ISTITUTO NAZIONALE DI FISICA NUCLEARE

Sezione di Milano

INFN/BE-94/05
26 Ottobre 1994

A Set of Parametric Operators, Algebraically Defined for Real Time Data Processing

M. Bartolucci, P.Guazzoni, G. Manfredi, G. Sechi and L.Zetta

Contribution selected for oral presentation at **EUROMICRO94**
20^{th} EUROMICRO Conference on System Architecture and Integration
September 5-8, 1994 - Liverpool, England
to be published on the Conference Proceedings

INFN - Istituto Nazionale di Fisica Nucleare
Sezione di Milano

INFN/BE-94/05
26 Ottobre 1994

A Set of Parametric Operators, Algebraically Defined, for Real Time Data Processing

Massimo Bartolucci (*), Paolo Guazzoni(+), Giorgio Manfredi(•), Giacomo Sechi(§), Luisa Zetta(+)

* Dipartimento di Fisica dell'Universita', via Celoria 16, I-20133 Milano, Italy and Guest Researcher at Istituto di Fisica Cosmica e Tecnologie Relative, Consiglio Nazionale delle Ricerche, via Bassini 15, I-20133 Milano, Italy
+ Dipartimento di Fisica dell'Universita' and Istituto Nazionale di Fisica Nucleare, via Celoria 16, I-20133 Milano, Italy
• Guest Researcher at Istituto di Fisica Cosmica e Tecnologie Relative, Consiglio Nazionale delle Ricerche, via Bassini 15, I-20133 Milano, Italy
§ Istituto di Fisica Cosmica e Tecnologie Relative, Consiglio Nazionale delle Ricerche, via Bassini 15, I-20133 Milano, Italy

Abstract

The computation complexity of a wide variety of applications, makes useful the development of new design strategies both in the direction of correctness (fault detection and avoidance) and reduction of computational time and physical device size.
Since large part of numerical algorithms involves the computation of polynomial expansions in terms of sum and products of floating point numbers, one possible way to reduce both dimension and execution time of the designed devices, is to adapt the elementary operators to the very computational needs. In this work a set of Elementary Floating Point Operators parametrically defined is presented.
Multiplier, adder and numeric format conversion operators, are defined in a custom way with variable mantissa and exponent word length. Furtherly an application is shown in which reduction of floating point mantissa length is particularly useful for the computational time and hardware size improvement.

Keywords: Arithmetic and conversion operators, FPGA, Real time processing, Nuclear electronics.

1. Introduction

In a typical nuclear experiment, the identification of reaction products is a basic problem. One of the most suitable algorithm used is the power-law [1,2] that requires the on-line computation of integer base and real exponent powers.
The randomness of nuclear events implies a constraint for the PIF (*Particle Identification Function*, i.e. powers of floating point double precision words) computation time: PIF values have to be on line computed in an upper bounded time (< 100 μs/datum), independently from the final hardware release. This request may be fulfilled only by a hardware able to implement asynchronous dynamic real time processes [3,4].
At the moment, except our previous preliminary works [5,6] to verify for this application the performances of a general architecture and of a commercial floating point processor, no care has been dedicated to an ad hoc system, fully programmable.
The algorithm chosen for the power computation is the Chebichev polynomial expansion. So a power computation may be done only by means of additions, subtractions and multiplications. Moreover the results we need, have to be discretized on a 1K-range of 16-bit integer words. To design an ad hoc computational unit, being designed with multiprocessor parallel (MIMD, Multiple Instructions Multiple data) architectures and at the end realized as ASIC (Application Specific Integrated Circuit) component, we need to construct an operator set performing arithmetic operations and format conversions.
The present contribution describes a set of *Elementary Floating Point Operators* (EFPO), multiplier, adder, numeric format conversion, characterized by the possibility of custom definition of the mantissa and exponent word length. Since the precision of a computation depends mainly on the mantissa word length of operands and results, using EFPO it is possible to design operators that fit well the effective computational needs, avoiding, in some cases, the useless waste of precision implied with long mantissa word computations (Double Precision, 68881 Extended Precision, etc.).
EFPO specified by using Boolean equations, formulated in a full general way, may easily fit a relatively high number

of different technologies (FPGA-Field Programmable Gate Arrays- such as Lattice [isp]LSI 10XX, Xilinx X3000, or ASIC as ES2 Standard Cells) differing in precision, computation parallelism and fault tolerance level, thanks to its algebraic parametric definition. In particular we have implemented the operator set in Lattice technology (Lattice ispLSI 1048 / 50 Mhz).
In section 2. a detailed description of the problem concerned with the particle identification by means of PIF computation is given. In section 3. the guidelines of the operators project by means of the multiplier example is described. In section 4. the main results achieved simulating computation of PIF by means of the defined set of operators are reported, together with the conclusions.

