Received September 8, 2021, accepted October 4, 2021, date of publication October 13, 2021, date of current version October 20, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3119598

A Low-Complexity Shifting-Based Conflict-Free Memory-Addressing Architecture for Higher-Radix FFT

SUMIT AGARWAL, SHAIK RAFI AHAMED, ANUP KUMAR GOGOI, AND GAURAV TRIVEDI, (Member, IEEE)
Indian Institute of Technology Guwahati, Guwahati, Assam 781039, India
Corresponding author: Shaik Rafi Ahamed (rafiahamed@iitg.ac.in)

ABSTRACT Conflict-free addressing (CFA) techniques are necessary for Fast Fourier Transform (FFT) hardware. For low radices, well-proven XOR-based addressing architectures are available in the literature. Applications such as wireless communication use higher FFT radices, more processing elements, and a larger number of memory sets for a continuous flow of data. In the existing CFA techniques, the complexity increases with increasing radices or memory sets. In the proposed technique, the higher number of memory sets and radix are leveraged to advantage. A novel scheme is suggested to reduce the complexity using a progressive shifting technique. The mathematical basis of the scheme is derived here and illustrated with an example of 512-point radix-16 FFT. The proposed and existing CFA architectures are designed using Verilog and implemented in the Semiconductor Laboratory, Chandigarh, 180 nm SCL library. The synthesis results show that the proposed scheme achieves a 33% area reduction, 23% lower power consumption, and 39% improvement in execution time performance compared with the existing XOR based schemes. The area reduction factor further improves with a higher number of FFT points.

INDEX TERMS Conflict free addressing, FFT, multiple memory, progressive shift.

I. INTRODUCTION

High-radix Fast Fourier Transform (FFT) processors are in great demand today for handling high-speed systems such as Orthogonal Frequency Division Modulation (OFDM) based communication standards, including Wireless Personal Area Network (WPAN) (IEEE 802.15c) and Wireless High Definition (HD) (IEEE 802.11ad, IEEE 802.11ay), vehicle and military radars, and medical imaging such as Magnetic Resonance Imaging (MRI). These processors are based on different radices of the FFT algorithm, which are radix 2, 4, 8, 16, etc. Higher radices (greater than 4) are needed for higher throughput, especially for FFT sizes of 512 points and above. For example, realizing architectures for 512 point FFT with higher radices (8 or 16) would only require two or three stages compared to nine stages when using radix 2. The existing XOR or modulo-addition schemes for accessing multiple banks of memory in a conflict-free manner work well for lower radices, but for higher radices, the complexity and area consumed increase significantly. Continuous-flow (CF) high-performance FFTs need at least two memories. Additional memories are required to feed more processing elements (PEs) to boost performance. Therefore, in high-performance FFTs, the number of memories is usually two or three or sometimes even more. A Conflict-Free Access (CFA) [1] scheme has to be designed for each memory, leading to a sizeable increase in area. In this paper, the higher number of memories is used to an advantage. When higher radices are used, the number of stages is also limited to approximately 3 to 4. When the number of stages and memories are in the same range, each memory will be tied up with only one or two stages. In such a case, the XOR or modulo addition logic can be replaced with our proposed ‘‘progressive shift’’ (PS) technique, which is much simpler. The progressive shifting technique is not easy to apply in architectures where one memory is used to supply data for more than two stages of the system. In order to demonstrate the advantages of the proposed technique, it is compared with existing techniques, and the significant differences are outlined in Table 1.

The associate editor coordinating the review of this manuscript and approving it for publication was Sun Junwei.

TABLE 1. A comparison of various techniques of conflict free access in FFT algorithms.

Design $\rightarrow$	[1], [2], [3]	Tsai [4]	Huang [5]	Tian [6]	Reisis [7]	Jinti [8]	Ours
Bank logic	XOR	Modulo Addition	XOR	Shifting	Permutation	XOR	Shifting
Radix	2	2^k	16	2^k	2^k	2	2^k
Stages per memory ($N = 512$, $k = 4$)	9	3	3	3	1	9	1
Nos. of memory	1	1	2	1	Same as stages	1	3 or more
Continuous flow	No	No	Yes	No	No	No	Yes

The motivation and main contributions of this paper are highlighted below:

- The progressive shift technique is proposed for FFT architectures with multiple memories and fewer stages serviced by a PE. Such types of FFT architectures usually find application in high throughput FFT designs. The main idea is to reduce the complexity of the CFA circuits, which tend to occupy larger areas as the number of memories increases to improve performance.
- A theoretical framework for the technique is also formulated.
- Progressive shifting and modulo addition/XOR switching are embedded in a single mathematical framework.
- Simulation studies are carried out to show the superiority of the proposed technique in terms of the usage of hardware resources and plotted in the form of a graph.
- To validate the proposed architecture, a chip is fabricated, and its layout is presented in Fig. 8.

The proposed architecture finds applications in OFDM based MIMO communication systems that demand high throughput and low area. The proposed multi-bank memory architecture can be extended to implement 5G communication systems and Internet of Things (IoT) architectures in the future.

II. CONFLICT FREE ADDRESSING

An FFT processor generally uses some form of the Cooley-Tukey algorithm and a butterfly structure for processing data using a PE. The PE needs r inputs and generates r outputs in every clock cycle, where r is the radix of the FFT algorithm. As per the usual requirement of FFT algorithms, data have to be reordered before giving at the input of PE. This reordering is commonly done by first storing the data in a memory and then retrieving it using an address different from the one used during storage. Since one memory can only give one output, to fulfill the need for r simultaneous outputs, r banks are required in a memory. Bank size is taken as N/r so that the total storage in the memory is N, where N is the number of FFT points. The main challenge is the distribution of data to the memory banks such that data required simultaneously by the butterfly are stored in different banks. This problem is called bank generation for CFA and uses switching to reorder the data. The switching circuit is called a forward commutator, which consists of a circuit (BAGU) to generate the necessary bank number and address, and a multiplexer array, which directs the incoming data and address to that bank. Additionally, after the memory, a realignment commutator is needed that realigns the data

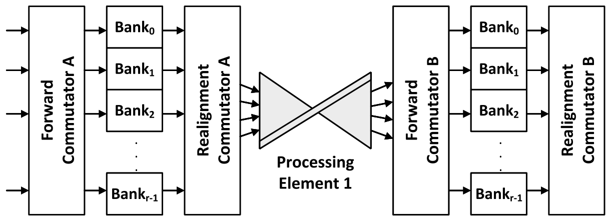


FIGURE 1. A generalized structure for CFA showing switches before and after the memory.

with the inputs of the PE. The commutators use a large number of multiplexers to do this and hence consume a significant area. Each memory will require its own switching circuitry, as shown in Fig. 1. Thus, in FFTs that use multiple memories for real-time continuous flow of data, the area for conflict-free switching increases significantly. Hence, developing a CFA scheme with a lower area is a challenging task that needs to be addressed.

