# Area-Efficient Architecture for Dual-Mode Double Precision Floating Point Division

Manish Kumar Jaiswal and Hayden K.-H. So

***Abstract*—Floating point division is a core arithmetic widely used in scientific and engineering applications. This paper proposed an architecture for double precision floating point division. This architecture is designed for dual-mode functionality, which can either compute on a pair of double precision operands or on two pairs of single precision operands in parallel. The architecture is based on the series expansion multiplicative methodology of mantissa computation. For this, a novel dual-mode Radix-4 Modified Booth multiplier is designed, which is used iteratively in the architecture of dual-mode mantissa computation. Other key components of floating point division flow (such as leading-one-detection, left/right dynamic shifters, rounding, etc) are also re-designed for the dual-mode operation. The proposed dual-mode architecture is synthesized using UMC $90nm$ technology ASIC implementation. Two versions of proposed architecture are presented, one with single stage multiplier and another with two stage multiplier. Compared to a standalone double precision division architecture, the proposed dual-mode architecture requires $17\%$ to $19\%$ extra hardware resources, with $3\%$ to $5\%$ period overhead. In comparison to prior art on this, the proposed architecture out-performs them in terms of required area, time-period and throughput.**

***Index Terms*—Arithmetic, Floating Point Division, Dual-Mode Division, ASIC, Multi-precision Arithmetic, Configurable Architecture.**

## I. Introduction

Floating point arithmetic (FPA) architectures underwent significant advancement by scientific research in the past several decades. FPA is a basic ingredient of a large set of scientific and engineering domain applications. To boost the application performances, the FPA architectures developed from scalar to vector architectures in various processing platforms. Arrays of single precision and double precision computing units are being used for floating point vector processing. The current research work is aimed towards the idea of unified vector-processing units. That is, instead of having separate vector arrays of single precision and double precision, it can have an array of configurable floating point arithmetic blocks. Where each of these configurable blocks can process either a double precision or two parallel single precision computations. This configurable block array arrangement can lead to significant area improvement, while providing the required performance.

Our research work is focused on the architecture design of configurable floating point arithmetic blocks. This paper is focused on the design of configurable dual-mode double precision division arithmetic unit. Floating point (FP) division is a core computation required in a multitude of applications. FP division is a complex arithmetic operation which requires larger area with poor performance than the basic arithmetic operations (adder, subtractor and multiplier). In view of large area requirement of division arithmetic per unit of computation, this work is aimed for a multi-precision dual-mode architecture for this computation. The proposed architecture can be configured either for a double precision or two parallel (dual) single precision division computations, and thus named as DPdSP division architecture.

The proposed architecture is based on the series expansion methodology of division algorithm [1], [2], [3], [4], [5]. Series expansion method is a multiplicative division method, like Newton-Raphson (NR) and Goldschmidt (GS) methods [2], [6], which are faster than the digit-recurrence methods like SRT. A detailed discussion on various methodology for FP division can be sought from [7], [2]. The mantissa division in multiplicative method is based on the usage of integer multipliers, which adds to their performance compared to traditional digit recurrence method (like SRT method). Further, series expansion method provides a hardware efficient architecture for a given precision requirement [5]. The series expansion method helps in reducing the memory requirement, and other related hardware resources compared to NR and GS methods. Based on the currently proposed dual-mode division architecture using series expansion methodology, other multiplicative division methods (NR and GS methods) can also be designed for dual-mode architecture, however, they need to be investigated for their feasibility in configurable dual-mode architecture, which is a part of our future study.

A novel dual-mode Radix-4 Modified Booth multiplier is designed for the purpose of mantissa division, which has minimal area and performance overhead over single-mode multiplier. It is based on the Radix-4 Modified Booth Algorithm [8] which is redesigned here, to accommodate dual-mode processing. Also, since the underlying integer multiplier in mantissa division unit has the major cost in terms of required area, an iterative architecture is proposed using a single 1-stage integer multiplier, to achieve area efficiency. Furthermore, to improve on the performance, an architecture with 2-stage integer multiplier is also proposed, which further demonstrates the use of multi-stage multiplier in the proposed dual-mode architecture.

The proposed DPdSP division architectures are designed for normal as well as sub-normal computational support, which also include the exceptional case handling and processing.

This work is partly supported by the "The University of Hong Kong" grant (Project Code. 201409176200), the "Research Grants Council" of Hong Kong (Project ECS 720012E), and the "Croucher Innovation Award" 2013.

Manish Kumar Jaiswal is with Department of EEE, The University of Hong Kong, Hong Kong. e-mail: manishkj@eee.hku.hk

Hayden K.-H. So is with Department of EEE, The University of Hong Kong, Hong Kong. e-mail: hso@eee.hku.hk

It is able to produce faithful rounded results with round-to-nearest rounding, both in double and single precision. Faithful rounding is suitable for a large set of application, however, the correct rounding can be included using remainder method for multiplicative division methodology [2], [9], which requires processing of one more multiplication. All the major building blocks (like mantissa division, leading-one-detection, dynamic right/left shifting, rounding) are designed and optimized for the efficient dual-mode processing. A single mode double precision division architecture, based on similar computational flow, is also designed for comparison purpose, to demonstrate the relative benefits of dual-mode division architecture.

Several papers have proposed FP architectures on the idea of configurable multi-precision floating point arithmetic processing. The majority of prior works are focused towards the adder architectures [10], [11], [12], [13], [14] and multiplier architectures [15], [16], [17]. Isseven *et al.* [18] is the only available work on dual-mode division arithmetic which presented an iterative dual-mode architecture for division, based on the Radix-4 SRT (digit recurrence) division method, and is aimed only for normal format of computation. The algorithmic idea of current work is presented by Jaiswal *et al.* in [19], with a single cycle fully unrolled design for the illustration purpose, which requires large area with poor performance. The present work is built upon the [19], with interesting and practical approach for DPdSP division architecture, included with some novel architectural improvements.

The main contributions of this work can be summarized as follows:

- Proposed dual-mode DPdSP division architectures with normal and sub-normal computational support, along with all the exceptional case handling. These architectures can be dynamically configured either for a double precision division or two parallel single precision divisions.
- A novel dual-mode Radix-4 Modified Booth multiplier architecture is proposed, which becomes the base of the proposed dual-mode mantissa division architecture.
- All the key components of the FP division flow is designed for efficient dual-mode functionality with minimal overhead.
- Proposed architectures are fully pipelined, and designed in an iterative fashion for area-efficiency.

This manuscript is organized as follows. Section II briefly discusses the algorithmic flow of the FP division arithmetic and the underlying mantissa division methodology used in this manuscript. Section III and Section IV describe the proposed dual-mode DPdSP division architectures with different pipelined stage multipliers. Section V discusses the single-mode DP division design, which is designed for the comparison purpose. The detail implementation results and related comparisons with previous literature work are presented in the Section VI. Finally, the manuscript is concluded in section VII.

## II. Background

The underlying computational flow for FP division arithmetic is presented in Algorithm 1. This algorithm is suitable for both normal and sub-normal processing. It also includes the exceptional case handling and processing. FPA implementation involves computing separately the sign, exponent and mantissa part of the operands, and later combining them after rounding and normalization [20].

**Algorithm 1** F.P. Division Computational Flow [20]

1: ($IN1$ ($Dividend$), $IN2$ ($Divisor$)) Input Operands;
2: **Data Extraction & Exceptional Check-up:**
  {S1(Sign1), E1(Exponent1), M1(Mantissa1)} $\leftarrow IN1$
  {S2, E2, M2} $\leftarrow IN2$
  Check for Infinity, Sub-Normal, Zero, Divide-By-Zero
3: **Process both Mantissa for Sub-Normal:**
  Leading One Detection of both Mantissa ($\rightarrow L\_Shift1, L\_Shift2$)
  Dynamic Left Shifting of both Mantissa
4: **Sign, Exponent & Right-Shift-Amount Computation:**
  $S \leftarrow S1 \oplus S2$
  $E \leftarrow (E1 - L\_Shift1) - (E2 - L\_Shift2) + BIAS$
  $R\_Shift\_Amount \leftarrow (E2 - L\_Shift2) - (E1 - L\_Shift1) - BIAS$
5: **Mantissa Computation:** $M \leftarrow M1/M2$
6: **Dynamic Right Shifting of Quotient Mantissa**
7: **Normalization & Rounding:**
  Determine Correct Rounding Position
  Compute ULP using Guard, Round & Sticky Bit
  Compute $M \leftarrow M + ULP$
  1-bit Right Shift Mantissa in Case of Mantissa Overflow
  Update Exponent
8: **Finalizing Output:**
  Determine STATUS signal & Resolve Exceptional Cases
  Determine Final Output

In the present work, all stages of the above computational flow are designed to support dual-mode operations.

### A. Underlying Mantissa Division Method

The mantissa division is the most complex part of the FP division arithmetic implementation. The algorithmic methodology for this computation is discussed here. It is based on the series expansion method of division, as follows.

