Ultra-Low-Power Adder Stage Design for Exascale Floating Point Units

ALBERTO A. DEL BARRIO, Complutense University of Madrid
NADER BAGHERZADEH, Center for Pervasive Communication and Computing, University of California at Irvine
ROMÁN HERMIDA, Complutense University of Madrid

Currently, the most powerful supercomputers can provide tens of petaflops. Future many-core systems are estimated to provide an exaflop. However, the power budget limitation makes these machines still unfeasible and unaffordable. Floating Point Units (FPUs) are critical from both the power consumption and performance points of view of today's microprocessors and supercomputers. Literature offers very different designs. Some of them are focused on increasing performance no matter the penalty, and others on decreasing power at the expense of lower performance. In this article, we propose a novel approach for reducing the power of the FPU without degrading the rest of parameters. Concretely, this power reduction is also accompanied by an area reduction and a performance improvement. Hence, an overall energy gain will be produced. According to our experiments, our proposed unit consumes 17.5%, 23% and 16.5% less energy for single, double and quadruple precision, with an additional 15%, 21.5% and 14.5% delay reduction, respectively. Furthermore, area is also diminished by 4%, 4.5 and 5%.

Categories and Subject Descriptors: B.2.4 [**Arithmetic and Logic Structures**]: High-Speed Arithmetic—*Cost/performance*

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: Floating point unit, multiply accumulation, multispeculation

ACM Reference Format:
Alberto A. Del Barrio, Nader Bagherzadeh, and Román Hermida. 2014. Ultra-low-power adder stage design for exascale floating point units. ACM Trans. Embedd. Comput. Syst. 13, 3s, Article 105 (March 2014), 24 pages.
DOI: http://dx.doi.org/10.1145/2567932

1. INTRODUCTION

As computing capabilities demand continues increasing, the importance of efficient Floating Point Units (FPUs) is becoming more critical. The ASCI Red was the first supercomputer able to surpass the teraflop barrier while executing the Linpack dense matrix library [Kogge 2008]. Nowadays these teraflop systems are being comercialized [Vangal et al. 2008], and top datacenters today are able to obtain several petaflops [Top500 2012], but the research for more powerful systems is still going on. The exaflop era is the future for supercomputers. This means that the upcoming machines will be a thousand times faster than the current petaflop systems. Over the past 40 years, progress in supercomputers has benefitted from the technology advances,

This work has been supported by the Complutense University Postdoctoral Grant Del Amo and the Spanish Government Research Grants TIN 2008/00508 and TECS2012-33892.
Author's address: A. A. Del Barrio; email: abarriog@ucm.es.

DOI: http://dx.doi.org/10.1145/2567932

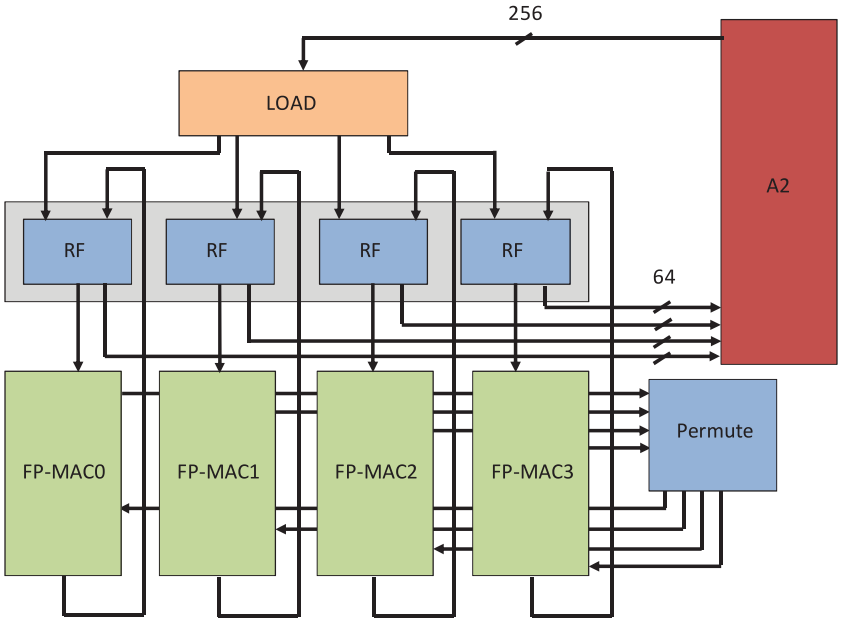


Fig. 1. The BG/Q Quad Floating Point Processing Unit [Haring et al. 2012].

that is, integrated circuits scaling. Nevertheless, for the first time in the last decades, these technology advances are at risk, because while transistor density on silicon is projected to increase with Moore's law, the energy efficiency of silicon is not. Hence, power has become the leading design constraint for future computers [Leavitt 2012].

Kogge et al. [2008] elaborated a report for DARPA in order to evaluate the challenges of developing an exascale system in the near future. The power budget of such system would be 20MW. Nonetheless, according to their estimations, the best exascale system that could be built to the best of their knowledge would require around 70MW, considering a power efficiency of 20Gflops/W per processor, 742 cores per processor and 4 Floating Point Units (FPUs) per core. And in these estimations, FPUs account for 21% of the total power budget. Hence it is crucial to develop fast ultra-low power FPUs.

The state of the art of FPUs is also evolving gradually. The FPUs considered in Kogge's report consist of a double precision FP-MAC with their corresponding register file, instruction RAM and L1 data RAM. Other examples are the FPUs that were utilized in the Blue Gene/Q supercomputer. These FPUs are composed of a Quad Processing Register file, 4 FP-MAC double precision units, and a Permute unit, as it can be observed in Figure 1 [Haring et al. 2012]. Similar tiles of FP-MACs can be found in GPUs [Owens et al. 2008] and many-core processors [Fan et al. 2012]. These many-core systems are becoming widespread nowadays and can be found in today's computer graphics cards [Nvidia 2012], such as the GeForce or the Quadro families, or in datacenters, such as the Tesla family. For example, the recently released Quadro K5000 card is able to provide 2.1 teraflops in single point precision, consuming 122W, that is, a power efficiency around 20 Gflops/W. This card is composed of 1536 cores organized in several streaming processors called SMX. Every core possesses a pipelined single precision FPU, and besides many double precision FPUs are shared among the cores inside the same SMX. Hence, many-core systems seem to be the key for reaching the exaflop, and FPUs are playing an important role.

Table I. IEEE Floating Point Wordlengths inside the FP-MAC for Different Precisions

Wordlength	Meaning	SP	DP	QP
W_t	Total bitwidth	32	64	128
W_s	Sign bitwidth	1	1	1
W_e	Exponent bitwidth	8	11	15
W_f	Mantissa bitwidth without implicit 1	23	52	112
W_{ep}	Exponent processing bits	10	13	17
W_{align}	Alignment shifter bitwidth	74	161	341
W_{csa}	3:2 CSA bitwidth	48	106	226
W_{sh}	Normalizing shift amount	7	8	9

From the application point of view, the multiply-add accumulation is essential in many scientific and engineering programs. Multimedia applications, such as 3-D graphics and signal processing, demand fast units able to perform this operation. For instance, let's consider the dot product, a common functionality that is utilized to identify the motion of objects. Thereby, it is very important to devise efficient FP-MACs from both performance and power points of view.

Over the last decade, a large number of different FP-MACs have been proposed [Jessani and Olson 1996; Jessani et al. 1998; Lang and Bruguera 2004, 2005; Vangal et al. 2006; Huang et al. 2012]. They all have three modules in common: the multiplier, the adder, and the rounder/normalizer. In this article we propose to optimize the adder module in order to reduce its power consumption, by applying multispeculative ideas that have been recently utilized for integer units. This reduction is accompanied by an area and execution time reduction. According to our experiments, our proposed FP-MAC consumes 17.5%, 23% and 16.5% less energy for single, double and quadruple precision, respectively. These gains are also accompanied by additional 15%, 21.5%, and 14.5% delay decreases and 4%, 4.5% and 5% area reductions.

The rest of the article is organized as follows. Section 2 describes the work related to FP-MACs, Section 3 motivates our techniques with an example, and Section 4 presents our proposal and some illustrative examples to help its understanding. Finally, Sections 5 and 6 show our experimental results and concluding remarks.

2. RELATED WORK

The FP-MAC implements the multiply-add operation according to IEEE-754 format [IEEE 2008]. In this format, a number X represents the value

$$X = (-1)^s * m * \beta^{e-\delta}, \tag{1}$$

where s, m and e represent the sign, normalized mantissa and exponent of the corresponding floating point number. Besides, we must take into account that the IEEE-754 includes an implicit 1 in the most significant position. Thus, if f is the bit chain stored for X, $m = 1.f$. The base is β, which for binary representation is 2, and δ is the exponent bias. It must be noted that if p is the number of bits devoted to represent the exponent, this exponent bias is calculated as $\beta^{(p-1)} - 1$. The number of bits utilized for representing each of these fields for single (SP), double (DP) and quadruple (QP) precision in a binary system is shown in Table I. Moreover, this table contains the different wordlengths inside the FP-MAC unit.

