Multi-Channel FFT Architectures Designed via Folding and Interleaving

Nanda K. Unnikrishnan
Dept. Electrical and Computer Engineering
University of Minnesota
Minneapolis MN, USA
unnik005@umn.edu

Keshab K. Parhi
Dept. Electrical and Computer Engineering
University of Minnesota
Minneapolis MN, USA
parhi@umn.edu

***Abstract*—Computing the FFT of a single channel is well understood in the literature. However, computing the FFT of multiple channels in a systematic manner has not been fully addressed. This paper presents a framework to design a family of multi-channel FFT architectures using *folding* and *interleaving*. Three distinct multi-channel FFT architectures are presented in this paper. These architectures differ in the input and output preprocessing steps and are based on different folding sets, i.e., different orders of execution.**

***Index Terms*—FFT, Multi-Channel FFT, Folding, Interleaving**

I. INTRODUCTION

Fast Fourier Transform (FFT) algorithm is a critical part of modern signal processing and machine learning systems, and is used in applications ranging from digital communication [1]–[4] to generating features for neural networks. There has been significant research on design of pipelined [5]–[7] and parallel [8], [9] FFT architectures for both complex and real-valued input signals [10]–[12]. These architectures were designed mostly with systematic principles. However, systematic design of architectures for multi-channel has not been fully explored.

We can broadly classify existing multi-channel approaches into two categories. The first approach is memory-based which store all input channels into a large memory bank [4], [13], [14]. While this approach has less data movement and reordering, it requires a significantly larger memory footprint, especially simultaneous writing and reading support. The second approach uses a combination of delay elements and switches to create data commutators [1], [15], [16]. These data commutators can manipulate input data into the proper format to interleave the channels and handle the data reordering for the first stage. While this approach is memory efficient, it lacks systematic exploration and is tailored for specific hardware applications or architectures. Therefore, this paper explores the systematic design of multi-channel FFT architectures by applying folding and interleaving to existing architectures.

The rest of the paper is organized as follows. Section II describes the proposed interleaving models and how to derive them. Section III compares the different proposed approaches and their advantages over existing designs. Finally, section IV summarizes the main conclusions of the paper.

This research was supported in part by the National Science Foundation under grant number CCF-1954749.

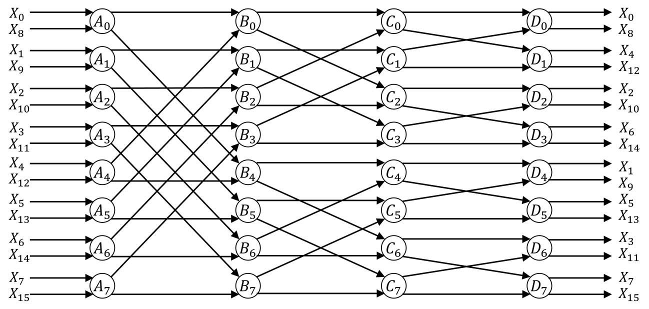
x_0 x_8 x_1 x_9 x_2 x_{10} x_3 x_{11} x_4 x_{12} x_5 x_{13} x_6 x_{14} x_7 x_{15}
A_0 A_1 A_2 A_3 A_4 A_5 A_6 A_7
B_0 B_1 B_2 B_3 B_4 B_5 B_6 B_7
C_0 C_1 C_2 C_3 C_4 C_5 C_6 C_7
D_0 D_1 D_2 D_3 D_4 D_5 D_6 D_7
X_0 X_8 X_4 X_{12} X_2 X_{10} X_6 X_{14} X_1 X_9 X_5 X_{13} X_3 X_{11} X_7 X_{15}

Fig. 1. DFG of a Radix-2 16-point DIF FFT with processors allocation for folding.