2. Problem definition and parallel solution

In a typical nuclear reaction, an incident ion hit a pure isotope target, creating different reaction products, characterized by an own mass and charge. These products may be detected by a telescope of two detectors, the first one transparent to the reaction products that stop in the second detector. The identification is obtained by computing PIF, used as a trigger to accept or reject particular kind of particles.
PIF has the form $PIF=(E+\Delta E)^x-(E)^x$ (where ΔE and E are the energies lost in the two detectors and x is a parameter dependent on the nuclear reaction) and is characterized by having different peaks corresponding to the different ions.
E and ΔE values are 12-bit integer words while x is a real number. For PIF computation E and ΔE are converted to floating point format.
So the central problem of PIF computation is reduced to the calculation of A^x, the positive real base and exponent power. This function may be computed by composing transcendental functions as follows:

$$A^x = 2^{\frac{x\ln(A)}{\ln(2)}}$$

The computation of natural logarithm and of real power of 2 requires to expand them as polynomials, thus obtaining a composition of sums and multiplications of real numbers.

$$f(\underline{z}) \equiv \sum_{j=1}^{n} c_j \underline{z}^j = P(\underline{z})$$

In order to check the choices made in EFPO design, we have chosen a scheme to compute polynomial expansion (fig.1). This is not the only possible scheme but, employing more sums and products than other parallel schemes[7], it is a good test platform for EFPO. In fact the errors introduced by a computation increase on the average with the increasing number of floating point operations performed.

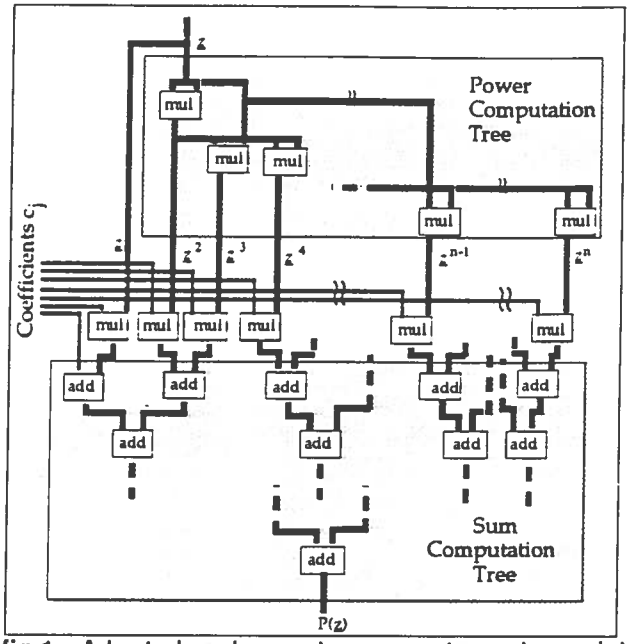

fig.1. Adopted scheme to compute polynomial expansions

Referring to fig.1, first the *n* increasing powers of the expansion variable ($\underline{z}$) are computed as repeated multiplications, then the results are multiplied by the *n* coefficients and the monomials are added along a binary tree to obtain the final result $P(\underline{z})$.

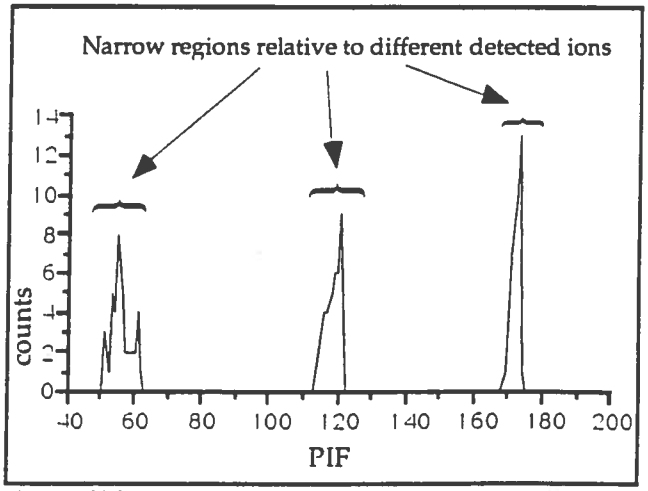

fig.2. PIF value plot

Only two kinds of operators are used to compute the expansion: adders and multipliers. So independently of the

As an example of exception handling in the multiplication we consider the following two classes of R_{FP}:

$$[Q0] = \{y \in R_{FP} / \text{y exponent is } -(2^{m-1}-1)\}$$

$$[\infty] = \{y \in R_{FP} / \text{y exponent is } 2^{m-1}\}$$

We use Q0 and ∞ to represent the two classes.
By means of the definition, the exception detection is quite simple and it is independent of the multiplier input mantissa. The analysis of the exponent of the input vectors is enough. After having introduced the exception it is necessary to define the multiplications involving them. The semantic of exception multiplication and that of corresponding real number multiplication, are similar (see table 1).
It is possible that a Q0 result is acceptable or not as a computational result. Since it is not possible to encode this information in the floating point word, we add to the multiplication output word an underflow flag bit (UFW).