Hence, given A, B and C, the result after the multiply-add operation must be a number Z such that [Koren 2002; Ercegovac and Lang 2003],

$$\begin{aligned} A &= (-1)^{s_a} * m_a * 2^{e_a-\delta}, B = (-1)^{s_b} * m_b * 2^{e_b-\delta}, C = (-1)^{s_c} * m_c * 2^{e_c-\delta}, \\ Z &= round\left[(-1)^{s_a \oplus s_b} * m_a m_b * 2^{e_a+e_b-\delta} + (-1)^{s_c} * m_c * 2^{e_c-\delta}\right]. \end{aligned} \tag{2}$$

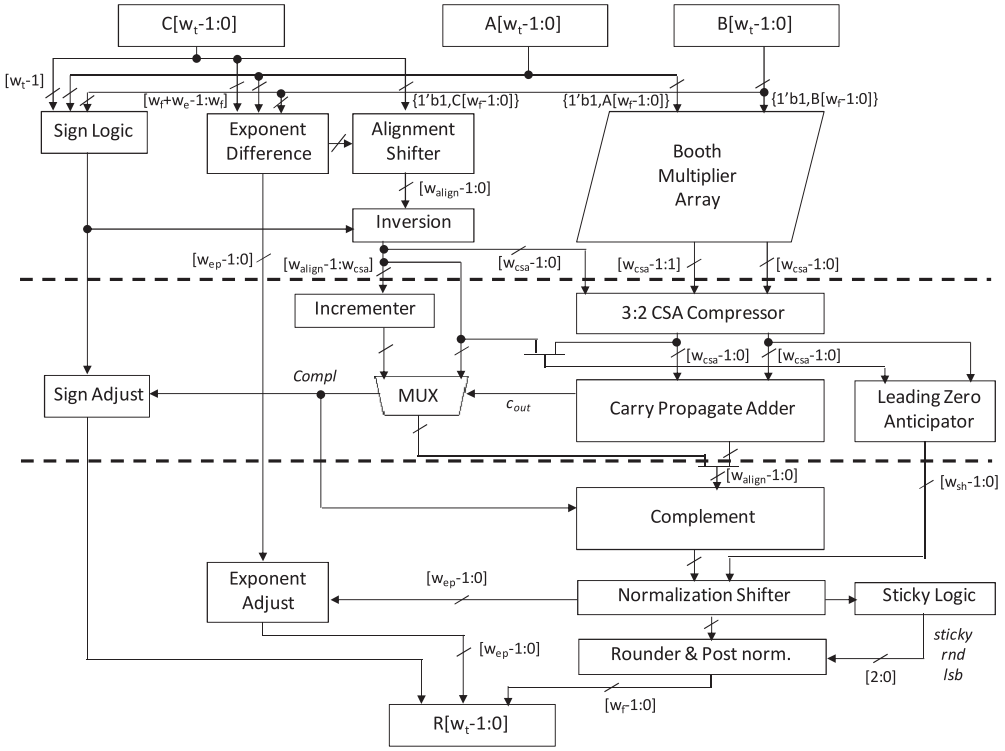


Fig. 2. Baseline FP-MAC design.

As can be observed, the computation of Z requires a product, an addition, a rounding and normalization, and some shifts in order to align the operands properly. It should be noted that the product is totally unsigned, while C can be negative, despite the fact that IEEE-754 is a sign and magnitude format. Thus, the FP-MAC must be able to perform subtractions, but produce a result in sign and magnitude, that is, nonnegative. So, depending on the implementation, some one's complements, that is, inverters, or two's complements are necessary too.

Figure 2 depicts the baseline FP-MAC [Jessani and Putrino 1998; Raman et al. 2000; Chatterjee et al. 2005] with more details. Although the number of pipeline stages and modules that belong to each stage depend on the target cycle time, an implementation divided into three stages will be considered as baseline as follows.

—*Multiplication*. In this stage the product of A^*B is performed. Most implementations utilize a Booth recoding multiplier [Koren 2002; Ercegovac and Lang 2003], and produce the result in Carry Save format. Moreover, the sign calculation, the exponent difference, the alignment of operand C and its inversion, if necessary, are computed in parallel. In the rest of the article, the output of the inverter module will be called inverted C, C_{inv}, regardless the sign of C.

—*Addition*. In this stage, the addition of the two vectors coming from the previous stage plus the C_{inv} is produced. Hence, first it is necessary a Carry Save tree (CSA) to reduce the operands from 3 to 2. Second, the addition is split into two fragments that are computed in parallel. The least significant fragment adds both vectors coming from the CSA stage, while the most significant one must only add the carry-out proceeding from the least significant module. Hence, the most significant fragment

is implemented with the carry-select technique, by multiplexing both C_{inv} and $C_{inv}+1$. Finally, the sign is adjusted by utilizing the most significant bit of the most significant fragment. If this bit is '1', the addition result is negative, so in the next stage it will be converted to sign and magnitude format. Furthermore, the detection of the Leading Zero or Leading One bit [Oklobdzija 1992; Schmookler and Nowka 2001; Verma et al. 2009] is performed in parallel. The purpose of this calculation is to determine the amount of bits that the addition result will have to be left shifted during the next stage to produce a normalized mantissa.

—*Rounding and normalization*. In this last stage the result is complemented, normalized and rounded, if required. Moreover, the exponent is adjusted taking into account the number of bits that the result is shifted.

First reported techniques in literature [Montoye et al. 1990; Jessani and Putrino 1996; Jessani and Olson 1998] utilized the addition in one's complement, which requires an end around carry adjustment for effective subtraction. Later, more aggressive designs [Lang and Bruguera 2004, 2005; Vangal et al. 2006; Huang et al. 2012] substituted it by a two's complement addition. It is true that if the result is negative, it will need the two's complement to produce the correct mantissa, but the optimizations performed in those works are easier to apply when working in two's complement.

Some researches [Tan et al. 2009; Huang et al. 2012] have focused on the multiplication stage optimization, by utilizing smaller iterative multipliers. Thus, power consumption is reduced, but large precision formats require more cycles to complete the operation. The proposal described in Vangal et al. [2006] considers delayed addition, as in Luo and Martonosi [2000], for optimizing the delay of large sums. Moreover, authors optimize the SP FP-MAC by adding in 32 base. Thus, they substitute the shifter by constant shifters, as the change of base provokes a decrease in the exponent length from 8 to 3 bits. In this way, the number of cases to consider is small and there is no need to use large shifters. However, the mantissa to be accumulated is increased from 24 to 55 bits. Besides, this solution is not scalable for DP and larger sizes.

Works presented in Lang and Bruguera [2004, 2005] propose to split the addition of the two bit vectors proceeding from the CSA, performing a part during the addition stage and another part during the rounding stage. The idea is to start normalizing the result from the addition stage as soon as possible. With this technique, authors claim a 15–20% execution time reduction. However, it requires the utilization of dual adders, which replicate large adder modules, and some costly hardware for achieving a correct timing. For example, as the normalization starts before the whole addition finishes, the result proceeding from the first part of the addition must be complemented without knowing the sign of the result, that is, the most significant bit of the second part of the addition. Hence, a sign detector is required for anticipating the sign bit.

The idea of splitting the addition can also be found in integer arithmetic [Nowick 1996; Lu 2003; Verma et al. 2008; Del Barrio et al. 2012]. An n-bit adder is divided into several k-bit fragments operating in parallel, $k \ll n$, because if k is large enough the carry-in of a chunk will be quasi-independent from the carry-out of the previous fragment. This fact achieves a great average execution time reduction, but the probability of finding interdependent carries still exists. Hence, an error detector and correction circuit must be included in order to guarantee a correct result at the output of the adder. In the rest of the article, the existence of interdependent carries will be called *failure* or *misprediction*.

Our purpose is to utilize the aforementioned integer ideas to develop an ultra-low-power addition stage inside the FP-MAC. The first consideration that must be made is the fact that FP-MACs are pipelined units. Thus, such approaches as [Nowick 1996] are not valid, because they rely on asynchronous adders. Secondly, in order to reduce

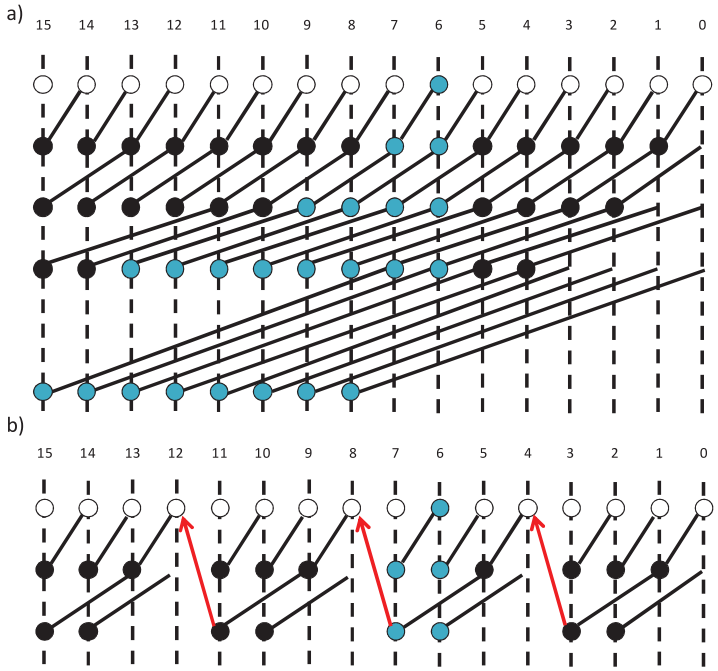


Fig. 3. a) 16-bit Kogge-Stone adder, b) 16-bit Multispeculative Kogge-Stone adder with $k = 4$ bits.

the power consumption as much as possible, the replication of many k-bit modules, as proposed in [Lu 2003; Verma et al. 2008], is not suitable because of their area and power overhead. On the contrary, the work presented in Del Barrio et al. [2012]. Proposes to divide the n-bit adder into exactly n/k k-bit fragments working in parallel. Several predictors will provide the carry-in signals. This simplifies the error detection and correction, as well as it diminishes the area of the resulting *Multispeculative Adders*. However, while the vast majority of additions will be correct after 2 cycles at the most, the variable latency of these adders is not suitable for a pipelined structure such as the FP-MACs, because every misprediction will stall the whole unit.

Therefore, it is critical to reduce the number of mispredictions. In this work, we propose to utilize the two's complementer in order to correct all the mispredictions simultaneously, at the same time that it performs the complement operation when necessary. In the following sections, these ideas will be explained in detail.

3. A MOTIVATIONAL EXAMPLE

In order to show our expected performance and energy benefits, let us consider the example shown in Figure 3, where the dot diagrams correspond with a 16-bit Kogge-Stone adder (KS) and its multispeculative version (MSKS) with 4-bit fragments. The white dots represent the initial calculation of signals $p_i = a_i \oplus b_i$ and $g_i = a_i \cdot b_i$, while the black dots correspond with operator $*$. This operator works such that $(G', P') * (G'', P'') = (G' + P' \cdot G'', P' \cdot P'')$, where the "signals are located on the left and the" signals on the right of every node in Figure 3(a) and 3(b). The blue dots represent those nodes whose signals can potentially switch. Finally, in Figure 3(b), the red arrows indicate how the carries are interconnected from a fragment to the following one.