Let $m_1$ be the normalized dividend mantissa and $m_2$ be the normalized divisor mantissa, then $q$, the mantissa quotient, can be computed as:

$$q = \frac{m_1}{m_2} = \frac{m_1}{a_1 + a_2} = m_1 \times (a_1 + a_2)^{-1} \tag{1}$$

Here, the divisor mantissa $m_2$ is partitioned into two parts as $a_1$ (with $W+1$-bit), and $a_2$ (all remaining bits) as below.

$$m_2 \rightarrow \overbrace{1.\underbrace{xxxxxxxx}_{W}}^{a_1}\overbrace{xx.....xxxxxxx}^{a_2}$$

By using Taylor Series expansion,

$$(a_1 + a_2)^{-1} = a_1^{-1} - a_1^{-2}a_2 + a_1^{-3}a_2^2 - a_1^{-4}a_2^3 + \cdots \tag{2}$$

The above equation can be evaluated by using only multipliers, adders and subtractors, provided that the value of $a_1^{-1}$ is available. The pre-computed value of $a_1^{-1}$ can be accessed from a pre-stored look-up table to perform the entire computation; which is easily realizable in hardware implementation. The pre-computed value of $a_1^{-1}$ acts as an initial approximation for $m_2^{-1}$, which further improved with remaining computation in (2). Here, the size $W$ (bit width) of $a_1$ (here, the hidden '1' bit place in $a_1$ is not counted, as it remains as constant value in the normalized format) determines the size of memory (for look-up table to store $a_1^{-1}$) and the number of terms from

TABLE I: Look-up table address space and Required numbers of terms (N), for a given $W$, needed for double precision accuracy

| $W$ | N | Max Absolute Error | | Look-up Table Address Space |
|---|---|---|---|---|
| 6 | 9 | $a_{2max}^{9}$ | $5.551\ E-17$ | $2^6$ |
| 8 | 7 | $a_{2max}^{7}$ | $1.387\ E-17$ | $2^8$ |
| 10 | 6 | $a_{2max}^{6}$ | $8.673\ E-19$ | $2^{10}$ |
| 12 | 5 | $a_{2max}^{5}$ | $8.673\ E-19$ | $2^{12}$ |

the series expansion, to perform the computation for a given precision.

The number of terms ($N$) (with a given $W$) for a given precision requirement ($2^{-P}$) can be determined by following inequality:

$$\begin{aligned}|E_N| &= |a_1^{(N+1)} a_2^N (1 - a_1^{-1}a_2 + a_1^{-2}a_2^2 - a_1^{-3}a_2^3 - \ldots)| \\ &= |\frac{a_1^{(N+1)} a_2^N}{1 + a_1^{-1}a_2}| \leq 2^{-P} \end{aligned} \tag{3}$$

where, $E_N$ is error caused by all the ignored terms in (2). For maximum error, numerator of (3) should be maximum with the minimum value for numerator. So, for most pessimistic estimation (for minimum denominator, let $(1 + a_1^{-1}a_2) \approx 1$, and for maximum numerator let $a_1^{-1} = 1$),

$$|E_N| = |a_2^N| \leq 2^{-P} \tag{4}$$

Thus, it can be seen that for a given precision requirement, increase in $W$ would reduce the required number of terms $N$ and vice-verse. Here, the value of $W$ determines the size of memory (to store the pre-computed $a_1^{-1}$), and $N$ determines the amount of other hardware (multipliers, adders, subtractors). For double precision requirement ($P = 53$), a variation on value of $W$ and required number of terms (N) is shown in Table- I.

For a good balance between $W$ and $N$, bit width of $W = 8$ for $a_1$ is selected, which requires 7 terms (up to $a_1^{-7}a_2^6$) for double precision. Similarly, it needs 3 terms (up to $a_1^{-3}a_2^2$) for single precision requirement with $W = 8$. The respective quotient equation for double and single precision are as follows:

For double precision:

$$\begin{aligned} q &= m_1 \times [a_1^{-1} - a_1^{-2}a_2 + a_1^{-3}a_2^2 - \cdots + a_1^{-7}a_2^6] \\ &= m_1 a_1^{-1} - m_1 a_1^{-1}(a_1^{-1}a_2 - a_1^{-2}a_2^2)(1 + a_1^{-2}a_2^2 + a_1^{-4}a_2^4) \end{aligned} \tag{5}$$

For single precision:

$$\begin{aligned} q &= m_1 \times [a_1^{-1} - a_1^{-2}a_2 + a_1^{-3}a_2^2] \\ &= m_1 a_1^{-1} - m_1 a_1^{-1}(a_1^{-1}a_2 - a_1^{-2}a_2^2) \end{aligned} \tag{6}$$

Here, it can be easily seen that the (6) looks like a sub-set of (5). Thus, both can share the same computational flow. Also, (5) and (6) are framed in such way, so that, the (5) acts as a super-set of both equations as follows:

$$q = \underbrace{\overbrace{m_1 a_1^{-1} - m_1 a_1^{-1}(a_1^{-1}a_2 - a_1^{-2}a_2^2)}^{SP}(1 + a_1^{-2}a_2^2 + a_1^{-4}a_2^4)}_{DP} \tag{7}$$

This interesting feature of (7) forms the basis of sharing hardware resources to efficiently model the dual-mode architecture for mantissa division computation, which is capable of processing either a DP mantissa or two SP mantissa divisions.

The size of look-up table to store $a_1^{-1}$ is taken as $2^8 \times 53$ (13.5 KB) for DP and $2^8 \times 24$ (6 KB) for SP, which provides sufficient precision for remaining computations of DP and SP. To compute all the terms of (7) for dual-mode functioning, a dual-mode multiplier of size 54x54 with dual 24x24 support is designed, which is used iteratively for entire computation of (7). The full mantissa width multiplication is used (for DP as well as dual SPs) for all the computation, which helps in preserving the accuracy of all the terms [9], [2].

Thus, using series expansion method, the mantissa division required following steps:

- Partition divisor mantissa ($m_2$) in two parts, $a_1$ and $a_2$.
- Store the pre-computed value of $a_1^{-1}$ in look-up table.
- Based on required precision, determine the number of series expansion terms to process.
- Compute for mantissa quotient using the value of $a_1^{-1}$, $a_2$ and $m_1$ in finalized expression.

## III. Proposed DPdSP Division Architecture (With 1-Stage Multiplier)

The proposed architecture is shown in Fig. 1. It is composed of three pipelined stages. The details of each stage architecture is discussed below in following subsections one-by-one. Two 64-bit operands, one dividend ($in_1$) and another divisor ($in_2$) are the primary inputs along with the mode-control signal $dp\_sp$ (double precision or dual single precision). Both of the input operands either contains DP operands (as entire 64-bit pair) or two parallel SP operands (as two sets of 32-bit pair), as shown in Fig. 2.

### A. First-Stage Architecture

First stage comprised of steps 2 and 3 of Algorithm 1, which includes the basic processing for data-extraction, exceptional case handling, and sub-normal processing. It also includes the part of mantissa division unit, the pre-fetching of initial approximation of divisor mantissa inverse from look-up table.

The data extraction computation is shown in Fig 3. These take the primary operands and extract the signs ($sp1\_s1$, $sp1\_s2$, $sp2\_s1$, $sp2\_s2$, $dp\_s1$ and $dp\_s2$), exponents ($sp1\_e1$, $sp1\_e2$, $sp2\_e1$, $sp2\_e2$, $dp\_e1$ and $dp\_e2$) and mantissas ($sp1\_m1$, $sp1\_m2$, $sp2\_m1$, $sp2\_m2$, $dp\_m1$ and $dp\_m2$) components for double precision and both single precision, based on their standard formats as shown in Fig 2. The input/output encoding is based on the IEEE standard binary format [20].

The sub-normal (_sn) handling and exceptional checks computations are shown in Fig.4. As the 8 MSB of DP exponent overlap with SP-2 exponent,

$$\overbrace{\underbrace{xxxxxxxx}_{SP-2\ Exponent}\ xxx}^{DP\ Exponent}$$

the checks for sub-normal, infinity and NaN (Not-A-Number) have been shared among SP-2 and DP, as shown in Fig.4. It

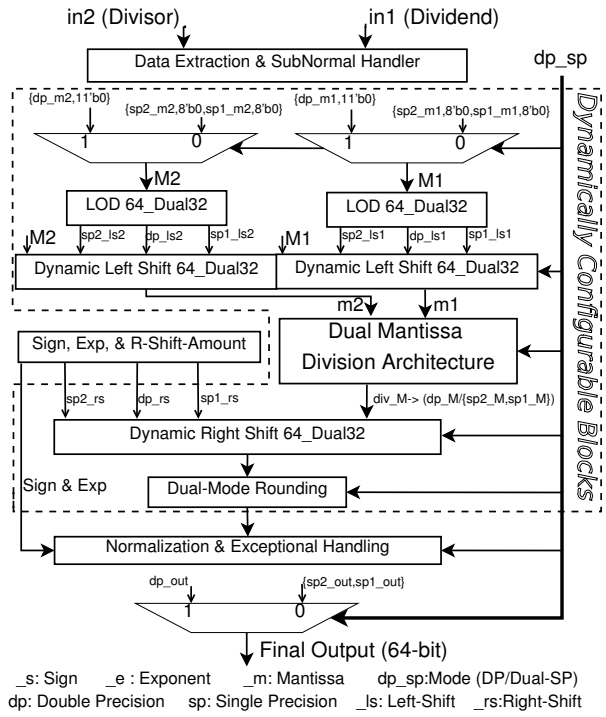


Fig. 1: DPdSP Division Architecture

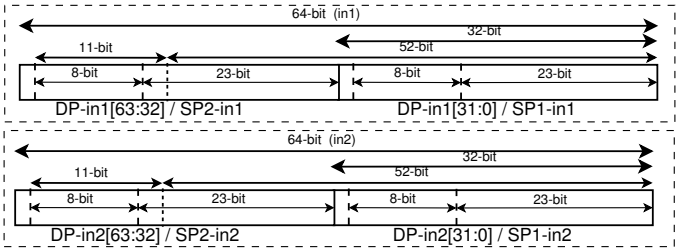


Fig. 2: DPdSP Input Output Format

| | | |
|---|---|---|
| sp1_s1=in1[31] | sp2_s1=in1[63] | dp_s1=in1[63] |
| sp1_s2=in2[31] | sp2_s2=in2[63] | dp_s2=in2[63] |
| sp1_e1={in1[30:24],in1[23] \| sp1_sn1} | sp2_e1={in1[62:56],in1[55] \| sp2_sn1} | dp_e1={in1[62:53],in1[52] \| dp_sn1} |
| sp1_e2={in2[30:24],in2[23] \| sp1_sn2} | sp2_e2={in2[62:56],in2[55] \| sp2_sn2} | dp_e2={in2[62:53],in2[52] \| dp_sn2} |
| sp1_m1={~sp1_sn1,in1[22:0]} | sp2_m1={~sp2_sn1,in1[54:32]} | dp_m1={~dp_sn1,in1[51:0]} |
| sp1_m2={~sp1_sn2,in2[22:0]} | sp2_m2={~sp2_sn2,in2[54:32]} | dp_m2={~dp_sn2,in2[51:0]} |