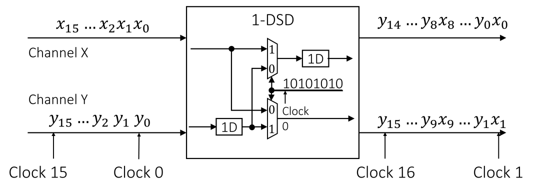
$x_{15} \dots x_2 x_1 x_0$
Channel X
Channel Y
$y_{15} \dots y_2\ y_1\ y_0$
1-DSD
1D
10101010
Clock
$y_{14} \dots y_8 x_8 \dots y_0 x_0$
$y_{15} \dots y_9 x_9 \dots y_1 x_1$
Clock 15
Clock 0
Clock 16
Clock 1

Fig. 2. Pre-processing multi-channel input into an interleaved 2-parallel form using a 1-DSD circuit.

II. PROPOSED MULTI-CHANNEL FFT ARCHITECTURES

This section presents the multi-channel interleaved architectures for complex-valued signals with the radix-2 algorithm. These architectures were designed with the use of folding sets [17]. Folding sets are ordered sets that describe how operations map to a hardware resource in a time-multiplexed manner. Through folding, we can derive different FFT architectures by varying the operations' order. Additionally, we can use the process of interleaving [18] to alternate between the computation of each channel. Using folding sets, we present three architectures for a multi-channel FFT. For brevity, we only present the folding sets and the final architectures.

A. Architecture 1: 2-parallel FFT architecture with interleaving factor 2

A direct approach to multi-channel interleaving is to take existing FFT architectures and perform the interleaving operation. There are multiple candidates for the choice of a base architecture [19], and for this example, we use the 16-point 2-parallel DIF FFT proposed in Fig. 12 of [5] as the starting

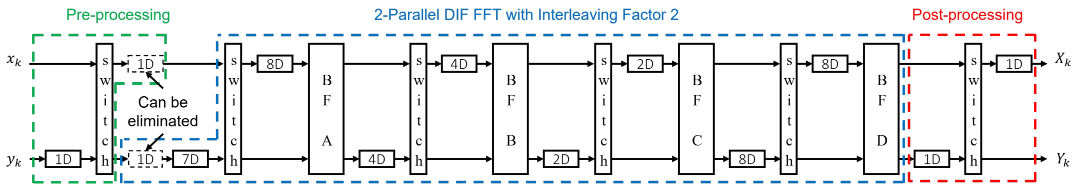

Fig. 3. Proposed 2-parallel interleaved architecture (Architecture 1) for the computation of a 2 channel 16-point DIF FFT.

point for interleaving. The base architecture defines the folding sets in Eq. (1) for the dataflow graph shown in Fig. 1.

Interleaving by a factor of 2 doubles the number of delays in the architecture. In terms of the folding set, we insert null operations in the new locations as shown in Eq. (2). This change leads to a 2-parallel FFT architecture that accepts inputs on alternate cycle. The system still maintains the throughput of a single channel system, albeit with 50% utilization.

$$\begin{aligned} A &= \{A_0, A_2, A_4, A_6, A_1, A_3, A_5, A_7\} \\ B &= \{B_5, B_7, B_0, B_2, B_4, B_6, B_1, B_3\} \\ C &= \{C_3, C_5, C_7, C_0, C_2, C_4, C_6, C_1\} \\ D &= \{D_2, D_4, D_6, D_1, D_3, D_5, D_7, D_0\} \end{aligned} \tag{1}$$

$$\begin{aligned} A &= \{A_0, \emptyset, A_2, \emptyset, A_4, \emptyset, A_6, \emptyset, A_1, \emptyset, A_3, \emptyset, A_5, \emptyset, A_7, \emptyset\} \\ B &= \{B_5, \emptyset, B_7, \emptyset, B_0, \emptyset, B_2, \emptyset, B_4, \emptyset, B_6, \emptyset, B_1, \emptyset, B_3, \emptyset\} \\ C &= \{C_3, \emptyset, C_5, \emptyset, C_7, \emptyset, C_0, \emptyset, C_2, \emptyset, C_4, \emptyset, C_6, \emptyset, C_1, \emptyset\} \\ D &= \{D_2, \emptyset, D_4, \emptyset, D_6, \emptyset, D_1, \emptyset, D_3, \emptyset, D_5, \emptyset, D_7, \emptyset, D_0, \emptyset\} \end{aligned} \tag{2}$$