In this example, we have assumed that the inputs at the 6th bit are changing. As it can be observed in Figure 3(a), a change in the inputs of bit 6 can produce commutations

up to 23 nodes. This problem is even worse if we consider more complex adders such as the ones presented in Lu [2003] and Verma et al. [2008], where the same bit flip affects several fragments at the same time. On the contrary, in the multispeculative case, as it is shown in Figure 3(b), the same flip at the 6th bit can only affect up to 5 nodes. This happens if there is a hit in the prediction of the carry-in to the following fragment. Otherwise, in the following cycle the correction can potentially affect 4, in general k, more nodes. Nevertheless, our purpose is to design a FP-MAC, so the techniques that are deployed in the following sections will explain how to apply these multispeculative concepts without actually mispredicting while adding mantissas.

Furthermore, as can be observed in Figure 3(b), the number of required nodes is also diminished. Thereby, the introduction of multispeculation will contribute to an area reduction in logarithmic-like adders, such as the KS, that are utilized in the adder stage of the FP-MAC.

4. OUR PROPOSED DESIGN

Most optimized FP-MAC adders are composed of dual or compound adders [Lang and Bruguera 2004, 2005]. This means that several adder modules are replicated for calculating the addition with both carry-in signals 0 and 1, that is, in a carry select fashion to reduce execution time. Besides, every module is usually implemented as a KS, the fastest but also the largest type of design. Nevertheless, as noted in Del Barrio et al. [2012], KS area behaves linear when its width is small. Hence, a Multispeculative Adder (MSADD) composed of several small KS modules working in parallel has a linear area and a reduced execution time. In terms of energy, MSADDs are also efficient, because the carry chain is shorter and hence the number of commutations is lower. This fact leads to a reduction of the dynamic power.

The main complication is to feed the correct carry-in to every module, without replicating it as in the compound adder. Del Barrio et al. [2012] there are several approaches to predict these carries, but the important fact is that if there is a failure in the prediction, a single cycle will be enough for correcting simultaneously all the mispredictions. Nevertheless, the possibility of propagating these mispredictions through the most significant modules, although extremely low, still exists. However, when considering pipelined structures, such as FP-MACs, the number of stalls due to the necessary corrections is critical. Hence, it is necessary to devise a mechanism for hiding the mispredictions.

FP-MACs contain a two's complementer module (C2C), which is located after the addition, as it can be observed in Figure 4. If the result is negative, it must be complemented, because the IEEE-754 representation consists of sign and magnitude. A C2C is basically an adder that adds 1 to the one's complement (C1) of its input. Therefore, our idea is to perform a probably inexact addition with a MSADD, and afterwards utilize a modified C2C to correct on the fly all the mispredictions. In this way, neither the compound adders nor the incrementer of Figure 2 are required and an important energy reduction is achieved.

4.1. Design and Implementation

The starting point is substituting the adder by a MSADD built with KS modules, which is more area and energy efficient, as it will be shown next. This shall be named MSKS. In order to simplify the logic, a Static Zero Predictor will be assumed, that is, there will be a failure iff the carry-out of any of the k-bit modules is 1.

Let S be the concatenation of the addition results of the individual k-bit modules, and C a vector composed of the concatenation of corresponding carry vectors. Note that every k-bit carry vector consists of the carry-in concatenated with $k-1$ zeros on the left. Thus, C can be viewed as an n-bit vector. Hence, at the output of the MSKS there

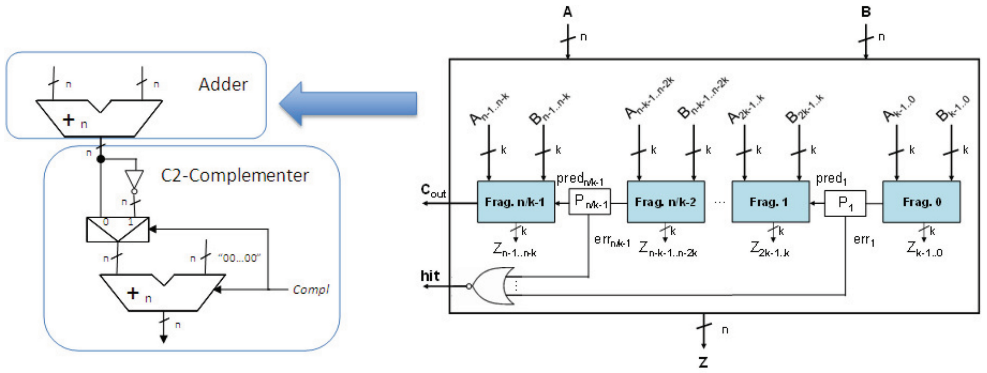


Fig. 4. Conceptual scheme for integrating the Multispeculative Adder inside the FP-MAC.

will be a couple of vectors (S, C) such that $Z = S + C$. Thus, our complementer must complement the result Z and/or correct the mispredictions, when necessary. However, Z is not available because the adder result is given in Carry Save form. The most straightforward implementation consists of utilizing the following trivial property: $C2(S + C) = C2(S) + C2(C)$. Thereby, it requires two C2Cs and a final addition, that is, two $+1$ adders and a final Carry-Propagate n-bit adder, which is worse than the baseline implementation.

In order to overcome the aforementioned Carry Save format limitation, the following trivial property will be utilized: $C2(X) = C1(X) + 1$. As shown in Figure 4, the C2C contains an adder, because given X, $C2(X) = C1(X) + 1$. In this way, one of the adder inputs is fixed as the constant 1. Nevertheless, the '1' constant can feed the carry-in input of the adder, leaving thus a free n-bit input. This free n-bit input will be used in our design for correcting the results when necessary.

Let *Compl* be the signal that indicates that the result must be complemented, and *Mi* the signal that points out that there is a misprediction in fragment $i + 1$. Note that utilizing a Static Zero Prediction, *Mi* is actually the real carry-out of the previous fragment, that is, fragment i. It must be noticed as well that *Compl* is common to all the modules, while *Mi* is local to every k-bit module. Hence, the following 4 cases can be distinguished.

(1) *Compl* = '0', *Mi* = '0'. The k-bit result is correct and it must not be complemented.
(2) *Compl* = '0', *Mi* = '1'. The k-bit result is not correct and it must not be complemented. Hence, we must perform the operation $X + 1$.
(3) *Compl* = '1', *Mi* = '0'. The k-bit result is correct and it must be complemented. Hence, we must perform the operation $C1(X)$.
(4) *Compl* = '1', *Mi* = '1'. The k-bit result is incorrect and it must be complemented. Hence, we must perform the operation $C1(X + 1)$. It must be noticed that in the first fragment (fragment 0) there is no misprediction, as the carry-in is not being speculated. It will add 1 just in case a complement is required. Therefore in fragment 0, as the result is not incorrect, the operation $C1(X) + 1$ will be performed instead.

The first three cases are straightforward to implement. In order to implement the fourth case, we will utilize the following lemma and corollary.

LEMMA 1. *Let S, C be two N-bit vectors. Hence,* $C1(S + C) = C1(S) + C1(C) + 1$

PROOF. On the one hand, $C2(S + C) = 2^N - (S + C) = (2^N - S) - C = C2(S) + C2(C) = C1(S) + 1 + C1(C) + 1$

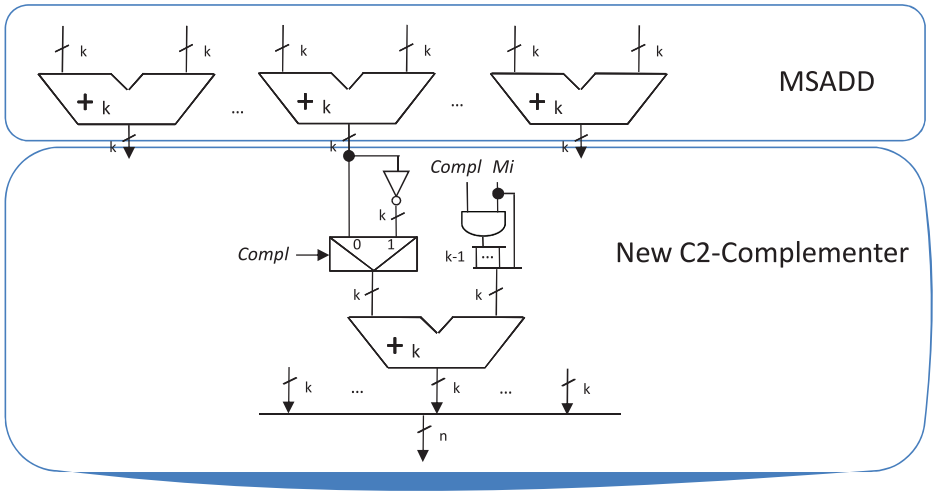


Fig. 5. Multispeculative Adder and new two's complementer.

We also have, $C2(S + C) = C1(S + C) + 1$
This implies,
$C1(S) + 1 + C1(C) + 1 = C1(S+C) + 1$
Therefore, $C1(S + C) = C1(S) + C1(C) + 1$.

COROLLARY 1. *Let X be an N-bit vector. Hence,* $C1(X + 1) = C1(X) + 2^N - 1$

PROOF. Using Lemma 1, $C1(X + 1) = C1(X) + C1(1) + 1 = C1(X) +$ "11 ...10" $+ 1 =$ $C1(X) +$ "11... 11" $= C1(X) + 2^N - 1$. □

Thereby, in order to implement case 4 it is necessary to complement the input and add a chain of k 1's. Figure 5 depicts the generic architecture that implements the four aforementioned cases. It must be noticed that the k-bit adders which compose the two's complementer are not fed with any carry-in. They work in parallel as well as the k-bit modules composing the MSADD, resulting in good performance.

Nevertheless, the correctness of the results cannot be guaranteed yet with this structure. It is true that with the previous four cases it is possible to correct and/or complement every k-bit result in parallel. However, as stated in Del Barrio et al. [2012], a misprediction can be propagated to the most significant module. Although most mispredictions can be corrected in a single step, there still exist cases where more steps are necessary. For example, let us consider the following example with $n = 16$ and $k = 4$ bits. In order to make it clearer, we are going to consider just an integer addition.

