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

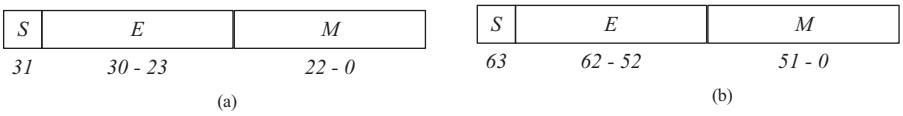

Fig. 1. (a) IEEE-754 Representation - single precision. (b) IEEE-754 Representation – double precision.

- 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

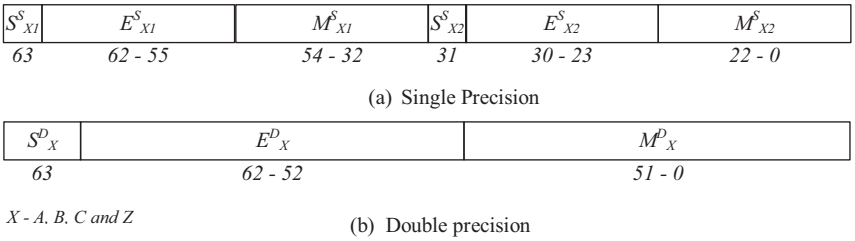

Fig. 2. Arrangements of data in a 64 bit input/output register.

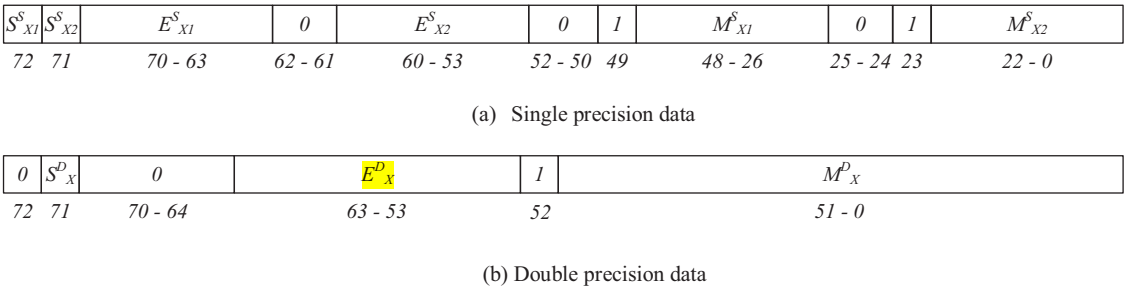

Fig. 3. Rearrangement of sign (1 bit), exponent (18 bit) and mantissa (51 bit) for *A* and *B*.
The inputs A & B are rearranged for feeding in to the later stages of the architecture.

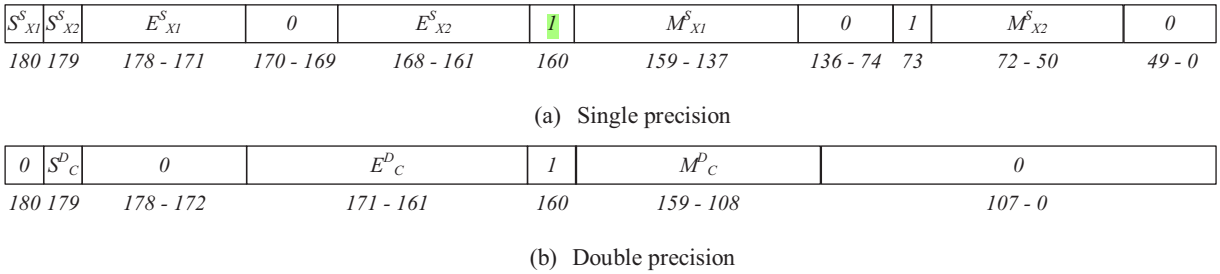

Fig. 4. Rearrangement of sign, exponent and mantissa for *C*.
The input C is rearranged in to 2-bit sign, 18-bit exponent and 161-bit mantissa for feeding in to the later stages of the architecture.

computation involves three operands (exponents of A, B and C). Hence two carry bits are expected. In case of single precision, the 18-bit exponent field is arranged as two exponents of 8-bit and two extra bits for the expected carry inserted between the two exponents as given in the Figs. 3 and 4. This avoids the interference on higher order SP operation by the carry resulting from lower order operation.

Mantissa for A and B are 53-bits for DP as per IEEE-754 standard. Mantissa of C has to be shifted right by 106 bits to align with the result of A × B. Also 2-bits are added in to the right of 53-bit mantissa of C to avoid overflow due to addition of C with A × B. Therefore M'_C has 161 (53 + 2 + 106) bits in total. The input C is rearranged into 2-bit sign, 18-bit exponent and 161-bit mantissa as shown in Fig. 4.

The proposed architecture of 3-stage, 8-level pipelined, dual-precision FMA is illustrated in the Fig. 5. The operational units as well as rearrangement units are designed to operate in both SISD and SIMD modes. Each functional unit is explained in detail in the following subsections.

The rearrangement units use groups of 2:1 multiplexers to rearrange the input data into the required format as per the selection of precision by SP_DPb. These multiplexers contribute to hardware overhead in the proposed architecture. But these make the arithmetic functional units structure more regular, uniform for both modes of computation and hence making hardware sharing possible. In Stage 1 the mantissa multiplier is 6-level pipelined and data synchronization is taken care by inserting dummy pipeline registers wherever necessary. Next two stages have 2 more levels of pipelining.

