A LOW-LATENCY FFT-IFFT CASCADE ARCHITECTURE

Keshab K. Parhi, Fellow, IEEE

University of Minnesota at Twin Cities

ABSTRACT

This paper addresses the design of a partly-parallel cascaded FFT-IFFT architecture that does not require any intermediate buffer. Folding can be used to design partly-parallel architectures for FFT and IFFT. While many cascaded FFT-IFFT architectures can be designed using various folding sets for the FFT and the IFFT, for a specified folded FFT architecture, there exists a unique folding set to design the IFFT architecture that does not require an intermediate buffer. Such a folding set is designed by processing the output of the FFT as soon as possible (ASAP) in the folded IFFT. Elimination of the intermediate buffer reduces latency and saves area. The proposed approach is also extended to interleaved processing of multi-channel time-series. The proposed FFT-IFFT cascade architecture saves about $N/2$ memory elements and $N/4$ clock cycles of latency compared to a design with identical folding sets. For the 2-interleaved FFT-IFFT cascade, the memory and latency savings are, respectively, $N/2$ units and $N/2$ clock cycles, compared to a design with identical folding sets.

***Index Terms*—** FFT, IFFT, FFT-IFFT Cascade, Folding, Multi-channel FFT

1. INTRODUCTION

The fast Fourier transform (FFT) algorithm [1] is an important operation in many digital signal processing and machine learning systems. FFT is used for frequency-domain and time-frequency representation of signals, and for extracting features for machine learning classifiers. FFT is used in applications such as digital communication [2, 3], medical imaging [4, 5], and convolutional neural network (CNN) [6, 7], in the context of both software and hardware implementations.

FFT hardware architectures can be classified into two categories: pipelined and parallel architectures [8, 9, 10, 11, 12], and compact and memory-based FFT architectures [13, 14]. This paper addresses the optimization of the architecture for FFT-IFFT cascade for partly-parallel architectures. In a typical long convolution operation, two FFTs are computed, a pointwise multiplication is carried out, and the IFFT of the product is computed. If the second signal is constant, then the second FFT can be precomputed. This is the first paper to address the architecture for the partly-parallel FFT-IFFT cascade architectures. Partly-parallel, such as 2-parallel and 4-parallel, architectures can be designed by time-multiplexing or folding based on folding sets. If the folding sets are not selected correctly, then an intermediate buffer is needed between the FFT and IFFT hardware blocks, as shown in Fig. 1 (top). This increases latency and hardware. This paper shows that by designing a folding set for the IFFT that processes the FFT outputs as soon as possible (ASAP) can eliminate the intermediate buffer requirement, thus reducing latency and area. This is illustrated in Fig. 1(bottom).

The remainder of the paper is organized as follows. Section 2 describes the proposed novel FFT-IFFT cascaded architecture design. Section 3 extends the proposed FFT-IFFT cascade design for processing multi-channel time-series using an interleaving approach. Section 4 compares the area and latency of the proposed approaches with standard designs.

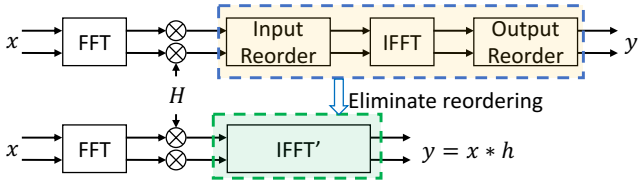


Fig. 1. Cascaded FFT-IFFT architecture with and without intermediate buffer.

2. CASCADED FFT-IFFT ARCHITECTURE DESIGN

In this section, we present the limitations and overheads associated with design of a cascaded FFT-IFFT architecture that requires an intermediate buffer. Then we introduce a novel and systematic FFT and IFFT cascaded architecture design via folding transformation [15, 16] and as soon as scheduling (ASAP) scheduling of the FFT outputs at the input of the IFFT.

