<table>
<thead>
<tr>
<th><strong>Title</strong></th>
<th>Configurable Architectures For Multi-mode Floating Point Adders</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Author(s)</strong></td>
<td>Jaiswal, MK; Bogaraju, SV; So, HKH; Balakrishnan, M; Paul, K; Cheung, RCC</td>
</tr>
<tr>
<td><strong>Citation</strong></td>
<td>IEEE Transactions On Circuits And Systems I: Regular Papers, 2015, v. 62, p. 2079-2090</td>
</tr>
<tr>
<td><strong>Issued Date</strong></td>
<td>2015</td>
</tr>
<tr>
<td><strong>URL</strong></td>
<td><a href="http://hdl.handle.net/10722/214166">http://hdl.handle.net/10722/214166</a></td>
</tr>
<tr>
<td><strong>Rights</strong></td>
<td>This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.; ©2015 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.</td>
</tr>
</tbody>
</table>
Configurable Architectures for Multi-Mode Floating Point Adders
Manish Kumar Jaiswal, B. Sharat Chandra Varma, Hayden K.-H. So, M. Balakrishnan, Kolin Paul, and Ray C. C. Cheung

Abstract—This paper presents two architectures for floating point (FP) adders, which operates in multi-mode configuration with multi-precision support. First architecture (named QPdDP) works in dual-mode which can operate either for quadruple precision or two-parallel double precision. The second architecture (named QPdDPqSP) works in tri-mode which is able to compute either of a quadruple precision, two-parallel double precision and four-parallel single precision computations. The architectures are based on the standard state-of-the-art flow for FP adder which supports the computation of normal and sub-normal operands, along with the support for the exceptional case handling. The key components in the architecture, such as comparator, swap, dynamic shifters, leading-one-detector (LOD), mantissa adders/subtractors, and rounding circuit, are re-designed and optimized for multi-mode computation, to enable efficient resource sharing for multi-precision operands. The data-path in each multi-mode architecture is tuned for multi-precision support with minimal multiplexing circuit overhead. These proposed architectures provide multi-precision SIMD support for lower precision operands, along with high precision computational support, and thus, have a better resource utilization. A fully pipelined version of both adder architectures are presented. The proposed adder architectures are synthesized using UMC 90 nm technology ASIC implementation. The proposed architectures are compared with the best available literature works, and have shown better design metrics in terms of area, delay and area × period, along with more computational support.

Index Terms—ASIC, configurable architecture, digital arithmetic, floating point addition, multi-mode multi-precision arithmetic, SIMD.

I. INTRODUCTION

The floating point (FP) number system [1], due to its wide dynamic range, is a common choice for a large set of scientific, engineering and numerical processing computations. Generally, the performance of these computations greatly depends on the underlying floating point arithmetic processing unit. Several contemporary general purpose processors provide SIMD support for parallel floating point arithmetic computation. This is achieved by using multiple units of single precision and double precision arithmetic hardware.

Manuscript received February 25, 2015; revised May 04, 2015; accepted May 31, 2015. Date of current version July 24, 2015. This work is party supported by the The University of Hong Kong grant (Project Code, 201409176200), the Research Grants Council of Hong Kong (Project ECS 729012E), and the Croucher Innovation Award 2013. This paper was recommended by Associate Editor S. Ghosh.

M. K. Jaiswal, B. S. C. Varma, and H. K.-H. So are with Department of EEE, the University of Hong Kong, Hong Kong, China (e-mail: manishkjj@ee.c.u.hk; varmaf@ee.c.u.hk; hrs@ee.c.u.hk).
M. Balakrishnan and K. Paul are with Department of CSE, Indian Institute of Technology, Delhi, India (e-mail: mbalaa@cse.itsd.ernet.in; kolin@cse.itsd.ernet.in).
R. C. C. Cheung is with Department of EE, City University of Hong Kong, Hong Kong, China (e-mail: r.cheung@cityu.edu.hk).

Digital Object Identifier 10.1109/TCSI.2015.2452351

To provide much higher floating point computational support, several custom high performance computing machines from major semiconductor companies like Intel, IBM, ARM, and Nvidia, provide huge multi-core computing systems. Each core in these generally contains a larger vector floating point unit (VFU). The VFU units in these computing system cores contain separate vector processing arrays of single precision and double precision computational units. Like the Synergistic Processing Element (SPE), in Cell-BE processor [2] from IBM, contains a vector array of 4 single precision and an array of 2 double precision. The ARM VFU coprocessor (VFU9-S) [3] provides a vector array of 16 single precision FP units and 8 double precision vector array. Like wise, the Intel has developed a 60 core Xeon Phi\textsuperscript{TM} computing machine, in which each core contains an array of 16 single precision units and an array of 8 double precision units. Similarly, the Nvidia’s next generation CUDA\textsuperscript{TM} architecture: Kepler\textsuperscript{TM} G110 [4] contains 15 Streaming Multiprocessor (SMX), in which each SMX contains 192 single precision core and 64 double precision core. Generally, these processing systems contain separate units/arrays for single precision and double precision computations. However, if an unified dynamically configurable computational unit can support a double precision with dual/two-parallel single precision (DPdSP) arithmetic, or quadruple precision with dual/two-parallel double precision (QPdDP) arithmetic, it can save a large silicon area in the above computing machines.

