

Fast Arithmetic and Fault Tolerance in the FERMI System

Luca Breveglieri Luigi Dadda Vincenzo Piuri

Dipartimento di Elettronica e Informazione - Politecnico di Milano
Piazza Leonardo Da Vinci 32 - I-20133 Milano, Italy
fax: +39 2 2399 3411 – e-mail: breveglieri,dadda,piuri@elet.polimi.it

Abstract

The FERMI [3,4,5] is a data acquisition system for calorimetry experiments in high energy physics at the LHC, CERN. The system contains a large number of acquisition channels, with a precision of 16 bits and a sampling rate of 40 MHz. A large part of the information driven by the channels is processed locally, to reduce the amount of data. This requires to cluster several channels by adding them. The paper presents the design of a fast, low cost adder chip, based on the implementation of column compression techniques [2,7,8] for the computation of integer addition. Since the system is operating in a radiation-hard environment [6,10], fault tolerance (namely fault detection) is implemented by means of arithmetic codes [1,9].

1. Introduction

The FERMI [3,4,5] is a data acquisition system for calorimetry experiments in high energy collision experiments planned at the Large Hadron Collider (LHC) of CERN (Geneva, Switzerland) in the future. The transmission and elaboration of information in the system is done by means of VLSI technologies. The waveforms generated by the sensors placed inside the calorimeter are immediately digitized by ADC's; the following segments of the acquisition channels are completely digital. The samples are integer numbers, represented in two's complement. The system contains a high number of acquisition channels for integer data, from about 10^5 channels for the largest experiments down to some hundreds for the smallest ones, with a precision of 16 bits and a sampling rate of 40 MHz. All the information supplied by the channels is preprocessed locally, to reduce drastically the amount of data; several reduction stages are present. At every reduction stage a trigger may be generated, to indicate the presence of significant events. Processing is then completed remotely, but only on triggered data. Since most neighbouring channels are highly correlated, the reduction process consists mainly in clustering several channels into a single data flow; this is accomplished by adding the channels. From physical considerations, it proves that clustering should be performed for groups of 9 channels. Therefore a large number of VLSI adder devices are integrated in the system. Such adder devices should operate at the same clock frequency as the acquisition channels (40 Mhz), should be made as small as possible, since they will be eventually mounted together with other components of FERMI onto a silicon substrate by means of a flip-chip technology, and should possess fault detection and/or fault tolerance capabilities [1,9,10] since the operating environment of most components of FERMI is radiation-hard [6].

An "Adder Chip" has already been designed and fabricated as a single chip (version I, [3,5]), and is currently being used for testing purposes. However, this adder chip is implemented by means of traditional techniques, and has the structure of a tree of fast (lookahead) adders. This

implies that its cost, measured in terms of silicon area, is considerable. Moreover, it has neither fault detection nor fault tolerance capabilities. The summands and the sum are two's complement integer numbers. The adder chip has 9 inputs of 16 bits and produces their sum as a single output of 16 bits, plus an overflow signal.

In this paper the Adder Chip is completely redesigned (version II), by extensively using column compression techniques [2,8] for the computation of addition. Such techniques are normally used for parallel multiplication [7], and require some adaptation to work with addition, especially because the treatment of sign is different.

The paper is organized as follows. Section 1 summarizes briefly Adder Chip version I. Section 2 presents the architecture of Adder Chip version II. Section 3 shows how to implement efficiently fault detection in the Adder Chip. Section 4 presents a comparison between Adder Chip versions I and II.

2. The Adder Chip version I

The Adder Chip version I has been designed by a grouping on a single chip some preexisting functional blocks, that had been previously implemented and verified on different chips. Figure 1 shows a block schematic of the chip. In particular:

- The addition tree is formed at the first level by three blocks that add together three channels (that had been previously implemented in the "Channel Chip" of the FERMI, version I, [3,4,5]). Each block outputs the partial sum of a group of three channels, for a total of three partial sums, and produces the overflow detection signal of the sum of three channels, for a total of three partial overflow signals.
- The addition tree is formed at the second level by a functional block that adds together the three sums computed in the first level of the tree. It outputs the total sum of the nine channels, and produces the overflow detection signal of this sum.
- The four partial overflow signals, generated at the various levels, are put together by an OR gate, to generate a single global overflow signal.
- The addition tree is pipelined, by introducing pipeline registers in the appropriate points, to shorten the propagation delays (at the price of some clock cycles latency).