There have been two distinct approaches for CFA in the literature. The first is based on finding the bits in the data index, which would help in identifying a bank for it [2], [3], [9]. However, all these schemes explored the radix-2 architecture. To achieve higher throughputs, in [4], a high radix scheme was proposed using parallel butterflies. On similar lines, [10] derived schedules for CFA. The scheme was elaborated for the case where multiple butterflies operated in parallel in the same stage. It also took pipelining delays into account to take care of folded or pipelined butterflies and hence, different read and write times for a set of data. In [11], the work of [4] was extended, and CFA schemes for the cases of both prime factor FFT and common factor FFT were presented. Various constraint sets were derived and based on them, banks and addresses were generated using XOR or modulo-r addition.

In all these cases, the bank number is generated using XOR or modulo addition logic for a particular data index. It is difficult to decipher any definite pattern. This randomness necessitates movement of not only data but also "row address" from some input port of a switch to some other output port, resulting in a complex switching mechanism. The other addressing scheme was designed by Reisis and Vlassopoulos [7], where the concepts of forward permutation and reverse permutation were proposed. However, rules or guidelines for a hardware efficient permutation were not discussed anywhere. Additionally, it assumed an independent memory for each stage. It dealt with the particular case of one memory per stage, and no solution was provided for the case when a memory fed more stages. Hence, this permutation technique has limited applications. Hsiao and Lee [12] focused on modulo addition and suggested a form of permutation but did not go into any study on its applications or theory. Chen *et al.* [13] used a similar technique for the output stage for the simple case of power-of-two radix implementation, but they did not deal with other stages. For real FFT realization, Ma in [14] suggested a procedure to fill up the entire memory bank sequentially rather than filling up multiple banks simultaneously. Memory bandwidth usage was inefficient, as other banks were unused during the input or output phases. This inefficiency was later taken care of in [15] at the cost of extra exchange stages to attain a bit-order as required for CF. The CFA scheme from the classic work of Johnson [3] was used. Hence, these works did not truly add anything new to CFA, although they came up with a more efficient architecture for real valued FFTs. In [16], an XOR based scheme was derived based on the postulates and the basic principles as suggested by Johnson with some modification. The design in [8] proposed a CFA for real FFT. The scheme was explained for a 32-point FFT. The derivation of the CFA scheme was not provided. The implementation used counters to store data depending on the count, but in effect, the switching complexities remained similar. In [17], mixed radix algorithms were discussed. The design considered only a single memory without banks. Address generation was discussed to obtain the operands. However, the architecture did not use parallelism and was a low-throughput structure. Long [18] reduced the complexity of the CFA circuits developed by [19], which in turn was an improvement of [3]. The advantage of this design was the lower complexity of implementation compared to [3]. However, the design was limited to radix-2 and radix-4; additionally, the complexity increased when the number of memories or PE was increased. The work in [20] was on a flexible number of FFT points. They used a single PE with 16-way parallelism. The CFA scheme was challenging, as the PE had to cater to a large variety of points. The number of memories used for CF was three. The presented scheme was directly based on [12] and used a sophisticated switching mechanism for data and addresses. In [6], a storage pattern that initially looked similar to progressive shifting was suggested. However, it was inefficient because it used two multibank memories to realize CFA, whereas two memories could be used to realize both CFA and CF simultaneously [21]. In [22], Wang followed a new trend of using single port memories. The CFA challenge increased tremendously in this design, which was solved using modulo addition techniques. Although the design saved a significant area in memory by replacing the dual port with a single port, the actual advantage was offset due to the use of more complicated CFA circuits.

In order to overcome the above mentioned limitations, in this paper, we propose a progressive shifting permutation, which is applicable for cases where one or two stages are fed by one memory set. The proposed technique avoids the complicated XOR based switching mechanism presented earlier. Moreover, as the shifting mechanism in progressive shifting is regular, shifting of addresses is eliminated, thus making the switch simpler. Hence, the proposed scheme is more efficient. It may be extended by combining with the recently proposed single port memories of [22] to further improve chip efficiency.

TABLE 2. Mathematical symbols used in the paper.

Symbol	Represents
N	Number of FFT points
r	Radix of the FFT algorithm
B	Number of banks in a memory
D	Data-width
S	Total number of stages
q	$log_2 N$

III. PROPOSED BANK GENERATION SCHEME FOR CFA USING PROGRESSIVE SHIFTS

Hsiao and Lee [12] outlined a procedure for bank generation based on modulo addition of the characteristic variable of each stage. Following a similar approach, a method is proposed to determine bank generation with additional help from Lemmas 1 and 2. For ease of reading, a table of important symbols is provided in Table 2.

For a N point FFT, let the factors of N be as

$$N = N_1 N_2 N_3 \dots N_S \tag{1}$$

where each factor defines a stage and S is the total number of stages. We define the characteristic variable n_i of each stage $i = 1, 2, \dots S$:

$$n_1 = 0 \dots N_1 - 1, \quad n_2 = 0 \dots N_2 - 1 \; etc. \tag{2}$$

The input equation of an FFT algorithm is expressed as

$$n = Xn_1 + Yn_2 + Zn_3 + \dots \tag{3}$$

where the constants X, Y, Z, etc. are defined as

$$\begin{aligned} X &= N_2 \, N_3 \dots N_S \\ Y &= N_3 \, N_4 \dots N_S \\ Z &= N_4 \, N_5 \dots N_S \end{aligned}$$

The last constant would be 1. This is the method of the common factor algorithm. For the prime factor algorithm, it is difficult to apply the progressive shifting method, as the flow of data is complicated owing to modulo operations [12], [23]).

