# An Efficient Multiple Precision Floating-Point Multiply-Add Fused Unit

K. Manolopoulos[a], D. Reisis[a,*], V.A. Chouliaras[b]

[a] *Department of Physics, Electronics Laboratory, National Kapodistrian University of Athens, Greece*
[b] *Department of Electronics and Electrical Engineering, Loughborough University, UK*

**Abstract**

Multiply-Add Fused (MAF) units play a key role in the processor's performance for a variety of applications. The objective of this paper is to present a multi-functional, multiple precision floating-point Multiply-Add Fused (MAF) unit. The proposed MAF is reconfigurable and able to execute a quadruple precision MAF instruction, or two double precision instructions, or four single precision instructions in parallel. The MAF architecture features a dual-path organization reducing the latency of the floating-point add (FADD) instruction and utilizes the minimum number of operating components to keep the area low. The proposed MAF design was implemented on a 65nm silicon process achieving a maximum operating frequency of 293.5 MHz at 381 mW power.

*Keywords:* floating-point, Multiply-Add Fused, multiple precision, VLSI

## 1. Introduction

Widely used applications such as graphics, transforms, digital filters, to name a few, impose the need for Floating-Point calculations, which consequently are a feature of all high performance computers, digital signal processors (DSPs) and graphic accelerators [1], [2], [3], [4]. Moreover, applications in the area of 3D graphics require the parallel execution of floating point

*Corresponding author. Tel.:+302107276720, fax:+302107276801
Email address: dreisis@phys.uoa.gr (D. Reisis).
Addr.: Panepistimiopolis, Physics Buildings IV & V, Athens, Greece, 15784

operations, which usually involve operands with single and/or double precision format. To meet these demands, a significant number of novel processor designs provides Floating-Point Units (FPUs), which support not only single but also double precision operations.

While the two most common floating-point formats, namely single and double precision, are satisfactory for the vast majority of computational applications, there are a few emerging applications where they are not adequate. Examples have been identified in the fields of climate modeling and computational physics [6], [7], [8], among others, that require even higher precision. To support such applications, IEEE has now included a standard for 128-bit ("quad") precision as part of its IEEE 754-2008 Standard for Floating-Point Arithmetic [10]. Quadruple precision computations can be handled either by software or hardware, but hardware solutions are more effective in terms of performance. Most of the aforementioned applications involve algorithms utilizing the $(A \times B) + C$ single-instruction equation, which is realized in general purpose processors (GPPs) by combining a floating-point multiplier and a floating-point adder into a Multiply-Add Fused (MAF) unit. Due to the importance of MAF instructions, many GPPs include embedded MAF units to speed-up the execution of SIMD instructions, while others use them to replace entirely the floating-point adder and multiplier units [11], [12], [13]. Furthermore, the importance of combining high precision floating point operations and the MAF instruction can be assessed by the efforts of major manufacturers towards providing commercially available solutions. Notable examples are first, the AMD "Bulldozer" microarchitecture which provides two 128-bit MAF cores per module, supporting single, double and extended precision format [14]; second, the IBM z13 microprocessor which performs MAF instructions for single, double and quadruple precision formats [15].

Aiming at improving the MAF performance, this paper presents a multifunctional, multiple precision MAF architecture based on a dual-path organization, which operates efficiently in quadruple, double or single precision. The proposed design can perform one quadruple precision MAF operation, or two double precision operations in parallel, or four single precision operations in parallel. Moreover, it can also perform –in each precision mode– multiple stand-alone multiply or add operations.

The motivation for this work was first, to present a MAF architecture supporting a variety of functions without significantly increasing area and delay; second, to investigate how the dual-path algorithm can efficiently be used to provide a MAF design with multiple functionalities; and third, to

evaluate its performance by identifying the hardware cost and the critical path delays. For sake of comparison, this paper also presents the synthesis results of implementing four MAF architectures by using the same technology with the proposed MAF: first, a typical double precision MAF architecture; second, a dual-mode, dual-path MAF unit that operates in both double and single precision; third, a typical quadruple precision; and fourth, a multi-block design including a typical quadruple precision MAF and two dual-mode dual-path MAF units.

The paper is organized as follows: Section 2 reviews the related work in the area of floating-point Multiply-Add Fused units. Section 3 describes the structure of a conventional MAF unit. Section 4 presents the architecture of the proposed multiple precision floating-point MAF unit and describes in detail the basic modules of the MAF design. Section 5 shows the area and delay results and finally, Section 6 concludes the paper.

## 2. Related Work

The Fused Multiply-Add unit has been introduced in the IBM POWER1 processor (1990), also known as RS/6000 [16]. Since then, several designs have been proposed and realized in an effort to improve the efficiency of the MAF architecture. The authors of [17] compare the design complexity of single or dual-pass (through the multiplier array) when realizing a fused multiply-add floating-point unit. In [18] a MAF floating-point unit with signed digit addition is presented: a signed digit addition along with a two step normalization method reduces the latency of the addition. [19] presents a floating-point MAF unit that computes floating-point addition with lower latency than floating-point multiplication and MAF. This is accomplished by bypassing, in case of an addition, the pipeline stages related to the multiplication. [20] introduces a two data-path MAF floating-point unit; and based on whether a subtraction occurs, it activates only one of the paths to reduce the average latency. [21] demonstrates a MAF floating-point unit able to process denormalized numbers at the cost of slightly increased hardware complexity and operation latency. [22] presents a fully pipelined MAF unit combining the final addition with rounding. In [23] a bridged MAF design is presented. This architecture includes a common floating-point multiplier and adder and by adding specific hardware components between them, creates a "bridge" sending data from the multiplier unit to the adder in order to perform the MAF instruction. More complex architectures are presented in [24] and [25]:

[24] proposes a dual-mode floating-point MAF unit with the SIMD feature executing instructions of either double or single precision and [25] describes a multi-functional double and quadruple precision floating-point MAF unit.

