# Novel Access Patterns Based on Overlapping Loading and Processing Times to Reduce Latency and Increase Throughput in Memory-based FFTs

Zeynep Kaya
*Department of Electricity and Energy*
*Bilecik Seyh Edebali University*
Bilecik, Türkiye
zeynep.kaya@bilecik.edu.tr

Mario Garrido
*Department of Electronic Engineering*
*Universidad Politécnica de Madrid*
Madrid, Spain
mario.garrido@upm.es

***Abstract*—This paper presents novel access patterns for P-parallel N-point radix-2 memory-based fast Fourier transform (FFT) architectures. This work aims to reduce the latency and increase the throughput by changing the data order and/or choosing different places of the architectures to input/output data. In this way, we can eliminate the loading time and/or the time to collect the output data. This results in a reduction in latency and an increase in throughput. Likewise, the architectures use the same permutation circuits for each iteration, which simplifies the circuit. In addition to improvements in latency and throughput with different access patterns for memory-based FFTs, this work also offers bit-reversed or natural input/output alternatives.**

***Index Terms*—Fast Fourier transform (FFT), radix-2, memory-based, low latency, high throughput.**

## I. Introduction

Fast Fourier transform (FFT) architectures play an important role in many signal processing applications. They are generally implemented on field-programmable gate arrays (FPGAs) or application-specific integrated circuits (ASICs) to achieve high-performance capabilities. This is the case of applications such as 5G and 6G communications [1]–[5], where the FFT is a key part of the system, and an efficient implementation of the FFT architectures is crucial to meeting the stringent requirements in terms of throughput, power consumption, and latency.

Motivated by the low latency requirements in 6G communications [6], in this paper, we explore how to reduce the latency in memory-based FFTs. Memory-based FFTs [7]–[13] are a type of FFT architecture that calculates the FFT algorithm iteratively. To achieve this, a memory-based FFT consists of a set of memories, permutation circuits [14], and processing elements (PEs) that include butterflies and rotators. Butterflies calculate the additions and subtractions of the algorithm, whereas rotators calculate rotations of the data in the complex plane. With these components, the memory-based FFT first loads the input data to the memories. Once the loading is finished, an iterative process to carry out the calculations of the FFT starts. This is done by loading data from memories, processing them in the PEs, and storing the results back in memories. This process repeats for each stage of the FFT until all the calculations have finished. Finally, after the processing time, data are outputted from the FFT architecture.

To achieve low latency in memory-based architectures, we have observed that it is possible to load data at different points of the circuit. For instance, data can be loaded directly to the butterflies instead of loading them first into memories. This reduces the loading time of the architecture. Likewise, for outputting data, after the FFT calculations have finished, it is possible to choose different points of the circuit. All of this leads to different access patterns for the memory-based FFT architecture, each of them with specific characteristics. These access patterns differ in the data order that is expected for the inputs and outputs. For instance, some configurations allow for natural (or normal) input order [15], [16], which means that data are received in order from the first value, $x[0]$, to the last value, $x[N-1]$, where $N$ is the FFT size. Conversely, in other configurations, the input order must be different. As a result, by changing the point of the circuit where data are loaded or output, we can modify the latency of the circuit and the input and output order of the data. The goal of this paper is to analyze the different architectures that result from modifying the access pattern and compare their characteristics.

To analyze the different access patterns, we consider the architecture proposed in [7] and modify it to achieve different configurations. This architecture implements a conflict-free access scheme based on the perfect shuffle permutation [17] that allows for simplifying the permutation circuits. This results in a reduction in the number of multiplexers in the architecture and a simplification of the memories thanks to the use of the same read and write addresses for all of them. Therefore, this advanced design is an excellent candidate to explore the new access patterns. Additionally, we present and analyze an analogous architecture based on the perfect

This work was supported in part by MCIN/AEI/10.13039/501100011033 and "ERDF A way of making Europe" under Project PID2021-126991NA-I00; in part by MCIN/AEI/10.13039/501100011033 and "ESF Investing in your future" under Grant RYC2018-025384-I.

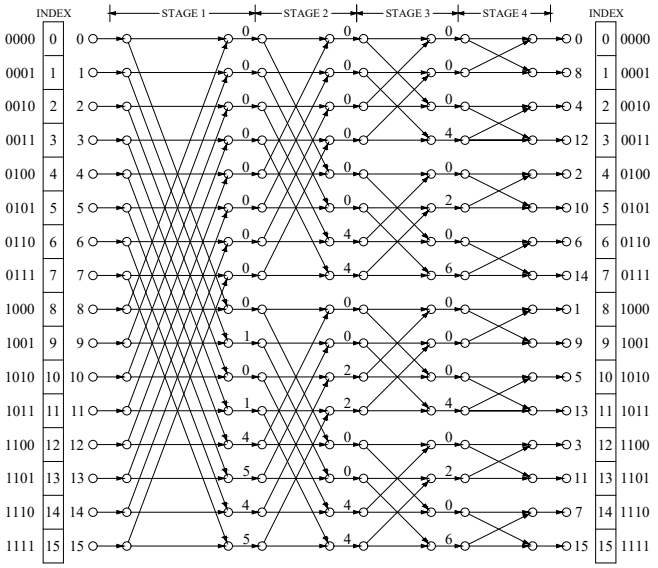


Fig. 1. Flow graph of a 16-point DIF FFT.

unshuffle permutation. All these alternative architectures are modeled in the paper in terms of latency and throughput.

The paper is organized as follows. In Section II, we review the main concepts needed to understand the analysis in this paper. In Section III, we present the proposed access patterns and analyze their characteristics. In Section IV, we compare the different access patterns and discuss the advantages and disadvantages of each of them. Finally, in Section V, we summarize the main conclusions of the paper.