Fig. 3: DPdSP Division: Data Extraction

SubNormal Checks:

| | | |
|---|---|---|
| sp1_sn1=~\|in1[30:23] | sp2_sn1=~\|in1[62:55] | dp_sn1=~\|in1[56:52] & sp2_sn1 |
| sp1_sn2=~\|in2[30:23] | sp2_sn2=~\|in2[62:55] | dp_sn2=~\|in2[56:52] & sp2_sn2 |
| sp1_sn = sp1_sn1 & sp1_sn2 | sp2_sn = sp1_sn1 & sp2_sn2 | dp_sn = dp_sn1 & dp_sn2 |

Exceptional Checks:

INFINITY:

sp1_inf1=&in1[30:23], sp2_inf1=&in1[62:55], dp_inf1=(&in1[56:52] & sp2_inf1)
sp2_inf2=&in2[30:23], sp2_inf2=&in2[62:55], dp_inf2=(&in2[56:52] & sp2_inf2)

NaN:

sp1_NaN1=&in1[30:22], sp2_NaN1=&in1[62:54], dp_NaN1=(&in1[55:51] & sp2_NaN1)
sp2_NaN2=&in2[30:22], sp2_NaN2=&in2[62:54], dp_NaN2=(&in2[55:51] & sp2_NaN2)

ZERO: sp1_z=~|in1[30:0], sp2_z=~in1[62:32], dp_z=(sp1_z & sp2_z & ~in1[31])

DIV-BY-ZERO: sp1_dbz=~|in2[30:0], sp2_dbz=~in2[62:32], dp_dbz=(sp1_dbz & sp2_dbz & ~in2[31])

Fig. 4: DPdSP Division: Sub-Normal and Exceptional Handling

also performs checks for `divide-by-zero` (_dbz) and `zero` (_z), and have been shared among DP and both SPs.

After data extraction and exceptional checks, a unified set

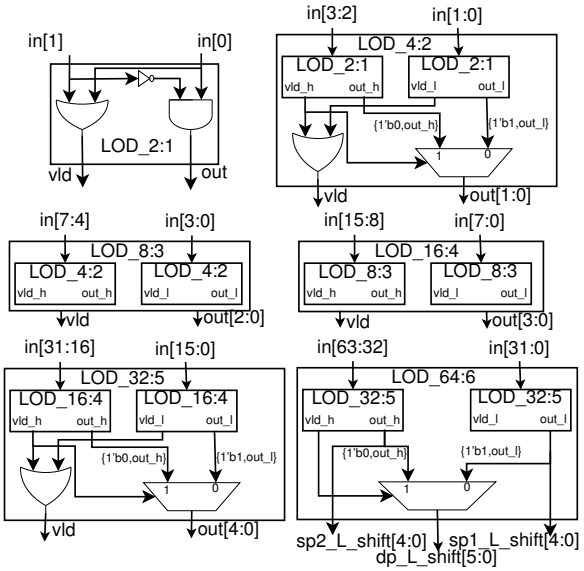


Fig. 5: DPdSP Division: Dual-Mode LOD

of mantissa (M1 and M2) is generated using two MUXes (as shown in Fig. 1). Based on the mode of operation, these contain the mantissa either for DP or for both SPs. This unification of mantissas helps in designing a tuned data-path processing for later stage computation, which results in efficient resource sharing.

The next two units, the leading-one-detector (LOD) and dynamic left shifter, in this stage perform sub-normal processing. They bring the sub-normal mantissa (if any) into the normalized format. First it computes the amount of left-shift using dual-mode LOD and then shifts the mantissa with the dual-mode dynamic left shifter. The corresponding left shifting amount is further adjusted in the exponent. The architecture for dual-mode LOD is shown in Fig 5. The dual-mode LOD is designed in a hierarchical fashion, using a basic building block of 2:1 LOD which consists of a AND, a OR, and a NOT gates. The final 64:6 LOD unit consists of two 32:5 LOD units, which work individually for SPs (on each 32-bit parts of input mantissa), and their output combination provides the left-shifting amount for DP mantissa. The entire resources in dual mode LOD unit is shared among DP and SPs processing, and it costs no overhead compared to only DP LOD.

The unified mantissas (M1 and M2) are then fed in to the dual-mode dynamic left shifter along with the corresponding left-shifting-amount from LOD units (as shown in Fig. 1). In case of DP mode processing, the SPs left-shift amounts are set to zero, otherwise, the DP left shift amounts are set to zero.

The architecture for dual mode dynamic left shifter is shown in Fig. 6. It is a 6-stage barrel shifter unit, in which first stage is a single mode unit, and the 5 remaining stages are dual-mode units. The first stage unit is a simple left-shift barrel shifter which performs shifting only in double precision mode, based on the MSB of DP shift bits. The dual mode stage is presented in generic form in Fig. 6, and it can be extended for any size dual mode design. It works either for DP or for dual SP dynamic left shifting. A dual mode stage contains two

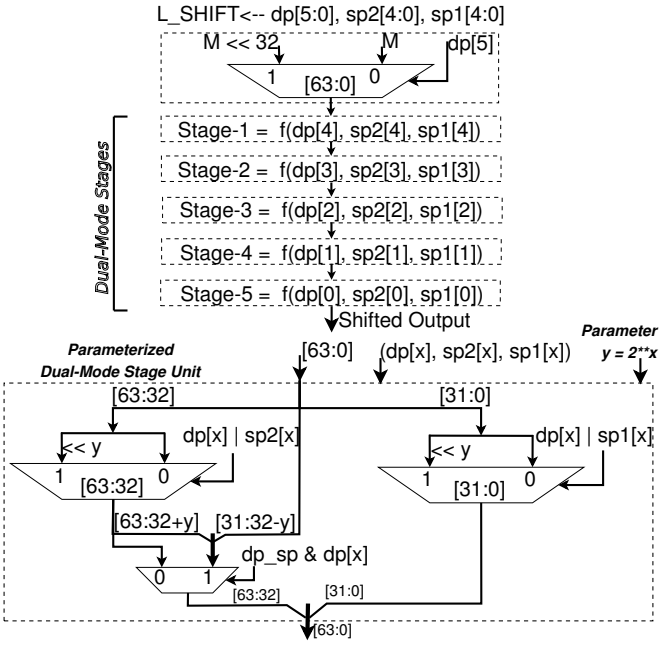


Fig. 6: DPdSP Division: Dual-Mode Dynamic Left Shifter

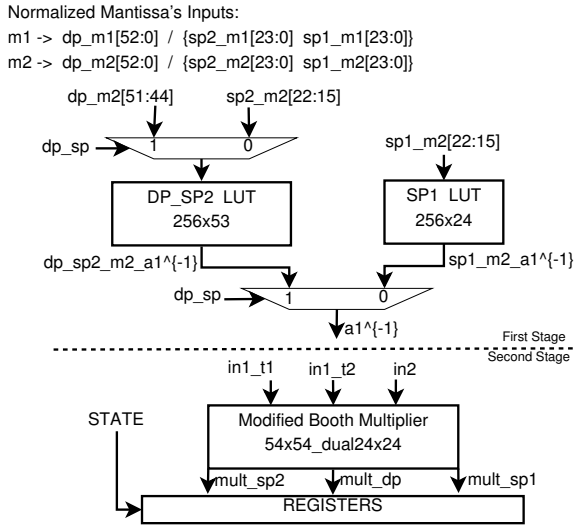


Fig. 7: DPdSP Dual Mode Mantissa Division Architecture

multiplexers for each 32-bit blocks, which shift their inputs based on the corresponding shifting bits (either of DP or both SPs). The shifting amount for a given dual-mode stage is given by $y = 2^x$, where, 'x' is the shifting-bit position for that stage. The dual-mode stage also contains a multiplexer which selects between 32-bit MSB shifting output or their combination with primary 64-bit input to the stage, based on the true $dp\_sp$ and corresponding shifting bit of DP left shift. Except this multiplexer, the dual-mode stages behaves like two separate 32-bit barrel shifter, which are constructed to support dual mode left shifting operation.

After left shifting, mantissas appears into normalized form $m_1$ and $m_2$, as shown in Fig. 1. In the next unit in this stage of division architecture, the 8-bit (after decimal point position) MSB part ($a_1$) of normalized divisor mantissas ($m_2$) are used to

Sign:
sp1_s = sp1_s1 ^ sp1_s2
dp_s = sp2_s = sp2_s1 ^ sp2_s2

Exponent:
sp1_e = 7'h7F + (sp1_e1 - sp1_ls1) - (sp1_e2 - sp1_ls2)
sp2_e = 7'h7F + (sp2_e1 - sp2_ls1) - (sp2_e2 - sp2_ls2)
dp_e = 10'h3FF + (dp_e1 - dp_ls1) - (dp_e2 - dp_ls2)

Right-Shift-Amount:
sp1_rs = (sp1_e2 - sp1_ls2) - 7'h7F - (sp1_e1 - sp1_ls1)
sp2_rs = (sp2_e2 - sp2_ls2) - 7'h7F - (sp2_e1 - sp2_ls1)
dp_rs = (dp_e2 - dp_ls2) - 10'h3FF - (dp_e1 - dp_ls1)

Fig. 8: DPdSP Division: Sign, Exponent and Right Shift Amount

fetch the pre-computed initial approximation of their inverse, as discussed in the Section II. It is shown in the first-stage part of Fig. 7. Two look-up tables are used here. One is shared for DP and SP-2 initial approximation with size of $2^8 \times 53$ (13.5 KB), which acts as $2^8 \times 53$ for DP and $2^8 \times 24$ for SP-2. Other look-up table with size of $2^8 \times 24$ (6 KB) works for SP-1 only. Thus, it requires a total of $(2^8 \times (53+24))$ 19.7 KB memory to store initial approximations of $a_1^{-1}$.