2.1. Traditional FFT/IFFT Cascade Architecture

Consider the flow-graph for a 16-point FFT shown in Fig. 2(a). In a 2-parallel design, two samples are processed in parallel and N samples are processed in $N/2$ clock cycles. Furthermore, the $N/2$ butterflies in every stage are mapped

(folded) to one hardware butterfly. This corresponds to a folding factor of $N/2$. Folding sets are not unique, but two types of folding sets are commonly used. In one design, the butterflies are scheduled in consecutive cycles from top to bottom as illustrated in Fig. 2(a) where the clock cycles are marked in red. In another design, the even nodes are scheduled first followed by odd nodes (not described in this paper).

Consider the folding sets for a 16-point FFT for a folding factor of 8 corresponding to the flow-graph in Fig. 2(a):

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

In a 2-parallel design, the input x_8 is available in clock cycke 4, and the butterfly A_0 can be processed as soon as clock cycle 4. Every folding set is associated with a distinct PE from A to D. Each element within the set represents the time partition at which the operation is scheduled, given by the clock cycle modulo the folding factor. The timings of these computations establish a direct relation with the folding set. Based on the timings of the butterfly operations, the FFT architecture is depicted in the upper block of Fig. 3. Specifically, the delay-switch-delay (DSD) unit is used for the input and output data-flow control of each butterfly block (BF).

A naïve approach to implementing the IFFT algorithm is to reuse the same scheduling used for the FFT architecture. Such a folding set is expressed as:

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

The corresponding schedule for the IFFT dataflow graph is illustrated in Fig. 2(b). The hardware IFFT architecture is shown in the middle of Fig. 3 where the reorder circuit block REOC3 corresponds to an intermediate buffer. The REOC3 buffer comprises two register sets coupled with four multiplexers (MUXs) for data-flow management, and guarantees that the IFFT architecture's input data sequence is aligned correctly. The FFT-IFFT cascade using the top and middle blocks of Fig. 3 suffers from two drawbacks. First, the intermediate buffer increases area and latency. Second, the outputs of the combined FFT-IFFT cascade using the top and middle blocks are out of order and need to be reordered. These two drawbacks can be overcome by using an ASAP schedule for the IFFT such that the outputs from the FFT can be processed immediately in the IFFT architecture.

2.2. FFT-IFFT cascade using ASAP Scheduling

In the proposed systematic approach, the primary design objective lies in eliminating the need for the reordering operation after point-wise multiplication. Consequently, the FFT architecture's output sequence indices must align with the input sequence indices of the IFFT. ASAP scheduling of the FFT outputs requires a specific folding set that is different from that used for the FFT. The IFFT folding set using ASAP scheduling is described by:

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

In this folding set, the even nodes are scheduled first and the odd nodes are scheduled next; this corresponds to bit-reversed ordering of the nodes.

By leveraging this optimized folding set and seamlessly integrating it within the IFFT's pipelined and parallel architecture, we can effectively eliminate the REOC reordering unit. The proposed FFT-IFFT cascade consists of the top and bottom parts of Fig. 3. As elucidated in Fig. 3 (presented within the green box at the bottom), the required hardware resources are identical to those of the FFT architecture. Furthermore, the output sequence inherently adheres to a natural order.