## II. Background

### A. The FFT Algorithm

The discrete Fourier transform (DFT) is a well-known algorithm used to transform data in the time domain to the frequency domain. For an input sequence $x[n]$, the $N$-point DFT is calculated as

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{nk}, k = 0, 1, ..., N-1, \qquad (1)$$

where $k$ is the output frequency, $X[k]$ is the output at frequency $k$, and $W_N^{nk} = e^{-j\frac{2\pi}{N}nk}$ are rotations in the complex plane called twiddle factors.

The FFT is an algorithm that optimizes the calculations of the DFT by exploiting the fact that some operations of the DFT are repeated for different frequencies. This way, the FFT reduces the number of computations of the DFT from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log_2 N)$.

The FFT is commonly represented by its flow graph. The flow graph for a 16-point radix-2 FFT decomposed according to decimation in frequency (DIF) is shown in Fig. 1. The flow graph consists of $n = \log_2 N$ stages in the range $s = 1 \dots n$.

To explain the flow graph, an index $I$ is included at the left and the right of the figure. This index ranges from 0 to $N-1$ from top to bottom of the flow graph. The figure also includes the binary representation of the index, $b_{n-1} \dots b_0$, at the very left and right of the figure. Throughout the paper, we use ($\equiv$) to relate a number and its binary representation. Therefore, for the case of the index, we write $I \equiv b_{n-1} \dots b_0$.

The time index of the input data is shown at the input of the flow graph. It can be observed that the time index at the input is equal to the index $I$. Likewise, the frequencies of the outputs are shown at the output of the flow graph. In this case, the frequency is related to the index $I$ according to the bit reversal algorithm [18]. Thus, for any value of the index $I \equiv b_{n-1} \dots b_0$, the output frequency for that index is $k(I) \equiv b_0 \dots b_{n-1}$. As a consequence, if the index is in natural order at the input, the input data are received in natural order, whereas if the index is in natural order at the output of the FFT, the output data are provided in bit-reversed order. We will take into account this idea for the explanation of the architectures in this paper.

By analyzing the flow graph itself, we can observe that it consists of butterflies and rotations. The butterflies at stage $s$ add and subtract data whose index differ in $b_{n-s}$ [19]. For instance, the upper butterfly at stage 1 adds and subtracts $x[0]$ and $x[8]$, whose index differs in $b_{n-s} = b_{4-1} = b_3$. This applies to the butterflies at any stage $s$.

The rotators calculate rotations in the complex plane. The numbers between FFT stages, represented as $\phi$, indicate the values of these rotations. In an $N$-point FFT, any number $\phi$ represents a rotation by

$$e^{-j\frac{2\pi}{N}\phi}. \qquad (2)$$

### B. Permutations in FFT Architectures

In a hardware architecture that calculates the FFT algorithm, data are generally permuted between different stages. This comes from the fact that $b_{n-s}$ is different for each stage, which requires a reordering of the data. The way to carry out this reordering is using bit-dimension permutations [14].

To model an FFT architecture using bit-dimension permutations, we define the order of the dataflow with its position

$$\mathcal{P} = t|T, \qquad (3)$$

where $t$ is the arrival time of the data and corresponds to the serial nature of the data flow, and $T$ is the terminal where data arrives and corresponds to the parallel nature of the data flow. Note that a vertical bar (|) is used to separate the serial and parallel parts. As an example, let us consider a position $\mathcal{P} \equiv b_2 b_0 b_3 | b_1$. Thus, the sample with index $I = 6$, which has $b_3 b_2 b_1 b_0 = 0110$, will be in position $\mathcal{P} \equiv b_2 b_0 b_3 | b_1 = 100|1$, so it arrives in time $t = 4 \equiv 100$ at the terminal $T = 1$. Therefore, with this notation the place that any sample takes in the data flow is directly determined by its position.

The changes in the data order are carried out by using bit-dimension permutations. In this paper, we consider the perfect shuffle [17], which corresponds to the permutation

$$\sigma(u_{n-1} u_{n-2} \dots u_0) = u_{n-2} \dots u_0 u_{n-1}, \qquad (4)$$

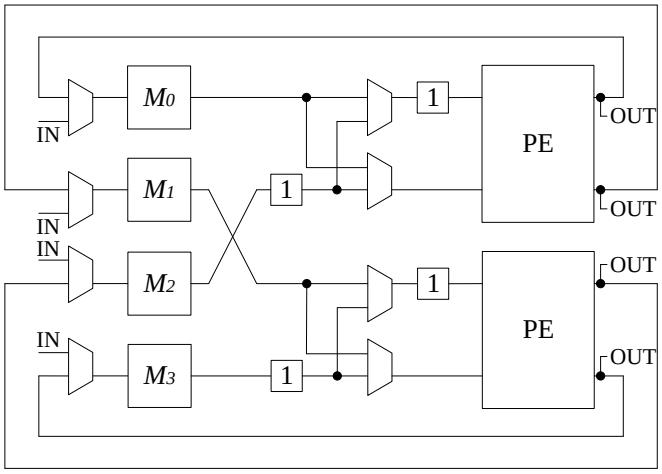


Fig. 2. 4-parallel memory-based FFT architecture based on the perfect shuffle. It corresponds to both the conventional approach and Case 1 under study. They have natural input order and bit-reversed output order.

and the perfect unshuffle according to

$$\sigma(u_{n-1}u_{n-2}\dots u_0) = u_0u_{n-1}\dots u_1. \tag{5}$$

Note that bit-dimension permutations are applied on $n$ bits, but they infer a permutation of $N = 2^n$ elements.

