ISTITUTO NAZIONALE DI FISICA NUCLEARE

Sezione di Milano

INFN/TC-95/18
19 Giugno 1995

M. Bartolucci:

FUNCTIONAL EQUATIONS OF AN ELEMENTARY OPERATOR SET

SIS–Pubblicazioni
dei Laboratori Nazionali di Frascati

INFN - Istituto Nazionale di Fisica Nucleare
Sezione di Milano

INFN/TC–95/18
19 Giugno 1995

FUNCTIONAL EQUATIONS OF AN ELEMENTARY OPERATOR SET

M. Bartolucci
Dipartimento di Fisica dell'Univrsità di Milano and
INFN–Sezione di Milano, Via Celoria 16, I–20133 Milano, Italy

This report represents an extended discussion and a full representation of the equations of a set of algebraically operators, parametrically defined, elsewhere already presented [1,2].

1. Introduction

Many complex algorithm computations heavily involve the use of floating point arithmetic. Numerical algorithms are in general decomposed in the repetition of a limited number of operations on integers or floating point numbers. As a consequence the overall algorithm execution time may be substantially reduced to the number of operations performed times the average time needed to perform such an operation.

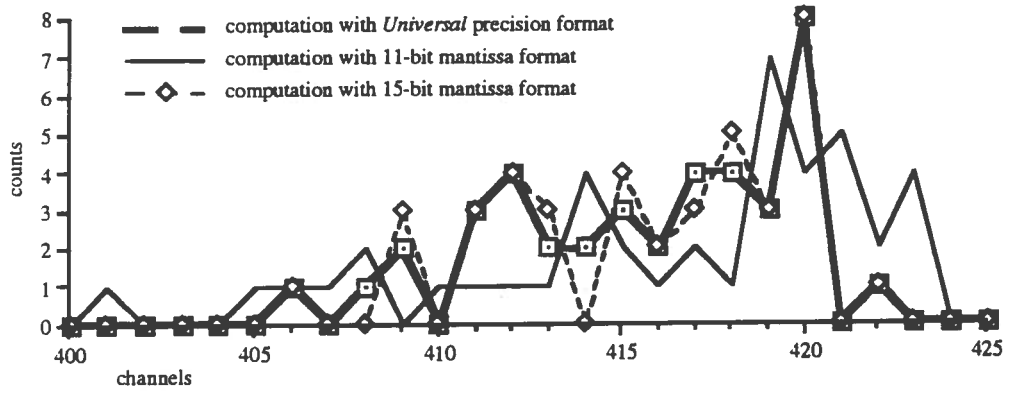

fig.1. Particle Identification Function computed with three different floating point formats.

The single-operation execution time does not depend only on adopted implementation technology, but also on design and target application constraints.

In ref.1 is shown how the computation of particle identification function (PIF=(E+DE)^x-E^x), i.e. of an algorithm essentially based on the computation of two integer basis, real exponent powers, may be performed using custom defined floating point format that allow a computation faster than that obtained with operators defined on standard floating point formats. In particular the computational outcomes when modifying the floating point format mantissa size have been compared. The agreement obtained in the computation of particle identification function with 64, 15 and 11-bit mantissa format is shown in fig.1.

In the following some considerations on floating point representation are made in order to introduce the Boolean description of the multiplier and adder operators.

2. General considerations on the floating point representation.

Real number properties are altered in finite floating point representation (from here on R_{FP}). As a consequence the algebraic properties of $\Re$ (real number set) lack in general for R_{FP} (*e.g.* the closure for addition and multiplication). For the aims of the present work only a base-2 parametric floating point representation (R_{FP}) is introduced. A generic element **X** of the set of positive floating point numbers is $X = sg_x 2^q (1+M)$, where sg_X is the operand sign and q is an integer exponent, with $-(2^{-(m-1)}-1) \le q \le 2^{-(m-1)}$ where m is the number of bits composing the binary word used to encode the exponent $\left(q \to \sum_{i=1}^{m} 2^{i-1} EA(i)\right)$ and M is the mantissa of **X** with $0 \le M < 1$.

The mantissa algebraic expression is $M=\sum_{i=1}^{n} 2^{-n+i-1} DA(i)$ where $DA(j)$ is the j-th bit of the n-bit word encoding mantissa (DA(j), 1≤j≤n). The exponent encoding is obtained biasing q exponent by an additive quantity, $2^{(m-1)}-1$: the exponent word is always an unsigned integer. So when the encoding word is $\underline{0}$ ($\underline{0}$ and $\underline{1}$ are 0..0 and 1..1 vectors of arity implicitly defined in the equalities) it represent the largest negative exponent ($-(2^{-(m-1)}-1)$) while when it is $\underline{1}$ it encodes the largest positive one ($2^{(m-1)}$).

As an extension the term 'mantissa' is also used for 1+M depending on the context; in fact the 1 added to M is usually omitted in the mantissa representation and concatenated, for computational aims, to the word representing M in position n+1 ((n+1)-bit $\underline{DA}$ word).

In the following R_{FP}^{+} is the label for positive subset of R_{FP} and R_{FP}^{-} the negative one.

Multiplication and addition operators are defined on R_{FP}^{2}. The natural way of multiplying elements of R_{FP} is:

$$X \bullet Y = sg_x 2^{q_x}(1+M_x) \cdot sg_y 2^{q_y}(1+M_y)$$

while that of adding them is given by:

$$X + Y = sg_x 2^{q_x}(1+M_x) + sg_y 2^{q_y}(1+M_y).$$

It is simple to show in both cases that the operation is not closed on R_{FP}: so exceptions and exception handling rules are introduced. Four subsets (exception sets) have been defined in R_{FP} :

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

tab.1. Multiplication exception table.