Lemma 1: The input indices to a butterfly have a one-digit difference among them. It is this digit that is used to decide the banks. Here, the digits are $n_1, n_2 \dots n_S$.

Proof: A stage of FFT is defined by the $n_i's$. Hence, for a particular stage, all other $n_i's$ will remain constant except for the stage in question. Thus, the fundamental definition of stage ensures that for each stage, the different digit is the one that controls the stage. Furthermore, it seems logical to use this digit to differentiate banks.

Lemma 2: If a memory is feeding several stages, all the corresponding stage variables must be taken into account while deciding the bank. If any variable is left out, the corresponding stage cannot be fed by the memory in a conflict-free manner.

Proof: Let only a single-stage variable be used to find the bank. Then, $Bank = n_1$ ensures that for the first stage, the data are in conflict-free banks. For any other stage, there will be at least one situation where all other $n_i's$ are the same, and only the stage variable is different. However, since this variable is not taken into account, the bank for those inputs is guaranteed to be the same, leading to conflict. Hence, all variables have to be considered while determining the banks for a stage, as in (4).

$$Bank = f(n_1, n_2, \ldots n_s) \tag{4}$$

Theorem 1 (Progressive Shifting): If only one of the $n_i's$ is changed in the bank generation function, it will lead to the CFA scheme of progressive shifting. This shall happen when the function is addition, and it is conducted modulo B, where B is the number of banks, as shown in (5).

It should be noted that usually $B \geq r$, where r is the radix of the FFT algorithm. In the case of $B > r$, instead of continuous progressive shifts, shifts may occur every other cycle or even after more cycles.

$$Bank = (n_1 + n_2 + \ldots n_s) \, mod \, B \tag{5}$$

When only one- or two-stage variables are changed, the flow in which data have to be stored follows a simple pattern leading to simplifications in switching commutators. In (4), the digits n_1, n_2, etc. are added together to find the banks. For a stage, only one digit is supposed to be different; this ensures that the required inputs go to different banks. In higher-radix algorithms, the banks are large in number, while the number of stages comes down drastically. Hence, the interaction among the stages does not severely affect the flow of data. Additionally, in cases where each stage has its own memory [7] for demanding throughputs or in input or output memories [13], there is virtually no interaction among the stages. Under such circumstances, the switching complexity may be reduced considerably using progressive shifts, as derived below.

The derivation of the technique is based on (5). If a memory has to feed only one stage, only one stage variable will increase by 1. Thus, the bank for data changes only by 1 unit. This means that the data are stored in the banks in a progressively shifting manner. The data required simultaneously for a PE can be shown in a row of a table as in Table 3. Thus, the progressive shift represents increasing the shift amount by one for every consecutive or alternate row. The modulo operation ensures that the data are shifted circularly in a progressive fashion so that the required data are directly stored in different columns, as shown in Table 3. The corresponding hardware is proposed in section III-A.

TABLE 3. Progressive shifts (0, r, $2r$ . . . are the required data indices for the first butterfly operation).

$Bank_0$	$Bank_1$	$Bank_2$			$Bank_{r-1}$
0	1	2	3	...	$r-1$
$2r-1$	r	$r+1$	$r+2$	...	$2r-2$
...					
...	...	...	...	...	$N-1$

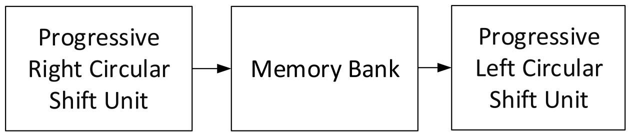

Progressive Right Circular Shift Unit
Memory Bank
Progressive Left Circular Shift Unit

FIGURE 2. Progressive forward and reverse circular shift units.

A. HARDWARE FOR PROGRESSIVE SHIFTING

In this scheme, the banks to which the data are to be shifted are known a priori. Hence, they can be stored in a series of registers, and the permutations can be easily derived by appropriately shifting the stored bank values. Then, the registers can be hardwired to the bank selection logic, as shown in Fig. 3. This design is more straightforward than the other technique where the bank values have to be made available at any of the multiplexers. The two main components of the proposed architecture are the progressive circular right shift unit (PCRSU) and progressive circular left shift unit (PCLSU), as shown in Fig. 2. PCRSU is placed before the memory bank. Its purpose is to increase the input data shift right by 1 whenever a shift is required. A shift may be required after every clock cycle, alternate cycles, or even after a few more cycles. The maximum shift amount is r, assuming a radix r butterfly.

To achieve circular progressive switching, multiplexer banks and a shift register array (SRA) of data-width $D = log_2 r$ and length r are used. The multiplexer bank consists of r multiplexers, each with r data inputs of width d (d is the width of the data) and a select input of size D. The r data values from the previous stage are connected to each of the r inputs of all the data multiplexers, as shown in Fig. 3. The select logic is designed using SRA. It is preloaded with values $0, 1, 2, \ldots, r-1$. The outputs of the r registers of SRA are connected to the r select inputs of the data multiplexers. On every clock cycle or as required, the data in SRA are shifted by one register. At the start, the selected inputs of data multiplexers are $0, 1, 2, \ldots, r-1$. This leads to data output without any shifts. Another clock edge now leads to the shifting of addresses in the register array to the right in a circular fashion. Hence the selected inputs now become $r-1, 0, 1, 2, \ldots, r-2$. Now, the data output of the PCRSU is as if shifted right by one unit. The multiple clock cycle logic for activating the shifting in the SRA can be realized using a counter of an appropriate size. This is demonstrated by considering a 3-bit address as an example. The shift in the output of the multiplexers with respect to the change in the addresses of the SRA is shown in Table 4. An example of radix 4 continuous right progressive shifting for 32-point FFT of [21] is shown in Tables 5 and 6. PCLSU is placed after