We can use these null operations to interleave the second FFT channel to the same hardware as shown in Eq. (3), where prime operations refer to the second channel.

$$\begin{aligned} A &= \{A_0, A_0', A_2, A_2', A_4, A_4', A_6, A_6', A_1, A_1', A_3, A_3', A_5, A_5', A_7, A_7'\} \\ B &= \{B_5, B_5', B_7, B_7', B_0, B_0', B_2, B_2', B_4, B_4', B_6, B_6', B_1, B_1', B_3, B_3'\} \\ C &= \{C_3, C_3', C_5, C_5', C_7, C_7', C_0, C_0', C_2, C_2', C_4, C_4', C_6, C_6', C_1, C_1'\} \\ D &= \{D_2, D_2', D_4, D_4', D_6, D_6', D_1, D_1', D_3, D_3', D_5, D_5', D_7, D_7', D_0, D_0'\} \end{aligned} \tag{3}$$

Fig. 3 describes the proposed architecture for the 2-parallel DIF FFT interleaved by a factor of 2. Using interleaving, we can derive this architecture by doubling the number of delays in the original circuit and interleaving the input signals. For example, the above folding set alternately accepts two inputs from each channel. To pre-process the data into this format, we make use of a one delay-switch-delay (1-DSD) shown in Fig. 2. The same 1-DSD circuit can be used at the final output to post-process the information such that the corresponding results of each channel line up. This ordering is advantageous if we need to multiply the related FFT results of the two channels. In addition, there is a delay in both the upper and lower path after the pre-processing step. We can eliminate these delays from the circuit without any consequence by the principle of reverse pipelining [20]. If we require the outputs in the natural order, then we must implement a reorder circuit (REOC) in [5] with seven registers per channel or seven registers in total if we multiply and use each channel.

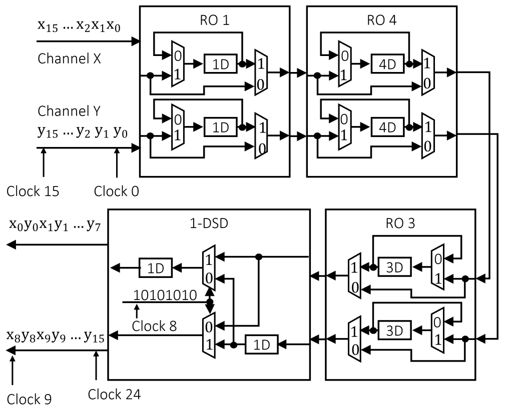
$x_{15} \dots x_2 x_1 x_0$
Channel X
Channel Y
$y_{15} \dots y_2\, y_1\, y_0$
Clock 15
Clock 0
RO 1
RO 4
1D
4D
$x_0 y_0 x_1 y_1 \dots y_7$
1-DSD
RO 3
3D
10101010
Clock 8
$x_8 y_8 x_9 y_9 \dots y_{15}$
Clock 9
Clock 24

Fig. 4. Pre-processing multi-channel input into an alternative interleaved 2-parallel form.

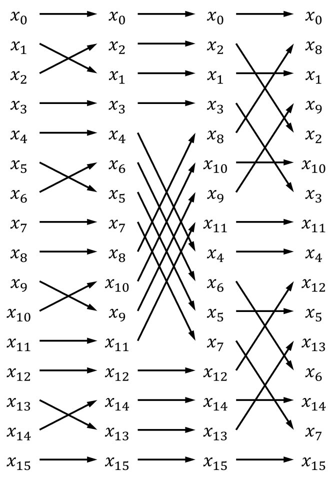

Fig. 5. Data movement in the reorder circuits.

B. Architecture 2: Alternate 2-parallel FFT architecture with interleaving factor 2

The main difference between Eq. (4) and Eq. (3) is the ordering of the computations within the folding set. Eq. (4) applies a simple ordering scheme for mapping the computations to the processor. As a consequence, the output order of the circuit follows the output order of the DFG in Fig. 1. To use the new folding sets, we design a pre-processing step, shown in Fig. 4, that takes the two input channels and generates a sequence required for the circuit.