Some algebraic rules for the exception handling in the product

1. $\infty \bullet \infty = \infty \bullet X = X \bullet \infty = \infty \qquad X \in NER_{FP}$
2. $q0 \bullet q0 = q0 \bullet X = X \bullet q0 = q0 \quad X \in NER_{FP}$
3. $q0 \bullet \infty = \infty \bullet q0 = NAN$
4. $X \bullet Y = u0 \qquad X,Y \in NER_{FP}$ e $EXP(X \bullet Y) < -(2^{m-1}-1)$
5. $X \bullet Y = q0 \qquad X,Y \in NER_{FP}$ e $EXP(X \bullet Y) = -(2^{m-1}-1)$
6. $X \bullet Y = \infty \qquad X,Y \in NER_{FP}$ e $EXP(X \bullet Y) \geq 2^{m-1}$

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

In the following any element of the above exception sets has been represented with the symbols ±Q0, ±∞ or Q0,∞ respectively when the sign information is determinant (addition) or not (multiplication).

The flag UFW is used to signal an 'underflow zero' u0 that is to say a Q0 element generated by multiplication according to tab.1 and tab.2 rules. The flag OFW is set when ∞ result is determined. The symbol q0 represents Q0 'quiet zero' elements (input operands Q0 are always q0, the rules for their generation as result of operations is in tab.1 and tab.2). When set together they signal Not A Number (NAN) operation results (refer to the tables).

In tab.1 and tab.2 the rules defined for exception handling in the product and sum are shown.

With the semantic given in tab.1, q0 behaves like the real zero when multiplied by itself or some other not-exception floating point number.

Before introducing boolean equations of multiplication and addition operators, the notation used to write equations is explained:

X(j) is the j-th component of the vector $\underline{X}$ of boolean elements (0,1).

$\underline{X}(i,j)$ represents the word extracted from $\underline{X}$ with i starting and j final positions.

$X_j(i)$ represents the i-th bit of the j-th vector of X, where X is a sequence of vectors.

tab.2. Addition exceptions.

Exception handling in the computation of floating point addition

1. $(+\infty) + (-\infty) = NAN = (-\infty) + (+\infty)$
2. $(\pm\infty) + (\pm\infty) = \pm\infty$
3. $(\pm\infty) + q0 = q0 + (\pm\infty) = \pm\infty$
4. $(\pm q0) + (\pm q0) = \pm q0$
5. $X + q0 = q0 + X = X \qquad X \in NER_{FP}$
6. $(\pm\infty) + X = X + (\pm\infty) = \pm\infty \qquad X \in NER_{FP}$
7. $X + Y = \infty \quad X,Y \in NER_{FP} \wedge EXP([X+Y]) \geq 2^{m-1}$
   $(X + Y = Z \quad X,Y,Z \in NER_{FP} \wedge EXP([X+Y]) < 2^{m-1})$
8. $X + Y = u0 \quad X,Y \in NER_{FP} \wedge EXP([X+Y]) < -(2^{m-1}-1)$
   $(X + Y = Z \quad X,Y,Z \in NER_{FP} \wedge EXP([X+Y]) > -(2^{m-1}-1))$
9. $X + Y = q0 \quad X,Y \in NER_{FP} \wedge EXP([X+Y]) = -(2^{m-1}-1)$
10. $X + Y = q0 \quad X,Y \in NER_{FP} \wedge EXP(X) = EXP(Y)$
    $MANT(X) = MANT(Y) \quad SG(X) = -SG(Y)$

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

In order to simplify the equation form the following notation for a special functions used in the adder that is not in explicit form (AND-OR-EXOR) is shown:

$$X^j = \begin{cases} 1 & \text{if } X \text{ boolean word equals the integer } j \\ 0 & \text{otherwise} \end{cases}$$

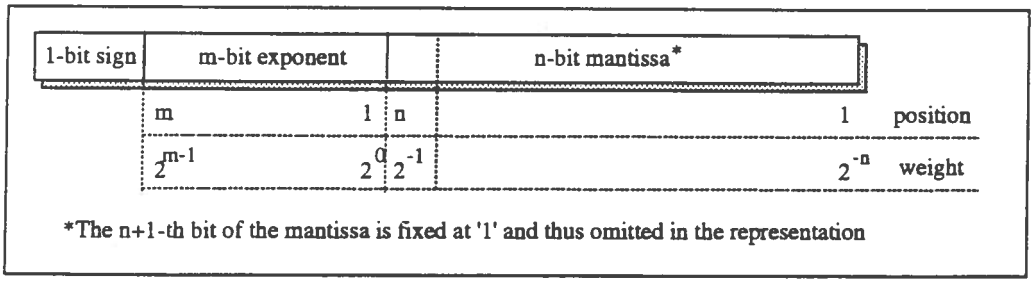

fig.2. The general floating point format chosen.