3.2. Exponent processing (EP) unit

The EP unit has to find the number of right shifts on the mantissa of C based on exponent of the inputs A, B and C. This is computed in two steps using Eqs. (1) and (2).

$$tempE = (\overline{E}A + \overline{E}B - P) \tag{1}$$

$$shiftC = M + tempE - \overline{E}C \tag{2}$$

where, $\overline{E}A$, $\overline{E}B$ and $\overline{E}C$ are rearranged exponents of *A*, *B* and *C* respectively.

For double, P_D = 000000001111111111 and M_D = 0000000000 0111001.

For single, P_S = 011111110001111111 and M_S = 000111000000 011100.

The exponent processing unit works uniformly in both modes, since the inputs and the constants *P* and *M* are rearranged in a proper format. The detailed architecture of the exponent processing unit is presented in the Fig. 6. The *temp_E* is calculated using 2's complement adder and *shift_C* uses 1's complement adder. The inputs to the 1's complement adder *temp_E* and $\overline{E}C$ have been processed with some rearrangements, for single and double precision. This is completed using a 2-bit multiplexer.

3.3. Mantissa multiplier (MM) unit

The MM is the most critical unit in the FMA structure in terms of hardware area and power consumption. Due to multi precision capability the structure becomes more complex. This can be optimized by implementing a single *53* × *53* bit multiplier unit which is common for both the precisions. The Karatsuba–Offman (KO) technique offers a method to implement this multiplier unit as a collection of smaller multipliers [15]. The hardware efficiency of the implementation is improved by using optimum number of smaller multipliers. The following paragraph explains this.

The product is given as in Eq. (3), where *m* is the width of W_0 and X_0. KO algorithm reduces the number of scalar multipliers required in implementing larger multiplication as shown by Karatsuba and Ofman [15]. For the above case, the product is expressed as in Eq. (4). The number of multiplications is reduced from 4 to 3, though weaker adder operations increase.

$$W \times X = W_1X_1 2^{2m} + W_1X_0 2^m + W_0X_1 2^m + W_0X_0 \tag{3}$$

$$W \times X = W_1X_1 2^{2m} + [(W_1 + W_0)(X_1 + X_0) - W_1X_1 - W_0X_0]2^m + W_0X_0 \tag{4}$$

A similar strategy can be extended to multiple segments which can handle either a double or two single precision mantissa multiplications in parallel.

With higher levels of segmentation, the area and critical path (smaller than 53 bit word length) get reduced as the sufficient number of pipeline stages were used. But when the number of segments increases, the number of pipeline registers and the wiring complexity also increases as stated by Mathew et al. [16]. Table 1, analyses the various segmentation possibilities and their corresponding word lengths in each segment based on whether dual mode is possible, the number of multipliers actually used in each mode, delay and the throughput. Considering the size of the mantissas, odd number of segments were not analysed here.

To achieve an acceptable trade-off between area and throughput, the {14, 13, 13, 13} bit segmentation strategy was considered and it is as shown by the Eqs. (5)–(8).

The mantissas can be segmented as:

$$M_A = M_{A3}2^{(3\times13)} + M_{A2}2^{(2\times13)} + M_{A1}2^{(1\times13)} + M_{A0} \tag{5}$$

$$M_B = M_{B3}2^{(3\times13)} + M_{B2}2^{(2\times13)} + M_{B1}2^{(1\times13)} + M_{B0} \tag{6}$$

For double precision the product by MM is,

$$\begin{aligned} M_A \times M_B = {} & \{M_{A3}M_{B3}2^{(2\times13)} + [(M_{A3} + M_{A2})(M_{B3} + M_{B2}) - (M_{A3}M_{B3} + M_{A2}M_{B2})]2^{13} \\ & + M_{A2}M_{B2}\}2^{4\times13} \\ & + [(M_{A3} + M_{A1})(M_{B3} + M_{B1}) - (M_{A3}M_{B3} + M_{A1}M_{B1})]2^{4\times13} \\ & + [(M_{A3} + M_{A0})(M_{B3} + M_{B0}) - (M_{A3}M_{B3} + M_{A0}M_{B0})]2^{3\times13} \\ & + [(M_{A2} + M_{A1})(M_{B2} + M_{B1}) - (M_{A2}M_{B2} + M_{A1}M_{B1})]2^{3\times13} \\ & + [(M_{A2} + M_{A0})(M_{B2} + M_{B0}) - (M_{A2}M_{B2} + M_{A0}M_{B0})]2^{2\times13} \\ & + M_{A1}M_{B1}2^{(2\times13)} + [(M_{A1} + M_{A0})(M_{B1} + M_{B0}) - (M_{A1}M_{B1} + M_{A0}M_{B0})]2^{13} \\ & + M_{A0}M_{B0} \end{aligned} \tag{7}$$