In addition to the above MAF designs there are several other architectures presented in the literature, which involve stand-alone floating-point multipliers and adders. Notable examples of relevant research are published in [26], [27], [28], [29]. The authors in [26] present a floating-point multiplier executing addition and rounding in parallel. In [27] a dual-precision multiplier is proposed. It utilizes half-sized multiplication arrays to perform a double precision multiplication in three cycles or a single precision multiplication in two cycles. [28] presents two designs performing dual-mode floating-point multiplication. The first architecture executes either one quad-precision multiplication or two double precision multiplications in parallel. The second design performs one double precision multiplication or two single precision multiplications in parallel. In [29] two architectures performing dual-mode floating-point addition are presented. The first is based on the single-path algorithm [30] and supports a double precision or two single precision additions in parallel. The second design implements the two path algorithm [30] and performs either a quad-precision addition or two double precision additions in parallel.

Compared to the literature above, our proposed architecture is the only design, which provides triple functionality allowing it to operate in all the precision modes, while being capable of performing multiple operations.

## 3. Conventional MAF Unit

This section highlights the IEEE floating-point standard format [10] and a typical Multiply-Add Fused unit. According to the IEEE standard for binary floating-point arithmetic a floating-point number requires three fields: sign $S$, biased exponent $E$ and fraction $F$. Fig. 1 depicts the fields of the different precision formats and the following equation shows the value X of a normalized floating-point number:

$X = (-1)^S \times 2^{E-bias} \times 1.F$

A brief description of a typical double precision MAF architecture (Fig. 2) included in several general purpose processors [17], [18], [21], [22], [24], is given in the following paragraphs.

The IEEE representation of a double precision floating point number requires 1-bit as a sign, 11-bit for the biased exponent and 53-bit fraction.

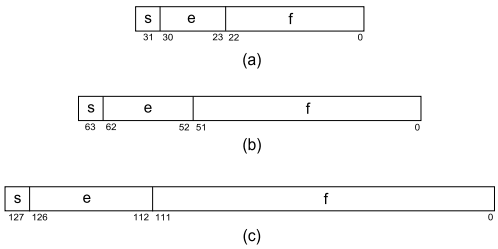


Figure 1: (a) Single precision format, (b) Double precision format and (c) Quadruple precision format.

The execution of a MAF instruction is accomplished through a number of serial steps, that are categorized in three stages:

1. *Multiplication Stage*: the first stage multiplies the 53-bit significands of the operands A and B. At the same time, the third operand C is inverted and aligned. This is performed in parallel with the multiplication, in order to reduce the latency, with the addend C positioned 56 bits left of the product. Next, the exponent difference is calculated: $diff = expC - (expA + expB - 1023)$ where expA, expB, expC are the biased exponents of the operands A, B and C respectively and 1023 is the bias for IEEE754 double precision arithmetic. Hence, the necessary shift amount equals:
   $sh = 56 - diff \Rightarrow sh = expA + expB - expC - 967$
   Then, the alignment is performed as a right-shift by the 161-bit aligner.
2. *Addition Stage*: the second stage adds the 106-bit product $A \times B$ and the 161-bit aligned operand C by using a 106-bit 3:2 CSA and a 161-bit adder. In parallel with the addition, the Leading Zero Anticipator (LZA) unit processes the lower 106-bits of the 3:2 CSA and it computes the necessary shift amount for normalization. If it is required, the adder result will be complemented before forwarding the result to the last stage.
3. *Normalization and Rounding*: the third stage performs normalization and rounding to produce the final result. The 161-bit result is normalized based on the estimation of the LZA unit. Next, rounding is performed and, if required, a post-normalization. At the same time, the sign and the exponent of the final result are prepared.

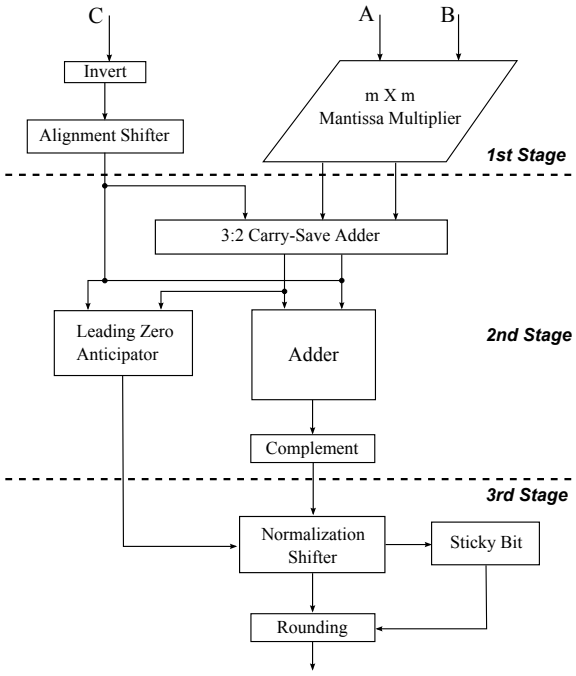


Figure 2: Block diagram of a basic MAF architecture.

## 4. Proposed architecture overview

The motivation behind this work has been to research and design an efficient MAF architecture able to operate in different precision modes while being capable of performing multiple instructions in an SIMD fashion. We follow a design approach leading to an architecture with high performance and reduced latency, at relatively modest hardware cost. We have designed the MAF unit to support three different precision modes: quadruple, double and single precision. In each of these modes the design performs a number of different operations: MAF instruction, stand-alone multiplication, or stand-alone addition.

Our study towards improving the latency of the MAF unit led us to follow the multiple-path approach [30]. The multiple-path design strategy divides the critical path into several paths according to the exponent difference. The resulting architecture allows each path to carry out a less complex scenario and therefore, to keep the delay at lower levels compared to the traditional implementations. The multiple-path approach is widely used to speedup the FP adder [31] and MAF (Multiply-Addition Fused) [32], [33] effectively.