The equations are built with logical operators AND (represented in equations with symbol "·" or most commonly omitted, while symbol "$\prod$" stands for for multiple-operand AND), OR (symbol "+" or "$\sum$" for multiple-operand OR), EXOR (symbol "$\oplus$") and NOT (a bar over the variable or expression as symbol) and if not explicitely put in

evidence with underlining, they involve single bits. When variables involved in equations are underlined, those equations hold bitwise.
The floating point representation choosen is shown in fig.2.
The partameters n,m,k, stand respectively for:

n { *number of mantissa bits in the chosen format*
m { *number of exponent bits*
k { *number fo bits saved in the shift to avoid rounding error [8]*

3. Multiplier functionalities and boolean equations

In this paragraph the equations of the multiplier are unrolled: the equation discussion is briefly introduced by multiplication generic properties followed by detailed functional step-by-step algebraical discussion of the operator.
The multiplier functionalities are mainly divided in two parts: an integer multiplier section (the most critical part of the whole multiplier design) and an exponent and mantissa handling section.
In the following equations the integer multiplier is represented as a composition of adders with carry look ahead[3]. On the contrary, today fast integer multiplier use algebraic techniques to reduce the carry propagation overhead [4,5,6]; but it is not possible in the present general work to propose a fully optimized product, because final optimization depends strictly on the technology chosen to implement the design.
The result of the product may be in one of two possible configurations *cf1* and *cf2* .
Given two R_{FP}^{+} operands (sign handling is trivial in the product) **X** and **Y** of DA and DB mantissa and q and r exponent, the product result satisfies algebraic constraints.
The operands are:

$$X = 2^{q}\left(1+\sum_{j=1}^{n} 2^{-n+i-1} DA(i)\right) \qquad Y = 2^{r}\left(1+\sum_{j=1}^{n} 2^{-n+i-1} DB(i)\right)$$

Fixed the exponents q and r a lower bound for the product when DA(1,n)=0 (from here on X(k,l) is the portion of X word ranging from position k to l included) and DB(1,n)=0 is determined, while the upper bound is obtained when DA(1,n)=1 and DB(1,n)=1:

$$(X\cdot Y)_{\min} = 2^{q+r} \qquad (X\cdot Y)_{\max} = 2^{q+r}\left(\sum_{i=0}^{n} 2^{-n+i-1}\right)^{2} < 2^{q+r+2} \qquad (1)$$

It is easy to observe that it is possible to transform the product of the two input mantissas in an integer product multiplied for a fixed factor (2^{-2n}):

$$Mant(X)Mant(Y) = 2^{-2n}\left(2^{n}+\sum_{i=0}^{n-1} 2^{i} DA_{i+1}\right)\left(2^{n}+\sum_{i=0}^{n-1} 2^{i} DB_{i+1}\right) = 2^{-2n} I_A I_B \quad 2^{n} \le I_A, I_B \le 2^{n+1}-1$$

The exponents q and r are coded in the biased two m-bit words EA and EB respectively.

In the following the boolean equation of the multiplier are described referring to fig.3 directly referring in the text to the corresponding modules in figures.

MUL. (multiplier)

The n+1-bit words DA and DB are multiplied as integer words to yield the (2n+2)-bit word integer mantissa product PP.

The partial product VPP are computed as function of the input operand mantissas and then they are all added (carry look ahead adders) to form PP.

$DA(n+1)=1$

$DB(n+1)=1$

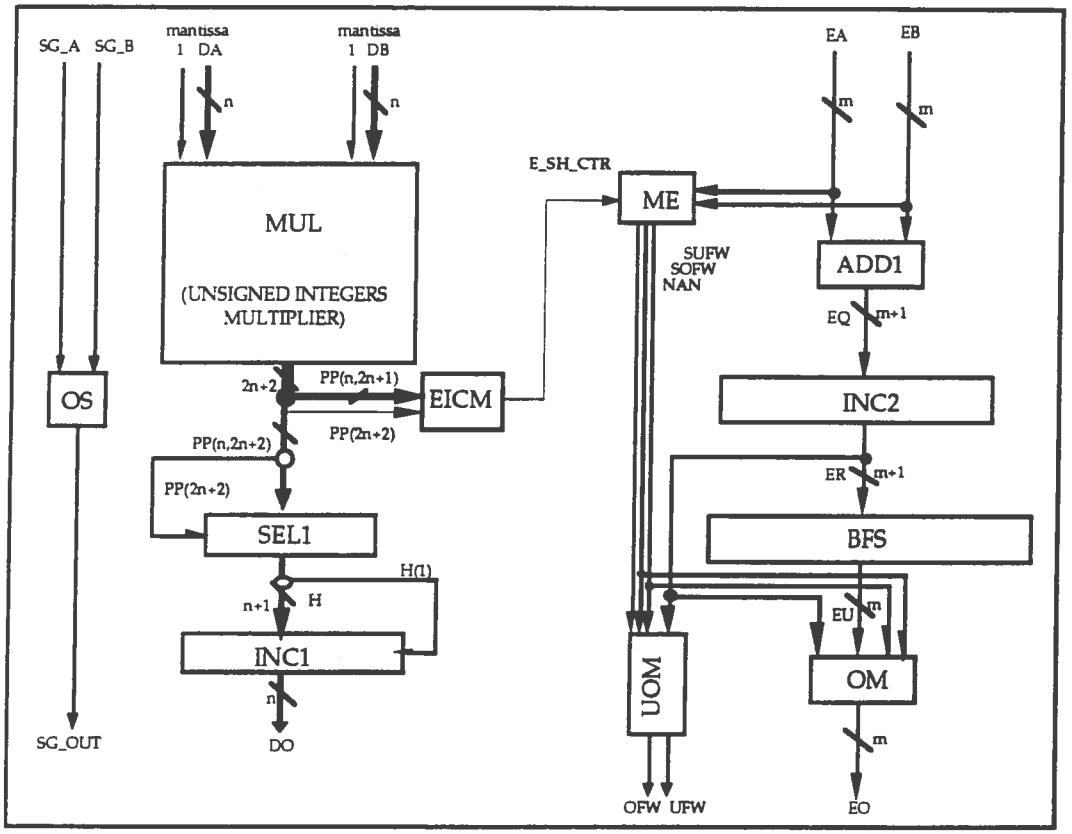

fig.3. Floating point multiplier scheme.

