An efficient Radix-4 butterfly structure based on the complex binary number system and distributed arithmetic

Kevin Bowlyn[1], Sena Hounsinou[2], Jordan Tewell[1]
[1]School of Computer Science and Engineering, Faculty of Computer Science and Engineering, Sacred Heart University, Fairfield, United States of America
[2]Department of Computer Science and Cybersecurity, Metro State University, St. Paul, United States of America

Article Info

Article history:

Received Apr 23, 2024
Revised Aug 12, 2024
Accepted Aug 20, 2024

Keywords:

Complex binary number system
Distributed arithmetic
Fast Fourier transform
Look-up table
Radix-4 FFT
SystemVerilog
VHDL

ABSTRACT

Complex number arithmetic is pivotal in various applications, requiring the selection of an efficient multiplier for high-performance computations. Fast Fourier transform (FFT)-based multipliers are widely employed for computing complex number products, but their reliance on using dedicated multipliers and treating the real and imaginary parts as two entities significantly add to the cost and complexity of the system. Distributed arithmetic (DA) is a technique that replaces complex multiplications with a bit-level shift and addition mechanism. The complex binary number system (CBNS) utilizes binary arithmetic, which treats the real and imaginary parts as a single entity, which can simplify complex number arithmetic and computations. This paper introduces an approach integrating the CBNS with DA in a Radix-4 decimation in time FFT 8-bit and 16-bit butterfly structure. The proposed design significantly reduces arithmetic computations and eliminates dedicated multipliers, demonstrating a reduction in power consumption, area size, and lookup tables, as well as increasing overall clock performance compared to the conventional FFT architecture on Artix-7, Kintex-7, and Virtex-7 field-programmable gate array chips.

Corresponding Author:

Kevin Bowlyn
School of Computer Science and Engineering, Faculty of Computer Science and Engineering, Sacred Heart University
West Campus 3135 Easton Turnpike, Fairfield, Connecticut 06825, United States of America
Email: bowlynk@sacredheart.edu

1. INTRODUCTION

Complex inner products play a crucial role in various computational fields. In the realm of digital signal processing (DSP), they have extensive applications in digital filtering. This fundamental operation underpins multiple domains, including data transmission, speech processing, imaging processing, video processing, and more. The fast Fourier transform (FFT) algorithm has been used in various signal processing applications [1]–[3], utilizing complex number arithmetic operations to efficiently compute the Fourier transform. A well-known problem with the FFT algorithm is that these operations rely heavily on the use of dedicated multipliers, which are fast but occupy a large volume of hardware and are costly to implement, as each complex multiplier comprises four real multipliers and two real adders/subtractors. A disadvantage is that as the number of input bits increases, so do the hardware resources needed to implement such circuitry. A second known problem is that DSP applications often rely on data represented as complex numbers. Typically, these numbers are treated as having two distinct components (real and imaginary), which are processed separately. This process of computing complex numbers as two separate components has been used widely in various implementations of the FFT algorithm, such as those using the divide-and-conquer approach. However, there is a growing interest in developing a more efficient and unified representation of complex numbers to improve the performance of the FFT algorithm. Therefore, space, cost, and power are additional factors that limit DSP applications in resource-constrained systems, such as embedded or internet of things (IoT) devices.

A possible solution is incorporating the distributed arithmetic (DA) and complex binary number system (CBNS) approach. DA [4], [5] is an approach that provides an optimized technique to multiply numbers without relying on the costly circuitry required by multipliers for DSP applications on field-programmable gate arrays (FPGAs). In contrast to the conventional FFT algorithm, DA only requires one dedicated multiplier. A DA block typically consists of a shift register unit, a DA based unit, and an adder/shifter unit. The shift register unit is utilized to compute a series of partial products. Each partial product is fed to the DA based unit, which uses a lookup table (LUT) and memory resources to generate pre-computed partial results based on the input provided by the shift register unit. Each partial output is then accumulated in the adder/shifter unit, starting from the input's least significant bit (LSB) to its most significant bit (MSB). The CBNS approach is an alternative number representation system in which complex numbers are expressed and processed as a single entity using binary digits. Jamil [6] has developed various techniques to facilitate the execution of arithmetic operations using the CBNS to allow the conversion of complex numbers to and from the complex binary base $(-1+j)$. A fractional decimal number can be expressed in CBNS as (1):

$$F = r_i = 2^{-i} f_i = 2^{-1} f_1 + 2^{-2} f_2 + 2^{-3} f_3 + .. \quad (1)$$

where 2^{-1} is equivalent to $(-1+j)^{-1}$, r_i is the remainder of the fraction form, and f_i, represents the binary coefficients of the CBNS base, which is either 0 or 1. To find the value of each coefficient f_i, the steps described below are followed until the remainder r_i becomes zero or when the limit of the non-terminating fractional number has been attained [6]:

Step 1: If $[2 \times r_0] - 1 < 0$ then $f_1 = 0$ and $r_1 = [2 \times r_0]$; else $f_1 = 1$ and $r_1 = [2 \times r_0] - 1$
Step 2: If $[2 \times r_i] - 1 < 0$ then $f_{i+1} = 0$ and $r_{i+1} = [2 \times r_i]$; else $f_{i+1} = 1$ and $r_{i+1} = [2 \times r_i] - 1$

Combining the CBNS with DA could potentially offer a highly efficient approach for hardware implementations, particularly in optimizing FFT operations. The CBNS is particularly well-suited for hardware implementations, such as on FPGAs, where binary operations can be efficiently realized in hardware compared with standard base (octal, decimal, and hexadecimal) arithmetic operations, as a binary system has only two digits: 0 and 1. Additionally, by leveraging the parallelism and simplicity of DA LUT operations, DA can further reduce computational complexity and power consumption compared to standard arithmetic operations. Thus, applying the CBNS coupled with DA (referred to as DA-CBNS herein) in the context of FFT optimization presents an intriguing prospective solution for enhancing the efficiency and efficacy of FFT implementations. DA has been explored to optimize finite impulse response (FIR) filters to reduce area cost and power consumption while still preserving processing speed [7], where each complex number is computed as two separate entities. The CBNS approach has also been explored in the discrete Fourier transform (DFT) algorithm, resulting in an enhanced performance matrix regarding delay and power consumption [8]. Therefore, by incorporating the CBNS and DA approach, the CBNS approach will allow for complex numbers to be computed as a single entity instead of two. The integrated approach will result in a 100% reduction in the multiplier architecture structure as no dedicated multipliers are used to compute the Radix-4 butterfly structure, potentially decreasing area size, power consumption, and cost.

While other works have successfully utilized FPGAs or special processors to operate on the DA-based approach [9]–[16] and/or the CBNS approach [8], [17]–[23], very few works have examined the use of the CNBS and DA together. Bowlyn and Botros [24] have used the CBNS and DA to create a multiplier less design to compute complex products. In their work, the CBNS was used to reduce complex numbers to single units instead of pairs, and a DA LUT was used to store the pre-computed coefficients. The authors claimed this approach significantly reduced arithmetic calculations, real adders, and power consumption (when implemented in a 3-tap filter) compared with the conventional approach to computing complex dot products. This work has been extended to Radix-2 butterfly structures [25] and in computing the Fourier transform with a Radix-2 FFT algorithm [26].

