FLOATING-POINT BUTTERFLY ARCHITECTURE BASED ON MULTI OPERAND ADDERS

S.Gowthami\textsuperscript{1} K.Prasad Babu\textsuperscript{2} S.Ahmed Basha\textsuperscript{3} K.Sudhakar\textsuperscript{4}

\textsuperscript{1}MTECH DSCE, Dept of ECE, ST.Johns College of Engineering and Technology, Kurnool, Andhra Pradesh, India.
\textsuperscript{2,3}Assistant Professor, Dept of ECE, ST. John's College of Engineering and Technology, Kurnool, Andhra Pradesh, India.
\textsuperscript{4}H.O.D, Associate Professor, Dept of ECE, ST. John's College of Engineering and Technology, Kurnool, Andhra Pradesh, India.

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, since many signal processing applications need high throughput more than low latency. The FFT function consists of consecutive multiply add operations over complex numbers, dubbed as butterfly units. This paper is concerned with the efficient floating-point implementation of the butterfly units that perform the computations in FFT processors. It offloads compute-intensive tasks from general-purpose processors by eliminating FP concerns (e.g., scaling and overflow/underflow). However, the major disadvantage of FP butterfly is its latency in comparison with its fixed-point counterpart. This reveals the incentive to develop a high-speed FP butterfly architecture to reduce FP slowness. This brief proposes a fused floating-point operations and applies them to the implementation of fast Fourier transform (FFT) processors. The fused operations are a two-term dot product and an add-subtract unit, FP fused-dotproduct-add (FDPA) unit, to compute \(AB \pm CD \pm E\), 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 speed up the BSD multiplier. The synthesis results verify that the proposed FP butterfly architecture has more speed than previous counterparts.

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. Floating-point arithmetic provides a wide dynamic range, freeing special purpose processor designers from the scaling and overflow/underflow concerns that arise with fixed-point arithmetic. Use of the IEEE-754 standard 32-bit floating-point format also facilitates using the fast Fourier transform (FFT) processors as coprocessors in collaboration with general purpose processors. Representations of floating-point data in the binary interchange formats are encoded in \(k\) bits in the following three fields ordered as shown
in Figure: a) 1-bit sign S b) w-bit biased exponent E= e+bias c) (t=p−1)-bit trailing significand field digit string T=d1 d2…dp−1; the leading bit of the significand, d0, is implicitly encoded in the biased exponent E. 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 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.

IEEE-754-2008 standard

FLOATING-POINT arithmetic provides a wide dynamic range, freeing special purpose processor designers from the scaling and overflow/underflow concerns that arise with fixed-point arithmetic. Use of the IEEE-754 standard 32-bit floating-point format also facilitates using the fast Fourier transform (FFT) processors as coprocessors in collaboration with general purpose processors.

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. The floating-point fused dot product unit is useful for a wide variety of digital signal processing (DSP) applications including complex multiplication and fast Fourier transform (FFT) and discrete cosine transform (DCT) butterfly operations. In order to improve the performance, a new alignment scheme, early normalization, a four-input leading zero anticipation (LZA), a dualpath algorithm, and pipelining are applied. 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 $[\alpha, \beta]$, is redundant iff $\beta - \alpha + 1 > r$. Fig. 1. FFT butterfly architecture with expanded complex numbers.

The conversion, from nonredundant, to a redundant format is a carry-free operation, however, the reverse conversion requires carry propagation. 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., $AB \pm CD \pm E$) for operands with BSD significands.
II. Existing System