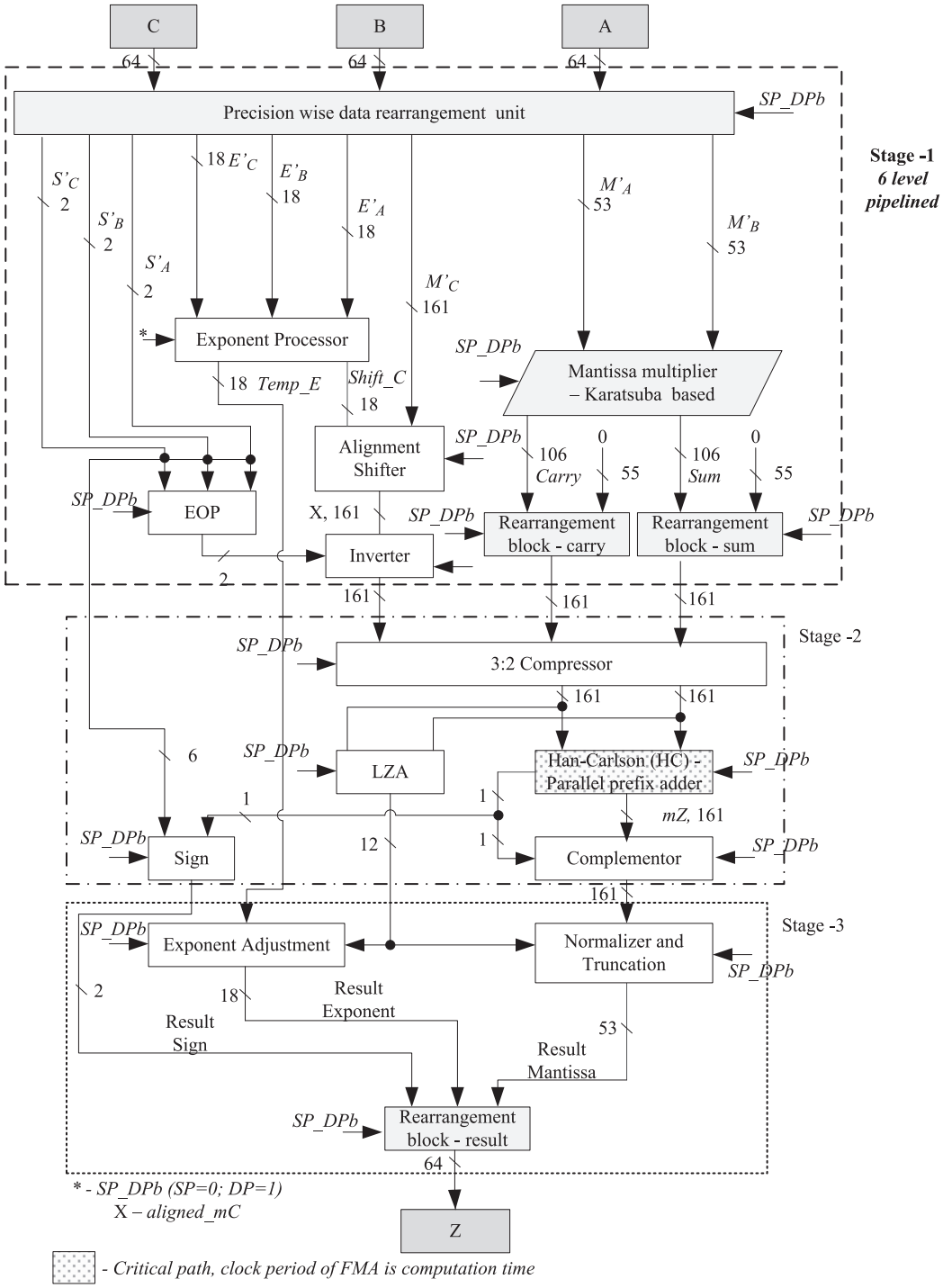

Fig. 5. Architecture of the proposed 3 – stage, 8 - level pipelined dual-precision floating point FMA.

For single precision the product by MM is,

$$M_A \times M_B = \{M_{A3}M_{B3}2^{(2\times13)} + [(M_{A3} + M_{A2})(M_{B3} + M_{B2}) - (M_{A3}M_{B3} + M_{A2}M_{B2})]2^{13} + M_{A2}M_{B2}\}2^{4\times13} + M_{A1}M_{B1}2^{(2\times13)} + [(M_{A1} + M_{A0})(M_{B1} + M_{B0}) - (M_{A1}M_{B1} + M_{A0}M_{B0})]2^{13} + M_{A0}M_{B0} \quad (8)$$

From the Eqs. (7) and (8), it is clear that the proposed partitioning scheme reuses 6 out of the 10 smaller multipliers and 13 out of 33 adder/subtractors to implement two single precision multiplications. Booth encoded multipliers could be used to realize faster individual multipliers in the above implementation which makes the structure highly irregular [12].

Hence, array multipliers have been preferred in this reconfigurable system. In such multipliers, carry save adder (CSA) trees are used to sum up the partial products [17]. Though CSA trees can have different sized compressors, in this implementation 3:2 (full adder) and 2:2 (half adder) compressors are used for simplicity. In this schema of $n \times n$-bit multiplier, the first stage has $n - 1$ half adders. Each one of the following $n - 2$ stages has $n - 1$ full adders. At the end of $n - 1$ stages, two $2n - 1$ bit summands remain. A fast (ripple carry or carry look ahead) adder is required to give the final product. However, in an FMA, this final adder can be ignored and it is adequate to represent the product in carry-sum format (i.e. two final summands) because these summands can directly be added with the third operand (C) in the following adder.

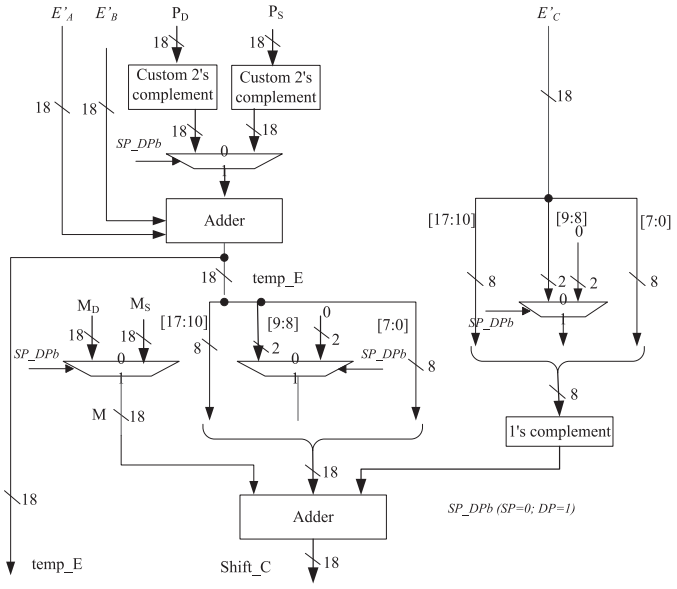

Fig. 6. Exponent processing unit.
Computes temp_E and shift_C in both the modes using mantissas of A, B and C. Multiplexers are selecting inputs according to the selection SP_DPb (0/1).