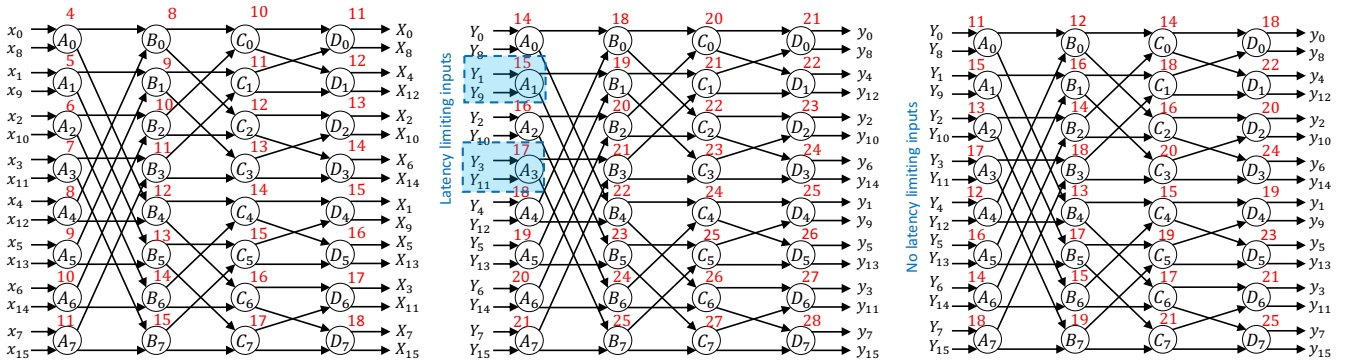
3. INTERLEAVED FFT-IFFT CASCADE ARCHITECTURE

The proposed architecture can be extended to a multi-channel framework to design an interleaved FFT and IFFT cascaded architecture. This is realized by adopting the interleaving method [15], a technique that has found previous applications in numerous domains of digital signal processing and beyond [17, 18, 19].

We utilize the folding set delineated in [19] for the FFT architecture design to begin with our optimization. An example representation of the 16-point FFT computation and its associated folding set is elaborated below:

$$\begin{aligned} A =& \big\{A'_0, A'_1, A'_2, A'_3, A'_4, A'_5, A'_6, A'_7, \\ & A_0, A_1, A_2, A_3, A_4, A_5, A_6, A_7\big\} \\ B =& \big\{B_4, B_5, B_6, B_7, B'_0, B'_1, B'_2, B'_3, \\ & B'_4, B'_5, B'_6, B'_7, B_0, B_1, B_2, B_3\big\} \\ C =& \big\{C_2, C_3, C_4, C_5, C_6, C_7, C'_0, C'_1, \\ & C'_2, C'_3, C'_4, C'_5, C'_6, C'_7, C_0, C_1\big\} \\ D =& \big\{D_1, D_2, D_3, D_4, D_5, D_6, D_7, D'_0, \\ & D'_1, D'_2, D'_3, D'_4, D'_5, D'_6, D'_7, D_0\big\}. \end{aligned} \tag{4}$$

In contrast to Eq. (1), the uniqueness of this folding set is revealed in its composition: it contains two distinct data sequences with an interleaving factor of two. Notably, these



(a) Data-flow graph and schedule for the FFT (b) Data-flow graph and schedule for the traditional IFFT architecture. (c) Data-flow graph and schedule for the optimized IFFT architecture.

Fig. 2. Data-flow graphs for FFT and IFFT with scheduling. Clock cyles are marked in red.

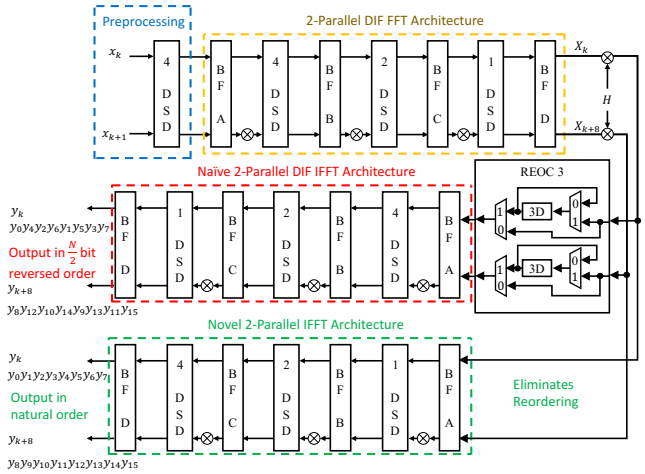


Fig. 3. Cascaded 16-Point FFT-IFFT architectures. Top-Middle cascade represents a traditional design. Top-bottom cascade represents the proposed design.

sequences denote the X-channel and the Y-channel, respectively. Another feature of this folding set is its inherent efficiency, ensuring a 100% resource utilization across the FFT architecture. The PEs conduct computations for both data sequences in a time-multiplexed and interleaved fashion. This operation is depicted in Fig. 4(a), wherein the X-channel and Y-channel execution timings are discernibly represented in colors of red and blue, respectively.