However, it may not be an optimal solution to group onto a single chip functional blocks that were previously implemented onto different chips. The presence of 2_to_1 adders inside the addition tree, at the junction between the first and the second level of the tree, is contradictory, as in the second level of the tree the addition process goes on again by using 3_to_2 adders. It is known that 2_to_1 adders are much larger and/or slower than 3_to_2 adders and that the only real need to perform a 2_to_1 addition is at the end of the summation process. The computation of three partial overflow signals as early as in the first level of the addition tree is useless if not disadvantageous, as they are soon after ORed together. Such partial overflow signals would allow to locate the position of the overflow event in the three groups of the nine channels, but this location information is actually not used by the adder chip, for in case of overflow the sum is unusable and, in any case, the nine inputs are also stored individually in other parts of the FERMI system.

Therefore is advisable to redesign the addition tree of the Adder Chip, eliminating or reducing the above defects. In the following we show an optimized solution. Such optimization should provide the following improvements:

- Silicon area reduction, due to the replacement of 2_to_1 by 3_to_2 adders.

- Time speed up, due to the elimination of 2_to_1 adders. In fact:
- a 2_to_1 adder must be designed either as a ripple or as a carry lookahead architecture;
- if designed as ripple carry, it is far slower than a 3_to_2 adder, because it contains a carry propagation chain;
- if designed as carry lookahead, it certainly consumes a silicon area larger than that of a 3_to_2 adder, due to the presence of the lookahead circuit, and thus in any case it is slower than a 3_to_2 adder.

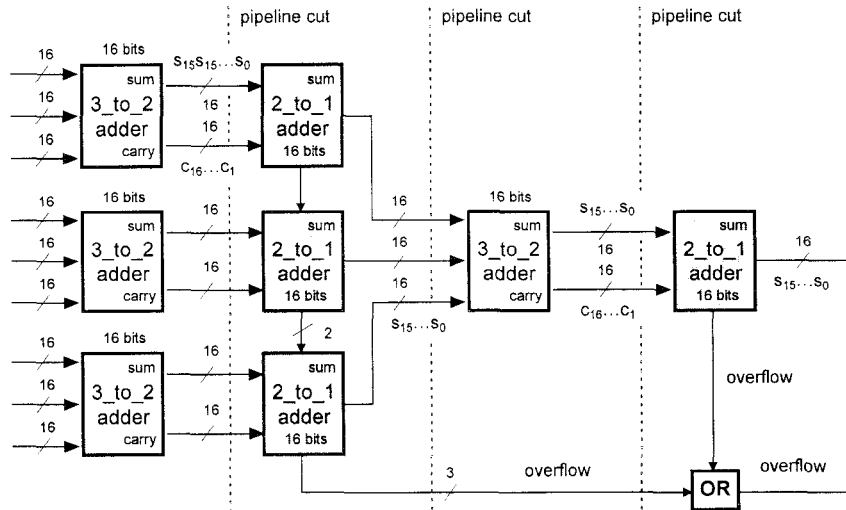


Figure 1: Structure of the Adder Chip version I.

The above requirements suggest to redesign the Adder Chip by using the well known column compression (CC) technique for the computation of binary addition [7,8]. This technique has been widely adopted in the past for the design of parallel multipliers. Applications to addition are less common and require a different way to deal with the sign, because the disposition of the bits in an addition matrix of bits is fairly different from that of the bits in a partial product matrix of bits deriving from a multiplication.

3. The Adder Chip version II

We report extensively the functional requirements for the Adder Chip version II:

- The chip has 9 inputs of 16 bits and one output of 16 bits, plus the overflow.
- The inputs are two's complement integers.
- The output is a two's complement integer.
- The input and output range interval is $[-2^{15}, 2^{15} - 1]$.
- The overflow is generated if and only if the sum of the nine inputs exceeds the output range, i.e. it requires over 16 bits to be correctly represented as a two's complement integer number.
- The correct result requires $16 + \lceil \log_2 9 \rceil = 16 + 4 = 20$ bits to be represented correctly, and is actually computed completely.