$$\left.\begin{aligned} VPP_r(j) &= 0 && 1 \le j \le r-1 \\ VPP_r(j) &= DB(r)DA(j+1-r) && r \le j \le n+r \\ VPP_r(j) &= 0 && n+r+1 \le j \le 2n+2 \end{aligned}\right\} 1 \le r \le n+1$$

$$\left.\begin{aligned} SPP_1(j) &= VPP_1(j) \\ SPP_{r+1}(j) &= \left(\sum_{i=1}^{j-1} G_{r+1}(i) \prod_{l=i+1}^{j-1} P_{r+1}(l)\right) \oplus P_{r+1}(j) \\ P_{r+1}(j) &= SPP_r(j) \oplus VPP_{r+1}(j) \\ G_{r+1}(j) &= SPP_r(j) VPP_{r+1}(j) \end{aligned}\right\} \begin{aligned} &1 \le j \le 2n+2 \\ &1 \le r \le n \end{aligned}$$

$$PP(j) = SPP_{r+1}(j) \qquad 1 \le j \le 2n+2$$

SEL1. (multiplexer)

The product of two mantissas (as integer product) is a vector with MSB always in position 2n+1 or 2n+2.

Not all these bits are useful for the product output mantissa computation, so H is extracted from PP depending on the following two configurations of PP:

cf2 ≡ PP(2n+2)=1 that implies H(0,n)=PP(n+1,2n+1)

cf1 ≡ PP(2n+2)=0 and PP(2n+1)=1 that implies H(0,n)=PP(n,2n)

These are the only two possible configurations (from eq.1).

The not-rounded output mantissa is selected depending on the integer product output configuration.

In cf1 configuration PP(2n+1)=1 so this bit is not passed to H word (the most significant mantissa bit is always omitted in the representation). The final H equation is:

$$H(j) = PP(n+j)\overline{PP}(2n+2) + PP(n+j+1)PP(2n+2) \qquad 0 \le j \le n$$

INC1. (incrementer)

The rounding of output mantissa is performed on the base of H(0) bit:

$$\underline{DO}(1,n) = \begin{cases} \underline{H}(1,n) & H(0)=0 \\ increment(\underline{H}(1,n)) & H(0)=1 \end{cases}$$

where the *increment* function adds 1 to H(1,n) considered as an integer.

By means of the rounding bit (least significant of H) the remaining part of the H word is incremented or not. The rounded mantissa (output mantissa) is so obtained.

$$DO(j) = \left(\prod_{i=0}^{j-1} H(i)\right) \oplus H(j) \qquad 1 \le j \le n$$

EICM. (exponent increment control module)

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 have to be incremented by 1.

It is possible that even if the input mantissa product is *cf1*, the rounding operation, by means of a carry generation, 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 the output exponent increment.

- If PP(2n+2)=0 and PP(j)=1 with n≤j≤2n (PP(2n+1)=1 necessarily) then E_SH_CTR = 1:

  the *cf1* -> *cf2* transition determines the output exponent increment.

- If PP(2n+2)=0 and ∃j / PP(j)=0 with n≤j≤2n then: E_SH_CTR = 0 :

  the configuration *cf1* does not change the output exponent sum.

The E_SH_CTR equation is:

$$E\_SH\_CTR = PP(2n+2) + \prod_{j=n}^{2n} PP(j)$$

ADD1. (adder)

The two m-bit exponents are added together obtaining an (m+1)-bit result (EQ word): the integer result equals 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 bias.

The adder is again a carry look ahead adder:

$$EQ(m+1) = \sum_{i=1}^{m} G(i) \prod_{r=i+1}^{m+1} P(r)$$

$$\left.\begin{aligned} EQ(j) &= \left(\sum_{i=1}^{j-1} G(i) \prod_{r=i+1}^{j-1} P(r) \right) \oplus P(j) \\ P(j) &= EA(j) \oplus EB(j) \\ G(j) &= EA(j)\ EB(j) \end{aligned}\right] \qquad 1 \le j \le m$$

INC2. (incrementer)

Depending on E_SH_CTR value, EQ is incremented or not: the result is an (m+1)-bit word (the maximum possible value for EQ is 11...10).

$$ER(j) = \left(E\_SH\_CTR \prod_{i=1}^{j-1} EQ(i) \right) \oplus EQ(j) \qquad 1 \le j \le m+1$$

BFS. (bias factor subtractor)

The bias $2^{(m-1)}-1$ is subtracted from EQ to obtain ER:

$$EU(m) = \overline{\left(\prod_{i=1}^{m-1} ER(i) \right) \oplus ER(m)}$$

$$EU(j) = \left(\prod_{i=1}^{j-1} ER(i) \right) \oplus ER(j) \qquad 2 \le j \le m-1$$

$$EU(1) = \overline{ER}(1)$$

OM. (output module)

In absence of exceptions the output exponent EQ is set to ER. If one of the two inputs is q0 (exponent '0'), and the other is different from ∞ ('1'), the output value is q0.