Example 1. Let $I1 = 1111\ 1111\ 1111\ 1111$ and $I2 = 0000\ 0000\ 0000\ 0001$. Applying multispeculation, the result of $I1 + I2$ after the first step is:

$S = 1111\ 1111\ 1111\ 0000$
$C = 0000\ 0000\ 0001\ 0000$

As the carry-out of fragment 0, that is, M1, is equal to 1, the new two's complementer must add this to the addition result of fragment 1. The new vectors would be:

$S = 1111\ 1111\ 0000\ 0000$
$C = 0000\ 0001\ 0000\ 0000$

As it can be observed, fragment 1 produces a new carry-out that must be propagated to generate a correct result. Continuing with this process, the initial misprediction will be propagated until the most significant module, that is, $n/k = 4$ steps will be

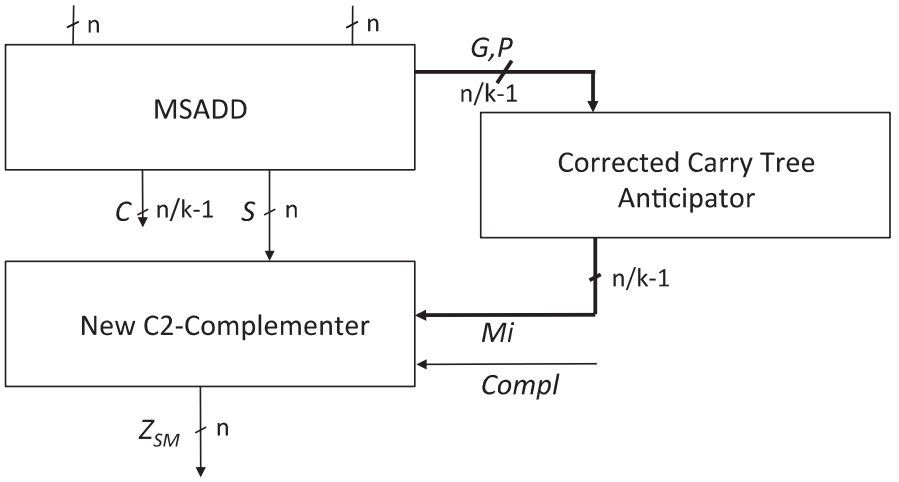


Fig. 6. Modified architecture with Corrected Carry Tree Anticipator.

necessary. This is the worst case, and although quite unusual, it must be taken into account. However, in an extremely optimized pipelined structure as the FP-MAC, so many additional steps cannot be allowed.

In this example, if M0, M1 and M3 were all 1, the final result would be correct after the second step. Thereby, it is necessary to devise a mechanism in order to provide in parallel the *correct Mi* signals to each k-bit module inside the new two's complementer. Moreover, it should be noticed that in this example the *Compl* signal has been overlooked. In a conventional implementation, the *Compl* signal is just the most significant bit of the addition. In other words, if this bit is 1, the result is negative and it must be converted to sign and magnitude format. Nevertheless, utilizing a MSADD it is not sure whether or not the most significant bit is correct.

In the following subsections, both the generation of the correct *Mi* signals and the anticipation of the *Compl* signal will be addressed in order to provide a correct result with no additional cycle penalties with a probability greater than 99.9%.

4.2. The Corrected Carry Tree Anticipator

As it has been explained before, the problem of the design proposed in Figure 5 is that each *Mi* signal corresponds to the carry-out generated by fragment i, as if no multispeculation were considered.

It must be noticed that every fragment consists of a k-bit KS adder. Hence, in order to produce both the S vector and the carry-out, they must calculate the *fragment generate* and *propagate* signals. When compared with a conventional n-bit KS adder, these signals contribute to the generate and propagate signals associated to the n-bit module. Thereby, it is necessary to incorporate a module able to receive the k-bit fragment or *group* generate and propagate signals and use them to compute the n-bit generate and propagate signals. In this way, the carry-out associated to every fragment will be the true carry and thus the *Mi* signals will be correct. The overview of the modified architecture is depicted in Figure 6.

As it can be observed in Figure 6, the MSADD feeds the *Corrected Carry Tree Anticipator* (CCTA) with the generate and propagate k-bit group signals. As there are n/k modules, and fragment 0 is correct for sure, only $n/k - 1$ propagates and generates are actually necessary. The CCTA is responsible for calculaing the $n/k - 1$ correct *Mi* signals, which are then driven to each k-bit module inside the new two's complementer

(NC2C). Finally, the four cases described in Section 3.1 are applied to produce a correct result in sign and magnitude format (Z_{SM}).

The CCTA consists of a carry tree as the prefix-adders one [Koren 2002; Ercegovac and Lang 2003], whose inputs are the generate and propagate k-bit group signals generated from the MSADD (G, P). In our case, a KS carry propagation tree has been utilized. The carry tree delay is O(log(n/k)), while the area and energy consumption remain linear as in the case of MSADDs [Del Barrio et al. 2012] because typically $n/k \ll n$. Furthermore, it must be observed that it is not necessary to calculate either the initial generate and propagate signals, as in the first level of a logarithmic adder, or the addition result. Thereby, the area and energy penalty due to this module are small, as it will be demonstrated in the experiments section.

Finally, it should be noted that in spite of calculating the correct carries thanks to the CCTA, the C2 module should not be implemented with the carry-select technique, that is, replicating several fragments both carry-in 0 and 1 in order to reduce the delay. As the adder is fragmented, the sign is uncertain. Thus, the four cases mentioned in Section 4.1 must be considered in parallel to generate the correct result. Thereby, every fragment must be replicated by 4, plus a 4 to 1 multiplexer, which is considered as excessive for an ultra-low-power approach as ours.

4.2.1. An Illustrative Example of Implementation. In order to show the simplicity of the carry trees in the multispeculative implementation, let us consider the dot notation of a 16-bit nonspeculative and a multispeculative implementation, as shown in Figure 7. Note that the nonsolid dots correspond with the calculation of the initial propagates and generates signals, while the solid dots perform the computation of $(G", P") * (G', P') = (G" + G' \cdot P", P" \cdot P')$, where $+$ and $\cdot$ are the logic OR and AND operations, respectively. Figure 7(a) depicts the dot notation of a conventional KS adder. Besides, it must be taken into account that the whole nonspeculative design needs 2 KS adders, as described in Figure 4. Figure 7(b) shows a multispeculative implementation using 4-bit fragments. Each $Ui = (Gi, Pi)$ represents the generate and propagate 4-bit group signals. It must be noticed that each Mi signal is driven to the carry-in entry of the following module. Hence, *M3* corresponds with the addition carry-out. Moreover, as it has already been mentioned, fragment 0 carry-in consists of the *Compl* signal.

As it can be observed, the multispeculative version is less expensive than the nonspeculative one, due to the greater complexity of large carry propagation trees. Furthermore, our proposed implementation is also faster, since the number of nodes in the critical path is lower: the conventional design requires $5 + 5 = 10$ nodes, while our proposal requires $3 + 2 + 3 = 8$ nodes. In general, in a conventional implementation there are two stages with a delay O(log(n)). On the contrary, in the multispeculative design both the MSADD and the C2-Complementer possess a delay O(log(k)), $k \ll n$. On the other hand, the CCTA delay is O(log(n/k)). It can start its execution when the generate and propagate k-bit group signals are available, which is actually some time before finishing the MSADD addition. Considering the worst case, that is, MSADD and CCTA are serialized, the whole delay of the multispeculative structure would be proportional to $\log(k) + \log(n/k) + \log(k) = \log(k) + \log(n)$, which is lower than the $2\log(n)$ delay in the conventional case.

It must be remembered that the location of the pipelining registers has not been taken into account in our estimations. As stated in Section 2, the number of pipeline stages depends on the final design, the delay of the whole FP-MAC and the frequency of the system. However, as both the addition stage and the two's complementer are in the critical path of the FP-MAC, diminishing their delay will contribute to the overall FP-MAC delay reduction.

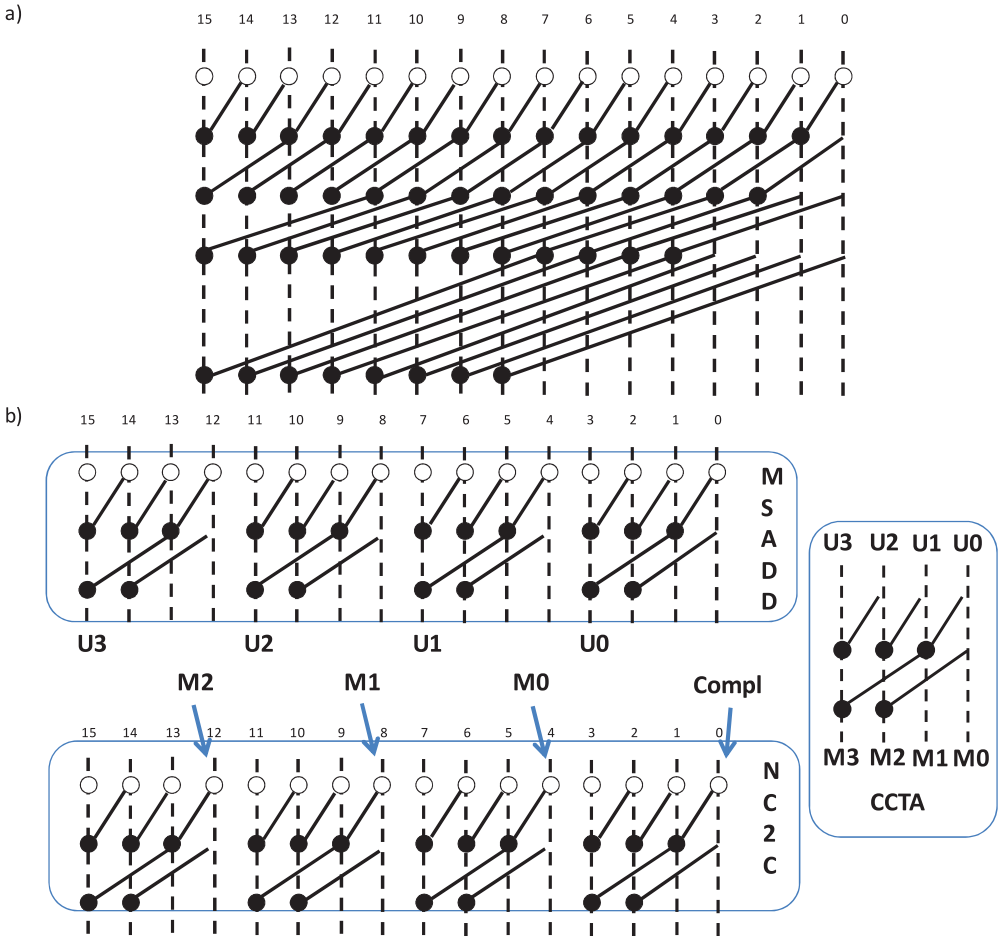


Fig. 7. Dot representation for 16-bit a) conventional KS, b) multispeculative addition and two's complement with CCTA

4.2.2. An Example of Execution. In order to illustrate the application of our principles, in this subsection a couple of examples will be described. Despite the fact that the IEEE-754 single precision standard imposes 74 bits for the adder stage, only 16 bits will be considered for making it clearer. In both implementations, let Z and Z_{SM} be the outputs of the adder and the two's complementer, respectively, and let M and T be the correction vector and transformed S vector, respectively, for the multispeculative case. Besides, it will be supposed that the *Compl* signal is ready before the C2 operation. Finally, it must be taken into account that at this point of the FP-MAC execution flow, both operands are aligned and complemented if required.