Furthermore, the availability for double precision arithmetic computation is not enough and the demand for high precision arithmetic is increasing in many application areas [5], [6]. In this view, this paper is aimed towards the configurable multi-mode multi-precision floating arithmetic architecture design, currently aiming towards the addition/subtraction arithmetic, with high precision support.

Some literature have focused on the standard cell based ASIC architectures for configurable dual-mode multi-precision floating point arithmetic, included with quadruple precision support [7]–[17]. Many of these works [7]–[11] are dedicated to the dual-mode multiplier design, and [12] proposed a dual-mode FMA architecture. Issen et al. [14] proposed a dual-mode division architecture. Some literature [15]–[17] have proposed dual-mode architectures for adder. These works have tried to improve the resource utilization for the hardware with multi-precision computational support. However, the overhead of extra hardware, and unoptimized data-path and resource sharing lead to large overhead of area and delay metrics. Furthermore, they have limited support only for normal operands. The dual-mode adder architectures of [15], [16] used a large number of multiplexers (to support dual mode) at various level of architecture, and have less tuned data path for
The present work on the multi-mode floating point adder architecture follows the basic single-path algorithm for this computation. A floating point arithmetic computation involves computing separately the sign, exponent and mantissa part of the operands, and later combine them after rounding and normalization [1]. The standard format for floating point numbers are as follows:

\[
\begin{align*}
SP : & \begin{array}{c}
\text{Sign} \quad \text{Exponent} \quad \text{Mantissa}
\end{array} \\
& \begin{array}{c}
\text{bit} \quad \text{8} \quad \text{bit} \quad \text{23} \quad \text{bit}
\end{array} \\
DP : & \begin{array}{c}
\text{Sign} \quad \text{Exponent} \quad \text{Mantissa}
\end{array} \\
& \begin{array}{c}
\text{bit} \quad \text{11} \quad \text{bit} \quad \text{52} \quad \text{bit}
\end{array} \\
QP : & \begin{array}{c}
\text{Sign} \quad \text{Exponent} \quad \text{Mantissa}
\end{array} \\
& \begin{array}{c}
\text{bit} \quad \text{15} \quad \text{bit} \quad \text{112} \quad \text{bit}
\end{array}
\end{align*}
\]

A basic state-of-the-art computational flow of the floating point adder is shown in the Algorithm 1. Here, steps 6–7 and step-22 require for sub-normal processing. Sub-normals represents the numbers whose magnitude fall beyond the minimum normal format value. The sub-normal representation helps in preserving the underflow result of a computation, which can not be represented by a valid normal number. The unification of these normal and sub-normal number in floating point representation, generally, makes the floating point computation a tough task. In present work of multi-mode multi-precision architectures, each steps of the flow are constructed for the support of the multi-mode operation with resource sharing and tuned data-path with minimum multiplexing circuitry.

Algorithm 1 FP Adder Computational Flow [1]

1: \{IN1,IN2\} Input Operands
2: Data Extraction & Exceptional Check-up:
3: \{S1(Sign1), E1(Exponent1), M1(Mantissa1)\} ← IN1
4: \{S2, E2, M2\} ← IN2
5: Check for INFINITY, NAN
6: Check for SUB-NORMALs
7: Update Exponents & Mantissa’s MSB for SUB-NORMALs
8: COMPARE, SWAP & Dynamic Right SHIFT:
9: \{IN1, IN2\} ← \{E1, M1\} > \{E2, M2\}
10: Large_E, M ← IN1 ? E1, M1 : E2, M2
11: Small_E, M ← IN2 ? E2, M2 : E1, M1
12: Right Shift ← Large E - Small E
13: Small_M ← Small M × Right Shift
14: Mantissa Computation:
15: \(OP = S1 \oplus S2\)
16: if \(OP = 1\) then
17: \(\text{Add}_M ← \text{Large}_M \times \text{Small}_M\)
18: else
19: \(\text{Add}_M ← \text{Large}_M \times \text{Small}_M\)
20: Leading-One-Detection & Dynamic Left SHIFT:
21: Left Shift ← LOD(Add M)
22: Left Shift ← Adjustメント for SUB-NORMAL or Underflow
23: Add M ← Add M ← Left Shift
24: Normalization & Rounding:
25: Mantissa Normalization & Compute Rounding ULP based on Guard, Round & Sticky Bit
26: \(\text{Large}_E ← \text{Large}_E + \text{Add}_M(\text{MSB}) - \text{Left}_\text{Shift}\)
27: Finalizing Output:
28: Update Exponent & Mantissa for Exceptional Cases
29: Determine Final Output
30: }

III. CONFIGURABLE QUADRUPLE PRECISION/DUAL
(TWO-PARALLEL) DOUBLE PRECISION (QPdDP) ADDER
ARCHITECTURE

The architecture for proposed QPdDP adder is presented here to provide the higher precision requirements of the applications, with dual-mode support. The computational flow of the QPdDP...
The dual-mode comparator unit for dual-mode QPdDP adder is shown in Fig. 4. The comparator unit determines which operand is large and which one is small. This unit is shared among the QP and both DP operands. It is comprised of two comparator units for both DPs operands, which generates their corresponding comparison results. These DP results are further combined to form QP comparison. The individual “greater-than” and “equivalent” functions are designed using following Boolean functions:

