Unified Architecture for Double / Two-Parallel Single Precision Floating Point Adder

Manish Kumar Jaiswal, Ray C.C. Cheung, M. Balakrishnan and Kolin Paul

***Abstract*—Floating point (F.P.) addition is a core operation for a wide range of applications. This paper presents an area-efficient, dynamically configurable, multi-precision architecture for F.P. addition. We propose an architecture of double precision (DP) adder, which also support dual (two parallel) single precision (SP) computational feature. Key components involved in the F.P. adder architecture, such as comparator, swap, dynamic shifters, leading one-detector (LOD), mantissa adders/subtractors and rounding circuit, have been re-designed in order to efficiently enable resource sharing for both precision operands with minimal multiplexing circuitry. The proposed design supports both normal and sub-normal numbers. The proposed architecture has been synthesized for OSUcells Cell $0.18\mu m$ technology ASIC implementation. Compared to a standalone DP adder with two SP adders, the proposed unified architecture can reduce the hardware resources by $\approx 35\%$, with a minor delay overhead. Compared to previous works, the proposed dual mode architecture has 40% smaller $area \times delay$, and has better area & delay overhead over only DP adder.**

***Index Terms*—Floating Point Addition, Multi-precision Arithmetic, ASIC, Digital Arithmetic.**

I. Introduction

Floating point (F.P.) addition is a core arithmetic operation in a multitude of scientific and engineering computations. Over the past few decades, considerable work have been done to improve the architecture of floating point arithmetic [1], [2]. In view of the large area requirement of F.P. arithmetic per unit computation, we aim for a unified multi-precision architecture.

In literature, some authors have focused on multi-precision floating point arithmetic architecture design. Many of these are focused on multi-precision F.P. multiplier architecture design [3], [4], which supports only normalized numbers. Isseven *et. al.* [5] has presented a multi-precision divider for quadruple and dual double precision operands. A. Akkas [6] has shown multi-precision architectures for F.P. addition, which has been further extended in [7] with single path and two path design. However both of them have designed to support only normalized numbers. The computation related to sub-normal numbers and exceptional case handling were left for software processing. Further, Ozbilen *et. al.* [8] has presented an adder architecture for double precision with dual single precision support, targeted for normalized operands.

This work was supported in part by Croucher Startup Allowance.

Manish Kumar Jaiswal and Ray C.C. Cheung are with Department of EE, City University of Hong Kong, Hong Kong (e-mail: manish.kj@my.cityu.edu.hk; r.cheung@cityu.edu.hk)

M. Balakrishnan and Kolin Paul are with Department of CSE, Indian Institute of Technology, Delhi, India. e-mail: {mbala,kolin}@cse.iitd.ernet.in

In this paper, we have developed an architecture for addition arithmetic with double precision F.P. numbers which can also support on-the-fly dual (two parallel) single precision F.P. numbers computations, named as DPdSP adder architecture. We have designed and/or configured the key elements of F.P. adder, in order to share them among different precision operands, to support the multi-precision computation. The proposed architecture fully supports normal as well as sub-normal computations, with round-to-nearest rounding method. Other rounding methods can be easily included. We have compared our results with the best optimized implementations available in the literature. The main contributions of this work can be summarized as follows:

- Proposed an architecture for DPdSP adder, which can perform on-the-fly either a Double Precision or dual (two parallel) Single Precision addition/subtraction.
- Components have been optimized/configured with tuned data path, to minimize the multiplexing circuitry, for reducing the area and delay metrics. It can be easily extend for any dual precision F.P. adder implementation.
- Compared to previous works, the proposed work provides more computational support, and has smaller area overhead over only DP design with similar or smaller delay overhead.

II. Proposed DPdSP Adder Architecture

A basic state-of-the-art flow [9] of the floating point addition is given below in Algo 1. In the present work of DPdSP adder architecture, we have followed the complete steps described in Algo. 1 and constructed them for the support of the dual mode operation.

