A Fully Parametrized IEEE Floating-Point Operators Library For Use In Digital Signal Processing

M. Aberbour, S. Gounaud, A. Houelle, H. Mehrez, N. Vaucher

MASI/CAO-VLSI laboratory, Université Pierre et Marie Curie
4, Place Jussieu, 75252 Paris Cedex 05, France
e-mail: mehrez@masi.ibp.fr

Abstract

In this paper, the design of various generators of floating point operators is discussed. These operators are designed to fulfill the need for high-performance VLSI digital signal processors. The number representation is the IEEE standard for binary floating point numbers. The generators are built up as a hierarchical composition of parametrized macro-blocks, so as to allow a great flexibility in the design process and in the field usage of the generators. We target by our designs the basic and common operations of addition, multiplication, division and square root. We will discuss many architectural styles ranging from parallel-parallel to full-serial implementations.

Introduction

The design of integrated circuits for signal processing needs a certain number of arithmetic operators such as adders, multipliers, dividers, etc. Since all these operations are carried out on variable word length data, the concept of generators has been adopted. The generators, even optimized in surface, can lead to a considerable increase in the overall surface of the circuit. A cure for this would be to select an architecture that minimizes the surface. This gain in surface usually penalizes the time performance. We investigated many types of architectures to get the best compromise between purely parallel operators (best time performance but worst surface) and purely series operators (worst time performance but best surface).
The number representation used is the IEEE standard for binary floating point numbers. According to this standard, there exist many number formats, four different rounding modes, and a big set of exception signals to be provided at the end of the operation. A complete implementation of these specifications leads to a considerable Silicon surface. A mathematical coprocessor implements all these specifications. But in digital signal processing applications, we know already the ranges and types of our data, so not all the formats nor the rounding modes nor the exceptions are needed at the same time. For example, the denormalized format is used to represent very small quantities and in some digital image processing computations there is no need for this format since the data are never too small. This is to say that the generators should be able to choose a format type, a particular rounding mode and even a subset of the exceptions to be computed without any loss in accuracy. It should be noted that the generators are able to generate operators complying to the full IEEE standard.
The generators are built on a powerful and multiview design environment which provides, a hierarchical and parametrized netlist view based on a virtual technology independent standard cells library, a layout and a VHDL behavior view, and finally a test view which includes both functional and test patterns with a very high fault coverage.
The paper will be composed of several sections describing the algorithms, the architectures and tools used to design the generators of the IEEE floating point operators. The first section presents the IEEE standard for floating point numbers. We then describe the design environment and the CAD tools. In the third section we investigate some example architectures of operators. Finally we draw up a conclusion and we present some implementation results.

1. The IEEE floating point standard

Several different representations of real numbers have been proposed, but by far the most widely used is the floating-point representation. An IEEE binary floating-point number [2,3] is a bit-string characterized by three components: a sign, a signed exponent (or biased exponent), and a significand.

```
|__| |_______|(leading bit)|_______________|
 S      E                          M
```

Various formats are used in the IEEE standard:

1.1. Normalised format

$IEEE=(-1)^S * (1.M) * 2^{-127}$

The floating-point representation is not necessarily unique. For example, both $0.01*10^1$ and $1.00*10^{-1}$ represent 0.1. If the leading digit of the significand is nonzero the representation is said to be normalized.

Requiring that a floating-point representation be normalized makes the representation unique.

1.2. Denormalised format

$IEEE=(-1)^S * (0.M) * 2^{E-126}$

This representation is used to code smaller numbers that the normalized format cannot represent. The leading digit of the significand is always zero. This format seems to be useless when considering such applications where data are certainly not smaller than the smallest normalized number.

1.3. Precision

The IEEE standard defines four different precisions: single, double, single extended and double extended. In 754, single and double precisions correspond roughly to what most floating-point hardware provides. Single precision occupies a single 32 bit word, double precision two consecutive 32 bit words. Extended precision is a format that offers just a little extra precision and exponent range (figure 1). The IEEE standard only specifies a lower bound on how many extra bits extended precision provides. Thus, the precision can be customized for specific applications.