The ∞ 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 $\geq 2\left(2^{m-1}-1\right)+2^{m-1}$ the output exponent is set to '1' (OM module).

$$EO(j) = EU(j)ER(m+1)\overline{SUFW} + EU(j)ER(m)\overline{SUFW} +$$
$$EU(j)\ \overline{SUFW}\ \overline{ER(m)}\ \overline{ER(m+1)}\ \prod_{i=1}^{m-1} ER(i) +$$
$$ER(m+1)\prod_{i=2}^{m-1} ER(i) + ER(m)\ ER(m+1) + SOFW \qquad 1 \leq j \leq m$$

ME. (exception early detection module)

The following first two lines detect respectively at least one q0 or one ∞ input while the third detects input couples q0·∞ or ∞·q0.

$$SUFW = \prod_{i=1}^{m} \overline{EA}(j) + \prod_{i=1}^{m} \overline{EB}(j) \qquad \text{(Q0 on an input operand)}$$

$$SOFW = \prod_{i=1}^{m} EA(j) + \prod_{i=1}^{m} EB(j) \qquad \text{(overflow on an input operand)}$$

$$NAN = \prod_{i=1}^{m} \overline{EA}(j)\ \prod_{i=1}^{m} EB(j) + \prod_{i=1}^{m} EA(j) \prod_{i=1}^{m} \overline{EB}(j) \qquad \text{(not a number)}$$

UOM. (underflow-overflow module)

The detection of u0 is based directly on ER analysis if q0 is not detected and flagged in input by SUFW.

One of the two inputs may be ∞; in this case if the other operand is not q0, then OFW is set. This situation is detected by ME module and flagged by the SOFW line to UOM. When the product ∞·q0 is detected NAN forces 1 both to UFW and OFW line.

$$UFW = \overline{ER}(m+1)\overline{ER}(m)\overline{SUFW}\left(\sum_{i=1}^{m-1} \overline{ER}(i)\right) + NAN \qquad \text{(underflow bit)}$$

$$OFW = ER(m+1)\prod_{i=2}^{m-1} ER(i) + ER(m+1)ER(m) + SOFW + NAN \qquad \text{(overflow bit)}$$

OS. (sign determination module)

The output sign is determined simply as EXOR of the operand signs.

$$SG\_OUT = SG\_A \oplus SG\_B$$

4. Adder Functionalities and Boolean Equations

As for the multiplier after a generic discussion, the equations are detailed.

The addition of equal sign floating point operands in the format of fig.2 is explained first. Called **X** and **Y** the two operands, their addition satisfies the following two constraints. The first is obtained when the mantissa of one of the two operands (e.g. **X**) DA(j)=0 with $0 \le j \le n$ and its exponent q is at least equal to r+n+2 where r is **Y** exponent. In this case DB, the second operand mantissa, is shifted away when adjusting exponents:

$$X+Y=2^{q}\left(1+\sum_{j=1}^{n}2^{-n+j-1}DA_{j}\right)+2^{r}\left(1+\sum_{j=1}^{n}2^{-n+j-1}DB_{j}\right)\ge 2^{q} \qquad (2)$$

The second constraint is obtained when DA(j)=DB(j)=1 with $0 \le j \le n$ and q=r:

$$X+Y\le 2^{q}\left(1+\sum_{j=1}^{n}2^{-n+j-1}+1+\sum_{j=1}^{n}2^{-n+j-1}\right)=2^{q+2}\left(1-2^{-(n+1)}\right)<2^{q+2} \qquad (3)$$

If the input operand signs are opposite, a subtraction of the module of input operands is performed: the mantissa of the less exponent input operand, is shifted and subtracted from the other.

tab.3. Positional reference.

Relative weight	2^{0}	2^{-1}	.	.	.	.	.	.	2^{-n}	2^{-n-1}	.	.	2^{-n-k}	.	.	.	2^{n-q}
Position	n+1	.	.	.	.	.	.	.	1	0	.	.	-k+1	-k	.	.	-q+1
A	1	X	X	.	.	.	.	.	X	0	.	.	0	0	.	.	0
B	1	X	X	.	.	.	.	.	X	0	.	.	0	0	.	.	0
C	0	0	0	.	0	1	X	.	X	X	.	.	X	X	.	.	X

The result of the subtraction presents more than two simple configurations with respect to the equal sign addition.

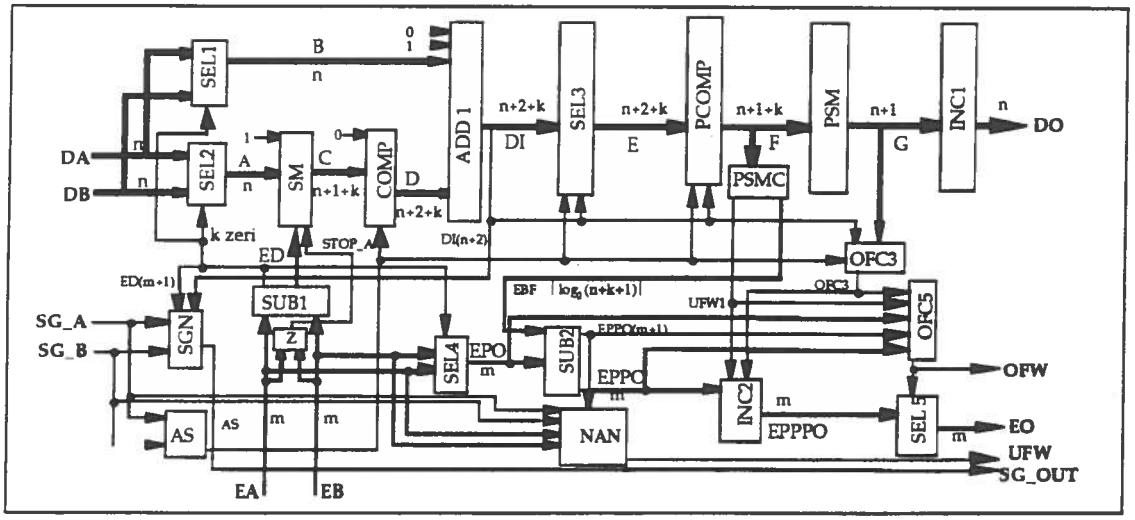

fig.4. Floating point adder scheme.