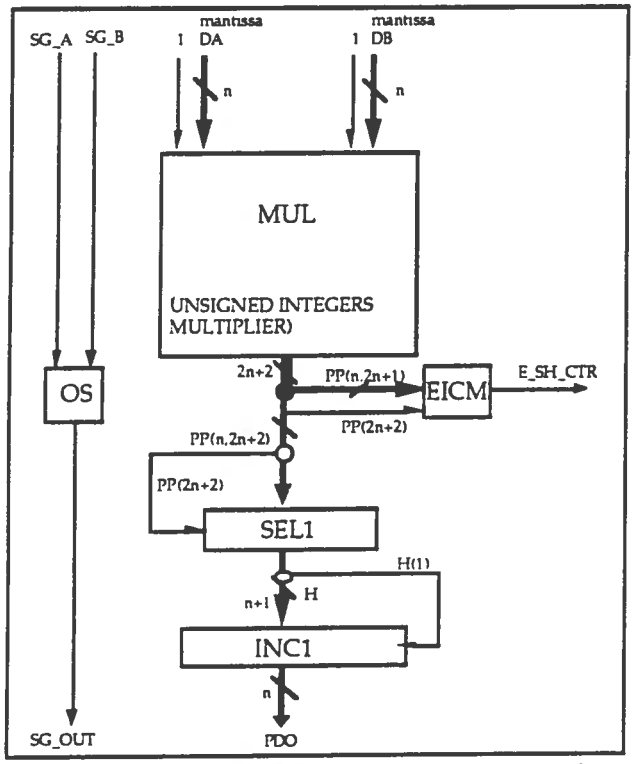

fig.3. Output mantissa computation section of the parametric floating point multiplier.

If the multiplier output is Q0 and UFW=1, the result is considered 'underflow zero' and, if necessary, it may be rejected as invalid (u0). This is the case of the product of operands having an exponent less than that of Q0 class elements (the minimum exponent representable in the chosen floating point format). On the contrary when the output exponent is exactly equal to that of Q0 elements, the output is an unflagged 'quiet zero' q0 (UFW is set to 0).
With the semantic given in table 1, q0 behaves like the real zero when multiplied by itself or some other not-exception floating point number.
Another auxiliary flag (OFW) may be defined to code the undefined product q0 • ∞ (NAN). By means of the two flags OFW and UFW it is possible to encode also the overflow exception (in order to simplify the exception identification by means of a controlling unit).
The study of the multiplier may be subdivided in two sections, for mantissa and exponent handling. These two sections work on their inputs almost independently one from each other except for a control byte (E_SH_CTR) that is early determined in the mantissa chain of computation (fig.3).

a] the mantissa section of the multiplier.

The elementary functionalities relative to the mantissa handling may be expressed in a pre-formal way as follows (refer to fig.3 and 4):

1. The operands are (n+m+1)-bit words where *n* is the mantissa (DA, DB) length and *m* is the biased exponent (EXP_A, EXP_B) length. The additional bit is for the sign (SG_A, SG_B).
The exponent bias is $2^{m-1}-1$.

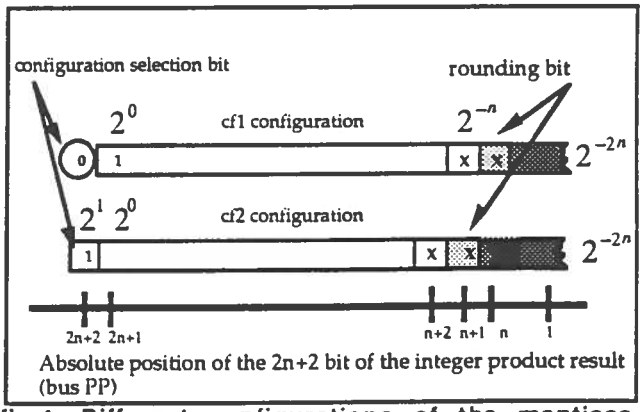

fig.4. Different configurations of the mantissa product result.