Precision	exponent width	significand width	exponent bias
single	8	23	127
double	11	52	1023
ext. single	11	32	unspecified
ext. double	15	63	unspecified

Figure 1: IEEE standard precision

1.4. IEEE floating-point operator

The IEEE floating-point operators must be created with generators because the precision (the width of the exponent-significand in bits) is not fixed (as shown in figure 1, mainly the extended format). The precision will be chosen following the needs of the application.
According to figure 2, 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.

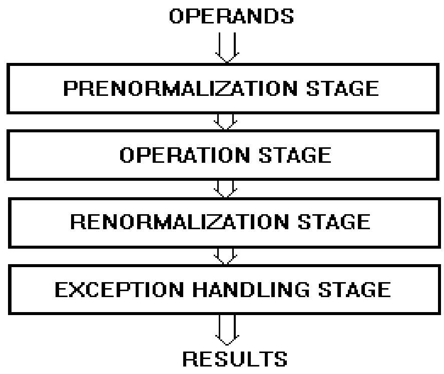

Figure 2 : Main stages of an IEEE operator.

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 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 realize these operators three main alternatives may be considered. The first is a full-custom realization. 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 realization 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 realizing 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 realizations.
The generators are based around a concept of virtual cells making them portable independently from the target technology.

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 set of C functions which allow the user to describe the netlist, the layout, the test vectors and the behavioral views as shown in Figure 3.

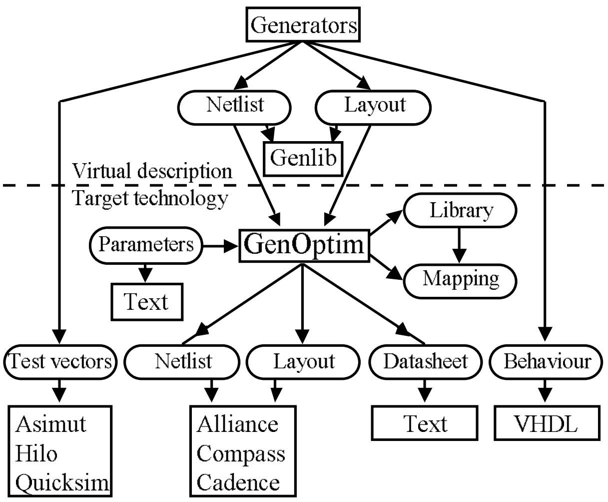

Figure 3: GenOptim description.

Generators created with GenOptim can be mapped on any standard cell library (actually we have tried three different libraries: Cmos standard cell library, Cmos datapath cell library and GaAs standard cell library).
As depicted in Fig. 3, 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 optimizes 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 summarizing all the circuit characteristics. The generators are written in the C high level language. This language has been chosen for its large portability and diffusion.
The generators are built up from two distinct parts. One proper to each generator and describes logically the algorithm to be realized 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 analyzing the circuit and the logic cells library used.
GenOptim allows the designer to concentrate exclusively on the algorithm to be realized. 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.

3. Fixed point operators architectures

In the block diagram of the IEEE operation, the "operation stage" constitutes the heart of the floating point operator. The other elements, save the exponent stage, perform the pre- and post-processing of the operands such as normalization, rounding and exceptions computation. The operation block occupies the main surface of the complete circuit. It is around this block that the efforts must be focused to get a considerable Silicon surface gain.
The gain in surface is always followed by a loss in time performance. The figure below (fig. 4) illustrates this for three different types of architectures.