Based on the above idea, the current work follows a dual-path design strategy to reduce the latency of the floating-point addition with respect to the standard MAF implementations. Its functionality is based on combining the final addition with rounding by using a dual-adder at the final stage, while anticipating the normalization before that addition. This scheme was initially utilized in floating-point addition, in order to take advantage of the fact that the operations of full length alignment and normalization shifts are mutually exclusive. Therefore, only one full length shifter appears on the critical path.

To achieve the above mentioned objectives the proposed design incorporates several schemes:

1) The design follows a dual-path approach [19] to reduce the latency of instructions that involve stand-alone floating-point additions. The dual-path scheme divides the datapath of the add stage into two different paths: the *Close* path and the *Far* path. The *Close* path handles cases where massive cancellation occurs, while the *Far* path handles the remaining cases. The idea is to reduce the latency of the FP ADD instruction by bypassing the multiplier and by placing the alignment shifter at the next stage. The architecture avoids the duplication of the alignment shifter in both paths by taking advantage of the fact that full-length alignment and full-length normalization shifts are mutually exclusive operations and therefore, each datapath needs to utilize only one full length shifter. Hence, the *Far* path requires only one large alignment shifter (and a much smaller normalization shifter) and the *Close* path requires only one large normalization shifter (and a small alignment shifter). Avoiding the two large shifters from both datapaths leads to reduced critical delay and area.

2) In order to further improve the efficiency of our design we avoid the implementation of a typical dual-path architecture and by following the approach of [35] the multiplier results are added at the beginning of the second stage. Thus, we manage to significantly reduce the area of the modules following this addition and more importantly to reduce the size of the alignment shifters, which incur significant overheads on area and delay in prior designs. The proposed approach however, follows a standard 3 stage implementation (instead of the 4 stage implementation of [35]), in which both the *Close* and *Far* paths are extended to the third stage. The *Far* path performs normalization in the second stage, before the final addition and rounding, while the *Close* path resembles that of a typical MAF design performing normalization and rounding in the final stage.

3) The proposed dual-path design is based on a quadruple precision MAF unit and it is augmented to support the execution of two double precision instructions or four single precision instructions in SIMD fashion. For this reason, several components have been carefully redesigned and additional logic has been introduced. We note here that, although we have used additional hardware resources, the area results are still improved compared to the approach of merely duplicating the components for each precision mode.

4) The architecture utilizes three 128-bit registers. Each of these registers -depending on the operation mode- can be used to store one quadruple, two double, or four single precision operands. The design is a three-stage pipeline and has a throughput of one result per cycle.

Fig. 3 illustrates the block diagram of the proposed multiple precision floating-point MAF architecture. The remaining of this section describes the overall functionality of the proposed design and provides a detailed analysis of the design's behavior in each precision mode.

### *4.1. Quadruple precision mode*

The quadruple precision (QP) mode operation involves the execution of any of the following three: the MAF single instruction equation, quad multiplication and quad addition. The following paragraphs describe the functionality of the three stages in this mode.

*First stage*: calculations begin by transforming the operand mantissas to the form: ['1', R1(111:0)] and ['1', R2(111:0)] in order to include the hidden bits. Following the above operation, the exponent processing module calculates the exponent difference, the necessary shift amount and control signals. In the cases of executing MAF instructions or quad multiplications, the multiplication exponent processing is performed in parallel with the multiplication. The array multiplier (Fig. 3, 1st stage) performs the modified mantissas multiplication. For quad-precision the multiplier operates as a typical $113 \times 113$ bit array multiplier producing the results in carry-save format.

In the second and third stage, the datapath is divided into two paths: the *Close* and the *Far* path. Only one path is activated depending on the outcome of the exponent difference. The *Close* datapath is activated in the cases of effective subtraction with exponent difference $diff = -1, 0, 1$, or in the case of effective subtraction with exponent difference $diff = 2$ and multiplication overflow. For the remaining cases the *Far* datapath is activated.

