Title	Configurable Architecture for Double/Two-Parallel Single Precision Floating Point Division
Author(s)	Jaiswal, MK; Cheung, RCC; Balakrishnan, M; Paul, K
Citation	Proceedings of 2014 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Tampa, FL, USA, 9-11 July 2014, p. 332-337
Issued Date	2014
URL	http://hdl.handle.net/10722/249248
Rights	IEEE Computer Society Annual Symposium on VLSI. Copyright © IEEE Computer Society.; ©2014 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.; This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.

Configurable Architecture for Double / Two-Parallel Single Precision Floating Point Division

Manish Kumar Jaiswal, Ray C.C. Cheung
Department of EE, City University of Hong Kong, Hong Kong
Email: manish.kj@my.cityu.edu.hk, r.cheung@cityu.edu.hk

M. Balakrishnan and Kolin Paul
Department of CSE, IIT Delhi, India
Email:{mbala, kolin}@cse.iitd.ernet.in

***Abstract*—This paper presents a dynamically configurable and area-efficient multi-precision architecture for Floating Point (FP) division. FP division is a core arithmetic in scientific and engineering domain. We propose an architecture for double precision (DP) division which is also capable of processing dual (two-parallel) single precision (SP) computation, named as DPdSP FP divider. The architecture is based on series expansion methodology of computing division. Key components involved in the floating point division architecture are re-designed in order to efficiently enable the resource sharing and tune the data-path for processing both precision operands with minimum hardware overhead. We have targeted the proposed architecture using "OSUcells Cell Library"** $0.18\mu m$ **technology ASIC implementation. Compared to a standalone double precision divider, the proposed dual mode unified architecture needs** $\approx 7\%$ **extra hardware, with** $\approx 5\%$ **delay overhead. When compared to the previous work in literature, the proposed dual mode architecture out-perform them in terms of required area, throughput, and** $area \times delay$**; has smaller area & delay overhead over only DP divider, and has more computational support.**

***Index Terms*—Floating Point Division, ASIC, Multi-precision Arithmetic, Dynamic Configurable Computing.**

I. Introduction

Floating point division is a core arithmetic needed in a multitude of scientific and engineering computations. The hardware complexity of floating point division arithmetic is more than the other basic arithmetic operations (adder, subtractor and multiplier), and it requires larger area while achieving relatively lower performance. In view of the large area requirement of division arithmetic per unit computation, we aim for a unified & dynamically configurable, multi-precision architecture for this computation.

Researchers have proposed several multi-precision floating point arithmetic architecture designs, which have mainly focused on multipliers ([1], [2], [3]) and adders ([4], [5], [6]). The only multi-precision divider for quadruple and dual double precision operands, based on radix-4 SRT (digit recurrence) division method, has been proposed by Isseven *et. al.* [7]. It is an iterative architecture with a throughput of 29 clock cycles; which can support only normal operands, and sub-normal have been treated as zero. Other division methodology are based on the multiplicative (Newton-Raphson, Goldscmidts) and approximation techniques [8]. All these methods estimate significant trade-offs in required area and delay, and their suitability is based on the operand size, required precision, and implementation platform (software, hardware (FPGA/ASIC)).

This paper has proposed an architecture for division arithmetic which can be dynamically configured to be used for either a double precision operand or two-parallel (dual) single precision operands, called as DPdSP division architecture. Our design is based on the series expansion method (approximation technique) of division ([9], [10], [11]). The proposed DPdSP architecture supports normal as well as sub-normal operands, with round-to-nearest rounding method. A design with only normal support has also been implemented for the purpose of comparison with earlier available method in literature. Further, a DP only design, with same state-of-the-art data path flow, has also been implemented for the area & delay overhead measurements. All the implemented designs take care of corner cases, like infinity, divide-by-zero, zero.

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

- Proposed an architecture for DPdSP division, with both, normal & sub-normal support, with all the exceptional case handling. Major components (like leading-one-detection, dynamic left & right shifting, mantissa computation, rounding, etc) have been optimized/configured with tuned data path, to minimize the resource overhead.
- To the best of our knowledge, this is the only known multiplicative-based, dynamically configurable, on-the-fly multi-precision supported floating point division architecture. The architecture supports the processing of required corner cases.
- Compared to previous literature work, the proposed work has smaller area & delay overhead over only DP divider; has better area, throughput, and $area \times delay$ metric, and has more computational support.