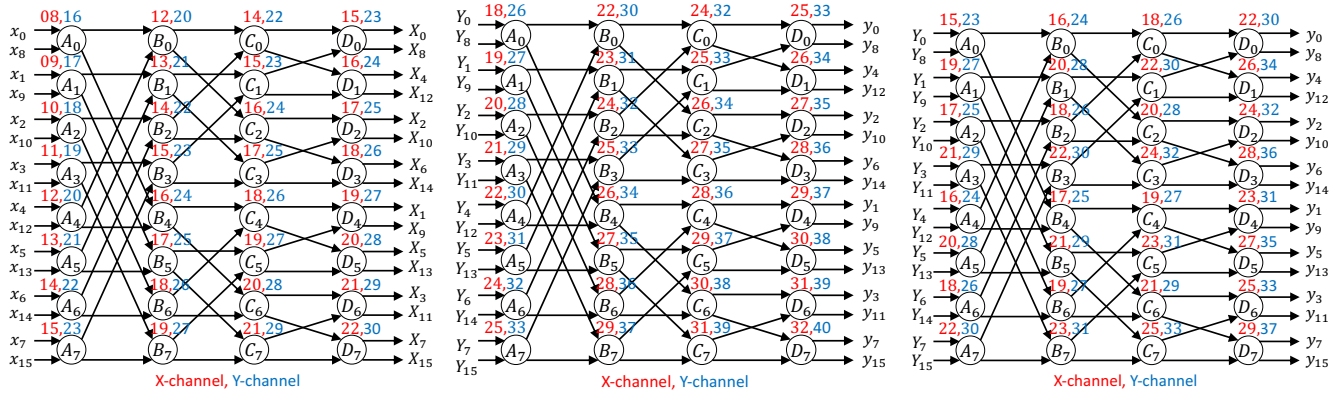
A naïve approach to implementing the IFFT algorithm is to straightforwardly employ the FFT architecture, encompassing the subsequent folding set:

$$\begin{aligned} A =&\{A'_6, A'_7, A_0, A_1, A_2, A_3, A_4, A_5, \\ &A_6, A_7, A'_0, A'_1, A'_2, A'_3, A'_4, A'_5\} \\ B =&\{B'_2, B'_3, B'_4, B'_5, B'_6, B'_7, B_0, B_1, \\ &B_2, B_3, B_4, B_5, B_6, B_7, B'_0, B'_1\} \\ C =&\{C'_0, C'_1, C'_2, C'_3, C'_4, C'_5, C'_6, C'_7, \\ &C_0, C_1, C_2, C_3, C_4, C_5, C_6, C_7\} \\ D =&\{D_7, D'_0, D'_1, D'_2, D'_3, D'_4, D'_5, D'_6, \\ &D'_7, D_0, D_1, D_2, D_3, D_4, D_5, D_6\}. \end{aligned} \tag{5}$$

This approach is shown in Fig. 4(b) for the data-flow graph and Fig. 5 (emphasized with a red box) for the architecture, respectively. Notably, while this might appear straightforward, it introduces several inefficiencies in the process. One of the drawbacks is the latency-limiting input, which mandates the use of a REOC unit. Additionally, the architecture consumes two more DSD units engaged in the tasks of de-interleaving and re-interleaving before and after the pointwise multiplication.

