)

operation	transistors nb.	size (μm^2)	propagation delay
8:23 divider & square root	60000	14000x5000	340 ns

Figure 14. Characteristics of a simple precision combined operator for division and square root

A layout of the IEEE floating point operator for combined division and square root is depicted in Fig. 18.

5. CONCLUSION

This paper has been considering the design and implementation of portable standard-cell generators of arithmetic operators satisfying the IEEE 754 standard for binary floating-point arithmetic under the GenOptim environment.

The GenOptim VLSI CAD tool is of great value as it allows the rapid and efficient design of macro-functions that can be reused in larger designs. We believe that in light of production of Intellectual Property (IP) cores, this tool will be of great interest.

The motivation behind the realisation of the combined operator are the similarities of the division and square root operations and the borrow save notation used in both of the cellular arrays to implement the two operators.

The combined operator offers the advantages of speeding up of the overall system and considerable gain in surface, thus cost. The table below shows some results of the floating point IEEE arithmetic operators implemented using the GenOptim CAD environment. Fig. 15.

operator	transistors	size	delay
8:23 divider	50000	9888x5598	300 ns
8:23 square root	40000	11328x4684	290 ns
8:23 square root & division	60000	14000x5000	340 ns

Figure 15. Implementation results

Our contributions in the field of computer arithmetic and VLSI design and architectures are the GenOptim tool and the implementation of up to now theoretical algorithms and the combination of the two operations of division and square root in a single operator, both in fixed-point and floating-point schemes.

REFERENCES

[1] Houelle A., Mehrez H., Vaucher N. "*On portable Macro-cell generators using the fully 754-IEEE standard* ", proc. ICSPAT 94, Dallas Texas USA, Oct. 18-21, 1994, V2 pp1749-1754.

[2] Houelle A., Mehrez H., Vaucher N., *Méhodologie de conception de générateurs portables de macro-blocs fonctionnels optimisés en surface et en performance*, MASI report, December 1993.

[3] ANSI/IEEE,*IEEE Standard For Binary Floating-Point Arithmetic*, ANSI/IEEE transaction, Std 354-1985.

[4] Cavanagh J.J.F., *Digital Computer Arithmetic, Design and Implementation*, McGraw-Hill, computer science series,1984.

[5] Avizienis A., *Signed number representation for fast parralel arithmetic* , IRE Transaction on Electronic Computer vol. EC-10, September 1961.

[6] Robertson J.E., *The correspondance between methods of digital division and multiplier recoding procedures*, IEEE Transactions on Computers, vol. C-19 Nb. 8, August 1970.

[7] Tocher K.D., *Techniques of multiplication and division for automatic binary computers*, Quart. Journ. Mech. Appl. Math. Vol. 11 Nb. 3 1958.

[8] Vandemeulebroecke A., Vanzieleghem E., & al., *A new carry free division algorithm and application to a single chip RSA processor*, proc. ESSCIRC'89, Vienna, Austria, 1989.

[9] William J., Hamacher V.C.,"*A linear-time divider array* ", Canadian Elec. Eng. Jour. Vol. 6, Nb. 4, 1981.

[10] Moussa I., Guyot A., Rost. P.,"*Design and comparison of GaAs and Cmos redundant dividers* ", proc. ESSCIRC'93, Sevilla, Spain, 1993.

[11] Brent R., Kung, H. T., *A Regular Layout for Parrallel Adders,* IEEE Transactions on Computers, vol C-31, 1982.

[12] D. Goldberg, "*What Every Computer Scientist Should Know About Floating-Point Arithmetic* " ACM Computing Surveys, Vol. 23, No. 1, March 1991.

[13] D. Johansen, "*Bristle blocks: a silicon compiler* " proc 16th ACM IEEE DAC, San Diego, June 1979.

[14] A. Greiner, F. Pêcheux "*ALLIANCE: A complete Set of CAD Tools for teaching VLSI Design* ", proc. Third Eurochip Workshop, p 230-237, 1992.

[15] "*The VHDL Language Reference Manual* ", Intermetrics, Inc., version 7.2, Aug. 1985.

[16] A. Compan, V. Delorme, J-A. François, H. Mehrez, F. Pêcheux "*F-Risc: A 32 bits floating-point unit integrated in a Risc processor for embedded system* " proc. ICM92, Monastir Dec 19-21 1992 pp 6.1.2.1-4.

[17] McQuillan S.E., McCanny J.V.,"*New algorithms and VLSI Architectures for SRT division and Square Root* ", proc. IEEE symp. on Computer Arithmetic, 1993, pp80-86.

[18] Muller J-M., " *Arithmétique des ordinateurs, opérations et fonctions élémentaires* ", Masson, 1989.