Table 1
Analysis of the factors – to determine the width and number of segmentation for KO implementation – 53-bit mantissa multiplier.

Number of segments (n)	2	4	6
Word length of segments (bits)	26, 27	14, 13, 13, 13	9, 9, 9, 8, 9, 9
Maximum number of multipliers (n^2)	4	16	36
Number of multipliers ($n + {}^nC_2$)	3	10	21
Single precision configuration	26, 27	(14,13) & (13,13)	(8,9,9) & (9,9,9)
Throughput (samples/s)	568.2M	926M	952.4M
Pipeline (levels) / delay (cycles)	6	6	8
Area (μm^2) × 10^5	1.22	0.937	1.46

To accommodate the two single precision MMs in parallel on a 53 bit MM, the single mode MM to be used are 27 and 26 bits (reuse of the hardware). Best segmentation policy is selected based on hardware area and delay for the computation. The overall area and delay are computed using TSMC 130 nm 1P6M technology libraries.

Hence, the product of each individual smaller multiplier is maintained in carry-sum format. This also enhances the hardware efficiency of the multiplier. The architecture of the dual precision, six-level pipelined mantissa multiplier for the proposed FMA is shown in the Fig. 7.

To have meaningful comparisons, the proposed architecture is implemented using TSMC 130 nm technology libraries. Later, it is compared with the design presented in [18]. Table 2 provides the comparisons and it clearly indicates that the proposed design is efficient due to efficient hardware sharing among the modes and retaining the product in carry-sum format.

3.4. Alignment shifter

Two floating point numbers' mantissas can be added when their exponents are equal; if they are not equal then one of the number's mantissa should be aligned with the other. Therefore the operand C has to be aligned with the result of A × B and then added together. For double precision, a 161-bit barrel shifter is required, which comprises 8-stages of 161 2:1 multiplexers. Each stage shifts the input by 2^i where, i is the stage index (0 to 7). For single precision, two 74-bit alignment shifters are required. The 8-stage, 161-bit barrel shifter, can be used for both single and double precision computations. To achieve this, the 8-stage, 161-bit barrel shifter is segmented bitwise into three segments, each having 8-stages and individual select lines. The segments are as follows: segment – 2 [160 to 87], segment – 1 [86 to 74] and segment – 0 [73 to 0]. The number of shifts on the mantissa of C, Shift_C is computed using EP unit. Single precision computation requires three 8-bit select lines to control each segment. The arrangement of these select lines for both the precision is illustrated in Fig. 8(a). In order to realize another 74-bit shifter from the third segment, inputs to ith stage of multiplexers from $(i - 1)$th stage outputs are modified at lines from 74 to 74 $\pm$ 2^{i-1} as in Fig. 8(b).

3.5. Sign of result and effective operation

The sign of the result depends on the sign of inputs {S'$_A$, S'$_B$, S'$_C$} and the result of the mantissa adder, mZ as shown in Fig. 9. The sign of A and B are either {sAB1, sAB2} for single precision or sAB1 for double precision. The final computations using these signals are given in Tables 3 and 4. The sign of the final result depends on the sign of the addend and the augend. In the case of double precision, the 162nd bit of the mantissa sum is used in the determination of the sign. Whereas in single precision mode, the 75th bit is used for the first set of data and 162nd bit for the second set.

3.6. Inverter

The inverter block passes either the X or complemented X (~X) with respect to output of EOP unit. The multiplexers are used for satisfying the above mentioned operation on both modes effectively. In case of double precision, all 161-bits are considered for conversion whereas in single precision, only the rightmost and leftmost 74-bits (160-87 and 73-0) are considered for inversion, leaving 13-bits in the middle unaltered.

3.7. Adder

The adder has to process the results of KO based segmented data at dual precisions (single/double). Therefore the reusability of adders is necessary and it has to be in uniform structures. Use of Booth multiplier in a KO based implementation results in highly irregular structures [12]. Thus the three 161-bit inputs, mantissa(s) of C and the outputs of MM (sum and carry) are fed to 3:2 compressor modules (the three summands are now reduced to two) and these are processed by the Han-Carlson (HC) parallel prefix adder, which computes faster and consumes relatively lesser area than carry-save adder [19]. The resulting output is 161-bit mZ. The complementor will be active only when subtraction is the effective operation, i.e. output of EOP ≠ 0. The complementor either complements the mZ when the carry is logic 0 or increments by 1 otherwise and the same is as shown in Tables 3 and 4.