To demonstrate the utility of the Fused DP and Fused AS units for FFT implementation, FFT butterfly unit designs using both the discrete and the fused units have been made. First, radix-2 decimation in frequency FFT butterfly was designed. It is shown in Fig. 2. All lines carry complex pairs of 32-bit IEEE-754 numbers and all operations are complex. The complex add, subtract, and multiply operations shown in Fig. 3 can be realized with a discrete implementation that uses two real adders to perform the complex add or subtract and four real multipliers and two real adders to perform the complex multiply. The complete butterfly consists of six real adders and four real multipliers as shown in the figure. In this and the following figure, all lines are 32-bits wide for the IEEE-754 single-precision data. Alternatively, as shown in Fig. 4, the complex add and subtract can be performed with two fused add-subtract units (marked as FAS in the figure) and the complex multiplication can be realized with two fused dot product units (marked as FDP). The area of the required standard cells and the worst-case delay for the radix-2 butterfly units are shown in Table 4. These results are obtained from having done a complete layout for the butterfly unit. Thus, the areas are greater than the sum of the parts (i.e., four multipliers and six adders for the discrete version or two Fused DP and two Fused AS units for the fused version). The disparity is due to the clock distribution and I/O circuits. Similarly, the delays are less than the sum of the delays of the parts. This is because the conditions that produce the worst-case delay for one part are different from the conditions that produce the worst-case delay for other parts.

III. Proposed System

The FFT could be implemented in hardware based on an efficient algorithm 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.

Fig. 5. BSD adder (two-digit slice).

According to Fig. 1, the constituent operations for butterfly unit are a dot-product (e.g., $B_{re}W_{im} + B_{im}W_{re}$) followed by an addition/subtraction which leads to the proposed FDPA operation (e.g., $B_{re}W_{im} + B_{im}W_{re} + A_{im}$). Implementation details of FDPA, over FP operands, are discussed below.

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 $W$ and $W$ 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 $W$, represented in modified Booth encoding. The critical path delay of this adder consists of three full-adders.

The exponents of all the inputs are assumed and represented in two's complement (after subtracting the bias), while the significands of $A_{re}$, $A_{im}$, $B_{re}$, 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. 5, where capital (small) letters symbolize 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.

### 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 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 $W$ and $W$ 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 $W$, represented in modified Booth encoding. Given the modified Booth representation of $W$ and $W$, one PP, selected from multiplicand $B$, is generated per two binary positions of the multiplier $W$, as shown in Table I. Fig. 6 shows the required circuitry for the generation of PP 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. 5 (two-digit slice).

Since each PP (PP) 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 $2n$. 
Assuming that the sign-embedded significands of inputs \( A \) and \( B \) (24 bits) are represented in BSD; while that of \( W \) is represented in modified Booth encoding (25 bits), the last PP has 24-(binary position) width (instead of 25), given that the most significant bit of \( W \) is always 1. The reduction of the PPs is done in four levels using 12 BSD adders. Given that \( B \) is in \( \pm [1, 2) \) and \( W \) in \( [1, 4) \), the final product is in \( \pm [1, 4) \) and would fit into 48 binary position (47...0). Consequently, positions 45 down to 0 are fractions. Similar to standard binary representation, Guard (G) and Round (R) positions are sufficient for correct rounding. Therefore, only 23 + 2 fractional binary positions of the final product are required to guarantee the final error <2−23. Selecting 25 binary positions out of 46 fractional positions of the final product dismisses positions 0 to 20. However, the next step addition may produce carries to G and R positions. Nevertheless, because of the carry-limited BSD addition, contrary to standard binary addition, only positions 20 and 19 may produce such carries. In overall, positions 0 to 18 of the final product are not useful and hence a simpler PPR tree is possible. Fig. 7 shows the required digits passed to the three-operand adder.

### 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. In the proposed three-operand FP adder, a new alignment block is implemented and CSA–CPA are replaced by the BSD adders (Fig. 5). 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 \( A = E_{\text{Big}} - E_A + 30 \) 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
Fig. 9 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. It should be noted that the leading zero detection (LZD) block could be replaced by a four-input leading-zero-anticipation 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. 10 in which new alignment and addition blocks are introduced. Moreover, due to the sign-embedded presentation of the significands (i.e., BSD), a sign logic is not required.

**Results:**

**Adder:**

**Simulation Results:**

The written Verilog HDL code is simulated by using ISim simulator and results are shown in below fig.