- Only the 16 least significant bits of the real value over 20 bits of the sum are retained, i.e. the two's complement integer number output by the circuit is actually meaningless whenever an overflow occurs.

The architecture of the Adder Chip version II is designed according to the column compression technique. The nine rows of bits representing the inputs are grouped together to form a rectangular matrix, which is then reduced by applying carry save addition in stages. Figures 2a and 2b show the reduction process and the partitioning into stages. The shaded areas put into evidence the grouping of the rows for the addition, and the dash traits show the propagation of the carries. In the last row the conditions for the detection of overflow are sketched. Figures 2a and 2b refer to the addition of 9 two's complement integers of 6 bits, but the example can be easily generalized to integers of 16 bits, without substantial modifications.

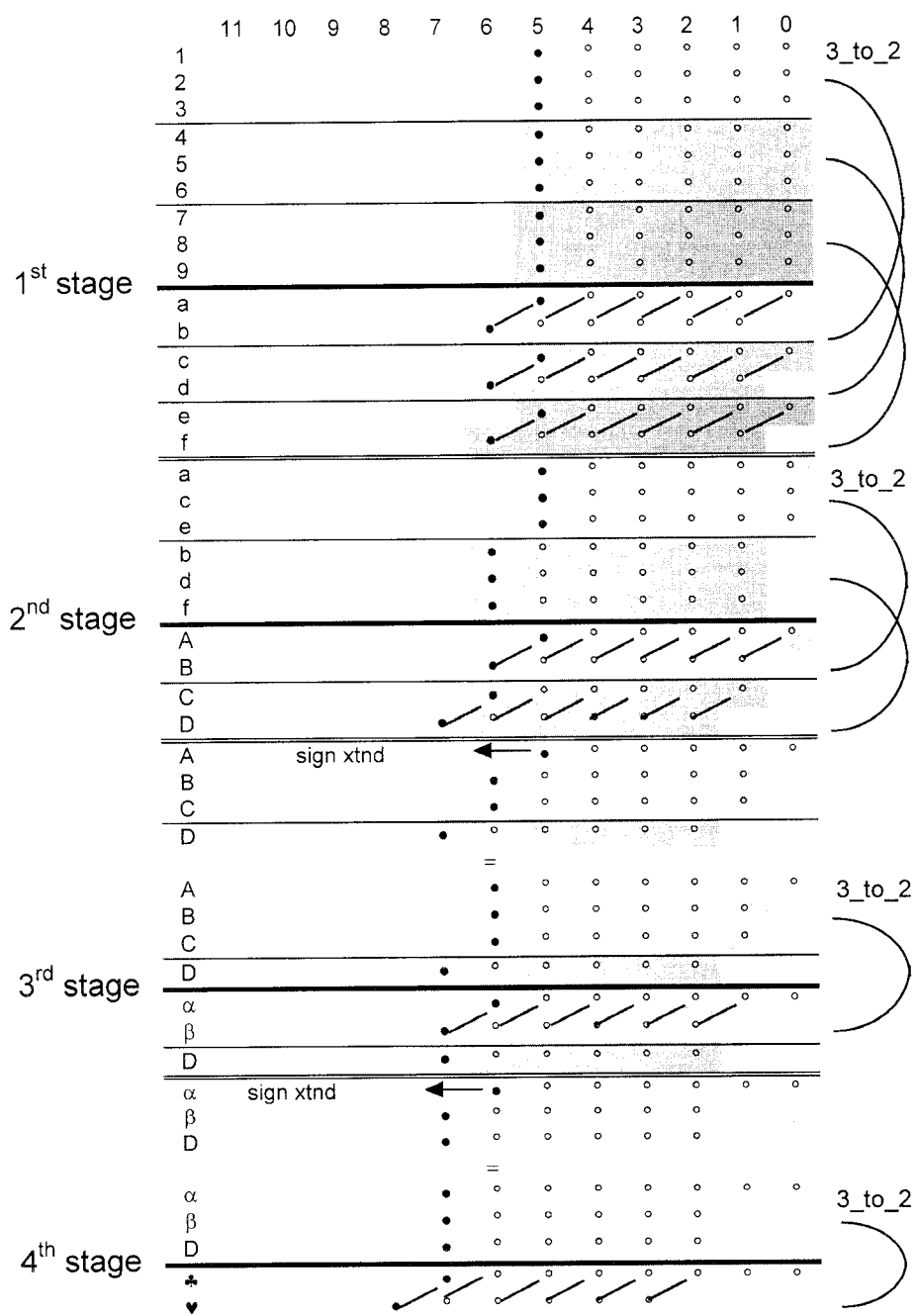
The symbol $\circ$ represents a generic bit of positive weight. The symbol $\bullet$ represents a generic bit of negative weight; in the tables above it is always used as the sign bit (the most significant one) of two's complement integer numbers. The symbol "of" represents the overflow of the two's complement integer sum Ω of the two two's complement integer numbers $\clubsuit$ and $\heartsuit$; it is not part of Ω . The symbol $\leftarrow \bullet$ represents a sign extension: the sign bit $\bullet$ is extended one position to the left, i.e. towards the consecutive most significant position. The extended sign bit is simply a duplication of the original sign bit. The extended bit has a negative weight, so it is the new sign bit, whereas the original sign bit loses its original negative weight and acquires a positive weight.