II. Background

Floating point arithmetic implementation involves computing separately the sign, exponent and mantissa part of the operands, and further combining them after rounding and normalization [12], [13]. A basic state-of-the-art flow of the floating point division (including sub-normal processing) is given below in Algorithm 1.

In the present work, we have followed all the steps described in Algorithm 1 for the implementation of the proposed DPdSP division architecture. Each stage of the architecture has been constructed for the support of the dual precision arithmetic.

Algorithm 1 F.P. Division Computational Flow [12], [13]

1: $(IN1\ (Dividend), IN2\ (Divisor))$ Input Operands;
2: **Data Extraction & Exceptional Check-up:**
  $\{S1(\text{Sign1}), E1(\text{Exponent1}), M1(\text{Mantissa1})\} \leftarrow IN1$
  $\{S2, E2, M2\} \leftarrow IN2$
  Check for Infinity, Sub-Normal, Zero, Divide-By-Zero
3: **Process both Mantissa for Sub-Normal:**
  Leading One Detection of both Mantissa
  Dynamic Left Shifting of both Mantissa
4: **Sign, Exponent & Right-Shift Computation:**
  $S \leftarrow S1 \oplus S2$
  $E \leftarrow (E1 - L\_Shift\_E1) - (E2 - L\_Shift\_E2) + BIAS$
  $R\_Shift \leftarrow (E2 - L\_Shift\_E2) - (E1 - L\_Shift\_E1) - BIAS$
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 & Check Exceptional Cases
  Determine Final Output

III. Proposed DPdSP Division Architecture

The proposed floating point division architecture for double precision with dual (two-parallel) single precision support (DPdSP) is shown in Fig. 1. Two 64-bit input operands, may contains either 1-set of double precision or 2-set of single precision operands. All the computational steps in dual mode is discussed below in details.

A. Data Extraction, Sub-normal and Exceptional Handler

In this part, the sign, exponent and mantissa of single or double precision operands have been extracted from the input operands, according to the floating point formats of single and double precision as follows.

$$\textit{Single Precision:}\quad \overbrace{Sign-bit}^{1-bit}\ \overbrace{exponent}^{8-bit}\ \overbrace{mantissa}^{23-bit}$$

$$\textit{Double Precision:}\quad \overbrace{Sign-bit}^{1-bit}\ \overbrace{exponent}^{11-bit}\ \overbrace{mantissa}^{52-bit}$$

The operands are checked for `divide-by-zero`, in which DP `divide-by-zero` signal is obtained by ANDing both SP `divide-by-zero` signal with SP-1 sign-bit. Similar check is done for zero. Then the exponents are checked for sub-normal conditions and updated along with relevant mantissa. Since, 8-bit exponents of DP and second SP overlapped, their sub-normal checks have been shared to save resources. Similarly, the checks for `infinity` and `nan` has been shared among DP and SP. In this part, compared to only double precision, we need extra resources for the checks on first single precision operands.

B. Sub-normal Processing

This section of the architecture, either processes the mantissas of DP or the mantissas for the two SP's. The sub-normal processing of input mantissa, first includes the leading-one-detector (LOD) which detects the position of the leading one, as a left-shift amount for relevant mantissa. Later, it requires a dynamic left shifting of mantissa to bring them in normalized format.