In general it is necessary to normalize mantissa and to balance exponents. Moreover in the addition of opposite sign operands, problems may arise when the truncation of the shifted mantissa influences the rounding operation [2].
In the following the positional reference defined in the tab.3 is used: as a consequence, once defined positions and weights for all bits of B, positions and weights of other word bits are referred to that of B.

SUB1. (input exponent subtracter)

The difference ED of the two input exponents EA and EB is computed (CLA subtracter). The two operands are unsigned m-bit operands, and so ED(m+1) holds the subtraction result sign.

$$\left.\begin{aligned} ED(m+1) &= \overline{\prod_{i=1}^{m} P(i) + \sum_{i=1}^{m} G(i) \prod_{r=i+1}^{m} P(r)} \\ ED(j) &= \left(\prod_{i=1}^{j-1} P(i) + \sum_{i=1}^{j-1} G(i) \prod_{r=i+1}^{j-1} P(r) \right) \oplus P(j) \\ P(j) &= \overline{EA(j) \oplus EB(j)} \\ G(j) &= EA(j)\overline{EB}(j) \end{aligned}\right] \quad 1 \le j \le m$$

SEL1. (selector)

Depending on the sign of EA and EB input exponents difference ED, the two input mantissas DA and DB are distributed on B and A word: the B word contains the mantissa of greater exponent input that is not shifted to match exponents. ED(m+1) is used as control for the selector.

$$B(n+1) = 1$$
$$B(j) = DA(j)\overline{ED}(m+1) + DB(j)ED(m+1) \qquad 1 \le j \le n$$

SEL2. (selector)

A contains the mantissa word of the less exponent operand.

$$A(n+1) = 1$$
$$A(j) = DA(j)ED(m+1) + DB(j)\overline{ED}(m+1) \qquad 1 \le j \le n$$
$$A(j) = 0 \qquad -k+1 \le j \le 0$$

Z.

If one of the two operands (**X**) is q0, the shift module (MS) erases its mantissa, to let the other operand mantissa to pass unchanged in the adder.

$$STOP\_A = \prod_{i=1}^{m} \overline{EA}(i) + \prod_{i=1}^{m} \overline{EB}(i)$$

SM. (shift module)

C word contains the A word left shifted a number of places that equals ED word. Only k-bit of the shifted word are saved in C, the other are lost: it is possible to determine optimal values for k to avoid or limit rounding error propagation[8].

$$C(n-j)=\overline{STOP\_A}\left(\sum_{i=0}^{j+1} ED^{i}A(n-j+i)+\sum_{i=1}^{j+1} ED^{2^{m+1}-i}A(n-j+i)\right) \qquad -1\le j\le n+k-1$$

AS. (addition-subtraction selector)

AS commutes the floating operator in adder or subtracter of the absolute value of the input operands depending on their signs.

$$AS=SG\_A\oplus SG\_B$$

COMP. (two's complement operator module)

If the operand signs are opposite, the less exponent operand is always shifted and complemented (D word). So the ADD1 module (adder) performs two's complement subtraction.

$$D(n+2)=AS\sum_{i=-k+1}^{n+1}C(i)$$

$$D(j)=\left(AS\prod_{i=-k+1}^{j-1}\overline{C}(i)\right)\oplus\left(C(j)\overline{AS}+\overline{C}(j)AS\right) \qquad -k+1\le j\le n+1$$

ADD1. (mantissa adder)

The adder performs the addition of B and D words yielding DI: if the operand signs are opposite the result of addition may be a positive number or a negative two's complement number. The sign of the result is determined by DI(n+2).

If the operand signs are equal, DI addition result may be in one of two possible configurations (from (2) and (3)). The configurations are shown in fig.5.

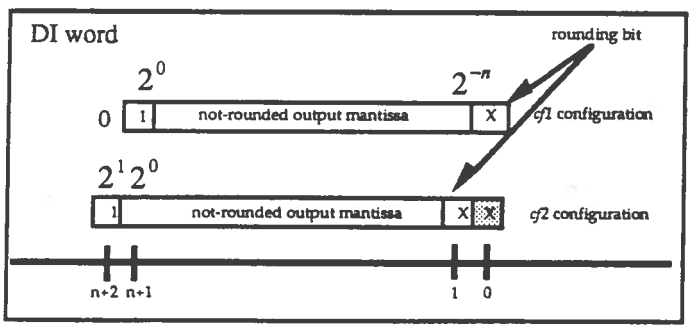

fig.5. Possible configurations in the addition of equal sign operands.

The positions and weights in the figure are set according to positional reference of tab.3.

If the signs of the operand are opposite DI(n+2) is a sign bit for the two's complement subtraction performed (B operand is always positive while D may be negative and represented as a two's complement number).

