Abstract—Fast Fourier transform (FFT) coprocessor, having a significant impact on the performance of communication systems, has been a hot topic of research for many years. The FFT function consists of consecutive multiply add operations over complex numbers, dubbed as butterfly units. Applying floating-point (FP) arithmetic to FFT architectures, specifically butterfly units, has become more popular recently. It offloads compute-intensive tasks from general-purpose processors by dismissing FP concerns (e.g., scaling and overflow/underflow). However, the major downside of FP butterfly is its slowness in comparison with its fixed-point counterpart. This reveals the incentive to develop a high-speed FP butterfly architecture to mitigate FP slowness. This brief proposes a fast FP butterfly unit using a devised FP fused-dot-product-add (FDPA) unit, to compute AB + CD ± E, based on binary-signed-digit (BSD) representation. The FP three-operand BSD adder and the FP BSD constant multiplier are the constituents of the proposed FDPA unit. A carry-limited BSD adder is proposed and used in the three-operand adder and the parallel BSD multiplier so as to improve the speed of the FDPA unit. Moreover, modified Booth encoding is used to accelerate the BSD multiplier. The synthesis results show that the proposed FP butterfly architecture is much faster than previous counterparts but at the cost of more area.

Index Terms—Binary-signed digit (BSD) representation, butterfly unit, complex number system, fast Fourier transform (FFT), floating-point (FP), redundant number system, three-operand addition.

I. INTRODUCTION

Fast Fourier transform (FFT) circuitry consists of several consecutive multipliers and adders over complex numbers; hence an appropriate number representation must be chosen wisely. Most of the FFT architectures have been using fixed-point arithmetic, until recently that FFTs based on floating-point (FP) operations grow [1], [2]. The main advantage of FP over fixed-point arithmetic is the wide dynamic range it introduces; but at the expense of higher cost. Moreover, use of IEEE-754-2008 standard [3] for FP arithmetic allows for an FFT coprocessor in collaboration with general purpose processors. This offloads compute-intensive tasks from the processors and leads to higher performance.

The main drawback of the FP operations is their slowness in comparison with the fixed-point counterparts. A way to speed up the FP arithmetic is to merge several operations in a single FP unit, and hence save delay, area, and power consumption [2]. Using redundant number systems is another well-known way of overcoming FP slowness, where there is no word-wide carry propagation within the intermediate operations.

A number system, defined by a radix r and a digit-set [α, β], is redundant iff β – α + 1 > r [4].

Manuscript received February 20, 2014; revised July 14, 2014, October 22, 2014, and December 2, 2014; accepted December 3, 2014. Date of publication June 15, 2015; date of current version February 23, 2016. This work was supported by the Natural Sciences and Engineering Research Council of Canada.

The authors are with the Department of Electrical and Computer Engineering, University of Saskatchewan, Saskatoon, SK S7N 5B5, Canada (e-mail: amir.kaivani@usask.ca; seokbum.ko@usask.ca).

Digital Object Identifier 10.1109/TVLSI.2015.2437999

II. PROPOSED BUTTERFLY ARCHITECTURE

The FFT could be implemented in hardware based on an efficient algorithm [5] in which the N-input FFT computation is simplified to the computation of two (N/2)-input FFT. Continuing this decomposition leads to 2-input FFT block, also known as butterfly unit.

The proposed butterfly unit is actually a complex fused-multiply–add with FP operands. Expanding the complex numbers, Fig. 1 shows the required modules.

The conversion, from nonredundant, to a redundant format is a carry-free operation, however, the reverse conversion requires carry-propagation [4]. This makes redundant representation more useful where many consecutive arithmetic operations are performed prior to the final result.

This brief proposes a butterfly architecture using redundant FP arithmetic, which is useful for FP FFT coprocessors and contributes to digital signal processing applications. Although there are other works on the use of redundant FP number systems, they are not optimized for butterfly architecture in which both redundant FP multiplier and adder are required. The novelties and techniques used in the proposed design include the following.