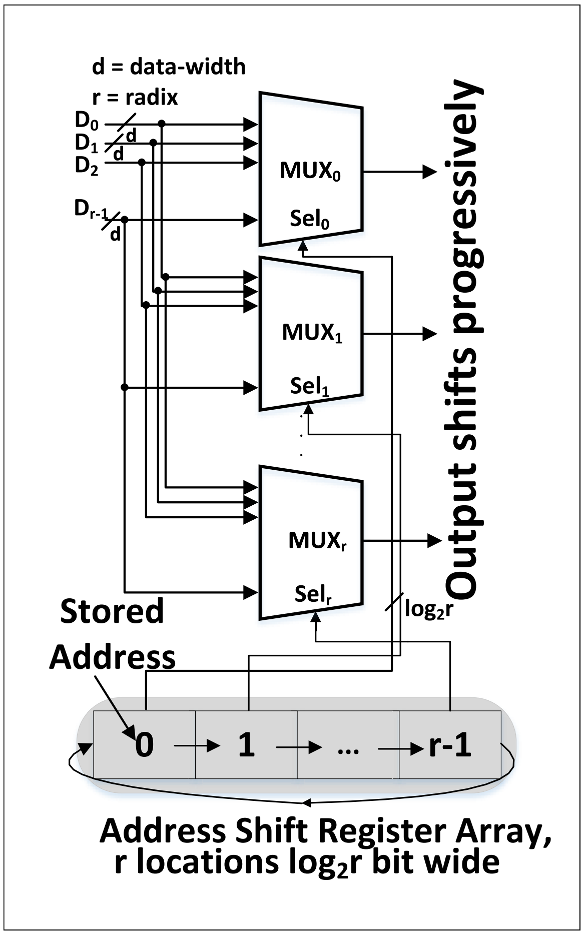


FIGURE 3. Right progressive shift unit.

TABLE 4. Output of the multiplexers with respect to the change in data of the shift register unit.

Shift register unit					Output of multiplexer				
a_{r-1}	a_{r-2}	a_{r-3}	...	a_0	1^{st}	2^{nd}	3^{rd}	...	r^{th}
7	6	5		0	d_0	d_1	d_2	...	d_7
6	5	4		7	d_7	d_0	d_1		d_6
5	4	3		6	d_6	d_7	d_0		d_5
...					...				
0	7	6		1	d_1	d_2	d_3		d_0

the memory bank to align the right-shifted data in line with the inputs of the PE. The design of this unit is similar to that of the PCRSU. Only the direction of the shift is changed to be left circular.

IV. BANK GENERATION USING XOR LOGIC

In [3], it is suggested that the banks for radix-2 FFT can be differentiated on the basis of parity of the data indices. A derivation is given based on the simultaneous solutions of the conditions for in-place memory access. Here, the Bank is derived with the help of equations formulated for progressive shifting so as to show the relation between the two techniques.

For radix 2 decomposition, the digits n_i's of (5) can be replaced by bits b_i's.

$$Bank = b_1 + b_2 + b_3 \ldots + b_S \tag{6}$$

TABLE 5. Original digit-reversed output sequence for 32 point FFT if stored serially [21].

$Bank_0$	$Bank_1$	$Bank_2$	$Bank_3$
0	8	16	24
4	12	20	28
1	9	17	25
5	13	21	29
2	10	18	26
6	14	22	30
3	11	19	27
7	15	23	3

TABLE 6. Shifted digit-reversed output data indices for the output bank for 32 point FFT.

$Bank_0$	$Bank_1$	$Bank_2$	$Bank_3$
0	8	16	24
28	4	12	20
17	25	1	9
13	21	29	5
26	2	10	18
22	30	6	14
11	19	27	3
7	15	23	3

Single-bit addition is equivalent to an XOR operation. The result of XOR defines the parity of the data index. It is thus shown that the two methods are related by a common mathematical framework. The differentiating factor is the constraints imposed on the number of b_i's that are allowed to change simultaneously.

V. A CASE STUDY: CFA FOR 512 POINT RADIX-16 FFT

A 512 point FFT chip was designed in Huang [5] using a radix-16 algorithm. The architecture used two radix-16 stages and a radix-2 stage. In our design, an additional PE is introduced in the pipeline for stage 2 processing to double the throughput. The new architecture is shown in Fig. 4. The design is thus modified so that the input memory feeds only PE_1, and there is a second memory to feed PE_2. An output memory is finally used to obtain the outputs in the natural order. This circuit is aptly suited for the progressive shifting technique of CFA. The hardware for this design consists of several subunits, as described in III-A. These are detailed in Table 7. For XOR-based addressing in higher and mixed radix FFTs, the results of [4] can be used. The equation derived by Huang is reproduced in (7) for reference:

$$\begin{aligned} &Bank \\ &= (8(b_5 \oplus b_1) + 4b_0 + 4b_8 + ((b_8b_7)_2 + 2b_6 + (b_4b_3)_2 \\ &\quad + (b_2b_1)_2 + (2b_5 - 1)(b_5 \oplus b_1))mod\ 4)mod\ 16 \end{aligned} \tag{7}$$

The bank generation logic defined in (7) shows that it is not possible to identify the bank simply by inspection. In Huang, the circuit was implemented using XOR logic, but details were not presented. We redesign and implement the circuit using Verilog and analyze it for the purpose of comparison with the PS technique. The detailed design of XOR based forward commutator is shown in Fig. 5. The first step is to generate the index of the data using q-bit counters with offset adders, where $q = log_2N$. This data index is then fed to a bank

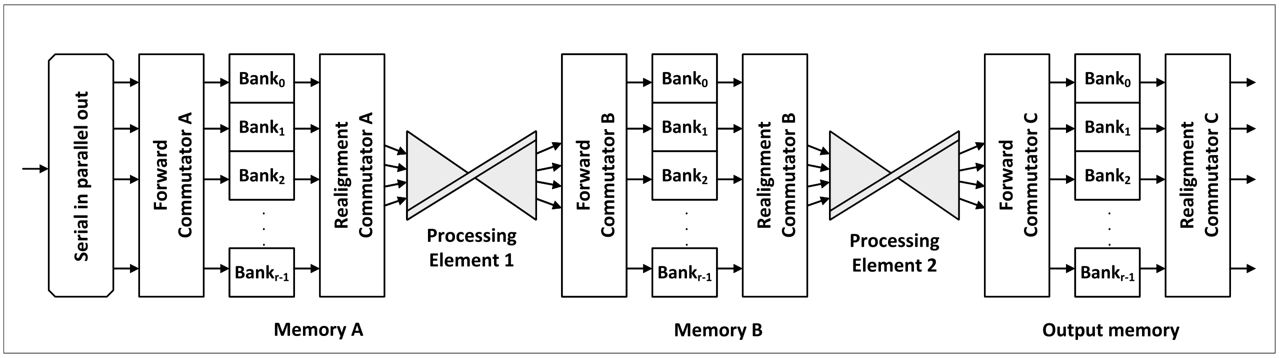


FIGURE 4. Architecture for 512 point FFT showing memories and CFA logic.