2. The sign of the result is the *exclusive or* of the input vector signs independently of the size parameters of the operands (module OS in fig.3).
3. The two input mantissa are extended of one bit (set at '1') in the MSB position. This bit is usually omitted in the mantissa representation, but it is necessary for the computation.
The product of the two input mantissas is executed as integer multiplication not depending on the point position. The product length is a (2n+2)-bit word (PP in fig.3).

algorithm, the major effort to speed up the computation is to reduce the signal delays in the devices, implementing the operator set.
To obtain this goal we have chosen to reduce the gate level complexity of digital devices fitting the computation precision of the operators to the problem needs. So it is necessary to model elementary operations by means of a parametric description of the functionalities. Then with simulations it will be possible to evaluate what precision is really necessary for a target problem, varying the description parameters.
For example, since discretized values of PIF, concentrate in disjoint narrow regions of the whole range (fig.2) -each corresponding to a precise detected particle-, it is interesting only the presence of PIF values on the narrow region. As an outcome, if the narrow regions in which PIF values concentrate are well separated, a few channels shift introduced in the computation of PIF does not compromise the final identification.

3. Description of the elementary operator set

Starting from an algebra of positional representation of floating point numbers, with variable length mantissa and exponent (parametric format), and studying properties of those number representations, the design of elementary operators has been performed, step by step, up to a formal definition by means of Boolean logic equations.
Considering a real number instead of its floating point representation N_p, the relative error introduced may be majored, in chopping and rounding mode respectively, as follows:

$$\varepsilon_{r-chopping} \le \frac{|N_{p+1} - N_p|}{2^{q(p)}} = 2^{-n}$$

$$\varepsilon_{r-rounding} \le \frac{|N_{p+1} - N_p|}{2^{q(p)+1}} = 2^{-n-1} \qquad (2)$$

where N_{p+1} is the N_p successor in an increasing numbering and q(p) represents N_p exponent.
As a consequence of the above formulas the precision of the representation depends only on the chosen length (n) of mantissa. So starting from the above considerations, to reach our goal, i.e. the PIF computation with the truncation method, we have defined three operators on the floating point number set (R_{FP}): multiplication, sum and subtraction.
By means of algebraic properties it is possible, independently of the mantissa and exponent length, to formulate the functionalities necessary to implement the operations.

EFPO functionalities have been further translated in Boolean logic functions, and, where possible, reduced to a minimal SOP (*Sum Of Products*) or And-Or-Exor form.
It is not possible, at the functional level of the elementary operators description, to decide the optimal form of the equations, since it strictly depends on the hardware resources available to implement the functionalities.
Particular care has been devoted to the definition and handling of exceptions arising in the computation, in order to consider the use of the operators as building blocks of the architecture of a computational unit. The definition of the elementary operators was carried out as an heuristic activity: we paid the major efforts to parallelize and reduce the elementary operator building blocks. The result is a fully combinatorial device.

3.1. The multiplier design

With regards to the adopted design strategy, we present here the description of the parametric floating point multiplier.

Some algebraic rules for the exception handling in the product

1. $\infty \bullet \infty = \infty \bullet A = A \bullet \infty = \infty \qquad A \in NER_{FP}$
2. $q0 \bullet q0 = q0 \bullet A = A \bullet q0 = q0 \qquad A \in NER_{FP}$
3. $q0 \bullet \infty = \infty \bullet q0 = NAN$
4. $A \bullet B = u0 \qquad A, B \in NER_{FP} \wedge EXP(A \bullet B) < -(2^{m-1} - 1)$
5. $A \bullet B = q0 \qquad A, B \in NER_{FP} \wedge EXP(A \bullet B) = -(2^{m-1} - 1)$
6. $A \bullet B = \infty \qquad A, B \in NER_{FP} \wedge EXP(A \bullet B) \ge 2^{m-1}$

NAN is Not A Number
NER_{FP} is the R_{FP} subset of not - exceptions
EXP() is the real exponent of the number in brackets

table 1. Multiplication exception table.

As it will be shown, the multiplication of floating point numbers may be reduced substantially to integer multiplication and exponent adjustment independently of the input size parameters (n, m are respectively the mantissa and exponent number of bits).