The proposed architecture of double precision with dual single precision support (DPdSP) floating point adder is shown in Fig. 1. Two 64-bit input operands, may contain either 1-set of double precision or 2-sets of single precision operands. First operand contains either first input of DP or first inputs of both SPs, and second operand contains second inputs of either of DP or both SPs. Based on the signal dp_sp, it can be dynamically switched to either double precision or dual single precision mode (dp_sp: $1 \rightarrow$ DP Mode, dp_sp: $0 \rightarrow$ Dual SP Mode). All the computational steps in dual mode is discussed below in detail. The explanation has been in relation to the state-of-the-art flow discussed in Algo. 1.

A. Data Extraction and Subnormal Handler

Computation of this sub-component is shown in Fig. 2. In this part, the sign, exponent and mantissa of single or

Algorithm 1 F.P. Adder Computational Flow [9]

(*IN*1,*IN*2) Input Operands;
Data Extraction & Exceptional Check-up:
{S1(Sign1), E1(Exponent1), M1(Mantissa1)} ← *IN*1
{S2, E2, M2} ← *IN*2
Check for INFINITY, SUB-NORMALs, NAN
Update hidden bit of Mantissa's for SUB-NORMALs
COMPARE, SWAP & Dynamic Right SHIFT:
*IN*1_*gt*_*IN*2 ← {*E*1,*M*1} ≥ {*E*2,*M*2}
Large_E,M ← *IN*1_*gt*_*IN*2 ? E1,M1 : E2,M2
Small_E,M ← *IN*1_*gt*_*IN*2 ? E2,M2 : E1,M1
Right_Shift ← Large_E - Small_E
Small_M ← Small_M >> Right_Shift
Mantissa Computation:
$OP \leftarrow \overline{S1 \oplus S2}$
if *OP* == 1 **then**
    Add_M ← Large_M + Small_M
else
    Add_M ← Large_M - Small_M
Leading-One-Detection & Dynamic Left SHIFT:
Left_Shift ← LOD(Add_M)
Left_Shift ← Adjustment for SUB-NORMAL or Underflow or No-Shift(True Add_M MSBs)
Add_M ← Add_M << Left_Shift
Normalization & Rounding:
Mantissa Normalization & Compute Rounding ULP based on Guard, Round & Sticky Bit
Add_M ← Add_M + ULP
Large_E ← Large_E + Add_M[MSB] - Left_Shift
Finalizing Output:
Update Exponent & Mantissa for Exceptional Cases
Determine Final Output

double precision operands have been extracted from the input operands, according to the floating point formats of single and double precision [9].

The exponents have been checked for sub-normal condition by NOR of their bits. Since, 8-bit exponents of double precision and second single precision overlapped, their sub-normal check have been shared to save some resources. Further, all the exponents and mantissas have been updated according to the result of sub-normal checks'. In this part, compared to only double precision, we need extra resources for sub-normal check and update for first single precision operands. Similar to sub-normal checks, the checks for infinity and nan has been shared among DP and SP

B. Comparator

This component has been shared among operands, with the resource used only for double precision operands. Related computation of this sub-unit is shown in Fig. 2. A 31-bit comparator (for "greater than") is used for first single precision operands comparison. Another, 31-bit "greater than" comparator is used for second single precision operands. By combining these two outputs with a 31-bit "equal to" and 1-bit "greater than equal to" comparator, the double precision comparison has been established. Even for only double precision, we require the same/similar components for comparator, which have been configured to support DP with dual SP processing.