Example 2. Let $I1 = 1111\ 1111\ 1111\ 1111$ and $I2 = 0000\ 0000\ 0000\ 0011$. The result $(-1) + 3 = 2$ is positive, hence $Compl =$ '0'.

With a conventional flow $Z = Z_{SM} = 0000\ 0000\ 0000\ 0010$.

Applying multispeculation in the addition, the result of $I1 + I2$ is:

$S = 1111\ 1111\ 1111\ 0010$

$C = 0000\ 0000\ 0001\ 0000$

The CCTA calculates the correct carries as depicted by Figure 8:

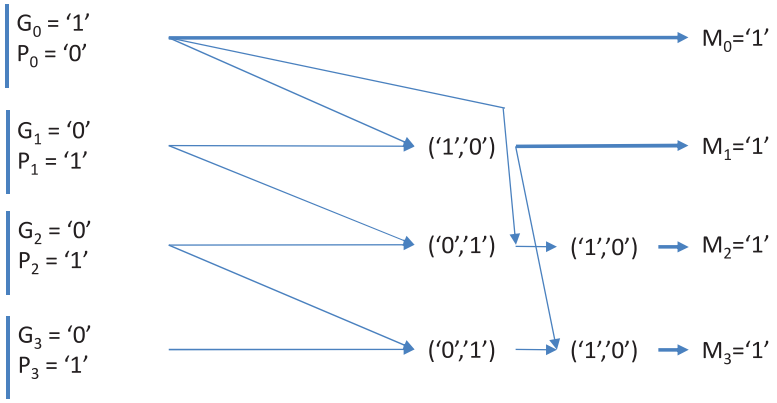


Fig. 8. Example 2 CCTA.

It must be remembered that given (*G"*, *P"*) and (*G'*, *P'*), being *X"* more significant than *X'*, the operation performed in every node is (*G"*, *P"*)*(*G'*, *P'*) = (*G"* + *G'* · *P"*, *P"* · *P'*). In the end, the *Mi* signals are stored in the generate variable with the longest depth in the tree. It must be noticed, that every *Mi*, is generated by fragment *i* in the MSADD, but feeds the carry-in of fragment *i* + 1 inside the NC2C. On the contrary, the NC2C fragment 0 is fed by the *Compl* signal itself. Hence, the operation *S* + *M* performed inside the NC2C is as follows:

$T = 1111\ 1111\ 1111\ 0010$

$M = 0001\ 0001\ 0001\ 0000$

$Z_{SM} = 0000\ 0000\ 0000\ 0010$

In this case, as *Compl* = '0', *S* = *T*. Hence, it is enough to apply case 1 to fragment 0, and case 2 to the rest of the fragments in order to produce a correct result.

Now let's suppose a second example where *Compl* = '1'.

Example 3. Let *I1* = 1111 1110 1111 1001 and *I2* = 0000 0000 0001 0011. The result (−263) + 19 = (−244) is negative.

With a conventional implementation $Z = 1111\ 1111\ 0000\ 1100$, and $Z_{SM} = 0000\ 0000\ 1111\ 0100$

Applying multispeculation in the addition, the result of *I*1 + *I*2 is:

$S = 1111\ 1110\ 0000\ 1100$

$C = 0000\ 0001\ 0000\ 0000$

The CCTA calculates the correct carries as depicted by Figure 9:

Let *Si* be the result vector calculated by fragment *i*. According to the 4 cases described in Section 3.1, inside the NC2C fragment 0 must perform the operation C1(*S0*) + 1 (because *Compl* = '1'), fragment 1 must perform C1(*S1* + 1) (case 4), fragment 2 must perform C1(*S2*) (case 3) and fragment 3 must perform C1(*S3*) (case 3). Hence,

$S = 1111\ 1110\ 0000\ 1100$

$C = 0000\ 0001\ 0000\ 0001$

are transformed into

$T = 0000\ 0001\ 1111\ 0011$

$M = 0000\ 1111\ 0000\ 0001$

Thereby, $Z_{SM} = 0000\ 0000\ 1111\ 0100$

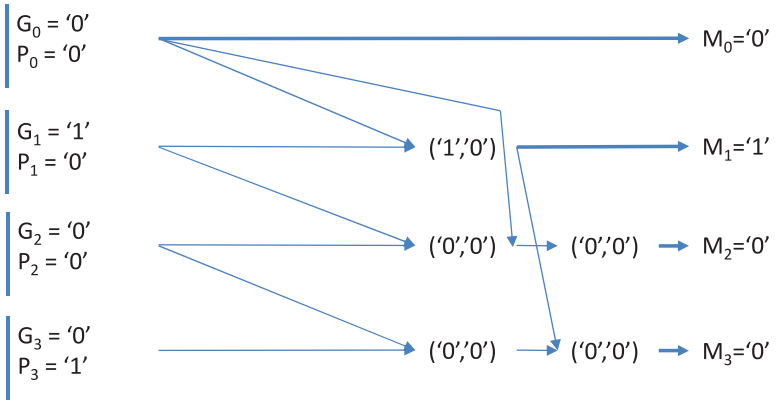


Fig. 9. Example 3 CCTA.

4.3. The Sign Predictor

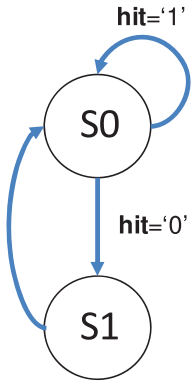
As it has been mentioned before, it has been assumed that the *Compl* signal is available before complementing the result. In the conventional implementation, the *Compl* signal is the most significant bit of the addition result, that is, if this bit is 1, the result must be complemented. However, with the multispeculative approach this choice is not possible because we do not know whether this bit is correct or not before going through the NC2C module.

In order to solve this situation, some works that divide the addition into several stages [Lang and Bruguera 2004, 2005] utilize a sign detector unit. This sign detector unit depends on two signals, namely: sub and d. Given the operation $A \times B + C$, the sub signal is only active if the sign of $A \times B$ and the sign of C are different, that is, it is necessary to perform a subtraction. The Sign Logic module, located in the multiplication stage of the FP-MAC, provides this signal. Given that Exp(X) corresponds with the exponent bits of the FP-number X, $d = \text{Exp}(C) - (\text{Exp}(A) + \text{Exp}(B)) = e_c - \delta - (e_a - \delta + e_b - \delta) = e_c - (e_a + e_b) + \delta$, and it indicates the number of positions to shift the C operand for aligning it with the result of the product. Hence, both signals are available prior to compute the addition.

Therefore, it is necessary to complement the addition result only if $sub = 1$. Regarding the d signal, if $d < 0$ the result will be positive, and if $d > 1$ the result will be negative for sure. If $d = 0$ or $d = 1$, a full comparison is necessary among the outputs of the 3:2 compressor and C [Lang and Bruguera 2004]. If $d = 0$ both C and $A \times B$ have the same weight. The $d = 1$ case stems from the fact that the output of the multiplier can range between 0 and 4 (not inclusive), as both multiplier inputs range between 0 and 2 (not inclusive). This takes long and it requires a tree comparator, although it can be performed in parallel with the addition.

As our purpose is to develop an ultra-low-power FP-MAC, such a costly comparator will not be implemented. Instead, the sign will be predicted with a high accuracy, and if there is a failure, a 1-cycle stall will be introduced in the FP-MAC for recalculating the complement.

In order to implement the predictor, a two states finite state machine and certain logic for estimating the carry are necessary. These two states represent the common operation mode (*S0*), and the correction mode (*S1*). In a first approach, the estimation *est* will be 1 iff it is sure that the result of the addition requires to be complemented, that is, iff sub = '1' and $d > 1$. In order to check whether this estimation is correct or not, the most significant bit after the C2 calculation is considered. If this bit is "0", then



hit	Q(i)	Compl	Q(i+1)	Comments
0	0	est	1	S0 and failure, transition to S1
0	1	-	-	This case never happens, In S1 there is always a hit
1	0	est	0	S0 and hit, remain in S0
1	1	¬(est)	0	S1 and hit, transition to S0

Fig. 10. FSM for controlling the possible stalls due to sign mispredictions.

the result is positive and there is a hit, otherwise there is a failure and a transition to the correction state. Only one cycle is necessary for performing the *recomplement* operation to produce a positive mantissa. Thus, after the correction, the state machine must recover its original state. Figure 10 summarizes the possible cases.

4.3.1. Sign Predictor Accuracy and Possible Optimizations. If in every prediction scheme the accuracy is important, when considering a pipelined FU the accuracy is critical, since every misprediction will stall the whole FU. The calculation of the number of hits and mispredictions can only be done via simulation, since equiprobability of cases is not a realistic scenario. According to the studies performed in Hamming [1970] and Pillai et al. [1998], it is more probable to find floating point numbers with close exponents than numbers with large exponent differences. As it can be observed in our approach, it is only possible to mispredict the sign if there is a subtraction and $d = 0$ or $d = 1$, that is, the inputs possess close exponents. Hence, the computation of *est* must be improved.