4. The result of the product may be in one of two possible configurations *cf1* and *cf2* (refer to fig.4).
The not-rounded output mantissa is selected depending on the integer product output configuration.
Only (n+1) most significant bits of the product are considered as H output vector (see fig.3) after configuration selection (the 'always 1' most significant bit is omitted).
5. By means of the rounding bit (least significant of the H word) the remaining part of the H word is incremented or not (module INC1 in fig.3). The rounded mantissa (output mantissa) is so obtained.
6. The PP word is sufficient to obtain information useful for the determination of the output exponent. If the input mantissa product is in *cf1* configuration, the output exponent is determined only by the exponents of the two input words.
If the configuration is *cf2* the resulting output exponent is that obtained by the two input exponents incremented by 1.
It is possible that even if the input mantissa product is *cf1*, the rounding operation produces a *cf1* -> *cf2* transition.
The EICM module in fig.3 determines whether to increment or not the output exponent (by means of E_SH_CTR control bit).

- If PP(2n+2)=1 then E_SH_CTR =1 :
  the *cf2* configuration determines output exponent increment.

- If PP(2n+2)=0 and PP(j)=1 with n<j<2n then E_SH_CTR=1:
  the *cf1* -> *cf2* transition determines the output exponent increment.

- If PP(2n+2)=0 and $\exists j$ / PP(j)=0 with n<j<2n then: E_SH_CTR=0 :
  the configuration *cf1* does not change the output exponent sum.

b] The exponent section of the multiplier.

1. The two m-bit exponents are added together obtaining an (m+1)-bit result (EQ word in fig.5): the integer result is equal to the sum of the unbiased integer values of the two input exponents plus $2(2^{m-1}-1)$ where $(2^{m-1}-1)$ is the general expression of the bias factor.
2. Depending on E_SH_CTR value, the EQ word is incremented or not: the result is again an (m+1)-bit word (the maximum possible value for EQ is 11..10).
3. The bias factor is subtracted from EQ to obtain ER word (module BFS fig.5).
4. In absence of anomalous condition (see later on) the output exponent ES is equal ER.
5. If one of the two inputs is q0 (exponent 00..00), and the other is different from ∞ (11..11), the output value is q0. (UFW flag is inhibited in UOM by SUFW line, fig.5).

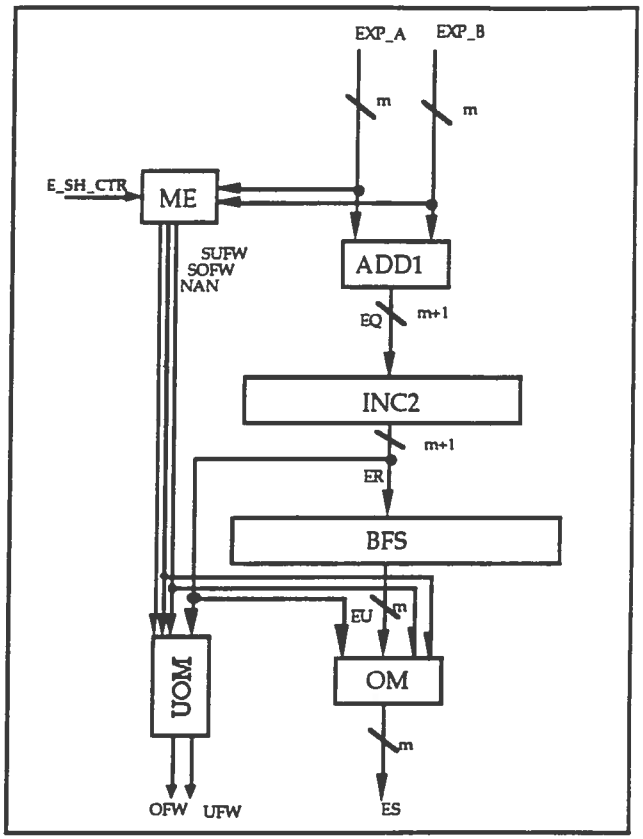

fig.5. Output exponent computation section of the parametric floating point multiplier.

6. ∞ exception may be determined by the computation: to detect this situation it is enough to analyze the ER word. If the current integer value of this word is $\geq 2(2^{m-1}-1)+2^{m-1}$ the ∞ exception is flagged. In this case the output exponent is set to '11..11' (OM module).
7. One of the two inputs may be ∞; in this case if the other operand is not q0, then the output is ∞ (OFW is flagged by means of 'MUO'). This situation is detected by ME module (fig.5) and flagged by the SOFW line to UOM.
8. The product ∞ • q0 is detected directly analyzing the input exponent: as a consequence the NAN is flagged (NAN line in fig.5) on the two lines OFW and UFW ('11' as output of the OUM module). The output exponent is '11..11' in this case.

3.2. Elementary operators implementation.

As a further step, it has been studied a possible implementation of the elementary operators using *programmable logic devices*. PLD are well suited for the prototypal stage of the design because of their low cost and easy reconfigurability.

