Microprocessors and Microsystems 57 (2018) 23–31

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Efficient dual-precision floating-point fused-multiply-add architecture

V. Arunachalam[a,*], Alex Noel Joseph Raj[b], Naveen Hampannavar[c], C.B. Bidul[d]

[a] VLSI Division, School of Electronics Engineering, VIT University, Vellore, India
[b] Department of Electronic Engineering, College of Engineering, Shantou University, China
[c] Aceic Design Technologies, Bengaluru, India
[d] TATA Elxsi Ltd., Trivandrum, India

ABSTRACT

The fused-multiply-add (FMA) instruction is a common instruction in RISC processors since 1990. A 3-stage, 8-level pipelined, dual-precision FMA is proposed here that can perform operations either at one double precision (SISD) or at two single precision in parallel (SIMD). The 53-bit mantissa-multiplier (MM) is optimally segmented by Karatsuba–Offman (KO) algorithm such that both modes can be performed. The 6-stage pipelined MM uses only 6 of 10 multipliers and 13 of 33 adder/subtractors in SIMD. Thus hardware area of the proposed MM is reduced by 23.82% and throughput is maintained to be 923M samples/s. The arithmetic operational units in the data path are shared among the modes by having four data rearrangement units (DRU) which rearranges the data systematically at the input, the outputs of MM and the final output. Though these DRUs bring some hardware overhead, the resulting architecture is modular and uniform for both modes of computation. The proposed FMA has been implemented using TSMC 1P6M CMOS 130 nm library and takes 48% less overall area and consumes 49% less power at 308.7 MHz compared to previous results. The area-delay-product (ADP), 0.48×10^{-15} shows that the area optimization by proposed KO based MM can also keep the computation time as 3.24 ns.

1. Introduction

Modern multimedia processor instruction set architectures have novel instructions to meet the high accuracy and fast computation needs. One of such is fused multiply-add (FMA). A fast FMA can improve the performance of computations involving dot product calculation, matrix multiplication and polynomial evaluation. This executes the expression $(A \times B) + C$, with a single rounding, where the operands are in IEEE-754 standard floating-point representation. A multiply and accumulate (MAC) unit can be used to compute the expression. But it has rounding operation after multiply also after addition. It can also be computed using FMA, which combines addition and multiplication as a single computation step and hence rounding is required only once. This in turn improves accuracy of the computation with FMA.

The FMA unit was first proposed by Montoye et al. [1] in the IBM RISC System/6000. Since then, FMA has been used as a key feature in many processors as mentioned by several authors [2–5]. The FMA has been implemented as an instruction set in architectures like CELL and PowerPC of IBM, ARMv7-A and ARMv7-R from ARM, Piledriver and Bulldozer of AMD and in Haswell of Intel as mentioned by Wait [6], Shainer et al. [7] and Kurd [8].

A Three Path FMA and a Bridge FMA have been implemented in [9], but there is a trade-off between area and speed in each of these and they work only with double precision (DP). An improved version of FMA was implemented in [10] which supports both single precision (SP) and double precision (DP). It uses a modified dual path scheme to reduce latency. Though the area consumed is higher than the conventional DP FMA by 23%, the area overhead is justified by the increased throughput in parallel computing of two sets of SP inputs and also the delay is reduced by 13%. A mixed precision FMA has been implemented in [11], in which A and B inputs can be of a particular precision (say SP) while the input C and the result can be of another precision (say DP). The FMA architecture proposed in [12] supports one DP or two SP in parallel with increased throughput. The delay and area are increased by 9% and 18% respectively than the conventional DP FMA in [10].

This paper proposes an architecture for a FMA which can accept either one set of double precision inputs to give a double precision output (SISD) or take two sets of single precision inputs to give two single precision outputs concurrently (SIMD). Only one rounding is done at the final stage after the normalization as done in the conventional FMA unit.

The main contributions in this work are:

* Corresponding author.
E-mail address: varunachalam@vit.ac.in (V. Arunachalam).

https://doi.org/10.1016/j.micpro.2017.12.009
Received 13 November 2015; Received in revised form 28 November 2017; Accepted 17 December 2017
Available online 20 December 2017

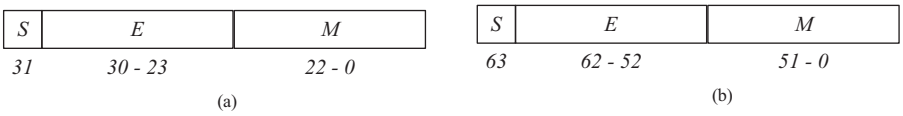

Fig. 1. (a) IEEE-754 Representation - single precision. (b) IEEE-754 Representation – double precision.