In order to increase the accuracy of the sign predictor, we will utilize an approach similar to the carry estimation of asynchonous Ripple Carry Adders (RCAs), which is described in Ashmila et al. [2005]. In this work, authors divide the adder into two halves and estimate the carry-in to the most significant fragment by using the most significant bits of the least significant fragment. For instance, if these most significant input bits, namely x_i and y_i, are equal, the carry is known for sure, and if they are different, the carry is uncertain. Hence, the estimated carry $est_i = x_i \cdot y_i$. In order to increase the accuracy, authors propose to use more bits belonging to the least significant fragment. For example, considering two bits $est_i = x_i \cdot y_i + (x_i + y_i) \cdot (x_{i-1} \cdot y_{i-1})$, with three bits $est_i = x_i \cdot y_i + (x_i + y_i) \cdot (x_{i-1} \cdot y_{i-1} + (x_{i-1} + y_{i-1}) \cdot (x_{i-2} \cdot y_{i-2}))$, etc. Utilizing this methodology, the probability of mispredicting is $1\text{-}2^{-(w+1)}$, being w the number of input bits employed in the estimation. Although this probability has been calculated considering equiprobability of cases, it is clear that the accuracy of the estimation increases as w increases.

In our case, we have applied a similar methodology considering the effective most significant inputs, that is, those corresponding to the nonspeculative CPA. As the troublesome cases are $d = 0$ or $d = 1$, both the output of the multiplier and C_{inv} are aligned or quasi-aligned, respectively. Hence, the procedure consists of estimating the carry-in to the effective most significant cell of the addition. Thus, applying the classical Full Adder equation, that is, $z_i = x_i \cdot y_i + x_i \cdot c_{i-1} + y_i \cdot c_{i-1}$, the most significant bit is predicted.

In order to understand the estimation, without loss of generality, let us suppose single precision. Then, the multiplication is 24×24 bits, and its result *mul* is 48-bit

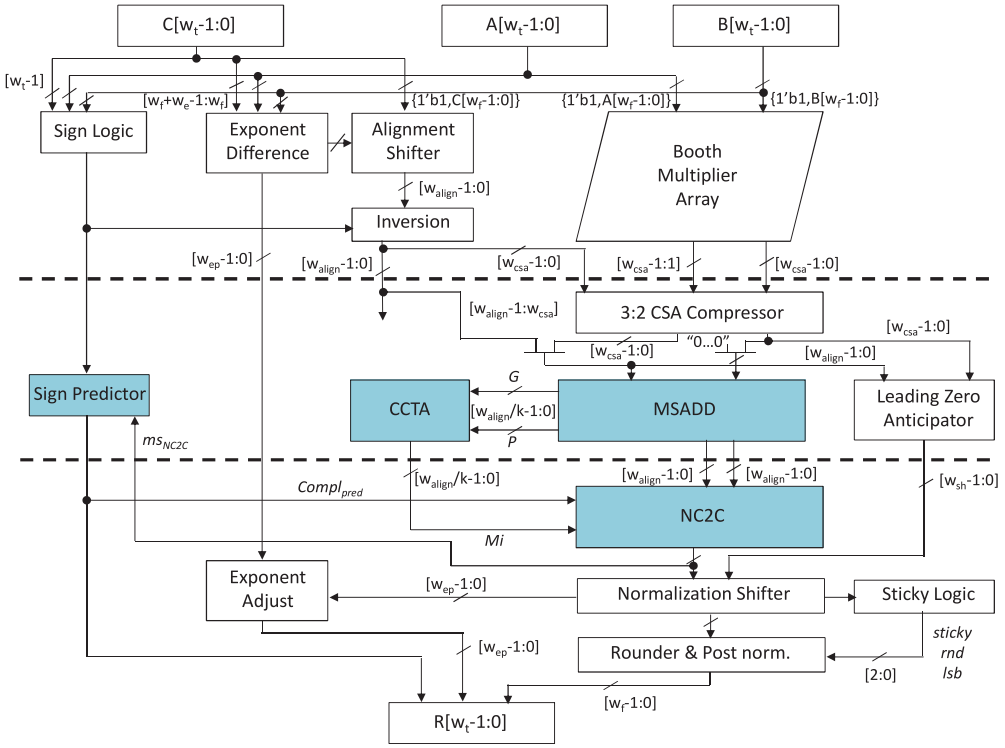


Fig. 11. Multispeculative FP-MAC.

wide, where mul_{47} and mul_{46} possess integer weight. If $d = 0$, the most significant bit of C_{inv} is aligned with mul_{46}, and if $d = 1$ it is aligned with mul_{47}. Applying our procedure, the carry-in corresponding to the addition of mul_{47} and C_{inv47} is estimated, and with this carry-in the effective most significant bit is predicted. In single precision, the actual addition is 74-bit wide, but those more significant bits will be an extension of this effective most significant bit.

Finally, it must be noticed that *est* calculation must be faster than the k-bit addition plus the $n/k - 1$ bit CCTA for providing the *Compl* signal before the NC2C starts its computation.

4.4. Overall Architecture

Figure 11 depicts how our proposal is introduced into the baseline FP-MAC design, which was shown in Figure 2.

Solid blocks represent the new modules. It must be noticed that the initial division into three stages is maintained just for making the comparisons between the two figures easier, but the number of pipeline stages will be determined considering the delay of the whole structure and the target system frequency. As it can be observed, the MSADD and the CCTA substitute the conventional carry propagate adder, the incrementer and the multiplexer, while the NC2C substitutes the conventional C2C in order to produce a correct and complemented result, when necessary. The group generate (G) and propagate (P) signals are driven to the CCTA for producing the correct carries (Mi), which will feed the NC2C. Finally, the Sign Predictor is introduced instead of the Sign Adjust module. It will provide the complement signal prediction ($Compl_{Pred}$), and

will check if the most significant bit of the complemented result (ms_{NC2C}) is 1, thus needing to recomplement the result in the following cycle.

It should be noted that with this design there is no need to modify neither the Load-Store Queue nor the processor fetch-issue logic. The only consequence of a sign misprediction is to delay the operation inside the FP-MAC. Hence, the result will be copied into the Reorder Buffer only when it is ready, and those operations depending on it will be issued as usual, provided that there are enough resources in the Reorder Buffer and the reservation stations. In other words, there is no difference with regard to conventional Out Of Order architectures.

5. EXPERIMENTS

In this section, we first describe our experimental framework. Next, we discuss our results.

5.1. Framework

In order to measure the delay, area and power of our proposed design, we have coded our designs in VHDL. Afterwards they have been synthesized with *Synopsys Design Compiler*, using a 65 nm library, and finally *Synopsys Power Compiler* has been utilized for measuring the power.

Thus, the energy per operation is computed as the product of both power and delay. Power is measured in mW and delay in ns. Hence, energy is measured in pJ. It must be noticed that the average delay and average energy per operation are very close to those values, as the Sign Predictor is able to guess the vast majority of the signs.

The accuracy of our Sign Predictor has been measured by running the PARSEC benchmark suite [Bienia et al. 2008; Bienia and Li. 2010; Bienia 2011], as well as some benchmarks belonging to the Splash-2x suite [Woo et al. 1995] in a 64-bit x86 machine. In order to accomplish this goal, the *soft-fp* emulation library [*soft-fp* 2012] has been modified and compiled in order to get benchmark traces. Afterwards, these traces were used to apply our prediction schemes and the corresponding hit results were obtained.

5.2. Results

In order to check the performance, area, power and energy of our proposal, several experiments have been performed.

5.2.1. Synthesis of the Modified Adder and Complementer. In this first experiment, the architecture shown in Figure 6 and the basic Sign Predictor, that is, with no additional information bits, have been synthesized. Besides, as the 3:2 compressor belongs to the critical path of the addition stage, it has also been included to study how much the delay of this stage can be reduced. These results have been compared with a baseline implementation that consists of a compound adder and a traditional complementer, as the ones that are utilized by the FP-MAC depicted in Figure 2.

It must be observed that the synthesis results have been obtained using no constraints, so the relationship between delay, area and power depends on how the *Design Compiler* is running its optimizations. For instance, a design optimized with a low delay needs more area and consumes more power. Nevertheless, energy consumption, that is, the product of both delay and power, gives a good idea about the efficiency for different implementations.

Tables II, III, and IV depict the delay, area, power and energy of a conventional non-speculative implementation and our proposed design for single, double and quadruple precision, respectively. The leftmost column contains the identifier of the implementation: conventional (Conv) and KX, where X is the addition fragment width for the

Table II. Synthesis Results of 3:2 Compressor + Adder + Complementer for Single Precision

	Delay (ns)	Area (um2)	Power (mW)	Energy (pJ)
Conv	2.06	3829.3	1.771	3.65
K32	2.01	3000.2	1.63	3.28
K16	1.47	2898	1.70	2.50
K8	1.51	2466.7	1.442	2.18
K4	1.29	2551.7	1.44	1.86

Table III. Synthesis Results of 3:2 Compressor + Adder + Complementer for Double Precision

	Delay (ns)	Area (um2)	Power (mW)	Energy (pJ)
Conv	2.49	9652	3.955	9.85
K64	2.15	8141.8	4.259	9.14
K32	2.16	6654.2	3.672	7.93
K16	1.73	6441.5	3.69	6.36
K8	1.73	5379.8	3.145	5.44
K4	1.33	5736.6	3.237	4.31

Table IV. Synthesis Results of 3:2 Compressor + Adder + Complementer for Quadruple Precision

	Delay (ns)	Area (um2)	Power (mW)	Energy (pJ)
Conv	2.81	23542.9	8.656	24.29
K128	2.40	19158.5	9.381	22.50
K64	2.35	17653.3	9.303	21.85
K32	2.36	13957.2	7.854	18.50
K16	2.11	13502.2	7.89	16.64
K8	1.77	11531.5	6.707	11.85
K4	1.79	12348.4	6.937	12.44

multispeculative implementations. The rest of columns contain the delay, area, power and energy per operation of the corresponding design.

As it can be observed, the multispeculative implementations are faster, smaller and more power and energy efficient. Concretely, the best designs are able to obtain these gains.

—For single precision, the K4 design is 37% faster, 33% smaller and consumes 49% less energy.
—For double precision, the K4 design is 47% faster, 41% smaller and consumes 56% less energy.
—For quadruple precision, the K8 design is 37% faster, 51% smaller and consumes 51% less energy.

In general, utilizing 4-bit or 8-bit fragments achieves the highest energy reductions because the Kogge-Stone area and power behave linearly, that is, similar to Ripple Carry Adder, when the size is small, but when it increases, it penalizes too much. Nevertheless, it must be taken into account the additional penalty due to the wiring augmentation, provoked by the larger number of fragments. This is the reason why with quadruple precision the K4 design is worse than the K8. In the case of double precision, K4 is worse than K8 in terms of power, because the delay has been significantly optimized. In fact, K4 is the best when considering overall energy in double precision.