If we apply the perfect shuffle to a position $\mathcal{P} \equiv b_{n-1}\dots b_0$, we obtain the new data order $b_{n-2}\dots b_0b_{n-1}$. If we apply it again, we transform the order in $b_{n-3}\dots b_0b_{n-1}b_{n-2}$. And after the perfect shuffle is applied $n$ times, the position will end up in the same initial order $\mathcal{P} \equiv b_{n-1}\dots b_0$.

### C. Memory-Based FFT Based on the Perfect Shuffle

The idea of calculating the perfect shuffle of the data $n$ times has been applied to memory-based FFTs [7] and the resulting architecture is shown in Fig. 2. It consists of a group of memories $M_0$ to $M_3$, followed by shuffling circuits, where 1's in squares are buffers of length 1, and processing elements (PE) that consist of butterflies and rotators. The architecture processes $P = 4$ branches in parallel and is suitable for any FFT size, $N$. Note the difference between $P$, which shows the parallel branches of the architecture and $\mathcal{P}$, which is used for positions. A detailed explanation of this architecture is included in [7].

In this architecture, the perfect shuffle is calculated at each stage of the FFT, $s = 1\dots n$, by combining a permutation in the memory and the permutation of the shuffling circuits. The consequence of this is two-fold. First, at each stage, $s$, the permutation places $b_{n-s}$ in the lowest parallel dimension, where the butterflies operate, which guarantees that the FFT algorithm is calculated. Second, the permutation is the same for all the stages of the FFT. As the memory-based FFT calculates the algorithm iteratively and at each stage data pass through the same circuit, the fact that the same permutation is calculated at all the stages simplifies the architecture because a single permutation circuit is enough for all the stages.

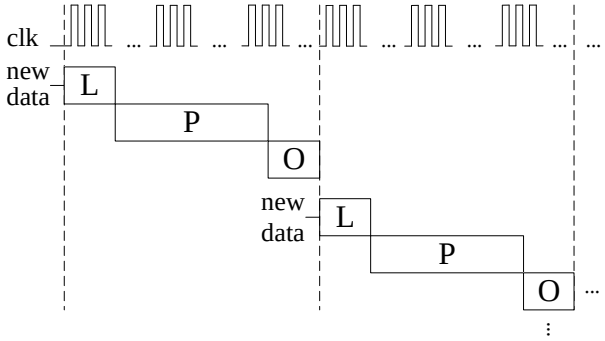


Fig. 3. Timing diagram of loading, processing, and output for conventional memory-based FFTs.

### D. Data Flow, Latency, and Throughput in Memory-Based FFTs

In FFTs, the latency and throughput are important metrics of performance. The throughput provides the average number of samples processed per clock cycle, whereas the latency is defined as the total clock cycles that are needed for the whole FFT calculation. It is important to remark that in this paper we consider latency as the time difference from the arrival of the first input data to the output of the last output data. Memory-based FFTs generally process bursts of data, where they are loaded, processed, and output. Thus, this latency represents the time since data start to be loaded in the architecture until the architecture is completely empty and new data can be loaded. An alternative criterion is to consider the time from the first input data to the last output data. This criterion is often used in FFT architectures that process data in a continuous flow.

To define the latency and throughput, it is needed to calculate the loading time of the input data, $T_{LOAD}$, the processing time, $T_{PROC}$, and the time to output results, $T_{OUT}$. Then, the latency is calculated as

$$T_{LAT} = T_{LOAD} + T_{PROC} + T_{OUT}, \tag{6}$$

and the throughput is

$$\text{Th} = \frac{N}{\Delta T_{FFT}}, \tag{7}$$

where $N$ refers to the number of data and $\Delta T_{FFT}$ refers the time difference between two consecutive FFTs.

Fig. 3 shows the timing diagram of conventional memory-based FFTs. In the figure, dashed lines separate the calculations of consecutive FFTs, L refers to the loading time of new data, P refers to the FFT processing time, and the time to get FFT results is represented as O.

## III. Proposed Access Patterns

In memory-based FFT architectures, the loading, processing, and output times depend on the number of parallel branches of the architecture, $P$, and the FFT size, $N$, being

$$T_{LOAD} = \frac{N}{P}, \tag{8}$$

$$T_{PROC} = \frac{N}{P} \cdot \log_r N, \tag{9}$$

$$T_{OUT} = \frac{N}{P}, \tag{10}$$

where $r$ is radix. Therefore, all FFT operations are completed in

$$T_{LAT} = \frac{N}{P} \cdot (2 + \log_r N) \tag{11}$$

clock cycles. However, by changing the place where data are input/output and/or the input/output data orders, the time of the whole FFT calculations can be reduced. In light of this information, we have proposed four cases to reduce the operation time, which leads to a reduction in latency and an increase in throughput. For simplicity, these cases are illustrated with figures with 4-parallel branches ($P = 4$). However, the explanations in the paper are general for any number of parallel branches, $P = 2^p$, and any FFT size, $N$.

### A. Data Order, Permutations, and Memory Addresses

To be able to understand the proposed access patterns, this section provides some key ideas about the data orders at the input and output, the permutations that are calculated in the memory-based FFTs, and the way the memory is accessed.