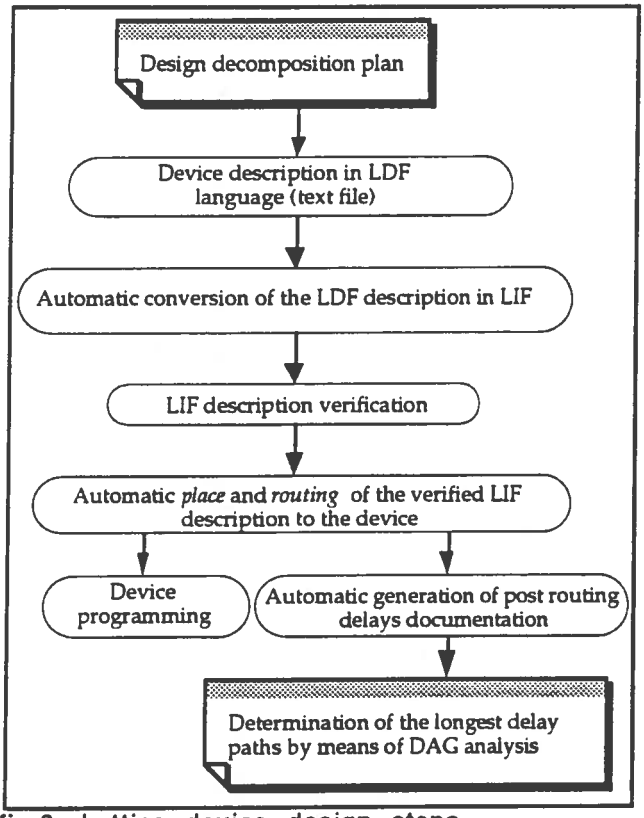

fig.6. Lattice device design steps.

We have chosen *Lattice* ispLSI 1048/50 Mhz [8] to implement the Boolean equations defining the elementary operators. These devices are composed of a relatively high number of interconnected logic cells: each of the 48 cells may support up to 4 product terms of up to 18 input variables each (SOP form). It is also possible to configure the cell output in order to implement And-Or-Exor logic [8].
To implement the operators in Lattice device technology we have used Lattice pDS (pLSI, ispLSI Development System[9]). The device design steps are summed up in fig.6.
The original design splitting into the target devices has to be planned before starting to use the development system.
The SOP form equations defining the operators are transformed where possible in And-Or-Exor form and fitted to the device GLBs (Generic Logic Blocks). The Boolean equations are written in *Lattice Device Format* (LDF) language with other information relative to the I/O connections and cells placement in the devices.
After verification and routing, information relative to the average internal signal delays is available in order to determine the longest delay paths by means of *Directed Acyclic Graphs* (DAG).
In the prototypal stage of the design it has been possible to use different solutions for the elementary operator implementation in order to compare the device size (in terms of number of components, cells and I/O used) and the maximum toggle frequency allowed, for different values of mantissa length.

4. Results and conclusions

We have checked EFPO in simple computations in enlarged and reduced precision and compared the results with those obtained with the standard operator set of the Think C compiler mathematical libraries on *Universal Format*[10] floating point number representation (same precision as MC68881 extended format) on a Macintosh Quadra 610.
At the end, we have used EFPO in the frame of a minimal and fast computing unit, ad hoc defined for computing $PIF=(E+\Delta E)^X - E^X$, where E and ΔE are two 12-bit words representing experimental data.
Considering the 1K discretization adopted in the experiment we have evaluated the possibility of computing PIF with a reduced 15-bit mantissa, reduced with respect to IEEE single precision 23-bit mantissa.
We have implemented and tested the three floating point operators, previously generally defined in parametric format, with only 15-bit mantissa.

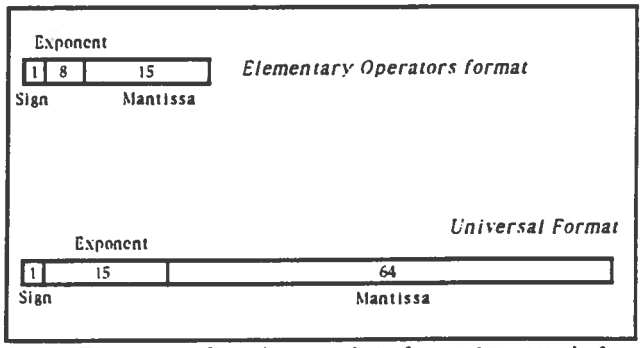

fig.7. The two floating point formats used for the PIF test.