The summands have 16 bits and the sum is truncated to 16 bits. The circuit computes all the 20 bits of the real value of the sum. Since the sum is represented in two's complement integer notation, to be truncated to 16 bits without overflow it would be necessary that the five most significant bits of the real value of the sum were a sign extension, i.e. they were either all 1 or all 0. Then it would be possible to eliminate the four most significant ones, just leaving the fifth bit (which is the 16-th one) as the sign bit of the sum truncated to 16 bits.

Actually the circuit shown in Figure 3 uses a slightly more complex overflow detection circuit, to avoid a defect of the very simple and natural overflow detection condition above discussed: the fact that it would require that the last summation in the addition tree (the summation performed by the 2_to_1 adder) be of 17 bits, which would make the structure of the addition tree irregular (see Figure 3). So the function detecting the overflow is different and is explained in the following.

The symbol s_i indicates a bit of weight i , i.e. the weighted value of s_i is $\pm s_i 2^i$, where $i \geq 0$. The bit s_i may have a negative weight or not, depending on whether it is a sign bit or not. The sequence $s_{15} \dots s_0$ indicates the bits of a two's complement integer number represented over 16 bits, where s_{15} is the most significant bit (also called sign bit) and s_0 is the least significant one.

The interpretation of a two's complement integer number is $s_{15} \dots s_0 = -s_{15} 2^{15} + \sum_{i=0}^{14} s_i 2^i$. The sequence $s_{15} s_{15} \dots s_1$ (and similarly for $s_{16} s_{16} \dots s_2$ and $s_{17} s_{17} \dots s_3$) indicates a two's complement integer number represented over 16 bits, formed by the 15 bits $s_{15} \dots s_1$ sign extended by duplicating the sign bit s_{15} in the most significant position. It represents the two's complement sign extension to 16 bits of the 15 bits two's complement integer number $s_{15} \dots s_1$.



The comparator checks the bits s_{15} s_{16} s_{17} and s_{18} : if they differ then an overflow occurs. Remember in fact that the bit s_{15} will become the sign bit of the result of 16 bits output by the addition tree. So the function of the comparator is to check whether the four bits s_{15} s_{16} s_{17} and s_{18} are either all 0 or all 1. In few words, the output “diff” of the comparator must assume the logical value 1 if and only if either $s_{15} s_{16} s_{17} s_{18} = 0 0 0 0$ or $s_{15} s_{16} s_{17} s_{18} = 1 1 1 1$. In all other situations the output “diff” of the comparator must assume the logical value 0.