### *B. Second-Stage Architecture*

The second stage architecture performs core operation of division architecture. It computes on the core sign, exponent and mantissa processing of FP division arithmetic and the computation related to right shift amount, which all correspond to the steps 4 and 5 of Algorithm 1.

The computations related to the sign, exponent and right shift amount processing are shown in Fig. 8. The sign computation is a simple XOR operation among both input operands sign bits. The related exponent computation is the difference of dividend ($in_1$) exponent and divisor ($in_2$) exponent, with proper BIASing and the adjustment of mantissa left shift amount (LSA):

$$BIAS + (Exp\_in_1 - LSA\_in_1) - (Exp\_in_2 - LSA\_in_2)$$

In case above computation returns a negative value (i.e. when the effective divisor exponent ($Exp\_in_2 - LSA\_in_2$) is larger than the effective dividend exponent ($Exp\_in_1 - LSA\_in_1$) while including BIASing), the resultant mantissa division result requires to be shifted right by right-shift-amount (RSA). This will results into a subnormal output.

$$RSA = (Exp\_in_2 - LSA\_in_2) - (Exp\_in_1 - LSA\_in_1) - BIAS$$

All these computations are done separately for DP and both SPs.

The mantissa computation is the most complex part of the floating point division arithmetic. Here, it's related architecture includes the unified and dual-mode implementation of (7). As shown in Fig. 7, the initial inverse approximation of divisor mantissa is fetched in first stage of architecture. The remaining computation is built around a dual-mode booth multiplier, in an iterative fashion. A dual-mode finite state machine (FSM) is designed which decides the effective inputs for multiplier in each state and which will be discussed shortly after the description of dual mode multiplier architecture.

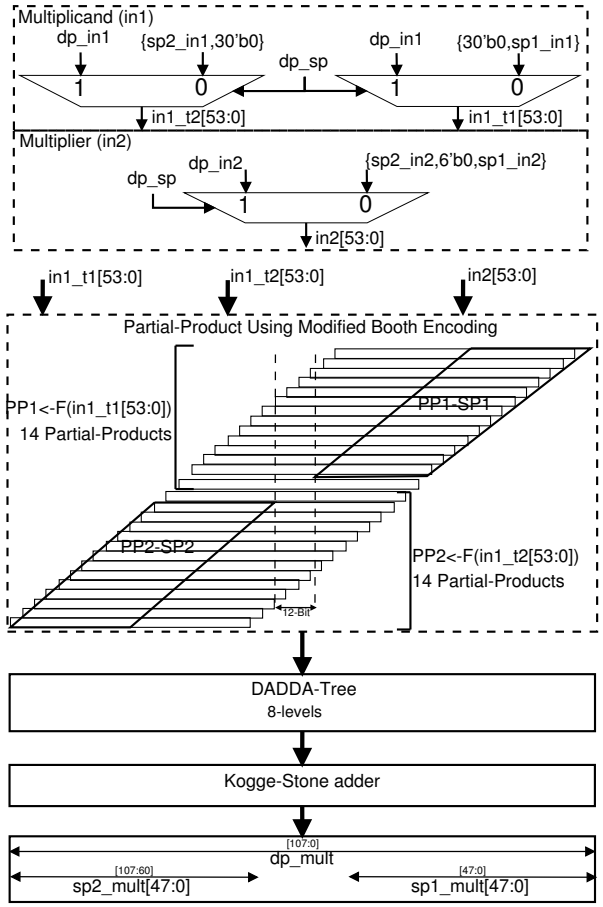


Fig. 9: Dual-Mode Modified Booth Multiplier Architecture

*1) Dual-Mode Radix-4 Modified Booth Multiplier Architecture:* The architecture for the dual-mode Booth multiplier is shown in Fig. 9. The architecture is based on the Radix-4 Modified Booth Encoding, which reduces the number of partial products at most to $(\frac{n}{2}+1)$ [8]. Here, it is a 54-bit integer multiplier (for DP processing), which is also designed to process two parallel sets of 24-bit unsigned operands (for two SPs processing) multiplication. In fact it can process two parallel sets of 26-bit unsigned operand multiplication, using entire 14 partial products of PP1 for first set and entire PP2 for second set operand, without corrupting each other. However, here it is presented for current requirement only.

The presented dual-mode Booth architecture has three input operands (two multiplicands and a multiplier). A set of two inputs ($in_1\_t_1$ and $in_1\_t_2$) forms the multiplicand operands. Here, $in_1\_t_1$ consists of either 'DP multiplicand operand' or 'SP-1 multiplicand operand at the LSB side' and, $in_1\_t_2$ consists of either 'DP multiplicand' or 'SP-2 multiplicand operand at the MSB side'. While, the multiplier input ($in_2$) contains multiplier operands either for DP, or for both SPs with 6-bit zeros in between (see top portion of Fig. 9). Correspondingly, two-sets of partial products (PP1 and PP2) are generated. Partial products PP1 are the result of $in_1\_t_1$ and $in_2$, and PP2 is derived from $in_1\_t_2$ and $in_2$. In DP mode of operation all the partial products of PP1 and PP2 collectively produce the multiplication result. Whereas, in dual-SP mode, first 13 partial products of PP1 (i.e. PP1-SP1, with all MSBs beyond 25th-bit are zero) results for SP-1 and, the last 13 partial products of PP2 (i.e. PP2-SP2, with all the LSBs till 30th-bit are zeros) results for SP-2, while the 14th and 15th partial products will be null, containing all zeros. Here, the inputs $in_1\_t_1$, $in_1\_t_2$ and $in_2$ are built so that, in dual-SP mode processing the single precision partial products (PP1-SP1 and PP2-SP2) and their reduction do not overlap (Fig. 9), and produce two distinct results for SP-1 and SP-2 multiplication, respectively.

Therefore, the sum of all partial products will generate product for DP operands in DP-mode or for both SPs in dual-SP mode. A DADDA-tree of 8 levels is designed to compress all the partial products into two operands, which are further added using a parallel-prefix Kogge-Stone final adder. The final product contains either DP or dual-SP results as shown in Fig. 9. Thus, the few key modification in the Modified Booth multiplication flow leads to a novel dual-mode functioning. The structure of the partial product generation in presented dual-mode Modified Booth multiplier, in terms of hardware requirements, is similar to the contemporary Modified Booth multiplier. However, how the same structure is tweaked (by inclusion of input operands multiplexing, and partial product assignments accordingly) for dual-mode processing, is the novelty of proposal. Compared to the contemporary Modified Booth multiplier, the proposed dual-mode Modified Booth multiplier requires only three 2:1 MUXs as an area overhead, which are needed for the input operands multiplexing.

*2) Dual-Mode Iterative Mantissa Division Architecture:* The mantissa division is designed in an iterative fashion to have an area efficient architecture. The architecture is based on the unified implementation of (7), which can process either a DP mantissa division or two parallel SPs mantissa divisions, with help of above discussed dual-mode modified Booth multiplier. Equation (7) is listed below for an easy reference.

$$\overbrace{\underbrace{m_1 a_1^{-1} - m_1 a_1^{-1}\{(a_1^{-1}a_2 - a_1^{-2}a_2^2)}_{Single\ \ Precision} \times (1 + a_1^{-2}a_2^2 + a_1^{-4}a_2^4)\}}^{Double\ \ Precision}$$

Here, $m_1$ is the normalized dividend mantissa; and $m_2$ is the normalized divisor mantissa, where $m_2$ is partitioned into $a_1$ (first 8-bit right to the decimal point) and $a_2$ (all remaining bits right to the $a_1$).

$$m_2 \rightarrow \overbrace{1.\underbrace{xxxxxxxx}_{8-bit}}^{a_1}\overbrace{\underbrace{xxxxxxx.....xxxxxxx}_{DP:44-bit,\ \ SP:15-bit}}^{a_2}$$

Both, $m_1$ and $m_2$, are appearing in the normalized format from first-stage processing, and they contain either DP mantissas ($dp\_m_1[52:0]$ and $dp\_m_2[52:0]$) or both SPs mantissas ($sp_1\_m_1[23:0]$, $sp_1\_m_2[23:0]$, $sp_2\_m_1[23:0]$ and $sp_2\_m_2[23:0]$), as shown in Fig. 7.

Here, for the ease of understanding the later description, various combinations of terms in above unified equation are listed as follows:

$$A = m_1 a_1^{-1}, \quad B = a_1^{-1} a_2, \quad C = a_1^{-2} a_2^2, \quad D = a_1^{-4} a_2^4$$
$$E = a_1^{-1} a_2 - a_1^{-2} a_2^2, \quad F = 1 + a_1^{-2} a_2^2 + a_1^{-4} a_2^4, \quad G = EF$$
$$H_{DP} = AG, \quad H_{SP} = AE, \quad I = A - H \tag{8}$$

Thus, from above set of abbreviations, for SPs computation, it only requires to skip the computation of $D$, $F$, $G$ and $H_{DP}$ from DP flow. The implementation is achieved by designing a FSM, which consists of 9 states (S0 to S8). In each state of FSM, inputs ($in_1\_t_1, in_1\_t_2$ and $in_2$) for dual-mode modified booth multiplier are determined, and its output is assigned to the designated terms. It is shown below in (9).