\[
\begin{align*}
\text{QP Comparison:} & \quad \text{dp}_1\_\text{in1-gt-in2} = (\text{in}[127:64] > \text{in}[127:2]) \quad ? : 0 \\
& \quad \text{dp}_2\_\text{in1-gt-in2} = (\text{in}[127:64] > \text{in}[127:2]) \quad ? : 0 \\
\text{QP-Comparisons:} & \quad \text{dp}_1\_\text{in1-eq-in2} = (\text{in}[127:64] = \text{in}[127:64]) \quad ? : 0 \\
& \quad \text{dp}_2\_\text{in1-eq-in2} = (\text{in}[127:64] = \text{in}[127:64]) \quad ? : 0 \\
& \quad \text{qp\_in1-gt-in2} = \text{dp}_1\_\text{in1-gt-in2} \land (\text{dp}_2\_\text{in1-eq-in2} \land (\text{in}[127:64] = \text{in}[127:64] \land \text{dp}_1\_\text{in1-gt-in2} ))
\end{align*}
\]

This scenario is used to share the resources related to sub-normal, infinity, and NaN detection of QP and second DP operands. The detection of sub-normals are shown in the Fig. 3, which detects the zero value of corresponding exponent. Similarly, the checks for infinity and NaN are handled (which detects the maximum binary value of corresponding exponent). After these exceptional checks the exponent and mantissa are updated accordingly, as shown in Fig. 3. In comparison to only QP computation, this unit requires extra related resources for first DP (DP-1) operands.

The dual-mode comparator unit for dual-mode QPdDP adder is shown in Fig. 4. The comparator unit determines which operand is large and which one is small. This unit is shared among the QP and both DP operands. It is comprised of two comparator units for both DPs operands, which generates their corresponding comparison results. These DP results are further combined to form QP comparison. The individual “greater-than” and “equivalent” functions are designed using following Boolean functions:

\[
\begin{align*}
\text{QP Comparison:} & \quad \text{dp}_1\_\text{in1-gt-in2} = (\text{in}[127:64] > \text{in}[127:2]) \quad ? : 0 \\
& \quad \text{dp}_2\_\text{in1-gt-in2} = (\text{in}[127:64] > \text{in}[127:2]) \quad ? : 0 \\
\text{QP-Comparisons:} & \quad \text{dp}_1\_\text{in1-eq-in2} = (\text{in}[127:64] = \text{in}[127:64]) \quad ? : 0 \\
& \quad \text{dp}_2\_\text{in1-eq-in2} = (\text{in}[127:64] = \text{in}[127:64]) \quad ? : 0 \\
& \quad \text{qp\_in1-gt-in2} = \text{dp}_1\_\text{in1-gt-in2} \land (\text{dp}_2\_\text{in1-eq-in2} \land (\text{in}[127:64] = \text{in}[127:64] \land \text{dp}_1\_\text{in1-gt-in2} ))
\end{align*}
\]

This scenario is used to share the resources related to sub-normal, infinity, and NaN detection of QP and second DP operands. The detection of sub-normals are shown in the Fig. 3, which detects the zero value of corresponding exponent. Similarly, the checks for infinity and NaN are handled (which detects the maximum binary value of corresponding exponent). After these exceptional checks the exponent and mantissa are updated accordingly, as shown in Fig. 3. In comparison to only QP computation, this unit requires extra related resources for first DP (DP-1) operands.

The dual-mode comparator unit for dual-mode QPdDP adder is shown in Fig. 4. The comparator unit determines which operand is large and which one is small. This unit is shared among the QP and both DP operands. It is comprised of two comparator units for both DPs operands, which generates their corresponding comparison results. These DP results are further combined to form QP comparison. The individual “greater-than” and “equivalent” functions are designed using following Boolean functions:

\[
\begin{align*}
\text{QP Comparison:} & \quad \text{dp}_1\_\text{in1-gt-in2} = (\text{in}[127:64] > \text{in}[127:2]) \quad ? : 0 \\
& \quad \text{dp}_2\_\text{in1-gt-in2} = (\text{in}[127:64] > \text{in}[127:2]) \quad ? : 0 \\
\text{QP-Comparisons:} & \quad \text{dp}_1\_\text{in1-eq-in2} = (\text{in}[127:64] = \text{in}[127:64]) \quad ? : 0 \\
& \quad \text{dp}_2\_\text{in1-eq-in2} = (\text{in}[127:64] = \text{in}[127:64]) \quad ? : 0 \\
& \quad \text{qp\_in1-gt-in2} = \text{dp}_1\_\text{in1-gt-in2} \land (\text{dp}_2\_\text{in1-eq-in2} \land (\text{in}[127:64] = \text{in}[127:64] \land \text{dp}_1\_\text{in1-gt-in2} ))
\end{align*}
\]

This scenario is used to share the resources related to sub-normal, infinity, and NaN detection of QP and second DP operands. The detection of sub-normals are shown in the Fig. 3, which detects the zero value of corresponding exponent. Similarly, the checks for infinity and NaN are handled (which detects the maximum binary value of corresponding exponent). After these exceptional checks the exponent and mantissa are updated accordingly, as shown in Fig. 3. In comparison to only QP computation, this unit requires extra related resources for first DP (DP-1) operands.