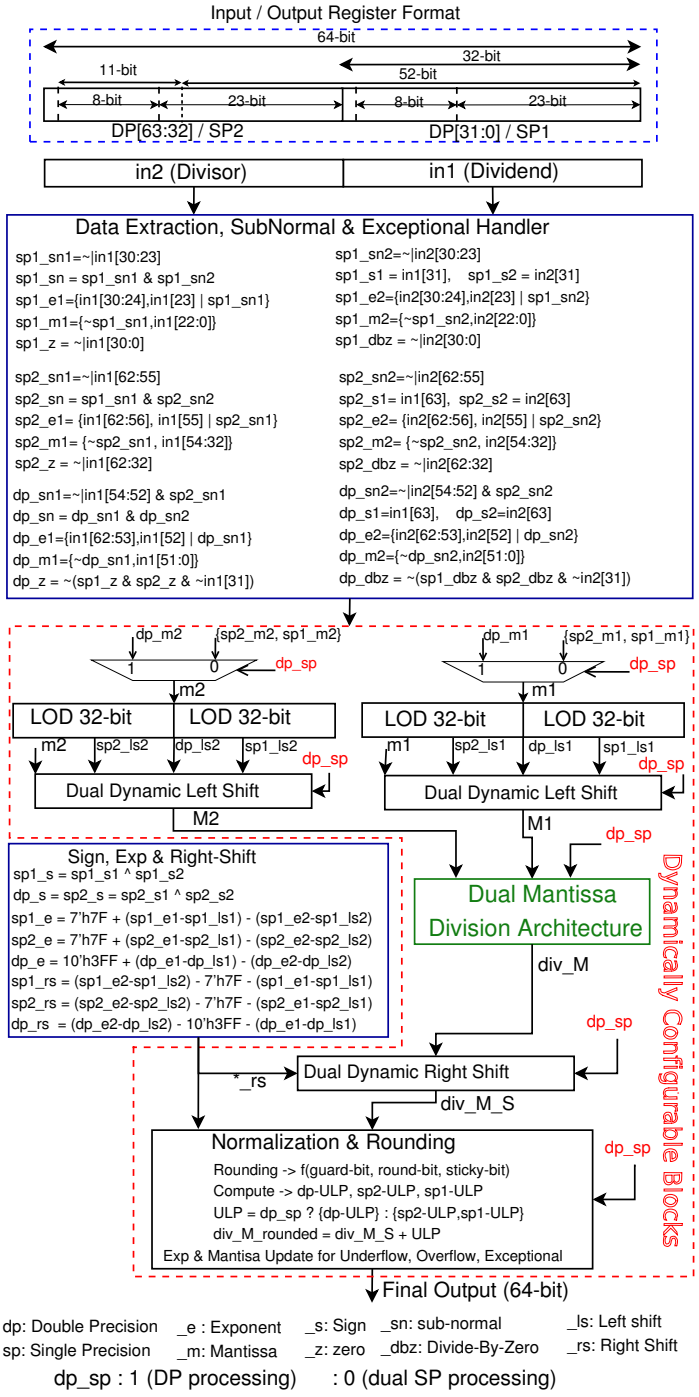

Fig. 1: DPdSP Division Architecture

The architecture of a dual mode leading-one-detector is shown in Fig. 2. The basic building block LOD2:1, consists of three gates which used in a hierarchical manner to get LOD64:6. The output of sub-units of LOD64:6 (the two LOD32:5) are taken as the left shift amount for two SP's, whereas the combined result of them is the left shift amount for the DP mantissa. The resource requirement of dual mode LOD is same as the single mode (for DP only) LOD.