$$\begin{aligned}
S_0:\ & in_1_t_1 = dp_sp\ ?\ \{1'b0, dp_m_1\}\ :\ \{30'b0, sp_1_m_1\} \\
& in_1_t_2 = dp_sp\ ?\ \{1'b0, dp_m_1\}\ :\ \{sp_2_m_1, 30'b0\} \\
& in_2 = dp_sp\ ?\ \{1'b0, dp_m_2_a_1^{-1}\} \\
& \qquad :\ \{sp_2_m_2_a_1^{-1}[52:29], 6'b0, sp_1_m_2_a_1^{-1}\} \\
S_1:\ & in_1_t_1 = dp_sp\ ?\ \{10'b0, dp_m_2_a_2\} : \{30'b0, 9'b0, sp_1_m_2_a_2\} \\
& in_1_t_2 = dp_sp\ ?\ \{10'b0, dp_m_2_a_2\} : \{9'b0, sp_2_m_2_a_2, 30'b0\} \\
& in_2 = dp_sp\ ?\ \{1'b0, dp_m_2_a_1^{-1}\} \\
& \qquad :\ \{sp_2_m_2_a_1^{-1}[52:29], 6'b0, sp_1_m_2_a_1^{-1}\} \\
& A[63:0] = dp_sp\ ?\ dp_mult[105:42] \\
& \qquad :\ \{sp_2_mult[47:16], sp_1_mult[47:16]\} \\
S_2:\ & in_1_t_1 = dp_sp\ ?\ dp_mult[96:43]\ :\ \{30'b0, sp_1_mult[38:15]\} \\
& in_1_t_2 = dp_sp\ ?\ dp_mult[96:43]\ :\ \{sp_2_mult[38:15], 30'b0\} \\
& in_2 = dp_sp\ ?\ dp_mult[96:43] \\
& \qquad :\ \{sp_2_mult[38:15], 6'b0, sp_1_mult[38:15]\} \\
& B[63:0] = dp_sp\ ?\ dp_mult[96:43] \\
& \qquad :\ \{sp_2_mult[38:12], sp_1_mult[38:12]\} \\
S_3:\ & in_1_t_1 = in_1_t_2 = in_2 = dp_mult[107:54], \quad C_{DP} = dp_mult \\
& C = dp_sp\ ?\ \{8'b0, dp_mult[107:62]\} \\
& \qquad :\ \{8'b0, sp_2_mult[47:29], 8'b0, sp_1_mult[47:29]\} \\
& E = B - C \\
S_4:\ & in_1_t_1 = in_1_t_2 = in_2 = 0, \quad D_{DP} = dp_mult[107:87] \\
& F_{DP}[53:0] = \{1'b1, 16'b0, C_{DP}[107:71]\} + \{33'b0, D_{DP}\} \\
S_5:\ & in_1_t_1 = in_1_t_2 = E \quad in_2 = F_{DP} \\
S_6:\ & in_2 = A, \quad G = dp_mult[107:54] \\
& in_1_t_1 = dp_sp\ ?\ G\ :\ \{30'b0, E[26:3]\} \\
& in_1_t_2 = dp_sp\ ?\ G\ :\ \{E[26:3], 30'b0\} \\
S_7:\ & in_1_t_1 = in_1_t_2 = in_2 = 0, \quad AG = \{7'b0, dp_mult[107:51]\} \\
& AE = \{8'b0, sp_2_mult[47:24], 8'b0, sp_1_mult[47:24]\} \\
& H = dp_sp\ ?\ AG\ :\ AE \\
S_8:\ & I = A - H, \quad in_1_t_1 = in_1_t_2 = in_2 = 0
\end{aligned} \tag{9}$$

The finite state machine (FSM) is shown in Fig. 10. For DP processing it goes through all the states, whereas for dual-SP it skips states $S4$ and $S5$ which performs only DP related computations. The selection of bits for a term is based on the position of decimal point and processing-mode. Generally, for DP mode, the multiplications are done in 54-bit (sufficient for it's precision requirement) and add/sub are performed in 64-bit (to preserve precision), whereas, for dual-SPs, the multiplications are done in 24-bit and add/sub are performed in 32-bit.

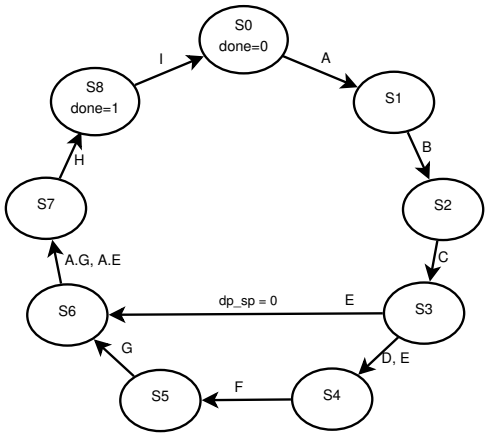


Fig. 10: DPdSP Dual-Mode Iterative Mantissa Division FSM

In state $S_0$ of FSM, the dividend mantissas are applied to $in_1\_t_1$ and $in_1\_t_2$ and, $a_1^{-1}$ is applied at $in_2$, either for DP or both SPs. This produces $A = m_1 a_1^{-1}$ in the next state. In state $S_1$, $a_2$ is applied at $in_1\_t_1$ and $in_1\_t_2$, either for DP or both SPs, and $in_2$ remains with $a_1^{-1}$. This produces $B = a_1^{-1} a_2$ in the next state. The state $S_2$ applies previous multiplication output ($B = a_1^{-1} a_2$) to all the multiplier inputs in desired format, based on the mode of operation, which produces $C = a_1^{-2} a_2^2$ in the next state. Further, the $C_{DP}$, the DP form of $C$, is applied to all inputs, in the state $S_3$, as the corresponding output is meant only for DP processing, which produces $D_{DP} = a_1^{-4} a_2^4$ in the next state. State $S_3$ also processes the term $E = B - C$. All inputs of multiplier are set to zero in state $S_4$, and this state processes the term $F_{DP}$. The state $S_5$ applies $E$, in DP format, at $in_1\_t_1$ and $in_1\_t_2$ and, $F_{DP}$ at $in_2$, and this produces $G$ in the next state. The next state $S_6$ applies either $G$ or "$E$ in SP format only" at the multiplicand inputs $in_1\_t_1$ and $in_1\_t_2$ and, $A$ at the multiplier input $in_2$, which produces $AG$ or $AE$ in the next state. In state $S_7$, based on the mode of operation, the term $H$ will be either of $AG$ or $AE$. And finally, the last term $I = A - H$ is computed in state $S_8$, with the asserted `done` signal. The term $I$ contains the mantissa quotient either for DP (as a whole) or for two SPs (in each 32-bit portion).

The mantissa division FSM requires 9 cycles for DP-mode processing and 7-cycles for dual-SPs processing. Compared to the only DP mantissa division FSM (as discussed in Section V), the DPdSP mantissa division FSM requires 14 54-bit 2:1 MUXs as an overhead.

### C. Third-Stage Architecture

The third stage of the FP division architecture corresponds to the computations of steps 6,7 and 8 of the Algorithm 1. In this stage, for the case of exponent underflow (if dividend exponent is smaller than the divisor exponent), mantissa division quotient is first process for the dynamic right shifting. This is followed by the dual-mode rounding of the quotient mantissa, and then it undergoes normalization and exceptional case processing.

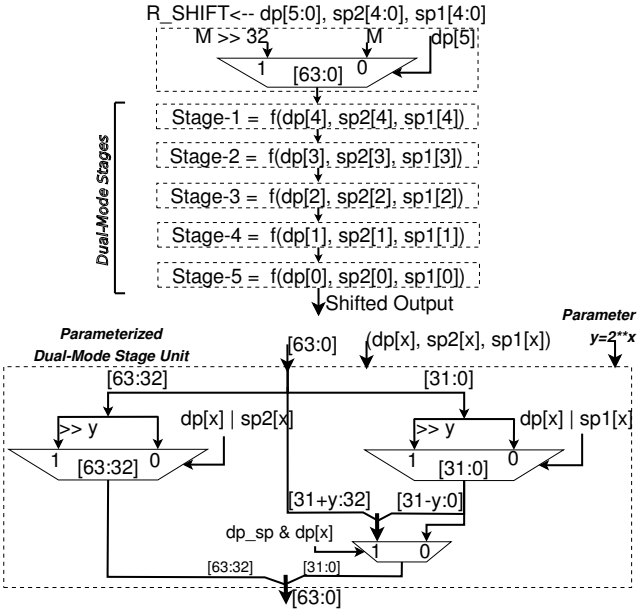


Fig. 11: DPdSP Dual-Mode Dynamic Right Shifter

*1) Dual-Mode Dynamic Right Shifting:* The architecture for dual-mode dynamic right shifter is shown in Fig. 11. The right-shift-amount (Fig. 8) and mantissa quotient acts as primary inputs to the dual-mode dynamic right shifter. Similar to the dual-mode dynamic left shifter architecture, this unit has 6-stages, the first-stage is single-mode for DP processing only, and 5 remaining stages are dual-mode in nature. The generic architecture for dual-mode stage is shown in Fig. 11. The top two MUXs work for each SP mantissa quotient dynamic right shifting, which are combined by the third MUX to produce the DP mantissa dynamic right shifting.

*2) Dual-Mode Rounding:* The proposed dual mode architecture is designed with faithful rounding, using `round-to-nearest` method. It is comprised of two steps, first the unit-at-last-place (ULP) computation and then ULP addition with quotient mantissa. ULP computation is based on the rounding position bit, Guard-bit, Round-bit and Sticky-bit. The rounding position is determined by the MSB of the quotient mantissa. Guard and Round bits are next to the rounding position bit at LSB side, while all the remaining LSB bits of mantissa quotient produce the Sticky-bit.

The ULP-computation is done separately for DP and both SPs quotients, and each of them requires few logic-gates for this purpose, as shown in Fig. 12. Whereas, the ULP-addition with quotient mantissa is shared among DP and both SPs. As, mantissa quotient contains either DP or dual-SPs quotients, its ULP-addition is shared as shown in Fig. 12. It is done using two 32-bit incrementer, which individually acts like a SP ULP-adder, however, their combination (by propagating carry) also performs for DP ULP-addition. Thus, the dual-mode rounding requires few logic-gates (for SPs ULP-computation) and two 1-bit 2:1 MUXs (for ULP-addition) as an overhead over single-mode DP rounding.