The dual-mode comparator unit for dual-mode QPdDP adder is shown in Fig. 4. The comparator unit determines which operand is large and which one is small. This unit is shared among the QP and both DP operands. It is comprised of two comparator units for both DPs operands, which generates their corresponding comparison results. These DP results are further combined to form QP comparison. The individual “greater-than” and “equivalent” functions are designed using following Boolean functions:

\[
\begin{align*}
\text{QP Comparison:} & \quad \text{dp}_1\_\text{in1-gt-in2} = (\text{in}[127:64] > \text{in}[127:2]) \quad ? : 0 \\
& \quad \text{dp}_2\_\text{in1-gt-in2} = (\text{in}[127:64] > \text{in}[127:2]) \quad ? : 0 \\
\text{QP-Comparisons:} & \quad \text{dp}_1\_\text{in1-eq-in2} = (\text{in}[127:64] = \text{in}[127:64]) \quad ? : 0 \\
& \quad \text{dp}_2\_\text{in1-eq-in2} = (\text{in}[127:64] = \text{in}[127:64]) \quad ? : 0 \\
& \quad \text{qp\_in1-gt-in2} = \text{dp}_1\_\text{in1-gt-in2} \land (\text{dp}_2\_\text{in1-eq-in2} \land (\text{in}[127:64] = \text{in}[127:64] \land \text{dp}_1\_\text{in1-gt-in2} ))
\end{align*}
\]
exponents), four 53-bit (for both DP mantissas) and two 113-bit (for QP mantissa) SWAP components for all the computations of this section. However, to minimize the swapping overhead, the unified exponents, mantissas, and greater-than control signals are generated by multiplexing either of the quadruple precision or both double precision operands (as shown in Fig. 5).

This is an important step included in the dual-mode QPdDP architectural flow, which helps to design a tuned data-path computation in later stages, with reduced multiplexing circuitry. Using these unified exponents, mantissas and greater-than control signals, it requires only four 11-bit (for exponents) and four 64-bit (for mantissas) SWAP circuitry for entire processing. Effectively, it needs SWAP components slightly more than it requires for only QP (only QP requires two 15-bit SWAP for exponents and two 113-bit SWAP for mantissas), along with extra multiplexing circuitry needed to generate unified signals, however, facilitates the tuned data-path processing. Further, 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 basic functional component in SWAP unit is 2:1 MUX (for unified data generation) and few nand gates (for effective operation, OP). The 2:1 MUX is implemented using simple Boolean logic as \( y = sel.A + sel.B \). The \( m_L \) contains mantissa of either large QP operand or both of large DP operands. Similarly, \( m_S \) contains small mantissas. Likewise, \( e_L \) contains large exponent, and \( e_S \) contains small exponents, either of QP or both DP operands.

For right shifting of small mantissas of quadruple and both double precision operands, a dual-mode dynamic right shifter (DRS) is designed. The QPdDP dual-mode dynamic right shifter is shown in Fig. 6, which is used to right-shift the small mantissas of either QP or both DPs. The initial step in it right-shifts the operand by 64-bit in case of QP mode with its true shift bit. The later 6-stages in it works in dual mode, either for QP or for both DPs operands. Each dual-mode stage contains two shifters for each of 64-bit blocks, which right-shifts their inputs corresponding to their shifting bit (either for quadruple or double precision). Each of the stage also includes one multiplexer which selects between lower shifting output or their combination with primary input to the stage, based on the mode of the operation. The root functional component in this unit is 2:1 MUX which is implemented with \( y = sel.A + sel.B \) logic.

Further to the right shifting of small mantissas, the core operation of mantissa addition/subtraction fall in the computational flow. The large mantissas and right-shifted small mantissas undergo addition/subtraction based on their effective operation. This computation is performed in dual-mode using two 64-bit integer adder-subtraction unit (implemented using Carry-look-ahead method), which individually works for each double precision, and collectively works for quadruple precision computation (as shown in Fig. 7). This unit generates the lower
and upper parts of addition/subtraction separately. This component requires effectively similar resources as present in only QP adder.

The lower and upper mantissa addition/subtraction results generated in previous unit combined in “Mantissa SUM and LOD_in unit,” to provide the actual sum (either for QP or both DPs), mantissa overflow, and the input for next level unit, leading-one-detector (LOD). It requires a couple of 2:1 MUX and OR-gates. This unit is shown in Fig. 8.

The mantissa sum now requires to check for any underflow, which requires a leading-one-detector (LOD), and further a dynamic left shifter for mantissa. This situation occurs when two very close mantissa undergoes subtraction operation. The LOD requires to compute the left-shift amount. In present context, the dual-mode leading-one-detector for QPdDP processing is shown in Fig. 9. The input of LOD is either a QP LOD_in or two DP LOD_in. The dual mode LOD is designed in a hierarchical manner (using basic unit of LOD_2:1, which consists of a AND, OR and NOT gate), which leads to 64-bit LOD. It is comprised of two 64-bit LOD. The individual 64-bit LOD provides left shift information for both DP operands, and collectively for QP operand. It effectively requires resources equivalent to that of only QP LOD.

The left shift amount, thus generated from LOD, is then updated for sub-normal input cases (both sub-normal input operands) and underflow cases (if left shift amount exceeds or is equal to the corresponding large exponent). For both sub-normal input operand case, the corresponding left shift is forced to zero, and for the underflow case, the corresponding left shift will be equal to corresponding large exponent decremented by one. For the exponent decrements, one of the related subtractor is shared for the QP and first DP, as done in the case of computation of right shift amount. This becomes possible because the required LSBs of $e_L$ are shared among the exponents of QP and first DP. This exponent decrements requires one 7-bit (shared for QP and a DP) and one 6-bit (for another DP) decrement. All other computations, related to left shift update need to be computed separately for QP and both DPs. With true $qp_{dp}$, both DPs’ left shift are set to zero, and for false $qp_{dp}$ the QP left shift is forced to zero.