As explained in Section II-A, the data index is equal to the time index of the inputs and is related by the bit reversal with the frequencies of the outputs. As a consequence, if the data order at the input is

$$\mathscr{P} \equiv b_{n-1} \dots b_p | b_{p-1} \dots b_0, \tag{12}$$

then input data arrive in natural order to the input, whereas if this position appears at the output, then output data are in bit reversed order. Likewise, if the data order at the input is

$$\mathscr{P} \equiv b_0 \dots b_{p-1} | b_p \dots b_{n-1}, \tag{13}$$

then inputs arrive in bit-reversed order, whereas this position at the output indicates that output data are provided in natural order. Note that these data orders only depend on the $b_i$ bits placed as $b_{n-1}$ down to $b_0$ or as $b_0$ up to $b_{n-1}$, which is independent of the number of parallel dimensions, $p$. The fact that there exist $p$ parallel dimensions simply means that $P = 2^p$ values are provided per clock cycle.

Regarding the permutations in the memory-based FFT architectures, for the proposed architectures the perfect shuffle permutation in (4) is expressed as

$$\begin{aligned} &\sigma(u_{n-1} \dots u_p | u_{p-1} \dots u_0) = \\ &u_{n-2} \dots u_{p-1} | u_{p-2} \dots u_0 u_{n-1}. \end{aligned} \tag{14}$$

This equation includes the fact that the architecture has $P = 2^p$ parallel branches. Therefore, a vertical bar appears in the equation to separate the arrival time of the data and the terminal.

The perfect shuffle permutation is carried out in three steps as [7]

$$\sigma = \sigma_{sp} \circ \sigma_{pp} \circ \sigma_{ss}, \tag{15}$$

where $\sigma_{sp}$ is a serial-parallel permutation, $\sigma_{pp}$ is a parallel-parallel permutation, and $\sigma_{ss}$ is a serial-serial permutation [14]. The permutation $\sigma_{ss}$ is related to memory reading and writing addresses, which is calculated by a circular counter that is rotated right by one bit at each stage, being

$$\begin{aligned} W_{A_1} &= c_{n-p-1} c_{n-p-2} \dots c_0, \\ W_{A_2} &= R_{A_1} = c_0 c_{n-p-1} c_{n-p-2} \dots c_1, \\ W_{A_3} &= R_{A_2} = c_1 c_0 c_{n-p-1} \dots c_2, \\ &\vdots \\ W_{A_i} &= R_{A_{i-1}} = c_{i-2} \dots c_0 c_{n-p} \dots c_{i-1}, \end{aligned} \tag{16}$$

where $W_{A_i}$ and $R_{A_1}$ are the writing and reading addresses at the $i$-th iteration of the architecture, respectively, and $c_j$ is the $j$-th bit of the counter.

The permutations $\sigma_{pp}$ and $\sigma_{sp}$ correspond to the shuffling circuits between the memories and the PEs in Fig. 5 and are described in detail in [7].

In this work, we introduce the possibility of using the perfect unshuffle (5) permutation in memory-based FFTs. This permutation is expressed as

$$\begin{aligned} &\sigma(u_{n-1} \dots u_p | u_{p-1} \dots u_0) = \\ &u_0 u_{n-1} \dots u_{p+1} | u_p \dots u_1, \end{aligned} \tag{17}$$

and consists of three permutations as

$$\sigma = \sigma_{ss} \circ \sigma_{pp} \circ \sigma_{sp}. \tag{18}$$

Note that the order of these permutations is opposite to the order of the permutations in the perfect shuffle according to (15).

The circuits that calculate the permutations $\sigma_{pp}$ and $\sigma_{sp}$ are the same as in (18). However, $\sigma_{ss}$ permutation is achieved by substituting the MSB of the circular counter with LSB and shifting the other bits to the left and corresponds to

$$\begin{aligned} W_{A_1} &= c_{n-p-1} c_{n-p-2} \dots c_0, \\ W_{A_2} &= R_{A_1} = c_{n-p-2} \dots c_0 c_{n-p-1}, \\ W_{A_3} &= R_{A_2} = c_{n-p-3} \dots c_0 c_{n-p-1} c_{n-p-2}, \\ &\vdots \\ W_{A_i} &= R_{A_{i-1}} = c_{n-p-i} \dots c_0 c_{n-p-1} \dots c_{n-p-i+1}. \end{aligned} \tag{19}$$

As we will observe later in the paper, using the perfect unshuffle instead of the perfect shuffle will lead to different input and output data orders.

### B. Case 1: Natural Input - Bit-Reversed Output

Fig. 2 shows the architecture of Case 1, which corresponds to the architecture in [7]. In the figure, IN indicates where data are input to the architecture and OUT shows the place where the output results are obtained. Note that a set of multiplexers is used to either save input data in the memories or take data from the PEs to calculate the iterations of the FFT.

In this architecture, the input and output data orders are

$$\begin{aligned} \mathscr{P}_{IN} &\equiv b_{n-1} \dots b_2 | b_1 b_0, \\ \mathscr{P}_{OUT} &\equiv b_{n-1} \dots b_2 | b_1 b_0. \end{aligned} \tag{20}$$

According to (12), input data are received in natural order and output data are provided in bit-reversed order.

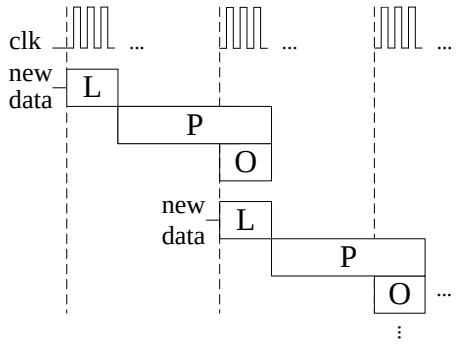


Fig. 4. Timing diagram in terms of loading, processing, and output times for the architecture in Fig. 2 (Case 1).

It should be noted that the bit-reversed data order in $\mathscr{P}_{OUT}$ is achieved by applying the perfect shuffle permutation (14) at each iteration. Thus, after the $n$ iterations of the memory-based FFT architecture, the position $\mathscr{P}_{OUT}$ takes the same value as $\mathscr{P}_{IN}$.