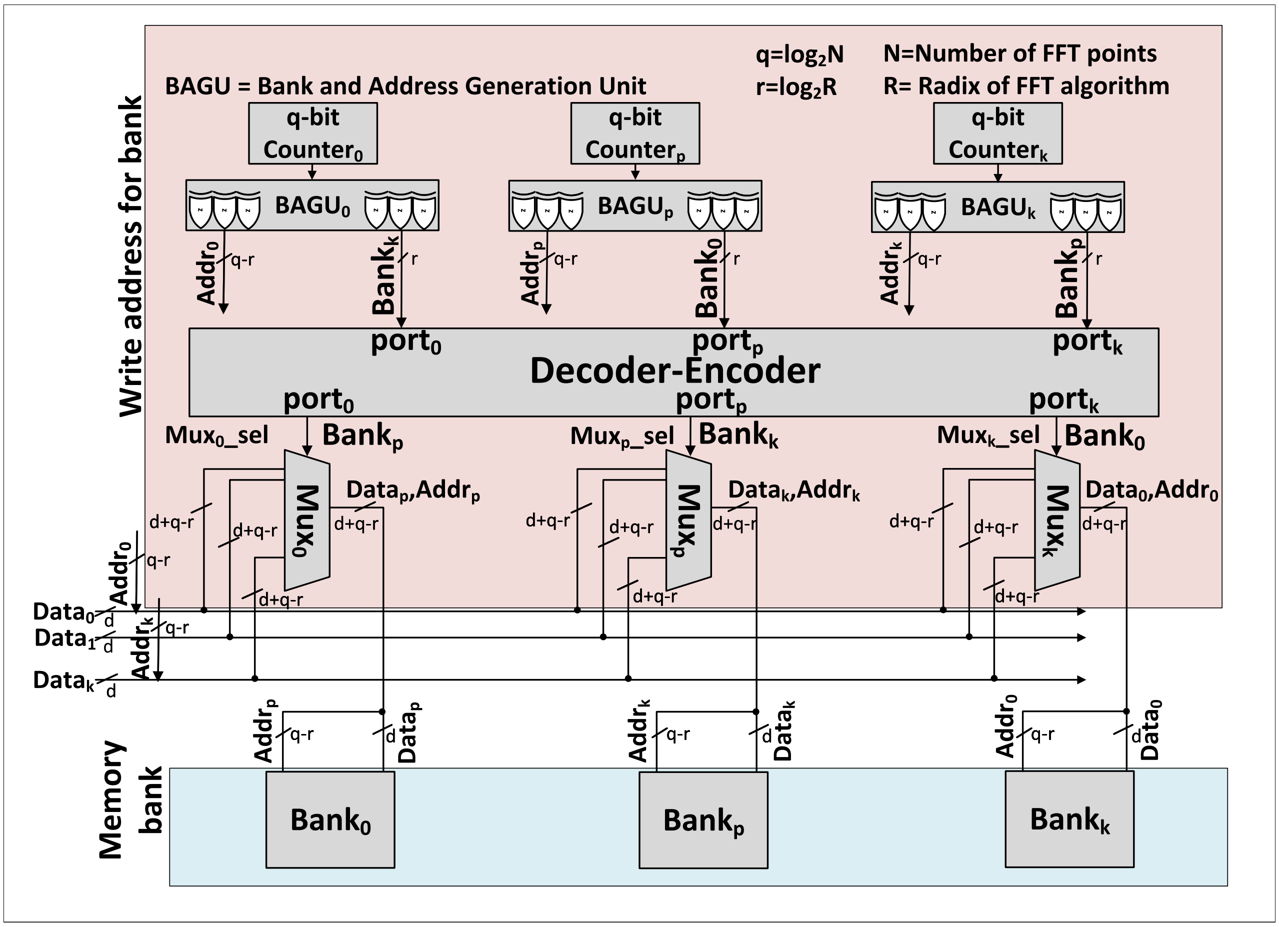


FIGURE 5. Forward commutator in CFA using XOR-based logic.

and address generator unit (BAGU). A BAGU implements (7) using XOR gates. In Fig. 5, $Bank_k$ is calculated by $BAGU_0$. This means that $Data_0$ is to be stored at $Bank_k$. Hence, the multiplexer at k^{th} port must have the selection input as $Bank_0$. For this transfer, a unique switching unit called decoder-encoder switch is required, which transfers data from the input port to the output port in such a way that if the input is $Bank_k$ at $port_0$, then the output at $port_k$ is $Bank_0$. Additionally, the address generated by $BAGU_0$ is for $Bank_k$, and it has to be transported to that bank. Thus, this design requires switching of both data and address, hence requiring extra multiplexers for address switching.

To read data from the banks, a reverse commutator as shown in Fig. 6 is used. The reading order is determined by the FFT algorithm. BAGUs determine the banks and addresses at which the requisitioned data is available. The decoder-encoder switch transfers this information to the select inputs of an address-select multiplexer, and the address is directed to the proper bank. The read data have to be realigned with the inputs of the PE. A second array of

TABLE 7. Calculation of the area for radix (r = 16),(N = 512) points using the SCL 180 nm library for (d = 12) bit wordlength.