The architecture of dual mode dynamic left shifter is shown

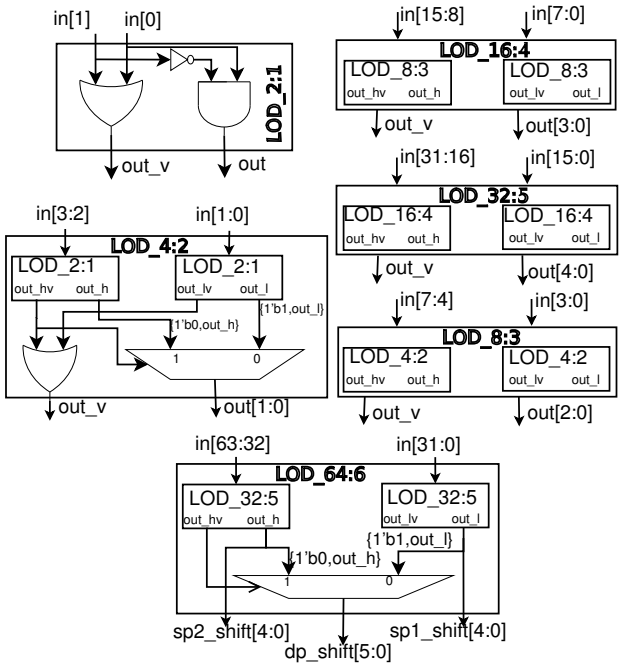

Fig. 2: Dual Mode Leading-One-Detector

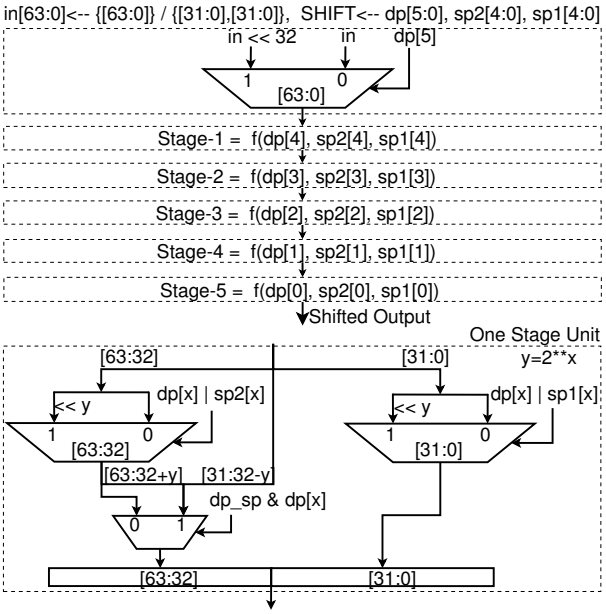

Fig. 3: Dual Mode Dynamic Left Shifter

in Fig. 3. The left shift amount from the previous LOD unit along with relevant mantissa is provided as input to this unit. For true dp_sp the SP's left shift amount is set to zero, and similarly, for false dp_sp the DP left shift amount is forced to zero. In the initial stage, it processes only the MSB of DP left shift (with true dp_sp). All the other stages (Stage-1 to Stage-5) work on the dual shift mode. Each of these stages contains two multiplexers for each 32-bit blocks which shift their inputs based on the corresponding shifting bit (either of double or single precision). Along with this, it contains a multiplexer which can select between lower shifting output or their combination with primary input to the stage, based on the true dp_sp and corresponding shifting bit of DP. Except this multiplexer, the architecture behaves like two 32-bit barrel shifter, which have been constructed to support dual mode shifting operation. The proposed dual mode dynamic left shifter has a minor area & delay overhead than a single mode 64-bit dynamic right shifter.

C. Sign, Exponent and Right Shift Computation

The sign and exponent computation are trivial and simpler. In case of underflow-ed (negatively computed) exponent, the right shift amount need to be computed for the right shifting of quotient mantissa. All the relevant computation of this section have been done separately for DP and both SP operands, as shown in Fig. 1.

D. Mantissa Computation

The mantissa computation is the most critical part in division architecture. This has been implemented using series expansion method as follows.

Let x represent the dividend mantissa, y represent divisor mantissa and q be the mantissa quotient, which can be computed as follows,

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

where, the divisor mantissa has been partitioned to a_1 and a_2, which can be written as follows,

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

Here, the precomputed value of a_1^{-1} is used to perform the remaining computation. Based on the bit width ($m1$) of a_1, the size of memory (to store a_1^{-1}) and the number of terms from the series expansion can be decided. For a balanced case, the value $m1 = 8 - bit$ is a preferred choice for double precision computation. With $m1 = 8 - bit$, it require seven terms (up to $a_1^{-7}.a_2^6$) for double precision and three terms (up to $a_1^{-3}.a_2^2$) for single precision computation. The quotient expression can be written as follows.

For double precision it will be,

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

For single precision it will be,

$$\begin{aligned} q &= x \times [a_1^{-1} - a_1^{-2}.a_2 + a_1^{-3}.a_2^2] \\ &= x.a_1^{-1} - x.a_1^{-1}(a_1^{-1}.a_2 - a_1^{-2}.a_2^2) \end{aligned} \tag{4}$$

In both the eqs.(3 and 4), simplification has been done to achieve the maximum overlapping of the terms. Here, eq.(4) can be fully superimposed on eq.(3), and both can share the