Fig. 4 shows the timing diagram of the architecture in Fig. 2 in terms of loading, processing, and output times. In this architecture, new data can be loaded at the same time that we obtain the results of the previous FFT. However, this possibility is not considered in [7], where new data are loaded only after all the outputs have left the architecture. Additionally, output data are provided during the last processing stage. Therefore, in Fig. 2, the last processing stage and the output time overlap.

To receive new data at the same time that the outputs of the previous FFT leave the architecture it is necessary to modify the architecture in [7] slightly. In [7] the memory addresses are obtained according to (16), and this pattern repeats for each new FFT, being $W_{A_1}$ always equal to $c_{n-p-1}c_{n-p-2}\dots c_0$. However, if new data are loaded at the same time that the last stage of the FFT is processed and data are output, then the new data must be stored in the memory addresses that are being emptied. This forces the writing address of the $i$-th FFT to be the same as the reading address of the $(i-1)$-th FFT, i.e.,

$$W_{A_1}(i) = R_{A_n}(i-1). \tag{21}$$

In practice, this simply means that the control counter continues shifting circularly from the value that it takes in the last iteration, instead of restarting it to $c_{n-p-1}c_{n-p-2}\dots c_0$ at the beginning of the next FFT.

According to the timing diagram in Fig. 4, the latency of the architecture from the arrival of the first inputs until all the outputs leave the architecture can be calculated as the sum of the loading and processing times. As data are output directly after the processing element at the same time that the last iteration of the architecture is calculated and are not stored in memory, the last FFT stage overlaps with the output time. As a result, the latency is calculated from the values of $T_{LOAD}$ and $T_{PROC}$ in (8) and (9) as

$$T_{LAT} = T_{LOAD} + T_{PROC} = \frac{N}{P}\cdot(1+\log_2 N). \tag{22}$$

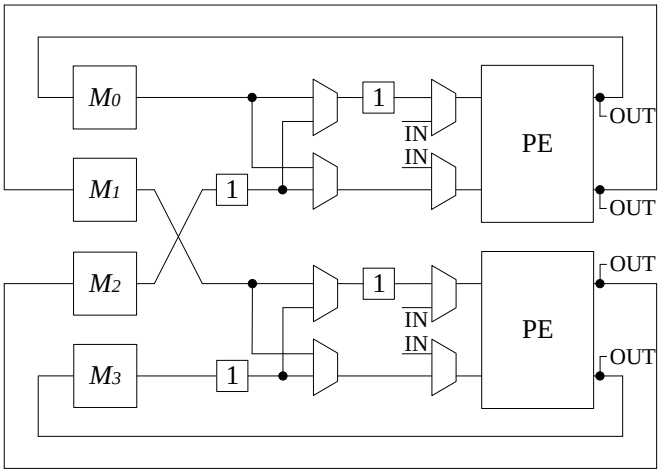


Fig. 5. 4-parallel memory-based FFT architecture based on the perfect shuffle. It corresponds to Case 2 under study, which has scramabled input order and bit-reversed output order.

The throughput of the architecture is also obtained from the timing diagram in Fig. 4. In this case, the time between two consecutive FFTs is

$$\Delta T_{FFT} = T_{LOAD} + T_{PROC} - T_{OUT} = \frac{N}{P}\cdot\log_2 N, \tag{23}$$

which, according to (7), results in a throughput of

$$\text{Th} = \frac{P}{\log_2 N}. \tag{24}$$

### C. Case 2: Scrambled Input - Bit-Reversed Output

Fig. 5 shows the architecture of Case 2, which is scrambled input and bit-reversed output. The input and output positions correspond to

$$\begin{aligned}\mathscr{P}_{IN} &\equiv b_{n-2}\dots b_1|b_0 b_{n-1},\\ \mathscr{P}_{OUT} &\equiv b_{n-1}\dots b_2|b_1 b_0.\end{aligned} \tag{25}$$

In this architecture, the permutations are the same as in Case 1. The only difference is the place of the circuit where data are input. Specifically, in this case data are input before the PEs. This avoids the initial permutation of Case 1, and data can start to be processed immediately as they are input. The consequence of this is that input data can not be received in natural order and must be received in a scrambled order, which is needed for the first stage. This scrambled input order is shown in (25).

Additionally, we obtain the output results just after the butterfly operations, so the output time overlaps with the processing of the last stage.

Fig. 6 shows the timing diagram of the architecture in Fig. 5. As can be seen in the timing diagram, L and P start at the same time, due to the fact that the FFT calculations can start as soon as data are received. Likewise, P and O end up at the same time. This means that the output results can be obtained during last stage operations.

It must also be noted that the condition in equation (21) must also be fulfilled in Case 2, to guarantee that input data

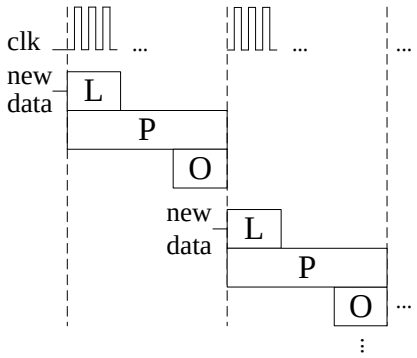


Fig. 6. Timing diagram in terms of loading, processing, and output times for the architecture in Fig. 5 (Case 2) and the architecture in Fig. 9 (Case 4).