Furtherly PIF has been computed by means of EFPO and the results compared with those obtained with Think C compiler standard libraries (fig.7).
The inputs are groups of 39 pairs of energy values (ΔE, E) for the products of the simulated nuclear reactions [11]):

1) $p + {}^{27}Al$ at $\vartheta_{lab} = 10°$

100 μm thin detector, x = 1.73 E_{inc} = 27 MeV
detected particles are *protons, deuterons, tritons, helium-3 and alpha.*

2) $^{12}C + ^{24}Mg$ at $\vartheta_{lab} = 30°$
30 μm thin detector, x = 1.6, E_{inc} = 84 MeV
detected products range from *alpha* to *nitrogen.*

Fig.8a shows the plot of the PIF computed in Universal Format precision. In fig.8b are reported the results obtained using the same algebra with the 15-bit mantissa EFPO set.
It is evident from the figures that an extended precision computation is not really necessary to maintain the separation between PIF values relative to different particles.
The good agreement between the two methods is more clearly evident in fig.9, where the frequency distributions for only one reaction product (tritons) are drawn in more enlarged representation.

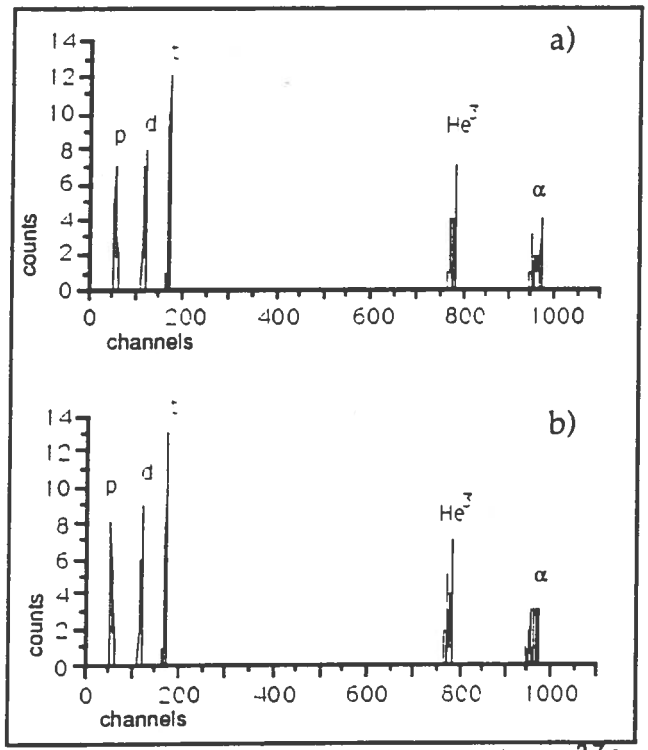

fig.8. PIF frequency distribution for the p+^{27}Al reaction computed in: a) Think C Universal Format, b) 15-bit mantissa format.

The results shown in fig.10 confirms the good agreement between the two methods also in the case of the second reaction.
It is to notice that the mantissa reduction implies a mean loss of precision of only 0.2%, for the full not discretized PIF range ([1.885e+04 , 2.382e+05]).

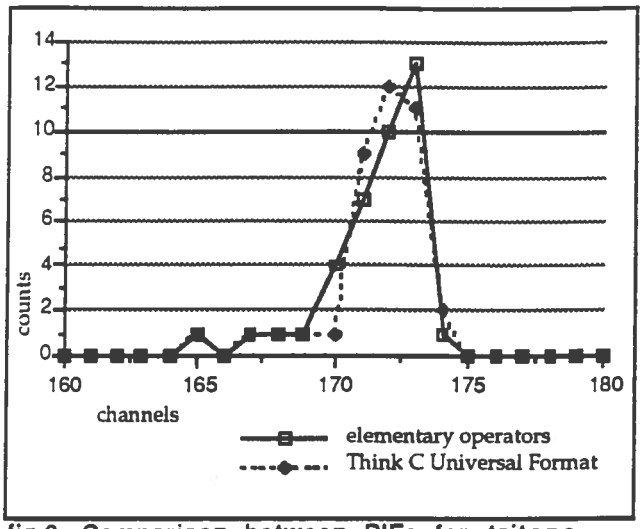

fig.9. Comparison between PIFs for *tritons* computed with the two different precisions.

In table 2 are shown some of the preliminary results obtained with Lattice pDS[9] and the Boolean equations defining the algebraic floating point operators, in the design of a possible hardware implementation of EFPO set with FPGAs.