The circuit in Fig. 4 uses REOC circuits to swap data entries

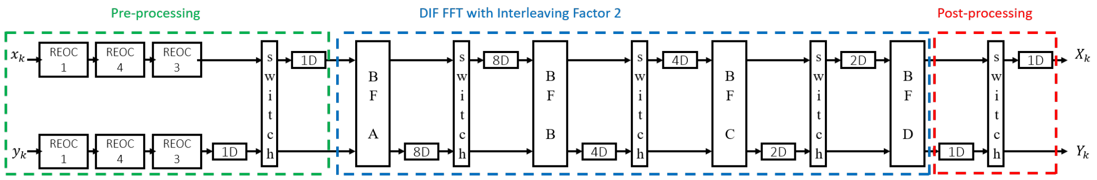

Fig. 6. Proposed interleaved architecture (Architecture 2) for the computation of a 2-channel 16-point DIF FFT.

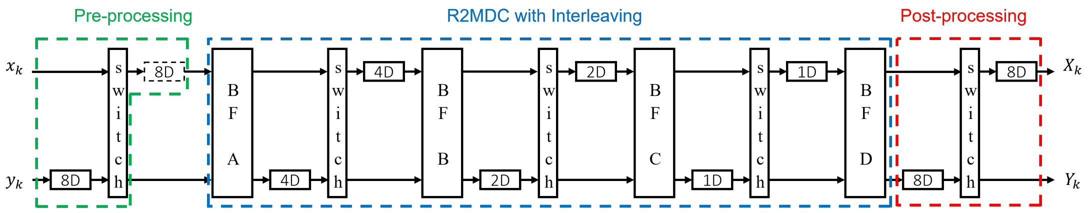

Fig. 7. Proposed interleaved architecture (Architecture 3) based on R2MDC for the computation of a 2-channel 16-point DIF FFT.

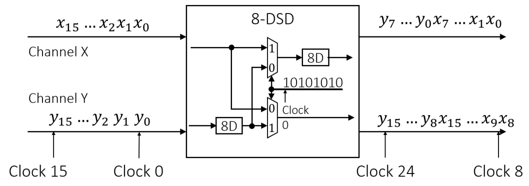
8-DSD
$x_{15} \dots x_2 x_1 x_0$
Channel X
Channel Y
$y_{15} \dots y_2\, y_1\, y_0$
$y_7 \dots y_0 x_7 \dots x_1 x_0$
$y_{15} \dots y_8 x_{15} \dots x_9 x_8$
8D
10101010
Clock
Clock 15
Clock 0
Clock 24
Clock 8

Fig. 8. Pre-processing multi-channel input into an interleaved form, where FFT is performed on one channel at a time.

that are a fixed delay apart. For this architecture, the data movement function of the REOC circuit is shown in Fig. 5. The first step in the pre-processing uses a combination of $RO1{+}RO3{+}RO4$ to bring all the inputs used for a butterfly to adjacent cycles. Additionally, the circuit ensures that the order of operations matches the order in the DFG in Fig. 1. Finally, the post-processing step uses a 1-DSD to un-interleave the two FFT channel outputs. The overall architecture showing the pre-processing, post-processing, and FFT modules is shown in Fig. 6. The outputs of the circuit are in a standard bit-reversed order, and we use bit reversal circuits like [21] to bring them to a natural order. This reversal circuit corresponds to 9 registers per output channel for a 16 point FFT.

$$\begin{aligned}
A &= \{A_0, A_0', A_1, A_1', A_2, A_2', A_3, A_3', A_4, A_4', A_5, A_5', A_6, A_6', A_7, A_7'\}\\
B &= \{B_4, B_4', B_5, B_5', B_6, B_6', B_7, B_7', B_0, B_0', B_1, B_1', B_2, B_2', B_3, B_3'\}\\
C &= \{C_2, C_2', C_3, C_3', C_4, C_4', C_5, C_5', C_6, C_6', C_7, C_7', C_0, C_0', C_1, C_1'\}\\
D &= \{D_1, D_1', D_2, D_2', D_3, D_3', D_4, D_4', D_5, D_5', D_6, D_6', D_7, D_7', D_0, D_0'\}
\end{aligned} \tag{4}$$