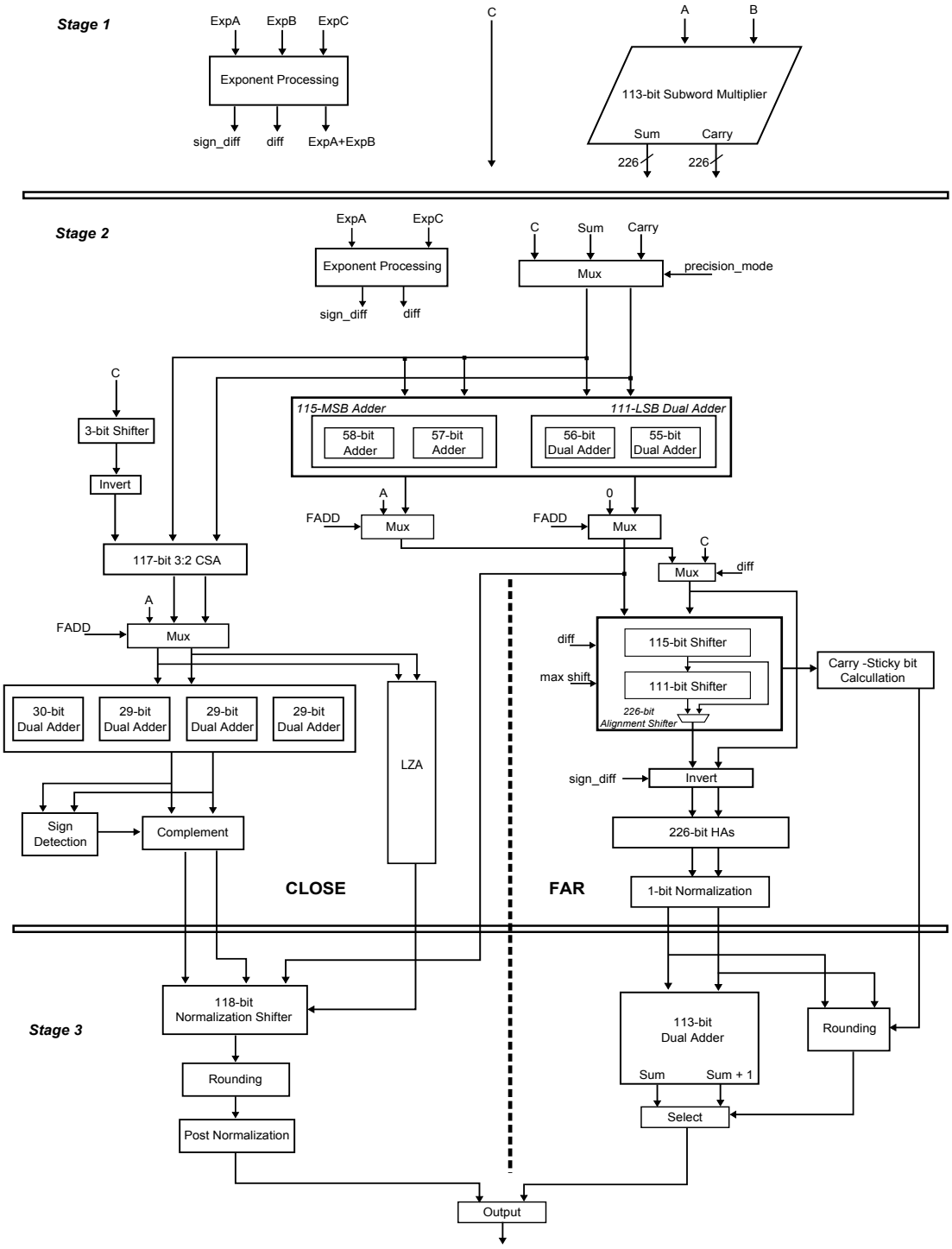


Figure 3: Block diagram of the proposed MAF architecture.

*Second stage*: For stand-alone quadruple addition we bypass the first stage and we use the duplicated exponent processing module at the beginning of the second stage. In all other cases, the second stage receives as input the multiplier products $A \times B$ along with the third operand $C$, which are both outputs of the first stage. Depending on the exponent difference, the $A \times B$ and $C$ either follow the *Close* or the *Far* path. Regardless of the chosen path, the multiplier products are added using four adders: two dual adders of 58- and 57-bit, and two 56- and 55-bit adders. The dual adders are also used to calculate in advance the complement of the addition, which will be required, if the result from the *Close* path is negative. In quad-precision mode all four adders perform as a single device; in other precision modes they operate as individual adders. By adding the multiplier results we significantly reduce the size of the alignment shifters and thus, we reduce the overall area.

The *Close* path performs alignment of the operand C by shifting it at most 1-bit right or 2-bit left using the 3-bit shifter. Hence, for quad precision mode the aligned operand is 116 bits (Fig. 4). Following the alignment operation, operand C is always inverted to ensure a positive result and in case of a negative result the sign is inverted again at the end of the second stage. Note that, a 2's complement representation is preferred, because it avoids the end-around carry adjustment. Next, the operand C and the multiplier results are forwarded into a 117-bit 3:2 CSA to obtain a redundant representation of the intermediate vectors. To complete the 2's complement an additional 1 is added using an empty slot of the 3:2 CSA. The two resulting vectors are then added by using four dual adders. In quad precision mode these adders are combined together, while in other precision modes they are used separately in order to accommodate parallel additions. To avoid additional delays in the *Close* path, the LZA operation is performed in parallel with the addition. The LZA unit was designed following the method presented in [36]. A sign detection unit determines the correct sign based on the addition results and it complements if necessary.

The *Far* path performs word-length shift operations in order to align either $A \times B$ or C based on the sign of the exponent difference. If $diff > 0$ the $A \times B$ will be right-shifted by at most 115 bits and if $diff \leq 0$ the C will be right-shifted by at most by 226 bits (Fig. 4). Since the multiplier results have been added at the beginning of the second stage, there are only two vectors to be handled and the alignment can be performed by using one shifter instead of two. This shifter is shared by vectors $A \times B$ and $C$ and it is used to align one of them depending on the case. For this purpose,

instead of using a 226-bit shifter we utilize two smaller shifters of 115 and 111 bits. The first shifter is used to align vector $A \times B$ and both shifters are used to align vector C. During the alignment operation the shifted-out bits are used for the calculation of the sticky bit. In the next section we further analyze the internal structure of these shifters and explain how they are designed to accommodate the parallel alignment operations in different precision modes. If the alignment operation provides a negative result, one of the aligned vectors $A \times B$ and $C$ will be inverted in order to obtain positive results. The following function forwards the vectors into a row of Half Adders (HAs): an empty slot in that row is used to add the necessary 1 to complete 2's complement representation of the inversion. In the *Far*-path, the normalization is performed before the addition. This is accomplished by a 2-bit normalization shifter handling a possible 1-bit right or 1-bit left shift operation to correct a possible overflow or an unnormalized result.

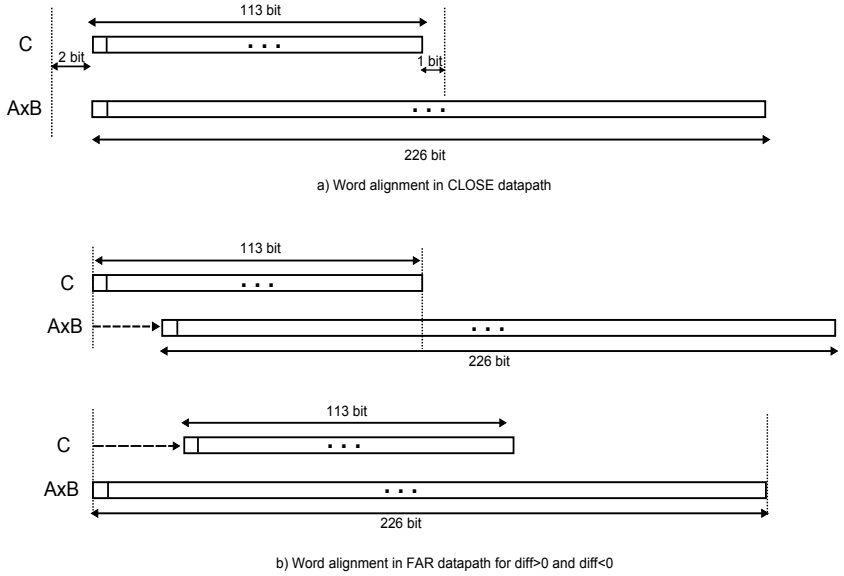


Figure 4: Alignment in the *Close* and *Far* datapath.