3.8. Leading zero anticipator (LZA)

To speed up the normalization operation a LZA unit works in parallel with the adder. The LZA uses the same summands and computes the number of zeros before the most significant digit in the summation result. Here, LZA is implemented by modifying the algorithm proposed in Javier et al. [20]. The modification is to consider only the cases of (i) unsigned addition and (ii) unsigned subtraction with both positive and negative result. This supports one 161-bit prediction for double precision computations whereas, for single precision computations, two 74-bit predictions have been considered. The inputs to LZA module are 161-bit sum and carry outputs of 3:2 compressor. There could be an error of 1-bit, due to in-exact method of predicting leading number of zeros (addition) or ones (subtraction) [21]. Here, to keep the circuit simple, LZA does not include error detection and correction mechanism. Rather it will be taken care after the normalization shift. The pre-encoding process produces the string, E (E0…. E160), using the expressions (9) and (10). This gives the position of leading one with one bit

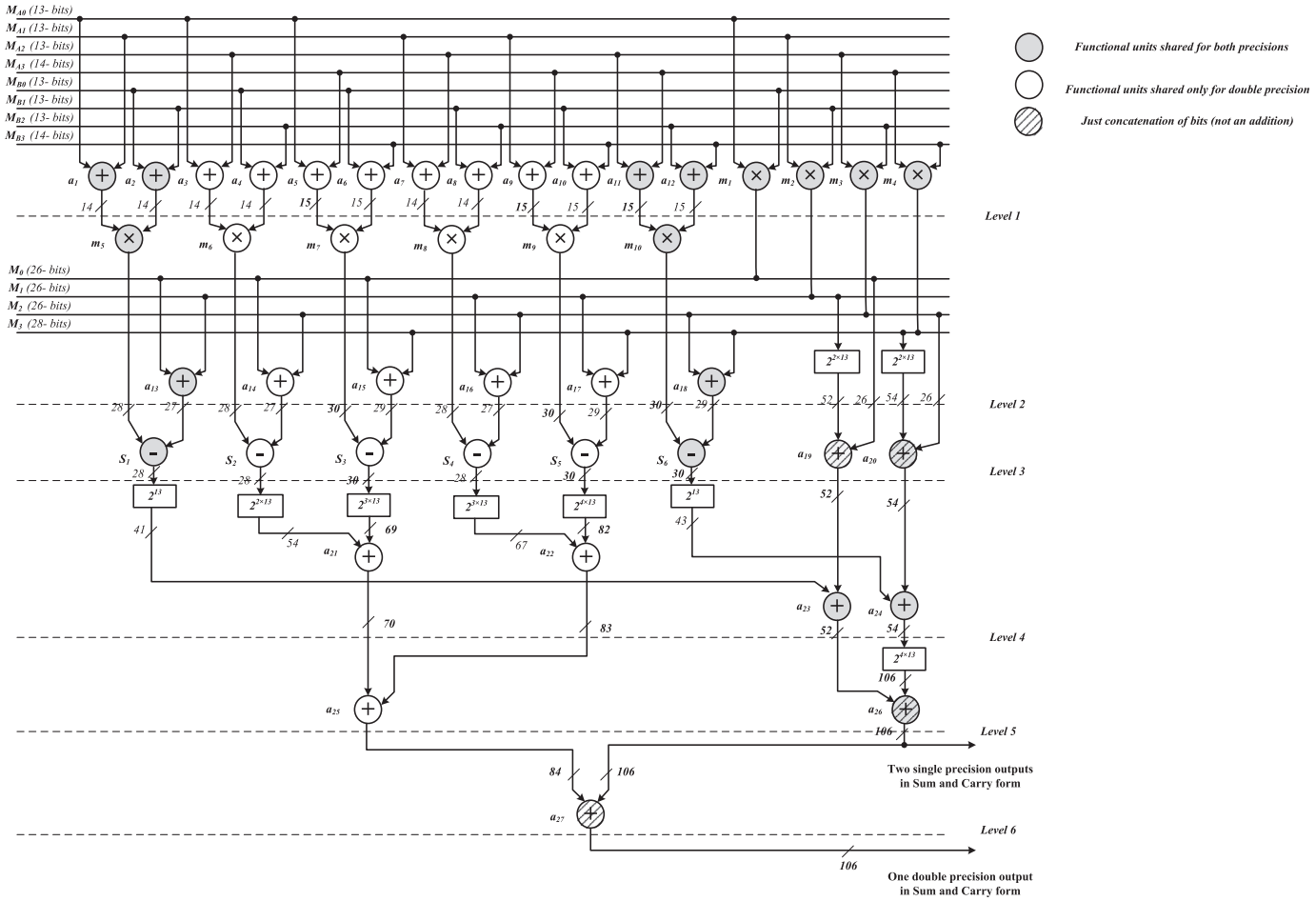

Fig. 7. Architecture of dual precision mantissa multiplier (MM) – segmented using Karasuba Offman technique and six level pipelined.

Table 2
Comparison of hardware area and delay of proposed mantissa multiplier with the [18].

Number of multipliers	[18] One 53 and two 34 bit	Proposed 10 multipliers: (Three15-bit, Four 14-bit, Three 13-bit)
Hardware sharing in double precision	One 53-bit multiplier	All 10 multipliers
Hardware sharing in single precision	Two 34-bit multipliers	6 of 10 multipliers: (One 15-bit, Two 14-bit, Three 13-bit)
Hardware area of mantissa multiplier (μm^2)	1.23×10^5	0.937×10^5
Throughput (samples /s)	546.45M	546.45M
Pipelining levels	1	6

The mantissa multiplier in FMA contains smaller multipliers and necessary adders.

error that has to be corrected by one more left shift.

$$P_i = A_i \oplus B_i, \quad 3G_i = A_i B_i, \quad Z_i = \overline{A_i B_i} \tag{9}$$

$$E_i = \overline{P}_{i-2} Z_i \overline{Z}_{i+1} + P_{i-2}(P_{i-1} G_i + P_{i-1} Z_i) \tag{10}$$

With the string E, the position of the leading one can be obtained using leading-one-detector (LOD) tree which is a priority encoder. Two 74-bit encoders are placed in parallel with a distance of 13 bits between them in the tree to realize 161-bit encoder. To encode leading-one-predictor (LOP) in 161-bit predictor, 8-bits are needed. Similarly each of the 74-bit predictors needs 7-bits, thus a total of 14-bits is used as the LZA output.