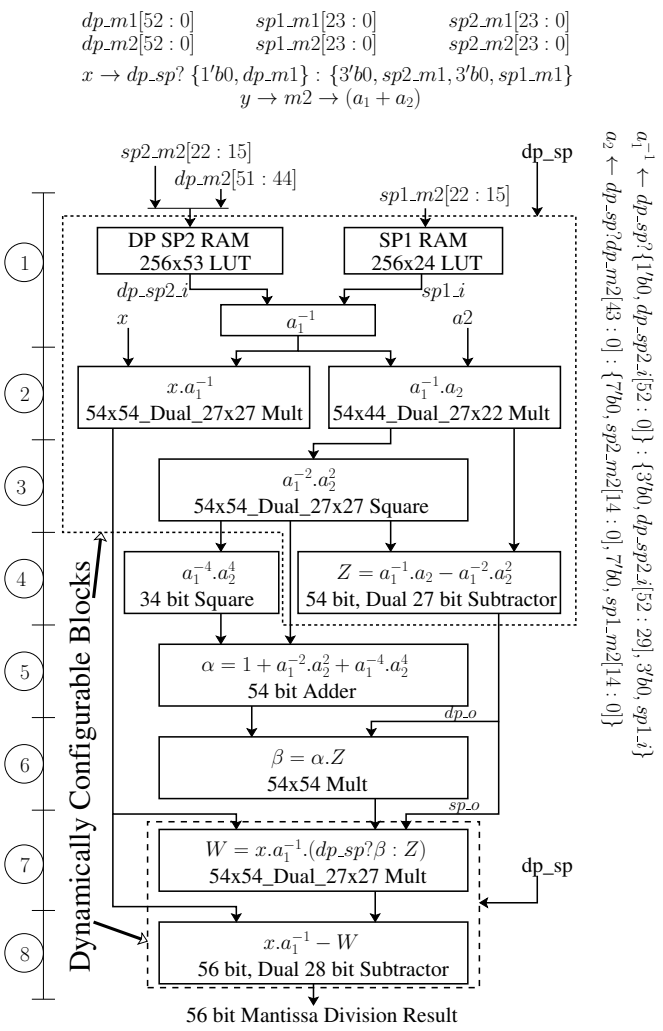

Fig. 4: DPdSP Dual Mode Mantissa Division Architecture

same computational flow. This overlapping leads us to model efficiently the architecture for mantissa quotient computation, which can compute either a DP mantissa or two SP mantissa.

The DPdSP mantissa division architecture is shown in Fig. 4. This proposed dual mode mantissa division architecture computes either of the eq.(3) and eq.(4) on the same data path with efficient resource sharing and minimal overhead. In this architecture, except the computation of $a_1^{-4}.a_2^4$, α and β (in steps 4, 5 & 6) the entire components are shared and dynamically configurable for DP as well as both the SP's computations. The $a_1^{-4}.a_2^4$, α and β blocks takes part in DP computation only. The computational flow is as follows.

Initially, in step-1 the precomputed value of a_1^{-1} has been obtained from look-up-table (LUTs). This architecture used two LUTs; one targeted either for DP or SP-2 operand (of size 258x53), and other for SP-1 operand (of size 256x24). First LUT has been shared for DP and SP-2, based on the respective computation. The a_1^{-1} value is determined using a multiplexer (mux), either for DP or for both SP's.

In step-2, the computation of $x.a_1^{-1}$ and $a_1^{-1}.a_2$ is performed. For $x.a_1^{-1}$, we have used a dual mode multiplier (54x54_Dual_27x27 Multiplier, Fig. 5), which can perform either a 54x54 multiplication (for DP) or two 27x27 multiplication (for both SP). Here, the input operands size is 54-bits, which either contains DP data or both of SP's data. The multiplication has been performed using Karatsuba method [14]. Using Karatsuba method, the multiplication logic reduced by 25% incurring an additional cost of some adders and q subtractor. This helps in reducing the area. Computation of $x.a_1^{-1}$ has been performed as in eq.(5). For a 54x54 multiplication, it requires two 27x27 multipliers, one 28x28 multiplier, two 27-bit adders, one 56-bit subtractor and one 91-bit adder. This dual mode multiplication does not need any extra hardware cost over only DP, except for an additional mux.

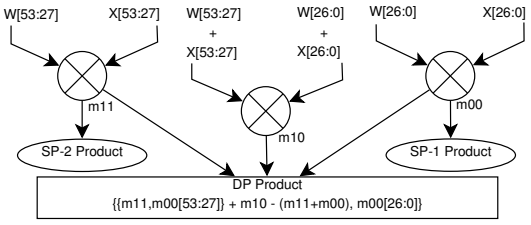

Fig. 5: 54x54_Dual_27x27 Multiplier

$$
\begin{aligned}
&sp1\_o[53:0] = x[26:0] * a_1^{-1}[26:0] \\
&sp2\_o[53:0] = x[53:27] * a_1^{-1}[53:27] \\
&tmp[55:0] = ((a_1^{-1}[53:27] + a_1^{-1}[26:0]) \\
&\quad * (x[53:27] + x[26:0])) - (sp2\_o + sp1\_o) \\
&dp\_o[107:0] = \{sp2\_o, sp1\_o\} + \{tmp, 27'h0\} \\
&x.a_1^{-1} = dp\_sp\ ?\ dp\_o : \{sp2\_o, sp1\_o\}
\end{aligned} \tag{5}
$$