In this paper, an FPGA-based implementation of a Radix-4 DA-CBNS butterfly structure is proposed and is evaluated over a conventional Radix-4 implementation in terms of power consumption, area size, LUTs, and overall clock performance. The contributions can be summarized as follows:
– Design of a Radix-4 butterfly structure using an optimized DA and CBNS design.
– Implementation of the DA-CBNS structure on the Artix-7, Kintex-7, and Virtex-7 FPGA platforms.
– Comparative study of an 8-bit and a 16-bit DA-CBNS design vs. a "conventional" (non-optimized) decimation-in-time (DIT) Radix-4 butterfly design.

The remainder of the paper is organized as follows: section 2 provides a detailed description of the proposed implementation of the DA-CBNS Radix-4 butterfly structure design. Section 3 presents a comparison between the proposed FPGA-based DA-CBNS butterfly structure and the conventional implementation along with the results, followed by a discussion. Finally, we summarize our findings in section 4.

2. METHOD

2.1. CBNS arithmetic

In the context of the FFT, complex arithmetic plays a crucial role in computing the FFT algorithm efficiently. Complex arithmetic in the FFT typically utilizes algorithms and data structures optimized for complex number operations to ensure the FFT's high computational efficiency. This efficiency is essential for the use of FFT in signal processing, data analysis, and various other applications where efficient frequency domain analysis is required.

Traditionally, complex numbers in binary are represented with separate real and imaginary parts. However, with the CBNS approach, complex numbers can be represented as a single entity using the $(-1+j)$ CBNS-based algorithm. For example, the complex number (0.70703125+j0.70703125) can be represented in CBNS as 1110.1110011001101110 in the $(-1+j)$ base as a single entity, compared to the conventional, binary representation of (00.10110101, 00.10110101) in base-2, where the real and imaginary are separated.

In designing the CBNS arithmetic circuit, the CBNS algorithm was further studied. Table 1 provides a comprehensive overview of how addition and subtraction are performed using the CBNS approach with the $(-1+j)$ base and shows the truth table for computing a CBNS adder and subtractor circuit. Inputs A and B represent the binary input values to the CBNS adder and subtractor circuit. For addition, the Carries column indicates the carry bits generated after carrying over the values to the next significant bit, and the Sum column shows the final result after considering the carry bits. For subtraction, the Borrow column indicates the borrowed bits generated after borrowing from a higher bit, and the Difference column shows the final result of the subtraction operation after considering the borrowed bits.

Table 1. Truth table for $(-1+j)$-based CBNS addition and subtraction

Input A	Input B	Addition output A+B		Addition output A+B	
		Carries	Sum	Borrow	Difference
0	0	000	0	0000	0
0	1	000	1	1110	1
1	0	000	1	0000	1
1	1	110	0	0000	0

2.2. CBNS adder unit design

In designing the CBNS adder unit, a slightly modified version of the traditional base-2 ripple carry adder was used. In the traditional 2's complement binary approach, adding two ones $(1 + 1)$ results in 10 (base-2), where the first bit is the sum bit, and the second is the carry bit. The rules for the CBNS adder for base $(-1+j)$, however, are different. Adding 1 + 1 results in 1100 $(-1+j)$ [18], where the first bit is the sum bit (Stage 0), and the remaining three bits are the carry bits (stages 1 - 3). In other words, with the CBNS adder approach, adding two ones always generates two carries of one to stages 2 and 3 of the CBNS adder, while Stage 1 will have no carry as its carry is always zero. Therefore, the adder unit was designed and structured into three different stages:

– Stages 0 and 1: Implemented using two half adders as no carry-ins were needed.
– Stage 2: Implemented using a full adder, as the carry-in from stage 0 is propagated to stage 2.
– Stage 3: implemented using a four-input bit adder, which includes the carry-ins from stages 1 and 2

Extended carries occur when more than three ones are added together. For example, adding four ones in stage 3 produces an output of 1 1101 0000, which represents 4 in base $(-1+j)$. In this case:

– The first bit is the sum bit.
– The second, third, and fourth bits are the carry bits.
– The remaining five bits are the extended carries.

The propagation approach to handle extended carries is as follows. The carry bits (bits three and four) are propagated to stages 5 and 6, as these stages handle the less significant carries. The extended carry bit (1) is propagated to stages 7, 9, 10, and 11, as these stages handle the more significant extended carries. Stages 4 and 8 will always have carry-ins and extended carries of zero and can be ignored (bits two and six), as they do not influence the carry and extended carry propagation. This approach was built using the respective adder sizes for each stage respectively and simulated in very high-speed integrated circuit (VHSIC) hardware description language (VHDL) using for loops. Figure 1 shows a gate-level schematic design of an 8-bit CBNS adder.

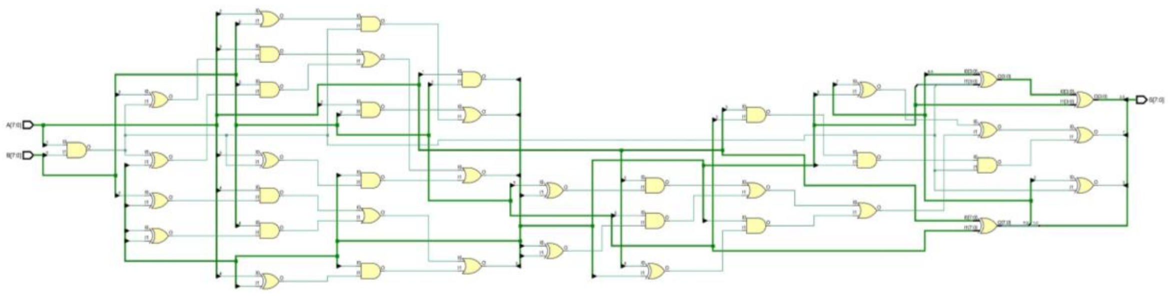


Figure 1. 8-bit CBNS adder gate level design

2.3. CBNS subtractor unit design

The CBNS subtractor unit operates similarly to a standard base-2 algorithm, with the key difference occurring in the subtraction of 1 from 0. In the $(-1+j)$ base algorithm, subtracting 1 from 0 yields three additional carriers of 1s refer to Table 1. Specifically, 0 – 1 = 11101 [27]. The LSB bit represents the difference output, while the remaining four bits represent the carries. Since the carry following the difference term is always zero, it can be ignored. When performing subtraction, the subtrahend is subtracted from the minuend, and the resulting carries are added to the minuend bits using the CBNS adder algorithm.