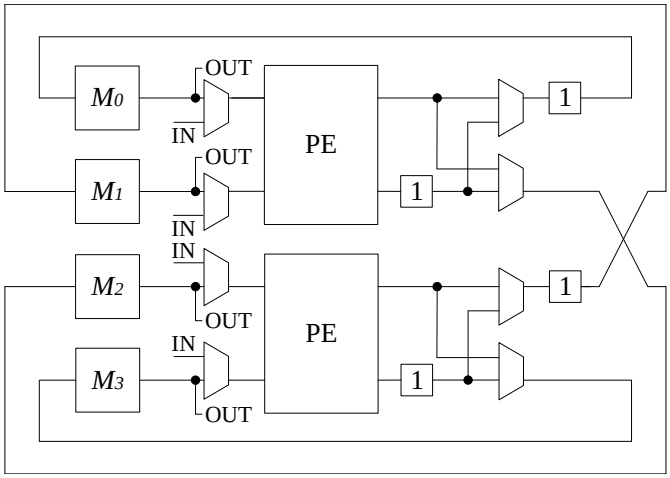


Fig. 7. 4-parallel memory-based FFT architecture based on the perfect unshuffle. It corresponds to Case 3 under study, which has bit-reversed input order and natural output order.

are written in the addresses that are emptied when the outputs of the previous FFT leave the architecture.

According to the timing diagram in Fig. 6, the latency of the architecture is equal to the processing time, i.e.,

$$T_{LAT} = \frac{N}{P} \cdot \log_2 N, \tag{26}$$

and $\Delta T_{FFT} = T_{PROC}$, which leads to a throughput

$$\text{Th} = \frac{P}{\log_2 N}. \tag{27}$$

### D. Case 3: Bit-Reversed Input - Natural Output

Fig. 7 shows the architecture of Case 3, which has been derived with the goal of obtaining natural output order. The input data of the architecture are received just before the PEs and the outputs are taken after the memories. Therefore, the initial data are received in bit-reversed order and perfect unshuffle is applied so that the output can be obtained as natural order. The input and output positions corresponds to

$$\begin{aligned} \mathcal{P}_{IN} &\equiv b_0 \dots b_{n-3} | b_{n-2} b_{n-1}, \\ \mathcal{P}_{OUT} &\equiv b_0 \dots b_{n-3} | b_{n-2} b_{n-1}, \end{aligned} \tag{28}$$

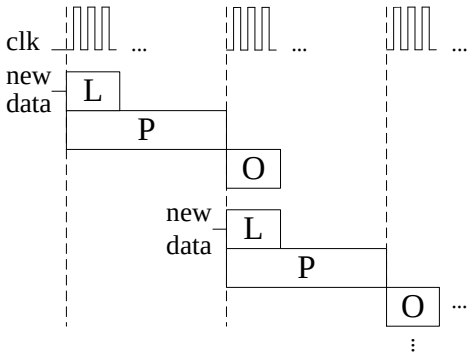


Fig. 8. Timing diagram in terms of loading, processing, and output times for the architecture in Fig. 7 (Case 3).

which are bit-reversed input order and natural output order according to (13). Note that $\mathcal{P}_{OUT}$ is obtained by applying $n$ times the perfect unshuffle permutation in (5) to $\mathcal{P}_{IN}$.

The reading and writing addresses of the memories correspond to (19).

Fig. 8 shows the timing diagram of loading, processing, and output data of the architecture in Fig. 7. The initial data comes before PEs so the butterfly operations start at the same time that data are loaded. This is why L and P are overlapped in the figure. In order to obtain the output data in a natural order, after the FFT operations of the last stage, data must be permuted once again. This is shown clearly in the fact that O occurs at the end of P in the timing diagram. However, the next FFT operations can start while the output data are taken from the memories. To make this possible, after new data are processed in the PEs, they must be stored in the memory addresses that are emptied from the previous FFT. Therefore, in Case 3, the condition in equation (21) must be fulfilled.

The latency of the architecture in Case 3 is the sum of $T_{PROC}$ and $T_{OUT}$, which results in

$$T_{LAT} = \frac{N}{P} \cdot (1 + \log_2 N), \tag{29}$$

and $\Delta T_{FFT} = T_{PROC}$, which results in a throughput

$$\text{Th} = \frac{P}{\log_2 N}. \tag{30}$$

### E. Case 4: Bit-reversed Input - Scrambled Output

Fig. 9 shows the architecture of Case 4. As shown in the figure, initial data comes before butterflies and output data are taken after the butterflies. This allows to initiate FFT operations immediately as data are received to the circuit, and also output data as it is processed in the last stage of the memory-based FFT architecture. In this circuit, the positions at the input and output are

$$\begin{aligned} \mathcal{P}_{IN} &\equiv b_0 \dots b_{n-3} | b_{n-2} b_{n-1}, \\ \mathcal{P}_{OUT} &\equiv b_1 \dots b_{n-2} | b_{n-1} b_0. \end{aligned} \tag{31}$$

According to (5), input data are received in bit-reversed order. By contrast, output data are provided in a scrambled order. Compared to Case 3, in the architecture of Case 4, the

TABLE I

COMPARISON OF ALL CASES.