*Third stage*: the proposed design features distinct functionalities for the *Close* and the *Far* paths in the third stage, in contrast to the traditional dual path approach [19] suggesting two paths only in the second stage. The third stage includes the rounding step and it supports all the rounding modes described by the IEEE standard. The *Close* path in this stage performs normalization and rounding. Since a full length normalization is required,

a 118-bit shifter is used for normalization. As it will be explained in the following section, this shifter is a combination of smaller shifters in order to accommodate the double and single precision operations as well. Note here that, the LZA operations performed previously in the second stage can cause a 2-bit uncertainty: to overcome this problem, in the third stage we perform rounding and post-normalization to calculate and determine the correct result, which is then forwarded to the output.

The *Far* path performs the final addition and rounding. The upper 113 bits of the data vector from the previous stage are used to perform dual addition while the remaining bits are forwarded to the rounding module. Two dual adders (58- and 53-bits) are combined together to compute sum and sum+1. At the same time, the rounding module uses the least significant bits along with the sticky bit and the rounding bit to calculate the rounding result. Based on the latter, the correct output is selected.

### *4.2. Double and Single precision mode*

The proposed MAF operates in double or in single precision mode as described in the previous section (4.1). This section focuses on how the specific components of the architecture can accommodate the different precision formats.

In double precision mode (DP) the proposed design can accommodate the parallel (SIMD) execution of two MAF instructions, or the parallel (SIMD) execution of two double precision floating-point multiplications or additions. In DP mode each register contains two double precision operands.

Operating in single precision mode (SP) the design can accommodate the parallel (SIMD) execution of four single instruction equations, or the parallel (SIMD) execution of four single precision floating-point multiplications or additions, while in SP mode each register contains four single precision operands.

*Exponent processing*: In order to attain the parallel execution of double and single precision instructions, one double and two single precision exponent processing units have been added. Two exponent processing operations belonging to two different double precision MAF instructions can be performed in parallel: the first in the exponent processing unit made for quad precision and the second in the additional double precision unit. In the single precision mode, each exponent processing unit (the quad, the double and the two single) is used to support a single precision instruction.

*Multiplication*: The multiplier of the proposed MAF performs simultaneously two 53-bit multiplications in double precision, or four parallel 24-bit multiplications in single precision. As mentioned in the previous section (4.1), one array multiplier is used to accommodate all three precision modes. The choice of an array multiplier over a Booth encoding design is justified by the following: first, Booth encoding requires more complex control logic in order to handle three precision formats, which in turn would introduce additional latency to the design. Second, in Booth encoding any subword carry bits, in double and single precision mode have to be detected and suppressed in the reduction tree and CPA [24]. Therefore, even though the array multiplier requires a larger compression tree, it is far more suitable for a multiple precision MAF design than the Booth multiplier [21], [22], [35].

Fig. 5a shows how the two double precision multiplications can be performed in parallel. The four mantissas, extended with the hidden 1, are input to the multiplier and the products $A_1 \times B_1$, $A_2 \times B_2$ are computed within the two $53 \times 53$ submatrixes $P_1$ and $P_2$ by using the subword parallel multiplication [34]. The bits of the partial products within the regions labeled $Z$ are set to zero to avoid altering the final results. Moreover, as shown in Fig. 5b the multiplier can also perform in parallel four single precision multiplications by applying the same technique as in the case of double precision mode.

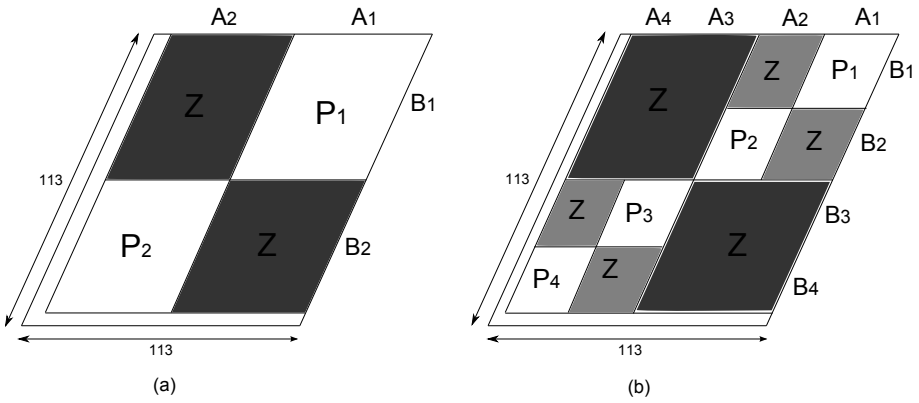


Figure 5: Subword parallel multiplication.

*Alignment and normalization shifters*: The alignment shifter used in the *Far* path is able to perform parallel alignment shift operations. When the design operates in double precision mode (DP) the maximum alignment shift of vectors $A \times B$ and $C$ is 55 bits and 106 bits respectively, while in single

precision mode (SP) it is 26 bits and 48 bits respectively. Furthermore, in DP mode the design has to perform at most two parallel shift operations, while in SP mode is has to accommodate four alignment shifts.

As described in the previous section (4.1), the alignment shifter of the *Far* path consists of two shifters of 115 and 111 bits. However, in order to support every precision mode these shifters are further divided. The 115-bit shifter is divided in two shifters with size 60-bit and 55-bit, while the 111-bit shift operation is accomplished by combining two smaller shifters with size 65-bit and 56-bit. Hence, in DP mode the first two shifters are combined to perform the alignment of the first instruction, and the second pair of shifters form a single device for the second instruction. Finally, in SP mode each of the four small shifters is used to accommodate the alignment operation of a single instruction.