The mantissa sum is then shifted left using a dual-mode dynamic left shifter (DLS) (Fig. 10). The basic concept for this architecture is similar to the dual-mode dynamic right shifter, except that there is change in the shifting direction. The first stage

works in single mode only for QP shifting, and later 6-stages works in dual-mode either for QP or both DP.
TABLE I
RESOURCE SHARING IN QPdDPqSP ADDER SUB-COMPONENTS

<table>
<thead>
<tr>
<th>Architectural Components</th>
<th>QPdDP Adder Architecture</th>
<th>QPdDPqSP Adder Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Extraction and Sub-normal Handler</td>
<td>&quot;Subnormal, infinity, and NAN&quot; checks of QP and one DP</td>
<td>&quot;Subnormal, infinity, and NAN&quot; check of QP, DP-2 and SP-4</td>
</tr>
<tr>
<td>Comparator, Dynamic Shifters, LOD</td>
<td>For one DP</td>
<td>For one DP and two DPs</td>
</tr>
<tr>
<td>SWAP: Large Sign, Exp, Mant and OP</td>
<td>Shared QP and both DP</td>
<td>Shared QP; both DP and all SPs</td>
</tr>
<tr>
<td>Right Shift Amount</td>
<td>Shared SWAP of QP and both DP</td>
<td>Shared SWAP of QP, both DP and all SPs</td>
</tr>
<tr>
<td>Mantissa Add/Sub, Sum Normalization</td>
<td>Subtraction for QP and both DP</td>
<td>Subtraction for QP, both DP and all SPs</td>
</tr>
<tr>
<td>Left Shift Update</td>
<td>Shared QP and both DP</td>
<td>Shared QP, both DP and all SPs</td>
</tr>
<tr>
<td>1-bit Left Shifter</td>
<td>two 1-bit MUX</td>
<td>four 1-bit 3:1 MUX and small control logic</td>
</tr>
<tr>
<td>Rounding</td>
<td>ULP addition shared among QP and both DP</td>
<td>ULP addition shared among QP and both DP and all SPs</td>
</tr>
<tr>
<td>Exponent Update</td>
<td>Shared the update of QP and one DP</td>
<td>Shared the update of QP, one DP and one SP</td>
</tr>
<tr>
<td>Final Processing</td>
<td>Post Round Update of Exponent and Mantissa</td>
<td>Post Round Update of Exponent and Mantissa</td>
</tr>
</tbody>
</table>

![Fig. 11. QPdDP dual mode 1-bit left shifter.](image1)

![Fig. 12. ULP generation and QPdDP dual mode ULP addition.](image2)

The output of dual-mode DLS then undergoes 1-bit left shifting (normalization), in-case of mantissa overflow in mantissa-addition. The dual-mode 1-bit left-shifter unit is shown in Fig. 11. It either performs a 1-bit left shifting for QP mode, or carries out 1-bit left-shifting for both DPs, with respect to their corresponding mantissa overflow. The lowest functional component required here is 2:1 MUX. The resource requirement for this shifter-unit is similar to that of a only QP 1-bit shifter, except two 1-bit 2:1 MUX.

The output from 1-bit left shifter is further processed for rounding computation and ULP-addition (Fig. 12). In present work, the round-to-nearest method is included, however, other method can be included easily. The rounding ULP computations are done based on LSB precision bit, Guard bit, Round bit, and Sticky bit. Here, the ULP computation is implemented separately for each of QP and both DPs. However, the ULP-addition is shared among both, as shown in Fig. 12.

Parallel to above mantissa processing, in Exponent-update unit, the exponents are updated for mantissa overflow and mantissa underflow. In this, the large exponents need to be incremented by one or decremented by left shift amount (\(\text{LargeExp} + \text{mant.add} - \text{leftShift}\)). Since large exponent \((e.L)\) either contains large QP exponent or both DPs exponents, this update is shared for the QP and DP-1, by sharing a subtractor, similar to left shift update computation. In effect it requires a 15-bit shared subtractor and a 11-bit subtractor for DP-2. Thus it needs an extra 11-bit subtractor for DP-2 processing, and a 7-bit multiplexer for left shift amount multiplexing for the shared subtractor, as an overhead over only QP processing.

Finally, the exponents and mantissas are updated for underflow, overflow, sub-normal, and exceptional cases to produce the final output, and each requires separate units for QP and both DPs. For overflow, the exponent will be set to infinity and mantissa will be set to zero, and for underflow case exponent will be set to zero and mantissa will take its related computed value. The computed signs, exponents and mantissas of quadruple precision and both double precision are finally multiplexed to produce the final 128-bit output, which either contains a QP output or two DP outputs. A qualitative resource sharing and overhead of dual mode QPdDP over only QP adder is shown in Table I.