Figures 12, 13, and 14 depict the power breakdown, in mW, for single, double and quadruple precision, respectively, and for both the baseline implementation, labeled as

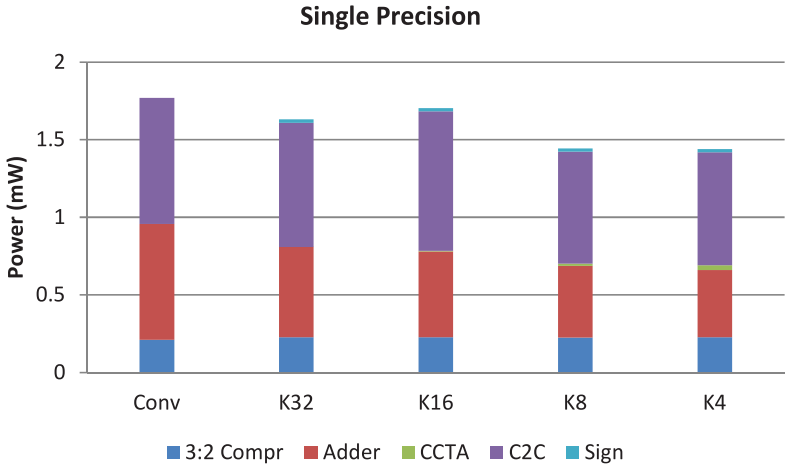


Fig. 12. Power breakdown of 3:2 Compressor + Adder + Complementer for single precision (mW).

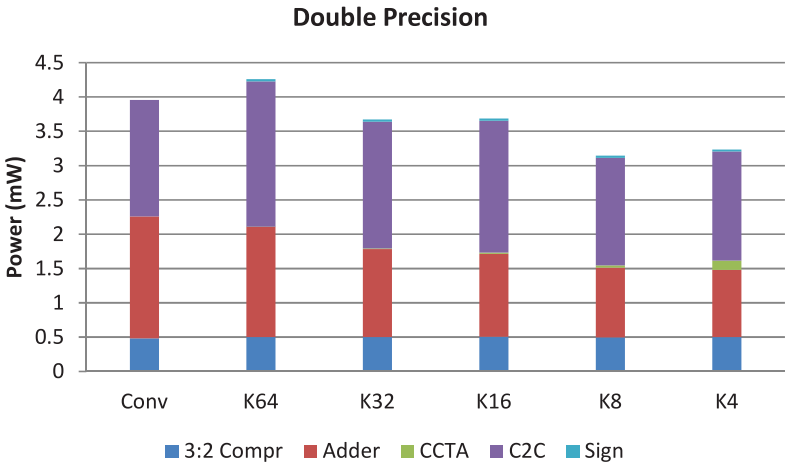


Fig. 13. Power breakdown of 3:2 Compressor + Adder + Complementer for double precision (mW).

Conv, and the multispeculative ones, labeled as KX, where X is the addition fragment width. Each column depicts the power consumed by the 3:2 Compressor (3:2 Compr), the adder (Adder), the Corrected Carry Tree Anticipator (CCTA), the two's complementer (C2C) and the basic Sign Predictor (Sign). Several interesting facts can be observed. In general, both the adder and C2C power consumption decrease as the fragment width decreases. The slight variations are due to *Synopsys* optimizations. On the contrary, the CCTA consumption increases as the fragment chunk is diminished. This is logical, since the CCTA inputs bitwidth is $n/k - 1$. Nevertheless, the power penalty due to this new module is low. The reason is that in spite of implementing a Kogge-Stone carry-propagation tree, the CCTA does not compute either the first or the last stage of a Kogge-Stone adder. In the first stage of a Kogge-Stone adder, the calculation of the initial generate and propagate signals takes place, while in the last stage the final computation of the addition result bits is performed. These are the most costly stages, since there is a cell per bit, contrarily to the middle stages of the carry-propagation

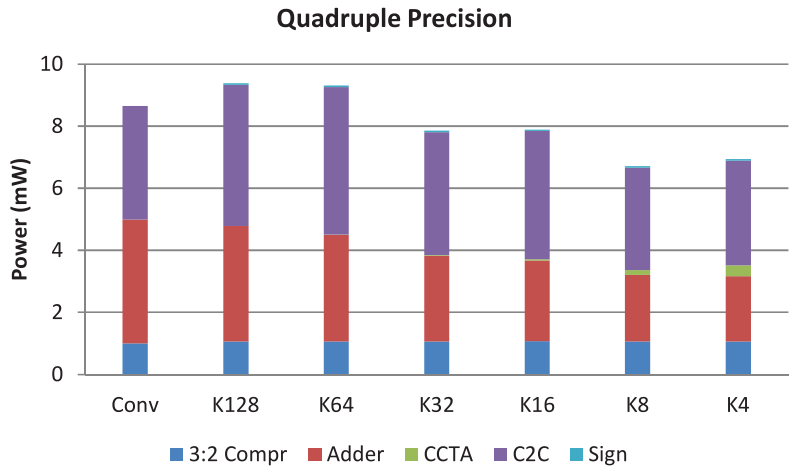


Fig. 14. Power breakdown of 3:2 Compressor + Adder + Complementer for quadruple precision (mW).

tree. As these cells do not exist inside the CCTA, it is possible to maintain a low power consumption.

Finally, it must also be observed that the power penalty due to the Sign Predictor is almost negligible. This is also reasonable, since it is composed of a two state FSM with some additional gates.

5.2.2. Accuracy of the Sign Predictor. In this second experiment, the effectiveness of the Sign Predictor is evaluated. Several benchmarks belonging to the PARSEC and Splash-2x suites have been run and the corresponding traces have been processed as explained in the framework section. Figures 15 and 16 show the hit rates of the basic Sign Predictor (0-bits), and the optimized one with 1-bit, 2-bits, and 3-bits of information. The implementation has been performed as explained in Section 3.3.1.

As it is observed, the accuracy of the predictor even in its basic form is really high. In most of the cases the hit rate is 1, and in those cases where it is lower, it is possible to exceed the 0.99 hit rate by utilizing up to three bits of information. Moreover, the delay of the whole addition stage is not affected, since the additional combinational logic is only composed of few levels that operate in parallel with the 3:2 compressor, the adder and the two's complementer.

It must be noticed that Figures 15 and 16 correspond with the *float* (32-bit) and *double* (64-bit) data types, respectively. To the best of our knowledge, with the current approach it is not possible to get the traces for larger data types, as the *long double*. In the x86-64 architecture, the *long double* type corresponds with the extended format (80-bit), which requires enabling the x87 registers for returning values, which disables the software library. On the other hand, the *long double* type is mapped to a quadruple precision type (128-bit) in SPARC, but we have not found support enough from *libgcc* to compile the soft-fp emulation library under this architecture.

Nevertheless, as most architectures implement the quadruple precision operations with double precision routines, the results shown in Figure 16 can serve to illustrate the accuracy of quadruple precision units.

5.2.3. FP-MAC Synthesis Results. In the last experiment, a complete FP-MAC without pipelining registers has been synthesized. The baseline case implements the architecture shown in Figure 2, and our approach the one depicted in Figure 11. Table V shows area, delay, power and energy, (with the units in brackets) of the corresponding row.

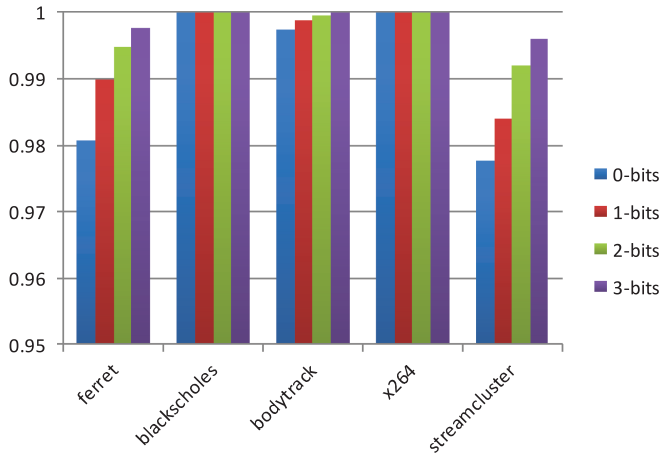


Fig. 15. Hit rate in single precision, using 0, 1, 2 and 3 bits of information.

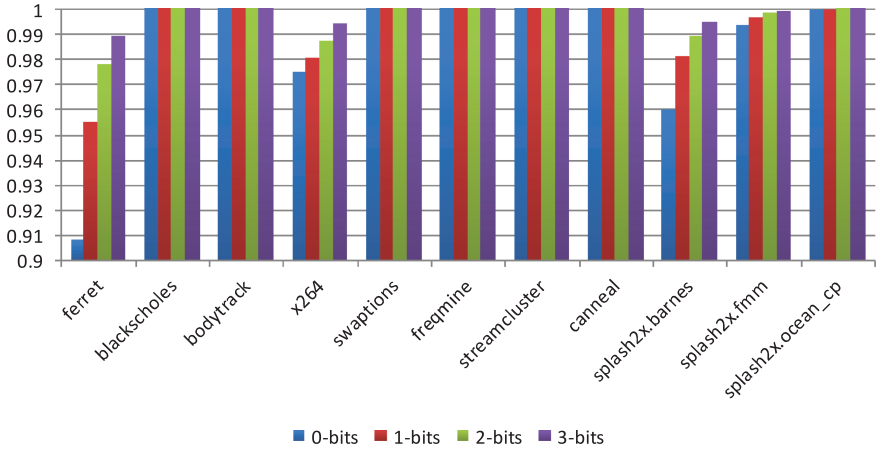


Fig. 16. Hit rate in double precision, using 0, 1, 2 and 3 bits of information.

Table V. Synthesis Results of FP-MACs for Different Precisions

	Precision	Area (um2)	Delay (ns)	Power (mW)	%P Gain	Energy (pJ)	%E Gain
Conv	Single	23676.1	4.98	15.10	-	75.22	-
	Double	80851.3	5.64	52.85	-	298.09	-
	Quad	301294.4	6.23	191.56	-	1193.43	-
MS	Single	22478	4.24	14.67	−2.85	62.22	−17.28
	Double	77074.56	4.44	51.77	−2.06	229.84	−22.90
	Quad	289275.5	5.34	186.72	−2.53	997.06	−16.45

Columns 6 and 8 are the power and energy gain with respect to the corresponding baseline case. Finally, it must be noticed that in the multispeculative implementations (*MS*) $k = 4$ has been utilized for single and double precision, while $k = 8$ has been chosen for quadruple precision, respectively.