Unit	Sub-unit	XOR			This work			
		Complexity	Specification	Area (μm^2)	Complexity	Specification	Area (μm^2)	Savings (%)
Address generator	Counters	$Rlog_2 N$	9 bit counters: 32 nos.:16 each for read and write	28800	$Rlog_2 \frac{N}{R}$	5 bit counters: 32 nos.: 16 each for read and write	10100	65
	BAGU	$Rlog_2 N$	32 nos: 16 each for read and write	20800	N/A	N/A	0	100
Multiplexer Unit	Mux Array	$R^2(d + log_2 \frac{N}{R})$	2 nos: 1 each for forward and realignment commutator	364400	$R^2 d$	2 nos.: 1 each for forward and realignment commutator	268000	26
	Select logic	$R^2 log_2 R$	Decoder Encoder: 2 nos: 1 each for forward and realignment commutator	48000	$R^2 log_2 \frac{N}{R}$	Shift Registers: 2 nos.: 1 each for forward and reverse commutators	23000	52
Buffers		$R(d + log_2 \frac{N}{R})$	Inserted at multiplexer outputs by synthesis tool	24800	Rd	Inserted at multiplexer outputs by synthesis tool	23800	4
Total Area				**486800**			**324900**	**33**

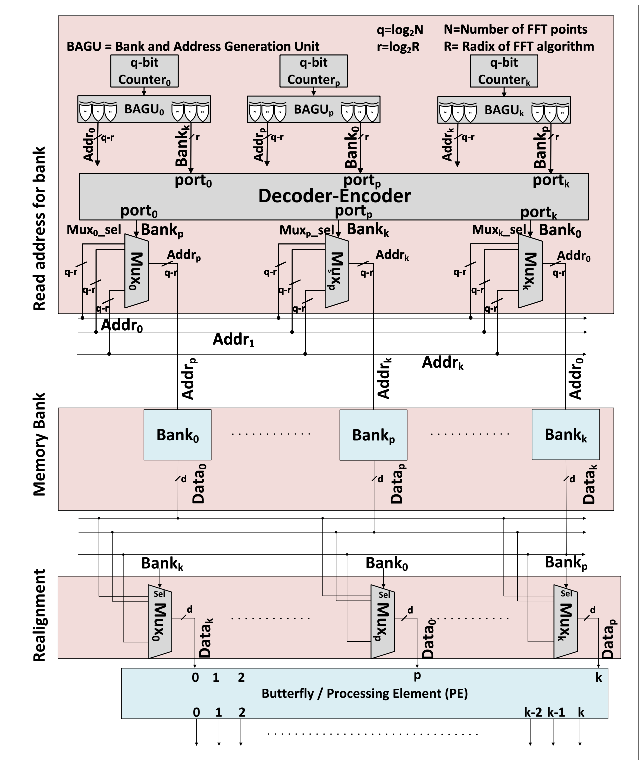


FIGURE 6. Reverse commutator for CFA using XOR-based logic.

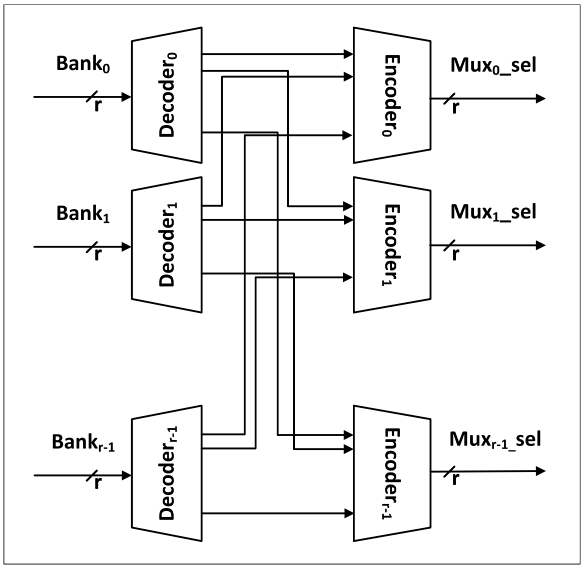


FIGURE 7. Decoder encoder used as a switch in XOR-based CFA.

TABLE 8. Comparison of hardware complexity using the SCL 180 nm library for 12 bit wordlength.

Metric	XOR	This work (PS)	Savings (%)
Area $(mm)^2$	0.49	0.33	**33**
Power (mW)	22	17	**23**
Execution Time (ns)	7.4	4.5	**39**

multiplexer bank is used to achieve this, where the select input is the bank generated by the associated BAGU. Thus, CFA is achieved with the help of both the forward and reverse commutator, the former placed before the memory to write data and the latter placed after the memory to read data. The decoder-encoder switch is made with an array of decoders and encoders, as shown in Fig. 7.

VI. SYNTHESIS RESULTS AND COMPARISON

The design implementations of both the XOR-based technique and the proposed PS-based technique are carried out using Verilog. The correctness of the designs is verified through simulations carried out using Synopsys VCS. For synthesis, an SCL Chandigarh foundry node of 180 nm [24] is used, and synthesis is carried out using Synopsys DC.

Additionally, various parts of the addressing logic are synthesized separately to evaluate their relative complexity, and the corresponding results are presented in Table 7. The table shows that the proposed scheme uses 65% fewer counters, 26% fewer multiplexers, 52% less selection logic, and achieves an overall 33% area efficiency, compared with the XOR-based scheme. The area, power, and execution time are obtained from the synthesis result and are shown in Table 8. In terms of power, the proposed scheme is 23% better, and the timing performance is 39% faster. Complexity analysis is performed to study the dependence of the area of logic elements on radix R, the number of FFT points N,

TABLE 9. Comparison of hardware complexity for various FFT sizes (N) and radices (r) for d-bit wordlength (d = 12).