C. Architecture 3: Direct channel interleaving

Prior architectures perform interleaving on existing architectures with 100% utilization. However, we can derive architecture with full utilization using interleaving even when starting from architectures with half utilization. For example, consider the radix-2 multipath delay commutator (R2MDC) with the folding sets in Eq. (5).

$$\begin{aligned}
A &= \{A_0, A_1, A_2, A_3, A_4, A_5, A_6, A_7, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset\}\\
B &= \{\emptyset, \emptyset, \emptyset, \emptyset, B_0, B_1, B_2, B_3, B_4, B_5, B_6, B_7, \emptyset, \emptyset, \emptyset, \emptyset\}\\
C &= \{\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, C_0, C_1, C_2, C_3, C_4, C_5, C_6, C_7, \emptyset, \emptyset\}\\
D &= \{\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, D_0, D_1, D_2, D_3, D_4, D_5, D_6, D_7, \emptyset\}
\end{aligned} \tag{5}$$

There has been extensive research on using different techniques like 2-parallel approaches serial commutators to overcome these inefficiencies. However, we can exploit these null operations in the folding sets to perform interleaving as seen in the folding sets in Eq. (6).

$$\begin{aligned}
A &= \{A_0, A_1, A_2, A_3, A_4, A_5, A_6, A_7, A_0', A_1', A_2', A_3', A_4', A_5', A_6', A_7'\}\\
B &= \{B_4', B_5', B_6', B_7', B_0, B_1, B_2, B_3, B_4, B_5, B_6, B_7, B_0', B_1', B_2', B_3'\}\\
C &= \{C_2', C_3', C_4', C_5', C_6', C_7', C_0, C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_0', C_1'\}\\
D &= \{D_1', D_2', D_3', D_4', D_5', D_6', D_7', D_0, D_1, D_2, D_3, D_4, D_5, D_6, D_7, D_0'\}
\end{aligned} \tag{6}$$

The advantage of the proposed folding set is that it dramatically simplifies the pre-processing step. For example, we can use an 8-DSD circuit as shown in Fig. 8 to interleave the two channels while also aligning the inputs for the first butterfly stage. Then, after performing the FFT, we can post-process the outputs with another 8-DSD circuit to realign the two channels. The overall architecture showing the pre-processing, post-processing, and FFT modules is shown in Fig. 7. This architecture has further advantages in that it also reduces the complexity to bring the result back into a natural order. As the outputs of the circuit are in a bit-reversed order for only half the FFT size, we use half-size bit reversal circuits to bring them to a natural order. This reversal circuit corresponds to 3 registers per output channel for a 16 point FFT.

D. Extension to powers of 2 channels

The previous subsections described the process for interleaving two channels. However, this approach can be generalized to any number of channels that is a power of 2. Fig. 9 shows the proposed interleaving pre-processing circuit for Architectures 1 and 3. This is a generalized circuit for

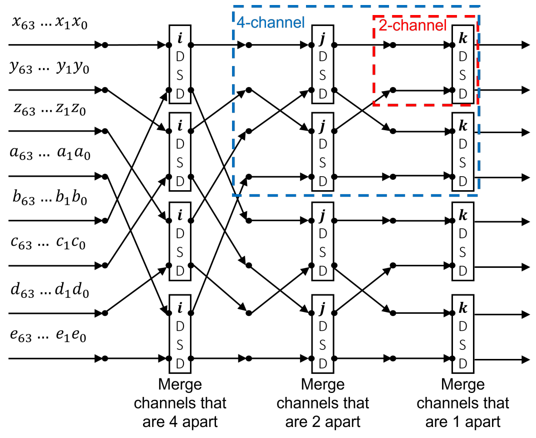

Fig. 9. Common circuitry to interleave 2, 4, or 8 channels. The values of i, j, and k depend on whether we are processing architecture 1 or 3.