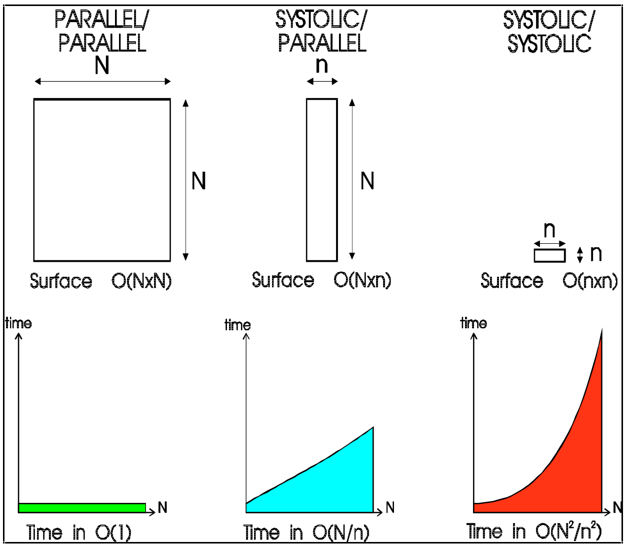

Figure 4. Surface and time of the operators

We investigated many architectures to get the best compromise. The combination of the parallel and serial schemes seems to be the best solution. In the next two sections we will investigate two main architectures. First the parallel architecture is discussed, and then the systolic architecture which offers the best surface-time compromise is presented.

3.1 Parallel operators

Parallel operators offer the advantage of speed but at the price of a large Silicon surface. Considering the multiplication operation, the parallel multipliers have a time performance ranging from a quadratic function, for a simple successive addition multiplier, to a linear dependence in O(n) ,for a Wallace tree multiplier. But the surface of these operators is always quadratic.

3.2 Systolic operators

A systolic architecture is an interconnection of a set of basic cells, very often called PEs (Processing Elements). These cells perform the same simple operation. The cells are arranged as an array. The systolic arrays enlighten the burden of I/O exchanges in I/O bound applications since each data input item can be used a number of times once it is accessed. Other advantages of systolic arrays are their ready configurability, modularity and cascadability. Moreover systolic arrays make use of simple regular PEs that are easily interconnectable. All these schemes make the VLSI implementation of systolic architectures more practical and desirable.

3.2.1 Systolic IEEE adder

In the block diagram of the IEEE floating point adder. The operation block is the heart of the system.

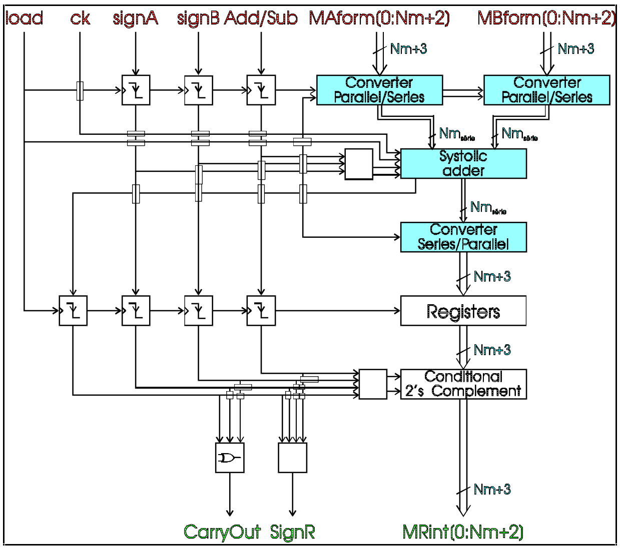

Figure 5. Operation block architecture

In the operation block (fig. 5) we perform the true addition. Once the two mantissas are pre-processed they are put in a systolic form and added in the systolic adder. Then the result is reparallelized. The result is complemented according to the sign of the mantissas and the operation carried out.

3.2.1.1 Converters

To process systolicaly parallel operands, it is necessary to convert them before the computations into a systolic format, and then reconvert the result to parallel form. The Parallel/Systolic converter and the Systolic/Parallel converters perform these operations.

3.2.1.2 Systolic Adder/Subtractor

This stage performs a systolic addition or subtraction of two operands. Two packets of the operands are added and the carry-out of the (i-1)th stage is injected in as the carry in to the i th stage. See fig. 6.

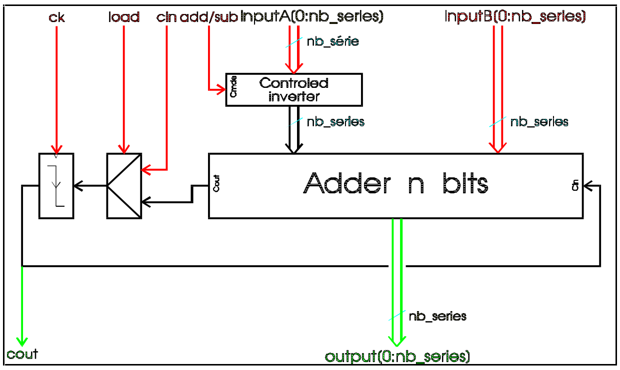

Figure 6. A systolic adder/subtractor

4. IEEE floating point adder

The IEEE floating point adder architecture is shown in the figure below. (Fig. 7) The operators generated perform any precision floating point operations. The main part of this operator is the operation stage "addition". All the other parts serve to process the data to satisfy the IEEE standard.