| Access Pattern | Order | | Performance | | Memory | | Multiplexers | | Complex |
|---|---|---|---|---|---|---|---|---|---|
| | Input | Output | Latency | Throughput | Total Size | Banks | Input | Arch. | Mult. |
| Conventional | Natural | Bit-reversed | $(2+\log_2 N)\cdot N/P$ | $P/(2+\log_2 N)$ | $N+P$ | $P$ | $P$ | $P$ | $P/2$ |
| Case 1 [7] | Natural | Bit-reversed | $(1+\log_2 N)\cdot N/P$ | $P/(\log_2 N)$ | $N+P$ | $P$ | $P$ | $P$ | $P/2$ |
| Case 2 | Scrambled | Bit-reversed | $(\log_2 N)\cdot N/P$ | $P/(\log_2 N)$ | $N+P$ | $P$ | $P$ | $P$ | $P/2$ |
| Case 3 | Bit-reversed | Natural | $(1+\log_2 N)\cdot N/P$ | $P/(\log_2 N)$ | $N+P$ | $P$ | $P$ | $P$ | $P/2$ |
| Case 4 | Bit-reversed | Scrambled | $(\log_2 N)\cdot N/P$ | $P/(\log_2 N)$ | $N+P$ | $P$ | $P$ | $P$ | $P/2$ |

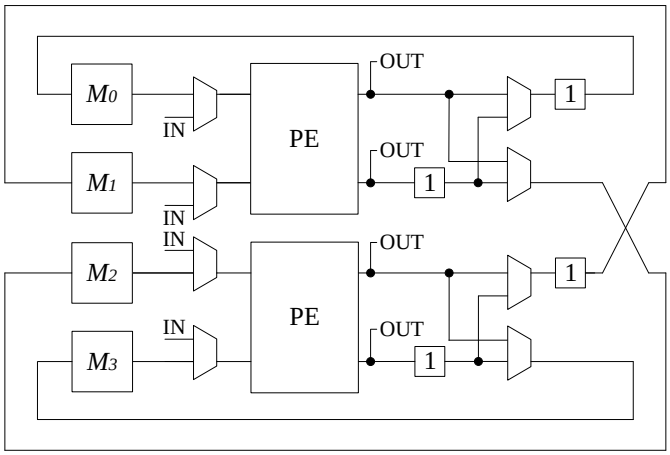


Fig. 9. 4-parallel memory-based FFT architecture based on the perfect unshuffle. It corresponds to Case 4 under study, which has bit-reversed input order and scrambled output order.

permutation in the memories after the PEs is not calculated, so the output data do not reach a natural output order.

The memory read and write address are the same as in Case 3, according to (19) and they must meet equation (21).

The timing diagram of the architecture in Fig. 9 is the same as that of the Case 2 architecture, and is shown in Fig. 6. Since processing begins as soon as data are received, L and P start concurrently. At the last stage operations data can be outputted, so the new data can be received and start to be processed. This causes the O and L times to overlap for two consecutive FFTs. Finally, it must be noted that although the timing diagram in terms of loading, processing, and output times is the same for Case 2 and Case 4, the data order in both architectures is different.

The latency of the architecture for Case 4 in Fig. 9

$$T_{LAT} = \frac{N}{P}\cdot \log_2 N, \tag{32}$$

and the time between two consecutive FFTs is $\Delta T_{FFT} = T_{PROC}$, which results in the same throughput as in the previous cases, i.e.,

$$\text{Th} = \frac{P}{\log_2 N}. \tag{33}$$

## IV. DISCUSSION AND COMPARISON

In this paper, we have presented two different types of architecture that perform the perfect shuffle and perfect unshuffle algorithms. Each of these two types is divided into two cases according to the placement of the input and output data. In Case 1 and Case 2, the perfect shuffle algorithm is applied and the permutation circuits and memory read and write addresses are the same. They only differ in the order and placement of the input data. This causes a variation in latency but the throughput remains the same. In Case 3 and Case 4, the perfect unshuffle algorithm is applied. The memory read/write addresses and the permutation circuits are the same in these two cases. Although both cases have the same input order, the order of the output data and the place where they are taken have differences. This causes a variation in latency but the throughput remains the same.

Table I compares all the proposed cases to the conventional access pattern according to Figs. 2 and 3 in terms of input/output order, latency, throughput, and area as a function of $N$ and $P$. It can be observed that all the proposed cases improve both latency and throughput with respect to the conventional case. This occurs due to two key contributions of our work. On the one hand, the loading, processing, and output times of the architectures are overlapped, which is only possible due to writing new data in the memory addresses that are being emptied in the previous FFT. This is achieved thanks to the condition in equation (21). By contrast, in previous memory-based FFTs, new data can not be loaded until all outputs of the previous FFT have left the architecture. On the other hand, we propose novel access patterns with different places in the architecture where data are input and output. This reduces the latency in Cases 2 and 4 even further.

In Table I we can also observe that conventional memory-based FFTs generally consider natural input and bit-reversed output orders. By contrast, in the proposed architectures a larger variety of input and output orders is achieved. In fact, the architectures based on the perfect unshuffle that we have proposed in this work open a new perspective on the possibilities of the data order in memory-based architectures. In this regard, note that the architecture in Case 3 provides output data in natural order, which was not possible by following a conventional approach.

By comparing the proposed access patterns, it can be

observed that there is a trade-off between input/output orders and latency. Between Case 1 and Case 2, Case 1 allows for natural input order, but has higher latency. The same occurs between Case 3 and Case 4. We can also observe that Cases 1 and 2 are analogous to Cases 3 and 4, and the selection among them depends on the desired input/output data order.

Regarding area, all approaches have the same number of resources in terms of total memory size, number of memory banks, multiplexers, and complex multipliers. Therefore, the proposed access patterns have the additional advantage that the improvement in terms of latency and throughput is achieved without increasing the hardware resources of the conventional architecture.

## V. Conclusions

In this paper, we have proposed novel access patterns for the radix-2 $P$-parallel memory-based FFT architecture. Apart from the access pattern based on the perfect shuffle permutation presented in previous works, in this work we have introduced the possibility of using the perfect unshuffle permutation in memory-based FFTs. Likewise, we have suggested new places in the architectures to input and output data. All of this leads to a variety of new access patterns that have been analyzed in the paper.