C. SWAP: Large Sign, Exponent, Mantissa & OPERATION

The underlying computation of this sub-component is shown in Fig. 2. This section of the architecture determines the effective operation between large and small mantissa (addition/subtraction). This also produces the large sign (in effect output sign bits), small & large exponents, and small & large mantissa. For SWAP, in general, we need four 8-bit (for both SP exponents), two 10-bit (for DP exponent), four 24-bit (for both SP mantissas) and two 53-bit (for DP mantissa) SWAP components for all the computations of this section. However, by multiplexing either of the double precision or both single precision operands, we need only four 8-bit (for exponents) and four 32-bit (for mantissas) SWAP circuitry for entire processing. Effectively, it needs SWAP components slightly more than we require for only DP, along with extra multiplexing circuitry. Among extra appended LSB ZEROs in mantissa multiplexing (for $m1$ and $m2$), 3-bit are for Guard, Round and Sticky bit computations in Rounding phase, and remaining can provide extended precision support to the operands. The output *m_L* contains mantissa of either large DP operand or both of large SP operands. Similarly, *m_S* contains small mantissas. Likewise, *e_L* contains large exponent, and *e_S* contains small exponents, either of DP or both SP operands.

D. Right Shift Amount

The right shift amount has been determined by the difference between large and small exponents, generated in SWAP unit. In general, it requires two 8-bit subtractors for single precision and one 11-bit subtractor for double precision. However, because of effective multiplexing of operands in SWAP section, we need only one 16-bit subtractor for this. It will produce either shift amount for double precision or for both single precision. For shift amount, compared to only double precision, it requires extra resources for 5-bit subtraction. Other processing in this section are bit-wise operations, and are done separately for all operands. Computation of this subcomponent is shown in Fig. 2.

E. Dynamic Right Shifter

This component in the adder architecture is for the right shifting of small mantissa which is used to align the decimal points of mantissas. The architecture of the dual mode dynamic right shifter is shown in Fig. 3a. The input to this unit is small mantissas *m_S* from SWAP unit, and right shifted amount dp_r_shift, $sp2\_r\_shift$ and $sp1\_r\_shift$. When dp_sp is true, the $sp2\_r\_shift = sp1\_r\_shift = 0$, and when it is false, $dp\_r\_shift = 0$. When MSB of dp_r_shift is true (in case $dp\_sp = 1$) the 64-bit input is right shifted by 32-bit. All the other stages (Stage-1 to Stage-5) work on the dual shift mode, and are parameterized for their inputs. The proposed dynamic right shifter can be easily extended for any size dual mode dynamic shifting. 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_r_shift. Except this multiplexer, the architecture behaves like two 32-bit barrel shifter, which have been constructed to support dual

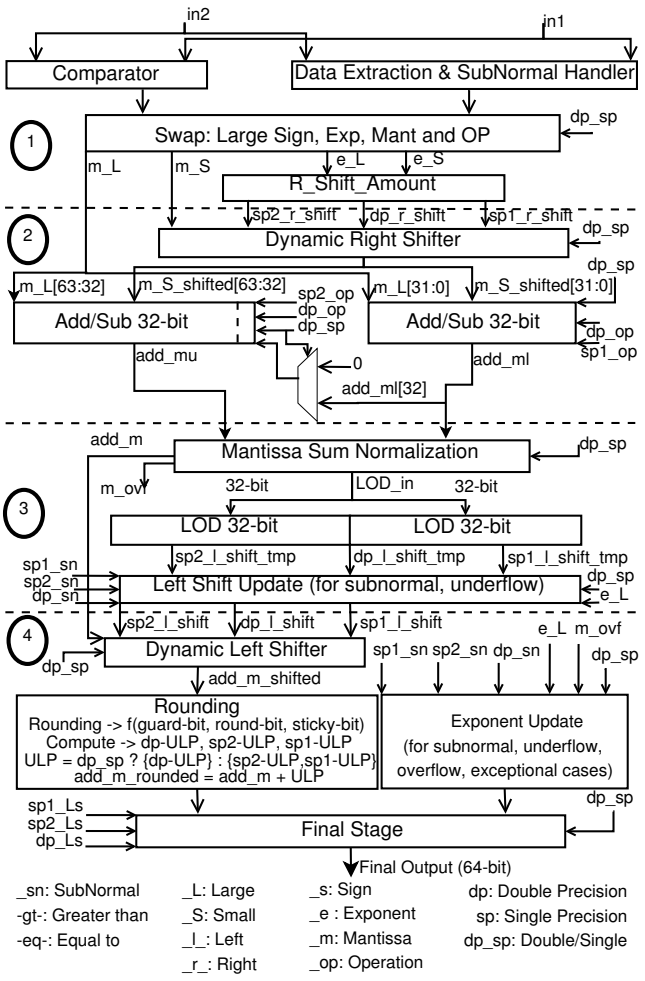


Fig. 1: DPdSP Adder Architecture (with 4-Stage Pipeline)