The correct rounding in multiplicative based division methods requires the computation of remainder to produce the

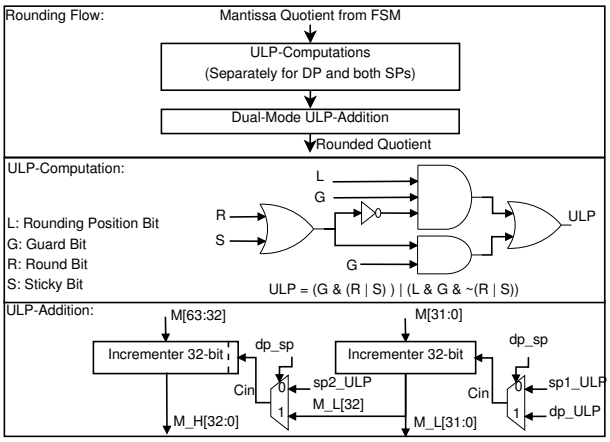


Fig. 12: DPdSP Dual-Mode Rounding

Sticky-bit [2], [9]. The remainder computation requires one more multiplication and a subtraction, which would need two more states in the FSM with 1-stage multiplier, and 3 more states in FSM with 2-stage multiplier. These can be included based on the requirement. Also, similar to the `round-to-nearest` rounding method, other methods can also be included based on the requirement. As, the ULP computation can not be shared among DP and SPs and it requires small logical computation, it is only presented for `round-to-nearest` method, which is most widely used.

*3) Final Processing:* The rounded mantissa quotient is further normalized separately for DP and both SPs, in-case of any mantissa overflow due to rounding ULP-addition. It requires 1-bit right shift for overflow. And correspondingly, due to the ULP-addition overflow, the exponents are incremented by one accordingly for DP and both SPs, separately. Further to this, each exponent and mantissa is updated for exceptional cases infinity, divide-by-zero, NaN, along with with overflow handling (as shown in Algorithm 2), and the output underflow cases produce the computed subnormal outputs, which all needs separate units for DP and both SPs.

**Algorithm 2** Exceptional Case Processing at Output

1: $(IN1\ (Dividend), IN2\ (Divisor))$;
2: **if** IN1 = NaN or IN2 = NaN or IN1=IN2=INFINITY or IN1=IN2=ZERO **then**
3: IN1/IN2 ← NaN
4: **else if** (IN2 = INFINITY & IN1 != INFINITY/NaN) or IN1 = ZERO **then**
5: IN1/IN2 ← ZERO
6: **else if** (IN1 = INFINITY & IN2 != INFINITY/NaN) or (IN2 = ZERO → DIV-BY-ZERO) or (Output OVERFLOW) **then**
7: IN1/IN2 ← INFINITY
8: **else if** For Normal or Underflow Case **then**
9: IN1/IN2 ← Computed Results

Finally, the computed signs, exponents and mantissas for double precision and both single precision are multiplexed using a 64-bit 2:1 MUX to produce the final 64-bit output floating point quotient result, which either contains the DP quotient or two SPs quotients .

A brief summary of extra resource overhead of proposed DPdSP division architecture over only DP division (DP division is presented in Section V) is shown in Table-II.

TABLE II: Resource Overhead in DPdSP over DP only Division

| DPdSP Sub-Components | Extra resource over DP only |
|---|---|
| SubNormal & Exceptional case handler | For one SP computations |
| LOD for SubNormal Processing | Two 64-bit 2:1 MUXs to generate unified mantissa, but no overhead in LOD component |
| Dynamic Left/Right Shifter | Minor overhead |
| Sign, Exponent and Right-Shift-Amount | Overhead of both SPs computations |
| Mantissa Division | One 256x24 Look-up table for SP-1, and 14 64-bit 2:1 MUXs in FSM |
| Rounding | ULP-computation of both SP's |
| Normalization & Final Processing | Processing of both SP's & one 64-bit 2:1 MUX |

## IV. Proposed DPdSP Division Architecture (With 2-Stage Multiplier)

This architecture is aimed to improve the speed of the previous architecture along with to demonstrate the trade-offs among various design metrics between two architecture. This also shows, how to use multiple stage multiplier for the current purpose.

This architecture uses a two-stage dual-mode multiplier for mantissa division computation, and to manage the critical path, all stages of previous architecture are partitioned into two pipelined stages as discussed below.

The first stage of previous architecture is split into two-stages by inserting a pipeline register in the dual-mode dynamic left shifter unit (after it's fourth stage). In the second stage of previous architecture, the sign, exponent and right-shift-amount related processing are still compressed in a single stage due to smaller delay of these computations, whereas, in the mantissa division part, the dual-mode Modified Booth multiplier is pipelined in 2 stages. A pipeline register is inserted after the 6th level of DADDA-tree in the multiplier architecture. Further, a pipelined register is inserted after the dual-mode rounding processing in the third stage of previous architecture. After insertion of these pipeline registers, it becomes a 6-stage architecture. All processing in the 6-stage architecture remains the same, except the mantissa division FSM processing.

The mantissa division FSM is redesigned for this case and is shown in Fig. 13. Instead of a regular FSM presentation, here it is presented over the previous FSM (Fig. 10), to show a clear difference among both and for better understanding. It consists of 14-states. For DP processing it goes through all the states, and for dual-SP processing it skips 4 states. Compare to the previous FSM (with 1-stage multiplier), a NOP state is inserted, whenever, the output of the multiplier becomes an input in next state computation, like the insertion of states $S_2\_T$, $S_3\_T$, $S_5\_T$ and $S_6\_T$ with all inputs of the multiplier asserted to zero in these states. Whereas, when input in next state does not depends on the multiplier output, the continuous inputs are provided in a pipelined fashion, like the insertion of state $S_1\_T$. Except the transfer of 1) the assignment of term $A$ from state $S_1$ (in previous FSM) to state $S_1\_T$ (in new FSM) and 2) transfer of the multiplier input assignments from state $S_2$ (in previous FSM) to $S_1\_T$ (in new FSM), the processing in the states $S_0$, $S_1$, $S_2$, $S_3$, $S_4$, $S_5$, $S_6$, $S_7$ and $S_8$ are the same as discussed in (9).

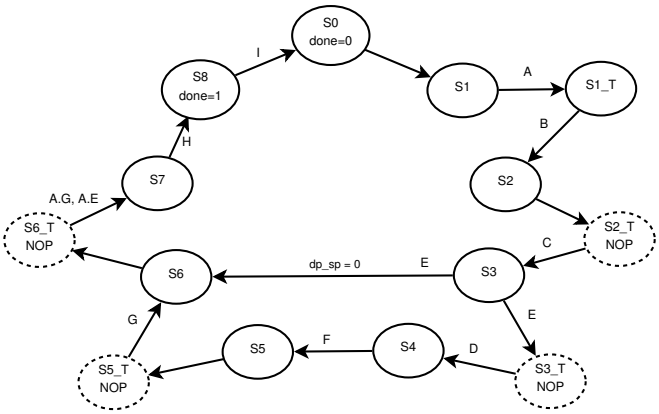


Fig. 13: DPdSP Dual-Mode Iterative Mantissa FSM with 2-Stage Multiplier

## V. DP Division Implementation

Based on a similar computational flow, architectures for only double precision division is also implemented for the purpose of comparison. Both architectures, with 1-stage multiplier and 2-stage multiplier are constructed respectively. In this, all the computations are done only for the double precision related processing and can be easily sought from the above descriptions of DPdSP architectures. All the sub-components (data extraction unit, sub-normal and exceptional checks unit, LOD, Dynamic Left/Right shifter, look-up table for $a_1^{-1}$, mantissa division unit, normalization, rounding and final updates units) are implemented in their single-mode format. A 53-bit single mode Modified Booth multiplier is used in it's mantissa division unit. The processing in it's mantissa division FSM (with 1-stage multiplier) is shown in (10), and the state transition would be similar to the DP flow of Fig. 10. Likewise, it is implemented for FSM with 2-stage multiplier. Here also, there are 9-states in FSM of division with 1-stage multiplier architecture, and 14-states with 2-stage multiplier architecture, both without any SP processing. All the pipeline registers in double precision architectures are placed at the similar levels as that in DPdSP division architectures.

$$
\begin{aligned}
&S_0: \ in_1 = dp\_m_1, && in_2 = dp\_m_2\_a_1^{-1} \\
&S_1: \ in_1 = \{9'b0, dp\_m_2\_a_2\}, && in_2 = dp\_m_2\_a_1^{-1} \\
&\quad A[63:0] = dp\_mult[105:42] \\
&S_2: \ in_1 = dp\_mult[96:44], && in_2 = dp\_mult[96:44] \\
&\quad B[63:0] = dp\_mult[96:43] \\
&S_3: \ in_1 = in_2 = dp\_mult[105:52], \quad C = dp\_mult, \quad E = B - C \\
&S_4: \ in_1 = in_2 = 0, \quad D = dp\_mult[105:86] \\
&\quad F[52:0] = \{1'b1, 16'b0, C[105:70]\} + \{33'b0, D\} \\
&S_5: \ in_1 = E, && in_2 = F \\
&S_6: \ in_1 = G = dp\_mult[105:52], && in_2 = A \\
&S_7: \ H = AG = \{7'b0, dp\_mult[105:49]\}, && in_1 = in_2 = 0 \\
&S_8: \ I = A - H, && in_1 = in_2 = 0
\end{aligned}
\tag{10}
$$

TABLE III: ASIC Implementation Details @ UMC 90nm

| | With 1-Stage Multiplier | | With 2-Stage Multiplier | |
|---|---|---|---|---|
| | DP | DPdSP | DP | DPdSP |
| Latency (cycle) | 11 | 11/9[#] | 18 | 18/14[#] |
| Throughput (cycle) | 10 | 10/8[#] | 15 | 15/11[#] |
| Area ($\mu m^2$) | 167300 | 199249 | 175802 | 206701 |
| Gate Count[1] | 55767 | 66416 | 58601 | 68900 |
| Period (ns) | 1.66 | 1.72 | 0.93 | 0.98 |
| Period (FO4)[2] | 36.89 | 38.22 | 20.67 | 21.78 |
| Power (mW) | 30.67 | 30.9 | 61.87 | 64.21 |

[1] Based on minimum size inverter [#]Cycle Counts for DP/dSP.
[2] Fan-Out-of-4 Delay

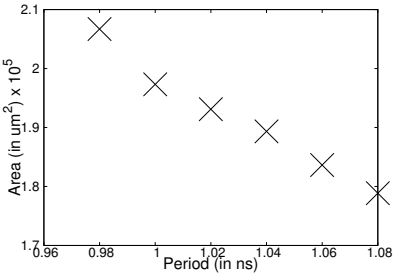


Fig. 14: Area-Period Variation of DPdSP Division Architecture with 2-Stage Multiplier @ UMC 90nm

## VI. Implementation Results

The proposed dual-mode DPdSP iterative division architectures, along with single mode (only) double precision architectures, are synthesized with UMC 90*nm* standard cell ASIC library, using Synopsys Design Compiler. The implementation details are shown in Table-III. The designs are synthesized with best achievable timing constraints, with constraint of `max-area` set to zero and global operating voltage of 1V. A variation of area with respect to the time period constraint, for DPdSP division architecture with 2-stage multiplier, is also shown in Fig 14. The first point of Fig 14 corresponds to the best-timing result, which is presented in Table-III and is used for comparison purpose. The total latency of the architectures consists of pre-FSM cycles, FSM cycles and post-FSM cycles. Also, as the proposed architectures are iterative in nature, every next input can be applied soon after FSM finishes the current processing. Thus, the DPdSP division architecture with 1-stage multiplier will have a latency of 11 cycles and throughput of 10 cycles for DP computation, a latency of 9 cycles and throughput of 8 cycles for dual-SP computations. Similarly, in case with 2-stage multiplier, for DP mode latency and throughput are 18 cycles and 14 cycles, respectively, and for dual-SP mode, these are 15 cycles and 11 cycles, respectively.