The new access patterns have been analyzed in terms of throughput and latency. Based on this analysis, it has been observed that they allow for reducing latency and increasing throughput with respect to conventional approaches. Furthermore, this improvement is achieved without increasing the number of hardware resources of the architecture, which makes these access patterns very useful to increase the performance in memory-based architectures. This is crucial for 5G and 6G systems, which present stringent requirements in terms of latency and throughput.

## References


[1] C.-X. Wang, X. You, X. Gao, X. Zhu, Z. Li, C. Zhang, H. Wang, Y. Huang, Y. Chen, H. Haas, J. S. Thompson, E. G. Larsson, M. D. Renzo, W. Tong, P. Zhu, X. Shen, H. V. Poor, and L. Hanzo, "On the road to 6G: Visions, requirements, key technologies and testbeds," *IEEE Commun. Surveys Tuts.*, vol. 25, no. 2, pp. 905–974, 2nd Quart. 2023.

[2] D. C. Nguyen, M. Ding, P. N. Pathirana, A. Seneviratne, J. Li, D. Niyato, O. Dobre, and H. V. Poor, "6G internet of things: A comprehensive survey," *IEEE Internet Things J.*, vol. 9, no. 1, pp. 359–383, Aug. 2022.

[3] C. De Alwis, A. Kalla, Q.-V. Pham, P. Kumar, K. Dev, W.-J. Hwang, and M. Liyanage, "Survey on 6G frontiers: Trends, applications, requirements, technologies and future research," *IEEE Open J. Commun. Soc.*, vol. 2, no. 1, pp. 836–886, Apr. 2021.

[4] S. Hakak, T. R. Gadekallu, P. K. R. Maddikunta, S. P. Ramu, M. Parimala, C. De Alwis, and M. Liyanage, "Autonomous vehicles in 5G and beyond: A survey," *Veh. Commun.*, vol. 39, Article 100551, pp. 1–31, Nov. 2022.

[5] S. Nayak and R. Patgiri, "6G communication technology: A vision on intelligent healthcare," in *Health Informatics: A Computational Perspective in Healthcare*, R. Patgiri, A. Biswas, and P. Roy, Eds. Springer Singapore, Jan. 2021, pp. 1–18.

[6] N. Rajatheva, I. Atzeni, E. Björnson, A. Bourdoux, S. Buzzi, J.-B. Doré, S. Erkucuk, M. Fuentes, K. Guan, Y. Hu, X. Huang, J. Hulkkonen, J. M. Jornet, M. Katz, R. Nilsson, E. Panayirci, K. Rabie, N. Rajapaksha, M. J. Salehi, H. Sarieddeen, S. Shahabuddin, T. Svensson, O. Tervo, A. Tölli, Q. Wu, and W. Xu, "White paper on broadband connectivity in 6G," 6G Research Vision, University of Oulu, Tech. Rep. 10, Jun. 2020.

[7] Z. Kaya, M. Garrido, and J. Takala, "Memory-based FFT architecture with optimized number of multiplexers and memory usage," *IEEE Trans. Circuits Syst. II*, vol. 70, no. 8, pp. 3084–3088, Aug. 2023.

[8] Z. Kaya and M. Garrido, "Low-latency 64-parallel 4096-point memory-based FFT for 6G," *IEEE Trans. Circuits Syst. I*, vol. 70, no. 10, pp. 4004–4014, Oct. 2023.

[9] Y. Guo, Z. Wang, Q. Hong, H. Luo, X. Qiu, and L. Liang, "A 60-mode high-throughput parallel-processing FFT processor for 5G/4G applications," *IEEE Trans. VLSI Syst.*, vol. 31, no. 2, pp. 219–232, Dec. 2023.

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

[11] Z. Kaya and E. Seke, "A novel addressing algorithm of radix-2 FFT using single-bank dual-port memory," *Circuit World*, vol. 48, no. 1, pp. 64–70, Jan. 2022.

[12] S. J. Huang and S. G. Chen, "A high-parallelism memory-based FFT processor with high SQNR and novel addressing scheme," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2016, pp. 2671–2674.

[13] 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*, vol. 59, no. 8, pp. 1752–1765, Aug. 2012.

[14] M. Garrido, J. Grajal, and O. Gustafsson, "Optimum circuits for bit-dimension permutations," *IEEE Trans. VLSI Syst.*, vol. 27, no. 5, pp. 1148–1160, May 2019.

[15] A. X. G. Xavier Chelliah, "A normal I/O order optimized dual-mode pipelined FFT architecture for processing real-valued signals and complex-valued signals," *AEÜ Int. J. Electron. Comm.*, vol. 170, no. 154782, pp. 1–9, Oct. 2023.

[16] Z. Wang, X. Liu, B. He, and F. Yu, "A combined SDC-SDF architecture for normal I/O pipelined radix-2 FFT," *IEEE Trans. VLSI Syst.*, vol. 23, no. 5, pp. 973–977, May 2015.

[17] H. Stone, "Parallel processing with the perfect shuffle," *IEEE Trans. Comput.*, vol. C-20, no. 2, pp. 153–161, Feb. 1971.

[18] M. Garrido, "Multiplexer and memory-efficient circuits for parallel bit reversal," *IEEE Trans. Circuits Syst. II*, vol. 66, no. 4, pp. 657–661, Apr. 2019.

[19] M. Garrido, J. Grajal, M. A. Sánchez, and O. Gustafsson, "Pipelined radix-$2^k$ feedforward FFT architectures," *IEEE Trans. VLSI Syst.*, vol. 21, no. 1, pp. 23–32, Jan. 2013.