The design of the subtractor unit is similar to a modified traditional ripple borrow subtractor. However, it was noted that the borrow and adder bit could be OR'd or XOR'd simultaneously. Therefore, if there is a borrow, the adder signal will be zero (indicating no addition), and conversely, if there is an addition, the borrow signal will be zero. The subtractor unit incorporates both the subtrahend and minuend bits in one single block, which also includes the CBNS adder algorithm. This procedure was implemented using multiple for loops in a cascading ripple circuit to compute the output values at different stages. Figure 2 shows the gate-level design for an 8-bit CBNS subtractor.

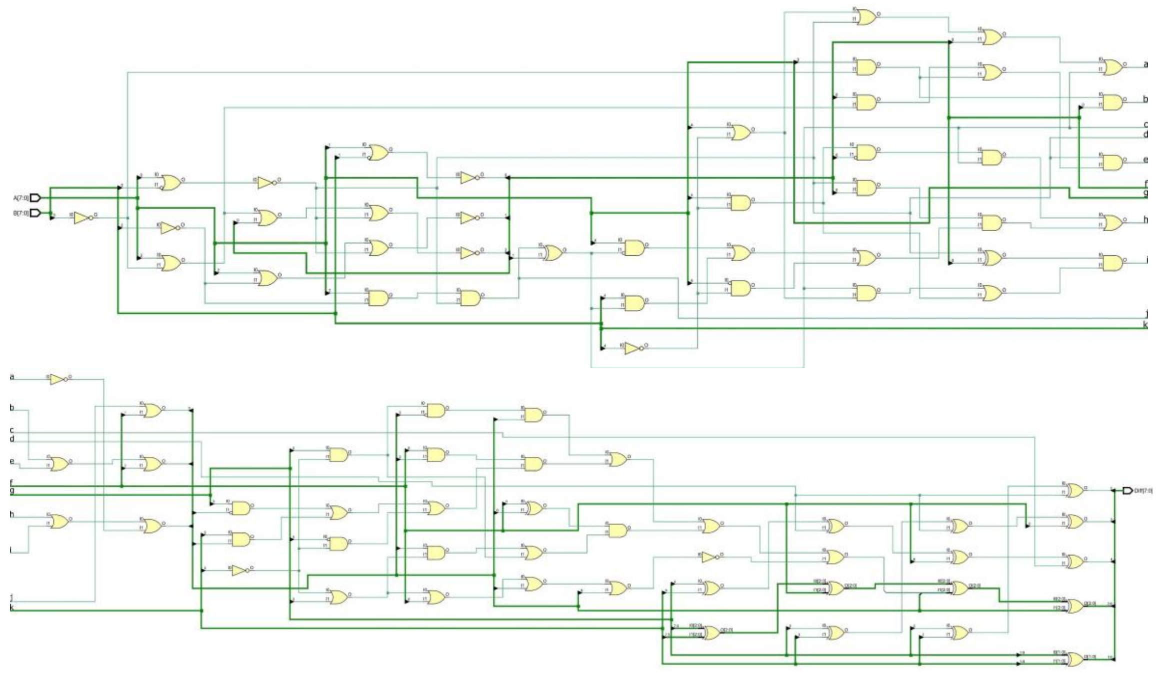


Figure 2. 8-bit subtractor gate-level design circuit

2.4. DA-CBNS multiplier unit design

DA is highly beneficial in FFT computations, especially for computing inner dot products, and it eliminates the need for explicit multipliers, reducing hardware complexity and resource usage. The main idea behind DA is to precompute all possible partial products and store them in LUTs, which can then be quickly retrieved and summed using simple arithmetic operations. This approach is widely embraced for implementing FIR filters, as DA can efficiently utilize LUTs, shifters, and adders to compute the essential sum of products necessary for FIR filters. Additionally, DA lowers power consumption by minimizing complex operations, enabling efficient, high-performance, and power-efficient digital signal processing. Thus, by optimizing the use of LUTs in FPGAs, DA enhances efficiency and leverages its inherent parallelism and pipelining capabilities to significantly boost throughput and performance.

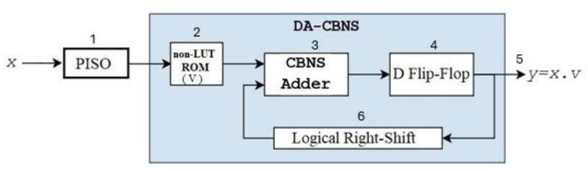
The overall block diagram of the conventional DA algorithm [24] operates as follows. First, the inputs are shifted in parallel-in-to-serial-out (PISO) fashion to the shift register unit. The output of the shift register unit is then serially output one bit at a time into the read-only memory (ROM)-LUT (DA Unit). The ROM-LUT directly addresses the memory of the shift register outputs. These values are then added, stored in an accumulator, and shifted so that the final output, $y[n]$, is available after the N-bit clock cycle. A combinational adder/subtractor control signal T_s is used with the DA structure to manipulate signed bit binary numbers. When T_s is zero, addition occurs until the MSB is reached, at which point T_s becomes one, and subtraction follows.

The proposed DA-CBNS multiplication circuit leverages both the DA and CBNS techniques. Unlike the conventional ROM-based structure, which speeds up multiplication through pre-computed values in the ROM-LUT, the proposed DA-based design uses a non-LUT ROM component to store the constant twiddle factor values rather than a ROM bank. This modification results in a logic gate-based implementation for the DA-LUT, leading to a slight increase in gate count but a 100% reduction in memory usage.

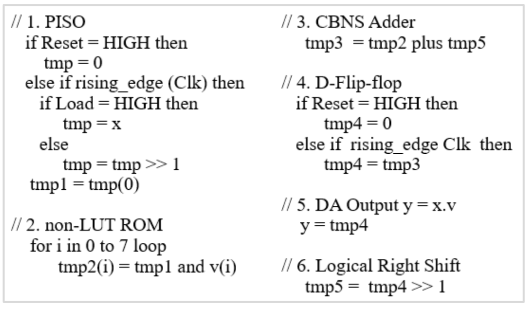
Another improvement involves adjusting the DA structure. The conventional DA-based design typically operates on binary signed numbers, using an arithmetic right shift to preserve the product's sign during the accumulation phase. However, an arithmetic right shift is unsuitable for our DA-CBNS design since a leading 0 or 1 does not indicate the sign of numbers in the complex $(-1+j)$-base. As a solution, we replaced the arithmetic right shift with a logical right shift.

Furthermore, the DA structure's conventional binary adder/subtractor was substituted with an implementation suitable for the $(-1+j)$-based CBNS adder structure. The conventional binary adder/subtractor, while functional, can lead to inefficiencies due to the increased number of arithmetic computations needed to separate and combine the real and imaginary parts of complex numbers. The $(-1+j)$-based CBNS adder structure, on the other hand, represents the output sum as a single compact entity, reducing the number of such computations.

Figures 3(a) and 3(b) shows the block diagram of the DA-CBNS multiplier block and its pseudocode. The original input operand x is represented in the $(-1+j)$-base, and the resulting DA product is also expressed in this base as $y = x.v$. The non-LUT ROM stores the constant fixed-point twiddle factor v and by using the CBNS technique, this design enables complex multiplication by shifting and adding without using the divide and conquer approach, where the real and imaginary parts are computed separately.