Similarly, the $a_1^{-1}.a_2$ has been computed using a 54x44_Dual_27x22 Mult, which performs either a 54x44 multiplication (for DP) or two 27x22 multiplication (for both SP's), using the Karatsuba method, as in eq.(6). Here, a_2 is 44-bit wide and contains data as shown in Fig. 4.

$$
\begin{aligned}
&sp1\_o[48:0] = a_1^{-1}[26:0] * a_2[21:0] \\
&sp2\_o[48:0] = a_1^{-1}[53:27] * a_2[43:22] \\
&tmp[55:0] = ((\{a_1^{-1}[53:27], 5'b0\} + a_1^{-1}[26:0]) \\
&\quad * (a_2[43:22] + a_2[21:0])) - (\{sp2\_o, 5'b0\} + sp1\_o) \\
&dp\_o[97:0] = \{sp2\_o, sp1\_o\} + \{tmp, 22'h0\} \\
&a_1^{-1}.a_2 = dp\_sp\ ?\ dp\_o : \{sp2\_o, sp1\_o\}
\end{aligned} \tag{6}
$$

Step-3 computes $a_1^{-2}.a_2^2$ with a dual mode square, 54x54_Dual_27x27 Square. This has been done on block basis, with block size of 27-bit, needs three 27x27 multiplier and one 90-bit adder as follows:

$$
\begin{aligned}
&operand \leftarrow i[53:0], \qquad sp1\_o = i[26:0] * i[26:0] \\
&sp2\_o = i[53:27] * i[53:27], \quad tmp = i[53:27] * i[26:0] \\
&dp\_o = \{sp2\_o, sp1\_o\} + \{tmp, 28'h0\} \\
&a_1^{-2}.a_2^2 = dp\_sp\ ?\ dp\_o : \{sp2\_o, sp1\_o\}
\end{aligned} \tag{7}
$$

Step-4 needs a 34-bit square to compute $a_1^{-4}.a_2^4$. This needed only for DP flow and is implemented using block basis (with block size of 17-bit) similar to eq.(7). Since the contribution of this term in DP result falls after 34-bit precision, a smaller multiplier is required. This step also computes $a_1^{-1}.a2 - a_1^{-2}.a_2^2$, using two 27-bit subtractors (for both SP's), combined to form a 54-bit subtractor (for DP).

Step-5 and step-6 performs computation related to DP only. Step-5 computes the last sum $(1 + a_1^{-2}.a_2^2 + a_1^{-4}.a_2^4)$ term of eq.(3), using a 54-bit adder. Whereas, step-6 computes the multiplication of previous sum (α) with DP output components of Z (in step-4), using a 54x54 multiplier using Karatsuba method.

Step-7 computes the dual multiplication using a 54x54_Dual_27x27 multiplier (as in Fig. 5). It multiplies $x.a_1^{-1}$ with either β of step-6 or single precision outputs of Z (in step-4), to generate W $(x.a_1^{-1}.\{(a_1^{-1}.a_2 - a_1^{-2}.a_2^2) \times (1 + a_1^{-2}.a_2^2 + a_1^{-4}.a_2^4)\}$ for DP or $x.a_1^{-1}.(a_1^{-1}.a_2 - a_1^{-2}.a_2^2)$ for both SP's). Finally, the W has been subtracted by $x.a_1^{-1}$ in step-8 using two 28-bit subtractors (for both SP's), collectively performs 56-bit subtraction for DP mantissa quotient.

Thus, the proposed dual mode mantissa division architecture performs either a double precision division or two single precision division, with extra costs of a $2^8 \times 24$ LUT for SP-1, and multiplexers in each dual mode step.

E. Dynamic Right Shifting

The architecture of dual mode dynamic right shifter is shown in Fig. 6. The input to this unit are mantissa quotient, and computed right shift amount. The underlying concept of it's architecture is similar to the dual mode dynamic left shifter. In comparison to left shifter, the additional multiplexer in stage-1 to stage-5 is used to process the lower right shift output or its combination with primary input of the stage.

F. Normalization, Rounding and Final Processing

This section processes the previously computed exponents and dual mantissa division result to obtain rounded normalized format. The output of dual mode mantissa division either consists of DP mantissa division quotient or consists of both SP mantissa division quotients in each of its 32-bit parts. Based on the MSBs of the quotient result, the rounding position is determined. Further, based on the rounding position bit, Guard-bit, Round-bit, Sticky-bit and MSB-bit, the round ULP (unit at last place) has been computed. This ULP computation has been performed separately for DP and both SP and requires few gates for each. These round ULP has later been added to mantissa quotient using two 28-bit adders, individually works for SP's computations, and collectively produce the output for DP. This rounded mantissa sum has been normalized. The rounding adder in effect is similar to that required for only DP processing.