To compute the global overflow, the following reasoning is made, using the summation table shown in Figures 2a and 2b. The table refers to two’s complement integer numbers of 6 bits, instead of 16, but the example is easily generalizable. Let us call SUM the complete value of the summation of the 9 channels of 6 bits, represented in two’s complement. The value SUM requires $6 + \lceil \log_2 9 \rceil = 6 + 4 = 10$ bits to be always correctly represented in two’s complement. From the table above it shows that the computation of SUM is eventually reduced to the summation of two integers of 9 bits, called ♣ and ♥, i.e. $SUM = \clubsuit + \heartsuit$. Such two integers are always correctly represented in two’s complement over 9 bits. If the summation of ♣ and ♥ overflows over 9 bits, i.e. it should require 10 bits for a correct representation in two’s complement, then obviously the summation of ♣ and ♥ overflows over 6 bits, too, i.e. the value SUM overflows over 6 bits, else the summation of ♣ and ♥ is correctly representable as an integer of 9 bits, i.e. the value SUM is correctly representable as an integer of 9 bits. Now, if the value SUM is a correct integer of 9 bits, for it to be representable as an integer of 6 bits the four most significant bits need be a sign extension, i.e. they must be either all 0 or all 1. Therefore, the global overflow is computable by ORing the overflow signal of the addition ♣ + ♥ with a signal that is activated if and only if the four most significant bits of the 9 bit summation of ♣ and ♥ are neither all 0 nor all 1. To generalize, just replace the integers of 6 bits with integers of 16 bits, so that SUM requires 20 bits and ♣ and ♥ are integers of 19 bits.

4. Fault tolerance

Most of the other arithmetic parts of FERMI have fault detection capabilities, but no error correction or fault masking. Therefore it is advisable to introduce fault detection also in the Adder Chip version II [1]. This can be done by using the residue code [1,7,8], in the base 3. Figure 4 shows the implementation of such a code.

The significant architectural problem to be solved is to design a circuit capable of computing the remainder of the integer division by 3 of signed integers of 16 bits. We recall that if N is a signed integer (dividend) and D is a non-null natural number (divisor), then integer division is defined as follows:

$$N = Q \times D + R \quad \text{with} \quad 0 \leq R < D$$

where the quotient Q and the remainder R are integers as well. Thus for instance $4 = 1 \times 3 + 1$, but $-4 = -2 \times 3 + 2$, because the remainder cannot be negative. It is then easy to see that two’s complement integers of 16 bits can be decomposed as follows:

$$\begin{aligned} S &= \text{value of the two's complement bit sequence } s_{15} \dots s_0 = -s_{15} 2^{15} + \sum_{i=0}^{14} s_i 2^i \\ S \bmod 3 &= (-s_{15} 2^{15} + s_{14} 2^{14} + s_{13} 2^{13} + \dots + s_2 2^2 + s_1 2 + s_0) \bmod 3 = \\ &= (s_{15} + s_{14} + s_{13} 2 + s_{12} + s_{11} 2 + \dots + s_3 2 + s_2 + s_1 2 + s_0) \bmod 3 = \\ &= ((s_{15} + s_{14} + s_{12} + \dots + s_2 + s_0) + 2(s_{13} + s_{11} + \dots + s_3 + s_1)) \bmod 3 \end{aligned}$$

The last member of this equality says that by putting the bits of S alternatively in two columns (with the only exception of the sign bit s_{15}), and then by computing the sum (possibly repeating this procedure if a result of more than two bits is obtained), the remainder is obtained. Note that sign bits disappear, so that all bits have a positive weight. Figure 5 shows the procedure, which is a “folding” of even and odd weighted bits of the operand S .

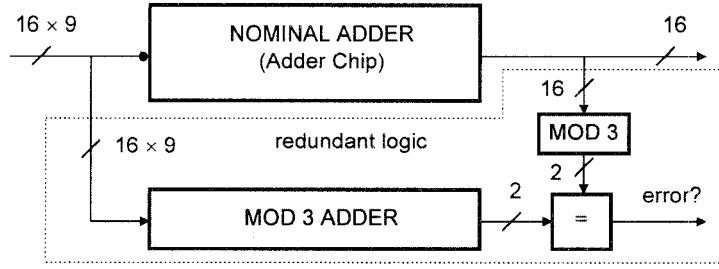


Figure 4: Implementation in the Adder Chip of the residue code in the base 3.

The residue in the base 3 should be represented over 2 bits. As a matter of fact, the remainder obtained in Figure 5 is still represented in redundant form over 6 bits, but is easily reducible by applying the same procedure.

The simplicity of the whole reduction procedure shown in Figure 5 depends on the fact that the residues in the base 3 of the powers of 2 cycle through the succession 1, 2, 1, 2, 1, 2, ..., whose terms are powers of 2 themselves. This allows to fold even and odd weighted columns of bits of the summation matrices. The bottom part of Figure 5 (below the equal symbols) shows the folding of the carries produced during the addition of the columns. But this property is more general, for we have the following statement.