(a)



```
// 1. PISO
  if Reset = HIGH then
     tmp = 0
  else if rising_edge (Clk) then
     if Load = HIGH then
        tmp = x
     else
        tmp = tmp >> 1
  tmp1 = tmp(0)

// 2. non-LUT ROM
  for i in 0 to 7 loop
     tmp2(i) = tmp1 and v(i)

// 3. CBNS Adder
     tmp3 = tmp2 plus tmp5

// 4. D-Flip-flop
   if Reset = HIGH then
        tmp4 = 0
   else if rising_edge Clk then
        tmp4 = tmp3

// 5. DA Output y = x.v
   y = tmp4

// 6. Logical Right Shift
     tmp5 = tmp4 >> 1
```

(b)

Figure 3. The block diagram of the DA-CBNS (a) DA-CBNS multiplier block diagram and (b) pseudocode

2.5. Radix-4 implementation design

The FFT algorithm is a fast algorithm that computes the DFT and its inverse. The Radix-4 FFT algorithm offers computational advantages over simpler Radix-2 algorithms for large transform sizes because it reduces the number of required arithmetic operations. However, it may not be as memory efficient as Radix-2 algorithms for smaller transform sizes. Overall, the Radix-4 FFT algorithm is a key component in many signal processing applications where fast and efficient Fourier transform computation is required, such as in audio processing, image processing, telecommunications, and more.

DIT and decimation in frequency (DIF) are two FFT approaches for efficiently computing the DFT algorithm. The Radix-4 DIT FFT decomposes the DFT in the time domain using a base-4 approach, involving recursive division of the input sequence into smaller sub-sequences of length four until each is a single sample. This method uses Radix-4 butterflies to minimize operations and reduce the number of required twiddle factors, enhancing memory access and computational efficiency. Conversely, the Radix-4 DIF FFT decomposes the DFT in the frequency domain, splitting it into even and odd frequency components of length four, progressing from the lowest to highest frequencies. It also utilizes Radix-4 butterflies, optimizing arithmetic operations and memory access.

The Radix-4 DIF FFT generally performs better due to fewer operations and more regular memory access patterns, making it suitable for parallel processing. However, the Radix-4 DIT FFT is preferred for specific specialized applications or hardware environments where a time-domain approach is advantageous. A key computation difference between DIT and DIF is the order of operations: DIT performs multiplication first, followed by addition, while DIF performs addition first, followed by multiplication as shown in Figure 4. The conventional calculation for a Radix-4 DIT/DIF butterfly structure requires twelve real multiplications and twenty-two real additions [28] for each structure. This includes cases of repeated multiplications. In our proposed design, we chose to implement a DIT FFT structure as applications of DIT FFT in the time domain are prevalent in various fields due to its efficiency in handling time-domain data such as audio, image, and speech processing where data are represented as two separate entities, real and imaginary. We aim to implement our technique in designing a DA-CBNS DIT FFT structure to perform an application in image processing where complex numbers are treated as a single entity.

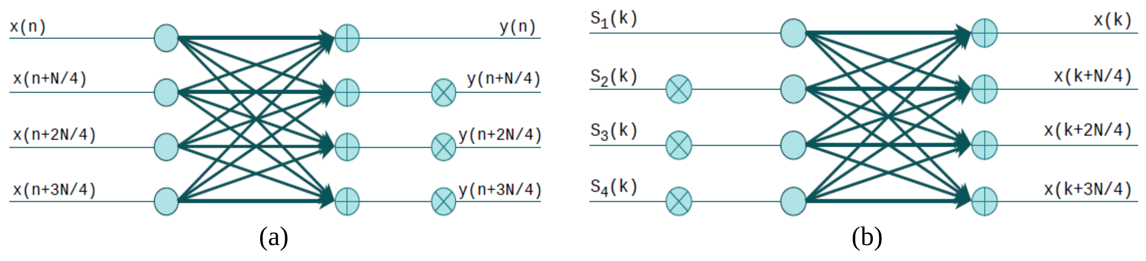


Figure 4. Radix-4 DIF and DIT 4-point butterfly structures

Theoretically, each Radix-4 butterfly structure employs three complex multiplications at each butterfly stage, with each complex multiplication involving four real multiplications and two real additions. Additionally, each butterfly structure at each stage also requires eight real complex additions, as one complex addition entails two real additions. From (2) (DIT) and (3) (DIF), the following matrix coefficients M can be obtained from each of the equation terms where k and $n = 0, 1, \ldots, N/4 - 1$.

$$\begin{aligned}
X(k) &= S_1(k) + W_N^{k} S_2(k) + W_N^{2k} S_3(k) + W_N^{3k} S_4(k)\\
X\left(k+\tfrac{N}{4}\right) &= S_1(k) - jW_N^{k} S_2(k) - W_N^{2k} S_3(k) + jW_N^{3k} S_4(k)\\
X\left(k+\tfrac{2N}{4}\right) &= S_1(k) - W_N^{k} S_2(k) + W_N^{2k} S_3(k) - W_N^{3k} S_4(k)\\
X\left(k+\tfrac{3N}{4}\right) &= S_1(k) + jW_N^{k} S_2(k) - W_N^{2k} S_3(k) - jW_N^{3k} S_4(k)
\end{aligned} \tag{2}$$

$$\begin{aligned}
y(n) &= \left\{x(n) + x\left(n+\tfrac{N}{4}\right) + x\left(n+\tfrac{2N}{4}\right) + x\left(n+\tfrac{3N}{4}\right)\right\} W_N^{0}\\
y\left(n+\tfrac{N}{4}\right) &= \left\{x(n) - jx\left(n+\tfrac{N}{4}\right) - x\left(n+\tfrac{2N}{4}\right) + jx\left(n+\tfrac{3N}{4}\right)\right\} W_N^{n}\\
y(n+(2N)/4) &= \left\{x(n) - x\left(n+\tfrac{N}{4}\right) + x\left(n+\tfrac{2N}{4}\right) - x\left(n+\tfrac{3N}{4}\right)\right\} W_N^{2n}\\
y(n+(3N)/4) &= \left\{x(n) + jx\left(n+\tfrac{N}{4}\right) - x\left(n+\tfrac{2N}{4}\right) - jx\left(n+\tfrac{3N}{4}\right)\right\} W_N^{3n}
\end{aligned} \tag{3}$$

$$M = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \tag{4}$$

Using M, we obtain the expressions for the outputs A', B', C', and D' as (5):

$$\begin{aligned} A' &= A + BW_b + CW_c + DW_d \\ B' &= A - jBW_b - CW_c + jDW_d \\ C' &= A - BW_b + CW_c - DW_d \\ D' &= A + jBW_b - CW_c - jDW_d \end{aligned} \tag{5}$$

By regrouping and rearranging the terms in the expressions of (5), we can identify specific terms that appear in more than one of the expressions. Specifically, the term $(A + CWc)$ is common to both A' and C', $(A - CW_c)$ is used in B' and D', $(BW_b + DW_d)$ is present in both A' and C', while the term $(BW_b - DW_d)$ (factoring out the j term) shows up in B' and D'. Therefore, it is possible to reduce the load on computational resources by reusing the results of these terms rather than recomputing them as shown in Figure 5.