- Data rearrangement units (DRUs) are used to rearrange the data (inputs and outputs) and thus the proposed architecture for both modes, SISD and SIMD is more regular.
- The area of mantissa multiplier is reduced comparably by suitably designing KO based segmented multipliers and is reusable in both modes.
- The optimal KO segmentation (word length of smaller multiplier, associated adders and critical path) is done in such a manner that it can accommodate both modes of computation and 6-level pipelining provides the necessary timing requirements as in the [12] and [13]. Hence, this proposed architecture is area efficient without compromising the speed of computation.
- Other functional units such as Exponent process (EP), Alignment shift (AS), Sign of result processing, Inverter, Adder, Leading-zero anticipator (LZA), Normalizer and Exponent adjust (EA) are modified to process the proposed data arrangement in both modes of the FMA.

The remaining sections of the paper are organized as follows: Section 2 gives an idea of the working of a conventional FMA. In Section 3, general architecture of the proposed dual mode FMA, design of KO based MM, other arithmetic functional units and rearrangement units are described. The hardware implementation of the proposed FMA is explained, also the results are discussed in detail in Section 4 and the final section forms the conclusion.

2. Conventional FMA

A binary Floating point number is represented as $(-1)^S \times M \times 2^{E+P}$, where S is sign bit; M is mantissa; E is exponent; P is bias for the given precision (*127* for SP and *1023* for DP). IEEE 754 standardises the floating point representations for single and double precision as shown in Fig. 1.

Conventionally the FMA architecture for performing the operation Z = (A × B) ± C has three stages as mentioned in [2,3] and [14]. The operation is as follows:

1. First stage contains mantissa multiplier (MM), exponent processor (EP) and alignment shifter (AS). MM gives product A × B in sum and carry form, the EP finds the number of right shifts the AS has to make to align C. Subsequently the C is inverted for effective subtraction.
2. Second stage contains a compound adder (CA), a sign-computation unit (SCU) and a leading-zero anticipator (LZA). The CA sums up the result of A × B (in sum and carry form) with aligned C. LZA gives the number of shifts for normalization in the next step.
3. The final stage contains normalization and rounding unit (NRU) and exponent adjustment unit (EAU). NRU normalizes the result of SCU in stage 2 by shifting as per the result of LZA and formats the final result to the required precision.

3. Architecture of proposed FMA unit

The proposed FMA architecture can process either one set of double precision data in Single instruction single data (SISD) mode or two sets of single precision data concurrently in single instruction multi data (SIMD) mode as presented in [12] and [13]. Each input (A, B, C) and output (Z) data can be accommodated on a single 64 bit register either as double or single precision forms as illustrated in Fig. 2.

3.1. Rearrangement unit – inputs

The parts of the data are rearranged and hence the processing can be completed in common arithmetic operational units. The details of rearrangement of sign, exponent and mantissa of A and B are presented in Fig. 3. Sign is represented as 2-bits to accommodate two single precision data. Exponent is given as 18-bits for the input operands A, B and C. The exponent processing unit (EPU) adds the exponents of A and B to find A × B, and also determines the number of right shifts required by the mantissa of C which has to be added later. Therefore, this

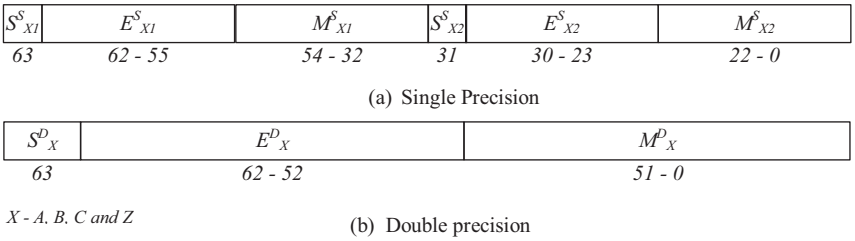

Fig. 2. Arrangements of data in a 64 bit input/output register.

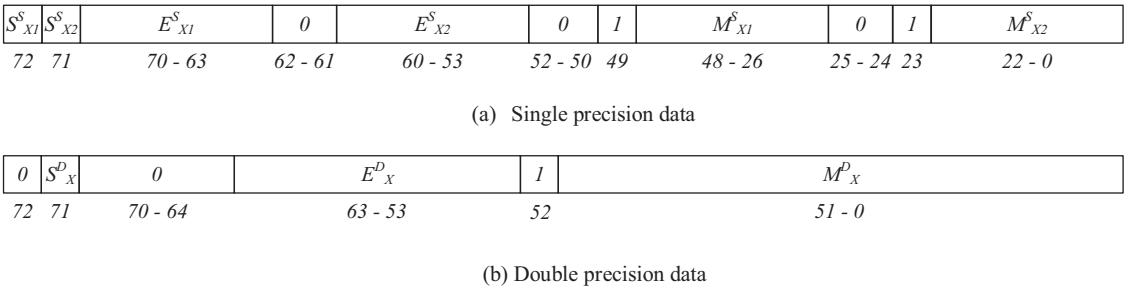

Fig. 3. Rearrangement of sign (1 bit), exponent (18 bit) and mantissa (51 bit) for *A* and *B*.
The inputs A & B are rearranged for feeding in to the later stages of the architecture.