IV. CONFIGURABLE QUADRUPLE PRECISION/DUAL (TWO-PARALLEL) DOUBLE PRECISION/QUAD (FOUR-PARALLEL) SINGLE PRECISION (QPdDPqSP) ADDER ARCHITECTURE

Based on the idea of proposed dual-mode adder architecture and with further investigation, an architecture for QPdDPqSP adder is proposed here to have a tri-mode multi-precision adder computation. This architecture works in three modes: Quadruple/Dual (Two-Parallel) Double/Quad (Four-Parallel) Single Precision. This architecture is devised by, replacing double precision components of the earlier dual-mode QPdDP adder architecture, by a dual-mode DPdSP (double precision with dual/two-parallel single precision) adder component. This key idea, along with some related reconstruction, facilitates the tri-mode operation. The computational flow of the QPdDPqSP adder architecture is again based on the computational flow of Algorithm 1.
The input/output register for the tri-mode architecture is assumed as shown in Fig. 13. The 128-bit input operands, contain either 1 set of quadruple precision or 2 sets of double precision or 4 sets of single precision operands. Based on the two-bit control signal \( q_d, d_s, s_p \), it can be dynamically switched to either quadruple precision or dual double precision or quad single precision mode as follows:

\[
\begin{align*}
q_d & : \text{QP Mode} \\
d_s & : \text{DP Mode} \\
q_p & : \text{SP Mode}
\end{align*}
\]

All the computational steps in QPdDPqSP tri-mode adder are discussed below with their architectural details.

Similar to QPdDP, in its data extraction unit, the sub-normal, infinity and NaN checks are shared for QP, second DP and fourth SP operands. It is also shared for first DP and second SP (Fig. 14). The tri-mode comparator unit is also shown in Fig. 14, which first compares all the SPs’ operands, and then combines them to produce for both DPs’ operands comparison. The DPs’ comparator outcomes are combined to produce QP comparison result. This unit effectively requires similar resources as in only QP comparator.

The SWAP unit of QPdDPqSP adder architecture is shown in Fig. 15. Based on the mode of the operation, it initially generates effective unified “greater than” signals (\( e_1, e_2, c_3, \) and \( c_4 \)). The exponents of QP, both DPs and all SPs are then multiplexed in to unified 32-bit exponents \( e_1 \) (first exponent) and \( e_2 \) (second exponent). Each 8-bit of these \( e_1 \) and \( e_2 \) acts as SPs’ exponents, or each 16-bit acts as DPs’ exponents, or it will act for QP as a whole. Similarly, the mantissas are multiplexed in to unified 128-bit \( m_1 \) and \( m_2 \). These unified exponents and unified mantissas helps to have a tuned data-path flow in the architecture, and enables to minimize the multiplexing circuitry. Based on the effective “greater than” signals (\( e_1, e_2, c_3, \) and \( c_4 \)) unified exponents \( (e_1, e_2) \) and unified mantissas \( (m_1, m_2) \) serve the purpose for either of QP or DPs or SPs. Furthermore, the large sign and effective operation are computed for each mode’s operands.

The next unit, also shown in Fig. 15, computes the right shift amount for smaller mantissas. The shift amount is computed by a 32-bit subtraction of unified large exponent \( e_1 \) and unified small exponent \( e_2 \). This serves the purpose for the right shift amount of either of QP, or both DPs or all SPs as shown in Fig. 15. The small mantissa is then right shifted by the tri-mode dynamic right shifter unit (Fig. 16). The first stage of this shifter unit right shifts the input by 64-bit, and works for QP purpose. The second stage works in dual mode fashion, as in case
on QPdDP, either for both DP or collectively for QP. The remaining stages (stage 3–7) of this unit are formed by combining the two dual mode stage of DpDSP shifter, which collectively works in tri-mode, either for QP or both DPs or four SPs, based on the effective mode of the operation. Fig. 16 further shows the tri-mode mantissa addition/subtraction module. It uses four 32-bit add/sub units with control logic to accomplish this task. By combining their outputs, this unit later generates the unified effective mantissa addition/subtraction result and input for the leading-one-detector, which contains the data either for QP, or both DPs or all four SPs.

The left shift amount is then updated for sub-normal input cases, underflow cases (if left shift amount exceeds or equal to the corresponding large exponent). The underflow case requires the left shift amount equate to exponent decrement by one. Since the large exponents for all mode is shared, the exponent decrement is shared among the operands. One 7-bit decrement is shared for QP, first DP and first SP, and one 6-bit decrement is shared for second DP and third SP. Remaining SPs' decrements are done separately.

Now, to shift the mantissa addition/subtraction result by left shift amount, the QPdDPqSP tri-mode dynamic left shifter is used (as shown in Fig. 18). The working principle of tri-mode dynamic left shifter is similar to that of tri-mode dynamic right shifter. Its first stage works for only QP, second stage works in dual mode (either for QP or both DPs), and remaining stages perform in tri-mode (either for QPs or both DPs or all four SPs).

After left shifting of mantissa result, rounding is performed. As in case of QPdDP, the ULP computation requires separate
units for each of QP, DPs and SPs as shown in Fig. 12. The ULP addition is shared among all mode of the operation, as shown in Fig. 18. Then each exponent is updated for corresponding mantissa’s overflow or underflow, which needs them to be either incremented by one or decremented by left shift amount. This portion is shared as discussed for left shift update. In the end, final normalization, and, exponents and mantissas updates for exceptional cases are performed, and output are multiplexed for a 128-bit output for given mode of the operation. The resource overhead of QPdDPqSP adder over only QP adder is shown in the Table I.