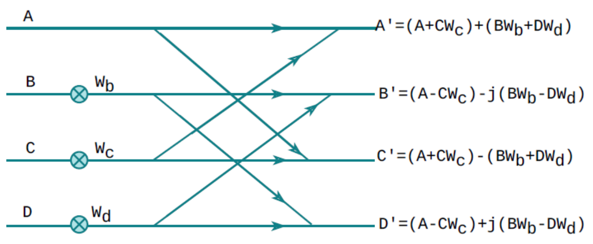
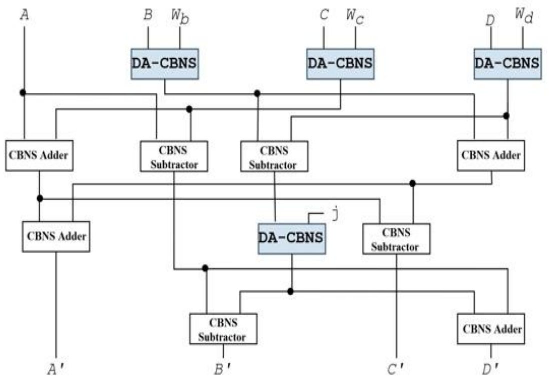
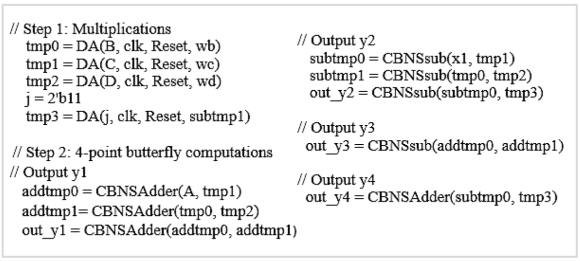

A
B
C
D
W_b
W_c
W_d
$A'=(A+CW_c)+(BW_b+DW_d)$
$B'=(A-CW_c)-j(BW_b-DW_d)$
$C'=(A+CW_c)-(BW_b+DW_d)$
$D'=(A-CW_c)+j(BW_b-DW_d)$

Figure 5. Radix-4 DIT butterfly structure: partial sums are rearranged to improve the butterfly computation

In our proposed DA-CBNS approach, four DA-CBNS multiplier blocks and twelve CBNS adders/subtractors are essential for computing each butterfly structure. Eight of the CBNS adders and subtractors are allocated to calculating the overall Radix-4 outputs, and the remaining four CBNS adders (DA-CBNS multiplier block) are dedicated to the DA-based structure as shown in Figure 6(a). This DA-CBNS block diagram implementation incorporates the regrouping and rearranging of terms for computing the output results of A', B', C', and D' as illustrated in Figure 5. The algorithm as shown in Figure 6(b) illustrates the pseudocode for computing an 8-bit 4-point DA-CBNS butterfly design as a top model design by using port-based module instantiation of the required design blocks, which include the DA, the CBNS adder, and the CBNS subtractor blocks.



(a)



```
// Step 1: Multiplications
   tmp0 = DA(B, clk, Reset, wb)
   tmp1 = DA(C, clk, Reset, wc)
   tmp2 = DA(D, clk, Reset, wd)
   j = 2'b11
   tmp3 = DA(j, clk, Reset, subtmp1)

// Step 2: 4-point butterfly computations
// Output y1
  addtmp0 = CBNSAdder(A, tmp1)
  addtmp1= CBNSAdder(tmp0, tmp2)
  out_y1 = CBNSAdder(addtmp0, addtmp1)

// Output y2
  subtmp0 = CBNSsub(x1, tmp1)
  subtmp1 = CBNSsub(tmp0, tmp2)
  out_y2 = CBNSsub(subtmp0, tmp3)

// Output y3
 out_y3 = CBNSsub(addtmp0, addtmp1)

// Output y4
 out_y4 = CBNSAdder(subtmp0, tmp3)
```

(b)

Figure 6. DA-CBNS-based Radix-4 FFT implementation: (a) Block diagram of the butterfly computation using CBNS and DA technique and (b) pseudocode for generating the butterfly outputs

To improve this overall design and structure, each input of the butterfly structure ($X(k)$, $X(k+N/4), X(k+2N/4)$ and $X(k+3N/4)$, where k ranges from 0 to $(N/4-1)$) must initially undergo conversion into the $(-1+j)$-base. Figure 7 shows the overall implementation process to compute the Radix-4 butterfly structure using our proposed DA-CBNS technique. To implement this structure, the fixed-point binary complex numbers and the twiddle factors (which transition into fixed constant coefficient values) undergo an initial conversion into the $(-1+j)$-base representation. Following this conversion, the transformed input data serve as the inputs $X(k), X(k+N/4), X(k+2N/4)$ and $X(k+3N/4)$.

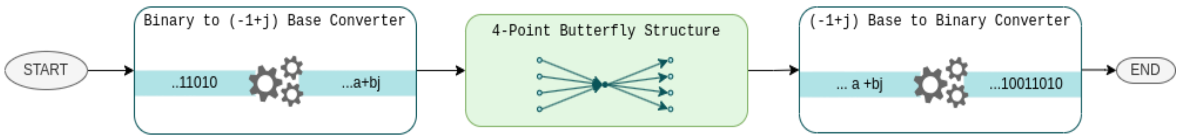

START
Binary to (-1+j) Base Converter
...11010
...a+bj
4-Point Butterfly Structure
(-1+j) Base to Binary Converter
... a +bj
...10011010
END

Figure 7. Proposed implementation procedure for computing a Radix-4 butterfly structure

These datasets then become the driving data values for the 4-point butterfly structure. In computing the butterfly structure, the twiddle factor is loaded into the LUT. The DA structure subsequently computes partial products with equal shifts, adding them before summing to the next partial product shift. This iterative process continues for a number of cycles equal to *n* (the length of the input bits) until $y[n]$ encapsulates the final result. The final outputs are generated in the $(-1+j)$-base representation and are later reconverted into their base-2 formats.

The Radix-4 algorithm reduces the number of required arithmetic operations compared with the Radix-2 algorithm. Table 2 shows the total number of real adders and multipliers for the conventional Radix-4 DIT implementation using only one butterfly structure, where each complex multiplication and addition are computed as two separate entities. Note that the number of multipliers decreases as the number of different butterfly structures used increases. This is contrasted to our DA-CBNS-based design, which shows the total number of real complex multiplications and additions, both with and without the proposed DA technique that computes complex arithmetic as a single entity.