2, 4 and 8 channels. However, adding additional stages can easily extend this approach to more channels. For Architecture 1, the values of the registers i, j, and k are 4, 2, and 1, respectively. In this example, an 8-channel interleaver, the output is directly in the form required for an 8-parallel FFT circuit. For Architecture 3, with a 16-point FFT, the values for i, j, and k are 8, 4, and 2, respectively. For the 64-Point example shown in Fig. 9, Architecture 3 would require i, j, and k to be 32, 16, and 8, respectively.

III. Comparison of Architectures

All three proposed architectures have identical throughput and processors. Therefore, we compare the memory footprint of these approaches and focus primarily on the pre-processing, post-processing, and reorder steps. These components are the differentiating factors when interleaving multiple channels. The pre-processing step interleaves the multiple channels as required by the architecture. The post-processing separates the results into their corresponding channels. The output after post-processing is not in the natural order and requires reordering. The details for each of these components are described in their respective architectures. Table I shows the breakdown of the memory footprint into pre-processing, post-processing, architecture, and reordering registers.

TABLE I
Comparison of the memory footprint, in terms of number of registers, of the three proposed architectures on a 16-point DIF FFT

Architecture	Pre-processing	FFT	Post-processing	Reordering
Arch 1	17	28	2	14
Arch 2	18	28	2	18
Arch 3	16	14	16	6

Table I shows that each architecture focuses on different aspects of the design for optimization. We have combined the pre-processing step with the first stage data reordering as these reported together in some architectures for a fair comparison. Architecture 2 is a direct implementation of interleaving while maintaining the order of computations. However, to bring the inputs to the required order requires complicated pre-processing step that consists of multiple REOCs. Architecture 1 provides elementary pre-processing and post-processing steps and keeps the base 2-parallel FFT block, that is optimized for performance and output reordering, relatively intact. As a result, Architecture 1 maintains the advantages of the base architecture performing better than Architecture 2. Architecture 3 has the same advantages of Architecture 1 while also simplifying the final reorder circuit. The post-processing step for Architecture 3 un-interleaves the channels and performs half the movement required for the data reordering. Thus we only need an N/2 REOC circuit to perform reordering, where N is the size of the FFT.

TABLE II
Comparison of the proposed interleaving pre-processing step versus existing approaches

Pre-possessing circuit	Memory elements	Latency	Control complexity
Memory banks [4], [13], [14]	$M \times N$	N	Complex memory reads
Multi-channel commutators [1], [15], [16]	$(M-1)\times N$	$(M-1)\times N/M$	Complex pre-processing control
Proposed architecture 3	$(M-1)\times N$	$(M-1)\times N/M$	Counter based pre-processing control

Table II shows a detailed comparison of the proposed pre-processing method with existing literature. Here, M is the number of interleaved channels, and N is the size of the FFT. The prior approaches are based on using a large memory bank [4], [13], [14] to store the information from all channels and retrieving the information as required. However, this method often employs complicated memory access patterns to retrieve the data in the required form. Additionally, they have a larger memory footprint as they do not optimize register utilization by performing a lifetime analysis.

Multi-channel commutators are increasingly common in MIMO-OFDM applications [1], [15], [16], and they overcome some of the shortcomings of memory bank approaches. This method minimizes the number of registers required to pre-process the inputs and have optimal latency. However, existing approaches are ad-hoc and do not design generalized structures for various channels and points. Additionally, the control circuit complexity of the pre-processing step increases with the number of channels, M, and FFT size, N.

IV. Conclusion

This paper presented a novel approach to design multi-channel interleaved architectures from existing FFT architectures systematically. The paper proposed three separate approaches to deriving pipeline models based on folding and interleaving. In addition, the paper presents a framework for efficiently interleaving any number of channels with elementary control logic. Thus the paper can be used as a baseline to deriving any C channel FFT module from a C parallel baseline FFT architecture.

References

[1] K.-J. Yang, S.-H. Tsai, and G. C. Chuang, “MDC FFT/IFFT processor with variable length for MIMO-OFDM systems,” *IEEE transactions on very large scale integration (VLSI) systems*, vol. 21, no. 4, pp. 720–731, 2012.