3.9. Normalization shifter and truncation

The result from the complementer is given to the normalization shifter along with the output of the LZA. For double precision, the input from complementer can be treated as one 161-bit data. A barrel shifter shifts these 161-bits by an amount LOP, as given by LZA. However, this normalization shift may have one bit error as mentioned in the previous section. The error is found if MSB is not set. The correction is one more left shift to the normalized result. The correction in normalization and truncation process is illustrated in the Fig. 10.

3.10. Exponent adjustment

To get the final exponent (eZ) of the result, the temporary exponent, temp_E is adjusted with respect to LOP computed from LZA.

$$eZ = tempE - LOP + 56$$
$$\text{eZ=eZ} - 1, \quad \text{if } normalization\ error \tag{11}$$

$$eZ = tempE - LOP + 27, \quad for\ tempE - E'_C < 0$$
$$\text{eZ=eZ} - 1, \quad \text{if } normalization\ error \tag{12}$$

The calculations are as per (11) for double precision and (12) for single precision. Precision wise allotment of bits for eZ is the same as noted for exponents in Table 2.

4. Synthesis, placement & routing and results

The proposed 3-stage, 8-level pipelined dual-precisions FMA architecture was described in Verilog HDL. The functional correctness of the

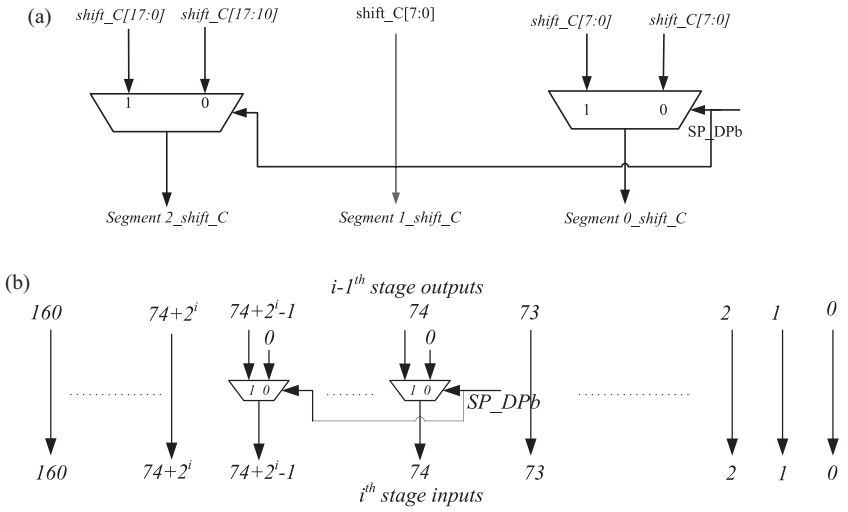

Fig. 8. (a) Alignment shifter's select lines at the three 8-bit segments of shift_C. (b) Alignment shifter's input lines at *i*th stage. Based on the SP_DPb bit, Shift_C and the inputs to the *i*th stage are set, thus the same alignment shifter used in both modes.

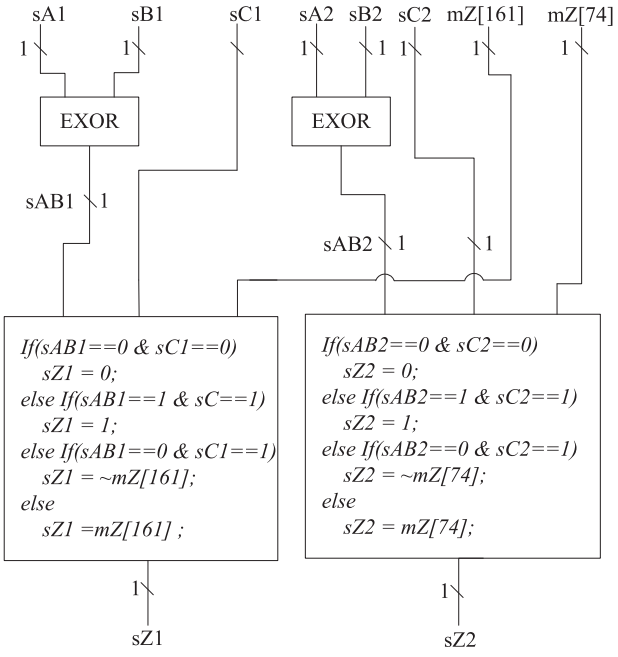

Fig. 9. Sign Processing Unit.

Table 3
Sign of the result – double precision.

sAB1	sC1	sZ1
0	0	0
1	1	1
0	1	~mZ[161]
1	0	mZ [161]

sZ1 is the sign of the result, sAB1 is the sign of product of A × B and sC1 is the sign of C. ~mZ – complement of mZ.

design was verified using ModelSim simulations. The test bench uses: (1) trivial inputs and outputs as in the Table 5 [13] and (2) a set of 100 random inputs and their outputs in the binary IEEE 734 double precision and single precision format.