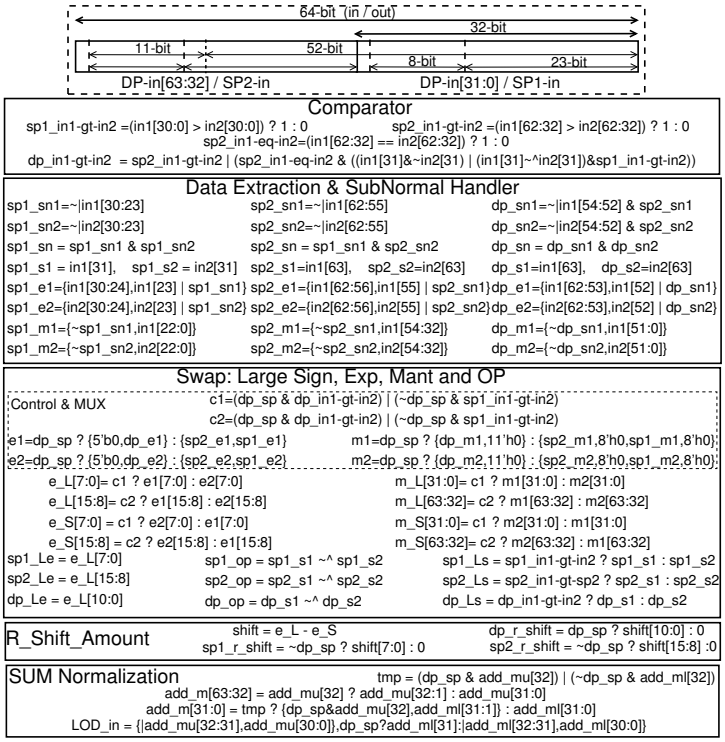


Fig. 2: DPdSP Adder: Sub-Components

mode shifting operation. The proposed dual mode dynamic right shifter takes approximately similar area with minor delay overhead than a single mode 64-bit dynamic right shifter.

F. Mantissa Addition & SUM Normalization

This unit uses two 32-bit adder-subtraction (add-sub) units. The input operands to this section are *m_L* from SWAP unit and right shifted small mantissa from dynamic right shifter. For true dp_sp, both adders perform together for DP processing, and for false dp_sp, they perform individually for both SP's. In effect, this unit requires the resources similar to only double precision processing. The "Mantissa SUM Normalization" unit combine the previous two 32-bit sum operation, to generate actual sum (either for DP or SP's), mantissa overflow bits, and the inputs for LOD. generally, for only DP it requires a 1-bit shifter of 64-bits, whereas, in dual mode it has been accomplished by two 1-bit shifters of 31-bits each, along with some gates for small control logics, as shown in Fig. 2.