Statement: Fixed $k \geq 1$ bits for the representation of the residue, the two following statements are equivalent for residue number systems associated with binary integer numbers (unsigned or two's complement).

1. The residue number system has maximum error detection capability.
2. Addition in the residue number system can be derived from ordinary addition by folding module k the bits of the summands.

Proof: As it is well known, fixing the number k of bits for the representation of the remainder, residue codes provide maximum error detection capability for all and only the bases of value $b = 2^k - 1$ (see [9]). Folding the bits of the binary integer summands (unsigned or two's complement) in the addition is equivalent to saying that the succession of the residues of the weights $\pm 2^i$, for $i = 0, 1, 2, \dots$, of the bits of the summands is a periodic succession of powers in the same base 2. We consider the case of positive weights. The first $k = \lceil \log_2 b \rceil$ terms of the succession of residues $r_i = 2^i \bmod b$, of index $i \geq 0$ and for a fixed base b , are the powers 2^i themselves, i.e. $r_i = 2^i$; moreover it always holds $r_0 = 1$.

To prove that (1) implies (2), note that dividing by b means that $b = 0$, hence if we pose $b = 2^k - 1$ it follows $2^k - 1 = 0$, which implies $2^k = 1$; therefore every k terms the succession of residues r_i restarts from 1, so the prefix of k powers 2^i is repeated. Conversely, to prove that (2) implies (1), note that equating $r_k = r_0$ means that $2^k \bmod b = 1$, hence it follows $2^k - 1 = 0 \bmod b$, which implies $b = 2^k - 1$; therefore the only base that makes the succession of residues r_i periodic of period k is $b = 2^k - 1$. Both implications hold also for bits of negative weight. ■

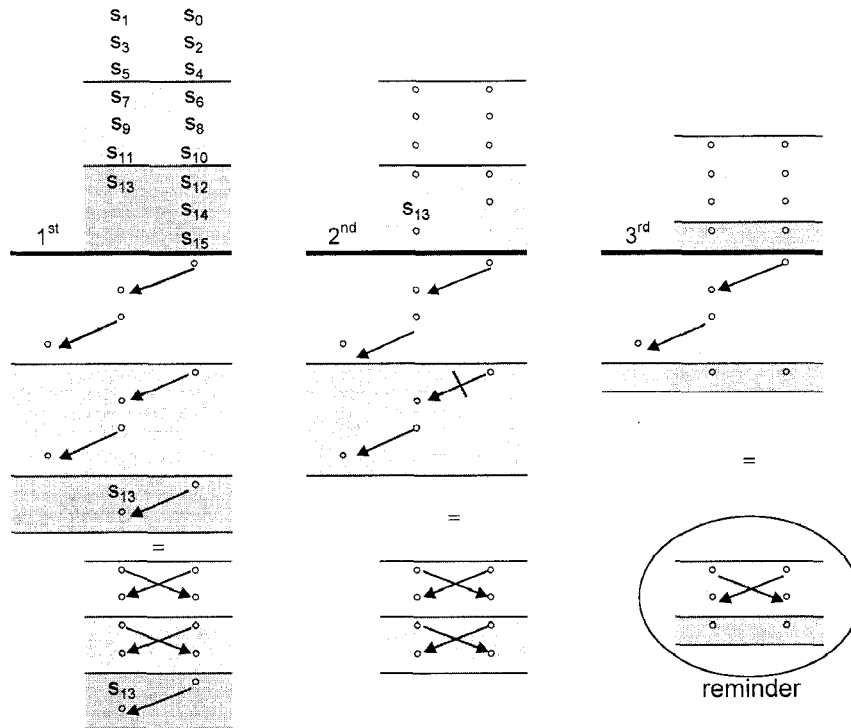


Figure 5: Reduction stages for the computation of the residue in the base 3.

This statement has a deep interpretation: the least complex and most natural way to compute addition in a residue number system by means of column compression techniques (i.e. folding the columns) is in one-to-one correspondence with the maximum error detection capability of the residue number system.

This column compression procedure for the computation of the residue in the base 3 is easily extensible to sums of several operands. Such a column compression procedure is similar to the one already used for the nominal part of the Adder Chip, and is actually simpler because sign bits can be dealt with in the same way as normal bits. So the same structure used for the nominal part of the Adder Chip can be used also for the computation of the remainder, reducing the number of bits from 16 to 2.