The detailed implementation results of the dual mode QPdDP and tri-mode QPdDPqSP adder architectures are shown in the Section V. It also discusses the comparisons with the previous work available in the literature, along with the improvements in the proposed architectures.

V. IMPLEMENTATION RESULTS AND COMPARISONS

Both of the proposed dual-mode QPdDP and tri-mode QPdDPqSP architectures are implemented and synthesized for standard-cell based ASIC platform. Synthesis is performed using the UMC90 nm technology, using Synopsys Design Compiler. An architecture for QP only adder is also designed (using similar data path computational flow) and synthesized for area and delay overhead measurements. These architectures are designed with four pipeline stages (as shown in Fig. 1). Similar architectures for DP only and SP only adders also synthesized, using same computational flow. The implementation details are shown in Table II. Each module is synthesized with the options of best possible clock-period, minimum-area, and medium-effort for synthesis. Second pipeline stage in all the architectures appears in critical path, and decides the clock-period. The proposed architectures are functionally verified for 5 million random test cases in each mode, with different pairs of
TABLE III

<table>
<thead>
<tr>
<th></th>
<th>QPdDP Architecture</th>
<th>QPdDPqSP Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.0%</td>
<td>Proposed 90nm</td>
</tr>
<tr>
<td></td>
<td>Proposed 90nm</td>
<td></td>
</tr>
<tr>
<td>Latency</td>
<td>3 4</td>
<td>3 4</td>
</tr>
<tr>
<td>Area OH(^1)</td>
<td>15.3% 14.01%</td>
<td>35.80% 27.31%</td>
</tr>
<tr>
<td>Period OH(^1)</td>
<td>14.12% 8.71%</td>
<td>18.65% 10.11%</td>
</tr>
<tr>
<td>Scalled Area(^2)</td>
<td>26967 33702</td>
<td>239250 199723</td>
</tr>
<tr>
<td>Gate Count(^3)</td>
<td>- -</td>
<td>90116 91071</td>
</tr>
<tr>
<td>Period (FO4)(^4)</td>
<td>65.38 35.92</td>
<td>28.9 17.81</td>
</tr>
<tr>
<td>Scalled-Area(^2) × FO4-Peiod(^4)</td>
<td>1.76 1.21</td>
<td>2.26 2.87</td>
</tr>
<tr>
<td>Gate-Count(^3) × FO4-Peiod(^4)</td>
<td>- -</td>
<td>- -</td>
</tr>
</tbody>
</table>

\(^1\) Area/Period OH = \text{(QP<sub>c</sub>DP - QP)}/QP

\(^2\) in \(\mu m^2\) 90 n.m = (Area @110 n.m) × (90/110)^2

\(^3\) Based on minimum size inverter

\(^4\) 1 FO4 (ns) ≈ (Tech. in \(\mu m\))/2

operands, like normal-normal, subnormal-normal, normal-subnormal, and subnormal-subnormal; all mixed with exceptional cases.

The proposed dual-mode QPdDP adder architecture requires approximately 17% more hardware resources and roughly 5.45% extra period than only QP adder. Furthermore, the proposed QPdDP adder requires approximately 35.86% smaller area when compared with the combination of a QP only and two-units of DP only adder ((QP+2*DP-QPdDP)/(QP+2*DP)).

Similarly, the tri-mode QPdDPqSP adder architectures require roughly 19.78% extra hardware and 31.81% large period when compared to corresponding only QP adder. However, when compared with the combination of a QP only, two-units of DP only and four-units of SP only adder, the QPdDPqSP adder requires roughly 55.81% smaller area ((QP+2*DP+4*SP-QPdDPqSP)/(QP+2*DP+4*SP)).

Literature contains very limited work on the dual-mode QPdDP architecture, whereas the proposed tri-mode QPdDPqSP adder architecture stands as a fresh proposal. A comparison of dual-mode QPdDP architecture with previous works is shown in Table III. The related information for tri-mode QPdDPqSP architecture are also included in Table III, to show its merit. The comparisons are carried out in terms of % area and period/delay overhead over corresponding only QP adder. However, when compared with the combination of a QP only, two-units of DP only and four-units of SP only adder, the QPdDPqSP adder requires roughly 55.81% smaller area ((QP+2*DP+4*SP-QPdDPqSP)/(QP+2*DP+4*SP)).

Similarly, [16] has proposed dual-mode DPdSP and QPdDP adder architectures, with two pipeline versions, 3-stage and 5-stage pipelines. These architectures do not provide computational support for sub-normal operands and without any exceptional case handling. These were synthesized with 110 nm standard-cell ASIC library. For its 3-stage pipeline QPdDP design it has 35.8% area-overhead and 18.65% period overhead, however, for its 5-stage design the area-overhead is 27.31% and period-overhead is 10.11%. Compared to this work, the proposed dual-mode QPdDP architecture outperforms them in terms of design overheads, as well as in terms of design metrics: the area, period, and area × period.

Thus, compared to previous works, the proposed dual-mode QPdDP adder architecture has smaller area-overhead and delay-overhead when compared to only QP adder. The proposed QPdDP architecture shows an improvement of approximately 50% in terms of area × period product.