The adder equations are:

$$DI(n+2)=\left(\prod_{i=1}^{n+1}P(i)\right)\oplus D(n+2)$$

$$\left.\begin{aligned}DI(j)&=\left(\sum_{i=1}^{j-1}G(i)\prod_{r=i+1}^{j-1}P(r)\right)\oplus P(j)\\P(j)&=B(j)\oplus D(j)\\G(j)&=B(j)D(j)\end{aligned}\right\}\quad 1\le j\le n+1$$

SEL3. (selector)

If the input operand sign are opposite (AS=1), E word is made of DI word and a part of D word. No configuration shifts are performed in this case (E(1,n+1)=DI(1,n+1);E(-k,0)=DI(-k,0) since the result is eventually shifted in the following.

When the input operands signs are equal (AS=0), SEL3 sets E to different words depending on possible adder output configurations.

$$E(j)=DI(j)AS+DI(j)\overline{DI}(n+2)\overline{ED}(m+1)+DI(j+1)DI(n+2)\overline{AS}\qquad 1\le j\le n+1$$

$$E(0)=D(0)AS+D(0)\overline{DI}(n+2)\overline{ED}(m+1)+DI(1)DI(n+2)\overline{AS}$$

$$E(j)=D(j)AS+D(j)\overline{DI}(n+2)\overline{ED}(m+1)+D(j+1)DI(n+2)\overline{AS}\qquad -k+1\le j\le -1$$

PCOMP. (two's complement operator module)

F word holds the two's complement of DI word if DI(n+2)=1 (negative addition result) and AS=1 (opposite sign input operands). Otherwise F(-k+1,n+1)=DI(-k+1,n+1).

$$F(j)=\left(DI(n+2)AS\prod_{i=-k+1}^{j-1}\overline{E}(i)\right)\oplus\left(\overline{E}(j)DI(n+2)AS+E(j)\overline{AS}+E(j)\overline{DI(n+2)}\right)\qquad -k+1\le j\le n+1$$

PSM. (normalization module)

The normalization of F is performed left shifting F a number of places equal to the number of most significant zeros. At the end the first not null bit on the right (if any) is discarded, while the other fills G(0,n).

$$G(n-j)=\sum_{i=-1}^{n+k-j-2}\left(\prod_{r=0}^{i}\overline{F}(n+1-r)\right)F(n-i)F(n-i-j-1)\qquad 0\le j\le n$$

INC1. (incrementer)

This module performs the rounding of G word to match final output format of DO. The rule applied for rounding is the same of module INC1 of the multiplier.

$$DO(j)=\left(\prod_{i=0}^{j-1}G(i)\right)\oplus G(j)\qquad 1\le j\le n$$

SEL4. (selector)

EPO holds the greater input exponent determined, subtracting EB from EA, by ED(m+1).

$$EPO(j) = EA(j)\overline{ED}(m+1) + EB(j)ED(m+1) \qquad 1 \le j \le m$$

PSMC. (F=0 detection and exponent balancing control determination)

UFW1 detects a null F word.

$$UFW1 = \prod_{i=-k+1}^{n+1} \overline{F}(i)$$

EBF is an m-bit word where only the least significant $\lceil \log_2(n+k+1) \rceil$ bits differ from 0 (assumed m>$\lceil \log_2(n+k+1) \rceil$). This word computes the number of F leading zeros when F(-k+1,n+1)≠0 while EBF=0 when F(-k+1,n+1)=0. This word is used to adjust exponent after normalization.

$$EBF(p) = \sum_{l=0}^{\left\lfloor \frac{n+k+1}{2^{p+1}} \right\rfloor - 1} \left\{ \left[\prod_{r=0}^{l} \prod_{u=0}^{2^p-1} \overline{F}\left(n+1-2^{p+1}r-u\right) \right] \sum_{u=0}^{2^p-1} F\left(n+1-2^p-2^{p+1}l-u\right) \right\} +$$

$$\left[\prod_{r=0}^{\left\lfloor \frac{n+k+1}{2^{p+1}} \right\rfloor} \prod_{u=0}^{2^p-1} \overline{F}\left(n+1-2^{p+1}r-u\right) \right] \sum_{u=0}^{n+k-2^{p+1}\left\lfloor \frac{n+k+1}{2^{p+1}} \right\rfloor - 2^p} F\left(n+1-2^p-2^{p+1}\left\lfloor \frac{n+k+1}{2^{p+1}} \right\rfloor - u\right)$$

$$0 \le p \le \lceil \log_2(n+k+1) \rceil - 1$$

$$EBF(p) = 0 \qquad \lceil \log_2(n+k+1) \rceil \le p \le m$$

SUB2. (exponent balancing)

The difference of EPO and EBF is computed to balance normalization on F yielding EPPO.

$$EPPO(m+1) = \overline{\prod_{i=1}^{m} P(i) + \sum_{i=1}^{m} G(i) \prod_{k=i+1}^{m} P(k)}$$

$$\left.\begin{aligned} EPPO(j) &= \left(\prod_{i=1}^{j-1} P(i) + \sum_{i=1}^{j-1} G(i) \prod_{k=i+1}^{j-1} P(k) \right) \oplus P(j) \\ P(j) &= \overline{EPO(j) \oplus EBF(j)} \\ G(j) &= \overline{EPO}(j) EBF(j) \end{aligned}\right\} \quad 1 \le j \le m$$

OFC3. (configuration transition detector)

This bit detects *cf1* → *cf2* transition induced by rounding operation on G when the input operand signs agree.

$$OFC3 = DI(n+2)\ \overline{AS} + \prod_{i=0}^{n} G(i)$$

INC2. (incrementer)