As shown in Table 2, the utilization of the DA structure reduces the number of complex multiplications from $3(N/4)log_4N$ to 0, eliminating the need to use dedicated complex multipliers. However, the total number of complex additions for the DA-CBNS-based design increases significantly due to the extra adders required to compute each of the four DA structures. This includes the three DA multipliers for multiplying the twiddle factor plus the DA multiplication of j for the Radix-4 butterfly structure as shown in Figure 6(a). This results in an increase in complex additions from $8(N/4)log_4N$ to $12(N/4)log_4N$. Nonetheless, the number of complex adders are the same as the dedicated real multipliers needed to compute the conventional Radix-4 butterfly algorithm using the divide-and-conquer approach. Additionally, the DA-CBNS adder/subtractor uses fewer LUT resources than a dedicated multiplier.

Table 2. Comparison of the conventional Radix-4 DIT FFT vs. our proposed DA-CBNS DIT structure with and without the optimized DA-based design

Number of Points	Radix-4 DIT FFT		Radix-4 DIT CBNS FFT and DA-CBNS FFT			
$N=4^P$	Real Multiplications	Real Additions	Complex Multiplications		Complex Additions	
			Without DA	With DA	Without DA	With DA
	$12\left(\frac{N}{4}\right)\log_4 N$	$22\left(\frac{N}{4}\right)\log_4 N$	$3\left(\frac{N}{4}\right)\log_4 N$		$8\left(\frac{N}{4}\right)\log_4 N$	$12\left(\frac{N}{4}\right)\log_4 N$
4	12	22	3	0	8	12
16	96	176	24	0	64	96
64	576	1056	144	0	384	576
256	3072	5362	768	0	2048	3072
1024	15360	28160	3840	0	10240	15360
4096	73728	135168	18432	0	49152	73728
16384	344064	630784	86016	0	229376	344064

3. RESULTS AND DISCUSSION

3.1. Experimental setup

To evaluate the proposed DA-CBNS Radix-4 DIT butterfly structure, we compared its performance and design requirements against those of the conventional Radix-4 structure. We implemented two configurations for each type of structure: one with N=8 bits and the other with N=16 bits. Since the CBNS technique treats complex numbers as a single entity, each DA-CBNS configuration with input size N was compared to an equivalent conventional configuration using N real components and N imaginary components. Our Radix-4 butterfly structure design was implemented and coded in SystemVerilog and VHDL within the Xilinx Vivado Design Suite. It was then synthesized and implemented on three target-size FPGA platforms: Artix-7, Kintex-7, and Virtex-7.

3.2. Evaluation metrics

The assessment included power consumption, LUT utilization, area size, and clock frequency data from the post-place and route results. The clock signal's worst negative slack (WNS) was used to determine the maximum frequency of a design, given by (6):

$$F_{max} = \frac{1}{T-WNS} \tag{6}$$

where T is the target clock period and WNS is the positive worst negative slack of the clock signal in the intra-clock paths section of the timing analysis report. Finally, the area for each design was computed as (7):

$$area = \frac{L_{used}+F_{used}}{L_{total}+F_{total}} \tag{7}$$

where L_{used} and F_{used} represent the number of LUTs and flip flops used in the design, respectively, while L_{total} and F_{total} are the total number of LUTs and flip flops available on the platform, respectively.

3.3. Results

Table 3 shows the results of our design compared across three FPGA target chips: Artix-7, Kintex-7, and Virtex-7. For all chips, the 8-bit Radix-4 DA-CBNS DIT butterfly structure was compared with its 8-bit (real and imaginary) conventional counterpart. The DA-CBNS design exhibited a 77% reduction in area size and an 89% decrease in LUT count. Power consumption rates were also significantly reduced, with a 60% reduction for the Artix-7 chip, 63% for the Virtex-7 chip, and 61% for the Kintex-7 chip. Additionally, the DA-CBNS structure showed an 18% increase in speed for both the Artix-7 and Virtex-7 chips and an overall speed performance increase of 12% for the Kintex-7 chip.

Table 3. 8-bit and 16-bit DA-CBNS and conventional DIT structure statistics using Virtex-7, Artix-7, and Kintex-7 chips

Artix-7 xc7k480tffv1156-3	8-bit butterfly structure		16-bit butterfly structure	
	DA-CBNS (8-bit)	Conventional (8-bit) (8 real + 8 imaginary)	DA-CBNS (16 bit)	Conventional (16-bit) (16 real + 16 imaginary)
IOB/Total	91/500	178/500	179/500	353/500
LUT/Total	167/134600	1501/134600	700/134600	6139/134600
Worst negative slack (ns)	15.010	12.266	15.242	2.632
Frequency (MHz)	100.10	78.53	102.48	44.71
Dynamic power consumption (μW)	18	45	40	139
Area (%)	0.101040	0.442051	0.292472	1.66295
Virtex-7 xc7v585tffg1157-3	8-bit Butterfly Structure		16-bit butterfly structure	
	DA-CBNS (8-bit)	Conventional (8-bit) (8 real + 8 imaginary)	DA-CBNS (16 bit)	Conventional (16-bit) (16 real + 16 imaginary)
IOB/Total	91/600	178/600	179/600	353/600
LUT/Total	167/364200	1501/364200	701/364200	6139/364200
Worst negative slack (ns)	17.363	15.985	16.294	8.638
Frequency (MHz)	130.94	110.93	114.86	61.12
Dynamic power consumption (μW)	17	46	41	146
Area (%)	0.037342	0.163372	0.108182	0.6145589
Kintex-7 xc7k480tffv1156-3	8-bit butterfly structure		16-bit butterfly structure	
	DA-CBNS (8-bit)	Conventional (8-bit) (8 real + 8 imaginary)	DA-CBNS (16 bit)	Conventional (16-bit) (16 real + 16 imaginary)
IOB/Total	91/400	178/400	179/400	353/400
LUT/Total	167/298600	1501/298600	702/298600	6139/298600
Worst negative slack (ns)	15.811	14.710	14.843	6.697
Frequency (MHz)	108.83	97.18	98.45	54.64
Dynamic power consumption (μW)	18	46	42	146
Area (%)	0.045546	0.199263	0.132172	0.749609

Comparable enhancements were observed in the 16-bit scenario. The DA-CBNS design showcased reductions of 82% in area size and 89% in logic gate cost across all three chips. For the Artix-7 and Virtex-7 chips, the DA-CBNS approach featured a 71% reduction in power consumption, while the Kintex-7 chip showed a 68% reduction. The DA-CBNS method also resulted in different clock speeds across all three chips: on the Artix-7 chip, our method demonstrated an improved rate of 129%. On the Kintex-7 chip, it delivered an 88% improvement in speed, and on the Virtex-7 chip, the DA-CBNS design showcased an increase in speed of around 80% compared with the conventional approach. Overall, our findings reveal that the DA-CBNS design significantly economized on memory resources (LUTs), area size, and power consumption while also increasing speed on all three FPGA target chips when compared to the conventional 8-bit and 16-bit DIT butterfly structures.