1) All the significands are represented in binary signed-digit (BSD) format and the corresponding carry-limited adder is designed.
2) Design of FP constant multipliers for operands with BSD significands.
3) Design of FP three-operand adders for operands with BSD significands.
4) Design of FP fused-dot-product-add (FDPA) units (i.e., AB ± CD ± E) for operands with BSD significands.

The rest of this brief is organized as follows. Section II is devoted to the proposed FDPA butterfly architecture while the evaluations are discussed in Section III. Finally, the conclusion is drawn in Section IV.
The exponents of all the inputs are assumed and represented in two's complement (after subtracting the bias), while the significands of \( A, A_{im}, B, \) and \( B_{im} \) are represented in BSD. Within this representation every binary position takes values of \([-1, 0, 1]\) represented by one negative-weighted bit (negabit) and one positive-weighted bit (posibit).

The carry-limited addition circuitry for BSD numbers is shown in Fig. 2, where capital (small) letters symbolizes negabits (posibits). The critical path delay of this adder consists of three full-adders.

The proposed FDPA consists of a redundant FP multiplier followed by a redundant FP three-operand adder.

### A. Proposed Redundant Floating-Point Multiplier

The proposed multiplier, likewise other parallel multipliers, consists of two major steps, namely, partial product generation (PPG) and PP reduction (PPR). However, contrary to the conventional multipliers, our multiplier keeps the product in redundant format and hence there is no need for the final carry-propagating adder.

The exponents of the input operands are taken care of in the same way as is done in the conventional FP multipliers; however, normalization and rounding are left to be done in the next block of the butterfly architecture (i.e., three-operand adder).

#### 1) Partial Product Generation:

The PPG step of the proposed multiplier is completely different from that of the conventional one because of the representation of the input operands \((B, W, B', W')\). Moreover, given that \( W_{re} \) and \( W_{im} \) are constants [5], the multiplications in Fig. 1 (over significands) can be computed through a series of shifters and adders.

With the intention of reducing the number of adders, we store the significand of \( W \) in modified Booth encoding [4].

Given the modified Booth representation of \( W_{re} \) and \( W_{im} \), one PP, selected from multiplicand \( B \), is generated per two binary positions of the multiplier \( W \), as shown in Table I.

Fig. 3 shows the required circuitry for the generation of PP\(_i\), based on Table I where each PP consists of \((n + 1)\) digits (i.e., binary positions).

#### 2) Partial Product Reduction:

The major constituent of the PPR step is the proposed carry-limited addition over the operands represented in BSD format. This carry-limited addition circuitry is shown in Fig. 2 (two-digit slice).

Since each PP (PP\(_i\)) is \((n + 1)\)-digit \((n, \ldots, 0)\) which is either \( B \) \((n - 1, \ldots, 0)\) or \( 2B \) \((n, \ldots, 1)\), the length of the final product may be more than \(2n\).

### B. Proposed Redundant Floating-Point Three-Operand Adder

The straightforward approach to perform a three-operand FP addition is to concatenate two FP adders which leads to...
high latency, power, and area consumption. A better way is to use fused three-operand FP adders [6], [7].

In the proposed three-operand FP adder, a new alignment block is implemented and CSA–CPA are replaced by the BSD adders (Fig. 2). Moreover, sign logic is eliminated.

The bigger exponent between $E_X$ and $E_Y$ (called $E_{\text{Big}}$) is determined using a binary subtractor ($\Delta = E_X - E_Y$); and the significand of the operand with smaller exponent ($X$ or $Y$) is shifted $\|\Delta\|$-bit to the right. Next, a BSD adder computes the addition result ($\text{SUM} = X + Y$), using the aligned $X$ and $Y$.

Adding third operand (i.e., $\text{SUM} + A$) requires another alignment. This second alignment is done in a different way so as to reduce the critical path delay of the three-operand adder. First, the value of $\Delta_3 = E_{\text{Big}} - E_A + 30$ is computed which shows the amount of right shifts required to be performed on extended $A$ (with the initial position of 30 digits shifted to left). Fig. 6 shows the alignments implemented in the proposed three-operand FP adder.