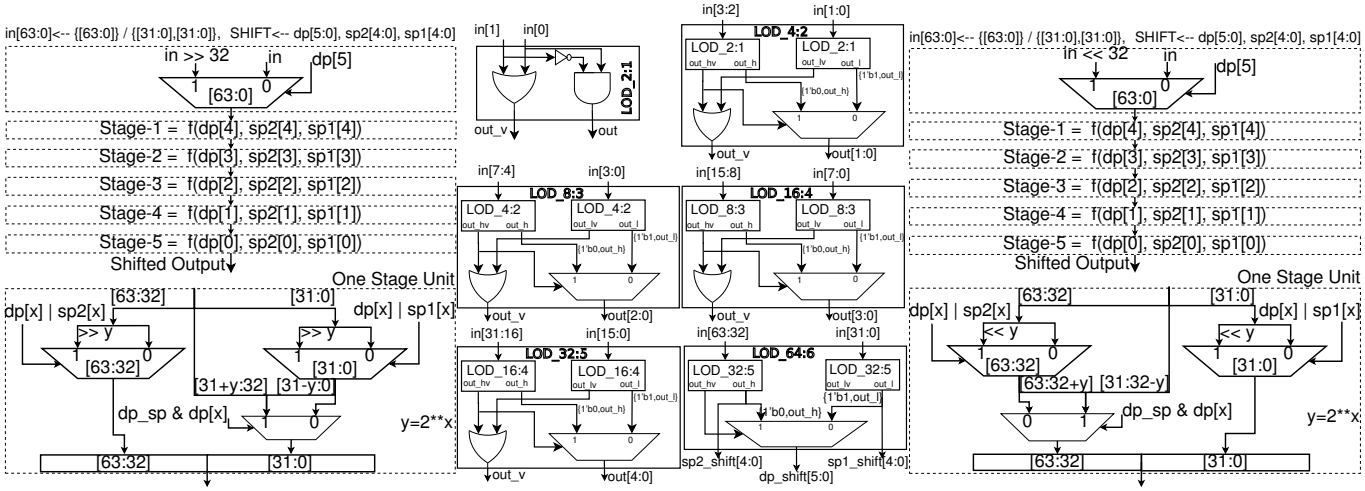
G. Leading-One-Detector (LOD) and Left Shift Update

This section of the architecture is meant to detect leading one in the *add_m*, in case it has lost its MSB, in order to bring it in to normalized format after left shifting. It happens when two mantissa, values are very close to each other, undergoes subtraction operation. Input to the LOD would be either a DP operand or two SP operands, generated from mantissa sum normalizer unit. LOD unit architecture is shown in Fig. 3b. It consists of two 32-bit LOD, each of which processes 32-bit input on *LOD_in*. Their individual outputs acts as SP left shift amount, which after combining act as DP left shift amount. The architecture of 32-bit LOD is based on a easy hierarchical flow shown in Fig. 3b. Based on the above approach, this can be easily extend to larger size LOD. The LOD, in effect, uses same resources as used for only DP processing.

Further to quantify the left shift amount, it has been updated for sub-normal cases (both sub-normal input operands), underflow cases (if the left shift amount exceeds or equals to the corresponding large exponent), and no-shift case (*add_m* MSBs are true, either for DP or SPs). For both sub-normal input operands case or the no-shift case, the corresponding left shift is forced to zero, and for the underflow case, the corresponding left shift is equal to corresponding large exponent decremented by one. Further, for true dp_sp, the SP left shifts are forced to zero, and for false dp_sp the DP left shift is forced to zero. In the left shift update section, the exponent decremented part of DP and first SP has been shared. This becomes possible because the required LSBs of *e_L* has been shared among them. All other computation, related to left shift update need to be computed separately for DP and both SP.

H. Dynamic Left Shifter

The architecture of dual mode dynamic left shifter is shown in Fig. 3c. The input to this unit are mantissa addition *add_m*,



(a) Dual Mode Dynamic Right Shifter (b) Dual Mode Leading-One-Detector (c) Dual Mode Dynamic Left Shifter

Fig. 3: DPdSP Adder : Dynamic Left/Right Shifter, LOD

TABLE I: Resource Sharing in DPdSP Adder Sub-Components

DPdSP Architectural Components	Shared Resources	Extra resource over Only DP
Data Extraction & Subnormal Handler	"Subnormal, infinity, & NAN" checks of DP and one SP	For one SP
Comparator, Dynamic Shifters, LOD	Shared DP and both SP	Nil
SWAP: Large Sign, Exp, Mant & OP	Shared SWAP of DP and both SP	Two 80-bit MUX, Control Logic
Right Shift Amount	Subtraction for DP & both SP	5-bit sub, some bit-wise ANDing
Mantissa Add/Sub, Sum Normalization	Shared DP and both SP	two 2:1 MUX & small control logic
Left Shift Update	Exponent difference of DP and one SP	Remaining computation of both SP
Rounding	ULP addition shared among DP and both SP	ULP-computation of both SP
Exponent Update	Shared the update of DP and one SP	some bit-wise AND operations
Final Processing	Post Round Update of Exponent & Mantissa	Remaining processing of both SP, one 64-bit MUX to discharge final output