Radix	r=8				r=16				r=32			
FFT points, N→	256	512	1024	2048	256	512	1024	2048	256	512	1024	2048
Scheme ↓	Area (mm^2)											
This work (PS)	0.090	0.092	0.094	0.096	0.318	0.325	0.332	0.338	1.187	1.209	1.232	1.254
XOR-based	0.135	0.143	0.152	0.161	0.458	0.487	0.515	0.544	1.658	1.758	1.857	1.957
Savings (%)	**33.3**	**35.9**	**38.2**	**40.3**	**30.6**	**33.3**	**35.6**	**37.8**	**28.4**	**31.2**	**33.7**	**35.9**

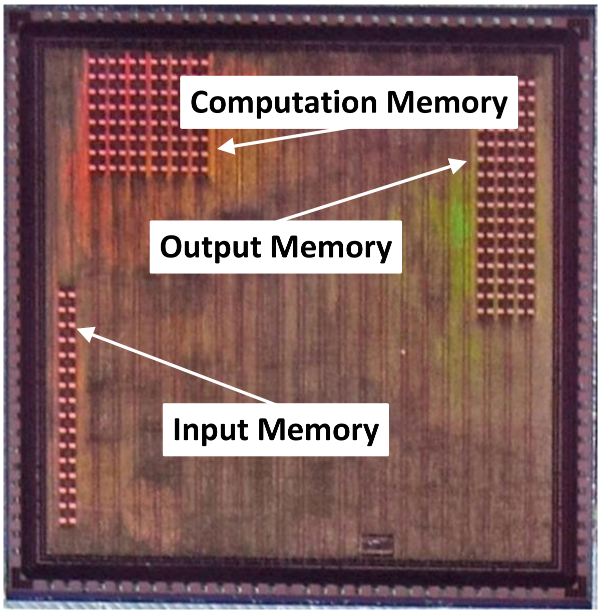


FIGURE 8. Micrograph of the 512 point FFT chip designed using a PS CFA circuit.

and wordlength D, and the results are tabulated in columns 3 and 6 of Table 7. Simulations are carried out for different numbers of FFT points with variable radix sizes of 8, 16, and 32 to determine the scalability of the proposed technique. The results of the simulations are tabulated in Table 9 and plotted in the graph shown in Fig. 9. From the table, it is clear that as the number of points increases, the savings in hardware by using the proposed PS over conventional XOR-based design improve. For both schemes, as radix increases, complexity grows in an exponential manner, but the PS scheme remains more or less 30-40% efficient for all radices.

To estimate overall savings on silicon area, it must be observed that one CFA circuit is used per memory. Hence, for an FFT with continuous flow using two memories, there are two CFA circuits. In order to compare the performance, the 512-point CF FFT chip was designed using both PS and XOR based techniques. The results showed that an area of 3 mm^2 is needed for PS and 3.35 mm^2 for XOR based design. It is also observed that the CFA logic portion of the proposed architecture occupies 0.65 mm^2 area while that of the XOR based design occupies 1 mm^2 area. Hence, it can be concluded that the area savings of the CFA logic and the complete chip of the proposed architecture is 35% and 10%, respectively, compared with XOR based architecture. The FFT chip with PS CFA circuit was fabricated in 180 nm SCL Chandigarh foundry, and its micrograph is shown in Fig. 8.

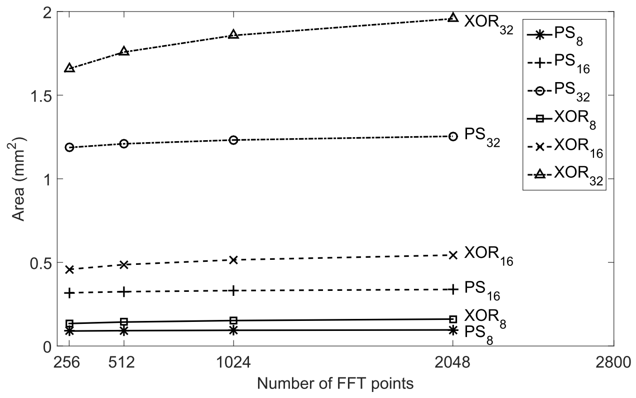


FIGURE 9. Comparison of the area for XOR-based design vs. progressive shifting for different numbers of FFT points. (The subscript r in XOR_r and PS_r represent the radix of the FFT).

From the above observations, it is inferred that due to the following features, the proposed scheme outperforms the XOR-based bank generation scheme:

- Bank and address generation logic is simpler.
- Since the address for a bank is generated by its linked address module, multiplexers for address shifting are not required.
- The decoder-encoder switch is replaced with an SRA, which is less complex.

VII. CONCLUSION

CFA circuits have to be repeated as many times as the number of memories in the FFT architecture. Hence, they do contribute to the overall area and complexity of the design. The progressive shifting shown in this work is an efficient technique; in a typical case, it saves approximately 33% of the area in commutation circuits. It is best suited for FFTs with a high radix and a lower number of stages or for architectures with multiple memories. Additionally, designs with a memory dedicated to a particular stage of PE can take advantage of this technique. On the other hand, this scheme is not easy to use for designs in which one memory feeds more than two stages, as in low-radix architectures. More investigation is necessary to adapt it for such cases. Another advantage of this design is that it does not change much from one architecture to another, facilitating its reusability once designed.

Comparatively, the other addressing techniques are mathematically rigorous and require the design of individual addressing logic for different FFT architectures and need to start from scratch each time the architecture changes. In conclusion, for high-throughput FFTs, the progressive shifter is expected to result in substantial benefits in terms of area and design effort.

ACKNOWLEDGMENT

The authors would like to thank the Semiconductor Laboratory, Chandigarh (SCL), for sharing the libraries of their foundry and fabricating the chip.

REFERENCES

[1] Y. Ma and L. Wanhammar, "A hardware efficient control of memory addressing for high-performance FFT processors," *IEEE Trans. Signal Process.*, vol. 48, no. 3, pp. 917–921, Mar. 2000.

[2] D. Cohen, "Simplified control of FFT hardware," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 24, no. 6, pp. 577–579, Dec. 1976.

[3] L. G. Johnson, "Conflict free memory addressing for dedicated FFT hardware," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 39, no. 5, pp. 312–316, May 1992.

[4] P.-Y. Tsai and C.-Y. Lin, "A generalized conflict-free memory addressing scheme for continuous-flow parallel-processing FFT processors with rescheduling," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 19, no. 12, pp. 2290–2302, Dec. 2011.

[5] S.-J. Huang and S.-G. Chen, "A high-throughput radix-16 FFT processor with parallel and normal input/output ordering for IEEE 802.15.3C systems," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 59, no. 8, pp. 1752–1765, Aug. 2012.

[6] Y. Tian, Y. Hei, Z. Liu, Q. Shen, Z. Di, and T. Chen, "A modified signal flow graph and corresponding conflict-free strategy for memory-based FFT processor design," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 66, no. 1, pp. 106–110, Jan. 2019.