**Synthesis Result:**

The written Verilog HDL code is synthesized by using Xilinx ISE and results are shown in below figs.
Design Summary:

<table>
<thead>
<tr>
<th>Device Utilization Summary (estimated values)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic Utilization</td>
<td></td>
</tr>
<tr>
<td>Number of Gates</td>
<td>64</td>
</tr>
<tr>
<td>Available</td>
<td>465</td>
</tr>
<tr>
<td>Utilization</td>
<td>3%</td>
</tr>
<tr>
<td>Number of FlipFlops</td>
<td>335</td>
</tr>
<tr>
<td>Available</td>
<td>9312</td>
</tr>
<tr>
<td>Utilization</td>
<td>3%</td>
</tr>
<tr>
<td>Number of Slices</td>
<td>128</td>
</tr>
<tr>
<td>Available</td>
<td>222</td>
</tr>
<tr>
<td>Utilization</td>
<td>53%</td>
</tr>
</tbody>
</table>

Timing Report:

Multiplier:

Simulation Result:

Synthesis Results:

The written Verilog HDL code is synthesized by using XilinxISE and results are shown in below figs.

RTL Schematic:
Design Summary:

Timing Report:

<table>
<thead>
<tr>
<th>Data Path:</th>
<th>Delay</th>
<th>Gate</th>
<th>Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell1-Cell2</td>
<td>0.120</td>
<td>0.120</td>
<td>B_2</td>
</tr>
<tr>
<td>USET:1</td>
<td>0</td>
<td>0</td>
<td>0.629</td>
</tr>
<tr>
<td>USET:1</td>
<td>0</td>
<td>1</td>
<td>0.149</td>
</tr>
<tr>
<td>Total</td>
<td>5.76ns (6.87ns logic, 5.67ns route)</td>
<td></td>
<td>(5.46 logic, 1.24 route)</td>
</tr>
</tbody>
</table>

Extension Work:

Double Precision:

In case of double precision 64 bits format, Mantissa is represented in 52 bits, 1 bit is added to the MSB for normalization, Exponent is represented in 11 bits which is biased to 1023 and MSB of double precision is reserved for sign bit as shown below

<table>
<thead>
<tr>
<th>1BIT</th>
<th>11BIT</th>
<th>52BIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGN</td>
<td>EXPONENT</td>
<td>MANTISSA</td>
</tr>
</tbody>
</table>

IEEE Format for double precision

The value of the floating point number represented in double precision format is

\[ F = (-1)^S \times 1.M \times 2^{E-1023} \]

Where 1023 is the value of bias in double precision data format. Exponent E ranges between 1 to 2046, and E = 0 and E = 2047 are reserved for special values. The double precision format offers range from \(2^{-1023}\) to \(2^{+1023}\), which is equivalent to \(10^{-308}\) to \(10^{308}\).

IV. CONCLUSION

This paper describes the design of two new fused floating-point arithmetic units and their application to the implementation of FFT butterfly operations. Although the fused add-subtract unit is specific to FFT applications, the fused dot product is applicable to a wide variety of signal processing applications. Both the fused dot product unit and the fused add-subtract unit are smaller than parallel implementations constructed with discrete floating-point adders and multipliers. The fused dot product is faster than the conventional implementation, since rounding and normalization is not required as a part of each multiplication. Due to longer interconnections, the fused add-subtract unit is slightly slower than the discrete implementation. The area of the fused radix-2 butterfly is 35 percent smaller and the latency is 15 percent less than the discrete radix-2 FFT butterfly parallel implementation. The area of the fused radix-4 butterfly is 26 percent smaller and the latency is 13 percent less than the discrete radix-4 FFT butterfly parallel implementation. Both fused butterflies use fewer rounding operations resulting in more accurate results than the discrete approaches. The errors for a 64K point FFT are about 25 percent less for the fused implementations.

REFERENCES