[19] Guyot A., Montalvo L., Houelle A., Mehrez H., Vaucher N.,"*Comparaison of the layout synthesis of radix-2 and pseudo-radix-4 dividers* ", proc. 8Th IEEE International Conference on VLSI design, New Delhi, India, January 1995.

FIGURES

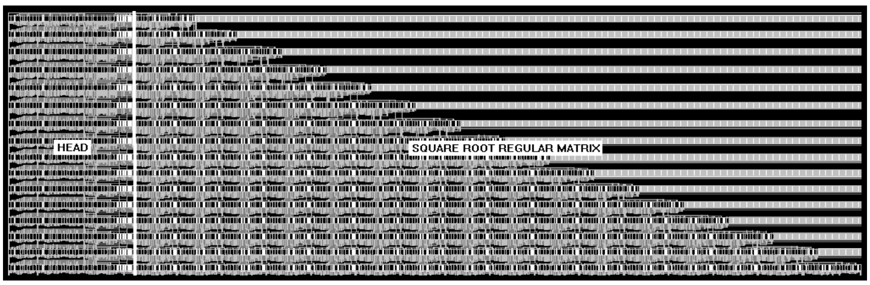


Figure 16. A fixed point regular array square root operator without placement optimisation.

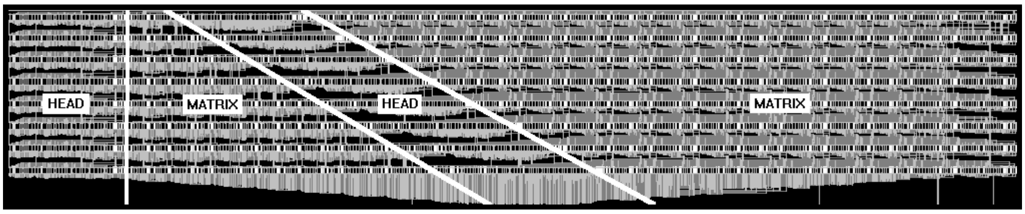


Figure 17. A GenOptim optimised fixed point square root operator.

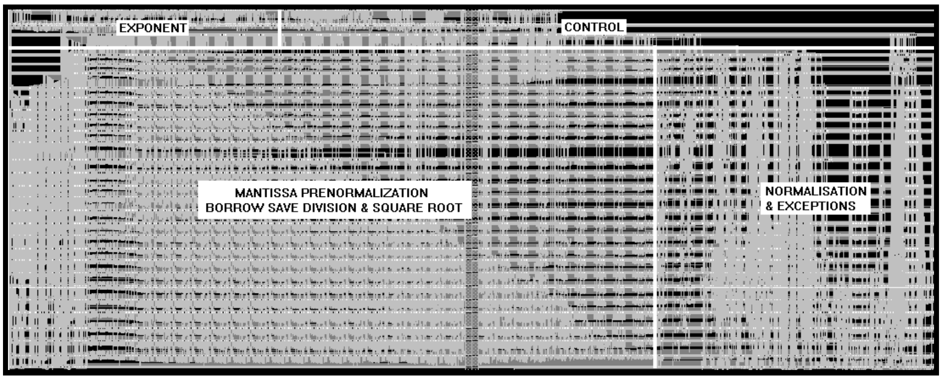


Figure 18. Layout of a simple precision combined operator for division and square root.

BIOGRAPHY

Mourad Aberbour is a Ph.D. student at the LIP6 laboratory in the CAD-VLSI team of the Pierre and Marie Curie University of Paris. His research interests concern the mapping of Computer Vision algorithms onto VLSI architectures and neural network architectures and their hardware implementations.

Alain Houelle is a computer science researcher at the LIP6 Laboratory of the Pierre and Marie Curie University of Paris. He obtained the Ph.D. degree from this university in 1997. His research work deals with the development of GenOptim: an environment for the design of portable VLSI blocks optimised in both performance and area.

Habib Mehrez is currently a computer science researcher and teacher at the Pierre and Marie Curie University of Paris. He obtained the "Doctorat d'Etat" degree from the same university in 1991. His research concerns mainly the elaboration of a unified approach of FFT algorithms and their efficient VLSI implementation. He is leading the computer arithmetic and pattern recognition VLSI architectures groups within the CAD&VLSI team of the LIP6 Laboratory.

Nicolas Vaucher is a computer science researcher at the LIP6 Laboratory of the Pierre and Marie Curie University of Paris. He obtained the Ph.D. degree from this university in 1997. He is mainly interested in the development of design methodologies for the VLSI generic architectures targeted to signal processing.

Alain Guyot received his MS, Ph.D. and Habilitation from INPG (Institut National Polytechnique) in Grenoble, France. He his currently an associate professor within the same university where he teaches computer architecture, computer arithmetic and VLSI design. He is responsible of the "Integrated System Design" research group in the TIMA Laboratory. Prof. Guyot is the author or co-author of over 100 papers in conference proceedings or journals.