[7] D. Reisis and N. Vlassopoulos, "Address generation techniques for conflict free parallel memory accessing in FFT architectures," in *Proc. 13th IEEE Int. Conf. Electron., Circuits Syst.*, Dec. 2006, pp. 1188–1191.

[8] J. Hazarika, M. T. Khan, and S. R. Ahamed, "Low-complexity continuous-flow memory-based FFT architectures for real-valued signals," in *Proc. 32nd Int. Conf. VLSI Design 18th Int. Conf. Embedded Syst. (VLSID)*, Jan. 2019, pp. 46–51.

[9] Y. Ma, "An effective memory addressing scheme for FFT processors," *IEEE Trans. Signal Process.*, vol. 47, no. 3, pp. 907–911, Mar. 1999.

[10] S. Richardson D. Marković, A. Danowitz, J. Brunhaver, and M. Horowitz, "Building conflict-free FFT schedules," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 62, no. 4, pp. 1146–1155, Apr. 2015.

[11] K. F. Xia, B. Wu, T. Xiong, and T. C. Ye, "A memory-based FFT processor design with generalized efficient conflict-free address schemes," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 6, pp. 1919–1929, Jun. 2017.

[12] C.-F. Hsiao, Y. Chen, and C.-Y. Lee, "A generalized mixed-radix algorithm for memory-based FFT processors," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 57, no. 1, pp. 26–30, Jan. 2010.

[13] S.-G. Chen, S.-J. Huang, M. Garrido, and S.-J. Jou, "Continuous-flow parallel bit-reversal circuit for MDF and MDC FFT architectures," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 61, no. 10, pp. 2869–2877, Oct. 2014.

[14] Z.-G. Ma, X.-B. Yin, and F. Yu, "A novel memory-based FFT architecture for real-valued signals based on a radix-2 decimation-in-frequency algorithm," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 62, no. 9, pp. 876–880, Sep. 2015.

[15] X.-B. Mao, Z.-G. Ma, F. Yu, and Q.-J. Xing, "A continuous-flow memory-based architecture for real-valued FFT," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 64, no. 11, pp. 1352–1356, Nov. 2017.

[16] M. Yuan, Z. Ma, F. Yu, and Q. Xing, "A novel address scheme for continuous-flow parallel memory-based real-valued FFT processor," *Electronics*, vol. 8, no. 9, p. 1042, Sep. 2019.

[17] C. Ma, H. Chen, Y. Liu, and Y. Wang, "An efficient design for general mixed radix FFT processors," *IEICE Electron. Exp.*, vol. 13, no. 6, pp. 1–7, Mar. 2016.

[18] S.-S. Long, M.-Y. Hong, and M.-T. Shiue, "A low-complexity generalized memory addressing scheme for continuous-flow fast Fourier transform," in *Proc. 3rd Int. Conf. Comput. Commun. Syst. (ICCCS)*, Apr. 2018, pp. 492–496.

[19] X. Xiao, E. Oruklu, and J. Saniie, "An efficient FFT engine with reduced addressing logic," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 55, no. 11, pp. 1149–1153, Nov. 2008.

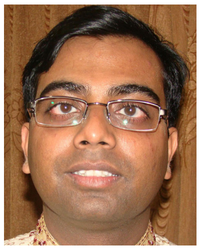
[20] S. Liu and D. Liu, "A high-flexible low-latency memory-based FFT processor for 4G, WLAN, and future 5G," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 3, pp. 511–523, Mar. 2019.

[21] B. G. Jo and M. H. Sunwoo, "New continuous-flow mixed-radix (CFMR) FFT processor using novel in-place strategy," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 52, no. 5, pp. 911–919, May 2005.

[22] J. Wang, S. Li, and X. Li, "Scheduling of data access for the radix-2^k FFT processor using single-port memory," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 7, pp. 1676–1689, Jul. 2020.

[23] C. Burrus, "Index mappings for multidimensional formulation of the DFT and convolution," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 25, no. 3, pp. 239–242, Jul. 1977.

[24] H. S. Jatana and N. M. Desai, "SCL 180 nm CMOS foundry: High reliability ASIC design for aerospace applications," in *Proc. 19th Int. Symp. VLSI Design Test*, Jun. 2015, pp. 1–2.



SUMIT AGARWAL received the B.E. degree in electronics and telecom from the Assam Engineering College, in 2002. He is currently pursuing the Ph.D. degree with IIT Guwahati. From 2004 to 2007, he worked with Indian Defence Research and Development Organisation. He worked as a Senior Design Engineer with GE Healthcare. His work in defense research and development led him to publish several papers in journals and conferences in the field of phased array microstrip antenna. His research interests include OFDM and high-speed digital design for communication systems.

SHAIK RAFI AHAMED received the B.Tech. and M.Tech. degrees in electronics and communication engineering from Sri Venkateswara University, Tirupati, India, in 1991 and 1993, respectively, and the Ph.D. degree from IIT Kharagpur, India, in 2008. He is currently a Professor with the Department of Electronics and Electrical Engineering, IIT Guwahati, Guwahati, India. His research interests include digital, adaptive, biomedical, and VLSI signal processing.

ANUP KUMAR GOGOI was born in Assam, India, in 1953. He received the B.E. (electrical) degree from Assam Engineering College, in 1976, and the master's and Ph.D. degrees in electronics from IIT Kanpur, in 1981 and 1991, respectively. He recently retired as a Professor with IIT Guwahati, Assam. He has published numerous articles in various fields of VLSI. His current research interests include VLSI circuits and systems.

GAURAV TRIVEDI (Member, IEEE) received the Ph.D. degree in electrical engineering from the Indian Institute of Technology Bombay, India, in 2007. He worked as a Senior Member of Technical Staff with Cadence Design Systems and Berkeley Design Automation (presently, Mentor-Siemens) for a period of three years and as a Postdoctoral Fellow for a period of two years at the Indian Institute of Technology Bombay, before joining the Indian Institute of Technology Guwahati (IIT Guwahati), India, as a Faculty Member. He is currently an Associate Professor with the Department of Electronics and Electrical Engineering, IIT Guwahati. His research interests include VLSI CAD, semiconductor devices, digital and analog circuit design, high-performance computing, computer architecture and algorithms, embedded and the IoT, and quantum computing.