As it can be observed, the power percentage reduction ranges between 2% and 3%, while the energy gain ranges between 16.5% and 23%. The percentage reduction is less than in Tables II, III, and IV, as FP-MAC power and area are dominated by the

multiplier module. Nevertheless, the introduction of multispeculation in the adder and complementer allows a more energy efficient design, at the same time it achieves a faster and smaller design. Our results show that the delay reduction ranges between 14.3% and 21.3%, while the area decrease is about 4%–5%.

5.2.4. Exascaling the FP-MAC. Finally, let us give an estimation about the benefits that our proposed FP-MAC may achieve in an exascale system such as the one presented in Kogge's report [Kogge 2008]. The reported FPUs account for 21% of the total 70MW, that is, 14.7 MW. The arithmetic operation alone, that is, the FP-MAC, consumes 45.5% of these 14.7 MW, that is, 6.7 MW, while the register file and the RAM memories consume the rest. Thanks to our ultra-low power addition stage, the double precision FP-MAC consumes 22.9% less energy than the baseline case. In other words, with our proposal the power budget can be diminished by 1.53 MW. Therefore, although this is not enough for meeting the proposed 20MW exascale power budget yet, it can contribute to do it.

Our results were achieved using 65 nm technology. However, exascale systems are planned for finer process technologies, for instance, 22 nm or lower. Hence, power and delay will be scaled down because of physical parameters [Huang et al. 2011]. Nevertheless, our techniques have been developed regardless of the technology. In the work presented in this article, the chosen process technology was only used to demonstrate the power and delay results for a given design parameter. But our proposed new ideas do not rely on the target technology and could easily be extended for more advanced device parameters as they become available.

6. CONCLUSIONS

This article presents a novel technique for developing ultra-low-power Floating Point Units, based on the application of multispeculative ideas to the addition stage of the FP-MAC. The adder is divided into several small fragments that initially assume that their carry-in signals are '0'. Afterwards, the possible mispredictions are corrected during the two's complementer operation. The two's complementer is also fragmented, while the correct carries are provided by the Corrected Carry Tree Anticipator (CCTA), which starts its computation while the adder is finishing its calculus. Dividing both the adder and the two's complementer into small chunks produces a significative area and power reduction, since the conventional implementations are based on fast logarithmic adders, that are large and power hungry when the bitwidth is wide. Moreover, the CCTA is an efficient structure, since its size depends on the number of fragments and it works as a Kogge-Stone adder without the first and last stages, which are the most power consuming ones.

The only limitation for applying this multispeculative s is the determination of the sign before the two's complement operation. In order to overcome this problem, a sign predictor is proposed instead of a costly mantissas comparator, as in the state of the art more aggressive designs. This predictor is based on the fact that only few cases produce an uncertain sign. Experimental results show its accuracy even in its basic form. Moreover, if more accuracy is required, an automatic scheme is proposed for increasing it rapidly. According to our results, with only 3 bits of additional information, the hit rates are greater than 0.99 even in the worst cases.

Finally, the energy efficiency of our implementation is demonstrated by synthesizing a whole FP-MAC for every precision. Our proposed design reduces energy by 17.5%–23%, with an additional 14.5%–21.5% delay and 4%–5% area decreases.

In the future, we need to evaluate where to introduce the pipeline registers in order to cope with the frequency of the target core, and study possible optimizations for the multiplier and normalizer modules, that according to our experiments have proved to be really power hungry.

REFERENCES

E. M. Ashmila, S. Dlay, and O. Hinton. 2005. Adder methodology and design using probabilistic multiple carry estimates. *IEE Proc. Computer Digital Tech.* 152, 6, 697–703.

C. Bienia. 2011. Benchmarking modern multiprocessors. Ph.D. Thesis, Princeton University.

C. Bienia and K. Li. 2010. Characteristics of workloads using the pipeline programming model. In *Proceedings of the 3rd Workshop on Emerging Applications and Many-core Architectures*. IEEE, 161–171.

C. Bienia, S. Kumar, J. P. Singh, and K. Li. 2008. The PARSEC Benchmark Suite: Characterization and architectural implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*. 72–81.

S. Chatterjee, L. R. Bachega, et al. 2005. Design and exploitation of a high-performance SIMD floating-point unit for Blue Gene/L. *IBM J. Res. Dev.* 49, 2, 377–392.

A. A. Del Barrio, R. Hermida, S. O. Memik, J. M. Mendías, and M. C. Molina. 2012. Multispeculative addition applied to datapath synthesis. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 31, 12, 1817–1830.

M. Ercegovac and T. Lang. 2003. *Digital Arithmetic*. 1st Ed. Morgan Kaufmann.

D. Fan, H. Zhang, D. Wang, et al. 2012. Godson-T: An efficient many-core processor exploring thread-level parallelism. *IEEE Micro* 32, 2, 38–47.

R. W. Hamming. 1970. On the distribution of numbers. *Bell Syst. Tech. J.* 49, 8, 1609–1626.

R. A. Haring, M. Ohmatch, T. W. Fox, et al. 2012. The Blue Gene/Q Compute chip. *IEEE Micro* 32, 2, 48–60.

L. Huang, S. Ma, L. Shen, Z. Wang, and N. Xiao. 2012. Low-cost binary128 floating-point FMA unit design with SIMD support. *IEEE Trans. Comput.* 61, 5, 745–751.

W. Huang, K. Rajamani, M. R. Stan, and K. Skadron. 2011. Scaling with design constraints: Predicting the future of big chips. *IEEE Micro* 31, 4, 16–29.

IEEE. 2008. 754-2008: IEEE standard for floating-point arithmetic. IEEE Standards, 1–58.

R. M. Jessani and M. Putrino. 1998. Comparison of single and dual-pass multiply-add fused floating-point units. *IEEE Trans. Comput.* 47, 9, 927–937.

R. Jessani and C. Olson. 1996. The floating-point unit of the PowerPC 603e. *IBM J. Res. Dev.* 40, 5, 559–566.

P. Kogge. 2008. The challenges of petascale architectures. *Comput. Sci. Eng.* 11, 5, 10–16.

P. Kogge, K. Bergman, S. Borkar, et al. 2008. ExaScale computing study: Technology challenges in achieving exascale systems. www.cse.nd.edu/Reports/2008/TR-2008-13.pdf

I. Koren. 2002. *Computer Arithmetic Algorithms*. 2nd Ed. AK Peters.

T. Lang and J. D. Bruguera. 2004. Floating-point multiply-add-fused with reduced latency. *IEEE Trans. Comput.* 53, 8, 988–1003.

T. Lang and J. D. Bruguera. 2005. Floating-Point Fused Multiply-Add Reduced Latency for Floating-Point Addition. In *Proceedings of the 17th Symposium on Computer Arithmetic*. 42–51.

N. Leavitt. 2012. Big iron moves toward exascale computing. *IEEE Computer* 45, 11, 14–17.

S. L. Lu. 2004. Speeding up processing with approximation circuits. *IEEE Computer* 37, 3, 67–73.

Z. Luo and M. Martonosi. 2000. Accelerating pipelined integer and floating-point accumulations in configurable hardware with delayed addition techniques. *IEEE Trans. Comput.* 49, 3, 208–218.

R. K. Montoye, E. Hokenek, and P. W. Cook. 1990. Design of the IBM RISC System/6000 floating-point execution unit. *IBM J. Res. Dev.* 34, 1, 59–70.

S. M. Nowick. 1996. Design of a low-latency asynchronous adder using speculative completion. *IEE Proc. Computer Digital Tech.* 143, 5, 301–307.

NVIDIA. 2012. www.nvidia.com.

V. G. Oklobdzija. 1992. An implementation algorithm and design of a novel leading zero detector circuit. In *Proceedings of the 26th IEEE Asilomar Conference on Signals, Systems and Computers*. 391–395.

J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips. 2008. GPU computing. *Proc. IEEE* 96, 5, 879–899.

R. V. K. Pillai, D. Al-Khalili, and A. J. Al-Khalili. 1998. On the distribution of exponents differences during floating point addition. In *Proceedings of the IEEE Canadian Conference on Electrical and Computer Engineering*. IEEE, 105–108.

S. K. Raman, V. Pentkovski, and J. Keshava. 2000. Implementing streaming SIMD extensions on the Pentium III processor. *IEEE Micro* 20, 4, 47–57.

M. S. Schmookler and K. J. Nowka. 2001. Leading zero anticipation and detection: A comparison of methods. In *Proceedings of the 15th Symposium on Computer Arithmetic*. 7–12.

SOFT-FP. 2012. http://gcc.gnu.org/wiki/Software_floating_point.

D. Tan, C. E. Lemonds, and M. J. Schulte. 2009. Low-power multiple-precision iterative floating-point multiplier with SIMD support. *IEEE Trans. Comput.* 58, 2, 175–187.

TOP500. 2012. http://www.top500.org/.

S. R. Vangal, Y. V. Hoskote, N. Y. Borkar, and A. Alvandpour. 2006. A 6.2-GFlops floating-point multiply-accumulator with conditional normalization. *IEEE J. Solid State Circuits* 41, 10, 2314–2323.

S. R. Vangal, J. Howard, G. Ruhl, et al. 2008. An 80-tile sub-100-W TeraFLOPS processor in 65 nm CMOS. *IEEE J. Solid State Circuits* 43, 1, 29–41.

A. Verma, A. K. Verma, P. Brisk, and P. Ienne. 2009. Hybrid LZA: A near optimal implementation of the leading zero anticipator. In *Proceedings of the Asia and South Pacific Design Automation Conference*. 203–209.

A. K. Verma, P. Brisk, and P. Ienne. 2008. Variable latency speculative addition: A new paradigm for arithmetic circuit design. In *Proceedings of the Conference and Exhibition on Design, Automation and Test in Europe*. 1250–1255.

S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. 1995. The SPLASH-2 programs: Characterization and methodological considerations. In *Proceedings of the 22nd International Symposium on Computer Architecture*. 24–36

Received December 2012; revised May 2013, August 2013; accepted August 2013