Hence, an elaborate exploration into the systematic design for the interleaved FFT and IFFT architecture becomes imperative to remove these unnecessary components for reordering. Specifically, the input sequences for both the X-channel and Y-channel in the IFFT are aligned with the output sequences of the FFT architecture. To achieve this, the optimized folding set is presented as follows:

$$\begin{aligned} A =&\{A_4, A_2, A_6, A_1, A_5, A_3, A_7, A'_0, \\ &A'_4, A'_2, A'_6, A'_1, A'_5, A'_3, A'_7, A_0\} \\ B =&\{B_0, B_4, B_2, B_6, B_1, B_5, B_3, B_7, \\ &B'_0, B'_4, B'_2, B'_6, B'_1, B'_5, B'_3, B'_7\} \\ C =&\{C'_3, C'_7, C_0, C_4, C_2, C_6, C_1, C_5, \\ &C_3, C_7, C'_0, C'_4, C'_2, C'_6, C'_1, C'_5\} \\ D =&\{D'_2, D'_6, D'_1, D'_5, D'_3, D'_7, D_0, D_2, \\ &D_4, D_6, D_1, D_5, D_3, D_7, D'_0, D'_4\}. \end{aligned} \tag{6}$$



(a) Data-flow graph and schedule for an interleaved FFT.

(b) Data-flow graph and schedule for a traditional interleaved IFFT.

(c) Data-flow graph and schedule for an optimized interleaved IFFT.

Fig. 4. Data-flow graphs and schedules for Interleaved FFT and IFFT.

Table 1. Performance Analysis and comparison of the proposed designs and the prior works (for $N = 1024$)

Design	# BF	# Memory Elem.	Latency	# MUXs	Throughput
FFT+IFFT [9]	$\log_2 N$ (10)	$3N - 6$ (3066)	$1.5N - 3$ (1533)	$4\log_2 N + 2$ (42)	2
Proposed I	$\log_2 N$ (10)	$2.5N - 4$ (2556)	$1.25N - 2$ (1278)	$4\log_2 N - 2$ (38)	2
FFT+IFFT [19]	$\log_2 N$ (10)	$4.5N - 6$ (4602)	$2N - 3$ (2045)	$4\log_2 N + 4$ (44)	2
Proposed II	$\log_2 N$ (10)	$4N - 4$ (4092)	$1.5N - 2$ (2046)	$4\log_2 N$ (40)	2

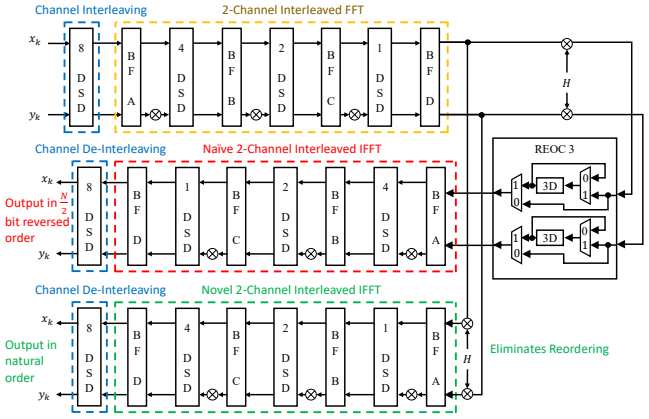


Fig. 5. Cascaded interleaved 16-Point FFT-IFFT architectures. Top-Middle cascade represents a traditional design. Top-bottom cascade represents the proposed design.

This folding set serves as the basis for deriving the schedule shown in Fig. 4(c) and the architecture shown at the bottom of Fig. 5. The proposed low-latency and low-area architecture eliminates the need for additional DSD and REOC units, as displayed in Fig. 5 (enclosed in a green box).

4. COMPARISON AND PERFORMANCE ANALYSIS

In this section, we compare our proposed designs against baseline designs from the prior works, particularly those highlighted in [9] and [19], with same folding order for FFT and IFFT. Table 1 presents the timing and area performance metrics for FFT size N. Within Table 1, the proposed FFT and IFFT cascaded architecture is represented as "Proposed I", and it is benchmarked against the design from [9]. Concurrently, another proposed 2-interleaved FFT and IFFT cascaded design, labeled "Proposed II", is compared against the design in [19].

The proposed FFT and IFFT cascaded architecture saves about $N/2$ (16.67%) memory elements and $N/4$ clock cycles of latency (16.67%) than its baseline design in [9]. For the interleaved FFT-IFFT cascade, the memory and latency savings are, respectively, $N/2$ units (11%) and $N/2$ clock cycles (25%), compared to [19]. The percentage savings in brackets correspond to N=1024.