The DPdSP division architecture with 1-stage multiplier requires $\approx 19\%$ more hardware resources and $\approx 3.6\%$ more period than only DP division architecture, whereas with 2-stage multiplier it needs $\approx 17.5\%$ more resources and $\approx 5.3\%$ more period than only DP architecture. The Area/Period overhead is calculated as (DPdSP - DP)/DP.

### A. Functional Verification

The functional verification of the proposed dual-mode DPdSP division architecture is carried out using 5-millions random test cases for each of the normal-normal, normal-subnormal, subnormal-normal and subnormal-subnormal operands combination, along with the other exceptional case verification, for both DP and dual-SP mode. It produces a maximum of 1-ULP (unit at last place) precision loss. It is compared against the fully IEEE compliant digit-recurrence method with round-to-nearest rounding method. The method used for the mantissa division in proposed architecture is able to produce faithful rounded result [5]. Faithful rounding result is suitable for a large set of applications. Rounding methods in multiplicative-based division methods (Newton-Raphson, Goldschmidt, Series-Expansion) is itself a key research area, and [2], [21] contains a rich information on different methods used for this. Accordingly, the correctly rounded result in proposed architecture can be obtained by processing one more full multiplication [9], [2], as discussed in Section III-C2.

### B. Related Work and Comparisons

A comparison with prior art on dual-mode division architecture is shown in Table-IV. A technological independent comparison is presented in terms of Gate-Count for area, FO4-delay for timings, and latency and throughput in terms of clock-cycle counts. Comparison is also made in term of a unified metric, $Area \times Period\ (FO4) \times Throughput\ (in\ clock-cycle)$, which should be smaller for a better design.

We have also included, in Table-IV, the synthesis results of proposed architectures with "only normal" computation support. This is achieved by removing the components related to the Step-3 and Step-6 of Algorithm 1. Step-3 corresponds to the pre-normalization of subnormal operands and it consists of LOD and Dynamic-Left-Shifter in Fig. 1. Whereas Step-6 corresponds to post-normalization of any subnormal quotient mantissa and it requires a dynamic right shifter. Thus, in this architecture subnormals are treated as zero, both at the input and output side. These changes will reduce the latency of the architecture (with 2-stage multiplier) by two clock cycle, as the first and third stages remain under single cycle processing. However, in the architecture with 1-stage multiplier, the latency will remains same for "only normal" support and "with subnormal" computational support. Further, since the throughput of proposed architectures depends of the mantissa division FSM processing, it remains same for architecture with "only normal" and "with subnormal" processing support.

Moreover, for a fair comparison, similar to [18], we have also included the synthesis results of our proposed architectures and [19] using TSMC 250nm based Standard cell ASIC library implementation; with best achievable timing, constraint of max-area set to zero, and global operating voltage of 2.5V.

In [18], Isseven *et. al.* has proposed an iterative dual-mode DPdSP division architecture using Radix-4 SRT division algorithm, a digit-recurrence method, and synthesized using TSMC 250nm library. Their architecture has a throughput of 29 clock cycles for double precision, and 15 clock cycles for single precision. Their architecture is designed only for normal support and can not process sub-normal computations and exceptional cases.

TABLE IV: Comparison of DPdSP Division Architecture

| | [18] | | [19] | | | Proposed (with 1-Stage Multiplier) | | | | Proposed (with 2-Stage Multiplier) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SubNormal | × | | ✓ | | | ✓ | | × | | ✓ | | × | |
| Tech. | 250nm | 90nm[#] | 180nm | 250nm | 90nm | 250nm | 90nm | 250nm | 90nm | 250nm | 90nm | 250nm | 90nm |
| Gate Count[1] | 212854 | 212854 | 163194 | 167999 | 118085 | 60231 | 66416 | 56041 | 62649 | 60842 | 68900 | 55537 | 60468 |
| Area (in $\mu m^2$) | - | 638562 | 2611098 | 6047994 | 354257 | 2168328 | 199249 | 2017485 | 187949 | 2190336 | 181406 | 1999344 | 206701 |
| Period (FO4)[2] | 31.4 | 31.4 | 437.5 | 432 | 422.2 | 45.92 | 38.22 | 45.6 | 37.11 | 26.24 | 21.78 | 26.08 | 21.78 |
| Period (in *ns*) | 3.92 | 1.413 | 39.38 | 54 | 19 | 5.74 | 1.72 | 5.7 | 1.67 | 3.28 | 0.98 | 3.26 | 0.98 |
| Power (mW) | - | - | 135.63 | 317.27 | 11.92 | 331.99 | 30.9 | 332.63 | 33.45 | 664.06 | 64.21 | 587.11 | 54.81 |
| Power(mW)×Period(ns) | - | - | 5341 | 17132.6 | 226.48 | 1905.62 | 53.15 | 1895.99 | 55.86 | 2178.12 | 62.93 | 1913.98 | 53.71 |
| Throughput[3] | 29/15 | | 1/1 | | | 10/8 | | | | 15/11 | | | |
| Area × Period × Throughput ($10^6$) [4] | 193.82 | | 71.39 | 72.57 | 49.85 | 27.65 | 25.38 | 25.55 | 23.24 | 23.94 | 22.50 | 21.72 | 19.75 |

[#] Scaled at 90nm [1]Based on minimum size inverter [2]1 FO4 (ns) ≈ (Tech. in $\mu m$) / 2 [3]in clock-cycles for DP/dSP [4]Gate Count × Period (FO4) × Throughput

In comparison to the currently proposed DPdSP iterative division architecture, Isseven *et. al.*'s dual-mode architecture requires much larger area. The throughput of the proposed DPdSP architectures for double precision processing (10 and 15 cycles for 1-stage and 2-stage, respectively) is better than Isseven *et. al.*'s architecture throughput (29 cycles). For dual single precision processing also, the throughput of Isseven *et. al.* is larger than the proposed work. The unified metric $Area \times Period\ (FO4) \times Throughput\ (in\ cycles)$ of the proposed architecture is much better than the architecture of Isseven *et. al.*'s architecture.

The previous single cycle implementation of a similar multiplicative based dual-mode architecture presented in [19] requires an area of 168K equivalent gates with a value of $72 \times 10^6$ for $Area \times Period\ (FO4) \times Throughput\ (in\ cycles)$. Due to its single cycle implementation, this design is not practical. Thus, in comparison to the architecture in [19], the currently proposed architectures are better in terms of design metrics and are practical in nature.

To the best of author's knowledge, literature does not contains any other dual-mode division architecture, which can support DP with two parallel SP divisions. Thus, due to limited literature of dual-mode DPdSP division architecture, a discussion based on the methods used in some (only) double precision (DP) implementation is provided here.

A FPGA based SRT digit-recurrence double precision division architecture is proposed by Hemmert *et al.* [22] with 62 cycles latency and 4100 slices. Thus, as also seen in the case of Isseven *et. al.*'s DPdSP division work using SRT method, this method needs a larger latency and area. Also, performance of digit-recurrence method lacks behind multiplicative methods [2].

Antelo *et al.* [23] has proposed a combination digit-recurrence approximation and Newton-Raphson (NR) iteration on a FPGA platform, and its implementation for double precision requires an address space of 15-bit (*approx* 28 18k BRAMs on Xilinx FPGA), and an equivalent of 29 MULT18x18 IPs. Malik [24] has recently presented the single precision division implementation of FPGA device using Newton-Raphson and Goldschmidt methods. Pasca [9] has proposed a combination of polynomial approximation and NR method on an Altera Stratix V device, which usages roughly 1000 ALUTs (ALUTs on Stratix is computationally richer that Xilinx LUTs) and an Xilinx equivalent of 4 BRAMs and 35 multiplier IPs. Wang *et al.* [25] has reported a 41-bit (10-bit exp and 29-bit mantissa) floating point format division architecture. It requires a large area with 62 BRAMs, and reported to have precision loss. This method requires a huge look-up table with address space of half size of operands, ie $2^{27}$ for DP. Another DP division implementation presented by Fang *et al.* [26] is based on an initial approximation with Goldschmidt method. This method for DP division needs a look-up table with address space of $2^{14}$, and some multipliers. A recent article appeared in [27] discussed a methodology for single mode division, however, with in a restricted scope of normalized operands and also without an actual implementation. All these discussed architectures are in fully unrolled form. These methods for double precision division, require large look-up tables (some are impractical), however, their iterative implementation would require only one multiplier, as in case of proposed architecture. Moreover, as these methods are discussed only for single mode (or only) double precision architecture, a thorough investigation is required on their feasibility and suitability for dual-mode implementation. This study is the part for our future endeavor on this research.