The normalization shifter in the *Close* path has a similar design to the alignment shifter in the *Far* path. The 118-bit normalization shifter of the *Close* path consists of four shifters (two 30-bit and two 29-bit shifters). In DP mode these shifters are combined in pairs to perform two parallel normalization operations, while is SP mode they operate as stand-alone normalization shifters.

## 5. Implementation results and analysis

The proposed architecture has been designed and implemented in VHDL and the design functionality has been verified through simulation analysis. For comparison purposes besides the presented design, we have also implemented four more MAF architectures. The first is a typical double precision MAF unit similar to that described in section 3. The second is a dual-mode, dual-path MAF design [19], which performs one double precision MAF instruction, or two single precision instructions in parallel. The third is a typical quad precision MAF unit. The fourth is a multi-block design including a typical quadruple precision MAF and two dual-mode dual-path MAF units.

All the designs were synthesized in the Faraday/UMC 65 nm Library (10M layers) targeting the LL-RVT (LowK) UMC process. Worst-case conditions (WCCOM, $V_{CC}$=1.08V, 125C) were selected for front-end synthesis with Cadence RTL Compiler (RC) V10.10. The designs were constrained for 500 MHz operation (2000 ps period), with input and output delays set at half that value (1000 ps). For the clock uncertainty, 100 ps were also

Table 1: MAF designs post-layout results

| | Typical DP | Dual mode | Typical QP | Multi Block | Prop. MAF |
|---|---|---|---|---|---|
| Area ($um^2$) | 119668 | 149343 | 515123 | 812952 | 672046 |
| AND2 | | | | | |
| Eq. Gates | 74792 | 93339 | 321952 | 508095 | 420028 |
| Power (mW) | 246.9 | 268.9 | 295 | 320 | 381 |
| Fmax (MHz) | 444.24 | 427 | 343 | 326 | 293.5 |

factored in. The retiming and auto-ungrouping features of RC were used, as well as low-power implementation techniques such as operand isolation and clock-gating. Front-end synthesis was RTL-simulation-activity-driven with the activity produced by 1000 RTL testbench test vectors used to drive dynamic and leakage power optimizations. The spatial mode of RC was used, as the use of a reasonably recent process node rendered selection (and use) of wireload models invalid. Finally, to enable more back-end freedom, certain switches were enabled to allow for Total-Negative-Slack optimization. Note here that, following the final Place-and-Route flow the power was extracted from within the Cadence Encounter environment using statistical means. Thus, the average power consumption of all designs (Table 1) was established at the maximum (post-route) operating frequency of that design and the following activity factors were used: Input activity: 0.5; Dominant Frequency: Design max. Frequency; Flop activity: 0.5; Clock gate activity: 0.5. We note that the activity factors are very high; however, our study is consistent as all the MAF designs were measured at these activity levels. The post layout results for all designs are summarized in Table 1, while Table 2 shows the post layout results for the multiplier used in the proposed design. Table 3 shows the energy per operation results for all the implemented MAF designs. The figures of this table were produced by multiplying the power consumption with the period of the design to derive an "Energy-per-operation" metric. Fig. 6 shows the VLSI layouts of the three additional MAF architectures and Fig. 7 presents the VLSI layouts of the multi-block MAF and the proposed MAF design.

In order to evaluate the performance of the presented architecture we first highlight the main aspects of our design and compare them with other multiple precision architectures. Then, we assess our design's performance

Table 2: Proposed MAF multiplier results

| | Area ($um^2$) | AND2-Eq. Gates | Fmax (MHz) |
|---|---|---|---|
| Prop. Quad Multiplier | 389882 | 243676 | 341 |

Table 3: Energy per operation of the MAF designs

| | Typical DP | Dual mode | Typical QP | Multi Block | Prop. MAF |
|---|---|---|---|---|---|
| Energy per operation (pJ/op) | 555.52 | 618.47 | 855.5 | 979.2 | 1295.4 |

with respect to hardware cost and latency, by comparing it to the typical quad precision MAF and the multi-block MAF, while at the same time we present the area and latency results for the typical and the dual-mode MAF designs. Using these results, we finally consider the approach of utilizing multiple double or single precision MAF architectures and we compare the corresponding results to our single architecture solution.

The proposed MAF follows a modified dual-path approach, in which the *Close* path operates as a typical MAF architecture performing addition in the second stage and normalization and rounding in the third stage. The *Far* path however, anticipates normalization in the second stage, before the addition, while utilizing only one alignment shifter to accommodate the various shift operations. Moreover, our design combines the multiplier products at the beginning of the second stage to reduce the size of the vectorized modules and all the additions and alignment operations are performed by combining small scale adders and shifters. These features allow the accommodation of different precision modes and they maintain low latency. The comparison of the proposed design to existing dual-path architectures shows that the design in [19] requires two full length alignment shifters instead of one in our case, while the algorithm of [35] divides the *Far* path in two sub-paths and even though it reduces the latency (compared to [19]), it requires 4 stages and utilizes additional hardware for the *Far* path compared to our design.