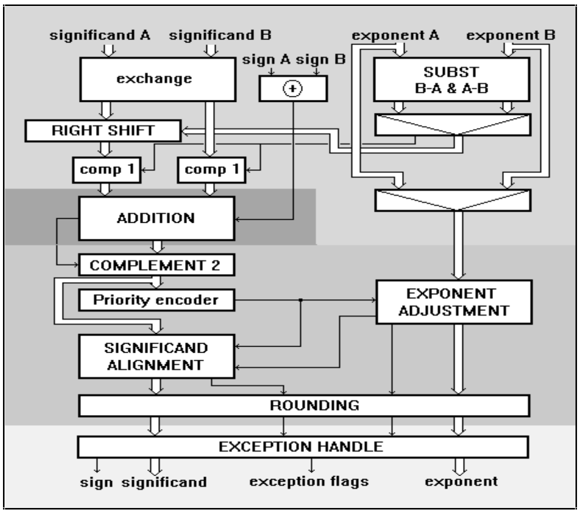

Figure 7. IEEE adder architecture

In DSP applications we know already the types and ranges of our data so we can specify exactly what types of data processing must do the operator. Thus we can be in a situation where we don't need the prenormalization stage or the exception stage or the rounding stage. In this situation our generators are able to generate personalized operators. Using this, we allow a considerable gain in surface without any loss in accuracy.

5. Implementation results

32 bits IEEE parallel floating point adder			
Block	Gain	Surface	Time
Complete		16,10 mm	90 ns
Without Rounding	7 %	14,85 mm	84 ns
Without Exceptions	5 %	15,21 mm	86 ns
Without Normalization	18 %	12,90 mm	74 ns
Without All	30 %	11,20 mm	63 ns

Figure 8. Results table for an IEEE adder

This table summarizes the implementation results of an IEEE floating point adder. We notice the gains in surface according to the type of parametrization. In fact when we generate an operator not considering the rounding modes, the exceptions and the normalization we can reach 30% surface gain and we allow at the same time a better timing performance.

6. Conclusion

This paper presented our work on IEEE floating point operators generators. These generators are fully parametrized in the sense that we are first able to realize operators to process any word length data and second in the sense that we are able to realize degraded IEEE floating point operators very useful in DSP applications. These degraded operators do not consider overflow or underflow exceptions, the denormalized format, and the special formats (NaN, infinity,...) and all the rounding modes at the same time. We presented as an example an architecture of a floating point IEEE systolic adder. Considering the implementation results we conclude that the parametrability in terms of degraded operators permits 30% surface gain.
The generators are built up using hierarchical macro-blocks, they can help realizing future floating point operators and to design generators of a higher hierarchy such as signal processors or more complex arithmetic operators.

BIBLIOGRAPHY

[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 optimiseé en surface et en performance*, MASI report, December 1993.

[3] Vaucher N., Houelle A., *Algorithmes et Architectures VLSI pour le Traitement Numérique à Virgule Flottante*, UPMC, Laboratoire MASI/CAO-VLSI, 1992.

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

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

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

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

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

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

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

[11] 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.

[12] Muller J-M., " *Arithmetique des ordinateurs, operations et fonctions elementaires* ", Masson, 1989.

[13] Mehrez H., *Algorithmes et Architectures VLSI pour le Traitement Numerique*, UPMC, Laboratoire MASI/CAO-VLSI, 1992.

[14] Guyot A., Montalvo L., Houelle A., Mehrez H., Vaucher N.,"*Comparaison of the layout synthesis of radix-2 and pseudo-radix-4 dividers* ", VLSI Design '95, New Delhi, India