Next, a BSD adder adds the aligned third significand (58-digit) to SUM (33-digit) generated from the first BSD adder. Since the input operands have different number of digits, this adder is a simplified 58-digit BSD adder.

Next steps are normalization and rounding, which are done using conventional methods for BSD representation [8], [9]. It should be noted that the leading zero detection (LZD) block could be replaced by a four-input leading-zero-anticipation [2] for speed up but at the cost of more area consumption. The other modification would replace our single path architecture with the dual path to sacrifice area for speed.

The proposed FP three-operand adder is implemented as shown in Fig. 7 in which new alignment and addition blocks are introduced. Moreover, due to the sign-embedded representation of the significands (i.e., BSD), a sign logic is not required.

A comparison of the proposed design with the conventional one is shown in Table II.

The critical path of the three-operand adder, according to Fig. 7, consists of two 8-bit carry-propagating subtractors (0.25 ns each), a MUX (0.07 ns), a 30 block (0.17 ns), a barrel shifter (0.29 ns), and the final BSD adder (0.16 ns); plus normalization and rounding (0.75 ns) and registers (0.22 ns).

The conversion, from nonredundant, to any redundant format is a carry-free operation, however, the reverse conversion requires carry-propagation.

Given that the proposed butterfly architecture is meant to be used in an FFT unit, the reverse conversion is done in the very last iteration of the FFT unit. Therefore, the latency of each stage is equal to that of a butterfly unit plus those of registers. Moreover, a lookup table is required to store the constant values of 256 twiddle factors.

There might be a need for one more step in the end to convert BSD result to nonredundant representation. This step, if not fused by other FP operations, adds an extra cycle to the whole FFT unit.

It should be noted that the output of the proposed butterfly unit is represented in redundant format and is fed to the next stage (in FFT) without being converted to nonredundant representation.

### III. Evaluations and Comparison

The proposed design is synthesized by Synopsys design compiler using the STM 90-nm CMOS standard library [10] for 1.00 VDD and 25 °C temperature in which an FO4 latency is 45 ps and the area of a NAND2 is 4.4 μm².

The critical path delay of the proposed butterfly architecture, equal to that of FDPA, consists of a constant multiplier, a three-operand FP adder plus registers (Table III).

The total consumed area of the proposed butterfly unit is evaluated as 375,347 μm² of which 8337 μm² is for registers.

The major works on FP butterfly architecture are [1] and [11]. However, Sohn and Swartzlander, Jr., [2] have proposed a very fast FP dot-product unit which can be used in the design of a high-performance butterfly unit. Replacing the dot-product unit of [1] with this one faster, one leads to a high-speed butterfly architecture.

Table IV shows the comparison of the proposed butterfly architecture with those of the fastest previous works. As a result, the proposed design, simulated in 90 nm (versus 45 nm), is yet much faster than those of previous works. Moreover, scaling the area of the proposed design to 45-nm technology results in the value of about $(\sqrt{375,347/2}) = 93,836 \mu m^2$ which is almost equal to that of [11].
For the sake of fair comparison, we also synthesized our design using 45-nm Nangate Open Cell Library with 1.25 VDD. We got the area of our design as 56,338 $\mu m^2$ with wire load model: 5K_hvratio_1_1 and the delay constraint set to 3.15 ns (i.e., equal to that of the fastest previous works).

### IV. Conclusion

We proposed a high-speed FP butterfly architecture, which is faster than previous works but at the cost of higher area. The reason for this speed improvement is twofold: 1) BSD representation of the significands which eliminates carry-propagation and 2) the new FDPA unit proposed in this brief. This unit combines multiplications and additions required in FP butterfly; thus higher speed is achieved by eliminating extra LZD, normalization, and rounding units.

Further research may be envisaged on applying dual-path FP architecture to the three-operand FP adder and using other redundant FP representations. Moreover, use of improved techniques in the termination phase of the design (i.e., redundant LZD, normalization, and rounding) would lead to faster architectures, though higher area costs are expected.

### REFERENCES