Table 1 shows that compared to the typical quad precision architecture the proposed design demonstrates a 23% increase in area and a much smaller increase in latency 14%. The proposed MAF requires approximately four times the area of the dual mode implementation, which performs only one double and two single precision MAF instructions. Hence, in order to perform

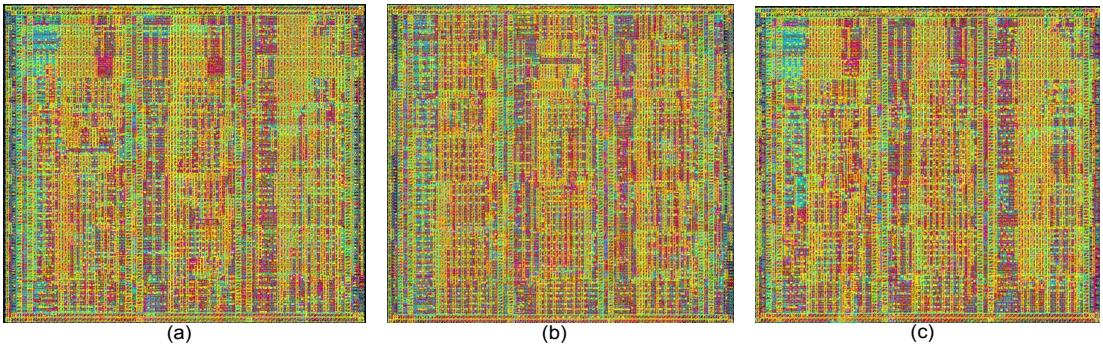


Figure 6: The VLSI layout of (a)the typical double precision MAF, (b)the dual-mode, dual-path MAF and (c)the typical quad-precision MAF design.

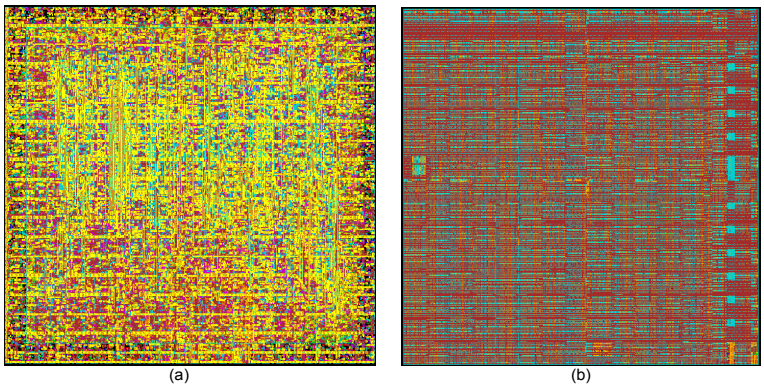


Figure 7: The VLSI layout of (a)the multi-block MAF design and (b)the proposed MAF design.

two double precision or four single precision instructions, we would need to utilize two dual-path or four typical MAF units, leading to a substantial increase in area, while still not being able to accommodate quad precision instructions.

To assess the benefit of using the proposed MAF we highlight the functionality and throughput capabilities of the MAF architectures used for the comparison:

1. *Typical QP MAF*: this unit can execute one QP instruction per cycle.
2. *The typical double precision MAF*: this unit can execute one DP instruction per cycle.

3. *Dual-mode, dual-path DP MAF unit*: it can execute one DP or two SP instructions per cycle, but does not support the execution of QP instructions.
4. *Multi-Block MAF*: it supports all types of floating-point precision and it can execute one QP instruction per cycle, two DP instructions per cycle or four SP instructions per cycle.
5. *Proposed MAF*: this unit supports all types of floating-point precision and it can execute one QP instruction per cycle, two DP instructions per cycle or four SP instructions per cycle.

As shown above, the proposed MAF has as equivalent only the multi-block MAF with respect to the functionality and throughput. The multi-block MAF requires less energy per operation (Table 3), but as Table 1 shows, the proposed design prevails in terms of area. Therefore, the proposed design is well suited for applications, which demand high throughput execution of multiple instructions and the performance of all floating-point precision modes, while at the same time the MAF design is limited by the area constraints.

## 6. Conclusion

The current paper presents a multi-functional, multiple precision MAF architecture based on a dual-path organization, which operates efficiently in quadruple, double and single precision format. The proposed MAF unit is designed to perform one quad, or two double, or four single precision MAF instructions in parallel. Moreover, the presented architecture features multiple stand-alone multiply or add operations in any precision mode.

The MAF unit introduced in this paper can execute more instructions than other presently available dual path architectures. For comparison purposes four additional MAF units have been implemented in the same technology: a typical double, a typical quad precision MAF architecture; a dual mode MAF unit preforming instructions in double and single precision format; and a multi-block MAF consisting of a typical quadruple precision MAF and two dual-mode dual-path MAF units. The area and performance results presented in our paper show that the proposed MAF architecture achieves an excellent balance of performance for a given area. These features benefit applications, by providing three different precision levels, which are executed at high throughput rates, in a compact package.

## References


[1] C. Hinds, "An Enhanced Floating Point Coprocessor for Embedded Signal Processing and Graphics Applications," IEEE Conference Record of the Thirty-Third Asilomar Conference on Signals, Systems, and Computers, pp. 147-151, 1999.

[2] D. Bossen, J. Tendler, K. Reick, "POWER4 system design for high reliability," IEEE Micro 22 (2002) 161-724.

[3] D. R. Lutz, C. N. Hinds, "A new floating-point architecture for wireless 3D graphics," Proceedings of the 38th Asilomar Conference on Signals, Systems and Computers, November 2004, pp. 1879171883.

[4] A. Naini, A. Dhablania, W. James, D. Sarma, "1 GHz HAL SPARC64 dual floatingpoint unit with RAS features, " Proceedings of 15th Symposium on Computer Arithmetic, 2001, pp. 173-183.

[5] R. Kolla, "The IAX Architecture: Interval Arithmetic Extension," Technical Report 225, Universitat Wurtxburg, 1999.

[6] D. H. Bailey, "High-Precision Arithmetic and Applications to Physics and Mathematics," International Symposium on Nonlinear Theory and its Applications (NOLTA2007), Sep 2007.

[7] Y. He, C. Ding, "Using accurate arithmetics to improve numerical reproducibility and stability in parallel applications," Journal of Supercomputing 18 (2001) 259-277.

[8] Y. Hida, X. S. Li, D. H. Bailey, "Algorithms for quad-double precision floating point arithmetic," Proceedings of 15th IEEE Symposium on Computer Arithmetic, 2001, pp. 155-162.