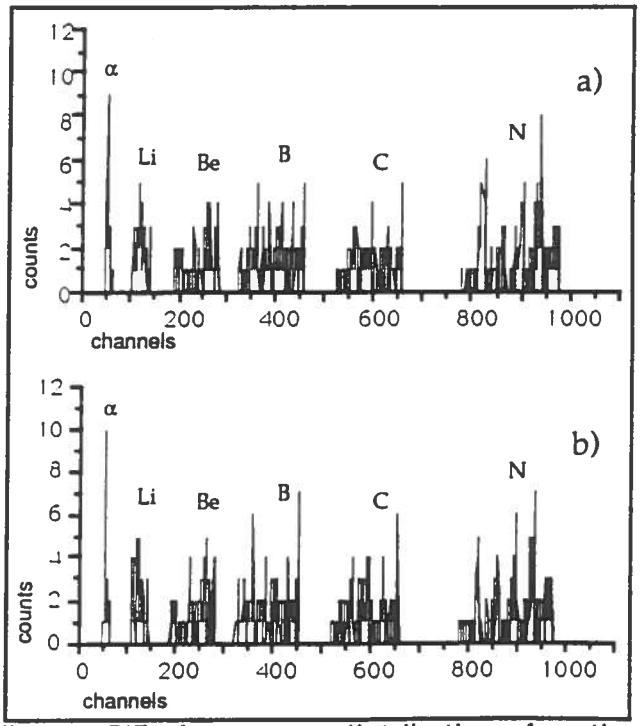

fig.10. PIF frequency distribution for the $^{12}C+^{24}Mg$ reaction computed in: a) Think C Universal Format, b) 15-bit mantissa format.

In the table are shown the delays and size of a floating point adder in two different EFPO release: a 32-bit (single

precision adder, 23-bit mantissa and 8-bit exponent), and a 24-bit (15-bit mantissa and 8-bit exponent) adder.
The size is computed in terms of the number of Lattice ispLSI 1048 devices necessary to implement the two adders.

Operator release	n° of devices*	timing (ns)
Addition 32 bit 1 sign 8 exponent 23 mantissa	8	560.7
Addition 24 bit 1 sign 8 exponent 15 mantissa	4	395.3
*Lattice ispLSI 1048 / 50 Mhz		

table 2. Some prototypal study results.

The size of the single precision adder is two times that of the 24-bit adder. Also longest path delay is reduced significantly using 24-bit addition operator. It is to note that the reduction in size and delay of the 24-bit adder operator are due to the reduced mantissa word (15-bit long), since in the two operator realized the exponent section is maintained unchanged (8-bit long).
As conclusion we may affirm that EFPO represents a tentative unusual approach to the problem of saving hardware and time computation, especially in the case of some algorithms of relevant complexity, when their computation requires big effort in building a hardware ad hoc designed, to allow an overall precision that may be redundant with respect to the very needs of the applications as in the case of PIF.

References

[1] GOULDING F.S. - HARVEY B. G.: "Identification of Nuclear Particles", Ann. Rev. Nuc. Sci, 25 pp. 167-240 (1979)

[2] ENGLAND J. B. A.: "A fast analogue particle identification system", Nucl. Instr. and Meth., 106 pp. 45-59 (1973)

[3] JOSEPH M.; "Problems, premises and performances: some questions for real time system specification", Proceedings REX Workshop, The Netherlands, June 1991, Lecture Notes in Computer Science, Springer-Verlag Ed.

[4] NATARAJAN S., ZHAO W., "Issues in Building Dynamic Real-Time Systems", IEEE Software Magazine, pp. 16-21 September 1992

[5] M. ANNUNZIATA, P. GUAZZONI, L. LISCA, G.R. SECHI, L. ZETTA, "A.M.D.A.S. An Advanced Microprogrammed Data Acquisition System, A First Evaluation Prototype", Microprocessing and Microprogramming, vol.18, pp. 515-524, (1986)

[6] BRAGAGNINI W., GUAZZONI P. , PITALIERI M., ZETTA L.: "Computational Logic Unit for a Microprogrammed Data Acquisition System: an evaluation prototype", Microprocessing and Microprogramming, vol. 30, pp. 67-74, (1990)

[7] ANDRIESSEN J.H.M., "Polynomials on the Delft Parallel Processor", Microprocessing and Microprogramming, vol. 20, pp. 107-112, (1987)

[8] LATTICE, "The Lattice Data Book", 1992

[9] LATTICE, "pDS User Manual", 1992

[10] BORENSTEIN P., MATTSON J.: "THINK C User Manual", SYMANTEC, 1991

[11] GUAZZONI P., SECHI G., ZETTA L.: "Computer Simulation of Ion Discrimination in Mass and Charge", Nucl. Instr. and Meth., 227, pp. 526-534, (1984)