Some processing architectures also relies on the software implementation of (single mode) division arithmetic using FMA (fused multiply-add) instructions. Like, IA-64 [28] implement it, first by computing an initial approximation of reciprocal of divisor and then computing the quotient using a sequence of FMA instructions. This approach is based on the work [29]. In the similar scope, idea of current approach might also be used as a technique using FMA. Like, in traditional FMA division approach, the initial approximation of reciprocal is taken as a seed for other computation, here, $a_1^{-1}a_2$ would act as a seed for all other computations to produce the quotient, and thus, inherits the benefit of FMA. Though, this description/idea is still half-baked, it may be an interesting approach provided that FMA architecture incorporates the idea of proposed dual-mode Modified Booth multiplier, and thus, it will provide a dual-mode area-efficient functioning. Our future work will focus on this subject in more details.

## VII. Conclusions

This paper has presented two dual-mode iterative architectures for double precision floating point division arithmetic. It can be dynamically configured for double precision with dual single precision (DPdSP) floating point division arithmetic. Two architectures with different pipeline levels, 3 and 6 stages architectures, comprised of 1-stage multiplier

and with 2-stage multiplier, respectively, are proposed with area, period and throughput trade-offs. The mantissa division is based on the series expansion methodology of division arithmetic. All the components are designed for efficient dual-mode processing. A novel dual-mode Radix-4 Modified Booth multiplier architecture is proposed with minimal overhead, for the purpose of dual-mode mantissa processing. The entire data path has been tuned to perform the dual mode computation with minimal hardware overhead. The proposed dual-mode iterative DPdSP architecture have $\approx 17\% - 19\%$ area and $\approx 3\% - 5\%$ period overhead over DP only architecture. The proposed architecture outperforms the prior art on this in terms of required area, period, throughput in cycles, and unified metric $Area \times Period\ (FO4) \times Throughput\ (in\ cycles)$.

Based on the current proposed DPdSP division architecture, similar architectures for dual-mode division can be formed using other multiplicative based methods (like Newton-Raphson, Goldschmidt) of division. Our future work on this will be targeting these architectures.

## REFERENCES


[1] J.-C. Jeong, W.-C. Park, W. Jeong, T.-D. Han, and M.-K. Lee, "A cost-effective pipelined divider with a small lookup table," *Computers, IEEE Transactions on*, vol. 53, no. 4, pp. 489–495, 2004.

[2] S. F. Obermann and M. J. Flynn, "Division algorithms and implementations," *Computers, IEEE Transactions on*, vol. 46, no. 8, pp. 833–854, Aug. 1997.

[3] M. K. Jaiswal and R. C. C. Cheung, "High Performance Reconfigurable Architecture for Double Precision Floating Point Division," in *8th International Symposium on Applied Reconfigurable Computing (ARC-2012)*. Hong Kong: Springer LNCS, March 2012, pp. 302–313.

[4] X. Wang and M. Leeser, "Vfloat: A variable precision fixed- and floating-point library for reconfigurable hardware," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 3, no. 3, pp. 16:1–16:34, Sep. 2010.

[5] M. K. Jaiswal, R. Cheung, M. Balakrishnan, and K. Paul, "Series expansion based efficient architectures for double precision floating point division," *Circuits, Systems, and Signal Processing*, vol. 33, no. 11, pp. 3499–3526, 2014. [Online]. Available: http://dx.doi.org/10.1007/s00034-014-9811-8

[6] J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres, *Handbook of Floating-Point Arithmetic*, 1st ed. Birkhäuser Basel, 2009.

[7] P. Soderquist and M. Leeser, "Area and Performance Tradeoffs in Floating-point Divide and Square-root Implementations," *ACM Comput. Surv.*, vol. 28, no. 3, pp. 518–564, Sep. 1996.

[8] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs, 2nd edition*. Oxford University Press, New York, 2010.

[9] B. Pasca, "Correctly rounded floating-point division for dsp-enabled fpgas," in *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on*, aug. 2012, pp. 249 –254.

[10] A. Akkaş, "Dual-Mode Quadruple Precision Floating-Point Adder," *Digital Systems Design, Euromicro Symposium on*, vol. 0, pp. 211–220, 2006.

[11] M. Ozbilen and M. Gok, "A multi-precision floating-point adder," in *Research in Microelectronics and Electronics, 2008. PRIME 2008. Ph.D.*, 2008, pp. 117–120.

[12] M. Jaiswal, R. Cheung, M. Balakrishnan, and K. Paul, "Unified architecture for double/two-parallel single precision floating point adder," *Circuits and Systems II: Express Briefs, IEEE Transactions on*, vol. 61, no. 7, pp. 521–525, July 2014.

[13] A. Akkaş, "Dual-mode floating-point adder architectures," *Journal of Systems Architecture*, vol. 54, no. 12, pp. 1129–1142, Dec. 2008.

[14] M. Jaiswal, B. Varma, H.-H. So, M. Balakrishnan, K. Paul, and R. Cheung, "Configurable architectures for multi-mode floating point adders," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 62, no. 8, pp. 2079–2090, Aug 2015.

[15] A. Baluni, F. Merchant, S. K. Nandy, and S. Balakrishnan, "A fully pipelined modular multiple precision floating point multiplier with vector support," in *Electronic System Design (ISED), 2011 International Symposium on*, 2011, pp. 45–50.

[16] K. Manolopoulos, D. Reisis, and V. Chouliaras, "An efficient multiple precision floating-point multiplier," in *Electronics, Circuits and Systems (ICECS), 2011 18th IEEE International Conference on*, 2011, pp. 153–156.

[17] A. Akkaş and M. J. Schulte, "Dual-mode floating-point multiplier architectures with parallel operations," *Journal of Systems Architecture*, vol. 52, no. 10, pp. 549 – 562, 2006.

[18] A. Isseven and A. Akkaş, "A dual-mode quadruple precision floating-point divider," in *Signals, Systems and Computers, 2006. ACSSC '06. Fortieth Asilomar Conference on*, 2006, pp. 1697–1701.

[19] M. Jaiswal, R. Cheung, M. Balakrishnan, and K. Paul, "Configurable architecture for double/two-parallel single precision floating point division," in *VLSI (ISVLSI), 2014 IEEE Computer Society Annual Symposium on*, July 2014, pp. 332–337.

[20] "IEEE standard for floating-point arithmetic," *IEEE Std 754-2008*, pp. 1–70, Aug 2008.

[21] L. D. McFearin and D. W. Matula, "Generation and analysis of hard to round cases for binary floating point division," in *Computer Arithmetic, 2001. Proceedings. 15th IEEE Symposium on*, 2001, pp. 119–127.

[22] K. S. Hemmert and K. D. Underwood, "Floating-point divider design for FPGAs," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 15, no. 1, pp. 115–118, 2007.

[23] E. Antelo, T. Lang, P. Montuschi, and A. Nannarelli, "Low latency digit-recurrence reciprocal and square-root reciprocal algorithm and architecture," in *17th IEEE Symposium on Computer Arithmetic*, Jun. 2005, pp. 147–154.

[24] P. Malik, "High throughput floating-point dividers implemented in fpga," in *Design and Diagnostics of Electronic Circuits Systems (DDECS), 2015 IEEE 18th International Symposium on*, April 2015, pp. 291–294.

[25] X. Wang and M. Leeser, "Vfloat: A variable precision fixed- and floating-point library for reconfigurable hardware," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 3, no. 3, pp. 16:1–16:34, Sep. 2010. [Online]. Available: http://doi.acm.org/10.1145/1839480.1839486

[26] X. Fang and M. Leeser, "Vendor Agnostic, High Performance, Double Precision Floating Point Division for FPGAs," in *The 17th IEEE High Performance Extreme Computing (HPEC)*, Waltham, MA, 2013.

[27] D. W. Matula, M. T. Panu, and J. Y. Zhang, "Multiplicative division employing independent factors," *IEEE Transactions on Computers*, vol. 64, no. 7, pp. 2012–2019, July 2015.

[28] J. Harrison, *Theorem Proving in Higher Order Logics: 13th International Conference, TPHOLs 2000 Portland, OR, USA, August 14–18, 2000 Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, ch. Formal Verification of IA-64 Division Algorithms, pp. 233–251. [Online]. Available: http://dx.doi.org/10.1007/3-540-44659-1_15

[29] P. Markstein, *IA-64 and Elementary Functions: Speed and Precision*. Hewlett-Packard Professional Books. Prentice-Hall, Englewood Cliffs, NJ, 2000.


**Manish Kumar Jaiswal** (S'12–M'14) is currently a Post-Doctorate Fellow in Dept. of EEE at The University of Hong Kong. He received his B.Sc. and M.Sc. in Electronics from D.D.U. Gorakhpur University in 2002 & 2004 respectively. He obtained his M.S.(By Research) from EE Dept. - I.I.T. Madras in 2009 and Ph.D. (with Outstanding Academic Performance award) from EE Dept. - City University of Hong Kong in 2014. He worked as a Lecturer in the Dept. of Electronics at D.D.U. Gorakhpur University for a year (2005-06), and as a Faculty Member in Dept. of EE at The ICFAI University, Dehradun, India, for two years (2009-11). He also spent 6-months in IBM India Pvt. Ltd. Bangalore in 2008, as a project intern. His research interest includes Digital VLSI Design, Reconfigurable Computing, ASIC/FPGA SoC Design, VLSI Implementation of DSP, Biomedical VLSI and High-Performance Algorithmic Synthesis.

**Hayden K. H. So** (S'03-M'07-SM'15) received the B.S., M.S., and Ph.D. degrees in electrical engineering and computer sciences from University of California, Berkeley, CA in 1998, 2000, and 2007 respectively. He is currently an Assistant Professor of in the Department of Electrical and Electronic Engineering at the University of Hong Kong. He received the Croucher Innovation Award in 2013 for his work in power-efficient high-performance heterogeneous computing system. He was also awarded the University Outstanding Teaching Award (Team) in 2012, as well as the Faculty Best Teacher Award in 2011.