The functionally correct code was synthesized using Cadence RTL compiler. The RTL synthesis was processed with typical synthesis effort and using TSMC 1P6M 130 nm CMOS standard cell library. The intended functionalities of the resulting net-list from the RTL synthesis were verified in Cadence NCsim. The Cadence Encounter was used for

Table 4
Sign of the result - single precision.

sAB1/ sAB2	sC1/ sC2	sZ1/ sZ2
0	0	0
1	1	1
0	1	~ mZ [161]/~ mZ [74]
1	0	mZ [161]/mZ [74]

sZ1is the output sign of the 1st set with inputs sAB1 and sC1.
sZ2is the output sign of the 2nd set with inputs sAB2 and sC2.

backend design steps such as placement and routing, adding clock and power network. Then power consumption of the implemented core is estimated using power analyzer. The power analyzer uses the value change dump (VCD) file and the test-bench with switch pattern coverage of 90% (assumed to be the worst case scenario). After this the overall hardware area was estimated from the resulting core of proposed FMA. The results are illustrated in Table 6 and compared with the results presented in [13].

Stage-2 contains the Han-Carlson fast adder, which has the worst propagation delay, and thus largest clock period is required. The same being used for the other two stages as well. The application of KO algorithm for the design of dual mode MM and placing four DRUs at the crucial instances of the architecture, make the arithmetic functional units to be shared among the modes. Therefore the overall hardware area is reduced in the proposed design. It is clearly shown from Table 6, that the proposed implementation reduces the hardware area by 48% in 130 nm. This design can work at a little higher frequency than the previous designs, hence the area delay product (ADP) is analyzed and that reveals that there is a 51.22% reduction in 130 nm.

5. Conclusions

Here, an area efficient 3-stage, 8-level pipelined dual precision FMA is designed and implemented without any increase in execution time. The area of the stage-1 of the proposed FMA is reduced by 69% due to the Karatsuba–Offman (KO) based MM design. The application of KO algorithm for the design of dual mode MM can be extended to triple mode i.e. quad, double and single precision computations. The 112-bit MM should be segmented in such a manner that the computations can be at the precision of either one quad or two double in parallel or four single in parallel. The rearrangement units were used at the optimal locations in the architecture for sharing the similar arithmetic functional units in SISD and SIMD modes. Thus the overall hardware area is reduced approximately by 50% irrespective of the process technology library used.

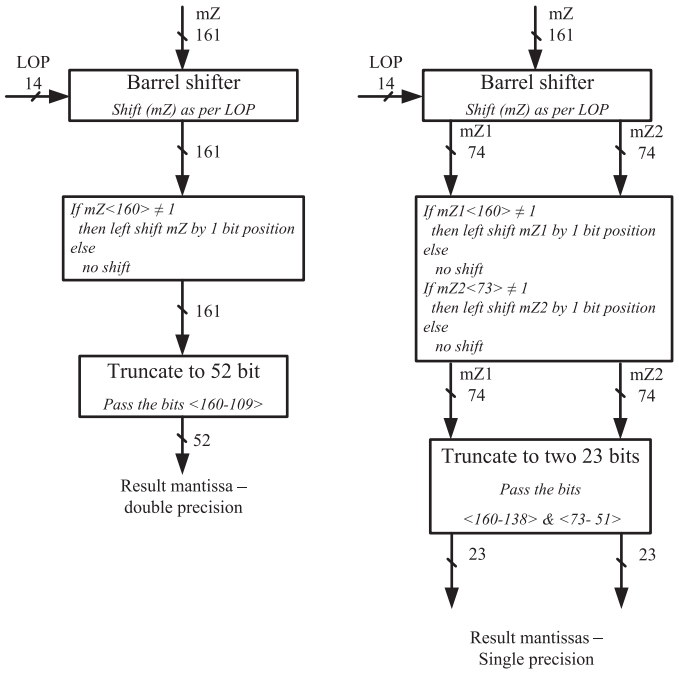

Fig. 10. Normalization shifter and truncation.

Table 5
Trivial inputs & outputs.

Inputs			Output
A	B	C	Z = (A × B) + C
A	1	0	A
1	B	0	B
1	1	C	C + 1
0	B	C	C
A	0	C	C

Table 6
Comparison of 3-stage, 8-level, dual precision FMA.

	[13]	Proposed
Technology (μm)	0.13	0.13
Latency (cycles)	3	8
Area (μm^2) × 10^5	2.86	1.49
Frequency (MHz)	291	308.7
Power (mW)	35.22	17.82
Area Delay product (ADP) × 10^{-15}	0.984	0.48

Efficient reusability of multipliers and other arithmetic units brings the improvement in the area efficiency.

Acknowledgments

The authors wish to thank Dr. V. Ramesh, Professor, School of Electrical Engineering, VIT University, Vellore for his valuable suggestions in the presentation of ideas and patient proof reading.