5. Evaluation of cost

As an approximate estimate, passing from version I to version II three 2_to_1 adders would be replaced by as many 3_to_2 adders, saving silicon area, power and time. An additional 3_to_2 adder is introduced, but it has a small delay impact, that is compensated by the replacement of 2_to_1 adders by faster 3_to_2 adders. In summary:

- Original addition tree: $3 + 1 = 4$ adders of type 3_to_2; $3 + 1 = 4$ adders of type 2_to_1; and critical path: 2 adders of type 3_to_2 plus 2 adders of type 2_to_1.

- Modified addition tree: $3 + 2 + 1 + 1 = 7$ adders of type 3_to_2; 1 adder of type 2_to_1; and critical path: 4 adders of type 3_to_2 plus 1 adder of type 2_to_1.

Therefore, the global number of adders is the same (8 adders with operands of 16 bits), but the modified solution uses fewer 2_to_1 adders than the original one. The critical path of the modified solution is slightly longer (5 against 4), but the number of 2_to_1 adders in the critical path is smaller (1 against 2); we recall that a 3_to_2 adder is much faster than a 2_to_1 adder. Moreover, to construct the modified solution it would not be necessary to redesign new functional blocks, but only to reorganize in a better way the existing ones. When an overflow in the whole summation occurs, the value of the sum could be dealt with in two possible ways: the least significant 16 bits of the real value of the sum are retained; or the most significant 16 bits of the real value of the sum are retained. In the latter case the retained value is meaningful, although some bits of precision are lost, but this solution requires to scale all operands to be included in the interval $[0, 1]$. Actually Figure 3 adopts the former solution, which is suggested by physical considerations.

6. Conclusions

The complete and detailed design of a new version for the FERMI Adder Chip has been developed. This version achieves a significant decrease of cost, measured in terms of silicon area, with respect to the previous one. This new version will be fabricated in the next prototyping cycle of the FERMI system. The procedure for the application of efficient CC techniques to the addition of (signed) integers can be easily generalized to any number of summands of any size.

References

- [1] A. Antola, L. Breveglieri, "Balancing of Fault Tolerance in the new Version of the FERMI Channel Chip: a functional Evaluation", Proc. of DFT '96, Boston, MA, U.S.A., IEEE Computer Society Press, pp. 249–257, November, 1996
- [2] L. Dadda, S. Inkinen, V. Piuri, "A Processor for Calorimetry at the Large Hadron Collider (LHC) in the FERMI Project", Proc. of ASAP '94, San Francisco, CA, U.S.A., IEEE Computer Society Press, pp. 188–199, August, 1994
- [3] The FERMI Collaboration, "FERMI: a digital Front End and Readout Microsystem for high Resolution Calorimetry", in *Transactions on Nuclear Instruments & Methods in Physics Research (NIM)*, North Holland, n° 357 sec. A, pp. 306–317, 1995
- [4] The FERMI Collaboration, "Optimised digital Feature Extraction in the FERMI Microsystem", in *Transactions on Nuclear Instruments & Methods in Physics Research (NIM)*, North Holland, n° 357 sec. A, pp. 318–328, 1995
- [5] The FERMI Collaboration, *Status Report on the FERMI Project*, Internal Report RD–16 CERN/LHCC 95-28, CERN, 1995
- [6] A. Kerek, "Front End Electronics in the Radiation Environment at LHC", Proc. of DFT '93, Venice, Italy, IEEE Computer Society Press, pp. 96–100, October, 1993
- [7] G. Jullien, W. Miller, Z. Wang, "A new Design Technique for Column Compression (CC) Multipliers", in *IEEE Transactions on Computers*, vol. 44 n° 8, August, 1995
- [8] K. Hwang, *Computer Arithmetic: Principles, Design and Tools*, John Wiley & Son, NY, 1974
- [9] W. W. Peterson, *Error correcting Codes*, John Wiley & Son, NY, U.S.A., 1961
- [10] D. P. Siewiorek, R. S. Swarz, *The Theory and the Practice of reliable Systems Design*, Digital Press, Bedford, U.S.A., 1982