Furthermore, the exponents have been updated accordingly. Then each exponent and mantissa is updated for one of infinity, sub-normal or underflow cases, and each require separate units.

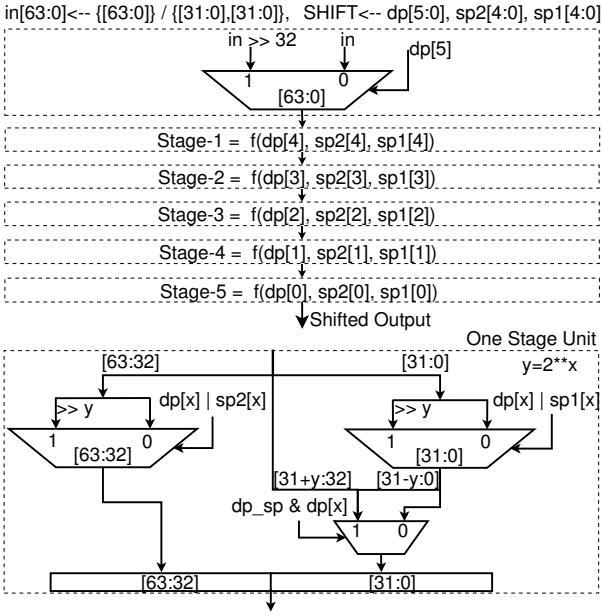

Fig. 6: Dual Mode Dynamic Right Shifter

TABLE I: Resource Overhead in DPdSP over DP only

DPdSP Sub-Components	Extra resource over DP only
Data Extraction, Sub-normal & Exceptional Handler	For one SP computations
LOD	Nil
Dynamic Left/Right Shift	$\approx$ Nil
Mantissa Division	a 256x24 LUT, and 7- 54-bit 2:1 MUXs
Normalization & Rounding	ULP-computation of both SP's
Final Processing	Processing of both SP's & a 64-bit 2:1 MUX

The computed signs, exponents and mantissas of double precision and both single precision have been finally multiplexed to produce the final 64-bit output, which either contains a DP output or two SP outputs. A brief summary of extra resource overhead of proposed DPdSP division architecture over only a DP division is shown in Table-I.

IV. Implementation Results

This section presents the implementation details of the proposed DPdSP divider architecture along with only DP divider implementation. The proposed DPdSP architecture has been synthesized with "OSUcells Cell [15]" 0.18μm technology, using Synopsys Design Compiler. The proposed architecture is currently aimed towards the single cycle design. The proposed DPdSP divider architecture has been synthesized with only normal support and with sub-normal support. A DP only division design, with same state-of-the-art data path flow, has also been synthesized (with both, only normal and with sub-normal) for area & delay overhead computation purpose. The implementation details has been shown in Table-II. The DPdSP design with normal support requires roughly 150K gate (based on minimum inverter size) and $\approx$ 163K gates when included with sub-normal support. With only normal support,

TABLE II: ASIC Implementation Details

	Only Normal		With Sub-normal	
	DPdSP	DP	DPdSP	DP
Area(μm^2)	2395728	2297697	2611098	2441050
Gate Count	149733	143606	163194	152566
Delay (ns)	35.7	34	39.38	38
Delay (FO4)	396.6	377.7	437.5	422.2

TABLE III: Comparison of DPdSP Division Architecture

	[7] (Only Normal)	Proposed (Only Normal)	Proposed (With Sub-normal)
Area OH[1]	22%	4.3%	6.9%
Period OH[1]	8%	5%	3.6%
Gate Count[2]	212854	149733	163194
Period (FO4)[3]	31.4	396.6	437.5
Throughput[4]	29/15 (DP/dSP)	1/1 (DP/dSP)	1/1 (DP/dSP)
Throughput[5]	909.4	396.6	437.5
Area × Delay [6]	1.93×10^8	0.59×10^8	0.71×10^8

[1]Area/Period OH = (DPdSP - DP) / DP, [2]Based on minimum size inverter
[3]1 FO4 (ns) ≈ (Tech. in μm) / 2 [4] Throughput (in cycle)
[5]Throughput (in FO4) = Throughput (in cycle) * Period (FO4)
[6]Gate Count × Throughput (FO4)

the proposed DPdSP division architecture requires ≈ 4.3% more hardware and ≈ 5% more delay than only DP division design, and it needs ≈ 6.9% more resources and ≈ 3.6% more delay when included with sub-normal support.