References

[1] R.K. Montoye, E. Hokenek, S.L. Runyon, Design of the IBM RISC System/6000 floating-point execution unit, IBM J. Res. Dev. 34 (1) (1990) 59–70.
[2] E. Hokenek, R.K. Montoye, P.W. Cook, Second-generation RISC floating point with multiply-add fused, IEEE J. Solid-State Circuits 25 (5) (1990) 1207–1213.
[3] S.M. Mueller, C. Jacobi, O.h. Hwa-Joon, et al., The vector floating-point unit in a synergistic processor element of a cell processor, Proc. 17th IEEE Symp. Computer Arithmetic (ARITH-17), Cape Cod, Massachusetts, USA, 2005, pp. 59–67.
[4] T. Chen, R. Raghavan, J.N. Dale, et al., Cell broadband engine architecture and its first implementation – a performance view, IBM J. Res. Dev. 51 (5) (2007) 559–572.
[5] R.A. Haring, M. Ohmacht, T.W. Fox, et al., The IBM Blue Gene/Q Compute hip, IEEE Micro 32 (2) (2012) 48–60.
[6] C.D. Wait, IBM PowerPC 440 FPU with complex-arithmetic extensions, IBM J. Res. Dev. 49 (2) (2005) 249–254.
[7] G. Shainer, P. Lui, M. Hilgeman, et al., Maximizing application performance in a multi-core, NUMA-aware compute cluster by multi-level tuning, Proc. 28th International Supercomputing Conference Leipzig, Germany, 2013, pp. 226–238.
[8] N. Kurd, 5.9 Haswell: a family of IA 22 nm processors, Proc. IEEE International Solid-State Circuits Conference Digest of Technical Papers, San Francisco, CA, USA, Feb, 2014, pp. 112–113.
[9] E. Quinnell, E.E. Swartzlander, C. Lemonds, Floating-point fused multiply-add architectures', Proc. 41st Asilomar Conference on Signals, Systems and Computers Pacific Grove, California, USA, 2007, pp. 331–337.
[10] Z. Qi, Q. Guo, G. Zhang, Design of low-cost high-performance floating-point fused multiply-add with reduced power, 23rd International Conference on VLSI Design, Bangalore, India, 2010, pp. 206–211.
[11] N. Brunie, F.d. Dinechin, B.d. Dinechin, A mixed-precision fused multiply and add, Proc. 45th Asilomar Conference on Signals, Systems and Computers, Pacific Grove, CA, USA, 2011, pp. 165–169.
[12] L. Huang, Changsha, L. Shen, et al., A new architecture for multiple-precision floating-point multiply-add fused unit design, Proc. 18th IEEE Symposium on Computer Arithmetic, Montepellier, France, 2007, pp. 69–76.
[13] K. Manolopoulos, D. Reisis, V.A. Chouliaras, An efficient dual-mode floating-point multiply-add fused unit, Proc. 17th IEEE International Conference on Electronics, Circuits, and Systems, Athens, Greece, 2010, pp. 5–8.
[14] F.P. O'Connell, S.W. White, POWER3: the next generation of PowerPC processors, IBM J. Res. Dev. 44 (6) (2000) 873–884.
[15] A. Karatsuba, Y. Ofman, Multiplication of multidigit numbers on automata, In Soviet Physics Doklady, 1963, 7, pp. 595–596.
[16] S. Mathew, M. Kounavis, F. Sheikh, et al., 3 GHz, 74 mW 2-level Karatsuba 64b Galois field multiplier for public-key encryption acceleration in 45 nm CMOS, Proc. 36th European Solid-State Circuits Conference, Sevilla, Spain, 2010, pp. 198–201.
[17] A. Habibi, P.A. Wintz, Fast multipliers, IEEE Trans. Comput. 19 (2) (1970) 153–157.
[18] M.K. Jaiswal, R.C.C. Cheung, VLSI implementation of double-precision floating-point multiplier using Karatsuba technique, Circuits Syst. Sig. Process. 32 (1) (2013) 15–27.
[19] S.M. Sudhakar, K.P. Chidambaram, E.E. Swartzlander, Hybrid Han-Carlson adder, Proc. IEEE 55th International Midwest Symposium on Circuits and Systems, Boise, ID, 2012, pp. 818–821.
[20] J.D. Bruguera, L. Tomás, Leading-one prediction with concurrent position correction, IEEE Trans. Comput. 48 (10) (1999) 1083–1097.
[21] Schmookler, S. Martin, K.J. Nowka, Leading zero anticipation and detection-a comparison of methods, Proc. 15th IEEE Symposium on Computer Arithmetic Vail, Colorado, 2001, pp. 7–12.

V. Arunachalam received the B.E. degree in Electrical and Electronics Engineering from University of Madras, India, in 1997 and M.E. degree in Power Electronics and Drives from Anna University, Chennai, India, in 2002. Since 2004, he has been a member of faculty in the Department of Micro and Nano Electronics, School of Electronics Engineering, Vellore Institute of Technology University, Vellore, India, where he is currently an Associate professor. His research interests are FPGA based system design, HW/SW partitioning, VLSI DSP and reconfigurable architecture. He is serving as IEEE student branch counselor, VIT University, Vellore.

Alex Noel Joseph Raj received the B.E. degree in Electrical Engineering from Madras University, India, in 2001, the M.E. degree in Applied Electronics from Anna University in 2005, and the Ph.D. degree in Engineering from the University of Warwick in 2009. From October 2009 to September 2011, he was with Valeport LTD Totnes, UK as Design Engineer. From March 2013 to March 2017 he was with the Department of Embedded technology, School of Electronics Engineering, VIT University, Vellore, India as a professor. Since March 2017, he is with Department of Electronic Engineering, College of Engineering, Shantou University, China. His research interests include signal, image and sonar processing, and FPGA implementations.

Naveen Hampannavar received the B.E degree in Electronic and Communications from Visheswarayya Technological University, India in 2008 and M.Tech degree in VLSI Design from VIT University, Vellore, India in 2014. He is currently with Aceic Design Technologies, Bengaluru, India as Senior Verification Engineer.

C.B. Bidhul received the B.Tech. degree in Electronics and Communication Engineering from College of Engineering, Munnar, India, in 2010. He worked with Travancore Analytics Pvt. Ltd, Trivandrum, India from 2010 to 2012 and received M.Tech. degree in VLSI Design from VIT University, Vellore, India, in 2014. Since 2014, he has been working as a Senior engineer at TATA Elxsi Ltd., Trivandrum, India. His research interests are software development for engineering applications, Image processing and VLSI for image processing.