[9] D. H. Bailey, "High-Precision Floating-Point Arithmetic in Scientific Computation," Computing in Science and Engineering, vol. 7, no. 3, pp. 54-61, May-June, 2005.

[10] 754-2008 IEEE Standard for Floating-Point Arithmetic. Available from: http://www.ieee.com

[11] E.N. Linzer, "Implementation of Efficient FFT Algorithms on Fused Multiply-Add Architectures," IEEE Transactions on Signal Processing, Vol. 41, No. 1, pp. 93-107, Jan, 1993.

[12] H. Sharangpani, K. Arora, "Itanium Processor Microarchitecture," IEEE Micro Mag. Vol. 20, No. 5. pp. 24-43, 2000.

[13] Y. Voronenko and M. Puschel, "Automatic Generation of Implementations for DSP Transforms on Fused Multiply-Add Architectures," International Conference on Acoustics, Speech and Signal Processing, pp. V-101-4, 2004.

[14] "AMD64 Architecture Programmers Manual Volume 6: 128-Bit and 256-Bit XOP, and FMA4 Instructions," 2015, Available from: http://support.amd.com/TechDocs/43479.pdf

[15] IBM z13 Overview, 2015, Available from: http://www.ibm.com

[16] R. K. Montoye, E. Hokenek, S. L. Runyon, "Design of the IBM RISC System/6000 floating-point execution unit," IBM Journal of Research & Development, Vol. 34, pp. 59-70, 1990.

[17] R. M. Jessani, M. Putrino, "Comparison of Single- and Dual-Pass Multiply-Add Fused Floating-Point Units," IEEE Transactions on Computers, vol. 47 no. 9, pp. 927-937, 1998.

[18] C. Chen, L-A. Chen, J-R. Chen, "Architectural Design of a Fast Floating-Point Multiplication-Add Fused Unit Using Signed-Digit Addition," Proceedings of the Euromicro Symposium on Digital Systems Design (DSD'01), pp. 346, 2001.

[19] J. D. Bruguera, T. Lang, "Floating-Point Fused Multiply-Add: Reduced Latency for Floating-Point Addition," 2002 IEEE International Conference on Computer Design (ICCD'02), pp.145, 2002.

[20] H. Sun, M. Gao, "A novel architecture for floating-point multiply-add-fused operation," ICICS-PCM, vol.3, pp. 1675 - 1679, December 2003.

[21] H. He, Z. Li, "Multiply-add fused float point unit with on-fly denormalized number processing," 48th IEEE Midwest Symposium on Circuits and Systems, Vol. 2, pp. 1466 - 1468, August 2005.

[22] G. Li, Z. Li, "Design of A Fully Pipelined Single-Precision Multiply-Add-Fused Unit," 20th International Conference on VLSI Design (VLSID'07), pp.318-323, 2007.

[23] E. Quinnell, E. E. Swartzlander, C. Lemonds, "Bridge Floating-Point Fused Multiply-Add Design," IEEE Transactions on VLSI Systems, Volume 16, Issue 12, pp. 1727-1731, December 2008.

[24] L. Huang, L. Shen, K. Dai, Z. Wang, "A New Architecture For Multiple-Precision Floating-Point Multiply-Add Fused Unit Design," Proceedings of the 18th IEEE Symposium on Computer Arithmetic, pp. 69-76, 2007.

[25] M. Gok, M. M. Ozbilen, "Multi-functional floating-point MAF designs with dot product support," Microelectronics Journal, Volume 39, Issue 1, pp. 30-43, January 2008.

[26] W-C. Park, T-D. Han, S-D. Kim, S-B. Yang, "A floating point multiplier performing IEEE rounding and addition in parallel," Journal of Systems Architecture: the EUROMICRO Journal, Volume 45, Issue 14, pp. 1195-1207, July 1999.

[27] G. Even, S.M. Mueller, P-M. Seidel, "A dual precision IEEE floating-point multiplier," Integration, the VLSI Journal, Volume 29, Issue 2, pp. 167-180, September 2000.

[28] A. Akkas, M.J. Schulte, "Dual-mode floating-point multiplier architectures with parallel operations," Journal of Systems Architecture: the EUROMICRO Journal archive, Volume 52, Issue 10, pp. 549 - 562, October 2006.

[29] A. Akkas, "Dual-mode floating-point adder architectures," Journal of Systems Architecture: the EUROMICRO Journal, Volume 54, Issue 12, pp. 1129-1142, December 2008.

[30] M. Ercegovac, T. Lang, "Digital Arithmetic," Morgan Caufmann, 2004.

[31] Peter-Michael Seidel, Guy Even, "Delay-Optimized Implementation of IEEE Floating-Point Addition" IEEE Transactions on computers,Vol. 53, No. 2, February, 2004.

[32] P.M. Seidel, "Multiple Path IEEE Floating-Point Fused MultiplyAdd," Proceedings of the 46th IEEE International Midwest Symposium on Circuits and Systems, pp. 1359- 1362, 2003.

[33] E. Quinnell, E.E Swartzlander, C Lemonds, "Floating-Point Fused Multiply-Add Architectures," Signals, Systems and Computers, 2007. ACSSC 2007. Conference Record of the Forty-First Asilomar Conference on, pp.331-337, 4-7 Nov. 2007, doi: 1O.1109/ACSSC.2007.4487224.

[34] S. Krithivasan, M. J. Schulte, "Multiplier Architecture for Media Processing," Proc. 37th Asilomar Conf. Signals, Systems, and Computers, pp.2193-2197, 2003.

[35] Z. Qi, Q. Guo, G. Zhang, X. Li, W. Hu "Design of Low-Cost High-performance Floating-point Fused Multiply-Add with Reduced Power," 23rd International Conference on VLSI Design, 2010.

[36] M. S.Schmookler, K. J. Nowka, "Leading Zero Anticipation and Detection - A comparison of Methods," Proceedings of the 15th IEEE Symposium on Computer Arithmetic (ARITH15), pp.7-12, 2001.