3.4. Discussion

Our design was compared with the conventional FFT approach regarding area size, speed, LUTs, and dynamic power consumption. For the 8-bit DA-CBNS structure, it used significantly less area size, dynamic power, and LUTs than its conventional counterpart on all three target chips. In terms of speed, the DA-CBNS design also demonstrated faster performance on the Artix-7, Kintex-7, and Virtex-7 chips. For the 16-bit structures, the DA-CBNS design consistently showed the best results across all four parameters, with reductions in area size, dynamic power, and LUTs, as well as an increase in speed compared to the conventional approach using dedicated multipliers. Therefore, adopting the DA-CBNS structure greatly improves the performance of the Radix-4 butterfly structure for both 8-bit and 16-bit structures.

The proposed design was also compared to other existing methodologies. Neuenfeld *et al.* [29] proposed 16-bit Radix-2 and Radix-4 DIT butterfly structures aimed at minimizing the number of arithmetic operators to produce power-efficient designs and reduce the number of real dedicated multipliers in the Radix-4 butterfly. Their results showed LUT usage and dynamic power consumption to be 4,152 and 554.2 µW, respectively. With our proposed design for a 16-bit (single entity) DIT butterfly structure, our LUT usage and dynamic power were found to be 702 and 42 µW on a Kintex-7 target design chip. This resulted in our proposed design having fewer reductions in LUT usage and dynamic power consumption, respectively

Ferreira *et al.* [30] proposed a 16-bit low-power hardware architecture combining an optimized split-Radix-4 DIT butterfly and 5-2 adder compressors (ACs). Their design, composed of eight multipliers and sixteen adders and subtractors, showed a maximum frequency of 55.79 MHz with a gate count of 4960. In contrast, our proposed 16-bit DA-CBNS structure, in which complex numbers are computed as a single entity with no need for dedicated multipliers, showed a gate count of approximately 700 on Kintex-7, Artix-7, and Virtex-7 FPGA chips. The speed of our 16-bit DA-CBNS design was 98.45, 102.48, and 114.86 MHz on Kintex-7, Artix-7, and Virtex-7 chips, respectively. This resulted in an increased clock-to-clock speed of about 76% on the Kintex-7 chip, 83% on the Artix-7 chip, and approximately 106% on the Virtex-7 chip compared to the author's maximum frequency result in [30].

Overall, our proposed DA-CBNS butterfly structure exhibited reductions in area size, LUT usage, and power consumption, and an increase in clock-to-clock performance. The significant reduction in arithmetic operators allowed the Radix-4 DA-CBNS butterfly structure to outperform the existing conventional Radix-4 butterfly structures in terms of area, power consumption, and LUT usage. However, since the DA-CBNS design represents each complex number as a single entity, the absolute precision of our data results is somewhat limited compared to the conventional approach that computes complex numbers as two entities.

4. CONCLUSION

This paper presented a unique design by incorporating the CBNS approach with the DA technique. By incorporating the CBNS technique within our design, the number of arithmetic computations was reduced to three real complex multiplications and eight real complex additions. However, with the DA technique, no dedicated multipliers were utilized, as the DA structure accomplishes multiplications by employing a shift-adder circuit rather than a CBNS multiplier circuit. Therefore, our proposed design utilizes zero multipliers and twelve complex adders. The four additional complex adders are due to the four DA structures used within our design. Two architecture models, 8-bit and 16-bit Radix-4 butterfly structures, were implemented in Xilinx Vivado, coded in SystemVerilog and VHDL, and then implemented on three target FPGA chips: Artix-7, Kintex-7, and Virtex-7.

The results showed that for the 8-bit and 16-bit Radix-4 butterfly structures, the DA-CBNS approach resulted in less power consumption, smaller area size, and fewer LUTs, as well as increased overall clock-to-clock performance speed on all three target FPGA chips compared to the conventional FFT counterpart. Therefore, given the great demand for new and improved technology for DSP applications, our

DA-CBNS structure demonstrates to have better performance regarding area size, LUTs, power consumption, and speed over a conventional FFT structure. By employing and adopting the DA-CBNS technique, we believe our approach can significantly improve the efficiency of floating-point DSP applications that rely on complex arithmetic. Our future goal is to implement our DA-CBNS design on real-time applications employing 16, 64, and 256-point FFTs.

REFERENCES

[1] K. V. Shanbhag and D. Sathish, "Low complexity physical layer security approach for 5G internet of things," *International Journal of Electrical and Computer Engineering*, vol. 13, no. 6, pp. 6466–6475, Dec. 2023, doi: 10.11591/ijece.v13i6.pp6466-6475.

[2] H. A. Ghani, M. R. Abdul Malek, M. F. Kamarul Azmi, M. J. Muril, and A. Azizan, "A review on sparse fast fourier transform applications in image processing," *International Journal of Electrical and Computer Engineering*, vol. 10, no. 2, pp. 1346–1351, Apr. 2020, doi: 10.11591/ijece.v10i2.pp1346-1351.

[3] C. Jittawiriyanukoon and V. Srisarkun, "Evaluation of graphic effects embedded image compression," *International Journal of Electrical and Computer Engineering*, vol. 10, no. 6, pp. 6606–6617, Dec. 2020, doi: 10.11591/ijece.v10i6.pp6606-6617.

[4] G. NagaJyothi and S. SriDevi, "Distributed arithmetic architectures for FIR filters-A comparative review," in *2017 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, Mar. 2017, vol. 24, pp. 2684–2690, doi: 10.1109/wispnet.2017.8300250.

[5] P. Sritha, R. S. Valarmathi, and P. Ramya, "Different distributed arithmetic multiplication schemes used in fir filter," *International Journal of Engineering and Advanced Technology*, vol. 8, pp. 33–36, 2018.

[6] T. Jamil, *Complex binary number system: algorithms and circuits*. Springer India, 2013.

[7] M. Saritha *et al.*, "Pipelined distributive arithmetic-based FIR filter using carry save and ripple carry adder," in *2021 2nd International Conference on Communication, Computing and Industry 4.0 (C2I4)*, Dec. 2021, vol. 12, pp. 1–6, doi: 10.1109/c2i454156.2021.9689396.

[8] A. Shadap and P. Saha, "Discrete fourier transformation processor based on complex Radix (−1 + j) number system," *Engineering Science and Technology, an International Journal*, vol. 20, no. 1, pp. 80–88, Feb. 2017, doi: 10.1016/j.jestch.2016.08.020.

[9] M. Bharathi, G. A. Sai, B. D. Sree, K. Bharadwaj Karthik, B. U. K. Naik, and Y. J. Shirur, "Designing 64-bit LUT based FFT structure for high-speed DSP applications," in *2023 IEEE 12th International Conference on Communication Systems and Network Technologies (CSNT)*, Apr. 2023, pp. 31–34, doi: 10.1109/csnt57126.2023.10134710.

[10] M. Bharathi and Y. J. M. Shirur, "Efficient realization of fast fourier transform based on distributed arithm," *Res Militaris*, vol. 12, no. 5, pp. 923–933, 2023.

[11] B. U. V Prashanth, M. R. Ahmed, and M. R. Kounte, "Design and implementation of DA FIR filter for bio-inspired computing architecture," *International Journal of Electrical and Computer Engineering*, vol. 11, no. 2, pp. 1709–1718, Apr. 2021, doi: 10.11591/ijece.v11i2.pp1709-1718.