and updated left shift amount from previous stage. The basic idea of this is similar to dual mode dynamic right shifter. It contains two left shifters to process each of the 32-bit inputs in this stage. In comparison to right shifter, the additional multiplexer is used to process the higher left shift output or its combination with primary input of the stage. Furthermore, this is also parameterized and can be easily extended for any amount of dual mode dynamic left shifting.

I. Exponent Update

In this unit the exponents have been updated for mantissa overflow and mantissa underflow. The exponents need to be incremented by one or decremented by left shift amount. This update has been shared for DP and one SP, by sharing a subtractor. This need an extra 5-bit adder for a SP processing as an overhead over DP processing.

J. Rounding and Final Processing

The primary operands for this section is the left shifted mantissa from previous dual mode dynamic left shifter. The $add\_m\_shifted$ consists either of DP or both SP in each of its 32-bit parts. Based on the MSBs of the $add\_m\_shifted$, the rounding position need to be determined. Right next bit to the rounding position is the Guard-bit, the next right is Round-Bit, and remaining right bits generate Sticky-bit. 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 need to perform separately for DP and both SP and requires few gates for each. Approximately we need thrice of DP only computation. After generating the ULP, it has been added to $add\_m\_shifted$ using two 32-bit adders, individually works for SP computation, and collectively produce the output for DP (similar to the case of mantissa addition). Further, this rounded mantissa sum has been normalized. The rounding adder in effect is similar to that required for only DP processing. Further to rounding of the mantissa, the exponents has been updated, for mantissa overflow. For this, each exponent update may require to be incremented by one. One adder has been shared for DP and one SP-1, because of shared operands in e_L. Further to this, each exponent has been updated for either of infinity, sub-normal or underflow cases, and each requires separate units. 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.

Thus, the complete architecture needs only one multiplexer for multiplexing the operands, and this belongs to the `SWAP` section. All other processing data path components have been tuned to follow those operands to support dual mode operations without any further extra multiplexing circuitry, except the last one to produce the final 64-bit output. A brief summary of shared resources and extra resources over only

TABLE II: ASIC Implementation Details

	DPdSP		DP		SP
Latency	1	4	1	4	1
Area (mm^2)	0.164	0.172	0.142	0.147	0.065
Period (ns)	7.84	2.56	7.31	2.47	6.07
Power (mW)	9.76	48.38	6.98	33.5	5.35

TABLE III: Comparison with Related Work

	[6] 0.25μm	[7] 0.11μm		Proposed 0.18μm	
Latency	3	5	3	1	4
Area OH[1]	24%	26%	33%	15%	17%
Period OH[1]	9%	9.6%	13.3%	7%	3.5%
Scaled Area[2]	-	0.370	0.433	0.164	0.172
Gate Count[3]	13224	-	-	10288	10794
Period (FO4)[4]	49	18	24.7	87	28.4
Total Delay (FO4)*	140*	65*	55*	87	
Area × Delay [#1]	-	24.05	23.815	14.268	
Area × Delay [#2]	1851360	-	-	895056	

[1]Area/Period OH = (DPdSP - DP) / DP
[2]in mm^2 @180μm = Area @110μm * $(180/110)^2$
[3]Based on minimum size inverter, [4]1 FO4 (ns) ≈ (Tech. in μm) / 2
[5]Obtained after combining all stages delay
[#1]Scaled Area × Total FO4 Delay, [#2]Gate Count × Total FO4 Delay

DP adder is shown in Table-I.