[2] Y.-W. Lin and C.-Y. Lee, “Design of an FFT/IFFT processor for MIMO OFDM systems,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 54, no. 4, pp. 807–815, 2007.

[3] P. O. Taiwo and A. Cole-Rhodes, “MIMO equalization of 16-QAM signal blocks using an FFT-based alphabet-matched CMA,” in *2017 51st Annual Conference on Information Sciences and Systems (CISS)*, 2017, pp. 1–6.

[4] M. Mahdavi, O. Edfors, V. Ówall, and L. Liu, “A low latency and area efficient FFT processor for massive MIMO systems,” in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2017, pp. 1–4.

[5] M. Ayinala, M. Brown, and K. K. Parhi, “Pipelined parallel FFT architectures via folding transformation,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 6, pp. 1068–1081, 2012.

[6] M. Garrido, “A survey on pipelined FFT hardware architectures,” *Journal of Signal Processing Systems*, 2021.

[7] S. He and M. Torkelson, “Design and implementation of a 1024-point pipeline FFT processor,” in *Proceedings of the IEEE 1998 Custom Integrated Circuits Conference (Cat. No.98CH36143)*, 1998, pp. 131–134.

[8] J. Wang, C. Xiong, K. Zhang, and J. Wei, “A mixed-decimation MDF architecture for radix-2k parallel FFT,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 01, pp. 67–78, jan 2016.

[9] M. Garrido, J. Grajal, M. A. Sánchez, and O. Gustafsson, “Pipelined radix-2k feedforward FFT architectures,” *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 21, no. 1, p. 23–32, Jan. 2013. [Online]. Available: https://doi.org/10.1109/TVLSI.2011.2178275

[10] N. K. Unnikrishnan, M. Garrido, and K. K. Parhi, “Effect of finite word-length on SQNR, area and power for real-valued serial FFT,” in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2019, pp. 1–5.

[11] M. Garrido, N. K. Unnikrishnan, and K. K. Parhi, “A serial commutator fast Fourier transform architecture for real-valued signals,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 65, no. 11, pp. 1693–1697, 2018.

[12] M. Ayinala and K. K. Parhi, “FFT architectures for real-valued signals based on radix-2^3 and radix-2^4 algorithms,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 60, no. 9, pp. 2422–2430, 2013.

[13] 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 Transactions on Circuits and Systems II: Express Briefs*, vol. 62, no. 9, pp. 876–880, 2015.

[14] J. Lee and H. Lee, “A high-speed two-parallel radix-24 FFT/IFFT processor for MB-OFDM UWB systems,” *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 91, no. 4, pp. 1206–1211, 2008.

[15] B. Kang and J. Kim, “Low complexity multi-point 4-channel FFT processor for ieee 802.11n MIMO-OFDM WLAN system,” in *2012 International Conference on Green and Ubiquitous Technology*, 2012, pp. 94–97.

[16] S. Yoshizawa, A. Orikasa, and Y. Miyanaga, “An area and power efficient pipeline FFT processor for 8×8 MIMO-OFDM systems,” in *2011 IEEE International Symposium of Circuits and Systems (ISCAS)*, 2011, pp. 2705–2708.

[17] K. Parhi, C.-Y. Wang, and A. Brown, “Synthesis of control circuits in folded pipelined DSP architectures,” *IEEE Journal of Solid-State Circuits*, vol. 27, no. 1, pp. 29–43, 1992.

[18] K. K. Parhi, “Hierarchical folding and synthesis of iterative data flow graphs,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 60, no. 9, pp. 597–601, 2013.

[19] C. Cheng and K. K. Parhi, “High-throughput VLSI architecture for FFT computation,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 54, no. 10, pp. 863–867, 2007.

[20] K. K. Parhi, *VLSI digital signal processing systems: Design and implementation*. John Wiley & Sons, 2007.

[21] M. Garrido, J. Grajal, and O. Gustafsson, “Optimum circuits for bit reversal,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 58, no. 10, pp. 657–661, 2011.