VI. CONCLUSIONS

This paper has presented two dynamically-configurable multi-mode architectures for floating point adder, with on-the-fly multi-precision support. The presented dual-mode QPdDP and tri-mode QPdDPqSP architectures provides normal and sub-normal computational support and exceptional case handling. Both architectures are presented in fully pipelined format, with 4-stages pipeline. The data path in both architectures has been tuned with minimal required multiplexing circuitry. The individual components of the architectures have been constructed for on-the-fly multi-mode computation, with minimum required multiplexing. The dual-mode QPdDP adder architecture needs approximately 17% more resources and 5.45% more delay-period than the QP only adder. Similarly, the tri-mode QPdDPqSP adder architecture has approximately 20% area overhead and 32% delay overhead over QP only adder. In comparison to previous works in literature, the proposed dual-mode QPdDP design has approximately 50% smaller area × period product, and has smaller area and delay
overhead when compared to only QP, and provide more computational support. Moreover, the proposed tri-mode QPdpDpSP adder architecture stands as a fresh proposal, while showing a promising design parameters when compared with dual-mode QPdpDp architectures. Our future work is targeted towards the promising design parameters when compared with dual-mode QP and providing more computational support.

**REFERENCES**


Manish Kumar Jaiswal (S’12–M’14) received his B.Sc. and M.Sc. degrees in electronics from D.D.U. Gorakhpur University, Gorakhpur, India, in 2002 and 2004, respectively. He obtained his M.S.(By Research) from EE Department—I.I.T. Madras in 2009 and Ph.D. from EE Department—City University of Hong Kong, China, in 2014. He is currently a Post-Doctorate Fellow in Department of EEE at The University of Hong Kong. He received an Outstanding Academic Performance award during his Ph.D. at CityU-HK. He worked as a Lecturer in the Department of Electronics at D.D.U. Gorakhpur University for a year (2005-2006), and as a Faculty Member in Department of EE at the ICFAI University, Dehradun, India, for two years (2009-2011). 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.

B. Sharat Chandra Varma received his B.Eng. degree in electronics and communications engineering from Visveswaraya Technological University, India, the M.S. degree in VLSI-CAD from Manipal University, India, and his Ph.D. from Amarnath and Shasi Khosla School of Information Technology at the Indian Institute of Technology, Delhi, India in January 2015. He is a Postdoctoral Research Fellow at the Department of Electronic and Electrical Engineering in the University of Hong Kong, China. He has also worked as a Software Engineer with QuickLogic India Pvt. Ltd. for one year, where he developed CAD tools for QuickLogic FPGAs. His research interests include FPGA, hardware accelerators, hardware-software co-design, and VLSI.

Hayden K.-H. So (S’03–M’07) received the B.S., M.S., and Ph.D. degrees in electrical engineering and computer sciences from University of California, Berkeley, CA, USA, 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, China. He received the Croucher Innovation Award in 2013 for his work in power-efficient high-performance heterogeneous computing systems. He was also awarded the University Outstanding Teaching Award (Team) in 2012, as well as the Faculty Best Teacher Award in 2011.

M. Balakrishnan received his B.E. (Hons.) degree in electronics and electrical engineering from BITS Pilani, India, in 1977 and the Ph.D. from the Electrical Engineering Department, IIT Delhi, India, in 1985. He is a Professor in the Department of Computer Science and Engineering at IIT Delhi. He worked as a Scientist in CARE, IIT Delhi, from 1977 to 1985 where he was involved in designing and implementing real-time DSP systems. For the last 27 years, he is involved in teaching and research in the areas of digital systems design, electronic design automation and embedded systems. He has supervised 10 Ph.D. students, 3 MSR students, 145 M.Tech/B.Tech projects, and published nearly 100 conference and journal papers. Further, he has held visiting positions in universities in Canada, the United States, and Germany. At IIT Delhi, he has been the Philips Chair Professor, Head of the Department of Computer Science and Engineering, Dean of Post Graduate Studies and Research at IIT Delhi and Deputy Director (Faculty) at IIT Delhi. His research interests include embedded systems design, low power, and system level design. Currently he is involved in developing a number of assistive devices for the visually impaired.
Kolin Paul received his B.E. degree in Electronics and Telecommunication Engineering from NIT Silchar, India, in 1992 and Ph.D. in Computer Science from BE College (DU), Shibpore, India, in 2002. He is an Associate Professor in the Department of Computer Science and Engineering at IIT Delhi, India. During 2002-2003 he did his postdoctoral studies at Colorado State University, Fort Collins, CO, USA. He has previously worked at IBM Software Labs. His last appointment was as a Lecturer in the Department of Computer Science at the University of Bristol, U.K. His research interests are in understanding high performance architectures and compilation systems. In particular he works in the area of adaptive/reconfigurable computing trying to understand its use and implications in embedded systems.

Ray C. C. Cheung (M'07) received the B.Eng. and M.Phil. degrees in computer engineering and computer science and engineering at the Chinese University of Hong Kong (CUHK), Hong Kong, China, in 1999 and 2001, respectively, and the Ph.D. degree in computing at Imperial College London, London, U.K., in 2007. In 2009, he worked as a Visiting Research Fellow in the Department of Electrical Engineering, Princeton University. He is currently an assistant professor at City University of Hong Kong (CityU). His research team, CityU Architecture Lab for Arithmetic and Security (CALAS) focuses on the following research topics: reconfigurable trusted computing, SoC VLSI designs, cryptography, and embedded biomedical VLSI designs.