A comparison with previously reported dual-mode (double precision with two-parallel single precision support) division design in literature has been shown in Table III. To the best of our knowledge there is no prior work on the related title using multiplicative methods of division. As reported earlier, Isseven *et. al.* has proposed an iterative dual-mode division architecture using radix-4 SRT division algorithm, a digit-recurrence method. Their proposed design has a throughput of 29 clock cycle for double precision, and 15 clock cycle for single precision. Further, it has been proposed for only normal support and without sub-normal support. They have synthesized their proposal using the TSMC 0.25μm technology. Table III has shown a comparison of the proposed work with their work. We have made a technology independent comparison, where, the area has been compared in terms gate count (based on minimum size inverter) and delay/throughput has been compared in terms of FO4 (Fan Out of 4) delay. Comparatively, Isseven *et. al.*'s dual-mode architecture needs a larger area than the proposed DPdSP architecture. The throughput for double precision processing is much better for the proposed design. For single precision, the throughput of Isseven *et. al.* is 15 clock cycle, an equivalently 471 FO4, is also larger than the proposed work. The $area \times delay$ of the proposed architecture is much better than the previous work. The proposed architecture can be easily pipelined to have a even better throughput. Also, the area & delay overhead, over only DP divider, of proposed work is smaller than the previous work in Isseven *et. al.*. Also, in addition to better design metrics, the proposed work also support the processing of sub-normal operands.

V. Conclusions

This paper has presented an architecture for floating point division with on-the-fly dual precision support. It supports configurable, double precision with dual (two-parallel) single precision (DPdSP) floating point division computation. It support normal and sub-normal operands processing. The data path of the architecture has been tuned to perform the dual mode computation with minimal hardware overhead. The crucial module of mantissa division has been tuned with other components (dual mode LOD, dual mode dynamic left/right shifter, rounding) for on-the-fly dual mode computation. It has $\approx 4.3\% - 6.9\%$ area and $\approx 5\%$ delay overhead over DP only module. The proposed division architecture is the only known multiplicative based, dynamically configurable, on-the-fly multi-precision supported design. Compared to previous literature, this work out-perform them in terms of required area, throughput, and $area \times delay$; has smaller area & delay overhead over only DP divider, with more computational support. Future work will focus on configurable architecture for other division methods (Newton-Raphson, Goldscmidts).

Acknowledgment

This work was supported by a grant from Croucher Startup Allowance (Project No. 9500015).

References

[1] 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.
[2] K. Manolopoulos, D. Reisis, and V. Chouliaras, "An efficient multiple precision floating-point multiplier," in *Electronics, Circuits and Systems, 2011 18th IEEE International Conference on*, 2011, pp. 153–156.
[3] A. AkkaÅ§ 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.
[4] A. Akkas, "Dual-Mode Quadruple Precision Floating-Point Adder," *Digital Systems Design, Euromicro Symposium on*, pp. 211–220, 2006.
[5] ——, "Dual-mode floating-point adder architectures," *Journal of Systems Architecture*, vol. 54, no. 12, pp. 1129–1142, Dec. 2008.
[6] 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.
[7] A. Isseven and A. Akkas, "A dual-mode quadruple precision floating-point divider," in *Signals, Systems and Computers, 2006. ACSSC '06. Fortieth Asilomar Conference on*, 2006, pp. 1697–1701.
[8] S. F. Obermann and M. J. Flynn, "Division algorithms and implementations," *Computers, IEEE Transactions on*, vol. 46, no. 8, pp. 833–854, Aug. 1997.
[9] 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.
[10] 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.
[11] 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.
[12] "IEEE Standard for Binary Floating-Point Arithmetic," *ANSI/IEEE Std 754-1985*, 1985.
[13] "IEEE Standard for Floating-Point Arithmetic," Tech. Rep., Aug. 2008.
[14] A. Karatsuba and Y. Ofman, "Multiplication of Many-Digital Numbers by Automatic Computers," in *Proceedings of the USSR Academy of Sciences*, vol. 145, 1962, pp. 293–294.
[15] Oklahoma State University, OSUCells, http://vlsiarch.ecen.okstate.edu.