III. Implementation Results and Comparisons

The proposed architecture is synthesized using the open-source "OSUcells Cell [10]" 0.18μm technology, using Synopsys Design Compiler. We have also synthesized only DP and only SP adder using similar data path computational flow, for comparison purpose. The implementation details have been shown in Table-II. Each module has been synthesized for best possible delay. The proposed DPdSP architecture needs roughly 15% more hardware & 7% extra delay than only DP adder, however has 37% saving when compared to combined one DP with 2 SP modules (Area(DP+2*SP)- Area(DPdSP) / Area(DP+2*SP)).

A comparison with previous works is shown in Table-III. Other previously reported DPdSP adder designs [6], [7], [8] support only normal implementation, and lacks exceptional case handling. Though the inclusion of sub-normal support and exceptional case handling is not difficult, it affects the overall area and critical path delay significantly [11], [12]. Because of different technology implementations, comparison is based on the % area and period/delay overhead over corresponding only DP adder based on the same technology. Ozbilen *et. al* [8] has shown a little implementation details, and (approximately) has more than 25% area and 15% delay overhead than their corresponding DP adder. A. Akkas [6] has proposed a DPdSP adder in 0.25μm technology and it needs 24% more hardware than their DP design for a 3 clock cycle latency. Further, [7] has extended his single-path design of [6] and proposed two-path DPdSP adder design. It needs roughly 26% and 33% extra hardware than their only DP adder. Due to their two path method the delay (and overhead) reduced than their earlier [6] design, however, with much larger hardware (and overhead) requirements. The designs in [6], [7] have used a large number of multiplexers (to support dual mode) at various level of architecture ,and have less tuned data path for dual mode operation. Further the extra use of resources (like more adders/subtractors for exponent & mantissa, relatively larger dual shifters, extra mantissa normalizing shifters for dual mode support) made their overhead larger. Whereas, proposed architecture has reduced the multiplexing circuitry (mainly two MUX: one in SWAP and one in Final Output section), with more shared and tuned data path. Compared to previous works, the proposed DPdSP adder design has smaller area & delay overhead when compared to only DP, and has $40\% - 50\%$ smaller $area \times delay$ product. The proposed DPdSP architecture provides full support to normal and sub-normal, along with relevant exceptional case handling.

IV. Conclusions

In this paper, we have presented an architecture for floating point adder with on-the-fly dual precision support, with both normal and sub-normal support, and exceptional case handling. It supports double precision with dual single precision (DPdSP) adder computation. The data path has been tuned with minimal required multiplexing circuitry. The supporting sub-components have been tuned for on-the-fly dual mode computation. It needs approx 15% more resources than DP module and has a benefit of more than 37% reduction in area when compared to combined single DP and two SP module. In comparison to previous works in literature, our proposed DPdSP design has $40\% - 50\%$ smaller $area \times delay$ product, and has smaller area & delay overhead when compared to only DP, and provide more computational support.

References

[1] 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.

[2] K. S. Hemmert and K. D. Underwood, "Fast, efficient floating-point adders and multipliers for fpgas," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 3, no. 3, pp. 11:1–11:30, Sep. 2010.

[3] 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.

[4] 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.

[5] 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.

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

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

[8] 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.

[9] "IEEE Standard for Floating-Point Arithmetic," Tech. Rep., Aug. 2008.

[10] Oklahoma State University, OSUCells, http://vlsiarch.ecen.okstate.edu.

[11] H.-J. Oh, S. Mueller, C. Jacobi, K. Tran, S. Cottier, B. Michael, H. Nishikawa, Y. Totsuka, T. Namatame, N. Yano, T. Machida, and S. H.Dhong, "A fully pipelined single-precision floating-point unit in the synergistic processor element of a cell processor," *Solid-State Circuits, IEEE Journal of*, vol. 41, no. 4, pp. 759–771, 2006.

[12] E. Schwarz, M. Schmookler, and S. Trong, "Hardware implementations of denormalized numbers," in *Computer Arithmetic, 2003. Proceedings. 16th IEEE Symposium on*, 2003, pp. 70–78.