When the following conditions arise together:

- OFC3 bit is high (there is a *cf1* → *cf2* transition),
- F(-k+1,n+1) is not null,

EPPO is incremented.

$$EPPPO(j) = \left(OFC3\ \overline{UFW1}\ \overline{EPPO(m+1)} \prod_{l=1}^{j-1} EPPO(l) \right) \oplus \overline{UFW1}\ \overline{EPPO(m+1)}\ EPPO(j) \qquad 1 \le j \le m$$

OFC5. (overflow module)

OFC5 (also OFW bit) is set when one of the input operands is ∞ (EPO=1) or when the following conditions arise together:

- OFC3 bit is high (there is a *cf1* → *cf2* transition),
- F(-k+1,n+1) is not null,
- EPPO=011...10

$$OFC5 = OFC3\ \overline{UFW1}\ \overline{EPPO(m+1)}\ \overline{EPPO(1)} \prod_{i=2}^{m} EPPO(i) + \prod_{i=1}^{m} EPO(i)$$

$$OFW = OFC5$$

SEL5. (selector)

The output exponent is chosen between EPPPO (OFC5=0) and 1 (OFC5=1; overflow is set);

$$EO(j) = EPPPO(j) + OFC5 \qquad 1 \le j \le m$$

SGN. (operation result sign determination)

This module determines the sign of the addition result. If the operands have equal signs, the sign of the result is simply the sign of one of the two operands. Otherwise the result sign depends on input signs, sign of the input exponent difference (ED(m+1)) and sign of mantissa subtraction (DI(n+2)).

$$\begin{aligned} SG\_OUT = {} & SG\_A\ \overline{SG\_B}\ \overline{ED(m+1)}\ \overline{DI(n+2)} + \\ & SG\_A\ \overline{SG\_B}\ ED(m+1)\ DI(n+2) + \\ & \overline{SG\_A}\ SG\_B\ \overline{ED(m+1)}\ DI(n+2) + \\ & \overline{SG\_A}\ SG\_B\ ED(m+1)\ \overline{DI(n+2)} + \\ & SG\_A\ SG\_B \end{aligned}$$

NAN. (underflow module)

If the exponent balancing (EPO-EBF) induced by normalization generates a negative EPPO (EPPO(m+1)=1), the u0 result is generated and flagged (UFW=1). UFW (and

OFW too) is set also if input operand sign are opposite and the input operands are ∞ (EA=1, EB=1).

$$UFW = EPPO(m+1) + (SG\_A \oplus SG\_B)\prod_{i=1}^{m} EA(i)\prod_{i=1}^{m} EB(i)$$

5. Conclusions.

The study of a parametric set of operators is suited, in the design phase of a project, to extablish the very computational needs for a given application (by means of slow but design compliant simulation package). Once the design parameters have been project tuned it is possible to think about implementation of the set with available technology: Gate Arrays (for small n, m, k parameter values), or ASIC when the parameter size becomes too wide.

Considering exact input operands, the relative error induced in multiplication or addition operations with respect to the exact one is at most a value about $2^{-(n+1)}$. This becomes the design correctness constraint. To evaluate equation correctness in particular frames software tools have been used: a C program simulation was written taking into account boolean functionality description. Boolean sets have been represented with characters while boolean operators were replaced by logic and bitwise character C operators.

The three design parameters n, m and k were fixed for each simulation. Input streams were generated using ANSI C standard library random number generation function.

Test operation results were computed for each input and compared with those computed in MC68882 extended precision format[7] (n=68 and m=15). For coherence reasons n, m parameters were kept slightly less than MC68882 correspondent one (n<25, m=8). A value of k=3 has been chosen: this value does not allow the complete elimination of rounding error[2] for the tests done.

At the end the reduction of boolean functionalities to the target technology may require complex fitting algorithms to decompose equations. In general the decomposition is not possible with equations in their general parametric form. So the above description of operators may be used as an intermediate level between design functional description and final technological implementation.

Acknowledgements.

I am grateful to dr. G. Sechi of IFCTR-CNR (Milan) for his interest and guidance throughout this work and the particular care given in all discussions.

References.

[1] BARTOLUCCI M., GUAZZONI P., MANFREDI G., SECHI G., ZETTA L.: "A Set of Parametric Operators, Algebraically Defined for Real Time Data Processing", EUROMICRO 94: SYSTEM ARCHITECTURE AND INTEGRATION, Liverpool September 5-8, 1994, published on the conference proceedings.

[2] BARTOLUCCI M., SECHI G.: "Rounding Error in the Computation of Opposite Sign Floating Point Number Parametric Addition: a Case Study", short note session of EUROMICRO 94: SYSTEM ARCHITECTURE AND INTEGRATION, Liverpool September 5-8, 1994, to be published on a special issue of Microprocessing and Microprogramming.

[3] UNGER STEPHEN, "Three Realization of Iterative Circuits", IEEE Transaction on Computers, Vol. C-26, No. 4, APRIL 1997, 365-383

[4] TAKAGI N.,YASUURA H.,YAJIMA S.: "High Speed VLSI Multiplication Algorithm with a Redundant Binary Addition Tree", IEEE, Trans. on Comp., September 1985, 789-796

[5] HAO -YUNG L.: "High Speed Signed Digit Multipliers for VLSI", Microprocessing and Microprogramming, 29, 1990, 205-215

[6] AVEZZINIS A.: "Signed-Digit Number Representation for Fast Parallel Arithmetic",IRE TRansaction on Electronic Computers, 1961, 389-400

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