5. CONCLUSION

This paper presents a novel approach for designing cascaded FFT-IFFT architectures where the intermediate buffer is eliminated. Several cascaded architectures can be designed by using different folding sets for the FFT; thus, these architectures are non-unique. The proposed FFT-IFFT cascade approach has inspired the design of low-latency NTT-iNTT cascade architectures for homomorphic encryption [20].

6. ACKNOWLEDGMENT

The author is grateful to Dr. Nanda Unnikrishnan and Dr. Weihang Tan for their help in preparation of this paper. This research was supported in part by the National Science Foundation under grant number CCF-2243053.

7. REFERENCES

[1] Alan V. Oppenheim and Ronald W. Schafer, *Discrete-time signal processing*, Prentice Hall Press, USA, 3rd edition, 2009.

[2] Mojtaba Mahdavi, Ove Edfors, Viktor Öwall, and Liang Liu, "A low latency and area efficient FFT processor for massive MIMO systems," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2017, pp. 1–4.

[3] Yu-Wei Lin and Chen-Yi 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.

[4] Sai Sanjeet, Bibhu Datta Sahoo, and Keshab K Parhi, "Low-energy real FFT architectures and their applications to seizure prediction from EEG," *Analog Integrated Circuits and Signal Processing*, vol. 114, no. 3, pp. 287–298, 2023.

[5] Mengni Zhou, Cheng Tian, Rui Cao, Bin Wang, Yan Niu, Ting Hu, Hao Guo, and Jie Xiang, "Epileptic seizure detection based on EEG signals and CNN," *Frontiers in neuroinformatics*, vol. 12, pp. 95, 2018.

[6] Tahmid Abtahi, Colin Shea, Amey Kulkarni, and Tinoosh Mohsenin, "Accelerating convolutional neural network with FFT on embedded hardware," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 9, pp. 1737–1749, 2018.

[7] Caiwen Ding, Siyu Liao, Yanzhi Wang, Zhe Li, Ning Liu, Youwei Zhuo, Chao Wang, Xuehai Qian, Yu Bai, Geng Yuan, et al., "Circnn: accelerating and compressing deep neural networks using block-circulant weight matrices," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, 2017, pp. 395–408.

[8] Shousheng 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.

[9] Manohar Ayinala, Michael Brown, and Keshab 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, 2011.

[10] Mario Garrido, Keshab K Parhi, and Jesús Grajal, "A pipelined FFT architecture for real-valued signals," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 56, no. 12, pp. 2634–2643, 2009.

[11] Jian Wang, Chunlin Xiong, Kangli Zhang, and Jibo Wei, "A mixed-decimation MDF architecture for radix-2^k parallel FFT," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 24, no. 1, pp. 67–78, 2015.

[12] Kai-Jiun Yang, Shang-Ho Tsai, and Gene CH 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.

[13] Manohar Ayinala, Yingjie Lao, and Keshab K Parhi, "An in-place FFT architecture for real-valued signals," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 60, no. 10, pp. 652–656, 2013.

[14] Zhen-Guo Ma, Xiao-Bo Yin, and Feng 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.

[15] Keshab K Parhi, *VLSI digital signal processing systems: design and implementation*, John Wiley & Sons, 1999.

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

[17] Keshab 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.

[18] Nanda K Unnikrishnan and Keshab K Parhi, "InterGrad: Energy-efficient training of convolutional neural networks via interleaved gradient scheduling," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2023.

[19] Nanda K Unnikrishnan and Keshab K Parhi, "Multichannel FFT architectures designed via folding and interleaving," in *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2022, pp. 142–146.

[20] Weihang Tan, Sin-Wei Chiu, Antian Wang, Yingjie Lao, and Keshab K. Parhi, "PaReNTT: Low-latency parallel residue number system and NTT-based long polynomial modular multiplication for homomorphic encryption," *IEEE Transactions on Information Forensics and Security*, vol. 19, pp. 1646–1659, 2024.