[12] Y. Lu, S. Duan, B. Halak, and T. Kazmierski, "A variation-aware design methodology for distributed arithmetic," *Electronics*, vol. 8, no. 1, Jan. 2019, doi: 10.3390/electronics8010108.

[13] K. S. Reddy, S. Madhavan, P. Falkowski-Gilski, P. B. Divakarachari, and A. Mathiyalagan, "Efficient FPGA implementation of an RFIR filter using the APC–OMS technique with WTM for high-throughput signal processing," *Electronics*, vol. 11, no. 19, Sep. 2022, doi: 10.3390/electronics11193118.

[14] B. M, K. Mohanarangam, Y. J. M Shirur, and J. R. Choi, "Accelerating DSP applications on a 16-bit processor: Block RAM integration and distributed arithmetic approach," *Electronics*, vol. 12, no. 20, Oct. 2023, doi: 10.3390/electronics12204236.

[15] C. R. K. J, R. D. Kulkarni, and D. M. A. Majid, "Energy-efficient architecture for high-performance FIR adaptive filter using hybridizing CSDTCSE-CRABRA based distributed arithmetic design: Noise removal application in IoT-based WSN," *Integration*, vol. 97, Jul. 2024, doi: 10.1016/j.vlsi.2024.102172.

[16] P. V Praveen Sundar, D. Ranjith, T. Karthikeyan, V. Vinoth Kumar, and B. Jeyakumar, "Low power area efficient adaptive FIR filter for hearing aids using distributed arithmetic architecture," *International Journal of Speech Technology*, vol. 23, no. 2, pp. 287–296, Mar. 2020, doi: 10.1007/s10772-020-09686-y.

[17] T. Jamil, M. Awadalla, and I. Mohammed, "Nibble-size multiplier circuit designs and their FPGA implementations for complex binary number system," *International Journal of Electrical Engineering and Technology (IJEET)*, vol. 12, no. 6, pp. 105–121, 2021, doi: 10.34218/IJEET.12.6.2021.012.

[18] T. Jamil, M. Awadalla, and I. Mohammed, "Complex binary adder designs and their hardware implementations," *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 7, 2019, doi: 10.14569/ijacsa.2019.0100734.

[19] M. Mukherjee and S. K. Sanyal, "Design of CBNS nibble size adder using pass transistor logic circuit: Extension to FPGA implementation," in *2017 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, Jul. 2017, pp. 1–6, doi: 10.1109/icccnt.2017.8203927.

[20] M. Mukherjee and S. K. Sanyal, "FPGA-based efficient implementation of CBNS computational circuits: a modular approach," in *Computational Advancement in Communication, Circuits and Systems*, Springer Singapore, 2021, pp. 109–123.

[21] M. Mukherjee and S. K. Sanyal, "Design of high-speed FPGA based CASU using CBNS arithmetic: extension to CFFT processor," in *International Conference on Innovative Computing and Communications*, Springer Singapore, 2020, pp. 835–854.

[22] M. Mukherjee and S. K. Sanyal, "2-D systolic array architecture of CBNS based discrete Hilbert transform processor," *Microprocessors and Microsystems*, vol. 87, Nov. 2021, doi: 10.1016/j.micpro.2020.103509.

[23] S. S. Santosh, T. S. Swaroop, T. Kavya, and R. Chinthala, "Complex binary number system-based co-processor design for signal processing applications," in *2021 5th International Conference on Electronics, Materials Engineering &amp; Nano-Technology (IEMENTech)*, Sep. 2021, vol. 1, pp. 1–6, doi: 10.1109/iementech53263.2021.9614893.

[24] K. N. Bowlyn and N. M. Botros, "A novel distributed arithmetic multiplierless approach for computing complex inner products," in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, 2015.

[25] K. N. Bowlyn and N. M. Botros, "A novel distributed arithmetic approach for computing a Radix-2 FFT butterfly implementation," in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, 2017, pp. 59–63.

[26] K. Bowlyn and S. Hounsinou, "An improved distributed multiplier-less approach for Radix-2 FFT," *IEEE Letters of the Computer*

Society, vol. 3, no. 2, pp. 54–57, Jul. 2020, doi: 10.1109/locs.2020.3014354.
[27] T. Jamil, M. Awadallah, and I. Mohammad, "Complex binary subtractor designs and their hardware implementations," *International Journal of Advanced Research in Engineering and Technology*, vol. 12, no. 5, pp. 111–125, 2021, doi: 10.34218/IJARET.12.5.2021.011.
[28] *Software optimization of FFTs and IFFTs using the SC3850 core*. Freescale Semiconductor, 2010.
[29] R. Neuenfeld, M. Fonseca, and E. Costa, "Design of optimized Radix-2 and Radix-4 butterflies from FFT with decimation in time," in *2016 IEEE 7th Latin American Symposium on Circuits and Systems (LASCAS)*, Feb. 2016, vol. 30, pp. 171–174, doi: 10.1109/LASCAS.2016.7451037.
[30] G. Ferreira *et al.*, "Low-power fast fourier transform hardware architecture combining a split-radix butterfly and efficient adder compressors," *IET Computers &amp; Digital Techniques*, vol. 15, no. 3, pp. 230–240, Mar. 2021, doi: 10.1049/cdt2.12015.

BIOGRAPHIES OF AUTHORS

Kevin Bowlyn received a dual B.S. from Eastern Illinois University and Southern Illinois University, Carbondale (SIUC), IL, USA, in engineering cooperative and electrical engineering in 2008. He earned a dual M.S. degree in 2010 in biomedical engineering and electrical and computer engineering, and a Ph.D. in electrical and computer engineering from SIUC in 2017. He is an assistant professor of computer engineering at Sacred Heart University, Fairfield, CT, USA. His research is focused on more efficient, low area-power circuit designs, for computing a fast fourier transform (FFT) algorithm and implementing circuit design on FPGA-based boards. Dr. Bowlyn can be contacted at: bowlynk@sacredheart.edu.

Sena Hounsinou received a B.S. in electrical engineering, M.S. and Ph.D. in electrical and computer engineering from Southern Illinois University Carbondale in 2018. She joined the Department of Electrical Engineering and Computer Science at Howard University as a Postdoctoral fellow in 2019, where she also taught undergraduate Computer Science courses. Her research interests include reconfigurable computing, FPGA-based systems, embedded systems security and cyber-physical systems. She is an assistant professor in the Department of Computer Science and Cybersecurity at Metro State University St. Paul. Dr. Hounsinou can be contacted at: sena.houeto@metrostate.edu.

Jordan Tewell received a B.S. in computer science from Youngstown State University in 2009. He earned a MET degree in entertainment technology from Carnegie Mellon University in 2012 and his Ph.D. from City, University of London in 2018. He joined Sacred Heart University as an assistant professor in the School of Computer Science and Engineering in 2019. His research focus is in human-computer interaction, working at the intersection of HCI with multi-sensory interfaces, wearable technology, and mixed reality. Dr. Tewell can be contacted at: tewellj@sacredheart.edu.